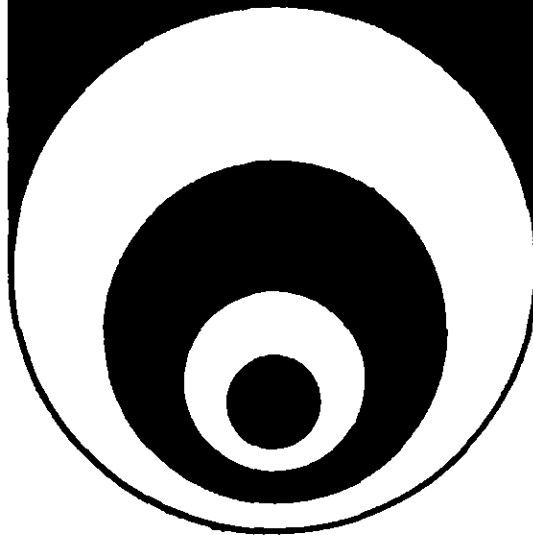


signetics

**MOS
MICROPROCESSORS**



2650 MICROPROCESSOR COURSE

**AN APPLICATIONS - ORIENTED
AUDIO - VISUAL LEARNING EXPERIENCE
IN
FUNDAMENTAL MICROPROCESSOR DESIGN**

**PREPARED BY:
MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Avenue
Sunnyvale, CA, 94086**

**© 1978
ALL RIGHTS RESERVED**

CREDITS:

The MOS MICROPROCESSOR TRAINING DEPARTMENT acknowledges with gratitude, the technical advice and support of the following named individuals, all staff in the Signetics* MOS MICROPROCESSOR DIVISION:

- Narpat Bhandari
- Alex Goldberger
- Dr. Donald F. Gorman
- Andrew Haines
- Paul Hansen
- Al Hazan; Division Manager
- John J. Keenan
- Mary Leon
- Barry Michael
- Gary Miller
- Bruce Schupp
- Edward E. Stelmach
- George Vashel
- Dr. Alan J. Weissberger
- Dave Wotring

ABOUT THE AUTHOR:

After taking his degree at St. Mary's University, Canada, in 1956, John C. Garceau has achieved an extensive background in Systems support and technical education activities. He has developed a number of courses in various media for Fairchild, Memorex, Trendata, and Information Storage Systems. Since joining Signetics* in April, 1976, Mr. Garceau has developed a number of presentations, including the highly popular 8X300 8ipolar Microprocessor Programming Course. The 2650 MICROPROCESSOR COURSE is his latest offering. An expert photographer, he's an enthusiast of ballooning, sailing, and flying. He's currently the Microprocessor Training Manager at Signetics*, is married, and resides in Menlo Park, California.

* Signetics is a wholly-owned subsidiary of U.S. Philips Corp.

SSSSSSSSSSSSSSSSSSSSSSSSSS

First, CONGRATULATIONS to you for selecting the

2650 MICROPROCESSOR COURSE

as the means by which you may become proficient in fundamental Microprocessor Design techniques. Here at Signetics, we sincerely believe that you will find the 2650 Microprocessor, and this course, entirely satisfactory to your needs.

A few quickly performed directions are provided to get you started in the course.

1. Install Tape 1; "CRAPGAME 2650" in your tapedeck, ready for playback.
2. At your option, make a 15 or 20 minute overview of the following:
 - a. The INSTRUCTOR 50 Reference Manual
 - Check for content and Features
 - Location of Operating Procedures
 - b. The 2650 MICROPROCESSOR Reference Manual
 - Contains all reference data and information concerned with the 2650.
 - Key sections are printed as the second half of the INSTRUCTOR 50 Ref. Manual. The separately bound volume is not included in this course documentation IN ORDER TO REDUCE THE COST OF THIS COURSE TO YOU.
 - c. The Topical Outline for Module I; "CRAPGAME 2650".
 - Outlines and Abstracts section; pages 1 to 3.
 - Module 1 provides an effective method by which you can become fully familiarized with INSTRUCTOR 50 operations in a practical applications setting. The Topical Outline sets the stage for your activities.
3. Turn to Page 1 in the Module I; "CRAPGAME 2650" text.
4. Listen to Tape 1. Stop your tapedeck when you hear the first 2-second 400 cycle tone.

IMPORTANT NOTE:

Several of the exercises and programs in this course revolve about applications which could be used for gaming purposes. Signetics provides these applications for your enjoyment only. Further, Signetics expressly prohibits the use of these applications for purposes which might violate the laws of your state as related to gaming or wagering!

All course materials are fully copyrighted by Signetics. Duplication of any course materials without the express written permission of Signetics, or its authorized representatives, is strictly prohibited!

Signetics

2650 MICROPROCESSOR COURSE

INDEX OF COURSE MATERIALS

<u>MODULE</u>	<u>TITLE</u>	<u>PAGES</u>	<u>TAPE NR.</u>
	Course Outlines and Abstracts	28	—
I	CRARGAME 2650 (INSTRUCTOR Familiarization)	66	1A
II-A	2650 Microprocessor	27	1B; 3A
II-B	2656 Systems Memory Interface(SMI)	5	3B
III-A	Introduction to Software and Glossary	36	4A
III-B	Number System Review	12	—
IV-A	Instruction Repertoire - Overview	1	4B
IV-B	2650 Reference Guide	6	4B
IV-C	Introduction to Instruction Set Functions	3	4B
IV-D	2650 Programming Form	9	4B; 5A
IV-E	Memory Concepts	13	5A
IV-F	Address Modes and Formats	108	5B; 6; 7
IV-G	Detailed Instruction Functions		
IV-H	Program Status Word	38	8A
IV-I	Monitor Subroutines available to User Program Access	9	8B
IV-J	Comprehensive Program Example	23	9A
IV-K	Extra Programs and Ideas	137	—
V	Interface and Interrupts	67	9B; 10; 11A
	Course Critique and Mailing Form	4	11A



2650 MICROPROCESSOR COURSE

COURSE OUTLINES AND ABSTRACTS

PREPARED BY:

**MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086**

OBJECTIVE: By completing a series of structured exercises, you will become fully familiarized with the INSTRUCTOR 50's operational modes including:

1. DISPLAY AND ALTERATION OF USER - AVAILABLE MEMORY
2. MEMORY FAST PATCH
3. REGISTER DISPLAY AND ALTERATION
4. PROGRAM COUNTER (START ADDRESS) DISPLAY AND ALTERATION
5. BREAKPOINT (ENDING) ADDRESS DISPLAY AND ALTERATION
6. STORE MEMORY ON CASSETTE
7. LOAD MEMORY FROM CASSETTE
8. ADJUST CASSETTE DATA PLAYBACK VOLUME CONTROL
9. PROGRAM EXECUTION {
 - RUN MODE
 - STEP MODE
 - RESET MODE
10. VISUAL INDICATORS {
 - 8 DIGIT
 - PARALLEL I/O LEDS
 - RUN AND FLAG LIGHTS
11. CONTROLS {
 - FUNCTION SELECTION KEYBOARD
 - HEXADECIMAL (16 DIGIT) KEYBOARD
 - INTERRUPT REAL - TIME/KBD SEL. SW
 - PARALLEL I/O SELECTION SWITCH

ABSTRACT: Of themselves, the INSTRUCTOR's many operational modes serve no really useful purpose; they can not be appreciated except in their selection and usage involved in support of a real application. In this application, CRAPGAME operates under Nevada casino "Passline" rules, while allowing maximum capability to buy into the game, place bets, and roll the dice by the user. Each of the INSTRUCTOR's operational modes is implemented in its logical support function to load CRAPGAME into memory, to exercise it completely, and to store it on cassette tape. Useful habits of observation and analysis are introduced as a necessary user function in programming microprocessor division application.

EXERCISE OUTLINE

1. PROGRAMMING "CRAPGAME" INTO THE INSTRUCTOR'S MEMORY

- USE EFFICIENT FAST PATCH MODE OF MEMORY LOAD.
- HEXADECIMAL NUMBERING INTRODUCED
- INITIALIZE INSTRUCTOR'S SELECTION SWITCH
- VERIFY DATA USING MEMORY DISPLAY AND ALTEK MODE
- RULES OF CRAPGAME ARE DESCRIBED

2. SEQUENCE PLAY

- SET PROGRAM EXECUTION START ADDRESS VIA PROGRAM COUNTER DISPLAY AND ALTER MODE
- EXECUTE CRAPGAME IN RUN MODE
- PLAY CRAPS -- BUY INTO GAME
- INTERPRET MESSAGES
- PLACE BETS

3. PROGRAM MODIFICATION - SINGLE BYTE

- SUBJECTIVE ANALYSIS OF APPLICATION - ORIENTED PROBLEMS INTRODUCED
- USE MEMORY DISPLAY AND ALTER MODE TO CORRECT PROBLEMS
 - * IN MESSAGE DISPLAY RATE
 - * IN DICE ROLL RATE
- EXECUTE CRAPGAME IN RESET MODE.

4. PROGRAM DEBUG BY MEMORY VERIFICATION

- USE MEMORY DISPLAY AND ALTER MODE TO:
 - * INSERT KNOWN PROBLEM
 - * CORRECT THE PROBLEM
- INTRODUCTION TO PROGRAM LISTING AND SUPPORT DOCUMENTATION
- OBSERVATION AND DECISIONS INVOLVED IN PROGRAM CHANGING ARE INTRODUCED
- OPTION: -- EXECUTE IN RUN OR RESET MODE.

5. PROGRAM EXECUTION IN STEP MODE

- DEMONSTRATION OF STEP MODE VERSATILITY IN PROGRAM DEBUG FUNCTION
- INTERACTIVE AND PRECISE OBSERVATIONS; CONCLUSIONS ARE DRAWN
- INTRODUCE PRACTICAL LIMITS OF STEP MODE - A "NEED"

6. PROGRAM EXECUTION IN BREAK POINT MODE.

- FROM PRACTICAL LIMITS OF STEP MODE, DEMONSTRATE PLACE OF BREAKPOINT MODE
 - SET PROGRAM EXECUTION ENDING ADDRESSES IN BREAKPOINT MODE
 - OBSERVATIONS AND CONCLUSIONS - PARALLEL I/O LED INDICATIONS
 - INTERACTION OF MEMORY DISPLAY AND ALTER
- | | |
|-----------------------------------|--------------------------------|
| PROGRAM COUNTER DISPLAY AND ALTER | } within
BREAKPOINT
MODE |
| REGISTER DISPLAY AND ALTER | |
| STEP MODE | |
| RUN MODE | |

7. WRITING A PROGRAM TO CASSETTE TAPE

- THEORY; WHY? WHEN?
- PRACTICAL DEMONSTRATION:
 - * CASSETTE TO INSTRUCTOR CONNECTION
 - * KEYBOARD COMMAND SEQUENCE
- SUPPORT DOCUMENTATION - RECORD KEEPING NOTATION

8. COMPLETE TAPE CASSETTE OPERATIONS

- WRITE ROUTINE "RLDICE" TO TAPE
- ADJUST PLAYBACK VOLUME CONTROL
- READ CASSETTE INTO MEMORY
- CONVERSION OF ROUTINE TO STANDALONE PROGRAM STATUS

9. USE OF I/O SWITCHES TO FINE - TUNE VALUES

- SUBSTITUTION BY PROGRAM MODIFICATION
- FINE TUNING - SWITCH VALUE
- RESTORATION AND EXECUTION OF ORIGINAL PROGRAM

SOURCE MATERIAL

- "CRAPGAME" PROGRAM LISTING
- "CRAPGAME" MEMORY MAP
- INSTRUCTOR 50 USAGE MODE DIAGRAMS

OBJECTIVE: By completing a series of structured exercises, you will become familiarized with basic computer concepts, and the functional blocks of the microprocessor implemented to satisfy each concept.

ABSTRACT: A foundation consisting of basic computer concepts, structure, and functions is first established. Then, based on this foundation, each of the microprocessor's functions is examined as if you were being introduced to a well stocked tool box chock full of finely crafted instruments. Emphasis is placed on selection of microprocessor features and functions best suited for the desired application.

OUTLINE:

INTRODUCTION:

- Definition of Computer Algorithm Program
- Overall computer organization

EXERCISES: Microprocessor Block Analysis

1. Preliminary Activities

- Reload of crapgame into the Instructor
- Software and hardware = total computing system

2. Data Bus and Data Bus Register, Timing Logic

- Timing Logic Demonstration
- Data Bus and DBUS Register Demonstration
 - * Connection between microprocessor and external hardware
 - * Read and Write

3. Instruction Register, Instruction Address Register, Holding Register, DBUS Register (Part 2)

- Relationship to Memory
- Instruction Access - 1, 2, or 3 bytes
- Execution - Subsequent Movement of Data
- Demonstrated Data Transfer
- I/O LED Pattern Interpretation

4. I/O Logic Block

- I/O Control Signal Definition Introduction
- Demonstration of Non-Extended I/O Transfer
- Demonstration of Memory Access and Data Transfer

5. Register Stack: Register 0

- General Purpose Register Organization, Selection and Multiplexing
- Demonstration of Prime, Non-Prime Register Selection and Usage
- Register Selection Controls

6. Arithmetic Logic Unit (ALU)

- Introduction to Each Function Performed by the ALU
- Data Manipulation/Movement Through ALU
- Status Generation by ALU

7. Condition Code and Branch Logic

- Need for Program Control - Response to Varying Conditions
- Demonstration of Program Control Decision Making
- Importance of Predictability, Observation and Examination of Microprocessor Indications

8. Program Status Word (PSU and PSL)

- Introduction to Concept of Program Status and Control
- Definition of Each Program Status Bit
- Demonstration of Instruction Execution Effect on Program Status Word
- Interpretation of Program Status

9. Return Address Stack

- Relationship to Program Flexibility and Control
- Concept and Logical Organization of the Stack

10. Instruction Address Register, Operand Address Register, Address Adder, Output Control

- Relationship to Address Mode Versatility of the Microprocessor
- Examination of Block Diagram Flow for Each Mode

OBJECTIVE: Via study of appropriate block diagrams, you will become familiar with the usage and relationship of the 2656 Systems Memory Interface IC in a microprocessor-based computing system.

ABSTRACT: The applications possibilities for use of the SMI are first introduced. Then, after its functions are defined, the SMI is illustrated in a series of diagrams showing its usage within a generalized microcomputing system. The presentation is completed by illustrating the usage of the SMI within a complex application; the INSTRUCTOR 50.

TOPICAL OUTLINE:

- Purpose and Functions
- Contents and Organization
- Microprocessor vs. Microcomputer Illustrated Concepts
- Block Diagram Layout; 2650/2656 SMI Microcomputer
- Block Diagram - INSTRUCTOR 50

OBJECTIVE: In this lesson, you will become familiar with the relationship between software and hardware design concepts, and achieve a reasonable understanding of microcomputer technical terms and definitions.

ABSTRACT: Software/hardware relationships are introduced by developing a detailed analogy with sheet music as it relates to a composer, conductor and orchestra. Most used terms and definitions involved in microprocessor design are then set forth in a glossary. The glossary is followed by a self-administered quiz by which you can determine the level of your understanding of microprocessor-oriented definitions.

OUTLINE:

- A/V presentation of Hardware/Software Function Comparisons
- Familiarization with microprocessor-oriented terms and definitions
- Self-administered quiz - microprocessor terms

NOTE: While optional, the material in this lesson should be studied by those students with little or no prior experience in computer-oriented applications. Instruction offered in course modules which follow presume your understanding of all terms listed, and of the systems relationship between software and hardware.

OBJECTIVE: At the conclusion of this lesson, you will be able to demonstrate your understanding of:

- the relationship of different number systems to each other in a computer environment
- functions performed by you to convert specific numbers from one number system notation to another, e.g.,
 - * binary to hexadecimal
 - * hexadecimal to decimal
 - * decimal to hexadecimal

ABSTRACT: This optional lesson provides its reader with the reasons for different number system usage in a computer environment. Following a comparison between binary and decimal numbering, the hexadecimal (16 digit) number system is introduced as a useful number system both for computers and human factors. Symbols commonly used to differentiate digit sequences written in binary, decimal, and hex are introduced from a practical standpoint. Following this, simple methods to convert numbers from one system to another are introduced, first by equation, then in practice, through use of any 4-function calculator. A self-administered quiz measures the student's familiarity with the different number systems, including practical conversion from one system to another.

- OUTLINE:**
- Number processing - binary by computer
- in decimal by people
 - Decimal to Binary Conversion
 - Reason for Hexadecimal (16 digit) number system
 - Binary to Hexadecimal Conversion
 - Number system identifying symbols
 - Decimal to Hexadecimal; hexadecimal to decimal conversions
 - Hexadecimal/decimal conversion with 4-function calculator

2650 MICROPROCESSOR COURSE
TOPICAL OUTLINE

MODULE IV-A
INSTRUCTION REPERTOIRE

ABSTRACT: Really not a lesson in the true sense, this section provides a means to organize all course materials required for detailed analysis of the Instruction Repertoire. A taped summary of activities completed thus far in the course, together with an introduction to the various sections involved in Module IV, is presented.

OBJECTIVE: At the conclusion of this lesson, you will be able to identify and define the purpose and usage of all 2650 PROGRAMMING FORM spaces.

ABSTRACT: Usage of the 2650 PROGRAMMING FORM is identified with the need to provide strong support documentation for all 2650 - executable routines and programs. The blocks which comprise the PROGRAMMING FORM are defined by function with their relationship to typical assembler listings highlighted. A practical exercise permits the student to design assemble and execute a routine in a series of clearly defined logical steps.

OUTLINE:

- Definition of Fields -- 2650 PROGRAMMING FORM
- Logical Extension for use in Programming the Assembler
- Overall sequence of Formal Program preparation illustrated
- Detailed Sequence to prepare CLEAR MEMORY routine.
 - * Specification card Flowchart of the routines
 - * Assignment of registers by proposed function
 - * Preparation of the Programming Form's Header.
 - * Symbolic instruction sequence (source statement prep.)
 - * Comment Listing
 - * Hex Code preparation
 - * Instruction address designation
 - * Load and execution of the routine in the INSTRUCTOR.

OBJECTIVE: Upon the completion of this lesson, you will be able to identify and define the different types of memory by function. You will also be able to demonstrate your understanding of memory concepts as they are applied to memory organization both within the INSTRUCTOR and as expanded outside the INSTRUCTOR.

ABSTRACT: In this lesson, an introduction to memory addressing and microprocessor memory access control leads to a detailed study of memory block assignments in a typical 4K program. This is followed by diagrams which illustrate existing INSTRUCTOR memory and possible expansion capabilities. Memory mapping is introduced, using program "CRAPGAME" as a model. Your study is reinforced by completion of a practical exercise involving inspection of the MONITOR USE (User System Executive) software and scratchpad memory.

OUTLINE:

- Concept of Memory Addressing
- Memory Address Control Signals and "Page" Organization
- Comparison Between ROM and RAM
- 2650 Memory Organization - Typical 4K Program
- INSTRUCTOR on-board memory organization and expansion
- Memory Map - "CRAPGAME"
- MONITOR USE Program Inspection
 - * Fixed and variable data fields in user program
 - * Monitor Scratchpad - field definition and usage
 - * Monitor Scratchpad - Breakpoint controls inspection and alteration of register contents saved by monitor activities
 - * Monitor ROM program inspection and alteration of variable data

OBJECTIVE

By completing a series of structured exercises, you will be able to demonstrate your knowledge of, and ability to program the 2650 Microprocessor instructions in all possible address mode variations.

ABSTRACT

A precise and highly detailed analysis of all instruction functions and address mode variations is provided in this section. Importantly, the presentation is based on practical situations you'll likely encounter when you program your application. As much as possible, variations and subtleties involved in the selection of the Instruction Repertoire are highlighted. Basic specification and flowcharting techniques are woven into this section with a primary dedication to practical programming techniques. Emphasis is placed on mastering the array of support documentation designed to make your programming experience simplified.....and.....EFFECTIVE.

OUTLINE

- Review and recommendations for Programming.
- SYMBOLIC EQUATE TABLES
 - Concept of Declaration
 - Compatibility with use of the Assembler and other Software Development Aids.
 - Definition of common symbols used in writing software.
- EXERCISE 1 REGISTER-ADDRESSED INSTRUCTIONS - FORMAT Z
 - Instruction Length and Timing.
 - Advantages and Limitations.
 - Detailed description and definition of OPCODE byte fields.
 - 2650 internal operation during decode and execution of Format Z instructions.
 - * Data Flow
 - * Block Analysis
 - Entry and execution of Format Z instructions - a working program
 - * Code and verification
 - * Practical observation
 - Review of Boolean functions (option)
 - * Definition of Boolean operations performed.
 - * Fully documented examples.
 - * Practical organization techniques

● EXERCISE 2 MEMORY DATA CONTROL OPERATIONS

- Basic routines for:
 - * Clearing a precise block of memory.
 - * Loading a precise block of memory with an incremented data pattern.
- Usage of Format 7 instructions in a practical application.
- Precise observation technique practices.

● EXERCISE 3 IMMEDIATE ADDRESSED INSTRUCTIONS - FORMAT I

- Definition and Specification
- Applications; Advantages and Limitations.
- Immediate Instruction Field definition; Relation of Symbolic Code to Object code.
- Practical Usage Techniques:
 - * in data manipulation program
 - * in Extended I/O address operations
 - * during code, execution, and debug functions.

● EXERCISE 4 EXPANDED MEMORY DATA CONTROL OPERATIONS

- Use of Immediate-addressed instructions to set fixed data and constants into a defined memory block.
- Implementation of Looping; Flowchart equivalence.

● EXERCISE 5 RELATIVE ADDRESSED INSTRUCTIONS - FORMAT R

- Introduction
 - * Variety of OPCODES permitting Relative addressing.
 - * Variations of Relative addressing.
 - * Theory of Relative Displacement.
- Application.
- Advantages and Limitations.
- Comparison of working program function programmed in Relative as contrasted with absolute address mode.
 - * Memory conservation
- Diagrammed Concept of Relative Address.
 - * Calculation of relative address range limits.
- Use of the Programming Form Relative Address Table.
- Practical code and execution of a working program.

● EXERCISE 6 PROCEDURE TO EXECUTE ROUTINE "FORMR"; RAHCO

- Demonstration of methods to effect rapid program changes.
- Subtle effect of improper coding on existing memory data.
- Analysis of program sequence via INSTRUCTOR facilities.
- Relative Address Hand-Code Computer (RAHCO)
 - * necessity to automate calculation of relative displacement between 2 absolute addresses.
 - * Assembly of RAHCO
 - * Directions for use of RAHCO
- Useful program preparation sequence hints

- Comparison of Literal/Symbolic notation for Displacement byte programming.
 - * literal (numeric) code limitations.
 - * Use of the "\$" symbol.
 - * Full symbolic coded examples -- memory mapped.
- Parameter diagrams and field definitions for
 - * Data Handling (non-Branch) instructions
 - * Program Control (Branch) instructions

EXERCISE 7 DEMONSTRATION OF RELATIVE ADDRESSED BRANCH INSTRUCTIONS

- Introduction
 - * Flexibility
 - * Application
- Practical Demonstration.
 - * Code and execute an UP/DOWN COUNTER application.
 - * Control of amplitude values.
 - * Time delay considerations.
- Expanded Application.
 - * routine use to control a Digital to Analog circuit
 - * Staircase, square wave, clock, and "flyback" variations.

EXERCISE 8 MODIFICATION OF "CUPDN1" TO SUBROUTINE CALL OPERATION

- Introduction
 - * Subroutine organization concepts-
 - * "Linking" instruction usage.
 - * Memory conservation
- Code and execute routine "CUPDN3"
- Modifications and methods to estimate execution time of repeatable processes with accuracy.

PAGE 0 REFERENCED RELATIVE ADDRESSED BRANCH INSTRUCTIONS

- Introduction.
 - * Advantages and Limitations
 - * Application.
- Concept of relative displacement from Page 0;Byte 0 in memory.
- Detailed Parameter and field definition; ZBRR/ZBSR instructions

EXERCISE 9 PRACTICAL APPLICATION OF ZERO-REFERENCED BRANCH INSTRUCTIONS

- Demonstration using "CRAPGAME" as model
- Analysis of indirect and effective address fields
- Subtle variations possible in using ZBRR/ZBSR instructions.

- Review: assembled program definition of ZBRR/ZBSR usage in MONITOR software.

- DATA HANDLING INSTRUCTION PARAMETERS - INDIRECT RELATIVE ADDRESS

- Full diagram and Memory map analysis
- Examples of practical symbolic coding methods
- Subtleties involved programming indirect address functions.

- EXERCISE 10 PRACTICAL USE OF INDIRECT RELATIVE ADDRESSED DATA TRANSFER INSTRUCTIONS

- Introduction.
 - * memory block allocation for indirect address tables
 - * Use of the 2 low-order bytes of an absolute addressed instructions as a relative indirect address.
 - * Program preparation - allowance for indirect addressing
- Practical demonstration of single and multiple entry Indirect address fields.

- DIAGRAMMED ANALYSIS - INDIRECT RELATIVE ADDRESSED BRANCH INSTRUCTIONS

- Detailed parameter and field definition
- Relation of symbolic code to memory address designation.

- EXERCISE 11 PRACTICAL USE OF RELATIVE INDIRECT ADDRESSED BRANCH INSTRUCTIONS

- Conversion of absolute addressed instruction to relative indirect.
- Identification of practical and impractical usage factors through analysis of "CRAPGAME".

- ABSOLUTE ADDRESS MODES - FORMAT A

- Introduction
 - * Different variations.
 - * concept of Indexing.
 - * Applications to table-driven programs
- Parameter definitions and diagram analysis of:
 - * Direct absolute addressed non-Branch instructions.
 - * Indexed direct absolute addressed non-Branch Instructions.
 - * Indirect absolute addressed non-Branch instructions.

- EXERCISE 12 PRACTICAL USE OF ABSOLUTE ADDRESSED NON-BRANCH INSTRUCTIONS - I
 - Introduction
 - * Review
 - * Tradeoffs between memory conservation and execution speed; direct vs. indirect variations.
 - Demonstration of Usage; alternatives.
 - Calculation of execution time differences.
- EXERCISE 13 PRACTICAL USE OF ABSOLUTE ADDRESSED NON-BRANCH INSTRUCTIONS - II
 - Memory block controls
 - Practical variations to:
 - * clear memory blocks
 - * set incrementing and decrementing patterns in memory.
- EXERCISE 14 LINKING OF ROUTINES INTO A SINGLE PROGRAM
 - Introduction
 - * Methods of linking
 - Alternative approaches and use.
- ABSOLUTE ADDRESSED BRANCH INSTRUCTIONS - FORMAT B NON-INDEXED
 - Introduction.
 - * Variations of absolute addressed Branch instructions
 - Analysis of Direct and Indirect absolute addressed Branch instructions.
 - * diagrammed parameters and fields.
 - * symbolic coding considerations.
 - * examples and memory mapping.
 - * advantages and limitations.
- EXERCISE 15 PRACTICAL USE OF ABSOLUTE ADDRESSED BRANCH INSTRUCTIONS - I
 - Practical demonstration of routine linkage using a combination of BSTR and BSTA instructions.
- EXERCISE 16 PRACTICAL USE OF ABSOLUTE ADDRESSED BRANCH INSTRUCTIONS - II
 - Introduction
 - * Application in I/O polling and selection routines.
 - * Status interpretation
 - * Command pattern generation.
 - Memory mapped I/O concepts
 - * Theory of "Handshaking".

- Practical demonstration of Device Polling and Selection.
 - * "Wait" routines
 - * Detailed flowcharting and specification.
- Use of the TMI and appropriate branches for BIT TESTING and decision.
- Code and execution of a useful "POLL" scheme.
- Practical alternatives and variations.

INDEXED BRANCH INSTRUCTIONS - DIRECT AND INDIRECT ABSOLUTE ADDRESS

- Parameter specification; field definition of BXA BSXA
- Memory mapping and organization.
- Relation to microprocessor internal block diagram.
- examples of symbolic coding; interpretation.

EXERCISE 17 PRACTICAL USE OF INDEXED ABSOLUTE ADDRESSED BRANCH VARIATIONS

- Introduction.
 - * Limitation of previous "POLL" program.
 - * Importance of "housekeepin" processes
 - memory organization
 - assignments
 - * Theory of VECTORED program control
- Description of program "SRVCIO".
- Load and execution of "SRVCIO"
- Variations of vectored polling schemes.
- Practical interpretation of possible changes and variations.

EXERCISE 18 PROGRAM MODIFICATION TO EXPAND THE NUMBER OF I/O DEVICES SERVICED BY THE MICROPROCESSOR

- Your recommendations are compared with stated possibilities.
- Optimizing Considerations
 - * memory block designation
 - * message formatting with the INSTRUCTOR
 - * address considerations

EXERCISE 19 ROUTINE RELOCATION

- Introduction.
 - * Necessity
 - * concepts and usage
 - * theory of auxiliary storage
 - * approaches

● EXERCISE 20 DIRECT ROUTINE RELOCATION - METHOD 1

- Practical demonstration of 1st approach
 - * advantages and limitations
 - * memory conservation vs. automation
 - * verification of results

● EXERCISE 21 DIRECT ROUTINE RELOCATION - METHOD 2

- subtle variations, given different input parameters
- Code and execute program.
- precise verification techniques.

● EXERCISE 22 INDIRECT ROUTINE RELOCATION

- Demonstrates effective 'automation' of relocation techniques.
- advantages and limitations
- Usage in practical program preparation

● MISCELLANEOUS INSTRUCTIONS - NOP; HALT

- Introduction
 - * useful functions
 - * variation of sequence continuation after HALT.
 - * Applications where HALT is effectively used.

● EXERCISE 23 PRACTICAL USE OF THE HALT INSTRUCTION

- Demonstration of HALT usage advantages and limits
- Conditional HALT usage
- variations.

OBJECTIVE: Upon completion of this lesson, you will be able to demonstrate your understanding of the Program Status Word's versatility and usage by the microprocessor.

ABSTRACT: This lesson emphasizes the versatility and importance of the Program Status Word as it relates to microprocessor operation and programming techniques. Emphasis is placed on exploiting each defined function in the PSW to its fullest flexibility in order to enhance your preparation of future applications. A series of carefully structured exercises reinforces each described topic.

OUTLINE:

PART 1 - OVERVIEW

- Definition of Instruction Repertoire related to the PSW
- Relationship of each PSW bit to the Microprocessor Block Diagram
- Review of PSW by bit definition

PART 2 - CONDITION CODE (PSL Bits 7,6)

- Functions which can alter the condition code
- Modification of condition code - by load/store instructions
 - Arithmetic/Logical instructions
 - Compare instructions/ Use of Compare Control bit
 - Test instructions - TPSU, TPSL, TMI
- Applications and extensions

PART 3 - INTERDIGIT CARRY, OVERFLOW, CARRY (PSL Bits 5, 2, 0)

- IDC, OVF, and C status - During "ADD" instruction execution
 - During "SUBTRACT" instruction execution
 - During "ROTATE" instruction execution
- Application and Interpretation; practical usage.

PART 4 - STACK POINTER (PSU Bits 2-0)

- Review of Definition and Control instructions
- Theory of subroutine nesting; limitations
- Program sequence error recognition and prevention
- Stack pointer "wraparound" characteristics; control
- Stack pointer modifications; constraints and limitations

OBJECTIVE: At the conclusion of this lesson, you will be fully familiarized with the definition and operation of monitor subroutines available to USER program access.

ABSTRACT: Primarily a reference section, the material in this lesson complements monitor subroutine descriptions within the INSTRUCTOR 50 Reference Manual. Where necessary, useful programming considerations and constraints are drawn to make your use of these powerful routines as effective as possible.

OUTLINE:

- Overview - Short Form listing of routines with pertinent addresses.
- USRDSP - Display user defined message on Hex Display Typical Program Parameterizing.
- DISLSD - Convert a byte in R0 to NIBLS in R0 and R1.
- GNP/
 GNPA } - Keyboard Data Entry to User Program.
- ROT - Exchange (rotate) NIBLS in R0.

OBJECTIVE: Upon completion of this lesson, you will be able to demonstrate your understanding of the activities performed by the programmer in sequencing an application from idea to final operational system.

ABSTRACT: Using the idea for an electronic "TRAIN" as a model, you will perform the following steps:

- Gather facts - based on written ideas and discussions with the inventor.
- Write the preliminary objective specification.
- Flowchart the overall and detailed sequences of operation.
- Write and debug the program.
- Systems checkout.
- Prepare object program tape for ROM

The application "TRAIN" permits selection of any one of 16 'trains' for display; to run at any one of 7 speeds, forward or reverse, or to stop.

After completion of each step, a model solution is proposed to reinforce your activities. There is no supplemental outline in this section since the steps you perform are themselves an outline of the contents of this lesson.

2650 MICROPROCESSOR COURSE TOPICAL OUTLINE

MODULE IV-K EXTRA PROGRAMS AND IDEAS

ABSTRACT:

This section contains additional prepared applications programs and ideas for other entertaining programs possible within the limitations imposed by the INSTRUCTOR's present configuration. While no instruction is offered in this section, you are encouraged to follow the Programmer's activity steps (Module IV-J) in preparing and executing one or more of these ideas.

This section also contains a series of formal APPLICATION NOTES and MEMOS, prepared in the last 2 years by Applications Engineering at Signetics. These include the following:

- MP-51 265D Initialization
- AS-51 Bit and Byte Testing Procedures
- AS-52 General Delay Routines
- AS-56 Sorting Routines
- AS-53 Binary Arithmetic Routines
- AS-55 Fixed-point Decimal Arithmetic Routines
- AS-54 Conversion Routines
- MP-53 Address and Data Bus Interfacing Techniques
- AS-50 Serial Input/Output
- - Interface Design with the Signetics 2650
- - DAC and ADC to Microprocessor Interface Techniques
- o - Analog to Digital Conversion - Successive Approximation Method

Upon completion of this lesson, you will be able to design and implement various interface and interrupt schemes with the 2650 microprocessor. You will be able to demonstrate a thorough understanding of timing and line control necessary for an effective choice of interface schemes.

The study of systems interfacing is introduced by defining the several types of I/O facilities available to the user. This leads to a detailed development of systems timing and control signal definition and usage. Particular stress is placed on the sequenced relationship of micro-processor signals which together define and control selection of different I/O configurations. Each Interface scheme (e.g., extended, serial, non-extended) is diagrammed, then given practical emphasis through specific exercises. Systems timing is emphasized by diagrams which detail various read and write sequences. Both minimum and expandable address decode and device response structures are introduced, first in concept, then from a practical dedication to potential applications in which one or more I/O facilities are selected to provide desired interfaces.

The subject of INTERRUPT control and interfacing is initiated with a detailed study of interrupt concepts and interrupt address organization in memory. This leads to a detailed analysis of the microprocessor's activities executed during its process of a typical external interrupt request. Control signal generation and timing are highlighted. Then, a study of interrupt hardware concepts is followed with detailed examination of priority recognition and priority resolution hardware design, including the practical scheme used in the INSTRUCTOR. Emphasis is placed on program preparation and selection of instructions to support microprocessor operation during interrupt processing. The session continues with a study of REAL TIME interrupt considerations, reinforced by the practical application "DESK CLOCK." To conclude the lesson, approaches to model register save and restore functions are described and diagrammed.

- Scope of the I/O Interface
- Definition of I/O Facilities - Single Bit
(as related to INSTRUCTOR) - Data I/O
- Control I/O
- Extended I/O
- Memory Mapped I/O
- Basic Microprocessor Timing and Control
- 2650 Microprocessor Interface Signals
- Direct Memory Access - Introduction
- 2650 Interface Concept - Bus and Control Signal Organization
- Serial I/O Concepts - Data Transfer
- Basic Theory of Data Communications
- Exercise; Practical simulation of serial data transmission

- External Device Selection Decoding Sequence - flowcharted
- Detailed Microprocessor Timing - AC characteristics
 - Memory or I/O Read Cycle Timing
 - Memory or I/O Write Cycle Timing
 - Address and Data Enable Timing
- Minimum Address Decode Structures - 1 device
 - 2 devices
 - 3-8 devices
 - 9-28 devices
- External Device Response Synchronization
- Memory and I/O Modular Interface Diagrams:
 - * Simple Memory Interface - 2K
 - * Memory Interface Expansion - to 16K
 - * Non-Extended I/O - Control and Port Description
 - * Extended I/O Interface - Address and Control Definition
 - * Direct Memory Access (DMA) Interface
 - * Memory Mapped I/O Interface
 - * 16-Bit A/D and D/A Converter Interface - synchronized
 - * Integrated A/D and D/A Converters
 - * General Purpose Serial Communications Interface

INTERRUPTS

- Scope of Interrupt - Definition and Delimit of Problem
- Polled vs. Vectored Interrupt Concepts
- Concept of Memory Organization for Interrupt Addresses
- Microprocessor Internal Interrupt Sequence - Detailed Analysis
- Interrupt Sequence Timing - AC Characteristics
 - Detailed Waveforms
- Exercise - 2650 Microprocessor Vectored Interrupt Recognition (programming considerations)
- Interrupt Hardware Concept
- Priority Resolution Hardware Concept
- Detailed Interrupt recognition and response interface - hardware configuration - memory map
- Interrupt program implementation and considerations
 - * on entry
 - * subsequent interrupt recognition before interrupt processing complete
 - * on exit from interrupt service

- Real Time Interrupts - concept
 - minimum hardware configuration for real time clock
 - INSTRUCTOR 50 Interrupt Scheme
- Exercise - Real Time Interrupt Facility - Program 'DESKCLK'
 - * Preset AM/PM, hours, minutes, seconds
 - * Synchronize to real-time of day by SENSE input
- Save/Restore Register Operations - detailed considerations and programming

ABSTRACT

One of the most essential parts of any highly technical course is its CRITIQUE. In the Critique, you are provided the opportunity to summarize your thoughts about the course....what you found to be useful, and where you found it deficient. Then, during subsequent revision cycles, we'll use your feedback to improve the course materials. And send you copies of the updated pages (or script) as they become available.

Secondly, your completion and mailing of the Critique automatically places you on the selected Signetics Product Mailing List. You'll be the first to receive detailed specs. on the latest state-of-the-art ICs. And a variety of Applications Notes and Memos to increase your knowledge, and flexible usage, of the microprocessor.

OUTLINE

- YOUR BACKGROUND
 - prior microprocessor or technical experience
 - your previous education
- YOUR OBJECTIVES
 - You'll measure your confidence to prepare a microprocessor-driven application.
- TRAINING METHOD AND CONCEPT
 - Your views on the method of presentation.
- TECHNICAL CONTENT
 - You'll tell us what parts of the course need to be beefed up or toned down.
- AUDIOVISUAL PRESENTATION
 - You'll tell us a few details concerning some very specific methods and presentation aspects of the course.
- MAILING INFORMATION FORM
 - You'll complete this form, and send it with the Course Critique to Signetics, so that we can continue to provide you with the best information possible.



2650 MICROPROCESSOR COURSE

MODULE I

CRAPGAME 2650

PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086

REFERENCE:

Outlines and Abstracts
pages 1 to 3 .

INTRODUCTION:

The Microprocessor audio-visual course is centered on your activity respecting the usage of the "INSTRUCTOR" Microprocessor Training facility. In Module 1, the CRAPGAME package, you will become completely familiarized with the operational capabilities of this facility. In itself, the "INSTRUCTOR" is really a powerful set of tools available to you in the design of microprocessors into your application. The CRAPGAME package demonstrates how many of the "INSTRUCTOR"'s capabilities have been selected, as from a tool box, to implement one such application, an interactive automated crapgame.

Just follow the directions, step by step. When directed to do so, jot down the completion of a specific requirement or an answer to a specific question. And take notes - there's plenty of room. And while you are at it, enjoy the CRAPGAME and perhaps make a wager with your associates.

One more thing: From time to time in this and other modules, you'll be asked to halt your step by step progress and listen briefly to the specified tape. On tape, you'll receive some very useful "over the shoulder" advice which is designed to answer questions or amplify on some function you'll be performing. You will hear a two second tone at the end of each short message. That's your cue to stop the tape in place and return to the exercise you are performing.

DIRECTIONS:

1. Tape 1; "CRAPGAME 2650" is currently installed in your tape deck. Do not playback further until directed to do so later in this module.
2. Complete the requirements of Exercise 1 on the next page.

EXERCISE 1PROGRAMMING "CRAPGAME" INTO THE INSTRUCTORS MEMORYINTRODUCTION:

Like all "thinking" machines (and people as well) the "INSTRUCTOR" must be told what to do. When first powered up, the "INSTRUCTOR"'s User memory is a blank slate; this is why nothing predictable happens when you depress RST** to execute a USER PROGRAM, (Try it!) In the first of several interactive exercises, you'll enable the INSTRUCTOR to execute the program "CRAPGAME" by loading its memory with that program. Simply, the program consists of a series of instructions to the computer, coded in numbers (and letters) that the computer within the INSTRUCTOR can understand.

Among its many facilities, the INSTRUCTOR contains a permanent program which permits you to load instructions (and data) rapidly into its user memory via the HEX KEYBOARD. This program is called FAST MEMORY PATCH.

FAST PATCH THEORY OF OPERATION

The following sequence takes place when FAST PATCH mode is implemented:

1. After entering FAST PATCH mode, the INSTRUCTOR's MONITOR program responds with the display: .AD=
2. The operator keys in the start address of the memory block to be loaded and depresses the ENT/NXT key.
3. The MONITOR responds with the display: .NNNN where NNNN equals the keyed address.
4. The operator keys 2 digits (0-F) to set data (1 byte) ** into memory at the address indicated.
5. The MONITOR responds with the display: .NNNN XX, where XX equals the keyed data.
6. The operator depresses 2 more digits representing the 2nd byte to be loaded as he keys the first digit, the first byte is stored in memory at the start address, and the MONITOR responds with the display: .NNN+1 Y where NNN+1 is the start address incremented by 1 and Y is the first digit. After the second digit is keyed, the MONITOR displays NNN+1 YY

* e.g.: the RESET key.

** each byte consists of 8 bits
e.g.: logic '1' or '0'.
(256 possible combinations)

7. The operator keys the first digit of the next byte*. The previously keyed byte is stored (at address NN+1) and the MONITOR displays NNN+2-Y. The sequence described in steps 6 and 7 is repeated as many times as there are bytes keyed.
8. If incorrect data digits are depressed mistakenly, the error cannot be corrected without re-entry to FAST PATCH mode at the specific address. When entering long strings of data, complete the byte by depressing any digit and circle the byte in the documentation for LATER CORRECTION.
9. The operator depresses any FUNCTION key to exit from FAST PATCH mode.

* Each byte is coded with 2 digits. Each digit represents the binary pattern of the most significant and least significant 4 bits. Since each 4-bit pattern has 16 possible combinations, the digits are called "hexadecimal numbers", or simply: HEX.

In ascending order from 0 to 15 (16 combinations) the digits are specified: $[\underline{6} (+) \underline{10}] = 16$

0,1,2,3,4,5,6,7,8,9,A(10),
B(11),C(12),D(13),E(14),
and F(15)

DIRECTION:

If you are not familiar with HEX numbering, stop your further study in Module 1, and immediatly study Module III-B "Number System Review". Then continue where you left off in Module 1.

If you are not familiar with terms and "buzz-words" used in this section (e.g.: byte, register, Program counter;) refer to the GLOSSARY; Module III-A; pages 4 through 29.

DIRECTION:

Go right on to the next page.

PROCEDURE:

1. Access the CRAPGAME - ABSOLUTE CODE table on the next page. You'll be keying in the data in this table.
2. Power up the INSTRUCTOR. Display: "HELLO"

***** PROCEDURE CHANGE *****

Perform steps 16 through 20 on pages 7 and 8, then continue with step 3. This is necessary in order to prevent possible loss of data to be loaded into the INSTRUCTOR memory during this exercise.

3. Depress **REG** **F** To enter FAST PATCH mode.
4. Depress **0** **E/N** To set the START ADDRESS of the block of memory into which data is to be stored. E/N = ENT NXT KEY.
5. Reference the CRAPGAME ABSOLUTE CODE table. Extract of table (top left side of page) is reprinted below (Diagram 1).

Diagram 1 Extract from CRAPGAME Absolute Code

ADDRESS X=	0	1	2	3	4	5	6	7	8	9	A	B	C
000X	1B	88	01	B1	01	3A	01	83	01	5C	01	0D	00
001X	77	0A	74	60	05	DC	CD	01	AF	08	74	8B	F4
002X	CD	17	89	04	7F	C2	C3	BB	84	05	17	06	87
003X	05	BB	FC	3F	01	2A	00	C1	C0	06	8F	BB	84
004X	01	00	05	01	C9	7B	20	CC	01	81	06	97	07

6. Working from left to right, top to bottom, depress Hex digit keys to enter and store the required codes in user memory from addresses '0000' to '01FF,' e.g.:

1 **B** **8** **8** **0** **1** **B** **1** **0** **1**

Display:

.0000 1b .0001 88 .0002 01 .0003 b1 .0004 01 .. etc.

..... **1** **5** **5** **2** Circle any mistakes as made.

DISPLAY ADDRS .01FE .01FF

DIRECTION:

Procedure steps continue on page 6, this module CRAPGAME absolute code on next page.

CRAPGAME - ABSOLUTE CODE

ADDRESS X=	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
000X	1B	88	01	B1	01	3A	01	83	01	5C	01	0D	00	D0	00	00
001X	77	0A	74	60	05	DC	CD	01	AF	08	74	BB	F4	CC	17	88
002X	CD	17	89	04	7F	C2	C3	BB	84	05	17	06	87	BB	FE	04
003X	05	BB	FC	3F	01	2A	00	C1	C0	06	8F	BB	84	BB	86	1B
004X	01	00	05	01	C9	7B	20	CC	01	81	06	97	07	7F	BB	84
005X	BB	88	89	6D	06	00	CA	69	E5	03	1C	00	9E	E5	09	1C
006X	00	B1	1B	D2	C0	C0	CC	01	81	0C	17	9E	CC	17	BE	0C
007X	17	9F	CC	17	BF	07	7F	06	87	BB	84	07	7F	06	B7	BB
008X	84	BB	82	BB	86	07	7F	06	97	BB	84	BB	88	E5	02	1C
009X	00	B1	C0	C0	C0	C0	E5	04	18	02	1B	59	06	00	CE	01
00AX	81	07	7F	06	9F	BB	84	07	7F	06	A7	BB	84	05	9F	9B
00BX	8C	0C	00	0E	CC	17	B7	07	7F	06	AF	BB	84	05	AF	9B
00CX	8C	E5	21	18	5A	E5	41	18	68	1F	00	64	C0	00	10	00
00DX	E5	AF	18	25	0C	00	0F	84	66	8C	00	0E	B5	01	18	09
00EX	94	C8	F2	20	CC	17	8F	1B	E4	20	C8	E9	C8	EC	C8	F5
00FX	04	FF	D4	07	75	05	1F	01	0D	0C	00	0F	AC	00	0E	E4
010X	FB	9A	66	1B	5B	12	17	0B	12	1B	17	16	17	05	01	06
011X	04	BB	FE	04	01	BB	FC	CC	00	0F	1F	00	10	00	05	02
012X	17	00	00	05	20	17	00	00	00	00	44	0F	CC	00	0E	CC
013X	17	8F	BB	82	07	7F	17	00	00	00	CA	7B	05	02	C9	79
014X	CB	76	05	17	0A	71	BB	E6	0B	6E	E7	00	14	E7	01	18
015X	02	FB	6D	09	64	F9	01	17	07	7F	1B	62	08	24	E4	07
016X	1C	01	1E	E4	11	1C	01	23	E4	03	99	11	E4	12	18	07
017X	E8	0F	18	06	05	10	17	05	40	17	05	04	17	05	08	17
018X	00	XX	XX	08	22	09	22	CC	17	98	CD	17	9B	84	66	81
019X	94	C8	6F	BB	F4	CD	17	9F	E4	01	18	02	04	17	CC	17
01AX	9E	17	13	15	01	01	17	17	17	17	05	DB	C9	01	17	17
01BX	17	76	60	09	7A	85	01	B5	01	38	6F	0D	61	00	C9	6F
01CX	D4	07	BB	F4	C8	61	C9	61	07	10	CB	64	05	01	06	A1
01DX	BB	E6	0B	5C	FB	74	B4	80	98	59	17	34	41	13	36	24
01EX	51	25	66	21	31	61	16	12	56	46	45	53	64	54	23	32
01FX	33	35	65	11	14	63	42	62	26	43	34	55	22	44	15	52
178X	10	11	0A	0C	0E	0B	0E	87	17	17	17	0B	0E	07	17	17
179X	10	0A	05	05	17	0A	17	16	17	19	19	17	17	16	17	17
17AX	1B	00	12	17	0B	0E	0A	07	07	14	0E	14	00	12	05	0E
17BX	12	17	10	0A	1B	17	17	17	05	14	00	00	07	17	17	17

←ADDR
'1FF'

7. Depress **REG** **F** To exit from and re-enter FAST PATCH mode.
8. Depress **1** **7** **8** **0** To set the START ADDRESS for data in the last section of the table.
E/N
9. Referencing step 6 format, key in the codes for addresses '1780' through '178F.'
10. Depress **MEM** To exit FAST PATCH and to enter MEMORY DISPLAY for verification and correction of memory data stored in FAST PATCH mode.
Display: .Ad =
11. Depress **0** **E/N** To set a START ADDRESS for memory data verification.
Display = .0000 1B
12. Read both parts of this step before you perform it!
If data is O.K. in the inspected location....depress **E/N**
(Refer to code table for comparison.)
The display address increments
To correct data code errors where XX equals the correct code digit pairdepress **X** **X** **E/N**
E/N is the **ENT** **NXT** key. It causes the memory address to increment.
13. Repeat step 12 for all data stored in addresses '0001' through '01FF.'
14. Repeat steps 10 and 8 To set a start address for inspection of memory from location '1780.'
in that order.

DIRECTION:

Go right on to the next page.

15. Repeat step 12 for all data stored in addresses '1780' through '17BF.' Then, go on to step 21 (next page).

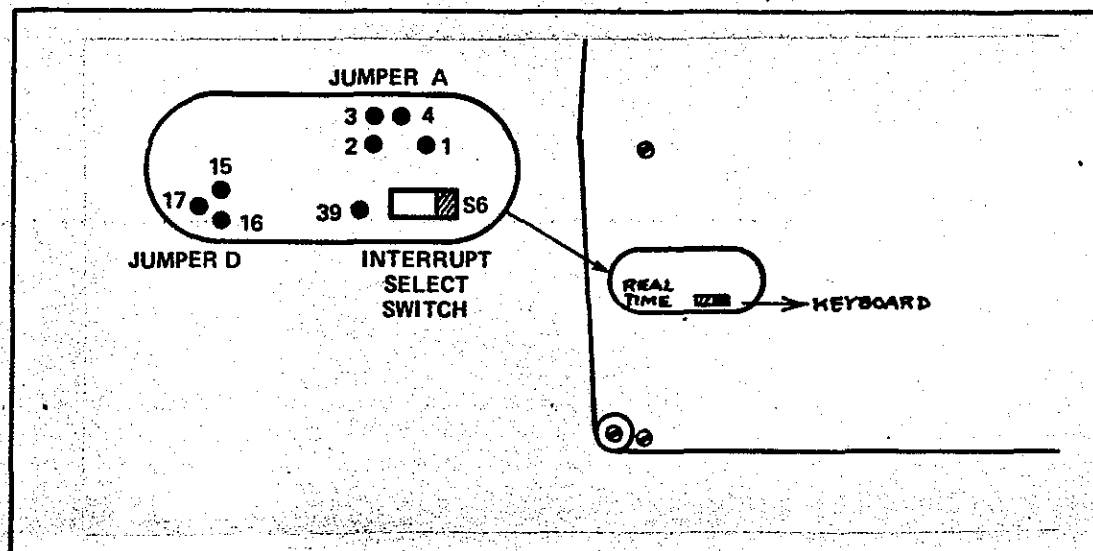
NOTE: At this point in the exercise, you have loaded the program "CRAPGAME" into the INSTRUCTOR's MEMORY and have verified its accuracy. In addition, the INSTRUCTOR's selection switches must be preset in order for CRAPGAME to execute properly.

16. Set the I/O SELECTION switch to mid-position (EXTENDED I/O PORT 07).
17. Set the INTERRUPT SELECTION switch to either position.
18. Carefully rotate INSTRUCTOR so that it is positioned FACE DOWN and front edge towards you.
19. Verify that the (REALTIME) INTERRUPT SELECTION SLIDE SWITCH (diagram 2 below) is positioned in the direction indicated by the arrow.

NOTE: "CRAPGAME" execution does not require the use of interrupts, therefore REAL TIME INTERRUPT (details in Module 5 of this course) must be defeated. The slide switch must be in the KEYBOARD position.

Diagram 2

Real Time Interrupt Switch Location



20. Restore the INSTRUCTOR carefully to its upright position.

DIRECTION: Return to step 3 on page 4 of this module.

- 21.. Listen to the brief summary of your activities involved in loading and verifying the program "CRAPGAME" into the INSTRUCTOR's memory. Further directions are provided on tape.

NOTES:

DIRECTION:

Read the PLAYING RULES FOR CRAPGAME (next page).

PLAYING RULES:

- A. This electronic CRAPGAME operates under NEVADA CASINO PASS LINE rules. On first pass of dice, the player:
- Wins if the rolls 7 or 11.
 - Loses if he rolls 2, 3, or 12.
 - Must roll again if he rolls 4, 5, 6, or 8, 9, or 10. On the first pass his "win" point is established.
- B. On subsequent rolls of the dice when "shooting his point," the player:
- Loses if he rolls a 7 (any combination).
 - Wins if he rolls his point (any combination).
 - Rolls again if he rolls neither (a) or (b) above.

WIN/LOSE TABLE - NEVADA CASINO PASS LINE PLAY

ROLL OF DICE →	2	3	4	5	6	7	8	9	10	11	12
FIRST PASS	CRAP OUT LOSES BET		SETS POINT ROLL AGAIN			WINS HOUSE PAYS	SETS POINT ROLL AGAIN			WINS HOUSE PAYS	CRAPS LOSES BET
SUBSEQUENT PASSES	ROLL AGAIN		MATCH WINS NO MATCH - ROLL			LOSES PAY HOUSE	MATCH WINS NO MATCH - ROLL			ROLL AGAIN	

LIMITATIONS:

- Bet $1 < N < 9$
- Buy into game: CHIPS = $1 < N < 99$
- Program permits 1:1 odds (even money bet) only. No doubling or redoubling.

DIRECTION:

Go right on to the next page for the SEQUENCE OF PLAY.

EXERCISE 2SEQUENCE OF PLAYINTRODUCTION:

Exercise 2 details the sequence in which the INSTRUCTOR and the USER interact to play CRAPS. Take a few minutes to read the sequence (next 3 pages) then complete the following steps:

1. Depress **REG** **C**

To access the MONITOR's "PROGRAM COUNTER DISPLAY & ALTER" routine.

NOTE: This routine must be executed to provide the microprocessor with a START ADDRESS for program execution.

2. Depress **0**

The START ADDRESS in User memory is '0000.'

3. Depress **RUN**

The INSTRUCTOR executes CRAPGAME in an interactive sequence detailed on pages 11-13.

NOTE: CRAPGAME may also be executed by OMITTING steps 1 through 3. Instead, depress **RST**!

NOTE: Depressing **RST** always resets the program counter and causes the microprocessor to execute from address '0000.'

DISPLAY	PLAYER RESPONSE	DESCRIPTION
(a) U BUY	Depress one or two numeric keys (0-9). Change if desired. Terminate by depressing any function key. (e.g., MEM, REG, BKPT)	At this point, player "buys into the game" by purchasing any number of chips between 1 and 99. His entry is displayed until he terminates the procedure.
(b) PLACE BET	Prompt by 2650 INSTRUCTOR to tell player to bet on this pass. Runs 1 to 2 seconds then next message is displayed.
(c) XX BET Y	Where XX equals number of chips on hand, Y equals previous bet, 0 or 7.
	Depress any numeric key (0-9) to place a bet on upcoming roll of dice. Depress any key except SENS,INT,MON, or RST to set bet and start the roll.	Prior to the first roll of dice, player enters his bet. The bet thus entered is frozen until he wins or loses.

GO ON TO NEXT PAGE

DISPLAYPLAYER RESPONSEDESCRIPTION

ROLL X Y

Depress **SENS** to stop the roll of dice.

The roll of dice is displayed automatically upon setting the bet. X and Y represent each die, and repeatedly change value within the limits 1-6 once the bet is set. Roll of the dice is via a random selection.

While theoretically it is possible for the player to anticipate the roll by depressing SENS at a predetermined point, in fact his "touch" is no guarantee since all combinations are possible within a 1.5 sec. time interval.

NOTE:

AT THIS POINT THE AUTOMATED MESSAGES COME THICK AND FAST!! ASSUME THAT THIS IS THE FIRST PASS WITH THE DICE.

PASS 1

This message only appears following the first roll of dice after a bet is placed.

X--X = Y

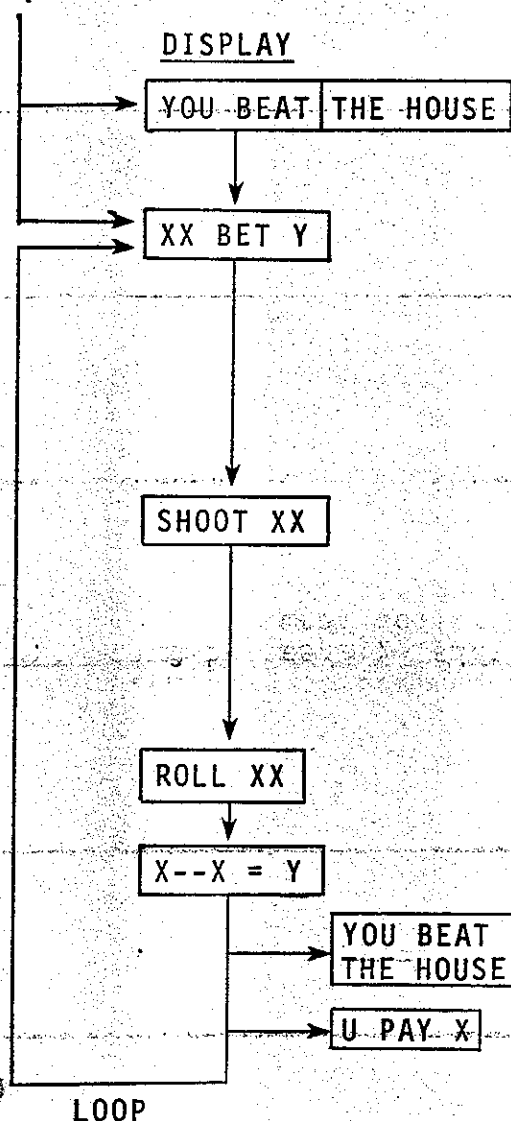
X and Y represent the face-up values of the dice upon stopping the roll of dice. Y equals the decimal sum of the two dice (2-12).

U PAY X

X is the value of previously placed bet. This message is displayed if the player rolls 2, 3, or 12 on the first roll of dice. Exit to (b) previous page.

(SEE NEXT PAGE)

(FROM PREVIOUS PAGE)

DESCRIPTION

This message is displayed if the player rolls 7 or 11 on first pass. Exit to (b) on page 11.

This message, the same as that described in (c)(p.11), reminds the player of the number of chips remaining and of the value of his current bet. No change is possible. This message is displayed if his first pass is 4, 5, 6, or 8, 9, or 10.

This message, delivered on any pass not involving a win or loss, tells the player his "point," e.g., what he must roll to win in succeeding passes. XX equals the decimal sum of dice derived in this pass. After shoot XX times out, the dice are rolled automatically ... and stopped when player depresses

SENS

Individual dice and assembled roll.

if he matches his point (and wins), exit to (b).

if he rolls a 7 (and loses), exit to (b)

If he neither matches his point nor rolls a 7; the roll continues.

U BUY = and

parallel I/O
LEDs on

If at any time, the player loses more chips than he has on hand, or if his chip total exceeds 99, this display prompts player to buy into a new game. This tells the whole world he either won or lost the game. Then, exit to (a) on page 11.

DIRECTION:

Play CRAPGAME against the INSTRUCTOR until you are thoroughly familiar with its sequence; about 10 or 15 minutes. Then go on to Exercise 3.

EXERCISE 3PROGRAM MODIFICATION - SINGLE BYTEINTRODUCTION:

In this exercise you'll use the MEMORY DISPLAY & ALTER facilities of the INSTRUCTOR to achieve clearly visible changes in the program's operation.

NOTICE:SPEED WITH WHICH RUNNING MESSAGES ARE DISPLAYEDTHEORY:

Running message display time is controlled by the value of a single byte of data in the "DISPLAY MESSAGE" routine of CRAPGAME. While you don't understand how this routine functions, you can alter this byte's value ... thus increasing or decreasing message display time to suit yourself.

MODIFICATION:

1. Depress MON

The INSTRUCTOR exits from CRAPGAME (User program) execution to the System MONITOR
Display = HELLO

2. Depress MEM 1 3 D

To select the memory address whose contents are to be altered.
Display = .013D 02 '02' is orig. data.

E/N

3. Depress 0 4

Display = .013D 04
'04' is modified data.

4. Depress E/N

To store the data ('04') in location '013D.'

5. Depress RST

To execute the program from address 0. Play for a few minutes, then respond to the following observation.

EXERCISE 3 (CONTINUED)PROGRAM MODIFICATION - SINGLE BYTE6. OBSERVATION:

By changing the data in location '13D' from 02 to 04, the time for display of each running message was _____. (increased slightly)(cut by 50%)(doubled)

DIRECTION:

Listen to Tape 1 for a short commentary.

REINFORCE:

Repeat steps 1 through 5, varying the entry in step 2 between the limits '01' and '09.' Then, settle on the message display rate YOU like best and continue to play. As you become familiar with the displayed messages, decrease the time of messages to speed up the game.

NOTICE:

ROLL RATE OF "DICE."

THEORY:

This time, the time-controlling byte is in the routine "RLDICE" located in a different part of memory.

MODIFICATION:

Given the following parameters, modify the ROLL RATE as suits you.

Memory Address:	'01C9'
Contents (original):	'10'
Change to:	'30'

OBSERVATION:

The displayed roll rate _____. (speeds up)(slows down)

DIRECTION:

Listen to Tape 1A for further commentary.

EXERCISE 3 (CONTINUED)PROGRAM MODIFICATION - SINGLE BYTEREINFORCE:

Experiment with the roll rate, controlled by data in location '1C9.' Execute steps to vary the roll rate between the limits of 1 and '7F' until you find the roll rate YOU like best. Then continue play.

THEORY:

The START ADDRESS need not always be set to the program's natural start address (e.g., '0000' for CRAPGAME). Routine RLDICE is executed from a start address of '01B1.' In this activity, hit RST and wait until a string of messages is displayed, then depress MON. Then perform the following sequence of steps.

PROCEDURE:

1. Depress MON REG C To access the MONITOR's PROGRAM COUNTER DISPLAY AND ALTER ROUTINE.
2. Depress 1 B 1 E/N To set a START ADDRESS at location '01B1.' (The start of routine RLDICE.)
3. Depress RUN To execute routine RLDICE.
4. Did it make any difference what the program was doing when you depressed MON ? _____ (yes)(no)

Was the routine RLDICE executed as soon as you depressed RUN ? _____

What happened when you depressed SENS _____

EXERCISE 3 (CONTINUED)

PROGRAM MODIFICATION - SINGLE BYTE

5. Listen to tape 1A for a short commentary.

NOTES:

DIRECTION:

Go right on to the next page.

EXERCISE 4PROGRAM DEBUG* BY MEMORY VERIFICATIONINTRODUCTION:

In this section, you'll first alter the contents of a few locations in memory. Then you'll execute the program CRAPGAME. When the "dice" are rolling, you'll notice that the parallel I/O LEDs are not being switched on and off. Your objective: to find the problem and enter the required data code to switch the LEDs on and off.

PROCEDURE:

1. Use MEMORY DISPLAY & ALTER routine to change the following memory locations:

a. Address '01C0' Problem is inserted by changing the
 '01C1' contents of these locations to 'C0.'

2. Depress **RST**. Execute program per instructions until you see the dice "rolling" in the 8 digit display.

3. OBSERVATION:

The dice appear to be changing value in the B digit display.
The parallel I/O LEDs are: _____.

- a. All ON.
- b. ON in a fixed pattern.
- c. All OFF.
- d. ON and changing pattern in time with the roll of dice in the display.

4. **DIRECTION:** Select your answer, then listen to the tape to compare your response with your narrator's.

* DEBUG: A short term that means find the problem in the program and correct it.

5. ISOLATE THE PROBLEM:

FACT • All of the program with exception of RLDICE routine works perfectly.

• Therefore, problem is in the roll routine.

DIRECTION: Go right on to the next page.

EXERCISE 4 (CONTINUED) PROGRAM DEBUG* BY MEMORY VERIFICATION6. ACCESS SUPPORTING DOCUMENTATION:

The program "CRAPGAME" is fully documented on pages 41 through 52 of this module. Obtain the program at this time. Access page 51 on which is found a listing of the routine "RLDICE."

7. VERIFY SUSPECTED MEMORY:

In MEMORY DISPLAY & ALTER MODE select a start address = '01B1.' (Don't forget to depress E/N)

On the programming form "RLDICE" at line 7, locate the address and data columns. Opposite '01B1' address is the value 76. This is the data in location '01B1.'

Depress E/N If data is O.K. Increments to next address.

Depress NU DATA E/N ... To change erroneous data to desired value where NU DATA equals appropriate hex keyboard pair.

OBSERVATION: Complete the table as indicated:

AT LOCATION	PROGRAM REQUIRES	DISPLAY INDICATES	CHANGE REQUIRED
1BE	C9		
1BF	6F		
1C0	D4		
1C1	07		
1C2	BB		

FILL IN THESE COLUMNS

Therefore, a change in data is required at locations _____ and _____.

DIRECTION:

Listen to tape 1A for a short commentary.

EXERCISE 4 (CONTINUED) PROGRAM DEBUG* BY MEMORY VERIFICATION8. CHECKOUT MEMORY MODIFICATIONS:

At your option, set a Program Counter start address at location '01B1' and depress **RUN** or depress **RST**. Then go on to the next page.

PROGRAMMING NOTES:

1. When composing a program, try to organize it in FUNCTIONAL ROUTINES.
2. Easiest way to debug is to execute the routines as separate programs, verifying each routine's performance before linking it to the rest of the program.

////////////////////////////////////

EXERCISE 5PROGRAM EXECUTION IN STEP MODEINTRODUCTION:

STEP Mode, another of the MONITOR's executable programs, is necessary when a "bug" is suspected in the User program and therefore the code itself can not be relied upon. You'll use **STEP** to execute a routine on an instruction by instruction basis. At your option you may exercise any of the following MONITOR modes after executing a single "step" without affecting the continuity of the User program.

- a. MEMORY DISPLAY AND ALTER.
- b. REGISTER DISPLAY AND ALTER.
- c. BREAKPOINT DISPLAY AND ALTER.
- d. PROGRAM COUNTER DISPLAY AND ALTER.

REFERENCE READING: STEP mode is fully described on page 62 in this Module.

EXERCISE 5 (CONTINUED)PROGRAM EXECUTION IN STEP MODE

9 times out of 10, you'll determine the nature of the problem by inspecting the contents of the microprocessor's general purpose registers and (by inspection of Registers "7" and "8") its current internal STATUS. For example, suppose a given instruction is to store the data '6B' (contained in the instruction) in register R3. You use STEP mode to execute the instruction on inspecting the registers, you find that '6B' is stored in R2, not R3. You can then use MEMORY DISPLAY and ALTER mode to correct the instruction in memory - immediately.

PROCEDURE:

1. Set a start address (01B1) into the Program Counter. You'll be executing routine RLDICE in STEP mode.

2. Depress **STEP** Display = ____

3. Inspect the CPU's registers R0-R3

Depress **REG** **0** ... **E/N** ... **R/N** ... **E/N**

OBSERVATION: R0 = ____ R1 = ____ R2 = ____ R3 = ____

4. Repeat steps 2 and 3

OBSERVATION: Display after **STEP** is depressed

Display for R0 = ____ R1 = ____ R2 = ____ R3 = ____

5. Inspect the Program Counter (address of next instruction to be executed).

Depress **REG** **C** Display: .PC = ____

6. Repeat steps 4 and 5

OBSERVATION: Display after **STEP** is depressed: ____

Display for R0 = ____ R1 = ____ R2 = ____ R3 = ____

Display for Program Counter (next instruction):
PC = ____

DIRECTION: Listen to the tape for a brief commentary on the observation taken in step 6.

EXERCISE 5 (CONTINUED)PROGRAM EXECUTION IN STEP MODE

7. Continue stepping through address '1C2.'

OBSERVATION: Next address after '1C2' is '____.'

8. Depress **STEP**

OBSERVATION: Display = _____

RO = _____

DIRECTION: Go right on to Exercise 6.

////////////////////////////////////

EXERCISE 6PROGRAM EXECUTION IN BREAKPOINT MODEINTRODUCTION:

While STEP mode is very useful for program debugging on a step-by-step basis, there are many conditions in which it may be desirable to exercise proved sections of the program at the microprocessor's normal speed. The following are examples of such conditions:

- a. When execution of a sequence consisting of one or more instructions is to be repeated many times.
- b. When externally controlled I/O devices must be exercised at normal service rates (e.g., at the microprocessor's normal instruction execution rate).

EXERCISE 6 (CONTINUED)PROGRAM EXECUTION IN BREAKPOINT MODE

- c. When executing instructions which cause transfer of program control to a routine contained in the INSTRUCTOR's MONITOR program.

NOTE: Attempt to step into a MONITOR routine will cause the MONITOR to generate an ERROR message.

What then is BREAKPOINT mode? Simply, BREAKPOINT is a mode in which the user can define any location in memory as a user-program ending address. When the MONITOR detects this address while executing the USER program in RUN mode, execution of the USER program terminates and the BREAKPOINT address is displayed. The user may then inspect registers and memory locations as you did in STEP mode. Or you may set a new BREAKPOINT. By setting the Program Counter to a known START ADDRESS and by setting BREAKPOINT to a known ENDING ADDRESS, you establish precise start/stop program execution limits.

READING REFERENCE:

Prior to performing the sequence of steps in Exercise 6, you may wish to review the documentation concerning BREAKPOINT. Page references are provided as follows:

INSTRUCTOR 50 REFERENCE MANUAL

This module: page 61.

PROCEDURE:

1. Set a START ADDRESS for program execution at address '01C0.'
2. Depress **BKPT** To enter BREAKPOINT mode.
Display: .b.p =
3. Depress **1** **C** **4** **E/N** To enter and set a program execution ENDING address
Display: b.p - 1C4

OBSERVATION: RO = ____

EXERCISE 6 (CONTINUED)PROGRAM EXECUTION IN BREAKPOINT MODE

4. Depress **RUN** To execute program from address 1C0 through 1C4.

5. Depress **BKPT 1 C 0 RUN** Display R0 = ____

6. Depress **BKPT 1 C 4 RUN** Display R0 = ____ R1 ____

7. Repeat steps 5 and 6 (8 times) completing TABLE 1 after executing program as indicated.

TABLE 1	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
AFTER EXECUTING BKPT AT ADDR '1C0', R0 =								
AFTER EXECUTING BKPT AT ADDR '1C4', R0 =								
R1 =								

OBSERVATION: From the data gathered in step 7, you verify that the byte of data located in R₀ after execution of a BKPT at address '1C0' is split into 2 NIBLs (half bytes) located in R₀ and R₁ after execution of the BKPT at location '1C4.'

Listen to tape for verification of your answer.

- B. Using step 5 as a model, set a BKPT at address '1DA' and depress **RUN** to execute.

OBSERVATION: _____

EXERCISE 6 (CONTINUED) PROGRAM EXECUTION IN BREAKPOINT MODE

9. Now, depress the **SENS** key to stop the roll.

OBSERVATION: Display = ____

10. Depress **RUN**

OBSERVATION: _____

Listen to audio tape for further commentary.

11. Depress **SENS** to stop the roll again.

12. Using step 5 as a model, set a BKPT at location '1D8.'

Depress **RUN** . Listen to audio tape for further commentary.

OBSERVATION: Inspect the Program Counter and jot down the location of the next instruction.

REG **C** Display: .PC ____

13. Now, depress **STEP**. Display: ____

14. Depressing **STEP** sequentially, execute the routine RLDICE until the address '01BE' is displayed.

OBSERVATION: R0 = ____ R1 = ____

15. Add the contents of R1 to '100.' '100' + ____ = ____
Address '1__'

16. Referring to the last page of program documentation (page 52 in this module), jot down the contents of the location determined in step 15:

At location '1 __,' data should be: ' __ .'

EXERCISE 6 (CONTINUED)PROGRAM EXECUTION IN BREAKPOINT MODE

17. Now inspect memory at that precise location; i.e.:

Depress ☐ MEM ☐ 1 ☐ X ☐ X ☐ E/N , where XX = data in R1

OBSERVATION: At location '1 _ _,' data = _ _

. My prediction in step 16 ____ (was)(was not) correct.

. Further, it ____ (agrees)(disagrees) with data observed in R0 during step 14.

Listen to brief commentary on audio tape.

NOTE: _____

Conclusion: _____

DIRECTION:

Perform the following sequence, using STEP mode to observe effect of single instruction execution.

1. Inspect the Program Counter and jot down the location of the next instruction.

☐ REG ☐ C Display: PC = _ _ _ _

2. Verify the contents of R1.

☐ REG ☐ 1 Display: R1 = _ _

3. On line 13 of the routine RLDICE (sheet 1, page 51 this module) read the symbolic instruction.

OBSERVATION: Symbolic Instruction:

OPCODE OPERANDS

EXERCISE 6 (CONTINUED)PROGRAM EXECUTION IN BREAKPOINT MODE

4. Using the programming form's LABEL field as a guide, locate the address of DICEXT.

OBSERVATION: "DICEXT" is a ____, (1)(2)(3) byte field at location '____'.

Verify your observation with that provided on audio tape.

5. Now, determine the PRESENT contents of "DICEXT":

Depress MEM 1 A F E/N

OBSERVATION: Display .01AF ____, then display R1 = ____

6. Now, depress STEP to execute the instruction defined in step 1.

" REG 1 to examine R1 contents.
R1 = __

Depress MEM 1 A F E/N to examine DICEXT.

DICEXT = __

OBSERVATION: The instruction executed at location '1BE':

Listen to the audio tape to verify your observation.

7. Perform steps 1 and 2, substituting R0 for R1 in step 2.

PC = ____ R0 = ____

8. Perform step 3, substituting line 14 for line 13 in the program.

Symbolic Instruction:

OPCODE OPERANDS

EXERCISE 6 (CONTINUED)PROGRAM EXECUTION IN BREAKPOINT MODE

9. In the left hand block provided, indicate the condition of the parallel I/O LEDs before execution of the instruction at location '1C0.'

○○○○○○○○ BEFORE EXECUTION OF INSTRUCTION AT LOCATION '1C0'	ON: BLK OFF: WHT	○○○○○○○○ AFTER EXECUTION OF INSTRUCTION AT LOCATION 1C0
--	---------------------	---

10. Depress **STEP**. Repeat step 9, except use the right hand block.
11. Establish a correlation between the contents of R0 and the turn-on of the parallel I/O LEDs by depressing the following keys, and completing a TABLE 2 entry AFTER the parallel I/O LED pattern changes.

PROCEDURE:

Depress:

RUN
STEP
STEP
STEP
STEP
STEP
STEP
STEP

Example: →

TABLE 2

R0	PARALLEL I/O LED'S
32	● ● ○ ○ ● ● ○ ●
	○ ○ ○ ○ ○ ○ ○ ○
	○ ○ ○ ○ ○ ○ ○ ○
	○ ○ ○ ○ ○ ○ ○ ○
	○ ○ ○ ○ ○ ○ ○ ○
	○ ○ ○ ○ ○ ○ ○ ○
	○ ○ ○ ○ ○ ○ ○ ○
	○ ○ ○ ○ ○ ○ ○ ○

Then...

complete an en-
 try in Table 2
 and repeat procedure
 until all entries in
 Table 2 are completed. Then go on
 to step 12 on the next page.

EXERCISE 6 (CONTINUED) PROGRAM EXECUTION IN BREAKPOINT MODE

12. In the space provided, jot down your conclusion of the effect upon the parallel I/O LEDs caused by execution of the WRTE, RO instruction at location '1C0.'

13. Now, compare your response with that provided on tape.

NOTES:

EXERCISE 7WRITING A PROGRAM TO TAPEINTRODUCTION:

Recall that you used considerable time, for example, in keying the program CRAPGAME into User memory at the start of this module. The INSTRUCTOR contains hardware and fixed routines which make it possible to store programs which you'll be exercising on tape. In this exercise, you'll store the entire program "CRAPGAME" on tape as a permanent file. When you reload CRAPGAME (in a later exercise) you'll see it only takes about 2 minutes time.

READING REFERENCE:

The following pages provide considerable explanation of the WRITE CASSETTE features of the INSTRUCTOR.

- INSTRUCTOR 50 REFERENCE MANUAL
- Documentation in this MODULE: — page 60.

DIRECTION:

Take a few minutes to read these references, then go on to the procedure on the next page.

EXERCISE 7 (CONTINUED)WRITING A PROGRAM TO TAPEPROCEDURE:

1. Prepare your tape deck for operation in RECORD mode.
2. Ensure that the I/O SELECTION SWITCH is in the "Extended I/O Port 07" position. Do not rewind Tape 1.
3. Obtain tape 2 "SCRATCHTAPE" from the album. Install it (side A up) in your tape deck.

NOTE: This 30 minute unrecorded tape (15 minutes each side) is provided for your convenience in storing and retrieving the programs you write. WE RECOMMEND THAT YOU WRITE EACH PROGRAM TWICE ON TAPE IN ORDER TO OVERCOME PROBLEMS DUE TO TAPE IMPERFECTIONS!

4. Connect the audio cable from the INSTRUCTOR's "MIC" jack to the MICROPHONE input of your tape deck.
5. Playback the tape briefly until your deck's recording heads are positioned over tape (e.g., off of clear leader). Then STOP the tape recorder.
6. Depress MON WCAS To access WRITE CASSETTE mode
Display: L.Ad. =
- Depress 0 E/N To set the lower (start) address
of memory to be written
Display: U.Ad. =
- Depress 1 F F E/N To set the upper (ending) address
of memory to be written.
Display: S.Ad. =
- Depress 0 E/N To set a START address for pro-
gram execution. The stored
start address is loaded into
the Program Counter when tape
is read back into the INSTRUCTOR.
You'll simply depress RUN
to execute
Display: .F =
- Depress 1 To set a FILE NUMBER.
Do not depress E/N yet.

EXERCISE 7 (CONTINUED)WRITING A PROGRAM TO TAPE

7. At this point, place your tape deck in RECORD mode and immediately

8. Depress E/N

CRAPGAME will be stored on tape.
When complete, Display: HELLO
Record time ~ 1:40.

9. Stop your tape recorder

CRAPGAME is stored in 2 sections of memory. You stored the first section in steps 6-9 as File 1. The second section (File 1A) is stored as follows.

	Before <u>E/N</u>	After <u>E/N</u>
10. <u>Depress</u> <u>WCAS</u>	<u>Display:</u> L.Ad =	
<u>Depress</u> <u>1</u> <u>7</u> <u>8</u> <u>0</u>	<u>Display:</u> L.Ad = 1780	U.Ad =
<u>E/N</u>		
<u>Depress</u> <u>1</u> <u>7</u> <u>B</u> <u>F</u>	<u>Display:</u> U.Ad = 17BF	S.Ad =
<u>E/N</u>		
<u>Depress</u> <u>0</u> <u>E/N</u>	<u>Display:</u> S.Ad = 0	.F =
<u>Depress</u> <u>1</u> <u>A</u>	<u>Display:</u> .F = 1A	
11. <u>Repeat</u> steps <u>7</u> through <u>9</u>	<u>Display:</u> (BLANK) after <u>E/N</u>	
	<u>Display:</u> "HELLO" after File 1A dump to tape is complete.	
	Record time 0:20.	

DIRECTION:

Go right on to the next page.

EXERCISE 7 (CONTINUED)WRITING A PROGRAM TO TAPE

12. Repeat steps 6 through 11 ... to store CRAPGAME for the second time.

NOTE 1: If your tape deck has a counter, jot down the count and rewind the SCRATCHTAPE. Jot down the 2nd count and subtract from the first.

NOTE 2: Documentation - Records - Keeping:

On a separate sheet of paper (or in a notebook) keep a record of programs recorded on the SCRATCHTAPE. You'll save time later when you have to access 1 of several taped programs.

e.g.

CRAPGAME Files 1, 1A (twice) cts 0-47
DESCLK File 3 (twice) cts 50 - 67

If your tape deck does not have a counter, you should at least keep a record of the sequence in which files are stored.

NOTE 3: For this course, dedicate SCRATCHTAPE Side A as a permanent file, and Side B as a practice tape.

13. Fast-Forward SCRATCHTAPE to end of tape and turn it over to Side B.

14. Repeat step 5, then go on to the next exercise. Leave the cable connections in their WRITE CASSETTE configuration.

////////////////////

EXERCISE 8COMPLETE TAPE CASSETTE OPERATIONSINTRODUCTION:

In this procedure you'll store the routine "RLDICE" on SCRATCH-TAPE Side B. The INSTRUCTOR's User memory is then erased by pulling the power plug. You'll then perform the ADJUST CASSETTE procedure to set playback level of your tape deck for compatibility with the INSTRUCTOR. Then you'll implement the INSTRUCTOR's "READ CASSETTE" mode to restore the routine RLDICE to User memory. And finally, you'll use INSTRUCTOR facilities previously mastered to permit execution of RLDICE as an INDEPENDENT program.

PROCEDURE:

1. Obtain the listing of routine RLDICE (pages 51 and 52); this module. Set aside for later use.
2. WRITE routine RLDICE to tape. Use step 6 or 10 (preceding exercise) as a guide.

PARAMETERS: L.Ad = 1AA beginning memory address
U.Ad = 1FF ending " "
S.Ad = 1B1 start address - execution
F = 3

3. Repeat step 2 on blank tape.
4. Rewind tape to start of Side B.
5. Play back tape (at low volume) for about 20 seconds. After going off clear leader, you'll hear about 5 seconds of "ringing" tone, then about 7-15 seconds of ringing that sounds quite "scratchy." If you hear nothing then perform step 6, otherwise go on to step 7.
6. Ensure that the recorder is connected as described in the Reference Manual. Ensure that the jumper for output tape record selection (mic/aux) is installed per the jumper installation procedure in the Reference Manual; then repeat steps 2 through 5.

DIRECTION:

Go right on to the next page.

EXERCISE 8 (CONTINUED)COMPLETE TAPE CASSETTE OPERATIONS

7. Remove the "INSTRUCTOR"'s power pack from the AC wall socket. Wait a few seconds, then re-install it. Display: "HELLO."

8. Set the Program Counter to zero and execute:

REG **C** **O** **RUN**

OBSERVATION: Display: _____

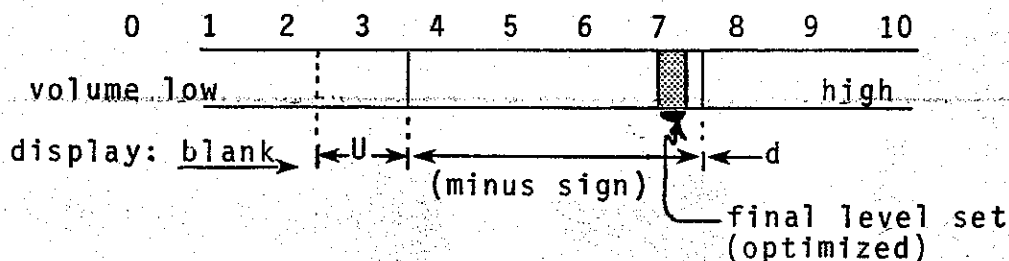
9. Remove and set aside the SCRATCHTAPE. Reinstall and listen to Tape 1A for a short commentary.

10. Reinstall SCRATCHTAPE (Side B up), and restore the POWER PACK to the AC wall socket. Set aside Tape 1.

11. Reference the ADJUST CASSETTE procedure in the INSTRUCTOR 50 REFERENCE MANUAL; Section 4: Read, then perform the INSTALLATION AND ADJUSTMENT procedures in that manual, also the NOTES below.

NOTE: If your tape deck has a TONE control, preset it in the TREBLE range prior to making the CASSETTE OUTPUT LEVEL ADJUSTMENT on a scale of 0 (full bass) to 10 (full treble), 8 or 9 is a good setting. By placing the tone control in the treble range, you permit data (consisting of coded pulses) transfer between tape deck and INSTRUCTOR with greater integrity. Thus a more flexible adjustment range is possible.

Your narrator has found that the ADJUST CASSETTE procedure can be OPTIMIZED by positioning the tape deck's level control to a point just below a setting which would cause the "d" to be displayed. See the diagram (below):



12. Rewind the tape.

DIRECTION: Go on to the next page.

EXERCISE 8 (CONTINUED)COMPLETE TAPE CASSETTE OPERATIONS

13. Take a few minutes to read the READ CASSETTE documentation on page 59 in this module.
14. Referring as necessary to the READ CASSETTE installation and operation procedure contained in the REFERENCE MANUAL, perform the following steps:

Depress MON RCAS

To access the INSTRUCTOR's
READ CASSETTE mode.

Display: HELLD then .F =

Depress 3 E/N

To recognize File 3 (RLDICE)
in readback.

Display: .F = 3, then blank

Start the tape recorder in
PLAYBACK mode.

15. If no errors (see REF. MANUAL), then display: HELLO after File 3 is stored in User memory. REWIND the tape.
16. Referring to the routine "RLDICE," verify that data read from tape was loaded successfully into the INSTRUCTOR's memory, using the MEMORY DISPLAY routine.

Verify all memory locations from addresses '1AA' through '1FF.'

DO NOT ATTEMPT TO VERIFY LOCATIONS '1A2' THROUGH '1A9.'

17. Disconnect the interconnecting audio cable. Re-install Tape 1 in the tape deck.
18. Listen to the tape for commentary on steps 19 and 20.
19. Depress MEM 1 D A E/N B O E/N

Refer to line
27 of routine
"RLDICE"

Replace '17'
with 'B0' to
make routine
RLDICE operate
as a standalone

DIRECTION:

Go right on to the next page.

EXERCISE 8 (CONTINUED)COMPLETE TAPE CASSETTE OPERATIONS

20. Depress MEM 0 E/N 1 F E/N 0 1 E/N B 1 E/N

Add-
ress .0000 1F .0001 01 .0002 B1 .0003 ...

This allows the routine "RLDICE" to be accessed and executed simply by depressing RST.

NOTE: By performing steps 19 and 20, you have done all that is necessary to permit routine "RLDICE" operation as an INDEPENDENT PROGRAM.

21. Depress RST

OBSERVATION: Display (digit): 1 2 3 4 5 6 7 8 (use X if value is changing)

Parallel I/O LEDs 0 0 0 0 0 0 0 0.

22. Listen to audio tape for short commentary.

23. Refer to sheet 2 (last page) of the routine "RLDICE"; lines 3 through 5. (Reference: page 52 in this module.)

Depress SENS To exit to the Monitor.

Depress REG F To enter Fast Memory Patch Mode.

Depress 1 A 2 E/N To set a START ADDRESS for memory Fast Patch.

24. Starting at data Byte 0, line 3 on sheet 2, depress the following keys:

1	3	1	5	0	1	0	1	1	7	0	0	1	7	0	0
1A2	1A3	1A4	1A5	1A6	1A7	1A8	1A9								

DIRECTION:

Go right on to the next page.

EXERCISE 8 (CONTINUED)COMPLETE TAPE CASSETTE OPERATIONS

25. Depress E/N To cease Fast Memory Patch
26. Depress RST To execute the program "RLDICE"
- OBSERVATION: Display: (use X if value is changing)

////////////////////////////////////

EXERCISE 9USE OF I/O SWITCHES TO FINE-TUNE VALUESINTRODUCTION:

Recall in Exercise 3 (page 14 this module) that you employed the INSTRUCTOR's MEMORY DISPLAY and ALTER routine to alter the "dice" roll rate. There is an easier way to try out (and fine tune) values until you are satisfied with the result. This method involves substituting an instruction which reads the PARALLEL I/O SWITCHES for the instruction which directly specified the value in question. Having substituted the instruction, you need only execute the desired routine, and manipulate the switches until the desired result is achieved. Then, note the value of the switches, translating their position into a 2 digit hex code. The original instruction is then restored with the hex code value placed in the 2nd byte of the instruction.

NOTE: Your effective application of this procedure is not possible until you have completed Module 4 of the course. However, the following step-by-step procedure is provided so that you can see its effect directly. In its place, use of the PARALLEL I/O switches can constitute a very flexible tool for program debugging purposes.

PROCEDURE:

1. Access routine RLDICE documentation (page 51 this module) once again and locate line 18. This is the instruction you'll be deleting temporarily. Read the comment field pertaining to this instruction (lines 17 and 18).

EXERCISE 9 (CONTINUED) USE OF I/O SWITCHES TO FINE-TUNE VALUES2. Depress **MON** **MEM** **1****C** **8** **E/N**

To set a start address in MEMORY DISPLAY and ALTER mode.

Display: .01C8 07 after **E/N**3. Depress **5** **7** **E/N****0** **7** **E/N**

This is the code for the SU8-SSTITUTE INSTRUCTIDN. When executed, it will read the PARALLEL I/O SWITCH pattern into Register R3.

Display: 01CA C8 after 2nd **E/N**

4. Toggle the left most PARALLEL I/O SWITCH (#7) OFF (towards you)

Leave it off otherwise the routine will hang up.5. Depress **RST**

To execute program "RLDICE."

6. Manipulate the 7 remaining PARALLEL I/O SWITCHES (#6 - 0) until the roll rate is satisfactory.

POSITION: { AWAY FROM YOU = ON = LOGIC 1.

{ TOWARD YOU = OFF = LOGIC 0

7. Translate the switch pattern to HEX CODE as indicated in the following table.

ON =	1	1	1	1	1	1	1	1	1
SWITCH =	7	6	5	4	3	2	1	0	
OFF =	0	0	0	0	0	0	0	0	0

	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1
2	0	0	0	1	0	0	0	0
3	0	0	0	1	1	0	0	0
4	0	1	0	0	0	0	0	0
5	0	1	0	0	1	0	0	0
6	0	1	1	0	0	0	0	0
7	0	1	1	1	0	0	0	0
8	1	0	0	0	0	0	0	0
9	1	0	0	0	1	0	0	0
A	1	0	1	0	0	0	0	0
B	1	0	1	1	0	0	0	0
C	1	1	0	0	0	0	0	0
D	1	1	0	0	1	0	0	0
E	1	1	1	0	0	0	0	0
F	1	1	1	1	0	0	0	0

illegal
for this
exercise
only

FIRST HEX DIGIT

SECOND HEX DIGIT

EXERCISE 9 (CONTINUED) USE OF I/O SWITCHES TO FINE-TUNE VALUES

Examples: 1. Switches 4, 3, 2 ON; all others OFF;
Code translation is '1C'

2. Switches 3 and 1 ON; all others OFF
Code translation is '0A'

3. Switch 5 ON; all others OFF
Code translation is '20'

4. All switches except 7 ON.
Code translation is '7F'

8. Repeat step 2

To initiate the restoration of the original instruction.

9. Depress

0	7	E/N
X	X	E/N

Where XX = the 2 digit code translation obtained in step 7.

10. Depress

RST

Your program now runs with the roll delay you desired. The PARALLEL I/O SWITCHES have no effect.

DIRECTION:

Listen to the tape for a summary discussion of activities covered in Module I and a review of the INSTRUCTOR's operational functions.

NOTES:

EXERCISE 9 (CONTINUED) USE OF I/O SWITCHES TO FINE-TUNE VALUESCLOSING CHECKLIST:

1. Using reference and instructional documentation previously studied, reload program CRAPGAME from your permanent tape (Tape 2 Side A) into the INSTRUCTOR's memory. Recall that there are 2 files: 1 and 1A.
2. Rewind Tape 2 ("SCRATCHTAPE") to the start of Side A. Store it.
3. Fast Forward Tape 1 ("CRAPGAME 2650") to the end of Side B. Store it.
4. At your option, review the short form procedures for INSTRUCTOR operation. (Module I documentation (pages 1-53) to its proper sequence in the binder.
5. Access Module II training documentation, open to page 1.

DIRECT RELATIVE ADDRESSING - SECOND BYTE

+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
H	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71	
+	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
H	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	7D	7E	7F	6D	6C	6B	6A	69	68	67	66	65	64	63	62	61
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
H	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
-	60	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52	51
+	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
H	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
-	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42	41

INDIRECT RELATIVE ADDRESS: Add H'60' TO DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE CRAPGAME START ADDR 0DESCRIPTION FILE B, BA; Start addr: 0

ADDR '02-0D = INDIRECT ZBSR/ZBAR vector

ROUTINE SHEET 1 OF 6 MAIN PROGRAM

MEMORY LOCATIONS THIS SHEET _____

	ADDRES	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	0000	1B	88		CRAPGAME	BCTR, UN	* CHIPSET	Branch to start crapgame
2	2	01	B1				* RLDICE	ROLL THE DICE
3	4	01	3A				* DISMES	(RCONS) DISPLAY MESSAGE
4	6	01	83				* ASMROL	INDIR. ASSEMBLE ROLL
5	8	01	5C				* DCDROL	TO DECODE THE ROLL
6	A	01	0D				* CHIPSET	BUY INTO GAME
7	C	00	D0				* WINNINGS	CALC. WIN/LOSS
8	E	XX					BET	CURRENT BET AND
9	0F	XX					CHIPS	CHIPS ON HAND
10	10	77	0A		NEWGAME	PSSL	WC, COM	Set WC and logical compr.
11	2	74	60			CRSU	FLAG, II	Clear Flag and Int. inhib.
12	4	05	DC			LODI, R1	H'DC'	Now, set DIC TABLE cons-
13	6	CD	01	AF		STRA, R1	DICEXT	tant and store away.
14	9	08	74			LODR, R0	CHIPS	Now, get chips on hand
15	B	BB	F4			ZBSR	* DISLSD	and break into MS and
16	1D	CC	17	88		STRA, R0	XXBET=	LS NIBLS. Store these in
17	20	CD	17	89		STRA, R1	XXBET=+1	the XXBET= table.
18	3	04	7F			LODI, R0	H'7F'	Now, set up running
19	5	C2				STRZ	R2	display to PLACE BET.
20	6	C3				STRZ	R3	Now, ...:
21	7	BB	84			ZBSR	* DISMES	display that message.
22	9	05	17			LODI, R1	<XXBET=-1	then set up constants to
23	B	06	87			LODI, R2	>XXBET=-1	place actual bet. It is
24	D	BB	FE			ZBSR	* MOV	entered via the keyboard
25	2F	04	05			LODI, R0	5	One digit limits 1<X<9
26	31	BB	FC			ZBSR	* GNPA	NO CHANGE! Next digit
27								exits routine.

DIRECT RELATIVE ADDRESSING - SECOND BYTE

+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
H	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71	
+	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
H	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	70	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62	61
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
H	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
-	60	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52	51
+	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
H	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
-	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42	41

INDIRECT RELATIVE ADDRESS: Add H'60' to DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE CRAPGAME START ADDR _____

DESCRIPTION _____

ROUTINE SHEET 2 OF 6 MAIN PROGRAM

MEMORY LOCATIONS THIS SHEET _____

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	0033	3F	01	2A		BSTA, UN	CORRSUB	Branch to correct subrou-
2	6	00	C1		DCDCOR	ACON	PASS1XP	tine - Program expansion
3	8	CO				NOP		was necessary. On re-
4	9	06	BF		PASS1	LODI, R2	BF	turn, set INDEX to PASS1=
5	B	BB	84			ZBSR	* DISMES	Display Message, then
6	D	BB	86			ZBSR	* ASMR0L	Assemble 1 st roll of dice
7	3F	1B	01			BCTR, UN	\$+3	Branch around PASS1 cont.
8	41	XX			PASS1X	RES	1	PASS1 IDENT CONSTANT loc
9	2	05	01			LODI, R1	1	Now set PASS1 ID. and
10	4	C9	7B			STRR, R1	PASS1X	store. Then, clean
11	6	20				EORZ	R0	previous saved dice roll
12	7	CC	01	B1		STRA, R0	SAVPAS1	Now, set constants for
13	A	06	97			LODI, R2	H'97'	display of
14	C	07	7F			LODI, R3	H'7F'	X--X=YY (timed out)
15	4E	BB	84			ZBSR	* DISMES	and display. Then,
16	50	BB	88			ZBSR	* DCDROL	decode this dice roll
17	2	89	6D			ADDR, R1	PASS1X	and identify as PASS1
18	4	06	00			LODI, R2	0	then clean the PASS1 ID.
19	6	CA	69			STRR, R2	PASS1X	for subsequent rolls.
20	8	E5	03			COMI, R1	3	Was 1 st pass a 7?
21	A	1C	00	9E		BCTA, EQ	BEATP1	Yes! Player won. Exit
22	D	E5	09			COMI, R1	9	No! Was it craps? (2 or 3)
23	5F	1C	00	B1		BCTA, EQ	BOMBOUT	Yes! player lost. Exit.
24	62	1B	D2			BCTR, UN	* DCDCOR	No! Now, check other decodes
25	4	CO			PASCON	NOP		via exit to PASS1 EXPAN-
26	65	CO				NOP		sion (*) at loc. '36' On re-
27								turn, wind up PASS1.

DIRECT RELATIVE ADDRESSING - SECOND BYTE															
+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E
+	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71
+	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E
+	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
-	70	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62
+	28	21	22	23	24	25	26	27	20	29	2A	2B	2C	2D	2E
+	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
-	60	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52
+	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E
+	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
-	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42
+	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54

INDIRECT RELATIVE ADDRESS: Add H'60' to displacement

2650 PROGRAMMING FORM

ROUTINE CRAPGAME START ADDR _____

DESCRIPTION _____

ROUTINE SHEET 3 OF 6 MAIN PROGRAM

MEMORY LOCATIONS THIS SHEET _____

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1								No, roll was NOT 2, 3, 7,
2								11, or 12, so player must
3	0066	CC	01	81	PASS1	STRA, R0	SAVPAS1	continue roll. So save
4	9 0C	17	9E		* CONTINUE	L0DA, R0	XXFOR+6	the first roll. Then set
5	C CC	17	BE			STRA, R0	SHOOT+6	up the display table to tell
6	6F 0C	17	9F			L0DA, R0	XXFOR+7	him what point he has
7	72 CC	17	BF			STRA, R0	SHOOT+7	to roll. Then prompt his
8	5 07	7F			CONROL	L0DI, R3	H'7F'	memory about his cur-
9	7 06	87				L0DI, R2	H'87'	rent chips and bet by
10	9 BB	84				ZBSR	* DISMES	displaying XX BET Y
11	B 07	7F				L0DI, R3	H'7F'	Now, tell him the point
12	D 06	B7				L0DI, R2	H'87'	to shoot for by display
13	7F BB	84				ZBSR	* DISMES	of SHOOT ZZ. On rtn,
14	81 BB	82				ZBSR	* RLDICE	roll dice. When he hits
15	3 BB	86				ZBSR	* ASMROL	SENS, stop roll, assemble
16	5 07	7F				L0DI, R3	H'7F'	then show him the
17	7 06	97				L0DI, R2	H'97'	roll of dice by display
18	9 BB	84				ZBSR	* DISMES	X--X=YY. Then de-
19	B BB	88				ZBSR	* DCDROL	code the latest roll.
20	D E5	02				COMI, R1	2	Was it a 7?
21	6F 1C	00	B1			BCTA, EQ	BOMBOUT	Yes! he lost this pass
22	92 C0	C0				NOP		Exit to tell him so.
23	4 C0	C0				NOP		No! Then did he match
24	6 E5	04				COMI, R1	4	his point?
25	8 18	02				BCTR, EQ	BEATP1-2	Yes! He won. Tell him.
26	9A 1B	59				BCTR, UN	CONROL	No! Didn't match. Didn't
27								lose. Roll again.

ROUTINE CRAPGAME START ADDRS.....

DESCRIPTION _____

ROUTINE SHEET 4 OF 6 MAIN PROGRAM

MEMORY LOCATIONS THIS SHEET

page 44

DIRECT RELATIVE ADDRESSING - SECOND BYTE															
+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E
-	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
+	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C
-	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B
+	3C	3D	3E	3F	40	41	42	43	44	45	46	47	48	49	4A
-	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59
+	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67	68
-	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74	75	76	77
+	78	79	7A	7B	7C	7D	7E	7F	80	81	82	83	84	85	86
-	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95
+	96	97	98	99	9A	9B	9C	9D	9E	9F	00	01	02	03	04
-	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
+	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22
-	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31
+	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	40
-	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
+	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E
-	5F	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D
+	6E	6F	70	71	72	73	74	75	76	77	78	79	7A	7B	7C
-	7D	7E	7F	80	81	82	83	84	85	86	87	88	89	8A	8B
+	8C	8D	8E	8F	90	91	92	93	94	95	96	97	98	99	00
-	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F

INDIRECT RELATIVE ADDRESS: Add H'80' TO DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE CRAPGAME START ADDR _____

DESCRIPTION _____

ROUTINE SHEET 5 OF 6 MAIN PROGRAM

MEMORY LOCATIONS THIS SHEET _____

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1								First, find out if player
2	00CD	00	10		*NEWGAME			won or lost. If he won,
3	CF 00				* Fill in data			const = 9F, ∴ no compare
4	00D0	E5	AF		WINNINGS	COMI, R1	UPAY	If he lost, compare mat-
5	2 18	25				BCTR, EQ	CALCLOS	ches AF const, so EXIT
6	4 0C	00	0F		WON	LODA, R0	CHIPS	to calc loss. HE WON.
7	7 84	66				ADDI, R0	H'66'	Fetch chips and bet and
8	9 8C	00	0E			ADDA, R0	BET	add (decimally). Is the
9	C B5	01				TPSL	CARRY	result ≥ 100? YES! Exit
10	DE 18	09				BCTR, EQ	GAMEOVER	to finish this game. No!
11	E0 94				WINDUP	DAR	R0	Adjust chips and bet result
12	1 C8	F2				STRR, R0	* CHIPS	and store as new chips.
13	3 20					EORZ	R0	He's finished with current
14	4 CC	17	BF			STRA, R0	XXBET = +7	bet, so zero it and setup
15	E7 1B	E4				BCTR, UN	* NEWGAME	for next bet + roll of dice
16								
17	E9 20				GAMEOVR	EORZ	R0	GAME'S OVER, so zero re-
18	A C8	E9				STRR, R0	* CHIPS	sidue chips, bet, and bet
19	C C8	EC				STRR, R0	* BET	value in XX BET display.
20	EE C8	F5				STRR, R0	* XXBET = +7	(message table). Now set.
21	FO 04	FF				LODI, R0	FF	prelim chip entry cur
22	2 D4	07				WRTE, R0	PORT7	and display in lights!
23	4 75	05				CPSL	CARRY	Now turn off carry bit.
24	00F6	1F	01	0D		BCTR, UN	CHIPSET	(if set), and exit to
25								set new CHIPS value.
26								
27								

DIRECT RELATIVE ADDRESSING - SECOND BYTE																
+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
-	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
-	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
+	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
-	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
+	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
-	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
+	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
-	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
+	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
-	B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
+	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
-	D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
+	E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
-	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

INDIRECT RELATIVE ADDRESS: Add H'60' to displacement

2650 PROGRAMMING FORM

ROUTINE CRAPGAME START ADDR _____

DESCRIPTION _____

ROUTINE SHEET 6 OF 6 MAIN PROGRAM

MEMORY LOCATIONS THIS SHEET _____

ADDRS	DATA B0 B1 B2	LABEL	SYMBOLIC INSTRUCTION OPCODE OPERANDS	COMMENT
1 00F9	0C 00 0F	CALCLOS	L0DA, R0 CHIPS	He lost, so subtract bet from
2 C	AC 00 0E		SUBA, R0 BET	chips total. Did he lose
3 FE	E4 FB		COMI, R0 H'F8'	more than he had avail?
4 101	9A 66		BCFR, LT GAMEOVER	YES! EXIT to terminate this
5 3	1B 5B		BCTR, UN WINDUP	game. No! Wind up and get
6				set to place next bet
7 105	12 17 0B	CHIPSXX	RES B	CHIPS MESSAGE:
8 8	12 1B 17			"U BUY = "
9 8	16 17			
10 10D	05 01	CHIPSET	L0DI, R1 <CHIPSXX-1	Put chips message into dis-
11 F	06 04		L0DI, R2 >CHIPSXX-1	play buffer, then set
12 111	BB FE		ZBSR * MOV	GNP parameters for 2
13 3	04 01		L0DI, R0 1	digit entry of chips, which
14 5	BB FC		ZBSR * GNP	= value of buy in to game
15 7	CC 00 0F		STRA, R0 CHIPS	Player can change until fun-
16 11A	1F 00 10		BCTA, UN NEWGAME	ction key depressed. Store
17 D	00	* FILLER	DATA	selected value in CHIPS &
18 011E	XX XX XX	* DECODE	ROLL FIX TO EXPAND - DOCUMENTED IN DCDROL SUBR	exit to place a bet.
19 to 0125	XX XX XX			
20				
21 012A	44 0F	CORRSUB	ANDI, R0 0F	This subroutine inserted
22 C	CC 00 0E		STRA, R0 BET	to correct program error
23 12F	CC 17 BF		STRA, R0 XXBET+7	Mask off residual M.S.
24 132	BB 82		ZBSR * RLDICE	Nibl, then store away
25 4	07 7F	PASS1	L0DI, R3 7F	the roll of dice. After
26 0136	17		RETC, UN	the roll, set up timed
27				"PASS1" = "parameter &
				return to main progr.

DIRECT RELATIVE ADDRESSING - SECOND BYTE															
+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E
-	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
+	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C
-	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B
+	3C	3D	3E	3F	40	41	42	43	44	45	46	47	48	49	4A
-	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59
+	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67	68
-	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74	75	76	77
+	78	79	7A	7B	7C	7D	7E	7F	80	81	82	83	84	85	86
-	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95
+	96	97	98	99	9A	9B	9C	9D	9E	9F	00	01	02	03	04
-	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
+	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22
-	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31
+	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	40
-	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
+	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E
-	5F	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D
+	6E	6F	70	71	72	73	74	75	76	77	78	79	7A	7B	7C
-	7D	7E	7F	80	81	82	83	84	85	86	87	88	89	8A	8B
+	8C	8D	8E	8F	90	91	92	93	94	95	96	97	98	99	9A
-	9B	9C	9D	9E	9F	00	01	02	03	04	05	06	07	08	09
+	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
-	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
+	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36
-	37	38	39	3A	3B	3C	3D	3E	3F	40	41	42	43	44	45
+	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54
-	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63
+	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72
-	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	80	81
+	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90
-	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E
-	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
+	1E	1F	20	21	22	23	24	25	26	27	28	29	2A	2B	2C
-	2D	2E	2F	30	31	32	33	34	35	36	37	38	39	3A	3B
+	3C	3D	3E	3F	40	41	42	43	44	45	46	47	48	49	4A
-	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59
+	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67	68
-	69	6A	6B	6C	6D	6E	6F	70	71	72	73	74	75	76	77
+	78	79	7A	7B	7C	7D	7E	7F	80	81	82	83	84	85	86
-	87	88	89	8A	8B	8C	8D	8E	8F	90	91	92	93	94	95
+	96	97	98	99	9A	9B	9C	9D	9E	9F	00	01	02	03	04
-	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
+	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22
-	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	30	31
+	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	40
-	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
+	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E
-	5F	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D
+	6E	6F	70	71	72	73	74	75	76	77	78	79	7A	7B	7C
-	7D	7E	7F	80	81	82	83	84	85	86	87	88	89	8A	8B
+	8C	8D	8E	8F	90	91	92	93	94	95	96	97	98	99	9A
-	9B	9C	9D	9E	9F	00	01	02	03	04	05	06	07	08	09
+	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18
-	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
+	28	29	2A	2B	2C	2D	2E	2F	30	31	32	33	34	35	36
-	37	38	39	3A	3B	3C	3D	3E	3F	40	41	42	43	44	45
+	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54
-	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63
+	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	70	71	72
-	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	80	81
+	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F	90
-	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F

INDIRECT RELATIVE ADDRESS: Add H'60' TO DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE DISMES START ADDR 13A

DESCRIPTION CONTROLS ACCESS &
display time of 8 messages lo-
cated in SMI memory (locations
H'1780 - H'17BF'

ROUTINE SHEET 1 OF 2. MP = CRAPGAM
 MEMORY LOCATIONS THIS SHEET

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	0137	XX			>MESLOC-1	RES	1	
2	8	XX			DSPLDLY	RES	1	
3	9	XX			RUNDSPX	RES	1	
4								
5	013A	CA	7B		DISMES	STRR, R2	>MESLOC-1	Store current index to mes-
6	C	05	02			LODI, R1	2	sage table. Now, setup
7	13E	C9	79		REPEATM	STRR, R1	RUNDSPX	running display timeout
8	140	CB	76		DSPAGN	STRR, R3	DSPLDLY	const & store. Also store
9	2	05	17			LODI, R1	<MESLOC-1	static/dynamic displ.par-
10	4	0A	71			LODR, R2	>MESLOC-1	ameter. Now set index to
11	L	BB	E6			ZBSR	*USRDSP	specified msg & display.
12	8	0B	6E			LODR, R3	DSPLDLY	(If static, player depresses
13	A	E7	00			COMI, R3	0	key to exit) Fetch displ.Xt.
14	C	14				RETC, EQ		if static, exit. If dynamic,
15	D	E7	01			COMI, R3	1	is it a 1 yet? Yes, exit to
16	14F	18	02			BCTR, EQ	CONTDSP	continue display. No, re-
17	151	FB	6D			BDRR, R3	DSPAGN	peat single pass display.
18	3	09	64		CONTDSP	LODR, R1	RUNDSPX	To continue running disp-
19	5	F9	01			BDRR, R1	\$+3	lay, get its constant and
20	7	17				RETC, UN		decrement. If zero, exit
21	8	07	7F			LODI, R3	7F	to calling routine. If not,
22	A	1B	62			BCTR, UN	REPEATM	set dynamic parameter
23								again & repeat.
24								last addr = '15B.
25								next routine = DCDROL
26								
27								

DIRECT RELATIVE ADDRESSING - SECOND BYTE																
+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
H	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71	
+	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
H	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	70	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62	61
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
H	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
-	60	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52	51
+	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
H	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
-	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42	41

INDIRECT RELATIVE ADDRESS: Add H'60' to DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE DISMES START ADDR _____

DESCRIPTION MESSAGE TABLE

(8 8-byte messages in SMI memory
locations H'1780-17BF')

ROUTINE SHEET 2 OF 2 MP = CRATGAME

MEMORY LOCATIONS THIS SHEET _____

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1								
2	1780	10	11	0A	PLACEB	RES	8	"PLACE BET"
3		0C	0E	0B				
4		0E	87					
5	1788	17	17	17	XXBET	RES	8	"XX BET Y"
6		0B	0E	07				XX = CHIPS ON HAND
7		17	17					Y = CURRENT BET
8	1790	10	0A	05	PASS1	RES	8	"PASS 1 = "
9		05	17	01				
10		17	16					
11	1798	XX	19	19	XXFOR	RES	8	X--X = YY
12		XX	17	16				X+X = INDIV. DICE
13		YY	YY					YY = added dice
14	17A0	1B	00	12	YOU BEAT	RES	8	YOU BEAT
15		17	0B	0E				
16		0A	07					
17	17A8	07	14	0E	THEHOUSE	RES	8	THE HOUSE
18		14	00	12				
19		05	0E					
20	17B0	12	17	10	UPAYUP	RES	8	U PAY X
21		0A	1B	17				X = BET HE LOST
22		17	XX					
23	17B8	05	14	00	SHOOT	RES	8	SHOOT XX
24		00	07	17				XX = the point player
25		17	17					must make to win
26								on second or greater
27								pass.

DIRECT RELATIVE ADDRESSING - SECOND BYTE

+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
-	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
-	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
+	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
-	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
+	60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
-	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F

INDIRECT RELATIVE ADDRESS: Add H'80' TO DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE DCDROL START ADDRS 15CDESCRIPTION decodes current roll
of dice for win, lose, or throw
againROUTINE SHEET 1 OF 1 MP = CRAPGAME
MEMORY LOCATIONS THIS SHEET _____

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	181	XX					*SAVPAS1 AT LOCATION H'181'	
2	182	XX					*DICERSM AT LOCATION H'182'	
3								
4	015C	08	24		DCDROL	LODR, RO	DICERSM	Fetch dice. Was this roll
5	E	E4	07			COMI, RO	H'07'	of dice a 7? Yes! Exit
6	160	1C	01	1E		BCTA, EQ	SEVEN	to set 7 decode in R1
7	3	E4	11			COMI, RO	H'11'	No! Was it an 11? Yes!
8	5	1C	01	23		BCTA, EQ	ELEVEN	EXIT: Set 11 Decode in R1
9	8	E4	03			COMI, RO	3	No! Was it a crapout?
10	A	99	11			BCFR, GT	CRAPOUT	Yes! go set crapout decode
11	C	E4	12			COMI, RO	H'12'	Was it a 12? Yes! set
12	16E	18	07			BCTR, EQ	PAS112	12 on 1st pass decode. No!
13	170	E8	0F			COMR, RO	SAVPAS1	Did this roll match previous
14	2	18	06			BCTR, EQ	MATCH	pass1? Yes! Set match
15	4	05	10			LODI, R1	H'10'	decode. No! No match.
16	6	17				RETC, UN		set no match and return.
17	7	05	40		PAS112	LODI, R1	H'40'	set Pass1: 12 decode &
18	9	17				RETC, UN		return.
19	A	05	04		MATCH	LODI, R1	4	set Match decode and
20	C	17				RETC, UN		return.
21	D	05	08		CRAPOUT	LODI, R1	8	set crapout decode and
22	017F	17				RETC, UN		return. [Needed more
23								space so relocated below]
24	011E	05	02		SEVEN	LODI, R1	2	set 7 decode and
25	120	17				RETC, UN		return.
26	0123	05	20		ELEVEN	LODI, R1	H'20'	set 11 decode and
27	125	17				RETC, UN		return.

DIRECT RELATIVE ADDRESSING - SECOND BYTE																
+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
N	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71	70
+	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
N	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	70	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62	61
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
N	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
-	60	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52	51
+	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
N	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
-	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42	41

INDIRECT RELATIVE ADDRESS: Add H'80' TO DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE ASMROL START ADDR 0183

DESCRIPTION This routine adds the
roll of individual dice prior to
decode of the roll

ROUTINE SHEET 1 OF 1 MP = CRAPGAME
 MEMORY LOCATIONS THIS SHEET _____

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1								
2	0181	XX			SAVPR51	RES	1	Saved 1 st roll of dice
3	182	XX			DICEASM	RES	1	Value of dice added
4								from current roll.
5								
6	0183	08	22		ASMROL	LDDR, R0	ROLL+5	Get current roll of dice when
7	5	09	22			LDDR, R1	ROLL+7	stopped. Store in XXFOR
8	7	CC	17	9B		STRA, R0	XXFOR	message. Convert from
9	A	CD	17	9B		STRA, R1	XXFOR+3	hex to decimal notation:
10	D	84	66			ADDI, R0	H'66'	1 st die (R0) + H'66' +
11	18F	81				ADDZ	R1	2 nd die → R0
12	190	94				DAR	R0	then do decimal adjust
13	1	CB	6F			STRR, R0	DICEASM	then store in DICEASM.
14	3	BB	F4			ZBSR	*DISLSD	Now break result into 2
15	5	CD	17	9F		STRA, R1	XXFOR+7	nibls and put LS nibl in
16	8	E4	01			COMI, R0	1	XXFOR+7. Was roll ≥ 10?
17	A	18	02			BCTR, EQ	\$+4	Yes! skip next instruction
18	C	04	17			LODI, R0	H'17'	No! get a space code
19	19E	CC	17	9E		STRA, R0	XXFOR+6	store a 1 or space in
20	1A1	17				RETC, UN		XXFOR message, then
21								exit to calling routine.
22					*ROLL	+5	AT LOCATION H'1A7'	last address: H'01A1'
23					*ROLL	+7	AT LOCATION H'1A9'	
24								
25								
26								
27								

+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
-	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
+	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71	
+	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
-	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
-	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
+	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
-	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62	61	
+	70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
-	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
+	90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
-	AF	AE	AD	AC	AB	AA	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0

INDIRECT RELATIVE ADDRESS: Add H'50' to DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE RL DICE START ADDRS 0131

DESCRIPTION This routine provides a rapid timed sequence through all dice combinations to establish roll of dice - Exit when player depresses SEN

ROUTINE SHEET 1 OF 2 MP=CRRPGAME

MEMORY LOCATIONS THIS SHEET

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	01AA	05	DB		GETXT	LODI, R1	H'DB'	GET DICE TABLE CONST AND
2	C	C9	01			STRR, R1	DICEXT	STORE IN DICEXT, and
3	E	17	.			RETC, UN		return
4	01AF	XX			DICEXT			loc for dice table const. and
5	1B0	XX			ROLDLY			loc for roll dice delay.
6								
7	01B1	76	60		RLDICE	PPSU	FLAG, II	Turn on Flag light & intrah.
8	3	09	7A		ROLLING	LDDR, R1	DICEXT	Now get current DICEXT &
9	5	85	01			ADDI, R1	1	increment. Dice table all
10	7	B5	01			TPSL	CARRY	read yet? Yes! reinitialize
11	9	3B	6F			BSTR, EQ	GETXT	index. No! index into next
12	B	0D	61 00			L0DA, RC	DICETB, R1	entry and store updated
13	1BE	C9	6F			STRR, R1	DICEXT	index. Now, show en-
14	1C0	D4	07			WRTE, R0	PORT7	try in Lights, then break
15	2	BB	E4			ZBSR	*DISLED	entry into nibls and put in
16	4	C8	61			STRR, R0	ROLL + 5	"ROLL X X" message.
17	6	C9	61			STRR, R1	ROLL + 7	Now, set the roll display
18	8	07	10			LODI, R3	H'10'	constant and place in
19	A	CB	64		AGAIN	STRR, R3	ROLDLY	"ROLL DELAY". Then, fetch
20	C	05	01			LODI, R1	<ROLL - 1	Address constants to ROLL
21	1CE	06	A1			LODI, R2	>ROLL - 1	message and display the
22	1D0	BB	E6			ZBSR	*USRDSP	current roll. Then, recall
23	2	0B	5C			LDDR, R3	ROLDLY	ROLL display delay and see
24	4	FB	74			BDRR, R3	AGAIN	if finished. If not, do again
25	6	B4	80			TPSU	SENSE	When finished, see if player
26	8	9B	59			BCFR, EQ	ROLLING	stopped the roll. Yes! Exit
27	A	17				RETC, UN		to calling routine. No! Keep rolling dice!

Keep rolling dice!

DIRECT RELATIVE ADDRESSING - SECOND BYTE

+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
N	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71	
+	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
N	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	70	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62	61
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
N	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
-	60	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52	51
+	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
N	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
-	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42	41

INDIRECT RELATIVE ADDRESS: Add H'60' to displacement

2650 PROGRAMMING FORM

ROUTINE RLDICE START ADDR DESCRIPTION ROLL message and Dice Table.ROUTINE SHEET 2 OF 2 MP = CRAPGAMEMEMORY LOCATIONS THIS SHEET

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1								
2								
3	01A2	13	15	01	ROLL	RES	8	"ROLL X X" MESSAGE
4	5	01	17	XX				variables into ROLL+5;
5	01A8	17	XX					ROLL+7 = INDIV. DICE
6								
7								
8	01DB	34	41	13	DICETB	RES	37	DICE TABLE holds all
9	1DE	36	24	51				possible dice combina-
10	1E1	25	66	21				tions + 1 extra 7.
11	4	31	61	16				Access by RLDICE sub-
12	7	12	56	46				routine only.
13	A	45	53	64				
14	1ED	54	23	32				
15	1F0	33	35	65				
16	3	11	14	63				
17	6	42	62	26				
18	9	43	34	55				
19	C	22	44	15				
20	01FF	52						
21								
22								
23								
24								
25								
26								
27								

CRAPGAME MEMORY MAP

USER MEMORY (0-1FF)

0000	
0010	CRAPGAME
0036	NEW GAME
0039	DCDCOR
0075	PASS1
009E	CONROL
00B1	BEATP1
00C1	BOMBOUT
00D0	PAS1XP
00E0	WINNINGS
00E9	WINDUP
00F9	GAMEOVER
010D	CRLCLOS
011E	CHIPSET
012A	DCDROL (EXP)
013A	CORRSUB
015C	DISMES
0183	DCDROL
01B1	ASMROL
01DB	RLDICE
01FF	DICETS

SMI MEMORY - RAM

1780	PLACEBET	8-ENTRY MESSAGE TABLE FROM DISMES ONLY
1788	XXBET	
1790	PASS1	
1798	XXFOR	
17A0	YOUBEAT	
17A8	THEHOUSE	
17B0	UPAYUP	
17B8	SHOOT	
17BF		
17C0	MONITOR SCRATCHPAD (64 BYTES)	
	DISBUF(8 BYTES)	
17FF		

SMI ROM - MONITOR

1800	USE
	USER SYSTEM EXECUTIVE
	INSTRUCTOR FUNCTIONAL CONTROL PROGRAM
1A76	
1B3B	DISLSD
	GNP
1DB6	
1FD5	MOV
1FE6	USRDSP
1FFF	ZBSR VECTORS

TO { DISLSD
GNP
MOV
USRDSP

INSTRUCTOR 50 USAGE MODESINTRODUCTION:

For your reference, each of the "INSTRUCTOR's" usage modes are detailed in short form to complement their description as documented in the "INSTRUCTOR 50 REFERENCE MANUAL." At the conclusion of this section, an illustrated "Typical User Session" with the INSTRUCTOR is provided. Each of the usage modes described in this section are incorporated with emphasis on their logical selection and sequence.

The following usage modes are detailed as follows:

1. MEMORY LOAD - DISPLAY and ALTER MEMORY

FAST PATCH COMMAND

2. REGISTERS - DISPLAY and ALTER REGISTERS

DISPLAY and ALTER PROGRAM STATUS WORD

ENTRY TO:

- o Adjust Cassette
- o Display & Alter Program Counter
- o Memory Fast Patch

3. DISPLAY and ALTER PROGRAM COUNTER

4. WRITE CASSETTE COMMAND

5. READ CASSETTE COMMAND

6. ADJUST CASSETTE (VOLUME CONTROL) COMMAND

7. BREAKPOINT

8. USER PROGRAM EXECUTION - RST mode

- RUN mode

- STEP mode

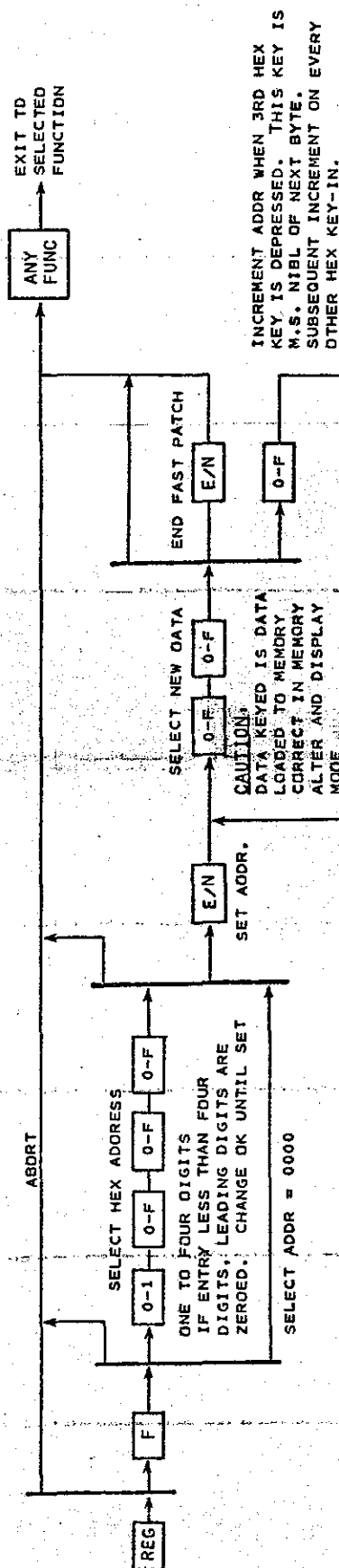
At the conclusion of this section, a typical user session involving entry of a simple program into the INSTRUCTOR's memory and its subsequent debug is provided. At your option, follow the steps in this procedure as indicated.

DIRECTION:

When you have completed your study of this subsection, go on to Module II. Further directions are contained on the 1st page of that module.

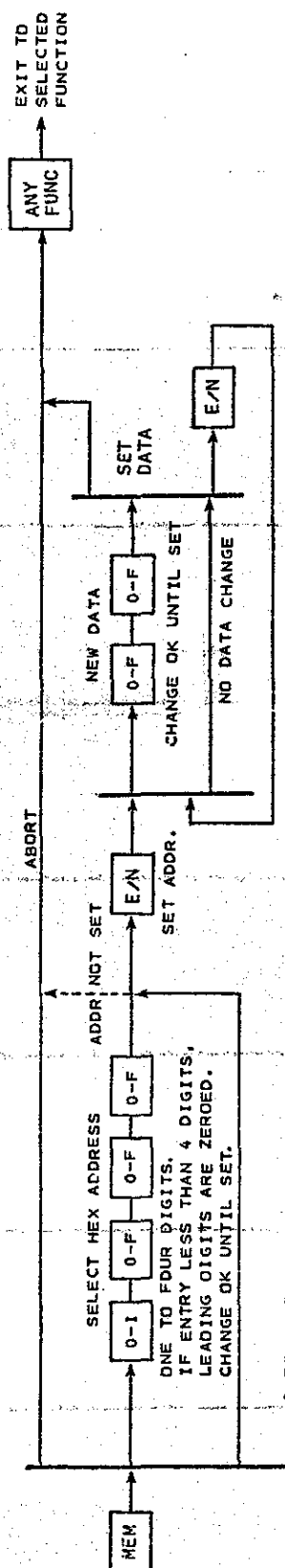
FAST PATCH COMMAND

- 0 THIS PROCEDURE IS USED TO ENTER LONG STRINGS OF CONTIGUOUS DATA INTO MEMORY QUICKLY.
 0 USER SHOULD VERIFY PATCHED DATA AND CORRECT IF NECESSARY IN MEMORY ALTER & DISPLAY MODE.
 0 ERROR = 3 IF DATA ENTERED BUT NOT STORED.

DISPLAY AND ALTER MEMORY

- 0 THIS PROCEDURE IS USED TO:
 0 ENTER SHORT STRINGS OF DATA INTO USER MEMORY.
 0 ENTER OR CHANGE NON-CONTIGUOUS BYTES IN USER RAM.
 0 VERIFY AND INSPECT USER MEMORY.
 0 CORRECT DATA ENTRY ERRORS MADE DURING FAST PATCH.

0 ERROR = 3: DISPLAYED IF
 DATA ENTERED
 BUT NOT STORED.

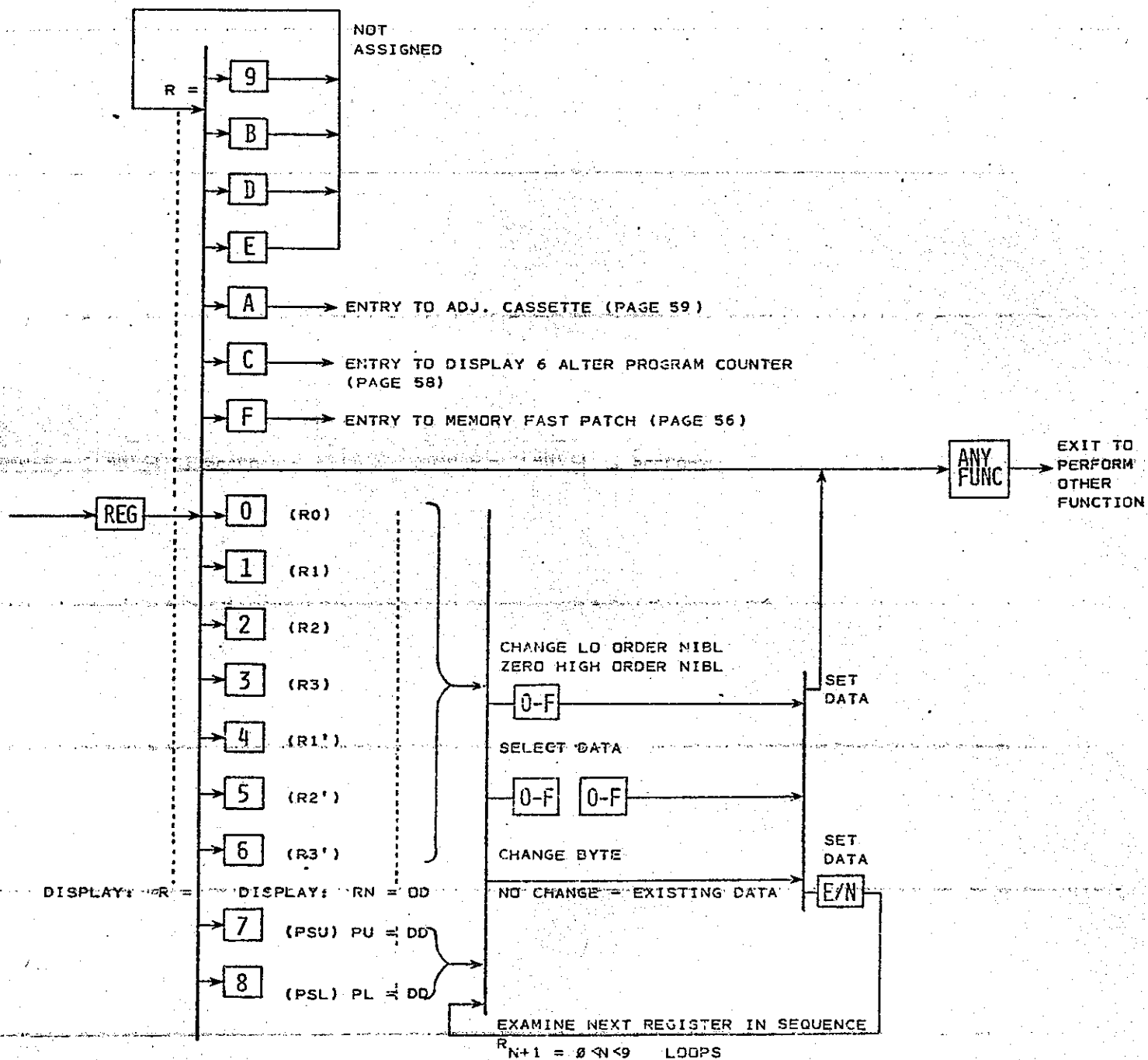


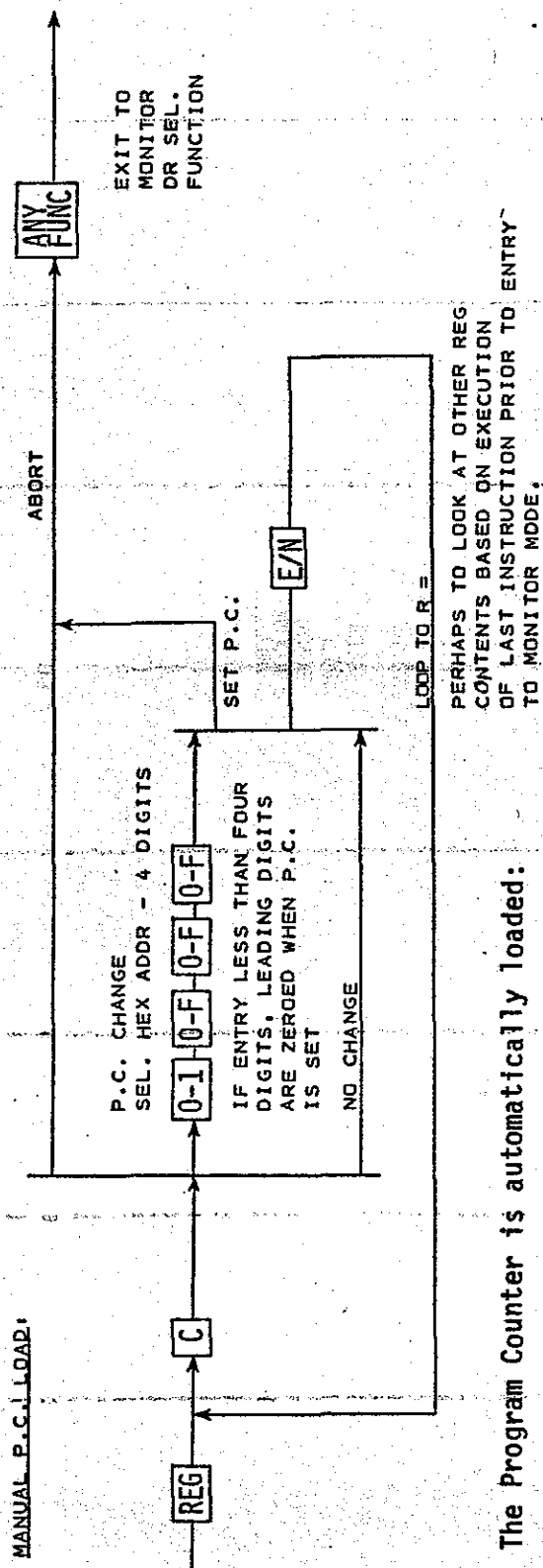
- 0 ZERO IF INITIAL POWER UP.
 0 END ADDR WHEN LAST EXIT FROM
 MEM ROUTINE TOOK PLACE.
 0 SAD OF PROGRAM IF LOADED
 FROM CASSETTE

NOTE: USER MEMORY AVAILABLE ADDRESSES:

- 0 '0000'-'01FF' 512 BYTES FOR CONTROLLING PROGRAMS
 0 '1780'-'17BF' 64 BYTES FOR DATA TABLES
 0 '17C0'-'17FF' 64 BYTES FOR DATA TABLES
 0 DO NOT ATTEMPT TO CHANGE OR STORE DATA
 IN MONITOR RAM (ADDRESSES '17C0-17FF)
 0 SCRATCH PAD, ETC.

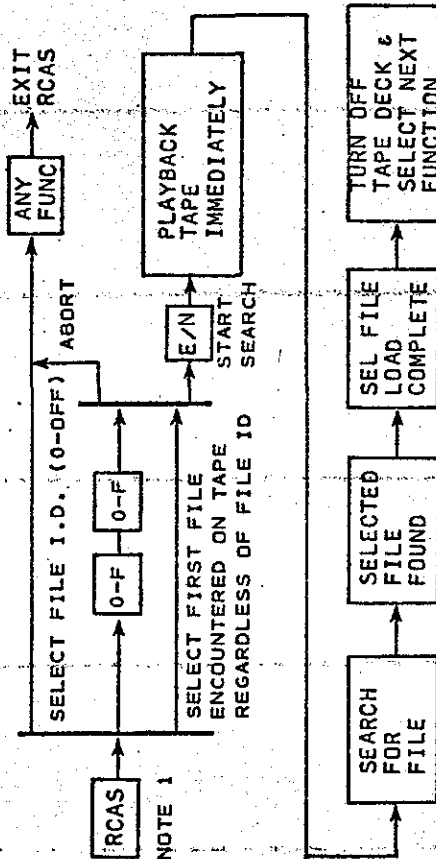
DISPLAY AND ALTER REGISTERS



DISPLAY AND ALTER PROGRAM COUNTER

The Program Counter is automatically loaded:

1. with the address of the next instruction's 1st byte in the User program when exit takes place to another function from RUN mode.
2. If in MONITOR mode, will point to some address in Keyboard Scan routine, but not necessarily at the OPCODE byte, because MONITOR key activity is asynchronous with other functions.
3. to '0000' during initial power-up.

READ CASSETTE COMMANDPROCEDURE

I/O LEDS FLASH ALTERNATELY WITH 5 SECOND OFF PERIODS WHILE SEARCH TAKES PLACE AND SELECTED FILE IS LOADED TO MEMORY. NON-SELECTED FILES ARE TRANSPARENT TO MEMORY. NO HEX DISPLAY.

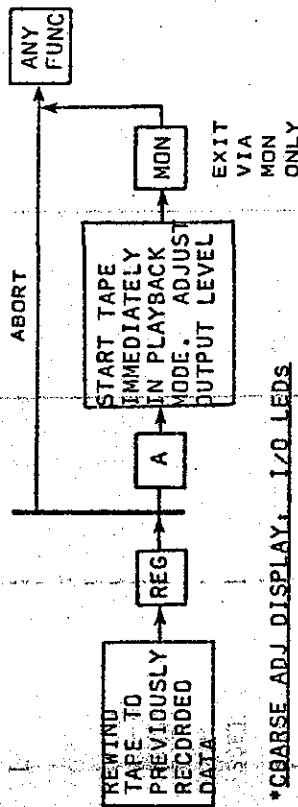
NOTE 1

PRIOR TO INITIATING RCAS, ASSURE THAT TAPE IS REWOUND SO THAT DESIRED FILE CAN BE DETECTED. IF NECESSARY, PERFORM THE PROCEDURE TO ADJUST THE CASSETTE DECK'S PLAYBACK VOLUME.

ERROR MESSAGES.

- ERROR 4: CASSETTE ECC ERROR
PROBABLE CAUSE WAS INCORRECT CODE GENERATED DURING PLAYBACK. REWIND AND REPEAT RCAS SEQUENCE.
- 5: READ CASSETTE MEMORY WRITE ERROR
CHECK KNOWN LAD AND UAD MEMORY CHIP MAY BE DEFECTIVE REPEAT RCAS.
CAUTION. READBACK TO NON-EXISTENT ADDRESSES IN MEMORY IS NOT DETECTED AS AN ERROR IN RCAS MODE.

- 6: CHAR FROM TAPE NOT ASCII HEX
CAN HAPPEN IF GLITCH OCCURS DURING DATA SERIALIZATION IN WRITE MODE OR DESERIALIZATION IN READ MODE. WRITE TO TAPE OR READ FROM TAPE TO ISOLATE. OCCURS IF TAPE TRANSPORT MOTION WAS ERRATIC DURING WCAS OR RCAS MODES.

ADJUST CASSETTE COMMANDPROCEDURE*COARSE ADJ DISPLAY, I/O LEDS

- ALL LEDS OFF PROPER OPERATION OR NO DATA
- SOME NEGATIVE NUMBER (LED BIT 7 ON) NOT ENOUGH PULSES/BIT DETECTION THRESHOLD TOO HIGH
- SOME POSITIVE NUMBER (LED BIT 7 OFF) TAPE 'NOISE' BEING DETECTED

*FINE ADJUST DISPLAY, HEX DISPLAY L.S.D.

PROPER OPERATION - (MINUS SIGN)
DECREASE VOLUME D.
INCREASE VOLUME U.

HINT. DURING ADJUSTMENT, 'SWING' CONTROL TO GET 'U,' THEN 'D,' THEN CENTER CONTROL.

CASSETTE TAPE DECK MAINTENANCE

PERIODICALLY: • INSPECT AND CLEAN HEADS.
• INSPECT AND CLEAN VISIBLE MOVING PARTS OF TRANSPORT. (PINCH ROLLER, CAPSTAN, GUIDES, ETC.).
• INSPECT TRANSPORT WITH TAPE INSTALLED AND RUNNING FOR SMOOTH OPERATION.

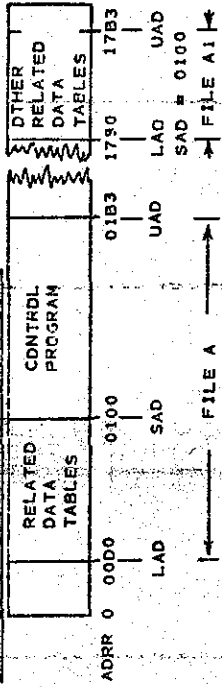
BEFORE RCAS OR WCAS OPERATIONS, VERIFY AS NECESSARY, • CABLING CONNECTIONS.
• SETTING OF VOLUME CONTROL (RCAS).

WRITE CASSETTE COMMAND

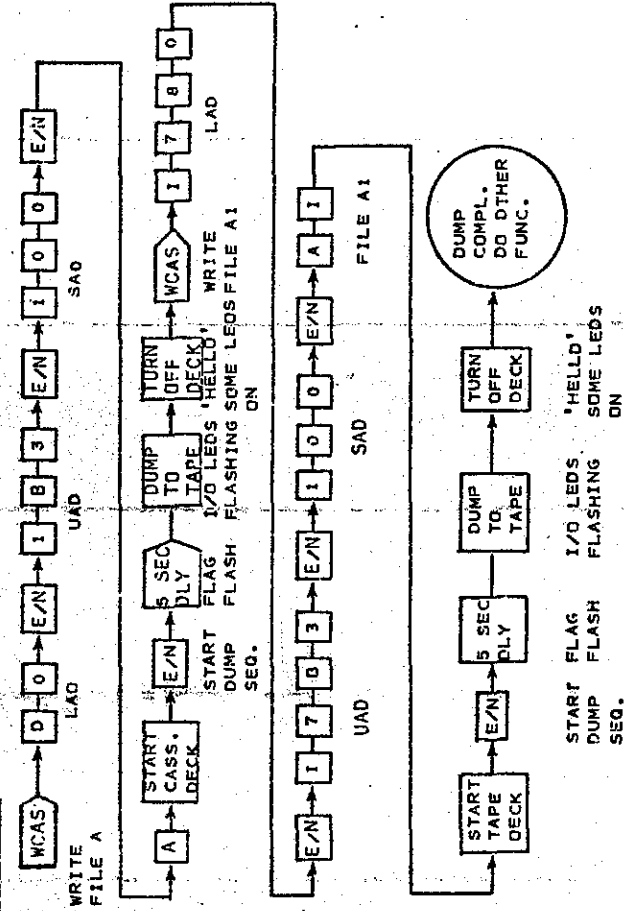
ERROR MESSAGE

ERROR 7 GENERATED IF USER KEYS AN LAD GREATER THAN U.A.D.
NOTE: NO ERROR MESSAGE IF RECORDER NOT ON WHILE DUMP
IS BEING EXECUTED. SEE CAUTION BELOW.

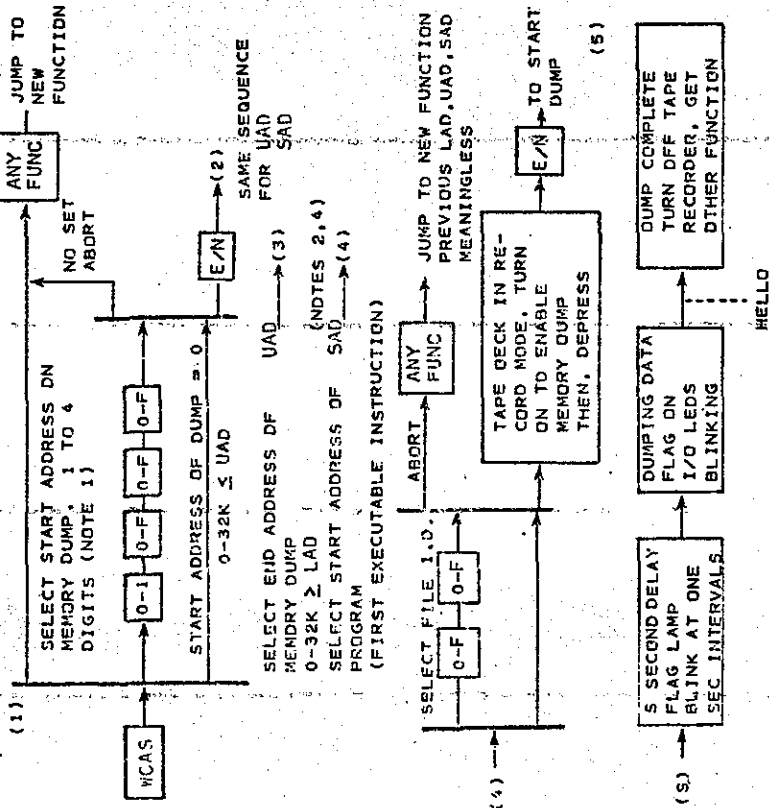
EXAMPLE OF DESIGNATED MEMORY DUMP



PROCEDURE



CAUTION:
ATTEMPT TO WRITE TO TAPE FROM NON-EXISTENT ADDRESSES IN MEMORY DOES NOT GENERATE AN ERROR MESSAGE. NO ERROR GENERATED ON SUBSEQUENT READBACK.



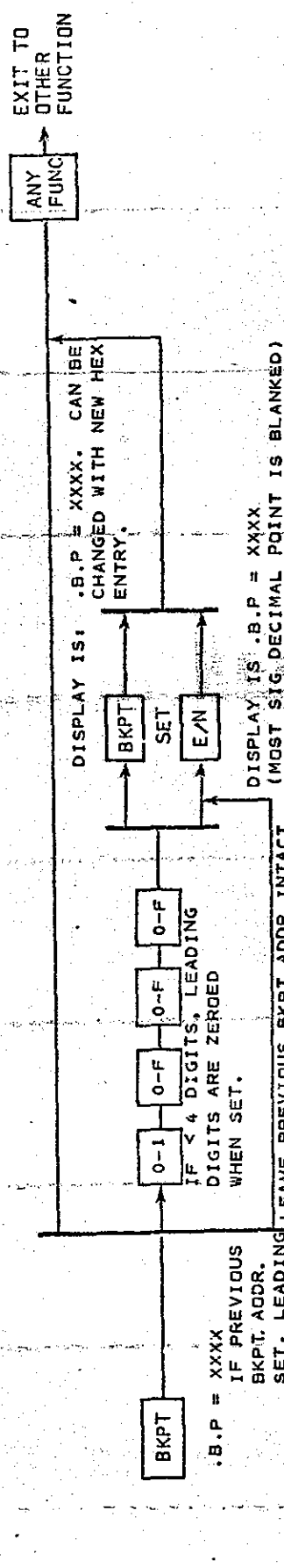
NOTE 1:
IF <4 DIGITS ENTERED, LEADING ADDRS DIGITS ARE ZERO-PADDED.

NOTE 2:
SAD NEED NOT BE SPECIFIED WITHIN LIMITS LAD < NN < UAD

NOTE 3:
IF PROGRAM ELEMENTS ARE CONTAINED BOTH IN USER MEMORY AND IN SM1 RAM (ADDRESSES 1780-17BF), PERFORM 2 SEQUENTIAL DUMPS TO TAPE UNDER SEPARATE FILE I.D.S. KEY PROGRAM START ADDRESSES (SAD) EQUAL IN BOTH FILES, ASSIGNED FILE I.D.S. NEED NOT BE CONTIGUOUS.

NOTE 4:
SAD CAN NOT BE RELOCATED FOR READBACK.

B R E A K P O I N T



ERROR MESSAGE:

ERROR 1 BKPT CAN NOT BE SET/CLEARED IF BKPT ADDRESS IS IN MONITOR OR TO NON-EXISTANT RAM.

ERROR 2 INVALID COMMAND IF ATTEMPT TO ENTER NEW BKPT ADDR AFTER DEPRESSING E/N. CALL BKPT FUNCTION AGAIN.

NOTE 1: SET P.C. SEQUENCE MUST BE COMPLETED UNLESS DESIRED PC KNOWN DUE TO:

1. ARRIVING AT PREVIOUS BREAKPOINT IN RUN MODE.
2. LOADING A FILE FROM CASSETTE (SAD).
3. ARRIVING AT KNOWN P.C. IN SINGLE STEP MODE.
IF USER PROGRAM IS EXECUTED BY DEPRESSING RESEL,
MONITOR CLEARS BKPT FLAG. PROGRAM EXECUTES FROM
ADDRESS 0 WITHOUT BKPT CONTROLS.
4. MONITOR NEVER TESTS FOR BREAKPOINT IN SINGLE STEP
MODE, THUS BREAKPOINT IS IGNORED.

NOTE 2:

NOTE 3: MONITOR NEVER TESTS FOR BREAKPOINT IN SINGLE STEP MODE, THUS BREAKPOINT IS IGNORED.

NOTE 4: INTERRUPT RECOGNITION IS INHIBITED ONCE STEP 3 OR BKPT SEQUENCE IS REACHED.

CONTINUATION FROM BKPT IN STEP MODE

1. MONITOR EXECUTES USER INSTRUCTION AT LOCATION POINTED TO BY P.C. (PREVIOUSLY UPDATED IN BREAKPOINT MODE).
 2. REGISTER AND PSW CONTENTS RESULTING FROM THAT INSTRUCTION ARE SAVED IN MONITOR RAM.
 3. MONITOR FETCHES FIRST BYTE IN NEXT INSTRUCTION AND UPDATES P.C.
 4. MONITOR DISPLAYS UPDATED P.C. AND FETCHED FIRST BYTE.
- CAUTION:** ABOVE SEQUENCE CAN APPEAR TO 'SKIP' THE INSTRUCTION IDENTIFIED IN STEP 1 UNLESS USER RECOGNIZES THAT,
- A) P.C. UPDATED IN BREAKPOINT SEQUENCE (STEP 7) IS NOT DISPLAYED UNLESS P.C. IS EXAMINED, AND
 - G) INSTRUCTION ASSOCIATED WITH THAT P.C. IS EXECUTED WHEN STEP IS DEPRECATED, THEN NEW P.C. AND FETCHED 1ST BYTE ARE DISPLAYED.

USEFUL HINT: IF BKPT SET TO BIR-OR BDR-INSTRUCTION, ONLY 1 PASS IS MADE BEFORE USER PROGRAM EXECUTION STOPS. CHANGE REGISTER CONTENTS TO PERMIT FALL-THROUGH TO NEXT INSTRUCTION IN STEP MODE.

THE FOLLOWING SEQUENCE IS EXECUTED BY THE MONITOR WHEN RUN IS DEPPRESSED AFTER A VALID BREAKPOINT IS SET.

BREAKPOINT SEQUENCE

1. SAVE USER BYTE AT BREAKPOINT ADDRESS IN RAM AND SETS BKPT FLAG.
2. JAMS WRTC (H'BQ') INSTRUCTION INTO USER PROGRAM AT BREAKPOINT ADDR.
3. PROGRAM EXECUTES FROM START ADDRESS UNTIL HARDWARE DETECTS JAMMED WRTC INSTRUCTION.
4. CONTROL VECTORS BACK TO MONITOR.
5. MONITOR VERIFIES BKPT VALIDITY BY WRTC INSTRUCTION AND ADDRESS COMPARES.
6. MONITOR RESTORES SAVED BYTE (STEP 1) TO USER PROGRAM AT BKPT ADDRESS, THEN EXECUTES THAT INSTRUCTION (HIDDEN SINGLE STEP).
7. MONITOR UPDATES P.C. TO ADDRESS OF NEXT INSTRUCTION FETCH.
8. MONITOR DISPLAYS ADDRESS AND FIRST BYTE OF THE BREAKPOINT INSTRUCTION WHICH IT JUST EXECUTED.
9. TO REPEAT STEPS 1-8, ENSURE THAT P.C. IS SET TO START ADDRESS OF PROGRAM WHICH WILL EXECUTE TO DEFINED BKPT. ADDR.

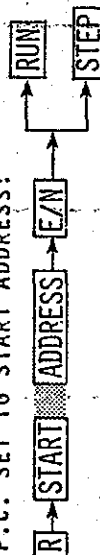
NOTE: IF INSTRUCTION AT BKPT LOCATION IS A HALT (H'40'), PC IS NOT UPDATED. PC = BKPT ADDR USER MUST DECIDE WHETHER TO CONTINUE (BY REPLACING HALT WITH OTHER INSTRUCTION) OR TO STOP. IF REPEAT NECESSARY, SET PC = START ADDRESS AND DEPRESS RUN.

USER PROGRAM EXECUTION

CONDITIONS:

1. Requires P.C. set to start address unless P.C. is known due to:
 - a. Arriving at previous BKPT in RUN mode.
 - b. Loading a file from tape cassette.
 - c. Arriving at known P.C. via SINGLE STEP
 - d. Continuation from BREAKPOINT.

2. P.C. SET TO START ADDRESS:



EXECUTION OPTIONS:

- RST**
 - User program executes from location 0000.
 - BKPT, if previously set, is bypassed.
 - Program executes until
 - a. A HALT (H'40') instruction is detected.
 - b. PAUSE signal is raised by USER.
- RUN**
 - User Program executed from P.C. start address.
 - If no BKPT set, program executes and terminates same as for RESET.
 - If BKPT set, program executes from start address to BKPT address, halts, and displays BKPT address and data.
 - *Monitor functional sequence in BKPT is described on page - of this manual.
- STEP**
 - User program executes single instructions from start address.
 - BKPT controls are bypassed.
 - *MONITOR functional sequence in SINGLE STEP mode is described in the next col.
 - Is permitted from BKPT address.
 - RESTRICTION:
 - P.C. is not valid if any of following instructions are encountered during SINGLE STEP of user program:
 - BCTA, UN MONITOR (H'18', IS'00')
 - BSTA, UN MONITOR (H'3B', IS'00')
 - P.C. not valid if MON depressed.
 - P.C. not valid if program entered from keyboard.

MONITOR FUNCTIONAL SEQUENCE IN SINGLE STEP MODE

The SINGLE STEP function is accomplished using a combination of software and hardware-implemented logic. A complete description is contained in the "INSTRUCTOR" reference manual. The following provides in outline form a description of the sequence executed by the monitor when **STEP** is depressed.

1. The SINGLE STEP flag is set.
2. Register contents previously stored upon entry to the monitor are restored to the 2650.
3. The monitor executes a "hidden single step" to determine how many OPREQS are contained in the instruction to be stepped. Test of the BKPT flag is bypassed.
4. The MONITOR permits execution of one user program instruction by counting predetermined number of OPREQS.
5. P.C. is updated to the next instruction.
6. The registers (RO-R3, RI-R-3', and PSW) are saved.
7. Next address (PC) and data are displayed. No MINUS sign (-) precedes the address. SINGLE STEP flag is cleared.
8. The monitor exits to KBD SCAN routine to await user's input.

NOTES:

1. **STEP** will not proceed if HALT (H'40') was last instruction executed. User must:
 - a. Exit to monitor.
 - b. Set P.C. and **STEP** to repeat sequence.
2. RESTRICTION:
 - Attempt to **STEP** into monitor address space (H'1800'-1FFF) is not permitted. ERROR 9 (INVALID COMMAND) is generated.
3. SMI RAM:
 - Locations H'1780'-17C0' (65 Bytes) available to user for data storage (tables, constants, etc.)
 - Locations H'17C1'-17FF' (63 Bytes) RESTRICTED to MONITOR programs.
4. If **STEP** is to execute looping instruction, instruction will execute once each time **STEP** is depressed until condition to exit loop is satisfied. Display does NOT change. Register contents are updated.

USE OF THE "INSTRUCTOR" TO ENTER AND DEBUG A PROGRAM IN A TYPICAL SESSION

DIRECTION:

Complete the following steps, referring as necessary to the program "ALGOLITE" at end of this section.

1. Enter program via MEMORY FAST PATCH:

REG **F** **1** **0** **E/N** XX....XX **E/N**

data entry: ADDR 10 - 19

2. Inspect data in MEMORY mode.

Change if necessary

MEM **1** **0** **E/N** ... **E/N** to address
D019

3. SET PC to start address

REG **C** **1** **0** **E/N** start address = 0010

4. Execute program in RUN mode

RUN - lights H'20'

5. Desired display was changing pattern of light. Single step through program by

a. SET PC: **REG** **C** **1** **0** **E/N**

b. **STEP** **STEP**

c. Inspect registers

REG **0** **E/N** for R1;

E/N for R2, etc.

STEP to execute next
instruction

d. Reason for constant H'20' on LEDs:

$R0 + R1 = 25 + 20 = '20'$ result

6. Change EOR2,R1 to IORZ,R1

MEM **1** **5** **E/N** **6** **1** **E/N**

7. Set PC and execute program

REG **C** **1** **0** **E/N** **RUN** Appears all
lights on and
no random pattern.

8. Execute single passes through program
ALGOLITE.

a. **BP** **1** **8** **E/N** set breakpoint at
address '18'

b. **REG** **C** **1** **0** **E/N** Set PC
c. **RUN** and execute
single pass
and repeat.
- pattern
changes.

..... **RUN**

9. Add a subroutine. Instruction WRTE
at location '16' is replaced with
unconditional branch to subroutine
at location 30.

MEM **1** **6** **E/N** **3** **B** **E/N** **1**
0 **E/N**

10. Enter the subroutine WRTSUB at location
30 via MEMORY FAST PATCH

REG **F** **3** **0** **E/N** XX...XX **EN**

11. Execute single pass through program
ALGOLITE.

a. **BP** **1** **8** **E/N** - set breakpoint
and

b. **REG** **C** **1** **0** **E/N** - set start
address

c. **RUN** ... **RUN** - program doesn't
work.

12. Debug program by
 - a. **REG** **C** **1** **0** **E/N** ... resetting PC to start address.
 - b. **SS** - single step and note jump to address '28' not enough displacement in BCTR instruction at location '16.'
13. Displace 8 more locations
 - a. **MEM** **1** **7** **E/N**
 - b. **1** **8** **E/N** to change from '10' to '18' in byte 17.
14. Set PC and single step. Appears to work.
15. Set breakpoint (H '18') and PC (H '10') and execute single passes - program works.
16. Write program ALGOLITE to tape cassette
Low addr = 0010; Upper addr = 0034
Start addr = 0010
File = H '03' **WCAS** ... etc.
17. Wipe out subroutine WRTSUB VIA MEM FAST PATCH.
REG **F** **3** **0** **E/N** and 00...00 through location '34.'
18. Repeat step 15 - program hangs.
19. Adjust cassette input for proper level.
REG **A** **E/N** ... adjust till MSD of display is - sign.
20. Read File 3 from tape to MEMORY
RCAS **3** **E/N** ... HELLO. And repeat step 15.

MODULE I

CRAPGAME 2650

DIRECT RELATIVE ADDRESSING - SECOND BYTE																
+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
-	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71	
+	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
-	70	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62	61
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
-	60	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52	51
+	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
-	40	3F	3E	3D	3C	3B	3A	39	38	37	36	35	34	33	32	31
+	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
-	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42	41

INDIRECT RELATIVE ADDRESS: Add H'80' TO DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE ALGOLITE START ADDR '10'DESCRIPTION After initializing R0 and R1 with constants, this routine computes: $R1 + R0 \rightarrow R0 \oplus R1 \rightarrow R0$ and transmitsresult to LED's; repeats foreverROUTINE SHEET 1 OF 1

MEMORY LOCATIONS THIS SHEET _____

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1					* SET	I/O SELECTION SWITCH TO "EXTENDED I/O PORT 07"		
	0010	05	05		ALGOLITE	LODI, R1	H'05'	initialize R1 = 05' & R0
3	12	04	20			LODI, R0	H'20'	= '20' then do algorithm:
4	14	81				ADDZ	R1	$R1 + R0 \rightarrow R0 \oplus R1 \rightarrow$
5	15	21				EORZ	R1	$\rightarrow R0$. Then, transmit
6	16	D4	07			WRTE, R0	PORT 7	to parallel I/O LED's &
7	18	1B	7A			BCTR, UN	\$-4	repeat forever
8								

Assume that the instructions at memory locations '1A' through '2F' can not be altered. You wish to implement a subtract function after displaying the result.

Method: Include the function: $R1 - X \rightarrow R1$ (where X is a constant = '3F' as a subroutine.

At location '16':

Substitute for WRTE, R0 PORT 7 instruction.

10	0016	3B	10		BSTR, UN	WRTSUB	WRTSUB at Address '30'
----	------	----	----	--	----------	--------	------------------------

At location '30':

Subroutine "WRTSUB" is loaded into memory.

13	0030	D4	07		WRTSUB	WRTE, R0	PORT 7	transmit to I/O LED's
14		A5	3F			SUBI, R1	H'3F'	then $R1 - H'3F' \rightarrow R1$
15		17				RETC, UN		and return to repeat.



2650 MICROPROCESSOR COURSE

MODULE II - A

2650 MICROPROCESSOR

PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086

REFERENCE:

Outlines and Abstracts
pages 4 to 5.

INTRODUCTION:

This lesson first describes the function of a computer in concept, then moves rapidly into a discussion of 2650 Microprocessor features and characteristics. A complete block diagram analysis of the 2650 is made. As each feature is exposed, your understanding will be reinforced through performance of selected exercise related to the "INSTRUCTOR."

The Microprocessor is like a well stocked toolbox chock-full of finely crafted instruments. In the hands of a skilled craftsman, these instruments (its functional capabilities) can be selected to perform a myriad of tasks. As this Microprocessor course progresses, you'll be introduced to, and master the capabilities of, this 'box of tools.' During this lesson, you'll recognize the validity of a unique approach to designing a Microprocessor into your application. Simply stated, your approach to designing with microprocessors should be one of selecting, with confidence, a composite of microprocessor features and capabilities BEST SUITED for implementation of YOUR application.

DIRECTION:

Select Tape 1-B; "2650 Features and Characteristics" from the Cassette Album, and install it in your tapedeck.

Playback until you hear the first 2-second tone.

FIGURE 1WHAT IS A COMPUTER?

- A COMPUTER IS A DEVICE:
 - CAPABLE OF EXECUTING ANY ALGORITHM
 - GIVEN SUFFICIENT
 - TIME, AND
 - SPACE

FIGURE 2WHAT IS AN ALGORITHM?

- AN ALGORITHM IS:
 - A SEQUENCE OF WELL-DEFINED OPERATIONS FOR SOLVING A SPECIFIC TYPE OF PROBLEM
- AN ALGORITHM HAS ZERO OR MORE INPUTS (VALUES GIVEN TO IT INITIALLY BEFORE IT BEGINS)
- AN ALGORITHM HAS ONE OR MORE OUTPUTS (RESULTS WHICH HAVE A SPECIFIED RELATION TO THE INPUTS)
- AN ALGORITHM TERMINATES (REACHES ITS CONCLUSION) AFTER A FINITE NUMBER OF STEPS

FIGURE 3

EXAMPLE OF AN ALGORITHM

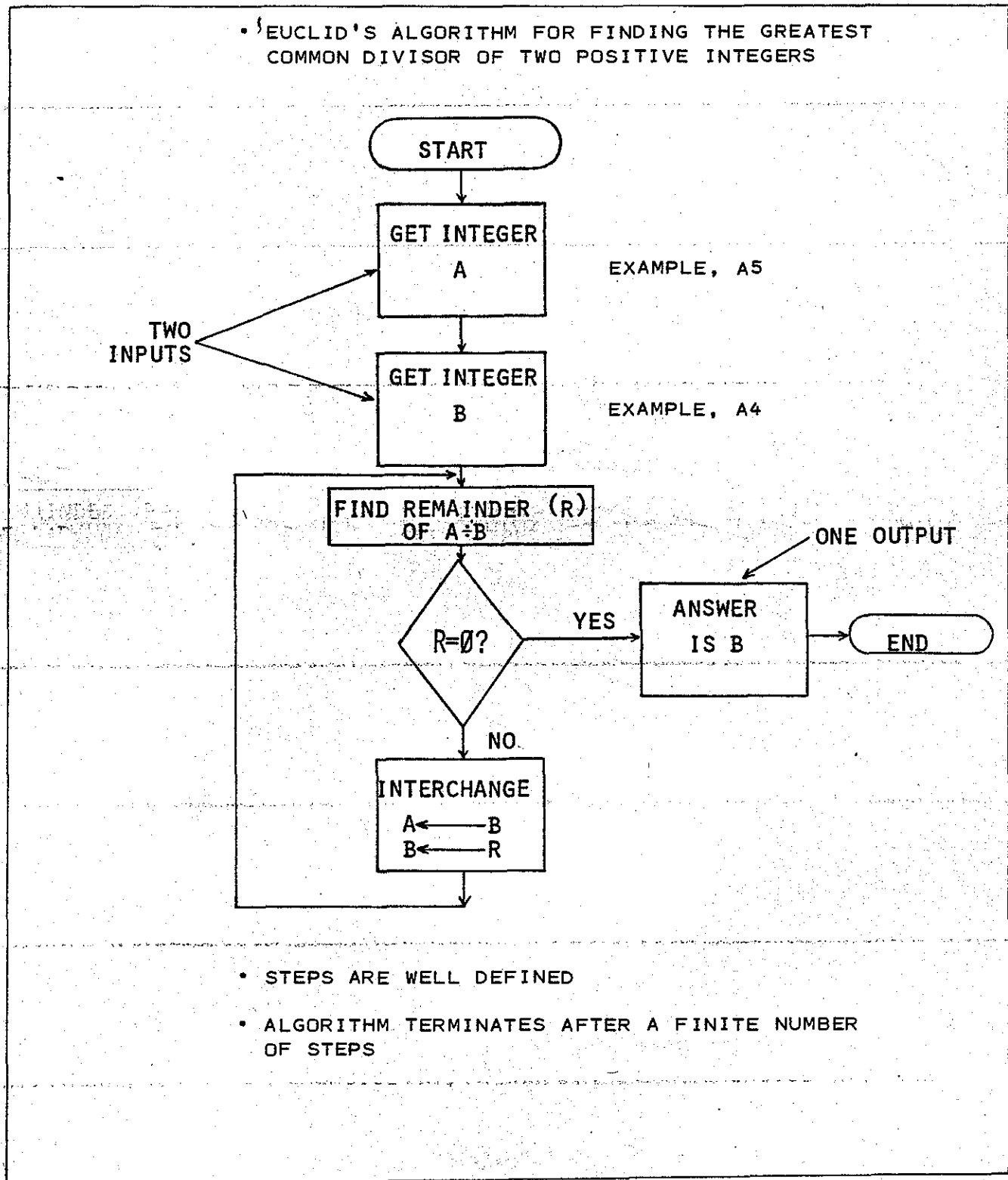
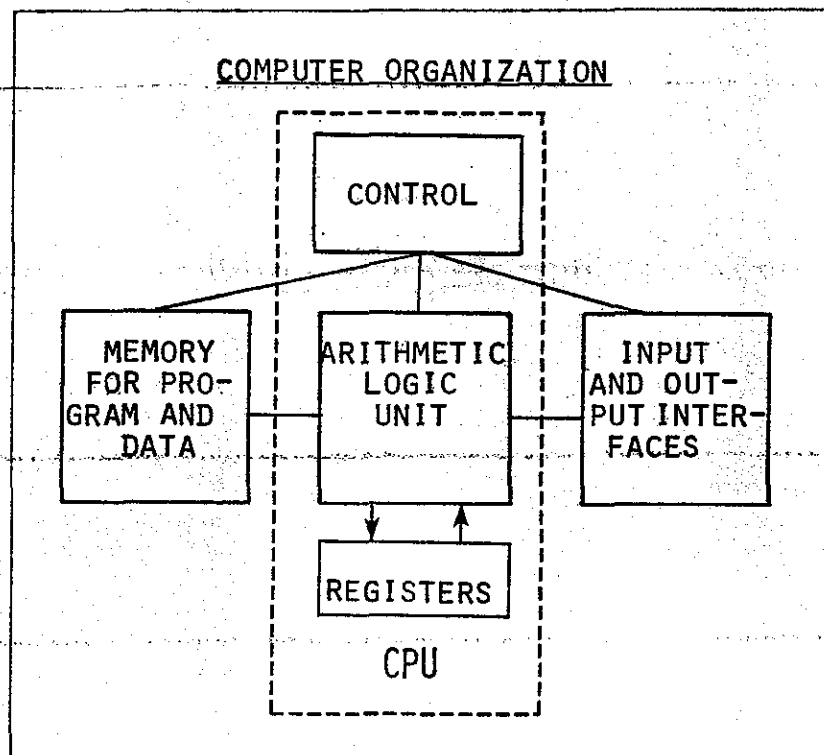


FIGURE 4

WHAT IS A COMPUTER PROGRAM?

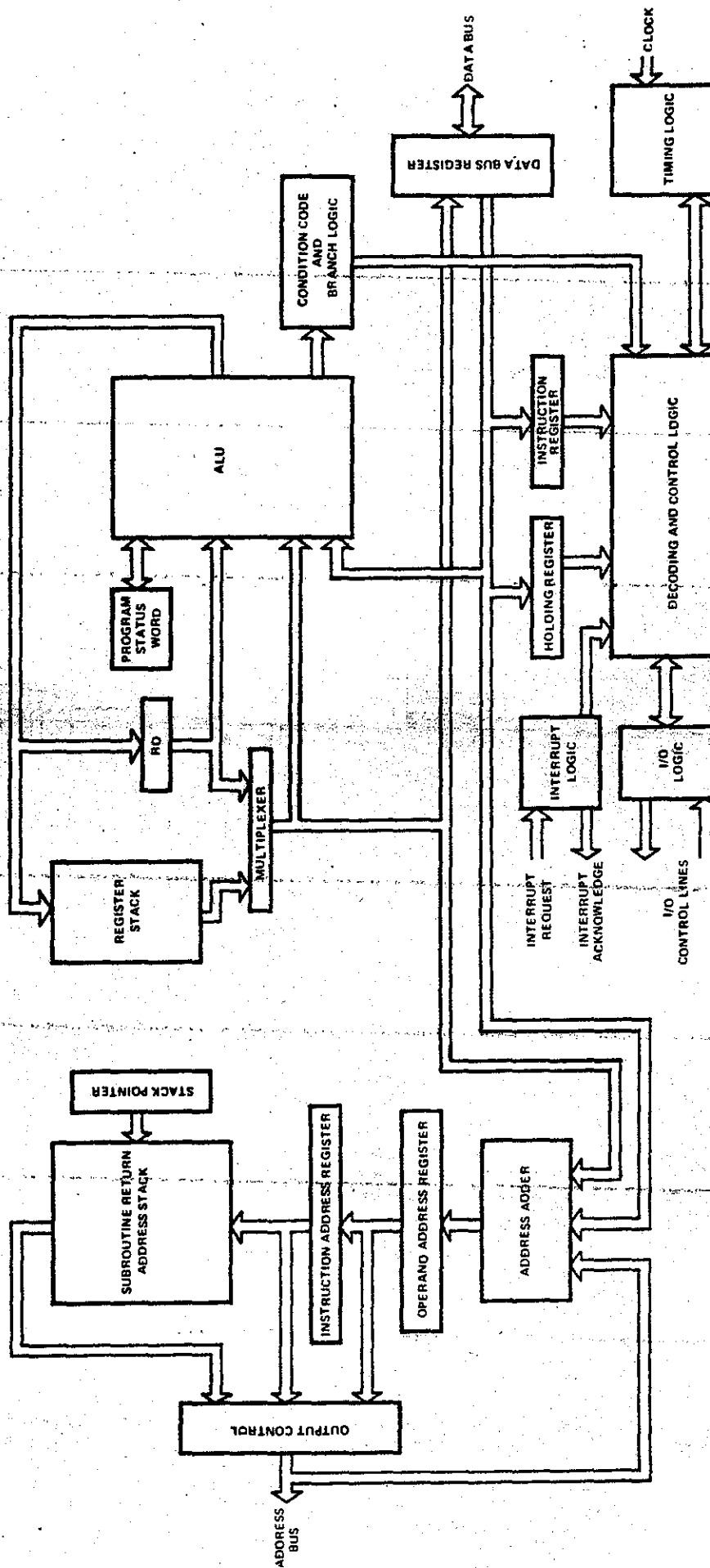
- A COMPUTER PROGRAM IS:
 - THE IMPLEMENTATION OF ONE OR MORE ALGORITHMS USING COMPUTER OPERATIONS AND EXPRESSED IN A COMPUTER LANGUAGE.

FIGURE 5

**DIRECTION:**

WHEN DIRECTED TO DO SO ON TAPE, EXTRACT THIS AND THE FOLLOWING PAGE (2650 MICROPROCESSOR BLOCK DIAGRAM) AND SET THEM SIDE BY SIDE. THEN PERFORM THE REQUIREMENTS OF EXERCISE 1 ON PAGE 6.

FIGURE 6



EXERCISE 1MICROPROCESSOR BLOCK ANALYSIS
PRELIMINARY ACTIVITIES**DIRECTION:**

Complete the following steps as required.

1. If you have not already loaded CRAPGAME into the INSTRUCTOR from your permanent storage file (Tape 2 Side A) do it now. Rewind and store Tape 2 after you have completed the RCAS procedure.
2. Do not execute CRAPGAME at this time.
3. Reinstall Tape 1 (side B up) in place. Do not playback the tape until directed to do so.
4. Read the INTRODUCTION on this page.

INTRODUCTION:

Basically, there are 2 major "compartments" - blocks - involved in definition of the microprocessor. One such block is its SOFTWARE - its set of instructions - its DIRECTIONS. Another such block is its HARDWARE - its internal components which are acted upon and directed to perform some distinct task through execution of one or more of its instructions. A third such block includes EXTERNAL HARDWARE - components outside of the microprocessor, which, nevertheless have specific functions and which receive direction for their activity from the microprocessor. When the third block is considered, you are defining not only the functional capability of the microprocessor, you are defining the functional capability (and activity) of a total computing system.

The process of SELECTING, from the microprocessor's capabilities, those functions which best suit your application involves a 2-part approach. The 2 parts to the approach include (1) introduction and definition, and (2) practical knowledge. In this lesson, we'll introduce and define some of the "compartments," the specialized instruments of the microprocessor, using the 2650 microprocessor block diagram as a model. Later in this course, you'll gain valuable insight as to the use of each "microtool."

DIRECTION:

Go right on to the next page.

EXERCISE 2MICROPROCESSOR BLOCK ANALYSISTIMING LOGIC; DATA BUS ANDDATA BUS REGISTERINTRODUCTION:

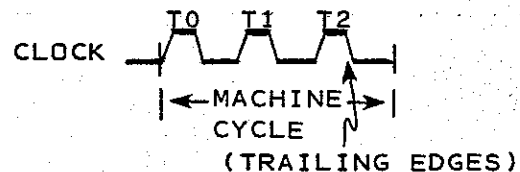
As each diagrammed block is explained, take notes on the key points. The discussion of each functional block is accompanied by a short procedure involving the use of the INSTRUCTOR to reinforce each topic.

The timing logic provides a crucial phased clocking sequence to the internal control circuits of the microprocessor, in order to synchronize the orderly, yet complex transfer of data and data manipulation. It also provides phased timing of all output signals which the microprocessor must raise in order to communicate effectively with attached memory and external devices.

The DATA BUS is the main path for transfer of information between the microprocessor and all other devices, including its memory. Both input and output data transfers are implemented, thus the DATA BUS is said to be BIDIRECTIONAL. During input data transfer, the DATA BUS REGISTER, a group of 8 latches, serves as temporary storage for data until it can be synchronized into a specific location within the processor.

DIRECTION:

Listen to Tape 1B for a short commentary on the TIMING LOGIC, DATA BUS, and DATA BUS REGISTER. Further directions are provided on tape.

NOTES:TIMING LOGICDATA BUSDATA BUS REGISTER

PROCEDURE:TIMING LOGIC

1. Execute the program "CRAPGAME." Enter "CHIPS" and your "BET" in the usual way. Play out a few passes of the dice, then complete the following:

2. OBSERVATIONS:

- Dice roll rate:

It takes about ____ (2)(3)(4)(5) seconds for all combinations of the dice to be displayed when the dice are rolling.

Message Display rate:

About ____ (1)(2)(3)(4) seconds are required to display the messages ...

... X--X=YY

XX BET Y

SHOOT XX, as a group, and to start roll XX
during subsequent roll of the dice
after PASS 1.

As observed in Module 1 exercises, both the roll rate and the message display rate could be modified.

____ (true)(false)

From these observations, we can conclude that execution of the program "CRAPGAME" is performed as a series of TIMED EVENTS.

____ (true)(false)

DIRECTION:

Listen to Tape 1B. At the tone, go on to the next procedure.

PROCEDURE: DATA BUS AND DATA BUS REGISTER

1. Depress MON ... to stop "CRAPGAME" program execution.
2. Access the routine 'RLDICE' documentation from Module 1 (page 51).

3. In routine RLDICE, locate address '1B1' at line 7.

4. Depress REG C 1 to set the Program Counter to address '1B1', the next instruction to be executed.
B 1 E/N.

5. Depress STEP sequentially until display = 01C0 D4.

OBSERVATION: Parallel I/O LED's = 00000000 0 = ON
 0 = OFF

DIRECTIONS: Blackout the zeros in the positions corresponding to LED's which are off.

6. Depress STEP once.

OBSERVATION: Parallel I/O LED's = 00000000

7. Listen to audio tape for short commentary.

NOTES: _____

8. Depress BKPT 1 C 4 to set a BREAKPOINT ADDRESS for later examination of memory.

9. Depress RUN to execute the program up to and including the BREAKPOINT address '1C4.'

10. Depress REG 0 E/N

OBSERVATION: R0 = ____ R1 = ____

11. Depress MEM 1 A to display R0LL+5 at location 1A7.

7 E/N

OBSERVATION: Display '1A7' = ____

12. Depress E/N E/N to increment display to R0LL+7 (address 1A9).

OBSERVATION: Display '1A9' = ____

13. Listen briefly to tape 1B.

CONCLUSION: Due to execution of the instruction at location 1C4, the contents of R0 were moved to memory location '1A7.'

14. Depress REG C to inspect the next address of program execution (Program Counter).

OBSERVATION: Display = PC = _ _ _ _

15. Depress STEP to execute the instruction at the location observed in step 14.

16. Depress MEM 1 A to inspect ROLL+7 in location '1A9.'

9 E/N

OBSERVATION: Display = '01A9' _ _

Compare the contents of location '01A9' (taken in this observation) to the contents of R1 (observed in step 10). Are they equal? _ _ (yes)(no).

From this, can you conclude that the contents of R1 were moved to location '1A9' when the instruction at location '1C6' was executed? _ _ (yes)(no).

17. Listen to tape for verification.

NOTES: _____

DIRECTIONS:

1. Set aside the documentation for routine RLDICE at this time. You'll need it in the next exercise.
2. Go right on to Exercise 3 (next page).

EXERCISE 3 INSTRUCTION REGISTER, INSTRUCTION ADDRESS REGISTER, HOLDING REGISTER, AND DATA BUS REGISTER (PART 2)

INTRODUCTION:

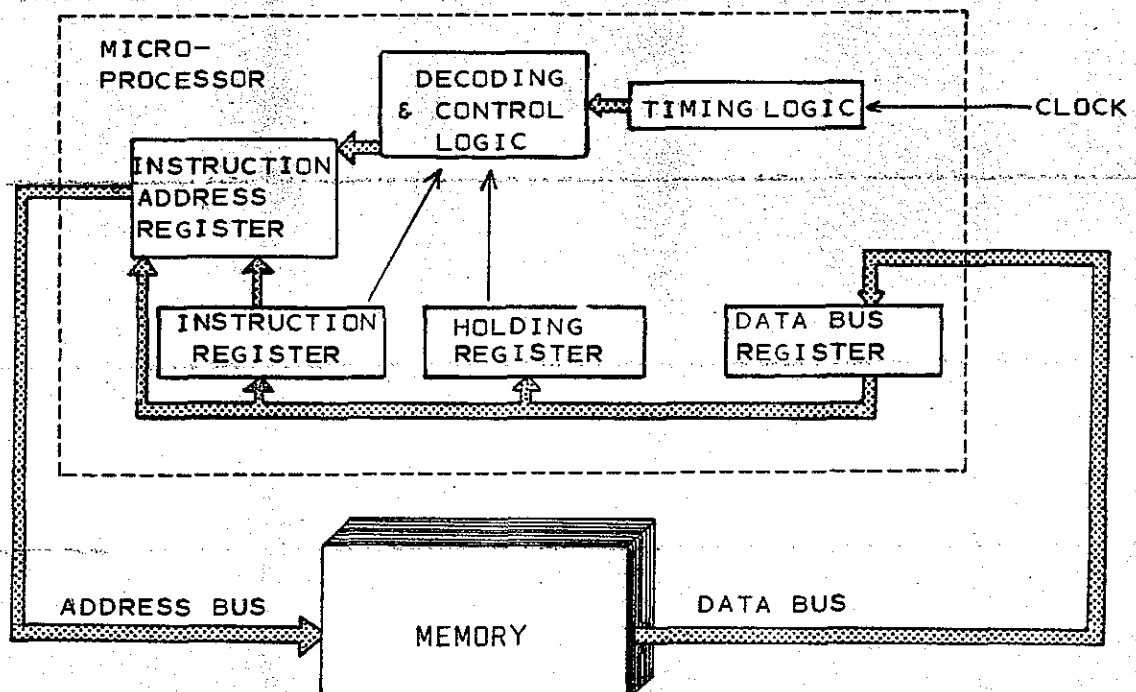
A considerable number of circuits within the microprocessor are dedicated to control of addresses asserted to memory from the microprocessor. At one time, the microprocessor may fetch an instruction of one or more coded bytes from memory. At other times, input data may be fetched for subsequent calculation, then stored later at another address in memory. Address circuits are discussed later in Exercise 10; right now let's take a look at the registers involved in receiving instructions from the microprocessor's memory.

DIRECTION:

Listen to Tape 1B for a brief description of Figure 7. Further directions are provided on tape.

FIGURE 7

INSTRUCTION ADDRESS SCHEME SIMPLIFIED



NOTES:

PROCEDURE:

1. Set a START ADDRESS for program execution at address '1B1.'
2. Access routine RLDICE once again (Module 1 Page 51).
3. Depress STEP After each STEP, and starting with address '1B3' display, compare the data field of the display with the code in "DATA-B0" column of routine RLDICE's listing. Start at line 8.
4. Repeat step 3 five times.
5. OBSERVATION: The Data fields ____ (do)(do not) compare.
6. Listen to tape 1B for a brief commentary, then proceed as directed.

DATA BUS & DATA BUS REGISTER USAGE TO ACCESS RAW DATA IN MEMORYPROCEDURE:

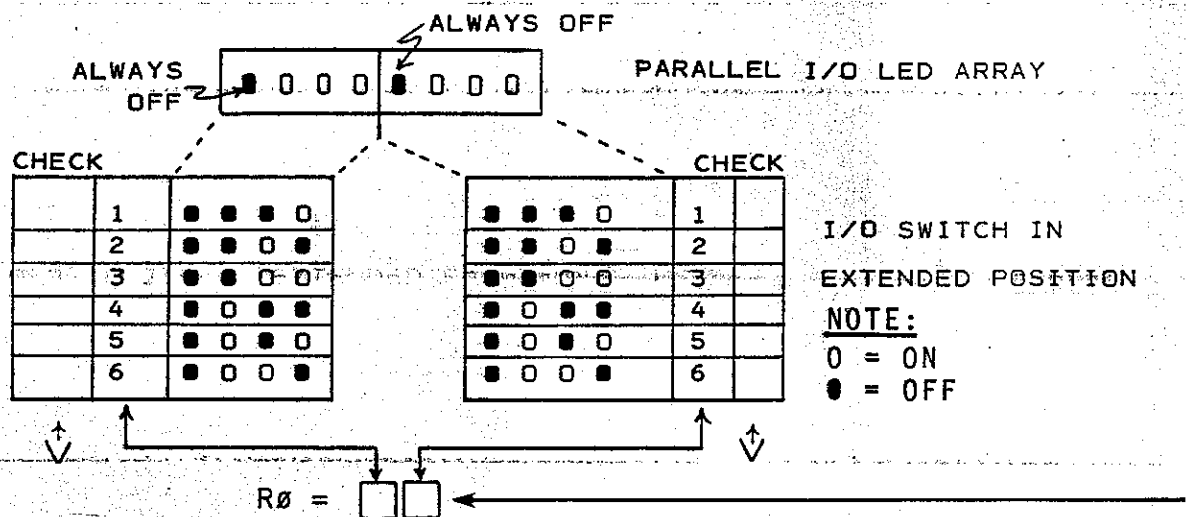
1. Set a START ADDRESS for program execution at location '1B1.'
2. Depress STEP sequentially ... until Display = 01BB 0D
3. Depress REG 1 OBSERVATION: R1 = __
4. Depress REG 0 OBSERVATION: R0 = __
5. Depress MEM 1 X where XX = observed contents of R1 (above)
X E/N OBSERVATION: Display = 01XX __
6. Depress STEP and inspect registers. OBSERVATION: R0 = __
R1 = __
7. Compare R0 contents to those observed in step 5. OBSERVATION: The contents ____ (are) (are not) equal. From this we can conclude that the data in memory at an address indexed by the contents of R1 was moved via the Data Bus into Register R0, replacing the contents observed in step 4.
8. Listen to Tape 3A for verification.

DIRECTION:

Go right on to the next page.

DATA BUS USAGE IN TRANSFER OF DATA TO EXTERNAL DEVICE:PROCEDURE:

1. Depress STEP sequentially ... until display = 01C0 D4
2. Inspect Register R0 OBSERVATION: R0 = }
3. While examining Figure 8, listen to Tape 3A for a short commentary.

FIGURE 8I/O LED PATTERN TRANSLATION

4. Transfer the contents of R0 (step 2) to Figure 8 in the spaces provided.
5. Predict the pattern of LED's (after the contents of R0 are transferred to external I/O port 7) by placing a checkmark in the appropriate columns of Figure 8.
6. Depress STEP OBSERVATION: The pattern of external I/O LED's matches the pattern which you predicted in Step 5.
 (yes) (no)
7. REINFORCE YOUR UNDERSTANDING:
 - a. Set a BREAKPOINT at address '1B8.'

NOTE: Try to complete multiple BREAKPOINT operations as a sequence WITHOUT depressing the MON key. Exit to the MONITOR after reaching a BKPT causes the breakpoint sense code 'B0' to remain in the breakpoint location, even if other BKPTs are set. When one of these codes is detected subsequently, the MONITOR is accessed, and "HELLO" is displayed.

NOTE: (continued) If an undesired 'B0' code remains at the BKPT location after a new breakpoint is set, use MEMORY DISPLAY & ALTER mode to restore the correct code.

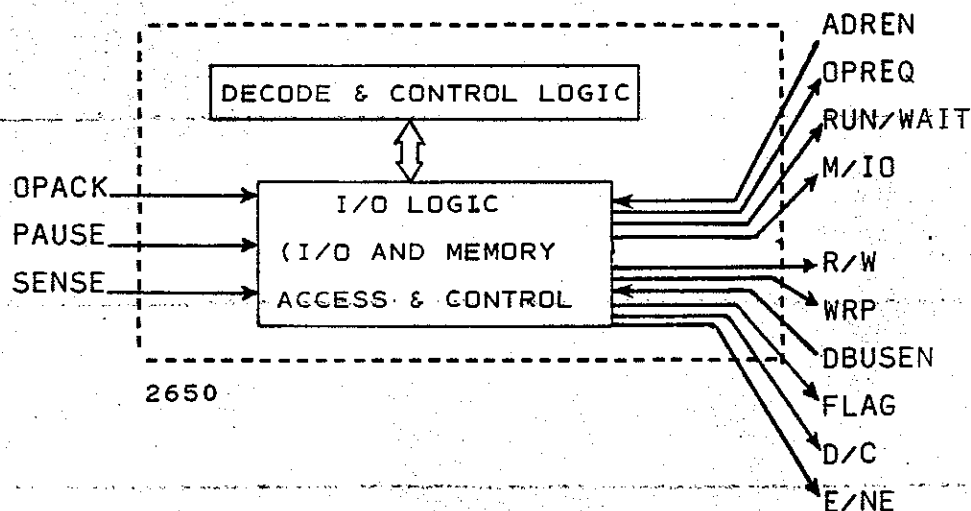
- b. Depress **RUN** to execute another pass of RLDICE.
- c. Repeat steps 1, 2, } above.
3, 4, 5, 6. } as explained previously.
- d. Repeat steps B and C.... until you can firmly establish that the contents of R0 are being transferred over the Data Bus to condition parallel I/O LED's (Extended I/O Port 7).

EXERCISE 4I/O LOGIC BLOCKINTRODUCTION:

At this point you are probably wondering exactly how the microprocessor can direct information to very specific external devices such as the parallel I/O LED's, or fetch from a precise location in memory. This is the function of the I/O Logic block which responds to the microprocessor's internal instruction decode and control logic.

DIRECTION:

Listen to Tape 3A for a brief discussion of the I/O Logic Block (Figure 9).

FIGURE 9I/O LOGIC BLOCKNOTES:

REFERENCE READING: INSTRUCTOR 50 Reference Manual; 2650 Line Description

DIRECTION:

Go right on to the next page.

DEMONSTRATION OF NON-EXTENDED I/O

In which the routine 'RLDICE' is modified to drive the external I/O LED's in NON-EXTENDED I/O mode.

PROCEDURE:

1. Set a program execution START ADDRESS at address '1B1.'
2. Toggle the I/O SELECTION SWITCH. to "NON-EXT DATA PORT." This selects "non extended port D" instead of "EXTENDED I/O PORT 07."
3. Refer to "RLDICE" Line 14. Note that the instruction at '1C0' causes the contents of R0 to be transmitted to (extended I/O) Port 7.
4. Depress

MEM	1	C
0	E/N	

 To access location '1C0' in memory (MEMORY DISPLAY AND ALTER mode)
5. Depress

F	0	E/N
C	0	E/N

 To ALTER memory locations '1C0' and '1C1.' By performing this step, you have changed the instruction sequence to cause data in R0 to be written to Port D.
6. Depress

STEP

 7 times sequentially OBSERVATION: Display = 01C0 F0
7. Depress

STEP

 once OBSERVATION: Display = 01C1 C0
8. Toggle the I/O SELECTION SWITCH to "EXTENDED PORT 07."
9. Repeat steps 1, 6, and 7. OBSERVATION: Display = 01C1 C0
External I/O LED's _____ (change)(no change)
10. Set a BKPT at address 1B3. so as to permit different "rolls" to take place.
11. Set the I/O Selection Switch in NON EXTENDED DATA PORT position.
12. Depress

RUN

 {If you have a problem, read, and perform the note at the bottom of page 13.
13. Depress

STEP

 6 times. OBSERVATION: Display = 01C0 C0
I/O LED's _____ (change)(no change)
14. Repeat steps 12 and 13 a few times.
15. Listen to tape 3A for a short commentary, then continue as directed.

NOTE: Steps 16 through 21 restore RLDICE to its original values.

- | | | | | | | | | | | |
|--|------|-----|-----|---|---|-----|------|------|-----|--|
| <p>16. Refer to routine RLDICE line 14.</p> <p>17. Repeat step 4.</p> <p>18. Depress <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>D</td><td>4</td><td>E/N</td></tr><tr><td>0</td><td>7</td><td>E/N</td></tr></table></p> <p>19. Repeat step 8.</p> <p>20. Depress <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>BKPT</td><td>BKPT</td></tr></table></p> <p>21. Repeat step 1 and depress RUN.</p> <p>22. Depress <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>RST</td></tr></table></p> | D | 4 | E/N | 0 | 7 | E/N | BKPT | BKPT | RST | <p>Reference documentation required for program restoration.</p> <p>Memory Display & Alter mode.</p> <p>To restore data at locations '1C0' and '1C1' to original values.</p> <p>I/O Selection SW = "EXTENDED I/O PORT 07."</p> <p>BKPT depressed <u>twice</u> cancels the outstanding 8BREAKPOINT.</p> <p>Program execution start address.</p> <p>To execute CRAPGAME. It should operate properly. Play for a few dice rolls then go on to Exercise 5.</p> |
| D | 4 | E/N | | | | | | | | |
| 0 | 7 | E/N | | | | | | | | |
| BKPT | BKPT | | | | | | | | | |
| RST | | | | | | | | | | |

NOTE: You may have had a few problems trying to complete this exercise. Its understandable... and very good if you found your problem..... and took the necessary steps to correct it.

EXERCISE 5

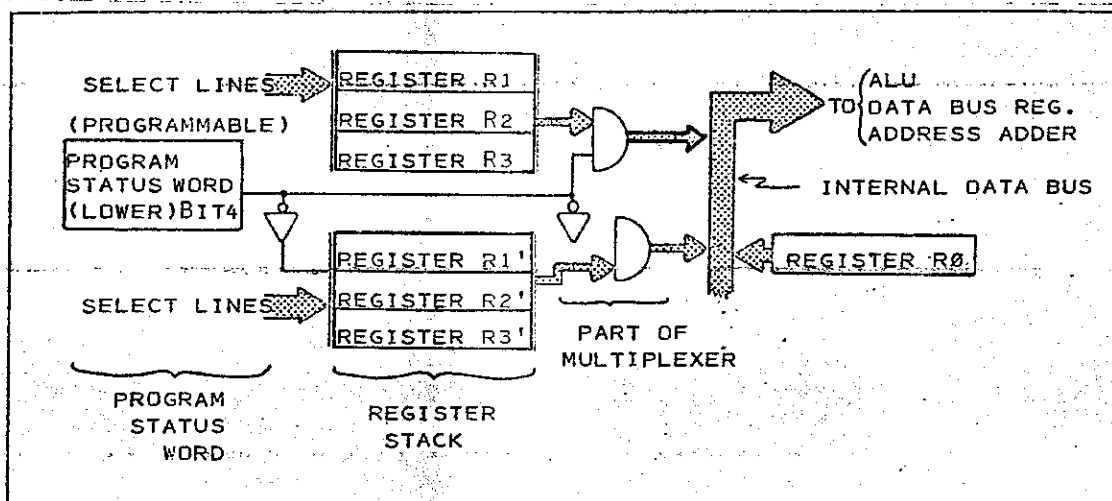
REGISTER STACK - REGISTER 0

INTRODUCTION:

There are 7 general purpose registers in the microprocessor, one of which (R0) is common to all register operations. Referring to Figure 10 (below) listen to tape 3A for further commentary.

FIGURE 10

GENERAL PURPOSE REGISTER BLOCK



PROCEDURE:TO INSPECT AND ALTER THE REGISTERS

In this procedure, you'll gain additional experience in displaying and altering the contents of the microprocessor's general purpose registers.

1. Depress MON to stop CRAPGAME in progress if running.
2. Set a START address at address '1B1.' for execution of routine RLDICE.
3. Set a BREAKPOINT at address '1C4.'
4. Depress RUN to execute routine RLDICE at processor speed (addresses '1B1' - '1C4.')
5. Depress REG C OBSERVATION: NEXT INSTRUCTION EXECUTION address at address _ _ _ _
6. Depress STEP once.
7. Depress REG 0 OBSERVATION: Record the contents of the registers in Table 1 (below). Use the column corresponding to PC = '1C6'
- E/N
E/N ... until you have inspected all registers.
- B. Repeat steps 6 and 7.....Record results in next column.
9. Repeat step 8.....until the table is completed.
10. Listen to Tape 3A for additional commentary:

TABLE 1

PC= →	'1C6'	'1C8'	'1CA'	'1CC'	'1CE'
R0 =					
R1 =					
R2 =					
R3 =					
(R1') R4 =					
(R2') R5 =					
(R3') R6 =					

NOTE:

NO CHANGE
IN THESE
REGISTERS

DIRECTION:

Go right on to the next page.

11. This step disables R1 through R3 and enables R1 'through R3.'

Depress MEM 1 1 E/N 1 A E/N

to disable R1 through R3 and enable R1' thru R3.'

12. Depress RST

to execute CRAPGAME.

13. Play until dice roll a few seconds

to establish random data.

14. Depress MON

to stop play after dice are "rolling."

15. Employ REGISTERS DISPLAY AND ALTER mode to zero R1 through R3 (non prime register bank)

16. Repeat steps 2 through 6

to set up same RUN conditions as when data was recorded previously.

17. Repeat steps 7 through 9

to record data in Table 2.

18. Listen to Tape 3A for further commentary.

TABLE 2

PC =	'1C6'	'1C8'	'1CA'	'1CC'	'1CE'
R0 =					
R1 =					
R2 =					
R3 =					
(R1') R4 =					
(R2') R5 =					
(R3') R6 =					

NOTES: _____

DIRECTION:

Go right on to the next page.

RESTORATION OF "CRAPGAME" TO ORIGINAL VALUES

19. Use MEMORY DISPLAY and ALTER mode to restore the contents of location '11' in memory to '0A.'
20. Use MEMORY DISPLAY and ALTER mode to write a single 2-byte instruction in (spare) memory at location '127'
 address '127' = '75'
 '128' = '10'

NOTE: When executed, this instruction will
 reset the REGISTER SELECT bit in the
 lower Program Status Word.

21. Set a START ADDRESS for program execution at location '127.'
22. Depress STEP to execute. Do NOT depress RUN.
23. Depress RST to ensure that "CRAPGAME" is executing correctly.
24. Go right on to Exercise 6.

////////////////////////////////////

EXERCISE 6ARITHMETIC LOGIC UNIT (ALU)INTRODUCTION:

One of the criteria by which the power and flexibility of a microprocessor is measured is in its ability to compute. In large measure, the ability of a microprocessor to compute is determined by the versatility of its Arithmetic Logic Unit. Now, listen to the tape as some of the Arithmetic Logic Unit's features are described.

NOTES: Reference Figure 6 page 5 this module.

REFERENCE READING:

2650 MICROPROCESSOR REFERENCE MANUAL
 "INTERNAL ORGANIZATION"

NOTE: Since ALU versatility and operation is demonstrated in detail later in this course, there is no procedure of steps associated with its introduction in this lesson. Go right on to the next exercise.

EXERCISE 7CONDITION CODE AND BRANCH LOGICINTRODUCTION:

As mentioned previously on tape, the CONDITION CODE provides a logical summary of the results of each operation in which the Arithmetic Logic Unit is involved. This 2-bit summary requires less internal logic than the 8-bit absolute ALU result. In fact it is through interpretation of the condition code that many of the program sequence control instructions decide which direction (Branch) program execution is to take.

DIRECTION:

Listen to tape 3A for a short commentary on the Condition Code and Branch Logic.

PROCEDURE:

1. Refer to routine "RLDICE" in the CRAPGAME (Module I, page 51) lines 1 through 11, as necessary.
2. Set a START ADDRESS for program execution at location '0000.'
3. Set a BREAKPOINT ADDRESS at location '1AA.'

NOTE: This step sets a BREAKPOINT address to the instruction where DICEXT (Dice Table Index, a constant) is reinitialized.

4. Depress

RUN

To execute CRAPGAME from address '0' to the BREAKPOINT address.

5. OBSERVATION:

In your own words, describe the activity of the 8-digit display once the dice start rolling. Do not depress SENS.

6. Depress **RUN** several more times at 5 to 7 second intervals.

OBSERVATION: Your observation made in step 5 is _____
(confirmed)(not confirmed)

7. Depress **REG** **1**

OBSERVATION: R1 = _ _

Depress **MEM** **1** **A**

Memory loc 1AF = _ _

F **E/N**

DIRECTION:

Go right on to the next page.

8. Depress OBSERVATION: Address '1AF' =

9. Observe the program counter "next address" and depress sequentially 4 times.

10. Read routine RLDICE, lines 7 through 12, particularly the comment field.

NOTE: The decision made by the BSTR instruction in line 11 depends on the logical state of the CONDITION CODE. That state in turn is determined by the activity of the carry bit. It is set when 1 is added to FF by the ALU.

OPTION: Use MEMORY DISPLAY AND ALTER mode to delete the instruction at line 10. Substitute 'C0' codes at location '1B7' and '1B8.' Depress 'C0' codes define "NO OPERATION." Does the program operate the same? (yes)(no). Therefore, was the TPSL instruction at line 10 necessary? (yes)(no).

Prove it by :

(a) Depressing to arrive at BREAKPOINT address '1AA.'

(b) Using REGISTER DISPLAY AND ALTER mode to change R1 contents from 'DB' to 'FF.'

NOTE: This step makes the processor 'think' that it has executed the RLDICE routine until the entire dice code table has been cycled.

(c) Set a START ADDRESS at location 1B5.

(d) Depress 4 times exactly. Carefully observe the sequence of program execution after each step.

<input type="button" value="STEP"/>	(1) Display <u>01B7 C0</u>
<input type="button" value="STEP"/>	(2) <u>01B8 C0</u>
<input type="button" value="STEP"/>	(3) <u>01B9 38</u> (the BSTR instruction)
<input type="button" value="STEP"/>	(4) <u>01AA 05</u> (... to the subroutine) "GETXT."

- (e) Now repeat steps (b) through (d), except modify the contents of R1 from 'FF' to 'DB.'

OBSERVATION Display: 01B7 C0
 WHEN STEP 01BB C0
 DEPRESSED: 01B9 38
 01BB)D

- (f) Was subroutine "GETXT" at location '1AA' accessed? _____ (yes)(no)

DIRECTION: Listen to tape 3A for further comment.

Depress MEM 1 B 7 E/N B 5 E/N 0 1 E/N

to restore program to original condition.

Ensure that the Breakpoint code at location '1AA' is deleted!

////////////////////

EXERCISE 8

PROGRAM STATUS WORD (PSU AND PSL)

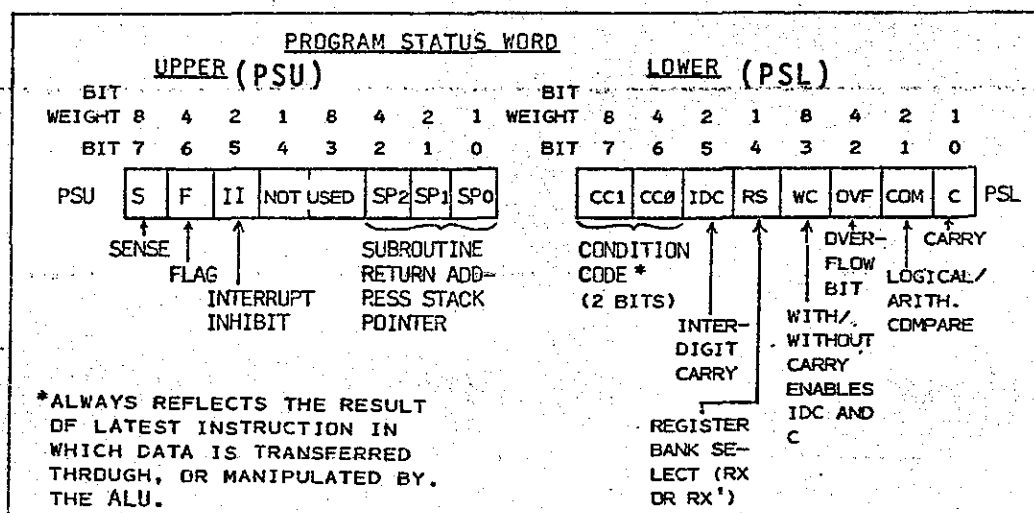
INTRODUCTION:

The PROGRAM STATUS WORD (PSW) is contained in two 8-bit registers in the microprocessor, designated PROGRAM STATUS UPPER (PSU) and PROGRAM STATUS LOWER (PSL). The PSW contains both STATUS indications of and control bits for, the microprocessor's activities. You will receive detailed instruction on the PSW in a separate module of this course.

DIRECTION: Listen to tape 3A for a brief introduction to the Program Status word, referring as necessary to Figure 6 (page 5), and Figure 11 (below).

FIGURE 11

PROGRAM STATUS WORD DIAGRAM



PROCEDURE:

NOTE: Steps 3 and following demonstrate the dependence of program operation upon the activity of the Program Status Word. Perform the sequence exactly as indicated. Refer to routine "RLDICE," page 51 lines 1-5.

1. Set a START ADDRESS for program execution at location '0.'
2. Set a BREAKPOINT (ending) ADDRESS at location '1AA.'
3. Depress **RUN** to execute to the roll of dice. Do not depress **SENS** when execution halts at the BREAKPOINT.
4. Alter the contents of R1 from 'DB' to 'FE.'
5. Depress **REG** **7** ----- OBSERVATION: PU = __ (PSU)
E/N ----- PL = __ (PSL)
6. Depress **STEP** twice Display should show a return from the "GETXT" subroutine to routine RLDICE at location '1BB.'
7. Inspect the PSU OBSERVATION: PU = . The PSU should decrease by 1 when compared to step 5 observation.
8. Depress **BKPT** **1** **B** **3** To set new BREAKPOINT.
9. Depress **RUN** To execute routine "RLDICE" once from location '1BB' to '1B3.'
10. Inspect R1 OBSERVATION: R1 = . Should equal 'FE.'
11. Depress **STEP** once To execute the ADD instruction (line 9 in routine "RLDICE").
12. Inspect R1, R7, R8 OBSERVATION: R1 = __ PU = __
PL = __
13. Depress **RUN** once OBSERVATIONS: R1 = __ PU = __
PL = __
(should equal results obtained in step 13)
14. Repeat steps 12 and 13 OBSERVATIONS: R1 = __ PU = __
PL = __

INTERPRETATION:

To observe effect on R1 and PL (PSW-lower) after execution of the ADD instruction.

- R1 should equal '00.'
- PU (Program Status Upper) should not change.

- PL (Program Status Lower) Most significant digit should be 2.

- Least significant digit (LSD) should be an ODD number (e.g., 3).

15. Listen to tape 3A for a brief commentary on the interpretation of step 14 results.

NOTES: 'FF' + 1 = '00' and a CARRY.
'FE' + 1 = 'FF' and no CARRY.

16. Repeat steps 12 and 13

OBSERVATION: R1 = PU =
PL =

There should have been NO change in the Program Status word. PSL's most significant digit should still be a 2, indicating that the condition code bits 7 and 6 of the PSL are still reset.

17. Listen to the audio tape for an introduction to the next step. Make the observations (below) when instructed to do so.

Depress STEP (5 times)

STEP ... STEP ..

OBSERVATIONS: Display =

18. Now, depress RUN, then alternately depress STEP and observe the display.....

NOTE: You may also wish to inspect the PSL to see if the condition code changes, particularly when the 2 instructions preceding the BSTR instructions are executed.

Note that subroutine "GETXT" was not accessed this time, because the condition for the branch was not satisfied.

DIRECTION:

Go on to Exercise 9 on the next page.

EXERCISE 9RETURN ADDRESS STACK

INTRODUCTION: (Ref. Figures 6 and 12; this module)

Whenever a program subroutine is called, a transfer of program sequence control to the new routine is performed. As the transfer is initiated, the next address in the main line program is saved in the return address stack, and the stack pointer in the PSW (upper) is incremented by one. Upon completion of the accessed subroutine, the microprocessor retrieves the saved address, using it to return program sequence control to the original routine.

DIRECTION: Take a few moments to study Figure 12 briefly, then go on to Exercise 10. The STACK POINTER is discussed in detail in Module IV-H of this course.

FIGURE 12 RETURN ADDRESS STACK & POINTER

RETURN ADDRESS STACK				PSU (PROGRAM STATUS WORD - UPPER)			DISPLAY
				DECR. SP2	SP1	SP0	LSD FOR PSU
LEVEL	15-BIT RETURN ADDR						
0				0	0	0	0
1	"	"	"	0	0	1	1
2	"	"	"	0	1	0	2
3	"	"	"	0	1	1	3
4	"	"	"	1	0	0	4
5	"	"	"	1	0	1	5
6	"	"	"	1	1	0	6
7	"	"	"	1	1	1	7

INCR.

STK PTR

NOTES: _____

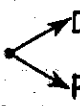
EXERCISE 10

INSTRUCTION ADDRESS REGISTER
OPERAND ADDRESS REGISTER
ADDRESS ADDER
OUTPUT CONTROL

INTRODUCTION:

Figure 13 diagrams the microprocessor's internal addressing configuration. The power of the microprocessor is determined both by its speed and the variety of instructions it can execute (e.g., the microprocessor's versatility). In turn, the variety of instructions is determined by the number of different operations (e.g., LOAD, ADD, STORE) it can perform and the number of distinct addressing modes in which each operation can be implemented.

The 2650 microprocessor has several distinct address modes, these include:

- REGISTER ADDRESSING
- IMMEDIATE ADDRESSING
- RELATIVE ADDRESSING
- ABSOLUTE ADDRESSING 
- ZERO-REFERENCED RELATIVE
- INDIRECT AND INDEXED ADDRESS VARIATIONS
- I/O ADDRESSING

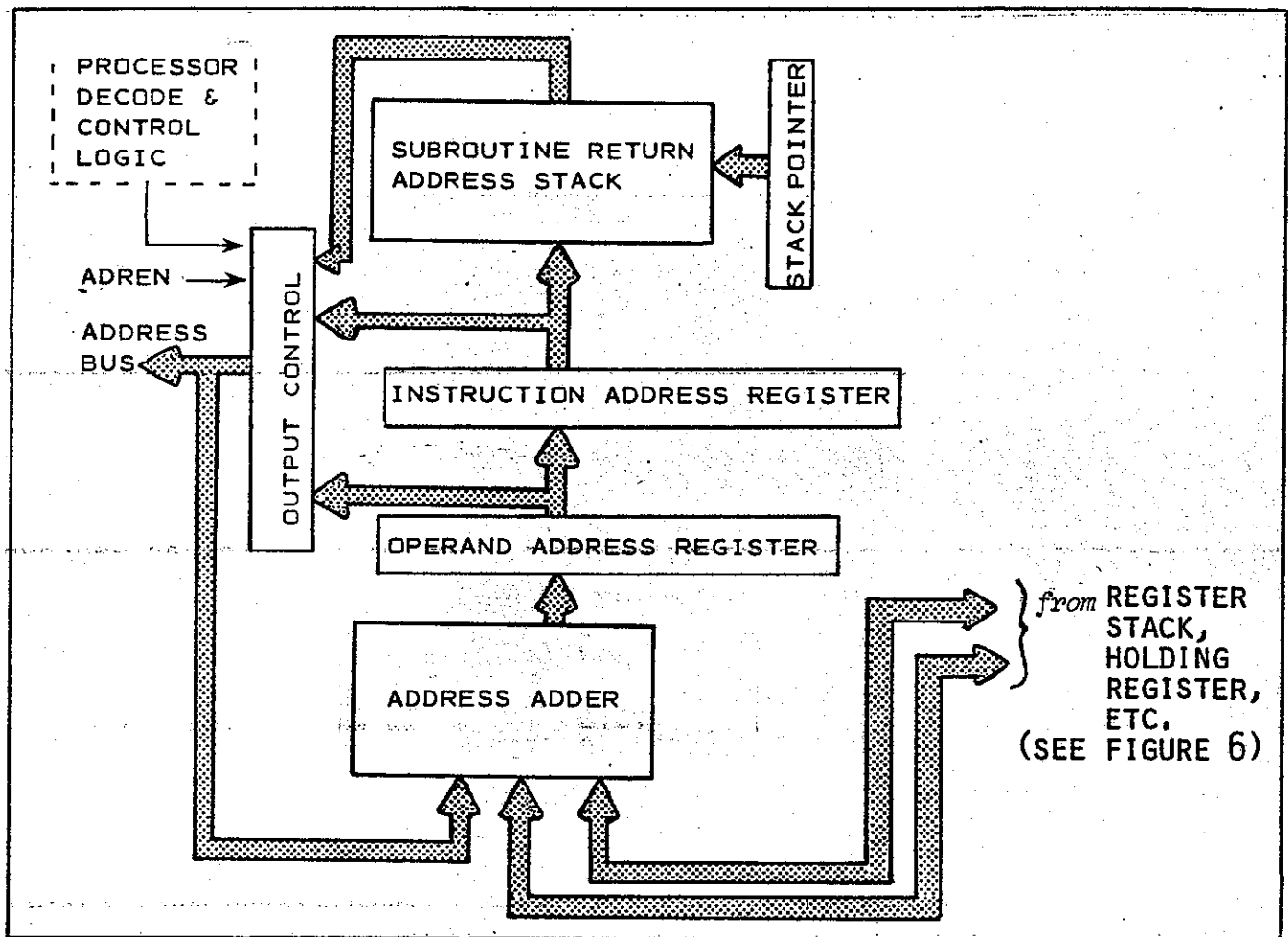
The subject of addressing is so important (and complex) that an entire subsection of this course (Module IV-F) is dedicated to its explanation and demonstration. Even before we make a direct study of addressing concepts and control, it is inevitable that the effects of different addressing modes are demonstrable. In your activities concerning CRAPGAME (a major part of the course) you've seen this, both in the documentation of CRAPGAME's program, and in its demonstration within the exercises of this module and Module I.

DIRECTION:

Listen to tape 3A for a brief introduction to each address control block, referenced to Figure 13 on the next page. Further directions are provided on tape.

FIGURE 13

MICROPROCESSOR ADDRESSING CONFIGURATION

**NOTES:****DIRECTIONS:**

This concludes your activities in Module II-A of the course. At this time, perform the following steps:

- Prepare Tape 3 for playback; side B up. Do not rewind from where you stopped after the last taped discussion on side A.
- Turn to page 1 in Module II-B; "SYSTEMS MEMORY INTERFACE (SMI)".
- Read the introduction. Restart the tape when directed to do so.



2650 MICROPROCESSOR COURSE

MODULE II - B

2656 SYSTEMS MEMORY INTERFACE (SMI)

PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086

REFERENCE:

Outlines and Abstracts
page 6

INTRODUCTION

The 2656 Systems Memory Interface (SMI) provides all the functions necessary to support the MICROPROCESSOR as a fully-operational 2-chip microcomputer. In summary, these functions include:

- 2K (2048 8-bit bytes) of Read-Only Memory (ROM)

- 128 bytes Random Access Memory (RAM)

- Complete CPU clock logic and oscillator

- Fully-programmable dedicated external I/O Control Lines (8)

The SMI is particularly useful in applications where the physical size of the computer's operational memory is limited. One such application is its use within the "INSTRUCTOR." The entire operating program (User System Executive - USE) resides within the SMI's 2K of ROM.

This program also makes use of 64 bytes of RAM, also resident within the SMI, for storage of variables such as breakpoint addresses, register contents and file identification. The other 64 bytes of SMI RAM may be designated by the user for storage of his program. For example, the messages "YOU BEAT THE HOUSE," "PLACE BET," etc., associated with the CRAPGAME are located in this available RAM area. (The INSTRUCTOR also has 512 bytes of USER RAM located on the INSTRUCTOR's printed circuit board, but external to the SMI - this is where routines such as 'RLDICE' are located).

Further, the SMI contains clock oscillation and generation circuitry for itself, the microprocessor, and externally controlled I/O. Thus, timing is generated simply by the output of an inexpensive crystal applied to the clock input lines of the SMI.

Finally, the SMI contains highly complex decode circuits in a Mask Programmable Gate Array (PGA). These decode circuits, operating from inputs supplied by the Microprocessor, generate a variety of signals for control of external I/O via the "INSTRUCTOR's" S100 Bus. These signals are fully compatible with industry-standard S100 interface control and operation.**

The functions performed by the SMI are equivalent to those performed formerly by at least a dozen MSI integrated circuits. Thus, use of the SMI permits not only a reduction of total chip count, but minimization of PCB space ... and design time required.

** The S100 bus consists of a group of defined input/output control lines and signals whose purpose is to sequence the orderly address of selected I/O devices, and orderly transfer of data between the microprocessor and its controlled I/O. These lines are provided on a 100-pin edge connector at the back of the INSTRUCTOR.

DIRECTION:

1. Access and install tape 3B, System Memory Interface,
in your tape recorder.
2. Take a look at Figure 1; "BASIC MICROCOMPUTER SYSTEM" as you
listen to the audio tape. The rest of this page provides/
convenient space for your observations and notes.

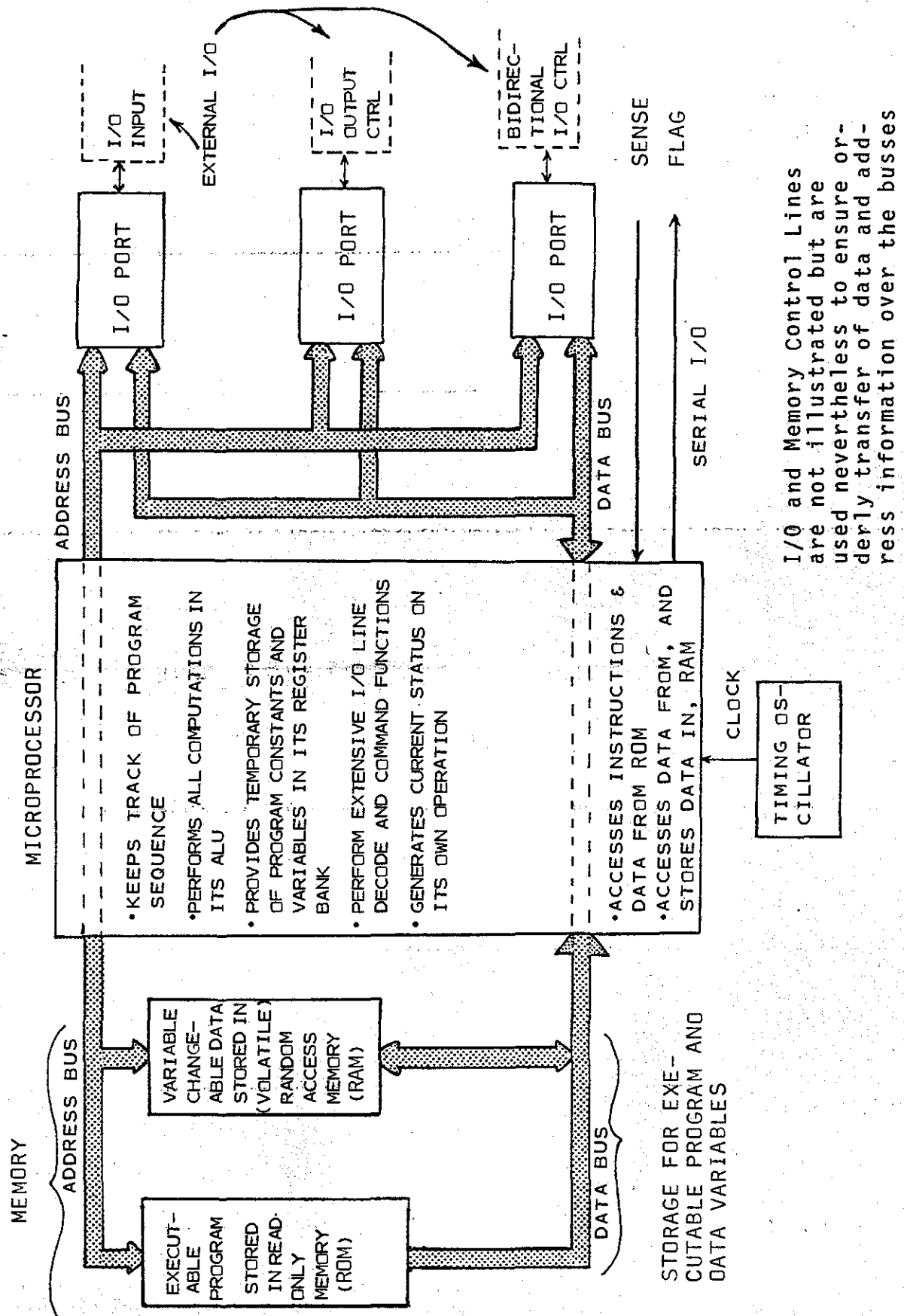
NOTES: _____

when directed to do so on tape, examine Figure 2.

NOTES: _____

_____Reference Reading: 2650 REFERENCE MANUAL; pages ____ to ____.SUMMARY NOTES: _____

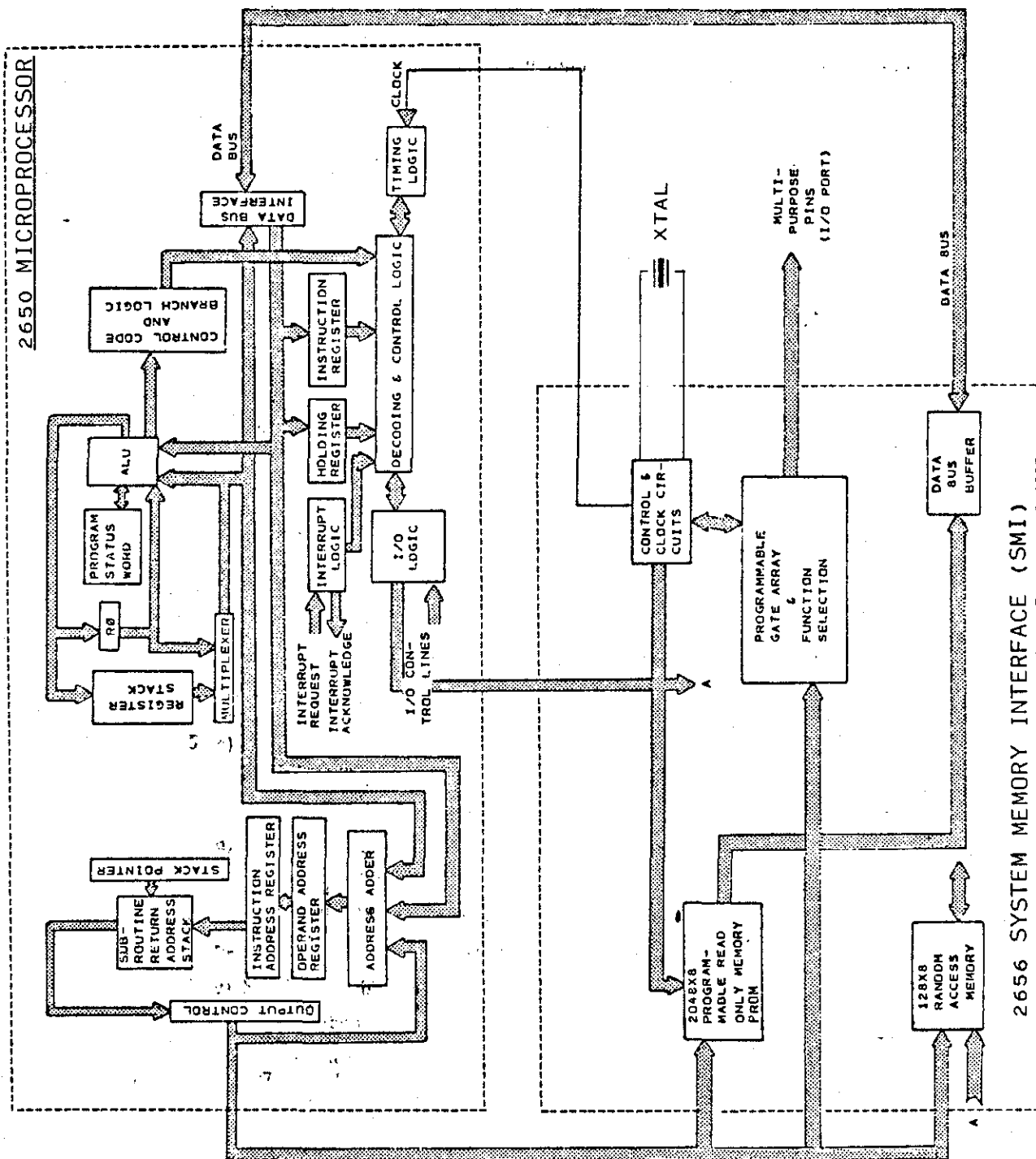
FIGURE 1 - BASIC MICROCOMPUTER SYSTEM



I/O and Memory Control Lines are not illustrated but are used nevertheless to ensure orderly transfer of data and address information over the buses

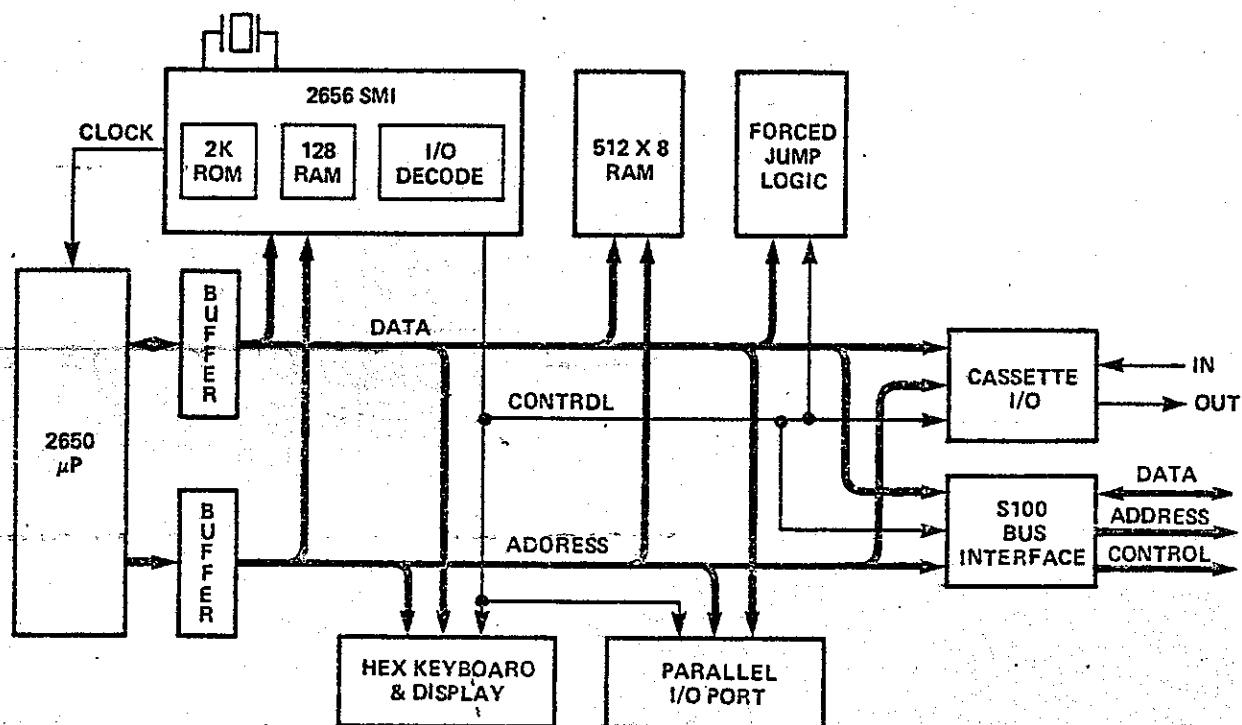
FIGURE 2

2650/SMI MICROCOMPUTER DETAIL



DIRECTION:

The Block Diagram of the "INSTRUCTOR" is reprinted on this page. When directed to do so on tape, study it in order to reinforce your concept of the usage of the MICROPROCESSOR and the SMI in the "INSTRUCTOR."

FIGURE 3**INSTRUCTOR BLOCK DIAGRAM**



2650 MICROPROCESSOR COURSE

MODULE III-A

INTRODUCTION TO SOFTWARE AND GLOSSARY

PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086

REFERENCE:

Outlines and Abstracts
page 7

Signetics

2650 MICROPROCESSOR COURSE

MODULE IV-B

2650 REFERENCE GUIDE

PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086

REFERENCE:

Outlines and Abstracts
page 10

INTRODUCTION:

one of the major tools that you'll be using in preparation of the software for your application is the 2650 REFERENCE GUIDE, otherwise known as the "HARD CARD." Folded accordion style for easy shirt pocket portability, the hard card contains a wealth of concentrated information useful for programming the microprocessor. In this section of Module IV, we'll discuss its layout and contents ... and what you may wish to do to improve it.

DIRECTION:

Turn to the next page in this module, and listen to Tape 4B "Instruction Repertoire I for a brief description of each section of the 2650 REFERENCE GUIDE.

**SIGNETICS
2650
REFERENCE
GUIDE****2650
2650
2650
2650**

INSTRUCTION REFERTOIRE - PART 1

FOLD LINE

FOLD LINE

PSW BITS AFFECTED															NOTE	2650 REF. MAN. PAGE			
DESCRIPTION OF OPERATION	F O R M A T	MNE- MONIC	OP CODE R r r CC				BIT FOR- MAT*	PSW BITS AFFECTED											
			0	1	2	3		BYTES	CYCLES	CC	IDC	C	OVF	SP			H	F	
LOAD/STORE	Load register zero	2	LOO	Z	00	01	02	03	1	2	Z	•						1	52
	Load immediate	5		I	04	05	06	07	2	2	I	•						1	
	Load relative	7		R	08	09	0A	0B	2	3	R	•						1,6	
	Load absolute	10	A	0C	0D	0E	0F	3	4	A	•						6	53	
	Store register zero	2	STR	Z	—	C1	C2	C3	1	2	Z	•						1	54
	Store relative	7		R	C8	C9	CA	CB	2	3	R	•						6	
Store absolute	10	A		CC	CD	CE	CF	3	4	A	•						6	55	
ARITHMETIC	Add to register zero w/w/o carry	2	ADD	Z	80	81	82	83	1	2	Z	•	•	•	•			1	
	Add immediate w/w/o carry	5		I	84	85	86	87	2	2	I	•	•	•	•			1	56
	Add relative w/w/o carry	7		R	88	89	8A	8B	2	3	R	•	•	•	•			1,6	
	Add absolute w/w/o carry	10	A	8C	8D	8E	8F	3	4	A	•	•	•	•			1,6		
	Subtract from register zero w/w/o borrow	2	SUB	Z	A0	A1	A2	A3	1	2	Z	•	•	•	•			1	57
	Subtract immediate w/w/o borrow	5		I	A4	A5	A6	A7	2	2	I	•	•	•	•			1	58
	Subtract relative w/w/o borrow	7		R	A8	A9	AA	AB	2	3	R	•	•	•	•			1,6	
	Subtract absolute w/w/o borrow	10	A	AC	AD	AE	AF	3	4	A	•	•	•	•			1,6	59	
	Decimal adjust register	3	DAR		94	95	96	97	1	3	Z	•						1,10	89
	LOGICAL	AND to register zero	2	AND	Z	—	41	42	43	1	2	Z	•						1
AND immediate		5	I		44	45	46	47	2	2	I	•						1	
AND relative		7	R		48	49	4A	4B	2	3	R	•						1,6	60
AND absolute		10	A	4C	4D	4E	4F	3	4	A	•						1,6		
Inclusive-OR to register zero		2	ICR	Z	50	51	52	53	1	2	Z	•						1	61
Inclusive-OR immediate		5		I	54	55	56	57	2	2	I	•						1	
Inclusive-OR relative		7		R	58	59	5A	5B	2	3	R	•						1,6	62
Inclusive-OR absolute		10	A	5C	5D	5E	5F	3	4	A	•						1,6		
Exclusive-OR to register zero		2	ECR	Z	20	21	22	23	1	2	Z	•						1	63
Exclusive-OR immediate		5		I	24	25	26	27	2	2	I	•						1	
Exclusive-OR relative	7	R		28	29	2A	2B	2	3	R	•						1,6	64	
Exclusive-OR absolute	10	A	2C	2D	2E	2F	3	4	A	•						1,6			
ROTATE/COMPARE	Compare to register zero arithmetic/logical	2	COM	Z	E0	E1	E2	E3	1	2	Z	•						2	65
	Compare immediate arithmetic/logical	5		I	E4	E5	E6	E7	2	2	I	•						3	
	Compare relative arithmetic/logical	7		R	E8	E9	EA	EB	2	3	R	•						3,6	66
	Compare absolute arithmetic/logical	10	A	EC	ED	EE	EF	3	4	A	•						3,6	67	
	Rotate register w/w/o carry	3	RRR		50	51	52	53	1	2	Z	•	•	•	•			1	68
	Rotate register left w/w/o carry	3	RRL		00	01	02	03	1	2	Z	•	•	•	•			1	67
BRANCH	Branch on condition true relative	7	BCR	R	12	13	1A	1B	2	3	R							7,8	
	Branch on condition true absolute	8		A	10	1D	1E	1F	3	3	B							7,8	74
	Branch on condition false relative	7	BCF	R	96	99	9A	—	2	3	R							7	
	Branch on condition false absolute	8		A	9C	9D	9E	—	3	3	B							7	75
	Branch on register non-zero relative	7	BRN	R	98	99	9A	9B	2	3	R							7,8	
	Branch on register non-zero absolute	8		A	9C	9D	9E	9F	3	3	B							7,8	78

GENERAL USE
OF INSTRUCTION
GROUP

DETAILED DEFINITION
OF OPERATION

ASSEMBLER FORMAT
(LAST PAGE OF HAND
CARD)

SYMBOLIC
INSTRUCTION
OPCODE

HEXADECIMAL
CODE FOR
OPCODE (1ST)
BYTE OF
INSTRUCTION

MEMORY
LOCATIONS
REQUIRED
(IN BYTES)

MACHINE
CYCLES
NX3(TCLK)*
=EXECUTION
TIME OF
INSTRUCTION

PROGRAM STATUS WORD
CONTROL AND STATUS
BITS AFFECTED
(SEE REVERSE SIDE
OF HARD CARD
• = AFFECTED)

DETAILED FORMAT
ADDRESS MODE
(SEE REVERSE SIDE
OF HARD CARD)

PAGE REF.
2650 MICRO-
PROCESSOR
MANUAL
(ADD TO
YOUR CARD)

[TCLK]

*TCLK = DURATION OF CLOCK INPUT TO 2650 (50% LEADING EDGE TO LEADING EDGE)



2650 MICROPROCESSOR COURSE

MODULE IV-D

2650 PROGRAMMING FORM

PREPARED BY:

**MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086**

REFERENCE:

**Outlines and Abstracts
page 12**

INTRODUCTION:

As you implement the microprocessor's instruction set, you'll record the major port of your documentation on the PROGRAMMING FORM. In this module, the use and flexibility of this form is described. Particular emphasis is stressed on relating the form contents to projected code to be stored in memory.

DIRECTION

Listen to tape 4B for a brief commentary on the 2650 PROGRAMMING FORM and its relationship to memory (Figure 1). Key points to be discussed are indicated by the circled numbers (e.g., ③).

FIGURE 1: 2650 PROGRAMMING FORM;

2650 PROGRAMMING FORM

ROUTINE _____ START ADDR _____

DESCRIPTION _____

ROUTINE SHEET ____ OF ____

MEMORY LOCATIONS THIS SHEET _____

DIRECT RELATIVE ADDRESSING - SECOND BYTE

+	03	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
-	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71	70

INDIRECT RELATIVE ADDRESS: Add H'40' TO DISPLACEMENT

ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
	B2	B1	B2		OPCODE	OPERANDS	
1	0033	3F	01	2A	BSTA, UN	CORRSUB	
2	6	00	C1		ACON	PAS1XP	
3	8	C0			NOP		
4	39	06	85	PAS1	LODI, R2	H'85'	

HEX ADDR	7	6	5	4	3	2	1	0	EQUALS
0032									
0033	0	0	1	1	1	1	1	1	H'3F'
0034	0	0	0	0	0	0	0	1	H'01'
0035	0	0	1	0	1	0	1	0	H'2A'
0036	0	0	0	0	0	0	0	0	H'00'
0037	1	1	0	0	0	0	0	1	H'C1'
0038	1	1	0	0	0	0	0	0	H'C0'
0039	0	0	0	0	0	1	1	0	H'06'
003A	1	0	0	0	1	1	1	1	H'8F'
003B									

FOR THIS EXAMPLE, THE COMMENT FIELD IS NOT FILLED IN

3-BYTE INSTRUCTION

2-BYTE DATA

1-BYTE INSTRUCTION

2-BYTE INSTRUCTION

DIRECTION

Go right on to the next page when directed to do so on tape.

FIGURE 2

EXAMPLE TO ILLUSTRATE COMPATIBILITY OF PROGRAMMING FORM LAYOUT WITH MICROPROCESSOR COMPUTER-GENERATED ASSEMBLER LISTING

2650 PROGRAMMING FORM

ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
	B0	B1	B2		OPCODE	OPERANDS	
1							
8				*T TEMP	INDIRECT ADDRESS		
9				*			
10				***			***
11				*			
12	1D96	CD	17	MOVI	STRA, R1	T	SET INDIRECT ADDRESS
13	1D99	CE	17		STRA, R2	T+1	
14	1D9C	06	08		LODI, R2	8	SET INDEX TO MOVE 8 BYTES
15	1D9E	DE	F7	MOVI	LDA, R0	*T, R2	GET A BYTE
16	1DA1	CE	77		STRA, R0	DISBUF-1, R2	MOVE TO BUFFER
17	1DA4	FA	78		BDRR, R2	MOVI	

FORM/LISTING
LINE NUMBER;
NO RELATION-
SHIP

LINE
NUMBER

ADD-
RESS
1, 2,
OF 1ST
BYTE

DATA
1, 2,
OR 3
BYTES

LABEL
FIELD

OPCODE
OPERANDS

SYMBOLIC
INSTRUCTION

COMMENT
FIELD

ASSEMBLER
LISTING

FIGURE 3

EXTRACT FROM SAMPLE PROGRAM

DIRECT RELATIVE ADDRESSING - SECOND BYTE

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F

DIRECT RELATIVE ADDRESSING - ADD A'00' TO DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE TRAIN B START ADDR 0000

DESCRIPTION continuation.

Location: File B; Tape 1A at 70-82

ROUTINE SHEET 2 OF 6 File B

MEMORY LOCATIONS THIS SHEET '35' (or 53₁₀)

ADDRS	B2	B1	B2	LABEL	OPCODE	SYMBOLIC INSTRUCTION OPERANDS	COMMENT
1				SWYARD	ACON	H'0038'	
2	0038	97	02	* SWYARD		IS an 8 BYTE data table	Original message at
3	3	05	00	SWYARD	RES	8	start-up:
4	E	12	10				" 2650 UP.
5				TEMP	ACON	H'0040'	
6	0040	XX	XX	* TEMP		IS 8-byte temporary data table for SWYARD motion	
7	3	XX	XX	* AND	DIRECTION	control	
8	6	XX	XX	TEMP	RES	8	
9							
10	0048	84	80	CHKSEN	TPSU	SENS	Test for REVERSE (SENSE KEY
11	A	18	04		BCTR, EQ	DELAY	down) could be; go find out
12	C	74	40		CPSU	FLAG	It wasn't, clear Flag like if
13	4E	1B	48		BCTR, UN	ROLLON	exit to ROLLON (fwd).
14	50	D8	7E	DELAY	BIRR, RO	\$	Set 1ms delay and clear
15	2	34	80		TPSU	SENS	if SENS depressed deliberately.
16	4	98	42		BCTR, EQ	ROLLON	It wasn't! Go fwd.
17	6	76	40	REVERSE	PPSU	FLAG	It was, set flag LED on
18	8	C1			STRZ	R1	and set constants to reverse
19	1	07	F9		LODI, R3	H'F9'	direction - Now, move
20	3	0D	60	LOOP1	LODA, RO	SWYARD, R1	SWYARD → TEMP. This time
21	5E	CD	20		STRA, RO	TEMP, R1, +	byte shift right, 7 bytes
22	61	DB	78		BIRR, R3	LOOP1	moved? NO! Finish.
23	3	08	5A		LODR, RO	SWYARD + 7	Yes! Move 25 SWYARD in.
24	5	C8	51		STRA, RO	SWYARD	to M.S. TEMP then move
25	7	0D	60	LOOP2	LODA, RO	TEMP, R1	all of temp back into SW.
26	006A	CD	60		STRA, RO	SWYARD, R1	YARD ... no shift.
27							

D

C

A

B

completes the address column before moving on to next sheet in program.

Using "hard-card" enter the code for the symbolic instructions, step by step.

From detailed flow chart, write down symbolic instructions, step by step.

Use "free form" style to complete "comment" field - give "application" meaning to symbolic instructions.

4

3

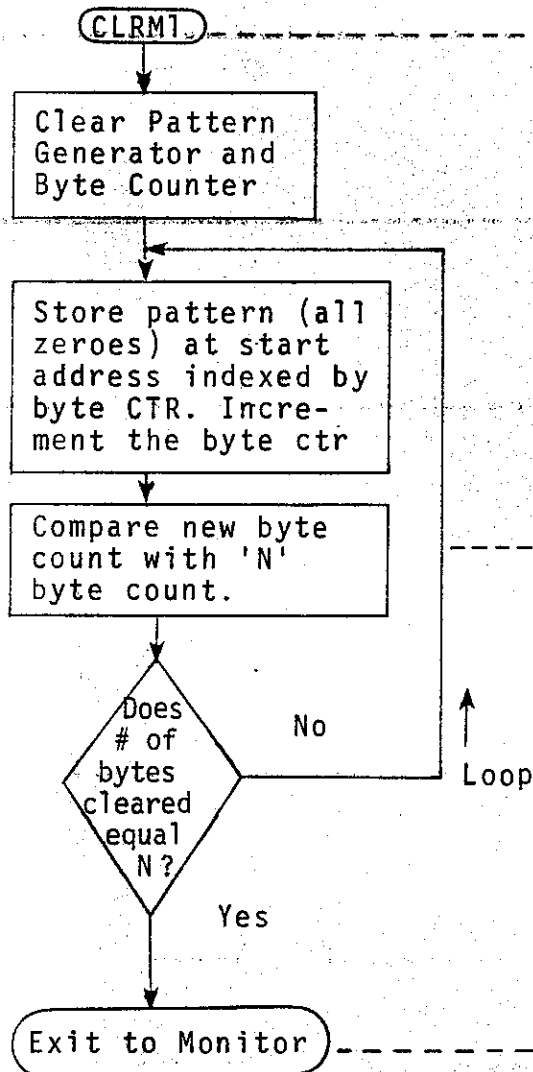
1

2

Exercise:

In which a short routine to zero N bytes of USER memory is developed, from a basic flow chart to an executable program.

1. Listen to Tape 5-A for a short description of the routine "CLRM1". The tape will reference the flow chart (Figure 4). Proceed with steps 2 and following when directed on tape to do so.

FIGURE 4CLEAR MEMORY FLOWCHART

A short routine to clear N bytes of memory, given a defined start address and the number of bytes ('N') to be zeroed.

Register assignment (proposed):
 R0 = Pattern Generator (step 2)
 R3 = Byte Counter

Keep zeroing bytes until the byte count = desired zero bytes (N)

Display = "HELLO"

2. Note the assignments given to the microprocessor's working registers PRIOR to writing the symbolic instruction sequence:
 - a. Pattern Generator: R0
 - b. Byte Counter: R3 ... serves as an index into memory
3. In this case, the routine will be written to zero 27 bytes of memory, from address H'103'. The START ADDRESS of routine CLRM1 is at memory location H'10'.
4. On a 2650 PRDGRAMMING FORM, identify the symbolic name of the routine, its START ADDRESS, and briefly describe it as shown in Figure 5.

FIGURE 5

<p><u>Assignments:</u></p> <p>R0 - pattern generator - all 0</p> <p>R3 - byte ctr/index</p> <p>STRTADDR = H'103 (this is the initial address of the block of memory to be cleared)</p> <p>N bytes = 27₁₀ = H'1B'</p> <p>NN = 27₁₀ = H'1B'</p> <p>(the number of bytes to be zeroed)</p>	<div style="border: 1px solid black; padding: 2px; width: 20px; margin: 0 auto;">4 0</div>	<p>2650 PROGRAMMING FORM</p> <p>ROUTINE <u>CLRM1</u> START ADDR <u>'10'</u></p> <p>DESCRIPTION <u>This routine, as written, clears 27₁₀ bytes of user memory, starting at location H'103'</u></p> <hr/> <p>ROUTINE SHEET <u>1</u> OF <u>1</u></p> <p>MEMORY LOCATIONS THIS SHEET _____</p>
---	--	---

NOTE: STRTADDR and NN are symbolic equates.

**** NOTE:** All assignments, labels, equates, address constants, memory reservations and terms may be defined by the programmer as he wishes in order to provide sufficient support documentation to the program. Thus, in this program, R0 is assigned the function of a PATTERN GENERATOR; the start address in memory is labeled (symbolically) as "STRTADDR".

5. Continuing with entry on the Programming form, the SYMBOLIC INSTRUCTION SEQUENCE is written next (Figure 6). Any line number may be chosen for the initial instruction. A START ADDRESS may also be entered. Use pencil for all entries as they are subject to change.

FIGURE 6

SYMBOLIC INSTRUCTION SEQUENCE SELECTED

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION	
		B0	B1	B2		OPCODE	OPERANDS
1							
2	0010				CLRM1	EORZ	R0
3						STRZ	R3
4					AGAIN	STRA, R0	STRTADDR, R3+
5						COMI, R3	NN
6						BCFR, EQ	AGAIN
7						BCTA, UN	MONITOR

NOTE

INSTRUCTIONS ARE SELECTED FOR THEIR USE IN SATISFYING EACH FLOW CHART ENTRY IN SEQUENCE.

SOURCE DOCUMENTATION IS 2650 HARD CARD AND REFERENCE MANUAL.

6. Normally, a few COMMENTS are written next, to tie the individual symbolic instructions loosely together. These may or may not change depending on the final sequence of code. Further, the comments need not be restricted to one line. Some instructions may require no comment. Others may require 2 or 3 lines for adequate description. If you are writing several pages of symbolic instructions, JOT DOWN the comments as you go. Otherwise you'll forget what the program is to do. Refer to Figure 7 for an example.

FIGURE 7

COMMENTS ARE LISTED

LABEL	SYMBOLIC INSTRUCTION		COMMENT
	OPCODE	OPERANDS	
CLRM1	EORZ	R0	In "CLRM1", first clear pattern generator and the byte ctr/index
	STRZ	R3	
AGAIN	STRA, R0	STRTADDR-1, R3, +	Then, store the zero pattern in memory, indexed by incremented byte ctr R3. Now, have the
	COMI, R3	NN	
	BCFR, EQ	AGAIN	required bytes been zeroed?
	BCTA, UN	MONITOR	No! go zero another byte.
			Yes! Job's done. Go tell
			user "HELLO".

NOTE:

In 1st write of program, several symbolic instruction lines may be left blank to leave room for expansion or change.

7. At this point, referencing the HARD CARD (and Reference Manual if necessary) the symbolic instructions are coded into hex suitable for entry into the INSTRUCTOR's memory. At this point, you can ignore the comment field. Refer to Figure 8.

FIGURE 8**HEX CODING**

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION	
		B0	B1	B2		OPCODE	OPERANDS
1							
2	0010	20			CLRM1	EORZ	R0
3		C3				STRZ	R3
4		CF	21	02	AGAIN	STRA, R0	STRTADDR-1, R3, +
5		E7	1B			COMI, R3	NN
6		98	79			BCFR, EQ	AGAIN
7		1F	18	00		BCTA, UN	MONITOR

NOTE:

There may be certain bytes (e.g., the one at (A)) that you are unable to code immediately. Reasons for this will be offered later in the course.

8. Finally, in the ADDRESS column, write the numerical addresses corresponding to the 1st byte of each instruction or data entry (Figure 9). AT your option, put in a comment to indicate the last address of a given routine (B).

FIGURE 9

MEMORY ADDRESS ENTRY

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION	
		B0	B1	B2		OPCODE	OPERANDS
1							
2	0010	20			CLRM1	EORZ	
3	1	C3				STRZ	
4	2	CF	21	02	AGAIN	STRA, R0	
5	5	E7	1B			COMI, R3	
6	7	98	79			BCFR, EQ	
7	19	1F	18	00		BCTA, UN	
8					* last	address =	H'1B'

(B)

9. Use the "INSTRUCTOR's" FAST PATCH mode to load routine CLRM1 into memory.

Depress **REG** **F** **1**

To set a start address for Fast Patch.

0 **E/N**

Enter the data from H'20' through H'00.'

Via the hex keyboard - memory addresses '10' through '1B.'

10. Verify the memory load Using MEMORY DISPLAY and ALTER mode.

11. Set a start address for program execution at location '10,' and depress RUN .

Alternative: Execute in STEP mode.

12. Verify that memory locations '103' through '11D' are zeroed. Using MEMORY DISPLAY and ALTER mode.

DIRECTION:

Listen to tape 5A for a summary, then go on to Module IV-E.

NOTES:

Signetics

2650 MICROPROCESSOR COURSE

MODULE IV-E

MEMORY CONCEPTS

PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086

REFERENCE:

Outlines and Abstracts
page 13

INTRODUCTION:

The selection of MEMORY components for use in your microprocessor-driven application is just as important as the selection of the microprocessor itself. Hardware aspects (e.g., timing control, size, and type) are discussed in another module of this course. This module describes memory as it is partitioned in order to implement the desired application. In particular, usage of ROM (Read-Only Memory) vs. RAM (Random Access Memory) is functionally differentiated. Addressing and data access is described as a basis for exercising the memory-dependent instruction set in subsequent course modules.

DIRECTION:

Listen to the audio tape for a description of FIGURES 1 through 6.

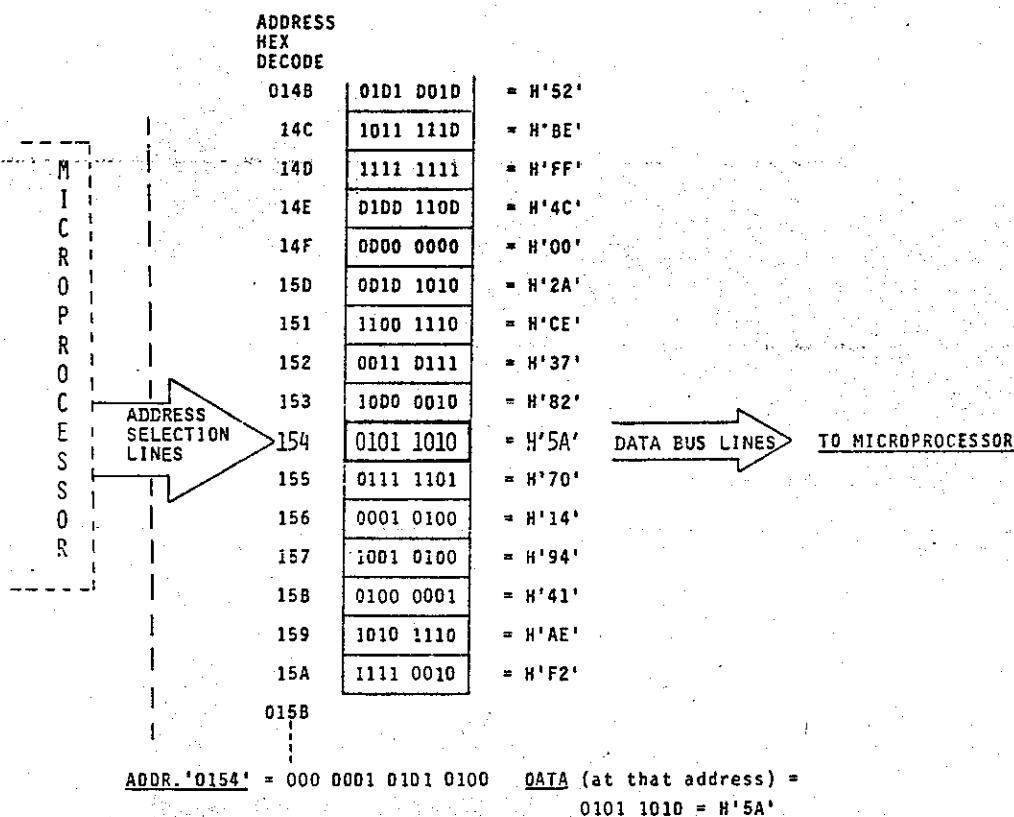
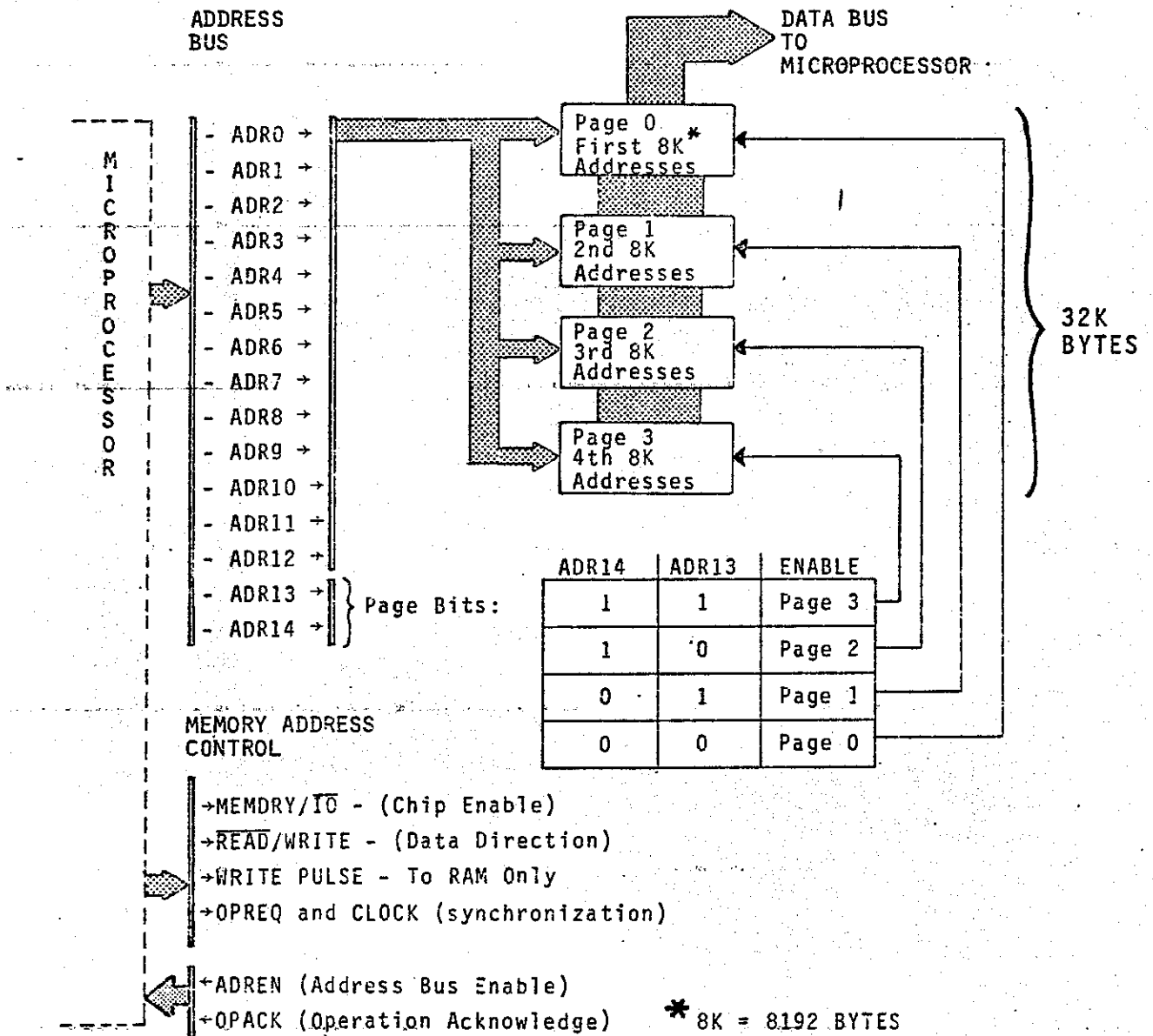
FIGURE 1CONCEPT OF MEMORY ADDRESS STORAGENOTES:

FIGURE 2

MEMORY ADDRESS SELECTION



NOTES:

FIGURE 3ROM vs RAM

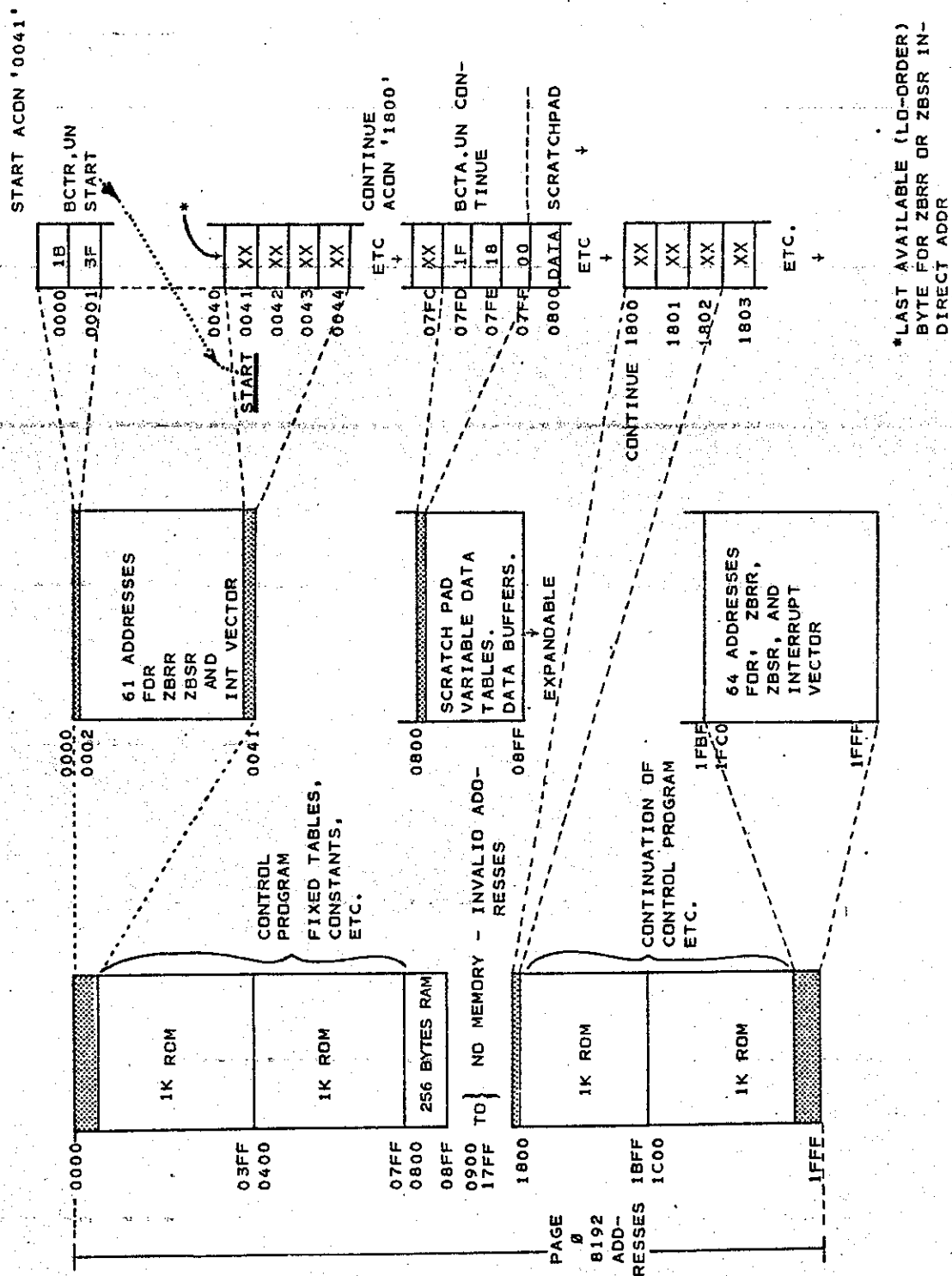
<u>ROM</u> READ ONLY MEMORY	<u>RAM</u> RANDOM ACCESS MEMORY
<ul style="list-style-type: none"> ● Provides storage for: <ul style="list-style-type: none"> * CONTROL PROGRAM * FIXED DATA - Messages <ul style="list-style-type: none"> - Tables - Constants ● <u>PROGRAMMING</u> <ul style="list-style-type: none"> * MASK (ROM) * FUSED-LINK (PROM) * ULTRA-VIOLET (EPROM) ● COST: (MASK) Lowest <ul style="list-style-type: none"> (FL) Higher (UV) Higher still 	<ul style="list-style-type: none"> ● Provides storage for: <ul style="list-style-type: none"> * UNDEBUGGED USER PROGRAM * VARIABLE DATA - Messages <ul style="list-style-type: none"> - Int. Results - Scratchpad ● <u>PROGRAMMING</u> <ul style="list-style-type: none"> * Direct to any address location under WRITE control * Data <u>lost</u> when power is turned off ● COST PER BIT: <ul style="list-style-type: none"> Always higher than ROM. less bits per package.

NOTES:DIRECTION:

Go right on to the next page.

FIGURE 4

2650 MEMORY ORGANIZATION - 4K PROGRAM



NOTES:

FIGURE 5

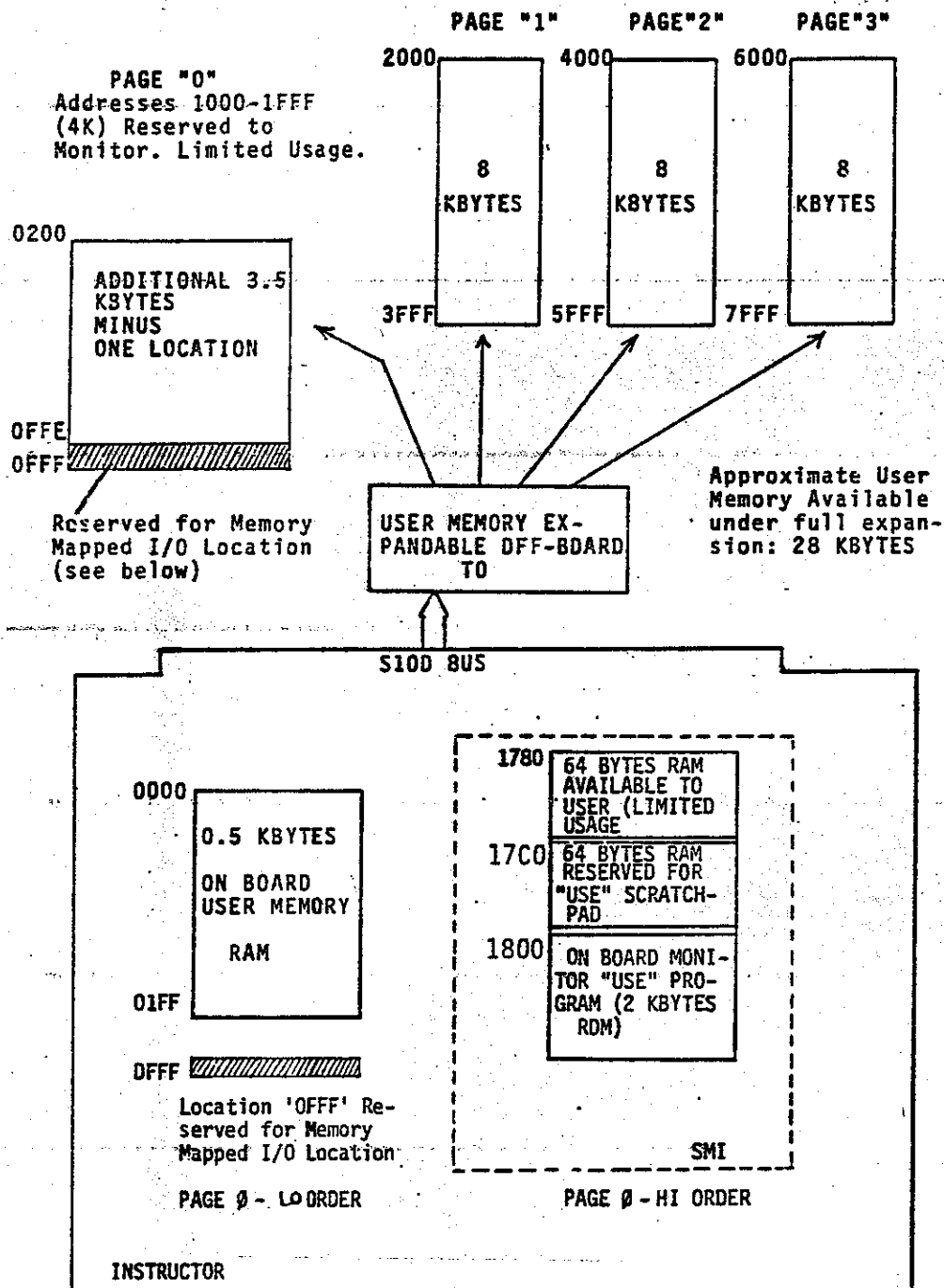
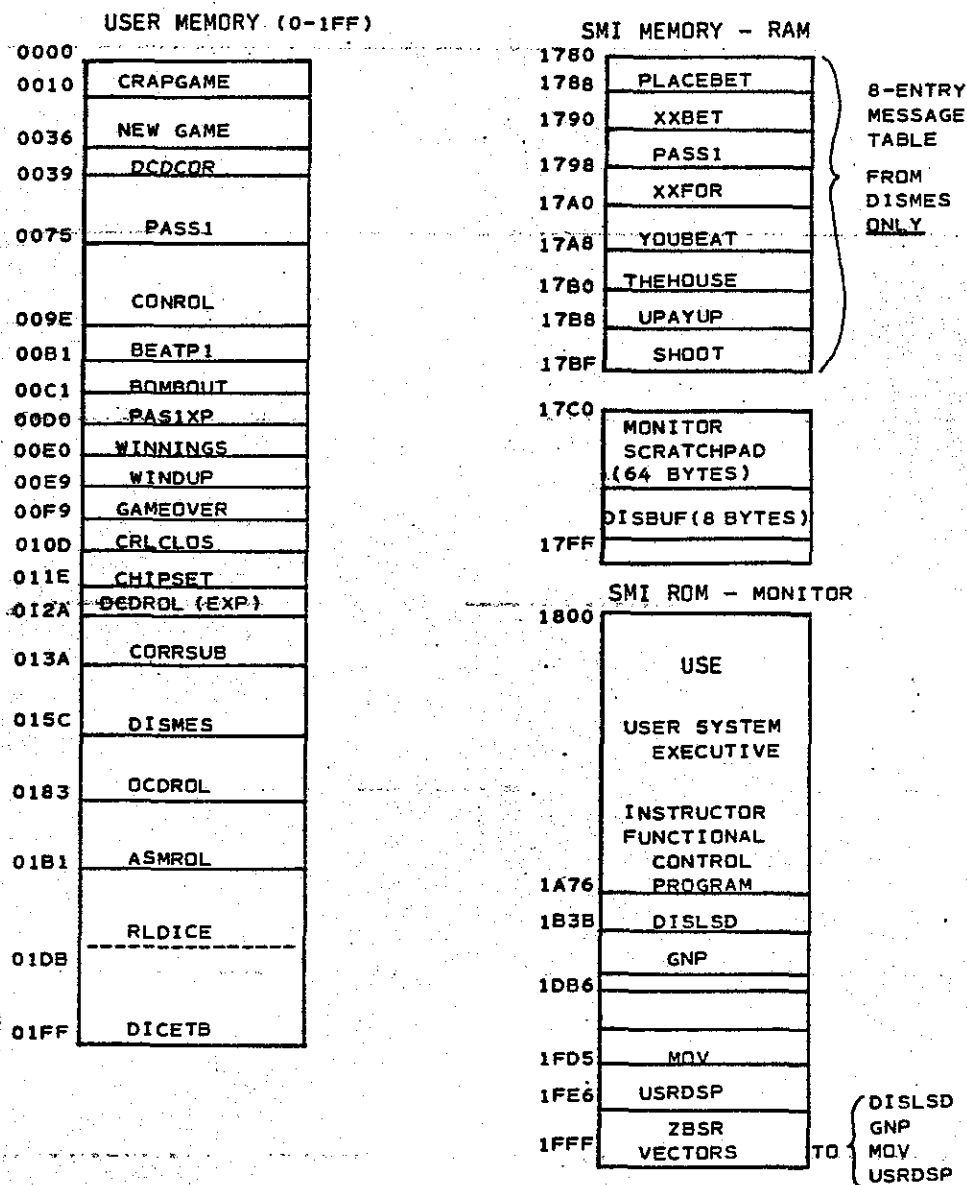
"INSTRUCTOR" MEMORY CONFIGURATIONNOTES:

FIGURE 6

"CRAPGAME" MEMORY MAPNOTES:

EXERCISE:**MONITOR USE PROGRAM INSPECTION****INTRODUCTION:**

In this exercise, you'll be inspecting selected data in the INSTRUCTOR's ROM (Monitor USE) and RAM (Monitor Scratchpad and USER Storage) memories. In certain cases, you'll modify this data, with some interesting results.

DIRECTION:

Complete the requirements of the following procedures.

PROCEDURE A:

1. Load "CRAPGAME" from your "SCRATCHTAPE" into the INSTRUCTDR's memory.
2. Ensure that CRAPGAME operates correctly, then go on to the next procedure.

PROCEDURE B:

Referring to Figure 7 on the next page, verify the contents of USER MEMORY from address '0' to address '27.'

1. Question: Which (if any) locations between addresses '0' and '27' contain DATA VARIABLES? _____
 - (a) locations '0' and '01'
 - (b) locations '0E' and '0F'
 - (c) locations '1D,' '1E,' and '1F'
 - (d) locations '27' and '28'
 - (e) none of the locations (above) contain variable data.

If your answer was a, b, c, or d (above), could the program as written be programmed into ROM? _____ (yes)(no).

Explain _____

2. Compare your answer to that provided on tape 5A.
3. To prove your answer, perform the following series of steps.
 - (a) Set a BREAKPOINT address at location '12F.'
 - (b) Set the PROGRAM COUNTER (REG C) at location '0.'

- (c) Depress REG 7 0 to reset the Upper Program Status Word.
- (d) Depress RUN
- (e) Make necessary manual entries (CHIPS and BET) until the message "-012F CC" is displayed.
- (f) Inspect locations 'OE' and 'OF.' Do they contain step the current BET and CHIPS which you entered in step d? _____ (yes)(no)
- (g) Repeat steps (b) through (f) a few more times, then listen to tape 5A for further direction.

FIGURE 7

CRAPGAME MAIN PROGRAM SHEET 1

	ADDRES			LABEL	SYMBOLIC INSTRUCTION		COMMENT
	80	81	82		OPCODE	OPERANDS	
1	0000	1B	88	CRAPGAME	BCTR, UM	* CHIPSET	Branch to start crapgame
2	2	01	B1			* RLDICE	ROLL THE DICE
3	4	01	3A			* DISMES	(ACONS) DISPLAY MESSAGE
4	6	01	83			* ASMROL	(INDIR-ADDS) ASSEMBLE ROLL
5	8	01	5C			* DCDROL	DECODE THE ROLL
6	A	01	0D			* CHIPSET	BUY INTO GAME
7	C	00	D0			* WINNINGS	CALC. WIN/LOSS
8	E	XX				BET	CURRENT BET AND
9	OF	XX				CHIPS	CHIPS ON HAND
10	10	77	0A	NEWGAME	PPSL	WC, COM	Set WC and logical compr
11	2	74	60		CPSU	FLAG, II	Clear Flag and Int. inhib.
12	4	05	DC		LODI, R1	H'DC'	Now, set DICETABLE constant and store away.
13	6	CD	01 AF		STRA, R1	DICEXT	
14	8	0B	74		LODR, R0	CHIPS	Now, get chips on hand
15	B	BB	F4		ZBSR	* DISLSD	and break into MS and
16	1D	CC	17 88		STRA, R0	XXBET=	LS nibs. Store these in
17	20	CD	17 89		STRA, R1	XXBET+1	the XXBET= table.
18	3	04	7F		LODI, R0	H'7F'	Now, set up running
19	5	C2			STRZ	R2	display to PLACE BET.
20	6	C3			STRZ	R3	Now, ...:
21	7	BB	84		ZBSR	* DISMES	display that message.
22	9	05	17		LODI, R1	<XXBET=-1	then set up constants to
23	B	06	87		LODI, R2	>XXBET=-1	place actual bet. It is
24	D	BB	FE		ZBSR	* MOV	entered via the keyboard
25	2F	04	05		LODI, R0	5	One digit limits 1<x<9
26	31	BB	FC		ZBSR	* GNPA	NO CHANGE! Next digit
27							exits routine.

PROCEDURE C:

In the following procedures, you'll be inspecting (and modifying) the contents of the Monitor's SCRATCHPAD memory in the "INSTRUCTOR." Refer to Figure 8 (below) as required for completion of this procedure. Figure 8 is a reprint of the listing for the MONITOR's USE Program.

FIGURE 8

MONITOR "SCRATCHPAD" LISTING

12	0061 177F	ORG	H'1800'-64	TRAINING CARD OS RAM AREA
13	0062	*		
14	0063 17C0	SCDCH RES	8	8 BYTE SCRATCH AREA
15	0064 17C6	TEMP EQU	SCDCH+6	TEMP STORAGE
16	0065 17C8	EAD RES	2	STOP ADDRESS FOR MCAS
17	0066 17CA	BAD RES	2	BEGINNING ADDRESS FOR MCAS
18	0067 17CC	BPD RES	1	DATA TO BE RESTORED IN BREAK LOC
19	0068 17CD	BPL RES	2	ADDRESS OF BREAK POINT LOC
20	0069 17CF	BPF RES	1	BREAK POINT SET FLAG
21	0070 17D0	SSF RES	1	SINGLE STEP SET FLAG
22	0071 17D1	DISBUF RES	8	8 BYTE DISPLAY REGISTER
23	0072 17D9	SAVREG RES	4	A PLACE TO SAVE R0 THRU R3 OF ONE BANK
24	0073 17DD	MEM RES	2	ADDRESS FOR ALTER OR PATCH COMMAND
25	0074 17DF	FID RES	2	FILE ID FLAG AND FILE ID
26	0075 17E1	BCC RES	1	BLOCK CHECK CHAR
27	0076 17E2	BSTI RES	1	SAVE UNITS DIGIT
28	0077 17E3	T RES	2	TEMP REGISTER
29	0078 17E5	T1 RES	1	TEMP REGISTER
30	0079 17E6	T2 RES	1	TEMP REGISTER
31	0080 17E7	T3 RES	1	TEMP REGISTER
32	0081 17E8	LADR RES	2	COPY OF LAST ADDRESS REGISTER
33	0082 17EA	SLADR RES	2	SAVE LOCATION FOR LADR
34	0083 17EC	KFLG RES	2	KBD SCAN FLAGS
35	0084 17EE	RES	1	KBD DEBOUNCE COUNT
36	0085 17EF	ALTF RES	1	DISPLAY AND ALTER FLAG
37	0086 17F0	RESTF RES	1	RESTORE REGISTERS FLAG
38	0087 17F1	IFLG RES	1	INTERUPT INHIBIT FLAG
39	0088 17F2	UREG RES	12	STORAGE FOR USER REGISTERS
40	0089 17FE	POWER RES	2	WHEN POWER ON THESE LOC CONTAIN H'5946'
41	0090			*****
42				

Steps 1 through 3 compare a manually loaded BREAKPOINT address to that stored in SCRATCHPAD.

1. At locations '17CD' and '17CE,' inspect memory contents:

OBSERVATION: loc. '17CD' = __; loc. '17CE' = __.

2. Depress **BKPT**

OBSERVATION: .bp= _ _ _ _ (should compare with data taken in step 1).

3. Set any other BREAKPOINT address (4 digits), then repeat step 1. (Should compare with new address)

PROCEDURE D

BREAKPOINT ADDRESS INSPECTION IN THE MONITOR'S SCRATCHPAD

INTRODUCTION:

When a BKPT is executed, the MONITOR saves the data from the BKPT location in its SCRATCHPAD. Use the following procedure as a model to inspect this data.

1. Set a BREAKPOINT at location '1B.'
2. Referring to Figure 7, jot down the contents of location '1B.'

Observation data in location '1B' = _ _

3. Referring to Figure 8, inspect SCRATCHPAD memory at the location which is used for data to be restored to the BREAKPOINT location.

Observation of display: _ _ _ _ _ (should not compare
address data with observation
taken in step 2)

4. Now, set the PROGRAM COUNTER (**REG** **C**) to address 0.
5. Depress **RUN** and make "CHIPS" entry, then depress any
FUNCTION key. This causes the program "CRAPGAME" to execute
to the BREAKPOINT address. Observation of display:

_ _ _ _ _
address data

6. Now, repeat step 3

Observation of display: _ _ _ _ _
address data

} should compare

DIRECTION:

Go right on to the next page.

PROCEDURE E:INSPECTION OF SAVED REGISTER CONTENTS
IN MDNITOR SCRATCHPADINTRODUCTION:

When the INSTRUCTOR exits from a user program, the CONTENTS of all microprocessor REGISTERS and the PROGRAM STATUS WORD are saved in the SCRATCHPAD memory. Subsequently, it is these locations which are inspected when the operator depresses REG followed by any hex key from 0 to 8. Similarly, changes of data to registers are first entered into the SCRATCHPAD, then moved from SCRATCHPAD to the microprocessor's registers just prior to execution of the next user instruction. The following steps verify this statement.

1. Inspect all of the registers (R0 through R6 and the PSW) (R7 and R8).

Observation of display:

R0 = _ _ R1 = _ _ R2 = _ _ R3 = _ _ R4 = _ _ R5 = _ _

R6 = _ _ PV = _ _ PL = _ _

2. Referring to Figure 8, inspect the first 9 (of 12) locations assigned for STORAGE of USER registers in the SCRATCHPAD.

Observation of display:

(1) _ _ _ _ (4) _ _ _ _ (7) _ _ _ _

(2) _ _ _ _ (5) _ _ _ _ (8) _ _ _ _

(3) _ _ _ _ (6) _ _ _ _ (9) _ _ _ _

3. Starting at the FIRST address you inspected, modify 9 locations of SCRATCHPAD memory as follows. Use MEMORY or FAST PATCH modes to enter data.

(1) '33' (4) '44' (7) FF

(2) '22' (5) '55' (8) 04

(3) '11' (6) '66' (9) 44

4. Now inspect the "Microprocessor's registers" (R0 through R6; PV, PL). Observation: data in these registers _____ (does)(does not) compare.

5. Listen to audio tape for a short commentary and further instructions.

NOTES:

PROCEDURE F:INSPECTION OF MONITOR ROM

INTRODUCTION: This routine determines the condition under which the MONITOR program was accessed. Among other things, this routine checks to see if the POWER ON FLAG (locations '17FE' and '17FF' in SCRATCHPAD memory) is set to H'5946.' If not, a short routine called "INIT" sets this 2-byte value into the locations mentioned.

Perform the following steps, first to inspect, then to attempt modification of routine "BEGin." Then, you'll modify the POWER-ON flag in SCRATCHPAD memory ... and verify indirectly that routine "BEGin" is executed in a later step. For your convenience, a listing of routine "BEGin" is provided in Figure 9 (below).

FIGURE 9

MONITOR'S ROUTINE "BEGin": LISTING

LINE	ADDR	OBJECT	E	SOURCE
	0950	DATA		*****
1	0951		*	
2	0952		*	
3	0953		*	
4	0954		*	*PROGRAM ENTRY ROUTINE
5	0955		*	
6	0956		*	
7	0957		*	*DECIDES HOW REACHED ENTRY POINT OF PROGRAM
8	0958		*	
9	0959		*	*1 POWER ON
10	0960		*	*2 SINGLE STEP
11	0961		*	*3 MONITOR PUSHBUTTON ON KEY BOARD
12	0962		*	*4 BREAKPOINT
13	0963		*	
14	0964		*	
15	0965		*	
16	0966		*	
17	0967		*	*****
18	0968		*	
19	0969	182E 084E	BEG	LODR,R0 PMRON CHECK POWER ON FLAG
20	0970	1830 E459		COMI,R0 H'59' AFTER POWER VALUE OF FLAG IS
21	0971		*	H'5946'
22	0972	1832 9813	BEG1	BCFR,EQ INIT IF NOT CORRECT THIS IS POWER ON
23	0973		*	GO INITIALIZE THE MONITOR FLAGS
24	0974	1834 0849	BEG3	LODR,R0 PMRON+1 CHECK LO BYTE OF POWER ON FLAG
25	0975	1836 E446		COMI,R0 H'46' IS SECOND BYTE CORRECT
26	0976	1838 9800		BCFR,EQ INIT IF NOT INITIALIZE THE PROGRAM
27	0977	183A 0C17D0		LODR,R0 SSF CHECK THE SINGLE STEP FLAG
28	0978	183D 9C19C9		BCFA,EQ SGLSTP IF FLAG THEN GO SINGLE STEP
29	0979	1840 0899		LODR,R0 #MON+1 SEE IF BREAK POINT ENABLED
30	0980	1842 9C197A	BEG2	BCFA,EQ BRKPT GO EXECUTE THE BREAK POINT ROUTINE
31	0981	1845 1B13		BCTR,UN MON MUST BE MONITOR KEY
32	0982		*	

DIRECTION:

Step-by-step sequence on next page.

1. Referring as necessary to Figure 9, employ the "INSTRUCTOR's" memory mode to verify the data in the program entry routine "BEGin"; addresses '182E' through '1846.'

2. Attempt to change data at location '1832' from '98' to 'AA.'
When you depress **E/N**, display = _ _ _ _ _

NOTE: "Error 3" indicates that an attempt to write new data into memory failed.

3. Now depress **MEM** and **E/N**.

Observation of display: _ _ _ _ _

4. From this display, you can surmise that the routine "BEGin" is stored in Read Only Memory (ROM) _____ (true)(false).

5. Referring to Figure 8, modify the PWR ON flag in SCRATCHPAD memory from H'5946' to H'2345.'

6. Verify the modification.

7. Depress **RST** ... to exit from the Monitor to the User program.

8. Depress **MON** ... to re-enter the Monitor Program.

9. Inspect the PWR ON flag in SCRATCHPAD memory.

Observation of display: _ _ _ _ _ **E/N** = _ _ _ _ _

From this display, you can conclude that the PWR ON flag, located in SCRATCHPAD at address _ _ _ _ _ and _ _ _ _ _ , was changed from H'2345' (step 5) to H' _ _ _ _ _ .

10. Listen to the tape for verification and further directions.

NOTES:



2650 MICROPROCESSOR COURSE

MODULE IV - F / G

ADDRESS MODES AND FORMATS DETAILED INSTRUCTION FUNCTIONS

PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086

REFERENCE:

Outlines and Abstracts
pages 14 to 20.

INTRODUCTION:

In Module IV-C, you were provided with an overview of the microprocessor's instruction set functions. The need for various methods of developing addresses was introduced, based on specific opcode and operand requirements for data manipulation and program control.

At this point, most microprocessor-oriented courses describe their respective instruction sets in tremendous theoretical detail. Practical applications involving multi-instruction program sequences are demonstrated only as an after thought.

Within this course, tutorial detail respecting instruction set functions and format implementation is sandwiched between larger layers of useful program analysis and interpretation. In that sense, the formal descriptions you receive will be sometimes of an indirect nature. If at any time you desire a more formal and concise presentation, you need only access the applicable pages in the "2650 Microprocessor or INSTRUCTOR Reference Manuals".

As this module progresses, always ask the question: "Why"? "Why this choice (or instruction)"? "Why this format"? "Could (the application or step) have been accomplished differently"? "If so, how"? Many times you'll hear the answer to these ever-present questions. Often you won't. However, one fact is certain. The degree to which you question now will have a direct bearing on the expertise you demonstrate in design of your microprocessor-driven application.

One more point: You may be tempted to stop at a particular point in the presentation. To experiment; to explore some point which is not clearly understood; to exercise the age-old dream of every inquiring man: "WHAT WOULD HAPPEN IF ..."?

The recommendation is simple. Do it! The "INSTRUCTOR" in front of you is literally in your hands. Control it! Make it work for you! And after you've answered your question, resume the tutorial instruction. Unlike tightly structured lecture/lab seminars, you have complete freedom to set your pace.

A very personal word: It was during the generation of this course that its author often halted formal course material preparation and simply ... played. Many of the results of this playing are incorporated in this version of the course. For example, most of "CRAP-GAME" which you've exercised previously and ... programs such as those to automate relocation of routines in memory, which you'll analyze in this module. It is programs such as these that give this course its flexibility. Later, your participation in developing like programs will enhance the flexibility of your application's control program. And that's what software - programming - is all about!

INTRODUCTION:SYMBOLIC EQUATE TABLES

Throughout all of your programming activities, you'll be specifying REGISTERS, the PROGRAM STATUS WORD, I/O Ports and other Microprocessor functions in symbolic terms. This practice provides easy-to-read documentation; in addition it may be incorporated into more sophisticated programming techniques such as in use of the assembler or other software development tools.

In all cases, symbols such as those listed, must be defined to the assembler prior to their usage in a given program. Once defined through use of suitable EQUATE, ADDRESS CONSTANT or (MEMORY) RESERVATION statements, these symbols will be recognized.

The practice of declaring symbolic definitions in terms of their hex code equivalents is performed with somewhat more freedom in HANDCODING TASKS. Since you are performing the translation from symbolic to hex code on the programming form, you do not have to declare symbols which you use regularly; you'll memorize them. However, it is a good practice to jot down (as a comment, if nothing else) the hex definition of address labels, symbolic address and data names, and constants as you develop them. Then later, if you have access to another software development aid, the process of symbolic definition will be rapid ... and accurate.

Table 1 (next page) is a reprint of the INSTRUCTOR's listing of symbolic equates. You'll find it useful to detach this page and post it in a clearly-readable position for the rest of this course. By making use of the equates contained in Table 1, you'll be standardizing the most commonly used operand symbols in today's microprocessor technology

DIRECTION: Go right on to the next page.

TABLE 1SYMBOLIC EQUATE TABLES

TWIN ASSEMBLER-VER 2.0 TRAINING CARD 05 27 MAY 77 PAGE 0012

LINE ADDR OBJECT E SOURCE

```

0383 *****
0384 *   EQUATE TABLES
0385 *
0386 *   REGISTER EQUATES
0387 0000 R0   EQU   0   REGISTER 0
0388 0001 R1   EQU   1   REGISTER 1
0389 0002 R2   EQU   2   REGISTER 2
0390 0003 R3   EQU   3   REGISTER 3
0391 *   CONDITION CODES
0392 0001 P    EQU   1   POSITIVE RESULT
0393 0000 Z    EQU   0   ZERO RESULT
0394 0002 NG   EQU   2   NEGATIVE RESULT
0395 0002 LT   EQU   2   LESS THAN
0396 0000 EQ   EQU   0   EQUAL TO
0397 0001 GT   EQU   1   GREATER THAN
0398 0003 UN   EQU   3   UNCONDITIONAL
0399 *   PSW LOWER EQUATES
0400 00C0 CC   EQU   H'C0'  CONDITIONAL CODES
0401 0020 IDC  EQU   H'20'  INTERDIGIT CARRY
0402 0010 RS   EQU   H'10'  REGISTER BANK
0403 0008 WC   EQU   H'08'  1=WITH 0=WITHOUT CARRY
0404 0004 OVF  EQU   H'04'  OVERFLOW
0405 0002 COM  EQU   H'02'  1=LOGIC 0=ARITHMETIC COMPARE
0406 0001 C    EQU   H'01'  CARRY/BORROW
0407 *   PSW UPPER EQUATES
0408 0080 SENS  EQU   H'80'  SENSE BIT
0409 0040 FLAG  EQU   H'40'  FLAG BIT
0410 0020 II   EQU   H'20'  INTERRUPT INHIBIT
0411 0007 SP   EQU   H'07'  STACK POINTER
0412 *   IO PORT DEFINITIONS
0413 17FF UMEM  EQU   H'17FF' USER INTERNAL MEMORY MAPPED IO LOCATION
0414 0007 LED5  EQU   H'07'  USER INTERNAL EXTENDED IO PORT
0415 0017 LED51 EQU   H'17'  USER EXTERNAL EXTENDED IO PORT
0416 *   INTERRUPT VECTORS
0417 0007 UINTV EQU   H'07'  USER DIRECT INTERRUPT VECTOR
0418 0087 UINTVI EQU  H'87'  USER INDIRECT INTERRUPT VECTOR
0419 *

```

DIRECTION:

Go right on to the next page.

During your studies of CRAPGAME program documentation in Modules 1 and 2, you may have noticed symbolic instruction entries as illustrated below.

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1					MESLOC	RES	1	
2					RLDICE	ACON	H'01B1	
3					ROLL	RES	8	
4					DICETB	RES	37	
5					DEDCOR	ACON	PAS1XP	
6								

"RES" and "ACON" are directives to the ASSEMBLER, not the micro-processor.

ACON means ADDRESS CONSTANT. For example, the address constant (start address) for "RLDICE", a subroutine in "CRAPGAME", is H'01B1'.

RES means 'reserve a block of memory, N bytes long, for the DATA table defined symbolically in the label field'. N is defined in the symbolic operand field and is always a decimal number. For example, 1 byte in memory is reserved for "MESLOC". 37 bytes are reserved for the dice table accessed by routine "RLDICE" in CRAPGAME. An address constant could also have been formally assigned to "DICETB". It would be written DICETB ACON H'01DB'. In summary, the RES and ACON statements define to the assembler the exact length and start address of a given label.

In short, in hand-coded programs such as you'll prepare with the "INSTRUCTOR", RES, ACON, and EQ (symbolic equate) statements are not translated into absolute code. However, their use tends to strengthen their associated program documentation. If later you have access to a microprocessor assembler, you'll be able to generate accurate RES, ACON, and EQ statements required by the assembler. In summary, each symbolic label, field, operand, or address must be defined to an assembler before it is employed in an executable program. Since this course does not involve assembler usage, symbols are only partially defined in the interest of saving space.

DIRECTION:

Go right on to the next page.

NOTE: The following series of exercise is self-paced. Complete each exercise in order and listen to the taped commentary as directed. Generally, the exercises detail each address mode and variation in sequence. Details as to the operation of the different instructions are described as the exercises proceed.

EXERCISE 1**REGISTER-ADDRESSED INSTRUCTIONS-FORMAT Z****INTRODUCTION:**

In the first of several such exercises, a detailed examination is made of 1 byte - register addressed (Format Z) instructions. There are two major advantages to the use of Format Z instructions; these are:

- a. LENGTH ... Since Format Z instructions are 1 byte, better memory conservation (more instructions per unit of control storage) is possible.
- b. TIMING ... All Format Z instructions require only 2 machine (instruction execution) cycles, e.g.: 6 clock periods, to execute. Other instructions require 3, 4 or more machine cycles.

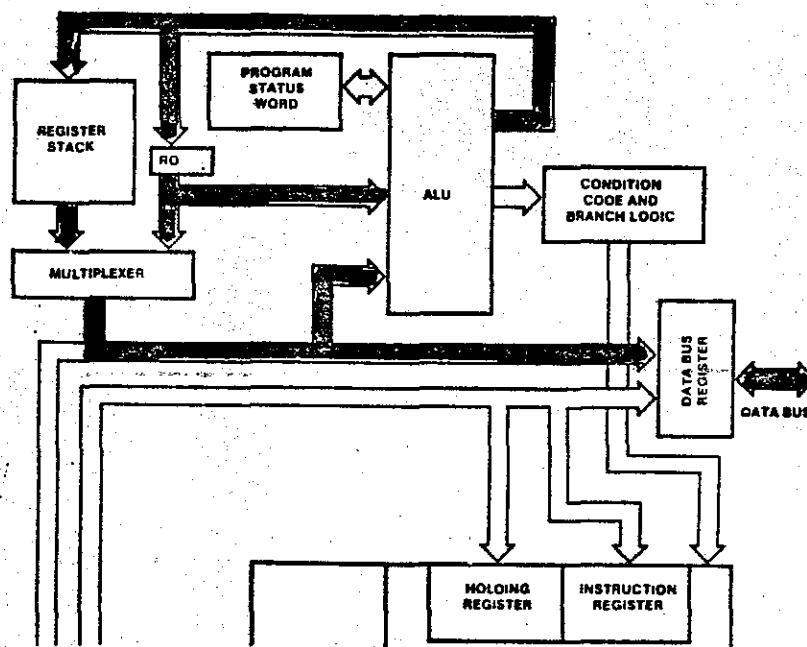
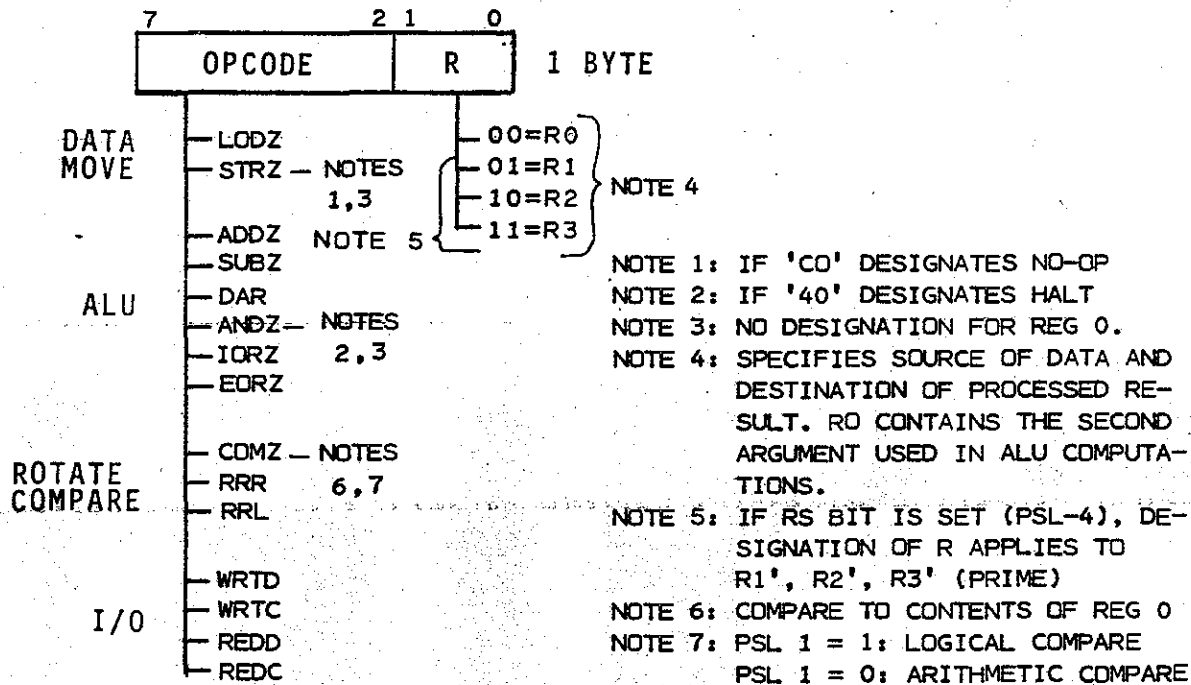
However, there is a major disadvantage. Since each OPCODE (examined in Module IV-C) requires a minimum of 6 bits for definition, only 2 bits are available for OPERAND definition. Thus, except for selection of non-extended ports C and D, external I/O may not be accessed in Format Z. Address of any location in memory (even address 0) is ruled out. And, where 2 operands must be specified to give definition to the OPCODE, one of them is implicitly R0. For example, data transfer between R2 and R3 is not possible. Two instructions (one to move data from R2 to R0, the second to store data from R0 to R3) are necessary.

DIRECTIONS:

1. Take a few minutes to study Figure 1: "REGISTER ADDRESSING-FORMAT Z."
2. Relate each symbolic instruction opcode (mnemonic) to its hex-code equivalent on the 2650 Programming Reference Guide (Hard Card).
3. Listen to tape 5B for a brief commentary on Figure 1 *(next page).
4. Complete the procedural steps for this exercise on page 7.

*Refer to the 2650 REFERENCE MANUAL for the formal description of each instruction.

REGISTER ADDRESSING - FORMAT Z

**DIRECTION:**

Listen to tape 5B for a brief explanation of material on this page, then complete the procedure steps which follow.

PROCEDURE:

- Using the "INSTRUCTOR's" DISPLAY and ALTER REGISTERS command, store the following data in Registers 0 through 3. Then set the PARALLEL I/O SELECTION SWITCH to the NON EXTENDED position.

REG	DATA	REG	DATA	
R0	33	R4	00	=(R1')
R1	40	R5	00	=(R2')
R2	5D	R6	00	=(R3')
R3	0F			

- In this step, use the programming form on page 8 to code and load Format Z instructions into memory, starting at address '0'. This sequence manipulates and moves data as indicated by the instructions. Use the "hardcard" or Reference Manual as informational sources. Do NOT execute the program until instructed to do so.

NOTE: At various points, you'll be selecting the second (PRIME) bank of working registers (R2', R2', R3'). The codes for register bank selection are as follows:

CODE SYMBOLIC
OPCODE OPERAND

77 10 PPSL RS ... to select R1', R2', R3'
75 10 CPSL RS ... to select R1, R2, R3

In addition, the WC bit in the Program Status Word (Lower) is cleared with the code '75'18' in line 3. To be explained later in the module.

- Set the Program Counter to address 0. Execute program 'FORMZ' in STEP mode. Complete Table 1 as indicated. At your option, compare your results to those provided on page 9 of this module.

TABLE 1 "FORMZ" OBSERVATIONS								
AFTER EXECUTING THE INSTRUCTION AT ADDRESS:	INSPECT. THEN RECORD THE CONTENTS OF IN- DICATION REGISTERS IN THIS TABLE.							
	R0	R1	R2	R3	R1'	R2'	R3'	
0 (START)	33	40	5D	0F	00	00	00	
0002		XX	XX	XX		XX	XX	
0005		XX	XX	XX	XX	XX	XX	
0006		XX	XX	XX	XX	XX	XX	
0007		XX	XX	XX	XX	XX	XX	
000A		XX	XX	XX	XX		XX	
000B		XX	XX	XX		XX	XX	
0010				XX		XX	XX	
0011	XX		XX	XX	XX	XX	XX	
0012		XX	XX	XX	XX		XX	
0013		XX	XX		XX	XX	XX	
0016					XX	XX		
001C	XX	XX	XX		XX	XX	XX	
001D	XX	XX	XX		XX	XX	XX	
001F	XX	XX	XX		XX	XX	XX	

DIRECT RELATIVE ADDRESSING - SECOND BYTE															
+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E
N	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
-	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71
+	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E
N	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
-	70	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E
N	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46
-	60	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52
+	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E
N	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
-	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42

INDIRECT RELATIVE ADDRESS: Add H'80' to DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE FORM Z START ADDR 0DESCRIPTION Exercise to investigate
the effect of executing Format Z

INSTRUCTIONS (all types)

I/O SELECTION SW: "NON EXT DATA PORT" Pos.

ROUTINE SHEET 1 OF 1MEMORY LOCATIONS THIS SHEET 21

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	0000	77	10		FORMZ	PSSL	R5	Select prime regs & move
2						STRZ	R1	R0 → R1. Select non prime
3		75	18			CPSL	R5, WC	regs and clear WC. Now
4						ADDZ	R1	Add R1 + R0 then sub-
5						SUBZ	R2	tract R2 from result &
6						IORZ	R3	Incl. or with R3. Then
7						PSSL	R5	store the result in R2'
8						STRZ	R2	and fetch R1 (original
9						LODZ	R1	R0 data) back into R0.
10		75	10			CPSL	R5	Then re-select non prime
11						EORZ	R2	regs. Now, excl OR R2 w
12						STRZ	R2	R0; result → R2. Then
13						EORZ	R0	clean R0...
14						STRZ	R1	... and R1, then fetch
15						LODZ	R2	R2 → R0 and MASK with
16						ANDZ	R3	R3 contents.
17						PSSL	R5	Select R3' and store
18						STRZ	R3	result in R3'. Then re-
19						CPSL	R5	turn to non prime regs.
20						WRTD	R0	Now, show contents of
21						WRTD	R1	R0, R1, R2, and R3 on
22						WRTD	R2	parallel I/O LEDs
23						WRTD	R3	... then
24						RRL	R3	shift R3 1 bit left &
25						WRTD	R3	display.
26						RRR	R3	Shift R3 right 1 bit and
27						WRTD	R3	display. End of FORMZ

SUGGESTED CODE - PROGRAM "FORMZ"

	ADDRES	DATA			LABEL	SYMBOLIC INSTRUCTION	
		B0	B1	B2		OPCODE	OPERANDS
1	0000	77	10		FORMZ	PPSL	R5
2	2	C1				STRZ	R1
3	3	75	18			CPSL	R5, WC
4	5	81				ADDZ	R1
5	6	A2				SUBZ	R2
6	7	63				IORZ	R3
7	8	77	10			PPSL	R5
8	A	C2				STRZ	R2
9	B	01				LODZ	R1
10	C	75	10			CPSL	R5
11	E	22				EORZ	R2
12	0F	C2				STRZ	R2
13	10	20				EORZ	R0
14	1	C1				STRZ	R1
15	2	02				LODZ	R2
16	3	43				ANDZ	R3
17	4	77	10			PPSL	R5
18	6	C3				STRZ	R3
19	7	75	10			CPSL	R5
20	9	F0				WRTD	R0
21	A	F1				WRTD	R1
22	B	F2				WRTD	R2
23	C	F3				WRTD	R3
24	D	D3				RRL	R3
25	E	F3				WRTD	R3
26	1F	53				RRR	R3
27	0020	F3				WRTD	R3

SUGGESTED SOLUTION
TABLE 1 OBSERVATIONS:

TABLE 1 "FORMZ" OBSERVATIONS

AFTER EXECUTING THE INSTRUCTION AT ADDRESS:	INSPECT, THEN RECORD THE CONTENTS OF IN- DICATION REGISTERS IN THIS TABLE.						
	R0	R1	R2	R3	R1'	R2'	R3'
0 (START)	33	40	5D	0F	00	00	00
0002	33	XX	XX	XX	33	XX	XX
0005	73	XX	XX	XX	XX	XX	XX
0006	16	XX	XX	XX	XX	XX	XX
0007	1F	XX	XX	XX	XX	XX	XX
000A	1F	XX	XX	XX	XX	1F	XX
000B	33	XX	XX	XX	33	XX	XX
0010	00	40	6E	XX	33	XX	XX
0011	XX	00	XX	XX	XX	XX	XX
0012	6E	XX	XX	XX	XX	1F	XX
0013	0E	XX	XX	0F	XX	XX	XX
0016	0E	00	6E	0F	XX	XX	0E
001C	XX	XX	XX	0F	XX	XX	XX
001D	XX	XX	XX	1E	XX	XX	XX
001F	XX	XX	XX	0F	XX	XX	XX

DIRECTION:

After comparing your answers with those at the left, go on to the next page of this module.

DIRECTION:

If you are not familiar with arithmetic and logical functions which can be performed by the microprocessor's ALU, material on this page will serve as a precise summary, with detail provided to a bit level. If you are familiar with arithmetic and logical data manipulation, go immediately to page 14.

SYMBOLIC INSTRUCTION	CODE	ALGORITHM PERFORMED	DIAGRAMMED DATA MANIPULATION
LODZ R1 R2 R3	01 02 03	(R1) → (R0) IMPLICIT (R2) → (R0) SOURCE (R3) → (R0) REG DEST. R0	DATA MOVED FROM OPERAND-SPECIFIED REGISTER INTO R0 WITHOUT ALTERATION. LODZ R0 1'00' YIELDS INDETERMINATE RESULTS.
STRZ R1 R2 R3	C1 C2 C3	(R0) → (R1) IMPLICIT (R0) → (R2) SOURCE (R0) → (R3) R0 DEST. REG	DATA MOVED FROM R0 INTO OPERAND-SPECIFIED REGISTER WITHOUT ALTERATION. NO VALID CODE TO MOVE R0 INTO ITSELF CODE 'C0' SPECIFIES A NOP (NO OPERATION)
ADDZ R0 R1 R2 R3	80 81 82 83	(R0) + (R0) → (R0) ... AND CARRY (R1) + (R0) → (R0) BIT INTO (R2) + (R0) → (R0) PSL IF WC (R3) + (R0) → (R0) SPECI- IMPLI- IMPLICIT FIED C1T DESTINATION SOURCE 2ND REG REG SOURCE reg	EXAMPLE R2 = 'C3' = 11000011 R0 = '67' = 01100111 RESULT: 11000011 + 01100111 = 00101010 = '2A' CARRY REPLACES CONTENTS OF R0. STORED IN PSL; BIT 8 IF WC ACTIVE DROPPED IF WC NOT ACTIVE.
SUBZ R0 R1 R2 R3	A0 A1 A2 A3	(R0) - (R0) → (R0) ... AND BORROW (R0) - (R1) → (R0) (CARRY) BIT (R0) - (R2) → (R0) IN PSL IF (R0) - (R3) → (R0) WC ACTIVE IMPLI- SPECI- IMPLICIT C1T FIED DEST SOURCE SOURCE REG REG REG NOTE: FURTHER DETAIL IN REF. MANUAL.	EXAMPLE R3 = '38' = 00111000 11000111 ONE'S COMPL. 1 PLUS 1 FOR 11001000 2'S COMPL. R0 = '65' = 01100101 RESULT: 01100101 - 00111000 = 01101101 = '2D' BORROW REPLACES CONTENTS OF R0 STORED IN PSL BIT 8. IF ACTIVE, SIGN OF RESULT IS POSITIVE.
ANDZ R1 R2 R3	41 42 43	(R1) & (R0) → (R0) ... WHERE & (R2) & (R0) → (R0) EQUALS (R3) & (R0) → (R0) LOGICAL SPECI- IMPLI- IMPLICIT and FIED C1T DEST SOURCE 2ND REG REG SOURCE	EXAMPLE R1 = '50' = 01011101 R0 = '66' = 01100110 RESULT: 01011101 & 01100110 = 01000100 = '44' REPLACES CONTENTS OF R0 CODE 40 SPECIFIES A 'HALT' OPERATION.
IORZ R0 R1 R2 R3	60 61 62 63	(R0) V (R0) → (R0) ... WHERE V (R1) V (R0) → (R0) EQUALS (R2) V (R0) → (R0) LOGICAL (R3) V (R0) → (R0) INCLUSIVE SPECI- IMPLI- IMPLICIT or FIED C1T DEST SOURCE 2ND REG REG SOURCE	EXAMPLE R2 = '34' = 00110100 R0 = '61' = 01100001 RESULT: 00110100 V 01100001 = 01110101 = '75' REPLACES CONTENTS OF R0
EORZ R0 R1 R2 R3	20 21 22 23	(R0) ⊕ (R0) → (R0) ... WHERE ⊕ (R1) ⊕ (R0) → (R0) EQUALS THE (R2) ⊕ (R0) → (R0) LOGICAL (R3) ⊕ (R0) → (R0) EXCLUSIVE SPECI- IMPLI- IMPLICIT OR FIED C1T DEST SOURCE SOURCE REG REG REG	EXAMPLE R3 = 'E6' = 11100110 R0 = '55' = 01010101 RESULT: 11100110 ⊕ 01010101 = 10110011 = 'B3' REPLACES CONTENTS OF R0 EXAMPLE: R2 = 3A = 00111010 R0 = FF = 11111111 RESULT: 00111010 ⊕ 11111111 = 11000101 = C5 = EQUALS ... EQUALS ONE'S COMPLEMENT OF 3A

DIRECTION:

The following problems are designed to reinforce your understanding of ARITHMETIC and LOGICAL functions performed by the microprocessor. Using the 2650 Programming Reference (HARD) card, code and execute the following algorithms. Refer to the 2650 Reference Manual and page 10 of this documentation for further detailed information. Load your code into any user memory ('00' - '1FF') locations in the INSTRUCTOR and execute. Problem 1 is fully detailed for example purposes.

1. Given R2 = '3F' and R0 = '2A', perform the algorithm: R2+R0.
Prediction: After execution of the instruction, the value residing in R0 will be '69'.

Solution:

Depress **REG** **8** **0** **8** To set PSL WC bit (explained later).

Depress **REG** **0** **2** **A** **E/N** } To enter data into the specified registers.
E/N **3** **F**

Depress **MEM** **1** **0** **0** **E/N** .. To select a start address to load the algorithm performing instruction into memory ...

Code the instruction ADDZ R2, then

Depress **8** **2** **E/N** And load the instruction code for ADDZ R2; ('82'), then:

Depress **REG** **C** **1** **0** **0** ... To set the Program Counter to a selected START ADDRESS. Then:

Depress **STEP** Execute the single instruction program ... and

Depress **REG** **0** Inspect Register 0

Observe of Hex Display: **.R0 = 69** ... Should compare with prediction.

2. Given R3 = '12' and R0 = 'F2', perform the algorithm: R0R3
Prediction: R0 = '___'.
3. Given R1 = '4B' and R0 = '8C', perform the algorithm: R0 + R1
Prediction: R0 = '___'.
4. Given R2 = '3A' and R0 = '90', perform the algorithm R0VR2.
Prediction: R0 = '___'.
5. Given R1 = '0F' and R2 = 'E6', perform the algorithm R1 ⊕ R2.
Prediction: R2 = '___'. Note: This sequence requires 3 instructions; all Z format.
6. Given R0 = 'DC' and 'R3' = '85', perform the algorithm R0-R3.
Prediction: R0 = '___'.
7. Given R3 = 'FF' and R0 = '33', perform the algorithm R0 ⊕ R3,
Prediction: R0 = '___'.

- NOTE:**

PREDICTION:

9. Given $R2 = '46'$, and $R0 = '84'$, perform the algorithm $R0 \vee R2$.

PREDICTION: RO = ' '.

DIRECTION:

Suggested solutions to these problems are provided on the next page.

INTRODUCTION

In this first of 2 optional lessons, the relationship between SOFTWARE and HARDWARE is described, using an analogy which draws on well known musical relationships.

DIRECTIONS:

1. Listen to tape 4A - "INTRODUCTION TO SOFTWARE," (about 13 minutes).
2. Examine the "HARDWARE/SOFTWARE FUNCTIONAL COMPARISON CHARTS" on the next two pages.
3. A glossary of terms, definitions and "buswords" commonly employed by those associated with the microprocessor industry is contained in pages 4 through 29 of this section. You should ensure your understanding of these terms since the following modules of this course constantly implement their usage.
4. A short self administered quiz is provided on page 30. Complete its requirements, then go on to Module III-B (optional) or Module IV: 2650 Instruction Repertoire.

HARDWARE/SOFTWARE FUNCTIONAL COMPARISON

PAGE 1

FUNCTION COMPARED		TO HARDWARE		TO SOFTWARE		ANALOGY TO MUSIC	
AUTHOR		SYSTEMS LOGIC DESIGNER		PROGRAMMER		COMPOSER	
GUIDELINE		OBJECTIVE SPECIFICATION FUNCTIONAL SPECIFICATION DEFINE PROBABLE SUPPORT DEVICES		OBJECTIVE SPECIFICATION FLOWCHARTS DEFINE DEPENDENT VARIABLES		OUTLINE OF MUSICAL PASSAGES. SPECIFICATION OF INSTRUMENTS	
OVERALL CONTROL		WHAT APPLICATION WILL DO WHAT IT WILL LOOK LIKE HOW IT WILL PERFORM		DIRECTION AND CONTROL OF HARDWARE		WHAT FINAL RENDITION WILL SOUND LIKE	
APPROACH AND IMPLEMENTATION OF DESIGN		COMPONENT DATA SHEETS SELECTION GUIDED BY OBJECTIVE SPECIFICATION		INSTRUCTION SET GUIDED BY OBJECTIVE SPEC. AND KNOWLEDGE OF CONTROLLED DEVICE CAPABILITIES		KNOWLEDGE OF EACH INSTRUMENTS CAPABILITIES GUIDED BY OUTLINE	
LANGUAGE		HARDWARE AND LOGIC SYMBOLS CONNECTING LINES, INTERFACING RULES, LOADING FACTORS		PROGRAMMING TERMS, MNEMONICS OPCODES, OPERANDS, SYNTAX, BIT DEFINITION, CONSTANTS, FIELDS		NOTES, DYNAMICS, RHYTHM, VOLUME TONE, RULES OF MUSIC SYNTAX	
MAJOR FUNCTION		SPECIFY SYSTEMS COMPONENTS LAYOUT SCHEMATICS ESTABLISH INTERFACES		WRITE THE PROGRAM INSTRUCTION SEQUENCE IN SOURCE CODE. DOCUMENT PROGRAM FLOW		WRITE ORIGINAL MUSIC ON PAPER	
GENERAL PURPOSE COMPUTER		INCLUDES CPU AND MANY DEDICATED SUPPORT PERIPHERAL DEVICES		POWERFUL AND FLEXIBLE INSTRUCTION SET CAN BE IMPLEMENTED IN MOST APPLICATIONS		MAJOR SYMPHONY ORCHESTRA EXTENSIVE REPERTOIRE	
SPECIAL PURPOSE COMPUTER		HARDWARE NOT SOPHISTICATED LIMITED APPLICATION. FUNCTIONS MAXIMIZED IN APPLICATION DESIGN		LIMITED INSTRUCTION SET MAY REQUIRE MORE SOPHISTICATED PROGRAMMING		ROCK GROUP, STRING QUARTET JAZZ ENSEMBLE LIMITED REPERTOIRE	
TIMING		EXTERNAL AND INTERNAL CLOCK AND TIMING CIRCUITS		INSTRUCTION CYCLE		CONDUCTOR SETS BEAT, TEMPO	

HARDWARE/SOFTWARE FUNCTIONAL COMPARISON

PAGE 2

FUNCTION COMPARED		TO HARDWARE		TO SOFTWARE		ANALOGY TO MUSIC	
CONTROL STORAGE MEMORY	ROM OR PROM	MEMORY MAP FOR PROGRAM SEGMENTS	BOOK OF PRINTED SHEET MUSIC				
WORKING STORAGE	RAM	SCRATCH PAD	NO EQUIVALENT				
CONTROLLER I/O DEVICES	I/O PORTS HARDWARE INVOLVED IN INTERFACE, CONTROL, AND TIMING	SYMBOLIC DESIGNATION SOFTWARE INVOLVED IN ADDRESS, CONTROL, AND TIMING	MUSICIANS AND INSTRUMENTS; CAPABILITIES AND LIMITA- TIONS				
ERROR CORRECTION	TROUBLE SHOOTING FUNCTIONAL TESTING DIAGNOSE AND CORRECT HARDWARE PROBLEM	DEBUG PROGRAM, SIMULATE FIRST, THEN INTEGRATE WITH CONTROLLED HARDWARE	TRY OUT COMPOSITION ON PIANO WORK IN OTHER PARTS AND CORRECT MUSICAL SCORE				
DOCUMENT GENERATION	FINAL DRAFT OPERATIONS MANUALS, SCHEMATICS, ASSEMBLY DRAWINGS, SYSTEMS CONFIGURATION ASSISTED BY ARTISTS, ETC.	TRANSLATE SOURCE CODE TO OB- JECT CODE, ASSISTED BY ASSEMBLER, GENER- ATE PROGRAM TAPES	FINALIZE COMPOSITION HAVE IT PREPARED FOR PUBLISHING				
PACKAGING	INSTALLATION PLANNING CHASSIS OR BOX LAYOUT POWER REQUIREMENTS, COOLING, ETC.	GENERATION OF COMPLETE DOCU- MENTATION INCLUDING SOURCE AND OBJECT CODE LISTINGS, DEFINI- TIONS TABLES, ETC.	FUNCTIONS PERFORMED BY STAGE MANAGERS, SYMPHONY HALL DE- SIGN, AUDIO AND LIGHT ENGINEERS, ETC.				
DISSEMINATION	PUBLICATION AND DISTRIBUTION OF DOCUMENTS PRODUCTION AND DISTRIBUTION OF APPLICATION	DUPPLICATION OF PROGRAM IN ROM OR PROM INSTALLATION IN CONTROLLED APPLICATION	PUBLICATION AND DISTRIBUTION OF SHEET MUSIC TO ORCHESTRAS AS ORDERED				
RESULT	DESIRED APPLICATION FABRICATED AND IN USE		RENDITION OF SYMPHONIC MASTERPIECE				

GLOSSARY OF MICROPROCESSOR TERMS

A

ABBREVIATED ADDRESSING:

A modification of the Direct Address mode which uses only part of the full address and provides a faster means of processing data because of the shortened code.

ACCUMULATOR:

One or more registers associated with the ALU which temporarily store sums and other arithmetical and logical results of the ALU.

ADAPTER:

A device used to effect operative capability between different parts of one or more systems or subsystems.

ADDRESSING MODES:

An address is a coded instruction designating the location of data or program segments in storage. The address may refer to storage in registers or memories or both. The address code itself may be stored so that a location may contain the address of data rather than the data itself. This form of addressing is common in microprocessors. Addressing modes vary considerably because of efforts to reduce program execution time.

ALU (ARITHMETIC AND LOGIC UNIT):

The ALU is one of the three essential components of a microprocessor, the other two being the registers and the control block. The ALU performs various forms of addition and subtraction; the logic mode performs such logic operations as ANDing the contents of two registers, or masking the contents of a register.

ARCHITECTURE:

Any design or orderly arrangement perceived by man; the architecture of the microprocessor. Since the extant microprocessors vary considerably in design, their architecture has become a bone of contention among specialists.

ASSEMBLER
ASSEMBLER PROGRAM

Computer program whose purpose is to translate easily-read symbolic instructions into object code.

ASSEMBLY LANGUAGE:

A machine oriented language. Normally the program is written as a series of source statements using mnemonic symbols that suggest the definition of the instruction and is then translated into machine language.

ASSIGNMENT:

Designation of registers, selected I/O or working storage locations for specific purposes.

ASYNCHRONOUS:

Operation of a switching network by a free-running signal which signals successive instructions, the completion of one instruction triggering the next. There is no fixed time per cycle.

B**BAUD RATE:**

A measure of data flow. The number of signal elements per second based on the duration of the shortest element. When each element carries one bit, the Baud rate is numerically equal to bits per second (bps). The Baud rates on UART data sheets are interchangeable with bps.

BCD (BINARY CODED DECIMAL):

Each decimal digit is binary coded into 4-bit words. The decimal number 11 would become 0001 0001 in BCD. Also known as the 8421 code.

BENCHMARK:

Originally a surveyor's mark used as a reference point in surveys. In connection with microprocessors, the benchmark is a frequently used routine or program selected for the purpose of comparing different makes of microprocessors. A flow chart in assembly language is written out for each microprocessor and the execution of the benchmark by each unit is evaluated on paper. It is not necessary to use hardware to measure capability by benchmark.

BIDIRECTIONAL:

A term applied to a port or bus line that can be used to transfer data in either direction.

BINARY:

A system of numbers using 2 as a base in contrast to the decimal system which uses 10 as a base. The binary system requires only two symbols, 0 and 1. Two is expressed in binary by the number 10 (read one, zero). Each digit after the initial 1 is multiplied by the base 2. Hence the following table expresses the first ten numbers in decimal and binary:

DECIMAL	BINARY
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

BRANCH:

Refers to the capability of a microprocessor to modify the function or program sequence. Such modification depends on the actual content of the data being processed at any given instant.

BREAKPOINT:

A program point indicated by a breakpoint flag which invites interruption to give the user the opportunity to check his program before continuing to its completion.

BUFFER:

A circuit inserted between other circuit elements to prevent interactions, to match impedances, to supply additional drive capability, or to delay rate of information flow. Buffers may be inverting or non-inverting.

BUS DRIVER:

An integrated circuit which is added to the data bus system to facilitate proper levels to the CPU when several memories are tied to the data bus line. These are necessary because of capacitive loading which slows down the data rate and prevents proper time sequencing of microprocessor operations.

BUS SYSTEM:

A set of paths (usually wires) that provide a communications path in a system. The important busses are the Data Bus, Control Bus, and the Address Bus. They communicate information between the microprocessor and controlled devices.

BYTE:

Indicates a pre-determined number of consecutive bits treated as an entity. For example, 4-bit or 8-bit bytes. "Word" and "Byte" are used interchangeably.

C**CLOCK:**

A generator of pulses which controls the timing of switching circuits in a microprocessor. Clock frequency is not the only criterion of data manipulation speed. Hardware architecture and programming skill are more important. Clocks are a requisite for most microprocessors and multiple phased clocks are common in MOS processors.

COMBINATIONAL LOGIC:

A circuit arrangement in which the output state is determined by the present state of the input. Also called Combinatorial Logic. (See also Sequential Logic.)

COMPILERS:

Compilers translate higher-level languages into machine codes.

CONDITION CODE:

Refers to a limited group of program conditions such as carry, borrow, overflow, etc., which are pertinent to the execution of instructions. The codes are contained in a Condition Code Register.

CONSTANTS

In conversational language or in Assembler source statements, constants are written in decimal. Constants are coded in binary within immediate operands.

CONTROL BLOCK:

This is the circuitry which performs the control functions of the CPU. It is responsible for decoding microprogrammed instructions and then generating the internal control signals that perform the operations requested.

CONTROL BUS:

Conveys a series of signals which regulate system operation. These "traffic" signals are commands which may also originate in peripherals for transfer to the CPU or the reverse.

CONTROL PROGRAM:

The control Program is a sequence of instructions that will guide the CPU through the various operations it must perform. This program is stored permanently in ROM memory where it can be accessed by the CPU during operations.

CPU (CENTRAL PROCESSING UNIT):

The heart of any computer system. Basically the CPU is made up of storage elements called registers, computational circuits in the ALU, and the Control Block, and I/O. As soon as LSI technology was able to build a CPU on an IC chip, the microprocessor became a reality. One-chip microprocessors have limited storage space, so memory implementation is added in modular fashion. Most current microcomputers consist of a set of chips, one or two of which form the CPU.

CROM (CONTROL READ ONLY MEMORY):

A part of the control block of some microprocessors, this ROM is microprogrammed to provide instruction decode and control logic functions.

CROSS-ASSEMBLER:

When the program is assembled by the microprocessor that it will run on, the program that performs the assembly is referred to simply as an assembler. If the program is assembled by some other microprocessor or computer, the process is referred to as crossassembly. Occasionally the phrase "native assembler" will be used to distinguish it from a crossassembler.

D

DAISY CHAIN:

A bus line which is interconnected with units in such a way that the signal passes from one unit to the next in serial fashion. The architecture of the Fairchild F-8 provides an example of daisy-chained memory chips. Each chip connects to its neighbors to accomplish daisy-chaining of interrupt priorities beginning with the chip closest to the CPU.

DATA BUS:

The microprocessor communicates internally and externally by means of the data bus. It is bidirectional, and can transfer data to and from the CPU, memory storage, and peripheral devices.

DATA COUNTER:

*See below.

DEBUG:

As used in connection with microprocessor software, debugging involves searching for, and eliminating sources of error in, programming routines. Finding a bug in software routine is said to be as difficult as finding a needle in the proverbial haystack. A single step tester is the suggested method, so that each instruction operation can be checked individually.

DECREMENT:

A programming instruction which decreases the contents of a storage location. (See also increment.)

DEDICATED:

To set apart for some special use. A dedicated microprocessor is one that has been specifically programmed for a single application such as weight measurement by scale, traffic light control, etc. ROMs by their very nature (Read-Only) are "dedicated" memories.

* The Data counter is a processor register which contains a memory address. The data counter, which is referenced by instructions, provides information on the location of specific data in memory. Also called OPERAND ADDRESS REGISTER.

DIRECT ADDRESSING:

This is the standard addressing mode. It is characterized by an ability to reach any point in main storage directly. In Direct address mode, the contents of the addressed location are the specific operand.

DISPLACEMENT
(SET)

Refers to difference in address locations between "this address" and some other memory address, however referenced in the program.

Refers to difference in address locations between the current instruction address and the next effective address.

The displacement is negative (-) if the absolute address of the current instruction is greater than the next effective address.

Displacement is positive (+) if the absolute address of the current instruction is less than the next effective address.

Applies particularly to relative addressing.

DMA (DIRECT MEMORY ACCESS):

A method of gaining direct access to main storage to achieve data transfer without involving the CPU. The manner in which CPU is disabled while DMA is in progress differs in different models. Some use several methods to accomplish DMA.

E

EXECUTION TIME:

Usually expressed in clock cycles necessary to carry out an instruction. Since the clock frequency is known, the actual time can be calculated. Clock frequencies can be varied.

EXTENDED ADDRESSING:

Refers to an addressing mode that can reach any place in memory. (See also Direct Addressing.)

F

FETCH:

To go after and return with things. In a microprocessor, the "objects" fetched are instructions which are entered in the instruction register. The next, or a later step in the program, will cause the machine to execute what it was programmed to do with the fetched instructions. Often referred to as an "instruction fetch."

FIELDS:

A source statement is made up of a number of code fields, usually four, which are acceptable by the assembler. The four fields may connote Label, Operator, Operand, and Comment. Fields are also applicable to data storage. The eight bits stored in a memory location might contain two 4-bit fields, or eight 1-bit fields, etc.

FIRMWARE:

Software instructions which have been permanently frozen into a ROM are sometimes referred to as Firmware.

FLAG BIT:

An information bit which indicates some form of demarcation has been reached such as overflow or carry. Also an indicator of special conditions such as interrupts.

FLOW CHART OR FLOW
DIAGRAM:

A sequence of operations charted with the aid of symbols, diagrams, or other representations to indicate an executive program. Flowcharts enable the designer to visualize the procedure necessary for each item on the program. A complete flowchart leads directly to the final code.

G

GIVEN:

Existing parameters such as contents of selected I/O W.S., and registers, or pertinent instruction addresses, written in octal.

H

HANDSHAKING:

A colloquial term which describes the method used by a Modem to establish contact with another Modem at the other end of a telephone line. Often used interchangeably with buffering and interfacing, but with a fine line of difference in which handshaking implies a direct package to package connection regardless of functional circuitry.*

HARDWARE:

The individual components of a circuit, both passive and active have long been characterized as hardware in the jargon of the engineer. Today, any piece of data processing equipment is informally called hardware.

HARD-WIRED LOGIC:

Random Logic design solutions require interconnection of numerous integrated circuits representing the logic elements. An example of hardwired logic is the use of a hard-wired diode matrix instead of a ROM. These interconnections, whether done with soldering iron or by printed circuit board, are referred to as hard-wired logic in contrast to the software solutions achieved by a programmed ROM or Microprocessor.

HIGH LEVEL LANGUAGE:

This is a problem-oriented programming language as distinguished from a machine-oriented programming language. The former's instruction approach is closer to the needs of the problems to be handled than the language of the machine on which they are to be implemented.

*Addition to Handshaking: In micro-processor discussion, refers to the sequenced conditioning of control signals between the microprocessor and any other device for the purpose of data transfer.

HEXADECIMAL:

Whole numbers in positional notation using 16 as a base. (See Octal and Binary.) Since there are 16 hexadecimal digits (0 through 15) and there are only ten numerical digits (0 through 9) an additional six digits representing 10 through 15 must be introduced. Recourse is had to the alphabet to provide the extra digits. Hence, in order of significance, hexadecimal digits read: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. The decimal number 16 becomes the hexadecimal number 10. The decimal number 26 becomes the hexadecimal number 1A.

I**IMMEDIATE ADDRESSING:**

In this mode of addressing, the operand contains the value to be operated on, and no address reference is required.

IMMEDIATE OPERAND:

An 8-bit constant (arithmetic 2's complement number or logical mask) contained in the microprocessor's immediate addressed instruction.

INCREMENT (AND DECREMENT):

These two words are software operations most often associated with the stack and pointer. Bytes of information are stored in the stack register at the addresses contained in the stack pointer. The stack pointer is decremented after each byte of information is entered into the stack; it is incremented after each byte is removed from the stack. The terms can also refer to any addressable register.

INDEX REGISTER:

Contains a value used to modify or offset from a specific memory address as a function of itself. INDEXING is an address mode in which the address operand of an instruction is modified by the index register's contents during execution.

INDIRECT ADDRESSING:

Addressing a memory location which contains the address of data rather than the data itself.

**INSTRUCTION SET OR
REPERTOIRE**

Constitutes the total list of instructions which can be executed by a given microprocessor and is supplied to the user to provide the basic information necessary to assemble a program.

INTERFACE:

Indicates a common boundary between adjacent components, circuits, or systems enabling the devices to yield and/or acquire information from one another. In the face of common usage, one must regretfully add that the words Buffer, Handshake, and Adapter are interchangeable with interface.

INTERRUPT:

An interrupt involves the suspension of the normal programming routine of a microprocessor in order to handle a sudden request for service. The importance of the interrupt capability of a microprocessor depends on the kind of applications to which it will be exposed. When a number of peripheral devices interface the microprocessor, one or several simultaneous interrupts may occur on a frequent basis. Multiple interrupts may occur on a frequent basis. Multiple interrupt requests require the processor to be able to accomplish the following: to delay or prevent further interrupts; to break into an interrupt in order to handle a more urgent interrupt; to establish a method of interrupt priorities; and, after completion of interrupt service, to resume the interrupted program from the point where it was interrupted.

INTERRUPT MASK BIT:

The Interrupt Mask Bit prevents the CPU from responding to further interrupt requests until cleared by execution of programmed instructions. It may also be manipulated by specific mask bit instructions.

I/O (INPUT/OUTPUT):

Package pins which are tied directly to the internal bus network to enable I/O to interface the microprocessor with the outside world.

J

JUMP:

The Jump operation, like the Branch operation, is used to control the transfer of operations from one point to a more distant point in the control program. Jumps differ from Branching in not using the Relative Addressing mode.

L

LABEL:

A label may correspond to a numerical value or a memory location in the programmable system. The specific absolute address is not necessary since the intent of the label is a general destination. Labels are a requisite for jump and branch instructions.

LIBRARY:

A collection of complete programs written for a particular computer, minicomputer, or microprocessor. For example, Second Order Differential Equation may be the name of a program in the Library of a particular computer; this program will contain all the subroutines necessary to perform the solution of second order differential equations written in machine language and using the instruction set of this machine.

LIFO:

Last-In-First-Out buffer.
(See Push Down Stack.)

LISTING:

Refers to the pages of documentation which list the symbolic instructions, memory addresses, comments, and code of a given program sequence. In using the INSTRUCTOR, your listing is contained on the program forms you prepare. Larger computers typically output their listings to terminals or high speed line printers.

EXAMPLE EXTRACT FROM LISTING:

LINE ADDR OBJECT E SOURCE

2282	1E75	0506	LODI, R1 PULS0	GET NUMBER OF PULSES FOR A ZERO
2283	1E77	F401	TML, R0 H'01'	TEST FOR A ONE
2284	1E79	9801	BCFR, 0 OUT2	
2285	1E7B	51	RRR, R1	DIVIDE COUNT IF A ONE
2286	1E7C	FB02	OUT2 BDRR, R3 OUT3	CHECK FOR LAST BIT
2287	1E7E	8506	ADDI, R1 PULS0	YES, ADD LAST BIT PULSES
2288	1E80	0611	OUT3 LODI, R2 FREQ	LENGTH OF PULSE
2289	1E82	0710	LODI, R3 H'18'	SET ENW AND FREQ
2290	1E84	D7F8	WRTI, R3 CAS	
2291	1E86	FA7E	BDRR, R2 \$	DELAY 10 MICRO-SEC PER ITERATION
2292	1E88	0611	LODI, R2 FREQ	LENGTH OF PULSE
2293	1E8A	0710	LODI, R3 H'10'	RESET FREQ
2294	1E8C	D7F8	WRTI, R3 CAS	
2295	1E8E	FA7E	BDRR, R2 \$	DELAY 10 MICRO-SEC PER ITERATION
2296	1E90	F96E	BDRR, R1 OUT3	DO NEXT PULSE
2297	1E92	0688	LODI, R2 SPOLY	INTER-BIT SPACE
2298	1E94	0700	LODI, R3 H'00'	TURN OFF ENW AND FREQ

LOGIC:

A mathematical treatment of formal logic in which a system of symbols is used to represent quantities and relationships. The symbols or logical functions are called AND, OR, NOT, to mention a few examples. Each function can be translated into a switching circuit, more commonly referred to as a "gate." Since a switch (or gate) has only two states-open or closed-it makes possible the application of binary numbers for the solution of problems. The basic logic functions obtained from gate circuits are the foundation of complex computing machines.

LOOK AHEAD:

- (1) A feature of the CPU which allows the machine to mask an interrupt request until the following instruction has been completed.
- (2) A feature of adder circuits and ALUs which allow these devices to look ahead to see that all carries generated are available for addition.

LOOPING:

is repetition of one or more instructions in a sequence until a final value is determined. The loop is typically stored in ROM as a subroutine, then accessed when needed. Looping may also be executed within a CPU's WAIT state. Looping is typically used to achieve programmed delays, in various device polling schemes, and in other repetitive functions.

LSI (LARGE SCALE INTEGRATION):

At the beginning of the LSI era a count of 100 gates qualified for LSI. Today an 8-bit CPU can be fabricated on a single chip.

LSB

Least significant bit (or byte).

LSD

Least significant digit.

M

MACHINE LANGUAGE:

The only language the microprocessor can understand is binary. All other programming languages must be translated into binary code before entering the processor and decoded back into the original language after leaving it.

MACRO COMMAND:

A program entity formed by a string of standard, but related, commands which are put into effect by means of a single macro command. Any group of frequently used commands can be combined into a macro command. The many become one.

MASK

Refers to a logical pattern (in software) which is used to permit transfer of bit significant data from one point to another in a microprocessor. Data is said to be masked or mapped if the transfer takes place, and masked off if the mask inhibits the transfer.

MNEMONIC CODE:

These are designed to assist the human memory. The microprocessor language consists of binary words which are a series of 0's and 1's making it difficult for the programmer to remember the instructions corresponding to a given operation. To assist the human memory, the binary numbered codes are assigned groups of letters (or mnemonic symbols) that suggest the definition of the instruction. `LODZ R1` for "LOAD R1 into R0," etc. Source statements can be written in this symbolic language and then translated into machine language.

MEMORY:

The part of a computer system into which information can be inserted and held for future use. Storage and Memory are interchangeable expressions. Memories accept and hold binary numbers only. Memory types are core, disk, drum, and semiconductor.

MOS (METAL OXIDE SEMICONDUCTOR):

The structure of a MOS Field Effect Transistor (FET) is metal over silicon oxide over silicon. The metal electrode is the gate; the silicon oxide is the insulator; and carrier doped regions in the silicon substrate become the drain and source. The result is a sandwich very much like a capacitor, which explains why MOS is slower than bipolar since the 'capacitor sandwich' must charge up before current can flow. The three great advantages of MOS are its process simplicity because of reduced fabrication stages; the savings in chip real estate resulting in functional density; and the ease of interconnection on chip. These qualities enable MOS to break the LSI barrier, something bipolar is just beginning to achieve. The hand-held calculator and the microprocessor are triumphs of MOS-LSI technology.

MICROPROCESSOR:

The microprocessor is a Central Processing Unit fabricated on one or two chips. While no standard design is visible in existing units, a number of well-delineated areas are present in all of them: Arithmetic & Logic Unit, Control Block, and Register Array. When joined to a memory storage system, the resulting combination is referred to in today's usage as a micro-computer. It should be added that each microprocessor is supplied with an instruction Set, and this software manual may be just as important to the user as the hardware.

MODEM:

An electronic device in a serial data communications network that:

- (1) converts serial data from digital voltage levels to precise tonal frequencies (MO) for transmission.
- (2) Converts received frequencies into corresponding digital voltage levels. (DEM)
- (3) Performs handshaking operations to link its controlling device (terminal, cpu, etc.) with other stations in the communications network.

MSB:

Most significant bit (or byte).

MSD:

Most significant digit.

MULTIPLEXING:

Multiplexing describes a process of transmitting more than one signal at a time over a single link, route, or channel. Of the two methods, in use, one frequency shares the bandwidth of a channel in the same way hurdlers run and jump in their assigned lanes, thus permitting many contestants to compete simultaneously on the same track. The second way is to time-share multiple signals in the same way that pole vaulters jump over the same bar one after the other. The two methods may be described as parallel and serial processing. Time-sharing may not seem "simultaneous" but it should be remembered that the signal speed is so fast that it is possible to multiplex four different numbers through a single decoder-driver and have them appear on four different displays without a flicker to disturb the eye.

N

NESTING:

Nesting is referred to when a subroutine is enclosed inside a larger routine, but is not necessarily part of the outer routine. A series of looping instructions may be nested within each other.

NIBL ; NIBBLE

Is a data code occupying 4 bits (1/2 byte) 2 nibls equal one byte.

NOP OR NO-OP

Instruction code meaning no operation usually inserted into the program sequence to provide a short delay. In the 2650, this delay is 6 usec, given an input frequency clock equal to 1 MHz.

O

OBJECT CODE

Machine language code resulting from translation of symbolic machine instructions by the Assembler or by hand-coding.

OBJECT PROGRAM:

The end result of the source language program after it has been translated into machine language.

Sequence of object codes which may be executed by a microprocessor or other computer in order to perform some task.

OCTAL:

Whole numbers in positional notation using 8 as a base. The decimal or base 10 number, 125, becomes 175 in octal or base 8. Here is a convenient way to convert a decimal number into an octal number:

$\begin{array}{r} 175 \\ 8 \overline{) 125} \\ \underline{80} \\ 45 \\ \underline{40} \\ 5 \end{array}$	<p>Divide the decimal number by 8. The answer is 15 and 5 left over. Divide the answer, 15, by 8 again. The answer is 1 and 7 left over. The octal number is 175.</p>
--	---

To prove your answer is correct, do the following:

$\begin{array}{r} 5 \times 1 = 5 \\ 7 \times 8 = 56 \\ 1 \times 64 = 64 \\ \hline 125 \end{array}$	<p>Arrange the octal number vertically with the least significant digit on top. The least significant digit represents one's, so multiply $5 \times 1 = 5$. The next digit in the octal number represents 8's, so multiply $7 \times 8 = 56$. The third digit of the octal number represents 64's so multiply $1 \times 64 = 64$. The sum is the decimal number 125.</p>
--	---

OPERAND:

A quantity on which a mathematical operation is performed. One of the instruction fields in an addressing statement. Usually the statement consists of an operator and an operand. The operator may indicate an "add" instruction; the operand will indicate what is to be added.

OVERFLOW:

Overflow results when an arithmetic operation generates a quantity beyond the capacity of the register. Also referred to as arithmetical overflow. An overflow status bit in the Program Status register can be checked to determine if the previous operation caused an overflow.

OPERATING CODE (OPCODE):

Source statements which generate machine codes after assembly are referred to as operating codes.

P**PARALLEL OPERATION:**

Processing all the digits of a word or byte simultaneously by transmitting each digit on a separate channel or bus line.

PARTY-LINE:

Party-line as used in its telephone sense to indicate a large number of devices connected to a single line originating in the CPU.

PCI (PARALLEL COMMUNICATIONS INTERFACE):

A device which interfaces the microprocessor's bus-organized system with incoming serial synchronous communication information. The parallel data of the multi-bus system is serially transmitted by the asynchronous data terminal. The PCI interfaces directly with low-speed Modems to enable microprocessor communications over telephone lines.

PIPELINE:

Computers which execute serial programs only are referred to as pipeline computers.*

PLA (PROGRAMMED LOGIC ARRAYS):

The PLA is an orderly arrangement of logical AND logical OR functions. Its application is very much like a glorified ROM. It is primarily a combinational logic device.

- * Pipeline is a Hardware technique that permits different sections of a microprocessor to work simultaneously instead of sequentially, thus speeding up processor throughput.

POLLING:

Polling is the method used to identify the source of interrupt requests. When several interrupts occur at one time, the control program decides which one to service first.

Polling is a process performed by the control program to enable the microprocessor to interrogate each attached device's status or requirements.

PORT:

Device terminals which provide electrical access to a system or circuit. The point at which the I/O is in contact with the outside world.

PROGRAM:

A procedure for solving a problem and frequently referred to as Software.

PROGRAM COUNTER:

A processor register that contains a memory address of the current instruction to be executed. In some microprocessors, the Program Counter designates the next instruction. Also called the Instruction address register. During interrupt processing or subroutine access, the P.C. is typically saved in a Return Address Stack.

PUSH DOWN STACK:

A register that receives information from the Program Counter and stores the address locations of the instructions which have been pushed down during an interrupt. This stack can be used for subroutining. Its size determines the level of subroutine nesting (one less than its size or 15 levels of subroutine nesting in a 16 word register. When instructions are returned they are popped back on a last-in-first-out (LIFO) basis.

RRALU (REGISTER, ARITHMETIC,
AND LOGIC UNIT):

Unlike the discrete ALU package which functions as an Arithmetic and Logic unit only, the ALU in the microprocessor is equipped with a number of registers.

RAM (RANDOM ACCESS MEMORY):

Random in the sense of providing access to any storage location point in the memory immediately by means of vertical and horizontal co-ordinates. Information may be "written" in or "read" out in the same rapid way.

RANDOM LOGIC DESIGN:

Designing a system using discrete logic circuits. Numerous gates are required to implement the logic equations until the problem is solved. Even then, the design is not completed until all redundant gates are weeded out. Random logic design is no guarantee of optimum gate count.

REAL TIME OPERATION:

Data processing technique used to allow the machine to utilize information as it becomes available, as opposed to batch processing at a time unrelated to the time the information was generated.

REFERENCE DOCUMENTATION:

Specific source material for use in completing example or exercise requirements.

REGISTER:

A register is a memory on a smaller scale. The words stored therein may involve arithmetical, logical, or transferral operation. Storage in registers may be temporary, but even more important is their accessibility by the CPU. The number of registers in a microprocessor is considered one of the most important features of its architecture.

RELATIVE ADDRESSING:

The relative addressing mode specifies a memory location in the CPU's Program Location Counter register. This addressing mode is used for Branch instructions in which case an opcode is added to the Relative Address to complete the branching instruction. See "DISPLACEMENT."

ROM (READ ONLY MEMORY):

In its virgin state the ROM consists of a mosaic of undifferentiated cells. One type of ROM is programmed by mask pattern as part of the last manufacturing stage. Another more popular type, better known as P/ROM, is programmable in the field with the aid of programmer equipment. Program data stored in ROMs is often called firmware because it cannot be altered. However, another type of P/ROM is now on the market called EPROM which is erasable by ultra violet irradiation and electrically reprogrammable.

S**SCRATCHPAD:**

This term is applied to information which the Processing unit stores or holds temporarily. It is a RAM memory containing subtotals for various unknowns which are needed for final results.

SEQUENTIAL LOGIC:

A circuit arrangement in which the output state is determined by the previous state of the input. (See also Combination Logic.)

SOFTWARE:

What sheet music is to the piano, software is to the computer. Looked at from a practical point of view, one might say that software is the computer's instruction manual. The name, software, was obviously chosen to contrast with the formidable hardware which confronted the first programmer to communicate with the computer. Since the only language spoken by a computer is mathematical, the programmer must convert his verbal instructions into numbers. In the case of microprocessors, which vary from maker to maker, software libraries are assembled by the manufacturer for the benefit of the user.

SOURCE STATEMENT:

Is written in other than machine language within a program, to suggest the definition of an instruction in symbolic format. There are 2 types of symbolic source statements:

- a. executable instructions; those which can be interpreted by the processor, and
- b. assembly directives; those which tell the Assembler how to proceed in its translation of instructions, or which document the program.

SOURCE PROGRAM:

In Assembler, includes symbolic machine instructions, assembly directives and declarations, and comments.

SIMULATOR:

A special program that simulates the logical operation of the microprocessor. It is designed to execute object programs generated by a cross-assembler on a machine other than the one being worked on and is useful for checking and debugging programs prior to committing them to ROM firmware.

STACK:

The stack is a block of successive memory locations which is accessible from one end on a last-in-first-out basis (LIFO). The stack is coordinated with the stack pointer which keeps track of storage and retrieval of each type of information in the stack. A stack may be any block of successive information locations in the read/write memory.

SLICE:

A type of chip architecture which permits the cascading or stacking of devices to increase word bit size.

STACK POINTER:

The stack pointer is coordinated with the storing and retrieval of information in the stack. The stack pointer is incremented by one immediately following the storage in the stack of each word of information. Conversely, the stackpointer is decremented by one immediately before retrieving each word of information from the stack. The stack pointer may be manipulated for transferring its contents to the Index register or vice versa.

STATUS WORD REGISTER:

Is a specific register in the microprocessor which summarizes the status of the microprocessor after each operation. Also may contain one or more control bits for internal access by the processor. Refer to Figure 11, Module II-A.

STORAGE:

The word storage is used interchangeably with memory. In fact, it has been recommended as the preferred term by people who would rather not imply that the computer has any relationship with the human brain.

SUBROUTINE:

Part of the master program which may be used at will in a variety of master routines. The object of a Branch or Jump Command.

SYMBOLIC MACHINE INSTRUCTIONS

Instructions written in symbolic conversational format. Must be translated into computer-useable machine language by the Assembler or by hand-coding operations.

For example:

BCTR,UN \$-8 = 0001101101110110
means "Branch unconditionally to a location in memory whose effective address precedes this instruction's location by 8 bytes."

I

THROUGHPUT:

The speed with which problems or segments of problems are performed is called Throughput. Defined in this way, it is obvious that throughput will vary from application to application. As an index of speed, throughput is meaningful only in terms of your own application.

TWO'S COMPLEMENT NUMBERS:

The ALU performs standard binary addition using the 2's complement numbering system to represent both positive and negative numbers. The positive numbers in 2's complement representation are identical to the positive numbers in standard binary. +127 in standard binary = 01111111 +127 in 2's complement = 01111111. The most significant (leftmost) bit indicates the sign.

1 = negative. 0 = zero or positive.

However, the negative 2's complement is the complement of the negative standard binary plus 1. -127 in standard binary = 11111111. To form the 2's complement of -127:

First, complement all bits, except the SIGN bit:
 bit: = 10000000
 +1
 10000001 = -127 in 2's complement.

U

UART (UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER):

This device will interface a word parallel controller or data terminal to a bit serial communication network.

V

VECTORED INTERRUPT:

This term is used to describe a micro-processor system in which each interrupt, both internal and external, have its own uniquely recognizable address. This enables the microprocessor to perform a set of specified operations which are pre-programmed by the user to handle each interrupt in a distinctively manner.

W

WORD:

A group of "characters" treated as a unit and given a single location in computer memory. Presumably a byte is a group of bits in contrast to a word which is a group of numeric and/or alphabetic characters and symbols, but the two words are used interchangeably more often than not.

WORKING STORAGE:

That part of Random Access memory set aside for intermediate calculations, temporary results, other non-fixed data or soft routines. In larger computers, files are read into working storage from peripheral disk drives or tapes. When computation is complete, the resultant file is written back to the long-term storage device. The 512 bytes of USER RAM in the INSTRUCTOR is working storage.

DIRECTION:

You have an opportunity to check your understanding of the terms defined in this glossary by completing the self-administered quiz on the next page. Once you finish, compare your answers to those provided on page 34 and following.

SELF-ADMINISTERED QUIZSOFTWARE/HARDWARE
MICROPROCESSOR TERMS**DIRECTION:**

Complete the requirements of this quiz, then compare your answers to those provided on page 34 to 36. Use the glossary as a reference.

1. The 3 essential components of a microprocessor include:

(1) _____ (2) _____ (3) _____

2. What is the PRIMARY FUNCTION of any assembler program?

3. A port or data base is said to be _____ if data can be transferred in either direction.

4. How many binary combinations can be generated within a byte (8 bits).

5. How many bytes of memory would be required for the following numbers when stored as BCD codes?

(a) 463 _____

(b) 4,096 _____

(c) 65,365 _____

6. What is the unique difference between synchronous and asynchronous timing?

DIRECTIONS:

Go right on to the next page.

7. What is the difference between a byte and a nibl?

8. Is the clock frequency input to a microprocessor the major determining factor of the processing speed of the microprocessor?
(yes)(no) _____
9. What is the difference between an assembler and a cross-assembler?

10. Define execution time _____
11. During Direct Memory Access operations, the host C.P.U. is
(enabled)(disabled) _____
12. You would be concerned with DISPLACEMENT particularly if you are writing _____ addressed instructions.
13. When the absolute address of the current instruction is greater than the next effective address, displacement is said to be
(positive)(negative) _____
14. What's the major distinction between a JUMP and a LOOP?

15. "SCRATCHPAD" memory is used for storage of _____
(Temporary data and intermediate results)(constants)(fixed data tables)

DIRECTION:

Go right on to the next page.

16. What's the major difference between a microprocessor and a microcomputer _____
17. You'd normally use _____ (RAM)(ROM) for storage of a debugged instruction sequence in the manufactured version of a given application.
18. Could "CRAPGAME" be considered a "pipeline" program?

19. What's the difference between source and object programs?

20. Multiple external devices are normally attached to a microprocessor system through separate _____
21. A return address stack operates on a _____ (FIFO)(LIFO) basis in which addresses are pushed into the stack when program control is transferred to a subroutine, and popped from the stack when program control is returned to the calling routine.
22. If a binary number is added to its 2's complement, the result is 0 with a carry from the most significant bit _____ (true)(false).
23. What differentiates a vectored interrupt from a polled interrupt?

24. A UART of some form could be implemented to interface

25. Is "MNEMONIC CODE" the same as "SYMBOLIC MACHINE LANGUAGE"?
(yes)(no) _____

DIRECTION:

Go right on to the next page.

26. In the following example, which is the MSD?
74F3B _____
27. In the above example, which is the LSD? _____
28. What's the binary code of 'F'? _____
29. Are OVERFLOW and CARRY functions normally flagged in a CPU?
(yes)(no) _____
30. An instruction sequence which has been stored in ROM is known as
_____ (software)(firmware)(hardware).


DIRECTION:

Compare your responses to those provided on the next three pages.

SUGGESTED ANSWERS TO THE SELF ADMINISTERED QUIZ

1. ALU, internal registers, control (logic) block.
2. To translate symbolic source statements into binary object code.
3. Bidirectional.
4. 256 (00000000) ----- (11111111)
5. (a) 2 (b) 2 (c) 3 (see the * note on page 36)
6. Synchronous timing is precise in timed increments. Asynchronous has no fixed timing cycle.
7. A byte commonly has a length of 8 bits. Nib1 is 4 bits.
2 nibls = 1 byte.
8. No. (Its internal components and complexity are the major factors.
9. The cross-assembler program translates source statements into object code using another computer. The assembler uses the host computer or microprocessor.
10. Execution time is the time, in processor input clock cycles, necessary to carry out the process of one instruction. Execution time is usually expressed in terms of 2 or more machine cycles; each cycle requiring 3 clock times in the case of the 2650 microprocessor.
11. Disabled. Some external device has control of line which effect memory access.

12. Relative addressed instruction.
13. Negative.
14. In a JUMP, program control is turned over to a separate set of instructions. In a LOOP, the same instruction or instruction sequence is repeated a number of times.
15. Temporary data and intermediate results, also (as in the INSTRUCTOR) for new untried programs prior to production.
16. A microcomputer is a microprocessor plus memory.
17. ROM - you want the program to be non-destructible.
18. No! Consider the number of directions "CRAPGAME" can take, dependent on the user's manual inputs . . . and luck!
19. Source programs are written in quasi conversational statements (mnemonics or symbolic machine language). Object programs consist of the binary codes resulting from source program translation by an assembler or compiler.
20. "PORTS."
21. "LIFO" - last in, first out.
22. True

Given the number 26	=	00011010	00011010
Its one's complement	=	11100101	plus
plus 1	=	11100110	11100110 +
Add	-26		
	+26		
	<u>00</u>		
		= 1	00000000
			
		CARRY	

23. A device operating in a vectored interrupt system asserts a unique address to the processor so that the microprocessor can differentiate which device has interrupted - immediately. A polled interrupt requires the microprocessor to interrogate each device to determine which one raised the interrupt.
24. A (parallel-operating) microprocessor to a data communications network involving serial bit transfer.
25. Yes.
26. 7 *
27. B *
28. 1111 *
29. Yes.
30. Firmware.

*If you did not respond quickly and without error to these questions, you should elect the option of completing the next section in this module.

DIRECTION:

Set aside the documentation related to this section and, at your option, access the next section (Module III-B Number System Review) or Module IV-A Introduction to the 2650 Instruction Repertoire.

If you entered this module from page 3 of Module I, complete Module III-B or return to Module I at your option.



2650 MICROPROCESSOR COURSE

MODULE III-B

NUMBER SYSTEM REVIEW

PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086

REFERENCE:

Outlines and Abstracts
page 8

INTRODUCTION

For some obscure reason, known only by Roman-Greek ancients, we in the Western world are schooled in the DECIMAL system, the system of counting by tens. Depending on which way you look at it, this is fortunate or unfortunate. From the view point of a COMPUTER, decimal system usage is somewhat unfortunate.

For the COMPUTER, as we understand it today, performs all of its calculations, its algorithms, its data manipulation in terms of the simplest counting system. This is the BINARY number system. Whereas the decimal system generates a carry at 9, 99, 999, etc., in the BINARY system, carry is generated at 1, 11, 111, 1111, etc.

Thus, counting in decimals from 0 to 16 looks like Figure 1 when expressed in BINARY notation.

FIGURE 1

COMPARISON OF BINARY/DECIMAL NOTATION

DECIMAL	0	1	2	3	4	5	6	7
BINARY	0	1	10	11	100	101	110	111

DECIMAL	8	9	10	11	12	13	14	15	16
BINARY	1000	1001	1010	1011	1100	1101	1110	1111	10000

● = CARRY

Why does the computer (costing \$7 million or \$7.00) "think" in BINARY? Simply because it is the essence of a logical machine. There are no "gray areas" to consider. To a computer, an answer is either "YES" or "NO"; a signal is "ON" or "OFF"; a number is either "0" or "1".

When, for instance, a computer adds 7+6, it performs the ADD algorithm in BINARY. Thus, the computation is not simply 7+6=13; it is 0111+0110 which equals 1101. Since we as humans don't think in terms of zeros and ones, the programmer may instruct the computer to perform a further algorithm, this time one of conversion of its computed result from BINARY (e.g. 1101) to DECIMAL (e.g., 13) before it is printed or displayed by some externally controlled device.

This takes TIME. And increases the number of operations that the computer (Central Processing Unit or CPU) must perform.

Most microprocessors have instructions that can be implemented to perform conversions from one number system to another. And certainly all calculators, whether simple 4 function or fully programmable, have this built-in capability. You've seen the conversion function implemented hundreds of times by the "INSTRUCTOR". Specifically, in the operation of the program "CRAPGAME". For example, in routine RLDICE, the display of "individual dice" and subsequently their sum (in the messages X--X = YY and SHOOT YY) is in decimal. Later in this course, you examine the instructions which made this possible.

In the previous section of this module, you were introduced to the concept and purpose of an ASSEMBLER. In its operating program, the ASSEMBLER contains algorithms to convert numbers from DECIMAL input to BINARY computation and (sometimes) output. The "INSTRUCTOR" has the same capability, to a limited degree, simply because it is limited by the size of its operating program (USE). Thus, some help is required, on your part, in the process of converting numbers from decimal to their binary equivalent or at least to some form which can be "read" by the microprocessor. For example, you'll see later on that the symbolic instruction:

BCTR,EQ \$+50 must be hand-coded to the values 1B 32.

Which brings up the question: Where are the (Binary) zeros and ones? Let's examine that now.

The value 1B 32 consists of 2 bytes of data, each 8 bits in length.

The value: 1B 32 could just as easily be written in BINARY form, the number system that the CPU understands. In binary, 1B 32 = 00011011 0011 0010. Recognize however that 16 zeros and ones are much more difficult to read than the numerals 1B 32. So the operating system of the "INSTRUCTOR" (and that of most other computers) contains a conversion program which accepts numbers like 1B 32, and converts them to computer-acceptable (machine) code like 00011011 00110010. Notice that each number in the expression 1B 32 is equivalent to 4 binary bit values.

With 4 binary bit values available, there are 2^4 or 16 possible binary patterns possible. The computer assigns each pattern a specific numeric value, ranging from 0 to F. These are shown in Table 1 (next page).

It's easy to assign the equivalent for numbers 0 through 9; these are in effect, straight decimal. However, note that with 16 possible combinations, we don't want to carry after 9 (as in decimal) but at 15 (as shown in the transition from 1111 to 10000 (A)). Thus, some bright computer ancient thought up a separate number system with 10 numerals and 6 (numerically assigned) alpha characters. The 10 numerals (0-9) equal the equivalents 0000→1001 binary. The 6 letters (A-F) equal their equivalents 1010 1111 binary. And in decimal, these 6 letters carry weights *numerical values) from 10 through 15.

TABLE 1

BINARY	EQUIVALENT
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A (10)
1011	B (11)
1100	C (12)
1101	D (13)
1110	E (14)
1111	F (15)

A {

10000	10 (16)
10001	11 (17)
10010	12 (18)
10011	13 (19)
...	...
etc.	etc.

To cut the story short, the number system thus illustrated was given a name: "HEXADECIMAL", abbreviated "HEX."

Simply, hexadecimal implies 2 components, hex and decimal. HEX for 6 letters and DECIMAL for 10 distinct numbers. Instead of 10 numbers with a carry generated when 9 is incremented by 1 (as in decimal) the hexadecimal system contains 16 numbers (0 through 9; A through F) with carry generated when F(15) is incremented by 1. Further, when considered in hexadecimal, the symbols 0 through 9 and A through F all have numeric values or weights; every bit as much as 0 through 9 do in the DECIMAL system or 0 through 1 do in the BINARY system.

Now, armed with 16 distinct numbers, it is an easy matter for the programmer to write an algorithm instructing the CPU to convert each HEXa-decimal digit to a 4-bit binary pattern. Thus, as illustrated previously, the numbers 1B 32 are converted to:

0001 1011 0011 0010

Conversely, the computer may execute an algorithm to convert the binary values it has been manipulating into (easily read) HEXadecimal code. For example:

0110 1111 1001 0100 = 6F 94

During output operations, you may wish to convert data from HEX to its decimal equivalent. This will occur when you verify memory contents. For example, if you read the code 07 3B, this is equivalent to the symbolic instruction `LODI,R3 H'3B'`. In actuality, you would write the instruction `LODI,R3 59`. For `H'3B' = 59` decimal.

Table 3 illustrates a conversion table from HEX to DECIMAL. Note that each column demonstrates all possible combinations for each 4-bit group (nibble) involved in any 2 byte combination.

Two examples are provided:

A Convert 145_{10} to HEX

TABLE 3

145
 $-144 \rightarrow = H'90' \rightarrow$ 2nd column
 $1 \rightarrow = H'01' \rightarrow$ right-most
 $H'91'$ column
 (remainder between 1 and 15)

B Convert `H'2B6'` to DECIMAL

`H'2xx'` = 512 (column 3)
`H'xBx'` = 176 (column 2)
`H'xx6'` = 6 (column 1)

Sum = 694

Therefore, `H'2B6'` = 694_{10}

Conversion to BINARY from HEX is simple:

$694_{10} = '2B6' = 001010110110B$

HEXADECIMAL COLUMNS							
4		3		2		1	
HEX=DEC		HEX=DEC		HEX=DEC		HEX=DEC	
0	0	0	0	0	0	0	0
1	4096	1	256	1	16	1	1
2	8192	2	512	2	32	2	2
3	12288	3	768	3	48	3	3
4	16384	4	1024	4	64	4	4
5	20480	5	1280	5	80	5	5
6	24576	6	1536	6	96	6	6
7	28672	7	1792	7	112	7	7
8	32768	8	2048	8	128	8	8
9	36864	9	2304	9	144	9	9
A	40960	A	2560	A	160	A	10
B	45056	B	2816	B	176	B	11
C	49152	C	3072	C	192	C	12
D	53248	D	3328	D	208	D	13
E	57344	E	3584	E	224	E	14
F	61440	F	3840	F	240	F	15
BYTE 1				BYTE 2			

If the value `H'2B6'` were placed in 2 microprocessor registers (e.g. R1 and R2) the actual contents of each would be as follows:

MOST SIGNIFICANT BYTE								LEAST SIGNIFICANT BYTE									
	7	6	5	4	3	2	1	0		7	6	5	4	3	2	1	0
R1	0	0	0	0	0	0	1	0	R2	1	0	1	1	0	1	1	0

And, if these registers were displayed on the INSTRUCTOR, the following would appear:

OBSERVATION: R1 = 02 R2 = B6

Now, however, if you wished to execute a routine whose START ADDRESS is at the 694th location in Memory, you would not depress

REG C 6 9 4 RUN

Instead, you would depress REG C 2 B 6 RUN simply because the "INSTRUCTOR" (and its microprocessor) think in HEX.

If, subsequently, you verify the Program Counter by depressing REG C you would observe .PC=02B6 on the display.

DECIMAL/HEX and HEX/DECIMAL CONVERSION WITH A 4-FUNCTION CALCULATOR

INTRODUCTION:

While the subtract (decimal to hex conversion) and addition (hex to decimal conversion) methods are workable using Table 3 data, they are time consuming, especially when converting large numbers. Most of you have access to a simple 4-function calculator. The next few pages detail procedures for rapid number conversion easily, implemented with a calculator. Use Table 4 as directed.

TABLE 4

DECIMAL FRACTION REMAINDER			
IF DECIMAL=	WRITE	IF DECIMAL=	WRITE
.000	0	.5	8
.0625	1	.5625	9
.125	2	.625	A
.1875	3	.6875	B
.25	4	.75	C
.3125	5	.8125	D
.375	6	.875	E
.4375	7	.9375	F

DECIMAL TO HEX CONVERSION:

Procedure:

- 1) Enter decimal value to calculator
- 2) $\div 16$
- 3) Convert fractional remainder to hex equivalent using table

MODULE III-B

NUMBER SYSTEM REVIEW

- 4) This HEX equivalent is the least significant digit (LSD) of the HEX value.
- 5) Subtract fractional remainder
- 6) If the whole number remaining is greater than 16, repeat steps 2 through 5. Then again convert the fractional remainder to its HEX equivalent, and write as the next least significant digit. Continue this process until the whole number remaining is less than 16. Write that number as the most significant digit.

EXAMPLE 1

To find the hex equivalent of 7495:

$$\begin{array}{rcl}
 7495 \div 16 & = & 468 \text{ (375)} \\
 & & \underline{-.375} \\
 468 \div 16 & = & 29 \text{ (25)} \\
 & & \underline{-.25} \\
 29 \div 16 & = & 1 \text{ (8125)}
 \end{array}$$

CONVERT

H '1 D 4 6'

EXAMPLE 2

To find the hex equivalent of 77,389:

$$\begin{array}{rcl}
 77389 \div 16 & = & 4836.8125 \\
 4836 \div 16 & = & 302.25 \\
 302 \div 16 & = & 18.875 \\
 18 \div 16 & = & 1.125
 \end{array}$$

'1 2 E 4 D'

HEX TO DECIMAL CONVERSION

This procedure works on the principle that each hex digit, working from the least significant, is equal to $(N) \times (16^Y)$

where N is the fractional decimal remainder (Table 2) equal to the hex digit and

Y is the power to which 16 is raised, determined by the position of the hex digit.

POSITION

6	5	4	3	2	1
6 LSD	5 LSD	4 LSD	3 LSD	2 LSD	LSD
16^6	16^5	16^4	16^3	16^2	16^1
16777216	1048576	65536	4096	256	16

EXAMPLE 1

To find the decimal equivalent of H'39DE'

$\begin{array}{c} \nearrow \nearrow \nearrow \nearrow \\ \text{position } 4 \quad 3 \quad 2 \quad 1 \end{array}$

POSITION 4 3 = fract. equivalent of .1875 x 65536 = 12288

POSITION 3 9 = fract. equivalent of .5625 x 4096 = 2304

POSITION 2 D = fract. equivalent of .8125 x 256 = 208

POSITION 1 E = fract. equivalent of .875 x 16 = 14

TOTAL EQUALS 14814

Therefore, H'39DE' = 14,814₁₀

EXAMPLE 2

To find the decimal equivalent of H'D309BB'NOTE; only the result of each calculation is shown

D = 13631488

3 = 196608

0 = 0

9 = 2304

B = 176

B = 11

13,830,587

Therefore, H'D309BB' = 13,830,587₁₀

And, converted to binary, 13,830,587₁₀ = 1101 0011 0000 1001 1011 1011B and would require 3 registers (or 3 memory locations) for storage.

////////////////////////////////////

EXERCISE:

The following problems are provided to reinforce your understanding of HEXadecimal to DECIMAL and BINARY equivalent numbers. The correct answers are provided on page 11 of this module.

1. DECIMAL to HEX CONVERSION:

- a) $491_{10} = H' \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad}$
 b) $1000_{10} = H' \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad}$
 c) $10000_{10} = H' \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad}$
 d) $3335_{10} = \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad}$
 e) $4096_{10} = \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad}$
 f) $754,987 = \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad}$
 g) $95,688 = \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad} \underline{\quad}$

DIRECTION

Check your answers with those provided on page 11,
then complete part 2.

2. HEX to BINARY CONVERSION

Convert the HEX numbers derived in part 1 to their BINARY equivalent. In each case, consider that the LEAST significant 2 digits are to be stored in R1, the 2nd L.S. digits in R2, and the 3rd L.S. digits in R3. Show the binary value (0 or 1) for each bit location. If a register is NOT used, show its contents = 0, as applicable.

	HEX	R3	R2	R1
a.	_____	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>
b.	_____	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>
c.	_____	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>
d.	_____	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>
e.	_____	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>
f.	_____	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>
g.	_____	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>	<div style="display: flex; flex-direction: row-reverse;"><div>1</div><div>2</div><div>3</div><div>4</div><div>5</div><div>6</div><div>7</div><div>8</div></div>

3) HEX to DECIMAL CONVERSION

a) H'14DD' =

b) H'35' =

c) '1FFF' =

d) 'CDEFF' =

e) '3DF' =

f) '10000' =

g) 'FAB' =

h) '123456' =

4) ADD each pair of HEX numbers in part 3. e.g., H'14DD' + H'35' = H'____'. Express each result as a BINARY number.

3a + 3b = H'____' = _____ B

3c + 3d = H'____' = _____ B

3e + 3f = H'____' = _____ B

3g + 3h = H'____' = _____ B

DIRECTION

After completing each part of the exercise, compare your answers with those on the next page. Then go on to Module IV of the course.

ANSWERS TO EXERCISE

PART 1

- a) H'1EB'
- b) H'3E8'
- c) H'2710'
- d) H'D07'
- e) H'1000'
- f) H'B852B'
- g) H'175C8'

PART 2

- a) R3=0 R2=0000 0001 R1=1110 1011
- b) R3=0 R2=0000 0011 R1=1110 1000
- c) R3=0 R2=0010 0111 R1=0001 0000
- d) R3=0 R2=0000 1101 R1=0000 0111
- e) R3=0 R2=0001 0000 R1=0
- f) R3=0000 1011 R2=1000 0101 R1=0010 1011
- g) R3=0000 0001 R2=0111 0101 R1=1100 1000

PART 3

- a) 5341
- b) 53
- c) 8191
- d) 843519
- e) 991
- f) 65536
- g) 4011
- h) 1193046

PART 4

3a + 3b = H'1512' = 1 0101 0001 0010B - leading zeros of M.S. digit dropped

3c + 3d = H'CFEFe' = 1100 1111 1110 1111 1110B

3e + 3f = H'103DF' = 1 0000 0011 1101 1111B - leading zeros of MS digit dropped

3g + 3h = H'124401 = 1 0010 0100 0100 0000 0001 "

SUMMARY of HEXadecimal and DECIMAL SYMBOLIC NOTATION

- HEX numbers: 0 through 9; A through F; identified as H'XX...X'
'XX...X', or
XX...X_x
- DECIMAL numbers 0 through 9; identified as NN or NN...N₁₀

DIRECTION

Go right on to Module 4.



2650 MICROPROCESSOR COURSE

MODULE IV - A

INSTRUCTION REPERTOIRE - OVERVIEW

PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086

REFERENCE:

Outlines and Abstracts
page 9

DIRECTION: Install tape 4B "INSTRUCTION REPERTOIRE I" side A up.

Listen to tape 4B through the first tone. This provides a summary of key points studied in Modules 1 through 3, and an introduction to Module 4. SEE NOTE BELOW:

COURSE MATERIALS:

Before proceeding further in Module IV, you should obtain the following course materials.

1. 2650 REFERENCE GUIDE fan-folded pocket-size hard card.
2. 2650 MICROPROCESSOR 174 page bound copy. Otherwise known as the 2650 REFERENCE MANUAL.
3. 2650 PROGRAMMING FORMS 50 sheet glued tear-off programmers pad.

Then, continue your studies in Module IV-B documentation, page 1.

INDEX TO MODULE IV - "INSTRUCTION REPERTOIRE"

IV-A	SUMMARY AND INTRODUCTION TO INSTRUCTION REPERTOIRE	tape only
IV-B	2650-REFERENCE GUIDE	
IV-C	INTRODUCTION TO THE INSTRUCTION SET FUNCTIONS, ADDRESS MODES	
IV-D	2650 PROGRAMMING FORM	
IV-E	MICROPROCESSOR MEMORY CONCEPTS	
IV-F	ADDRESS MODES AND FORMATS	} highly detailed for maximum impact, the subjects covered in these sections are merged.
IV-G	INSTRUCTION FUNCTIONS	
IV-H	PSW MANIPULATION	detailed description of PSW control and usage.
IV-I	MONITOR SUBROUTINE ANALYSIS	
IV-K	MISCELLANEOUS PROGRAMMING TOPICS	A complex program is described, from idea to implementation on the INSTRUCTOR.

*** TAPE SEQUENCE NOTE ***

From the start of Side B, there are approximately 2 minutes of music before instructional material



2650 MICROPROCESSOR COURSE

MODULE IV - C

INTRODUCTION TO INSTRUCTION SET FUNCTIONS

PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT
Signetics Corporation
811 E. Arques Ave.
Sunnyvale, CA, 94086

REFERENCE:

Outlines and Abstracts
page 11

INTRODUCTION:

Generally, instructions performed by any microprocessor or computer fall into one of 3 major functional categories. These include:

- DATA MOVEMENT
- ARITHMETICAL/LOGICAL COMPUTATION
- PROGRAM (SEQUENCE EXECUTION) CONTROL

Data Movement instructions facilitate the transfer of information between the microprocessor and its memory or controlled I/O.

The power of the microprocessor to perform its primary function, that of computation, is determined by the number and variety of arithmetic and logical instructions it can execute.

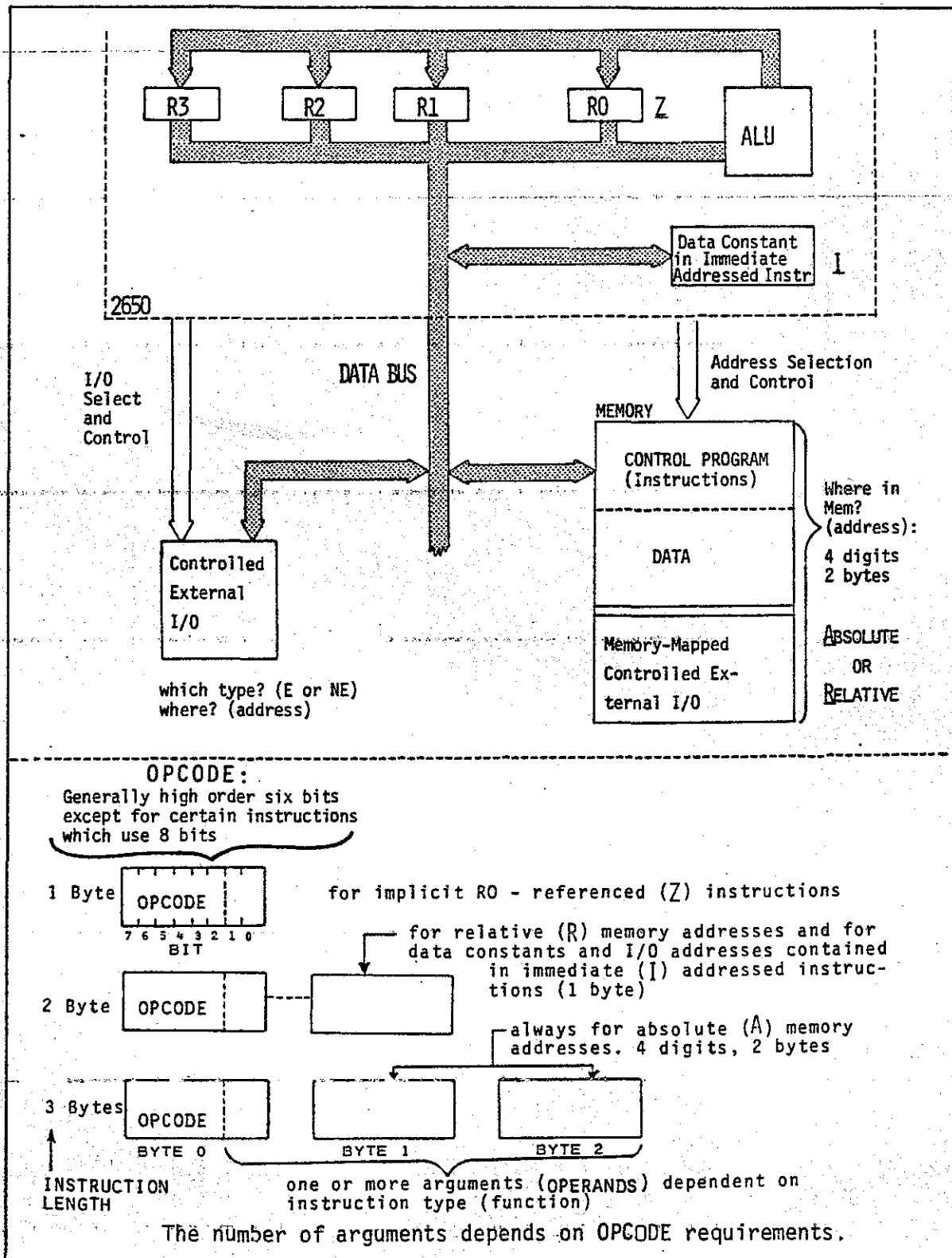
By themselves, data transfer and arith/logical instructions are only the steps or building blocks in a program sequence. By executing Program Control instructions, the microprocessor acquires a power not only to do, but to think; not only to manipulate data, but to reason, ask questions and get answers which determine the flow or sequence of the program. In short, program control instructions give cohesion and flexibility to the program.

DIRECTION:

Listen to tape 4B for a brief description of FIGURE 1.

FIGURE 2

CONCEPT OF FLEXIBLE DATA TRANSFER



	DESCRIPTION OF OPERATION	FORMAT	MNE-MONIC	OP CODE R or CC				BYTES	CYCLES	BIT FDR- MAT	PSW BITS AFFECTED							NOTE	2650 REF. MAN. PG.
				8	1	2	3				CC	IDC	C	OVF	SP	II	F		
BRANCH	Branch on incrementing register relative	7	BIR	R	D8	D9	DA	DB	2	3	R							7.8	76
	Branch on incrementing register absolute	8		A	DC	DD	DE	DF	3	3	B							7.8	
	Branch on decrementing register relative	7	BDR	R	F6	F9	FA	FB	2	3	R							7.8	77
	Branch on decrementing register absolute	8		A	FC	FD	FE	FF	3	3	B							7.8	
	Zero branch relative, unconditional	6	ZBRR		—	—	—	9B	2	3	ER							6	73
	Branch indexed absolute, unconditional	9	BXA		—	—	—	9F	3	3	EB							5.6	79
SUBROUTINE BRANCH/RETURN	Branch to subroutine on condition true, relative	7	BST	R	38	39	3A	3B	2	3	R							7.8	80
	Branch to subroutine on condition true, absolute	8		A	3C	3D	3E	3F	3	3	B							7.8	
	Branch to subroutine on condition false, relative	7	BSF	R	B8	B9	BA	—	2	3	R							7	81
	Branch to subroutine on condition false, absolute	8		A	BC	BD	BE	—	3	3	B							7	
	Branch to subroutine on non-zero register, relative	7	BSN	R	78	79	7A	7B	2	3	R							7.8	82
	Branch to subroutine on non-zero register, absolute	8		A	7C	7D	7E	7F	3	3	B							7.8	
	Zero branch to subroutine relative, unconditional	6	ZBSR		—	—	—	8B	2	3	ER							8	79
	Branch to subroutine, indexed, absolute unconditional	9	BSXA		—	—	—	8F	3	3	EB							5.8	83
	Return from subroutine, conditional	3	RET	C	14	15	16	17	1	3	Z							8	84
	Return from subroutine and enable interrupt, conditional	3		E	34	35	36	37	1	3	Z							8	
INPUT/OUTPUT	Write data	3	WRTD		F0	F1	F2	F3	1	2	Z							1	86
	Read data	3	REDD		70	71	72	73	1	2	Z							1	84
	Write control	3	WRTC		S0	S1	S2	S3	1	2	Z							1	85
	Read control	3	REDC		30	31	32	33	1	2	Z							1	85
	Write extended	5	WRTE		D4	D5	D6	D7	2	3	I							1	87
MISC.	Read extended	5	REDE		54	55	56	57	2	3	I							1	85
	Halt, enter wait state	1	HALT		40	—	—	—	1	1	E								90
	No operation	1	NOP		C0	—	—	—	1	1	E								87
	Test under mask immediate	5	TMI		F4	F5	F6	F7	2	3	I							4	88
PROGRAM STATUS	Load program status, upper	1	LPS	U	—	—	82	—	1	2	E							9	68
	Load program status, lower	1		L	—	—	93	—	1	2	E							9	
	Store program status, upper	1	SPS	U	—	—	12	—	1	2	E							1	70
	Store program status, lower	1		L	—	—	13	—	1	2	E							1	
	Clear program status, upper, masked	4	CPS	U	—	—	74	—	2	3	EI							9	71
	Clear program status, lower, masked	4		L	—	—	75	—	2	3	EI							9	
	Preset program status, upper, masked	4	PPS	U	—	—	78	—	2	3	EI							9	70
	Preset program status, lower, masked	4		L	—	—	77	—	2	3	EI							9	
	Test program status, upper, masked	4	TPS	U	—	—	84	—	2	3	EI							4	72
	Test program status, lower, masked	4		L	—	—	85	—	2	3	EI							4	

OTHER TABLES: PP. 151 AND FOLLOWING
TIMING PP. 34-37
INTERFACE PP. 29-33, P. 39-44
147-148

INTERNAL ORGANIZATION AND PSW } PP. 21-26

MEMORY ORGANIZATION PP. 27-28

ADDRESS MODES DESCRIPTION PP. 47-51

SEE NEXT PAGE →

PAGE REFERENCE IN
2650 MICROPROCESSOR
MANUAL

PAGE REFERENCES TO 2650 MICROPROCESSOR
MANUAL; © 1975 REVISED 1977.

FURTHER QUALIFICATION OF INSTRUCTIONS LISTED IN REPERTOIRE.

SYMBOLIC INSTRUCTION FORMATS ACCEPTABLE TO 2650 ASSEMBLER AS SOURCE CODE ENTRIES.

SHORT FORM PINOUTS FOR HARDWARE DESIGN ASSISTANCE

NOTES

1. Condition code (CC): CC0: 01 if positive, 00 if zero, 10 if negative.
2. Condition code (CC): CC0: 01 if $R0 > r$, 00 if $R0 = r$, 10 if $R0 < r$.
3. Condition code (CC): CC0: 01 if $r > V$, 00 if $r = V$, 10 if $r < V$.
4. Condition code (CC): CC0: 00 if all selected bits are 1s, 10 if not all the selected bits are 1s.
5. Index register must be register 3 or 3.
6. Requires two additional cycles if indirection is specified.
7. Requires two additional cycles if indirection is specified and branch is taken.
8. Specify CC = 11 for unconditional branch.
9. RS, WC and COM bits in PSW are also affected.
10. CC assumes number in register is a binary number.

ASSEMBLER FORMATS

Format No.	Format	Addressing Mode
1	OP	E
2	OP	Z
3	OP, r/c	Z
4	OP	I
5	OP, r	I
6	OP, r/c, [z]	R
7	OP, r/c	R
8	OP, r/c	B
9	OP	JB
10	OP, r	A

Optional data enclosed in []. Brackets are not part of syntax.

OP = operation code

r = register number (0-3)

c = condition code (0-3)

z = value (0-255)

* indicates indirect addressing

z = zero page addresses (0-63, 8128-8191)

d = relative displacement addresses (-64 to +63)

a = addresses (0-32767)

p = page addresses (0-8191)

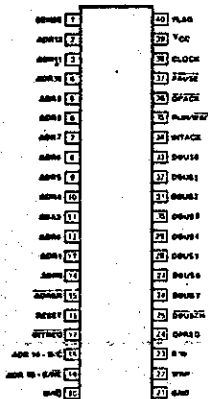
x = index register number (0-3)

+ = auto-increment

- = auto-decrement

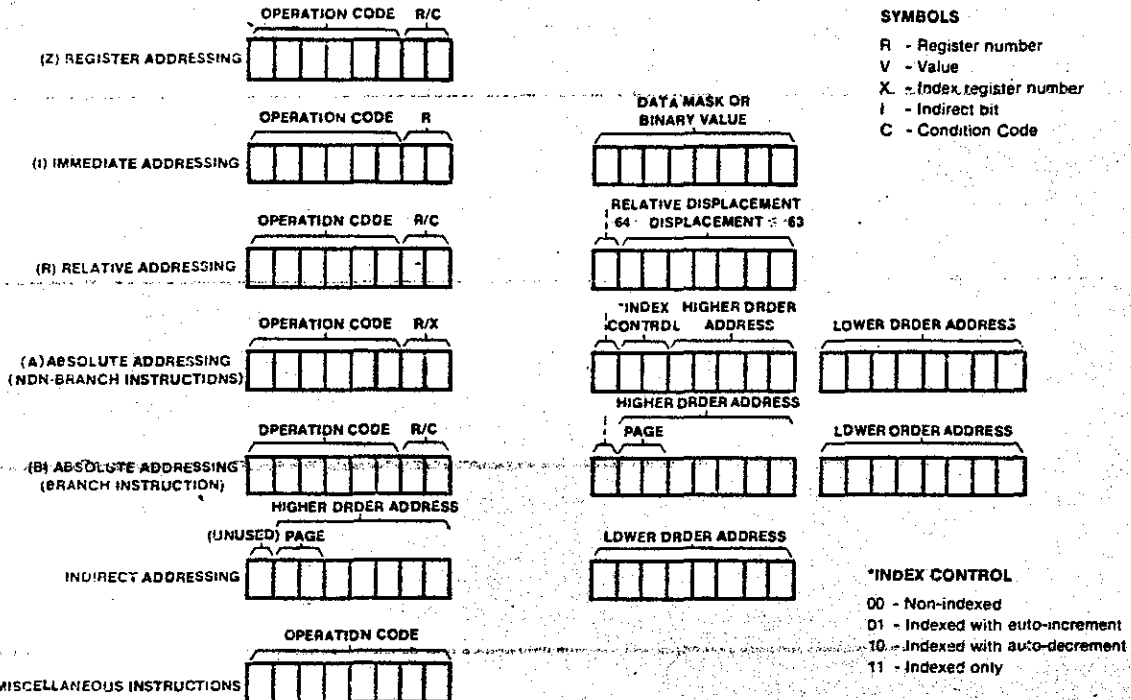
3-bit operation code (the r/c field required)

PIN CONFIGURATION



FOLD LINE

ADDRESSING MODES



PSU

7	6	5	4	3	2	1	0
S	F	II	Nbr Used	Nbr Used	SP2	SP1	SP0

S - Sense

F - Flag

II - Interrupt Inhibit

SP2 - Stack Pointer Two

SP1 - Stack Pointer One

SP0 - Stack Pointer Zero

PSL

7	6	5	4	3	2	1	0
CC1	CC0	IDC	RS	WC	OVF	COM	C

CC1 - Condition Code One

CC0 - Condition Code Zero

IDC - Interdigit Carry

RS - Register Bank Select

WC - With/Without Carry

OVF - Overflow

COM - Logical/Arith Compare

C - Carry/Borrow

DIRECT RELATIVE ADDRESSING—SECOND BYTE

+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
N	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71	
+	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
N	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	70	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62	61
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
N	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
-	60	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52	51
+	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
N	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
-	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42	41

INDIRECT RELATIVE ADDRESS: (80) 16 TO DISPLACEMENT

FOLD LINE

OPERATION CODES IN NUMERICAL ORDER*

0 - 7F			80 - FF		
00-03	LODZ	(1)	80-83	ADDZ	(1)
04-07	LODI	(2)	84-87	ADDI	(2)
08-0B	LODR	(2)	88-8B	ADDR	(2)
0C-0F	LODA	(3)	8C-8F	ADDA	(3)
10-11	---	---	90-91	---	---
12	SPSU	(1)	92	LPSU	(1)
13	SPSL	(1)	93	LPST	(1)
14-17	RETC	(1)	94-97	DAR	(1)
18-1B	BCTR	(2)	98-9A	BCFR	(2)
1C-1F	BCTA	(3)	9B	ZBRR	(2)
20-23	EORZ	(1)	9C-9E	BCFA	(3)
24-27	EORI	(2)	9F	BXA	(3)
28-2B	EORR	(2)	A0-A3	SUBZ	(1)
2C-2F	EORA	(3)	A4-A7	SUBI	(2)
30-33	REDC	(1)	A8-AB	SUBR	(2)
34-37	RETE	(1)	AC-AF	SUBA	(3)
38-3B	BSTR	(2)	B0-B3	WRTC	(1)
3C-3F	BSTA	(3)	B4	TPSU	(2)
40	HALT	(1)	B5	TPSL	(2)
41-43	ANDZ	(1)	B6-B7	---	---
44-47	ANDI	(2)	B8-BA	BSFR	(2)
48-4B	ANDR	(2)	BB	ZBSR	(2)
4C-4F	ANDA	(3)	BC-BE	BSFA	(3)
50-53	RRR	(1)	BF	BSXA	(3)
54-57	REDE	(2)	C0	NOP	(1)
58-5B	BRNR	(2)	C1-C3	STRZ	(1)
5C-5F	BRNA	(3)	C4-C7	---	---
60-63	IORZ	(1)	C8-CB	STRB	(2)
64-67	IORI	(2)	CC-CF	STRA	(3)
68-6B	IDRR	(2)	D0-D3	RRL	(1)
6C-6F	IORA	(3)	D4-D7	WRTI	(2)
70-73	REDD	(1)	08-DB	BIRR	(2)
74	CPSU	(2)	OC-DF	BIRA	(3)
75	CPSL	(2)	E0-E3	COMZ	(1)
76	PPSU	(2)	E4-E7	COMI	(2)
77	PPSL	(2)	E8-EB	COMR	(2)
78-7B	BSNR	(2)	EC-EF	COMA	(2)
7C-7F	BSNA	(3)	F0-F3	WRTD	(1)
			F4-F7	TMI	(2)
			F8-FB	BDRR	(2)
			FC-FF	BDRA	(3)

NOTE

*The number enclosed in parentheses is the number of bytes in the instruction.

HEXADECIMAL—DECIMAL CONVERSION

Hexadecimal Columns			
4	3	2	1
HEX = DEC	HEX = DEC	HEX = DEC	HEX = DEC
C 0	0 0	0 0	0 0
1 4,096	1 256	1 16	1 1
2 8,192	2 512	2 32	2 2
3 12,288	3 768	3 48	3 3
4 16,384	4 1,024	4 64	4 4
5 20,480	5 1,280	5 80	5 5
6 24,576	6 1,536	6 96	6 6
7 28,672	7 1,792	7 112	7 7
8 32,768	8 2,048	8 128	8 8
9 36,864	9 2,304	9 144	9 9
A 40,960	A 2,560	A 160	A 10
B 45,056	B 2,816	B 176	B 11
C 49,152	C 3,072	C 192	C 12
D 53,248	D 3,328	D 208	D 13
E 57,344	E 3,584	E 224	E 14
F 61,440	F 3,840	F 240	F 15
MOST SIGNIFICANT BYTE		LEAST SIGNIFICANT BYTE	

SIGNETICS SALES OFFICES

AURORA, COLORADO—9224
Telephone: (303) 751-5011

CHERRY HILL, NEW JERSEY—9227
Telephone: (609) 665-5071

COLUMBIA, MARYLAND—8227
Telephone: (301) 730-8100
(301) 730-8101

DALLAS, TEXAS—8229
Telephone: (214) 661-1296

EDINA, MINNESOTA—8232
Telephone: (612) 835-7455

HUNTSVILLE, ALABAMA—9227
Telephone: (205) 533-5800

INGLEWOOD, CALIFORNIA—9225
Telephone: (213) 670-1101
(213) 391-7146

IRVINE, CALIFORNIA—9225
Telephone: (714) 833-8980
(213) 924-1668

PHOENIX, ARIZONA—9229
Telephone: (602) 265-4444

PISCATAWAY, NEW JERSEY—9221
Telephone: (201) 981-0123

POMPANO BEACH, FLORIDA—9227
Telephone: (305) 782-8225

ROLLING MEADOWS, ILLINOIS—9232
Telephone: (312) 259-8300

SAN DIEGO, CALIFORNIA—9225
Telephone: (714) 560-0242

SUNNYVALE, CALIFORNIA—9224
Telephone: (408) 736-7565

WAPPINGERS FALLS, NEW YORK—9221
Telephone: (914) 297-4074

WICHITA, KANSAS—9232
Telephone: (316) 683-5652

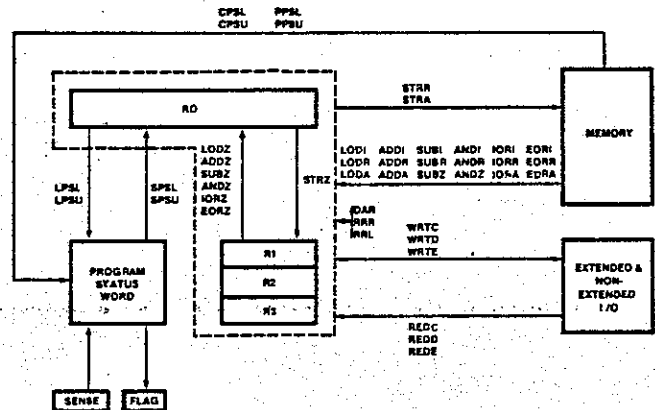
WOBBURN, MASSACHUSETTS—9223
Telephone: (617) 933-8450

WOODBURY, LONG ISLAND,
NEW YORK—9221

Telephone: (516) 364-9100

WORTHINGTON, OHIO—9231
Telephone: (614) 868-7143

2650 DATA TRANSFER INSTRUCTIONS



For further information refer to the 2650 Manual.

Signetics

a subsidiary of U.S. Philips Corporation

Signetics Corporation
PO Box 9062
251 East Avenue Avenue
Sunnyvale, California 94086
Telephone: 408-736-7100

SUGGESTED SOLUTION TO PROBLEMS ON PRECEDING PAGE; Start Addr = '80'

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	0080	82			PROB1	ADDZ	R2	SINGLE STEP ALL INS -
2	1	43			PROB2	ANDZ	R3	TRUCTIONS. PREDICT RE-
3	2	21			PROB3	EORZ	R1	SULT PRIOR TO EXECU-
4	3	62			PROB4	IORZ	R2	TING
5	4	02			PROB5	L0DZ	R2	MOVE R2 → R0, THEN
6	5	21				EORZ	R1	EXCL OR WITH R1, then
7	6	C2				STRZ	R2	store result in R2
8	7	A3			PROB6	SUBZ	R3	
9	8	23			PROB7	EORZ	R3	
10	9	81			PROB8	ADDZ	R1	
11	008A	62			PROB9	IORZ	R2	

PROBLEM	PREDICTION	OBSERVED RESULT
1	R0=69	same
2	R0=12	same
3	R0=C7	same
4	R0=BA	same
5	R2=E9	same
6	R0=56	same
7	R0=CC	same
8	R0=23	same
	Carry (PSL Bit 0)=1	same
9	R0=C6	same

{ ON INSPECTION OF PSL,
DISPLAY SHOULD HAVE BEEN
PL = 49 = 01001001

↑ CARRY

DIRECTION:

1. If the observed results varied from your prediction, check your work. Prepare and execute other examples until you can predict, without error, the results of these instructions. This practice is invaluable especially when you are involved in debugging a complex programmed algorithm for your application.
2. Go on to the next page of documentation.

EXERCISE 2MEMORY DATA CONTROL OPERATIONS

INTRODUCTION: In this exercise, you'll first load all of user memory (loc '00' to '1FF') with an incrementing pattern. Then, you'll clear selected blocks of memory using the routines CLRM1 and CLRM2 listed in this section. At specific points in the exercise, listen to the tape for further commentary. If you are unsure of the "INSTRUCTORS' manual entry procedures, consult the appropriate sections in the INSTRUCTOR REFERENCE MANUAL or refer to Module I; pages 56 and following.

PROCEDURE:

1. Using FAST PATCH mode, enter the following routine "INCPTRN" into the INSTRUCTOR's SMI RAM, starting at location H '1780'.

21	1780	20			INCPTRN	EORZ	RO	CLEAR RO, then store in-
22		CC	20	00	AGAIN1	STRA, RO	LO256, RO, +	cremented contents of RO
23		5B	7B			BRNR, RO	AGAIN1	in 1 st 256 Bytes of memory
24		CC	21	00	AGAIN2	STRA, RO	H1256, RO, +	Then repeat for 2 nd 256
25		5B	7B			BRNR, RO	AGAIN2	Bytes of memory. Then
26		B0				WRTC,	RO	exit to MONITOR

2. Before executing this program, listen to a brief commentary on audio tape.
3. Inspect 4 or 5 locations in USER memory (addresses '0'-'1FF'). Observation: data is variable - no logical pattern.
4. Set the Program Counter to address '1780' and depress RUN to execute.
5. Repeat step C (locations 0-5 and locations 'FE' through '104'. Observation: _____)
6. Listen to the tape for further instructions, then perform step 7.
7. In the routine "CLRM1", code the indicated instructions, then use FAST PATCH mode to load USER memory. Start address is location '100'. Compare your code to the suggested solution on the next page. Do not execute until instructed to do so.

STRTADDR =H'20'. CLEAR 32₁₀ BYTES.

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	'100'				CLRM1	EORZ	RO	clear pattern generator RO
2						STAZ	R3	and INDEX: R3. then store
3		CF	20	1F	AGAIN	STRA, RO	STADR-1, R3, +	clear pattern in memory
4		E7	20			COMI, R3	32	block from start address
5		98	79			BCFR, EQ	AGAIN	(loc. 20). Count is 32 (H'20'
6						WRTC	R3	locations. Exit to MONITOR
7								when finished.

SUGGESTED SOLUTION:

ROUTINE "CLRM1"

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	'100'	20			CLRM1	EORZ	R0	clear pattern generator R0
2	1	C3				STAZ	R3	and INDEX: R3, then store
3	2	CF	20	1F	AGAIN	STRA, R0	STADR-1, R3, +	clear pattern in memory
4	5	E7	20			COMI, R3	32	block from start address
5	7	9B	79			BCFR, EQ	AGAIN	(loc. 20), Count is 32 (H'20')
6	109	B3				WRTC	R3	locations. Exit to MONITOR
7								when finished.

8. Listen to tape 5B for a brief commentary.

Notes: _____

9. Set the Program Counter to location '100' and execute routine "CLRM1" by depressing **RUN**.
10. Inspect memory locations '1E' through '42'. Identify the LOCATION of the first and last bytes cleared.

OBSERVATION: First byte cleared at loc '____'. Last byte at loc '____'.

11. Listen to audio tape for a brief commentary.
12. Clear other blocks of memory as specified in Table 2 (below).

TABLE 2

ROUTINE "CLRM1" OPERATIONS

FROM START ADDRESS:	BYTES TO CLEAR (decimal)	IMPLEMENTATION: CHANGE INSTRUCTION CODE AT LOCATION TO READ					LAST ADDRESS
		'102'	'102'	'104'	'105'	'106'	
'54'	16	CF	20	53	E7	10	'____'
'E2'	25	CF	20	E1	E7	19	'____'
'155'	60	CF	21	54	E7	3C	'____'
'1F1'	10	CF	21	F0	E7	0A	'____'

DIRECTION: Go right on to the next page.

- a. Use the "INSTRUCTOR's" MEMORY DISPLAY and ALTER command to implement desired changes in the memory.
(ALTERNATIVE: Use MEMORY FAST PATCH).
- b. Set "CLRM1"s start address and execute.
- c. Verify that data has been zeroed in the specified locations.
- d. In Table 2, indicate the last (upper) address in memory in which the contents are zero after execution of ROUTINE "CLRM1". Use Table 2 on the preceding page.
- e. After comparing your solution to that provided on this page, go on to Exercise 3.

SOLUTION: CORRECT LAST ADDRESS INDICATIONS

TABLE 2

ROUTINE "CLRM1" OPERATIONS

FROM START ADDRESS:	BYTES TO CLEAR (decimal)	IMPLEMENTATION: CHANGE INSTRUCTION CODE AT LOCATION TO READ					LAST ADDRESS
		'102'	'102'	'104'	'105'	'106'	
'54'	16	CF	2D	53	E7	1D	'00 63'
'E2'	25	CF	20	E1	E7	19	'00 FA'
'155'	60	CF	21	54	E7	3C	'01 90'
'1F1'	10	CF	21	F0	E7	0A	'01 FA'

////////////////////////////////////

EXERCISE 3

IMMEDIATE ADDRESSED INSTRUCTIONS - FORMAT IINTRODUCTION:

All immediate addressed instructions are 2 bytes in length. The first byte of the Immediate-addressed instruction contains the opcode (6 bits) and operand register designation (2 bits). The second byte of the immediate instruction contains 8 bits of data which is used as an argument during instruction execution. This constant is defined as a LOGICAL MASK when a logical instruction is specified. When arithmetic operations are specified, the 2nd byte is a signed 2's COMPLEMENTED BINARY NUMBER with value equal to $-128 < N < +127$.

APPLICATION:

Usage of the immediate addressed instruction represents the easiest way to access new data from memory for use by the microprocessor. By its nature, however, this data is:

- (1) Fixed ... the pattern can not be varied since it is stored in the control program's ROM.

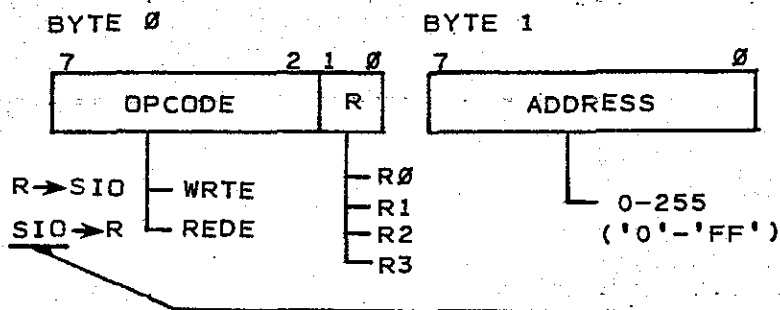
Exception: When employed in user programs stored in ("INSTRUCTOR") RAM, data in a Format I instruction may be modified by execution of other appropriate instructions. This method was employed in order to MINIMIZE the amount of memory required by the program "CRAPGAME".

- (2) Single Byte ... length of data field is maximum of 8 bits. Data organized in tables or blocks can not be accessed via Format I instruction execution.

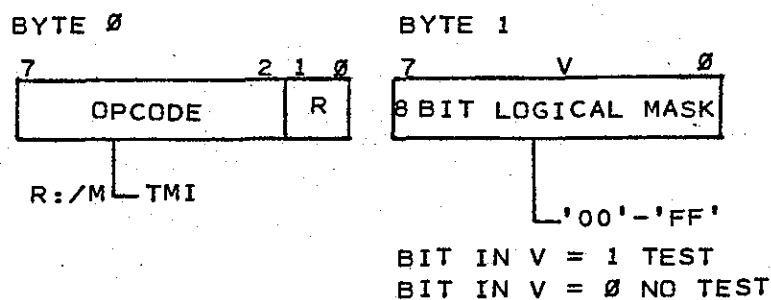
However, the cost of storing data in ROM when practical, is lower per byte than that for RAM. Further, during the debugging process, the programmer can determine the value of programmed immediate data easily by checking its value in the instruction. This contrasts with more involved methods of memory data access, in which the operands define the address of data and not the data value itself.

DIRECTION:

Access the next page in this module and study it for a few minutes. Then listen to tape 5B.



WHERE SIO = SELECTED
EXTENDED I/O PORT DE-
FINED BY ADDRESS IN BYTE 1



I = VALUE IN IMMEDIATE FIELD
R = DESIGNATED REG
SIO = ADDRESSED I/O PORT
- = MINUS
+ = PLUS
Δ = LOGICAL AND
∇ = INCLUSIVE OR
+ = EXCLUSIVE OR
:: = COMPARE WITH
:/ = TEST FOR LOGICAL
ACTIVITY THRU
MASK

page 18

PROCEDURE:

1. Place the I/O SELECTION SWITCH in the EXTENDED position.
2. Code and load the program "FORM-I" into memory at location '00'. Use the 2650 PROGRAMMING REFERENCE (HARD) CARD as much as possible while referring to other documentation only as necessary.
3. Compare your code to that provided on the next page. Do not execute this program until directed to do so.

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	0000				FORM I	LODI, R1	H'0F'	Load data constants into
2						LODI, R2	H'83'	R1, R2, and R3 as indicated
3						LODI, R3	H'7A	Now, add constant '83'
4						ADDI, R2	26	to 26 ₁₀ = result. Sub-
5						SUBI, R2	48	tract 48 ₁₀ = 2 ND result.
6						IORI, R1	H'60'	Now, incl. or constant '0F'
7						LODZ	R1	with H'60'. this result into
8						EORI, R0	H'46'	R0. Xor it with H'46
9						ADDZ	R3	and add '7A' constant.
10						ANDI, R0	H'72'	AND the result with '72'
11						STRZ	R1	and store in R1. Then
12						IORI, R0	H'55'	incl. or R0 with H'55'.
13						WRTE, R0	PORT 7	Then show R0, R1, R2,
14						WRTE, R1	PORT 7	and R3 results on para-
15						WRTE, R2	PORT 7	llel I/O LEDs.
16						WRTE, R3	PORT 7	
17					GETI/O	REDE, R3	PORT 7	Now, read up parallel I/O
18						WRTE, R3	PORT 7	switches & display that
19		FD	00	21	DELAY	BDRA, R1	DELAY	pattern on LEDs until you
20		1F	00	1D		BCTA, UN	GETI/O	depress MON key. Change
21								switch pattern as desired.

SUGGESTED CODE

PROGRAM FORM-I

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION	
		B0	B1	B2		OPCODE	OPERANDS
1	0000	05	0F		FORM I	LODI, R1	H'0F'
2		06	83			LODI, R2	H'83'
3		07	7A			LODI, R3	H'7A'
4		86	1A			ADDI, R2	26
5		A6	30			SUBI, R2	48
6		65	60			IORI, R1	H'60'
7		01				LODZ	R1
8		24	46			EORI, R0	H'46'
9		83				ADDZ	R3
10		44	72			ANDI, R0	H'72'
11		C1				STRZ	R1
12		64	55			IORI, R0	H'55'
13		D4	07			WRTE, R0	PORT 7
14		D5	07			WRTE, R1	PORT 7
15		D6	07			WRTE, R2	PORT 7
16		D7	07			WRTE, R3	PORT 7
17		57	07		GETI/O	REDE, R3	PORT 7
* 18		D7	07			WRTE, R3	PORT 7
19		FD	00	21	DELAY	BDRA, R1	DELAY
20		1F	00	1D		BCTA, UN	GETI/O
21							

OBSERVATIONS

R0	R1	R2	R3	Carry 0 or 1
XX		XX	XX	
XX	XX		XX	
XX	XX	XX		
XX	XX		XX	
XX	XX		XX	
XX		XX	XX	
		XX	XX	
	XX	XX	XX	
	XX	XX		
	XX	XX	XX	
		XX	XX	
	XX	XX	XX	

PSL Bit 0

COMPARE PARALLEL I/O LED
DISPLAY WITH APPROPRIATE
REGISTER CONTENTS.

4. After comparing your code to that illustrated above, listen to the tape 5B for a short commentary.
5. Use the "INSTRUCTOR's" DISPLAY and ALTER REGISTERS command to
 - (1) Set the Program Status Word (Lower - PSL) to H'08'.
 - (2) Set the START ADDRESS of program "FORM I" into the Program Counter.
6. Execute each instruction in **STEP** mode. After each STEP inspect the working registers and enter their values as indicated in the "Observation" columns (above). Also, inspect and indicate the LOGICAL STATUS of the CARRY bit (PSL Bit 0). Correct entries are provided on the next page.
- * 7. After executing the instruction at line 18 (above), depress **RUN** to execute the rest of program "FORM-I" at speed. Modify the settings of the Parallel I/D selection switches. Notice how the program permits interaction between the input switches and output LED display.
8. After performing step 7 to your satisfaction, depress **MDN** to exit from the program, and go on to the next page.

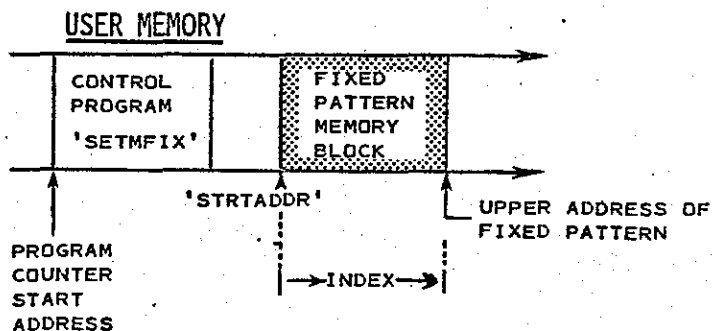
EXERCISE 4INTRODUCTION:

It may be useful for you to set a particular data pattern into a defined block of memory. There are several routines available for this, one of which is illustrated below.

This routine permits you to set a fixed pattern into X locations of user RAM, starting at address 'YY'.

Assignments:

R0 = Pattern Generator
R3 = Index into memory from STARTADDRESS YY
R2 = Length of the memory block, in bytes.

CORRECT ENTRIES-PRECEDING PAGEOBSERVATIONS

R0	R1	R2	R3	Carry 0 or 1
XX	0F	XX	XX	
XX	XX	83	XX	
XX	XX	XX	7A	
XX	XX	9D	XX	0
XX	XX	6C	XX	1
XX	6F	XX	XX	
6F	6F	XX	XX	PSL Bit 0
29	XX	XX	XX	
A4	XX	XX		0
20	XX	XX	XX	
20	20	XX	XX	
75	XX	XX	XX	

NOTE: "BDRROW" was generated to indicate result is a positive number.

PROCEDURE:

1. Given that a pattern ('D5') is to be loaded into 27₁₀ bytes of memory, starting at location '49', use the space provided below to code program "SETMFIX" into memory at location '20' to accomplish this.
2. Execute the program "INCPTRN" at location '1780' (reference page 14 in this module) to load user memory with an incremented pattern.
3. Compare your program with the solution provided on the next page.

22				SETMFIX	LODI, R0		Set desired pattern into R0
23					LODI, R3		then zero the index and set
24					LODI, R2		number of bytes to be loaded.
25		CF	20	48	AGAIN	STRA, R0 STADR-1, R3, +	Store the fixed pattern
26		FA	7B			BDRR, R2 AGAIN	in spec. memory. When finished,
27						WRTC R0	exit to MONITOR.

CODING SOLUTION

PROGRAM "SETMEIX"

22	0020	04	D5	SETMEIX	LODI, R0	H'D5'	Set R0 pattern generator
23	2	07	00		LODI, R3	0	then zero the index and
24	4	06	1B		LODI, R2	27	set # of bytes to be loaded
25	6	CF	20	AGAIN	STRA, R0	STATADDR-1, R3, +	Now, store the fixed pattern
26	9	FA	7B		BDRR, R2	AGAIN	in defined memory. When
27	B	BO			WATC	R0	finished, exit to MONITOR.

4. Listen to tape 5B for a brief commentary on the instructions in lines 25-27, then proceed with the next step.

NOTES: _____

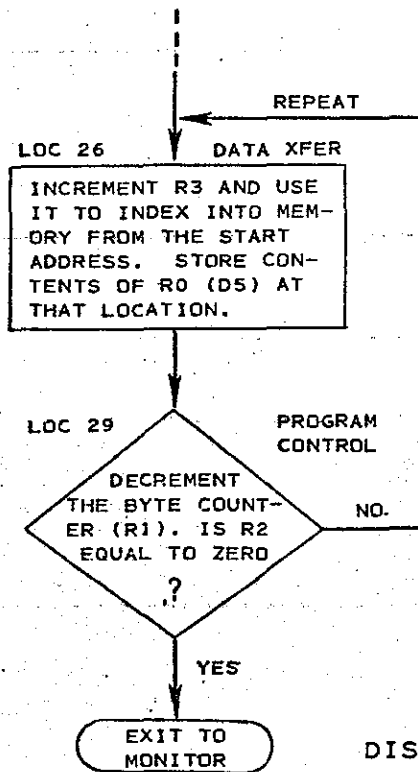
5. Set the Program Counter to a program execution start address at location '20'.
6. Execute the program in STEP mode. After depressing STEP each time (15 times) enter your observation in Table 3 (below) as required. Then compare your entries with those provided on the next page.

TABLE 3

AFTER EXECUTION OF THE INSTRUCTION AT LOCATION:	REGISTER CONTENTS AS FOLLOWS:			VERIFY CONTENTS OF MEMORY LOCATIONS:
	R0	R2	R3	
20		XX	XX	
22	XX	XX		'49' = '4A' = '4B' =
24	XX		XX	
26	XX	XX		'49' = '4A' = '4B' =
29	XX		XX	
26	XX	XX		'49' = '4A' = '4B' =
29	XX		XX	
26	XX	XX		'49' = '4A' = '4B' =
29	XX		XX	
26	XX	XX		'4C' = '4D' = '4E' =
29	XX		XX	
26	XX	XX		'4C' = '4D' = '4E' =
29	XX		XX	
26	XX	XX		'4C' = '4D' = '4E' =
29	XX		XX	

OBSERVATION COMPARISON

TABLE 3



DISPLAY = 'HELLO'

AFTER EXECUTION OF THE INSTRUCTION AT LOCATION:	REGISTER CONTENTS AS FOLLOWS:			VERIFY CONTENTS OF MEMORY LOCATIONS:
	R0	R2	R3	
20	05	XX	XX	
22	XX	XX	00	'49'='49' '4A'='4A' '4B'='4B'
24	XX	18	XX	
26	XX	XX	01	'49'='D5' '4A'='4A' '4B'='4B'
28	XX	1A	XX	
26	XX	XX	02	'49'='D5' '4A'='D5' '4B'='4B'
28	XX	19	XX	
26	XX	XX	03	'49'='D5' '4A'='D5' '4B'='D5'
28	XX	18	XX	
26	XX	XX	04	'4C'='D5' '4D'='4D' '4E'='4E'
28	XX	17	XX	
26	XX	XX	05	'4C'='D5' '4D'='D5' '4E'='4E'
28	XX	16	XX	
26	XX	XX	06	'4C'='D5' '4D'='D5' '4E'='D5'
28	XX	15	XX	

7. Listen to a brief commentary on tape 5B. At the tone, go on to the next step.
8. Depress **RUN** to execute the rest of the program. Then verify that the pattern 'D5' is stored in memory from locations '49' through '63'.
9. Reinforce your understanding of this program by defining different patterns and memory blocks in program "SETMFIX". Execute then verify the results. Then go on to Exercise 5 (next page).

EXERCISE 5RELATIVE ADDRESSED INSTRUCTIONS - FORMAT RINTRODUCTION:

One of the most powerful addressing modes available to the micro-processor programmer is that of RELATIVE ADDRESSING. The power of an addressing mode is measured by its number of variations, possible. These variations, all of which will be discussed in this section of the course, include:

DIRECT RELATIVE ADDRESSING - Data Transfer and computation

- Program Control

*Conditional Branches based on register contents.

results of previous instruction execution involving data transfer, computation, or comparison.

*Conditional Branches to sub-routines.

*Page Zero-Byte Zero referenced branches.

INDIRECT RELATIVE ADDRESSING - All of the above, but with the capability to access any location in memory without regard to page or relative address boundary limits.

Altogether, there are 63 distinct opcodes, representing 18 distinct instruction types which comprise the family of relative addressed instructions.

Relative addressed instructions are 2 bytes in length. The first byte contains the opcode and designates a register or (in certain program control branch instructions) a specified value upon which a decision is to be based. The second byte contains:

1. A direct/indirect address identifier (1 bit).
2. A 7-bit 2's complement binary number to give a RELATIVE ADDRESSING RANGE from -64 (H'40') to +63 (H'3A').

This binary number is used by the microprocessor to calculate an EFFECTIVE address of memory data or instruction execution (the argument) dependent on the instruction opcode type. The EFFECTIVE address is DISPLACED from the location in memory defined by the Program Counter (next instruction address). The value of this displacement is determined by the RELATIVE ADDRESS supplied in the second byte. Since the Program Counter can point to any location in memory, the EFFECTIVE address is said to be RELATIVE to that location (within the range of -64 to +63 memory locations from the Program Counter address).

APPLICATION:

You'll see that the use of limited user memory (such as contained onboard the INSTRUCTOR) is maximized through choice of RELATIVE memory referenced instructions (2 bytes) rather than ABSOLUTE memory referenced instructions whenever possible. However, the very important function of indexing into memory (to access 1 of a group of numbers in a data table, for example) is NOT POSSIBLE using relative addressed instructions. Indexing is restricted to implementation with an ABSOLUTE addressed format.

DIRECTION:

The examples (next page) demonstrate a comparison of memory allocated for a given routine ("DISMES" in the Program 'CRAPGAME') using RELATIVE and ABSOLUTE address modes. Take a few minute to study the memory requirements effected by both methods, then continue.

The same function is performed by both versions of "DISMES".

"DISMES" - ABSOLUTE FORMAT

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION	
		B2	B1	B2		OPCODE	OPERANDS
1	0137	XX			>MESLOC	RES	1
2	8				DSPLYDLY	RES	1
3	9				RUNDSPX	RES	1
4							
5	013A	CE	01	37	DISMES	STRA, R2	>MESLOC-1
6	D	05	02			LODI, R1	2
7	13F	CD	01	39	REPEATM	STRA, R1	RUNDSPX
8	142	CF	01	38	DSPAGN	STRA, R3	DSPLYDLY
9	5	05	17			LODI, R1	<MESLOC-1
10	7	0E	01	37		LODA, R2	>MESLOC-1
11	A	3F	9F	E6		BSTA, UN	*USRDSP
12	14D	0F	01	38		LODA, R3	DSPLYDLY
13	150	E7	00			COMI, R3	0
14	2	14				RETC, EQ	
15	3	E7	01			COMI, R3	1
16	5	1C	01	5B		BCTA, EQ	CONTDSP
17	8	FF	01	42		BDRA, R3	DSPAGN
18	9	0D	01	39	CONTDSP	LODA, R1	RUNDSPX
19	15E	FD	01	62		BDRA, R1	RUNAGN
20	161	17				RETC, UN	
21	2	07	7F		RUNAGN	LODI, R3	7F
22	4	1F	01	3F		BCTA, UN	REPEATM
23					* LAST	ADDRESS	= H'166'

"DISMES" - RELATIVE FORMAT

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION	
		B2	B1	B2		OPCODE	OPERANDS
1	0137	XX			>MESLOC	RES	1
2	8	XX			DSPLYDLY	RES	1
3	9	XX			RUNDSPX	RES	1
4							
5	013A	CA	7B		DISMES	STRA, R2	>MESLOC-1
6	C	05	02			LODI, R1	2
7	13E	C9	79		REPEATM	STRA, R1	RUNDSPX
8	140	CB	76		DSPAGN	STRA, R3	DSPLYDLY
9	2	05	17			LODI, R1	<MESLOC-1
10	4	0A	71			LODR, R2	>MESLOC-1
11	6	BB	E6			ZBSR	*USRDSP
12	8	0B	6E			LODR, R3	DSPLYDLY
13	A	E7	00			COMI, R3	0
14	C	14				RETC, EQ	
15	D	E7	01			COMI, R3	1
16	14F	18	02			BCTR, EQ	CONTDSP
17	151	FB	6D			BDRA, R3	DSPAGN
18	3	09	64		CONTDSP	LODR, R1	RUNDSPX
19	5	F9	01			BDRA, R1	\$+3
20	7	17				RETC, UN	
21	8	07	7F		RUNAGN	LODI, R3	7F
22	A	1B	62			BCTR, UN	REPEATM
23					* LAST	ADDRESS	= H'15B'

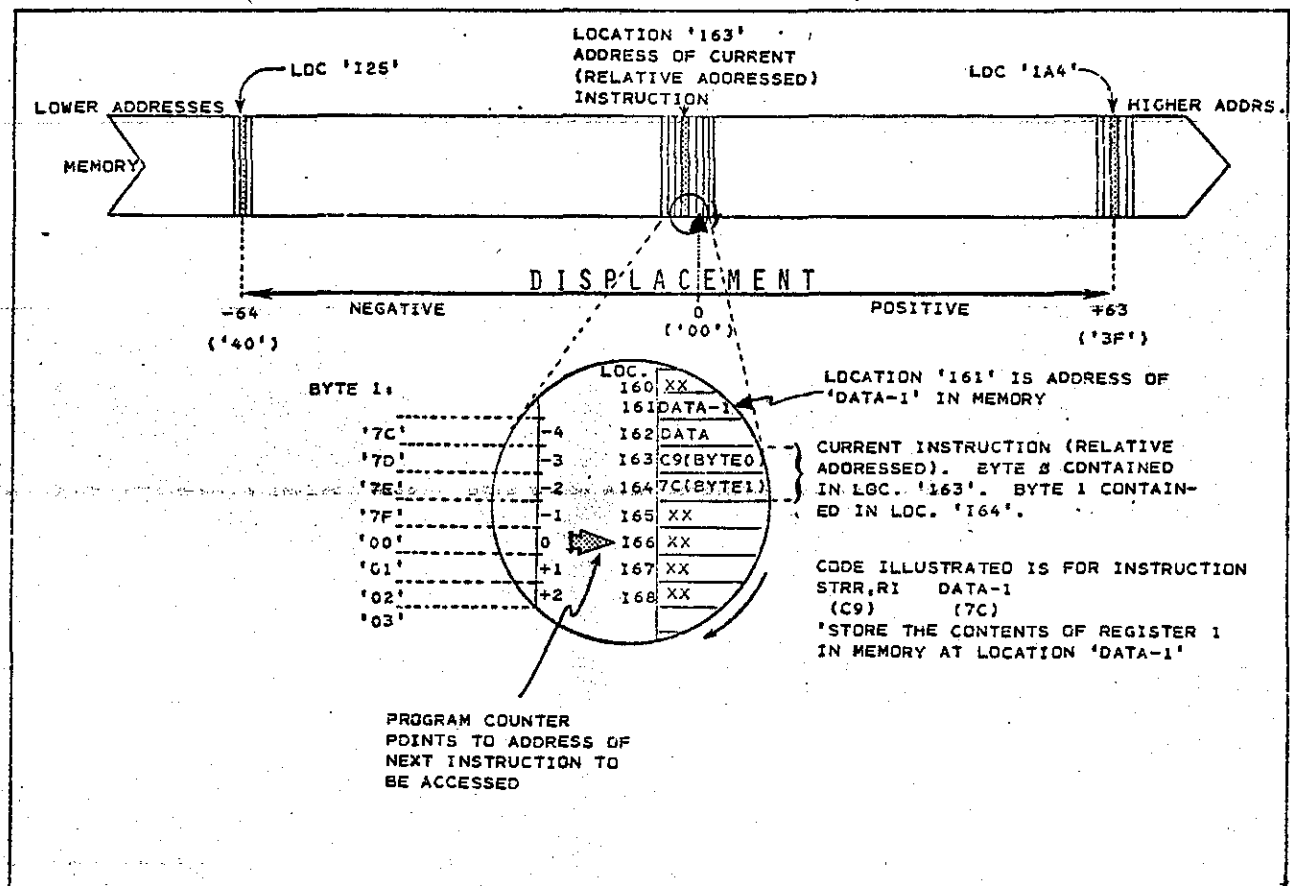
DIRECTION:

Listen to tape 5B for a brief commentary, then go on to the next page when directed to do so.

NOTES:

FIGURE 3

CONCEPT OF RELATIVE ADDRESSING



NOTES:

DIRECTION:

In the following problems, indicate in terms of absolute addresses, the possible range of relative displacement, given the address contained in the Program Counter.

1. PC='0057'. Absolute range from loc '____' to loc '____'
2. PC='019B'. Absolute range from loc '____' to loc '____'
3. PC='00D3'. Absolute range from loc '____' to loc '____'

Given the address of Byte 0 in relative addressed instructions, calculate the maximum limits of displacement possible in terms of absolute effective addresses.

4. Byte 0 at loc. '0078'. Effective addresses: '____' to '____'.
 5. " " " " '0055'. " " '____' to '____'.
 6. " " " " '00AF'. " " '____' to '____'.

Compare your answers to those provided below.

SOLUTIONS TO EFFECTIVE ADDRESS DEFINITION

PC(HEX)	RANGE	
	LOCATION (LO) TO LOCATION (HI)	
'0057'	'0017'	'0096'
'019B'	'015B'	'01DA'
'00D3'	'0093'	'0112'
LOC OF BYTE 0	EFFECTIVE RELATIVE ADDRESSES	
'0078'	'003A'	TO '00B9'
'0055'	'0017'	TO '0096'
'00AF'	'0071'	TO '00F0'

DIRECTION:

After comparing your responses to those provided above, listen to tape 6A for a brief commentary.

RELATIVE ADDRESS TABLE

+	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
N	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-		7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71
+	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
N	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
-	70	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62	61
+	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
N	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
-	60	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52	51
+	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
N	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
-	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42	41

INDIRECT RELATIVE ADDRESS: Add H'80' TO DISPLACEMENT

EXAMPLE 1

EXAMPLE 2

EXAMPLE 3

VALUE OF BYTE 1 (HEX)
IF DISPLACEMENT IS
POSITIVE; I.E., NEXT
INSTRUCTION ADDR
PRECEDES EFFECTIVE
ADDR.

MAGNITUDE OF
DISPLACEMENT
(DECIMAL)

VALUE OF BYTE 1 (HEX)
IF DISPLACEMENT IS
NEGATIVE; I.E.,
EFFECTIVE ADDRESS
PRECEDES NEXT
INSTR. ADDR.

DIRECTION:

Using the RELATIVE ADDRESSING TABLE in the routine "FORM-R" (below), code the instructions as indicated. The correct solution is provided on the next page.

		DIRECT RELATIVE ADDRESSING - SECOND BYTE															
+	H	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
-	H	7F	7E	7D	7C	7B	7A	79	78	77	76	75	74	73	72	71	70
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
+	H	20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
-	H	60	6F	6E	6D	6C	6B	6A	69	68	67	66	65	64	63	62	61
		50	5F	5E	5D	5C	5B	5A	59	58	57	56	55	54	53	52	51
+	H	30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
		48	49	4A	4B	4C	4D	4E	4F	48	49	4A	4B	4C	4D	4E	4F
-	H	50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42	41
		50	4F	4E	4D	4C	4B	4A	49	48	47	46	45	44	43	42	41

INDIRECT RELATIVE ADDRESS: Add H'80' TO DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE FORM-R START ADDR 046DESCRIPTION PRACTICE CODING OFDISPLACEMENT BYTE OF REL. ADDR'DINSTRUCTIONSROUTINE SHEET 1 OF 1

MEMORY LOCATIONS THIS SHEET _____

	ADDRES	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	0046				FORM-R	LODI, R1	H'27'	1st byte in data is DATA
2						LODI, R2	H'33'	2nd " " " " DATA+1
3		0F	01	30	DATA	RES	3	3rd byte in data is DATA+2
4						ADDR, R2	SHARE+1	
5						SUBR, R1	DATA	
6					AGAIN	BDRR, R3	AGAIN	DELAY
7						LODR, R2	DATA-1	
8						LODR, R0	DATA-3	
9						ADDZ	R2	
10						STRR, R0	SHARE+2	
11						LODR, R3	DATA+2	
12						WRITE, R3	PORT7	
13					ANOTHER	SUBI, R3	1	
14						COMR, R3	SHARE+5	
15						BCTR, EQ	NEXT	
16						BCTR, UN	ANOTHER	LOOP
17					NEXT	ANDR, R1	ANOTHER-1	
18						STRR, R1	DATA+2	
19						BCTR, UV	COMPUTE	
20		22	33	44	SHARE	RES	6	6 bytes in SHARE; in order
21		55	66	77				SHARE, SHARE+1.... SHARE+5
22		3F	01	30	COMPUTE	BSTA, UN	SUBR4	
23								* Last addr = H'074
24								
25								
26								
27								

DIRECTION:

After comparing your code to the solution on the right, listen to tape 6A.

NOTES: _____

CODING SOLUTION - ROUTINE "FORM-R"

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION	
		B0	B1	B2		OPCODE	OPERANDS
1	0046	05	27		FORM-R		
2	B	06	33				
3	A	0F	01	30	DATA		
4	D	8A	1E				
5	4F	A9	79				
6	51	FB	7E		AGAIN		
7	3	0A	74				
8	5	08	70				
9	7	B2					
10	8	CB	14				
11	A	0B	70				
12	C	D7	07				
13	5E	A7	01		ANOTHER		
14	60	EB	0F				
15	2	18	02				
16	9	1B	78				
17	C	49	75		NEXT		
18	8	C9	62				
19	A	1B	06				
20	C	22	33	44	SHARE		
21	6F	55	66	77			
22	0072	3F	01	30	COMPUTE		
23							

////////////////////////////////////

EXERCISE 6**PROCEDURE TO EXECUTE ROUTINE FORM-R****INTRODUCTION:**

With a few additions you can execute routine "FORM-R" on the "INSTRUCTOR". Directions are minimal, consistent with your level of experience in "INSTRUCTOR" usage and your ability to reference into the User Guide and 2650 Manuals. Perform the following steps and respond to the questions as indicated in the procedure.

PROCEDURE:EXERCISE 6

1. Use MEMORY FAST PATCH mode to load routine "FORM-R" into memory at location '46'.
2. Use REGISTER DISPLAY and ALTER mode to store H'08' into the PSW (lower). This sets the With Carry bit.
3. USE MEMORY DISPLAY & ALTER mode to enter code into memory at the following locations.

Address 130 'D6' } WRTE,R2 PORT7 displays R2 on parallel
" 131 '07' } I/O LEDs.
" 132 '17' } RETC return from subroutine instruction
called by the unconditional branch to
subroutine instruction at location 72.
The instructions at locations '130' to
'132' comprise a 2 instruction sub-
routine called "SUBR4".

Address 75 'B0' } WRTC,R0 instruction causes exit to the
MONITOR after SUBR4 is finished.

4. Set a start address for "FORM-R" execution at location '46'.
5. Execute routine "FORM-R" in STEP mode to the BDRR instruction at location '51'. Read the NOTE before executing the BDRR instruction.

NOTE: This Branch on decrementing register relative instruction loops on itself until the contents of R3 are reduced to zero. Note the initial contents of R3 before stepping the BDRR instruction at location '51' the first time.

6. Set a Breakpoint at location '53' and depress RUN. Then, continue on the next page, responding to the questions as indicated.

CAUTION:

After routine "FORM-R" is executed, the contents of some memory locations in the original program are altered. If you wish to repeat the procedure, verify memory occupied by "FORM-R" before performing steps 4 and following. Try to identify exactly which instructions were executed to cause the change in routine "FORM-R"s data.

DIRECTION:

Go right on to the next page.

QUESTION A: What is the value of R3 after the routine has sequenced through memory address '53'. R3= _ _

7. After execution through location '5A' in **STEP** mode, compare the contents of: R0 and memory location '6E'.
R3 and memory location '4C'.

QUESTION B: Are the compared contents (in each case) equal? _____(yes)(no).

If no, state the values: R0= _ _ loc '6E'= _ _
R3= _ _ loc '4C'= _ _

8. Set a BREAKPOINT at location '66', then depress **RUN** .

QUESTION C: What is the value of R3 after the routine has sequenced through memory address '66'? R3= _ _.

Values: R3= ' _ _ '
loc '71'= ' _ _ '

9. Set a BREAKPOINT at location '6A'. Depress **RUN** .

QUESTION E: What is the location of the next instruction to be executed? Location is H' _ _ _ _ '.

10. Depress **STEP** . At this time, the program executes the sub-routine "SUBR4" at location '130'.
11. Depress **STEP** , sequentially noting the addresses of instruction execution.

QUESTION F: Do these compare with the addresses in the routine as modified in step 2? _____(yes)(no).

12. Depress **RUN** :

QUESTION G: How do you know that program execution exited into the MONITOR?

ANSWER: _____

DIRECTION:

Compare your answers with those provided on the next page.

ANSWERS TO QUESTIONS A THROUGH G (PRECEDING PAGE)

A. R3=00

As written, the BDRR instruction repeats its own execution, decrementing R3 until R3=0. Then program execution falls through to the next instruction (at loc '53').

B. YES

Load and Store instruction NEVER modify data.

C. R3='77'

By observation (REGISTER INSPECT command).

D. YES

Same as explanation for answer B.

E. '0072'

This UNCONDITIONAL BRANCH causes program execution to BYPASS the 6-byte "SHARE" data field.

F. YES

An absolute-addressed unconditional Branch to subroutine instruction (loc '72') and the return instruction (loc '132') permitted this program sequence.

G. Display = "HELLO"

Note however that the Program Counter is set to address '0D75', the location of the 'B0' instruction you loaded in step 3.

////////////////////////////////////

DIRECTION:

Go on to the next page. Listen to the audio tape for a short discussion of the RELATIVE ADDRESS HAND-CODE COMPUTER.

////////////////////////////////////

RELATIVE ADDRESS HAND-CODE COMPUTER (RAHCO)

NOTE: *The method for assembly of RAHCO assumes that you have access to BOTH a xerographic copier (Xerox, IBM, SAVIN, or equivalent) and a TRANSPARENCY machine (3M dry heat transfer or equivalent).

1. Detach and make 1 copy each of Tables A and B (pages 34 and 35).
2. Cut out TABLE B-ABSOLUTE ADDRESSES and center mount on a piece of heavy card stock (120 lb. 8-1/2x11 paper is sufficient). (Diagram 2A).
3. Make a transparency of TABLE A-DISPLACEMENT CHART.*
4. In a second piece of card stock, cut a rectangular hole lengthwise and large enough so that all numbers of the DISPLACEMENT transparency are visible. (Diagram 1)
5. Cut the TRANSPARENCY to size, leaving a 1/4" border (MIN) around the numbers.
6. Mount the TRANSPARENCY face-up in back of the window you cut out in step 4. (DIAGRAM 2B)
7. You are now ready to use the RAHCO. Directions follow the detached pages.

DIAGRAM 1

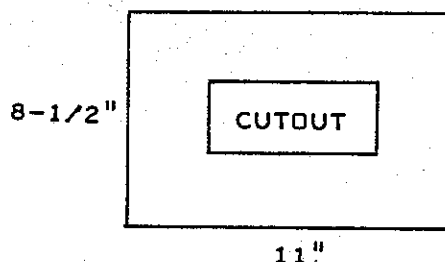


DIAGRAM 2A

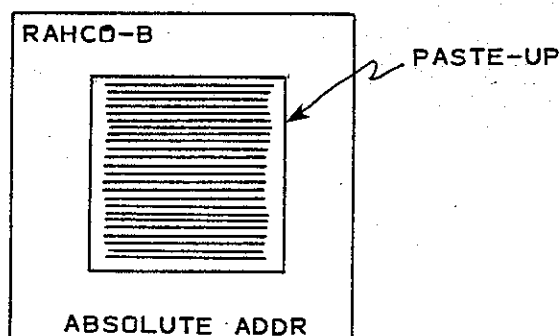
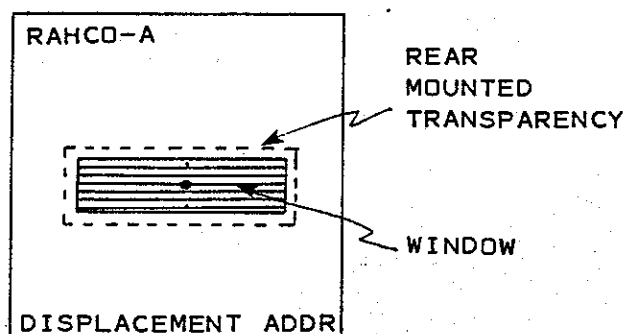


DIAGRAM 2B



*NOTE: If you do not have access to in-house copying facilities, you should request a local print-shop or photo studio to make a 1:1 positive transparency of Table A, the DISPLACEMENT CHART. Use the original of Table B in step 2.

RAHCO - TABLE ARELATIVE ADDRESS
HANDCODE COMPUTER

40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	
61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F	
71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F	
01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	
11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	
21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F	
31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F	

NOTE:

The horizontal parallel lines either side of the box are for alignment purposes only.

RAHCO - TABLE B

RELATIVE ADDRESS HANDCODE COMPUTER

A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	EB	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	6B	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	B5	86	87	88	89	8A	8B	8C	BD	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
BD	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	CB	C9	CA	CB	CC	CD	CE	CF
DD	D1	D2	D3	D4	D5	D6	D7	DB	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	EB	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F

DIRECTIONS FOR USE OF RAHCO (REL. ADDR HAND-CODE COMPUTER)

1. Position RAHCO-A OVER RAHCO-B so that the small box (☐) on RAHCO-A encloses the absolute address (least significant digit pair) of the "next instruction" on RAHCO-B.

NOTE: The microprocessor's Program Counter always points to the "next instruction address". This is the location of the byte immediately following the 2nd byte of the relative addressed instruction.

2. On RAHCO-B (under RAHCO-A), locate the least significant digit pair of the EFFECTIVE ADDRESS that your relative addressed instruction is displacing to.
3. On RAHCO-A, read the DISPLACEMENT digit pair which is located DIRECTLY ABOVE the desired EFFECTIVE ADDRESS digit pair.
4. If the instruction is DIRECT relative addressed, enter the DISPLACEMENT digit pair into BYTE 1 (2nd Byte) of the current relative instruction without modification.

If the instruction is INDIRECT relative addressed, ADD H'80' to the DISPLACEMENT digit pair and enter THAT value into BYTE 1.

NOTE: All digit pairs on RAHCO are HEX values.
No decimal to hex code conversion is necessary.

USEFUL PROGRAMMING HINTS:

1. When coding a block of instructions, prepare and enter the OPCODE bytes first. Place a dot (.) in the APPLICABLE OPERAND bytes →
2. Complete the ADDRESS column, allowing sufficient locations for the uncoded bytes →
3. Calculate ABSOLUTE ADDRESS and RELATIVE DISPLACEMENT OPERANDS based on EFFECTIVE addresses derived in the address column →

1		XX	XX	XX	DATA	RES	3 -
2		C9	.		HINT	STRR, R1	DATA+1
3		89	.		AGAIN	ADDR, R1	NEW
4		C9	.			STRR, R1	NEW
5		1B	.			BCTR, UN	CONTIN
6		XX			NEW	RES	1
7		FB	.		CONTIN	BDRR, R3	AGAIN

1	0037	XX	XX	XX	DATA	RES	3 -
2	A	C9	.		HINT	STRR, R1	DATA+1
3	C	89	.		AGAIN	ADDR, R1	NEW
4	3E	C9	.			STRR, R1	NEW
5	40	1B	.			BCTR, UN	CONTIN
6	2	XX			NEW	RES	1
7	3	FB	.		CONTIN	BDRR, R3	AGAIN

1	0037	XX	XX	XX	DATA	RES	3 -
2	A	C9	7E		HINT	STRR, R1	DATA+1
3	C	89	04		AGAIN	ADDR, R1	NEW
4	3E	C9	02			STRR, R1	NEW
5	40	1B	01			BCTR, UN	CONTIN
6	2	XX			NEW	RES	1
7	3	FB	77		CONTIN	BDRR, R3	AGAIN

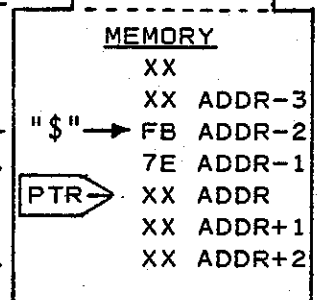
DIRECTION:

Go right on to the next page.

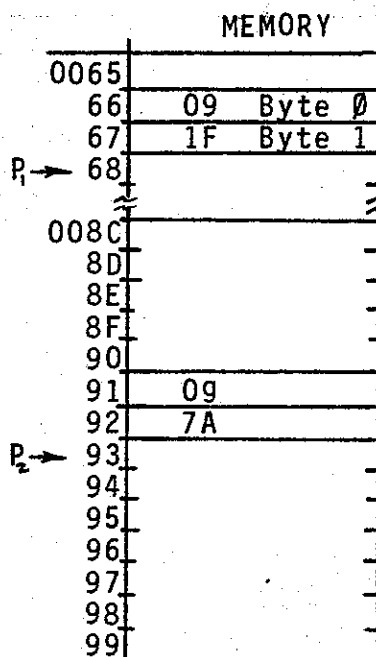
COMPARISON OF LITERAL/SYMBOLIC NOTATION FOR CODING OF BYTE 1
OF RELATIVE ADDRESSED INSTRUCTIONINTRODUCTION:

Good programming practices require that standards be set and adhered to by the programmer. The following examples detail the advantages and limitations involved in preparing the program's instructions sequence literally as compared with various degrees of symbolic notation.

One word about the use of the symbolic "dollar sign" (\$): BDRR,R3 \$
If you have access to the 2650 Assembler, that software development tool translates the \$ to mean the memory location of the CURRENT INSTRUCTION'S BYTE 0. When referenced to RELATIVE-ADDRESSED instructions, Byte 0 precedes the pointer (Program Counter "next instruction address") by 2 locations (diagram at right). Thus, in writing an instruction designed to loop on itself (e.g., BDRR,R3 \$), the relative address, '7E', simply defines a negative displacement of 2 locations.

DIRECTION:

Take a few minutes to study the examples of literal and symbolic notation on this and the following page, then listen to tape 6A for an introduction to the diagrams which follow.

EXAMPLE 1 LITERAL CODING-HEX

LODR,R1 31

"Move contents of address in memory displaced 31 locations from instruction address pointer into R1"

• LIMITATION:

Value in instruction Byte 1 must be rewritten if any instructions are added or deleted between pointer and effective addr. If literal number used in source program, makes no sense when displacement is modified.

EXAMPLE 2 - SYMBOLIC/DECIMAL CODING

008C	
8D	XX
8E	
8F	
90	
91	09
92	7A
P ₂ 93	
94	
95	
96	
97	
98	
99	
9A	
9B	CA
9C	16
0090	
P ₃	
0082	
83	"CTRLB"
84	"CTRLB+1"
85	
86	
87	0A
88	7B
00B9	
P _{3A}	

LDDR,R1 \$ - 4 Where \$ = address of current instruction, byte 0
 = negative displacement
 4 = amount of displacement in decimal

"Move contents of an address displaced - 4 locations from the current instruction into R1."

- **NOTE:** Next instruction pointer is displaced +2 from current instruction. \$ recognized by assembler.

- **LIMITATION:** Same as example 1. Difference between address locations of P and \$ can cause confusion.

- **RECOMMENDED USE:** When relative displacement is a known factor - e.g., in delay loop.

Note: At address 0092, '7A' denotes -6 byte displacement. = \$ -4.

EXAMPLE 3 - FULL SYMBOLIC NOTATION

STRR,R2 CTRLB

"Move the contents of a location in memory defined as CTRLB (CONTROL BUFFER) into R2"
 current displ = $22_{10} = 16_{16}$

- Source program coding (symbols) don't change.
- Tentative, then final object code equivalent can be inserted during debug. No confusion with P/\$ differences.
- Reinforces comment field of structured program.
- Recommended.

EXAMPLE 3A

LDOR,R2 CTRLB+1

"Move the contents of 'CTRLB+1' into R2"

- Current displ is \$ - 3 = -5 = '7B'
- Object code in second byte can be written, then modified easily during debug. Source symbolic label does not change.
- Key: Reserve required bytes for symbolic function in memory.
 - Assembler will assign absolute addrs.
 - Handcoder can " " "

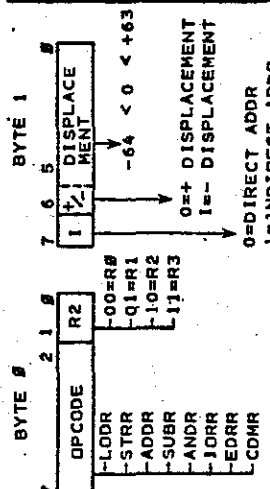
DATA HANDLING INSTRUCTIONS

DIRECTION:

Listen to tape 6A for a brief commentary on relative address data transfer and manipulation instructions. Go on to the next page when instructed to do so.

DIRECT RELATIVE ADDRESSING - SECOND BYTE

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F



INSTRUCTION - SYMBOLIC LOCATION OF 1ST BYTE IS \$-2 BYTE DISPLACEMENT FROM POINTER.

INSTRUCTION ADDR POINTER (POINTS AT DPCODE BYTE OF NEXT INSTRUCTION DURING EXECUTION OF CURRENT INSTRUCTION)

CODE FOR INSTRUCTION LODR, R1 KEYB1+2 IS 09 3E ← +68 LOC FROM PTR

CODE FOR INSTRUCTION STRR, R2 \$-4 IS CA 7A ← -6 LOC FROM POINTER

MICROPROCESSOR BLOCK DIAGRAM

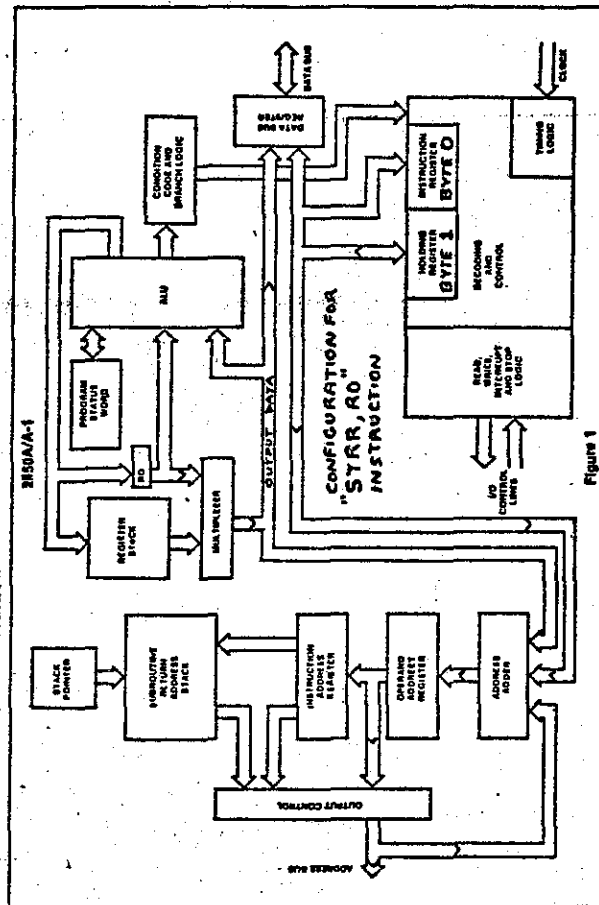
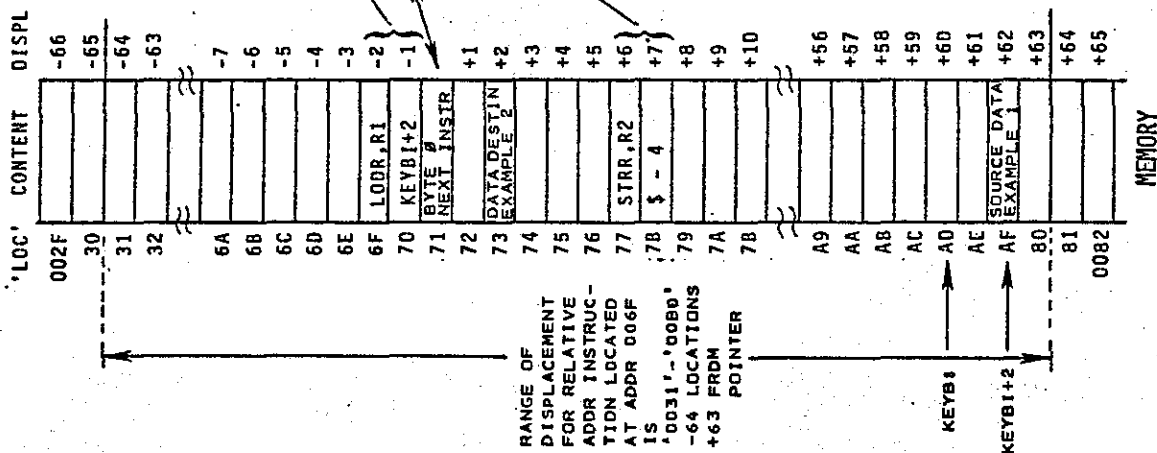
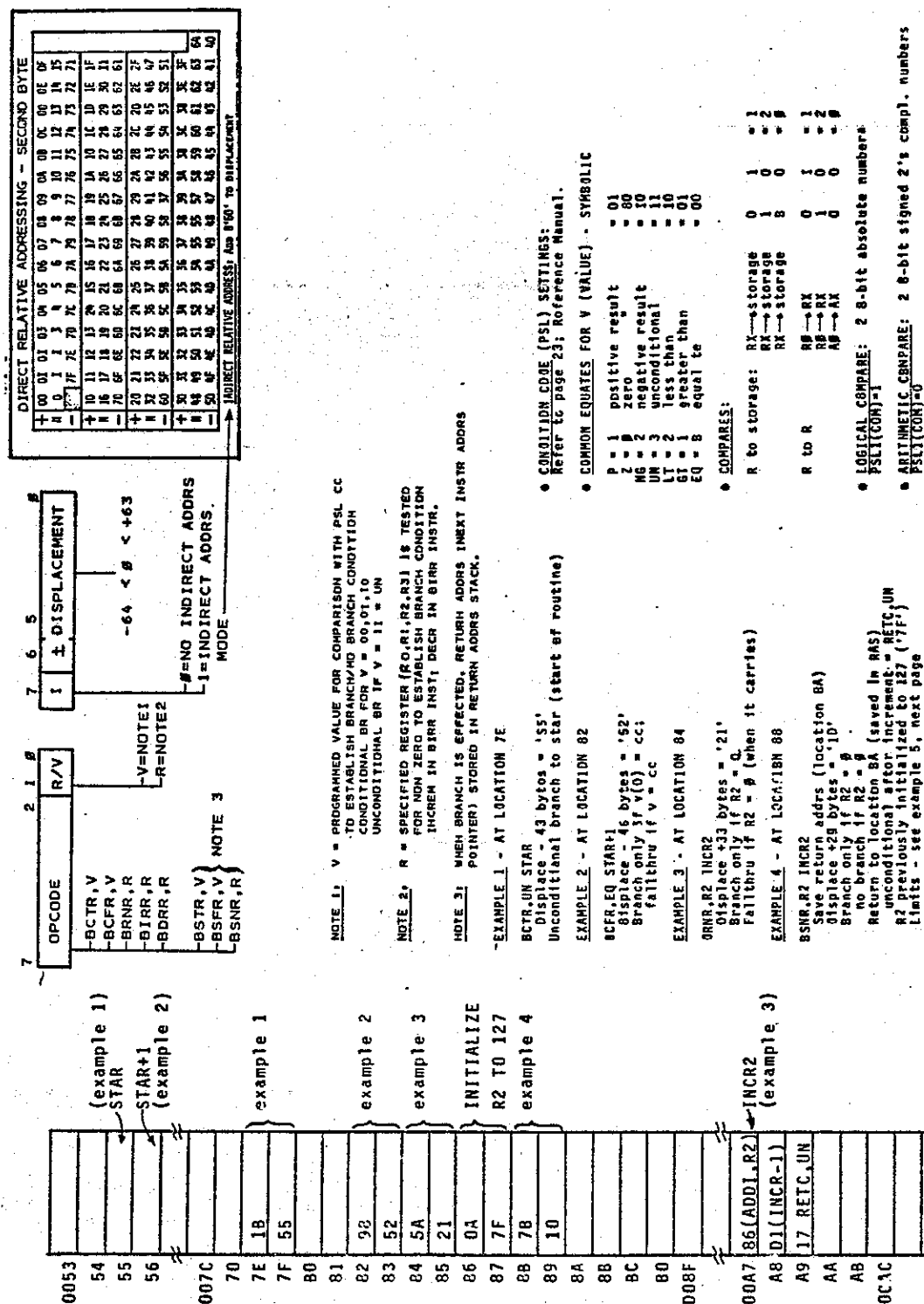


Figure 1



PROGRAM CONTROL BRANCH INSTRUCTIONS - I

NOTE: The examples illustrated on this and the next page describe all relative-addressed branch instructions with the exception of those which are ZERO BRANCH RELATIVE. ZBSR and ZBRR instructions are described separately in this module.



PROGRAM CONTROL BRANCH INSTRUCTIONS - II

C2	(BSTR,EQ)	
C3	(ROUT1)	
C4		
C5		
C6		
C7		
C8		
C9		
CA		
CB		
CC		
~~~~~		
000C		
0E	06	} INIT. R2
DF	64	
E0		} LOOP
E1		
E2	05	} INIT. R1
E3	8C	
E4	D9	} EXAMPLE 6
E5	7E	
E6		◀ P6
E7	FA	} EXAMPLE 5
E8	77	
E9		◀ P5
~~~~~		
00F9	(COMI,R0)	
FA	('00')	} EXAMPLE 7
FB	38	
FC	08	
FD		
FE	(ADDI,R0)	◀ P7
00FF	('01')	
0100	(BCTR,UN)	
01	(\$-9)	
02		
03		
04	(ADDI,R0)	◀ ROUT1
05	('01')	
0106	(RETC,UN)	

EXAMPLE 5 - AT LOCATION E5

LODI,R2 100 (initialize R2 = '64')(appendix J 167)
BDRR,R2 LOOP

- Loop seq is repeated 100 times; until R2 = 0
- Displace -11 bytes if condition satisfied
- Fallthru to addrs 00E9 when R2 = 0

EXAMPLE 6 - AT LOCATION E2

LODI,R1 156 (initialize R1 = '8C')
BIRR,R1 \$

- Single instruction is repeated 100 times (256-156) until R1 = 0
- \$ = -2 byte displacement
- Any counter (modulo 0-255) may be set up in this technique.

- * For delays
- * For events

EXAMPLE 7 - AT LOCATION FB

BSFR,EQ ROUT1

- Compare instr tests R0; sets PSL condition code.
- Branch is satisfied until R0 = 0
- Return address ('00FD') saved.
- R0 incremented in subroutine ROUT1 (addrs '0104')
- Unconditional return to addrs '00FD' at conclusion of routine.

QUESTION?

Is the instruction BSTR,EQ ROUT1 at location '00C2' valid in this program?

Yes No Why or Why Not? _____

DIRECTION: _____

EXERCISE 7INTRODUCTION:

The program "CUPDN1" (Count UP Down #1) provides a useful means by which you may demonstrate the effect of several relative-addressed program control branch instructions. In this exercise, you'll be varying the contents of several instructions in order to achieve predictably different results. In many applications, you'll select various sequences from routines such as "CUPDN1", and tailor them to fit your specific requirement.

DIRECTION:

Complete the requirements of Exercise 7.

PROCEDURE:

1. Code program "CUPDN1" (below). Compare your code with the solution on the next page.

ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
	B6	B1	B2		OPCODE	OPERANDS	
1	0000			CUPDN1	LODI, R2	0	INITIALIZE R2 AS LED DRIVER
2				SETUP	LODI, R1	XX	Then initialize R1 and R0
3				LOOP1A	LODI, R0	H'FF'	As delay constants. Now
4				LOOP1	BDRR, R0	LOOP1	execute the delay
5					BDRR, R1	LOOP1A	and display the
6					WRT, R2	PORT7	current "up-count" in LEDs
7					ADDI, R2	1	Increment the LED driver
8					COMI, R2	255	then see if "upcount" is
9					BCFR, EQ	SETUP	complete? No! Do another.
10				SETDN	LODI, R1	XX	Yes! initialize countdown
11				LOOP2A	LODI, R0	H'FF'	delay and execute delay
12				LOOP2	BDRR, R0	\$	for each count,
13					BDRR, R1	\$-4	then, display the "down
14					WRT, R2	PORT7	-count" on LEDs, and de-
15					SUBI, R2	1	crement the down-counter.
16					COMI, R2	0	Is the countdown complete?
17					BCFR, EQ	SETDN	No! Count down another time
18					BCFR, UN	SETUP	Yes! go count up again
19				* SET	value in loc '03'	for desired count-up delay.	
20				* SET	value in loc '13'	for desired count-down delay	
21				* SET	value in loc '0F'	for desired RANGE of up-count (upper limit)	
22				* SET	value in loc '1F'	for desired RANGE of down-count (lower limit)	

2. Referring to the comments provided in Program CUPDN1 on the previous page (line 19-20), select delay and range values as follows:

Location '03' = '60'
" '13' = '30'
3. Load your program into the INSTRUCTOR's memory in FAST PATCH mode.
4. Set a BREAKPOINT address at location '22' to permit one pass of program "CUPDN1".
5. Set the Program Counter to a start address at location '0' and depress **RUN**.
6. OBSERVATION:

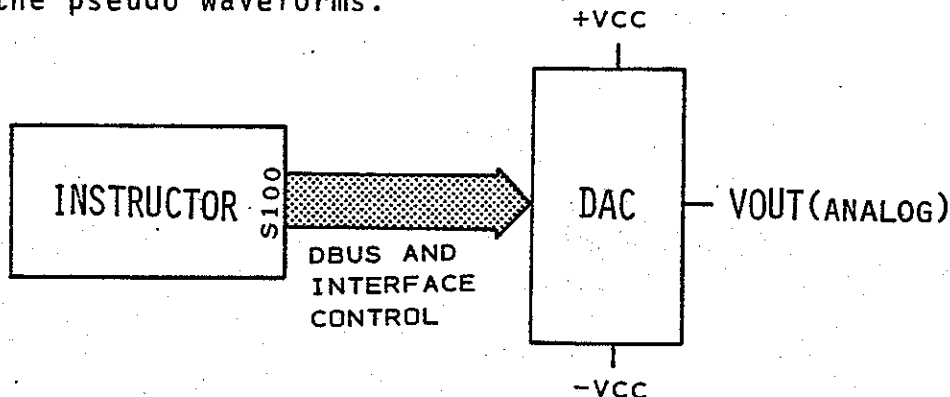
SUGGESTED CODE

"CUPDN1"

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION	
		B0	B1	B2		OPCODE	OPERANDS
1	0000	06	00		CUPDN1	LODI, R2	0
2	2	05	XX		SETUP	LODI, R1	XX
3	4	04	FF		LOOP1A	LODI, R0	H'FF'
4	6	F8	7E		LOOP1	BDRR, R0	LOOP1
5	8	F9	7A			BDRR, R1	LOOP1A
6	A	D6	07			WRITE, R2	PORT7
7	C	86	01			ADDI, R2	1
8	0E	E6	FF			COMI, R2	255
9	10	98	70			BCFR, EQ	SETUP
10	2	05	XX		SETDN	LODI, R1	XX
11	4	04	FF		LOOP2A	LODI, R0	H'FF'
12	6	F8	7E		LOOP2	BDRR, R0	\$
13	8	F9	7A			BDRR, R1	\$-4
14	A	D6	07			WRITE, R2	PORT7
15	C	A6	01			SUBI, R2	1
16	1E	E6	00			COMI, R2	0
17	20	98	70			BCFR, EQ	SETDN
18	22	1B	5E			BCFR, UN	SETUP

- The duration of count-up was _____ the duration of count-down.
- a. About equal to
 - b. One-half
 - c. Twice
 - d. Visibly shorter than
7. Compare your answer to that provided on tape 6A.
 8. APPLICATIONS:

You have the means to implement the digital (clocking) portion of a Digital to Analog converter (DAC). Consider that the 2650 microprocessor on the "INSTRUCTOR" can be interfaced via the S100 BUS to an external addressable DAC. By varying the programmed values for delay and range, the DAC may generate a variety of "staircase", triangular, and square waves of different amplitude. Subprocedures 9 through 12 demonstrate this capability, using the I/O LEDs to display the "digital slopes" of the pseudo waveforms.



9. SYNCHRONOUS UP-DOWN STAIRCASE (T₁=T₂)

1. Modify LDOPS 1, 1A, 2, and 2A

e.g., COUNT UP COUNT DOWN
T₁ T₂

..... to be equal.

Suggestion: Load Memory Locations as follows:

loc '03' = '06' loc '05' = 'FF' loc '13' = '06' loc '15' = 'FF'

2. Execute the program.

3. "Voltage Range" modification:

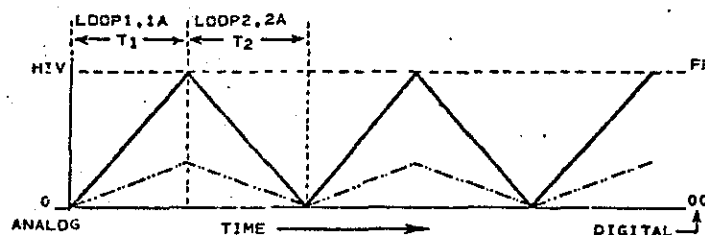
Modify loc '0F' from 'FF' (full range) to '1F' (1/8 of full range).

4. Execute the program. If it is running too fast,
- increase
- the delays modified in step A, by altering the following locations:

loc '03' from '06' to '50'
loc '13' from '06' to '50'

5. Execute the program.

6. Perform steps 1 through 5 in order to test the capability of the microprocessor to generate synchronous waveforms.

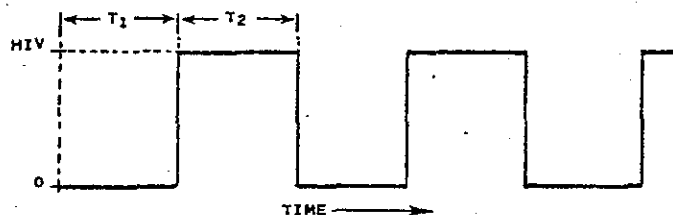
Note: Modification of locations '05' and '15' will also affect time duration. For very short waveform generation, modify all locations specified in step 1 of subprocedure 9.10. SYNCHRONOUS SQUARE WAVE (T₁ = T₂) (0 → H/V → 0)

1. Reprogram the
- longest
- time delay for count up and count down.

loc '03'	} set to = 'FF'
'05'	
'13'	
'15'	

2. Modify: loc '01' from '00' to FF
-
- '0F' from 'xx' to 0
-
- '20' from '98' to '99'

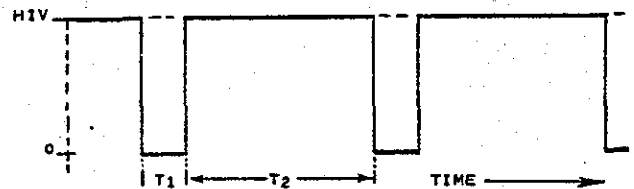
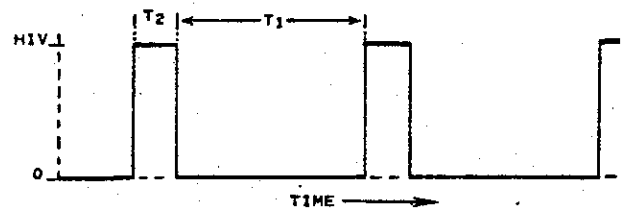
This will cause cycling between '00' and 'FF' with no intermediate steps (e.g., 01, 02 ... etc.)



3. Execute program then repeat step 1, loading the specified locations with H'50', then H'30'.
4. Listen to audio tape for a short commentary, then go on to subprocedure 11.

11. ASYNCHRONOUS SQUARE WAVE (CLOCK FUNCTION)

1. Set LOOP1 and 1A constants (locations '03' and '05') to 'FF' (τ_1).
2. Set LOOP2 and 2A constants (locations '13' and '15') to '40' (τ_2).
3. Execute the program, varying the values in steps 1 and 2 to vary pulse width and period.
4. To complement the waveform, (HIV going to zero pulse), exchange LOOP1 and 1A constants with LOOP2.



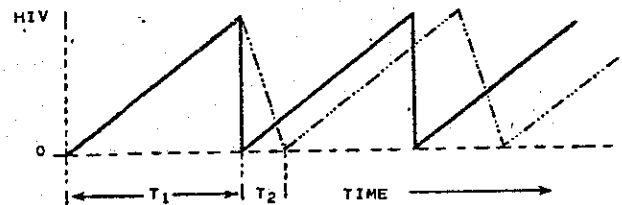
12. ASYNCHRONOUS STAIRCASE ("FLYBACK VARIATIONS")

The simplest method is to eliminate τ_2 , in order to reduce "flyback" time to a minimum. To accomplish this; restore the program to original value, and

1. Load a BCTR, UN CUPDN1 instruction into memory at location '12'.
2. Set the RANGE in location '0F'.
3. Set desired τ_1 (count-up time) in location '3' and execute the program.

To establish a KNOWN "rate of slope decay" (----- lines), restore the instruction at loc '12' to its original value and:

4. Set LOOP1 and 1A constants to fairly high values (procedure 11 step 1).
5. Set LOOP2 and 2A constants to very low values (procedure 11 step 2 - use a '10' or less).
6. Execute the program, then
7. Listen to tape 6A for a short commentary and further directions.



EXERCISE 8 - MODIFICATION OF "CUPDN1" TO SUBROUTINE CALL OPERATION

INTRODUCTION: Very often, you can save considerable memory by incorporating into subroutines those parts of the program which are most often used. In the program "CUPDN1", instructions to initialize and execute the delays were repeated. In an updated version, ("CUPDN3"), we'll see these functions accessed through execution of appropriate subroutine call instructions.

Also included in "CUPDN3" is the ability to control the number of times the program sequence for counting up and down is repeated. The function of EVENT counting is implemented in order to execute a routine a precise number of times, then exit to execute the rest of the program.

DIRECTION:

1. Code the routine "CUPDN3" (below). Compare your code to that provided in the suggested solution on the next page.
2. In line 12 and 13, programmed delay constants 10 and 80 will provide a count-up-down cycle time of approximately 11 seconds.

ROUTINE "CUPDN3"

	ADDRES	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B7	B1	B2		OPCODE	OPERANDS	
1					* SET I/O SWITCHES TO COUNT OF UP/DN			* CYCLES TO RUN *
2	0030				CUPDN3	REDE, R3	PORT7	Fetch run cycles, and
3						LODI, R2	0	initialize an LED driver
4					AGAIN	BSTR, UN	INITCT	then go delay. Increment
5						BIRR, R2	\$-2	the count until all LED's
6						BSTR, UN	INITCT	turned on. Now decrement
7						BDRR, R2	\$-2	count until all LED's off.
8						BDRR, R3	AGAIN	When up/dn cycles com-
9						WRTE	RO	plete, exit to the
10								Monitor
11								
12	0040		10		INITCT	LODI, R1	XX	Setup delay constants.
13			80			LODI, R0	FF	then time out the delay
14						BDRR, R0	\$	and show the
15						BDRR, R1	\$-4	current count (binary)
16						WRTE, R2	PORT7	ON LED's. Then exit
17						RETC, UN		to calling routine.

DIRECTION:

3. Listen to the audio tape for a brief commentary on routine "CUPDN3". Then continue with step 4 and following.

NOTES:

SUGGESTED CDDE: "CUPDN3"

2	0030	57	07		CUPDN3	RE
3		06	00			LOI
4		3B	0A		AGAIN	BSI
5		DA	7C			BIT
6		3B	06			BSI
7		FA	7C			BD
8		FB	7C			BD
9		B0				WR
10						
11						
12	0040	05	10		INITCT	LO
13		04	80			LOI
14		F8	7E			BD
15		F9	7A			BD
16		D6	07			WR
17		17				RE

4. Set the I/O input switches to '03' (switches 7 through 2 OFF; switches 1 and 0 ON).

5. Set the I/O SELECTION SWITCH to EXTENDED.

6. Set a routine START ADDRESS (at location '30') and depress **STEP**.

Observation: R3 = ' _ ' (should be '03').

7. Depress **RUN** to execute the routine.

Question: Before routine "CUPDN3" exits to the monitor, how many times is the count-up/down cycle repeated?

_____ (1)(2)(3) time(s).

8. Compare your answer with that provided on tape 6A, then go on to step 9.

9. Write and code a relative-addressed unconditional branch instruction at location '00' to transfer program control to routine "CUPDN3".

ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
	B2	B1	B2		OPCODE	OPERANDS	
1							

STEP 9 CODE

	ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
		B0	B1	B2		OPCODE	OPERANDS	
1	0000	1B	2E			BCTR,UN	CUPDN3	Branch to routine CUPDN3
2								

NOTE: In a complete program, it is likely that you would have defined a symbolic label (in the label column). For this exercise, this is not necessary.

11. On your own, vary the delay constants in lines 11 and 12, then execute the program, while observing the time it takes to execute the program.

NOTE: If you have access to a stop watch capable of timing in 0.1 sec increments, you can make some rather precise observations.

NOTE: When you are measuring short time duration, it is useful to input a high up/down cycle count into the parallel I/O toggle switches (e.g., 100₁₀ or 200₁₀ ('64' or 'C8')). Then divide the total time measured by the number of cycles entered to arrive at a time/cycle value.

Notes and observations: _____

////////////////////////////////////

DIRECTION:

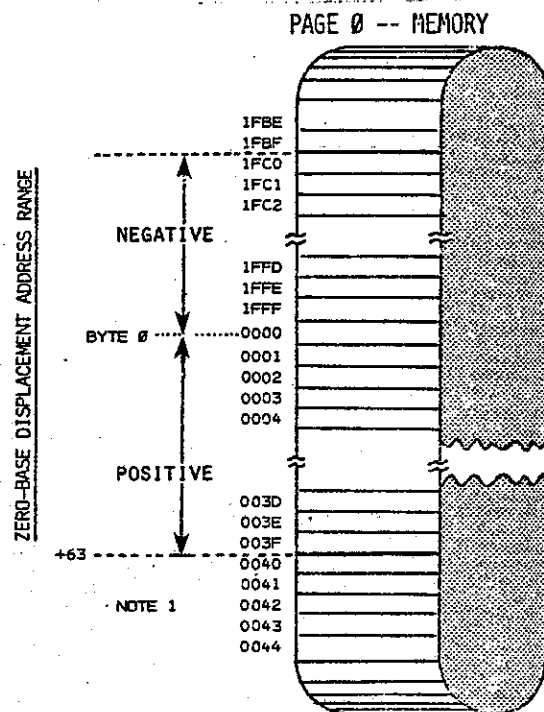
Go on to the next page for a study of PAGE ZERO-REFERENCED BRANCH INSTRUCTIONS.

PAGE ZERO-REFERENCED BRANCH INSTRUCTIONSINTRODUCTION:

Two branch instructions (ZBRR and ZBSR) permit an effective address to be established relative to Page 0 Byte 0 in the microprocessor's memory, regardless of the location of the branch instruction.

As seen in the diagram (right), the address structure of page 0 is "wrap around". Thus the last 64 locations in page 0 (addresses 1FC0-1FFF) are accessed by programming a negative displacement in the zero-based branch instruction's Byte 1. Likewise, the 1st 64 locations in page 0 are accessible by programming a positive displacement in Byte 1.

When used in the indirect address mode, any location in memory may be accessed through execution of the zero-based referenced instruction. The effective address of the desired memory location is contained in the 2 byte field within page 0 byte 0 reference limits (+63 to -64 locations).

NOTE 1:

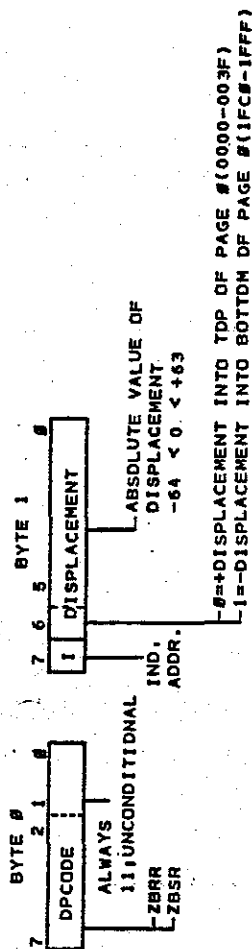
Although technically outside the displacement range of zero-based instructions, address 40 (+64 bytes from location 0) may be specified as the 2nd (low-order) byte of indirect address described in the previous paragraph.

////////////////////////////////////

DIRECTION:

Study the ZBRR/ZBSR Parameter diagrams on the next page, then listen to tape 6A.

PAGE ZERO-REFERENCED BRANCH INSTRUCTION PARAMETERS



- Execute ZBSR or ZBRR from any addressable location (0 → 32767₁₀) in memory.
- Unconditional branch to page 0 within calculated range of addresses: 0-63 (9988-003F) = ZBRR:ZBSR operating zone 8128-8191 (1FC8-1FFF)
- Address bits 14 and 13 cleared unconditionally.
- Instruction address reg contents replaced with calculated effective address (example 1).

If indirect addressing is specified, 2 locations (displacement address and displacement address+1) must be reserved.

PRACTICAL NOTE: Because of 128 byte limitation of ZBRR and ZBSR accesses, indirect addressing is employed in all but the shortest of programs (example 2).

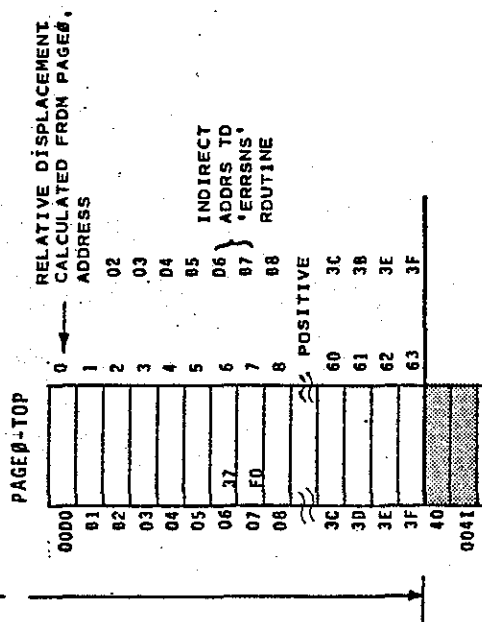
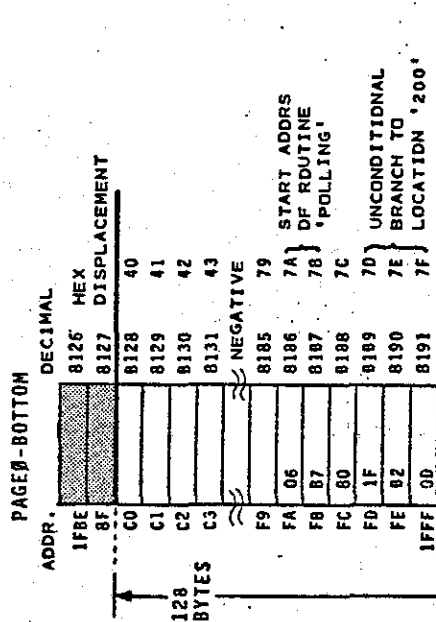
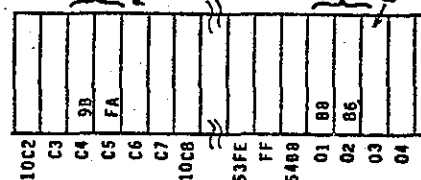
EXAMPLE 1 - AT LOCATION '10C4' (PAGE 0)

- ZBRR POLLING
- Unconditional branch to location 'FFA' to access routine 'polling'.

Processor immediately executes the routine 'polling' at location 1FF8, executes unconditional branch to continue from location '200'.

EXAMPLE 2

- ZBRR * ERRSNS - AT LOCATION '5481' (page 3)
- Save return address (at pointer p2) in stack
- Unconditional branch to location '8006' (page 0) to access indirect address.
- Processor immediately loads the instruction address with absolute address '37F8', the first location of routine 'ERROR SENSE' execution.
- At conclusion of that routine's execution, processing of a RETC instruction permits return to the calling program at address '5403'.



EXERCISE 9: PRACTICAL APPLICATION OF ZERO-REFERENCED BRANCH INSTRUCTIONS - ZBRR; ZBSRINTRODUCTION:

The program "CRAPGAME" makes generous use of ZBSR and ZBRR instructions in order to achieve maximum program flexibility and organization within minimum available memory space. In this exercise, you'll be examining the sequence of program memory access and execution as influenced by the Zero branch relative instructions. Both direct and indirect address variations are demonstrated.

PROCEDURE:

1. From Module 1, pages 41 to 52, access the listing of the program "CRAPGAME".
2. Access Tape 2, containing program "CRAPGAME" and load it into the "INSTRUCTOR's" memory.

NOTE: If you don't recall the method for cassette entry, refer to Module 1, page 59.

3. Play the "CRAPGAME" for a few minutes to verify correct operation, then proceed to the next step.
4. Listen to tape 6B for a brief explanation of the Indirect Address Table in sheet 1 of the CRAPGAME listing, Module 1, page 41.
5. Set a START ADDRESS at location '00'.
6. Set a BREAKPOINT ADDRESS at location '38'. Depress **RUN**.

NOTE: Make the necessary entries for CHIPS and BET as requested, then stop the roll by depressing the **SENS** key.

7. Now, depress **STEP** once. Display = _ _ _ _ _
8. On the 2nd sheet of the "CRAPGAME" listing, take a few minutes to interpret the ZBSR instruction at line 5. In the space below, indicate the location of the INDIRECT zero-referenced address. Then, from sheet 1, indicate the value of the effective address.

"Z" indirect address at location ' _ _ _ _ '. Effective address is ' _ _ _ _ '.

9. Compare your findings to those provided on the next page.

Correct answer to questions in step 8:

"Z" address at location '0004'.
Effective " " " '013A'.

10. Depress **STEP** once.
Does the "next address" equal the effective address indicated above? ____ (yes)(no). (There's a problem if it doesn't).
11. Now, referring to the routine "DISMES" in the "CRAPGAME", (Module 1, page 47) set a BREAKPOINT ADDRESS at location '157'. Depress **RUN**.
Display: -0157 17 this is the BREAKPOINT address.
The instruction at this location has been executed.
12. Inspect the Program Counter for "next instruction address".
Observation: Next instruction at location '____'.
Question: Upon execution of the instruction at the above location, what is the effective address of the following instruction?
(Use the program listing only) '____'.
13. Depress **STEP** once. The effective address of the "next instruction" is _____.
Question: What was the location of the INDIRECT address to this effective address?
Answer; a 2 byte field starting at loc: '____'.
14. The Table below contains a series of BREAKPOINT addresses, all referenced to the program "CRAPGAME". In succession, load each specified BREAKPOINT address into the "INSTRUCTOR". Before depressing **RUN** predict the INDIRECT and EFFECTIVE ADDRESS Locations, using the PREDICTION columns of the Table for your entries. Upon depressing **RUN**, inspect the Program Counter for the next address and enter it in the ACTUAL address column. After completing the Table, compare your entries with those provided on the next page.

	BREAKPOINT ADDRESS	** PREDICTION **		ACTUAL ADDRESS
		INDIRECT ADDR	EFFECTIVE ADDR	
(1)	7F			
(2)	81			
(3)	83			
(4)	85			
(5)	87			
(6)	89			
(7)	8B			

	BREAKPOINT ADDRESS	** PREDICTION **		ACTUAL ADDRESS
		INDIRECT ADDR	EFFECTIVE ADDR	
(1)	7F	0004	013A	013A
(2)	81	0002	01B1	01B1
(3)	83	0006	0183	0183
(4)	85	—	0087	0087 *
(5)	87	—	0089	0089 *
(6)	89	0004	013A	013A
(7)	8B	0008	015C	015C

*NEXT ADDR. IN SEQUENCE.
NO Z-REFERENCED INDIRECT ADDR.

15. Refer to sheet 6 of the "CRAPGAME's" main program (Module 1, page 46). Code an appropriate page zero referenced instruction to replace the BCTA instruction at location '11A' (line 16).

NOTE: The BCTA instruction in this case:

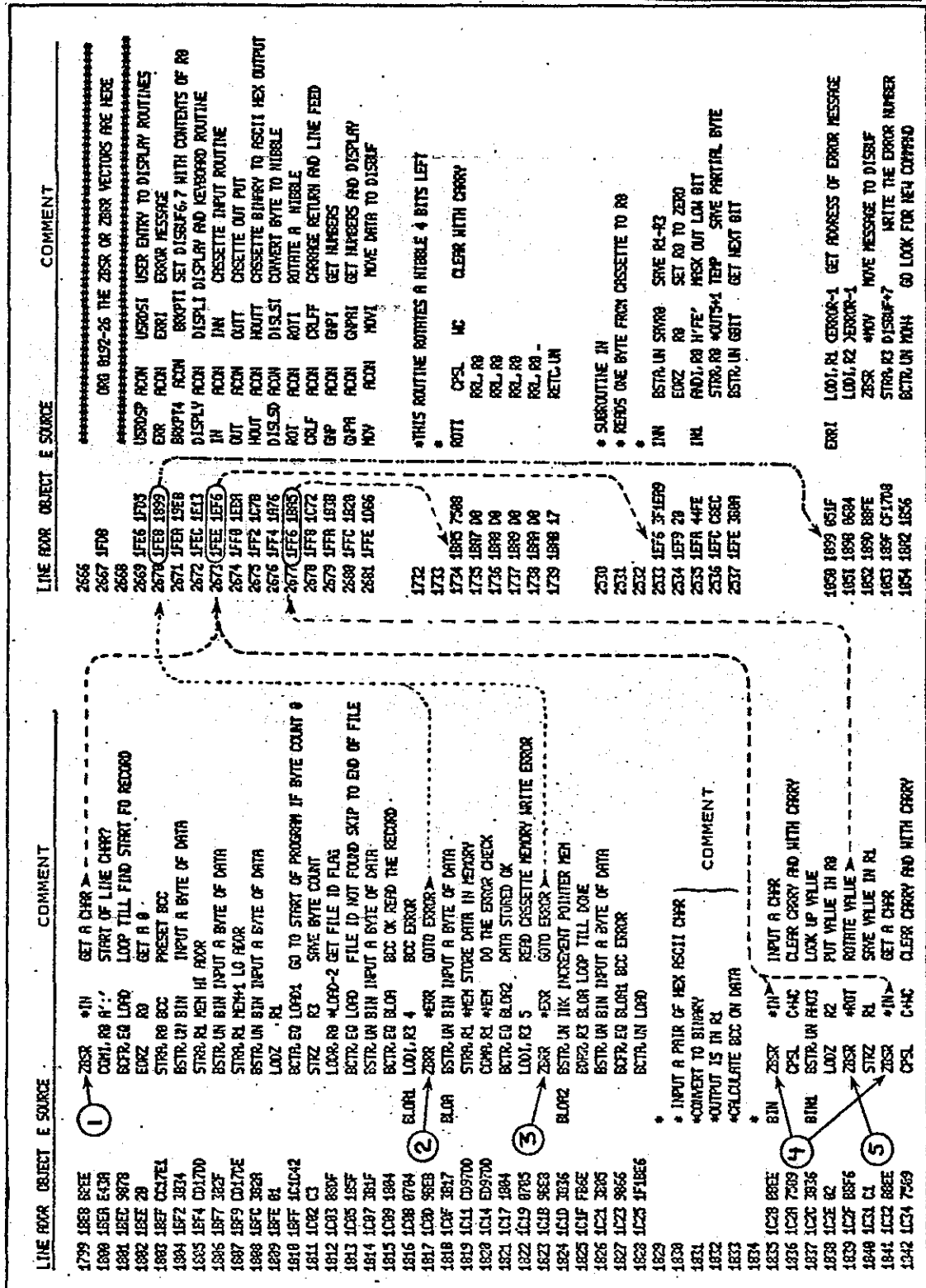
- (1) Transfers program control unconditionally to "NEWGAME" at loc. '10'.
 - (2) Is NOT a subroutine call instruction.
 - (3) Is written in DIRECT, not indirect, absolute address format.
16. Compare your code with the suggested solution on the next page. Do not load the new instruction into memory until you perform step 20.
17. SET a BREAKPOINT at location '117' and START address at location '0'. Depress **RUN**.
18. Make the manual entry to "buy into the game".
19. Upon reaching the BREAKPOINT address, inspect the Program Counter for "next address", then depress **STEP**. Display should indicate a transfer of program control to location '10', the address of "NEWGAME".
20. Load the ZBRR instruction at location '11A'.
21. Repeat steps 17 through 19. Display results should be the same as that observed during the 1st pass of step 19.
22. Listen to tape 6B for a brief summary of the page 0 - referenced branch instructions. Use the next page as a reference. Then, as directed on tape, go on to the following page.

NOTES: _____

CODING SOLUTION - STEP 15 - INDIRECT ADDRESSED

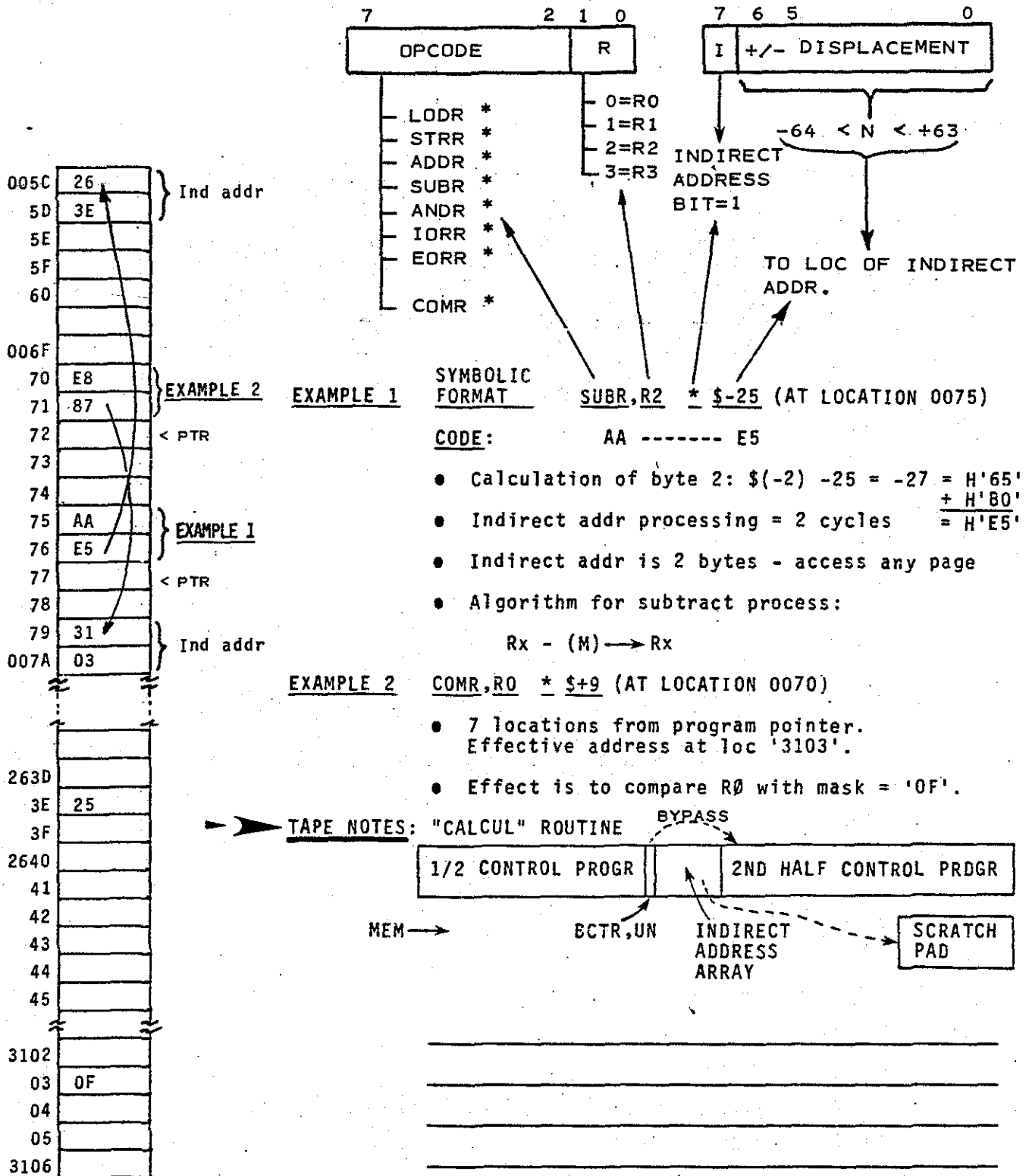
ADDRES	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
	E0	E1	E2		OPCODE	OPERANDS	
11A	9B	10			ZBRR	NEWGAME	Exit to place a bet

ASSEMBLER EXAMPLES

INDIRECT RELATIVE ADDRESSING
ZBSR, ZBRR BRANCHES

DATA HANDLING INSTRUCTION PARAMETERS ~ RELATIVE INDIRECT ADDRESS

(Exclusive of ZBRR,ZBSR instructions)



EXERCISE 10 PRACTICAL USE OF INDIRECT RELATIVE ADDRESSED DATA
TRANSFER INSTRUCTIONSINTRODUCTION:

You heard previously on audio tape a recommendation to allocate a block of control storage within a routine for indirect address purposes. There is another case in which use of indirect addressing is effectively implemented to conserve memory. Consider that the last 2 bytes of a 3 byte ABSOLUTE ADDRESSED instruction defines a specific location in memory. Any relative addressed instruction whose displacement range includes a 2-byte field of this nature, may access that field as an indirect address. This operation will be demonstrated in Exercise 10.

Let's consider a few more facts relative to program preparation. Initially, you specify the overall sequence of the program - what it will accomplish and when -- using objective and internal specifications of your application's requirements. Then, it is a usual practice to "write" the software - the instruction sequence - for each part of the program. As you write each routine, it is inevitable that you will implement absolute addressed instructions to select data "located", presumably, in some distant memory locations. If for no other reason than you know the address, and can complete the code. Each routine is then debugged and saved. At some point in time, you have to LINK the different routines together as one cohesive operational program. It is when you perform this LINKING function that you may detect multiple use of absolute addressed instructions to call one specific location. If this is the case, AND IF your program might possibly exceed specified memory size (e.g., 1 or 2K of ROM), then see if you can convert some of the absolute addressed instructions to relative indirect. For each conversion, you save a byte. In a 1K program, this savings might approach 50 bytes.

Exercise 10 provides a means by which you can demonstrate a capability to recognize - and implement - conversions from direct absolute address to indirect relative addressed data transfer instructions. The procedure starts on the next page.

////////////////////////////////////

DIRECTION:

Go right on to the next page.

PROCEDURE:

1. From Module 1 documentation, access sheets 5 and 6 of program "CRAPGAME's" listing (page 45 and 46; Module 1).

2. Load the program "CRAPGAME" into the "INSTRUCTOR's" memory.

NOTE: Disregard step 2 if "CRAPGAME" is still resident in user storage.

3. Examine sheet 5 of the program listing for "CRAPGAME"; lines 6 and 8. The indirect addresses for 'CHIPS' and 'BET' are contained in memory locations 'D4'-'D5' and 'DA'-'DB' respectively.
4. Referencing sheet 5 ONLY, select the MOST CORRECT answer to complete the following statement.

"INDIRECT relative-addressed data transfer instructions which access 'CHIPS' and 'BET' are located at addresses:"

- a. 'E1', 'EE', 'F6'
- b. 'E1', 'EA', 'EC'
- c. 'E4', 'EA', 'EC'
- d. 'D2', 'E1', 'EA'

5. Listen to the audio tape for the correct answer, then go on to step 6.
6. Referencing sheet 5, is there another indirect addressed relative data transfer instruction?

ANSWER: _____ (yes)(no) If Yes, write down the SYMBOLIC instruction and indicate the EFFECTIVE address of the data byte in memory. Also indicate the memory location of the INDIRECT address.

SYMBOLIC INSTRUCTION: _____

EFFECTIVE ADDRESS _____

INDIRECT ADDRESS '____'-'____'

7. Listen to the tape for the correct answers to this question, then go on to step 8.
8. Reference sheets 5 and 6. With the routines "WON", "WINDUP", "GAMEOVR", "CALC LOC", and "CORRSUB" located in the addresses as shown, which of the following absolute addressed instructions on sheet 6 could be re-written as relative indirect-addressed instructions? Check the box at the right of each location as applicable:

LOCATION	CHECK (✓)
00F9	
00FC	
0117	
012C	
012F	

RELATIVE NEGATIVE
DISPLACEMENT FROM
THESE LOCATIONS DO
BRACKET EXISTING
ABSOLUTE ADDRESSES
ON SHEET 5

RELATIVE NEGATIVE
DISPLACEMENT FROM
THESE LOCATIONS
DOES NOT BRACKET
EXISTING ABSOLUTE
ADDRESSES ON PAGE 5.

LOCATION	CHECK (✓)
00F9	✓
00FC	✓
0117	
012C	
012F	

9. Using the facilities of the "INSTRUCTOR", code and load relative indirect addressed instructions into the locations identified with checkmarks (✓) in step 8. Replace the 3rd byte (byte 2) of each original instruction with a NOP ('CO'). The NOP (NO OPERATION) instruction permits program execution to sequence through unused memory locations. Suggested code is provided on next page.

NOTE: In calculating the relative displacement for the second byte of the new instructions, remember that the Program Counter points to the "next instruction address". For each instruction, the "next instruction address" is at the location occupied by the NOP.

CAUTION: DON'T FORGET TO ADD H'80' TO THE CALCULATED DISPLACEMENT.

10. To check out your modifications, set a BREAKPOINT address at location 'F9'. Set a START ADDRESS at location '0' and depress RUN. Play CRAPGAME until you LOSE.

NOTE: The routine "CALCLOS" at location 'F9' subtracts the bet from the player's current chips.

The observed display should be -00F9 08 to indicate that the program has executed through the desired BREAKPOINT address.

11. Inspect R0 and memory location 'OF' ("CHIPS"). The contents of each should be equal. Their value is .
12. Inspect memory location 'OE' ("BET"). Note the value: BET= .
13. Depress STEP twice. Then inspect R0. R0= .

NOTE: Since no decimal adjust takes place on the contents of R0 until the routine "WINDUP" is executed, the value in R0 may be a hex number. This hex number equals CHIPS minus BET.

14. Set a BREAKPOINT address at location 'E1' (see sheet 5 - "WINDUP" routine).
15. Depress **RUN**. Then, inspect R0 and memory location 'OF'. They should be equal to each other and the value is the DECIMAL DIFFERENCE between the values derived in steps 11 and 12.
16. If you had unpredictable results, compare your code to that provided at the right, then repeat steps 9 through 15.

NOTE: Most problems arise when the programmer:

- (a) Fails to code the INDIRECT ADDRESS IDENTIFIER (*='H'80') in Byte 1.
- (b) Fails to convert the opcode of the absolute instruction into a relative address opcode. (Byte 0).
- (c) Does not allow for the "next address" pointer in his calculation of relative displacement in Byte 1.

CORRECT CODE - STEP 9

7	00F9	08	DA		CALCLOS	LODA,R0	#CHIPS
8	B	CO				NOP	
9	C	A8	DC			SUBR,R0	#BET
10	E	CO				NOP	
11	F	E4	FB			COMI,R0	H'FB'
12							

OUTSIDE PROJECT: On your own, inspect memory locations '00B1' through '0131'. Determine the feasibility of arranging the indirect addresses of BET,CHIPS, AND XXBET+7 near the mid-point of this block. If feasible, rewrite the instructions in this block of memory to accomplish this, eliminating absolute addressed instructions wherever possible and substituting relative indirect-addressed data transfer and manipulation instructions.

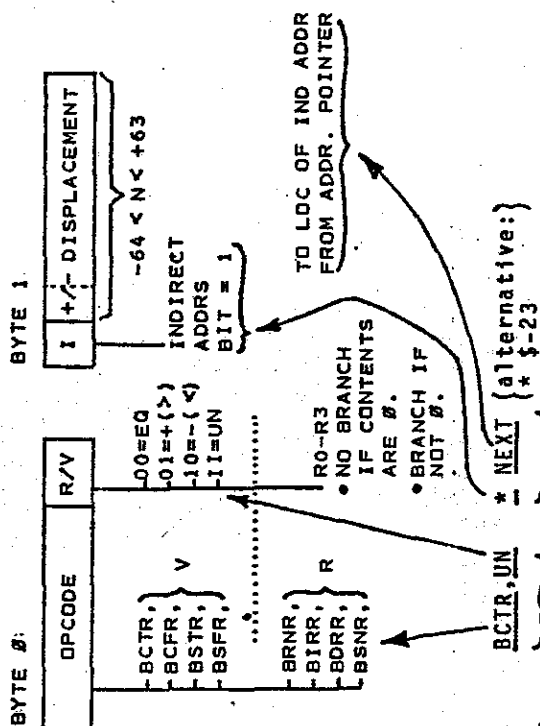
////////////////////////////////////

DIRECTION:

Go right on to the next page and study it for a few minutes. Then, listen to the tape for a short commentary.

NOTE: Leave the "CRAPGAME" resident in user storage. You'll need it in the next exercise.

PROGRAM BRANCH INSTRUCTIONS (EXCLUDING ZBRR,ZBSR BRANCHES) INDIRECT ADDRESS



EXAMPLE 1

SYMBOLIC
FORMAT:

CODE:

- Calculation of Byte 2. PTR at location 0126
Ind. addr. at location 010D
Displacement = H'67' = -25 locations
Ind. addr. (*) = H'80'
= H'E7' = *-25 locations
- Indirect Rel Addr. Branch = 5 cycles.
- Effective Addr. to "Next Routine" is '137D'
- Unconditional Branch. (UN)

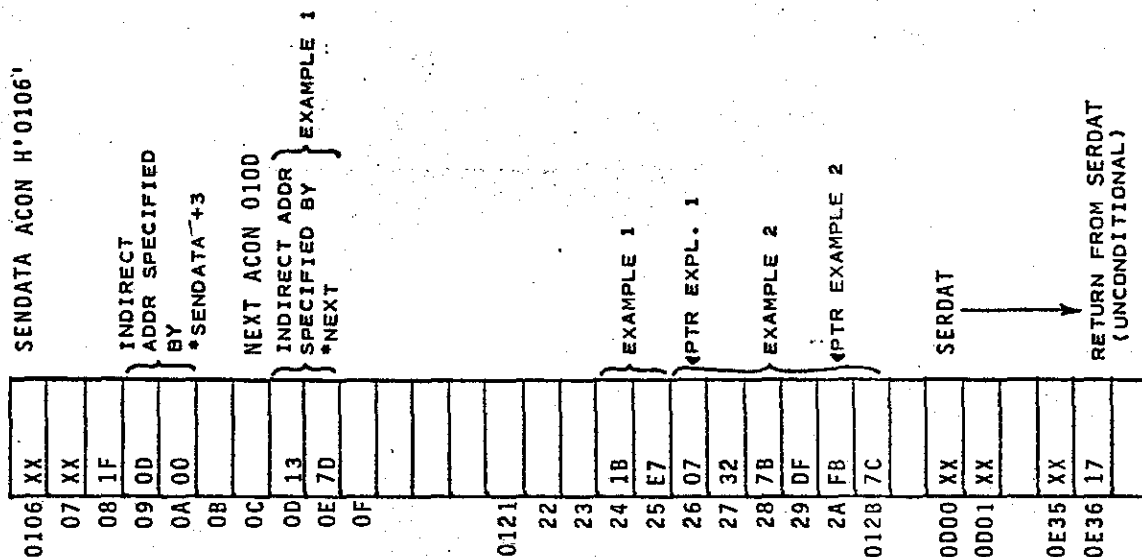
EXAMPLE 2

L001,R3 50

BSNR, R3 *SENDATA+3

BDRR.R3 1-2

- Note that effective address to "SERDAT" is part of a BCTA instruction.
- Technique of relative displacement from location D12A (◀) to loc. 0109 saves a byte.



EXERCISE 11 - PRACTICAL USE OF PROGRAM CONTROL BRANCHES - INDIRECT
RELATIVE ADDRESSINTRODUCTION:

In this exercise, you'll identify the use of indirect relative addressed program control instructions with the program "CRAPGAME". In addition you'll convert a direct absolute addressed branch instruction into one which is indirect relative addressed yet which performs the same function.

DIRECTION:

Complete the steps in the following procedure.

1. Access the program documentation for "CRAPGAME"; sheets 5 and 6 of the Main Program (Module 1, pages 45 and 46).
2. On sheet 5, locate the unconditional branch to NEWGAME in line 15.
3. Now, locate the INDIRECT ADDRESS field identified in Byte 1 of the BCTR instruction in line 15.

OBSERVATION: The Indirect Address of "NEWGAME" is a 2-byte field in memory locations: '___' and '___'. The BCTR instruction in line 15 will use this field to vector program sequence control to memory locations '___' and following.

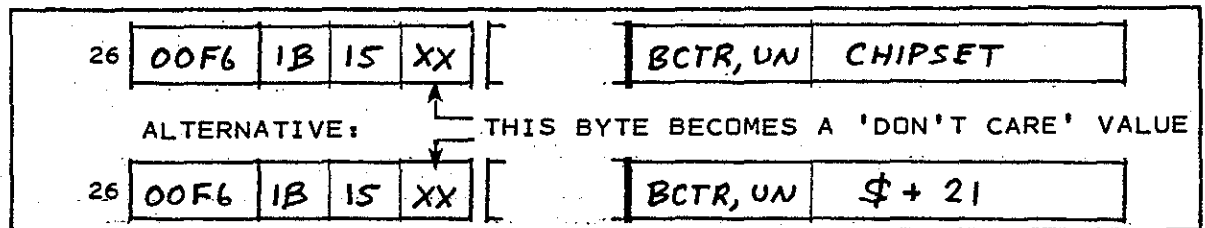
4. Listen to tape 6B for verification of your answers, then go on to step 5.
5. Referencing sheets 5 and 6 of the Main Program, convert the unconditional absolute addressed Branch Instruction (at line 24 sheet 5) into a relative addressed branch instruction.

NOTE: Decide, BEFORE you code the instruction, whether the relative instruction requires a direct or indirect address.

ADDRS	DATA			LABEL	SYMBOLIC INSTRUCTION		COMMENT
	B0	B1	B2		OPCODE	OPERANDS	

6. Compare your code to the solution on the next page, then go on to step 7.

CODE CONVERSION SOLUTION - STEP. 5



7. To Verify the operation of your modified instruction:

- (a) Set a START ADDRESS at location 'E9'.
- (b) Depress STEP sequentially, checking that the branch to "CHIPSET" takes place.

8. Access sheets 2 through 4 of the Main Program in "CRAPGAME" DOCUMENTATION (Module 1, pages 42-44) to answer the following questions: (Consider that the routines can NOT be relocated)

- (a) Can the BCTA instruction at address '5A' be converted to a BCTR, UN* instruction? ____ (yes)(no).
- (b) Can the BCTA instruction at address '5F' be converted to a BCTR, UN* instruction? ____ (yes)(no).
- (c) Can the STRA instruction at address '47' be converted to a STRR, R0* instruction? ____ (yes)(no).
- (d) Can the BCTA instruction at address '8B' be converted to a BCTR, EQ instruction? ____ (yes)(no).

If "yes", would this affect your answer in question (b) above? ____ (yes)(no). Explain: _____

- (e) Can the STRA instruction at address 9E be converted to a STRR, R2* instruction? ____ (yes)(no).
- (f) What is the correct code which should be entered in the unused byte resulting from a conversion from a 3 byte to a 2 byte instruction?

- (1) '00'
- (2) 'C0'
- (3) Leave original 3rd byte as is.
- (4) Don't care ('XX')

Answer: _____

9. The answers to the questions in step 8 are provided below. Without looking at the answers, code your modifications and use the facilities of the "INSTRUCTOR" (START ADDRESS, BREAK-POINT, and STEP, etc.) to prove your answers. Then compare your answers to those provided below.

NOTE: Code 'C0' in excess bytes remaining after program changes.

ANSWERS TO QUESTIONS - STEP 8

- (A) NO - NO INDIR. ADDR. IN DISPLACEMENT RANGE.
(B) YES - INDIR. ADDR. AT LOCATIONS '90' AND '91'
(C) YES - " " " " '67' AND '68'
(D) YES - "BOMBOUT" AT LOCATIONS 'B1', WITHIN
DIRECT REL. ADDR. DISPLACEMENT RANGE.
YES - ANSWER TO QUESTION (B) WOULD BE NO! THE
CONVERSION OF BCTA AT LOCATION 8F TO
A RELATIVE ADDRESSED INSTRUCTION WOULD
DESTROY THE INDIRECT ADDRESS REQUIRED
BY A 'YES' ANSWER IN QUESTION (B).
(E) YES - INDIRECT ADDR. AT LOCATIONS '67' AND '68'
(F) (2): 'C0' (A NOP INSTRUCTION)

10. At your option, restore (or leave as is) the instructions you modified in step 9.

NOTE: If you leave the instructions modified,
update your program documentation to
reflect the changes.

11. From a MEMORY CONSERVATION point of view, would there be any advantage in coding the BCTR instruction at location 'E7' (CRAPGAME Main Program) as a ZBRR instruction?

ANSWER: _____ (yes)(no) EXPLAIN: _____

12. Listen to the audio tape for the correct answer, then go on to the next page.

ABSOLUTE ADDRESS MODESINTRODUCTION:

The next section of this module describes the use of ABSOLUTE addressed instructions. Two separate instruction formats are assigned for use in preparation of absolute addressed instructions; these are Format A for data handling and manipulation, and Format B for program control branches. Within each format, both direct and indirect addressing is permitted.

When direct addressing is considered, only Format B branch instructions permit program sequence execution between pages (8K blocks) of memory. 15 bits of address are specified in the operand bytes of the Format B instruction. In Format A, only 13 bits of address may be specified in the operand bytes. Thus, data must be accessed from the same page of memory in which the calling absolute addressed non-branch instruction is located. If desired data is located in another memory page. INDIRECT addressing must be implemented.

The important feature of INDEXING can be selected for use in all non-branch (Format A) absolute addressed instructions. Two operand bits (INDEX CONTROL) specify 1 of 4 index arguments. The effective address of selected data is determined by adding the contents of the register specified in the opcode byte to the absolute 13 bit address specified in the operand bytes. In fact, the specified register is the index. Specified register contents may be incremented by 1, decremented by 1, or unchanged, depending on which index argument is specified. Where indexing is specified, Register 0 becomes an implicit source or destination of the 2nd byte of data in non-branch instructions specifying 2 bytes (e.g., LODA, ADDA, COMA). Non branch instructions which specify 1 byte (e.g., DAR) are not written in absolute address format. If the index argument specifies NO INDEX, the register specified in the OPCODE byte is the source or destination of the 2nd byte. Indexing may be implemented with great flexibility in application involving multiple and sequential data manipulation and byte transfer. Indexing particularly lends itself to table driven programs, examples of which are demonstrated later in this module.

REFERENCE READING:

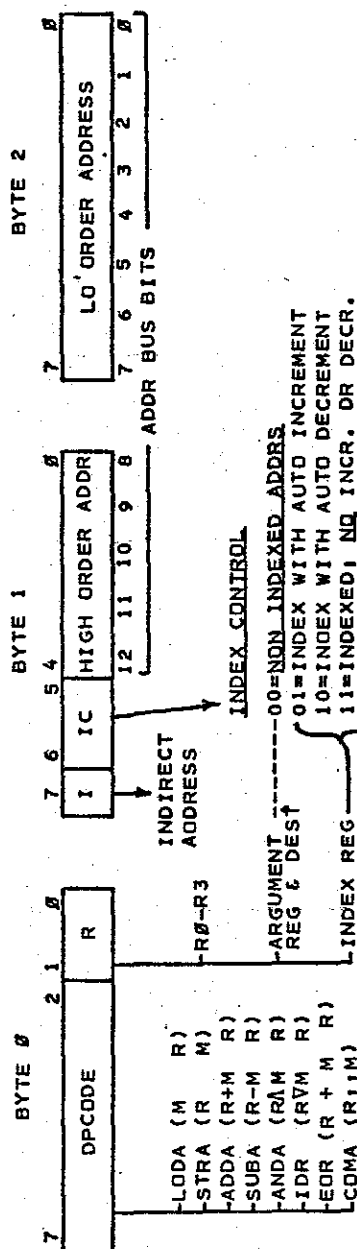
A fine overview of ABSOLUTE ADDRESSING principles, including a thorough discussion of INDEX CONTROL, is provided in the 2650 Reference Manual within the section describing the INSTRUCTION SET.

DIRECTION:

The next 3 pages provide a detailed analysis of absolute-addressed Data Handling instruction variations. Take some time to study all three pages, then listen to tape 6B, open to the next page.

FORMAT A

NON-BRANCH INSTRUCTIONS - ABSOLUTE DIRECT ADDRESS



- When INDEX CONTROL (byte 1 bits 6 or 5) turned on, R0 is implicit first argument
RX (byte 0 bits 1 & 0) is index reg.
- High order address page bits (14 and 13) are unchanged.
Access of memory byte within same pages as fetch instruction.
- Single byte is fetched from memory.

● ABSOLUTE ADDRESS MODES

Absolute Address (non indexed)(direct)

*IC=Ø; I=Ø

*R is first argument and destination

*MA = high and low order absolute address

EXAMPLE 1 - At Location 0136

ADDA.R3 DATA where DATA is at addr 006E

- Result: R3 + 1F (at loc '006E') R3

EXAMPLE 2 - At Location 013A

SIRA-B2 DATA + 1

- Result: R2 = .55 R2

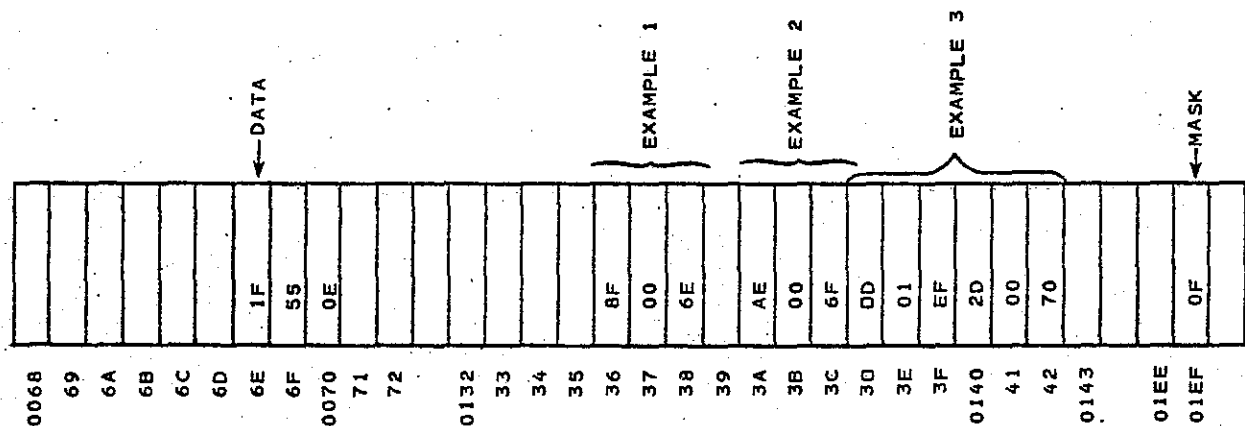
EXAMPLE 3 - At location 0130

28 JUL 1966

LODA, R1 MASK

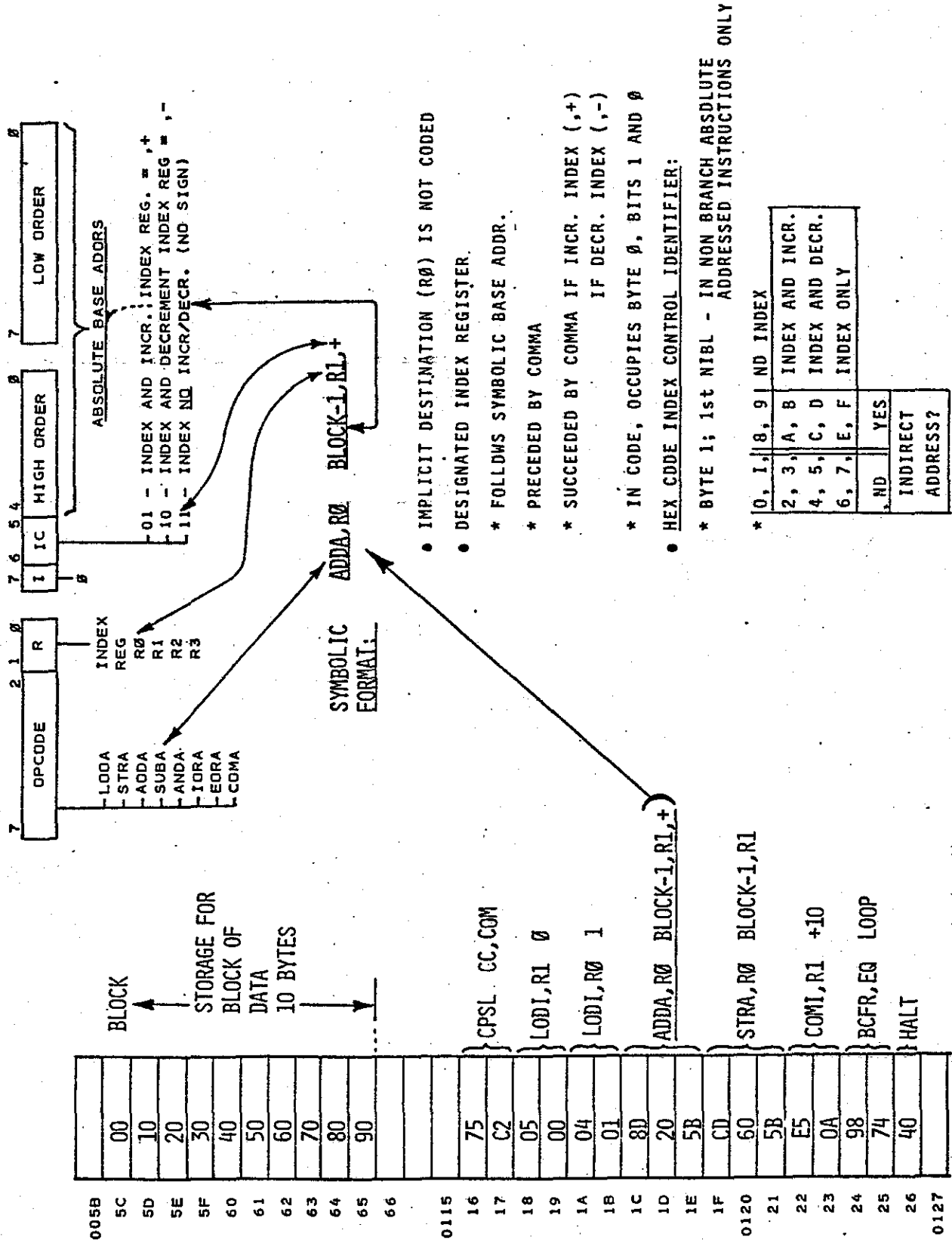
- Note data access from widely separated memory locations.

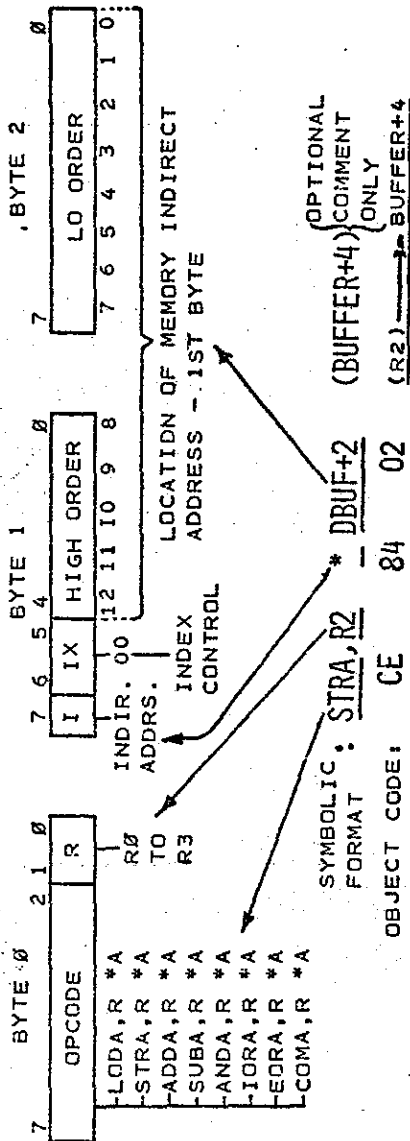
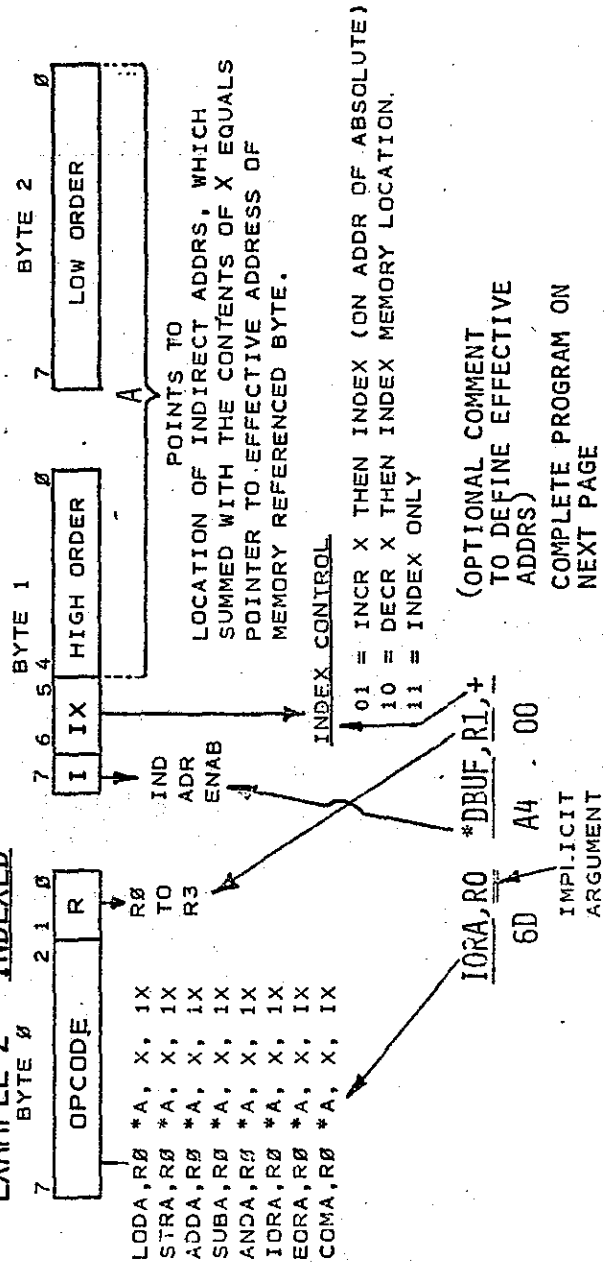
- ```
● Calculate result = 'HEX'
```



INDEXED  
FORMAT A

NON-BRANCH INSTRUCTIONS - ABSOLUTE DIRECT ADDRESS

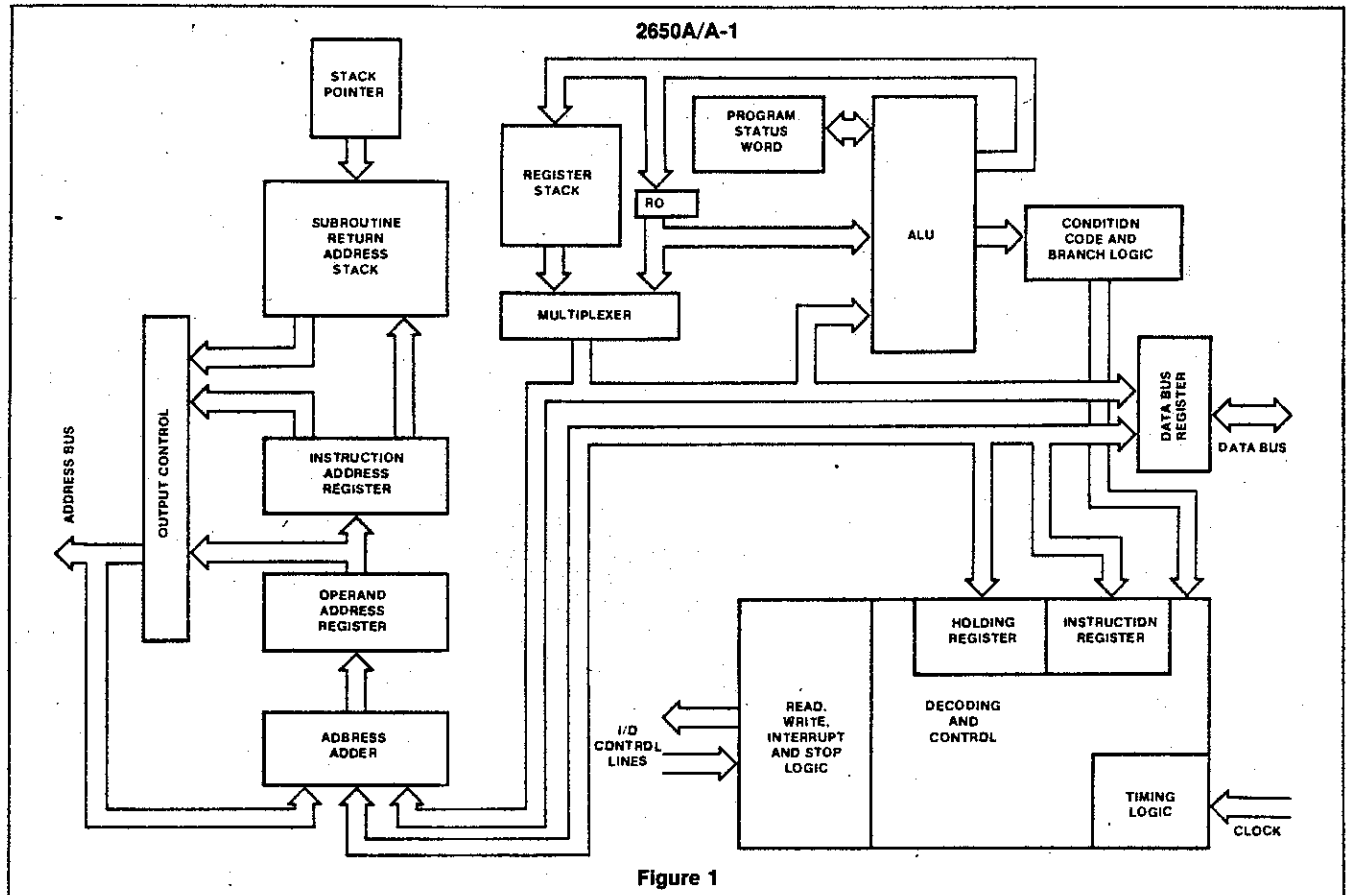


INDIRECT ABSOLUTE ADDRESSED NON-BRANCH INSTRUCTIONS**EXAMPLE 1 NON-INDEXED****EXAMPLE 2 INDEXED**(R0) V BUFFER(X) → (R0)

• USE OF IND. ADDR. PERMITS DATA MANIPULATION FROM LOCATIONS ON ANY PAGE IN MEMORY.

| EXAMPLE 1 |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
|-----------|----|----|----|----|----|----|----|----|----|----|--|--|--|--|--|
| 0109      | CE | 84 | 02 |    |    |    |    |    |    |    |  |  |  |  |  |
| 0A        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 0B        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 0C        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 0D        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 0E        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 0F        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 0110      | 6D | A4 | 00 |    |    |    |    |    |    |    |  |  |  |  |  |
| 11        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 12        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 13        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 14        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 15        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| EXAMPLE 2 |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 0400      | 10 | F0 | 10 | F4 |    |    |    |    |    |    |  |  |  |  |  |
| 01        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 02        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 03        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 04        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 05        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 06        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| 10F0      | 00 | 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 |  |  |  |  |  |
| F1        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| F2        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| F3        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| F4        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| F5        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| F6        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| F7        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| F8        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |
| F9        |    |    |    |    |    |    |    |    |    |    |  |  |  |  |  |

## MICROPROCESSOR BLOCK DIAGRAM



THIS DIAGRAM IS REFERENCED AT LEAST 4 TIMES ON TAPE. YOU MAY WISH TO PARTITION THE SPACE FOR NOTES.

NOTES:

EXERCISE 12 - PRACTICAL USE OF ABSOLUTE ADDRESSED NON BRANCH INSTRUCTIONSINTRODUCTION:

Starting with an examination of absolute direct addressed data transfer instructions used in program "CRAPGAME", this exercise goes on to demonstrate the subtleties of absolute address modes, including index variations. As you sequence through this exercise's steps, consider the merits, both advantages and limitations, imposed by selection of each instruction. This is the process you'll use when you program your application.

Up to this point in the course, you've coded many routines. However, each routine's symbolic instruction sequence has always been provided. Now, in this and future exercises, you'll be COMPOSING, as well as coding routines either in whole or in part. When you perform this function, consider your approach. Consider the tradeoffs between memory conservation and speed of execution of a specific routine's function. Consider the ever-increasing choice of instructions you have mastered and which are now at your fingertips. There will always be a suggested solution to each problem included as part of the exercise. As your skill increases and independence grows, you'll often find that your approach ... and instruction sequence differ markedly with that solution. So what! What is important is that the given routine performs its specified function within tolerable time and memory (size) limits. Use the approach outlined in this paragraph and you'll discover some solutions which really are more cohesive and better structured.

By necessity, this course is structured around the suggested solutions to problems when it was written. Your "instructor" will be the first to admit that some solutions are not the best. They are valid, however, and should serve adequately as a basis for learning. So take the time to prove out your solutions, using the facilities of the "INSTRUCTOR". And build, from this point onward, a LIBRARY of programs and routines, both those contained in this course, and more importantly, the ones which you write. Keep an up-to-date index. Revise it constantly to reflect the contents of your library. When you program your application, you'll borrow extensively from it ... as every programmer who preceded you has!

One more word about TIME of INSTRUCTION SEQUENCE EXECUTION. Reference the PROGRAMMING "HARD" CARD - 265D MICROPROCESSOR INSTRUCTION SET: Specifically the column labeled "CYCLES". While this course has not considered instruction execution time independently as yet, it is sufficient to indicate that the numbers in the CYCLES column indicate a function of time to execute a given instruction type ... in a given address format. The higher the number, the more time is required to execute. Whenever you select an INDIRECT ADDRESS variation, add 2 cycles to the execution time. Consider also that all branch instructions require 3 machine cycles (minimum-direct address) execution time. For example, you can save a memory byte by converting an absolute direct addressed branch into a ZBSR indirect instruction. But you increase the time for execution of that function by 2 cycles. If your application can spare the time, do it. If time is a problem, avoid it.

**DIRECTION:**

Complete the following steps, listening to tape 7A as required.

1. Access CRAPGAME listing main program, sheets 2-3 (Module 1, pages 42-43).
2. Referring to sheet 2; Set a BREAKPOINT ADDRESS at location '65'.

Set a START ADDRESS at location '00' and depress **RUN**.

Play the CRAPGAME until the BREAKPOINT is displayed.

3. Inspect and jot down the contents of the following locations:

SAVPAS1 = \_ \_  
XXFOR+6 = \_ \_  
SHOOT+6 = \_ \_  
XXFOR+7 = \_ \_  
SHOOT+7 = \_ \_

NOTE: Refer to sheet 3, lines 3-7 to derive the memory locations from the code.

4. Also inspect and jot down the contents of REGISTER 0. R0= \_ \_.
5. Set a BREAKPOINT at location '72' and depress **RUN**.
6. Inspect and jot down the contents of the same locations defined in step 5.

SAVPAS1 = \_ \_  
XXFOR+6 = \_ \_  
SHOOT+6 = \_ \_  
XXFOR+7 = \_ \_  
SHOOT+7 = \_ \_

7. In your own words, explain what was accomplished by executing the block of instructions from locations '66' to '72'. Compare your answer to that provided in tape 7A.
8. Using instructions to which you have been introduced, program an alternative approach to moving the roll from XXFOR to SHOOT. Two of the instructions must be indexed absolute addressed data transfer. For other instructions, use any address mode. Use memory locations between '69' and '74' for your routine. Pad unused bytes with NOP instructions: DO NOT inspect the suggested solution until you have debugged your routine using the "INSTRUCTOR" facilities ... (or unless you are totally lost!).

## SUGGESTED SOLUTION - PROBLEM OUTLINED IN STEP 8

|   |      |    |    |    |          |               |                            |
|---|------|----|----|----|----------|---------------|----------------------------|
| 4 | 0069 | 06 | 02 |    | LODI, R2 | 2             | Set roll bytes to be moved |
| 5 | B    | 0E | 77 | 9D | LODA, R0 | XXFOR+6-1, R2 | Now, move a byte from      |
| 6 | 6E   | CE | 77 | BD | STRA, R2 | SHOOT+6-1, R2 | XXFOR to SHOOT message     |
| 7 | 71   | FA | 78 |    | BDRR, R2 | \$-6          | Finished? No! Move another |
| 8 | 73   | CO | CO |    | NOP      |               | Yes! move on to CONTROL    |

## ALTERNATIVE SUGGESTED SOLUTION

|    |      |    |    |    |          |                |                            |
|----|------|----|----|----|----------|----------------|----------------------------|
| 10 | 0069 | 06 | 02 |    | LODI, R2 | 2              | Set roll bytes to be moved |
| 11 | B    | 0E | 57 | 9E | LODA, R0 | XXFOR+6, R2, - | Decrement Index and move   |
| 12 | 6E   | CE | 77 | BE | STRA, R0 | SHOOT+6, R2    | a byte from XXFOR to SHOOT |
| 13 | 71   | 5A | 78 |    | BRNR, R2 | \$-6           | Finished? No! Move another |
| 14 | 73   | CO | CO |    | NOP      |                | Yes! move on to CONTROL    |

9. Listen to the audio tape for a brief commentary on the suggested solutions (above).
10. Compare either of the alternatives (above) with the original program in lines 4 through 7 of sheet 2 (CRAPGAME main program) as to instruction cycles time for execution. Jot down the total number of cycles in the space provided below.
- Original Program: \_\_\_\_\_ cycles. Either alternative (above): \_\_\_\_\_ cycles.
11. Compare your answers with those provided on tape, then go on to Exercise 13.

////////////////////////////////////

### EXERCISE 13 - PRACTICAL USE OF ABSOLUTE ADDRESSED NON BRANCH INSTRUCTIONS (CONTINUED)

#### Introduction:

In many applications, it is often necessary that a block of memory be cleared (zeroed), or that a specified block of addresses be loaded with a KNOWN pattern. The pattern may be any of the following types:

- Fixed ..... a single code (e.g., '3C')
- Incremented ... e.g., '02', '03', '04', ... '33', '34' ... stored in successive locations.
- Decrementd ... e.g., 'A1', 'A0', '9F', ... '7C', '7B' ... stored in successive locations.

**DIRECTION:** Perform the procedure steps on the next page.

PROCEDURE:1. Clear a specific block of memory

- Label: CLBLOC • Start Address: loc '30'.
- Memory Symbolic Location: "BLOC" ACON '1780'.
  - Pattern: '00'.
  - Operate on Memory Locations: '1780' to '17BF'.
  - Loop Labels (if required) use "AGAIN", "ANOTHER", etc.

**DIRECTIONS:**

Compose, code, and checkout your program using the facilities of the INSTRUCTOR. Make each program as concise as possible. Then compare your routine to the suggested solution on the next page. Listen to tape 7A for a brief commentary on the suggested solution.

2. Load a FIXED PATTERN into specified memory

- Label: FIXPAT • Start Address: loc '40'.
- Pattern: '3D'.
  - Operate on Memory Locations: '1780' to '17BF'.
  - Memory Symbolic Location: "BLOC" ACON '1780'.
  - Loop Labels - as in step 1 or your choice.

**DIRECTIONS:**

Exactly as listed in step 1.

3. Increment (by 1) the contents of a specified memory block

- Label: INCBLOC • Start Address: loc '60'.
- Pattern '05' '34'.
  - Operate on Memory Locations: '120' to '14F'.
  - Memory Symbolic Location: "IBLOC" ACON '120'.
  - Loop Labels: Your Choice.

**DIRECTIONS:**

Exactly as listed in step 1 except suggested solution on page 74.

4. Decrement (by 1) the contents of a specified memory block

- Label: DECBLOC • Start Address: loc '80'.
- Pattern: 'FF' 'CB'.
  - Operate on Memory Locations: '150' to '184'.
  - Memory Symbolic Location: "DBLOC" ACON '120'.
  - Loop Labels: Your Choice.

**DIRECTIONS:**

Exactly as listed in step 1 except suggested solution on page 74.



**NOTES:**

| SUGGESTED SOLUTION |       |      |    |       | ROUTINE 'CLBLOC' |                      |               |                             |
|--------------------|-------|------|----|-------|------------------|----------------------|---------------|-----------------------------|
|                    | ADDRS | DATA |    |       | LABEL            | SYMBOLIC INSTRUCTION |               | COMMENT                     |
|                    |       | B0   | B1 | B2    |                  | OPCODE               | OPERANDS      |                             |
| 1                  |       |      |    |       |                  |                      |               |                             |
| 2                  | 0030  | 20   |    |       | CLBLOC           | EORZ                 | R0            | Clear R0 - pattern gen.     |
| 3                  |       | C1   |    |       |                  | STRZ                 | R1            | Transfer to R1 - index, and |
| 4                  |       | 2    | 06 | 40    |                  | LODI, R2             | 64            | set R2 as a byte counter    |
| 5                  |       | 4    | CD | 3F 7F | AGAIN            | STRB, R0             | BLOC-1, R1, + | Now clear a byte in BLOC    |
| 6                  |       | 7    | FA | 7B    |                  | BDRR, R2             | AGAIN         | Finished? No! Clear another |
| 7                  |       | 4    | B0 |       |                  | WRTE                 | R0            | Yes! Exit to monitor.       |

~~~~~

NOTES:

| SUGGESTED SOLUTION |      |    |    |       | ROUTINE 'FIXPAT' |          |                |                           |
|--------------------|------|----|----|-------|------------------|----------|----------------|---------------------------|
| 9                  | 0040 | 04 | 3D |       | FIXPAT           | LODI, R0 | H'3D'          | Set pattern in R0 then    |
| 10                 |      | 2  | 05 | 00    |                  | LODI, R1 | 0              | set initial index and     |
| 11                 |      | 9  | 06 | 0A    |                  | LODI, R2 | 10             | number of bytes.          |
| 12                 |      | 6  | CD | 21 0F | ANOTHER          | STRA, R0 | FBLOC-1, R1, + | Now, store the pattern in |
| 13                 |      | 9  | FA | 7B    |                  | BRR, R2  | ANOTHER        | FBLOC. When finished,     |
| 14                 |      | 8  | 30 |       |                  | WRTC     | R0             | exit to MONITOR           |

NOTES:

| ALTERNATIVE SOLUTION |      |            |  | ROUTINE 'FIXPAT' |          |                |                            |
|----------------------|------|------------|--|------------------|----------|----------------|----------------------------|
| 16                   | 0040 | 04 3D      |  | FIXPAT           | LODI, R0 | H'3D           | same as placed in a        |
| 17                   |      | 2 05 00    |  |                  | LODI, R1 | 0              | above secret in loading    |
| 18                   |      | 4 06 F6    |  |                  | LODI, R2 | H'F6'          | R2 (Buts Ct.) with a       |
| 19                   |      | 6 CD 21 0F |  |                  | STRA, R0 | FBLOC-1, R1, + | number which will inc.     |
| 20                   |      | 9 DA 7B    |  |                  | BR, R2   | ANOTHER        | permanent from F6 to 60    |
| 21                   |      | B B0       |  |                  | WRTC, R2 | R0             | This method not as easy to |

NOTES:

## SUGGESTED SOLUTION

## ROUTINE 'INCBLOC'

|   | ADDRS | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |                | COMMENT                    |
|---|-------|------|----|----|---------|----------------------|----------------|----------------------------|
|   |       | B0   | B1 | B2 |         | OPCODE               | OPERANDS       |                            |
| 1 |       |      |    |    |         |                      |                |                            |
| 2 | 0060  | 04   | 05 |    | INCBLOC | LODI, R0             | 5              | Set initial pattern in R0  |
| 3 | 2     | 05   | 00 |    |         | LODI, R1             | 0              | initialize the index and   |
| 4 | 4     | 06   | 30 |    |         | LODI, R2             | 48             | Byte Counter, then store   |
| 5 | 6     | CD   | 21 | 1F | INCBYTE | STRA, R0             | IBLOC-1, R1, + | pattern in a byte of mem.  |
| 6 | 8     | B4   | 01 |    |         | ADDI, R0             | 1              | ary. Increment the pattern |
| 7 | 8     | FA   | 79 |    |         | BDRR, R2             | INCBYTE        | All stored yet? No! store. |
| 8 | D     | B0   |    |    |         | WRTC                 | R0             | Yes! exit to Monitor.      |

## ALTERNATIVE SOLUTION

## ROUTINE 'INCBLOC'

|    |      |    |    |    |         |          |                |                        |
|----|------|----|----|----|---------|----------|----------------|------------------------|
| 9  | 0060 | 04 | 05 |    | INCBLOC | LODI, R0 | 5              | same explanation as    |
| 10 | 2    | 05 | 00 |    |         | LODI, R1 | 0              | above, except R3       |
| 11 | 4    | 06 | 30 |    |         | LODI, R2 | 48             | contains constant "1"  |
| 12 | 6    | 07 | 01 |    |         | LODI, R3 | 1              | for addition to R0     |
| 13 | 8    | CD | 21 | 1F | INCBYTE | STRA, R0 | IBLOC-1, R1, + | saves a time cycle for |
| 14 | 8    | B3 |    |    |         | ADD      | R3             | each byte transferred. |
| 15 | C    | FA | 7A |    |         | BDRR, R2 | INCBYTE        |                        |
| 16 | E    | B0 |    |    |         | WRTC     | R0             |                        |

## RECOMMENDED SOLUTION:

## ROUTINE 'INCBLOC'

|    |      |    |    |    |         |          |                |                           |
|----|------|----|----|----|---------|----------|----------------|---------------------------|
| 18 | 0060 | 04 | 04 |    | INCBLOC | LODI, R0 | 4              | R0 dual function as INDEX |
| 19 | 2    | 06 | 30 |    |         | LODI, R2 | 48             | and PATTERN GENERATOR     |
| 20 | 4    | CC | 21 | 1B | INCBYTE | STRA, R0 | IBLOC-5, R0, + | R2 is Byte Ctr. Store a   |
| 21 | 7    | FA | 7B |    |         | BDRR, R2 | INCBYTE        | byte in IBLOC. When done, |
| 22 | 9    | B0 |    |    |         | WRTC     | R0             | exit to Monitor.          |

////////////////////////////////////

NOTES:

## SUGGESTED SOLUTION

## ROUTINE 'DECBLOC'

|    |      |    |    |    |         |          |              |                           |
|----|------|----|----|----|---------|----------|--------------|---------------------------|
| 14 | 0080 | 04 | FF |    | DECBLOC | LODI, R0 | FF           | init. pattern generator & |
| 15 | 2    | C1 |    |    |         | STRZ     | R1           | set an INDEX, also set    |
| 16 | 3    | 06 | 35 |    |         | LODI, R2 | H'35'        | up byte counter. Then     |
| 17 | 5    | CD | 21 | 50 | DECREM  | STRA, R0 | DBLOC, R1, + | store decrem. pattern     |
| 18 | 8    | A4 | 01 |    |         | SUBI, R0 | 1            | in defined memory         |
| 19 | A    | FA | 79 |    |         | BDRR, R2 | DECREM       | When finished, exit to    |
| 20 | C    | B0 |    |    |         | WRTC     | R0           | the monitor.              |

DIRECTION:

After listening to the commentary on tape 7A for routine DECBLOC, go on to the next page.

EXERCISE 14LINKING OF ROUTINES INTO A SINGLE PROGRAM

INTRODUCTION: Sooner or later, the various routines which comprise the program design must be linked together, making it possible for the microprocessor to execute one routine after another without outside assistance on the part of the programmer. In this exercise, you'll link routines "CLBLOC", "FIXPAT", "INCBLOC", and "DECBLOC" together, execute from address 0, and re-turn to the MONITOR only after the last routine is executed.

PROCEDURE:

1. LINK the routines: "CLBLOC", "FIXPAT" (alternative solution), "INCBLOC" (recommended solution) and "DECBLOC" together as a PROGRAM labeled "EXEC". Use one of three basic branch instruction alternatives: BCTR,UN; ZBSR; or BSTR,UN. Modify each routine's ending instruction as necessary. Do Not Execute Until Step 6.
2. After coding your sequence and modifying the routines' ending instructions, compare your program to the applicable solution on the next page.
3. Listen to tape 7A for a short analysis of each alternative.
4. Code and execute a short program, at location 'A0', to clear memory locations '100' through '1FF'.
5. Use FAST PATCH mode to enter random data in several memory locations between addresses '1780' and '17BF'.
6. Execute any version of "EXEC", preferably one in which the entire MAIN PROGRAM is fully documented.

NOTE: ZBSR and BSTR versions are fully documented.

DIRECTION:

After completing this exercise go on to the study of absolute addressed branch instructions which follow.

NOTES:

PROGRAM 'EXEC' CODE

BCTR, UN METHOD

|   | ADDRS | DATA |    |    | LABEL | SYMBOLIC INSTRUCTION |          | COMMENT                      |
|---|-------|------|----|----|-------|----------------------|----------|------------------------------|
|   |       | B0   | B1 | B2 |       | OPCODE               | OPERANDS |                              |
| 1 |       |      |    |    |       |                      |          |                              |
| 2 | 0000  | 1B   | 2E |    | EXEC  | BCTR, UN             | CLBLOC   | Jump from S4B = 0            |
| 3 | 39    | 1B   | 05 |    |       | BCTR, UN             | FIXPAT   | Jump from CLBLOC to FIXPAT   |
| 4 | 4B    | 1B   | 13 |    |       | BCTR, UN             | INCBLOC  | Jump from FIXPAT to INCBLOC  |
| 5 | 69    | 1B   | 15 |    |       | BCTR, UN             | DECBLOC  | Jump from INCBLOC to DECBLOC |
| 6 |       |      |    |    |       |                      |          |                              |
| 7 |       |      |    |    |       |                      |          |                              |

NOTES:

PROGRAM 'EXEC' CODE

ZBSR METHOD

|    | ADDRS | DATA |    |    | LABEL    | SYMBOLIC INSTRUCTION |            | COMMENT                     |
|----|-------|------|----|----|----------|----------------------|------------|-----------------------------|
|    |       | B0   | B1 | B2 |          | OPCODE               | OPERANDS   |                             |
| 8  | 0000  | 3B   | 30 |    | EXEC     | ZBSR                 | CLBLOC     | Call subroutine CLBLOC      |
| 9  | 2     | BB   | 89 |    |          | ZBSR                 | * FIXPAT1  | On return, call FIXPAT (*)  |
| 10 | 4     | BB   | 8B |    |          | ZBSR                 | * INCBLOC1 | On return, call INCBLOC (*) |
| 11 | 6     | BB   | 8D |    |          | ZBSR                 | * DECBLOC1 | On return, call DECBLOC (*) |
| 12 | 8     | B0   |    |    |          | WRTC                 | RO         | On return, exit to MONITOR  |
| 13 | 9     | 00   | 40 |    | FIXPAT1  | ACON                 | FIXPAT     | } INDIRECT ADDRESS TABLE    |
| 14 | B     | 00   | 60 |    | INCBLOC1 | ACON                 | INCBLOC    |                             |
| 15 | D     | 00   | 80 |    | DECBLOC1 | ACON                 | DECBLOC    |                             |
| 16 | 0039  | 17   |    |    |          | RETC, UN             |            | } UNAND UNCONDITIONAL       |
| 17 | 004B  | 17   |    |    |          | RETC, UN             |            |                             |
| 18 | 0069  | 17   |    |    |          | RETC, UN             |            |                             |
| 19 | 008C  | 17   |    |    |          | RETC, UN             |            | RETURN FROM SUBROUTINE      |

NOTES:

PROGRAM 'EXEC' CODE

BSTR, UN METHOD

|    | ADDRS | DATA |    |    | LABEL    | SYMBOLIC INSTRUCTION |            | COMMENT                    |
|----|-------|------|----|----|----------|----------------------|------------|----------------------------|
|    |       | B0   | B1 | B2 |          | OPCODE               | OPERANDS   |                            |
| 4  | 0000  | 3B   | 2E |    | EXEC     | BSTR, UN             | CLBLOC     | Call subroutine CLBLOC     |
| 5  | 2     | 3B   | 3C |    |          | BSTR, UN             | FIXPAT     | On return, call FIXPAT     |
| 6  | 4     | 3B   | 83 |    |          | BSTR, UN             | * INCBLOC1 | On return, call INCBLOC    |
| 7  | 6     | 3B   | 83 |    |          | BSTR, UN             | * DECBLOC1 | On return, call DECBLOC    |
| 8  | 8     | B0   |    |    |          | WRTC                 | RO         | On return, call to MONITOR |
| 9  | 9     | 00   | 60 |    | INCBLOC1 | ACON                 | INCBLOC    | } INDIRECT ADDR. TABLE...  |
| 10 | B     | 00   | 80 |    | DECBLOC1 | ACON                 | INCBLOC    |                            |
| 11 | 0039  | 17   |    |    |          | RETC, UN             |            | } UNAND UNCONDITIONAL      |
| 12 | 4B    | 17   |    |    |          | RETC, UN             |            |                            |
| 13 | 69    | 17   |    |    |          | RETC, UN             |            |                            |
| 14 | 8C    | 17   |    |    |          | RETC, UN             |            | RETURN FROM SUBROUTINE     |

ABSOLUTE ADDRESSED BRANCH INSTRUCTIONSFORMAT BINTRODUCTION:

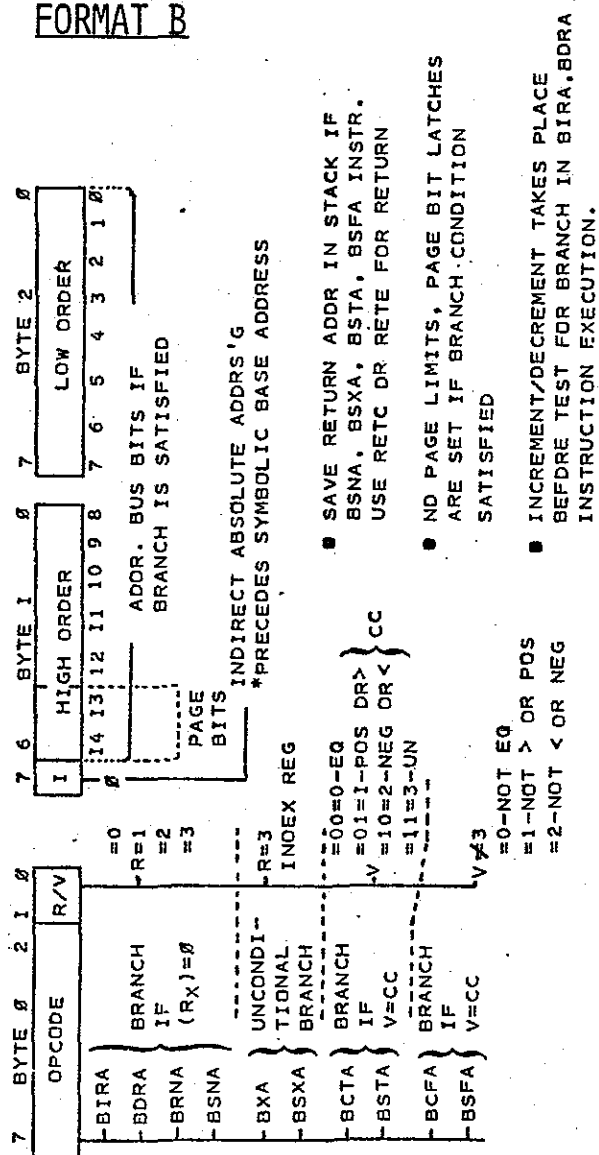
The flexibility of the microprocessor's instruction set is enhanced by its ability to process a complete repertoire of absolute addressed program control branch instructions. Except for zero-referenced (ZBSR,ZBRR) instructions, each of the branch instructions previously described may be specified in absolute address mode as well as in relative address. In addition, the microprocessor may execute two variations of INDEXED branches; these include the BXA and BSXA instructions.

The general variations of Direct and Indirect absolute addressed branch instructions are analyzed in the next 2 pages. A series of exercises follow to demonstrate the practical aspects of their usage and flexibility. Indexed Branch instructions are described later.

DIRECTION:

Take a few minutes to study the parameter diagrams for absolute direct addressed branch on the next page. Then, listen to tape 7A for a brief commentary.

PROGRAM BRANCH INSTRUCTIONS - I - DIRECT ABSOLUTE ADDRESS  
FORMAT B



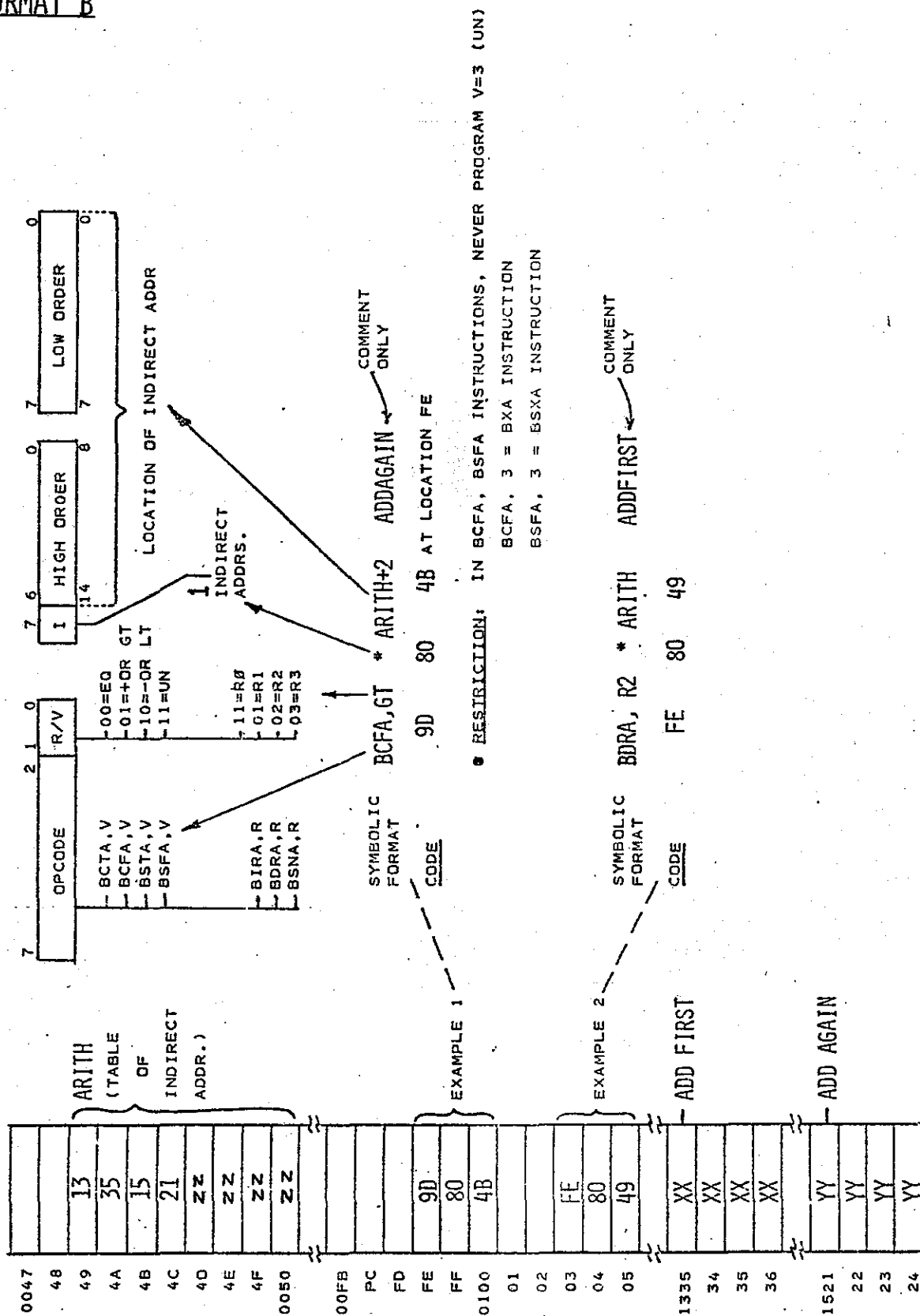
**EXAMPLE 1**    BCTA,UN ADRTN AT LOCATION '0072'

EXAMPLE 2 BIRA, RI, CHKSNS AT LOCATION '0077'

00 CHKSENSE TO SEE IF STILL INACTIVE?  
 01 YES -- GO SET FLAG & FIND OUT WHY  
 02 NO -- ITS ACTIVE, GO CONTINUE ROUTINE

CODE

[illegible]

PROGRAM BRANCH INSTRUCTIONS - INDIRECT ABSOLUTE ADDRESS  
FORMAT B

EXERCISE 15 PRACTICAL USE OF ABSOLUTE-ADDRESSED BRANCH INSTRUCTIONSINTRODUCTION:

In Exercise 14, LINK-up of a series of routines into a single program was demonstrated, using 3 variations of RELATIVE addressed branch instructions as models. In this exercise, rewrite the program "EXEC", using a combination of BSTR and BSTA instructions which will eliminate all indirect addresses. Use minimum memory for storage of "EXEC" while retaining "EXEC"'s character as a main program from which the subroutines CLBLOC, FIXPAT, etc., are called.

After coding and debugging your program, compare it to the suggested solution on the next page. Then listen to a brief commentary and further directions on tape 7A.

////////////////////////////////////

EXERCISE 16 PRACTICAL USE OF ABSOLUTE ADDRESSED BRANCH INSTRUCTIONSINTRODUCTION:

The power of a microprocessor is often measured by its capability to react to the varied demands of its controlled external I/O devices. In this application, the "demand" is made KNOWN by externally controlled I/O to the microprocessor via dedicated I/O sense or status lines. The microprocessor is programmed to read this "status" via an input I/O port. Instructions are then executed to interpret the logical condition of I/O status and to vector program execution to one of several routines designed to service the input command request. Just which routine is selected depends upon "device sense or status".

With respect to program organization, routines such as those which read and interpret external device demands are usually written as part of the main control program. If a number of external devices are attached to the microprocessor, the main program may also contain a routine which will "poll" each device in turn, interrogating each as to its current "busy status" or new demand. Or, the main program may contain a routine which only polls controlled devices for "busy" status. When it finds one which isn't "busy", the control program may for example vector into a data table from which it can determine the last command asserted to that device. Then the processor may index into another table, this time to assert a new command to the device.

In programs involving device polling, selection, status interpretation, and command assertion (i.e., device control programs), you'll find great flexibility in use of absolute addressed branch instructions, particularly use of those in which indexing is permitted.

**DIRECTION:**

Exercise 16 continues at the mid-point of the next page.



'EXEC'  
MAIN PROGRAM

|    |      |    |    |    |      |          |         |                                          |
|----|------|----|----|----|------|----------|---------|------------------------------------------|
| 16 | 0000 | 3B | 2E |    | EXEC | BSTR, UN | CLBLOC  | Call subroutine CLBLOC.                  |
| 17 | 2    | 3B | 3C |    |      | BSTR, UN | FIXPAT  | On return, call FIXPAT                   |
| 18 | 4    | 3F | 00 | 60 |      | BSTR, UN | INCBLOC | On return, call INCBLOC                  |
| 19 | 7    | 3F | 00 | 80 |      | BSTR, UN | DECBLOC | On return, call DECBLOC                  |
| 20 | A    | 30 |    |    |      | WRTC     | RO      | On return, exit to monitor               |
| 21 | 0039 | 17 |    |    |      | RETC, UN |         | UNCONDITIONAL RETURN<br>FROM SUBROUTINES |
| 22 |      | 17 |    |    |      | RETC, UN |         |                                          |
| 23 |      | 17 |    |    |      | RETC, UN |         |                                          |
| 24 |      | 17 |    |    |      | RETC, UN |         |                                          |

NOTES:

**DIRECTION:**

After listening to the tape, go back to Exercise 16 on the previous page.

////////////////////

**EXERCISE 16 INTRODUCTION (CONTINUED)**

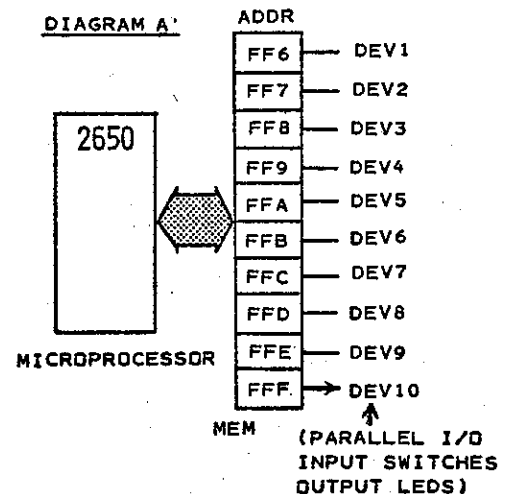
Exercises 16 through 18 provide means by which you will become familiar with programming techniques required to implement I/D control and service functions.

**EXERCISE 16 DESCRIPTION**

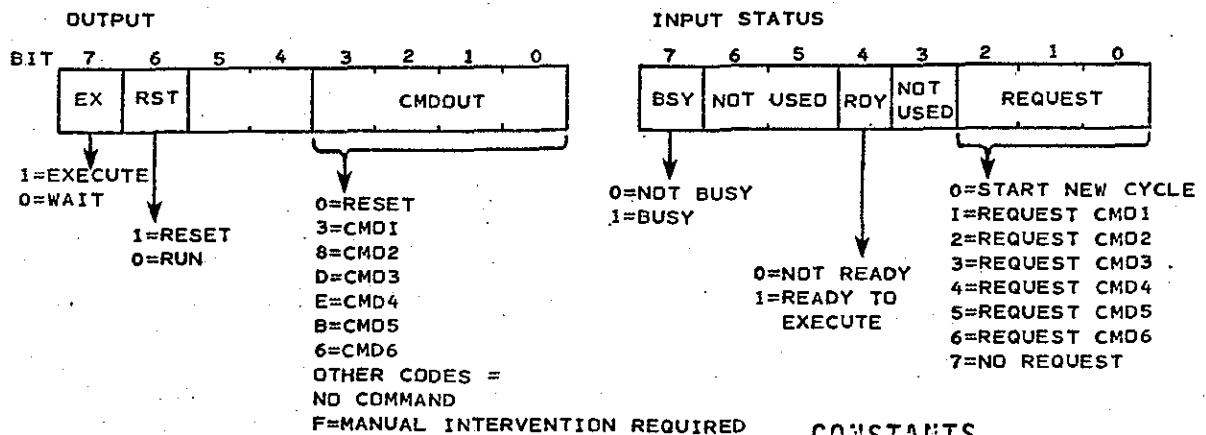
The "INSTRUCTOR"'s on-board parallel I/O capability will be implemented in MEMDRY MAPPED I/D MODE. In the program "POLL", which you will code, 10 memory addresses ('FF6" through "FFF") are dedicated as memory-mapped I/O parts to service 10 "pseudo" external controlled I/O Devices (Diagram A).

Through your manipulation of the parallel I/D input toggle switches, only one port, at address 'FFF', will be able to request service. The others will always input a "BUSY-ND REQUEST" status.

Input and Output lines between the microprocessor and "controlled I/D" are defined on the next page. Refer to these as you write selected portions of the program.



## SYMBOLIC LINE DEFINITION - "DEV1" THROUGH "DEV10" - (1 PORT ILLUSTRATED)



## CONSTANTS

R1 = DEVICE ADDRESS INDEX  
\$ = H'7E' (1ST BYTE CURRENT INSTRUCTION)

## ADDRESS CONSTANTS

POLL = H'0000' - PROGRAM NAME  
UPDATE = H'0004' - LABEL  
READCMD = H'0100' - SUBROUTINE  
1ORST = H'0120' - SUBROUTINE  
WAIT = H'0125' - LABEL  
DELAY = H'0150' - SUBROUTINE  
CMDPAT = H'015B' - SUBROUTINE  
CMDTBL = H'0190' - DATA TABLE  
CONTAINS CMDOUT CODES  
DEVX = H'0FF6' - DEVICE ADDRESS  
EQUATES ADDR OF DEVICE 1

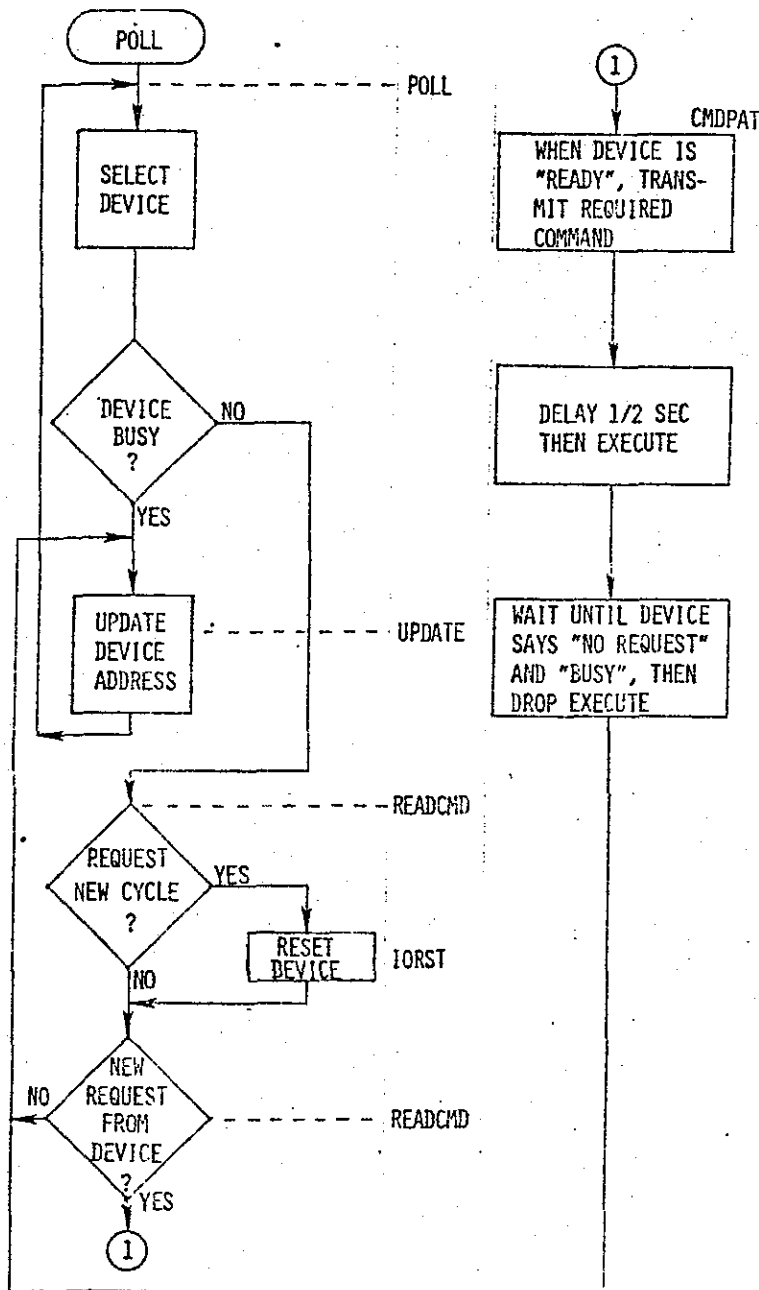
RS = H'10' - REGISTER SELECT  
BSY = H'80' - BUSY  
REQUEST = H'07'  
NOREQUEST = H'07'  
MANINT = H'0F' - MANUAL INTERVENTION  
REQUIRED CMD

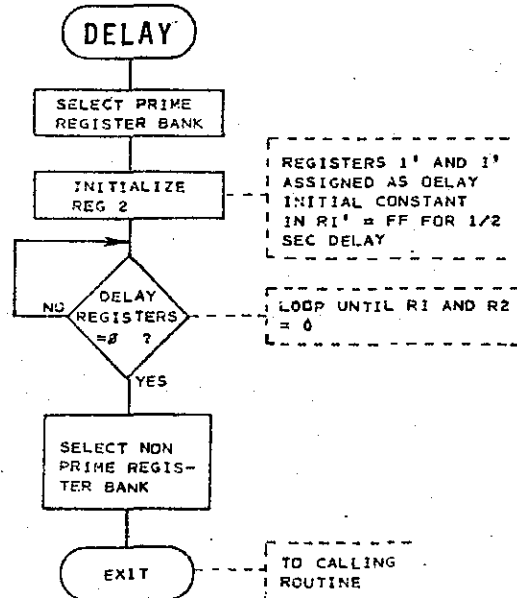
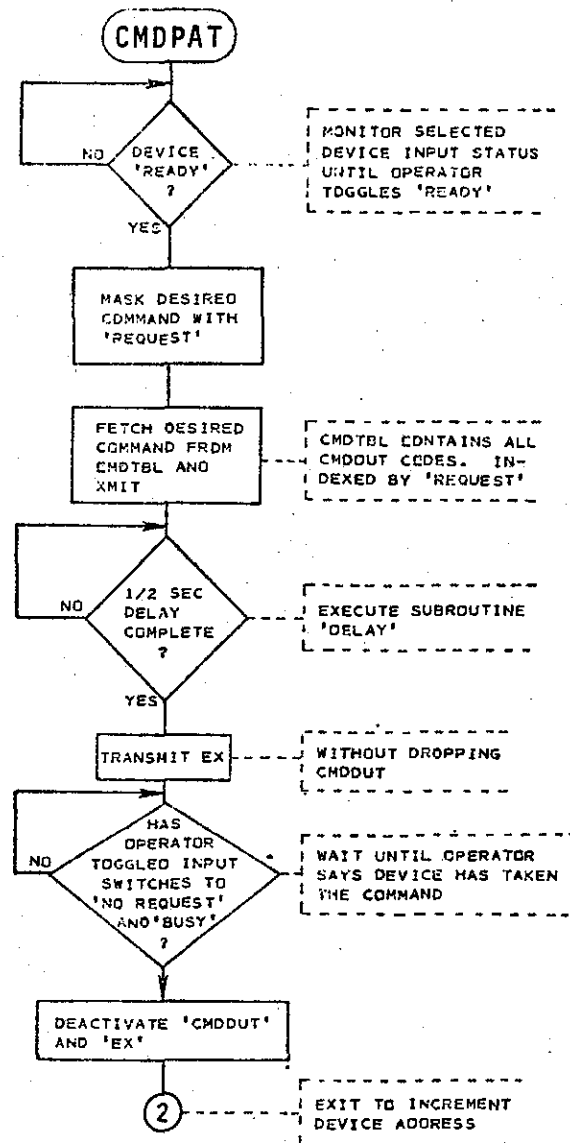
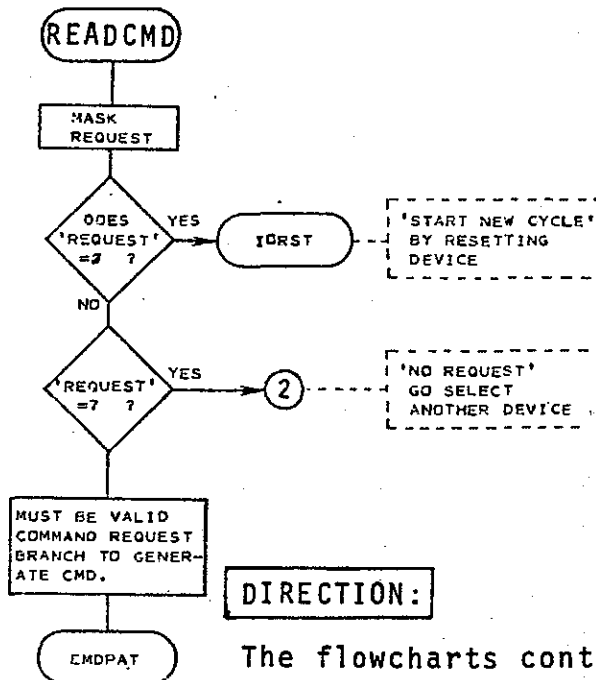
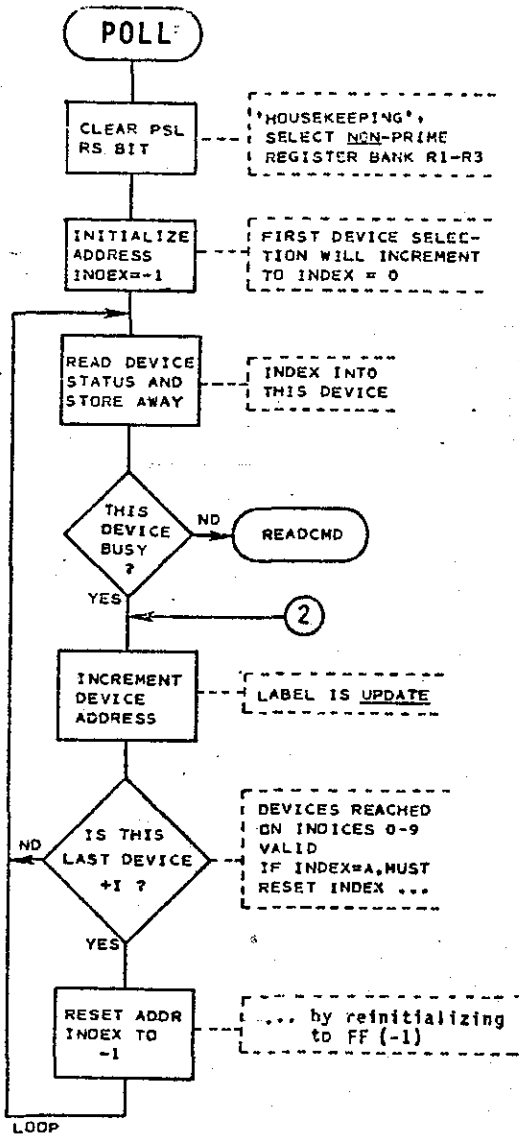
RDY = H'10' - DEVICE READY  
RST = H'40' - DEVICE RESET/RUN  
EX = H'80' - EXECUTE/WAIT  
RESET = H'00' - RESET COMMAND  
LT = H'02' - LESS THAN

INSTRUCTOR I/O SELECTION SWITCH:  
'MEMORY OFF' POSITION

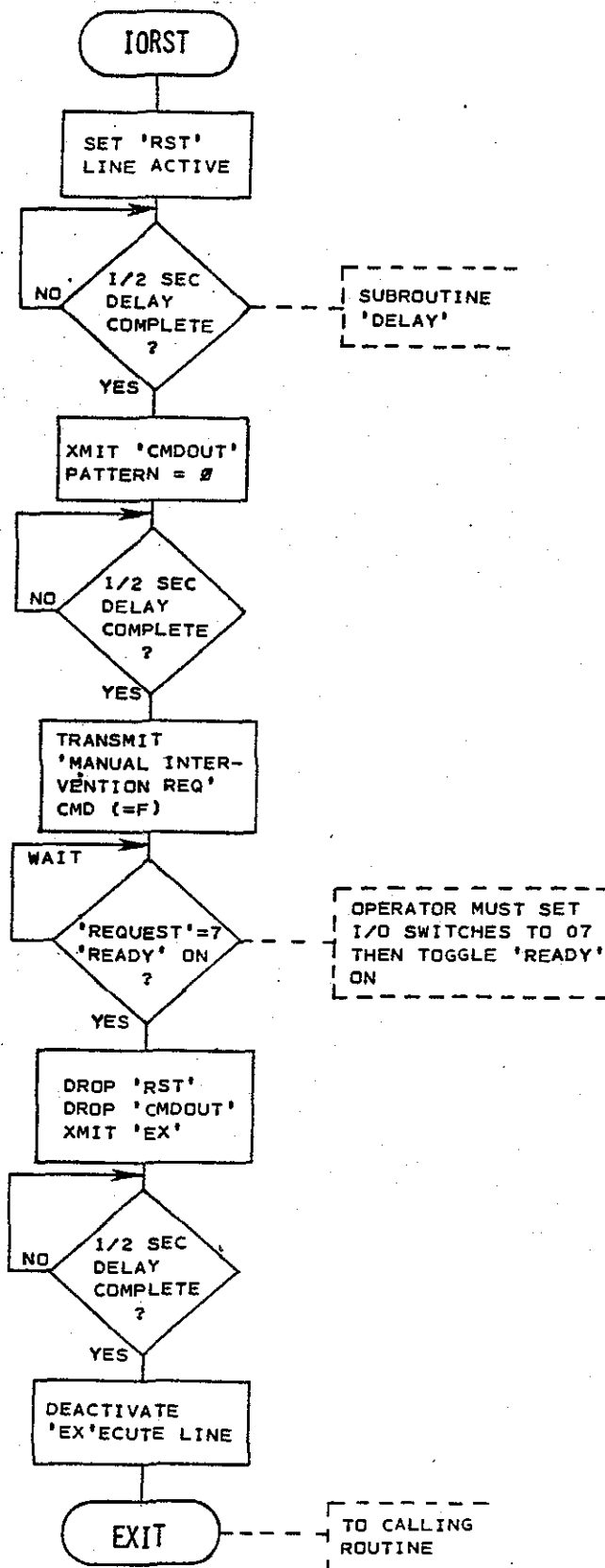
## DIRECTION:

Listen to Tape 7B for further explanation of Exercise 16, referenced to Diagram A on page 81.





The flowcharts continue on the next page.



**DIRECTION:** Go on to the next page when directed to do so on audio tape.

EXERCISE 16 - PRACTICAL USE OF ABSOLUTE ADDRESSED BRANCH INSTRUCTIONS (CONTINUED)PROCEDURE:

1. Working from the detailed specifications and flowcharts on the three previous pages, compose and code the program "POLL" on forms from the programming pad.
2. As you complete each process or subroutine of program "POLL", load it into the "INSTRUCTOR"'s memory and debug it. Select instructions which you feel will execute each desired function. Take the time to experiment with variations, particularly with branch instruction alternatives.
3. During debug, make extensive use of the "INSTRUCTOR"'s BKPT and SINGLE STEP facilities. Use the flowcharts to determine your manipulation of the PARALLEL I/O INPUT TOGGLE switches.
4. You are URGED to consult, but not necessarily copy, the suggested coded solution of program "PDLL" at any time during your activities. The following points should be considered:
  - (1) In the version illustrated, NO ATTEMPT is made to conserve memory usage. A rewrite of the program could save upwards of 20 bytes. If you can spare the time, rewrite "PDLL".
  - (2) The TMI (Test under Mask Immediate) instruction is implemented several times.

NOTE: The TMI instruction is designed for use in testing the logical condition of selected bits such as BSY and RDY, and selected BIT PATTERNS such as REQUEST, as described in the detailed flowcharts.

If all of the selected bits are logic 1, the condition code of the PSW (lower) is set to 00 (EQ). If any tested bit is logic 0, the condition code is set to 10 (LT).

In a program sequence, the TMI instruction is followed by a suitable Branch instruction (e.g., BCTR/A BCFR/A BSTR/A BSFR/A). Decisions performed by these instructions are based on the condition code generated by execution of the TMI instruction. Since the desired CC is predictable, selection of the appropriate branch instruction should be easy ... and accurate.

Refer to the 2650 Reference manual for further information. The TMI instruction is described further in Module IV-H of this course.

- (3) For demonstration purposes, most branch instructions and many non branch instructions are illustrated in absolute address modes.
- (4) There is no further taped commentary on this exercise. The suggested code, however, is fully documented.

**DIRECTION:**

After completing your activities, go on to study the parameter diagrams for indexed absolute addressed branch instructions. These diagrams are located on pages 88 and 89. Listen to tape 7B after a few minutes of study.

## SUGGESTED CODE

## PROGRAM "POLL"

## NOTES:

USE THIS SPACE TO  
JOT DOWN POINTS YOU  
WISH TO RECALL.  
YOU MAY ALSO WISH  
TO DENOTE CHANGES  
YOU MIGHT MAKE TO  
CONSERVE MEMORY --  
OR TO EXECUTE IN  
LESS TIME -- .

AN EXAMPLE TO  
CONSERVE MEMORY  
IS SHOWN OPPOSITE  
LINE 9.\*

## CHANGE:

ELIMINATE THIS  
INSTRUCTION.

NEXT INSTRUCTION:  
BCTR,UN UPDATE-2  
AT LOCATION 11.

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| +                                        | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 |
| -                                        | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 |
| +                                        | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |
| +                                        | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| -                                        | 70 | 6F | 6E | 6D | 6C | 6B | 6A | 69 | 68 | 67 | 66 | 65 | 64 | 63 | 62 |
| +                                        | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E |
| +                                        | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A |
| -                                        | 60 | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 |
| +                                        | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E |
| +                                        | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A |
| -                                        | 40 | 3F | 3E | 3D | 3C | 3B | 3A | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| +                                        | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E |
| +                                        | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 5A |
| -                                        | 50 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 |

INDIRECT RELATIVE ADDRESS: Add N\*60 to displacement

## 2650 PROGRAMMING FORM

ROUTINE POLL START ADDR 0000

DESCRIPTION This program reads and trans-  
ports status from 10 controlled I/O devices.  
Upon request, it transmits a specific output  
command to the selected device.

ROUTINE SHEET 1 OF 3

MEMORY LOCATIONS THIS SHEET \_\_\_\_\_

| ADDRES          | DATA<br>B2 B1 B0 | LABEL   | SYMBOLIC INSTRUCTION<br>OPCODE OPERANDS | COMMENT                                |
|-----------------|------------------|---------|-----------------------------------------|----------------------------------------|
| 1 0000          | 75 10            | POLL    | CPSL RS                                 | SELECT NON-PRIME REGISTERS             |
| 2 2 05 FF       |                  |         | LODI, R1 'FF'                           | INITIALIZE INDEX TO I/O                |
| 3 4 0D 2F F6    |                  | UPDATE  | LODA, R0 DEVX, R1, +                    | Now, get selected device sta-          |
| 4 7 C3          |                  |         | STRZ R3                                 | two and temp. store in R3              |
| 5 8 44 80       |                  |         | ANDI, R0 BSY                            | MAST Busy! Is this device              |
| 6 A BE 01 00    |                  |         | BSFA, LT READCMD                        | busy? No! Go read CMDREQ.              |
| 7 D E5 0A       |                  |         | COMI, R1 10                             | Yes! Any more devices this             |
| 8 0F 9B 73      |                  |         | BCFR, EQ UPDATE                         | pass? Yes. Select next de-             |
| 9 11 05 FF      |                  |         | LODI, R1 'FF'                           | vice. No! Reinitialize index           |
| 10 013 1B 6F    |                  |         | BCTR, UN UPDATE                         | and go select 12 <sup>th</sup> device. |
| 11              |                  |         |                                         |                                        |
| 12 0100 03      |                  | READCMD | LODZ R3                                 | Get saved status and mask              |
| 13 1 44 07      |                  |         | ANDI, R0 REQUEST                        | REQUEST. IF REQUEST = 0,               |
| 14 3 3C 01 20   |                  |         | BSTA, EQ IORST                          | go reset the device. Well, if          |
| 15 6 E4 07      |                  |         | COMI, R0 NOREQUEST                      | there is NOREQUEST, exit to            |
| 16 8 1B 03      |                  |         | BCTR, EQ \$+5                           | poll next device. IF REQUEST           |
| 17 A 3F 01 5B   |                  |         | BSTA, UN CMDPAT                         | is for valid cmd, go setup             |
| 18 10D 17       |                  |         | RETC, UN                                | Command Pattern sequence.              |
| 19              |                  |         |                                         |                                        |
| 20 0120 04 40   |                  | IORST   | LODI, R0 RST                            | Raise RST and transmit                 |
| 21 2 CD 6F F6   |                  |         | STRA, R0 DEVX, R1                       | to selected device. Hold it            |
| 22 5 3F 01 50   |                  | WRIT    | BSTR, UN DELAY                          | for 1/2 sec, then, arrange             |
| 23 8 44 F0      |                  |         | ANDI, R0 H'F0'                          | that CMDOUT = 0 and                    |
| 24 A CD 6F F6   |                  |         | STRA, R0 DEVX, R1                       | transmit with RST. Hold                |
| 25 D 3B F7      |                  |         | BSTR, UN *WAIT+1                        | it another 1/2 sec, then set           |
| 26 12F 24 0F    |                  |         | EORI, R0 MANINT                         | "MANUAL INTERVENTION RE-               |
|                 |                  |         |                                         | QUIRED" command.                       |
| 2 0131 CD 6F F6 |                  |         | STRA, R0 DEVX, R1                       | Transmit MANINT + RST                  |
| 3 4 0D 6F F6    |                  |         | LODA, R0 DEVX, R1                       | When operator responds by              |
| 4 7 17          |                  |         | TMI, R0 RBY, NOREQUEST                  | together RBY and NOREQUEST             |
| 5 9 79          |                  |         | BCFR, EQ \$-5                           | switches, drop RST and MAN-            |
| 6 8 80          |                  |         | LODI, R0 EX                             | INT and raise "EX"ecute                |
| 7 3D CD 6F F6   |                  |         | STRA, R0 DEVX, R1                       | Transmit, then wait 1/2 sec            |
| 8 0140 3B E4    |                  |         | BSTR, UN *WAIT+1                        | On return clean R0                     |
| 9 2 20          |                  |         | EORZ R0                                 | and                                    |
| 10 3 C F6       |                  |         | STRA, R0 DEVX, R1                       | drop "EX"ecute, then return            |
| 11 046 17       |                  |         | RETC, UN                                | to calling routine.                    |

NOTE: THE ORIGINAL PROGRAM WAS CODED ON 3 FORMS. THIS IS THE REASON  
FOR THE LINE NUMBER SEQUENCE BREAKS AT LOCATION '131' AND '16C'.

## NOTES:

## SUGGESTED CODE

## PROGRAM "POLL"

2650 PROGRAMMING FORM

ROUTINE \_\_\_\_\_ START ADDR \_\_\_\_\_

DESCRIPTION \_\_\_\_\_

ROUTINE SHEET \_\_\_\_\_ OF \_\_\_\_\_

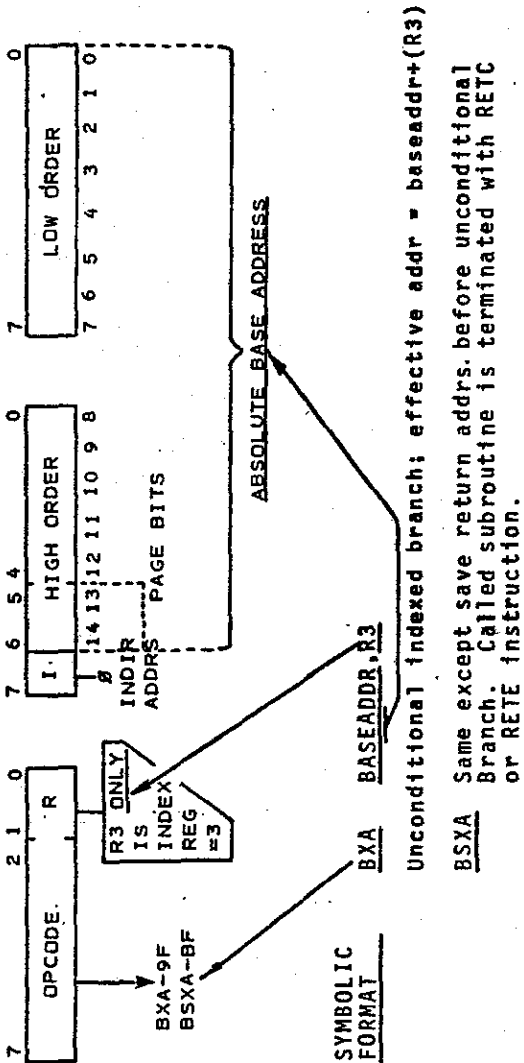
MEMORY LOCATIONS THIS SHEET \_\_\_\_\_

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| +                                        | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| +                                        | 1E | 1F | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C |
| +                                        | 2D | 2E | 2F | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B |
| +                                        | 3C | 3D | 3E | 3F | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A |
| +                                        | 4B | 4C | 4D | 4E | 4F | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| +                                        | 5A | 5B | 5C | 5D | 5E | 5F | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 |
| +                                        | 69 | 6A | 6B | 6C | 6D | 6E | 6F | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| +                                        | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F | 80 | 81 | 82 | 83 | 84 | 85 | 86 |
| +                                        | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F | 90 | 91 | 92 | 93 | 94 | 95 |
| +                                        | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F | A0 | A1 | A2 | A3 | A4 |
| +                                        | A5 | A6 | A7 | A8 | A9 | AA | AB | AC | AD | AE | AF | B0 | B1 | B2 | B3 |
| +                                        | B4 | B5 | B6 | B7 | B8 | B9 | BA | BB | BC | BD | BE | BF | C0 | C1 | C2 |
| +                                        | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF | D0 | D1 |
| +                                        | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | DA | DB | DC | DD | DE | DF | E0 |
| +                                        | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | EA | EB | EC | ED | EE | EF |
| +                                        | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | FA | FB | FC | FD | FE |
| +                                        | FF |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

INDIRECT RELATIVE ADDRESS: Add N\*60' to displacement

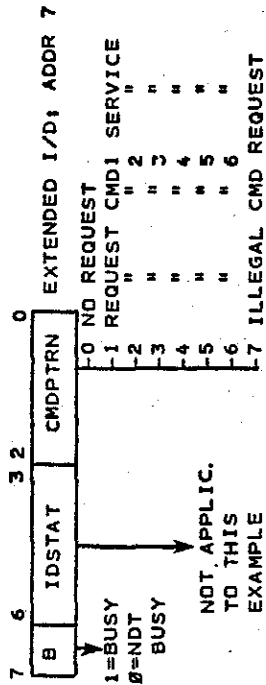
| ADDRS | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |                | COMMENT                       |
|-------|------|----|----|--------|----------------------|----------------|-------------------------------|
|       | B0   | B1 | B2 |        | OPCODE               | OPERANDS       |                               |
| 13    | 0150 | 77 | 10 | DELAY  | PPSL                 | RS             | On entry, select prime regis- |
| 14    | 2    | 06 | B0 |        | LODI, R2             | 'B0'           | ter bank, set delay constant  |
| 15    | 4    | F9 | 7E |        | BDRR, R1             | \$             | for 1/2 sec. Delay, on        |
| 16    | 6    | FA | 7C |        | BDRR, R2             | \$-2           | completion, select non-prime  |
| 17    | 8    | 75 | 10 |        | CPSL                 | RS             | bank again, then exit to      |
| 18    | A    | 17 |    |        | RETC, UN             |                | calling routine.              |
| 19    |      |    |    |        |                      |                |                               |
| 20    | 015B | 0D | 6F | CMDPAT | LODA, R0             | DEVX, R1       | On entry, wait til operator   |
| 21    | 5E   | F4 | 10 |        | TMI, R0              | RDY            | has device ready to accept    |
| 22    | 60   | 98 | 79 |        | BCFR, EQ             | \$-5           | command. Then mask REQ-       |
| 23    | 2    | 44 | 07 |        | ANDI, R0             | REQUEST        | UEST and use it as an index   |
| 24    | 4    | 0C | 61 |        | LODA, R0             | CMDTBL, R0     | to fetch valid cmd pattern.   |
| 25    | 7    | CD | 6F |        | STRA, R0             | DEVX, R1       | and transmit to the device.   |
| 26    | 16A  | 3F | 01 |        | BSTA, UN             | DELAY          | Now, delay about 1/2 sec      |
| 27    |      |    |    |        |                      |                |                               |
| 1     |      |    |    |        |                      |                |                               |
| 2     | 016D | 64 | 80 |        | LORI, R0             | EX             | On return tell device to take |
| 3     | 16F  | CD | 6F |        | STRA, R0             | DEVX, R1       | the command. He will when     |
| 4     | 172  | 0D | 6F |        | LODA, R0             | DEVX, R1       | he flips the switch to busy   |
| 5     | 5    | F4 | 87 |        | TMI, R0              | BSY, NOREQUEST | and NOREQUEST (?) code.       |
| 6     | 7    | 98 | 79 |        | BCFR, EQ             | \$-5           | Wait til he does, then        |
| 7     | 9    | 2D |    |        | EORZ                 | R0             | clear R0 mode to drop         |
| 8     | A    | CD | 6F |        | STRA, R0             | DEVX, R1       | EX and CMDOUT. Then           |
| 9     | 17D  | 17 |    |        | RETC, UN             |                | return to calling routine     |
| 10    |      |    |    |        |                      |                |                               |
| 11    | 0190 | 00 | 03 | CMDTBL | RES                  | 8              | CMDOUT DATA TABLE             |
| 12    |      | 0D | 0E |        |                      |                | This table contains all       |
| 13    |      | 06 | 0F |        |                      |                | valid output cmd. codes.      |
| 14    |      |    |    |        |                      |                |                               |

INDEXED BRANCH INSTRUCTIONS - DIRECT ABSOLUTE ADDRESS



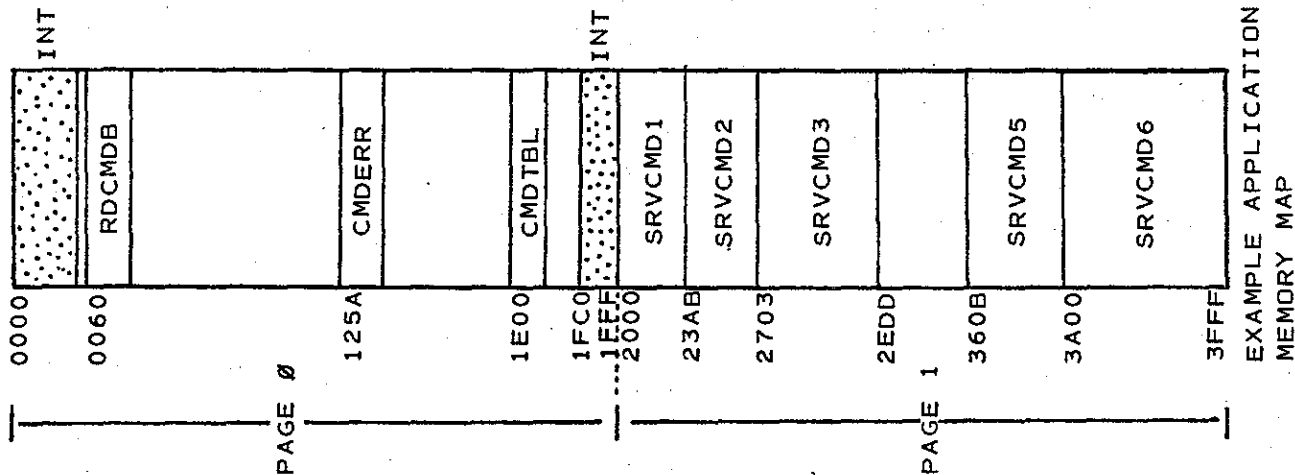
EXAMPLE

APPLICATION: Routine in which selected I/O requests one of several command processes based on input pattern.

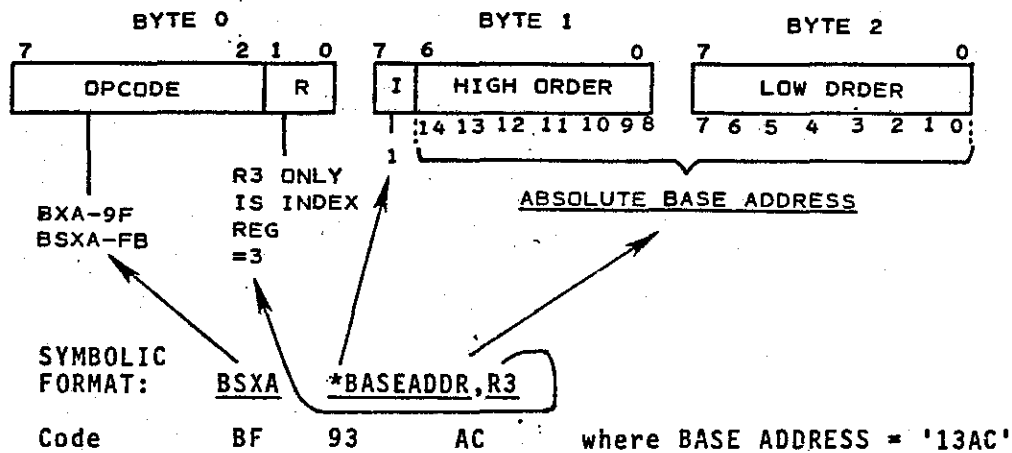
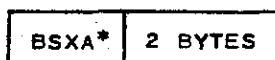


SYMBOLIC EQUATES

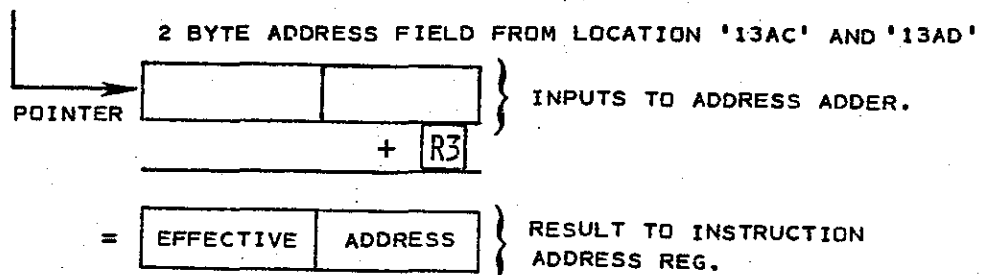
| SYMBOLIC EQUATES  | SYMBOLIC ADDRESS CONSTANTS |
|-------------------|----------------------------|
| BUSY EQU H'80'    | OTHER ROUTINE ACON XXXX    |
| CMDPTRN EQU H'07' | RDCMDB ACON 006D           |
| RX EQU X          | ACDN 1E00                  |
| XCMTDBL EQU 3     | ACON 2DD0                  |
|                   | ACON 23AB                  |
|                   | ACON 2703                  |
|                   | ACON 2EDD                  |
|                   | ACON 360B                  |
|                   | ACON 3A00                  |





INDEXED BRANCH INSTRUCTIONS - INDIRECT ABSOLUTE ADDRESSINSTRUCTION

INDIRECT ADDRESS TO  
OPERAND ADDRESS REG

NOTES:**DIRECTION:**

Go on to the next page as indicated on tape 7B.

EXERCISE 17 - PRACTICAL USE OF INDEXED ABSOLUTE BRANCH VARIATIONSINTRODUCTION:

In EXERCISE 16, you wrote one of many variations of I/O - interactive programs. In concept, the program "POLL" interfaced the microprocessor with "SMART I/O", devices which could perform their designed functions independently from the microprocessor. The microprocessor's task was limited to issuing simple commands upon request. Further, only the simplest of "handshaking" took place in the assertion of EX and RST to the selected I/O device. If the "functions" performed by the selected device were at all complex, we must assume that their execution was controlled by each device's internal logic.

This costs money! It is precisely in the interest of saving device logic expense that many systems houses have introduced microprocessor control of multiplexed I/O devices. The concept is simple: Replace costly internal logic with an inexpensive memory.

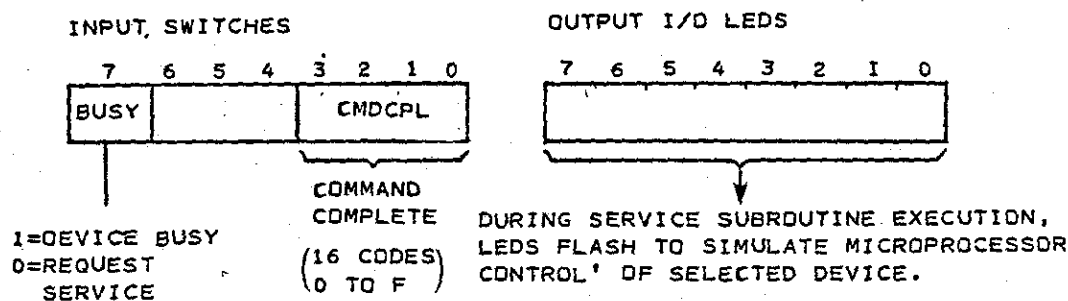
The program itself may consist of many routines, each with an instruction sequence of several hundred bytes. In designing programs such as these, the programmer must implement effective housekeeping functions; those processes which keep the microprocessor informed of each device's status and which will vector program control to routines required to service each controlled I/O device. While the microprocessor may control a number of the same device type, each device may be performing a different function at any given time. Which leads us to a description of Exercise 17.

PROGRAM SRVCIO DESCRIPTION:

In this exercise, a program is executed to vector program control to any one of 16 routines. Initially, the microprocessor polls each of 8 extended I/O ports (addresses '04'-'0B'). Each port is presumably attached to an 'external controlled device'. Since the INSTRUCTOR on-board parallel (I/O capability permits access to one port, only that port (at address '07') can respond. All other "ports" will input 'FF' - "not attached". Upon detecting that Port 7 is not BUSY, the microprocessor reads the port's "COMMAND COMPLETE" code. This code is manipulated as an index to a table of 16 I/O SERVICE routines. The subroutine number is always 1 higher than the COMMAND COMPLETE code. If CMD CPL=F, Subroutine 0 is executed. After the subroutine is processed, the microprocessor returns to poll all devices again.

Each service routine can access a visual "process simulation" routine. Process simulation simultaneously decrements the I/O LEDs and displays the port address and subroutine number selected. The duration of process simulation is determined by the service subroutine selected.

**DIRECTION:** Go right on to the next page.

OBJECTIVE LINE DEFINITION - EACH PORT:SYMBOLIC DEFINITIONS:EQUATES:

RS = H'10' = REGISTER SELECT  
 RC = H'08' = WITH CARRY  
 BUSY = H'80' = TEMP STORAGE  
           FOR PORT ADDR  
           INDEX (SEQUENTIAL)

PORT4  
 THRU = H'04' THRU H'0B'

PORTB

OSPLXT = H'7F' = DISPLAY CONSTANT  
 DLYXT = H'FF' = DELAY CONSTANT  
 CMD CPL = H'0F' = COMMAND COMPLETE MASK  
 LT = H'02' = LESS THAN  
 GT = H'01' = GREATER THAN

MEMORY ADDRESS CONSTANTS:

SRVCIO = H'0000' = SERVICE I/O PROGRAM  
 PORTIND = H'1780' = TEMP. STORAGE FOR  
           PORT ADDRESS INDEX  
 INPORT = H'0004' = PROCESS TO READ  
           DEVICE STATUS  
 RDPDRT = H'0019' = TABLE OF INPUT  
           PORT CALL STATEMENTS  
 MESSAGE = H'1788' = 8 BYTE MESSAGE TABLE  
 GETPORT = H'0051' = PORT ADDRESS ENCODE  
           PROCESS

SUBO = H'0100' AND  
 THRU = FOLLOWING AT = TABLE OF 16 SER-  
 SUBF = H'10' INTERVALS VICE SUBROUTINES  
 SIMPRC = H'0058' = SIMULATE I/O SERVICE  
           PROCESS

PROCEDURE:

1. Use FAST PATCH mode to load program "SRVCIO" into memory (next 3 pages).
2. Take a few minutes to read the COMMENT field.
3. Execute from start address = '0'. Use STEP and BREAKPOINT modes to examine each routine thoroughly. Use the appropriate parallel I/O toggle switches to modify BUSY and the CMD CPL specified in the Objective Line Definition (above). Also execute by depressing **RST**.
4. Vary the program as described in the VARIATIONS section (page 94)
5. Modify the various delay constants to see how much timing control you can exercise on the program.
6. Listen to tape 7B for a short commentary on specific instruction selection related to this program. Do not perform Step 7 until directed to do so on tape.
7. Respond to the questions on page 95.

**DIRECTION:** Ignore the circled numbers until you are requested to respond to questions in step 7. The circled numbers identify points of question.

**NOTES:**

| DIRECT RELATIVE ADDRESSING - SECOND BYTE                |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---------------------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                                       | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| -                                                       | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 |
|                                                         | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 |
| +                                                       | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |
| -                                                       | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
|                                                         | 70 | 6F | 6E | 6D | 6C | 6B | 6A | 69 | 68 | 67 | 66 | 65 | 64 | 63 | 62 |
| +                                                       | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E |
| -                                                       | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 |
|                                                         | 60 | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 |
| +                                                       | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E |
| -                                                       | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 |
|                                                         | 52 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 |
| INDIRECT RELATIVE ADDRESSING: Add N*50° TO DISPLACEMENT |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

## NOTES:

|    | ADDRS | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |           | COMMENT                        |
|----|-------|------|----|----|---------|----------------------|-----------|--------------------------------|
|    |       | E3   | S1 | S2 |         | OPCODE               | OPERANDS  |                                |
| 12 | 6     | 02   |    |    |         | LODZ                 | R2        | Now, get new command           |
| 13 | 7     | 05   | 04 |    |         | LODI, R1             | 4         | (= CMDCTL+1); shift left       |
| 14 | 9     | D0   |    |    |         | RRL                  | R0        | 4 times to establish index to  |
| 15 | A     | F9   | 7D |    |         | BDRR, R1             | \$-1      | correct subroutine and put     |
| 16 | C     | C3   |    |    |         | STRZ                 | R3        | IN BRANCH INDEX REG (R3)       |
| 17 | 4D    | BF   | 81 | 00 |         | BSXA                 | *SUB0, R3 | Then branch to execute         |
| 18 | 50    | 17   |    |    |         | RETC, UN             |           | Based SUBROUTINE. On           |
| 19 |       |      |    |    |         |                      |           | return, go back and poll       |
| 20 | 0051  | 84   | 01 |    | GETPORT | ADDI, R0             | 1         | next device                    |
| 21 | 3     | A7   | 03 |    |         | SUBI, R3             | 3         | (1) Establish correct port by  |
| 22 | 5     | 5B   | 7A |    |         | BRNR, R3             | \$-4      | incrementing port # and decre- |
| 23 | 57    | 17   |    |    |         | RETC                 |           | menting index by 3 to 0        |
| 24 |       |      |    |    |         |                      |           | Repeat until 0, then exit=0    |
| 25 | 1788  | 10   | XX | 16 | MESSAGE | RES                  | 8         | Px = SUBx. Displays            |
| 26 | 8     | 05   | 12 | 0B |         |                      |           | Port address and current       |
| 27 | E     | 17   | XX |    |         |                      |           | subroutine being processed     |

|    | ADDRS | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |             | COMMENT                        |
|----|-------|------|----|----|--------|----------------------|-------------|--------------------------------|
|    |       | E3   | S1 | S2 |        | OPCODE               | OPERANDS    |                                |
| 1  |       |      |    |    |        |                      |             | To simulate process, transmit  |
| 2  | 0058  | D7   | 07 |    | SIMPRC | WRITE, R3            | PORT7       | R3 (former branch index) to    |
| 3  | A     | 77   | 10 |    |        | PSSL                 | RS          | selected port. Now select prim |
| 4  | C     | 07   | 7F |    |        | LODI, R3             | DSPLXT      | reg, bank, and load USER       |
| 5  | 5E    | 05   | 17 |    |        | LODI, R1             | > MESSAGE-1 | DISPLAY constants - MESSAGE    |
| 6  | 60    | 06   | 87 |    |        | LODI, R2             | < MESSAGE-1 | and LS byte of address (-1)    |
| 7  | 2     | BB   | E6 |    |        | ZBSR                 | *USRDSP     | and exit to display. On re-    |
| 8  | 9     | 75   | 10 |    |        | CPSL                 | RS          | turn, reselect non-prime       |
| 9  | 6     | 05   | FF |    |        | LODI, R1             | DLYXT       | vars and set a short delay.    |
| 10 | 8     | F9   | 7E |    |        | BDRR, R1             | \$          | Affandelay, decrement R3.      |
| 11 | A     | FB   | 6E |    |        | BDRR, R3             | SIMPRC      | When R3=0, simulated pro-      |
| 12 | C     | D7   | 07 |    |        | WRITE, R3            | PORT7       | cess is complete, so turn off  |
| 13 | 6E    | 17   |    |    |        | RETC, UN             |             | LEDS, and return to calling    |
| 14 |       |      |    |    |        |                      |             | subroutine.                    |
| 15 | 0100  | 3F   | 00 | 5B | SUB0   | BSTA, UN             | SIMPRC      | Explanation for all subrou-    |
| 16 | 3     | 17   |    |    |        | RETC, UN             |             | tines is the same.             |
| 17 | 110   | 3F   | 00 | 5B | SUB1   | BSTA, UN             | SIMPRC      | On entry, go simulate the      |
| 18 | 3     | 17   |    |    |        | RETC, UN             |             | process to service the se-     |
| 19 | 120   | 3F   | 00 | 5B | SUB2   | BSTA, UN             | SIMPRC      | lected port. The duration      |
| 20 | 3     | 17   |    |    |        | RETC, UN             |             | of the process is proportional |
| 21 | 130   | 3F   | 00 | 5B | SUB3   | BSTA, UN             | SIMPRC      | to the magnitude of the        |
| 22 | 3     | 17   |    |    |        | RETC, UN             |             | subroutine number and          |
| 23 | 140   | 3F   | 00 | 5B | SUB4   | BSTA, UN             | SIMPRC      | branch index contained         |
| 24 | 3     | 17   |    |    |        | RETC, UN             |             | in R3. On return, exit         |
| 25 | 150   | 3F   | 00 | 5B | SUB5   | BSTA, UN             | SIMPRC      | to calling routine.            |
| 26 | 3     | 17   |    |    |        | RETC, UN             |             | END OF COMMENT                 |
| 27 |       |      |    |    |        |                      |             |                                |

NOTES:

|    | ADDRS | DATA |    |    | LABEL | SYMBOLIC INSTRUCTION |          |
|----|-------|------|----|----|-------|----------------------|----------|
|    |       | B2   | B1 | B2 |       | OPCODE               | OPERANDS |
| 1  | 0160  | 3F   | 00 | 58 | SUB 6 | BSTR, UN             | SIMPRC   |
| 2  | 3     | 17   |    |    |       | RETC, UN             |          |
| 3  | 170   | 3F   | 00 | 58 | SUB 7 | BSTR, UN             | SIMPRC   |
| 4  | 3     | 17   |    |    |       | RETC, UN             |          |
| 5  | 180   | 3F   | 00 | 58 | SUB 8 | BSTR, UN             | SIMPRC   |
| 6  | 3     | 17   |    |    |       | RETC, UN             |          |
| 7  | 190   | 3F   | 00 | 58 | SUB 9 | BSTR, UN             | SIMPRC   |
| 8  | 3     | 17   |    |    |       | RETC, UN             |          |
| 9  | 1A0   | 3F   | 00 | 58 | SUB A | BSTR, UN             | SIMPRC   |
| 10 | 3     | 17   |    |    |       | RETC, UN             |          |
| 11 | 1B0   | 3F   | 00 | 58 | SUB B | BSTR, UN             | SIMPRC   |
| 12 | 3     | 17   |    |    |       | RETC, UN             |          |
| 13 | 1C0   | 3F   | 00 | 58 | SUB C | BSTR, UN             | SIMPRC   |
| 14 | 3     | 17   |    |    |       | RETC, UN             |          |
| 15 | 1D0   | 3F   | 00 | 58 | SUB D | BSTR, UN             | SIMPRC   |
| 16 | 3     | 17   |    |    |       | RETC, UN             |          |
| 17 | 1E0   | 3F   | 00 | 58 | SUB E | BSTR, UN             | SIMPRC   |
| 18 | 3     | 17   |    |    |       | RETC, UN             |          |
| 19 | 1F0   | 3F   | 00 | 58 | SUB F | BSTR, UN             | SIMPRC   |
| 20 | 3     | 17   |    |    |       | RETC, UN             |          |

VARIATIONS:

1.

This variation turns off all I/D LEDs, holding them off 1/3 sec before exiting to calling subroutine.

2. "SCAN"

Address scheme required to access "SCAN".

This subroutine simulates a scan of the devices polled by lighting each parallel I/O LED in sequence, from bit 7 to bit 0. The actual poll of "devices" follows this routine and takes about 200  $\mu$ sec to complete.

|    | ADDRS | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |             | COMMENT                      |
|----|-------|------|----|----|--------|----------------------|-------------|------------------------------|
|    |       | B2   | B1 | B2 |        | OPCODE               | OPERANDS    |                              |
| 1  |       |      |    |    |        |                      |             |                              |
| 2  |       |      |    |    |        |                      |             |                              |
| 3  | 006C  | D7   | 07 |    |        | WRITE, R3            | PORT7       | Transmit R3 (=0) to Port     |
| 4  | E     | 05   | 80 |    |        | LODI, R1             | DLYZX       | Then set clean delay count   |
| 5  | 70    | FA   | 7E |    |        | BDRR, R2             | \$          | and Delay                    |
| 6  | 2     | F9   | 7C |    |        | BDRR, R1             | \$-2        | then return to               |
| 7  | 4     | 17   |    |    |        | RETC, UN             |             | calling subroutine.          |
| 8  |       |      |    |    |        |                      |             |                              |
| 9  |       |      |    |    |        |                      |             | Addressing changes to main   |
| 10 | 0017  | 1B   | 99 |    |        | BCTR, UN             | * CALLSUB-3 | program to show visual scan  |
| 11 |       |      |    |    |        |                      |             | BCTR instruction calls SCAN  |
| 12 | 0032  | 00   | 7E |    |        |                      |             | via this indirect address.   |
| 13 |       |      |    |    |        |                      |             | To scan, load scan reg       |
| 14 | 007E  | 05   | 80 |    | SCAN   | LODI, R1             | SCANXT      | with H'80' and eat WC.       |
| 15 | 80    | 77   | 08 |    |        | PPSL                 | WC          | Then set the scan delay      |
| 16 | 2     | 06   | 20 |    | SCANXT | LODI, R2             | DLY3X       | constant (vary to suit)      |
| 17 | 4     | D5   | 07 |    |        | WRITE, R1            | PORT7       | and Transmit visual scan     |
| 18 | 6     | 51   |    |    |        | RDR                  | R1          | to Port7. Now shift scan     |
| 19 | 7     | F8   | 7E |    |        | BDRR, R0             | \$          | constant right 1 bit and     |
| 20 | 9     | FA   | 7C |    |        | BDRR, R2             | \$-2        | execute delay. Has scan      |
| 21 | B     | 59   | 75 |    |        | BRNR, R1             | SCANXT      | const been displayed in all  |
| 22 | D     | 9B   | 00 |    |        | ZBR                  | SRVCIO      | bits? No! scan again.        |
| 23 |       |      |    |    |        |                      |             | Yes Exit to start of program |

**DIRECTION:** Respond to the following questions as indicated after you have listened to tape 7B. Reference the program "SRVCIO" as specified by the circled numbers. Then, compare your answers to those provided on the next page.

1. Reference (1) ; page 92:
  - a. What was the purpose of clearing R0 ? \_\_\_\_\_
  - b. Could R3 have been zeroed by selection of another instruction ? \_\_\_\_\_  
If "YES", Write the location and symbolic instruction:  
ADDRESS: \_\_\_\_\_ SYMBOLIC INSTRUCTION: \_\_\_\_\_
2. Reference (2) ; page 92:
  - a. Could the EORZ instruction be eliminated ? \_\_\_\_\_ (YES)(NO)
  - b. Why or why not ? \_\_\_\_\_
  - c. If "YES", what would be the correct replacement code ? ' \_\_\_\_\_ '
3. Reference (3) ; page 92:
  - a. Could you have selected a BCTR instruction to replace the BCFR ?  
\_\_\_\_\_ (YES)(NO)
  - b. If "YES", WRITE the symbolic instruction: \_\_\_\_\_
4. a. If you knew that this program (addresses '0' to '1FF') is to be stored in a PROM, would you have to alter any existing NON-BRANCH instructions, knowing that the data which they access is variable ?  
\_\_\_\_\_ (YES)(NO) NOTE: Assume that SMI RAM (locations '17B0' to '17BF') is still available.
- b. Qualify your answer by checking ONE of the following statements:
  - (1) Yes; the instructions referenced on page 92; (4)
  - (2) Yes; the STRA instruction on page 92; (5)
  - (3) No; the program is perfect as written
  - (4) Yes; the message constants identified by (6) ; page 93.
  - (5) No; But you would have to add a short routine to initialize the message constants:  
" P, =,S,U,B,(space) "
5. Reference page 93, (7) :
  - a. What would be the effect of changing DLYXT from 'FF' to '20' ?  
\_\_\_\_\_
  - b. Does this change affect the BRIGHTNESS of the HEX DISPLAY as well ?  
\_\_\_\_\_ (YES)(NO)
6. Reference page 93; (8) :
  - a. Suppose you had used BCTA instead of BSTA instructions to transfer program control to routine "SIMPRC". Would there be a need to modify the RETC instruction at location '006E' ? \_\_\_\_\_ (YES)(NO)  
Select 1 alternative to qualify your answer.
    - (1) Yes; With a BCTA,UN instruction to the calling routine.
    - (2) No ; Leave the RETC instruction as is.
    - (3) IMPOSSIBLE!!! this would create a program sequence error.
7. Reference the "SCAN" variation; page 94; (9) :  
Could you replace the BCTR,UN indirect addr'd instruction with a BCTA,UN absolute direct addr'd instruction ? \_\_\_\_\_ ((YES)(NO)  
Explain: \_\_\_\_\_

ANSWERS TO QUESTIONS ON PAGE 95

1. a. To set up clearing of R3 - no other function.  
b. Yes. Address 0002; LODI,R3 0.
2. a. Yes. (As the program is written)  
b. On entry to "CALLSUB", RO = 0 always.  
c. 'CO' - NOP instruction.
3. a. No! (I tried every variation)  
b.- Not applicable.

NOTE:

|            | BRANCH                  | NO BRANCH                 |
|------------|-------------------------|---------------------------|
| BCTR,EQ    | IF VALUE = CC           | IF CC = 01(GT) OR 10 (LT) |
| BCTR,GT(+) | IF CC = 01(GT)          | IF CC = 00(EQ) OR 10 (LT) |
| BCTR,LT(-) | IF CC = 10(LT)          | IF CC = 00(EQ) OR 01 (GT) |
| BCFR,EQ    | IF CC = 01(GT)OR 10(LT) | IF CC = 00(EQ)            |
| BCFR,GT    | IF CC = 00(EQ)OR 10(LT) | IF CC = 01(GT)            |
| BCFR,LT    | IF CC = 00(EQ)OR 01(GT) | IF CC = 10(LT)            |

4. a. No.  
b. (5) And those message constants could be stored in scratch pad as they are now.
5. a. Shortens execution time of SIMPRC considerably; e.g., when called from SUBF, execution time decreases from about 1-1/2 sec to 1 sec.  
b. Yes. Since countdown delay is shortened, display appears brighter because of decreased interval between calls of USRDSP.
6. a. (3) Try it and see what happens.
7. a. With no other program changes. NO, or YES. By: (1) relocating the ROPORT table in memory between addresses '001A' and '0031'.  
(2) changing the value at location '0009' from '19' to '1A'.  
(3) coding BCTA,UN SCAN at location '0017' ( ... it doesn't appear to be worth it.)

**DIRECTION:**

After comparing your answers, go on to exercise 18.

EXERCISE 18 - PROGRAM MODIFICATION TO EXPAND NUMBER OF I/O DEVICES SERVICED

**SECTION:**

Take about 15 minutes to jot down, on a separate sheet of paper, suggestions you might make to expand the polling capability of this program from 8 to 20 I/O devices. Consider which instructions you would change. Consider also the location of table ROPORT. Assume that all routines, including those associated with the 2 variations, are to be located as illustrated in the program documentation. However, this does not prevent modification or additions to them. After writing your suggestions, compare them to those suggested on the next page.



## SUGGESTED PROGRAM MODIFICATIONS

## EXERCISE 18

1. RDPORT table entry for each device = 3 bytes; 20 devices = 60 bytes  
= '3C' loc in memory
2. 60 bytes must be contiguous - a single block of memory --  
(action: find a suitable block of memory).
3. Arbitrary decision: Initial address of RDPORT table at location  
'90' uses memory addresses '90' through 'CB'.
4. Action: Change code at location '0007' (BSXA instruction) to  
'CF' 00 '90'.
5. Action: Change instruction at (loc '0013' to COMI,R3 60. ('E7' '3C')).
6. Port addresses: from: 04 08 (PORT4 PORT8)  
to : 04 17 (PORT04 PORT17)
7. Available memory left: locations '00C4' through '00FF'  
'0019' through '0031'

(Super! I've got about 70 bytes to play with.)

8. Present Display message:

|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
| P | X | = | S | U | B | X |
|---|---|---|---|---|---|---|

DEPENDS ON  
PORT SELECTED

SUBROUTINE  
NUMBER

(Action - must have 2 digits port selection ('04'-'17'))

9. Possible messages: (a)

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| P | X | X | = | S | U | B | X |
| X | X | = | S | U | B |   | X |
| P | X | X | = | S | r |   | X |
| P | X | X | = | S | b |   | X |

(TOO CROWDED)

(DOESN'T IDENT.  
XX AS A PORT)

(NOT BAD)

(NOT BAD EXCEPT 'B'  
MIGHT BE MISTAKEN  
FOR A 6)

Action: Go with message (c)

10. Message Constants: Locations '1788'-'178F': (from INSTRUCTOR REF MAN)

code (10)(XX)(XX)(16)(05)(93)(17)(XX)

equals P X X = S r. Blank X

11. Revamp "CALLSUB" routine to accommodate message changes.

Desired action: • no change to 1st three instructions - they  
park subroutine number in message.

• anticipate that rest of "CALLSUB" and satellite  
subroutine "GETPORT" will be modified.

• expect port address will go as high as '17',  
therefore user-available monitor routine "NIBL"  
must be used to break calculated address into  
two half bytes, each in a separate register.

• therefore, compose CALLSUB in existing locations  
and do unconditional branch into unused memory  
to access extra locations required by CALLSUB  
and/or GETPORT.

## DIRECTION:

As an outside project, try out these changes; programming  
modifications and new instructions as necessary. Now, go  
on to Exercise 19.

EXERCISE 19ROUTINE RELOCATIONINTRODUCTION:

When using low cost software development tools such as the "INSTRUCTOR", many programmers prefer to write their routines initially in low-order memory. Considerable TIME can be saved in setting breakpoints, changing memory data, etc., during debug. After a routine is debugged, it is useful to relocate it in locations available in high order user RAM. Moreover, the routines may be located precisely adjacent to each other. In so doing, you'll make available those unused bytes which normally separate each routine during debug.

The concept of routine relocation may also be applied to operations involving batch data transfer from one part of memory to another part. For example, considerable memory may be required for storage of multiple files of data. The processor may be programmed to retrieve one file at a time, process it and store it away, then retrieve the next file, etc.

Larger computers have mass storage devices - disc drives and high speed tape decks - available for batch files. On a smaller scale, the INSTRUCTOR has this capability, however limited by manual operations necessary to store to, or fetch a file from, cassette. The most immediate method to create file storage space for a microprocessor driven application is to design sufficient memory expansion. Normally, data will be transferred in blocks from expanded memory to a block of RAM assigned as working storage. In concept, you'll implement processes like those involved in routine relocation to perform this function.

In the following series of programs, several approaches to ROUTINE RELOCATION are examined. Before you load the code for each alternative, store a KNOWN incrementing pattern in memory. By doing this, you'll be able to verify your program's operation exactly.

PROCEDURE: FUNCTIONAL PROGRAM TO GENERATE AN INCREMENTING PATTERNDIRECTION:

Code and load the program INCPAT2 into the "INSTRUCTOR" at location '1780'. This program loads an incrementing pattern (00→FF) into the first 512 locations (0 - '1FF') of user storage. Execute the program, then go on to the next page.

ADDRESS CDNSTANTS: Hi 256=H'0100'=2nd 256 Byte memory block.  
Lo 256=H'0000'=1st 256 Byte memory block.

| ADDRS | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |              | COMMENT                     |
|-------|------|----|----|---------|----------------------|--------------|-----------------------------|
|       | B0   | B1 | B2 |         | OPCODE               | OPERANDS     |                             |
| 1     |      |    |    |         |                      |              |                             |
| 2     | 1780 |    |    | INCPAT2 | EDRZ                 | RO           | zero pattern generator      |
| 3     |      |    |    |         | STRA, RO             | LO256, RO, + | then, incr. pattern & store |
| 4     |      |    |    |         | BRNR, RO             | \$-3         | in 1st 256 byte memory      |
| 5     |      |    |    |         | STRA, RO             | H1256, RO, + | repeat for 2nd 256          |
| 6     |      |    |    |         | BRNR, RO             | \$-3         | bytes, then                 |
| 7     |      |    |    |         | WRTC                 | RO           | exit to monitor             |

## SUGGESTED SOLUTION

## PROGRAM 'INCPAT2'

| ADDRS | DATA |    |       | LABEL   | SYMBOLIC INSTRUCTION |              | COMMENT                      |
|-------|------|----|-------|---------|----------------------|--------------|------------------------------|
|       | B0   | B1 | B2    |         | OPCODE               | OPERANDS     |                              |
| 1     |      |    |       |         |                      |              |                              |
| 2     | 1780 | 20 |       | INCPAT2 | EORZ                 | RO           | clear pattern register, then |
| 3     | 1    | CC | 20 00 |         | STRA, RO             | LO256, RO, + | store inc pattern in         |
| 4     | 4    | 58 | 7B    |         | BRNR, RO             | \$-3         | 1st 256 bytes memory         |
| 5     | 6    | CC | 21 00 |         | STRA, RO             | H1256, RO, + | same for 2nd 256             |
| 6     | 9    | 58 | 7B    |         | BRNR, RO             | \$-3         | Bytes, then                  |
| 7     | B    | B0 |       |         | WRTC                 | RO           | exit to Monitor              |

## EXERCISE 20

## DIRECT ROUTINE RELOCATION - METHOD 1

In which source and destination data start addresses, and the number of bytes to be moved are KNOWN.  $N$  bytes =  $1 < N < 256$ .

Example: SOURCE DATA START ADDRESS = loc '0033'  
 DESTINATION DATA START ADDRESS = loc '0100'  
 move 35 bytes from source to destination.

1. Compose and code a routine (RELOC1) at location '0'. Use the form below:

NOTE: If you wish to compare your routine to that provided on the next page, do so. Define and employ suitable symbolic equates and labels as necessary.

2. Execute your routine.
3. Verify that you have moved the desired 35 bytes (no more; no less)

NOTE: If you move more or less than 35 bytes, correct your program.

| ADDRS | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |          | COMMENT |
|-------|------|----|----|--------|----------------------|----------|---------|
|       | B0   | B1 | B2 |        | OPCODE               | OPERANDS |         |
| 1     | 0000 |    |    | RELOC1 |                      |          |         |
| 2     |      |    |    |        |                      |          |         |
| 3     |      |    |    |        |                      |          |         |
| 4     |      |    |    |        |                      |          |         |
| 5     |      |    |    |        |                      |          |         |
| 6     |      |    |    |        |                      |          |         |
| 7     |      |    |    |        |                      |          |         |
| 8     |      |    |    |        |                      |          |         |

4. Modify the constants in your program to move other blocks of data to different locations in User memory. Repeat step 3, as it fits your modifications.
5. Re-execute program "INCPAT2", then go on to the next exercise.

SUGGESTED SOLUTIONROUTINE RELOC1

|    |      |    |    |    |        |         |               |                     |
|----|------|----|----|----|--------|---------|---------------|---------------------|
| 19 | 0000 | 20 |    |    | RELOC1 | EORZ    | R0            | COMMENTS ARE FREE - |
| 20 | 1    | C1 |    |    |        | STRZ    | R1            | -FORM.              |
| 21 | 2    | 06 | 23 |    |        | LODI,R2 | 35            | BYTE COUNT          |
| 22 | 4    | 0D | 20 | 32 | AGAIN  | LODA,R0 | SOURCE-1,R1,+ |                     |
| 23 | 7    | CD | 60 | FF |        | STRA,R0 | DESTIN-1,R1   |                     |
| 24 | A    | FA | 78 |    |        | BDRR,R2 | AGAIN         |                     |
| 25 | C    | BO |    |    |        | WRTC    | R0            | Exit to MONITOR     |

EXERCISE 21DIRECT ROUTINE RELOCATION - METHOD 2

In which SOURCE data START ADDRESS, DESTINATION data ending (upper) ADDRESS, and the NUMBER of bytes to be moved are known. This routine is appropriately used when routines are to be relocated precisely adjacent to each other, thus eliminating memory addresses which serve no useful function.

EXAMPLE: SOURCE DATA START ADDRESS = loc '0000'  
 DESTINATION DATA ENDING ADDRESS = loc '01FF' } routine  
 BYTE COUNT = 47 RELOC2

PROCEDURE:

1. Compose and code a routine (RELOC2) at address '0E'. Use the form below.

NOTE: Define and implement suitable symbolic equates and labels as necessary. Suggested solution on next page.

2. Execute the routine "RELOC2".
3. Verify that you have moved 47 bytes from source to destination memory as required.

NOTE: If you don't relocate exactly 47 bytes, correct your program.

| ADORS | DATA |    |    | LABEL | SYMBOLIC INSTRUCTION |          | COMMENT |
|-------|------|----|----|-------|----------------------|----------|---------|
|       | B0   | B1 | B2 |       | DPCODE               | OPERANDS |         |
|       |      |    |    |       |                      |          |         |
|       |      |    |    |       |                      |          |         |
|       |      |    |    |       |                      |          |         |
|       |      |    |    |       |                      |          |         |
|       |      |    |    |       |                      |          |         |
|       |      |    |    |       |                      |          |         |

4. Perform steps 4 and 5 of Exercise 20, then go on to the next exercise.

## SUGGESTED SOLUTION

## ROUTINE 'RELOC2'

|    |      |    |    |    |        |          |                 |                                |
|----|------|----|----|----|--------|----------|-----------------|--------------------------------|
| 14 | 000E | 07 | FF |    | RELOC2 | LODI, R3 | > DESTUAD       | subtract byte ct from des.     |
| 15 | 10   | A7 | 2F |    |        | SUBI, R3 | BYTECT          | termination ending addr. and   |
| 16 | 2    | CB | 09 |    |        | STRR, R3 | AGAIN+5         | store as dest. start addr. - 1 |
| 17 | 4    | 0B | 7B |    |        | LODR, R3 | \$-3            | Now set byte count & in-       |
| 18 | 6    | 05 | 00 |    |        | LODI, R1 | 0               | dex, then move a byte          |
| 19 | 8    | 0D | 3F | FF | AGAIN  | LDA, R0  | SRCRAD-1, R1, + | from source to destination     |
| 20 | 8    | CD | 61 | XX |        | STRA, R0 | DESSAD-1, R1    | All done yet?                  |
| 21 | 1E   | FB | 7B |    |        | BDRR, R3 | AGAIN           | No; Move another byte          |
| 22 | 20   | BO |    |    |        | WRTC     | R0              | Yes! EXIT to MONITOR           |

## EXERCISE 22

## INDIRECT ROUTINE RELOCATION

In which the BYTE TRANSFER COUNT, SOURCE START ADDRESS, and DESTINATION ENDING ADDRESS are User-located in 6 contiguous bytes of memory. Program calculates SRCLAD-1 and DESTLAD-1; then executes the data transfer indirectly, using the resultant addresses.

ADVANTAGE: Having loaded the program, the user need only parametrize SRCLAD, DESTUAD, and BYTECT, then execute to effect the transfer.

LIMITATIONS: Operation of the program is page boundary-limited. Source start address may not be H'0000'. BYTECT  $\leq$  256

NOTE: The format of this routine is particularly adaptable to storage of the routine's control program within ROM, and its VARIABLES in other RAM defined memory locations. This program's application need not be restricted to routine relocation; in concept it may be implemented to move large segments of data from one area of memory to another.

PROCEDURE:

1. Load the program "INREL" into memory at location '1796'. (See next page)
2. You are to transfer 70<sub>10</sub> bytes of data from memory, starting at location '30', to memory in which the ending address (H'0115') is known. Code and load the necessary variables into memory from location '178C'.
3. Execute the program "INREL". Verify the transfer of data, using DESTLAD at location '1794' as a pointer.
4. Execute program "INCPAT2" (location '1780') then repeat steps 1 through 3. In step 2, select and code variables for any desired data transfer.

## SUGGESTED SOLUTION

## ROUTINE 'INREL'

VARIABLES  
STORE IN  
SCRATCHPAD  
RAM

PROGRAM DE-  
SIGNER FOR  
STORAGE IN  
READ ONLY  
MEMORY

| ADDRS | DATA |    |       | LABEL   | SYMBOLIC INSTRUCTION |                 | COMMENT                      |
|-------|------|----|-------|---------|----------------------|-----------------|------------------------------|
|       | B2   | B1 | B0    |         | OPCODE               | OPERANDS        |                              |
| 1     |      |    |       |         |                      |                 |                              |
| 2     | 178C |    |       | SRCLAD  | RES                  | 2               |                              |
| 3     | E    |    |       | DESTUAD | RES                  | 2               |                              |
| 4     | 1790 |    |       | BYTECT  | RES                  | 2               |                              |
| 5     | 2    | XX | XX    | CALSRC  | RES                  | 2               |                              |
| 6     | 4    | XX | XX    | DESTLAD | RES                  | 2               |                              |
| 7     | 1796 | 77 | 08    | INREL   | PPSL                 | WC              | Program With Carry status.   |
| 8     | 8    | 08 | 75    |         | LODR, RO             | > DESTUAD       | Subtract LS Byte Ct from     |
| 9     | A    | AB | 75    |         | SUBR, RO             | > BYTECT        | LS DEST. Upper addr. Result  |
| 10    | C    | CB | 77    |         | STAR, RO             | > DESTLAD       | to LS Dest lower addr.       |
| 11    | E    | 08 | 6E    |         | LODR, RO             | < DESTUAD       | Now Subtract MS BYTECT       |
| 12    | 17AD | AB | 6E    |         | SUBR, RO             | < BYTECT        | from MS DEST upper addr.     |
| 13    | 2    | CB | 70    |         | STAR, RO             | < DESTLAD       | Result to MS DEST LAD        |
| 14    | 4    | 09 | 66    |         | LODR, R1             | < SRCLAD        | Now, get both bytes of       |
| 15    | 6    | 08 | 65    |         | LODR, RO             | > SRCLAD        | SRC Lower address. If LS     |
| 16    | 8    | 58 | 02    |         | BRNR, RO             | \$ + 4          | Byte of SRCLAD = 0, subtract |
| 17    | A    | A5 | 01    |         | SUBI, R1             | 1               | 1 from MS Byte, otherwise    |
| 18    | C    | A4 | 01    |         | SUBI, RO             | 1               | leave alone. Now, subtract   |
| 19    | E    | C8 | 63    |         | STRR, RO             | > CALSRC        | 1 from LS Byte and store     |
| 20    | 17B0 | C9 | 60    |         | STRR, R1             | < CALSRC        | away in Calculated Src addr. |
| 21    | 2    | 0B | 5D    |         | LODR, R3             | > BYTECT        | Now, fetch byte count and    |
| 22    | 4    | 05 | 00    |         | LODI, R1             | 0               | initialize the index, then   |
| 23    | 6    | 0D | B7 92 | AGAIN   | LODA, RO             | * CALSRC, R1, + | transfer from source to      |
| 24    | 8    | CD | F7 94 |         | STRA, RO             | * DESTAD, R1    | destination, 2 byte rotation |
| 25    | C    | FB | 78    |         | BDRR, R3             | AGAIN           | When finished total trans-   |
| 26    | 17BE | B0 |       |         | WRTC                 | RO              | fer, exit to monitor.        |
| 27    |      |    |       |         |                      |                 |                              |

## SYMBOLIC CONSTANT SPECIFICATIONS:

SRCLAD = SOURCE DATA START (Lower) address

DESTUAD = DESTINATION DATA ENDING (Upper) address

> = LEAST SIGNIFICANT BYTE OF SPECIFIED 2-BYTE FIELD

< = MDST

CALSRC = Calculated SRCLAD-1 address {must be in place prior to data

DESTAD = Calculated DESTLAD-1 address {transfer via indirect address

DIRECTION:

Go right on to the next page.

MISCELLANEOUS INSTRUCTIONS:

NOP - CODE = H'CO'

HALT - CODE = H'40'INTRDDUCTION:NOP

The NOP (NO OPERATION) instruction may be implemented effectively in the following applications:

1. As a "filler" instruction when you are preparing a previously untested functional instruction sequence.

NOTE 1: THE 'SPACE' THUS CREATED IN THE 'INSTRUCTOR'S' MEMORY MAY BE USED FOR ADDITIONAL INSTRUCTIONS WITHOUT GOING THROUGH THE TEDIOUS PROCESS OF RELOADING ALL OR PART OF THE ROUTINE INTO MEMORY. LATER, UNUSED NOPS MAY BE CONDENSED OUT OF THE FINISHED ROUTINE IN A 2ND REWRITE IF DESIRED.

NOTE 2: THE NOP MAY BE INSERTED IN SPECIFIC LOCATIONS OF A PROGRAM WHERE (1) YOU FEEL IT WILL BE ABSOLUTELY NECESSARY TO BREAKPOINT AND INSPECT REGISTER OR MEMORY CONTENTS BOTH BEFORE AND AFTER EXECUTION OF A SPECIFIC INSTRUCTION AND (2) THE STRUCTURE OF THE PROGRAM WOULD OTHERWISE PROHIBIT SETTING OF A VALID BREAKPOINT.

EXAMPLE: DURING DEBUG, YDU KNOW THAT YOU'LL WANT TO EXAMINE REGISTER CONTENTS UPON RETURN FROM A USER-AVAILABLE MONITOR ROUTINE (E.G., USRDSP;GNP;) WHEN COMPOSING THE PROGRAM, INSERT THE NOP FOLLOWING THE APPROPRIATE ZBSR (TO USER-AVAIL. ROUTINE) DURING DEBUG, BREAK-POINT ON THE NOP. WHEN YOU'VE PROVED WHAT YOU DESIRED, ELIMINATE THE NOP IN THE FINAL REWRITE OF THE ROUTINE.

2. As a "fine adjustment" to a critical programmed delay in a working program.

NOTE: EXECUTION TIME OF THE NOP INSTRUCTION IS 2 MACHINE CYCLES OR 6 MICROPROCESSOR CLOCK CYCLES. WITH A 1 MHZ CLOCK, A DELAY OR TIME OUT CAN BE INCREASED BY 6  $\mu$ SEC PER ADDITIONAL NOP.

3. During DEBUG:

- a. The NOP is used to replace instruction data previously written, then found to be unnecessary to the routines functional operation.

EXAMPLE: AS PART OF THE SETUP OF A ROUTINE, A PROGRAMMER INITIALIZES R1 TO 0. DURING DEBUG, HE FINDS THAT R1 IS ALWAYS 0 AS PROGRAM CONTROL IS TRANSFERRED TO THAT ROUTINE. THE INITIALIZING INSTRUCTION MAY BE REPLACED WITH NOPS WHICH ARE LATER CONDENSED OUT OF THE ROUTINE WHEN IT IS REWRITTEN.

HALT

Upon detecting the HALT instruction, the microprocessor stops executing instructions and enters the WAIT state. Concurrently the RUN/WAIT line from the microprocessor is driven to a logical '0' state, thus signaling external microprocessor-controlled hardware that program execution is in the WAIT mode. The "INSTRUCTOR's" RUN LED is turned off.

There are 2 ways for the microprocessor to re-enter the RUN state:

1. RESET the microprocessor.

NOTE: When RESET is asserted to the MICROPROCESSOR, program execution is initiated at memory address 0. Refer to the 2650 Reference Manual for a detailed description of RESET.

2. INTERRUPT the microprocessor.

NOTE: The subject of Interrupt operations as sequenced by the 2650 is described in detail in Module V-E of this course. A short exercise following this introduction employs an interrupt sequence to demonstrate the activity of the microprocessor following execution of the HALT instruction.

Although the function of HALT is quite simple, its usage can be applied in a number of variations:

1. Certain microprocessor-controlled applications require a program in which the microprocessor services the controlled device with a precise sequence. Having serviced the device, its operation may be halted until that device responds, via an interrupt request. The interrupt routine may be as simple as to return program control to the instruction following the HALT, in order to execute another segment of the total program. Or this routine may be very complex, interacting with the controlled device in entirely different functions as compared to the interactive sequence. Once the interrupt service routine is terminated program control returns to the interrupted sequence which itself may be terminated with a HALT instruction.
2. Many applications require a program that specifies a definite completion of functional processing. In cases such as this, use of the HALT instruction, coupled with external assertion of RESET to the processor, may be the easiest implemented solution.
3. In time-critical applications, the speed with which the processor responds to a device's interrupt request may spell the difference between smooth operation of the device as opposed to its damage or destruction. Consider the implication in programming a HALT instruction in a process as opposed to programming a BCTR,UN \$ instruction. Both effectively terminate further processing of the instruction sequence. However, when the BCTR,UN \$ is employed, its execution must be completed before the processor can respond to the device's interrupt request. This could take up to 9 clock periods (9 usec if processor clock = 1 Mhz). In comparison, interrupt processing is initiated immediately from a WAIT state if an interrupt request is received by the processor.



EXERCISE 23PRACTICAL USE OF THE HALT INSTRUCTIONINTRODUCTION:

The main program segment of "HALT1" (below) consists of 3 passes. Within each pass, the parallel I/O LEDs are switched on in patterns ranging from H'00' to H'F0', in H'10' increments. After the third pass, the microprocessor goes into WAIT state. RESET can reinitialize the program. Upon detection of an external interrupt request (User **INT** Key), routine "INTRPT" is executed. "INTRPT" successively turns on each LED in sequence, starting with the most significant LED. After one pass, program sequence control is returned to the main program. If **INT** is depressed while the microprocessor is in the WAIT state, "HALT1" is reinitialized after a single pass through "INTRPT".

DIRECTION:

Code the program "HALT1" in the space provided below, then compare your code with the suggested solution on the next page.

**"HALT1"**NOTE:

If the space provided is too small, use a separate programming form.

| ADDRESSES | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |           | COMMENT                       |
|-----------|------|----|----|--------|----------------------|-----------|-------------------------------|
|           | BF   | BI | BS |        | OPCODE               | OPERANDS  |                               |
| 1         |      |    |    |        |                      |           |                               |
| 2         | 0000 |    |    | HALT1  | EORZ                 | R0        | On entry, initialize          |
| 3         |      |    |    |        | STRZ                 | R1        | the normal pattern.           |
| 4         |      |    |    |        | LPSL                 |           | also, clean all of            |
| 5         |      |    |    |        | LPSU                 |           | the PSW, then go around       |
| 6         |      |    |    |        | BCTA, UN             | CONT      | interrupt service to CONT     |
| 7         |      |    |    |        |                      |           |                               |
| 8         | 0007 |    |    | INTRPT | STRA, R0             | DOWNCT    | On entry, store current ct    |
| 9         | A    |    |    |        | PSSL                 | RS, WC    | down, then clean REG ports    |
| 10        | C    |    |    |        | CPSL                 | C         | and WC (Prime rel.) and clean |
| 11        | E    |    |    |        | LODI, R3             | H'80'     | Carry. Now set count int      |
| 12        | 10   |    |    | CTDN   | LODI, R0             | H'0C'     | pattern & delay constant.     |
| 13        | 2    |    |    |        | WRITE, R3            | PORT7     | Show current count on LEDs    |
| 14        | 4    |    |    |        | BDRR, R2             | \$        | then delay                    |
| 15        | 6    |    |    |        | BDRR, R0             | \$-2      | and                           |
| 16        | 8    |    |    |        | RRR                  | R3        | push count right. Is it       |
| 17        | 9    |    |    |        | BRNR, R3             | CTDN      | pushed out? No, push again    |
| 18        | 3    |    |    |        | CPSL                 | RS, WC, C | Yes, clean prime rel. and     |
| 19        | D    |    |    |        | LODA, R0             | DOWNCT    | Carry parameters, then        |
| 20        | 0020 |    |    |        | RETE, UN             |           | Fetch prev. stored ct. down   |
| 21        | 0100 | XX |    | DOWNCT |                      |           | and exit.                     |
| 22        |      |    |    |        |                      |           |                               |
| 23        | 002E |    |    | CONT   | LODI, R0             | 48        | Now, set count counter        |
| 24        | 30   |    |    |        | ADDI, R1             | H'10'     | and increment pattern         |
| 25        | 2    |    |    |        | WRITE, R1            | PORT7     | and display on LEDs           |
| 26        | 4    |    |    |        | LODI, R2             | H'60'     | Then set and run 1/4 sec      |
| 27        | 6    |    |    |        | BDRR, R3             | \$        | delay between pattern         |
| 28        | 8    |    |    |        | BDRR, R2             | \$-2      | changes. 48 changes           |
| 29        | A    |    |    |        | BDRR, R0             | CONT+2    | run yet? No, do another       |
| 30        | C    |    |    |        | HALT                 |           | Yes. Stop processing and      |
| 31        | 3D   |    |    |        | BCTA, UN             | HALT+1    | wait for interrupt. Once      |
| 32        |      |    |    |        |                      |           | turn from int., repeat prog.  |

PROCEDURE:

1. Load and execute "HALT1" on the "INSTRUCTOR".

NOTE: Prior to executing "HALT1," set the "INSTRUCTOR" toggle switches as follows:

I/O SELECTION: EXTENDED  
INTERRUPT: DIRECT

2. Respond to the following questions, using the "INSTRUCTOR's" facilities to verify your responses. After completing each requirement, restore the program to its original content.

- a) What is the location and code necessary to increase the number of passes through the main program from 3 to 7?
- b) What would be the effect (on program operation) of employing a HALT instruction within the "INTRPT" service routine?
- c) What would be the effect of replacing the HALT instruction at location '3C' with a NOP?
- d) What would be the effect of inserting a HALT instruction BETWEEN the WRTE and LODI instructions (locations '32'-'35')?

NOTE: Relocate the final instructions of routine "CONT" ONE location forward. BDRR,RO CONT+2 code = 'F8' '73'

- e) "It is desired to exercise the "INTRPT" routine ONCE immediately after a RESET is input to the microprocessor." Use a separate form to program changes you would recommend to accomplish this.
- f) What would be the effect of an UNCONDITIONAL BRANCH to CONT (instead of HALT+1) at location '3D'?
- g) What will be the result if the Carry bit is not cleared by the instruction at location '0C'?
- h) "A SINGLE BYTE can be changed to INCREASE the duration of the "INTRPT" routine's single pass." What is the location of this byte?
- i) What is the address of the next instruction following execution of the HALT instruction?

NOTE: Use STEP mode. Do not depress the RST key.

SUGGESTED SOLUTION:  
'HALT1' CODE

| ADDRS | B0   | B1 | B2 | LABEL |
|-------|------|----|----|-------|
| 1     |      |    |    |       |
| 2     | 0000 | 20 |    | HALT  |
| 3     |      | C1 |    |       |
| 4     |      | 93 |    |       |
| 5     |      | 92 |    |       |
| 6     |      | 1F | 00 | 2E    |
| 7     |      |    |    |       |
| 8     | 0007 | CC | 01 | 00    |
| 9     |      | A7 | 1B |       |
| 10    |      | C7 | 50 |       |
| 11    |      | E0 | 7B |       |
| 12    |      | 10 | 04 | 0C    |
| 13    |      | D7 | 07 |       |
| 14    |      | FA | 7E |       |
| 15    |      | E8 | 7C |       |
| 16    |      | 53 |    |       |
| 17    |      | 5B | 75 |       |
| 18    |      | 75 | 19 |       |
| 19    |      | 0C | 01 | 00    |
| 20    | 0020 | 37 |    |       |
| 21    | 0100 | XX |    |       |
| 22    |      |    |    |       |
| 23    | 002E | 04 | 30 |       |
| 24    |      | 30 | 85 | 10    |
| 25    |      | 2  | D5 | 07    |
| 26    |      |    | 06 | 60    |
| 27    |      |    |    |       |
| 28    |      | F3 | 7E |       |
| 29    |      | FA | 7C |       |
| 30    |      | FB | 74 |       |
| 31    |      | C  | 40 |       |
| 32    | 3D   | 1F | 00 | 01    |
| 33    |      |    |    |       |
| 34    |      |    |    |       |
| 35    |      |    |    |       |
| 36    |      |    |    |       |
| 37    |      |    |    |       |
| 38    |      |    |    |       |
| 39    |      |    |    |       |
| 40    |      |    |    |       |
| 41    |      |    |    |       |
| 42    |      |    |    |       |
| 43    |      |    |    |       |
| 44    |      |    |    |       |
| 45    |      |    |    |       |
| 46    |      |    |    |       |
| 47    |      |    |    |       |
| 48    |      |    |    |       |
| 49    |      |    |    |       |
| 50    |      |    |    |       |
| 51    |      |    |    |       |
| 52    |      |    |    |       |
| 53    |      |    |    |       |
| 54    |      |    |    |       |
| 55    |      |    |    |       |
| 56    |      |    |    |       |
| 57    |      |    |    |       |
| 58    |      |    |    |       |
| 59    |      |    |    |       |
| 60    |      |    |    |       |
| 61    |      |    |    |       |
| 62    |      |    |    |       |
| 63    |      |    |    |       |
| 64    |      |    |    |       |
| 65    |      |    |    |       |
| 66    |      |    |    |       |
| 67    |      |    |    |       |
| 68    |      |    |    |       |
| 69    |      |    |    |       |
| 70    |      |    |    |       |
| 71    |      |    |    |       |
| 72    |      |    |    |       |
| 73    |      |    |    |       |
| 74    |      |    |    |       |
| 75    |      |    |    |       |
| 76    |      |    |    |       |
| 77    |      |    |    |       |
| 78    |      |    |    |       |
| 79    |      |    |    |       |
| 80    |      |    |    |       |
| 81    |      |    |    |       |
| 82    |      |    |    |       |
| 83    |      |    |    |       |
| 84    |      |    |    |       |
| 85    |      |    |    |       |
| 86    |      |    |    |       |
| 87    |      |    |    |       |
| 88    |      |    |    |       |
| 89    |      |    |    |       |
| 90    |      |    |    |       |
| 91    |      |    |    |       |
| 92    |      |    |    |       |
| 93    |      |    |    |       |
| 94    |      |    |    |       |
| 95    |      |    |    |       |
| 96    |      |    |    |       |
| 97    |      |    |    |       |
| 98    |      |    |    |       |
| 99    |      |    |    |       |
| 100   |      |    |    |       |

DIRECTION:

Compare your response with the suggested answers provided on the next page.

ANSWERS TO QUESTIONS ON PREVIOUS PAGE - STEP 2

- a) loc. '2E': '04'70' (LODI,R0 112) ... each "pass" actually consists of 16 events.  $7 \times 16 = 112_{16} = H'70'$
- b) • Briefly; very strange things: Try it at location '0C'.  
" " " " '18'.  
" " " " '1B'.
- If HALT at location '1B' (followed by a NOP), note the effect of depressing RESET once: ..... and twice. One fact is certain, there is no way of exiting from "INTRPT" (once the microprocessor accesses it) unless RST is depressed.
- Insert a HALT between the LODA and RETE instructions. Breakpoint to the HALT, depress STEP RST. Then depress STEP sequentially. NOTE the best procedure to execute in STEP mode.
- c) The main program will run "forever". After required number of passes, program execution loops back to HALT+1.
- d) Each count in the main program is interleaved with execution of the "INTRPT" routine. Depressing RST only permits program execution to the first count in the main program. INT must be depressed to advance the count.

**NOTE:** At the very least, insertion of a HALT instruction anywhere in the program must be carefully considered, and serious diagnostic debugging performed, in order to ensure desired program sequence control.

- e) The process involves relocation of a few existing instructions and addition of a few subroutine linking instructions. Consider your changes with respect to the following known facts:
  - 1) Routine INTRPT must remain programmed as a subroutine, concluded with an RETE instruction.
  - 2) Input of RESET always resets the Program Counter to '00'.

One approach is illustrated on the next page.
- f) No problem! The REGISTERS and PSW cleared by the instructions at locations '00' to '03' are zero when inspected after asserting a BREAKPOINT at location '3D'.
- g) There can be a problem. Consider routine "CONT" in which H'10' is added successively to R1. When R1 = F0 (at the end of each pass), addition of H'10' causes Carry to be set. If an INTERRUPT REQUEST is received before the next addition of H'10', the Carry bit is itself "carried into the INTRPT routine processing, moving into R3 when the RRL,R3 instruction is executed: The following instruction, BRNR, detecting an active bit in R3 at all times, causes program sequence to loop forever between itself and 'CTDN'.
- h) Address H'11'.
- i) Address H'07' (BREAKPOINT to loc '3C', depress STEP and INT.)

One approach to program change required to satisfy conditions specified in question E.

|    |      |    |    |                                                                                                          |          |        |
|----|------|----|----|----------------------------------------------------------------------------------------------------------|----------|--------|
|    |      |    |    | HILL INTERRUPT REQUEST DURING INITIAL PASS OF INTRPT AND CLEARING. So set the                            |          |        |
|    |      |    |    | II Bit and exit to RELOC                                                                                 |          |        |
|    |      |    |    | On return, enable interrupt request and branch to                                                        |          |        |
|    |      |    |    | CONT.                                                                                                    |          |        |
| 4  | 0000 | 76 | 20 | HALT1                                                                                                    | PPSU     | II     |
| 5  | 2    | BB | 21 |                                                                                                          | ZBSR     | RELOC  |
| 6  | 4    | 92 |    |                                                                                                          | LPSU     |        |
| 7  | 5    | 1B | XX |                                                                                                          | BCTR, UN | CONT   |
| 8  |      |    |    |                                                                                                          |          |        |
| 9  | 0021 | BB | 07 | RELOC                                                                                                    | ZBSR     | INTRPT |
| 10 | 3    | 20 |    |                                                                                                          | EORZ     | RO     |
| 11 | 4    | C1 |    |                                                                                                          | STRZ     | R1     |
| 12 | 5    | 93 |    |                                                                                                          | LPSL     |        |
| 13 | 6    | 17 |    |                                                                                                          | RETC, UN |        |
|    |      |    |    | On entry, execute routine INTRPT. On return, initialize regs and clear PSL then, return to main program. |          |        |

## DIRECTIONS:

1. This concludes the presentation of material in Modules IV-F and IV-G. Set aside the course materials for these sections. Access Module IV-H, "PROGRAM STATUS WORD", open to page 1.
2. Also, Fast forward this tape to the end of Side B and store it in the album. Then access tape 8 and install it Side A up, ready for playback in the next module.

# Signetics

## 2650 MICROPROCESSOR COURSE

### MODULE IV - H

#### PROGRAM STATUS WORD ( PSW )

#### PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT  
Signetics Corporation  
811 E. Arques Ave.  
Sunnyvale, CA, 94086

#### REFERENCE:

Outlines and Abstracts  
page 21

## PART 1

## OVERVIEW

INTRODUCTION:

At this point in the course, there is a major feature of the 2650 which must be given formal recognition; this is the PROGRAM STATUS WORD. Simply, the PSW is a 16-bit special purpose register which contains both STATUS and CONTROL information. Further, it is fully programmable. When manipulated by a skillful programmer, its usage adds a powerful degree of flexibility in processing power and program control. Put another way, it would be difficult not to consider the PSW in all but the simplest of programs.

The PSW register is divided into 2 bytes; designated PROGRAM STATUS - UPPER (PSU) and PROGRAM STATUS-LOWER (PSL). Manual access to the PSU or PSL via facilities of the INSTRUCTOR is easy. Simply select the MONITOR's Display and Alter Registers mode, and key a '7' (for PSU) or an '8' (for PSL).

There are 10 instructions available to TEST, PRESET, CLEAR, STORE, (to register 0), or LOAD (from register 0) specific bits contained in the PSW. These are defined in the following table:

|     |         |          |                                                                              |
|-----|---------|----------|------------------------------------------------------------------------------|
| LPS | {U<br>L | 92<br>93 | Load Program Status, Upper }<br>Load Program Status, Lower } From R0         |
| SPS | {U<br>L | 12<br>13 | Store Program Status, Upper }<br>Store Program Status, Lower } To R0         |
| CPS | {U<br>L | 74<br>75 | Clear Program Status, Upper, Masked<br>Clear Program Status, Lower, Masked   |
| PPS | {U<br>L | 76<br>77 | Preset Program Status, Upper, Masked<br>Preset Program Status, Lower, Masked |
| TPS | {U<br>L | B4<br>B5 | Test Program Status, Upper, Masked<br>Test Program Status, Lower, Masked     |

The LOAD and STORE instructions involve 8-byte data transfers between the PSW and R0. The PRESET, TEST, and CLEAR instructions perform their respective functions on specific PSW bits. The bits are selected via a programmable mask contained in the second byte of these instructions. Unselected bits (masked off) are not affected.

The contents of the PSW are defined as diagrammed on the next page. While the PSW is well explained in the 2650 Reference Manual, there are many points involving its flexibility and application which must be raised; these are described as this module progresses.

**DIRECTION:** Select tape 8A entitled "Program Status Word". Listen to the commentary for details on the diagrams contained on the next page.

**NOTE:** For your convenience, the script is printed on 4 pages following the diagrams.

## PRINTED SCRIPT COPY FOR PART 1 :

Starting with number 1, the SENSE bit at the top of the page, let's describe Program Status Word content.

The SENSE bit always equals the logical state of the SENSE input line to the microprocessor. SENSE is not programmable. It is often used as the input function of a serial I/O interface. Before going further, we should distinguish Serial I/O from Parallel I/O. Simply, serial I/O involves data transfer on a sequential bit stream basis. Parallel I/O transfer permits simultaneous transfer of data on many lines. Both Serial and Parallel I/O, including hardware interface and programming considerations, are described in Module 5 of this course.

In applications involving distributed processing, the SENSE line is often activated to enable one of several microprocessors in the total computing network. Basically, the control program of each microprocessor may be written to test the status of the Sense line when it is not performing any other process function. The Sense line is a direct input to the microprocessor from the outside world. Sure, there are other inputs to the microprocessor, described generally as I/O Control lines and detailed in Module 5 of the course. But the Sense line operates on the PSU Sense bit without dependence on the condition of the other I/O control lines. Recognition of this fact provides you, the programmer, with added flexibility when implementing peripheral I/O interface schemes. Especially those in which asynchronous timing is a fact of life.

Now, let's consider the Flag Bit; PSU Bit 6. Like Sense, Flag is routed as an I/O Control line via the microprocessor's Read Write logic. Unlike SENSE, Flag is fully programmable. Further, the FLAG signal output from the microprocessor always reflects the latest programmed logical status of FLAG. This being the case, the FLAG bit is easily implemented as the OUTPUT function of a serial I/O interface like SENSE, however, FLAG may be raised or dropped independently of other I/O Control lines.

Once programmed, FLAG remains latched in its logical state unless reprogrammed. There are some distinct possibilities available here. For example, your application might require 50K bytes of memory storage. Recall that direct addressing only permits access to 32K of storage. There is nothing to prevent the use of FLAG as an additional address or memory enable line in order to permit access to one, or the other, of 2 32K memory banks.

Nothing prevents the use of FLAG and SENSE for serial I/O operations within a system involving both parallel and serial I/O interfaces. Instructions which implement serial and parallel I/O operations are fully independent of each other. ((pause))

Now, consider the Interrupt Inhibit bit; PSU Bit 5. The subject of Interrupt Handling by the microprocessor is described fully in the final module of this course. While the procedure is complex, its introduction is necessary at this point. Simply, an input signal to the microprocessor, defined as INTERRUPT REQUEST, may be raised by some outside source. When this takes place, the microprocessor examines the current status of Interrupt Inhibit, PSU Bit 5. If Interrupt Inhibit is active, the microprocessor ignores the interrupt request and continues its current program execution. If the inhibit bit is disabled, the microprocessor aborts current program execution in order to process the interrupt instead. As soon as the microprocessor aborts the original program, it sets the interrupt inhibit bit, thus preventing recognition of further interrupt requests until the interrupt inhibit bit is turned off. The interrupt inhibit bit may be controlled by execution of an appropriate instruction.

Direct reset is accomplished by the CPSU instruction. Conditional reset may be implemented via execution of the RETE - that's RETURN CONDITIONALLY FROM SUBROUTINE and ENABLE INTERRUPT - instruction. And it may be set active through execution of the PPSL instruction.

Note that processing of interrupt sequences is on an unscheduled basis. It is for this reason that a variety of instructions are available to the programmer for the purpose of inhibiting - or recognizing - interrupt requests.

Now, let's move on to the STACK POINTER, PSU Bits 2, 1, and 0. Again, an introduction is necessary. Each time a Branch to Subroutine instruction is executed, and the branch condition is honored, the microprocessor saves the next sequential instruction address in the Return Address Stack. The Stack Pointer, organized as a binary counter, is then incremented and Program Control is transferred to the first instruction in the subroutine. At the conclusion of the subroutine, the Program Counter reads up a return instruction. If the condition for the return is satisfied, the Stack Pointer is decremented and the previously saved address is routed via the output control to the Address Bus. Program sequence control is thus returned to the calling program. The current position of the Stack Pointer is reflected in the SP bits; PSU Bits 2, 1 and 0.

» The words "program sequence control" have been stressed deliberately in this discussion. Any time you manipulate the STACK POINTER, you should be aware that you are manipulating program sequence control. And, through implementation of appropriate PPSU, CPSU, or LPSU instructions, you can modify the STACK POINTER, and thus modify the program sequence. In a complex program with many possible paths, sequence errors are among the most difficult problems to debug.

For this reason, a subsection of this module explores the Stack Pointer via exercises involving planned programming under known conditions. We should recognize as well that programmed modification of the Stack Pointer can become a very powerful tool in the hands of the serious programmer, particularly when his application has many conditional - but TIME DEPENDENT sequences.

Now, let's move on to the condition code.

» The Condition Code is contained in the PSL, Bits 7 and 6. This 2-bit code is set whenever the contents of any general purpose register are loaded or modified. Within this framework, all data transfer and data manipulation, that is (arithmetic, logical, rotate), instructions affect the condition code. In addition, I/O READ and Program Status Store instructions set the condition code. Further, the CONDITION CODE is set to reflect the relative value of 2 bytes whenever a compare instruction is executed.

There is one common factor involved. Regardless of the instruction executed, the ALU must be involved (in data transfer, manipulation, or comparison) in order for the condition code to be set automatically.

The 2650 Reference Manual provides an excellent description of condition code status. Whenever data is stored into a general purpose register, the ALU determines the condition code in terms of the data's value as an 8-bit 2's complement number. Register Bit 7 is the sign bit. If set, the number is negative, that is (-1 to -128), and the condition code is set to one zero. If the register contents equal 0, the CC is set to 00. If the sign bit is off, and some number greater than 0 exists in the 7 remaining bits, the number is positive that is (+1 to +127) and the CC is set to 01. All data transfer and manipulation instructions previously mentioned cause conditions code setting as described above.



During Compare instruction execution, setting of the Condition Code is determined by two factors; these include:

- 1) The setting of the COMPARE control, PSL Bit 1.
- 2) The value of the relative comparison.

The relative comparison will be made in terms of signed arithmetic values if the CDM bit is reset, and in terms of 8-bit absolute numbers if the CDM bit is set to 1.

Now, consider condition code generation from a program sequence control viewpoint. Conditional branch decisions are basically determined by examination of one of 2 distinct function variables; one group of branch instructions evaluates register contents directly as a condition for program sequence control transfer. Examples include the BIR, BDR, and BRN instructions. Another group evaluates the condition code as a basis for program sequence control transfer. Examples include the BCT, BCF, and BST instructions. It follows that program sequence errors will be reduced when the programmer completely understands the condition code upon which his program's branch decision is based. Branch instruction execution does not affect the current condition code.

Thus, Condition Code Generation is determined by several variables. For this reason, a separate subsection of this module examines condition code generation and interpretation in a variety of situations.

Additionally, the condition code may be set directly through execution of an appropriate PPSL, CPSL, or LPSL instruction. Since the condition code is indicative of program status, this is rarely done. On occasion, however, it is a useful method of controlling program sequence destination directly. Of the four possible condition codes available, only three may be generated automatically by the ALU. The fourth condition code = 11, is never set by the ALU.

The ALU also controls the setting of the arithmetic computation status bits. These status bits include:

- 1) The Interdigit Carry - IDC in PSL Bit 5.
- 2) Overflow - OVF in PSL Bit 2 and
- 3) CARRY - C in PSL Bit 0.

In addition, program of the CONTROL bit With Carry - PSL Bit 3, determines generation and usage of the Carry, Interdigit Carry, and the Overflow bits during arithmetic and ROTATE instruction execution.

Through implementation of the INTERDIGIT CARRY bit - the 2650 gains a flexibility to process packed decimal data. In packed decimal format, two binary coded decimal digits, each four bits in width, comprise one byte of data. The IDC (PSL Bit 5) always reflects the carry or borrow out of bit 3 of the register designated in the previous add or subtract instruction.

OVERFLOW, PSL Bit 2, is set automatically during add or subtract operations if both initial operands have the same sign and if the result has a different sign. No OVF is generated if the initial arguments are signed differently.

Carry, PSL Bit 0, is set automatically during execution of an arithmetic instruction if a carry or borrow of the ALU's most significant bit takes place. Rotate instructions also affect the condition of Carry, IDC, and OVERFLOW when the control bit With Carry is set.

The 2650 Reference Manual contains detailed descriptions of the With Carry control bit and status bits Carry, Interdigit Carry, and Overflow. In addition, a separate subsection of this module provides structured exercises for effective implementation of these bits in several unique situations.

The Register Select Control - PSL Bit 4, determines which of 2 internal general purpose register banks are operative and subject to program modification. When RS is set, the prime bank, consisting of Registers R1', R2', and R3', is selected. When RS is reset, the non-prime registers, R1, R2, and R3, are selected. Register R0 is not subject to selection; it is always available to program modification.

**DIRECTION:** Program Status bits requiring further detail are described in the subsections which follow. Go right on to part 2 of this module.



## EXERCISE 1

## MODIFY PROGRAM STATUS - PART 1

INTRODUCTION:

In this exercise, you'll examine Condition Code generation during execution of program "LDSTRC". As you complete each step, familiarize yourself with the concept of SIGNED (+ or -) and UNSIGNED (8-bit absolute binary) numbers.

PROCEDURE:

1. Load program "LDSTRC" into User storage.
2. Execute in STEP mode. After depressing STEP each time, complete an entry in Table A (next page) by decoding the condition code in PSL bits 7 and 6. The suggested decode is provided on page 4.
3. Now, set the COM bit (PSL bit 1) active by modifying memory location '03' from '00' to '02'.

PROGRAM LDSTRC

| ADDRS  | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |          | COMMENT            |
|--------|------|----|----|--------|----------------------|----------|--------------------|
|        | B0   | B1 | B2 |        | OPCODE               | OPERANDS |                    |
| 1 0000 | 75   | FF |    | LDSTRC | CPSL                 | FF       |                    |
| 2 2    | 77   | 00 |    |        | PPSL                 | 0        | PSL leave zero     |
| 3 4    | 04   | 00 |    | LDR0   | LODI, R0             | 0        |                    |
| 4 6    | 05   | 01 |    |        | LODI, R1             | 1        |                    |
| 5 8    | 06   | 7F |    |        | LODI, R2             | H'7F'    |                    |
| 6 A    | 07   | 80 |    | LDR3   | LODI, R3             | H'80'    |                    |
| 7 C    | 04   | 81 |    |        | LODI, R0             | H'81'    |                    |
| 8 0E   | 04   | FF |    |        | LODI, R1             | H'FF'    |                    |
| 9 10   | 0B   | 74 |    |        | LODI, R0             | \$-10    | H'05' in loc. '06' |
| 10 2   | 03   |    |    |        | LODZ                 | R3       |                    |
| 11 3   | 0D   | 00 | 04 |        | LDA, R1              | LDR0     |                    |
| 12 6   | C2   |    |    |        | STRZ                 | R2       | R0 = H'80'         |
| 13 7   | CB   | 72 |    |        | STRR, R0             | \$-12    |                    |
| 14 9   | 20   |    |    | EORR0  | EORZ                 | R0       |                    |
| 15 A   | CA   | 6F |    |        | STRB, R2             | \$-15    |                    |
| 16 C   | C2   |    |    |        | STRZ                 | R2       |                    |
| 17 1D  | 0C   | 00 | 19 |        | LDA, R0              | EORR0    |                    |
| 18 20  | CF   | 00 | 0B |        | STRA, R3             | LDR3+1   |                    |

4. Repeat step 2, this time entering the condition in Table B on the next page.
5. Respond to the questions on the next page.

QUESTIONS -- STEP 5

- DIRECTION:** Compare your responses with those provided on the next page.

|   | CC1 | CC0 |
|---|-----|-----|
| 0 |     |     |
| 2 |     |     |
| 4 |     |     |
| 6 |     |     |
| 8 |     |     |
| A |     |     |
| C |     |     |
| E |     |     |
| 0 |     |     |
| 2 |     |     |
| 3 |     |     |
| 5 |     |     |
| 7 |     |     |
| 9 |     |     |
| A |     |     |
| C |     |     |
| 0 |     |     |
| 0 |     |     |

after execution of  
the instruction at  
address:

|    | CC1 | CC0 |
|----|-----|-----|
| 0  |     |     |
| 2  |     |     |
| 4  |     |     |
| 6  |     |     |
| 8  |     |     |
| A  |     |     |
| C  |     |     |
| E  |     |     |
| 10 |     |     |
| 12 |     |     |
| 13 |     |     |
| 16 |     |     |
| 17 |     |     |
| 19 |     |     |
| 1A |     |     |
| 1C |     |     |
| 1D |     |     |
| 20 |     |     |

| Bit 7 | Bit 6 |
|-------|-------|
| 0     | 0     |
| 0     | 1     |
| 1     | 0     |
| 1     | 1     |

SUGGESTED ENTRIES FOR  
TABLES A & B on the  
next page.



## EXERCISE 2 (CONTINUED)

These rules are interpreted with SLIGHT difference during execution of the COMPARE instruction. This discussion follows in Exercise 3.

- If an UNCONDITIONAL BRANCH is desired, the values are designated as follows:

| SYMBOLIC | CODE | COMMENT    |
|----------|------|------------|
| UN       | 3    | no CC test |

- The TEST code (0, 1, 2, or 3) determines the content of the 2 low-order opcode bits in related CC-testing instructions. Thus, if a "branch on condition true-greater than" is to take place, the OPCODE mnemonic in the symbolic instruction is written:

BCTR,GT the opcode is H'19'

PROCEDURE:

1. Enter data tables "OXNBR" and "XONBR" (below) into memory. After completion of each exercise, verify data integrity since certain data bytes may be changed during program execution.

DATA TABLES "OXNBR" AND "XONBR"

|    | ADDRS | DATA |    |    | LABEL | SYMBOLIC INSTRUCTION |          | COMMENT                 |
|----|-------|------|----|----|-------|----------------------|----------|-------------------------|
|    |       | B2   | B1 | B2 |       | OPCODE               | OPERANDS |                         |
| 1  | 0000  | 00   | 01 | 02 | OXNBR | RES                  | 16       | Data table of numbers   |
| 2  | 3     | 03   | 04 | 05 |       |                      |          | '00' through '0F' to be |
| 3  | 6     | 06   | 07 | 08 |       |                      |          | used in following pro-  |
| 4  | 9     | 09   | 0A | 0B |       |                      |          | grammed exercises.      |
| 5  | C     | 0C   | 0D | 0E |       |                      |          | Do NOT overlay          |
| 6  | 0F    | 0F   |    |    |       |                      |          |                         |
| 7  | 10    | 10   | 20 | 30 | XONBR | RES                  | 15       | Data table of numbers   |
| 8  | 3     | 40   | 50 | 60 |       |                      |          | '10' through 'FO. Do    |
| 9  | 6     | 70   | 80 | 90 |       |                      |          | NOT overlay.            |
| 10 | 9     | A0   | B0 | C0 |       |                      |          |                         |
| 11 | 1C    | D0   | E0 | F0 |       |                      |          |                         |

2. Code and load the program "ARITHCC" (next page) into memory. Compare your code with the suggested solution on page 7.

NOTE: This program is designed so that the 1st BCTR,NEG instruction (at loc. '33') is modified by the STR instruction at line 14. If you desire to re-run program "ARITHCC", restore locations '33' and '34' to original values.

3. Set the I/O Selection Switch to "EXTENDED" position, and execute in STEP mode from address '1F'.

**DIRECTION:** Go right on to the next page.

## EXERCISE 2 (CONTINUED)

## PROGRAM "ARITHCC"

| ADDRES | DATA |    |    | LABEL                                     | SYMBOLIC INSTRUCTION |          | COMMENT                  |
|--------|------|----|----|-------------------------------------------|----------------------|----------|--------------------------|
|        | B3   | B1 | B2 |                                           | OPCODE               | OPERANDS |                          |
| 1      |      |    |    | ARITHCC                                   | CPSL                 | H'FF'    | CLEAR PSL AND LEAVE      |
| 2      |      |    |    |                                           | PPSL                 | 0        | SPACE TO PRESET IT       |
| 3      |      |    |    |                                           | LODI, RO             | 1        | LATER, NOW GET 1 AND     |
| 4      |      |    |    |                                           | ADDR, RO             | 0XNBR+1  | ADD IT TO 1 (RELATIVE)=2 |
| 5      |      |    |    |                                           | SUBI, RO             | 2        | minus 2 = 0              |
| 6      |      |    |    |                                           | SUBR, RO             | 0XNBR    | minus 0 = 0              |
| 7      |      |    |    |                                           | SUBR, RO             | 0XNBR+1  | minus 1 = FF             |
| 8      |      |    |    |                                           | ADDR, RO             | 0XNBR+7  | plus '80' = 7F           |
| 9      |      |    |    |                                           | SUBI, RO             | 4        | minus 4 = 7B             |
| 10     |      |    |    | NEWSET                                    | ADDI, RO             | 1        | Now, increment RO by 1   |
| 11     |      |    |    |                                           | BCTR, NEG            | \$+4     | until RO=0, then branch  |
| 12     |      |    |    |                                           | BCTR, UN             | \$-4     | past return to increment |
| 13     |      |    |    |                                           | LODI, RO             | H'19'    | to set up new instruc-   |
| 14     |      |    |    |                                           | STRI, RO             | \$-6     | tion at location '0033'  |
| 15     |      |    |    |                                           | LODR, RI             | 0XNBR+14 | (TOUGH WAY TO GET A      |
| 16     |      |    |    |                                           | STRI, RI             | NEWSET+3 | '0E'!).                  |
| 17     |      |    |    |                                           | LODI, RO             | H'FC'    | When new instruction     |
| 18     |      |    |    |                                           | BCTR, NEG            | NEWSET   | set up is finished, get  |
| 19     |      |    |    | LIGHT                                     | WRITE, RO            | PORT7    | H'FC' (= -4) and BRANCH  |
| 20     |      |    |    |                                           | BCTA, UN             | MONITOR  | back to NEWSET. When     |
| 21     |      |    |    | Finished                                  |                      |          |                          |
|        |      |    |    | # Display RO on LEDs and exit to Monitor. |                      |          |                          |

4. After each STEP, complete an entry in TABLE C as indicated:

| AFTER EXECUTING THE INSTRUCTION AT LOCATION |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 25                                          | 27 | 29 | 2B | 2D | 2F | 31 | 33 | 35 | 31 | 33 | 35 | 31 | 33 | 35 | 31 | 33 | 35 | 31 | 33 |
| 37                                          |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| PSL =                                       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| RO =                                        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| CC1 =                                       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| CC0 =                                       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| Loc.                                        | 39 | 3B | 3D | 3F | 41 | 43 | 45 | 47 | 49 | 4B | 4D | 4F | 51 | 53 | 55 | 57 | 59 | 5B | 5D |
| PSL =                                       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| RO =                                        |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| CC1 =                                       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| CC0 =                                       |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

5. After completing Table C, compare your answers with those provided on the next page.

**DIRECTION:** Go right on to the next page.





## EXERCISE 3 (CONTINUED)

PROCEDURE:

1. Code and load the program "COMPAR" (below) into memory at location '100'. Suggested code is on the next page.

NOTE: Program "COMPAR" illustrates the activity of the CC for both Arithmetic (COM=0) and Logical (COM=1) compares. Set the I/O Selection Switch to "EXTENDED".

## PROGRAM "COMPAR"

| ADDRS | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |               | COMMENT                    |
|-------|------|----|----|--------|----------------------|---------------|----------------------------|
|       | B7   | B1 | B2 |        | OPCODE               | OPERANDS      |                            |
| 1     |      |    |    | COMPAR | CPSL                 | COM           | CLEAR COM FOR ARITH        |
| 2     |      |    |    | REPEAT | LODI, R2             | 9             | COMPARE, THEN SET 'AGAIN'  |
| 3     |      |    |    |        | LODI, R1             | 0             | CTR AND DATA INDEX AND     |
| 4     |      |    |    | AGAIN  | LODI, R0             | H 'XX'        | INITIALIZE R0 AS REFERENCE |
| 5     |      |    |    |        | COMA, R0             | DATA-1, R1, + | SEE TABLES BELOW. NOW      |
| 6     |      |    |    |        | SPSL                 |               | COMPARE MEMORY DATA        |
| 7     |      |    |    |        | WRTS, R0             | PORT 7        | TO R0 AND SAVE RESULTING   |
| 8     |      |    |    |        | BDRR, R2             | AGAIN         | CC. DISPLAY THE CC ON      |
| 9     |      |    |    |        | PPSL                 | COM           | PARALLEL I/O REPEAT        |
| 10    |      |    |    |        | RCTR, UN             | COMPAR        | UNTIL ALL NUMBERS COMPARED |
| 11    | 14   | 00 | 01 | DATA   | RES                  | 9             | THEN TO THE SAME IN LOG    |
| 12    | 7    | 7F | 80 |        |                      |               | ICAL COMPARE SELECTED      |
| 13    | 1A   | A0 | B0 |        |                      |               | DATA, TO PROVIDE COM-      |
| 14    |      |    |    |        |                      |               | PARISON TO R0 IN ALL       |
| 15    |      |    |    |        |                      |               | MODES & VARIATIONS.        |

2. Load location '107' with the "R0 COMPARE CONSTANT" as indicated in Table D on the next page.
3. Set a BREAKPOINT at location '10C'.
4. Set the PROGRAM COUNTER to location '100'.
5. Depress **RUN**. After executing "COMPAR" to the BKPT, inspect the PSL and R1 contents to determine each table location for Condition Code entry.

NOTE: For your convenience, the PSL is displayed on the I/O LEDs. Inspection of the PSL via Monitor facilities is not necessary.

6. Repeat step 5 until BOTH the ARITH and COM columns are completed for each "R0=0" compare constant.
7. Repeat step 2, this time entering the second compare constant in Table D into location '107'.

**DIRECTION:** Continue with step 8 on the next page.

## EXERCISE 3 (CONTINUED)

TABLE D ARITHMETIC / LOGICAL COMPARES

| (R1) | DATA | R0 = 0 |     | R0 = 7F |     | R0 = 80 |     | R0 = 81 |     | R0 = FF |     |
|------|------|--------|-----|---------|-----|---------|-----|---------|-----|---------|-----|
|      |      | ARITH  | LOG | ARITH   | LOG | ARITH   | LOG | ARITH   | LOG | ARITH   | LOG |
| 1    | 00   |        |     |         |     |         |     |         |     |         |     |
| 2    | 01   |        |     |         |     |         |     |         |     |         |     |
| 3    | 70   |        |     |         |     |         |     |         |     |         |     |
| 4    | 7F   |        |     |         |     |         |     |         |     |         |     |
| 5    | 80   |        |     |         |     |         |     |         |     |         |     |
| 6    | 90   |        |     |         |     |         |     |         |     |         |     |
| 7    | A0   |        |     |         |     |         |     |         |     |         |     |
| 8    | B0   |        |     |         |     |         |     |         |     |         |     |
| 9    | FF   |        |     |         |     |         |     |         |     |         |     |

## CONDITION CODE AFTER COMPARISON

00 = R0 and Mem are equal

01 = R0 &gt; or more positive than Mem

10 = R0 &lt; or more negative than Mem

8. Repeat steps 3, 5, and 6 until the second pair of columns are completed.
9. Repeat steps 7 and 8 until Table D is fully completed.
10. Compare the observations taken in Table D to those provided on the next page.
11. Listen to tape 8A for a short commentary and further directions.

## SUGGESTED CODE

## PRDGRAM "COMPAR"

| ADDRS | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |               | COMMENT                    |
|-------|------|----|----|--------|----------------------|---------------|----------------------------|
|       | EX   | B1 | B2 |        | OPCODE               | OPERANDS      |                            |
| 1     | 0100 | 75 | 02 | COMPAR | CPSL                 | COM           | CLEAR COM FOR ARITH        |
| 2     | 2    | 06 | 09 | REPEAT | LODI, R2             | 9             | COMPARE, THEN SET "AGAIN"  |
| 3     | 4    | 05 | 00 |        | LODI, R1             | 0             | CTR AND DATA INDEX AND     |
| 4     | 6    | 04 | XX | AGAIN  | LODI, R0             | H 'XX'        | INITIALIZE R0 AS REFERENCE |
| 5     | 8    | ED | 21 |        | COMA, R0             | DATA-1, R1, + | SEE TABLES BELOW. NOW      |
| 6     | B    | 13 |    |        | SPSL                 |               | COMPARE MEMORY DATA        |
| 7     | C    | D4 | 07 |        | WRITE, R0            | PORT7         | TO R0 AND SAVE RESULTING   |
| 8     | 0E   | FA | 76 |        | BDRR, R2             | AGAIN         | CC. DISPLAY THE CC ON      |
| 9     | 10   | 77 | 02 |        | PPSL                 | COM           | PARALLEL I/O. REPEAT       |
| 10    | 2    | 1B | 6E |        | BCTR, UN             | COMPAR        | UNTIL ALL NUMBERS COMPARED |
| 11    | 14   | 00 | 01 | DATA   | RES                  | 9             | THEN DO THE SAME IN LOG    |
| 12    | 7    | 7F | 80 |        |                      |               | ICAL COMPARE SELECTED      |
| 13    | 1A   | A0 | B0 |        |                      |               | DATA, TO PROVIDE COM-      |
| 14    |      |    |    |        |                      |               | PARISON TO R0 IN ALL       |
| 15    |      |    |    |        |                      |               | MODES & VARIATIONS.        |

**DIRECTION:** Go right on to the next page

## EXERCISE 3 (CONTINUED)

COMPLETED ENTRIES  
TABLE D

## ARITHMETIC / LOGICAL COMPARES

| (R1) | DATA | RO = 0 |     | RO = 7F |     | RO = 80 |     | RO = 81 |     | RO = FF |     |
|------|------|--------|-----|---------|-----|---------|-----|---------|-----|---------|-----|
|      |      | ARITH  | LOG | ARITH   | LOG | ARITH   | LOG | ARITH   | LOG | ARITH   | LOG |
| 1    | 00   | 00     | 00  | 01      | 01  | 10      | 01  | 10      | 01  | 10      | 01  |
| 2    | 01   | 10     | 10  | 01      | 01  | 10      | 01  | 10      | 01  | 10      | 01  |
| 3    | 70   | 10     | 10  | 01      | 01  | 10      | 01  | 10      | 01  | 10      | 01  |
| 4    | 7F   | 10     | 10  | 00      | 00  | 10      | 01  | 10      | 01  | 10      | 01  |
| 5    | 80   | 01     | 10  | 01      | 10  | 00      | 00  | 01      | 01  | 01      | 01  |
| 6    | 90   | 01     | 10  | 01      | 10  | 10      | 10  | 10      | 10  | 01      | 01  |
| 7    | A0   | 01     | 10  | 01      | 10  | 10      | 10  | 10      | 10  | 01      | 01  |
| 8    | B0   | 01     | 10  | 01      | 10  | 10      | 10  | 10      | 10  | 01      | 01  |
| 9    | FF   | 01     | 10  | 01      | 10  | 10      | 10  | 10      | 10  | 00      | 00  |
|      |      | 0      |     | 7F      |     | -128    |     | -127    |     | -1      |     |

**NOTES:** The discussion on tape references key entries in Table D by callout of the circled numbers.

**DIRECTION:** Go right on to the next page.

## EXERCISE 4

TEST PROGRAM STATUS - TPSU,TPSL  
- TMIINTRODUCTION:

There are 3 instructions which can be used to TEST the logical contents of the 2650's internal working registers. The RESULTS of the test determine the condition code status. TPSU and TPSL instructions test selected bits in the PSU and PSL respectively. The TMI instruction permits test of 1 or more selected bits in the specified register.

All TEST instructions are 2 bytes in nature; the second byte is programmed as a bit selection MASK. In all 3 cases, the condition code is set as follows:

00 if ALL selected bits are active (equal to 1)  
10 if any of the selected bits are  
inactive; e.g. equal to 0.

The COM bit (PSL bit 1) has no effect on execution of the TEST instructions.

REFERENCE: The 2650 Reference Manual provides an excellent description of the TEST instructions.

PROCEDURE:

1. Write and execute a routine to load memory locations '00' to 'FF' with incremented data. The START ADDRESS for this routine is at location '1780'.
2. Write, but do NOT execute, a subroutine at location '17A0' to CLEAR memory locations '100' through '1FF'. End this routine with a RETC,UN instruction.
3. Code and load program "TEST1" (below) into memory at location '1F'. Compare your solution to the one provided on page 13.

PROGRAM  
"TEST1"

| ADDRS | DATA<br>R2 R1 R0 | LABEL   | SYMBOLIC INSTRUCTION<br>OPCODE REFERENCES | COMMENT                     |
|-------|------------------|---------|-------------------------------------------|-----------------------------|
| 1     |                  | TEST1   | EORZ R0                                   | Initialize input byte index |
| 2     |                  |         | STRZ R1                                   | dex output byte index       |
| 3     |                  |         | STRZ R3                                   | also to clear PSL, then     |
| 4     |                  |         | LPSL                                      | set WC in PSL. Now load     |
| 5     |                  |         | FPSL WC                                   | the constant to be sub-     |
| 6     |                  |         | LODI,R2 'xx'                              | tracted. then go get        |
| 7     |                  | ANOTHER | LDA,R1 MEMO-1,R1,+                        | an input byte               |
| 8     |                  |         | COMI,R1 0                                 | Finished yet? Yes! go       |
| 9     |                  |         | BCTR,EQ FINISH                            | finish. No! Subtract the    |
| 10    |                  |         | SUBZ R2                                   | constant & see if OVF       |
| 11    |                  |         | TPSL OVF                                  | set? Not yet. Go get a-     |
| 12    |                  |         | BCFR,EQ ANOTHER                           | nother byte. OVF did set!   |
| 13    |                  |         | LDA,R0 MEMO-1,R1                          | Save the byte in upper      |
| 14    |                  |         | STRA,R0 GTRXX-1,R3,+                      | storage. (loc 100' and so)  |
| 15    |                  |         | BCTR,UN ANOTHER                           | and go get another byte     |
| 16    |                  | FINISH  | WRITE,R3 PORT7                            | Finished! Write the num     |
| 17    |                  |         | ZBSR *CLEAR                               | ber of bytes saved and      |
| 18    |                  |         | BCTR,UN TEST1                             | exit to clean memory.       |
| 19    |                  |         |                                           | On return, repeat TEST1     |

## EXERCISE 4 (CONTINUED)

4. Link the programs "TEST1" and "CLEAR" with an indirect address at locations '02' and '03'. The address is '17A0'.
5. At your option, write a BCTR,UN TEST1 instruction at loc. '00'.
6. Now, enter the first "R2 constant" from Table E into memory at location '26'. The first constant is '00'.
7. Listen to tape 8A for a brief description of program "TEST1".
8. Set a BREAKPOINT at location '3B' (FINISH) and RUN the program. Then complete the entries in the first column of Table E (below).

TABLE E

OBSERVATIONS TAKEN IN EXERCISE 4

|          | R2 CONSTANT                    | 00 | 03 | 20 | 7F | 82 | FC | FE |
|----------|--------------------------------|----|----|----|----|----|----|----|
| SUBTRACT | Number of saved bytes (in HEX) |    |    |    |    |    |    |    |
|          | Saved bytes L.A.D, e.g., '100' |    |    |    |    |    |    |    |
|          | Saved bytes U.A.D.             |    |    |    |    |    |    |    |
|          | Range of HEX Data e.g., E0-EF  |    |    |    |    |    |    |    |
| ADDRESS  | Number Saved                   |    |    |    |    |    |    |    |
|          | Lower ADDR                     |    |    |    |    |    |    |    |
|          | Upper ADDR                     |    |    |    |    |    |    |    |
|          | Range                          |    |    |    |    |    |    |    |

9. Repeat step 6 (for new R2 CONSTANT) and step 8 until the upper half of Table E is completed. Specified constants are in the table.
10. At location '2E', substitute an ADDZ,R2 for the SUBZ,R2 instruction.
11. Repeat steps 6, 8, and 9 and complete the lower half of Table E. Compare your results with those provided on the next page.
12. After comparison of your observations with those provided on the next page, listen to the tape for a discussion on this and other SORT routines.
13. You can automate this program in order to determine how many bytes are saved, given any variable loaded into R2 (at loc '25'). Just replace the LODI,R2 XX with an REDE,R2 PORT7 instruction. Then, depress RST and enter the constant via the PARALLEL I/O INPUT SWITCHES. For any constant entered, the parallel I/O LEDs will indicate, in binary, the number of saved bytes. It is a good demonstration of the concept that INPUT and OUTPUT functions of the SAME I/O port can be specified for totally different functions.

## EXERCISE 4 (CONTINUED)

NOTES:SUGGESTED SOLUTION"TEST1"

|    | ADDRS | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |               |
|----|-------|------|----|----|---------|----------------------|---------------|
|    |       | B2   | B1 | B0 |         | CRCODE               | OPERANDS      |
| 1  | 001F  | 20   |    |    | TEST1   | ECR2                 | R0            |
| 2  | 20    | C1   |    |    |         | STRZ                 | R1            |
| 3  | 1     | C3   |    |    |         | STRZ                 | R3            |
| 4  | 2     | 93   |    |    |         | LPSL                 |               |
| 5  | 3     | 77   | 0B |    |         | PSSL                 | WC            |
| 6  | 5     | 06   | XX |    |         | LODI, R2             | 'XX'          |
| 7  | 7     | 0D   | 3F | FF | ANOTHER | LDA, R1              | MEMO-1, R1, + |
| 8  | A     | E5   | 00 |    |         | COMI, R1             | 0             |
| 9  | C     | 18   | 0D |    |         | BCTR, EQ             | FINISH        |
| 10 | E     | A2   |    |    |         | SUBZ                 | R2            |
| 11 | 2F    | B5   | 04 |    |         | TDSL                 | OVF           |
| 12 | 31    | 98   | 74 |    |         | PCR, EQ              | ANOTHER       |
| 13 | 3     | 0D   | 7F | FF |         | LDA, R0              | MEMO-1, R1    |
| 14 | 6     | CF   | 20 | FF |         | STRA, R0             | GRAY-1, R3, + |
| 15 | 9     | 1B   | 6C |    |         | BCTR, IN             | ANOTHER       |
| 16 | B     | D7   | 07 |    | FINISH  | WRITE, R3            | PORT?         |
| 17 | D     | BB   | 82 |    |         | ZPSR                 | *CLEARM       |
| 18 | F     | 1B   | 5E |    |         | BCTR, IN             | TEST1         |

TYPICAL OBSERVATIONS TAKEN IN STEP 9 -- TABLE E

| R2 CONSTANT                          |                                | 00  | 03    | 20    | 7F    | 82    | FC    | FE  |
|--------------------------------------|--------------------------------|-----|-------|-------|-------|-------|-------|-----|
| S<br>U<br>E<br>T<br>R<br>A<br>C<br>T | Number of saved bytes (in HEX) | 0   | 4     | '24'  | 8E    | 6F    | 4     | 2   |
|                                      | Saved bytes L.A.D. e.g., '100' | 100 | 100   | 100   | 100   | 100   | 100   | 100 |
|                                      | Saved bytes U.A.D.             | —   | 103   | 123   | 18D   | 16E   | 103   | 101 |
|                                      | Range of HEX Data e.g., E0-EF  | —   | 80-82 | 80-9F | 80-FE | 04-7F | 7D-7F | 7F  |
| A<br>D<br>D<br>R                     | Number Saved                   | 0   | 4     | 24    | 71    | 8A    | 5     | 1   |
|                                      | Lower ADDR                     | 100 | 100   | 100   | 100   | 100   | 100   | 100 |
|                                      | Upper ADDR                     | —   | 103   | 123   | 170   | 189   | 104   | 102 |
|                                      | Range                          | —   | 7D-7F | 60-7F | 04-7F | 80-FC | 80-82 | 80  |

**DIRECTION:**

Go right on to Exercise 5 on the next page.

## EXERCISE 5

## TEST UNDER MASK IMMEDIATE - TMI

INTRODUCTION:

The TMI instruction is exceptionally useful in testing INPUT STATUS from external I/O devices, particularly when the condition of untested bits is immaterial. The TMI is very similar to the COMPARE instruction in its operation. However, it is not affected by the condition of the COM control bit (PSL bit 1). The condition code is set to one of the following as a result of TMI execution:

00 ....If all tested bits are at logic 1 levels  
 10 ....If any tested bit is at a logic 0 level

Since resultant CC definition is simplified, you may choose to invoke subsequent branches with greater certainty that your program will execute properly. Particularly if your program involves a number of ARITHMETIC sequences (COM = 0) in which a LOGICAL COMPARE may be necessary. In this case, you don't have to execute a PPSL COM instruction before the logical compare using the TMI instruction.

In the exercise which you'll be performing, memory locations '100' through '1FF' will be scanned for the presence of a distinct pattern of 'set' bits. When the specific pattern is detected, the 'index' of the pattern (its "address") is saved, starting at location '80'. Assuming that the saved addresses represent I/O addresses, the program could later execute routines to "service" each I/O "device" responding with the appropriate bit pattern. The pattern is selected via the Parallel I/O input switches operating in NON-EXTENDED (D Port) MODE. The NUMBER of saved addresses is posted to the 8-digit Hex display at the conclusion of each 'pass'. The tested pattern is displayed on the Parallel I/O LEDs.

PROCEDURE:

1. Load the final version of "CRAPGAME" into memory; locations '0' through '1FF' only.

NOTE: This step is necessary to provide a useful data base to be scanned.

2. Code and load the program "SCAN1" (next page) into memory. Compare your code with that provided on the following page.
3. Code and load a subroutine, "CLEARM", into memory at location '1780'. "CLEARM" is to clear '90' bytes in low-order storage, starting from location '70'.  
Link subroutine "CLEARM" to program "SCAN1" with the first instruction in "SCAN1", and with a RETC,UN instruction after execution of "CLEARM".
4. Listen to tape 8A for a brief commentary on program "SCAN1".

NOTES:

**DIRECTION:** The procedure steps continue on page 16.



## EXERCISE 5 (CONTINUED)

## PROGRAM "SCAN1"

| ADDRS | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |                | COMMENT                     |
|-------|------|----|----|---------|----------------------|----------------|-----------------------------|
|       | R0   | R1 | R2 |         | OPCODE               | OPERANDS       |                             |
| 1     | 0000 |    |    | SCAN1   | RSTA, UN             | CLEARM         | Clear Save index area (loc  |
| 2     |      |    |    |         | LPSL                 |                | 70-FF). On return R0=0      |
| 3     |      |    |    |         | STRZ                 | R1             | Then clear the PSL and set  |
| 4     |      |    |    |         | STRZ                 | R3             | up Ryle test and index      |
| 5     |      |    |    |         | REDD                 | R2             | save indices. Now, get de-  |
| 6     |      |    |    |         | STRB, R2             | TEST+1         | sired bit test pattern from |
| 7     |      |    |    | AGAIN   | LODA, R0             | HIMEM, R1      | I/O Port D and store it.    |
| 8     |      |    |    | TEST    | TML, R0              | XX             | Then, fetch a byte from     |
| 9     |      |    |    |         | ASTR, EA             | STRADDR        | Loc 100-1FF and see if      |
| 10    |      |    |    |         | BIRR, R1             | AGAIN          | it matches. Yes! Go store   |
| 11    |      |    |    | FINISH  | WRTD                 | R2             | the current index No. End   |
| 12    |      |    |    |         | LODZ                 | R3             | another "I" done. When      |
| 13    | BB   | F4 |    |         | ZBSR                 | *DISLSD        | done, write the test pat-   |
| 14    |      |    |    |         | STRB, R1             | TOTAL+7        | tern to Port D, then get    |
| 15    |      |    |    |         | STRB, R0             | TOTAL+6        | SUM of saved addresses      |
| 16    |      |    |    |         | EORZ                 | R0             | (p2). Break it into 2 NIBLS |
| 17    |      |    |    |         | STRZ                 | R1             | and store in the message    |
| 18    |      |    |    |         | STRZ                 | R3             | TOTAL=XX. Now, clear        |
| 19    |      |    |    |         | LODI, R2             | TOTAL-1        | R0, R1, and R3 and set      |
| 20    | BB   | E6 |    |         | ZBSR                 | *USRDSP        | up the LS Ryle of           |
| 21    |      |    |    |         | LODI, R0             | 17             | TOTAL+5 address MINUS 1     |
| 22    |      |    |    |         | STRB, R0             | TOTAL+6        | Then display the Saved      |
| 23    |      |    |    |         | STRB, R0             | TOTAL+7        | total. After inspecting,    |
| 24    |      |    |    |         | ZBRB                 | SCAN1          | hit a hex key. This exits   |
| 25    | 07   | 00 | 07 | TOTAL   | RES                  | R              | from message display &      |
| 19    | 0A   | 11 | 16 |         |                      |                | now you can put spaces      |
| 20    | 17   | 17 |    |         |                      |                | in message table and        |
| 21    |      |    |    |         |                      |                | loop back to repeat         |
| 22    | 01   |    |    | STRADDR | LODZ                 | R1             | DATA TABLE "total="         |
| 23    | CF   | 20 | 6F |         | STRA, R0             | INDEX-1, R3, + | The test said byte indexed  |
| 24    | 17   |    |    |         | RETC, UN             |                | by R1 matched, so save      |
| 25    |      |    |    |         |                      |                | indexes starting at loc 77  |
| 26    |      |    |    |         |                      |                | and exit to continue search |
| 27    |      |    |    |         |                      |                | for more matching bytes     |

NOTES:

## EXERCISE 5 (CONTINUED)

## EXERCISE 5 -- CONTINUATION OF PROCEDURE

5. Enter the TEST PATTERN (from Column 1 in Table A) via the Parallel I/O toggle switches. Ensure that the I/O Selection Sw. is in the NON-EXTENDED position.
6. Set a START ADDR to loc. '0'.  
Set a BREAKPOINT to loc. '27', and  
Depress RUN to execute.
7. Complete Column 1 in Table A.
8. Repeat steps 5 through 7 until the table is completed.

TABLE A:

| COLUMN                      | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  |
|-----------------------------|----|----|----|----|----|----|----|----|
| Pattern selected            | 00 | 01 | 02 | 04 | 10 | 20 | 40 | 80 |
| Number of Saved ADDR        | 0  |    |    |    |    |    |    |    |
| Address of Last Index Saved |    |    |    |    |    |    |    |    |
| COLUMN                      | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Pattern Selected            | 03 | C0 | C3 | D9 | 0F | E7 | F0 | 7F |
| Number of Saved ADDR        |    |    |    |    |    |    |    |    |
| Address of Last Index Saved |    |    |    |    |    |    |    |    |

9. Using the pattern 'C3' (Column 11), complete Table B. Then go on to step 10.

TABLE B

| COLUMN               | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   | 10  |
|----------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| Location in STRADDR  | 70  | 71  | 72  | 73  | 74  | 75  | 76  | 77  | 78  | 79  |
| ADDR of Indexed Byte | 1__ | 1__ | 1__ | 1__ | 1__ | 1__ | 1__ | 1__ | 1__ | 1__ |
| Indexed Byte         |     |     |     |     |     |     |     |     |     |     |
| COLUMN               | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 20  |
| Location in STRADDR  | 7A  | 7B  | 7C  | 7D  | 7E  | 7F  | 80  | 81  | 82  | 83  |
| ADDR of Indexed Byte | 1__ | 1__ | 1__ | 1__ | 1__ | 1__ | 1__ | 1__ | 1__ | 1__ |
| Indexed Byte         |     |     |     |     |     |     |     |     |     |     |

## SUGGESTED SOLUTION PROGRAM "SCAN1"

| ADDRS | DATA |    |    | LABEL   |
|-------|------|----|----|---------|
| B0    | B1   | B2 |    |         |
| 0000  | 3F   | 17 | 50 | SCAN1   |
| 3     | 93   |    |    |         |
| 4     | C1   |    |    |         |
| 5     | C3   |    |    |         |
| 6     | 72   |    |    |         |
| 7     | CA   | 04 |    |         |
| 9     | 0D   | 61 | 00 | AGAIN   |
| C     | F4   | XX |    | TEST    |
| 0E    | 3E   | 21 |    |         |
| 10    | D9   | 77 |    |         |
| 2     | F2   |    |    | FINISH  |
| 3     | 03   |    |    |         |
| 4     | BB   | F4 |    |         |
| 6     | C9   | 18 |    |         |
| 8     | C8   | 15 |    |         |
| A     | 20   |    |    |         |
| B     | C1   |    |    |         |
| C     | C3   |    |    |         |
| D     | 06   | 28 |    |         |
| 1E    | BB   | E6 |    |         |
| 21    | 04   | 17 |    |         |
| 3     | C8   | 0A |    |         |
| 5     | C8   | 09 |    |         |
| 7     | 9B   | 00 |    |         |
| 9     | 07   | 00 | 07 | TOTAL   |
| C     | 0A   | 11 | 16 |         |
| 2F    | 17   | 17 |    |         |
| 31    | 01   |    |    |         |
| 2     | CF   | 20 | 6F | STRADDR |
| 35    | 17   |    |    |         |





EXERCISE 1 (CONTINUED) :

5. Depress **RUN** once, then complete a single entry in Table 1.

NOTE: The first 2 entries in Table 1 are completed for example purposes.

- 5A. Repeat step 5 until all entries are complete.

6. Listen to tape 8B for a brief commentary on the results you observed in Table 1. A completed version is on the next page.

## ROUTINE "IDCCHK"

| ADDRS   | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |          | COMMENT                   |
|---------|------|----|----|--------|----------------------|----------|---------------------------|
|         | R2   | R1 | R0 |        | OPCODE               | OPERANDS |                           |
| 1 0000  | 75   | FF |    | IDCCHK | CPSL                 | FF       | Clear PSL, then set the   |
| 2 2 77  | 08   |    |    |        | PPSL                 | WC       | "With Carry" bit. Now in- |
| 3 4 04  | 07   |    |    |        | LODI, R0             | 7        | italize. R0=7. Leave      |
| 4 6 C0  | C0   | C0 | C0 |        |                      |          | space to expand, then     |
| 5 8 80  |      |    |    | AGAIN  | ADDZ                 | R0       | add R0 to itself. And     |
| 6 A D4  | 07   |    |    |        | WRITE, R0            | PORT 7   | write R0 to Port 7.       |
| 7 C FB  | 7E   |    |    |        | BDRR, R3             | \$       | Delay ~ 1/2 sec.          |
| 8 0E FA | 7C   |    |    |        | BDRR, R2             | \$-2     | and                       |
| 9 10 18 | 77   |    |    |        | BCTR, UN             | AGAIN    | repeat from Add forever   |

NOTES:

TABLE 1

|    | EXT I/O PORT 7 LEDs |   |   |   |   |   |   |   | PSL (HEX) | IDC | OVF | C |
|----|---------------------|---|---|---|---|---|---|---|-----------|-----|-----|---|
|    | 7                   | 6 | 5 | 4 | 3 | 2 | 1 | 0 |           |     |     |   |
| 1  |                     |   |   |   | X | X | X |   | 48        |     |     |   |
| 2  |                     |   |   | X | X | X |   |   | 68        | X   |     |   |
| 3  |                     |   |   |   |   |   |   |   |           |     |     |   |
| 4  |                     |   |   |   |   |   |   |   |           |     |     |   |
| 5  |                     |   |   |   |   |   |   |   |           |     |     |   |
| 6  |                     |   |   |   |   |   |   |   |           |     |     |   |
| 7  |                     |   |   |   |   |   |   |   |           |     |     |   |
| 8  |                     |   |   |   |   |   |   |   |           |     |     |   |
| 9  |                     |   |   |   |   |   |   |   |           |     |     |   |
| 10 |                     |   |   |   |   |   |   |   |           |     |     |   |
| 11 |                     |   |   |   |   |   |   |   |           |     |     |   |

Examination of PSW Bits IDC, OVF, and C resulting from ADD execution. Entries 1 and 2 are illustrated. X = ON (logic 1). Suggested completion on next page.



NOTES:

SOLUTION -----TABLE A

|    | EXT I/O PORT 7 LED <sub>s</sub> |   |   |   |   |   |   |   | PSL<br>(HEX) | IDC | OVF | C |
|----|---------------------------------|---|---|---|---|---|---|---|--------------|-----|-----|---|
|    | 7                               | 6 | 5 | 4 | 3 | 2 | 1 | 0 |              |     |     |   |
| 1  |                                 |   |   |   | X |   | X | X | 69           | X   |     | X |
| 2  |                                 |   |   |   | X |   |   |   | 69           | X   |     | X |
| 3  |                                 |   |   |   |   | X |   | X | 69           | X   |     | X |
| 4  |                                 |   |   |   |   |   | X |   | 69           | X   |     | X |
| 5  | X                               | X | X | X | X | X | X | X | 88           |     |     |   |
| 6  | X                               | X | X | X | X |   | X | X | A9           | X   |     | X |
| 7  | X                               | X | X | X | X |   |   |   | A9           | X   |     | X |
| 8  | X                               | X | X | X |   | X |   | X | A9           | X   |     | X |
| 9  | X                               | X | X | X |   |   | X |   | A9           | X   |     | X |
| 10 | X                               | X | X |   | X | X | X | X | 89           |     |     | X |
| 11 | X                               | X | X |   | X | X |   |   | A9           | X   |     | X |
| 12 | X                               | X | X |   | X |   |   | X | A9           | X   |     | X |
| 13 | X                               | X | X |   |   | X | X |   | A9           | X   |     | X |
| 14 | X                               | X | X |   |   |   | X | X | A9           | X   |     | X |
| 15 | X                               | X | X |   |   |   |   |   | A9           | X   |     | X |
| 16 | X                               | X |   | X | X | X |   | X | 89           |     |     | X |
| 17 | X                               | X |   | X | X |   | X |   | A9           | X   |     | X |
| 18 | X                               | X |   | X |   | X | X | X | A9           | X   |     | X |
| 19 | X                               | X |   | X |   | X |   |   | A9           | X   |     | X |

NOTE:

Do not complete this box until requested to do so on tape!

SUMMARY OF IDC,OVF, & C  
STATUS WHEN WC BIT IS RESET

SOLUTION ----- TABLE B

|    | EXT I/O PORT 7 LED <sub>s</sub> |   |   |   |   |   |   |   | PSL<br>(HEX) | IDC | OVF | C |
|----|---------------------------------|---|---|---|---|---|---|---|--------------|-----|-----|---|
|    | 7                               | 6 | 5 | 4 | 3 | 2 | 1 | 0 |              |     |     |   |
| 1  | X                               | X |   |   |   | X |   | X | 89           |     |     | X |
| 2  | X                               |   | X | X |   | X | X |   | 89           |     |     | X |
| 3  | X                               |   | X |   |   | X | X | X | 89           |     |     | X |
| 4  | X                               |   |   | X | X |   |   |   | 89           |     |     | X |
| 5  | X                               |   |   |   | X |   |   | X | 89           |     |     | X |
| 6  |                                 | X | X | X | X |   | X |   | 4D           |     | X   | X |
| 7  |                                 | X | X |   | X |   | X | X | 49           |     |     | X |
| 8  |                                 | X |   | X | X | X |   |   | 49           |     |     | X |
| 9  |                                 | X |   |   | X | X |   | X | 49           |     |     | X |
| 10 |                                 |   | X | X | X | X | X |   | 49           |     |     | X |
| 11 |                                 |   | X |   | X | X | X | X | 49           |     |     | X |
| 12 |                                 |   | X |   |   |   |   |   | 69           | X   |     | X |
| 13 |                                 |   |   | X |   |   |   | X | 49           |     |     | X |
| 14 |                                 |   |   |   |   |   | X |   | 49           |     |     | X |
| 15 | X                               | X | X | X |   |   | X | X | 89           |     |     |   |
| 16 | X                               | X | X |   |   |   | X | X | 89           |     |     | X |

**DIRECTION:** After completing the boxed summary, go on to Exercise :

# EXERCISE 3

## IDC, OVF, AND C DURING ROTATE INSTRUCTION EXECUTION

### INTRODUCTION:

Most programmers do not test the condition of the IDC and OVF bits after ROTATE instruction execution. Nevertheless, the ALU provides a certain manipulation of these bits. The CARRY bit also comes under manipulation during ROTATE execution. In certain cases, it MUST be recognized in order to prevent DATA SHIFT errors. You'll also see that the condition of the WC control bit is a very important determinant of results obtained when ROTATE is executed.

### PROCEDURE:

1. Code and load routine IDCROT into memory. The suggested solution is provided on the next page.

#### ROUTINE "IDCROT"

| 1 | ADDRS | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |          | COMMENT             |
|---|-------|------|----|----|--------|----------------------|----------|---------------------|
|   |       | P2   | E1 | P2 |        | OPCODE               | OPERANDS |                     |
| 1 | 0000  |      |    |    | IDCROT | CPSL                 | H'FF'    | Clear PSL and set   |
| 2 | 2     |      |    |    |        | PPSL                 | H'08'    | the WC bit. Then    |
| 3 | 4     |      |    |    |        | LODI, RI             | 1        | get a 1 and write   |
| 4 | 6     |      |    |    | AGAIN  | WRTE, RI             | PORT 7   | to Ext. % LEDs      |
| 5 | 8     |      |    |    |        | NOP                  |          | Do 2 NO-ops, then   |
| 6 | A     |      |    |    |        | RRL                  | RI       | Rotate RI left and  |
| 7 | B     |      |    |    |        | BCTR, UN             | AGAIN    | display before next |

2. Set the Program Counter to a START ADDRESS = '0'.
3. Set a BREAKPOINT ADDRESS = '08'.
4. Alternately depress **RUN**, and complete each entry in Table C (next page).

NOTE: The 1st entry in Table C should reflect the condition of the PSL before execution of the ROTATE instruction for the first time.

5. After completion of Table C, compare your results to those provided on the following page. Then, modify routine "IDCROT" as required for completion of Tables D, E, and F. Set the Program Counter to '00' before starting any new table entry.
6. After completion of all tables, AND comparison with suggested results, go on to PART 4 in this module. There is no taped commentary for this exercise.

NOTE: Completion of Tables E and F is optional.





## PART 4

## STACK POINTER

INTRODUCTION:

The STACK POINTER (PSU Bits 2,1, and 0) keeps the microprocessor informed of which one of 8 levels of program it is currently executing. Diagram A (next page) illustrates the concept of subroutine nesting. Whenever the 2650 accesses a new subroutine, the STACK POINTER is first incremented by 1. The contents of the Instruction Address Register are then stored in the RETURN ADDRESS STACK, a 15-bit register, 8 levels deep. The location in the RAS is determined by the STACK POINTER, whose value may range from 0 to 7. After storage of the return address (next instruction, program execution exits to the new subroutine.

Upon completion of the subroutine, a RETURN instruction may be executed to return program control to the calling routine. When a return (RETE or RETC) instruction condition is satisfied, the return address is transferred from the RAS to the Instruction Address Register. Then, the STACK POINTER is decremented by 1, and program is executed, starting with the instruction identified by the saved address.

When an INTERRUPT REQUEST is acknowledged by the 2650, the CPU executes an IMPLICIT ZBSR instruction. The current contents of the Program Counter, containing the address of the next executable instruction in the sequence, are stored in the RAS. The Stack Pointer is incremented, and program execution continues with process of the Interrupt Service routine.

NOTE 1: The process and sequence of INTERRUPT CONTROL are fully described in Module 5 of this course.

Return to the interrupted sequence from the Interrupt Service routine is enabled by execution of the RETC (no further interrupt enable) or RETE (enable interrupt recognition) instructions.

The Stack Pointer may be modified as well by execution of any appropriate PSW(UPPER) instruction. These include the PPSU, CPSU, and LPSU instructions. It is not a recommended practice to modify the Stack, as you'll hear shortly on tape.

NOTE 2: In STEP and BREAKPOINT modes, the INSTRUCTOR's Monitor program uses 2 levels of the Return Address Stack. Thus, user program stack usage is limited to 6 levels when STEP and BREAKPOINT are employed.

**DIRECTION:** Go right on to the next page.

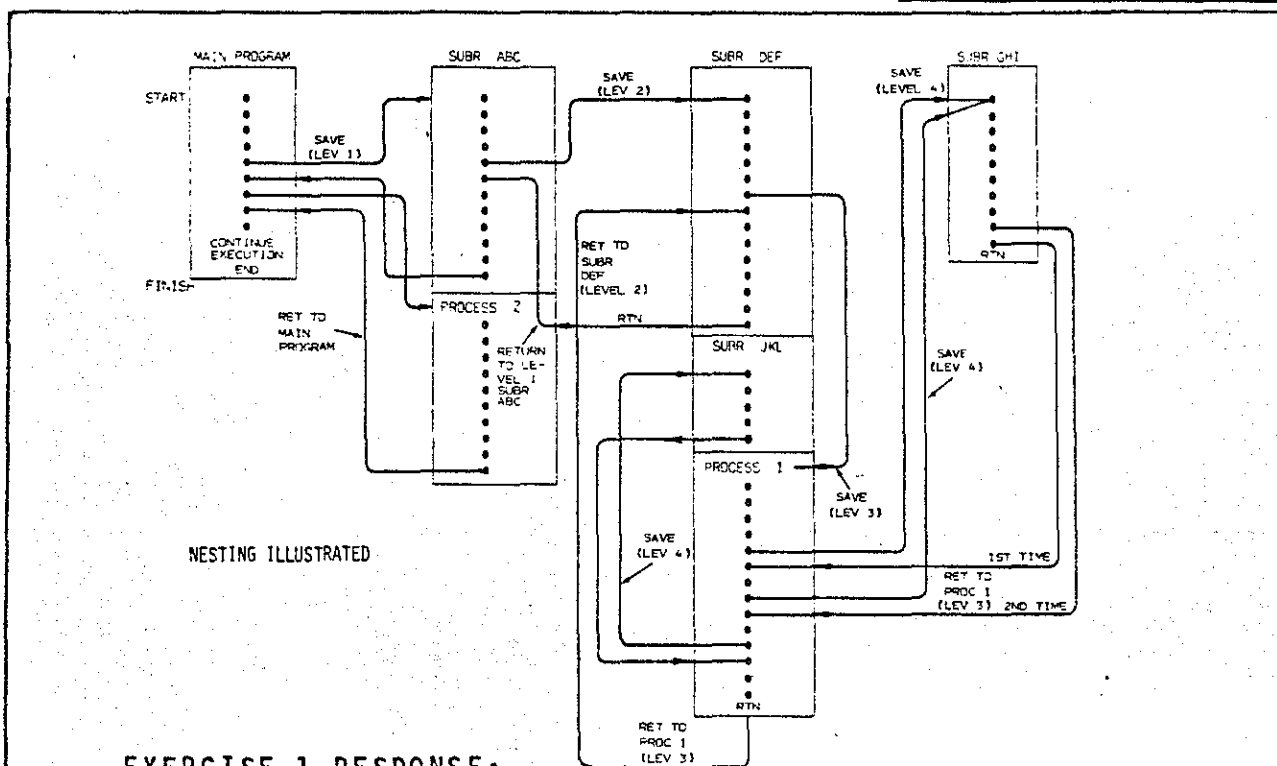
## EXERCISE 1

## SUBROUTINE NESTING CONCEPTS

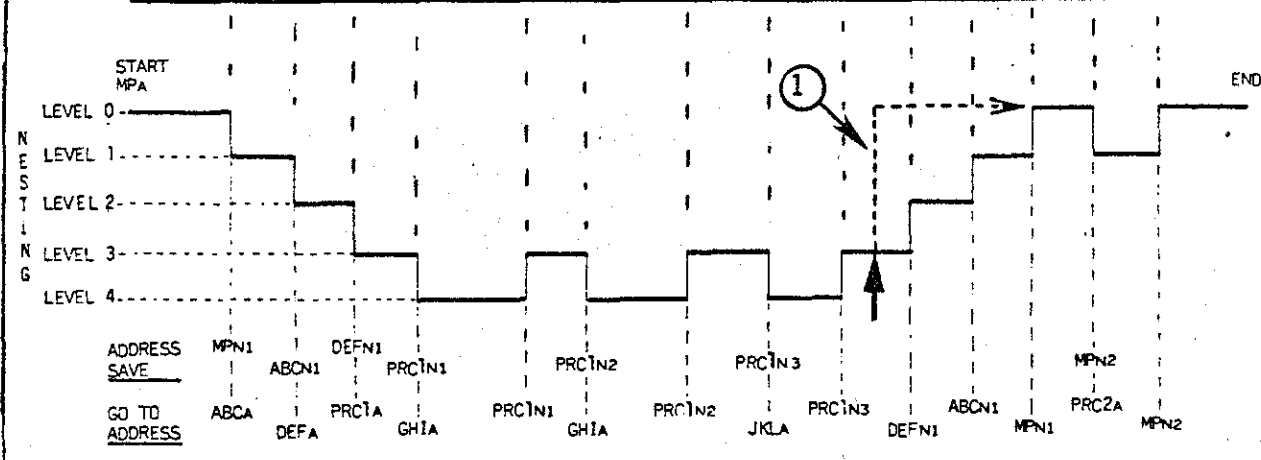
1. Take a few minutes to study the SUBROUTINE NESTING DIAGRAM (below).
2. Listen to the short commentary on tape BB.
3. In the space below, indicate the value of the Stack Pointer (0-7) as program execution sequences as shown in the diagram. Do not complete this step until directed on tape to do so. The suggested solution is provided on the next page.
4. After completion, go right on to the next page.

DIAGRAM 1

SUBROUTINE NESTING



EXERCISE 1 RESPONSE:  
Stack Pointer equals:

[illegible]



## PROGRAM "WRAP"

| ADDRS   | DATA |    |    | LABEL | SYMBOLIC INSTRUCTION |          | COMMENT                    |
|---------|------|----|----|-------|----------------------|----------|----------------------------|
|         | B0   | B1 | B2 |       | OPCODE               | OPERANDS |                            |
| 1 0000  | 74   | 07 |    | WRAP  | CPSU                 | SP       | clear the stack ptr. Set   |
| 2 2     | 05   | 80 |    |       | LODI, R1             | H'80'    | R1 as a visible constant   |
| 3 4     | D5   | 07 |    |       | WRTE, R1             | PORT7    | to display. Then go call   |
| 4 6     | 3F   | 00 | 10 |       | BSTA, UN             | SUBR1    | Subroutine 1. On return    |
| 5 9     | 05   | E0 |    |       | LODI, R1             | H'E0'    | Set R1 = H'E0' and dis-    |
| 6 3     | D5   | 07 |    |       | WRTE, R1             | PORT7    | play, then, exit to        |
| 7 D     | 30   |    |    |       | WRTC                 | RO       | the MONITOR.               |
| 8       |      |    |    |       |                      |          |                            |
| 9 10    | 12   |    |    | SUBR1 | SPSU                 |          | Fetch PSU and mask for     |
| 10 1    | 44   | 07 |    |       | ANDI, RO             | SP       | str ptr. Now add to visi-  |
| 11 3    | 81   |    |    |       | ADDZ                 | R1       | ble constant and display   |
| 12 4    | D4   | 07 |    |       | WRTE, RO             | PORT7    | on LEDs. Then exit to      |
| 13 6    | 3B   | 08 |    |       | BSTR, UN             | SUBR2    | next subroutine. On re-    |
| 14 8    | 12   |    |    |       | SPSU                 |          | turn, fetch PSU and mask   |
| 15 9    | 44   | 07 |    |       | ANDI, RO             | SP       | str ptr again. Add the vi- |
| 16 B    | 81   |    |    |       | ADDZ                 | R1       | sible constant again &     |
| 17 C    | D4   | 07 |    |       | WRTE, RO             | PORT7    | display on LEDs. Then re-  |
| 18 1E   | 17   |    |    |       | RETC, UN             |          | turn to calling routine    |
| 19      |      |    |    |       |                      |          |                            |
| 20 20   |      |    |    | SUBR2 | RES                  | 15       | Same code as SUBR1         |
| 21 30   |      |    |    | SUBR3 | RES                  | 15       | START addresses as in-     |
| 22 40   |      |    |    | SUBR4 | RES                  | 15       | dicated. NOTE: Use a sui-  |
| 23 50   |      |    |    | SUBR5 | RES                  | 15       | table RELOC. ROUTINE W     |
| 24 60   |      |    |    | SUBR6 | RES                  | 15       | start addr @ H'100' for    |
| 25 70   |      |    |    | SUBR7 | RES                  | 15       | loading SUBR2-9 into       |
| 26 80   |      |    |    | SUBR8 | RES                  | 15       | USER stg. Enter SUBRA      |
| 27 90   |      |    |    | SUBR9 | RES                  | 15       | (follows) separately.      |
| 22      |      |    |    |       |                      |          |                            |
| 23 00A0 | 12   |    |    | SUBRA | SPSU                 |          | Fetch PSU and mask the     |
| 24 1    | 44   | 07 |    |       | ANDI, RO             | SP       | str. ptr, then add the vi- |
| 25 3    | 81   |    |    |       | ADDZ                 | R1       | sible constant. Display    |
| 26 4    | D4   | 07 |    |       | WRTE, RO             | PORT7    | on LEDs AND return         |
| 27 A6   | 17   |    |    |       | RETC, UN             |          | to calling routine.        |

EXERCISE 2 PROCEDURE (continued):

4. After completing all Table 1 entries on the next page, compare your responses to those shown on the following page.

TABLE 1 PROGRAM "WRAP ENTRIES"

| BKPT ADDR | PSU | STK PTR | NEXT ADDR           | COMMENTS |
|-----------|-----|---------|---------------------|----------|
| 04 •      |     |         | (inspect P.C.)<br>↓ |          |
| 14        |     |         |                     |          |
| 24        |     |         |                     |          |
| 34        |     |         |                     |          |
| 44        |     |         |                     |          |
| 54        |     |         |                     |          |
| 64        |     |         |                     |          |
| 74        |     |         |                     |          |
| 84        |     |         |                     |          |
| 94        |     |         |                     |          |
| A4        |     |         |                     |          |
| A6 •      |     |         |                     |          |
| 9C        |     |         |                     |          |
| 9E •      |     |         |                     |          |
| BC        |     |         |                     |          |
| 8E •      |     |         |                     |          |
| 7C        |     |         |                     |          |
| 7E •      |     |         |                     |          |
| 6C        |     |         |                     |          |
| 6E •      |     |         |                     |          |
| 5C        |     |         |                     |          |
| 5E •      |     |         |                     |          |
| 4C        |     |         |                     |          |
| 4E •      |     |         |                     |          |
| 3C        |     |         |                     |          |
| 3E •      |     |         |                     |          |
| 2C        |     |         |                     |          |
| 2E •      |     |         |                     |          |
| 2C        |     |         |                     |          |
| 2E •      |     |         |                     |          |

ADDITIONAL DIRECTIONS:NOTE:

STEP addresses identified with a "•" suffix. For all other addresses, set a BREAKPOINT and depress RUN.

NOTE:

The most significant digit of the PSU is "don't care" for this exercise.

\* After completing Table 1 at location '6E', set a Start address = '0'; BKPT. = '6E'; and depress RUN. Otherwise, the MONITOR's use of the RAS will interfere with further steps.

OPTION:

Try single stepping from location 6E. At what point do you exit to the MONITOR's USE program ?

\*\*

After completion of this entry set a START ADDR at '0' and BKPT at '2C'. Depress RUN and complete the entries below the heavy line.

OPTION:

Try single stepping from location '2E'. Do you eventually access the MONITOR USE program ?

DIRECTION:

Compare your entries with those provided on the next page.

EXERCISE 3 (continued):

8. Listen to tape 8B for further commentary, then go on to Step 9.

**NOTES:** \_\_\_\_\_

9. Now, depress RESET. Is the instruction at location '0B' executed?

(Yes)(No). Why or why not ?

**DIRECTION:**

Compare your answer to that provided on tape BB.  
Then go on to Exercise 4.

SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

## EXERCISE 4

## STACK POINTER MODIFICATION

INTRODUCTION:

In this exercise, the Stack Pointer is modified to permit BYPASS of functions contained in subroutines\_\_\_\_through\_\_\_\_. This practice is NOT RECOMMENDED for general use because of possible generation of PROGRAM SEQUENCE ERRORS. The KEY to your effective modification of the Stack Pointer is your capability of PREDICTING the results that will take place.

NOTE: THE BLANK SPACES IN LINE 2 OF THIS PARAGRAPH ARE INTENTIONAL, YOU'LL COMPLETE THE SENTENCE IN STEP 4.

PROCEDURE:

1. Modify the addresses in "SUBR5" as shown in the listing (below)

| ADDRS | DATA |    |    | LABEL | SYMBOLIC INSTRUCTION |          | COMMENT                |
|-------|------|----|----|-------|----------------------|----------|------------------------|
|       | B0   | B1 | B2 |       | OPCODE               | OPERANDS |                        |
| 56    | 74   | 04 |    |       | CPSU                 | SP2      | CLEAR SP2, STK PTR = 1 |
| 8     | 12   |    |    |       | SPSU                 |          | THEN GET PSU AND DIS-  |
| 9     | 04   | 07 |    |       | WRTE, R0             | PORT7    | PLAY IT ON LED'S.      |
| 8     | 17   |    |    |       | RETC, IIN            |          | THEN, RETURN           |

2. Before you execute "WRAP", predict the location of the NEXT INSTRUCTION after the RETE,UN instruction is executed. \_\_\_\_\_
3. Set a Start Addr. at location '0', then STEP rapidly to location '56'. Then, SLOWLY STEP through location '58'.  
"The next instruction to be fetched is at location: \_\_\_\_\_"
4. Now, complete the FIRST sentence of the Introduction to this exercise. Compare your answer to that provided on Tape 8B.

**NOTE:** If your predicted and observed responses varied, try Exercise 4 again until you are sure you can predict the "next instruction" with accuracy.

**DIRECTION:**

Go right on to Module IV-I in the course. There is no further instruction on the PSW.

# Signetics

## 2650 MICROPROCESSOR COURSE

### MODULE IV - I

#### MONITOR PROGRAMS AVAILABLE TO USER PROGRAM ACCESS

**PREPARED BY:**

MICROPROCESSOR TRAINING DEPARTMENT  
Signetics Corporation  
811 E. Arques Ave.  
Sunnyvale, CA, 94086

**REFERENCE:**

Outlines and Abstracts  
page 22



INTRODUCTION:

Extensive use of MONITOR subroutines has been made in several of the programs illustrated in this course. For example, routines USRDSP, GNPA, AND DISLSD were incorporated into program "CRAPGAME" through appropriate use of ZBSR \* instructions. Use of these routines provided an extra dimension to "CRAPGAME", a capability not otherwise possible in programs of less than 1 kbytes.

The INSTRUCTOR 5D Reference Manual contains a brief description of routines accessible to the User program.

The following table provides a short-form listing of routines available to your INSTRUCTOR-executed programs. Each routine is fully listed and commented in the Reference Manual; Appendix C. Routines preceded by an asterisk (\*) are further amplified in the remaining pages of this section.

TABLE 1 INDEX TO USER-ACCESSABLE MONITOR SUBROUTINES

| MNEMONIC       | INDIRECT ADDRESS LOCATION | ABSOLUTE ADDRESS LOCATION | RELATIVE INDIRECT ADDRESS | SYMBOLIC ADDRESS CONSTANT | FUNCTION                                                                  | REF. PAGE |
|----------------|---------------------------|---------------------------|---------------------------|---------------------------|---------------------------------------------------------------------------|-----------|
| * USROSP       | 1FE6                      | 1FD5                      | E6                        | USROSI                    | Entry to display routines                                                 | C39       |
| ERR            | 1FEB                      | 1B99                      | E8                        | ERRI                      | Error message generation                                                  | C6        |
| BRKPT4         | 1FEA                      | 19E8                      | EA                        | BRKPTI                    | Load Display Buffer 6 & 7 with the contents of R0. Calls DISLSD.          | C11       |
| DISPLY         | 1FEC                      | 1E13                      | EC                        | DISPLI                    | Display and Keyboard routine (Note 2)                                     | C29       |
| IN             | 1FEE                      | 1EF6                      | EE                        | INN                       | Cassette input routine                                                    | C35       |
| DUT            | 1FF0                      | 1EBA                      | F0                        | OUTT                      | Cassette output (print) routine                                           | C34       |
| HOUT           | 1FF2                      | 1C7B                      | F2                        | HOUTT                     | Cassette: Convert binary input to ASCII hex output                        | C21       |
| * DISLSD       | 1FF4                      | 1A76                      | F4                        | DISLSI                    | Convert a byte into 2 nibbles                                             | C13       |
| * RDT          | 1FF6                      | 18A5                      | F6                        | ROTI                      | Exchange high and low order nibbles in R0.                                | C18       |
| CRLF           | 1FFB                      | 1C72                      | FB                        | CRLFF                     | Generate carriage return and line-feed codes and output to cassette.      | C21       |
| * GNP          | 1FFA                      | 1B3B                      | FA                        | GNPI                      | Keyboard entry of data into user program. Display 4 or 5 digits.          | C17       |
| * GNPA         | 1FFC                      | 1B20                      | FC                        | GNPAI                     | Entry message same as GNP, except for display of entry data throughout.   | C17       |
| * MOV (Note 1) | 1FFE                      | 1086                      | FE                        | MOVI                      | Move 8 bytes of data from user-specified location into the display buffer | C27       |
| SAVRG          | --                        | 1EAG                      | --                        | SAVRG                     | Save 1 bank of registers excluding R0 in the display buffer.              | C33       |
| RESTRO         | --                        | 1EB3                      | --                        | RESTRO                    | Restore display buffer to one bank of registers.                          | C33       |

Note 1: Used in GNP, GNPA, & USROSP routines.

Note 2: called by USRDSP routine.

**DIRECTION:**

Go right on to the next page.

## USRDSP

## DISPLAY USER-DEFINED MESSAGE

INTRODUCTION:

The user may include the following procedure into his program to permit display of specific 8-character messages. The controlling access routine requires 8 bytes of user memory space, in addition to 1 byte for each character to be displayed.

CHARACTER FONT:

| <u>SYMBOL</u> | <u>CODE</u> | <u>SYMBOL</u> | <u>CODE</u> | <u>SYMBOL</u>     | <u>CODE</u> |
|---------------|-------------|---------------|-------------|-------------------|-------------|
| 0,0           | 00          | B             | 0B          | O                 | 15          |
| 1,I           | 01          | C             | 0C          | =                 | 16          |
| 2,Z           | 02          | D             | 0D          | blank             | 17          |
| 3             | 03          | E             | 0E          | J                 | 18          |
| 4             | 04          | F             | 0F          | - (DASH OR MINUS) | 19          |
| 5,S           | 05          | P             | 10          | *                 | 1A          |
| 6,G           | 06          | L             | 11          | Y                 | 1B          |
| 7,T           | 07          | U             | 12          | □ (SMALL BOX)     | 95          |
| 8             | 08          | R             | 13          | N                 | 1C          |
| 9             | 09          | H             | 14          | T                 | 6C          |
| A             | 0A          |               |             |                   |             |

Note: For symbol plus right justified decimal point, add H'80' to the character code

EXAMPLE:

In an application where the user wishes to display a "speed function" applied to controlled I70, the following program might be executed.

```

addr. data
0100 05 01 } Control
 06 0F } Program
 07 00 }
 BB E6 }
 continue
 .
 .
 .
 .

```

(explained on next page)

At location '0110', enter the character codes desired in the 8-byte display.

Example: To display "SPEED 3", enter:

05 10 0E 0E 0D 17 17 03

in addresses 0110 to 0107

The "speed factor":3, might have been assembled previously by the microprocessor, then moved to loc. '0107' prior to calling the display parameter control program at loc. '0100'.

DIRECTION:

Go right on, to the next page.

## USRDSP (CONTINUED)

CONTROLLING PROGRAM

- LODI,R1 H'XX' } where H'XXYY' = the address of the first byte
  - LODI,R2 H'YY' } in the Data Table MINUS.1
  - LODI,R3 Z where Z = one of the following display function options:
- 0 ..... display of data until any hex or FUNCTION key is depressed to exit the routine. See Note 1.
- 1 to 127 Any positive number permits ONE pass through the display routine, then exit to the calling routine. No manual intervention is required. You should incorporate a COUNTER function to repeat display in TIME DEPENDENT programs. See Note 2.
- 128 to 255 e.g. any negative number. Same as 0 except decimal point is on in MSD of display. Exit the routine by depressing any key. See Note 1.
- ZBSR \*USRDSP to transfer control to display routine. Code for this instruction is BB E6. See Note 3.

Note 1: KEY-PRESSED CODE RESIDING IN R0 ON RETURN:

| key | 'code' | key | 'code' | key   | 'code' |
|-----|--------|-----|--------|-------|--------|
| 0   | 00     | 8   | 08     | WCAS  | 80     |
| 1   | 01     | 9   | 09     | BKPT  | 81     |
| 2   | 02     | A   | 0A     | RCAS  | 82     |
| 3   | 03     | B   | 0B     | REG   | 83     |
| 4   | 04     | C   | 0C     | STEP  | 84     |
| 5   | 05     | D   | 0D     | MEM   | 85     |
| 6   | 06     | E   | 0E     | RUN   | 86     |
| 7   | 07     | F   | 0F     | ENT/} | 87     |
|     |        |     |        | NXT } |        |

NOT VALID:

SENS  
INT  
MON  
RST

Note 2: If single pass option (R3=1 to 127) is selected, a "No Key Depressed" code ('88') resides in R0 on exit from the display routine.

Note 3: Alternative instructions to access the display routine:

BSTA,UN \*USRDSP address is '1FE6'  
BSTR,UN \*USRDSP

**DIRECTION:** Go right on to the next page.

## USRDSP (CONTINUED)

## INDIRECT CALL

INTRODUCTION:

Your application may require that some messages be timed when output to the HEX DISPLAY, and that others remain until some key is depressed. This dual requirement is easily implemented by including a separate DISPLAY CONTROL subroutine in your program. "DISMES", an example reprinted from the program 'CRAPGAME', is illustrated below.

DIRECTION:

Listen to tape 8B for a brief commentary on MESSAGE DISPLAY CONTROL routine "DISMES". Mainline program parameter definition requirements are listed on the next page.

|    | ADDRS | DATA |    |    | LABEL     | SYMBOLIC INSTRUCTION |           | COMMENT                       |
|----|-------|------|----|----|-----------|----------------------|-----------|-------------------------------|
|    |       | B0   | B1 | B2 |           | OPCODE               | OPERANDS  |                               |
| 1  | 0137  | XX   |    |    | >MESLOC-1 | RES                  | 1         |                               |
| 2  | 8     | XX   |    |    | DSPLDLY   | RES                  | 1         |                               |
| 3  | 9     | XX   |    |    | RUNDSPX   | RES                  | 1         |                               |
| 4  |       |      |    |    |           |                      |           |                               |
| 5  | 013A  | CA   | 7B |    | DISMES    | STRR, R2             | >MESLOC-1 | Store current index to mes-   |
| 6  | C     | 05   | 02 |    |           | LODI, R1             | 2         | sage table. Now, setup        |
| 7  | 13E   | C9   | 79 |    | REPEATM   | STRR, R1             | RUNDSPX   | running display timeout       |
| 8  | 140   | CB   | 76 |    | DSPAGN    | STRR, R3             | DSPLDLY   | const & store. Also store     |
| 9  | 2     | 05   | 17 |    |           | LODI, R1             | <MESLOC-1 | static/dynamic displ. par-    |
| 10 | 4     | 0A   | 71 |    |           | LODR, R2             | >MESLOC-1 | ameter. Now set index to      |
| 11 | 6     | BB   | E6 |    |           | ZBSR                 | *USRDSP   | specified msg & display.      |
| 12 | 8     | 0B   | 6E |    |           | LODR, R3             | DSPLDLY   | (If static, player depresses  |
| 13 | A     | E7   | 00 |    |           | COMI, R3             | 0         | key to exit) Fetch displ. st. |
| 14 | C     | 14   |    |    |           | RETC, EQ             |           | if static, exit. If dynamic,  |
| 15 | D     | E7   | 01 |    |           | COMI, R3             | 1         | is it a 1 yet? Yes, exit to   |
| 16 | 14F   | 18   | 02 |    |           | BCTR, EQ             | CONTDSP   | continuous display. No, re-   |
| 17 | 151   | FB   | 6D |    |           | BDRR, R3             | DSPAGN    | peat single pass display.     |
| 18 | 3     | 09   | 64 |    | CONTDSP   | LODR, R1             | RUNDSPX   | To continuous running disp-   |
| 19 | 5     | F9   | 01 |    |           | BDRR, R1             | \$+3      | lay, get its constant and     |
| 20 | 7     | 17   |    |    |           | RETC, UN             |           | decrement. If zero, exit      |
| 21 | 8     | 07   | 7F |    |           | LODI, R3             | 7F        | to calling routine. If not    |
| 22 | A     | 1B   | 62 |    |           | BCTR, UN             | REPEATM   | set dynamic parameter         |
| 23 |       |      |    |    |           |                      |           | again & repeat.               |
| 24 |       |      |    |    |           |                      |           | last addr = '15B.             |
| 25 |       |      |    |    |           |                      |           | next routine = DCDROL         |
| 26 |       |      |    |    |           |                      |           |                               |
| 27 |       |      |    |    |           |                      |           |                               |

"DISMES" PARAMETER SPECIFICATION:● MEMORY ORGANIZATION

- Dedicate a block of memory  $N \times 8$  bytes in length, where  $N$  = the number of messages to be accessed and displayed.

LIMITS:  $1 < N < 32$  MESSAGES

● SEQUENCE:

Normally, messages are arrayed in the order in which they will appear. If main program incremental indexing is required (e.g.: several 'words' in the same message), the message segments must be sequenced in their appearance order.

e.g.: YOU BEAT  
THEHOUSE

This same rule applies if the messages are accessed on an EVENT basis

- The message table must reside within a given 256-byte memory block, so that the most sig. byte of message address can be programmed as a constant.

● USER MAINLINE PROGRAM PARAMETERS:

Prior to calling DISMES, the following parameters are entered:

1. >MESLOC-1 → R2

The low-order addr. of the message's first byte MINUS 1 is stored in R2.

2. DSPLDLY → R3

- is the number of times the display will repeat in one display event.

Program: H'01' to H'7F'

3. RUNDSPX → R1

- If R3 = '00' prior to calling "DISMES", the desired message will be displayed until a HEX or FUNCTION key is hit. The EVENT-testing software (RUNDSPX-next parameter) is NOT executed.

The number of RUN DISPLAY Events. If the running time for all auto-timed messages is equal, RUNDSPX may be programmed as a constant.

● SUBROUTINE NESTING

- 3 levels of address are saved:

call DISMES

    ↪ call USRDSP

        ↪ call DISPLY

- If an interrupt request is made, add another level.
- Interrupt Inhibit is not automatic, nor are interrupts enabled automatically upon return from subroutine, as this routine is presently written.

**DIRECTION:** Go right on to the next page

**DISLSD**

CONVERT RO BYTE TO NIBBLES IN RO AND R1

FUNCTION:

This routine converts a byte of hex data in R0 into 2 hex nibbles. The nibbles, padded with leading zeros, are stored in R0 (MSD) and R1 (LSD) as the routine's execution is completed. WITH CARRY (PSL Bit 3) is also reset.

APPLICATION:

This routine is particularly useful if the contents of any memory location, register, or I/O port are to be displayed on the 8-digit hex readout.

SYMBOLIC ADDRESS AND LOCATION:

Absolute: DISLSI at location '1A76'

Indirect: \*DISLSD at location '1FF4'

Relative indirect: \*DISLSD = 'F4'  
(ZBSR)

NOTE: Program access by subroutine call instructions only to ensure proper return.

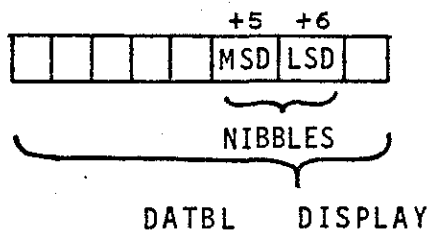
During execution of DISLSI, subroutine ROT is called. ROT exchanges the nibbles in R0.

EXAMPLE:

```
REDE,R0 PORT7 Extended I/O Address 7
ZBSR *DISLSD
STRA,R0 DATBL+5
STRA,R0 DATBL+6
```

- 
- .....and continue with user message display routine.

The sequence described above prepares the current condition of I/O Port 7 switches for subsequent hex display. The 8-byte message is located in "DATBL".

**DIRECTION:**

Go right on to the next page.

GNP  
GNPA

## KEYBOARD DATA ENTRY TO USER PROGRAM

INTRODUCTION:

The user may include MANUAL entry of data variables into his program via the keyboard by accessing the following MONITOR routines:

- \*MOV To place fixed display data in the MONITOR's display buffer.
- \*GNP To permit keyboard entry of new data
- \*GNPA See Note 1(below).

APPLICATIONS:

- During program debug, you can save much time involved in switching INSTRUCTOR operation modes when 'fine-tuning' delays, critical time paths, etc., for time-dependent applications.
- In controls applications, use of these routines provides the capability to "direct" externally controlled I/O devices.

PROGRAM:

```

LODI,R1 H'XX' }
LODI,R2 H'YY' } where H'XXYY' is the address of fixed display
 data in the user program MINUS 1.
ZBSR MOV to place display data (1 to 8 characters)
 in the MONITOR's display buffer
 See Note 4 (next page).

LODI,RD H'ZZ' where 'ZZ' defines keyboard and display input
 parameters. See the TABLE on the next
 page.

ZBSR GNP or }
ZBSR GNPA } to permit keyboard entry of new data.(NOTE 4)

```

ADDRESSES:

|      | <u>Absolute</u> | <u>Indirect</u> | <u>ZBSR</u> |
|------|-----------------|-----------------|-------------|
| MOV  | '1DB6'          | '1FFE'          | 'FE'        |
| GNP  | '1B3B'          | '1FFA'          | 'FA'        |
| GNPA | '1B2D'          | '1FFC'          | 'FC'        |

NOTE 1: Both GNP and GNPA routines display fixed data upon entry. Dependent on input conditions initialized in RD, routine GNP ("Get New Parameters") blanks 3 or 4 least sig. digits of display on entry. GNPA (Get New Parameters And display) displays the entire 8-character message prior to the user depression of the first data key. Execution of GNP and GNPA is identical thereafter.

**DIRECTION:** Go right on to the next page.

## GNP / GNPA (CONTINUED)

- NOTE 2:**
- Depression of **MON** to exit terminates execution of the User program. Program control is returned to the MONITOR. The PROGRAM COUNTER must be PRESET prior to execution of the user routine.
  - Do not depress **INT** or **SENS** to exit from GNP or GNPA

- NOTE 3:**
- Breakpoint to user locations is permitted.
  - Single step mode is permitted to user locations. The program aborts (with an 'ERROR 9' display) if the user attempts to **STEP** into MONITOR memory space.

- NOTE 4:** For Breakpoint purposes during debug, follow each ZBSR \* subroutine call with a NOP instruction ('CO').

## INPUT PARAMETERS TABLE

| R0 = | DESCRIPTION                                                                                                                      | R0  | R1  | R2                | R3 |
|------|----------------------------------------------------------------------------------------------------------------------------------|-----|-----|-------------------|----|
| 0    | Enter 1 to 4 digits. Change until exit by depressing FUNCTION key. 4 MSD of fixed data remain displayed until exit from routine. | LSB | MSB | FUNC. CODE        | 00 |
| 1    | Enter 1 or 2 digits. Change until exit by depressing FUNCTION key. 5 MSD of fixed data remain displayed until exit from routine. | LSB | 00  | FUNC. CODE        | 00 |
| 2    | Same as R0 = 0                                                                                                                   | LSB | MSB | F.C.              | 00 |
| 3    | Enter 1 or 2 digits. NO CHANGE. Exit on 3rd key depressed. 5 MSD of fixed data display until exit.                               | LSB | 00  | FUNC. or HEX CODE |    |
| 4    | Same as R0 = 0                                                                                                                   | LSB | MSB | F.C.              | 00 |
| 5    | Enter 1 digit. NO CHANGE. Exit on 2nd key depressed. 5 MSO of fixed data remain displayed until exit.                            | LSN | 00  | F.C. or H.C.      | 00 |
| 6    | Same as RD = 0                                                                                                                   | LSB | MSB | F.C.              | 00 |
| 7    | Same as R0 = 5                                                                                                                   | LSN | 00  | F.C. or H.C.      | 00 |

NOTE 2

**KEY:** LSB least significant byte contains 2 L.S. nibbles.  
 MSB most significant byte contains 2 M.S. nibbles.  
 F.C. Code of Function key depressed to exit.  
 DO NOT USE MON, RST, INT, OR SENS.  
 H.C. HEX digit code  
 LSN least sig. nibble - R0 = H'0X' where X is the hex code of the depressed key.

**DIRECTION:** Go right on to the next page.



ROTEXCHANGE NIBBLES IN ROINTRODUCTION:

This routine rotates data in RO 4 bits left, with data in Bit 7 wrapped around into Bit 0. Both carries, also Overflow, are unaffected since the WITH CARRY bit (PSL-3) is reset prior to data rotation.

ADDRESSES:

Symbolic: ROTI  
Absolute: '1BA5'  
Indirect: '1FF6'  
ZBSR \* \*RDT = 'F6'

ACCESS:

By subroutine call only (ZBSR, BSTA, etc.,). Upon executing ROTI, a RETC, UN instruction returns control to the calling user routine.

APPLICATION:

Exchange data for internal number detection.

EXAMPLE:

LODA, RO DATA  
ZBSR \*ROTI

**DIRECTION:**

There is no further audio commentary in this section. Go right on to MODULE IV-J, starting on the next page.



## 2650 MICROPROCESSOR COURSE

### MODULE IV - J

#### COMPREHENSIVE PROGRAM EXAMPLE

#### " TRAIN "

#### PREPARED BY:

MICROPROCESSOR TRAINING DEPARTMENT  
Signetics Corporation  
811 E. Arques Ave.  
Sunnyvale, CA, 94086

#### REFERENCE:

Outlines and Abstracts  
page 23

## OVERVIEW

INTRODUCTION:

It is fitting that the final section in this module provide an overall examination of activities performed by the programmer himself. For example, his specification of the application's internal and objective parameters. Like flowcharting the general and detailed sequences to be executed by the microprocessor and its controlled I/O. Like selection of the BEST instructions necessary to drive the application at the machine language level. And his activities involved in the program's debug, and its final integration with hardware.

During our examination of the instruction set's flexibility, you re-indirect instruction regarding specification of the routines you wrote and coded. And, considerable experience in debugging software problems, whether generated by yourself, or as a PLANNED function of the exercise you were performing.

Now, in this section, we'll trace an application, called "TRAIN", from its overall conception as an idea to its logical conclusion as a working program. In this conclusion, "TRAIN" will operate interactively with controlled I/O to perform its many functions.

The following SEQUENCE is performed:

1. GATHER FACTS

In this step, we'll interpret the "Inventor's" statements of what he wants to accomplish, and IMPORTANTLY, in what time sequence he wants his functions to take place.

2. WRITE THE PRELIMINARY OBJECTIVE SPECIFICATION

From the facts gathered in step 1, you will prepare short statements which define PRECISELY:

- what is controlled
- what operations are to take place
- their proposed sequence
- proposed division of the application into hardware and software elements.

3. FLOWCHART THE OVERALL AND DETAILED SEQUENCES OF OPERATIONS

Based on the facts and preliminary objective spec., you'll prepare a series of FLOWCHARTS to define the sequence of software to be executed by the microprocessor. During this step, you should also make a preliminary estimate of MEMORY REQUIREMENTS and dedication to specific functions, e.g.:

- tables
  - probable routine size
  - temporary scratchpad definition
- You'll also designate HARDWARE on board the INSTRUCTOR as 'pseudo' controlled external I/O.

**DIRECTION:**

Go right on to the next page.

4. WRITE THE SOFTWARE

After checking the truth of your flowcharts with the model provided in this section, you'll write the SOURCE PROGRAM and the OBJECT CODE for each routine.

5. DEBUG THE PROGRAM

Using the INSTRUCTOR's facilities, you'll debug each routine in terms of its sequence and control of external I/O. You'll ensure that each function defined in the Objective spec. and Flowcharts occur in PROPER SEQUENCE.

6. SYSTEMS CHECKOUT

You'll link all routines as one COHESIVE program, and, through their execution, you ensure that all selected I/O functions correctly.

7. PREPARE OBJECT PROGRAM TAPES FOR ROM

While you can not perform this step, you can ensure that all UNPROGRAMMED 'Control Store ROM' locations contain NOP instructions, or H'00 if no sequence is to be executed in those locations.

NOTE: For the purposes of this exercise, assume that SCRATCHPAD RAM may be located in non-adjacent blocks of memory, e.g.: anywhere. In practice however, it is normal to dedicate one or two blocks of addresses as SCRATCHPAD, with the remainder of memory designated as ROM.

ADDITIONAL INSTRUCTIONS:

1. You are free at any time to compare the details of your activities with those provided in this section. This applies to all steps performed in this sequence.
2. At the conclusion of each activity, you'll be directed to listen to Tape for further explanation. Taped commentaries must refer to suggested models contained in this section.
3. THE MORE OF THIS SEQUENCE YOU CAN PERFORM WITHOUT REFERENCE TO THE MODELS, THE BETTER!!!! This applies particularly to routine organization and source code preparation. As long as your product performs consistently within the guidelines detailed in the Objective Spec., there is no problem. If your sequence differs radically with the functions specified in this section, you should diagnose and debug the problem.

**DIRECTION:**

Go right on to the next page.

**DIRECTION:**

Note: In your outline, reference to data on this sheet is sufficient; you need not rewrite this information.

[illegible]

DIRECTION: Reverse: Depress **SENS**.  
Forward: no SENS key

TRAIN SELECTION:

Depress **INT** to execute.

INITIAL MESSAGE ON LOAD:

.2650 UP

TRAIN "CONSIST" : SWITCH BITS 3,2,1,&0

## DISPLAY



Consist  
I.D. = 2

DEFINITIONS:

| STOCK  | GRAPHIC | CODE  | DESCRIPTION                                                                                                       |
|--------|---------|-------|-------------------------------------------------------------------------------------------------------------------|
| HSE    |         | 95 EO | High-speed engine                                                                                                 |
| LSE    |         | 95 96 | low-speed engine                                                                                                  |
| MT     |         | C7    | Battery-powered mine tractor                                                                                      |
| EM     |         | 8D    | Overhead electric power mine engine                                                                               |
| FC     |         | 80    | Full freight car                                                                                                  |
| 1/2 FC |         | EE    | half-full freight car                                                                                             |
| EC     |         | 92    | Empty freight car                                                                                                 |
| C      |         | 95    | is a Caboose EXCEPT:<br><ul style="list-style-type: none"> <li>• in the "mine"</li> <li>• in CONSIST 7</li> </ul> |

**Note:** Numbers in parentheses indicate how many of each freight car are in the consist.

ADDITIONAL FACTS: FROM TAPED DISCUSSION

- 16 Train CONSISTS
  - codes on preceding page
  - each is 8 bytes - probably requires a 16 x 8-byte table (lots of indexing)
- Run FWD/BACKWARDS
  - SENS down to reverse.
  - FLAG on during reverse
- 7 SPEEDS RELATIVE
  - see preceding page
  - Ext I/O toggles 6 to 4
- DISPLAY 8-DIGIT
  - byte codes move right to left (FWD)
  - left to right (REV)
  - must set up byte movements prior to display each time. Can I index this? Do I need a temporary holding area for moving bytes around? ....Probably so, because if I make a 16x8 table, I can not overlay or destroy it!
- TRAIN SELECTION
  - by toggles 3 - 0. Let's call each consist 1 of 16 hex nrs.
  - effect by depressing INT key

AUTHOR'S NOTES:

Write a 2 byte indirect address to the START ADDR. of the routine which selects the Train consist consist. This address goes in locations '07' and '08'. INTERRUPT TOGGLE SW. SETTINGS:

Direct/Indirect = Indirect  
Keyboard/RealTime = Keyboard

- probably need 2 holding areas for selected train:
  - 1 for display
  - 1 for temp stg of moving bytes
  - involves double data xfer, but I can't use up all my regs just moving bytes around. Treat eachpart of the consist as a character or byte.
- SPEED AGAIN!
  - 1 through 7 = FAST → SLOW speed
  - 0 is STOP!!
  - Binary weights: Bit 6 = 4  
Bit 5 = 2  
Bit 4 = 1
  - Switch 7 NDT USED
- FIRST MESSAGE
  - 2650 UP
  - codes for that in USRDSP description Module IV-I; this course.

**DIRECTION:** The Fact Sheet continues on the next page.

- TRAIN CONSIST CODES I wonder how the Inventor came up with the graphic codes for the engines and different cars

AUTHOR'S NOTE:

I played until I figured out what I wanted. You are entitled to the graphics and the codes; that's why they are in the Definition Table on page 3

- TRACK TIES OR TRESTLE Pad with H'97' codes. '97' = space with right justified decimal point.

**DIRECTION:**

Keep the facts provided on this and the previous pages available.

Go right on to the next page.

**DETAILED DIRECTIONS:**

OVERALL, BE AS SPECIFIC AND PRECISE AS POSSIBLE!!!

Prepare your objective spec. as much as possible from the data supplied. You are not expected to write a spec. as detailed as the one which follows. Nor, necessarily, to use the same definition and approaches as I have. In fact, you may specify some approaches which are more effective than those I provide. THAT'S GREAT!!! As long as your spec satisfies the application's requirements, it is on the right track. And, where your comparison shows that it is "derailed", you'll get the assistance you need. And perhaps some ideas you can use later....when you prepare your own application.

When you have completed your preliminary objective spec. for TRAIN, compare your write-up with that provided on pages 7 through 11 of this module. You'll be listening to a short commentary after you make your comparison.

page 6



## PRELIMINARY OBJECTIVE SPECIFICATION

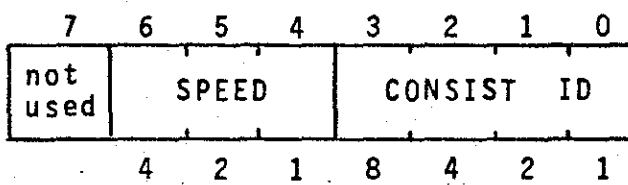
OVERVIEW:

The operator may select any one of 16 consists for display, causing the selected consist to run forward or in reverse at any one of 7 selected speeds. The selected consist 'runs' in the 8-digit Hex Display. Selection of the train is made via the 4 low-order I/O toggle switches, operating in Extended mode. Deletion of the previous consist, and selection of a new train is accomplished by depressing the interrupt key. On entry to the program, the initial message displayed is "2650.UP". The parallel I/O LED's are not used.

The SENS key is depressed to make the train run in reverse, otherwise motion is forward. SPEED is determined by I/O toggle switches 6 through 4. When these switches are off, the selected train is stopped in the current position. Motion is reinitiated by depressing the SENS key. Once the Train is moving, speed can be changed "on the fly" by toggling I/O SW 6-4. Parallel I/O Switch 7 is not used. The FLAG LED is on when the REVERSE motion command (SENS down) is taking place.

SUPPORTING SPECIFICATION:1. EXTENDED I/O PORT DESCRIPTION

Extended I/O Port 7 - Input Switches



SPEED: Switch bits 6,5,&4.

See table on next page.

|   |                          |
|---|--------------------------|
| 7 | 10 MPH                   |
| 6 | 12 "                     |
| 5 | 15 "                     |
| 4 | 19 "                     |
| 3 | 27 "                     |
| 2 | 40 "                     |
| 1 | 80 "                     |
| 0 | Stop in current position |

DIRECTION:

The preliminary objective specification continues on the next page.









## 2. TRAIN SELECTION SPECIFICATION

TRAIN "CONSIST" : SWITCH BITS 3,2,1,&0

ID. = 0 HSE  
 1 HSE + (3)FC  
 2 HSE + FC + (2)EC  
 3 HSE + FC + EC +  $\frac{1}{2}$ FC + C  
 4 HSE + (2)FC + C  
 5 HSE + (3)C  
 6 HSE + (3)EC + C  
 7 LSE + (3)C  
 8 LSE + (2)EC + C  
 9 LSE + (2) $\frac{1}{2}$ FC + C  
 10 LSE + (2)EC + C  
 11 LSE + (3)FC + C  
 12 LSE  
 13 MT  
 14 EM + (4)C  
 15 MT + (3)C

Note: Numbers in parentheses indicate how many of each freight car are in the consist.

### DEFINITIONS:

| STOCK            | GRAPHIC                                                                            | CODE  | DESCRIPTION                                               |
|------------------|------------------------------------------------------------------------------------|-------|-----------------------------------------------------------|
| HSE              |  | 95 E0 | High-speed engine                                         |
| LSE              |  | 95 96 | Low-speed engine                                          |
| MT               |  | C7    | Battery-powered mine tractor                              |
| EM               |  | 8D    | Overhead electric power mine engine                       |
| FC               |  | 8D    | Full freight car                                          |
| $\frac{1}{2}$ FC |  | EE    | half-full freight car                                     |
| EC               |  | 92    | Empty freight car                                         |
| C                |  | 95    | is a Caboose EXCEPT:<br>• in the "mine"<br>• in CONSIST 7 |

## 3. ORGANIZATION OF TRAIN "CONSISTS"

### CONSIST CODE TABLE:

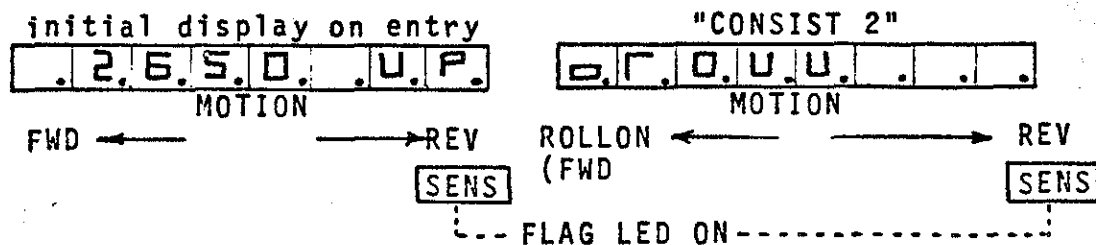
| ID | ADDR | +1 | +2 | +3 | +4 | +5 | +6 | +7 |
|----|------|----|----|----|----|----|----|----|
| 0  | '80' | 95 | E0 | 97 | 97 | 97 | 97 | 97 |
| 1  | '88' | 95 | E0 | 8D | 80 | 80 | 97 | 97 |
| 2  | '90' | 95 | E0 | 80 | 92 | 92 | 97 | 97 |
| 3  | '98' | 95 | E0 | 80 | 92 | EE | 95 | 97 |
| 4  | 'A0' | 95 | E0 | 80 | 80 | 95 | 97 | 97 |
| 5  | 'A8' | 95 | E0 | 95 | 95 | 95 | 97 | 97 |
| 6  | 'B0' | 95 | E0 | 80 | 80 | 80 | 95 | 97 |
| 7  | '88' | 95 | 96 | 95 | 95 | 95 | 97 | 97 |
| 8  | 'C0' | 95 | 96 | 92 | 92 | 95 | 97 | 97 |
| 9  | 'C8' | 95 | 96 | EE | EE | 95 | 97 | 97 |
| A  | 'D0' | 95 | 96 | 92 | 92 | 95 | 97 | 97 |
| B  | 'D8' | 95 | 96 | 80 | 80 | 80 | 95 | 97 |
| C  | 'ED' | 95 | 96 | 97 | 97 | 97 | 97 | 97 |
| D  | 'E8' | C7 | 97 | 97 | 97 | 97 | 97 | 97 |
| E  | 'F0' | 8D | 95 | 95 | 95 | 95 | 97 | 97 |
| F  | 'F8' | C7 | 95 | 95 | 95 | 97 | 97 | 97 |

- 16 x 8-byte table = 128 bytes
- memory addresses: \*\* '00B0' to '00FF'
- Selected contents may be fetched but not altered.
- See Part 6. of spec.

\*\* Code and addresses will be specified in the author's solution

## 4. 8-DIGIT DISPLAY DESCRIPTION

- Access via MONITOR's "USRDSP" routine
- Typical Displays:



- Initial display: "2650 UP" is replaced when first selected consist is displayed

## 5. PROPOSED FORWARD AND REVERSE MOTION -- CONCEPT OF SELECTED CONSIST ACCESS AND MOVEMENT

"SWYRDINT":

EXT I/O SWITCHES X8 + H'80' = \*CONADDR (LOW ORDER) IN LOC '1783'  
 (LOW ORDER 4) \*CONADDR (HI ORDER) IN LOC '1782' = 00

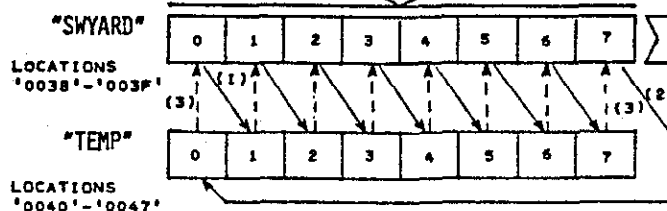
← \*CONADDR\*(R1) INDEX 8 TIMES TO EXTRACT CONSIST

"CONSIST" STORAGE - 128 BYTES

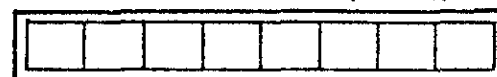
| CONSIST ID0 | '00F8' |
|-------------|--------|
| " " 1       | 88     |
| " " 2       | 90     |
| " " 3       | 98     |
| " " 4       | A0     |
| " " 5       | AB     |
| " " 6       | B0     |
| " " 7       | B8     |
| " " 8       | C0     |
| " " 9       | CB     |
| " " A       | DD     |
| " " B       | DB     |
| " " C       | E0     |
| " " D       | EB     |
| " " E       | F0     |
| " " F       | '00B0' |

CONSIST 10X START ADDR →

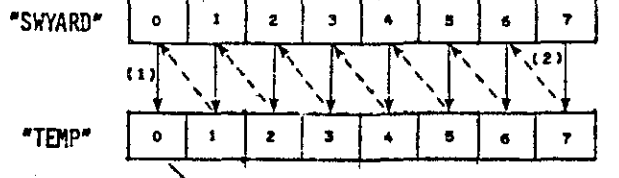
MEMORY TO SWYARD VIA R0



"INSTRUCTOR" HEX DISPLAY (TYPICAL)



SELECTED TRAIN  
TO DISPLAY FROM  
SWYARD BY EXECUT-  
ING 'USRDSP'



"ROLLON"

(1) 8 BYTES SWYARD → TEMP  
 (2) 7 BYTES TEMP → SWYARD  
 (3) TEMPO → SWYARD?

**DIRECTION:**

The preliminary objective specification continues on the next page.



8. PROPOSED MEMORY MAPAUTHOR'S NOTE:

In normal objective spec. preparation, a memory map as detailed as this is never included. Simply because the program is still only an idea. However, some proposed memory constants may be defined - even at this point.

## MEMORY MAP

|      |                                               |
|------|-----------------------------------------------|
| 0000 | DISPLAY                                       |
| 0007 | IND. INT. ADDRESS                             |
|      | ROLLON<br>FWD DIRECTION                       |
|      | SWYARD                                        |
|      | TEMP                                          |
|      | CHKSENS<br>REV. OR FWD.                       |
|      | REVERSE<br>REV DIRECTION                      |
| 0071 | SWYARDINT<br>select 1 of 16<br>train consists |
| 0080 | CONSIST<br>16 x 8 byte<br>train selection     |
| 00FF | matrix                                        |
|      | SWYARDINT<br>(cont.)                          |
|      | UNUSED<br>USER<br>MEMORY                      |
| D1FF |                                               |

NON EXISTANT MEMORY LDCS.

|      |                                                                          |
|------|--------------------------------------------------------------------------|
| 1780 | SMI USER<br>AVAILABLE<br>RAM                                             |
| 17BF |                                                                          |
| 17CD | MONITOR<br>SCRATCHPAD<br>AND USER<br>SYSTEM EX-<br>ECUTIVE ROU-<br>TINES |
| 1FFF |                                                                          |

DIRECTION:

Listen to Tape 9A for discussion of the Objective Specification for "TRAIN", with documentation open to page 7 of this module.

## PREPARE THE FLOWCHART

INTRODUCTION:

In this step, while working from details provided in the Preliminary Objective specification, you'll prepare a detailed flowchart of the probable sequence to be programmed. The flowchart is no supposed to be an 'instruction-by-instruction' detailed analysis. Think of each major function to be programmed, then chart its overall sequence.

For example: A half-second delay might be flowcharted with a block containing the words: .....

|                            |
|----------------------------|
| DELAY<br>$\frac{1}{2}$ SEC |
|----------------------------|

As an option, you CAN supplement your flowchart with other comments, such as your proposed use of registers and specific memory location to accomplish a given function.

For example: A supplementary comment to .....  
 the "DELAY  $\frac{1}{2}$  SEC" block :2 regs initialize, then  
 might be:..... :decrement, and loop un-  
 :til delay is complete

DIRECTIONS:

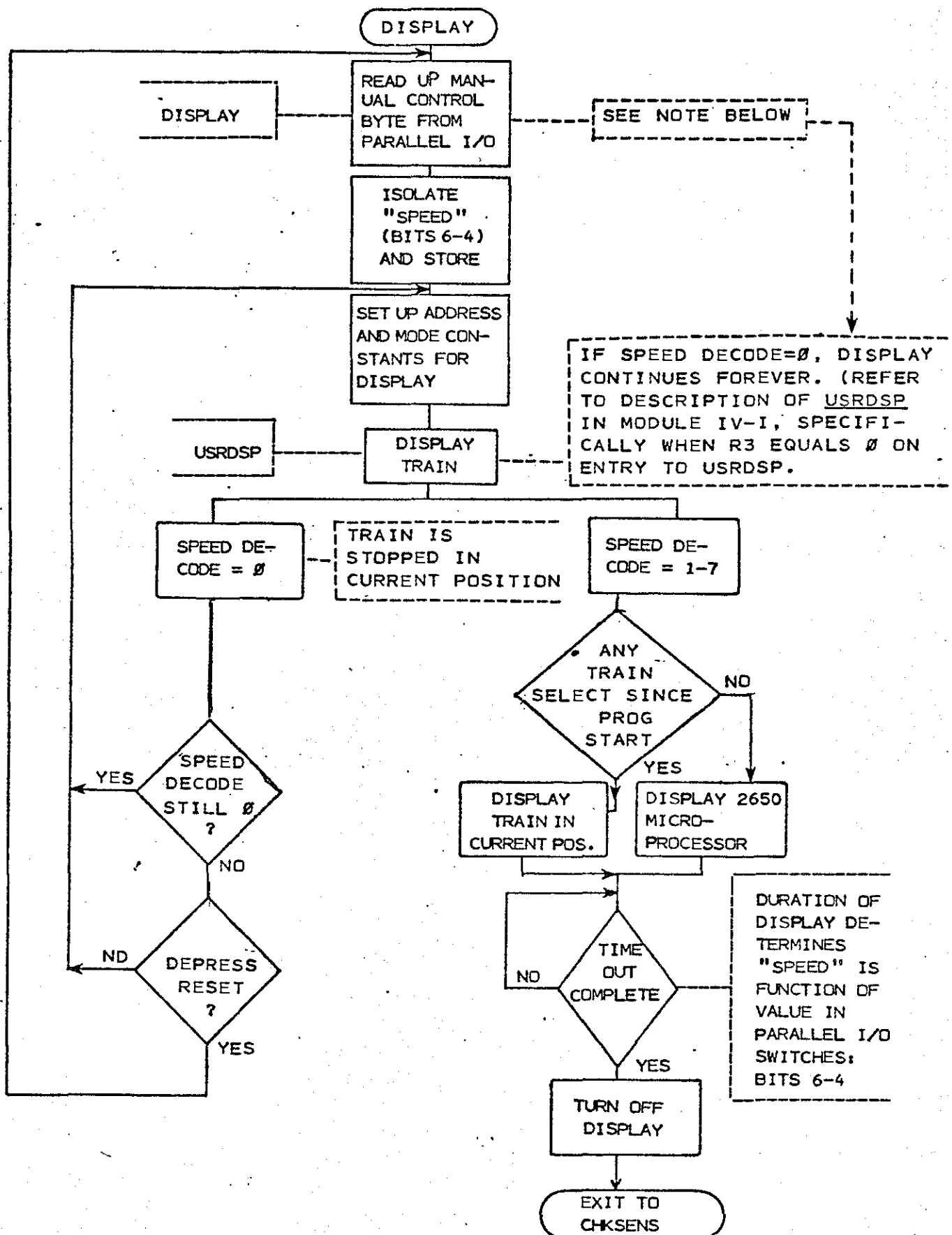
- A. If you are not familiar with flowcharting a logical sequence, including the use of basic symbols involved, don't try to write the flowchart at this time. Instead, make a thorough study of the proposed flowchart contained on the next 3 pages. After studying the flowcharts, listen to the appropriate tape for further commentary.
- B. If you are familiar with flowcharting logical sequences, and the symbols used for this purpose, prepare your flowchart from the details contained in the Objective Specification. Then, compare your work to the flowchart proposed in the next 3 pages. Then, listen to the short commentary on tape 9A, as referenced to the proposed flowchart.

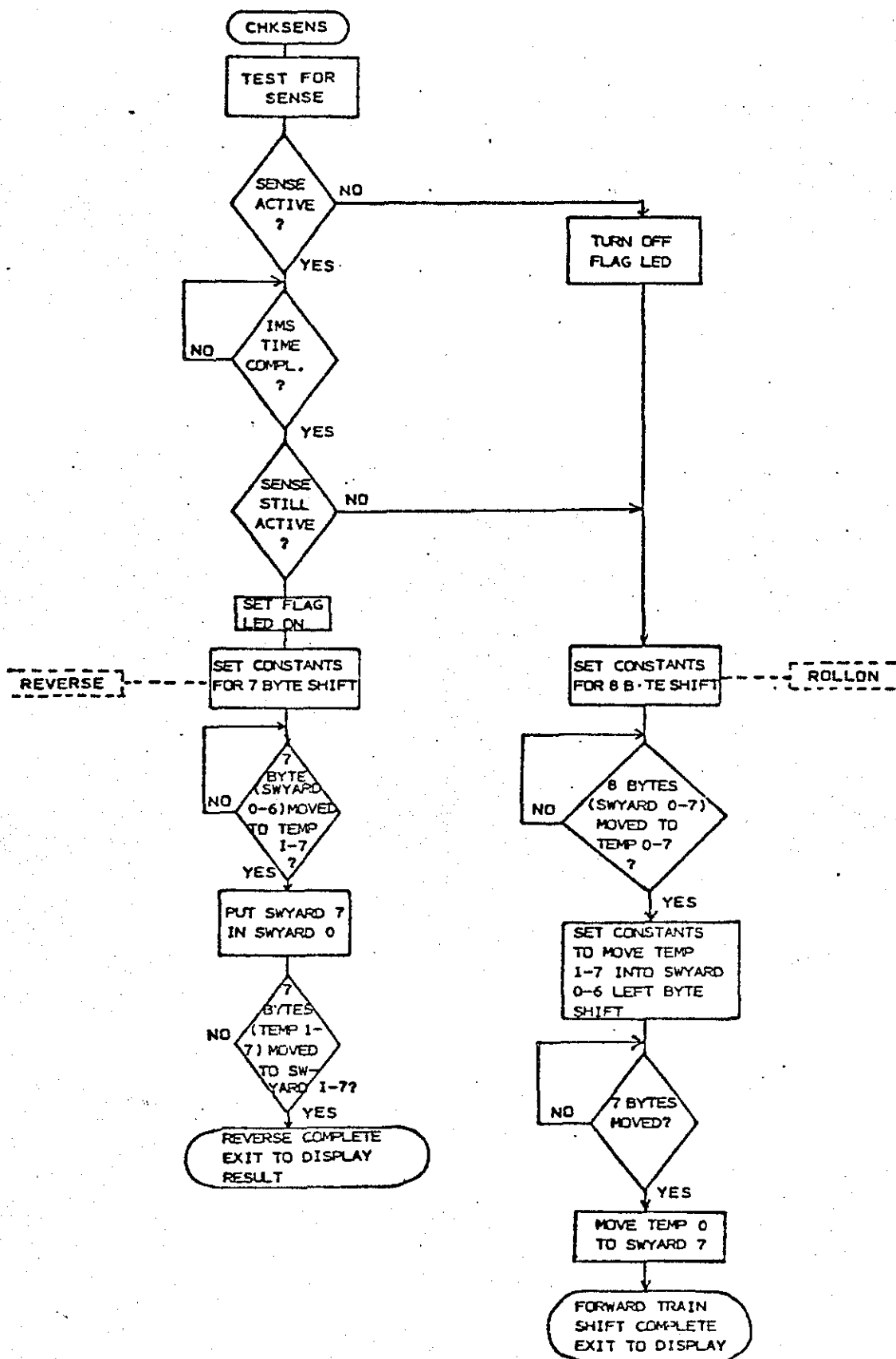
AUTHOR'S NOTE:

Although this direction references the printed Objective Spec., you are encouraged to write the flow for each proposed routine based on the preliminary Objective Spec. YOU prepared in Step 2.

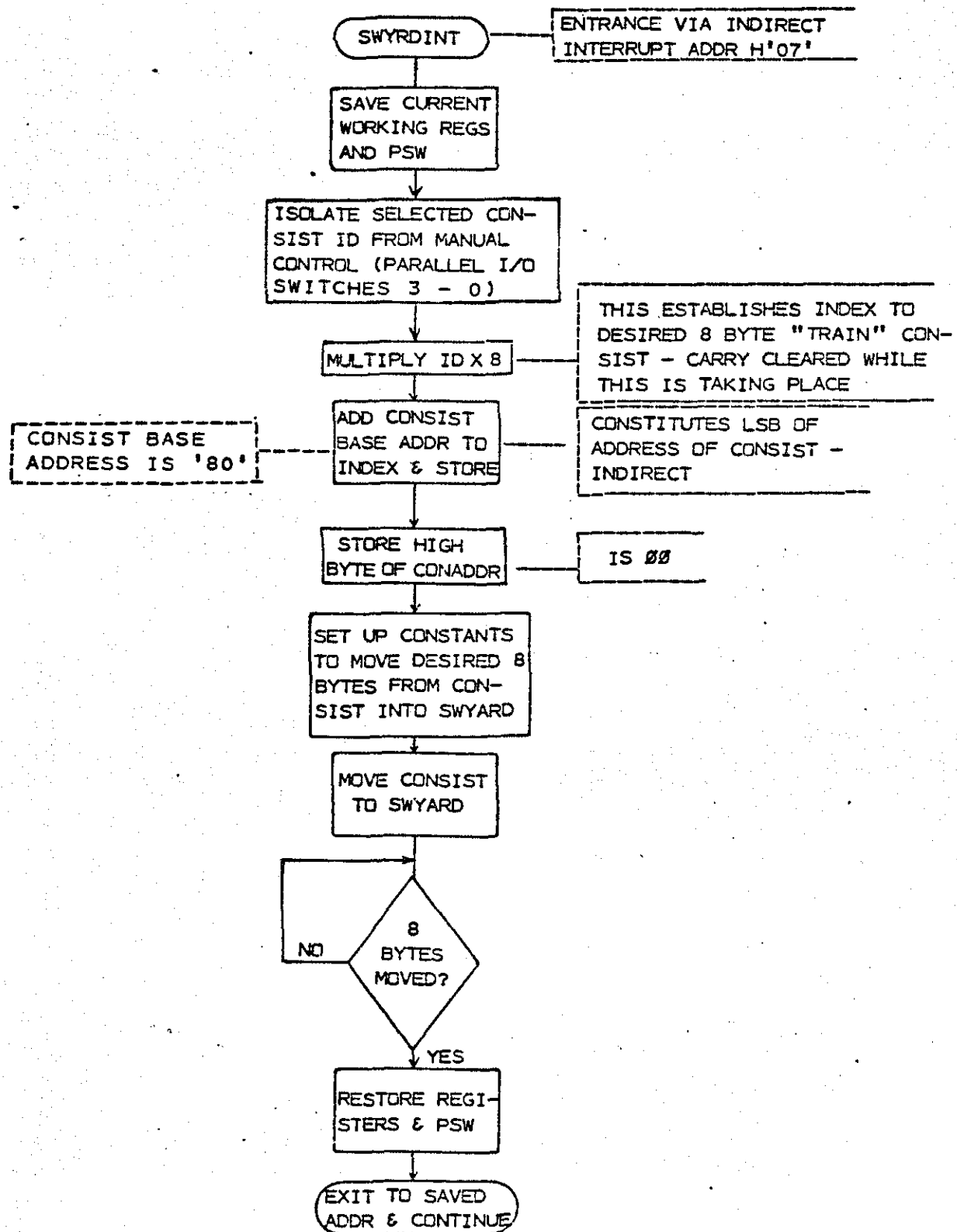
SSSSSSSSSSSSSSSSSS

## PROPOSED FLOWCHART - PROGRAM SEQUENCE



PROPOSED FLOWCHART - PROGRAM SEQUENCE (CONTINUED)



PROPOSED FLOWCHART - PROGRAM SEQUENCE (CONTINUED)**DIRECTION:**

After studying this flowchart, listen to tape 9A for further commentary. The commentary references page 12 first.

## OPERATING DIRECTIONS

- Initial Startup:
- On-board Switch Settings:
  - I/O Selection: EXTENDED I/O PORT 7
  - Interrupt Address: INDIRECT
  - Realtime/Keyboard Int.: KEYBOARD
  - I/O Input Toggle Switches:
    - Bit 7: OFF
    - Bit 6: ON
    - Bits 5 through 0: 'don't care'

- Depress RST to start.  
Message: 2650.U.P. is displayed

- To select 1 of 16 consists:
  - I/O Input Toggle Switches:
  - Bits 3 through 0: ON or OFF
  - per table below:

|    |                  |  |
|----|------------------|--|
| 1  | HSE+(3)FC        |  |
| 2  | HSE+FC+(2)EC     |  |
| 3  | HSE+FC=EC+1/2F+C |  |
| 4  | HSE+(2)FC+C      |  |
| 5  | HSE+(3)C         |  |
| 6  | HSE+(3)EC+C      |  |
| 7  | LSE+(3)C         |  |
| 8  | LSE+(2)EC+C      |  |
| 9  | LSE+(2)1/2F+C    |  |
| 10 | LSE+(2)EC+C      |  |
| 11 | LSE+(3)FC+C      |  |
| 12 | LSE              |  |
| 13 | MT               |  |
| 14 | EM+(4)C          |  |
| 15 | MT+(3)C          |  |

### DEFINITIONS

|       |                                                                                                         |
|-------|---------------------------------------------------------------------------------------------------------|
| HSE   | HI SPEED ENGINE                                                                                         |
| LSE   | LO SPEED ENGINE                                                                                         |
| MT    | BATTER POWERED<br>MINE TRACTOR                                                                          |
| EM    | OVERHEAD ELECTRIC<br>POWERED MINE ENGINE                                                                |
| FC    | FREIGHT CAR - FULL                                                                                      |
| 1/2FC | FREIGHT CAR - 1/2 FULL                                                                                  |
| EC    | FREIGHT CAR - EMPTY                                                                                     |
| C     | CABOOSE (IF USED<br>WITH HS OR LS ENGINE)<br>COAL CAR (IF IN THE<br>MINE OR WHEN CONSIST<br>7 SELECTED) |

- SPEED SELECTION: per table in Obj. Spec.
- To select consist when Train is stopped:

1. SELECT CONSIST I.D.
2. DEPRESS INT
3. IF DISPLAY DOES NOT CHANGE, DEPRESS ANY FUNCTION KEY. NEW TRAIN WILL DISPLAY LEFT JUSTIFIED. CONTINUE WITH STEP 4.  
IF DISPLAY CHANGES (SINGLE DIGIT OR RANDOM DECIMAL POINT), CONTINUE AT STEP 6.
4. SET SPEED 1-7
5. DEPRESS RESET. TRAIN WILL MOVE FORWARD AND REVERSE WHEN SENSE IS DEPRESSED.
6. HIT RESET TO DISPLAY SELECTED TRAIN.
7. SET PC=0. AN UNEXPECTED DISPLAY AT STEP 2 INDICATES THE P.C. IS POINTING TO SOME UNEXPECTED ADDRESS.
8. HIT RESET.

**INTRODUCTION:**

When you write the process to restore the registers and the PSW (last function performed by routine "SWYARDINT"), refer to page 4 of the listing. Restoration of the PSW(LOWER) containing the saved condition code, is somewhat tricky. Essentially, the PSL must be restored WITHOUT wiping out the saved condition code during the process of restoration.

NOTE: Ensure that all control program storage locations have known data in them as if you were going to prepare a ROM Program Tape.

SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

After you have finished writing, debugging, and checking out your program, go roigh on to Module IV-K (EXTRA PROGRAMS & IDEAS), then into the study of Module V (INTERFACE).

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| +                                        | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E | 1F |
| -                                        | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E | 2F |
| +                                        | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E | 3F |
| +                                        | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
| +                                        | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 5A | 5B | 5C | 5D | 5E | 5F |
| +                                        | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 6A | 6B | 6C | 6D | 6E | 6F |
| +                                        | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F |
| +                                        | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F |
| +                                        | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F |
| +                                        | A0 | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | AA | AB | AC | AD | AE | AF |
| +                                        | B0 | B1 | B2 | B3 | B4 | B5 | B6 | B7 | B8 | B9 | BA | BB | BC | BD | BE | BF |
| +                                        | C0 | C1 | C2 | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF |
| +                                        | D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | DA | DB | DC | DD | DE | DF |
| +                                        | E0 | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | EA | EB | EC | ED | EE | EF |
| +                                        | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | FA | FB | FC | FD | FE | FF |

INDIRECT RELATIVE ADDRESS: Add H'80' to displacement

INDIRECT RELATIVE ADDRESS: Add H'80' TO DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE TRAIN @ START ADDR'S '00'

DESCRIPTION This program permits selection of any 1 of 16 "trains", and any 1 of 7 speeds, forward or reverse plus stop and re-selection. Ext. I/O sw. select: EXT.

ROUTINE SHEET 1 OF 6

MEMORY LOCATIONS THIS SHEET \_\_\_\_\_

|    | ADDRES | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |                | COMMENT                      |
|----|--------|------|----|----|---------|----------------------|----------------|------------------------------|
|    |        | B0   | B1 | B2 |         | OPCODE               | OPERANDS       |                              |
| 1  | 0000   | 57   | 07 |    | DISPLAY | REDE, R3             | PORT 7         | get manual control byte &    |
| 2  | 2      | 47   | 70 |    |         | ANDI, R3             | H'70'          | isolate speed factor, then   |
| 3  | 4      | 1B   | 03 |    |         | BCTR, UN             | INTFREE        | branch past int. addr.       |
| 4  | 6      | XX   |    |    | HOLD    | RES                  | 1              | Temporary speed storage      |
| 5  | 7      | 00   | 71 |    | SWYDIN  | ACON                 | * SWYRDINT     | Switch Yard intpt. Address   |
| 6  | 9      | CB   | 7B |    | INTFREE | STRR, R3             | HOLD           | Put speed factor in "HOLD" & |
| 7  | 8      | 05   | 00 |    |         | LODI, R1             | <SWYARD-1      | then set up SWYARD           |
| 8  | D      | 06   | 37 |    |         | LODI, R2             | >SWYARD-1      | Address (-1) and display     |
| 9  | 000F   | BB   | E6 |    |         | ZBSR                 | * USRDSP       | selected train. Now, get     |
| 10 | 11     | 0B   | 73 |    |         | LODR, R3             | HOLD           | speed factor & decrement.    |
| 11 | 3      | FB   | 74 |    |         | BDRR, R3             | INTFREE        | display over? No! Do again.  |
| 12 | 5      | 1B   | 31 |    |         | BCTR, UN             | CHKSENS        | Yes! go read direction.      |
| 13 |        |      |    |    |         |                      |                |                              |
| 14 | 7      | C0   |    |    |         | NOP                  |                | (filler instr.)              |
| 15 | 0018   | 05   | 00 |    | ROLLON  | LODI, R1             | 0              | Initialize SWYARD index &    |
| 16 | A      | 07   | 08 |    |         | LODI, R3             | 8              | counter, then put SWYARD     |
| 17 | C      | 0D   | 60 | 38 | REPEAT1 | LODA, R0             | SWYARD, R1     | in TEMP.                     |
| 18 | 1F     | CD   | 20 | 3F |         | STRA, R0             | TEMP-1, R1, +  | Done yet? No! Move           |
| 19 | 22     | FB   | 78 |    |         | BDRR, R3             | REPEAT1        | another byte. Yes, set       |
| 20 | 4      | 05   | 08 |    |         | LODI, R1             | 8              | up for shift (byte) and      |
| 21 | 6      | 0D   | 60 | 3F | REPEAT2 | LODA, R0             | TEMP-1, R1     | move TEMP → SWYARD           |
| 22 | 9      | CD   | 40 | 37 |         | STRA, R0             | SWYARD-1, R1 - | shifting 1 byte left (fwd)   |
| 23 | C      | 59   | 78 |    |         | BRNR, R1             | REPEAT2        | Done? No! Finish. Yes!       |
| 24 | 2E     | 08   | 10 |    |         | LODR, R0             | TEMP           | Move MSB of TEMP to LSB      |
| 25 | 30     | C8   | 0D |    |         | STRR, R0             | SWYARD+7       | of SWYARD, then go back      |
| 26 | 0032   | 1B   | 4C |    |         | BCTR, UN             | DISPLAY        | & display.                   |
| 27 |        |      |    |    |         |                      |                |                              |

|    | ADDRS | DATA |    |    | LABEL    | SYMBOLIC INSTRUCTION                             |             | COMMENT                        |
|----|-------|------|----|----|----------|--------------------------------------------------|-------------|--------------------------------|
|    |       | B0   | B1 | B2 |          | OPCODE                                           | OPERANDS    |                                |
| 1  |       |      |    |    | SWYARD   | ACON                                             | H'003B'     |                                |
| 2  | 003B  | 97   | 02 | 06 | * SWYARD | is an 8-BYTE data table                          |             | Original message at            |
| 3  | 3     | 05   | 00 | 97 | SWYARD   | RES                                              | 8           | start-up:                      |
| 4  | E     | 12   | 10 |    |          |                                                  |             | ". 2650 UP."                   |
| 5  |       |      |    |    | TEMP     | ACON                                             | H'0040'     |                                |
| 6  | 0040  | XX   | XX | XX | * TEMP   | is 8-byte temporary data table for SWYARD motion |             |                                |
| 7  | 3     | XX   | XX | XX | * AND    | DIRECTION                                        | control     |                                |
| 8  | 6     | XX   | XX |    | TEMP     | RES                                              | 8           |                                |
| 9  |       |      |    |    |          |                                                  |             |                                |
| 10 | 0048  | B4   | 80 |    | CHKSENS  | TPSU                                             | SENS        | Test for REVERSE (SENS KEY     |
| 11 | A     | 18   | 04 |    |          | BCTR, EQ                                         | DELAY       | down.) could be; go find out   |
| 12 | C     | 74   | 40 |    |          | CPSU                                             | FLAG        | It wasn't. clean flag like I   |
| 13 | 4E    | 1B   | 48 |    |          | BCTR, UN                                         | ROLLON      | exit to ROLLON (fwd).          |
| 14 | 50    | D8   | 7E |    | DELAY    | BIRR, RO                                         | \$          | Set 1msec delay and chr        |
| 15 | 2     | B4   | 80 |    |          | TPSU                                             | SENS        | if SENS depressed deliberately |
| 16 | 4     | 98   | 42 |    |          | BCTR, EQ                                         | ROLLON      | It wasn't! Go fwd.             |
| 17 | 6     | 76   | 40 |    | REVERSE  | PPSU                                             | FLAG        | It was, set flag LED on        |
| 18 | 8     | C1   |    |    |          | STRZ                                             | R1          | and set constants to reverse   |
| 19 | 9     | 07   | F9 |    |          | LODI, R3                                         | H'F9'       | direction. Now, move           |
| 20 | B     | 0D   | 60 | 38 | LOOP1    | LODA, RO                                         | SWYARD, R1  | SWYARD → TEMP. This time       |
| 21 | 5E    | CD   | 20 | 40 |          | STRA, RO                                         | TEMP, R1, + | byte shift right. 7 bytes      |
| 22 | 61    | DB   | 78 |    |          | BIRR, R3                                         | LOOP1       | moved? NO! Finish.             |
| 23 | 3     | 08   | 5A |    |          | LODR, RO                                         | SWYARD + 7  | Yes! Move LS SWYARD in.        |
| 24 | 5     | C8   | 51 |    |          | STRR, RO                                         | SWYARD      | to M.S. TEMP then move         |
| 25 | 7     | 0D   | 60 | 40 | LOOP2    | LODA, RO                                         | TEMP, R1    | all of temp back into SW-      |
| 26 | 006A  | CD   | 60 | 38 |          | STRA, RO                                         | SWYARD, R1  | YARD.... No shift.             |
| 27 |       |      |    |    |          |                                                  |             |                                |

|    | ADDRS    | DATA |    |    | LABEL    | SYMBOLIC INSTRUCTION                             |             | COMMENT                     |
|----|----------|------|----|----|----------|--------------------------------------------------|-------------|-----------------------------|
|    |          | B0   | B1 | B2 |          | OPCODE                                           | OPERANDS    |                             |
| 1  |          |      |    |    |          |                                                  |             | Is all of TEMP moved to SW- |
| 2  | 006D     | F9   | 78 |    |          | BDRR, R1                                         | LOOP2       | YARD yet? No! Move another  |
| 3  | F        | 9B   | 00 |    |          | ZBRR                                             | DISPLAY     | byte. Yea! exit to display  |
| 4  |          |      |    |    |          |                                                  |             | the "train".                |
| 5  |          |      |    |    |          | * 1 <sup>st</sup> PART OF SWYRDINT IS ROUTINE TO |             | SAVE R0-R3, ALSO THE        |
| 6  |          |      |    |    |          | * PSW (UPPER & LOWER) IN CASE THEY ARE           |             | NEEDED WHEN INTER-          |
| 7  |          |      |    |    |          | * RUPT SERVICE IS COMPLETE                       |             |                             |
| 8  | 0071     | CC   | 17 | 90 | SWYRDINT | STRA, R0                                         | SAVEREG     | Save Regist. 0 (loc '1790'  |
| 9  | 4        | CC   | 17 | 91 |          | STRA, R1                                         | SAVEREG + 1 | " " 1 '1791'                |
| 10 | 7        | CC   | 17 | 92 |          | STRA, R2                                         | SAVEREG + 2 | " " 2 '1792'                |
| 11 | A        | CC   | 17 | 93 |          | STRA, R3                                         | SAVEREG + 3 | " " 3 '1793'                |
| 12 | 007D     | 1F   | 01 | 00 |          | BCTA, UN                                         | CONTINT     | And continue the 'SAVE'     |
| 13 | 0080.... | XX   | XX | XX |          | * 128-BYTE Table (16x8) RESERVED                 |             | FOR TRAIN CONSIST SE-       |
| 14 | 00FF     | XX   | XX | XX |          | * LECTON                                         |             |                             |
| 15 |          |      |    |    | CONSIST  | ACON                                             | H '0080     | see sheet 5 of listing      |
| 16 |          |      |    |    |          |                                                  |             |                             |
| 17 | 0100     | 13   |    |    | CONTINT  | SPSL                                             |             | Now, save the PSW           |
| 18 | 1        | CC   | 17 | 94 |          | STRA, R0                                         | SAVEREG + 4 | (lower) at loc '1794'       |
| 19 | 4        | 12   |    |    |          | SPS U                                            |             | and save PSW (upper         |
| 20 | 5        | CC   | 17 | 95 |          | STRA, R0                                         | SAVEREG + 5 | in loc '1795'               |
| 21 | 8        | 75   | 0B |    |          | CPSL                                             | WC          | Clear WC bit, then read     |
| 22 | A        | 54   | 07 |    |          | REDE, R0                                         | PORT7       | up MANUAL CONTROL and       |
| 23 | C        | 44   | 0F |    |          | ANDI, R0                                         | H '0F'      | isolate the CONSIST I.D.    |
| 24 | E        | D0   |    |    |          | RRL                                              | R0          | Now, multiply I.D. by 8     |
| 25 | 10F      | D0   |    |    |          | RRL                                              | R0          | to get an 8-byte            |
| 26 | 110      | D0   |    |    |          | RRL                                              | R0          | index, then restore         |
| 27 | 0111     | 77   | 0B |    |          | PPSL                                             | WC          | the WC bit for future use   |

| 1  | ADDRES | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |                 | COMMENT                   |
|----|--------|------|----|----|---------|----------------------|-----------------|---------------------------|
|    |        | B0   | B1 | B2 |         | CPCODE               | OPERANDS        |                           |
| 1  |        |      |    |    | CONADDR | ACON                 | H'1783          | Now, add consist base ad- |
| 2  | 0113   | 84   | 80 |    |         | ADDI, R0             | H'80'           | dress to index (low) byte |
| 3  | 5      | CC   | 17 | 83 |         | STRA, R0             | CONADDR+1       | and store. Then clear     |
| 4  | 8      | 20   |    |    |         | EORZ                 | R0              | R0 and store as hi-byte   |
| 5  | 9      | CC   | 17 | 82 |         | STRA, R0             | CONADDR         | of addr. Now, set cons-   |
| 6  | C      | C1   |    |    |         | STAZ                 | R1              | stants to moved selected  |
| 7  | D      | 07   | 08 |    |         | LODI, R3             | 8               | consist (1 of 16 trains)  |
| 8  | 11F    | 0D   | F7 | 82 | LOOP3   | LODA, R0             | *CONADDR, R1    | into the swyard and       |
| 9  | 122    | CD   | 20 | 37 |         | STRA, R0             | SWYARD-1, R1, + | move it. Done yet?        |
| 10 | 5      | FB   | 78 |    |         | BDRR, R3             | LOOP3           | No! finish the transfer.  |
| 11 |        |      |    |    | RECALL  | ACON                 | H'1798          | (res. 3 bytes)            |
| 12 | 7      | 0C   | 17 | 95 | RESTORE | LODA, R0             | SAVEREG+5       | Yes! restore previously   |
| 13 | A      | 92   |    |    |         | LPSU                 |                 | saved PSW (upper)         |
| 14 | 8      | 0D   | 17 | 91 |         | LODA, R1             | SAVEREG+1       | also R1                   |
| 15 | 12E    | 0E   | 17 | 92 |         | LODA, R2             | SAVEREG+2       | and R2                    |
| 16 | 131    | 0F   | 17 | 93 |         | LODA, R3             | SAVEREG+3       | and R3, then build        |
| 17 | 9      | 04   | 37 |    |         | LODI, R0             | RETE            | a subroutine to restore   |
| 18 | 6      | CC   | 17 | 9A |         | STRA, R0             | RECALL+2        | previously saved PSL.     |
| 19 | 9      | 0C   | 17 | 94 |         | LODA, R0             | SAVEREG+4       | this routine is "RECALL"  |
| 20 | C      | CC   | 17 | 99 |         | STRA, R0             | RECALL+1        | with contents             |
| 21 | 13F    | 04   | 77 |    |         | LODI, R0             | PPSL            | Addr '1798' PPSL          |
| 22 | 141    | CC   | 17 | 98 |         | STRA, R0             | RECALL          | 1799 PSL (saved)          |
| 23 | 4      | 0C   | 17 | 90 |         | LODA, R0             | SAVEREG         | 179A RETE                 |
| 24 | 7      | 75   | FF |    |         | CPSL                 | H'FF'           | Now, clear the current    |
| 25 | 9      | 1F   | 17 | 98 |         | BCTA, UN             | RECALL          | Program Status (lower)    |
| 26 |        |      |    |    |         |                      |                 | and get "RECALL" to       |
| 27 |        |      |    |    |         |                      |                 | return                    |

|    | ADDRS | DATA |    |    | LABEL | SYMBOLIC INSTRUCTION |          | COMMENT                             |
|----|-------|------|----|----|-------|----------------------|----------|-------------------------------------|
|    |       | B0   | B1 | B2 |       | OPCODE               | OPERANDS |                                     |
| 1  | 0080  | 95   | E0 | 97 | ID 0  | RES                  | 8        | CONSIST 0                           |
| 2  |       | 97   | 97 | 97 |       |                      |          | High Speed Engine only              |
| 3  |       | 97   | 97 |    |       |                      |          |                                     |
| 4  | 88    | 95   | E0 | 80 | ID 1  | RES                  | 8        | CONSIST 1                           |
| 5  |       | 80   | 80 | 97 |       |                      |          | H.S. Engine + 3 full                |
| 6  |       | 97   | 97 |    |       |                      |          | freight cars (FC)                   |
| 7  | 90    | 95   | E0 | 80 | ID 2  | RES                  | 8        | CONSIST 2                           |
| 8  |       | 80   | 80 | 97 |       |                      |          | HSE + 1 FC + 2 Empty                |
| 9  |       | 97   | 97 |    |       |                      |          | cars (EC)                           |
| 10 | 98    | 95   | E0 | 80 | ID 3  | RES                  | 8        | CONSIST 3                           |
| 11 |       | 92   | EE | 95 |       |                      |          | HSE + 1 FC + 1 EC + 1               |
| 12 |       | 97   | 97 |    |       |                      |          | half full car (1/2) + 1 caboose (C) |
| 13 | A0    | 95   | E0 | 80 | ID 4  | RES                  | 8        | CONSIST 4                           |
| 14 |       | 80   | 95 | 97 |       |                      |          | HSE + 2 FC + 1 C                    |
| 15 |       | 97   | 97 |    |       |                      |          |                                     |
| 16 | A8    | 95   | E0 | 95 | ID 5  | RES                  | 8        | CONSIST 5                           |
| 17 |       | 95   | 95 | 97 |       |                      |          | HSE + 3 C                           |
| 18 |       | 97   | 97 |    |       |                      |          |                                     |
| 19 | B0    | 95   | E0 | 80 | ID 6  | RES                  | 8        | CONSIST 6                           |
| 20 |       | 80   | 80 | 95 |       |                      |          | HSE + 3 EC + C                      |
| 21 |       | 97   | 97 |    |       |                      |          |                                     |
| 22 | B8    | 95   | 96 | 95 | ID 7  | RES                  | 8        | CONSIST 7                           |
| 23 |       | 95   | 95 | 97 |       |                      |          | Low speed engine (LSE)              |
| 24 |       | 97   | 97 |    |       |                      |          | plus 3 C                            |
| 25 | C0    | 95   | 96 | 92 | ID 8  | RES                  | 8        | CONSIST 8                           |
| 26 |       | 92   | 95 | 97 |       |                      |          | LSE + 2 EC + C                      |
| 27 |       | 97   |    |    |       |                      |          |                                     |



|    | ADDRS | DATA |    |    | LABEL                                                  | SYMBOLIC INSTRUCTION |          | COMMENT               |
|----|-------|------|----|----|--------------------------------------------------------|----------------------|----------|-----------------------|
|    |       | B0   | B1 | B2 |                                                        | OPCODE               | OPERANDS |                       |
| 1  | 00C8  | 95   | 96 | EE | ID 9                                                   | RES                  | 8        | CONSIST 9             |
| 2  |       | EE   | 95 | 97 |                                                        |                      |          | LSE + 2 1/2 + C       |
| 3  |       | 97   | 97 |    |                                                        |                      |          |                       |
| 4  | D0    | 95   | 96 | 92 | ID A                                                   | RES                  | 8        | CONSIST 10            |
| 5  |       | 92   | 95 | 97 |                                                        |                      |          | LSE + 2 EC + C        |
| 6  |       | 97   | 97 |    |                                                        |                      |          |                       |
| 7  | D8    | 95   | 96 | 80 | ID B                                                   | RES                  | 8        | CONSIST 11            |
| 8  |       | 80   | 80 | 95 |                                                        |                      |          | LSE + 3 FC + C        |
| 9  |       | 97   | 97 |    |                                                        |                      |          |                       |
| 10 | E0    | 95   | 96 | 97 | ID C                                                   | RES                  | 8        | CONSIST 12            |
| 11 |       | 97   | 97 | 97 |                                                        |                      |          | LSE (alone)           |
| 12 |       | 97   | 97 |    |                                                        |                      |          |                       |
| 13 | E8    | C7   | 97 | 97 | ID D                                                   | RES                  | 8        | CONSIST 13            |
| 14 |       | 97   | 97 | 97 |                                                        |                      |          | MINE engine (M) alone |
| 15 |       | 97   | 97 |    |                                                        |                      |          |                       |
| 16 | F0    | 8D   | 95 | 95 | ID E                                                   | RES                  | 8        | CONSIST 14            |
| 17 |       | 95   | 95 | 97 |                                                        |                      |          | Electric M + 4 C      |
| 18 |       | 97   | 97 |    |                                                        |                      |          |                       |
| 19 | F8    | C7   | 95 | 95 | ID F                                                   | RES                  | 8        | CONSIST 15            |
| 20 |       | 95   | 97 | 97 |                                                        |                      |          | M + 3 C               |
| 21 |       | 97   | 97 |    |                                                        |                      |          |                       |
| 22 |       |      |    |    | * AT START OF PROGRAM, message: "2650 UP" is displayed |                      |          |                       |
| 23 |       |      |    |    | * "2650 UP" is lost upon selection of any "train" via  |                      |          |                       |
| 24 |       |      |    |    | * INTERRUPT key depression.                            |                      |          |                       |
| 25 |       |      |    |    | * For different display, change any consist of 8 cha-  |                      |          |                       |
| 26 |       |      |    |    | * racters and select appropriate                       |                      |          | CONSIST I.D. THRU     |
| 27 |       |      |    |    | * parallel I/O switches.                               |                      |          |                       |



## 2650 MICROPROCESSOR COURSE

### MODULE IV - K

#### EXTRA PROGRAMS AND IDEAS

**PREPARED BY:**

MICROPROCESSOR TRAINING DEPARTMENT  
Signetics Corporation  
811 E. Arques Ave.  
Sunnyvale, CA, 94086

**REFERENCE:**

Outlines and Abstracts  
page 24

INTRODUCTION:

This section contains several ideas for INSTRUCTOR-based programs, also several routines and programs in various stages of completion. There is no formal instruction on this material since it is intended for your use and practice outside the framework of this course.

The course continues in Module V; "INTERFACE", starting on page 1.

\* \* \* I N D E X \* \* \*EXTRA PROGRAMS AND IDEAS:

|                                                                                        |    |
|----------------------------------------------------------------------------------------|----|
| CLEAR MEMORY (3 Variations)                                                            | 2  |
| BUBBLE SORT (program only - not commented)                                             | 5  |
| BEAT THE ODDS (specification and program)                                              | 6  |
| " " " - VARIATION: (with program)                                                      | 10 |
| SHOOTING GALLERY - IDEA                                                                | 12 |
| REACTION - IDEA                                                                        | 12 |
| SLOT MACHINE - IDEA                                                                    | 13 |
| BINGO - IDEA                                                                           | 14 |
| KENO - IDEA                                                                            | 15 |
| HALT1 (Demonstration - Reentry to process on<br>RESET & INTERRUPT - a working program) | 16 |
| ROUTINE RELOCATION (4 variations with exercise and<br>programs)                        | 17 |
| MEMORY PATTERN ENTRY                                                                   | 22 |

FORMAL APPLICATION NOTES and MEMOS:

|         |                                                                   |     |
|---------|-------------------------------------------------------------------|-----|
| • MP-51 | 2650 Initialization                                               | 24  |
| • AS-51 | Bit and Byte Testing Procedures                                   | 25  |
| • AS-52 | General Delay Routines                                            | 28  |
| • AS-56 | Sorting Routines                                                  | 30  |
| • AS-53 | Binary Arithmetic Routines                                        | 46  |
| • AS-55 | Fixed-point Decimal Arithmetic Routines                           | 61  |
| • AS-54 | Conversion Routines                                               | 79  |
| • MP-53 | Address and Data Bus Interfacing Techniques                       | 97  |
| • AS-50 | Serial Input/Output                                               | 105 |
| • -     | Interface Design with the Signetics 2650                          | 108 |
| • -     | DAC and ADC to Microprocessor Interface<br>Techniques             | 123 |
| o       | Analog to Digital Conversion - Successive<br>Approximation Method | 137 |

INTRODUCTION:

The following routines detail methods to clear defined segments or all of contiguous user memory. In all cases, the selected program is loaded into, and executed from, locations NOT designated to be zeroed by the program.

- I. To clear memory, given a defined START ADDRESS and the (decimal) NUMBER OF BYTES (limits  $0 < N < 256$ ).

EXAMPLE Clear 64 bytes from initial address = '0000'.

| ADDRS | DATA |    |    |    | LABEL | SYMBOLIC INSTRUCTION |                 | COMMENT                          |
|-------|------|----|----|----|-------|----------------------|-----------------|----------------------------------|
|       | B0   | B1 | B2 | B3 |       | OPCODE               | OPERANDS        |                                  |
| 1     | 100  | 20 |    |    | CLRM1 | EDRZ                 | R0              | clear R0 and R3, then store      |
| 2     | 1    | C3 |    |    |       | STRZ                 | R3              | R0 in defined start addr (0000)  |
| 3     | 2    | CF | 3F | FF | AGAIN | STRA, R0             | STRADD-1, R3, + | to 1FFF, indexed by R3 and lines |
| 4     | 6    | E7 | 40 |    |       | COMZ R3              | H'NN'           | Now, is R3 = to number of bytes  |
| 5     | 8    | 9B | 79 |    |       | BCFR, EQ             | AGAIN           | to be transferred? No! Clear     |
| 6     | 9    | 1F | 1B | 00 |       | BCTA, UN             | MONITOR         | another byte. Yes! say hello     |
| 7     |      |    |    |    | *LAST | ADDRESS =            | '10B'           |                                  |
| 8     |      |    |    |    |       |                      |                 |                                  |
| 9     |      |    |    |    |       |                      |                 |                                  |

| HEX CONVERSION |     |         |    |
|----------------|-----|---------|----|
| 2              |     | 1       |    |
| HEX=DEC        |     | HEX=DEC |    |
| 0              | 0   | 0       | 0  |
| 1              | 16  | 1       | 1  |
| 2              | 32  | 2       | 2  |
| 3              | 48  | 3       | 3  |
| 4              | 64  | 4       | 4  |
| 5              | 80  | 5       | 5  |
| 6              | 96  | 6       | 6  |
| 7              | 112 | 7       | 7  |
| 8              | 128 | 8       | 8  |
| 9              | 144 | 9       | 9  |
| A              | 160 | A       | 10 |
| B              | 176 | B       | 11 |
| C              | 192 | C       | 12 |
| D              | 208 | D       | 13 |
| E              | 224 | E       | 14 |
| F              | 240 | F       | 15 |
| 0 1 2 3        |     | 4 5 6 7 |    |
| BYTE           |     |         |    |

**NOTE:**

Recall use of indexed absolute addressed instructions (e.g., STRA at location H'102') is page limited. Address 0000-1 is '1FFF'.

ALTERNATIVE: Clear 32 bytes, starting at address 0000.

e.g.  $100_{10} = H'64'$

|    |     |    |    |    |        |          |                 |                                  |
|----|-----|----|----|----|--------|----------|-----------------|----------------------------------|
| 18 |     |    |    |    |        |          |                 |                                  |
| 19 | 100 | 20 |    |    | CLRM-2 | EDRZ     | R0              | clear R0 and R3, then initialize |
| 20 | 1   | C3 |    |    |        | STRZ     | R3              | R2 to number of bytes to be      |
| 21 | 2   | 06 | 20 |    |        | LODI, R2 | H'NN'           | cleared, then clear them start-  |
| 22 | 4   | CF | 3F | FF | AGAIN  | STRA, R0 | STRADD-1, R3, + | ing at the defined initial       |
| 23 | 7   | FA | 7B |    |        | BDBR, R2 | AGAIN           | address. All done? No!           |
| 24 | 9   | 1F | 1B | 00 |        | BCTA, UN | MONITOR         | Finish, Yea! Exit to monitor     |
| 25 |     |    |    |    |        |          |                 |                                  |
| 26 |     |    |    |    |        |          |                 |                                  |
| 27 |     |    |    |    |        |          |                 |                                  |

- II. To clear 1 or more non-contiguous 256-byte segments in memory, given the START ADDRESS of each 256 byte segment.

|    |      |    |    |    |       |          |                  |                             |
|----|------|----|----|----|-------|----------|------------------|-----------------------------|
| 1  |      |    |    |    | CLRM3 | FILE 3   | (of 15)          | Alternate clear Mem program |
| 2  | 1793 | 20 |    |    |       | EDBZ     | R0               |                             |
| 3  | 4    | C3 |    |    |       | STRZ     | R3               | LD256 ADDR H'0000'          |
| 4  | 5    | CE | 20 | 00 |       | STRA, R0 | BLOCK 1-1, R3, + | H1256 ADDR H'0100'          |
| 5  | 8    | 5B | 7B |    |       | BRNB, R3 | \$-3             | this alternate program      |
| 6  | A    | CE | 21 | 00 |       | STRA, R0 | BLOCK 2-1, R3, + | zeros user memory           |
| 7  | D    | 5B | 7B |    |       | BRNB, R3 | \$-3             | addresses 0-1FF.            |
| 8  | F    | 1F | 1B | 00 |       | BCTA, UN | MONITOR          |                             |
| 9  |      |    |    |    |       |          |                  | LAST ADDR '17A1             |
| 10 |      |    |    |    |       |          |                  |                             |
| 11 |      |    |    |    |       |          |                  |                             |
| 12 |      |    |    |    |       |          |                  |                             |

## NOTE:

Each 256 byte segment requires dedicated memory of 5 bytes for clearing instructions, once constants are set up.

- III. To clear one to thirty-two contiguous 256-byte segments in USER MEMORY.

|    |      |    |    |    |         |          |                  |                                   |
|----|------|----|----|----|---------|----------|------------------|-----------------------------------|
| 13 |      |    |    |    |         |          |                  |                                   |
| 14 | 17A2 | 20 |    |    | CLRM4   | EDBZ, R0 |                  | clear R0 and R3, then load        |
| 15 | 3    | C3 |    |    |         | STRZ, R3 |                  | R1 with number (0 ≤ X ≤ 1F)       |
| 16 | 4    | 05 | 02 |    |         | LOOT, R1 | X                | of 256-byte blocks to be          |
| 17 | 6    | CE | 20 | 00 | AGAIN   | STRA, R0 | STRTADD-1, R3, + | zeroed. Now clear a block         |
| 18 | 9    | 5B | 7B |    |         | BRNB, R3 | AGAIN            | of 256 bytes. Cleared?            |
| 19 | B    | F9 | 03 |    |         | BDRP, R1 | NEW ADDR         | Yes! Is there another con-        |
| 20 | AD   | 1F | 1B | 00 |         | BCTA, UN | MONITOR          | tiguous block? No, exit to        |
| 21 | BD   | 0A | 75 |    | NEWADDR | LODR, R2 | AGAIN + 1        | monitor. Yes. Fetch MSB of        |
| 22 | 2    | B6 | 01 |    |         | ADDI, R2 | 1                | STRTADD and increment             |
| 23 | 4    | CA | 71 |    |         | STRB, R2 | AGAIN + 1        | it, then store it back as new     |
| 24 | 6    | 1B | 6E |    |         | BCTR, UN | AGAIN            | STRTADD. Then repeat clear-       |
| 25 |      |    |    |    |         |          |                  | ing of this block. Last addr 17A2 |
| 26 |      |    |    |    |         |          |                  |                                   |

See Note on next page.



INTRODUCTION:

This routine rearranges data in a specified block of memory in order of its ascending value. Other sort routines are contained in the APPLICATIONS section of the 2650 MICROPROCESSOR Reference Manual.

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| +                                        | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| +                                        | 1E | 1F | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C |
| +                                        | 2D | 2E | 2F | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B |
| +                                        | 3C | 3D | 3E | 3F | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A |
| +                                        | 4B | 4C | 4D | 4E | 4F | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| +                                        | 5A | 5B | 5C | 5D | 5E | 5F | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 |
| +                                        | 69 | 6A | 6B | 6C | 6D | 6E | 6F | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| +                                        | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F | 80 | 81 | 82 | 83 | 84 | 85 | 86 |
| +                                        | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F | 90 | 91 | 92 | 93 | 94 | 95 |
| +                                        | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F | A0 | A1 | A2 | A3 | A4 |
| +                                        | A5 | A6 | A7 | A8 | A9 | AA | AB | AC | AD | AE | AF | B0 | B1 | B2 | B3 |
| +                                        | B4 | B5 | B6 | B7 | B8 | B9 | BA | BB | BC | BD | BE | BF | C0 | C1 | C2 |
| +                                        | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF | D0 | D1 |
| +                                        | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | DA | DB | DC | DD | DE | DF | E0 |
| +                                        | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | EA | EB | EC | ED | EE | EF |
| +                                        | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | FA | FB | FC | FD | FE |
| +                                        | FF |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

INDIRECT RELATIVE ADDRESS: Add H'80' TO DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE GETSORT START ADDR C

DESCRIPTION:

a "bubble sort" routine to arrange random data in order of ascending value. - program works!

ROUTINE SHEET    OF   MEMORY LOCATIONS THIS SHEET   

| ADDRS | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION                |             | COMMENT                    |
|-------|------|----|----|---------|-------------------------------------|-------------|----------------------------|
|       | B3   | B1 | B2 |         | OPCODE                              | OPERANDS    |                            |
| 1     |      |    |    |         | * put random data in locations '80' |             | through 'C0'               |
| 2     |      |    |    |         |                                     |             |                            |
| 3     | 0000 | 3F | 01 | GETSORT | BSTA, UN                            | SORT        | Exit via reset             |
| 4     |      | B0 |    |         | WRTC                                | R0          | on return, exit to Monitor |
| 5     |      |    |    |         |                                     |             |                            |
| 6     | 0100 | 0B | 23 | SORT    | LODR, R3                            | LEN-1       | a good exercise: write     |
| 7     | 2    | 03 |    | PASS    | LODZ                                | R3          | the comments for this      |
| 8     | 3    | C2 |    |         | STRZ                                | R2          | routine.                   |
| 9     | 4    | 0E | E1 | LOOP    | LODA, R0                            | *LIST, R2   |                            |
| 10    | 7    | EE | E1 |         | COMA, R0                            | *LIST-1, R2 |                            |
| 11    | A    | 9A | 11 |         | BCFR, LT                            | LOC         |                            |
| 12    | C    | C1 |    |         | STRZ                                | R1          |                            |
| 13    | D    | 0E | E1 |         | LOD, R0                             | *LIST-1, R2 |                            |
| 14    | 110  | CE | E1 |         | STRA, R0                            | *LIST, R2   |                            |
| 15    | 3    | 01 |    |         | LODZ                                | R1          |                            |
| 16    | 4    | CE | E1 |         | STRA, R0                            | *LIST-1, R2 |                            |
| 17    | 7    | DA | 00 |         | BIRR, R2                            | \$+2        |                            |
| 18    | 9    | EA | 09 |         | COMR, R2                            | LEN         |                            |
| 19    | B    | 9B | 67 |         | BCFR, EQ                            | LOOP        |                            |
| 20    | D    | FB | 63 | LOC     | BDRR, R3                            | PASS        |                            |
| 21    | 11F  | 17 |    |         | RETC, UN                            |             |                            |
| 22    |      |    |    |         |                                     |             |                            |
| 23    |      |    |    |         |                                     |             |                            |
| 24    | 120  | 00 | 80 | LIST    |                                     |             |                            |
| 25    | 122  | 00 | 7F | LIST-1  |                                     |             |                            |
| 26    | 124  | 10 |    | LEN     |                                     |             |                            |
| 27    | 125  | 0F |    | LEN-1   |                                     |             |                            |

INTRODUCTION:

This simple game will provide more action in an evening than CRAP-GAME, particularly if your friends enjoy some friendly wagering. After you have loaded the program into the INSTRUCTOR, play as follows:

1. Depress **RST** then **INT**      When the FLAG LED comes on, your friends place their bets in the field of their choice. Then....
2. Depress **SENS**      to start the random generation of numbers. Then.....
3. Depress **SENS** again ...      when any player calls for the display of the generated number. The INSTRUCTOR displays the number both on the parallel I/O LEDs and in the display. Bets are paid by "the house" as indicated in the table (below).
4. Depress **INT** .....      to call for placing new bets.
5. Repeat steps 2, 3, & 4 ..until the message "YOU HOPE" is displayed. At that point, the game is over. Repeat from step 1 to start a new game.

| LEDs - ANY PATTERN: | ODDS  | HEX DISPLAY:                               | ODDS  |
|---------------------|-------|--------------------------------------------|-------|
| all off or all on   | 250:1 | Specific 2-digit number<br>(e.g. 00 to FF) | 250:1 |
| 1 or 7 LEDs on      | 15:1  | Specific M.S. digit                        | 15:1  |
| 2 or 6 LEDs on      | 7:1   | Specific L.S. digit                        | 15:1  |
| 3 or 5 LEDs on      | 7:2   | Any one of 5 specified<br>2-digit numbers  | 50:1  |
| 4 LEDs on           | 5:2   | Any one of 10 specified<br>2-digit numbers | 25:1  |
| 0 or 1 LED on       | 25:1  |                                            |       |
| 2 or 3 LEDs on      | 2:1   |                                            |       |
| 4 or 5 LEDs on      | 1:1   |                                            |       |
| 6, 7, or 8 LEDs on  | 6:1   |                                            |       |
| 0, 1, or 2 LEDs on  | 11:2  |                                            |       |
| 3, 4, or 5 LEDs on  | 2:5   |                                            |       |
| 6, 7, or 8 LEDs on  | 11:2  |                                            |       |

## \*\*\*\*\* WARNING \*\*\*\*\*

Gambling laws and regulations vary from state to state. Signetics furnishes this and other games for your enjoyment only. Further permission to use this program for gaming purposes forbidden or otherwise restricted by the laws of your state, is expressly prohibited by Signetics.



Set: INTERRUPT SELECTION SWITCH to KEYBOARD  
 INTERRUPT ADDRESS SWITCH to INDIRECT  
 I/O SELECTION SWITCH to EXTENDED I/O PORT 7

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| +                                        | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 |
| -                                        | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 |
| +                                        | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |
| +                                        | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| -                                        | 70 | 6F | 6E | 6D | 6C | 6B | 6A | 69 | 68 | 67 | 66 | 65 | 64 | 63 | 62 |
| +                                        | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E |
| +                                        | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 |
| -                                        | 60 | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 |
| +                                        | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E |
| +                                        | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 |
| -                                        | 50 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 |

INDIRECT RELATIVE ADDRESS: Add H'80' to displacement

## 2650 PROGRAMMING FORM

ROUTINE ODDS START ADDR 000E

DESCRIPTION INDIRECT ADDR'D INTERRUPT

at location '07'

ROUTINE SHEET 1 OF     

MEMORY LOCATIONS THIS SHEET     

|    | ADDRES | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |               | COMMENT                       |
|----|--------|------|----|----|---------|----------------------|---------------|-------------------------------|
|    |        | B0   | B1 | B2 |         | OPCODE               | OPERANDS      |                               |
| 1  | 0000   | 1B   | 0E |    |         | BCTR, UN             | NEWGAME       | ON RST, GO START NEWGAME      |
| 2  |        |      |    |    |         |                      |               |                               |
| 3  | 07     | 00   | 80 |    | INTADD  | ACON                 | INTSUC        | ENTER routine to place bet    |
| 4  |        |      |    |    |         |                      |               | ON int. KEY only              |
| 5  | 0010   | 20   |    |    | NEWGAME | EORZ                 | R0            | Prep: init. R0=R3=0           |
| 6  | 1      | C3   |    |    |         | STRZ                 | R3            | then, clear SAVECT, 1256      |
| 7  | 2      | CF   | 21 | 00 |         | STRA, R0             | SAVECT, R3, + | location starting at '100'    |
| 8  | 5      | 5B   | 7B |    |         | BRNR, R3             | \$-3          | Finished?                     |
| 9  | 7      | 74   | 40 |    |         | CPSU                 | FLAG          | Yes! turn off FLAG, and       |
| 10 | 19     | 1B   | 7E |    | PLAY    | BCTR, UN             | PLAY          | loop until (INT) depressed to |
| 11 | 0020   | 77   | 08 |    |         | PPSL                 | WC            | place bets. // Set carry for  |
| 12 | 2      | 0D   | 1E | 70 |         | LODA, R1             | RIGID 1       | build up. Now, get a byte     |
| 13 |        |      |    |    |         |                      |               | from RIGID addr (twice,)      |
| 14 | 5      | 0D   | 5C | 89 |         | LODA, R0             | RIGID 2, R    | and enter computation         |
| 15 |        |      |    |    |         |                      |               | First set roll rate then      |
| 16 | 8      | 07   | 80 |    | ROLLING | LODI, R3             | H'80'         | incr. SAVECT, R1 indexed      |
| 17 | A      | 04   | 01 |    |         | LODI, R0             | 1             | Now, test if                  |
| 18 | C      | 8D   | 61 | 00 |         | ADDA, R0             | SAVECT, R1    | location has carried          |
| 19 | 2F     | B5   | 01 |    |         | TPSL                 | C             | Yes! Exit to GAMEOVER         |
| 20 | 31     | 1B   | 11 |    |         | BCTR, EQ             | GAMEOVER      | No! Save the increment        |
| 21 | 3      | CD   | 61 | 00 |         | STRA, R0             | SAVECT, R1    | add it and do roll rate       |
| 22 | 6      | FB   | 7B |    |         | BDRR, R3             | \$-3          | Done? Yes! Fetch another      |
| 23 |        |      |    |    |         |                      |               | random number, add to         |
| 24 | 8      | 8D   | 5D | 00 |         | ADDA, R0             | NEWNR, R1, -  | R0 content, then slow         |
| 25 | B      | D4   | 07 |    |         | WRITE, R0            | PORT7         | high speed roll in lights     |
| 26 | 3D     | B4   | 80 |    |         | TPSU                 | SENSE         | Does Player want to see       |
| 27 |        |      |    |    |         |                      |               | his roll?                     |

The program continues on the next page.

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| +                                        | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| +                                        | 1E | 1F | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C |
| +                                        | 2D | 2E | 2F | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B |
| +                                        | 3C | 3D | 3E | 3F | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A |
| +                                        | 4B | 4C | 4D | 4E | 4F | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| +                                        | 5A | 5B | 5C | 5D | 5E | 5F | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 |
| +                                        | 69 | 6A | 6B | 6C | 6D | 6E | 6F | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| +                                        | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F | 80 | 81 | 82 | 83 | 84 | 85 | 86 |
| +                                        | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F | 90 | 91 | 92 | 93 | 94 | 95 |
| +                                        | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F | A0 | A1 | A2 | A3 | A4 |
| +                                        | A5 | A6 | A7 | A8 | A9 | AA | AB | AC | AD | AE | AF | B0 | B1 | B2 | B3 |
| +                                        | B4 | B5 | B6 | B7 | B8 | B9 | BA | BB | BC | BD | BE | BF | C0 | C1 | C2 |
| +                                        | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF | D0 | D1 |
| +                                        | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | DA | DB | DC | DD | DE | DF | E0 |
| +                                        | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | EA | EB | EC | ED | EE | EF |
| +                                        | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | FA | FB | FC | FD | FE |
| +                                        | FF |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

INDIRECT RELATIVE ADDRESSING: Add H'80' to displacement

## 2650 PROGRAMMING FORM

ROUTINE ODDS START ADDR \_\_\_\_\_

DESCRIPTION \_\_\_\_\_

ROUTINE SHEET 2 OF \_\_\_\_\_

MEMORY LOCATIONS THIS SHEET \_\_\_\_\_

|    | ADDRS | DATA |    |    | LABEL    | SYMBOLIC INSTRUCTION |           | COMMENT                    |
|----|-------|------|----|----|----------|----------------------|-----------|----------------------------|
|    |       | E2   | B1 | B2 |          | OPCODE               | OPERANDS  |                            |
| 1  | 003F  | 3B   | 0D |    |          | BCTR, EQ             | DISPLAY   | Yes! Go show his roll      |
| 2  | 41    | 1B   | 65 |    |          | BCTR, UN             | ROLLING   | Not yet! keep rolling.     |
| 3  |       |      |    |    |          |                      |           |                            |
| 4  | 44    | 05   | 00 |    | GAMEOVER | LODI, R1             | <OVER-1   | Set display constants to   |
| 5  | 6     | 06   | 77 |    |          | LODI, R2             | >OVER-1   | show game is over          |
| 6  | 8     | 07   | 01 |    |          | LODI, R3             | 1         | message = "YOU HOPE"       |
| 7  | A     | BB   | E6 |    |          | ZBSR                 | * USRDSP  | Display until player de-   |
| 8  | 4C    | 1B   | 76 |    |          | BCTR, UN             | GAMEOVER  | presses RST to start new   |
| 9  |       |      |    |    |          |                      |           | game                       |
| 10 |       |      |    |    |          |                      |           |                            |
| 11 | 4E    | CC   | 17 | D9 | DISPLAY  | STRA, R0             | SAVEREG   | Roll is stopped. He wants  |
| 12 | 51    | 76   | 60 |    |          | PPSU                 | FLAG, II  | his ct, so save the reg.   |
| 13 | 3     | 3F   | 1E | A9 |          | BSTR, UN             | SAVEREG   | and set FLAG and II        |
| 14 | 6     | D5   | 07 |    |          | WRITE, R1            | PORT7     | Now, show his roll on LEDs |
| 15 | 8     | 01   |    |    |          | LODZ, R1             |           | then split his roll into 2 |
| 16 | 9     | BB   | F4 |    |          | ZBSR                 | * DISLSD  | nibls and store in the     |
| 17 | B     | C8   | 19 |    |          | STRR, R0             | ROLMES+6  | ROLL message.              |
| 18 | D     | C9   | 1B |    |          | STRR, R1             | ROLMES+7  | Now                        |
| 19 | 5F    | 05   | 00 |    | DSPLAGN  | LODI, R1             | < DATA1-1 | Set up display constants   |
| 20 | 61    | 06   | 6F |    |          | LODI, R2             | > DATA1-1 | for his roll               |
| 21 | 3     | 07   | 01 |    |          | LODI, R3             | 1         | and                        |
| 22 | 5     | BB   | E6 |    |          | ZBSR                 | * USRDSP  | show it. When he de-       |
| 23 | 7     | 74   | 20 |    |          | CPSU                 | II        | presses INT, all payoffs   |
| 24 | 9     | DB   | 7E |    |          | BIRR, R3             | \$        | are made and he's ready    |
| 25 | B     | 76   | 20 |    |          | PPSU                 | II        | to place new bets.         |
| 26 | 6D    | 1B   | 70 |    |          | BCTR, UN             | DISPAGN   | Until then, display the    |
| 27 |       |      |    |    |          |                      |           | roll.                      |

The program continues on the next page.

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| +                                        | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 |
| -                                        | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 |
| +                                        | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |
| +                                        | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| -                                        | 70 | 6F | 6E | 6D | 6C | 6B | 6A | 69 | 68 | 67 | 66 | 65 | 64 | 63 | 62 |
| +                                        | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E |
| +                                        | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 |
| -                                        | 60 | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 |
| +                                        | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E |
| +                                        | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 |
| -                                        | 50 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 |

INDIRECT RELATIVE ADDRESS: Add H'00' TO DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE ODDS START ADDR \_\_\_\_\_

DESCRIPTION \_\_\_\_\_

ROUTINE SHEET 3 OF \_\_\_\_\_

MEMORY LOCATIONS THIS SHEET \_\_\_\_\_

|    | ADDRS | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |          | COMMENT                  |
|----|-------|------|----|----|--------|----------------------|----------|--------------------------|
|    |       | B2   | B1 | B2 |        | OPCODE               | OPERANDS |                          |
| 1  | 0070  | 13   | 15 | 11 | ROLMES | RES                  | 8        | "ROLL = " message        |
| 2  | 3     | 11   | 16 | 17 |        |                      |          |                          |
| 3  | 6     | XX   | XX |    |        |                      |          |                          |
| 4  | 78    | 1B   | 00 | 12 | OVER   | RES                  | 8        | "YOU HOPE" message       |
| 5  | 8     | 17   | 14 | 00 |        |                      |          |                          |
| 6  | E     | 10   | 0E |    |        |                      |          |                          |
| 7  |       |      |    |    |        |                      |          |                          |
| 8  | 0080  | 12   |    |    | INTSVC | SPSU                 |          | Fetch PSU, and clean     |
| 9  | 1     | A4   | 61 |    |        | SUBI                 | '61'     | FLAG, II, and subtract 1 |
| 10 | 2     | 92   |    |    |        | LPSU                 |          | From SP, then store      |
| 11 | 4     | 0C   | 17 | DA |        | LODA, R0             | SAVREC   | Now, restore the         |
| 12 | 7     | 3F   | 1E | B3 |        | BSTA, UN             | RESTRO   | previously saved regs    |
| 13 | A     | B4   | 80 |    |        | TPSU                 | SENS     | Then test for SENS. Loop |
| 14 | c     | 98   | 7C |    |        | BCFR, EQ             | \$-2     | 't, if you get it. Then  |
| 15 | 8E    | FB   | 7E |    |        | BDRR, R3             | \$       | delay 1/2 sec            |
| 16 | 90    | FA   | 7C |    |        | BDRR, R2             | \$-2     | and                      |
| 17 | 2     | 1F   | 00 | 28 |        | BCTA, UN             | ROLLING  | roll numbers again.      |
| 18 |       |      |    |    |        |                      |          |                          |

This is the last sheet of program "ODDS".

NOTES:

The programming variation on the next page provides the following functions:

1. On depressing **RST** and **INT** to start the game, and on subsequent **INT** to place bets, the message "PLACE BET" is displayed.
2. Depress any hex or function key to close the bets and start the roll.
3. Before the roll starts, the message "ROLL..." is displayed for  $\frac{1}{2}$  second. You do not have to depress **SENS** to start the roll.
4. Depress **SENS** to stop the roll of random generated numbers.

The ODDS remain the same.

SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

DIRECT RELATIVE ADDRESSING - SECOND BYTE

|   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| + | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| - | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E | 1F |
| + | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E | 2F |
| - | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E | 3F |
| + | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A | 4B | 4C | 4D | 4E | 4F |
| - | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 5A | 5B | 5C | 5D | 5E | 5F |
| + | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 6A | 6B | 6C | 6D | 6E | 6F |
| - | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F |

INDIRECT RELATIVE ADDRESS: Add H'80' TO DISPLACEMENT

2650 PROGRAMMING FORM

ROUTINE ODDS START ADDR VARIATION

DESCRIPTION VARIATION TO PROVIDE

"PLACE BET" AND "ROLL...." PROMPTS

FROM LOCATION '80', overlay original

ODDS PROGRAM WITH THE FOLLOWING

ROUTINE SHEET OF

MEMORY LOCATIONS THIS SHEET

| ADDRS | DATA |    |    | LABEL    | SYMBOLIC INSTRUCTION |              | COMMENT                   |
|-------|------|----|----|----------|----------------------|--------------|---------------------------|
|       | B2   | B1 | B2 |          | OPCODE               | OPERANDS     |                           |
| 0080  | 07   | 00 |    | INTSVC   | LODI, R3             | 0            | Rules and odds for        |
| 2     | 05   | 00 |    | PLACE    | LODI, R1             | < PLACEBET-1 | the variation have        |
| 4     | 06   | 8D |    |          | LODI, R2             | > PLACEBET-1 | minor changes: See        |
| 6     | BB   | E6 |    |          | ZBSR                 | * USRDSP     | preceding page.           |
| 8     | 12   |    |    |          | SPSU                 |              |                           |
| A     | A4   | 11 |    |          | SUBI, R0             | H'61'        |                           |
| C     | 92   |    |    |          | LPSU                 |              |                           |
| E     | 1B   | 09 |    |          | BCTR, UN             | ROLL BET     |                           |
| 8E    | 10   | 11 | 0A | PLACEBET | RES                  | 8            | message is "PLACE BET"    |
| 91    | 0C   | 0E | 0B |          |                      |              | Deprave any hex or func-  |
| B     | 0E   | 87 |    |          |                      |              | tion key to go start roll |
| D     | XX   |    |    | ROLLTIME |                      |              |                           |
| 7     | 07   | 7F |    | ROLLBET  | LODI, R3             | H'7F'        |                           |
| 9     | CB   | 7B |    |          | STRR, R3             | ROLLTIME     |                           |
| B     | 05   | 00 |    |          | LODI, R1             | < ROLL = -1  |                           |
| D     | 06   | AD |    |          | LODI, R2             | > ROLL = -1  |                           |
| 9F    | BB   | E6 |    |          | ZBSR                 | * USRDSP     |                           |
| A1    | 0B   | 73 |    |          | LODR, R3             | ROLLTIME     |                           |
| C     | FB   | 74 |    |          | BDRR, R3             | ROLL         |                           |
| 5     | 0C   | 17 | D9 |          | LODA, R0             | SAVREG       |                           |
| 8     | 3F   | 1E | B3 |          | BSTA, UN             | RESTRO       |                           |
| B     | 1F   | 00 | 2B |          | BCTA, UN             | ROLLING      |                           |
| AE    | 13   | 15 | 11 | ROLL =   | RES                  | 8            | message "ROLL = ..."      |
| B1    | 11   | 16 | 97 |          |                      |              | Flasher for 1/2 sec Ham   |
| 00B4  | 97   | 97 |    |          |                      |              | program exits to ROLL     |
|       |      |    |    |          |                      |              | for odds.                 |

CONOITIONS:

- Take off from "TRAIN"
- RESET key is available for game selection
- Possibility: work with SENS for edge trigger
- Alternative: Use GNP routine for position and SENS to fire at moving target.

AUTHOR'S NOTE: This is as far as I specified on this idea.

---

IDEA - - REACTIONCONOITIONS:

- Manipulate SENS key so as to keep the target in the middle of the display. The target could be the small box character used as a coal car or caboose in TRAIN.
- Count the total number of times you enter the display
  - one number display. Stop the game after X displays (e.g. 1024)
  - initialize counters
- Compare target pattern with pattern in SWYARD bit 4 at any time.
  - if compare, increment a counter
  - if no compare, no increment.
- Save number - ratio between the total count and your count count determines accuracy.
- Alternative: at end of fixed count, display his count and /or a message:
 

|                                      |   |                                                                              |
|--------------------------------------|---|------------------------------------------------------------------------------|
| SUPER<br>GOOD<br>FAIR<br>POOR<br>YUK | } | Better; All of these messages are possible with the INSTRUCTOR display font. |
|--------------------------------------|---|------------------------------------------------------------------------------|
- For message, compare against fixed count, get an index, and display message.
- Set up speed factor for flying target
- Set up programmable odds calculations.

## CHARACTER FONT:

b. BELL  
 C CHERRY  
 L LEMON  
 P PLUM  
 □ BAR

## EACH WHEEL: 16 characters

CHERRY  
 BELL  
 PLUM  
 CHERRY  
 BAR  
 LEMON  
 CHERRY  
 PLUM  
 CHERRY  
 LEMON  
 PLUM  
 CHERRY  
 PLUM  
 BELL  
 CHERRY  
 LEMON

} in order

## PAYOFF:

|                        |       |
|------------------------|-------|
| Cherry in wheel 1      | 2:1   |
| cherry in wheels 1 & 2 | 5:1   |
| 3 cherries             | 10:1  |
| 3 bells                | 20:1  |
| 3 Lemons               | 18:1  |
| 3 Plums                | 14:1  |
| 2 bells + 1 Bar        | 20:1  |
| 2 lemons + 1 Bar       | 18:1  |
| 2 Plums + 1 Bar        | 14:1  |
| 3 Bars (JACKPOT)       | 250:1 |

## POSSIBLE DISPLAYS \*\*

WHEELS

1 2 3  
 [ ] [ ] [ ] [ ] [ ] } Jackpot after 3 wheels stop 250:1.

[ J A C - P O T ] } Altern: Flash

[ X ] [ ] [ X ] [ ] [ C ] } cherry in 3rd wheel stable after 1 wheel is stopped

X = changing character in rolling wheels.

1 2 3  
 [ X ] [ ] [ L ] [ ] [ L ] } 2 lemons stabilize 1st wheel rotating

[ P ] [ ] [ P ] [ ] [ b ] } 2 plums and a bell stabilized after 3 wheels are stopped

[ P A Y ] [ 1 8 = 1 ] } "payoff, 18:1 on 3 lemons"

[ Y O U ] [ L O S T ] } " on no payoff "

[ P L A C E B E T ] } Flash 3 times then exit to wheel display  
 [ R O T A T E ]

## \*Alternate:

[ P A Y ] [ O F F ] } Flash 4 times

[ 1 8 ] [ 7 0 ] [ 1 ]

## \*2nd ROTATION ALTERNATE:

Have each "wheel" displayed 1 at a time on the display. Show horizontal motion. Strobe for a character at some position. Repeat 2 more times. Then, display the three characters which were strobed. This alternative provides more "action".

**Author's Note:** This program should be written with a medium degree of difficulty  
 Might want to include a variable delay routine to show the wheels slowing down.  
 Message tables: minimum 7 messages. I'd personally like to see your program. Drop a line to 40 Barney Ct, Menlo Park, CA 94025.

INTRODUCTION:

BINGO should be easily implemented via the INSTRUCTOR's facilities. Basically a game of random number selection, each letter in the word "Bingo" is assigned a group of 15 ascending numbers, e.g.:

|     |         |     |
|-----|---------|-----|
| B1  | through | B15 |
| I16 | "       | I30 |
| N31 | "       | N45 |
| G46 | "       | G60 |
| O61 | "       | O75 |

In some versions, each letter in the word "Bingo" is assigned 12 ascending numbers.

Prior to the numbers being called, each player obtains one or more Bingo cards. Each Bingo card contains, under each letter in the word Bingo, 5 of the possible numbers assigned to that letter. When a number is called, e.g. "N--43", the players with N43 on their cards cover that number with a token. Prior to calling the game, the caller announces the requirements for winning that particular game, e.g.:

"any diagonal covered"  
 "any horizontal row covered"  
 "any vertical column covered"  
 "BLACKOUT" - all numbers covered.

The player who satisfies the requirement first shouts "BINGO!!!". Upon verification of his pattern, he receives the prize.

SSSSSSSSSSSSSSSSSSSSSSSSSS

The program to be written should contain routines for the following:

- indices and tables for each number generation set.
- storage for saved numbers
- prevention of calling the same number more than once within the same game
- method to recall the saved numbers for verification.
  - alternative: use GNPA routine to enter the player's saved numbers into the instructor for comparison. A form of FAST MEMORY PATCH could be used, followed by a routine to compare the saved numbers with the player's numbers
- Suitable display routines and appropriate messages.



INTRODUCTION:

KENO is a form of lottery based on random selection of 16 numbers from a table of numbers 1 through 80. Before the numbers are called, each player selects 1 or more absolute numbers which he bets will be among the numbers selected by the computer. The odds vary as a function of how many numbers he selects, and how many of his numbers match. Both sides of a typical KENO ticket are reprinted below. The payoffs are shown as they relate to the purchase of an \$0.70 ticket in Nevada

PROGRAM CONSIDERATIONS:

- random number generation between 1 and 80
- routine to save the random number selected.
- routine to prevent selection of the same number within the same game.
- suitable messages for "rolling display" and selection of each saved number.
- Payoff message and summary of displayed numbers.

KENO -- COMPSITE TICKET

WINNING TICKET MUST BE CASHED BEFORE START OF NEXT KENO GAME

|    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|
| 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 |
| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 |

WE PAY ON TICKET PLAYED BY CUSTOMER EACH GAME

|    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|
| 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 50 |
| 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 |
| 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 70 |
| 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 80 |

KENO RUNNERS ARE AVAILABLE FOR YOUR CONVENIENCE  
WE ARE NOT RESPONSIBLE IF TICKETS ARE TOO LATE FOR CURRENT GAME

### TABLE OF TICKET "PRICES" & PAYOFFS

Tickets may be played  
for any multiple of rates.

| MARK 1 SPOT    | MARK 7 SPOTS   | MARK 11 SPOTS  | MARK 14 SPOTS  |
|----------------|----------------|----------------|----------------|
| Spots 70c Pays | Spots 70c Pays | Spots 70c Pays | Spots 70c Pays |
| 1 2.10         | 4 .70          | 6 7.00         | 6 .85          |
|                | 5 16.00        | 7 32.50        | 7 7.50         |
| MARK 2 SPOTS   | 6 260.00       | 8 266.00       | 8 25.00        |
| Spots 70c Pays | 7 6,000.00     | 9 1,400.00     | 9 210.00       |
| 2 8.50         |                | 10 8,750.00    | 10 800.00      |
|                |                | 11 13,650.00   | 11 2,330.00    |
| MARK 3 SPOTS   | MARK 8 SPOTS   |                | 12 9,975.00    |
| Spots 70c Pays | Spots 70c Pays |                | 13 25,000.00   |
| 2 .70          | 5 6.30         | MARK 12 SPOTS  | 14 25,000.00   |
| 3 30.00        | 6 63.00        | Spots 70c Pays |                |
|                | 7 1,155.00     | 6 4.00         |                |
|                | 8 12,600.00    | 7 20.00        |                |
| MARK 4 SPOTS   |                | 8 160.00       |                |
| Spots 70c Pays |                | 9 700.00       |                |
| 2 .70          | MARK 9 SPOTS   | 10 1,800.00    |                |
| 3 3.00         | Spots 70c Pays | 11 12,500.00   | MARK 15 SPOTS  |
| 4 75.00        | 5 2.10         | 12 25,000.00   | Spots 70c Pays |
|                | 6 31.50        |                | 7 .70          |
| MARK 5 SPOTS   | 7 234.50       |                | 8 6.20         |
| Spots 70c Pays | 8 3,290.00     |                | 9 61.00        |
| 3 .70          | 9 12,950.00    | MARK 13 SPOTS  | 10 175.00      |
| 4 6.50         |                | Spots 70c Pays | 11 1,750.00    |
| 5 580.00       | MARK 10 SPOTS  | 6 1.00         | 12 7,000.00    |
|                | Spots 70c Pays | 7 10.50        | 13 21,000.00   |
| MARK 6 SPOTS   | 5 1.40         | 8 60.00        | 14 25,000.00   |
| Spots 70c Pays | 6 14.00        | 9 500.00       | 15 25,000.00   |
| 3 .40          | 7 99.40        | 10 2,700.00    |                |
| 4 2.50         | 8 700.00       | 11 6,650.00    |                |
| 5 70.00        | 9 3,150.00     | 12 17,500.00   |                |
| 6 1,300.00     | 10 13,300.00   | 13 25,000.00   |                |

We pay according to original ticket, the ticket submitted to us by the player before the start of each game

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| +                                        | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 |
| -                                        | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 |
| +                                        | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |
| +                                        | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| -                                        | 70 | 6F | 6E | 6D | 6C | 6B | 6A | 69 | 68 | 67 | 66 | 65 | 64 | 63 | 62 |
| +                                        | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E |
| +                                        | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 |
| -                                        | 60 | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 |
| +                                        | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E |
| +                                        | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 |
| -                                        | 50 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 |
|                                          |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

INDIRECT RELATIVE ADDRESS: Add H'80' TO DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE HALT1 START ADDR 0DESCRIPTION DEMONSTRATE USE OF HALT  
INSTRUCTION; ENTRY TO PROCESSON (1) RESET and (2) INTERRUPT RE-  
QUESTROUTINE SHEET 1 OF 1MEMORY LOCATIONS THIS SHEET 52

| ADDRS | DATA<br>B2 B1 B2 |    |       | LABEL  | SYMBOLIC INSTRUCTION<br>OPCODE OPERANDS |           | COMMENT                        |
|-------|------------------|----|-------|--------|-----------------------------------------|-----------|--------------------------------|
| 1     |                  |    |       |        |                                         |           |                                |
| 2     | 0000             | 20 |       | HALT1  | EORZ                                    | RO        | On entry, initialize           |
| 3     |                  | C1 |       |        | STRZ                                    | R1        | the normal pattern.            |
| 4     |                  | 93 |       |        | LPSL                                    |           | also, clean all of             |
| 5     |                  | 92 |       |        | LPSU                                    |           | the PSW, then go around        |
| 6     |                  | 1F | 00 2E |        | BCTA, UN                                | CONT      | interrupt service to CONT      |
| 7     |                  |    |       |        |                                         |           |                                |
| 8     | 0007             | CC | 01 00 | INTRPT | STRA, RO                                | DOWNCT    | On entry, store current ct     |
| 9     | A                | 77 | 1B    |        | PPSL                                    | RS, WC    | down then clean REG bank       |
| 10    | C                | 75 | 01    |        | CPSL                                    | C         | and WC. (Prime sel.) and clean |
| 11    | E                | 07 | 80    |        | LODI, R3                                | H'80'     | Carry. Now set count, int      |
| 12    | 10               | 04 | 08    | CTDN   | LODI, RO                                | B         | pattern & for B+ase            |
| 13    | 2                | D7 | 07    |        | WRTE, R3                                | PORT7     | Show current count on LEDs     |
| 14    | 4                | FA | 7E    |        | BDRR, R2                                | \$        | then delay                     |
| 15    | 6                | FA | 7C    |        | BDRR, RO                                | \$-2      | and                            |
| 16    | 8                | 53 |       |        | RRR                                     | R3        | push count right. Is it        |
| 17    | 9                | 5B | 75    |        | BRNR, R3                                | CTD       | pushed out? No, push again     |
| 18    | B                | 75 | 19    |        | CPSL                                    | RS, WC, C | Yes, clean prime select and    |
| 19    | D                | 0C | 01 00 |        | LODA, RO                                | DOWNCT    | Carry parameters then          |
| 20    | 0020             | 37 |       |        | RETE, UN                                |           | fetch, prev. stored ct. down   |
| 21    | 0100             | XX |       | DOWNCT |                                         |           | and exit.                      |
| 22    |                  |    |       |        |                                         |           |                                |
| 23    | 002E             | 04 | 30    | CONT   | LODI, RO                                | 4B        | Now, set event counter         |
| 24    | 30               | 85 | 10    |        | ADDI, R1                                | H'10'     | and increment pattern          |
| 25    | 2                | D5 | 07    |        | WRTE, R1                                | PORT7     | and display on LED's           |
| 26    | 4                | 06 | 60    |        | LODI, R2                                | H'60'     | Then set and run 1/4 sec       |
| 22    | 6                | FB | 7E    |        | BDRR, R3                                | \$        | delay between pattern          |
| 23    | 8                | FA | 7C    |        | BDRR, R2                                | \$-2      | changes. 4B changes            |
| 24    | A                | FB | 74    |        | BDRR, RO                                | CONT+2    | run yet? No, do another        |
| 25    | C                | 40 |       |        | HALT                                    |           | Yes. Stop processing and       |
| 26    | 3D               | 1F | 00 01 |        | BCTA, UN                                | HALT+1    | wait for interrupt. On re-     |
| 27    |                  |    |       |        |                                         |           | turn from int., repeat proc.   |

INTRODUCTION:

Many programmers prefer to write their program's mainline sequence and routines in low-order storage. Considerable TIME can be saved in setting breakpoints, entering memory data changes, etc., during debug. After a routine is debugged, it is sensible to relocate it to available locations in high order User RAM. Minimum loss of memory results, since the routines may be located adjacent to each other, thus freeing up additional storage for new routine definition.

In the following series of programs, several approaches to routine relocation are examined. Before you enter each alternative, SET UP A KNOWN INCREMENTING PATTERN IN MEMORY. By doing this, you'll be able to verify your program's operation EXACTLY.

I DIRECT RELOCATION METHOD 1

In which source data and destination data lower addresses, and the number of bytes to be moved are known.  
N bytes: 1 N 256.

EXAMPLE: RELOC1: Source LAD = '60'; Destin.LAD = '1B0'; 55 Bytes.

|   |      |    |    |    |        |          |                 |                             |
|---|------|----|----|----|--------|----------|-----------------|-----------------------------|
| 1 | 17B0 | 20 |    |    | RELOC1 | EDRZ     | R0              | clear R0 and set index in   |
| 2 | 1    | C1 |    |    |        | STRZ     | R1              | R1, also set byte count.    |
| 3 | 2    | 06 | 37 |    |        | LODI, R2 | 55              | then relocate 55 bytes      |
| 4 | 4    | 0D | 20 | 5F | AGAIN  | LDDA, R0 | SOURCE-1, R1, + | Source data to              |
| 5 | 7    | CD | 61 | AF |        | STRA, R0 | DESTIN-1, R1    | Destination memory.         |
| 6 | A    | FA | 78 |    |        | BDRB, R2 | AGAIN           | Move complete? No!          |
| 7 | C    | 1F | 18 | 00 |        | BCTA, UN | MONITOR         | Finish it. Yes! exit to Mon |

II DIRECT RELOCATION METHOD 2

In which Source data LAD, Destination data UAD, and the number of bytes to be moved are known.

EXAMPLE: RELOC2: SrcLAD = '93'; DstUAD = '1FF'; 46 Bytes

|    |      |    |    |    |        |          |                  |                           |
|----|------|----|----|----|--------|----------|------------------|---------------------------|
| 13 | 1798 | 07 | FF |    | RELOC2 | LODI, R3 | >DESTUAD         | Fetch LSP of DESTUAD and  |
| 14 | A    | A7 | 2E |    |        | SUBT, R3 | BYTECT           | subtract BYTECT to derive |
| 15 | C    | CB | 09 |    |        | STBR, R3 | AGAIN+5          | a usable DESTUAD-1.       |
| 16 | 9E   | 07 | 2E |    |        | LODI, R3 | BYTECT           | store it. Now set byte    |
| 17 | AD   | 05 | 00 |    |        | LODI, R1 | 0                | count and set index to 0. |
| 18 | 2    | 0D | 20 | 92 | AGAIN  | LDDA, R0 | SRC LAD-1, R1, + | Then move desired chunk   |
| 19 | 5    | CD | 61 | XX |        | STRA, R0 | DESTUAD-1, R1    | of memory to new loca-    |
| 20 | 8    | FB | 78 |    |        | BDRB, R3 | AGAIN            | tion. All done? No!       |
| 21 | A    | 1F | 18 | 00 |        | BCTA, UN | MONITOR          | Finish the move. Yes!     |
|    |      |    |    |    |        |          |                  | Exit to MONITOR           |

NOTE: The result of subtraction must NOT result in effective address definition into the next LOWER 256-byte segment in order for this program to function properly.

RELOCATION METHOD 3 - INDIRECT

In which the byte transfer count, source data lower address, and destination data upper address are specified.

This routine effectively relocates a block of data while eliminating any need for the user to calculate the destination data Lower Address. Further, relocation is not restricted to boundaries imposed by a given 256-byte memory segment. Also, the user is provided with convenient locations to enter the byte count and Destination UPPER address variables in 4 sequential storage locations.

EXERCISE:

1. Write a short program at address '1780', to load an incrementing pattern (00-FF) into the 1st 512 bytes of user storage.
2. Debug and execute! Then compare your solution with the code provided on the next page.

3. Starting at location '0', load the routine RELOC3 into memory.

Note the specified variables:

- addr '0040': H'011F', the destination UPPER address
- addr '0042': H'002A', the specified byte count
- addr '0059': H'2064', the INDEXED SOURCE data LOWER ADDR-1

4. When executed, this routine will move 42 bytes from location '0065' and following to a block of memory whose last address is '011F'.
5. After executing, examine memory, starting minus 2 locations from the location specified at addresses '0044' & '0045'. Note the precise relocation of data, with transfer terminated at the Destination upper address; '011F'.

| ADDRES  | DATA |    |    | LABEL    | SYMBOLIC INSTRUCTION |                 | COMMENT                   |
|---------|------|----|----|----------|----------------------|-----------------|---------------------------|
|         | B2   | B1 | B2 |          | OPCODE               | OPERANDS        |                           |
| 1 0040  | 01   | 1F |    | DESTUAD  | RES                  | 2               | Destination upper address |
| 2 0042  | 00   | 2A |    | BYTECT   | RES                  | 2               | Specified byte count      |
| 3 0044  | XX   | XX |    | DESTLAD  | RES                  | 2               | Computed Dest. lower      |
| 4 0046  | 77   | 0B |    | RELOC3   | PPSL                 | WC, COM, C      | addr set WC, LOGICAL      |
| 5 0048  | 0B   | 77 |    | DLADCON  | LDDR, R0             | > DESTUAD       | and carry. Now fetch LSB  |
| 6 004A  | AB   | 77 |    |          | SUBR, R0             | > BYTECT        | of Dest. UAD and sub-     |
| 7 004C  | C8   | 77 |    |          | STRR, R0             | > DESTLAD       | tract spec. byte count.   |
| 8 004E  | 0B   | 70 |    |          | LDDR, R0             | < DESTUAD       | Store in DESTLAD 15 byte  |
| 9 0050  | AB   | 70 |    |          | SUBR, R0             | < BYTECT        | Get MSB of Dest. UAD and  |
| 10 0052 | C8   | 70 |    |          | STRR, R0             | < DESTLAD       | subtract MSB of Byte      |
| 11 0054 | 0B   | 6D |    | TRANSFER | LDDR, R3             | > BYTECT        | count. Store in MSB of    |
| 12 0056 | 05   | 00 |    |          | LODI, R1             | 0               | DESTLAD. Now, set byte    |
| 13 0058 | 0D   | 20 | 64 | AGAIN    | LODA, R0             | SACLAD-1, R1, + | count and zero index      |
| 14 005A | CD   | ED | 44 |          | STRA, R0             | * DESTLAD, R1   | Then fetch a byte from    |
| 15 005C | FB   | 7B |    |          | BDRR, R3             | AGAIN           | source data and store it  |
| 16 005E | 1F   | 1B | 00 |          | BCTA, UN             | MONITOR         | in destination. Finished? |

No! complete transfer.  
Yes! go to monitor.

SUGGESTED CODE - INCREMENTING PATTERN TO USER MEMORY

|   |      |    |    |    |         |        |          |                                      |
|---|------|----|----|----|---------|--------|----------|--------------------------------------|
| 2 | 178D | 20 |    |    | INCPAT2 | EDBZ   | RO       | Clear RO, then store in-             |
| 3 |      | 1  | CC | 20 | 00      | AGAIN1 | STRA, RO | cremented contents of                |
| 4 |      | 4  | 58 | 78 |         |        | BRNB, RO | RO in 1 <sup>st</sup> 256 bytes when |
| 5 |      | 6  | CC | 21 | 00      | AGAIN2 | STRA, RO | finished, repeat same pat-           |
| 6 |      | 9  | 58 | 78 |         |        | BRNB, RO | tern for 2 <sup>nd</sup> 256 bytes   |
| 7 |      | 8  | 1F | 18 | 00      |        | BCTA, UN | and then exit to monitor             |
|   |      |    |    |    |         |        |          | last address = H '178D'              |

EXERCISE:

- Given a DestUAD = '17BF', relocate the routine RELOC3. Starting at location '0040', make necessary changes in the program to accomplish this.
- After executing RELOC3 to relocate itself, verify the transfer.
  - Enter the new LOWER address of RELOC3: \_\_\_\_\_
  - Enter the new START address for execution of RELOC3: \_\_\_\_\_
- Write the new address of each instruction
- In the reprint of RELOC3 (below), enter the correct code.
- Set up the following parameters for a data transfer:
  - Bytes to transfer: 99
  - SrcLAD: ..... '0075' } Suggested code on
  - DesUAD: ..... '0140' } next page
- Execute routine INCPAT2.
- Execute routine RELOC3, previously coded in step 5.
- Verify and inspect the newly relocated data.

|    |  |    |    |    |          |          |                   |                         |
|----|--|----|----|----|----------|----------|-------------------|-------------------------|
| 8  |  |    |    |    | DESTUAD  | RES      | 2                 | last address = H '178D' |
| 9  |  |    |    |    | BYTECT   | RES      | 2                 |                         |
| 10 |  | XX | XX |    | DESTLAD  | RES      | 2                 |                         |
| 11 |  | 77 | 08 |    | RELOC3A  | PPSL     | WC, COM, C        |                         |
| 12 |  | 08 |    |    |          | LODR, RO | > DESTUAD         |                         |
| 13 |  | A8 |    |    |          | SUBR, RO | > BYTECT          |                         |
| 14 |  | CB |    |    |          | STRB, RO | > DESTLAD         |                         |
| 15 |  |    |    |    |          | LODR, RO | < DESTUAD         |                         |
| 16 |  |    |    |    |          | SUBR, RO | < BYTECT          |                         |
| 17 |  |    |    |    |          | STRB, RO | < DESTLAD         |                         |
| 18 |  | 08 |    |    | TRANSFER | LODR, R3 | > BYTECT          |                         |
| 19 |  | 05 | 00 |    |          | LODI, R1 | 0                 |                         |
| 20 |  | 0D |    |    |          | LODA, RO | SRCLAD - 1, R1, + |                         |
| 21 |  | CD |    |    |          | STRA, RO | * DESTLAD, R1     |                         |
| 22 |  | FB | 78 |    |          | BDRB, R3 | AGAIN             |                         |
| 23 |  | 1F | 18 | 00 |          | BCTA, UN | MONITOR           |                         |

SUGGESTED CODE  
routine RELOC3A

4. Execute the transfer and verify.

|    |      |    |    |    |      |
|----|------|----|----|----|------|
| 8  | 179D | 01 | 40 |    | DES  |
| 9  | 9F   | 00 | 63 |    | BYT  |
| 10 | A1   | XX | XX |    | DES1 |
| 11 |      | 77 | 0B |    | REL1 |
| 12 |      | 0B | 77 |    |      |
| 13 |      | A8 | 77 |    |      |
| 14 |      | C8 | 77 |    |      |
| 15 |      | 0B | 70 |    |      |
| 16 |      | A8 | 70 |    |      |
| 17 |      | C8 | 70 |    |      |
| 18 |      | 0B | 6D |    | TRA. |
| 19 |      | 05 | 00 |    |      |
| 20 |      | 0D | 20 | 74 | AGF  |
| 21 |      | CD | F7 | A1 |      |
| 22 |      | FB | 78 |    |      |
| 23 |      | 1F | 18 | 00 |      |

Your examination of this solution should take place only after completion of step 5 On the preceding page.

SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

NOTE: Program RELOC3B is located on the next page.

## LISTING

Program RELOC3B

|    | ADDRS | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |                 | COMMENT              |
|----|-------|------|----|----|---------|----------------------|-----------------|----------------------|
|    |       | B0   | B1 | B2 |         | OPCODE               | OPERANDS        |                      |
| 1  | 178A  |      |    |    | SRCLAD  | RES                  | 2               |                      |
| 2  | C     |      |    |    | DESTUAD | RES                  | 2               |                      |
| 3  | 8E    |      |    |    | BYTECT  | RES                  | 2               |                      |
| 4  | 1790  | XX   | XX |    | CALSRC  |                      |                 |                      |
| 5  | 2     | XX   | XX |    | DESTLAD |                      |                 |                      |
| 6  | 4     | 77   | 0B |    | RELOC3  | PPSI                 | WC, COM, C      |                      |
| 7  | 6     | 0B   | 75 |    |         | LDDR, R0             | > DESTUAD       |                      |
| 8  | 8     | AB   | 75 |    |         | SUBR, R0             | > BYTECT        |                      |
| 9  | A     | C8   | 77 |    |         | STRB, R0             | > DESTLAD       | DESTAD EQU DEST+1    |
| 10 | C     | 0B   | 6E |    |         | LDDR, R0             | < DESTUAD       |                      |
| 11 | 9E    | AB   | 6E |    |         | SUBR, R0             | < BYTECT        |                      |
| 12 | AD    | C8   | 70 |    |         | STRB, R0             | < DESTLAD       |                      |
| 13 | 2     | 09   | 66 |    |         | LDDR, R1             | < SRCLAD        |                      |
| 14 | 4     | 0B   | 65 |    |         | LDDR, R0             | > SRCLAD        |                      |
| 15 | 6     | 58   | 02 |    |         | BRNR, R0             | \$+4            |                      |
| 16 | 8     | A5   | 01 |    |         | SUBT, R1             | 1               |                      |
| 17 | A     | A4   | 01 |    |         | SUBT, R0             | 1               |                      |
| 18 | C     | C8   | 63 |    |         | STRB, R0             | > CALSRC        | CALSRC+1 EQU AGAIN+2 |
| 19 | AE    | C9   | 60 |    |         | STRB, R1             | < CALSRC        | CALSRC EQU AGAIN+1   |
| 20 | BD    | 0B   | 5D |    |         | LDDR, R3             | > BYTECT        |                      |
| 21 | 2     | 05   | 00 |    |         | LODT, R1             | 0               |                      |
| 22 | 4     | 0D   | B7 | 90 | AGAIN   | LDDA, R0             | * CALSRC, R1, + |                      |
| 23 | 7     | CD   | F7 | 92 | DEST    | STRA, R0             | * DESTAD, R1    |                      |
| 24 | A     | FB   | 7B |    |         | BDRR, R3             | AGAIN           |                      |
| 25 | C     | 1E   | 1B | 00 |         | BCTA, UN             | MONITOR         |                      |
| 26 |       |      |    |    |         |                      |                 |                      |
| 27 |       |      |    |    |         |                      |                 |                      |

INTRODUCTION:

You'll need programs such as the following examples to enter specific data patterns into memory when required. For example, in memory diagnostics, many designers write routines to check that all RAM bits set, then clear, in any combination.

EXAMPLE 1: To set a fixed pattern into User RAM; X locations starting at location 'YYYY'.

This routine sets 'FF' into 93 bytes, starting at loc '0176'

|   |      |    |    |    |        |          |                  |                               |
|---|------|----|----|----|--------|----------|------------------|-------------------------------|
| 1 | 0000 | 04 | FF |    | SETMFI | LODI, R0 | PATTERN          | Set desired pattern into R0,  |
| 2 | 2    | 07 | 00 |    |        | LODI, R3 | 0                | then zero R3 index and        |
| 3 | 4    | 06 | 5D |    |        | LODI, R2 | 93               | set # of bytes to be load-    |
| 4 | 6    | CF | 21 | 75 | AGAIN  | STRA, R0 | STRTADD-1, R3, + | ed, 93, is '5D'. Now exec-    |
| 5 | 9    | FA | 7B |    |        | BDRR, R2 | AGAIN            | ute pattern load when         |
| 6 | 8    | 1F | 1B | 00 |        | BCTA, UN | MONITOR          | Finished, exit to the MONITOR |

NOTE: This routine is particularly useful when extensive use of Memory-mapped I/O control pattern generation is necessary.

EXAMPLE 2: To set a fixed pattern (XX) into one or more non-contiguous 256-byte blocks in user RAM.

This routine loads the pattern 'D5' into the first 512 bytes of user RAM.

|    |      |    |    |    |         |          |              |                        |
|----|------|----|----|----|---------|----------|--------------|------------------------|
| 10 | 1780 | 20 |    |    | BLDCPAT | EORZ     | R0           | Zero R0 and set R3     |
| 11 | 1    | C3 |    |    |         | STRZ     | R3           | index to zero then set |
| 12 | 2    | 04 | 00 |    |         | LODI, R0 | XX           | R0 to any value. Store |
| 13 | 4    | CF | 20 | 00 |         | STRA, R0 | LO256, R3, + | what was loaded into   |
| 14 | 7    | 5B | 7B |    |         | BRNR, R3 | \$-3         | R0 in LO 256 BYTES     |
| 15 | 9    | CF | 21 | 00 |         | STRA, R0 | H1256, R3, + | repeat for H1256       |
| 16 | C    | 5B | 7B |    |         | BRNR, R3 | \$-3         | Bytes. Done? Yes!      |
| 17 | 178E | 1F | 1B | 00 |         | BCTA, UN | MONITOR      | Exit to monitor        |

NOTE: If R0 = 00, memory is cleared.

EXAMPLE 3: To set an incrementing pattern into 1 or more contiguous bytes of user RAM.

This routine loads an incrementing pattern, initially 'E5', into 116 memory bytes, starting at location '0037'.

|    |     |    |    |    |         |          |                  |                          |
|----|-----|----|----|----|---------|----------|------------------|--------------------------|
| 1  | 1D0 | 20 |    |    | INCPTN1 | EORZ     | R0               | clear R0 then set index  |
| 2  | 1   | C2 |    |    |         | STRZ     | R2               | to 0. Now subtract byte  |
| 3  | 2   | 07 | 74 |    |         | LODI, R3 | 116              | count from 0, and store  |
| 4  | 4   | A3 |    |    |         | SUBZ     | R3               | in R3. (116, is H'74')   |
| 5  | 5   | C3 |    |    |         | STRZ     | R3               | Then, set R0 equal to    |
| 6  | 6   | 04 | E5 |    |         | LODI, R0 | H'E5'            | pattern. Store it in 1st |
| 7  | 8   | CE | 20 | 36 |         | STRA, R0 | STRTADD-1, R2, + | location, then increment |
| 8  | B   | 84 | 01 |    |         | ADDI, R0 | 1                | the pattern when X bytes |
| 9  | D   | DB | 79 |    |         | BIRR, R3 | AGAIN            | have been stored, exit   |
| 10 | 1DF | 1F | 1B | 00 |         | BCTA, UN | MONITOR          | to monitor.              |



- uses BDRR instr. instead of BIRR, SUBZ, and STRZ. saves 2 bytes in Memory

This routine loads an incrementing pattern, initially '63', into 57 memory bytes, starting at location '00D0'

|    |     |    |    |    |        |          |                  |                                |
|----|-----|----|----|----|--------|----------|------------------|--------------------------------|
| 15 | IFD | 06 | 00 |    | INCPN2 | LODI, R2 | 0                | clear index, then set byte     |
| 16 | 2   | 07 | 39 |    |        | LODI, R3 | 57               | count, and initialize the      |
| 17 | 4   | 04 | 63 |    |        | LODI, R0 | H'63'            | first pattern. Now store       |
| 18 | 6   | CE | 20 | CF |        | STRA, R0 | STRTADD-1, R2, + | in selected memory, then       |
| 19 | 9   | 84 | 01 |    |        | ADDI, R0 | 1                | increment by 1. Repeat for     |
| 20 | 8   | FB | 79 |    |        | BDRR, R3 | AGAIN            | total byte count, then exit to |
| 21 | D   | 1F | 18 | 00 |        | BCTA, UN | MONITOR          | monitor.                       |

Note the coding for STRTADDR-1.

SSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

### APPLICATIONS MEMO

At power-up the status of the 2650 is undefined. The Reset signal should be raised for at least three clock periods. This forces execution of the instruction at location 0. Once the system is started up, the first program to run is generally responsible for initializing the microprocessor, memory, and I/O devices to their desired initial states. The type of I/O initialization is dependent on the particular device. Contents of RAM are undefined at power-up and must be set to their desired initial states.

#### Program status word initialization:

1. Interrupts can be inhibited as a first step in initialization. The Reset clears the Interrupt Inhibit bit and the internal Interrupt Waiting signal. After the remainder of the status bits, the memory, and the I/O is initialized, interrupts can be permitted. This procedure will prevent unwanted interrupts during system initialization. If the system does not utilize interrupts, the Interrupt Inhibit bit can be left set on when system initialization is complete. This approach will assure that a spurious interrupt will not occur.
2. The Stack Pointer may be initialized to zero. The Stack Pointer should not be modified during the execution of a program. This pointer is under the control of the processor. Modification by a program could have unwanted results, i.e., to the instruction address register.
3. It is generally unnecessary to initialize the Condition Code, Interdigit Carry, Overflow, and Carry bits. These bits are normally set by arithmetic and logical operations before they are tested. However, if the With Carry bit is set on, then the Carry bit should be initialized correctly for the first arithmetic instruction.
4. The Register Select bit should be set to a known state, e.g. if bank 1 registers are reserved for interrupt routines, the register select bit should be initialized to bank 0.
5. The With Carry bit can be initialized to the state desired for most arithmetic and rotate operations. Then if a different state is desired for some operations, the With Carry bit can be changed and then restored after these operations.
6. The same philosophy used for the With Carry bit also applies to the Compare bit. Set the Compare bit initially to the most frequent types of compares made, logical or arithmetic.
7. The Sense bit cannot be modified by a program. The Flag bit may need to be initialized if there is a device connected to it such as a TTY which needs stop bits (binary one) when not receiving data.

## SUMMARY

This applications memo describes several methods of testing the contents of the internal registers in the Signetics 2650 Microprocessor.

The following test examples are given:

- Specific bit(s) in a register.
- Positive, negative, or zero-contents of a register.
- Contents of a register compared with a value (equals, greater than, or less than).
- Interdigit-carry (IDC), overflow (OVF), and carry (C) flags in the program status word.

## INTRODUCTION

As a result of an operation on register(s) of the 2650 register bank, five bits (bits 7, 6, 5, 2, and 0) in the Program Status Lower (PSL) portion of the Program Status Word (PSW) register can be affected.

|     |     |     |    |    |     |     |   |
|-----|-----|-----|----|----|-----|-----|---|
| 7   | 6   | 5   | 4  | 3  | 2   | 1   | 0 |
| CC1 | CC0 | IDC | RS | WC | OVF | COM | C |

PROGRAM STATUS LOWER (PSL)

These bits are affected as follows:

**CC1, CC0: Condition Code Bits**

| CONDITION CODE |     | RESULT OF                                    |                     |                                             |
|----------------|-----|----------------------------------------------|---------------------|---------------------------------------------|
|                |     | LOAD/STORE, ARITHMETIC, LOGICAL INSTRUCTIONS | COMPARE INSTRUCTION | SELECTIVE TESTS ON BITS (TMI, TPSU, & TPSL) |
| CC1            | CC0 |                                              |                     |                                             |
| 0              | 0   | Zero                                         | Equal               | All bits 1                                  |
| 0              | 1   | Positive                                     | Greater Than        | —                                           |
| 1              | 0   | Negative                                     | Less Than           | Not all bits 1                              |

## IDC: Interdigit Carry/Borrow Bit

The IDC bit is affected by arithmetic operations as well as rotation.

- 0 = Interdigit borrow/no interdigit carry
- 1 = Interdigit carry/no interdigit borrow

## OVF: Overflow Bit. Arithmetic Operation

The overflow bit in arithmetic operations is set as follows:

$$\text{Operand 1} \pm \text{Operand 2} \rightarrow \text{Result}$$

| SIGN      |           |        | ADD OVF | SUB OVF |
|-----------|-----------|--------|---------|---------|
| OPERAND 1 | OPERAND 2 | RESULT |         |         |
| +         | +         | +      | 0       | 0       |
| +         | +         | -      | 1       | 0       |
| +         | -         | +      | 0       | 0       |
| +         | -         | -      | 0       | 1       |
| -         | +         | +      | 0       | 1       |
| -         | +         | -      | 0       | 0       |
| -         | -         | +      | 1       | 0       |
| -         | -         | -      | 0       | 0       |

## OVF: Overflow Bit. Rotate Operation

Condition: WC = 1; if WC = 0, the OVF bit is not affected.

The overflow bit is set as follows:

| OPERAND SIGN  |              | OVF |
|---------------|--------------|-----|
| BEFORE ROTATE | AFTER ROTATE |     |
| +             | +            | 0   |
| +             | -            | 1   |
| -             | +            | 0   |
| -             | -            | 0   |

## C: Carry/Borrow Bit

The Carry bit is affected by arithmetic operations as well as rotation.

- 0 = borrow/no carry
- 1 = carry/no borrow

## BIT TESTING PROCEDURES

The bits of a register Rx (register zero Ro or any register R1, R2 or R3 in the selected register bank) can be tested as follows:

### TEST FOR '0' IN BIT 3 OF Rx

|         |       |                           |   |   |
|---------|-------|---------------------------|---|---|
| TMI, Rx | H'08' | 1)                        | 2 | 3 |
| BCTR, 2 | LBL   | *Branch if bit 3 is zero. | 2 | 3 |
|         |       |                           | 4 | 6 |

or:

|          |       |                           |   |   |
|----------|-------|---------------------------|---|---|
| ANDI, Rx | H'08' | 2)                        | 2 | 2 |
| BCTR, 0  | LBL   | *Branch if bit 3 is zero. | 2 | 3 |
|          |       |                           | 4 | 5 |

While the second test is faster, it affects the contents of Rx.

## BIT TESTING PROCEDURES (Continued)

## TEST FOR '1' IN BIT 3 OF Rx

|         |       |                          |   |   |
|---------|-------|--------------------------|---|---|
| TMI, Rx | H'08' | 1)                       | 2 | 3 |
| BCTR, 0 | LBL   | *Branch if bit 3 is one. | 2 | 3 |
|         |       |                          | 4 | 6 |

or:

|          |       |                          |   |   |
|----------|-------|--------------------------|---|---|
| ANOI, Rx | H'08' | 2)                       | 2 | 2 |
| BCFR, 0  | LBL   | *Branch if bit 3 is one. | 2 | 3 |
|          |       |                          | 4 | 5 |

While the second test is faster, it affects the contents of Rx.

## TEST FOR '0' IN BIT 1 OR BIT 3 OR BIT 6 OF Rx

|         |       |                                            |   |   |
|---------|-------|--------------------------------------------|---|---|
| TMI, Rx | H'4A' | 1)                                         | 2 | 3 |
| BCTR, 2 | LBL   | *Branch if one of the tested bits is zero. | 2 | 3 |
|         |       |                                            | 4 | 6 |

## TEST FOR '1' IN BIT 1 OR BIT 3 OR BIT 6 OF Rx

|          |       |                                           |   |   |
|----------|-------|-------------------------------------------|---|---|
| ANDI, Rx | H'4A' | 2)                                        | 2 | 2 |
| BCFR, 0  | LBL   | *Branch if one of the tested bits is one. | 2 | 3 |
|          |       |                                           | 4 | 5 |

## TEST FOR '0' IN BIT 1 AND BIT 3 AND BIT 6 OF Rx

|          |       |                                      |   |   |
|----------|-------|--------------------------------------|---|---|
| ANOI, Rx | H'4A' | 2)                                   | 2 | 2 |
| BCTR, 0  | LBL   | *Branch if all tested bits are zero. | 2 | 3 |
|          |       |                                      | 4 | 5 |

## TEST FOR '1' IN BIT 1 AND BIT 3 AND BIT 6 OF Rx

|         |       |                                     |   |   |
|---------|-------|-------------------------------------|---|---|
| TMI, Rx | H'4A' | 1)                                  | 2 | 3 |
| BCTR, 0 | LBL   | *Branch if all tested bits are one. | 2 | 3 |
|         |       |                                     | 4 | 6 |

## TEST FOR PATTERN IN Rx; e.g., x10xx01x

x = don't care

|          |       |                             |   |   |
|----------|-------|-----------------------------|---|---|
| IORI, Rx | H'99' | 2)                          | 2 | 2 |
| COMI, Rx | H'DB' |                             | 2 | 2 |
| BCTR, 0  | LBL   | *Branch if pattern is true. | 2 | 3 |
|          |       |                             | 6 | 7 |

## BYTE TESTING PROCEDURES

## TEST FOR POSITIVE, NEGATIVE AND ZERO

All of the tests described below must be preceded by an operation on Rx which updates the contents of the condition register, e.g., by instructions such as LOAD, ADD, AND, COMPARE, ROTATE, I/O, etc.

|                        | CC       | OPERATION |
|------------------------|----------|-----------|
| Test for $(Rx) \geq 0$ | 00 or 01 | BCFR, 2   |
| Test for $(Rx) > 0$    | 01       | BCTR, 1   |
| Test for $(Rx) = 0$    | 00       | BCTR, 0   |
| Test for $(Rx) < 0$    | 10       | BCTR, 2   |
| Test for $(Rx) \leq 0$ | 00 or 10 | BCFR, 1   |

## TESTS ON THE CONTENTS OF A REGISTER BY USING COMPARE INSTRUCTIONS

Logical compare: (COM = 1 in PSL)

Comparison is made between two 8-bit unsigned binary numbers.

Arithmetic compare: (COM = 0 in PSL)

Comparison is made between two 8-bit signed numbers.

After execution of the logic or arithmetic compare instruction, the condition register (CC) is set to a specific value and tested as follows:

| REGISTER-TO-REGISTER COMPARE |          |         |
|------------------------------|----------|---------|
| Instruction used:<br>COMZ Rx |          |         |
| RESULT                       | CC       | TEST    |
| $(Ro) \geq (Rx)$             | 00 or 01 | BCFR, 2 |
| $(Ro) > (Rx)$                | 01       | BCTR, 1 |
| $(Ro) = (Rx)$                | 00       | BCTR, 0 |
| $(Ro) < (Rx)$                | 10       | BCTR, 2 |
| $(Ro) \leq (Rx)$             | 00 or 10 | BCFR, 1 |

| REGISTER TO CONSTANT OR MEMORY LOCATION                                                                |          |         |
|--------------------------------------------------------------------------------------------------------|----------|---------|
| Instructions used:<br>COMI, Rx DATA<br>COMR, Rx RELATIVE LOCATION OF DATA<br>COMA, Rx LOCATION OF DATA |          |         |
| RESULT<br>V=VALUE                                                                                      | CC       | TEST    |
| $(Rx) \geq V$                                                                                          | 00 or 01 | BCFR, 2 |
| $(Rx) > V$                                                                                             | 01       | BCTR, 1 |
| $(Rx) = V$                                                                                             | 00       | BCTR, 0 |
| $(Rx) < V$                                                                                             | 10       | BCTR, 2 |
| $(Rx) \leq V$                                                                                          | 00 or 10 | BCFR, 1 |

- 1) Contents of register Rx kept  
2) Contents of register Rx lost

Whenever a compare instruction is used, the IDC, OVf, or C bits in the PSL are *not* affected.

## SIGNETICS BIT AND BYTE TESTING PROCEDURES ■ AS51

### TEST ON OVERFLOW (OVF in PSL)

The overflow bit is affected whenever arithmetic or rotate instructions are executed.

The *OVF bit* is set during an addition whenever the two operands have the same sign and the result has a different sign. During a subtraction, the *OVF bit* is set when the operands differ in sign and the result has a different sign than the first operand.

**Examples:**

|                      |     |
|----------------------|-----|
| $(+A) + (+B) = (-C)$ | OVF |
| $(-A) + (-B) = (+C)$ | OVF |
| $(+A) - (-B) = (-C)$ | OVF |
| $(-A) - (+B) = (+C)$ | OVF |

**Test:**   TPSL    H'04'   \*OVF test  
          BCTR, 0   LBL    \*Branch if OVF = set

The *OVF bit* is set during rotate instructions with WC = 1 whenever the sign changes from positive to negative. If WC = 0, then rotate instructions do not affect the OVF bit.

**Example:**

|               |                      |
|---------------|----------------------|
| RRR, Rx       | *Rotate right        |
| TPSL    H'04' | *Test OVF bit        |
| BCTR, 0   LBL | *Branch if OVF = set |

### TEST ON CARRY (C in PSL)

The carry bit is set to 1 by an add instruction that generates a carry and a sub-instruction that does *not* generate a borrow.

**Example:**

#### ADDITION

|          |       |                  |
|----------|-------|------------------|
| LODI, Rx | H'88' |                  |
| ADDI, Rx | H'99' |                  |
| TPSL     | H'01' | *Test carry      |
| BCTR, 0  | LBL   | *Branch if carry |

#### SUBTRACTION

|          |       |                             |
|----------|-------|-----------------------------|
| LODI, Rx | H'40' |                             |
| SUBI, Rx | H'30' |                             |
| TPSL     | H'01' | *Test borrow                |
| BCTR, 0  | LBL   | *Branch if <i>no</i> borrow |

When a rotate instruction is executed with WC = 1, the carry bit is also affected. Refer to the Signetics 2650 Microprocessor manual for a description of this operation.

#### SUMMARY

In microprocessing applications, delay times are often required. A typical example is a delay time for a serial Teletypewriter interface. While delay times can be generated by counters, monostables, multivibrators, and other hardware, it is often simpler and more economical to use a short software routine.

This applications memo describes several ways of writing software delay time routines for the Signetics 2650 microprocessor. Time restrictions and formulas for calculating the delay time are given for each routine.

#### DELAY ROUTINES

In general, a delay can be implemented by setting a counter with a number  $N$  and decrementing this number by one until it is zero. If decrementing the number takes one clock period, then the total delay time is  $N$  clock periods.

In the 2650 microprocessor, the internal registers may be used as counters. The most useful instructions for decrementing are the "Branch on Decrementing Register" (BDRR and BDRA) instructions, which also test the content of a register for zero.

Figure 1 illustrates a flowchart of a delay routine. This routine consists of a setup part and a count loop. The count loop will be executed  $n$  times and the setup only once. Hence, the delay time is:

$$t_d = t_{su} + n \cdot t_{ct}$$

It is possible to increase the delay time by increasing  $n$  or by making  $t_{ct}$  longer. The latter can be done by inserting a fixed delay such as a No Operation (NOP) instruction in the count loop.

#### DELAY ROUTINE FLOWCHART

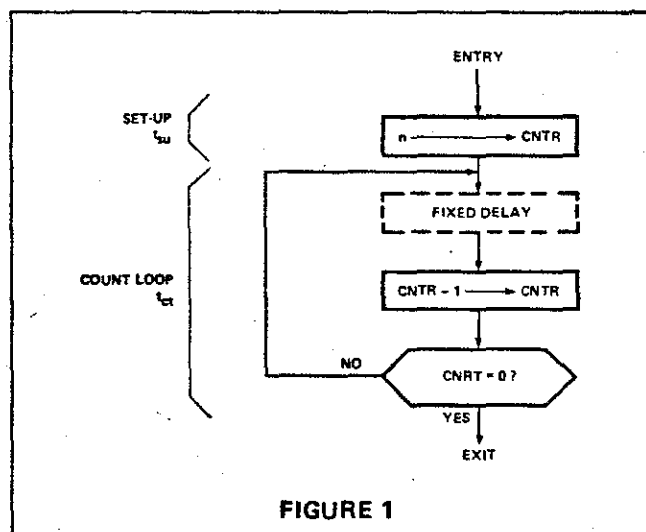


FIGURE 1

The program of the routine shown in Figure 1 is as follows:

|               |                                                        |       |
|---------------|--------------------------------------------------------|-------|
| LODI, Rx n    | Load n into register Rx                                | 6 cp* |
| LOOP NOP      | No operation; fixed delay of 6 cp                      | 6 cp  |
| BDRR, Rx LDOP | Decrement Rx; branch to loop if the result is not zero | 9 cp  |

\*cp = clock periods

With one NOP, the delay time is:  $t_d = (6 + 15 \cdot n)$  cp. Without the NOP, the delay time is:  $t_d = (6 + 9 \cdot n)$  cp. The maximum delay time is obtained when Rx is loaded with zero, since Rx will cycle through all the 256 possible states. When  $Rx = R0$ , the LODI, R0 0 instruction can be replaced by the EORZ R0 instruction, which saves one byte of code.

#### DELAY ROUTINE WITH FOUR REGISTERS

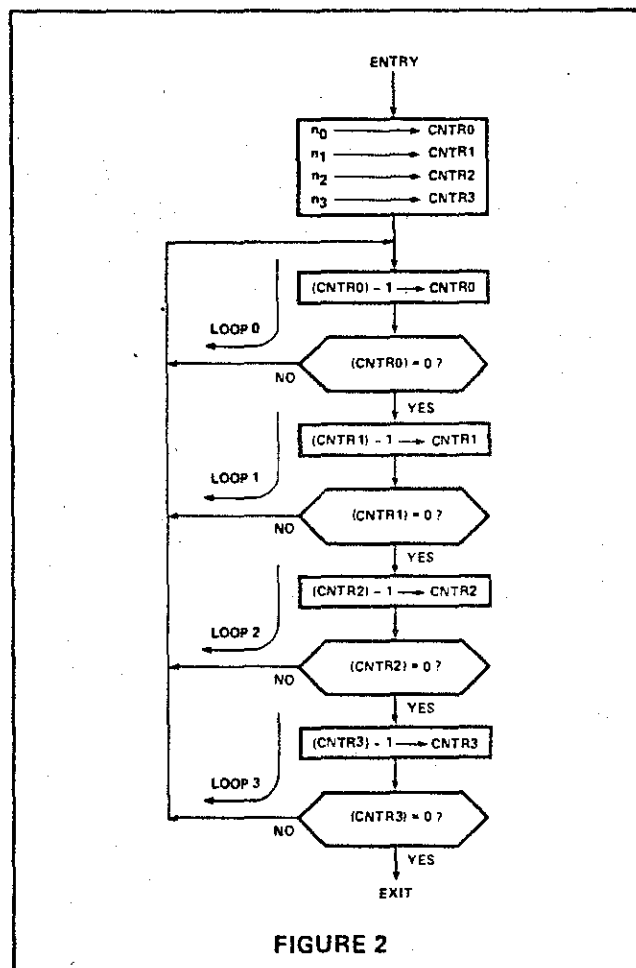


FIGURE 2

Another possible way of increasing the delay time is to repeat the count loop of Figure 1 several times. This can be done by repeating the instructions or by counting the repetitions of the count loop in another register. For example, this latter method can be expanded to include four internal registers. A flowchart of a delay routine using this technique is illustrated in Figure 2.

The number of times the processor executes the different loops shown in Figure 2 are:

loop 3  $n_3$   
 loop 2  $n_2 + (n_3 - 1) 256$   
 loop 1  $n_1 + (n_2 - 1) 256 + (n_3 - 1) 256^2$   
 loop 0  $n_0 + (n_1 - 1) 256 + (n_2 - 1) 256^2 + (n_3 - 1) 256^3$

Hence, the delay time of this routine is:

$$t_d = [24 + \{n_0 + n_1 + (n_1 - 1) 256 + n_2 + (n_2 - 1) (256 + 256^2) + n_3 + (n_3 - 1) (256 + 256^2 + 256^3)\} 9] \text{ cp}$$

(If Rx is loaded with a zero, then  $n = 256$  in the formula):

Table 1 shows six different delay routine programs along with specifications for each program. The delay time for these routines can be computed from the following equations.

| Routine | Delay Time                                                                                                                        |
|---------|-----------------------------------------------------------------------------------------------------------------------------------|
| a       | $t_d = (6 + 9 \cdot n_0) \text{ cp}$                                                                                              |
| b       | $t_d = (6 + 15 \cdot n_0) \text{ cp}$                                                                                             |
| c       | $t_d = (2310 + 9 \cdot n_0) \text{ cp}$                                                                                           |
| d       | $t_d = \{12 + [n_0 + n_1 + (n_1 - 1) 256] 9\} \text{ cp}$                                                                         |
| e       | $t_d = \{18 + [n_0 + n_1 + (n_1 - 1) 256 + n_2 + (n_2 - 1) (256^2 + 256)] 9\} \text{ cp}$                                         |
| f       | $t_d = \{24 + [n_0 + n_1 + (n_1 - 1) 256 + n_2 + (n_2 - 1) (256^2 + 256) + n_3 + (n_3 - 1) (256^3 + 256^2 + 256)] 9\} \text{ cp}$ |

TABLE 1

| ROUTINE | POSSIBLE DELAY TIME (cp) |                                | DELAY STEP (cp) | NUMBER OF BYTES | NUMBER OF REGISTERS | PROGRAM                                                                                                                                       |
|---------|--------------------------|--------------------------------|-----------------|-----------------|---------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
|         | MIN*                     | MAX                            |                 |                 |                     |                                                                                                                                               |
| a       | 15                       | 2310                           | 9               | 4               | 1                   | LODI, R0 $n_0$<br>LOOP BDRR, R0 LOOP                                                                                                          |
| b       | 21                       | 3846                           | 15              | 5               | 1                   | LODI, R0 $n_0$<br>LOOP NOP<br>BDRR, R0 LOOP                                                                                                   |
| c       | 2319                     | 4614                           | 9               | 6               | 1                   | LODI, R0 $n_0$<br>LOP 1 BDRR, R0 LOP 1<br>LOP 2 BDRR, R0 LOP 2                                                                                |
| d       | 30                       | 592.140                        | 9               | 8               | 2                   | LODI, R0 $n_0$<br>LODI, R1 $n_1$<br>LOOP BDRR, R0 LOOP<br>BDRR, R1 LOOP                                                                       |
| e       | 45                       | $\approx 151.6 \times 10^6$ ** | 9               | 12              | 3                   | LODI, R0 $n_0$<br>LODI, R1 $n_1$<br>LODI, R2 $n_2$<br>LOOP BDRR, R0 LOOP<br>BDRR, R1 LOOP<br>BDRR, R2 LOOP                                    |
| f       | 60                       | $\approx 38.8 \times 10^9$ *** | 9               | 16              | 4                   | LODI, R0 $n_0$<br>LODI, R1 $n_1$<br>LODI, R2 $n_2$<br>LODI, R3 $n_3$<br>LOOP BDRR, R0 LOOP<br>BDRR, R1 LOOP<br>BDRR, R2 LOOP<br>BDRR, R3 LOOP |

\* cp = clock period. For 1MHz clock 1 cp = 1 $\mu$ s.

\*\* For 1MHz clock this is about 2.5 minutes.

\*\*\* For 1MHz clock this is about 10.46 hours.

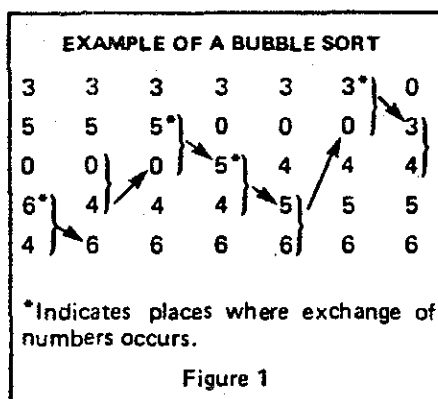
## **INTRODUCTION**

Sorting routines are often required as part of the software design for microprocessor-based systems. This applications memo provides several examples for implementing sort routines on the 2650 microprocessor. These examples include routines for sorting single byte and multiple byte numbers, both signed and unsigned, in fixed or variable length lists. The techniques demonstrated are the "bubble" sort, the "search" sort, and the "linear" sort.

## **BUBBLE SORT**

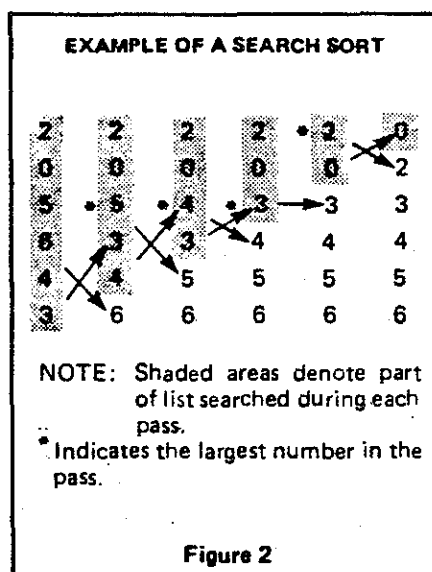
An easy way of sorting is to compare two of the listed numbers at a time and exchange them when they are not in the right sequence. One such sorting technique is known as the "bubble" sort. Bubble sorts normally take the longest time to execute.

In a bubble sort, the last two numbers in the list are compared and exchanged if they are not in the right sequence. Then the next to the last number is compared with the number above it and exchanged if they are not in the right sequence. This process continues as each number in turn is compared with those above it. Each time a pair of numbers is exchanged, the order of searching is reversed (now going towards the end of the list), and the part of the list which has already been sorted is examined pair by pair until the larger number is in its proper position. The sort then resumes (in the original direction, towards the top of the list) at the point in the list where the larger number was found. This bottom-to-top (and reverse order) comparison cycle is repeated until all the numbers in the list are in the right sequence. The execution time is proportional to the number of exchanges required. An example of a bubble sort is shown in Figure 1.



## **SEARCH SORT**

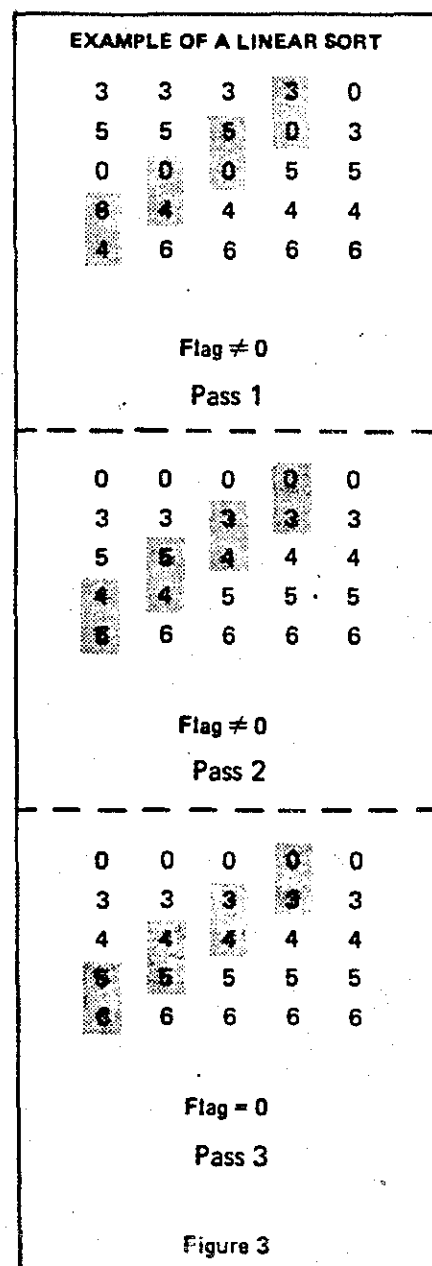
Another way of sorting is to search the list each time for the number with the largest value and then insert this number in the right place in the list. This sorting technique is known as the "search" sort. First, all the numbers are tested, and the largest-valued number and the last number in the list are exchanged. The list length is then decremented by one, and the list is searched again for the next largest-valued number. This process continues until all the numbers in the list are in the right sequence, as shown in Figure 2.



The execution time of a search sort is proportional to the list length. On very unsorted lists, the search sort requires less time to execute than the bubble sort.

## **LINEAR SORT**

A "linear" sort consists of several passes. In each pass, the list is examined from the bottom upwards. Two numbers are compared at a time, and the number with the larger value is placed at the bottom of the pair. Any exchange of numbers sets a flag. When a pass is finished, the program tests the flag and, if the flag is set, it clears the flag and begins a new pass. The sort is completed when the flag is not set at the end of a pass. In some cases, linear sorts can be executed faster than bubble or search sorts. An example of a linear sort is given in Figure 3.



## **REMARKS FOR SAMPLE PROGRAMS**

The sample programs below illustrate the use of the techniques described previously to sort various types of lists. The programs sort the numbers into ascending order. This can be modified to descending order by changing the test instructions marked with a '#' in the comment column from the Greater Than (GT) to Less Than (LT) and vice versa.

Figure 4 defines the symbols used in all of the example programs.



## DEFINITION OF SYMBOLS

```

0001 *****
0002 * DEFINITIONS OF SYMBOLS
0003 * REGISTER EQUATES
0004 0000 R0 EQU 0 REGISTER 0
0005 0001 R1 EQU 1 REGISTER 1
0006 0002 R2 EQU 2 REGISTER 2
0007 0003 R3 EQU 3 REGISTER 3
0008 * CONDITION CODES
0009 0001 P EQU 1 POSITIVE RESULT
0010 0000 Z EQU 0 ZERO RESULT
0011 0002 N EQU 2 NEGATIVE RESULT
0012 0002 LT EQU 2 LESS THAN
0013 0000 EQ EQU 0 EQUAL TO
0014 0001 GT EQU 1 GREATER THAN
0015 0003 UN EQU 3 UNCONDITIONAL
0016 * PSW LOWER EQUATES
0017 0000 CC EQU H'00' CONDITIONAL CODES
0018 0020 IDC EQU H'20' INTERDIGIT CARRY
0019 0010 RS EQU H'10' REGISTER BANK
0020 0000 WC EQU H'00' 1=WITH 0=WITHOUT CARRY
0021 0004 OVF EQU H'04' OVERFLOW
0022 0002 COM EQU H'02' 1=LOGIC 0=ARITHMETIC COMPARE
0023 0001 C EQU H'01' CARRY/BORROW
0024 * PSW UPPER EQUATES
0025 0000 SENS EQU H'00' SENSE BIT
0026 0040 FLAG EQU H'40' FLAG BIT
0027 0020 II EQU H'20' INTERRUPT INHIBIT
0028 0007 SP EQU H'07' STACK POINTER
0029 * END OF EQUATES

```

FIGURE 4

## FLOW CHART FOR FIXED-LIST BUBBLE SORT

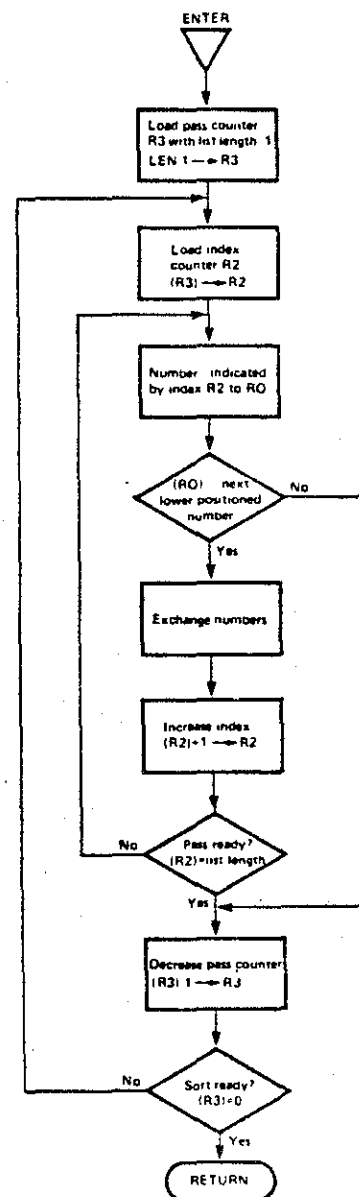


Figure 5

### Program Title

**BUBBLE SORT FOR A FIXED LIST**

### Function

This program sorts single-byte numbers (signed or unsigned) into their incrementing order. The bytes are held in a list with a fixed address and a fixed length.

### Parameters

#### Input:

Unsorted list.

The compare flag indicates if the numbers are signed or unsigned.

COM=1 means unsigned numbers.

COM=2 means signed numbers.

#### Output:

Sorted list.

Refer to Figures 5 and 6 for flowchart and program listing.

| HARDWARE AFFECTED |         |         |         |         |          |          |          | RAM REQUIRED (BYTES):                 |  | NONE         |
|-------------------|---------|---------|---------|---------|----------|----------|----------|---------------------------------------|--|--------------|
| REGISTERS         | R0<br>X | R1<br>X | R2<br>X | R3<br>X | R1'<br>X | R2'<br>X | R3'<br>X | ROM REQUIRED (BYTES):                 |  | 32           |
| PSU               | F       | II      | SP      |         |          |          |          | EXECUTION TIME:                       |  | VARIABLE     |
| PSL               | CC<br>X | IDC     | RS      | WC      | OVF      | COM      | C        | MAXIMUM SUBROUTINE<br>NESTING LEVELS: |  | NONE         |
|                   |         |         |         |         |          |          |          | ASSEMBLER/COMPILER USED:              |  | TWIN VER 1.0 |

## PROGRAM LISTING FOR FIXED-LIST BUBBLE SORT

TWIN ASSEMBLER VER 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```

0021 *
0022 *****
0023 * PD760060 *
0024 *****
0025 * BUBBLE SORT FOR FIXED LIST *
0026 *****
0027 * THIS PROGRAM SORTS A LIST OF SINGLE-BYTE NUMBERS
0028 * INTO THEIR INCREMENTING ORDER.
0029 * THE LIST HAS A FIXED LENGTH AND A FIXED ADDRESS.
0030 * THE MAXIMUM LIST LENGTH IS 256 BYTES.
0031 * UPON ENTRY TO THIS SUBROUTINE, THE COMPARE FLAG
0032 * INDICATES IF THE NUMBERS TO BE SORTED
0033 * ARE SIGNED OR UNSIGNED:
0034 * COM=1 MEANS UNSIGNED NUMBERS.
0035 * COM=0 MEANS SIGNED NUMBERS.
0036 *
0037 .ORG H'500' SORTING SUBROUTINE
0038 SORT LODI,R3 LEN-1 LOAD PASS COUNTER R3
0039 PASS LODZ R3 LOAD INDEX R2
0040 STRZ R2
0041 LOOP LODA,R0 LIST,R2 LOAD FIRST NUMBER IN R0
0042 COMA,R0 LIST-1,R2 COMPARE WITH SECOND NUMBER
0043 BCFR,LT LOC # BRANCH IF THE NUMBERS ARE IN
0044 * THE RIGHT SEQUENCE
0045 STRZ R1 EXCHANGE THE TWO NUMBERS
0046 LODA,R0 LIST-1,R2
0047 STRA,R0 LIST,R2
0048 LODZ R1
0049 STRA,R0 LIST-1,R2
0050 BIRR,R2 $+2 INCREMENT INDEX
0051 COMI,R2 LEN COMPARE INDEX WITH LENGTH
0052 BCFR,EQ LOOP BRANCH IF PASS NOT READY
0053 BDRR,R3 PASS BRANCH IF SORT NOT READY
0054 RETC,UN RETURN TO MAIN PROGRAM
0055 *
0056 *****
0057 * SORTING LIST *
0058 *****
0059 .ORG H'600' LIST
0060 LEN EQU 200 LENGTH OF THE LIST
0061 LIST RES LEN ADDRESS OF THE LIST
0062 *
0063 END SORT

```

TOTAL ASSEMBLY ERRORS = 0000

Figure 6

## Program Title

## SEARCH SORT FOR A FIXED LIST

### Function

This program sorts single-byte numbers (signed or unsigned) into their incrementing order. The bytes are held in the list with a fixed address and fixed length. The maximum list length is 256 bytes.

### Parameters

#### Input:

Unsorted list.

The compare flag indicates if the numbers are signed or unsigned.

COM=1 means unsigned numbers.

COM=0 means signed numbers.

#### Output:

Sorted list.

Refer to Figures 7 and 8 for flowchart and program listing.

FLOW CHART FOR A FIXED-LIST SEARCH SORT

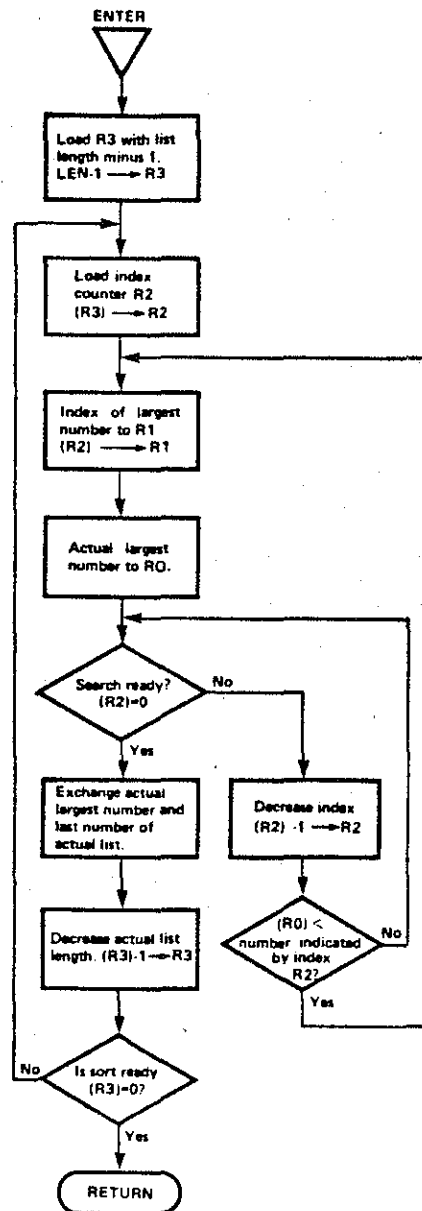


Figure 7

| HARDWARE AFFECTED |         |         |         |         |     |     |     | RAM REQUIRED (BYTES):                 |  | NONE         |  |
|-------------------|---------|---------|---------|---------|-----|-----|-----|---------------------------------------|--|--------------|--|
| REGISTERS         | R0<br>X | R1<br>X | R2<br>X | R3<br>X | R1' | R2' | R3' | ROM REQUIRED (BYTES):                 |  | 32           |  |
| PSU               | F       | II      | SP      |         |     |     |     | EXECUTION TIME:                       |  | VARIABLE     |  |
| PSL               | CC<br>X | IDC     | RS      | WC      | OVF | COM | C   | MAXIMUM SUBROUTINE<br>NESTING LEVELS: |  | NONE         |  |
|                   |         |         |         |         |     |     |     | ASSEMBLER/COMPILER USED:              |  | TWIN VER 1.0 |  |

**PROGRAM LISTING FOR FIXED-LIST SEARCH SORT**

TWIN ASSEMBLER VER. 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```

0031 *
0032 *****
0033 * PD760061 *
0034 *****
0035 * SEARCH SORT FOR A FIXED LIST *
0036 *****
0037 * THIS PROGRAM SORTS A LIST OF SINGLE-BYTE NUMBERS
0038 * INTO THEIR INCREMENTING ORDER.
0039 * THE LIST HAS A FIXED LENGTH AND A FIXED ADDRESS.
0040 * THE MAXIMUM LIST LENGTH IS 256 BYTES.
0041 * UPON ENTRY TO THIS SUBROUTINE, THE COMPARE FLAG
0042 * INDICATES IF THE NUMBERS TO BE SORTED
0043 * ARE SIGNED OR UNSIGNED:
0044 * COM=1 MEANS UNSIGNED NUMBERS
0045 * COM=0 MEANS SIGNED NUMBERS
0046 *
0047 0000 ORG H'500' SORTING SUBROUTINE
0048 0500 07C7 SORT L001,R3 LEN-1 LOAD ACTUAL LIST LENGTH IN R3
0049 0502 02 PASS L002 R3 LOAD INDEX R2
0050 0503 C2 STRZ R2
0051 0504 02 SLEC L002 R2 INDEX OF LARGEST NUMBER TO R1
0052 0505 C1 STRZ R1
0053 0506 006600 LOAD,R0 LIST,R1 LOAD PRESENT LARGEST NUMBER IN R0
0054 0509 5A0E LOOP BRNP,R2 COMP BRANCH IF PASS NOT READY
0055 050E C2 STRZ R2 EXCHANGE LARGEST NUMBER WITH
0056 050C 0F6600 LOAD,R0 LIST,R3 LAST NUMBER IN ACTUAL LIST
0057 050F C16600 STRA,R0 LIST,R1
0058 0512 02 L002 R2
0059 0513 CFE600 STRA,R0 LIST,R3
0060 0516 FB5A BCRP,R3 PASS DECREASE ACTUAL LIST LENGTH
0061 * BRANCH TO NEXT PASS IF
0062 * LENGTH NOT ZERO
0063 0518 17 RETC,UN RETURN TO MAIN PROGRAM
0064 0519 EE4600 COMP COM,R0 LIST,R2 - COMPARE NUMBER WITH PRESENT
0065 * LARGEST NUMBER OF LIST
0066 051C 9A0B BCTR,LT LOOP # BRANCH FOR NEXT NUMBER
0067 051E 1B64 BCTR,UN SLEC BRANCH IF NEW NUMBER IS LARGER
0068 *
0069 *****
0070 * SORTING LIST *
0071 *****
0072 0520 ORG H'600' LIST
0073 06C8 LEN EQU 200 LENGTH OF THE LIST
0074 0600 LIST RES LEN ADDRESS OF THE LIST
0075 0500 END SORT

```

TOTAL ASSEMBLY ERRORS = 0000

**Figure 8**

## Program Title

## BUBBLE SORT FOR A VARIABLE-LENGTH LIST

## Function

This program sorts a list of single-byte numbers into their incrementing order. The maximum list length is 256 bytes.

## Parameters

### Input:

Unsorted list.

R1 contains the high-order address of the list.

R2 contains the low-order address.

R3 contains the list length minus 1.

The compare flag indicates if the numbers are signed or unsigned.

COM=1 means unsigned numbers.

COM=0 means signed numbers.

### Output:

Sorted list.

Refer to Figures 9 and 10 for flowchart and program listing.

FLOWCHART FOR A VARIABLE LENGTH LIST BUBBLE SORT

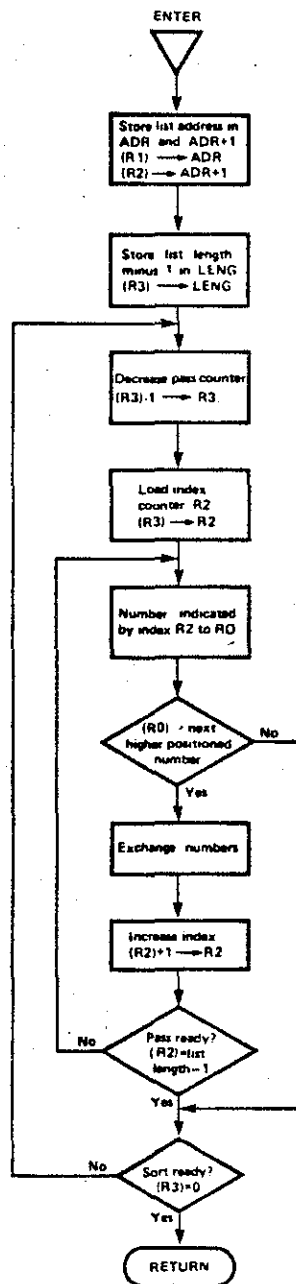


Figure 9

| HARDWARE AFFECTED |    |     |    |    |     |     |     | RAM REQUIRED (BYTES):                 | 3            |
|-------------------|----|-----|----|----|-----|-----|-----|---------------------------------------|--------------|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' | ROM REQUIRED (BYTES):                 | 40           |
| PSU               | F  | II  | SP |    |     |     |     | EXECUTION TIME:                       | VARIABLE     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   | MAXIMUM SUBROUTINE<br>NESTING LEVELS: | NONE         |
|                   | X  |     |    |    |     |     |     | ASSEMBLER/COMPILER USED:              | TWIN VER 1.0 |

**PROGRAM LISTING FOR A VARIABLE LIST BUBBLE SORT**

THIN ASSEMBLER VER 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```

0021 *
0022 *****
0023 * P0760052 *
0024 *****
0025 * BUBBLE SORT FOR VARIABLE LIST *
0026 *****
0027 * THIS PROGRAM SORTS A LIST OF SINGLE-BYTE NUMBERS
0028 * INTO THEIR INCREMENTING ORDER.
0029 * THE ADDRESS AND THE LENGTH OF THE LIST MUST BE
0040 * DEFINED IN THE MAIN PROGRAM. THE MAXIMUM LIST LENGTH
0041 * IS 256 BYTES.
0042 * UPON ENTRY TO THIS SUBROUTINE, THE COMPARE FLAG
0043 * INDICATES IF THE NUMBERS TO BE SORTED
0044 * ARE SIGNED OR UNSIGNED.
0045 * COM=1 MEANS UNSIGNED NUMBERS.
0046 * COM=0 MEANS SIGNED NUMBERS.
0047 *
0048 0000 ORG H'4F0'
0049 04F0 ADR RES 2 ADDRESS OF LIST
0050 04F2 LENG RES 1 LIST LENGTH MINUS 1
0051 *
0052 04F3 ORG H'500' SORTING SUBROUTINE
0053 0500 C004F0 SORT STRA,R1 ADR STORE HIGH ORDER ADDRESS OF
0054 * THE LIST
0055 0502 CE04F1 STRA,R2 ADR+1 STORE LOW ORDER ADDRESS
0056 0506 CF04F2 STRA,R3 LENG STORE LIST LENGTH MINUS 1
0057 0509 FE00 PASS BDRS,R3 $+2 DECREMENT PASS COUNTER
0058 050B 03 LODZ R3 LOAD INDEX
0059 050C C2 STRZ R2
0060 050D 0EE4F0 LOOP LODA,R0 *ADR,R2 FETCH FIRST NUMBER
0061 0510 EEA4F9 COMA,R0 *ADR,R2,+ COMPARE WITH SECOND NUMBER
0062 0513 9910 BCFB GT LOC # BRANCH IF THE NUMBERS ARE IN
* THE RIGHT SEQUENCE
0064 0515 C1 EXCH STRZ R1 EXCHANGE THE TWO NUMBERS
0065 0516 0EE4F0 LODA,R0 *ADR,R2
0066 0519 DEC4F0 STRA,R0 *ADR,R2,-
0067 051C 01 LODZ R1
0068 051D CE04F0 STRA,R0 *ADR,R2,+
0069 0520 EEA4F2 COMA,R2 LENG COMPARE (R2) WITH LENGTH
0070 0522 99F3 BCFB EQ LOOP BRANCH IF PASS NOT READY
0071 0525 5562 BRNB R3 PASS BRANCH IF SORT NOT READY
0072 0527 17 RETC,UN RETURN TO MAIN PROGRAM
0073 *
0074 0500 END SORT

```

TOTAL ASSEMBLY ERRORS = 0000

**Figure 10**

## Program Title

## SEARCH SORT FOR A VARIABLE-LENGTH LIST

### Function

This program sorts a list of single-byte numbers into their incrementing order. The maximum list length is 256 bytes.

### Parameters

#### Input:

Unsorted list.

R1 contains the high-order address of the list.

R2 contains the low-order address.

R3 contains the list length minus 1.

The compare flag indicates if the numbers are signed or unsigned.

COM=1 means unsigned numbers.

COM=2 means signed numbers.

#### Output:

Sorted list.

Refer to Figures 11 and 12 for flow-chart and program listing.

FLOW CHART FOR VARIABLE LENGTH LIST SEARCH SORT

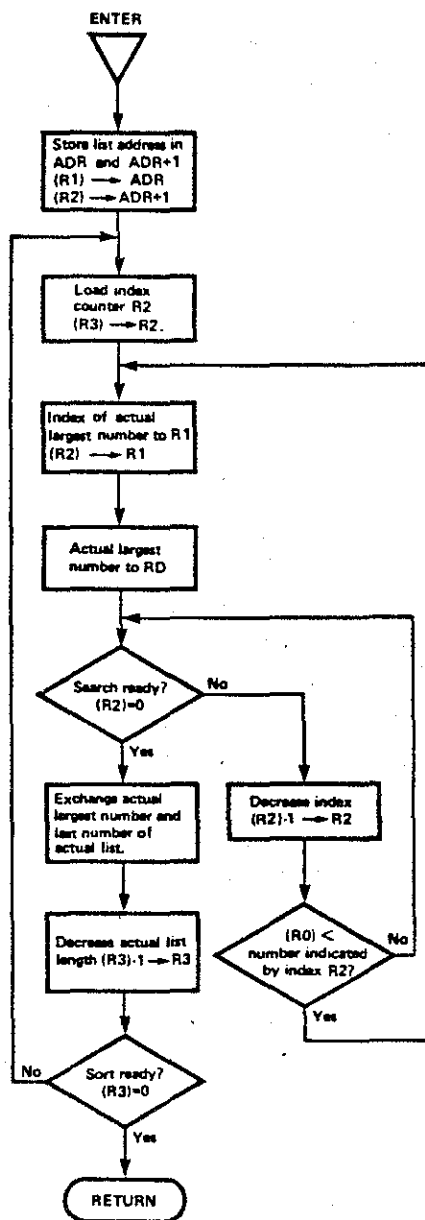


Figure 11

| HARDWARE AFFECTED |    |     |    |    |     |     |     | RAM REQUIRED (BYTES):                 |  | 2            |
|-------------------|----|-----|----|----|-----|-----|-----|---------------------------------------|--|--------------|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' | ROM REQUIRED (BYTES):                 |  | 36           |
|                   | X  | X   | X  | X  |     |     |     | EXECUTION TIME:                       |  | VARIABLE     |
| PSU               | F  | II  | SP |    |     |     |     | MAXIMUM SUBROUTINE<br>NESTING LEVELS: |  | NONE         |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   | ASSEMBLER/COMPILER USED:              |  | TWIN VER 1.0 |
|                   | X  |     |    |    |     |     |     |                                       |  |              |

**PROGRAM LISTING FOR A VARIABLE LIST SEARCH SORT**

TWIN ASSEMBLER VER. 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```

0031 *
0032 *****
0033 * PD760063 *
0034 *****
0035 * SEARCH SORT FOR VARIABLE LIST *
0036 *****
0037 * THIS PROGRAM SORTS A LIST OF SINGLE-BYTE NUMBERS
0038 * INTO THEIR INCREMENTING ORDER.
0039 * THE ADDRESS AND THE LENGTH OF THE LIST MUST BE
0040 * DEFINED IN THE MAIN PROGRAM. THE MAXIMUM LIST LENGTH
0041 * IS 256 BYTES.
0042 * UPON ENTRY TO THIS SUBROUTINE, THE COMPARE FLAG
0043 * INDICATES IF THE NUMBERS TO BE SORTED
0044 * ARE SIGNED OR UNSIGNED:
0045 * COM=1 MEANS UNSIGNED NUMBERS.
0046 * COM=0 MEANS SIGNED NUMBERS.
0047 *
0048 0000 ORG H'4F0'
0049 04F0 ADR RES 2 ADDRESS OF LIST
0050 *
0051 04F2 ORG H'500' SORTING SUBROUTINE
0052 0500 CD04F0 SORT STRA, P1 ADR STORE HIGH ORDER ADDRESS OF
0053 * THE LIST
0054 0503 CD04F1 STRA, P2 ADR+1 STORE LOW ORDER ADDRESS
0055 0506 03 PASS LODZ R3 LOAD INDEX R2
0056 0507 C2 STRZ R2
0057 0508 02 MAXN LODZ R2 INDEX OF LARGEST NUMBER TO R1
0058 0509 C1 STRZ R1
0059 050A 0DE4F0 LODA, R0 *ADR, P1 LOAD PRESENT LARGEST NUMBER IN R0
0060 050B 5A0E SRCH BRNR, R2 COMP BRANCH IF PASS NOT READY
0061 050F C2 STRZ R2 EXCHANGE LARGEST NUMBER WITH
0062 0510 0FE4F0 LODA, P0 *ADR, R3 LAST NUMBER IN ACTUAL LIST
0063 0513 CDE4F0 STRA, P0 *ADR, R1
0064 0516 02 LODZ R2
0065 0517 CFE4F0 STRA, P0 *ADR, R2
0066 051A FB6A BOPR, R3 PASS DECREASE ACTUAL LIST LENGTH,
0067 * BRANCH TO NEXT PASS IF
0068 * LENGTH NOT ZERO
0069 051C 17 RETC, UN RETURN TO MAIN PROGRAM
0070 051D EEC4F0 COMP COMA, P0 *ADR, R2, - COMPARE NUMBER WITH PRESENT
0071 * LARGEST NUMBER OF LIST
0072 0520 9A6E BCFR, LT SRCH # BRANCH FOR NEXT NUMBER
0073 0522 1B64 BCTR, UN MAXN BRANCH IF NEW NUMBER IS LARGER
0074 *
0075 0500 END SORT

```

TOTAL ASSEMBLY ERRORS = 0000

**Figure 12**



## Program Title

## LINEAR SORT SUBROUTINE

### Function

This program sorts multiple-byte numbers (signed or unsigned) into their incrementing order. In this example, the list contains 64 four-byte numbers. The list has a fixed starting address and a fixed length. The maximum list length is 256 bytes.

### Parameters

#### Input:

Unsorted list.

The SIGN flag indicates if the numbers are signed or unsigned.

SIGN = 0 means signed numbers.

SIGN = 1 means unsigned numbers.

#### Output:

Sorted list.

Refer to Figures 13 and 14 for flow-chart and program listing.

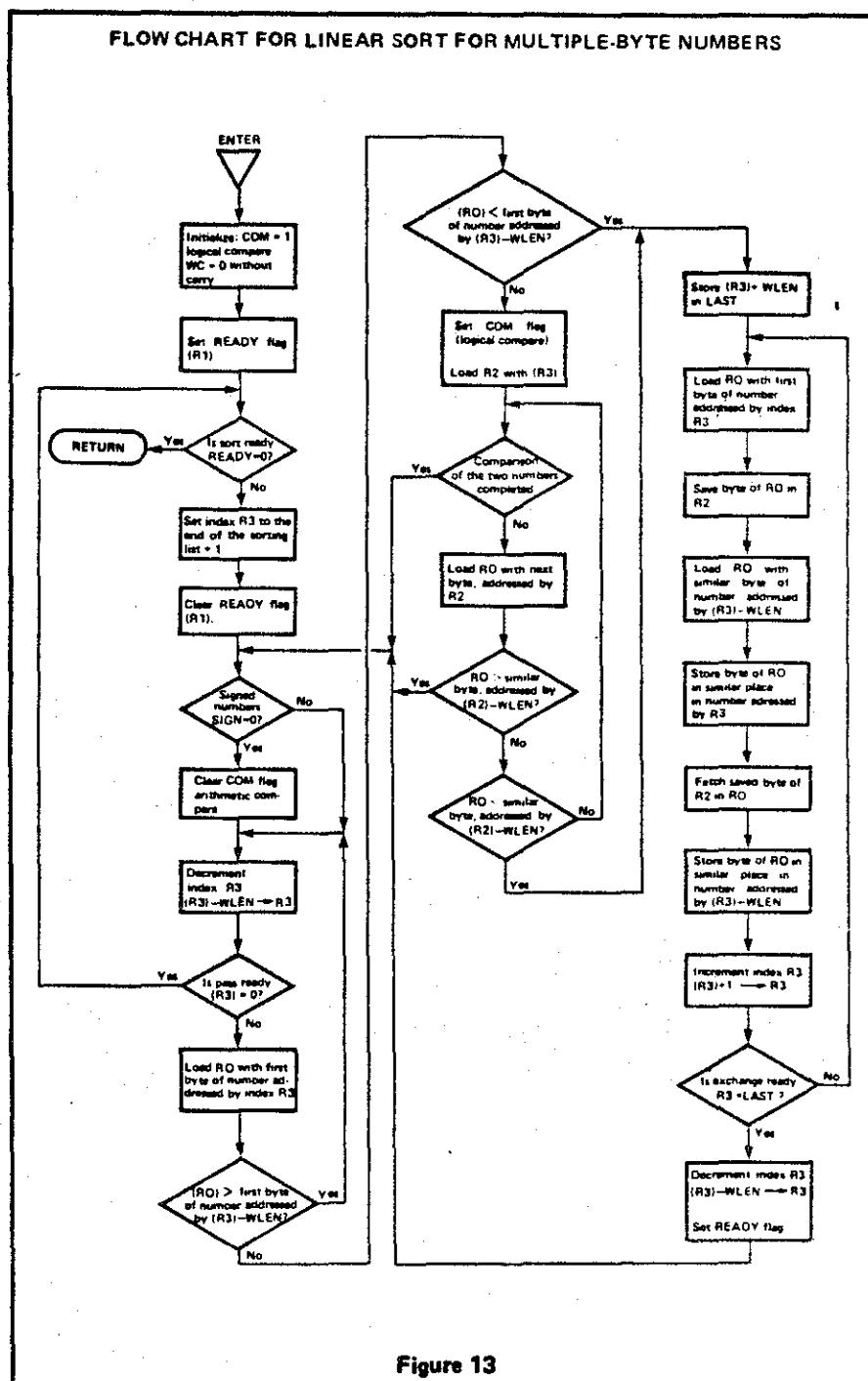


Figure 13

| HARDWARE AFFECTED |    |     |    |    |     |     |     | RAM REQUIRED (BYTES):              |  |
|-------------------|----|-----|----|----|-----|-----|-----|------------------------------------|--|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' | ROM REQUIRED (BYTES):              |  |
| PSU               | F  | II  | SP |    |     |     |     | EXECUTION TIME:                    |  |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   | MAXIMUM SUBROUTINE NESTING LEVELS: |  |
|                   | X  | X   |    | X  | X   | X   | X   | ASSEMBLER/COMPILER USED:           |  |
|                   |    |     |    |    |     |     |     | TWIN VER 1.0                       |  |

## Program Title

## LINEAR SORT SUBROUTINE

### Function

This program sorts multiple-byte numbers (signed or unsigned) into their incrementing order. In this example, the list contains 64 four-byte numbers. The list has a fixed starting address and a fixed length. The maximum list length is 256 bytes.

### Parameters

#### Input:

Unsorted list.

The SIGN flag indicates if the numbers are signed or unsigned.

SIGN = 0 means signed numbers.

SIGN = 1 means unsigned numbers.

#### Output:

Sorted list.

Refer to Figures 13 and 14 for flow-chart and program listing.

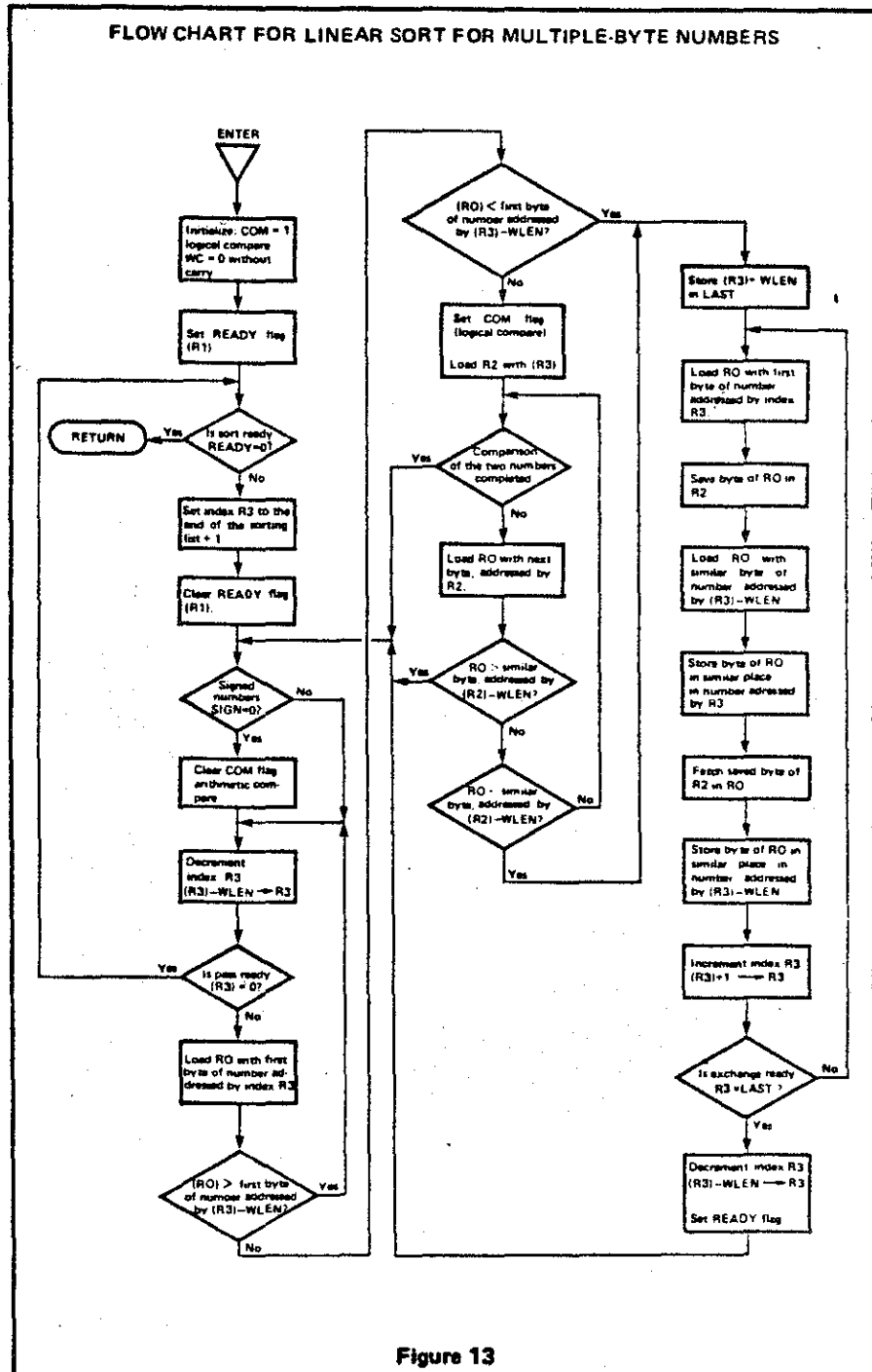


Figure 13

| HARDWARE AFFECTED |    |     |    |    |     |     |     | RAM REQUIRED (BYTES):              |  | 2            |
|-------------------|----|-----|----|----|-----|-----|-----|------------------------------------|--|--------------|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' | ROM REQUIRED (BYTES):              |  | 89           |
| PSU               | F  | II  | SP |    |     |     |     | EXECUTION TIME:                    |  | VARIABLE     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   | MAXIMUM SUBROUTINE NESTING LEVELS: |  | NONE         |
|                   | X  | X   |    | X  | X   | X   | X   | ASSEMBLER/COMPILER USED:           |  | TWIN VER 1.0 |

## PROGRAM LISTING FOR LINEAR SORT FOR MULTIPLE-BYTE NUMBERS

TWIN ASSEMBLER VER 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```

0021 *
0022 * *****
0023 * P0700004 *
0024 * *****
0025 * LINEAR SORTING SUBROUTINE *
0026 * *****
0027 * THIS PROGRAM SORTS A LIST OF MULTIPLE-BYTE NUMBERS
0028 * INTO INCREMENTING ORDER. THE MAXIMUM
0029 * NUMBER OF BYTES OF THE LIST TO BE SORTED IS
0030 * 256. THE START ADDRESS OF THE LIST IS '600'
0031 * THE NUMBER OF BYTES IN EACH NUMBER IS VARIABLE,
0032 * BUT IT MUST BE A POWER OF TWO IN THIS CASE.
0033 * THE LIST CONSISTS OF 64 NUMBERS OF 4 BYTES EACH.
0034 * UPON ENTRY TO THIS ROUTINE, THE SIGN FLAG INDICATES
0035 * IF THE NUMBERS TO BE SORTED ARE
0036 * SIGNED OR UNSIGNED. SIGN=0 MEANS SIGNED NUMBERS.
0037 * SIGN=NOT 0 MEANS UNSIGNED NUMBERS.
0038 * THE ORDER OF SORTING CAN BE CHANGED FROM AN
0039 * INCREMENTING ORDER TO A DECREMENTING ORDER BY
0040 * CHANGING THE INSTRUCTIONS MARKED WITH
0041 * A #. THE GREATER THAN (GT) TESTS MUST BE
0042 * CHANGED TO LESS THAN (LT) TESTS AND VICE VERSA.
0043 *
0044 ORG 'H'4FE'
0045 LAST RES 1 MEMORY LOCATION WHICH SAVES INDEX
0046 * OF NUMBER WHICH FOLLOWS NUMBER
0047 * ADDRESSED BY R3
0048 *
0049 SIGN RES 1 SIGN FLAG: SIGN=0 SIGNED NUMBER
0050 * SIGN= NOT 0 UNSIGNED NUMBER
0051 *
0052 ORG 'H'600'
0053 LEN EQU 'H'100' LENGTH OF SORTING LIST
0054 LIST RES LEN SORTING LIST
0055 WLEN EQU 4 WORD LENGTH IN BYTES
0056 *
0057 ORG 'H'500'
0058 SORT PPSL COM LOGICAL COMPARE
0059 * WITHOUT CARRY
0060 CPSL NC
0061 LODI,R1 'H'FF' SET READY FLAG
0062 PASS LODI R1 TEST AND RETURN IF READY FLAG
0063 * IS NOT SET
0064 RETC,Z
0065 LODI,R2 'LEN' LOAD INDEX R3
0066 LODI,R1 00 CLEAR READY FLAG
0067 COMP LODI,R0 SIGN TEST SIGN
0068 * SIGN= NOT 0, UNSIGNED NUMBER
0069 * SIGN=0, SIGNED NUMBER, CLEAR COM
0070 CPSL COM
0071 COMI SUBI,R2 WLEN DECREMENT INDEX R3
0072 * TEST AND BRANCH IF PASS READY
0073 BCTR,Z PASS
0074 LODI,R0 LIST,R3 LOAD R0 WITH FIRST BYTE OF
0075 * NUMBER ADDRESSED BY R3
0076 *
0077 COMI R0 LIST+WLEN,R2 COMPARE WITH FIRST BYTE OF
0078 * NEXT NUMBER, ADDRESSED BY
0079 * R2+WLEN
0080 *
0081 BCTR,GT COMI # IF GT, CONTINUE COMPARING

```

TWIN ASSEMBLER VER 1.0

PAGE 0003

LINE ADDR OBJECT E SOURCE

```

0086 051F 1A16 BCTR,LT EXCH # IF LT, EXCHANGE NUMBERS
0087 0521 7702 PPSL COM LOGICAL COMPARE
0088 0522 03 LODI R3 STORE INDEX R3 IN R2
0089 0524 C2 STRZ R2
0090 0525 0404 NEXT ADDI,R0 WLEN TEST AND BRANCH IF COMPARE
0091 0527 E2 COMI R2 OF BOTH NUMBERS IS READY
0092 0528 1902 BCTR,EQ COMP
0093 0529 0E2600 LODI,R0 LIST,R2 + LOAD R0 WITH NEXT BYTE OF
0094 * NUMBER, ADDRESSED BY R3
0095 052D EE65FC COMI R0 LIST+WLEN,R2 COMPARE WITH SIMILAR BYTE
0096 * OF NEXT NUMBER, ADDRESSED BY
0097 * R3+WLEN
0098 0530 190A BCTR,GT COMP # IF GT, CONTINUE COMPARING
0099 * NEXT TWO NUMBERS
0100 0532 1A02 BCTR,LT EXCH # IF LT, EXCHANGE NUMBERS
0101 0534 03 LODI R3
0102 0535 1A0E BCTR,UN NEXT CONTINUE COMPARING NEXT
0103 * SIMILAR BYTES OF NUMBERS
0104 0537 03 EXCH LODI R3 STORE INDEX OF NEXT NUMBER
0105 0538 0404 ADDI,R0 WLEN IN LAST
0106 053A C004FE STRA,R0 LAST
0107 053D 0F6600 EXCI LODI,R0 LIST,R3 EXCHANGE SIMILAR BYTE OF
0108 0540 C2 STRZ R2 BOTH NUMBERS
0109 0541 0F65FC LODI,R0 LIST+WLEN,R3
0110 0544 CF6600 STRA,R0 LIST,R3
0111 0547 02 LODI R2
0112 0548 CF65FC STRA,R0 LIST+WLEN,R3
0113 054B D000 BIRR,R3 #+2 INCREMENT INDEX R3
0114 054D EF04FE COMI R3 LAST TEST AND BRANCH IF
0115 0550 900B BCTR,EQ EXCI EXCHANGE IS NOT READY
0116 0552 A704 SUBI,R3 WLEN RESET INDEX R3
0117 0554 05FF LODI,R1 'H'FF' SET READY FLAG
0118 0556 1F050C BCTR,UN COMP CONTINUE COMPARING NEXT
0119 * TWO NUMBERS
0120 *
0121 0500 END SORT

```

TOTAL ASSEMBLY ERRORS = 0000

Figure 14

## Program Title

### SEARCH SORT SUBROUTINE FOR A FIXED LIST

## Function

This program sorts multiple-byte numbers (signed or unsigned) into their incrementing order. In this example, the list contains 64 four-byte numbers. The list has a fixed starting address and a fixed length. The maximum list length is 256 bytes.

## Parameters

### Input:

Unsorted list.

The SIGN flag indicates if the numbers are signed or unsigned.

SIGN = 0 means signed numbers.

SIGN = 1 means unsigned numbers.

### Output:

Sorted list.

Refer to Figures 15 and 16 for flowchart and program listing.

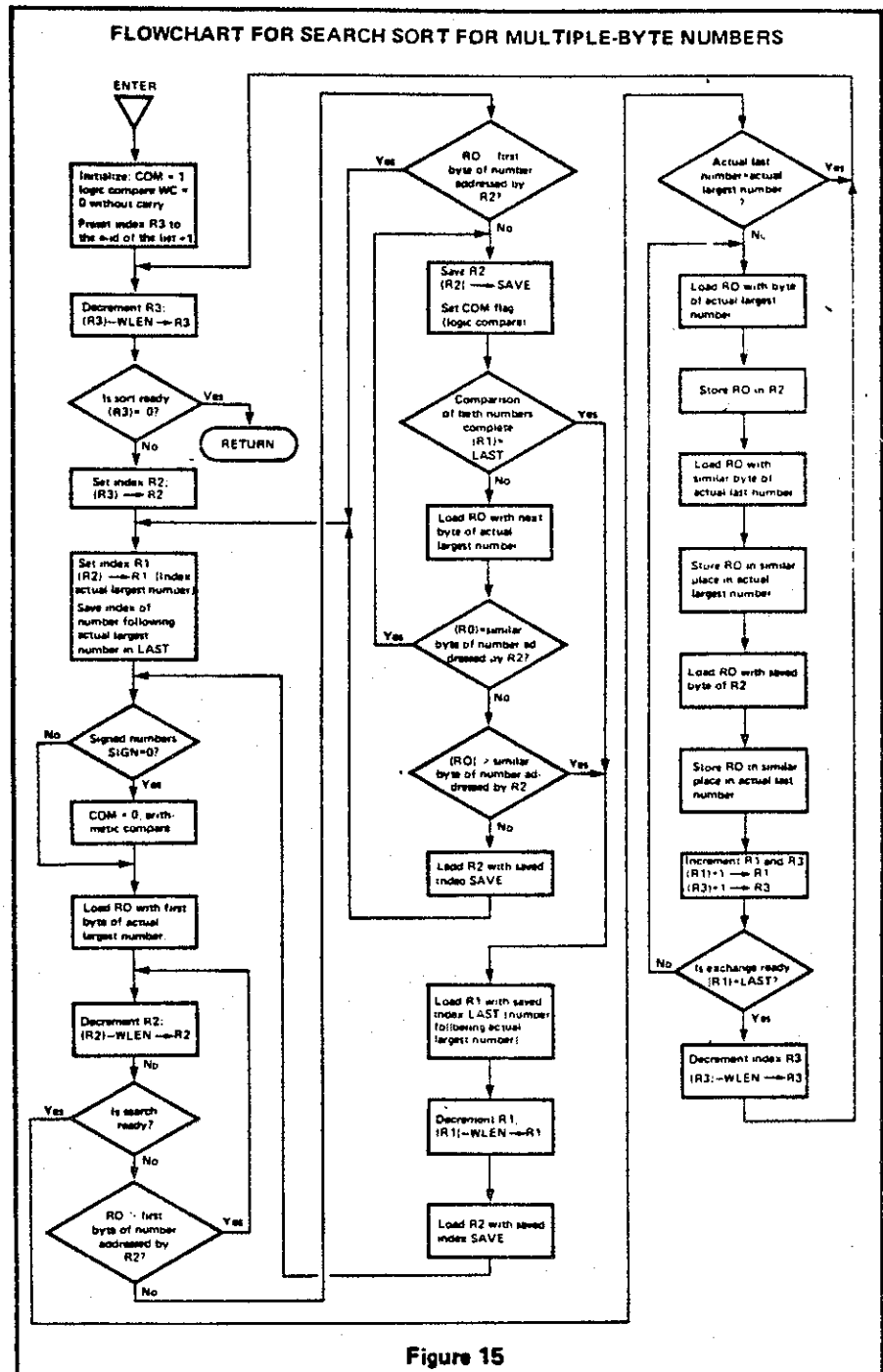


Figure 15

| HARDWARE AFFECTED |    |     |    |    |     |     |     | RAM REQUIRED (BYTES):              |  |
|-------------------|----|-----|----|----|-----|-----|-----|------------------------------------|--|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' | ROM REQUIRED (BYTES):              |  |
| PSU               | F  | II  | SP |    |     |     |     | EXECUTION TIME:                    |  |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   | MAXIMUM SUBROUTINE NESTING LEVELS: |  |
|                   | X  | X   |    | X  | X   | X   | X   | ASSEMBLER/COMPILER USED:           |  |
|                   |    |     |    |    |     |     |     | TWIN VER 1.0                       |  |

## PROGRAM LISTING FOR SEARCH SORT FOR MULTIPLE-BYTE NUMBERS

TWIN ASSEMBLER VEP 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```

0001 *
0002 *-----*
0003 * P0760025 *
0004 *-----*
0005 *SEARCH SORTING SUBROUTINE *
0006 *-----*
0007 * THIS PROGRAM SORTS A LIST OF MULTIPLE-BYTES NUMBERS
0008 * INTO THEIR INCREMENTING ORDER. THE MAXIMUM NUMBER
0009 * OF BYTES IN THE LIST TO BE SORTED IS 256
0010 * THE START ADDRESS OF THE SORTING LIST IS 600
0011 * THE NUMBER OF BYTES IN EACH NUMBER IS VARIABLE.
0012 * BUT IT MUST BE A POWER OF TWO IN THIS CASE THE
0013 * LIST CONSISTS OF 64 NUMBERS OF 4 BYTES EACH.
0014 * UPON ENTRY TO THIS SUBROUTINE, THE SIGN FLAG
0015 * INDICATES IF THE NUMBERS TO BE SORTED
0016 * ARE SIGNED OR UNSIGNED
0017 * SIGN = 0 MEANS UNSIGNED NUMBERS.
0018 * SIGN = 8 MEANS SIGNED NUMBERS.
0019 * THE ORDER OF SORTING CAN BE CHANGED FROM AN
0020 * INCREMENTING TO A DECREMENTING ORDER BY CHANGING
0021 * THE INSTRUCTIONS MARKED WITH A #
0022 * THE GREATER THAN (GT) TESTS MUST BE
0023 * CHANGED TO LESS THAN (LT) TESTS AND VICE VERSA
0024 *
0025 *-----*
0026 *
0027 *
0028 *
0029 *
0030 *
0031 *
0032 *
0033 *
0034 *
0035 *
0036 *
0037 *
0038 *
0039 *
0040 *
0041 *
0042 *
0043 *
0044 *
0045 *
0046 *
0047 *
0048 *
0049 *
0050 *
0051 *
0052 *
0053 *
0054 *
0055 *
0056 *
0057 *
0058 *
0059 *
0060 *
0061 *
0062 *
0063 *
0064 *
0065 *
0066 *
0067 *
0068 *
0069 *
0070 *
0071 *
0072 *
0073 *
0074 *
0075 *
0076 *
0077 *
0078 *
0079 *
0080 *
0081 *
0082 *
0083 *
0084 *
0085 *
0086 *
0087 *
0088 *
0089 *
0090 *
0091 *
0092 *
0093 *
0094 *
0095 *
0096 *
0097 *
0098 *
0099 *
0100 *
0101 *
0102 *
0103 *
0104 *
0105 *
0106 *
0107 *
0108 *
0109 *
0110 *
0111 *
0112 *
0113 *
0114 *
0115 *
0116 *
0117 *
0118 *
0119 *
0120 *
0121 *
0122 *
0123 *
0124 *
0125 *
0126 *
0127 *
0128 *
0129 *
0130 *
0131 *
0132 *
0133 *
0134 *
0135 *
0136 *
0137 *
0138 *
0139 *
0140 *
0141 *
0142 *
0143 *
0144 *
0145 *
0146 *
0147 *
0148 *
0149 *
0150 *
0151 *
0152 *
0153 *
0154 *
0155 *
0156 *
0157 *
0158 *
0159 *
0160 *
0161 *
0162 *
0163 *
0164 *
0165 *
0166 *
0167 *
0168 *
0169 *
0170 *
0171 *
0172 *
0173 *
0174 *
0175 *
0176 *
0177 *
0178 *
0179 *
0180 *
0181 *
0182 *
0183 *
0184 *
0185 *
0186 *
0187 *
0188 *
0189 *
0190 *
0191 *
0192 *
0193 *
0194 *
0195 *
0196 *
0197 *
0198 *
0199 *
0200 *
0201 *
0202 *
0203 *
0204 *
0205 *
0206 *
0207 *
0208 *
0209 *
0210 *
0211 *
0212 *
0213 *
0214 *
0215 *
0216 *
0217 *
0218 *
0219 *
0220 *
0221 *
0222 *
0223 *
0224 *
0225 *
0226 *
0227 *
0228 *
0229 *
0230 *
0231 *
0232 *
0233 *
0234 *
0235 *
0236 *
0237 *
0238 *
0239 *
0240 *
0241 *
0242 *
0243 *
0244 *
0245 *
0246 *
0247 *
0248 *
0249 *
0250 *
0251 *
0252 *
0253 *
0254 *
0255 *
0256 *
0257 *
0258 *
0259 *
0260 *
0261 *
0262 *
0263 *
0264 *
0265 *
0266 *
0267 *
0268 *
0269 *
0270 *
0271 *
0272 *
0273 *
0274 *
0275 *
0276 *
0277 *
0278 *
0279 *
0280 *
0281 *
0282 *
0283 *
0284 *
0285 *
0286 *
0287 *
0288 *
0289 *
0290 *
0291 *
0292 *
0293 *
0294 *
0295 *
0296 *
0297 *
0298 *
0299 *
0300 *
0301 *
0302 *
0303 *
0304 *
0305 *
0306 *
0307 *
0308 *
0309 *
0310 *
0311 *
0312 *
0313 *
0314 *
0315 *
0316 *
0317 *
0318 *
0319 *
0320 *
0321 *
0322 *
0323 *
0324 *
0325 *
0326 *
0327 *
0328 *
0329 *
0330 *
0331 *
0332 *
0333 *
0334 *
0335 *
0336 *
0337 *
0338 *
0339 *
0340 *
0341 *
0342 *
0343 *
0344 *
0345 *
0346 *
0347 *
0348 *
0349 *
0350 *
0351 *
0352 *
0353 *
0354 *
0355 *
0356 *
0357 *
0358 *
0359 *
0360 *
0361 *
0362 *
0363 *
0364 *
0365 *
0366 *
0367 *
0368 *
0369 *
0370 *
0371 *
0372 *
0373 *
0374 *
0375 *
0376 *
0377 *
0378 *
0379 *
0380 *
0381 *
0382 *
0383 *
0384 *
0385 *
0386 *
0387 *
0388 *
0389 *
0390 *
0391 *
0392 *
0393 *
0394 *
0395 *
0396 *
0397 *
0398 *
0399 *
0400 *
0401 *
0402 *
0403 *
0404 *
0405 *
0406 *
0407 *
0408 *
0409 *
0410 *
0411 *
0412 *
0413 *
0414 *
0415 *
0416 *
0417 *
0418 *
0419 *
0420 *
0421 *
0422 *
0423 *
0424 *
0425 *
0426 *
0427 *
0428 *
0429 *
0430 *
0431 *
0432 *
0433 *
0434 *
0435 *
0436 *
0437 *
0438 *
0439 *
0440 *
0441 *
0442 *
0443 *
0444 *
0445 *
0446 *
0447 *
0448 *
0449 *
0450 *
0451 *
0452 *
0453 *
0454 *
0455 *
0456 *
0457 *
0458 *
0459 *
0460 *
0461 *
0462 *
0463 *
0464 *
0465 *
0466 *
0467 *
0468 *
0469 *
0470 *
0471 *
0472 *
0473 *
0474 *
0475 *
0476 *
0477 *
0478 *
0479 *
0480 *
0481 *
0482 *
0483 *
0484 *
0485 *
0486 *
0487 *
0488 *
0489 *
0490 *
0491 *
0492 *
0493 *
0494 *
0495 *
0496 *
0497 *
0498 *
0499 *
0500 *
0501 *
0502 *
0503 *
0504 *
0505 *
0506 *
0507 *
0508 *
0509 *
0510 *
0511 *
0512 *
0513 *
0514 *
0515 *
0516 *
0517 *
0518 *
0519 *
0520 *
0521 *
0522 *
0523 *
0524 *
0525 *
0526 *
0527 *
0528 *
0529 *
0530 *
0531 *
0532 *
0533 *
0534 *
0535 *
0536 *
0537 *
0538 *
0539 *
0540 *
0541 *
0542 *
0543 *
0544 *
0545 *
0546 *
0547 *
0548 *
0549 *
0550 *
0551 *
0552 *
0553 *
0554 *
0555 *
0556 *
0557 *
0558 *
0559 *
0560 *
0561 *
0562 *
0563 *
0564 *
0565 *
0566 *
0567 *
0568 *
0569 *
0570 *
0571 *
0572 *
0573 *
0574 *
0575 *
0576 *
0577 *
0578 *
0579 *
0580 *
0581 *
0582 *
0583 *
0584 *
0585 *
0586 *
0587 *
0588 *
0589 *
0590 *
0591 *
0592 *
0593 *
0594 *
0595 *
0596 *
0597 *
0598 *
0599 *
0600 *
0601 *
0602 *
0603 *
0604 *
0605 *
0606 *
0607 *
0608 *
0609 *
0610 *
0611 *
0612 *
0613 *
0614 *
0615 *
0616 *
0617 *
0618 *
0619 *
0620 *
0621 *
0622 *
0623 *
0624 *
0625 *
0626 *
0627 *
0628 *
0629 *
0630 *
0631 *
0632 *
0633 *
0634 *
0635 *
0636 *
0637 *
0638 *
0639 *
0640 *
0641 *
0642 *
0643 *
0644 *
0645 *
0646 *
0647 *
0648 *
0649 *
0650 *
0651 *
0652 *
0653 *
0654 *
0655 *
0656 *
0657 *
0658 *
0659 *
0660 *
0661 *
0662 *
0663 *
0664 *
0665 *
0666 *
0667 *
0668 *
0669 *
0670 *
0671 *
0672 *
0673 *
0674 *
0675 *
0676 *
0677 *
0678 *
0679 *
0680 *
0681 *
0682 *
0683 *
0684 *
0685 *
0686 *
0687 *
0688 *
0689 *
0690 *
0691 *
0692 *
0693 *
0694 *
0695 *
0696 *
0697 *
0698 *
0699 *
0700 *
0701 *
0702 *
0703 *
0704 *
0705 *
0706 *
0707 *
0708 *
0709 *
0710 *
0711 *
0712 *
0713 *
0714 *
0715 *
0716 *
0717 *
0718 *
0719 *
0720 *
0721 *
0722 *
0723 *
0724 *
0725 *
0726 *
0727 *
0728 *
0729 *
0730 *
0731 *
0732 *
0733 *
0734 *
0735 *
0736 *
0737 *
0738 *
0739 *
0740 *
0741 *
0742 *
0743 *
0744 *
0745 *
0746 *
0747 *
0748 *
0749 *
0750 *
0751 *
0752 *
0753 *
0754 *
0755 *
0756 *
0757 *
0758 *
0759 *
0760 *
0761 *
0762 *
0763 *
0764 *
0765 *
0766 *
0767 *
0768 *
0769 *
0770 *
0771 *
0772 *
0773 *
0774 *
0775 *
0776 *
0777 *
0778 *
0779 *
0780 *
0781 *
0782 *
0783 *
0784 *
0785 *
0786 *
0787 *
0788 *
0789 *
0790 *
0791 *
0792 *
0793 *
0794 *
0795 *
0796 *
0797 *
0798 *
0799 *
0800 *
0801 *
0802 *
0803 *
0804 *
0805 *
0806 *
0807 *
0808 *
0809 *
0810 *
0811 *
0812 *
0813 *
0814 *
0815 *
0816 *
0817 *
0818 *
0819 *
0820 *
0821 *
0822 *
0823 *
0824 *
0825 *
0826 *
0827 *
0828 *
0829 *
0830 *
0831 *
0832 *
0833 *
0834 *
0835 *
0836 *
0837 *
0838 *
0839 *
0840 *
0841 *
0842 *
0843 *
0844 *
0845 *
0846 *
0847 *
0848 *
0849 *
0850 *
0851 *
0852 *
0853 *
0854 *
0855 *
0856 *
0857 *
0858 *
0859 *
0860 *
0861 *
0862 *
0863 *
0864 *
0865 *
0866 *
0867 *
0868 *
0869 *
0870 *
0871 *
0872 *
0873 *
0874 *
0875 *
0876 *
0877 *
0878 *
0879 *
0880 *
0881 *
0882 *
0883 *
0884 *
0885 *
0886 *
0887 *
0888 *
0889 *
0890 *
0891 *
0892 *
0893 *
0894 *
0895 *
0896 *
0897 *
0898 *
0899 *
0900 *
0901 *
0902 *
0903 *
0904 *
0905 *
0906 *
0907 *
0908 *
0909 *
0910 *
0911 *
0912 *
0913 *
0914 *
0915 *
0916 *
0917 *
0918 *
0919 *
0920 *
0921 *
0922 *
0923 *
0924 *
0925 *
0926 *
0927 *
0928 *
0929 *
0930 *
0931 *
0932 *
0933 *
0934 *
0935 *
0936 *
0937 *
0938 *
0939 *
0940 *
0941 *
0942 *
0943 *
0944 *
0945 *
0946 *
0947 *
0948 *
0949 *
0950 *
0951 *
0952 *
0953 *
0954 *
0955 *
0956 *
0957 *
0958 *
0959 *
0960 *
0961 *
0962 *
0963 *
0964 *
0965 *
0966 *
0967 *
0968 *
0969 *
0970 *
0971 *
0972 *
0973 *
0974 *
0975 *
0976 *
0977 *
0978 *
0979 *
0980 *
0981 *
0982 *
0983 *
0984 *
0985 *
0986 *
0987 *
0988 *
0989 *
0990 *
0991 *
0992 *
0993 *
0994 *
0995 *
0996 *
0997 *
0998 *
0999 *
1000 *

```

TWIN ASSEMBLER VEP 1.0

PAGE 0003

LINE ADDR OBJECT E SOURCE

```

0002 0519 006000 LORI LORI R0 LIST,R1 FETCH FIRST BYTE OF ACTUAL
0003 * * * * * * * *
0004 051C 0004 COMP SUB1,R2 ALIN DECREMENT INDEX R2
0005 051E 00FC COM1,R2 ALIST-ALIN TEST AND BRANCH IF SEARCH
0006 0520 102A BCTR,ED EXCH IS READY
0007 0522 006000 COM1,R0 LIST,R2 COMPARE R0 WITH FIRST BYTE
0008 * * * * * * * *
0009 0525 1075 BCTR,GT COMP # IF GT, THEN NUMBER ADDRESSED BY R2
0010 0527 * * * * * * * *
0011 0529 1062 BCTR,LT ALIN # IF LT, THEN NEW ACTUAL LARG-
0012 * * * * * * * *
0013 052B 0000 * * * * * * * *
0014 052D 0004F1 STRA,R2 SAVE SAVE INDEX R2
0015 052F 7702 PPSL COM LOGICAL COMPARE
0016 0531 0004F2 COM1,R1 LAST TEST AND BRANCH IF COMPARE OF
0017 0533 100F BCTR,ED PSET FOLLOWING BYTES IS READY
0018 0535 002600 LORI,R0 LIST,R1 COMPARE FOLLOWING BYTES
0019 0537 002600 COM1,R0 LIST,R2 OF BOTH NUMBERS
0020 0539 1072 BCTR,ED NEXT BYTES EQUAL, THEN CONTINUE
0021 053B 1005 BCTR,GT PSET # IF GT NUMBER ADDRESSED BY R1
0022 * * * * * * * *
0023 053D 0000 * * * * * * * *
0024 053F 0000 * * * * * * * *
0025 0541 0004F1 LORI,R2 SAVE FETCH SAVED INDEX
0026 0543 1019 BCTR,UN ALIN NEW ACTUAL LARGEST NUMBER, BRANCH
0027 0545 0004F2 PSET LORI,R1 LAST RESET INDEX LARGEST NUMBER
0028 0547 0004F1 SUB1,R1 ALIN
0029 0549 1046 LORI,R2 SAVE FETCH SAVED INDEX R2
0030 054B 0000 BCTR,UN LORI
0031 054D 0000 EXCH LORI,R2 TEST IF LARGEST NUMBER IS THE
0032 054F 0000 COM2,R1 SAME AS THE LAST NAME OF THE
0033 0551 0017 BCTR,ED BPOH ACTUAL LIST
0034 0553 0025FF EXCH LORI,R1 LIST-1,R1 EXCHANGE THE LAST NUMBER
0035 0555 0000 STR2,R2 OF THE ACTUAL LIST AND THE
0036 0557 006000 LORI,R0 LIST,R2 ACTUAL LARGEST NUMBER OF
0037 0559 0025FF STRA,R0 LIST-1,R1 THE LIST
0038 055B 0000 LORI,R2
0039 055D 006000 STRA,R0 LIST,R2
0040 055F 0000 BPOH,R2 #+2
0041 0561 0004F2 COM1,R1 LAST TEST AND BRANCH IF EXCHANGE
0042 0563 0000 BCTR,ED EXCH IS NOT READY
0043 0565 0004F1 SUB1,R1 ALIN RESET INDEX R2
0044 0567 102506 BPOH BCTR,UN PPSL NEXT PASS
0045 0569 *
0046 056B *
0047 056D *
0048 056F *
0049 0571 *
0050 0573 *
0051 0575 *
0052 0577 *
0053 0579 *
0054 057B *
0055 057D *
0056 057F *
0057 0581 *
0058 0583 *
0059 0585 *
0060 0587 *
0061 0589 *
0062 058B *
0063 058D *
0064 058F *
0065 0591 *
0066 0593 *
0067 0595 *
0068 0597 *
0069 0599 *
0070 059B *
0071 059D *
0072 059F *
0073 05A1 *
0074 05A3 *
0075 05A5 *
0076 05A7 *
0077 05A9 *
0078 05AB *
0079 05AD *
0080 05AF *
0081 05B1 *
0082 05B3 *
0083 05B5 *
0084 05B7 *
0085 05B9 *
0086 05BB *
0087 05BD *
0088 05BF *
0089 05C1 *
0090 05C3 *
0091 05C5 *
0092 05C7 *
0093 05C9 *
0094 05CB *
0095 05CD *
0096 05CF *
0097 05D1 *
0098 05D3 *
0099 05D5 *
0100 05D7 *
0101 05D9 *
0102 05DB *
0103 05DD *
0104 05DF *
0105 05E1 *
0106 05E3 *
0107 05E5 *
0108 05E7 *
0109 05E9 *
0110 05EB *
0111 05ED *
0112 05EF *
0113 05F1 *
0114 05F3 *
0115 05F5 *
0116 05F7 *
0117 05F9 *
0118 05FB *
0119 05FD *
0120 05FF *
0121 0601 *
0122 0603 *
0123 0605 *
0124 0607 *
0125 0609 *
0126 060B *
0127 060D *
0128 060F *
0129 0611 *
0130 0613 *
0131 0615 *
0132 0617 *
0133 0619 *
0134 061B *
0135 061D *
0136 061F *
0137 0621 *
0138 0623 *
0139 0625 *
0140 0627 *
0141 0629 *
0142 062B *
0143 062D *
0144 062F *
0145 0631 *
0146 0633 *
0147 0635 *
0148 0637 *
0149 0639 *
0150 063B *
0151 063D *
0152 063F *
0153 0641 *
0154 0643 *
0155 0645 *
0156 0647 *
0157 0649 *
0158 064B *
0159 064D *
0160 064F *
0161 0651 *
0162 0653 *
0163 0655 *
0164 0657 *
0165 0659 *
0166 065B *
0167 065D *
0168 065F *
0169 0661 *
0170 0663 *
0171 0665 *
0172 0667 *
0173 0669 *
0174 066B *
0175 066D *
0176 066F *
0177 0671 *
0178 0673 *
0179 0675 *
0180 0677 *
0181 0679 *
0182 067B *
0183 067D *
0184 067F *
0185 0681 *
0186 0683 *
0187 0685 *
0188 0687 *
0189 0689 *
0190 068B *
0191 068D *
0192 068F *
0193 0691 *
0194 0693 *
0195 0695 *
0196 0697 *
0197 0699 *
0198 069B *
0199 069D *
0200 069F *
0201 06A1 *
0202 06A3 *
0203 06A5 *
0204 06A7 *
0205 06A9 *
0206 06AB *
0207 06AD *
0208 06AF *
0209 06B1 *
0210 06B3 *
0211 06B5 *
0212 06B7 *
0213 06B9 *
0214 06BB *
0215 06BD *
0216 06BF *
0217 06C1 *
0218 06C3 *
0219 06C5 *
0220 06C7 *
0221 06C9 *
0222 06CB *
0223 06CD *
0224 06CF *
0225 06D1 *
0226 06D3 *
0227 06D5 *
0228 06D7 *
0229 06D9 *
0230 06DB *
0231 06DD *
0232 06DF *
0233 06E1 *
0234 06E3 *
0235 06E5 *
0236 06E7 *
0237 06E9 *
0238 06EB *
0239 06ED *
0240 06EF *
0241 06F1 *
0242 06F3 *
0243 06F5 *
0244 06F7 *
0245 06F9 *
0246 06FB *
0247 06FD *
0248 06FF *
0249 0701 *
0250 0703 *
0251 0705 *
0252 0707 *
0253 0709 *
0254 070B *
0255 070D *
0256 070F *
0257 0711 *
0258 0713 *
0259 0715 *
0260 0717 *
0261 0719 *
0262 071B *
0263 071D *
0264 071F *
0265 0721 *
0266 0723 *
0267 0725 *
0268 0727 *
0269 0729 *
0270 072B *
0271 072D *
0272 072F *
0273 0731 *
0274 0733 *
0275 0735 *
0276 0737 *
0277 0739 *
0278 073B *
0279 073D *
0280 073F *
0281 0741 *
0282 0743 *
0283 0745 *
0284 0747 *
0285 0749 *
0286 074B *
0287 074D *
0288 074F *
0289 0751 *
0290 0753 *
0291 0755 *
0292 0757 *
0293 0759 *
0294 075B *
0295 075D *
0296 075F *
0297 0761 *
0298 0763 *
0299 0765 *
0300 0767 *
0301 0769 *
0302 076B *
0303 076D *
0304 076F *
0305 0771 *
0306 0773 *
0307 0775 *
0308 0777 *
0309 0779 *
0310 077B *
0311 077D *
0312 077F *
0313 0781 *
0314 0783 *
0315 0785 *
0316 0787 *
0317 0789 *
0318 078B *
0319 078D *
0320 078F *
0321 0791 *
0322 0793 *
0323 0795 *
0324 0797 *
0325 0799 *
0326 079B *
0327 079D *
0328 079F *
0329 07A1 *
0330 07A3 *
0331 07A
```

## Program Title

## SEARCH SORT SUBROUTINE FOR A FIXED LIST

## Function

This program sorts multiple-byte numbers (signed or unsigned) into their incrementing order. The list to be sorted may contain more than 256 bytes. In this case, the list contains 256 eight-byte numbers. The list has a fixed starting address and length.

## Parameters

### Input:

Unsorted list.

The SIGN flag indicates if the numbers are signed or unsigned.

SIGN = 0 means signed numbers.

SIGN = 1 means unsigned numbers.

### Output:

Sorted list.

Refer to Figures 17 and 18 for flow-chart and program listing.

## FLOW CHART FOR SEARCH SORT SUBROUTINE (256 EIGHT-BYTE NUMBERS)

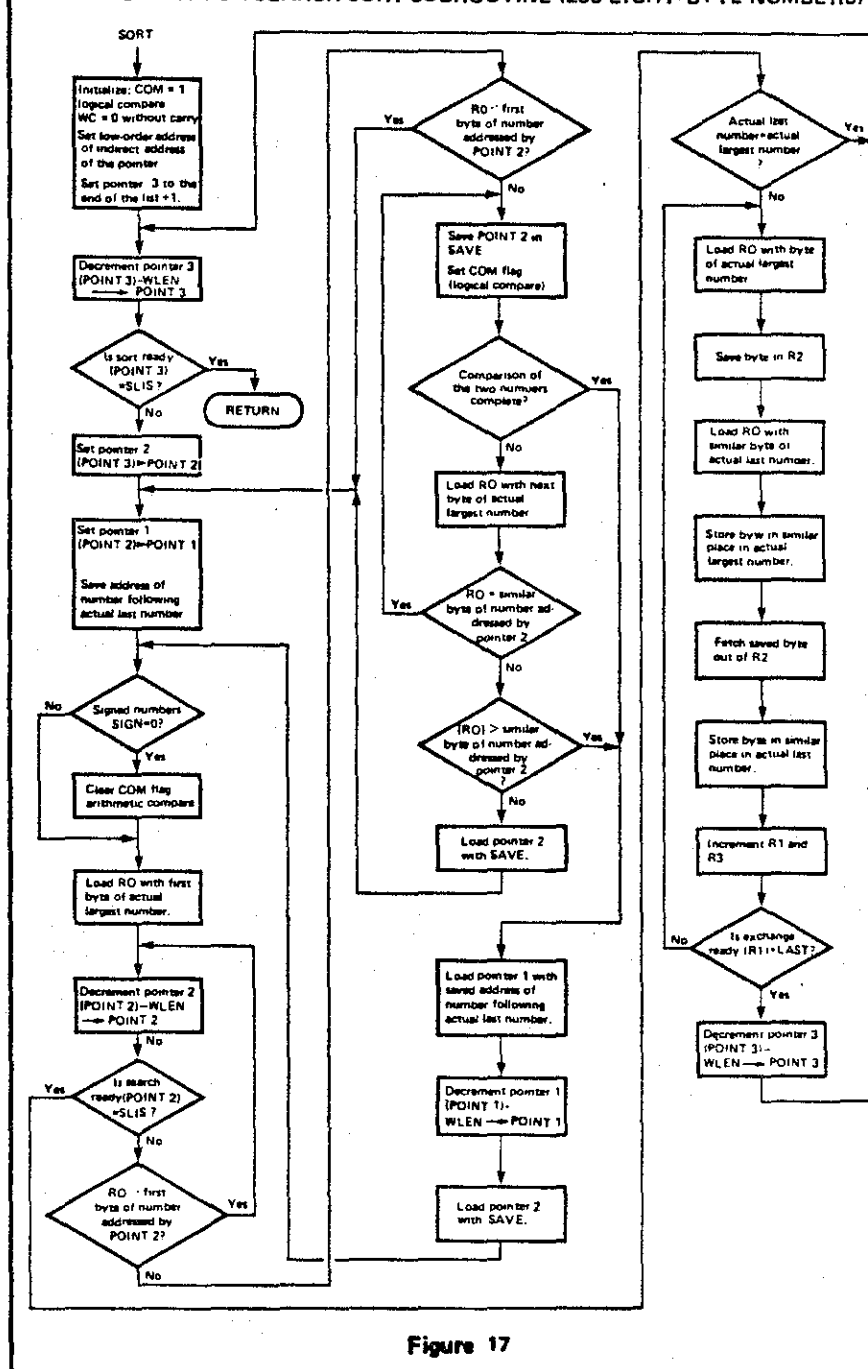


Figure 17

| HARDWARE AFFECTED |    |     |    |    |     |     |     | RAM REQUIRED (BYTES):    |  |
|-------------------|----|-----|----|----|-----|-----|-----|--------------------------|--|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' | ROM REQUIRED (BYTES):    |  |
| PSU               | F  | H   | SP |    |     |     |     | EXECUTION TIME:          |  |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   | MAXIMUM SUBROUTINE       |  |
|                   | X  | X   |    | X  | X   | X   | X   | NESTING LEVELS:          |  |
|                   |    |     |    |    |     |     |     | ASSEMBLER/COMPILER USED: |  |

9

167

VARIABLE

NONE

TWIN VER 1.0

## PROGRAM LISTING FOR SEARCH SORT SUBROUTINE (256 EIGHT-BYTE NUMBERS)

TWIN ASSEMBLER, VEP 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```

0031 *****
0032 * PD760066 *
0033 *****
0034 * SEARCH SORTING SUBROUTINE *
0035 *****
0036 * THIS PROGRAM SORTS A LIST OF MULTIPLE-BYTE NUMBERS
0037 * INTO THEIR INCREMENTING ORDER. THE
0038 * START ADDRESS OF THE LIST IS H'500' THE NUMBER OF
0039 * BYTES IN EACH NUMBER IS VARIABLE, BUT IT MUST BE A
0040 * POWER OF TWO. IN THIS CASE THE LIST CONSISTS OF
0041 * 256 EIGHT-BYTE NUMBERS. UPON ENTRY TO THIS
0042 * SUBROUTINE, THE SIGN FLAG INDICATES IF THE NUMBERS
0043 * ARE SIGNED OR UNSIGNED:
0044 * SIGN = NOT 0 MEANS UNSIGNED NUMBERS.
0045 * SIGN = 0 MEANS SIGNED NUMBERS.
0046 * THE ORDER OF SORTING CAN BE CHANGED FROM AN INCRE-
0047 * MENTING TO A DECREMENTING ORDER BY CHANGING
0048 * THE INSTRUCTIONS MARKED WITH A #.
0049 * THE GREATER THAN (GT) TESTS MUST BE
0050 * CHANGED TO LESS THAN (LT) TESTS AND VICE VERSA
0051 *
0052 0000 ORG H'4F7'
0053 04F7 SIGN RES 1 SIGN FLAG: SIGN=0 SIGNED NUMBER
0054 * SIGN= NOT 0 UNSIGNED NUMBER
0055 04F8 SAVE RES 1 SAVED LOW-ORDER ADDRESS POINTER 2
0056 04F9 LAST RES 1 SAVED INDEX OF NUMBER FOLLOWING
0057 * ACTUAL LARGEST NUMBER
0058 04FA R01 RES 2 HIGH ORDER ADDRESS POINTER 1
0059 04FC R02 RES 2 HIGH ORDER ADDRESS POINTER 2
0060 04FE R03 RES 2 HIGH ORDER ADDRESS POINTER 3
0061 *
0062 0500 ORG H'500'
0063 0500 SLIS RES 1 START ADDRESS OF SORTING LIST
0064 0500 ELIS RES 1 END ADDRESS OF SORTING LIST
0065 0900 MLEN EQU 8 WORD LENGTH (BYTES)
0066 *
0067 0091 ORG H'440'
0068 0440 7702 SORT PPSL COM LOGICAL COMPARE
0069 0442 7509 CPSL NC WITHOUT CARRY
0070 0444 0400 LOOI, R0 >SLIS SET LOW-ORDER ADDRESS OF INDIRECT
0071 0446 CC04FB STRA, R0 R01+1 ADDRESS
0072 0449 CC04FD STRA, R0 R02+1
0073 044C CC04FF STRA, R0 R03+1
0074 044F 0700 LOOI, R3 <ELIS SET POINTER AT THE END OF THE
0075 0451 CF04FE STRA, R3 R03 SORTING LIST
0076 0454 0700 LOOI, R3 <ELIS
0077 0456 BC04FE PRES LOAR, R0 R03 TEST AND RETURN IF SORT IS
0078 0459 5000 BMR, R3 PPS1 READY
0079 045B E405 COMI, R0 <SLIS
0080 045D 14 RETC ER
0081 045E F000 EDOR, R0 #+2 DECREMENT POINTER 3 TO LAST NUMBER
0082 0460 CC04FE STRA, R0 R03 OF ACTUAL LIST
0083 0462 0700 PPS1 SUBI, R3 MLEN
0084 0465 CC04FC STRA, R0 R02 SET POINTER 2 AT LAST NUMBER OF
0085 0468 03 LOOZ P2 THE ACTUAL LIST
0086 0469 C2 STRZ R2
0087 046B CC04FC MARM LOAR, R0 R02 SET POINTER 1 AT THE ACTUAL
0088 046D CC04FA STRA, R0 R01 LARGEST NUMBER OF THE ACTUAL
0089 0470 02 LOOZ R2 LIST
0090 0471 C1 STRZ R1

```

Figure 18

**PROGRAM LISTING FOR SEARCH SORT SUBROUTINE (256 EIGHT-BYTE NUMBERS) (Cont.)**

TWIN ASSEMBLER VER 1.0

PAGE 0003

LINE ADDR OBJECT E SOURCE

```

0091 0472 0400 ADDI R0 #LEN SAVE INDEX OF NUMBER FOLLOWING
0092 0474 0004F9 STRA R0 LAST THE LAST NUMBER OF THE ACTUAL
0093 * LIST
0094 0477 0004F7 LOAD R0,R0 SIGN IF SIGN IS 0, SET COMPARE, SIGNED
0095 0479 9002 BCFR Z LAST NUMBER ELSE CLEAR COMPARE,
0096 047C 7502 CPSL COM UNSIGNED NUMBERS
0097 047E 00E4FA LOA1 R0,R0 #AD1,R1 LOAD R0 WITH FIRST BYTE OF
0098 * ACTUAL LARGEST NUMBER
0099 0481 5A0E COMP B0,R0,R2 COM1 TEST AND BRANCH TO EXCH IF
0100 0483 00E4FC LOA1 R2,R2 AD2 SEARCH IS READY.
0101 0486 E005 COM1 R2 <SLIS
0102 0488 1025 BCTR EQ EXCH
0103 048A FA00 B0RR R2 #+2 DECREMENT POINTER 2
0104 048C 00E4FC STRA R2 AD2
0105 048F 0000 LOO1 R2 0
0106 0491 A600 COM1 SUB1,R2 #LEN
0107 0493 EEE4FC COM1 R0 #AD2,R2 COMPARE ACTUAL LARGEST WORD
0108 * WITH WORD ADDRESSED BY POINT 2
0109 0496 1909 BCTR GT COMP #IF GT, THEN IT IS STILL
0110 * ACTUAL LARGEST NUMBER
0111 0498 1A50 BCTR LT FROM #IF LT, THEN NEW ACTUAL
0112 * LARGEST NUMBER IS FOUND
0113 * ELSE COMPARE NEXT BYTES OF
0114 * BOTH NUMBERS.
0115 049A 00E4FB STRA R2 SAVE SAVE POINTER 2
0116 049D 7702 PPSL COM LOGICAL COMPARE
0117 049F ED04F9 NEXT COM1,R1 LAST TEST AND BRANCH IF COMPARE OF
0118 04A2 1010 BCTR EQ RSET THE FOLLOWING BYTES OF THE
0119 * TWO NUMBERS IS READY.
0120 04A4 00E4FA LOA1 R0 #AD1,R1 * COMPARE FOLLOWING BYTES
0121 04A7 EEA4FC COM1 R0 #AD2,R2 * OF BOTH NUMBERS.
0122 04A9 1073 BCTR EQ NEXT IF EQUAL, CONTINUE
0123 04AC 1906 BCTR GT RSET # IF GT, NUMBER ADDRESSED BY
0124 * POINTER 1 IS STILL ACTUAL
0125 * LARGEST NUMBER
0126 * ELSE NEW ACTUAL LARGEST
0127 * NUMBER FOUND
0128 04AE 00E4FB LOA1 R2 SAVE FETCH SAVED POINTER 2
0129 04B1 1F046A BCTR UN FROM
0130 04B4 00E4F9 RSET R0,R1 LAST RESET POINTER 1 AND
0131 04B7 A500 SUB1 R1 #LEN POINTER 2
0132 04B9 00E4FB LOA1 R2 SAVE
0133 04BC 1F0477 BCTR UN LOAD
0134 04BF 03 EXCH LOO2 R3 TEST AND BRANCH TO EXC2 IF
0135 04C0 E1 COM2 R1 ACTUAL LARGEST NUMBER IS
0136 04C1 9000 BCFR EQ EXC1 SAME AS THE LAST NUMBER OF
0137 04C3 00E4FE LOA1 R0 AD3 THE ACTUAL LIST.
0138 04C6 00E4FA COM1 R0 AD1
0139 04C9 1019 BCTR EQ EXC2
0140 04CB 00E4FA EXC1 LOA1 R0 #AD1,R1 EXCHANGE ACTUAL LAST NUMBER
0141 04CE C2 STR2 R2 AND LAST NUMBER OF ACTUAL
0142 04CF 0FE4FE LOA1 R0 #AD3,R3 LIST.
0143 04D2 00E4FA STRA R0 #AD1,R1
0144 04D5 02 LOO2 R2
0145 04D6 CFE4FE STRA R0 #AD3,R3
0146 04D9 D000 B1RR R3 #+2 INCREMENT BOTH POINTERS.
0147 04DB D900 B1RR R1 #+2
0148 04DD ED04F9 COM1 R1 LAST TEST AND BRANCH TO EXC1 IF
0149 04E0 9009 BCFR EQ EXC1 EXCHANGE IS NOT READY
0150 04E2 A700 SUB1 R3 #LEN RESET POINTER 3
0151 04E4 1F0456 EXC2 BCTR UN PASS CONTINUE NEW PASS
0152 *
0153 0440 END SORT

```

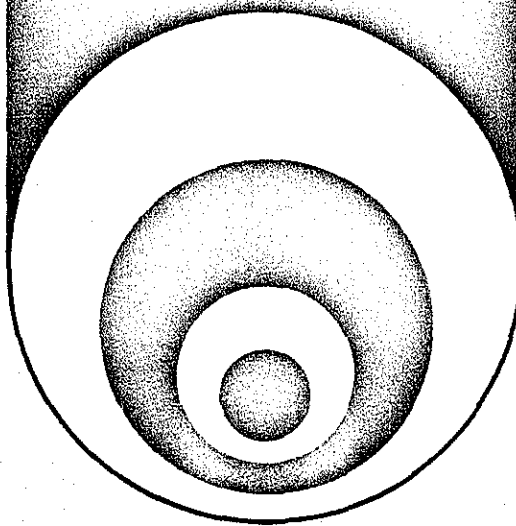
TOTAL ASSEMBLY ERRORS = 0000

**Figure 18**



**signetics**

**MOS  
MICROPROCESSOR**



**BINARY ARITHMETIC  
ROUTINES.....AS53**

## 2650 MICROPROCESSOR APPLICATIONS MEMO

### INTRODUCTION

Binary arithmetic routines, like addition, subtraction, multiplication, and division, are often used in microprocessor-based systems. This applications memo provides several suggested examples for processing binary arithmetic routines on the 2650 microprocessor. These examples include:

- **SIGNED BINARY ADDITION/SUBTRACTION**  
Two-byte operands giving a two-byte result.
- **UNSIGNED BINARY MULTIPLICATION**  
One-byte operands giving a two-byte result.  
Two-byte operands giving a four-byte result.
- **SIGNED BINARY MULTIPLICATION**  
One-byte operands giving a two-byte result.  
Two-byte operands giving a four-byte result.
- **BINARY DIVISION – UNSIGNED AND SIGNED**  
Two-byte dividend and quotient with one-byte divisor and remainder.

In these examples, emphasis is placed on minimizing program memory requirements rather than on processing speed. The different branch instructions and the indexing features of the Signetics 2650 proved useful in minimizing memory requirements.

### 1. BINARY ADDITION/SUBTRACTION FOR TWO-BYTE SIGNED INTEGERS

#### FUNCTION:

Performs the addition or subtraction of two 2-byte signed integers giving a two-byte result.

$(OPR1, OPR1 + 1) +/-(OPR2, OPR2 + 1) \longrightarrow RSLT, RSLT + 1$

#### PARAMETERS:

Input: OPR1, OPR1 + 1 contains augend/subtrahend  
OPR2, OPR2 + 1 contains addend/minuend  
COM-flag in PSL indicates addition/subtraction:  
COM = 0 addition  
COM = 1 subtraction

Output: RSLT, RSLT + 1 contains sum/difference.  
The condition code CC is set to the proper value of the two byte result.  
OPR1, OPR2 and RSLT are MS-bytes.

#### SPECIAL REQUIREMENTS

None

Refer to Figures 1.1 and 1.2 for flowchart and program listing.

| REGISTERS | HARDWARE AFFECTED |     |    |    |     |     |     |
|-----------|-------------------|-----|----|----|-----|-----|-----|
|           | R0                | R1  | R2 | R3 | R1' | R2' | R3' |
|           | X                 | X   |    |    |     |     |     |
| PSU       | F                 | II  | SP |    |     |     |     |
| PSL       | CC                | IDC | RS | WC | OVF | COM | C   |
|           | X                 | X   |    | X  | X   |     | X   |

RAM REQUIRED (BYTES): 6

ROM REQUIRED (BYTES): 45

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE

NESTING LEVELS: None

ASSEMBLER/COMPILER USED: PIPHASM

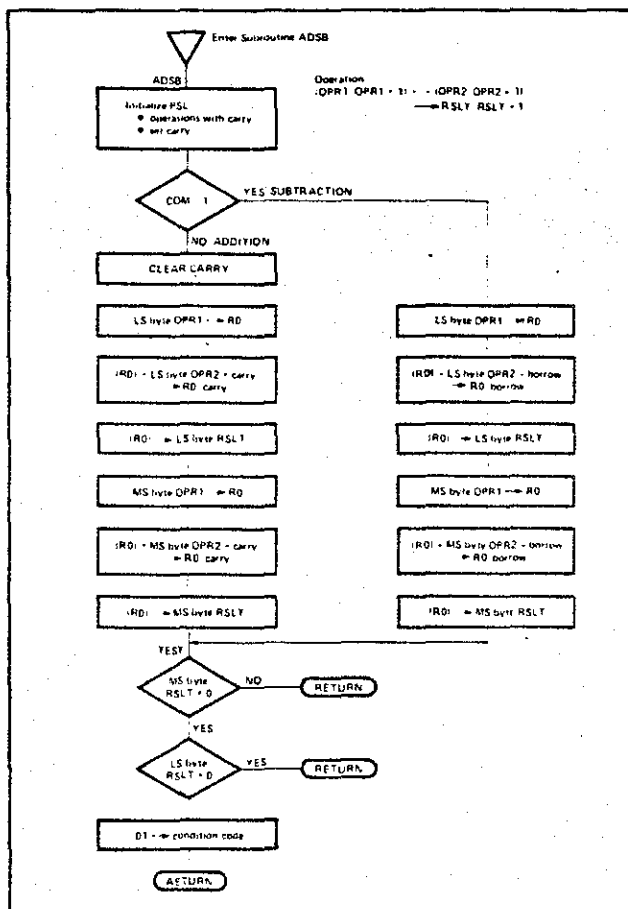


FIGURE 1-1 Flowchart for Double Precision Addition/Subtraction

|    |                    |                                                    |
|----|--------------------|----------------------------------------------------|
| 1  |                    | * PD760010                                         |
| 2  |                    | *****                                              |
| 3  |                    | * BINARY DOUBLE PRECISION ADDITION/SUBTRACTION     |
| 4  |                    | *****                                              |
| 5  |                    | * OPERATION:                                       |
| 6  |                    | * (OPR1,OPR1+1)+/- (OPR2,OPR2+1)-->RSLT,RSLT+1     |
| 7  |                    | * OPR1,OPR2,RSLT ARE MDST SIC BYTES                |
| 8  |                    | * COM IN PSL IS USED AS ADD/SUB FLAG               |
| 9  |                    | * COM=0 IS ADD; COM=1 IS SUBTRACT                  |
| 10 |                    | * AFTER ADD/SUB THE CC,OVF,AND C BITS IN PSL       |
| 11 |                    | * ARE VALID FOR THE RESULT                         |
| 12 |                    | *                                                  |
| 13 |                    | * DEFINITION OF SYMBOLS                            |
| 14 |                    | *                                                  |
| 15 | 0000               | R0 EQU 0 PRDCESSOR REGISTERS                       |
| 16 | 0001               | R1 EQU 1                                           |
| 17 | 0002               | R2 EQU 2                                           |
| 18 | 0003               | R3 EQU 3                                           |
| 19 | 0000               | CC1 EQU H'00' PSL: MSB OF CONDITION CODE           |
| 20 | 0040               | CC0 EQU H'40' LSB OF CONDITION CODE                |
| 21 | 0000               | WC EQU H'00' I=WITH;0=WITHOUT CARRY                |
| 22 | 0002               | COM EQU H'02' I=LOGICAL;0=ARITH COMP               |
| 23 | 0001               | C EQU H'01' CARRY/BORROW                           |
| 24 | 0000               | Z EQU 0 BRANCH COND: ZERO                          |
| 25 | 0003               | UN EQU 3 UNCONDITIDNAL                             |
| 26 | 0000               | ON EQU 0 ALL BITS ARE 1                            |
| 27 |                    | *                                                  |
| 28 |                    | ORG H'500' START OF SUBROUTINE                     |
| 29 |                    | *                                                  |
| 30 | 0500 0500 77 09    | ADSB PPSL WC+C ARITH WITH CARRY;SET CARRY          |
| 31 | 0502 05 02         | LODI,R1 2 LOAD INDEX REGISTER                      |
| 32 | 0504 05 02         | TPSL COM                                           |
| 33 | 0506 10 0F         | BCTR,ON LPSB BRANCH IF SUBTRACTION                 |
| 34 | 0508 75 01         | CPSL C ADDITION,CLEAR CARRY                        |
| 35 | 050A 050A 0D 45 2D | LPAD LODA,R0 OPR1,R1,- BYTE OF FIRST OPERAND TO R0 |
| 36 | 050D 0D 65 2F      | ADDA,R0 OPR2,R1 ADD BYTE OF SECOND DPERAND         |
| 37 | 0510 CD 65 31      | STRA,R0 RSLT,R1 STORE RESULT                       |
| 38 | 0513 59 75         | BRNR,R1 LPAD BRANCH IF NOT DONE                    |
| 39 | 0515 1B 0B         | BCTR,UN TEST                                       |
| 40 | 0517 0517 0D 45 2D | LPSB LODA,R0 OPR1,R1,- BYTE OF FIRST OPERAND TO R0 |
| 41 | 051A AD 65 2F      | SUBA,R0 OPR2,R1 SUB BYTE OF SECOND DPERAND         |
| 42 | 051D CD 65 31      | STRA,R0 RSLT,R1 STORE RESULT                       |
| 43 | 0520 59 75         | BRNR,R1 LPSB BRANCH IF NOT DONE                    |
| 44 | 0522 0522 98 08    | TEST BCFR,Z RTRN RETURN IF MS BYTE NOT ZERO        |
| 45 | 0524 0C 05 32      | LODA,R0 RSLT+1                                     |
| 46 | 0527 14            | RETC,Z RETURN IF LS BYTE ALSO ZERO                 |
| 47 | 0528 75 00         | CPSL CC1 SET CC. TO 01 (POSITIVE)                  |
| 48 | 052A 77 40         | PPSL CC0                                           |
| 49 | 052C 052C 17       | RTRN RETC,UN                                       |
| 50 |                    | *                                                  |
| 51 | 052D               | OPR1 RES 2 LOCATION OF: FIRST OPERAND              |
| 52 | 052F               | OPR2 RES 2 SECOND OPERAND                          |
| 53 | 0531               | RSLT RES 2 RESULT                                  |
| 54 |                    | END                                                |

FIGURE 1-2

## 2. BINARY MULTIPLICATION FOR ONE-BYTE UNSIGNED INTEGERS

### FUNCTION:

One byte by one byte multiplication for unsigned integers, giving a two-byte result.

$(OPR1) \times (OPR2) \rightarrow RSLT, RSLT + 1$

### PARAMETERS:

Input     OPR1 contains multiplier  
          OPR2 contains multiplicand

Output:   RSLT contains high-order product-byte.  
          RSLT + 1 contains low-order product-byte.

### SPECIAL REQUIREMENTS:

None

Refer to Figures 2.1 and 2.2 for flowchart and program listing.

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
|                   | X  | X   | X  | X  |     |     |     |
| PSU               | F  | II  | SP |    |     |     |     |
|                   |    |     |    |    |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |
|                   | X  | X   |    | X  | X   |     | X   |

RAM REQUIRED (BYTES): 4

ROM REQUIRED (BYTES): 29

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE  
NESTING LEVELS: None

ASSEMBLER/COMPILER USED: PIPHASM

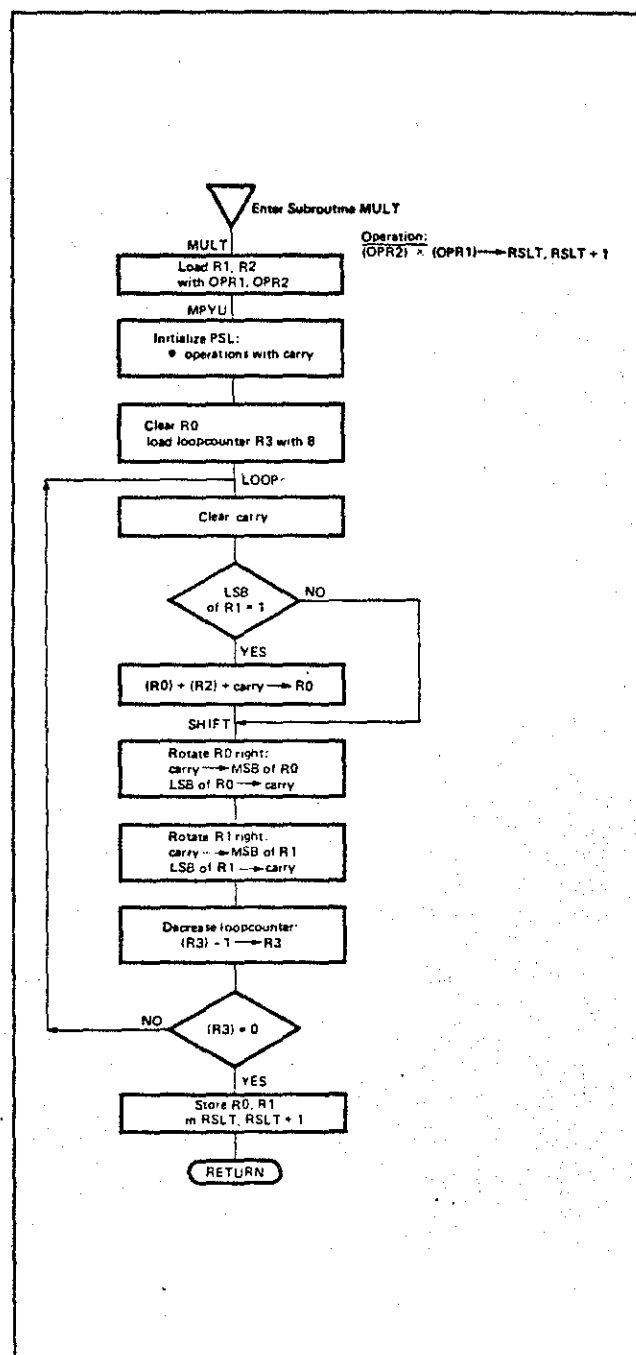


FIGURE 2-1 Flowchart for Unsigned Multiplication (One-Byte Operands; Two-Byte Result)

```

1 * PD760030
2 *
3 * *****
4 * BINARY MULTIPLICATION FOR 2 UNSIGNED INTEGERS
5 * *****
6 *
7 * MULTIPLIER IS IN OPR1
8 * MULTIPLICAND IS IN OPR2
9 * RESULT WILL BE STORED IN RSLT,RSLT+1 (RSLT = MS BYTE)
10 *
11 *
12 *
13 * SYMBOL DEFINITIONS
14 0000 R0 EQU 0
15 0001 R1 EQU 1
16 0002 R2 EQU 2
17 0003 R3 EQU 3
18 0001 R4 EQU 1
19 0002 R5 EQU 2
20 0003 R6 EQU 3
21 0003 UN EQU 3 UNCONDITIONAL BRANCHING
22 0000 ON EQU 0
23 0002 LT EQU 2
24 0000 Z EQU 0
25 0001 P EQU 1
26 0002 N EQU 2
27 0000 WC EQU 0
28 0001 C EQU 1
29 0000 F EQU 0
30 0004 OVF EQU 4
31 0002 CDM EQU 2
32 *
33 * R/W MEMORY
34 *
35 ORG H'500'
36 0500 OPR1 RES 2
37 0502 OPR2 RES 2
38 0504 RSLT RES 4
39 *
40 *
41 *
42 ORG H'600'
43 0600 0600 0D 05 00 MULT LODA,R1 DPR1 GET OPERAND IN R1
44 0603 0600 0E 05 02 LODA,R2 DPR2 GET OPERAND IN R2
45 0606 0606 77 00 MPYU PPSL WC ARITH
46 0608 20 EORZ R0 CLEAR R0
47 0609 07 00 LODI,R3 0 LDAD LOOP COUNTER R3
48 060B 060B 75 01 LDOP CPSL C CLEAR CARRY
49 060D F5 01 TMI,R1 H'01'
50 060F 98 01 BCFR,ON SHFT SKIP ADDITION IF LSB R1=0
51 0611 82 ADDZ R2 ADD MULTIPLICAND TO PARTIAL PROD
52 0612 0612 50 SHFT RRR,R0 RDTATE PARTIAL PROD AND MULTIPLIER
53 0613 51 RRR,R1
54 0614 FB 75 BDRR,R3 LOOP BRANCH TO LDOP IF NOT READY
55 0616 CC 05 04 STRA,R1 RSLT SAVE RESULT IN RESULT AREA
56 0619 CD 05 04 STRA,R1 RSLT+1 SAVE RESULT IN RESULT AREA
57 061C 17 RETC,UN RETURN TO MAIN PROGRAM

```

FIGURE 2-2

### 3. BINARY MULTIPLICATION FOR TWO-BYTE UNSIGNED INTEGERS

#### FUNCTION:

Two byte by two byte multiplication for unsigned integers, giving a four byte result.

$(OPR2, OPR2 + 1) \times (OPR1, OPR1 + 1) \longrightarrow$   
 $RSLT, RSLT + 1, RSLT + 2, RSLT + 3$

#### PARAMETERS:

Input:  $(OPR1, OPR1 + 1)$  contains multiplier  
 $(OPR2, OPR2 + 1)$  contains multiplicand

Output:  $RSLT, RSLT + 1, RSLT + 2, RSLT + 3$  contains product.  
 $OPR1, OPR2$ , and  $RSLT$  are most-significant bytes.

#### SPECIAL REQUIREMENTS:

None

Refer to Figures 3.1 and 3.2 for flowchart and program listing.

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
|                   | X  | X   |    | X  |     |     |     |
| PSU               | F  | II  | SP |    |     |     |     |
|                   |    |     |    |    |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |
|                   | X  | X   |    | X  | X   |     | X   |

RAM REQUIRED (BYTES): 8

ROM REQUIRED (BYTES): 57

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE

NESTING LEVELS: None

ASSEMBLER/COMPILER USED: PIPHASM

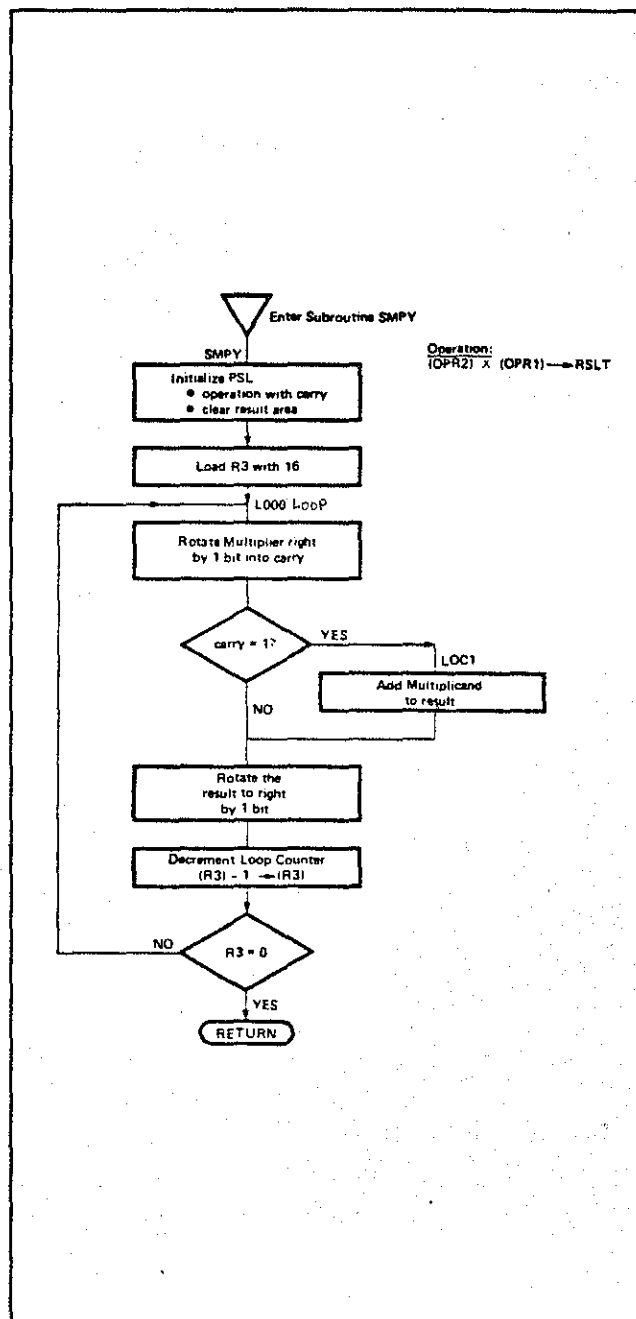


FIGURE 3-1 Flowchart for Unsigned Multiplication  
 (Two-Byte Operands; Four-Byte Result)

```

58 * PD76#031 *
59 *
60 * BINARY MULTIPLICATION FOR 2 TWO-BYTE INTEGERS
61 *
62 *
63 * MULTIPLIER IS IN OPR1 , OPR1+1
64 * MULTIPLICAND IS IN OPR2 ,OPR2+1
65 * RESULT WILL BE IN RSLT ,RSLT+1 ,RSLT+2 ,RSLT+3
66 ORC H'790'
67 *
68 0790 0790 77 0B SHPT PPSL WC SET MODE
69 0792 20 EORZ R0
70 0793 CC 05 04 STRA,R0 RSLT CLEAR RESULT
71 0796 CC 05 05 STRA,R0 RSLT+1 CLEAR RESULT +1
72 0799 07 10 LODI,R3 16 LOAD COUNT
73 079B 079B 05 FE LOOO LODI,R1 -2 TO GET 254
74 079D 75 01 CPSL C CLEAR CARRY
75 079F 079F 0D 64 02 LOCO LODA,R0 OPR1-256+2,R1 FOR INDEXING INTO OPR1
76 07A2 50 RRR,R0 ROTATE RIGHT WITH C
77 07A3 CD 64 02 STRA,R0 OPR1-256+2,R1
78 07A6 D9 77 BIRR,R1 LOCD ROTATE 2ND TIME
79 * * THIS ROTATES MULTIPLIER BY 1 BIT TO GET THE LSB
80 * * INTO CARRY
81 07A8 20 EORZ R0 CLEAR R0
82 07A9 D0 RRL,R0 GET CARRY INTO LSB
83 07AA FB 02 BDRR,R0 LOC1
84 07AC 1B 0D BCTR,UN LOC4
85 *
86 07AE 07AE 05 02 LOC1 LODI,R1 2 GET INDEX
87 07B0 07B0 0D 65 03 LOC2 LODA,R0 RSLT-1,R1 ADD MULTIPLICAND TO PRODUCT
88 07B3 0D 65 01 ADDA,R0 OPR2-1,R1
89 07B6 CD 65 03 STRA,R0 RSLT-1,R1
90 07B9 F9 75 BDRR,R1 LOC2 FINISH THE ADD
91 *
92 *
93 07BB 07BB 05 FC LOC4 LODI,R1 -4 ROTATE THE PRODUCT TO RIGHT
94 07BD 07BD 0D 64 0B LOC5 LODA,R0 RSLY-256+4,R1
95 07C0 50 RRR,R0 ROTATE RESULT
96 07C1 CD 64 0B STRA,R0 RSLT-256+4,R1
97 07C4 D9 77 BIRR,R1 LOC5
98 07C6 FB 53 BDRR,R3 LODO FINISH THE LOOP
99 07C8 17 RETC,UN

```

FIGURE 3-2

#### 4. BINARY MULTIPLICATION FOR ONE-BYTE SIGNED INTEGERS

##### FUNCTION:

One byte by one byte multiplication for signed integers giving a two-byte result.

$(OPR1) \times (OPR2) \rightarrow RSLT, RSLT + 1$

The Booth algorithm is used (see Figure 4.1).

##### PARAMETERS:

Input: OPR1 contains multiplier  
OPR2 contains multiplicand

Output: RSLT contains high-order product byte.  
RSLT + 1 contains low-order product byte.

##### SPECIAL REQUIREMENTS:

None

Refer to Figures 4.1 and 4.2 for flowcharts and to Figure 4.3 for program listing.

| REGISTERS | HARDWARE AFFECTED |     |    |    |     |     |     |  |
|-----------|-------------------|-----|----|----|-----|-----|-----|--|
|           | R0                | R1  | R2 | R3 | R1' | R2' | R3' |  |
| PSU       | F                 | II  | SP |    |     |     |     |  |
| PSL       | CC                | IDC | RS | WC | OVF | COM | C   |  |
|           | X                 | X   |    | X  | X   |     | X   |  |

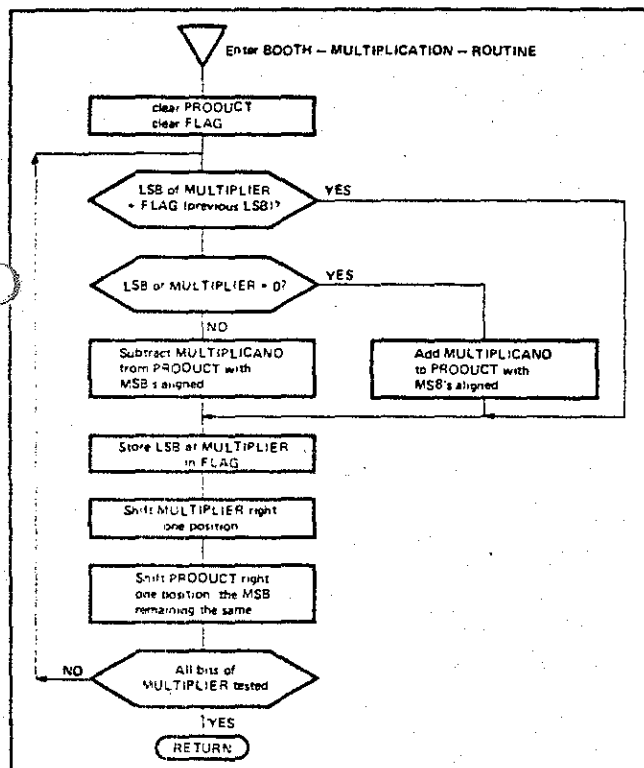


FIGURE 4-1 Flowchart of Booth Algorithm  
Multiplicand X Multiplier → Product

RAM REQUIRED (BYTES): 4

ROM REQUIRED (BYTES): 51

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE NESTING LEVELS: None

ASSEMBLER/COMPILER USED: PIPHASM

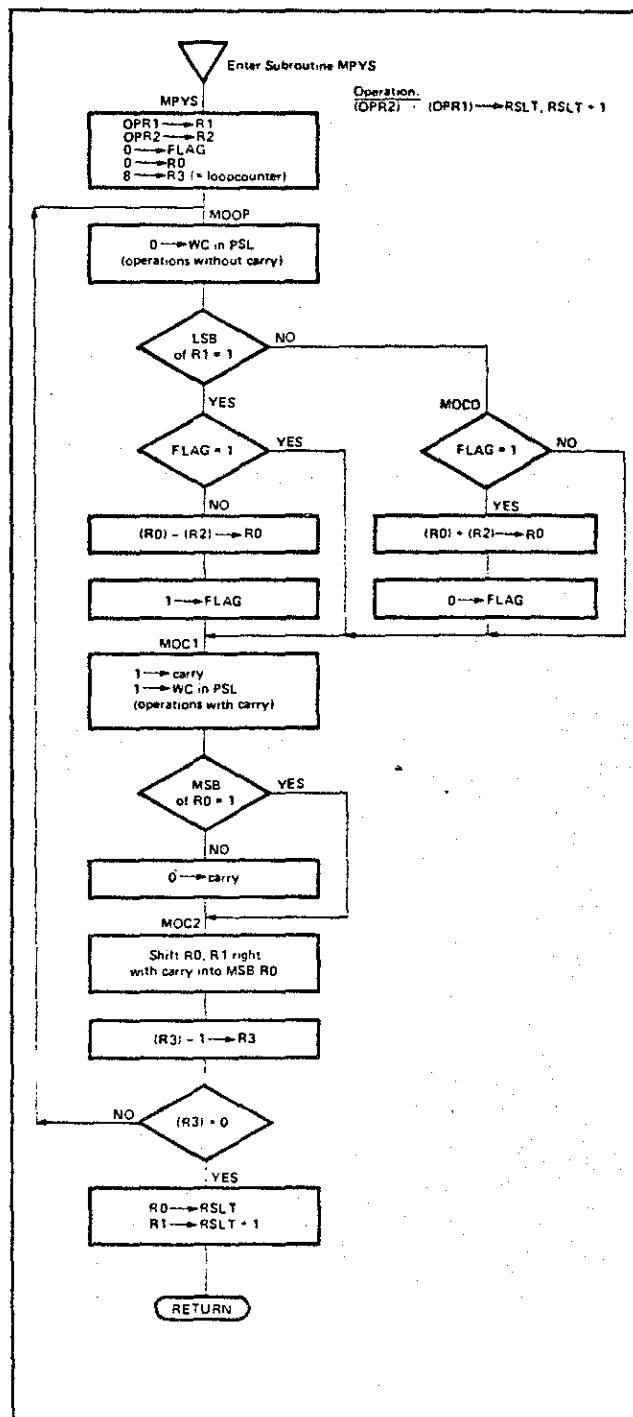


FIGURE 4-2 Flowchart for Signed Multiplication Using Booth Algorithm (One-Byte Operands; Two-Bytes Result)



## 5. BINARY MULTIPLICATION FOR TWO-BYTE SIGNED INTEGERS

### FUNCTION:

Two byte by two byte multiplication for signed integers giving a four byte result.

$(OPR1, OPR1 + 1) \times (OPR2, OPR2 + 1)$

→ RSLT, RSLT + 1, RSLT + 2, RSLT + 3.

The Booth algorithm (Figure 4.1) is used.

### PARAMETERS:

Input: OPR1, OPR1 + 1 contains multiplicand  
OPR2, OPR2 + 1 contains multiplier

Output: RSLT, RSLT + 1, RSLT + 2, RSLT + 3 contains product.  
OPR1, OPR2, and RSLT are most-significant bytes.

### SPECIAL REQUIREMENTS

None

Refer to Figure 5.1 for flowchart and to Figure 5.2 for program listing.

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
|                   | X  | X   | X  | X  |     |     |     |
| PSU               | F  | II  | SP |    |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |
|                   | X  | X   |    | X  | X   |     | X   |

RAM REQUIRED (BYTES): 8

ROM REQUIRED (BYTES): 71

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE  
NESTING LEVELS: None

ASSEMBLER/COMPILER USED: PIPHASM

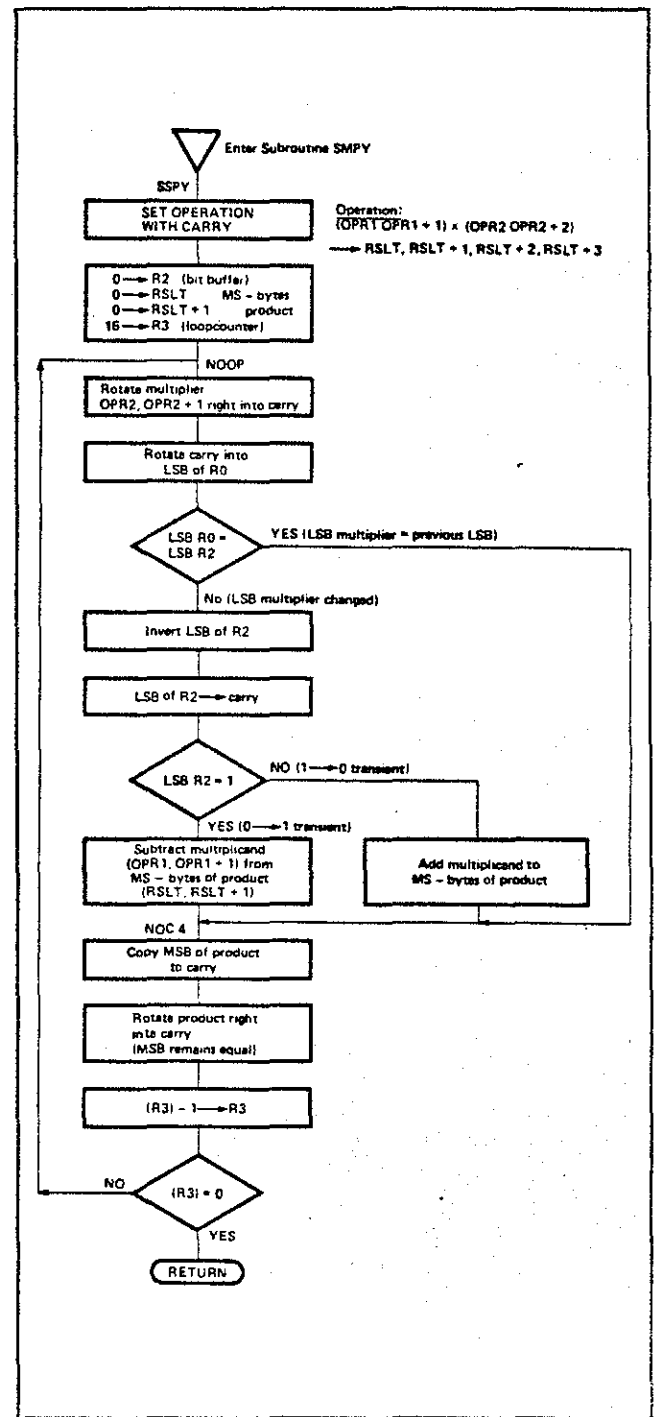


FIGURE 5-1 Flowchart for Signed Multiplication Using Booth Algorithm (Two-Byte Operands; Four-Byte Result)

```

138 * PD760033
139 *****
140 * BINARY MULTIPLICATION FOR TWO BYTE SIGNED INTEGERS
141 *****
142 * MULTIPLICAND IS IN LOCATIONS OPR1,OPR1+1
143 * MULTIPLIER IS IN LOCATIONS OPR2,OPR2+1
144 *
145 * RESULT WILL BE STORED IN RSLT,RSLT+1,RSLT+2,RSLT+3
146 *
147 * AFTER MULTIPLICATION THE MULTIPLICAND IS UNCHANGED
148 * THE MULTIPLIER IS DESTROYED
149 * THE MULTIPLICAND MUST BE UNEQUAL H'8000'
150 *****
151 0033 0033 77 00 SSPT PPSL WC ARITH AND ROTATE WITH C
152 0035 20 EORZ R0 CLEAR R0
153 0036 C2 STRZ R2 CLEAR R2
154 0037 CC 05 04 STRA,R0 RSLT CLEAR Z MSBYTES OF PRODUCT
155 003A CC 05 05 STRA,R0 RSLT+1
156 003D 07 10 LODI,R3 16 LOAD LOOP COUNTER R3
157 003F 003F 05 FE NOOP LODI,R1 -2 LOAD INDEX REG WITH 254
158 0041 0041 0D 64 04 NOC0 LODA,R0 OPR2-256+2,R1 ROTATE MULTIPLIER
159 0044 50 RRR,R0 INTO CARRY
160 0045 CD 64 04 STRA,R0 OPR2-256+2,R1
161 0048 D9 77 BIRR,R1 NOC0 BRANCH IF NOT DONE
162 004A 20 EORZ R0 CLEAR R0
163 004B D0 RRL,R0 ROTATE CARRY IN LSB OF R0
164 004C 22 EORZ R2 LSB OF R0 BECOMES 1 FOR CHANGE
165 004D 18 19 BCTR,Z NOC4 BRANCH IF NO CHANGE
166 004F 22 EORZ R2 INVERT LSB OF R2
167 0050 C2 STRZ R2 RESTORE NEW R2
168 0051 50 RRR,R0 LSB OF R2 INTO CARRY OR BORROW
169 0052 05 02 LOBI,R1 2 LOAD INDEX
170 0054 0054 0D 45 04 NOC1 LODA,R0 RSLT,R1,-1 LOAD BYTE OF RSLT IN R0
171 0057 F6 01 TH1,R2 1
172 0059 18 05 BCTR,0W NOC2 BRANCH TO SUBTRACT IF LSB R2=1
173 005B 8D 65 00 ADDA,R0 OPR1,R1 ADD BYTE MPLCND TO RSLT
174 005E 1B 03 BCTR,UN NOC3
175 0060 0060 AD 65 00 NOC2 SUBA,R0 OPR1,R1 SUB BYTE MPLCND FROM RSLT
176 0063 0063 CD 65 04 NOC3 STRA,R0 RSLT,R1 RESTORE INTERMEDIATE RSLT
177 0066 59 6C BRNR,R1 NOC1 BRANCH IF ADD SUBTRACT NOT READY
178 *
179 0068 0068 0C 05 04 NOC4 LODA,R0 RSLT
180 006B D0 RRL,R0
181 006C 04 FC LODI,R0 -4 LOAD INDEX
182 006E 006E 0D 64 00 NOC5 LODA,R0 RSLT-256+4,R1 FETCH MS BYTE PRODUCT
183 0071 50 RRR,R0 ROTATE RSLT,PROD+1 ETC TO RIGHT
184 0072 CD 64 00 STRA,R0 RSLT-256+4,R1 KEEPING MSB SAME
185 0075 D9 77 BIRR,R1 NOC5 BRANCH IF NOT DONE
186 0077 FB 46 PDRR,R3 NOOP BRANCH IF LOOP NOT READY
187 0079 17 RETC,UN RETURN TO MAIN PROGRAM
188 END

```

FIGURE 5-2

## 6. BINARY DIVISION

A. UNSIGNED INTEGERS  
TWO-BYTE DIVIDEND; ONE-BYTE DIVISOR

## FUNCTION:

Division of a two byte dividend by a one byte divisor, resulting in a two-byte quotient and a one-byte remainder.

$(DVDN, DVDN + 1) \xrightarrow{(DVSr)} \begin{cases} (DVDN, DVDN + 1) \text{ (quotient)} \\ R1 \text{ (remainder)} \end{cases}$

## PARAMETERS:

Input:  $DVDN, DVDN + 1$  contains dividend  
 $DVSr$  contains divisor  
 $DVDN$  is most-significant byte

Output:  $DVDN, DVDN + 1$  contains quotient  
 $R1$  contains remainder  
 $DVDN$  is most-significant byte.  
 Dividend is destroyed after execution of division.

## SPECIAL REQUIREMENTS:

None

Refer to Figure 6.1 for flowchart and to Figure 6.2 for program listing.

|           | HARDWARE AFFECTED |     |    |    |     |     |     |  |
|-----------|-------------------|-----|----|----|-----|-----|-----|--|
| REGISTERS | R0                | R1  | R2 | R3 | R1' | R2' | R3' |  |
|           | X                 | X   | X  | X  |     |     |     |  |
| PSU       | F                 | II  | SP |    |     |     |     |  |
|           |                   |     |    |    |     |     |     |  |
| PSL       | CC                | IDC | RS | WC | OVF | COM | C   |  |
|           | X                 | X   |    | X  | X   | X   | X   |  |

RAM REQUIRED (BYTES): 3

ROM REQUIRED (BYTES): 45

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE  
 NESTING LEVELS: None

ASSEMBLER/COMPILER USED: PIPHASM

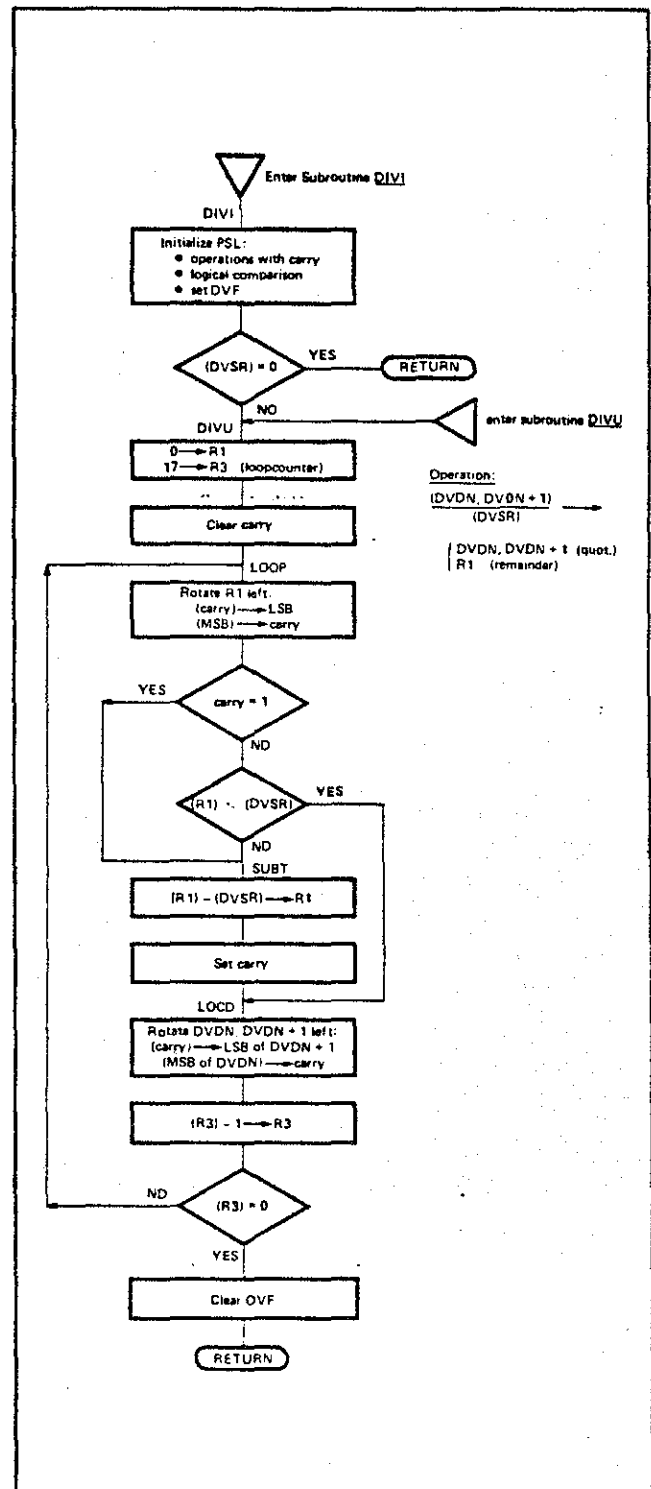


FIGURE 6-1 Flowchart for Unsigned Division (Dividend or Quotient: Two-Bytes; Divisor or Remainder: One-Byte)

```

1 * PD768848 *
2 *****
3 * BINARY DIVISIONS FOR INTEGERS
4 *****
5 * DIVIDEND IS IN DVDN,DVDN+1 16 BITS
6 * DIVISOR IS IN DVSR 8 BITS
7 * QUOTIENT WILL BE IN DVDN,DVDN+1 16 BITS
8 * AFTER DIVISION, DIVIDEND WILL BE DESTROYED
9 * R1 WILL HOLD REMAINDER
10 * DVF=1 IMPLIES OVERFLOW
11 *
12 *
13 * SYMBOL DEFINITIONS
14 R0 EQU 0
15 R1 EQU 1
16 R2 EQU 2
17 R3 EQU 3
18 R4 EQU 1
19 R5 EQU 2
20 R6 EQU 3
21 UN EQU 3
22 C EQU 1
23 ON EQU 0
24 LT EQU 2
25 Z EQU 0
26 EQ EQU 0
27 P EQU 1
28 N EQU 2
29 MC EQU 0
30 OV EQU 4
31 CM EQU 2
32 *
33 ORG H'500'
34 *
35 DIVI PPSL MC+OV+CM ARITH ROTATE WITH CARRY
36 LODA,R0 DVSR FETCH DIVISOR
37 RETC,Z RETURN WITH DVF =1 IF DVSR =0
38 *
39 DIVU LODI,R1 0 CLR R1
40 LODI,R3 17 LOAD LOOP COUNTER R3
41 CPSL C CLEAR CARRY
42 LOOP RRL,R1 ROTATE CARRY IN LSB OF R1
43 TPSL C
44 BCTR,ON SUBT GO TO SUBTRACT IF CARRY =1
45 COMA,R1 DVSR
46 BCTR,LT LOC0 IF R1<DVSR,NO SUBTRACTION
47 SUBT PPSL C CLR BORROW
48 SUBA,R1 DVSR SUBTR DVSR FROM REMAINDER
49 PPSL C SET CARRY
50 LOC0 LODI,R2 2 LOAD INDEX REGISTER
51 LOC1 LODA,R0 DVDN,R2,- ROTATE QUOTIENT BIT
52 RRL,R0 DVDN,DVDN+1 AND MSB DF
53 STRA,R0 DVDN,R2 DVDN INTO CARRY
54 BRNR,R2 LOC1 BRANCH IF ROTATE NOT READY
55 BDRR,R3 LOOP BRANCH IF DIVISION NOT READY
56 CPSL OV OVF CLEAR OVF IN PSL
57 RETC,UN RETURN TO MAIN PROGRAM
58 *

```

FIGURE 6-2

## B. SIGNED INTEGERS

### TWO-BYTE DIVIDEND; ONE-BYTE DIVISOR

#### FUNCTION:

Division of a two-byte dividend by a one-byte divisor, resulting in a two-byte quotient and a one-byte remainder.

$(DNDN, DNDN + 1) \xrightarrow{(DVSr)} \begin{cases} DNDN, DNDN + 1 & \text{(quotient)} \\ R1 & \text{(remainder)} \end{cases}$

#### PARAMETERS:

Input: DNDN, DNDN + 1 contains dividend  
DVSr contains divisor  
DNDN is most-significant byte.

Output: DNDN, DNDN + 1 contains quotient  
R1 contains remainder  
DNDN is most-significant byte.  
Dividend is destroyed after execution of division;  
negative divisor becomes positive

#### SPECIAL REQUIREMENTS:

Software: Unsigned division subroutine

Refer to Figure 6.3 for flowchart and to Figure 6.4 for program listing.

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
|                   | X  | X   | X  | X  |     |     |     |
| PSU               | F  | II  | SP |    |     |     |     |
| PSL               | CC | IDC | RS | WC | DVF | CDM | C   |
|                   | X  | X   |    | X  | X   | X   | X   |

|                          |          |
|--------------------------|----------|
| RAM REQUIRED (BYTES):    | 4        |
| ROM REQUIRED (BYTES):    | 61       |
| EXECUTION TIME:          | Variable |
| MAXIMUM SUBROUTINE       |          |
| NESTING LEVELS:          | 1        |
| ASSEMBLER/COMPILER USED: | PIPHASM  |

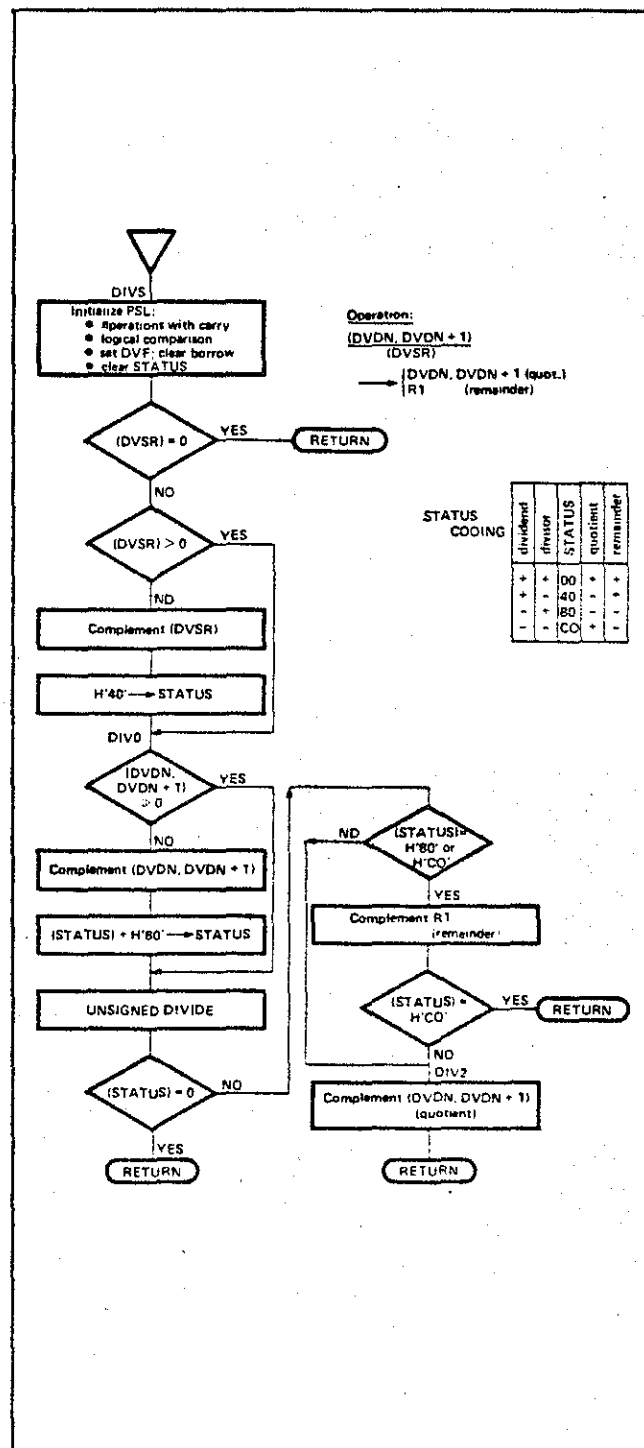


FIGURE 6-3 Flowchart for Signed Division (Dividend & Quotient: 2 Bytes; Divisor & Remainder: 1 Byte)

## INTRODUCTION

The numbers used in digital systems are usually expressed in binary notation. Some commonly used formats are:

- magnitudes only for unsigned numbers
- 1's complement and 2's complement for signed numbers.

However, binary numbers are difficult to interpret, and man-machine interface can be greatly improved by presenting numbers in decimal notation. Since virtually all digital systems operate on numbers in binary form (i.e., 1's and 0's), decimal numbers must be converted to binary during the input process, and reconverted to decimal notation during the output process. In cases where decimal input and/or output is required, the ideal solution would be a digital system capable of interpreting and processing decimal numbers.

This applications memo describes several methods of handling binary-coded-decimal (BCD) numbers with the Signetics 2650 microprocessor. Special provisions in the 2650 for these operations, including the Interdigit Carry (IDC) flag bit and the Decimal Adjust Register (DAR) instruction, are discussed. These provisions greatly simplify interfacing of the 2650 to decimal-oriented peripheral devices, such as CRT display terminals, printers, and keyboards. Basic arithmetic routines (add, subtract, multiply, and divide) for both signed integers and signed fixed-point numbers are given.

## BCD NOTATION

In BCD notation, each decimal digit requires a 4-bit code as indicated below:

|          |          |
|----------|----------|
| 0 = 0000 | 5 = 0101 |
| 1 = 0001 | 6 = 0110 |
| 2 = 0010 | 7 = 0111 |
| 3 = 0011 | 8 = 1000 |
| 4 = 0100 | 9 = 1001 |

Codes 1010 through 1111 are not used.

Two decimal digits can be packed into one 8-bit byte—the size of a 2650 data word. The range within 1 byte is consequently 00<sub>10</sub> through 99<sub>10</sub>. For instance, the number 15<sub>10</sub> is coded as 00010101.

## CARRY (C) AND INTERDIGIT CARRY (IDC) FLAGS

The Program Status Lower (PSL) of the 2650's Program Status Word (PSW) register contains 2 carry flags: Carry (C) and Interdigit Carry (IDC). During execution of any arithmetic instruction, both flags are set or

reset depending on the result of the operation, as illustrated in Figure 1:

- The Carry (C) flag is set as a result of a carry (or no borrow) out of the most-significant-bit (bit 7) of the affected register Rx, and hence out of the most-significant BCD digit.
- The Interdigit Carry (IDC) flag is set as a result of a carry (or no borrow) out of bit 3, and hence out of the least-significant BCD digit and into the most-significant BCD digit.

## DECIMAL ADJUST REGISTER (DAR) INSTRUCTION

If 2 BCD numbers are added or subtracted by means of binary arithmetic instructions, the result may not be a BCD number. For example:

$$\begin{aligned} 23_{10} + 56_{10} &= 79_{10}; \\ &\text{but} \\ 18_{10} + 35_{10} &= 4D_{10}. \end{aligned}$$

Since the binary codes 1010 (A<sub>16</sub>) through 1111 (F<sub>16</sub>) are not used in BCD, the result of a binary arithmetic instruction may need a correction of (+6) in case of an add operation or (-6) in case of a subtract operation. The 2650 performs this correction by means of the Decimal Adjust Register (DAR) instruction. This 1-byte instruction condition-

ally adds a decimal 10 (2's complement negative 6 in a 4-bit binary number system) to either the high order 4-bits and/or the low order 4 bits of a specified register Rx, which may be any of the 2650's seven CPU registers.

The truth table of Figure 2 indicates the logical operation performed. The operation proceeds based on the values of the Carry (C) and Interdigit Carry (IDC) flags in the Program Status Word. The C and IDC remain unchanged by the execution of this instruction.

The WC (With/Without Carry) bit in PSL has no influence on the DAR instruction.

## GENERAL SUBTRACTION RULES

In the case of subtraction, a correction of (-6) is required for the digit(s) which generate a borrow upon execution of the subtract instruction. This can be performed directly by the DAR instruction.

## Single-Byte Operands/Result:

Subtraction of single-byte operands is done by performing the subtract instruction and then performing the DAR instruction; the borrow bit must be cleared initially. See Example A.

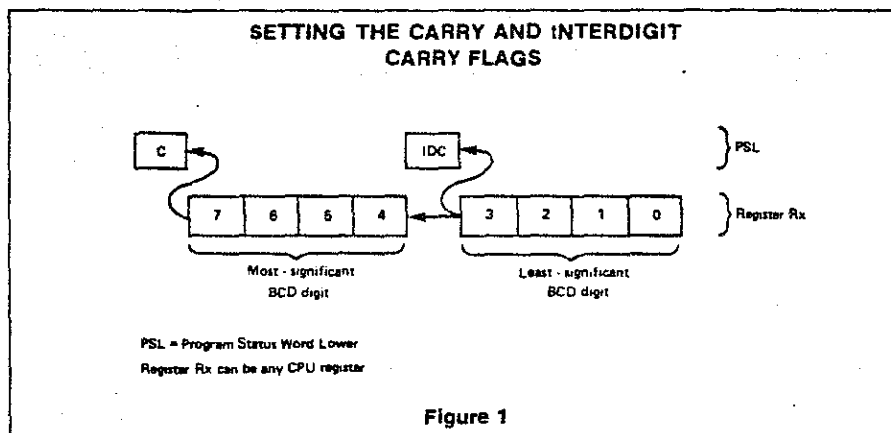


Figure 1

## TRUTH TABLE FOR DAR INSTRUCTION

| BEFORE: DAR, Rx |     |     |     | AFTER: DAR, Rx |     |                    |                    |
|-----------------|-----|-----|-----|----------------|-----|--------------------|--------------------|
| C               | IDC | Rx  |     | C              | IDC | Rx                 |                    |
|                 |     | MSD | LSD |                |     | MSD                | LSD                |
| 0               | 0   | a   | b   | 0              | 0   | a+10 <sub>10</sub> | b+10 <sub>10</sub> |
| 0               | 1   | a   | b   | 0              | 1   | a+10 <sub>10</sub> | b                  |
| 1               | 0   | a   | b   | 1              | 0   | a                  | b+10 <sub>10</sub> |
| 1               | 1   | a   | b   | 1              | 1   | a                  | b                  |

NOTE

IDC is not added to the upper digit in the 'a+10<sub>10</sub>' operation.

Figure 2

If the With Carry (WC) bit in PSL is zero (no carry/borrow), the first instruction is not required.

### Multiple-Byte Operands/

#### Result:

When dealing with multiple-byte operands, arithmetic operations *including carry*, are required. Hence, the WC bit in PSL must be set to 1 prior to execution. If indexing is used, multiple-byte subtraction is simple, as illustrated in Example B.

NOTE: OPR1, OPR2 and RSLT are the most-significant bytes.

### GENERAL ADDITION RULES

For addition, a correction of (+6) is required if the sum of the most-significant digits or least-significant digits exceeds 9. This is accomplished by first adding an offset of (+6) to each of the digits of the first operand (addition of H'66') and then adding the second operand.

If the sum of the least-significant digits did exceed 9, it now (including the (+6) correction) will exceed 15<sub>10</sub> (H'F'); an Interdigit Carry will be generated. If an IDC is generated, the result is correct and, as shown in Figure 2, the DAR instruction will have no effect on the sum. If not, the (+6) correction will be cancelled by adding 10 (equivalent to subtracting 6). Correction of the most-significant digit sum operates similarly, with the C bit controlling the final correction.

### Single-Byte Operands/Result:

If the 2650 is conditioned for arithmetic without carry (WC = 0), addition can be performed as shown in Example C.

In the case of arithmetic with carry (WC = 1), it should be noted that the addition of the offset H'66' may generate a carry (if OPR1 = 99 and carry was set); this carry will be added during the addition of OPR2, giving an incorrect sum.

### Multiple-Byte Operands/Result:

When using multiple-byte operands, linking of the bytes by means of the carry bit is required. Hence, arithmetic with carry must be performed (WC in PSL is set to 1). Because of the two successive additions (of the offset H'66' and of the second operand), the problem mentioned in the previous section can also arise here. Two straightforward solutions to this problem, listed below, are illustrated in the flowchart of Figure 3.

**Method 1:** In this method, each byte of the first operand is first increased by the offset H'66', after which addition of the second operand is performed. See Example D.

|         |      |                         |
|---------|------|-------------------------|
| PSSL    | C    | CLEAR BORROW            |
| LOOA,R3 | OPR1 | FETCH FIRST OPERAND     |
| SUBA,R3 | OPR2 | SUBTRACT SECOND OPERAND |
| OAR,R3  |      | DECIMAL ADJUST RESULT   |
| STRA,R3 | RSLT | STORE RESULT            |

Example A

|              |           |                                     |
|--------------|-----------|-------------------------------------|
| PSSL         | WC+C      | ARITHMETIC WITH CARRY, CLEAR BORROW |
| LODI,R3      | LENG      | LOAD INDEX REGISTER                 |
| DSUL LODA,R0 | OPR1,R3,- | FETCH BYTE OF OPERAND1              |
| SUBA,R0      | OPR2,R3   | SUBTRACT BYTE OF OPERAND2           |
| OAR,R0       |           | DECIMAL ADJUST RESULT               |
| STRA,R0      | RSLT,R3   | STORE RESULTING BYTE                |
| BRNR,R3      | DSUL      | CONTINUE LOOP IF NOT OONE           |

Example B

|         |       |                        |
|---------|-------|------------------------|
| LODA,R3 | OPR1  | FETCH FIRST OPERAND    |
| ADDI,R3 | H'66' | ADD OFFSET FOR BCD AOD |
| ADOA,R3 | OPR2  | AOD SECOND OPERAND     |
| DAR,R3  |       | DECIMAL ADJUST RESULT  |
| STRA,R3 | RSLT  | STORE RESULT           |

Example C

|      |         |           |                                |
|------|---------|-----------|--------------------------------|
|      | CPSL    | C         | CLEAR CARRY                    |
|      | PSSL    | WC        | ARITHMETIC WITH CARRY          |
|      | LOOI,R3 | LENG      | LOAD INDEX REGISTER            |
| ADD0 | LODA,R0 | OPR1,R3,- | FETCH BYTE OF OPERAND1         |
|      | ADDI,R0 | H'66'     | ADD OFFSET FOR BCD AOD         |
|      | STRA,R0 | RSLT,R3   | STORE INTERMEDIATE RESULT      |
|      | BRNR,R3 | ADO0      | BRANCH IF ALL BYTES NOT READY  |
| ADD1 | LOOI,R3 | LENG      | LOAD INDEX REGISTER            |
|      | LODA,R0 | RSLT,R3,- | FETCH BYTE OF INTERMEDIATE SUM |
|      | ADOA,R0 | OPR2,R3   | ADD BYTE OF OPERAND2           |
|      | OAR,R0  |           | DECIMAL ADJUST RESULT          |
|      | STRA,R0 | RSLT,R3   | STORE RESULT                   |
|      | BRNR,R3 | ADO1      | BRANCH IF ALL BYTES NOT READY  |

Example D

**Method 2:** In this method, the complete addition is handled on a byte-by-byte basis. This means that the true interbyte-carry must be saved and restored, and the carry must be cleared at the appropriate time. This can be performed by using one additional register to retain the interbyte-carry. See Example E.

The second method is faster and requires fewer bytes of code (24 versus 30) but requires an additional register.

The program listing of Figure 5 summarizes the basic BCD addition and subtraction routines.

### ROUTINES FOR SIGNED INTEGER ARITHMETIC

There are several possible ways of representing signed decimal numbers. The best known are ten's complement notation and sign-magnitude notation, illustrated in Figure 4, is used here because it is easy to interpret and lends itself to interfacing with peripherals. It is also simpler to use in multiplication, division, and in aligning and rounding routines. The numbers are stored in memory in the form of a sign followed by the absolute value of the number.

The length of the numbers is defined by the number of bytes (including the sign byte) they require. This parameter can be modified by changing the definition of LENG in the source program. Note that for clarity, each routine is written in a "stand-alone" form. If more than 1 routine is required in a program, considerable savings in the program space required can be realized by breaking out common operations as subroutines.

|         |         |                                     |
|---------|---------|-------------------------------------|
| CPSL    | C       | CLEAR CARRY                         |
| PPSL    | WC      | ARITHMETIC/ROTATE WITH CARRY        |
| LODI,R3 | LENG    | LOAD INDEX REGISTER                 |
| LODI,R1 | 0       | CLEAR INTERBYTE-CARRY REGISTER      |
| DADL    | LODA,R0 | FETCH BYTE OF OPERAND1              |
|         | ADDI,R0 | ADD OFFSET FOR BCD ADD              |
|         | RRR,R1  | RESTORE INTERBYTE-CARRY TO CARRY    |
|         | ADDA,R0 | ADD BYTE OF OPERAND2                |
|         | DAR,R0  | DECIMAL ADJUST RESULT               |
|         | STRA,R0 | STORE RESULT                        |
|         | RRL,R1  | SAVE INTERBYTE-CARRY IN R1, CLEAR C |
|         | BRNR,R3 | BRANCH IF NOT READY                 |

Example E

### GENERAL ADDITION FOR MULTIPLE-BYTE, UNSIGNED BCD NUMBERS

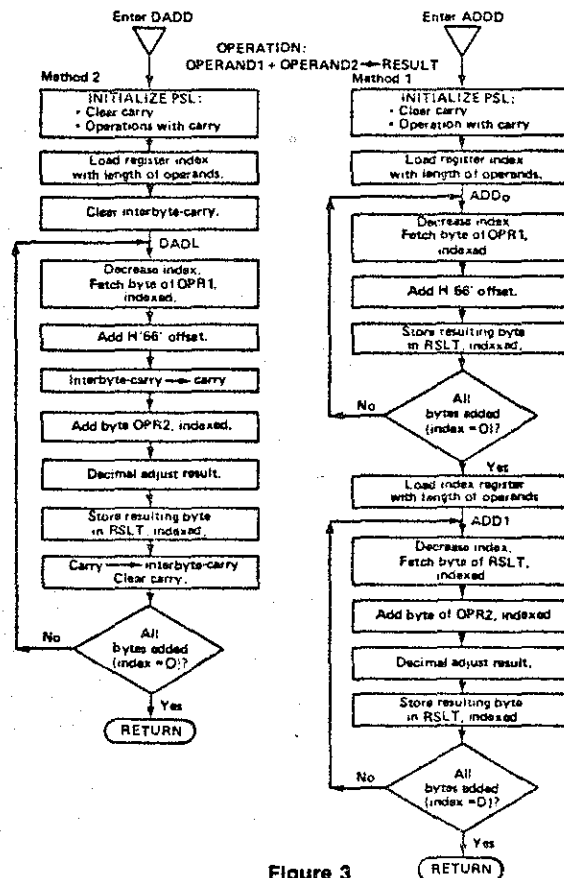


Figure 3

### SIGN-MAGNITUDE NOTATION

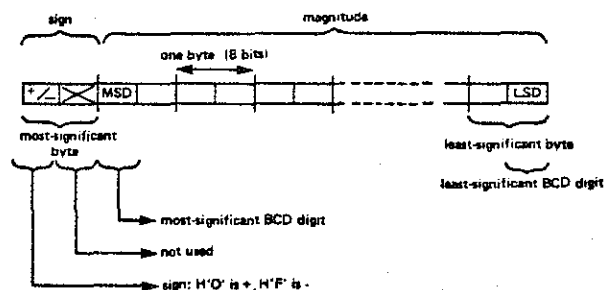


Figure 4



## BCD ADDITION AND SUBTRACTION ROUTINES

TWIN ASSEMBLER VER 1.0

PAGE 0001

LINE ADDR OBJECT E SOURCE

```

0001 * PD760007
0002 *-----*
0003 * DECIMAL ADDITION/SUBTRACTION FOR PACKED-BCD *
0004 *-----*
0005 * OPERATION OPERAND1 +/- OPERAND2 -> RESULT
0006 * OPERAND1 IS IN OPRI, OPRI+1, OPRI+2, ETC.
0007 * OPERAND2 IS IN OPRI2, OPRI2+1, OPRI2+2, ETC.
0008 * RESULT IS IN RSLT, RSLT+1, RSLT+2, ETC.
0009 * OPRI, OPRI2 AND RSLT ARE MOST-SIGNIFICANT BYTES.
0010 * ALL NUMBERS ARE OF EQUAL LENGTH (IN BYTES).
0011 * LENGTH IS DEFINED BY LENG
0012 *
0013 * DEFINITIONS OF SYMBOLS:
0014 *
0015 * EQU 0 PROCESSOR REGISTERS
0016 * EQU 1
0017 * EQU 2
0018 * EQU 3
0019 * EQU 4 PS: 1-WITH 0-WITHOUT CARRY
0020 * EQU 5 CARRY/BORROW
0021 * EQU 6 BRANCH CONDITION UNCONDITIONAL
0022 *
0023 * EQU 7 LENG EQU 5 LENGTH OF OPERANDS/RESULT IN BYTES
0024 *
0025 * EQU 8 ORG H'700' PARAMETERS
0026 *
0027 * EQU 9 OPRI RES LENG OPERAND1
0028 * EQU 10 OPRI2 RES LENG OPERAND2
0029 * EQU 11 RSLT RES LENG RESULT
0030 *
0031 * EQU 12 ORG H'400'
0032 *
0033 *-----*
0034 * ADDITION OF UNSIGNED, SINGLE-BYTE BCD NUMBERS *
0035 *-----*
0036 * OPERATION OPERAND1 + OPERAND2 -> RESULT
0037 *
0038 * EQU 13 ADD LODI, R3, OPRI FETCH FIRST OPERAND
0039 * EQU 14 ADDI, R3, H'66' ADD OFFSET FOR BCD ADD
0040 * EQU 15 ADDI, R3, OPRI2 ADD SECOND OPERAND
0041 * EQU 16 DAI, R3 DECIMAL ADJUST RESULT
0042 * EQU 17 STAI, R3, RSLT STORE RESULT
0043 *
0044 *-----*
0045 * SUBTRACTION OF UNSIGNED, SINGLE-BYTE BCD NUMBERS *
0046 *-----*
0047 * OPERATION OPERAND1 - OPERAND2 -> RESULT
0048 *
0049 * EQU 18 SUBI, R3, OPRI FETCH FIRST OPERAND
0050 * EQU 19 SUBI, R3, OPRI2 SUBTRACT SECOND OPERAND
0051 * EQU 20 DAI, R3 DECIMAL ADJUST RESULT
0052 * EQU 21 STAI, R3, RSLT STORE RESULT
0053 *

```

TWIN ASSEMBLER VER 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```

0054 *-----*
0055 * ADDITION OF UNSIGNED MULTIPLE-BYTE BCD NUMBERS *
0056 *-----*
0057 * OPERATION OPERAND1 + OPERAND2 -> RESULT
0058 *
0059 * EQU 22 ADDI, R3, C CLEAR CARRY
0060 * EQU 23 PPSL, MC ARITHMETIC/ROTATE WITH CARRY
0061 * EQU 24 LODI, R3, LENG LOAD INDEX REGISTER
0062 * EQU 25 LODI, R1, 0 CLEAR INTERBYTE-CARRY
0063 * EQU 26 DADI, R3, OPRI, R3 - FETCH BYTE OF OPERAND1
0064 * EQU 27 ADDI, R3, H'66' ADD OFFSET FOR BCD ADD
0065 * EQU 28 STAI, R3, RSLT, R3 STORE INTERMEDIATE RESULT
0066 * EQU 29 BRNC, R3, 0 BRANCH IF ALL BYTES NOT READY
0067 * EQU 30 LODI, R3, LENG LOAD INDEX REGISTER
0068 * EQU 31 ADDI, R3, OPRI, R3 - FETCH BYTE OF OPERAND2
0069 * EQU 32 ADDI, R3, OPRI2, R3 ADD BYTE OF OPERAND2
0070 * EQU 33 DAI, R3 DECIMAL ADJUST RESULT
0071 * EQU 34 STAI, R3, RSLT, R3 STORE RESULTING BYTE
0072 * EQU 35 BRNC, R3, 0 BRANCH IF NOT READY
0073 *
0074 *-----*
0075 * ADDITION OF UNSIGNED MULTIPLE-BYTE BCD NUMBERS *
0076 *-----*
0077 * OPERATION OPERAND1 + OPERAND2 -> RESULT
0078 *
0079 * EQU 36 ADDI, R3, C CLEAR CARRY
0080 * EQU 37 PPSL, MC ARITHMETIC WITH CARRY
0081 * EQU 38 LODI, R3, LENG LOAD INDEX REGISTER
0082 * EQU 39 ADDI, R3, OPRI, R3 - FETCH BYTE OF OPERAND1
0083 * EQU 40 ADDI, R3, H'66' ADD OFFSET FOR BCD-ADD
0084 * EQU 41 STAI, R3, RSLT, R3 STORE INTERMEDIATE RESULT
0085 * EQU 42 BRNC, R3, 0 BRANCH IF ALL BYTES NOT READY
0086 * EQU 43 LODI, R3, LENG LOAD INDEX REGISTER
0087 * EQU 44 ADDI, R3, OPRI, R3 - FETCH BYTE OF INTERMEDIATE SUM
0088 * EQU 45 ADDI, R3, OPRI2, R3 ADD BYTE OF OPERAND2
0089 * EQU 46 DAI, R3 DECIMAL ADJUST RESULT
0090 * EQU 47 STAI, R3, RSLT, R3 STORE RESULT
0091 * EQU 48 BRNC, R3, 0 BRANCH IF ALL BYTES NOT READY
0092 *
0093 *-----*
0094 * SUBTRACTION OF UNSIGNED MULTIPLE-BYTE BCD NUMBERS *
0095 *-----*
0096 * OPERATION OPERAND1 - OPERAND2 -> RESULT
0097 *
0098 * EQU 49 DSUB, PPSL, MC+C ARITHMETIC WITH CARRY, CLEAR BORROW
0099 * EQU 50 LODI, R3, LENG LOAD INDEX REGISTER
0100 * EQU 51 DSUB, R3, OPRI, R3 - FETCH BYTE OF OPERAND1
0101 * EQU 52 SUBI, R3, OPRI2, R3 SUBTRACT BYTE OF OPERAND2
0102 * EQU 53 DAI, R3 DECIMAL ADJUST RESULT
0103 * EQU 54 STAI, R3, RSLT, R3 STORE RESULTING BYTE
0104 * EQU 55 BRNC, R3, 0 BRANCH IF NOT READY
0105 *
0106 * EQU 56 END 0
0107 *

```

TOTAL ASSEMBLY ERRORS = 0000

Figure 5

## Program Title

DECIMAL ADDITION/SUBTRACTION  
FOR SIGNED INTEGERS (PACKED BCD)

## Function

Addition or subtraction of 2 decimal integers in sign-magnitude notation. Operands and result are of equal length, as defined by LENG.

OPERAND1 +/- OPERAND2 → OPERAND2

## Parameters

### Input:

Length of numbers (in bytes) is defined by LENG.

OPR1, OPR1+1, OPR1+2, etc., contain augend or subtrahend.

OPR2, OPR2+1, OPR2+2, etc., contain addend or minuend.

### Output:

OPR2, OPR2+1, OPR2+2, etc., contain sum or difference.

Overflow is detected.

## OPERATION

Subtraction is performed by changing the sign of the second operand before entering the signed addition routine. Prior to adding or subtracting, the sign of the result must be determined. This requires a comparison of the magnitudes of both operands if they have opposite signs. In this case, the subtrahend and minuend for the operation are also designated by the comparison.

Refer to Figures 6 and 7 for flowchart and program listing.

## FLOWCHART FOR DECIMAL ADDITION/SUBTRACTION FOR SIGNED INTEGERS

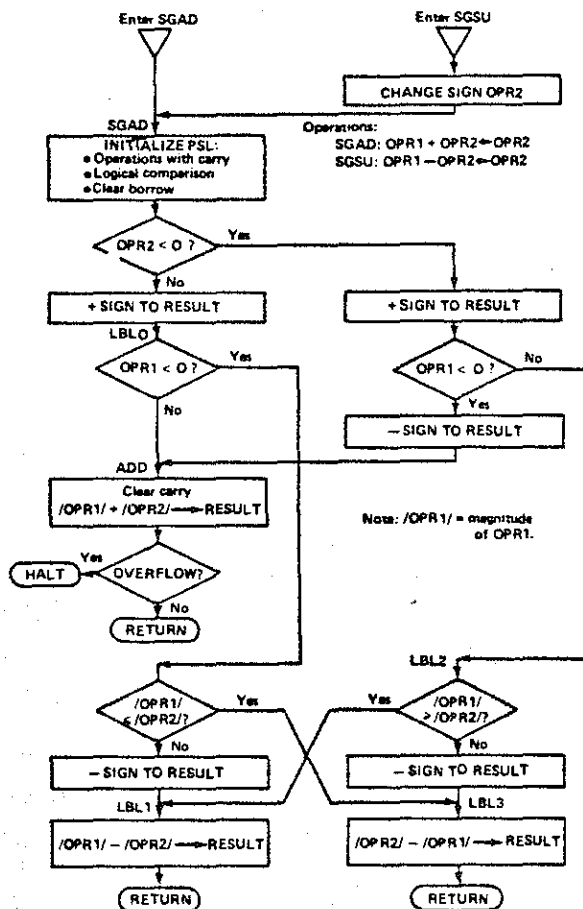


Figure 6

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
| PSU               | F  | II  | SP |    |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |

|                                       |              |
|---------------------------------------|--------------|
| RAM REQUIRED (BYTES):                 | 2 X LENG     |
| ROM REQUIRED (BYTES):                 | 127          |
| MAXIMUM SUBROUTINE<br>NESTING LEVELS: | 1            |
| ASSEMBLER/COMPILER USED:              | TWIN VER 1.0 |

PAGE 6001

LINE ADDR OBJECT E SOURCE

```

0067 051A 7700 SGNB PPSL MC=COM+D OPERATIONS WITH CARRY.
0068 * LOGICAL COMPARE, CLEAR BORROW
0069 051C 20 ED02 R0 CLEAR R0
0070 051D 008705 LODR R1 0P02 FETCH SIGN OF OPERAND0
0071 051E C08705 STRR R0 0P02 CLEAR SIGN OF OPERAND0 (<RESULT)
0072 0523 9023 B0CF R1 L0L0 BRANCH IF 0P02 NOT NEGATIVE
0073 0525 C08700 LODR R0 0P02 FETCH SIGN OF OPERAND1
0074 0529 902C B0CF R1 L0L2 BRANCH IF 0P01 NOT NEGATIVE
0075 052B 04F8 LODI R0 N'F8" FETCH MINUS SIGN
0076 052C C08705 STRR R0 0P02 STORE IN MS-BYTE RESULT
0077 *
0078 052F 7501 ADD CPSL C 0P01 + 0P02 -> 0P02
0079 * CLEAR CARRY
0080 0531 0704 LODI R3 L0E-1 LOAD INDEX REGISTER
0081 0533 0500 LODI R1 0 CLEAR INTERVENE-ARRY
0082 0535 0F6700 ADD0B LODR R0 0P01 R3 FETCH BYTE OF OPERAND0
0083 053B 0466 ADDI R0 N'66" ADD OFFSET
0084 053D 51 R0R1 R1 INTERVENE-CARRY TO CARRY
0085 053B 0F6705 ADD0B R0 0P02 R3 ADD BYTE OF OPERAND2
0086 053E 94 DRR R0 DECIMAL ADJUST RESULT
0087 053F 0F6705 STRR R0 0P02 R3 STORE RESULTING BYTE
0088 0542 D1 R0L R1 CARRY (+INTERVENE-CARRY) TO R0.
0089 * CLEAR CARRY
0090 0543 F070 B0RE R3 ADD0 BRANCH IF NOT READY
0091 0545 0038 B0CF 2 0VFL BRANCH IF OVERFLOW
0092 0547 17 RETC UN RETURN
0093 *

```

TWIN ASSEMBLER VER 1.0

PAGE 0505

| LINE | ADDR   | OBJECT | E      | SOURCE |
|------|--------|--------|--------|--------|
| 1    | 000000 | 000000 | 000000 | 000000 |
| 2    | 000001 | 000001 | 000001 | 000001 |
| 3    | 000002 | 000002 | 000002 | 000002 |
| 4    | 000003 | 000003 | 000003 | 000003 |
| 5    | 000004 | 000004 | 000004 | 000004 |
| 6    | 000005 | 000005 | 000005 | 000005 |
| 7    | 000006 | 000006 | 000006 | 000006 |
| 8    | 000007 | 000007 | 000007 | 000007 |
| 9    | 000008 | 000008 | 000008 | 000008 |
| 10   | 000009 | 000009 | 000009 | 000009 |
| 11   | 00000A | 00000A | 00000A | 00000A |
| 12   | 00000B | 00000B | 00000B | 00000B |
| 13   | 00000C | 00000C | 00000C | 00000C |
| 14   | 00000D | 00000D | 00000D | 00000D |
| 15   | 00000E | 00000E | 00000E | 00000E |
| 16   | 00000F | 00000F | 00000F | 00000F |
| 17   | 000010 | 000010 | 000010 | 000010 |
| 18   | 000011 | 000011 | 000011 | 000011 |
| 19   | 000012 | 000012 | 000012 | 000012 |
| 20   | 000013 | 000013 | 000013 | 000013 |
| 21   | 000014 | 000014 | 000014 | 000014 |
| 22   | 000015 | 000015 | 000015 | 000015 |
| 23   | 000016 | 000016 | 000016 | 000016 |
| 24   | 000017 | 000017 | 000017 | 000017 |
| 25   | 000018 | 000018 | 000018 | 000018 |
| 26   | 000019 | 000019 | 000019 | 000019 |
| 27   | 00001A | 00001A | 00001A | 00001A |
| 28   | 00001B | 00001B | 00001B | 00001B |
| 29   | 00001C | 00001C | 00001C | 00001C |
| 30   | 00001D | 00001D | 00001D | 00001D |
| 31   | 00001E | 00001E | 00001E | 00001E |
| 32   | 00001F | 00001F | 00001F | 00001F |
| 33   | 000020 | 000020 | 000020 | 000020 |
| 34   | 000021 | 000021 | 000021 | 000021 |
| 35   | 000022 | 000022 | 000022 | 000022 |
| 36   | 000023 | 000023 | 000023 | 000023 |
| 37   | 000024 | 000024 | 000024 | 000024 |
| 38   | 000025 | 000025 | 000025 | 000025 |
| 39   | 000026 | 000026 | 000026 | 000026 |
| 40   | 000027 | 000027 | 000027 | 000027 |
| 41   | 000028 | 000028 | 000028 | 000028 |
| 42   | 000029 | 000029 | 000029 | 000029 |
| 43   | 00002A | 00002A | 00002A | 00002A |
| 44   | 00002B | 00002B | 00002B | 00002B |
| 45   | 00002C | 00002C | 00002C | 00002C |
| 46   | 00002D | 00002D | 00002D | 00002D |
| 47   | 00002E | 00002E | 00002E | 00002E |
| 48   | 00002F | 00002F | 00002F | 00002F |
| 49   | 000030 | 000030 | 000030 | 000030 |
| 50   | 000031 | 000031 | 000031 | 000031 |
| 51   | 000032 | 000032 | 000032 | 000032 |
| 52   | 000033 | 000033 | 000033 | 000033 |
| 53   | 000034 | 000034 | 000034 | 000034 |
| 54   | 000035 | 000035 | 000035 | 000035 |
| 55   | 000036 | 000036 | 000036 | 000036 |
| 56   | 000037 | 000037 | 000037 | 000037 |
| 57   | 000038 | 000038 | 000038 | 000038 |
| 58   | 000039 | 000039 | 000039 | 000039 |
| 59   | 00003A | 00003A | 00003A | 00003A |
| 60   | 00003B | 00003B | 00003B | 00003B |
| 61   | 00003C | 00003C | 00003C | 00003C |
| 62   | 00003D | 00003D | 00003D | 00003D |
| 63   | 00003E | 00003E | 00003E | 00003E |
| 64   | 00003F | 00003F | 00003F | 00003F |
| 65   | 000    |        |        |        |

```

0095 0548 0C8780 LBL0 L0DA,R0 OPRI
0096 0548 9662 BCFR,M R00
0097 0540 3F8500 BSTR,UN C012
0098
0099 0530 991E BCFR,GT LBL1
0100 0532 04F0 L0D1,R0 W'F0
0101 0534 C08785 STRA,R0 OPRI
0102
0103
0104 0537 0704 LBL1 L0D1,R3 LENG=1
0105 0539 0F6700 SUI2 L0DA,R0 OPRI,R3
0106 0535 0F6785 SUBA,R0 OPRI,R3
0107 053F 94 DRA,R0
0108 0560 0F6785 STRA,R0 OPRI,R3
0109 0563 F874 B0R0,R3 SUI2
0110 0565 17 RETC,UN
0111
0112 0566 3F8500 LBL2 BSTR,UN C012
0113
0114 0569 966C BCFR,L1 LBL1
0115 0568 04F0 L0D1,R0 W'F0
0116 0560 C08785 STRA,R0 OPRI
0117
0118
0119 0570 0704 LBL3 L0D1,R3 LENG=1
0120 0572 0F6785 SUI2 L0DA,R0 OPRI,R3
0121 0575 0F6700 SUBA,R0 OPRI,R3
0122 0578 94 DRA,R0
0123 0579 0F6785 STRA,R0 OPRI,R3
0124 057C F874 B0R0,R3 SUI2
0125 057E 17 RETC,UN
0126
0127 057F 40 OVAL HALT
0128
0129 0000 END 0

FETCH SIGN OF OPERAND1
BRANCH IF OPRI NOT NEGATIVE
COMPARE OPRI WITH OPRI2
(PHAGNITUDES ONLY)
BRANCH IF OPRI < (OK = TO OPRI2)
FETCH MINUS SIGN
STORE IN MS-BYTE RESULT

OPRI - OPRI2 -> OPRI2
LOAD INDEX REGISTER
FETCH BYTE OF OPERAND1
SUBTRACT BYTE OF OPERAND2
DECIMAL ADJUST RESULT
STORE RESULTING BYTE IN OPRI2
BRANCH IF NOT READY
RETURN

OPRI - OPRI1 -> OPRI2
LOAD INDEX REGISTER
FETCH BYTE OF OPERAND1
SUBTRACT BYTE OF OPERAND2
DECIMAL ADJUST RESULT
STORE RESULTING BYTE
BRANCH IF NOT READY
RETURN

ARITHMETIC OVERFLOW

```

TWIN ASSEMBLER VER 1.0

PAGE 0042

LINE ADDR OBJECT E SOURCE

```
0057 *
0058 *-----*
0059 * SUBTRACTION FOR SIGNED INTEGERS *
0060 *-----*
0061 SCSG, R0, R0, R2 FETON SIGN OF OPERAND2
0062 LOCAL, R0, M'F0' CHANGE SIGN
0063 STRA, R0, R0, R2 RESTORE SIGN OF OPERAND2
0064 *-----*
0065 * ADDITION FOR SIGNED INTEGERS *
0066 *-----*
```

**Figure 7**

## Program Title

DECIMAL MULTIPLICATION FOR  
SIGNED INTEGERS (PACKED BCD)

## FUNCTION

Multiplication of 2 decimal integers in sign-magnitude notation.

Multiplicand, multiplier, and product are of equal length as defined by LENG.

MULTPLICAND X MULTIPLIER → MULTIPLIER

## Parameters

### Input:

Length of numbers (in bytes) is defined by LENG.

MPLC, MPLC+1, MPLC+2, etc., contain multiplicand.

MPLR, MPLR+1, MPLR+2, etc., contain multiplier.

### Output:

MPLR, MPLR+1, MPLR+2, etc., contain product.

Multiplier is destroyed after multiplication.

Overflow is detected.

## OPERATION

Prior to the multiplication algorithm (which is an unsigned operation), the sign of the product is determined. The multiplication gives a double-length result, of which only the least-significant half is retained as the product. If the most-significant half is unequal to zero, an overflow is detected. A "minus-zero" is excluded by means of a test for zero product.

Refer to Figures 8 and 9 for flowchart and program listing.

## FLOWCHART FOR DECIMAL MULTIPLICATION OF SIGNED INTEGERS (PACKED BCD)

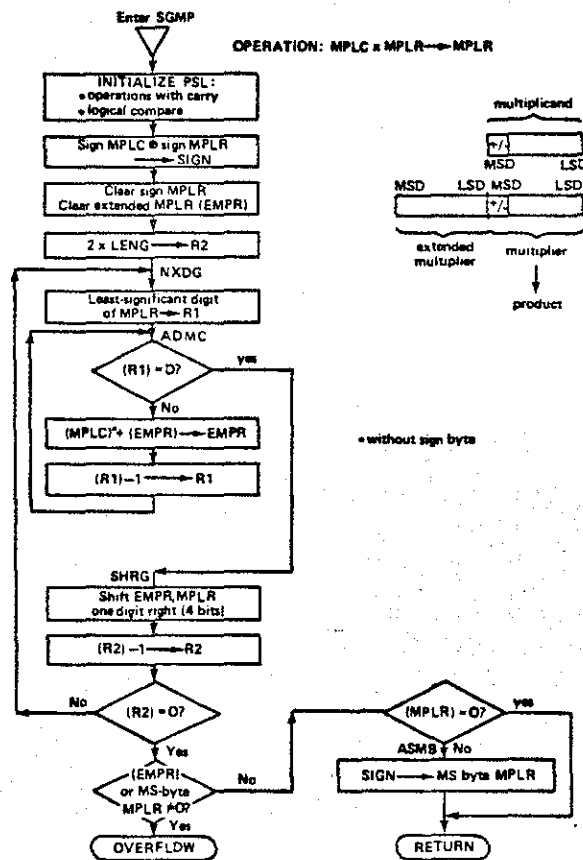


Figure 8

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
| PSU               | F  | II  | SP |    |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |
|                   | X  | X   |    | X  | X   | X   | X   |

|                                    |                       |
|------------------------------------|-----------------------|
| RAM REQUIRED (BYTES):              | $(3 \times LENG) + 1$ |
| ROM REQUIRED (BYTES):              | 111                   |
| MAXIMUM SUBROUTINE NESTING LEVELS: | None                  |
| ASSEMBLER/COMPILER USED:           | TWIN VER 1.0          |

## DECIMAL MULTIPLICATION FOR SIGNED INTEGERS

TWIN ASSEMBLER VER 1.0

PAGE 0001

LINE ADDR OBJECT E SOURCE

```

0001 * P0760005
0002 *-----*
0003 * DECIMAL MULTIPLICATION FOR SIGNED-INTegers *
0004 * NUMBERS ARE IN PACKED BCD, SIGN-MAGNITUDE NOTATION *
0005 *-----*
0006 * OPERATION: MULTIPLICAND X MULTIPLIER -> MULTIPLIER
0007 * MULTIPLICAND IS IN: MPLC, MPLC+1, MPLC+2, ETC.
0008 * MULTIPLIER IS IN: MPLR, MPLR+1, MPLR+2, ETC.
0009 * PRODUCT IS IN: MPLR, MPLR+1, MPLR+2, ETC.
0010 * MULTIPLIER IS DESTROYED AFTER MULTIPLICATION.
0011 * MPLC, MPLR ARE POST-SIGNIFICANT BYTES.
0012 * LENGTH OF NUMBERS (IN BYTES) IS DEFINED BY: LENG
0013 * ALLOWED RANGE: 1 < LENG < 65.
0014 * MS BYTE REPRESENTS SIGN: H'00' FOR +, H'FF' FOR -
0015 *
0016 * DEFINITIONS OF SYMBOLS
0017 *
0018 R0 EQU 0 PROCESSOR-REGISTERS
0019 R1 EQU 1
0020 R2 EQU 2
0021 R3 EQU 3
0022 MC EQU H'00' PSL: 1=WITH, 0=WITHOUT CARRY
0023 CM EQU H'00' LOGIC: 0=WITH COMPARE
0024 C EQU H'01' CARRY/BORROW
0025 Z EQU 0 BRANCH CONDITION: ZERO
0026 UN EQU 3 UNCONDITIONAL
0027 *
0028 * PARAMETERS *
0029 *
0030 LENG EQU 5 LENGTH OF OPERANDS (BYTES)
0031 *
0032 ORG H'700'
0033 *
0034 MPLC RES LENG MULTIPLICAND
0035 EMPL RES LENG EXTENDED MULTIPLIER
0036 MPLR RES LENG MULTIPLIER
0037 * NOTE: EMPL AND MPLR MUST BE IN SUCCESSIVE
0038 * RAM LOCATIONS FOR DOUBLE-LENGTH SHIFT.
0039 SIGN RES 1 TEMPORARY SIGN
0040 *
0041 *-----*
0042 ORG H'500' * MULTIPLICATION PROGRAM *
0043 *-----*
0044 *
0045 S0FF PPSL MC+CM OPERATIONS WITH CARRY, LOGICAL COMPARE
0046 LODR R0 MPLC FETCH SIGN MULTIPLICAND
0047 EDRR R0 MPLR TAKE EX-OR WITH SIGN MULTIPLIER
0048 STRR R0 SIGN SAVE PRODUCT SIGN IN SIGN
0049 EDZ R0 CLEAR R0
0050 LODI R3 LENG+1 LOAD INDEX REGISTER
0051 CLEN STRR R0 EMPL R3 - CLEAR EXTENDED MULTIPLIER AND SIGN OF MULTIPLIER
0052 BRNR R3 CLEN BRANCH IF NOT DONE
0053 LODI R2 LENG+LENG LOAD LOOP COUNTER WITH NUMBER OF DIGITS
0054 MOVG LODR R1 MPLR+LENG-1 FETCH LS-BYTE MULTIPLIER
0055 ANDI R1 H'0F' CLEAR MS-DIGIT
0056 BCTR Z SHRG BRANCH IF LS-DIGIT IS ZERO

```

TWIN ASSEMBLER VER 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```

0058 *
0059 * ADD MULTIPLICAND TO EXTENDED
0060 * MULTIPLIER WITHOUT SIGN
0061 ADVC CPSL C CLEAR CARRY
0062 LODI R3 LENG-1 LOAD INDEX REGISTER
0063 ADDB LODR R0 EMPL R3 FETCH BYTE OF EXTENDED MULTIPLIER
0064 ADDI R0 H'66' ADD OFFSET FOR DECIMAL ADJUST
0065 STRR R0 EMPL R3 RESTORE INTERMEDIATE SUM
0066 BRNR R3 ADDB BRANCH IF ALL BYTES NOT READY
0067 LODI R3 LENG-1 LOAD INDEX REGISTER
0068 ADDB LODR R0 EMPL R3 FETCH BYTE OF INTERMEDIATE SUM
0069 ADDB R0 MPLC R3 ADD BYTE OF MULTIPLICAND
0070 DFR R0 DECIMAL ADJUST RESULT
0071 STRR R0 EMPL R3 STORE RESULTING BYTE
0072 BRNR R3 ADDB BRANCH IF NOT READY
0073 LODR R0 EMPL FETCH MS-BYTE EXTENDED MULTIPLIER
0074 ADDI R0 0 ADD CARRY
0075 STRR R0 EMPL RESTORE MS-BYTE EXTENDED MULTIPLIER
0076 BRNR R3 ADVC BRANCH IF NOT READY WITH DIGIT
0077 *
0078 * SHIFT EMPL AND MPLR ONE DIGIT
0079 * POSITION RIGHT (4 BITS)
0080 SHRG LODI R1 4 LOAD LOOP COUNTER
0081 SHRG CPSL C CLEAR CARRY
0082 LODI R3 -LENG+LENG LOAD INDEX REGISTER
0083 SHRL LODR R0 EMPL-256+LENG+LENG R3 FETCH BYTE OF EXTENDED MULTIPLIER
0084 RRR R0 ROTATE RIGHT WITH CARRY
0085 STRR R0 EMPL-256+LENG+LENG R3 RESTORE BYTE
0086 BRNR R3 SHRG BRANCH IF ALL NOT SHIFTED
0087 BRNR R3 SHRG BRANCH IF 4 BITS NOT SHIFTED
0088 *
0089 BRNR R2 MOVG BRANCH IF ALL DIGITS NOT READY
0090 *
0091 * TEST FOR OVERFLOW: OVERFLOW IF
0092 * (EMPL) OR MS-BYTE MPLR ARE UNEQUAL TO ZERO
0093 LODI R3 LENG+1 LOAD INDEX REGISTER
0094 T0VF LODR R0 EMPL R3 FETCH BYTE OF EXTENDED MPLR
0095 BCFR Z 0VF BRANCH IF NOT ZERO
0096 BRNR R3 T0VF BRANCH IF ALL BYTES NOT TESTED
0097 LODI R3 LENG-1 LOAD INDEX REGISTER
0098 TZER LODR R0 MPLR R3 FETCH BYTE OF PRODUCT
0099 BCFR Z 0ZRB BRANCH IF NOT ZERO
0100 BRNR R3 TZER BRANCH IF ALL BYTES NOT TESTED
0101 RETC UN PRODUCT=0, SIGN REMAINS ZERO
0102 *
0103 R0VB LODR R0 SIGN FETCH PRODUCT SIGN
0104 STRR R0 MPLR STORE IN MS-BYTE MPLR
0105 RETC UN RETURN
0106 *
0107 OVFL HALT ARITHMETIC OVERFLOW
0108 *
0109 END 0

```

TOTAL ASSEMBLY ERRORS = 0000

Figure 9

**Program Title**DECIMAL DIVISION FOR SIGNED  
INTEGERS (PACKED BCD)**Function**Division of 2 decimal integers in sign-  
magnitude notation.Dividend, divisor, quotient, and remainder  
are of equal length as defined by LENG.DIVIDEND: DIVISOR → DIVIDEND,  
REMAINDER**Parameters****Input:**Length of numbers (in bytes) is defined by  
LENG.DVDN, DVDN+1, DVDN+2, etc., contain div-  
idend.DVSR, DVSR+1, DVSR+2, etc., contain divi-  
sor.**Output:**DVDN, DVDN+1, DVDN+2, etc., contain  
quotient.RMDR, RMDR+1, RMDR+2, etc., contain  
remainder.

Dividend is destroyed after division.

Overflow is detected.

**OPERATION:**

Prior to the division, which in itself is an unsigned operation, the signs of the remainder and quotient are determined. Because the division can result in a zero quotient and/or remainder, the possibility of a "minus zero" is excluded by tests. If the divisor is zero, overflow is detected.

Refer to Figures 10 and 11 for flowchart and program listing.

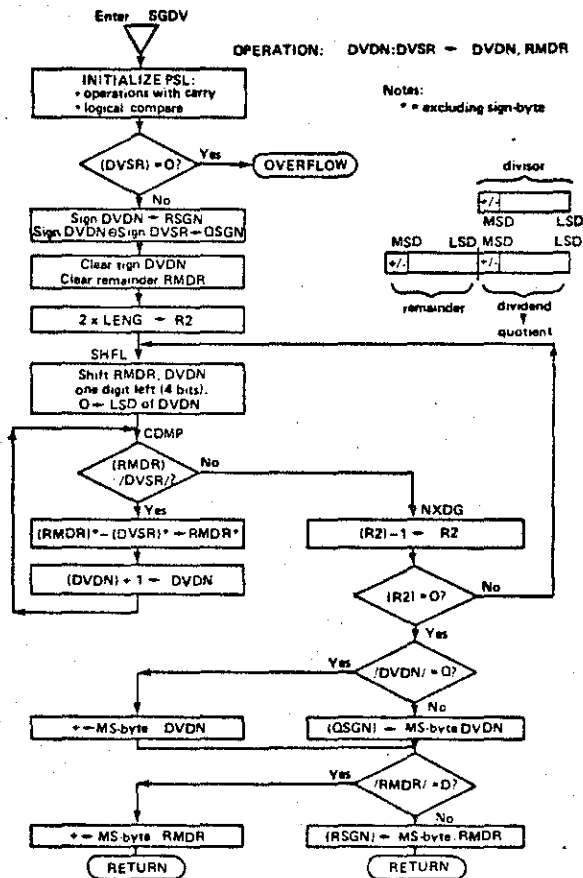
**FLOWCHART FOR DECIMAL DIVISION OF SIGNED INTEGERS**

Figure 10

| HARDWARE AFFECTED |         |          |         |         |          |          |         | RAM REQUIRED (BYTES): <u>(3 x LENG) + 4</u> | ROM REQUIRED (BYTES): <u>144</u> |                                                |
|-------------------|---------|----------|---------|---------|----------|----------|---------|---------------------------------------------|----------------------------------|------------------------------------------------|
| REGISTERS         | R0<br>X | R1<br>X  | R2<br>X | R3<br>X | R1'<br>  | R2'<br>  | R3'<br> |                                             |                                  |                                                |
| PSU               | F       | II       | SP      |         |          |          |         |                                             |                                  | MAXIMUM SUBROUTINE<br>NESTING LEVELS: <u>1</u> |
| PSL               | CC<br>X | IDC<br>X | RS      | WC<br>X | OVF<br>X | COM<br>X | C<br>X  |                                             |                                  |                                                |
|                   |         |          |         |         |          |          |         |                                             |                                  | ASSEMBLER/COMPILER USED: <u>TWIN VER 1.0</u>   |

## DECIMAL DIVISION FOR SIGNED INTEGERS

TWIN ASSEMBLER VER 1.0

PAGE 0001

LINE ADDR OBJECT E SOURCE

```

0001 * P0700004
0002 *-----*
0003 * DECIMAL DIVISION FOR SIGNED INTEGERS
0004 * NUMBERS ARE IN PACKED BCD. SIGN-MAGNITUDE NOTATION *
0005 *-----*
0006 * OPERATION
0007 * DIVIDEND DIVISOR -> DIVIDEND, REMAINDER
0008 * DIVIDEND IS IN DVMX,DVMX+1,DVMX+2, ETC
0009 * DIVISOR IS IN DVSR,DVSR+1,DVSR+2, ETC
0010 * QUOTIENT IS IN DMX,DVMX+1,DVMX+2, ETC
0011 * REMAINDER IS IN RMX,RMX+1,RMX+2, ETC
0012 * DIVIDEND IS DESTROYED AFTER DIVISION
0013 * DVMX,DVSR AND RMX ARE NEXT-SIGNIFICANT BYTES
0014 * LENGTH OF NUMBERS (IN BYTES) IS DEFINED BY LENG
0015 * ALLOWED RANGE: 1 < LENG < 65
0016 * MS-BYTE HOLDS SIGN INFORMATION: H'00' FOR +, H'F0' FOR -
0017 *
0018 * DEFINITIONS OF SYMBOLS
0019 *
0020 0000 R0 EQU 0 PROCESSOR REGISTERS
0021 0001 R1 EQU 1
0022 0002 R2 EQU 2
0023 0003 R3 EQU 3
0024 0004 MC EQU H'00' PSL: 1=WITH 0=WITHOUT CARRY
0025 0005 CM EQU H'02' 1=LOGIC 0=ARITH COMPARE
0026 0006 C EQU H'01' CARRY/BORROW
0027 0007 Z EQU 0 BRANCH COND: ZERO
0028 0008 P EQU 1 POSITIVE
0029 0009 N EQU 2 NEGATIVE
0030 0010 EQ EQU 0 EQUAL
0031 0011 LT EQU 2 LESS THAN
0032 0012 UN EQU 3 UNCONDITIONAL
0033 *
0034 * PARAMETERS *
0035 *
0036 0005 LENG EQU 5 LENGTH OF OPERANDS (IN BYTES)
0037 *
0038 0006 ORG H'700'
0039 *
0040 0700 RMX RES LENG REMAINDER
0041 0701 DVMX RES LENG DIVIDEND
0042 * NOTE: RMX AND DVMX MUST BE IN SUCCESSIVE
0043 * RAM LOCATIONS, BECAUSE OF DOUBLE-LENGTH SHIFT.
0044 0702 DVSR RES LENG DIVISOR
0045 0703 TEMP RES 2 TEMPORARY STORAGE FOR ADDRESS
0046 0704 QSGN RES 1 QUOTIENT SIGN
0047 0705 RSGN RES 1 REMAINDER SIGN
0048 *

```

TWIN ASSEMBLER VER 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```

0050 0711 ORG H'300'
0051 *
0052 * SUBROUTINE TO TEST OPERAND FOR ZERO *
0053 *-----*
0054 * OPERAND ADDRESS MUST BE IN R0,R1 (HIGH,LOW ADDR.)
0055 * ALL BYTES, EXCEPT MS-BYTE (H'F0) ARE TESTED
0056 * CONDITION CODE BECOMES 00 IF OPERAND WAS ZERO.
0057 *
0058 0500 0000 0000 0000 0000 0000 0000 0000
0059 0501 0000 0000 0000 0000 0000 0000 0000
0060 0502 0000 0000 0000 0000 0000 0000 0000
0061 0503 0000 0000 0000 0000 0000 0000 0000
0062 0504 0000 0000 0000 0000 0000 0000 0000
0063 0505 0000 0000 0000 0000 0000 0000 0000
0064 0506 0000 0000 0000 0000 0000 0000 0000
0065 0507 0000 0000 0000 0000 0000 0000 0000
0066 0508 0000 0000 0000 0000 0000 0000 0000
0067 *
0068 *
0069 * DIVISION PROGRAM *
0070 *
0071 *
0072 0510 7700 SMOV PPSL MC+CM OPERATIONS WITH CARRY,
0073 * LOGICAL COMPARISON
0074 0512 0407 LDDI R0 DVSR HIGH-ADDRESS DIVISOR TO R0
0075 0514 0500 LDDI R1 DVSR LOW-ADDRESS DIVISOR TO R1

```

```

0076 0516 3F0500 BSTR UN TZER TEST DIVISOR FOR ZERO
0077 0519 100505 BCTR Z OVFL BRANCH IF ZERO
0078 051C 0C0705 LODA R0 DVMX FETCH SIGN DIVIDEND
0079 051F 0C0712 STRA R0 RSGN SAVE IN REMAINDER SIGN
0080 0522 2C0700 EDRA R0 DVSR TAKE EX-OR WITH DVSR SIGN
0081 0525 0C0711 STRA R0 QSGN SAVE IN QUOTIENT SIGN
0082 0528 20 EDI2 R0 CLEAR R0
0083 0529 0706 LDDI R3 LENG+1 LOAD INDEX REGISTER
0084 052B 0F4700 CLM STRA R0 RMX,R3 - CLEAR REMAINDER AND SIGN DVMX
0085 052E 5B70 BARR R3 CLM BRANCH IF NOT DONE
0086 *
0087 0530 060A LDDI R2 LENG+LENG NUMBER OF DIGITS TO LOOP COUNTER
0088 *
0089 *
0090 * SHIFT RMX/DVMX 4 BITS LEFT
0091 0532 0504 SHFL LDDI R1 4 LOAD BIT COUNTER
0092 0534 7501 SHFB CPSL C INSERTING ZEROS IN LS-BITS
0093 0536 070A LDDI R3 LENG+LENG LOAD INDEX REGISTER
0094 0538 0F4700 SHF1 LODA R0 RMX,R3 - FETCH BYTE OF RMX/DVMX
0095 053B 00 RAL R0 ROTATE LEFT WITH CARRY
0096 053C 0F6700 STRA R0 RMX,R3 RESTORE SHIFTED BYTE
0097 053F 5B77 BARR R3 SHF1 BRANCH IF ALL NOT SHIFTED
0098 0541 7971 BARR R1 SHF0 BRANCH IF 4 BITS NOT SHIFTED

```

TWIN ASSEMBLER VER 1.0

PAGE 0003

LINE ADDR OBJECT E SOURCE

```

0100 *
0101 * COMPARE RMX AND DVSR TO TEST
0102 0543 0506 COMP LDDI R1 0 IF SUBTRACTION IS POSSIBLE
0103 * CLEAR R1. MS-BIT OF R1 BECOMES
0104 * 1 FOR RMX < DVSR
0105 0545 0704 LDDI R3 LENG-1 LOAD INDEX REGISTER
0106 0547 0F6700 COMB LODA R0 RMX,R3 FETCH BYTE OF REMAINDER
0107 0549 0F670A COMB LODA R0 DVSR,R3 COMPARE WITH BYTE OF DIVISOR
0108 054F 11 SPCL PSL TO R0
0109 0550 C1 STR2 R1 SAVE PSL IN R1
0110 0551 FB74 COMB BARR R3 COMB BRANCH IF ALL BYTES NOT TESTED
0111 0553 01 LOOZ R1 FETCH STATUS OF COMPARISON
0112 0554 1A1A BCTR LT ADDG BRANCH IF RMX < DVSR
0113 *
0114 *
0115 * SUBTRACT DIVISOR FROM
0116 * REMAINDER WITHOUT MS-BYTES.
0117 0556 7701 PPSL C CLEAR BORROW
0118 0558 0704 LDDI R3 LENG-1 LOAD INDEX REGISTER
0119 055A 0F6700 SUD LODA R0 RMX,R3 FETCH BYTE OF REMAINDER
0120 055C 0F670A SUD LODA R0 DVSR,R3 SUBTRACT BYTE OF DIVISOR
0121 055E 34 DMR R0 DECIMAL ADJUST RESULT
0122 0564 FB74 STRA R0 RMX,R3 RESTORE IN REMAINDER
0123 *
0124 0566 0C0700 LDDI R0 DVMX+LENG-1 FETCH LS-BYTE QUOTIENT
0125 0569 1000 BIRR R0 #+2 INCREASE R0
0126 056B 0C0700 STRA R0 DVMX+LENG-1 RESTORE INCREMENTED QUOTIENT
0127 056E 1B53 BCTR UN COMP BRANCH FOR NEXT COMPARISON
0128 *
0129 0570 F440 ADDG BARR R2 SHFL BRANCH IF DIVISION NOT READY
0130 0572 0E0711 LODA R2 QSGN FETCH SIGN QUOTIENT
0131 0575 0407 LDDI R0 QDVMX HIGH-ADDRESS QUOTIENT TO R0
0132 0577 0505 LDDI R1 QDVMX LOW-ADDRESS QUOTIENT TO R1
0133 0579 3F0500 BSTR UN TZER TEST IF QUOTIENT IS ZERO
0134 057C 3005 BCTR Z STOS BRANCH IF NOT ZERO
0135 057E 0600 LDDI R2 0 CLEAR R2
0136 0580 0C0705 STOS STRA R2 DVMX STORE SIGN IN MS-BYTE DVMX
0137 0582 0E0712 LODA R2 RSGN FETCH REMAINDER SIGN
0138 0584 0407 LDDI R0 QRMX HIGH-ADDRESS REMAINDER TO R0
0139 0586 0506 LODI R1 QRMX LOW-ADDRESS REMAINDER TO R1
0140 0588 3F0500 BSTR UN TZER TEST IF REMAINDER IS ZERO
0141 058B 0600 BCTR Z STOS BRANCH IF NOT ZERO
0142 058E 0600 LDDI R2 0 CLEAR R2
0143 0591 0C0700 STRS STRA R2 RMX STORE SIGN IN MS-BYTE RMX
0144 0594 17 RETC UN RETURN
0145 *
0146 0595 40 OVFL HALT OVERFLOW LOCATION
0147 *
0148 0600 END 0

```

TOTAL ASSEMBLY ERRORS = 0000

Figure 11

## ROUTINES FOR SIGNED FIXED-POINT ARITHMETIC

As illustrated in Figure 12, the numbers used in these arithmetic routines are in sign-magnitude notation with decimal point indication. The latter gives the number of decimals, and has a minimum of zero and a maximum limit of 15 or the number of digits, whichever is smaller.

The length of the numbers is defined by the number of bytes (including the sign byte) they require. This parameter can be modified by changing the definition of LENG in the source program. Note that for clarity, each routine is written in a "stand-alone" form. If more than one routine is required in a program, considerable savings in the program space required can be realized by breaking out common operations as sub-routines.

### Program Title

ALIGNMENT SUBROUTINE FOR FIXED-POINT NUMBERS

### Function

Aligns a fixed-point number to the decimal point position indicated by the contents of register DPNT. Performs rounding as specified.

### Parameters

#### Input:

RD contains the high address of the operand.

R1 contains the low address of the operand.

DPNT contains the required decimal point.

ROUN contains the rounding constant: (ROUN) = H'00' for no rounding; (ROUN) = H'05' for 5/4 rounding; and (ROUN) = H'09' for round-up.

Prior to entry, WC in PSL must be 1.

Length of operand (in bytes) is defined by LENG.

#### Output:

Aligned operand; rounded if specified.

Alignment overflow is detected.

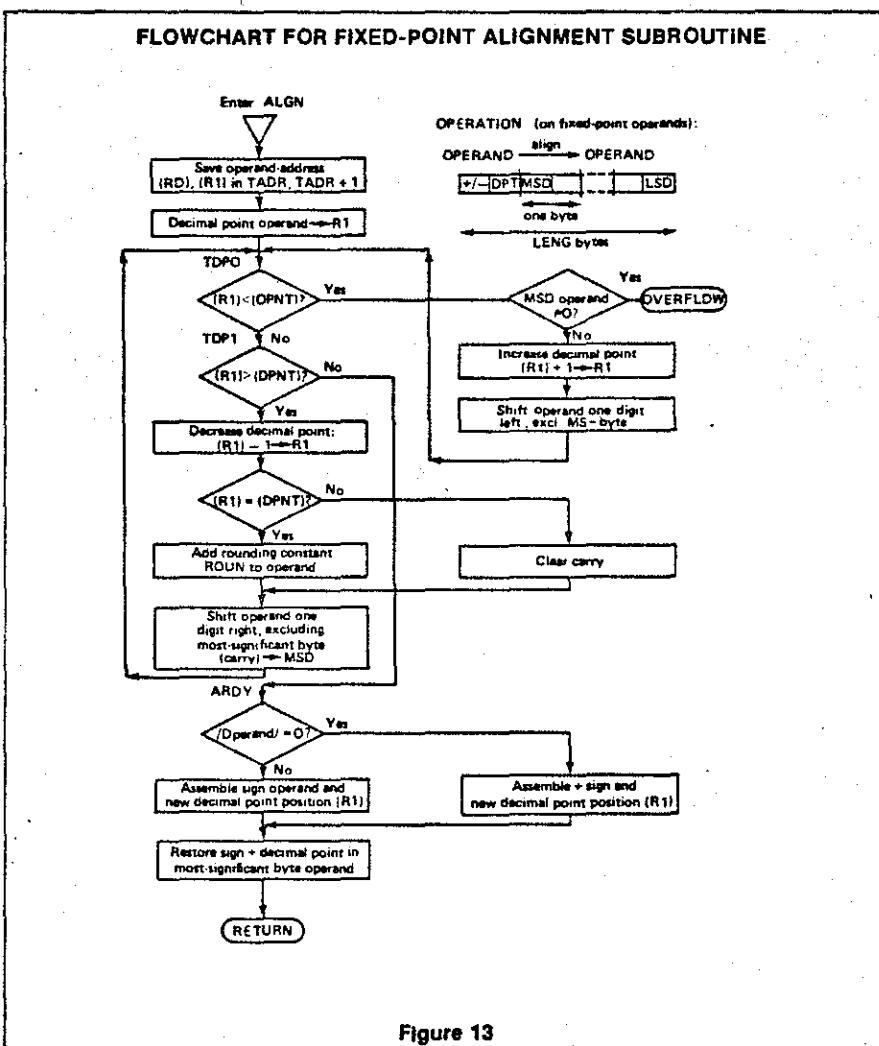
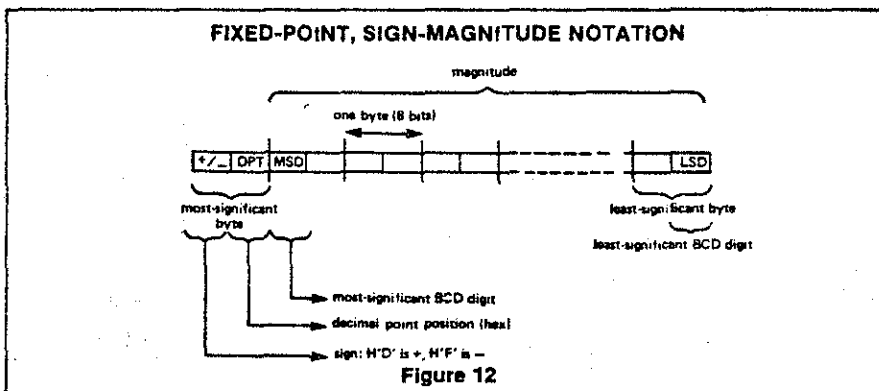
### Operation:

The results of a fixed-point operation must be aligned to the required number of decimals. By means of this aligning routine, the numbers are shifted left or right, if necessary, until the appropriate decimal point position is obtained. This position must have previously been stored in a register designated DPNT. During left alignment, overflow can occur if a non-zero digit drops out of the most-significant digit position.

During aligning it is also possible to perform rounding of the operand. This is done by adding a rounding digit to the most-significant digit of the decimals which are truncated by the right alignment. This rounding digit must have previously been stored in register ROUN and gives the possibilities listed above. Since rounding

can only be performed during right alignment, the required decimal point position must be less than 15 if rounding is desired. If the aligned result is minus zero, the sign is changed.

Refer to Figures 13 and 14 for flowchart and program listing.





| HARDWARE AFFECTED |    |     |    |    |     |     |     | RAM REQUIRED (BYTES):              |  | 4            |
|-------------------|----|-----|----|----|-----|-----|-----|------------------------------------|--|--------------|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' | ROM REQUIRED (BYTES):              |  | 120          |
| PSU               | F  | II  | SP |    |     |     |     | MAXIMUM SUBROUTINE NESTING LEVELS: |  | None         |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   | ASSEMBLER/COMPILER USED:           |  | TWIN VER 1.0 |
|                   | X  | X   |    |    | X   |     | X   |                                    |  |              |

## FIXED-POINT ALIGNMENT SUBROUTINE

TWIN ASSEMBLER VER 1.0

PAGE 0001

LINE ADDR OBJECT E SOURCE

```

0001 * P0700000
0002 *-----*
0003 * FIXED POINT ALIGNMENT SUBROUTINE *
0004 *-----*
0005 *
0006 * DEFINITIONS OF SYMBOLS
0007 *
0008 R0 EQU 0 PROCESSOR REGISTERS
0009 R1 EQU 1
0010 R2 EQU 2
0011 R3 EQU 3
0012 MC EQU 0'00' PSL 1-WITH 0-WITHOUT CARRY
0013 C EQU 0'01' CARRY/BORROW
0014 Z EQU 0 BRANCH COND : ZERO
0015 EQ EQU 0 EQUAL
0016 GT EQU 1 GREATER THAN
0017 LT EQU 2 LESS THAN
0018 UN EQU 3 UNCONDITIONAL
0019 *
0020 * PARAMETERS *
0021 *
0022 LENG EQU 3 LENGTH OF OPERAND (BYTES)
0023 *
0024 ORG M'400'
0025 *
0026 DPMT RES 1 REQUIRED DECIMAL POINT (0 THROUGH 9)
0027 ROUN RES 1 ROUNDING CONSTANT (0.5 OR 9)
0028 TADR RES 2 TEMPORARY STORAGE FOR ADDRESS
0029 *
0030 ORG M'450' START OF SUBROUTINE
0031 *
0032 * OPERAND IS ALIGNED TO DECIMAL POINT POSITION RS
0033 * INDICATED BY REGISTER DPMT
0034 * ROUNDING IS PERFORMED UNDER FOLLOWING CONDITIONS:
0035 * (ROUND) CONTAINS M'00' FOR NO ROUNDING
0036 * (ROUND) CONTAINS M'05' FOR 5/4 ROUNDING
0037 * (ROUND) CONTAINS M'09' FOR ROUND-UP
0038 * (DPMT) MUST BE < 15 IF ROUNDING IS REQUIRED
0039 * ALIGNMENT-OVERFLOW IS DETECTED
0040 * PRIOR TO ENTRY MC IN PSL MUST BE L
0041 *
0042 * R0 CONTAINS HIGH-ADDR OF OPERAND
0043 * R1 CONTAINS LOW- ADDR OF OPERAND
0044 * DPMT CONTAINS DECIMAL POINT
0045 * ROUN CONTAINS ROUNDING CONSTANT
0046 *
0047 ALGN STRL R0 TADR SAVE M1-ADDRESS OF OPERAND
0048 STRL R1 TADR+1 SAVE L0-ADDRESS OF OPERAND
0049 LODL R1 TADR FETCH M5-BYTE OF OPERAND
0050 ANDL R1 M'0F' REMOVE SIGN, KEEP DECIMAL POINT
0051 TOP0 LODL R2 4 LOAD LOOP COUNTER
0052 COMPL R1 DPMT COMPARE ACTUAL DECIMAL POINT WITH
0053 REQUIRED DECIMAL POINT
0054 BCFR LT TOP1 BRANCH IF EQUAL OR TOO BIG
0055 EDORZ R0 CLEAR R0
0056 LODL R0 TADR, R0, + FETCH M5-DIGITS OF OPERAND
0057 ANDL R0 M'FF' CLEAR LS-DIGIT (TEST M50 = 0)

```

TWIN ASSEMBLER VER 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```

0057 BCFR Z 0'0F0 BRANCH IF ALIGNMENT OVERFLOW
0058 BTRR R1 0+2 INCREASE DECIMAL POINT
0059 *
0060 *
0061 SHLB CPSL C CLEAR CARRY
0062 LODL R3 LENG-1 LOAD INDEX REG
0063 SHL1 LODL R0 TADR, R3 FETCH BYTE OF OPERAND
0064 RRL R0 ROTATE LEFT WITH CARRY
0065 STRL R0 TADR, R3 RESTORE
0066 BBR0 R3 SHL1 BRANCH IF ALL NOT SHIFTED
0067 BBR0 R2 SHL0 BRANCH IF 4 BITS NOT SHIFTED
0068 BCTR UN TOP0 BRANCH FOR NEXT TEST
0069 *
0070 TOP1 BCFR GT AR0Y BRANCH IF DECIMAL POINT IS CORRECT
0071 BBR0 R1 0+2 DECREASE DECIMAL POINT
0072 COMPL R1 DPMT TEST IF LS-DIGIT IS ROUNDING DIGIT
0073 BCFR EQ SH00 BRANCH IF NOT
0074 *
0075 CPSL C CLEAR CARRY
0076 LODL R3 LENG-1 LOAD INDEX REGISTER
0077 RRR0 LODL R0 TADR, R3 FETCH BYTE OF OPERAND
0078 ADDL R0 M'66' ADD OFFSET FOR BCD ADD
0079 COMPL R3 LENG-1 COMPL R3 LENG-1
0080 BCFR EQ RND1 BRANCH IF NOT LS-BYTE
0081 RRR0 R0 ROUN ADD ROUNDING CONSTANT
0082 DRR1 R0 DECIMAL ADJUST RESULT
0083 STRL R0 TADR, R3 RESTORE RESULT
0084 BBR0 R3 RND0 BRANCH IF ALL BYTES NOT READY
0085 BCTR UN SH01 BRANCH TO RIGHT-SHIFT OPERAND
0086 *
0087 *
0088 SHRB CPSL C CLEAR CARRY
0089 SHRL LODL R3 0 CLEAR INDEX
0090 SHR2 LODL R0 TADR, R3 + FETCH BYTE OF OPERAND
0091 RRR0 R0 ROTATE RIGHT WITH CARRY
0092 STRL R0 TADR, R3 RESTORE BYTE
0093 COMPL R3 LENG-1 COMPL R3 LENG-1
0094 BCFR EQ SHR2 BRANCH IF ALL NOT SHIFTED
0095 BBR0 R2 SHR0 BRANCH IF 4 BITS NOT SHIFTED
0096 BCTR UN TOP0 BRANCH FOR NEXT TEST
0097 *
0098 AR0Y LODL R2 TADR FETCH M5-BYTE OF OPERAND
0099 ANDL R2 M'FF' REMOVE DECIMAL POINT, KEEP SIGN
0100 LODL R3 LENG-1 LOAD INDEX REGISTER FOR ZERO TEST
0101 TZR0 LODL R0 TADR, R3 FETCH BYTE OF ALIGNED OPERAND
0102 BCFR Z ACER BRANCH IF NON-ZERO
0103 BBR0 R3 TZR0 BRANCH IF ALL BYTES NOT READY
0104 STRZ R2 ZERO RESULT, CLEAR SIGN
0105 NZER LODZ R2 FETCH SIGN
0106 ORCZ R1 ASSEMBLE SIGN AND DECIMAL POINT
0107 STRL R0 TADR STORE IN M5-BYTE OF OPERAND
0108 RETC UN RETURN
0109 *
0110 OVF0 MFLT ALIGNMENT OVERFLOW
0111 *
0112 END 0

```

TOTAL ASSEMBLY ERRORS = 0000

Figure 14

**Program Title**FIXED-POINT ADDITION/SUBTRACTION  
OF SIGNED, PACKED BCD NUMBERS**Function**

Addition/subtraction of 2 decimal fixed-point numbers.

Operands and result are of equal length as defined by LENG.

OPERAND1 +/- OPERAND2 →  
OPERAND2**Parameters****Input:**

Length of numbers (in bytes) defined by LENG.

OPR1, OPR1+1, OPR1+2, etc. contain augend or subtrahend.

OPR2, OPR2+1, OPR2+2, etc., contain addend or minuend.

In the alignment subroutine, the decimal-point position is in OPNT and the rounding constant is in ROUN.

**Output:**

OPR2, OPR2+1, OPR2+2, etc., contain sum or difference.

Result and operand1 are aligned (and rounded if specified).

Overflow is detected.

**Special Requirements**

Software: Fixed-point alignment subroutine ALGN.

**Operation**

Subtraction is performed by changing the sign of the second operand before entering the (signed) addition routine. Prior to the addition/subtraction of the magnitudes of the operands, both operands are aligned (and rounded if programmed), the sign of the result is determined and, in the event the operands have opposite signs, the subtrahend and minuend are designated.

Refer to Figures 15 and 16 for flowchart and program listing.

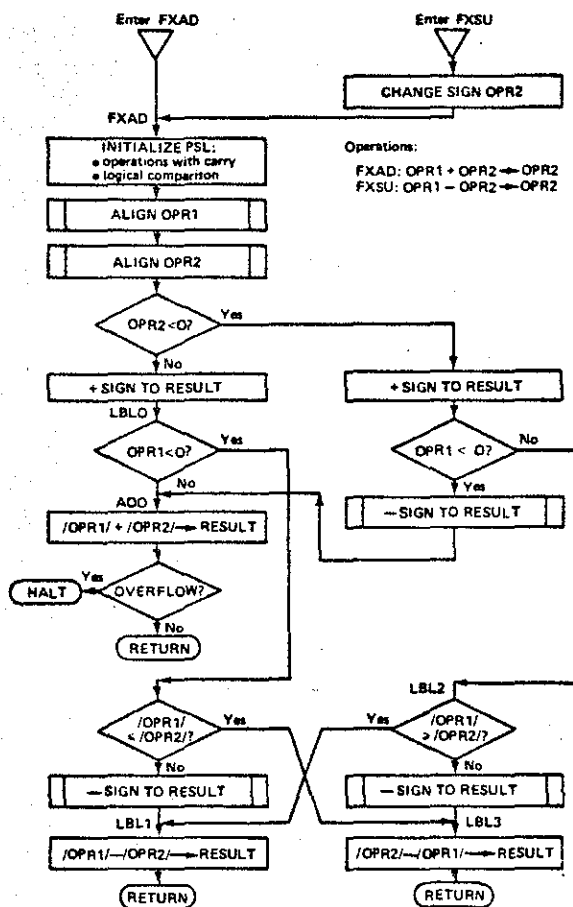
**FLOWCHART FOR ADDITION/SUBTRACTION OF FIXED-POINT NUMBERS**

Figure 15

| HARDWARE AFFECTED |         |          |         |         |          |          |        | RAM REQUIRED (BYTES): <u>2 x LENG</u>        | ROM REQUIRED (BYTES): <u>151</u> |
|-------------------|---------|----------|---------|---------|----------|----------|--------|----------------------------------------------|----------------------------------|
| REGISTERS         | R0<br>X | R1<br>X  | R2<br>X | R3<br>X | R1'      | R2'      | R3'    |                                              |                                  |
| PSU               | F       | II       | SP      |         |          |          |        | ASSEMBLER/COMPILER USED: <u>TWIN VER 1.0</u> |                                  |
| PSL               | CC<br>X | IDC<br>X | RS      | WC<br>X | OVF<br>X | COM<br>X | C<br>X |                                              |                                  |

# FIXED-POINT DECIMAL ADDITION/SUBTRACTION FOR SIGNED, PACKED BCD NUMBERS

TWIN ASSEMBLER VER 1.0

PAGE 0001

LINE ADDR OBJECT E SOURCE

```

0001 * P0760002
0002 * =====
0003 * FIXED-POINT DECIMAL ADDITION/SUBTRACTION *
0004 * FOR SIGNED, PACKED BCD NUMBERS *
0005 * =====
0006 * OPERATION OPERAND1 +/- OPERAND2 -> OPERAND2
0007 * OPERAND1 IS IN OPRL,OPR1+1,OPR1+2, ETC.
0008 * OPERAND2 IS IN OPR2,OPR2+1,OPR2+2, ETC.
0009 * SUM/DIFFERENCE IS IN OPR2,OPR2+1,OPR2+2, ETC.
0010 * OPERAND2 IS DESTROYED AFTER ADD/SUBTRACT
0011 * OPR1,OPR2 ARE MOST-SIGNIFICANT BYTES
0012 * LENGTH OF NUMBERS (IN BYTES) IS DEFINED BY LENG
0013 * ALLOWED RANGE 1 < LENG < 255
0014 * NUMBERS ARE IN SIGN-MAGNITUDE NOTATION
0015 * MS-BYTE HOLDS SIGN AND DECIMAL POINT INFORMATION:
0016 * SIGN IS IN MS 4 BITS 'M' IS +, 'M' IS -
0017 * DECIMAL POINT IS IN LS 4 BITS, BINARY CODED,
0018 * RANGE (0 THRU 15) EQUALS NUMBER OF DECIMALS
0019 *
0020 * DEFINITIONS OF SYMBOLS:
0021 *
0022 * R0 EQU 0 PROCESSOR REGISTERS
0023 * R1 EQU 1
0024 * R2 EQU 2
0025 * R3 EQU 3
0026 * MC EQU 'M' PSL 1-WITH 0-WITHOUT CARRY
0027 * CM EQU 'M' LOGIC 0-ARITH COMPARE
0028 * C EQU 'M' CARRY/BORROW
0029 * Z EQU 0 ZERO
0030 * N EQU 1 NEGATIVE
0031 * EQ EQU 0 EQUAL
0032 * GT EQU 1 GREATER THAN
0033 * LT EQU 2 LESS THAN
0034 * UN EQU 3 UNCONDITIONAL
0035 *
0036 * PARAMETERS *
0037 *
0038 * ALGN EQU 'M' ADDRESS OF ALIGNMENT SUBROUTINE
0039 * LENG EQU 5 LENGTH OF OPERANDS (IN BYTES)
0040 *
0041 * ORG 'M'
0042 *
0043 * OPR1 RES LENG OPERAND1
0044 * OPR2 RES LENG OPERAND2/RESULT
0045 *

```

TWIN ASSEMBLER VER 1.0

PAGE 0008

LINE ADDR OBJECT E SOURCE

```

0047 * ORG 'M'
0048 *
0049 * =====
0050 * SUBROUTINE TO COMPARE OPERAND1 WITH OPERAND2 (UPDATE CC) *
0051 * =====
0052 * 0500 0500
0053 * C012 LOD1,R1,0 CLEAR R1, MS-BITS ARE USED
0054 * LOD1,R3,LENG-1 TO SAVE CC INFORMATION
0055 * COMB LOD1,R0,OPR1,R3 LOAD INDEX REGISTER
0056 * COMB LOD1,R0,OPR2,R3 FETCH BYTE OF OPERAND1
0057 * BCTR,ED,COMB COMPARE WITH BYTE OF OPERAND2
0058 * BCTR,ED,COMB BRANCH IF EQUAL
0059 * PSL,R1 PSL TO R0
0060 * STR2,R1 SAVE PSL IN R1
0061 * COMB B000,R3,COMB BRANCH IF ALL BYTES NOT TESTED
0062 * LOD2,R1 UPDATE CC WITH STATUS COMPARE
0063 * RETC,UN RETURN
0064 *
0065 * =====
0066 * SUBROUTINE TO SET SIGN OF RESULT TO NEGATIVE *
0067 * =====
0068 * 0512 0512 0500 0500
0069 * 0515 0515 0500 0500
0070 * 0517 0517 0500 0500
0071 * 051A 051A 0500 0500
0072 *
0073 *
0074 *
0075 *
0076 *
0077 *
0078 *
0079 *
0080 *
0081 *
0082 *
0083 *
0084 *
0085 *
0086 *
0087 *
0088 *
0089 *
0090 *
0091 *
0092 *
0093 *
0094 *
0095 *
0096 *
0097 *
0098 *
0099 *
0100 *
0101 *
0102 *
0103 *
0104 *
0105 *
0106 *
0107 *
0108 *
0109 *
0110 *
0111 *
0112 *
0113 *
0114 *
0115 *
0116 *
0117 *
0118 *
0119 *
0120 *
0121 *
0122 *
0123 *
0124 *
0125 *
0126 *
0127 *
0128 *
0129 *
0130 *
0131 *
0132 *
0133 *
0134 *
0135 *
0136 *
0137 *
0138 *
0139 *
0140 *
0141 *
0142 *
0143 *
0144 *
0145 *
0146 *
0147 *
0148 *
0149 *
0150 *
0151 *
0152 *
0153 *
0154 *
0155 *
0156 *
0157 *
0158 *
0159 *
0160 *
0161 *
0162 *
0163 *
0164 *
0165 *
0166 *
0167 *
0168 *
0169 *
0170 *
0171 *
0172 *
0173 *
0174 *
0175 *
0176 *
0177 *
0178 *
0179 *
0180 *
0181 *
0182 *
0183 *
0184 *
0185 *
0186 *
0187 *
0188 *
0189 *
0190 *
0191 *
0192 *
0193 *
0194 *
0195 *
0196 *
0197 *
0198 *
0199 *
0200 *
0201 *
0202 *
0203 *
0204 *
0205 *
0206 *
0207 *
0208 *
0209 *
0210 *
0211 *
0212 *
0213 *
0214 *
0215 *
0216 *
0217 *
0218 *
0219 *
0220 *
0221 *
0222 *
0223 *
0224 *
0225 *
0226 *
0227 *
0228 *
0229 *
0230 *
0231 *
0232 *
0233 *
0234 *
0235 *
0236 *
0237 *
0238 *
0239 *
0240 *
0241 *
0242 *
0243 *
0244 *
0245 *
0246 *
0247 *
0248 *
0249 *
0250 *
0251 *
0252 *
0253 *
0254 *
0255 *
0256 *
0257 *
0258 *
0259 *
0260 *
0261 *
0262 *
0263 *
0264 *
0265 *
0266 *
0267 *
0268 *
0269 *
0270 *
0271 *
0272 *
0273 *
0274 *
0275 *
0276 *
0277 *
0278 *
0279 *
0280 *
0281 *
0282 *
0283 *
0284 *
0285 *
0286 *
0287 *
0288 *
0289 *
0290 *
0291 *
0292 *
0293 *
0294 *
0295 *
0296 *
0297 *
0298 *
0299 *
0300 *
0301 *
0302 *
0303 *
0304 *
0305 *
0306 *
0307 *
0308 *
0309 *
0310 *
0311 *
0312 *
0313 *
0314 *
0315 *
0316 *
0317 *
0318 *
0319 *
0320 *
0321 *
0322 *
0323 *
0324 *
0325 *
0326 *
0327 *
0328 *
0329 *
0330 *
0331 *
0332 *
0333 *
0334 *
0335 *
0336 *
0337 *
0338 *
0339 *
0340 *
0341 *
0342 *
0343 *
0344 *
0345 *
0346 *
0347 *
0348 *
0349 *
0350 *
0351 *
0352 *
0353 *
0354 *
0355 *
0356 *
0357 *
0358 *
0359 *
0360 *
0361 *
0362 *
0363 *
0364 *
0365 *
0366 *
0367 *
0368 *
0369 *
0370 *
0371 *
0372 *
0373 *
0374 *
0375 *
0376 *
0377 *
0378 *
0379 *
0380 *
0381 *
0382 *
0383 *
0384 *
0385 *
0386 *
0387 *
0388 *
0389 *
0390 *
0391 *
0392 *
0393 *
0394 *
0395 *
0396 *
0397 *
0398 *
0399 *
0400 *
0401 *
0402 *
0403 *
0404 *
0405 *
0406 *
0407 *
0408 *
0409 *
0410 *
0411 *
0412 *
0413 *
0414 *
0415 *
0416 *
0417 *
0418 *
0419 *
0420 *
0421 *
0422 *
0423 *
0424 *
0425 *
0426 *
0427 *
0428 *
0429 *
0430 *
0431 *
0432 *
0433 *
0434 *
0435 *
0436 *
0437 *
0438 *
0439 *
0440 *
0441 *
0442 *
0443 *
0444 *
0445 *
0446 *
0447 *
0448 *
0449 *
0450 *
0451 *
0452 *
0453 *
0454 *
0455 *
0456 *
0457 *
0458 *
0459 *
0460 *
0461 *
0462 *
0463 *
0464 *
0465 *
0466 *
0467 *
0468 *
0469 *
0470 *
0471 *
0472 *
0473 *
0474 *
0475 *
0476 *
0477 *
0478 *
0479 *
0480 *
0481 *
0482 *
0483 *
0484 *
0485 *
0486 *
0487 *
0488 *
0489 *
0490 *
0491 *
0492 *
0493 *
0494 *
0495 *
0496 *
0497 *
0498 *
0499 *
0500 *
0501 *
0502 *
0503 *
0504 *
0505 *
0506 *
0507 *
0508 *
0509 *
0510 *
0511 *
0512 *
0513 *
0514 *
0515 *
0516 *
0517 *
0518 *
0519 *
0520 *
0521 *
0522 *
0523 *
0524 *
0525 *
0526 *
0527 *
0528 *
0529 *
0530 *
0531 *
0532 *
0533 *
0534 *
0535 *
0536 *
0537 *
0538 *
0539 *
0540 *
0541 *
0542 *
0543 *
0544 *
0545 *
0546 *
0547 *
0548 *
0549 *
0550 *
0551 *
0552 *
0553 *
0554 *
0555 *
0556 *
0557 *
0558 *
0559 *
0560 *
0561 *
0562 *
0563 *
0564 *
0565 *
0566 *
0567 *
0568 *
0569 *
0570 *
0571 *
0572 *
0573 *
0574 *
0575 *
0576 *
0577 *
0578 *
0579 *
0580 *
0581 *
0582 *
0583 *
0584 *
0585 *
0586 *
0587 *
0588 *
0589 *
0590 *
0591 *
0592 *
0593 *
0594 *
0595 *
0596 *
0597 *
0598 *
0599 *
0600 *
0601 *
0602 *
0603 *
0604 *
0605 *
0606 *
0607 *
0608 *
0609 *
0610 *
0611 *
0612 *
0613 *
0614 *
0615 *
0616 *
0617 *
0618 *
0619 *
0620 *
0621 *
0622 *
0623 *
0624 *
0625 *
0626 *
0627 *
0628 *
0629 *
0630 *
0631 *
0632 *
0633 *
0634 *
0635 *
0636 *
0637 *
0638 *
0639 *
0640 *
0641 *
0642 *
0643 *
0644 *
0645 *
0646 *
0647 *
0648 *
0649 *
0650 *
0651 *
0652 *
0653 *
0654 *
0655 *
0656 *
0657 *
0658 *
0659 *
0660 *
0661 *
0662 *
0663 *
0664 *
0665 *
0666 *
0667 *
0668 *
0669 *
0670 *
0671 *
0672 *
0673 *
0674 *
0675 *
0676 *
0677 *
0678 *
0679 *
0680 *
0681 *
0682 *
0683 *
0684 *
0685 *
0686 *
0687 *
0688 *
0689 *
0690 *
0691 *
0692 *
0693 *
0694 *
0695 *
0696 *
0697 *
0698 *
0699 *
0700 *
0701 *
0702 *
0703 *
0704 *
0705 *
0706 *
0707 *
0708 *
0709 *
0710 *
0711 *
0712 *
0713 *
0714 *
0715 *
0716 *
0717 *
0718 *
0719 *
0720 *
0721 *
0722 *
0723 *
0724 *
0725 *
0726 *
0727 *
0728 *
0729 *
0730 *
0731 *
0732 *
0733 *
0734 *
0735 *
0736 *
0737 *
0738 *
0739 *
0740 *
0741 *
0742 *
0743 *
0744 *
0745 *
0746 *
0747 *
0748 *
0749 *
0750 *
0751 *
0752 *
0753 *
0754 *
0755 *
0756 *
0757 *
0758 *
0759 *
0760 *
0761 *
0762 *
0763 *
0764 *
0765 *
0766 *
0767 *
0768 *
0769 *
0770 *
0771 *
0772 *
0773 *
0774 *
0775 *
0776 *
0777 *
0778 *
0779 *
0780 *
0781 *
0782 *
0783 *
0784 *
0785 *
0786 *
0787 *
0788 *
0789 *
0790 *
0791 *
0792 *
0793 *
0794 *
0795 *
0796 *
0797 *
0798 *
0799 *
0800 *
0801 *
0802 *
0803 *
0804 *
0805 *
0806 *
0807 *
0808 *
0809 *
0810 *
0811 *
0812 *
0813 *
0814 *
0815 *
0816 *
0817 *
0818 *
0819 *
0820 *
0821 *
0822 *
0823 *
0824 *
0825 *
0826 *
0827 *
0828 *
0829 *
0830 *
0831 *
0832 *
0833 *
0834 *
0835 *
0836 *
0837 *
0838 *
0839 *
0840 *
0841 *
0842 *
0843 *
0844 *
0845 *
0846 *
0847 *
0848 *
0849 *
0850 *
0851 *
0852 *
0853 *
0854 *
0855 *
0856 *
0857 *
0858 *
0859 *
0860 *
0861 *
0862 *
0863 *
0864 *
0865 *
0866 *
0867 *
0868 *
0869 *
0870 *
0871 *
0872 *
0873 *
0874 *
0875 *
0876 *
0877 *
0878 *
0879 *
0880 *
0881 *
0882 *
0883 *
0884 *
0885 *
0886 *
0887 *
0888 *
0889 *
0890 *
0891 *
0892 *
0893 *
0894 *
0895 *
0896 *
0897 *
0898 *
0899 *
0900 *
0901 *
0902 *
0903 *
0904 *
0905 *
0906 *
0907 *
0908 *
0909 *
0910 *
0911 *
0912 *
0913 *
0914 *
0915 *
0916 *
0917 *
0918 *
0919 *
0920 *
0921 *
0922 *
0923 *
0924 *
0925 *
0926 *
0927 *
0928 *
0929 *
0930 *
0931 *
0932 *
0933 *
0934 *
0935 *
0936 *
0937 *
0938 *
0939 *
0940 *
0941 *
0942 *
0943 *
0944 *
0945 *
0946 *
0947 *
0948 *
0949 *
0950 *
0951 *
0952 *
0953 *
0954 *
0955 *
0956 *
0957 *
0958 *
0959 *
0960 *
0961 *
0962 *
0963 *
0964 *
0965 *
0966 *
0967 *
0968 *
0969 *
0970 *
0971 *
0972 *
0973 *
0974 *
0975 *
0976 *
0977 *
0978 *
0979 *
0980 *
0981 *
0982 *
0983 *
0984 *
0985 *
0986 *
0987 *
0988 *
0989 *
0990 *
0991 *
0992 *
0993 *
0994 *
0995 *
0996 *
0997 *
0998 *
0999 *
1000 *

```

```

0076 * 051B 051B 0500 0500
0077 * 051E 051E 0500 0500
0078 * 0520 0520 0500 0500
0079 *
0080 *
0081 *
0082 *
0083 *
0084 * 0523 0523 0500 0500
0085 * 0525 0525 0500 0500
0086 * 0527 0527 0500 0500
0087 * 0529 0529 0500 0500
0088 * 052B 052B 0500 0500
0089 * 052D 052D 0500 0500
0090 * 052F 052F 0500 0500
0091 * 0531 0531 0500 0500
0092 * 0533 0533 0500 0500
0093 * 0535 0535 0500 0500
0094 * 0537 0537 0500 0500
0095 * 0539 0539 0500 0500
0096 * 053B 053B 0500 0500
0097 * 053D 053D 0500 0500
0098 * 053F 053F 0500 0500
0099 * 0541 0541 0500 0500
0100 * 0543 0543 0500 0500
0101 * 0545 0545 0500 0500
0102 * 0547 0547 0500 0500
0103 * 0549 0549 0500 0500
0104 * 054B 054B 0500 0500
0105 * 054D 054D 0500 0500
0106 * 054F 054F 0500 0500
0107 * 0551 0551 0500 0500
0108 * 0553 0553 0500 0500
0109 * 0555 0555 0500 0500
0110 * 0557 0557 0500 0500
0111 * 0559 0559 0500 0500
0112 * 055B 055B 0500 0500
0113 * 055D 055D 0500 0500
0114 * 055F 055F 0500 0500
0115 * 0561 0561 0500 0500
0116 * 0563 0563 0500 0500
0117 * 0565 0565 0500 0500
0118 * 0567 0567 0500 0500
0119 * 0569 0569 0500 0500
0120 * 056B 056B 0500 0500
0121 * 056D 056D 0500 0500
0122 * 056F 056F 0500 0500
0123 * 0571 0571 0500 0500
0124 * 0573 0573 0500 0500
0125 * 0575 0575 0500 0500
0126 * 0577 0577 0500 0500
0127 * 0579 0579 0500 0500
0128 * 057B 057B 0500 0500
0129 * 057D 057D 0500 0500
0130 * 057F 057F 0500 0500
0131 * 0581 0581 0500 0500
0132 * 0583 0583 0500 0500
0133 * 0585 0585 0500 0500
0134 * 0587 0587 0500 0500
0135 * 0589 0589 0500 0500
0136 * 058B 058B 0500 0500
0137 * 058D 058D 0500 0500
0138 * 058F 058F 0500 0500
0139 * 0591 0591 0500 0500
0140 * 0593 0593 0500 0500
0141 * 0595 0595 0500 0500
0142 * 0597 0597 0500 0500
0143 * 0599 0599 0500 0500
0144 * 059B 059B 0500 0500
0145 * 059D 059D 0500 0500
0146 * 059F 059F 0500 0500
0147 * 05A1 05A1 0500 0500
0148 * 05A3 05A3 0500 0500
0149 * 05A5 05A5 0500 0500
0150 * 05A7 05A7 0500 0500
0151 * 05A9 05A9 0500 0500
0152 * 05AB 05AB 0500 0500
0153 * 05AD 05AD 0500 0500
0154 * 05AF 05AF 0500 0500
0155 * 05B1 05B1 0500 0500
0156 * 05B3 05B3 0500 0500
0157 * 05B5 05B5 0500 0500
0158 * 05B7 05B7 0500 0500
0159 * 05B9 05B9 0500 0500
0160 * 05BB 05BB 0500 0500
0161 * 05BD 05BD 0500 0500
0162 * 05BF 05BF 0500 0500
0163 * 05C1 05C1 0500 0500
0164 * 05C3 05C3 0500 0500
0165 * 05C5 05C5 0500 0500
0166 * 05C7 05C7 0500 0500
0167 * 05C9 05C9 0500 0500
0168 * 05CB 05CB 0500 0500
0169 * 05CD 05CD 0500 0500
0170 * 05CF 05CF 0500 0500
0171 * 05D1 05D1 0500 0500
0172 * 05D3 05D3 0500 0500
0173 * 05D5 05D5 0500 0500
0174 * 05D7 05D7 0500 0500
0175 * 05D9 05D9 0500 0500
0176 * 05DB 05DB 0500 0500
0177 * 05DD 05DD 0500 0500
0178 * 05DF 05DF 0500 0500
0179 * 05E1 05E1 0500 0500
0180 * 05E3 05E3 0500 0500
0181 * 05E5 05E5 0500 0500
0182 * 05E7 05E7 0500 0500
0183 * 05E9 05E9 0500 0500
0184 * 05EB 05EB 0500 0500
0185 * 05ED 05ED 0500 0500
0186 * 05EF 05EF 0500 0500
0187 * 05F1 05F1 0500 0500
0188 * 05F3 05F3 0500 0500
0189 * 05F5 05F5 0500 0500
0190 * 05F7 05F7 0500 0500
0191 * 05F9 05F9 0500 0500
0192 * 05FB 05FB 0500 0500
0193 * 05FD 05FD 0500 0500
0194 * 05FF 05FF 0500 0500
0195 * 0601 0601 0500 0500
0196 * 0603 0603 0500 0500
0197 * 0605 0605 0500 0500
0198 * 0607 0607 0500 0500
0199 * 0609 0609 0500 0500
0200 * 060B 060B 0500 0500
0201 * 060D 060D 0500 0500
0202 * 060F 060F 0500 0500
0203 * 0611 0611 0500 0500
0204 * 0613 0613 0500 0500
0205 * 0615 0615 0500 0500
0206 * 0617 0617 0500 0500
0207 * 0619 0619 0500 0500
0208 * 061B 061B 0500 0500
0209 * 061D 061D 0500 0500
0210 * 061F 061F 0500 0500
0211 * 0621 0621 0500 0500
0212 * 0623 0623 0500 0500
0213 * 0625 0625 0500 0500
0214 * 0627 0627 0500 0500
0215 * 0629 0629 0500 0500
0216 * 062B 062B 0500 0500
0217 * 062D 062D 0500 0500
0218 * 062F 062F 0500 0500
0219 * 0631 0631 0500 0500
0220 * 0633 0633 0500 0500
0221 * 0635 0635 0500 0500
0222 * 0637 0637 0500 0500
0223 * 0639 0639 0500 0500
0224 * 063B 063B 0500 0500
0225 * 063D 063D 0500 0500
0226 * 063F 063F 0500 0500
0227 * 0641 0641 0500 0500
0228 * 0643 0643 0500 0500
0229 * 0645 0645 0500 0500
0230 * 0647 0647 0500 0500
0231 * 0649 0649 0500 0500
0232 * 064B 064B 0500 0500
0233 * 064D 064D 0500 0500
0234 * 064F 064F 0500 0500
0235 * 0651 0651 0500 0500
0236 * 0653 065
```

**Program Title**

FIXED-POINT DECIMAL MULTIPLICATION FOR SIGNED, PACKED BCD NUMBERS

### Function

**Multiplication of 2 decimal fixed-point numbers.**

Multiplicand, multiplier, and product are of equal length as defined by LENG.

MULTIPLICAND x MULTIPLIER →  
MULTIPLIER

### Parameters

**input:**

Length of numbers (in bytes) is defined by LENG.

MPLC, MPLC+1, MPLC+2, etc., contain multiplicand.

● MPLR, MPLR+1, MPLR+2, etc., contain multiplier.

**Output:**

MPLR, MPLR+1, MPLR+2, etc., contain product.

**Multiplier is destroyed after multiplication.**

Overflow is detected.

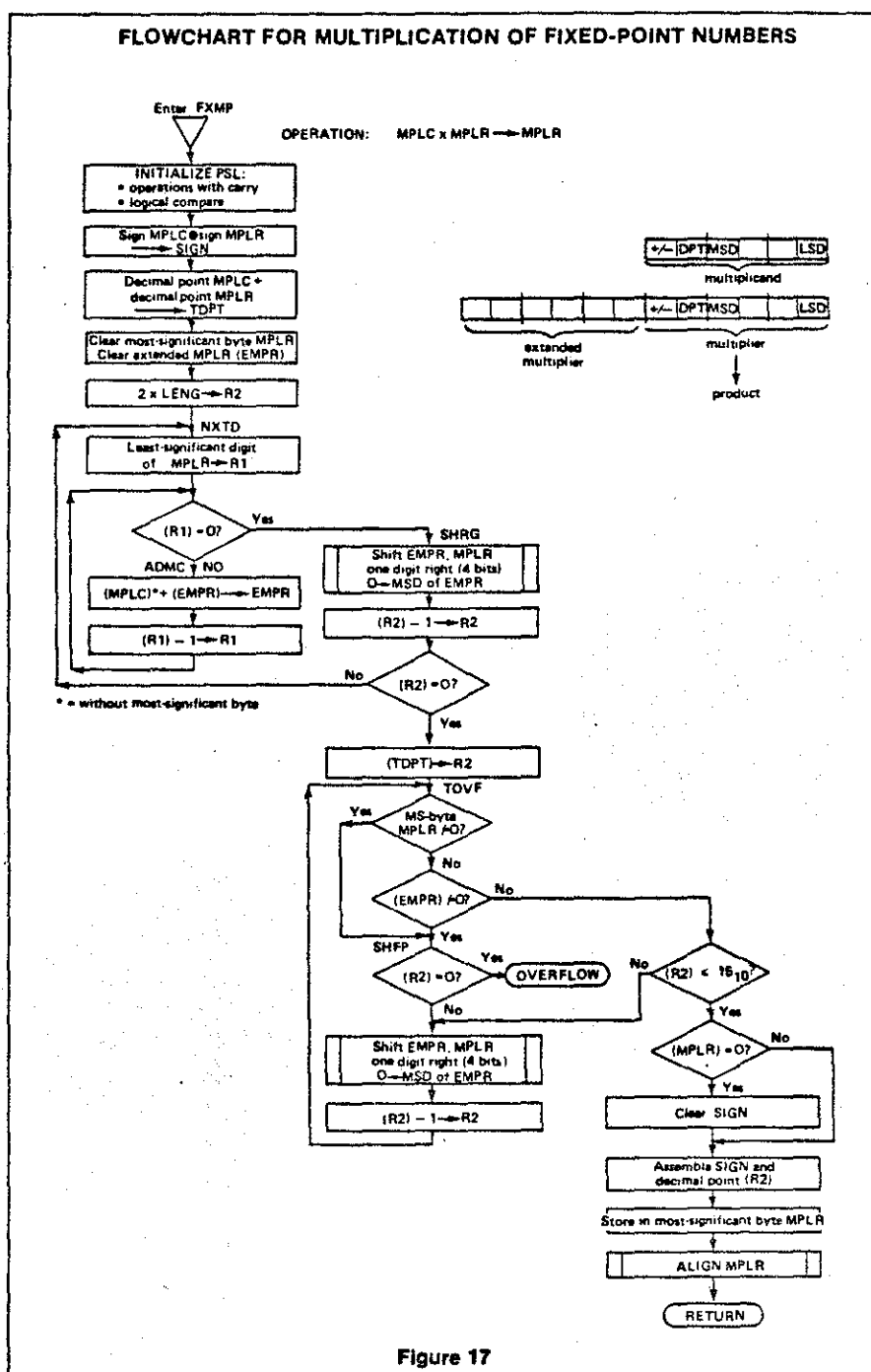
### Special Requirements

Software: Fixed-point alignment subroutine  
ALGN

## Operation

Prior to the multiplication algorithm (which is an unsigned operation), the product sign is determined. The product is formed in a double-length register and is right aligned until the decimal point is 15 or less; this is required due to the fixed-point format. Then the product length is reduced to the single-length, fixed-point format; if this is not possible, overflow is detected. A "minus zero" product result is excluded by means of a test during aligning.

Refer to Figures 17 and 18 for flowchart and program listing.



**Figure 17**

| HARDWARE AFFECTED |         |          |         |         |          |          |        |
|-------------------|---------|----------|---------|---------|----------|----------|--------|
| REGISTERS         | R0<br>X | R1<br>X  | R2<br>X | R3<br>X | R1'      | R2'      | R3'    |
| PSU               | F       | II       | SP      |         |          |          |        |
| PSL               | CC<br>X | IDC<br>X | RS      | WC<br>X | OVF<br>X | COM<br>X | C<br>X |

RAM REQUIRED (BYTES): (3 X LENG) +4

ROM REQUIRED (BYTES): 144

MAXIMUM SUBROUTINE NESTING LEVELS: 1

ASSEMBLER/COMPILER USED: TWIN VER 1.0

FIXED-POINT DECIMAL MULTIPLICATION FOR  
SIGNED, PACKED BCD NUMBERS

TWIN ASSEMBLER VER 1.0

PAGE 0001

LINE ADDR OBJECT E SOURCE

```

0001 * PD750003
0002 *=====
0003 * FIXED POINT DECIMAL MULTIPLICATION FOR *
0004 * SIGNED, PACKED-BCD NUMBERS *
0005 *=====
0006 * OPERATION: MULTIPLICAND X MULTIPLIER -> MULTIPLIER
0007 * MULTIPLICAND IS IN MPLC,MPLC+1,MPLC+2, ETC
0008 * MULTIPLIER IS IN MPLR,MPLR+1,MPLR+2, ETC
0009 * PRODUCT IS IN MPLR,MPLR+1,MPLR+2, ETC
0010 * MULTIPLIER IS DESTROYED AFTER MULTIPLICATION
0011 * MPLC,MPLR ARE POST-SIGNIFICANT BYTES
0012 * LENGTH OF NUMBERS (IN BYTES) IS DEFINED BY LENG
0013 * ALLOWED RANGE: 1 < LENG < 65
0014 * REQUIRED NUMBER OF DECIMALS IN PRODUCT MUST BE
0015 * STORED IN LOCATION DPM1 (MAX = 15).
0016 * NUMBERS ARE IN SIGN-MAGNITUDE NOTATION.
0017 * MS-BYTE HOLDS SIGN AND DECIMAL POINT INFORMATION:
0018 * SIGN IS IN MS 4 BITS. M'0' IS 4, M'1' IS -
0019 * DECIMAL POINT IS IN LS 4 BITS BINARY CODED.
0020 * RANGE (0 THRU 15) EQUALS NUMBER OF DECIMALS
0021 *
0022 * DEFINITIONS OF SYMBOLS.
0023 *
0024 0000 R0 EAU 0 PROCESSOR REGISTERS
0025 0001 R1 EAU 1
0026 0002 R2 EAU 2
0027 0003 R3 EAU 3
0028 0006 MC EAU M'00' PSL: 1=WITH, 0=WITHOUT CARRY
0029 0002 COM EAU M'02' 1=LOGIC, 0=ARITH COMPARE
0030 0001 C EAU M'01' CARRY/BORROW
0031 0000 Z EAU 0 BRANCH COND.: ZERO
0032 0002 LT EAU 2 LESS THAN
0033 0003 UN EAU 3 UNCONDITIONAL
0034 *
0035 * PARAMETERS *
0036 *
0037 0450 ALGN EAU M'450' ADDRESS OF ALIGNMENT SUBROUTINE
0038 0005 LENG EAU 5 LENGTH OF PARAMETERS (BYTES)
0039 *
0040 0000 ORG M'700'
0041 *
0042 0700 MPLC RES LENG MULTIPLICAND
0043 0705 EMPL RES LENG EXTENDED MULTIPLIER
0044 0708 MPLR RES LENG MULTIPLIER
0045 * NOTE: EMPL AND MPLR MUST BE IN SUCCESSIVE
0046 * ROM LOCATIONS FOR DOUBLE-LENGTH SHIFT.
0047 070F SIGN RES 1 TEMPORARY SIGN
0048 0710 TEMP RES 2 TEMPORARY STORAGE FOR ADDRESS
0049 0712 TDPT RES 1 TEMPORARY STORAGE FOR DECIMAL POINT
0050 *
0051 0713 ORG M'500'

```

TWIN ASSEMBLER VER 1.0

PAGE 0000

LINE ADDR OBJECT E SOURCE

```

0052 * SUBROUTINE TO SHIFT EMPL AND MPLR ONE DIGIT RIGHT *
0053 *=====
0054 * PRIOR TO ENTRY: MC IN PSL MUST BE 1
0055 *
0056 0500 SHEN LODI R1 4 LOAD LOOP COUNTER
0057 0509 SHEB LODI R3 -LENG-LENG LOAD INDEX REGISTER
0058 0504 CPSE C CLEAR CARRY
0059 0506 SHEB LODA R0 EMPL-256+LENG-LENG R3 FETCH BYTE
0060 0509 R0D R0 ROTATE RIGHT
0061 050A STRA R0 EMPL-256+LENG-LENG R3 RESTORE BYTE
0062 050B BIRR R3 SHEB BRANCH IF ALL NOT SHIFTED
0063 050C BORS R1 SHEB BRANCH IF 4 BITS NOT SHIFTED
0064 0511 RETG UN RETURN
0065 *
0066 *=====
0067 * FIXED POINT MULTIPLICATION *
0068 *=====
0069 *
0070 0512 7704 PAMP PPSL MC=COM OPERATIONS WITH CARRY, LOGICAL COMPARE
0071 0514 000700 LODA R1 MPLC FETCH MS-BYTE MULTIPLICAND
0072 0517 01 LODZ R1 SAVE IN R0

```

```

0075 0510 00070A LODA R2 MPLR FETCH MS-BYTE MULTIPLIER
0076 0510 22 EORZ R2 TAKE EX-OR OF SIGNS
0077 051C 44F0 ANDI R0 M'0F' REMOVE NON-SIGN DIGIT
0078 051E C0070F STRA R0 SIGN SAVE SIGN
0079 0521 01 LODZ R1 MS-BYTE OF MPLC TO R0
0080 0522 440F ANDI R0 M'0F' REMOVE SIGN MPLC, KEEP DECIMAL POINT
0081 0524 40F0 ANDI R2 M'0F' REMOVE SIGN MPLR, KEEP DECIMAL POINT
0082 0526 7501 CPSE C CLEAR CARRY
0083 0528 02 R0DZ R2 ADD DECIMAL POINT POSITIONS
0084 0529 C00712 STRA R0 TDPT SAVE NEW DECIMAL POINT POSITION
0085 *
0086 052C 20 EORZ R0 CLEAR R0
0087 052D 0704 LODI R3 LENG+1 LOAD INDEX REGISTER
0088 052F C4705 CLEN STRA R0 EMPL-R3 - CLEAR MS-BYTE MPLR, FULL EMPL
0089 0532 5070 BBRZ R3 CLEN BRANCH IF NOT DONE
0090 *
0091 0534 000A LODI R2 LENG-LENG NUMBER OF DIGITS TO LOOP COUNTER
0092 0536 00070E MXTD LODA R1 MPLR+LENG-1 FETCH LS-BYTE MULTIPLIER
0093 0539 450F ANDI R1 M'0F' TAKE ONLY LS-DIGIT
0094 053B 1026 BCTR Z SHRG BRANCH IF ZERO
0095 *
0096 *
0097 053D 7501 RDMC CPSE C ADD MPLC (WITHOUT MS-BYTE) TO EMPL
0098 053D 0704 LODI R3 LENG+1 LOAD INDEX REGISTER
0099 0541 0F6705 RDMB LODA R0 EMPL-R3 - FETCH BYTE OF EXTENDED MULTIPLIER
0100 0544 0466 ADDI R0 M'66' ADD OFFSET
0101 0546 C4705 STRA R0 EMPL-R3 RESTORE INTERMEDIATE SUM
0102 0549 F076 BBRZ R3 RDMB BRANCH IF ALL BYTES NOT ADDED
0103 054B 0704 LODI R3 LENG+1 LOAD INDEX REGISTER
0104 054D 0F6705 RDMB LODA R0 EMPL-R3 FETCH BYTE OF INTERMEDIATE SUM
0105 0550 0F6708 ADDA R0 MPLC-R3 ADD BYTE OF MULTIPLICAND
0106 0553 94 DAR R0 DECIMAL ADJUST RESULT
0107 0554 0F6705 STRA R0 EMPL-R3 RESTORE RESULTING BYTE

```

TWIN ASSEMBLER VER 1.0

PAGE 0003

LINE ADDR OBJECT E SOURCE

```

0108 0557 F074 BBRZ R3 RDMB BRANCH IF NOT READY
0109 0559 0C0705 LODA R0 EMPL RESTORE MS-BYTE EXTENDED MULTIPLIER
0110 055C 0400 ALGI R0 0 ALG CARRY
0111 055E C00705 STRA R0 EMPL RESTORE
0112 0561 F95A BBRZ R1 RDMC DECREMENT DIGIT, BRANCH IF NOT 0
0113 0563 2F0500 SHRG BSTR UN SHEN SHIFT EMPL AND MPLR RIGHT ONE DIGIT POSITION
0114 0566 F04E BBRZ R2 NOTD BRANCH IF MULTIPLICATION NOT READY
0115 *
0116 0568 000712 LODA R2 TDPT - DECIMAL POINT TO R2
0117 056B 0706 TOWF LODI R3 LENG+1 TEST OVERFLOW, LOAD INDEX REGISTER
0118 056D 0F4705 TOMB LODA R0 EMPL-R3 - FETCH BYTE OF EMPL OR MS-BYTE
0119 *
0120 0570 0014 BCTR Z SHFF BRANCH IF NOT ZERO
0121 0572 5079 BBRZ R3 TOMB BRANCH IF ALL NOT TESTED
0122 *
0123 0574 E610 COMI R2 16 TEST IF DECIMAL POINT IS < 16
0124 0576 0011 BCTR LT SHFF BRANCH IF TOO BIG
0125 0579 00070F 100A R2 SIGN ASSEMBLE SIGN AND DECIMAL POINT
0126 057B C0070A RDMB STRA R2 MPLR STORE IN MS-BYTE MPLR
0127 057E 0407 LODI R0 CMLR HIGH-ORDER ADDRESS MPLR TO R0
0128 0580 050A LODI R1 MPLR LOW-ORDER ADDRESS MPLR TO R1
0129 0582 3F0450 BSTR UN ALGN ALIGN PRODUCT, SET + SIGN IF
0130 *
0131 0585 17 RETG UN PRODUCT IS ZERO
0132 *
0133 0586 02 SHFF LODZ R2 UPDATE CC FOR NUMBER OF DECIMALS
0134 0587 1007 BCTR Z OVFL BRANCH IF ZERO OVERFLOW
0135 0589 F006 SHFB BBRZ R2 14Z DECREASE DECIMAL POINT
0136 058B 3F0500 BSTR UN SHEN SHIFT EMPL + MPLR RIGHT
0137 058E 102B BCTR UN TOWF BRANCH FOR OVERFLOW TEST
0138 *
0139 0590 40 OVFL HALT ARITHMETIC OVERFLOW
0140 *
0141 0000 END 0

```

TOTAL ASSEMBLY ERRORS = 0000

Figure 18

### Program Title

FIXED-POINT DECIMAL DIVISION FOR  
SIGNED, PACKED BCD NUMBERS

### Function

Division of 2 decimal numbers (fixed point).

Dividend, divisor, and quotient are of equal  
length as defined by LENG.

DIVND:DIVISOR → DIVDND.

### Parameters

#### Input:

Length of numbers (in bytes) is defined by  
LENG.

DVDN, DVDN+1, DVDN+2, etc., contain div-  
idend.

DVSR, DVSR+1, DVSR+2, etc., contain divi-  
sor.

#### Output:

DVDN, DVDN+1, DVDN+2, etc., contain  
quotient.

Dividend is destroyed after division.

Overflow is detected.

### Special Requirements

Software: Fixed-point alignment subroutine  
ALGN.

### Operation

Prior to the division algorithm (which is an  
unsigned operation), the sign of the quo-  
tient is determined. To obtain maximum  
precision, the division procedure is contin-  
ued until either a non-zero most-significant  
digit is detected or the maximum allowed  
decimal point position is reached. Then the  
resulting quotient is aligned with a minus  
zero result suppressed. Overflow is detect-  
ed if the divisor is zero.

Refer to Figures 19 and 20 for flowchart and  
program listing.

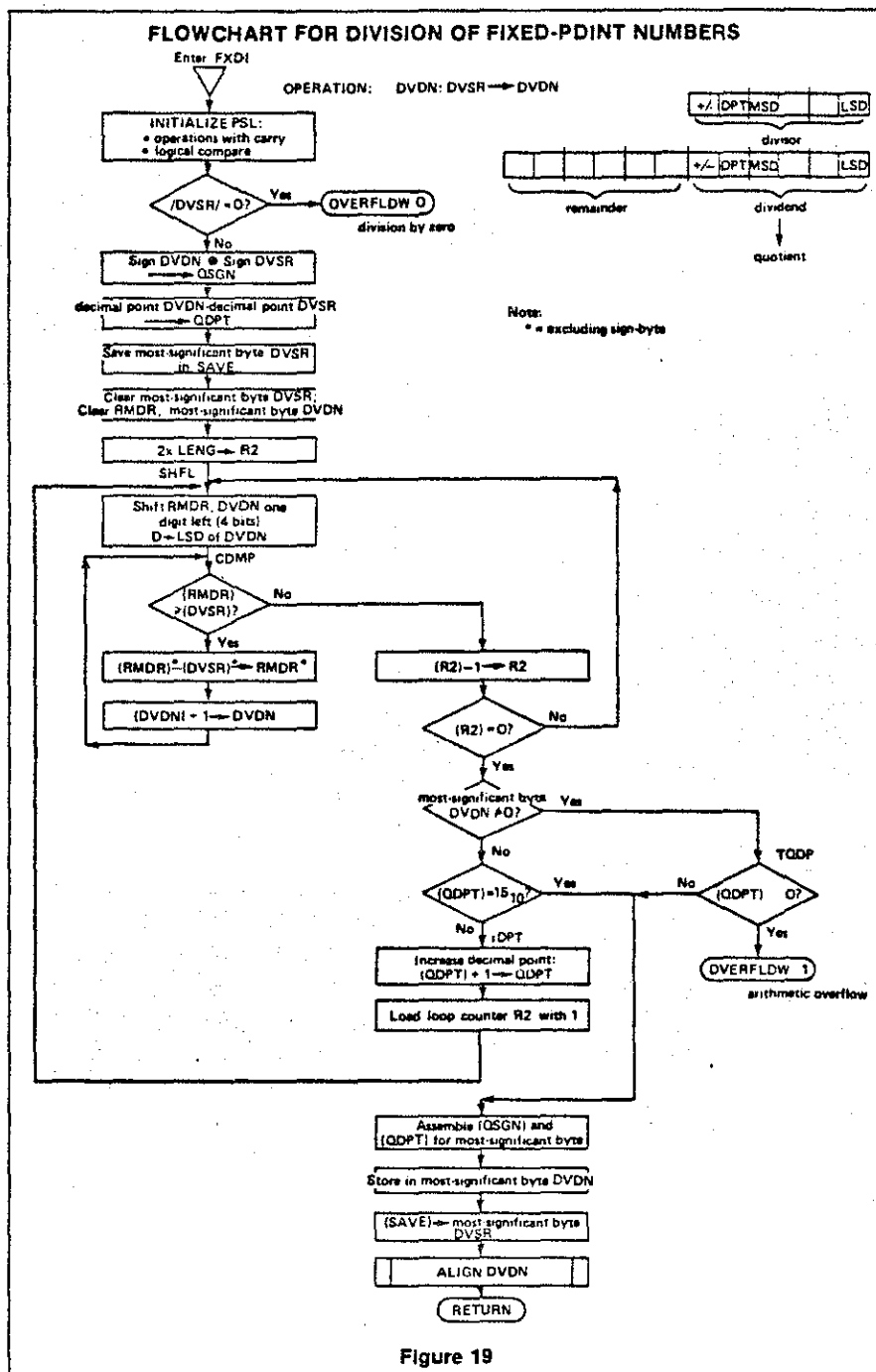


Figure 19

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
| PSU               | F  | II  | SP |    |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |
|                   | X  | X   |    | X  | X   | X   | X   |

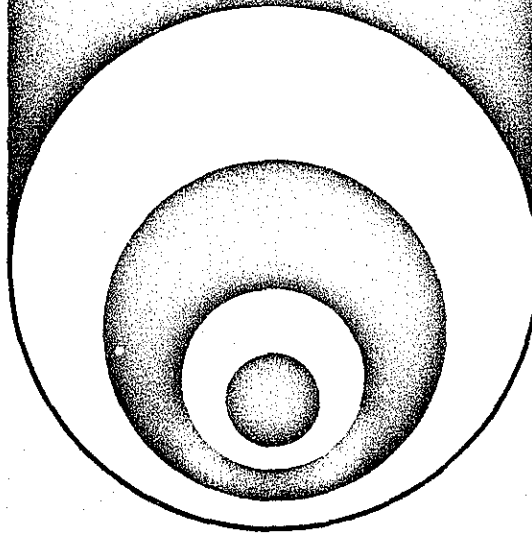
  

|                                       |                |
|---------------------------------------|----------------|
| RAM REQUIRED (BYTES):                 | (3 × LENG) + 5 |
| ROM REQUIRED (BYTES):                 | 166            |
| MAXIMUM SUBROUTINE<br>NESTING LEVELS: | 1              |
| ASSEMBLER/COMPILER USED:              | TWIN VER 1.0   |

**signetics**

**MOS**

**MICROPROCESSOR**



**CONVERSION ROUTINES. . . .AS54**

## 2650 MICROPROCESSOR APPLICATIONS MEMO

### INTRODUCTION

Conversion routines like binary to BCD, BCD to binary, and BCD to ASCII are often used in microprocessor based systems. This applications memo describes routines for converting:

- Eight-bit unsigned binary to BCD.
- Sixteen-bit signed binary to BCD.
- Signed BCD to binary conversion 1 (using an addition method).
- Signed BCD to binary conversion 2 (using a multiplication method).
- Signed BCD to ASCII
- ASCII to BCD
- Hexadecimal to ASCII
- ASCII to Hexadecimal

### 1. EIGHT-BIT UNSIGNED BINARY-TO-BCD CONVERSION

#### FUNCTION:

Converts an unsigned binary number to a BCD number (3 digits).

(BINN)  $\xrightarrow{\text{Conversion}}$  R0, R1

A multiplication method is used.

#### PARAMETERS:

Input: BINN contains the binary number (8 bits unsigned).

Output: Registers R0, R1 contain the BCD result (3 BCD digits).

R0 is the most-significant byte.

The maximum BCD result is 256 decimal.

Refer to figures 1.1 and 1.2 for flowchart and program listing.

|           | HARDWARE AFFECTED |          |    |         |          |          |        |
|-----------|-------------------|----------|----|---------|----------|----------|--------|
| REGISTERS | R0<br>X           | R1<br>X  | R2 | R3      | R1'      | R2'      | R3'    |
| PSU       | F                 | II       | SP |         |          |          |        |
| PSL       | CC<br>X           | IDC<br>X | RS | WC<br>X | OVF<br>X | COM<br>X | C<br>X |

RAM REQUIRED (BYTES): 1

ROM REQUIRED (BYTES): 28

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE NESTING LEVELS: 0

ASSEMBLER/CDMPILER USED: PIPHASM

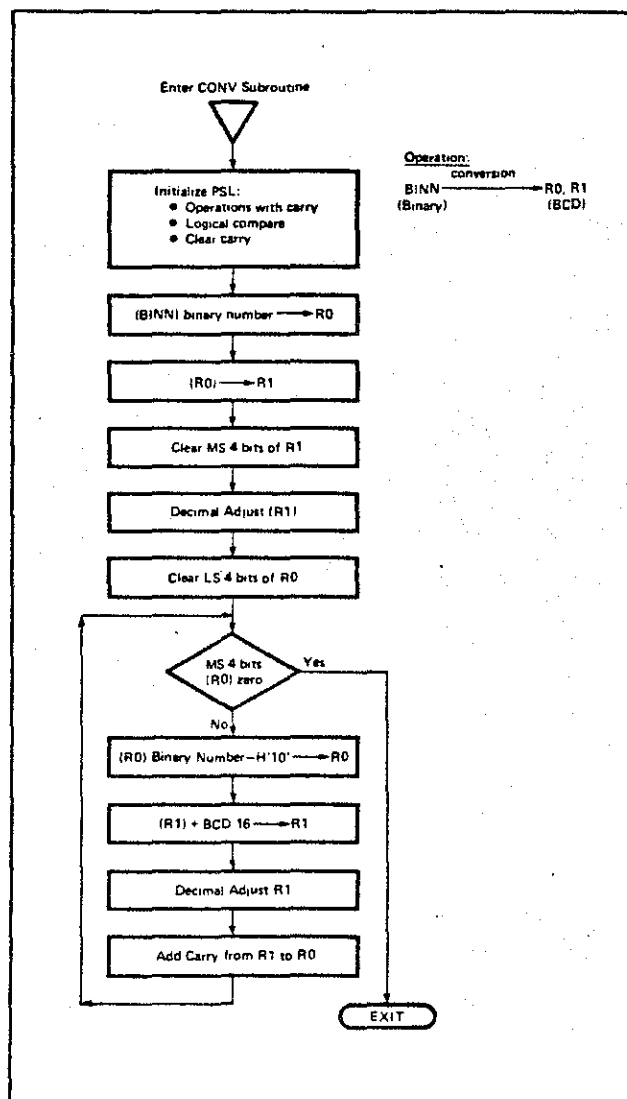


FIGURE 1-1 Flowchart for Eight-Bit Unsigned Binary-to-BCD Conversion (Multiplication Method)



|    |                 |                                                 |                                |
|----|-----------------|-------------------------------------------------|--------------------------------|
| 1  |                 | * PD760050                                      |                                |
| 2  |                 | *****                                           |                                |
| 3  |                 | * 8 BIT UNSIGNED BINARY TO BCD CONVERSION       |                                |
| 4  |                 | *****                                           |                                |
| 5  |                 | *                                               |                                |
| 6  |                 | *THIS ROUTINE CONVERTS AN 8 BIT UNSIGNED BINARY |                                |
| 7  |                 | *NUMBER INTO AN UNSIGNED BCD NUMBER.            |                                |
| 8  |                 | *                                               |                                |
| 9  |                 | *BINARY NUMBER IS IN BINN.                      |                                |
| 10 |                 | *BCD NUMBER (AFTER CONVERSION) IS IN R0,R1.     |                                |
| 11 |                 | * HUNDREDS IN R0                                |                                |
| 12 |                 | * TENS,UNITS IN R1.                             |                                |
| 13 |                 | *                                               |                                |
| 14 |                 | *DEFINITIONS OF SYMBOLS:                        |                                |
| 15 |                 | *                                               |                                |
| 16 | 0000            | R0 EQU 0                                        | PROCESSOR-REGISTERS            |
| 17 | 0001            | R1 EQU 1                                        |                                |
| 18 | 0008            | WC EQU H'08'                                    | PSL: 1=WITH, 0=WITHOUT CARRY   |
| 19 | 0002            | COM EQU H'02'                                   | 1=LOGIC, 0=ARITH.COMPARE       |
| 20 | 0001            | C EQU H'01'                                     | CARRY=BORROW                   |
| 21 | 0003            | UN EQU 3                                        | BRANCH COND.: UNCONDITIONAL    |
| 22 | 0002            | LT EQU 2                                        | LESS THAN                      |
| 23 |                 | *                                               |                                |
| 24 |                 | *                                               |                                |
| 25 |                 | ORG H'600'                                      |                                |
| 26 |                 | *                                               |                                |
| 27 | 0600            | BINN RES 1                                      | BINARY NUMBER.                 |
| 28 |                 | *                                               |                                |
| 29 |                 | ORG H'500'                                      | START ADDRESS OF ROUTINE.      |
| 30 |                 | *                                               |                                |
| 31 |                 | *                                               | INITIALISATION:                |
| 32 | 0500 0500 77 0A | CONV PPSL WC+COM                                | WITH CARRY, LOGICAL COMPARE    |
| 33 | 0502 75 01      | CPSL C                                          | CLEAR CARRY FLAG IN PSL.       |
| 34 | 0504 0C 06 00   | LODA,R0 BINN                                    | 8 BIT BIN.NUMBER -> R0.        |
| 35 | 0507 C1         | STRZ R1                                         | (R0) -> R1.                    |
| 36 | 0508 45 0F      | ANDI,R1 H'0F'                                   | CLEAR MS 4 BITS BIN. NUMBER    |
| 37 | 050A 05 66      | ADDI,R1 H'66'                                   | PREPARE R1 FOR DECIMAL ADJUST. |
| 38 | 050C 95         | DAR,R1                                          |                                |
| 39 |                 | *                                               |                                |
| 40 | 050D 44 F0      | ANDI,R0 H'F0'                                   | CLEAR LS 4 BITS.               |
| 41 |                 | *                                               |                                |
| 42 | 050F 050F E4 10 | LOOP COMI,R0 H'10'                              |                                |
| 43 | 0511 1A 09      | BCTR,LT EXIT                                    | IF MS 4 BITS ZERO THEN RETRUN. |
| 44 | 0513 A4 0F      | SUBI,R0 H'10'-1                                 | SUBTRACT 1 FROM MS 4 BITS      |
| 45 | 0515 05 7B      | ADDI,R1 H'16'+H'66'-1                           | ADD BCD 16 AND PREPARE         |
| 46 | 0517 95         | DAR,R1                                          | FOR DECIMAL ADJUST.            |
| 47 | 051B 04 00      | ADDI,R0 0                                       | ADD CARRY TO MS BCD DIGIT      |
| 48 | 051A 1B 73      | BCTR,UN LDOP                                    | BRANCH AGAIN                   |
| 49 |                 | *                                               |                                |
| 50 | 051C 051C 40    | EXIT HALT                                       | END OF CONVERSION.             |
| 51 |                 | END                                             |                                |

FIGURE 1-2 Program Listing for Eight-Bit Unsigned Binary-to-BCD Conversion

## 2. SIXTEEN-BIT SIGNED BINARY-TO-BCD CONVERSION

### FUNCTION:

Converts a signed 16-bit binary number to a signed BCD number.

Subtraction of base numbers is used.

### PARAMETERS:

Input: BINN, BINN+1 contain the signed binary number.

BINN is the most-significant byte.

Binary number is destroyed after conversion.

Output: BCDD, BCDD+1, BCDD+2 contain the BCD result.

BCDD contains the sign and the most-significant BCD digit.

The minimum BCD result is -32768 decimal.

The maximum BCD result is +32767 decimal.

Refer to figures 2.1 and 2.2 for flowchart and program listing.

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
| PSU               | F  | II  | SP |    |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |
|                   | X  | X   |    | X  | X   |     | X   |

RAM REQUIRED (BYTES): 5

ROM REQUIRED (BYTES): 106

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE

NESTING LEVELS: 0

ASSEMBLER/COMPILER USED: PIPHASM

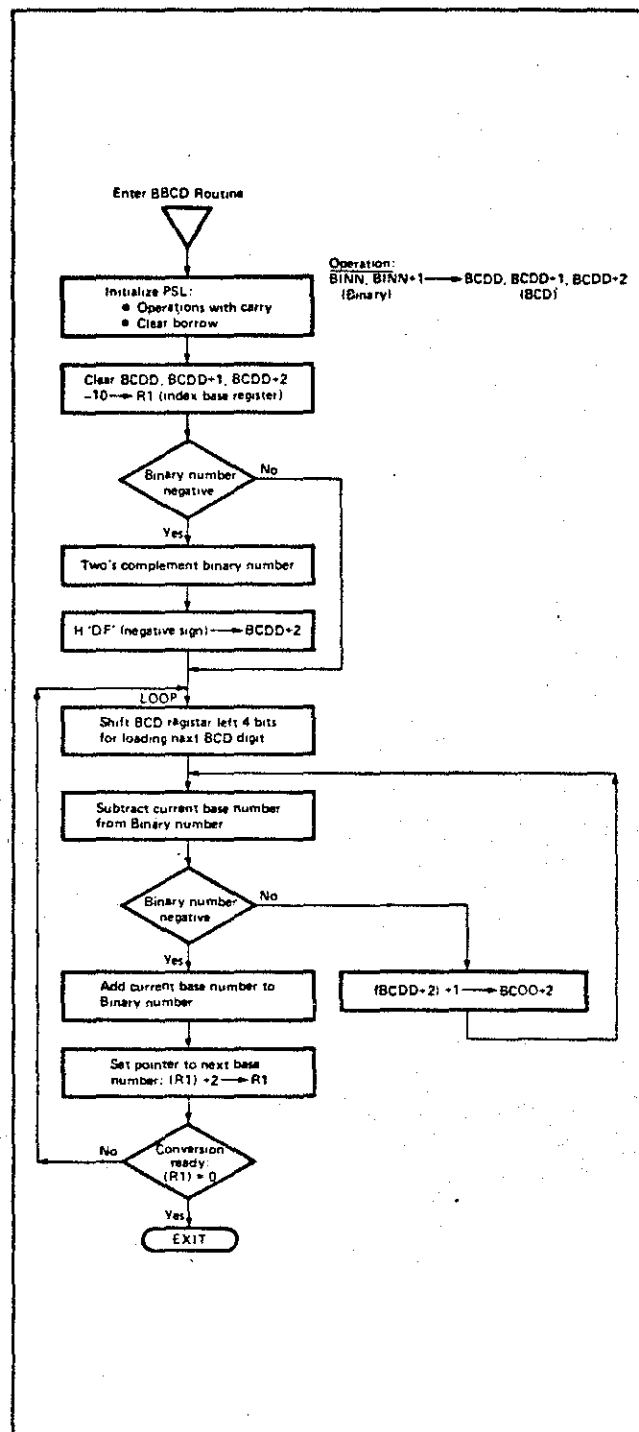


FIGURE 2-1 Flowchart for Signed Binary-to-BCD Conversion

```

1 * P0740051
2 +-----+
3 * BINARY TO BCD CONVERSION
4 +-----+
5
6 *THIS ROUTINE CONVERTS A SIGNED BINARY NUMBER
7 * (16 BITS) INTO A SIGNED BCD NUMBER
8 * (124 BITS) SIGN + 5 BCD DIGITS).
9
10
11 *THE BINARY NUMBER IS IN BINM,BINM+1.
12 *THE BCD NUMBER IS IN BCD0,BCD0+1,BCD0+2.
13 *BINM AND BCD0 ARE MOST SIGNIFICANT BYTES.
14 *MS NIBBLE OF BCD0+0 FOR POSITIVE BINARY NUMBERS.
15 *MS NIBBLE OF BCD0+9 FOR NEGATIVE BINARY NUMBERS.
16
17 *SUBTRAHENDS ARE PLACED IN REGISTER BASE (10 BYTES)
18
19 *DEFINITION OF SYMBOLS:
20
21 R0 EQU 0 PROCESSOR-REGISTERS
22 R1 EQU 1
23 R2 EQU 2
24 R3 EQU 3
25 MC EQU H'00' PSL: 1=WITH, 0=WITHOUT CARRY
26 C EQU H'01' CARRY-BORROW
27 N EQU 2 BRANCH COND.: NEGATIVE
28 UN EQU 3 UNCONDITIONALY
29
30 *
31 ORG H'400' START ADDRESS
32
33 BINM RES 2 BINARY NUMBER MEMORY LOCATION
34 BCD0 RES 3 BCD REGISTER
35 BASE DATA H'27,10' 10000
36 DATA H'03,00' 1000
37 DATA H'00,00' 100
38 DATA H'00,00' 10
39 DATA H'00,01' 1
40 LEN EQU 8-BASE LENGTH BASE REGISTER
41 ORG H'500' START ADDR. OF PROGRAM
42
43 BCD0 PPSL MC+C ARITHMETIC+ROTATE WITH CARRY:
44 CLEAR BORROW.
45
46 EOR2 R0 INITIALISATION: CLEAR R0.
47 LDBI,R3 3
48 LOCP STRA,R0 BCD0,R3,- CLEAR 3 BYTES OF BCD REGISTER.
49 BRNR,R3 LOCP
50 LDBI,R1 -LEN LENGTH OF BASE REGISTER.
51
52 LDBA,R2 BINM MS 4 BITS BINARY NUMBER.
53 BCFR,N LOOP IF POS. GO TO LOOP
54 COMP LDBI,R2 2 LOAD INDEX REGISTER.
55 LOCI EOR2 R0 TWO'S COMPLEMENT BY
56 SUBA,R0 BINM,R2- SUBTRACTING FROM ZERO.
57 STRR,R0 BINM,R2
58 BRNR,R2 LOC1 RETURN IF NOT READY.
59 LDBI,R0 H'00' NEGATIVE SIGN INDICATION.
60 STRA,R0 BCD0+2 SIGN IN LSB OF BCD REGISTER.
61
62 *
63 LOOP CPSL C SHIFT BCD REG. LEFT 4 TIMES.
64 LDBI,R2 4 CLEAR CARRY FOR ROTATE.
65 LPZ LDBI,R3 3 BIT COUNT.
66 LP1 LDBA,R0 BCD0,R3- INDEX BYTE SHIFT.
67 RRL,R0 BCD0,R3- BCD OICIT INTO R0.
68 STRR,R0 BCD0,R3 CARRY (PREVIOUS MS BIT)-> LSB
69 BRNR,R2 LP1 AND MS BIT -> CARRY.
70
71 *
72 SUBL ADDI,R1 2 RESTORE BASE INDEX.
73 LDBI,R2 2 INDEX REGISTER
74 PPSL C CLEAR BORROW
75 LOCP LDBA,R0 BINM,R2- LOAD BINM AND SUBTRACT
76 SUBA,R0 BASE-Z56+LEN+2,R1- CORRESPONDING
77 STRA,R0 BINM,R2 BASE OICIT
78 BRNR,R2 LOC2
79 BCFR,N CORR IF BINM NEG. THEN CORRECTION.
80 LDBA,R0 BCD0+2
81 RDDZ R2 ADD 1 TO LSB OF BCD NUMBER
82 STRA,R0 BCD0+2 C+1 IN PSL AND (R2)=0
83 BCFR,N SUBL
84
85 *
86 CORR LDBI,R2 2 INDEX COUNT.
87 LOCP LDBA,R0 BINM,R2- ADD CORRESPONDING BASE BYTE TO
88 ADDA,R0 BASE-Z56+LEN+2,R1- BINARY NUMBER.
89 STRA,R0 BINM,R2
90 BRNR,R2 LOC3 RETURN IF NOT READY
91 ADDI,R1 3 UPDATE BASE POINTER+1 IN PSL
92 BRNR,R1 LOOP RETURN IF CONVERSION NOT READY
93 EXIT HALT END OF CONVERSION
94
95 END

```

**FIGURE 2-2 Program Listing for Signed Binary-to-BCD Conversion**

## 3. SIGNED BCD-TO-BINARY CONVERSION 1

## FUNCTION:

Converts a five-digit signed BCD number to a sixteen-bit signed binary number.

Addition of base numbers is used.

## PARAMETERS:

Input: BCDD, BCDD+1, BCDD+2 contain the BCD number.

BCDD contains the sign plus the most-significant BCD digit.

The range of BCD numbers is:  $-32768 < \text{BCD Number} < +32767$ .

BCDD is destroyed after the conversion.

Output: BINN, BINN+1 contain the signed binary number.

BINN is the most-significant byte.

Refer to figures 3.1 and 3.2 for flowchart and program listing.

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
|                   | X  | X   | X  | X  |     |     |     |
| PSU               | F  | II  | SP |    |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |
|                   | X  | X   |    | X  | X   |     | X   |

RAM REQUIRED (BYTES): 5

ROM REQUIRED (BYTES): 86

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE

NESTING LEVELS: 0

ASSEMBLER/COMPILER USED: PIPHASM

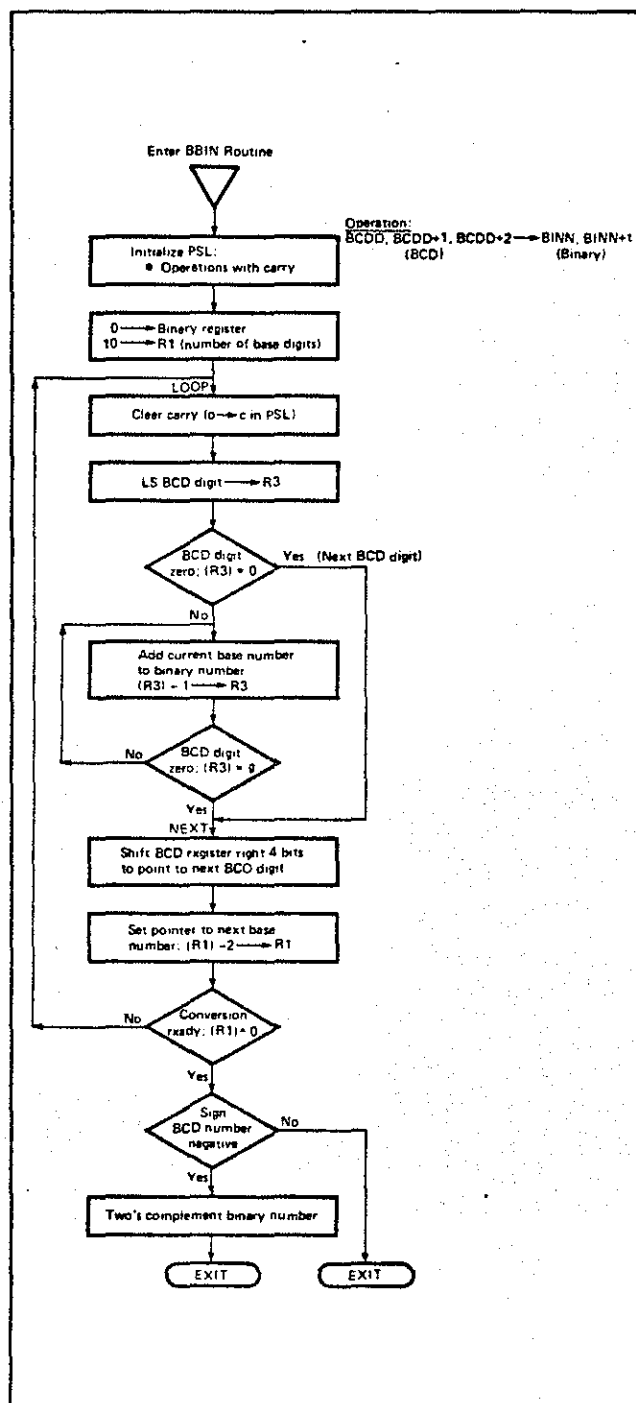


FIGURE 3-1: Flowchart for signed BCD-to-Binary Conversion

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 0000
24 0001
25 0002
26 0003
27 0008
28 0001
29 0000
30 0000
31 0000
32 0000
33
34
35
36 0000
37 0002
38 0005 0005 27 10
39 0007 00 00
40 0009 00 00
41 0008 00 00
42 0000 00 01
43
44
45 0450 0450 77 00
46 0452 20
47 0453 00 00 00
48 0454 00 00 01
49 0459 00 00
50 0458 0450 75 01
51 0450 00 00 00
52 0460 47 00
53 0462 10 11
54 0464 0464 00 02
55 0466 0466 00 00
56 0469 00 00 00
57 046C 00 00 00
58 046F 00 00
59 0471 00 00
60 0473 00 00
61
62 0475 0475 00 00
63 0477 0477 00 00
64 0479 0479 00 00
65 047C 00
66 047D 00 00
67 0480 00 00
68 0482 00 00
69 0484 00 00
70 0486 00 00
71 0488 00 00
72
73 048A 00 00
74 048C 00 00
75 048E 048E 00 01
76 0490 00 00
77 0492 0492 00
78 0493 00 00
79 0496 00 00
80 0499 00 00
81
82 049B 049B 00
83

```

```

* PD760952

* BCD TO BINARY CONVERSION

*
* THIS ROUTINE CONVERTS A SIGNED BCD NUMBER
* (24 BITS: SIGN+5 BCD DIGITS) INTO A SIGNED
* BINARY NUMBER (16 BITS).
* -32768 < BCD NUMBER < +32767
* BCD NUMBER IS LOST AFTER CONVERSION.
*
* THE BINARY NUMBER IS IN BINH, BINH+1.
* THE BCD NUMBER IS IN BCDB, BCDB+1, BCDB+2 (R0-R4).
* THE BASE NUMBERS ARE IN BASE, -BASE+9 (R0, R4).
* BINH AND BCDB ARE MOST SIGNIFICANT BYTES.
*
* PRINCIPLE OF CONVERSION IS:
* BINH = A0.R0+ A1.R1+ A2.R2+ A3.R3+ A4.R4
* A0-A4 = NUMBER OF DIGITS OF BCD NUMBER.
* R0-R4 = BASE NUMBERS FOR CONVERSION.
*
* DEFINITIONS OF SYMBOLS:
R0 EQU 0 PROCESSOR-REGISTERS
R1 EQU 1
R2 EQU 2
R3 EQU 3
MC EQU 0'B0' PSL: 1=WITH, 0=WITHOUT CARRY
C EQU 0'B1' CARRY-BORROW
Z EQU 0 BRANCH COND: ZERO
ON EQU 0 ALL BITS ARE 1
SIGN EQU 0'B0' TB TEST BCD. NUMBER
LEN EQU 10 INDEX NUMBER (LENGTH BASE REG)
*
ORG H'450'
*
BINH RES 2 BINARY NUMBER
BCDB RES 3 BCD NUMBER
BASE DATA H'27:10' 10000
DATA H'03:0B' 1000
DATA H'09:6A' 100
DATA H'00:0A' 10
DATA H'00:01' 1
ORG H'450' START OF PROGRAM
BBIN PPSL MC ARITHMETIC+ROTATE WITH CARRY
EDRZ R0 CLEAR R0
STRA,R0 BINH CLEAR BINARY REGISTERS
STRA,R0 BINH+1
LOBI,R1 LEN INDEX FOR BASE DIGITS
LOOP CPSL C CLEAR CARRY
LDA,R3 BCDB+2 LOAD LS BCD DIGIT IN R3
ANBI,R3 H'0F' CLEAR MS 4 BITS
BCTR,Z NEIT IF ZERO GO TO NEIT
LOCI LOBI,R2 2 LOAD INDEX
LOC2 LDA,R0 BINH,R2,-
ADDA,R0 BASE,R1,- ADB BASE DIGIT TO BIN. NUMBER
STRA,R0 BINH,R2
BRNR,R2 LOC2
ADDI,R1 2 RESTORE BASE POINTER
BDRR,R3 LOC1 IF NOT READY RETURN TO LOC1
*
NEIT LOBI,R2 4 BIT COUNT
LP2 LOBI,R3 -3 INDEX FOR BYTE COUNT
LPI LDA,R0 BCDB-256+3,R3 BCD DIGIT INTO R0
RRR,R0 CARRY (PREVIOUS LS BIT) -> MSB
STRA,R0 BCDB-256+3,R3 AND LS BIT -> CARRY.
BIRR,R3 LPI NEXT BCD DIGIT
CPSL C
BDRR,R2 LP2 NEXT SHIFT OF BCD REG. BIT
BDRR,R1 0+2 UPDATE BASE POINTER WITHOUT
 AFFECTING C FLAG IN PSL AND
 GO TO LOOP IF NOT READY
*
THI,R0 SIGN IF SIGN POS. THEN REABT.
BCFR,ON EIIT CLEAR BORROW
COMP PPSL C NUMBER OF DIGITS
LOBI,R2 2 TWO'S COMPLEMENT BY
LP3 EDRZ R0 SUBTRACTION FROM ZERO
SUBA,R0 BINH,R2,-
STRA,R0 BINH,R2
DRNR,R2 LP3
*
EXIT MALT END OF CONVERSION
END

```

FIGURE 3-2 Program Listing for Signed BCD-to-Binary Conversion

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 0000
23 0001
24 0002
25 0003
26 000B
27 0001
28 0002
29 0005
30
31
32
33 0000
34 0002
35 0005
36
37
38
39
40
41 0450 0450 77 00
42 0452 20
43 0453 CC 06 00
44 0456 CC 06 01
45 0459 07 05
46
47 045B 0C 06 02
48 045E CC 06 05
49
50
51 0461 0461 75 01
52 0463 00 06 00
53 0466 0E 06 01
54 0469 02
55 046A 01
56 046B 02
57 046C 01
58 046D 0E 06 01
59 0470 00 06 00
60 0473 02
61 0474 01
62
63
64 0475 0C 06 02
65 0478 44 0F
66 047A 02
67 047B 05 00
68 047D CD 06 00
69 0490 CC 06 01
70
71
72 0483 05 04
73 0485 0485 06 03
74 0487 0487 0E 06 02
75 048A 00
76 048B CE 06 02
77 048E SA 77
78 0490 F9 73
79 0492 FB 4D
80
81 0494 0C 06 05
82 0497 9A 0D
83 0499 77 01
84 049B 06 02
85 049D 049D 20
86 049E AE 06 00
87 04A1 CC 06 00
88 04A4 SA 77
89
90 04A6 04A6 40
91

```

```

* PB760053

* BCD TO BINARY CONVERSION

*
* THIS ROUTINE CONVERTS A SIGNED BCD NUMBER
* (24 BITS: SIGN + 5 BCD DIGITS) INTO A SIGNED
* BINARY NUMBER (16 BITS).
* -32768 < BCD NUMBER < +32767
* BCD NUMBER IS LOST AFTER CONVERSION
*
* PRINCIPLE:
* BIN. = ((((((A*10) +B) *10) +C) *10) +D) *10) +E
* ABCDE = BCD NUMBER
*
* MULTIPLICATION BY 10 IS DONE BY:
* LOAD R2,R1 WITH BIN. NUMBER, SHIFT LEFT TWICE,
* ADD BIN. NUMBER TO R2,R1, SHIFT LEFT ONCE,
* STORE R2,R1 IN BINN,BINN+1 AS RESULT
*
* DEFINITION OF SYMBOLS:
R0 EQU 0 PROCESSOR-REGISTERS
R1 EQU 1
R2 EQU 2
R3 EQU 3
WC EQU 0'0' PSL: 1=WITH, 0=WITHOUT CARRY
C EQU 0'0' CARRY: BORROW
M EQU 2 COMB: NEGATIVE
NUM EQU 5 INDEX FOR NUMBER OF BCD DIGITS
*
* ORG 0'450'
*
BINN RES 2 BINARY NUMBER
BCDD RES 3 BCD NUMBER AND SIGN
SIGN RES 1 SAVE SIGN BIT
*
* ORG 0'450' START OF PROGRAM
*
BCDC PPSL WC ARJTN: ROTATE WITH CARRY
EDR2 R0 CLEAR R0
STRA,R0 BINN CLEAR BINARY NUMBERS
STRA,R0 BINN+1
LBB1,R3 NUM BCD INDEX REGISTER
*
LDDA,R0 BCDD SAVE SIGN OF BCD NUMBER IN
STRA,R0 SIGN MEMORY LOC. SIGN
*
* MULTIPLY BINARY NUMBER BY 10
* CLEAR CARRY
* LOAD BIN. NUMBER IN R1,R2
* ROTATE REGISTERS R1,R2 LEFT 2
*
LOOP CPSL C
LDDA,R1 BINN
LDDA,R2 BINN+1
RRL,R2
RRL,R1
RRL,R2
RRL,R1
ADDA,R2 BINN+1 ADD BIN. NUMBER TO R1,R2
ADDA,R1 BINN
RRL,R2
RRL,R1 SHIFT R1,R2 LEFT ONCE
*
*
LDDA,R0 BCDD LOAD MS BCD DIGIT IN R0
ANDI,R0 0'0F' CLEAR MS 4 BITS
ABR2 R2 ADD BCD TO BINARY NUMBER
ADDE,R1 0 ADD CARRY TO MS BYTE
STRR,R1 BINN STORE RESULT IN BINN,BINN+1
STRA,R0 BINN+1
*
* ROTATE BCD NUMBER 4 TIMES LEFT
* TO POINT TO NEXT BCD DIGIT
* BIT COUNT
*
LP2 LODI,R2 3 INDEX FOR BYTE COUNT
LPI LDDA,R0 BCDD,R2,-
RRL,R0 SHIFT BCD BYTE LEFT
STRR,R0 BCDD,R2
BRNR,R2 LP1 NEXT BYTE OF BCD REGISTER
BORR,R1 LP2 NEXT BIT SHIFT
BORR,R3 LOOP TO LOOP IF MULTIPLY NOT READY
*
*
LDDA,R0 SIGN
BCFAR 0 EXIT IF SIGN POS. THEN READY
PPSL C CLEAR CARRY
INDEX LOADING
LP3 EDR2 R0 TWO'S COMPLEMENT BY
SUBA,R0 BINN,R2 SUBTRACTING FROM "R0"
STRA,R0 BINN
BRNR,R2 LP3
*
*
EXIT HALT END OF CONVERSION
END

```

FIGURE 4-2 Program Listing for Signed BCD-to-Binary Conversion

## 4. SIGNED BCD-TO-BINARY CONVERSION 2

## FUNCTION:

Converts a five-digit signed BCD number to a sixteen-bit signed binary number.

A multiplication method is used.

## PARAMETERS:

**Input:** BCDD, BCDD+1 contain the BCD number.  
BCDD contains the sign plus the most-significant BCD digit.  
The range of BCD numbers is:  $-32768 < \text{BCD Number} < +32767$

**Output:** BINN, BINN+1 contain the signed binary number.  
BINN is the most-significant byte.

Refer to figures 4.1 and 4.2 for flowchart and program listing.

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
| PSU               | F  | II  | SP |    |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |
|                   | X  | X   |    | X  | X   |     | X   |

RAM REQUIRED (BYTES): 6

ROM REQUIRED (BYTES): 87

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE  
NESTING LEVELS: 0

ASSEMBLER/COMPILER USED: PIPHASM

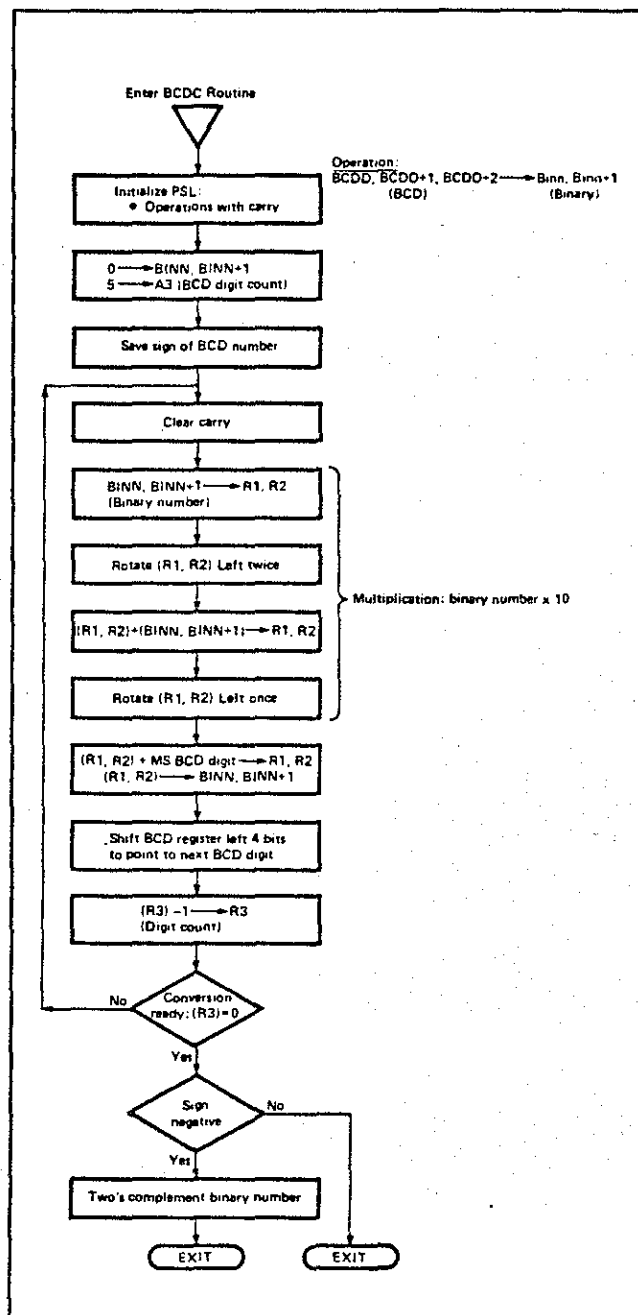


FIGURE 4-1: Flowchart for signed BCD-to-Binary Conversion (Multiplication Method).

## 5. SIGNED BCD-TO-ASCII CONVERSION

## FUNCTION:

Converts  $n$  BCD digits plus sign to  $n + 1$  ASCII characters (sign included).

## PARAMETERS:

**Input:** BCDD, BCDD+1, ..., BCDD+ (numb - 1)  
BCDD contains the sign plus the most-significant digit (2 BCD/byte).  
Numb is the number of BCD bytes.

**Output:** ASCII, ASCII+1, ..., ASCII + (numb - 1) contains the signed result.  
ASCII contains the sign.  
ASCII+1 contains the most-significant byte.

Refer to figures 5.1 and 5.2 for flowchart and program listing.

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
|                   | X  | X   | X  | X  |     |     |     |
| PSU               | F  | II  | SP |    |     |     |     |
|                   |    |     |    |    |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |
|                   | X  | X   |    | X  | X   |     | X   |

RAM REQUIRED (BYTES): N Numb, N Numb+1

ROM REQUIRED (BYTES): 53

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE  
NESTING LEVELS: 0

ASSEMBLER/COMPILER USED: PIPHASM

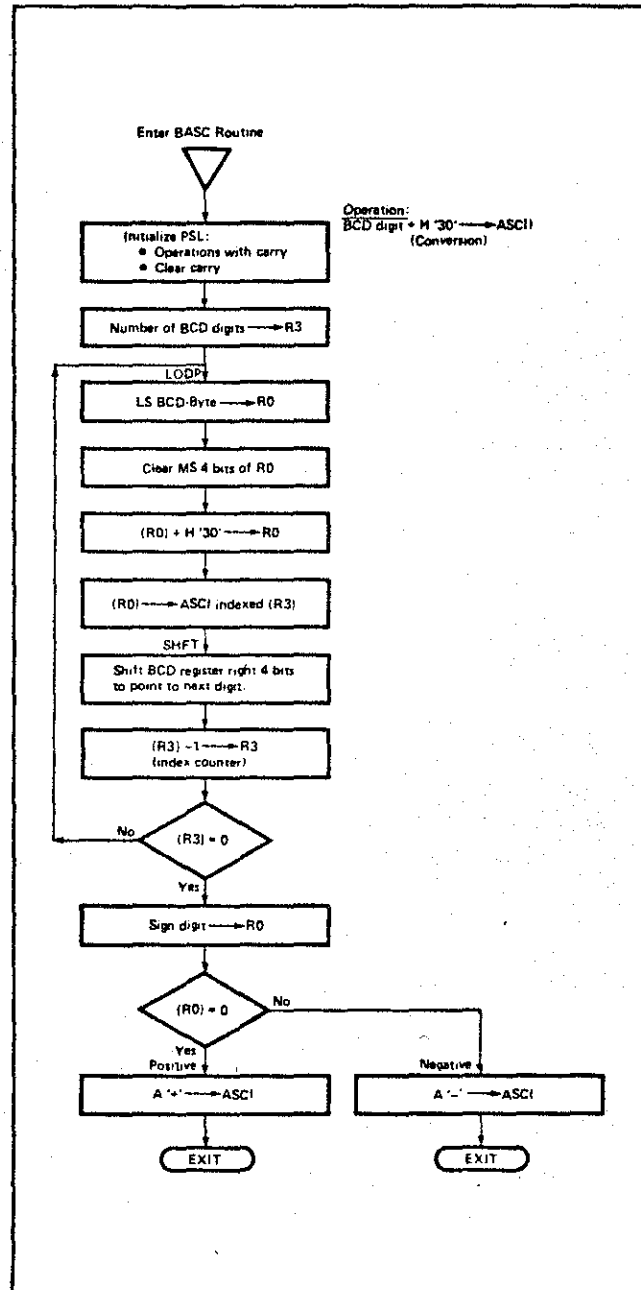


FIGURE 5-1 Flowchart for BCD-to-ASCII Conversion (signed)



```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70

```

```

* PD760954
*+++++
* BCD TO ASCII CONVERSION
*+++++
*
* THIS ROUTINE CONVERTS A SIGNED BCD NUMBER
* INTO ASCII CHARACTERS (SIGN INCLUDED).
* BCD FORMAT: SIGN + BCD DIGITS (TWO DIGITS:BYTES)
* THE NUMBER OF BCD DIGITS -> R3 = NUMB
* THE NUMBER OF BCB BYTES -> R2 = NUMB
* BCD NUMBER IS IN BCDD,BCDD+1,---,BCDD+(N-1)
* ASCII CHARACTERS ARE IN ASCII,ASCII+1,---,ASCII+NUMB
* (SIGN) (BCD DIGITS)
*
* DEFINITIONS OF SYMBOLS:
*
R0 EQU 0
R1 EQU 1
R2 EQU 2
R3 EQU 3
WC EQU H'00' PSL: 1=WITH, 0=WITHOUT CARRY
C EQU H'01' CARRY: BORROW
UN EQU 3 COND: UNCONDITIONAL
Z EQU 0 ZERO
*
* IN THIS EXAMPLE THE CONVERSION OF 5 BCD DIGITS
* IS PERFORMED.
*
NUMB EQU 3 NUMBER OF BCB BYTES
NUM EQU 5 NUMBER OF BCD DIGITS
*
ORG H'4E0'
*
BCDD RES NUMB RESERVE FOR BCD NUMBER
ASCII RES NUMB+1 RESERVE FOR SIGN+ASCII DIGITS
*
ORG H'500' PROGRAM START HERE
*
BASC PPSL WC ARITHMETIC: ROTATE WITH CARRY
CPSL C CLEAR CARRY
LODI,R3 NUM INDEX REGISTER
*
LOOP LODA,R0 BCDD+NUMB-1 LOAD LS BCD DIGIT IN R0
ANDI,R0 H'0F' CLEAR MS 4 BITS
ADDI,R0 H'30' ASCII CHARACTER
STRA,R0 ASCII,R3 STORE ASCII CHARACTER
*
SHFT LODI,R1 4 BIT COUNT
LP2 LODI,R2 -NUMB INDEX FOR BYTE SHIFT
LP1 LODA,R0 BCDD-256+NUMB,R2
RRR,R0 CARRY (PREVIOUS LS BIT) -> MSB
STRA,R0 BCDD-256+NUMB,R2 AND LS BIT -> CARRY
BIRR,R2 LP1
CPSL C CLEAR CARRY
BDRR,R1 LP2
BDRR,R3 LODP IF NOT READY GO TO LOOP
*
SIGN LODA,R0 BCDD+NUMB-1 SIGN -> R0
BCTR,Z POS
NEG LODI,R0 A'-
STRA,R0 ASCII
BCTR,UN EXIT
POS LODI,R0 A'+
STRA,R0 ASCII
*
EXIT HALT END OF CONVERSION
END

```

FIGURE 5-2 Program Listing for BCD-to-ASCII Conversion (Signed)

## 6. ASCII-TO-BCD CONVERSION

## FUNCTION:

Converts  $n$  ASCII digits to  $n$  BCD digits.  
 ASCII  $\longrightarrow$  BCD

## PARAMETERS:

Input: ADIG, ADIG+1, ..., ADIG+( $n-1$ ) contain ASCII digits.  
 The most-significant digit is in ADIG (byte/digit).

Output: BCDD, BCDD+1, ..., BCDD + ( $n-1$ ) contains BCD digits.  
 The most-significant digit is in BCDD (2 digits/byte).

Refer to figures 6.1 and 6.2 for flowchart and program listing.

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
|                   | X  |     | X  | X  |     |     |     |
| PSU               | F  | II  | SP |    |     |     |     |
|                   |    |     |    |    |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |
|                   | X  | X   |    | X  | X   |     | X   |

|                                    |                               |
|------------------------------------|-------------------------------|
| RAM REQUIRED (BYTES):              | $n\text{ADIG} + n\text{BCDD}$ |
| ROM REQUIRED (BYTES):              | 37                            |
| EXECUTION TIME:                    | Variable                      |
| MAXIMUM SUBROUTINE NESTING LEVELS: | 0                             |
| ASSEMBLER/COMPILER USED:           | PIPHASM                       |

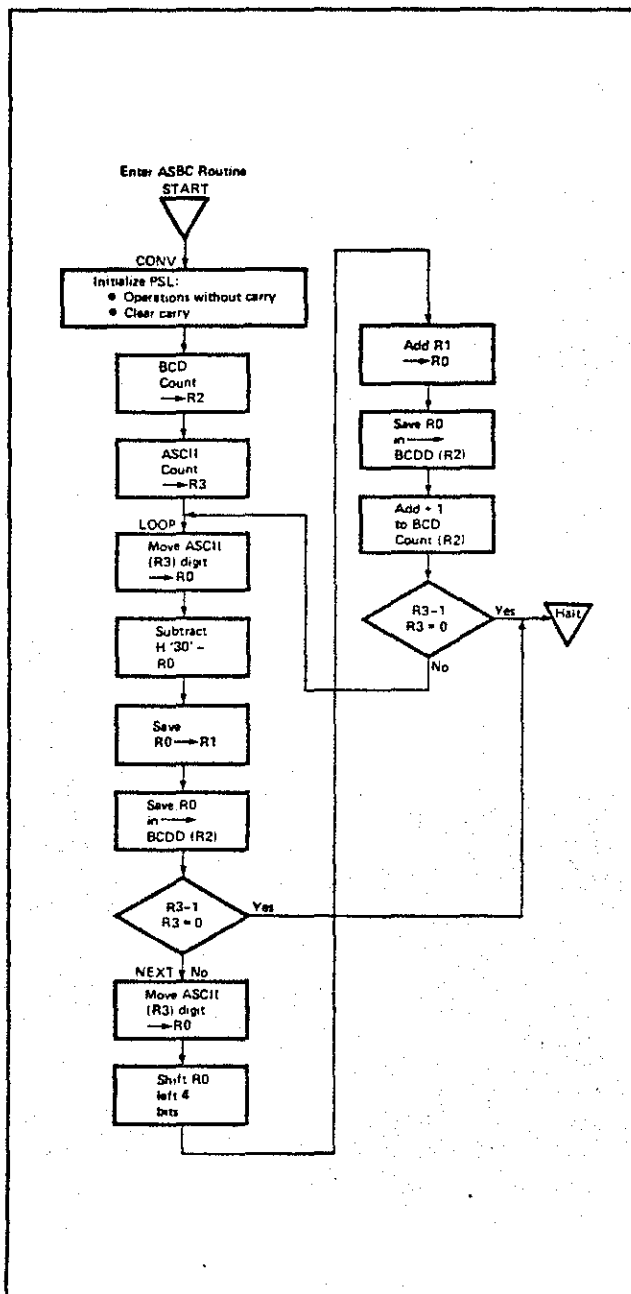


FIGURE 6-1 Flowchart for ASCII-to-BCD Conversion

```

1 * PD760055
2 *
3 * ASCII TO BCD CONVERSION *
4 *
5 *
6 * THIS ROUTINE CONVERTS A STRING OF ASCII
7 * DIGITS TO A STRING OF BCD DIGITS.
8 *
9 * ADIG IS MS DIGIT ASCII
10 * BCDD IS MS DIGIT BCD
11 *
12 * DEFINITIONS OF SYMBOLS:
13 R0 EQU 0 PROCESSOR-REGISTERS
14 R1 EQU 1
15 R2 EQU 2
16 R3 EQU 3
17 WC EQU H'00' PSL: 1-WITH, 0-WITHOUT
18 C EQU H'01' CARRY: BORROW
19 UN EQU 3 BR.COND: ALWAYS
20 *
21 * IN THIS EXAMPLE THE CONVERSION OF 5
22 * ASCII CHARACTERS IS PERFORMED.
23 *
24 NUM EQU 5
25 NUM1 EQU 3
26 *
27 ORG H'750' RAM DEFINITIONS
28 ADIG RES NUM ASCII BYTES RESERVED
29 ACNT EQU $-ADIG ASCII DIGIT COUNT
30 BCDD RES NUM1 BCD BYTES RESERVED
31 BCNT EQU $-BCDD BCD BYTE COUNT
32 *
33 ORG H'500' START OF SUBROUTINE
34 CONV CPSL WC+C ARITH.WITHOUT: NO CARRY
35 LODI,R2 BCNT BCD COUNT -> R2
36 LODI,R3 ACNT ASCII COUNT -> R3
37 LODA,R0 ADIG-1,R3 R0 HAS ASCII DIGIT
38 SUBI,R0 H'30' MAKE IT BCD
39 STRZ R1 R0 -> R1
40 STRA,R0 BCDD-1,R2 SAVE 1 BCD DIGIT
41 BDRR,R3 NEXT DECREMENT -NON ZERO BR
42 BCTA,UN BYTE CONVERSION COMPLETE
43 NEXT LODA,R0 ADIG-1,R3 NEXT ASCII DIGIT
44 SUBI,R0 H'30' MAKE IT BCD
45 RRL,R0 SHIFT LEFT 4 BITS
46 RRL,R0
47 RRL,R0
48 RRL,R0
49 IORZ R1 INCLUSIVE OR LOW ORDER
50 STRA,R0 BCDD-1,R2 STORE 2 BCD DIGITS
51 BDRR,R2 $+2 DECREMENT BCD COUNT
52 BDRR,R3 LOOP DECREMENT-NON ZERO BR.
53 BYE HALT END OF ASCII -> BCD
54 END
55

```

FIGURE 6-2 Program Listing for ASCII-to-BCD Conversion

## 7. HEXADECIMAL-TO-ASCII CONVERSION

## FUNCTION:

Converts a string of hexadecimal digits to a string of ASCII digits.

## PARAMETERS:

Input: HEX, HEX+1, ..., HEX + (n - 1)  
 HEX is the most-significant digit (2 Hex. digit/byte).

Output: ASCI, ASCI+1, ..., ASCI + (n - 1)  
 ASCI is the most-significant digit.

Refer to figures 7.1 and 7.2 for flowchart and program listing.

| HARDWARE AFFECTED |    |     |    |    |     |     |     |
|-------------------|----|-----|----|----|-----|-----|-----|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |
| PSU               | X  | X   | X  | X  |     |     |     |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |
|                   | X  | X   |    | X  | X   |     | X   |

|                                    |                              |
|------------------------------------|------------------------------|
| RAM REQUIRED (BYTES):              | $n\text{HEX} + n\text{ASCI}$ |
| ROM REQUIRED (BYTES):              | 59                           |
| EXECUTION TIME:                    | Variable                     |
| MAXIMUM SUBROUTINE NESTING LEVELS: | 0                            |
| ASSEMBLER/COMPILER USED:           | PIPHASM                      |

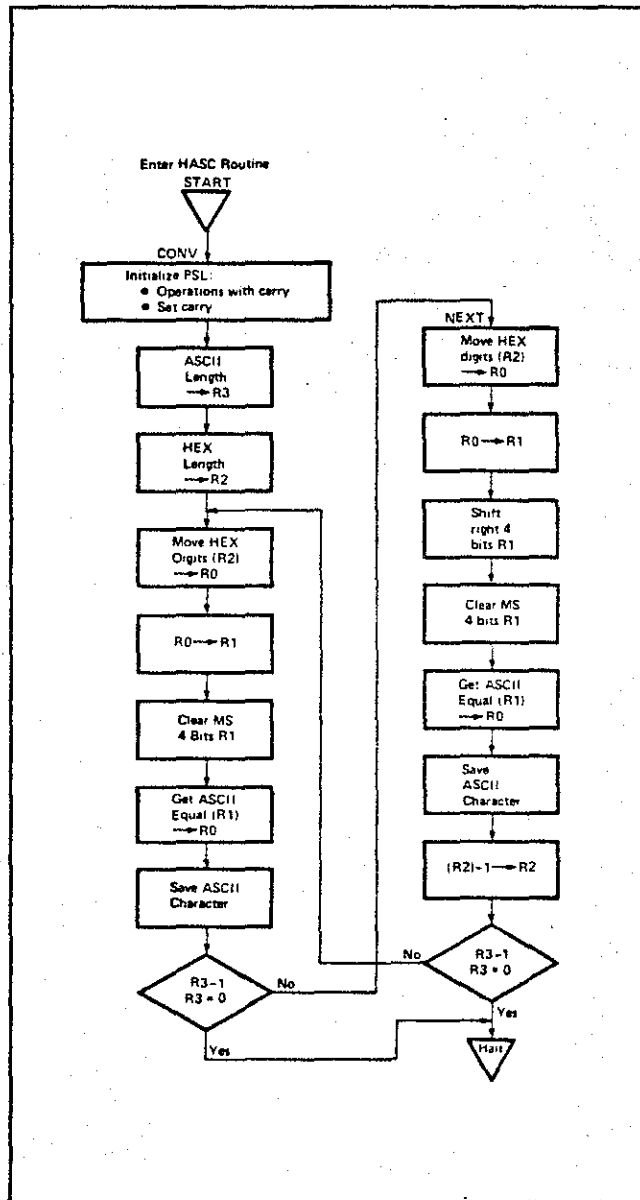


FIGURE 7-1 Flowchart for Hexadecimal-to-ASCII Conversion

```

1 * PD760056
2 *
3 * *****
4 * HEXIDECIMAL TO ASCII CONVERSION *
5 * *****
6 *
7 * THIS ROUTINE CONVERTS A STRING OF ASCII
8 * DIGITS TO A STRING OF HEX DIGITS.
9 *
10 * ASCII IS MS DIGIT ASCII.
11 * HEX IS MS DIGIT HEXIDECIMAL.
12 *
13 * DEFINITION OF SYMBOLS:
14 R0 EQU 0 PROCESSOR-REGISTER
15 R1 EQU 1
16 R2 EQU 2
17 R3 EQU 3
18 WC EQU H'0B' ARITHMETIC CARRY
19 C EQU H'01' CARRY-BORROW
20 UN EQU 3 UNCOND. BRANCH
21 *
22 * IN THIS EXAMPLE 3 HEXIDECIMAL
23 * CHARACTERS ARE CONVERTED.
24 NUM EQU 2 HEX BYTE COUNT
25 NUM1 EQU 3 ASCII BYTE COUNT
26 *
27 DRG H'600' RAM DEFINITIONS
28 HEX RES NUM RESERVES HEX BYTES
29 HLEN EQU $-HEX LENGTH OF HEX
30 ASCII RES NUM1 RESERVES ASCII BYTES
31 ALEN EQU $-ASCII LENGTH OF ASCII
32 *
33 ORG H'500' START OF ROUTINE
34 CONV PPSL WC+C ARITH.WITH, SET CARRY
35 LODI,R3 ALEN R3= ASCII LENGTH
36 LODI,R2 HLEN R2= HEX LENGTH
37 CHEX LODA,R0 HEX-1,R2 GET HEX DIGITS
38 STRZ R1 R0 -> R1
39 ANDI,R1 H'0F' CLEAR MS 4 BITS
40 LODA,R0 ANSI,R1 LOAD ASCII CORRESPOND
41 STRA,R0 ASCII-1,R3 SAVE IT
42 BDRR,R3 NEXT R3-1, R3(<) BRANCH
43 BCTA,UN BYTE END OF CONVERSION
44 NEXT LODA,R0 HEX-1,R2 GET HEX DIGITS
45 STRZ R1 R0 -> R1
46 RRR,R1 SHIFT RIGHT 4 BITS
47 RRR,R1
48 RRR,R1
49 RRR,R1
50 ANDI,R1 H'0F' CLEAR MS 4 BITS
51 LODA,R0 ANSI,R1 LOAD ASCII CORRESPOND
52 STRA,R0 ASCII-1,R3 SAVE IT
53 BDRR,R2 $+2 R2 - 1 CONT.
54 BDRR,R3 CHEX R3-1, R3(<) BRANCH
55 *
56 BYE HALT END OF CONVERSION
57 *
58 ANSI DATA A'0123456789ABCDEF'
59
60 END

```

FIGURE 7-2 Program Listing for Hexadecimal-to-ASCII Conversion

## 8. ASCII-TO-HEXADECIMAL CONVERSION

## FUNCTION:

Converts a string of ASCII digits to a string of hexadecimal digits. The conversion is done by table look-up. Non-numeric ASCII halts this routine. It may be changed to report non-numeric.

## PARAMETERS:

Input: ASCII, ASCII+1, ---, ASCII + (n - 1)  
ASCII is the most-significant digit.

Output: HEX, HEX+1, ---, HEX + (n - 1)  
HEX is the most-significant digit (2 Hex. digits/byte)

Refer to figures 8.1 and 8.2 for flowchart and program listing.

| HARDWARE AFFECTED |    |     |    |    |     |     |     |  |
|-------------------|----|-----|----|----|-----|-----|-----|--|
| REGISTERS         | R0 | R1  | R2 | R3 | R1' | R2' | R3' |  |
|                   | X  | X   | X  | X  |     |     |     |  |
| PSU               | F  | II  | SP |    |     |     |     |  |
| PSL               | CC | IDC | RS | WC | OVF | COM | C   |  |
|                   | X  | X   |    | X  | X   |     | X   |  |

RAM REQUIRED (BYTES): nASCII + nHEX

ROM REQUIRED (BYTES): 68

EXECUTION TIME: Variable

MAXIMUM SUBROUTINE

NESTING LEVELS: 1

ASSEMBLER/COMPILER USED: PIPHASM

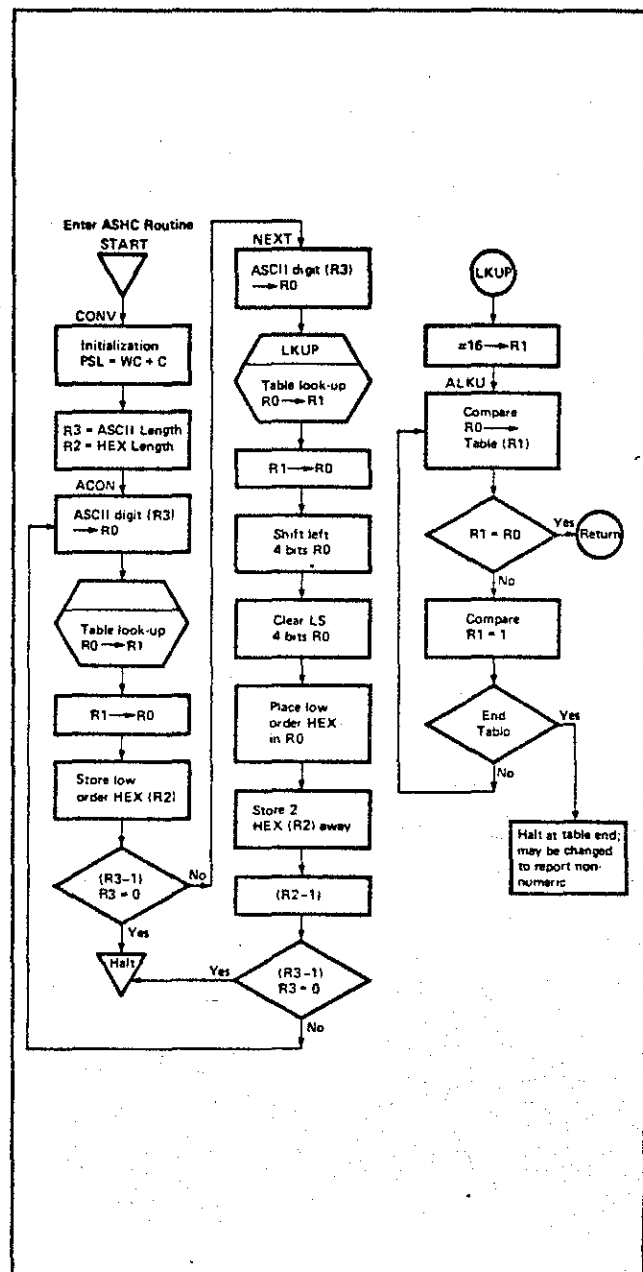


FIGURE 8-1 Flowchart for ASCII-to-Hexadecimal Conversion

```

1 * PD760057
2 *+++++*****
3 * ASCII TO HEX CONVERSION *
4 *+++++*****
5 *
6 * THIS ROUTINE CONVERTS A STRING OF ASCII
7 * DIGITS TO A STRING OF HEXIDECIMAL DIGITS
8 * ASCII IS MS DIGIT ASCII
9 * HEX IS MS DIGIT HEXIDECIMAL
10 * CONVERSION DONE BY TABLE LOOKUP
11 * NON NUMERIC ASCII HALT ROUTINE
12 *
13 * DEFINITION OF SYMBOLS:
14 R0 EQU 0 REGISTER-PROCESSOR
15 R1 EQU 1
16 R2 EQU 2
17 R3 EQU 3
18 MC EQU H'00' ARITHMETIC CARRY
19 C EQU H'01' CARRY-BORROW
20 UN EQU 3 BRANCH UNCOMB.
21 LT EQU 2 LESS THAN
22 EQ EQU 0 EQUAL
23 *
24 * IN THIS EXAMPLE 3 ASCII DIGITS
25 * ARE CONVERTED TO HEXIDECIMAL
26 *
27 NUM EQU 3 ASCII BYTE COUNT
28 NUM1 EQU 2 HEX BYTE COUNT
29 *
30 *
31 *
32 ORG H'600' RAM DEFINITIONS
33 ASCII RES NUM RESERVED ASCII BYTES
34 ALEN EQU 4-ASCII LENGTH OF ASCII
35 HEX RES NUM1 RESERVED HEX BYTES
36 HLEN EQU 4-HEX LENGTH OF HEX
37 *
38 ORG H'500' START OF ROUTINE
39 CONV PPSL MC+C ARITH.WITH + CARRY SET
40 LODI,R3 ALEN R3 = ASCII LENGTH
41 LODI,R2 HLEN R2 = HEX LENGTH
42 ACON LODA,R0 ASCII-1,R3 GET ASCII DIGIT
43 BSTR,UN LKUP LOOKUP SUBROUTINE
44 LOBZ R1 R1 -> R0
45 STRA,R0 HEX-1,R2 SAVE HEX CORRESPONDING
46 BDRR,R3 MCIT (A3-1), R3 (<) BRANCH
47 BCTR,UN BTE END OF CONVERSION
48 *
49 LKUP LODI,R1 16 LOOP CONSTANT
50 ALKU COMA,R0 ANSI,R1,- COMPARE TO TABLE
51 RETC,EB RETURN - MATCH FOUND
52 COMI,R1 1 TEST END OF TABLE
53 BCFR,LT ALKU NO- LOOK AGAIN
54 HALT ERROR - NON NUMERIC HE
55 *
56 ANSI DATA A'0123456789ABCDEF'
57 *
58 NEXT LODA,R0 ASCII-1,R3 GET NEXT ASCII DIGIT
59 BSTR,UN LKUP LOOK UP SUBROUTINE
60 LOBZ R1 R1 -> R0
61 RRL,R0 SHIFT LEFT 4 BITS
62 RRL,R0
63 RRL,R0
64 RRL,R0
65 ANDI,R0 H'F0' CLEAR LS 4 BITS
66 IORA,R0 HEX-1,R2 COMBINE LOW ORDER
67 STRA,R0 HEX-1,R2 SAVE 2 HEX DIGITS
68 BDRR,R2 4+2 (R2-1), CONTINUE
69 BDRR,R3 ACON (R3-1), R3 (<) BRANCH
70 *
71 BTE HALT ENB OF CONVERSION
72 END

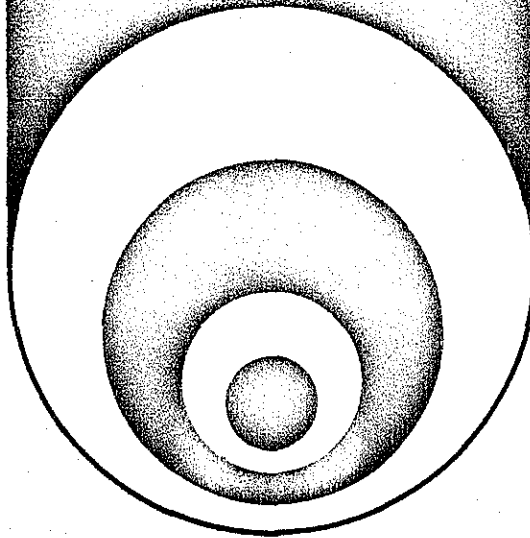
```

FIGURE 8-2 Program Listing for ASCII-to-Hexadecimal Conversion

**synetics**

**MOS**

**MICROPROCESSOR**



**ADDRESS AND DATA BUS  
INTERFACING TECHNIQUES . . . . . MP53**



## 1. INTRODUCTION

The Signetics 2650 Microprocessor has a 15-bit address bus and an 8-bit bi-directional data bus. The address bus allows a maximum of 32K words of memory. The drive capability of the 2650 address and data busses limits the number of chips that can be connected to the system. If the system load exceeds the 2650 drive capability, buffer circuits must be added.

This applications memo provides several examples of interfacing the 2650 address and data busses with ROMs and RAMs such as the 2608, 2606, and 2602. Examples are included for both small and large systems.

## 2. SMALL SYSTEMS WITHOUT BUFFERING

### Address Bus Loading

All 2650 output signals are TTL-compatible. Each output can source 100  $\mu$ A at 2.4V minimum and sink 1.6 mA at 0.45V maximum. The 2650 inputs require a load current of only 10  $\mu$ A regardless of the logic level on the input.

The 2608, 2606, 2604, and 2602 MOS ROMs and RAMs all require an input current of 10  $\mu$ A. This means that, based on d-c loading considerations, a maximum of ten inputs of this type can be driven from one 2650 address output without the use of buffering.

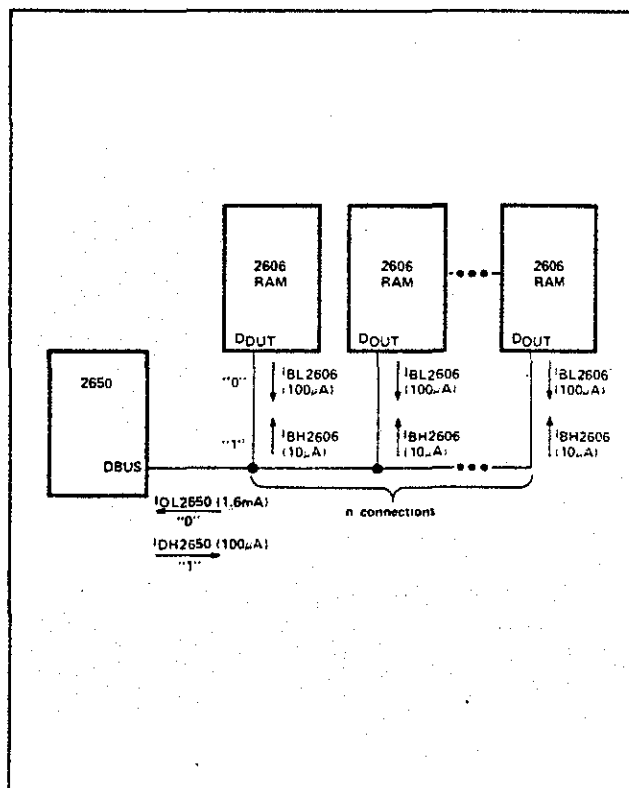
**TABLE I**  
**TYPICAL 2650 MEMORY CONFIGURATIONS**  
**WITHOUT BUFFERING**

| Number of Chips Connected to One Address Output        | Memory Capacity              |
|--------------------------------------------------------|------------------------------|
| Eight 2606 RAMs (256 x 4)<br>Two 2608 ROMs (1024 x 8)  | 1K byte RAM;<br>2K bytes ROM |
| Eight 2602 RAMs (1024 x 1)<br>Two 2608 ROMs (1024 x 8) | 1K byte RAM;<br>2K bytes ROM |

If bipolar PROMs such as the 82S114 or 82S115 are used, fewer chips can be connected because of higher input current requirements.

### Data Bus Loading with the 2606 RAM (256 x 4)

The bi-directional data bus of the 2606 RAM (256 x 4) makes this device ideally suited for use with the 2650 Microprocessor. The maximum number of input/output connections can be calculated from the diagram shown in Figure 1.



**FIGURE 1** The 2606 RAM with the 2650

In Figure 1, n 2606 memory chips are driven by the 2650. The 2606 memory chips load the bus with a leakage current of 100  $\mu$ A in the logic ZERO state and with 10  $\mu$ A in the logic ONE state. When the data bus is driven to a logic "1", the required source current of the 2650 output will be:

$$I_{OH2650} = (n) \cdot I_{BH2606} \\ = (n) \cdot (10 \mu A)$$

where:

$I_{BH2606}$  = output logic ONE leakage current of the 2606 RAM;

and

$I_{OH2650}$  = output logic ONE drive current of the 2650.

From this equation we calculate  $n_{max}$ :

$$n_{max} = \frac{I_{OH2650 \max}}{10 \mu A} = \frac{100 \mu A}{10 \mu A} = 10$$

In the logic ZERO state, the output current required of the 2650 is:

$$\begin{aligned} I_{OL2650} &= (n) \cdot I_{BL2606} \\ &= (10) \cdot (100 \mu A) = 1000 \mu A \end{aligned}$$

where:

$I_{BL2606}$  = output logic ZERO leakage current of the 2606 RAM;

and

$I_{OL2650}$  = output logic ZERO drive current of the 2650.

This is less than the maximum drive capability of 1.6 mA for the 2650.

When the 2606 drives the data bus, the logic ONE loading is the same as that seen by a 2650 driving a data bus (previously described as  $I_{OH2650}$ ). The logic ZERO load on the 2606 chip is:

$$\begin{aligned} I_{OL2606} &= (n-1) I_{BL2606} + I_{LOL2650} \\ &= [(9) \cdot (100 \mu A)] + 10 \mu A \\ &= 910 \mu A \end{aligned}$$

where:

$I_{LOL2650}$  = output logic ZERO leakage current of the 2650.

This is below the 1.9 mA sink current capability of the 2606. It can thus be concluded that when using MOS RAMs or ROMs with the 2650, the number  $n$  is normally limited by the maximum output logic ONE current of the driving device.

### Data Bus Loading with the 2602 RAM

In contrast to the 2606, the 2602 RAM (1024 x 1) has separate input and output data paths. The data output for this device is switched to tri-state with the chip enable input. For bi-directional data transfers, however, the data output signal must be disabled during the write mode to avoid a drive conflict between the 2650 and the RAM. This is done by inserting a tri-state buffer into the data-out line as shown in Figure 2. The buffers are only enabled when  $OPREQ$  is a "HIGH",  $\bar{R}/W$  (the READ/WRITE control line from the 2650) is a "LOW", and the RAM is selected for access.

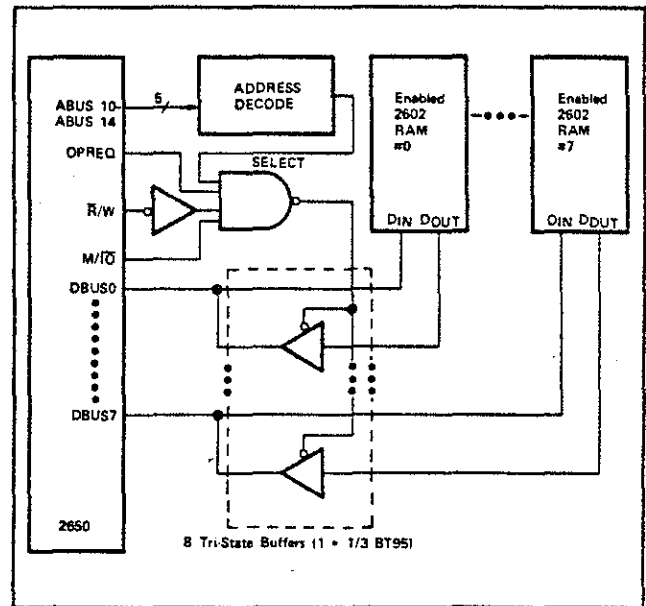


FIGURE 2 The 2602 RAM with the 2650

### A-C Loading Considerations

The 2650 address bus, data bus, and control lines will drive a 100 pF capacitive load and one standard TTL load. The capacitive loading calculations must include the 2650 output capacitance and the external wiring capacitance. The 2606 presents a 10 pF capacitive load to the data bus and a 7 pF load to all other inputs. The number ( $n$ ) of 2606 RAMs that can be driven directly by the 2650 is given by the following equations:

$$C_{LOAD} = C_{OUT2650} + C_{WIRING} + [(n_d) \cdot C_{OUT2606}]$$

or

$$C_{LOAD} = C_{OUT2650} + C_{WIRING} + [(n_a) \cdot C_{IN2606}]$$

where:

$C_{OUT2650}$  = Output capacitance for the 2650  
= 10 pF

$C_{WIRING}$  = Wiring capacitance  
= 10 pF

$C_{IN2606}$  = Load capacitance for the 2606 address bus  
= 7 pF

$C_{OUT2606}$  = Load capacitance for the 2606 data bus  
= 10 pF

$C_{LOAD}$  = 100 pF

therefore:

$$n_a = \frac{80 \text{ pF}}{7 \text{ pF}} \approx 11 \text{ address bus loads}$$

$$n_d = \frac{80 \text{ pF}}{10 \text{ pF}} \approx 8 \text{ data bus loads}$$

The 2606 is a 256-location by 4-bit RAM and requires two chips for each 256 bytes. As seen from the above calcula-

tions, the 2650 will drive eleven 2606s ( $n_a$ ), or five pairs ( $n_a/2$ ) of 2606s (1280 bytes) directly. Since this number is less than the number of d-c loads that the 2650 is capable of driving (10), it can be concluded that the a-c loading is the limitation for full-speed operation.

### Increasing Fan-Out by Pull-Up Resistor

The fan-out of the 2650 bus in the logic ONE state can be increased with a pull-up resistor. This increases the d-c fan-out of the outputs in the logic ONE state by supplying supplementary drive current. This can be seen from the example shown in Figure 3.

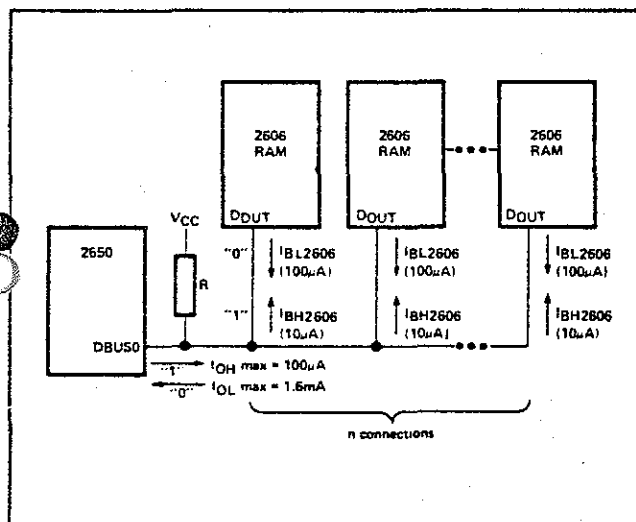


FIGURE 3 Pull-up Resistors for Increased Fan-out

$$\text{Logic ONE state } I_R = \frac{V_{CCmin} - V_{OHmax2650}}{R} = \frac{[(n) \cdot I_{BH2606}] - I_{OH2650}}{R}$$

$$\text{Logic ZERO state } I_R = \frac{V_{CCmax} - V_{OLmin2650}}{R} = \frac{I_{OL2650} - [(n) \cdot I_{BL2606}]}{R}$$

where:

$$V_{CCmin} = 4.75 \text{ volts}$$

$$V_{CCmax} = 5.25 \text{ volts}$$

$$V_{OHmax2650} = 2650 \text{ maximum logic ONE output voltage}$$

$$= V_{CC} - 0.5 \text{ volts}$$

$$V_{OLmin2650} = 2650 \text{ minimum logic ZERO output voltage}$$

$$= 0 \text{ volts}$$

$$I_{BH2606} = \text{output logic ONE leakage current of the 2606 RAM}$$

$$= 10 \mu A$$

$$I_{BL2606} = \text{output logic ZERO leakage current of the 2606 RAM}$$

$$= 100 \mu A$$

$$I_{OH2650} = \text{output logic ONE current of the 2650}$$

$$= 100 \mu A$$

$$I_{OL2650} = \text{output logic ZERO current of the 2650}$$

$$= 1.6 \text{ mA}$$

$n$  = the number of 2606 type loads that can be driven by the 2650

From the above equations,  $R$  can be calculated to be 17.5K ohms. The number of 2606 loads ( $n$ ) is calculated to be 12. Six pairs of 2606 chips can be driven when the pull-ups are added. These calculations are for d-c loading, and the a-c (capacitive) load limitations must still be considered.

$$\text{With } V_{CCmin} = 4.5V \text{ and } V_{CCmax} = 5.5V: n = 10$$

$$R = 9K\Omega$$

$$\text{With } V_{CCmin} = 4.75V \text{ and } V_{CCmax} = 5.25V: n = 12$$

$$R = 12K\Omega$$

### 3. LARGE BUFFERED SYSTEMS

In larger microcomputers it is necessary to increase the drive capability of the CPU by adding drivers to the outputs. A generalized 2650 microcomputer system using additional bus drivers is illustrated in Figure 4.

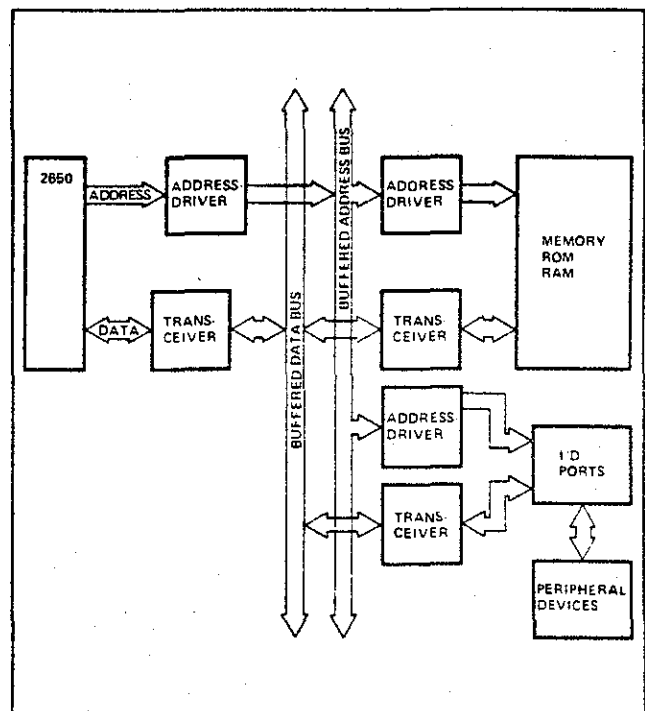


FIGURE 4 General-Purpose Microcomputer System

This system has a buffered address bus and a buffered data bus. To ensure minimal loading, buffers are also included between the memory and I/O ports. With this arrangement, the system can easily be expanded, and each additional device adds a single load to the shared bus.

In some cases, the configuration in Figure 4 can be simplified as shown in Figure 5. The memory and I/O ports are directly driven by the address driver and transceiver circuits.

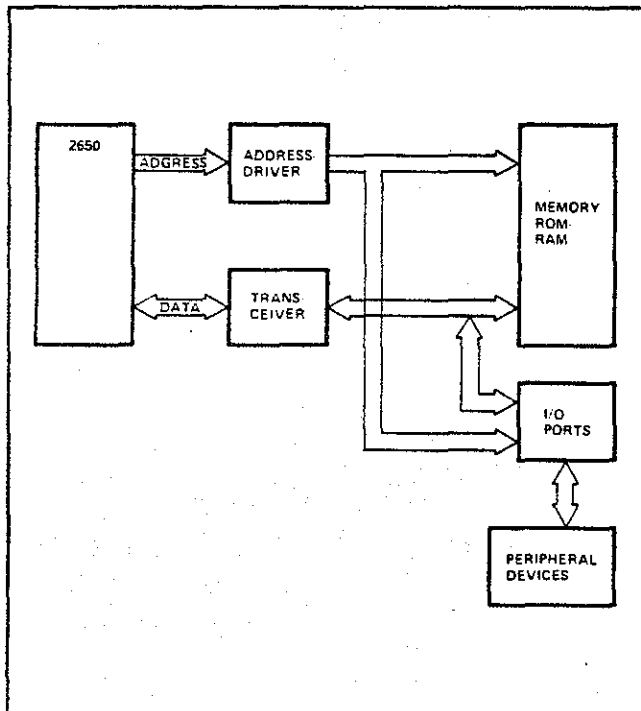


FIGURE 5 Microprocessor with Buffered Address and Data Bus

### Address Driver

The address bus driver may be a non-inverting interface element of the 8T family, such as the 8T95 or 8T97 shown in Figure 6.

The tri-state control inputs (DIS4 and DIS2) can be connected to ground if these buffers are always active. For DMA operations, the control inputs can be switched to a HIGH to disconnect the processor from the bus. These Schottky-TTL devices have typical propagation delays of 6 ns. (See Table III.)

Standard TTL buffers may be used to drive the address bus. If buffers with open-collector outputs or tri-state capability are used, DMA operations can be performed.

### Data Transceivers

The 2650 bi-directional data lines can be driven with the 8T26 (inverting) and 8T28 (non-inverting) transceivers (Figure 7).

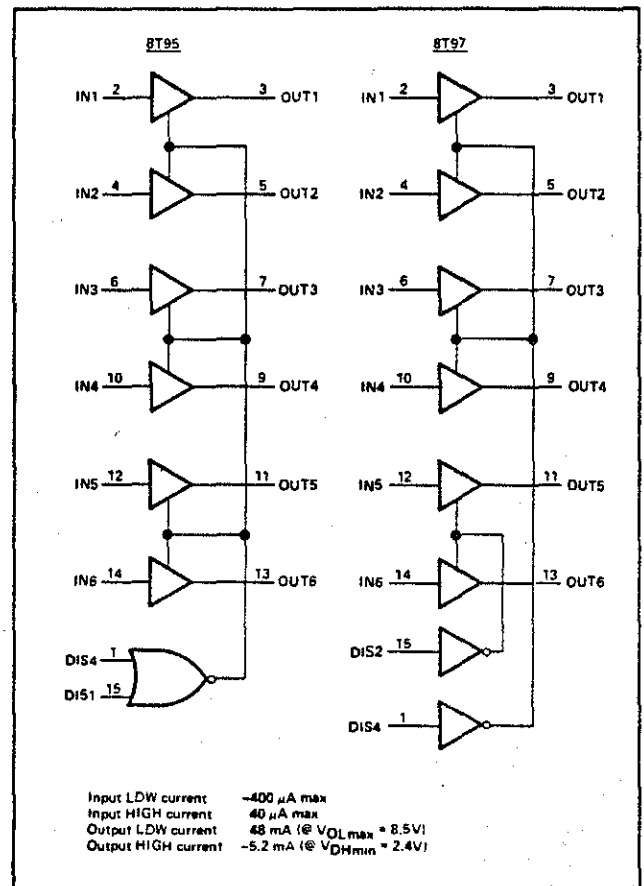


FIGURE 6 8T95 and 8T97 Hex Tri-State Buffers

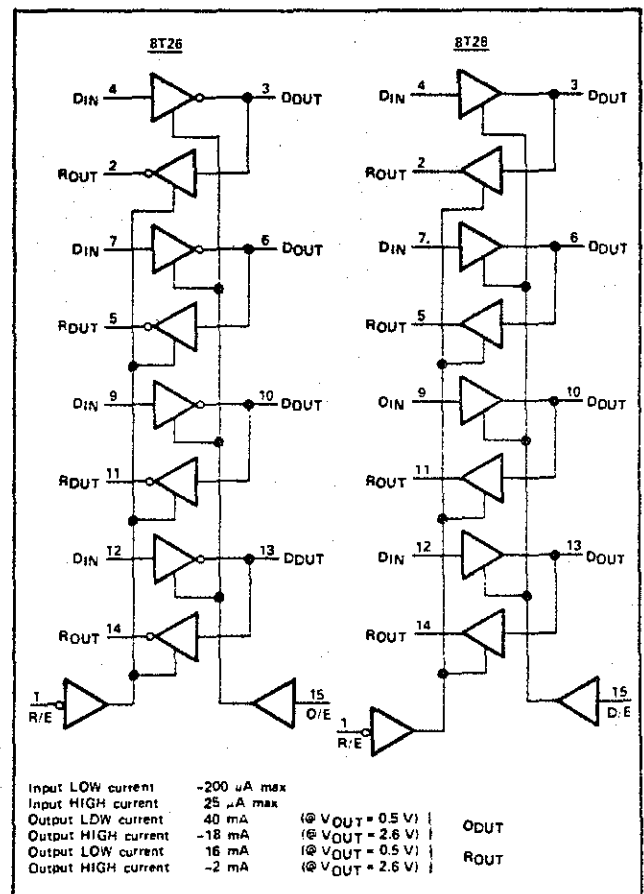


FIGURE 7 Tri-State Quad Bus Transceivers

The driver can be enabled by the driver enable line (D/E, active high). The receiver can be enabled by the receiver enable line (R/E, active low). To drive the 2650 bi-directional data bus, the DIN and ROUT signals can be tied together to provide a bi-directional data path.

Figure 8 shows a typical application of the transceiver circuit for bi-directional data buffering. The 8T28 features a propagation delay of 20 ns with a 300 pF capacitive load.

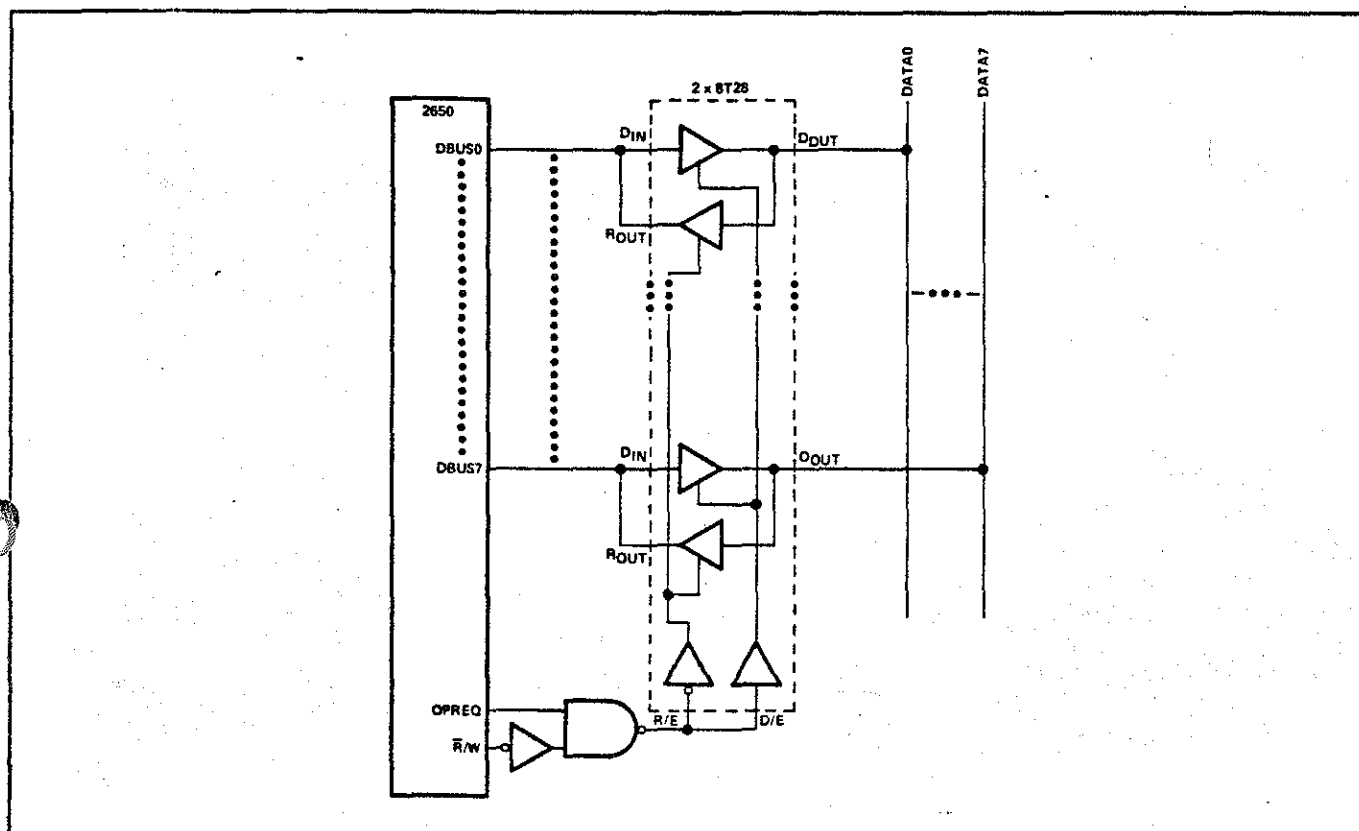


FIGURE 8 Typical Application of the Transceiver Circuit

TABLE II  
MOS RAMs - SURVEY OF D-C ELECTRICAL CHARACTERISTICS

| PARAMETER                           | SYMBOL    | 2606<br>(256 x 4) | 2602<br>(1024 x 1) | 2604<br>(4096 x 1) | UNIT    | TEST CONDITIONS                       |
|-------------------------------------|-----------|-------------------|--------------------|--------------------|---------|---------------------------------------|
| Maximum input load current          | $I_{IL}$  | 10                | 10                 |                    | $\mu A$ | $V_{IN} = 0$ to 5.25V                 |
|                                     |           |                   |                    | 10                 | $\mu A$ | $V_{IN} = +5V$                        |
| Maximum input LOW voltage           | $V_{IL}$  | 0.65              | 0.65               | 0.6                | V       |                                       |
| Minimum input HIGH voltage          | $V_{IH}$  | 2.2               | 2.2                | 2.2                | V       |                                       |
| Maximum output LOW voltage          | $V_{OL}$  | 0.45              | 0.45               |                    | V       | $I_{OL} = 1.9$ mA                     |
|                                     |           |                   |                    | 0.4                | V       | $I_{OL} = 3.2$ mA                     |
| Minimum output HIGH voltage         | $V_{OH}$  | 2.4               | 2.4                |                    | V       | $I_{OH} = -100$ $\mu A$               |
|                                     |           |                   |                    | 2.4                | V       | $I_{OH} = -2$ mA                      |
| Maximum output HIGH leakage current | $I_{BH}$  | 10                | 10                 | 10*                | $\mu A$ | $\bar{C}E = 2.2V$ ; $V_{OUT} = 4.0V$  |
| Maximum output LOW leakage current  | $I_{BL}$  | -100              | -100               |                    | $\mu A$ | $\bar{C}E = 2.2V$ ; $V_{OUT} = 0.45V$ |
| Maximum input capacitance           | $C_{IN}$  | 7                 | 5                  | 7                  | pF      | $V_{IN} = 0V$                         |
| Maximum bus input capacitance       | $C_{OUT}$ | 10                | 10                 | 6                  | pF      | $V_{OUT} = 0V$                        |
| Common I/O                          |           | X                 |                    |                    |         |                                       |
| Separate I/O                        |           |                   | X                  | X                  |         |                                       |

\*Test conditions  $\bar{C}S = 2.2V$ ;  $V_{OUT} = 5V$

**TABLE III**  
**BUFFERS - SURVEY OF ELECTRICAL CHARACTERISTICS**

| PARAMETER                             | SYMBOL      | 8T09<br>(quad) | 8T95/97<br>(hex) | 8T96/98<br>(hex) | UNIT    | TEST CONDITIONS                     |
|---------------------------------------|-------------|----------------|------------------|------------------|---------|-------------------------------------|
| Inverting                             |             | X              |                  | X                |         |                                     |
| Non-inverting                         |             |                | X                |                  |         |                                     |
| Maximum input LOW current             | $I_{ILmax}$ | -2             |                  |                  | mA      | $V_I = 0.4V$ ; $DIS = 0.4V$         |
|                                       |             |                | -0.4             | -0.4             | mA      | $V_I = 0.5V$ ; $DIS = 0.5V$         |
| Maximum input HIGH current            | $I_{IHmax}$ | 40             |                  |                  | $\mu A$ | $DIS = 4.5V$                        |
|                                       |             |                | 40               | 40               | $\mu A$ | $V_I = 2.4V$                        |
| Maximum input LOW voltage             | $V_{ILmax}$ | 0.8            | 0.8              | 0.8              | V       | $V_{CC} = MIN$ ; $T_A = 25^\circ C$ |
| Minimum input HIGH voltage            | $V_{IHmin}$ | 2.0            | 2.0              | 2.0              | V       | $V_{CC} = MIN$ ; $T_A = 25^\circ C$ |
| Maximum output LOW voltage            | $V_{OLmax}$ | 0.4            |                  |                  | V       | $I_{OL} = 40 mA$                    |
|                                       |             |                | 0.5              | 0.5              | V       | $I_{OL} = 48 mA$                    |
| Minimum output HIGH voltage           | $V_{OHmin}$ | 2.4            | 2.4              | 2.4              | V       | $I_{OH} = -5.2 mA$                  |
| Maximum output leakage current HIGH   | $I_{BH}$    | 40             | 40               | 40               | $\mu A$ | $V_O = 2.4V$                        |
| Maximum output leakage current LOW    | $I_{BL}$    | -40            | -40*             | -40*             | $\mu A$ | $V_O = 0.4V$                        |
| Propagation delay (data to output)    | $t_{ON}$    | 20**           | 5***             | 5***             | ns      |                                     |
|                                       | $t_{OFF}$   | 20             | 6                | 6                | ns      |                                     |
| Propagation delay (disable to output) | High Z/0    | 22             | 12               | 12               | ns      |                                     |
|                                       | High Z/1    | 22             | 10               | 10               | ns      |                                     |

\*Test condition  $V_O = 0.5V$ \*\*Test condition  $C_L = 300 pF$ \*\*\*Test condition  $C_L = 50 pF$ 

**TABLE IV**  
**(P)ROMs - SURVEY OF D-C ELECTRICAL CHARACTERISTICS**

| PARAMETER                        | SYMBOL      | 2608<br>(1024 x 8) | 82S114<br>(256 x 8)<br>82S115<br>(512 x 8) | 82S130<br>(512 x 4)<br>82S131<br>(512 x 4) | 82S126<br>(256 x 4)<br>82S129<br>(256 x 4) | UNIT    | TEST CONDITIONS                                   |
|----------------------------------|-------------|--------------------|--------------------------------------------|--------------------------------------------|--------------------------------------------|---------|---------------------------------------------------|
| Maximum input load current       | $I_{IL}$    | 10                 |                                            |                                            |                                            | $\mu A$ |                                                   |
| Maximum input LOW current        | $I_{ILmax}$ | 10                 | -100                                       | -100                                       | -100                                       | $\mu A$ | $V_{IN} = 0.45V$                                  |
| Maximum input HIGH current       | $I_{IHmax}$ | 10                 | 25                                         | 40                                         | 40                                         | $\mu A$ | $V_{IN} = 5.5V$                                   |
| Maximum input LOW voltage        | $V_{ILmax}$ | 0.65               | 0.85                                       | .85                                        | 0.85                                       | V       |                                                   |
| Minimum input HIGH voltage       | $V_{IHmin}$ | 2.2                | 2.0                                        | 2.0                                        | 2.0                                        | V       |                                                   |
| Maximum output LOW voltage       | $V_{OLmax}$ | 0.45               |                                            |                                            |                                            | V       | $I_{OL} = 1.6 mA$ (2608)                          |
|                                  |             |                    | 0.5                                        |                                            |                                            | V       | $I_{OL} = 9.6 mA$ (82S114, 82S115)                |
|                                  |             |                    |                                            | 0.45                                       | 0.5                                        | V       | $I_{OL} = 16 mA$ (82S130, 82S131, 82S126, 82S129) |
| Minimum output HIGH voltage      | $V_{OHmin}$ | 2.4                |                                            |                                            |                                            | V       | $I_{OH} = -100 \mu A$                             |
|                                  |             |                    | 2.7                                        |                                            |                                            | V       | $I_{OH} = -2 mA$                                  |
|                                  |             |                    |                                            | 2.4                                        | 2.4                                        | V       | $I_{OH} = -2.4 mA$                                |
| Maximum output leakage current   | $I_{BH}$    | 10*                | 40                                         | 40                                         | 40                                         | $\mu A$ | $V_O = 5.5V$ ; device deselected                  |
| Maximum output leakage current L | $I_{BH}$    | -10**              | -40                                        | -40                                        | -40                                        | $\mu A$ | $V_O = 0.5V$ ; device deselected                  |
| Maximum input capacitance        | $C_{IN}$    | 7.5                | 5                                          | 5                                          | 5                                          | pF      |                                                   |
| Maximum output capacitance       | $C_{OUT}$   | 15                 | 8                                          | 8                                          | 8                                          | pF      |                                                   |

\*Test conditions  $V_O = 2.4V$ \*\*Test conditions  $V_O = 0.4V$

**TABLE III**  
**BUFFERS - SURVEY OF ELECTRICAL CHARACTERISTICS**

| PARAMETER                                | SYMBOL      | 8T09<br>(quad) | 8T95/97<br>(hex) | 8T96/98<br>(hex) | UNIT    | TEST CONDITIONS                     |
|------------------------------------------|-------------|----------------|------------------|------------------|---------|-------------------------------------|
| Inverting                                |             | X              |                  | X                |         |                                     |
| Non-inverting                            |             |                | X                |                  |         |                                     |
| Maximum input LOW current                | $I_{ILmax}$ | -2             |                  |                  | mA      | $V_I = 0.4V$ ; $DIS = 0.4V$         |
|                                          |             |                | -0.4             | -0.4             | mA      | $V_I = 0.5V$ ; $DIS = 0.5V$         |
| Maximum input HIGH current               | $I_{IHmax}$ | 40             |                  |                  | $\mu A$ | $DIS = 4.5V$                        |
|                                          |             |                | 40               | 40               | $\mu A$ | $V_I = 2.4V$                        |
| Maximum input LOW voltage                | $V_{ILmax}$ | 0.8            | 0.8              | 0.8              | V       | $V_{CC} = MIN$ ; $T_A = 25^\circ C$ |
| Minimum input HIGH voltage               | $V_{IHmin}$ | 2.0            | 2.0              | 2.0              | V       | $V_{CC} = MIN$ ; $T_A = 25^\circ C$ |
| Maximum output LOW voltage               | $V_{OLmax}$ | 0.4            |                  |                  | V       | $I_{OL} = 40\text{ mA}$             |
|                                          |             |                | 0.5              | 0.5              | V       | $I_{OL} = 48\text{ mA}$             |
| Minimum output HIGH voltage              | $V_{OHmin}$ | 2.4            | 2.4              | 2.4              | V       | $I_{OH} = -5.2\text{ mA}$           |
| Maximum output leakage current HIGH      | $I_{BH}$    | 40             | 40               | 40               | $\mu A$ | $V_O = 2.4V$                        |
| Maximum output leakage current LOW       | $I_{BL}$    | -40            | -40*             | -40*             | $\mu A$ | $V_O = 0.4V$                        |
| Propagation delay<br>(data to output)    | $t_{ON}$    | 20**           | 5***             | 5***             | ns      |                                     |
|                                          | $t_{OFF}$   | 20             | 6                | 6                | ns      |                                     |
| Propagation delay<br>(disable to output) | High Z/0    | 22             | 12               | 12               | ns      |                                     |
|                                          | High Z/1    | 22             | 10               | 10               | ns      |                                     |

\*Test condition  $V_O = 0.5V$ \*\*Test condition  $C_L = 300\text{ pF}$ \*\*\*Test condition  $C_L = 50\text{ pF}$ 

**TABLE IV**  
**(P)ROMs - SURVEY OF D-C ELECTRICAL CHARACTERISTICS**

| PARAMETER                        | SYMBOL      | 2608<br>(1024 x 8) | 82S114<br>(256 x 8)<br>82S115<br>(512 x 8) | 82S130<br>(512 x 4)<br>82S131<br>(512 x 4) | 82S126<br>(256 x 4)<br>82S129<br>(256 x 4) | UNIT    | TEST CONDITIONS                                          |
|----------------------------------|-------------|--------------------|--------------------------------------------|--------------------------------------------|--------------------------------------------|---------|----------------------------------------------------------|
| Maximum input load current       | $I_{IL}$    | 10                 |                                            |                                            |                                            | $\mu A$ |                                                          |
| Maximum input LOW current        | $I_{ILmax}$ | 10                 | -100                                       | -100                                       | -100                                       | $\mu A$ | $V_{IN} = 0.45V$                                         |
| Maximum input HIGH current       | $I_{IHmax}$ | 10                 | 25                                         | 40                                         | 40                                         | $\mu A$ | $V_{IN} = 5.5V$                                          |
| Maximum input LOW voltage        | $V_{ILmax}$ | 0.65               | 0.85                                       | .85                                        | 0.85                                       | V       |                                                          |
| Minimum input HIGH voltage       | $V_{IHmin}$ | 2.2                | 2.0                                        | 2.0                                        | 2.0                                        | V       |                                                          |
| Maximum output LOW voltage       | $V_{OLmax}$ | 0.45               |                                            |                                            |                                            | V       | $I_{OL} = 1.6\text{ mA}$ (2608)                          |
|                                  |             |                    | 0.5                                        |                                            |                                            | V       | $I_{OL} = 9.6\text{ mA}$ (82S114, 82S115)                |
|                                  |             |                    |                                            | 0.45                                       | 0.5                                        | V       | $I_{OL} = 16\text{ mA}$ (82S130, 82S131, 82S126, 82S129) |
| Minimum output HIGH voltage      | $V_{OHmin}$ | 2.4                |                                            |                                            |                                            | V       | $I_{OH} = -100\text{ }\mu A$                             |
|                                  |             |                    | 2.7                                        |                                            |                                            | V       | $I_{OH} = -2\text{ mA}$                                  |
|                                  |             |                    |                                            | 2.4                                        | 2.4                                        | V       | $I_{OH} = -2.4\text{ mA}$                                |
| Maximum output leakage current   | $I_{BH}$    | 10*                | 40                                         | 40                                         | 40                                         | $\mu A$ | $V_O = 5.5V$ ; device deselected                         |
| Maximum output leakage current L | $I_{BH}$    | -10**              | -40                                        | -40                                        | -40                                        | $\mu A$ | $V_O = 0.5V$ ; device deselected                         |
| Maximum input capacitance        | $C_{IN}$    | 7.5                | 5                                          | 5                                          | 5                                          | pF      |                                                          |
| Maximum output capacitance       | $C_{OUT}$   | 15                 | 8                                          | 8                                          | 8                                          | pF      |                                                          |

\*Test conditions  $V_O = 2.4V$ \*\*Test conditions  $V_O = 0.4V$

**Signetics 2650 Microprocessor application memos currently available:**

|      |                                                                                |
|------|--------------------------------------------------------------------------------|
| AS50 | Serial Input/Output                                                            |
| AS51 | Bit and Byte Testing Procedures                                                |
| AS52 | General Delay Routines                                                         |
| AS53 | Binary Arithmetic Routines                                                     |
| AS54 | Conversion Routines                                                            |
| SP50 | 2650 Evaluation Printed Circuit Board Level System (PC1001)                    |
| SP51 | 2650 Demo Systems                                                              |
| SP52 | Support Software for use with NCSS Timesharing System                          |
| SP53 | Simulator, Version 1.2                                                         |
| SP54 | Support Software for use with the General Electric Mark III Timesharing System |
| SS50 | PIPBUG                                                                         |
| SS51 | Absolute Object Format (Revision 1)                                            |
| MP51 | 2650 Initialization                                                            |
| MP52 | Low Cost Clock Generator Circuits                                              |
| MP53 | Address and Data Bus Interfacing Techniques                                    |

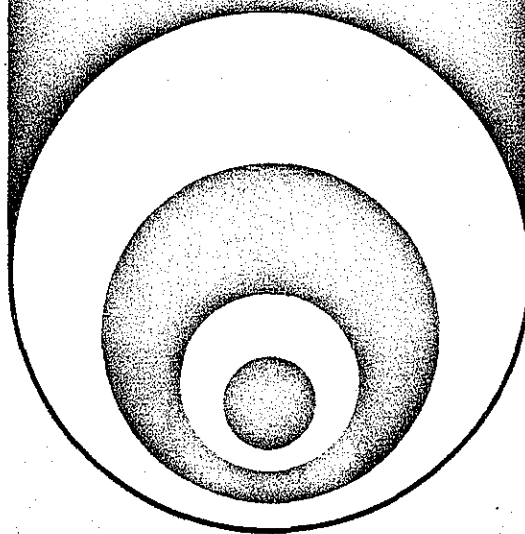
**Signetics**  
a subsidiary of U.S. Philips Corporation

Signetics Corporation  
811 East Arques Avenue  
Sunnyvale, California 94086  
Telephone 408/739-7700



**synetics**

**MICROPROCESSOR**



SERIAL INPUT/OUTPUT.....AS50

## 2650 MICROPROCESSOR APPLICATION MEMO

### INTRODUCTION

The Sense/Flag capability of the Signetics 2650 microprocessor can be used for serial I/O interfaces. The Sense input pin is directly connected to a bit in the microprocessor's Program Status Word. A high level on the Sense pin appears as a binary one while a low level appears as a binary zero. The Sense bit in the PSW can be stored or tested by the program. The Flag bit in the PSW is a simple latch that drives the Flag output pin. A program can set the Flag bit to a binary one, which causes a high level, one TTL load on the flag output pin. Setting the Flag bit to binary zero causes a low level on the Flag output pin.

### APPLICATIONS

The most common use for the Sense/Flag capability would be in interfacing to a keyboard based terminal where the data is received or transmitted as bit serial. All bit manipulation and timings such as 8-bit serial-to-parallel conversion can be done by software running on the 2650. The software works by storing or setting the two bits in the Program Status Word which reflect or control the levels at the pins of the chip. External hardware is required simply to interface with line levels. No clock synchronization or address decoding hardware is necessary, since the Sense and Flag pins are independent of the normal I/O bus structure.

Two examples of device interfaces and software are given below; for a 1200 baud RS232-type CRT terminal and for a 110 baud Teletype. Figure 1 shows the RS232 interface. Half of the Signetics 8T15 dual line driver is used to transmit to the terminal from the Flag pin, while half of a

Signetics 8T16 dual line receiver is used to receive from the device into the Sense pin. The interface to a Teletype model 33 is shown in Figure II. A TTL open collector gate is used to provide the 20 milli-amp loop to the TTY

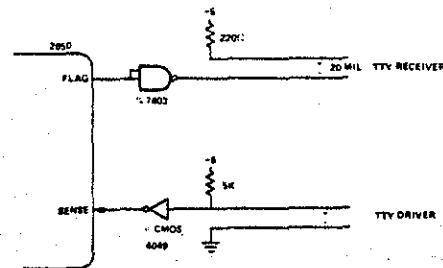


FIGURE II  
TTY MODEL 33 INTERFACE FULL DUPLEX

receiver. For receiving from the TTY a CMOS gate is used to provide the necessary noise immunity.

### SOFTWARE

All definitions of baud rate, character formats, and line characteristics are done in the software. For these examples, communication is asynchronous bit-serial over a full duplex line. Figure III shows the format of a 8-bit character (seven bits plus parity) headed by a start bit and followed by stop bits. The line levels are:

- low = start bit or binary zero
- high = stop bit or binary one



FIGURE III

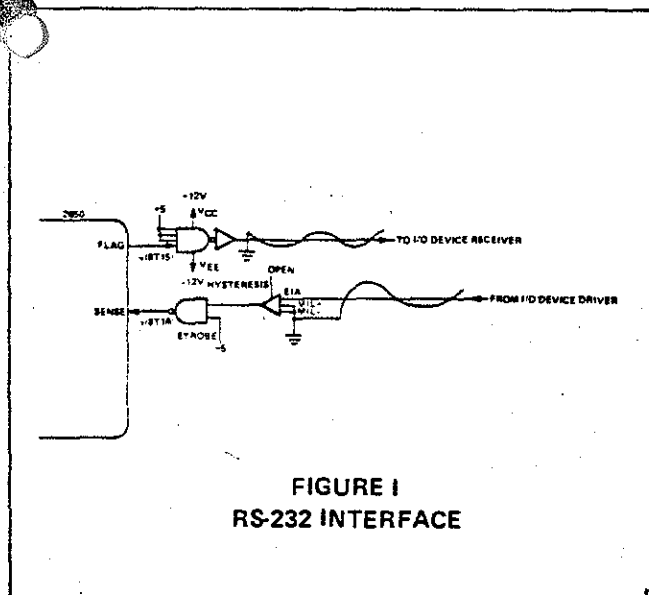
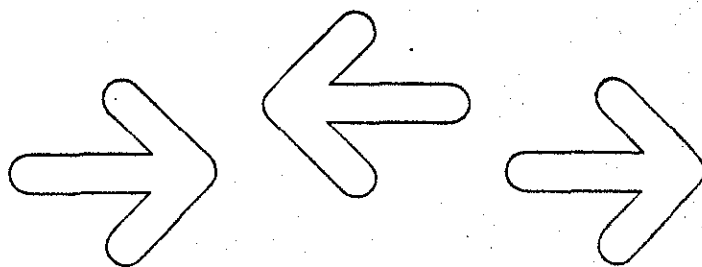
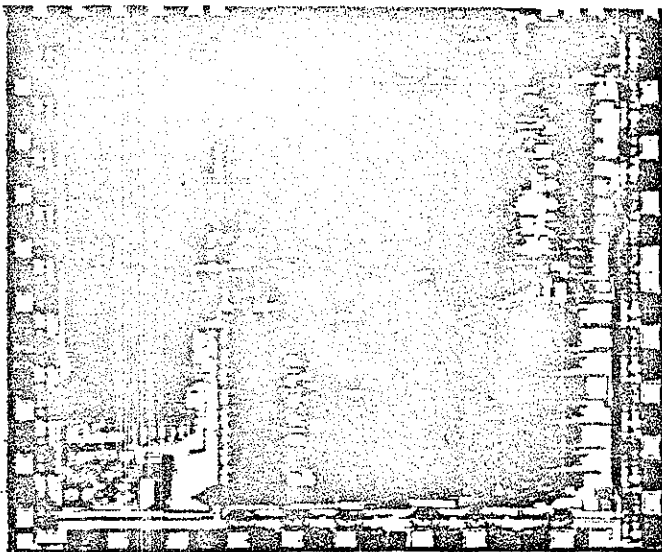


FIGURE I  
RS-232 INTERFACE



# INTERFACE DESIGN WITH

Interfacing a microprocessor to peripheral devices is an important part of a total microcomputer system design. The characteristics of the interface depend to a large extent on total system requirements and other factors such as CPU loading and data speed. The use of interrupts and/or DMA structures also have an impact on the system input/output structure. The design of an I/O interface is not limited to hardware, and hardware/software trade-offs must be considered.

This article examines the use of the 2650's set of I/O instructions and the interface between the 2650 and I/O ports. Interrupt and DMA-controlled I/O are not discussed. A number of application examples for both serial and parallel I/O are given. Several types of input, output, and bidirectional interface devices are also examined.

## BASIC I/O STRUCTURE

The 2650 is equipped with input and output facilities which can perform both single bit input/output and 8-bit parallel input/output.

The single bit input and output, called Sense (pin 1) and Flag (pin 40), are associated with the Program Status Word Upper (PSWU). The Flag output always reflects the value of bit 6 of the PSWU, while bit 7 of the PSWU always reflects the value of the Sense input signal. The Sense and Flag signals can be monitored and controlled with the PSW instructions.

Parallel I/O can be accomplished using the extended or non-extended read and write instructions. The extended and non-extended types are distinguished by the state of the E/NE output of the microprocessor.

The non-extended I/O instructions are single-byte instructions which accomplish a 1-byte data transfer into or out of the 2650. They also control the state of the D/C output, which can be used as a 1-bit device address in small systems.

The extended I/O instructions are 2-byte instructions. When executing extended I/O instructions, the second byte of the instruction is output on the lower 8 bits of the address bus (ADRO-ADR7). This information is normally used as an I/O device address to select 1 of up to 256 input or output devices, but may also be used to output control or status signals.

Parallel I/O operations may use any CPU register as the data source or destination. This offers significant flexibility in writing I/O software, because there is not a single accumulator register to create a "bottle-neck" in the data flow. The functional block diagram in Figure 1 illustrates the various I/O facilities.

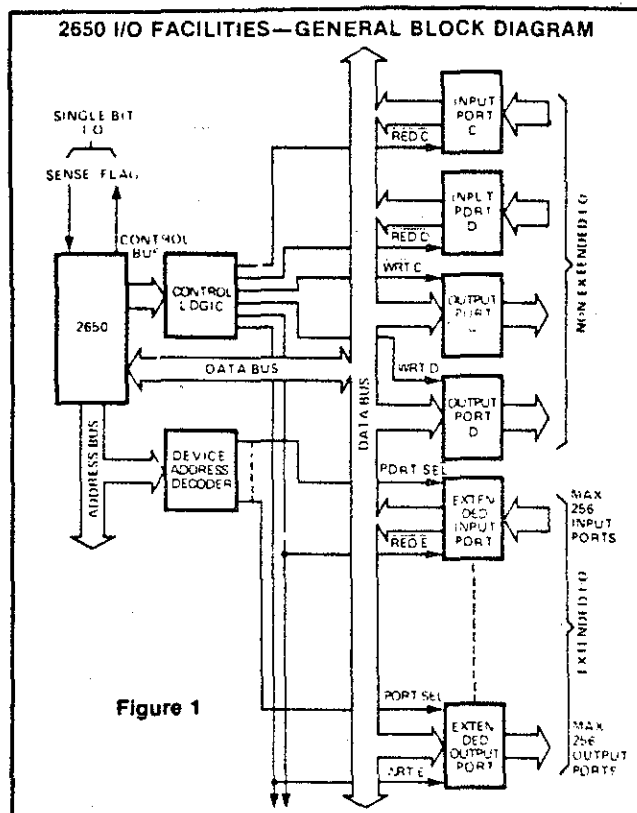
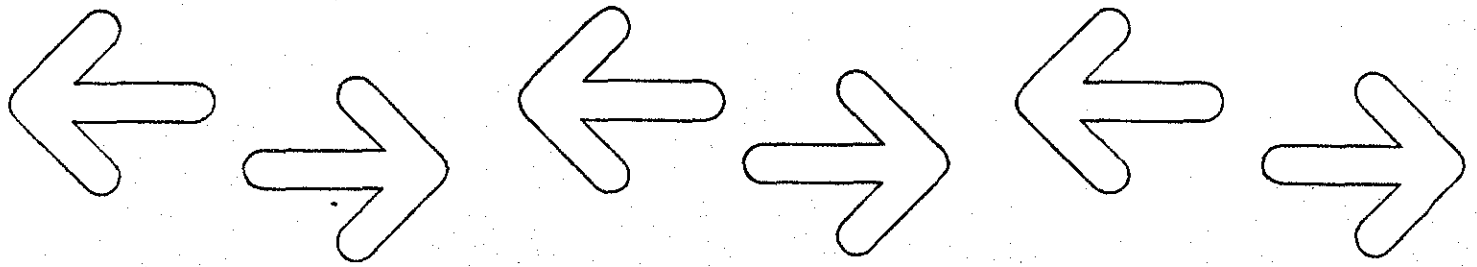


Figure 1

## I/O AS PART OF THE MEMORY ADDRESS SPACE

The 2650 user may choose to transfer data into or out of the processor using the memory control signals. The advantage of this technique is that the data can be read or written by the program with memory load and store instructions, and data may be directly operated upon with



# SIGNETICS 2650

logical and arithmetic instructions. The memory referencing instructions can take advantage of the flexible addressing modes provided by the system, such as indexing and indirect addressing. A possible disadvantage of this method is that it may be necessary to decode more address lines to determine the device address than with the other I/O facilities.

To make use of this technique, the designer must assign memory addresses to I/O devices and design the device interfaces to respond to the same signals as memory.

## I/O INTERFACE SIGNALS

Table 1 summarizes the state of the 2650 I/O interface signals for the various methods of I/O which are available.

## SENSE INPUT AND FLAG OUTPUT

One of the I/O capabilities of the 2650 is provided by the sense input and flag output. The sense and flag pins may be used for single-bit input or output of status or control information. They can also be used to implement a serial data communications channel. Two examples of this application are given below.

**ASYNCHRONOUS SERIAL COMMUNICATIONS PORT:** In applications where a serial type of terminal (like a teletypewriter) must be connected to the microcomputer systems, the sense pin and flag pin can be used to interface with the terminal. The basic character format for asynchronous serial I/O is shown in Figure 2.

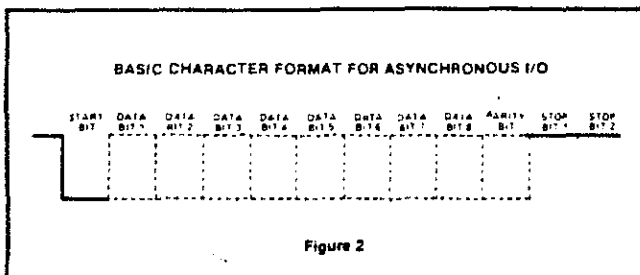


Figure 2

A number of parameters of this character format, and the transmission speed, is different for various types of terminals. The variable parameters are:

Baud rate (bits per second): 110, 150, 300, 600,

1200, 1300, 4800, and 9600 baud.

Number of bits per character: 5, 6, 7, or 8 bits.

Parity mode: even, odd, and no parity.

Number of stop bits: 1 or 2.

The control of the sense and flag pins for asynchronous serial I/O, with the appropriate parameters and baud rate, can be done completely with software. The hardware involved is limited to a simple line driver and receiver circuit which may be either an RS-232 interface or a 20mA current loop interface. The interface hardware is shown in Figure 3.

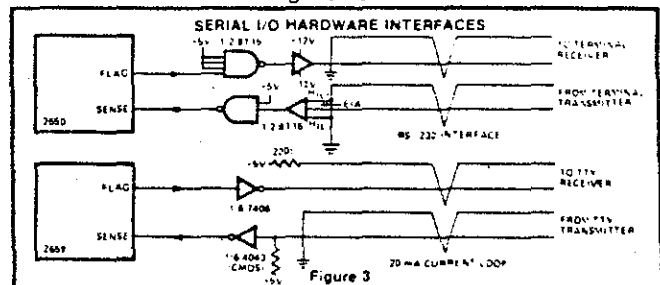


Figure 3

The software necessary to accomplish the serial I/O for a full-duplex line can be divided into 3 parts: 1) The start bit detection and verification. After each start bit detection, the start-bit level is verified for a low level at time intervals of 1/6 of 1-bit time. This prevents false start-bit recognition caused by line noise. 2) The sampling of the data bits at the mid-bit time, echoing the data bit to the flag output, and loading the data bit into a CPU register. 3) The input, echo and check of parity bit and stop bits.

A timing diagram showing the start bit sampling and the bit echo appears in Figure 4.

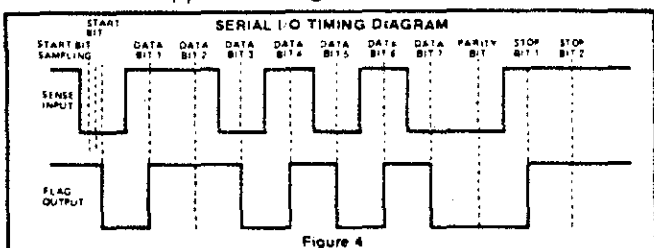


Figure 4

| TYPE OF I/O OPERATION | OPRED | M/I/O | R/W | ADR0-ADR7                  | ADR13 (E/NE) | ADR14 (D/C) |
|-----------------------|-------|-------|-----|----------------------------|--------------|-------------|
| Sense (Input)         | X     | X     | X   | X                          | X            | X           |
| Flag (Output)         | X     | X     | X   | X                          | X            | X           |
| Extended Read         | H     | L     | L   | Second Byte of Instruction | H            | X           |
| Extended Write        | H     | L     | H   | Second Byte of Instruction | H            | X           |
| Non-Extended Read C   | H     | L     | L   | X                          | L            | L           |
| Non-Extended Read D   | H     | L     | L   | X                          | L            | H           |
| Non-Extended Write C  | H     | L     | H   | X                          | L            | L           |
| Non-Extended Write D  | H     | L     | H   | X                          | L            | H           |
| Memory I/O Read       | H     | H     | L   | ADR0-ADR7                  | ADR13        | ADR14       |
| Memory I/O Write      | H     | H     | H   | ADR0-ADR7                  | ADR13        | ADR14       |

X : Don't Care

Table 1. I/O Interface Signal State

| BAUD RATE | SAMPLE DELAY NUMBER AT 1.25MHz | BIT DELAY NUMBER AT 1.25MHz | NUMBER OF BDRR,RO INSTRUCTIONS AT 1.25MHz | NUMBER OF BDRR,RO INSTRUCTIONS AT 1MHz |
|-----------|--------------------------------|-----------------------------|-------------------------------------------|----------------------------------------|
| 100       | 00                             | E5                          | 5                                         | 4                                      |
| 500       | 4A                             | C5                          | 2                                         | 2                                      |
| 900       | 2A                             | DE                          | 1                                         | 1                                      |
| 1200      | 1A                             | 6A                          | 1                                         | 1                                      |
| 2400      | 07                             | 30                          | 1                                         | 1                                      |

Table II

| BUS A |     |     |                   |    |                   |
|-------|-----|-----|-------------------|----|-------------------|
| RBA   | WBA | CLK | BUS A             |    |                   |
| X     | 0   | 1   | WRITE (A ← latch) |    |                   |
| 0     | 1   | X   | READ (latch → A)  |    |                   |
| X     | X   | X   | Hi-Z (Tri-state)  |    |                   |
| BUS B |     |     |                   |    |                   |
| RBB   | WBB | WBA | CLK               | ME | BUS B             |
| X     | X   | X   | 1                 | 1  | Hi-Z              |
| X     | 0   | X   | 0                 | 0  | Hi-Z              |
| X     | 1   | 0   | X                 | 0  | Hi-Z              |
| 0     | 0   | X   | X                 | 0  | READ (latch → B)  |
| X     | X   | 1   | 1                 | 0  | WRITE (B ← latch) |

Table III. 8T31 Control Functions

Three examples of the serial I/O routine with different speed and parameters are presented in Figures 5 through 9. The bit and sample delay number (hexadecimal) in the definition listing (Figure 6) are for a CPU clock frequency of 1MHz. The hexadecimal delay numbers for a frequency of 1.25MHz are given in Table 11. This table also lists the number of BDRR,RO instructions that are necessary in the "bit delay and echo subroutine" to count cycles for the appropriate baud rate.

The serial I/O routine uses four CPU registers (1 band and RO) and affects seven of the Program Status Word bits; namely, Sense, Flag, Overflow, Carry Interdigit Carry, and the two Condition Code Bits. The program also uses one level of the return address stack.

A parity error will set the Overflow bit, and a framing error (wrong stop bit level) will set the Interdigit Carry bit. At the end of the routine, the input character is stored in register R2.

**DATA STRING OUTPUT:** A typical application for the flag output is a data string output. The advantage for this output method is that it can provide a large number of output bits with little address or control logic decoding. For example, this method can be used to output data for an array of numeric displays, single bit indicators, or column drivers of a parallel numeric printer. An example of the hardware required to implement this type of output channel is given in Figure 10.

Here, the Address 14 output is used as a data strobe signal. However, the data strobe signal could also be built up by decoding more address bits so that the system memory size would not be limited to 16K bytes as in this example.

A listing of the program required is given in Figure 14. The data is assumed to be located in the system's RAM as illustrated in Figure 11.

The least-significant bit of the least-significant byte will be output first. The table length (TLEN) and the number of bits per byte (BPW) can be adapted as necessary by software modifications. The data strobe pulse on output ADR14 is generated by doing the dummy instruction STRA, RO to address H'4000'.

## PARALLEL INPUT/OUTPUT

The 2650 instruction set contains the following six input/output instructions:

|               |                | NO. BYTES |
|---------------|----------------|-----------|
| WRTC, RX      | Write Control  | 1         |
| REDC, RX      | Read Control   | 1         |
| WRTD, RX      | Write Data     | 1         |
| REDD, RX      | Read Data      | 1         |
| WRTe, RX DEVA | Write Extended | 2         |
| REDe, RX DEVA | Read Extended  | 2         |

The control signals generated by each I/O instruction simplify the interface circuitry required to generate I/O selection and timing signals. A low-cost control signal interface with related timing is shown in Figures 15 and 16.

When using standard TTL and 8T series I/O ports, the I/O operations can be done without slowing down the system. In this case the OPACK input could be controlled directly for all I/O operations.

**NON-EXTENDED I/O:** The single-byte I/O instructions of the 2650 are referred to as non-extended I/O. In small systems with only two 8-bit input ports and two 8-bit output ports, this I/O facility requires a minimum of hardware interfacing between the CPU and I/O ports. The signals WRTC, WRTD, REDC, and REDD generated by the control logic decoder in Figure 15 can be used

## FLOWCHART OF THE SERIAL I/O ROUTINE

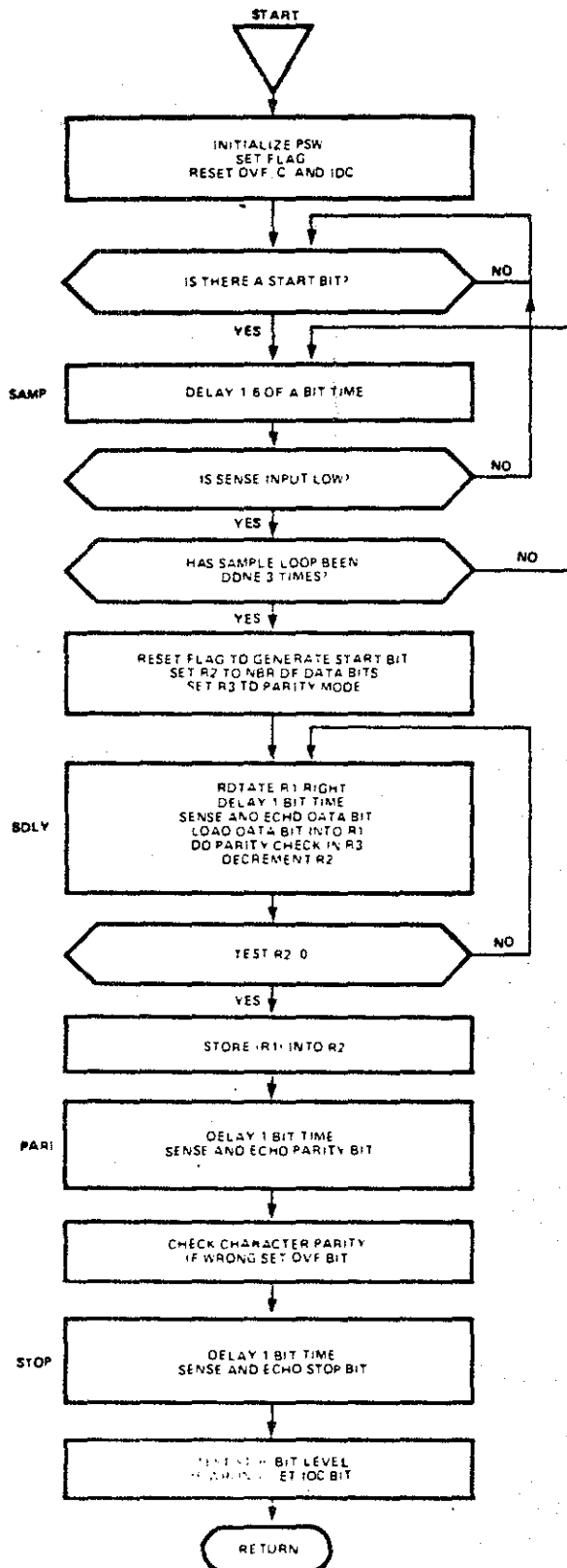


Figure 5

## SERIAL I/O PARAMETER DEFINITIONS

TWIN ASSEMBLER VER 1.0

PAGE 0001

LINE ADDR OBJECT E SOURCE

```

0001 * PG760091
0002 ****
0003 *
0004 * **** PROGRAMMABLE SERIAL I/O ROUTINE ****
0005 *
0006 * WITH THIS PROGRAM THE SENSE AND FLAG INPUT/OUTPUT OF
0007 * THE 2650 ARE USED TO INTERFACE WITH TERMINALS
0008 * SUCH AS TV, CRT TERMINALS, ETC. VIA THE BIT SERIAL
0009 * ASYNCHRONOUS LINE DISCIPLINE
0010 *
0011 * ALL CHARACTER AND LINE PARAMETERS CAN BE MODIFIED
0012 * SIMPLY IN THE SOFTWARE. THESE PARAMETERS ARE BRUD
0013 * RATE, NUMBER OF DATA BITS, PARITY MODE AND STOP BITS
0014 *
0015 * THE PROGRAM HAS BEEN SET UP FOR A FULL DUPLEX LINE
0016 * BUT CAN EASILY BE MODIFIED TO HALF DUPLEX MODE
0017 *
0018 ****
0019 *
0020 * DEFINITIONS OF SYMBOLS
0021 *
0022 PS EQU 0 PROCESSOR REGISTERS
0023 R1 EQU 1
0024 R2 EQU 2
0025 R3 EQU 3
0026 S EQU M 50 PSW SENSE
0027 F EQU M 40 FLAG
0028 ICR EQU M 20 INTERDIGIT CARRY
0029 OVF EQU M 04 OVERFLOW
0030 L EQU M 01 CARRY/SCROLL
0031 N EQU 2 BRANCH CONDITION NEGATIVE
0032 UN EQU 3 UNCONDITIONAL
0033 *
0034 ****
0035 *
0036 * SOFTWARE DEFINITIONS OF BRUD RATE, CHARACTER FORMAT, PARITY,
0037 * PARITY MODE, ETC.
0038 *
0039 *
0040 * NUMBER OF DATA BITS
0041 *
0042 DB9 EQU M 06 CHARACTER HAS 9 DATA BITS
0043 DB8 EQU M 06 CHARACTER HAS 8 DATA BITS
0044 DB7 EQU M 07 CHARACTER HAS 7 DATA BITS
0045 DB6 EQU M 06 CHARACTER HAS 6 DATA BITS
0046 DB5 EQU M 05 CHARACTER HAS 5 DATA BITS
0047 DB4 EQU M 04 CHARACTER HAS 4 DATA BITS
0048 DB3 EQU M 03 CHARACTER HAS 3 DATA BITS
0049 DB2 EQU M 02 CHARACTER HAS 2 DATA BITS
0050 DB1 EQU M 01 CHARACTER HAS 1 DATA BIT
0051 *
0052 * BIT DELAYS AT 1 Mhz CLOCK FREQUENCY
0053 *
0054 BR01 EQU M 08 BIT DELAY AT 110 BRUD
0055 BR02 EQU M 09 BIT DELAY AT 100 BRUD
0056 BR03 EQU M 08 BIT DELAY AT 900 BRUD
0057 BR04 EQU M 05 BIT DELAY AT 1100 BRUD
0058 BR05 EQU M 05 BIT DELAY AT 2400 BRUD
0059 *
0060 * START BIT SAMPLE DELAYS AT 1 Mhz CLOCK FREQUENCY
0061 *
0062 SR01 EQU M 05 SAMPLE DELAY AT 110 BRUD
0063 SR02 EQU M 06 SAMPLE DELAY AT 100 BRUD
0064 SR03 EQU M 06 SAMPLE DELAY AT 900 BRUD
0065 SR04 EQU M 06 SAMPLE DELAY AT 1100 BRUD
0066 SR05 EQU M 05 SAMPLE DELAY AT 2400 BRUD
0067 *
0068 * PARITY MODE
0069 *
0070 EP EQU M 06 EVEN PARITY
0071 OP EQU M 06 ODD PARITY

```

Figure 6

directly as output port clock pulses and input port enable signals, respectively.

**SEQUENTIAL I/O WITH NON-EXTENDED I/O INSTRUCTIONS:** In systems where a larger number of devices must be serviced in sequence, the use of a simple 8-bit output port can offer considerable savings in software. Normally the devices could be serviced with extended I/O instructions. However, since the device address is the second byte in this type of instruction, a series of data fetch and I/O instructions would be required to service the devices in sequence.

With an 8-bit output port functioning as a device address register, the device address can be modified under software control. In this way, a simple program loop can serve up to eight I/O ports by rotating a single '1' through a CPU register that is output as a device address. This I/O addressing technique may also be used advantageously in systems where I/O operation requests are detected by software polling. A functional block diagram of this technique is shown in Figure 17.

**EXTENDED I/O:** There are two extended I/O instructions in the 2650 instruction set. In these 2-byte instructions, the first byte specifies the operation code and the data source or destination register in the CPU. The second byte provides an 8-bit device address code that is output on the eight least-significant bits of the address bus, ADR0 through ADR7.

The control signal decoding diagram (Figure 15) can be simplified for systems using only extended I/O, as shown in Figure 18. The timing diagram of Figure 16 also applies to this decoding technique.

**DEVICE ADDRESS DECODING SCHEMES:** For extended I/O it is necessary to decode the address lines ADR0 through ADR7 in order to generate appropriate port selection signals. The choice of an address decoding scheme depends on factors such as total I/O requirements, the type of I/O ports used, and the total system configuration.

In principle, there are two basic methods of device address decoding. One method is the use of hardwired logic in which the device address is fixed; the other is a hardware programmable method in which the device addresses are individually set with jumpers or switches. Some examples of these methods are given in Figures 19 and 20.

In many applications a combination of these two methods is used. In addition, the control logic can be implemented as an integral part of the device address decoding. An example is shown in Figure 21.

**MEMORY MAPPED I/O:** In memory mapped I/O, the I/O devices are treated as memory locations. An advantage of this technique is that all memory referencing instruction types (store, load, arithmetic, logical, etc.) can be used directly for I/O data. Device address decoding is

#### SERIAL I/O ASSEMBLY LISTING—EXAMPLE 1 110 Baud, 7 Data Bits, Even Parity and 1 Stop Bit

LINE ADDR OBJECT E SOURCE

```

0073 *****
0074 * EXAMPLE 1. FULL DUPLEX (BIT BY BIT ECHO), 110 BAUD.
0075 * 7 DATA BITS, EVEN PARITY AND 1 STOP BIT
0076 *
0077 0000 ORG M 0500
0078 0500 7640 START PPSU F SET FLAG TO SWITCH OFF THE LINE
0079 0502 7525 CPSL ONF=10C
0080 0504 12 TEST SPSU WAIT FOR START BIT
0081 0505 1A70 BCTR.N TEST
0082 0507 0602 LODI.R2 M 03 SET R2 TO NUMBER OF SAMPLES
0083 0509 0540 SHPP LODI.R1 S000 SET R1 TO SAMPLE DELAY
0084 050B F97E BARR.R1 0
0085 050C 12 SPSU
0086 050E 1A74 BCTR.N TEST TEST FOR START BIT VALIDITY
0087 0510 FA77 BARR.R2 SHPP IF NOT VALID, GO BACK TO TEST
0088 0512 0700 LODI.R3 0F SET R3 TO EVEN PARITY MODE
0089 0514 0607 LODI.R2 067 SET R2 TO NUMBER OF DATA BITS
0090 0516 7440 CPSU F GENERATE START BIT
0091 0518 01 BITS RRR R1
0092 0519 381A BSTR.UN B0LY GO TO DELAY AND ECHO ROUTINE
0093 051B FA7B BARR.R2 BITS TEST FOR NUMBER OF DATA BITS
0094 051D 01 LODI R1
0095 051E 02 STRZ R2
0096 051F 3814 PRR1 BSTR.UN B0LY LOAD R2 WITH CHARACTER
0097 0521 9A02 BCTR.N STOP
0098 0523 7704 PPSL ONF IF WRONG PARITY, SET ONF
0099 0525 0700 STOP LODI.R3 0 CLEAR R3
0100 0527 380C BSTR.UN B0LY
0101 0529 1A02 BCTR.N STOP TEST STOP BIT LEVEL
0102 052B 7720 PPSL IDC IF WRONG, SET IDC BIT
0103 052D 0700 STOP LODI.R3 0 CLEAR R3
0104 052F 3804 BSTR.UN B0LY
0105 0531 16 EXIT RETC.M TEST STOP BIT 2 LEVEL
0106 0532 7720 PPSL IDC IF WRONG, SET IDC BIT
0107 0534 17 EXIT RETC.UN
0108 *****
0109 * BIT DELAY AND ECHO SUBROUTINE
0110 *
0111 *
0112 0535 0400 B0LY LODI.R0 B000 SET R0 TO BIT DELAY NUMBER
0113 0537 FA7E BARR.R0 0
0114 0539 12 SPSU
0115 053B 1A04 BCTR.N ONE TEST DATA BIT LEVEL
0116 053E 7440 CPSU F IF LOW, ECHO A ZERO
0117 0540 1B04 BCTR.UN BITS
0118 0542 7640 ONE PPSU F IF HIGH, ECHO A ONE
0119 0544 23 LODI.R1 0F7 INSERT DATA BIT INTO R1
0120 0546 03 STRZ R2
0121 0548 17 RETC.UN DO PARITY CHECK
0122 *****
0123 0000 END 0
TOTAL ASSEMBLY ERRORS = 0000

```

Figure 7

#### EXAMPLE 2 600 Baud, 7 Data Bits, Odd Parity and 2 Stop Bits

LINE ADDR OBJECT E SOURCE

```

0073 *****
0074 * EXAMPLE 2. FULL DUPLEX (BIT BY BIT ECHO), 600 BAUD.
0075 * 7 DATA BITS, ODD PARITY AND 2 STOP BITS
0076 *
0077 0000 ORG M 0500
0078 0500 7640 START PPSU F SET FLAG TO SWITCH OFF THE LINE
0079 0502 7525 CPSL ONF=10C
0080 0504 12 TEST SPSU WAIT FOR START BIT
0081 0505 1A70 BCTR.N TEST
0082 0507 0603 LODI.R2 M 03 SET R2 TO NUMBER OF SAMPLES
0083 0509 0510 SHPP LODI.R1 S000 SET R1 TO SAMPLE DELAY
0084 050B F97E BARR.R1 0
0085 050C 12 SPSU
0086 050E 1A74 BCTR.N TEST TEST FOR START BIT VALIDITY
0087 0510 FA77 BARR.R2 SHPP IF NOT VALID, GO BACK TO TEST
0088 0512 0700 LODI.R3 0F SET R3 TO ODD PARITY MODE
0089 0514 0607 LODI.R2 067 SET R2 TO NUMBER OF DATA BITS
0090 0516 7440 CPSU F GENERATE START BIT
0091 0518 01 BITS RRR R1
0092 0519 381A BSTR.UN B0LY GO TO DELAY AND ECHO ROUTINE
0093 051B FA7B BARR.R2 BITS TEST FOR NUMBER OF DATA BITS
0094 051D 01 LODI R1
0095 051E 02 STRZ R2
0096 051F 3814 PRR1 BSTR.UN B0LY LOAD R2 WITH CHARACTER
0097 0521 9A02 BCTR.N STOP
0098 0523 7704 PPSL ONF IF WRONG PARITY, SET ONF
0099 0525 0700 STOP LODI.R3 0 CLEAR R3
0100 0527 380C BSTR.UN B0LY
0101 0529 1A02 BCTR.N STOP TEST STOP BIT LEVEL
0102 052B 7720 PPSL IDC IF WRONG, SET IDC BIT
0103 052D 0700 STOP LODI.R3 0 CLEAR R3
0104 052F 3804 BSTR.UN B0LY
0105 0531 16 EXIT RETC.M TEST STOP BIT 2 LEVEL
0106 0532 7720 PPSL IDC IF WRONG, SET IDC BIT
0107 0534 17 EXIT RETC.UN
0108 *****
0109 * BIT DELAY AND ECHO SUBROUTINE
0110 *
0111 *
0112 0535 0400 B0LY LODI.R0 B000 SET R0 TO BIT DELAY NUMBER
0113 0537 FA7E BARR.R0 0
0114 0539 12 SPSU
0115 053B 1A04 BCTR.N ONE TEST DATA BIT LEVEL
0116 053E 7440 CPSU F IF LOW, ECHO A ZERO
0117 0540 1B04 BCTR.UN BITS
0118 0542 7640 ONE PPSU F IF HIGH, ECHO A ONE
0119 0544 23 LODI.R1 0F7 INSERT DATA BIT INTO R1
0120 0546 03 STRZ R2
0121 0548 17 RETC.UN DO PARITY CHECK
0122 *****
0123 0000 END 0
TOTAL ASSEMBLY ERRORS = 0000

```

Figure 8

not necessarily more complex than for normal extended I/O, since all I/O addresses could be located in a specific address block. Of course, this technique can only be used in systems which do not use the full memory address space for programs. A diagram of the I/O control logic, using the ADR14 output to discriminate between memory and I/O operations, is given in Figure 22. The device address decoding methods described earlier can also be applied to memory mapped I/O.

### SINGLE POINT CONTROL

In many applications, the capability to set, clear, or test a single output point selected from a large number of output points is required. Designs of this type can be implemented using the 2650 I/O instructions. When used as described below, the WRTE, WRTC, and WRD instructions become "set/clear single-bit" instructions, while the REDE instruction becomes a "test single-bit" instruction.

**SINGLE BIT OUTPUT-DIRECT ADDRESS:** The write extended instruction can be used to select and set or clear a single output bit. The two bytes of the instruction can be interpreted as follows:

#### BYTE 0



#### BYTE 1



### EXAMPLE 3

2400 Baud, 8 Data Bits, No Parity and 1 Stop Bit

LINE ADDR OBJECT E SOURCE

```

0073
0074 * EXAMPLE 1: FULL DUPLEX 4 BIT BY 6 BIT ECHO, 2400 BAUD.
0075 * 8 DATA BITS, NO PARITY AND 1 STOP BIT
0076 *
0077 0000 ORG M 0000
0078 0500 7540 STP1 PPSU F SET FLAG TO SWITCH OFF THE LINE
0079 0502 7525 CPSL ONF-H+100
0080 0504 12 TEST SP5U WAIT FOR START BIT
0081 0505 1A7D BCTR.N TEST
0082 0507 0A02 LOD1.R2 M 05 SET R2 TO NUMBER OF SAMPLES
0083 0509 0505 SAMP LOD1.R1 S024 SET R1 TO SAMPLE DELAY
0084 0506 F97E BARR.R1 8
0085 0508 12 SP5U TEST FOR START BIT VALIDITY
0086 0506 1A74 BCTR.N TEST IF NOT VALID. GO BACK TO TEST
0087 0510 FA77 BARR.R2 SAMP
0088 0512 0606 LOD1.R2 066 SET R2 TO NUMBER OF DATA BITS
0089 0514 7440 CPSU F GENERATE START BIT
0090 0516 51 BIT5 RPR.R1
0091 0517 258C BSTR.UN B0LY GO TO DELAY AND ECHO ROUTINE
0092 0519 FA76 BARR.R2 61T5 TEST FOR NUMBER OF DATA BITS
0093 0516 01 LOD2 R1
0094 0510 02 STR2 R2 LOAD R2 WITH CHARACTER
0095 0510 0706 STOP LOD1.R3 0 CLEAR R3
0096 051F 2604 BSTR.UN B0LY
0097 0521 14 EX11 RETC.N TEST STOP BIT LEVEL
0098 0522 7726 PPSL IDC IF WRONG, SET IDC BIT
0099 0524 17 EX12 RETC.UN
0100
0101 * BIT DELAY AND ECHO SUBROUTINE
0102 *
0103 *
0104 0525 0425 B0LY LOD1.R0 0A24 SET R0 TO BIT DELAY NUMBER
0105 0527 F87E BARR.R0 8
0106 0529 12 SP5U TEST DATA BIT LEVEL
0107 052A 1A04 BCTR.N ONE
0108 052C 7440 CPSU F IF LOW, ECHO A ZERO
0109 052E 1A04 BCTR.UN BIT1
0110 0530 7540 ONE PPSU F IF HIGH, ECHO A ONE
0111 0532 0500 TOR1.R1 0F6 INSERT DATA BIT INTO R1
0112 0534 03 BIT1 STR2 R3
0113 0535 17 RETC.UN
0114 *
0115 0000 END 8

```

TOTAL ASSEMBLY ERRORS = 0000

Figure 9

### INTERFACE DIAGRAM FOR DATA STRING OUTPUT

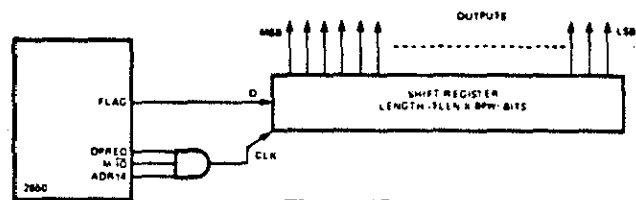


Figure 10.

### DATA ORGANIZATION FOR DATA STRING OUTPUT

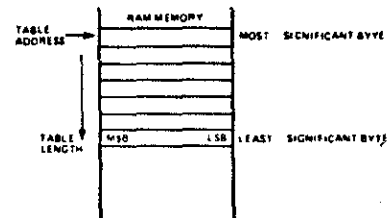


Figure 11.

### TIMING DIAGRAM OF DATA STRING OUTPUT ROUTINE

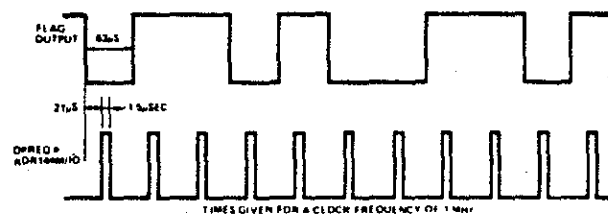


Figure 12.

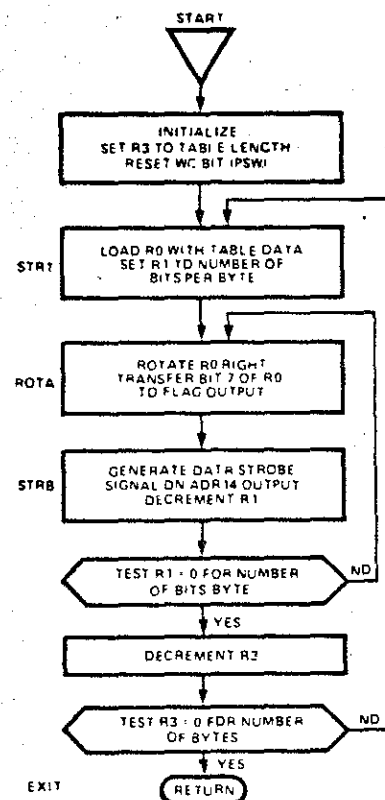


Figure 13.



A<sub>0</sub> through A<sub>7</sub> of the second byte specify the output selected. The S/C bit specifies whether the bit is set or cleared. A typical hardware configuration controlling 64 points is shown in Figure 23. Here, the control line decoding and partial address decoding is done by the 74LS138, which selects one of the eight 9334s. One of the eight latches in the selected 9334 is enabled by ADR<sub>0</sub>, ADR<sub>1</sub>, and ADR<sub>2</sub> and is either cleared or set, as determined by the value of ADR<sub>7</sub>.

The XX field in the first byte selects one of the four available registers and outputs in its contents on the data bus. Since this information is not used in this application, the value of XX is not important. However, it could be used to output an 8-bit control or status word in conjunction with the set/clear operation.

**SINGLE BIT OUTPUT-INDIRECT ADDRESS:** If the address of the output to be set or cleared must be determined at program run time, the WRTD and WRTC instructions can be used. The address of the output bit is first loaded into one of the 2650 registers. A WRTD, Rx instruction is

| ASSEMBLY LISTING OF DATA STRING OUTPUT ROUTINE |             |        |                                                                   |
|------------------------------------------------|-------------|--------|-------------------------------------------------------------------|
| LINE                                           | ADDR        | OBJECT | SOURCE                                                            |
| 0001                                           |             |        | • P0760054                                                        |
| 0002                                           |             |        | *****                                                             |
| 0003                                           |             |        | •                                                                 |
| 0004                                           |             |        | • DATA STRING OUTPUT ROUTINE •                                    |
| 0005                                           |             |        | •                                                                 |
| 0006                                           |             |        | • THIS PROGRAM TRANSFERS THE CONTENTS OF A MEMORY TABLE IN BIT BY |
| 0007                                           |             |        | • BIT SERIAL FORM TO THE FLAG OUTPUT OF THE 2650                  |
| 0008                                           |             |        | •                                                                 |
| 0009                                           |             |        | • THE TABLE LENGTH AND THE NUMBER OF BITS ARE SOFTWARE PROGRAMMED |
| 0010                                           |             |        | •                                                                 |
| 0011                                           |             |        | • A DATA STROBE OUTPUT IS GENERATED ON THE ADDRESS 14 OUTPUT      |
| 0012                                           |             |        | •                                                                 |
| 0013                                           |             |        | *****                                                             |
| 0014                                           |             |        | •                                                                 |
| 0015                                           |             |        | • DEFINITIONS OF SYMBOLS                                          |
| 0016                                           |             |        | •                                                                 |
| 0017                                           | 0000        | R0     | EQU 0 PROCESSOR REGISTERS                                         |
| 0018                                           | 0001        | R1     | EQU 1                                                             |
| 0019                                           | 0002        | R2     | EQU 2                                                             |
| 0020                                           | 0003        | R3     | EQU 3                                                             |
| 0021                                           | 0000        | S      | EQU M 00 PSU SENSE                                                |
| 0022                                           | 0000        | F      | EQU M 40 FLAG                                                     |
| 0023                                           | 0000        | MC     | EQU M 00' PSL 1-WITH 0-WITHOUT CARRY                              |
| 0024                                           | 0000        | N      | EQU 2 BRANCH COND. NEGATIVE                                       |
| 0025                                           | 0003        | UN     | EQU 3 UNCONDITIONAL                                               |
| 0026                                           |             |        | •                                                                 |
| 0027                                           | 0007        | TLEN   | EQU M 07 TABLE LENGTH                                             |
| 0028                                           | 0006        | BPW    | EQU M 06 NUMBER OF BITS PER BYTE                                  |
| 0029                                           |             |        | •                                                                 |
| 0030                                           | 0000        | ORG    | M 0000                                                            |
| 0031                                           | 0000        | TABL   | RES TLEN LOCATION OF TABLE                                        |
| 0032                                           |             |        | •                                                                 |
| 0033                                           |             |        | *****                                                             |
| 0034                                           |             |        | •                                                                 |
| 0035                                           | 0007        |        | ORG M 0000                                                        |
| 0036                                           | 0500 0707   | STRT   | LDI R3 TLEN                                                       |
| 0037                                           | 0501 7506   |        | CPSL MC                                                           |
| 0038                                           | 0504 0F0000 | STR1   | LDAR R0 TABL R3 LOAD R0 WITH TABLE DATA                           |
| 0039                                           | 0507 0506   |        | LDI R1 BPW SET R1 TO NUMBER OF BITS PER BYTE                      |
| 0040                                           | 0509 50     | ROTA   | RRR R0                                                            |
| 0041                                           | 050A 1A06   |        | BCTR R1 ONE TEST BIT                                              |
| 0042                                           | 050C 7440   | ZERO   | CPSU F IF ZERO, RESET FLAG                                        |
| 0043                                           | 050E 1004   |        | BCTR UN STR0                                                      |
| 0044                                           |             |        | •                                                                 |
| 0045                                           | 0510 4000   | RLA    | DATA M 40 00                                                      |
| 0046                                           |             |        | •                                                                 |
| 0047                                           | 0512 7040   | ONE    | PSU F IF ONE, SET FLAG                                            |
| 0048                                           | 0514 CC0510 | STR0   | STR0 R0 ADDR GENERATE STROBE SIGNAL ON RLA                        |
| 0049                                           | 0517 F970   |        | DOAR R1 ROTA TEST FOR NUMBER OF BITS                              |
| 0050                                           | 0519 F009   |        | DOAR R3 STR1 TEST FOR NUMBER OF BYTES                             |
| 0051                                           | 051B 17     | EXIT   | RETU UN                                                           |
| 0052                                           | 0000        |        | END B                                                             |
| TOTAL ASSEMBLY ERRORS = 0000                   |             |        |                                                                   |

Figure 14

## CONTROL SIGNAL INTERFACE USING THE 74(L) S138 DECODER

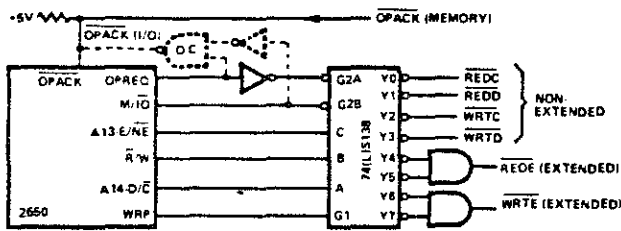


Figure 15.

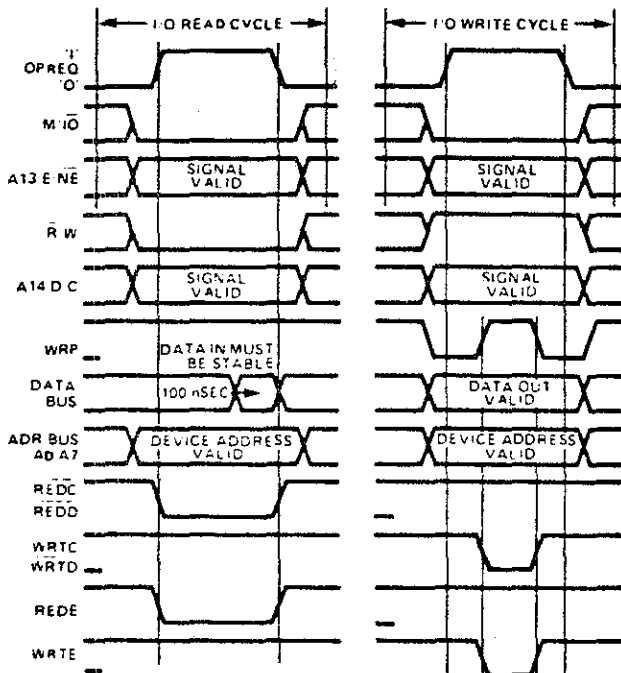


Figure 16

then issued if the bit is to be set, and a  $\overline{WRTC}$ , Rx instruction is issued if the bit is to be cleared. The bit select is output on the data bus, and the D/C output carries the *set/clear* information. The hardware implementation can be the same as shown in Figure 23, except that ADR0-ADR5 are replaced by DBUS0-DBUS5, and ADR7 is replaced by D/C.

**The 8T31 can be used to implement a flag register without the use of a memory byte in RAM. No additional hardware required and memory savings are considerable.**

**SINGLE BIT INPUT:** One way of doing single bit input uses the techniques described earlier. The address of the bit that is to be tested is loaded into one of the 2650 registers and output to an 8-bit latch using an extended or non-extended write instruction. The latch output is decoded to select the desired bit, which is then applied to the Sense input pin. The 2650 Program Status Word instructions can then be used to test the state of the Sense input and to take appropriate program action.

The technique described above must be used if "indirect" bit addressing is required. If this is not a requirement, a more efficient implementation can be accomplished using the extended *read* instruction. This technique makes use of the fact that the 2650 automatically tests the contents of a register every time it is used as the destination of an operation. Thus, when the *read* extended operation reads data from an input port, the condition code bits in the program status word are set to reflect whether the new register contents is positive, negative, or zero.

## SEQUENTIAL I/O TECHNIQUE

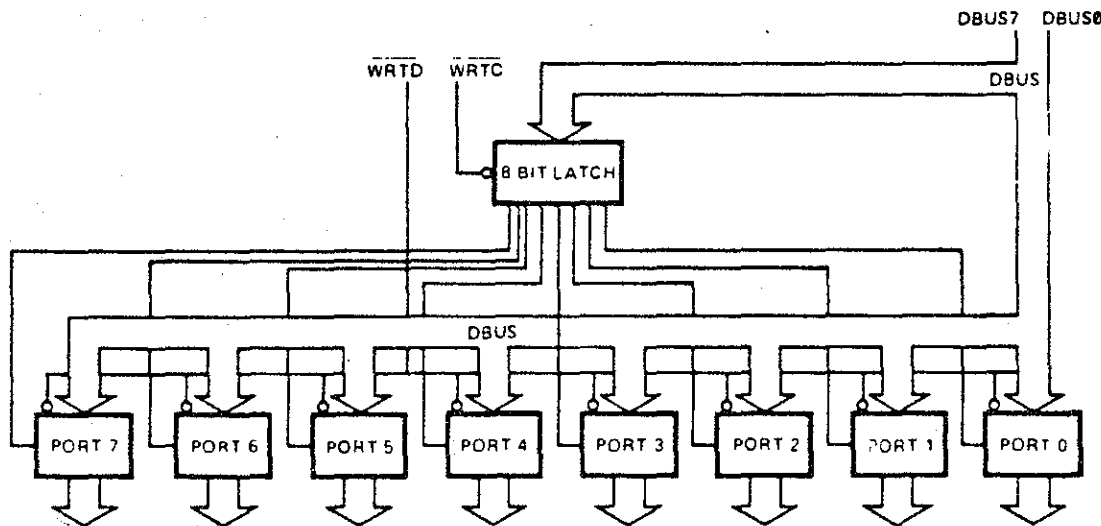
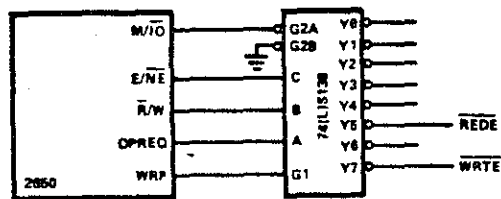
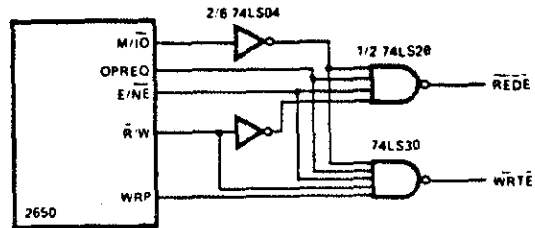


Figure 17

# SIMPLIFIED CONTROL LOGIC WHEN USING EXTENDED I/O ONLY



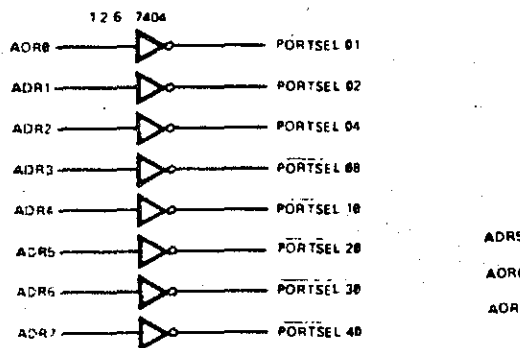
(A) Using 1-of-8 Decoder



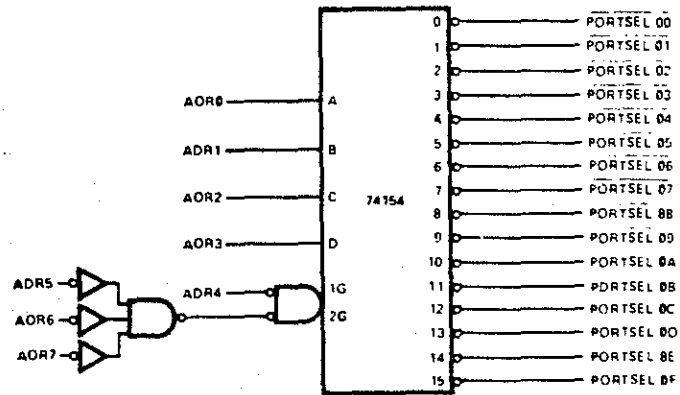
(B) Using Logic Gates

Figure 18

# SOME POSSIBLE TECHNIQUES FOR DEVICE ADDRESS DECODING

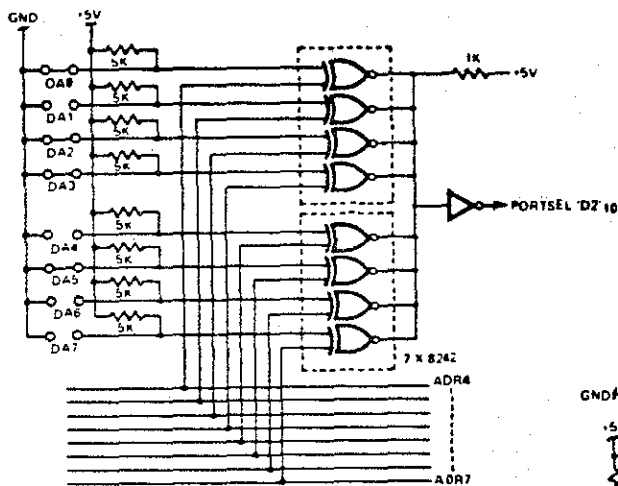


(A) Each address line selects one device (maximum of eight)

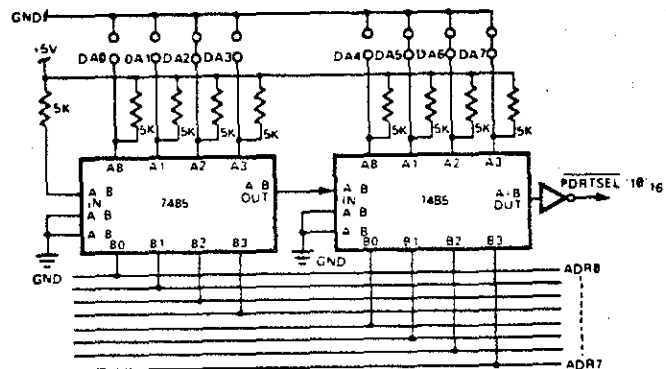


(B) Using a 1-out-of-16 decoder

Figure 19



(A) Using Exclusive-OR Gates



(B) Using Comparators

Figure 20

For the single bit input application, the second byte of the RETE, Rx instruction contains the address of the input bit to be tested. This data is applied to a bank of data selectors to select the addressed bit, which is then applied to the most-significant bit of the data bus, DBUS7. Since this is interpreted as the sign bit, the condition code bits in PSL will be set to reflect whether the bit being tested is a one or zero. A conditional branch instruction can then be used to affect the desired program action. A hardware implementation for 64 inputs is shown in Figure 24. Note that an address latch is not required for this method.

### INPUT PORT DEVICES

**GATED INPUT PORTS:** The simplest form of an input port is the tri-state gate. Figure 25 illustrates the use of the 8T97 high-speed HEX tri-state buffer for gated input ports. The 8T97 is non-inverting, and the tri-state control signals enable the buffers in groups of four and groups of two, so that 8-bit ports can be implemented efficiently.

An effective circuit for systems using 8-gated input ports is the 74251 8-to-1 multiplexer, which has tri-state outputs that can interface directly with the data bus. The advantage of this circuit is that no external address decoding logic is needed. A configuration using gated input ports with the 74251 multiplexer is illustrated in Figure 26.

In addition to these two configurations, many other input port configurations are possible using standard TTL or Signetics 8T series logic circuits.

**LATCHING INPUT PORTS:** Latching input ports may be required to store data from an external device, which is available only momentarily, before the actual input operation to the microprocessor takes place. This type of input port can be realized by connecting TTL-latch or D-type flip-flop circuits, such as the 7475, 74100, or 74175, to the inputs of a gated input port. As illustrated in Figure 27, by using the Signetics 8T10 Quad D-type flip-flop with tri-state outputs, an 8-bit latching input port can be implemented with only two packages. The 8T10 is functionally identical to the 74173.

### OUTPUT PORT DEVICES

Output ports can be configured with a variety of standard TTL and 8T series flip-flops and registers. Typical circuits include:

- 9334 Addressable 8-bit latch
- 7475 Quadruple latch
- 74100 8-bit latch
- 74175 Quadruple D-type flip-flop
- 8T10 Quadruple D-type flip-flop with tri-state outputs

The 7475 and 74175 both have true and complement outputs. One special feature of the 8T10 is that the outputs may be disabled (placed in a high-impedance output mode) by the device that is connected to this output port. A logic diagram using these circuits for output ports appears in Figure 28.

The 9334 is useful in systems requiring a large number of latched outputs, since a portion of the decoding can be done using the on-chip 3-input decoder. A typical application of this was shown in Figure 23. It is also an efficient circuit for implementing eight 8-bit output ports.

### I/O CONFIGURATIONS USING THE 8T31 BIDIRECTIONAL PORT

The 8T31 is an 8-bit bidirectional I/O port consisting of eight clocked latches with two bidirectional I/O buses, each of which has its own control logic. Each bus (A and B) has a read and a write control input, and there is a

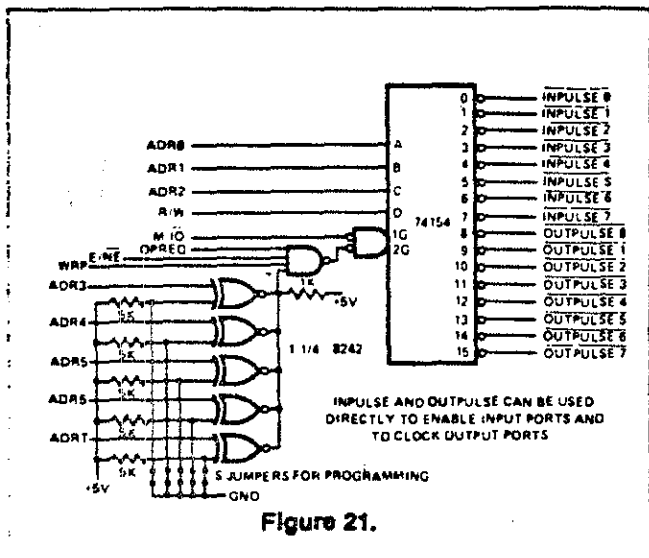


Figure 21.

### I/O CONTROL SIGNAL GENERATION FOR MEMORY MAPPED I/O

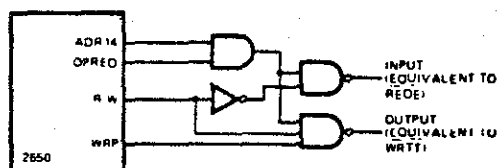


Figure 22.

### SIXTY-FOUR SINGLE BIT OUTPUTS USING THE 9334

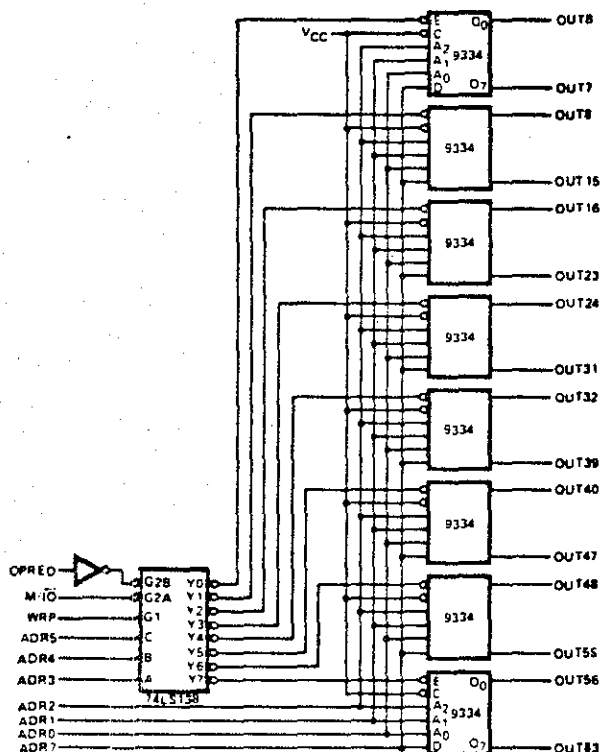


Figure 23.

master enable input for bus B only. The outputs of the latches follow the inputs when the clock is high, and latching will occur when the clock returns low.

The 8T31 is also equipped with a "power-on clear" circuit. If the clock input is held low until the power supply reaches 3.5V, the latches will be cleared. There is a logic inversion between bus A and bus B. As a result, when the 8T31 is cleared, bus A will have all logic "1" outputs and bus B all logic "0" outputs.

The control functions of the 8T31 are listed in Table 111. A functional block diagram and a symbolic diagram of the 8T31 are illustrated in Figures 29 and 30, respectively. As shown in Table 111, each bus can operate independently except for the care of writing from both bus A and B. In this case writing from bus A will override any attempt to write from bus B.

The control functions of the 8T31 allow it to be used in various microcomputer input/output applications. In the I/O system diagram of Figure 31, the 8T31 is used to implement gated input ports, latching input ports, output ports, and a bidirectional data bus driver. All I/O ports can be controlled directly with the device select and REDE and WRITE lines coming from device decoders and I/O control logic.

In applications where interfacing is necessary with peripheral devices that need data transfers in two directions, like digital cassettes and data link communication circuits, the 8T31 can be used as a bidirectional I/O port. In this application, the I/O operation should be requested by interrupt or polling to prevent simultaneous

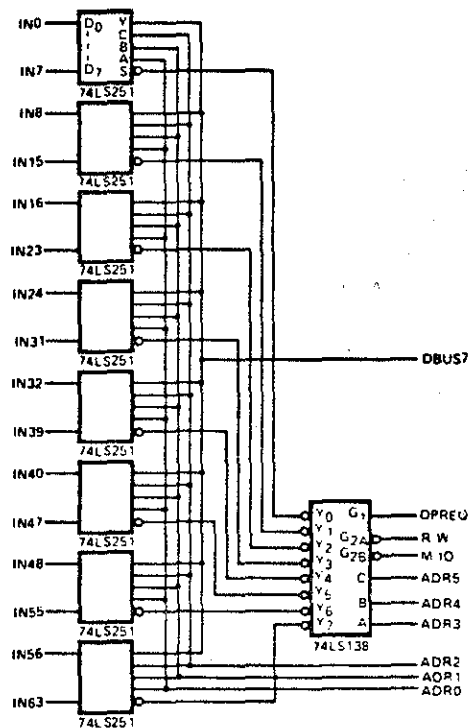


Figure 24

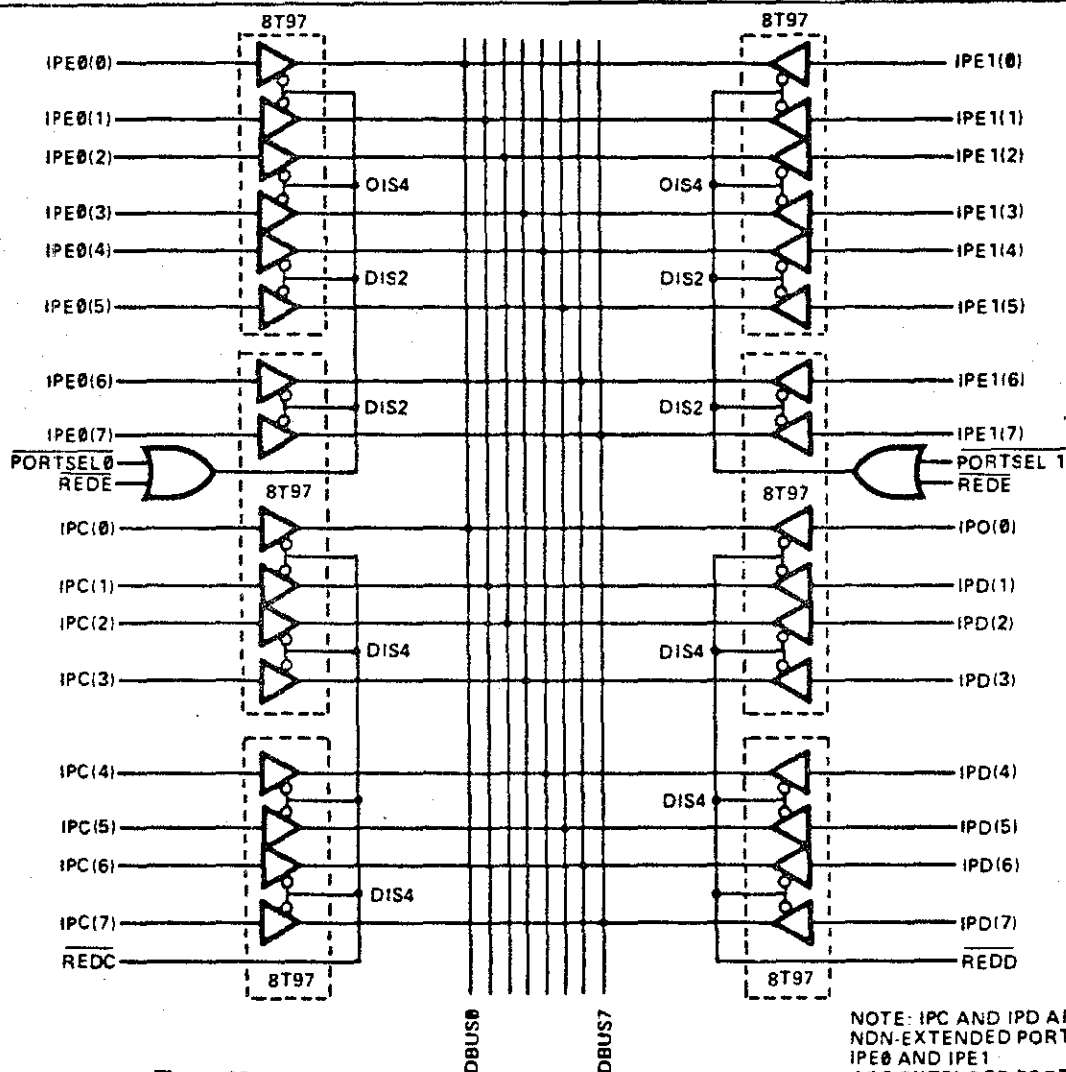


Figure 25

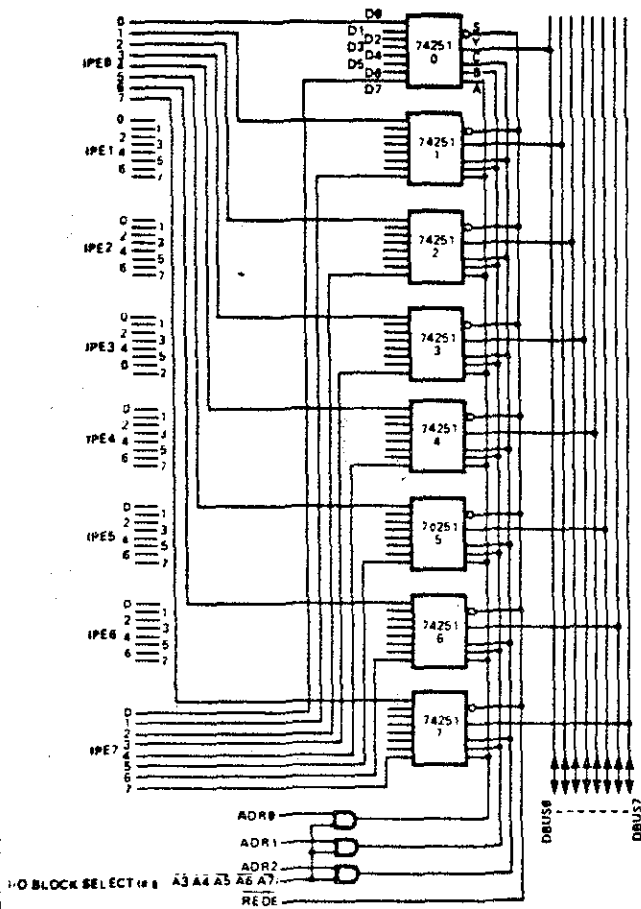


Figure 26

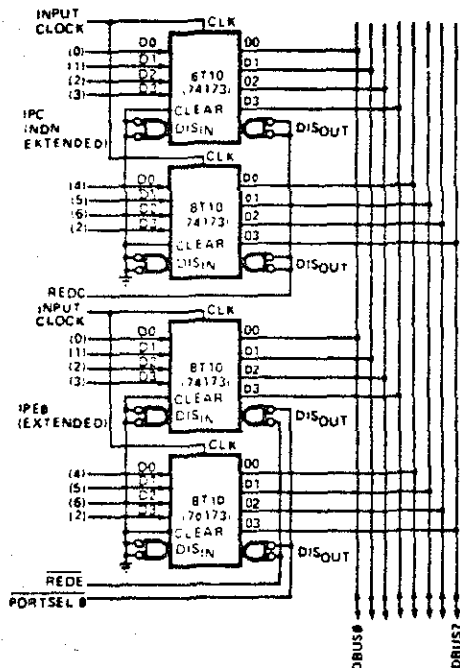


Figure 27

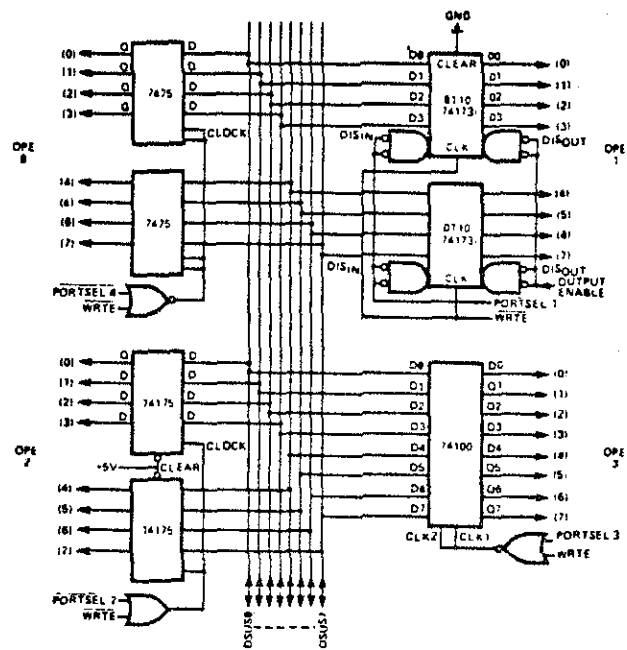


Figure 28

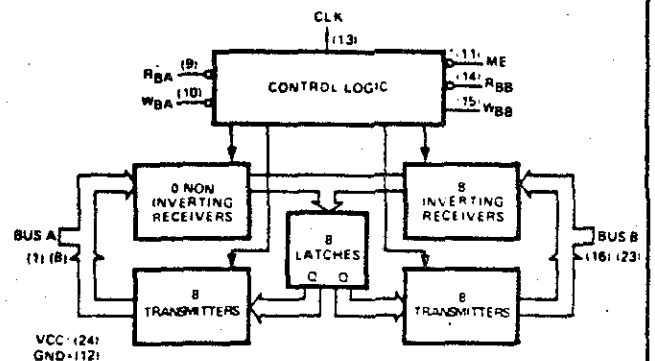


Figure 29

## 8T31 SYMBOLIC DIAGRAM

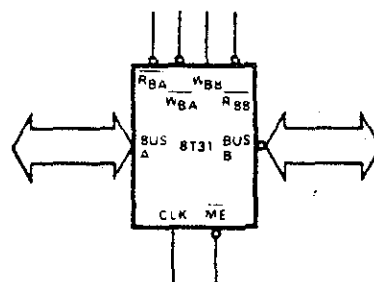


Figure 30

THE BT31 USED AS A GATED INPUT PORT,  
LATCHING INPUT PORT, OUTPUT PORT, AND DATA BUS DRIVE

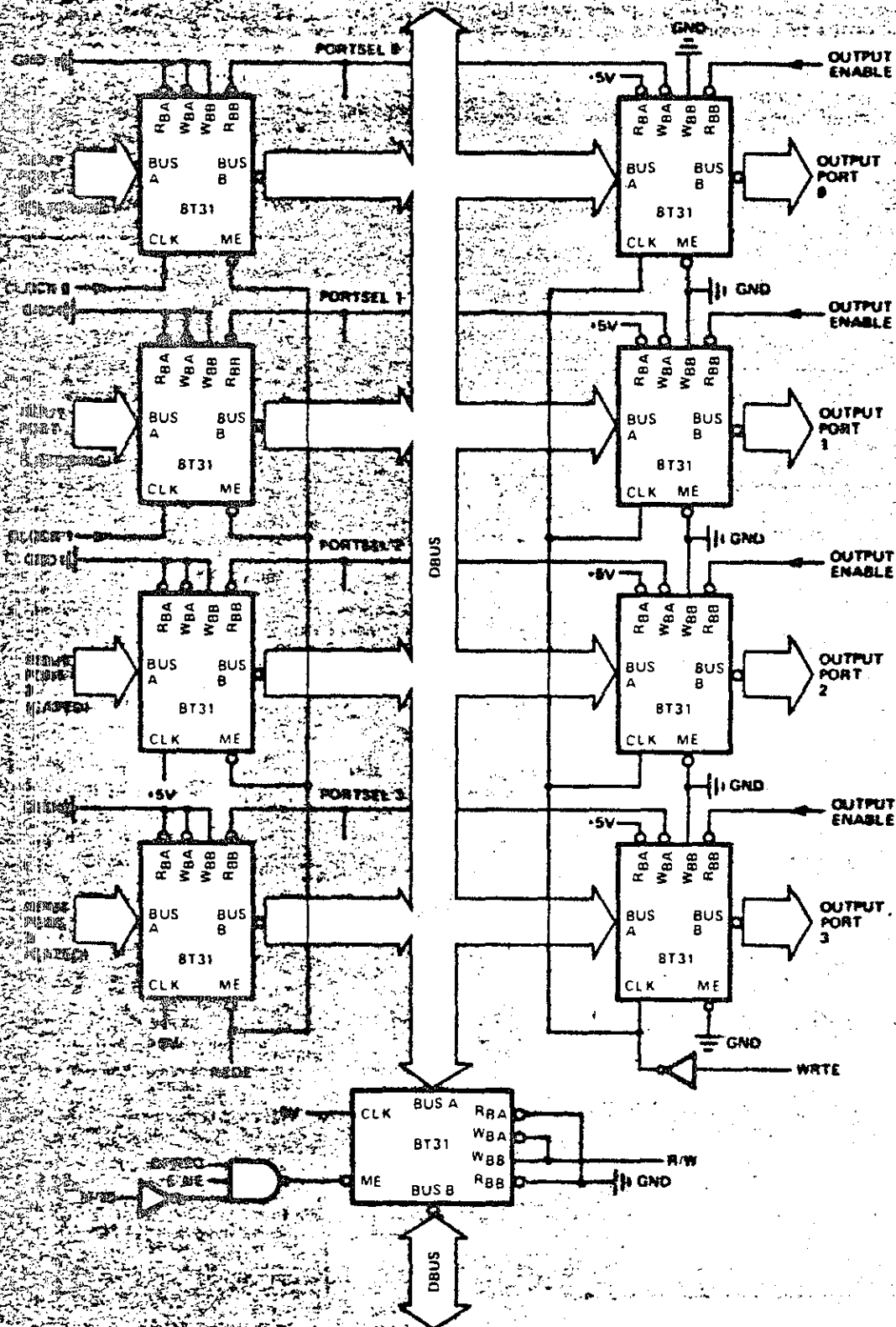


Figure 31

write operations from peripheral and CPU. The bidirectional I/O port concept is illustrated in Figure 32.

In many industrial applications, such as process control, single bit inputs and outputs are used to monitor switches and detectors or to drive relays and lamps. A possible solution for such an 8-bit flag register would be an 8-bit output port and a memory byte reserved as a flag register in the system's RAM. The setting, resetting, or testing of individual bits with this method of implementing a flag register requires many bytes of program memory. The output port and the memory location reserved as a flag register image must be updated after each bit operation.

The 8T31 can be used to implement a flag register without the use of a memory byte in the system's RAM. No additional hardware is required, and the saving in program memory bytes for flag operations is considerable. A logic diagram of this application is given in Figure 33. Listings of basic software to set, reset, and test individual flags for both positive and negative true outputs are given in Figures 34 and 35.

THE 8T31 USED AS A BIDIRECTIONAL I/O PORT

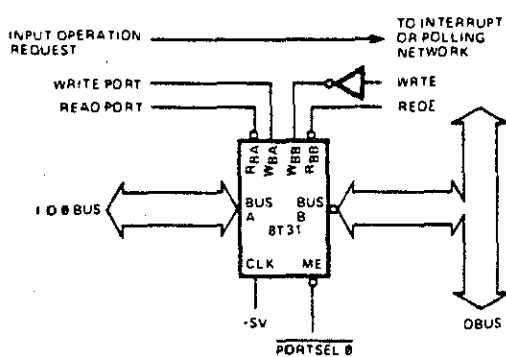


Figure 32

A FLAG REGISTER IMPLEMENTED WITH THE 8T31

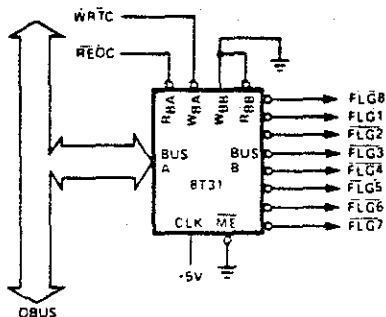


Figure 33

TWIN ASSEMBLER VER. 1.0

PAGE 0001

LINE ADDR OBJECT E SOURCE

```
0001 * PD768090
0002 *****
0003 *
0004 * **** FLAG MANIPULATION EXAMPLES ****
0005 *
0006 * THIS LISTING GIVES SOME EXAMPLES HOW TO SET, RESET
0007 * AND TEST INDIVIDUAL BITS OF AN EXTERNAL FLAG REGISTER
0008 * BUILT WITH THE 8T31 BIDIRECTIONAL I/O PORT
0009 * INSTRUCTIONS ARE GIVEN FOR BOTH ACTIVE 'HIGH' AND
0010 * ACTIVE 'LOW' OUTPUTS
0011 *
0012 *****
0013 *
```

```
0014
0015
0016 0000
0017 0001
0018 0002
0019 0003
0020 0000
0021 0001
0022 0000
0023
0024 0001
0025 0002
0026 0004
0027 0000
0028 0010
0029 0020
0030 0040
0031 0080
0032
0033 0000
0034 0050
0035
0036
0037
0038
0039
0040 0000
0041
0042
0043
0044 0500 30
0045 0501 6404
0046 0503 00
0047
0048 0504 30
0049 0505 6400
0050 0507 00
0051
0052
0053
0054 0500 30
0055 0509 44F0
0056 0508 00
```

#### DEFINITIONS OF SYMBOLS

```
*
R0 EQU 0 PROCESSOR REGISTERS
R1 EQU 1
R2 EQU 2
R3 EQU 3
Z EQU 0 BRANCH COND ZERO
UN EQU 3 UNCONDITIONAL
RL EQU 0 ALL BITS ARE 1
*
FLG0 EQU '01' FLAG 0
FLG1 EQU '02' FLAG 1
FLG2 EQU '04' FLAG 2
FLG3 EQU '08' FLAG 3
FLG4 EQU '10' FLAG 4
FLG5 EQU '20' FLAG 5
FLG6 EQU '40' FLAG 6
FLG7 EQU '80' FLAG 7
*
ONE EQU '0600' DUMMY ADDRESS OF ROUTINE 'ONE'
ONES EQU '0650' DUMMY ADDRESS OF ROUTINE 'ONES'
*
```

#### INSTRUCTIONS FOR ACTIVE 'LOW' OUTPUTS

```
*
ORG '0550'
*
SET FLAG(S)
*
SAFG RECD:R0 LOAD FLAG REGISTER IN R0
IORI:R0 FLG2 SET FLAG 2
WRTC:R0 RESTORE FLAG REGISTER
*
SAFS RECD:R0
IORI:R0 FLG5+FLG6 SET FLAGS 5 AND 6
WRTC:R0 RESTORE
*
RESET FLAG(S)
*
RAFG RECD:R0
ANDI:R0 'FF'-FLG2 RESET FLAG 2
WRTC:R0 RESTORE
```

Figure 34

TWIN ASSEMBLER VER. 1.0

PAGE 0002

LINE ADDR OBJECT E SOURCE

```
0058 0500 30 RAFFS RECD:R0
0059 0500 44F0 ANDI:R0 'FF'-FLG5-FLG6 RESET FLAGS 5 AND 6
0060 0500 00 WRTC:R0 RESTORE
*
TEST FLAG(S)
*
TAFFS RECD:R0
TNI:R0 FLG2 TEST FLAG 2
BCFA:R0 ONE BRANCH IF ONE
*
TAFFS RECD:R0
TNI:R0 FLG5+FLG6 TEST FLAGS 5 AND 6
BCFA:R0 ONES BRANCH IF BOTH ARE ONE
*

INSTRUCTIONS FOR ACTIVE 'HIGH' OUTPUTS
*
ORG '0550'
*
SET FLAG(S)
*
SAFH RECD:R0
ANDI:R0 'FF'-FLG2 SET FLAG 2
WRTC:R0 RESTORE
*
SAFHS RECD:R0
ANDI:R0 'FF'-FLG1-FLG4 SET FLAGS 1 AND 4
WRTC:R0 RESTORE
*
RESET FLAG(S)
*
RAFH RECD:R0
IORI:R0 FLG2 RESET FLAG 2
WRTC:R0 RESTORE
*
RAFFS RECD:R0
IORI:R0 FLG1+FLG4 SET FLAGS 1 AND 4
WRTC:R0 RESTORE
*
TEST FLAG(S)
*
TAFFH RECD:R0
TNI:R0 FLG2 TEST FLAG 2
BCFA:R0 ONE BRANCH IF ONE
*
TAFFHS RECD:R0
TNI:R0 FLG1+FLG4 TEST FLAGS 1 AND 4
BCFA:R0 ONES BRANCH IF BOTH ARE ONE
*
END 0
```

TOTAL ASSEMBLY ERRORS = 0000



D I G I T A L   T O   A N A L O G  
A N D  
A N A L O G   T O   D I G I T A L  
C O N V E R T E R  
  
T D  
  
M I C R D P R D C E S S O R  
  
I N T E R F A C E   T E C H N I Q U E S

A paper by:

Alan J. Weissberger  
Narpat Bhandari

© 1977

Signetics Corporation  
811 E. Arques Avenue  
Sunnyvale, CA, 94086

- Introduction
- The Basic  $\mu$ P-Convertor Interface
- Microprocessor to D/A
- The NE5018 - 8 Bit System DAC
- A/D to Microprocessor
- The NE5030/5031 8 Bit Triple Slope Integrating A/D
- The NE5034 8 Bit High Speed A/D
- The NE5537 Sample & Hold Amplifier
- A/D Systems Considerations
- Data Acquisition System Configurations
- Conclusions

## MONOLITHIC CONVERTERS AND MICROPROCESSOR INTERFACE

Harpat Bhandari, Alan J. Weisberger

Signetics Corporation

811 East Arques Avenue, Sunnyvale, California 94086

### ABSTRACT

The growing use of microprocessors in analog control and data acquisition systems has created a need for "micro" Analog-to-Digital (A/D) and Digital-to-Analog (D/A) converters. The semiconductor industry's response to this need has been the development of monolithic converter chips. This paper surveys several currently available bus-compatible monolithic converters and examines data acquisition system configurations and microprocessor interface techniques using these products.

### INTRODUCTION

The ever-increasing popularity of microprocessors has stimulated the growth of conversion circuitry necessary to communicate with the analog world. Various data acquisition modules and boards are available to serve the needs of the user who wishes to purchase a pre-designed, packaged system for his mini or microcomputer. Unfortunately, many of these systems are often too expensive, too large, or consume too much power for the user who wishes to extend the advantages of micro circuitry throughout his system.

New monolithic Digital-to-Analog (D/A) and Analog-to-Digital (A/D) integrated circuits can minimize system cost, power, and size, while providing reliability and moderate speed. Monolithic devices can be viewed as small building blocks to be incorporated in a data acquisition system that can be configured without constraints. Microprocessor compatibility enables the converter logic to be controlled by software and makes data transfers bus oriented. Minimal external circuitry is required for system operation.

### THE BASIC MICROPROCESSOR-CONVERTER INTERFACE

The most popular means of transferring data between a microprocessor and an A/D or D/A converter is as a parallel word over the microprocessor's data bus. The objective is to read the A/D or load the D/A in the shortest time using the fewest instructions and minimal peripheral hardware.

Information on the microprocessor address and control bus is decoded and used to select the A/D or D/A converter. Data is then transferred on the system data bus from the tri-state

output of the A/D or to the input latch of the D/A. One of two instruction types is used to access or load data:

1. An input/output (I/O) instruction transfers data between the microprocessor's accumulator and the converter.
2. The converter appears as a memory location, with data being fetched from the A/D or stored in the D/A.

The latter approach appears more effective for the following reasons:

- Memory reference instructions are faster than conventional I/O instructions.
- No separate I/O bus is required.
- All memory reference instructions can be used. In particular, the converter data can be loaded, added or subtracted from or to any accumulator.
- A very large number of ports become available with a wide range of addressing options dependent on the microprocessor's instruction set.

If the converter word length is greater than the width of the microprocessor's data bus, the data must be read or loaded in groups. For example, a 12-bit converter interfaced to an 8-bit processor requires two data transfers over the 8-bit data bus. Care must be taken to hold the A/D input or D/A output stable when reading from or writing to the appropriate converter.

### MICROPROCESSOR TO D/A

Each analog output channel under  $\mu P$  control will normally have its own D/A converter. Low cost and system simplicity dictate this choice. The  $\mu P$ -compatible D/A should have the following characteristics:

1. An input latch to capture the digital data to be converted and hold it stable.

2. A Latch Enable input or Chip Select and Write Strobe inputs for loading the latch. These input(s) are active when the  $\mu P$  addresses the D/A and the digital data is present on the system data bus.
3. Low input current loading on all digital inputs.
4. Low power dissipation.
5. Internal voltage reference and internal output buffer amplifier.
6. Converter accuracy to within  $\pm 1/2$  least-significant bit.

Additional capabilities might include:

1. Double-buffered latched inputs when the  $\mu P$  bus width is less than the resolution of the D/A and the  $\mu P$  must load the D/A in groups. This ensures that the analog output switches synchronously and cleanly from one programmed voltage to another.
2. Capability to read back the digital data for software control routines. If not provided in hardware and read back is required, the program must maintain an "image" of the D/A contents in memory.

#### THE NE5018 8-BIT-SYSTEM DAC

The NE5018 is a complete monolithic 8-bit Digital-to-Analog Converter (DAC) built in bipolar technology with a proprietary low temperature coefficient ion implant process. The device is designed for use with the Signetics 2650, the 8080A, and other microprocessors and has all characteristics described above in items 1 through 6.

The NE5018, shown in Figure 1, contains an 8-bit data latch, stable voltage reference, high-speed output Op-Amp, and an 8-bit D/A converter. Combining all of these elements on a single chip gives a system designer considerable flexibility in putting together a high performance and low-cost system. The NE5018 requires a minimum of external components and operates from standard  $\pm 15$  VDC power supplies. The output analog voltage range can be programmed from 0V to +10V for unipolar operation or from -5V to +5V for bipolar operation. The device has been described earlier and the main features of the NE5018 DAC are summarized below:

- 8-bit resolution
- Relative accuracy: 0.2% Max.

- Output voltage: 0 to 10V or  $\pm 5$ V
- Output setting time: 2  $\mu s$  to 0.2% of F.S.
- Full scale temperature coefficient:  $\pm 10$  ppm/°C
- $\overline{CE}$  pulse width: 250 ns (TTL compatible)
- Data inputs: TTL compatible (input current required only 2  $\mu A$ )

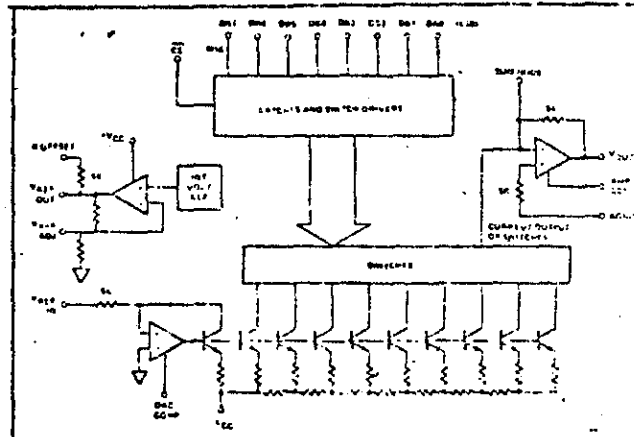


Fig. 1. NE5018 Block Diagram 8 Bit System DAC

#### A/D TO MICROPROCESSOR

The Analog-to-Digital (A/D) converter is a more complex device than the D/A. There is more internal circuitry and more external control required. The main considerations in selecting A/D converters for microprocessors are:

1. The A/D should have accuracy, linearity, speed and other performance parameters similar to conventional A/Ds.
2. A STRT input that can be controlled by the microprocessor.
3. An End of Conversion (EOC) output that informs the microprocessor when data can be read.
4. An output latch to hold converted data stable.
5. A Chip Enable ( $\overline{CE}$ ) input to control the placement of converted data onto the data bus. At all other times the data should be tri-state.
6. Good data bus current drive; otherwise additional tri-state buffers will be required.
7. Internal clock with provisions for overriding with an external clock.

8. On-chip voltage reference.

9. An over-range (OVR) indicator that informs the microprocessor that the analog input is out of range.

#### A/D Converter Types

There are basically two types of monolithic Analog-to-Digital (A/D) converters:<sup>2</sup> the integrating type and the successive approximation type. The integrating converter uses a slope technique. Its advantages include inherent accuracy, non-critical components, excellent noise rejection, and no missing codes. However, the integrating type of converter is slow. The typical conversion speed is 10 to 100 readings per second.

While the successive approximation type of converter is quite fast, its circuitry has several critical components, tough timing requirements, and it can have missing codes. A well-designed successive approximation type of converter is excellent for high-speed applications such as data acquisition and control systems.

Both of these converter types are designed specifically for microprocessors. The following sections describe both types in more detail.

#### NE5030/5031 8-BIT TRIPLE SLOPE INTEGRATING A/D

The NE5030/5031 is a complete 8-bit bipolar monolithic Analog-to-Digital converter subsystem currently under development by Signetics. This device uses a unique triple slope integration technique for conversion as opposed to single or dual slope. The triple-slope technique eliminates errors due to comparator offsets, drifts, etc., and allows the device to operate from a single 5V power supply. The block diagram of the device, shown in Figure 2, has all of the necessary characteristics to operate in a microprocessor environment. The analog input voltage and the reference input voltage are high-impedance amplifiers that provide easy interfacing with analog systems. The input range for both inputs is 0-2V. The 8-bit parallel data output bus uses tri-state buffers. The output is straight binary to facilitate direct microprocessor compatibility. A chip enable ( $\overline{CE}$ ) control line, when low, puts the data on the bus. The EOC and OVR flags are open-collector outputs. The STRT command initializes the device and starts the new conversion. At the end of the conversion, EOC goes high. If the input data is more than full scale, OVR also occurs. During the conversion, the output data will not be displayed even if the  $\overline{CE}$  is low. The internal logic inhibits the data from being displayed on the bus during the conversion. A new STRT command

during the conversion will reinitialize the device and start the conversion all over again.

Both devices have all the elements of a complete A/D converter on a single chip. The devices are available in 18-pin dual in-line packages. Only two external capacitors are required for a complete 8-bit A/D. The NE5030 provides an additional on-chip voltage reference while the NE5031 provides an OVR flag. All other functions are the same on both devices.

#### Triple Slope Operating Principles

The third slope avoids switching of the capacitor voltage to zero at the start of the conversion. This slope also eliminates errors caused by input offsets of the comparator, because the comparator switches twice at a predetermined level. By adding the third slope, the dynamic range is reduced by a few hundred millivolts depending on the level set at the comparator. The triple slope technique also keeps the circuit simple and straightforward and minimizes the need for controls and logic. The operating principle of the triple slope integrating A/D converter is illustrated in Figures 3 and 4.

At the start of a conversion,  $I_{REF}$  is turned off and  $I_{CLAMP}$  is turned on. The capacitor starts charging through  $I_{CLAMP}$ . When the voltage at the capacitor reaches  $V_{CLAMP}$ , the comparator goes high. At this time the counter starts counting and also  $I_{CLAMP}$  is turned off and  $I_{IN}$  is turned on. The capacitor continues to charge for a fixed 256 counts, i.e., until the counter overflows. When the OVF arrives,  $I_{IN}$  is turned off, and  $I_{REF}$  is turned on. This allows the capacitor to discharge through a fixed reference source. When the voltage at the capacitor equals  $V_{CLAMP}$ , the comparator goes low. This, in turn, stops the counter, and the contents of the counter is equal to the input voltage. At this time, reference is left on so that the capacitor is allowed to discharge. This allows the comparator to remain low for the subsequent conversion cycles.

Triple slope devices are designed as general purpose, 8-bit A/D converters for bus-oriented data acquisition and control systems. The main features of these devices are summarized below:

- 8-bit resolution
- Accurate to  $\pm 1/2$  LSB
- Triple Slope Integration
- Analog input range 0 - 2V

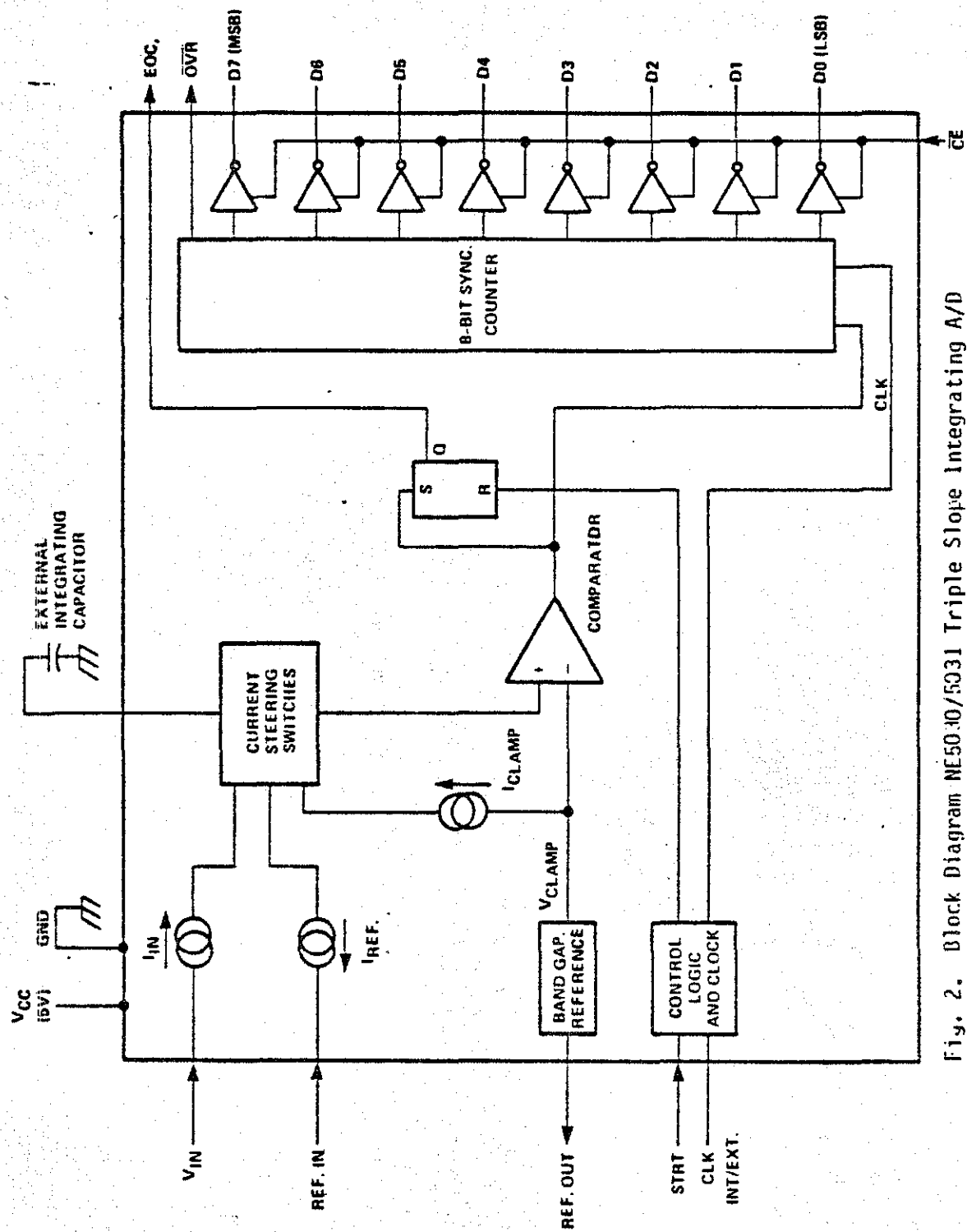


Fig. 2. Block Diagram NE5010/5031 Triple Slope Integrating A/D

- 2-bit parallel tri-state output data
- EOC and  $\overline{OVR}$  - open collector outputs
- STRT and  $\overline{CE}$  - TTL inputs
- On-chip clock, voltage reference
- Provision for external clock
- Single 5V power supply
- Low power dissipation - 100 mW
- 18-pin DIP

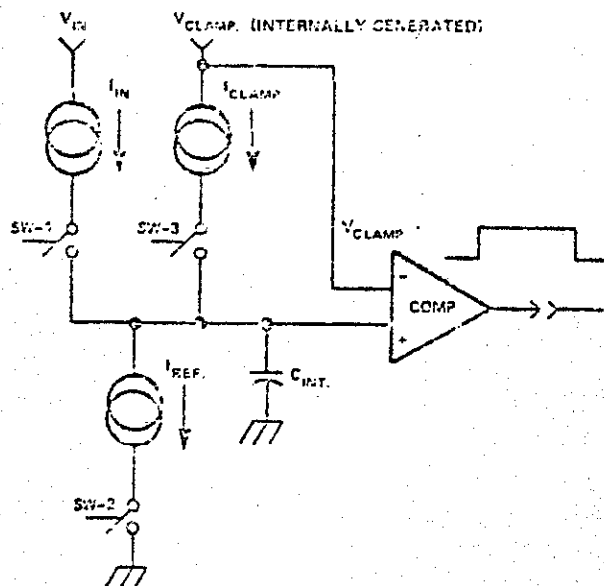


Fig. 3. Simplified Description of Triple Slope Integration

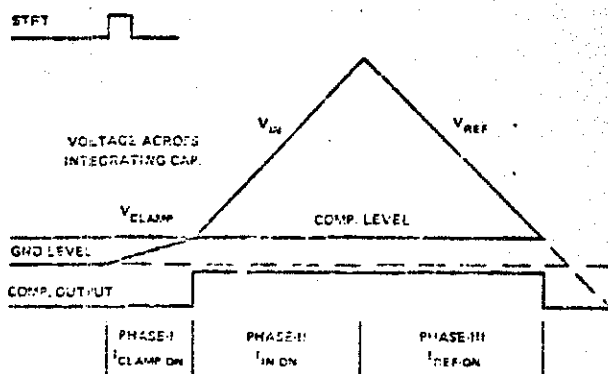


Fig. 4. Voltage Across  $C_{INT.}$  (Integrating Capacitor) and Comparator Output

### NE5034 8-BIT HIGH-SPEED A/D CONVERTER

The NE5034, currently under development at Signetics, is an 8-bit Analog-to-Digital converter subsystem built in high voltage linear I<sup>2</sup>L proprietary process. The device, shown in Figure 5, uses a Successive Approximation Register (SAR), an 8-bit D/A, and a comparator for the conversion. It also has a built-in clock with provisions for the external clock. Typical conversion time is approximately 40  $\mu$ s.

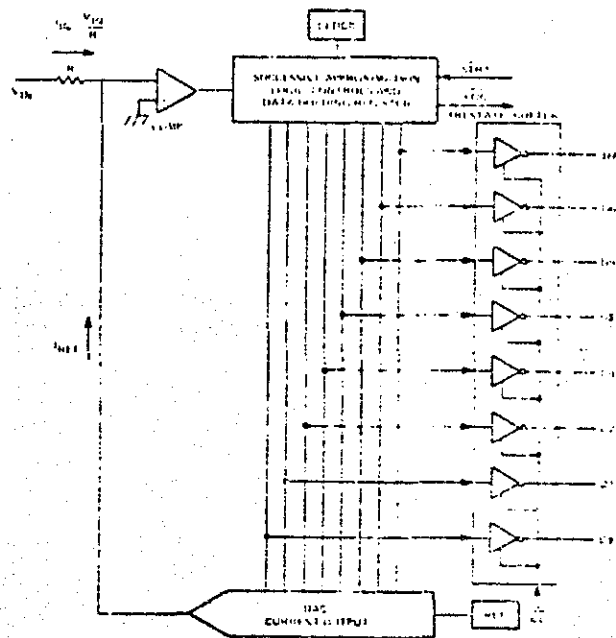


Fig. 5. NE5034 - 8 Bit High Speed A/D

### Principles of Operation

At the start of the command, a series of clock pulses is generated in a Successive Approximation Register (SAR). The first clock pulse initializes the internal logic, resets all flip-flops, and readies for the conversion. The leading edge of the next clock pulse turns on the most-significant bit (MSB) of the D/A converter. This current is compared with the  $V_{IN}$  signal current ( $V_{IN}/R$ ). Depending on the magnitude of the input signal current, the comparator either goes high or remains low during this clock period. The information in the comparator must settle down long before the trailing edge of the pulse. At the trailing edge, the information in the comparator is clocked into the data holding register. If the input signal current is greater than the current output of the MSB (approximately half of full scale), then the data holding register will

remain latched. At the third clock pulse the next most-significant bit is turned on and tried in a similar way. If the comparator is low at the trailing edge of the third clock pulse, the data holding register for this bit will reset and remove the D/A converter output from the comparator input. This trial process continues for all eight bits. At the end of all trials, an EOC signal is sent out, and the information in the data holding register is equivalent to the input signal  $V_{IN}$ . For clarification, the trial mechanism is shown in Figure 6.

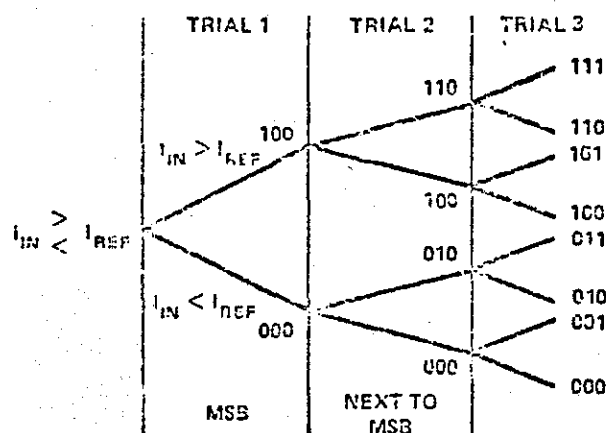


Fig. 6. Trial Mechanism for NE5034 A/D

The main features of the NE5034 A/D converter are:

- 8-bit resolution
- Accurate to  $\pm 1/2$  LSB
- Successive Approximation type
- Conversion time - 40  $\mu$ s
- Linear  $I^2L$  process
- Analog input range - 0-10V
- On-chip clock, with provisions for an external clock
- 8-bit parallel data output (tri-state)
- Power supplies: +5V, -12V
- 18-pin DIP

#### NE5537 SAMPLE AND HOLD AMPLIFIER

The NE5537 monolithic Sample and Hold Amplifier combines the best features of ion implanted JFET's with bipolar devices to obtain high accuracy, fast acquisition time, and low droop rate. This device is pin compatible with the LF198, and features a superior performance in droop rate and output drive capability. The circuit, shown in Figure 7,

contains two operational amplifiers which function as a unity gain amplifier in the sample mode. The first amplifier has bipolar input transistors which gives the system a low offset voltage. The second amplifier has JFET input transistors to achieve low leakage current from the hold capacitor. A unique circuit design for leakage current cancellation using current mirrors gives the NE5537 a lower droop rate at higher temperature. The output stage has been redesigned to drive a 2K load. The logic input has a high-impedance buffer and is compatible with TTL, PMOS or CMOS logic. The differential logic threshold is 1.4V with the sample mode occurring where the logic input is high. The main features of the NE5537 are listed below:

- Acquisition time: 10  $\mu$ s
- Offset voltage: 1 mV (typical)
- Hold leakage current:  $I_{ns}$  @ 125°C
- Gain Accuracy: 0.002% with  $R_L = 2K$
- Hold step error: 0.5 mV,  $C_{hold} = 0.01 \mu$ F

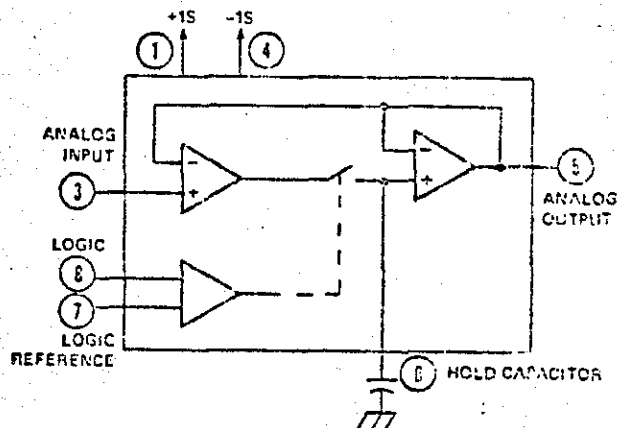


Fig. 7. NE5537 Block Diagram Sample & Hold Circuit

#### A/D SYSTEMS CONSIDERATIONS

The A/D presents many interfacing possibilities for the system designer. With monolithic A/D's, the designer must determine:

1. Whether to multiplex analog input channels or provide an A/D per channel. If analog inputs are multiplexed, the analog channel selection and analog set-up time must be controlled by software before starting the A/D.
2. Whether to start the conversion (SOC) under  $\mu$ P control or leave the converter free-running.



3. Whether to wait for the A/D End of Convert (EOC) signal in a software interrogation loop or to request an interrupt when EOC is active.
4. How to respond to interrupts if this mechanism is chosen to signal the  $\mu P$  to read the A/D. Specifically, the interrupting device must be identified, priorities must be resolved if there is more than one interrupt, the data must be taken, and the interrupt must be reset.
5. What software routines will be needed for processing the converter data and handling over-range or out-of-limit values.

#### DATA ACQUISITION SYSTEM CONFIGURATIONS

Three important configurations for data acquisition and analog control will be examined.

1. The multiplexed random-channel-addressed system uses I/O instructions and software control.
2. Parallel A/D converters (one per analog input channel) are addressed with memory mapped I/O instructions. The A/D's are started under  $\mu P$  control and generate interrupts on the End of Convert (EOC) signal.
3. Parallel A/D Converters that generate interrupts only on the detection of new data. These A/D's are started under the control of an external scan time oscillator.

#### System 1

In the multiplexed channel system, a 2650  $\mu P$  generates and detects all A/D and O/A control signals and transfers converter data with a minimum of external logic. This configuration is most effective when several high-speed analog inputs are to be read by the  $\mu P$ .

A combination of extended and non-extended I/O instructions "isolated I/O" are used to directly address three D/A converters and 16 analog inputs through an analog multiplexer, sample and hold circuit (S&H), and a successive approximation (SA) A/D converter. Interrupts are not necessary, since A/D settling time is fast (20 to 40  $\mu sec$ ), and this enables the 2650 to interrogate the EOC signal in a software loop.

The operating sequence for this system is:

1. The 2650 REDE instruction addresses an analog input channel while simultaneously reading the A/D output of the previous channel. The four least significant address bits of the extended I/O read instruction (REDE signal) contain the channel number.
2. The analog multiplexer selects the desired channel which is held stable in the 74L75 latch.
3. The 5537 S&H circuit acquires the analog signal in the sample mode.
4. The 2650  $\mu P$  starts the conversion under software control by pulsing its flag output which places the 5537 in the hold mode.
5. The 5034 A/D converter digitizes the analog input and activates the EOC signal when the conversion is complete.
6. The 2650  $\mu P$  senses EOC on its sense input and issues the extended I/O read instruction to obtain the converted data and select the next channel. The data is stored in memory by the software while the hardware goes to step 2.

The D/A converters are addressed as three dedicated "write-only" ports. Data to be converted is output using the WRTC, WRTO, and WRTS instructions. Note that REDE, REDD signals are available to control two additional input ports. Other I/O devices could be memory-mapped.

#### System 2

While the random-addressed system may be the simplest to interface with most  $\mu P$ 's, a parallel conversion system can be more cost effective, particularly if the number of input channels is low or the process variables change slowly.

This system uses one A/D per input channel while eliminating the analog multiplexer and S&H. If high sampling rates are not required, a low speed, low-cost integrating converter such as NE5030/5031 can be used. The A/D's should have tri-state outputs and chip enable inputs to control the placing of converter data on the system data bus.

With conversion times on the order of 1.8 to 40 milliseconds, the  $\mu P$  cannot be preoccupied with monitoring the EOC signal. Consequently, an interrupt should be requested when the conversion is complete and the

resulting data is ready to be read. The A/D's start convert input is controlled by the  $\mu$ P interrupt service routine.

In the example shown, the A/D and D/A converters are treated as memory-mapped devices by an 8080A  $\mu$ P. Address bit 15 (A15), the most-significant address bit, is the I/O Control Flag that distinguishes between memory and I/O devices. Address bit 14 (A14) distinguishes peripheral I/O from data acquisition I/O, while address bit 3 (A3) differentiates A/D addresses from D/A addresses. This permits each A/D and D/A to be treated as a true Read/Write memory location by the software. An address map of this system is given in Table 1:

TABLE 1: ADDRESS MAP FOR SYSTEM 2

| Bytes        | A15 | A14 | A13 | A3 | A2-A0                                           |
|--------------|-----|-----|-----|----|-------------------------------------------------|
| 16K ROM      | 0   | 0   | X   | X  | X                                               |
| 16K RAM      | 0   | 1   | X   | X  | X                                               |
| 16K Parallel | 1   | 0   | X   | X  | X                                               |
| 16K A/D      | 1   | 1   | X   | 0  | Read Interrupt Status = 0<br>Selected A/D = 1-7 |
| D/A          | 1   | 1   | X   | 1  | Write SOC register = 0<br>Selected D/A = 1-7    |

The sequence of operations for this system is:

1. The 8080A loads a hex register (74LS174) with the individual Start of Convert (SOC) signals for each 8030 A/D converter. Since the internal clock of the 8030 A/D converter is used, the SOC pulse width must be at least 80 nsec wide. The  $\mu$ P can cyclicly shift a bit in an internal register to generate SOC pulses for each A/D in turn, or it could start several conversions simultaneously.
2. When an A/D completes a conversion, the EOC signal clocks a "1" into an interrupt request flip-flop (7474). The Q outputs of these flip-flops are logically OR'd together to form a single Interrupt Request (IR) input to the 8080A. Since there are no other interrupts in this system, the 8228 bus controller has been configured to generate a RST 7 interrupt vector in response to IR. This forces the program to

location 56 when the 8080A recognizes the interrupt.

3. The interrupt service routine must save the machine state on the stack through a series of PUSH instructions.
4. The interrupting A/D must be identified. The interrupt service routine executes a "Read Interrupt Status" instruction which activates the R15 output of a decoder (74LS138). This enables the interrupt status of all A/D's to be gated onto the system data bus and into the 8080A's accumulator. The interrupt service routine tests each bit to determine which A/D is requesting service. The order of bit testing determines the interrupt priority.
5. The converted data is accessed through a "Read A/D" instruction for converter i (RADi signal). The converted data is read into the 8080's accumulator and the corresponding interrupt request flip-flop is reset when RADi is active. It might be desirable to have this same instruction initiate the next conversion by tying RADi to the positive edge trigger of a one-shot which would produce the next SOC pulse.
6. The data is processed and stored in memory after which the next SOC sequence is output to the hex latch.
7. To exit from the interrupt service routine the software must restore the machine state through POP instructions, enable future interrupts (EI instruction) and return (RET instruction) to the main program.
8. Analog outputs are generated by the D/A converters which are loaded by "Write D/A" instructions.

### System 3

If either a large number of process variables are to be monitored or a large amount of data processing is required, then a busy  $\mu$ P might lose interrupts in a parallel channel system. If the analog inputs change slowly, logic can be added to off load the  $\mu$ P and thereby increase the effective capacity of the system.

In the configuration shown, the A/D's interrupt the  $\mu$ P only when different data has been converted. Selection of the  $\mu$ P and interrupt architecture is left to the reader. The system operates in the following way:

1. The START convert pulses are generated by an external scan-time oscillator whose output may be phased or sent to all A/D's simultaneously. This frees the  $\mu$ P from SCC pulse generation. The scan-time period is chosen based on the rate change of the analog variables, the A/D conversion time, and the number of interrupts to be serviced.

2. An octal D-type register, initialized by a system reset (RESET) pulse, stores each A/D output read by the  $\mu$ P. When data is converted, a hardware comparison is made between the old and new data. Two exclusive NOR gates with open-collector outputs (74LS266) are WIRE OR'd to generate a compare signal. If that signal is false and the EOC signal is present, then an interrupt request is generated through a flip-flop.

3. The interrupt priority mechanism could be implemented through an MSI priority encoder and mask-erale register or an LSI interrupt chip.

4. The interrupt must be recognized and identified within the scan-time period; otherwise, a new conversion could be taking place when the  $\mu$ P reads the A/D. EOC should be true just before converter data is read.

5. The interrupt service routine executes a "Read A/D" instruction which:

- a) reads the new converted data;
- b) transfers the new data into the octal register; and
- c) resets the interrupt request flip-flop.

All other interrupt processing is similar to System 2.

6. A 5031 A/D Converter provides an over-range output (OVR) which might be a separate  $\mu$ P interrupt condition.

#### CONCLUSIONS

The converter chips discussed are typical of the new breed of monolithic devices for data acquisition. As shown in the previous examples, these converters can interface directly with 8-bit  $\mu$ P's in a variety of system configurations with minimal external components.

With continued reduction in converter prices and concomitant increases in accuracy and speed, the parallel channel type systems will become more popular. D/A's and A/D's in this environment are treated as read/write memory locations by the software. Interface control is integral to the converter chips with device selection logic consisting of standard MSI decoders.

#### REFERENCES

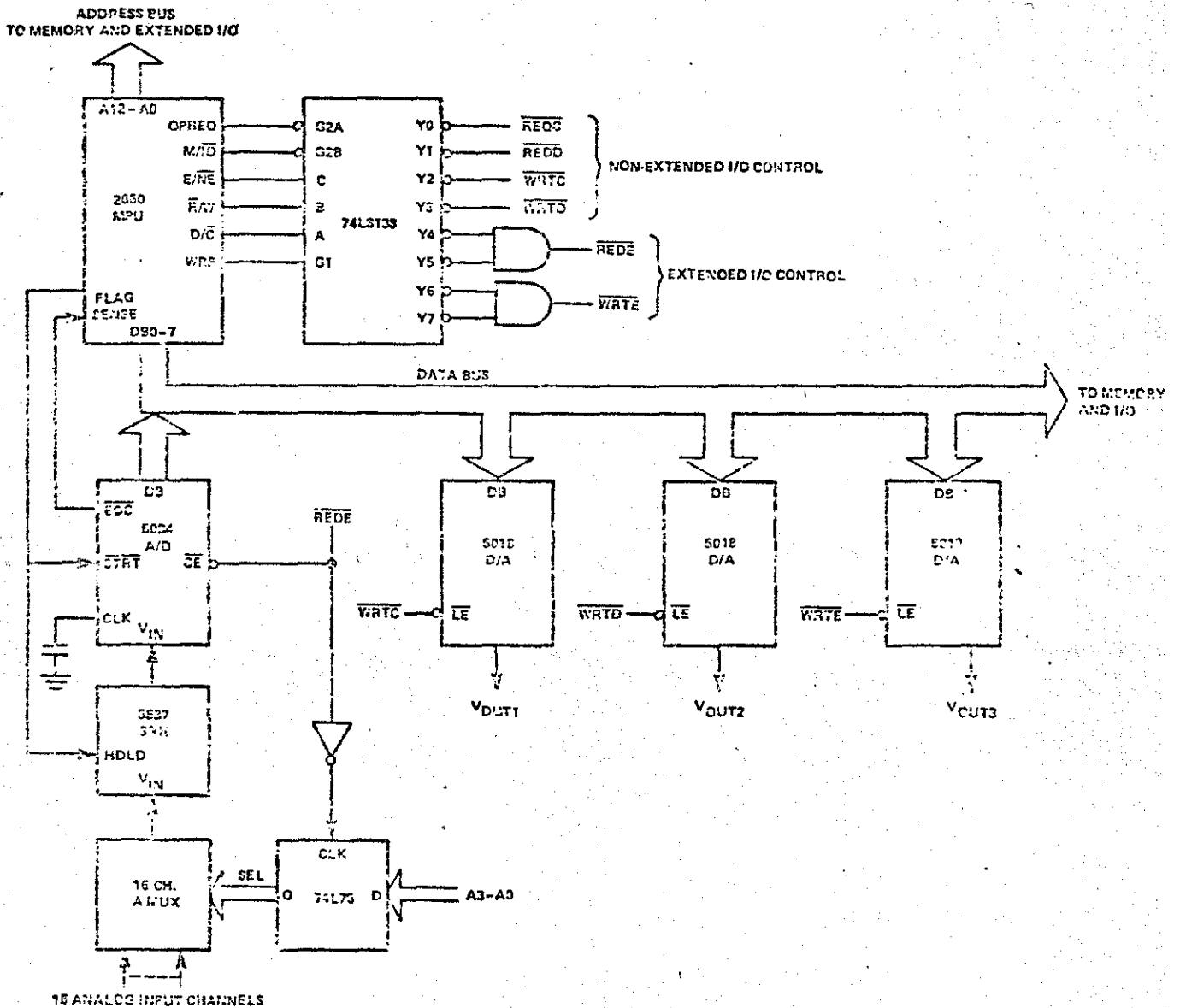
1. Signetics App. Note - 5018.
2. "Selection Criteria for A-D and D-A Converters" Sidney Davis, Associate Editor, Computer Design, Sept. 1972, pp. 67-79.
3. "A/D and D/A Converters: Do you know what the parameters mean?" Roger Allen, EDM, Dec. 15, 1972, pp. 18-23.

#### ACKNOWLEDGEMENT

The authors would like to thank R. Lovelace and K. N. Sundaram for their helpful suggestions in device design and H. Luft for typing the manuscript.

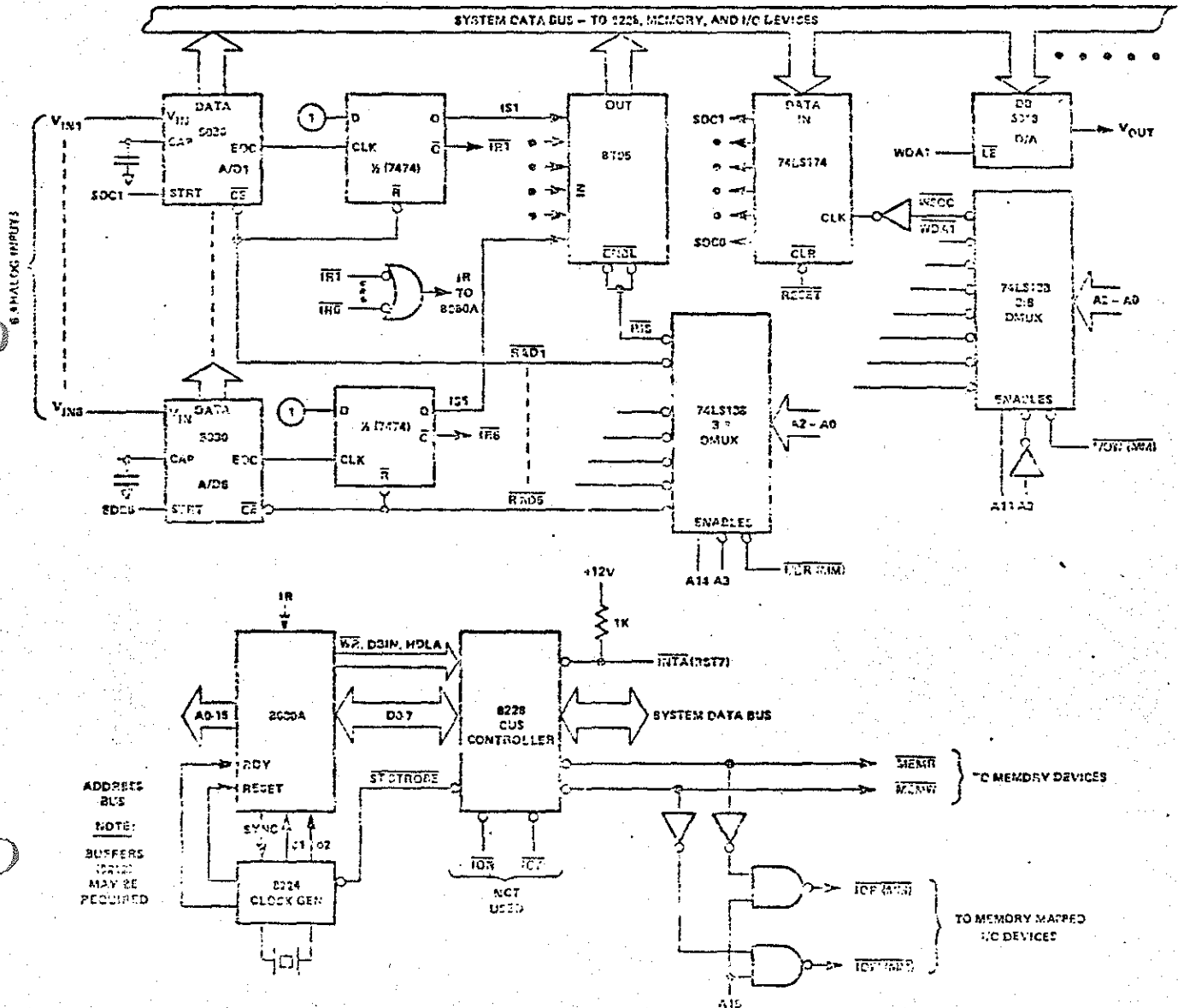
# SYSTEM 1

2650 MPU - ISOLATED I/O, 16 MULTIPLEXED ANALOG INPUTS, HIGH SPEED S.A. A/D, 3 D/A'S



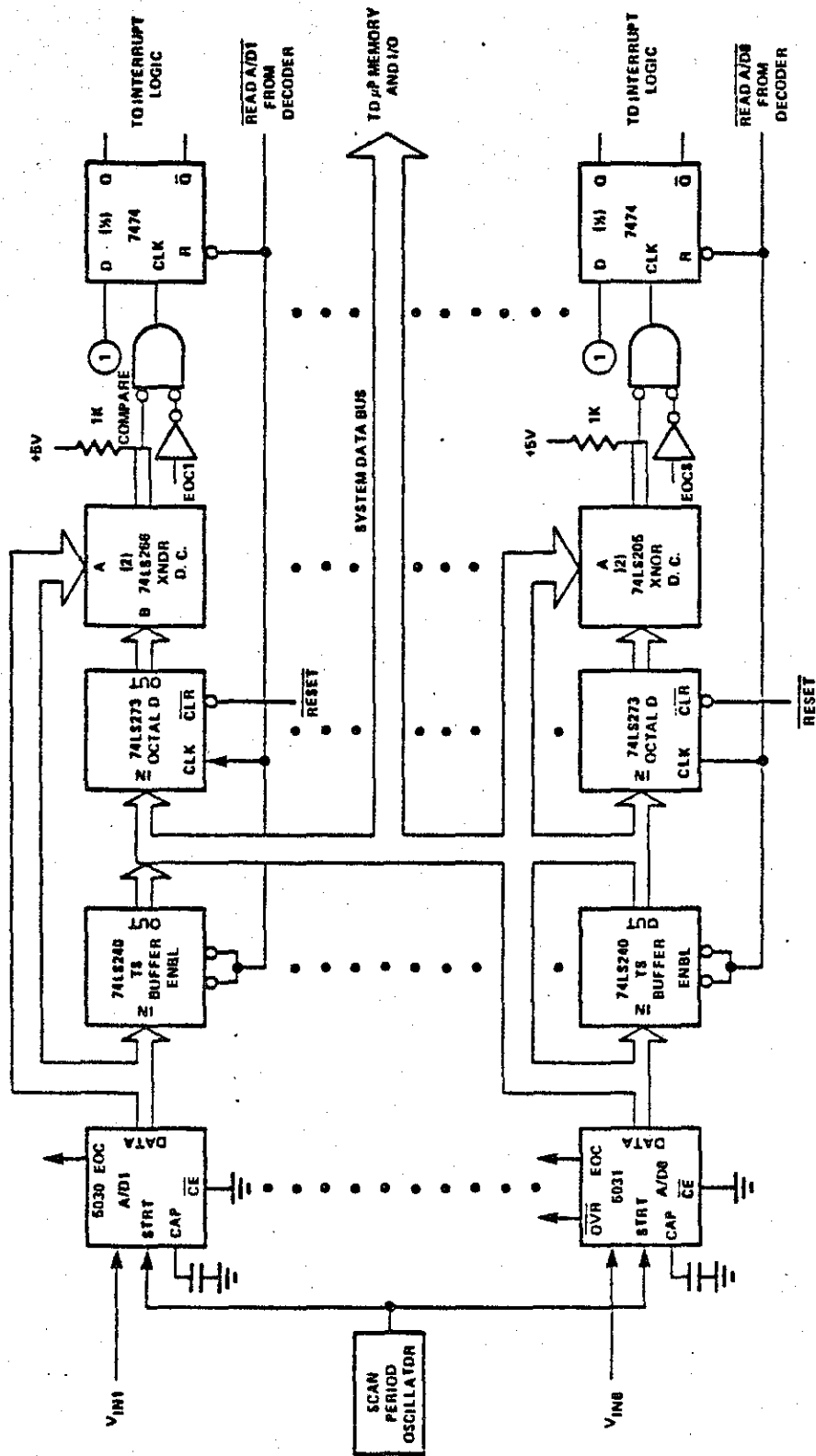
## SYSTEM 2

PARALLEL A/Ds INTERRUPT 8080A ON EOC - 1. D/A's ARE CONTROLLED BY WOA/SIGNALS.



### SYSTEM 3

A/D<sup>1</sup> INTERRUPT  $\mu$ P ONLY ON NEW DATA. 5030 A/D SUPPLIES OUTPUT  $V_{REF}$ . 5031 A/D SUPPLIES OVERANGE (OVR) OUTPUT.



# ANALOG TO DIGITAL CONVERSION - SUCCESSIVE APPROXIMATION METHOD

## INTRODUCTION:

Measurement of an unknown analog value (voltage or current) is fairly easy to implement in 2650 software via SUCCESSIVE APPROXIMATION. A minimum of hardware (Diagram 1) is required.

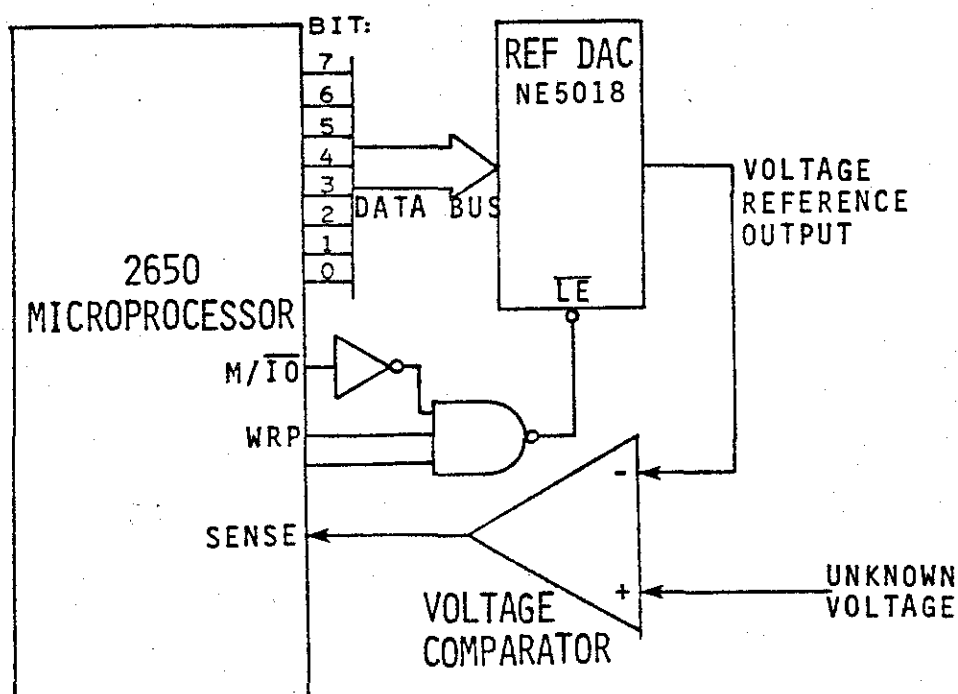
Basically, each input bit to the Reference DAC is turned on in succession, starting with the most significant. The Voltage Comparator compares the resultant DAC output with the unknown voltage. The output of the Voltage Comparator is routed to the microprocessor's SENSE input. This input is high if the unknown voltage exceeds the DAC output; low if the DAC output is more positive than the unknown voltage. Based on a test of the SENSE bit, the microprocessor performs one of the following.

- (1) SENSE LOW: turn off the DAC input bit upon which the test is based, and turn on the next bit in succession.
- (2) SENSE HI: turn on the next bit in succession while leaving the bit upon which the test was based, on.

Each successive bit is turned on, tested, and left on (turned off) in sequence. Only 1 bit is tested at any particular time. Each DAC input bit carries a weight double that of the next most significant bit. Thus, unknown voltage approximation is successively narrowed by a factor of 2. The final result provides a measurement accuracy:  $2^{-N}$ :1, where N equals the number of DAC input bits from the microprocessor.

DIAGRAM 1

ANALOG TO DIGITAL CONVERSION INTERFACE



Routines must be written to:

- (1) Initialize registers, including a Test bit generator, and pattern accumulator.
- (2) Output the accumulated pattern AND the latest test bit to the DAC.
- (3) Test the activity of the SENSE bit.

NOTE: Some DAC and Voltage Comparator amplifiers may require that the microprocessor be placed in a WAIT state to allow for settling time. Use of the PAUSE line is permitted.

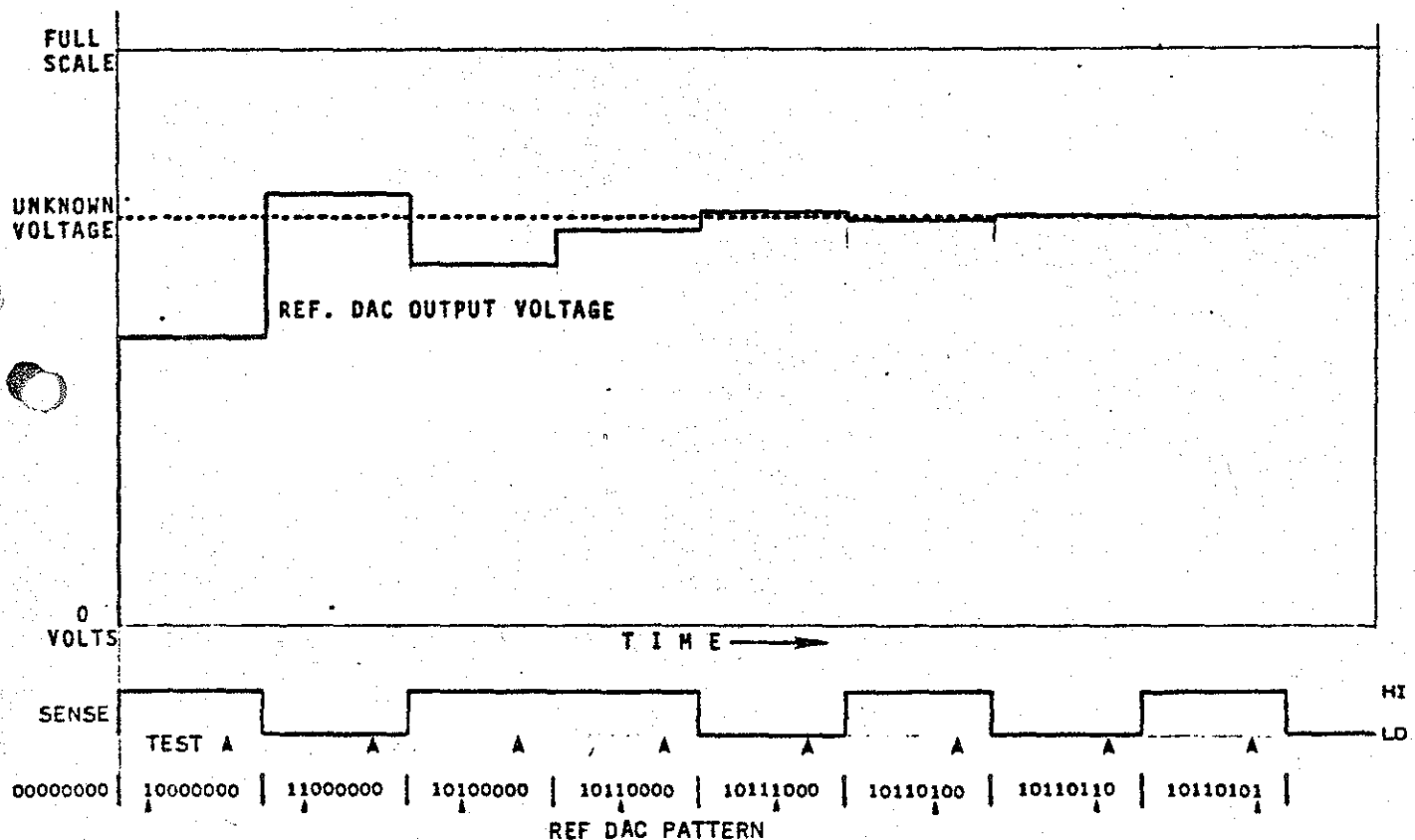
- (4) Add or discard the current bit, based in the activity of SENSE
- (5) Convert the binary-weighted pattern representing the final approximation to its decimal (and fractional) equivalent.

NOTE: Refer to App Note AS-55 "Fixed Point Decimal Arithmetic Routines" in this module.

A typical analog waveform/digital pattern generation scheme for analog to digital conversion is illustrated in Diagram 2.

DIAGRAM 2

SUCCESSIVE APPROXIMATION WAVEFORMS





THIS PAGE IS LEFT INTENTIONALLY BLANK FOR YOUR PERSONAL DESIGN  
NOTES:



## 2650 MICROPROCESSOR COURSE

### MODULE V

#### INTERFACE AND INTERRUPTS

**PREPARED BY:**

MICROPROCESSOR TRAINING DEPARTMENT  
Signetics Corporation  
811 E. Arques Ave.  
Sunnyvale, CA, 94086

**REFERENCE:**

Outlines and Abstracts  
pages 25 to 27.

INTRODUCTION:

Up to this time in the course, we have addressed the 2650 almost exclusively with respect to its SOFTWARE capability - its flexibility to execute a variety of functions by manipulation of its instruction set. In most of the demonstrated programs, visual indication of 2650 processing has been provided via the parallel I/O Port LED's, FLAG, and the 8 digit display. And without explanation, you have exercised a moderate degree of input control to the microprocessor via the I/O port toggle switches and keyboard switch SENS. However, no attempt was made to define precisely how input/output devices such as the display or parallel I/O facility should be configured for use with the microprocessor; i.e., how the controlling INTERFACE between the microprocessor and its controlled input/output devices should be established.

What then is the purpose of the controlling I/O INTERFACE? Simply, its purpose is to ensure orderly transfer of data and synchronization of operations between the microprocessor and external I/O devices. In recognizing that the microprocessor is essentially a logical device, we can presume that its co-operation with externally controlled devices must also be logical.

Of what then does the INTERFACE consist? Essentially the INTERFACE consists of a number of signal lines between the microprocessor and its controlled device. These signal lines may be distinguished as follows:

- a. I/O CONTROL SIGNALS: Those signals which operate in an interlocked or handshaking mode, and which are SYNCHRONIZED.
- b. ADDRESS BUS: A group of parallel lines which together assert address information to SELECT a specific memory location or externally located I/O device - 13 bits + 2 page bits.
- c. DATA BUS: A group of parallel lines whose purpose is to effect transfer of data between the microprocessor and any SELECTED memory or I/O device.

With the mention of the term "signal lines," definition of the INTERFACE in both hardware and software terms is obviated. In terms of HARDWARE, the interface consists of all wires and support logic necessary to effect its proper operation. The terms

INTRODUCTION: (Continued)

"synchronized," "handshaking," and "interlocked" were applied previously to the definition of I/O CONTROL SIGNALS. Use of these terms implies a generous use, and inclusion within the microprocessor, of logic whose function is to provide a timed interlock between the microprocessor and controlled I/O (or memory). It follows that the interface (or external devices via the interface) must contain logic elements capable of generating handshaking signals back to the microprocessor. Signals which are responsive to, and compatible with, microprocessor operation. As this module progresses, take the time to examine each circuit presented with this in mind.

In terms of SOFTWARE, the interface is controlled by execution of instructions causing the microprocessor to condition the I/O control lines for memory or I/O device access and data transfer. Recall that the power of a microprocessor is measured by the flexibility of its instruction set. In terms of I/O, the microprocessor's power is determined by the number of MODES by which the microprocessor may perform operations with external device. The 2650 has several such modes:

1. Single Bit I/O facility      Controlled through execution of appropriate PSW (upper) instructions. FLAG is output. SENSE is input for this mode, commonly used for ASYNCHRONOUS serial I/O data transfer.
2. DATA I/O facility      Controlled through implementation of specific 1-byte instructions (WRTD, REDD) to transmit data to, or access data from, a single port "DATA." Any register may be designated as source or destination.
3. CONTROL I/O facility      Same implementation as DATA I/O facility, except instructions WRTC and REDC are used. Together, the DATA and CONTROL I/O facilities may be used to implement a simple 2-port 8-bit parallel data I/O system. These facilities are defined as NON-EXTENDED I/O. Address differentiation between DATA and CONTROL ports is determined by the condition of a single I/O control line.

NOTE: The "INSTRUCTOR" reserves the use of the WRTC instruction to its own monitor USE program. Thus, when using non extended I/O, only one port (DATA) is available.

INTRODUCTION: (Continued)4. EXTENDED I/O Facility

Controlled through execution of specific 2-byte instructions (WRTE, REDE). This facility permits selection of up to 256 separately addressed I/O ports, hence the designation EXTENDED. Addressing is always specified in the 2nd byte of the instruction; the specified location is determined by the logical condition of the 8 low order Address Bus lines. Source or destination of data is any of the general purpose registers.

NOTE: The INSTRUCTOR provides a fully configured extended I/O facility (Port 7). In addition, Ports at addresses F8-FF are reserved for KEYBOARD and DISPLAY I/O functions implemented by the User System Executive program.

5. MEMORY MAPPED I/O Facility

In this configuration, I/O Ports may be configured at any location outside of physical memory. Control is provided via execution of any data handling (non branch) instruction (absolute or rel. address). The device, rather than memory decodes the address.

NOTE: The "INSTRUCTOR's" memory-mapped I/O location; at location 'OFFF,' was demonstrated in Module IV-G.

With the exception of STRA instructions, memory referenced instructions addressing memory-mapped I/O are considered READ in nature. STRA instructions are executed to WRITE data to memory-mapped I/O devices.

There are several objectives to be satisfied in the study of the Interface between the microprocessor and its controlled I/O devices. These involve:

- a. A thorough understanding of the purpose of each I/O Control Signal and its function in concert with other I/O Signals necessary to control a given I/O or memory access and data transfer.

INTRODUCTION: (Continued)

- b. In-depth comprehension of timing, and synchronization of I/O Control Signals considered from both microprocessor and external device "viewpoints."

NOTE: A description of the microprocessor's AC characteristics, including dynamic wave forms, is included in the discussion of timing and synchronization.

- c. Introduction to a minimum system hardware configuration for each I/O mode, and configuration of the microprocessor to external devices of varying speed, are presented. In these cases, your objective will be one of familiarization. Later, when you develop your own application, your interface will optimize both microprocessor and external I/O design characteristics.

NOTE: While this course is not structured for formal use of test equipment, it will be very worthwhile for you to obtain the use of an oscilloscope for use in examination of 2650 timing and control waveforms. Desirable characteristics include:

2 channel input minimum; 4 channel optimum  
Bandwidth:  $\geq$  5 mhz  
Sweep: DC to 5 Mhz  
Sync: Internal and External Trigger (3rd input)

All 2650 microprocessor inputs and outputs are pinned as well to the S100 BUS recessed within the back panel of the instructor. All pins are defined, by number and function, in the INSTRUCTOR REFERENCE MANUAL.

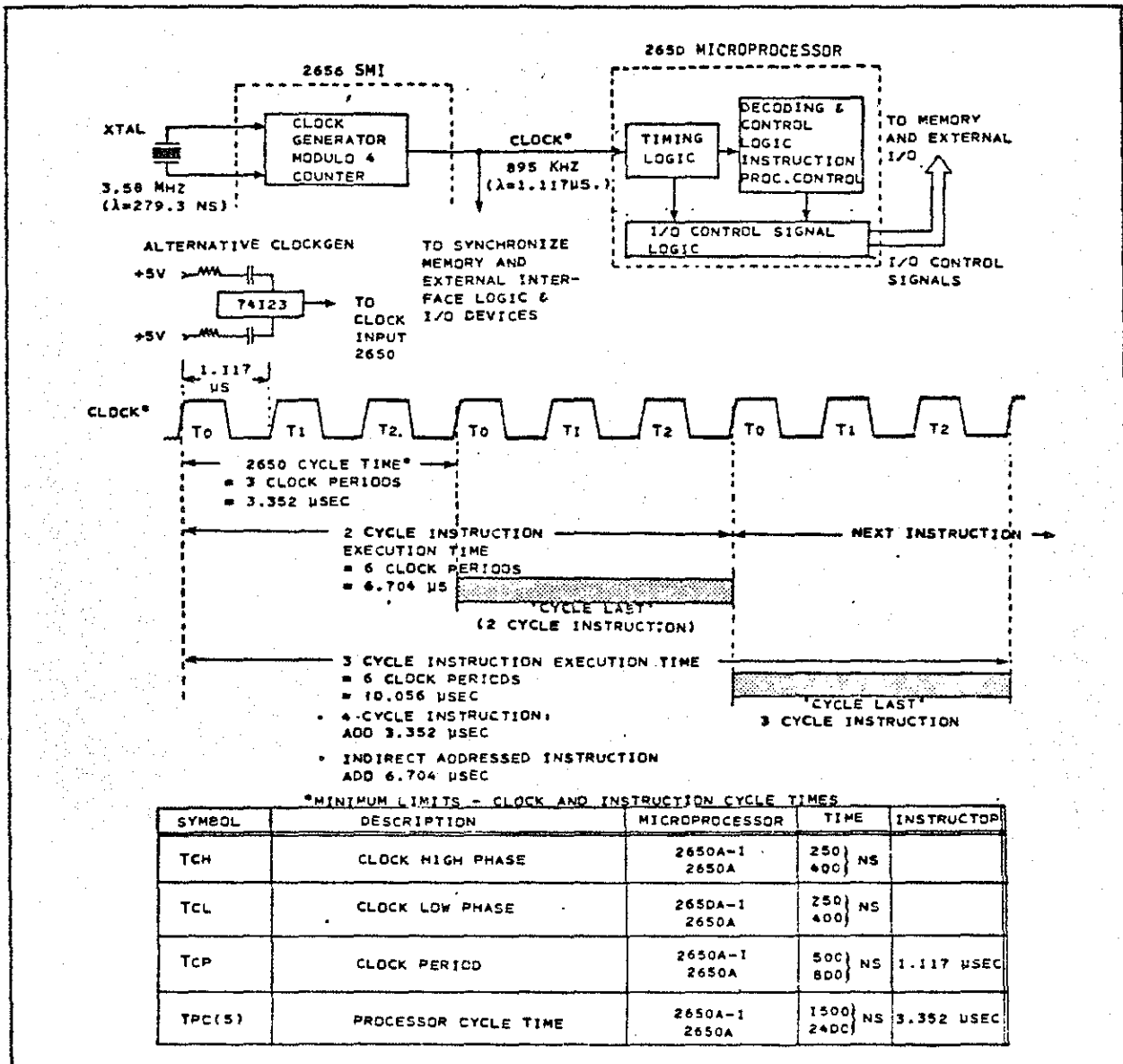
////////////////////////////////////

BASIC MICROPROCESSOR TIMING CONTROLINTRODUCTION:

Numerous timing charts are provided in the 2650 REFERENCE MANUAL to substantiate the synchronization relationship of the various I/O Control Signals. Further, you'll be studying the definition of each signal in its relationship to the others. This section provides a brief introduction to generation of basic microprocessor timing and synchronization. All signals output from the microprocessor, AND the duration of all instructions executed are determined as a function of the microprocessor's CLOCK input.

**DIRECTION:**

Listen to the audio tape (tape 9 side B) for a brief discussion of Diagram 1 (below) which illustrates the basic timing cycle of the 2650 as employed in the INSTRUCTOR.

**DIAGRAM 1****BASIC MICROPROCESSOR TIMING****2650 REFERENCE MANUAL**

- Interface Control Line Definitions PP
- Signal Timing
- Memory Read Timing
- Input Output Timing
- AC Characteristics
- Input/Output Facilities

**DIRECTION:**

Using Diagram 2 (below) as a reference, listen to Tape 9B for an additional commentary on the microprocessor's I/O control signals. Further direction will be provided on tape.

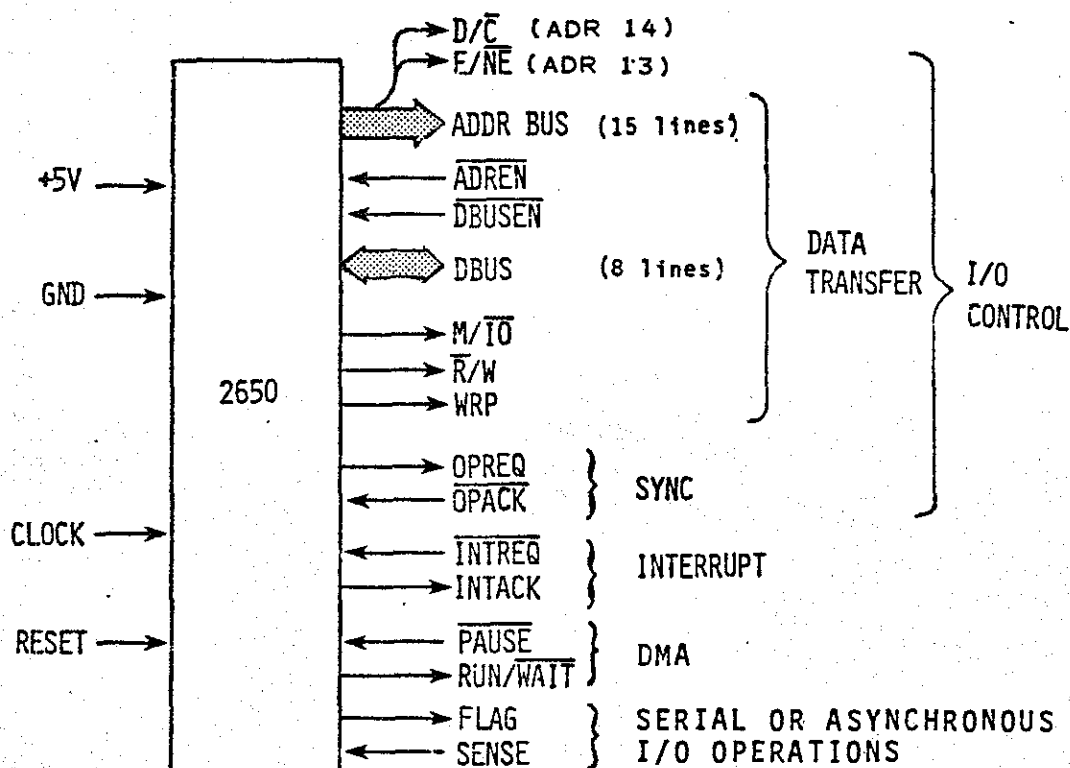
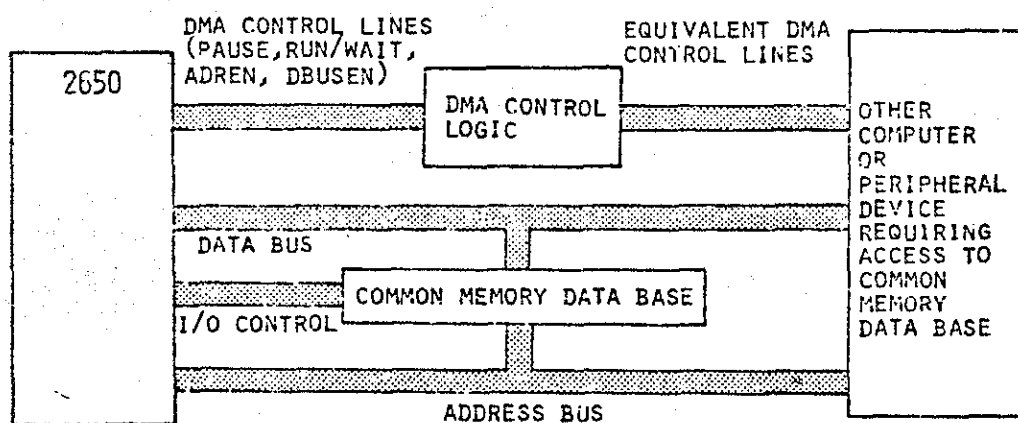
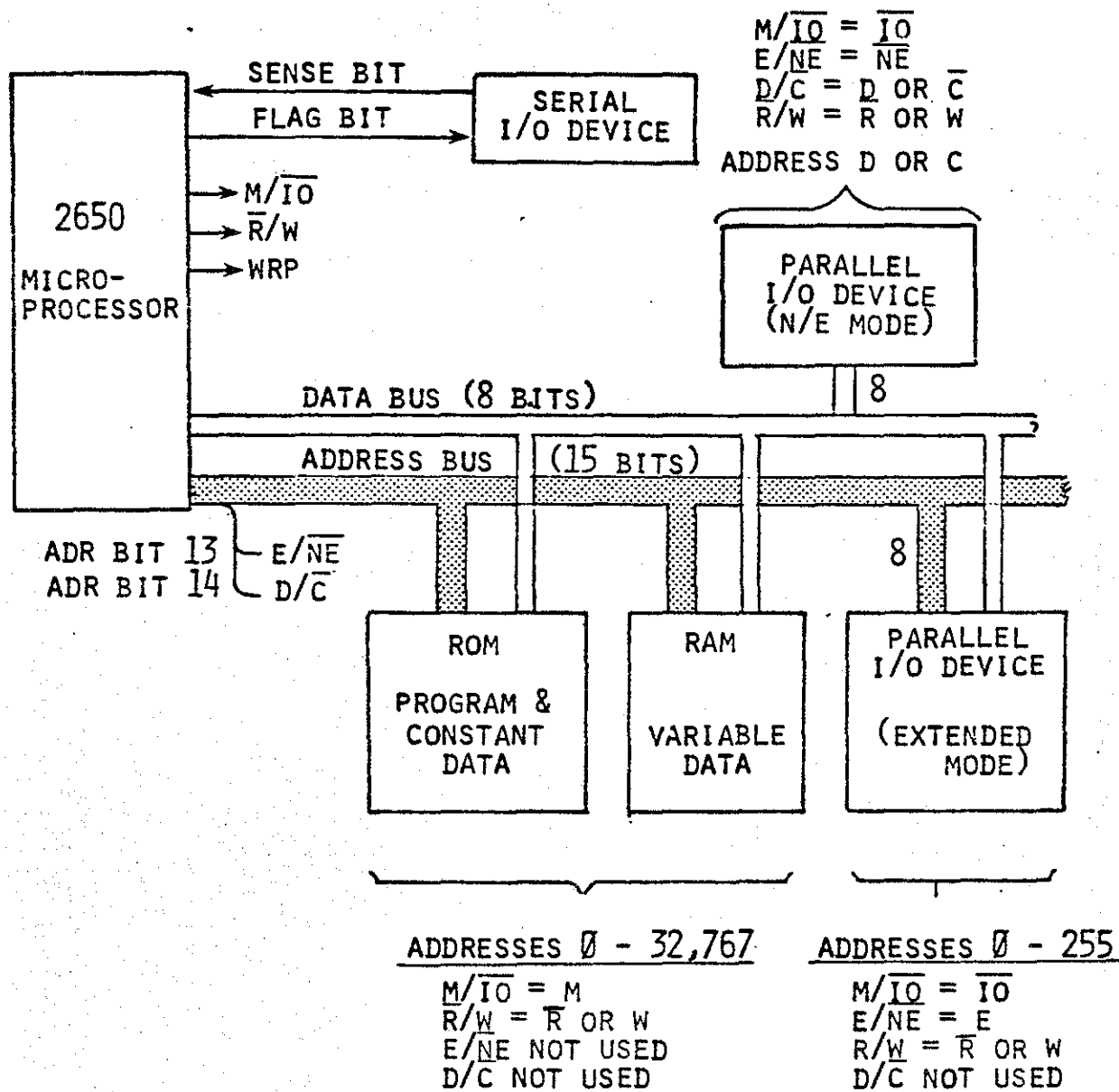
DIAGRAM 22650 INTERFACE SIGNALSDIAGRAM 3DIRECT MEMORY ACCESS (DMA) SYSTEM



DIAGRAM 4

## 2650 INTERFACE CONCEPT



NOTES:

SERIAL I/O CONCEPTSINTRODUCTION:

Implementation of the PSW SENSE and FLAG bits is probably the simplest method of establishing a serial I/O channel between the microprocessor and peripheral device such as terminals, etc. Hardware is minimal. In preparation of the software, the following factors must be considered:

(a) RATE OF DESERIALIZATION OF DATA

suitable delays must be introduced to strobe data into or out of the microprocessor at a rate compatible with the terminal's operation. Transmission rates vary between 110 BPS and 19.2 KBPS.

(b) CHARACTER CODE STRUCTURE

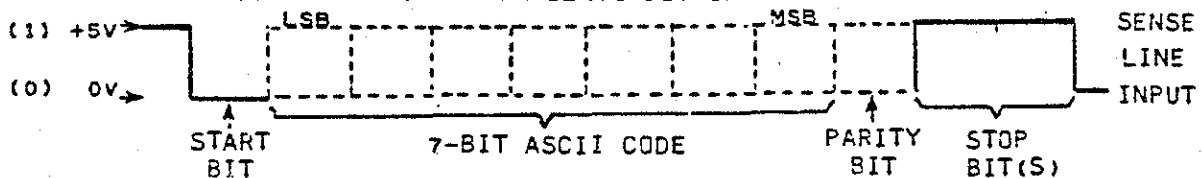
In lower speed terminals, each character is normally framed with one or more start and stop bits. As illustrated in Diagram 5, one such code structure comprises 11 bits, consisting of framing start and stop bits, a 7-bit data code, and a parity bit.

**DIRECTION:**

Take a few minutes to study the detailed specification for a typical serial data transfer (Diagram 5), then go on to the next page.

DIAGRAM 5SERIAL I/O DATA TRANSFER

## A. DATA FROM I/O DEVICE ARRIVES VIA SENSE BIT OF PSU. (10 OR 11 BIT CODE)



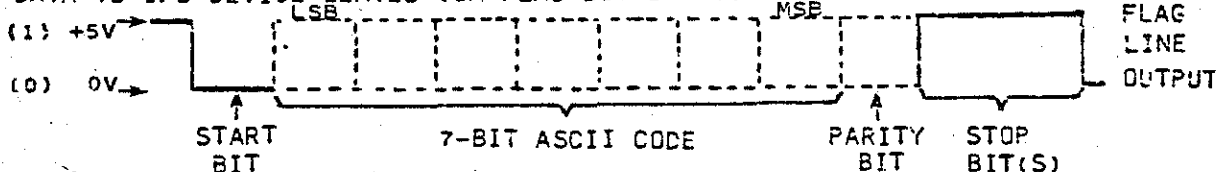
THE SENSE BIT IS NORMALLY A 1 BETWEEN DATA TRANSFERS.

THE LINE DROPS TO ZERO VOLTS (0) TO INDICATE A START BIT.

DATA PLUS PARITY \* IS TRANSFERRED  $\begin{cases} 0V = 0 \\ +5V = 1 \end{cases}$  FOR 8 BIT TIMES.

THE LINE WILL GO BACK TO A 1 (+5V) FOR 1 OR 2 STOP BIT TIMES (DEPENDS ON BAUD RATE).\*

## B. DATA TO I/O DEVICE LEAVES VIA FLAG BIT OF PSU.



TO TRANSMIT A START BIT, SET FLAG = 0

TO TRANSMIT A STOP BIT, SET FLAG = 1

TO TRANSMIT A 1 DATA BIT, SET FLAG = 1

TO TRANSMIT A 0 DATA BIT, SET FLAG = 0

REFERENCE ADDITIONAL NOTES ON NEXT PAGE.

NOTES:

- (1) SPECIFICATION OF ASCII (AMERICAN STANDARD CODE FOR INFORMATION INTERCHANGE) IS PROVIDED IN THE APPENDICES OF BOTH INSTRUCTOR AND 2650 REFERENCE MANUALS.
- (2) PARITY: IS THE SIMPLEST METHOD OF PROVIDING ERROR CHECKING TO DETERMINE IF THE CHARACTER TRANSMITTED IS RECEIVED WITHOUT ERROR BY THE DEVICE TO WHICH IT IS TRANSMITTED. IN ODD PARITY TRANSMISSION SCHEMES, THE PARITY BIT IS RESET (LOGIC 0) IF THE DATA CODE CONTAINS AN ODD NUMBER OF ACTIVE (LOGIC 1) BITS. IN EVEN PARITY SCHEMES, THE PARITY BIT IS 0 IF AN EVEN NUMBER OF DATA CODE BITS ARE LOGIC 1. THE RECEIVING DEVICE COUNTS THE NUMBER OF ACTIVE DATA CODE BITS AND GENERATES ITS OWN PARITY BITS. AFTER THE TRANSMITTING DEVICE SENDS ITS PARITY BIT, THE 2 ARE COMPARED AND AN ERROR IS DETECTED IF TRANSMITTED AND RECEIVED PARITY ARE NOT THE SAME.
- (3) BAUD RATE: THE RATE MEASURED IN BITS PER SECOND AT WHICH DATA IS SERIALIZED OR DESERIALIZED. AT 110 BAUD (STANDARD LOW SPEED TELETYPE) 10 CHARACTERS/SECOND, EACH FORMATTED IN AN 11-BIT CODE, MAY BE TRANSMITTED. AT 300 BAUD OR HIGHER, EACH CHARACTER IS FORMATTED IN A 10 BIT CODE, WITH ONE STOP BIT DROPPED.
- (4) START BIT STOP BIT(S) THESE BITS ARE TRANSMITTED TO FRAME EACH CODE. BY 'KNOWING' IN ADVANCE THE LOGIC LEVEL OF START AND STOP BITS, THE RECEIVING DEVICE CAN DIFFERENTIATE BETWEEN THE TRANSMISSION OF A CHARACTER AND RANDOM NOISE.

THERE ARE NUMEROUS PUBLICATIONS DEDICATED TO THE DESCRIPTION AND DEFINITION OF DATA COMMUNICATIONS CONCEPTS. CONTACT THE SIGNETICS MOS MARKETING IN SUNNYVALE, CALIFORNIA OR YOUR LOCAL SIGNETICS SALES OFFICE FOR FURTHER INFORMATION.

////////////////////////////////////

EXERCISE 1SIMULATION OF SERIAL DATA TRANSMISSIONINTRODUCTION:

In this exercise, you'll use the INSTRUCTOR to simulate a transmission of 9 ASCII character codes located in memory at addresses '1780' through '1788.' Each byte will be transferred to a general purpose register, then displayed on extended I/O Port 7's LEDs.

Suitable delays are inserted to provide a "slow time" visual indication of FLAG ("the output transmission line") as each bit is serialized. Also, the process of byte rotation within the buffering general purpose register will be seen visually via the LED's. The chosen characters spell the word "SERIALIZE." Odd parity generation is specified.

### PROCEDURE:

1. Manually load the following characters into memory locations '1780' through '1788.'

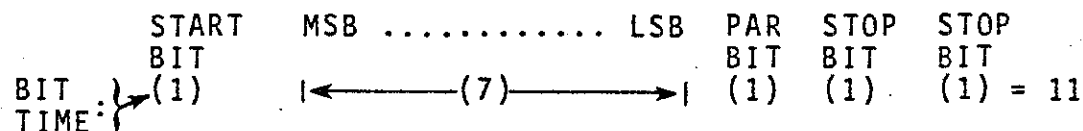
|       |        |        |        |        |        |        |        |        |        |
|-------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|       | S      | E      | R      | I      | A      | L      | I      | Z      | E      |
| Hex:  | '53'   | '45'   | '52'   | '49'   | '41'   | '4C'   | '49'   | '5A'   | '45'   |
| Addr: | '1780' | '1781' | '1782' | '1783' | '1784' | '1785' | '1786' | '1787' | '1788' |

Note that each code excludes the presence of a logic "1" in bit 7 location. The program will take advantage of this fact to use bit 7 as a (low) START bit.

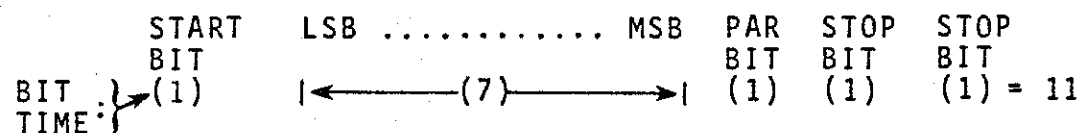
2. Code and load the program "SERIAL" (pages 12-13) into the "INSTRUCTOR'S" memory. As you code the program, take a few minutes to examine its organization and details, particularly the routines "GETCHARA," "XMITA," and "PARGENA."

### Note also the following:

- a. The indirect address table (addr '06' - '13') is written in a symbolic language compatible to existing 2650 assemblers.
- b. Since DELAY 1 is accessed once only, it could just as well have been written in the main program. Access via ZBSR\* execution was not necessary.
- c. As the program is written, the order of transmission for each character is:



To be industry-standard compatible (for low speed devices such as teletype writers), the code should be transmitted as follows:



- You'll be given an opportunity to make the program compatible with the industry standard later in this module.

- d. In routine PARGENA, examine the generation of parity with great care. While there are several approaches to parity generation, this is one of the simplest. Given any character code, each bit is isolated, then tabulated with respect to previously isolated bits. The following truth table illustrates the algorithm.

| IF                |                      | THEN                     |
|-------------------|----------------------|--------------------------|
| ISOLATED<br>BIT = | R2 "FLIP-<br>FLOP" = | RESULT STORED<br>in R2 = |
| 0                 | 0                    | 0 (even)                 |
| 1                 | 0                    | 1 (odd)                  |
| 0                 | 1                    | 1 (odd)                  |
| 1                 | 1                    | 0 (even)                 |

After all bits are checked, the contents of R2 (0 or 1 in bit 0 location) are tested. If R2 = 0, an even number of active bits (0, 2, 4, or 6) were counted; therefore the parity bit must be set active (loc. '8C': PPSU FLAG). If R2 = 1, an ODD number of active bits (1, 3, 5, or 7) were detected in the code; therefore the parity bit must be reset (loc. '91': CPSU FLAG). ODD parity is thus effectively generated.

- e. In routine SERDLYA, a serial bit time (transmission time for each bit) is approximately 1/4 sec. (247.13 msec). You can make this time 249.7 msec. (0.1% accurate) by increasing the constant from 96<sub>10</sub> to 97<sub>10</sub> (loc. '73').
3. Compare your code to the listing supplied on page 14, then execute the program from location '0'. Then perform the steps provided on page 14. Answers to the questions are provided on page 15.

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| +                                        | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| +                                        | 1E | 1F | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C |
| +                                        | 2D | 2E | 2F | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B |
| +                                        | 3C | 3D | 3E | 3F | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A |
| +                                        | 4B | 4C | 4D | 4E | 4F | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| +                                        | 5A | 5B | 5C | 5D | 5E | 5F | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 |
| +                                        | 69 | 6A | 6B | 6C | 6D | 6E | 6F | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| +                                        | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F | 80 | 81 | 82 | 83 | 84 | 85 | 86 |
| +                                        | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F | 90 | 91 | 92 | 93 | 94 | 95 |
| +                                        | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F | A0 | A1 | A2 | A3 | A4 |
| +                                        | A5 | A6 | A7 | A8 | A9 | AA | AB | AC | AD | AE | AF | B0 | B1 | B2 | B3 |
| +                                        | B4 | B5 | B6 | B7 | B8 | B9 | BA | BB | BC | BD | BE | BF | C0 | C1 | C2 |
| +                                        | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF | D0 | D1 |
| +                                        | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | DA | DB | DC | DD | DE | DF | E0 |
| +                                        | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | EA | EB | EC | ED | EE | EF |
| +                                        | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | FA | FB | FC | FD | FE |
| +                                        | FF |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

INDIRECT RELATIVE ADDRESSING: ADD 1100' TO DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE SERIAL START ADDR 0

DESCRIPTION \_\_\_\_\_

ROUTINE SHEET    OF   MEMORY LOCATIONS THIS SHEET   

| ADDRESSES | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |               | COMMENT                          |
|-----------|------|----|----|---------|----------------------|---------------|----------------------------------|
|           | B0   | B1 | B2 |         | OPCODE               | OPERANDS      |                                  |
| 1         | 0000 |    |    | SERIAL  | PPSU                 | FLAG, IX      | On entry, set FLAG and will      |
| 2         |      |    |    |         | CPSU                 | RS, WC, C     | interpret - also RUC and RS then |
| 3         |      |    |    |         | BSTR                 | INIT          | go to INIT                       |
| 4         |      |    |    |         |                      |               |                                  |
| 5         | 0006 |    |    | REGSC   | ACON                 | REGSCA        | Indirect address table           |
| 6         |      |    |    | DELAY I | ACON                 | DELAYIA       | to routines                      |
| 7         |      |    |    | SERDLY  | ACON                 | SERDLYA       |                                  |
| 8         |      |    |    | XMIT    | ACON                 | XMITA         |                                  |
| 9         |      |    |    | GETCHAR | ACON                 | GETCHARA      |                                  |
| 10        |      |    |    | PARGEN  | ACON                 | PARGENA       |                                  |
| 11        |      |    |    | STOPBT  | ACON                 | STOPBITA      |                                  |
| 12        |      |    |    |         |                      |               |                                  |
| 13        | 0016 |    |    | INIT    | ZBSR                 | * REGSC       | On entry, clean non prime        |
| 14        |      |    |    |         | WRITE, RO            | PORT?         | regs, then show no chan.         |
| 15        |      |    |    |         | PPSL                 | RS            | access, clean prime              |
| 16        |      |    |    |         | ZBSR                 | * REGSC       | regs. On return, de select       |
| 17        |      |    |    |         | CPSEL                | RS            | prime regs and go delay          |
| 18        |      |    |    |         | ZBSR                 | * DELAY I     | 0.6 sec. On return initia-       |
| 19        |      |    |    |         | STAR, RO             | INDEX         | 1120 Byte counter index.         |
| 20        |      |    |    | CALL    | LODR, RI             | INDEX         | In this loop, get data in-       |
| 21        |      |    |    |         | COMI, RI             | 9             | dex. 9 Bytes transmitted?        |
| 22        |      |    |    |         | BCTR, EG             | DONE          | Yes! Return to monitor           |
| 23        |      |    |    |         | STAR, RI             | INDEX         | No! Store index and exit         |
| 24        |      |    |    |         | ZBSR                 | * GETCHAR     | to access and serializa c-       |
| 25        |      |    |    |         | ZBSR                 | * PARGEN      | character. On return, gene-      |
| 26        |      |    |    |         | ZBSR                 | * STOPBT      | rate its parity and frame w      |
| 27        |      |    |    |         | BCTR, UN             | CALL          | stant bit, then loop to call     |
| 1         |      |    |    | DONE    | WRITE                | RO            | Another chan. Exit to Monitor.   |
| 2         |      |    |    | INDEX   | RES                  | 1             | Byte Ct. index storage.          |
| 3         |      |    |    |         |                      |               |                                  |
| 4         |      |    |    |         |                      |               |                                  |
| 5         | 0040 |    |    | GETCHAR | LODR, RO             | DATA-1, RI, + | On entry, read up a chan-        |
| 6         |      |    |    |         | RRR                  | RO            | acter and shift right to         |
| 7         |      |    |    |         | STRZ                 | R3            | set up for transmission. R3      |
| 8         |      |    |    |         | STAR, RI             | INDEX         | is Bit Xmit reg. Store byte      |
| 9         |      |    |    |         | ZBSR                 | * XMIT        | index and go transmit the        |
| 10        |      |    |    |         | RETC, UN             |               | byte, bit by bit, then           |
|           |      |    |    |         |                      |               | return to generate parity bit.   |

| ADDRS | DATA |    |          |           |           |                                 |
|-------|------|----|----------|-----------|-----------|---------------------------------|
| R2    | R1   | R2 |          |           |           |                                 |
| 0000  |      |    | YMITA    | LODI, R2  | B         | Quantity, initialize bit etc.   |
|       |      |    |          | WRITE, R3 | PORT7     | and show shifted char in        |
|       |      |    |          | LODI, R0  | H'40'     | lights. Now isolate bit         |
|       |      |    |          | ANDZ      | R3        | for transmission. If a 1,       |
|       |      |    |          | BRNR, R0  | SETFLAG   | go transmit FLAG. If not,       |
|       |      |    |          | CPSU      | FLAG      | if not, show FLAG off           |
|       |      |    | BTDLY    | ZBSR      | * SERDLY  | and delay 1 bit time. On        |
|       |      |    |          | RRL       | R3        | return, shift code left.        |
|       |      |    |          | EDRR, R2  | XMIT+2    | Get transmission count?         |
|       |      |    |          | RETC, UN  |           | not, go transmit another bit    |
|       |      |    | SETFLAG  | PPSU      | FLAG      | yes! Return for another char.   |
|       |      |    |          | BCTR, UN  | BTDLY     | Was a 1 bit transmit FLAG       |
|       |      |    |          |           |           | and loop to bit delay time      |
| 005A  |      |    | REGICA   | EORZ      | R0        | To clear reg, clear R0          |
|       |      |    |          | STAZ      | R1        | then store in R1.               |
|       |      |    |          | STAZ      | R2        | R2,                             |
|       |      |    |          | STAZ      | R3        | and R3, then                    |
|       |      |    |          | RETC, UN  |           | return                          |
| 0070  |      |    | SERDLYA  | PPSL      | R5        | Quantity, go prime reg. off.    |
|       |      |    |          | LODI, R2  | 96        | then initialize bit serial      |
|       |      |    |          | EDRR, R1  | \$        | delay and execute.              |
|       |      |    |          | EDRR, R2  | \$-2      | When finished, select new       |
|       |      |    |          | CPSL      | R5        | prime reg, again and            |
|       |      |    |          | RETC, UN  |           | return                          |
| 007B  |      |    | DELYA    | EDRR, R1  | \$        | This is 0.6 usec delay at       |
|       |      |    |          | EDRR, R2  | \$-2      | start of program. Can-          |
|       |      |    |          | RETC, UN  |           | plate it, then return.          |
| 0080  |      |    | PARCENA  | LODI, R1  | B         | R1 contains the character       |
|       |      |    |          | LODI, R0  | 1         | Quantity, bit # in byte         |
|       |      |    |          | ANDZ      | R3        | and R0 as a bit to mask.        |
|       |      |    |          | EORZ      | R2        | Now isolate a bit, using        |
|       |      |    |          | STAZ      | R2        | R2 as a bit flag to indicate    |
|       |      |    |          | RRR       | R3        | odd or even of active bit       |
|       |      |    |          | EDRR, R1  | PARCENA+2 | check all 8 bits of the char.   |
|       |      |    |          | BRNR, R2  | PARITY 0  | If R2 = 0, not odd or even      |
|       |      |    |          | PPSU      | FLAG      | evenness of active bit was      |
|       |      |    |          | ZBSR      | * SERDLY  | counted so transmission active  |
|       |      |    |          | RETC, UN  |           | parity bit and delay 1 bit time |
|       |      |    | PARITY 0 | CPSU      | FLAG      | then return                     |
|       |      |    |          | BCTR, UN  | \$-5      | REXO say odd number of bits     |
|       |      |    |          |           |           | bits were active so Xmit 0      |
|       |      |    |          |           |           | for parity.                     |
| 009B  |      |    | STOPSTA  | LODI, R0  | 0         | Quantity, show character        |
|       |      |    |          | WRITE, R0 | PORT7     | and push transmission if        |
|       |      |    |          | PPSU      | FLAG      | complete, then transmit         |
|       |      |    |          | ZBSR      | * SERDLY  | stop bits (FLAG+5) 2 bit        |
|       |      |    |          | ZBSR      | * SERDLY  | serial delay to simulate 2      |
|       |      |    |          | RETC, UN  |           | stop bits. When timeout         |
|       |      |    |          |           |           | complete, return.               |

SUGGESTED CODE - PROGRAM SERIAL

4. Modify the bit time delay (location '73') to any values of your choosing. If you disregard the 0.6 sec DELAY1, bit time delay could permit a serialization of more than 33 characters per sec at its fastest rate.

NOTES: \_\_\_\_\_

5. What modifications could you make within routine "SERDLYA" in order to permit serialization of data faster than 33 characters/sec?

6. To increase the number of characters transferred during execution of program "SERIAL," you must change the contents of loc ' ,

7. By changing the byte at the memory location specified in step 6, you could permit a transfer of \_\_\_\_\_ bytes maximum.

8. Fill the DATA TABLE (locations '1780' to '17BF' with ASCII codes. Modify the byte specified in step 6 to permit serial transmission of these 64 bytes of data. NOTE:

9. At location '28,' modify the relative address to cause a branch to location 0. What is the correct code? 18

10. What is the activity of the program after this change is implemented?

| 1  | ADDRS | DATA  |          |    | LABEL   |
|----|-------|-------|----------|----|---------|
|    |       | B0    | B1       | B2 |         |
| 1  | 0000  | 76    | 60       |    | SERIAL  |
| 2  |       | 75    | 19       |    |         |
| 3  |       | 1B    | 10       |    |         |
| 4  |       |       |          |    |         |
| 5  | 0006  | 00    | 6A       |    | REGSC   |
| 6  |       | 8 00  | 7B       |    | DELAY1  |
| 7  |       | A 00  | 70       |    | SERDLY  |
| 8  |       | C 00  | 50       |    | XMIT    |
| 9  |       | E 00  | 40       |    | GETCHAR |
| 10 |       | 1B 00 | 80       |    | PARSEN  |
| 11 |       | 2 00  | 98       |    | STOPBT  |
| 12 |       |       |          |    |         |
| 13 |       | 00 16 | BB 86    |    | INIT    |
| 14 |       | 8 D4  | 07       |    |         |
| 15 |       | A 77  | 10       |    |         |
| 16 |       | C BB  | 86       |    |         |
| 17 |       | E 75  | 10       |    |         |
| 18 |       | 20 BB | 8B       |    |         |
| 19 |       | 2 C8  | 11       |    |         |
| 20 |       | 4 09  | 0F       |    | CALL    |
| 21 |       | 6 E5  | 09       |    |         |
| 22 |       | 8 1B  | 0A       |    |         |
| 23 |       | A C9  | 09       |    |         |
| 24 |       | C BB  | 8E       |    |         |
| 25 |       | E BB  | 90       |    |         |
| 26 |       | 30 BB | 92       |    |         |
| 27 |       | 2 1B  | 70       |    |         |
| 1  |       | 4 B0  |          |    | DONE    |
| 2  |       | 35 XX |          |    | INDEX   |
| 3  |       |       |          |    |         |
| 4  |       |       |          |    |         |
| 5  |       | 00 40 | 0D 37 7F |    | GETCHAR |
| 6  |       | 3 50  |          |    |         |
| 7  |       | 4 C3  |          |    |         |
| 8  |       | 5 C9  | 6E       |    |         |
| 9  |       | 7 BB  | 8C       |    |         |
| 10 |       | 9 17  |          |    |         |

ASCII CODES ARE  
CONTAINED IN THE  
550 REFERENCE  
ANNUAL APPENDICES.

page 14



11. Could DELAY 1 have been used to generate a time delay in "STOPBT" routine instead of executing SERDLY twice? \_\_\_\_\_ (yes)(no)
12. What effect does this change have on data transmission?

////////////////////////////////////

#### SUGGESTED ANSWERS TO QUESTIONS - PAGE 14

- (5) DELETE THE INSTRUCTION 'F9' '7E' AT LOCATION '74,' SUBSTITUTE NOPS CONTROL THE TIME OUT (NOW  $\leq$  2.5 MSEC BIT TIME) BY MODIFYING THE CONSTANT IN LOC '73' ... TRY IT.
- (6) LOCATION '27.'
- (7) 255 BYTES .. 256 IMPOSSIBLE IN THIS SCHEME.
- (9) '18' '56.'
- (10) LOOP FOREVER - TRANSMITS X BYTES OVER AND OVER AGAIN.
- (11) SURE .. THE INDUSTRIAL SPEC REQUIRES A MINIMUM OF 1 OR 2 BIT TIME BEFORE THE NEXT CHARACTER IS TRANSMITTED. THERE IS NO MAXIMUM. CONSIDER YOURSELF TYPING AT A TERMINAL.
- (12) OBVIOUS - IT SLOWS IT DOWN.

////////////////////////////////////

**DIRECTION:** Complete the next 4 steps. Answers to the questions are provided on the next page.

1. What single instruction would you change to permit bit serial delay time control from the parallel I/O input toggle switches?  
loc: ' \_ \_ \_ ' symbolic instruction: \_\_\_\_\_ code: ' \_ \_ ' \_ \_
2. Could program "SERIAL" be modified so that all of its instructions (locations '0000' - '00A2') may themselves be serialized to the outside world via FLAG? \_\_\_\_\_ (yes)(no) If "NO," why not?  
If "YES," indicate the modifications:  
\_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_
3. What modification would you make to program "SERIAL" if the initial timeout of 0.6 seconds (DELAY1) were not required for your application?  
\_\_\_\_\_

4. At location '28,' would a change of the code from '18''0A' to '18''6C' cause any functional modification to the program's operation? \_\_\_\_\_ (yes)(no) if "yes," explain: \_\_\_\_\_

**DIRECTION:** Compare your answers with those provided below, then proceed as directed.

SUGGESTED ANSWERS TO QUESTIONS - pages 15-16

1. LOCATION '72,' REDE, R2 PORT7; '56''07.'

NOTE: THIS METHOD IS PARTICULARLY USEFUL FOR FAST 'FINE TUNING' OF TIME-DEPENDENT FUNCTIONS. AFTER SETTING THE SWITCHES TO THE DESIRED VALUE, SUBSTITUTE THE REDE INSTRUCTION WITH THE APPROPRIATE IMMEDIATE ADDRESSED LOAD INSTRUCTION. THE SWITCH PATTERN PROVIDES THE CORRECT DATA FOR THE 2ND BYTE. NOTE THAT ONE INSTRUCTION CYCLE IS LOST IN SUBSTITUTION OF A LODI FOR AN REDE. IF A REDC OR REDD INSTRUCTION IS USED (NON EXTENDED I/O), THERE IS NO INSTRUCTION CYCLE LOST.

2. YES: AT LOCATION '27' CHANGE THE COMPARE VALUE (BYTE 1) FROM '09' TO 'A3' (LAST ADDRESS IS '00A2').

AT LOCATION '40,' THE INSTRUCTION CODE SHOULD BE MODIFIED TO '0D'3E'FF.' THUS 1ST BYTE OF DATA TO BE SERIALIZED IS LOCATED IN ADDRESS 0.

3. AT LOCATION '0020,' DELETE THE ZBSR INSTRUCTIONS. SUBSTITUTE NOPS (2).

4. NO: NO FUNCTIONAL CHANGE. INSTRUCTIONS AT LOCATIONS '0000 - 0003' ARE INITIALIZING THE PSW ONLY.

**DIRECTION:**

////////////////////////////////////

Listen to tape 9B for a brief discussion of deserializing techniques, using the SENSE line as an input to the microprocessor. Refer to page 8 of this module; Diagram 5, Part A.

NOTES: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

**DIRECTION:**

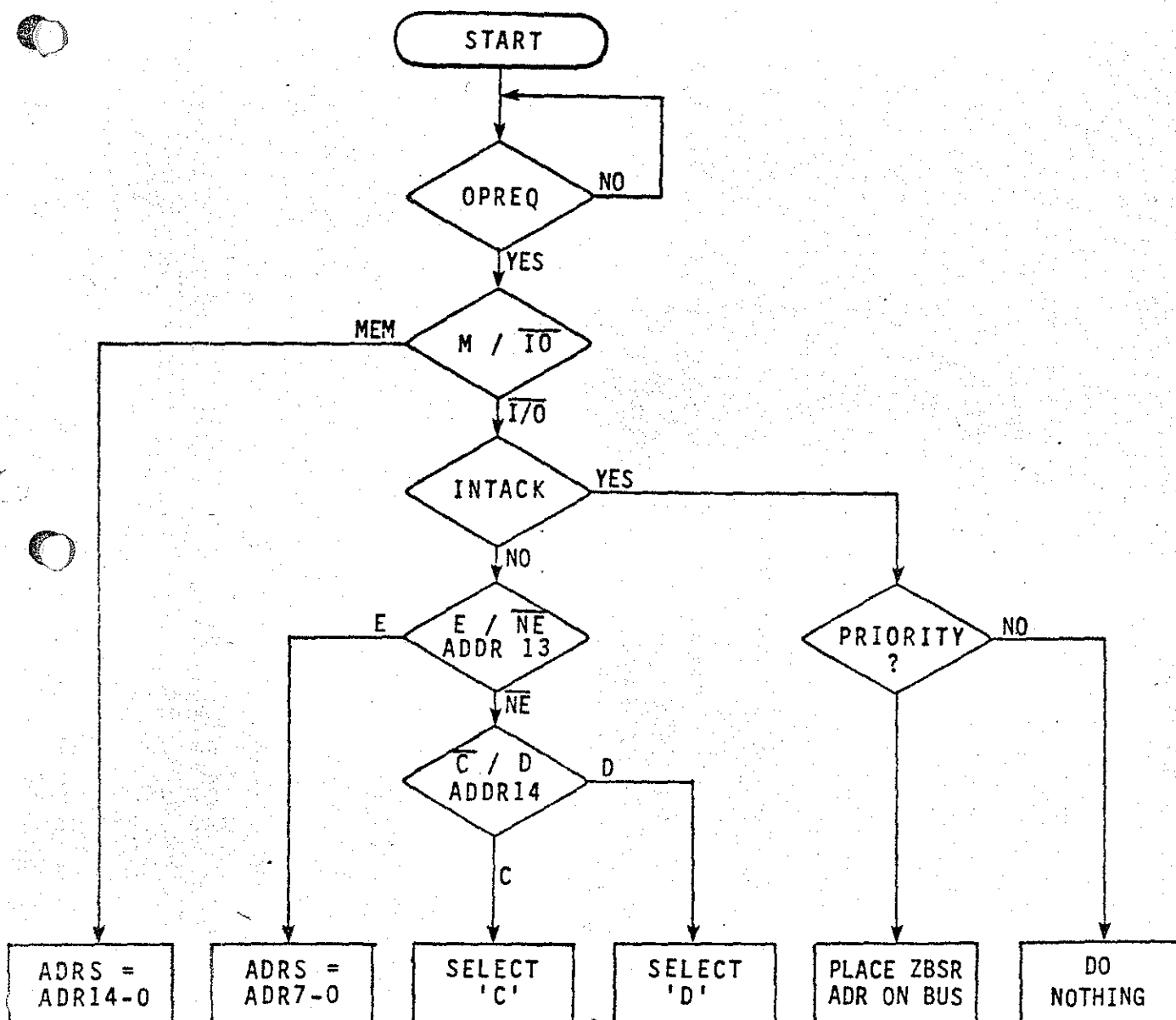
When directed to do so on tape 9B, go on to the next page in this module.

DEVICE SELECT DECODINGINTRODUCTION:

On pages 1 through 7 of this module, you were introduced to the various parallel I/O control modes which the microprocessor is capable of handling. These included non extended, extended and memory mapped I/O modes. Diagram 6 flowcharts the decisions to be made by external device enable logic in order to recognize an output address as selecting that device, to the exclusion of all others.

DIRECTION:

Listen to tape 9B for a brief commentary on the Device Select Decoding Decision Flowchart.

DIAGRAM 6DEVICE SELECTION DECODING DECISION FLOWCHART

2650 MICROPROCESSOR TIMINGINTRODUCTION:

In order to design effective interfaces between the microprocessor and its memory and controlled I/O, one must have a solid understanding not only of the definition of the control lines used, but their TIMING with respect to each other. The following few pages and accompanying taped discussion complement the timing description and specification found in the 2650A Data Sheets and Reference Manual.

For the purpose of this discussion, we'll use the timing specification for the 2650A, assuming that the clock input is 1.25 Mhz, yielding a duration of 800 nsec, and 50% duty cycle, e.g., the clock's high and low phases are equal at 400 nsec.

There are several timing diagrams, due to the multiple of interfaces (memory and various I/O modes) associated with the 2650. When you design a given interface, these diagrams (next 3 pages) provide a handy reference to required timing considerations. You'll be examining each timing diagram as directed on tape. Discussion of Interrupt Timing is reserved to the discussion of that subject.

The AC ELECTRICAL CHARACTERISTICS (switching times) of the 2650A and 2650A-1 are provided below. As the discussion proceeds, refer to them as necessary.

**DIRECTION:** Turn to the next page and listen to tape 10A.

**AC ELECTRICAL CHARACTERISTICS**  $T_A = 0^\circ\text{C to } +70^\circ\text{C}$ ,  $V_{CC} = +5V \pm 5\%$ .

| PARAMETER                                    | 2650A                         |     |                              | 2650A-1                       |     |                              | UNIT |
|----------------------------------------------|-------------------------------|-----|------------------------------|-------------------------------|-----|------------------------------|------|
|                                              | Min                           | Typ | Max                          | Min                           | Typ | Max                          |      |
| $T_{CH}$ Clock high phase                    | 400                           |     |                              | 250                           |     |                              | ns   |
| $T_{CL}$ Clock low phase                     | 400                           |     |                              | 250                           |     |                              | ns   |
| $T_{CP}$ Clock period                        | 800                           |     |                              | 500                           |     |                              | ns   |
| $T_{PC}$ Processor cycle time <sup>5,7</sup> | 2400                          |     |                              | 1500                          |     |                              | ns   |
| $T_{OPR}$ OPREQ pulse width <sup>7</sup>     | $2T_{CH} +$<br>$T_{CL} - 100$ |     | $2T_{CH} +$<br>$T_{CL} + 50$ | $2T_{CH} +$<br>$T_{CL} - 100$ |     | $2T_{CH} +$<br>$T_{CL} + 50$ | ns   |
| $T_{COR}$ Clock to OPREQ time                | 100                           |     | 300                          | 100                           |     | 200                          | ns   |
| $T_{AS}$ Address stable                      | 50                            |     |                              | 50                            |     |                              | ns   |
| $T_{AD}$ Address delay                       | 50                            | 50  |                              |                               |     |                              | ns   |
| $T_{CS}$ Control signal stable               | 50                            |     |                              | 50                            |     |                              | ns   |
| $T_{DIS}$ Data in setup                      | 0                             |     |                              | 0                             |     |                              | ns   |
| $T_{DIH}$ Data in hold                       | 10                            |     |                              | 10                            |     |                              | ns   |
| $T_{OD}$ Data out delay                      | 50                            |     |                              | 50                            |     |                              | ns   |
| $T_{OS}$ Data out stable                     | 50                            |     |                              | 50                            |     |                              | ns   |
| $T_{OAS}$ OPACK setup time                   | 100                           |     |                              | 100                           |     |                              | ns   |
| $T_{OAH}$ OPACK hold time                    | 150                           |     |                              | 150                           |     |                              | ns   |
| $T_{WPD}$ Write pulse delay                  | 100                           |     | 450                          | 100                           |     | 300                          | ns   |
| $T_{WPV}$ Write pulse width <sup>7</sup>     | $T_{CL} - 100$                |     | $T_{CL}$                     | $T_{CL} - 100$                |     | $T_{CL}$                     | ns   |
| $T_{IRS}$ INTREQ setup time                  |                               |     | 150                          |                               |     | 150                          |      |
| $T_{IRH}$ INTREQ hold time                   | 0                             |     |                              | 0                             |     |                              |      |
| $T_{ABD}$ Address bus tri-state delay        |                               |     | 180                          |                               |     | 180                          | ns   |
| $T_{OBD}$ Data bus tri-state delay           |                               |     | 150                          |                               |     | 150                          | ns   |

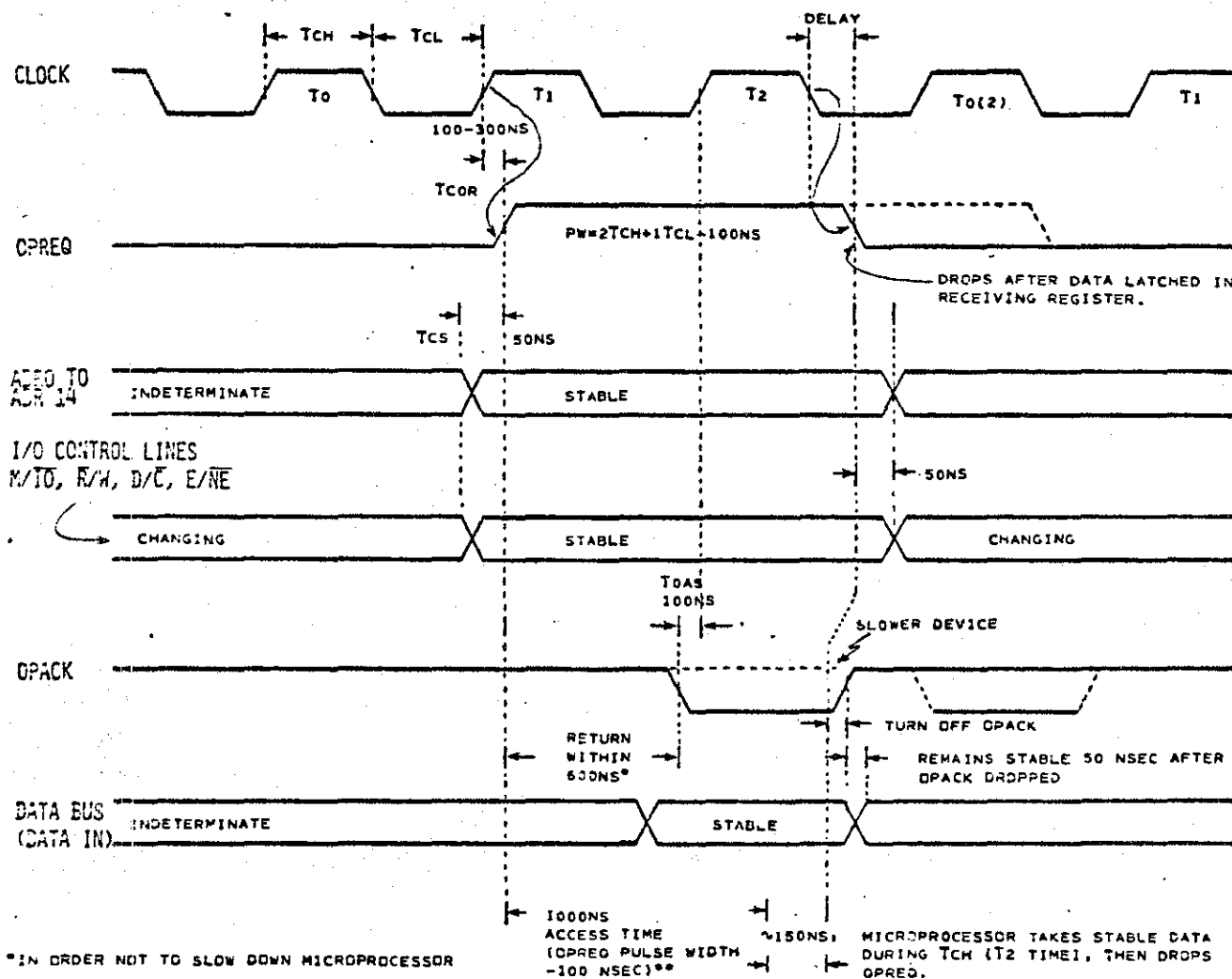
## NOTES

1. Input levels swing between 0.80 and 2.2 volts.
2. Input signal transition times are 20ns.
3. Timing reference level is 1.5 volts.

5. Processor cycles time consists of three clock periods.
6. Output buffer rise time is 150ns maximum.
7. These values assume that OPACK is returned in time to not cause the processor to

DIAGRAM 7

## MEMORY OR I/O READ CYCLE TIMING



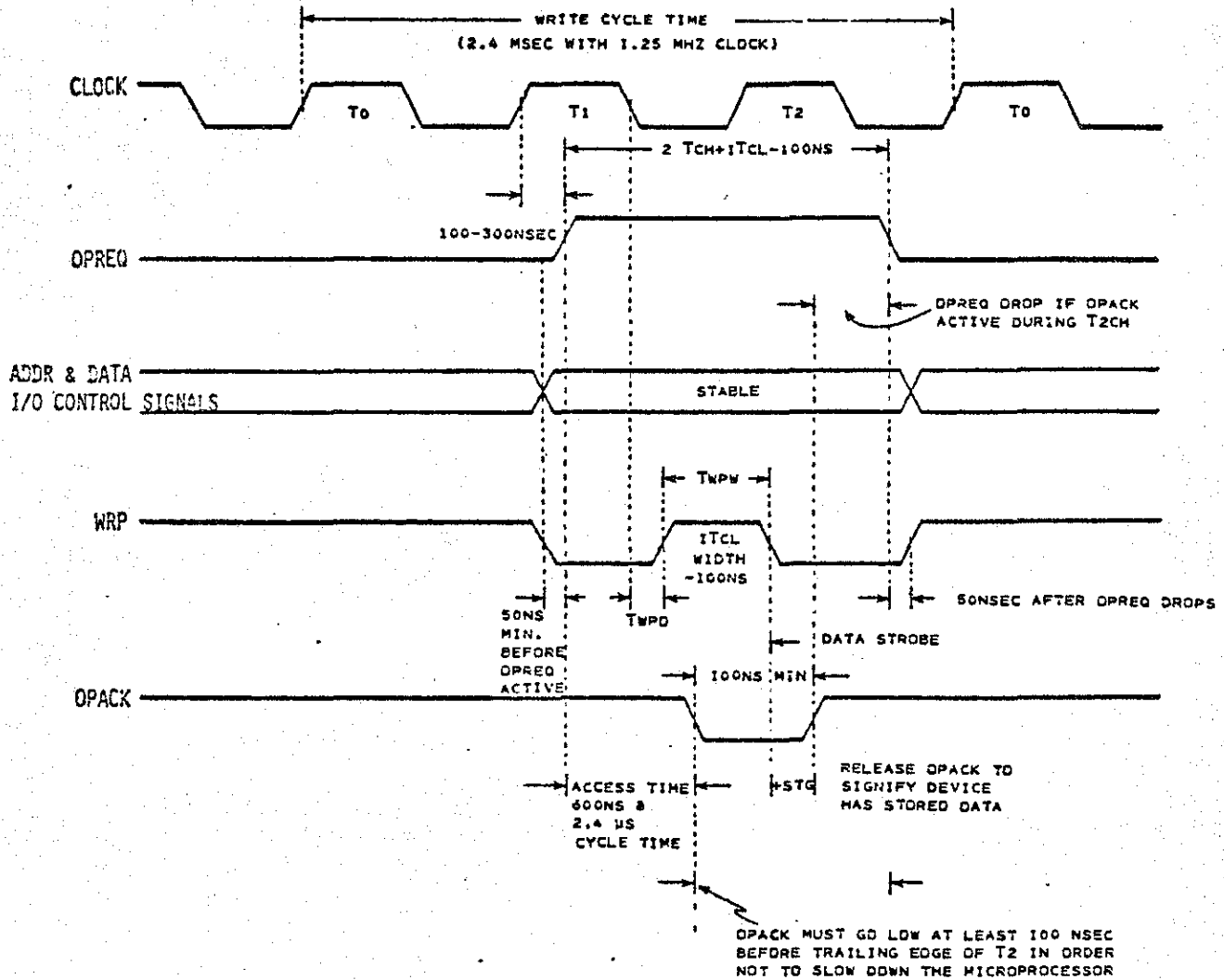
\*IN ORDER NOT TO SLOW DOWN MICROPROCESSOR

\*\*ACCESS TIME  $\leq 2 T_{CH} + T_{CL} - 200 NS$ . WITH SYNC 1 MHZ CLOCK,  $T_{ACC} = 1.3 \mu SEC$

## NOTES:

DIAGRAM 8

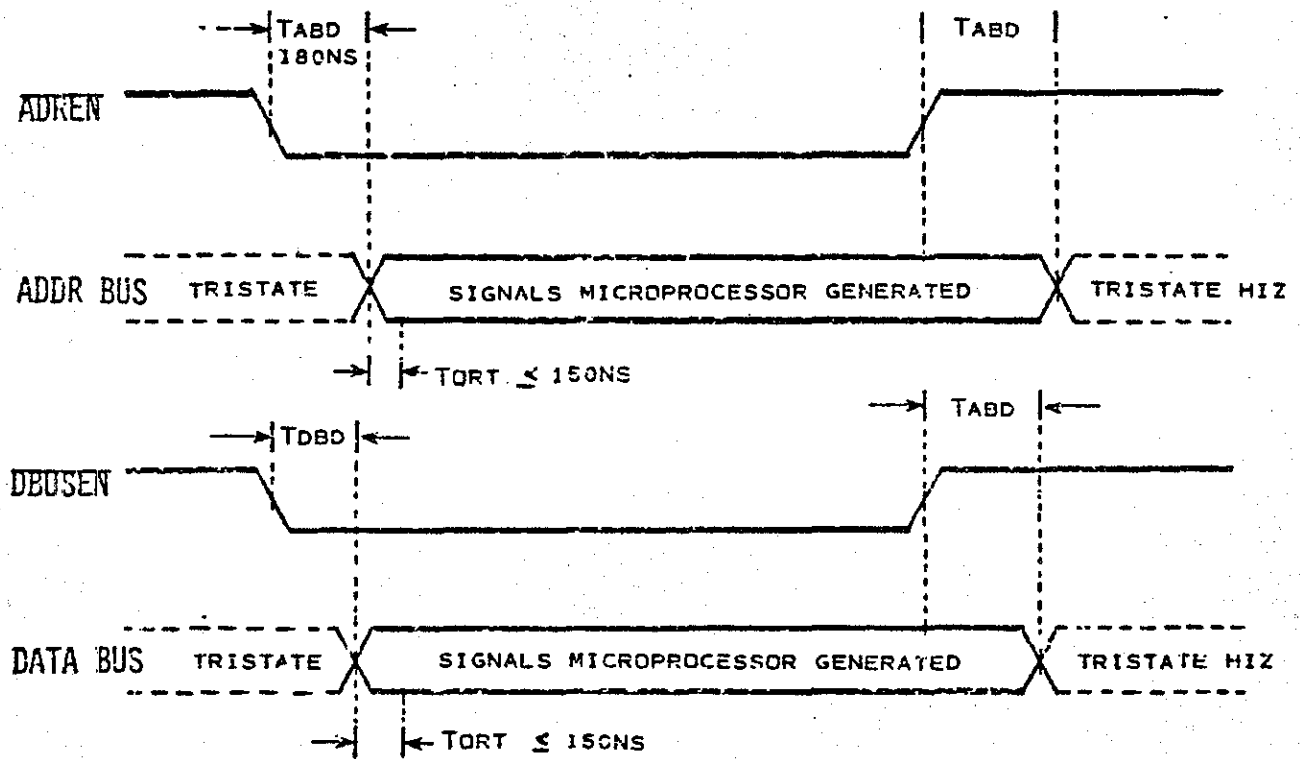
## MEMORY OR I/O WRITE CYCLE TIMING



## NOTES:

DIAGRAM 9

## ADDRESS AND DATA BUS ENABLES

NOTES:**DIRECTION:**

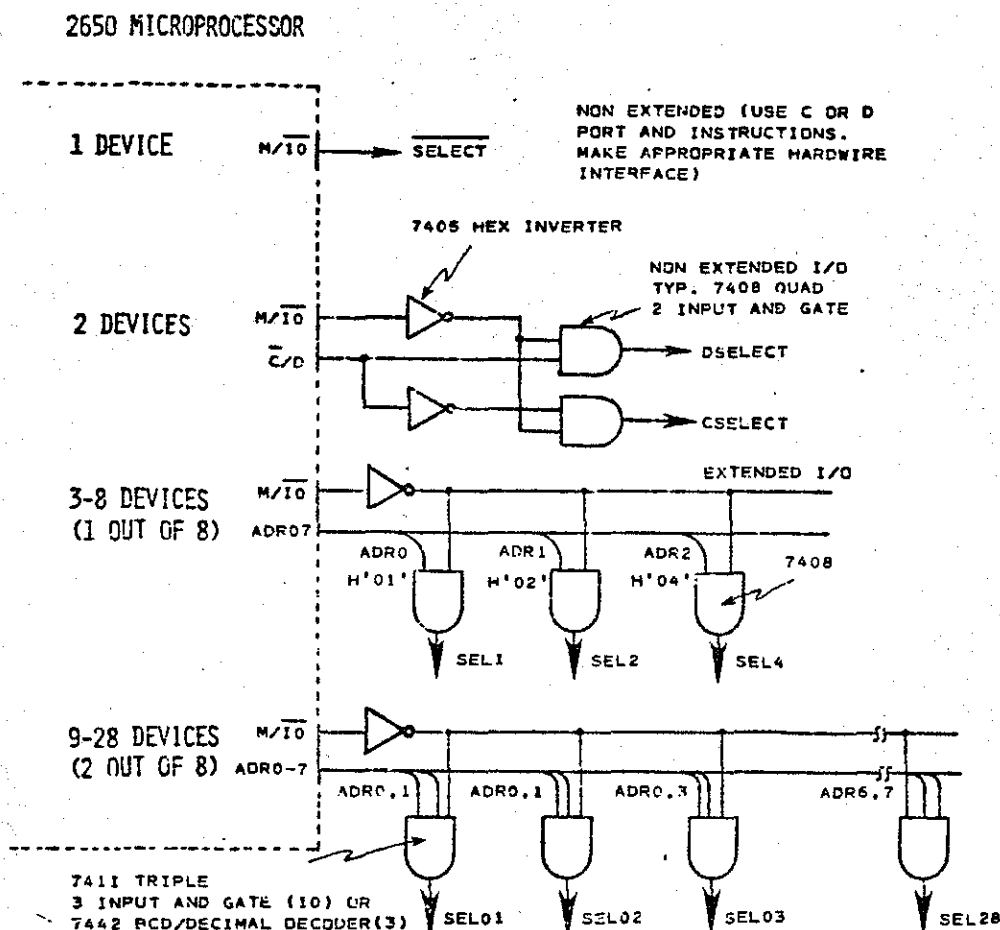
Go right on to the next page when directed to do so on tape.

DEVICE ADDRESS AND SYNCHRONIZATION MINIMUM REQUIREMENTSINTRODUCTION:

Before describing minimum hardware interface for microprocessor operation with memory and various I/O modes, a general discussion of minimum address decoding and device response synchronization techniques is provided.

MINIMUM ADDRESS DECODE STRUCTURES

Diagram 10 illustrates 4 methods for interfacing an address to external I/O devices, dependent only on the number of controlled devices. Use structures as simple as possible, consistent with the requirements of your application. You'll find production of your application more economical and design/debug problems reduced in complexity. The minimum address structures illustrated are offered without additional comment.

DIAGRAM 10MINIMUM ADDRESS DECODE STRUCTURES

DIRECTION:

Go right on to the next page.

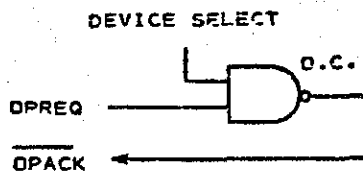


DEVICE RESPONSE SYNCHRONIZATION TECHNIQUES

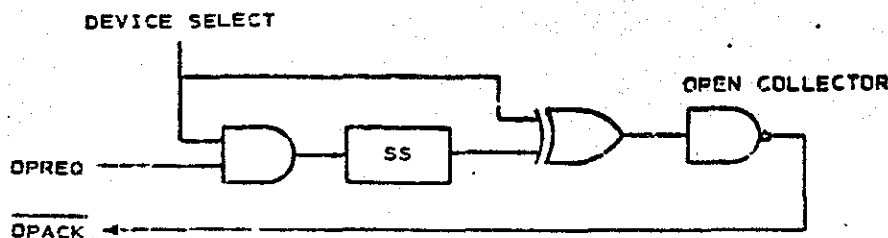
In designing the interface between the microprocessor and external I/O devices, you must consider the response time of the devices; i.e., their generation of OPACK upon reception of OPREQ. Diagram 11 illustrates 3 different approaches to the generation of an acceptable OPACK. In subsequent diagrams which provide details relative to I/O mode interfacing (e.g., extended, non extended or memory mapped), OPACK may or may not be illustrated. Typically, if all interfaced devices can deliver a valid response to the microprocessor WITHOUT slowing down its operation, OPACK may be hard-wired to your system's ground.

**DIRECTION:**

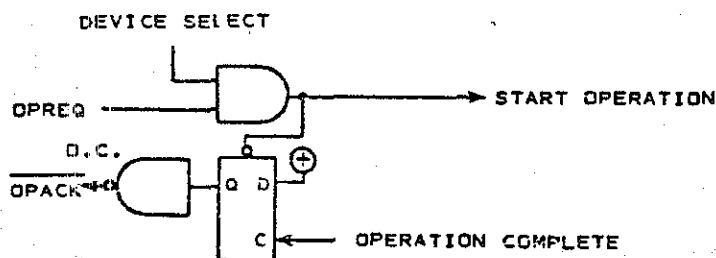
Listen to tape 10A for a brief commentary regarding synchronization of device response within a given system, referenced to Diagram 11.

DIAGRAM 11DEVICE RESPONSE SYNCHRONIZATION TECHNIQUES

CASE 1.  
FAST DEVICE



CASE 2.  
SLOW DEVICE WITH  
FIXED RESPONSE  
TIME.



CASE 3.  
SLOW DEVICE WITH  
VARIABLE RESPONSE  
TIME.

## MEMORY AND I/O MODULAR INTERFACE SCHEMES

INTRODUCTION:

Diagrams 12 through 20 illustrate approaches to design of the interface between the microprocessor and its memory or controlled I/O operating in any of the following modes:

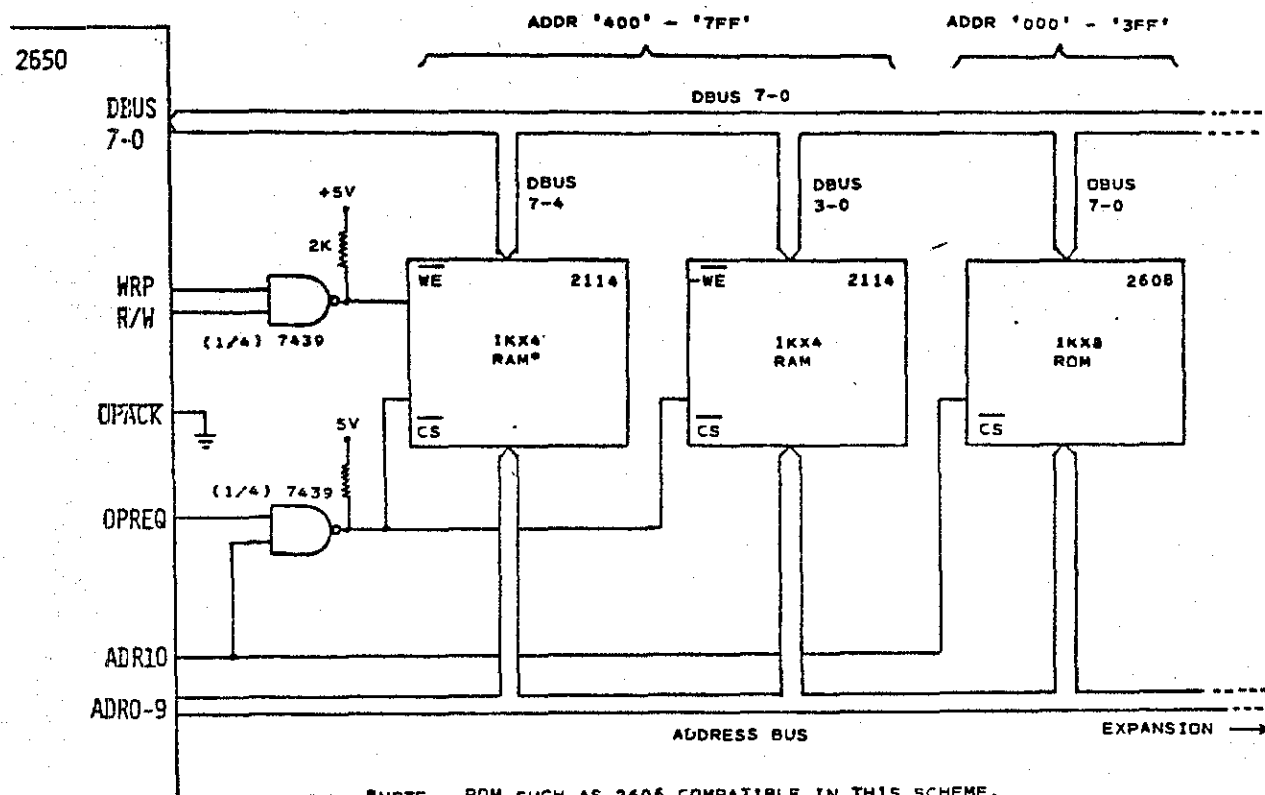
- EXTENDED
- NON EXTENDED
- MEMORY MAPPED
- SERIAL COMMUNICATIONS ((De)serialization by UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER)

**DIRECTION:**

Listen to tape 10A for commentary on each of the following diagrams.

DIAGRAM 12

## SIMPLE MEMORY INTERFACE

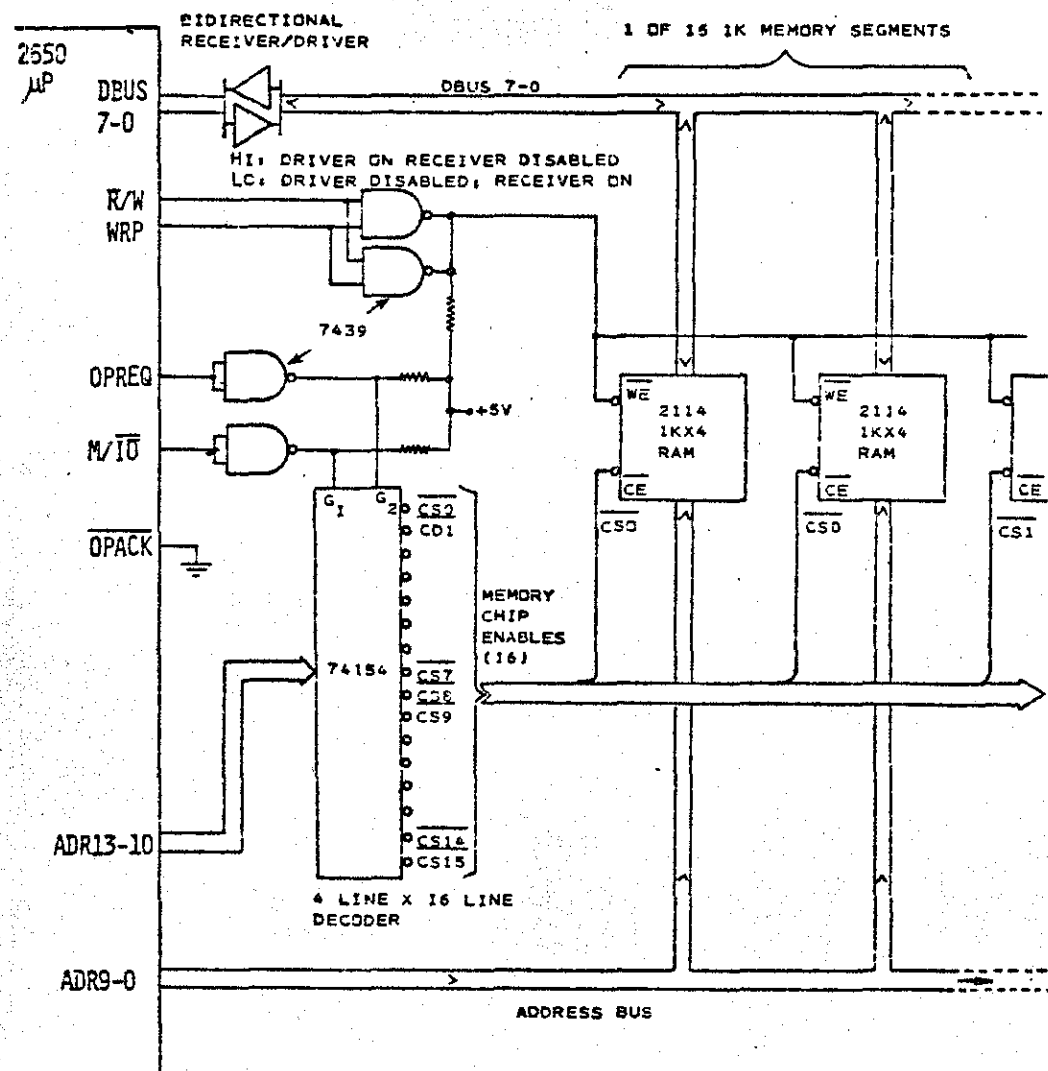


\*NOTE: ROM SUCH AS 2606 COMPATIBLE IN THIS SCHEME.  
DROP WE.

NOTES:

DIAGRAM 13

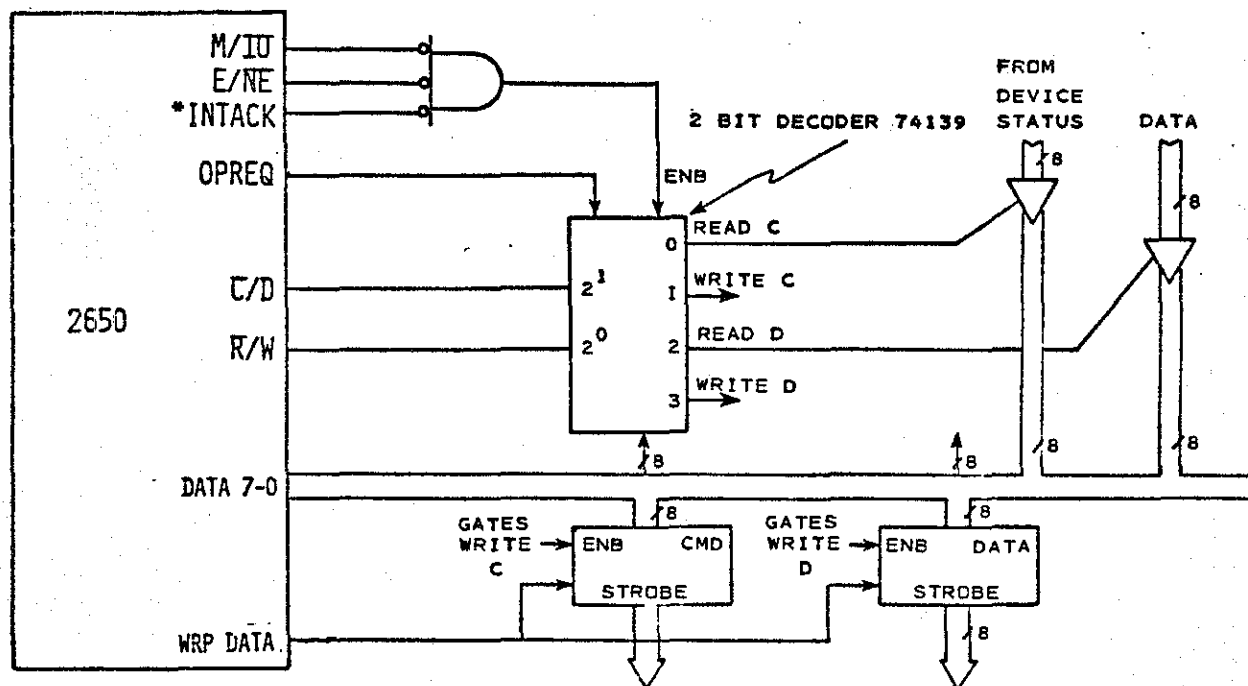
## MEMORY INTERFACE EXPANSION



NOTES:

DIAGRAM 14

## NON EXTENDED I/O INTERFACE



**\*NOTES:** INTACK: In systems implementing INTERRUPT processing or for INTERRUPT configured REAL-TIME CLOCKS, etc., INTACK must be gated (in inactive state) to generate I/O enables.

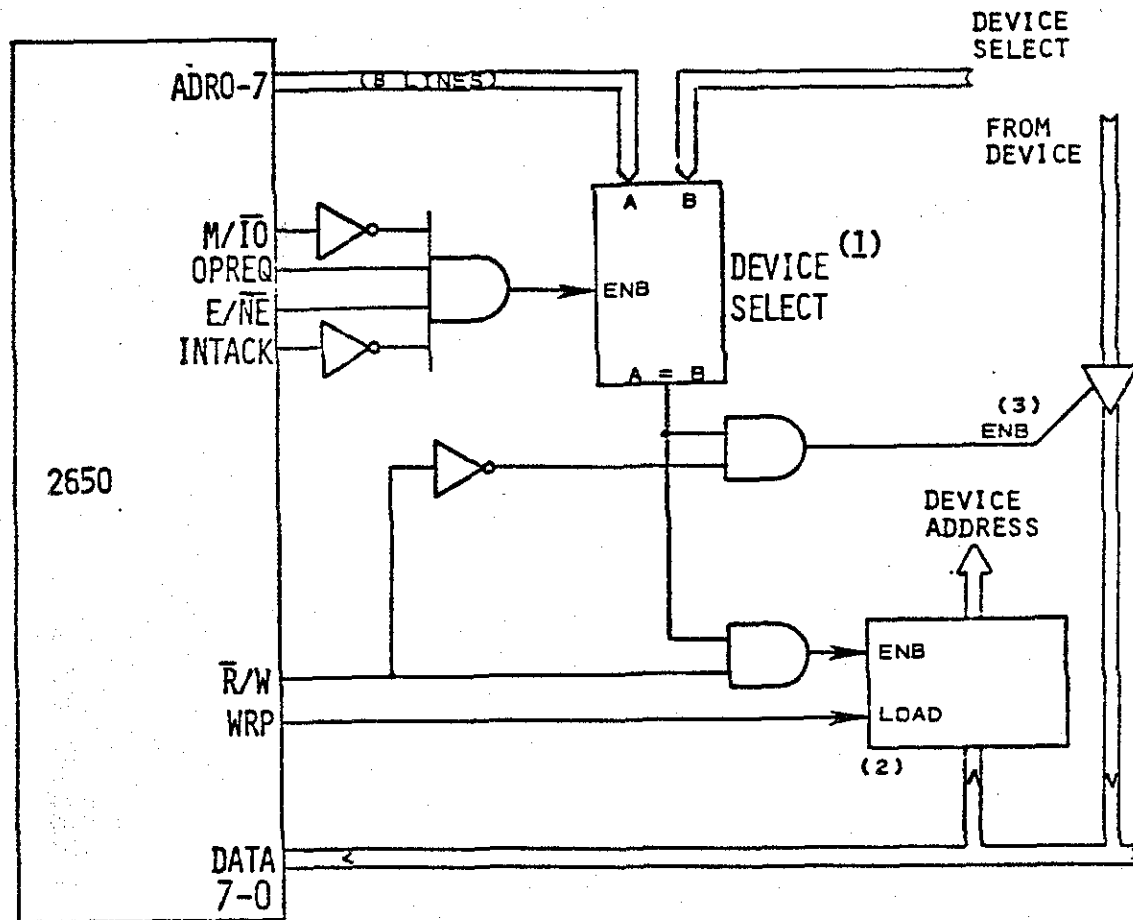
REASON: 2650 generates READ I/O line conditioning during its interrupt recognition sequence in order to access the interrupting devices's vector. Full explanation in section on interrupts.

CMD OUT (WRTC)  
STATUS IN (REDC) } could be any defined 8-bit field.

ADDITIONAL NOTES:

DIAGRAM 15

## EXTENDED I/O INTERFACE SCHEME



NOTES: INTACK: Same as notes for Diagram 15.

(1) DEVICE SELECT LOGIC: Could use 74154 4x16 DECODER/DE-MULTIPLEXER to provide UNIQUE DEVICE SEL. LINES TO EACH OF 16 "devices."

Alternative: Configure each I/O port with bipolar 8T32 family field-programmable 8-bit addressable bi-directional I/O port. Layout simplified; decreased number of chips required.

(2) OUTPUT DRIVERS

7477 QUAD BISTABLE LATCH  
74174 QUAD BISTABLE LATCH

(3) INPUT DRIVERS

8T95/96/97/98 Hi-speed Hex Tristate buffers.

DIAGRAM 16

## DIRECT MEMORY ACCESS (DMA) INTERFACE

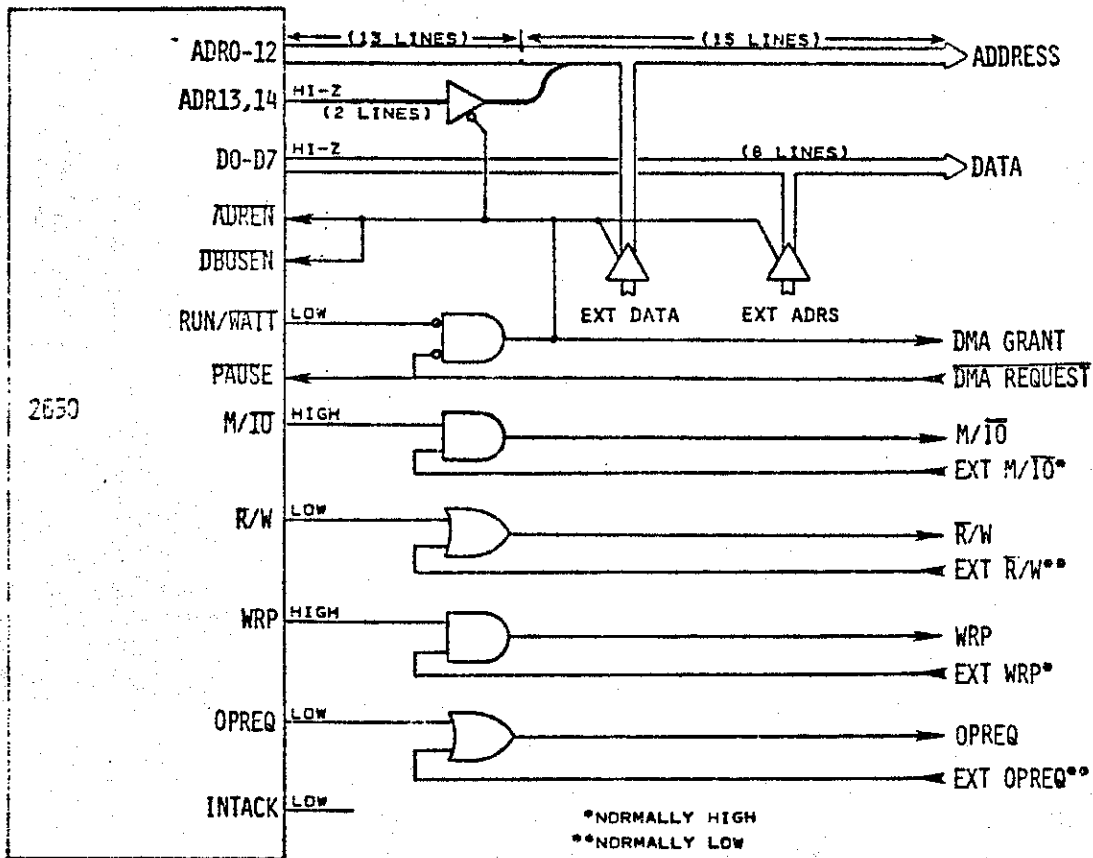
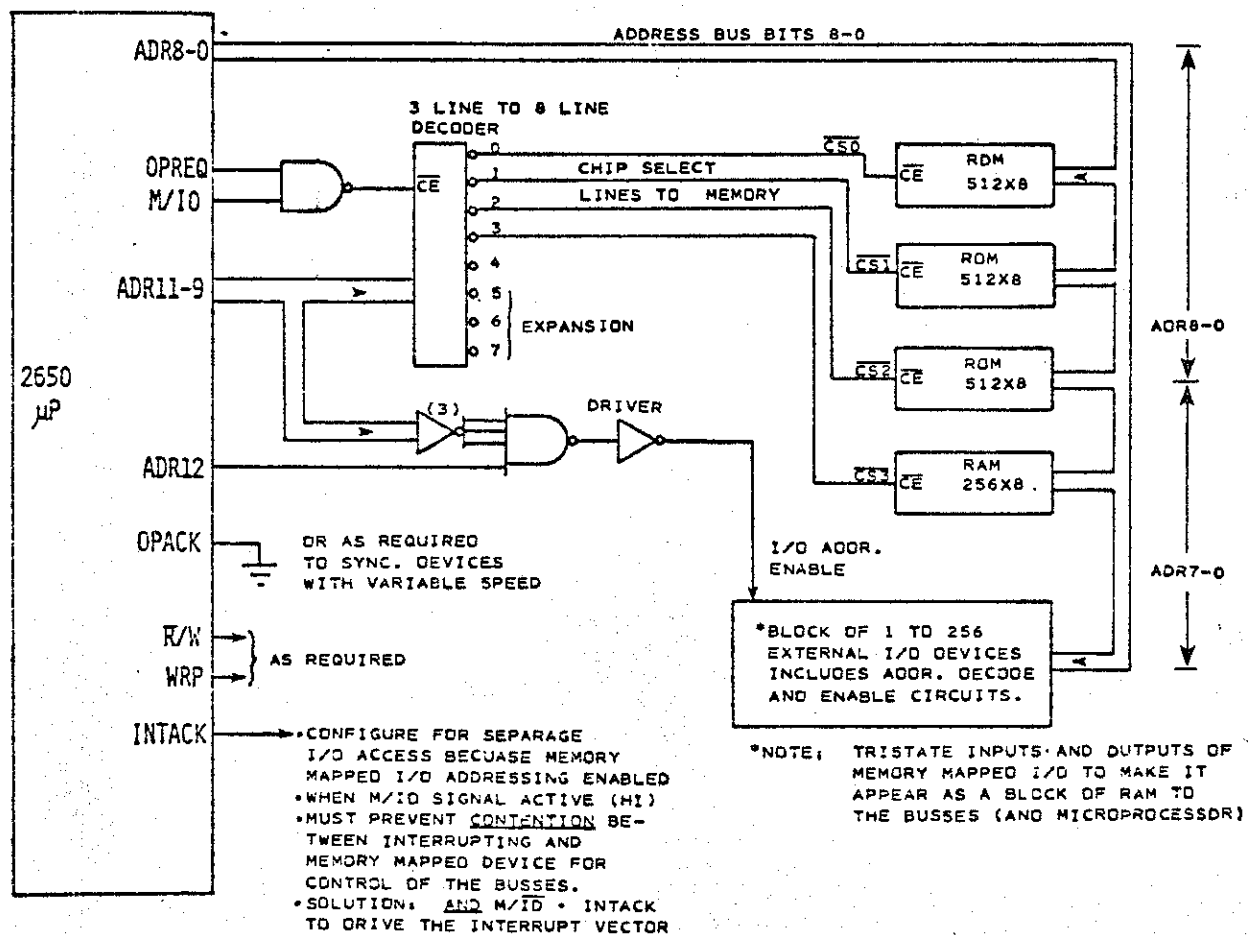
NOTES:

DIAGRAM 17

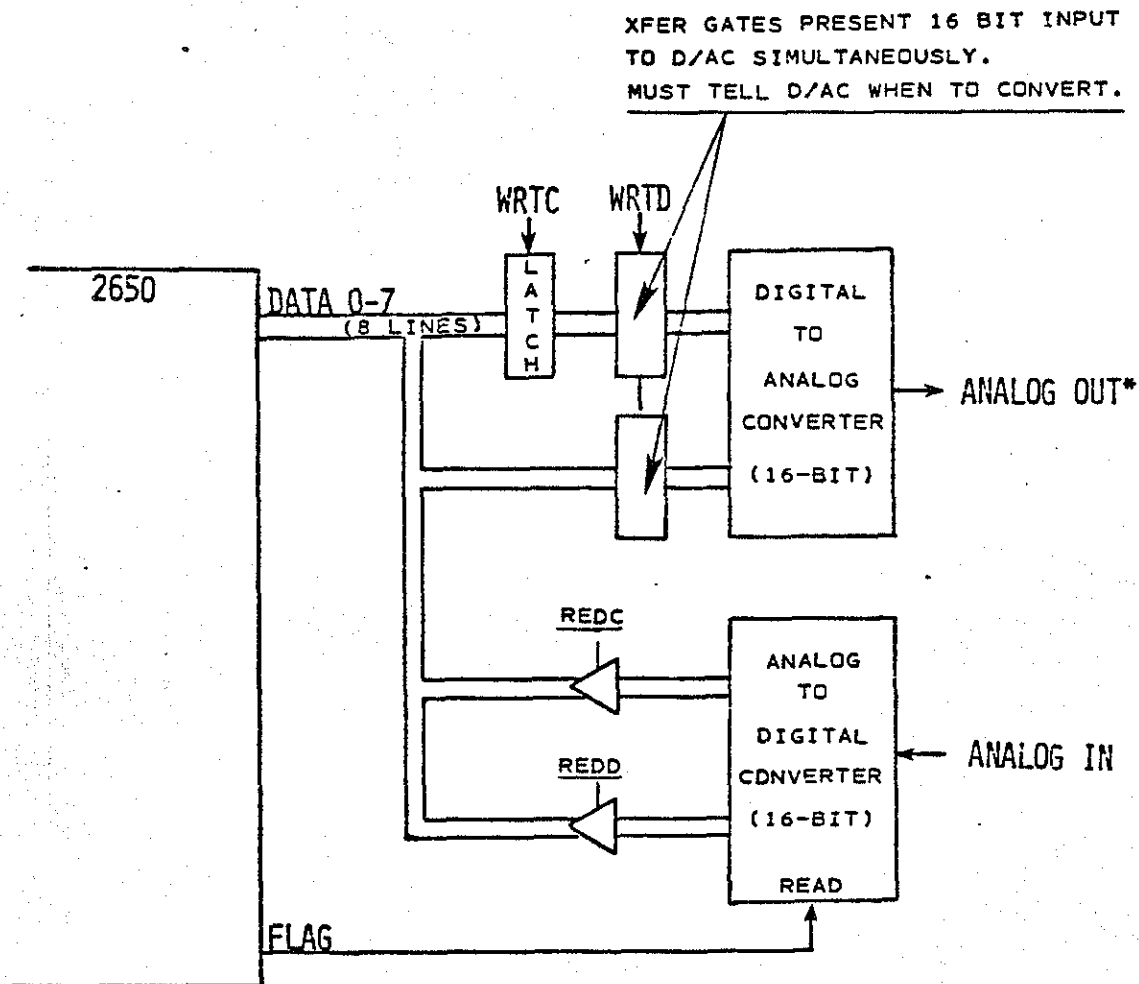
## MEMORY MAPPED I/O INTERFACE



## NOTES:

DIAGRAM 18

## 16-BIT A/D AND D/A CONVERTER INTERFACE



\*MUST BE PRECISE VALUE DESIRED, CAN NOT STEP IN 8-BIT INCREMENTS.  
NOTE LOGICAL CONTROL USING NON EXTENDED I/O INSTRUCTIONS.

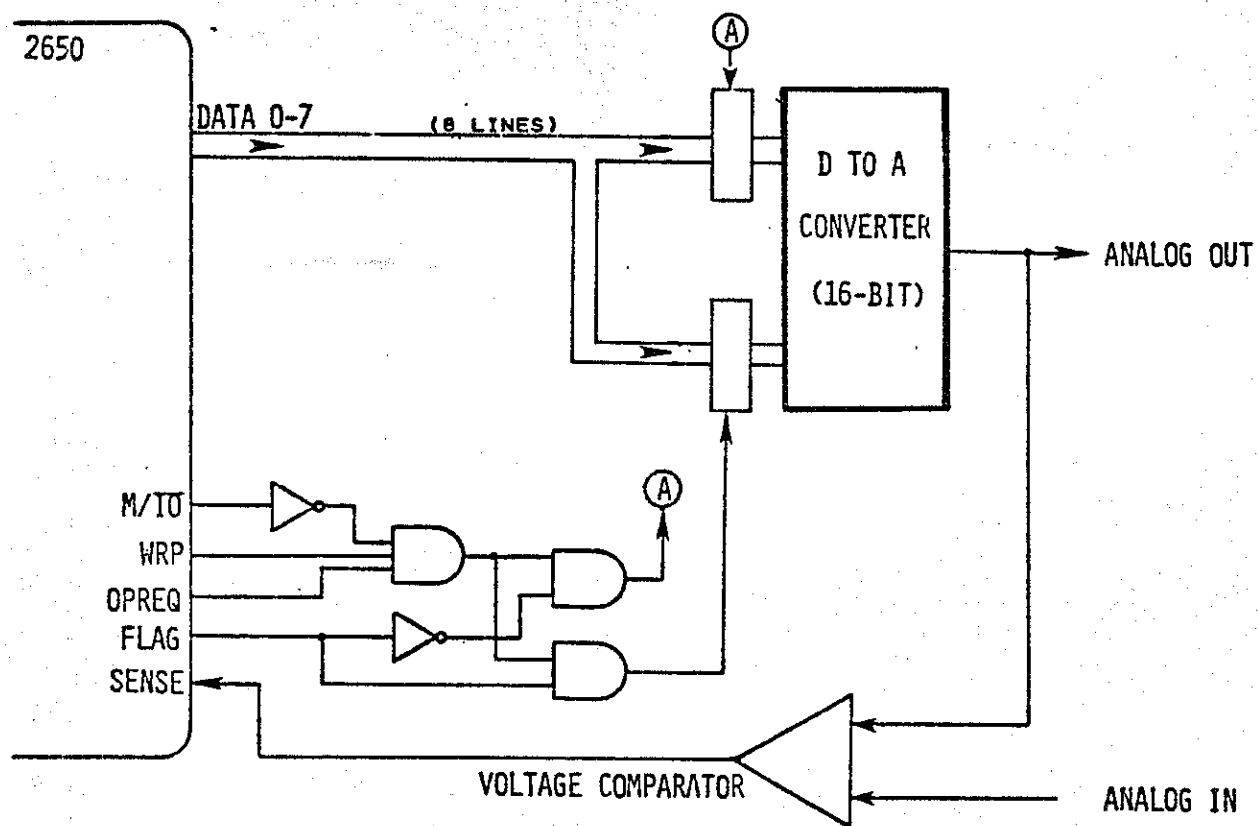
NOTES: Same concept can be applied to other multi-byte driven devices.

1. For output synchronization, set up series of latches. Store the data from microprocessor in 1 byte increments. Manipulate control lines to enable simultaneous input of stored data to output device. Choice of instructions is key consideration.
2. For read back of digital value of device input, instructions will be executed to access 1-byte increments. Logic may be configured to FREEZE the input analog signal so that digital value is stable for time duration of microprocessor access.
3. Suitable decodes and control line manipulation can be employed for memory mapped and extended I/O modes.



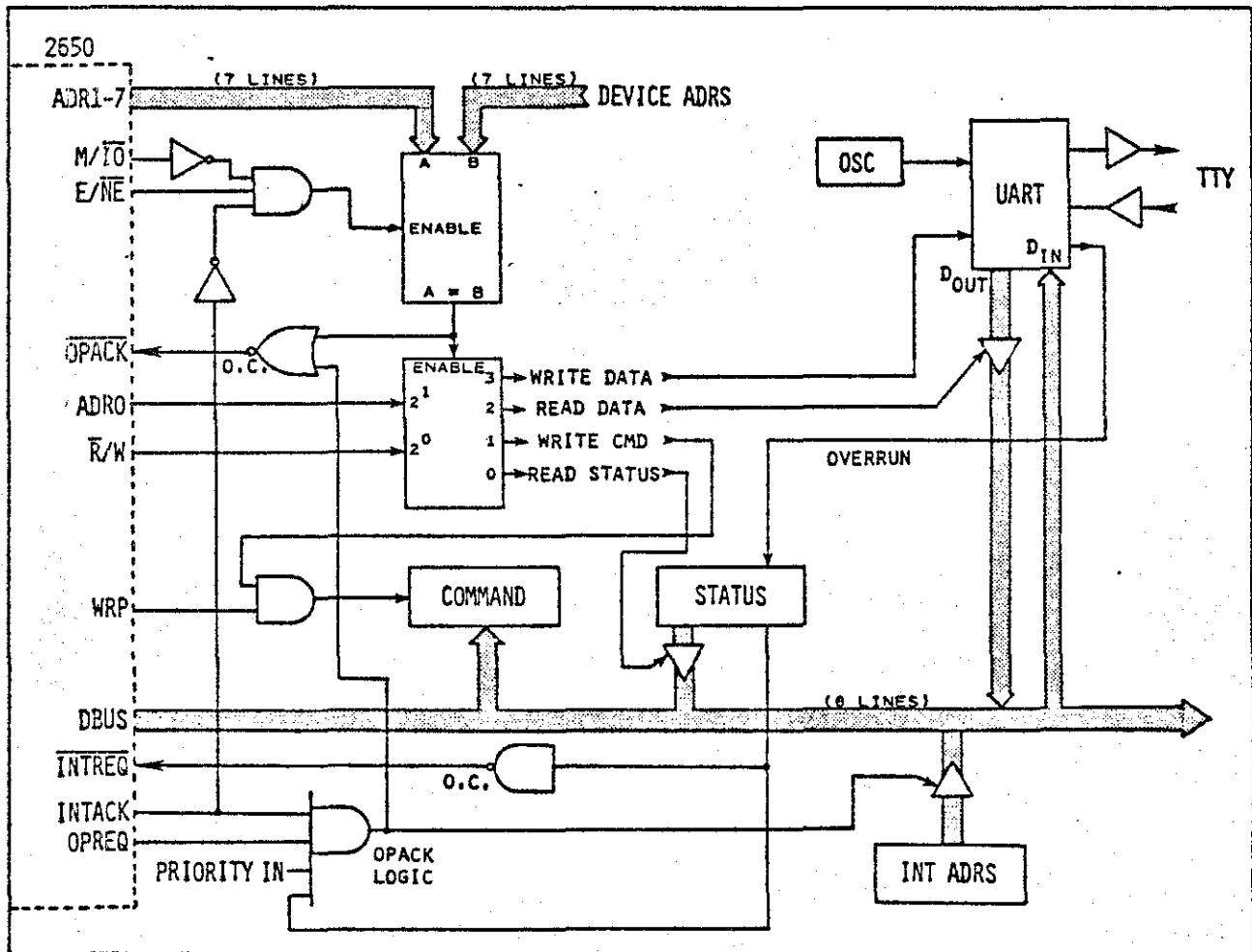
DIAGRAM 19

## INTEGRATED A/D &amp; D/A CONVERTER

NOTES:

REFERENCE: Further information concerned with DAC and ADC interface is provided in Module IV-K.

DIAGRAM 20      GENERAL PURPOSE SERIAL COMMUNICATIONS INTERFACE



NOTES:

## INTERRUPTS

### INTRODUCTION:

The subject of device-initiated interrupts and their handling by the microprocessor is quite complex. In previous course modules, particularly this which details I/O operation, you've seen interrupts referenced through the exercising of signals INTREQ and INTACK. A thorough understanding of the interrupt mechanism can be achieved only if it is preceded by a complete study of I/O signal definition and control sequencing. Also, the interrupt mechanism study borrows heavily from your knowledge of the Return Address Stack, and subroutine control, specifically in the use of the ZBSR instruction. Further, the PSW is affected; specifically, the INTERRUPT INHIBIT BIT (PSU Bit 5) is set whenever an interrupt is processed by the microprocessor. Subsequent reset of II is enabled by RESET, or by execution of appropriate PSL control instructions (CPSL, LPSL), or by the RETE (return from subroutine and enable interrupt-conditional) instruction. The ground work has been laid, now let's address the subject of INTERRUPT definition and control.

### POLLED vs. VECTORED INTERRUPT

The substance of an interrupt is that any suitable connected external device may request the microprocessor to stop its normal processing and service that device, simply by pulling down the signal INTREQ. The microprocessor completes the current instruction then acknowledges the interrupt by raising INTACK. Since INTREQ is low when active, it may be "wired-or'd" to all devices which might interrupt. There are two major methods by which the microprocessor may determine which device has requested interrupt service. These are "polled" and "vectored" interrupts.

Most microprocessors on the market today implement a POLLED interrupt scheme. Upon a receipt of an INTREQ, the microprocessor exits from its main program to a general INTERRUPT SERVICE routine. The first part of this routine consists of a POLLING sequence, in which each device's (ext. I/O port) status is interrogated to see which device raised the request. Normally, priority resolution is established by the order in which each device is interrogated. Once the microprocessor finds the interrupting device, it may execute a routine to service that device directly. It is immediately apparent that the TIME required to determine which device forced an interrupt is a direct function of that device's priority. In today's world of high speed electronics, time may spell the difference between proper device operation and ... disaster! Simply, an interrupt will not be posted unless the originating device needs attention - NOW!

INTRODUCTION TO INTERRUPTS (Continued)

Typical I/O POLLING sequences were described in Module IV. And, if the time necessary to poll devices does not affect their desired performance, polling within an interrupt scheme may be entirely suitable to your application.

Let's consider a VECTORED interrupt scheme. Like polled interrupt, any attached I/O device may demand service by pulling down the line INTREQ. Upon completion of the current instruction's execution, the 2650 requests that the device place its INTERRUPT ADDRESS on the DBUS. Within microseconds, the microprocessor employs the received address as the 2nd byte of a forced ZBSR instruction, executed as part of the interrupt recognition process. Execution of the ZBSR instruction causes the processor to vector to a 2-byte field relative to memory location zero. In a scheme where many external devices are connected for interrupts, the interrupting device should include the indirect address bit in the address byte which it delivers. The microprocessor thus may access any location in memory specified by the 2-byte indirect address. Just which 2-byte address relative to location zero is determined by the interrupting device's address vector.

Given that the ZBSR relative address range is +63 and -64 bytes from location 0, and that each indirect address requires 2 memory bytes, up to 63 separate interrupt vectors may be defined. This concept is illustrated in Diagram 21 (next page). Note, however, that designation of locations for interrupt addresses subtracts from those available for access by other page zero byte zero relative addressed instructions (ZBRR,ZBSR). There is nothing to prevent indirect access of one of the interrupt service routines through execution of a ZBSR instruction within any of the program's normal (non-interrupt servicing) routines.

DIRECTION:

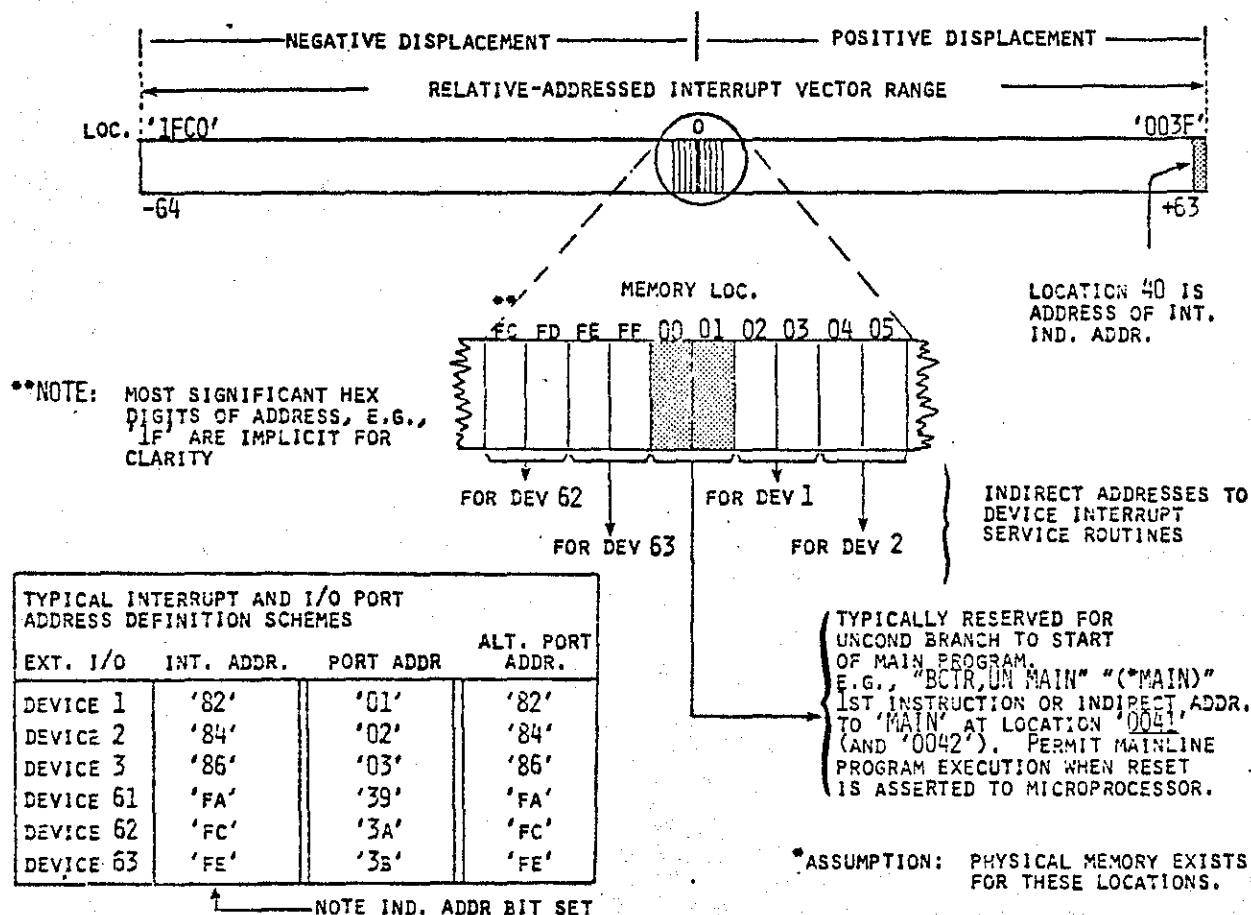
Take a few minutes to study the interrupt address organization diagram (next page). Then continue reading:

It is normal to assign a block of memory locations (relative to address '0000') for definition of the service routine indirect addresses. For example, you might assign negatively displaced addresses (e.g., loc 1FC0 - 1FFF) to provide indirect addressing to 32 distinct interrupt service routines (representing interrupt access by that many attached devices).

When RESET is input to the processor, execution of a BCTR,UN instruction with second byte = 'BF' causes the processor to continue program execution anywhere in memory at the (2 byte) address defined in locations 41 and 42. Recall that the (next instruction) pointer is at loc '02' when the BCTR instruction at loc '00' is executed. And location '41' is within the maximum positive displacement of the relative address.

DIAGRAM 21

## MEMORY ORGANIZATION FOR INTERRUPT ADDRESSES



## NOTES:

Examine the I/O interrupt and port address scheme table.

There is no principle violated by not assigning sequential binary coded decimal (BCD) port numbers to the I/O devices on hand. You can use any interface scheme that you wish, consistent with address decode schemes previously described in this module. A useful rule of design, however, is to minimize hardware components and wiring as much as possible. Under one scheme, device 1's port address might be '01.' Alternatively, you could assign '82' as Device 1's port address. With suitable decode and gating, '82' would be placed on the D Bus when Device 1 interrupts. During normal operations, the program would reference Device 1 as Port '82' e.g., WRTE, R3 PORT '82'; REDE, R0 PORT '82.'

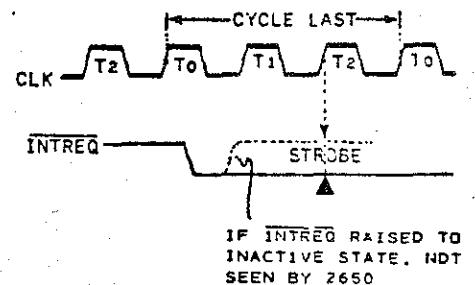
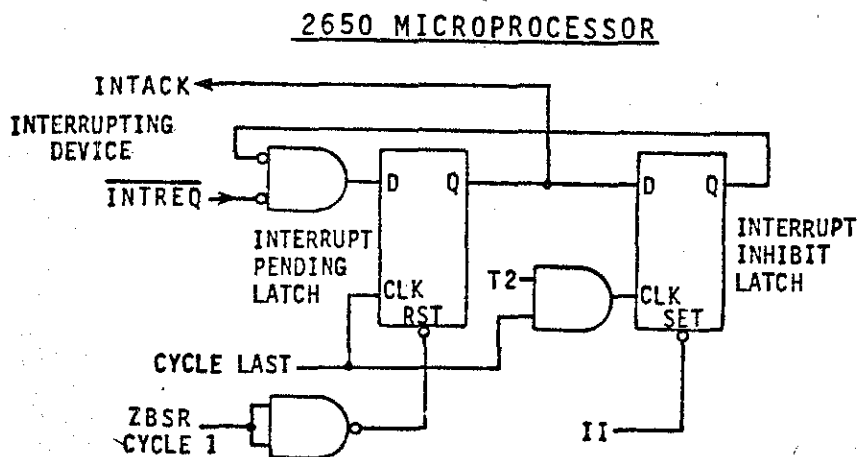
## MICROPROCESSOR INTERNAL INTERRUPT SEQUENCE

The following steps define precisely the internal interrupt handling sequence by the 2650 microprocessor.

1. The INTERRUPT INHIBIT (II-PSU Bit 5) must be cleared:
  - By input of RESET to the 2650, or
  - By execution of the CPSU instruction; e.g., CPSU II:'74''20,' or
  - By execution of a RETE instruction at the conclusion of a subroutine, e.g., RETE, EQ:'34'; RETE, UN '37'
2. The external device desiring interrupt service sets INTREQ low active.
  - Normally, this latches INTREQ until the interrupt request is acknowledged by the microprocessor.
  - With a multiple of devices capable of generating INTREQ, a priority recognition and grant design is normally included. This is discussed later in this section. By implementing a simple priority network, the microprocessor will not be confused with an interrupt request originating with more than 1 device.
3. The processor finishes its execution of the current instruction, then:
  - The 2650 strobes and latches INTREQ internally, coincident with the rising edge of T2 clock within CYCLE LAST (Diagram 22). Recall cycle last is the final cycle of any instruction, regardless of the number of cycles required.

DIAGRAM 22

INTREQ STROBE & LATCH



- During CYCLE LAST, the first byte of the next instruction is normally read from memory. If INTREQ is recognized by the 2650, it
  - Sets the II bit in the PSU.
  - Jams the first byte of a ZBSR instruction into the instruction register. The code is 'BB'. The 1st byte of the next instruction is not read into the instruction register from memory, and INTACK is raised active.

4. The processor accesses the data bus to retrieve the 2nd byte of the ZBSR instruction. Takes 2nd byte when OPREQ raised.

- During the 1st cycle of the jammed ZBSR execution, the I/O control lines are conditioned:

$$\left. \begin{array}{l} E/\overline{NE} = \overline{NE} \\ D/\overline{C} = \overline{C} \\ \overline{R}/W = \overline{R} \end{array} \right\} \begin{array}{l} \text{during } T_0 \text{ time in the first cycle.} \\ \text{e.g., all specified lines are driven LOW.} \end{array}$$

- When OPREQ is raised (at  $T_1$  time), the responding device must have provided a stable INTERRUPT VECTOR address on the D BUS. OPACK is optional, depending on device speed.
- Upon reading the interrupt VECTOR, the 2650 drops OPREQ and INTACK. The interrupting device drops INTREQ.
- The interrupting device puts its address on the DBUS when it recognizes INTACK.

NOTE: Complete timing is diagrammed later in this section.

5. The processor saves the address of the next instruction it would have executed had there not been an INTREQ.

- The return address STACK POINTER (PSU bits 2,1, and 0) is incremented.
- During the second cycle of ZBSR execution.
- The next address is saved in the first available return address stack location.

6. The processor executes the ZBSR instruction.

- If indirect address is specified, 2 machine cycles are added.
  - 2-byte address at relative location specified in the INTERRUPT vector is fetched into the Instruction Address Register.

- Program execution continues in memory at location specified by the indirect address.
  - If direct address is specified, the processor executes instructions starting at the relative location (from page 0, byte 0) specified in the INTERRUPT VECTOR.
7. Upon completion of the INTERRUPT SERVICE ROUTINE, the processor executes a RETC or RETE instruction.
- The stack pointer is decremented by ONE.
  - The saved address is moved into the Instruction Address Register.
  - If the RETC instruction is executed, the II bit is not disabled. Subsequent interrupt requests will not be recognized.
  - If a RETE instruction is executed, the II bit is disabled.
  - If the condition specified in the RETE (RETC) instruction is not satisfied, execution falls through to the next instruction in the subroutine. No return is made. The II bit is not reset.

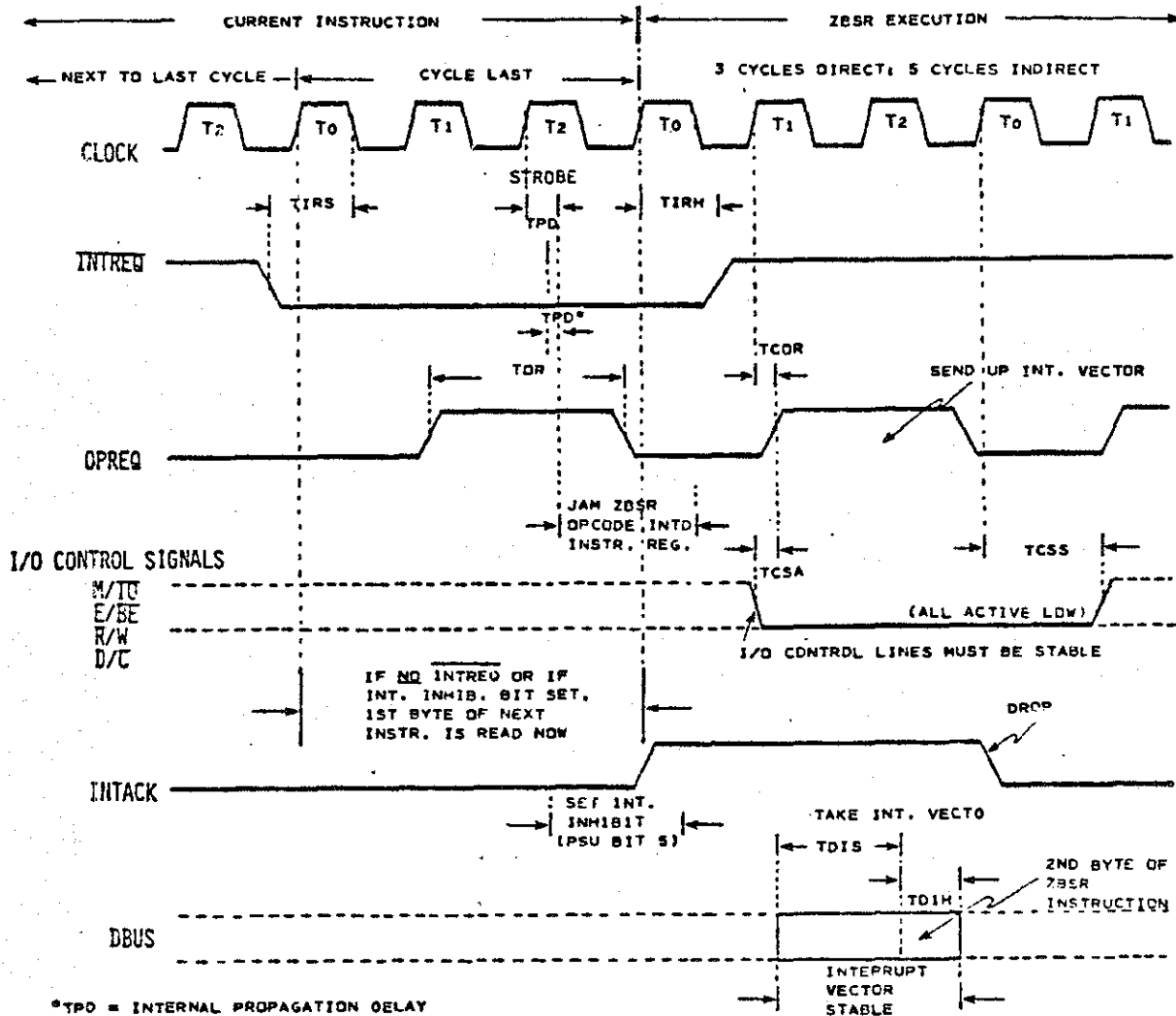
**DIRECTION:**

Go right on to the next page.



INTERRUPT SEQUENCE TIMING**DIRECTION:**

Diagram 23 illustrates the minimum timing requirements which will permit the microprocessor to recognize an input INTREQ and to vector program control to the interrupt service routine. Listen to tape 10B.

**DIAGRAM 23****2650 INTERNAL INTERRUPT RECOGNITION**

\*TPD = INTERNAL PROPAGATION DELAY

| PARAMETER |                          | LIMITS           |              | UNIT         |    |
|-----------|--------------------------|------------------|--------------|--------------|----|
|           |                          | MIN              | MAX          |              |    |
| TIRS      | INTREQ SET UP TIME       | -                | 150          | NS           |    |
| TIRH      | INTREQ HOLD TIME         | 0                | -            | NS           |    |
| TCR       | OPREQ PULSE WIDTH        | 2TCH+TCL         | 2TCH+TCL+100 | NS           |    |
| TCDR      | CLOCK TO OPREQ TIME      | 2650A-1<br>2650A | 100<br>150   | 200<br>300   | NS |
| TCSS      | CONTROL SIGNAL STABLE    | 2650A-1<br>2650A | 100<br>100   | 400<br>500   | NS |
| TCSA      | CONTROL SIGNAL AVAILABLE |                  | 200          | -            | NS |
| TDIS      | DATA IN STABLE           |                  | -            | 2TCH+TCL-200 | NS |
| TDIH      | DATA IN HOLD             |                  | 50           | -            | NS |

EXERCISE 2650 MICROPROCESSOR VECTORED INTERRUPT RECOGNITION**DIRECTION:**

1. Load the program "INTPT" into the INSTRUCTOR at the addresses indicated.
2. Execute the program in SINGLE STEP mode. Note step 9 then return to step 3.
3. At any time during execution in STEP mode:
  - A. Toggle the INTERRUPT SELECT switch to DIRECT OR INDIRECT.
  - B. Depress the interrupt key **INT**.
4. As you **STEP**, predict then note the SEQUENCE of execution as indicated in the display. Set the program counter to zero as often as necessary.

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| +                                        | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| +                                        | 1E | 1F | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C |
| +                                        | 2D | 2E | 2F | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B |
| +                                        | 3C | 3D | 3E | 3F | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A |
| +                                        | 4B | 4C | 4D | 4E | 4F | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| +                                        | 5A | 5B | 5C | 5D | 5E | 5F | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 |
| +                                        | 69 | 6A | 6B | 6C | 6D | 6E | 6F | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| +                                        | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F | 80 | 81 | 82 | 83 | 84 | 85 | 86 |
| +                                        | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F | 90 | 91 | 92 | 93 | 94 | 95 |
| +                                        | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F | A0 | A1 | A2 | A3 | A4 |
| +                                        | A5 | A6 | A7 | A8 | A9 | AA | AB | AC | AD | AE | AF | B0 | B1 | B2 | B3 |
| +                                        | B4 | B5 | B6 | B7 | B8 | B9 | BA | BB | BC | BD | BE | BF | C0 | C1 | C2 |
| +                                        | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF | D0 | D1 |
| +                                        | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | DA | DB | DC | DD | DE | DF | E0 |
| +                                        | E1 | E2 | E3 | E4 | E5 | E6 | E7 | E8 | E9 | EA | EB | EC | ED | EE | EF |
| +                                        | F0 | F1 | F2 | F3 | F4 | F5 | F6 | F7 | F8 | F9 | FA | FB | FC | FD | FE |
| +                                        | FF |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

INDIRECT RELATIVE ADDRESSING: ADD 128 TO DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE INTPT START ADDR CDESCRIPTION THIS PROGRAM RECOGNIZESVIA A METHOD TO SET THE INTERRUPT SELECTQUEUEING IN SLOW TIME (SINGLE STEP)MODEROUTINE SHEET 1 OF 1MEMORY LOCATIONS THIS SHEET 

| ADDRS | DATA<br>B3 B2 B1 B0 | LABEL  | SYMBOLIC INSTRUCTION<br>OPCODE | OPERANDS | COMMENT                   |
|-------|---------------------|--------|--------------------------------|----------|---------------------------|
| 1     |                     |        |                                |          | ON ENTRY, DEFEAT IN-      |
| 2     | 0000 76 20          | INTPT  | RPSU                           | II       | TERUPT RECOGNITION AND    |
| 3     | 2 1B 3C             |        | BCTR                           | MAIN     | EXIT TO MAIN (PROGRAM)    |
| 4     |                     |        |                                |          | IF INT SEL SW IS INDIRECT |
| 5     | 0007 00 20          | INTSV1 | ACON                           | INTSVC   | GO EXECUTE INTSVC OTHER   |
| 6     | 1 C0                |        | NOP                            |          | WUE EXECUTE THIS ROUTINE  |
| 7     | A C0                |        | NOP                            |          | THEN RETURN & ENABLE      |
| 8     | 8 37                |        | RETE UN                        |          | INTERRUPTS                |
| 9     |                     |        |                                |          |                           |
| 10    |                     |        |                                |          |                           |
| 11    | 002D C0             | INTSVC | NOP                            |          | ON ENTRY DO 3 NOPS AND    |
| 12    | 1 C0                |        | NOP                            |          | RETR & EN INTERRUPT       |
| 13    | 2 C0                |        | NOP                            |          |                           |
| 14    | 1 37                |        | RETE UN                        |          |                           |
| 15    |                     |        |                                |          |                           |
| 16    | 004D C0             | MAIN   | NOP                            |          | ON ENTRY - EXECUTE 3 NOPS |
| 17    | 1 C0                |        | NOP                            |          | before enabling interrupt |
| 18    | 2 C0                |        | NOP                            |          | recognition               |
| 19    | 3 74 20             |        | CPSU                           | II       |                           |
| 20    | 5 C0                | LOOP   | NOP                            |          | then loop forever in      |
| 21    | 6 C0                |        | NOP                            |          | remainder of MAIN         |
| 22    | 7 C0                |        | NOP                            |          |                           |
| 23    | 8 1B 7B             |        | BCTR UN                        | LOOP     |                           |

5. Try keying  while "INTSV1" (direct address) or "INTSVC" (indirect address is being executed).

6. Repeat steps 3 to 5 until you can predict the sequence of program flow.

NOTE: If the INT. SEL. SWITCH is set to DIRECT, the processor will attempt to execute the (invalid) code '00.' In this simplified program, this makes no difference, it executes about the same as a NOP. In sophisticated programs, avoid use of INDIRECT and DIRECT address identifiers within a given device interrupt address.

7. At location '46,' substitute a HALT ('40') for the NOP ('C0'). What happens when you depress  at location '46.'

(a) Before depressing  ? \_\_\_\_\_

(b) After depressing  ? \_\_\_\_\_

8. If a HALT is programmed at location '41,' can the program continue executing if  is depressed? \_\_\_\_\_ (yes)(no).

Explain: \_\_\_\_\_

NOTE: Consider straight line program execution from address '00' when answering this question.

9. You should examine the PSU (II - Bit 5 and SP Bits 2-0) several times during your investigation.

10. Answers to questions 7 and 8 are on the next page.

////////////////////////////////////

## 2650 INTERRUPT SCHEME

### INTRODUCTION:

During examination of the GENERAL PURPOSE SERIAL I/O INTERFACE (Diagram 20, page 31 this module), one of the signals which qualified input of the interrupt vector address to the microprocessor was PRIORITY IN. In this subsection, a short discussion of PRIORITY RESOLUTION for attached devices is offered.

Basically, priorities are assigned (to external devices in the order of their importance) in order to inhibit contention between the devices for interrupt recognition by the microprocessor. If the devices share an equal importance, priorities are assigned arbitrarily.

Diagram 24 illustrates in concept the configuration of hardware required by EACH DEVICE in order to generate an INTREQ to the microprocessor and to be responsive to the microprocessor's interrupt processing sequence.

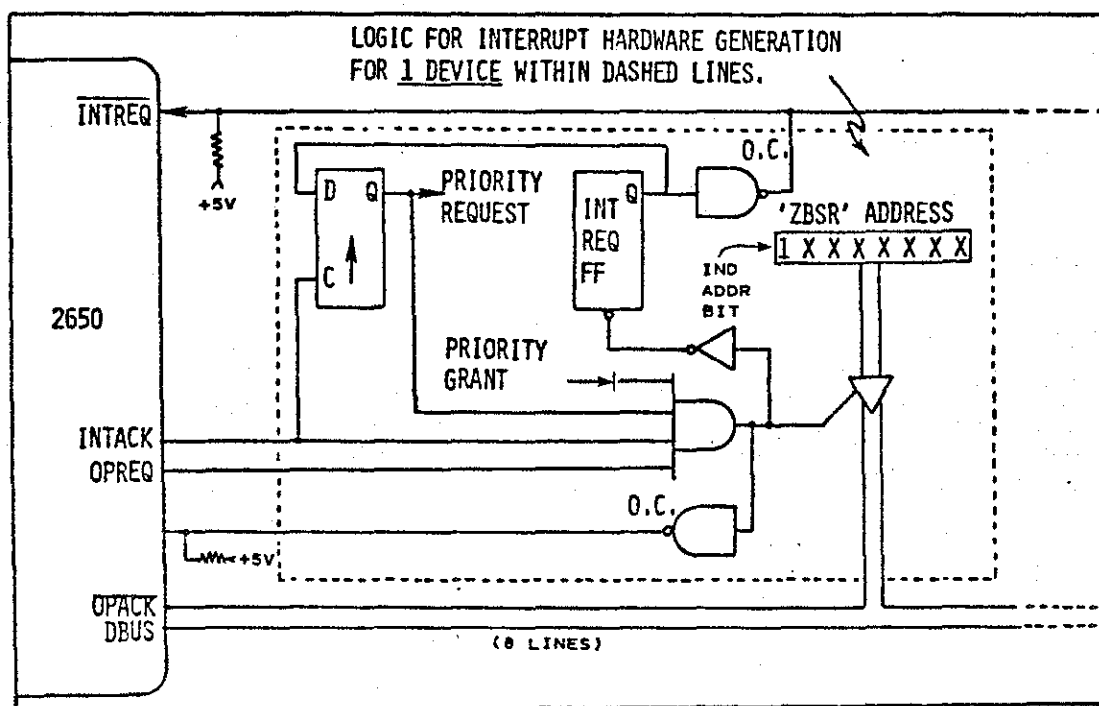
DIRECTION:

Listen to audio tape 10B for a brief discussion of Diagram 24.  
When directed to do so, go on to the next page.

NOTES:

DIAGRAM 24

## 2650 INTERRUPT HARDWARE CONCEPT



## ANSWERS TO QUESTIONS (steps 7 and 8; page 41)

7. BEFORE [INT] : RUN LED OFF                      AFTER [INT] : RUN LED ON  
                  DISPLAY BLANK                         DISPLAY INDICATES:  
                  MICROPROCESSOR                        '07' IF DIRECT ADDR INTRPT.  
                  IN HALT CONDITION.                    '20' IF INDIRECT ADDR INTRPT.
8. NO! II BIT IS SET. INTERRUPT MAY BE LATCHED BY EXTERNAL HARDWARE  
                         BUT THE MICROPROCESSOR CANNOT RECOGNIZE NDR  
                         ACKNOWLEDGE IT.

6. NO: II BIT IS SET. INTERRUPT MAY BE LATCHED BY EXTERNAL HARDWARE BUT THE MICROPROCESSOR CANNOT RECOGNIZE NDR ACKNOWLEDGE IT.

## PRIORITY RESOLUTION

### INTRODUCTION:

Priority resolution networks may be configured in one (or a combination of both) of the following methods. In concept, each is illustrated in Diagram 25.

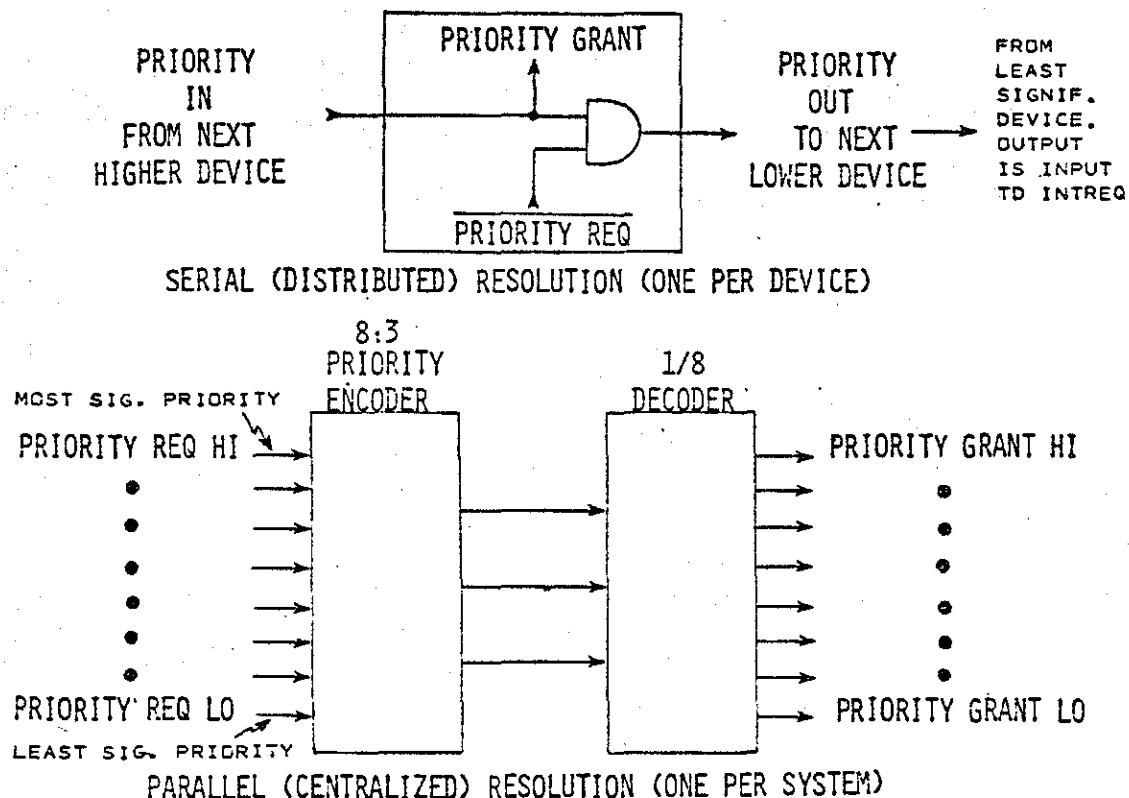
1. Serial or distributed resolution in which each device is assigned a priority circuit, with all circuits connected serially to each other in the order of their priority. This is commonly known as a "DAISY CHAIN."
2. Parallel or centralized resolution in which a single system of encode/decode logic determines priority grant based on the importance of the priority request. Device priority is determined by its connection order to the 8:3 encoder. The MOST significant device is connected to the highest priority input of the 8:3 encoder. The MOST significant device is connected to the highest priority input of the 8:3 encoder.

### DIRECTION:

Listen to the tape 10B for a brief commentary on Diagram 25. Then go on to the next page.

DIAGRAM 25

### PRIORITY RESOLUTION SCHEMES



## AN ALTERNATIVE APPROACH TO DEVICE INTREQ AND RESPONSE

INTRODUCTION:

Diagram 26 (next page) illustrates an interrupt request and response scheme which implements both serial (daisy chain) and parallel (centralized) priority recognition qualities. The major advantage is that hardware associated with this scheme is minimized yet streamlined to perform necessary interrupt request and response sequences.

**.DIRECTION:**

Listen to tape 108 for a brief commentary on Diagram 26, then go on to the next page as directed.

NOTES:

This image shows a single sheet of white paper with horizontal blue or grey ruling lines. The lines are evenly spaced and run across the width of the page. There is no handwriting or printed text on the paper.

DIAGRAM 26

## INTERRUPT RECOGNITION AND RESPONSE INTERFACE

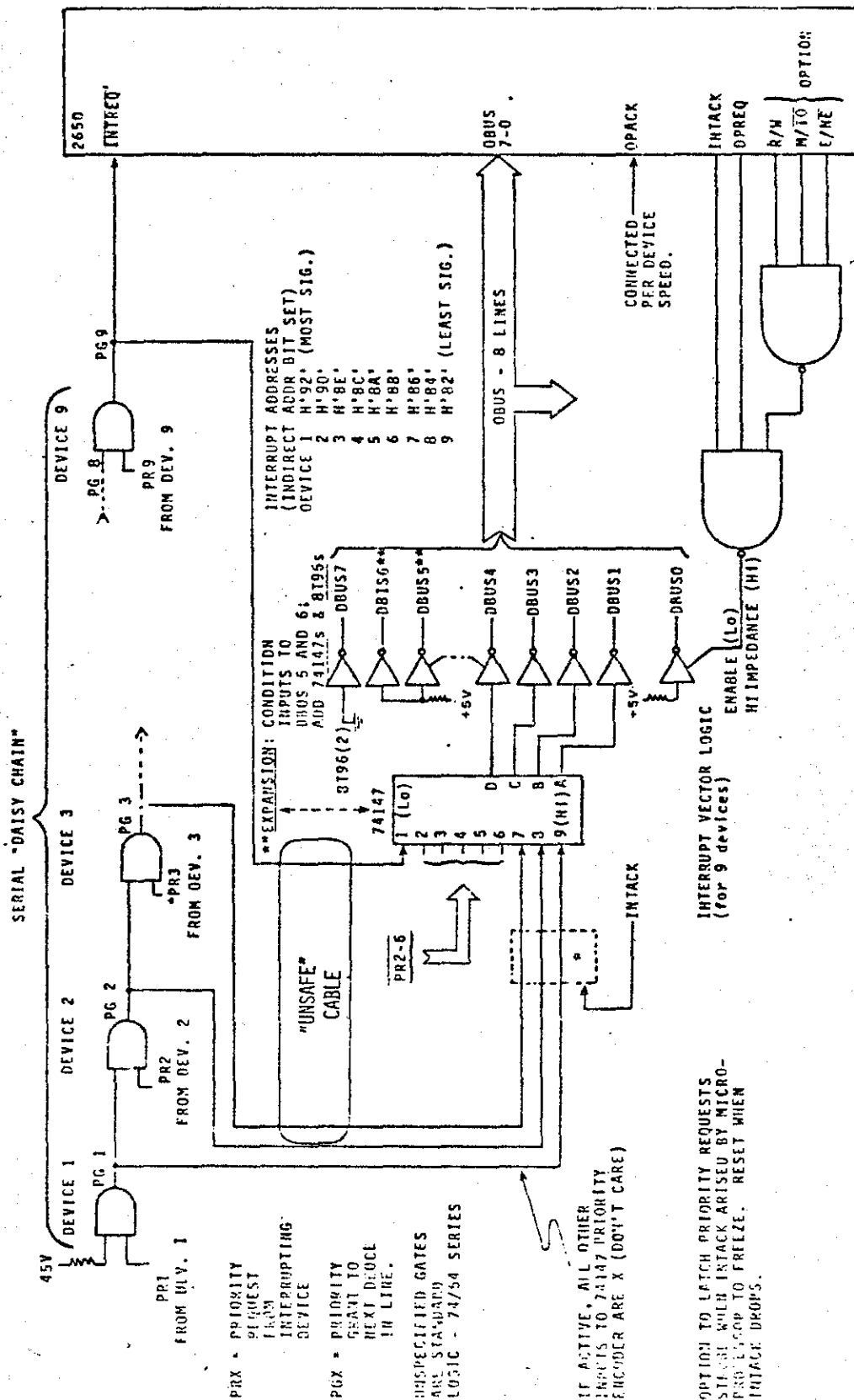
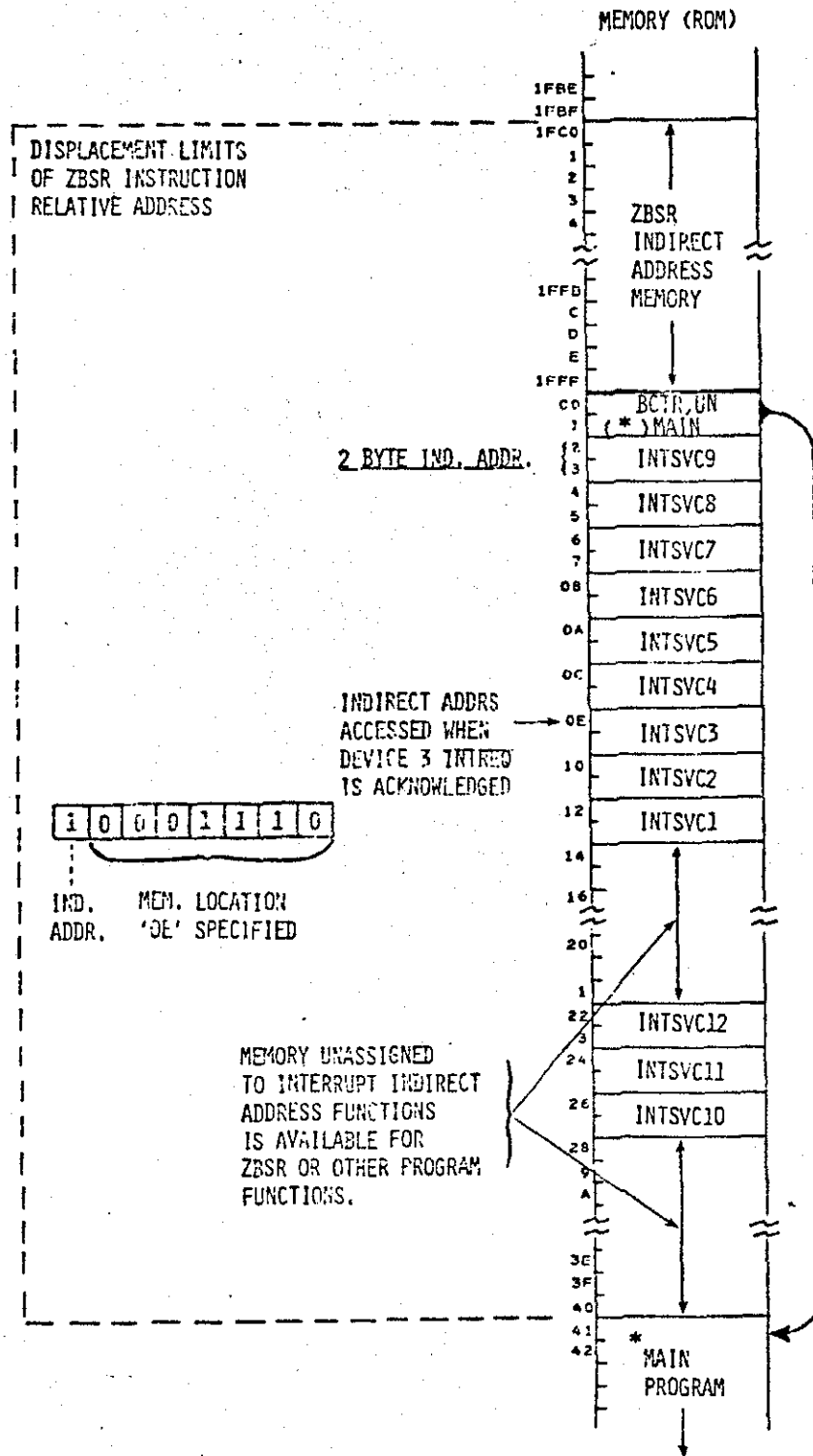


DIAGRAM 27

## ZBSR INDIRECT ADDRESS ORGANIZATION



\*OR 2-BYTE IND. ADDR. TO 'MAIN' AT LOCATIONS '41' AND '42'.



## INTERRUPT PROGRAM IMPLEMENTATION

### INTRODUCTION:

The following specific points are listed to provide a convenient reference when you design the software for interrupt service routines.

1. On entry to an interrupt service routine, SAVE the contents of all general purpose registers and the PSL: (See the NOTE under point 4 (below)).
  - Unless the interrupt is to take place after the processor has executed a HALT instruction (e.g., RUN/WAIT = WAIT)
  - Unless interrupts are permitted at specific times. This procedure involves reset of the INTERRUPT INHIBIT bit at specific points within the instruction sequence. If no interrupt is posted, the program may execute an instruction to close the interrupt recognition "WINDOW." (e.g., PPSU II).
  - If interrupt recognition is not windowed, the contents of the registers (and the PSL condition code) may be required for further processing upon return from the interrupt service routine.

Subsequent interrupts permitted after HALT or WINDOWING are considered SCHEDULED.

2. Interrupt recognition may be inhibited by employing the RETC rather than the RETE instruction to exit from a routine accessed via an interrupt generated ZBSR.
  - No CPSU II instruction required until interrupt enables are desired.
3. Do not nest subroutines to a level greater than the capacity of the Return Address Stack.
  - Interrupt generated ZBSR pushes the stack down 1 level.
4. Before exit from the Interrupt Service subroutine, restore the (saved) contents of the registers and the PSL to their original values.

NOTE: The INSTRUCTOR's USE (User System Executive) program contains routines for saving and restoring certain register's contents. Some of these routines may be accessed by the User program. Examples are illustrated later in this module.

**DIRECTION:** Go right on to the next page.

**CAUTION:**

In certain applications, you may be required to enable interrupts BEFORE execution of an interrupt service routine is completed. ENSURE THAT YOU CONSIDER ALL POSSIBLE PROGRAM SEQUENCE ERRORS THAT MIGHT RESULT, ESPECIALLY IF THE EXPECTED INTERRUPT IS UNSCHEDULED.

**REAL TIME INTERRUPTS****INTRODUCTION:**

One of the most useful applications involving the use of interrupts involves generation of REAL-TIME processor/controlled I/O sequences. In a typical real time application, the processor waits until a timed external interrupt is received from external hardware. The necessary multi-instruction sequence is then executed. The processor is then programmed to HALT or to LOOP FOREVER (e.g., BCTR,UN \$) after exiting from the interrupt service routine.

Some real-time applications specify the microprocessor and most system hardware to be powered down until a new interval is initiated. Systems placed in remote locations, where battery operation is a supporting factor, and pulsed monitoring security systems fall within this category.

There is one basic limitation involved in designing real-time application programs. The TIME required to process the desired sequence must NEVER EXCEED the duration between intervals! Otherwise, program overflow may take place. If internal and process times approach equality because of software constraints, you will probably design some of the functions in terms of hardware, particularly those functions which would require a long and time-consuming instruction sequence.

**MINIMUM HARDWARE REAL-TIME CLOCK**

Usage of real-time interrupts implies the generation of suitable interval clocking circuits in order to establish a reference with the external world. A few variations are illustrated in diagrams 28 and 29.

**DIRECTION:**

Listen to tape 11A for a short commentary on the real-time clock circuits in diagrams 28 and 29. Further directions are provided on tape.

DIAGRAM 28

## MINIMUM REAL TIME CLOCK GENERATION CIRCUIT

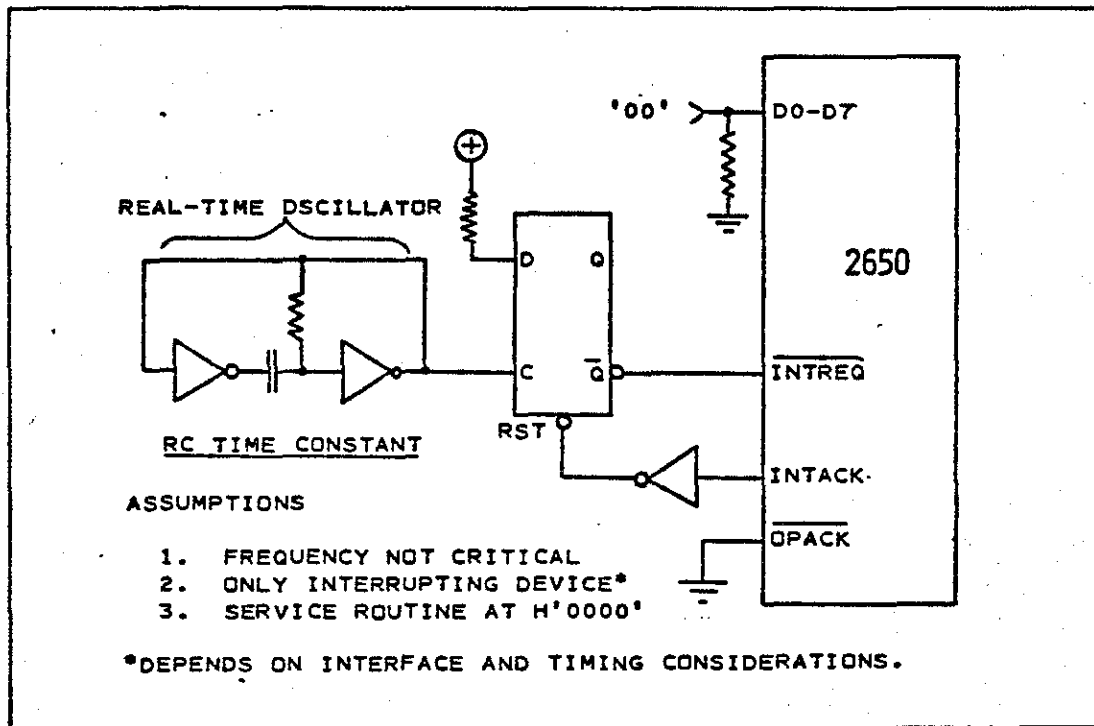
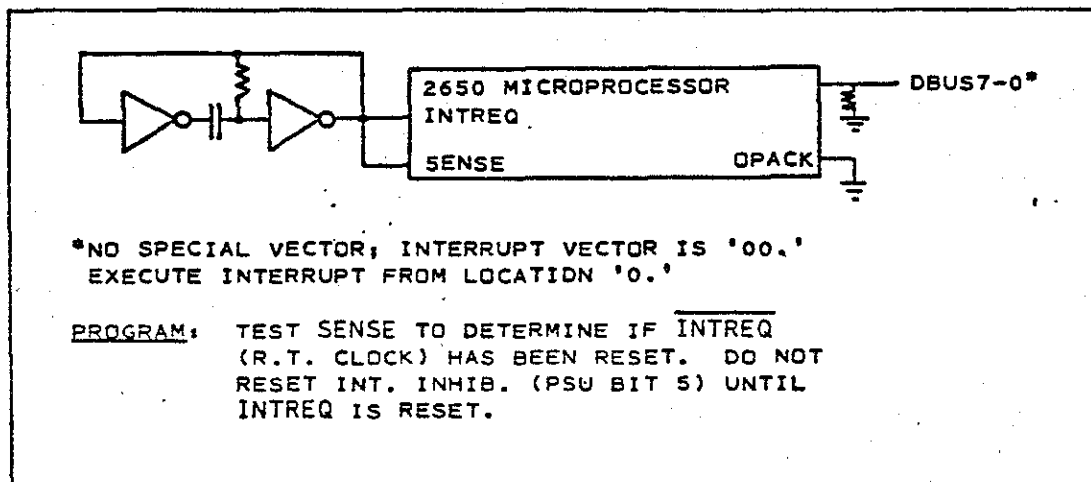


DIAGRAM 29

## REAL TIME CLOCK VARIATION 1



**NOTE:** One of simplest real-time clocks is AC Line Frequency (50 or 60 Hz). Interval is 20 msec (50 Hz) or 16.67 msec (60 Hz). AC FREQ is basis for the REAL-TIME CLOCK on the INSTRUCTOR.

DIAGRAM 28

MINIMUM REAL TIME CLOCK GENERATION CIRCUIT

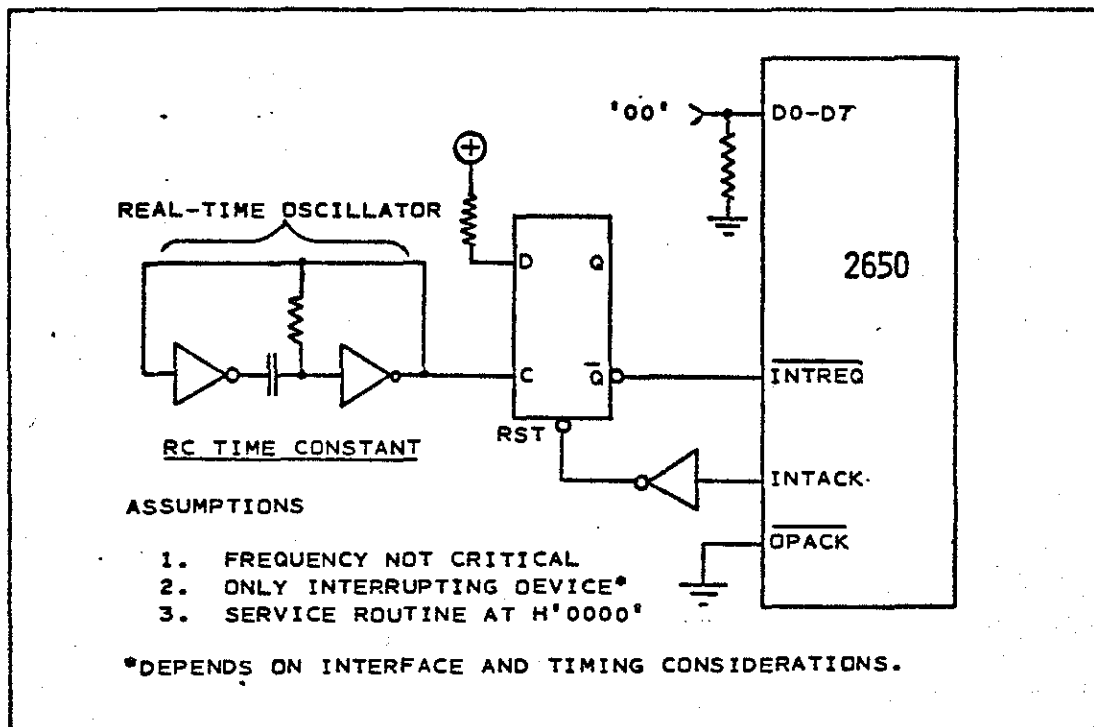
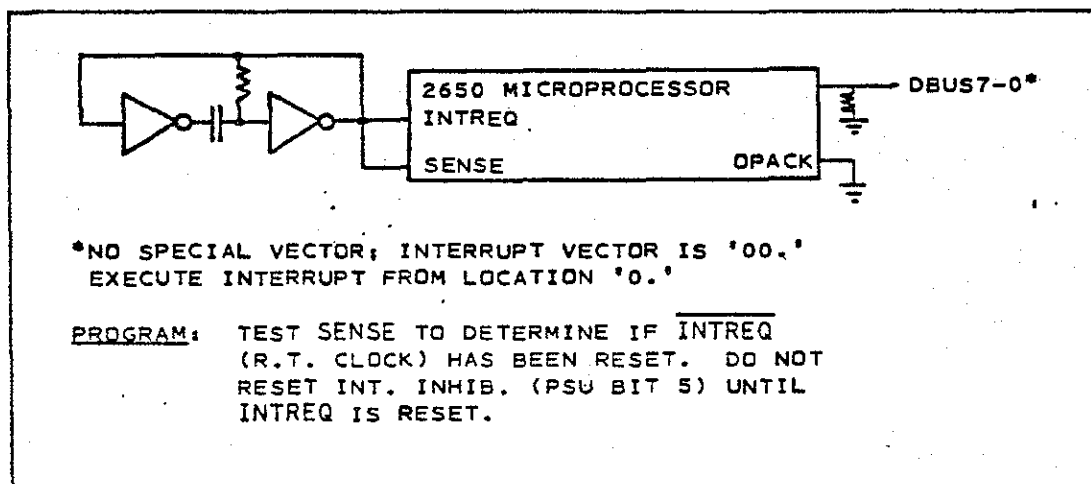


DIAGRAM 29

REAL TIME CLOCK VARIATION 1



**NOTE:** One of simplest real-time clocks is AC Line Frequency (50 or 60 Hz). Interval is 20 msec (50 Hz) or 16.67 msec (60 Hz). AC FREQ is basis for the REAL-TIME CLOCK on the INSTRUCTOR.

## "INSTRUCTOR 50" INTERRUPT SCHEME

## INTRODUCTION:

The following points outline the features of the "INSTRUCTOR's" interrupt scheme, which is diagrammed below and detailed thoroughly in the INSTRUCTOR Reference Manual.

● INTERRUPT SELECTION SWITCH

```
OIRECT Interrupt service at location '07.'
INDIRECT Interrupt service at memory add-
 ress specified by 2-byte address
 in locations '07' and '08.'
```

## ● INTERRUPT JUMPERS

(Located on reverse side of IN-  
STRUCTOR PCB)(see Diagram 30  
below).

- Internal (via INT function key).
- External (INTREQ from S10D Bus)

- REAL-TIME INTERRUPT SWITCH (located on reverse side of PCB)\*

\*IN LATEST VERSION OF THE INSTRUCTOR,  
THIS SWITCH IS LOCATED ON THE TOP  
PANEL. IGNORE THE NOTES ON THE RIGHT.

Slide switch in position closest  
to right side panel of "INSTRUCTOR"  
provides INTREQ clock at 60 Hz.

## KEYBOARD

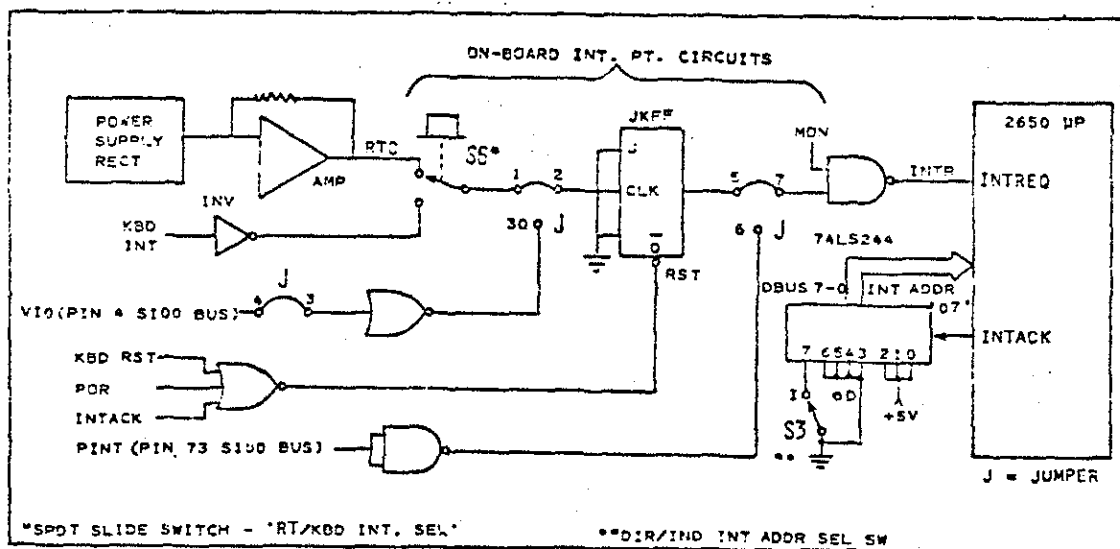
Slide switch in position furthest  
from RIGHT side panel. Routes  
keyboard interrupt to INTREQ  
generation circuits.

**DIRECTION:**

Listen to tape 11A for a brief description of Diagram 30.  
Further directions are provided on the tape.

DIAGRAM 30

### BLOCK DIAGRAM - INSTRUCTOR INTERRUPT CIRCUIT



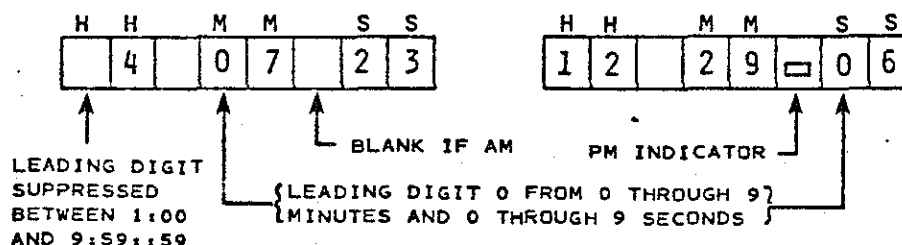
NOTES: Ext. processor INTPT (S100 BUS): connect jumpers 7-6.  
Ext. VIO INTPT (S100 BUS): connect jumpers 4-3;  
39-2 for inversion. Connect 4-2 non-inverting.  
Internal INTPT (on board RTC or K80 INT): connect  
jumpers 1-2; 5-7.

EXERCISE - REAL-TIME INTERRUPT FACILITYINTRODUCTION:

There is no better way to demonstrate the REAL-TIME INTERRUPT facility of the INSTRUCTOR than by implementing a functional DESK CLOCK in user software.

SPECIFICATION

The desk clock program (mnemonic DESCLK) is a full 4-function operational clock providing the exact time, in hours, minutes, seconds and AM/PM indication on the "INSTRUCTOR's" hex display. The typical readout is shown in two examples:



The "INSTRUCTOR's" switches are set as follows:

I/O SELECTION:

ANY POSITION

INTERRUPT ADDRESS:

DIRECT

REAL-TIME/NORMAL SLIDE SWITCH:

REAL-TIME

DESCLK executes from memory address '0'. Real-time interrupt (60 Hz) is inhibited until the clock is preset to the time of day. Clock preset operations are performed via access of the INSTRUCTOR's MOV, GNPA, DISLSD and USRDSP user-available subroutines. The following table summarizes the messages displayed during preset of the DESCLK:

| MESSAGE                     | DESCRIPTION                                                                                                                         | USER RESPONSE                                                                                                                                                                       |
|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SET CLOC                    | 2-second prompt which indicates PRESET routines are executing.                                                                      | Wait 2 seconds.                                                                                                                                                                     |
| .A or P.                    | Prompt to preset AM or PM indication.                                                                                               | <input type="checkbox"/> A <input type="checkbox"/> E/N if AM<br><input type="checkbox"/> E/N if PM                                                                                 |
| .HOURS.                     | Preset correct time in hours one or two digits $1 \leq N \leq 12$                                                                   | <input type="checkbox"/> N <input type="checkbox"/> E/N before 10:00<br><input type="checkbox"/> N <input type="checkbox"/> N <input type="checkbox"/> E/N between 10:00 and 12:59. |
| .FrACT.                     | Preset correct time in MINUTES one or two digits $0 < N < 59$ .                                                                     | Same as hours within limits.                                                                                                                                                        |
| .SECS.                      | Same as for minutes.                                                                                                                | Prior to setting seconds, telephone local time check. Response same as minutes.                                                                                                     |
| HHH MM SS<br>A<br>PM<br>INO | Display after entering seconds and depressing <input type="checkbox"/> E/N. FLAG light off. Clock is preset but <u>not</u> running. | Depress <input type="checkbox"/> SENS to start clock running. Indication: FLAG <u>on</u> and SECS "ticking" off.                                                                    |

**DIRECTION:**

Complete the steps to this exercise as indicated.

1. Take about 10 minutes to read the entire specification in the listing of DESCLK (page 54 - 62, this module).
  2. Code the instructions which are not previously coded. Compare your code with the suggested solution (short form) on page 63.
  3. Load DESCLK's code into memory and execute from address '0'.
  4. Respond to the following questions related to program "DESCLK."  
The answers are located on page 64 of this module.
- 
- A. When real-time interrupt service is not taking place, the current time of day is displayed. What is the RANGE of instruction addresses being executed within the main program?
  - B. If "DESCLK" is executed in a 50 Hz environment, what single instruction would you change to ensure that the clock is synchronized with real-time of day?
  - C. What change would you make to the program in order to permit the INTERRUPT ADDRESS selection switch to be set to INDIRECT?
  - D. Assume it is 10:00 a.m., exactly. If you pre-set that time, about how long a delay will transpire before the program would execute to a breakpoint at location 'BB'?
  - E. Given the same conditions as above, how long would you have to wait before the 2650 executed the instruction at location 'CE'?
  - F. What is the indirect address fetched by the microprocessor when it executes the instruction at location '15D'?
  - G. When are the NOPs (3) at location 'B5' executed?
  - H. During execution of "DESCLK," how many times is the routine "NOWSET" (at location 1B0) executed?
  - I. When the RETC,UN instruction (at location '1B5') is executed, what are the contents of R1?
  - J. What is the significance of executing a LODZ,R1 instruction at location '1A9'?

- K. Reference routine 'NOWSET' at location '180'. If R0 contains '37' when the program executes the NOP at location '189', what are its contents after program execution returns from the 'DISLSD' routine?
- L. If you wish to DOUBLE the running time of the message "SET CLOC" (from 2 seconds to 4 seconds), indicate the single program change you would make.
- M. Reference routine "ORDER" (location '1C0'). What would be the significance of changing the RETC,UN instruction (loc '1DA') to RETE,UN?
- N. List the conditions under which R0 will be loaded with a space code by executing the instruction at location '179'. Reference subroutine "BLNKZ" (location '170').
- O. Reference "DISCLK" at location '100' and "AMORPM" at location '140'. Does program execution cause a space code (H '17') to be written into 'DISCLK' message at location '102'? (e.g., could 'XX' be a part of this program at location '102'?)
- P. Assume that it is 12:00 Midnight and the clock is running. Indicate by location the sequence in which instructions in subroutine "AM PM" (location 'B8') are executed.
- Q. Reference routine "CONT" (location '13'). Before depressing SENS to synchronize "DESCLK," what is the sequence in which the instructions are performed.
- R. Assume the same conditions as in question Q, except you have just depressed SENS. Indicate by location the sequence in which the routine "CONT" is executed.

**DIRECTION:**

After comparing your responses to those provided on page 64, go on to page 65. Further directions are provided on that page.



| DIRECT RELATIVE ADDRESSING - SECOND BYTE |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|------------------------------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| +                                        | 00  | 01  | 02  | 03  | 04  | 05  | 06  | 07  | 08  | 09  | 0A  | 0B  | 0C  | 0D  | 0E  |
| +                                        | 10  | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 1A  | 1B  | 1C  | 1D  | 1E  |
| +                                        | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 2A  | 2B  | 2C  | 2D  | 2E  |
| +                                        | 30  | 31  | 32  | 33  | 34  | 35  | 36  | 37  | 38  | 39  | 3A  | 3B  | 3C  | 3D  | 3E  |
| +                                        | 40  | 41  | 42  | 43  | 44  | 45  | 46  | 47  | 48  | 49  | 4A  | 4B  | 4C  | 4D  | 4E  |
| +                                        | 50  | 51  | 52  | 53  | 54  | 55  | 56  | 57  | 58  | 59  | 5A  | 5B  | 5C  | 5D  | 5E  |
| +                                        | 60  | 61  | 62  | 63  | 64  | 65  | 66  | 67  | 68  | 69  | 6A  | 6B  | 6C  | 6D  | 6E  |
| +                                        | 70  | 71  | 72  | 73  | 74  | 75  | 76  | 77  | 78  | 79  | 7A  | 7B  | 7C  | 7D  | 7E  |
| +                                        | 80  | 81  | 82  | 83  | 84  | 85  | 86  | 87  | 88  | 89  | 8A  | 8B  | 8C  | 8D  | 8E  |
| +                                        | 90  | 91  | 92  | 93  | 94  | 95  | 96  | 97  | 98  | 99  | 9A  | 9B  | 9C  | 9D  | 9E  |
| +                                        | A0  | A1  | A2  | A3  | A4  | A5  | A6  | A7  | A8  | A9  | AA  | AB  | AC  | AD  | AE  |
| +                                        | B0  | B1  | B2  | B3  | B4  | B5  | B6  | B7  | B8  | B9  | BA  | BB  | BC  | BD  | BE  |
| +                                        | C0  | C1  | C2  | C3  | C4  | C5  | C6  | C7  | C8  | C9  | CA  | CB  | CC  | CD  | CE  |
| +                                        | D0  | D1  | D2  | D3  | D4  | D5  | D6  | D7  | D8  | D9  | DA  | DB  | DC  | DD  | DE  |
| +                                        | E0  | E1  | E2  | E3  | E4  | E5  | E6  | E7  | E8  | E9  | EA  | EB  | EC  | ED  | EE  |
| +                                        | F0  | F1  | F2  | F3  | F4  | F5  | F6  | F7  | F8  | F9  | FA  | FB  | FC  | FD  | FE  |
| +                                        | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 10A | 10B | 10C | 10D | 10E |
| +                                        | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 11A | 11B | 11C | 11D | 11E |
| +                                        | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 | 12A | 12B | 12C | 12D | 12E |
| +                                        | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 13A | 13B | 13C | 13D | 13E |
| +                                        | 140 | 141 | 142 | 143 | 144 | 145 | 146 | 147 | 148 | 149 | 14A | 14B | 14C | 14D | 14E |
| +                                        | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 | 15A | 15B | 15C | 15D | 15E |
| +                                        | 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 168 | 169 | 16A | 16B | 16C | 16D | 16E |
| +                                        | 170 | 171 | 172 | 173 | 174 | 175 | 176 | 177 | 178 | 179 | 17A | 17B | 17C | 17D | 17E |
| +                                        | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | 188 | 189 | 18A | 18B | 18C | 18D | 18E |
| +                                        | 190 | 191 | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 19A | 19B | 19C | 19D | 19E |
| +                                        | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 208 | 209 | 20A | 20B | 20C | 20D | 20E |
| +                                        | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 21A | 21B | 21C | 21D | 21E |
| +                                        | 220 | 221 | 222 | 223 | 224 | 225 | 226 | 227 | 228 | 229 | 22A | 22B | 22C | 22D | 22E |
| +                                        | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 | 23A | 23B | 23C | 23D | 23E |
| +                                        | 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 24A | 24B | 24C | 24D | 24E |
| +                                        | 250 | 251 | 252 | 253 | 254 | 255 | 256 | 257 | 258 | 259 | 25A | 25B | 25C | 25D | 25E |
| +                                        | 260 | 261 | 262 | 263 | 264 | 265 | 266 | 267 | 268 | 269 | 26A | 26B | 26C | 26D | 26E |
| +                                        | 270 | 271 | 272 | 273 | 274 | 275 | 276 | 277 | 278 | 279 | 27A | 27B | 27C | 27D | 27E |
| +                                        | 280 | 281 | 282 | 283 | 284 | 285 | 286 | 287 | 288 | 289 | 28A | 28B | 28C | 28D | 28E |
| +                                        | 290 | 291 | 292 | 293 | 294 | 295 | 296 | 297 | 298 | 299 | 29A | 29B | 29C | 29D | 29E |
| +                                        | 300 | 301 | 302 | 303 | 304 | 305 | 306 | 307 | 308 | 309 | 30A | 30B | 30C | 30D | 30E |
| +                                        | 310 | 311 | 312 | 313 | 314 | 315 | 316 | 317 | 318 | 319 | 31A | 31B | 31C | 31D | 31E |
| +                                        | 320 | 321 | 322 | 323 | 324 | 325 | 326 | 327 | 328 | 329 | 32A | 32B | 32C | 32D | 32E |
| +                                        | 330 | 331 | 332 | 333 | 334 | 335 | 336 | 337 | 338 | 339 | 33A | 33B | 33C | 33D | 33E |
| +                                        | 340 | 341 | 342 | 343 | 344 | 345 | 346 | 347 | 348 | 349 | 34A | 34B | 34C | 34D | 34E |
| +                                        | 350 | 351 | 352 | 353 | 354 | 355 | 356 | 357 | 358 | 359 | 35A | 35B | 35C | 35D | 35E |
| +                                        | 360 | 361 | 362 | 363 | 364 | 365 | 366 | 367 | 368 | 369 | 36A | 36B | 36C | 36D | 36E |
| +                                        | 370 | 371 | 372 | 373 | 374 | 375 | 376 | 377 | 378 | 379 | 37A | 37B | 37C | 37D | 37E |
| +                                        | 380 | 381 | 382 | 383 | 384 | 385 | 386 | 387 | 388 | 389 | 38A | 38B | 38C | 38D | 38E |
| +                                        | 390 | 391 | 392 | 393 | 394 | 395 | 396 | 397 | 398 | 399 | 39A | 39B | 39C | 39D | 39E |
| +                                        | 400 | 401 | 402 | 403 | 404 | 405 | 406 | 407 | 408 | 409 | 40A | 40B | 40C | 40D | 40E |
| +                                        | 410 | 411 | 412 | 413 | 414 | 415 | 416 | 417 | 418 | 419 | 41A | 41B | 41C | 41D | 41E |
| +                                        | 420 | 421 | 422 | 423 | 424 | 425 | 426 | 427 | 428 | 429 | 42A | 42B | 42C | 42D | 42E |
| +                                        | 430 | 431 | 432 | 433 | 434 | 435 | 436 | 437 | 438 | 439 | 43A | 43B | 43C | 43D | 43E |
| +                                        | 440 | 441 | 442 | 443 | 444 | 445 | 446 | 447 | 448 | 449 | 44A | 44B | 44C | 44D | 44E |
| +                                        | 450 | 451 | 452 | 453 | 454 | 455 | 456 | 457 | 458 | 459 | 45A | 45B | 45C | 45D | 45E |
| +                                        | 460 | 461 | 462 | 463 | 464 | 465 | 466 | 467 | 468 | 469 | 46A | 46B | 46C | 46D | 46E |
| +                                        | 470 | 471 | 472 | 473 | 474 | 475 | 476 | 477 | 478 | 479 | 47A | 47B | 47C | 47D | 47E |
| +                                        | 480 | 481 | 482 | 483 | 484 | 485 | 486 | 487 | 488 | 489 | 48A | 48B | 48C | 48D | 48E |
| +                                        | 490 | 491 | 492 | 493 | 494 | 495 | 496 | 497 | 498 | 499 | 49A | 49B | 49C | 49D | 49E |
| +                                        | 500 | 501 | 502 | 503 | 504 | 505 | 506 | 507 | 508 | 509 | 50A | 50B | 50C | 50D | 50E |
| +                                        | 510 | 511 | 512 | 513 | 514 | 515 | 516 | 517 | 518 | 519 | 51A | 51B | 51C | 51D | 51E |
| +                                        | 520 | 521 | 522 | 523 | 524 | 525 | 526 | 527 | 528 | 529 | 52A | 52B | 52C | 52D | 52E |
| +                                        | 530 | 531 | 532 | 533 | 534 | 535 | 536 | 537 | 538 | 539 | 53A | 53B | 53C | 53D | 53E |
| +                                        | 540 | 541 | 542 | 543 | 544 | 545 | 546 | 547 | 548 | 549 | 54A | 54B | 54C | 54D | 54E |
| +                                        | 550 | 551 | 552 | 553 | 554 | 555 | 556 | 557 | 558 | 559 | 55A | 55B | 55C | 55D | 55E |
| +                                        | 560 | 561 | 562 | 563 | 564 | 565 | 566 | 567 | 568 | 569 | 56A | 56B | 56C | 56D | 56E |
| +                                        | 570 | 571 | 572 | 573 | 574 | 575 | 576 | 577 | 578 | 579 | 57A | 57B | 57C | 57D | 57E |
| +                                        | 580 | 581 | 582 | 583 | 584 | 585 | 586 | 587 | 588 | 589 | 58A | 58B | 58C | 58D | 58E |
| +                                        | 590 | 591 | 592 | 593 | 594 | 595 | 596 | 597 | 598 | 599 | 59A | 59B | 59C | 59D | 59E |
| +                                        | 600 | 601 | 602 | 603 | 604 | 605 | 606 | 607 | 608 | 609 | 60A | 60B | 60C | 60D | 60E |
| +                                        | 610 | 611 | 612 | 613 | 614 | 615 | 616 | 617 | 618 | 619 | 61A | 61B | 61C | 61D | 61E |
| +                                        | 620 | 621 | 622 | 623 | 624 | 625 | 626 | 627 | 628 | 629 | 62A | 62B | 62C | 62D | 62E |
| +                                        | 630 | 631 | 632 | 633 | 634 | 635 | 636 | 637 | 638 | 639 | 63A | 63B | 63C | 63D | 63E |
| +                                        | 640 | 641 | 642 | 643 | 644 | 645 | 646 | 647 | 648 | 649 | 64A | 64B | 64C | 64D | 64E |
| +                                        | 650 | 651 | 652 | 653 | 654 | 655 | 656 | 657 | 658 | 659 | 65A | 65B | 65C | 65D | 65E |
| +                                        | 660 | 661 | 662 | 663 | 664 | 665 | 666 | 667 | 668 | 669 | 66A | 66B | 66C | 66D | 66E |
| +                                        | 670 | 671 | 672 | 673 | 674 | 675 | 676 | 677 | 678 | 679 | 67A | 67B | 67C | 67D | 67E |
| +                                        | 680 | 681 | 682 | 683 | 684 | 685 | 686 | 687 | 688 | 689 | 68A | 68B | 68C | 68D | 68E |
| +                                        | 690 | 691 | 692 | 693 | 694 | 695 | 696 | 697 | 698 | 699 | 69A | 69B | 69C | 69D | 69E |
| +                                        | 700 | 701 | 702 | 703 | 704 | 705 | 706 | 707 | 708 | 709 | 70A | 70B | 70C | 70D | 70E |
| +                                        | 710 | 711 | 712 | 713 | 714 | 715 | 716 | 717 | 718 | 719 | 71A | 71B | 71C | 71D | 71E |
| +                                        | 720 | 721 | 722 | 723 | 724 | 725 | 726 | 727 | 728 | 729 | 72A | 72B | 72C | 72D | 72E |
| +                                        | 730 | 731 | 732 | 733 | 734 | 735 | 736 | 737 | 738 | 739 | 73A | 73B | 73C | 73D | 73E |
| +                                        | 740 | 741 | 742 | 743 | 744 | 745 | 746 | 747 | 748 | 749 | 74A | 74B | 74C | 74D | 74E |
| +                                        | 750 | 751 | 752 | 753 | 754 | 755 | 756 | 757 | 758 | 759 | 75A | 75B | 75C | 75D | 75E |
| +                                        | 760 | 761 | 762 | 763 | 764 | 765 | 766 | 767 | 768 | 769 | 76A | 76B | 76C | 76D | 76E |
| +                                        | 770 | 771 | 772 | 773 | 774 | 775 | 776 | 777 | 778 | 779 | 77A | 77B | 77C | 77D | 77E |
| +                                        | 780 | 781 | 782 | 783 | 784 | 785 | 786 | 787 | 788 | 789 | 78A | 78B | 78C | 78D | 78E |
| +                                        | 790 | 791 | 792 | 793 | 794 | 795 | 796 | 797 | 798 | 799 | 79A | 79B | 79C | 79D | 79E |
| +                                        | 800 | 801 | 802 | 803 | 804 | 805 | 806 | 807 | 808 | 809 | 80A | 80B | 80C | 80D | 80E |
| +                                        | 810 | 811 | 812 | 813 | 814 | 815 | 816 | 817 | 818 | 819 | 81A | 81B | 81C | 81D | 81E |
| +                                        | 820 | 821 | 822 | 823 | 824 | 825 | 826 | 827 | 828 | 829 | 82A | 82B | 82C | 82D | 82E |
| +                                        | 830 | 831 | 832 | 833 | 834 | 835 | 836 | 837 | 838 | 839 | 83A | 83B | 83C | 83D | 83E |
| +                                        | 840 | 841 | 842 | 843 | 844 | 845 | 846 | 847 | 848 | 849 | 84A | 84B | 84C | 84D | 84E |
| +                                        | 850 | 851 | 852 | 853 | 854 | 855 | 856 | 857 | 858 | 859 | 85A | 85B | 85C | 85D | 85E |
| +                                        | 860 | 861 | 862 | 863 | 864 | 865 | 866 | 867 | 868 | 869 | 86A | 86B | 86C | 86D | 86E |
| +                                        | 870 | 871 | 872 | 873 | 874 | 875 | 876 | 877 | 878 | 879 | 87A | 87B | 87C | 87D | 87E |
| +                                        | 880 | 881 | 882 | 883 | 884 | 885 | 886 | 887 | 888 | 889 | 88A | 88B | 88C | 88D | 88E |
| +                                        | 890 | 891 | 8   |     |     |     |     |     |     |     |     |     |     |     |     |



| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| N                                        | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| -                                        | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 |
| +                                        | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |
| N                                        | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| -                                        | 70 | 6F | 6E | 6D | 6C | 6B | 6A | 69 | 68 | 67 | 66 | 65 | 64 | 63 | 62 |
| +                                        | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E |
| N                                        | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 |
| -                                        | 60 | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 |
| +                                        | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E |
| N                                        | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 |
| -                                        | 50 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 |

INDIRECT RELATIVE ADDRESS: Add H'60' to DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE DESCLK START ADDR \_\_\_\_\_

DESCRIPTION \_\_\_\_\_

ROUTINE SHEET 3 OF 9

MEMORY LOCATIONS THIS SHEET \_\_\_\_\_

| ADDRS | DATA<br>B0   B1   B2 | LABEL   | SYMBOLIC INSTRUCTION<br>OPCODE   OPERANDS | COMMENT                       |
|-------|----------------------|---------|-------------------------------------------|-------------------------------|
| 1     |                      |         |                                           |                               |
| 2     | 0067                 | INCMIN  | EORZ RO                                   | Comments same as              |
| 3     | 8                    |         | STRA,RO MSSEC                             | for INCSEC (pre-              |
| 4     | 8                    |         | LODR,RO LSMIN                             | vious page.) Following        |
| 5     | 6E                   |         | ADDI,RO 1                                 | comment describes var-        |
| 6     | 70                   |         | COMI,RO 10                                | iables to be analyzed         |
| 7     | 2                    |         | BCTR,EQ \$+6                              | in connection with HOURS      |
| 8     | 4                    |         | STRA,RO LSMIN                             | increment and reset; also     |
| 9     | 7                    |         | RETE,UN                                   | AM/PM indication changes      |
| 10    | 8                    |         | EORZ RO                                   | (1) AM/PM indicator on        |
| 11    | 9                    |         | STRA,RO LSMIN                             | from 12 noon to 11:59:59 PM   |
| 12    | C                    |         | LODR,RO MSMIN                             | off for 12 hrs. (2) LS hours  |
| 13    | 7F                   |         | ADDI,RO 1                                 | must increment to 9 (if       |
| 14    | E1                   |         | COMI,RO 6                                 | MSHRS (a 2) is off. Otherwise |
| 15    | 3                    |         | BCTR,EQ TESTHRS                           | LSHRS increment to 2. (3)     |
| 16    | 5                    |         | STRA,RO MSMIN                             | Reset hours from 12 to 1      |
| 17    | 8                    |         | RETE,UN                                   | at 1 AM, 1 PM (Carry from     |
| 18    |                      |         |                                           | 10:59:59 to 11:00 is an       |
| 19    | 89                   | TESTHRS | EORZ RO                                   | odd case] First, zero         |
| 20    | A                    |         | STRA,RO MSMIN                             | MSMIN in prep to change       |
| 21    | 7                    |         | LODR,RO MSHRS                             | hours. Now fetch all hours    |
| 22    | 90                   |         | LODR,RI LSHRS                             | Then see if its 10:00 or      |
| 23    | 3                    |         | TMI,RO H'17'                              | later yet. No! It's early     |
| 24    | 5                    |         | BCTR,EQ INCHRS                            | so go increment the LS        |
| 25    |                      |         |                                           | hours. Next routine (H1-      |
| 26    |                      |         |                                           | HRS) handles from 10:00 on    |
| 27    |                      |         |                                           |                               |

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| -                                        | 0F | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D |
| +                                        | 1E | 1F | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C |
| -                                        | 2D | 2E | 2F | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B |
| +                                        | 3C | 3D | 3E | 3F | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 4A |
| -                                        | 4B | 4C | 4D | 4E | 4F | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| +                                        | 5A | 5B | 5C | 5D | 5E | 5F | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 |
| -                                        | 69 | 6A | 6B | 6C | 6D | 6E | 6F | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 |
| +                                        | 78 | 79 | 7A | 7B | 7C | 7D | 7E | 7F | 80 | 81 | 82 | 83 | 84 | 85 | 86 |
| -                                        | 87 | 88 | 89 | 8A | 8B | 8C | 8D | 8E | 8F | 90 | 91 | 92 | 93 | 94 | 95 |
| +                                        | 96 | 97 | 98 | 99 | 9A | 9B | 9C | 9D | 9E | 9F | A0 | A1 | A2 | A3 | A4 |
| -                                        | A5 | A6 | A7 | A8 | A9 | AA | AB | AC | AD | AE | AF | B0 | B1 | B2 | B3 |
| +                                        | B4 | B5 | B6 | B7 | B8 | B9 | BA | BB | BC | BD | BE | BF | C0 | C1 | C2 |
| -                                        | C3 | C4 | C5 | C6 | C7 | C8 | C9 | CA | CB | CC | CD | CE | CF | D0 | D1 |
| +                                        | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | DA | DB | DC | DD | DE | DF | 00 |
| -                                        | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |

INDIRECT RELATIVE ADDRESS: Add H'20' TO DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE DESCLK START ADDR \_\_\_\_\_

DESCRIPTION \_\_\_\_\_

ROUTINE SHEET 4 OF 9

MEMORY LOCATIONS THIS SHEET \_\_\_\_\_

| ADDRES | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION |          | COMMENT                             |
|--------|------|----|----|--------|----------------------|----------|-------------------------------------|
|        | B0   | B1 | B2 |        | OPCODE               | OPERANDS |                                     |
| 1      |      |    |    |        |                      |          |                                     |
| 2      | 0097 |    |    | HIHRS  | ADDI, R1             | 1        | Now, increment LSHRS.               |
| 3      | 9    |    |    |        | COMI, R1             | 2        | Is it 12:00 yet? Yes                |
| 4      | 8    |    |    |        | BSTR, EQ             | AMPM     | go switch the AM/PM ind.            |
| 5      | D    |    |    |        | BCTR, UN             | TENELEV  | No! worth the 10:00 going to        |
| 6      |      |    |    |        |                      |          | 11:00, so service that unique case. |
| 7      | 00A0 |    |    | INCHRS | ADDI, R1             | 1        | This routine increments             |
| 8      | 2    |    |    |        | COMI, R1             | 10       | the hours when accessed             |
| 9      | 4    |    |    |        | BCTR, EQ             | \$+6     | between 1:00 and 10:00              |
| 10     | 1    |    |    |        | STRA, R1             | LSHRS    | same comment as for                 |
| 11     | 9    |    |    |        | RETE, UN             |          | incrementing seconds.               |
| 12     | A    |    |    |        | LODI, R1             | 0        |                                     |
| 13     | C    |    |    |        | STRA, R1             | LSHRS    |                                     |
| 14     | AF   |    |    |        | LODI, R0             | 1        |                                     |
| 15     | B1   |    |    |        | STRA, R0             | MSHRS    |                                     |
| 16     | 4    |    |    |        | RETE, UN             |          |                                     |
| 17     | 5    |    |    |        | NOP (3)              |          |                                     |
| 18     |      |    |    |        |                      |          | On entry, it's 12:00, so            |
| 19     | B8   | 01 | 05 | AMPM   | LODA, R2             | PMIND    | get the AM/PM indicator and         |
| 20     | 8    | 95 |    |        | COMI, R2             | H'95'    | see if it was PM? Yes!              |
| 21     | D    |    |    |        | BCTR, EQ             | SETAM    | go set AM (space) ind.              |
| 22     | BF   |    |    |        | LODI, R2             | H'95'    | No it wasn't. Set the PM            |
| 23     | C1   |    |    | STORIT | STRA, R2             | PMIND    | indicator and return to             |
| 24     | 4    |    |    |        | RETC, UN             |          | HIHRS.                              |
| 25     | 5    | 17 |    | SETAM  | LODI, R2             | SPACE    | get a space code and                |
| 26     | 7    |    |    |        | BCTR, UN             | STORIT   | put in message as an AM             |
| 27     |      |    |    |        |                      |          | indicator.                          |

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| +                                        | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| -                                        | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 |
| +                                        | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| +                                        | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 |
| -                                        | 60 | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 |
| +                                        | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 |
| +                                        | 45 | 46 | 47 | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 |
| -                                        | 50 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 |

INDIRECT RELATIVE ADDRESS: Add H'80' TO DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE DESCLK START ADDR \_\_\_\_\_

DESCRIPTION \_\_\_\_\_

ROUTINE SHEET 5 OF 9

MEMORY LOCATIONS THIS SHEET \_\_\_\_\_

| ADDRS | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |          | COMMENT                      |
|-------|------|----|----|---------|----------------------|----------|------------------------------|
|       | B3   | B1 | B2 |         | OPCODE               | OPERANDS |                              |
| 1     |      |    |    |         |                      |          |                              |
| 2     | 00CE |    |    | END     | LODI, R0             | H'17'    | On entry, initialize a       |
| 3     | DO   |    |    |         | STRA, R0             | MSHRS    | space for MSHRS and          |
| 4     | 3    |    |    |         | LODI, R0             | 1        | a 1 for LSHRS. Its           |
| 5     | 5    |    |    |         | STRA, R0             | LSHRS    | 1:00::00 in next             |
| 6     | 8    |    |    |         | RETE, UN             |          | display, so show it.         |
| 7     |      |    |    |         |                      |          |                              |
| 8     | E0   |    |    | TENELEV | STRA, R1             | LSHRS    | On entry, store previously   |
| 9     | 3    |    |    |         | TMI, R1              | 3        | incr. LSHRS. Is it 12:00     |
| 10    | 5    |    |    |         | RETE, LT             |          | going on to 1:00?            |
| 11    | 6    |    |    |         | BCTA, UN             | END      | NO. return, it was 12:00     |
| 12    |      |    |    |         |                      |          | 11:59, not 12:59. Yes. Its   |
| 13    | 0100 | XX | XX | DISCLK  | RES                  | 8        | 1:00, so reset hours to 1:00 |
| 14    | 3    | XX | XX |         |                      |          | Message table to dis-        |
| 15    | 6    | XX | XX |         |                      |          | play current time            |
| 16    | 0109 | XX |    |         |                      |          | HRS - MIN - AM - PM - SEC    |
| 17    | 0110 |    |    | FREQCT  | RES                  | 1        |                              |
| 18    | 013D |    |    | FREQZ   |                      |          | TEMP STR FOR FREQ. CT        |
| 19    | 0140 |    |    | AORPM   | BSTA, UN             | ORDER    | INDIRECT ADDRESS to FREQCT   |
| 20    | 3    |    |    |         | LODI, R1             | LAMPMS-1 | First, tell user to set clk. |
| 21    | 5    |    |    |         | LODI, R2             | LAMPMS-1 | On return, load and dis-     |
| 22    | 7    | FE |    |         | ZBSR                 | * MOV    | play a prompt for him        |
| 23    | 9    |    |    |         | NOP                  |          | to set AM or PM indicator    |
| 24    | A    |    |    |         | LODI, R0             | 5        | He'll depress 'A' for AM     |
| 25    | C    | FC |    |         | ZBSR                 | * GNPA   | and any other hex key for    |
| 26    | E    |    |    |         | NOP                  |          | PM. After he makes up        |
| 27    |      |    |    |         |                      |          | his mind,                    |

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| R                                        | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 |
| -                                        | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 |
| +                                        | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |
| N                                        | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| -                                        | 70 | 6F | 6E | 6D | 6C | 6B | 6A | 69 | 68 | 67 | 66 | 65 | 64 | 63 | 62 |
| +                                        | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E |
| N                                        | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 |
| -                                        | 60 | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 |
| +                                        | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E |
| N                                        | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 |
| -                                        | 50 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 |

INDIRECT RELATIVE ADDRESS: Add H'80' to DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE DESCLK START ADDR       

DESCRIPTION NOTE: If you would rather have the PM indicator located between HOURS and minutes, write '02' in locs; '15B', 'BA', 'C3'

ROUTINE SHEET 6 OF 9MEMORY LOCATIONS THIS SHEET       

|    | ADDRS | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |          | COMMENT                       |
|----|-------|------|----|----|---------|----------------------|----------|-------------------------------|
|    |       | B0   | B1 | B2 |         | OPCODE               | OPERANDS |                               |
| 1  |       |      |    |    |         |                      |          |                               |
| 2  | 014F  |      |    |    |         | COMI, RO             | H'0A'    | see if its morning?           |
| 3  | 151   |      |    |    |         | BCTR, EQ             | MORNING  | Yes! get set AM indication    |
| 4  | 3     |      |    |    | EVENING | LODI, RO             | H'95'    | No! after noon. So get        |
| 5  | 5     |      |    |    |         | BCTR, UN             | \$+4     | the PM indicator (a '0') and  |
| 6  | 7     |      |    |    | MORNING | LODI, RO             | H'17'    | morning: get a space and      |
| 7  | 9     |      |    |    |         | STRA, RO             | PMIND    | store it in the clock message |
| 8  | C     |      |    |    |         | EORZ                 | RO       | Now initialize Freq Ct.       |
| 9  | D     | DE   |    |    |         | STRR, RO             | *FREQZ   | to zero, then branch          |
| 10 | 15F   |      |    |    |         | BCTR, UN             | SPCKLK   | to complete clock mess.       |
| 11 |       |      |    |    |         |                      |          |                               |
| 12 | 161   | 0A   | 17 | 95 | AMPMES  | RES                  | 8        | "A or P = ?                   |
| 13 | 13    | 17   | 10 |    |         |                      |          |                               |
| 14 | 7     | 16   | 17 |    |         |                      |          |                               |
| 15 |       |      |    |    |         |                      |          |                               |
| 16 | 170   |      |    |    | BLNKZ   | LODI, R2             | 3        | On entry, time ready to       |
| 17 | 2     | 01   | BB |    |         | COMA, R2             | EVENTS   | be put in clock message       |
| 18 | 5     |      |    |    |         | BCTR, EQ             | NOBLNK   | Now, is it hours? If not      |
| 19 | 7     |      |    |    |         | BRNR, RO             | NOBLNK   | exit! Yes! Is it between      |
| 20 | 9     | 17   |    |    |         | LODI, RO             | SPACE    | 1-9 pm? No! leave MS.         |
| 21 | 17B   |      |    |    | NOBLNK  | RETC, UN             |          | hour as is Yes! get a space   |
| 22 |       |      |    |    |         |                      |          | for MSHR and exit.            |
| 23 | 169   |      |    |    | SPCKLK  | LODI, RO             | H'17'    | To initialize clock mes-      |
| 24 | 8     |      |    |    |         | STRA, RO             | DISCLK+2 | sage, get a space in          |
| 25 | E     |      |    |    |         | RETC, UN             |          | its 3rd digit, then return    |
| 26 |       |      |    |    |         |                      |          |                               |
| 27 |       |      |    |    |         |                      |          |                               |

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |     |     |     |     |     |     |     |     |     |     |     |     |     |     |     |
|------------------------------------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| +                                        | 00  | 01  | 02  | 03  | 04  | 05  | 06  | 07  | 08  | 09  | 0A  | 0B  | 0C  | 0D  | 0E  |
| +                                        | 10  | 11  | 12  | 13  | 14  | 15  | 16  | 17  | 18  | 19  | 1A  | 1B  | 1C  | 1D  | 1E  |
| +                                        | 20  | 21  | 22  | 23  | 24  | 25  | 26  | 27  | 28  | 29  | 2A  | 2B  | 2C  | 2D  | 2E  |
| +                                        | 30  | 31  | 32  | 33  | 34  | 35  | 36  | 37  | 38  | 39  | 3A  | 3B  | 3C  | 3D  | 3E  |
| +                                        | 40  | 41  | 42  | 43  | 44  | 45  | 46  | 47  | 48  | 49  | 4A  | 4B  | 4C  | 4D  | 4E  |
| +                                        | 50  | 51  | 52  | 53  | 54  | 55  | 56  | 57  | 58  | 59  | 5A  | 5B  | 5C  | 5D  | 5E  |
| +                                        | 60  | 61  | 62  | 63  | 64  | 65  | 66  | 67  | 68  | 69  | 6A  | 6B  | 6C  | 6D  | 6E  |
| +                                        | 70  | 71  | 72  | 73  | 74  | 75  | 76  | 77  | 78  | 79  | 7A  | 7B  | 7C  | 7D  | 7E  |
| +                                        | 80  | 81  | 82  | 83  | 84  | 85  | 86  | 87  | 88  | 89  | 8A  | 8B  | 8C  | 8D  | 8E  |
| +                                        | 90  | 91  | 92  | 93  | 94  | 95  | 96  | 97  | 98  | 99  | 9A  | 9B  | 9C  | 9D  | 9E  |
| +                                        | A0  | A1  | A2  | A3  | A4  | A5  | A6  | A7  | A8  | A9  | AA  | AB  | AC  | AD  | AE  |
| +                                        | B0  | B1  | B2  | B3  | B4  | B5  | B6  | B7  | B8  | B9  | BA  | BB  | BC  | BD  | BE  |
| +                                        | C0  | C1  | C2  | C3  | C4  | C5  | C6  | C7  | C8  | C9  | CA  | CB  | CC  | CD  | CE  |
| +                                        | D0  | D1  | D2  | D3  | D4  | D5  | D6  | D7  | D8  | D9  | DA  | DB  | DC  | DD  | DE  |
| +                                        | E0  | E1  | E2  | E3  | E4  | E5  | E6  | E7  | E8  | E9  | EA  | EB  | EC  | ED  | EE  |
| +                                        | F0  | F1  | F2  | F3  | F4  | F5  | F6  | F7  | F8  | F9  | FA  | FB  | FC  | FD  | FE  |
| +                                        | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 10A | 10B | 10C | 10D | 10E |
| +                                        | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 11A | 11B | 11C | 11D | 11E |
| +                                        | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 | 12A | 12B | 12C | 12D | 12E |
| +                                        | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 13A | 13B | 13C | 13D | 13E |
| +                                        | 140 | 141 | 142 | 143 | 144 | 145 | 146 | 147 | 148 | 149 | 14A | 14B | 14C | 14D | 14E |
| +                                        | 150 | 151 | 152 | 153 | 154 | 155 | 156 | 157 | 158 | 159 | 15A | 15B | 15C | 15D | 15E |
| +                                        | 160 | 161 | 162 | 163 | 164 | 165 | 166 | 167 | 168 | 169 | 16A | 16B | 16C | 16D | 16E |
| +                                        | 170 | 171 | 172 | 173 | 174 | 175 | 176 | 177 | 178 | 179 | 17A | 17B | 17C | 17D | 17E |
| +                                        | 180 | 181 | 182 | 183 | 184 | 185 | 186 | 187 | 188 | 189 | 18A | 18B | 18C | 18D | 18E |
| +                                        | 190 | 191 | 192 | 193 | 194 | 195 | 196 | 197 | 198 | 199 | 19A | 19B | 19C | 19D | 19E |
| +                                        | 200 | 201 | 202 | 203 | 204 | 205 | 206 | 207 | 208 | 209 | 20A | 20B | 20C | 20D | 20E |
| +                                        | 210 | 211 | 212 | 213 | 214 | 215 | 216 | 217 | 218 | 219 | 21A | 21B | 21C | 21D | 21E |
| +                                        | 220 | 221 | 222 | 223 | 224 | 225 | 226 | 227 | 228 | 229 | 22A | 22B | 22C | 22D | 22E |
| +                                        | 230 | 231 | 232 | 233 | 234 | 235 | 236 | 237 | 238 | 239 | 23A | 23B | 23C | 23D | 23E |
| +                                        | 240 | 241 | 242 | 243 | 244 | 245 | 246 | 247 | 248 | 249 | 24A | 24B | 24C | 24D | 24E |
| +                                        | 250 | 251 | 252 | 253 | 254 | 255 | 256 | 257 | 258 | 259 | 25A | 25B | 25C | 25D | 25E |
| +                                        | 260 | 261 | 262 | 263 | 264 | 265 | 266 | 267 | 268 | 269 | 26A | 26B | 26C | 26D | 26E |
| +                                        | 270 | 271 | 272 | 273 | 274 | 275 | 276 | 277 | 278 | 279 | 27A | 27B | 27C | 27D | 27E |
| +                                        | 280 | 281 | 282 | 283 | 284 | 285 | 286 | 287 | 288 | 289 | 28A | 28B | 28C | 28D | 28E |
| +                                        | 290 | 291 | 292 | 293 | 294 | 295 | 296 | 297 | 298 | 299 | 29A | 29B | 29C | 29D | 29E |
| +                                        | 300 | 301 | 302 | 303 | 304 | 305 | 306 | 307 | 308 | 309 | 30A | 30B | 30C | 30D | 30E |
| +                                        | 310 | 311 | 312 | 313 | 314 | 315 | 316 | 317 | 318 | 319 | 31A | 31B | 31C | 31D | 31E |
| +                                        | 320 | 321 | 322 | 323 | 324 | 325 | 326 | 327 | 328 | 329 | 32A | 32B | 32C | 32D | 32E |
| +                                        | 330 | 331 | 332 | 333 | 334 | 335 | 336 | 337 | 338 | 339 | 33A | 33B | 33C | 33D | 33E |
| +                                        | 340 | 341 | 342 | 343 | 344 | 345 | 346 | 347 | 348 | 349 | 34A | 34B | 34C | 34D | 34E |
| +                                        | 350 | 351 | 352 | 353 | 354 | 355 | 356 | 357 | 358 | 359 | 35A | 35B | 35C | 35D | 35E |
| +                                        | 360 | 361 | 362 | 363 | 364 | 365 | 366 | 367 | 368 | 369 | 36A | 36B | 36C | 36D | 36E |
| +                                        | 370 | 371 | 372 | 373 | 374 | 375 | 376 | 377 | 378 | 379 | 37A | 37B | 37C | 37D | 37E |
| +                                        | 380 | 381 | 382 | 383 | 384 | 385 | 386 | 387 | 388 | 389 | 38A | 38B | 38C | 38D | 38E |
| +                                        | 390 | 391 | 392 | 393 | 394 | 395 | 396 | 397 | 398 | 399 | 39A | 39B | 39C | 39D | 39E |
| +                                        | 400 | 401 | 402 | 403 | 404 | 405 | 406 | 407 | 408 | 409 | 40A | 40B | 40C | 40D | 40E |
| +                                        | 410 | 411 | 412 | 413 | 414 | 415 | 416 | 417 | 418 | 419 | 41A | 41B | 41C | 41D | 41E |
| +                                        | 420 | 421 | 422 | 423 | 424 | 425 | 426 | 427 | 428 | 429 | 42A | 42B | 42C | 42D | 42E |
| +                                        | 430 | 431 | 432 | 433 | 434 | 435 | 436 | 437 | 438 | 439 | 43A | 43B | 43C | 43D | 43E |
| +                                        | 440 | 441 | 442 | 443 | 444 | 445 | 446 | 447 | 448 | 449 | 44A | 44B | 44C | 44D | 44E |
| +                                        | 450 | 451 | 452 | 453 | 454 | 455 | 456 | 457 | 458 | 459 | 45A | 45B | 45C | 45D | 45E |
| +                                        | 460 | 461 | 462 | 463 | 464 | 465 | 466 | 467 | 468 | 469 | 46A | 46B | 46C | 46D | 46E |
| +                                        | 470 | 471 | 472 | 473 | 474 | 475 | 476 | 477 | 478 | 479 | 47A | 47B | 47C | 47D | 47E |
| +                                        | 480 | 481 | 482 | 483 | 484 | 485 | 486 | 487 | 488 | 489 | 48A | 48B | 48C | 48D | 48E |
| +                                        | 490 | 491 | 492 | 493 | 494 | 495 | 496 | 497 | 498 | 499 | 49A | 49B | 49C | 49D | 49E |
| +                                        | 500 | 501 | 502 | 503 | 504 | 505 | 506 | 507 | 508 | 509 | 50A | 50B | 50C | 50D | 50E |
| +                                        | 510 | 511 | 512 | 513 | 514 | 515 | 516 | 517 | 518 | 519 | 51A | 51B | 51C | 51D | 51E |
| +                                        | 520 | 521 | 522 | 523 | 524 | 525 | 526 | 527 | 528 | 529 | 52A | 52B | 52C | 52D | 52E |
| +                                        | 530 | 531 | 532 | 533 | 534 | 535 | 536 | 537 | 538 | 539 | 53A | 53B | 53C | 53D | 53E |
| +                                        | 540 | 541 | 542 | 543 | 544 | 545 | 546 | 547 | 548 | 549 | 54A | 54B | 54C | 54D | 54E |
| +                                        | 550 | 551 | 552 | 553 | 554 | 555 | 556 | 557 | 558 | 559 | 55A | 55B | 55C | 55D | 55E |
| +                                        | 560 | 561 | 562 | 563 | 564 | 565 | 566 | 567 | 568 | 569 | 56A | 56B | 56C | 56D | 56E |
| +                                        | 570 | 571 | 572 | 573 | 574 | 575 | 576 | 577 | 578 | 579 | 57A | 57B | 57C | 57D | 57E |
| +                                        | 580 | 581 | 582 | 583 | 584 | 585 | 586 | 587 | 588 | 589 | 58A | 58B | 58C | 58D | 58E |
| +                                        | 590 | 591 | 592 | 593 | 594 | 595 | 596 | 597 | 598 | 599 | 59A | 59B | 59C | 59D | 59E |
| +                                        | 600 | 601 | 602 | 603 | 604 | 605 | 606 | 607 | 608 | 609 | 60A | 60B | 60C | 60D | 60E |
| +                                        | 610 | 611 | 612 | 613 | 614 | 615 | 616 | 617 | 618 | 619 | 61A | 61B | 61C | 61D | 61E |
| +                                        | 620 | 621 | 622 | 623 | 624 | 625 | 626 | 627 | 628 | 629 | 62A | 62B | 62C | 62D | 62E |
| +                                        | 630 | 631 | 632 | 633 | 634 | 635 | 636 | 637 | 638 | 639 | 63A | 63B | 63C | 63D | 63E |
| +                                        | 640 | 641 | 642 | 643 | 644 | 645 | 646 | 647 | 648 | 649 | 64A | 64B | 64C | 64D | 64E |
| +                                        | 650 | 651 | 652 | 653 | 654 | 655 | 656 | 657 | 658 | 659 | 65A | 65B | 65C | 65D | 65E |
| +                                        | 660 | 661 | 662 | 663 | 664 | 665 | 666 | 667 | 668 | 669 | 66A | 66B | 66C | 66D | 66E |
| +                                        | 670 | 671 | 672 | 673 | 674 | 675 | 676 | 677 | 678 | 679 | 67A | 67B | 67C | 67D | 67E |
| +                                        | 680 | 681 | 682 | 683 | 684 | 685 | 686 | 687 | 688 | 689 | 68A | 68B | 68C | 68D | 68E |
| +                                        | 690 | 691 | 692 | 693 | 694 | 695 | 696 | 697 | 698 | 699 | 69A | 69B | 69C | 69D | 69E |
| +                                        | 700 | 701 | 702 | 703 | 704 | 705 | 706 | 707 | 708 | 709 | 70A | 70B | 70C | 70D | 70E |
| +                                        | 710 | 711 | 712 | 713 | 714 | 715 | 716 | 717 | 718 | 719 | 71A | 71B | 71C | 71D | 71E |
| +                                        | 720 | 721 | 722 | 723 | 724 | 725 | 726 | 727 | 728 | 729 | 72A | 72B | 72C | 72D | 72E |
| +                                        | 730 | 731 | 732 | 733 | 734 | 735 | 736 | 737 | 738 | 739 | 73A | 73B | 73C | 73D | 73E |
| +                                        | 740 | 741 | 742 | 743 | 744 | 745 | 746 | 747 | 748 | 749 | 74A | 74B | 74C | 74D | 74E |
| +                                        | 750 | 751 | 752 | 753 | 754 | 755 | 756 | 757 | 758 | 759 | 75A | 75B | 75C | 75D | 75E |
| +                                        | 760 | 761 | 762 | 763 | 764 | 765 | 766 | 767 | 768 | 769 | 76A | 76B | 76C | 76D | 76E |
| +                                        | 770 | 771 | 772 | 773 | 774 | 775 | 776 | 777 | 778 | 779 | 77A | 77B | 77C | 77D | 77E |
| +                                        | 780 | 781 | 782 | 783 | 784 | 785 | 786 | 787 | 788 | 789 | 78A | 78B | 78C | 78D | 78E |
| +                                        | 790 | 791 | 792 | 793 | 794 | 795 | 796 | 797 | 798 | 799 | 79A | 79B | 79C | 79D | 79E |
| +                                        | 800 | 801 | 802 | 803 | 804 | 805 | 806 | 807 | 808 | 809 | 80A | 80B | 80C | 80D | 80E |
| +                                        | 810 | 811 | 812 | 813 | 814 | 815 | 816 | 817 | 818 | 819 | 81A | 81B | 81C | 81D | 81E |
| +                                        | 820 | 821 | 822 | 823 | 824 | 825 | 826 | 827 | 828 | 829 | 82A | 82B | 82C | 82D | 82E |
| +                                        | 830 | 831 | 832 | 833 | 834 | 835 | 836 | 837 | 838 | 839 | 83A | 83B | 83C | 83D | 83E |
| +                                        | 840 | 841 | 842 | 843 | 844 | 845 | 846 | 847 | 848 | 849 | 84A | 84B | 84C | 84D | 84E |
| +                                        | 850 | 851 | 852 | 853 | 854 | 855 | 856 | 857 | 858 | 859 | 85A | 85B | 85C | 85D | 85E |
| +                                        | 860 | 861 | 862 | 863 | 864 | 865 | 866 | 867 | 868 | 869 | 86A | 86B | 86C | 86D | 86E |
| +                                        | 870 | 871 | 872 | 873 | 874 | 875 | 876 | 877 | 878 | 879 | 87A | 87B | 87C | 87D | 87E |
| +                                        | 880 | 881 | 882 | 883 | 884 | 885 | 886 | 887 | 888 | 889 | 88A | 88B | 88C | 88D | 88E |
| +                                        | 890 | 891 | 8   |     |     |     |     |     |     |     |     |     |     |     |     |

| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 03 | 04 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E |
| H                                        | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 |
| -                                        | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 |
| +                                        | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E |
| H                                        | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 |
| -                                        | 7C | 7B | 7A | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 | 70 | 6F | 6E |
| +                                        | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E |
| H                                        | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E | 3F | 40 |
| -                                        | 60 | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 |
| +                                        | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E |
| H                                        | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 |
| -                                        | 50 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 |

INDIRECT RELATIVE ADDRESS: ADD H'80' TO DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE DESCLK START ADDR \_\_\_\_\_

DESCRIPTION \_\_\_\_\_

ROUTINE SHEET 8 OF 9

MEMORY LOCATIONS THIS SHEET \_\_\_\_\_

|    | ADDRS | DATA |    |    | LABEL   | SYMBOLIC INSTRUCTION |           | COMMENT                         |
|----|-------|------|----|----|---------|----------------------|-----------|---------------------------------|
|    |       | B2   | B1 | B2 |         | OPCODE               | OPERANDS  |                                 |
| 1  |       |      |    |    |         |                      |           |                                 |
| 2  | 01AF  |      |    |    |         | STRR, R3             | MESDX     | store it til needed. Get        |
| 3  | 1B1   |      |    |    |         | LODR, R1             | EVENTS    | events! Have HRS, MIN, and      |
| 4  | 3     |      |    |    |         | BDRR, R1             | NEXT      | SECS been keyed? No! fi-        |
| 5  | 5     |      |    |    |         | RETC, UN             |           | nish. Yes! Exit.                |
| 6  |       |      |    |    |         |                      |           |                                 |
| 7  | 1BS   | XX   |    |    | EVENTS  | RES                  | 1         | Entry of HRS, MIN, SECS (3 max) |
| 8  | C     | XX   |    |    | >SELMX  | RES                  | 1         | HOURS, MIN, SECS enter next ind |
| 9  | D     | XX   |    |    | MESDX   | RES                  | 1         | Display CLK index               |
| 10 | E     | XX   |    |    | RUNDSPX | RES                  | 1         | Order run time constant         |
| 11 | F     | XX   |    |    | DSPDLY  | RES                  | 1         | Order display delay.            |
| 12 |       |      |    |    |         |                      |           |                                 |
| 13 | 1CO   |      |    |    | ORDER   | LODI, R1             | 3         | On entry set up constants       |
| 14 | 2     |      |    |    |         | LODI, R3             | H'7F'     | for 2 sec display of the        |
| 15 | 4     |      |    |    | REPEAT  | STRR, R1             | RUNDSPX   | message "SET CLK"               |
| 16 | 6     |      |    |    | DSPAGN  | STRR, R3             | DSPDLY    | store the constants and         |
| 17 | 8     |      |    |    |         | LODI, R1             | <SETCLK-1 | initialize "SET CLK" mes.       |
| 18 | A     |      |    |    |         | LODI, R2             | >SETCLK-1 | page index and display          |
| 19 | C     | E6   |    |    |         | ZBSR                 | *USRDSP   | it. On return, call             |
| 20 | CE    |      |    |    |         | LODR, R3             | DSPDLY    | display delay complete.         |
| 21 | DO    |      |    |    |         | COMI, R3             | 1         | If it is, reinitialize          |
| 22 | 2     |      |    |    |         | BCTR, EQ             | CONTDSP   | If not, go display again.       |
| 23 | 4     |      |    |    |         | BDRR, R3             | DSPAGN    |                                 |
| 24 | 6     |      |    |    | CONTDSP | LODR, R1             | RUNDSPX   | Is display run time over        |
| 25 | 8     |      |    |    |         | BDRR, R1             | \$+3      | yet? No! go repeat it.          |
| 26 | 1DA   |      |    |    |         | RETC, UN             |           | Yes! go set time.               |
| 27 |       |      |    |    |         |                      |           |                                 |



| DIRECT RELATIVE ADDRESSING - SECOND BYTE |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|------------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| +                                        | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F |
| H                                        | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 |
| -                                        | 7F | 7E | 7D | 7C | 7B | 7A | 79 | 78 | 77 | 76 | 75 | 74 | 73 | 72 | 71 |    |
| +                                        | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 1A | 1B | 1C | 1D | 1E | 1F |
| H                                        | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| -                                        | 70 | 6F | 6E | 6D | 6C | 6B | 6A | 69 | 68 | 67 | 66 | 65 | 64 | 63 | 62 | 61 |
| +                                        | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 2A | 2B | 2C | 2D | 2E | 2F |
| H                                        | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 |
| -                                        | 60 | 5F | 5E | 5D | 5C | 5B | 5A | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 |
| +                                        | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 3A | 3B | 3C | 3D | 3E | 3F |
| H                                        | 48 | 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 |
| -                                        | 50 | 4F | 4E | 4D | 4C | 4B | 4A | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 |

INDIRECT RELATIVE ADDRESS: Add H'60' TO DISPLACEMENT

## 2650 PROGRAMMING FORM

ROUTINE DESCLK START ADDR \_\_\_\_\_

DESCRIPTION \_\_\_\_\_

ROUTINE SHEET 9 OF 9

MEMORY LOCATIONS THIS SHEET \_\_\_\_\_

|    | ADDRS | DATA |    |    | LABEL  | SYMBOLIC INSTRUCTION                |          | COMMENT                             |
|----|-------|------|----|----|--------|-------------------------------------|----------|-------------------------------------|
|    |       | B0   | B1 | B2 |        | OPCODE                              | OPERANDS |                                     |
| 1  |       |      |    |    |        |                                     |          |                                     |
| 2  | 01DB  |      |    |    |        | LODI, R3                            | 7F       | To repeat, reinitialize             |
| 3  | IDD   |      |    |    |        | BCTR, UN                            | REPEAT   | display delay and loop.             |
| 4  |       |      |    |    |        |                                     |          |                                     |
| 5  | IED   | 05   | 0E | 07 | SELMES | RES                                 | 32       | SET CLOC                            |
| 6  | 3     | 17   | 0C | 11 |        |                                     |          | Access from "ORDER" only            |
| 7  | 6     | 00   | 0C |    |        |                                     |          |                                     |
| 8  | IEB   | 14   | 00 | 12 |        |                                     |          | HOURS } EVENT 3 (1 <sup>st</sup> )  |
| 9  | B     | 13   | 05 | 97 |        |                                     |          | access from                         |
| 10 | E     | 17   | 17 |    |        |                                     |          | "NOWSET" only                       |
| 11 | IF0   | 0F   | 13 | 0A |        |                                     |          | FRACT. } EVENT 2 (2 <sup>nd</sup> ) |
| 12 | 3     | 0C   | 07 | 97 |        |                                     |          |                                     |
| 13 | 6     | 17   | 17 |    |        |                                     |          |                                     |
| 14 | IFB   | 05   | 0E | 0C |        |                                     |          | SECS } EVENT 1 (3 <sup>rd</sup> )   |
| 15 | B     | 05   | 16 | 97 |        |                                     |          |                                     |
| 16 | IFE   | 17   | 17 |    |        |                                     |          |                                     |
| 17 |       |      |    |    | * THIS | CLOCK BEATS ANY HAND-HELD WATCH !!! |          |                                     |
| 18 |       |      |    |    |        |                                     |          |                                     |
| 19 |       |      |    |    |        |                                     |          |                                     |
| 20 |       |      |    |    |        |                                     |          |                                     |
| 21 |       |      |    |    |        |                                     |          |                                     |
| 22 |       |      |    |    |        |                                     |          |                                     |
| 23 |       |      |    |    |        |                                     |          |                                     |
| 24 |       |      |    |    |        |                                     |          |                                     |
| 25 |       |      |    |    |        |                                     |          |                                     |
| 26 |       |      |    |    |        |                                     |          |                                     |
| 27 |       |      |    |    |        |                                     |          |                                     |

## SUGGESTED CODE - PROGRAM "DESCLK"

## NOTES:

PAGE 51

| ADDRS | DATA       |
|-------|------------|
| B7    | B1 B2      |
| 1     | 0000 76 20 |
| 2     | 4 1B 07    |
| 3     | 4 01 40    |
| 4     | 6 XX       |
| 5     | 7 1B 30    |
| 6     | 4 01 90    |
| 7     | 8 74 40    |
| 8     | 6 BB 84    |
| 9     | 0F BB 89   |
| 10    | 11 75 FE   |
| 11    | 3 05 00    |
| 12    | 5 06 FF    |
| 13    | 7 07 01    |
| 14    | 4 BB E6    |
| 15    | 8 84 40    |
| 16    | 9 1A 07    |
| 17    | 1F 74 20   |
| 18    | 21 C0      |
| 19    | 2 76 20    |
| 20    | 4 1B 6D    |
| 21    | 6 34 80    |
| 22    | 8 98 69    |
| 23    | A 76 40    |
| 24    | 2C 1B 65   |
| 25    |            |
| 26    |            |
| 27    |            |

PAGE 52

| ADDRS | DATA          |
|-------|---------------|
| B7    | B1 B2         |
| 1     |               |
| 2     | 0039 0D 01 10 |
| 3     | 4 85 01       |
| 4     | 3E E5 3C      |
| 5     | 40 1B 03      |
| 6     | 2 C9 F6       |
| 7     | 44 17         |
| 8     |               |
| 9     | 45 20         |
| 10    | 6 CC 01 10    |
| 11    | 4 0C 01 07    |
| 12    | C B4 01       |
| 13    | 4E E4 0A      |
| 14    | 60 1B 04      |
| 15    | 2 CC 01 07    |
| 16    | 5 37          |
| 17    | 6 20          |
| 18    | 7 CC 01 07    |
| 19    | A 0C 01 06    |
| 20    | 8 B4 01       |
| 21    | 5F E4 06      |
| 22    | 61 1B 04      |
| 23    | 3 CC 01 06    |
| 24    | 6 37          |
| 25    |               |
| 26    |               |
| 27    |               |

PAGE 53

| ADDRS | DATA        |
|-------|-------------|
| B7    | B1 B2       |
| 1     |             |
| 2     | 0067 20     |
| 3     | 8 CC 01 06  |
| 4     | 9 0C 01 04  |
| 5     | 6E 84 01    |
| 6     | 70 E4 0A    |
| 7     | 2 1B 04     |
| 8     | 4 CC 01 04  |
| 9     | 7 37        |
| 10    | 8 20        |
| 11    | 4 CC 01 04  |
| 12    | C 0C 01 03  |
| 13    | 7F B4 01    |
| 14    | B1 E4 06    |
| 15    | 3 1B 04     |
| 16    | 5 CC 01 03  |
| 17    | 8 37        |
| 18    |             |
| 19    | 89 20       |
| 20    | A CC 01 03  |
| 21    | 9 0C 01 00  |
| 22    | 90 0D 01 01 |
| 23    | 3 F4 17     |
| 24    | F 1B 09     |
| 25    |             |
| 26    |             |
| 27    |             |

PAGE 54

| ADDRS | DATA        |
|-------|-------------|
| B7    | B1 B2       |
| 1     |             |
| 2     | 0097 B5 01  |
| 3     | 4 E5 02     |
| 4     | 8 3B 1B     |
| 5     | D 1F 00 E0  |
| 6     |             |
| 7     | 00A0 85 01  |
| 8     | 2 E5 0A     |
| 9     | 4 1B 04     |
| 10    | 1 CD 01 01  |
| 11    | 4 37        |
| 12    | A 05 00     |
| 13    | C CD 01 01  |
| 14    | AF 04 01    |
| 15    | B1 CC 01 00 |
| 16    | 4 37        |
| 17    | 5 C0 C0 C0  |
| 18    |             |
| 19    | 28 0E 01 05 |
| 20    | 3 E6 95     |
| 21    | 7 1B 06     |
| 22    | 34 06 95    |
| 23    | C1 CE 01 05 |
| 24    | 4 17        |
| 25    | 5 06 17     |
| 26    | 7 1B 78     |
| 27    |             |

PAGE 55

| ADDRS | DATA          |
|-------|---------------|
| B7    | B1 B2         |
| 1     |               |
| 2     | 00CE C4 17    |
| 3     | 70 CC 01 00   |
| 4     | 3 04 01       |
| 5     | 5 CC 01 01    |
| 6     | 8 37          |
| 7     |               |
| 8     | E0 CD 01 01   |
| 9     | 3 F5 03       |
| 10    | 5 36          |
| 11    | 6 1F 00 CE    |
| 12    |               |
| 13    | 0100 XX XX 17 |
| 14    | 3 XX XX XX    |
| 15    | 4 XX XX       |
| 16    |               |
| 17    | 0110 XX       |
| 18    | D13D 01 10    |
| 19    | 0140 3F 01 C0 |
| 20    | 3 05 01       |
| 21    | 5 06 60       |
| 22    | 7 BB FE       |
| 23    | 4 C0          |
| 24    | A 04 05       |
| 25    | C BB FC       |
| 26    | E C0          |
| 27    |               |

PAGE 57

| ADDRS | DATA         |
|-------|--------------|
| B7    | B1 B2        |
| 1     |              |
| 2     | 014F E4 0A   |
| 3     | 151 1B 04    |
| 4     | 3 04 95      |
| 5     | 5 1B 02      |
| 6     | 7 04 17      |
| 7     | 4 CC 01 05   |
| 8     | C 20         |
| 9     | 0 CB DE      |
| 10    |              |
| 11    | 15F 1B 0B    |
| 12    |              |
| 13    | 161 0A 17 95 |
| 14    | 4 13 17 10   |
| 15    | 7 16 17      |
| 16    |              |
| 17    | 170 06 03    |
| 18    | 2 EE 01 BB   |
| 19    | 5 98 04      |
| 20    | 7 5B 02      |
| 21    | 4 04 17      |
| 22    | 17B 17       |
| 23    | 169 04 17    |
| 24    | 3 CC 01 02   |
| 25    | 2 17         |
| 26    |              |
| 27    |              |

PAGE 58

| ADDRS | DATA        |
|-------|-------------|
| B7    | B1 B2       |
| 1     |             |
| 2     | 01B0 05 01  |
| 3     | 2 BB FE     |
| 4     | 4 C0        |
| 5     | F 04 01     |
| 6     | 7 BB FC     |
| 7     | 4 C0        |
| 8     | A BB F4     |
| 9     | C 3B 62     |
| 10    | 18E 17      |
| 11    |             |
| 12    | 190 20      |
| 13    | 4 CB 2A     |
| 14    | 3 06 DE     |
| 15    | 5 CA 25     |
| 16    | 7 05 03     |
| 17    | 4 CA 20     |
| 18    | 6 CA 1F     |
| 19    | 0 B6 C8     |
| 20    | 4F CA 1B    |
| 21    | A1 3F 01 B0 |
| 22    | 4 0B 17     |
| 23    | 6 CF 61 00  |
| 24    | 4 01        |
| 25    | A CF 21 00  |
| 26    | 1A0 B7 02   |
| 27    |             |

PAGE 59

| ADDRS | DATA       |
|-------|------------|
| B7    | B1 B2      |
| 1     |            |
| 2     | 01AF CB 0C |
| 3     | 1B1 09 08  |
| 4     | 3 F9 64    |
| 5     | 5 17       |
| 6     |            |
| 7     | 1B5 XX     |
| 8     | C XX       |
| 9     | D XX       |
| 10    | E XX       |
| 11    | F XX       |
| 12    |            |
| 13    | 1C0 05 03  |
| 14    | 4 07 7F    |
| 15    | 4 CA 7B    |
| 16    | 1 CB 77    |
| 17    | 4 05 01    |
| 18    | A 06 DE    |
| 19    | C BB E6    |
| 20    | C 06 6F    |
| 21    | D0E7 01    |
| 22    | 2 1B 02    |
| 23    | 4 FB 70    |
| 24    | 6 09 66    |
| 25    | 3 F9 01    |
| 26    | 1BA 17     |
| 27    | 5 07 7F    |

- Q. Sequence: locations '13', '15', '17', '19' (and execute 'USRDSP'), '1B', '1D', '26', '28', and loop to '13'.
- R. Sequence: locations '13', '15', '17', '19' (and execute 'USRDSP'), '1B', '1D', '26', '28', '2A', '2C', '13' through '1D', '1F', '07', '39' (to execute CLK), return and execute '21', '22', '24', and loop to '13'.

## SAVE/RESTORE REGISTER OPERATIONS ON ENTRY TO EXIT FROM INTERRUPT SERVICE ROUTINE

### INTRODUCTION:

As you have noticed previously, the microprocessor can be programmed to execute some rather complex algorithms. Intermediate computations are stored in registers. Many of the branch instructions depend on condition code status as a basis for transfer of program sequence control. If an UNSCHEDULED interrupt (defined on page 44 this section) is received while complex algorithm processing is taking place, AND if a return to that process is anticipated after executing the interrupt service routine, the contents of the working registers and PSW must be saved upon entry to the interrupt service routine.

The "INSTRUCTOR's" monitor program contains 2 user-accessable sub-routines which will save (and restore) the contents of the currently selected bank (prime or non-prime) of general purpose registers. Use BSTR,UN XX instructions where XX = SAVR0 ..... or ..... RESTRO. Absolute addressing is required.

Start Address: '1EA9'

|             |                         |
|-------------|-------------------------|
| 1EA9 0D17DA | SAVR0 STRA,R1 SAVREG+1  |
| 1EAC 0E17DB | SAVR01 STRA,R2 SAVREG+2 |
| 1EAF 0F17DC | SAVR02 STRA,R3 SAVREG+3 |
| 1EB2 17     | RETC,UN                 |

Saves contents of current selected G.P. register bank in monitor RAM (loc '17DA' to '17DC')

Start Address: '1EB3'

|           |                         |
|-----------|-------------------------|
| 1EB3 09F5 | RESTRO LODR,R1 *SAVR0+1 |
| 1EB5 09F6 | LODR,R2 *SAVR01+1       |
| 1EB7 09F7 | LODR,R3 *SAVR02+1       |
| 1EB9 17   | RETC,UN                 |

Restores contents of locations '17DA' to '17DC' to current selected G.P. register bank. Note relative addressing.

### LIMITATION:

During monitor operations (BKPT, SINGLE STEP, etc.) these routines will be accessed by the monitor. Previously saved desired USER contents will be overlaid. Therefore, use of these routines is NOT ADVISED unless the user program is FULLY DEBUGGED and will run without error when RST is depressed.

## SOLUTION:

The MONITOR program also contains two "long form" save and restore registers routines which are NOT accessible to the user program. In these routines, all seven G.P. registers, and the PSW (upper and lower) are saved. Note particularly the handling of the PSL which contains the dynamic condition code.

## DIRECTION:

Detach Diagram 30B on the next page, and set it next to this page.

Then listen to tape 11A for a brief commentary on routines "SAVRG" and "RESTRG".

## DIAGRAM 30-A\*

## SAVE AND RESTORE REGS - GENERAL FORMAT

```
SAVRG STRA, R0 UREG SAVE R0
 SPSL GET PSL
 STRA, R0 UREG+8 SAVE PSL
 STRA, R0 UREG+10 SAVE PSL
 SPSU GET PSL
 STRA, R0 UREG+7 SAVE PSU
 PPSU 11 SET INTERRUPT INHIBIT
 CPSL R5 CLEAR REGISTER SWITCH
 STRA, R1 UREG+1 SAVE R1
 STRA, R2 UREG+2 SAVE R2
 STRA, R3 UREG+3 SAVE R3
 FPSL R5 SET REGISTER SWITCH
 STRA, R1 UREG+4 SAVE R1'
 STRA, R2 UREG+5 SAVE R2'
 STRA, R3 UREG+6 SAVE R3'
 CPSL 255 CLEAR PSL
 RETC, UN
```

```
RESTRG LODI, R1 H'77' PPSL INSTRUCTION OPCODE
 STRA, R1 UREG+9 CREATE A SUBROUTINE TO RESTORE PSL
 LODI, R1 H'17' RETC, UN INSTRUCTION OPCODE
 STRA, R1 UREG+11
 CPSL R5 CLEAR REGISTER SWITCH
 LODA, R1 UREG+1 RESTORE R1
 LODA, R2 UREG+2 RESTORE R2
 LODA, R3 UREG+3 RESTORE R3
 PPSL R5 SET THE REGISTER SWITCH
 LODA, R1 UREG+4 RESTORE R1'
 LODA, R2 UREG+5 RESTORE R2'
 LODA, R3 UREG+6 RESTORE R3'
 RESTR LODA, R0 UREG+7 GET PSU DATA
 IORA, R0 IFLG SET INTERRUPT INHIBIT IF REQUIRED
 LPSU RESTORE PSU
 LODA, R0 UREG RESTORE R0
 CPSL 255 CLEAR PSL
 BSTA, UN UREG+9 RESTORE PSL
```

NOTE: If temporary storage is within +63-64 byte displacement range, use RELATIVE address mode in RESTRG subroutine. In final application, physical memory layout may prevent this option.

\*TEMPORARY STORAGE ALLOCATION ASSOCIATED WITH DIAGRAM 30A IS ON THE NEXT PAGE.