265Ø MICRO BASIC

START ADDRESS: 8EF

265Ø MICRO BASIC

EDITOR

commands:
N.-NEW
E -ENTER
A -ALTER.
I -INSERT
L -LIST
G -GOTO
X -ETX
P -PIPBUG

special operators:
CR -carriage return
BS -backspace
DEL-delete
ESC-escape

INTERPRETER

statements:
L -LET
T -TEST
G -GOTO
S -GOSUB
R -RETURN
F -FOR
N -NEXT
I -INPUT
A -ACCEPT
P -PRINT
O -OUT
M -MEMORY
C -CALL
] -INCREMENT
[-DECREMENT
$ -REMARK
E -END

operators:
+ -ADD
- -SUBTRACT
* -MULTIPLY
/ -DIVIDE ($-255 \leq$ divisor ≤ 255)
-CHANGE SIGN
% -BYTE SWAP
! -RANDOM

relational operators:
= -EQUAL
: -NOT EQUAL
> -GREATER THAN
< -LESS THAN

numbers: INTEGER -32768 to 32767 (two's complement)

variables: A to Z -all are 16 bit binary.

line numbers: 1,2,3,...,32767. (incremental order)

NOTE: The program has been intensively tested and to the best of my knowledge is bug free. If you come across any bugs, or write anything interesting, please let me know. If you have any queries please send a self-addressed, stamped envelope for a reply. Unlike some software sources, I give you the complete source listing so that you can make mods, improvements and understand how it works.

Micro BASIC cannot be put in PROM because it uses self-modifying code to save space. Also PIPBUG must be resident. I.O. routines can't be changed because it uses relative indirect branches to get to COUT & CR+LF. again to save space.

(2)

EDITOR

Entry point - 8EF The editor will then prompt with >
Editor Commands: (¢=carrige return)

>N	NEW	; initialisers memory for a new program.
>E(line number)¢	ENTER	; enters a new line.
>A(line number)¢	ALTER	; prints out the existing line and allows you alter it.
>I(line number)¢	INSERT	; inserts a new line at the designated line number and moves the existing line and all lines following it along.
>L(line number)¢	LIST	; lists the text starting at the designated line until the end of text is reached or a key press is detected.
>G(line number)¢	GO	; causes entry into the micro BASIC interpretor, program execution will begin at the designated line number.
>X	ETX	; prints out the hex address of the end of text so that the boundaries for a dump are known.
>P	PIPBUG	; returns to PIPBUG.
>(any other key press)		; results in a reprompt.

Operation

The program text begins at BØØ with a start of text character, STX(Ø2), and expands up thru memory to the end of text character, ETX(Ø3). STX is always at BØØ. ETX moves up or down in memory depending on where the text ends. Adding or lengthening lines causes ETX to move up in memory; deleting or shortening lines causes ETX to move down. Any attempt to move outside these markers causes an error message and a reprompt.

Before entering text STX & ETX must be set. This is done by N.

>N ⟶ BØØ Ø2 STX
BØ1 ØD CR
BØ2 Ø3 ETX

The CR represents the first line. Because the text is defined between STX and the first ETX, N will effectively delete any text in memory, even though there may be other ETX's further on.

From N you can only enter line 1.

ESC ; Escape. At any time escape causes reprompt.

All working on a line is carried out in a line buffer. With Enter and Insert the line buffer is filled with the characters typed in from the terminal. With Alter the line buffer is first filled up with the line from text. BS(backspace) can be used to change characters. CR causes the contents of the line buffer to be put into the text at the appropriate place. ESC causes an immediate return to the editor command loop ie. a reprompt. Hence hitting ESC will not cause the line buffer to be stored in text and the text is left unchanged.

When in Enter or Alter, hitting DEL(delete) will cause that line in text to be deleted and all following lines will be moved down one line.

eg. to delete line 4. >E4¢
4 DEL
4

The orignal line 4 has been deleted. The same line number is reprinted because all the following lines have been moved down so what was once line5 has been moved to become line4. This can also be deleted if you wish. Hence to delete 5 lines starting ewith line4 one would delete line 4 five times. If you want to see a line or look for a line to delete, then use Alter. Dont use DEL when in Insert, because it will act like a CR and cause insertion.

>E(line number)¢ -prints out the line number, a space, then accepts characters into the line buffer. To get to the designated line the editor goes back to STX and counts the number of CR's along in memory until the number of CR's = line number. From N you can only enter Line 1.

eg.
```
>N                          BØØ  Ø2  STX
>E1¢          ⟶            BØ1  ØD  CR
1 HI!¢                      BØ2  48  H
2 ESC                       BØ3  49  I
>                           BØ4  21  !
                            BØ5  ØD  CR
                            BØ6  Ø3  ETX
```

CR causes the line buffer to be put in text. The new line replaces the old line in text (if there was one). If the new line is shorter,then all the following lines are moved down in memory so that only a single CR separates lines. If the new line is longer the following lines are moved up in memory to make room.

If there are say 25 lines in text, then you can Enter any line up to and including line 26. Trying to Enter line 27 or greater results in a reprompt because you tried to go past ETX.

>A(line number)¢ -fills up the line buffer with the specified line from text and prints it. Because the line buffer now has the line in it you can add characters to it and/or use BS to change existing characters. Hitting CR stores the line back into memory. The line exists from the first character to where the CR is hit. This means to change a character in the middle of the line, and to have the whole line put back into text, BS to it, retype it, then you must retype all the characters following it to get to the point where you can hit CR.

If, after Alter prints out a line you immediately hit CR, then the line is left unaltered in text. So if looking for a certain line to change, you can step thru the lines using CR until you reach the wanted line.

>I(line number)¢ -causes the line typed in to be stored in text at the designated line number. The previous line with that number and all lines following it are moved along in memory to make room for the new line. While in Insert all lines entered from the terminal will be inserted one after the other beginning at the specified line number.

eg.
```
before→ 4 LINE 4  →  >I5¢                →   4 LINE 4
        5 LINE 5     5 NEW LINE 5¢           5 NEW LINE 5
                     6 NEW LINE 6¢           6 NEW LINE 6
                     7 ESC                   7 LINE 5
```

>L(line number)¢ -lists all lines from the specified line to the end of text or until a key press is detected. Because the keyboard is looked at only at certain times it is necessary to hit a key several times. Use ESC so if you hit it too many times it will just reprompt.

>G(line number)¢ -jumps to the interpretor. Program execution begins at the specified line.

>X -causes the HEX address where the end of text is,to be printed out. STX is always at BØØ,therefore to dump a program onto tape, dump from BØØ to the address given by X.

>P -causes a return to PIPBUG.

Tape - use PIPBUG to load a program from tape, then type G8EF¢ to enter the editor, then type G(line number)¢. Don't hit N or the program is deleted.

Tip -a line can consist of only a CR if you want,so lines can be spaced apart without using much memory,so forgotten lines or mods. can be added without changing the line numbers of the other lines. Hence GOTO and GOSUB wont be upset.

eg.
```
5 LINE 5 (CR)                >L5(CR)        >E7(CR)          >L5(CR)
6 (CR)          ⟶           5 LINE 5       7 LINE 7(CR) →   5 LINE 5
7 (CR)                       6              8 (ESC)          6
8 LINE 8 (CR)                7                               7 LINE 7
                             8 LINE 8                        8 LINE 8
```

265ϕ MICRO BASIC LANGUAGE DEFINITION

Alan Peek.

[Program] ::= [Line]* ETX

[Line] ::= [statement]* (CR)

[Statement] ::= L [expr] = [vble] {, [expr] = [vble]}* (SP)|(CR)
- OR T [expr] [relop] [expr] {, [expr] [relop] [expr]}* (SP)|(CR)
- OR G [expr] {< OR >} (CR)
- OR S [expr] {< OR >} (CR)
- OR R (CR)
- OR F [expr] = [vble] (SP)|(CR)
- OR N [vble] (SP)|(CR)
- OR I [vble] {, [vble]}* (SP)|(CR)
- OR A [vble] (SP)|(CR)
- OR P {[expr]}* {[string]}* {,}* (SP)|(CR)
- OR O [vble] (SP)|(CR)
- OR M [vble] [<|>] [expr] (SP)|(CR)
- OR C [expr] (SP)|(CR)
- OR] [vble] (SP)|(CR)
- OR [[vble] (SP)|(CR)
- OR $... remark ...$
- OR E

[expr] ::= [decimal number]|[vble]
- OR [decimal number] [dec. number]* [operator]*
- OR [vble] [vble]* [operator]*
- OR [dec. number]* [vble]* [operator]*

[relop] ::= [=]|[<]|[>]|[:]

[dec. number] ::= [decimal digit]*

[vble] ::= A|B|C... X|Y|Z

[operator] ::= + - * /

[string] ::= " any character "

[single operator] ::= # ! % ∧
(in any expr)

relop = relational operator.

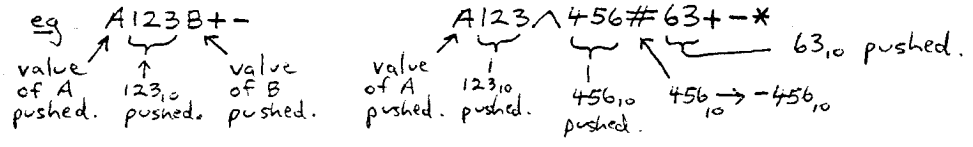
{ } means- optional * means: 1 or more times expr = expression

::= means:- is defined as | means: exclusive OR vble = variable

The program that MICRO BASIC interprets consists of a series of 1 or more lines. Each line consists of one or more statements and a CR that marks the end of the line. A statement consists of a letter which indicates the statement, followed by the information that the statement needs to carry out the statement task. Some statements are terminated by a space or CR, others are terminated by a CR.

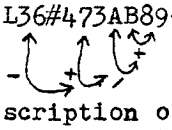
Expression Evaluation - is done in Reverse Polish Notation (RPN), similar to a H-P calculator. With RPN, numbers are pushed onto a first in, last out stack. A stack pointer points to the last number pushed. Double operand operators work on the last two numbers pushed. Single operand operators work on the last number pushed ie. the number the stack pointer is pointing to presently.

The expression consists of decimal digits and/or variables. The expression is evaluated from left to right. Each operand is pushed on the stack. When evaluating an expression each text character is looked at from left to right. If a character is a letter then this means a vble, and the value of the vble is fetched from the vble table and pushed on the stack. Because vbles are single characters, a single vble can be immediately followed by another vble and they are known to be separate. But, when the character is a decimal digit it may be followed by more digits to make up a decimal number. Hence when a digit is seen each following character is looked at until a non-digit is seen. This will mark the end of the decimal number, and so all the previous digits will be processed so that the binary value of the decimal number is left on the stack. Obviously a vble will terminate a number and so will a single or double operand operator. But another character is needed to separate (terminate) a number which is immediately followed by another number when an operation is not yet wanted. This is done by ^ which is equalivalent to the ENTER key found on H-P calculators. It's used to separate a group of digits from another group.



Note - that unlike the stack on the H-P calculator, the stack can't be used as a memory between statements. Everything that is pushed on the stack in a statement must be operated on so that at the end of the expression there is only one number left on the stack. ie. after the expression has been evaluated the stack pointer must have moved down one number. Then according to the statement, this number the stack pointer is pointing to is treated as the result of the expression and then this number is popped off the stack and delt with, the stack pointer being left at the same place where it was before the statement. This doesn't apply to GOSUB.

eg L36#473AB89+/+-=C



not allowed → L4^333G+=K
because 4 left on stack.

Description of how the stack is used by each statement is given under the statement descriptions.

Operators.

Double Operand Operators. -work on the two previous operands.

- \+ -adds together the two previous operands.
- \- -subtracts the two previous operands.
- * -multiplies the two previous operands.
- / -divides the two previous operands. Note the maximum absolute number you can divide by is 255 because I'm using a Signetics routine.

Single operand operators. -work on the previous operand ie. the number the stack pointer is currently pointing to.

\# -change sign. All numbers are assumed to be positive unless followed by #. If the operand is +ve then becomes -ve. If the operand is -ve then becomes +ve.

eg L6#=K PK → -6
L K#=M PM → 6

% -swap bytes. The most significant byte of the number the stack pointer is pointing to is swapped with the least significant byte. When storing or reading a vble to or from memory using the statement M, only the least significant byte is used. This is because a vble is 16 bits (two bytes) whereas a memory location is 8 bits (1 byte). Hence % allows both bytes to be handled. See under the statement M for examples.

! -RANDOM. ! following an operand in an expression results in a random number from Ø to operand - 1 being left on the stack in place of the original. Note it won't change the value of the vble on the vble table. ! can appear anywhere in an expression as long as it's preceeded by an operand. eg L69AB5Ø!4++-*=G causes a random number between Ø and 49

Note-Random works by taking the operand off the stack and entering a loop where the operand is continually decremented. After each decrement the terminal is looked at for a key press. This keeps going and when the operand reaches Ø the original operand is reloaded and the loop continues. When a key press is detected the current value of the operand in the loop is left on the stack. To indicate that the computer is waiting for a key press it prints ! at the terminal and you can then hit any key. The computer will acknowledge with * and will continue.

ERROR MESSAGES.

Most, but not all, errors are detected and result in an error message on the terminal and a return to the editor.

?
S(line number) -means the stack was popped too far. eg. L6+=A
?
L(line number) -means an improper line eg. a missing vble.
Note-if no apparent line error then means a divide by Ø occurred at that line.
?
T(line number) -means an attempt to go past STX or ETX was made.
eg. G2Ø when only 1Ø lines in program.
Note-the line number is meaningless and is only an artifact of the simple error handling.

The error messages are helpful but there is no substitute for deduction!

STATEMENTS.

Multiple statements per line are allowed, each statement being separated from each other by one or more spaces. The line is terminated by a carriage return. Putting more than one statement per line saves memory and speeds execution. Each statement is represented by a single letter eg. the statement PRINT would consist of a single P . No spaces are allowed in a statement (except in a string "..." in PRINT), because a space means statement termination. Between statements more than one space is allowed and this can be used to increase readability of the text by allowing in-stepping.

```
eg.   23 F6=A
      24          FC=D      P"SMALL IS GREAT"
      25          S34       $SUBROUTINE JUMP$
      26          ND
      27 NA
```

```
eg.   6 L9#AB+/=C P"HI" LC=T G12     same as:
      6      L9#AB+/=C       P"HI"          LC=T      G12
```

note: no spaces within a statement.

Note: there are 4 statements not represented by a single letter but by a single character.

$..$ -REMARK
' -means PRINT but without a CR+LF first.
] -means increment the vble value in the vble table.
[-means decrement the vble value in the vble table.

STATEMENTS.

LET L[expr]=[vble]{,[expr]=[vble]}* (SP)|(CR)

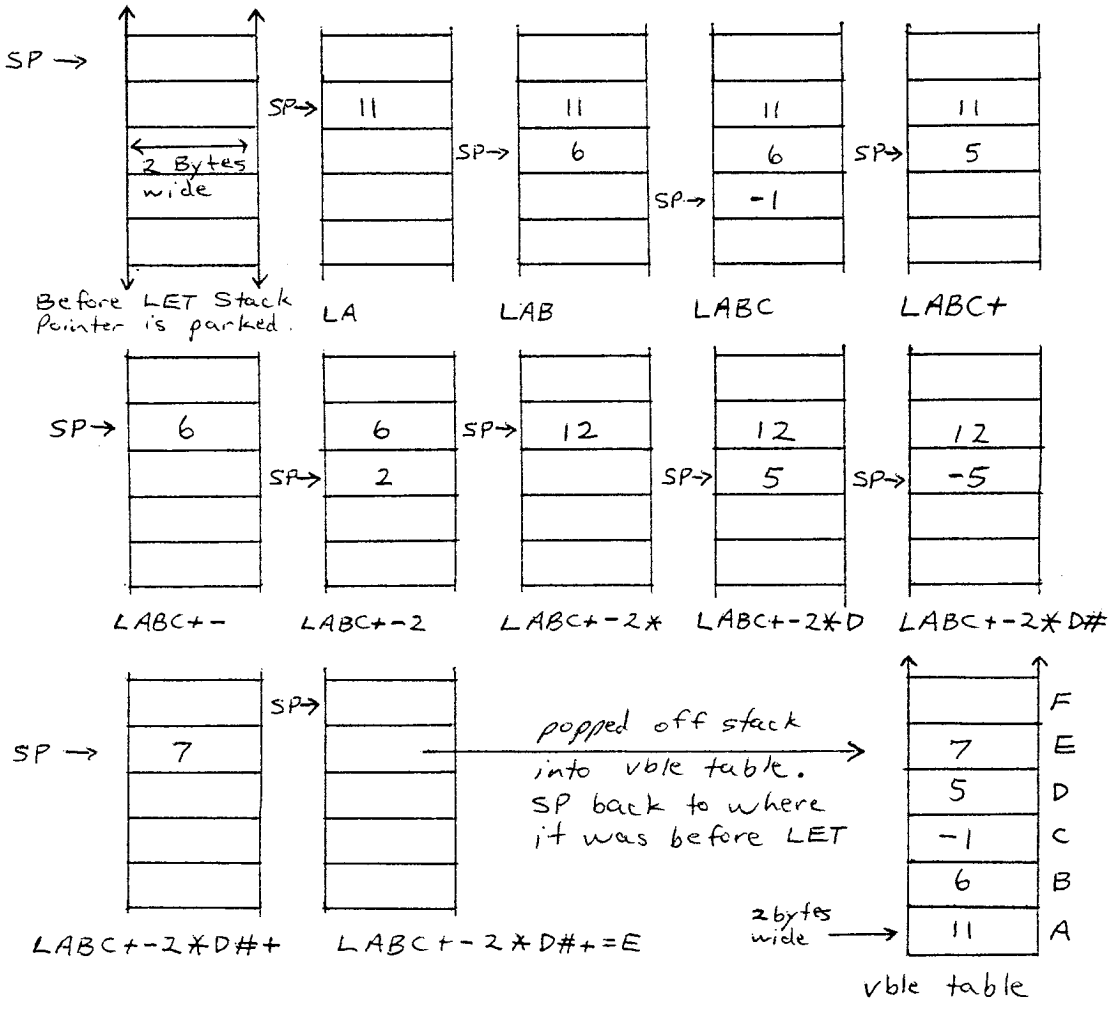
The expression to the left of the "=" sign is evaluated and the result is then popped off the stack and stored in the variable table under the label given by (vble). A space or CR terminates LET, a comma means there is another (expr)=(vble) to follow and hence stay in LET.

eg's.

```
L215=V
LA22+=C
LA165#*=E
L36^4-=F
LABCD++*=K
  or LCD+B+A*=K
LB%=B
LAA*=T

LØ=R,Ø=S,15=T,3469=B,56#=J,F#=Y
LB=G
L1111#=A
L1ØØ11+=R
LBB*4AC**-=R
LM1+=M
LG2/=L
L1ØØE/4*8/=X
```

example of stack operation. eg. LABC+-2*D#+=E say A=11 B=6 C=-1 D= 5



Note. Where the stack pointer is parked before doing LET can vary because the stack is used to store GOSUB return addresses, and loop addresses in FOR/NEXT. The important thing is that after LET is finished the stack pointer is back to where it was before LET.

The stack expands downward in memory. ie. the more things pushed on the stack the lower in memory the stack pointer is pointing. If there are no return addresses on the stack then the SP=464 HEX. If one number is now pushed on the stack then SP=462 HEX. It moves up or down in jumps of two bytes because each number is 16 bits to represent + or - 32K. When a number is popped from the stack then the SP moves up 2 bytes. If the SP tries to go above 464 HEX then an error message occurrs. The stack is quite deep and can expand from 464 HEX down to 4ØØ HEX.

TEST T[expr][relop][expr]{,[expr][relop][expr]}* SP|CR

This is equalivalent to BASIC'S IF. Because statements are represented by single letters, and I is used to represent INPUT, then T is used to represent TEST.

The expression on each side of the relational operator is evaluated and the results are compared. If the comparison is true with respect to the relational operator then program execution continues along the current line. If the comparison is false then the rest of the line is skipped and program execution continues on the next line. If (expr)(relop)(expr) is followed by a comma then it means another (expr)(relop)(expr) is to follow ie. stay in TEST. As soon as one of the tests fail then the line is skipped. So in effect a comma is acting like AND - only if all the (expr)(relop)(expr)'s in a TEST are true will execution along the current line continue.

= means if (expr)=(expr) continue along the current line.
< means if (expr) less than (expr) continue along current line.
> " " " greater than " " " " " "
: means if (expr) does not equal (expr) continue along current line.

eg.s. TA= 5 G51 if A=5 then a jump to line 51 will be done.
G23 otherwise a jump to line 23 will be done.

TA<B,Y=8,6:R,4589#=L
TA67+=J,BCTG++->H,RT-8+<67#W/,1=A

to simulate ≤ and ≥ do:		
X≤Y	put	X<Y1+
X≥Y	put	X>Y1-

Note-that expressions, of course, are evaluated but you can't assign a result to a vble label- that's the job of LET. Hence nothing in the vble table is changed by TEST.

GOTO G[expr]{< OR >}CR

The expression is evaluated and then a jump is made to the line number given by the result. If you try to jump past STX or ETX an error message will result. G is always at the end of a line.

If the expression is followed by < or > then this means a relative jump.

< -means go back the number of lines given by the expression result
> -means go forward the " " " " " " " "

Relative jumps make program execution faster and relocatable.

eg's. G1Ø GK
GA6+
G3> if on line 4 then will go to line 7.
GR1-< if on line 4 and R=3 then will go to line 2.
GØ< will repeat that line eg.a TEST loop on a single line.

GOSUB S[expr]{< OR >}CR

Is the same as GOTO only the line number following the line in which the S occurred is pushed on the stack. Then the program jumps to the subroutine -the line number of the subroutine being given by the result of the expression. Relative subroutine jumps can also be made. Return from the subroutine occurrs when R (RETURN) is encountered.

The way the MICRO BASIC gets to a specified line is to go back to STX and count all the CR's thru in the text until the CR count= line number. With relative jumps the MICRO BASIC either counts forward or backward the CR's until the CR count = relative jump. Hence relative jumps are much faster because counting does not start right back at the start of text.

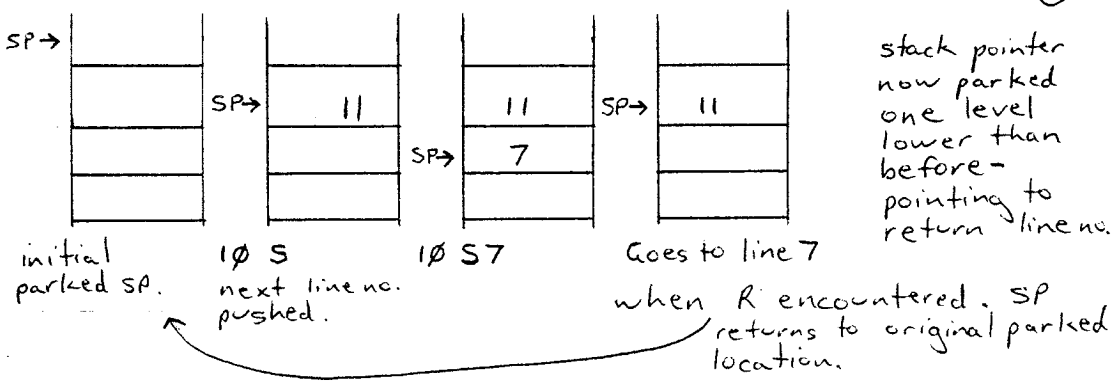
S is always at the end of a line.

Subroutines can be nested to the level of the stack

RETURN R CR

—causes return from a subroutine. R is always at the end of a line because as soon as R is seen then RETURN occurrs and anything further on the line would never be executed.

eg on line 10 · 10 S7 (CR)



A subroutine can call another subroutine -subroutines can be nested to the depth of the stack. The reason why G, S, R must be at the end of a line is because anything following them on the line would never be executed.

FOR F[expr]=[vble] (SP)|(CR)

FOR is like LET except -only one (expr)=(vble) is allowed.
-the line number following the line where FOR appears is pushed on the stack. Other statements can follow the FOR statement on the same line (although they will not be part of a FOR/NEXT loop). The step in FOR is always a -1 ie-the vble is continually decremented until it reaches zero (this is easy to test for)

NEXT N[vble](SP)|(CR)

When an N is encountered the value of the vble is tested for zero. If the vble is not zero then it is decremented by 1 and the new value put back into the vble table. Then the address of the line number after the FOR is read from the stack (the SP is does not change) and program execution loops back to that line number. The loop will keep looping until the vble reaches zero. When the vble reaches zero then execution will continue along the line NEXT is on (and the stack will be popped so as to throw away the no longer needed loop address).

eg.
```
11 F3=B                    when run:    LINE 12 B=3
12 P"LINE 12 B="B                       LINE 12 B=2
13 NB                                   LINE 12 B=1
14 P"LINE 14 B="B                       LINE 14 B=Ø
```

FOR/NEXT loops can be nested to the depth of the stack. FOR/NEXT loops can't be overlapped.

```
F3=X          F3=X
F67H+=K       F67H+=K
NK            NX
NX            NK
```
acceptable not acceptable.

Note-you must always have an (expr)=(vble) ie. if you want to loop the number of times given by say D then you must have FD=D

IN I[vble]{,[vble]}*(SP)|(CR)

-prompts with a ? and gets a number input. After typing the decimal digits hit CR. The number is then converted to binary and stored in the vble table at the label given by (vble). Hitting ESC causes return to the editor. Any other key causes a reprompt- use if you make an error.
eg. IA IZ IA,Z IT,J,P

ACCEPT A[vble](SP)|(CR)

takes any ASCKI character from the keyboard (except ESC) and stores it in the low byte of the vble, high byte=Ø. No ? is printed so A(vble) can be repeatedly used to input a string. Note-the ASCKI character value will be in decimal eg CR=ØD HEX =13 DEC.
Accept does not try to convert the fetched character to a number.
see later for string examples.

(10)

PRINT P{[expr]}* {[string]}* {,}* (SP)|(CR)

When a P is seen then any following expression(s) is evaluated and the result is printed. Any characters between inverted commas "...." are printed exactly as is on the terminal. A comma (which is not part of a string) causes a space to be printed. P by itself will cause a CR+LF. ' -means PRINT but without first doing a CR+LF. When P prints a vble, it first converts the value of the vble to decimal and then prints a decimal number. If you want to output a vble as an ASCKI character eg - when outputing a string from memory then use OUT.

eg's.	P34^6+	4Ø
	PA	-69 if A=-69
	PABC	-69Ø23 if A=-69 B=Ø C=23
	PA,B,C	-69 Ø 23
	P"A="A,,,"B="B	A=-69 B=Ø
	PA PB	-69
		Ø
	PA,, L1=E 'E	-69 1

OUT O[vble]

causes the vble to be treated as an ASCKI character and to be printed on the terminal as is, ie. it is not converted to a decimal number No CR+LF is done so OUT can be used to print a string from memory.

eg. OT - causes the letter A to be printed if the decimal value of T is 65.

See later of examples of how to implement strings.

MEMORY M[vble][<or>][expr](SP)|(CR)

This is equalivalent to the PEEK and POKE of BASIC.

M(vble)<(expr) - the expression is evaluated and used as the decimal address to read memory. The value of the memory byte read is put in the low byte of the specified vble. The high byte is set to zero. The address is in decimal, so to read memory location 1ØØØ HEX do MB<4Ø96

M(vble)>(expr) - causes the low byte of the specified vble to be stored in the memory location specified by the result of the expression evaluation. eg. to store 5 at 1ØØØ HEX L5=K
MK>4Ø96

to store the high byte use the function % to swap MSB and LSB.
eg. -to store a 16 bit vble say X
say Q contains memory address MX>Q LX%=X,Q1+=Q MX>Q

-to read a 16 bit value into a vble MX<Q LQ1+=Q MZ<Q LZ%X+=X

If using memory mapped I.O. then you can talk to it using MEMORY.

CALL C[expr](SP)|(CR)

The expression is evaluated and used as an absolute address then an absolute jump is done to this address. This allows jumps to machine language routines. To return from the routine do a jump back to the command loop ie. BCTA,UN to cmd loop. - 1F Ø7 35
Note - that MICRO BASIC expects WC=Ø, C=Ø, COM=Ø . Therefore if your machine routine changes any of these to 1 it must change it back to Ø before jumping back to the cmd loop.

Remember that the address is in decimal.

INCREMENT]vble SP|CR

the value of the vble in the vble table is incremented by one. Note -that this is a statement and can not be used in an expression. ie. if incrementing in LET must do say: LA1+=A cant do: L]A
It is often useful to be able to increment a vble without using LET which would use more memory. To remember that] means increment think of] as being similar to > (greater than). The new value is greater than the old value

DECREMENT [vble SP|CR

the value of the vble in the vble table is decremented by one.
[is similar to < (less than). new value < old value
Like INCREMENT, DECREMENT can't be used in expressions.

REMARK $.....anything.....$

Everything between the dollar signs is ignored. $ like other statements must be separated by at least one space from other statements. When a $ is seen, the interpretor enters a loop and keeps skipping along in the text until another $ is seen. Hence everything is ignored and so REMARK can continue over many lines. REMARK can exist between statements, at the start of a line, at the end of a line, or can be the whole line.

eg. L6=Z $ THIS IS A REMARK $ P"LINE 12"

END E

Prints an E at the terminal, then the line number where the E occurred, then goes to the Editor. More than one E can be in the program. Can also be useful in debugging to check on program flow.

SAMPLE PROGRAM

```
1 P"GUESS" L4Ø96=A $string will be stored at 1ØØØhex and on in memory$
2 P"WHAT IS YOUR NAME? "
3 AL                $get an ASCKI character and store in memory$
4 ML>A ]A TL:13 G1< $keep getting & storing until a CR(13decimal)$
5 P S17             $CR+LF then print out string$
6 '",NUMBER FROM 1 TO " IN  $get the upper limit of guess range$
7 P LN!1+=R,Ø=T     $T will keep count of the number of guesses$
8 P"GUESS = " IG ]T   $increment guess count$
9 TG>R P"TOO BIG" G8
1Ø TG<R P"TOO SMALL" G8
11 P S17            $print string(name)$
12 '",GOT IT IN",T,"TRIES"
13 P"WANT TO PLAY AGAIN? "
14 AB TB=89 G7      $89decimal=the letter Y $
15 P"GOODBYE " S17  $if not Yes then sign off$
16 E
17 L4Ø96=A          $subroutine to print out string$
18 ML<A
19 TL:13 ØL ]A G1<  $ keep outputting till CR $
2Ø R
```

To enter hit N for NEW, then E1 to start entering the lines. You dont have to enter the comments which are between dollar signs. To run type G1.

Happy computing,
Alan Peek.

PROGRAMS

```
 1 P"BLACKJACK" $AP 14/2/79$
 2 P L1ØØ=F
 3 P"YOU HAVE $"F
 4 P P P"YOUR BET" IW
5  TW>F P"TOO MUCH"     G3
 6 TW<1Ø P"TOO LITTLE"     G4
 7 S46
 8 P"YOUR 1ST CARD IS " S38
 9 LY=A,2=T
 1Ø S46
 11 P"YOUR 2ND CARD IS " S38
 12 LY=B
 13 S46
 14 P"DEALER SHOWS " S49
 15 LY=C
 16  LAB+=A
 17  P"YOU HAVE",A,"SHOWING"
 18  TT>4 LW2*=W G37
 19  TA>21 P"YOU'RE BUSTED" G32
 2Ø  P"HIT?" AZ TZ:89 G5>
 21  S46
 22  P"YOUR NEXT CARD IS " S38
 23  LY=B ]T
 24  G8<
 25 S46
 26 P"DEALER GETS " S49
 27 LYC+=C P"DEALER SHOWS",C
 28 TC<17 G3<
 29 TC>21 P"DEALER BUSTED" G8>
 3Ø TA>C G7>
 31 P"YOU LOSE"
 32 LFW-=F TF>1Ø G3
 33 P"YOU'RE OUT"
 34 P"WANT TO PLAY AGAIN?"
 35 AA TA=89 G2
 36 E
 37 P"*YOU WIN*" LFW+=F G3
 38  TX:1 G4>
 39  '"ACE" P"OPTION" IQ
 4Ø  TQ=1 R
 41  L11=Y R
 42  TX=11 '"JACK" R
 43  TX=12 '"QUEEN" R
 44  TX=13 '"KING" R
 45  'X R
 46   P L13!1+=Y,Y=X
 47   TY>1Ø L1Ø=Y
 48   R
 49 TX:1 G7<
 5Ø '"ACE" T11C+>21 R
 51 T11C+>A G1Ø<
 52 R
```

Blackjack:one closer to 21 wins.Ace option=1or11.Face card =1Ø.Dealer holds on 17 or above If delt 5 cards and still under 21 you win double your bet.
Acey-Ducey:you are delt two cards and you bet that the next card is between those two.
Life:one dimension.?E allows you to enter line pattern.Any other key will cause a prompt for the no. of generations.
Birth:2or3 live neighbors.
Survival:2or4 live neighbours.
See BYTE Dec/1978,page 68.

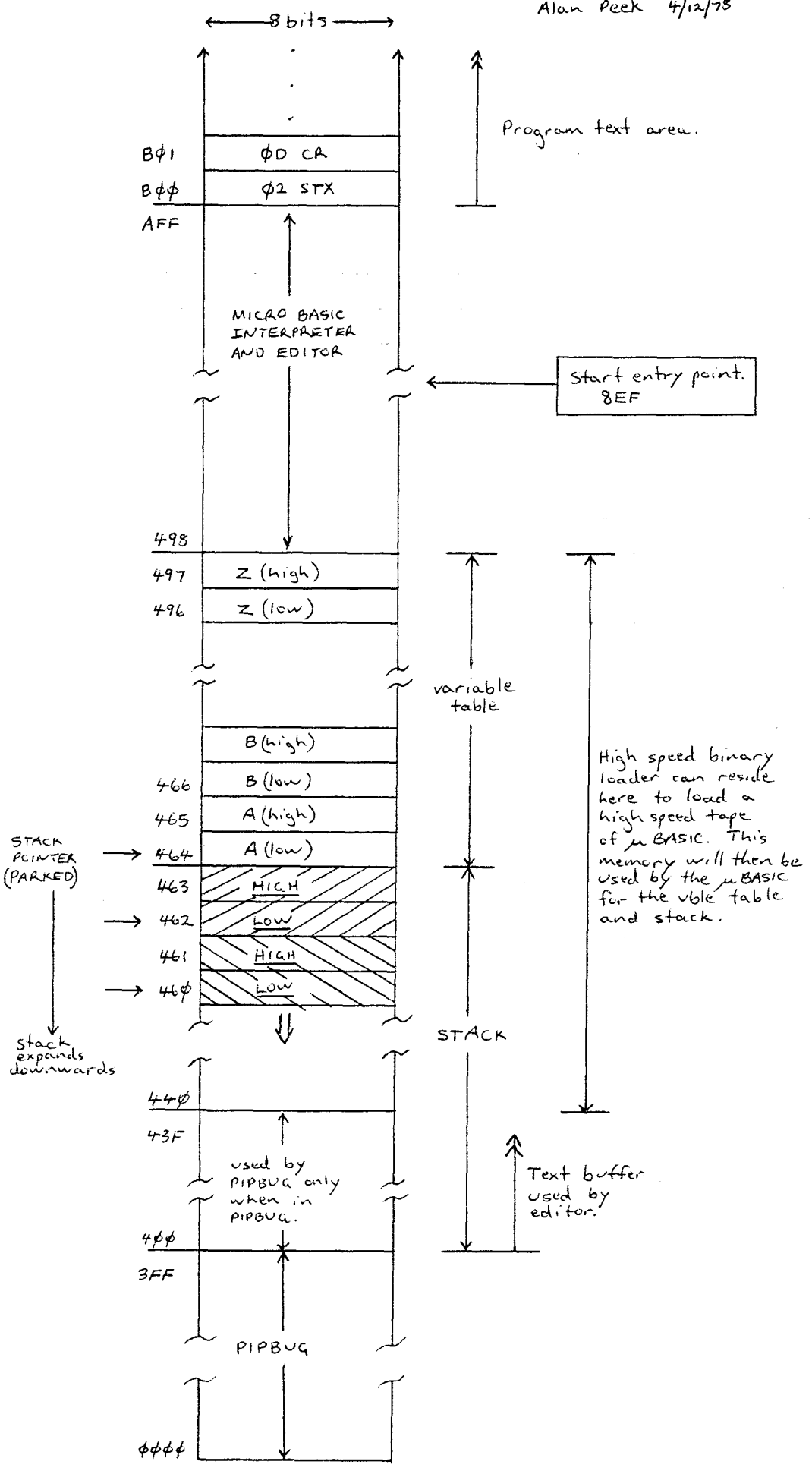
```
 1 P"LIFE" L4Ø96=Y,413Ø=H,Ø=C
 2 MC>Y2- MC>Y1-          ← put spaces in front of line.
 3 P"?" AA TA:69 IG G13
 4 P LY=X
 5 AC
 6 TC=13 G4>
 7 TC=32 LØ=C G2>          } enter line and start storing it at 4Ø96 (1ØØØ Hex).
 8 L1=C
 9 MC>X ]X G4<
 1Ø   LØ=C
 11   TX>H1- G3            } fill up line with spaces if need be
 12   MC>X ]X G1<
 13 FG=G
 14 P LY=X
 15 TX>4127 G3>            } Print line Ø ⇒ space 1 ⇒ O
 16 MC<X ]X TC=Ø ', G1<
 17 '"O" G2<
 18 LY=X,H=K
 19 F32=Z
 2Ø MA<X2- MB<X1- MC<X ]X  } check out the two neighbours each side
 21 MD<X ME<X1+ LABDE+++=F
 22 TC=1 G3>               } work out birth or survival
 23  TF:2,F:3 G3>
 24  L1=C G2>
 25   TF:2,F:4 LØ=C
 26 MC>K ]K                ← produce new line in memory starting at 413Ø
 27 NZ
 28    LY=X,H=K
 29    F32=Z
 3Ø    MC<K MC>X ]X ]K     ← Move new line back to replace old line
 31    NZ
 32 NG                     ← keep going for specified number of generations.
 33 G3
```

```
 1 P"ACEY-DUCEY" $AP 13/2/79$
 2 L1ØØ=Q
 3 P"YOU NOW HAVE",Q,"DOLLARS"
 4    G7
 5 LQM+=Q   G3
 6 LQM-=Q G3
 7 P P L13!1+=A,13!1+=B
 8 TA>B LA=C,B=A,C=B
 9 P"HERE ARE YOUR NEXT TWO CARDS"
 1Ø LA=X S27
 11 LB=X S27
 12 P"WHAT IS YOUR BET" IM
 13 TM=Ø P"CHICKEN" G7
 14 TM<Q1+ G3>
 15 P"YOU ONLY HAVE",Q,"DOLLARS"
 16 G4<
 17 P L13!1+=X S27
 18 TX>A G2>
 19 G3>
 2Ø TX>B1- G2>
 21 P"YOU WIN" G5
 22 P"YOU LOSE" TM<Q G6
 23 P"SORRY,YOU BLEW IT"
 24 P"WANT TO PLAY AGAIN ?"
 25 AZ TZ=69 P G2
 26 E     89
 27     TX=1 P"ACE" R      } subroutine to print card value
 28     TX=11 P"JACK" R
 29     TX=12 P"QUEEN" R
 3Ø     TX=13 P"KING" R
 31     PX R
```

O_OOO -is a glider

OO_OOO_OO -is pretty

265Ø MICRO BASIC MEMORY MAP



2650 BASIC
SOURCE LISTING

Alan Peek 4/12/78

"PC"+1 & load R1 — VERSION 12/1/79

"PC"+1 & load →	498	09 1Ø	LODR, R1	'PC' low
	9A	D9 ØØ	BIRR, R1	increment 'PC' low
	9C	C9 Ø6	STRR, R1	put back into 'PC'
	9E	E5 ØØ	COMI, R1 Ø	was result zero
	AØ	98 Ø6	BCFR, 'Ø'	if zero then do the same to 'PC' high
	A2	Ø9 Ø5	LODR, R1	'PC' high
	A4	D9 ØØ	BIRR, R1	increment 'PC' high
$2	A6	C9 Ø1	STRR, R1	then put back
"PC" $1	A8	ØD ØB ØØ	LODA, R1	get text character into R1
	AB	17	RETC, UN	

"PC" - 1 & load R1

"PC"-1 & load →	4AC	Ø9 7C	LODR, R1	'PC' low
	AE	F9 ØØ	BDRR, R1	decrement 'PC' low
	BØ	C9 78	STRR, R1	put back into 'PC'
	B2	E5 FF	COMI, R1, FF	if new 'PC' low = FF then must decrement 'PC' high
	B4	98 72	BCFR, true $1	to load R1 & Return
	B6	Ø9 71	LODR, R1	'PC' high
	B8	F9 6C	BDRR, R1 $2	decrement 'PC' high & jump to store & load 'PC' high never equals zero.

RESET "PC"

Reset "PC" →	4BA	Ø6 ØØ	LODI, R2	"PC" low — This resets the 'PC' back to the start
	BC	Ø5 ØB	LODI, R1	"PC" high
$4	BE	CA 6A	STRR, R2	"PC" low of text
	CØ	1B 64	BCTR, UN $2	store R1 & Return

POP/PUT TOP OF STACK INTO "PC"

Pop stack→"PC"→	4C2	3B 34	BSTR, UN	Pop stack into R1 & R2
	C4	1B Ø2	BCTR, UN $3	jmp to put R1 & R2 into 'PC'
Put stack→"PC"→	C6	3B 36	BSTR, UN	Put top of stack into R1 & R2
$3	C8	3B 74	BSTR, UN $4	R1, R2 → "PC" Then continue into PC+1 till CR.

"PC"+1 & load — LOOP TILL CR

"PC"+1 + loop till CR→	4CA	3B 4C	BSTR, UN	"PC"+1 & load
	CC	E5 Ø3	COMI, R1 ETX	if trying to go past end of text then error
	CE	18 8A	BCTR, '=' *	to error
	DØ	E5 ØD	COMI, R1 'CR'	
	D2	98 76	BCFR, '='	PC+1 loop till CR
	D4	17	RETC, UN	

"PC"-1 & load LOOP TILL CR

"PC"-1 loop till CR→	4D5	3B 55	BSTR, UN	"PC"-1 & load
	D7	E5 Ø2	COMI, R1 STX	if trying to go past start of text then error
	D9	1C ØB BD	BCTA, '='	error
	DC	E5 ØD	COMI, R1 CR	
	DE	98 75	BCFR, '='	PC-1 loop till CR
	EØ	17	RETC, UN	

PUSH "PC" ONTO STACK

PUSH "PC" →	4E1	ØA 46	LODR, R2	"PC" high
	E3	Ø9 45	LODR, R1	"PC" low
	E5	1B 35	BCTR, UN	Push R1 & R2 and Return

POP STACK INTO VBLE TABLE, VBLE in R1

Pop Stack →	4E7	D1	RRL, R1	so that each letter can address 2 bytes
	E8	Ø8 B9	LODR, R9	* stack high
	EA	CD 23 E2	STRA, R9, R1+	vble high
	ED	3B Ø5	BSTR, UN	SP+1
	EF	Ø8 B2	LODR, R9	* stack low
	F1	CD 43 E2	STRA, R9, R1-	vble low then run down into SP+1

STACK POINTER +1 or +2

SP+1 →	4F4	Ø4 Ø1	LODI, R9	1
	F6	1B 12	BCTR, UN	$ SP add
Pop stack →R1,R2→	F8	3B Ø4	BSTR, UN	put stack into R1 & R2
SP+2 →	FA	Ø4 Ø2	LODI, R9	2
	FC	1B Øc	BCTR, UN	$ SP add

```
                        PUT TOP OF STACK INTO R1 & R2, SP unchanged.
Put stack into R1,R2 →  4FE  ϕ9 A3      LODR, R1 *SP    MSB on top of stack
                        5ϕϕ  3B 72      BSTR, UN  SP+1
                        ϕ2   ϕA 9F      LODR, R2 * SP   LSB on top of stack
                        STACK POINTER -1 or -2
SP-1 →                  5ϕ4  ϕ4 FF      LODI, Rϕ  -1
                        ϕ6   1B ϕ2      BCTR, UN  ↑ SP add
SP-2 →                  ϕ8   ϕ4 FE      LODI, Rϕ  -2
                        ϕA   88 18      ADDR, Rϕ  SP low  WC=ϕ
                        ϕC   E4 64      COMI, Rϕ  if SP low >64 then
                        ϕE   19 D8      BCTR, GT *  error because stack has
                        1ϕ   C8 12      STRR, Rϕ  SP low   been popped too much.
                        12   17         RETC, UN
                        PUSH STACK WITH VBLE   letter in R1
Push Stack → ΣVble      513  D1         RRL, R1
                        14   ϕD 23 E2   LODA, Rϕ, R1+  vble high byte
                        17   C2         STR2, R2
                        18   ϕD 43 E2   LODA, Rϕ, R1-  vble low byte
                        1B   C1         STR2, R1
                        PUSH R1, R2 onto stack
Push R1,R2 → onto stack ϕ5   51C  3B 66     BSTR, UN  SP-1
                        1E   C9 83      STRR, R1 * to SP   LSB
                        2ϕ   3B 62      BSTR, UN  SP-1
STACK POINTER           22   CE ϕ4 64   STRA, R2  onto stack MSB
                        25   17         RETC
Change Sign →           526  ϕA FB      LODR, R2 *  MSB        To change sign must complement then add 1
                        28   3B 4A      BSTR, UN  SP+1
                        2A   ϕ9 F7      LODR, R1 *  LSB
                        2C   26 FF      EORI, R2   } complement
                        2E   25 FF      EORI, R1   }
                        53ϕ  3B 6C      BSTR, UN $5   Push R1,R2 but dont do SP-1 then continue into stack+1
                        STACK +1, -1
Stack+1 →               532  ϕ5 ϕ1      LODI, R1  1  }
                        34   ϕ6 ϕϕ      LODI, R2  ϕ  } +1
                        36   1B ϕ4      BCTR, UN  to push R1,R2 then add and RETURN
Stack-1 →               38   ϕ5 FF      LODI, R1 FF  }
                        3A   ϕ6 FF      LODI, R2 FF  } -1
Push R1,R2 and ADD →    3C   3B 5E      BSTR, UN  push R1,R2 then run into ADD
                        ADD / SUBTRACT   The only difference between add & subtract is instruction's carry
ADD →                   53E  ϕ4 8E      LODI, Rϕ  set up instruction for ADD
                        4ϕ   75 ϕ1      CPSL, C=ϕ
                        42   1B ϕ4      BCTR, UN  jump over SUBTRACT
SUBTRACT →              44   ϕ4 AE      LODI, Rϕ  set up instruction for SUB
                        46   77 ϕ1      PPSL, C=1
                        48   C8 ϕ9      STRR, Rϕ  set up for add or subtract
                        4A   77 18      PPSL, WC=1, RS=1
                        4C   ϕ5 ϕ4      LODI, R1
                        4E   ϕ6 ϕ2      LODI, R2
loop                    55ϕ  ϕD C5 23   LODA, Rϕ, R1,- *
                        53   [ ] C5 23   ADDA/SUBA, Rϕ, R2,- *
                        56   CD E5 23   STRA, Rϕ *
                        59   5A 75      BRNR, R2  loop
                        5B   75 19      CPSL  RS=ϕ, WC=ϕ, C=ϕ
                        5D   1F ϕ4 FA   BCTA, UN  SP+2 and RETURN
                        TEST TOP OF STACK   stack +ve → Rϕ +ve; stack -ve → Rϕ -ve; stack ϕ → Rϕ ϕ
Test Stack →            56ϕ  ϕ8 C1      LODR, Rϕ *  MSB (check high byte)
                        62   16         RETC, -ve  if number -ve, MSB -ve
                        63   15         RETC, +ve  if number +ve MSB may be +ve
                        64   ϕ7 ϕ1      LODI, R3  index
                        66   ϕF E5 23   LODA, Rϕ, *, R3  LSB (check low byte)
                        69   14         RETC, ϕ  if low byte = ϕ then no. = ϕ
                        6A   ϕ3         LODZ, R3  ∴ must be +ve
                        6B   17         RETC, UN
```

```
                    TEST R1
Test R1→ 56C  E5 Ø3    COMI, R1 'ETX'        R1 number→RØ=FF letter→RØ=+ve neither→RØ=Ø c.c. set
         6E   1C Ø8 BD  BCTA, true to error
         71   E5 3Ø     COMI, R1  R1<3Ø
         73   9A Ø2     BCFR, true  <
$6       75   2Ø        EOR2  set RØ to zero
         76   17        RETC, UN  not a digit or letter
         77   E5 3A     COMI, R1  R1< 3A
         79   9A Ø3     BCFR, <
         7B   Ø4 FF     LODI, RØ FF  to make RØ -ve
         7D   17        RETC, UN  R1 = a number
         7E   E5 41     COMI, R1
        58Ø   1A 73     BCTR, -ve <  $6 ∴ not a letter or no
         82   E5 5B     COMI, R1
         84   9A 6F     BCFR, <  $6 ∴ not a letter or no.
         86   Ø1        LODZ, R1 set RØ to +ve
         87   17        RETC, UN
                    PUT TOP TWO NUMBERS ON STACK INTO */ TABLE
        588   3B 56     BSTR, UN  test stack
         8A   C8 ØF     STRR, RØ  store test in sign flag
         8C   3E Ø5 26  BSTA, -ve  change sign if -ve
         8F   3F Ø4 F8  BSTA, UN  pop stack into R1, R2
         92   C9 15     STRR, R1 High } put into */ table
         94   CA 14     STRR, R2 low  } (OPR 1)
         96   3B 48     BSTR, UN *  test stack
         98   9A Ø8     BCFR, -ve  skip if -ve
         9A   Ø4 □      LODI, RØ  with saved sign flag
         9C   24 8Ø     EORI, RØ, 8Ø  if +ve now -ve, -ve now +ve
         9E   C8 7B     STRR, RØ  put sign flag back
        5AØ   3B EB     BSTR, UN *  change sign
         A2   3B EC     BSTR, UN *  pop stack → R1, R2
         A4   C9 Ø5     STRR, R1 high } */ table (OPR 2)
         A6   CA Ø4     STRR, R2 low  }
         A8   17        RETC, UN
*/ TABLE 5A9  [H][L]    OPR 1 multiplier   }
         AB   [H][L]    OPR 2 multiplicand } */ TABLE
         AD   [H][L]    RSLT H             }
         A5   [H][L]    RSLT L             }
MULTIPLY→5B1  3B 55     BSTR, UN  Put stack into */table and
         B3   77 Ø8     PPSL, WC=1  adjust sign flag
         B5   2Ø        EOR2
         B6   C8 75     STRR, RØ } clear result area.
         B8   C8 74     STRR, RØ }
        58A   Ø7 1Ø     LODI, R3 16
loop     BC   75 Ø1     CPSL  C=Ø
         BE   Ø8 69     LODR, RØ   }
         CØ   5Ø        RRR, RØ    } rotate multiplier
         C1   C8 66     STRR, RØ   } right into carry.
         C3   Ø8 65     LODR, RØ   }
         C5   5Ø        RRR, RØ    }
         C6   C8 62     STRR, RØ   }
         C8   2Ø        EOR2
         C9   DØ        RRL, RØ  to test carry
         CA   18 ØC     BCTR, Ø'  rotate product
         CC   Ø8 5E     LODR, RØ   }
         CE   88 5E     ADDR, RØ   } add multiplicand
         DØ   C8 5C     STRR, RØ   } to product.
         D2   Ø8 57     LODR, RØ   }
         D4   88 57     ADDR, RØ   }
         D6   C8 55     STRR, RØ   }
         D8   Ø5 FC     LODI, R1 -4
$7       DA   ØD 64 B1  LODA, RØ   } Rotate product
         DD   5Ø        RRR, RØ    }
```

```
                DE  CD 64 B1  STRA, RØ
                E1  D9 77     BIRR, R1 $7
                E3  FB 57     BDRR, R3 ' loop
                E5  Ø9 49     LODR, R1  low
                E7  ØA 46     LODR, R2  high
$push answer    E9  75 ØF     CPSL  WC=Ø, C=Ø, COM=Ø, OVF=Ø
                EB  3F Ø5 1C  BSTA, UN  Push R1, R2 onto stack
                EE  ØC Ø5 9B  LODA, RØ  sign flag
                F1  3E Ø5 26  BSTA, -VE if sign flag -ve then
                F4  17        RETC, UN                change sign
                                                      of answer
                      DIVIDE
DIVIDE →        5F5 3F Ø5 88  BSTA, UN  */ subroutine
                F8  ØC Ø5 AA  LODA, RØ  DVSR
                FB  1C Ø8 C4  BCTA "Ø"  error  cant ÷ by Ø
                FE  77 ØE     PPSL  WC=1, OVF=1, COM=1
                6ØØ Ø5 ØØ     LODI, R1
                Ø2  Ø7 11     LODI, R3
                Ø4  75 Ø1     CPSL  C=Ø
                Ø6  D1        RRL, R1
                Ø7  B5 Ø1     TPSL, C
                Ø9  18 Ø4     BCTR, subt
                ØB  E9 EC     COMR, R1*
                6ØD 1A Ø6     BCTR, true
                ØF  77 Ø1     PPSL  C=1
                11  A9 E6     SUBR, R1 *
                13  77 Ø1     PPSL, C
                15  Ø6 Ø2     LODI, R2
                17  ØE 45 AB  LODA,
                1A  DØ        RRL, RØ
                1B  CE 65 AB  STRA, RØ.
                1E  5A 77     BRNR
                2Ø  FB 64     BDRR      loop.
                22  ØD Ø5 AC  LODA, R1  low
                25  ØE Ø5 AB  LODA, R2  high
                28  1F Ø5 E9  BCTA, UN  to push answer
                      PRINT TOP OF STACK and SP+2
Print                                     check sign of stack
stack →         62B ØC 55 23  LODA, RØ * SP
                2E  9A ØB     BCFR, -VE $5   if want a +ve no. to be
                                             preceded by space change
                                             jmp from ØB to Ø7
                3Ø  3F Ø5 26  BSTA, UN  change sign
                33  Ø4 2D     LODI, RØ '-'
                35  1B Ø2     BCTR, UN $9
                37  Ø4 2Ø     LODI, RØ  space
$9              39  3B AA     BSTR, UN * COUT
$5              3B  Ø4 ØA     LODI, RØ  data index
                3D  C8 Ø3     STRR, RØ  save for loop
                3F  C8 2D     STRR, RØ  set print flag to non-zero
                41  Ø7 □      LODI, R3  data index
                43  ØF 46 7D  LODA, RØ, R3-
                46  C1        STR2, R1  low byte of power of 10
                47  ØF 46 7D  LODA, RØ  R3-
                4A  C2        STR2, R2
                4B  CB 75     STRR, R3  save R3 in data index
                4D  Ø7 ØØ     LODI, R3 Ø  set initial subtract
                                          count to Ø.
                4F  3F Ø5 1C  BSTA, UN push R1, R2
                52  3F Ø5 44  BSTA, UN  subtract
                55  ØC 85 23  LODA, RØ  * SP
                58  1A Ø2     BCTR, -VE  $adjust
                5A  DB 73     BIRR, R3  loop subtract count
$adjust         5C  3F Ø5 3C  BSTA, UN  push R1 & R2 & add
                5F  Ø3        LODZ, R3  to test R3
                6Ø  18 15     BCTR, Ø  $check print
                62  24 3Ø     EORI, RØ  put on ASCII bias.
                64  3F Ø2 B4  BSTA, UN  COUT
                67  C8 Ø5     STRR, RØ  RØ=Ø after return from COUT
                                        clear print flag
```

```
669  Ø9 57     LODR, R1   data index
6B   98 54     BCFR, `Ø'  $loop
6D   Ø4 □      LODI, RØ   print flag
6F   98 Ø3     BCFR, `Ø'
71   1F Ø4 FA  BCTA, UN   SP+2 & Return
74   2Ø        EOR2, RØ
75   1B 6B     BCTR, UN   print
77   Ø8 75     LODR, RØ   print flag
79   18 67     BCTR, `Ø'  print
7B   1B 6C     BCTR, UN
7D   ØØ Ø1     1 DATA
7F   ØØ ØA     1Ø
81   ØØ 64     1ØØ
83   Ø3 E8     1ØØØ
85   27 1Ø     1ØØØØ
     MULTIPLY  STACK BY 1Ø THEN ADD NEW DIGIT
Decimal to binary → 687  C9 ØA   STRA, R1   to save R1
89   Ø5 ØA     LODI, R1   ØA  } 1Ø
8B   Ø6 ØØ     LODI, R2   ØØ  }
8D   3B 9Ø     BSTR, UN   *  push R1 & R2
8F   3F Ø5 B1  BSTA, UN   Multiply
92   Ø5 □      LODI, R1   (saved R1)
94   45 ØF     ANDI, R1   strip off ASCII bias
96   Ø6 ØØ     LODI, R2
98   1B C3     BCTR, UN   * push R1, R2, add, return.
     PUSH FIRST DIGIT ONTO STACK
Push digit 69A  45 ØF     ANDI, R1   strip off ASCII bias
Push number 9C  Ø6 ØØ     LODI, R2   Ø
9E   1F Ø5 1C  BCTA, UN   push R1, R2 & return
     INPUT & ECHO & TEST   ESC → editor
Input, Echo, Test → 6A1  3F Ø2 86  BSTA, UN   CHIN
A4   C1        STRZ, R1
A5   E4 1B     COMI, RØ   ESC
A7   1C Ø8 EF  BCTA, true  ESC → editor
AA   BB AØ     ZBSR, *  COUT  (ECHO)
AC   1B A2     BCTR, UN *  test R1 & return
     INPUT A   NUMBER ONTO STACK
NUMBER INPUT → 6AE  3B 71     BSTR, UN   get 1st digit
BØ   9A 14     BCFR, -ve  if not digit error $?
B2   3B 66     BSTR, UN   push 1st digit
B4   3B 6B     BSTR, UN   get next digit
B6   9A Ø4     BCFR, -ve  if not digit jmp to SP+2 &?
6B8  3B 4D     BSTR, UN   decimal → binary
BA   1B 78     BCTR, UN   digit loop
Bc   E5 23     COMI, R1   "#" change sign
BE   18 BA     BCTR,  true *  to change sign & return
CØ   E5 ØD     COMI, R1   'CR'
C2   14        RETC, true
C3   3F Ø4 FA  BSTA, UN   SP+2  Restore stack [& made error]
$?   C6 Ø4 3F  LODI, RØ   '?'  error made i.e. input was not number, CR, ESC
C8   BB AØ     ZBSR, *  COUT
CA   1B 62     BCTR, UN   Input number. (Try again)
     PC+1 + load + TEST R1
PC+1+load +Test R1 → 6CC  3F Ø4 98  BSTA, UN   PC+1+load
CF   1F Ø5 6C  BCTA, UN   Test R1 and Return
     EVALUATE AND ERROR
6D2  3B Ø4     BSTR, UN   to evaluator and SP check
D4   1C Ø8 C4  BCTA, `='  if the pre SP = post SP then
D7   17        RETC       means nothing pushed on stack ∴ error
     EVALUATOR AND STACK POINTER CHECK
6D8  Øc Ø5 24  LODA, RØ   stack pointer
DB   CC Ø7 2C  STRA, RØ   save pre SP for later check
```

```
                    DE  3B 6C     BSTR, UN  PC+1+load + Test R1
                    6Eφ 9A φC     BCFR, -ve if non digit jump
                    E2  3F φ6 9A  BSTA, UN  push 1st digit
Decimal loop ⟶      E5  3B 65     BSTR, UN  PC+1 + load + Test R1
                    E7  9A φ5     BCFR, -ve if non digit jump
                    E9  3F φ6 87  BSTA, UN  decimal ⟶ binary
                    Ec  1B 77     BCTR, UN  decimal loop
                    EE  99 φ5     BCFR, +ve if non-letter jump
                    6Fφ 3F φ5 13  BSTA, UN  push vble onto stack
                    F3  1B 69     BCTR, UN  evaluator loop
                    F5  E5 23     COMI, R1 '#'  Change Sign
                    F7  98 φ5     BCFR, true  if not, test for +
                    F9  3F φ5 26  BSTA, UN  change sign of stacktop
                    FC  1B 6φ     BCTR, UN  evaluator loop
                    FE  E5 2B     COMI, R1 '+'
                    7φφ 98 φ5     BCFR, true if not test for '-'
                    φ2  3F φ5 3E  BSTA, UN  ADD
                    φ5  1B 57     BCTR, UN  evaluator loop
                    φ7  E5 2D     COMI, R1 '-'
                    φ9  98 φ5     BCFR, true
                    φB  3F φ5 44  BSTA, UN  SUBTRACT
                    7φE 1B 4E     BCTR, UN evaluator loop
                    1φ  E5 2A     COMI, R1 '*'
                    12  98 φ5     BCFR, true
                    14  3F φ5 B1  BSTA, UN  MULTIPLY
                    17  1B 45     BCTR, UN evaluator loop
                    19  E5 2F     COMI, R1 '/'
                    1B  98 φ5     BCFR, true
                    1D  3F φ5 F5  BSTA, UN  DIVIDE
                    2φ  1B 75     BCTR, UN evaluator loop via stepping stone
                    22  E5 21     COMI, R1 '!'  RANDOM
                    24  1F φA Ac   BCTA, UN to Patch
                    27  E5 5E     COMI, R1 '^'
                    29  18 6C     BCTR, true  to evaluator loop via SS
                    2B  φ4 □      LODI, Rφ  with pre SP
                    2D  EC φ5 24  COMA, Rφ  with current SP
                    3φ  17        RETC, UN
                        COMMAND LOOP
Editor entry ⟶  731 φ4 64        LODI, Rφ   initial SP on entry
                    33  C8 F9     STRA, Rφ *  from editor
COMMAND LOOP ⇒  735 3F φ4 98     BSTA, UN  PC+1 & load
                    38  E5 4C     COMI, R1 'L'  LET
                    3A  98 16     BCFR, to test for IN
Let loop ⟶          3C  3B AD     BSTR, UN * evaluate & error
                    3E  3B φ4     BSTR, UN  to right side handler
                    4φ  18 7A     BCTR,  true if comma let loop
                    42  1B 71     BCTR, UN cmd loop
                        Right side of expression handler
                    744 3F φ6 CC  BSTA, UN  PC+1 & load & test
                    47  9D φ8 C4  BCFA, +ve  not letter ⟶ error
                    4A  3F φ4 E7  BSTA, UN  pop stack into vble table
                    4D  3B E7     BSTR, UN * PC+1 & load
                    4F  E5 2C     COMI, R1 ',' comma   test for comma so that already done when return
                    51  17        RETC, UN
                    752 E5 49     COMI, R1 'I'  IN
                    54  98 φD     BCFR, to test for For
In loop ⟶           56  φ4 3F     LODI, Rφ '?'
                    58  BB Aφ     ZBSR * COUT
                    5A  3F φ6 AE  BSTA, UN input no.
                    5D  3B 65     BSTR, UN  right side handler
                    5F  19 75     BCTR, true  if comma in loop
                    61  1B 52     BCTR, UN  cmd loop.
```

```
                763  E5 46    COMI,R1 'F'          FOR
                 65  98 ϕA    BCFR, to test for Gosub
                 67  3F ϕ4 E1 BSTA, UN push 'PC' onto stack
                 6A  3F ϕ6 D2 BSTA, UN evaluate & error
                 6D  3B 55    BSTR, UN right side handler
                 6F  1B 44    BCTR, UN cmd loop
                771  E5 53    COMI,R1 'S'          GOSUB
                 73  98 ϕ4    BCFR, to test for Goto
                 75  3B F1    BSTA, UN * Push PC onto stack
                 77  1B ϕ4    BCTR, UN into GoTo
                779  E5 47    COMI,R1 'G'          GOTO
                 7B  98 3C    BCFR
                 7D  3B EC    BSTR, UN * evaluate & error
                 7F  E5 3E    COMI,R1 '>'          Relative jump forward
                 81  98 21    BCFR, to test for <
                 83  3B 11    BSTA, UN to get to line $>
                 85  1B 68    BCTR, UN to cmd loop
                     Get to Line   used by editor & interpreter
editor entry →   787 3B D2    BSTR, UN * input line no.
$sub entry →      89 ϕ5 41    LODI, R1 'A'
                 8B  3F ϕ4 E7 BSTA, UN pop stack into vble table
                 8E  ϕ5 41    LODI, R1 'A'
                 9ϕ  3F ϕ5 13 BSTA, UN push stack & vble
Normal GoTo →    93  3F ϕ4 BA BSTA, UN reset 'PC'
$2 →             96  3F ϕ4 CA BSTA, UN PC+1 till CR
loop till stack=ϕ 99 3F ϕ5 38 BSTA, UN stack-1
                 9C  3F ϕ5 6ϕ BSTA, UN test stack
                 9F  98 75    BCFR, 'ϕ' loop till stack=0 $>
                 A1  1F ϕ4 FA BCTA, UN SP+2 & return
                7A4  E5 3C    COMI,R1 'L'          Relative Jump Backward
                 A6  98 ϕD    BCFR
loop till -ve →  A8  3F ϕ4 D5 BSTA, UN PC-1 loop till CR
                 AB  3B ED    BSTR, UN * stack-1
                 AD  3B EE    BSTR, UN * test stack
                 AF  9A 77    BCFR, -ve loop till stack -ve
SP+2 then Cmd loop → 7B1 3B EF BSTR, UN * SP+2
                 B3  1B 5ϕ    BCTR, UN to cmd loop
                7B5  3B 5C    BSTR, UN Normal Go To ie Absolute
                 B7  1B 4C    BCTR, UN cmd loop
                7B9  E5 4E    COMI,R1 'N'          NEXT
                 BB  98 24    BCFR to test if not N
                 BD  3F ϕ6 CC BSTA, UN PC+1 & load & test
                7Cϕ  9D ϕ8 C4 BCFA, +ve error if non vble
                 C3  C9 ϕD    STRR, R1 save R1 (ie vble letter)
                 C5  3B CA    BSTR, UN * push vble onto stack
                 C7  3B D4    BSTR, UN * test stack
                 C9  18 66    BCTR, 'ϕ' to SP+2 & cmd loop
                 CB  3B CD    BSTR, UN * stack-1
                 CD  3B CE    BSTR, UN * test stack.
                 CF  98 ϕ6    BCFR, 'ϕ' $1 if ϕ then For/Next loop finished
                7D1  ϕ5 □     LODI, R1 saved R1
                 D3  3B 85    BSTR, UN * pop stack into vble table
                 D5  1B 5A    BCTR, UN SP+2 + cmd loop
$1               D7  ϕ9 79    LODR, R1 saved R1
                 D9  3F ϕ4 E7 BSTA, UN pop stack into vble table
                 DC  3F ϕ4 C6 BSTA, UN put stack into 'PC' loop till CR
                 DF  1B 52    BCTR UN cmd loop
                7E1  E5 54    COMI, R1 'T'         TEST (IF)
                 E3  98 3E    BCFR  NB too far for rel jump ∴ land short and load R3 to get to next instruction
IF loop          E5  3B 83    BSTR, UN * evaluator & error
                 E7  C9 ϕ9    STRR, R1 save relation
                 E9  3F ϕ6 D2 BSTA, UN evaluator & error
                 EC  C9 11    STRR, R1 save terminator
                 EE  3F ϕ5 44 BSTA, UN SUBTRACT
```

```
                7F1 | ϕ5 □      | LODI, R1  saved relation
                 F3 | E5 3D     | COMI, R1 '='   equals
                 F5 | 98 ϕF     | BCFR
                 F7 | 3B A2     | BSTR, UN * test top of stack
                 F9 | 98 24     | BCFR, ϕ' to fail if not zero
continue →       FB | 3F ϕ4 FA  | BSTA, UN SP+2  clean up stack
                 FE | ϕ5 □      | LODI, R1  terminator
                8ϕϕ | E5 2C     | COMI, R1 ',' comma.
                ϕ2  | 18 61     | BCTR, true  IF ie keep going
                ϕ4  | 1B 59     | BCTR, UN to cmd loop
                ϕ6  | E5 3C     | COMI, R1 '<'   less than
                ϕ8  | 98 ϕ4     | BCFR
                ϕA  | 3B 8F     | BSTR, UN * test top of stack
                ϕC  | 1A 6D     | BCTR, -ve continue  if fail then ripple on down to fail.
                ϕE  | E5 3E     | COMI, R1 '>'   greater than
               81ϕ  | 98 ϕ4     | BCFR
                12  | 3B 87     | BSTR, UN * test top of stack.
               814  | 19 65     | BCTR, +ve continue
                16  | E5 3A     | COMI, R1 ':'   not equal
                18  | 98 ϕ5     | BCFR fail  if not go to fail
                1A  | 3F ϕ5 6ϕ  | BSTA, UN test stack
                1D  | 98 5C     | BCFR, '=' continue
Fail →          1F  | 3F ϕ4 CA  | BSTA, UN PC+1 till CR
ie skip current 22  | 1F ϕ7 B1  | BCTA, UN SP+2 & cmd loop
line
               825  | E5 27     | COMI, R1 '↑'  PRINT no CR, LF
                27  | 18 ϕ6     | BCTR, true  jmp into Print less CR, LF
                29  | E5 5ϕ     | COMI, R1 'P'  PRINT CR LF
                2B  | 98 28     | BCFR  to Memory
                2D  | BB A5     | ZBSR * CR+LF
Print            2F | 3F ϕ6 D8  | BSTA, UN evaluator & SP check
loop →           32 | 18 ϕ7     | BCTR, '='
                 34 | C9 ϕ4     | STRR, R1  save R1
                 36 | 3F ϕ6 2B  | BSTA, UN  Print stack
                 39 | ϕ5 □      | LODI, R1  saved R1
                 3B | E5 22     | COMI, R1 '"'
                 3D | 98 ϕC     | BCFR, true
String           3F | 3F ϕ6 CC  | BSTA, UN  PC+1 & load & test
loop →           42 | E5 22     | COMI, R1 '"'
                 44 | 18 69     | BCTR, true  Print loop
                 46 | ϕ1        | LODZ, R1
                 47 | BB Aϕ     | ZBSR, * COUT
                 49 | 1B 74     | BCTR, UN  string loop
                 4B | E5 2C     | COMI, R1 ','
                 4D | 98 B1     | BCFR, true  * to cmd loop
                 4F | ϕ4 2ϕ     | LODI, Rϕ SPACE  comma => space
                 51 | BB Aϕ     | ZBSR, * COUT
                 53 | 1B 5A     | BCTR, UN  Print loop
                855 | E5 4D     | COMI, R1 'M'  MEMORY (PEEK OR POKE)
                 57 | 98 34     | BCFR
                 59 | 3B E5     | BSTR, UN * PC+1 + load + test
                 5B | 9D ϕ8 C4  | BCFA, +ve  error non vble
                 5E | C9 1B     | STRR, R1  save R1 (letter)
                 6ϕ | 3B DE     | BSTR, UN * PC+1 + load + test
                 62 | C9 ϕB     | STRR, R1  save R1 (arrow)
                 64 | 3F ϕ6 D2  | BSTA, UN evaluator & error
                 67 | 3F ϕ4 F8  | BSTA, UN pop stack into R1 & R2
                 6A | C9 ϕ9     | STRR, R1  high
                 6C | CA ϕ8     | STRR, R2  low
               86E  | ϕ5 □      | LODI, R1  saved R1 (arrow)
                7ϕ  | E5 3C     | COMI, R1 '<'  READ FROM MEMORY
                72  | 98 ϕE     | BCFR  if not
                74  | ϕD □ □    | LODA, R1
                77  | 3F ϕ6 9C  | BSTA, UN push R1, R2 onto stack R2=ϕ
                7A  | ϕ5 □      | LODI, R1 vble
                7C  | 3F ϕ4 E7  | BSTA, UN pop stack into vble table
```

```
                7F | 1F φ7 35 | BCTA, UN   to cmd loop.
                   |  Implied '>' ie if not < then must be>   WRITE
              882 | φ9 77      | LODR, R1  saved vble
               84 | 3F φ5 13   | BSTA, UN  push vble onto stack
               87 | 3B DF      | BSTR, UN  * pop stack into R1, R2
               89 | CA EA      | STRR, R2  * store LSB at address
               8B | 1B F3      | BCTR, UN  * to cmd loop.
              88D | E5 43      | COMI, R1 'C'          CALL
To return      8F | 98 φB      | BCFR  to return
to μBASIC      91 | 3B D2      | BSTR, UN  * evalcator + error
from your      93 | 3B D3      | BSTR, UN  * pop stack into R1, R2
machine        95 | C9 φ3      | STRR, R1  high
routine        97 | CA φ2      | STRR, R2  low
do a           99 | 1F □ □     | BCTA, UN  Jump to specified location
BCTA to
cmd loop:
1Fφ735
              89C | E5 52      | COMI, R1 'R'          RETURN
               9E | 98 φ5      | BCFR  if not
               Aφ | 3F φ4 C2   | BSTA, UN  pop stack into 'Pc'  then loop till CR.
               A3 | 1B DB      | BCTR, UN  cmd loop
              8A5 | E5 45      | COMI, R1 'E'          END
               A7 | 1C φ8 D3   | BCTA, true  Print line no. then editor
              8AA | E5 24      | COMI, R1 '$'          REMARK
               Ac | 9C φA BB   | BCFA, to cmd Patch if not '$'
$loop          AF | 3F φ4 98   | BSTA, UN  Pc+1 + load
               B2 | E5 24      | COMI, R1 '$'
               B4 | 98 79      | BCFR, true $loop   keep looping till see another '$'
               B6 | 1B C8 Cφ   | BCTR, UN  cmd loop     NOP
                  |  ERROR ROUTINES
SP error →    8B9 | φ5 53      | LODI, R1 'S'  if stack popped too far
               BB | 1B φ9      | BCTR, UN $error
STX/ETX        BD | 3F φ4 98   | BSTA, UN Pc+1  if trying to jump out
error →        8Cφ | φ5 54      | LODI, R1 'T'   of program
               C2 | 1B φ2      | BCTR, UN $error   In case had jumped out because of STX
line error →   C4 | φ5 4C      | LODI, R1 'L'
$error         C6 | φ4 64      | LODI, Rφ
               C8 | CC φ5 24   | STRA, Rφ   reset stack pointer
               CB | 74 φF      | CPSU  reset 2650 stack in case jumped out of subroutine
               CD | BB A5      | ZBSR, * CR+LF
               CF | φ4 3F      | LODI, Rφ '?'
              8D1 | BB Aφ      | ZBSR, * COUT  print question mark
               D3 | BB A5      | ZBSR, *   CR+LF
               D5 | φ1         | LODZ, R1  load error letter
               D6 | BB Aφ      | ZBSR, * COUT  print error letter
               D8 | C1         | STRZ, R1 (after COUT Rφ = φ)
               D9 | 3F φ6 9C   | BSTA, UN  push φ onto stack
loop →         Dc | 3F φ4 Ac   | BSTA, UN  Pc-1
               DF | E5 φ2      | COMI, R1 'STX'
              8E1 | 18 φ9      | BCTR, true $print
               E3 | E5 φD      | COMI, R1 'CR'
               E5 | 98 75      | BCFR, true loop
               E7 | 3F φ5 32   | BSTA, UN  stack+1
               EA | 1B 7φ      | BCTR, UN  loop
$print         EC | 3F φ6 2B   | BSTA, UN -Print top of stack then run down into editor
                  |  EDITOR
START UP →    8EF | 74 φF      | CPSU   } initial 265φ   ←φ8EF Entry point from PIPBUG
               F1 | 75 FF      | CPSL   }
               F3 | φ4 64      | LODI, Rφ   } initial stack pointer
               F5 | C8 D2      | STRR, Rφ *  }
Editor loop    F7 | BB A5      | ZBSR, *  CR+LF
               F9 | φ4 3E      | LODI, Rφ ">"   Prompt character
               FB | BB Aφ      | ZBSR, * COUT   Prompt
               FD | 3F φ6 A1   | BSTA, UN  to input and echo
              9φφ | E5 4E      | COMI, R1 'N'  NEW
               φ2 | 98 φF      | BCFR   to list if not New
               φ4 | φ4 φ2      | LODI, R1 'STX'
```

From the current line where error occurred, go back to STX counting CR's on the way. This will give line number

```
        ϕ6  CC ϕB ϕϕ   STRA, R1
        ϕ9  ϕ4 ϕD      LODI, R1  'CR'
        ϕB  CC ϕB ϕ1   STRA, R1
        ϕE  ϕ4 ϕ3      LODI, R1  'ETX'
        1ϕ  CC ϕB ϕ2   STRA, R1
```

set up start and end markers. Text exists between STX and first ETX

```
       913  E5 4C      COMI,R1 'L'      LIST

        15  98 2A      BCFR   to enter if not list
        17  3F ϕ7 87    BSTA, UN   get to line
$list
loop →  1A  3F ϕ9 92    BSTA, UN   print line number
$line
loop →  1D  3F ϕ4 98    BSTA, UN   PC+1 + load (get text char.)
      92ϕ   E5 ϕ3       COMI, R1  'ETX'
        22  18 53       BCTR, true - to editor if reached ETX
        24  E5 ϕD       COMI, R1  'CR'
        26  98 ϕF       BCFR, true (if not CR then print text character)
        28  ϕ5 41       LODI, R1  'A' - line number.
        2A  3F ϕ5 13    BSTA, UN  push stack with line no.
        2D  3F ϕ5 32    BSTA, UN  stack+1
      93ϕ   ϕ5 41       LODI, R1  'A'
        32  3F ϕ4 E7    BSTA, UN  pop stack
        35  1B 63       BCTR, UN  $list loop (print new line no.)
        37  ϕ1          LODZ, R1  - contains text character
        38  BB Aϕ        ZBSR, *  COUT - print text character
        3A  B4 8ϕ       TPSU      test for key press
        3C  18 5F       BCTR  true $line loop - if no key press
        3E  1F ϕ8 F7    BCTA, UN  editor if key press
       941  E5 45       COMI, R1 'E'     ENTER
        43  98 ϕA       BCFR     if not E test for Alter
        45  2ϕ          EORZ  clear Rϕ
        46  CC ϕ9 C9    STRA, Rϕ   clear flag 1 to ϕ
$jump   49  CD ϕ9 FF    STRA, R1   set flag 2 to non-zero
        4C  1F ϕ9 Aϕ    BCTA, UN   (R1 contains 'E')
       94F  E5 41       COMI, R1 'A'     ALTER
        51  98 ϕ4       BCFR   if not A test for Insert
        53  C9 F2       STRR, R1 *   set flag 1 to non-zero
        55  1B 72       BCTR, UN  $jump
       957  E5 49       COMI, R1 'I'     INSERT
        59  98 ϕ7       BCFR   if not test for G
        5B  2ϕ          EORZ
        5C  C5 EC       STRR, Rϕ *   clear flag② to ϕ
        5E  C8 E7       STRR, Rϕ *   clear flag 1 to ϕ
        6ϕ  1B EB       BCTR, UN *   to A.E.I.
       962  E5 47       COMI, R1 'G'     GO
        64  98 ϕ6       BCFR,  if not test for P
        66  3F ϕ7 87    BSTA, UN   Get to line
        69  1F ϕ7 31    BCTA, UN  to Command loop
       96C  E5 5ϕ       COMI, R1 'P'     PIPBUG
        6E  1C ϕϕ ϕϕ    BCTA, true  PIPBUG.
       971  E5 58       COMI, R1 'X'   END OF TEXT
        73  9C ϕ8 F3    BCFA, true  to editor
        76  3B ϕF       BSTR, UN  to find ETX
        78  3F ϕ4 E1    BSTA, UN  push 'PC' onto stack
        7B  3F ϕ4 F8    BSTA, UN  pop stack into R1, R2
        7E  3F ϕ2 69    BSTA, UN  BOUT
        81  ϕ2          LODZ, R2
        82  C1          STRZ, R1
        83  3B FA       BSTR, UN * BOUT
        85  1B ED       BCTA, UN * editor loop
                FIND END OF TEXT
       987  3F ϕ4 BA    BSTA, UN  reset 'PC'
        8A  3F ϕ4 98    BSTA, UN  PC+1 & load
        8D  E5 ϕ3       COMI, R1  ETX
        8F  98 79       BCFR, true
        91  17          RETC
```

vble A is used to store line number

Editor uses vble A to store line no. vble B to save 'PC'

	992	ϕ541	LODI, R1 'A'
	94	3Eϕ513	BSTA, UN push stack
	97	BBA5	ZBSR * CR+LF
	99	3Fϕ628	BSTA, UN print stack
	9C	ϕ4 2ϕ	LODI, Rϕ space
	9E	9BAϕ	ZBCR * COUT & Return
		ALTER/ENTER/INSERT	direction flow determined by flags.
	9Aϕ	3Fϕ787	BSTA, UN Get to line
$ALE reentry →	A3	ϕ7$\phi\phi$	LODI, R3 'ϕ' clear buffer index
	A5	3Fϕ498	BSTA, UN PC+1 & load
	A8	ϕ1	LODZ, R1
	A9	CF24$\phi\phi$	STRA, Rϕ, R3+ into character buff
	AC	E5ϕD	COMI, R1 CR
	AE	18ϕ6	BCTR, true $2
	Bϕ	E5ϕ3	COMI, R1 ETX
	B2	9871	BCFR, true if not loop
	B4	ϕ7$\phi\phi$	LODI, R3, ϕ if ETX clear index
	9B6	3Fϕ4D5	BSTA, UN PC-1 & load till CR
	B9	3Fϕ4E1	BSTA, UN 'PC' → stack
	BC	ϕ542	LODI, R1 'B'
	BE	3Fϕ4E7	BSTA, UN pop stack into vble table
	C1	CFϕAϕ6	STRA, R3 save buffer index
$print	9C4	3B4C	BSTR, UN print line number
	C6	ϕ7$\phi\phi$	LODI, R3 clear buffer index
	C8	ϕ4☐	LODI, Rϕ flag ① flag=ϕ ⇒ enter ∴ dont print buffer; flag=1 ⇒ alter ∴ print buffer
	CA	1811	BCTR, 'ϕ' TCA
print buff.	9CC	ϕF24$\phi\phi$	LODA, Rϕ, R3, +
	CF	E4ϕ3	COMI, Rϕ, ETX
	D1	18ϕ8	BCTR, true print exit
	D3	E4ϕD	COMI, Rϕ CR
	D5	18ϕ4	BCTR, true print exit
	D7	BBAϕ	ZBSR, * COUT
	D9	1B71	BCTR, UN print buffer
print exit	DB	FB$\phi\phi$	BDRR, R3, R3-1
TCA⇒	9DD	3Fϕ6A1	BSTA, UN input & echo, ESC → editor
	Eϕ	E5ϕ8	COMI, R1 'BS'
	E2	98ϕ7	BCFR, true
	E4	FB$\phi\phi$	BDRR, R3, R3-1
	E6	ϕ3	LODZ, R3 to test R3
	E7	9A74	BCFR, -ve to TCA
	E9	1B59	BCTR, UN $ print
	EB	CDϕA74	STRA, R1 in flag to test latter for DEL
	EE	E57F	COMI, R1 DEL
	Fϕ	98ϕ4	BCFR, true $7
	F2	ϕ7$\phi\phi$	LODI, R3, ϕ Delete
	F4	1Bϕ8	BCTR, UN exit
	F6	ϕ1	LODZ, R1
	F7	CF24$\phi\phi$	STRA, Rϕ, R3, + in line buffer
	FA	E5ϕD	COMI, 'CR'
	FC	985F	BCFR, true TCA (loop)
exit⇒	9FE	ϕ4☐	LODI, Rϕ flag ② =ϕ ⇒ insert ≠ϕ ⇒ alter/center
	A$\phi\phi$	98ϕ3	BCFR, 'ϕ'
	ϕ2	2ϕ	EORZ
	ϕ3	C8ϕ1	STRR, Rϕ set old index to ϕ if insert
	ϕ5	A7☐	SUBI, R3 R3 - old index, WC=ϕ
	ϕ7	1Eϕ4AC	BCTA, -ve if -ve then new line shorter

NEW LINE SAME LENGTH OR LONGER THAN OLD LINE

```
          A0A  3F 09 87   BSTA, UN   find ETX
          0D   0C 84 A9   LODA, R0   * 'PC'
          10   CF E4 A9   STRA, R0   * 'PC', R3 indexed
          13   3F 04 AC   BSTA, UN   PC-1
          16   CB 14      STRR, R3   save R3
          18   05 42      LODI, R1   'B'
          1A   3F 05 13   BSTA, UN   push saved 'PC' onto stack
          1D   3F 04 E1   BSTA, UN   'PC' --> stack
          20   3F 05 44   BSTA, UN   SUBTRACT
          23   3F 05 60   BSTA, UN   test stack
          26   C8 06      STAR, R0   save test answer
          28   3F 04 FA   BSTA, UN   SP+2
          2B   07 [ ]     LODI, R3   saved R3
          2D   04 [ ]     LODI, R0   saved test
          2F   98 5C      BCFR, '0'  $2 when saved 'PC'=current'PC'
                                     continue
Put in
new line→ A31  07 00      LODI, R3   clear buffer index
loop $3   33   3B 98      BSTR, UN   * PC+1 + load.
          35   0F 24 00   LODA, R0   R3,+ buffer
          38   CC 84 A9   STRA, R0   * PC
          3B   E4 0D      COMI, R0   'CR'
          3D   98 74      BCFR, true loop $3
Next
line  →   A3F  05 41      LODI, R1   'A'
          41   3B D8      BSTR, UN   * push stack
          43   3F 05 32   BSTA, UN   stack+1
          46   3F 07 89   BSTA, UN   to GetTo Line ($subentry)
          49   1F 09 A3   BCTA, UN   $ re-entry into A1E

          NEW LINE SHORTER THAN OLD LINE

          A4C  3F 04 CA   BSTA, UN   PC+1 till CR
          4F   3F 04 98   BSTA, UN   PC+1 & load
          52   C9 0A      STRA,      save R1
          54   3B C8      BSTR, UN   * push PC --> stack
          56   03         LODZ, R3
          57   C1         STRZ, R1   low
          58   06 FF      LODI, R2   high
          5A   3F 05 3C   BSTA, UN   push R1, R2 & add.
          5D   05 [ ]     LODI, R1   saved R1
          5F   CD 84 62   STRA, R1   * stack

          62   3B C5      BSTR, UN   * SP+2
          64   E5 03      COMI, R1   ETX
          66   98 67      BCFR,      loop
          68   05 42      LODI, R1   'B'
          6A   3F 05 13   BSTA, UN   push saved PC
          6D   3F 05 38   BSTA, UN   stack-1
          70   3F 04 C2   BSTA, UN   pop top of stack into 'PC'
          A73  04 [ ]     LODI, R0   see if doing Delete
          75   E4 7F      COMI, R0   Del
          77   18 50      BCTR, true to $re-entry of A1E via jump
          79   1F 0A 31   BCTA, UN   put in new line
```

RANDOM

Random	A7D	ϕ1	LODZ, R1 (R1 will have ! in it)
	7E	BB Aϕ	ZBSR, * COUT
	8ϕ	ϕF 85 23	LODA, R3 high byte
	83	3F ϕ4 F4	BSTA, UN SP+1
	86	ϕE 85 23	LODA, R2 low byte
reload	89	ϕ3	LODZ, R3
	8A	C1	STRZ, R1 high
	8B	ϕ2	LODZ, R2 low
	8C	F8 $\phi\phi$	BDRR, Rϕ decrement (not jumping anywhere)
	8E	E4 FF	COMI, Rϕ
	9ϕ	98 ϕ6	BCFR to sense
	92	E5 $\phi\phi$	COMI, R1
	94	18 73	BCTR, 'ϕ' if zero then reload
	96	F9 $\phi\phi$	BDRR, R1 decrement (not jumping anywhere)
	98	B4 8ϕ	TPSU, look at sense
	9A	18 7ϕ	BCTR, loop (loop till key pressed)
	9C	CC 85 23	STRA, Rϕ low on stack
	9F	3F ϕ5 ϕ4	BSTA, UN SP-1
	A2	CD 85 23	STRA, R1 high on stack
	A5	ϕ4 2A	LODI, Rϕ '*'
	A7	BB Aϕ	ZBSR, * COUT acknowledge keypress
	A9	1F ϕ6 DE	BCTA, UN back to evaluator

EXCHANGE BYTES ON TOP OF STACK MSB ↔ LSB

Patch %	AAC	18 4F	BCTR, true, to RND (already tested for !)
	AE	E5 25	COMI, R1 '%' test for byte swap.
	Bϕ	9C ϕ7 27	BCFA, true if not go back to evaluate at ^
	B3	3F ϕ4 F8	BSTA, UN pop stack into R1, R2
	B6	3F ϕ5 1C	BSTA, UN push R1, R2 onto stack (HIGH ↔ LOW)
	B9	1B EF	BCTR, UN * to evaluate loop.

COMMAND LOOP EXTENSIONS

Patch	ABB	E5 41	COMI, R1 'A' ACCEPT
	BD	98 ϕA	BCFR, if not (accepts a single ASCII character into a vble)
	BF	3F ϕ6 A1	BSTA, UN input + echo
	C2	3F ϕ6 9C	BSTA, UN push character (in R1) onto stack
	C5	3B 96	BSTR, UN * PC+1+load (get vble)
	C7	1B 2ϕ	BCTR, UN into J&[to save stack into vble table & return to cmd loop.
	AC9	E5 4F	COMI, R1 'O' OUT
	CB	98 ϕB	BCFR, if not (prints the LSB of a vble as an ASCII character)
	CD	3B 8E	BSTR, UN * PC+1+load
	CF	3B 91	BSTR, UN * push vble
	D1	3B E1	BSTR, UN * pop stack into R1 & R2
	D3	ϕ2	LODZ, R2 ASCII character in R2 (LSB)
	D4	BB Aϕ	ZBSR, * COUT
	D6	1B 97	BCTR, UN * back to cmd loop
	AD8	E5 5D	COMI, R1 ']' INCREMENT
	DA	98 1ϕ	BCFR if not
	DC	3F ϕ4 98	BSTA, UN PC+1+load
	DF	C9 ϕ7	STRR, R1 save vble
	E1	3F ϕ5 13	BSTA, UN push vble onto stack
	E4	3F ϕ5 32	BSTA, UN stack+1
$finish	E7	ϕ5 ☐	LODI, R1 saved vble
	E9	3F ϕ4 E7	BSTA, UN pop stack into
	EC	E5 5B	COMI, R1 '[' D
	EE	9C ϕ7 35	BCFA, if not return to
	F1	3B EA	BSTR, UN * PC+1+load
	F3	C9 73	STRR, R1 save vble
	F5	3B EB	BSTR, UN * push st
	F7	3F ϕ5 38	BSTA, UN stack-
	FA	1B 6B	BCTR, UN $finish