PROGRAMS FOR THE COSMAC ELF

ELFISH

AN INTERPRETIVE DEVELOPMENT SYSTEM

for noncommercial use only

Oct. 20, 2010 PCM

PROGRAMS FOR THE COSMIC ELF _ "ELFISH"

An Interpretive Development System

by Paul Moews

List of Sections

1. Introduction ---------- 3
2. The Interpreter ---------- 6
   - Use of Memory ---------- 6
   - The Instruction Set ---------- 8
   - Summary of Instruction Set ---------- 10
   - A Simple Program in Elfish ---------- 13
3. The Editor-Assembler ---------- 16
   - Use of Editor ---------- 16
   - The Assembler ---------- 19
   - A Simple Program in Elfish (with Labels) ---------- 19

Program Listings

1. A Simple Program in Elfish (Assembled) ---------- 15
2. A Simple Program in Elfish (with Labels) ---------- 19
3. The Interpreter ---------- 22
4. The Editor-Assembler ---------- 25

Published February, 1982 by Paul C. Moews

Manufactured in the United States of America

Introduction

RCA's COSMAC 1802 microprocessor was designed to simplify the writing of interpreters. As an example, RCA's VIP computer is supplied with a 512-byte interpreter; the corresponding language is called CHIP-8. This language is surprisingly powerful, and quite complex programs, e.g. blackjack, lunar lander, etc., have been written using it. Further details may be found in an article by Joseph Weisbecker (An Easy Programming System in the December, 1978 BYTE, p. 108). RCA's CHIP-8 was designed for games and has a display routine with features that are not necessary for control applications. It also has a number of other drawbacks: the interpretive code is not relocatable, the address range of the interpreter is limited to 4K, the variables are only 8-bit, and there is no provision for displaying ASCII symbols. I have written a development system for the 1802 that I hope will remedy these defects and which will introduce ideas to make simple interpreters easier to use. My goal was to write a 1 K interpreter and a 1 K editor-assembler using the interpretive language. Both fit in ROM and are page relocatable.

The new interpretive language, ELFISH, is written for 4 K and larger Super Elf's and Elf II's and was developed on a 4K Super Elf. While similar to CHIP-8, the language is an attempt to improve on CHIP-8 by introducing a number of features that make it more versatile and easier to use. First the interpretive code is relocatable; this means that interpretive programs can be located anywhere in memory, or moved to a new position in memory without making any changes. The

relocatability of the interpretive code is done by using a base register and the whole address field of the 1802 chip is available to the interpreter by simply changing the value of the base register. The 8-bit CHIP-8 variables were replaced by 16-bit variables. These variables are a kind of image of the 1802's registers, and, as with the registers, there are some 8-bit instructions and some 16-bit instructions. Sixteen-bit variables are very useful with an 8-bit microprocessor as they can store memory locations, making it easier to write utility programs of all kinds. The instruction set is more powerful; it includes multiplication and division instructions and has the ability to display ASCII symbols. Like the VIP, the Elf has only a hex keyboard and the language is designed to accomodate the keyboard, that is, it is hexadecimally oriented. However the Elf has a latched keyboard while the VIP has a scanned keyboard. A latched keyboard has advantages over a scanned keyboard, and these advantages were made use of in writing the interpreter and its companion editor-assembler.

The other half of the package, a 1 K editor-assembler, was written in the new language. An editor greatly increases the productivity of a programmer, and I have tried to make this editor easy to use. The editor displays 4 lines of code together with their addresses, a choice of absolute or relative addresses is possible. Keyboard bytes are entered at the bottom of the display, and there is an indication of the mode of the editor at the lower left of the screen. Changes and corrections occur on the second line of the

display, the "current line". The editor will accept 9 commands: replace, insert, delete, go to, scan up, scan down, change address mode, execute, and assemble. Changing between these 9 commands or modes can be done at any time. The relative address mode simplifies hand assembly or the built in assembler can be used. To use the assembler, labels are inserted in front of subroutines and at all program entry points. Only interpretive jump and subroutine calls are assembled so the assembler is a partial one. Three passes are made by the assembler, the first two of which are interrupted by the occurrence of errors. The assembler halts if duplicate labels are found or if jumps or subroutine calls are found without a corresponding label. If the first two passes are successful the code is assembled in a third pass.

An advantage of this type of programming system is its inherent stability. The interpreter instructions do not effect memory locations aside from the variables and the buffers, even in an all RAM system you should experience few "crashes". Perhaps the biggest disadvantage is that the interpretive instructions do not serve as their own mnemonics, and in this the interpretive code is like machine code. However assigning mnemonics would require a much more complex assembler and a larger computer.

Interpreter Listing

An annotated hexadecimal dump of the interpreter is in Listing 1. All of the code is relatively straightforword although there are many "paths" through it for some of the instructions. If you are not familiar with the 1802, RCA's "User Manual for the 1802 Microprocessor" - MPM-201A - is well written, thoroughly explains all of the instructions, and is recommended.

Use of Memory

The editor-assembler followed by the interpreter go on any 8 contiguous memory pages, ROM or RAM. On my Super Elf I use either RAM pages 06 through 0D or ROM pages 84 through 8B. Two additional pages of RAM are required, one for the display (default display page is 0E) and one for buffers, variables and the stack (default is 0F).

The interpreter provides sixteen 16-bit variables and so 32 memory locations are necessary (on page 0F) to store these variables. As well the interpreter has save and restore instructions for the variables, and an additional 32 locations are required for the save locations. The assignment of the 64 variable locations is shown in the following table:

Variable Locations

Variable	Low	High	Save Low	Save High	Variable	Low	High	Save Low	Save High
0	E0	E1	C0	C1	8	F0	F1	D0	D1
1	E2	E3	C2	C3	9	F2	F3	D2	D3
2	E4	E5	C4	C5	A	F4	F5	D4	D5
3	E6	E7	C6	C7	B	F6	F7	D6	D7
4	E8	E9	C8	C9	C	F8	F9	D8	D9
5	EA	EB	CA	CB	D	FA	FB	DA	DB
6	EC	ED	CC	ED	E	FC	FD	DC	DD
7	EE	EF	CE	CF	F	FE	FF	DE	DF

Conversions and the displaying of symbols are carried out with the aid of three buffers. A 4-byte hexadecimal buffer is used to store the four hexadecimal digits that correspond to a variable, while a 5-byte decimal buffer stores the five decimal digits that correspond to a variable. The 5-byte display buffer is used to store a bit pattern for a symbol to be displayed, a display instruction then transfers this pattern to the display screen. The locations of these buffers, again on page 0F, are given in the following table.

Buffer Locations

Hexidecimal	AC AD AE AF
Decimal	B6 B7 B8 B9 BA
Display	EB BC BD BE BF

Finally there is a stack, also on page 0F, which extends downwards from location 9F. The interpreter does not check for stack overflow; if the stack extends below page 0F, it will begin to appear on the display page.

The instruction set

All Elfish instructions consist of four hex digits. The first hex digit determines the type of instruction, E is reserved for labels, and there are therefore 15 basic types of instruction. The next 3 hex digits are used in several different ways. They can be used to specify a memory location and as there are 3 hex digits available, any memory location from 000 to FFF can be specified. Memory locations are addressed relative to a base register. The base register is set to the address of the first location or entry point of a program, so that all references to memory are relative to the starting address of the program.

Elfish provides 16 two-byte variables, designated V0 through VF. In some cases it is necessary to refer to the high or low order byte of a variable, and this is done in a manner analogous to that used for the registers, i.e. V(7).1 refers to the high order byte of variable 7 and V(4).0 refers to the low order byte of variable 4. A single hex digit can be used to specify one of these variables. In some instructions the second most significant hex digit is used for this purpose leaving the last two hex digits to specify different instructions. In arithmetic operations, the two variables to be added etc. are specified by the second and third hex digits, leaving the last hex digit to designate the type of arithmetic operation to be carried out.

The Elf's display screen is a bit map; i.e. the 2048 bits on the display page are mapped as 2048 small squares on the display, 64 horizontal squares by 32 vertical squares. Each square is either black if its corresponding bit is 0, or white if its bit is 1. The language is designed for use with these low resolution graphics and the display instruction and the instructions which generate patterns for display are among the important instructions. There are a number of commands which create 5-byte display patterns in the display buffer. For example if variable V(5).0 was 0A, C556 would create a bit pattern for the symbol "A" in the display buffer, while C55B would create a bit pattern for the symbol "J" (ASCII symbol for 0A). The instruction CX56 creates display patterns for hex symbols while CX5B creates display patterns for ASCII symbols. The display instruction DXYN can be used to transfer the contents of the display buffer to any location on the display screen. To do this, VX would have to point to the display buffer, while V(Y).0 would be preset to the desired horizontal address on the screen (0 through 64) and V(Y).1 would be set to the vertical address (0 through 32). N indicates how many bytes to display and in the case of the display buffer it would be set to 5. Say we point VB to the display buffer and set V(A).0 = 00 and V(A).1 = 00, then when the instruction DBA5 was executed, the contents of the display buffer would appear in the upper left corner of the screen.

Perhaps the best way to understand how the language works is to introduce the instruction set and demonstrate its use with a simple program.

SUMMARY OF INTERPRETER INSTRUCTIONS

Most Significant Hex Digit	Result of Instruction
0 (MMM)	Do machine code sub. at relative address MMM
1 (MMM)	Go to relative address MMM
2 (MMM)	Do interpreter sub. at relative address MMM, 4000 instruction ends interpreter subroutines
3 (MMM)	Set V(0) to point to memory location MMM, relative address
4 (XZZ)	4000 is return from subroutine, 4FFE is erase display-plus a variety of keyboard instructions involving one variable, X, distinguised by ZZ
5 (XKK)	sets V(X).0 to KK + V(X).0
6 (XKK)	sets V(X).1 to KK + V(X).1
7 (XYZ)	8-bit instructions involving two variables V(X).0 and V(Y).0 or V(X).1 and V(Y).1
8 (XYZ)	eight 16-bit instructions as well as eight instructions involving V(X).0 and V(Y).1
9 (XY0), (XY1)	XY0 is V(X) (16 bit) = V(X).0 times V(Y).0 XY1 is V(X).0 = V(X).0/V(Y).0 with V(X).1 equal to the remainder
A (XKK)	V(X).0 = KK
B (XKK)	V(X).1 = KK
C (XZZ)	a variety of instructions referencing one variable V(X).0 or V(X).1, ZZ codes for type of instruction
D (XYN)	display instruction, V(X) (16 bit) is memory pointer for location to be displayed V(Y) is display pointer, V(Y).1 is vertical pointer and V(Y).0 is horizontal pointer N indicates number of bytes to display
E (ZZZ)	E instructions are ignored, used for labels
F (XZZ)	a variety of 16-bit instructions, referring to one variable, VX, ZZ codes for the different instructions.

The 4, 7, 8, C and F instructions are subdivided further and a table is given for each class of instruction.

SUMMARY OF 4 INSTRUCTIONS

Instruction	Result
4000	return from interpreter subroutine
4XD7	transfers contents of keyboard to V(X).0
4XD8	transfers contents of keyboard to V(X).1
4XDB	keys to V(X).0, waits for in on, off
4XDC	keys to V(X).1, waits for in on, off
4XE2	waits for in button, on then off
4XE5	skip if in pushed
4XE9	skip if in pushed, wait for in off
4XED	wait for in off
4FFE	erase the display

SUMMARY OF 7 AND 8 INSTRUCTIONS

7XYZ X(0),Y(0) Z	7XYZ X(1),Y(1) Z	8XYZ X(16),Y(16) Z	8XYZ X(0),Y(1) Z	Result
0	8	0	8	X = Y
1	9	1	9	X = X and Y
2	A	2	A	X = X xor Y
3	B	3	B	X = X + Y
4	C	4	C	X = X - Y
5	D	5	D	skip if X = Y
6	E	6	E	skip if X does not = Y
7	F	7	F	skip if X > Y

SUMMARY OF C INSTRUCTIONS

Instruction	Result
CX03	convert V(X).0 to decimal, result to decimal buffer
CX04	convert V(X).1 to decimal, result to decimal buffer
CX43	byte pointed to by V0 to V(X).0, V0 = V0 + 1
CX44	byte pointed to by V0 to V(X).1, V0 = V0 + 1
CX56	hex symbol for least significant part of V(X).0, result to display buffer
CX57	hex symbol for least significant part of V(X).1, result to display buffer
CX5B	ASCII symbol for V(X).0 to display buffer
CX5C	ASCII symbol for V(X).1 to display buffer
CX66	length of symbol in display buffer to V(X).0
CX67	length of symbol in display buffer to V(X).1
CXE9	decrement V(X).0, skip if V(X).0 not equal 0
CXEA	decrement V(X).1, skip if V(X).1 not equal 0

SUMMARY OF F INSTRUCTIONS

Instruction	Result
FX0A	convert VX to decimal, result to decimal buffer
FX8A	VX to hex buffer (+ pointed to 3 bytes)
FXB3	save VX
FXB9	restore VX
FXC4	symbol pointed to to display buffer (hex) VX = VX + 1
FXC8	symbol pointed to to display buffer (ASCII) VX = VX + 1
FXDA	point VX to hex buffer
FXDE	point VX to decimal buffer
FXE2	point VX to display buffer
FXF3	save base register in VX
FXF9	load base register from VX

A Simple Program Written in Elfish

This kind of interpretive language lies part way between assembly language and a higher level language like BASIC. The instructions do "tasks" which are smaller than those done in a higher level language and there is more "bookkeeping" to keep track of. However the instructions are simple, the programming is no more difficult than programming in BASIC - there is just a little more code to write to achieve the same result. As an example let's study a program which displays 50 consecutive ASCII symbols on the display as 5 lines of 10 symbols. (See Listing p. 15)

We will read a keyboard byte into a variable, convert the variable to a bit pattern for its ASCII equivalent, and display it; increment the variable and repeat the ASCII conversion and display, etc. until 50 symbols are shown. To do this we have to move the display pointer to the proper positions, keeping track of the number of symbols in each row, and stop when we have shown 50 symbols.

The initialization of variables and the calling of the display subroutine are done in the main program. Variable A is the display pointer, V(A).0 and V(A).1 are initially set to 00 so that the first symbol will appear in the upper left of the screen. VB is set to point to the display buffer where the bit patterns for the symbols to be displayed are assembled. V2.0 is intialized to decimal 50 and decremented each time a symbol is displayed, when it reaches zero the

program returns to the beginning. V1.0 is the hex value for the symbol to be displayed and is incremented after each symbol is shown, initially it is read from the keyboard.

The display subroutine shows the symbols and does the "bookkeeping" for the display pointers. Each time a symbol is shown, 06 is added to VA.0 so that the next symbol will be properly placed along the line. After 10 symbols have been shown 06 is added to VA.1 to start the next line and VA.0 is reset to 00. VC.0 is the counter that keeps track of the number of symbols on a line; it is initialized to 10, and is decremented to count the number of symbols on a line, it is reset to 10 when a line is full.

The program can be placed anywhere in memory as it is relatively addressed, normally this would be done with the editor-assembler. The following listing gives an assembled version of the program; a version which utilizes labels and requires assembly is shown later. Once the program is entered and running enter a byte to the keyboard and push the "in" button, 50 ASCII symbols should appear on the display.

Program Listing

Rel. Address	Code	Remarks
	main program	
0000	41DB	waits, read a byte to V1.0
0002	4FFE	erase display screen
0004	AA00	sets VA.0 to 00
0006	BA00	sets VA.1 to 00
0008	FBE2	point VB to display buffer
000A	AC0A	set VC.0 to 10 (decimal)
000C	A232	set V2.0 to 50 (decimal)
000E	C15B	make ASCII symbol for V1.0
0010	201A	call subroutine to display it
0012	5101	increment V1.0
0014	C2E9	decement V2.0, skip unless done
0016	1000	return to read another byte
0018	100E	else return to do next symbol
	display subroutine	
001A	DBA5	display symbol in buffer
001C	5A06	add 06 to row display pointer
001E	CCE9	decrement row counter, skip unless full
0020	1024	go reset line pointer if row full
0022	4000	else return from subroutine
0024	AA00	reset row pointer to 00
0026	6A06	add 06 to line pointer
0028	AC0A	reset line counter to 10 (decimal)
002A	4000	return from subroutine

The Editor-Assembler (see Listing 2)

Use of Editor

The editor requires the entry of two 16 byte addresses before it enters the "command" mode. These addresses establish a region of memory for editing; the editor does not allow memory changes outside of this range, all other memory is protected. As well, the assembler treats code only in the region of memory established for editing.

On entry the editor displays SA (Starting Address) and waits for the user to enter the starting address of the region of memory to be edited. Once this is done, END (End Code) is displayed and the editor waits for the entry of the end of the region of memory to be entered. If the end of code address is less than the starting address, the editor will not accept it and returns to the entry point.

The editor then inquires:

```
CLEAR ?      C=YES
```

A single hex digit is to be entered. If it is C, all memory locations in the edited region are set to 00. Any other digit results in no action. If the assembler is to be used over this region, it is best to clear memory so that the assembler will not be confused by random bytes; alternatively, the editor could be reentered and the edited region redefined to only that portion which contains the code to be assembled.

Finally the edit mode "command" is entered. Four lines, each consisting of a 2 byte address followed by two bytes of memory contents, are shown as follows:

```
 0000    0000
>0002    0000< A
 0004    0000
 0006    0000
   ??    0
```

The mode of the editor is indicated by the command at the lower left corner of the screen, ?? indicates that the editor is in the command mode. Other modes are entered by typing a single hexadecimal digit (from 1 through 9) and pushing the in button. Possible modes are given in the following list, the ?? in the lower left will be replaced by the current mode.

```
0    ?? (command mode)
1    REPLAce
2    INSERt
3    DELETe
4    GOTO
5    SCAN Up
6    SCAN Down
7    Change address MODE
8    EXECute
9    ASSEMble
```

If 1 is entered and the in button pushed the editor goes into the replace mode; REPLA appears in place of the ??. Successive two byte interpreter instructions can now be entered from the keyboard, each replaces the two bytes contained in memory on the current line and causes the current line to be incremented by 1.

To change from one command to another you must reenter the

command mode. If a full keyboard byte is shown on the screen, enter 00 and hold the in button down for a second, the command mode will be entered. If a single hex digit shows, enter 0 and push the in button.

Most of the other editor modes are fairly obvious, i.e. insert, delete, scan up, scan down, goto, and execute. An insertion occurs above the current line which remains unchanged. If the current line is deleted the next line becomes the current line. Scan up and scan down only continue while the in button is being held down and the goto instruction requires that a 2 byte address be entered. In the relative address mode the goto instruction refers to relative addresses not absolute ones, it changes with the address mode. The execute mode makes the first command in the edited code the next instruction to be interpreted and the interpreter executes the code in the edited region.

The interpretive code is addressed relative to the first instruction which has the address 0000. If the change address mode command is invoked, the displayed addresses are changed from absolute to relative addresses. The current address mode is indicated by a symbol to the right of the current line, A for absolute addresses and X for relative addresses. Relative addresses are useful if one wishes to assemble the interpretive code by hand, the editor itself was written using earlier versions of the editor which had this feature. Hand assembly is also necessary for calls to machine code routines and when using the memory pointer - these instructions are not assembled.

The assembler requires the use of labels; these are instructions which begin with the hexadecimal digit "E". The interpreter ignores such instructions - they have meaning only for the assembler. As an example let's redo the display program using labels.

Program Listing

```
        main program

0000    EAA0 (E000) label reentry pt
0002    41DB        waits, read a byte to V1.0
0004    4FFE        erase display screen
0006    AA00        sets VA.0 to 00
0008    BA00        sets VA.1 to 00
000A    FBE2        point VB to display buffer
000C    AC0A        set VC.0 to 10 (decimal)
000E    A232        set V2.0 to 50 (decimal)
0010    EAA1 (E010) label rentry pt
0012    C15B        make ASCII symbol for V1.0
0014    2BB0 (201E) call subroutine
0016    5101        increment V1.0
0018    C2E9        decement V2.0, skip unless done
001A    1AA0 (1000) return to label EAA0
001C    1AA1 (1010) else return to label EAA1

        display subroutine

001E    EBB0 (E01E) label entry to subroutine
0020    DBA5        display symbol in buffer
0022    5A06        add 06 to row display pointer
0024    CCE9        decrement row counter, skip unless full
0026    1BB1 (102A) go to label EBB1 if row full
0028    4000        else return from subroutine
002A    EBB1 (E02A) entry point
002C    AA00        reset row pointer to 00
002E    6A06        add 06 to line pointer
0030    AC0A        reset line counter to 10 (decimal)
0032    4000        return from subroutine
```

Labels, instructions beginning with E, have been placed at all entry points and at the beginning of the subroutine. The three hexadecimal digits following the E are used to distinguish labels from each other.

The listing shows the labels as they appear before assembly; the digits following the E are chosen arbitrarily. Here I have used EAA(N) in the main program and EBB(N) in the subroutine. Note that 1 and 2 instructions now refer to labels instead of to addresses. For example the 2 instruction at address 0014 is now 2BB0 and references the label at the beginning of the subroutine.

Once the program and labels are entered into memory, assembly can be carried out. From the command mode enter "9" and push the in button, this places the editor-assembler in assemble mode. A second push of the in button starts the assembly process. The assembler first checks for duplicate labels; if one occurs, the assembler stops on the first occurrence of the duplicate label and returns to the command mode. Next a check is made for 1 or 2 instructions without a corresponding label; if one occurs, it becomes the current line and the command mode is entered. The assembler stops when errors occur so that they can be corrected. If the first two passes are successful, the code is assembled; the three least significant digits of the labels are replaced by their relative addresses and the 1 and 2 instructions are changed to correspond. The instructions in parenthesis show the changes that take place on assembly. If assembly

is successful, the last line of the edited region is made the current line and it is surrounded by OK.

There are a number of interesting features to such an assembly process. The code can be repeatedly assembled without harm, the assembled code contains valid labels, jumps, and subroutine calls so that it can be properly reassembled. Thus one can insert new lines of code, or delete lines of code, or move subroutines to new locations without entering new labels, just by reassembling the program. Additional labels or subroutines can be added to already assembled code at any time, as long as the new labels do not duplicate existing labels. Once the code has been assembled additional calls to existing subroutines do have to take account of their new labels. However most of the work of assembly is done, freeing the programmer to concentrate on the code.

Machine code subroutine calls (0 instructions), and the instruction which points a variable to memory (3 instructions), are not handled by the assembler. This was deliberate, as machine code subroutines can contain inadvertant E's which would cause errors. In the same fashion the 3 instructions normally are used to point to data, which again might contain misleading E's. It seems best to assemble these instructions by hand as they will not occur often. They can be placed at the end of the program, the edited region reassigned before assembly, and the 0 and 3 instructions then assembled by hand using the relative address mode.

Listing 1

Listing of Interpreter

Use of Registers

R0	direct memory access
R1	interrupt address
R2	stack pointer
R3	subroutine prog. counter
R4	calling section program counter
R5	interpreter prog. counter
R6	X pointer - R(6).1 page for variables
R7	Y pointer
R8	base register
R9	decremented in interrupt
RA	R(A).1 - display page
RB-RF	scratch

initialize registers, goto control section

```
Address      Code

000     F8 01 B5 F8   set address 1st
004     00 A5 F8 0E   instrc. (0100)
008     BA F8 0F B6   display page(0E)
00C     95 B8 85 A8   var., stack page
010     96 B2 F8 9F   (0F) base reg.
014     A2 E3 70 23   R(8), stack add.
018     93 B4 B1 F8   interrupt add.
01C     7F A1 F8 23   turn on TV make
020     A4 69 D4      R(4) p.c.

            begin control section

023             96    R(4) is p.c.
024     B7 45 BB F6   control section
028     F6 F6 F6 32   R(3) is calling
02C     4E FC 5D AC   p.c., point R6
030     9B F9 F0 FE   at VX.0, R7 at
034     A6 05 F6 F6   VY.0, R(3) to
038     F6 F6 F9 F0   correct sub.,
03C     FE A7 94 BC   make R3 p.c.
040     EC F4 B3 8C   0 instructions
044     FC 0F AC 0C   go to 4E
048     A3 E2 D3 30
04C     23 00

            0 instruc. (MC calls)

04E           9B E2   handle MC calls
050     22 73 45 52   change rel. add.
054     88 F4 A3 12   to absolute and
058     98 74 B3 12   load to R3
05C     30 4A

            sub. relative addresses

05E           00 00   15 high order
060     01 01 01 01   addresses, then
064     01 01 01 00   15 low order
068     00 02 00 01
06C     02 EA E3 06
070     D4 CF CE 3B
074     2B 92 FC FB
078     00 9B 71 01

            display interrupt routine

07C     00 42 70 C4   standard display
080     22 78 22 52   interrupt rout.
084     E2 E2 29 9A   except R9 is
088     B0 F8 00 A0   decremented for
08C     80 E2 E2 20   benefit of
090     A0 E2 20 A0   timing MC
094     E2 20 A0 3C   routines
098     8C 30 7D

            display routine

09B             47    R7 points
09C     FA 3F F6 F6   originally at
0A0     F6 22 52 07   VY.0 - R6 to
0A4     27 FE FE FE   locations of
0A8     F1 AC 9A BC   bytes to
0AC     45 FA 0F AA   display (VX),
0B0     46 AB 06 BB   display page is
0B4     F8 00 BF 8A   obtained from
0B8     32 E1 2A 4B   RA.1 - RC is
0BC     BE 07 FA 07   loaded with
0C0     AE 8E 32 CD   display page
0C4     9E F6 BE 9F   word address
0C8     76 BF 2E 30
0CC     C1 9E EC F3
0D0     5C 02 FB 07
0D4     32 DB 1C 9F
0D8     F3 5C 2C 8C
0DC     FC 08 AC 3B
0E0     B4 12 D4

            2 followed by 1 instruc.

0E3             15    push interp.
0E4     95 22 73 85   p.c. (R5)
0E8     52 25 25 45   to stack -
0EC     FA 0F 22 73   set interp.
0F0     45 52 88 F4   p.c. for
0F4     A5 12 98 74   entry point
0F8     B5 12 D4
```

```
                A followed by B instrc.
                                                  192          22 07
0FB                   16                          194    52 96 BE 86   ready registers
0FC     45 56 D4 00                               198    AE 1E F8 00   decide multiply
                                                  19C    5E F8 09 AD   or divide
                End of page 00, begin             1A0    45 F6 33 B5
                page 01, 1st 4000 instrc.         1A4    0E 76 5E 06   multiply starts
                return from inter. sub.           1A8    76 56 2D 8D   at 01A4, divide
                                                  1AC    32 19 3B A4   at 01B5
100     42 A5 42 B5   restore interp.             1B0    0E F4 5E 30
104     D4 00         p.c. from stack             1B4    A4 07 3A BD   on division by
                                                  1B8    F8 FF 5E 30   0 quotient and
                3 Instruction                     1BC    18 0E F7 3B   remainder are
                                                  1C0    C2 5E 06 7E   set to FF
106           F8 E0   add contents                1C4    56 2D 8D 32
108     A6 25 45 FA   of base                     1C8    19 0E 7E 5E
10C     0F 22 73 45   register to                 1CC    30 BD
110     52 88 F4 56   MMM, load
114     12 16 98 74   to V0                             begin 5 and 6 instruc.
118     56 12 D4
                                                  1CE          16 45
                7 and 8 instructions              1D0    E6 F4 56 D4

11B                5F   first 16 bytes                  begin 4 instructions
11C     5E 61 64 67   are  a table of
120     6A 73 77 7C   entry points                1D4    16 45 A3 26   keyboard
124     7B 7E 81 84                               1D8    E6 6C D4 26   instructions,
128     87 8B 8F 93   16 bit entry                1DC    3F DC 37 DE   and erase
12C     BC 05 FA 08   is 012B                     1E0    30 D8 3F E2   display page,
130     3A 3A 05 FA                               1E4    D4 3F E4 30   in button
134     07 FC 23 AC                               1E8    70 3F EF 15   controls
138     30 58 26 93                               1EC    15 37 ED D4   3F MM and
13C     BC 05 FA 04   3 bit entry                 1F0    9A BC F8 FF   37 MM
140     F6 F6 F6 05   is 013B                     1F4    AC F8 00 5C   instructions
144     FA 08 3A 4B                               1F8    8C 32 EF 2C
148     30 4D DC 16                               1FC    30 F5 30 F0
14C     17 22 76 52
150     45 FA 07 FC                                     end of page 01
154     1B AC 42 7E                                     begin page 02, C and F
158     0C AC E6 07                                     instructions
15C     30 4A F2 56
160     D4 F3 56 D4                               200    16 45 A3 26   hex to decimal
164     74 56 D4 75                               204    06 AF F8 00   routine -
168     56 D4 F3 3A                               208    30 0E 46 AF   8 bit low entry
16C     72 30 70 15                               20C    06 26 22 52   is 0203, 8 bit
170     15 15 D4 F3                               210    F8 B5 A7 93   high entry is
174     3A 70 D4 77                               214    BE F8 38 AE   0204, 16 bit
178     3B 70 D4 F2                               218    17 1E F8 00   entry is 020A
17C     56 D3 F3 56                               21C    57 E6 4E F5
180     D3 F4 56 D3                               220    E2 0E 75 2E
184     F5 56 D3 F3                               224    3B 31 52 E6
188     3A 71 D3 F3                               228    0E F5 56 07   table of powers
18C     3A 6F D3 F7                               22C    FC 01 57 30   of ten starts at
190     D3 00                                     230    1E 4E F6 3B   0239 ends at
                                                  234    18 12 8F 56   0242
                9 instructions                    238    D4 10 27 E8
```

```
23C   03 64 00 0A
240   00 01 00 26   CX43 & CX44
244   F8 E0 A7 47   byte pointed to
248   AD 07 BD 4D   by V0 to VX
24C   56 9D 57 27   V0 = V0 + 1
250   8D 57 D4 00
254   00 00 26 F8   CX56 & CX57
258   B0 30 5E 26   CX5B & CX5C
25C   F8 C0 AC 86   generate symbols
260   A7 93 FC 01   call sub. pg 03
264   BC DC 26 F8
268   08 56 96 BD   CX66 & CX67
26C   F8 BB AD ED   length of symbol
270   72 F1 1D F1   in display buff.
274   1D F1 1D F1   to VX lo or high
278   32 86 AD 8D   blank is given a
27C   F6 AD 33 89   length of 04
280   06 FF 01 56
284   30 7B F8 04
288   56 D4 93 BE   FX8A VX to hex
28C   F8 A5 AE 86   buffer +
290   A7 17 F8 AC   pointed to
294   AD 96 BD DE   3 bytes for
298   27 DE 46 A7   benefit of
29C   06 B7 DE 17   editor-assembler
2A0   DE 17 DE D4
2A4   D3 07 F6 F6
2A8   F6 F6 5D 1D
2AC   07 FA CF 5D
2B0   1D 30 A4 86   FXB3 save VX
2B4   FF 20 A7 30
2B8   BE 86 A7 FF   FXB9 restore VX
2BC   20 A6 46 57
2C0   06 17 57 D4
2C4   F8 B0 30 CA   FXC4 hex symbol
2C8   F8 C0 AC 86   FXC8 ASCII
2CC   A7 46 A7 06   symbol
2D0   B7 17 97 56
2D4   26 87 56 27
2D8   30 61 F8 AC   FXDA pt hex buf
2DC   30 E4 F8 B6   FXDE pt dec buf
2E0   30 E4 F8 BB   FXE2 point to
2E4   56 16 96 56   display buffer
2E8   D4 26 06 FF
2EC   01 56 32 F2   CXE9 & CXEA
2F0   15 15 D4 88   dec. skp = 0
2F4   56 16 98 56   FXF3 save base
2F8   D4 46 A8 06   FXF9 restore
2FC   B8 D4 00 00   base

          end of page 02
          begin page 03 display
          subroutines and symbols

300   00 10 20 88   first 16 bytes
304   A8 50 F8 70   are combined to
308   80 90 A0 B0   form symbols
30C   C0 D0 E0 F0
310   46 3E 96 F9 F9     @ A   00,01
315   5F 57 FF 88 F8     B C   02,03
31A   5F 55 FF F8 F8     D E   04,05
31F   88 8F FF B9 F8     F G   06,07
324   99 9F 79 22 72     H I   08,09
329   AE 22 97 CA 9A     J K   0A,0B
32E   8F 88 38 44 36     L M   0C,0D
333   99 DB F9 99 F9     N O   0E,0F
338   88 9F FF B9 F9     P Q   10,11
33D   A9 9F FF F1 F8     R S   12,13
342   22 22 F6 99 99     T U   14,15
347   22 55 53 45 44     V W   16,17
34C   53 52 23 22 35     X Y   18,19
351   CF 12 CF 88 C8     Z [   1A,1B
356   10 C2 E0 22 E2     \ ]   1C,1D
35B   30 25 60 00 00     ^ _   1E,1F
360   00 00 C0 C0 CC     sp !  20,21
365   00 50 55 56 56     " #   22,23
36A   46 46 F6 FD FB     $ %   24,25
36F   5F AF 0E 00 88     & '   26,27
374   21 22 21 11 21     ( )   28,29
379   53 56 23 62 22     * +   2A,2B
37E   2E 00 00 60 00     , -   2C,2D
383   CC 00 00 2C 01     . /   2E,2F
388   9F 99 2F 22 22     0 1   30,31
38D   8F 1F FF F1 F1     2 3   32,33
392   22 AF FA F1 F8     4 5   34,35
397   9F 8F 1F 11 F1     6 7   36,37
39C   9F 9F FF F1 F9     8 9   38,39
3A1   80 80 E0 02 02     : ;   3A,3B
3A6   21 2C 01 0E 0E     < =   3C,3D
3AB   2C 21 8C F8 F1     > ?   3E,3F
3B0   07 AF FA 0F
3B4   F9 30 57 FD   entry point to
3B8   39 33 C2 FD   generate hex
3BC   40 57 30 C2   symbols, 03B0
3C0   07 AF 96 BB   entry for ASCII
3C4   F8 BF AB 9C   symbols, 03C0
3C8   B3 BD EB 07
3CC   FA 3F 5B F4
3D0   F4 F4 F4 76
3D4   3B DB FC 10
3D8   AD 30 E9 FC
3DC   10 AD 0D FA
3E0   0F A3 8B FB
3E4   BA 32 F8 03
3E8   73 4D F6 F6
3EC   F6 F6 A3 8B
3F0   FB BA 32 F8
3F4   03 73 30 DE
3F8   8F 57 D4 00
3FC   00 00 00 00
```

Listing 2

Listing of Editor-Assembler

Use of Variables in Editor

V(0) points to messages
V(1) the display pointer
V(2).0 number of symbols to display in display subroutines
V(3).1 marker, set if illegal move or a bad assemble
V(3).0 counter in routine to display current line (abbv. CL)
V(4).0 marker 0 ordinary keyboard, 1 special keyboard
V(4).1 marker 0 absolute address, 1 relative address
V(5) scratch for keyboard subroutine
V(6) keyboard bytes are passed in V6
V(7) stores starting address of edited code (abbv. SA)
V(8) holds current line (abbv. CL)
V(9) holds end line (EL)
V(A) points to display buffer
V(B) through V(E), scratch
V(F) set to 0002, much used constant

To use the editor-assembler enter at the first location of the editor-assembler with R3 the program counter, and locate the interpreter on the four pages immediately following the editor-assembler.

```
Abs.  Rel.
add.  add.

000         93 BC     machine code to
002         B5 F8     enter
004         06 AC     interpreter at
006         DC F8     an address
008         12 A5     relative to the
00A         9C FC     0000 here, i.e.
00C         04 B3     at 0406 higher
00E         F8 06     in memory
010         A3 D3

             Begin Main Program

012  000    BF00  initialize
014  002    AF02  V(F)=0002
016  004    B400  marker=abs. add.
018  006    A400  marker=simple key
01A  008    FAE2  V(A) to display
01C  00A    4FFE  erase display
01E  00C    326A  pt. to message
020  00E    A203  3 letters
022  010    2114  show S A
024  012    2148  keys to V(6)
026  014    8760  save as SA
028  016    8860  also as CL
02A  018    A203  3 letters
02C  01A    2116  show END
02E  01C    2148  keys to V(6)
030  01E    8960  save as EL
032  020    8797  skip V7>V9
034  022    1026  OK go on
036  024    1000  NO start over
038  026    A20D  13 letters
03A  028    2116  show CLEAR? C=YES
03C  02A    21F8  1 digit to V(6).0
03E  02C    A50C  V(5).0 = 0C
040  02E    7656  skp V(6).0=V(5).0
042  030    2262  call CLEAR
044  032    4FFE  erase display
046  034    21C2  show >< A or X
048  036    2192  put up CL
04A  038    02FB  erase bot(rentry)
04C  03A    327D  point to ??
04E  03C    210E  display ??
050  03E    40ED  wait in off
052  040    21F8  1 digit to V(6).0
054  042    A40A  1 more no. instr.
056  044    7467  skp V(4).0>V(6).0
058  046    1038  illegal, do over
05A  048    A505  V(5).0 = 05
05C  04A    9560  V5 = V5*V6
05E  04C    327D  point to ??
060  04E    8053  point to message
062  050    02FB  erase bottom
064  052    210E  show message
066  054    0377  go to right sub
068  056    1038  to command mode
06A  058    10AC  to REPLACE
06C  05A    10BA  to INSERT
06E  05C    10C8  to DELETE
070  05E    10D8  to GOTO
072  060    10E2  to SCAN UP
074  062    10E8  to SCAN DOWN
076  064    10F8  to CHANGE MODE
078  066    1102  to EXECUTE
07A  068    2178  ASSEMBLE (rd dig)
07C  06A    86FE  skp V(6).0≠0
07E  06C    1038  to command mode
080  06E    8870  start DL search
082  070    8897  skp if CL>SA
084  072    1076  go on
086  074    1080  done DL search
088  076    220C  call check DL
08A  078    73FD  skp loc OK
08C  07A    1032  bad L,to command
08E  07C    88F3  add 02 to CL
090  07E    1070  back to do more
092  080    A502  start NL search
094  082    8870  CL = SA
```

```
096  084   8897  skp CL>SA
098  086   108A  go on
09A  088   1094  chk pass no.
09C  08A   2216  call assemble
09E  08C   73FD  skp loc OK
0A0  08E   1032  to control, bad
0A2  090   88F3  +2 to CL
0A4  092   1084  continue
0A6  094   C5E9  dec. V(5), skp≠0
0A8  096   109A  done, goto fix
0AA  098   1082  back, 2nd pass
0AC  09A   8870  success fix labels
0AE  09C   8897  skp if CL>SA
0B0  09E   10A6  go on
0B2  0A0   02EE  erase top screen
0B4  0A2   21C4  show OK
0B6  0A4   1036  go show CL
0B8  0A6   2240  call fix
0BA  0A8   88F3  +2 to CL
0BC  0AA   109C  return till done
0BE  0AC   21FE  REPLACE, read keys
0C0  0AE   21E0  show >< A or X
0C2  0B0   73FE  skp illegal try
0C4  0B2   0339  do replace
0C6  0B4   88F3  +2 to CL
0C8  0B6   2192  show new CL
0CA  0B8   10AC  do over
0CC  0BA   21FE  INSERT, read keys
0CE  0BC   21E0  show >< A or X
0D0  0BE   73FE  skp illegal try
0D2  0C0   2258  do insert
0D4  0C2   88F3  +2 to CL
0D6  0C4   2192  show new CL
0D8  0C6   10BA  do over
0DA  0C8   21F8  DELETE, read keys
0DC  0CA   86FE  skp unless = 0
0DE  0CC   1038  return to control
0E0  0CE   21E0  show >< A or X
0E2  0D0   73FE  skp illegal try
0E4  0D2   2252  do delete
0E6  0D4   2192  show new CL
0E8  0D6   10C8  do over
0EA  0D8   21FE  GOTO read keys
0EC  0DA   74FD  skp unless rel add
0EE  0DC   8673  V(6) = V(6)+V(7)
0F0  0DE   8860  CL = V(6)
0F2  0E0   1032  show new CL
0F4  0E2   BEFF  SCAN UP
0F6  0E4   AEFE  V(E) = -2
0F8  0E6   10EA
0FA  0E8   8EF0  SCAN DOWN
0FC  0EA   2174  read keyboard
0FE  0EC   86FE  skp unless = 0
100  0EE   103E  goto command
102  0F0   02EE  erase top
104  0F2   88E3  CL = CL + V(E)
106  0F4   21BC  show new CL
108  0F6   10EA  do over

10A  0F8   74FD  CMODE
10C  0FA   10FE  jmp to 0FE
10E  0FC   B402  marker,V(6).1=02
110  0FE   64FF  mark=mark-1
112  100   1032  return to control
114  102   21F8  EXECUTE, read keys
116  104   86FE  skp unless = 0
118  106   1038  return to control
11A  108   4FFE  erase display
11C  10A   F7F9  base register=SA
11E  10C.  1000  goto SA

                 End of Main Program
                 Sub to Show Messages

120  10E   A205  5 characters
122  110   B11B  V(1).1 = 1B
124  112   1118  go set V(1).0
126  114   B1FA  entry point
128  116   6106  entry point
12A  118   A100  V(1).0 = 00
12C  11A   F0C8  symbol to show
12E  11C   DA15  display symbol
130  11E   C267  length to V(2).1
132  120   6201  increment V(2).1
134  122   812B  V1.0=V1.0+V2.1
136  124   C2E9  dec V2.0, skp=0
138  126   4000  return
13A  128   111A  else do another

                 End of Show Messages
                 Sub to Show Numbers

13C  12A   FBDA  VB to hex buffer
13E  12C   A204  4 characters
140  12E   1134  jmp to display
142  130   FBDA  entry 2 characters
144  132   A202  2 characters
146  134   FBC4  symbol to show
148  136   5105  V1.0=V1.0+05
14A  138   DA15  display symbol
14C  13A   C2E9  dec, skp unless=0
14E  13C   1140  goto exit
150  13E   1134  else do another
152  140   51F6  reset V1.0 for key
154  142   4000  return

                 End of Show Numbers
                 Keyboard Caller

156  144   A11C  set display point
158  146   B11B  set display point
15A  148   2156  call first byte
15C  14A   76E8  save in V(6).1
15E  14C   510A  add 0A to pointer
160  14E   2156  call second byte
162  150   86E8  save in V(6).0
164  152   4000  return
```

```
                    End of Keyboard Caller
                    Keyboard Subroutine

166  154   2130  show to erase
168  156   4ED8  entry,keys to VE.1
16A  158   FE8A  to hex buffer
16C  15A   2130  call display
16E  15C   75E8  V5.1=VE.1
170  15E   4ED8  keys to VE.1
172  160   75ED  skp if V5.1=VE.1
174  162   1154  else go to erase
176  164   84FD  skp if V4.0=VF.1
178  166   116E  go to MC routine
17A  168   40E9  skp if in on,off
17C  16A   115E  stay in loop
17E  16C   1172  return
180  16E   0318  special MC keys
182  170   115E  stay in loop
184  172   4000  return

                    End of Keyboard Routine
                    One Digit Keyboard

186  174   0308  erase keys
188  176   46D7  keys to V6.0
18A  178   C656  hex symbol to buf
18C  17A   A121  display pointer
18E  17C   B11B  display pointer
190  17E   DA15  show one digit
192  180   7560  V5.0=v6.0
194  182   46D7  keys to V6.0
196  184   7565  skp if V5.0=V6.0
198  186   1174  erase, key pushed
19A  188   40E5  return, skp in on
19C  18A   1182  stay key loop
19E  18C   A50F  V5.0=0F
1A0  18E   7651  V6.0=v6.0+V5.0
1A2  190   4000  return

                    End One Digit Keyboard
                    Display Subroutine

1A4  192   B100  set display point
1A6  194   A304  marker, 4 lines
1A8  196   8C80  VC = CL
1AA  198   8CF4  VC = VC - 02
1AC  19A   8CF4  VC = VC - 02
1AE  19C   8CF3  VC = VC + 02
1B0  19E   FC8A  VC + to hex buf
1B2  1A0   74FD  skp on RA mark
1B4  1A2   21B4  call RA sub
1B6  1A4   A100  set display point
1B8  1A6   212A  show 4 symbols
1BA  1A8   5112  set display point
1BC  1AA   212C  show 4 symbols
1BE  1AC   6106  point next line
1C0  1AE   C3E9  4 lines?, return
1C2  1B0   4000  return

1C4  1B2   119C  else do another
1C6  1B4   8DC0  RA sub, VD=display
1C8  1B6   8D74  VD=VD-SA (RA)
1CA  1B8   0394  MC, add RA to buf
1CC  1BA   4000  return
1CE  1BC   21C2  show ><A or X
1D0  1BE   1192  go to CL
1D2  1C0   0000  unused

                    End of Display Routine
                    Sub to Add >< A or X

1D4  1C2   32B3  point to ><
1D6  1C4   B106  set display point
1D8  1C6   A201  one symbol
1DA  1C8   2118  show it
1DC  1CA   5130  set display point
1DE  1CC   A201  one symbol
1E0  1CE   211A  show it
1E2  1D0   5101  set display point
1E4  1D2   A201  one symbol
1E6  1D4   74FD  skp RA marker
1E8  1D6   5001  point X if RA
1EA  1D8   211A  show A or X
1EC  1DA   4000  return
1EE  1DC   32B7  point to ??
1F0  1DE   11C4  go put it up

                    End of Routine for ><
                    Sub Is Command Legal?

1F2  1E0   02EE  MC erase top
1F4  1E2   B302  V3.1 = 02
1F6  1E4   8897  skp CL>EL
1F8  1E6   63FF  V3.1=V3.1-01
1FA  1E8   8787  skp SA>CL
1FC  1EA   63FF  V3.1=V3.1-01
1FE  1EC   73FE  skp if V3.1=0
200  1EE   11F4  legal show ><
202  1F0   21DC  illegal show ??
204  1F2   4000  return
206  1F4   21C2  show ><
208  1F6   4000  return

                    End Legal Check Sub
                    One Hex Symbol Caller

20A  1F8   2174  call one digit key
20C  1FA   40ED  wait in off
20E  1FC   4000  return

                    End One Symbol Caller
                    Erase Keys, Goto Keys

210  1FE   0308  MC, erase keys
212  200   1144  goto key caller

                    End erase keys, etc.
```

```
                    Register Set Utility
214  202    8E80    VE=CL
216  204    8D90    VD=EL
218  206    8D84    VD=EL-CL
21A  208    034A    MC,RD=VE;RE,RF=VF
21C  20A    4000    Return

                    End Register Utility
                    Sub for Double Label
21E  20C    B300    set marker
220  20E    2202    call set register
222  210    02BB    MC DL, skips good
224  212    B301    set mark DL found
226  214    4000    return

                    End Double Label Sub
                    Assemble Subroutine
228  216    B300    set marker
22A  218    8E80    VE=CL
22C  21A    8D90    VD=EL
22E  21C    8D74    VD=EL-SA
230  21E    034A    MC, set registers
232  220    03A7    MC, looks for 1,2
234  222    4000    return, no 1 or 2
236  224    02BB    MC no skp found
238  226    8DE6    no skp found self
23A  228    02BC    continue MC sub
23C  22A    122E    go set label
23E  22C    123A    no, set flag
240  22E    75F6    ? 1st or 2nd pass
242  230    123C    1st return
244  232    8E74    2nd, VE=RA
246  234    034A    call set register
248  236    03C9    MC set correct RA
24A  238    4000    return
24C  23A    B301    bad, set label
24E  23C    03C6    MC, restore
250  23E    4000    return

                    End assemble subroutine
                    Fix Subroutine
252  240    8E80    VE=CL
254  242    034A    MC, set registers
256  244    03D7    MC look for E
258  246    4000    return, no E
25A  248    8E74    VE=RA
25C  24A    8D80    VD=CL
25E  24C    034A    set registers
260  24E    03C9    MC set label
262  250    4000    return

                    End Fix Subroutine
                    Delete Subroutine
264  252    2202    set registers
266  254    0368    MC delete
268  256    4000    return

                    End Delete subroutine
                    Insert subroutine
26A  258    8E90    VE=EL
26C  25A    2204    set registers
26E  25C    0358    MC insert
270  25E    0339    MC replace
272  260    4000    return

                    End Insert Subroutine
                    Clear Memory Sub
274  262    8E70    VE=SA
276  264    2204    set registers
278  266    0384    MC call clear
27A  268    4000    return

                    End Clear Memory
                    Begin Messages
27C  26A    1320    S(blank)
27E  26C    0105    AE
280  26E    0E04    ND
282  270    030C    CL
284  272    0501    EA
286  274    1220    R(blank)
288  276    3F20    ?(blank)
28A  278    033D    C=
28C  27A    1905    YE
28E  27C    1320    S(blank)
290  27E    2020    (blank)(blank)
292  280    3F3F    ??
294  282    1205    RE
296  284    100C    PL
298  286    0109    AI
29A  288    0E13    NS
29C  28A    0512    ER
29E  28C    0405    DE
2A0  28E    0C05    LE
2A2  290    1407    TG
2A4  292    0F14    OT
2A6  294    0F20    O(blank)
2A8  296    1303    SC
2AA  298    010E    AN
2AC  29A    1513    US
2AE  29C    0301    CA
2B0  29E    0E04    ND
2B2  2A0    030D    CM
2B4  2A2    0FC4    OD
2B6  2A4    0505    EE
2B8  2A6    1805    XE
2BA  2A8    0320    C(blank)
2BC  2AA    0113    AS
2BE  2AC    1305    SE
```

```
2C0  2AE     0D0F  MO
2C2  2B0     0B01  KA
2C4  2B2     183E  X>
2C6  2B4     3C01  <A
2C8  2B6     183F  X?
2CA  2B8     3F01  ?A
2CC  2BA     18    X
```

End Messages

Machine Code Routine
to look for DL

```
2CD  2BB        1D
2CE  2BC     8D FA
2D0  2BE     FE AD
2D2  2C0     0E FA
2D4  2C2     F0 FB
2D6  2C4     E0 EF
2D8  2C6     32 DE
2DA  2C8     15 15
2DC  2CA     D4 2E
2DE  2CC     1F 1F
2E0  2CE     2D 2D
2E2  2D0     9D 3A
2E4  2D2     E8 8D
2E6  2D4     32 DA
2E8  2D6     4E F3
2EA  2D8     3A DD
2EC  2DA     1F 0E
2EE  2DC     F3 2E
2F0  2DE     3A DF
2F2  2E0     2F F8
2F4  2E2     FD A6
2F6  2E4     E6 9F
2F8  2E6     73 8F
2FA  2E8     73 9E
2FC  2EA     73 8E
2FE  2EC     56 D4
```

machine code to erase
top of display

```
300  2EE     9A BC
302  2F0     F8 C9
304  2F2     AC 2C
306  2F4     F8 00
308  2F6     5C 8C
30A  2F8     3A 05
30C  2FA     D4
```

machine code to erase
bottom of display

```
30D  2FB        9A
30E  2FC     BC F8
310  2FE     C8 AC
312  300     F8 00
314  302     5C 1C
316  304     8C 3A
318  306     12 D4
```

machine code to erase
key display

```
31A  308     9A BC
31C  30A     F8 CC
31E  30C     AE F9
320  30E     04 AC
322  310     F8 00
324  312     5C 1E
326  314     8E 3A
328  316     1F D4
```

special keys,
machine code to
handle keys to command

```
32A  318     F8 FD
32C  31A     A6 3F
32E  31C     39 F8
330  31E     28 A9
332  320     89 32
334  322     3A 37
336  324     32 15
338  326     15 D4
33A  328     06 3A
33C  32A     2F 93
33E  32C     FF 03
340  32E     B5 F8
342  330     4A A5
344  332     12 12
346  334     12 12
348  336     D4 00
34A  338     00
```

machine code replace

```
34B  339        F8
34C  33A     F0 A6
34E  33C     46 AE
350  33E     06 BE
352  340     1E F8
354  342     EC A6
356  344     46 5E
358  346     2E 06
35A  348     5E D4
```

machine code to set
registers

```
35C  34A     F8 FA
35E  34C     A6 46
360  34E     AD 46
362  350     BD 46
364  352     AE AF
```

```
366  354     06 BE
368  356     BF D4

  machine code insert

36A  358     2F 1E
36C  35A     9D 3A
36E  35C     72 8D
370  35E     32 79
372  360     0F 5E
374  362     2F 2E
376  364     2D 30
378  366     6C D4

  machine code delete

37A  368     1F 1F
37C  36A     9D 3A
37E  36C     82 8D
380  36E     32 88
382  370     4F 5E
384  372     1E 2D
386  374     30 7C
388  376     D4

  machine code to skip to
  proper subroutine

389  377        F8
38A  378     EC A6
38C  37A     06 32
38E  37C     95 15
390  37E     15 FF
392  380     01 30
394  382     8D D4

  machine code to clear
  memory

396  384     1D 9D
398  386     3A 9D
39A  388     8D 32
39C  38A     A4 F8
39E  38C     00 5E
3A0  38E     2D 1E
3A2  390     30 97
3A4  392     D4 00

  relative address sub

3A6  394     94 FC
3A8  396     02 BE
3AA  398     F8 A5
3AC  39A     AE F8
3AE  39C     FB A7
3B0  39E     F8 AC

3B2  3A0     AD 96
3B4  3A2     BD DE
3B6  3A4     27 DE
3B8  3A6     D4

  main machine code
  sub for assemble

3B9  3A7        0E
3BA  3A8     AB FA
3BC  3AA     F0 FB
3BE  3AC     10 32
3C0  3AE     C6 FB
3C2  3B0     30 32
3C4  3B2     C6 D4
3C6  3B4     15 15
3C8  3B6     F8 EE
3CA  3B8     A6 46
3CC  3BA     AF 06
3CE  3BC     BF 2F
3D0  3BE     2F 0E
3D2  3C0     FA 0F
3D4  3C2     F9 E0
3D6  3C4     5E D4
3D8  3C6     8B 5E
3DA  3C8     D4

  machine code assemble
  sets labels and jumps

3DB  3C9        8B
3DC  3CA     FA F0
3DE  3CC     5D ED
3E0  3CE     9F FA
3E2  3D0     0F F1
3E4  3D2     5D 1D
3E6  3D4     8F 5D
3E8  3D6     D4

  machine code fix
  (looks for E, skips if
  found, saves in R(B).0

3E9  3D7        0E
3EA  3D8     AB FA
3EC  3DA     F0 FB
3EE  3DC     E0 3A
3F0  3DE     F3 15
3F2  3E0     15 D4
3F4  3E2     00 00
3F6  3E4     00 00
3F8  3E6     00 00
3FA  3E8     00 00
3FC  3EA     00 00
3FE  3EC     00 00
```

0 ?? (command mode)
1 REPLAce
2 INSERt
3 DELETe
4 GOTO
5 SCAN Up
6 SCAN Down
7 Change address MODE
8 EXECute
9 ASSEMble

```
0MMM  do machine code subroutine
1MMM  go to MMM (relative address)
2MMM  do interpretive subroutine (RA)
3MMM  point V0 to MMM (RA)
4XZZ  keyboard (4000 return sub.)
5XKK  VX.0 = VX.0 + KK
6XKK  VX.1 = VX.1 + KK
7XYZ  8 bit arithmetic
8XYZ  16 + 8 bit arithmetic
```

```
9XY0  VX(16) = VX.0 * VY.0
9XY1  VX.0 = VX.0/VY.0, rem. in VX.1
AXKK  VX.0 = KK
BXKK  VX.1 = KK
CXZZ  8-bit one variable inst.
DXYN  display, VX = mem. pointer,
      VY = display, N = bytes to show
EZZZ  ignored, labels for assembler
FXZZ  16-bit one variable inst.
```

```
4000  return from subroutine

4XD7  keys = VX.0
4XD8  keys = VX.1

4XDB  keys = VX.0, wait on, off
4XDC  keys = VX.1, wait on, off

4XE2  wait on

4XE5  skip in on
4XE9  skip in on, wait off

4XED  wait off

4XFE  erase display
```

```
7(X.0,Y.0)  7(X.1,Y.1)
    Z           Z        Result
    0           8        X = Y
    1           9        X = X and Y
    2           A        X = X xor Y
    3           B        X = X + Y
    4           C        X = X - Y
    5           D        skip if X = Y
    6           E        skip if X ≠ Y
    7           F        skip if X > Y
    Z           Z
8(X16,Y16)  8(X.0,Y.1)
```

8-bit one variable
(VX.0 followed by VX.1)

```
CX03  hexidecimal to decimal
CX04  conversion

CX43  byte pointed to by V0 to
CX44  VX.0 or VX.1, V0 = V0 + 1

CX56  symbol (hexidecimal) for VX.0
CX57  or VX.1 to display buffer

CX5B  symbol (ASCII) for VX.0 or
CX5C  VX.1 to display buffer

CX66  symbol length in display
CX67  buffer to VX.0 or VX.1

CXE9  decrement VX.0 or VX.1
CXEA  skip if result not equal 0
```

16-bit one variable

```
FX0A  VX to the decimal buffer

FX8A  VX to the hexidecimal buffer

FXB3  save VX
FXB9  restore VX

FXC4  symbol pointed at to display
      buffer (hex), increment VX
FXC8  symbol pointed at to display
      buffer (ASCII), increment VX

FXDA  point VX to hex buffer
FXDE  point VX to decimal buffer
FXE2  point VX to display buffer

FXF3  save base register in VX
FXF9  restore base reg. from VX
```