

PROGRAMS  
FOR THE  
COSMAC ELF  
INTERPRETERS

PAUL C. MOEWS

# PROGRAMS FOR THE COSMAC ELF INTERPRETERS

Paul C. Moews

## List of Sections

1. Introduction .....	3
2. A Demonstration Interpreter .....	4
3. The CHIP-8 Language .....	9
4. Hardware Differences between 1802 Computers .....	13
5. A Complete ELF CHIP-8 Interpreter .....	13
6. Extending the CHIP-8 Instruction Set .....	22
7. Appendix .....	28

## List of Programs

### *Machine Code*

1. Demonstration Interpreter .....	8
2. Complete CHIP-8 Interpreter .....	14
3. Additional Skip Instructions .....	22
4. Multiply, Divide and 16 Bit Display Instructions .....	23
5. Six Bit ASCII Symbols .....	25

### *Interpretive Code*

1. Addition (Demonstration Interpreter) .....	5
2. Subroutine Use (Demonstration Interpreter) .....	5
3. Addition Problems (Demonstration Interpreter) .....	6
4. Addition Problems (Full Interpreter) .....	10
5. Display ASCII Character (Full Interpreter) .....	26

*Copyright © 1979 by Paul C. Moews*

*All rights reserved*

*Published March, 1979 by Paul C. Moews*

*Printed by Parousia Press, Storrs, Connecticut*

## Introduction

This booklet's purpose is to explain the construction and operation of an interpreter for the COSMAC 1802 "ELF". It assumes that the reader has some knowledge of the 1802 instruction set and is able to write simple machine language programs. Mnemonics are not provided because most ELF owners do not have access to assemblers and must work directly in machine language. Instead, programs are explained in a documented, step-by-step fashion, that it is hoped will make the concepts involved easy to follow.

The interpretive language described is "CHIP-8", the language used by RCA Corporation in its "COSMAC VIP" computer. CHIP-8 is a simple language consisting of about 30 instructions. RCA's interpreter is elegant and well thought out; once understood it is easily changed and modified.

This booklet contains five sections; in the first section a simple demonstration interpreter is introduced. This demonstration interpreter runs in the basic 1/4 "ELF" and its instructions are a subset of the full CHIP-8 instruction set. While simple, the demonstration interpreter employs methods similar to those in the full interpreter.

Further sections discuss the full CHIP-8 instruction set, hardware differences between the "VIP" and the "ELF", and provide a listing of a complete ELF interpreter together with suggestions for implementing it on various machines. The final section discusses the extension of the CHIP-8 instruction set. Examples are provided for multiply and divide instructions together with an instruction which displays characters for the 64 six bit ASCII symbols.

I should like to thank RCA Corporation for permission to write about CHIP-8 and to modify it for the ELF. However RCA is not responsible for any of the material in this booklet. The programs described here have been thoroughly tested on a number of versions of the COSMAC "ELF" as described in Popular Electronics articles and are believed to be reliable but there is, of course, still the possibility that they contain unexpected errors. This kind of interpreter is rather hardware dependent and changes in input/output lines or in the use of flag lines will cause failures. An attempt was made to provide sufficient documentation so that the user can make the changes necessary to implement CHIP-8 on a variety of machines.

### A Demonstration Interpreter

The surprising power of computers is due to the development of languages which organize programming into different levels of complexity. Perhaps the simplest way to organize programming with a language is to use an interpreter. One can consider an interpreter to be a program that converts the basic instruction set to a new language, a set of instructions that better suits the programmer. Alternately an interpreter can be thought of as a program with a control section and a number of subroutines, the new language now instructs the interpreter as to which subroutines to call and in which order. The subroutines perform "tasks" which are more complicated than those performed by a single machine code operation. The ubiquitous basic interpreter is a good example.

RCA's CHIP-8 language is an interpretive one and it converts the 94 machine language instructions of the 1802 microprocessor to a new set of about 30 more powerful and convenient instructions. Each type of statement in the new language is implemented by a machine code subroutine which carries out the desired operation. It differs from a basic interpreter in that most of the operations carried out by the subroutines are small ones, consisting of only a few machine code instructions, and the language is therefore a simple one without many of the features of basic. However quite powerful programs can be written with a few hundred CHIP-8 instructions.

This section introduces a version of CHIP-8 for the 1/4K Elf. Ten of the instructions are a subset of the full CHIP-8 set and are identical to those in CHIP-8. Two additional instructions, read a byte from the keyboard and display a byte on the hex display, have no exact counterparts in the CHIP-8 set.

CHIP-8 instructions consist of four hex digits. The first hex digit determines the type of instruction; there are therefore 16 basic kinds of CHIP-8 instructions. 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. In the demonstration interpreter only the two least significant hex digits are needed for this purpose because it is necessary to address only a single page of memory.

A basic feature of CHIP-8 is that it provides 16 one byte variables, designated V0 through VF. Thus a single hex digit can be used to

specify one of these variables. In many of the CHIP-8 instructions the second most significant hex digit is used for this purpose, leaving the last two hex digits available for other uses. In arithmetic operations the two variables to be added, etc. are specified by the second and third hex digit leaving the last hex digit to designate the type of arithmetic operation to carry out.

Before beginning a discussion of how the interpreter works, it is necessary to have an understanding of the language and its use. The instructions available are shown in Table 1.

*Table 1*

#### Demonstration Interpreter Instructions

00MM	do a machine code subroutine at location MM (The machine code subroutine must end with D4)
10MM	go to MM; control is transferred to location MM in the interpretive code
20MM	do an interpreter subroutine at location MM (The interpreter subroutines must end with 009E)
4XKK	skip if VX≠KK; the next interpreter instruction is skipped over if VX does not equal KK
6XKK	set VX=KK; variable X is made equal to KK
8XY0	set VX=VY; variable X is made equal to variable Y
8XY1	set VX=VY or VY; variable X is made equal to the result of VX logically ored against VY (Note that VF is changed)
8XY2	set VX=VX and VY; variable X is made equal to the result of VX logically anded against VY (Note that VF is changed)
8XY3	set VX=VX xor VY; variable X is made equal to the result of VX logically xored against VY (note that VF is changed)
8XY4	set VX=VX+VY; variable X is made equal to the sum of VX and VY (Note that VF becomes 00 if the sum is less than or equal to FF and 01 if the sum is greater than FF)
8XY5	set VX=VX-VY; variable VX is made equal to the difference between VX and VY (Note that VF becomes 00 if VX is less than VY and 01 if VX is greater than or equal to VY)
8XY6	set VX equal to VY shifted right 1 bit position, (Note bit 0 is shifted into VF)
8XY7	set VX=VY-VX; variable VX is made equal to the difference between VY and VX (Note that VF becomes 00 if the

sum is less than or equal to FF and 01 if the sum is greater than FF)  
 8XYE set VX equal to VY shifted left 1 bit position (Note bit 7 is shifted into VF)  
 DXKK display VX on the hex display, KK indicates the length of a pause for display  
 FX00 set VX equal to the switch byte; waits for the input button to be pushed and released

An easy way to see how these instructions are used is to illustrate them with a simple program. The interpreter is listed at the end of the chapter and can be used to run these sample programs.

To start let's look at the following program. It reads 2 switch bytes, displays them, adds them, and displays the result. If overflow occurs, that is also displayed. The program uses only 10 interpreter instructions (The first instruction 3071 is actually machine code and transfers control on entry to the interpreter; It is not part of the interpretive code.) The interpreter has a program counter for interpretive code (R(5)) which is set on entry to the address of the first instruction (M(0002)). The first interpretive language instruction is 63EE which sets variable number 3 equal to EE.

#### Interpretive Addition Program

Add.	Code	Notes
00	3071	entry to interpreter
02	63EE	set V3 equal to EE
04	F400	set V4 equal to switch byte, waits for in on, off
06	D4FF	display V4 on hex display for about 1.8 seconds
08	F500	set V5 equal to switch byte
0A	D5FF	display V5 on hex display
0C	8454	set V4 equal to V4 + V5
0E	D4FF	display V4, now the sum of V4 + V5
10	4F01	skip next instruction if VF ≠ 01, remember VF will be set to 01 by the 8454 instruction if overflow occurs
12	D3FF	display V3 (V3 was set equal to EE) this instruction is skipped if VF is anything but 01
14	1004	go back to instruction 04 to wait for next number

The above program illustrates most of the demonstration interpreter instructions; an

important exception is the interpreter subroutine call. Unlike the SEP register technique used in simple machine code programs, interpreter subroutines do not have to return to the main program but can be called from other subroutines. A stack is employed to store the return address when a subroutine call is made and successive calls to subroutines, without returns, push the stack further down. In the demonstration interpreter the stack pointer, R(2), points to the last location used and is pushed down one before a new byte is added to the stack. Each time a return from a subroutine occurs the stack pointer is incremented by one.

The next program is a simple illustration of the use of an interpreter subroutine. A switch byte is entered and displayed. It is then counted down by three's until underflow occurs. A subroutine is used to implement the counting down by three.

#### Program to Illustrate Subroutine use

Add.	Code	Notes
00	3071	entry to interpreter
02	F500	set V5 equal to switch byte, waits for in on, off
04	D5FF	display V5 on hex display for about 1.8 seconds
06	200A	call interpreter subroutine at location 0A
08	1002	on return from subroutine go to location 02 to read another switch byte
-	-	begin interpretive subroutine
0A	6603	set V6 equal to 03
0C	8565	set V5 equal to V5 - V6
0E	D540	display V5 for ca 0.4 seconds
10	4F01	skip next instruction if underflow occurs during the subtraction, VF equals 00 on underflow
12	100C	transfer to location 0C to subtract three more
14	009E	return from subroutine

In the above program, the call to the subroutine uses one stack position to store the return address. When the interpreter is entered the stack pointer is set to location 71. On calling the subroutine it is decremented by one, to location 70, and 08, the location the interpreter should execute on return from subroutine, is stored there. If we examine location 70 after running this program 08 will be found there.

Two additional stack locations, 6E and 6F are used by 8565 instruction, these locations become F5 and D3 respectively. An explanation of why this occurs is given in the demonstration interpreter listing.

The interpreter also includes an instruction, 00MM, which executes a machine code subroutine at address MM. This is easily accomplished; the control section of the interpreter treats the machine code subroutine as if it were one of the subroutines written to execute CHIP-8 instruction. All the subroutines which execute CHIP-8 instructions end with a D4 byte.

The following program poses simple addition problems and illustrates most of the demonstration interpreter instructions. It contains a machine language subroutine which generates two random numbers when the in button is pushed. On entry, the program displays AA and the Q light comes on. When the input button is pressed a simple addition problem (base 10) is presented; for example 17AD (for and) 32E0 (for equals) may be displayed. If 00 is entered the problem is shown again, if the correct answer is entered it is displayed followed by AA. However if an incorrect answer is entered EE is shown followed by the correct answer. The program requires 36 interpreter instructions and a machine language subroutine of 25 bytes. An interpreter subroutine is used to generate two random numbers in VD and VE. The displayed numbers are all less than 99 (base 10) to accommodate the hex display and the simple hex to decimal conversion routine which fails for numbers greater or equal to 100 (base 10).

#### Program for Addition Problems

Add.	Code	Notes
00	3071	entry to interpreter
02	60E0	set V0 equal to E0
04	61EE	set V1 equal to EE
06	62AD	set V2 equal to AD
08	63AA	set V3 equal to AA
0A	D300	display V3 (AA) on the display but no delay for display
0C	004A	call machine language subroutine which generates random numbers in VD and VE when in is pushed
0E	8BE0	set VB equal to VE as preparation for summing the two random numbers
10	8BD4	set VB equal to VD + VE, sum of the two random numbers
12	203A	call the interpreter subroutine which converts from hex to decimal, answer is returned in VA and VB is changed
14	8CA0	save answer on return from subroutine by setting VC equal to VA
16	8BE0	set VB equal to VE, one of the random numbers
18	203A	call subroutine to make VA the decimal equivalent of VB
1A	DAFF	display VA, first random number (base 10)
1C	D2FF	display V2 (AD)
1E	8BD0	set VB equal to VE the other random number
20	203A	call subroutine to make VA the decimal equivalent of VB
22	DAFF	display VA, second random number
24	D0FF	display V0 (E0)
26	F600	make V6 the entered byte
28	4600	skip the next instruction if V6 is equal to 00
2A	1016	here only if V6 is 00, back to 16 to repeat display
2C	D6FF	display V6, the entered byte
2E	86C5	set V6 equal to V6 - VC, VC is the correct answer (base 10)
30	4600	skip next instruction unless V6 equals 00, i.e. skip on wrong answer
32	100A	transfer to 0A to show AA if answer is correct
34	D1FF	display V1 (EE)
36	DCFF	display VC, correct answer
38	100C	transfer to 0C to begin next problem
-	-	end of main, begin hex to decimal conversion subroutine, subroutine adds 06 to VB for every time 0A occurs, argument is passed in

		VB and returned in VA
3A	8AB0	set VA equal to VB
3C	6906	set V9 equal to 06
3E	680A	set V8 equal to 0A
40	8B85	set VB equal to VB - V8, i.e. subtract 0A from VB
42	4F00	skip next instruction if VF equals 00, i.e. skip unless underflow
44	009E	return from subroutine on underflow
46	8A94	set VA equal to VA + V9, i.e. add 06 to VA
48	1040	transfer to location 40 to subtract 0A from VB, this is the end of the subroutine
-	-	start of machine language subroutine, random numbers from 1 through 50 (base 10) are generated in VD and VE, R(6) is used to point to VD and VE, see interpreter listing for a better understanding of how this routine works
4A	7B	entry point, turn Q on
4B	E6	make R(6) the X register
4C	F8 FE A6	load the address of VE to R(6)
4F	F8 33	load 51 (base 10) to D
51	FF 01	subtract 01 from D
53	32 4F	transfer to 4F if D is zero
55	3F 51	transfer to 51 unless in pushed
57	73	here when in pushed, store number in VE point R(6) to VD
58	F8 32	load 50 (base 10) to D
5A	FF 01	subtract 01 from D
5C	32 58	transfer to 58 if D is zero
5E	37 5A	transfer to 5A unless in released
60	56	store number in VD
61	7A D4	turn Q off and return, end of program

The above program illustrates one of the weaknesses of CHIP-8. There is no way to pass arguments to interpreter subroutines except through the variables and we must execute a number of variable transfer instructions to use the hex to decimal interpreter subroutine. This weakness is partly overcome in the full interpreter by the inclusion of instructions which

transfer the variables to and from memory. The full interpreter also includes an instruction which generates random numbers and a hex to decimal conversion routine. In the next section this program has been rewritten for the full interpreter.

Now let's look at the listing for the demonstration interpreter. It uses the 16 locations F0 through FF to store the 16 variables. The interpreter examines each instruction in turn and carries out the desired operation by calling the correct subroutine. It uses the following registers:

#### Demonstration Interpreter Register Use

R(2)	stack pointer
R(3)	set to address of machine code subroutine that carries out instruction, i.e. subroutine program counter
R(4)	program counter for control section of interpreter
R(5)	program counter for interpretive code
R(6)	VX pointer, points to one of 16 variables
R(7)	VY pointer, points to one of 16 variables
R(C)	used to point to a table of addresses

The interpreter is designed for use on a single page of memory and will work in the basic 1/4K Elf as it stands. For expanded systems R(2), R(3), R(4), R(5), R(6), R(7) and R(C) have to have their high order bytes set to the page the interpreter resides on. Perhaps the simplest way to do this initialization for an expanded system is to change the entry point of the interpreter from 71 to 68 and add the following code from locations 68 through 73:

Add.	Code	Notes
68	F8 00	load page number to D, here 00 but interpreter can be on any page
6A	B2 B3 B4	initialize registers
6D	B5 B6 B7 BC	initialize register
71	F8 68 A2	establish top of stack at M(68) instead of at M(71)

Note that the stack pointer is now initialized at location 68 instead of at location 71. Alternately one can place the interpreter on a higher page in memory, do the initialization of the registers on page 00 and then transfer control to the interpreter. If this method is used the interpretive code can start at location 00 and R(5).0, the

address of the first interpreter instruction, can be set to 00.

### Demonstration Interpreter Listing

Add.	Code	Notes			
71	F8 71 A2	establish stack pointer	92	A7	establish R(7) as VY pointer
74	F8 7A A4	R(4) will be program counter for control section of interpreter	93	0C A3	pick up subroutine address from table and point R(3) to subroutine
77	F8 02 A5	R(5) is program counter for interpretive code, first instruction is at M(02)	95	D3	call subroutine to do instruction
7A	D4	establish program counter for control section	96	30 7B	on return from subroutine go to 7B for next instruction
7B	E2	make R(2) the X register, this is the entry point for return to control section after completing a subroutine call	98	45 30 94	here for machine code subroutine, load address to D and go to 94 to establish R(3)
7C	45 AF	load first half of instruction and save it in R(F).0	-	-	begin subroutine for 6XKK instruction
7E	F6 F6 F6 F6	shift right to get most significant digit most significant digit determines type of instruction	9B	45 56	load KK to D, store in VX
82	32 98	if D is zero (type 0 instruction) we have machine code subroutine call, transfer to location 98	9D	D4	return to control section
84	F9 A0	else or against A0 to get address from table of subroutine locations (see locations A1 to AF)	-	-	9E through A0 is a machine code subroutine that restores R(5) on return from interpreter subroutine
86	AC	save address in R(C).0	9E	42	load return address from stack
87	8F	bring back instruction or against F0 to get VX address	9F	A5 D4	restore R(5) and return the next 15 bytes are the subroutine locations
88	F9 F0	establish R(6) as VX pointer	-	-	i.e. go to B5 for 10MM instructions, go to B0 for 20MM instructions, etc. illegal instructions go to E5 where they are ignored
8A	A6	load second half of instruction, note that R(5) is left pointing to second half of instruction	A1	B5 B0 E5 B8	subroutine for 20MM instructions
8B	05	shift right to get VY pointer	A5	E5 9B E5 C0	load return address to D
8C	F6 F6 F6 F6	or against F0 to get VY address	A9	E5 E5 E5 E5	save on stack, push stack down first
90	F9 F0		AD	E7 E5 DD	restore R(5) so that it points to MM
			-	-	rest of this subroutine is shared with 10MM instructions
			B0	15 85	load MM change R(5) to point to new address
			B2	22 52	return
			B4	25	begin subroutine for 4XKK instruction
			-	-	load KK to D
			B7	D4	make R(6) the X register, the VX pointer
			-	-	x'or VX against KK
			B8	45	
			B9	E6	
			BA	F3	



BB	32 BF	return immediately if D equals 0, i.e. if VX equals KK	-	-	begin DXKK subroutine
BD	15 15	else increment instruction program counter twice	E7	E6	make VX pointer the X register
BF	D4	return	E8	64	display VX
-	-	here begin the 8XYN instructions	E9	45 BF	load KK to R(F).1
C0	45	load YN to D	EB	2F 9F	decrement R(F), load R(F).1
C1	FA 0F	and off N to get 0N in D	ED	3A EB	go to EB unless D is zero, delay loop
C3	3A C8	go to C8 unless N is zero	EF	D4	return – end of interpreter
C5	07 56	load VY, write to VX	F0-FF	-	locations where the 16 interpreter variables are stored
C7	D4	return			
-	-	here on other 8XYN instructions, makes up FN D3 on stack, transfers control to stack and obeys the two instructions, uses R(2) as program counter			
C8	AF	save 0N			
C9	22	push stack down			
CA	F8 D3 73	load D3 to D, write to stack			
CD	8F F9 F0	load 0N, or against F0 to get F1, F2, F3, F4, F5, F6, F7, or FE			
D0	52	write to stack			
D1	E6	make VX pointer the X register			
D2	07	load VY to D			
D3	D2	go to stack to obey FN D3 instructions			
D4	56	on return save result as VX			
D5	F8 FF A6	point R(6) to VF			
D8	F8 00	clear D			
DA	7E 56	shift DF into D and save as VF			
DC	D4	return			
-	-	begin FX00 subroutine			
DD	7B	Q on to indicate waiting for byte			
DE	3F DE	wait for in on			
E0	37 E0	wait for in off			
E2	E6	make VX pointer the X register			
E3	6C	switch byte to VX			
E4	7A	turn Q off			
E5	45 D4	advance instruction counter, return – also used for illegal instructions			

### The Chip-8 Language

This section contains a brief discussion of the CHIP-8 language and a list of the available instructions. Further information about RCA's VIP machine and about CHIP-8 can be found in two articles by Joseph Weisbecker ("COSMAC VIP, the RCA Fun Machine", in the August, 1977 Byte magazine p. 30, and "An Easy Programming System", in the December, 1978 Byte magazine p.108) and in RCA's literature. The full CHIP-8 instruction set is listed in the table at the end of this chapter.

Many of the basic features of the CHIP-8 language are explained and illustrated in section 2 and the demonstration interpreter contains ten instructions which are identical to those in the full CHIP-8 set. The complete language is designed for use with low resolution graphics and the display subroutine is the longest and most complex of the subroutines in the interpreter. A number of TV games have been written with CHIP-8 and it is well suited for this purpose. The display instruction is used in conjunction with a memory pointer and the CHIP-8 variables and has the form DXYN. The values of VX and VY indicate where on the video display to show information, and the value of N indicates how many bytes to display. A memory pointer, called I, gives the starting address of the information to be displayed and must be set by other instructions. Positions in the display field are determined by a rectangular coordinate system with the origin in the upper left corner; 64 horizontal positions, designated by VX and 32 vertical positions designated by VY, are available. The bytes to be displayed are exclusively ored against the display field; an important feature for TV games. Portions of memory bytes which extend beyond the display

field on the right or at the bottom are truncated, there is no wrap around.

Another important feature of the language is the 16 one byte variables, V0 through VF, which are held in random access memory. Two of these variables V0 and VF are used for special purposes. V0 is used in a kind of computed go statement, the BMMM instruction. Control is transferred to location MMM to which has been added the value of V0. As in the demonstration interpreter, VF is used to indicate overflow in arithmetic operations. It is also used to indicate when a display instruction attempts to show a position which is already being displayed. As the display instruction exclusively or's the data to be displayed against the display field, such an attempt turns off the displayed position. VF is set to 01 to indicate this occurrence. This serves as a simple way to determine if a missile has struck a target in a TV game.

A third important feature of CHIP-8, already mentioned in the discussion of the display routine, is the memory pointer, I. The memory pointer can be set both directly and indirectly; besides its use as a display pointer, it also serves as a pointer for transferring variables to and from memory.

The full CHIP-8 instruction set has six skip instructions all of which follow the principle of the skip instruction included in the demonstration interpreter. That is, the next interpreter instruction is skipped over if on testing a condition it is found to be true.

The instructions which have 8 as the first hexadecimal digit perform arithmetic and logic operations and are all included in the demonstration interpreter. Note again that VF is used to indicate overflow and that the value of VF is changed by 8XY1, 8XY2, 8XY3, 8XY4, 8XY5, 8XY6, 8XY7, and 8XYE instructions.

A number of instructions which are not included in the demonstration interpreter are the "F" instructions. Several of these are used in conjunction with the memory pointer. For example the FX29 instruction points I at a 5 byte memory pattern which corresponds to the least significant hex digit of VX. If V7 were 38 and F729 instruction were executed I would point to the first byte of the series F0, 90, F0, 90, F0 (a pattern for the symbol "8") and DXY5 instruction would show an "8" on the display. The FX33 instruction is a binary to decimal conversion routine. The value of VX is converted to a 3 digit decimal number with the hundreds digit stored at location I, the tens digit at location I + 1, and the units digit at location I

+ 2. The FX55 and FX56 instructions use the memory pointer to transfer values from memory to the variables, respectively.

Other "F" instructions include a settable tone generator (FX18) (see the section on Hardware Differences), an instruction to set a timer (FX15), an instruction to read the timer (FX07), and an instruction to read the keyboard (FX0A). An additional "F" instruction has been added for the Elf; FX75, which displays the value of VX on the hex display.

Other useful instructions which are not present in the demonstration interpreter include a random number generator (CXXK where KK is added against a random byte before being transferred to VX), and an instruction which adds a byte to one of the variables, 7XKK. Two of the CHIP-8 instructions 00E0 (erase display) and 0DEE (return from a CHIP-8 subroutine) are implemented as machine code subroutines resident in the interpreter itself. They are therefore dependant upon the page where CHIP-8 is located and will have to be changed if CHIP-8 is relocated. This also is the reason that the return from a subroutine is 009E in the demonstration interpreter and 00EE in the full CHIP-8 interpreter.

To illustrate the use of the full instruction set, let's rewrite one of the programs that used the demonstration interpreter, the one involving addition problems. The following program constructs simple addition problems using two randomly chosen numbers between 0 and 127. On entry to the program a problem is presented, e.g.  $076 + 093 = ?$ . An answer is entered through the keyboard one digit at a time (i.e. 1, 6, 3) and when the last digit is entered 163 is displayed. A C flows the entered number if it is correct and an E if it is incorrect. In the case of an incorrect answer the correct answer is also shown. Another problem is given when any key is entered. The program consists of 67 CHIP-8 instructions and also uses 32 bytes for constants and work space.

#### Program for Addition Problems

Add.	Code	Notes
0200	00E0	erase display
-	-	first set up problems and answer
0202	CD7F	VD equals random number
0204	CE7F	VE equals random number
0206	8CD0	VC = VD
0208	8CE4	VC = VD + VE (the answer)
-	-	next convert to decimal and display the problem
020A	A2A2	point I to work space

020C	6A00	set VA = 00, display pointer	0260	126A	go to 026A
020E	6B00	set VB = 00, display pointer	-	-	here if answer wrong
0210	FD33	M(I) equals 3 digit decimal equivalent of VD	0262	6A15	set VA = 15, display pointer
0212	F265	V0, V1, V2 equals M(I)	0264	6B10	set VB = 10, display pointer
0214	2276	call CHIP-8 subroutine (displays 3 digit number in V0, V1, and V2)	0266	2276	call subroutine to display correct answer
0216	A288	point I to + pattern	0268	660E	V6 = 0E
0218	7A07	VA = VA + 07, display pointer	026A	6A26	VA = 26, display pointer
021A	DAB5	display + pattern	026C	6B08	VB = 08, display pointer
021C	A2A2	point I to work space	026E	F629	point I to C or E pattern
021E	7A08	VA = VA + 08, display pointer	0270	DAB5	display C or E
0220	FE33	M(I) equals 3 digit decimal equivalent of VE	0272	F00A	wait for any input
0222	F265	V0, V1, V2 equals M(I)	0274	1200	to 0200 for next problem
0224	2276	call subroutine to display VE	-	-	subroutine to display 3 digit number held in V0, V1, V2
0226	A28E	point I to = pattern	0276	F029	point I to pattern for V0
0228	7A07	VA = VA + 07, display pointer	0278	DAB5	display it
022A	DAB4	display = pattern	027A	7A05	VA = VA + 05, display pointer
022C	A292	point I to ? pattern	027C	F129	point I to pattern for V1
022E	6A18	set VA = 18, display pointer	027E	DAB5	display it
0230	6B08	set VB = 08, display pointer	0280	7A05	VA = VA + 05, display pointer
0232	DABF	display ? pattern	0282	F229	point I to pattern for V1
-	-	now read in possible answer	0284	DAB5	display it
0234	F00A	V0 = least significant digit of switch byte	0286	00EE	return from subroutine
0236	F10A	V1 = switch byte (LSD)	-	-	patterns and work space
0238	F20A	V2 = switch byte (LSD)	0288	2020	pattern for + sign
023A	DABF	display ? pattern (erases it)	028A	F820	
023C	6A15	set VA = 15, display pointer	028C	2000	
023E	2276	call subroutine to display entered answer	028E	00FF	pattern for ? sign
-	-	now compute answers, right to 025c, wrong to 0262	0290	00FF	
0240	A2A5	point I to work space	0292	FFFF	
0242	F255	V0, V1, V2 – correct answer	0294	0303	
0244	A2A2	V3, V4, V5 – entered answer	0296	03FF	
0246	FC33	V3 = V3 – V0	0298	FFC0	
0248	F565	skip to 0262, error	029A	C0C0	
024A	8305	V4 = V4 – V1	029C	C0C0	
024C	3300	skip if V3 = 00	029E	00C0	
024E	1262	go to 0262, error	02A0	C000	
0250	8415	V4 = V4 – V1	02A2	-	work space
0252	3400	skip if V4 = 00	02A4	-	
0254	1262	go to 0262, error	02A6	-	
0256	8525	V5 = V5 – V2			
0258	3500	skip if V5 = 00			
025A	1262	go to 0262, error			
-	-	here if answer correct			
025C	660C	set V6 = 0C			
025E	F618	set tone duration (reward)			

Table 2

### Full Interpreter Instructions

0MMM	do a machine code subroutine at location 0MMM (The machine code subroutine must end with D4)
1MMM	go to 0MMM; control is transferred to location 0MMM in the interpretive code

2MMM	do an interpreter subroutine at location 0MMM (the interpreter subroutine must end with 00EE)			over if VX does not equal VY (see 5XY0)
3XKK	skip if VX = KK; the next interpreter instruction is skipped over if VX equals KK	AMMM		point I at 0MMM; the memory pointer is set to 0MMM
4XKK	skip if VX $\neq$ KK; the next interpreter instruction is skipped over if VX does not equal KK	BMMM		go to 0MMM + V0, the value of V0 is added to 0MMM and control is transferred to the resulting location
5XY0	skip if VX = VY; the next interpreter instruction is skipped over if VX equals VY (see 9XY0)	CXKK		set VX to a random byte; random byte is anded against KK first
6XKK	set VX = KK; variable X is made equal to KK	DXYN		display N byte pattern at coordinates VX, VY; I (memory pointer) gives starting locations to be displayed. The displayed locations are exclusively ored against display field. VF becomes 01 if some of the display field is already set, 00 if it is not.
7XKK	set VX = VX + KK; add KK to variable X			
8XY0	set VX = VY; variable X is made equal to variable Y			
8XY1	set VX = VX or VY; variable X is made equal to the result of VX logically or'ed against VY (Note that VF is changed)	EX9E		skip if VX = hex key; skip next instruction if the least significant digit of VX equals the least significant digit of the keyboard.
8XY2	set VX = VX and VY; variable X is made equal to the result of VX logically anded against VY (Note that VF is changed)	EXA1		skip if VX $\neq$ hex key; skip next instruction if the least significant digit of VX does not equal the least significant digit of the keyboard
8XY3	set VX = VX xor VY; variable X is made equal to the result of VX logically xor'ed against VY (Note that VF is changed)	FX07		set VX to the value of the timer; timer is counted down in interrupt routine
8XY4	set VX = VX + VY; variable X is made equal to the sum of VX and VY (Note that VF becomes 00 if the sum is less than or equal to FF and 01 if the sum is greater than FF)	FX0A		set VX = hex key; sets VX equal to the least significant digit of the keyboard, waits for in on, off
8XY5	set VX = VX - VY; variable X is made equal to the difference between VX and VY (Note that VF becomes 00 if VX is less than VY and 01 if VX is greater than or equal to VY)	FX15		set timer to VX; timer is counted down in interrupt routine so 01 is ca. 1/60th second
8XY6	set VX = VY shifted right 1 bit position (Note bit 0 is shifted into VF)	FX18		set tone duration to VX; turns Q on for duration specified by VX, 01 is ca. 1/60th second
8XY7	set VX = VY - VX; variable X is made equal to the difference between VY and VX (Note that VF becomes 00 if VY is less than VX and 01 if VY is greater than or equal to VX)	FX1E		set I to I + VX; add the value of VX to the memory pointer
8XYE	set VX = VY shifted left 1 bit position (Note bit 0 is shifted into VF)	FX29		point I to pattern for least significant digit of VX
9XY0	skip if VX $\neq$ VY; the next interpreter instruction is skipped	FX33		convert VX to decimal; 3 decimal digits are stored at M(I), M(I + 1), and M(I + 2), I does not change
		FX55		save V0 through VX in memory at locations specified by I, V0 at M(I), V1 at M(I+1), etc, I becomes I + X + 1
		FX65		transfer memory locations specified by I to variables V0 through VX, V0 becomes M(I), V(1) becomes M(I + 1), etc, I becomes I + X + 1
		FX75		display the value of VX on the hex display

00E0        erase the display (actually a  
             machine language subroutine  
             resident in the interpreter)

### **Hardware Differences between 1802 Computers**

The most important difference between the various versions of the CCOSMAC ELF and the COSMAC VIP is the keyboard. The COSMAC VIP has a hex keyboard; however it is not connected to an input port,. Instead the least significant 4 bits of a bus output byte (Out 2, 62) are decoded and the 16 output lines connected to the corresponding hex keys. Each key is connected to one of the flag lines (EF3). To determine which key is depressed requires a software routine which scans the keyboard. Scanning is done by repeatedly outputting the 16 possible least significant hex digits and examining the flag line to see which digits cause it to be pulled low. Debouncing is also carried out within the software routines; there is an approximately 1/15 second software delay to debounce both opening and closing of a keyboard switch.

COSMAC ELF computers on the other hand are variable in design and have a variety of ways to input information from keyboards or switches. Indeed the September, 1976 issue of Popular Electronics describes a way to connect a scanned hex keyboard, much like that contained in the VIP, to the ELF. However most of the commercially available ELF's (e.g. Super Elf and Elf-2) have latched hex keyboards with roll-over. The latches are connected to an input port and one can examine the contents of these latches at any time under software control. A hardware debounced button (the in button) can be used as a device to indicate to a software routine that we wish the switch latches read. An additional feature of the Elf is the ability to carry out direct memory access input from the keyboard by depressing the in button when the computer is in the load mode. This feature is not required by the VIP which has an operating system in ROM.

These different methods in inputting information from the keyboard have different advantages and disadvantages, neither is really totally satisfactory. The VIP's keyboard has one significant advantage. All of the keys are connected directly to a flag line and it is possible to tell, with software, when a key is being depressed and if so which one. A quick response to keyboard entry is therefore possible and this property is particularly desirable for TV games. It also makes possible an operating system which enters bytes directly from the keyboard to

memory without the necessity of pushing an in button. These features are more difficult with a roll-over latched keyboard like that found in many ELF's. Entered bytes can only be read from latches and there is no way, with software, to determine when a single key is repeatedly entered; that is we could never determine if B, B, B, B was entered because the contents of the latches would never change. This difficulty could, of course, be overcome with some simple hardware changes to the ELF.

The advantage of the ELF keyboard is that the contents of the keyboard latches can be transferred directly to memory by instituting a direct memory access cycle. This, in fact, is what makes the ELF a viable machine without read only memory. However the ELF would be easier to use if the contents of the keyboard latches were displayed and if a signal were provided which made it unnecessary to push the in button.

Another hardware difference is in the treatment of the Q line. In the VIP the Q line is attached to a simple oscillator, and this in turn can be connected to a speaker. Hence in the VIP when the Q line is turned on, a tone is heard in the loudspeaker. This feature can be added to an ELF without much difficulty. It should perhaps be mentioned that the VIP has room on board for one input and one output port, the output uses out-3 (63), and the input port uses in-3 (6B).

Rather than attempt to change the ELF to a VIP by making hardware changes, this booklet accepts the ELF's as they are and makes the software changes in CHIP-8 to accommodate ELF's. Unfortunately ELF's are not built to a standard design like the VIP and it is therefore difficult to write software which will suit all ELF users. To compensate for this a detailed listing of the interpreter is presented in the next section. It is hoped that sufficient information is given so that those with ELF's which differ from commercially available machines will be able to modify the interpreter to suit their machines.

### **A Complete Elf CHIP-8 Interpreter**

This section provides a listing and a discussion of a version of CHIP-8 for COSMAC ELF's. The main listing of the interpreter is designed for a 4K Elf with memory pages 00 through 0F, the configuration most commonly used by the commercially available ELF's. It is also possible to use CHIP-8 in the 1 1/4K ELF's described in the articles in Popular Electronics, but to do so is very tedious unless the switches are replaced with a latched decoded keyboard. This machine has memory pages 00, 04, 05, 06,

and 07 and a version of CHIP-8 for such a machine will also be described. The necessary changes to CHIP-8 will be discussed in the notes included with the full interpreter listing. Similar changes are required when CHIP-8 is relocated in memory and this example may aid those with other styles of machines.

The first consideration in modifying CHIP-8 for use on the ELF is page use. The following page use was chosen for the 4K Elf's with memory pages 00 through 0F:

Page	Use
00	first half of interpreter
01	second half of interpreter
02 – 0D	reserved for interpretive code
0E(first half)	character table and interrupt routine
0E(second half)	variables, work space and stack
0F	display page

This choice of page usage maximizes the similarity of ELF CHIP-8 and VIP CHIP-8. However it is possible to relocate the code to other places in memory and it might be better to accept the changes in CHIP-8 and place the interpreter on pages 0C and 0D. Relocation is necessary to implement the 1 1/4K version. Because of this, some changes in the language are necessary for the 1 1/4K version and the instruction 00E0 becomes 04E0 and 00EE becomes 04EE. Page use for the 1 1/4 K version is as follows:

Page	Use
00	display page
04	first half of interpreter
05	second half of interpreter
06 (first half)	character table and interrupt routine
06 (second half)	variables, work space and stack (There is room for a small operating system in the middle of page 6)
07	interpretive code

Register use is the same as it is in the VIP version of CHIP-8 as follows:

#### Use of Registers

High	Low
R(0)	DMA address
R(1)	interrupt address
R(2)	stack, sometimes X register

R(3)	program counter for interpreter subroutines
R(4)	program counter for control section of interpreter
R(5)	CHIP-8 instruction program counter
R(6)	variable pointer, the VX pointer
R(7)	variable pointer, the VY pointer
R(8)	timer timer
R(9)	random numbers random numbers
R(A)	the I pointer
R(B)	display page pointer
R(C)	used for scratch but available for machine code subroutines
R(D)	used for scratch but available for machine code subroutines
R(E)	used for scratch but available for machine code subroutines
R(F)	used for scratch but available for machine code subroutines

#### Complete CHIP-8 Interpreter Listing

Add.	Code	Notes
-	-	first initialize the registers
0000	F8 0E B1	high order interrupt address
03	F8 46 A1	low order interrupt address replace 00E with 06 for 1 1/4K Elf
06	F8 0F BB	establish display page, replace 0F with 00 for 1 1/4K Elf
09	F8 0E B2	establish a high order stack address replace 0E with 06 for 1 1/4K Elf
0C	B6	establish page for variables, work space (same as stack page)
0D	F8 CF A2	establish low order stack address
10	F8 01 B5	high order address for first CHIP-8 instruction, replace 01 with 05 for 1 1/4K Elf

13	F8 FC A5	low order address for first CHIP-8 instruction, replace FC with FA for 1 1/4K Elf	38	A7	save in R(7).0, the VY pointer
16	F8 00 B4	establish control section program counter, replace 00 with 04 for a 1 1/4K Elf	39	4C B3	interpreter high order subroutine address from table to R(3).1
19	F8 1C A4	establish low order address for control section program counter	3B	8C FC 0F AC	set up pointer to table of low order subroutine addresses
1C	D4	make R(4) the program counter, this ends initialization of registers	3F	0C A3	low order subroutine address from table to R(3).0, R(3) now points to correct interpreter subroutine
-	-	begin control section of interpreter, on return from interpreter subroutine location 1D is entered	41	D3	change to subroutine program counter
1D	96 B7	establish high order VY pointer	42	30 1D	subroutines end with D4, return here and go back to treat another interpreter instruction
1F	E2	establish x-register	-	-	comes to location 44 for machine code subroutines
20	94 BC	make R(C).1 the current page	44	8F	reload 1st byte of CHIP-8 instruction
22	45	load first byte of a CHIP-8 instruction in R(F).1	45	B3	save in R(3).1, high order machine code subroutine address
23	AF	save 1st byte of instruction to R(F).0	46	45	load advance – 2nd byte of interpreter instruction
24	F6 F6 F6 F6	shift right 4 times to get most significant digit	47	30 40	go to location 40 to set R(3).0 and call subroutine
28	32 44	go to 44 if most significant digit is 0, we have a machine language subroutine	-	-	end of control section, except see tables of addresses
2A	F9 50	else or immediate against 50 to make pointer to table of subroutine locations	49	22 69 12 D4	these 4 bytes are a machine code subroutine to turn on 1861 (TV) – obeyed in usual way as a machine code subroutine
2C	AC	save result in R(C).0, the register used as a pointer	4D	00 00 00 00	unused
2D	8F	bring back 1st byte of instruction	-	-	next 15 bytes are high order addresses for interpreter subroutines,
2E	F9 F0	or immediate against F0 to make VX pointer			notes show most significant digit of instruction (note add 04 to each address for 1 1/4K Elf)
30	A6	save in R(6).0, the VX pointer	51	01 01 01 01	1 2 3 4
31	05	load 2nd byte of instruction	55	01 00 01 01	5 6 7 8
32	F6 F6 F6 F6	shift right to get most significant digit	59	01 01 01 01	9 A B C
36	F9 F0	or immediate against F0 to make VY pointer	5D	00 01 01	D E F
			60	00	unused

-	-	low order addresses – same for 1 1/4K Elf	8D	32 F3	to location F3 for housekeeping if all done or if no bytes to display
61	7F 78 86 8E	1 2 3 4			
65	98 FC 00 C2	5 6 7 8			
69	94 F1 B2 DF	9 A B C	8F	27	decrement number of bytes to display
6D	70 9C 05	D E F			
-	-	Now starts the remainder of the interpreter subroutines	90	4A BD	load advance, load display byte and save in R(D).1
-	-	entry to the display subroutine instruction, DXYN, review material in section 3 to see what it does. R(6) is used to point to work space, R(A) is I (the memory pointer), R(7).0 and R(D).0 are used to store N the number of bytes to display, and R(C) is used as pointer in to display page	92	9E	reload VX
			93	FA 07 AE	and against 07, save in R(E).0, this is position in word – say R(A) pointed to a location containing FF (1111 1111) and least significant 3 bits of VX were (011) – routine from here to A9 would make two adjacent work locations (0001 1111) and (1110 0000) i.e. it would shift the word to be displayed over by 3 bits and fill in to the left and right with 0.
70	06 BE	load VX, save in R(E).1			
72	FA 3F	and against 3F (only 64 positions across display field)			
74	F6 F6 F6	shift right 3 times (gets row address, i.e. 0-7 in display page)	96	8E	load word position
			97	32 A2	to A2 if 00, no shift needed
77	22 52	save word address on stack	99	9D F6 BD	shift 1 bit to DF, 0 to MSB of D
79	07	load VY			
7A	FE FE FE	shift left 3 times to make space for row address	9C	8F 76 AF	transfer DF to R(F).0, DF to MSB, LSB to DF
			9F	2E 30 96	repeat number of times in word address
7D	F1	or on row address by setting R(C).1 to display page address	A2	9D 56	save 1st word in work
			A4	16 8F 56	save 2nd word in work
7E	AC	save in R(C).0	A7	16	point R(6) to next work
7F	9B BC	complete address by setting R(C).1 to display page address	A8	30 89	space
			A8	30 89	repeat till all display words treated
81	45	load advance, 2nd half of instruction	AA	00	idles here after housekeeping, sees locations F3 through FB, still here to transfer work to display – R(C) points to first word to change in display field
82	FA 0F	and off number of bytes to display			
84	AD A7	save in R(D).0 and R(7).0			
86	F8 D0	load starting address of work space	AB	EC	make R(C) the X register
88	A6	R(6) now points to work space	AC	F8 D0	load starting address of work
89	F8 00 AF	establish R(F).0 as a source of 00	AE	A6	R(6) points to work
8C	87	load number of bytes to display (a reentry point)	AF	F8 00 A7	00 to R(7).0 and eventually to VF



B2	8D	load number of bytes to display, reenters here until done	DF	00	unused – done with main part of display routine se F3 – FB, a patch for housekeeping entry point for 00E0 instruction (04E0 for 1 1/4k Elf) a machine code subroutine that erases the display page
B3	32 D8	all done?, to D8 to set VF and exit	-	-	load display page address to R(F).1
B5	06	load byte from work			load FF to R(F).0
B6	F2	and against display field			load 00 to D
B7	2D	decrement bytes to display	E0	9B BF	store via F
B8	32 BD	to BD if result of and is 00, i.e. no points already set	E2	F8 FF AF	load R(F).0, return from subroutine if D is 00, all done
BA	F8 01 A7	if points set make R(7).0 and eventually VF, 01	E5	F8 00	else decrement R(F) and go back to blank another memory location
BD	46	reload work to D (load advance)	E7	5F	entry point for 00EE instruction (04EE for 1 1/4k Elf) retrieves interpretive code address from stack
BE	F3	x'or against display field	E8	8F 32 DE	retrieve high order address
BF	5C	write result to display field	EB	2F 30 E5	then low order address R(5) now set
C0	02	reload VX	-	-	return to control section part of display routine, resets memory pointer
C1	FB 07	are we at the end of the row?			load number bytes to display, save in R(7).0
C3	32 D1	if we are quit, no wrap around	EE	42 B5	load R(7).0 to D
C5	1C	else increment R(C)			if 00 done, go to AA to wait for DMA
C6	06	load next word from work	F0	42 A5	decrement R(A) (memory pointer) and R(7)
C7	F2 32 CD	repeat test for already set bits	F2	D4	go back to check if done
CA	F8 01 A7	01 to R(7).0 if bits set	-	-	entry for 6XKK subroutine
CD	06	load from work again	F3	8D A7	load KK to D
CE	F3 5C	x'or against filed and write to field	F5	87	write to VX and return
D0	2C 16	decrement R(C), increment R(6)	F6	32 AA	unused, end of page 00 (04 for 1 1/4k Elf)
D2	8C FC 08	load R(C).0 add 08	F8	2A 27	begin page 01 (05 for 1 1/4k Elf)
D3	AC	load new address to R(C).0			entry for 7XKK subroutine
D6	3B B2	if DF is 0 go to B2 to do more, else we've run over bottom and should return	FA	30 F5	load KK to D
-	-	comes here when all done	-	-	make R(6), VX, the X register
D8	F8 FF A6	load VF address to R(6).0	FC	45	
DB	87 56	load R(7).0 (either 00 or 01) and store in VF	FD	56 D4	
DD	12 D4	fix up stack and return to control section	FF	00	
			0100	45	
			01	E6	

02	F4	add KK to VX	1F	8A	load low order memory
03	56	write result to VX			pointer address
04	D4	return to control section	20	F4 AA	add VX, restore R(A)
-	-	all F instructions enter	22	3B 28	to 28 if DF is zero, no
		here and are set to			overflow, exit
		correct subroutines by	24	9A FC 01	else increment high
		changing R(3)			order I address
05	45	load advance – 2nd byte	27	BA D4	restore it and return
		of F instruction is	-	-	entry for FX29
		location to transfer to			subroutine, table of
		on this page			display patterns is on
06	A3	change R(3) subroutine			page with interrupt
		program counter to			routine, pointer into
		correct address			table are at the
-	-	entry for FX07			beginning of the page
		subroutine	29	91 BA	load interrupt page
07	98	load timer value to D			address to R(A).1
		(see interrupt routine)	2B	06	load VX to D
08	56 D4	write VX and return	2C	FA 0F	and against 0F to get
-	-	entry for FX0A			least significant digit
		subroutine	2E	AA 0A AA	get low order R(A)
0A	3F 0A 37 0C	wait for in on, off			address from table of
0E	22	push down stack			pointers
0F	6C	read switch byte	31	D4	return
10	FA 0F	and against 0F to get	32	00	unused
		least significant digit	-	-	entry for FX33
		(This corresponds to			subroutine (hex to
		original Chip-8, could			decimal conversion)
		and against FF to read	33	E6	make R(6), VX pointer,
		complete byte)			the X register
12	12 56	restore stack, write to	34	06 BF	save VX in R(F).1
		VX	36	93 BE	point R(E) to 011B,
14	D4	return to control section			first
-	-	entry for FX15	38	F8 1B AE	entry of table
		subroutine	3B	2A	decrement memory
15	06	load VX to D			pointer
16	B8 D4	save in R(8).1 and	3C	1A	increment memory
		return			pointer, later enter here
-	-	entry for FX18	3D	F8 00 5A	write 00 to M(R(A))
		subroutine	40	0E	load table entry to D
18	06	load VX to D	41	F5	subtract VX
19	A8 D4	save in R(8).1 and	42	3B 4B	if overflow go to 4B
		return (see interrupt	44	56	else write remainder to
		routine for FX15 and			V6,
		FX18 explanation)	45	0A FC 01 5A	add 01 to M(R(A)),
-	-	the next 3 bytes are	49	30 40	and repeat
		used by the FX33	4B	4E	here if overflow – load
		subroutine			advance table entry
1B	64	100 (base 10)	4C	F6	shift right - if table
1C	0A	10 (base 10)			entry is 01 DF is set
1D	01	1 (base 10)	4D	3B 3C	back to do another digit
-	-	entry for FX1E			unless DF is set
		subroutine	4F	9F 56	here if done - restore
1E	E6	make R(6), VX pointer,			VX
		the X register	51	2A 2A	restore memory pointer
			53	D4	return to control section

54	00	unused	7E	25	restore R(5) to point to
-	-	entry for FX55	-	-	2nd half of instruction
		subroutine transfer			entry for 1MMM
		variables to memory			subroutine rest of code
55	22	push down stack			through location 85 is
56	86 52	load contents of R(6).0			shared
		to stack (one of F0-FF)	7F	45 A5	load MM to D and
58	F8 F0 A7	point R(7) to V0			transfer to R(5).0
5B	07	load V0, on later entry	81	86 FA 0F	retrieve M (most
		V1, etc.			significant part) from
5C	5A	write to M(R(A))			R(6).0
5D	87 F3	load R(7).0 and x'or	84	B5 D4	set R(5).1 and return
		against stack byte -	-	-	entry for 3XKK
		passed VX pointer - if			subroutine - skip if VX
		result is 00 we're done			equals KK
5F	17 1A	increment R(7) and	86	45	load KK to D
		memory pointer	87	E6 F3	make VX pointer X
61	3A 5B	go to 5B to transfer			register, x'or VX
		next VX unless done			against KK
63	12 D4	else restore stack	89	3A 8D	return if D does not
		pointer, return			equal zero
-	-	entry for FX65	8B	15 15	else skip
		subroutine transfer	8D	D4	return to control section
		memory to variables	-	-	entry for 4XKK
65	22	push down stack			subroutine
66	86 52	transfer contents of	8E	45	load KK to D
		R(6).0 to stack, on of	8F	E6 F3	make VX pointer X
		F0-FF			register, x'or VX
68	F8 F0 A7	point R(7) to V0			against KK
6B	0A	load M(R(A)) to D,	91	3A 8B	skip if D does not equal
		enters here later			zero
6C	57	write in V0, V1, V2,	93	D4	else return
		etc.	-	-	entry for 9XY0
6D	87 F3	load R(7).0 and x'or			subroutine, skip if VX
		against stack byte - if			does not equal VY
		result is 00 we're done	94	45	set R(5) to next
6F	17 1A	increment R(7) and			instruction
		memory pointer	95	07	load VY to D
71	3A 6B	go to 5B to transfer	96	30 8F	transfer to 8F to
		next byte unless done			complete instruction
73	12 D4	else restore stack	-	-	entry for 5XY0
		pointer, return			subroutine
-	-	entry for FX75	98	45	set R(5) to next
		subroutine transfer VX			instruction
		to hex display	99	07	load VY to D
75	E6	make VX pointer the X	9A	30 87	transfer to 87 to
		register			complete instruction
76	12 D4	output VX and return			
-	-	entry for 2MMM			
		subroutine, go to			
		interpreter subroutine			
78	15 85	store return interpreter			
		code			
7A	22 73	address on stack			
7C	95 52				

-	-	entry for E subroutine	C5	3A CA	go to CA unless N is zero
		EX9E - skip if VX equals keys (LSD),	C7	07 56 D4	if N is 00 load VY, write to VX, return here on other *XYN instructions, see demonstration
		EXA1 - skip if VX does not equal keys (LSD), see Section 4 Hardware Differences. Designed to be as close as possible to original use in VIP	-	-	interpreter for method used
9C	22	push down stack	CA	AF 22	save 0N in R(F),0, push down stack
9D	6C	switch byte to stack, D	CC	F8 D3 73	load D3, write to stack
9E	06 F3	load VX, x'or against switch byte	CF	8F F9 F0	load 0N, or against F0
A0	FA 0F	and off least significant digit of answer	D2	52	write one of F1, F2, F3, F4, F5, F6, F7, or FE to stack
A2	52	write result to stack	D3	E6	make VX pointer, X register
A3	45 F6	load advance - shift right 0 to DF for EX9E instruction, I to DF for EXA1 instruction	D4	07 D2	load VY and go to stack
			D6	56	on return save result as VX
A5	42	load back stack byte, restore stack	D7	F8 FF A6	point R(6) at VF
A6	3B AD	to AD for EX9E instruction, carry on for EXA1 instruction	DA	F8 00	make D equal 00
			DC	7E 56	shift DF into D and write to VF
A8	3F 8B	skip if in not depressed	DE	D4	return
AA	3A 8B	skip if in depressed but wrong key	-	-	entry for CXKK subroutine, random number generator
AC	D4	else return	DF	19	increment R(9) - random byte - see interrupt routine
AD	3F B1	skip if in not depressed			
AF	32 8B	skip if in depressed but wrong key	E0	89 AE 93 BE	print R(E) to some byte on this page
B1	D4	else return	E4	99	load R(9).1 - random byte from interrupt
-	-	entry for BMMM instruction, go to 0MMM plus V0	E5	EE	make R(E) the X register
B2	F8 F0 A7	point R(7) to V0	E6	F4 56	add the two random bytes, save in VX
B5	E7	make R(7) the X register	E8	76	shift right with carry - scramble D
B6	45	load MM	E9	E6	make VX pointer the X register
B7	F4	add V0 and D	EA	F4 B9	add, use result to change R(9).1 as it isn't changed often in interrupt routine
B8	A5	save it in R(5).0			
B9	86 FA 0F	load R(6).0 to retrieve most significant part of MMM, and off	EC	56	save result to VX
BC	3B C0	to C0 if no overflow on addition, all done	ED	45 F2	load KK and and against VX
BE	FC 01	else add 01 to D	EF	56 D4	save result as VX and return
C0	B5 D4	set R(5).1 and return	-	-	entry for AMMM subroutine, set I pointer
-	-	entry for 8XYN instructions, identical to those in demonstration interpreter			
C2	45	load YN to D			
C3	FA 0F	and off N to get 0N			

F1	45 AA	load MM - transfer to R(A).0	24	F0 80	start 6 display
F3	86 FA 0F	retrieve M from R(6),o (MSD)	26	F0 90	start 8 display
F6	BA	complete memory pointer	28	F0 90	start 9 display
F7	D4	end of interpreter subroutines	2A	F0 10	start 3 display
-	-	remaining 8 locations are used for interpretive code, starting address of interpretive code is 01	2C	F0 10	
		FC for 4k interpreter, 05FA for 1 1/4 interpreter	2E	F0 90	start A display
		unused, this is 4K version	30	F0 90	start 0 display
F8	00 00	unused	32	90 90	
FA	00 00	erase display page	34	F0 10	start 7 display
FC	00 E0	turn on TV	36	10 10	
FE	00 49	start of interpreter code for 1 1/4k version	38	10 60	start 1 display (starts at 39)
02 00	-	unused	3A	20 20	
-	-	unused	3C	20 70	
05 F8	00 00	erase display page	3E	A0 A0	start 4 display
FA	04 E0	turn on TV	40	F0 20	
FC	04 49	transfer to page 7 for interpreter code	42	20	end of display characters
FE	17 00		-	-	begin interrupt routine, entry point is 0E 46 (06 46 for 1 1/4k Elf)
			43	7A	Q (tone) off
			44	42 70	restore D and return from interrupt
			46	22	push stack down, entry to interrupt
			47	78 22 52	save X, P; push, save D
			4A	C4	no op, necessary 3 cycle instruction

#### Character Table and Interrupt Routine

Add.	Code	Notes			
-	-	This code could go on any page, as written it is on page 0E for the 4k version and page 06 for the 1 1/4k version	4B	19	increment R(9), random number (see instruction CXKK)
-	-	first 16 bytes are pointers for the characters 0-F	4C	F8 00 A0	set low order address of DMA pointer
0E 00	30 39 22 2A	pointers to 0, 1, 2, 3	4F	9B B0	set high order address of DMA pointer
04	3E 20 24 34	pointers to 4, 5, 6, 7	51	E2 E2	make up necessary 29 machine cycles
08	26 28 2E 18	pointers to 8, 9, A, B	53	80 E2	load R(0).0 to D
0C	14 1C 10 12	pointers to C, D, E, F	-	-	DMA 1
-	-	next 51 bytes are the display symbols for the characters, 5 bytes/symbol	55	E2 20 A0	restore DMA address
10	F0 80	start E display	-	-	DMA 2
12	F0 80	start F display	58	E2 20 A0	restore DMA address
14	F0 80	start C display	-	-	DMA 3
16	80 80		5B	E2 20 A0	restore DMA address
18	F0 50	start B display	-	-	DMA 4
1A	70 50		5E	3C 53	continue till done
1C	F0 50	start D display	60	9B	R(8).1 is timer, load it (see FX07 and FX15 instructions)
1E	50 50		61	32 67	if D is zero go to 67, timer is timed out, leave alone
20	F0 80	start 5 display			
22	F0 10	start 2 display			

63	AB 2B 8B B8	else subtract 01 from timer, method used does not disturb the DF flag. DF is not changed by the interrupt routine
67	88	load R(8).0 tone duration, see FX18 instruction
68	32 43	if tone duration is over go to 43
6A	7B	continue with or start tone
6B	28	decrement R(8).0, tone duration
6C	30 44	return, leaving tone on end of interpreter
-	-	

#### Extending the CHIP-8 Instruction Set

The CHIP-8 interpreter is well organized and constructed and as a result it is easy to modify and extend. If a specific task, for example the control of a robot, is to be programmed the interpretive language can be changed to suit the application. Let's look at how we might extend the current CHIP-8 instructions. There are two main types of instructions one might wish to add, those which involve pointers to two of the CHIP-8 variables, (e.g. like 8XYN) and those which require a pointer to a single CHIP-8 variable (e.g. 6XKK).

The first group of instructions might be created by expanding either the 5XY0 instruction or the 9XY0 instruction. Say we chose to expand the 5XY0 instruction. The entry point for the 5XY0 instruction would be changed to point to a third CHIP-8 page. The least significant hex digit of the instruction would be examined and if it was 00 the instruction would have its usual meaning. However if the last hex digit was 1, 2, etc., the new operations would be performed.

As an example let's expand the 5XY0 instruction to the following set:

5XY0	skip if VX=VY; the next interpreter instruction is skipped over if VX equals VY (original meaning)
5XY1	skip if VX>VY; the next interpreter instruction is skipped over if VX is greater than VY
5XY2	skip if VX<VY; the next interpreter instruction is skipped over if VX is less than VY
5XY3	skip if VX≠VY; the next interpreter instruction is skipped over if VX does not equal VY

We will place the new subroutines in the middle of page 0E between the interrupt routine

and the bottom of the CHIP-8 stack. The entry point of the new interpreter subroutine will be 0E 70 (06 70 for the 1 1/4k Elf). CHIP-8 must be modified so that the 5 instructions transfer control to this address in the interpreter. Replace the 01 at location 00 55 with 0E (06 in the corresponding place for the 1 1/4k Elf) and replace the 98 at location 00 65 with 70.

#### Additional Skip Instructions Expansion of the 5XY0 Instruction

Add.	Code	Notes
0E 70	93 BC	set R(C).1 to current page
72	45	load advance 2nd CHIP-8 byte, now VY and off 00, 01, 02, or 03 depending on instruction
73	FA 03	add starting address of table of locations
75	FC 7D	point R(C) to proper entry in table
77	AC	pick up table entry, point R(C) to proper subroutine address
78	0C AC	load VY, make R(6) the X register
7A	07 E6	go to one of four subroutines
7C	DC	address for 5XY0 instruction
7D	81	address for 5XY1 instruction
7E	8B	address for 5XY2 instruction
7F	8F	address for 5XY3 instruction
80	87	address for 5XY0 instruction
-	-	entry for 5XY0
81	F3	x'or VX against VY
82	3A 86	return if D does not equal 00
84	15 15 D4	else skip and return
-	-	entry for 5XY3
87	F3	x'or VX against VY
88	3A 84	skip if D does not equal 00
8A	D4	else return
-	-	entry for 5XY1
8B	F7	subtract VX from VY
8C	3B 84	skip if DF equals zero
8E	D4	else return
-	-	entry for 5XY2
8F	F5	subtract VY from VX
90	3B 84	skip if DF equals zero

92 D4 else return, end of  
5XYN subroutines

Among the instructions that the interpreter lacks are simple multiply and divide instructions to go along with its addition and subtraction instructions. Let's expand the 9XY0 instruction to add these instructions to CHIP-8. Multiply and divide instructions are necessarily 16 bit ones, the product of two 8 bit numbers may be up to 16 bit long and of course we need 16 bits to represent the quotient and remainder from the division of two 8 bit numbers. An additional variable will be required to hold the most significant byte from a multiplication and the remainder from a division. VF is already a special variable and will be used to hold the most significant [art of the product in a multiplication and the remainder in division. As well it would be nice to be able to represent the product of a multiplication as a decimal number and a 16 bit hex to decimal conversion routine will also be added.

The new "9" instructions will be located starting at the beginning of page 0D and we shall have to change the address of the "9" instructions in the interpreter. Memory location 00 59 should be changed from 01 to 0D and memory location 00 69 should be changed from 94 to 00.

The new instructions are:

9XY0 skip if VX≠VY; the next interpreter instruction is skipped over VX does not equal VY (unchanged)  
9XY1 set VF, VX equal to VX times VY where VF is the most significant part of a 16 bit word  
9XY2 set VX equal to VX divided by VY where VF is the remainder  
9XY3 let VX, VY be treated as a 16 bit word with VX the most significant part and convert to decimal; 5 decimal digits are stored at M(I), M(I+1), M(I+2), M(I+3), and M(I+4), I does not change

#### Multiply, Divide and 16 Bit Display Instructions Expansion of 9XY0 Instruction

Add.	Code	Notes
0D 00	93 BC	set R(C).1 to current page
02	45	load 2nd CHIP-8 byte, YN
03	FA 03	and off 00, 01, 02, or 03
05	FC 18	add starting address of table of locations

07	AC	point R(C) to proper entry in table
08	0C AC	pick up table entry , point R(C) to proper subroutine address before calling
-	-	subroutine get ready for multiply and divide
0A	E7	R(7), VY pointer the X register
0B	96 BE	point R(E) to VF
0D	F8 FF AE	
10	F8 00 5E	set VF to 00
13	F6	clear DF flag
14	F8 09 AD	initialize counter for shifts to 09
-	-	now call subroutines
17	DC	go to one of 4 subroutines
18	80	address for 9XY0 instruction
19	1C	address for 9XY1 instruction, multiply
1A	2D	address for 9XY2 instruction, divide
1B	46	address for 9XY3 instruction, hex to decimal conversion
-	-	multiply routine entry, works by shift and add method like pencil and paper multiplication
1C	0E 76 5E	shift double length
1F	06 76 56	bit to the left
22	2D 8D	decrement and load counter
24	32 34	done when counted out
26	3B 1C	back if DF is 00, nothing to add
28	0E F4 5E	else add VY to VF, before going back
2B	30 1C	end of multiply routine, begin divide routine -
-	-	first check for division by zero
2D	07	load VY to D
2E	3A 35	if not equal to zero go on
30	F8 FF	else set quotient and remainder to FF and return
32	56 5E D4	here if divisor greater than 0, division method similar to multiplication
-	-	load VF, subtract VY to 3A on overflow
35	0E F7	
37	3B 3A	

39	5E	else save result in VF	76	10 27	10000 (base 10)
3A	06 7E 56	shift one bit left			2710 (base 16)
3D	2D 8D	decrement, load counter	78	E8 03	1000 (base 10)
3F	32 34	return when counted			03E8 (base 16)
		out	7A	64 00	100 (base 10)
41	0E 7E 5E	shift one bit left			0064 (base 16)
44	30 35	return to 35 for next	7C	0A 00	10 (base 10)
		subtraction			000A (base 16)
-	-	entry to 9XY3	7E	01 00	1 (base 10)
		subroutine, hex to			0001 (base 16)
		decimal conversion (5	-	-	entry for (XY0
		decimal digits) method			subroutine (original
		is similar to that for			instructions)
		FX33 instruction	80	07	load VY
46	06 BF	save VX	81	E6	make VX pointer the X
48	07 AF	save VY			register
4A	9C BE	point R(E) to 1 less	82	F3	x;or VY against VX
		than starting address	83	3A 86	if D not equal to zero,
4C	F8 75 AE	of table of powers of 10			skip
4F	2A	decrement memory	85	D4	else return
		pointer	86	15 15 D4	skip and return
50	1A 1E	increment memory			
		pointer, table pointer			
52	F8 00 5A	set memory pointer			
		location to 00			
55	E7	VY pointer (least			
		significant byte) is the			
		X register			
56	4E F5	load table entry,			
		subtract from VY			
58	E6	VX pointer (most			
		significant byte) is the			
		X register			
59	0E 75	load table entry,			
		subtract with carry			
5B	2E	decrement table pointer			
5C	3B 69	to 69 if overflow done			
		with this digit			
5E	56	else update VX			
5F	E7 0E F5 57	and update VY			
63	0A FC 01 5A	increment memory			
		pointer location			
67	30 55	and go back till			
		overflow			
-	-	here on overflow			
69	4E F6	load table entry, check			
		for done			
6B	3B 50	if not done to 50 for			
		next digit			
-	-	here when done			
6D	9F 56	restore VX			
6F	8F 57	restore VY			
71	2A 2A 2A 2A	restore memory pointer			
75	D4	return			
-	-	table entries			

If one has an ASCII device connected to an ELF, perhaps a keyboard, it would be convenient to have a CHIP-8 instruction which would create symbols for the characters in ASCII code. Such an instruction is presented last, the FX94 instruction. This instruction uses the space left unused in the interpreter by the expansion of the "5" and "9" instructions and creates symbols for the 64 characters in 6 bit ASCII. In operation it works exactly like the FX29 instruction except that the memory pointer is set to the address of one of the 64 ASCII symbols corresponding to VX instead of to the address of one of the 16 symbols 0-F. If the "5" and "9" instructions have not been expanded this instruction can, as well, replace the FX29 instruction and ways to implement either alternative will be given.

The instruction fits on a single page; each of the 64 ASCII symbols are coded by 3 bytes which requires 192 memory locations and the remainder of the subroutine fits in the 64 locations remaining. The construction of this instruction is quite simple. The first 16 locations on the page are patterns which are available to construct the symbols. Each ASCII symbol is designated by 5 hex digits which correspond to the patterns needed to construct the symbol. The sixth hex digit in the three words used to code each symbol serves as an indicator of the length of the symbol. When an FX94 (FX29) instruction is carried out this value is transferred to V0 where it can be used to get a pleasing spacing of the symbols.



The symbols are relatively crude, both because of the poor resolution of the ELF graphics and also because they consist of combinations of only 16 patterns. However they are easily recognized and make the presentation of ASCII data relatively with the aid of a very simple interpreter program.

The method used to transfer control from the interpreter to the new subroutine is to change the program counter from R(3) to R(C). This change has to be done in the interpreter and the address of the new subroutine must first be loaded to R(C). If the ASCII subroutine is located on page 0C the proper entry point is 0C D0. To make an FX94 instruction add the following code to the interpreter on page 01 (4k version):

Add.	Code	Notes
01 94	F8 D0 AC	point R(C).0 to D0
97	F8 0C BC	point R(C).1 to page 0C
9A	DC	make R(C) the program counter

This code overwrites the locations which were used for the "5" and "9" instructions. The same code, but located starting at address 01 29, would change the FX29 instruction to the ASCII instruction.

#### Six-Bit ASCII Symbols Subroutine

Add.	Code	Notes
-	-	subroutine can reside on any page, here it is on page 0C
-	-	the first 16 locations are the patterns available to make up the symbols (blank)
0C 00	00	
01	10	*
02	20	*
03	88	* *
04	A8	* * *
05	50	* *
06	F8	*****
07	70	***
08	80	*
09	90	* *
0A	A0	* *
0B	B0	* * *
0C	C0	**
0D	D0	** *
0E	E0	***
0F	F0	****
-	-	locations 10 through CF are codings for the 64 ASCII symbols, 3 bytes to a symbol

- -

10	46 3E 56
13	99 9F 4F
16	5F 57 4F
19	8F 88 4F
1C	5F 55 4F
1F	8F 8F 4F
22	88 8F 4F
25	9F 8B 4F
28	99 9F 49
2B	27 22 47
2E	AE 22 47
31	A9 AC 49
34	8F 88 48
37	43 64 53
3A	99 DB 49
3D	9F 99 4F
40	88 9F 4F
43	9F 9B 4F
46	A9 9F 4F
49	1F 8F 4F
4C	22 22 56
4F	9F 99 49
52	22 55 53
55	55 44 54
58	53 52 53
5B	22 52 53
5E	CF 12 4F
61	8C 88 3C
64	10 C2 40
67	2E 22 3E
6A	30 25 50
6D	06 00 50
70	00 00 40
73	0C CC 2C
76	00 50 45
79	65 65 55
7C	46 46 56

A diagram giving the order in which the patterns are assembled from the bytes is:

XX XX XX

45 23 61

where the 6th hex digit contains the width of the character, at most 5 bits. The first ASCII character (hex 00) is @, its coding is 46, 3E, 56 which gives:

pattern 6 - \*\*\*\*\*  
 pattern 3 - \* \* \*  
 pattern E - \*\*\*  
 pattern 4 - \* \* \* \*  
 pattern 6 - \*\*\*\*\*

The character is 5 bits long

00 - @

01 - A

02 - B

03 - C

04 - D

05 - E

06 - F

07 - G

08 - H

09 - I

0A - J

0B - K

0C - L

0D - M

0E - N

0F - O

10 - P

11 - Q

12 - R

13 - S

14 - T

15 - U

16 - V

17 - W

18 - X

19 - Y

1A - Z

1B - [

1C - \

1D - ]

1E - ^

1F - \_

20 - space

21 - !

22 - "

23 - #

24 - \$

7F	DF BF 4F	25 - %	-	-	entry point for
82	5F AF 4E	26 - &			successive table bytes
85	00 80 18	27 - '	E6	0D FA 0F	load table entry, and off
88	21 22 41	28 - (			least significant digit
8B	12 11 42	29 - )	E9	A3	point R(3) to correct
8E	53 56 53	2A - *			entry in table of
91	22 26 52	2B - +			patterns (small table)
94	2E 00 30	2C - ,	EA	03 73	pick up pattern, write to
97	00 06 50	2D - -			random access memory,
9A	CC 00 20	2E - .			decrement I
9D	C0 12 40	2F - /	EC	4D	pick up byte again, this
A0	9F 99 4F	30 - 0			time advance R(D)
A3	22 22 32	31 - 1	ED	F6 F6 F6 F6	shift right to get most
A6	8F 1F 4F	32 - 2			significant digit
A9	1F 1F 4F	33 - 3	F1	A3	point R(3) to correct
AC	22 AF 4A	34 - 4			entry
AF	1F 8F 4F	35 - 5	F2	8A	load R(A).0
B2	9F 8F 4F	36 - 6	F3	FB 9A	check, have we done 5
B5	11 11 4F	37 - 7			patterns?
B8	9F 9F 4F	38 - 8	F5	32 FB	if D is 00 we're done,
BB	1F 9F 4F	39 - 9			go to set V0 and return
BE	80 80 10	3A - :	F7	03 73	else pick up pattern,
C1	2E 20 30	3B - ;			write to random access
C4	21 2C 41	3C - <			memory
C7	E0 E0 30	3D - =	F9	30 E6	and return for next table
CA	2C 21 4C	3E - >			entry
CD	88 1F 4F	3F - ?	-	-	here on return
-	-	end of character table,	FB	83	retrieve length of
		entry point for ASCII			symbol from R(3).0
		display subroutine	FC	57	write to V0
-	-	first point R(a),	FD	1A D4	fix up R(A) and return
		memory pointer to a			
		scratch place in random			
		access memory - here at			
		bottom of stack			
D0	F8 0E BA	point R(A).1 to page 0E			
D3	F8 9F AA	point R(A).0 to 9F, just			
		below stack, R(A).0			
		points to 9B when			
		returning from			
		subroutine			
D6	9C	load page number to D			
D7	B3 BD	point R(3).1 and R(D).1			
		to this page			
D9	F9 F0 A7	point R(7) to V0			
DC	Ea	make R(A), memory			
		pointer, the X register			
DD	06 FA 3F	load VX, and off 6 bits			
E0	5A F4 F4	write to M(R(X)), add			
		twice to get number			
		times 3			
E3	FC 10	add starting address of			
		character table			
E5	AD	R(D) now points to			
		correct location in large			
		table			

The reader would probably like to see what these characters look like when displayed. Here is an interpretive program which can be used to display all of the ASCII symbols. The program waits for a switch byte (0-F) and when it is entered displays the corresponding ASCII symbol in the upper left of the screen followed by as many ASCII symbols as the screen has room for. If the byte in the interpreter (4K) at location 01 11 is changed from 0F to FF complete switch bytes (00-FF) can be entered.

#### Program to Display ASCII Characters

Add.	Code	Notes
0200	F50A	V5 equals keys - waits
		for in button
02	6600	V6 = 00
04	6700	V7 = 00, display
		pointers
06	6B3F	VB = 3F, line length
08	F594	(F529?) set I to V5
		ASCII symbol, V0 =
		symbol length
0A	7501	V5 = V5 + 01

0C	D675	display the symbol at V6, V7
0E	8604	V6 = V6 + V0
10	7601	V6 = V6 + 01, space between symbols
12	8D60	VD = V6
14	F594	(F529?) set I, V0 for next symbol
16	8DD4	VD = VD + V0, add length of next symbol to VD
18	8DB5	VD = VD - VB, check will it extend past line end?
1A	3F01	skip if VF is 01, over the end of line
1C	1208	O.K. go back and display
1E	6600	reset to new line
20	7706	V7 = V7 + 06, set line down
22	471E	skip unless V7 is 1E, we're off bottom
24	1224	stop - screen is full
26	1208	return to do another line

to the interpreter, if less than 4k of memory is available it can be ignored.

It is hoped that these examples demonstrate the ease with which the CHIP-8 interpreter can be extended and modified. One of the limitations of CHIP-8, the fact that only memory locations 0000 through 0FFF are available to it, can be overcome by redesigning the interpreter to address memory in 4k fields. A field designation instruction is used to change from one 4k field to another. A relocatable 1k interpreter which includes all of the material presented in this booklet, as well as a field instruction, is listed in the Appendix. The field instruction is a four byte one which has the form, FFFF, MMMM. M is the new field and MMM is the address of the first instruction to be obeyed in the new field. For example to transfer to a new field:

Add.	Code	Notes
0FD0	6300	set V3 to 0D
D2	6400	set V4 to 00
D4	650a	set V5 to 0A
D6	FFFF	field instruction go to
D8	1004	field 1, 004
-	-	
10 04	F529	point to symbol for A
06	D345	display A
-	-	etc.

More ambitious programs can be written with the 4K memory restraint removed. The field designation is stored in R(B).0 and is set on entry

## Appendix

The interpreter listed below is relocatable and can be placed on any four contiguous pages (e.g. 0A00 - 0DFF for 4k Elf). It *must* be entered with R(3) as the program counter. Enter at location 0000 for default values for the first interpreter instruction (01FE), the display page (0F), and the page for variables and constants (0E). To change the default values set R(5) to the address of the

first interpreter instruction, set R(B).1 to the display page, set R(6).1 to the page for variables and constants, and enter the interpreter at location 000C. The default value for the location of the first interpreter instruction (01FE) allows space for an erase display instruction (00E0) before a program which starts at location 0200. The FX29 instruction in this interpreter does not alter the value of V0.

0000	F8 01 B5 F8	FE A5 F8 0F	0100	45 E6 F4 56	D4 45 A3 98
0008	BB F8 0E B6	95 FA F0 AB	0108	56 D4 3F 0A	37 0C 22 6C
0010	96 B2 F8 CF	A2 E3 70 23	0110	FA 0F 12 56	D4 06 B8 D4
0018	93 B4 FC 02	B1 F8 D3 A1	0118	06 A8 D4 64	0A 01 E6 8A
0020	F8 25 A4 69	D4 96 B7 45	0120	F4 AA 3B 28	9A FC 01 BA
0028	AF F6 F6 F6	F6 32 4D FC	0128	D4 F8 B0 30	8E 00 00 00
0030	69 AC 8F F9	F0 A6 05 F6	0130	15 15 D4 E6	06 BF 93 BE
0038	F6 F6 F6 F9	F0 A7 94 BC	0138	F8 1B AE 2A	1A F8 00 5A
0040	EC F4 B3 8C	FC 0F AC 0C	0140	0E F5 3B 4B	56 0A FC 01
0048	A3 E2 D3 30	25 8F 32 54	0148	5A 30 40 4E	F6 3B 3C 9F
0050	B3 45 30 48	94 FC 02 B3	0150	56 2A 2A D4	00 22 86 52
0058	05 FB EE 32	66 FB 0E 32	0158	F8 F0 A7 07	5A 87 F3 17
0060	64 8F 30 50	FC 05 FC 07	0160	1A 3A 5B 12	D4 22 86 52
0068	30 48 01 01	02 02 02 02	0168	F8 F0 A7 0A	57 87 F3 17
0070	01 01 02 01	01 01 00 01	0170	1A 3A 6B 12	D4 E6 64 D4
0078	01 7F 78 1B	1F 27 23 00	0178	15 95 22 73	85 52 25 45
0080	C4 4F F3 AD	E1 88 96 05	0180	A5 86 FA 0F	22 52 8B F1
0088	06 BE FA 3F	F6 F6 F6 22	0188	B5 12 D4 00	F8 C0 AC 93
0090	52 07 FE FE	FE F1 AC 9B	0190	FC 02 BC DC	30 BC 22 6C
0098	BC 45 FA 0F	AD A7 F8 D0	0198	06 F3 FA 0F	52 45 F6 42
00A0	A6 F8 00 AF	87 32 F7 27	01A0	3B A7 3F 30	3A 30 D4 3F
00A8	4A BD 9E FA	07 AE 8E 32	01A8	AB 32 30 D4	00 F8 F0 A7
00B0	BA 9D F6 BD	8F 76 AF 2E	01B0	E7 45 F4 A5	86 FA 0F 3B
00B8	30 AE 9D 56	16 8F 56 16	01B8	BB FC 01 E2	22 52 8B F1
00C0	30 A1 00 EC	F8 D0 A6 F8	01C0	B5 12 D4 00	45 FA 0F 3A
00C8	00 A7 8D 32	F0 06 F2 2D	01C8	CC 07 56 D4	AF 22 F8 D3
00D0	32 D5 F8 01	A7 46 F3 5C	01D0	73 8F F9 F0	52 E6 07 D2
00D8	02 FB 07 32	E9 1C 06 F2	01D8	56 F8 FF A6	F8 00 7E 56
00E0	32 E5 F8 01	A7 06 F3 5C	01E0	D4 19 89 AE	93 BE 99 EE
00E8	2C 16 8C FC	08 AC 3B CA	01E8	F4 56 76 E6	F4 B9 56 45
00F0	F8 FF A6 87	56 12 D4 8D	01F0	F2 56 D4 45	AA 86 FA 0F
00F8	A7 87 32 C2	2A 27 30 F9	01F8	22 52 8B F1	BA 12 D4 45

0200	22 73 FA F0	AB 05 52 42	0340	F1 F8 22 22	F6 99 99 22
0208	A5 42 B5 D4	15 9B BF F8	0348	55 53 45 44	53 52 23 22
0210	FF AF F8 00	5F 8F 32 0B	0350	35 CF 12 CF	88 C8 10 C2
0218	2F 30 12 45	E6 30 38 45	0358	E0 22 E2 30	25 60 00 00
0220	E6 30 3E 45	56 D4 00 93	0360	00 00 C0 C0	CC 00 50 55
0228	BC 45 FA 03	FC 34 AC 0C	0368	56 56 46 46	F6 FD FB 5F
0230	AC 07 E6 DC	38 42 46 3E	0370	AF 0E 00 88	21 22 21 11
0238	F3 3A 3D 15	15 D4 F3 3A	0378	21 53 56 23	62 22 2E 00
0240	3B D4 F7 3B	3B D4 F5 3B	0380	00 6D 00 CC	00 00 2C 01
0248	3B D4 07 E6	30 3E 00 93	0388	9F 99 2F 22	22 8F 1F FF
0250	BC 45 FA 03	32 4A FC 68	0390	F1 F1 22 AF	FA F1 F8 9F
0258	AC 0C AC E7	96 BE F8 FF	0398	8F 1F 11 F1	9F 9F FF F1
0260	AE F8 00 5E	F6 F8 09 AD	03A0	F9 80 80 E0	02 02 21 2C
0268	DC 6C 7D 96	0E 76 5E 06	03A8	01 0E 0E 2C	21 8C F8 F1
0270	76 56 2D 8D	32 84 3B 6C	03B0	06 AF FA 0F	F9 30 56 FD
0278	0E F4 5E 30	6C 07 3A 85	03B8	39 33 C2 FD	40 56 30 C2
0280	F8 FF 56 5E	D4 0E F7 3B	03C0	06 AF 96 BA	F8 9F AA 9C
0288	8A 5E 06 7E	56 2D 8D 32	03C8	B3 BD EA 06	FA 3F 5A F4
0290	84 0E 7E 5E	30 85 06 BF	03D0	F4 F4 F4 76	3B DB FC 10
0298	07 AF 9C BE	F8 C5 AE 2A	03D8	AD 30 E9 FC	10 AD 0D FA
02A0	1A 1E F8 00	5A E7 4E F5	03E0	0F A3 8A FB	9A 32 F8 03
02A8	E6 0E 75 2E	3B B9 56 E7	03E8	73 4D F6 F6	F6 F6 A3 8A
02B0	0E F5 57 0A	FC 01 5A 30	03F0	FB 9A 32 F8	03 73 30 DE
02B8	A5 4E F6 3B	A0 9F 56 8F	03F8	8F 56 1A D4	00 00 00 00
02C0	57 2A 2A 2A	2A D4 10 27			
02C8	E8 03 64 00	0A 00 01 00			
02D0	7A 42 70 22	78 22 52 C4			
02D8	19 F8 00 A0	9B B0 E2 E2			
02E0	80 E2 E2 20	A0 E2 20 A0			
02E8	E2 20 A0 3C	E0 22 76 52			
02F0	98 32 F6 FF	01 B8 42 7E			
02F8	88 32 D0 7B	28 30 D1 00			
0300	00 10 20 88	A8 50 F8 70			
0308	80 90 A0 B0	C0 D0 E0 F0			
0310	46 3E 96 F9	F9 5F 57 FF			
0318	88 F8 5F 55	FF F8 F8 88			
0320	8F FF B9 F8	99 9F 79 22			
0328	72 AE 22 97	CA 9A 8F 88			
0330	38 44 36 99	DB F9 99 F9			
0338	88 9F FF B9	F9 A9 9F FF			

### **Notes**

*The FX00 and FX75 instructions cause failures when X is F because R(6) "turns" a page; R(6) should be decremented after the use of an output (64) instruction.*

*When using the relocatable interpreter place all the machine code subroutines in field 0 (0000 to 0FFF); they are accessible to calls from any of the 16 fields.*

## Notes

Additional copies of this booklet can be ordered from:

Paul C. Moews  
16 B Yale Road  
Storrs, CT 06268

The price, \$5.50, includes first class postage and handling.

Two other booklets with programs for the basic 1/4k Elf are also available:

1. Music and Games
  2. Graphics
- for \$3 each, postpaid.