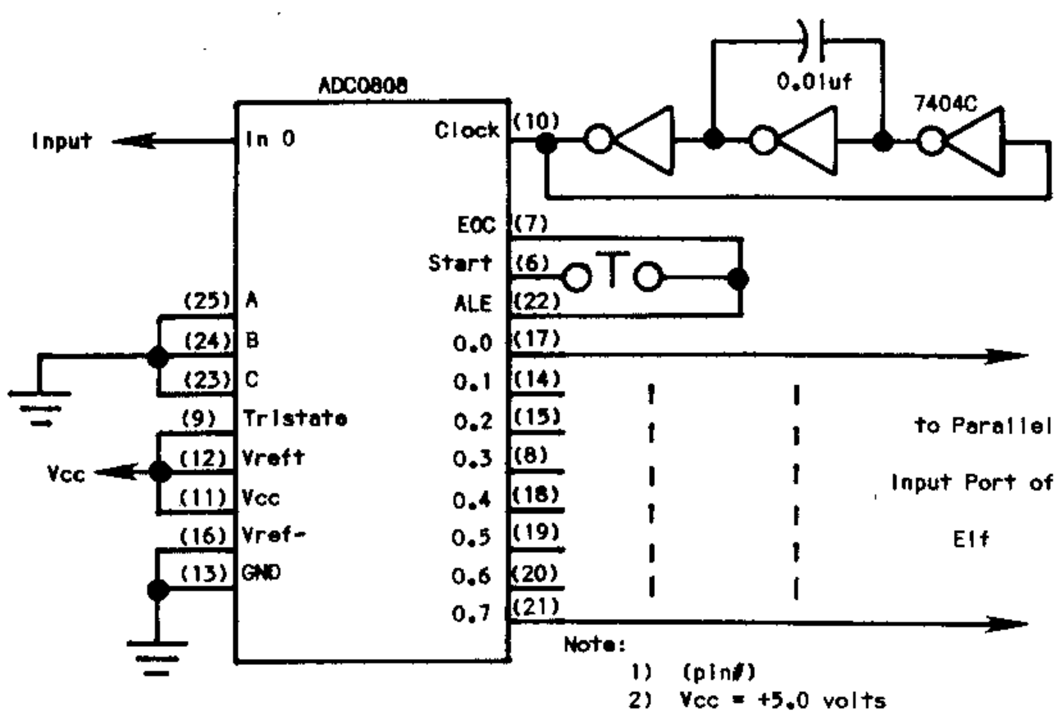
Questdata
Volume 2 Issue # 8 ©

AN ANALOG TO DIGITAL CONVERTER FOR THE SUPER ELF

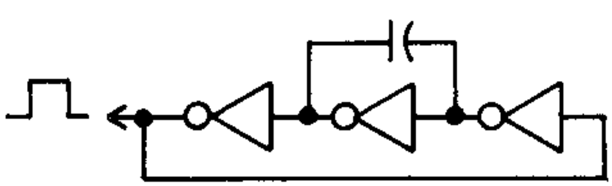
by
Phillip B. Liescheski III

An analog to digital (A/D) converter is a very useful device in processing empirical data which have been obtained in a laboratory. Many instruments found in the laboratory produce an analog signal for their output or readout. This data in analog form usually must be converted to numbers in order to process them. In digital form, the data is more easily manipulated by mathematical means. To increase the speed of data processing or for the purpose of automation, a computer can be used to treat the data. This is the purpose of the A/D converter. It converts the analog signal of the instrument into information which can be used by a computer. This device is an interface which bridges the gap between the worlds of analog instruments and the digital computer. Its use can greatly enhance the speed and accuracy of obtaining data and most important, it automates this task.



The heart of this A/D converter is the National Semiconductor ADC0808 monolithic CMOS integrated circuit. This integrated circuit contains all of the essential circuitry for the conversion. It contains an eight channel analog input in which an operator or computer can select or address any one of the eight input channels. This allows for the simultaneous treatment of several instruments. After the signal passes the input multiplexing circuit, the chosen input analog signal goes to the input of a voltage comparator circuit. Here the voltage of the input signal is compared with a voltage signal which has been generated internally by the integrated circuit. This internal reference signal is generated by a 256 resistor ladder circuit which serves as a digital to analog (D/A) converter. A counter known as the successive approximation register (SAR) generates a sequence of digital numbers which are converted to an analog signal by the resistor ladder. The number on the SAR which produces an internal analog voltage which is the same as the voltage of the input signal is the number of interest. This number is the digital correspondence for the input signal at that moment. As the voltage of the input signal increases, the value of this number increases. The number in the SAR is automatically latched into an output buffer by internal control and timing circuits. With this, the number is made available to an external digital device.

The ADC0808 is the heart of the A/D converter; however, a few external circuits are required for its support. First, the ADC0808 requires a clock which produces a series of pulses at a frequency of 500 kHz. The clock in this circuit is composed of a ring of three inverters with a 0.01 uf ceramic capacitor that extends across one of the inverters:



The inverters which are used are those contained in the 7404C CMOS hexinverter. Since only one input channel is desired, the input port In(0) is addressed by grounding the three input addressing pins A, B and C. Also since a continuous conversion is desired, the end-of-conversion (EOC) pin is connected to the start conversion pin and the address latch enable (ALE) pin. A normally closed switch is connected between the start pin and the EOC pin of the chip. This switch is used to manually start the conversion during power-up. To enable the output, the output enable or Tristate pin is connected to the V(cc) (+5 volts). For the reference voltage, the V(ref+) pin is connected to the V(cc) while the V(ref-) pin is merely grounded. It should be noted that the difference between the V(ref+) pin and the V(ref-) may range from 0.512 to 5.25 volts with a normal voltage of 5.000 volts. The actual external circuitry of this A/D converter is extremely simple, since the ADC0808 contains most of the circuitry.

This test circuit was constructed on a Proto-Board #100 with #22 solid-copper wire used for interconnections. To test the circuit, a potentiometer was used to generate the input analog test signal. For the output, a 16-pin cable was used to connect the binary output of the ADC0808 to the parallel input port of the Super Elf. The small program in Figure 2 was used to read the value from the input port and display it on the hex readout. From mere eye inspection, this circuit appears to convert signals very quickly. Also the circuit appears to be very stable. For the most part, the device functions very well.

As stated earlier, an A/D converter is a very useful device in the laboratory for the automation of data gathering and processing, but it can also be quite useful in the home and workshop. The output port of the ADC0808 can be directly connected to the parallel input port of the Super Elf. If several of the analog input ports of the ADC0808 are to be used, the 3-bit addressing port can be connected to the parallel output port of the Super Elf, so that the Elf can select the signal which it choses to analyze. The main difficulty with this circuit which has not been adequately resolved is the analog amplifier or buffer which is needed to adapt the output of some analog device with the input of the ADC0808. As the circuit is now, it can only be connected to a device which produces a signal with a voltage range of 0.0 to 5.0 volts. As it is now, a CdS photocell or thermistor could be employed in a voltage dropping resistor bridge in order to monitor light or temperature. Once this problem has been solved, it is hoped to use this A/D converter as a means to interface the Super Elf to a Gas Chromatograph. In short, a Gas Chromatograph (GC) is a useful instrument in analytical chemistry which separates and detects volatile components in a chemical mixture. By using the Super Elf and Tiny BASIC, the qualitative and quantitative identification of each chemical component in some unknown mixture will be enhanced and perhaps partially automated.

```
*ADC Tester
*
0000    F8 0F    Initialize Stack Pntr
        B2
        F8 FF
        A2
        E2       Set R2 as Stack Pntr
0007    6D       Read Parallel Input Port
        64       Display on Hex Readout
        22       Bump Stack Pntr back
        30 07    Try it again!
```

CHIP-8 PROGRAMMING

by
Richard Johnson

CHIP-8 is a simple but powerful language with video graphics capabilities developed by RCA for the VIP computer; a version of CHIP-8 resident on ROM appears to be the heart of RCA's Studio II video game. RCA recently released a new version called CHIP-8X which includes color commands for the VIP color board. Source listings of over thirty game programs written in CHIP-8 are now available in RCA publications. Programs written in CHIP-8 are unusually compact, because each instruction of this language is only two bytes long. This article discusses software and hardware needed to run CHIP-8 programs on the Super Elf; most of the article should also be valuable to owners of Elf and Elf-II computers. All it takes is a CHIP-8 interpreter written for the Elves and a little soldering to make a wealth of software available to your computer.

A machine code listing of a CHIP-8 interpreter for the VIP computer is included in the "RCA COSMAC VIP Instruction Manual" (RCA Publication VIP-311). I bought a copy of this manual (which also includes twenty video game programs) two years ago hoping to be able to use the interpreter on my Super Elf. This interpreter is only 512-bytes long, but extensively uses routines in the VIP's 512-byte operating system. A listing of the VIP's operating system is also given in the manual. I disassembled the operating system and most of the CHIP-8 interpreter by hand, and wrote my own monitor for the Super Elf using the RCA tape cassette I/O routines. Although I learned a lot about the interpreter, I soon realized that hardware differences would greatly complicate implementation of CHIP-8 on one of the Elves. I had given up on CHIP-8 for several months, but then I learned that sombody else had already done the job!

Paul C. Moews published the booklet "Programs for the COSMAC ELF: Interpreters" in March, 1979 (see Questdata, Vol. 1, #8, p. 13). He gives a CHIP-8 interpreter (including source listing) which can be run on 1-1/4 K or 4 K Elves. He also includes a relocatable CHIP-8 interpreter for 4 K Elves (but only a machine language listing). Moews' CHIP-8 interpreter is even more powerful than the original RCA version: His interpreter has additional features such as multiplication, division, a larger variety of skip instructions, the capability of displaying all the ASCII characters, and the capability of displaying a variable on the LED display.

Moews' interpreter has two main drawbacks, however, which he discusses in his booklet's section "Hardware Differences between 1802 Computers." One drawback is due to keyboard differences; the other drawback is due to audio output differences. I present below simple hardware modifications of the Super Elf and slight software changes in Moews' CHIP-8 interpreter which result in CHIP-8 programs written for the VIP computer being much more compatible with Moews' CHIP-8 interpreter.

RCA's VIP computer and Studio II video game use scanned keyboards which allow the rapid response of programs to a key depression. This feature is essential for many video games; paddles can be moved, for example, using different keys to indicate different directions. The Super Elf keyboard, however, is latched and uses a separate input key to signal a program that an input data byte is available. Moews' interpreter, therefore, requires two key depressions to input a single hexadecimal digit.

Kirk D. Bailey pointed out that adding a single jumper to the Super Elf allows the input of hexadecimal digits without the use of the input key (see Questdata, Vol. 1, #6, pp. 6,11). Bailey pointed out that the keyboard decoder chip U25 provides a data available strobe at pin number 13. (Note: Bailey's article said pin number 12, but that was a typographical error.) In the Super Elf design, this data available pin is connected only to pin number 9 of the keyboard latch chip U15.

Bailey suggested connecting this strobe to I/O flag -EF4 of the 1802 microprocessor. I chose to connect the strobe to I/O flag -EF2, however; -EF2 is normally reserved to be used for the Super Elf's parallel input port, RS232 input, or 20 mA current loop input, but I need none of these inputs when running CHIP-8 programs. Instead of connecting the jumper directly to pin 23 of the 1802, I found it more convenient to use pin C of the 44-pin connector. Be sure to take precautions against static discharge if you decide to add such a jumper to your system. This hardware modification voids the Super Elf warranty, but I had already made other modifications to my system which had voided the warranty long ago!

The hardware modification outlined above allows the input key to still be used for other programs not written in CHIP-8. Let us review the software normally used for inputting a byte:

LOC.	CODE	MNEM.	COMMENTS
0000	3F00	BN4	Loop until input key depressed
0002	3702	B4	Loop until input key released
0004	6C	INP4	Input latched byte

The last two digits entered by the keyboard would now be in the D register and in the stack. Note that the Super Elf has an inverter between the input key and the -EF4 pin of the 1802.

Use of the input key can be omitted after the hardware modification by using the following software:

LOC.	CODE	MNEM.	COMMENTS
0000	3500	B2	Loop until any digit key depressed
0002	3D02	BN2	Loop until that key released
0004	6C	INP4	Input latched byte

If you are using this software to enter a single digit, the most significant digit in the D register is "garbage" (the next-to-last key depressed), but this problem can easily be eliminated by ANDing with 0F. Moews' interpreter already contains the code to eliminate the garbage digit.

I was able to modify Moews' relocatable CHIP-8 interpreter to use the data available strobe connected to -EF2 by changing only four bytes as follows:

LOC.	NEW CODE	MNEM.
010A	35	B2
010C	3D	BN2
01A2	35	B2
01A7	35	B2

Moews' CHIP-8 interpreter for which he includes the source listing (and which can be run on 1-1/4 K systems) is slightly different, but should also require changing only four bytes as follows:

LOC.	NEW CODE	MNEM.
010A	35	B2
010C	3D	BN2
01A8	35	B2
01AD	35	B2

These simple hardware and software modifications make video games which require rapid keyboard response much more enjoyable.

The second drawback of using one of the Elves for CHIP-8 programs is the audio output. The RCA VIP computer uses the Q output of the 1802 to turn on an oscillator circuit which drives a small speaker. The speaker output of the Super Elf, however, is simply the Q output buffered by a transistor switch; tones are generated by flip-flopping the Q output. The proper sound effects for CHIP-8 programs can be obtained by using the Q output line to control an oscillator.

The Super Elf has a 100-ohm resistor (R4) in series with the speaker output; the manual states that optionally this resistor can be shorted by a jumper to use the speaker output to drive a relay. I replaced the resistor with a jumper, not to drive a relay, but so the circuit given in Fig. 1 could be used. I utilized parts available in my junk box. I bought the code practice oscillator (CPO) several years ago for a dollar or two; the CPO was designed to be controlled by a 1.5 V cell and a telegraph key connected in series with the +V input. The CPO module I have draws 30 mA for 5 V operation; it gave a very high-pitched tone when operated at 5 V, however, so I lowered the CPO's frequency by replacing the resistor determining the RC time constant with a higher-valued resistor. (An oscillator circuit using a 555 IC would have worked equally well, of course.) I gutted a long-dead transistor radio of everything but its speaker; the small radio case can hold all of the parts shown in Fig. 1. WARNING: Do not try to drive an oscillator circuit with an unbuffered Q line; the 1802 cannot provide much current and you would probably destroy your microprocessor chip!

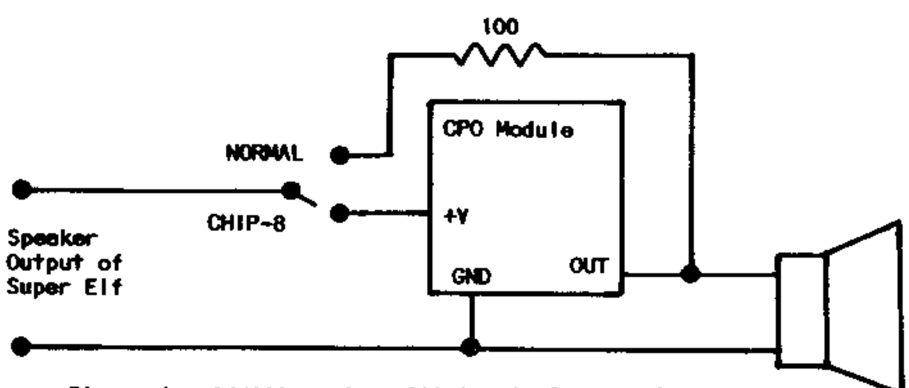


Figure 1. Addition of oscillator to Super Elf.

Most of the CHIP-8 language is easy to learn. Nevertheless, the display instructions are tricky, and the RCA documentation for those instructions should be studied with care. I have found it useful to prepare a one-page summary of all of the instructions of Moews' relocatable interpreter; that summary is given in Table I.

One last comment. With the hardware and software modifications I have discussed above, you should be able to run all of the programs written in CHIP-8 for the VIP. A program may use keys 2, 8, 4, and 6 to indicate moving a cursor or paddle up, down, left, or right. The Super Elf has keys in a different arrangement than does the VIP. Therefore, you would probably want to alter such a program so that keys 1, 9, 4, and 6 are used to indicate moves of up, down, left, and right. In a typical program, this would require changing only two bytes of code.

I hope my article will encourage owners of Elf, Super Elf, and Elf-II systems to use the CHIP-8 language. Readers having systems with color video boards or sound boards, for example, could develop CHIP-8 or machine language subroutines adding color or music control to CHIP-8 programs. I look forward to readers of Questdata submitting original programs written in the CHIP-8 language.

Bibliography

"RCA COSMAC VIP Instruction Manual," RCA Publication VIP-311.

"RCA COSMAC VIP User Manual," RCA Publication VIP-320.

"Game Manual," RCA Publication VIP-710.

Paul C. Moews, "Programs for the COSMAC ELF: Interperters."

	Instruction	Operation	Comments
CONTROL:	1MMM	GOTO 0MMM	
	BMMM	GOTO 0MMM + V0	
	FFFF NMMM	GOTO NMMM	
	2MMM	GOSUB 0MMM	
	00EE	RETURN	
	0MMM	USR 0MMM	Return from sub-routine with D4
SKIPS:	3XKK	SKIP IF VX = KK	Next CHIP-8 instruction is
	4XKK	SKIP IF VX ≠ KK	skipped if cond-ition met
	5XY0	SKIP IF VX = VY	
	5XY1	SKIP IF VX > VY	VX represents one of the
	5XY2	SKIP IF VX < VY	16 variables V0 - VF
	5XY3	SKIP IF VX ≠ VY	
	9XY0	SKIP IF VX ≠ VY	
MATH:	6XKK	VX = KK	
	8XY0	VX = VY	
	FX07	VX = TIME	TIME is a special variable
	FX15	TIME = VX	decremented by interrupt routine
	7XKK	VX = VX + KK	
	8XY4	VX = VX + VY	VF = carry
	8XY5	VX = VX - VY	VF = carry
	9XY1	VF,VX = VX x VY	VF = most significant byte
	9XY2	VX = VX/VY	VF = remainder
LOGIC:	8XY1	VX = VX OR VY	
	8XY2	VX = VX AND VY	
	CXKK	VX = RND AND KK	RND is a random number
MEMORY:	AMMM	I = 0MMM	I represents the memory pointer
	FX1E	I = I + VX	
	FX29	I = PATTERN(VX.0)	VX.0 = least sig-nificant digit
	FX33	VX -> M(I),M(I+1), M(I+2)	2 digit hex -> decimal conversion
	9XY3	VX,VY -> M(I), ...,M(I+4)	4 digit hex -> decimal conversion
	FX55	M(I+J) = VJ for J=0,X	I=I+X+1
	FX65	VJ = M(I+J) for J=0,X	I=I+X+1
	FX94	I = PATTERN(VX)	VX is code for an ASCII symbol
INPUT:	FX0A	VX = hex digit key	
	EX9E	SKIP IF VX = hex digit key	
	EXA1	SKIP IF VX ≠ hex digit key	
OUTPUT:	00E0	CLEAR SCREEN	
	FX75	LED Display = VX	Use only for X ≠ F
	DXYN	DISPLAY N@VX,VY	VF = 01 if pixel already was on
	FX18	TONE = VX	Q on for VX/60 seconds

A SCROLLED DISPLAY

by
Gary H. Price

The accompanying programs provide a fine-increment scrolling capability for 64 X 32 bit (4 video scans per display line) graphics generated using the ELF 1861 video driver. The longer program, Listing 1, uses portions of 2 pages in addition to whatever memory is allocated to the display area. The display may occupy as many contiguous pages of memory as are available (less one, used at the end of the display area to repeat the first display page in order to obtain a continuous-loop scroll). The shorter program, Listing 2, is tailored for the basic 1-page ELF and displays the entire page, one-half of which is free for the display itself.

The heart of both programs lies in the interrupt-service routine by which the 1861 display output is formatted. This routine is an elaboration of the standard 4-scan-lines-per-display-line routine. The display is advanced one scan line every N+1 display frames, where N is an input parameter. In order to scroll the display, the entry point into the line-repeat loop is changed by modification of the address of a branch instruction just preceding the loop. Such self-modification of its instructions by a program generally is not considered to be good programming practice, but the tight timing associated with the video display output severely limits the alternatives, and a better approach was not found.

The logic for controlling the roll rate and for updating or resetting when necessary the other counters and pointers involved in the display formatting is executed following the video output, prior to return of control from the interrupt routine to the main program. The contents of the control registers can also be modified external to the interrupt routine, for example to inhibit the roll action while data is being entered. This capability is used in the parameter-entry subroutine KEYS as well as, in the longer program, within a separate sequence in the main program for entry of data into the display itself.

The two versions of the program are described further in the next sections.

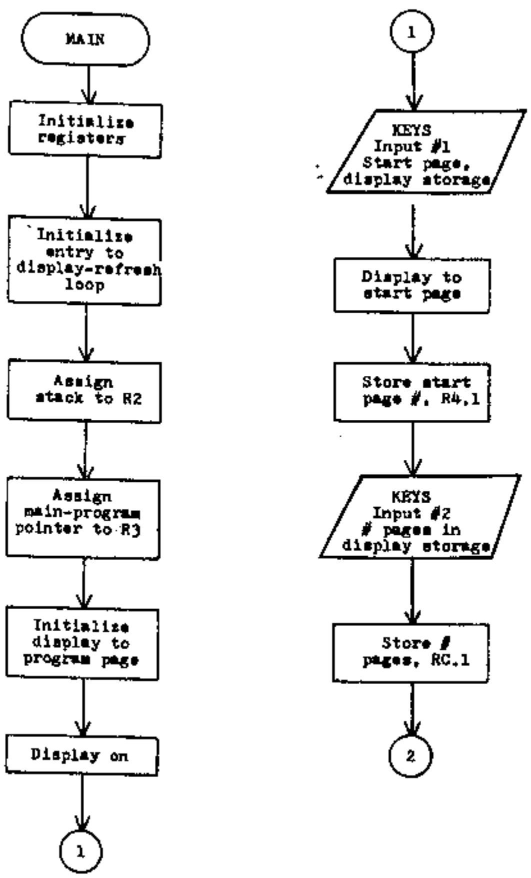
Multipage program and display, Listing 1. The organization of the longer program is outlined in the condensed flow chart presented as Figure 1. The program can be loaded to any two consecutive pages of memory and run without modification. It is recommended that program execution not be started by a jump from a monitor; roughly one-half of such starts will show improperly framed displays as a result of the 1861 being enabled in mid-frame. ————> Clean starts can reliably be obtained by insertion of a direct branch at location 0000 to START should the program be loaded above page 0.

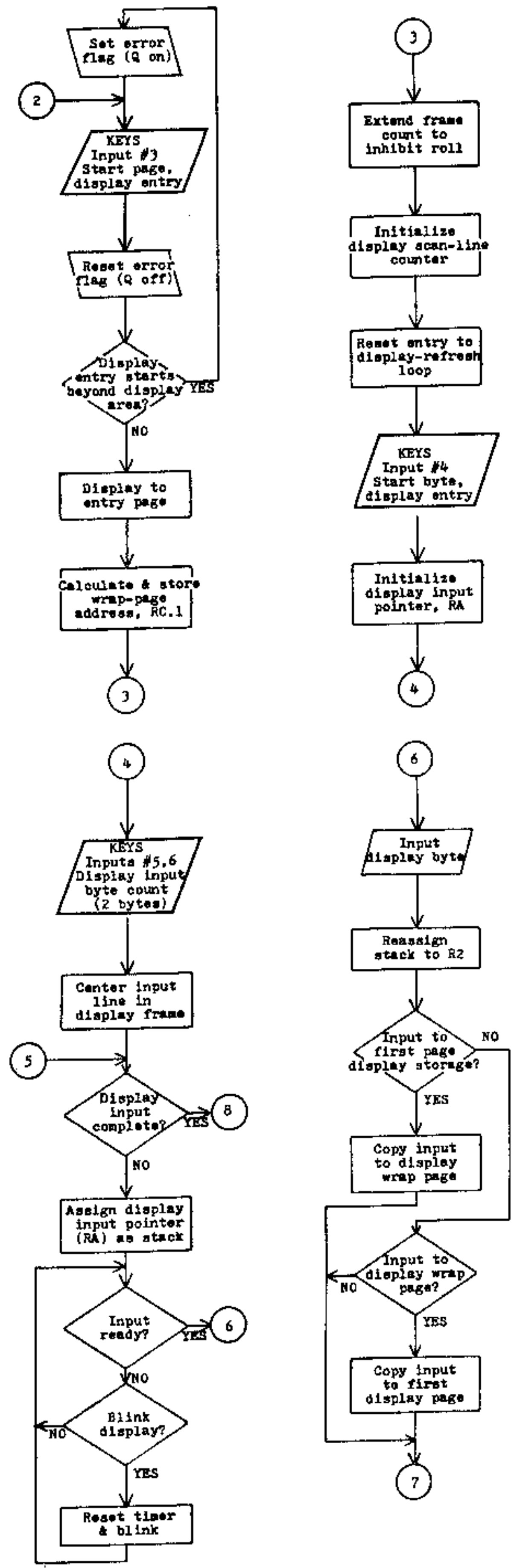
Features of this program, in addition to the scrolling display interrupt subroutine (located on the second page of program memory), include input prompts, listed in Table 1, to help keep track of the data-entry sequence and an error flag (Q set) if entry of display data is specified to begin outside the display storage area. The limits of this area are also defined through program input. Note that one page more than specified will automatically be allocated for the display in order to accomodate the first-page repeat. Data entry into the display is further aided by a cursor — a blinking of the current contents of the next display location to be loaded. The display roll can also be halted by pressing the input key at any time following initial data entry in order to load additional data into the display. It should be noted that no attempt has been made to prevent the display from including the program area; if this is done, it is not difficult to inadvertently overwrite the program and thereby to crash it. If the display storage area is properly set up, however, the program generally will protect itself adequately.

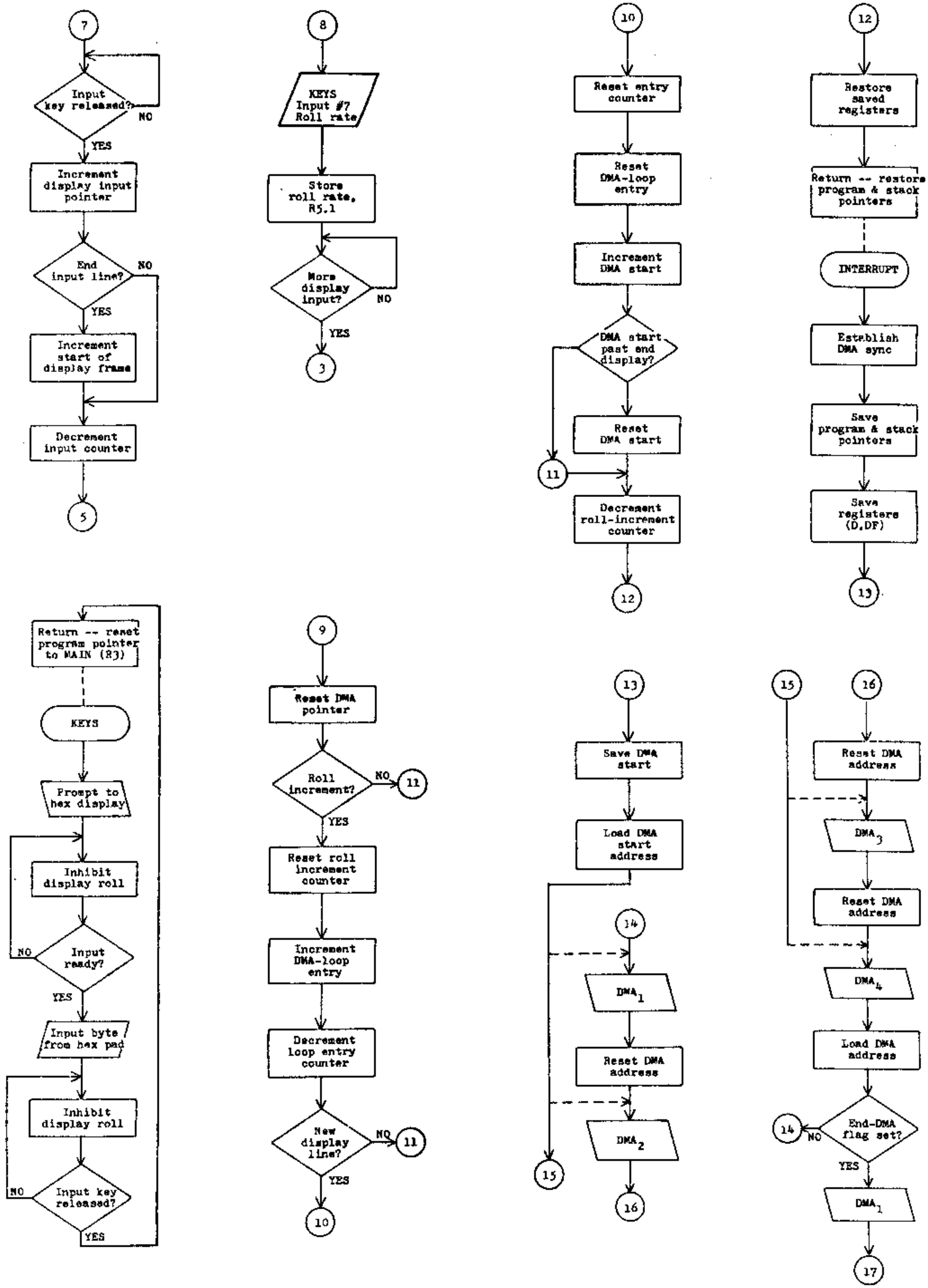
Single-page program and display. The single-page program, Listing 2, makes use of a hardware characteristic of the basic ELF to obtain a continuous-loop scroll. Namely, only the low byte of register-pointer memory addresses matters; the same memory location is addressed for a given low-byte value no matter what the value of the high byte. Consequently when the low byte of the display DMA-out pointer, R0, is advanced from FF to 00, the memory accessed automatically resets to the beginning of the page.

Should a ROM monitor be present somewhere in memory, however, some attention must be paid to page addresses. The high-byte reset of the DMA pointer, R0, that is included in the interrupt routine is present for this reason. Additionally, the high bytes of several other registers must be initialized. The changes to the program necessary to perform this initialization are described in Listing 3. The high-byte initialization section is placed at the end of the program area, where it can be overwritten without harm (after having been executed) by display input. If this is done, the contents of locations 00-02 should be replaced by the original coding (Listing 2) before the program is restarted, after having been halted. Note, however, that any use of the monitor will disturb the registers, and initialization must be repeated in this case (which means reloading the code if it has been overwritten by display input and restoring the branch at location 00) before the program will execute properly.

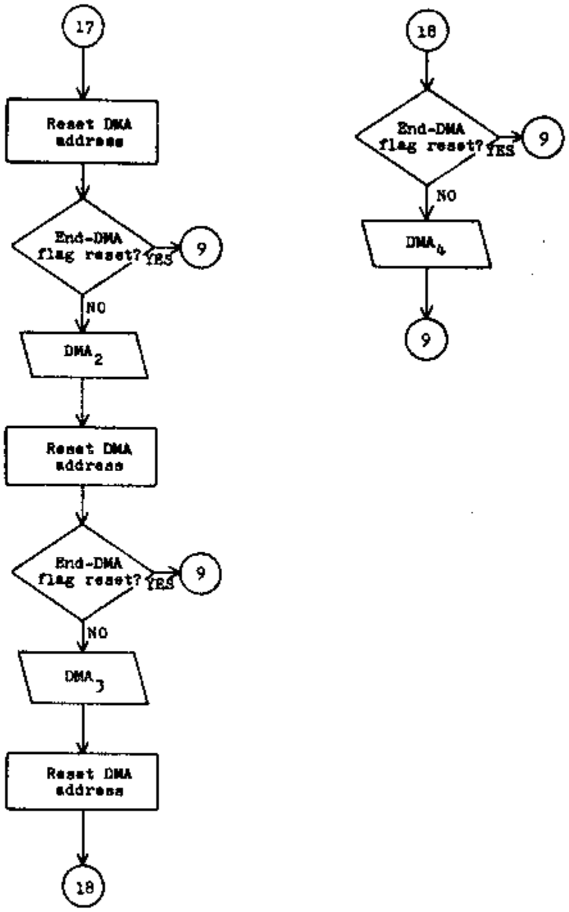
In order to maximize the memory area available for the display, minimal provision for entry of data into the display has been retained in the basic-ELF version of the program. Input consists of, first, the starting address for entry of display data, which should lie above the end of the program (i.e., above location 7F), followed by the number of bytes of display data that will be entered. Following completion of entry of the specified number of display bytes, the roll rate is entered. Note that entry of display data past the end of the page will cause the program to be overwritten. This causes no immediate harm up to a point (left for the reader to determine), but the overwritten code will have to be restored before the program can be restarted, once stopped.







P.O. Box 4430, Santa Clara, CA 95054



QUESTDATA
P.O. Box 4430
Santa Clara, CA 95054

Publisher ....Quest Electronics
Editor ....Paul Messinger
Assistant to Editor ...Jeanette Johnson
Associate Editor ....Allan Armstrong
Contributing Editors Ron Cenker
Van Baker
Art and Graphics ....Holly Olson
Proof Reading ....Judy Pitkin
Production ....John Larimer
Circulation ....Sue Orr

Listing 1. Multipage scroll program.

ADDR	CODE	LABEL	OPCODE	OPERAND
0000	90	START	GHI	0
0001	B2		PHI	2
0002	B3		PHI	3
0003	B8		PHI	8
0004	FC		ADI	
0005	01		01	
0006	B1		PHI	1
0007	B7		PHI	7
0008	F8		LDI	
0009	2A		INTlo	
000A	A1		PLO	1
000B	F8		LDI	
000C	CB		STKlo	
000D	A2		PLO	2
000E	A5		PLO	5
000F	F8		LDI	
0010	1D		MAINlo	
0011	A3		PLO	3
0012	F8		LDI	
0013	37		RFBRlo	
0014	A7		PLO	7
0015	F8		LDI	
0016	B6		KEYSlo	
0017	A8		PLO	8
0018	F8		LDI	
0019	38		RFIN1lo	
001A	57		STR	7
001B	E2		SEX	2
001C	D3		SEP	3
001D	F8	MAIN	LDI	
001E	00		00	
001F	A0		PLO	0
0020	69		IN	1
0021	D8		SEP	8
0022	01		01	
0023	B0		PHI	0
0024	B4		PHI	4
0025	D8		SEP	8
0026	02		02	
0027	BC		PHI	C
0028	38		SKP	
0029	7B	ERR	SEQ	
002A	D8		SEP	8
002B	03		03	
002C	7A		REQ	
002D	9C		GHI	C
002E	F5		SD	
002F	33		BPZ	
0030	29		ERR	
0031	94		GHI	4
0032	F4		ADD	
0033	B0		PHI	0
0034	9C		GHI	C
0035	52		STR	2
0036	94		GHI	4
0037	F4		ADD	
0038	BC		PHI	C
0039	F8	MODSP	LDI	
003A	04		04	
003B	A5		PLO	5
003C	B6		PHI	6
003D	A6		PLO	6
003E	F8		LDI	
003F	38		RFIN1lo	
0040	57		STR	7
0041	D8		SEP	8
0042	04		04	
0043	80		GLO	0
0044	F4		ADD	
0045	AA		PLO	A
0046	90		GHI	0
0047	7C		ADCI	
0048	00		00	

ADDR	CODE	LABEL	OPCODE	OPERAND
0049	BA		PHI	A
004A	D8		SEP	8
004B	05		05	
004C	BB		PHI	B
004D	D8		SEP	8
004E	06		06	
004F	AB		PLO	B
0050	8A		GLO	A
0051	FA		ANI	
0052	F8		F8	
0053	FF		SMI	
0054	80		80	
0055	A0		PLO	0
0056	9A		GHI	A
0057	7F		SMBI	
0058	00		00	
0059	52		STR	2
005A	94		GHI	4
005B	F5		SD	
005C	33		BPZ	
005D	63		CONTO	
005E	9C		GHI	C
005F	BA		PHI	A
0060	FF		SMI	
0061	01		01	
0062	38		SKP	
0063	02	CONTO	LDN	2
0064	B0		PHI	0
0065	8B	LDSPLY	GLO	B
0066	3A		BNZ	
0067	6B		LOAD	
0068	9B		GHI	B
0069	32		BZ	
006A	AC		DONE	
006B	EA	LOAD	SEX	A
006C	37	TESTL	B4	
006D	7A		BYTIN	
006E	85		GLO	5
006F	FF		SMI	
0070	F0		F0	
0071	33		BPZ	
0072	6C		TESTL	
0073	F8		LDI	
0074	FF		FF	
0075	A5		PLO	5
0076	F3		XOR	
0077	5A		STR	A
0078	30		BR	
0079	6C		TESTL	
007A	6C	BYTIN	IN	4
007B	E2		SEX	2
007C	9A		GHI	A
007D	52		STR	2
007E	94		GHI	4
007F	F7		SM	
0080	3B		BM	
0081	88		TESTH	
0082	8A		GLO	A
0083	AC		PLO	C
0084	0A		LDN	A
0085	5C		STR	C
0086	30		BR	
0087	90		WAITO	
0088	9C	TESTH	GHI	C
0089	F5		SD	
008A	3B		BM	
008B	90		WAITO	
008C	8A		GLO	A
008D	A4		PLO	4
008E	0A		LDN	A
008F	54		STR	4
0090	37	WAITO	B4	
0091	90		WAITO	
0092	1A		INC	A

ADDR	CODE	LABEL	OPCODE	OPERAND
0093	8A		GLO	A
0094	FA		ANI	
0095	07		07	
0096	3A		BNZ	
0097	A9		CONT2	
0098	80		GLO	0
0099	FC		ADI	
009A	08		08	
009B	A0		PLO	0
009C	90		GHI	0
009D	7C		ADCI	
009E	00		00	
009F	52		STR	2
00A0	9C		GHI	C
00A1	F5		SD	
00A2	3B		BM	
00A3	A7		CONT1	
00A4	94		GHI	4
00A5	BA		PHI	A
00A6	38		SKP	
00A7	02	CONT1	LDN	2
00A8	B0		PHI	0
00A9	2B	CONT2	DEC	B
00AA	30		BR	
00AB	65		LDSPLY	
00AC	D8	DONE	SEP	8
00AD	07		07	
00AE	B5		PHI	5
00AF	3F	IDLE	BN4	
00B0	AF		IDLE	
00B1	37	WAIT1	B4	
00B2	B1		WAIT1	
00B3	30		BR	
00B4	39		MODSP	
00B5	D3	RTNK	SEP	3
00B6	E3	KEYS	SEX	3
00B7	64		OUT	4
00B8	E2		SEX	2
00B9	F8		LDI	
00BA	FF		FF	
00BB	A5	LOOPK	PLO	5
00BC	3F		BN4	
00BD	BB		LOOPK	
00BE	6C		IN	4
00BF	F8		LDI	
00C0	FF		FF	
00C1	A5	WAITK	PLO	5
00C2	37		B4	
00C3	C1		WAITK	
00C4	F0		LDX	
00C5	30		BR	
00C6	B5		RTNK	
00C7	00		--	
00C8	00		--	
00C9	00		--	
00CA	00		--	
00CB	00	STK	--	
•				
•				
•				
0100	F0	RTN1	LDX	
0101	B0		PHI	0
0102	85		GLO	5
0103	3A		BNZ	
0104	24		FINI	
0105	95		GHI	5
0106	A5		PLO	5
0107	15		INC	5
0108	07		LDN	7
0109	FC		ADI	
010A	03		03	
010B	57		STR	7
010C	26		DEC	6
010D	86		GLO	6

```
ADDR CODE   LABEL   OPCODE OPERAND
010E 3A             BNZ
010F 24             FINI
0110 96             GHI    6
0111 A6             PLO    6
0112 F8             LDI
0113 38             RFIN1lo
0114 57             STR    7
0115 80             GLO    0
0116 FC             ADI
0117 08             08
0118 A0             PLO    0
0119 90             GHI    0
011A 7C             ADCI
011B 00             00
011C B0             PHI    0
011D 52             STR    2
011E 9C             GHI    C
011F F5             SD
0120 3B             BM
0121 24             FINI
0122 94             GHI    4
0123 B0             PHI    0
0124 25     FINI    DEC    5
0125 60             IRX
0126 72             LDXA
0127 FE             SHL
0128 72             LDXA
0129 70             RET
012A C4     INT     NOP
012B C4             NOP
012C C4             NOP
012D 22             DEC    2
012E 78             SAV
012F 22             DEC    2
0130 73             STXD
0131 76             RSHR
0132 73             STXD
0133 90             GHI    0
0134 52             STR    2
0135 80             GLO    0
0136 30             BR
0137 38     RFBR    RFIN1
 --   --            DMA1
0138 20     RFIN1   DEC    0
0139 A0             PLO    0
013A E2             SEX    2
 --   --            DMA2
013B 20     RFIN2   DEC    0
013C A0             PLO    0
013D E2             SEX    2
 --   --            DMA3
013E 20     RFIN3   DEC    0
013F A0             PLO    0
0140 E2             SEX    2
 --   --            DMA4
0141 80     RFIN4   GLO    0
0142 E2             SEX    2
0143 3C             BN1
0144 38             RFIN1
 --   --            DMA1
0145 20             DEC    0
0146 A0             PLO    0
0147 3C             BN1
0148 00             RTN1
 --   --            DMA2
0149 20             DEC    0
014A A0             PLO    0
014B 3C             BN1
014C 00             RTN1
 --   --            DMA3
014D 20             DEC    0
014E A0             PLO    0
014F 3C             BN1
0150 00             RTN1
```

```
ADDR CODE   LABEL   OPCODE OPERAND
 --   --            DMA4
0151 30             BR
0152 00             RTN1
```

```
REGISTER USAGE:
R0   : DMA output pointer
R1   : interrupt program pointer
R2   : stack pointer
R3   : main program pointer
R4.1: display area start page #
R5.0: display frame counter
R5.1: display frame-counter reset
R6.0: line-loop entry index
R6.1: line-loop entry-index reset
R7   : pointer to line-loop entry branch address
R8   : subroutine program pointer, KEYS
RA   : display load pointer
RB   : display-load byte counter
RC.1: display area wrap page #
```

Listing 2. Basic-Elf scroll program.

```
ADDR CODE   LABEL   OPCODE OPERAND
00   F8             LDI
01   61             RFBR
02   A7             PLO    7
03   F8     START   LDI
04   54             INT
05   A1             PLO    1
06   F8             LDI
07   7F             STK
08   A2             PLO    2
09   F8             LDI
0A   11             MAIN
0B   A3             PLO    3
0C   F8             LDI
0D   30             KEYS
0E   A8             PLO    8
0F   E2             SEX    2
10   D3             SEP    3
11   F8     MAIN    LDI
12   04             04
13   A5             PLO    5
14   B6             PHI    6
15   A6             PLO    6
16   F8             LDI
17   62             RFIN1
18   57             STR    7
19   F8             LDI
1A   00             00
1B   A0             PLO    0
1C   69             IN     1
1D   D8             SEP    8
1E   AA             PLO    A
1F   D8             SEP    8
20   AB             PLO    B
21   8B     LOAD    GLO    B
22   32             BZ
23   2A             ROLL
24   D8             SEP    8
25   5A             STR    A
26   1A             INC    A
27   2B             DEC    B
28   30             BR
29   21             LOAD
2A   D8     ROLL    SEP    8
2B   B5             PHI    5
```

ADDR	CODE	LABEL	OPCODE	OPERAND
2C	00	IDLE	IDL	
2D	30		BR	
2E	2C		IDLE	
2F	D3	RTNK	SEP	3
30	F8	KEYS	LDI	
31	FF		FF	
32	A5	LOOPK	PLO	5
33	3F		BN4	
34	32		LOOPK	
35	6C		IN	4
36	37	WAIT	B4	
37	36		WAIT	
38	30		BR	
39	2F		RTNK	
3A	85	RTNI	GLO	5
3B	3A		BNZ	
3C	51		FINI	
3D	95		GHI	5
3E	A5		PLO	5
3F	15		INC	5
40	07		LDN	7
41	FC		ADI	
42	03		03	
43	57		STR	7
44	26		DEC	6
45	86		GLO	6
46	3A		BNZ	
47	51		FINI	
48	96		GHI	6
49	A6		PLO	6
4A	F8		LDI	
4B	62		RFIN1	
4C	57		STR	7
4D	80		GLO	0
4E	FC		ADI	
4F	08		08	
50	A0		PLO	0
51	25	FINI	DEC	5
52	72		LDXA	
53	70		RET	
54	C4	INT	NOP	C4
55	C4		NOP	
56	C4		NOP	
57	22		DEC	2
58	78		SAV	
59	22		DEC	2
5A	52		STR	2
5B	9A		GHI	A
5C	B0		PHI	0
5D	E2		SEX	2
5E	E2		SEX	2
5F	80		GLO	0
60	30		BR	
61	62	RFBR	RFIN1	
--	--		DMA1	
62	20	RFIN1	DEC	0
63	A0		PLO	0
64	E2		SEX	2
--	--		DMA2	
65	20	RFIN2	DEC	0
66	A0		PLO	0
67	E2		SEX	2
--	--		DMA3	
68	20	RFIN3	DEC	0
69	A0		PLO	0
6A	E2		SEX	2
--	--		DMA4	
6B	80	RFIN4	GLO	0
6C	E2		SEX	2
6D	3C		BN1	
6E	62		RFIN1	
--	--		DMA1	
6F	20		DEC	0
70	A0		PLO	0
71	3C		BN1	
72	3A		RTNI	
--	--		DMA2	
73	20		DEC	0
74	A0		PLO	0
75	3C		BN1	
76	3A		RTNI	
--	--		DMA3	
77	20		DEC	0
78	A0		PLO	0
79	3C		BN1	
7A	3A		RTNI	
--	--		DMA4	
7B	30		BR	
7C	3A		RTNI	
7D	00		--	
7E	00		--	
7F	00	STK	--	

Listing 3. Modifications to basic-ELF program for use with ROM monitor.

ADDR	CODE	LABEL	OPCODE	OPERAND	COMMENT
00	30		BR		
01	80		INITH		
02	00		--		
•					
•					
•					
80	90	INITH	GHI	0	
81	B1		PHI	1	
82	B2		PHI	2	
83	B3		PHI	3	
84	B7		PHI	7	
85	B8		PHI	8	
86	BA		PHI	A	
87	F8		LDI		
88	61		RFBR		
89	A7		PLO	7	
8A	30		BR		
8B	03		START		

Table 1. Hex display prompts for multipage scroll program.

PROMPT	ENTRY
01	First page of display area.
02	Number of pages constituting display area.
03	Start page of date entry into display (relative to first page of display; i.e., 00=first page).
04	Location of first byte of data entry into display, relative to upper left corner (i.e., 00=upper left-most byte of display; FF = lower right-most byte).
05	High byte, number of bytes of display data to be entered.
06	Low byte, number of bytes of display data to be entered. Note -- hex display will continue to show 06 until entry of display data is complete.
07	Roll rate.

THE ADVANCED ORGAN

by
Nick Williams

Have you ever wanted to show off your ELF and everyone leaves or loses interest, while you sit there punching keys? If you have an unexpanded ELF with no type of mass storage or battery back-up this can be a problem.

My uncle always wants to see my SUPER ELF in action, but never really likes to wait. So one evening before a visit I sat down to figure out a pretty short but good program. Out came "The Advanced Organ" program.

The code is very simple to understand and to explain. However, if you don't understand it, here's the theory:

First of all R2 is set to location 00FE and made the official stack pointer via "E2". Next a byte is input from the Hex keypad and output at the Hex display (Providing your inputs at 6C and output at 64) R2 is decremented to point back at location 00FE.

Then the first timer is set up to determine the over all frequency, which is completely variable. Next a second timer is set up based upon the input Hex byte. The Q line is turned on, then the delay from the second timer, then Q is turned off.

After that the ELF checks to see if the input button is depressed. If not, back to 0000. If so, the input byte is loaded into the first timer to change the overall tone. It then returns the location 0000.

This program does not need any patches for an expanded system. It may be run in any location by changing both the address locations and the underlined bytes to suit that location.

First load the program, then depress RESET, and load a "22" into the second timer by pressing the 2, 2, and I keys. Then play the HEX key pad.

Registers Used:

R2 = Stack Pointer
RC = Input Data
RF = Input Pointer

```
ADDR CODE      LABEL  OPCODE OPERAND COMMENT
0000 F8 00 B2                        Initialize stack
                                     pointer high byte
0003 F8 F0 A2                        Initialize stack
                                     pointer low byte
0006 E2                              X = R2
0007 6C 64 22                        Input and output
                                     byte; R2 = R2-1
000A 32 00                           If input byte = 00
                                     branch to 00
000C AC                              Save input data in
                                     RC.0
000D F8 01                           Start counter 1.
                                     Byte 0E is
000F FF 01                           variable. D=D-1
0011 3A 0F                           loop till D=00
0013 7B                              Turn on Q (speaker
                                     line)
0014 8C                              Get input data
0015 FF 01                           data = data-1
0017 3A 15                           loop until data
                                     (in D) = 00
0019 7A                              Turn Q off
001A 3F 00                           branch to 00
                                     unless input is on
001C 37 1C                           Wait here until
                                     input is off
001E F8 00 BF                        Initialize
                                     Register F
0021 F8 0E AF                        high and low byte.
0024 8C 5F                           Load input data
                                     into Location 0E
0026 30 00                           Branch to 00

0000 F800 B2F8 F0A2 E26C 6422 3200 ACF8 01FF
0010 013A 0F7B 8CFF 013A 157A 3F00 371C F800
0020 BFF8 0EAF 8C5F 3000
```

TINY BASIC STRINGS

by
David Taylor

This program will permit the use of actual words in the response to yes or no questions from a program being run in TINY BASIC. It provides a short string function by peeking at the input buffer and reading the decimal value of the ASCII characters stored there. It can be located at any line number, I only used line 20000 to keep it out of the way of other programs.

You may have to change the value of Z (LINE NUMBER 20030) to a different value if your input buffer is at a location other than 48 in decimal.

I've found that using this program adds a little style to my programs rather than using the old standby of Y/N or YES = 0 NO = 1 type of inputs that were required with TINY in the past.

FOR TINY BASIC:

```
20000 REM YES OR NO SUBROUTINE
20010 REM IF YES, N= 0; IF NO, N= 1, IF NEITHER, N= 3
20020 LET N = 3
20030 LET Z= 38960
20040 INPUT X
20050 IF USR(33812,Z) = 78 IF USR(33812,Z+1) = 79
      LET N = 1
20060 IF USR(33812,Z) = 89 IF USR(33812,Z+1) = 69
      IF USR(33812, Z+2) = 83 LET N = 0
20070 P=USR(33816,Z+1,13)
20080 RETURN
```

FOR NETRONICS:

```
20000 REM YES OR NO SUBROUTINE
20010 REM IF YES, N= 0 ; IF NO, N= 1, IF NEITHER, N= 3
20020 LET N = 3
20030 LET Z=48
20040 INPUT X
20050 IF PEEK (Z) = 78 IF PEEK (Z+1) = 79 LET N = 1
20060 IF PEEK (Z) = 89 IF PEEK (Z+1) = 69
      IF PEEK (Z+2) = 83 LET N = 0
20070 POKE Z+1, 13
20080 RETURN
```

SAMPLE PROGRAM

```
10 PRINT "DO YOU NEED INSTRUCTIONS";
20 GOSUB 20000
30 IF N = 1 GOTO XXXX (NOTE: XXXX = LINE NUMBER OF
   START OF INSTRUCTIONS)
40 IF N = 0 GOTO XXYY (NOTE:  XXYY = LINE NUMBER OF
   PROGRAM BEYOND INSTRUCTIONS)
50 GOTO 10  (NOTE: IF PROGRAM REACHES THIS POINT
   N = 3 BY DEFAULT DUE TO VALUE SET IN LINE # 20020)
```

COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC CLUB COSMAC

20 QUESTDATA
P.O. Box 4430
Santa Clara, CA 95054

ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. Postage Paid
QUEST
Electronics
Permit No. 549
Santa Clara, CA

[redacted]