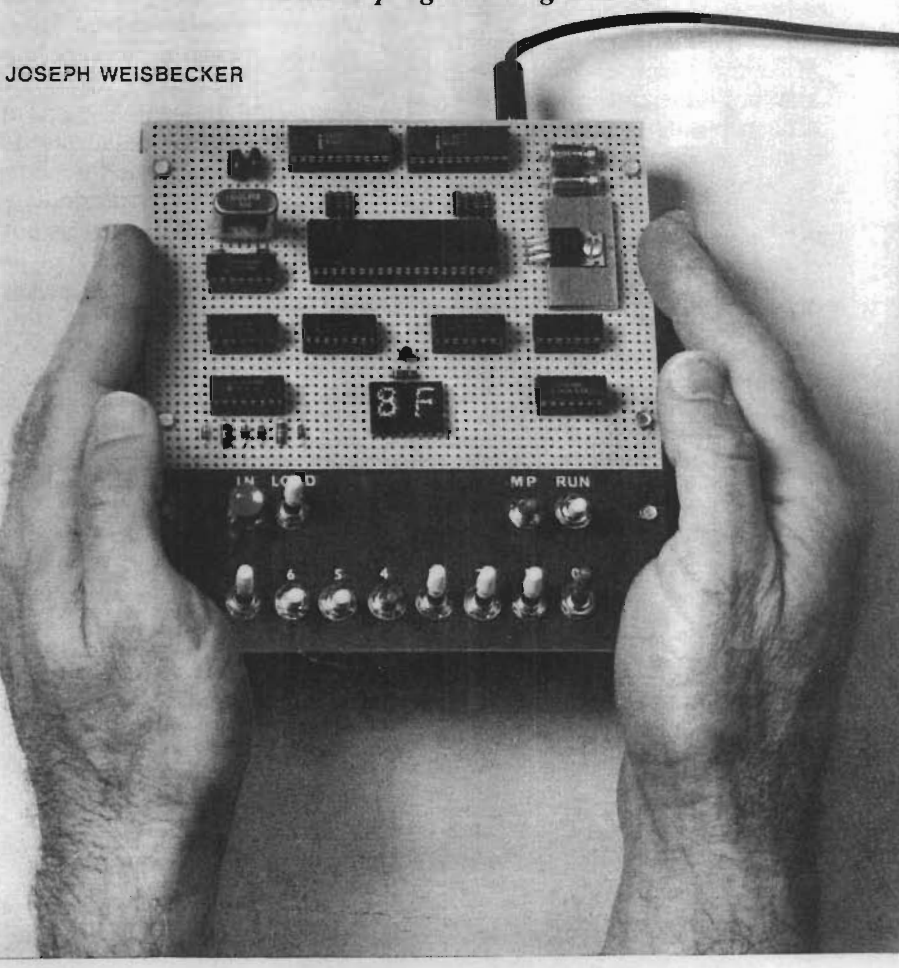
BUILD THE COSMAC "ELF"

A LOW-COST EXPERIMENTER'S MICROCOMPUTER

PART 2

Some hardware improvements and more programming details.

BY JOSEPH WEISBECKER



LAST MONTH, we discussed the construction of the low-cost Elf microcomputer/trainer and gave some examples of simple programming. This month, we will describe hardware and how to make a low-cost LED replacement for the relatively expensive hex display and add a simple 8-bit I/O port. Then we'll add a 16-switch monitor that, among other things, will allow you to use a hex keyboard. We'll finish up the hardware section by showing how to use a 9-volt battery as power for a RAM circuit to hold a program for as long as six months.

When we're finished with the hardware details, it's back to the software, continuing with our programming discussion.

The Hardware. The hex displays called for in the original Elf project can be replaced with a discrete LED circuit as shown in Fig. 1. You will need a CD4508 eight-bit register, eight low-current LED's, two 4049 hex inverters, and eight 470-ohm, ½-watt resistors. When the LED circuit is substituted for the hex displays, current consumption will be reduced by about 150 mA. The input comes from the data bus, which formerly went to hex displays *IC4* and *IC5*.

When you use the LED display, you must count the LED's to arrive at the hex number displayed. The upper four LED's form the first digit, the lower four the second digit.

You can mount the LED's on the front panel. Be sure you carefully identify each. Also, when making the conversion, don't forget to modify the RUN switch circuit as shown.

You can connect an inexpensive cadmium-sulfide (CdS) cell between the EF1 line and ground. Be sure to use a photocell that has a dark resistance in excess of 200,000 ohms and a light resistance of less than 10,000 ohms. If you use any other photocell, you may have to increase the value of the resistor to pull up the EF1 line of the 1802 microprocessor. The high input impedance of the CMOS logic eliminates the need for photocell amplification. Also, several photocell inputs can be used, each connected to a different flag (EF) line.

Using a photocell input, you can program the computer to start counting when an object moves past one photocell and stop counting when the object passes a second cell. This technique allows you to determine the speed of a moving object. It can also be used to count people, monitor motor speed, provide targets in a computer-controlled light gun or "eyes" for a computer-controlled robot, etc.

Magnetic reed switches, simple make/break switches, or similar devices can be connected to the computer via the flag-line inputs.

Several inexpensive methods of expanding the number of input and output lines can be used with this computer. One example is shown in Fig. 2. Here, a CD4058 IC is used in both the input and the output positions, while other IC's provide the necessary gating. A 69 instruction will store the values of the eight input lines in memory as a single byte.

In the output port section, a 61 instruction sets a memory byte into this port. The output port can control up to eight output lines, but you will have to add CD4050/CD4049 buffers if you wish to drive TTL loads. You can use

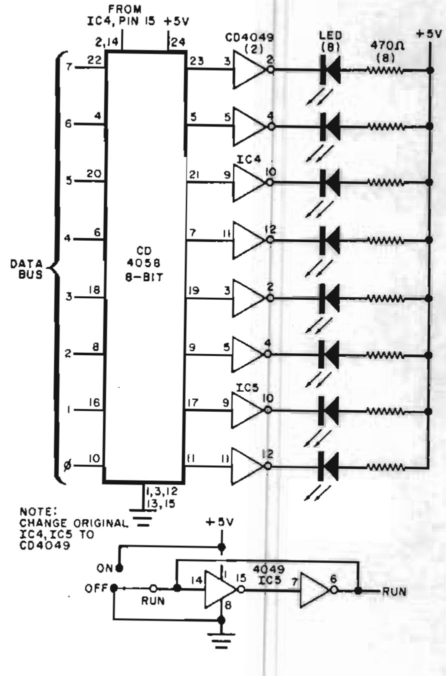


Fig. 1. Circuit for a discrete LED display.

these output lines to drive suitable transistors to control relays, lamps or LED's, or battery-powered motors, you can have the computer sequence lights, control animated displays or robots, or control a slide projector in response to tones from an audio tape. You can use the existing Q line output in the same manner for a single operation.

A simple method of controlling up to 16 output lines or monitoring the states of 16 switches is shown in Fig. 3. A 62 instruction will set the low-order digit of a memory byte into the 4-bit CD4515 register. The output line corresponding to this digit will go low, while the other 15 remain high. To make things more interesting, the computer can determine whether the switch attached to the selected output line is closed or not by testing EF2 with a branch instruction.

The following program continuously examines all 16 switches in sequence and stops with the number of any closed switch from 0 to F in the low-order digit of R3.0:

Step	M	Bytes	Comment
1	0000	F8 FF A2	FF→R2.0 (memory pointer)
2	0003	13 52 E2	R3 + 1, R3.0→M2, 2→X
3	0006	62 22	MX→CD4515 (select switch)
4	0008	3D 03	Repeat step 2 if switch is open
5	000A	30 0A	Stop with R3.0 = closed switch number

The diodes can be omitted if only one switch at a time will be closed. This circuit and an appropriate program could permit data and instruction bytes to be loaded into memory a digit at a time from a hex keyboard instead of toggle switches. Switch debouncing could be performed with a programmed delay following each key depression. A 64-character keyboard could be used by treating it as four groups of 16 keys each, with the common side of each key group connected to a different flag line. In fact, a program to generate the Morse code equivalent of each key could be written using the Q line as the output.

This circuit can also be used to select one of 16 external devices or I/O ports if desired. Using the latter technique would permit up to 128 I/O lines. Cascading CD4515's would permit even larger numbers of I/O lines to be handled.

A low-cost video terminal can be made using the "Scopewriter" (POPULAR ELECTRONICS, August 1974), or you can interface your computer with a cassette data interchange system.

We have only scratched the surface of I/O circuits for the Elf. The real fun (and program training) starts when you think of new things to attach to the output lines and start writing programs to activate them.

The major drawback with a RAM, or memory, system is that data stored in it is erased when the main power source is shut down. (Of course, if you could use a ROM, this wouldn't be a problem. However, ROM's must be preprogrammed with the memory data you wish to save, a costly and time-consuming approach.) Adding a cassette interface doesn't entirely eliminate the problem because a "bootstrap" is still required to be stored in memory to run the cassette.

The use of low-power COSMOS RAM IC's and a 9-volt mercury battery, as shown in Fig. 4, will allow you to save programs in memory for up to six months even with the main power to the computer turned off. The 1822 RAM's shown are pin-compatible with the 2101's specified for the original project, but some of the RAM's must be rewired as shown.

With the COSMOS RAM's installed, you can turn off power to the computer at any time. The mercury battery will supply the required standby power to the memory system so that the program will be ready to run immediately when the computer is again powered up. The newly added STANDBY switch should be turned on (+5 volts) only after power is turned on. It should be off to hold pin 17 of the RAM's at ground potential before removing power from the system.

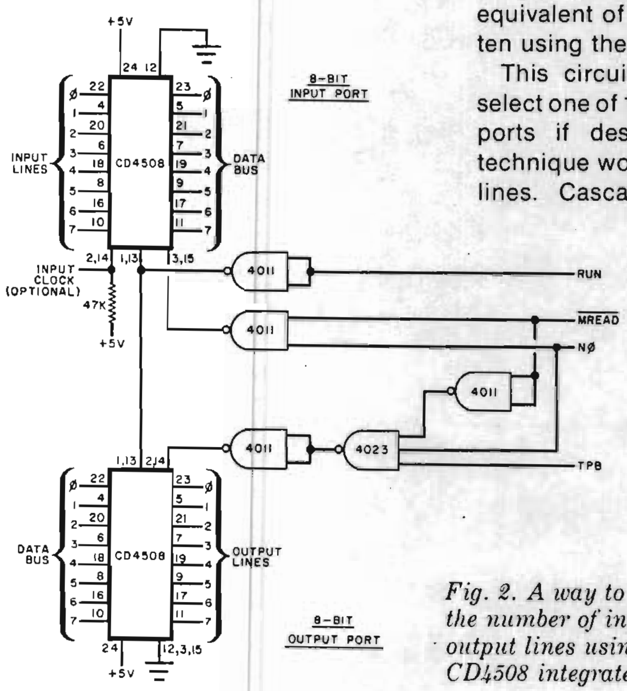


Fig. 2. A way to expand the number of input and output lines using two CD4508 integrated circuits.

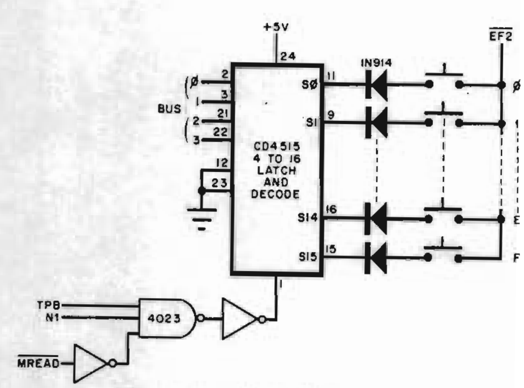


Fig. 3. A method of controlling up to 16 outputs.

Periodically check the battery's output; if it should fall too low, the memory system won't be able to hold data.

The last piece of hardware we will discuss here is the simple output driver shown schematically in Fig. 5. This is a conventional driver for almost anything that doesn't require more current than the transistor is capable of safely handling. The diode in the relay circuit removes the reverse transient spike that might otherwise damage the transistor. You can substitute a LED or even a load resistor for driving a power stage.

More Programming. The single-line output program shown below is a simple program that will flash the Q LED at a preset rate. It also provides a programmable square wave on the Q line.

Step	M	Bytes	Comment
1	0000	7A	0→Q
2	0001	F8 10 B1	10→R1.1
3	0004	21	R1-1
4	0005	91	R1.1→D
5	0006	3A 04	Repeat step 3 if D = 00
6	0008	31 00	Go to step 1 if Q = 1
7	000A	7B	1→Q
8	000B	30 01	Go to step 2

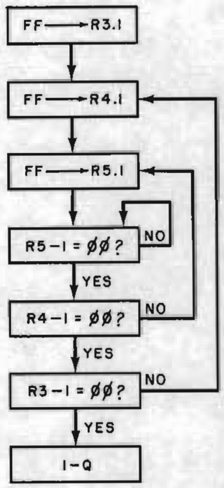
When you run this program, the square-wave frequency depends on the settings of the input switches. You can change frequency at any time. For higher frequencies, change B1 at M(0006) to A1 and 91 at M(0008) to 81. You can now select any of 256 different frequencies by altering the settings of the switches.

To modify the program to sweep the audio frequency range, use the following program:

Step	M	Bytes	Comment
1	0000	F8 FF A2	FF→R2.0
2	0003	7A	0→Q
3	0004	82 A1	R2.0→D; D→R1.0
4	0006	21 81	R1-1; R1.0→D
5	0008	3A 06	Repeat step 4 if D = 00
6	000A	31 03	Go to step 2 if Q = 1
7	000C	7B 22 82	1→Q; R2-1; R2.0→D
8	000F	32 00	Go to step 1 if D = 00
9	0011	30 04	Go to step 3

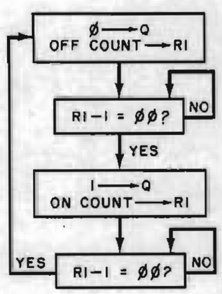
This program can be used in audio test applications. Note that R2 is used as a second counter that causes the square-wave frequency to change after each cycle. You can hear what this sounds like by using the circuit shown in Fig. 5.

Very low frequency square waves, or long-interval timing, can be programmed by cascading counters as illustrated in the following flow chart:



The Q line can then be used to activate a relay (as in Fig. 5), which can control house lights, motors, etc.

Suppose you wish to program a variable-pulse generator instead of square-wave generator. Use separate counts for the pulse off and on times as illustrated in the following flow chart:



This program will flash the Q LED and put a square wave on the Q line at a rate determined by the contents of memory M (0002) from a 10 to some other number. By referring back to the Instruction Subset Table in last month's article, you should be able to interpret the above program.

Note in the program that R1 is used as a 16-bit decrementing counter (steps 3, 4, and 5). When the high-order eight bits of this counter reaches 00, the Q line goes to its opposite stage. Changing steps 2 and 4 to use the low-order byte of R1 increases the Q line's output frequency by a factor of 256.

If you use a 1-MHz crystal in the clock, the above program can generate square waves at frequencies between 0.3 and 80 Hz, depending on the byte in M(0002). By changing the B1 instruction at M(0003) to A1 and the 91 instruction at M(0005) to 81, square waves between 80 and 20,000 Hz can be generated. In this manner, your basic computer becomes a pre-settable square-wave generator.

We can rewrite the program so that the square wave's frequency becomes a function of the settings of the toggle switches as follows:

Step	M	Bytes	Comments
1	0000	F8 FF A2	FF→R2.0
2	0003	E2	2→X
3	0004	7A	0→Q
4	0005	6C B1	Switch byte→ MX, D:D→R1.1
5	0007	21 91	R1-1; R1.1→D
6	0009	3A 07	Repeat step 5 if D = 00
7	000B	31 04	Go to step 3 if Q = 1
8	000D	7B 30 05	1→Q; Go to step 4

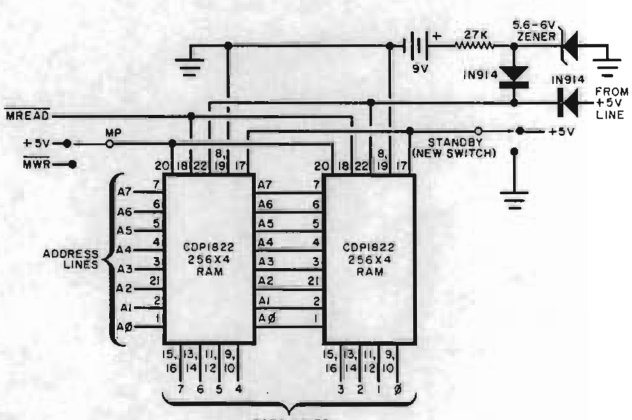


Fig. 4. Using a low-power COSMOS RAM and a 9-volt battery permits saving programs in memory.

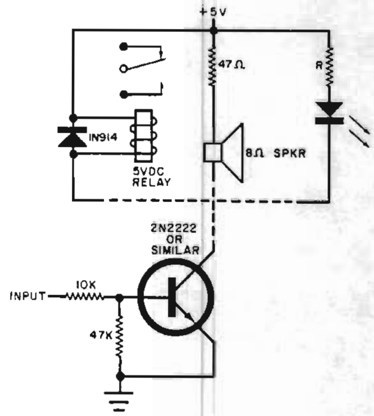


Fig. 5. Circuit to provide outputs used for testing.

In a similar manner, you can program bursts of pulses, variable-interval pulse trains, etc. You can even write a program where a list of bytes specifies a sequence of different tones to make a programmable music box.

The following two programs are "games" that demonstrate how the COSMAC instructions can be used. No added I/O circuits are required to run these programs.

Load the following sequence:

Step	M	Bytes	Comment
1	0000	E1	1→X
2	0001	F8 0F A1	0F→R1.0
3	0004	64	MX→display; X + 1
4	0005	3F 05	Wait for INPUT switch to be depressed
5	0007	6C	Switch byte → MX,D
6	0008	F8 0A F7	0A→D; D-MX→D
7	000B	51 64	D→M1; MX → display; X + 1
8	000D	30 0D 00	Stop; 00

Set both the LOAD and MP switches to off and then flip RUN to on. Have someone select any digit between 1 and 9 multiply by 10, add the original digit. Then multiply the sum by 9. Have the person who selected the digit tell you the result — but not the original digit. Set the binary code for the least-significant digit of the final answer into switches 3, 2, 1, and 0, and place the other input switches in the down position. When you depress the INPUT switch, the computer will display the unknown digit.

This program illustrates how to set a memory byte into the output display with a 6C instruction. Note the use of R1 as a memory pointer and the use of the binary subtract instruction in step 6.

The following program makes the computer "think" of a byte, which you must guess in no more than seven tries:

Step	M	Bytes	Comment
1	0000	8A AB	RA, 0→RB.0 = secret byte
2	0002	F8 AA A3	AA→R3.0 = memory pointer
3	0005	53 E3	D→M3; 3→X
4	0007	F8 07 A4	07→R4.0 = number of turns
5	000A	64 23	M3→display, 3 +1; 3 − 1
6	000C	2A 3F 0C	RA + 1 until INPUT is depressed
7	000F	37 0F	Wait for INPUT to be released
8	0011	6C 8B	Switch byte→M3; RB.0→D
9	0013	F5 33 1A	M3-D→D; Go to step 12 if M3 ≥ RB.0
10	0016	F8 01	01→D
11	0018	30 22	Go to step 16 (show D)
12	001A	3A 20	Go to step 15 if D = 00
13	001C	53 64	D→M3; M3→ display; 3 + 1
14	001E	30 1E	Stop loop
15	0020	F8 10	10→D
16	0022	53 64 23	D→M3→display; 3 + 1; 3 − 1
17	0025	24 84	R4-1, R4.0→D (turn counter)
18	0027	3A 0C	Go to step 6 if D = 00
19	0029	8B 7B	RB.0→D; 1→Q
20	002B	30 1C	Go to step 13 (show D and stop)

Place both the MP and LOAD switches in the off position after toggling the program. When you start the program by operating RUN; AA is displayed. Now, try to guess what byte the computer has selected by setting the eight INPUT switches and depressing the main INPUT switch. If 00 is displayed, you guessed correctly; if 01 is displayed, your guess is too low; if 10 is displayed, your guess is too high. You lose after seven wrong tries, at which point, the computer turns on its Q LED and the displays indicate the hidden byte. To try again, set RUN to off and then on.

The subtract instruction in step 9 sets an arithmetic overflow flag (DF) if MX is equal to or greater than D. The COSMAC instruction manual covers a detailed explanation of the use of this overflow flag in arithmetic and shift operations.

HEX NUMBER SYSTEM

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

In Closing. Now that you have some familiarity with programming for the Elf, look through your back issues of POPULAR ELECTRONICS for some challenging programs to write. Try the "Logidex" game in the November 1973 issue, "Tug-of-War" game in February 1975, "Electronic Dice" in July 1975, and the "Executive Digital Temper Countdowner" in December 1975. These are just a few of the many electronic games you can *program* instead of building. ◈

"Uh. . . About this loaner you sent us while our computer is being repaired. . ."