Thomas E. Hutchinson
35 Warrender Ave. Apt. 208
Islington, Ontario M9B5Z5
Canada

Modify Your COSMAC Elf

This modification to the Elf involves a lilliputian outlay of cash.

The COSMAC Elf microcomputer described in *Popular Electronics* (August and September 1976) has been one of the most interesting projects I've built. However, soon after wiring it together and running a few programs, it was obvious that I spent a lot of time loading programs by the eight front-panel toggle switches. And the frustrating part occurred when I would make an error at step 200, for example, and the only way to get back to that step was to press the In button 199 times and then make the correction. In subsequent issues of *Popular Electronics*, programs to interface the microcomputer with a keyboard and allow the machine to jump to a requested memory location to make a correction there were published. After looking over these programs, I decided they were not for me. First, you have to enter them using the eight front-panel toggle switches at the beginning of every program, which takes about 10 or 15 minutes every time you turn the computer on. Second, this program takes up valuable memory space; in fact, it uses 74 of the 256 available bytes in the memory. If you want to expand your memory, the latter may not concern you as much. Consequently, I devised the following circuit that interfaces a keyboard with the microcomputer and also allows you to jump to any desired memory location very quickly. The circuit requires no programming and uses no memory space.

Construction

The circuit was constructed on a 4 x 8 x 2 inch aluminum chassis. The chassis is turned upside down (open side up), and a piece of perfboard with a grid of 0.05 x 0.05 inch mounts over the open side using sheet metal screws; a second piece of perfboard with a 0.1 x 0.1 inch grid mounts inside the chassis. The hexadecimal keyboard was purchased from James Electronics, and the spacing of the pins conforms more or less to the 0.05 inch grid. A few extra holes had to be drilled to make it fit exactly into the perfboard. ICs 1,2,3,4 and 10 mount on the top perfboard with the

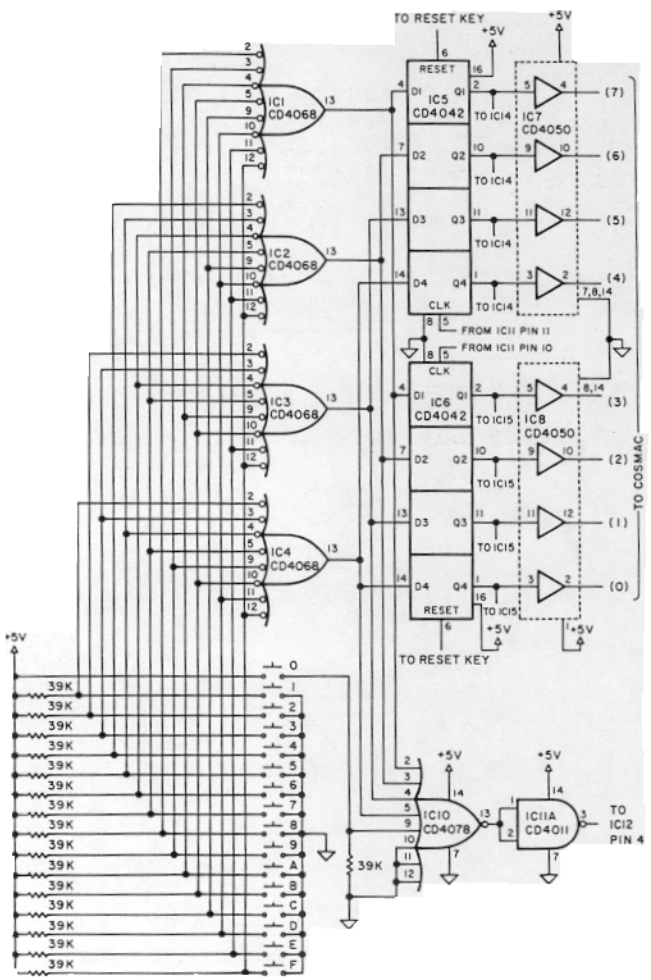


Fig. 1. Keyboard encoders and latches.

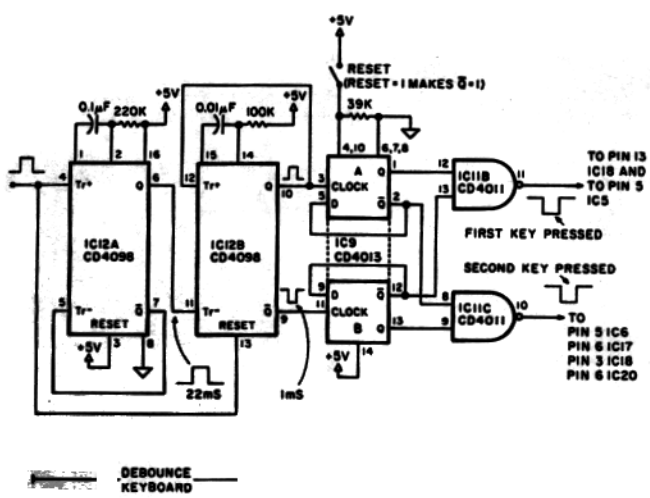


Fig. 2. Debounce and strobe circuits.

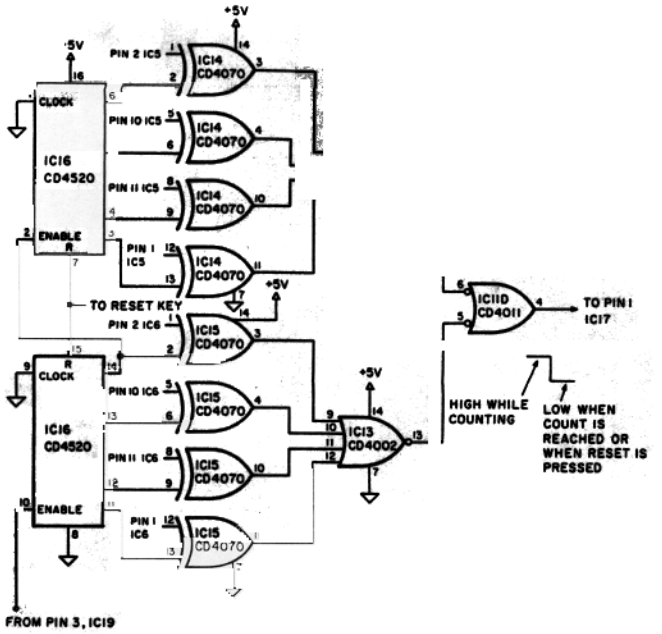
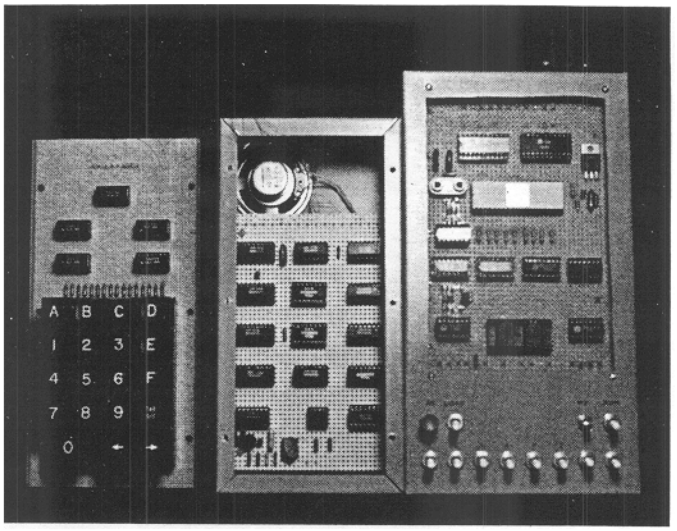


Fig. 3. Counter and compare circuits.



Right: The COSMAC Elf. Center: Keyboard support circuitry with top removed and shown at the left.

keyboard; the remaining ICs mount on the second perfboard. IC sockets were used throughout, and wiring was accomplished by working several evenings using a Vector wiring pencil. The keyboard has several uncommited keys, one of which you must designate as a reset.

Circuit Description

ICs 1, 2, 3, 4 (Fig. 1) produce the proper code for each key pressed; e.g., when D is pressed you get 1101. IC10 detects the closure of any key, and its output is inverted by IC11A, which feeds a debouncing circuit made up of the two monostable multivibrators in IC12. IC12A (Fig. 2) triggers on the rising edge of the input, sending out a 22 ms pulse. Assuming that all contact bounce is over in less than 22 ms, IC12B will trigger on the falling edge of this 22 ms pulse and send out a 1 ms pulse. When the key is released, IC12A may or may not generate another 22 ms pulse, depending on how bad the key bounce is. IC12B will not be triggered by this pulse because triggering occurs only on the falling edge, which does not occur until 22 ms after the key has been released. The moment the key is released, the low signal on the reset of IC12B prevents triggering. In summary, IC12 generates a single 1 ms pulse for each keystroke.

IC11B produces a 1 ms pulse on the first keystroke that gates the outputs of ICs 1, 2, 3, 4 into IC5, which then stores the binary code for the first key. IC11C produces a 1 ms pulse on the second keystroke that gates the outputs of IC1, 2, 3, 4 into IC6, which then stores the binary code for the second key. At the same time, this pulse goes to pin 3 of IC18, which inverts the pulse and signals COSMAC that the data is ready to go into memory. The pulse is not synchronized to COSMAC's internal timing, but it lasts much longer than a timing cycle, guaranteeing that the memory will get the data. The third and fourth keystrokes behave like the first and second, a pulse going out after the fourth to signal that the data is ready again. The whole operation is fully automatic, and there is never any need to press the input button. IC7 and IC8 are buffers required to drive the COSMAC circuitry.

Figs. 3 and 4 show the circuit I devised to advance to any desired step in the program. In effect, all these circuits do is push the input button the required number of times to get to whatever program step you enter on the keyboard. For example, you want to go to step OB — so you press reset, set memory protect, reset COSMAC with the load switch and enter OB. IC19 will send out 12 pulses, which, through IC18, advance to step OB. IC19 oscillates at 1000 Hz, so the time to get to the last step in the program, FF, is about 256 ms. All other steps will be reached in less time.

IC19 drives the first four-stage binary counter in IC16, and the output of this counter drives the second one. Together these can count to any desired value from 00 to FF, and their outputs are in binary form. Let's say you enter B7 on the keyboard. IC19 starts oscillating and IC16 starts counting the pulses. IC14 and IC15 compare the output of the counters to the B7 stored in IC5 and IC6; when the counters get to B7 a signal is sent out via IC13, IC11D and IC17 to stop IC19. IC19 sends out exactly 183 pulses, and assuming COSMAC was

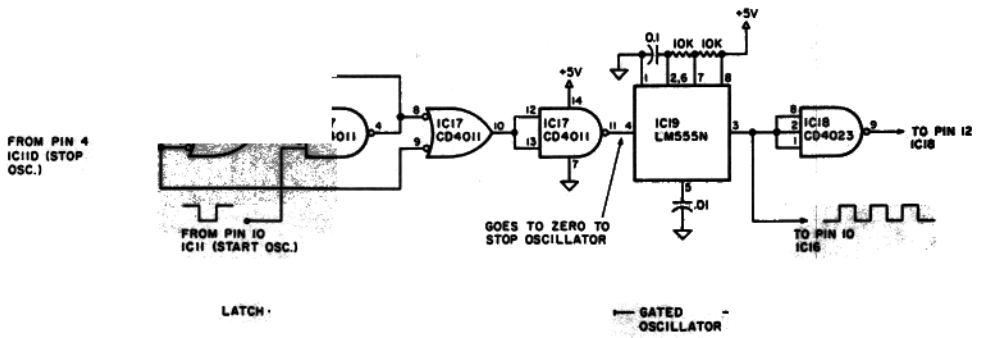


Fig. 4. Oscillator circuit.

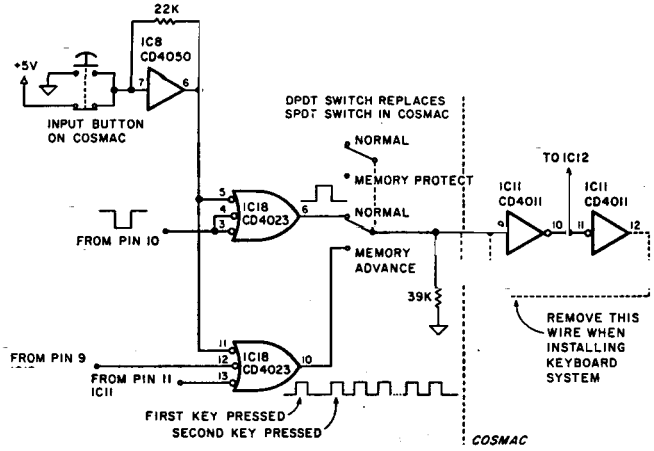


Fig. 5. COSMAC interface.

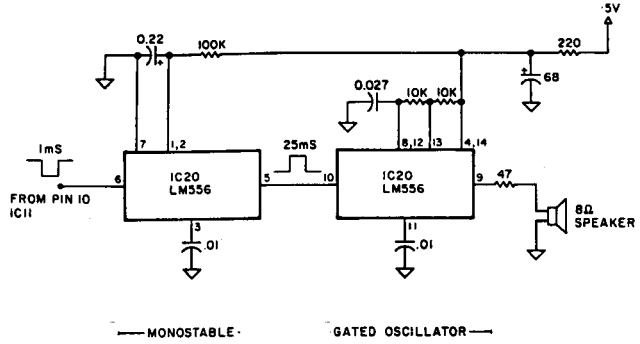


Fig. 6. Optional circuit to provide beep on every second keystroke.

at step 00 initially, you would arrive precisely at step B7. In Fig. 5 it will be noticed that at pin 10 of IC18 you get one extra pulse from the first keystroke. This is needed to set the program back to step 00, and it eliminates the need to press the input button.

Fig. 6 shows an optional circuit that provides an audible beep on every second keystroke. The reason for including this is that sometimes when your are entering a lengthy program you will strike a key, but not press it hard enough to actually make the contact. You go on your merry way, not realizing your mistake because you haven't been looking at your hex display. You end up having the last digit of each entry paired with the first digit of the following entry. It's quite a mess, believe me!

IC20 is a dual timer, the first half of which stretches that 1 ms pulse into a 25 ms pulse. The second half is connected as an astable multivibrator driving a loudspeaker. It is gated on for 25 ms and gives a beep. A 68 uF capacitor and 220 Ohm resistor decouple this circuit from the 5 volt line and reduce current consumption. If a longer beep is desired, increase the 0.22 uF capacitor. If a different pitch is desired, change the 0.027 uF capacitor. Any speaker will probably do; I used a 2-inch one from Radio Shack.

Modification of COSMAC Circuitry

The eight front-panel switches must be disconnected. The outputs of IC7 and IC8 in Fig. 1 are connected in their place. The memory-protect switch must be replaced with a DPDT toggle switch, one-half of which is wired up as in the original, while the other half is wired as shown in Fig. 5. The input button is rewired as shown and is debounced by an unused part of IC8. As shown, you must remove the wire joining pins 9 and 12 of IC11 in COSMAC.

Using the System

To enter data into memory, press reset, set COSMAC to load and start using the keyboard. To check what you've entered at a particular step set memory protect, reset the load switch and leave on load, press the reset key and enter the step you want to see on the keyboard. You can then change the data in the following step by just switching off the memory protect and keying in whatever you want.

The input button behaves as it did in the original design, and it can be used if the program calls for it to be pressed; or it can be used to advance the memory one step at a time.

Summary

The whole project cost me around $55 here in Canada where prices tend to be higher. It has been worth every penny in the increased convenience and speed it provides in programming. ■

11/78