Interfacing the Elf II

Did you enjoy "The Amazing 1802" back in August? Here's more info from the same source.

Robert J. Cotter
The Johns Hopkins
School of Medicine
Baltimore MD 21205

How many times, while entering a long program on my Elf II, have I suddenly wondered if I'd missed an entry? I'd then go back again to the beginning and check each set of bytes. Or I'd move my logic tester along the address bus to see where I was, compare that with my written program and see if the contents on the display corresponded to the correct address. Other times when I would make a mistake on an entry, I'd have to step the INPUT switch from the very beginning up to where I'd made my error and correct it.

At first I considered connecting an LED to each of the address lines, but the thought of reading binary numbers while my listing was in hexadecimal was not very appealing. Finally, I decided to hard-wire decode the address lines and display them in hexadecimal. And, while I was at it, why not decode the full 16-bit addresses, so that my circuit wouldn't become obsolete when I bought that new 4K memory expansion board that is now available for the Elf II?

In this article I describe two simple circuits for decoding and displaying all 65K address locations. In addition, I have written a short program that can be used with the second circuit as an *operating system* to display any address, change its contents and step through or execute a program beginning at any location. In combination with the address decoder and display, the operating system has been invaluable as a debugging tool.

16-Bit Addressing—How It Works

By placing eight high-order bits and eight low-order bits of an address on the lines at different times in the machine cycle, the 1802 needs only eight addressing lines to transfer all 16 address bits to and from its memory. The original Elf II, which has only 256 memory locations, utilizes only the low-order bits. The high-order bits are needed for memory expansion and are generally decoded for "page selection."

Fig. 1 shows a typical *timing diagram*. Note that the timing pulses TPA and TPB appear during the different parts of the cycle when the high-order and low-order bits are on the address bus. These timing pulses can be used to latch each half of the 16-bit address. Each of the 8-bit bytes can then be separately decoded for display as two hexadecimal digits.

Circuitry

A simple 16-bit address display is shown in Fig. 2. The eight address lines are first buffered using CMOS 4050 hex noninverting buffers. TPA and TPB are buffered with 4049 inverters to provide the necessary *low* enable pulses for latching the address bits onto the four 9368s.

The 9368s are latches, decoders and drivers all in one integrated circuit, and, unlike other 7-segment decoders, they decode all 16 binary states. When the high-order address bits appear on the lines, the inverted TPA pulse causes two of the 9368s to latch and display their 2-bit hexadecimal equivalent. When the low-order bits appear, $\overline{\text{TPB}}$ latches these on the other two displays.

Once you have built this circuit, you can enter a program and see both the memory contents and the address displayed simultaneously as you input each memory byte. As mentioned before, this can be quite helpful when you enter a long program, where it is easy to miss a memory byte. Furthermore, if you have built the 1 Hz clock recommended in "The Amazing 1802" (August 1978, p. 102), you can debug a program by executing it slowly (8 seconds/machine cycle) and watching the address display.

I have found many programming errors with this circuit by noting that a memory byte was fetched from a different location than I had intended. The address display will indicate just how far a program executed successfully.

You may, however, wish to have a circuit that also gives you access to the 16 address bits, which you can then use for selecting pages of memory or for memory-mapped I/O. This can be accomplished by modifying the circuit in Fig. 2 and placing a second set of latches between the address buffers

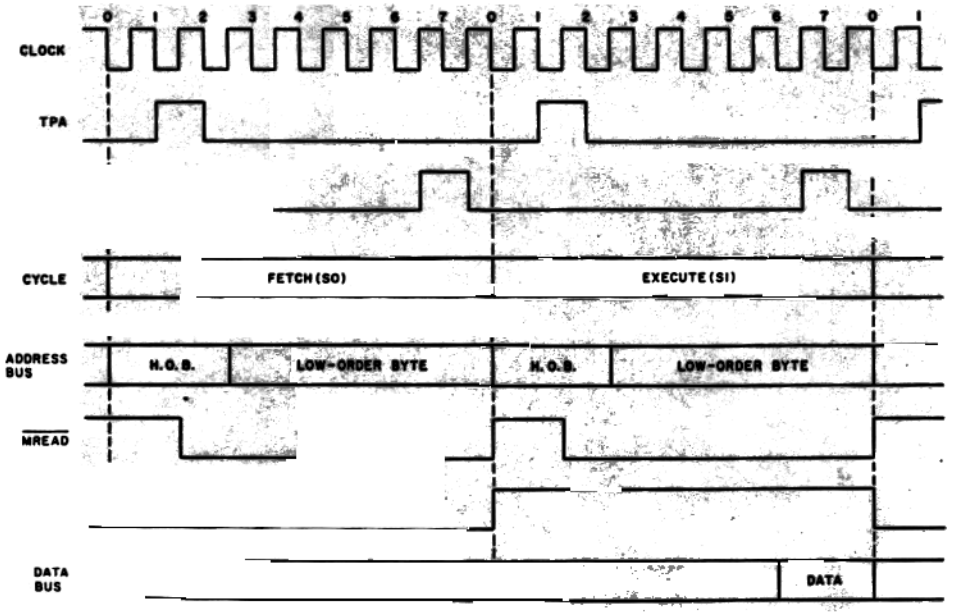

Fig. 1.

and the 9368 latch/decoder/drivers.

This circuit is shown in Fig. 3 and uses two 74100 8-bit latches. (Alternatively, four 7475 4-bit latches work just as well.) Since the 74100 is TTL logic, the latched address outputs (A0 to A15) can be connected to a number of memory boards or output devices. Circuit operation is similar to that of Fig. 2. TPA latches the high-order bits on one of the 74100s, and TPB latches the low-order bits on the other.

Note, however, that $\overline{\text{TPB}}$ now latches all 16 bits simultaneously on the 9368s, so that all of the displays change at exactly the same time. This is especially convenient when a program with the 1 Hz clock is being executed.

An Operating System

Whenever the RUN switch of the Elf II is depressed, the memory pointer always starts execution at location 00 00 in the memory. This is inconvenient if a number of programs are to be stored in different locations of the memory. While you could INPUT a GOTO command in the first few locations to start excution at any point, this would not solve the problem of correcting a single byte further along in memory or getting a listing of a program without having to step up to the beginning location.

A simple *operating system*, on the other hand, would allow the user to set the program pointer at any location in memory, examine the contents of that location, change the contents, step through the program or begin execution at that point. The *operating system* presented in Table 1 does just that. However, to make the address display useful for this system, one minor hardware change must be made.

When the *operating system* has been loaded and the RUN switch depressed, the operating program sits in a loop awaiting input from the operator. Input consists of a two-byte address location and an instruction either to examine an entry, change the contents of a memory location or begin execution. This loop results in a hopeless jumble of numbers on the address display. Since the

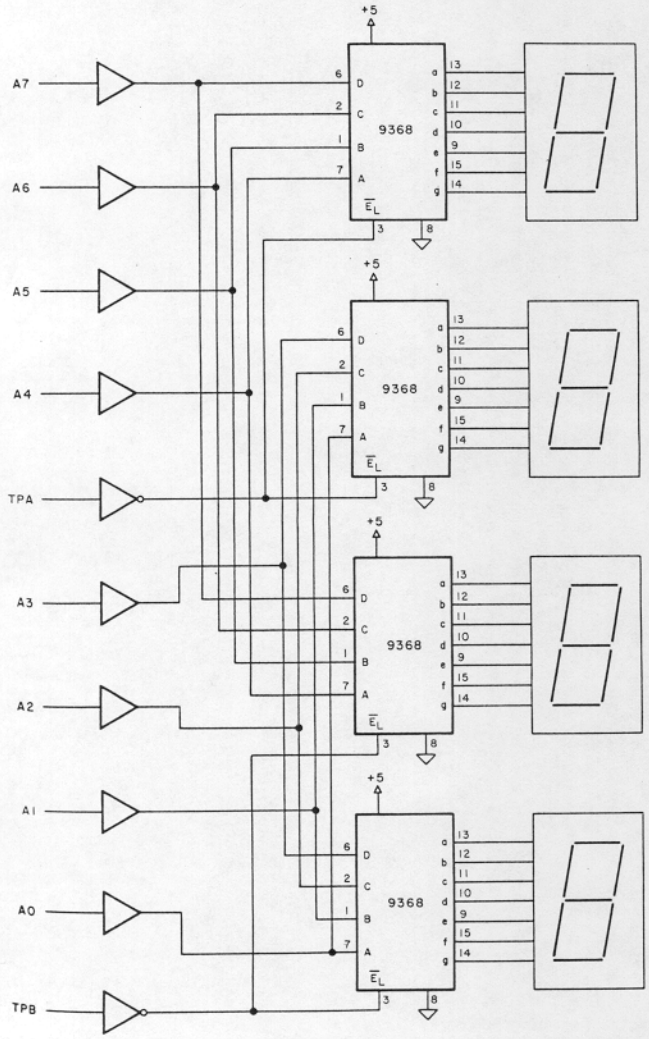

Fig. 2.

Address	Bytes	Comments
00 00	F8 00 B1	Set location for storage area
03	F8 38 A1	of starting address at 00 38, using R1 as pointer.
06	E1 64	Set X = 1. Show location 00 38 on address display, and contents on data display. R1 + 1.
08	7B 3F 08	Wait for INPUT switch ON
0B	7A 37 0B	and OFF.
0E	6C B2 64	Load the high-order byte of the starting address from the keyboard into R2.1 and location 00 39. Show 00 39 on address display and high-order byte on data display. R1 + 1.
11	7B 3F 11	Wait for INPUT switch ON
14	7A 37 14	and OFF.
17	6C A2 64	Load the low-order byte from the keyboard into R2.0 and location 00 3A. Show 00 3A on address display and low-order byte on data display. R1 + 1.
1A	7B 3F 1A	Wait for INPUT switch ON
1D	7A 37 1D	and OFF.
20	6C	Load keyboard number into register D and location 00 3B.
21	FB EE	If keyboard number is EE,
23	32 2C	then go to location 00 2C.
25	7B 6C	If not, light Q. If the keyboard
27	FB CE	number is CE, then go to lo-
29	32 2C	cation 00 2C.
2B	D2	If neither, then set program pointer at the starting location and begin execution.
2C	E2 64	Display the starting address and its contents. Increment register R2.
00 2E	3F 2E	Wait for INPUT switch ON
30	37 30	and OFF.
32	39 2C	If Q is OFF, skip the WRITE step and return to 00 2C to display the next location.
34	6C	If Q is ON, write the keyboard number into the next location.
35	30 2D	Return to 00 2D to display new contents and address.
37	00	END
00 38		Storage
39		area
3A		for
3B		register
3C		R1.
00 3D		Begin user programs.

Table 1.

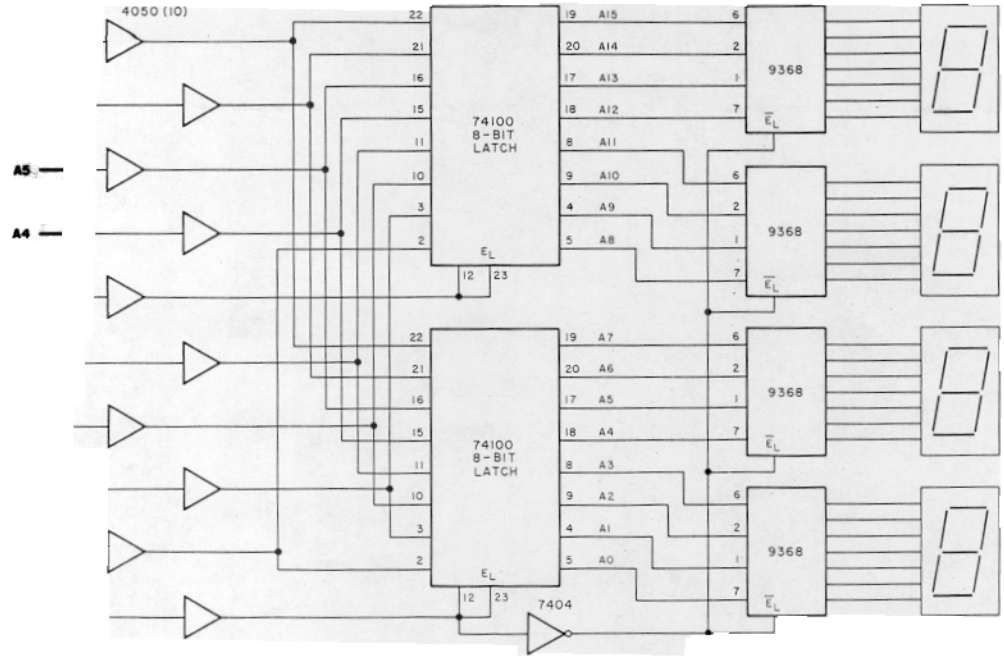

Fig. 3.

addresses of interest are those for which the contents are also being displayed, the 9368s should be enabled by a 64 instruction.

Fig. 1 also includes the timing relationships for the 64 instruction. In the FETCH cycle the 64 instruction is fetched from memory. In the EXECUTE cycle it will read a number from a location in memory and place that number on the DATA display.

During the EXECUTE cycle the N2 line goes *high*. The address display will show an address location following a 64 instruction if you replace the 7404 inverter in Fig. 3 with a NAND gate, with TPB and N2 as the inputs. The circuit is shown in Fig. 4 and includes a toggle switch to allow normal operation when you wish to watch a program execute slowly.

In the *timing diagram* (Fig. 1), the high-order address byte will be placed on one of the 74100 latches when TPA is *high*. When TPB is *high* and $\overline{\text{MREAD}}$ is *low*, the low-order address byte will be placed on the other 74100, and the contents of that location will be read. When N2 is *high*, both the DATA and ADDRESS displays are enabled.

Using the Operating System

With the toggle switch in the NORMAL position, load the operating program. Then switch the toggle switch to OPS and depress the RUN switch. The ADDRESS and DATA displays will show:

00 38 ww

giving the first location in the storage area for the operating system and its contents. Enter the high-order byte (e.g., AB) of the starting address. The ADDRESS and DATA displays will show:

00 39 AB

indicating that the high-order byte has been entered in the second storage location. Enter the low-order byte (e.g., 42). The displays will then show:

00 3A 42

If you wish to *examine* an *entry*, enter EE. The display will then indicate the full address and contents:

AB 42 xx

If you press the INPUT switch again, it will show the next address and contents:

AB 43 yy

so that you can also step through a program beginning at AB 42 and examine the contents at each location.

If you wish to *change* the entry at location AB 42, then depress the RUN switch and enter the previous address, AB 41. Then enter CE (change entry). The computer will display:

AB 41 zz

Enter the number you wish to place in AB 42 (e.g., F8). The displays will then show the new contents at that address:

AB 42 F8

and you can continue to enter numbers into each location starting from AB 42. This allows you to enter a program anywhere in the memory, as well as correcting an entry.

To run a program starting at location AB 42, enter AB 42 as before and depress the INPUT switch one more time.

Some Further Notes

The 9368s, when connected directly to common cathode 7-segment displays, draw a lot of power. This can be minimized by making all of the connections through 120 Ohm resistors as is done for the data displays on the Elf II. There seems to be no loss in brightness. Of course, all additions to the Elf II—address displays, input/output buffers, decoders, D/A converters, etc.—should be powered by a separate 5 volt supply, since the Elf II supply is capable of handling only the original circuits.

If you wish to keep the *operating system* stored in the memory, a battery backup system for the original 256 location RAM is a good idea. A circuit for doing this has been described by Joseph Weisbecker in *Popular Electronics* (September 1976).

Netronics, the manufacturer of the Elf II, has recently produced a ROM monitor for the Elf II, which contains an operating system. It takes a "software" approach to the address decoding problem and uses the DATA displays to display adresses. It also contains a cassette interface program. However, it presents only the low-order address byte on the displays and gives alternate, rather than simultaneous, readings of addresses and contents.

In addition, there are distinct advantages of having a hardwired connection to the address lines. The address decoding system described here can give memory address information during slow execution of a program, which the monitor cannot do.

However, since there are advantages to both systems, they may both be added to the basic Elf II. When the operating system described here has been loaded, entry of the address F0 00 will put the computer into the ROM monitor.■

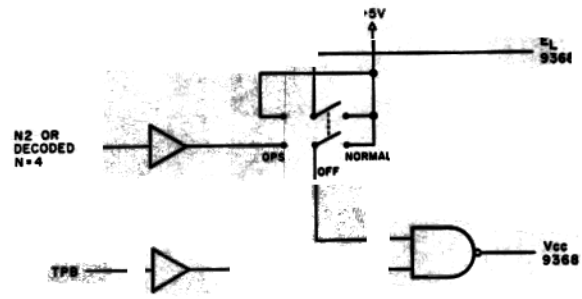

Fig. 4.