Experimenter's Corner

THE 74150 MULTIPLEXER

By Forrest M. Mims

DIGITAL logic circuits are amazingly versatile, and the multiplexer or data selector is no exception. The multiplexer, sometimes abbreviated MUX, provides a means of selecting one of several inputs and steering the logic level at that input to a single output. The selected input is determined by a binary address applied to one or more data select inputs. That's why multiplexers are often called *data selectors*.

Multiplexers have many applications. We'll look at several of them in this column, but first let's find out how the multiplexer works.

A Simple 1-of-2 Multiplexer. Figure 1 shows a simple two-input MUX with a single data select input. The circuit is called a 1-of-2 multiplexer because one of the two inputs is routed to the output at any given moment according to the status of the data select input. The binary bit pattern at the data select input is called an address because each possible data select bit combination (in this case 0 and 1) selects one and only one input.

Assume the data select input is logic 0. This means one of the inputs to AND gate A is logic 1. The gate is then able to provide a high or low output depending upon the state of the DATA A input. Simultaneously, one of the inputs to AND gate B is low so its output will be low no matter what logic state appears at its second input (DATA B). If DATA A input is low, both AND gates will have a low output. If DATA A is high, the output of AND gate A will go high. The OR gate (C) will respond to either condition with, respectively, a low or high output.

If this explanation seems hard to follow, Fig. 1 also shows the truth tables for the 1-of-2 MUX as well as those of the inverters and gates from which it is composed. With this information you should be able to decipher the circuit's operation on your own.

Advanced Multiplexers. The simple multiplexer shown in Fig. 1 illustrates the basic operating principal but has limited utility. You can easily breadboard a working version of it if you want to explore its operation in detail. Far more flexibility, however, is available in the form of single-chip multiplexers having eight or sixteen inputs. Several such chips are readily available. TTL versions include the 74150 1-of-16 multiplexer and the 74151 and 74152 1-of-8 multiplexers. Another TTL MUX is the 74153 *dual* 1-of-4 data selector.

A number of multiplexers that select one of several multiple-bit words rather than single bits are also available, as is a family of CMOS multiplexers.

The 74150 1-of-16 Multiplexer. The pinout of the 74150 1-of-16 MUX is shown in Fig. 2. This data selector has sixteen data inputs, any one of which can be selected by applying the appropriate 4-bit data select word or address to the four-line data select input. The 74150 also has an *enable* or *strobe* input. This input must be low (grounded) for the 74150 to function. When the enable input is high (disconnected or tied to V_{CC}), the output will be high regardless of the status of the selected input.

The simple 1-of-2 MUX we looked at earlier has an OR gate output. The 74150, however, uses a NOR gate output. This means the output is an inverted version of the selected input. Be sure to keep this in mind when using the 74150 in an actual circuit.

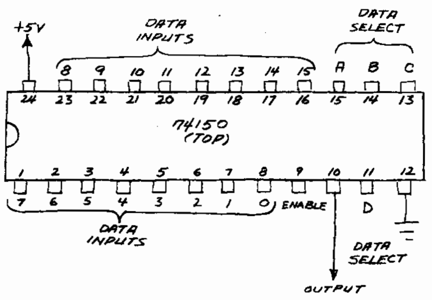


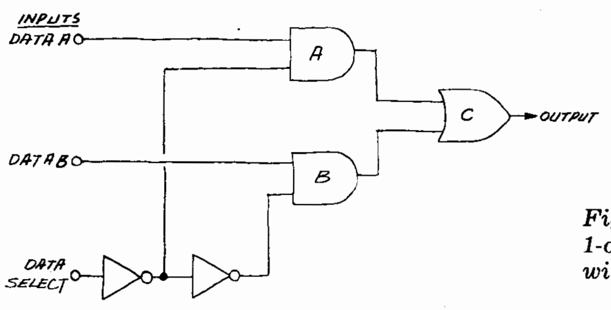
Fig. 2. Pin outline of 74150.

"Universal Logic Gate." It's often necessary for a digital circuit to generate the type of truth table that is not available from standard gates. Simple truth tables can be generated by interconnecting the gates in a 7400 quad NAND gate or other common gate package. Complex truth tables can be implemented with a ROM. The 74150 MUX offers a very simple way to generate a truth table with four inputs and a single output.

Since the 74150 has sixteen input addresses, there are 2^{16} or 65,536 possible input/output combinations. Here's a truth table for just one sequence.

	Address	Output
0	0 0 0 0	1
1	0 0 0 1	1
2	0 0 1 0	1
3	0 0 1 1	1
4	0 1 0 0	0
5	0 1 0 1	0
6	0 1 1 0	1
7	0 1 1 1	0
8	1 0 0 0	1
9	1 0 0 1	0
10	1 0 1 0	1
11	1 0 1 1	1
12	1 1 0 0	0
13	1 1 0 1	0
14	1 1 1 0	1
15	1 1 1 1	1

Designing a logic circuit to implement



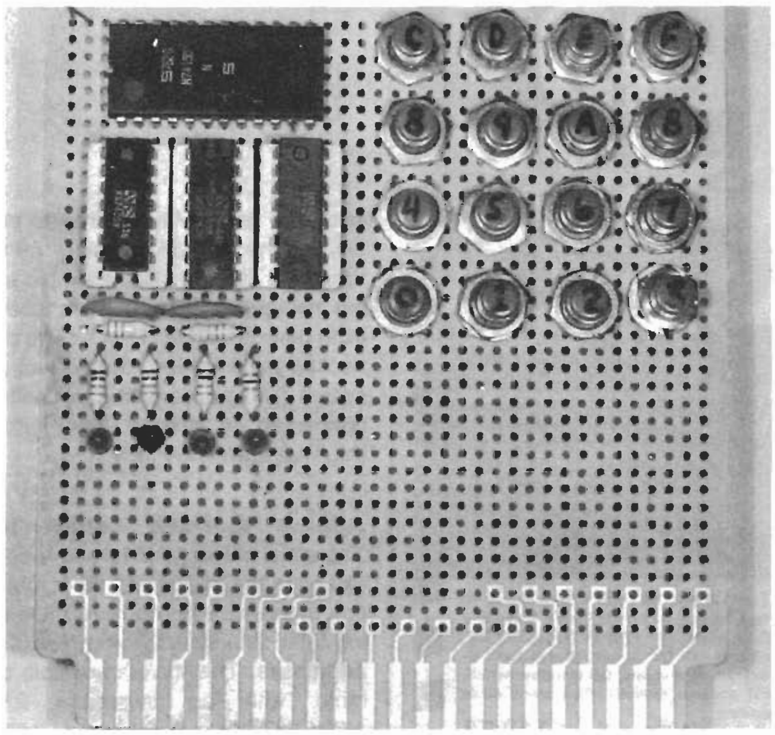
SELECT	DATA A	DATA B	OUTPUT
0	0	X	0
0	1	X	1
1	X	0	0
1	X	1	1

Fig. 1. A simple 1-of-2 multiplexer with truth table.

this truth table would be both tedious and time consuming, but we can complete the entire design in less than a minute with the help of a 74150! All that's necessary is to place the complement of the desired output for each address at the appropriate input.

Complements of the desired outputs are placed at the inputs because the 74150 inverts the data at its inputs. Figure 3 shows how the 74150 is wired to implement the truth table. The 74150 in-

PROJECT OF THE MONTH



A HEXADECIMAL KEYBOARD ENCODER

The simple hex keyboard encoder described in this month's column can be significantly improved by adding a 4-bit register to store the hex code of the key that has been pressed. This means the LED readout will display the 4-bit code for a particular switch when the switch is initially closed and continue to display it *after* the switch has been released. The display will change only when the scanning circuit detects a new key closure.

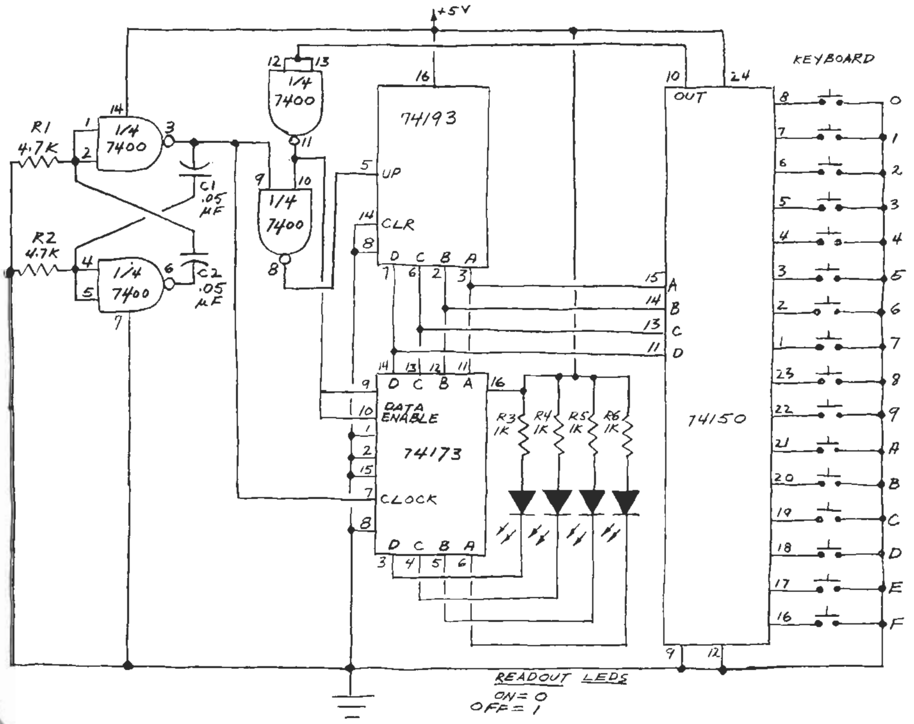
Compare the complete circuit diagram for the hex encoder shown here with Fig. 7 and you'll note that a 74173 4-bit register has been added and the readout LEDs have been moved from the counter output to the register output.

The clock and scanner portions of the circuit have already been analyzed, and the 74173 was described in the March 1978 Experimenter's Corner. It's a flexible storage register with 3-state outputs that can be readily tied to the address or data bus of a microcomputer or controller.

When the scanner circuitry detects a switch closure, the output of the 74150 sends a data enable pulse to the 74173 through one of the 7400 gates. The next clock pulse then loads the counter address into the 74173, and the circuit resumes its sequential scan of the switches. The four bits describing the previously closed switch, however, remain safely stored in the register.

The photograph shows a prototype version of the encoder assembled on a perforated board (Radio Shack 276-152 or equivalent). Note the extra space on the board for the addition of other circuits such as a RAM. Also note that a standard calculator keyboard was *not* used. These keyboards are inexpensive and readily available, but the switches are arranged in x-y format not compatible with this circuit. Instead, individual normally open pushbutton switches were used to make a custom hex keyboard.

I used wrapped-wire construction throughout with the exception of solder connections to the switch terminals, the LEDs and several of the resistor and capacitor leads. Total assembly time was about three hours. In a subsequent Project of the Month we'll add a 16-word RAM to the encoder. You'll find the resulting circuit very interesting, so be sure to consider building the basic encoder in the meantime. ◇



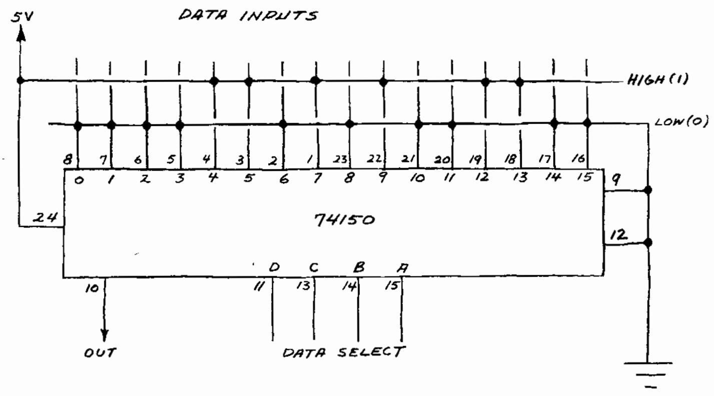


Fig. 3. How to connect a 74150 to implement truth table in text.

puts can be quickly rewired to provide any of the 65,536 possible combinations. For best results, a simple switching network can be used to speed up truth table changes. This is done by connecting each input to the pole of a spdt switch. The positions of each switch are connected to ground (low) and V_{CC} (high) as shown in Fig. 4.

Adding a Data Select Input. There's a clever way to add a data select bit to the 74150 and other multiplexers. In the case of the 74150, the resulting 5-bit data select word gives 32 input addresses. This makes possible a truth table with an incredible 2^{32} or 4,294,967,304 input/output combinations!

Let's assume you want to implement the following 8-input (3-bit address) truth table with a 4-input (2-bit address) MUX:

Address C B A	Output
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	1
1 1 0	0
1 1 1	0

A 4-input MUX normally implements a truth table with a 2-bit address (00-01-10-11). Look back at the truth table above and you'll notice this pattern is repeated *twice* under the B and A columns to give four pairs of 00, 01, 10 and 11. Rearrange the truth table so the identical B-A parts are adjacent to one another, and you'll notice the output for any of the four pairs can be low or high, C or the complement of C:

	A B C	OUTPUT
1.	0 0 0	0
	0 0 1	1
2.	0 1 0	1
	0 1 1	0
3.	1 0 0	1
	1 0 1	1
4.	1 1 0	0
	1 1 1	0

We're now ready to implement the truth table by connecting each of the four inputs of the MUX to low, high, C or the complement of C. The outputs of the first pair are identical to C, so C is connected to the first input. The outputs of the second pair are the opposite of C, so the complement of C is connected to the second input. Similarly, both outputs of the third pair are high so the third input is connected to logic 1. Both outputs of the fourth pair are low so the fourth input is

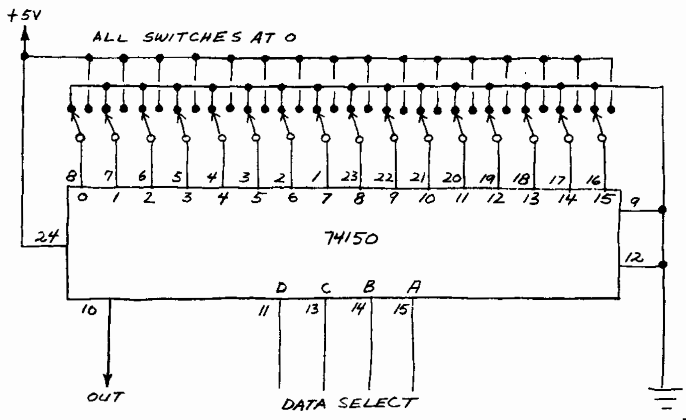


Fig. 4. Switch programmable 74150.

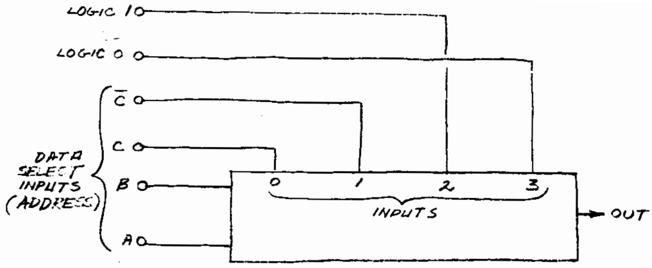


Fig. 5. Expanding a 4-input MUX to an 8-input MUS.

at logic 0. The resulting connection diagram for the expanded MUX is shown in Fig. 5.

Multiplexer Pattern Generator. One application for a MUX truth table circuit like the one in Fig. 4 is a *binary pattern generator* that produces a string of sixteen preselected bits over and over again. This is accomplished by connecting a 4-bit, modulo-16 counter to the address inputs of the 74150. Each pulse from a clock connected to the counter produces the next output in the binary sequence.

Figure 6 shows a working version of a clock and counter that can be connected to a 74150 to make a pattern generator. This circuit is fairly straightforward, and you can use other clock and counter combinations if you prefer.

Pattern generators have many applications. One simple possibility is to use the 74150 output to strobe a tone generator on and off to produce programmed tone patterns. Another is to produce programmed LED flashing sequences.

Hexadecimal Keyboard Encoder. A very useful application for the 74150 MUX is the hexadecimal keyboard encoder shown in Fig. 7. This circuit continuously scans each of sixteen normally open pushbutton switches. When a closed switch is detected, the LED readout connected to the data select (address) inputs of the 74150 identifies in binary form which switch has been closed.

Those of you who are microcomputer enthusiasts already know the value of a hex keyboard encoder, but *hexadecimal* probably sounds very intimidating to the uninitiated. The term simply means a number system based on sixteen. Binary, octal and decimal are number systems based, respectively, on 2, 8 and 10. Hexadecimal is important in digital logic because, as you already know, there are sixteen possible combinations of 0 and 1 in a 4-bit word (0000-1111). Here's a listing of the first sixteen decimal digits along with their binary and hex counterparts:

Decimal	Binary	Hexadecimal
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

A hex keyboard encoder allows any of the sixteen hex digits to be entered into a digital circuit by pressing a single key. This greatly speeds up the programming of read/write memories in microcomput-

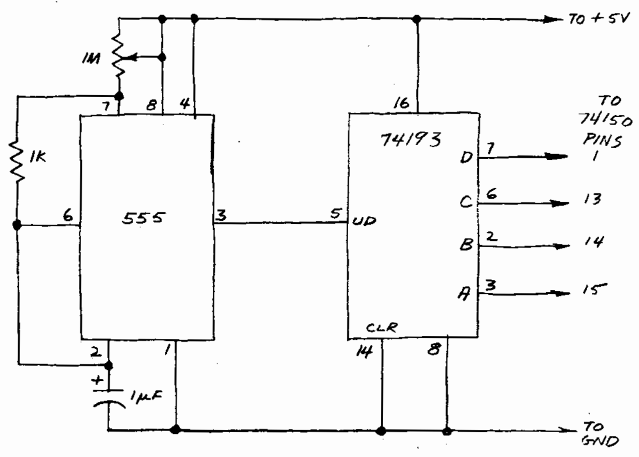


Fig. 6. Sequencer circuit for binary pattern generator.

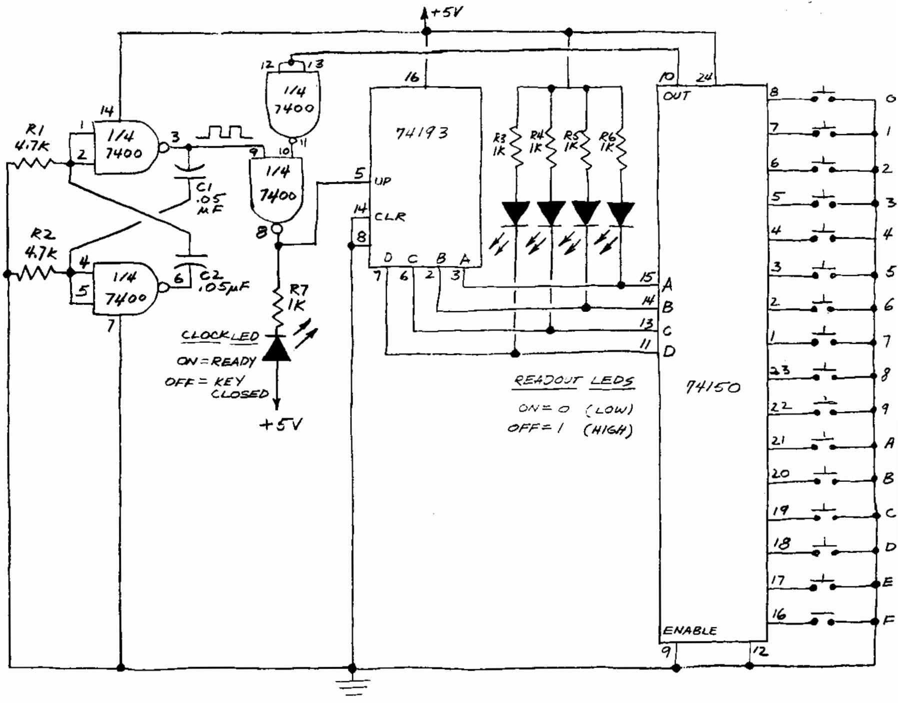


Fig. 7. Hex keyboard encoder circuit.

ers, controllers and other digital circuits that process data in 4-bit "nibbles."

Operation of the keyboard encoder shown in Fig. 7 is straightforward. An astable multivibrator comprising half the gates in a 7400 quad NAND gate serves as a clock. Pulses from the clock are steered to the count UP input of a 74193 counter through one of the remaining gates in the 7400. This gate permits clock pulses to pass as long as the output of the 74150 is low.

A switch closure causes the 74150 output to go high when the count from the 74193 matches the number of the closed switch. This, in turn, inhibits the clock gate, which prevents clock pulses from reaching the 74193. The four LED's connected to the 74193 output then indicate the bit pattern applied to the 74150 address in binary and thus the number of the closed switch.

When the closed switch is released, the 74150 output returns to the low state and the clock gate is no longer inhibited. The 74193 then continues to cycle the 74150 through a sequential scan of the switches.

An interesting feature of this circuit is that under normal conditions it operates as a form of *priority encoder*. This means it responds to the first of two or more key closures while ignoring all subsequent closures. Can you think of a circumstance where the encoder would respond to the *second* of two key closures? (Hint: Assume the clock is very slow or your fingers are very fast.)

The circuit shown in Fig. 7 is not necessarily usable in some applications. One reason is that the continuous scanning of the switches causes the four LEDs to appear continually on (binary 0000) between switch closures. This problem is partially alleviated by the single CLOCK LED that indicates when the clock pulses are getting through to the counter.

When the CLOCK LED is on, the circuit is ready to receive another switch closure. In other words, the 74150 is scanning the input switches and the fact that all four readout LEDs appear to indicate 0000 is irrelevant. When the CLOCK LED is off, a switch closure has been detected and the readout LEDs accurately indicate the selected switch. If, by coincidence, switch 0000 is selected, the four LEDs will all remain illuminated but their apparent brightness will increase significantly. This is because all four LEDs now receive a steady direct current rather than the very rapid pulsating current from the 0000-1111 count sequence.

Another drawback of the circuit in Fig. 7 is that it promptly "forgets" a switch closure once a switch has been released. Contrast this with the pocket calculator, which "remembers" each key closure until an operation is complete or a clear key is pressed. This disadvantage can be overcome by adding a data storage register to the circuit. For details, see the accompanying Project of the Month. ◇