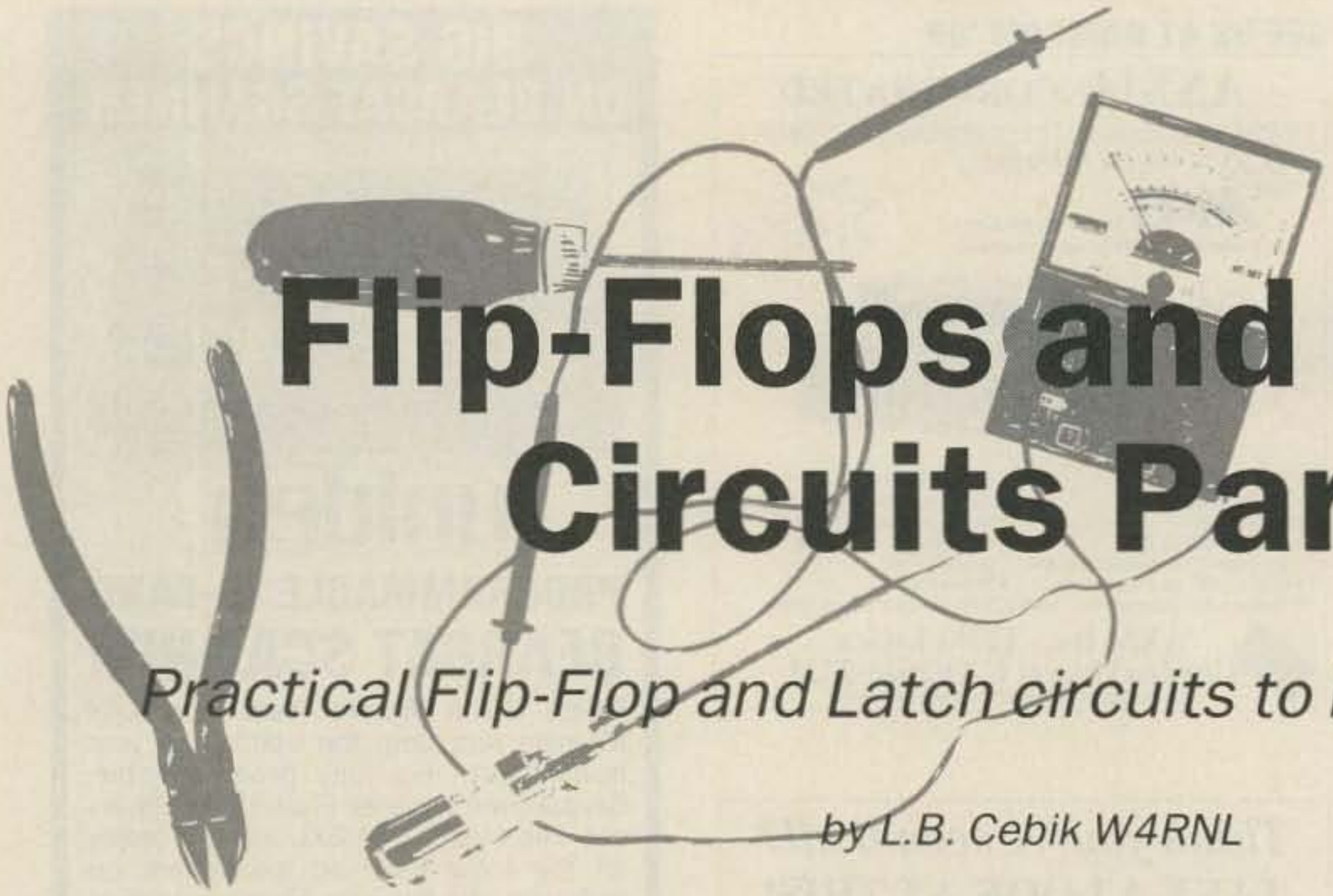

Flip-Flops and Latch Circuits Part 2

Practical Flip-Flop and Latch circuits to build, test, and use.

by L.B. Cebik W4RNL

Flip-flops as Shift Registers

A shift register is a circuit that takes data (one or more HIs or LOs) and moves it from one place to the next. Suppose there are 4 storage places, as in Figure 12. When taking the data from the state of the SI input, there is a serial input shift register. When taking data from the 4 PI inputs, the shift register is parallel input. It's possible then to move data from either source to the right toward output SO, the serial output. It shows the latest data after each clock pulse. The experimenter can also take the revised pattern of data from the PO terminals after each clock pulse. Thus, he gets the abbreviations SISO (serial in, serial out), SIPO (serial in, parallel out), PISO (parallel in, serial out), and PIPO (parallel in, parallel out), sometimes seen in shift register articles.

There are numerous all-in-one shift register chips, but these ICs have limited control capabilities. To see all of the controls one can exert on a shift register, consider basic flip-flops again. Just two 74HC74s can make a shift register, as shown in Figure 13. Once an experimenter has mastered the ways to control a shift register, he can select a more compact arrangement in a single chip if the application requires only a few of the controls.

The serial input goes to the data pin of the left-most flip-flop. Whatever value (HI or LO) appears here is shifted to the second flip-flop whenever the clock line shifts from LO to HI. That value continues to shift right with each clocking, since whatever appears on the D input appears at the Q output, which is connected to the next D input. The last Q output represents a serial output for the string of flip-flops. Note also that it's possible to use the −Q output for values opposite to the original.

The builder can also set the flip-flops with a parallel code. In Figure 13, LOs go wherever needed, since the resistors hold the Clear input HI in the rest state. LOs on the clear line immediately show up as LOs at Q and HIs at −Q, if beginning with the preset lines HI. In this case, tie the D input of the leftmost flip-flop to the positive supply voltage. Reverse this by holding the clear lines HI and impressing LOs on the preset lines (which would be HI at rest with pull-up resistors). A LO on preset appears as a HI at Q, and a LO at −Q. For this alternative, ground the D input of the leftmost flip-flop. Once the flip-flops are set, remove the LOs from the inputs. In other words, only a brief pulse at each flip-flop input line is needed to enter the parallel code. For these experiments, if the circuit is wired as in Figure 13, use a jumper to ground the clear line and then remove the jumper. Using the clear line removes the worry about debouncing the pulse.

Once a LO hits the clear input, the Q and −Q outputs change and stay changed until further input. As a start, place a LO on just the leftmost Clear input and watch it move along as the clock pulses proceed.

Once having entered the parallel code, it's possible to shift it right with a clocking pulse, just as was done in serial shifting. In fact, the experimenter can take the parallel code out the right end of the register in serial order, using either the Q or −Q output. Be sure that the first element desired is entered into the rightmost flip-flop and the last element desired is entered into the leftmost flip-flop. In a practical circuit, this operation requires a timing pattern consisting of a period of time

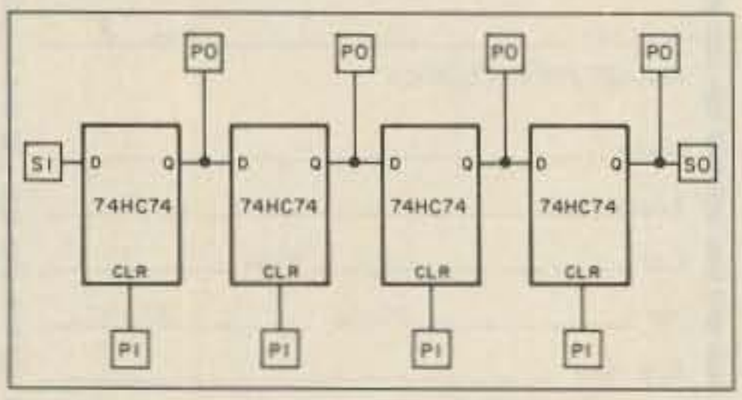

Figure 12. Basic flip-flop shift register.

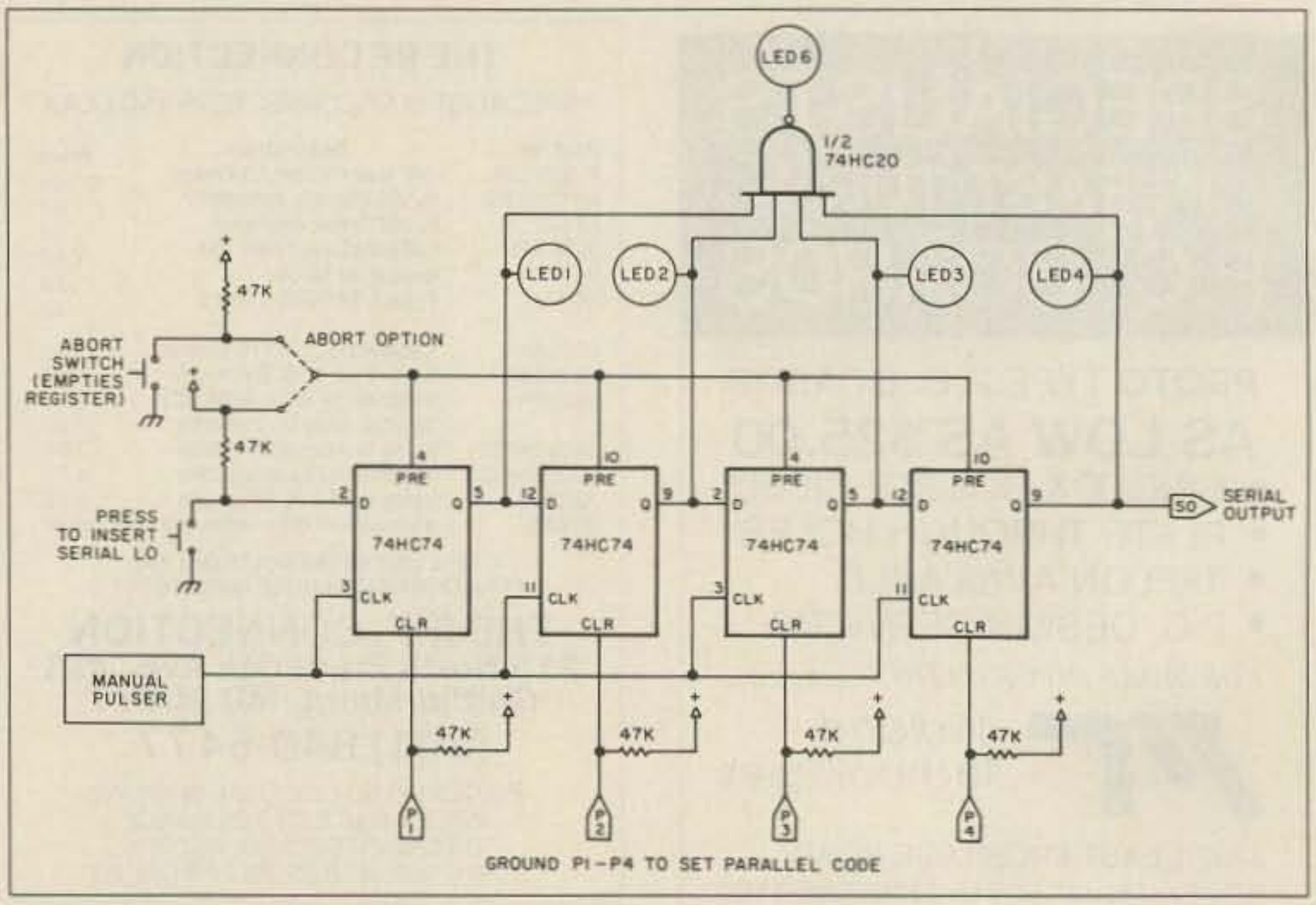

Figure 13. A parallel-code controlled shift register.

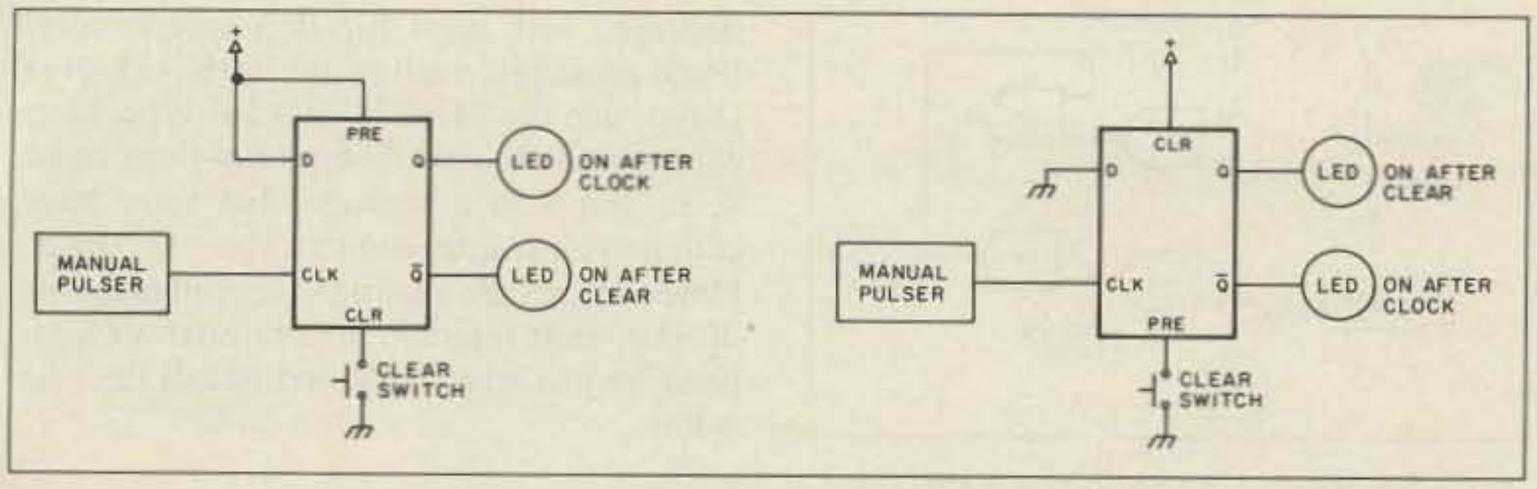

Figure 14. Basic flip-flop controller circuits.

to enter the parallel code and then four clock pulses to remove it serially. For these experiments, the manual pulser can move the code one step at a time and examine the results during these steps.

Why not take the outputs in parallel fashion from either the set of Q outputs or the set of −Q outputs? This allows some interesting possibilities. For example, if the register had 8 flip-flops, a user could enter an 8-bit code. Then he could use the rightmost four Q outputs to get the first four bits, clock four times, and take the last four bits from the rightmost outputs. Another application might use gates tied to each Q output to form a detector, such as the LED detector shown in Figure 13. If any flip-flop holds a LO from the clear input, then the clock keeps operating. When every flip-flop holds a HI (due to the HI on the leftmost D input, which fills in behind the shifting LOs), the detector might do something more than extinguish the LED. It might shut off the clock, preparing the register for a new code entry. This scheme would be applicable to both serial and parallel output uses.

Suppose a user wants to clear the register before it had gone through all its shifting cycles. In Figure 13, I made sure all the preset lines were HI so that the register would set properly using LOs to the clear lines and used only a brief set of pulses to the clear inputs. Setting the preset lines LO puts all the Q outputs to HI. With the detector just mentioned, the clock would stop, and the register would be ready for a new code entry. This type of abort operation may be handy in error detection circuits. Of course, if the register set initially with a code entry through the preset lines, making the clear line LO will force all the Q outputs LO, thus clearing the register in that mode of operation.

Given all these control possibilities, the user should probably start any register design on paper with individual flip-flops. Once having determined all the control needs, he then can look through the data books to see of there is a more compact shift register that provides what he needs with fewer chips and connections. The user can always go back to basics if he finds that a certain control will come in handy.

Among the 74HCOO-series ICs, there are numerous single package shift registers. The 74HC194 and -195 have 4-bit parallel inputs and outputs, but the 194 adds a serial input. The 74HC164 through -166, along with the -594, -595, -597, -598, -299, and -323 offer 8-bit shift registers in various configurations of serial and/or parallel inputs and serial or parallel outputs, with some limited controls that include tri-state outputs.

Flip-flops as Controllers

The basic flip-flop switch debouncing circuit in Figure 4 gave the most fundamental flip-flop control. That application didn't require data and clock inputs. However, for some types of control needs, those inputs come in handy.

Figure 14 shows one kind of control in two versions. Here a short enable pulse enters the preset or clear input to set Q and −Q. The control will involve a short-term operation turned on by the Q or −Q values. Note that the data input holds a constant value which—if clocked through to the Q and −Q outputs—would disable the operation. The applications requires a LO-to-HI transition from the operation that marks the completion of its work. (Of course, one can use a HI-to-LO transition by inserting an inverter in the line.) With the right transition, feeding it the clock input will shut down everything. For example, a builder might use the output of the shift register detector to signal when it was empty and shut down data code processing operations. The key points, again, are separate sources for start and stop signals and a constant value for the data input.

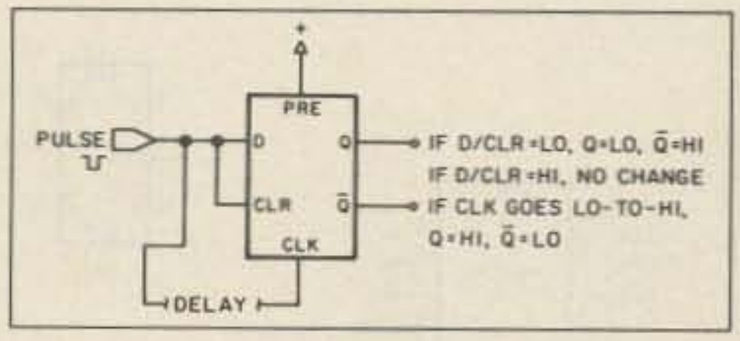

Figure 16. A delayed pulse controller.

Figure 15 operates by similar principles, but in reverse order. Whenever the source has anything to pass on, it sends a LO-to-HI clocking transition. This passes on a HI from the D input to the Q output, where it signals a processing chip to accept the data. If the processor cannot accept the data, perhaps because it is full, it sends a LO to the flip-flop clear input, which overrides the clocking and holds the Q output LO. This controller forces a repetition of the data entry, because the flip-flop ignores the clock as long as the processor is not ready for values to process. The user may then employ this circuit to control the input of a storage buffer that follows a keyboard. When the buffer is full or not ready, a keypress will produce nothing.

A variation on these circuits appears in Figure 16. In this circuit, the tester ties the data and clear lines together and feeds them values. When the input value is LO, the clear line forces Q to be LO and −Q to be HI. When the input value goes HI, no change appears at Q and −Q. However, D is now HI. When a clock LO-to-HI transition occurs, Q goes HI and −Q goes LO. This

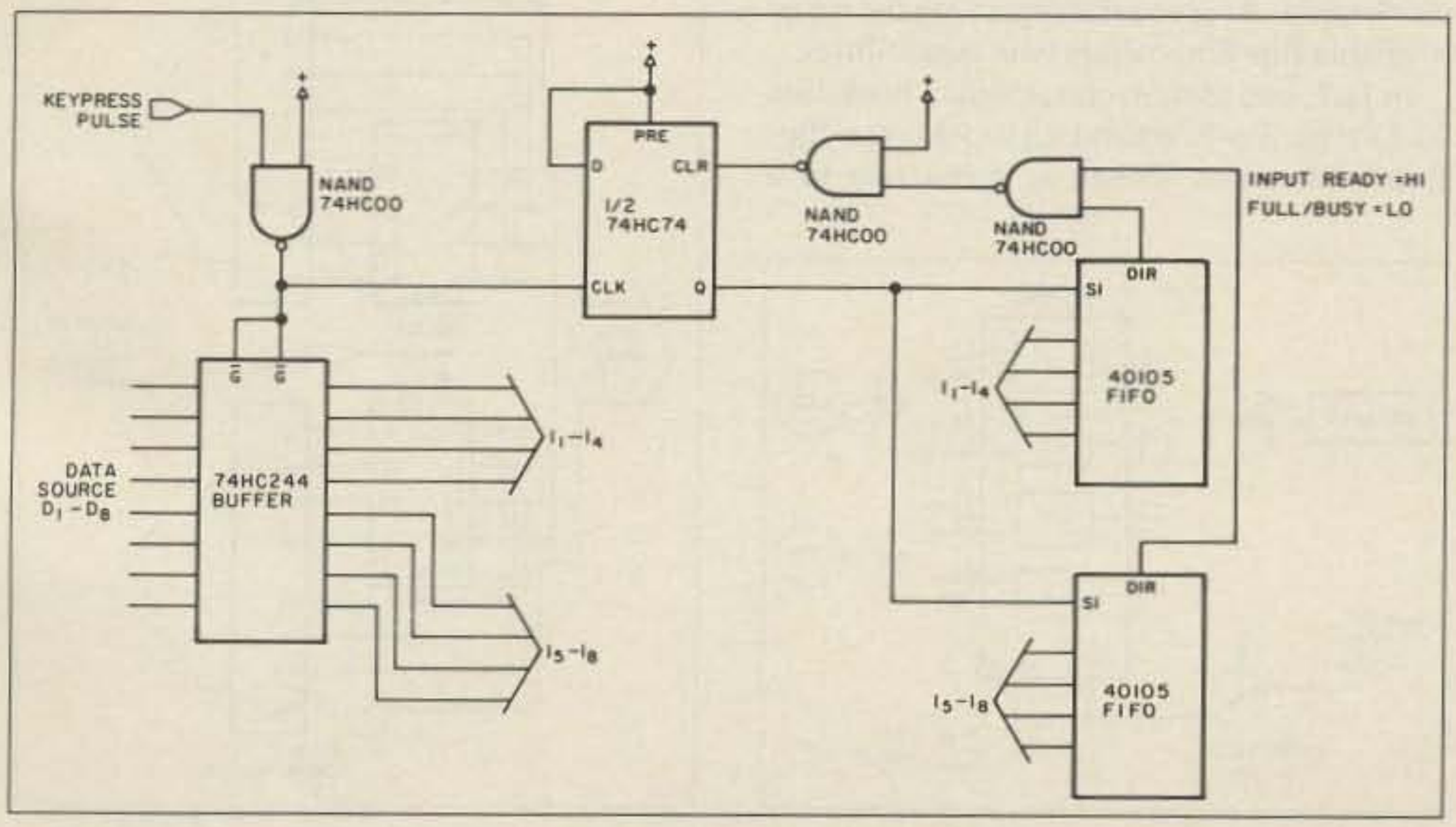

Figure 15. A practical flip-flop controller for a FIFO circuit.

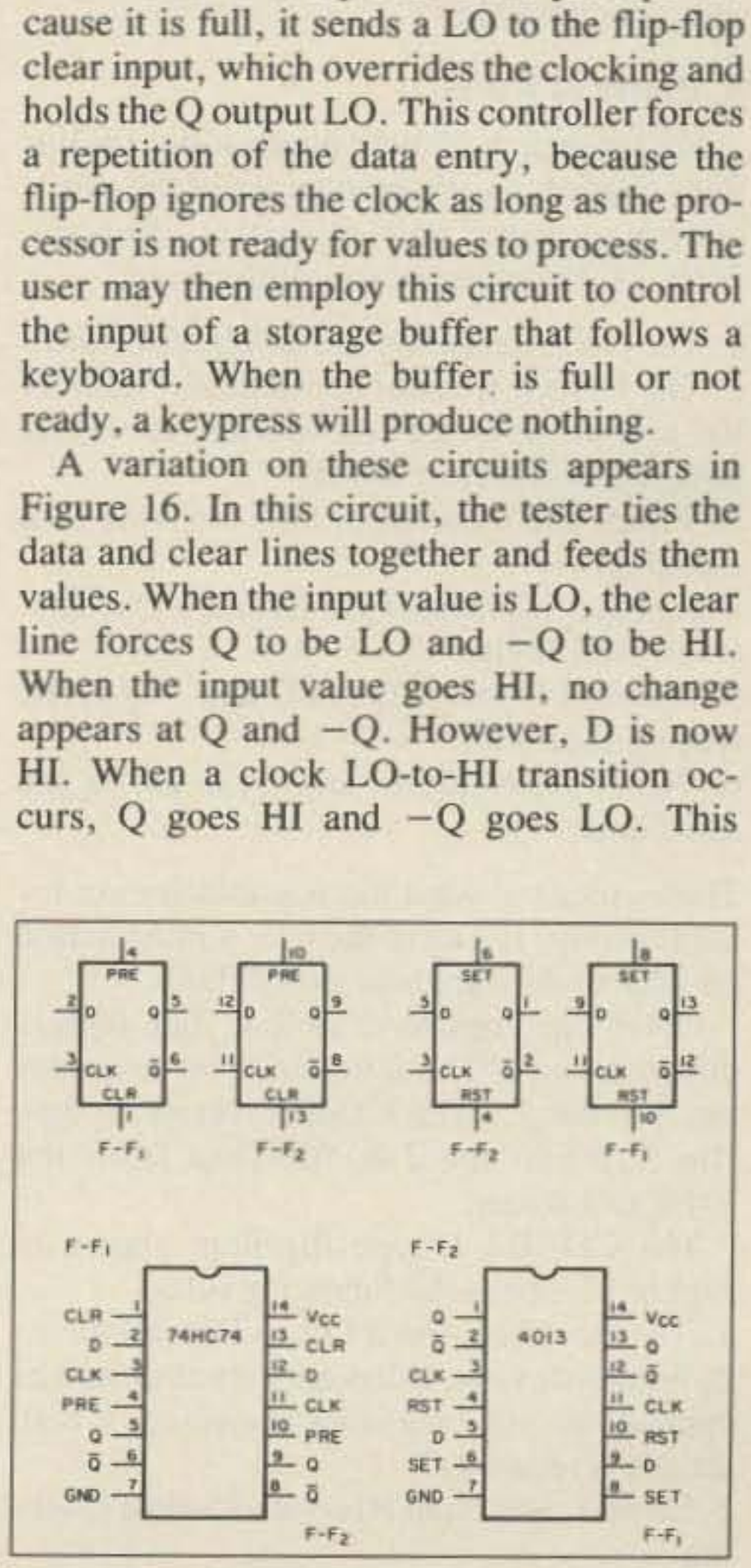

Figure 17. Comparative pinouts for the 74HC74 and the 4013.

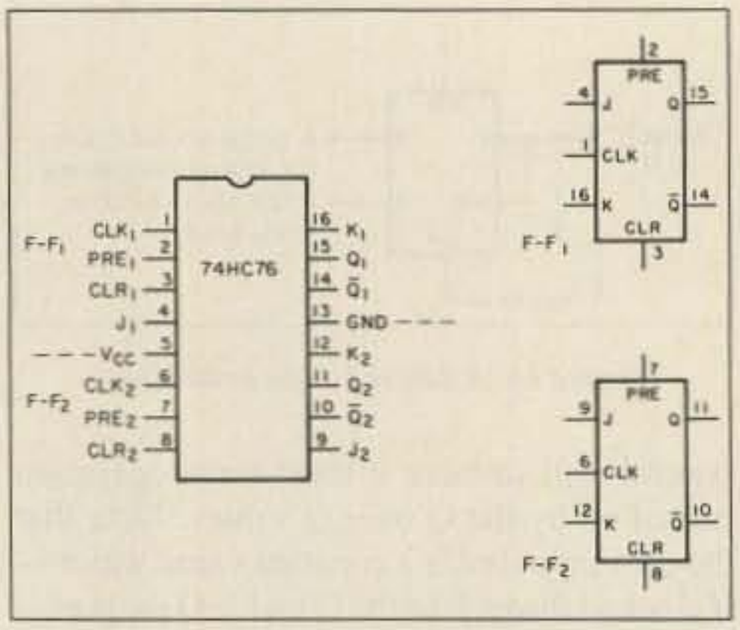

Figure 18. Pinout for the 74HC76 J-K flip-flop.

circuit is useful for controlling operations whenever the user wants to keep the operation going after an input (data and clear) value changes, but he doesn't want it to continue indefinitely. For example, he may feed the data and clear inputs regular pulses that are delayed farther down the line. The clock input may be the delayed pulse. Thus, Q or −Q might enable a certain process from the beginning of the regular pulse through the end of the delayed pulse.

These three circuits add considerable control versatility to the simple debounced switch in Figure 4. In most cases, a user will use signals inside his circuits rather than manual pulsers to control operations. Good practice will let the circuits control themselves wherever possible.

A Matter of Rules

Mastering any of these applications of flip-flops requires only keeping track of the chip's operating rules and priorities. For the 74HC74, the rules are straightforward:

1. The clock requires a LO-to-HI transition.
2. On a clock transition, whatever value D has appears at Q, and its opposite at −Q, if both preset and clear are HI.
3. Preset and clear input LOs override data clocking.
4. A preset LO forces Q to HI and −Q to LO, when clear is HI.
5. A clear LO forces Q to LO and −Q to HI, when preset is HI.
6. Never make preset and clear LO at the same time.

These rules tell what the possibilities are for the flip-flop. It's up to the user's imagination on what to do with these possibilities.

Other flip-flops have similar, but slightly different rules. The following two examples are for the CMOS CD4000-series D-type flip-flop and the J-K flip-flop from the 74HCOO-series.

The CD4013 D-type flip-flop, shown in Figure 17, obeys the following rules:

1. The clock requires a LO-to-HI transition.
2. Whatever value D has appears at Q, and its opposite at −Q, on a clock transition, if both set and reset are LO.
3. Set and reset input HIs override data clocking.
4. A set HI forces Q to HI and −Q to LO, when reset is LO.

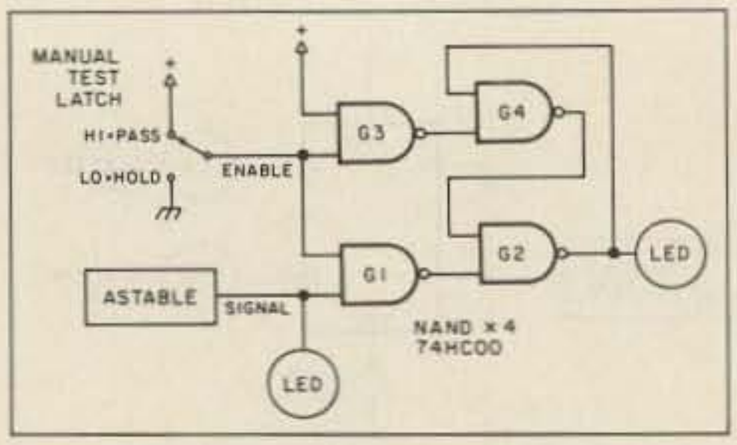

Figure 19. Basic latch circuit using NAND gates.

5. A reset HI forces Q to LO and −Q to HI, when set is LO.
6. Never make set and reset HI at the same time.

Note that the 4013 uses the terms set and reset rather than preset and clear to mark the difference in conditions that actively force values on Q and −Q. Also note that the pinout differs from the 74HC74.

The 74HC76 is a J-K flip-flop, meaning that it has two data input terminals, as shown in Figure 18. Its rules look something like this:

1. The clock requires a HI-to-LO transition.
2. If J is HI and K is LO, then a clock transition will yield Q as HI and −Q as LO, if both preset and clear are HI.
3. If J is LO and K is HI, then a clock transition will yield Q as LO and −Q as HI, if both preset and clear are HI.
4. If J and K are both HI, then clock transition will force Q and −Q to toggle or reverse their values, if both preset and clear are HI.
5. Preset and clear input LOs override J and K data clocking.
6. A preset LO forces Q to HI and −Q to LO, when clear is HI.
7. A clear LO forces Q to LO and −Q to HI, when preset is HI.
8. Never make preset and clear LO at the same time.

The 74HC76 uses a clock transition opposite that of the other two flip-flops featured here. The J and K data inputs offer the possibility of a two-line controlling circuit, as well as the potential for dividing by two by tying both J and K to the positive supply line and feeding the eariler 7555 astable output to the clock input. Every variation among the many available flip-flops offers new possibilities.

In fact, one manufacturer's data book lists 11 D-type flip-flops and eight J-K-type flip-flops. The ones looked at come two to a package, with each flip-flop independent. Some packages, such as the 74HC175 quad D-type and the 74HC78 dual J-K-type, have common clock lines. D-type flip-flops come 4, 6, and 8 to a package, but some have common clear lines and lack the −Q output. How compact the experimenter can make his divider, shift register, or controller may depend on just what input and output lines he wants.

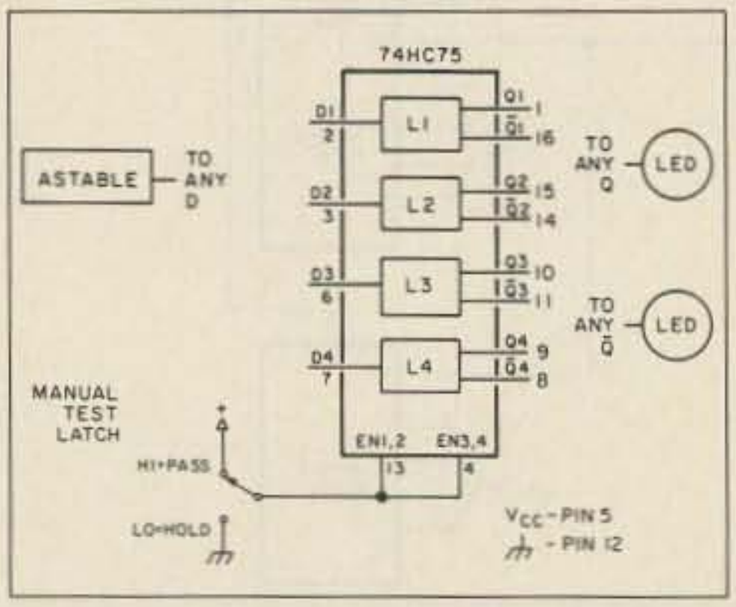

Figure 20. Test latch circuit using the 74HC75.

Latches

The flip-flop is useful because a user can latch into the outputs values that appear only briefly at the inputs. The feedback shown in the basic flip-flop is the key to latching. There are also special ICs called latches. Although associated with flip-flops, they are not true flip-flops. However, latches do use a feedback loop with pass/block gates to freeze the state of a gate whenever the Enable line is set properly. Latches tend to use a slightly different vocabulary from flip-flops, so here are some lessons on how to use latches effectively in circuits.

Every time the user clocks a data value to Q and −Q with a D-type flip-flop, he has latched that value. Without the clock pulse, he block it from passing to the flip-flop output. Latch chips operate in the reverse manner. In an unlatched condition, they pass the input value to the output instantly. Only when placing the correct value on the latch or enable terminal is it possible to hold the output at the value last received. Remember, the value must reach the input before it can be latched.

The experimenter can build a basic latch from a single NAND chip, like the 74HCOO. Figure 19 shows how. Gate 1 looks like the simple pass/block gate used earlier in Figure 3. When the enable line is LO, the gate output is HI, but when the enable line is HI, the signal on its other input line passes (inverted) to gate 2. With enable HI, the inverter (Gate 3) passes a LO to one input of gate 4. That LO forces gate 4 to a HI output, which shows up

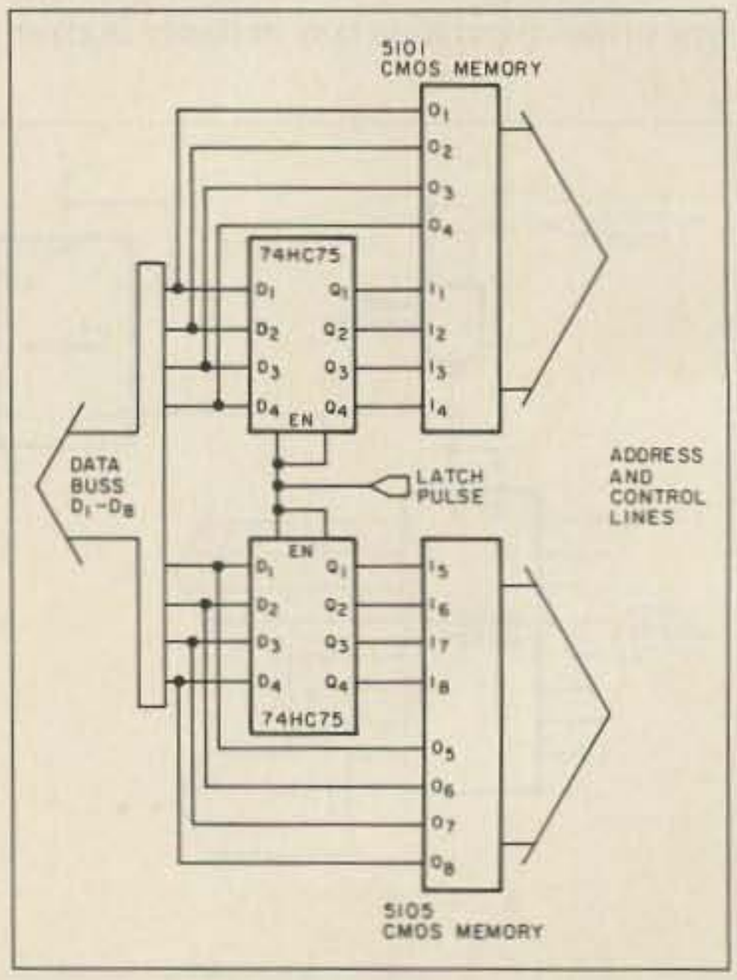

Figure 21. Latch control for memory input signals.

on the other input of gate 2, which in turn allows gate 2 to pass (and re-invert) the signal from gate 1. Under this condition, the output line Q looks just like the data input. When the enable input goes LO, the inverter puts a HI on one input of gate 4, allowing the gate to pass the signal fed back from the output of gate 2. This output will be whatever value gate 2 had when the enable line went LO, since that same LO now blocks signals from passing through gate 1. The gate 1 output stays HI, allowing gate 2 to pass the static signal circulating in the loop from gate 2 to gate 4. The user retains the output value of gate 2 as long as he holds the enable line LO. The condition of the circuit shows itself in the LEDs in Figure 19 as he catches and holds signals with the mechanical switch in the enable line.

He can build most of the circuitry of a latch directly on the IC chip, using only enable and data inputs, along with direct Q (and sometimes inverted −Q) outputs. This allows many latches on a single 14-pin or 16-pin IC. Figure 20 shows a latch circuit using the 74HC75 quad latch. The data input value appears at Q (and its opposite at −Q) so long as the latch input line is HI. When he brings the latch input LO, the Q and −Q outputs remain constant at the level of the last input value, regardless of date input changes. Using the astable pulser as a data source, the test latch uses the same mechanical switch to catch and latch the output before a pulse changes levels.

Latches are useful wherever the user wishes to hold a value temporarily. Figure 21 shows a simplified schematic of a set of latches receiving data in short pulses. However, the latches feed a memory chip that requires a relatively long time to write the data into its cells. (Here, short may mean 10 to 20 nanoseconds, while long means 200 to 300 nanoseconds. They are both short, but the difference is a ratio of ten-to-one.) If he can signal the enable line before the data disappears and hold it LO until the memory has finished writing, the proper data will be present at the memory inputs throughout the write cycle.

"It pays to master the flip-flop for its versatility."

Latches come in many packages for many purposes. Eight-bit (or octal) latches, such as the 74HC373, are useful for capturing computer and other kinds of parallel codes that do not last long. These latches usually have only the Q output. Once latched, a memory chip or other kind of processor can take the time it needs to do its job. Such chips usually have only one or two latch enable inputs to control all the latches simultaneously. When scanning data books in search of the right latch, be sure to read the rules. Most require a LO to latch, but some (such as the 74HC4511 Latch/Decoder/Driver) need a HI or a LO-to-HI transition. The experimenter may find the latch input called Latch, Enable, Control, or G. Newer chips have tri-state outputs for use on buss lines, so be sure to distinguish the latch control from the output enable or control line. One convention calls the output control OE if it requires a HI for output enable and OD if it requires a HI for output disable.

There are many other chips that use flip-flop and related circuitry, usually in conjunction with gates to perform specialized function. I have noted the 74HC4511, which latches a count, transforms it into signals for a 7-segment display, and provides enough drive to light the LED segments. Many data books show representations of internal circuits reduced to the level of gates and flip-flops. If the digital gate is the most fundamental internal IC circuit, the flip-flop is surely second. But remember that at heart a flip-flop is just a special arrangement of gates.

It pays to master the flip-flop for its versatility. And it pays to keep a data book handy when designing digital circuits. At today's prices, manufacturer's data books are a bargain, whether the tinkerer majors in LS-TTL, CMOS, or high speed CMOS. Once having learned how to decipher the rules for a particular flip-flop, the IC offers a large array of useful circuits and an unending source of experiments. 73