Shifting and Shifters

Bob Brown
Computer Science Department
Southern Polytechnic State University

Shifting

We can multiply a decimal number by ten by adding a zero on the right, like this: $13 \times 10 = 130$. Recall that the digits of a decimal number are coefficients of a power series in powers of ten. What has actually happened is that the original digits are shifted left one place in the series, so that the one is shifted from the tens place to the hundreds place and the three is shifted from the units place to the tens place. The added zero occupies the units place. Similarly we can multiply a binary number by two by adding a zero on the right, so $0101 \times 2 = 01010$. Binary digits are coefficients in a series of powers of two, and adding a zero shifts each digit to the next higher place in the series. For any positional number system of base *B*, adding a zero at the right multiplies the number by *B*.

Removing a digit on the right of a decimal number has the effect of dividing by ten; the digit removed is the remainder. Thus, $157 \div 10 = 15$ with remainder 7. The digits are shifted right with respect to the power series. Shifting a binary number to the right divides by two. Shifting any base *B* number to the right divides by *B*. The digit removed is the remainder, which will be between 0 and $B-1$.

When shifting numbers with pencil and paper, it is not necessary to consider what happens at the left end of a number. We just add and remove digits on the right. Computers represent numbers in fixed sizes. Adding a digit at the right necessarily means discarding a digit at the left. Consider an unsigned eight-bit number. We can shift 00000101 left one place and get the expected result: 00001010. A non-significant zero was dropped on the left. However, shifting 10100101 left produces 01001010, not the desired answer at all. As with other arithmetic operations, the rules are different when finite precision arithmetic is involved. If the leftmost digit is significant, shifting left no longer multiplies by *B*.

When shifting right, we must supply a digit on the left. When shifting unsigned or positive numbers, supplying a zero on the left produces the expected result. Such a shift is called a *logical shift.*

A problem arises when shifting signed numbers. Shifting 11110101 right produces 01111010 with remainder one. The result is no longer negative. What is needed is to supply a one, not a zero, on the left if the number is negative. More generally, we want to replicate the sign bit, supplying a zero for positive numbers and a one for negative numbers. Such a shift is called an *arithmetic shift*.

Shifters

A shifter is a combinational circuit with one or more inputs and an equal number of outputs. The outputs are shifted with respect to the inputs. If only a shift left or a shift right is required, no gates are needed; such a shift can be accomplished with wires. Refer to Figure 1. Figure 1 a) shows a shift left. The data inputs are D_0 through D_3 and the shifted outputs are S_0 through S_3. A zero is supplied to S_0, S_1 is connected to D_0, S_2 to D_1, and so on. The leftmost bit, in this example D_3, is discarded. * Similarly, a right shift can be wired as shown in Figure 1 b). A zero is supplied to S_3 on the left, each of the remaining data bits is shifted right one place. The rightmost data bit, D_0, is discarded.

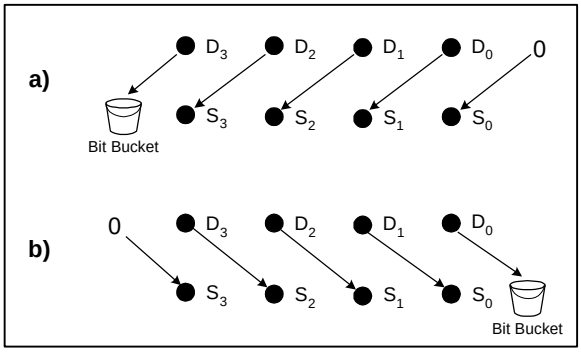

Figure 1. a) Shifting left, and b) shifting right.

If a constant shift left or right is needed, a circuit such as the one above will serve. Usually, more flexibility is needed. Let us examine a four-bit shifter that can shift either left or right, depending on one of two control lines. Examine the AND gates at the extreme left. If the LEFT control line is active, the leftmost AND gate sends its output to the "bit bucket." If the RIGHT control line is active, the other AND gate of the pair sends signal D_3 to output S_2 via the OR gate. Each of the remaining pairs of AND gates can send their outputs either left or right, depending on which control line is active. The leftmost and rightmost AND gates are not actually needed, but are shown for emphasis and because we will use them in the next circuit.

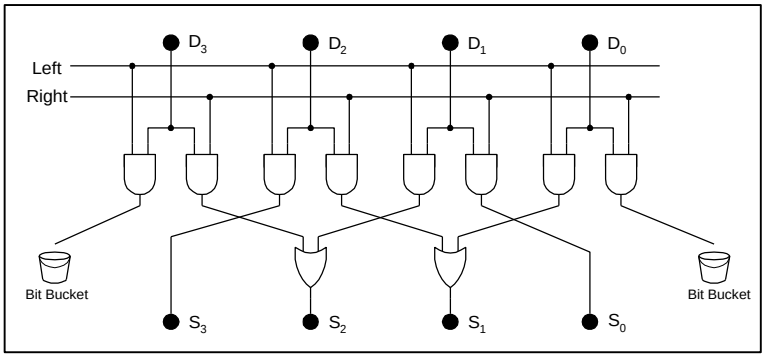

Figure 2. This circuit can shift the input bits left or right, depending on which control line is active.

If you examine the inner pairs of AND gates, you will see that a shifter like the one in Figure 2 can be constructed in any width. All that is required is to add two AND gates and an OR gate for each additional bit. Shifters can be constructed in "slices" that can be wired together. A 32-bit shifter can be built of four eight-bit shifter "slices".

* "Bit bucket" is computer jargon for the fictitious place where discarded data is sent.

The circuit of Figure 2 discards any signal shifted out. However, the bit shifted out is of interest. In a right shift, it is the remainder after division by two. In a shift left, it can be tested for significance so that the results of a left shift can be evaluated for validity. The bits shifted out are available from the AND gates at the extreme left and right of the circuit. Figure 3 adds an OR gate to capture the bit shifted out. For the circuit shown, this will be D_3 for a left shift or D_0 for a right shift. If a shifter like this were made a part of the data path of a computer, the signal generated at SHIFT OUT would be saved in a one-bit register and made available for inspection by the programmer. Often the bit shifted out is saved in the CARRY flag.

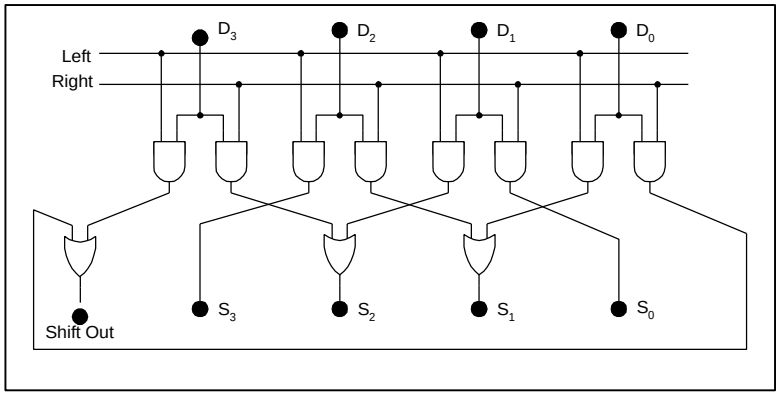

Figure 3. Capturing the bit shifted out so that it can be made available for inspection.

If we design a shifter that can shift left or right, data must be routed through the shifter when shifting is needed and around it at other times. This increases the complexity of the control unit. Another alternative is to design the shifter to have three options: shifting left, shifting right, or not shifting at all. This compromise adds two gate delays when shifting is not needed, but at a saving of complexity elsewhere. Adding a "no shift" option is accomplished with a third control line and a third AND gate for each bit. Figure 4 shows such a circuit. Each input bit now drives three AND gates, only one of which is selected by one of the three control lines. The middle AND gate sends input to output without shifting.

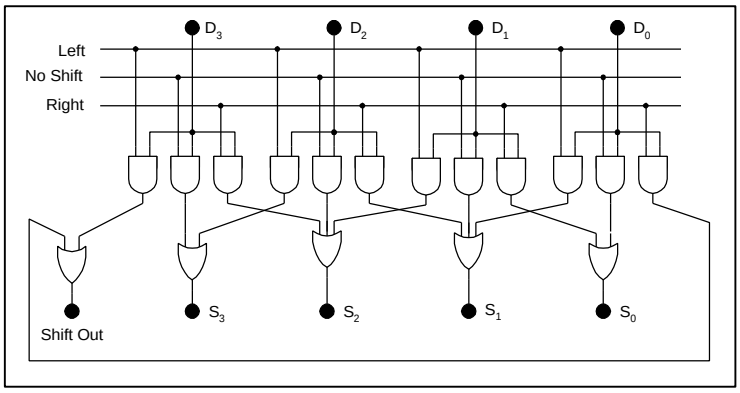

Figure 4. A shifter with a "no shift" option.

Completing our shifter requires attention to one more detail. So far we have no way to deal with the sign bit when performing a right shift on signed data. In other words, we cannot yet do an arithmetic right shift. For a logical right shift, a zero is supplied at the left. Each of the circuits above does this. For an arithmetic shift, the leftmost bit of the input is considered the sign bit. It must be shifted to the right *and also* copied to the leftmost bit of the output. The design for such a shifter is shown in Figure 5. A fourth control line is added for arithmetic right shift. An OR gate

drives the RIGHT control line when either an arithmetic or a logical right shift is commanded. A second OR gate drives the copy function for the leftmost bit when an arithmetic right shift is commanded.

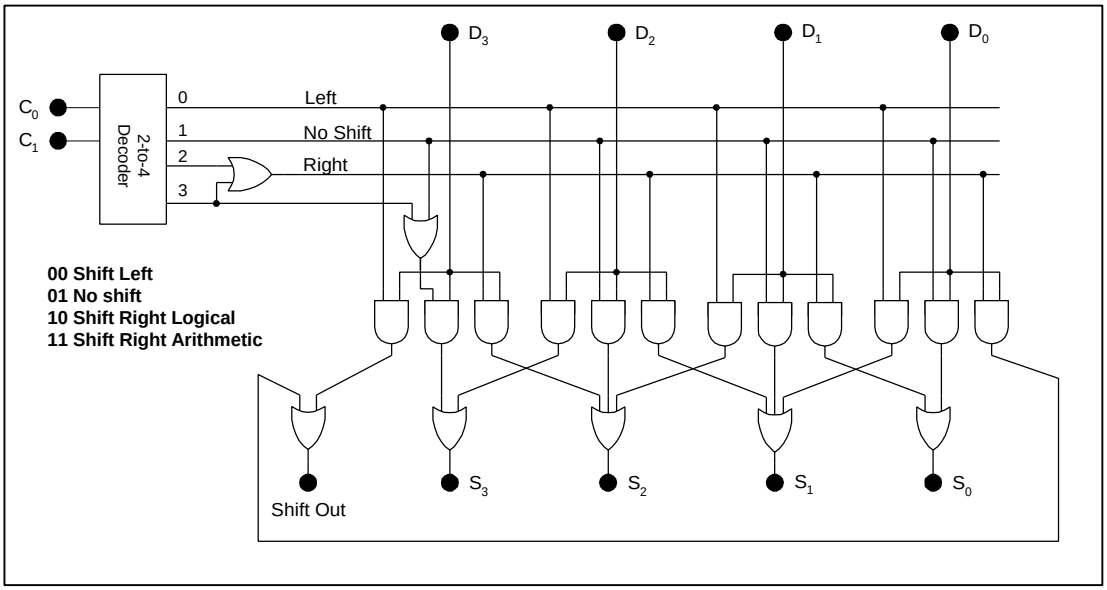

Figure 5. Final design of a shifter. This circuit can perform left shifts, logical and arithmetic right shifts, or no shift.

A decoder is used to generate the control lines. This means that the control unit has to generate only two bits to specify shifting and guarantees that one and only one set of control lines within the shifter is active at any time. The codes used for the four types of shifts are arbitrary. The codes used above were selected to minimize wire crossings in the drawing.