low, each bit of the input (labeled I_0 through I_3) is shifted left by one position into the outputs (labeled O_0 through O_3). When the control line is high, a right shift occurs. This shifter can easily be expanded to any number of bits, or combined with memory elements to create a shift register.

There are far too many combinational circuits for us to be able to cover them all in this brief chapter. The references at the end of this chapter provide much more information on combinational circuits than we can give here. However, before we finish the topic of combinational logic, there is one more combinational circuit we need to introduce. We have covered all of the components necessary to build an **arithmetic logic unit (ALU)**.

Figure 3.17 illustrates a very simple ALU with four basic operations—AND, OR, NOT, and addition—carried out on two machine words of 2 bits each. The control lines, f_0 and f_1, determine which operation is to be performed

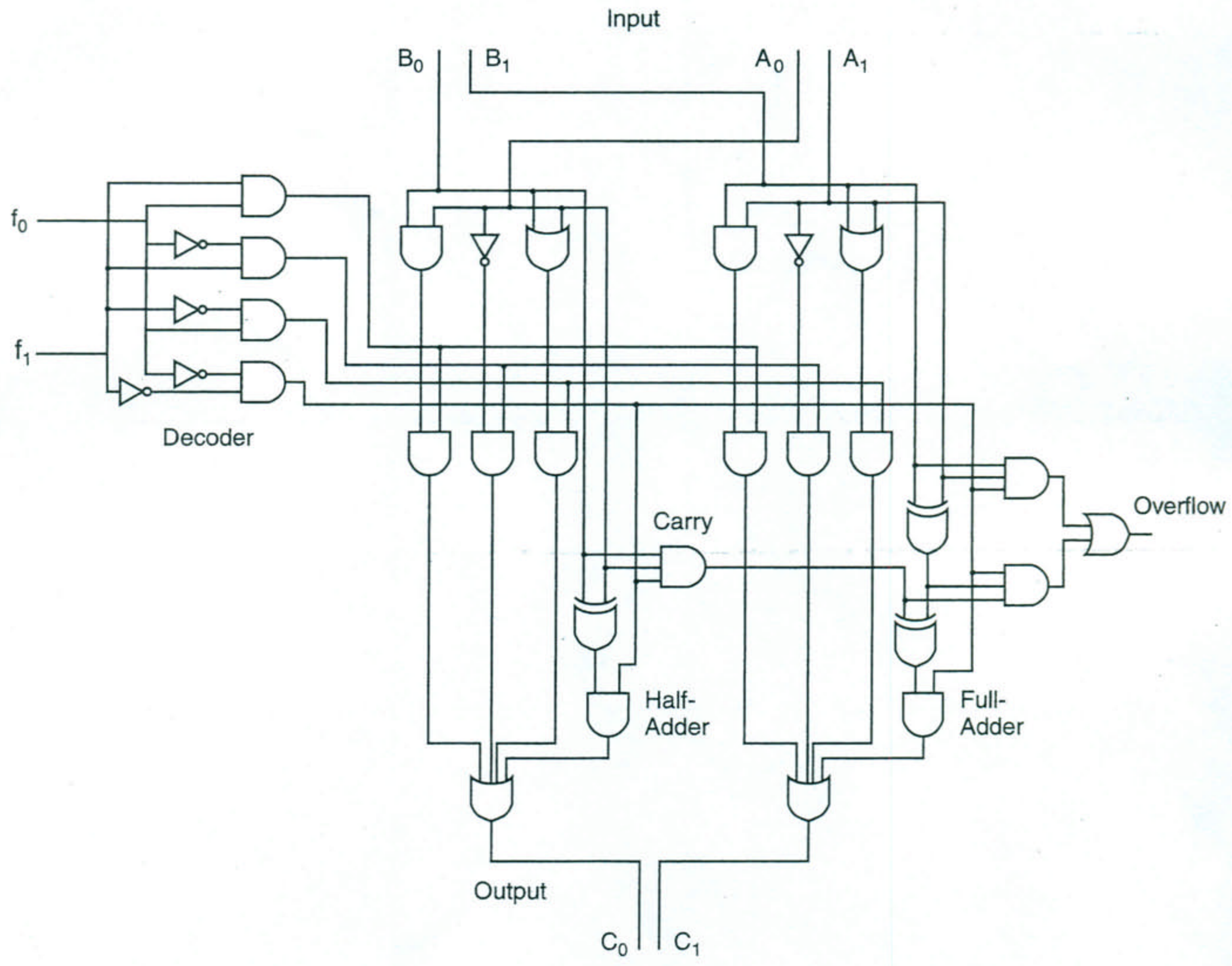


FIGURE 3.17 A Simple Two-Bit ALU

by the CPU. The signal 00 is used for addition (A + B); 01 for NOT A; 10 for A OR B, and 11 for A AND B. The input lines A_0 and A_1 indicate 2 bits of one word, and B_0 and B_1 indicate the second word. C_0 and C_1 represent the output lines.

3.6 SEQUENTIAL CIRCUITS

In the previous section we studied combinational logic. We have approached our study of Boolean functions by examining the variables, the values for those variables, and the function outputs that depend solely on the values of the inputs to the functions. If we change an input value, this has a direct and immediate impact on the value of the output. The major weakness of combinational circuits is that there is no concept of storage—they are memoryless. This presents us with a dilemma. We know that computers must have a way to remember values. Consider a much simpler digital circuit needed for a soda machine. When you put money into a soda machine, the machine remembers how much you have put in at any given instant. Without this ability to remember, it would be very difficult to use. A soda machine cannot be built using only combinational circuits. To understand how a soda machine works, and ultimately how a computer works, we must study sequential logic.

3.6.1 Basic Concepts

A sequential circuit defines its output as a function of both its current inputs and its previous inputs. Therefore, the output depends on past inputs. To remember previous inputs, sequential circuits must have some sort of storage element. We typically refer to this storage element as a **flip-flop**. The state of this flip-flop is a function of the previous inputs to the circuit. Therefore, pending output depends on both the current inputs and the current state of the circuit. In the same way that combinational circuits are generalizations of gates, sequential circuits are generalizations of flip-flops.

3.6.2 Clocks

Before we discuss sequential logic, we must first introduce a way to order events. (The fact that a sequential circuit uses past inputs to determine present outputs indicates we must have event ordering.) Some sequential circuits are **asynchronous**, which means they become active the moment any input value changes. **Synchronous** sequential circuits use clocks to order events. A **clock** is a circuit that emits a series of pulses with a precise pulse width and a precise interval between consecutive pulses. This interval is called the **clock cycle time**. Clock speed is generally measured in megahertz or gigahertz.

A clock is used by a sequential circuit to decide when to update the state of the circuit (when do "present" inputs become "past" inputs?). This means that inputs to the circuit can only affect the storage element at given, discrete instances of

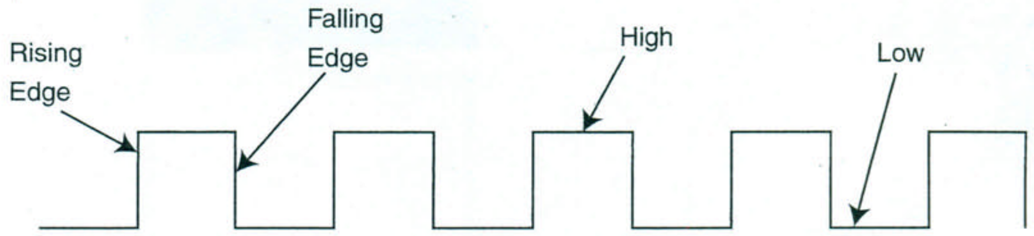


FIGURE 3.18 A Clock Signal Indicating Discrete Instances of Time

time. In this chapter, we examine synchronous sequential circuits because they are easier to understand than their asynchronous counterparts. From this point, when we refer to "sequential circuit," we are implying "synchronous sequential circuit."

Most sequential circuits are edge triggered (as opposed to being level triggered). This means they are allowed to change their states on either the rising or falling edge of the clock signal, as seen in Figure 3.18.

3.6.3 Flip-Flops

A level-triggered circuit is allowed to change state whenever the clock signal is either high or low. Many people use the terms *latch* and *flip-flop* interchangeably. Technically, a latch is level triggered, whereas a flip-flop is edge triggered. In this book, we use the term **flip-flop**.

To "remember" a past state, sequential circuits rely on a concept called **feedback**. This simply means the output of a circuit is fed back as an input to the same circuit. A very simple feedback circuit uses two NOT gates, as shown in Figure 3.19.

In this figure, if Q is 0, it will always be 0. If Q is 1, it will always be 1. This is not a very interesting or useful circuit, but it allows you to see how feedback works.

A more useful feedback circuit is composed of two NOR gates resulting in the most basic memory unit called an **SR flip-flop**. SR stands for "set/reset." The logic diagram for the SR flip-flop is given in Figure 3.20.

We can describe any flip-flop by using a **characteristic table**, which indicates what the next state should be based on the inputs and the current state, Q. The notation Q(t) represents the current state, and $Q(t + 1)$ indicates the next state, or the state the flip-flop should enter after the clock has been pulsed. Figure 3.21 shows the actual implementation of the SR sequential circuit and its characteristic table.

An SR flip-flop exhibits interesting behavior. There are three inputs: S, R, and the current output Q(t). We create the truth table shown in Table 3.12 to illustrate how this circuit works.

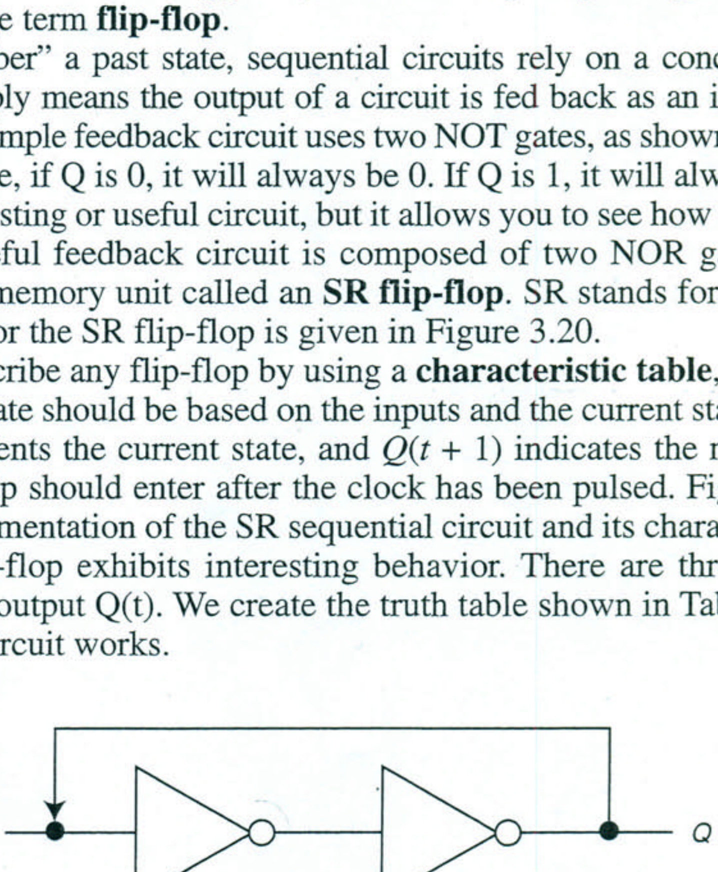


FIGURE 3.19 Example of Simple Feedback

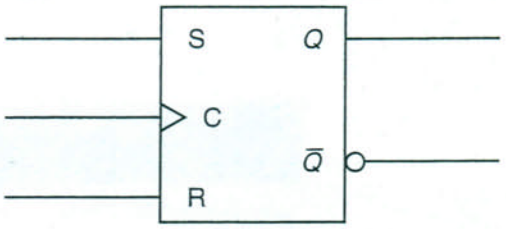
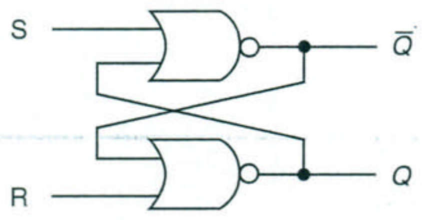


FIGURE 3.20 SR Flip-Flop Logic Diagram



(a)

S	R	Q(t+1)
0	0	Q(t) (no change)
0	1	0 (reset to 0)
1	0	1 (set to 1)
1	1	undefined

(b)

FIGURE 3.21 a) Actual SR Flip-Flop
b) Characteristic Table for the SR Flip-Flop

S	R	Present State Q(t)	Next State Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	undefined
1	1	1	undefined

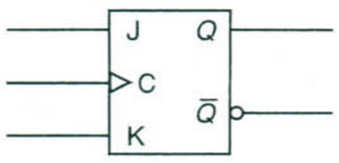
TABLE 3.12 Truth Table for SR Flip-Flop

For example, if S is 0 and R is 0, and the current state, Q(t), is 0, then the next state, Q(t + 1), is also 0. If S is 0 and R is 0, and Q(t) is 1, then Q(t+1) is 1. Actual inputs of (0,0) for (S,R) result in no change when the clock is pulsed. Following a similar argument, we can see that inputs (S,R) = (0,1) force the next state, Q(t + 1), to 0 regardless of the current state (thus forcing a **reset** on the circuit output). When (S,R) = (1,0), the circuit output is **set** to 1.

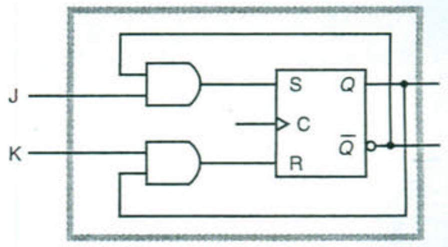
There is one oddity with this particular flip-flop. What happens if both S and R are set to 1 at the same time? This forces both Q and $\overline{Q}$ to 0, but how can $Q = 0 = \overline{Q}$? This results in an unstable circuit. Therefore, this combination of inputs is not allowed in an SR flip-flop.

We can add some conditioning logic to our SR flip-flop to ensure that the illegal state never arises—we simply modify the SR flip-flop as shown in Figure 3.22. This results in a **JK flip-flop**. JK flip-flops were named after the Texas Instruments engineer, Jack Kilby, who invented the integrated circuit in 1958.

Another variant of the SR flip-flop is the **D** *(***data***)* **flip-flop**. A D flip-flop is a true representation of physical computer memory. This sequential circuit stores one bit of information. If a 1 is asserted on the input line D, and the clock is pulsed, the output line Q becomes a 1. If a 0 is asserted on the input line and the clock is pulsed, the output becomes 0. Remember that output Q represents the current state of the circuit. Therefore, an output value of 1 means the circuit is currently "storing" a value of 1. Figure 3.23 illustrates the D flip-flop, lists its characteristic table, and reveals that the D flip-flop is actually a modified SR flip-flop.

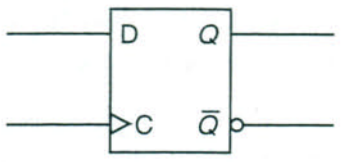


J	K	Q(t +1)
0	0	Q(t) (no change)
0	1	0 (reset to 0)
1	0	1 (set to 1)
1	1	$\overline{Q}(t)$

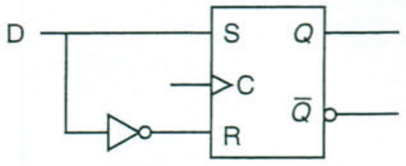


(a) (b) (c)

FIGURE 3.22 a) JK Flip-Flop
b) JK Characteristic Table
c) JK Flip-Flop as a Modified SR Flip-Flop



D	Q(t +1)
0	0
1	1



(a) (b) (c)

FIGURE 3.23 a) D Flip-Flop
b) D Flip-Flop Characteristic Table
c) D Flip-Flop as a Modified SR Flip-Flop

3.6.4 Finite State Machines

Characteristic tables are only one way to describe the behavior of flip-flops and sequential circuits. An equivalent graphical depiction is provided by a **finite state machine** (**FSM**). There are a number of different kinds of finite state machines, each suitable for a different purpose. Figure 3.24 shows a **Moore machine** representation of a JK flip-flop. The circles represent the two states of the flip-flop, which we have labeled *A* and *B*. The output, *Q*, is indicated in brackets, and the arcs illustrate the transitions between the states. This finite state machine is a Moore-type machine because each of the states is associated with the output of the machine. In fact, the reflexive arcs shown in the figure are not required because the output of the machine changes only when the state changes, and the state does not change through a reflexive arc. We can therefore draw a simplified Moore machine (Figure 3.25). Moore machines are named for Edward F. Moore, who invented this type of FSM in 1956.

A contemporary of Edward Moore, George H. Mealy, independently invented another type of FSM that has also been named after its inventor. Like a Moore machine, a **Mealy machine** consists of a circle for each state, and the circles are connected by arcs for each transition. Unlike a Moore machine, which associates an output with each state, a Mealy machine associates an output with each transition. This implies that a Mealy machine's outputs are a function of its current state and its input, and a Moore machine's output is a function only of its current state. Each transition arc is labeled with its input and output separated by a slash.

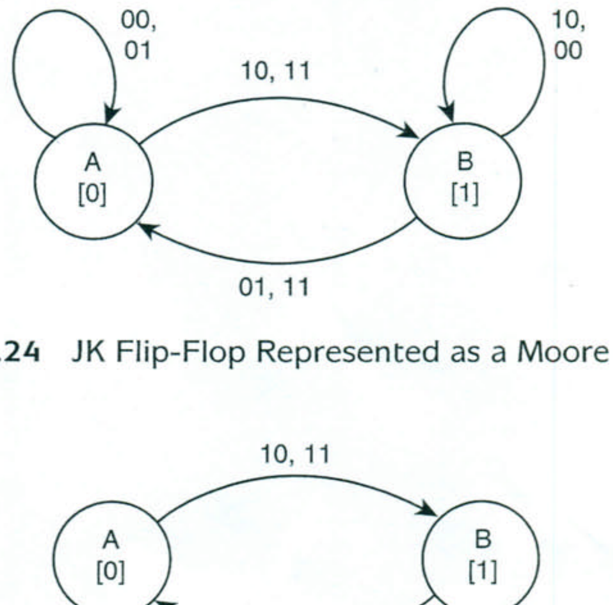


FIGURE 3.24 JK Flip-Flop Represented as a Moore Machine

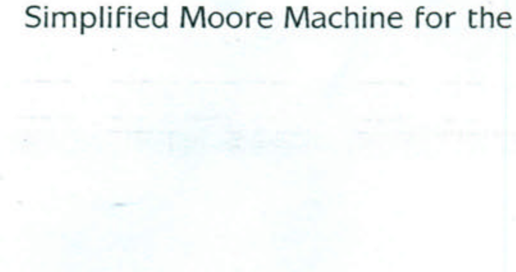


FIGURE 3.25 Simplified Moore Machine for the JK Flip-Flop

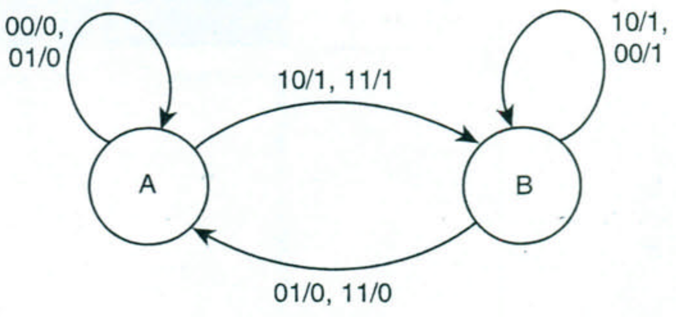


FIGURE 3.26 JK Flip-Flop Represented as a Mealy Machine

Reflexive arcs cannot be removed from Mealy machines because they depict an output of the machine. A Mealy machine for our JK flip-flop is shown in Figure 3.26.

In the actual implementation of either a Moore or Mealy machine, two things are required: a memory (register) to store the current state and combinational logic components that control the output and transitions from one state to another. Figure 3.27 illustrates this idea for both machines.

The graphical models and the block diagrams that we have presented for the Moore and Mealy machines are useful for high-level conceptual modeling of the behavior of circuits. However, once a circuit reaches a certain level of complexity, Moore and Mealy machines become unwieldy and only with great difficulty capture the details required for implementation. Consider, for example, a

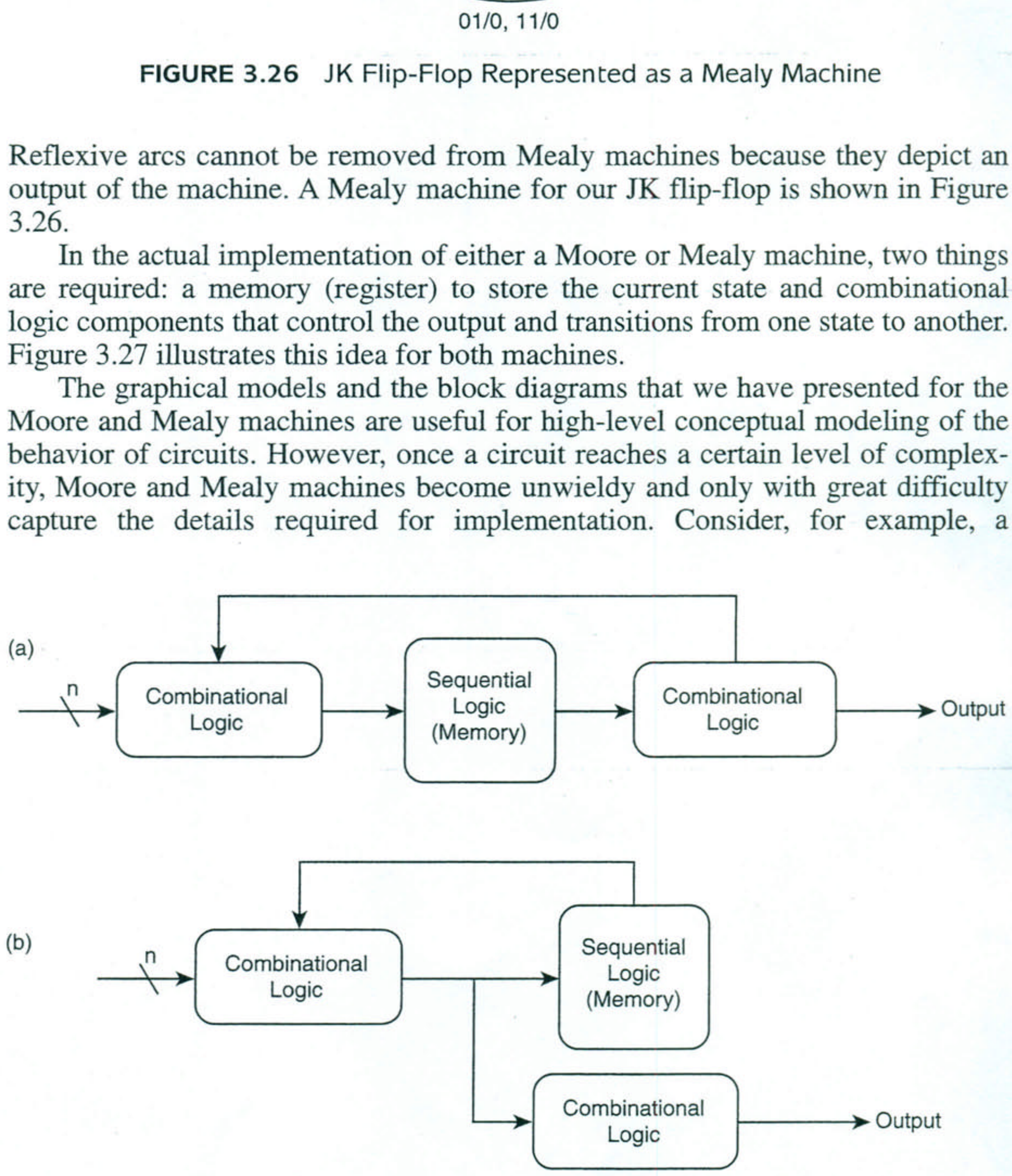


FIGURE 3.27 a) Block Diagram for Moore Machines
b) Block Diagram for Mealy Machines