high and low logic levels of V_h volts and V_l volts, respectively. What should the precharge voltage level V_p be for fastest symmetrical operation of a read?

Solution. To reach a high voltage level, the data line voltage as a function of time is

$$V(t) = V_{DD} - (V_{DD} - V_p)e^{-t/R_uC_L}$$

For the time to reach V_h, solve for t_h as

$$t_h = R_uC_L \ln \frac{V_{DD} - V_p}{V_{DD} - V_h}$$

To reach a low voltage level (ignore the small effect of R_u), the data line voltage as a function of time is

$$V(t) = V_p e^{-t/R_dC_L}$$

For the time to reach V_l, solve for t_l as

$$t_l = R_dC_L \ln \frac{V_p}{V_l}$$

Setting $t_h = t_l$ gives

$$R_u \ln \frac{V_{DD} - V_p}{V_{DD} - V_h} = R_d \ln \frac{V_p}{V_l}$$

Letting the pullup/pulldown ratio be $S = R_u/R_d$ and solving for S gives

$$S = \frac{\ln V_p/V_l}{\ln [(V_{DD} - V_p)/(V_{DD} - V_h)]}$$

as the relationship between S and V_p.

For a ratio-type memory cell, the pullup/pulldown ratio is set higher than the normal value for logic gates to minimize power dissipation in the memory cell. Assuming that $S = 10$, $V_{DD} = 5$ V, $V_h = 4$ V, and $V_l = 0.5$ V, then $V_p = 3.78$ V. In practice, S will be higher than 10 and V_p should be closer to V_h.

The basic cross-coupled SRAM storage cell has several variations. Figure 9.10-4*a* shows a depletion-load cell for an NMOS technology. Many newer static memory circuits use a polysilicon load resistor to form the circuit of Fig. 9.10-4*b*. This requires an additional mask step to provide a lightly doped polysilicon with a resistance of 100k to 1MΩ/□. If a high-resistance polysilicon pullup of minimum size is used, the cell size is reduced by elimination of the depletion transistor and its associated gate-to-source connection.

Another important structure for a static RAM uses CMOS inverters to implement a basic memory cell with extremely low quiescent power characteristics. Were it not for the size disadvantage of the CMOS cell, this cell would be the overwhelming choice for static RAM memories. However, because of the necessity to implement both p- and n-channel transistors and the corresponding n- or p-well spacing requirements, the CMOS memory cell is larger than its depletion- or resistive-load counterpart. Even with this disadvantage,

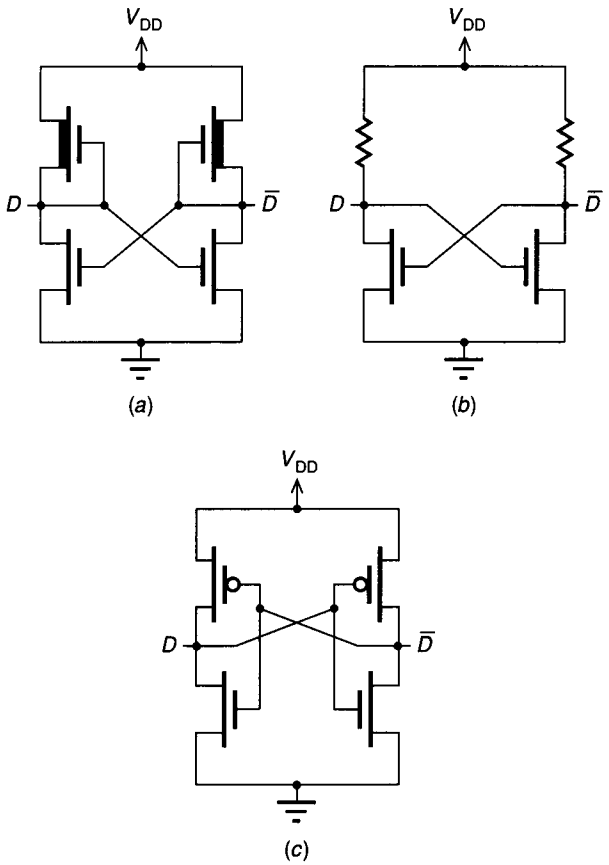

FIGURE 9.10-4
Static RAM cells: *(a)* Depletion load, *(b)* Polysilicon resistor load, *(c)* p-channel load.

many new static RAMs are built with CMOS to reduce power dissipation. The typical CMOS memory cell structure of Fig. 9.10-4*c* can be compared with the similar NMOS cell structure shown in Fig. 9.10-4*a*. The operation of the two cells is identical except that the CMOS cell dissipates negligible power because one of the series transistors from V_{DD} to ground is always turned off. The overall organization of a static memory is the same independent of the type of load device.

Unfortunately, SRAM memory is not as dense as the ROM memory types described earlier because a typical static RAM storage cell requires six transistors. The read-only memory cells of Sec. 9.9 required only one or two transistors per cell. Even when manufacturers replace the depletion pullup transistors with high-resistance polysilicon resistors, the SRAM memory cell still requires four transistors plus the polysilicon resistors. Another type of fast read/write memory, which requires only a single transistor and capacitor for a storage cell, is also available. This memory is described in the next section.

SRAMs are the fastest read/write semiconductor memory in wide use today. A speed advantage over DRAMs offsets the higher density and lower cost per

bit of DRAMs in many applications. For example, SRAMs are widely used in high-speed cache memories for modern computer systems. This section has shown the characteristic cross-coupled inverter structure used for SRAM cells. Additionally, examples showing operation of the highly capacitive row select and data bit lines were considered.

9.11 DYNAMIC RAM MEMORY

The dynamic RAM (DRAM) form of integrated circuit memory has surpassed all other random-access read/write memories in the number of cells or bits that can be placed on a memory chip. A DRAM memory circuit uses charge storage on a capacitor to represent binary data values. A few transistors (first three and now just one) are required to select the cell and access the stored data. Because SRAM memory requires more transistors per memory cell (either four or six, depending on how the pullup for the cross-coupled inverters is implemented), SRAM cannot be manufactured with the high memory density of DRAM. Historically, DRAM chips provide a ratio of about 4 to 1 in the number of memory cells provided relative to SRAM chips for the highest-density memory chips of each type. The DRAM memory array, requiring only enhancement transistors, can be fabricated in either CMOS or NMOS technologies. The peripheral circuitry such as decoders, selectors, sense amps, and output drivers can also be designed for either technology. Most new DRAMs are designed for CMOS processes to minimize power dissipation in the peripheral circuitry.

Dynamic RAM gets its name because the charge stored on the capacitor cell leaks off with time, causing the stored value to be dynamic. If a logic state is represented by a high voltage level on the capacitor cell, this voltage level decreases for a p-well or p-type substrate device because of various leakage paths until the value is indeterminate or changes to the complementary state. Conversely, for an n-well or n-type substrate, the cell voltage increases with leakage. The dynamic nature of this storage mechanism is described more fully in Sec. 9.6. To prevent loss of data, the voltage on the capacitor cell must be sampled and restored within a specified time period. This sample-and-restore operation is called a *memory refresh*; it takes additional external circuitry to ensure that all memory cells are refreshed periodically. A value of 2 ms is a typical specification for the maximum time period between refreshes for DRAM memories.

At one time, most DRAMs were manufactured using a three-transistor cell. This cell, shown in Fig. 9.11-1, is based on charge stored on a capacitor with one transistor acting as a buffer to drive the read data line, one transistor acting as a read-select switch, and a third transistor acting as a write-select switch. All transistors are minimum or near-minimum size to reduce layout area. The three-transistor cell requires four bus lines for operation. These bus lines include separate read and write selects and corresponding read and write data lines like those shown in Fig 9.8-1. Providing a buffer transistor to drive the data line during a read operation prevents degradation of the stored charge during a read operation. However, the charge on the capacitor must still be refreshed periodically because

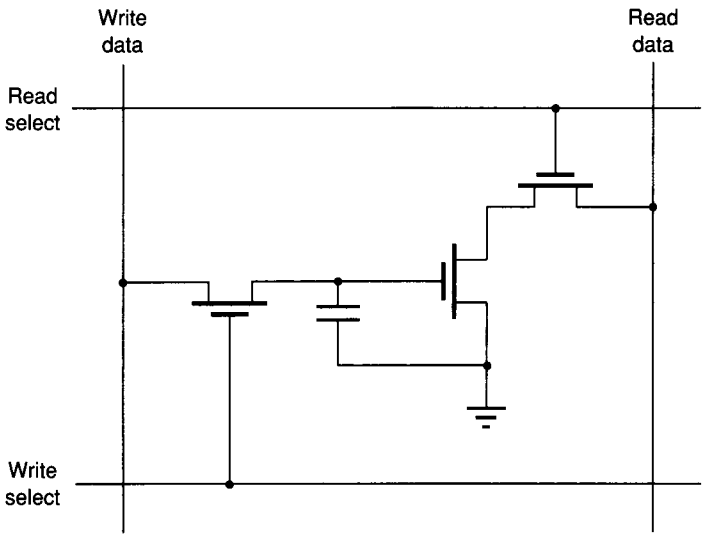

FIGURE 9.11-1
Three-transistor dynamic RAM cell.

of the leakage problems mentioned earlier. The memory refresh is performed by executing a read operation followed by a write operation. The three-transistor cell is robust with respect to the read operation because the stored charge is isolated by a buffer transistor.

Further search for increased memory density brought about the one-transistor DRAM cell. A typical cell with a single select transistor and capacitor for charge storage is shown in Fig. 9.11-2. The single transistor is a pass transistor that serves to connect the stored value to a data bus under control of a select line. The select line simultaneously selects all transistors along the same row, causing data to be placed on the column lines corresponding to each selected cell. Although valid data appear along every column, only one of these columns is further selected for connection to the output on typical DRAMs. These one-transistor cells are formed into a memory architecture as shown in Fig. 9.8-1.

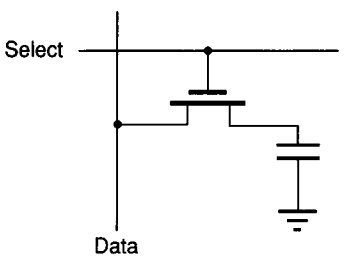

FIGURE 9.11-2
One-transistor dynamic RAM cell.

To keep the size of a dynamic memory cell small, both the select transistor and the storage capacitor must be small. The select transistor is a minimum or near-minimum size device. The small storage capacitor is required to charge the data line through the select transistor during a read operation. Because of the length of the data line, its capacitance is usually large compared with storage cell capacitance. When the select transistor connects the storage cell capacitance and the data line capacitance, the charge is shared to equalize the voltage across the two capacitors that now appear in parallel. Unfortunately, the charge on the larger data line capacitance will have more effect on the final data line voltage than the charge from the small memory cell capacitance. Thus, clever techniques and sensitive circuits are necessary to reliably sense the stored value of a DRAM cell.

One simple technique commonly used to sense the state of a dynamic memory cell involves splitting the data line into two equal halves, thereby splitting the capacitance. Both halves of the data line are precharged to a voltage approximately midway between the high and low logic levels. When a select line goes high, it connects a storage cell capacitor to one of the data line halves; the other half remains unselected. If a comparator circuit is connected with each data line half serving as an input, then even the small change in data line voltage caused by the selected capacitor cell can be detected. The inactive data line half serves as a reference point. This technique requires a comparator for each data line. A typical 256k $\times$ 1 DRAM has 512 data lines. The necessity of providing 512 comparators without using excessive area requires a simple comparator circuit.

Figure 9.11-3 shows a comparator circuit (also known as a sense amp) that has been used to sense the state of DRAM memory cells. This circuit is a flip-flop with special provision to break the cross-coupled links between the two inverters. Before a read operation, the column select, V_{FF}, and sense lines are set low. To execute a read operation, the data lines are precharged to equal voltages (V_{REF}); the desired data cell is gated by a row select to a column line, causing a slight voltage imbalance; the cross-coupled feedback lines of the flip-flop are connected (V_{FF}); and the flip-flop is enabled (Sense). The flip-flop was in a quasi-stable state before the sense line was asserted. The final state of the flip-flop is determined by the slight difference in voltage of the two data column halves caused by the selected memory cell. A later column select signal chooses one of the comparator outputs as the desired data.

Because of the regenerative action of the flip-flop, the data line half will be driven all the way to a high or low voltage, depending on the memory cell contents, and the selected memory cell on each column will be refreshed. That is, if the data cell voltage was higher than the precharged data line value, the flip-flop will switch to drive that half of the data line toward the supply voltage, thereby recharging the selected data cell. Conversely, a low data cell voltage will be discharged toward ground. All memory cells of a DRAM are refreshed by reading a cell on every row because all cells on a row are refreshed when any cell on that row is read. If a memory contains N storage cells and is organized as a square, the complete refresh operation requires a number of reads equal to the square root of N.

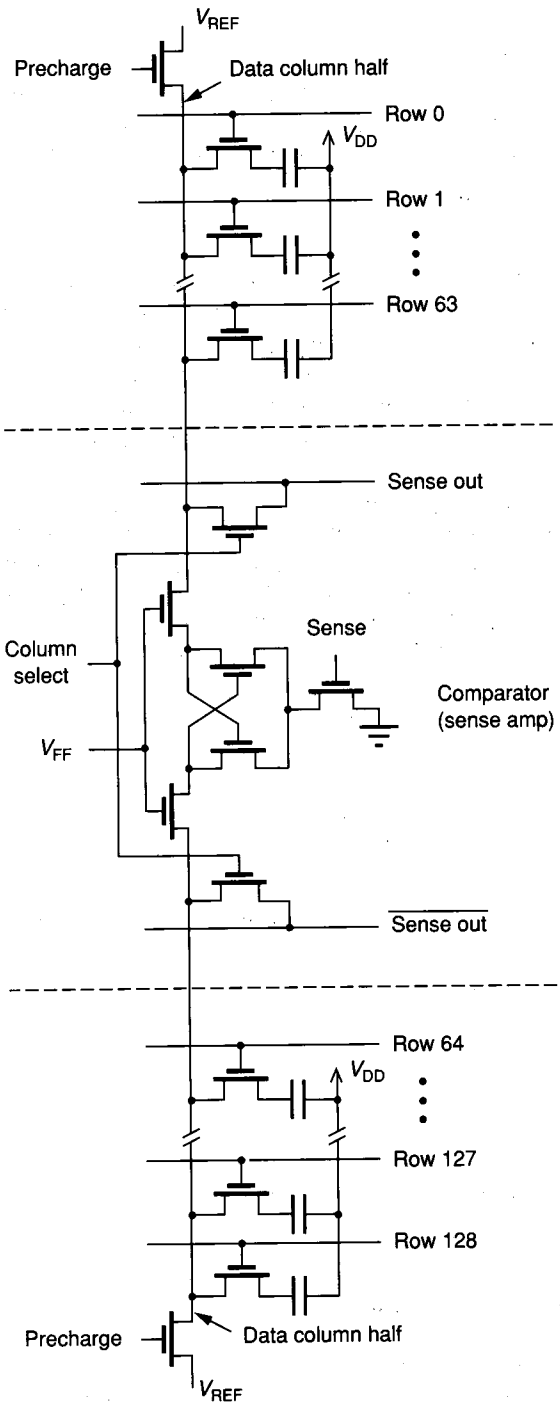

FIGURE 9.11-3
DRAM dynamic sense amplifier.

As DRAMs have gotten larger and storage cells smaller, the ratio between data-line capacitance and memory cell capacitance has increased because of longer data lines and smaller memory cells. To demonstrate how this ratio affects the sensing voltage, consider the following example.

Example 9.11-1. Sensing voltage versus cell capacitance Determine the voltage change on a DRAM data line caused by connection to a memory cell in terms of data line capacitance and memory cell capacitance.

Solution. Let the subscript c refer to the memory cell, and the subscript d refer to the data line. Before the memory cell is selected,

$$V_d = \frac{Q_d}{C_d}$$

and

$$V_c = \frac{Q_c}{C_c}$$

After the cell is selected, the charge is redistributed so that both capacitors are at the same voltage, V_f. Then

$$V_f = \frac{Q_f}{C_f} = \frac{Q_d + Q_c}{C_d + C_c}$$

The change in data-line voltage will be

$$V_d - V_f = (V_d - V_c)\frac{C_c}{C_d + C_c}$$

This analysis shows that the change in data-line voltage that must be sensed is the initial difference between the data-line and memory cell voltages diminished by the ratio of the memory cell capacitance to the total capacitance. Typical capacitance values are C_c = 40 fF and C_d = 1 pF. Thus, an initial 2.5 V difference is divided by 25, resulting in only a 100 mV change in the data-line voltage. It is difficult to detect such a small change, but modern DRAMs are able to do so reliably.

DRAMs have the highest sales volume of memory chips fabricated today. The simple storage cell described in this section leads to high density and low cost per bit. The requirement for refresh of DRAM cell contents is an important consideration in DRAM applications. The small cell/data-line capacitance ratio hinders rapid sensing of memory cell state. Further decrease in the cell/data-line capacitance ratio is an important factor in the design of next generation DRAMs.

9.12 REGISTER STORAGE CIRCUITS

Previous sections described the organization and characteristics of the major types of semiconductor memories. These descriptions were for memories that are usually implemented as stand-alone chips packed with as many memory cells as the current technology allows. Other applications for memory cells are found within sequential machines where the machine state must be stored. Sometimes this temporary storage is accomplished with the shift register described earlier.

Other times, the data must be stored for longer than one clock period. For example, a general-purpose register within a microprocessor typically must hold data while other operations are performed. The following sections describe two types of storage cells that are used within sequential digital systems.

9.12.1 Quasi-Static Register Cells

Figure 9.12-1 shows a way to combine two inverters, two pass transistors, and a nonoverlapping two-phase clock to provide a *quasi-static register cell*. Although the *quasi-static register cell* uses exactly the same components as a two-stage dynamic shift register (see Sec. 9.6), these components are interconnected in a different way. The output of a first inverter is connected directly to the input of a second inverter. One pass transistor, called the *input pass transistor*, controls the input to the first inverter. The second pass transistor, called the *feedback transistor*, controls a feedback path from the output of the second inverter to the input of the first inverter.

The operation of the sample circuit of Fig. 9.12-1 is as follows. When a binary value is to be stored in the register cell, the input pass transister is turned on, and the feedback transistor is turned off. This is accomplished through use of a LOAD signal ANDed with clock phase ϕ_1 to control the gate of the pass transistor. ϕ_2 is low so that the feedback path is broken at this time. When the input pass transistor is turned on, any signal applied to the D input of the register cell is passed to the gate of the first inverter, resulting in the same logic value at the output R of the second inverter (after two successive inversions). When the input pass transistor is turned off, the value at the input node of the first inverter is stored dynamically on the parasitic capacitance of that node. The value at the output of the second inverter is actively driven and is logically equivalent to the stored value at the input of the first inverter. During the ϕ_2 clock phase the output of the second inverter is fed back to the input of the first inverter, thus reinforcing its logic value. As long as this feedback condition is applied often enough, the quasi-static register cell will maintain its stored value.

If the register cell of Fig. 9.12-1 can maintain its stored value indefinitely, why is this circuit connection called a *quasi-static* register cell rather than a *static* register cell? The answer can be found by examining operation of the circuit

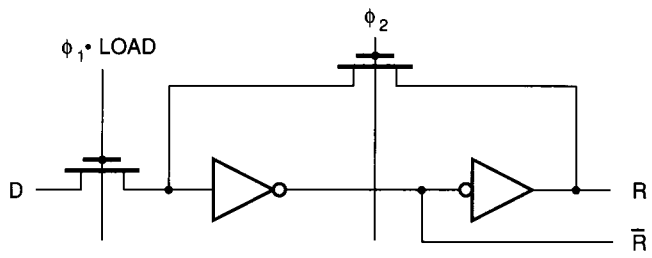

FIGURE 9.12-1
Quasi-static binary storage cell.

over a clock cycle in which a LOAD signal does not occur. Of course, while the ϕ_2 clock phase that controls the feedback transistor is high, the stored value is continuously reinforced. However, when the ϕ_1 clock phase that controls the input pass transistor is high but the LOAD signal is low, there is no active input to drive the gate of the first inverter. If this condition persists for too long a period, the logic value at this input gate may change because of charge leakage. Thus, there is a maximum time for the ϕ_2 clock connected to the feedback transistor to remain in the low state and still ensure the integrity of the data value stored in the cell. This condition places a lower bound on the clock frequency when quasi-static registers are used.

Quasi-static register cells were common in early microprocessors. For example, registers in the Motorola 6800 series of microprocessors were composed of an extension of the basic quasi-static cell that permitted dual-port read and write.[17] This cell, shown in Fig. 9.12-2, provides two gated load (write) signals on one clock phase, so the register can be loaded from either of two buses. A feedback path to refresh the stored logic value is provided on the alternate clock phase. The controller (not shown) that generates the write signals should logically AND them with ϕ_1 to avoid conflict with the feedback path that is controlled by ϕ_2 in Fig. 9.12-2. The register output, taken from the center of the register cell, drives a pulldown transistor. The output of this transistor is directed through pass transistors to one of two possible buses providing dual-port read. This cell requires four control signals (each externally gated by clock phase ϕ_1), an alternate clock signal ϕ_2 to control the feedback path, two bus lines (each bus line is common to one input and one output path), power, and ground. This cell, requiring a total of 10 transistors, will be compared with the static register cell described next.

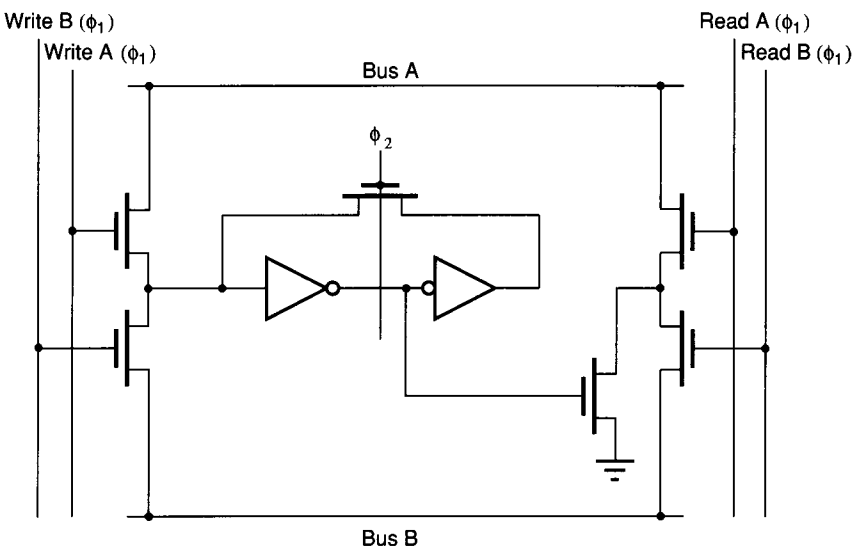

FIGURE 9.12-2
Motorola 6800 microprocessor register cell.

9.12.2 A Static Register Cell

Fully static register cells are frequently used within finite-state machines and within microprocessor register arrays. These static register cells are similar to the memory cell described previously for SRAMs, but often are designed with different constraints than those necessary for dedicated memory chips. One such static cell is based on the classical cross-coupled set-reset (SR) latch shown in Fig. 9.12-3*a*. This latch uses two cross-coupled NOR gates to achieve data storage. An equivalent NMOS transistor-level circuit for this latch is given in Fig. 9.12-3*b*. That this is a static register cell is obvious because the storage does not depend on clock signals, but only on a directly coupled feedback path.

To explain static register cell operation, the SR latch circuit of Fig. 9.12-3*b* will be transformed into a static register cell in two steps. Figure 9.12-4*a* shows the previous circuit split into a cross-coupled inverter pair with the set and reset pulldown transistors physically separated from the storage element by bus lines. These buses hold signals representing the register cell's logic state and its complement. Figure 9.12-4*b* completes the transformation by including pass transistors between the outputs of the cross-coupled inverter pair and the buses to the set and reset pulldown transistors. The pass transistors provide a way to isolate the register cell from the buses. Note that if both pass transistors are on, the circuit is equivalent to that of Fig. 9.12-3*b*, except for additional resistance in the set and reset pulldown paths because of the pass transistors. This basic static register cell consists of six transistors, four for the cross-coupled inverters and two for the connections to the buses. A CMOS version of this cell is created by replacing the NMOS inverters of Fig. 9.12-4 with CMOS inverters.

Because the basic register cell of Fig. 9.12-4*b* can be isolated from the buses, additional six-transistor register cells can be attached between the same

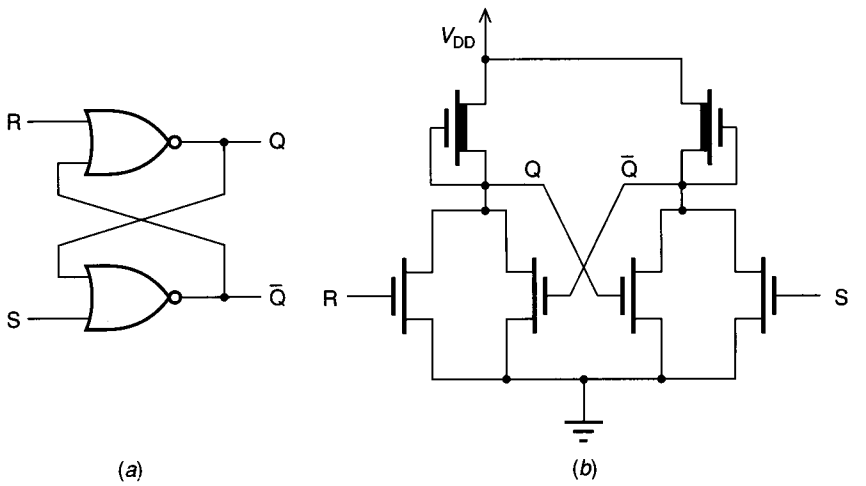

FIGURE 9.12-3
Cross-coupled NOR latch: *(a)* logic, *(b)* circuit.

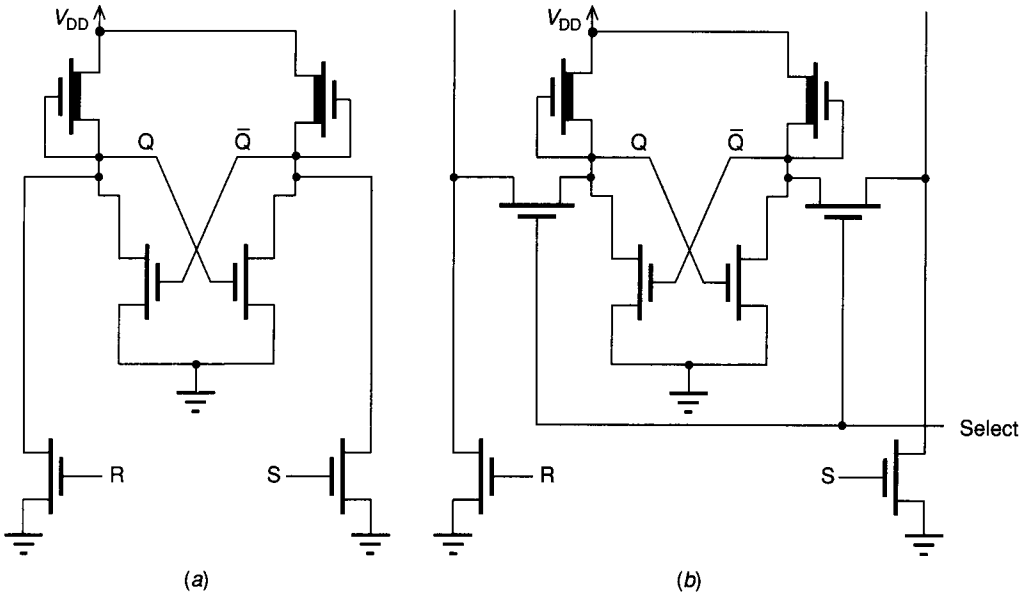

FIGURE 9.12-4
NMOS static storage cell.

two buses. Then a particular register cell is selected for read or write by selecting (turning on) both pass transistors associated with that cell. Figure 9.12-5 shows four CMOS static register cells that use the same two buses for read or write of cell data.

This static register cell is similar to those used in SRAMs. Many applications, however, do not require the considerable address decoding circuitry and sensitive read sense amplifiers necessary for large SRAM memory chips. There are two reasons for this. The first is size. A typical microprocessor application might require a register array with 1024 bits compared to commercial SRAMs with 256k bits. The smaller size reduces capacitive loading and diminishes noise sources, allowing simplified supporting circuitry. The second factor is organization. As explained in Sec. 9.8, a square organization requiring both row decoding and column selection to access a single bit is preferred for SRAMs. A typical 1024-bit microprocessor application might have 32 registers, each containing 32 bits. Each 32-bit register has its contents accessed as a unit. Thus, only a 5-to-32 address decoder is required to select a 32-bit register. Based on these concepts, then, a data value can be stored simply by selecting a cell and asserting a set or reset line. A stored value can be read by asserting the desired select line and accepting the logic value on the data bus. When this circuit is used within a microprocessor register array, a dual-port read is possible by controlling the two select transistors of a cell individually. Thus, one cell can have its stored value gated to the data bus, while a second cell has its stored value gated to the complement data bus. Many microprocessor instructions require two input operands, making the dual-port structure highly desirable for a register array.

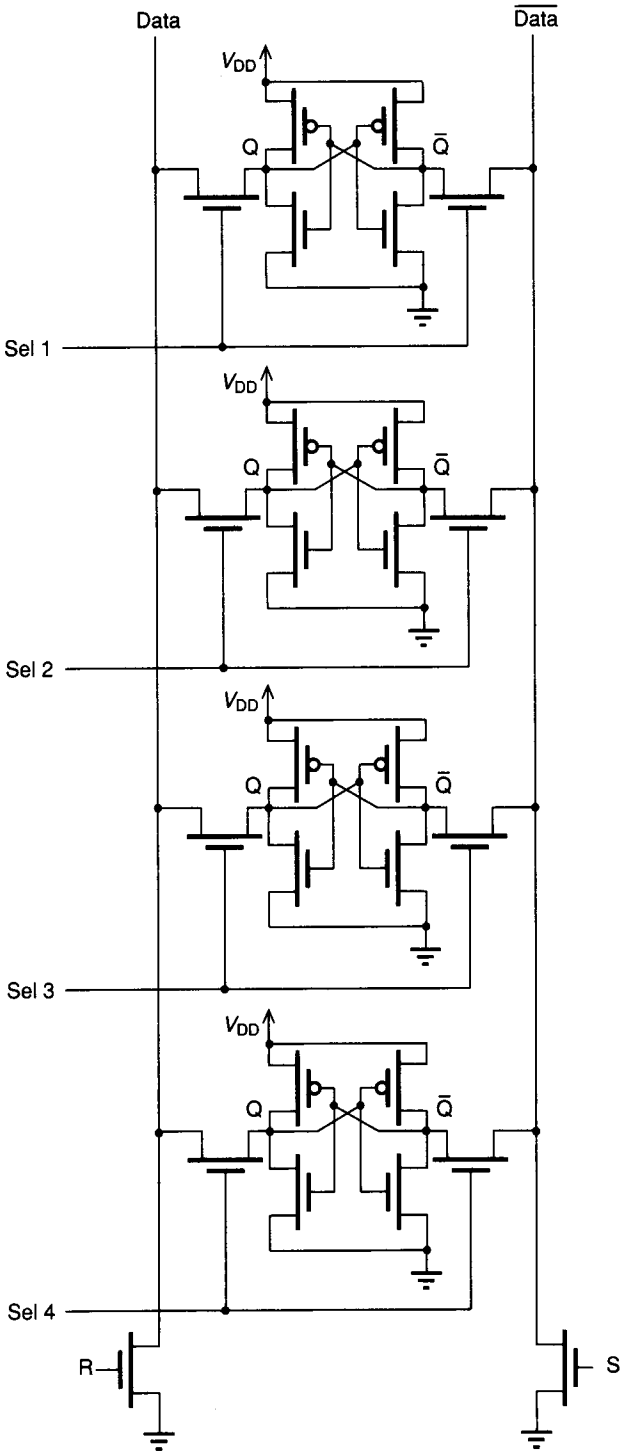

FIGURE 9.12-5
CMOS static register cell array.

Two simple storage cells were described in this section. These are important in digital design for applications that require static storage capability, for example FSMs and microprocessors. Both quasi-static and fully static storage cells were described. Individual storage cells are easily configured into n-bit wide registers where n is set by the width of the data word. The simple design and operation of these circuits make them ideal for many applications.

9.13 PLA-BASED FINITE-STATE MACHINES

Most digital systems are composed of combinational logic and memory combined in a form called a *finite-state machine (FSM)* or, equivalently, a *sequential machine*. A sequential machine is normally implemented as a forward path containing combinational logic and a feedback path that includes memory. In classical digital systems the memory is provided by flip-flops or latches. Within MOS integrated circuits a particularly simple form of sequential machine is possible. This simple finite-state machine consists of a PLA that realizes the combinational logic and a clocked shift register in the feedback path to serve as memory. A dynamic shift register such as the one described in Sec. 9.6 is often used.

Figure 9.13-1 shows the classical form for one type of FSM, called the *Moore machine*.[18] This FSM is characterized by outputs that are isolated from momentary input changes by memory. This type of FSM is of particular interest here because an excellent integrated circuit implementation based on a PLA is available. A block diagram of a PLA-based FSM is shown in Fig. 9.13-2, where a PLA is augmented with pass transistors to gate its inputs and outputs. These pass transistors in combination with the output buffers and double-rail drivers form a clocked shift register so that the next state presented by the PLA OR plane is available as the present state at the inputs to the PLA double-rail drivers after a ϕ_1, ϕ_2 clock sequence.

As discussed in Sec. 9.2, automatic PLA generation programs are available. Based on logic equations in the sum-of-products form, a complete PLA layout can be created and programmed to realize correctly the specified logic functions. It is a simple task to augment a PLA generation program to include clocked input drivers and clocked output buffers in the form shown in Fig. 9.13-2. Such a PLA generator can be used in one of two modes: it can generate a standard

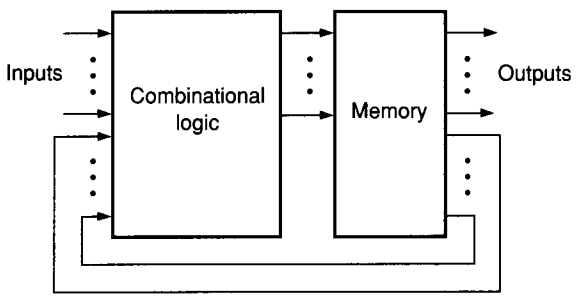

FIGURE 9.13-1
Classical finite-state machine.

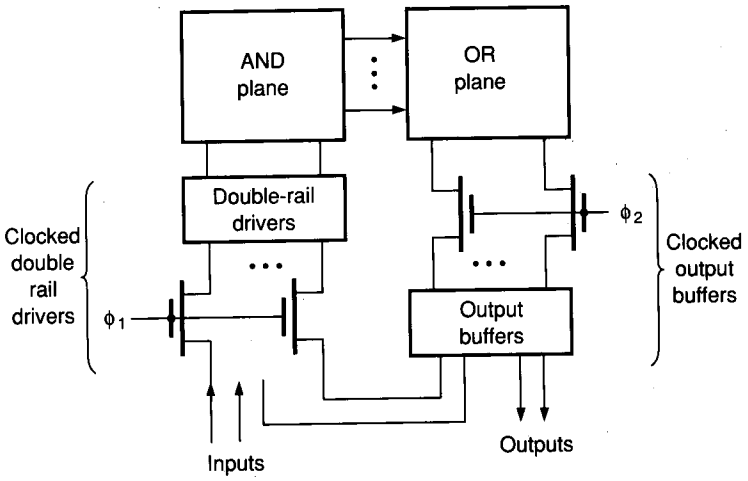

FIGURE 9.13-2
FSM based on PLA with clocked stages.

PLA consisting of combinational logic only, or it can generate a FSM formed from a standard PLA plus clocked shift register feedback created by connecting the output of a clocked output buffer to the input of a clocked double-rail driver.

To demonstrate the design of a FSM based on a PLA with clocked shift register feedback, consider the following example.

Example 9.13-1. Finite-state machine Assume that 16 magnetic switches are to be monitored remotely for a home security application. The 16 status bits corresponding to the state of the switches are available from the remote location through an asynchronous serial data link as alternating bytes of data. The serial data is received in 8-bit groups by a UART (universal asynchronous receiver/transmitter) whose parallel output must be stored in two eight-bit registers that drive LED indicators. Each register is composed of 8 D-type flip-flops activated by a rising clock signal. The first byte of data received is displayed in one set of LED indicators, and the alternate byte is displayed in a second set of LED indicators. Thus, there are 16 LED indicators, one for each magnetic switch. This task can be accomplished with a sequencer (FSM) that alternately selects one of two display registers, A or B, to store the received data bytes. To simplify the design, assume that the system is always synchronized with the first, third, and other odd bytes going to display register A and the even bytes to display register B. The FSM must also generate a data strobe signal (S) required by the UART to acknowledge that a byte is accepted. Of course, the UART generates a data ready signal (R) whenever a new status byte is available at its output. The logic components that compose the receiving system are shown in Fig. 9.13-3. Show the logical design for the PLA FSM block of this figure.

Solution. A PLA FSM that satisfies these requirements is described by the state diagram of Fig. 9.13-4 and the state transition table of Table 9.13-1. The FSM waits in state *a* until the UART indicates that a data byte is ready by asserting the

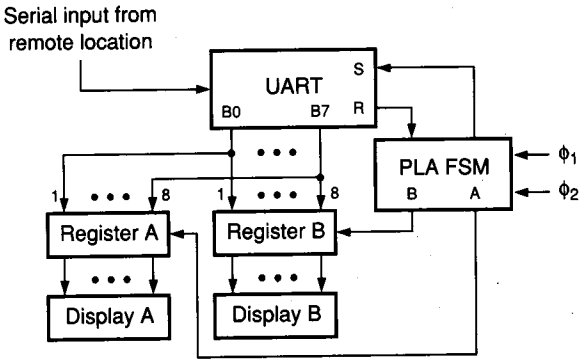

FIGURE 9.13-3
Block diagram for system display.

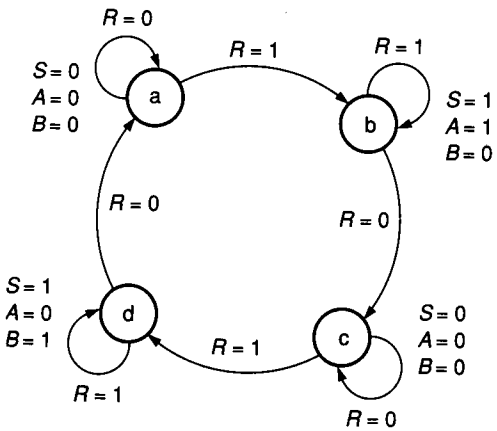

FIGURE 9.13-4
State diagram for status sequencer.

TABLE 9.13-1
State transition table for security monitoring system

Input	Present state	Next state	Outputs		
R	*XY*	*xy*	*S*	*B*	*A*
0	00	00	0	0	0
1	00	01	0	0	0
0	01	11	1	0	1
1	01	01	1	0	1
0	10	00	1	1	0
1	10	10	1	1	0
0	11	11	0	0	0
1	11	10	0	0	0

data ready signal (R). Data ready causes a transfer to state b at the next clock, causing a load signal (A) to display register A and a data strobe signal (S) to the UART. It is important that the display register is loaded by the rising edge of load signal A. Then the FSM waits in state b until the UART removes the data ready signal (R), causing a transfer to state c. At this point, one byte of data has been received and the value in register A updated. When a second byte from the UART is ready, the resulting data ready signal (R) causes a transfer to state d at the next clock, where the appropriate load signal (B) to display register B and data strobe signal (S) are generated. Later, after data ready is removed by the UART, the FSM returns to the first state and waits for new status data.

The state transition table (Table 9.13-1) provides the information necessary to specify the logic operations to be performed by the PLA. Two state variables are required to specify four different states. Call the present state variables X and Y and the corresponding next state variables x and y. The states are encoded with a Gray code (state $a = 00$, state $b = 01$, state $c = 11$, state $d = 10$) so that only one state variable changes for each state transition. The equations for the next state variables and the outputs are obtained from the state transition table and are given here.

$$x = \overline{R}Y + RX$$

$$y = \overline{R}Y + R\overline{X}$$

$$S = \overline{X}Y + X\overline{Y}$$

$$A = \overline{X}Y$$

$$B = X\overline{Y}$$

From these equations, it is easily determined that a (3,5,5) PLA is required—that is, three inputs by five product terms by five outputs. Two outputs form the next state variables, while three other outputs generate the data strobe (S), load register A (A), and load register B (B) signals. Figure 9.13-5 shows a complete PLA-based FSM that implements the controller described here.

The ability to create a FSM automatically from a set of Boolean logic equations is an extremely powerful tool for digital system design. Small PLA-based FSMs are frequently used as building blocks to construct larger digital systems such as microprocessors and communications processors. Large PLA-based FSMs suffer from two important limitations. A large PLA may be sparsely populated with programming sites, resulting in excessive area to realize a function. Also, large PLAs tend to be slower than alternative solutions when a large number of terms must be processed. One alternative is to use several small PLA FSMs rather than one large PLA FSM to implement required control logic; a second alternative is described in the next section.

9.14 MICROCODED CONTROLLERS

The clocked PLA structure for FSMs explained in the previous section is an excellent means to implement small digital controllers. The layout structure is regular and can be generated automatically and compactly from the logic

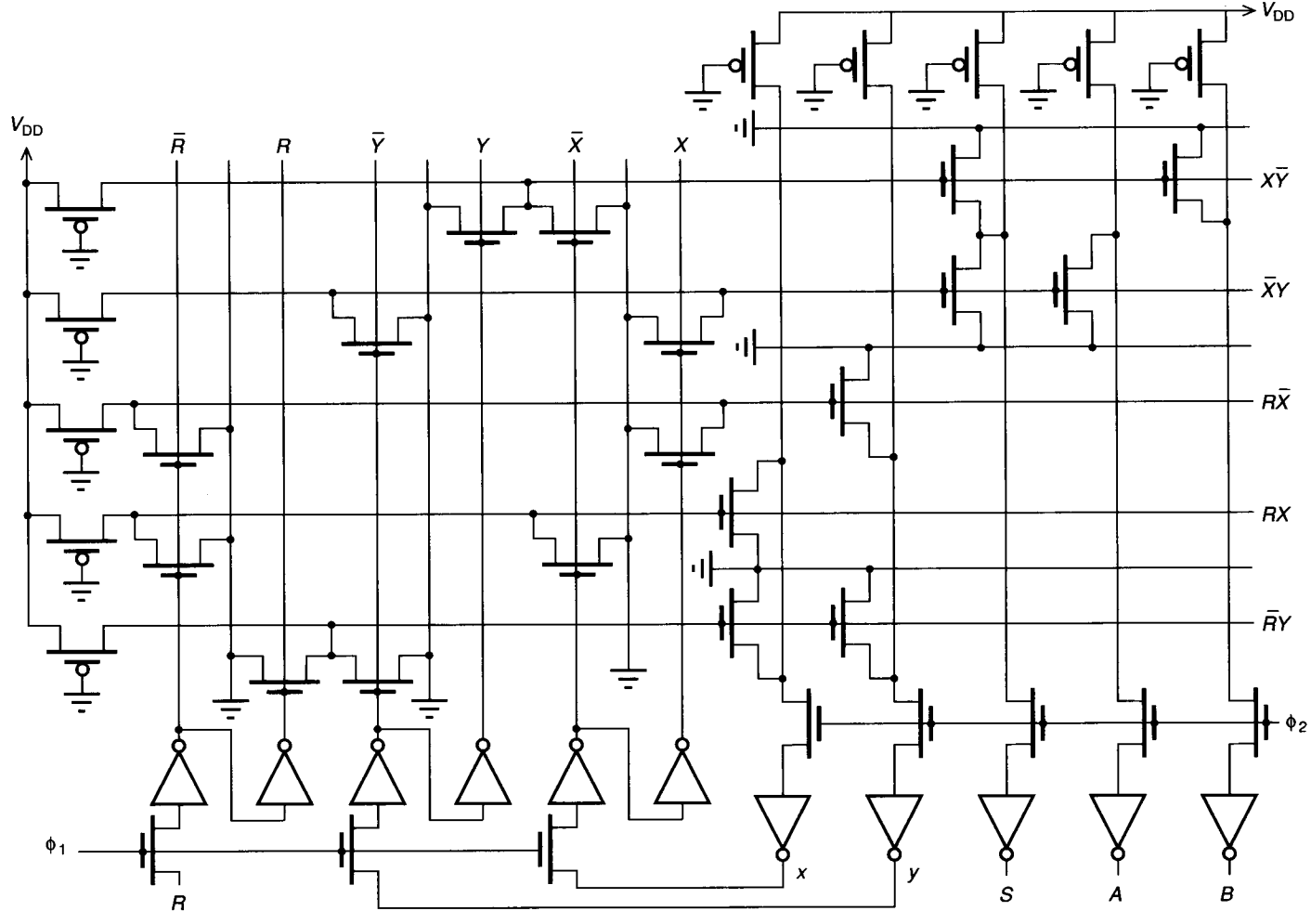

FIGURE 9.13-5
Clocked PLA FSM for sequencer.

equations for a system. For larger digital systems, the logic design to implement a clocked PLA FSM becomes unnecessarily complex and the resulting large PLA, if generated, would be slow. These larger systems require a method that overcomes the disadvantages of a large clocked PLA FSM yet emphasizes regularity in design and layout. A common method to implement complex digital systems in a regular way is to use a memory-based structure known as a *microcoded controller.*

A microcoded controller (shown in Fig. 9.14-1) comprises a memory whose contents are called *microinstructions* and a *next-address sequencer* that directs the execution sequence of microinstructions. A microinstruction is a set of (usually) encoded control bits that direct the operation of the logic during a clock cycle. In essence, a microcoded controller is a special form of computer. The execution hardware is fixed, and the functions performed are a result of instructions placed in the microinstruction memory. This memory is frequently read-only memory and is thus called *microROM.* As discussed earlier, memories are designed with a dense, regular structure. Because the microcoded controller consists primarily of memory, a microcoded controller can also be regular and dense. However, because microcoded controllers require the overhead of a next-address sequencer that requires design time and integrated circuit area, this technique is used primarily for larger machines.

In its simplest form, the microcoded controller of Fig. 9.14-1 does not require status inputs. The next-address sequencer simply generates the next instruction addresses in a fixed pattern, for example, by incrementing a counter. A microcoded controller configured in this way functions as an *open-loop controller* with a fixed execution sequence. If status inputs are provided, the next-address sequencer can modify the address of the next instruction, depending on conditions presented by the status inputs. This provides a *conditional branching capability.* In either case, the function of the microcoded controller is determined primarily by a program placed in its microROM. Conceptually, programming a microcoded controller is similar to programming a microprocessor in machine language. In practice, however, the programming task is extremely tedious because of the multiplicity of individual control bits whose state must be determined for each instruction.

Figure 9.14-2 shows a typical memory organization for a microcoded controller consisting of a microROM and a memory address register (MAR). While most semiconductor memory chips are organized with a wide address bus and a

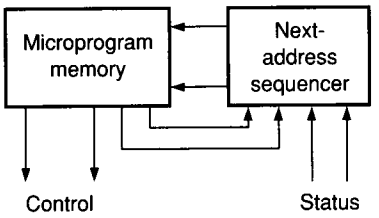

FIGURE 9.14-1
Simple microcoded controller.

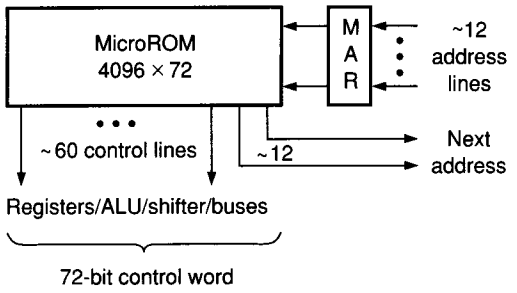

FIGURE 9.14-2
MicroROM architecture.

narrow data bus (nine multiplexed address lines and one data line for most 256k DRAMs), the memory (microROM) for a microcoded controller usually has a wide data bus relative to its address bus (perhaps 72 or more data lines compared with 12 or fewer address lines). Most of these data lines are dedicated to driving control points within the system. A few data lines are used to provide next-address information to the next-address sequencer. The next-address sequencer uses this address information along with status inputs from the controlled process to calculate the address of the next microinstruction.

A microROM organized as in Fig. 9.14-2 would contain almost 300k bits ($2^{12} \times 72 = 294,912$) of control information and would consume a correspondingly large silicon area. An alternative form for the microROM is shown in Fig. 9.14-3. This two-level microprogram memory consists of a relatively small microROM driving a secondary memory called a *nanoROM*. This organization is based on two reasonable assumptions. First, only a few of the 2^{72} possible control word combinations of Fig. 9.14-2 are necessary in a given system. Second, many of the control words that are necessary will be required repeatedly. If fewer than 256 unique control words are necessary, for example, and the microprogram memory is organized as shown in Fig. 9.14-3, only about 50k bits ($2^{12} \times 8 + 2^8 \times 72 = 51,200$) of control memory are required. This reduction in memory size is not free; the two-level microROM is slower than a single-level memory because a memory access must traverse two memory units to produce data. Overlapped instruction fetch and execution and careful design of control circuitry to minimize additional delay can partially offset the slower control memory.

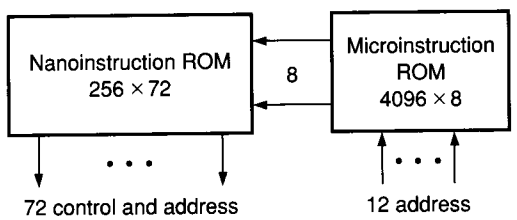

FIGURE 9.14-3
Two-level microprogram memory.

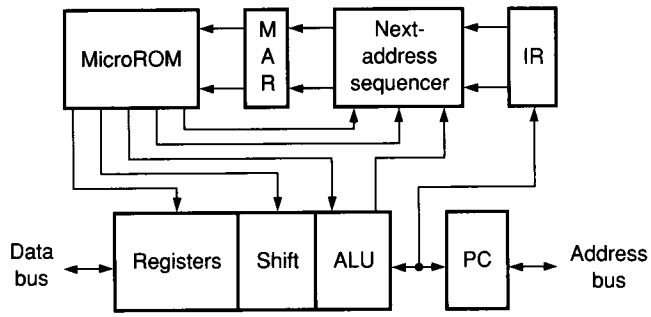

FIGURE 9.14-4
Microprogram-controlled microprocessor architecture.

A simplified block diagram for a microprogram-controlled microprocessor is given in Fig. 9.14-4. The microprogrammed controller drives a data path with registers, a shifter, and an arithmetic logic unit (ALU). A memory address register (MAR), an instruction register (IR), and a program counter (PC) are also shown. The operation of this design will be examined with the following example.

Example 9.14-1. Simple microprogrammed instruction execution Explain how the microprogrammed controller of Fig. 9.14-4 could be used to add the contents of two registers (register 4 and register 7) and return the sum to register 4. In register transfer form, the required operation is

$$\text{Register 4} \leftarrow \text{Register 4} + \text{Register 7}$$

Solution. To start execution, a program memory address is placed in the PC. The PC contents are placed on the address bus and a computer instruction is fetched from program memory (not shown) over the data bus and placed in the IR. The next-address sequencer uses the instruction in the IR to specify a particular starting address in the microROM. The corresponding microprogram control word is output from the microROM. This control word selects register 4 and subsequently gates register 4's contents to the ALU. If the register file is organized for dual-port read, the same control word from the microROM simultaneously selects register 7 and gates its contents to the ALU along a second bus. A next microROM control word address is generated by the next-address sequencer. This address selects another microROM control word, which causes the ALU to add its two input operands. Still another address is provided by the next-address sequencer, and a third microROM control word is selected. This last control word stores the result of the ALU operation in register 4 and prepares the microprocessor to fetch the next instruction for the IR from program memory by updating the PC contents.

This rather simplistic description of an ADD instruction demonstrates basic operation of a microprogram-controlled data path. In many instruction sequences, the next microROM address depends on results of an ALU operation. This allows conditional branching to be implemented. The preceding description omits many important considerations, including timing, pipelined operation, program counter update, and control signal generation.

It is appropriate at this point to compare the PLA and microprogrammed forms of FSMs. In general, a microprogrammed control unit is more complex than the corresponding PLA FSM because of the next-address generation circuitry. In fact, a PLA FSM can be compiled automatically once the state equations for the system are determined; this is much more difficult for a microprogrammed FSM. The peripheral circuitry for a microprogrammed FSM usually depends on the application and is thus not automatically generated. For these reasons, the PLA FSM is normally best for small, simple systems where minimum design time and circuit area are required. However, larger systems are sometimes created through use of several small PLA FSMs to offset the difficulties with large PLAs. The microprogram machine is usually more desirable for larger systems, where the additional design and area penalties can be offset by its advantages. A significant advantage is the ability to substantially change the details of operation by modifying the contents of the microROM prior to manufacture without changing the underlying circuit design and layout. This may be necessary for correction of design errors or to create new capabilities for a working design.

General agreement on the form of FSM that is most appropriate for commercial microprocessor design does not exist. Recent 32-bit microprocessors have been designed using each of these FSM forms. For example, the Bellmac 32 uses several small PLAs for its control circuitry, and the HP 9000 uses a large microprogrammed memory to control its operation.[19,20]

9.15 MICROPROCESSOR DESIGN

The focus of this chapter is structured forms of digital integrated circuits. The evolution of microprocessors provides an interesting study in the development of structured logic forms. The earliest microprocessors, the Intel 4004 and 8008, were born to counteract the high development costs for custom large-scale integrated (LSI) circuits.[21] Because custom large-scale circuits had to be designed for specific tasks, it was often difficult to reach the sales volume required to justify the development of a custom part. This literally forced the development of a logic form (the microprocessor) that could be tailored to many different applications by the addition of control logic (programs) contained in separate integrated circuit devices (memory chips). Only through the large application market that could be served could the development costs for a custom LSI circuit like the microprocessor be recovered.

As the complexity of microprocessors has increased, the design time and costs have also expanded. Development of structured designs using regular logic forms, such as those discussed in this chapter along with new computer-aided design tools, has been required to allow the evolution of microprocessor architecture. A comparison of the Intel 4004 die photograph (Fig. 9.15-1) with the Intel 80386 die photograph (Fig. 9.15-2) provides a vivid illustration of the relative percentages of silicon area used for regular structures and the relative complexity of these two microprocessors.

Today's basic microprocessor consists of a *control unit* and a *data path*. This is shown by Fig. 9.14-4 of the previous section, where the data path consists

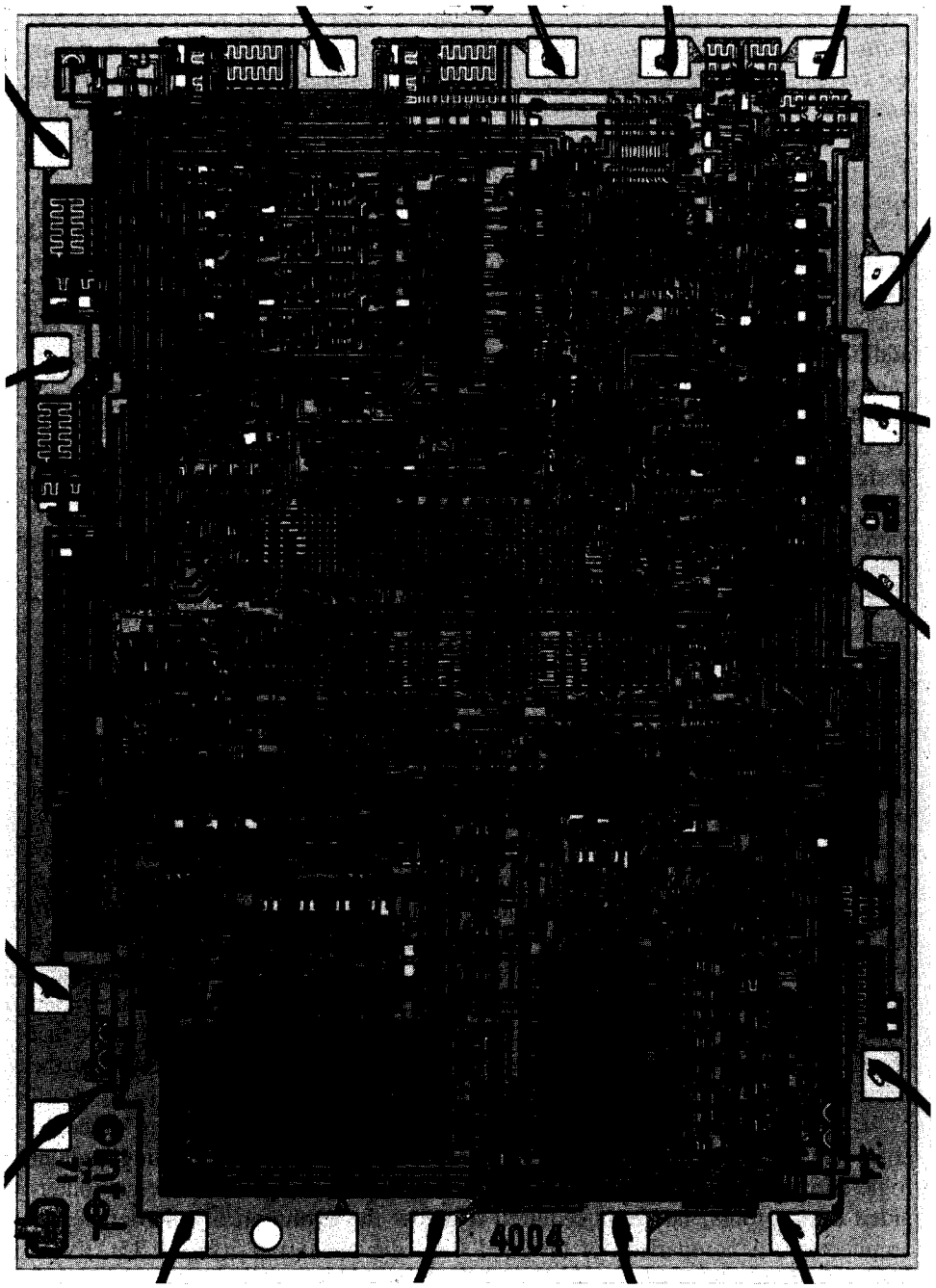

FIGURE 9.15-1
Intel 4004 die photograph (Courtesy Intel Corp.).

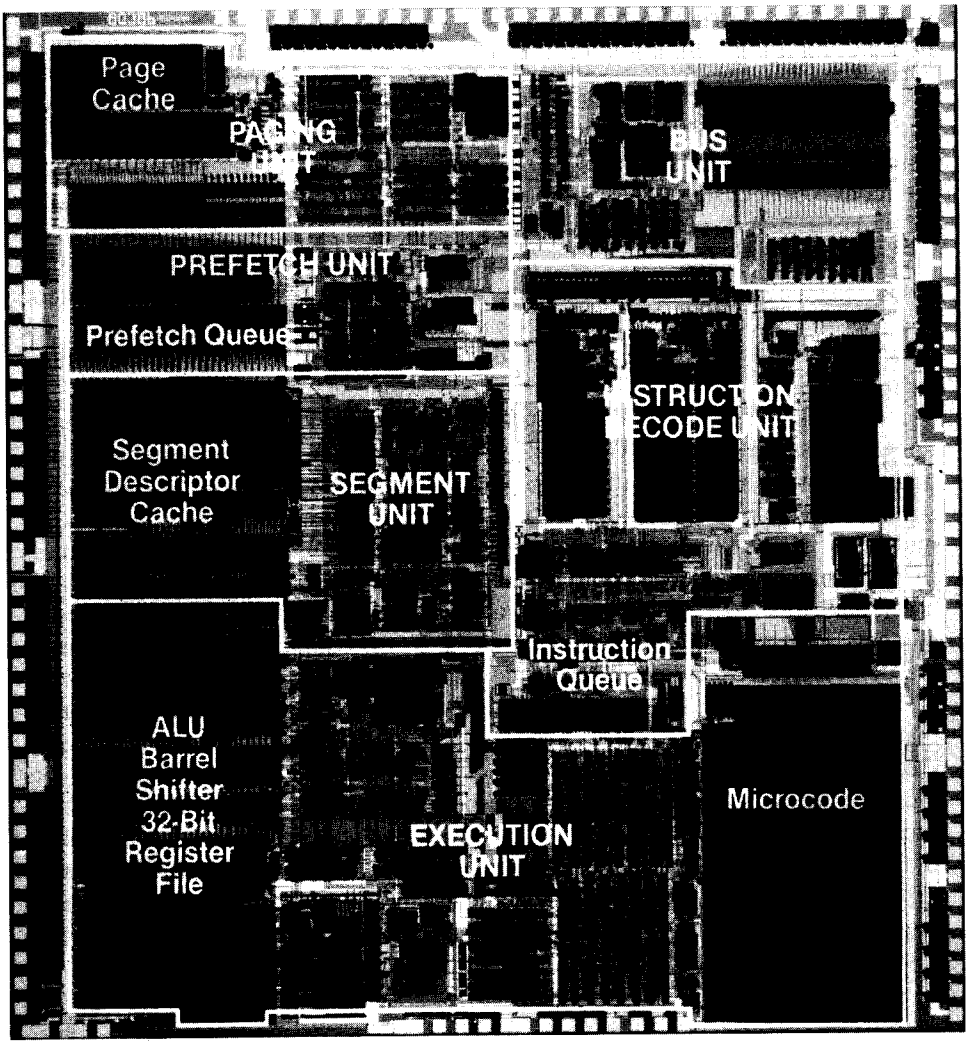

FIGURE 9.15-2
Intel 80386 die photograph (Courtesy Intel Corp.).

of registers, shifter, and ALU, and the control unit contains the microROM, MAR, next-address sequencer, IR, and PC. Although these two subsystems (control unit and data path) may be augmented with bus interface controllers, memory management units, cache memory, and other functions by different manufacturers, the present discussion will focus on the control unit and the data path as essential components of a microprocessor. The data path for a microprocessor is usually formed with 8, 16, or 32 identical bit paths. As a result of these identical bit paths, there is an inherent regularity within the data path for microprocessors. In contrast, the control units have varied structures, with most present manufacturers choosing microcoded or PLA style controllers.

9.15.1 Data Path Description

The data path, sometimes called the *execution unit*, is the place where the microprocessor executes operations such as addition, subtraction, shifts, rotates, and Boolean logical functions on data. Figure 9.15-3 shows a typical n-bit data path structure consisting of a dual-port register array, a barrel shifter, an ALU, interconnection buses, and support circuitry. Data flows along n parallel paths in the horizontal direction, while control of the data flow and ALU operations is provided vertically from the top of the data path. Execution of a typical data path operation (see Example 9.14-1) requires selection of operands from two registers, execution of an operation on the two selected operands, and placement of the result in a register. The elements of the data path must be designed to facilitate such operations.

The use of a dual-port register array is convenient for the fast execution of microprocessor programs. This local storage is usually provided within the data path as a small array of static memory cells. These are organized as an $n \times m$ structure where n is the width in bits of the microprocessor data bus and m is the number of registers provided. Because an ALU operation often requires access to the contents of two registers before execution can commence, most register arrays are organized with dual-port read access. With a dual-port register array, contents from two separate registers can be fetched simultaneously to minimize the delay before execution of an operation can begin.

The memory cell structure of a dual-port register array is quite similar to that of an SRAM cell. There is a need, however, for two data buses and a mechanism to allow the contents of each register to be switched to either data bus. The memory cell used for the dual-port register array of the Berkeley RISC processor[22] is shown in Fig. 9.15-4. In this circuit, the designers took advantage of the provision of double-rail data access to allow the contents from two registers to be obtained simultaneously. Remember that double-rail access is normally required to allow storage of data in a simple cross-coupled inverter storage cell.

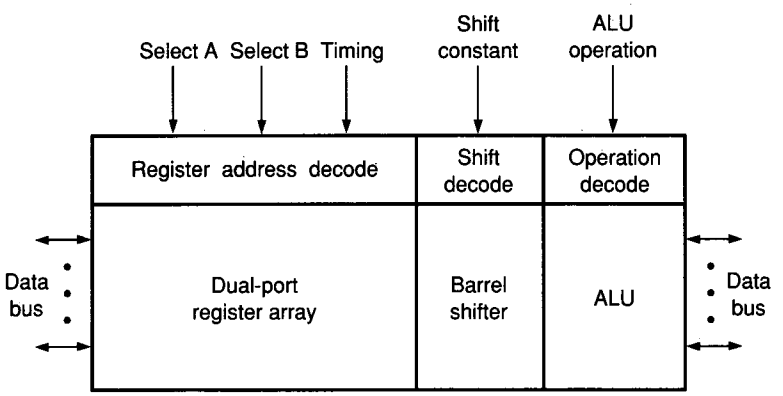

FIGURE 9.15-3
Microprocessor data path.

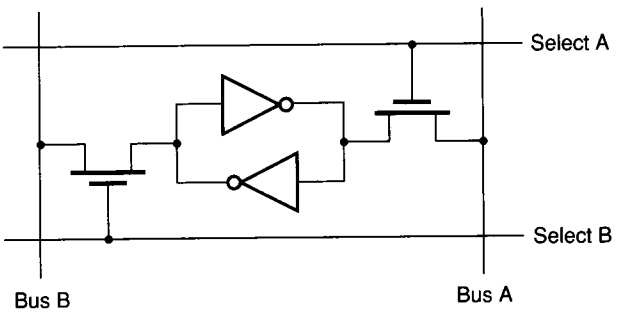

FIGURE 9.15-4
Dual-port register cell.

For a write operation, both data lines must be gated to the storage cell with complementary values. However, the contents of the storage cell may be read by gating the cell to either data line. If provision is made to drive the two select lines, A and B, separately for a read operation, then it is possible to obtain the data from a first register along one rail of the data bus (bus A), while the data from a second register is obtained along the other rail of the data bus (bus B). Of course, the data from the second bus will be the complement of the cell data and must be inverted.

9.15.2 Barrel Shifter

A second component that is included in the data path for many microprocessors is a structure that allows the contents of the data path to be shifted or rotated. A variable-length shift of a bit on the data path requires the possibility of connecting the selected bit to any one of several other bit paths. A 1-to-n multiplexer circuit for each bit will accomplish the desired connection. An ideal means of implementing multiplexer circuits is provided by the pass transistor available within MOS integrated circuits.

A particularly useful circuit structure to implement a shift or rotate is known as a *barrel shifter*. This circuit structure can be explained by first considering Fig. 9.15-5, which shows the circuit diagram of a general-purpose bus multiplexer for a 4-bit data path. This multiplexer circuit requires 16 pass transistors to allow connection of any bit line to any other bit line. If each pass transistor could be selected individually, 16 control lines would be required. Because most requirements are for parallel shifts with all bits moved the same number of bit positions, only four shift possibilities are really necessary. Figure 9.15-6 shows a better circuit with the pass transistors connected in groups of four, reducing control line requirements from 16 to 4 separate control lines, 50-53. A particular control line might be selected by encoding a 2-bit control field to drive a 2-to-4 decoder circuit. The individual decoder output would enable the proper shift control line. For a 32-bit data path, 1024 pass transistors are necessary to allow the desired shift operations. Assuming only parallel shifts, 32 control lines selected by a 5-bit encoded control field are sufficient.