2.2---:) https://github.com/AvinashShrivastav/MATHEMATICS-FOR-COMPUTING

Family	Speed (n sec)	Power Dissipation (m watts)	Faxout	Polar Supply (V)	High Logic Level (V)	Low Logic Level (V)
TTL	10	10	10	+5	+3	0.2
ECL	2	40	high	−5.2	−0.9	+1.75
CMOS	25	low	high	3–15	V_{cc}	0

Figure 2-1 Comparison of the basic logic families.

The metal-oxide semiconductor (MOS) is a unipolar transistor that depends on the flow of only one type of carrier, which may be electrons (*n*-channel) or holes (*p*-channel). This is in contrast to the bipolar transistor used in TTL and ECL gates, where both carriers exist during normal operation. A *p*-channel MOS is referred to as PMOS and an *n*-channel as NMOS. NMOS is the one that is commonly used in circuits with only one type of MOS transistor. The complementary MOS (CMOS) technology uses PMOS and NMOS transistors connected in a complementary fashion in all circuits. The most important advantages of CMOS over bipolar are the high packing density of circuits, a simpler processing technique during fabrication, and a more economical operation because of low power consumption. Figure 2-1 is a comparison of the basic logic families.

Because of their many advantages, integrated circuits are used exclusively to provide various digital components needed in the design of computer systems. To understand the organization and design of digital computers it is very important to be familiar with the various components encountered in integrated circuits. For this reason, the most basic components are introduced in this chapter with an explanation of their logical properties. These components provide a catalog of elementary digital functional units commonly used as basic building blocks in the design of digital computers.

2-2 Decoders

Discrete quantities of information are represented in digital computers with binary codes. A binary code of n bits is capable of representing up to 2^n distinct elements of the coded information. A decoder is a combinational circuit that converts binary information from the n coded inputs to a maximum of 2^n unique outputs. If the n-bit coded information has unused bit combinations, the decoder may have less than 2^n outputs.

The decoders presented in this section are called n-to-m-line decoders, where $m \leq 2^n$. Their purpose is to generate the 2^n (or fewer) binary combinations of the n input variables. A decoder has n inputs and m outputs and is also referred to as an $n \times m$ decoder.

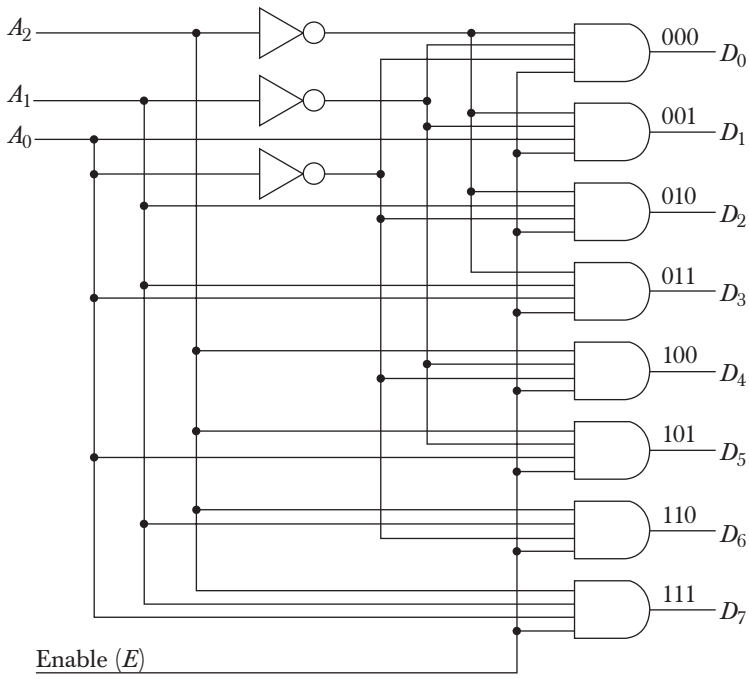


Figure 2-2 3-to-8-line decoder.

The logic diagram of a 3-to-8-line decoder is shown in Fig. 2-2. The three data inputs, A_0, A_1, and A_2 are decoded into eight outputs, each output representing one of the combinations of the three binary input variables. The three inverters provide the complement of the inputs, and each of the eight AND gates generates one of the binary combination. A particular application of this decoder is a binary-to-octal conversion. The input variables represent a binary number and the outputs represent the eight digits of the octal number system. However, a 3-to-8-line decoder can be used for decoding any 3-bit code to provide eight outputs, one for each combination of the binary code.

Enable input

Commercial decoders include one or more enable inputs to control the operation of the circuit. The decoder of Fig. 2-2 has one enable input, E. The decoder is enabled when E is equal to 1 and disabled when E is equal to 0.

The operation of the decoder can be clarified using the truth table listed in Table 2-1. When the enable input E is equal to 0, all the outputs are equal to 0 regardless of the values of the other three data inputs. The three $\times$'s in the table designate don't-care conditions. When the enable input is equal to 1, the decoder operates in a normal fashion. For each possible input combination, there are seven outputs that are equal to 0 and only one that is equal to 1. The output variable whose value is equal to 1 represents the octal number equivalent of the binary number that is available in the input data lines.

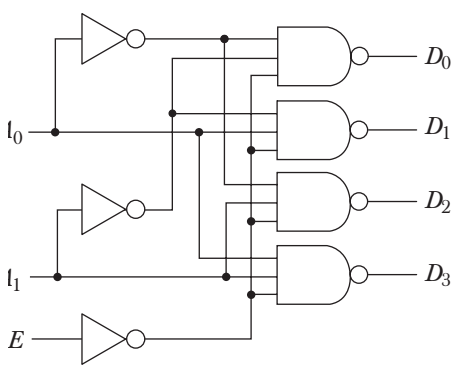
TABLE 2-1 Truth Table for 3-to-8-Line Decoder

Enable	Inputs			Outputs							
E	A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	×	×	×	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

NAND Gate Decoder

Some decoders are constructed with NAND instead of AND gates. Since a NAND gate produces the AND operation with an inverted output, it becomes more economical to generate the decoder outputs in their complement form. A 2-to-4-line decoder with an enable input constructed with NAND gates is shown in-Fig. 2-3. The circuit operates with complemented outputs and a complemented enable input E. The decoder is enabled when E is equal to 0. As indicated by the truth table, only one output is equal to 0 at any given time; the other three outputs are equal to 1. The output whose value is equal to 0 represents the equivalent binary number in inputs A_1 and A_0. The circuit is disabled when E is equal to 1, regardless of the values of the other two inputs. When the circuit is disabled, none of the outputs are selected and all outputs are equal to 1. In general, a decoder may operate with complemented or uncomplemented outputs. The enable input may be activated with a 0 or with a 1 signal level. Some decoders have two or

Figure 2-3 2-to-4-line decoder with NAND gates.



(a) Logic diagram

E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

(b) Truth table

more enable inputs that must satisfy a given logic condition in order to enable the circuit.

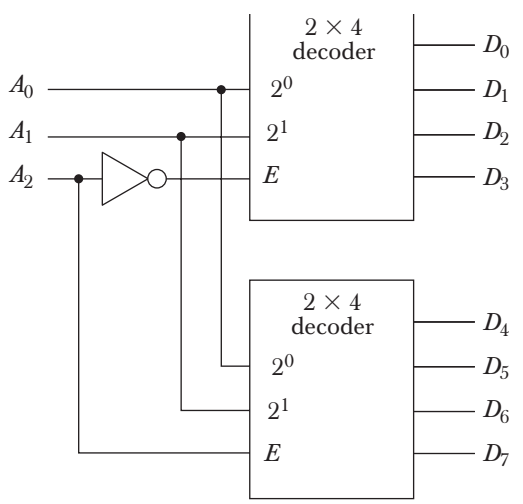
Decoder Expansion

There are occasions when a certain-size decoder is needed but only smaller sizes are available. When this occurs it is possible to combine two or more decoders with enable inputs to form a larger decoder. Thus if a 6-to-64-line decoder is needed, it is possible to construct it with four 4-to-16-line decoders.

Figure 2-4 shows how decoders with enable inputs can be connected to form a larger decoder. Two 2-to-4-line decoders are combined to achieve a 3-to-8-line decoder. The two least significant bits of the input are connected to both decoders. The most significant bit is connected to the enable input of one decoder and through an inverter to the enable input of the other decoder. It is assumed that each decoder is enabled when its E input is equal to 1. When E is equal to 0, the decoder is disabled and all its outputs are in the 0 level. When $A_2 = 0$, the upper decoder is enabled and the lower is disabled. The lower decoder outputs become inactive with all outputs at 0. The outputs of the upper decoder generate outputs D_0 through D_3, depending on the values of A_1 and A_0 (while $A_2 = 0$). When $A_2 = 1$, the lower decoder is enabled and the upper is disabled. The lower decoder output generates the binary equivalent D_4 through D_7 since these binary numbers have a 1 in the A_2 position.

The example demonstrates the usefulness of the enable input in decoders or any other combinational logic component. Enable inputs are a convenient feature for interconnecting two or more circuits for the purpose of expanding the digital component into a similar function but with more inputs and outputs.

Figure 2-4 A 3 × 8 decoder constructed with two 2 × 4 decoders.



Encoders

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or less) input lines and n outputs lines. The output lines generate the binary code corresponding to the input value. An example of an encoder is the octal-to-binary encoder, whose truth table is given in Table 2-2. It has eight inputs, one for each of the octal digits, and three outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time; otherwise, the circuit has no meaning.

TABLE 2-2 Truth Table for Octal-to-Binary Encoder

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output $A_0 = 1$ if the input octal digit is 1 or 3 or 5 or 7. Similar conditions apply for the other two outputs. These conditions can be expressed by the following Boollean functions:

$$A_0 = D_1 + D_3 + D_5 + D_7$$
$$A_1 = D_2 + D_3 + D_6 + D_7$$
$$A_2 = D_4 + D_5 + D_6 + D_7$$

The encoder can be implemented with three OR gates.

2-3 Multiplexers

multiplexer

A multiplexer is a combinational circuit that receives binary information from one of 2^n input data lines and directs it to a single output line. The selection of a particular input data line for the output is determined by a set of selection inputs. A 2^n-to-1 multiplexer has 2^n input data lines and n input selection lines whose bit combinations determine which input data are selected for the output.

A 4-to-l-line multiplexer is shown in Fig. 2-5. Each of the four data inputs I_0 through I_3 is applied to one input of an AND gate. The two selection inputs S_1 and

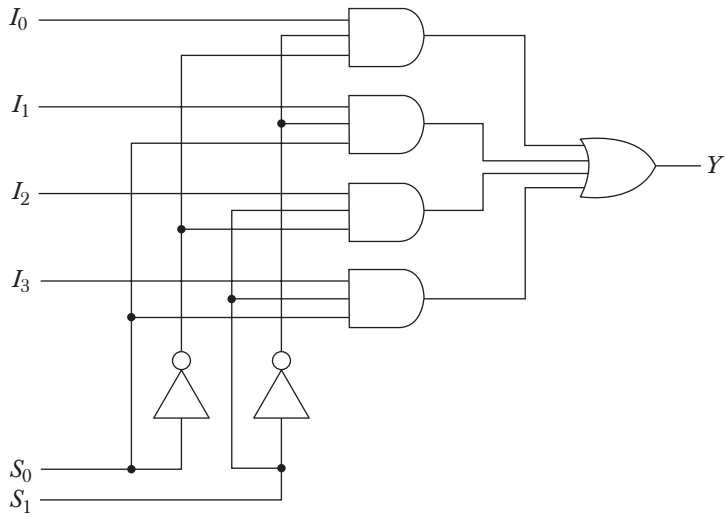


Figure 2-5 4-to-1-line multiplexer.

S_0 are decoded to select a particular AND gate. The outputs of the AND gates are applied to a single OR gate to provide the single output. To demonstrate the circuit operation, consider the case when $S_1S_0 = 10$. The AND gate associated with input I_2 has two of its inputs equal to 1. The third input of the gate is connected to I_2. The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to the value of I_2, thus providing a path from the selected input to the output.

The 4-to-l line multiplexer of Fig. 2-5 has six inputs and one output. A truth table describing the circuit needs 64 rows since six input variables can have 2^6 binary combinations. This is an excessively long table and will not be shown here. A more convenient way to describe the operation of multiplexers is by means of a function table. The function table for the multiplexer is shown in Table 2-3. The table demonstrates the relationship between the four data inputs and the single output as a function of the selection inputs S_1 and S_0. When the selection inputs

TABLE 2-3 Function Table for 4-to-1-Line Multiplexer

Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

are equal to 00, output Y is equal to input I_0. When the selection inputs are equal to 01, input I_1 has a path to output Y, and similarly for the other two combinations. The multiplexer is also called a *data selector,* since it selects one of many data inputs and steers the binary information to the output.

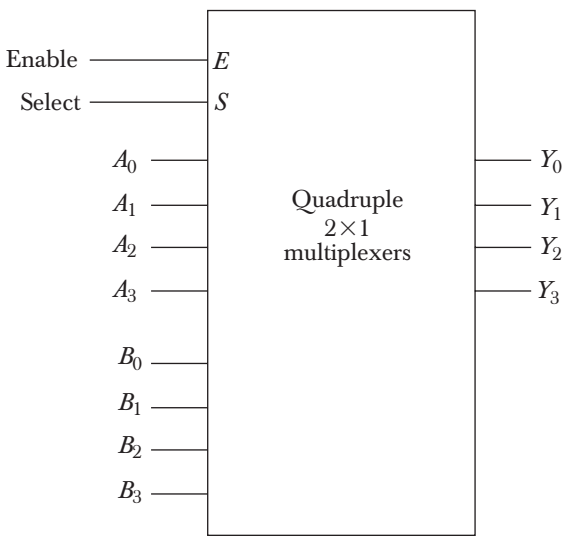
data selector

The AND gates and inverters in the multiplexer resemble a decoder circuit, and indeed they decode the input selection lines. In general, a 2^n-to-l-line multiplexer is constructed from an n-to-2^n decoder by adding to it 2^n input lines, one from each data input. The size of the multiplexer is specified by the number 2^n of its data inputs and the single output. It is then implied that it also contains n input selection lines. The multiplexer is often abbreviated as MUX.

As in decoders, multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the outputs are disabled, and when it is in the active state, the circuit functions as a normal multiplexer. The enable input is useful for expanding two or more multiplexers to a multiplexer with a larger number of inputs.

In some cases two or more multiplexers are enclosed within a single integrated circuit package. The selection and the enable inputs in multiple-unit construction are usually common to all multiplexers. As an illustration, the block diagram of a quadruple 2-to-l-line multiplexer is shown in Fig. 2-6. The circuit has four multiplexers, each capable of selecting one of two input lines. Output Y_0 can be selected to come from either input A_0 or B_0. Similarly, output Y_1 may have the value of A_1 or B_1 and so on. One input selection line S selects one of the lines in each of the four multiplexers. The enable input E must be active for normal

Figure 2-6 Quadruple 2-to-1 line multiplexers.



(a) Block diagram

E	S	Y
0	x	All 0's
1	0	A
1	1	B

(b) Function table

2.3---:) https://github.com/AvinashShrivastav/MATHEMATICS-FOR-COMPUTING

Encoders

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or less) input lines and n outputs lines. The output lines generate the binary code corresponding to the input value. An example of an encoder is the octal-to-binary encoder, whose truth table is given in Table 2-2. It has eight inputs, one for each of the octal digits, and three outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time; otherwise, the circuit has no meaning.

TABLE 2-2 Truth Table for Octal-to-Binary Encoder

Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	A_2	A_1	A_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output $A_0 = 1$ if the input octal digit is 1 or 3 or 5 or 7. Similar conditions apply for the other two outputs. These conditions can be expressed by the following Boollean functions:

$$A_0 = D_1 + D_3 + D_5 + D_7$$
$$A_1 = D_2 + D_3 + D_6 + D_7$$
$$A_2 = D_4 + D_5 + D_6 + D_7$$

The encoder can be implemented with three OR gates.

2-3 Multiplexers

multiplexer

A multiplexer is a combinational circuit that receives binary information from one of 2^n input data lines and directs it to a single output line. The selection of a particular input data line for the output is determined by a set of selection inputs. A 2^n-to-1 multiplexer has 2^n input data lines and n input selection lines whose bit combinations determine which input data are selected for the output.

A 4-to-l-line multiplexer is shown in Fig. 2-5. Each of the four data inputs I_0 through I_3 is applied to one input of an AND gate. The two selection inputs S_1 and

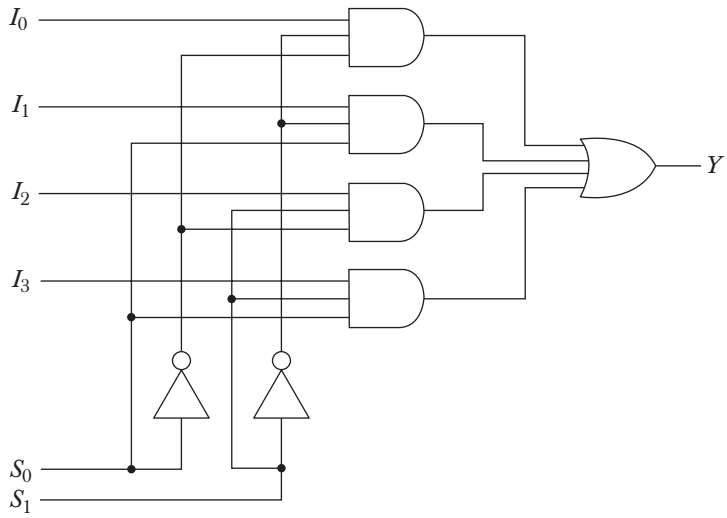


Figure 2-5 4-to-1-line multiplexer.

S_0 are decoded to select a particular AND gate. The outputs of the AND gates are applied to a single OR gate to provide the single output. To demonstrate the circuit operation, consider the case when $S_1S_0 = 10$. The AND gate associated with input I_2 has two of its inputs equal to 1. The third input of the gate is connected to I_2. The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The OR gate output is now equal to the value of I_2, thus providing a path from the selected input to the output.

The 4-to-l line multiplexer of Fig. 2-5 has six inputs and one output. A truth table describing the circuit needs 64 rows since six input variables can have 2^6 binary combinations. This is an excessively long table and will not be shown here. A more convenient way to describe the operation of multiplexers is by means of a function table. The function table for the multiplexer is shown in Table 2-3. The table demonstrates the relationship between the four data inputs and the single output as a function of the selection inputs S_1 and S_0. When the selection inputs

TABLE 2-3 Function Table for 4-to-1-Line Multiplexer

Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

are equal to 00, output Y is equal to input I_0. When the selection inputs are equal to 01, input I_1 has a path to output Y, and similarly for the other two combinations. The multiplexer is also called a *data selector,* since it selects one of many data inputs and steers the binary information to the output.

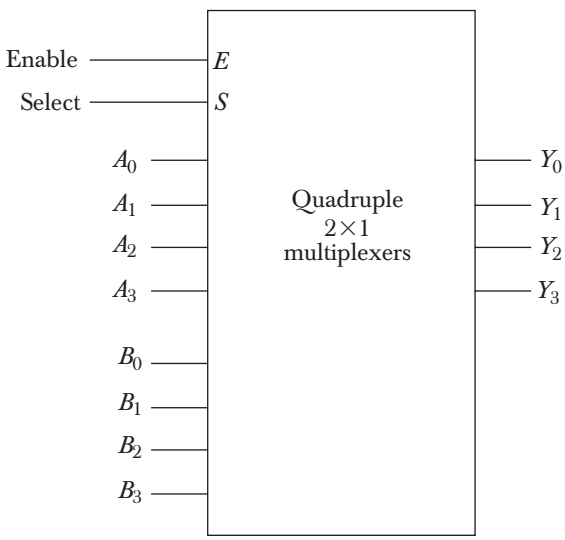
data selector

The AND gates and inverters in the multiplexer resemble a decoder circuit, and indeed they decode the input selection lines. In general, a 2^n-to-l-line multiplexer is constructed from an n-to-2^n decoder by adding to it 2^n input lines, one from each data input. The size of the multiplexer is specified by the number 2^n of its data inputs and the single output. It is then implied that it also contains n input selection lines. The multiplexer is often abbreviated as MUX.

As in decoders, multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the outputs are disabled, and when it is in the active state, the circuit functions as a normal multiplexer. The enable input is useful for expanding two or more multiplexers to a multiplexer with a larger number of inputs.

In some cases two or more multiplexers are enclosed within a single integrated circuit package. The selection and the enable inputs in multiple-unit construction are usually common to all multiplexers. As an illustration, the block diagram of a quadruple 2-to-l-line multiplexer is shown in Fig. 2-6. The circuit has four multiplexers, each capable of selecting one of two input lines. Output Y_0 can be selected to come from either input A_0 or B_0. Similarly, output Y_1 may have the value of A_1 or B_1 and so on. One input selection line S selects one of the lines in each of the four multiplexers. The enable input E must be active for normal

Figure 2-6 Quadruple 2-to-1 line multiplexers.



(a) Block diagram

E	S	Y
0	x	All 0's
1	0	A
1	1	B

(b) Function table

operation. Although the circuit contains four multiplexers, we can also think of it as a circuit that selects one of two 4-bit data lines. As shown in the function table, the unit is enabled when $E = 1$. Then, if $S = 0$, the four A inputs have a path to the four outputs. On the other hand, if $S = 1$, the four B inputs are applied to the outputs. The outputs have all 0's when $E = 0$, regardless of the values of S.

Typical applications of multiplexers are data routing, parallel-to-serial conversion, and logic function generation. An n-variable logic function can be generated using n-select inputs of a multiplexer. Digital Multiplexers are thus considered universal logic modules.

2-4 Registers

A register is a group of flip-flops with each flip-flop capable of storing one bit of information. An n-bit register has a group of n flip-flops and is capable of storing any binary information of n bits. In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks. In its broadest definition, a register consists of a group of flip-flops and gates that effect their transition. The flip-flops hold the binary information and the gates control when and how new information is transferred into the register.

Various types of registers are available commercially. The simplest register is one that consists only of flip-flops, with no external gates. Figure 2-7 shows such a register constructed with four D flip-flops. The common clock input triggers all flip-flops on the rising edge of each pulse, and the binary data available at the four inputs are transferred into the 4-bit register. The four outputs can be sampled at any time to obtain the binary information stored in the register. The *clear* input goes to a special terminal in each flip-flop. When this input goes to 0, all flip-flops are reset asynchronously. The clear input is useful for clearing the register to all 0's prior to its clocked operation. The clear input must be maintained at logic 1 during normal clocked operation. Note that the clock signal enables the D input but that the clear input is independent of the clock.

register load

The transfer of new information into a register is referred to as *loading* the register. If all the bits of the register are loaded simultaneously with a common clock pulse transition, we say that the loading is done in parallel. A clock transition applied to the C inputs of the register of Fig. 2-7 will load all four inputs I_0 through I_3 in parallel. In this configuration, the clock must be inhibited from the circuit if the content of the register must be left unchanged.

Register with Parallel Load

Most digital systems have a master clock generator that supplies a continuous train of clock pulses. The clock pulses are applied to all flip-flops and registers in the system. The master clock acts like a pump that supplies a constant beat to all parts of the system. A separate control signal must be used to decide which specific clock pulse will have an effect on a particular register.

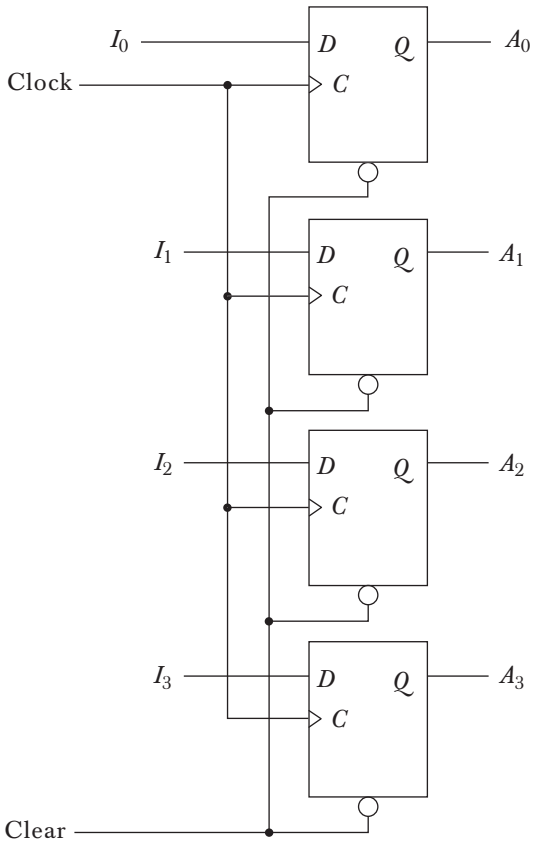


Figure 2-7 4-bit register.

A 4-bit register with a load control input that is directed through gates and into the *D* inputs is shown in Fig. 2-8. The *C* inputs receive clock pulses at all times. The buffer gate in the clock input reduces the power requirement from the clock generator. Less power is required when the clock is connected to only one input gate instead of the power consumption that four inputs would have required if the buffer were not used.

load input

The load input in the register determines the action to be taken with each clock pulse. When the load input is 1, the data in the four inputs are transferred into the register with the next positive transition of a clock pulse. When the load input is 0, the data inputs are inhibited and the *D* inputs of the flip-flops are connected to their outputs. The feedback connection from output to input is necessary because the *D* flip-flop does not have a "no change" condition. With each clock pulse, the *D* input determines the next state of the output. To leave the output unchanged, it is necessary to make the *D* input equal to the present value of the output.

Note that the clock pulses are applied to the *C* inputs at all times. The load input determines whether the next pulse will accept new information or leave the

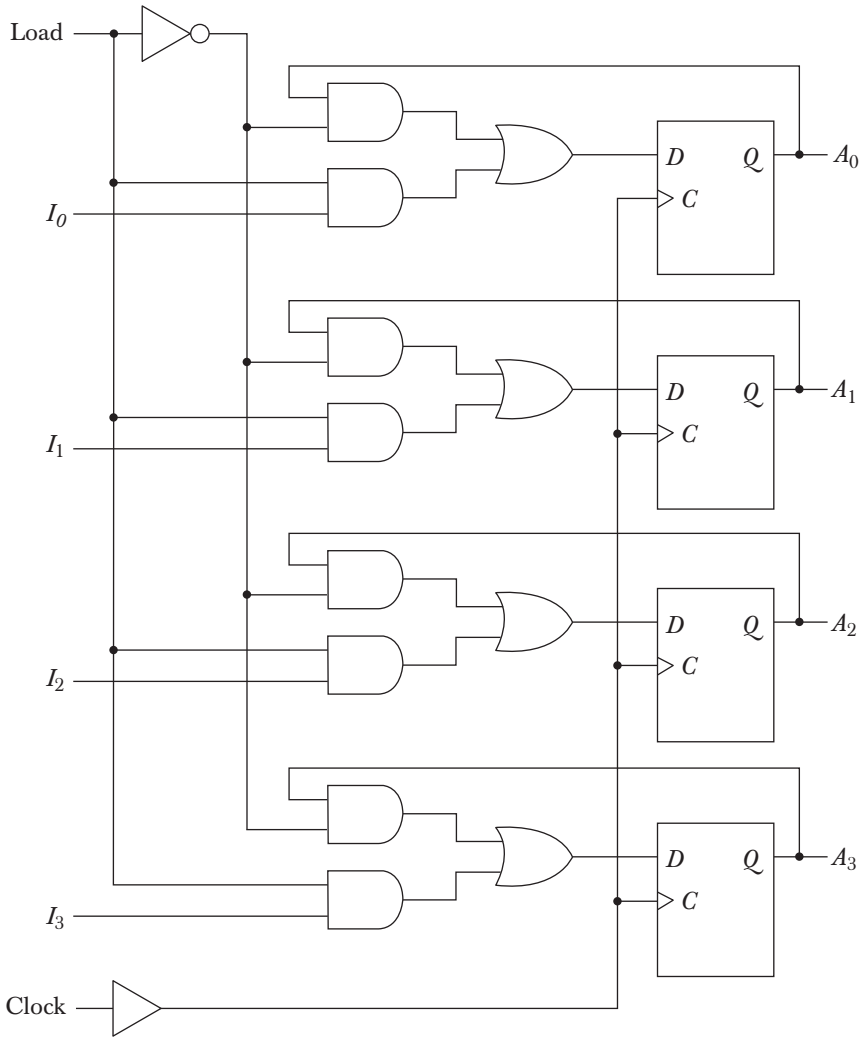


Figure 2-8 4-bit register with parallel load.

information in the register intact. The transfer of information from the inputs into the register is done simultaneously with all four bits during a single pulse transition.

2-5 Shift Registers

A register capable of shifting its binary information in one or both directions is called a shift register. The logical configuration of a shift register consists of a chain of flip-flops in cascade, with the output of one flip-flop connected to the input of

2.7---:) https://github.com/AvinashShrivastav/MATHEMATICS-FOR-COMPUTING

TABLE 2-5 Function Table for the Register of Fig. 2-12

Clock	Clear	Load	Increment	Operation
↑	0	0	0	No change
↑	0	0	1	Increment count by 1
↑	0	1	×	Load inputs I_0 through I_3
↑	1	×	×	Clear outputs to 0

increment

operations. The *increment* operation adds one to the content of a register. By enabling the count input during one clock period, the content of the register can be incremented by one.

2-7 Memory Unit

A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of storage. The memory stores binary

word

information in groups of bits called *words*. A word in memory is an entity of bits that move in and out of storage as a unit. A memory word is a group of l's and 0's and may represent a number, an instruction code, one or more alphanumeric characters, or any other binary-coded information. A group of eight bits is called

byte

a *byte*. Most computer memories use words whose number of bits is a multiple of 8. Thus a 16-bit word contains two bytes, and a 32-bit word is made up of four bytes. The capacity of memories in commercial computers is usually stated as the total number of bytes that can be stored.

The internal structure of a memory unit is specified by the number of words it contains and the number of bits in each word. Special input lines called address lines select one particular word. Each word in memory is assigned an identification number, called an address, starting from 0 and continuing with 1, 2, 3, up to $2^k - 1$ where k is the number of address lines. The selection of a specific word inside the memory is done by applying the k-bit binary address to the address lines. A decoder inside the memory accepts this address and opens the paths needed to select the bits of the specified word. Computer memories may range from 1024 words, requiring an address of 10 bits, to 2^{32} words, requiring 32 address bits. It is customary to refer to the number of words (or bytes) in a memory with one of the letters K (kilo), M (mega), or G (giga). K is equal to 2^{10}, M is equal to 2^{20}, and G is equal to 2^{30}. Thus, $64K = 2^{16}$, $2M = 2^{21}$, and $4G = 2^{32}$.

Two major types of memories are used in computer systems: random-access memory (RAM) and read-only memory (ROM). These semiconductor memories are classified into Random Access Memories (RAMs) and Sequential Access Memories (SAMs) based on access time. Memories constructed with shift registers, Charge Coupled Devices (CCDs), or bubble memories are examples of SAMs. RAMs are categorized into ROMs, Read Mostly Memories (RMMs), and Read Write Memories (RWMs). ROMs are of two types: Masked Programmed

ROMs and user Programmed PROMs. Two types of RMMs are Erasable and Programmable (EPROM), and Electrically Erasable (EEPROM). RWMs are Static RAM (SRAM) and Dynamic RAM (DRAM). Static RAMs have memory cells as common Flip-Flops. Dynamic RAMs have memory cells that must be refreshed, read and written periodically to avoid loss of memory cells.

Random-Access Memory

RAM

In random-access memory (RAM) the memory cells can be accessed for information transfer from any desired random location. That is, the process of locating a word in memory is the same and requires an equal amount of time no matter where the cells are located physically in memory: thus the name "random access."

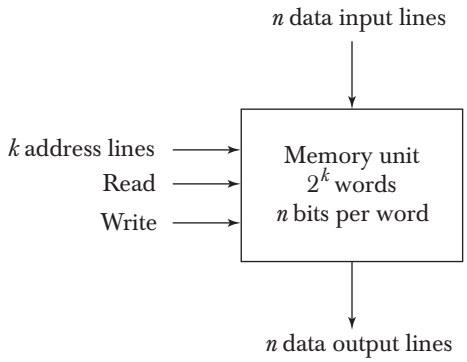
Communication between a memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer. A block diagram of a RAM unit is shown in Fig. 2-13. The n data input lines provide the information to be stored in memory, and the n data output lines supply the information coming out of memory. The k address lines provide a binary number of k bits that specify a particular word chosen among the 2^k available inside the memory. The two control inputs specify the direction of transfer desired.

write and read operations

The two operations that a random-access memory can perform are the write and read operations. The write signal specifies a transfer-in operation and the read signal specifies a transfer-out operation. On accepting one of these control signals, the internal circuits inside the memory provide the desired function. The steps that must be taken for the purpose of transferring a new word to be stored into memory are as follows:

1. Apply the binary address of the desired word into the address lines.
2. Apply the data bits that must be stored in memory into the data input lines.
3. Activate the *write* input.

Figure 2-13 Block diagram of random access memory (RAM).



The memory unit will then take the bits presently available in the input data lines and store them in the word specified by the address lines.

The steps that must be taken for the purpose of transferring a stored word out of memory are as follows:

1. Apply the binary address of the desired word into the address lines.
2. Activate the *read* input.

The memory unit will then take the bits from the word that has been selected by the address and apply them into the output data lines. The content of the selected word does not change after reading.

Read-Only Memory

As the name implies, a read-only memory (ROM) is a memory unit that performs the read operation only; it does not have a write capability. This implies that the binary information stored in a ROM is made permanent during the hardware production of the unit and cannot be altered by writing different words into it. Whereas a RAM is a general-purpose device whose contents can be altered during the computational process, a ROM is restricted to reading words that are permanently stored within the unit. The binary information to be stored, specified by the designer, is then embedded in the unit to form the required interconnection pattern. ROMs come with special internal electronic fuses that can be "programmed" for a specific configuration. Once the pattern is established, it stays within the unit even when power is turned off and on again.

An $m \times n$ ROM is an array of binary cells organized into m words of n bits each. As shown in the block diagram of Fig. 2-14, a ROM has k address input lines to select one of $2^k = m$ words of memory, and n output lines, one for each bit of the word. An integrated circuit ROM may also have one or more enable inputs for expanding a number of packages into a ROM with larger capacity.

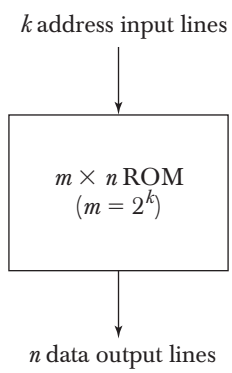


Figure 2-14 Block diagram of read only memory (ROM).

The ROM does not need a read-control line since at any given time, the output lines automatically provide the n bits of the word selected by the address value. Because the outputs are a function of only the present inputs (the address lines), a ROM is classified as a combinational circuit. In fact, a ROM is constructed internally with decoders and a set of OR gates. There is no need for providing storage capabilities as in a RAM, since the values of the bits in the ROM are permanently fixed.

ROMs find a wide range of applications in the design of digital systems. Basically, a ROM generates an input–output relation specified by a truth table. As such, it can implement any combinational circuit with k inputs and n outputs. When employed in a computer system as a memory unit, the ROM is used for storing fixed programs that are not to be altered and for tables of constants that are not subject to change. ROM is also employed in the design of control units for digital computers. As such, they are used to store coded information that represents the sequence of internal control variables needed for enabling the various operations in the computer. A control unit that utilizes a ROM to store binary control information is called a microprogrammed control unit. This subject is dicussed in more detail in Chapter 7.

Types of ROMs

The required paths in a ROM may be programmed in three different ways. The first, *mask programming,* is done by the semiconductor company during the last fabrication process of the unit. The procedure for fabricating a ROM requires that the customer fill out the truth table that he or she wishes the ROM to satisfy. The truth table may be submitted in a special form provided by the manufacturer or in a specified format on a computer output medium. The manufacturer makes the corresponding mask for the paths to produce the l's and 0's according to the customer's truth table. This procedure is costly because the vendor charges the customer a special fee for custom masking the particular ROM. For this reason, mask programming is economical only if a large quantity of the same ROM configuration is to be ordered.

PROM

For small quantities it is more economical to use a second type of ROM called a *programmable read-only memory* or PROM. When ordered, PROM units contain all the fuses intact, giving all l's in the bits of the stored words. The fuses in the PROM are blown by application of current pulses through the output terminals for each address. A blown fuse defines a binary 0 state, and an intact fuse gives a binary 1 state. This allows users to program PROMs in their own laboratories to achieve the desired relationship between input addresses and stored words. Special instruments called *PROM programmers* are available commercially to facilitate this procedure. In any case, all procedures for programming ROMs are hardware procedures even though the word "programming" is used.

The hardware procedure for programming ROMs or PROMs is irreversible, and once programmed, the fixed pattern is permanent and cannot be altered. Once a bit pattern has been established, the unit must be discarded if the bit pattern is to be changed. A third type of ROM available is called *erasable PROM* or EPROM.

The EPROM can be restructured to the initial value even though its fuses have been blown previously. When the EPROM is placed under a special ultraviolet light for a given period of time, the shortwave radiation discharges the internal gates that serve as fuses. After erasure, the EPROM returns to its initial state and can be reprogrammed to a new set of words. Certain PROMs can be erased with electrical signals instead of ultraviolet light. These PROMs are called *electrically erasable PROM* or EEPROM. Flash memory is a form of EEPROM in which a block of bytes can be erased in a very short duration. Example applications of EEPROM devices are:

EEPROM

1. storing current time and date in a machine.
2. storing port statusses.

Examples of flash memory device applications are:

1. storing messages in a mobile phone.
2. storing photographs in a digital camera.

PROBLEMS

2-1. TTL SSI come mostly in 14-pin 1C packages. Two pins are reserved for power supply and the other pins are used for input and output terminals. How many circuits are included in one such package if it contains the following type of circuits? (a) Inverters; (b) two-input exclusive-OR gates; (c) three-input OR gates; (d) four-input AND gates; (e) five-input NOR gates; (f) eight-input NAND gates; (g) clocked *JK* flip-flops with asynchronous clear.

2-2. MSI chips perform elementary digital functions such as decoders, multiplexers, registers, and counters. The following are TTL-type integrated circuits that provide such functions. Find their description in a data book and compare them with the corresponding component presented in this chapter.

a. IC type 74155 dual 2-to-4-line decoders.
b. IC type 74157 quadruple 2-to-l-line multiplexers.
c. IC type 74194 4-bit bidirectional shift register with parallel load.
d. IC type 74163 4-bit binary counter with parallel load and synchronous clear.

2-3. Construct a 5-to-32-line decoder with four 3-to-8-line decoders with enable and one 2-to-4-line decoder. Use block diagrams similar to Fig. 2-4.

2-4. Draw the logic diagram of a 2-to-4-line decoder with only NOR gates. Include an enable input.

2-5. Modify the decoder of Fig. 2-3 so that the circuit is enabled when $E = 1$ and disabled when $E = 0$. List the modified truth table.

2-6. Draw the logic diagram of an eight-input, three-output encoder whose truth table is given in Table 2-2. What is the output when all the inputs are equal to 0? What is the output when only input D_0 is equal to 0? Establish a procedure that will distinguish between these two cases.

2-7. Construct a 16-to-l-line multiplexer with two 8-to-l-line multiplexers and one 2-to-l-line multiplexer. Use block diagrams for the three multiplexers.

2-8. Draw the block diagram of a dual 4-to-l-line multiplexers and explain its operation by means of a function table.

2-9. Include a two-input AND gate with the register of Fig. 2-7 and connect the gate output to the clock inputs of all the flip-flops. One input of the AND gate receives the clock pulses from the clock pulse generator. The other input of the AND gate provides a parallel load control. Explain the operation of the modified register.

2-10. What is the purpose of the buffer gate in the clock input of the register of Fig. 2-8?

2-11. Include a synchronous clear capability to the register with parallel load of Fig. 2-8.

2-12. The content of a 4-bit register is initially 1101. The register is shifted six times to the right with the serial input being 101101. What is the content of the register after each shift?

2-13. What is the difference between serial and parallel transfer? Using a shift register with parallel load, explain how to convert serial input data to parallel output and parallel input data to serial output.

2-14. A ring counter is a shift register as in Fig. 2-9 with the serial output connected to the serial input. Starting from an initial state of 1000, list the sequence of states of the four flip-flops after each shift.

2-15. The 4-bit bidirectional shift register with parallel load shown in Fig. 2-10 is enclosed within one IC package.

a. Draw a block diagram of the IC showing all inputs and outputs. Include two pins for power supply.

b. Draw a block diagram using two ICs to produce an 8-bit bidirectional shift register with parallel load.

2-16. How many flip-flops will be complemented in a 10-bit binary counter to reach the next count after (a) 1001100111; (b) 0011111111?

2-17. Show the connections between four 4-bit binary counters with parallel load (Fig. 2-12) to produce a 16-bit binary counter with parallel load. Use a block diagram for each 4-bit counter.

2-18. Show how the binary counter with parallel load of Fig. 2-12 can be made to operate as a divide-by-N counter (i.e., a counter that counts from 0000 to N-and back to 0000). Specifically show the circuit for a divide-by-10 counter using the counter of Fig. 2-12 and an external AND gate.

2-19. The following memory units are specified by the number of words times the number of bits per word. How many address lines and input-output data lines are needed in each case? (a) 2K $\times$ 16; (b) 64K $\times$ 8; (c) 16M $\times$ 32; (d) 4G $\times$ 64.

2-20. Specify the number of bytes that can be stored in the memories listed in Prob. 2–19.

2-21. How many 128 $\times$ 8 memory chips are needed to provide a memory capacity of 4096 $\times$ 16?

2-22. Given a 32 $\times$ 8 ROM chip with an enable input, show the external connections necessary to construct a 128 $\times$ 8 ROM with four chips and a decoder.

2-23. A ROM chip of 4096 $\times$ 8 bits has two enable inputs and operates from a 5-volt power supply. How many pins are needed for the integrated circuit package? Draw a block diagram and label all input and output terminals in the ROM.

REFERENCES

1. Hill, F. J., and G. R. Peterson, *Introduction to Switching Theory and Logical Design,* 3rd ed. New York: John Wiley, 1981.
2. Mano, M. M., *Digital Design,* 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1991.
3. Roth, C. H., *Fundamentals of Logic Design,* 3rd ed. St. Paul, MN: West Publishing, 1985.
4. Sandige, R. S., *Modern Digital Design*. New York: McGraw-Hill, 1990.
5. Shiva, S. G., *Introduction to Logic Design*. Glenview, II: Scott, Foresman, 1988.
6. Wakerly, J. F., *Digital Design Principles and Practices*. Englewood Cliffs, NJ: Prentice Hall, 1990.
7. Ward, S. A., and R. H. Halstead, Jr., *Computation Structures*. Cambridge, MA: MIT Press, 1990.

4.4 (up to fig. 4.7) ---:) https://github.com/AvinashShrivastav/MATHEMATICS-FOR-COMPUTING

$R1$ and the address is in AR. The write operation can be stated symbolically as follows:

$$\text{Write:} \quad M[AR] \leftarrow R1$$

This causes a transfer of information from $R1$ into the memory word M selected by the address in AR.

4-4 Arithmetic Microoperations

A microoperation is an elementary operation performed with the data stored in registers. The microoperations most often encountered in digital computers are classified into four categories:

1. Register transfer microoperations transfer binary information from one register to another.
2. Arithmetic microoperations perform arithmetic operation on numeric data stored in registers.
3. Logic microoperations perform bit manipulation operations on nonnumeric data stored in registers.
4. Shift microoperations perform shift operations on data stored in registers.

The register transfer microoperation was introduced in Sec. 4-2. This type of microoperation does not change the information content when the binary information moves from the source register to the destination register. The other three types of microoperations change the information content during the transfer. In this section we introduce a set of arithmetic microoperations. In the next two sections we present the logic and shift microoperations.

The basic arithmetic microoperations are addition, subtraction, increment, decrement, and shift. Arithmetic shifts are explained later in conjunction with the shift microoperations. The arithmetic microoperation defined by the statement

$$R3 \leftarrow R1 + R2$$

add microoperation

specifies an *add* microoperation. It states that the contents of register $R1$ are added to the contents of register $R2$ and the sum transferred to register $R3$. To implement this statement with hardware we need three registers and the digital component that performs the addition operation. The other basic arithmetic microoperations are listed in Table 4-3. Subtraction is most often

subtract microoperation

implemented through complementation and addition. Instead of using the minus operator, we can specify the subtraction by the following statement:

$$R3 \leftarrow R1 + \overline{R2} + 1$$

$\overline{R2}$ is the symbol for the 1's complement of $R2$. Adding 1 to the 1's complement produces the 2's complement. Adding the contents of $R1$ to the 2's complement of $R2$ is equivalent to $R1 - R2$.

TABLE 4-3 Arithmetic Microoperations

Symbolic designation	Description
$R3 \leftarrow R1 + R2$	Contents of $R1$ plus $R2$ transferred to $R3$
$R3 \leftarrow R1 - R2$	Contents of $R1$ minus $R2$ transferred to $R3$
$R2 \leftarrow \overline{R2}$	Complement the contents of $R2$ (1's complement)
$R2 \leftarrow \overline{R2} + 1$	2's complement the contents of $R2$ (negate)
$R3 \leftarrow R1 + \overline{R2} + 1$	$R1$ plus the 2's complement of $R2$ (subtraction)
$R1 \leftarrow R1 + 1$	Increment the contents of $R1$ by one
$R1 \leftarrow R1 - 1$	Decrement the contents of $R1$ by one

The increment and decrement microoperations are symbolized by plus-one and minus-one operations, respectively. These microoperations are implemented with a combinational circuit or with a binary up-down counter.

The arithmetic operations of multiply and divide are not listed in Table 4-3. These two operations are valid arithmetic operations but are not included in the basic set of microoperations. The only place where these operations can be considered as microoperations is in a digital system, where they are implemented by means of a combinational circuit. In such a case, the signals that perform these operations propagate through gates, and the result of the operation can be transferred into a destination register by a clock pulse as soon as the output signal propagates through the combinational circuit. In most computers, the multiplication operation is implemented with a sequence of add and shift microoperations. Division is implemented with a sequence of subtract and shift microoperations. To specify the hardware in such a case requires a list of statements that use the basic microoperations of add, subtract, and shift (see Chapter 10).

Binary Adder

To implement the add microoperation with hardware, we need the registers that hold the data and the digital component that performs the arithmetic addition. The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a full-adder (see Fig. 1-17). The digital circuit that

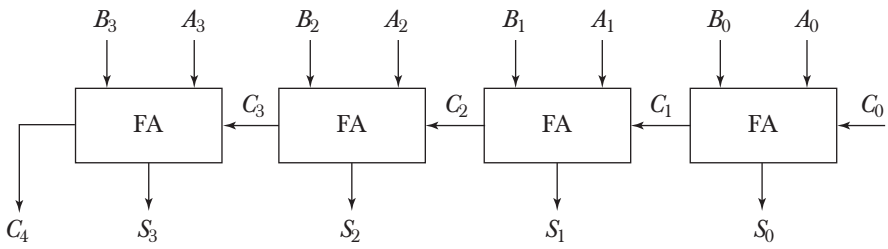


Figure 4-6 4-bit binary adder.

binary adder

full-adder

generates the arithmetic sum of two binary numbers of any length is called a binary adder. The binary adder is constructed with full-adder circuits connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder. Figure 4-6 shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder. The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit. The carries are connected in a chain through the full-adders. The input carry to the binary adder is C_0 and the output carry is C_4. The S outputs of the full-adders generate the required sum bits.

An n-bit binary adder requires n full-adders. The output carry from each full-adder is connected to the input carry of the next-high-order full-adder. The n data bits for the A inputs come from one register (such as $R1$), and the n data bits for the B inputs come from another register (such as $R2$). The sum can be transferred to a third register or to one of the source registers ($R1$ or $R2$), replacing its previous content.

Binary Adder-Subtractor

The subtraction of binary numbers can be done most conveniently by means of complements as discussed in Sec. 3-2. Remember that the subtraction $A - B$ can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pair of bits. The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry.

adder-subtractor

The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder. A 4-bit adder-subtractor circuit is shown in Fig. 4-7. The mode input M controls the operation. When $M = 0$ the circuit is an adder and when $M = 1$ the circuit becomes a subtractor. Each exclusive-OR gate receives input M and one of the inputs of B. When $M = 0$, we have $B \oplus 0 = B$. The full-adders receive the value of B, the input carry is 0, and the circuit performs A plus B. When $M = 1$, we have $B \oplus 1 = B'$ and $C_0 = 1$. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the

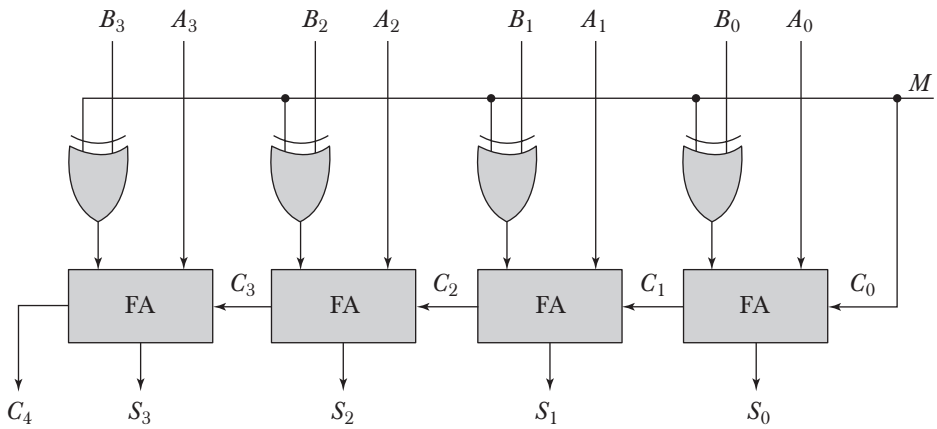


Figure 4-7 4-bit adder-subtractor.

2's complement of B. For unsigned numbers, this gives $A - B$ if $A \geq B$ or the 2's complement of $(B - A)$ if $A < B$. For signed numbers, the result is $A - B$ provided that there is no overflow.

Binary Incrementer

The increment microoperation adds one to a number in a register. For example, if a 4-bit register has a binary value 0110, it will go to 0111 after it is incremented. This microoperation is easily implemented with a binary counter (see Fig. 2-10). Every time the count enable is active, the clock pulse transition increments the content of the register by one. There may be occasions when the increment microoperation must be done with a combinational circuit independent of a particular register. This can be accomplished by means of half-adders (see Fig. 1-16) connected in cascade.

incrementer

The diagram of a 4-bit combinational circuit incrementer is shown in Fig. 4-8. One of the inputs to the least significant half-adder (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented. The output carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder. The circuit receives the four bits from A_0 through A_3, adds one to it, and generates the incremented output in S_0 through S_3. The output carry C_4 will be 1 only after incrementing binary 1111. This also causes outputs S_0 through S_3 to go to 0.

The circuit of Fig. 4-8 can be extended to an n-bit binary incrementer by extending the diagram to include n half-adders. The least significant bit must have one input connected to logic-1. The other inputs receive the number to be incremented or the carry from the previous stage.

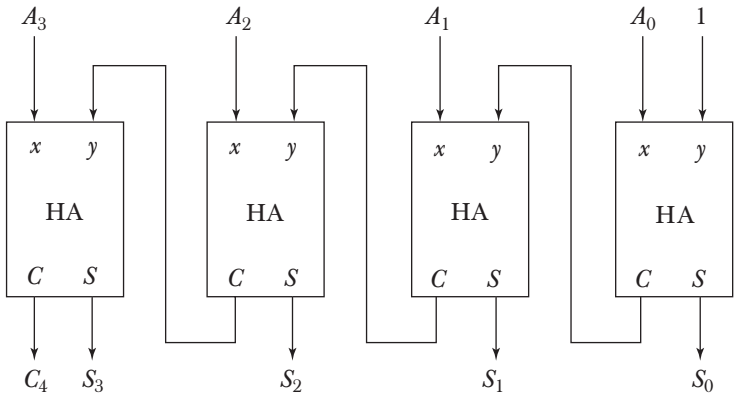


Figure 4-8 4-bit binary incrementer.

Arithmetic Circuit

arithmetic circuit

The arithmetic microoperations listed in Table 4-3 can be implemented in one composite arithmetic circuit. The basic component of an arithmetic circuit is the parallel adder. By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.

The diagram of a 4-bit arithmetic circuit is shown in Fig. 4-9. It has four full-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations. There are two 4-bit inputs A and B and a 4-bit output D. The four inputs from A go directly to the X inputs of the binary adder. Each of the four inputs from B are connected to the data inputs of the multiplexers. The multiplexers data inputs also receive the complement of B. The other two data inputs are connected to logic-0 and logic-1. Logic-0 is a fixed voltage value (0 volts for TTL integrated circuits) and the logic-1 signal can be generated through an inverter whose input is 0. The four multiplexers are controlled by two selection inputs, S_1 and S_0. The input carry C_{in} goes to the carry input of the FA in the least significant position. The other carries a connected from one stage to the next.

input carry

The output of the binary adder is calculated from the following arithmetic sum:

$$D = A + Y + C_{in}$$

where A is the 4-bit binary number at the X inputs and Y is the 4-bit binary number at the Y inputs of the binary adder. C_{in} is the input carry, which can be equal to 0 or 1. Note that the symbol + in the equation above denotes an arithmetic plus. By controlling the value of Y with the two selection inputs S_1 and S_0 and making C_{in} equal to 0 or 1, it is possible to generate the eight arithmetic microoperations listed in Table 4-4.

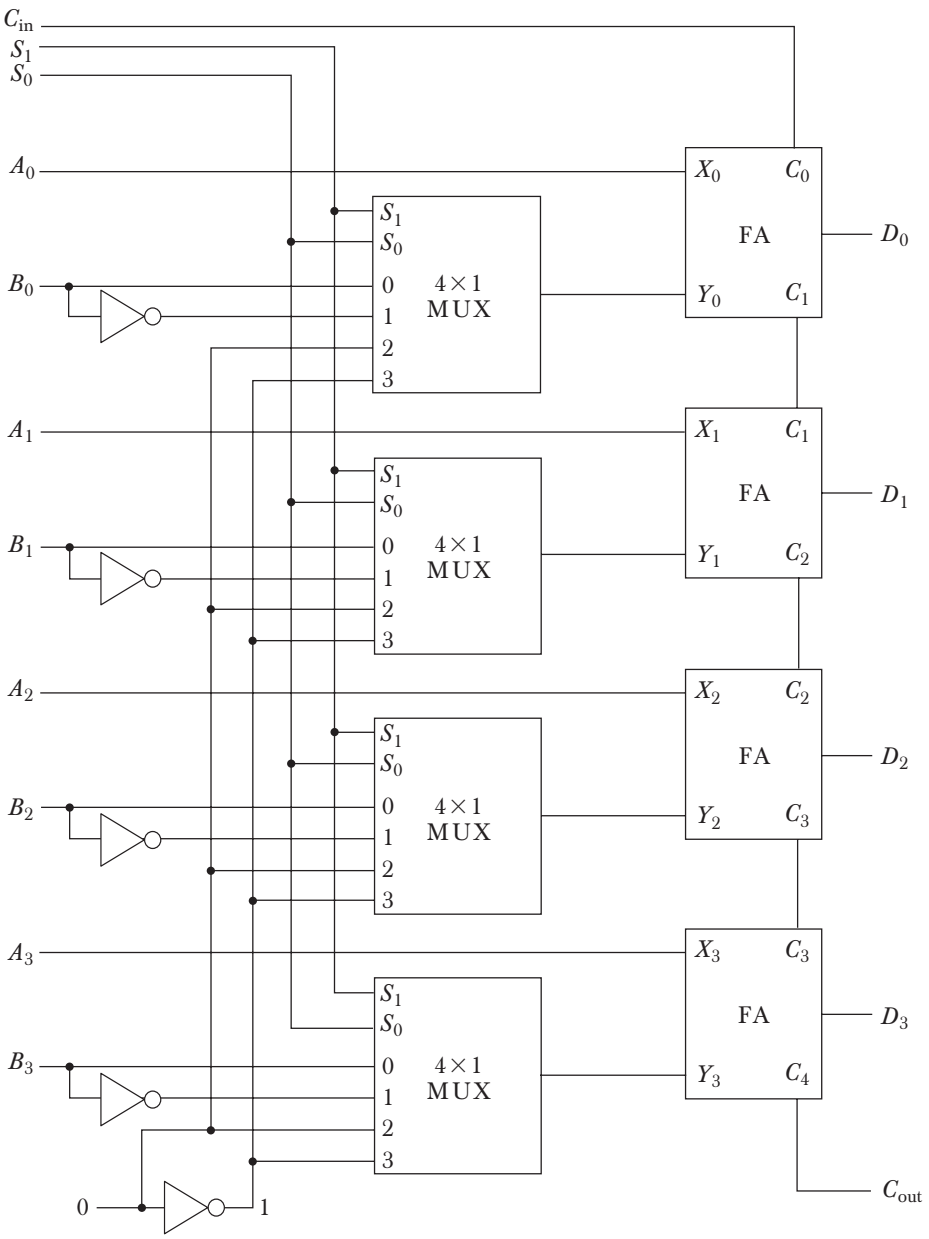


Figure 4-9 4-bit arithmetic circuit.