PART 1
DIGITAL PRINCIPLES

1

NUMBER SYSTEMS AND CODES

Modern computers don't work with decimal numbers. Instead, they process *binary numbers,* groups of 0s and 1s. Why binary numbers? Because electronic devices are most reliable when designed for two-state (binary) operation. This chapter discusses binary numbers and other concepts needed to understand computer operation.

1-1 DECIMAL ODOMETER

René Descartes (1596–1650) said that the way to learn a new subject is to go from the known to the unknown, from the simple to the complex. Let's try it.

The Known

Everyone has seen an odometer (miles indicator) in action. When a car is new, its odometer starts with

00000

After 1 mile the reading becomes

00001

Successive miles produce 00002, 00003, and so on, up to

00009

A familiar thing happens at the end of the tenth mile. When the units wheel turns from 9 back to 0, a tab on this wheel forces the tens wheel to advance by 1. This is why the numbers change to

00010

Reset-and-Carry

The units wheel has reset to 0 and sent a carry to the tens wheel. Let's call this familiar action *reset-and-carry.*

The other wheels also reset and carry. After 999 miles the odometer shows

00999

What does the next mile do? The units wheel resets and carries, the tens wheel resets and carries, the hundreds wheel resets and carries, and the thousands wheel advances by 1, to get

01000

Digits and Strings

The numbers on each odometer wheel are called *digits.* The decimal number system uses ten digits, 0 through 9. In a decimal odometer, each time the units wheel runs out of digits, it resets to 0 and sends a carry to the tens wheel. When the tens wheel runs out of digits, it resets to 0 and sends a carry to the hundreds wheel. And so on with the remaining wheels.

One more point. A *string* is a group of characters (either letters or digits) written one after another. For instance, 734 is a string of 7, 3, and 4. Similarly, 2C8A is a string of 2, C, 8, and A.

1-2 BINARY ODOMETER

Binary means two. The binary number system uses only two digits, 0 and 1. All other digits (2 through 9) are thrown away. In other words, binary numbers are strings of 0s and 1s.

An Unusual Odometer

Visualize an odometer whose wheels have only two digits, 0 and 1. When each wheel turns, it displays 0, then 1, then

back to 0, and the cycle repeats. Because each wheel has only two digits, we call this device a *binary odometer*.

In a car a binary odometer starts with

0000 (zero)

After 1 mile, it indicates

0001 (one)

The next mile forces the units wheel to reset and carry; so the numbers change to

0010 (two)

The third mile results in

0011 (three)

What happens after 4 miles? The units wheel resets and carries, the second wheel resets and carries, and the third wheel advances by 1. This gives

0100 (four)

Successive miles produce

0101 (five)
0110 (six)
0111 (seven)

After 8 miles, the units wheel resets and carries, the second wheel resets and carries, the third wheel resets and carries, and the fourth wheel advances by 1. The result is

1000 (eight)

The ninth mile gives

1001 (nine)

and the tenth mile produces

1010 (ten)

(Try working out a few more readings on your own.)

You should have the idea by now. Each mile advances the units wheel by 1. Whenever the units wheel runs out of digits, it resets and carries. Whenever the second wheel runs out of digits, it resets and carries. And so for the other wheels.

Binary Numbers

A binary odometer displays binary numbers, strings of 0s and 1s. The number 0001 stands for 1, 0010 for 2, 0011 for 3, and so forth. Binary numbers are long when large amounts are involved. For instance, 101010 represents decimal 42. As another example, 111100001111 stands for decimal 3,855.

Computer circuits are like binary odometers; they count and work with binary numbers. Therefore, you have to learn to count with binary numbers, to convert them to decimal numbers, and to do binary arithmetic. Then you will be ready to understand how computers operate.

A final point. When a decimal odometer shows 0036, we can drop the leading 0s and read the number as 36. Similarly, when a binary odometer indicates 0011, we can drop the leading 0s and read the number as 11. With the leading 0s omitted, the binary numbers are 0, 1, 10, 11, 100, 101, and so on. To avoid confusion with decimal numbers, read the binary numbers like this: zero, one, one-zero, one-one, one-zero-zero, one-zero-one, etc.

1-3 NUMBER CODES

People used to count with pebbles. The numbers 1, 2, 3 looked like ●, ●●, ●●●. Larger numbers were worse: seven appeared as ●●●●●●●.

Codes

From the earliest times, people have been creating codes that allow us to think, calculate, and communicate. The decimal numbers are an example of a code (see Table 1-1). It's an old idea now, but at the time it was as revolutionary; 1 stands for ●, 2 for ●●, 3 for ●●●, and so forth.

Table 1-1 also shows the binary code. 1 stands for ●, 10 for ●●, 11 for ●●●, and so on. A binary number and a decimal number are equivalent if each represents the same amount of pebbles. Binary 10 and decimal 2 are equivalent because each represents ●●. Binary 101 and decimal 5 are equivalent because each stands for ●●●●●.

TABLE 1-1. NUMBER CODES

Decimal	Pebbles	Binary
0	None	0
1	●	1
2	●●	10
3	●●●	11
4	●●●●	100
5	●●●●●	101
6	●●●●●●	110
7	●●●●●●●	111
8	●●●●●●●●	1000
9	●●●●●●●●●	1001

Equivalence is the common ground between us and computers; it tells us when we're talking about the same thing. If a computer comes up with a binary answer of 101, equivalence means that the decimal answer is 5. As a start to understanding computers, memorize the binary-decimal equivalences of Table 1-1.

EXAMPLE 1-1

Figure 1-1*a* shows four light-emitting diodes (LEDs). A dark circle means that the LED is off; a light circle means it's on. To read the display, use this code:

(a) (b)

Fig. 1-1 LED display of binary numbers.

LED	Binary
Off	0
On	1

What binary number does Fig. 1-1*a* indicate? Fig. 1-1*b*?

SOLUTION

Figure 1-1*a* shows off-off-on-on. This stands for binary 0011, equivalent to decimal 3.

Figure 1-1*b* is off-on-off-on, decoded as binary 0101 and equivalent to decimal 5.

EXAMPLE 1-2

A binary odometer has four wheels. What are the successive binary numbers?

SOLUTION

As previously discussed, the first eight binary numbers are 0000, 0001, 0010, 0011, 0100, 0101, 0110, and 0111. On the next count, the three wheels on the right reset and carry; the fourth wheel advances by one. So the next eight numbers are 1000, 1001, 1010, 1011, 1100, 1101, 1110, and 1111. The final reading of 1111 is equivalent to decimal 15. The next mile resets all wheels to 0, and the cycle repeats.

Being able to count in binary from 0000 to 1111 is essential for understanding the operation of computers.

TABLE 1-2. BINARY-TO-DECIMAL EQUIVALENCES

Decimal	Binary	Decimal	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Therefore, you should memorize the equivalences of Table 1-2.

1-4 WHY BINARY NUMBERS ARE USED

The word "computer" is misleading because it suggests a machine that can solve only numerical problems. But a computer is more than an automatic adding machine. It can play games, translate languages, draw pictures, and so on. To suggest this broad range of application, a computer is often referred to as a *data processor*.

Program and Data

Data means names, numbers, facts, anything needed to work out a problem. Data goes into a computer, where it is processed or manipulated to get new information. Before it goes into a computer, however, the data must be coded in binary form. The reason was given earlier: a computer's circuits can respond only to binary numbers.

Besides the data, someone has to work out a *program*, a list of instructions telling the computer what to do. These instructions spell out each and every step in the data processing. Like the data, the program must be coded in binary form before it goes into the computer.

So the two things we must input to a computer are the program and the data. These are stored inside the computer before the processing begins. Once the computer run starts, each instruction is executed and the data is processed.

Hardware and Software

The electronic, magnetic, and mechanical devices of a computer are known as *hardware*. Programs are called *software*. Without software, a computer is a pile of "dumb" metal.

An analogy may help. A phonograph is like hardware and records are like software. The phonograph is useless without records. Furthermore, the music you get depends on the record you play. A similar idea applies to computers. A computer is the hardware and programs are the software. The computer is useless without programs. The program stored in the computer determines what the computer will do; change the program and the computer processes the data in a different way.

Transistors

Computers use *integrated circuits* (ICs) with thousands of transistors, either bipolar or MOS. The parameters (β_{dc}, I_{CO}, g_m, etc.) can vary more than 50 percent with temperature change and from one transistor to the next. Yet these computer ICs work remarkably well despite the transistor variations. How is it possible?

The answer is *two-state* design, using only two points on the load line of each transistor. For instance, the common two-state design is the cutoff-saturation approach; each transistor is forced to operate at either cutoff or saturation. When a transistor is cut off or saturated, parameter variations have almost no effect. Because of this, it's possible to design reliable two-state circuits that are almost independent of temperature change and transistor variations.

Transistor Register

Here's an example of two-state design. Figure 1-2 shows a transistor register. (A *register* is a string of devices that store data.) The transistors on the left are cut off because the input base voltages are 0 V. The dark shading symbolizes the cutoff condition. The two transistors on the right have base drives of 5 V.

The transistors operate at either saturation or cutoff. A base voltage of 0 V forces each transistor to cut off, while a base voltage of 5 V drives it into saturation. Because of this two-state action, each transistor stays in a given state until the base voltage switches it to the opposite state.

Another Code

Two-state operation is universal in digital electronics. By deliberate design, all input and output voltages are either low or high. Here's how binary numbers come in: low voltage represents binary 0, and high voltage stands for binary 1. In other words, we use this code:

Voltage	Binary
Low	0
High	1

For instance, the base voltages of Fig. 1-2 are low-low-high-high, or binary 0011. The collector voltages are high-high-low-low, or binary 1100. By changing the base voltages we can store any binary number from 0000 to 1111 (decimal 0 to 15).

Bit

Bit is an abbreviation for binary digit. A binary number like 1100 has 4 bits; 110011 has 6 bits; and 11001100 has 8 bits. Figure 1-2 is a 4-bit register. To store larger binary numbers, it needs more transistors. Add two transistors and you get a 6-bit register. With four more transistors, you'd have an 8-bit register.

Nonsaturated Circuits

Don't get the idea that all two-state circuits switch between cutoff and saturation. When a bipolar transistor is heavily saturated, extra carriers are stored in the base region. If the base voltage suddenly switches from high to low, the transistor cannot come out of saturation until these extra carriers have a chance to leave the base region. The time it takes for these carriers to leave is called the *saturation delay time* t_d. Typically, t_d is in nanoseconds.

In most applications the saturation delay time is too short to matter. But some applications require the fastest possible

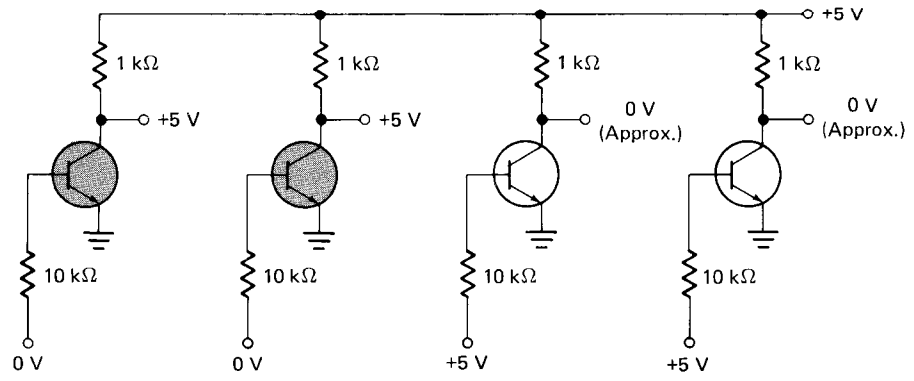


Fig. 1-2 Transistor register.

switching time. To get this maximum speed, designers have come up with circuits that switch from cutoff (or near cutoff) to a higher point on the load line (but short of saturation). These nonsaturated circuits rely on clamping diodes or heavy negative feedback to overcome transistor variations.

Remember this: whether saturated or nonsaturated circuits are used, the transistors switch between distinct points on the load line. This means that all input and output voltages are easily recognized as low or high, binary 0 or binary 1.

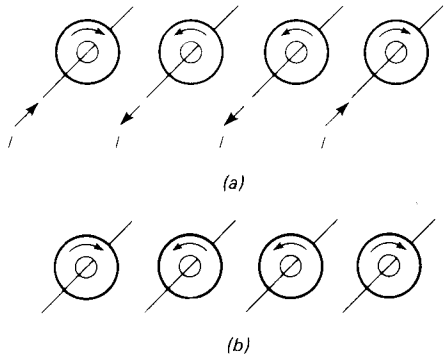


Fig. 1-3 Core register.

Magnetic Cores

Early digital computers used magnetic cores to store data. Figure 1-3*a* shows a 4-bit core register. With the right-hand rule, you can see that conventional current into a wire produces a clockwise flux; reversing the current gives a counterclockwise flux. (The same result is obtained if electron-flow is assumed and the left-hand rule is used.)

The cores have rectangular hysteresis loops; this means that flux remains in a core even though the magnetizing current is removed (see Fig. 1-3*b*). This is why a core register can store binary data indefinitely. For instance, let's use the following code:

Flux	Binary
Counterclockwise	0
Clockwise	1

Then, the core register of Fig. 1-3*b* stores binary 1001, equivalent to decimal 9. By changing the magnetizing currents in Fig. 1-3*a* we can change the stored data.

To store larger binary numbers, add more cores. Two cores added to Fig. 1-3*a* result in a 6-bit register; four more cores give an 8-bit register.

The *memory* is one of the main parts of a computer. Some memories contain thousands of core registers. These registers store the program and data needed to run the computer.

Other Two-State Examples

The simplest example of a two-state device is the on-off switch. When this switch is closed, it represents binary 1; when it's open, it stands for binary 0.

Punched cards are another example of the two-state concept. A hole in a card stands for binary 1, the absence of a hole for binary 0. Using a prearranged code, a card-punch machine with a keyboard can produce a stack of cards containing the program and data needed to run a computer.

Magnetic tape can also store binary numbers. Tape recorders magnetize some points on the tape (binary 1), while leaving other points unmagnetized (binary 0). By a prearranged code, a row of points represents either a coded instruction or data. In this way, a reel of tape can store thousands of binary instructions and data for later use in a computer.

Even the lights on the control panel of a large computer are binary; a light that's on stands for binary 1, and one that's off stands for binary 0. In a 16-bit computer, for instance, a row of 16 lights allows the operator to see the binary contents in different computer registers. The operator can then monitor the overall operation and, when necessary, troubleshoot.

In summary, switches, transistors, cores, cards, tape, lights, and almost all other devices used with computers are based on two-state operation. This is why we are forced to use binary numbers when analyzing computer action.

EXAMPLE 1-3

Figure 1-4 shows a strip of magnetic tape. The black circles are magnetized points and the white circles unmagnetized points. What binary number does each horizontal row represent?

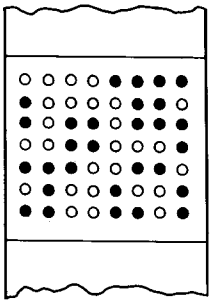


Fig. 1-4 Binary numbers on magnetic tape.

SOLUTION

The tape stores these binary numbers:

Row 1	00001111	Row 5	11100110
Row 2	10000110	Row 6	01001001
Row 3	10110111	Row 7	11001101
Row 4	00110001		

(*Note:* these binary numbers may represent either coded instructions or data.)

A string of 8 bits is called a *byte*. In this example, the magnetic tape stores 7 bytes. The first byte (row 1) is 00001111. The second byte (row 2) is 10000110. The third byte is 10110111. And so on.

A byte is the basic unit of data in computers. Most computers process data in strings of 8 bits or some multiple (16, 24, 32, and so on). Likewise, the memory stores data in strings of 8 bits or some multiple of 8 bits.

1-5 BINARY-TO-DECIMAL CONVERSION

You already know how to count to 15 using binary numbers. The next thing to learn is how to convert larger binary numbers to their decimal equivalents.

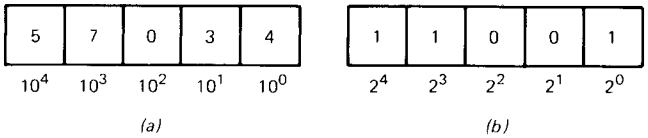


Fig. 1-5 (*a*) Decimal weights; (*b*) binary weights.

Decimal Weights

The decimal number system is an example of positional notation; each digit position has a *weight* or value. With decimal numbers the weights are units, tens, hundreds, thousands, and so on. The sum of all digits multiplied by their weights gives the total amount being represented.

For instance, Fig. 1-5*a* illustrates a decimal odometer. Below each digit is its weight. The digit on the right has a weight of 10^0 (units), the second digit has a weight of 10^1 (tens), the third digit a weight of 10^2 (hundreds), and so forth. The sum of all units multiplied by their weights is

$$(5 \times 10^4) + (7 \times 10^3) + (0 \times 10^2) + (3 \times 10^1) + (4 \times 10^0) = 50{,}000 + 7000 + 0 + 30 + 4 = 57{,}034$$

Binary Weights

Positional notation is also used with binary numbers because each digit position has a weight. Since only two digits are used, the weights are powers of 2 instead of 10. As shown in the binary odometer of Fig. 1-5*b*, these weights are 2^0 (units), 2^1 (twos), 2^2 (fours), 2^3 (eights), and 2^4 (sixteens). If longer binary numbers are involved, the weights continue in ascending powers of 2.

The decimal equivalent of a binary number equals the sum of all binary digits multiplied by their weights. For instance, the binary reading of Fig. 1-5*b* has a decimal equivalent of

$$(1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 16 + 8 + 0 + 0 + 1 = 25$$

Binary 11001 is therefore equivalent to decimal 25.

As another example, the byte 11001100 converts to decimal as follows:

$$(1 \times 2^7) + (1 \times 2^6) + (0 \times 2^5) + (0 \times 2^4) + (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 128 + 64 + 0 + 0 + 8 + 4 + 0 + 0 = 204$$

So, binary 11001100 is equivalent to decimal 204.

Fast and Easy Conversion

Here's a streamlined way to convert a binary number to its decimal equivalent:

1. Write the binary number.
2. Write the weights 1, 2, 4, 8, . . . , under the binary digits.
3. Cross out any weight under a 0.
4. Add the remaining weights.

For instance, binary 1101 converts to decimal as follows:

1.	1 1 0 1	(Write binary number)
2.	8 4 2 1	(Write weights)
3.	8 4 ~~2~~ 1	(Cross out weights)
4.	8 + 4 + 0 + 1 = 13	(Add weights)

You can compress the steps even further:

1 1 0 1	(Step 1)
8 4 ~~2~~ 1 → 13	(Steps 2 to 4)

As another example, here's the conversion of binary 1110101 in compressed form:

1 1 1 0 1 0 1
64 32 16 ~~8~~ 4 ~~2~~ 1 → 117

Base or Radix

The *base* or *radix* of a number system equals the number of digits it has. Decimal numbers have a base of 10 because digits 0 through 9 are used. Binary numbers have a base of 2 because only the digits 0 and 1 are used. (In terms of an odometer, the base or radix is the number of digits on each wheel.)

A subscript attached to a number indicates the base of the number. 100_2 means binary 100. On the other hand, 100_{10} stands for decimal 100. Subscripts help clarify equations where binary and decimal numbers are mixed. For instance, the last two examples of binary-to-decimal conversion can be written like this:

$$1101_2 = 13_{10}$$
$$1110101_2 = 117_{10}$$

In this book we will use subscripts when necessary for clarity.

1-6 MICROPROCESSORS

What is inside a computer? What is a microprocessor? What is a microcomputer?

Computer

The five main sections of a computer are input, memory, arithmetic and logic, control, and output. Here is a brief description of each.

Input This consists of all the circuits needed to get programs and data into the computer. In some computers the input section includes a typewriter keyboard that converts letters and numbers into strings of binary data.

Memory This stores the program and data before the computer run begins. It also can store partial solutions during a computer run, similar to the way we use a scratchpad while working out a problem.

Control This is the computer's center of gravity, analogous to the conscious part of the mind. The control section directs the operation of all other sections. Like the conductor of an orchestra, it tells the other sections what to do and when to do it.

Arithmetic and logic This is the number-crunching section of the machine. It can also make logical decisions. With control telling it what to do and with memory feeding it data, the arithmetic-logic unit (ALU) grinds out answers to number and logic problems.

Output This passes answers and other processed data to the outside world. The output section usually includes a video display to allow the user to see the processed data.

Microprocessor

The control section and the ALU are often combined physically into a single unit called the *central processing unit* (CPU). Furthermore, it's convenient to combine the input and output sections into a single unit called the *input-output* (I/O) *unit*. In earlier computers, the CPU, memory, and I/O unit filled an entire room.

With the advent of integrated circuits, the CPU, memory, and I/O unit have shrunk dramatically. Nowadays the CPU can be fabricated on a single semiconductor chip called a *microprocessor*. In other words, a microprocessor is nothing more than a CPU on a chip.

Likewise, the I/O circuits and memory can be fabricated on chips. In this way, the computer circuits that once filled a room now fit on a few chips.

Microcomputer

As the name implies, a *microcomputer* is a small computer. More specifically, a microcomputer is a computer that uses a microprocessor for its CPU. The typical microcomputer has three kinds of chips: microprocessor (usually one chip), memory (several chips), and I/O (one or more chips).

If a small memory is acceptable, a manufacturer can fabricate all computer circuits on a single chip. For instance, the 8048 from Intel Corporation is a one-chip microcomputer with an 8-bit CPU, 1,088 bytes of memory, and 27 I/O lines.

Powers of 2

Microprocessor design started with 4-bit devices, then evolved to 8- and 16-bit devices. In our later discussions of microprocessors, powers of 2 keep coming up because of the binary nature of computers. For this reason, you should study Table 1-3. It lists the powers of 2 encountered in microcomputer analysis. As shown, the abbreviation K stands for 1,024 (approximately 1,000).† Therefore, 1K means 1,024, 2K stands for 2,048, 4K for 4,096, and so on.

Most personal microcomputers have 640K (or greater) memories that can store 655,360 bytes (or more).

TABLE 1-3. POWERS OF 2

Powers of 2	Decimal equivalent	Abbreviation
2^0	1	
2^1	2	
2^2	4	
2^3	8	
2^4	16	
2^5	32	
2^6	64	
2^7	128	
2^8	256	
2^9	512	
2^{10}	1,024	1K
2^{11}	2,048	2K
2^{12}	4,096	4K
2^{13}	8,192	8K
2^{14}	16,384	16K
2^{15}	32,768	32K
2^{16}	65,536	64K

† The abbreviations 1K, 2K, and so on, became established before K- for *kilo-* was in common use. Retaining the capital K serves as a useful reminder that K only approximates 1,000.

1-7 DECIMAL-TO-BINARY CONVERSION

Next, you need to know how to convert from decimal to binary. After you know how it's done, you will be able to understand how circuits can be built to convert decimal numbers into binary numbers.

Double-Dabble

Double-dabble is a way of converting any decimal number to its binary equivalent. It requires successive division by 2, writing down each quotient and its remainder. The remainders are the binary equivalent of the decimal number. The only way to understand the method is to go through an example, step by step.

Here is how to convert decimal 13 to its binary equivalent.
Step 1. Divide 13 by 2, writing your work like this:

$$\begin{array}{rl} 6 & 1 \rightarrow \text{(first remainder)} \\ 2\overline{)13} & \end{array}$$

The quotient is 6 with a remainder of 1.
Step 2. Divide 6 by 2 to get

$$\begin{array}{rl} 3 & 0 \rightarrow \text{(second remainder)} \\ 2\overline{)6} & 1 \\ 2\overline{)13} & \end{array}$$

This division gives 3 with a remainder of 0.
Step 3. Again you divide by 2:

$$\begin{array}{rl} 1 & 1 \rightarrow \text{(third remainder)} \\ 2\overline{)3} & 0 \\ 2\overline{)6} & 1 \\ 2\overline{)13} & \end{array}$$

Here you get a quotient of 1 and a remainder of 1.
Step 4. One more division by 2 gives

$$\begin{array}{rl} & \text{Read down} \\ 0 & 1 \\ 2\overline{)1} & 1 \\ 2\overline{)3} & 0 \\ 2\overline{)6} & 1 \\ 2\overline{)13} & \end{array} \quad \downarrow$$

In this final division, 2 does not divide into 1; therefore, the quotient is 0 with a remainder of 1.

Whenever you arrive at a quotient of 0 with a remainder of 1, the conversion is finished. The remainders when read downward give the binary equivalent. In this example, binary 1101 is equivalent to decimal 13.

Double-dabble works with any decimal number. Progressively divide by 2, writing each quotient and its remainder. When you reach a quotient of 0 and a remainder of 1, you are finished; the remainders read downward are the binary equivalent of the decimal number.

Streamlined Double-Dabble

There's no need to keep writing down 2 before each division because you're always dividing by 2. From now on, here's how to show the conversion of decimal 13 to its binary equivalent:

$$\begin{array}{rl} 0 & 1 \\ \overline{)1} & 1 \\ \overline{)3} & 0 \\ \overline{)6} & 1 \\ 2\overline{)13} & \end{array} \quad \downarrow$$

EXAMPLE 1-4

Convert decimal 23 to binary.

SOLUTION

The first step in the conversion looks like this:

$$\begin{array}{rl} 11 & 1 \\ 2\overline{)23} & \end{array}$$

After all divisions, the finished work looks like this:

$$\begin{array}{rl} 0 & 1 \\ \overline{)1} & 0 \\ \overline{)2} & 1 \\ \overline{)5} & 1 \\ \overline{)11} & 1 \\ 2\overline{)23} & \end{array} \quad \downarrow$$

This says that binary 10111 is equivalent to decimal 23.

1-8 HEXADECIMAL NUMBERS

Hexadecimal numbers are extensively used in microprocessor work. To begin with, they are much shorter than binary numbers. This makes them easy to write and remember. Furthermore, you can mentally convert them to binary form whenever necessary.

An Unusual Odometer

Hexadecimal means 16. The hexadecimal number system has a base or radix of 16. This means that it uses 16 digits to represent all numbers. The digits are 0 through 9, and A through F as follows: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. Hexadecimal numbers are strings of these digits like 8A5, 4CF7, and EC58.

An easy way to understand hexadecimal numbers is to visualize a hexadecimal odometer. Each wheel has 16 digits on its circumference. As it turns, it displays 0 through 9 as before. But then, instead of resetting, it goes on to display A, B, C, D, E, and F.

The idea of reset and carry applies to a hexadecimal odometer. When a wheel turns from F back to 0, it forces the next higher wheel to advance by 1. In other words, when a wheel runs out of hexadecimal digits, it resets and carries.

If used in a car, a hexadecimal odometer would count as follows. When the car is new, the odometer shows all 0s:

0000 (zero)

The next 9 miles produce readings of

0001 (one)
0002 (two)
0003 (three)
0004 (four)
0005 (five)
0006 (six)
0007 (seven)
0008 (eight)
0009 (nine)

The next 6 miles give

000A (ten)
000B (eleven)
000C (twelve)
000D (thirteen)
000E (fourteen)
000F (fifteen)

At this point the least significant wheel has run out of digits. Therefore, the next mile forces a reset-and-carry to get

0010 (sixteen)

The next 15 miles produce these readings: 0011, 0012, 0013, 0014, 0015, 0016, 0017, 0018, 0019, 001A, 001B, 001C, 001D, 001E, and 001F. Once again, the least significant wheel has run out of digits. So, the next mile results in a reset-and-carry:

0020 (thirty-two)

Subsequent readings are 0021, 0022, 0023, 0024, 0025, 0026, 0027, 0028, 0029, 002A, 002B, 002C, 002D, 002E, and 002F.

You should have the idea by now. Each mile advances the least significant wheel by 1. When this wheel runs out of hexadecimal digits, it resets and carries. And so on for the other wheels. For instance, if the odometer reading is

835F

the next reading is 8360. As another example, given

5FFF

the next hexadecimal number is 6000.

Equivalences

Table 1-4 shows the equivalences between hexadecimal, binary, and decimal digits. Memorize this table. It's essential that you be able to convert instantly from one system to another.

TABLE 1-4. EQUIVALENCES

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

1-9 HEXADECIMAL-BINARY CONVERSIONS

After you know the equivalences of Table 1-4, you can mentally convert any hexadecimal string to its binary equivalent and vice versa.

Hexadecimal to Binary

To convert a hexadecimal number to a binary number, convert each hexadecimal digit to its 4-bit equivalent, using Table 1-4. For instance, here's how 9AF converts to binary:

9	A	F
↓	↓	↓
1001	1010	1111

As another example, C5E2 converts like this:

C	5	E	2
↓	↓	↓	↓
1100	0101	1110	0010

Incidentally, for easy reading it's common practice to leave a space between the 4-bit strings. For example, instead of writing

$$C5E2_{16} = 1100010111100010_2$$

we can write

$$C5E2_{16} = 1100\ 0101\ 1110\ 0010_2$$

Binary to Hexadecimal

To convert in the opposite direction, from binary to hexadecimal, you again use Table 1-4. Here are two examples. The byte 1000 1100 converts as follows:

1000	1100
↓	↓
8	C

The 16-bit number 1110 1000 1101 0110 converts like this:

1110	1000	1101	0110
↓	↓	↓	↓
E	8	D	6

In both these conversions, we start with a binary number and wind up with the equivalent hexadecimal number.

EXAMPLE 1-5

Solve the following equation for x:

$$x_{16} = 1111\ 1111\ 1111\ 1111_2$$

SOLUTION

This is the same as asking for the hexadecimal equivalent of binary 1111 1111 1111 1111. Since hexadecimal F is equivalent to 1111, x = FFFF. Therefore,

$$FFFF_{16} = 1111\ 1111\ 1111\ 1111_2$$

EXAMPLE 1-6

As mentioned earlier, the memory contains thousands of registers (core or semiconductor) that store the program and data needed for a computer run. These memory registers are known as *memory locations*. A typical microcomputer may have up to 65,536 memory locations, each storing 1 byte.

Suppose the first 16 memory locations contain these bytes:

0011 1100
1100 1101
0101 0111
0010 1000
1111 0001
0010 1010
1101 0100
0100 0000
0111 0111
1100 0011
1000 0100
0010 1000
0010 0001
0011 1010
0011 1110
0001 1111

Convert these bytes to their hexadecimal equivalents.

SOLUTION

Here are the stored bytes and their hexadecimal equivalents:

Memory Contents	Hex Equivalents
0011 1100	3C
1100 1101	CD
0101 0111	57
0010 1000	28
1111 0001	F1

0010 1010	2A
1101 0100	D4
0100 0000	40
0111 0111	77
1100 0011	C3
1000 0100	84
0010 1000	28
0010 0001	21
0011 1010	3A
0011 1110	3E
0001 1111	1F

What's the point of this example? When talking about the contents of a computer memory, we can use either binary numbers or hexadecimal numbers. For instance, we can say that the first memory location contains 0011 1100, or we can say that it contains 3C. Either string gives the same information. But notice how much easier it is to say, write, and think 3C than it is to say, write, and think 0011 1100. In other words, hexadecimal strings are much easier for people to work with. This is why everybody working with microprocessors uses hexadecimal notation to represent particular bytes.

What we have just done is known as *chunking*, replacing longer strings of data with shorter ones. At the first memory location we chunk the digits 0011 1100 into 3C. At the second memory location we chunk the digits 1100 1101 into CD, and so on.

EXAMPLE 1-7

The typical microcomputer has a typewriter keyboard that allows you to enter programs and data; a video screen displays answers and other information.

Suppose the video screen of a microcomputer displays the hexadecimal contents of the first eight memory locations as

A7
28
C3
19
5A
4D
2C
F8

What are the binary contents of the memory locations?

SOLUTION

Convert from hexadecimal to binary to get

1010 0111
0010 1000
1100 0011
0001 1001
0101 1010
0100 1101
0010 1100
1111 1000

The first memory location stores the byte 1010 0111, the second memory location stores the byte 0010 1000, and so on.

This example emphasizes a widespread industrial practice. Microcomputers are programmed to display chunked data, often hexadecimal. The user is expected to know hexadecimal-binary conversions. In other words, a computer manufacturer assumes that you know that A7 represents 1010 0111, 28 stands for 0010 1000, and so on.

One more point. Notice that each memory location in this example stores 1 byte. This is typical of first-generation microcomputers because they use 8-bit microprocessors.

1-10 HEXADECIMAL-TO-DECIMAL CONVERSION

You often need to convert a hexadecimal number to its decimal equivalent. This section discusses methods for doing it.

Hexadecimal to Binary to Decimal

One way to convert from hexadecimal to decimal is the two-step method of converting from hexadecimal to binary and then from binary to decimal. For instance, here's how to convert hexadecimal 3C to its decimal equivalent.

Step 1. Convert 3C to its binary equivalent:

3	C
↓	↓
0011	1100

Step 2. Convert 0011 1100 to its decimal equivalent:

0	0	1	1	1	1	0	0	
~~128~~	~~64~~	32	16	8	4	~~2~~	~~1~~	→ 60

Therefore, decimal 60 is equivalent to hexadecimal 3C. As an equation,

$$3C_{16} = 0011\ 1100_2 = 60_{10}$$

Positional-Notation Method

Positional notation is also used with hexadecimal numbers because each digit position has a weight. Since 16 digits are used, the weights are the powers of 16. As shown in

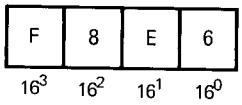


Fig. 1-6 Hexadecimal weights.

the hexadecimal odometer of Fig. 1-6, the weights are 16^0, 16^1, 16^2, and 16^3. If longer hexadecimal numbers are involved, the weights continue in ascending powers of 16.

The decimal equivalent of a hexadecimal string equals the sum of all hexadecimal digits multiplied by their weights. (In processing hexadecimal digits A through F, use 10 through 15.) For instance, the hexadecimal reading of Fig. 1-6 has a decimal equivalent of

$$\begin{aligned}(F \times 16^3) + (8 \times 16^2) + (E \times 16^1) + (6 \times 16^0) \\ = (15 \times 16^3) + (8 \times 16^2) + (14 \times 16^1) + (6 \times 16^0) \\ = 61{,}440 + 2{,}048 + 224 + 6 \\ = 63{,}718\end{aligned}$$

In other words,

$$F8E6_{16} = 63{,}718_{10}$$

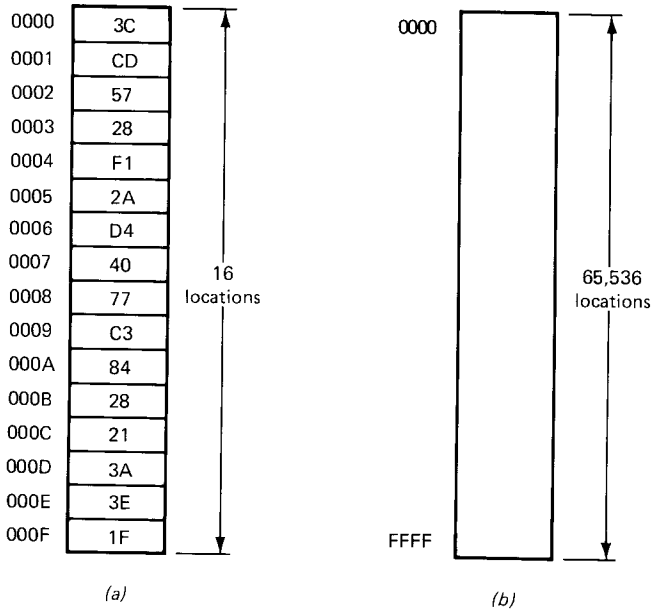


Fig. 1-7 (*a*) First 16 words in memory; (*b*) 64K memory.

Memory Locations and Addresses

If a certain microcomputer has 64K memory, meaning 65,536 memory locations, each is able to store 1 byte. The different memory locations are identified by hexadecimal numbers called *addresses*. For instance, Fig. 1-7*a* shows the first 16 memory locations; their addresses are from 0000 to 000F.

The address of a memory location is different from its stored contents, just as a house address is different from the people living in the house. Figure 1-7*a* emphasizes the point. At address 0000 the stored contents are 3C (equivalent to 0011 1100). At address 0001 the stored contents are CD, at address 0002 the stored contents are 57, and so on.

Figure 1-7*b* shows how to visualize a 64K memory. The first address is 0000, and the last is FFFF.

Table of Binary-Hexadecimal-Decimal Equivalents

A 64K memory has 65,536 hexadecimal addresses from 0000 to FFFF. The equivalent binary addresses are from

0000 0000 0000 0000

to

1111 1111 1111 1111

The first 8 bits are called the *upper byte* (UB); the second 8 bits are the *lower byte* (LB). If you have to do a lot of binary-hexadecimal-decimal conversions, use the table of equivalents in Appendix 2, which shows all the values for a 64K memory.

Appendix 2 has four headings: binary, hexadecimal, UB decimal, and LB decimal. Given a 16-bit address, you convert the upper byte to its decimal equivalent (UB decimal), the lower byte to its decimal equivalent (LB decimal), and then add the two decimal equivalents. For instance, suppose you want to convert

1101 0111 1010 0010

to its decimal equivalent. The upper byte is 1101 0111, or hexadecimal D7; the lower byte is 1010 0010, or A2. Using Appendix 2, find D7 and its UB decimal equivalent

$$D7 \rightarrow 55{,}040$$

Next, find A2 and its LB decimal equivalent

$$A2 \rightarrow 162$$

Add the UB and LB decimal equivalents to get

$$55{,}040 + 162 = 55{,}202$$

This is the decimal equivalent of hexadecimal D7A2 or binary 1101 0111 1010 0010.

Once familiar with Appendix 2, you will find it enormously helpful. It is faster, more accurate, and less tiring than other methods. The only calculation required is adding the UB and LB decimal, easily done mentally, with pencil and paper, or if necessary, on a calculator. Furthermore, if you are interested in converting only the lower byte, no calculation is required, as shown in the next example.

EXAMPLE 1-8

Convert hexadecimal 7E to its decimal equivalent.

SOLUTION

When converting only a single byte, all you are dealing with is the lower byte. With Appendix 2, look up 7E and its LB decimal equivalent to get

$$7E \rightarrow 126$$

In other words, Appendix 2 can be used to convert single bytes to their decimal equivalents (LB decimal) or double bytes to their decimal equivalents (UB decimal + LB decimal).

1-11 DECIMAL-TO-HEXADECIMAL CONVERSION

One way to perform decimal-to-hexadecimal conversion is to go from decimal to binary then to hexadecimal. Another way is *hex-dabble*. The idea is to divide successively by 16, writing down the remainders. (Hex-dabble is like double-dabble except that 16 is used for the divisor instead of 2.)

Here's an example of how to convert decimal 2,479 into hexadecimal form. The first division is

```
        154    15  F
16 ) 2,479
```

The next step is

```
          9    10  A
    ) 154      15  F
16 ) 2,479
```

The final step is

```
                          Read
                          down
          0    9  9        |
        ) 9    10 A        |
    ) 154      15 F        |
16 ) 2,479                 |
                           v
```

Notice how similar hex-dabble is to double-dabble. Also, remainders greater than 9 have to be changed to hexadecimal digits (10 becomes A, 15 becomes F, etc.).

If you prefer, use Appendix 2 to look up the decimal-hexadecimal equivalents. The next two examples show how.

EXAMPLE 1-9

Convert decimal 141 to hexadecimal.

SOLUTION

Whenever the decimal number is between 0 and 255, all you have to do is look up the decimal number and its hexadecimal equivalent. With Appendix 2, you can see at a glance that

$$8D \leftarrow 141$$

EXAMPLE 1-10

Convert decimal 36,020 to its hexadecimal equivalent.

SOLUTION

If the decimal number is between 256 and 65,535, you need to proceed as follows. First, locate the largest UB decimal that is less than 36,020. In Appendix 2, the largest UB decimal is

$$\text{UB decimal} = 35{,}840$$

which has a hexadecimal equivalent of

$$8C \leftarrow 35{,}840$$

This is the upper byte.

Next, subtract the UB decimal from the original decimal number:

$$36{,}020 - 35{,}840 = 180$$

The difference 180 has a hexadecimal equivalent

$$B4 \leftarrow 180$$

This is the lower byte.

By combining the upper and lower bytes, we get the complete answer: 8CB4. This is the hexadecimal equivalent of 36,020.

After a little practice, you will find Appendix 2 to be one of the fastest methods of decimal-hexadecimal conversion.

1-12 BCD NUMBERS

A *nibble* is a string of 4 bits. *Binary-coded-decimal* (BCD) numbers express each decimal digit as a nibble. For instance, decimal 2,945 converts to a BCD number as follows:

2	9	4	5
↓	↓	↓	↓
0010	1001	0100	0101

As you see, each decimal digit is coded as a nibble.

Here's another example: $9{,}863_{10}$ converts like this:

9	8	6	3
↓	↓	↓	↓
1001	1000	0110	0011

Therefore, 1001 1000 0110 0011 is the BCD equivalent of $9{,}863_{10}$.

The reverse conversion is similar. For instance, 0010 1000 0111 0100 converts as follows:

0010	1000	0111	0100
↓	↓	↓	↓
2	8	7	4

Applications

BCD numbers are useful wherever decimal information is transferred into or out of a digital system. The circuits inside pocket calculators, for example, can process BCD numbers because you enter decimal numbers through the keyboard and see decimal answers on the LED or liquid-crystal display. Other examples of BCD systems are electronic counters, digital voltmeters, and digital clocks; their circuits can work with BCD numbers.

BCD Computers

BCD numbers have limited value in computers. A few early computers processed BCD numbers but were slower and more complicated than binary computers. As previously mentioned, a computer is more than a number cruncher because it must handle names and other nonnumeric data. In other words, a modern computer must be able to process *alphanumerics* (alphabet letters, numbers, and other symbols). This why modern computers have CPUs that process binary numbers rather than BCD numbers.

Comparison of Number Systems

Table 1-5 shows the four number systems we have discussed. Each number system uses strings of digits to represent quantity. Above 9, equivalent strings appear different. For instance, decimal string 128, hexadecimal string 80, binary string 1000 0000, and BCD string 0001 0010 1000 are equivalent because they represent the same number of pebbles.

Machines have to use long strings of binary or BCD numbers, but people prefer to chunk the data in either decimal or hexadecimal form. As long as we know how to convert from one number system to the next, we can always get back to the ultimate meaning, which is the number of pebbles being represented.

TABLE 1-5. NUMBER SYSTEMS

Decimal	Hexadecimal	Binary	BCD
0	0	0000 0000	0000 0000 0000
1	1	0000 0001	0000 0000 0001
2	2	0000 0010	0000 0000 0010
3	3	0000 0011	0000 0000 0011
4	4	0000 0100	0000 0000 0100
5	5	0000 0101	0000 0000 0101
6	6	0000 0110	0000 0000 0110
7	7	0000 0111	0000 0000 0111
8	8	0000 1000	0000 0000 1000
9	9	0000 1001	0000 0000 1001
10	A	0000 1010	0000 0001 0000
11	B	0000 1011	0000 0001 0001
12	C	0000 1100	0000 0001 0010
13	D	0000 1101	0000 0001 0011
14	E	0000 1110	0000 0001 0100
15	F	0000 1111	0000 0001 0101
16	10	0001 0000	0000 0001 0110
32	20	0010 0000	0000 0011 0010
64	40	0100 0000	0000 0110 0100
128	80	1000 0000	0001 0010 1000
255	FF	1111 1111	0010 0101 0101

1-13 THE ASCII CODE

To get information into and out of a computer, we need to use numbers, letters, and other symbols. This implies some kind of alphanumeric code for the I/O unit of a computer. At one time, every manufacturer had a different code, which led to all kinds of confusion. Eventually, industry settled on an input-output code known as the *American Standard Code for Information Interchange* (abbreviated ASCII). This code allows manufacturers to standardize I/O hardware such as keyboards, printers, video displays, and so on.

The ASCII (pronounced *ask'-ee*) code is a 7-bit code whose format (arrangement) is

$$X_6X_5X_4X_3X_2X_1X_0$$

where each X is a 0 or a 1. For instance, the letter A is coded as

1000001

Sometimes, a space is inserted for easier reading:

100 0001

TABLE 1-6. THE ASCII CODE

$X_3X_2X_1X_0$	$X_6X_5X_4$					
	010	**011**	**100**	**101**	**110**	**111**
0000	SP	0	@	P		p
0001	!	1	A	Q	a	q
0010	"	2	B	R	b	r
0011	#	3	C	S	c	s
0100	$	4	D	T	d	t
0101	%	5	E	U	e	u
0110	&	6	F	V	f	v
0111	'	7	G	W	g	w
1000	(	8	H	X	h	x
1001	)	9	I	Y	i	y
1010	*	:	J	Z	j	z
1011	+	;	K		k	
1100	,	<	L		l	
1101	−	=	M		m	
1110	•	>	N		n	
1111	/	?	O		o	

Table 1-6 shows the ASCII code. Read the table the same as a graph. For instance, the letter A has an $X_6X_5X_4$ of 100 and an $X_3X_2X_1X_0$ of 0001. Therefore, its ASCII code is

100 0001 (A)

Table 1-6 includes the ASCII code for lowercase letters. The letter a is coded as

110 0001 (a)

More examples are

110 0010 (b)
110 0011 (c)
110 0100 (d)

and so on.

Also look at the punctuation and mathematical symbols. Some examples are

010 0100 ($)
010 1011 (+)
011 1101 (=)

In Table 1-6, SP stands for space (blank). Hitting the space bar of an ASCII keyboard sends this into a microcomputer:

010 0000 (space)

EXAMPLE 1-11

With an ASCII keyboard, each keystroke produces the ASCII equivalent of the designated character. Suppose you type

PRINT X

What is the output of an ASCII keyboard?

SOLUTION

P (101 0000), R (101 0010), I (100 1001), N (100 1110), T (101 0100), space (010 0000), X (101 1000).

GLOSSARY

address Each memory location has an address, analogous to a house address. Using addresses, we can tell the computer where desired data is stored.

alphanumeric Letters, numbers, and other symbols.

base The number of digits (basic symbols) in a number system. Decimal has a base of 10, binary a base of 2, and hexadecimal a base of 16. Also called the radix.

bit An abbreviation for binary digit.

byte A string of 8 bits. The byte is the basic unit of binary information. Most computers process data with a length of 8 bits or some multiple of 8 bits.

central processing unit The control section and the arithmetic-logic section. Abbreviated CPU.

chip An integrated circuit.

chunking Replacing a longer string by a shorter one.

data Names, numbers, and any other information needed to solve a problem.

digital Pertains to anything in the form of digits, for example, digital data.

hardware The electronic, magnetic, and mechanical devices used in a computer.

hexadecimal A number system with a base of 16. Hexadecimal numbers are used in microprocessor work.

input-output Abbreviated I/O. The input and output sections of a computer are often lumped into one unit known as the I/O unit.

microcomputer A computer that uses a microprocessor for its central processing unit (CPU).

microprocessor A CPU on a chip. It contains the control and arithmetic-logic sections. Sometimes abbreviated MPU (microprocessor unit).

nibble A string of 4 bits. Half of a byte.

program A sequence of instructions that tells the computer how to process the data. Also known as software.

register A group of electronic, magnetic, or mechanical devices that store digital data.

software Programs.

string A group of digits or other symbols.

SELF-TESTING REVIEW

Read each of the following and provide the missing words. Answers appear at the beginning of the next question.

1. Binary means _____. Binary numbers have a base of 2. The digits used in a binary number system are _____ and _____.
2. (*two; 0, 1*) Names, numbers, and other information needed to solve a problem are called _____. The _____ is a sequence of instructions that tells the computer how to process the data.
3. (*data, program*) Computer ICs work reliably because they are based on _____ design. When a transistor is cut off or saturated, transistor _____ have almost no effect.
4. (*two-state, variations*) A _____ is a group of devices that store digital data. _____ is an abbreviation for binary digit. A byte is a string of _____ bits.
5. (*register, Bit, 8*) The control and arithmetic-logic sections are called the _____ (CPU). A microprocessor is a CPU on a chip. A microcomputer is a computer that uses a _____ for its CPU.
6. (*central processing unit, microprocessor*) The abbreviation K indicates units of approximately 1,000 or precisely 1,024. Therefore, 1K means 1,024, 2K means 2,048, 4K means _____, and 64K means _____.
7. (*4,096, 65,536*) The hexadecimal number system is widely used in analyzing and programming _____. The hexadecimal digits are 0 to 9 and A to _____. The main advantage of hexadecimal numbers is the ease of conversion from hexadecimal to _____ and vice versa.
8. (*microprocessors, F, binary*) A typical microcomputer may have up to 65,536 registers in its memory. Each of these registers, usually called a _____, stores 1 byte. Such a memory is specified as a 64-kilobyte memory, or simply a _____ memory.
9. (*memory location, 64K*) Binary-coded-decimal (BCD) numbers express each decimal digit as a ____. BCD numbers are useful whenever _____ information is transferred into or out of a digital system. Equipment using BCD numbers includes pocket calculators, electronic counters, and digital voltmeters.
10. (*nibble, decimal*) The ASCII code is a 7-bit code for _____ (letters, numbers, and other symbols).
11. (*alphanumerics*) With the typical microcomputer, you enter the program and data with typewriter keyboard that converts each character into ASCII code.

PROBLEMS

1-1. How many bytes are there in each of these numbers?
   a. 1100 0101
   b. 1011 1001 0110 1110
   c. 1111 1011 0111 0100 1010

1-2. What are the equivalent decimal numbers for each of the following binary numbers: 10, 110, 111, 1011, 1100, and 1110?

1-3. What is the base for each of these numbers?
   a. 348_{10}
   b. $1100\ 0101_2$
   c. 2312_5
   d. $F4C3_{16}$

1-4. Write the equation

$$2 + 2 = 4$$

using binary numbers.

1-5. What is the decimal equivalent of 2^{10}? What does 4K represent? Express 8,192 in K units.

1-6. A 4-bit register has output voltages of high-low-high-low. What is the binary number stored in the register? The decimal equivalent?

Fig. 1-8 An 8-bit LED display.

1-7. Figure 1-8 shows an 8-bit LED display. A light circle means that a LED is ON (binary 1) and a dark circle means a LED is OFF (binary 0). What is the binary number being displayed? The decimal equivalent?

1-8. Convert the following binary numbers to decimal numbers:
   a. 00111
   b. 11001
   c. 10110
   d. 11110

1-9. Solve the following equation for x:

$$x_{10} = 11001001_2$$

1-10. An 8-bit transistor register has this output:

low-high-low-high-low-high-low-high

What is the equivalent decimal number being stored?

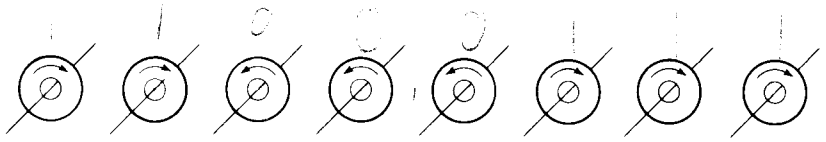


Fig. 1-9 An 8-bit core register.

1-11. In Fig. 1-9 clockwise flux stands for binary 1 and counterclockwise flux for binary 0. What is the binary number stored in the 8-bit core register? Convert this byte to an equivalent decimal number.

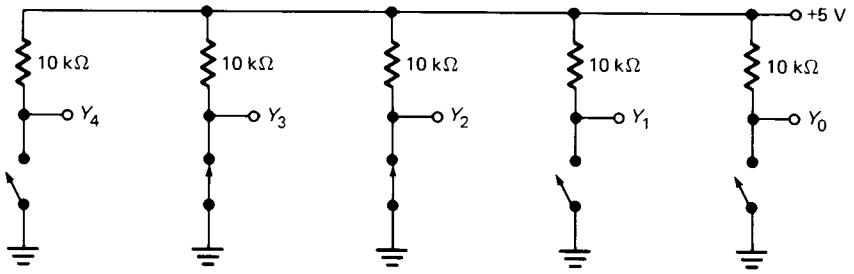


Fig. 1-10 A 5-bit switch register.

1-12. Figure 1-10 shows a 5-bit switch register. By opening and closing the switches you can set up different binary numbers. As usual, high output voltage stands for binary 1 and low output voltage for binary 0. What is the binary number stored in the switch register? The equivalent decimal number?

1-13. Convert decimal 56 to its binary equivalent.

1-14. Convert 72_{10} to a binary number.

1-15. An 8-bit transistor register stores decimal 150. What is the binary output of the register?

1-16. How would you set the switches of Fig. 1-10 to get a decimal output of 27?

1-17. A hexadecimal odometer displays F52A. What are the next six readings?

1-18. The reading on a hexadecimal odometer is 27FF. What is the next reading? Miles later, you see a reading of 8AFC. What are the next six readings?

1-19. Convert each of the following hexadecimal numbers to binary:
- a. FF
- b. ABC
- c. CD42
- d. F329

1-20. Convert each of these binary numbers to an equivalent hexadecimal number:
- a. 1110 1000
- b. 1100 1011
- c. 1010 1111 0110
- d. 1000 1011 1101 0110

1-21. Here is a program written for the 8085 microprocessor:

Address	Hex Contents
2000	3E
2001	0E
2002	D3
2003	20
2004	76

Convert the hex contents to equivalent binary numbers.

1-22. Convert each of these hexadecimal numbers to its decimal equivalent:
- a. FF
- b. A4
- c. 9B
- d. 3C

1-23. Convert the following hexadecimal numbers to their decimal equivalents:
- a. 0FFF
- b. 3FFF
- c. 7FE4
- d. B3D8

1-24. A microcomputer has memory locations from 0000 to 0FFF. Each memory location stores 1 byte. In decimal, how many bytes can the microcomputer store in its memory? How many kilobytes is this?

1-25. Suppose a microcomputer has memory locations from 0000 to 3FFF, each storing 1 byte. How

many bytes can the memory store? Express this in kilobytes.

1-26. A microcomputer has a 32K memory. How many bytes does this represent? If 0000 stands for the first memory location, what is the hexadecimal notation for the last memory location?

1-27. If a microcomputer has a 64K memory, what are the hexadecimal notations for the first and last memory locations?

1-28. Convert the following decimal numbers to hexadecimal:
- a. 4,095
- b. 16,383
- c. 32,767
- d. 65,535

1-29. Convert each of the following decimal numbers to hexadecimal numbers:
- a. 238
- b. 7,547
- c. 15,359
- d. 47,285

1-30. How many nibbles are there in each of the following:
- a. 1000 0111
- b. 1001 0000 0100 0011
- c. 0101 1001 0111 0010 0110 0110

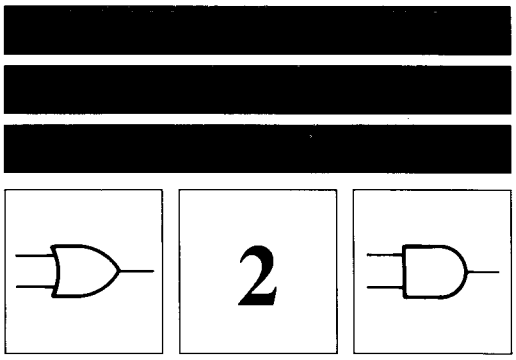
1-31. If the numbers in Prob. 1-30 are BCD numbers, what are the equivalent decimal numbers?

1-32. What is the ASCII code for each of the following:
- a. 7
- b. W
- c. f
- d. y

1-33. Suppose you type LIST with an ASCII keyboard. What is the binary output as you strike each letter?

1-34. For each of the following rows, provide the missing numbers in the bases indicated.

	Base 2	Base 10	Base 16
a.	0100 0001		
b.		200	
c.			3CD
d.		125	
e.	1101 1110 1111		
f.			FFFF
g.		2,000	



GATES

For centuries mathematicians felt there was a connection between mathematics and logic, but no one before George Boole could find this missing link. In 1854 he invented symbolic logic, known today as *boolean algebra*. Each variable in boolean algebra has either of two values: true or false. The original purpose of this two-state algebra was to solve logic problems.

Boolean algebra had no practical application until 1938, when Claude Shannon used it to analyze telephone switching circuits. He let the variables represent closed and open relays. In other words, Shannon came up with a new application for boolean algebra. Because of Shannon's work, engineers realized that boolean algebra could be applied to computer electronics.

This chapter introduces the *gate*, a circuit with one or more input signals but only one output signal. Gates are digital (two-state) circuits because the input and output signals are either low or high voltages. Gates are often called *logic circuits* because they can be analyzed with boolean algebra.

2-1 INVERTERS

An *inverter* is a gate with only one input signal and one output signal; the output state is always the opposite of the input state.

Transistor Inverter

Figure 2-1 shows a transistor inverter. This common-emitter amplifier switches between cutoff and saturation. When V_{IN} is low (approximately 0 V), the transistor cuts off and V_{OUT} is high. On the other hand, a high V_{IN} saturates the transistor, forcing V_{OUT} to go low.

Table 2-1 summarizes the operation. A low input produces a high output, and a high input results in a low output. Table 2-2 gives the same information in binary form; binary 0 stands for low voltage and binary 1 for high voltage.

An inverter is also called a NOT gate because the output is not the same as the input. The output is sometimes called the *complement* (opposite) of the input.

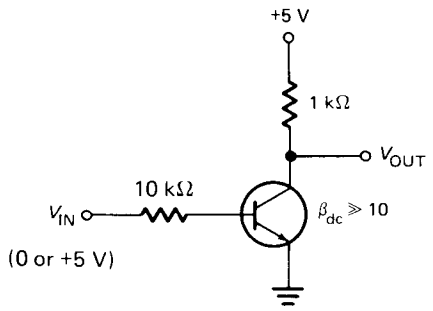


Fig. 2-1 Example of inverter design.

TABLE 2-1

V_{IN}	V_{OUT}
Low	High
High	Low

TABLE 2-2

V_{IN}	V_{OUT}
0	1
1	0

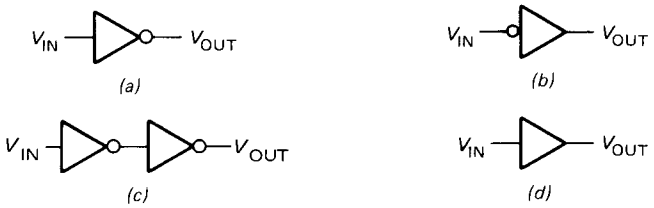


Fig. 2-2 Logic symbols: (*a*) inverter; (*b*) another inverter symbol; (*c*) double inverter; (*d*) buffer.

Inverter Symbol

Figure 2-2*a* is the symbol for an inverter of any design. Sometimes a schematic diagram will use the alternative symbol shown in Fig. 2-2*b*; the bubble (small circle) is on

the input side. Whenever you see either of these symbols, remember that the output is the complement of the input.

Noninverter Symbol

If you cascade two inverters (Fig. 2-2*c*), you get a noninverting amplifier. Figure 2-2*d* is the symbol for a noninverting amplifier. Regardless of the circuit design, the action is always the same: a low input voltage produces a low output voltage, and a high input voltage results in a high output voltage.

The main use of noninverting amplifier is buffering (isolating) two other circuits. More will be said about buffers in a later chapter.

EXAMPLE 2-1

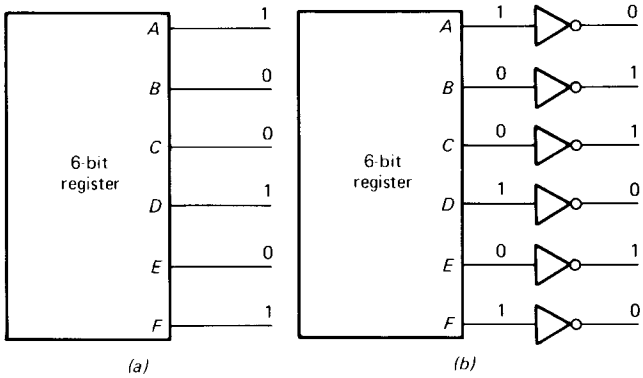


Fig. 2-3 Example 2-1.

Figure 2-3*a* has an output, *A* to *F*, of 100101. Show how to complement each bit.

SOLUTION

Easy. Use an inverter on each signal line (Fig. 2-3*b*). The final output is now 011010.

A *hex inverter* is a commercially available IC containing six separate inverters. Given a 6-bit register like Fig. 2-3*a*, we can connect a hex inverter to complement each bit as shown in Fig. 2-3*b*.

One more point. In Fig. 2-3*a* the bits may represent a coded instruction, number, letter, etc. To convey this variety of meaning, a string of bits is often called a binary word or simply a *word*. In Fig. 2-3*b* the word 100101 is complemented to get the word 011010.

2-2 OR GATES

The OR gate has two or more input signals but only one output signal. If any input signal is high, the output signal is high.

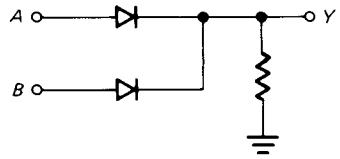


Fig. 2-4 A 2-input diode OR gate.

Diode OR Gate

Figure 2-4 shows one way to build an OR gate. If both inputs are low, the output is low. If either input is high, the diode with the high input conducts and the output is high. Because of the two inputs, we call this circuit a 2-input OR gate.

Table 2-3 summarizes the action; binary 0 stands for low voltage and binary 1 for high voltage. Notice that one or more high inputs produce a high output; this is why the circuit is called an OR gate.

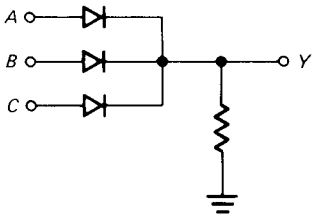


Fig. 2-5 A 3-input diode OR gate.

More than Two Inputs

Figure 2-5 shows a 3-input OR gate. If all inputs are low, all diodes are off and the output is low. If 1 or more inputs are high, the output is high.

Table 2-4 summarizes the action. A table like this is called a *truth table;* it lists all the input possibilities and the corresponding outputs. When constructing a truth table, always list the input words in a binary progression as shown (000, 001, 010, . . . , 111); this guarantees that all input possibilities will be accounted for.

An OR gate can have as many inputs as desired; add one diode for each additional input. Six diodes result in a 6-

TABLE 2-3. TWO INPUT OR GATE

A	*B*	*Y*
0	0	0
0	1	1
1	0	1
1	1	1

TABLE 2-4. THREE-INPUT OR GATE

A	*B*	*C*	*Y*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

input OR gate, nine diodes in a 9-input OR gate. No matter how many inputs, the action of any OR gate is summarized like this: one or more high inputs produce a high output.

Bipolar transistors and MOSFETs can also be used to build OR gates. But no matter what devices are used, OR gates always produce a high output when one or more inputs are high. Figure 2-6 shows the logic symbols for 2-, 3-, and 4-input OR gates.

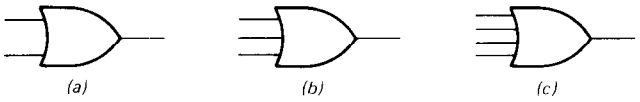


Fig. 2-6 OR-gate symbols.

EXAMPLE 2-2

Show the truth table of a 4-input OR gate.

SOLUTION

Let Y stand for the output bit and A, B, C, D for input bits. Then the truth table has input words of 0000, 0001, 0010, . . . , 1111, as shown in Table 2-5. As expected, output Y is 0 for input word 0000; Y is 1 for all other input words.

As a check, the number of input words in a truth table always equals 2^n, where n is the number of input bits. A 2-input OR gate has a truth table with 2^2 or 4 input words; a 3-input OR gate has 2^3 or 8 input words; and a 4-input OR gate has 2^4 or 16 input words.

TABLE 2-5. FOUR-INPUT OR GATE

A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

EXAMPLE 2-3

How many inputs words are in the truth table of an 8-input OR gate? Which input words produce a high output?

SOLUTION

The input words are 0000 0000, 0000 0001, . . . , 1111 1111. With the formula of the preceding example, the total number of input words is $2^n = 2^8 = 256$.

In any OR gate, 1 or more high inputs produce a high output. Therefore, the input word of 0000 0000 results in a low output; all other input words produce a high output.

EXAMPLE 2-4

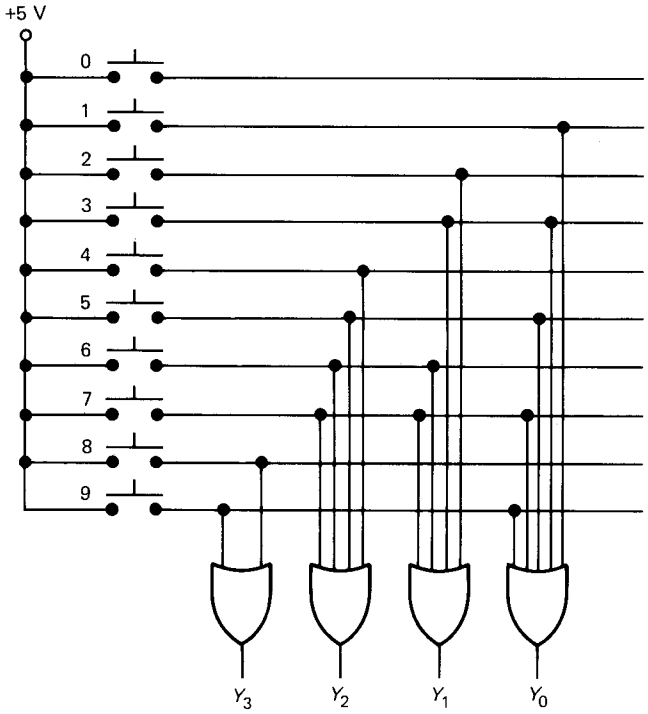


Fig. 2-7 Decimal-to-binary encoder.

The switches of Fig. 2-7 are push-button switches like those of a pocket calculator. The bits out of the OR gates form a 4-bit word, designated $Y_3Y_2Y_1Y_0$. What does the circuit do?

SOLUTION

Figure 2-7 is a decimal-to-binary *encoder*, a circuit that converts decimal to binary. For instance, when push button 3 is pressed, the Y_1 and Y_0 OR gates have high inputs; therefore, the output word is

$$Y_3Y_2Y_1Y_0 = 0011$$

If button 5 is keyed, the Y_2 and Y_0 OR gates have high inputs and the output word becomes

$$Y_3Y_2Y_1Y_0 = 0101$$

When switch 9 is pressed,

$$Y_3Y_2Y_1Y_0 = 1001$$

Check the other input switches to convince yourself that the output word always equals the binary equivalent of the switch being pressed.

2-3 AND GATES

The AND gate has two or more input signals but only one output signal. All inputs must be high to get a high output.

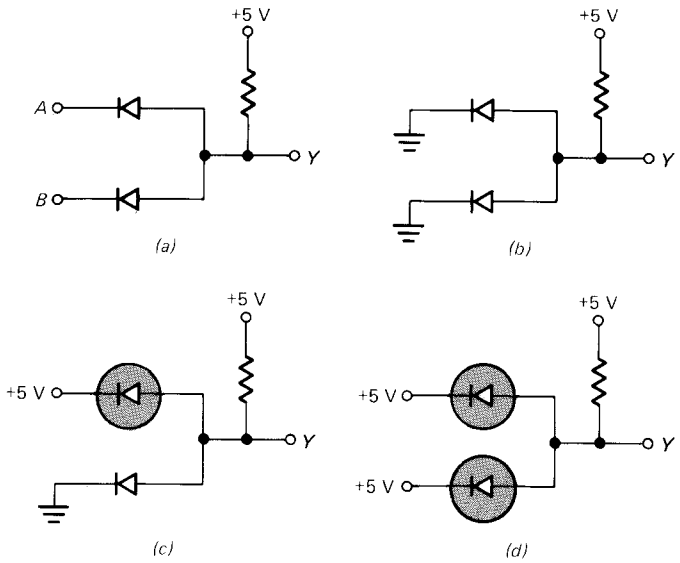


Fig. 2-8 A 2-input AND gate. (*a*) circuit; (*b*) both inputs low; (*c*)1 low input, 1 high; (*d*) both inputs high.

Diode AND Gate

Figure 2-8*a* shows one way to build an AND gate. In this circuit the inputs can be either low (ground) or high (+5 V). When both inputs are low (Fig. 2-8*b*), both diodes conduct and pull the output down to a low voltage. If one of the inputs is low and the other high (Fig. 2-8*c*), the diode with the low input conducts and this pulls the output down to a low voltage. The diode with the high input, on the other hand, is reverse-biased or cut off, symbolized by the dark shading in Fig. 2-8*c*.

When both inputs are high (Fig. 2-8*d*), both diodes are cut off. Since there is no current in the resistor, the supply voltage pulls the output up to a high voltage (+5 V).

TABLE 2-6. TWO-INPUT AND GATE

A	*B*	*Y*
0	0	0
0	1	0
1	0	0
1	1	1

Table 2-6 summarizes the action. As usual, binary zero stands for low voltage and binary 1 for high voltage. As you see, *A and B* must be high to get a high output; this is why the circuit is called an AND gate.

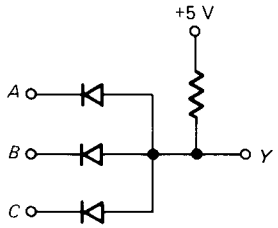


Fig. 2-9 A 3-input AND gate.

More than Two Inputs

Figure 2-9 is a 3-input AND gate. If all inputs are low, all diodes conduct and pull the output down to a low voltage. Even one conducting diode will pull the output down to a low voltage; therefore, the only way to get a high output is to have all inputs high. When all inputs are high, all diodes are nonconducting and the supply voltage pulls the output up to a high voltage.

Table 2-7 summarizes the 3-input AND gate. The output is 0 for all input words except 111. That is, all inputs must be high to get a high output.

AND gates can have as many inputs as desired; add one diode for each additional input. Eight diodes, for instance, result in an 8-input AND gate; sixteen diodes in a 16-input

TABLE 2-7. THREE-INPUT AND GATE

A	*B*	*C*	*Y*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

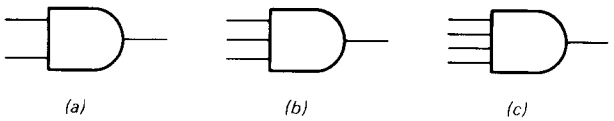


Fig. 2-10 AND-gate symbols.

AND gate. No matter how many inputs an AND gate has, the action can be summarized like this: All inputs must be high to get a high output.

Figure 2-10 shows the logic symbols for 2-, 3-, and 4-input AND gates.

EXAMPLE 2-5

Describe the truth table of an 8-input AND gate.

SOLUTION

The input words are from 0000 0000 to 1111 1111, following the binary progression. The total number of input words is

$$2^n = 2^8 = 256$$

The first 255 input words produce a 0 output. Only the last word, 1111 1111, results in a 1 output. This is because all inputs must be high to get a high output.

EXAMPLE 2-6

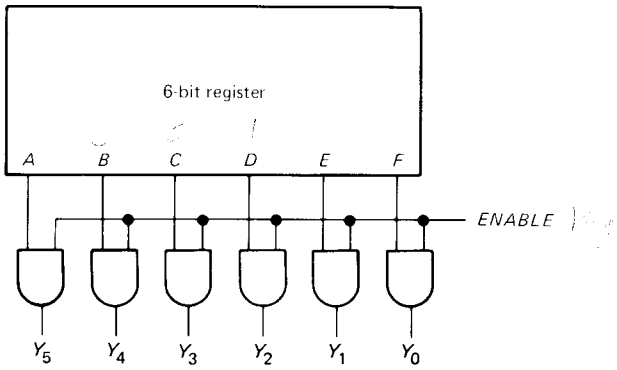


Fig. 2-11 Using AND gates to block or transmit data.

The 6-bit register of Fig. 2-11 stores the word ABCDEF. The *ENABLE* input can be low or high. What does the circuit do?

SOLUTION

One use of AND gates is to transmit data when certain conditions are satisfied. In Fig. 2-11 a low *ENABLE* blocks the register contents from the final output, but a high *ENABLE* transmits the register contents.

For instance, when

$$ENABLE = 0$$

each AND gate has a low *ENABLE* input. No matter what the register contents, the output of each AND gate must be low. Therefore, the final word is

$$Y_5Y_4Y_3Y_2Y_1Y_0 = 000000$$

As you see, a low *ENABLE* blocks the register contents from the final output.

On the other hand, when

$$ENABLE = 1$$

the output of each AND gate depends on the data inputs (A, B, C, . . .); a low data input results in a low output, and a high data input in a high output. For example, if ABCDEF = 100100, a high *ENABLE* gives

$$Y_5Y_4Y_3Y_2Y_1Y_0 = 100100$$

In general, a high *ENABLE* transmits the register contents to the final output to get

$$Y_5Y_4Y_3Y_2Y_1Y_0 = ABCDEF$$

2-4 BOOLEAN ALGEBRA

As mentioned earlier, Boole invented two-state algebra to solve logic problems. This new algebra had no practical use until Shannon applied it to telephone switching circuits. Today boolean algebra is the backbone of computer circuit analysis and design.

Inversion Sign

In boolean algebra a variable can be either a 0 or a 1. For digital circuits, this means that a signal voltage can be either low or high. Figure 2-12 is an example of a digital circuit because the input and output voltages are either low or high. Furthermore, because of the inversion, Y is always the complement of A.

Fig. 2-12 Inverter.

A word equation for Fig. 2-12 is

$$Y = \text{NOT } A \qquad (2\text{-}1)$$

If A is 0,

$$Y = \text{NOT } 0 = 1$$

On the other hand, if A is 1,

$$Y = \text{NOT } 1 = 0$$

In boolean algebra, the overbar stands for the NOT operation. This means that Eq. 2-1 can be written

$$Y = \overline{A} \tag{2-2}$$

Read this as "Y equals NOT A" or "Y equals the complement of A." Equation 2-2 is the standard way to write the output of an inverter.

Using the equation is easy. Given the value of A, substitute and solve for Y. For instance, if A is 0,

$$Y = \overline{A} = \overline{0} = 1$$

because NOT 0 is 1. On the other hand, if A is 1,

$$Y = \overline{A} = \overline{1} = 0$$

because NOT 1 is 0.

Fig. 2-13 OR gate.

OR Sign

A word equation for Fig. 2-13 is

$$Y = A \text{ OR } B \tag{2-3}$$

Given the inputs, you can solve for the output. For instance, if $A = 0$ and $B = 0$,

$$Y = 0 \text{ OR } 0 = 0$$

because 0 comes out of an OR gate when both inputs are 0s.

As another example, if $A = 0$ and $B = 1$,

$$Y = 0 \text{ OR } 1 = 1$$

because 1 comes out of an OR gate when either input is 1. Similarly, if $A = 1$ and $B = 0$,

$$Y = 1 \text{ OR } 0 = 1$$

If $A = 1$ and $B = 1$,

$$Y = 1 \text{ OR } 1 = 1$$

In boolean algebra the + sign stands for the OR operation. In other words, Eq. 2-3 can be written

$$Y = A + B \tag{2-4}$$

Read this as "Y equals A OR B." Equation 2-4 is the standard way to write the output of an OR gate.

Given the inputs, you can substitute and solve for the output. For instance, if $A = 0$ and $B = 0$,

$$Y = A + B = 0 + 0 = 0$$

If $A = 0$ and $B = 1$,

$$Y = A + B = 0 + 1 = 1$$

because 0 ORed with 1 results in 1. If $A = 1$ and $B = 0$,

$$Y = A + B = 1 + 0 = 1$$

If both inputs are high,

$$Y = A + B = 1 + 1 = 1$$

because 1 ORed with 1 gives 1.

Don't let the new meaning of the + sign bother you. There's nothing unusual about symbols having more than one meaning. For instance, "pot" may mean a cooking utensil, a flower container, the money wagered in a card game, a derivative of *cannabis sativa* and so forth; the intended meaning is clear from the sentence it's used in. Similarly, the + sign may stand for ordinary addition or OR addition; the intended meaning comes across in the way it's used. If we're talking about decimal numbers, + means ordinary addition, but when the discussion is about logic circuits, + stands for OR addition.

Fig. 2-14 AND gate.

AND Sign

A word equation for Fig. 2-14 is

$$Y = A \text{ AND } B \tag{2-5}$$

In boolean algebra the multiplication sign stands for the AND operation. Therefore, Eq. 2-5 can be written

$$Y = A \cdot B$$

or simply

$$Y = AB \tag{2-6}$$

Read this as "Y equals A AND B." Equation 2-6 is the standard way to write the output of an AND gate.

Given the inputs, you can substitute and solve for the output. For instance, if both inputs are low,

$$Y = AB = 0 \cdot 0 = 0$$

because 0 ANDed with 0 gives 0. If A is low and B is high,

$$Y = AB = 0 \cdot 1 = 0$$

because 0 comes out of an AND gate if any input is 0. If A is 1 and B is 0,

$$Y = AB = 1 \cdot 0 = 0$$

When both inputs are high,

$$Y = AB = 1 \cdot 1 = 1$$

because 1 ANDed with 1 gives 1.

Decision-Making Elements

The inverter, OR gate, and AND gate are often called *decision-making elements* because they can recognize some input words while disregarding others. A gate recognizes a word when its output is high; it disregards a word when its output is low. For example, the AND gate disregards all words with one or more 0s; it recognizes only the word whose bits are all 1s.

Notation

In later equations we need to distinguish between bits that are ANDed and bits that are part of a binary word. To do this we will use italic (slanted) letters (A, B, Y, etc.) for ANDed bits and roman (upright) letters (A, B, Y, etc.) for bits that form a word.

For example, $Y_3Y_2Y_1Y_0$ stands for the logical product (ANDing) of Y_3, Y_2, Y_1, and Y_0. If $Y_3 = 1$, $Y_2 = 0$, $Y_1 = 0$, and $Y_0 = 1$, the product $Y_3Y_2Y_1Y_0$ will reduce as follows:

$$Y_3Y_2Y_1Y_0 = 1 \cdot 0 \cdot 0 \cdot 1 = 0$$

In this case, the italic letters represent bits that are being ANDed.

On the other hand, $Y_3Y_2Y_1Y_0$ is our notation for a 4-bit word. With the Y values just given, we can write

$$Y_3Y_2Y_1Y_0 = 1001$$

In this equation, we are not dealing with bits that are ANDed; instead, we are dealing with bits that are part of a word.

The distinction between italic and roman notation will become clearer when we get to computer analysis.

Positive and Negative Logic

A final point. *Positive logic* means that 1 stands for the more positive of the two voltage levels. *Negative logic* means that 1 stands for the more negative of the two voltage levels. For instance, if the two voltage levels are 0 and -5 V, positive logic would have 1 stand for 0 V and 0 for -5 V, whereas negative logic would have 1 stand for -5 V and 0 for 0 V.

Ordinarily, people use positive logic with positive supply voltages and negative logic with negative supply voltages. Throughout this book, we will be using positive logic.

EXAMPLE 2-7

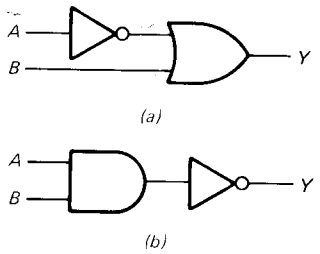


Fig. 2-15 Logic circuits.

What is the boolean equation for Fig. 2-15*a*? The output if both inputs are high?

SOLUTION

A is inverted before it reaches the OR gate; therefore, the upper input to the OR gate is $\overline{A}$. The final output is

$$Y = \overline{A} + B$$

This is the boolean equation for Fig. 2-15*a*.

To find the output when both inputs are high, either of two approaches can be used. First, you can substitute directly into the foregoing equation and solve for Y

$$Y = \overline{A} + B = \overline{1} + 1 = 0 + 1 = 1$$

Alternatively, you can analyze the operation of Fig. 2-15*a* like this. If both inputs are high, the inputs to the OR gate are 0 and 1. Now, 0 ORed with 1 gives 1. Therefore, the final output is high.

EXAMPLE 2-8

What is the boolean equation for Fig. 2-15*b*? If both inputs are high, what is the output?

SOLUTION

The AND gate forms the logical product AB, which is inverted to get

$$Y = \overline{AB}$$

Read this as "Y equals NOT AB" or "Y equals the complement of AB."

If both inputs are high, direct substitution into the equation gives

$$Y = \overline{AB} = \overline{1 \cdot 1} = \overline{1} = 0$$

Note the order of operations: the ANDing is done first, then the inversion.

Instead of using the equation, you can analyze Fig. 2-15*b* as follows. If both inputs are high, the AND gate has a high output. Therefore, the final output is low.

EXAMPLE 2-9

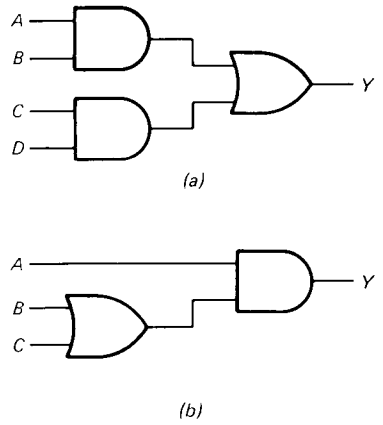


Fig. 2-16 Logic circuits.

What is the boolean equation for Fig. 2-16*a*? The truth table? Which input words does the circuit recognize?

SOLUTION

The upper AND gate forms the logical product AB, and the lower AND gate gives CD. ORing these products results in

$$Y = AB + CD$$

Read this as "Y equals AB OR CD."

Next, look at Fig. 2-16*a*. The final output is high if the OR gate has one or more high inputs. This happens when AB is 1, CD is 1, or both are 1s. In turn, AB is 1 when

$$A = 1 \quad \text{and} \quad B = 1$$

TABLE 2-8. TRUTH TABLE FOR $Y = AB + CD$

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

CD is 1 when

$$C = 1 \quad \text{and} \quad D = 1$$

Both products are 1s when

$$A = 1 \quad B = 1 \quad C = 1 \quad \text{and} \quad D = 1$$

Therefore, the final output is high when A and B are 1s, when C and D are 1s, or when all inputs are 1s.

Table 2-8 summarizes the foregoing analysis. From this it's clear that the circuit recognizes these input words: 0011, 0111, 1011, 1100, 1101, 1110, and 1111.

EXAMPLE 2-10

Write the boolean equation for Fig. 2-16*b*. If all inputs are high, what is the output?

SOLUTION

The OR gate forms the logical sum $B + C$. This sum is ANDed with A to get

$$Y = A(B + C)$$

(Parentheses indicate ANDing.)

One way to find the output when all inputs are high is to substitute and solve as follows:

$$Y = A(B + C) = 1(1 + 1) = 1(1) = 1$$

Alternatively, you can analyze Fig. 2-16*b* like this. If all inputs are high, the OR gate has a high output; therefore, both inputs to the AND gate are high. Since all high inputs to an AND gate result in a high output, the final output is high.

EXAMPLE 2-11

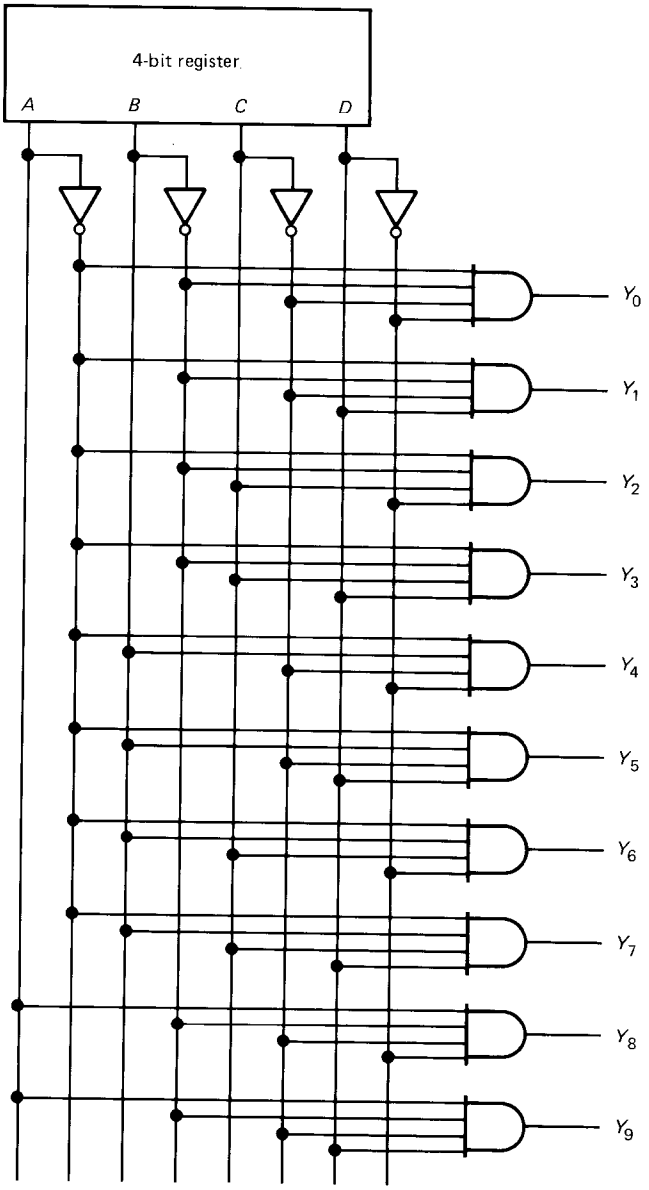


Fig. 2-17 A 1-of-10 decoder.

What is the boolean equation for each Y output in Fig. 2-17?

SOLUTION

Each AND gate forms the logical product of its input signals. The inputs to the top AND gate are $\overline{A}$, $\overline{B}$, $\overline{C}$ and $\overline{D}$; therefore,

$$Y_0 = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}$$

The inputs to the next AND gate are $\overline{A}$, $\overline{B}$, $\overline{C}$ and D; this means that

$$Y_1 = \overline{A}\,\overline{B}\,\overline{C}D$$

Analyzing the remaining gates gives

$$Y_2 = \overline{A}\,\overline{B}C\,\overline{D}$$
$$Y_3 = \overline{A}\,\overline{B}CD$$
$$Y_4 = \overline{A}B\,\overline{C}\,\overline{D}$$
$$Y_5 = \overline{A}B\,\overline{C}D$$
$$Y_6 = \overline{A}BC\,\overline{D}$$
$$Y_7 = \overline{A}BCD$$
$$Y_8 = A\,\overline{B}\,\overline{C}\,\overline{D}$$
$$Y_9 = A\,\overline{B}\,\overline{C}D$$

EXAMPLE 2-12

What does the circuit of Fig. 2-17 do?

SOLUTION

This is a binary-to-decimal decoder, a circuit that converts from binary to decimal. For instance, when the register contents are 0011, the Y_3 AND gate has all high inputs; therefore, Y_3 is high. Furthermore, register contents of 0011 mean that all other AND gates have at least one low input. As a result, all other AND gates have low outputs. (Analyze the circuit to convince yourself.)

If the register contents change to 0100, only the Y_4 AND gate has all high inputs; therefore, only Y_4 is high. If the register contents change to 0111, Y_7 is the only high output.

In general, the subscript of the high output equals the decimal equivalent of the binary number stored in the register. This is why the circuit is called a *binary-to-decimal* decoder.

The circuit of this example is also called a 4-line–to–10-line decoder because there are 4 input lines and 10 output lines. Another name for it is a 1-of-10 decoder because only 1 of 10 output lines has a high voltage.

GLOSSARY

AND *gate* A logic circuit whose output is high only when all inputs are high.

boolean algebra Originally known as symbolic logic, this modern algebra uses the set of numbers 0 and 1. The

operations OR, AND, and NOT are sometimes called *union, intersection,* and *inversion*. Boolean algebra is ideally suited to digital circuit analysis.

complement The output of an inverter.

gate A logic circuit with one or more input signals but only one output signal.

inverter A gate with only 1 input and 1 output. The output is always the complement of the input. Also known as a NOT gate.

logic circuit A circuit whose input and output signals are two-state, either low or high voltages. The basic logic circuits are OR, AND, and NOT gates.

OR *gate* A logic circuit with 2 or more inputs and only 1 output; 1 or more high inputs produce a high output.

truth table A table that shows all input and output possibilities for a logic circuit. The input words are listed in binary progression.

word A string of bits that represent a coded instruction or data.

SELF-TESTING REVIEW

Read each of the following and provide the missing words. Answers appear at the beginning of the next question.

1. A gate is a logic circuit with one or more input signals but only _____ output signal. These signals are either _____ or high.
2. (*one, low*) An inverter is a gate with only _____ input; the output is always in the opposite state from the input. An inverter is also called a _____ gate. Sometimes the output is referred to as the complement of the input.
3. (*1,* NOT) The OR gate has two or more input signals. If any input is _____, the output is high. The number of input words in a truth table always equals _____, where n is the number of input bits.
4. (*high,* 2^n) The _____ gate has two or more input signals. All inputs must be high to get a high output.
5. (AND) In boolean algebra, the overbar stands for the NOT operation, the plus sign stands for the _____ operation, and the times sign for the _____ operation.
6. (OR, AND) The inverter, OR gate, and AND gate are called decision-making elements because they can recognize some input _____ while disregarding others. A gate recognizes a word when its output is _____.
7. (*words, high*) A binary-to-decimal decoder is also called a 4-line–to–10-line decoder because it has 4 input lines and 10 output lines. Another name for it is the 1-of-10 decoder because only 1 of its 10 output lines is high at a time.

PROBLEMS

2-1. How many inputs signals can a gate have? How many output signals?

2-2. If you cascade seven inverters, does the overall circuit act like an inverter or noninverter?

2-3. Double inversion occurs when two inverters are cascaded. Does such a connection act like an inverter or noninverter?

2-4. The contents of the 6-bit register in Fig. 2-3*b* change to 101010. What is the decimal equivalent of the register contents? The decimal equivalent out of the hex inverter?

2-5. An OR gate has 6 inputs. How many input words are in its truth table? What is the only input word that produces a 0 output?

2-6. Figure 2-18 shows a hexadecimal encoder, a circuit that converts hexadecimal to binary. Pressing each push-button switch results in a different output word $Y_3Y_2Y_1Y_0$. Starting with switch 0, what are the output words? (NOTE: The new symbol in Fig. 2-18 is another way to draw an OR gate.

2-7. In Fig. 2-18 what switches would you press to produce

0011 1001 1100 1111

(Work from left to right.)

2-8. What is the 4-bit output in Fig. 2-18 when switch A is pressed? Switch 4? Switch E? Switch 6?

2-9. An AND gate has 7 inputs. How many input words are in its truth table? What is the only input word that produces a 1 output?

2-10. Visualize the register contents of Fig. 2-19 as the word $A_7A_6 \cdots A_0$, and the final output as the word $Y_7Y_6 \cdots Y_0$. What is the output word for each of the following conditions:

a. $A_7A_6 \cdots A_0$ = 1100 1010, *ENABLE* = 0.
b. $A_7A_6 \cdots A_0$ = 0101 1101, *ENABLE* = 1.
c. $A_7A_6 \cdots A_0$ = 1111 0000, *ENABLE* = 1.
d. $A_7A_6 \cdots A_0$ = 1010 1010, *ENABLE* = 0.

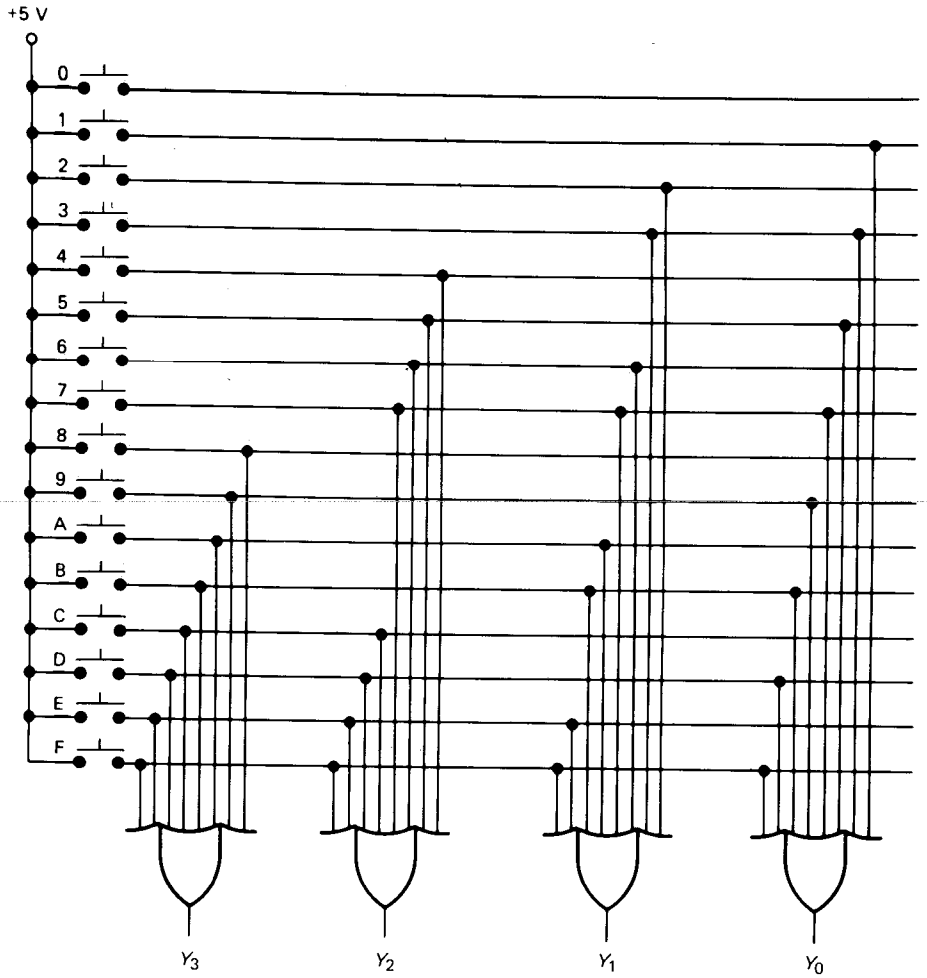


Fig. 2-18 Hexadecimal encoder.

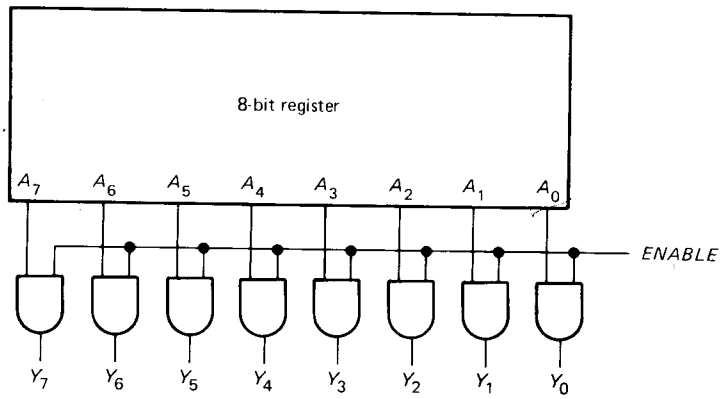


Fig. 2-19

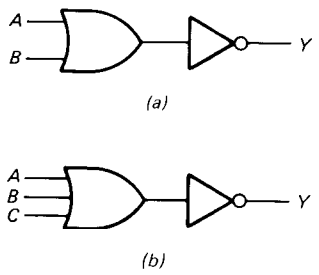


Fig. 2-20

2-11. The 8-bit register of Fig. 2-19 stores 59_{10}. What is the decimal equivalent of the final output word if *ENABLE* = 0? If *ENABLE* = 1?

2-12. Answer these questions:

a. What input words does a 6-input OR gate recognize? What word does it disregard?

b. What input word does an 8-input AND gate recognize? What words does it disregard?

2-13. What is the boolean equation for Fig. 2-20*a*? The output if both inputs are high?

2-14. If all inputs are high in Fig. 2-20*b*, what is the output? The boolean equation for the circuit? What is the only ABC input word the circuit recognizes?

2-15. If you constructed the truth table for Fig. 2-20*b*, how many input words would it contain?

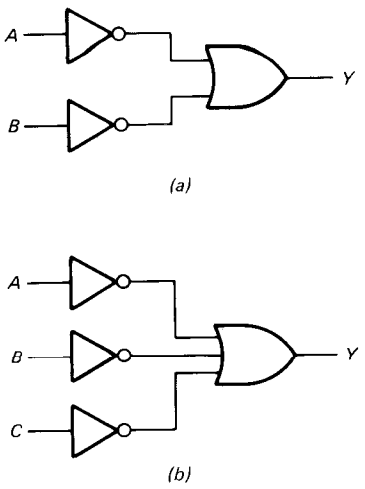


Fig. 2-21

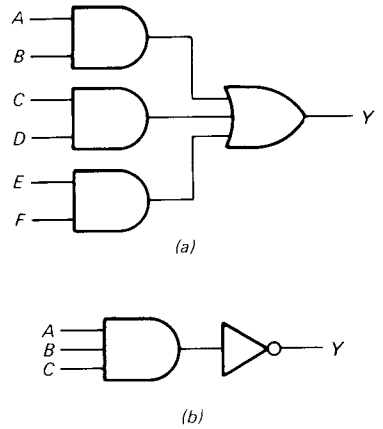


Fig. 2-22

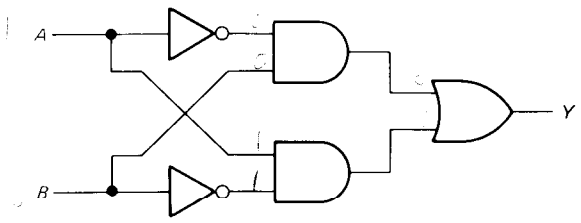


Fig. 2-23

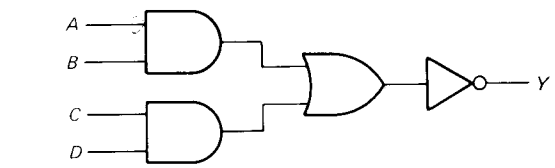


Fig. 2-24

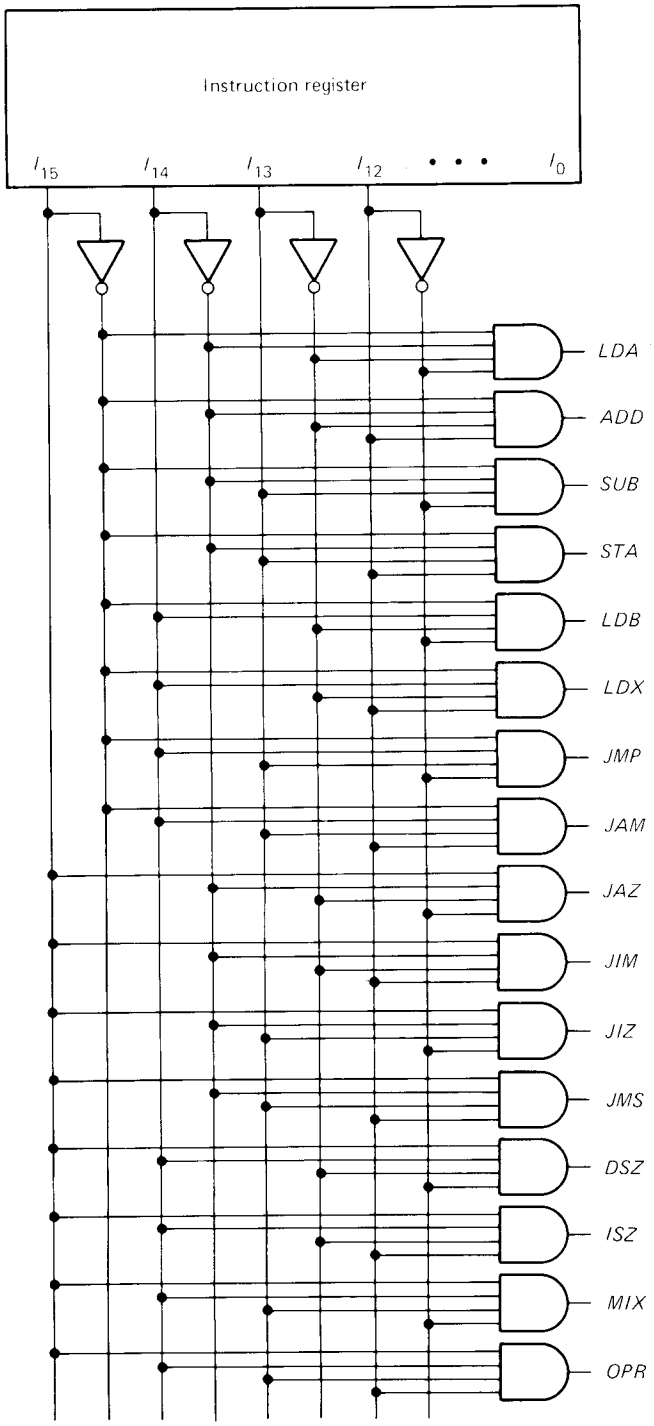


Fig. 2-25 A 1-of-16 decoder.

2-16. What is the boolean equation for Fig. 2-21*a*? The output if both inputs are high?

2-17. If all inputs are high in Fig. 2-21*b*, what is the output? What is the boolean equation of the circuit? What ABC input words does the circuit recognize? What is the only word it disregards?

2-18. What is the boolean equation for Fig. 2-22*a*? The output if all inputs are 1s? If you were to construct the truth table, how many input words would it have?

2-19. Write the boolean equation for Fig. 2-22*b*. If all inputs are 1s, what is the output?

2-20. If both inputs are high in Fig. 2-23, what is the output? What is the boolean equation for the circuit? Describe the truth table.

2-21. What is the boolean equation for Fig. 2-24? How many ABCD input words are in the truth table? Which input words does the circuit recognize?

2-22. Because of the historical connection between boolean algebra and logic, some people use the words "true" and "false" instead of "high" and "low" when discussing logic circuits. For instance, here's how an AND gate can be described. If any input is false, the output is false; if all inputs are true, the output is true.

a. If both inputs are false in Fig. 2-23, what is the output?

b. What is the output in Fig. 2-23 if one input is false and the other true?

c. In Fig. 2-23 what is the output if all inputs are true?

2-23. Figure 2-25 shows a 1-of-16 decoder. The signals coming out of the decoder are labeled *LDA, ADD, SUB*, and so on. The word formed by the 4 leftmost register bits is called the **OP CODE.** As an equation,

$$\textbf{OP CODE} = I_{15}I_{14}I_{13}I_{12}$$

a. If *LDA* is high, what does **OP CODE** equal?

b. If *ADD* is high, what does it equal?

c. When **OP CODE** = 1001, which of the output signals is high?

d. Which output signal is high if **OP CODE** = 1111?

2-24. In Fig. 2-25, list the **OP CODE** words and the corresponding high output signals. (Start with 0000 and proceed in binary to 1111.)

2-25. In the following equations the equals sign means "is equivalent to." Classify each of the following as positive or negative logic:

a. 0 = 0 V and 1 = +5 V.

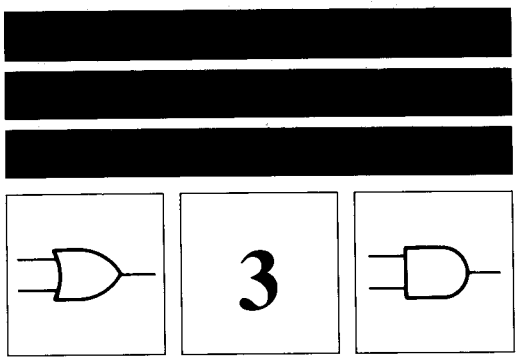
b. 0 = +5 V and 1 = 0 V.

c. 0 = −5 V and 1 = 0 V.

d. 0 = 0 V and 1 = −5 V.

2-26. In Fig. 2-25 four output lines come from the decoder. Is it possible to add more op codes without increasing the number of output lines?

2-27. How many output lines from the decoder would be needed to have 256 op codes?



MORE LOGIC GATES

This chapter introduces NOR and NAND gates, devices that are widely used in industry. You will also learn about De Morgan's theorems; they help you to rearrange and simplify logic circuits.

3-1 NOR GATES

The NOR gate has two or more input signals but only one output signal. All inputs must be low to get a high output. In other words, the NOR gate recognizes only the input word whose bits are all 0s.

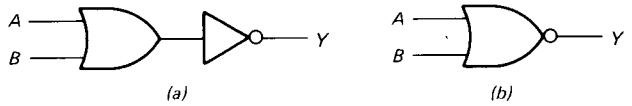


Fig. 3-1 NOR gate: (*a*) logical meaning; (*b*) standard symbol.

Two-Input Gate

Figure 3-1*a* shows the logical structure of a NOR gate, which is an OR gate followed by an inverter. Therefore, the final output is NOT the OR of the inputs. Originally called a NOT-OR gate, the circuit is now referred to as a NOR gate.

Figure 3-1*b* is the standard symbol for a NOR gate. Notice that the inverter triangle has been deleted and the small circle or bubble moved to the OR-gate output. The bubble is a reminder of the inversion that follows the ORing.

With Fig. 3-1*a* and *b* the following ideas are clear. If both inputs are low, the final output is high. If one input is low and the other high, the output is low. And if both inputs are high, the output is low.

Table 3-1 summarizes the circuit action. As you see, the NOR gate recognizes only the input word whose bits are all 0s. In other words, all inputs must be low to get a high output.

TABLE 3-1. TWO-INPUT NOR GATE

A	B	$\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

Incidentally, the boolean equation for a 2-input NOR gate is

$$Y = \overline{A + B} \tag{3-1}$$

Read this as "Y equals NOT A OR B." If you use this equation, remember that the ORing is done first, then the inversion.

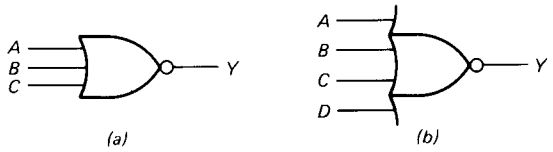


Fig. 3-2 NOR gates: (*a*) 3-input; (*b*) 4-input.

Three-Input Gate

Regardless of how many inputs a NOR gate has, it is still logically equivalent to an OR gate followed by an inverter. For instance, Fig. 3-2*a* shows a 3-input NOR gate. The 3 inputs are ORed, and the result is inverted. Therefore, the boolean equation is

$$Y = \overline{A + B + C} \tag{3-2}$$

The analysis of Fig. 3-2*a* goes like this. If all inputs are low, the result of ORing is low; therefore, the final output

TABLE 3-2. THREE-INPUT NOR GATE

A	B	C	$\overline{A+B+C}$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

is high. If one or more inputs are high, the result of ORing is high; so the final output is low.

Table 3-2 summarizes the action of a 3-input NOR gate. As you see, the circuit recognizes only the input word whose bits are 0s. In other words, all inputs must be low to get a high output.

Four-Input Gate

Figure 3-2*b* is the symbol for a 4-input NOR gate. The inputs are ORed, and the result is inverted. For this reason, the boolean equation is

$$Y = \overline{A + B + C + D} \tag{3-3}$$

The corresponding truth table has input words from 0000 to 1111. Word 0000 gives a 1 output; all other words produce a 0 output. (For practice, you should construct the truth table of the 4-input NOR gate.)

3-2 DE MORGAN'S FIRST THEOREM

Most mathematicians ignored boolean algbebra when it first appeared; some even ridiculed it. But Augustus De Morgan saw that it offered profound insights. He was the first to acclaim Boole's great achievement.

Always a warm and likable man, De Morgan himself had paved the way for boolean algebra by discovering two important theorems. This section introduces the first theorem.

The First Theorem

Figure 3-3*a* is a 2-input NOR gate, analyzed earlier. As you recall, the boolean equation is

$$Y = \overline{A + B}$$

and Table 3-3 is the truth table.

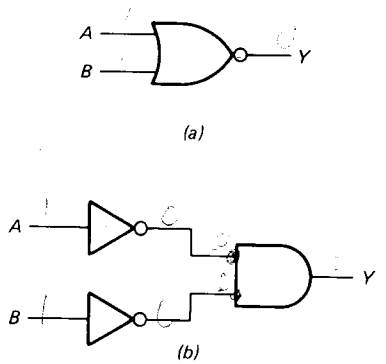


Fig. 3-3 De Morgan's first theorem: (*a*) NOR gate; (*b*) AND gate with inverted inputs.

Figure 3-3*b* has the inputs inverted before they reach the AND gate. Therefore, the boolean equation is

$$Y = \overline{A}\,\overline{B}$$

If both inputs are low in Fig. 3-3*b*, the AND gate has high inputs; therefore, the final output is high. If one or more inputs are high, one or more AND-gate inputs must be low and the final output is low. Table 3-4 summarizes these ideas.

TABLE 3-3

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

TABLE 3-4

A	B	$\overline{A}\,\overline{B}$
0	0	1
0	1	0
1	0	0
1	1	0

Compare Tables 3-3 and 3-4. They're identical. This means that the two circuits are logically equivalent; given the same inputs, the outputs are the same. In other words, the circuits of Fig. 3-3 are interchangeable.

De Morgan discovered the foregoing equivalence long before logic circuits were invented. His first theorem says

$$\overline{A + B} = \overline{A}\,\overline{B} \tag{3-4}$$

The left member of this equation represents Fig. 3-3*a*; the right member, Fig. 3-3*b*. Equation 3-4 says that Fig. 3-3*a* and *b* are equivalent (interchangeable).

Bubbled AND Gate

Figure 3-4*a* shows an AND gate with inverted inputs. This circuit is so widely used that the abbreviated logic symbol of Fig. 3-4*b* has been adopted. Notice that the inverter triangles have been deleted and the bubbles moved to the

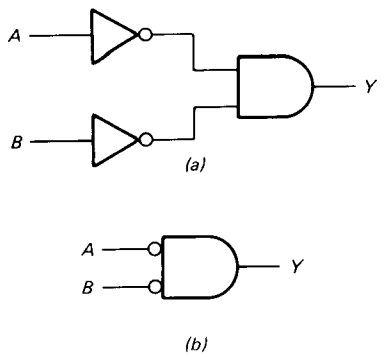


Fig. 3-4 AND gate with inverted inputs: (*a*) circuit; (*b*) abbreviated symbol.

AND-gate inputs. From now on, we will refer to Fig. 3-4*b* as a *bubbled* AND *gate;* the bubbles are a reminder of the inversion that takes place before ANDing.

Fig. 3-5 De Morgan's first theorem.

Figure 3-5 is a graphic summary of De Morgan's first theorem. A NOR gate and a bubbled AND gate are equivalent. As shown later, because the circuits are interchangeable, you can often reduce complicated logic circuits to simpler forms.

More than Two Inputs

When 3 inputs are involved, De Morgan's first theorem is written

$$\overline{A + B + C} = \overline{A}\,\overline{B}\,\overline{C} \tag{3-5}$$

For 4 inputs

$$\overline{A + B + C + D} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} \tag{3-6}$$

In both cases, the theorem says that the complement of a sum equals the product of the complements.

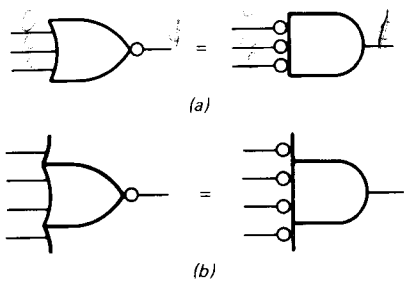


Fig. 3-6 De Morgan's first theorem: (*a*) 3-input circuits; (*b*) 4-input circuits.

Here's what really counts. Equation 3-5 says that a 3-input NOR gate and a 3-input bubbled AND gate are equivalent (see Fig. 3-6*a*). Equation 3-6 means that a 4-input NOR gate and a 4-input bubbled AND gate are equivalent (Fig. 3-6*b*). Memorize these equivalent circuits; they are a visual statement of De Morgan's first theorem.

Notice in Fig. 3-6*b* how the input edges of the NOR gate and the bubbled AND gate have been extended. This is common drafting practice when there are many input signals. The same idea applies to any type of gate.

EXAMPLE 3-1

Prove that Fig. 3-7*a* and *c* are equivalent.

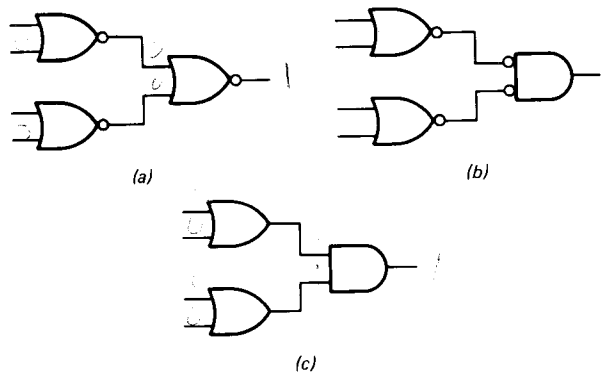


Fig. 3-7 Equivalent De Morgan circuits.

SOLUTION

The final NOR gate in Fig. 3-7*a* is equivalent to a bubbled AND gate. This allows us to redraw the circuit as shown in Fig. 3-7*b*.

Double inversion produces noninversion; therefore, each double inversion in Fig. 3-7*b* cancels out, leaving the simplified circuit of Fig. 3-7*c*. Figure 3-7*a* and *c* are therefore equivalent.

Remember the idea. Given a logic circuit, you can replace any NOR gate by a bubbled AND gate. Then any double inversion (a pair of bubbles in a series path) cancels out. Sometimes you wind up with a simpler logic circuit than you started with; sometimes not.

But the point remains. De Morgan's first theorem enables you to rearrange a logic circuit with the hope of finding a simpler equivalent circuit or perhaps getting more insight into how the original circuit works.

3-3 NAND GATES

The NAND gate has two or more input signals but only one output signal. All input signals must be high to get a low output.

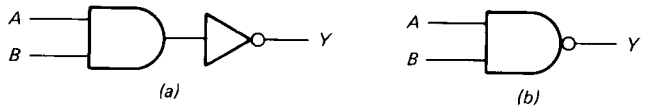


Fig. 3-8 NAND gate: (*a*) logical meaning; (*b*) standard symbol.

Two-Input Gate

Figure 3-8*a* shows the logical structure of a NAND gate, an AND gate followed by an inverter. Therefore, the final output is NOT the AND of the inputs. Originally called a NOT-AND gate, the circuit is now referred to as a NAND gate.

Figure 3-8*b* is the standard symbol for a NAND gate. The inverter triangle has been deleted and the bubble moved to the AND-gate output. If one or more inputs are low, the result of ANDing is low; therefore, the final inverted output is high. Only when all inputs are high does the ANDing produce a high signal; then the final output is low.

Table 3-5 summarizes the action of a 2-input NAND gate. As shown, the NAND gate recognizes any input word with one or more 0s. That is, one or more low inputs produce a high output. The boolean equation for a 2-input NAND gate is

$$Y = \overline{AB} \tag{3-7}$$

Read this as "Y equals NOT AB." If you use this equation, remember that the ANDing is done first then the inversion.

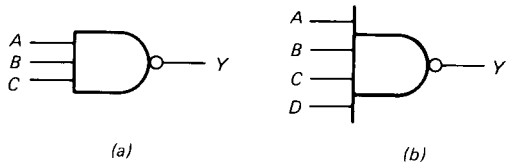


Fig. 3-9 NAND gates: (*a*) 3-input; (*b*) 4-input.

Three-Input Gate

Regardless of how many inputs a NAND gate has, it's still logically equivalent to an AND gate followed by an inverter. For example, Fig. 3-9*a* shows a 3-input NAND gate. The inputs are ANDed, and the product is inverted. Therefore, the boolean equation is

$$Y = \overline{ABC} \tag{3-8}$$

Here is the analysis of Fig. 3-9*a*. If one or more inputs are low, the result of ANDing is low; therefore, the final output is high. If all inputs are high, the ANDing gives a high signal; so the final output is low.

Table 3-6 is the truth table for a 3-input NAND gate. As indicated, the circuit recognizes words with one or more 0s. This means that one or more low inputs produce a high output.

TABLE 3-5. TWO-INPUT NAND GATE

A	*B*	$\overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

TABLE 3-6. THREE-INPUT NAND GATE

A	*B*	*C*	$\overline{ABC}$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Four-Input Gate

Figure 3-9*b* is the symbol for a 4-input NAND gate. The inputs are ANDed, and the result is inverted. Therefore, the boolean equation is

$$Y = \overline{ABCD} \tag{3-9}$$

If you construct the truth table, you will have input words from 0000 to 1111. All words from 0000 through 1110 produce a 1 output; only the word 1111 gives a 0 output.

3-4 DE MORGAN'S SECOND THEOREM

The proof of De Morgan's second theorem is similar to the proof given for the first theorem. What follows is a brief explanation.

The Second Theorem

When two inputs are used, De Morgan's second theorem says that

$$\overline{AB} = \overline{A} + \overline{B} \tag{3-10}$$

In words, the complement of a product equals the sum of the complements. The left member of this equation represents a NAND gate (Fig. 3-10*a*); the right member stands

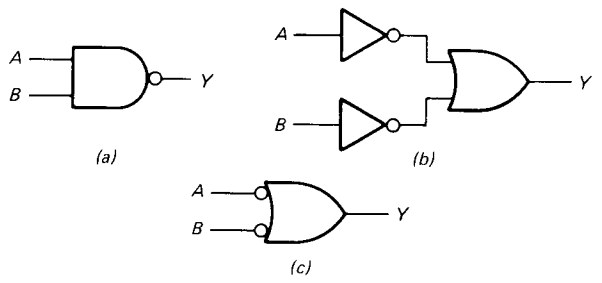


Fig. 3-10 De Morgan's second theorem: (*a*) NAND gate; (*b*) OR gate with inverted inputs; (*c*) bubbled OR gate.

for an OR gate with inverted inputs (Fig. 3-10*b*). Therefore, De Morgan's second theorem boils down to the fact that Fig. 3-10*a* and *b* are equivalent.

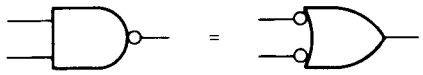


Fig. 3-11 De Morgan's second theorem.

Bubbled OR Gate

The circuit of Fig. 3-10*b* is so widely used that the abbreviated logic symbol of Fig. 3-10*c* has been adopted. From now on we will refer to Fig. 3-10*c* as a *bubbled* OR gate; the bubbles are a reminder of the inversion that takes place before ORing.

Figure 3-11 is a visual statement of De Morgan's second theorem: a NAND gate and a bubbled OR gate are equivalent. This equivalence allows you to replace one circuit by the other whenever desired. This may lead to a simpler logic circuit or give you more insight into how the original circuit works.

More than Two Inputs

When 3 inputs are involved, De Morgan's second theorem is written

$$\overline{ABC} = \overline{A} + \overline{B} + \overline{C} \qquad (3\text{-}11)$$

If 4 inputs are used,

$$\overline{ABCD} = \overline{A} + \overline{B} + \overline{C} + \overline{D} \qquad (3\text{-}12)$$

These equations say that the complement of a product equals the sum of the complements.

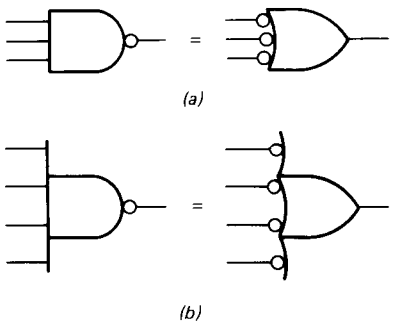


Fig. 3-12 De Morgan's second theorem: (*a*) 3-input circuits; (*b*) 4-input circuits.

Figure 3-12 is a visual summary of the second theorem. Whether 3 or 4 inputs are involved, a NAND gate and a bubbled OR gate are equivalent (interchangeable).

EXAMPLE 3-2

Prove that Fig. 3-13*a* and *c* are equivalent.

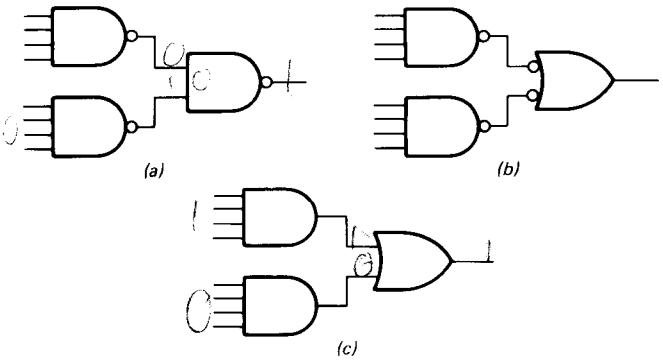


Fig. 3-13 Equivalent circuits.

SOLUTION

Replace the final NAND gate in Fig. 3-13*a* by a bubbled OR gate. This gives Fig. 3-13*b*. The double inversions cancel out, leaving the simplified circuit of Fig. 3-13*c*. Figure 3-13*a* and *c* are therefore equivalent. Driven by the same inputs, either circuit produces the same output as the other. So if you're loaded with NAND gates, build Fig. 3-13*a*. If your shelves are full of AND and OR gates, build Fig. 3-13*c*.

Incidentally, most people find Fig. 3-13*b* easier to analyze than Fig. 3-13*a*. For this reason, if you build Fig. 3-13*a*, draw the circuit like Fig. 3-13*b*. Anyone who sees Fig. 3-13*b* on a schematic diagram knows that the bubbled OR gate is the same as a NAND gate and that the built-up circuit is two NAND gates working into a NAND gate.

EXAMPLE 3-3

Figure 3-14 shows a circuit called a *control matrix*. At first, it looks complicated, but on closer inspection it is relatively simple because of the repetition of NAND gates. De Morgan's theorem tells us that NAND gates driving NAND gates are equivalent to AND gates driving OR gates.

The upper set of inputs T_1 to T_6 are called *timing signals;* only one of them is high at a time. T_1 goes high first, then T_2, then T_3, and so on. These signals control the rate and sequence of computer operations.

The lower set of inputs *LDA*, *ADD*, *SUB*, and *OUT* are computer instructions; only one of them is high at a time. The outputs C_P, E_P, L_M, . . . , to L_O control different registers in the computer.

Answer the following questions about the control matrix:

a. Which outputs are high when T_1 is high?
b. If T_4 and *LDA* are high, which outputs are high?
c. When T_6 and *SUB* are high, which outputs are high?

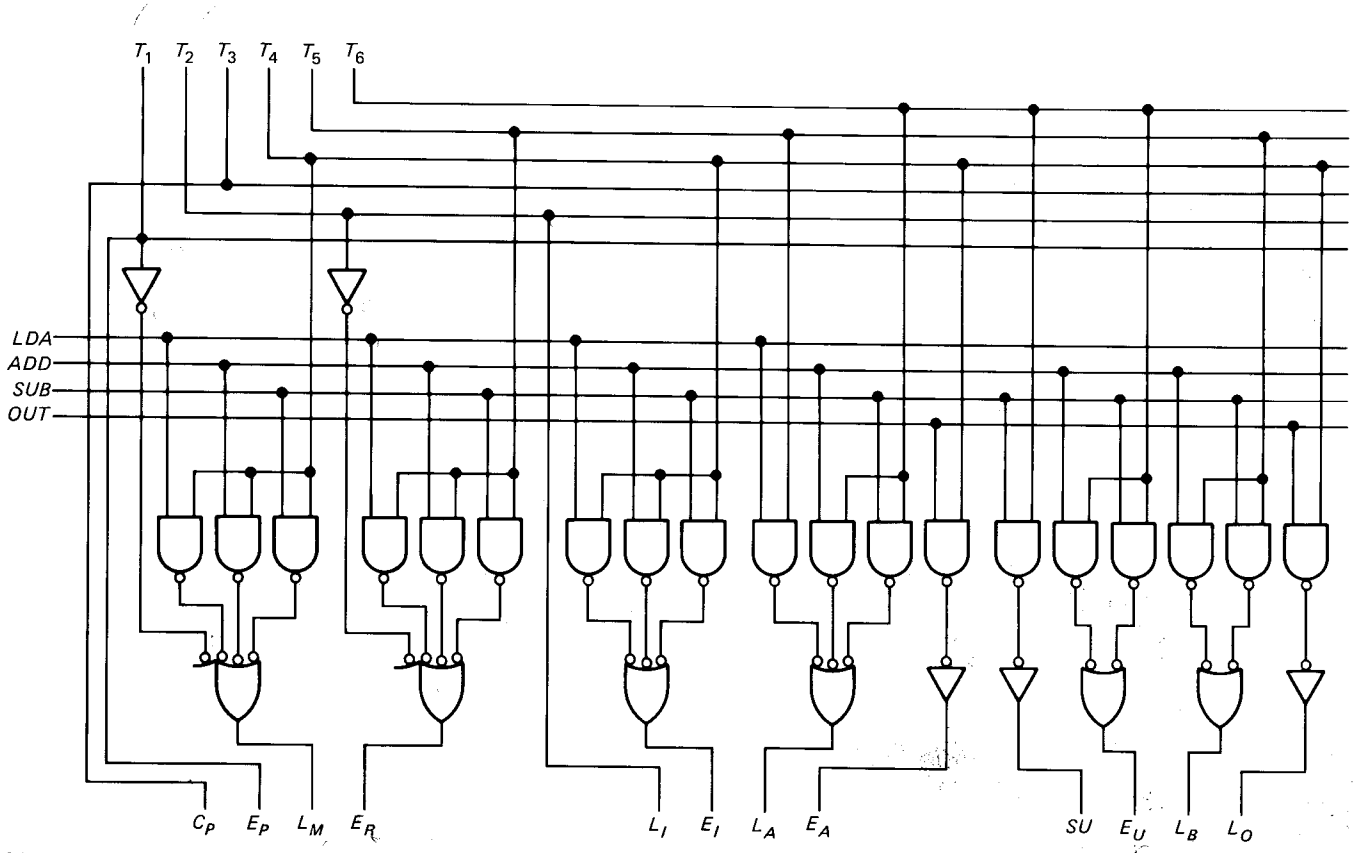


Fig. 3-14 Control matrix.

SOLUTION

a. Visualize T_1 high. You can quickly check out each gate and realize that E_P and L_M are the only high outputs.
b. This time T_4 and *LDA* are high. Check each gate and you can see that L_M and E_I are the only high outputs.
c. When T_6 and *SUB* are high, the high outputs are L_A, S_U, and E_U.

3-5 EXCLUSIVE-OR GATES

An OR gate recognizes words with one or more 1s. The EXCLUSIVE-OR gate is different; it recognizes only words that have an odd number of 1s.

Two Inputs

Figure 3-15*a* shows one way to build an EXCLUSIVE-OR gate, abbreviated XOR. The upper AND gate forms the product $\overline{A}B$, and the lower AND gate gives $A\overline{B}$. Therefore, the boolean equation is

$$Y = \overline{A}B + A\overline{B} \qquad (3\text{-}13)$$

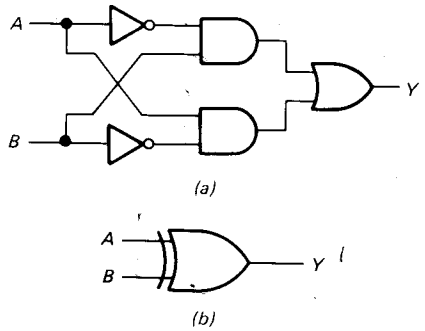


Fig. 3-15 (*a*) EXCLUSIVE-OR gate. (*b*) A 2-input EXCLUSIVE-OR gate.

Here's what the circuit does. In Fig. 3-15*a* two low inputs mean both AND gates have low outputs; so the final output is low. If A is low and B is high, the upper AND gate has a high output; therefore, the final output is high. Likewise, a high A and low B result in a final output that is high. If both inputs are high, both AND gates have low outputs and the final output is low.

Table 3-7 shows the truth table for a 2-input EXCLUSIVE-OR gate. The output is high when A or B is high but not both; this is why the circuit is known as an EXCLUSIVE-OR gate. In other words, the output is a 1 only when the inputs are different.

TABLE 3-7. TWO-INPUT XOR GATE

A	B	$\bar{A}B + A\bar{B}$
0	0	0
0	1	1
1	0	1
1	1	0

Logic Symbol and Boolean Sign

Figure 3-15*b* is the standard symbol for a 2-input XOR gate. Whenever you see this symbol, remember the action: the inputs must be different to get a high output.

A word equation for Fig. 3-15*b* is

$$Y = A \text{ XOR } B \tag{3-14}$$

In boolean algebra the sign $\oplus$ stands for XOR addition. This means that Eq. 3-14 can be written

$$Y = A \oplus B \tag{3-15}$$

Read this as "Y equals A XOR B."

Given the inputs, you can substitute and solve for the output. For instance, if both inputs are low,

$$Y = 0 \oplus 0 = 0$$

because 0 XORed with 0 gives 0. If one input is low and the other high,

$$Y = 0 \oplus 1 = 1$$

because 0 XORed with 1 produces 1. And so on.

Here's a summary of the four possible XOR additions:

$$0 \oplus 0 = 0$$
$$0 \oplus 1 = 1$$
$$1 \oplus 0 = 1$$
$$1 \oplus 1 = 0$$

Remember these four results; we will be using XOR addition when we get to arithmetic circuits.

Four Inputs

In Fig. 3-16*a* the upper gate produces $A \oplus B$, while the lower gate gives $C \oplus D$. The final gate XORs both of these sums to get

$$Y = (A \oplus B) \oplus (C \oplus D) \tag{3-16}$$

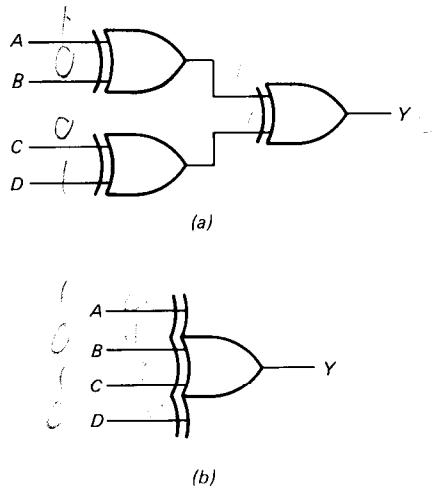


Fig. 3-16 A 4-input EXCLUSIVE-OR gate: (*a*) circuit with 2-input XOR gates; (*b*) logic symbol.

It's possible to substitute input values into the equation and solve for the output. For instance, if A through C are low and D is high,

$$\begin{aligned} Y &= (0 \oplus 0) \oplus (0 \oplus 1) \\ &= 0 \oplus 1 \\ &= 1 \end{aligned}$$

One way to get the truth table is to plow through all the input possibilities.

Alternatively, you can analyze Fig. 3-16*a* as follows. If all inputs are 0s, the first two gates have 0 outputs; so the final gate has a 0 output. If A to C are 0s and D is a 1, the upper gate has a 0 output, the lower gate has a 1 output, and the final gate has a 1 output. In this way, you can analyze the circuit action for all input words.

Table 3-8 summarizes the action. Here is an important property: each input word with an odd number of 1s produces a 1 output. For instance, the first input word to produce a 1 output is 0001; this word has an odd number of 1s. The next word with a 1 output is 0010; again an odd number of 1s. A 1 output also occurs for these words: 0100, 0111, 1000, 1011, 1101, and 1110, all of which have an odd number of 1s.

The circuit of Fig. 3-16*a* recognizes words with an odd number of 1s; it disregards words with an even number of 1s. Figure 3-16*a* is a 4-input XOR gate. In this book, we will use the abbreviated symbol of Fig. 3-16*b* to represent a 4-input XOR gate. When you see this symbol, remember the action: the circuit recognizes words with an odd number of 1s.

Any Number of Inputs

Using 2-input XOR gates as building blocks, we can make XOR gates with any number of inputs. For example, Fig.

TABLE 3-8. FOUR-INPUT XOR GATE

Comment	*A*	*B*	*C*	*D*	*Y*
Even	0	0	0	0	0
Odd	0	0	0	1	1
Odd	0	0	1	0	1
Even	0	0	1	1	0
Odd	0	1	0	0	1
Even	0	1	0	1	0
Even	0	1	1	0	0
Odd	0	1	1	1	1
Odd	1	0	0	0	1
Even	1	0	0	1	0
Even	1	0	1	0	0
Odd	1	0	1	1	1
Even	1	1	0	0	0
Odd	1	1	0	1	1
Odd	1	1	1	0	1
Even	1	1	1	1	0

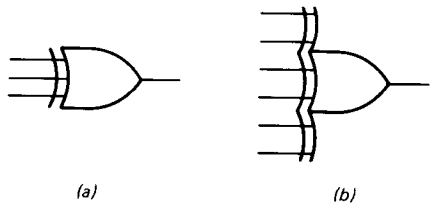


Fig. 3-17 XOR gates: (*a*) 3-input; (*b*) 6-input.

3-17*a* shows the abbreviated symbol for a 3-input XOR gate, and Fig. 3-17*b* is the symbol for a 6-input XOR gate. The final output of any XOR gate is the XOR sum of the inputs:

$$Y = A \oplus B \oplus C \cdots \qquad (3\text{-}17)$$

What you have to remember for practical work is this: an XOR gate, no matter how many inputs, recognizes only words with an odd number of 1s.

Parity

Even parity means a word has an even number of 1s. For instance, 110011 has even parity because it contains four 1s. *Odd parity* means a word has an odd number of 1s. As an example, 110001 has odd parity because it contains three 1s.

Here are two more examples:

1111 0000 1111 0011 (Even parity)
1111 0000 1111 0111 (Odd parity)

The first word has even parity because it contains ten 1s; the second word has odd parity because it contains eleven 1s.

XOR gates are ideal for testing the parity of a word. XOR gates recognize words with an odd number of 1s. Therefore, even-parity words produce a low output and odd-parity words produce a high output.

EXAMPLE 3-4

What is the output of Fig. 3-18 for each of these input words?

a. 1010 1100 1000 1100
b. 1010 1100 1000 1101

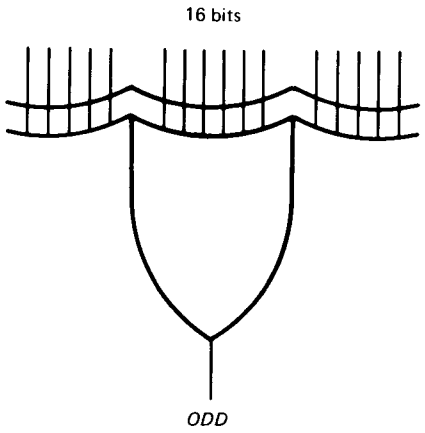


Fig. 3-18 Odd-parity tester.

SOLUTION

a. The word has seven 1s, an odd number. Therefore, the output signal is

$$ODD = 1$$

b. The word has eight 1s, an even number. Now

$$ODD = 0$$

This is an example of an odd-parity tester. An even-parity word produces a low output. An odd-parity word results in a high output.

EXAMPLE 3-5

The 7-bit register of Fig. 3-19 stores the letter A in ASCII form. What does the 8-bit output word equal?

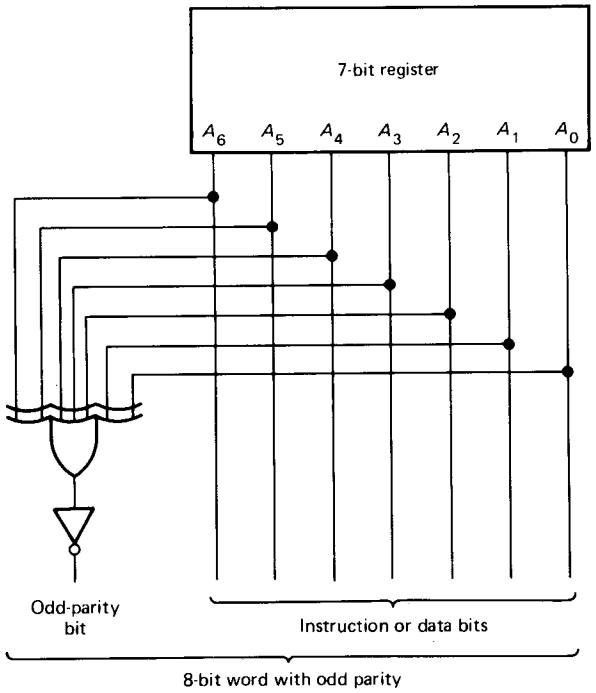


Fig. 3-19 Odd-parity generator.

SOLUTION

The ASCII code for letter A is

100 0001

(see Table 1-6 for the ASCII code). This word has an even parity, which means that the XOR gate has a 0 output. Because of the inverter, the overall output of the circuit is the 8-bit word

1100 0001

Notice that this has odd parity.

The circuit is called an *odd-parity generator* because it produces an 8-bit output word with odd parity. If the register word has even parity, 0 comes out of the XOR gate and the odd-parity bit is 1. On the other hand, if the register word has odd parity, a 1 comes out of the XOR gate and the odd-parity bit is 0. No matter what the register contents, the odd-parity bit and the register bits form a new 8-bit word that has odd parity.

What is the practical application? Because of transients, noise, and other disturbances, 1-bit errors sometimes occur in transmitted data. For instance, the letter A may be transmitted over phone lines in ASCII form:

100 0001 (A)

Somewhere along the line, one of the bits may be changed. If the X_1 bit changes, the received data will be

100 0011 (C)

Because of the 1-bit error, we receive letter C when letter A was actually sent.

One solution is to transmit an odd-parity bit along with the data word and have an XOR gate test each received word for odd parity. For instance, with a circuit like Fig. 3-19 the letter A would be transmitted as

1100 0001

An XOR gate will test this word when it is received. If no error has occurred, the XOR gate will recognize the word. On the other hand, if a 1-bit error has crept in, the XOR gate will disregard the received word and the data can be rejected.

A final point. When errors come, they are usually 1-bit errors. This is why the method described catches most of the errors in transmitted data.

EXAMPLE 3-6

What does the circuit of Fig. 3-20 do?

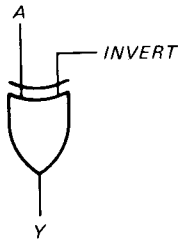


Fig. 3-20

SOLUTION

When *INVERT* = 0 and $A = 0$,

$$Y = 0 \oplus 0 = 0$$

When *INVERT* = 0 and $A = 1$,

$$Y = 0 \oplus 1 = 1$$

In either case, the output is the same as A; that is,

$$Y = A$$

for a low *INVERT* signal.

On the other hand, when *INVERT* = 1 and $A = 0$,

$$Y = 1 \oplus 0 = 1$$

When *INVERT* = 1 and $A = 1$,

$$Y = 1 \oplus 1 = 0$$

This time, the output is the complement of A. As an equation,

$$Y = \overline{A}$$

for a high *INVERT* signal.

To summarize, the circuit of Fig. 3-20 does either of two things. It transmits A when *INVERT* is 0 and $\overline{A}$ when INVERT is 1.

3-6 THE CONTROLLED INVERTER

The preceding example suggests the idea of a *controlled inverter*, a circuit that transmits a binary word or its *1's complement*.

The 1's Complement

Complement each bit in a word and the new word you get is the 1's complement. For instance, given

1100 0111

the 1's complement is

0011 1000

Each bit in the original word is inverted to get the 1's complement.

The Circuit

The XOR gates of Fig. 3-21 form a *controlled inverter* (sometimes called a programmed inverter). This circuit can transmit the register contents or the 1's complement of the register contents. As demonstrated in Example 3-6, each XOR gate acts like this. A low *INVERT* results in

$$Y_n = A_n$$

and a high *INVERT* gives

$$Y_n = \overline{A}_n$$

So each bit is either transmitted or inverted before reaching the final output.

Visualize the register contents as a word $A_7A_6 \cdots A_0$ and the final output as a word $Y_7Y_6 \cdots Y_0$. Then a low *INVERT* means

$$Y_7Y_6 \cdots Y_0 = A_7A_6 \cdots A_0$$

On the other hand, a high *INVERT* results in

$$Y_7Y_6 \cdots Y_0 = \overline{A}_7\overline{A}_6 \cdots \overline{A}_0$$

As a concrete example, suppose the register word is

$$A_7A_6 \cdots A_0 = 1110\ 0110$$

Then, a low *INVERT* gives an output word of

$$Y_7Y_6 \cdots Y_0 = 1110\ 0110$$

and a high *INVERT* produces

$$Y_7Y_6 \cdots Y_0 = 0001\ 1001$$

The controlled inverter of Fig. 3-21 is important. Later you will see how it is used in solving arithmetic and logic problems. For now, all you need to remember is the key idea. The output word from a controlled inverter equals the

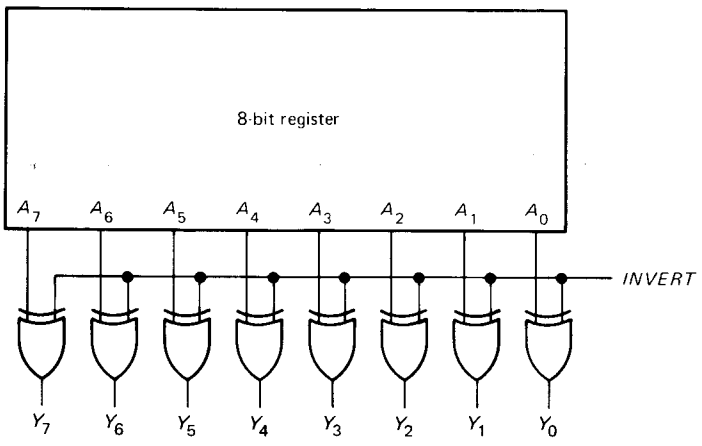


Fig. 3-21 Controlled inverter.

input word when *INVERT* is low; the output word equals the 1's complement when *INVERT* is high.

Boldface Notation

After you understand an idea, it simplifies discussions and equations if you use a symbol, letter, or other sign to represent the idea. From now on, boldface letters will stand for binary words.

For instance, instead of writing

$$A_7A_6 \cdots A_0 = 1110\ 0110$$

we can write

$$\mathbf{A} = 1110\ 0110$$

Likewise, instead of

$$Y_7Y_6 \cdots Y_0 = 0001\ 1001$$

the simpler equation

$$\mathbf{Y} = 0001\ 1001$$

can be used.

This is another example of chunking. We are replacing long strings like $A_7A_6 \cdots A_0$ and $Y_7Y_6 \cdots Y_0$ by **A** and **Y**. This chunked notation will be convenient when we get to computer analysis.

This is how to summarize the action of a controlled inverter:

$$\mathbf{Y} = \begin{cases} \mathbf{A} & \text{when } \textit{INVERT} = 0 \\ \overline{\mathbf{A}} & \text{when } \textit{INVERT} = 1 \end{cases}$$

(*Note:* A boldface letter with an overbar means that each bit in the word is complemented; if **A** is a word, $\overline{\mathbf{A}}$ is its 1's complement.)

3-7 EXCLUSIVE-NOR GATES

The EXCLUSIVE-NOR gate, abbreviated XNOR, is logically equivalent to an XOR gate followed by an inverter. For example, Fig. 3-22*a* shows a 2-input XNOR gate. Figure 3-22*b* is an abbreviated way to draw the same circuit.

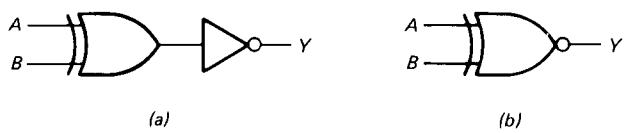


Fig. 3-22 A 2-input XNOR gate: (*a*) circuit; (*b*) abbreviated symbol.

TABLE 3-9. TWO-INPUT XNOR GATE

A	*B*	*Y*
0	0	1
0	1	0
1	0	0
1	1	1

Because of the inversion on the output side, the truth table of an XNOR gate is the complement of an XOR truth table. As shown in Table 3-9, the output is high when the inputs are the same. For this reason, the 2-input XNOR gate is ideally suited for *bit comparison*, recognizing when two input bits are identical. (Example 3-7 tells you more about bit comparison.)

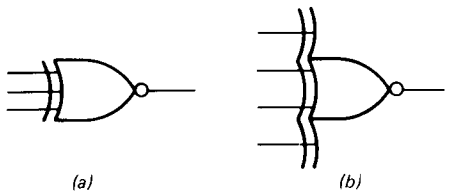


Fig. 3-23 XNOR gates: (*a*) 3-input; (*b*) 4-input.

Figure 3-23*a* is the symbol for a 3-input XNOR gate, and Fig. 3-23*b* is the 4-input XNOR gate. Because of the inversion on the output side, these XNOR gates perform the complementary function of XOR gates. Instead of recognizing odd-parity words, XNOR gates recognize even-parity words.

EXAMPLE 3-7

What does the circuit of Fig. 3-24 do?

SOLUTION

The circuit is a *word comparator;* it recognizes two identical words. Here is how it works. The leftmost XNOR gate compares A_5 and B_5; if they are the same, Y_5 is a 1. The second XNOR gate compares A_4 and B_4; if they are the same, Y_4 is a 1. In turn, the remaining XNOR gates compare the bits that are left, producing a 1 output for equal bits and a 0 output for unequal bits.

If the words **A** and **B** are identical, all XNOR gates have high outputs and the AND gate has a high *EQUAL*. If words **A** and **B** differ in one or more bit positions, the AND gate has a low *EQUAL*.

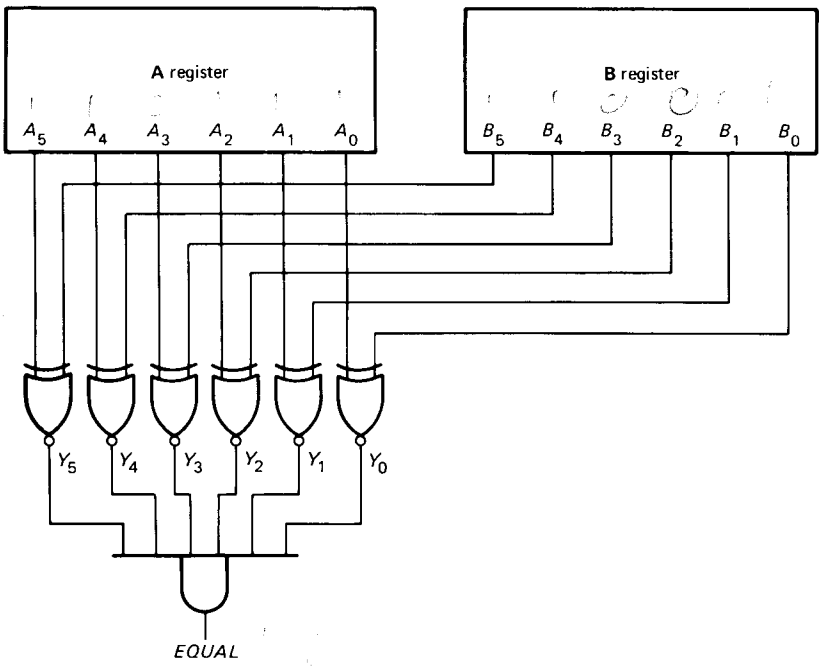


Fig. 3-24 Word comparator.

GLOSSARY

controlled inverter This circuit produces the 1's complement of the input word. One application is binary subtraction. It is sometimes called a programmed inverter.

De Morgan's theorems The first theorem says that a NOR gate is equivalent to a bubbled AND gate. The second theorem says that a NAND gate is equivalent to a bubbled OR gate.

even parity An even number of 1s in a binary word.

NAND *gate* Equivalent to an AND gate followed by an inverter. All inputs must be high to get a low output.

NOR *gate* Equivalent to an OR gate followed by an inverter. All inputs must be low to get a high output.

odd parity An odd number of 1s in a binary word.

parity generator A circuit that produces either an odd- or even-parity bit to go along with the data.

XNOR *gate* Equivalent to an EXCLUSIVE-OR gate followed by an inverter. The output is high only when the input word has even parity.

XOR *gate* An EXCLUSIVE-OR gate. It has a high output only when the input word has odd parity. For a 2-input XOR gate, the output is high only when the inputs are different.

SELF-TESTING REVIEW

Read each of the following and provide the missing words. Answers appear at the beginning of the next question.

1. A NOR gate has two or more input signals. All inputs must be _____ to get a high output. A NOR gate recognizes only the input word whose bits are _____. The NOR gate is logically equivalent to an OR gate followed by an _____.
2. (*low, 0s, inverter*) De Morgan's first theorem says that a NOR gate is equivalent to a bubbled _____ gate.
3. (AND) A NAND gate is equivalent to an AND gate followed by an inverter. All inputs must be _____ to get a low output. De Morgan's second theorem says that a NAND gate is equivalent to a bubbled _____ gate.
4. (*high*, OR) An XOR gate recognizes only words with an _____ number of 1s. The 2-input XOR gate has a high output only when the input bits are _____. XOR gates are ideal for testing parity because even-parity words produce a _____ output and odd-parity words produce a _____ output.
5. (*odd, different, low, high*) An odd-parity generator produces an odd-parity bit to go along with the data.

The parity of the transmitted data is _____. An XOR gate can test each received word for parity, rejecting words with _____ parity.

6. (*odd, even*) A controlled inverter is a logic circuit that transmits a binary word or its _____ complement.

7. (*1's*) The EXCLUSIVE-NOR gate is equivalent to an XOR gate followed by an inverter. Because of this, even-parity words produce a high output.

PROBLEMS

3-1. In Fig. 3-25*a* the two inputs are connected together. If A is low, what is Y? If A is high, what is Y? Does the circuit act like a noninverter or an inverter?

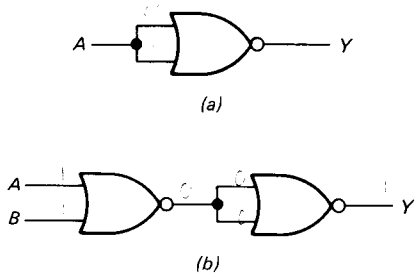


Fig. 3-25

3-2. What is the output in Fig. 3-25*b* if both inputs are low? If one is low and the other high? If both are high? Does the circuit act like an OR gate or an AND gate?

3-3. Figure 3-26 shows a NOR-gate *crossbar switch*. If all X and Y inputs are high, which of the Z outputs is high? If all inputs are high except X_1 and Y_2, which Z output is high? If X_2 and Y_0 are low and all other inputs are high, which Z output is high?

3-4. In Fig. 3-26, you want Z_7 to be 1 and all other Z outputs to be 0. What values must the X and Y inputs have?

3-5. The outputs in Fig. 3-27 are cross-coupled back to the inputs of the NOR gates. If $R = 0$ and $S = 1$, what do Q and $\overline{Q}$ equal?

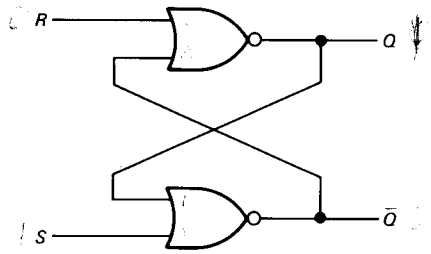


Fig. 3-27 Cross-coupled NOR gates.

3-6. If $R = 1$ and $S = 0$ in Fig. 3-27, what does Q equal? $\overline{Q}$?

3-7. Prove that Fig. 3-28*a* and *b* are equivalent.

3-8. What is the output in Fig. 3-28*a* if all inputs are 0s. If all inputs are 1s?

3-9. What is the output in Fig. 3-28*b* if all inputs are 0s. If all inputs are 1s?

3-10. A NOR has 6 inputs. How many input words are in its truth table? What is the only input word that produces a 1 output?

3-11. In Fig. 3-28*a* how many input words are there in the truth table?

3-12. What is the output in Fig. 3-29 if all inputs are low? If all inputs are high?

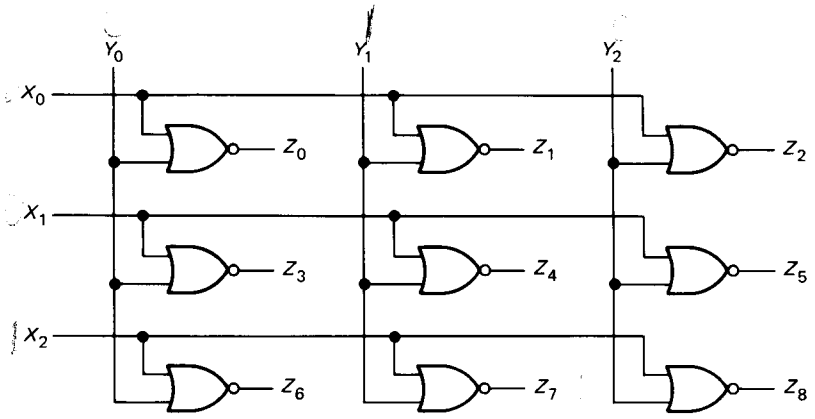


Fig. 3-26 NOR-gate crossbar switch.

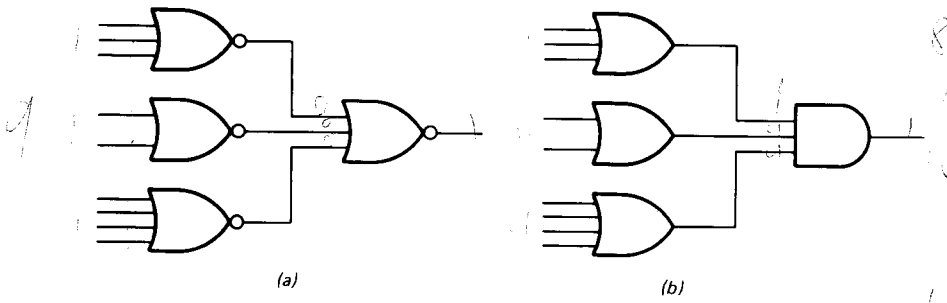


Fig. 3-28

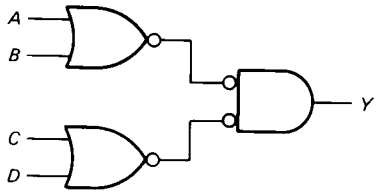


Fig. 3-29

3-13. How many words are in the truth table of Fig. 3-29. What is the value of Y for each of the following?

a. ABCD = 0011
b. ABCD = 0110
c. ABCD = 1001
d. ABCD = 1100

3-14. Which ABCD input words does the circuits of Fig. 3-29 recognize?

3-15. In Fig. 3-30*a* the two inputs are connected together. If $A = 0$ what does Y equal? If $A = 1$, what does Y equal? Does the circuit act like a noninverter or an inverter?

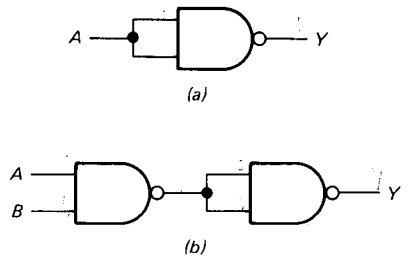


Fig. 3-30

3-16. What is the output in Fig. 3-30*b* if both inputs are low? If one input is low and the other high? If both are high? Does the circuit act like an OR gate or an AND gate?

3-17. Suppose the NOR gates of Fig. 3-26 are replaced by NAND gates. Then you've got a NAND-gate crossbar switch.

a. If all X and Y inputs are low, which Z output is low?
b. If all inputs are low except X_2 and Y_1, which Z output is low?
c. If all inputs are low except X_0 and Y_2, which Z output is low?
d. To get a low Z_8 output, which inputs must be high?

3-18. In Fig. 3-31, what are the outputs if $R = 0$ and $S = 1$?

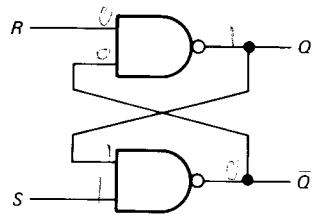


Fig. 3-31 Cross-coupled NAND gates.

3-19. If $R = 1$ and $S = 0$ in Fig. 3-31, what does Q equal? $\overline{Q}$?

3-20. What is the output in Fig. 3-32*a* if all inputs are 0s? If all inputs are 1s?

3-21. How many input words are there in the truth table of Fig. 3-32*a*?

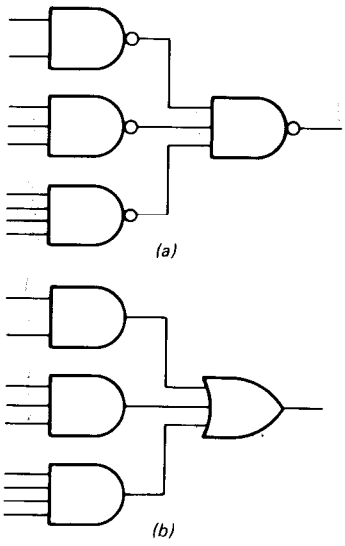


Fig. 3-32

3-22. Prove that Fig. 3-32*a* and *b* are equivalent.

3-23. What is the output in Fig. 3-33 if all inputs are low? If they are all high?

3-24. How many words are in the truth table of Fig. 3-33? What does Y equal for each of the following:

a. ABCDE = 00111
b. ABCDE = 10110
c. ABCDE = 11010
d. ABCDE = 10101

3-25. In Fig. 3-34 the inputs are T_4, *JMP*, *JAM*, *JAZ*, A_M, and A_Z; the output is L_P. What is the output for each of these input conditions?

a. All inputs are 0s.
b. All inputs are low except T_4 and *JMP*.
c. All inputs are low except T_4, *JAZ*, and A_Z.
d. The only high inputs are T_4, *JAM*, and A_M.

3-26. Figure 3-35 shows the control matrix discussed in Example 3-3. Only one of the timing signals T_1 to T_6 is high at a time. Also, only one of the instructions, *LDA* to *OUT*, is high at a time. Which are the high outputs for each of the following conditions?

a. T_1 high
b. T_2 high
c. T_3 high
d. T_4 and *LDA* high
e. T_5 and *LDA* high
f. T_4 and *ADD* high
g. T_5 and *ADD* high
h. T_6 and *ADD* high
i. T_4 and *SUB* high
j. T_5 and *SUB* high
k. T_6 and *SUB* high
l. T_4 and *OUT* high

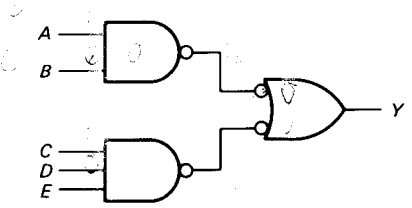


Fig. 3-33

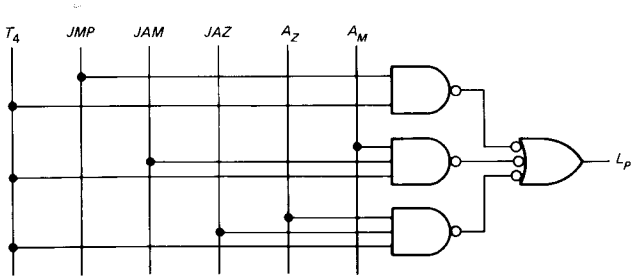


Fig. 3-34

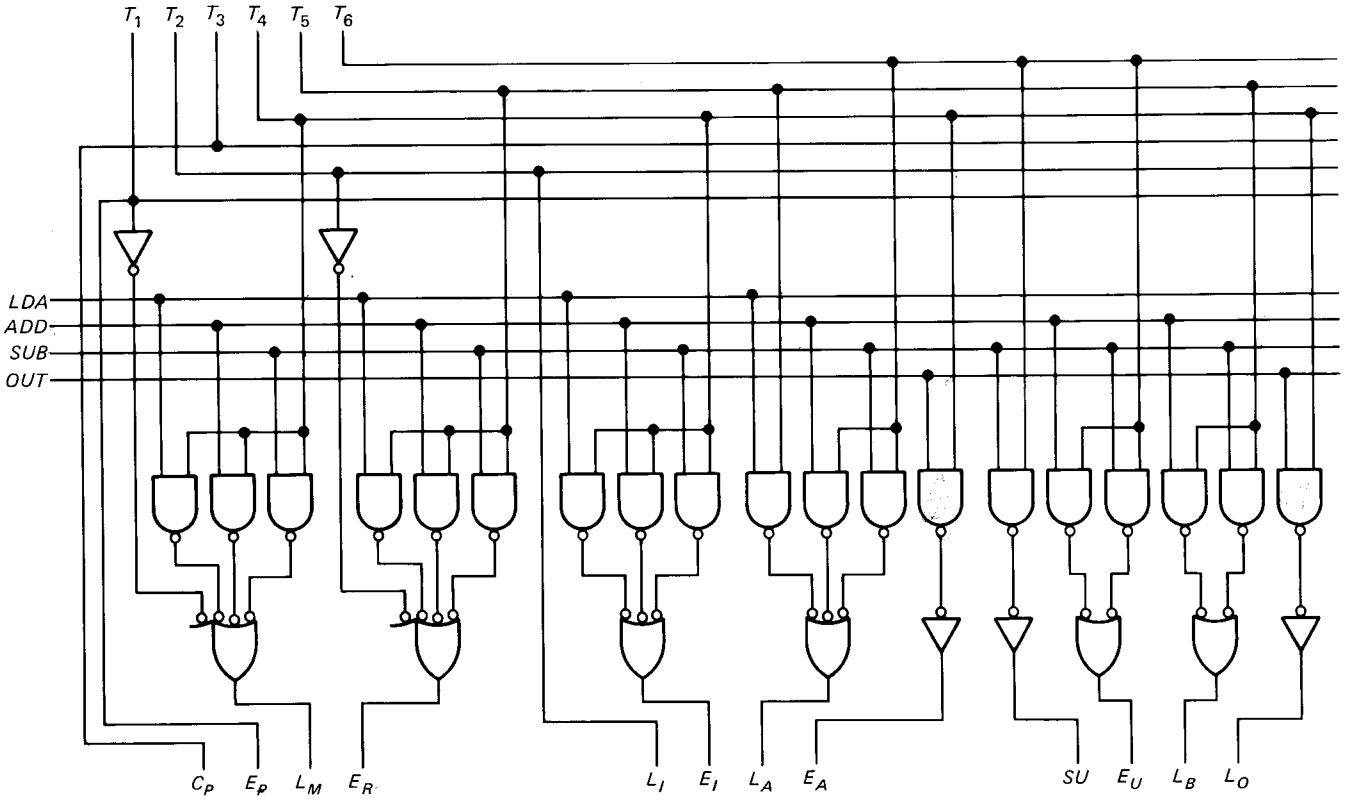


Fig. 3-35 Control matrix.

3-27. Figure 3-36 shows a binary–to–Gray-code converter. (Gray code is a special code used in analog-to-digital conversions.) The input word is $X_4X_3 \cdots X_0$, and the output word is $Y_4Y_3 \cdots Y_0$. What does the output word equal for each of these inputs?

a. $X_4X_3 \cdots X_0 = 10011$
b. $X_4X_3 \cdots X_0 = 01110$
c. $X_4X_3 \cdots X_0 = 10101$
d. $X_4X_3 \cdots X_0 = 11100$

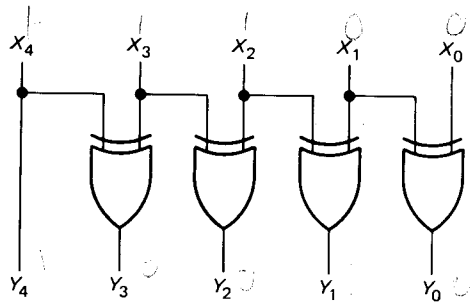


Fig. 3-36 Binary–to–Gray-code converter.

3-28. How many input words are there in the truth table of an 8-input XOR gate?

3-29. How can you modify Fig. 3-19 so that it produces an 8-bit output word with even parity?

3-30. In the controlled inverter of Fig. 3-21, what is the output word *Y* for each of these conditions?

a. **A** = 1100 1111 and *INVERT* = 0
b. **A** = 0101 0001 and *INVERT* = 1
c. **A** = 1110 1000 and *INVERT* = 1
d. **A** = 1010 0101 and *INVERT* = 0

3-31. The inputs *A* and *B* of Fig. 3-37 produce outputs of *CARRY* and *SUM*. What are the values of *CARRY* and *SUM* for each of these inputs?

a. $A = 0$ and $B = 0$
b. $A = 0$ and $B = 1$
c. $A = 1$ and $B = 0$
d. $A = 1$ and $B = 1$

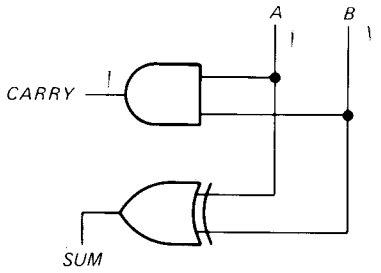


Fig. 3-37

3-32. In Fig. 3-37, what is the boolean equation for *CARRY*? For *SUM*?

3-33. What is the 1's complement for each of these numbers?

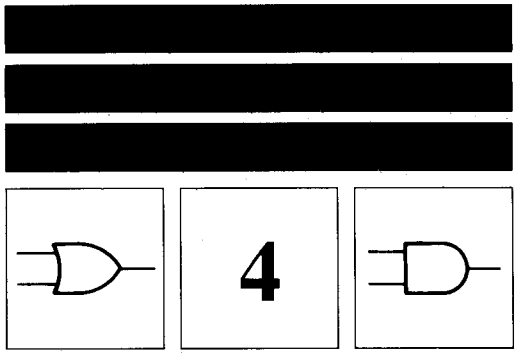
a. 1100 0011
b. 1010 1111 0011
c. 1110 0001 1010 0011
d. 0000 1111 0010 1101

3-34. What is the output of a 16-input XNOR gate for each of these input words?

a. 0000 0000 0000 1111
b. 1111 0101 1110 1100
c. 0101 1100 0001 0011
d. 1111 0000 1010 0110

3-35. The boolean equation for a certain logic circuit is $Y = AB + CD + AC$. What does Y equal for each of the following:

a. ABCD = 0000
b. ABCD = 0101
c. ABCD = 1010
d. ABCD = 1001



TTL CIRCUITS

In 1964 Texas Instruments introduced *transistor-transistor logic* (TTL), a widely used family of digital devices. TTL is fast, inexpensive, and easy to use. This chapter concentrates on TTL because once you are familiar with it, you can branch out to other logic families and technologies.

4-1 DIGITAL INTEGRATED CIRCUITS

Using advanced photographic techniques, a manufacturer can produce miniature circuits on the surface of a *chip* (a small piece of semiconductor material). The finished network is so small you need a microscope to see the connections. Such a circuit is called an *integrated circuit* (IC) because the components (transistors, diodes, resistors) are an integral part of the chip. This is different from a discrete circuit, in which the components are individually connected during assembly.

Levels of Integration

Small-scale integration (SSI) refers to ICs with fewer than 12 gates on the same chip. *Medium-scale integration* (MSI) means from 12 to 100 gates per chip. And *large-scale integration* (LSI) refers to more than 100 gates per chip. The typical microcomputer has its microprocessor, memory, and I/O circuits on LSI chips; a number of SSI and MSI chips are used to support the LSI chips.

Technologies and Families

The two basic technologies for manufacturing digital ICs are *bipolar* and *MOS*. The first fabricates bipolar transistors on a chip; the second, MOSFETS. Bipolar technology is preferred for SSI and MSI because it is faster. MOS technology dominates the LSI field because more MOSFETs can be packed on the same chip area.

A digital family is a group of compatible devices with the same logic levels and supply voltages ("compatible" means that you can connect the output of one device to the input of another). Compatibility permits a large number of different combinations.

Bipolar Families

In the bipolar category are these basic families:

DTL	Diode-transistor logic
TTL	Transistor-transistor logic
ECL	Emitter-coupled logic

DTL uses diodes and transistors; this design, once popular, is now obsolete. TTL uses transistors almost exclusively; it has become the most popular family of SSI and MSI chips. ECL, the fastest logic family, is used in high-speed applications.

MOS Families

In the MOS category are these families:

PMOS	p-Channel MOSFETs
NMOS	n-Channel MOSFETs
CMOS	Complementary MOSFETs

PMOS, the oldest and slowest type, is becoming obsolete. NMOS dominates the LSI field, being used for microprocessors and memories. CMOS, a push-pull arrangement of n- and p-channel MOSFETs, is extensively used where low power consumption is needed, as in pocket calculators, digital wristwatches, etc.

4-2 7400 DEVICES

The 7400 series, a line of TTL circuits introduced by Texas Instruments in 1964, has become the most widely used of all bipolar ICs. This TTL family contains a variety of SSI and MSI chips that allow you to build all kinds of digital circuits and systems.

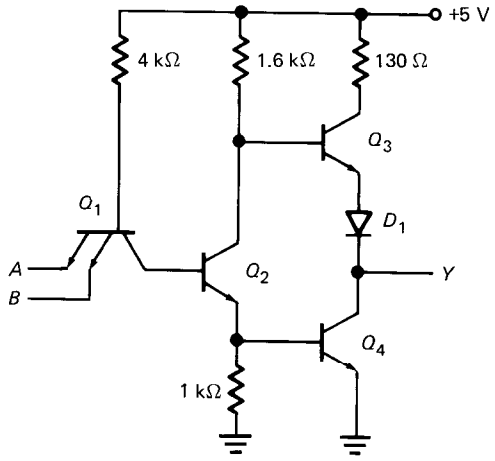


Fig. 4-1 Standard TTL NAND gate.

Standard TTL

Figure 4-1 shows a TTL NAND gate. The multiple-emitter input transistor is typical of all the gates and circuits in the 7400 series. Each emitter acts like a diode; therefore, Q_1 and the 4-kΩ resistor act like a 2-input AND gate. The rest of the circuit inverts the signal; therefore, the overall circuit acts like a 2-input NAND gate.

The output transistors (Q_3 and Q_4) form a totem-pole connection, typical of most TTL devices. Either one or the other is on. When Q_3 is on, the output is high; when Q_4 is on, the output is low. The advantage of a totem-pole connection is its low output impedance.

Ideally, the input voltages *A* and *B* are either low (grounded) or high (5 V). If *A* or *B* is low, Q_1 saturates. This reduces the base voltage of Q_2 to almost zero. Therefore, Q_2 cuts off, forcing Q_4 to cut off. Under these conditions, Q_3 acts like an emitter follower and couples a high voltage to the output.

On the other hand, when both *A* and *B* are high, the collector diode of Q_1 goes into forward conduction; this forces Q_2 and Q_4 into saturation, producing a low output. Table 4-1 summarizes all input and output conditions.

Incidentally, without diode D_1 in the circuit, Q_3 would conduct slightly when the output is low. To prevent this, the diode is inserted; its voltage drop keeps the base-emitter diode of Q_3 reverse-biased. In this way, only Q_4 conducts when the output is low.

TABLE 4-1. TWO-INPUT NAND GATE

A	*B*	*Y*
0	0	1
0	1	1
1	0	1
1	1	0

Totem-Pole Output

Why are totem-pole transistors used? Because they produce a low output impedance. Either Q_3 acts like an emitter follower (high output) or Q_4 is saturated (low output). Either way, the output impedance is very low. This is important because it reduces the switching time. In other words, when the output changes from low to high, or vice versa, the low output impedance implies a short *RC* time constant; this short time constant means that the output voltage can change quickly from one state to the other.

Propagation Delay Time and Power Dissipation

Two quantities needed for our later discussions are power dissipation and propagation delay time. A standard TTL gate has a power dissipation of about 10 mW. It may vary from this value because of signal levels, tolerances, etc., but on the average, it's 10 mW per gate.

The propagation delay time is the amount of time it takes for the output of a gate to change after the inputs have changed. The propagation delay time of a TTL gate is in the vicinity of 10 ns.

Device Numbers

By varying the design of Fig. 4-1 manufacturers can alter the number of inputs and the logic function. The multiple-emitter inputs and the totem-pole outputs are still used, no matter what the design. (The only exception is an open collector, discussed later.)

Table 4-2 lists some of the 7400-series TTL gates. For instance, the 7400 is a chip with four 2-input NAND gates in one package. Similarly, the 7402 has four 2-input NOR gates, the 7404 has six inverters, and so on.

TABLE 4-2. STANDARD TTL

Device number	Description
7400	Quad 2-input NAND gates
7402	Quad 2-input NOR gates
7404	Hex inverter
7408	Quad 2-input AND gates
7410	Triple 3-input NAND gates
7411	Triple 3-input AND gates
7420	Dual 4-input NAND gates
7421	Dual 4-input AND gates
7427	Triple 3-input NOR gates
7430	8-input NAND gate
7486	Quad 2-input XOR gates

5400 Series

Any device in the 7400 series works over a temperature range of 0° to 70°C and over a supply range of 4.75 to 5.25 V. This is adequate for commercial applications. The 5400 series, developed for the military applications, has the same logic functions as the 7400 series, except that it works over a temperature range of −55 to 125°C and over a supply range of 4.5 to 5.5 V. Although 5400-series devices can replace 7400-series devices, they are rarely used commercially because of their much higher cost.

High-Speed TTL

The circuit of Fig. 4-1 is called *standard TTL*. By decreasing the resistances a manufacturer can lower the internal time constants; this decreases the propagation delay time. The smaller resistances, however, increase the power dissipation. This variation is known as *high-speed TTL*. Devices of this type are numbered 74H00, 74H01, 74H02, and so on. A high-speed TTL gate has a power dissipation around 22 mW and a propagation delay time of approximately 6 ns.

Low-Power TTL

By increasing the internal resistances a manufacturer can reduce the power dissipation of TTL gates. Devices of this type are called *low-power TTL* and are numbered 74L00, 74L01, 74L02, etc. These devices are slower than standard TTL because of the larger internal time constants. A low-power TTL gate has a power dissipation of approximately 1 mW and a propagation delay time around 35 ns.

Schottky TTL

With standard TTL, high-speed TTL, and low-power TTL, the transistors go into saturation causing extra carriers to flood the base. If you try to switch this transistor from saturation to cutoff, you have to wait for the extra carriers to flow out of the base; the delay is known as the *saturation delay time*.

One way to reduce saturation delay time is with Schottky TTL. The idea is to fabricate a Schottky diode along with each bipolar transistor of a TTL circuit, as shown in Fig. 4-2. Because the Schottky diode has a forward voltage of only 0.4 V, it prevents the transistor from saturating fully. This virtually eliminates saturation delay time, which means better switching speed. This variation is called *Schottky TTL;* the devices are numbered 74S00, 74S01, 74S02, and so forth.

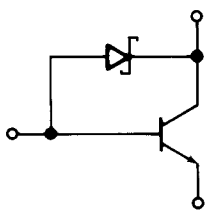


Fig. 4-2 Schottky diode prevents transistor saturation.

Schottky TTL devices are very fast, capable of operating reliably at 100 MHz. The 74S00 has a power dissipation around 20 mW per gate and a propagation delay time of approximately 3 ns.

Low-Power Schottky TTL

By increasing internal resistances as well as using Schottky diodes manufacturers have come up with the best compromise between low power and high speed: *low-power Schottky TTL*. Devices of this type are numbered 74LS00, 74LS01, 74LS02, etc. A low-power Schottky gate has a power dissipation of around 2 mW and a propagation delay time of approximately 10 ns, as shown in Table 4-3.

Standard TTL and low-power Schottky TTL are the mainstays of the digital designer. In other words, of the five TTL types listed in Table 4-3, standard TTL and low-power Schottky TTL have emerged as the favorites of the digital designers. You will see them used more than any other bipolar types.

4-3 TTL CHARACTERISTICS

7400-series devices are guaranteed to work reliably over a temperature range of 0 to 70°C and over a supply range of 4.75 to 5.25 V. In the discussion that follows, *worst case* means that the parameters (characteristics like maximum input current, minimum output voltage, and so on) are measured under the worst conditions of temperature and voltage—maximum temperature and minimum voltage for some parameters, minimum temperature and maximum voltage for others, or whatever combination produces the worst values.

Floating Inputs

When a TTL input is low or grounded, a current I_E (conventional direction) exists in the emitter, as shown in

TABLE 4-3. TTL POWER-DELAY VALUES

Type	Power, mW	Delay time, ns
Low-power	1	35
Low-power Schottky	2	10
Standard	10	10
High-speed	22	6
Schottky	20	3

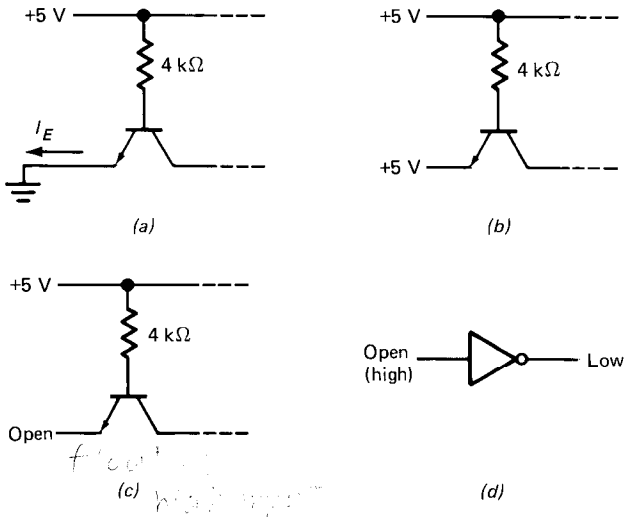


Fig. 4-3 Open or floating input is the same as a high input.

Fig. 4-3*a*. On the other hand, when a TTL input is high (Fig. 4-3*b*), the emitter diode cuts off and the emitter current is approximately zero.

When a TTL input is floating (unconnected), as shown in Fig. 4-3*c*, no emitter current is possible. Therefore, a floating TTL input is equivalent to a high input. In other words, Fig. 4-3*c* produces the same output as Fig. 4-3*b*. This is important to remember. In building circuits *any floating TTL input will act like a high input.*

Figure 4-3*d* emphasizes the point. The input is floating and is equivalent to a high input; therefore, the output of the inverter is low.

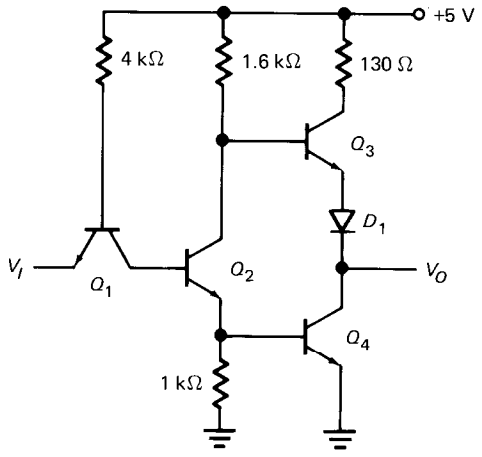


Fig. 4-4 TTL inverter.

Worst-Case Input Voltages

Figure 4-4 shows a TTL inverter with an input voltage of V_I and an output voltage of V_O. When V_I is 0 V (grounded), the output voltage is high. With TTL devices, we can raise V_I to 0.8 V and still have a high output. The maximum low-level input voltage is designated V_{IL}. Data sheets list this worst-case low input as

$$V_{IL} = 0.8 \text{ V}$$

Take the other extreme. Suppose V_I is 5 V in Fig. 4-4. This is a high input; therefore, the output of the inverter is low. V_I can decrease all the way down to 2 V, and the output will still be low. Data sheets list this worst-case high input as

$$V_{IH} = 2 \text{ V}$$

In other words, any input voltage from 2 to 5 V is a high input for TTL devices.

Worst-Case Output Voltages

Ideally, 0 V is the low output, and 5 V is the high output. We cannot attain these ideal values because of internal voltage drops. When the output is low in Fig. 4-4, Q_4 is saturated and has a small voltage drop across it. With TTL devices, any voltage from 0 to 0.4 V is a low output.

When the output is high, Q_3 acts like an emitter follower. Because of the drop across Q_3, D_1, and the 130-Ω resistor, the output is less than 5 V. With TTL devices, a high output is between 2.4 and 3.9 V, depending on the supply voltage, temperature, and load.

This means that the worst-case output values are

$$V_{OL} = 0.4 \text{ V} \qquad V_{OH} = 2.4 \text{ V}$$

Table 4-4 summarizes the worst-case values. Remember that they are valid over the temperature range (0 to 70°C) and supply range (4.75 to 5.25 V).

Compatibility

The values shown in Table 4-4 indicate that TTL devices are compatible. This means that the output of a TTL device can drive the input of another TTL device, as shown in Fig. 4-5*a*. To be specific, Fig. 4-5*b* shows a low TTL output (0 to 0.4 V). This is low enough to drive the second TTL device because any input less than 0.8 V is a low input.

TABLE 4-4. TTL STATES (WORST CASE)

	Output, V	Input, V
Low	0.4	0.8
High	2.4	2

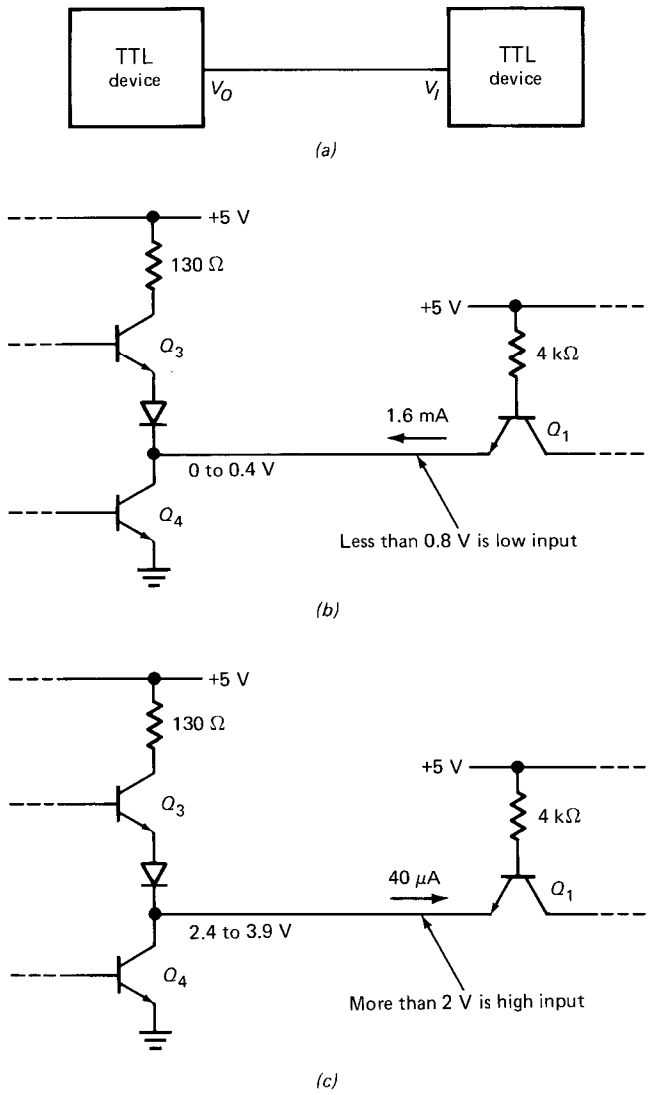


Fig. 4-5 Sourcing and sinking current.

Similarly, Fig. 4-5*c* shows a high TTL output (2.4 to 3.9 V). This is more than enough to drive the second TTL because any input greater than 2 V is a high input.

Noise Margin

In the worst case, there is a margin of 0.4 V between the driver and the load in Fig. 4-5*b* and *c*. This difference, called the *noise margin,* represents protection against noise. In other words, the connecting wire between a TTL driver and a TTL load may pick up stray noise voltages. As long as these induced voltages are less than 0.4 V, we get no false triggering of the TTL load.

Sourcing and Sinking

When a standard TTL output is low (Fig. 4-5*b*), an emitter current of approximately 1.6 mA (worst case) exists in the direction shown. The charges flow from the emitter of Q_1 to the collector of Q_4. Because it is saturated, Q_4 acts like a *current sink;* charges flow through it to ground like water flowing down a drain.

On the other hand, when a standard TTL output is high (Fig. 4-5*c*), a reverse emitter current of 40 μA (worst case) exists in the direction shown. Charges flow from Q_3 to the emitter of Q_1. In this case, Q_3 is acting like a *source.*

Data sheets lists the worst-case input currents as

$$I_{IL} = -1.6 \text{ mA} \qquad I_{IH} = 40 \text{ μA}$$

The minus sign indicates that the current is out of the device; plus means the current is into the device. All data sheets use this convention.

Standard Loading

A TTL device can source current (high output) or it can sink current (low output). Data sheets of standard TTL devices indicate that any 7400-series device can sink up to 16 mA, designated as

$$I_{OL} = 16 \text{ mA}$$

and can source up to 400 μA, designated

$$I_{OH} = -400 \text{ μA}$$

(Again, a minus sign means that the current is out of the device and a plus sign means that it's into the device.)

A single TTL load has a low-level input current of 1.6 mA (Fig., 4-5*b*) and a high-level input current of 40 μA (Fig. 4-5*c*). Since the maximum output currents are 10 times as large, we can connect up to 10 TTL emitters to any TTL output.

Figure 4-6*a* illustrates a low output. Here you see the TTL driver sinking 16 mA, the sum of 10 TTL load currents. In this state, the output voltage is guaranteed to be 0.4 V or less. If you try connecting more than 10 emitters, the output voltage may rise above 0.4 V.

Figure 4-6*b* shows a high output with the driver sourcing 400 μA for 10 TTL loads of 40 μA each. For this maximum loading, the output voltage is guaranteed to be 2.4 V or more under worst-case conditions.

Loading Rules

The maximum number of TTL emitters that can be reliably driven under worst-case conditions is called the *fanout.* With standard TTL, the fanout is 10, as shown in Fig. 4-6. Sometimes, we may want to use a standard TTL device to drive low-power Schottky devices. In this case, the fanout increases because low-power Schottky devices have less input current.

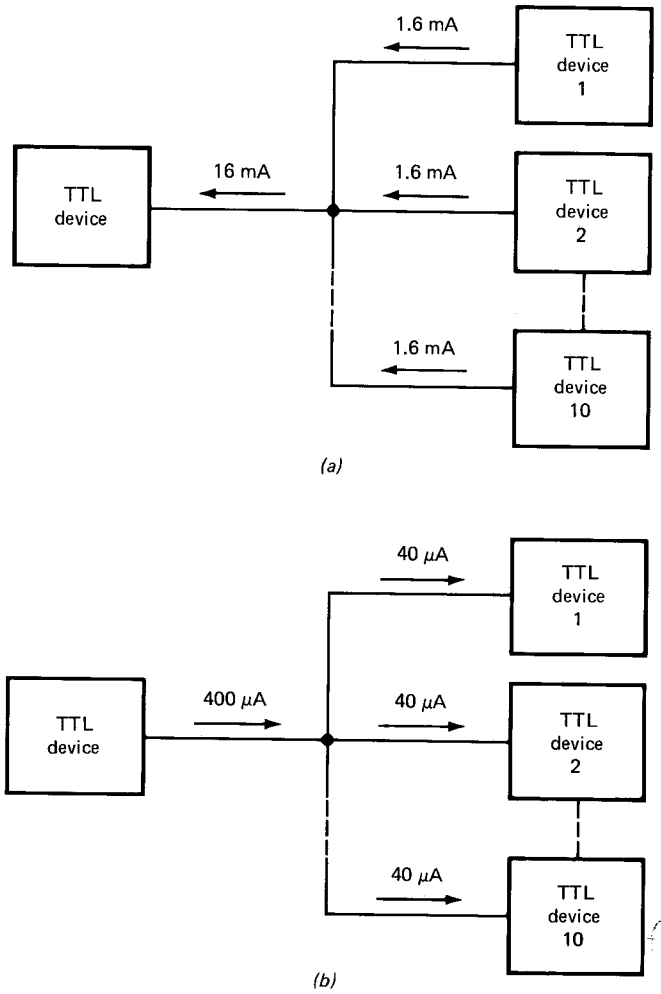


Fig. 4-6 Fanout of standard TTL devices: (*a*) low output; (*b*) high output.

By examining data sheets for the different TTL types we can calculate the fanout for all possible combinations. Table 4-5 summarizes these fanouts, which may be useful if you ever have to mix TTL types.

Read Table 4-5 as follows. The series numbers have been abbreviated; 74 stands for 7400 series, 74H for 74H00 series, and so forth. Drivers are on the left and loads on the right. Pick the driver, pick the load, and read the fanout at the intersection of the two. For instance, the fanout of a standard device (74) driving low-power Schottky devices (74LS) is 20. As another example, the fanout of a low-power device (74L) driving high-speed devices (74H) is only 1.

TABLE 4-5. FANOUTS

TTL driver	TTL load				
	74	74H	74L	74S	74LS
74	10	8	40	8	20
74H	12	10	50	10	25
74L	2	1	20	1	10
74S	12	10	100	10	50
74LS	5	4	40	4	20

4-4 TTL OVERVIEW

Let's take a look at the logic functions available in the 7400 series. This overview will give you an idea of the variety of gates and circuits found in the TTL family. As guide, Appendix 3 lists some of the 7400-series devices. You will find it useful when looking for a device number or logic function.

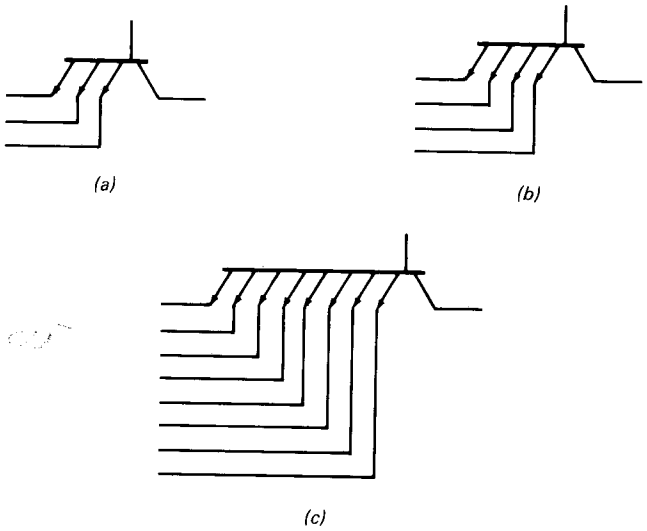


Fig. 4-7 Three, four, and eight inputs.

NAND Gates

To begin with, the NAND gate is the backbone of the entire series. All devices in the 7400 series are derived from the 2-input NAND gate shown in Fig. 4-1. To produce 3-, 4-, and 8-input NAND gates the manufacturer uses 3-, 4-, and 8-emitter transistors, as shown in Fig. 4-7. Because they are so basic, NAND gates are the least expensive devices in the 7400 series.

NOR Gates

To get other logic functions the manufacturer modifies the basic NAND-gate design. For instance, Fig. 4-8 shows a 2-input NOR gate. Q_1, Q_2, Q_3, and Q_4 are the same as in the basic design. Q_5 and Q_6 have been added to produce ORing. Notice that Q_2 and Q_6 are in parallel, the key to the ORing followed by inversion to get NORing.

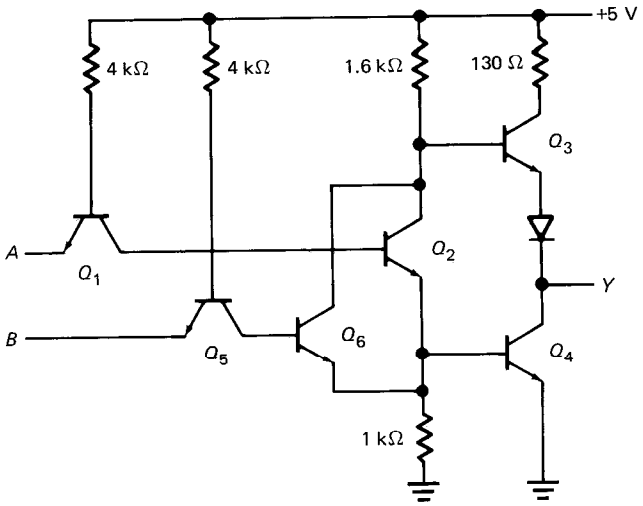


Fig. 4-8 TTL NOR gate.

When A and B are both low, Q_1 and Q_5 are saturated; this cuts off Q_2 and Q_6. Then Q_3 acts like an emitter follower and we get a high output.

If A or B or both are high, Q_1 or Q_5 or both are cut off, forcing Q_2 or Q_6 or both to turn on. When this happens, Q_4 saturates and pulls the output down to a low voltage.

With more transistors, manufacturers can produce 3- and 4-input NOR gates. (A TTL 8-input NOR gate is not available.)

AND and OR Gates

To produce the AND function, another common-emitter stage is inserted before the totem-pole output of the basic NAND gate design. The extra inversion converts the NAND gate to an AND gate. Similarly, another CE stage can be inserted before the totem-pole output of Fig. 4-8; this converts the NOR gate to an OR gate.

Buffer-Drivers

A *buffer* is a device that isolates two other devices. Typically, a buffer has a high input impedance and a low output impedance. In terms of digital ICs, this means a low input current and a high output current.

Since the output current of a standard TTL gate can be 10 times the input current, a basic gate does a certain amount of buffering (isolating). But it's only when the manufacturer optimizes the design for high output currents that we call a device a buffer or driver.

As an example, the 7437 is a quad 2-input NAND buffer, meaning four 2-input NAND gates optimized to get high output currents. Each gate has the following worst-case values of input and output currents:

$$I_{IL} = -1.6\text{ mA} \qquad I_{IH} = 40\ \mu\text{A}$$
$$I_{OL} = 48\text{ mA} \qquad I_{OH} = -1.2\text{ mA}$$

The input currents are the same as those of a standard NAND gate, but the output currents are 3 times as high, which means that the 7437 can drive heavier loads.

Appendix 3 includes several other buffer-drivers.

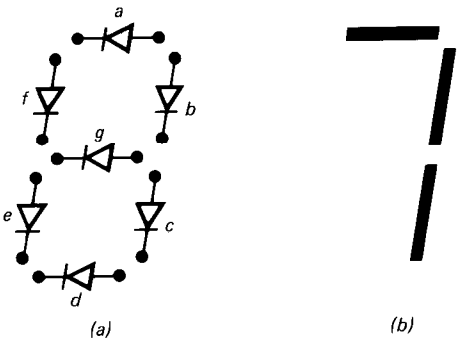


Fig. 4-9 Seven-segment display.

Encoders and Decoders

A number of TTL chips are available for encoding and decoding data. For instance, the 74147 is a decimal-to-BCD encoder. It has 10 input lines (decimal) and 4 output lines (BCD). As another example, the 74154 is a 1-of-16 decoder. It has 4 input lines (binary) and 16 output lines (hexadecimal).

Seven-segment decoders (7446, 7447, etc.) are useful for decimal displays. They convert a BCD nibble into an output that can drive a seven-segment display. Figure 4-9*a* illustrates the idea behind a seven-segment LED display. It has seven separate LEDs that allow you to display any digit between 0 and 9. To display a 7, the decoder will turn on LEDs *a*, *b*, and *c* (Fig. 4-9*b*).

Seven-segment displays are not limited to decimal numbers. For instance, in some microprocessor trainers, seven-segment displays are used to indicate hexadecimal digits. Digits A, C, E, and F are displayed in uppercase form; digit B is shown as a lowercase b (LEDs *c, d, e, f, g*); and digit D as a lowercase d (LEDs *b, c, d, e, g*).

Schmitt Triggers

When a computer is running, the outputs of gates are rapidly switching from one state to another. If you look at these signals with an oscilloscope, you see signals that ideally resemble rectangular waves like Fig. 4-10*a*.

When digital signals are transmitted and later received, they are often corrupted by noise, attenuation, or other factors and may wind up looking like the ragged waveform shown in Fig. 4-10*b*. If you try to use these nonrectangular signals to drive a gate or other digital device, you get unreliable operation.

This is where the *Schmitt trigger* comes in. It designed to clean up ragged looking pulses, producing almost vertical

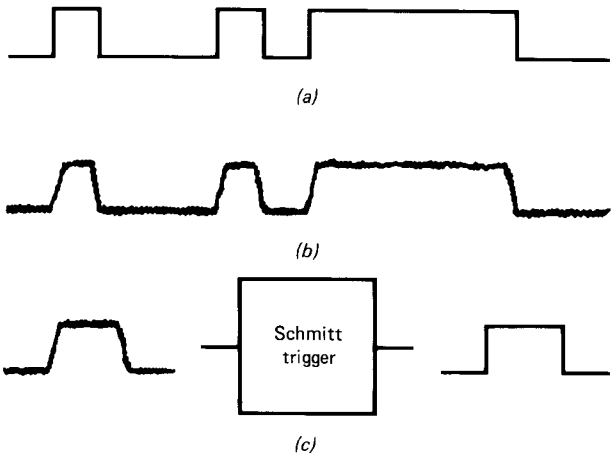


Fig. 4-10 Schmitt trigger produces rectangular output.

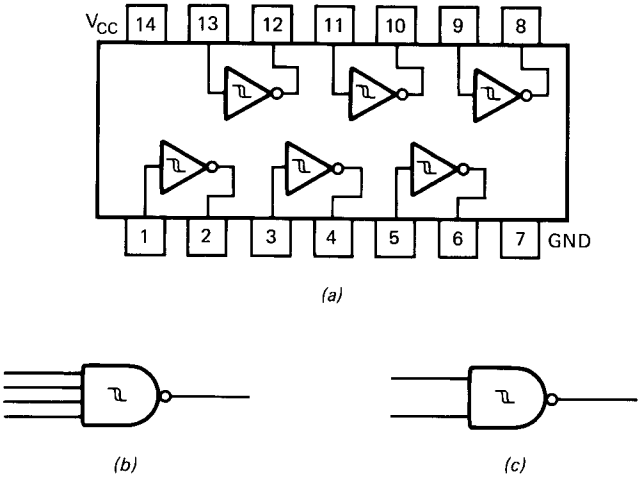


Fig. 4-11 (*a*) Hex Schmitt-trigger inverters; (*b*) 4-input NAND Schmitt trigger; (*c*) 2-input NAND Schmitt trigger.

transitions between the low and high state, and vice versa (Fig. 4-10*c*). In other words, the Schmitt trigger produces a rectangular output, regardless of the input waveform.

The 7414 is a hex Schmitt-trigger inverter, meaning six Schmitt-trigger inverters in one package like Fig. 4-11*a*. Notice the hysteresis symbol inside each inverter; it designates the Schmitt-trigger function.

Two other TTL Schmitt triggers are available. The 7413 is a dual 4-input NAND Schmitt trigger, two Schmitt-trigger gates like Fig. 4-11*b*. The 74132 is a quad 2-input NAND Schmitt trigger, four Schmitt-trigger gates like Fig. 4-11*c*.

Other Devices

The 7400 series also includes a number of other devices that you will find useful, such as AND-OR-INVERT gates (discussed in the next section), latches and flip-flops (Chap. 7), registers and counters (Chap. 8), and memories (Chap. 9).

4-5 AND-OR-INVERT GATES

Figure 4-12*a* shows an AND-OR circuit. Figure 4-12*b* shows the De Morgan equivalent circuit, a NAND-NAND network. In either case, the boolean equation is

$$Y = AB + CD \tag{4-1}$$

Since NAND gates are the preferred TTL gates, we would build the circuit of Fig. 4-12*b*. NAND-NAND circuits like this are important because with them you can build any desired logic circuit (discussed in Chap. 5).

TTL Devices

Is there any TTL device with the output given by Eq. 4-1? Yes, there are some AND-OR gates but they are not easily derived from the basic NAND-gate design. The gate that is easy to derive and comes close to having an expression like Eq. 4-1 is the AND-OR-INVERT gate shown in Fig. 4-12*c*. In other words, a variety of circuits like this are available on chips. Because of the inversion, the output has an equation of

$$Y = \overline{AB + CD} \tag{4-2}$$

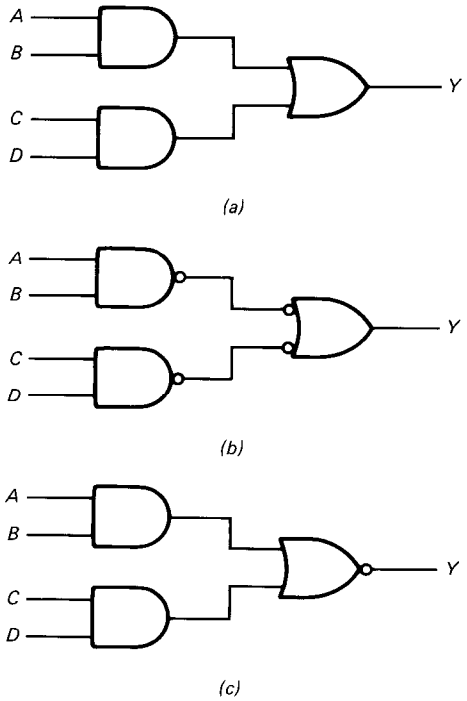


Fig. 4-12 (*a*) AND-OR circuit; (*b*) NAND-NAND circuit; (*c*) AND-OR-INVERT circuit.

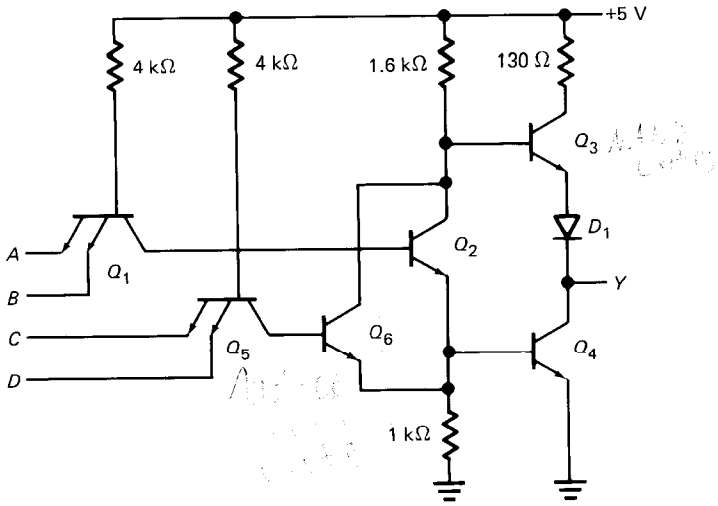


Fig. 4-13 AND-OR-INVERT schematic diagram.

Figure 4-13 shows the schematic diagram of a TTL AND-OR-INVERT gate. Q_1, Q_2, Q_3, and Q_4 form the basic 2-input NAND gate of the 7400 series. By adding Q_5 and Q_6 we convert the basic NAND gate to an AND-OR-INVERT gate.

Q_1 and Q_5 act like 2-input AND gates; Q_2 and Q_6 produce ORing and inversion. Because of this, the circuit is logically equivalent to Fig. 4-12*c*.

In Table 4-6, listing the AND-OR-INVERT gates available in the 7400 series, 2-wide means two AND gates across, 4-wide means four AND gates across, and so on. For instance, the 7454 is a 2-input 4-wide AND-OR-INVERT gate like Fig. 4-14*a*; each AND gate has two inputs (2-input) and there are four AND gates (4-wide). Figure 4-14*b* shows the 7464; it is a 2-2-3-4-input 4-wide AND-OR-INVERT gate.

When we want the output given by Eq. 4-1, we can connect the output of a 2-input 2-wide AND-OR-INVERT gate to another inverter. This cancels out the internal inversion, giving us the equivalent of an AND-OR circuit (Fig. 4-12*a*) or a NAND-NAND network (Fig. 4-12*b*).

Expandable AND-OR-INVERT Gates

The widest AND-OR-INVERT gate available in the 7400 series is 4-wide. What do we do when we need a 6- or 8-wide circuit? One solution is to use an *expandable* AND-OR-INVERT gate.

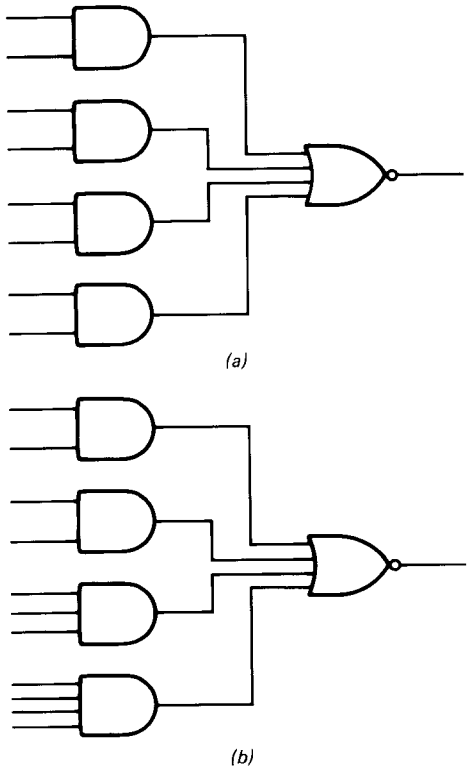


Fig. 4-14 Examples of AND-OR-INVERT circuits.

Figure 4-15*a* shows the schematic diagram of an expandable AND-OR-INVERT gate. The only difference between this and the preceding AND-OR-INVERT gate (Fig. 4-13) is collector and emitter tie points brought outside the package. Since Q_2 and Q_6 are the key to the ORing operation, we are being given access to the internal ORing function. By connecting other gates to these new inputs we can expand the width of the AND-OR-INVERT gate.

Figure 4-15*b* shows the logic symbol for an expandable AND-OR-INVERT gate. The arrow input represents the emitter, and the bubble stands for the collector. Table 4-7 lists the expandable AND-OR-INVERT gates in the 7400 series.

Expanders

What do we connect to the collector and emitter inputs of an expandable gate? The output of an *expander* like Fig. 4-16*a*. The input transistor acts like a 4-input AND gate. The output transistor is a phase splitter; it produces two

TABLE 4-6. AND-OR-INVERT GATES

Device	Description
7451	Dual 2-input 2-wide
7454	2-input 4-wide
7459	Dual 2-3 input 2-wide
7464	2-2-3-4 input 4-wide

TABLE 4-7. EXPANDABLE AND-OR-INVERT GATES

Device	Description
7450	Dual 2-input 2-wide
7453	2-input 4-wide
7455	4-input 2-wide

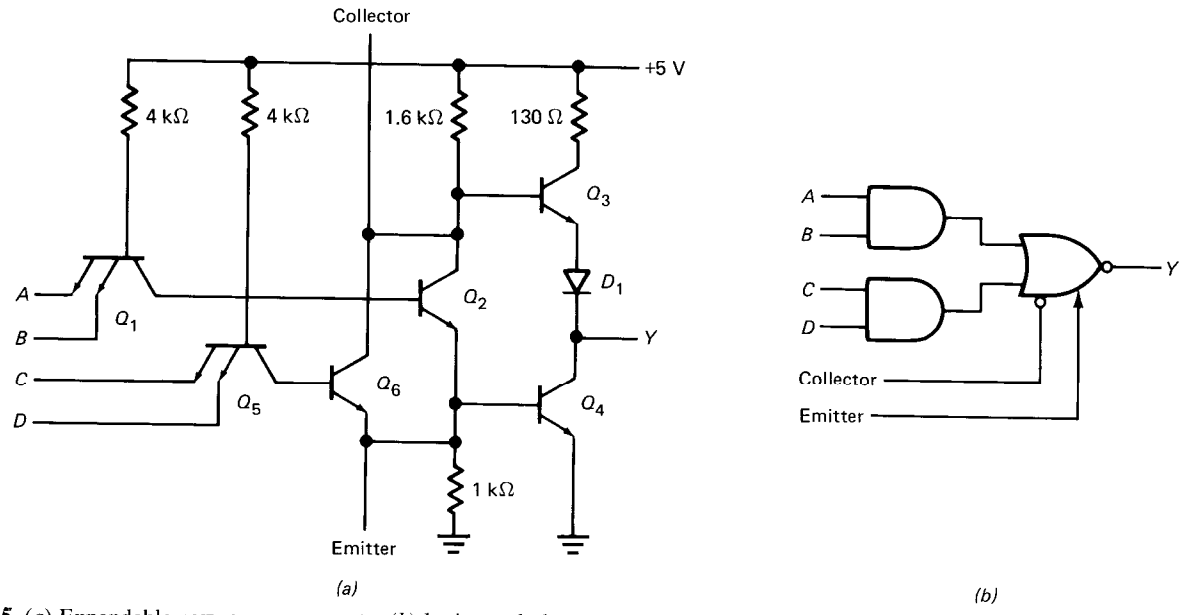


Fig. 4-15 (*a*) Expandable AND-OR-INVERT gate; (*b*) logic symbol.

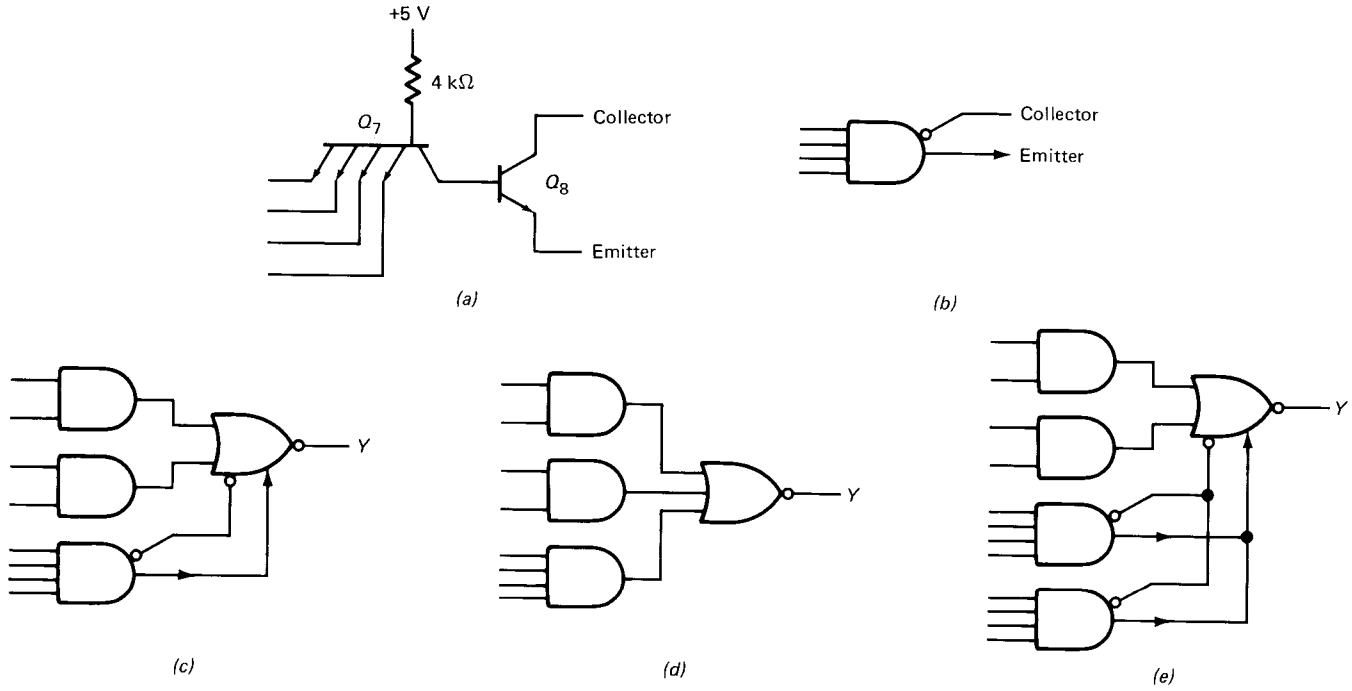


Fig. 4-16 (*a*) Expander; (*b*) symbol for expander; (*c*) expander driving expandable AND-OR-INVERT gate; (*d*) AND-OR-INVERT circuit; (*e*) expandable AND-OR-INVERT with two expanders.

output signals, one in phase (emitter) and the other inverted (collector). Figure 4-16*b* shows the symbol of a 4-input expander.

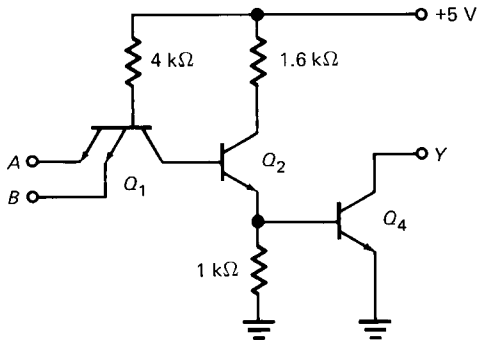
Visualize the outputs of Fig. 4-16*a* connected to the collector and emitter inputs of Fig. 4-15*a*. Then Q_8 is in parallel with Q_2 and Q_6. Figure 4-16*c* shows the logic circuit. This means that the expander outputs are being ORed with the signals of the AND-OR-INVERT gate. In other words, Fig. 4-16*c* is equivalent to the AND-OR-INVERT circuit of Fig. 4-16*d*.

We can connect more expanders. Figure 4-16*e* shows two expanders driving the expandable gate. Now we have a 2-2-4-4-input 4-wide AND-OR-INVERT circuit.

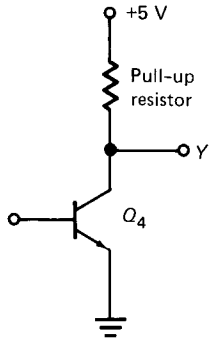
The 7460 is a dual 4-input expander. The 7450, a dual expandable AND-OR-INVERT gate, is designed for use with up to four 7460 expanders. This means that we can add two more expanders in Fig. 4-16*e* to get a 2-2-4-4-4-4-input 6-wide AND-OR-INVERT circuit.

4-6 OPEN-COLLECTOR GATES

Instead of a totem-pole output, some TTL devices have an *open-collector* output. This means they use only the lower transistor of a totem-pole pair. Figure 4-17*a* shows a 2-input NAND gate with an open-collector output. Because the collector of Q_4 is open, a gate like this won't work properly until you connect an external *pull-up* resistor, shown in Fig. 4-17*b*.



(a)



(b)

Fig. 4-17 Open-collector TTL: (*a*) circuit; (*b*) with pull-up resistor.

The outputs of open-collector gates can be wired together and connected to a common pull-up resistor. This is known as WIRE-OR. The big disadvantage of open-collector gates is their slow switching speed.

Open-collector gates are virtually obsolete because a new device called the *three-state switch* appeared in the early 1970s. Section 8-8 discusses three-state switches in detail.

4-7 MULTIPLEXERS

Multiplex means "many into one." A *multiplexer* is a circuit with many inputs but only one output. By applying control signals we can steer any input to the output.

Data Selection

Figure 4-18 shows a 16-to-1 multiplexer, also called a *data selector*. The input data bits are D_0 to D_{15}. Only one of these is transmitted to the output. Control word ABCD determines which data bit is passed to the output. For instance, when

$$\text{ABCD} = 0000$$

the upper AND gate is enabled but all other AND gates are disabled. Therefore, data bit D_0 is transmitted to the output, giving

$$Y = D_0$$

If the control word is changed to

$$\text{ABCD} = 1111$$

the bottom gate is enabled and all other gates are disabled. In this case,

$$Y = D_{15}$$

Boolean Function Generator

Digital design often starts with a truth table. The problem then is to come up with an equivalent logic circuit. Multiplexers give us a simple way to transform a truth table into an equivalent logic circuit. The idea is to use input data bits that are equal to the desired output bits of the truth table.

For example, look at the truth table of Table 4-8. When the input word ABCD is 0000, the output is 0; when ABCD

TABLE 4-8

A	*B*	*C*	*D*	*Y*
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

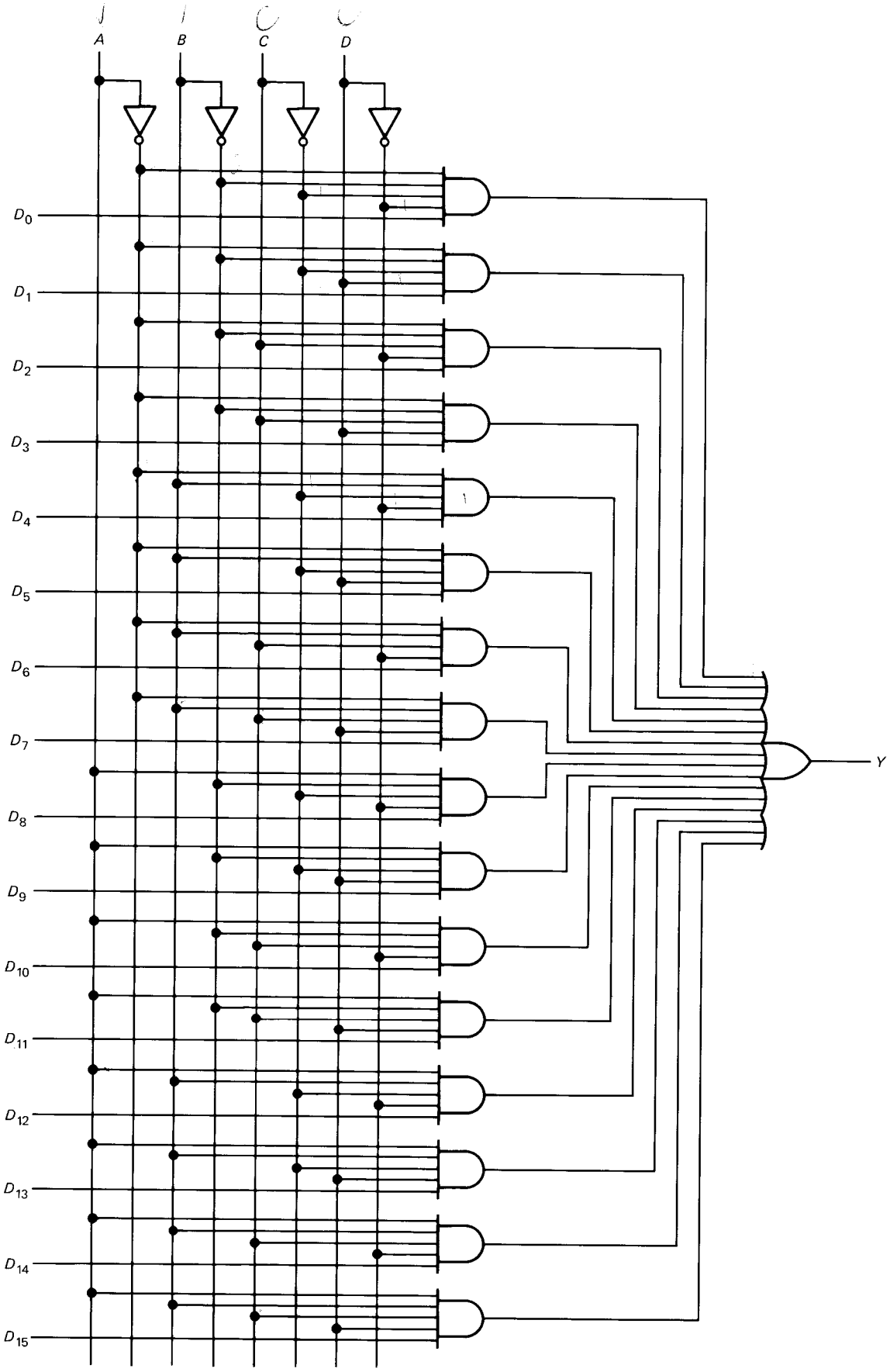


Fig. 4-18 A 16-to-1 multiplexer.

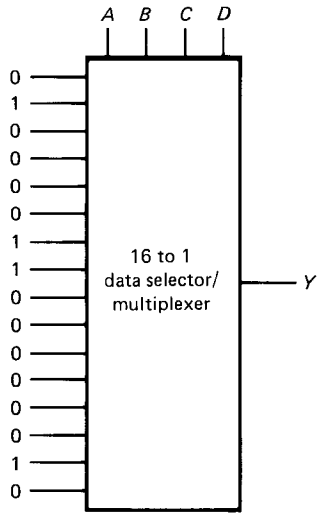


Fig. 4-19 Generating a boolean function.

= 0001, the output is 1; when ABCD = 0010, the output is 0; and so on. Figure 4-19 shows how to set up a multiplexer with the foregoing truth table. When ABCD = 0000, data bit 0 is steered to the output; when ABCD = 0001, data bit 1 is steered to the output; when ABCD = 0010, data bit 0 is steered to the output; and so forth. As a result, the truth table of this circuit is the same as Table 4-8.

Universal Logic Circuit

The 74150 is a 16-to-1 multiplexer. This TTL device is a universal logic circuit because you can use it to get the hardware equivalent of any four-variable truth table. In other words, by changing the input data bits the same IC can be made to generate thousands of different truth tables.

Multiplexing Words

Figure 4-20 illustrates a *word multiplexer* that has two input words and one output word. The input word on the left is $L_3L_2L_1L_0$ and the one on the right is $R_3R_2R_1R_0$. The control signal labeled *RIGHT* selects the input word that will be transmitted to the output. When *RIGHT* is low, the four NAND gates on the left are activated; therefore,

$$\mathbf{OUT} = L_3L_2L_1L_0$$

When *RIGHT* is high,

$$\mathbf{OUT} = R_3R_2R_1R_0$$

The 74157 is TTL multiplexer with an equivalent circuit like Fig. 4-20. Appendix 3 lists other multiplexers available in the 7400 series.

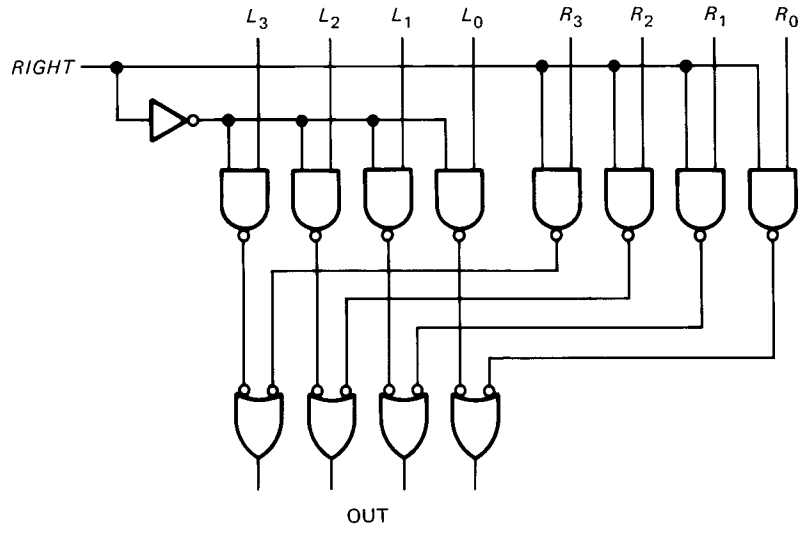


Fig. 4-20 Nibble multiplexer.

GLOSSARY

bipolar Having two types of charge carriers: free electrons and holes.

chip A small piece of semiconductor material. Sometimes, chip refers an IC device including its pins.

fanout The maximum number of TTL loads that a TTL device can drive reliably over the specified temperature range.

low-power Schottky TTL A modification of standard TTL

in which larger resistances and Schottky diodes are used. The increased resistances decrease the power dissipation, and the Schottky diodes increase the speed.

multiplexer A circuit with many inputs but only one output. Control signals select which input reaches the output.

noise margin The amount of noise voltage that causes unreliable operation. With TTL it is 0.4 V. As long as noise voltages induced on connecting lines are less than 0.4 V, the TTL devices will work reliably.

saturation delay time The time delay encountered when a transistor tries to come out of the saturation region. When the base drive switches from high to low, a transistor cannot instantaneously come out of saturation; extra carriers that flooded the base region must first flow out of the base.

Schmitt trigger A digital circuit that produces a rectangular output from any input large enough to drive the Schmitt trigger. The input waveform may be sinusoidal, triangular, distorted, and so on. The output is always rectangular.

sink A place where something is absorbed. When saturated, the lower transistor in a totem-pole output acts like a current sink because conventional charges flow through the transistor to ground.

source A place where something originates. The upper transistor of a totem-pole output acts like a source because charges flow out of its emitter into the load.

standard TTL The initial TTL design with resistance values that produce a power dissipation of 10 mW per gate and a propagation delay time of 10 ns.

SELF-TESTING REVIEW

Read each of the following and provide the missing words. Answers appear at the beginning of the next question.

1. Small-scale integration, abbreviated _____, refers to fewer than 12 gates on the same chip. Medium-scale integration (MSI) means 12 to 100 gates per chip. And large-scale integration (LSI) refers to more than _____ gates per chip.
2. (*SSI, 100*) The two basic technologies for digital ICs are bipolar and MOS. Bipolar technology is preferred for _____ and _____, whereas MOS technology is better suited to LSI. The reason MOS dominates the LSI field is that more _____ can be fabricated on the same chip area.
3. (*SSI, MSI, MOSFETs*) Some of the bipolar families include DTL, TTL, and ECL. _____ has become the most widely used bipolar family. _____ is the fastest logic family; it's used in high-speed applications.
4. (*TTL, ECL*) Some of the MOS families are PMOS, NMOS, and CMOS. _____ dominates the LSI field, and _____ is used extensively where lowest power consumption is necessary.
5. (*NMOS, CMOS*) The 7400 series, also called standard TTL, contains a variety of SSI and _____ chips that allow us to build all kinds of digital circuits and systems. Standard TTL has a multiple-emitter input transistor and a _____ output. The totem-pole output produces a low output impedance in either state.
6. (*MSI, totem-pole*) Besides standard TTL, there is high-speed TTL, low-power TTL, Schottky TTL, and low-power _____ TTL. Standard TTL and low-power _____ TTL have become the favorites of digital designers, used more than any other bipolar families.
7. (*Schottky, Schottky*) 7400-series devices are guaranteed to work reliably over a _____ range of 0 to 70°C and over a voltage range of 4.75 to 5.25 V. A floating TTL input has the same effect as a _____ input.
8. (*temperature, high*) A _____ TTL device can sink up to 16 mA and can source up to 400 μA. The maximum number of TTL loads a TTL device can drive is called the _____. With standard TTL, the fanout equals _____.
9. (*standard, fanout, 10*) A buffer is a device that isolates other devices. Typically, a buffer has a high input impedance and a _____ output impedance. In terms of digital ICs, this means a _____ input current and a high output current capability.
10. (*low, low*) A Schmitt trigger is a digital circuit that produces a _____ output regardless of the input waveform. It is used to clean up ragged looking pulses that have been distorted during transmission from one place to another.
11. (*rectangular*) A multiplexer is a circuit with many inputs but only one output. It is also called a data selector because data can be steered from one of the inputs to the output. A 74150 is a 16-to-1 multiplexer. With this TTL device you can implement the logic circuit for any four-variable truth table.

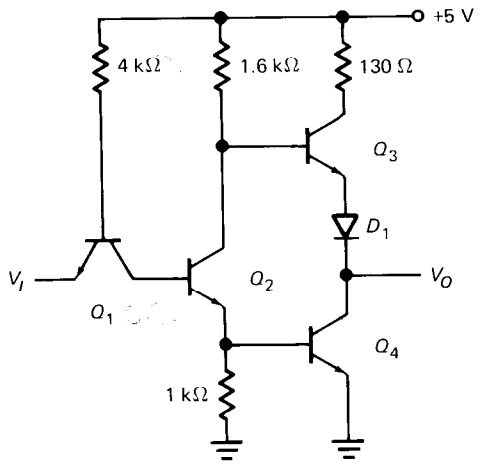


Fig. 4-21

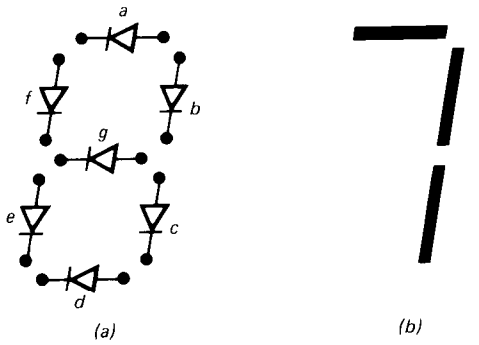


Fig. 4-22

4-1. In Fig. 4-21 a grounded input means that almost the entire supply voltage appears across the 4-kΩ resistor. Allowing 0.7 V for the emitter-base voltage of Q_1, how much input emitter current is there with a grounded input? The supply voltage can be as high as 5.25 V and the 4-kΩ resistance can be a low as 3.28 kΩ. What is the input emitter current in this case?

4-2. What is the fanout of a 74S00 device when it drives low-power TTL loads?

4-3. What is the fanout of a low-power Schottky device driving standard TTL devices?

4-4. Section 4-4 gave the input and output currents for a 7437 buffer. What is the fanout of a 7437 when it drives standard TTL loads?

4-5. A seven-segment decoder is driving a LED display like Fig. 4-22*a*. Which LEDs are on when digit 8 appears? Which LEDs are on when digit 4 appears?

4-6. Section 4-7 described the 74150, a 16-to-1 multiplexer. Refer to Fig. 4-23 and indicate the values the D_0 to D_{15} inputs of a 74150 should have to reproduce the following truth table: The output is high when ABCD = 0000, 0100, 0111, 1100, and 1111; the output is low for all other inputs.

4-7. What is propagation delay?

4-8. Why are 5400 series devices not normally used in commercial applications?

4-9. What do Schottky devices virtually eliminate which makes their high switching speeds possible?

4-10. What is the noise margin of TTL devices?

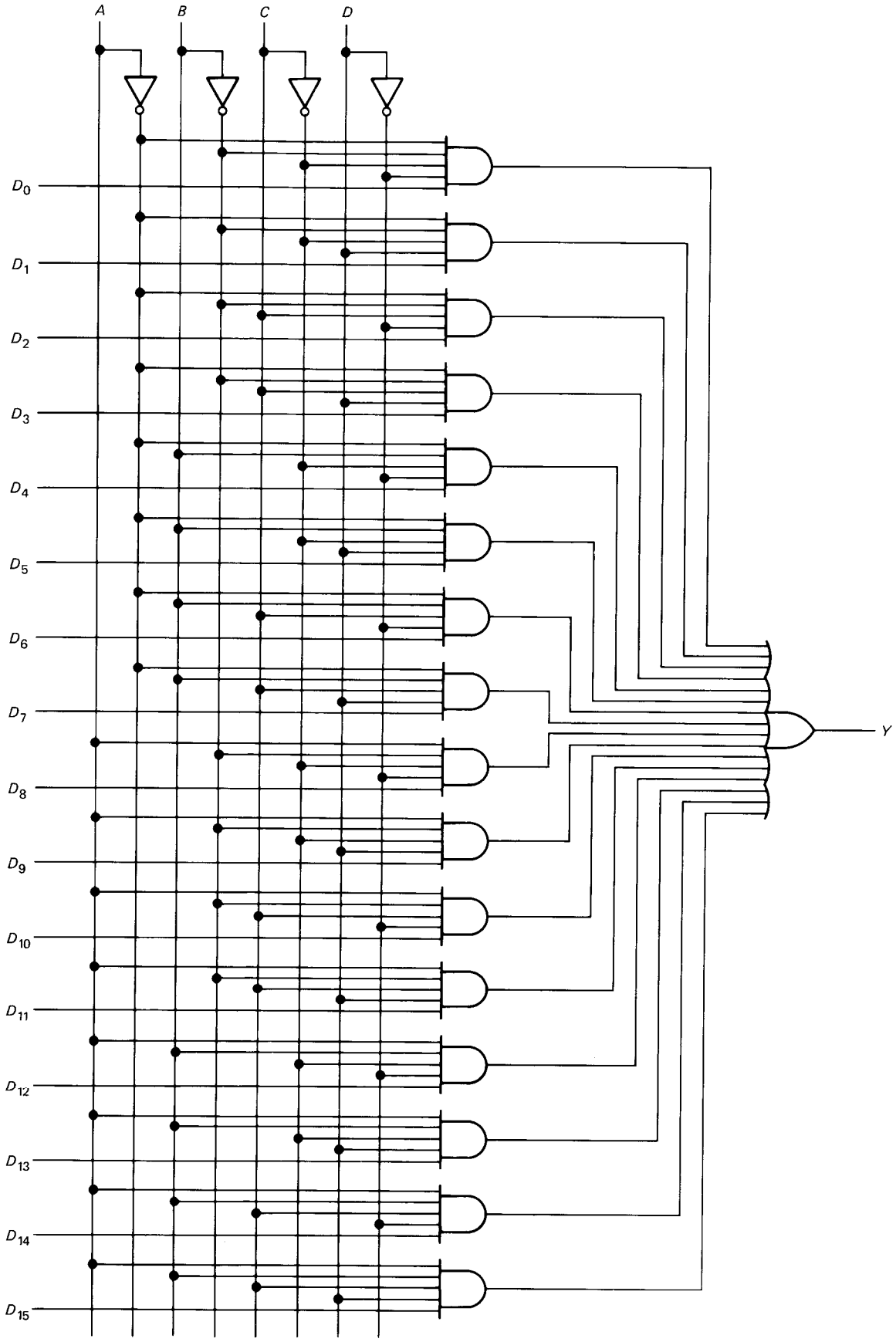


Fig. 4-23