6

ARITHMETIC-LOGIC UNITS

The arithmetic-logic unit (ALU) is the number-crunching part of a computer. This means not only arithmetic operations but logic as well (OR, AND, NOT, and so forth). In this chapter you will learn how the ALU adds and subtracts binary numbers. Later chapters will discuss the logic operations.

6-1 BINARY ADDITION

ALUs don't process decimal numbers; they process binary numbers. Before you can understand the circuits inside an ALU, you must learn how to add binary numbers. There are five basic cases that must be understood before going on.

Case 1

When no pebbles are added to no pebbles, the total is no pebbles. As a word equation,

None + none = none

With binary numbers, this equation is written as

$$0 + 0 = 0$$

Case 2

If no pebbles are added to one pebble, the total is one pebble:

None + ● = ●

In terms of binary numbers,

$$0 + 1 = 1$$

Case 3

Addition is commutative. This means you can transpose the numbers of the preceding case to get

● + none = ●

or

$$1 + 0 = 1$$

Case 4

Next, one pebble added to one pebble gives two pebbles:

● + ● = ●●

As a binary equation,

$$1 + 1 = 10$$

To avoid confusion with decimal numbers, read this as "one plus one equals one-zero." An alternative way of reading the equation is "one plus one equals zero, carry one."

Case 5

One pebble plus one pebble plus one pebble gives a total of three pebbles:

● + ● + ● = ●●●

The binary equation is

$$1 + 1 + 1 = 11$$

Read this as "one plus one plus one equals one-one." Alternatively, "one plus one plus one equals one, carry one."

Rules to Remember

The foregoing cases are all you need for more complicated binary addition. Therefore, memorize these five rules:

$$0 + 0 = 0 \tag{6-1}$$
$$0 + 1 = 1 \tag{6-2}$$
$$1 + 0 = 1 \tag{6-3}$$
$$1 + 1 = 10 \tag{6-4}$$
$$1 + 1 + 1 = 11 \tag{6-5}$$

Larger Binary Numbers

Column-by-column addition applies to binary numbers as well as decimal. For example, suppose you have this problem in binary addition:

$$\begin{array}{r} 11100 \\ +\ 11010 \\ \hline ? \end{array}$$

Start with the least significant column to get

$$\begin{array}{r} 11100 \\ +\ 11010 \\ \hline 0 \end{array}$$

Here, 0 + 0 gives 0.

Next, add the bits of the second column as follows:

$$\begin{array}{r} 11100 \\ +\ 11010 \\ \hline 10 \end{array}$$

This time, 0 + 1 results in 1.

The third column gives

$$\begin{array}{r} 11100 \\ +\ 11010 \\ \hline 110 \end{array}$$

In this case, 1 + 0 produces 1.

The fourth column results in

$$\begin{array}{rl} 11100 & \\ +\ 11010 & \\ \cline{1-1} 0110 & \text{(carry 1)} \end{array}$$

As you see, 1 + 1 equals 0 with a carry of 1.

Finally, the last column gives

$$\begin{array}{r} 11100 \\ +\ 11010 \\ \hline 110110 \end{array}$$

Here, 1 + 1 + 1 (carry) produces 11, recorded as 1 with a carry to the next higher column.

EXAMPLE 6-1

Add the binary numbers 01010111 and 00110101.

SOLUTION

This is the problem:

$$\begin{array}{r} 01010111 \\ +\ 00110101 \\ \hline ? \end{array}$$

If you add the bits column by column as previously demonstrated, you will get

$$\begin{array}{r} 01010111 \\ +\ 00110101 \\ \hline 10001100 \end{array}$$

Expressed in hexadecimal numbers, the foregoing addition is

$$\begin{array}{r} 57 \\ +\ 35 \\ \hline 8C \end{array}$$

For clarity, we can use subscripts:

$$\begin{array}{r} 57_{16} \\ +\ 35_{16} \\ \hline 8C_{16} \end{array}$$

In microprocessor work, it is more convenient to use the letter H to signify hexadecimal numbers. In other words, the usual way to express the foregoing addition is

$$\begin{array}{r} 57H \\ +\ 35H \\ \hline 8CH \end{array}$$

6-2 BINARY SUBTRACTION

To subtract binary numbers, we need to discuss four cases.

Case 1:	$0 - 0 = 0$
Case 2:	$1 - 0 = 1$
Case 3:	$1 - 1 = 0$
Case 4:	$10 - 1 = 1$

The last result represents

●● − ● = ●

which makes sense.

To subtract larger binary numbers, subtract column by column, borrowing from the next higher column when necessary. For instance, in subtracting 101 from 111, proceed like this:

$$\begin{array}{r} 7 \\ -\ 5 \\ \hline 2 \end{array} \qquad \begin{array}{r} 111 \\ -\ 101 \\ \hline 010 \end{array}$$

Starting on the right, 1 − 1 gives 0; then, 1 − 0 is 1; finally, 1 − 1 is 0.

Here is another example: subtract 1010 from 1101.

$$\begin{array}{r} 13 \\ -\ 10 \\ \hline 3 \end{array} \qquad \begin{array}{r} 1101 \\ -\ 1010 \\ \hline 0011 \end{array}$$

In the least significant column, 1 − 0 is 1. In the second column, we have to borrow from the next higher column; then, 10 − 1 is 1. In the third column, 0 (after borrow) − 0 is 0. In the fourth column, 1 − 1 = 0.

Direct subtraction like the foregoing has been used in computers; however, it is possible to subtract in a different way. Later sections of this chapter will show you how.

6-3 HALF-ADDERS

Figure 6-1 is a *half-adder,* a logic circuit that adds 2 bits. Notice the outputs: *SUM* and *CARRY*. The boolean equations for these outputs are

$$SUM = A \oplus B \qquad (6\text{-}6)$$
$$CARRY = AB \qquad (6\text{-}7)$$

The *SUM* output is *A* XOR *B*; the *CARRY* output is *A* AND *B*. Therefore, *SUM* is a 1 when *A* and *B* are different; *CARRY* is a 1 when *A* and *B* are 1s.

Table 6-1 summarizes the operation. When *A* and *B* are 0s, the *SUM* is 0 with a *CARRY* of 0. When *A* is 0 and *B* is 1, the *SUM* is 1 with a *CARRY* of 0. When *A* is 1 and *B* is 0, the *SUM* equals 1 with a *CARRY* of 0. Finally, when *A* is 1 and *B* is 1, the *SUM* is 0 with a *CARRY* of 1.

The logic circuit of Fig. 6-1 does electronically what we do mentally when we add 2 bits. Applications for the half-adder are limited. What we need is a circuit that can add 3 bits at a time.

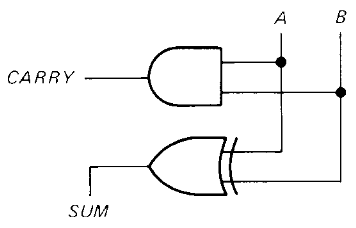


Fig. 6-1 Half-adder.

TABLE 6-1. HALF-ADDER

A	*B*	*CARRY*	*SUM*
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

6-4 FULL ADDERS

Figure 6-2 shows a *full adder,* a logic circuit that can add 3 bits. Again there are two outputs, *SUM* and *CARRY*. The boolean equations are

$$SUM = A \oplus B \oplus C \qquad (6\text{-}8)$$
$$CARRY = AB + AC + BC \qquad (6\text{-}9)$$

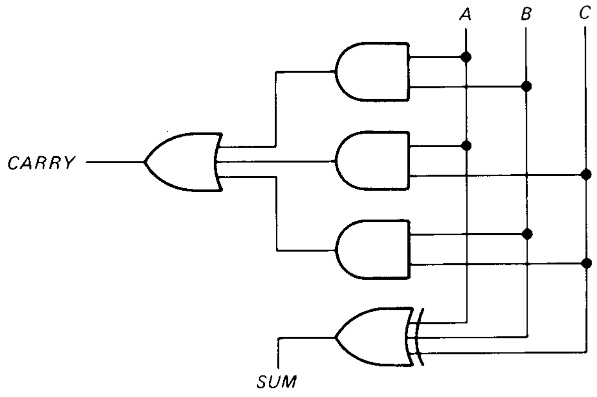


Fig. 6-2 Full adder.

In this case, *SUM* equals *A* XOR *B* XOR *C; CARRY* equals *AB* OR *AC* OR *BC*. Therefore, *SUM* is 1 when the number of input 1s is odd; *CARRY* is a 1 when two or more inputs are 1s.

Table 6-2 summarizes the circuit action. *A, B,* and *C* are the bits being added. If you check each entry, you will see that the circuit adds 3 bits at a time and comes up with the correct answer.

TABLE 6-2. FULL ADDER

A	*B*	*C*	*CARRY*	*SUM*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Here's the point. The circuit of Fig. 6-2 does electronically what we do mentally when we add 3 bits. The full adder can be cascaded to add large binary numbers. The next section tells you how.

6-5 BINARY ADDERS

Figure 6-3 shows a *binary adder,* a logic circuit that can add two binary numbers. The block on the right (labeled HA) represents a half-adder. The inputs are A_0 and B_0; the outputs are S_0 (*SUM*) and C_1 (*CARRY*). All other blocks are full adders (abbreviated FA). Each of these full adders has three inputs (A_n, B_n, and C_n) and two outputs.

The circuit adds two binary numbers. In other words, it carries out the following addition:

$$\begin{array}{r} A_3A_2A_1A_0 \\ +\ B_3B_2B_1B_0 \\ \hline C_4S_3S_2S_1S_0 \end{array}$$

Here's an example. Suppose **A** = 1100 and **B** = 1001. Then the problem is

$$\begin{array}{r} 1100 \\ +\ 1001 \\ \hline ? \end{array}$$

Figure 6-4 shows the binary adder with the same inputs, 1100 and 1001. The half-adder produces a sum of 1 and carry of 0, the first full adder produces a sum of 0 and a carry of 0, the second full adder produces a sum of 1 and a carry of 0, and the third full adder produces a sum of 0 and a carry of 1. The overall output is 10101, the same answer we would get with pencil and paper.

By using more full adders, we can build binary adders of any length. For example, to add 16-bit numbers, we need 1 half-adder and 15 full adders. From now on, we will use the abbreviated symbol of Fig. 6-5 to represent a binary adder of any length. Notice the solid arrows, the standard way to indicate words in motion. In Fig. 6-5, words **A** and **B** are added to get a sum of **S** plus a final *CARRY.*

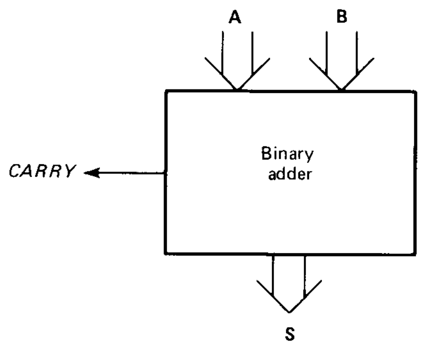


Fig. 6-5 Symbol for binary adder.

EXAMPLE 6-2

Find the output in Fig. 6-5 if the two input words are

A = 0000 0001 0000 1100
B = 0000 0000 0100 1001

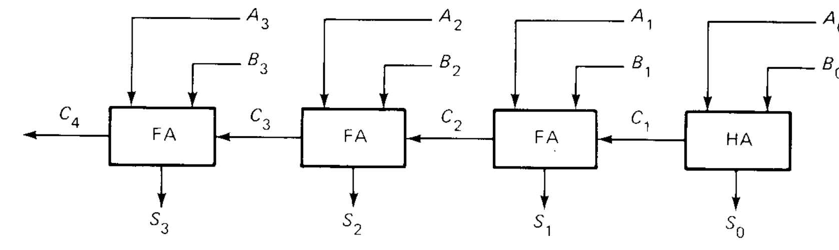


Fig. 6-3 Binary adder.

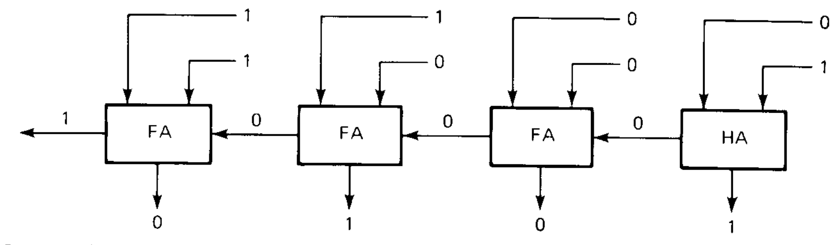


Fig. 6-4 Adding 12 and 9 to get 21.

SOLUTION

The binary adder adds the two inputs to get

$$\begin{array}{r} 0000\ 0001\ 0000\ 1100 \\ +\ \underline{0000\ 0000\ 0100\ 1001} \\ 0000\ 0001\ 0101\ 0101 \end{array}$$

In hexadecimal form, the foregoing addition is

$$\begin{array}{r} 010CH \\ +\ \underline{0049H} \\ 0155H \end{array}$$

6-6 SIGNED BINARY NUMBERS

The negative decimal numbers are −1, −2, −3, and so on. One way to code these as binary numbers is to convert the *magnitude* (1, 2, 3, . . .) to its binary equivalent and prefix the sign. With this approach, −1, −2, and −3 becomes −001, −010, and −011. It's customary to use 0 for the + sign and 1 for the − sign. Therefore, −001, −010, and −011 are coded as 1001, 1010, and 1011.

The foregoing numbers have the *sign bit* followed by the magnitude bits. Numbers in this form are called *signed binary numbers* or *sign-magnitude numbers*. For larger decimal numbers you need more than 4 bits. But the idea is still the same: the leading bit represents the sign and the remaining bits stand for the magnitude.

EXAMPLE 6-3

Express each of the following as 16-bit signed binary numbers.

a. +7
b. −7
c. +25
d. −25

SOLUTION

a. +7 = 0000 0000 0000 0111
b. −7 = 1000 0000 0000 0111
c. +25 = 0000 0000 0001 1001
d. −25 = 1000 0000 0001 1001

No subscripts are used in these equations because it's clear from the context that decimal numbers are being expressed in binary form. Nevertheless, you can use subscripts if you prefer. The first equation can be written as

$$+7_{10} = 0000\ 0000\ 0000\ 0111_2$$

the next equation as

$$-7_{10} = 1000\ 0000\ 0000\ 0111_2$$

and so forth.

EXAMPLE 6-4

Convert the following signed binary numbers to decimal numbers:

a. 0000 0000 0000 1001
b. 1000 0000 0000 1111
c. 1000 0000 0011 0000
d. 0000 0000 1010 0101

SOLUTION

As usual, the leading bit gives the sign and the remaining bits give the magnitude.

a. 0000 0000 0000 1001 = +9
b. 1000 0000 0000 1111 = −15
c. 1000 0000 0011 0000 = −48
d. 0000 0000 1010 0101 = +165

6-7 2's COMPLEMENT

Sign-magnitude numbers are easy to understand, but they require too much hardware for addition and subtraction. This has led to the widespread use of complements for binary arithmetic.

Definition

Recall that a high invert signal to a controlled inverter produces the 1's complement. For instance, if

$$\mathbf{A} = 0111 \tag{6-10a}$$

the 1's complement is

$$\overline{\mathbf{A}} = 1000 \tag{6-10b}$$

The 2's complement is defined as the new word obtained by adding 1 to 1's complement. As an equation,

$$\mathbf{A}' = \overline{\mathbf{A}} + 1 \tag{6-11}$$

where $\mathbf{A}'$ = 2's complement
$\overline{\mathbf{A}}$ = 1's complement

Here are some examples. If

$$\mathbf{A} = 0111$$

the 1's complement is

$$\overline{\mathbf{A}} = 1000$$

and the 2's complement is

$$\mathbf{A}' = 1001$$

In terms of a binary odometer, the 2's complement is the next reading after the 1's complement.

Another example. If

$$\mathbf{A} = 0000\ 1000$$

then

$$\overline{\mathbf{A}} = 1111\ 0111$$

and

$$\mathbf{A}' = 1111\ 1000$$

Double Complement

If you take the 2's complement twice, you get the original word back. For instance, if

$$\mathbf{A} = 0111$$

the 2's complement is

$$\mathbf{A}' = 1001$$

If you take the 2's complement of this, you get

$$\mathbf{A}'' = 0111$$

which is the original word.

In general, this means that

$$\mathbf{A}'' = \mathbf{A} \qquad (6\text{-}12)$$

Read this as "the double complement of **A** equals **A**." Because of this property, the 2's complement of a binary number is equivalent to the negative of a decimal number. This idea is explained in the following discussion.

Back to the Odometer

Chapter 1 used an odometer to introduce binary numbers. The discussion was about positive numbers only. But odometer readings can also indicate negative numbers. Here's how.

If a car has a binary odometer, all bits eventually reset to 0s. A few readings before and after a complete reset look like this:

```
1101
1110
1111
0000    (RESET)
0001
0010
0011
```

1101 is the reading 3 miles before reset, 1110 occurs 2 miles before reset, and 1111 indicates 1 mile before reset. Then, 0001 is the reading 1 mile after reset, 0010 occurs 2 miles after reset, and 0011 indicates 3 miles after reset.

"Before" and "after" are synonymous with "negative" and "positive." Figure 6-6 illustrates this idea with the number line learned in basic algebra: 0 marks the origin, positive decimal numbers are on the right, and negative decimal numbers are on the left. The odometer readings are the binary equivalent of positive and negative decimal numbers: 1101 is the binary equivalent of -3, 1110 stands for -2, 1111 for -1; 0000 for 0; 0001 for $+1$; 0010 for $+2$, and 0011 for $+3$.

The odometer readings of Fig. 6-6 demonstrate how positive and negative numbers are stored in a typical microcomputer. Positive decimal numbers are expressed in sign-magnitude form, but negative decimal numbers are represented as 2's complements. As before, positive numbers have a leading sign bit of 0, and negative numbers have a leading sign bit of 1.

2's Complement Same as Decimal Sign Change

Taking the 2's complement of a binary number is the same as changing the sign of the equivalent decimal number. For example, if

$$\mathbf{A} = 0001 \qquad (+1 \text{ in Fig. 6-6})$$

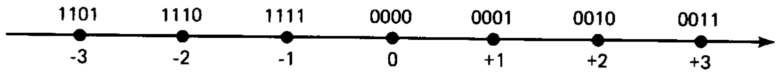


Fig. 6-6 Decimal numbers and odometer readings.

taking the 2's complement gives

A′ = 1111 (−1 in Fig. 6-6)

Similarly, if

A = 0010 (+2 in Fig. 6-6)

then the 2's complement is

A′ = 1110 (−2 in Fig. 6-6)

Again, if

A = 0011 (+3 in Fig. 6-6)

the 2's complement is

A′ = 1101 (−3 in Fig. 6-6)

The same principle applies to binary numbers of any length: taking the 2's complement of any binary number is the same as changing the sign of the equivalent decimal number. As will be shown later, this property allows us to use a binary adder for both addition and subtraction.

Summary

Here are the main things to remember about 2's complement representation:

1. The leading bit is the sign bit; 0 for plus, 1 for minus.
2. Positive decimal numbers are in sign-magnitude form.
3. Negative decimal numbers are in 2's-complement form.

EXAMPLE 6-5

What is the 2's complement of this word?

A = 0011 0101 1001 1100

SOLUTION

The 2's complement is

A′ = 1100 1010 0110 0100

EXAMPLE 6-6

What is the binary form of +5 and −5 in 2's-complement representation? Express the answers as 8-bit numbers.

SOLUTION

Decimal +5 is expressed in sign-magnitude form:

+5 = 0000 0101

On the other hand, −5 appears as the 2's complement:

−5 = 1111 1011

EXAMPLE 6-7

What is the 2's-complement representation of −24 in a 16-bit microcomputer?

SOLUTION

Start with the positive form:

+24 = 0000 0000 0001 1000

Then take the 2's complement to get the negative form:

−24 = 1111 1111 1110 1000

EXAMPLE 6-8

What decimal number does this represent in 2's-complement representation?

1111 0001

SOLUTION

Start by taking the 2's complement to get

0000 1111

This represents +15. Therefore, the original number is

1111 0001 = −15

6-8 2's-COMPLEMENT ADDER-SUBTRACTER

Early computers used signed binary for both positive and negative numbers. This led to complicated arithmetic circuits. Then, engineers discovered that the 2's-complement representation could greatly simplify arithmetic hardware.

This is why 2's-complement adder-subtracters are now the most widely used arithmetic circuits.

Addition

Figure 6-7 shows a 2's-complement adder-subtracter, a logic circuit that can add or subtract binary numbers. Here's how it works. When *SUB* is low, the *B* bits pass through the controlled inverter without inversion. Therefore, the full adders produce the sum

$$\mathbf{S} = \mathbf{A} + \mathbf{B} \qquad (6\text{-}13)$$

Incidentally, as indicated in Fig. 6-7, the final *CARRY* is not used. This is because S_3 is the sign bit and S_2 to S_0 are the numerical bits. The final *CARRY* therefore has no significance at this time.

Subtraction

When *SUB* is high, the controlled inverter produces the 1's complement. Furthermore, the high *SUB* adds a 1 to the first full adder. This addition of 1 to the 1's complement forms the 2's complement of **B**. In other words, the controlled inverter produces $\overline{\mathbf{B}}$, and adding 1 results in **B**′.

The output of the full adders is

$$\mathbf{S} = \mathbf{A} + \mathbf{B}' \qquad (6\text{-}14)$$

which is equivalent to

$$\mathbf{S} = \mathbf{A} - \mathbf{B} \qquad (6\text{-}15)$$

because the 2's complement is equivalent to a sign change.

EXAMPLE 6-9

A 7483 is a TTL circuit with four full adders. This means that it can add nibbles (4-bit numbers).

Figure 6-8 shows a TTL adder-subtracter. The *CARRY* out (pin 14) of the least significant nibble is used as the *CARRY* in (pin 13) for the most significant nibble. This allows the two 7483s to add 8-bit numbers. Two 7486s form the controlled inverter needed for subtraction.

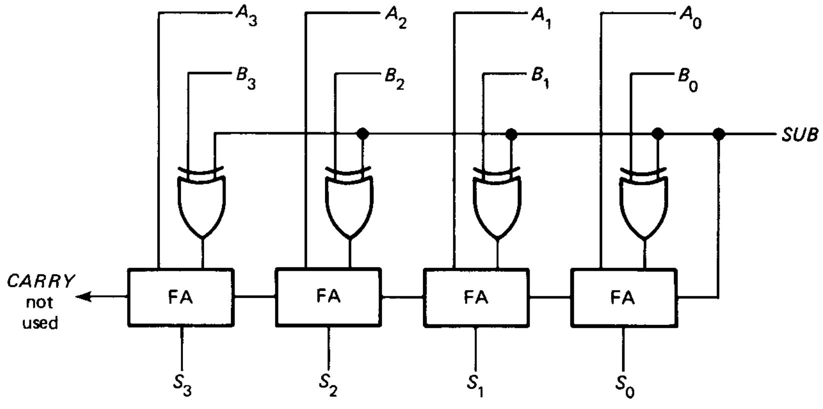


Fig. 6-7 A 2's-complement adder-subtracter.

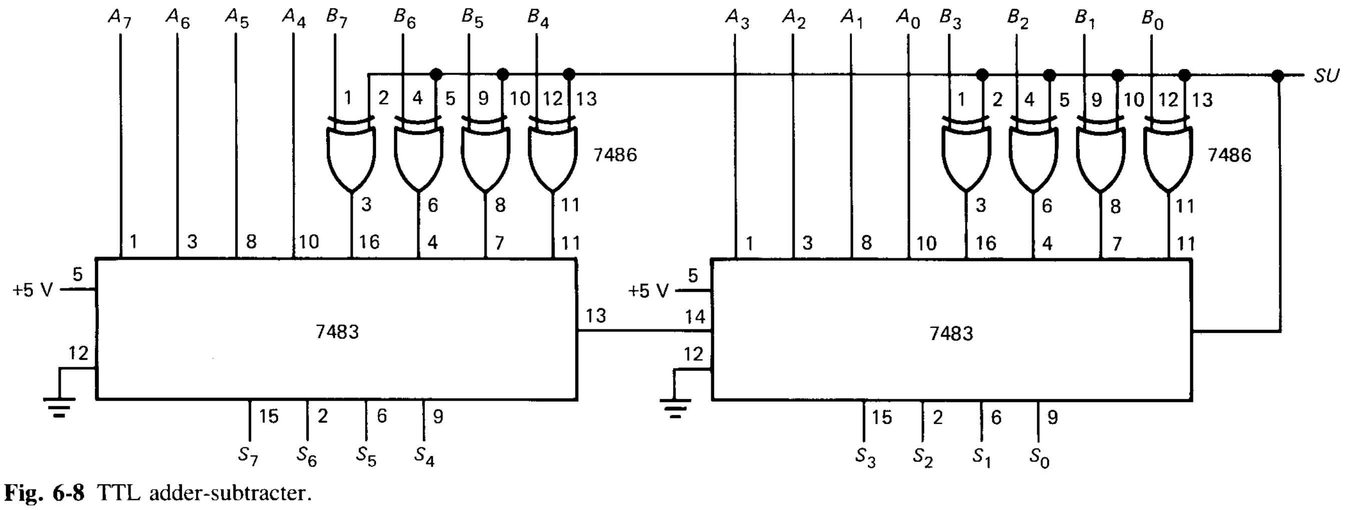


Fig. 6-8 TTL adder-subtracter.

Suppose the circuit has these inputs:

$$\mathbf{A} = 0001\ 1000$$
$$\mathbf{B} = 0001\ 0000$$

If $SUB = 0$, what is the output of the adder-subtracter?

SOLUTION

When SUB is 0, the adder-subtracter adds the two inputs as follows:

$$\begin{array}{r} 0001\ 1000 \\ +\ 0001\ 0000 \\ \hline 0010\ 1000 \end{array}$$

Therefore, the output is 0010 1000. Notice that the decimal equivalent of the foregoing addition is

$$\begin{array}{r} 24 \\ +\ 16 \\ \hline 40 \end{array}$$

EXAMPLE 6-10

Repeat the preceding example for $SUB = 1$.

SOLUTION

When SUB is 1, the adder-subtracter subtracts the inputs by adding the 2's complement as follows:

$$\begin{array}{r} 0001\ 1000 \\ +\ 1111\ 0000 \\ \hline 0000\ 1000 \end{array}$$

The decimal equivalent is

$$\begin{array}{r} 24 \\ +\ -16 \\ \hline 8 \end{array}$$

EXAMPLE 6-11

In Fig. 6-8, what are the largest positive and negative sums we can get?

SOLUTION

The largest positive output is

0111 1111

which represents decimal +127. The largest negative output is

1000 0000

which represents −128. With 8 bits, therefore, all answers must lie between −128 and +127. If you try to add numbers with a sum outside this range, you get an *overflow* into the sign-bit position, causing an error.

Chapter 12 discusses the overflow problem in more detail. All you have to remember for now is that an overflow or error will occur if the true sum lies outside the range of −128 to +127.

GLOSSARY

ALU Arithmetic-logic unit. The ALU carries out arithmetic and logic operations.
binary adder A logic circuit that can add two binary numbers.
full adder A logic circiut that can add 3 bits.
half-adder A logic circuit that adds 2 bits.
overflow In 2's-complement representation, a carry into the sign-bit position, which results in an error. For an 8-bit adder-substracter, the true sum must lie between −128 and +127 to avoid overflow.
signed binary A system in which the leading bit represents the sign and the remaining bits the magnitude of the number. Also called sign magnitude.
2's complement The new number you get when you take the 1's complement and then add 1.

SELF-TESTING REVIEW

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

1. The ALU carries out arithmetic and _________ operations (OR, AND, NOT, etc.). It processes _________ numbers rather than decimal numbers.
2. (*logic, binary*) A half-adder adds _________ bits. A full adder adds _________ bits, producing a SUM and a _________.
3. (*two, three,* CARRY) A binary adder is a logic cicuit that can add _________ binary numbers at a time. The 7483 is a TTL binary adder. It can add two 4-bit binary numbers.
4. (*two*) With signed binary numbers, also known as sign-magnitude numbers, the leading bit stands for the _________ and the remaining bits for the _________.
5. (*sign, magnitude*) Signed binary numbers require too much hardware. This has led to the use of _________ complements to represent negative numbers. To get the 2's complement of a binary number, you first take the _________ complement, then add _________.
6. (*2's, 1's, 1*) If you take the 2's complement twice, you get the original binary number back. Because of this property, taking the _________ complement of a binary number is equivalent to changing the sign of a decimal number.
7. (*2's*) In a microcomputer positive numbers are represented in _________ form and negative numbers in 2's-complement form. The leading bit still represents the _________.
8. (*sign-magnitude, sign*) A 2's-complement adder-subtracter can add or subtract binary numbers. Sign-magnitude numbers represent _________ decimal numbers, and 2's complements stand for _________ decimal numbers. You can tell one from the other by the leading bit, which represents the _________.
9. (*positive, negative, sign*) With 2's-complement representation and an 8-bit adder-subtracter no overflow is possible if the true sum is between −128 and +127.

PROBLEMS

6-1. Add these 8-bit numbers:
a. 0001 0000 and 0000 1000
b. 0001 1000 and 0000 1100
c. 0001 1100 and 0000 1110
d. 0010 1000 and 0011 1011
After you have each binary sum, convert it to hexadecimal form.

6-2. Add these 16-bit numbers:

```
  1000 0001 1100 1001
+ 0011 0011 0001 0111
```

Express the answer in hexadecimal form.

6-3. In each of the following, convert to binary to do the addition, then convert the answer back to hexadecimal:
a. 2CH + 4FH = ?
b. 5EH + 1AH = ?
c. 3BH + 6DH = ?
d. A5H + 2CH = ?

6-4. Convert each of the following decimal numbers to an 8-bit sign-magnitude number:
a. +27
b. −27
c. +80
d. −80
After you have the sign-magnitude numbers, convert them to hexadecimal form.

6-5. Convert each of these sign-magnitude numbers to its decimal equivalent:
a, 0001 1110
b. 1000 0111
c. 1001 1100
d. 0011 0001

6-6. The following hexadecimal numbers represent sign-magnitude numbers. Convert each to its decimal equivalent.
a. 8FH
b. 3AH
c. 7FH
d. FFH

6-7. Find the 2's complements:
a. 0000 0111
b. 1111 1111
c. 1111 1101
d. 1110 0001
Express your answers in hexadecimal form.

6-8. Convert each of the following to binary. Then take the 2's complement:
a. 4CH
b. 8DH
c. CBH
d. FFH

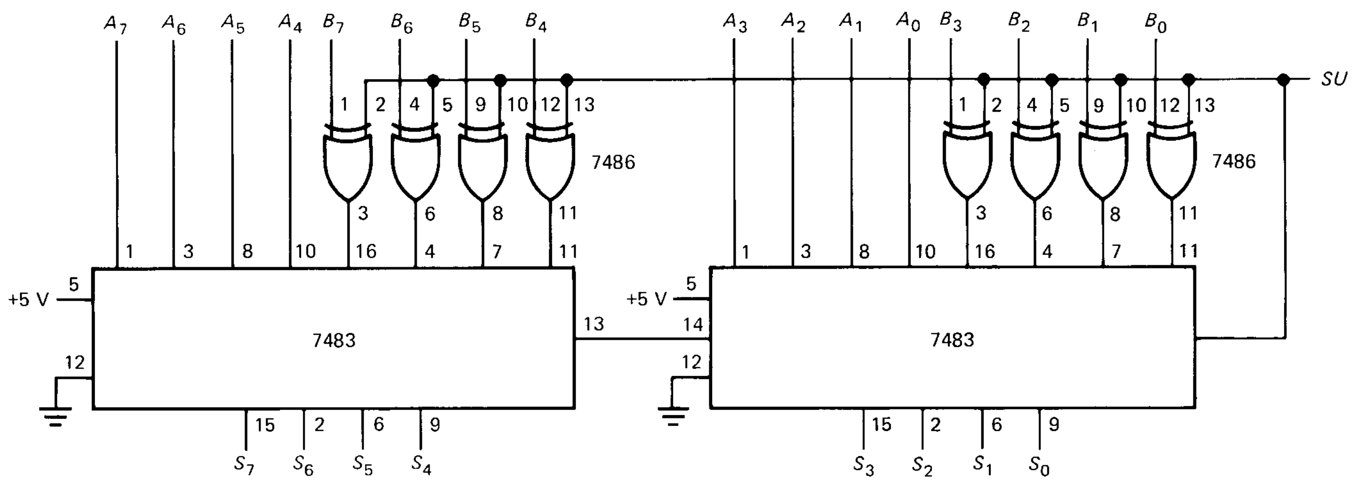


Fig. 6-9

After you have the 2's complements, convert them to hexadecimal form.

6-9. An 8-bit microprocessor uses 2's-complement representation. How do the following decimal numbers appear:

a. −19
b. −48
c. +37
d. −33

Express your answers in binary and hexadecimal form.

6-10. The output of an ALU is EEH. What decimal number does this represent in 2's-complement representation?

6-11. Suppose the inputs to Fig. 6-9 are **A** = 3CH and **B** = 5FH. What is the output for a low *SUB?* A high *SUB?* Express your final answers in hexadecimal form.

6-12. In Fig. 6-9 which of the following inputs cause an overflow when *SUB* is low?

a. 2DH and 4BH
b. 8FH and C3H
c. 5EH and B8H
d. 23H and 14H

6-13. Why are applications for the half-adder limited, what does the full adder do which makes it more useful than the half-adder, and what can be done with a full adder as a result of this feature?

6-14. Since sign-magnitude numbers are fairly easy to understand, why has the 2's-complement system become so widespread?

7

FLIP-FLOPS

Gates are decision-making elements. As shown in the preceding chapter, they can perform binary addition and subtraction. But decision-making elements are not enough. A computer also needs *memory elements,* devices that can store a binary digit. This chapter is about memory elements called *flip-flops*.

7-1 *RS* LATCHES

A flip-flop is a device with two stable states; it remains in one of these states until triggered into the other. The *RS* latch, discussed in this section, is one of the simplest flip-flops.

Transistor Latch

In Fig. 7-1*a* each collector drives the opposite base through a 100-kΩ resisitor. In a circuit like this, one of the transistors is saturated and the other is cut off.

For instance, if the right transistor is saturated, its collector voltage is approximately 0 V. This means that there is no base drive for the left transistor, so it cuts off and its collector voltage approaches +5 V. This high voltage produces enough base current in the right transistor to sustain its saturation. The overall circuit is *latched* with the left transistor cut off (dark shading) and the right transistor saturated. Q is approximately 0 V.

By a similar argument, if the left transistor is saturated, the right transistor is cut off. Figure 7-1*b* illustrates this other state. Q is approximately 5 V for this condition.

Output Q can be low or high, binary 0 or 1. If latched as shown in Fig. 7-1*a*, the circuit is storing a binary 0 because

$$Q = 0$$

On the other hand, when latched as shown in Fig. 7-1*b*, the circuit stores a binary 1 because

$$Q = 1$$

Control Inputs

To control the bit stored in the latch, we can add the inputs shown in Fig. 7-1*c*. These control inputs will be either low (0 V) or high (+5 V). A high *set* input S forces the left transistor to saturate. As soon as the left transistor saturates, the overall circuit latches and

$$Q = 1$$

Once set, the output will remain a 1 even though the S input goes back to 0 V.

A high *reset* input R drives the right transistor into saturation. Once this happens, the circuit latches and

$$Q = 0$$

The output stays latched in the 0 state, even though the R input returns to a low.

In Fig. 7-1*c*, Q represents the stored bit. A complementary output $\overline{Q}$ is available from the collector of the left transistor. This may or may not be used, depending on the application.

Truth Table

Table 7-1 summarizes the operation of the transistor latch. With both control inputs low, no change can occur in the output and the circuit remains latched in its last state. This condition is called the *inactive state* because nothing changes.

TABLE 7-1. TRANSISTOR LATCH

R	*S*	*Q*	Comments
0	0	NC	No change
0	1	1	Set
1	0	0	Reset
1	1	*	Race

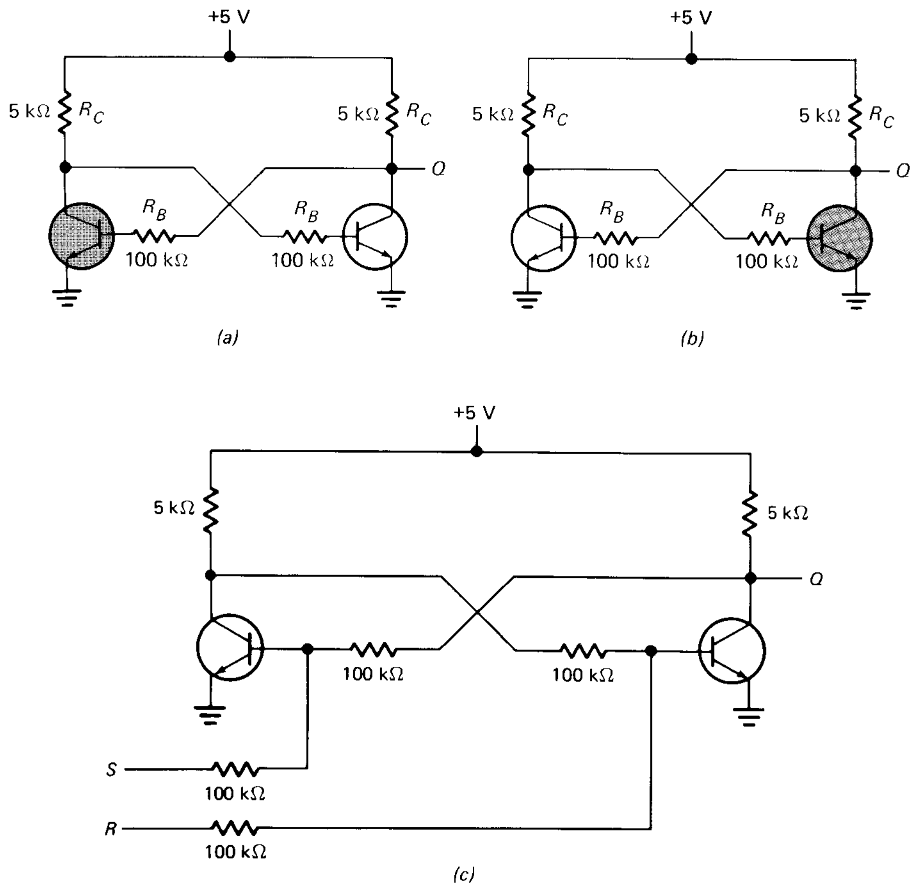


Fig. 7-1 (*a*) Latched state; (*b*) alternative state; (*c*) trigger inputs.

When R is low and S is high, the circuit sets the Q output to a high. On the other hand, if R is high and S is low, the Q output resets to a low.

Race Condition

Look at the last entry in Table 7-1. R and S are high simultaneously. This is called a *race condition;* it is never used because it leads to unpredictable operation.

Here's why. If both control inputs are high, both transistors saturate. When the R and S inputs return to low, both transistors try to come out of saturation. It is a race between the transistors to see which one desaturates first. The faster transistor (the one with the shorter saturation delay time) will win the race and latch the circuit. If the faster transistor is on the left side of Fig. 7-1*c*, the Q output will be low. If the faster transistor is on the right side, the Q output will go high. In mass production, either transistor can be faster; therefore, the Q output is unpredictable. This is why the race condition must be avoided.

Here's how to recognize a race condition. If simultaneously changing both inputs to a memory element leads to an unpredictable output, you've got a race condition. With the transistor latch, $R = 1$ and $S = 1$ is a race condition because simultaneously returning R and S to 0 forces Q into a random state.

From now on, an asterisk in a truth table (see Table 7-1) indicates a race condition, sometimes called a forbidden or invalid state.

NOR Latches

A discrete circuit like Fig. 7-1*c* is rarely used because we are in the age of integrated circuits. Nowadays, you build *RS* latches with NOR gates or NAND gates.

Figure 7-2*a* shows how it's done with NOR gates. Figure 7-2*b* is the De Morgan equivalent. As shown in Table 7-2, a low R and a low S give us the inactive state; the circuit stores or remembers. A low R and a high S represent the set state, while a high R and a low S give the reset state. Finally, a high R and a high S produce a race condition; therefore, we must avoid $R = 1$ and $S = 1$ when using a NOR latch.

Figure 7-2*c* is a *timing diagram;* it shows how the input signals interact to produce the output signal. As you see, the Q output goes high when S goes high. Q remains high after S goes low. Q returns to low when R goes high, and stays low after R returns to low.

TABLE 7-2. NOR LATCH

R	S	Q	Comment
0	0	NC	No change
0	1	1	Set
1	0	0	Reset
1	1	*	Race

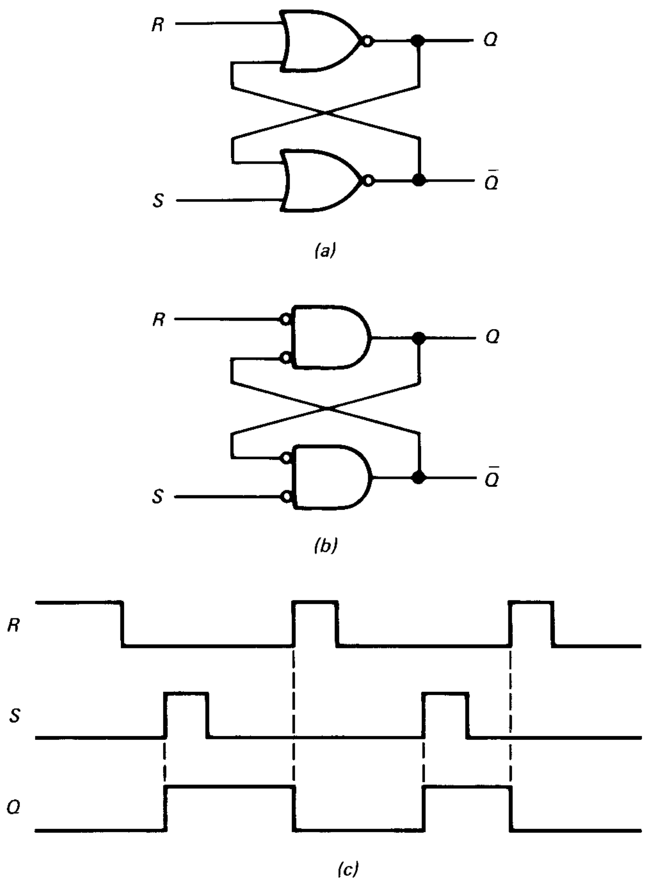


Fig. 7-2 (*a*) NOR latch; (*b*) De Morgan equivalent; (*c*) timing diagram.

NAND Latches

If you prefer using NAND gates, you can build an RS latch as shown in Fig. 7-3*a*. Sometimes it is convenient to draw the De Morgan equivalent shown in Fig. 7-3*b*. In either case, a low R and a high S set Q to high; a high R and a low S reset Q to low.

Because of the NAND-gate inversion, the inactive and race conditions are reversed. In other words, $R = 1$ and $S = 1$ becomes the inactive state; $R = 0$ and $S = 0$ becomes the race condition (see Table 7-3). Therefore, whenever you use a NAND latch, you must avoid having both inputs low at the same time. (To remember the race condition for a NAND latch, glance at Fig. 7-3*b*. If $R = 0$ and $S = 0$, then $Q = 1$ and $\overline{Q} = 1$; both outputs are the same, indicating an invalid condition.)

TABLE 7-3. NAND LATCH

R	S	Q	Comment
0	0	*	Race
0	1	1	Set
1	0	0	Reset
1	1	NC	No change

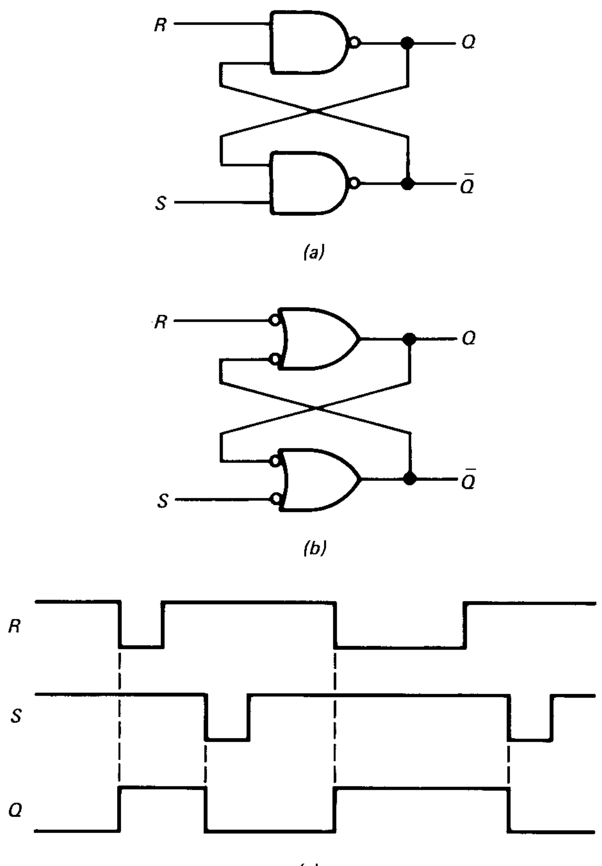
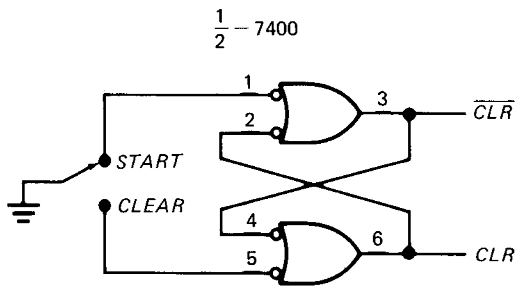


Fig. 7-3 (*a*) NAND latch; (*b*) De Morgan equivalent; (*c*) timing diagram.

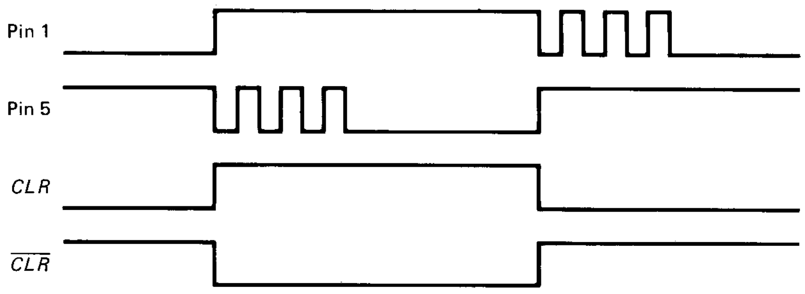
Figure 7-3*c* shows the timing diagram for a NAND latch. R and S are normally high to avoid the race condition. Only one of them goes low at any time. As you see, the Q output goes high whenever R goes low; the Q output goes low whenever S goes low.

Switch Debouncers

RS latches are often used as *switch debouncers*. Whenever you throw a switch from the open to the closed position, the contacts bounce and the switch alternately makes and breaks for a few milliseconds before finally settling in the closed position. One way to eliminate the effects of contact bounce is to use an *RS* latch in conjunction with the switch. The following example explains the idea.



(a)



(b)

Fig. 7-4 Switch debouncer.

EXAMPLE 7-1

Figure 7-4*a* shows a switch debouncer. What does it do?

SOLUTION

As discussed in Chap. 4, floating TTL inputs are equivalent to high inputs. With the switch in the START position, pin 1 is low and pin 5 is high; therefore, $\overline{CLR}$ is high and *CLR* is low. When the switch is thrown to the CLEAR position, pin 1 goes high, as shown in Fig. 7-4*b*. Because of contact bounce, pin 5 goes alternately low and high for a few milliseconds before settling in the low state, symbolized by the ideal pulses of Fig. 7-4*b*.The first time pin 5 goes low, the latch sets, *CLR* going high and $\overline{CLR}$ going low. Subsequent bounces have no effect on *CLR* and $\overline{CLR}$ because the latch stays set.

Similarly, when the switch is thrown back to START, pin 1 bounces low and high for a while. The first time pin 1 goes low, *CLR* goes back to low and $\overline{CLR}$ to high. Later bounces have no effect on *CLR* and $\overline{CLR}$.

Registers need clean signals like *CLR* and $\overline{CLR}$ of Fig. 7-4*b* to operate properly. If the bouncing signals on pins 1 and 5 drove the registers, the operation would be erratic. This is why you often see *RS* latches used as switch debouncers.

7-2 LEVEL CLOCKING

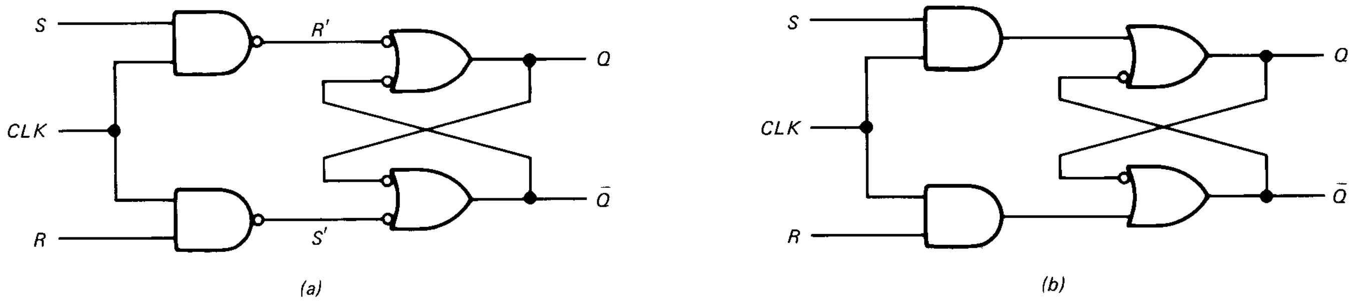
Computers use thousands of flip-flops. To coordinate the overall action, a square-wave signal called the *clock* is sent to each flip-flop. This signal prevents the flip-flops from changing states until the right time.

Clocked Latch

In Fig. 7-5*a* a pair of NAND gates drive a NAND latch. *S* and *R* signals drive the input gates. To avoid confusion, the inner control signals are labeled *R'* and *S'*. The NAND latch works as previously described; a low *R'* and a high *S'* set *Q* to 1, whereas a high *R'* and a low *S'* reset *Q* to 0. Furthermore, a low *R'* and *S'* represent the race condition; therefore, *R'* and *S'* are normally high when the latch is inactive. Because of the inversion through the input NAND gates, the *S* input has to drive the upper NAND input and the *R* input must drive the lower NAND input.

Double Inversions Cancel

When analyzing the operation of this and similar circuits, remember that a double inversion (two bubbles in a series path) cancels out; this makes it appear as though two AND gates drove OR gates, as shown in Fig. 7-5*b*. In this way, you can see at a glance that a high *S* and high *CLK* force



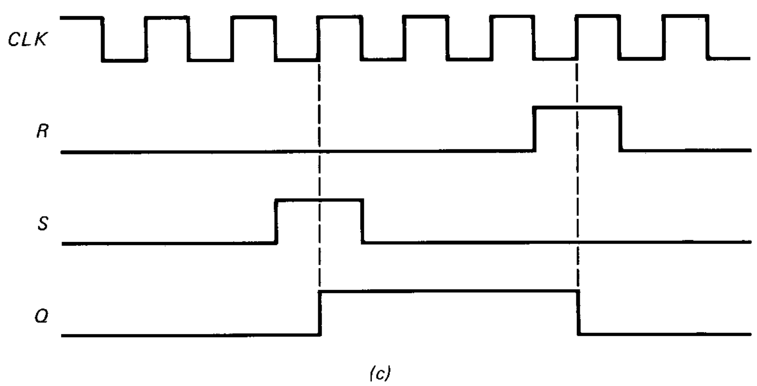


Fig. 7-5 (*a*) Clocked latch; (*b*) equivalent circuit; (*c*) timing diagram.

Q to go high. In other words, even though you are looking at Fig. 7-5*a*, in your mind you should see Fig. 7-5*b*.

Positive Clocking

In Fig. 7-5*a* the clock is a square-wave signal. Because the clock (abbreviated CLK) drives both NAND gates, a low *CLK* prevents *S* and *R* from controlling the latch. If a high *S* and a low *R* drive the gate inputs, the latch must wait until the clock goes high before *Q* can be set to 1. Similarly, given a low *S* and a high *R*, the latch must wait for a high *CLK* before *Q* can reset to 0. This is an example of *positive clocking,* making a latch wait until the clock signal is high before the output can change.

Negative clocking is similar. Visualize an inverter between CLK and the input gates of Fig. 7-5*a*. In this case, the latch must wait until *CLK* is low before the output can change.

Positive and negative clocking are often called *level clocking* because the flip-flop responds to the level (high or low) of the clock signal. Level clocking is the simplest way to control flip-flops with a clock. Later, we will discuss more advanced methods called edge triggering and master-slave clocking.

Race Condition

What about the race condition? When the clock is low in Fig. 7-5*a*, *R'* and *S'* are high, which is a stable condition. The only way to get a race condition is to have a high *CLK,* high *R,* and high *S*. Therefore, normal operation of this circuit requires that *R* and *S* never both be high when the clock goes high.

Timing Diagram and Truth Table

Figure 7-5*c* shows the timing diagram. *Q* goes high when *S* is high and *CLK* goes high. *Q* returns to the low state when *R* is high and *CLK* goes high. Using a common *CLK* signal to drive many flip-flops allows us to synchronize the operation of the different sections of a computer.

Table 7-4 summarizes the operation of the clocked NAND latch. When the clock is low, the output is latched in its last state. When the clock goes high, the circuit will set if *S* is high or reset if *R* is high. *CLK, R,* and *S* all high is a race condition, which is never used deliberately.

TABLE 7-4. CLOCKED NAND LATCH

CLK	*R*	*S*	*Q*
0	0	0	NC
0	0	1	NC
0	1	0	NC
0	1	1	NC
1	0	0	NC
1	0	1	1
1	1	0	0
1	1	1	*

7-3 *D* LATCHES

Since the *RS* flip-flop is susceptible to a race condition, we will modify the design to eliminate the possibility of a race condition. The result is a new kind of flip-flop known as a *D latch*.

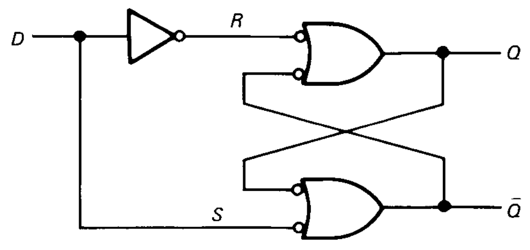


Fig. 7-6 *D* latch.

Unclocked

Figure 7-6 shows one way to build a *D* latch. Because of the inverter, data bit *D* drives the *S* input of a NAND latch and the complement $\bar{D}$ drives the *R* input. Therefore, a high *D* sets the latch, and a low *D* resets it. Table 7-5 summarizes the operation of the *D* latch. Especially important, there is no race condition in this truth table. The inverter guarantees that *S* and *R* will always be in opposite states; therefore, it's impossible to set up a race condition in the *D* latch.

The *D* latch of Fig. 7-6 is unclocked; it will set or reset as soon as *D* goes high or low. An unclocked flip-flop like this is almost never used.

TABLE 7-5. UNCLOCKED *D* LATCH

D	*Q*
0	0
1	1

TABLE 7-6. CLOCKED *D* LATCH

CLK	*D*	*Q*
0	X	NC
1	0	0
1	1	1

Clocked

Figure 7-7*a* is level-clocked. A low *CLK* disables the input gates and prevents the latch from changing states. In other words, while *CLK* is low, the latch is in the inactive state and the circuit stores or remembers. When *CLK* is high, *D* controls the output. A high *D* sets the latch, while a low *D* resets it.

Table 7-6 summarizes the operation. X represents a don't-care condition; it stands for either 0 or 1. While *CLK* is low, the output cannot change, no matter what *D* is. When *CLK* is high, however, the output equals the input

$$Q = D$$

Figure 7-7*b* shows a timing diagram. If the clock is low, the circuit is latched and the *Q* output cannot be changed. While the clock is high, however, *Q* equals *D*; when *D* goes high, *Q* goes high; when *D* goes low, *Q* goes low. The latch is *transparent*, meaning that the output follows the value of *D* while the clock is high.

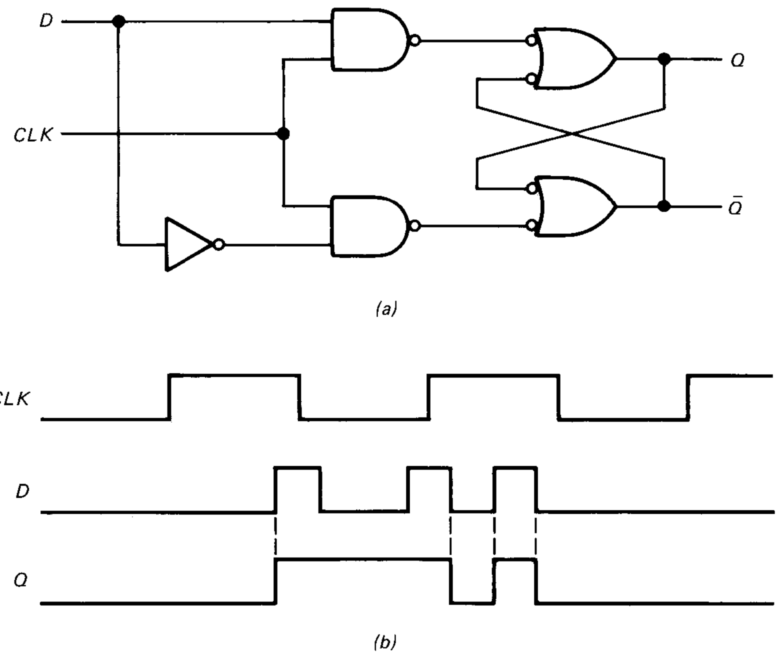


Fig. 7-7 Clocked *D* latch.

Disadvantage

Because the *D* latch is level-clocked, it has a serious disadvantage. While the clock is high, the output follows the value of *D*. Transparent latches may be all right in some applications but not in the computer circuits we will be discussing. To be truly useful, the circuit of Fig. 7-7*a* needs a slight modification.

7-4 EDGE-TRIGGERED *D* FLIP-FLOPS

Now we're ready to talk about the most common type of *D* flip-flop. What a practical computer needs is a *D* flip-flop that samples the data bit at a unique instant.

Edge Triggering

Figure 7-8*a* shows an *RC* circuit at the input of a *D* flip-flop. By deliberate design, the *RC* time constant is much smaller than the clock's pulse width. Because of this, the capacitor can charge fully when *CLK* goes high; this exponential charging produces a narrow positive voltage spike across the resistor. Later, the trailing edge of the clock pulse results in a narrow negative spike.

The narrow positive spike enables the input gates for an instant; the narrow negative spike does nothing. The effect is to activate the input gates during the positive spike, equivalent to sampling the value of *D* for an instant. At this unique time, *D* and its complement hit the flip-flop inputs, forcing *Q* to set or reset.

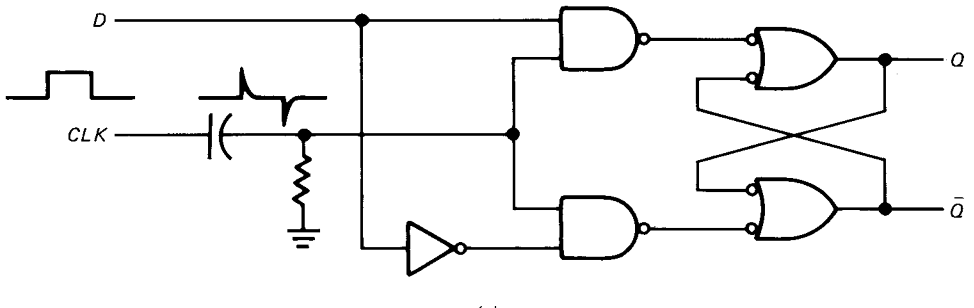
TABLE 7-7. EDGE-TRIGGERED *D* FLIP-FLOP

CLK	*D*	*Q*
0	X	NC
1	X	NC
↓	X	NC
↑	0	0
↑	1	1

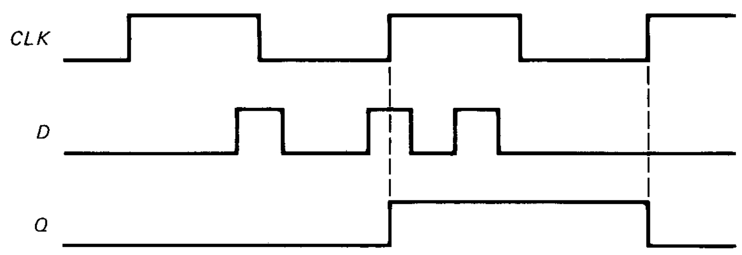
This kind of operation is called *edge triggering* because the flip-flop responds only when the clock is changing states. The triggering in Fig. 7-8*a* occurs on the positive-going edge of the clock; this is why it's referred to as *positive-edge triggering*.

Figure 7-8*b* illustrates the action. The crucial idea is that the output changes only on the rising edge of the clock. In other words, data is stored only on the positive-going edge.

Table 7-7 summarizes the operation of the positive-edge-triggered *D* flip-flop. The up and down arrows represent the rising and falling edges of the clock. The first three entries indicate that there's no output change when the clock is low, high, or on its negative edge. The last two entries indicate an output change on the positive edge of the clock. In other words, input data *D* is stored only on the positive-going edge of the clock.



(a)



(b)

Fig. 7-8 Edge-triggered *D* flip-flop.

Edge Triggering versus Level Clocking

When a circuit is edge-triggered, the output can change only on the rising (or falling) edge of the clock. But when the circuit is level-clocked, the output can change while the clock is high (or low). With edge triggering, the output can change only at one instant during the clock cycle; with level clocking, the output can change during an entire half cycle of the clock.

Preset and Clear

When power is first applied, flip-flops come up in random states. To get some computers started, an operator has to push a master reset button. This sends a *clear* (reset) signal to all flip-flops. Also, it is necessary in some computers to *preset* (synonymous with "set") certain flip-flops before a computer run.

Figure 7-9 shows how to include both functions in a *D* flip-flop. The edge triggering is the same as previously described. In addition, the AND gates allow us to slip in a low *PRESET* or low *CLEAR* when desired. A low *PRESET* forces *Q* to equal 1; a low *CLEAR* resets *Q* to 0.

Table 7-8 summarizes the circuit action. When *PRESET* and *CLEAR* are both low, we get a race condition; therefore, *PRESET* and *CLEAR* should be kept high when inactive. Take *PRESET* low by itself and you set the flip-flop; take *CLEAR* low by itself and you reset the flip-flop. As shown in the remaining entries, the output changes only on the positive-going edge of the clock.

Preset is sometimes called *direct set*, and clear is sometimes called *direct reset*. The word "direct" means unclocked. For instance, the clear signal may come from a push button; regardless of what the clock is doing, the output will reset when the operator pushes the clear button.

The preset and clear inputs override the other inputs; they have first priority. For example, when *PRESET* goes low, the *Q* output goes high and stays there no matter what the *D* and *CLK* inputs are doing. The output will remain high as long as *PRESET* is low. Therefore, the normal procedure in presetting is to take the *PRESET* low temporarily, then return it to high. Similarly, for the clear function: take *CLEAR* low briefly to reset the flip-flop, then take it back to high to allow the circuit to operate.

TABLE 7-8. *D* FLIP-FLOP WITH PRESET AND CLEAR

PRESET	*CLEAR*	*CLK*	*D*	*Q*
0	0	X	X	*
0	1	X	X	1
1	0	X	X	0
1	1	0	X	NC
1	1	1	X	NC
1	1	↓	X	NC
1	1	↑	0	0
1	1	↑	1	1

Direct-Coupled Edge-Triggered *D* Flip-Flop

Integrated *D* flip-flops do not use *RC* circuits to get narrow spikes because capacitors are difficult to fabricate on a chip. Instead, a variety of direct-coupled designs is used. As an example, Fig. 7-10 shows a positive-edge-triggered *D* flip-flop. This direct-coupled circuit has no capacitors, only NAND gates. The analysis is too long and complicated to go into here, but the idea is the same as previously discussed. The circuit responds only during the brief instant the clock switches from low to high. That is, data bit *D* is stored only on the positive-going edge of the clock.

Logic Symbol

Figure 7-11 is the symbol of a positive-edge-triggered *D* flip-flop. The *CLK* input has a small triangle, a reminder of the edge triggering. When you see this schematic symbol, remember what it means: the *D* input is stored on the rising edge of the clock.

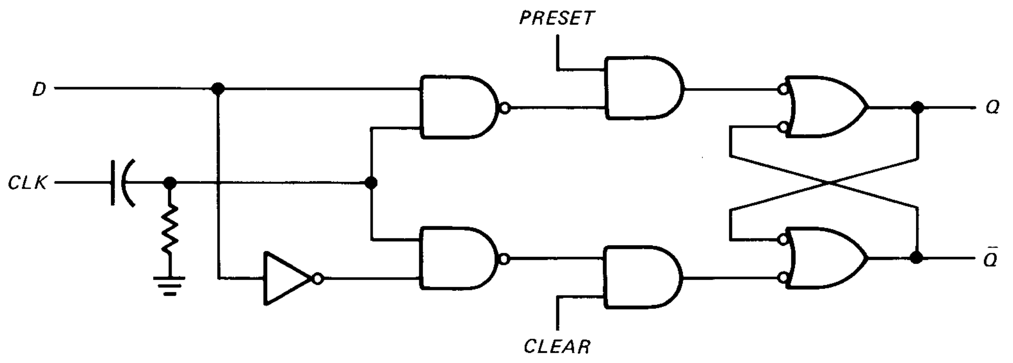


Fig. 7-9 Edge-triggered *D* flip-flop with preset and clear.

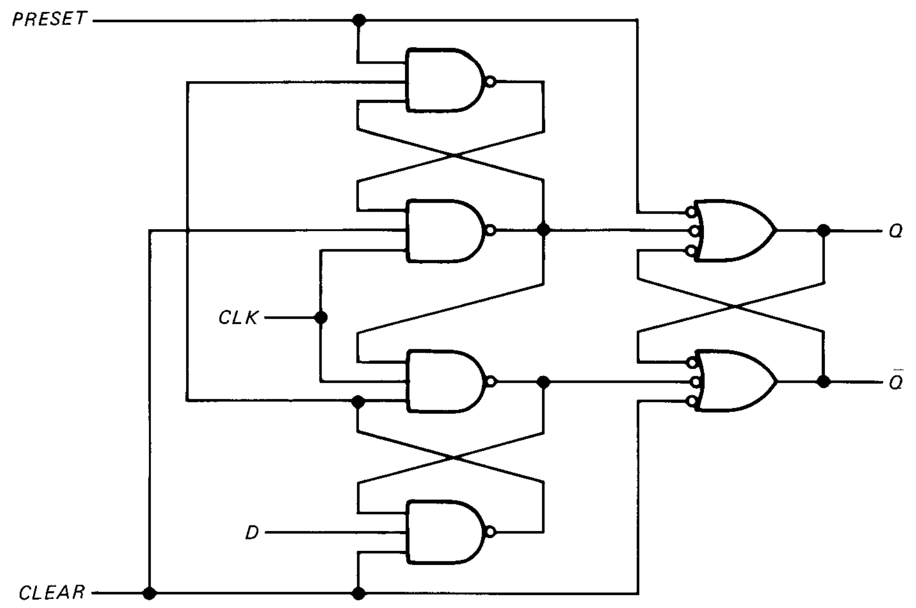


Fig. 7-10 Direct-coupled edge-triggered *D* flip-flop.

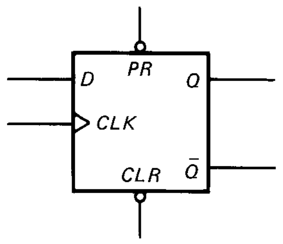


Fig. 7-11 Logic symbol for edge-triggered *D* flip-flop.

Figure 7-11 also includes preset (*PR*) and clear (*CLR*) inputs. The bubbles indicate an *active low state*. In other words, the preset and clear inputs are high when inactive. To preset the flip-flop, the preset input must go low temporarily and then be returned to high. Similarly, to reset the flip-flop, the clear input must go low, then back to high.

The same idea applies to circuits discussed later. A bubble at an input means an active low state: the input has to go low to produce an effect. When no bubble is present, the input has to go high to have an effect.

Propagation Delay Time

Diodes and transistors cannot switch states instantaneously. It always takes a small amount of time to turn a diode on or off. Likewise, it takes a time for a transistor to switch from saturation to cutoff or vice versa. For bipolar diodes and transistors, switching time is in the nanosecond region.

Switching time is the main cause of *propagation delay time* t_p. This represents the amount of time it takes for the output of a gate or flip-flop to change states. For instance, if the data sheet of a *D* flip-flop indicates a t_p of 10 ns, it takes approximately 10 ns for *Q* to change states after *D* has been sampled by the clock edge.

Propagation delay time is so small that it's negligible in many applications, but in high-speed circuits you have to take it into account. If a flip-flop has a t_p of 10 ns, this means that you have to wait 10 ns before the output can trigger another circuit.

Setup Time

Stray capacitance at the *D* input (plus other factors) makes it necessary for data bit *D* to be at the input before the *CLK* edge arrives. The *setup time* t_{setup} is the minimum length of time the data bit must be present before the *CLK* edge hits.

For instance, if the data sheet of a *D* flip-flop indicates a t_{setup} of 15 ns, the data bit to be stored must be at the *D* input at least 15 ns before the *CLK* edge arrives; otherwise, the IC manufacturer does not guarantee correct sampling and storing.

Hold Time

Furthermore, data bit *D* has to be held long enough for the internal transistors to switch states. Only after the transition is assured can we allow data bit *D* to change. *Hold time* t_{hold} is the minimum length of time the data bit must be present after the *CLK* edge has struck.

For example, if t_{setup} is 15 ns and t_{hold} is 5 ns, the data bit has to be at the *D* input at least 15 ns before the *CLK* edge arrives and held at least 5 ns after the *CLK* edge hits.

7-5 EDGE-TRIGGERED JK FLIP-FLOPS

The next chapter shows you how to build a counter, the electronic equivalent of a binary odometer. When it comes to circuits that count, the *JK flip-flop* is the ideal memory element to use.

Circuit

Figure 7-12*a* shows one way to build a *JK* flip-flop. As before, an *RC* circuit with a short time constant converts the rectangular *CLK* pulse to narrow spikes. Because of the double inversion through the NAND gates, the circuit is positive-edge-triggered. In other words, the input gates are enabled only on the rising edge of the clock.

Inactive

The *J* and *K* inputs are control inputs; they determine what the circuit will do on the positive clock edge. When *J* and *K* are low, both input gates are disabled and the circuit is inactive at all times including the rising edge of the clock.

Reset

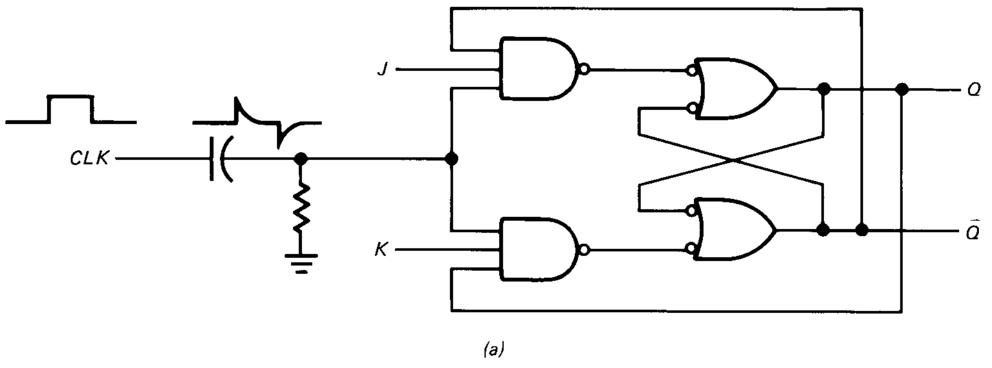
When *J* is low and *K* is high, the upper gate is disabled; so there's no way to set the flip-flop. The only possibility is reset. When *Q* is high, the lower gate passes a reset trigger as soon as the positive clock edge arrives. This forces *Q* to become low. Therefore, $J = 0$ and $K = 1$ means that a rising clock edge resets the flip-flop.

Set

When *J* is high and *K* is low, the lower gate is disabled; so it's impossible to reset the flip-flop. But you can set the flip-flop as follows. When *Q* is low, $\overline{Q}$ is high; therefore, the upper gate passes a set trigger on the positive clock edge. This drives *Q* into the high state. That is, $J = 1$ and $K = 0$ means that the next positive clock edge sets the flip-flop.

Toggle

When *J* and *K* are both high, it is possible to set or reset the flip-flop, depending on the current state of the output. If *Q* is high, the lower gate passes a reset trigger on the



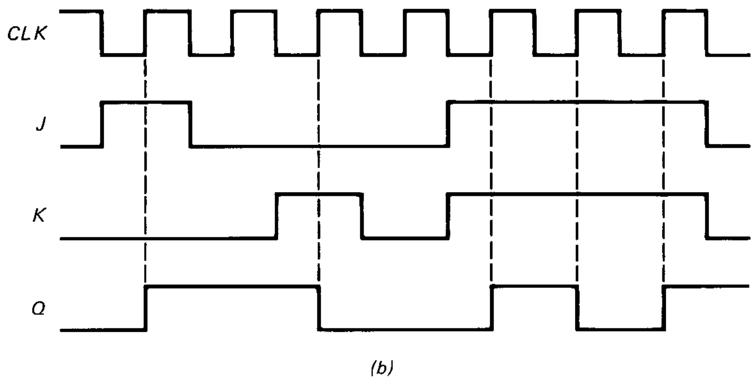


Fig. 7-12 (*a*) Edge-triggered *JK* flip-flop; (*b*) timing diagram.

TABLE 7-9. POSITIVE-EDGE-TRIGGERED *JK* FLIP-FLOP

CLK	*J*	*K*	*Q*
0	X	X	NC
1	X	X	NC
↓	X	X	NC
X	0	0	NC
↑	0	1	0
↑	1	0	1
↑	1	1	Toggle

next positive clock edge. On the other hand, when Q is low, the upper gate passes a set trigger on the next positive clock edge. Either way, Q changes to the complement of the last state. Therefore, $J = 1$ and $K = 1$ means that the flip-flop will *toggle* on the next positive clock edge. (''Toggle'' means switch to opposite state.)

Timing Diagram

The timing diagram of Fig. 7-12*b* is a visual summary of the action. When J is high and K is low, the rising clock edge sets Q to high. On the other hand, when J is low and K is high, the rising clock edge resets Q to low. When J and K are high simultaneously, the output toggles on each rising clock edge.

Truth Table

Table 7-9 summarizes the operation. The circuit is inactive when the clock is low, high, or on its negative edge. Likewise, the circuit is inactive when J and K are both low. Output changes occur only on the rising edge of the clock, as indicated by the last three entries of the table. The output either resets, sets, or toggles.

Racing

The *JK* flip-flop shown in Fig. 7-12*a* has to be edge-triggered to avoid oscillations. Why? Assume that the circuit is level-clocked. In other words, assume that we remove the *RC* circuit and run the clock straight into the gates. With a high J, high K, and high *CLK*, the output will toggle. New outputs are then fed back to the input gates. After two propagation times (input and output gates), the output toggles again. And once more, new outputs return to the input gates. In this way, the output can toggle repeatedly as long as the clock is high. That is, we get oscillations during the positive half cycle of the clock. Toggling more than once during a clock cycle is called *racing*.

Now assume that we put the *RC* circuit back in and return to edge triggering. Propagation delay time prevents the *JK* flip-flop from racing. Here's why. In Fig. 7-12*a* the outputs change after the positive clock edge has struck. By the time the new Q and $\overline{Q}$ signals return to the input gates, the positive spikes have decayed to zero. This is why we get only one toggle during each clock cycle.

For instance, if the total propagation delay time from input to output is 20 ns, the outputs change approximately 20 ns after the rising edge of the clock. If the spikes are narrower than 20 ns, the returning Q and $\overline{Q}$ arrive too late to cause false triggering.

Symbols

As previously mentioned, capacitors are too difficult to fabricate on a chip. This is why manufacturers prefer direct-coupled designs for edge-triggered *JK* flip-flops. Such designs are too complicated to reproduce here, but you can find them in manufacturers' IC data books.

Figure 7-13*a* is the standard symbol for a positive-edge-triggered *JK* flip-flop of any design.

Figure 7-13*b* is the symbol for a *JK* flip-flop with the preset and clear functions. As usual, *PR* and *CLR* have active low states. This means that they are normally high and taken low temporarily to preset or clear the circuit.

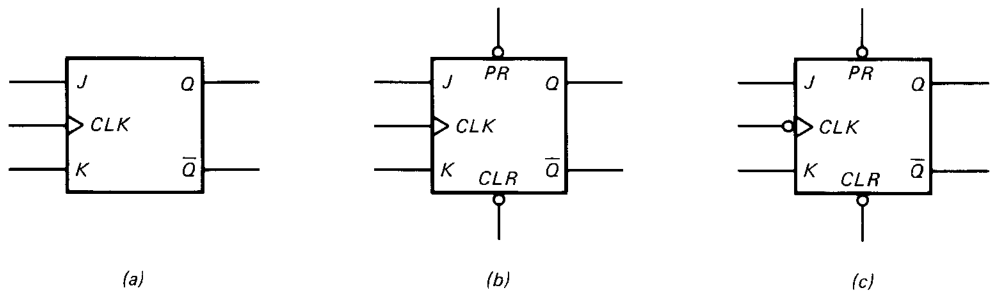
Figure 7-13*c* is another commercially available *JK* flip-flop. The bubble on the clock input is the standard way to indicate negative-edge triggering. As shown in Table 7-10, the output can change only on the *falling* edge of the clock. The timing diagram of Fig. 7-13*d* emphasizes this negative-edge triggering.

7-6 *JK* MASTER-SLAVE FLIP-FLOP

Figure 7-14 shows a *JK master-slave* flip-flop, another way to avoid racing. A master-slave flip-flop is a combination of two clocked latches; the first is called the *master,* and the second is the *slave*. Notice that the master is positively

TABLE 7-10. NEGATIVE-EDGE-TRIGGERED *JK* FLIP-FLOP

CLK	*J*	*K*	*Q*
0	X	X	NC
1	X	X	NC
↑	X	X	NC
X	0	0	NC
↓	0	1	0
↓	1	0	1
↓	1	1	Toggle



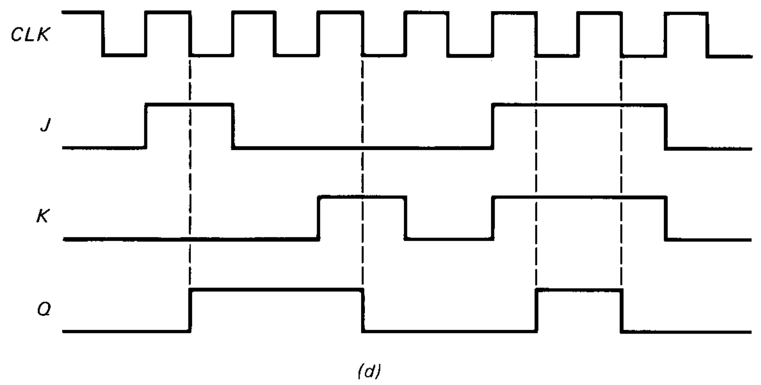


Fig. 7-13 (*a*) Positive-edge triggering; (*b*) active low preset and clear; (*c*) negative-edge triggering; (*d*) timing diagram.

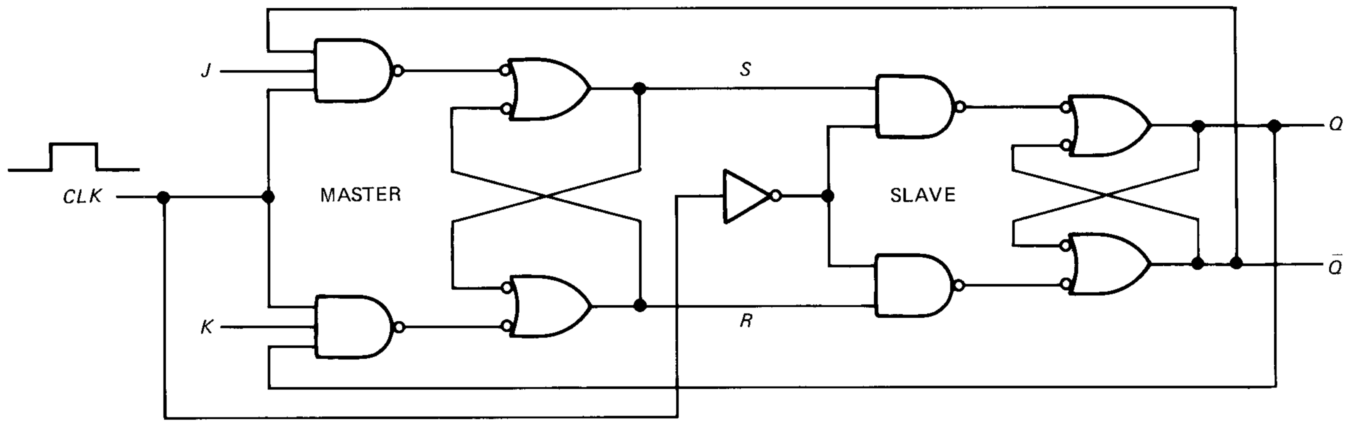


Fig. 7-14 Master-slave *JK* flip-flop.

clocked but the slave is negatively clocked. This implies the following:

1. While the clock is high, the master is active and the slave is inactive.
2. While the clock is low, the master is inactive and the slave is active.

Set

To start the analysis, let's assume low Q and high $\overline{Q}$. For an input condition of high J, low K, and high CLK, the master goes into the set state, producing high S and low R. Nothing happens to the Q and $\overline{Q}$ outputs because the slave is inactive while the clock is high. When the clock goes low, however, the high S and low R force the slave into the set state, producing a high Q and a low $\overline{Q}$.

There are two distinct steps in setting the final Q output. First, the master is set while the clock is high. Second, the slave is set while the clock is low. This action is sometimes called *cocking* and *triggering*. You cock the master during the positive half cycle of the clock, and you trigger the slave during the negative half cycle of the clock.

Reset

When the slave is set, Q is high and $\overline{Q}$ is low. For the input condition of low J, high K, and high CLK, the master will reset, forcing S to go low and R to go high. Again, no changes can occur in Q and $\overline{Q}$ because the slave is inactive while the clock is high. When the clock returns to the low state, the low S and high R force the slave to reset; this forces Q to go low and $\overline{Q}$ to go high.

Again, notice the cocking and triggering. This is the key idea behind the master-slave flip-flop. Every action of the master with a high *CLK* is copied by the slave when the clock goes low.

Toggle

If the J and K inputs are both high, the master toggles once while the clock is high; the slave then toggles once when the clock goes low. No matter what the master does, the slave copies it. If the master toggles into the set state, the slave toggles into the set state. If the master toggles into the reset state, the slave toggles into the reset state.

Level Clocking

The master-slave flip-flop is level-clocked in Fig. 7-14. While the clock is high, therefore, any changes in J and K can affect the S and R outputs. For this reason, you normally keep J and K constant during the positive half cycle of the clock. After the clock goes low, the master becomes inactive and you can allow J and K to change.

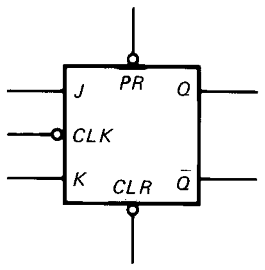


Fig. 7-15 Symbol for master-slave *JK* flip-flop.

Symbol

Figure 7-15 shows the symbol for a *JK* master-slave flip-flop with preset and clear functions. The bubble on the *CLK* input reminds us that the output changes when the clock goes low.

Truth Table

Table 7-11 summarizes the operation of a *JK* master-slave flip-flop. A low *PR* and low *CLR* produces a race condition; therefore, *PR* and *CLR* are normally kept at a high voltage when inactive. To clear, you take *CLR* low; to preset, you take *PR* low. In either case, you return them to high when ready to run.

TABLE 7-11. MASTER-SLAVE FLIP-FLOP

PR	*CLR*	*CLK*	*J*	*K*	*Q*
0	0	X	X	X	*
0	1	X	X	X	1
1	0	X	X	X	0
1	1	X	0	0	NC
1	1	⎍	0	1	0
1	1	⎍	1	0	1
1	1	⎍	1	1	Toggle

As before, low J and low K produce an inactive state, regardless of the what the clock is doing. If K goes high by itself, the next clock pulse resets the flip-flop. If J goes high by itself, the next clock pulse sets the flip-flop. When J and K are both high, each clock pulse produces one toggle.

EXAMPLE 7-2

Figure 7-16*a* shows a *clock generator*. What does it do when $\overline{HLT}$ is high?

SOLUTION

To begin with, the 555 is an IC that can generate a rectangular output when connected as shown in Fig. 7-16*a*. The frequency of the output is

$$f = \frac{1.44}{(R_A + 2R_B)C}$$

The duty cycle (ratio of high state to period) is

$$D = \frac{R_A + R_B}{R_A + 2R_B}$$

With the values shown in Fig. 7-16*a* the frequency of the output is

$$f = \frac{1.44}{(36\text{ k}\Omega + 36\text{ k}\Omega)(0.01\ \mu\text{F})} = 2\text{ kHz}$$

and the duty cycle is

$$D = \frac{36\text{ k}\Omega + 18\text{ k}\Omega}{36\text{ k}\Omega + 36\text{ k}\Omega} = 0.75$$

which is equivalent to 75 percent.

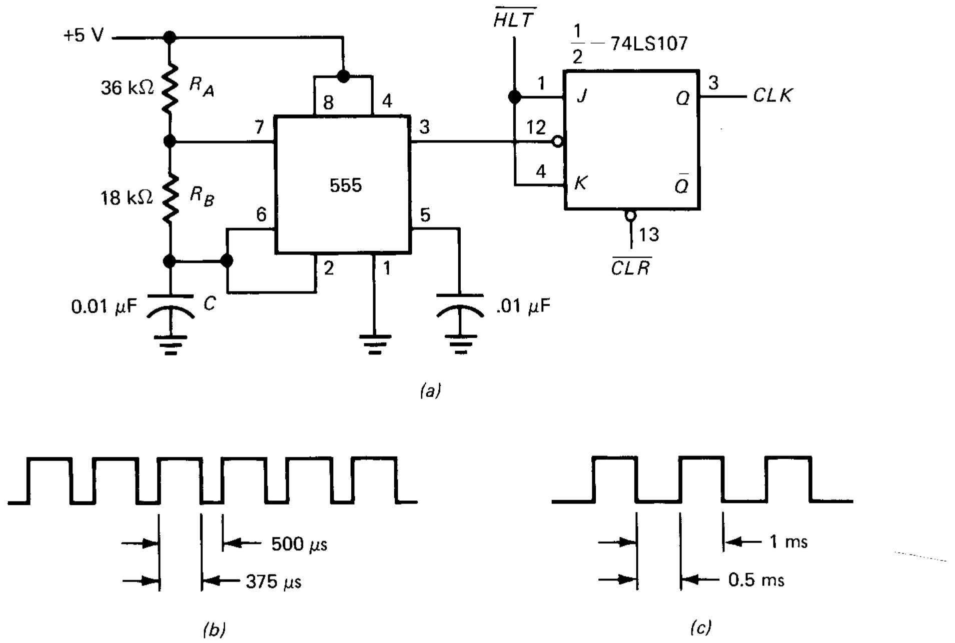


Fig. 7-16 Clock generator: (*a*) circuit; (*b*) 555 output; (*c*) *JK* flip-flop output.

Figure 7-16*b* illustrates how the output (pin 3) of the 555 looks. Note how the signal is high for 75 percent of the cycle. This unsymmetrical output drives the clock input of a *JK* master-slave flip-flop.

The *JK* master-slave flip-flop toggles once per input cycle; therefore, its output has a frequency of 1 kHz and a duty cycle of 50 percent. One of the reasons for using the flip-flop is to get the symmetrical output shown in Fig. 7-16*c*.

Another reason for using the flip-flop is to control the starting phase of the clock. A computer run starts with $\overline{CLR}$ going momentarily low, then back to high. This resets the flip-flop, forcing *CLK* to go low. Therefore, the starting phase of the *CLK* signal is always low. You will see the clock generator of Fig. 7-16*a* again in Chap. 10; remember that the *CLK* signal has a frequency of 1 kHz, a duty cycle of 50 percent, and starting phase of low.

GLOSSARY

contact bounce The making and breaking of contacts for a few milliseconds after a switch closes.

edge triggering Changing the output state of a flip-flop on the rising or falling edge of a clock pulse.

flip-flop A two-state circuit that can remain in either state indefinitely. Also called a bistable multivibrator. An external trigger can change the output state.

hold time The minimum amount of time the input signals must be held constant after the clock edge has struck. After a clock edge strikes a flip-flop, the internal transistors need time to change from one state to another. The input control signals (*D*, or *J* and *K*) must be held constant while these internal transistors are switching over.

latch The simplest type of flip-flop, consisting of two cross-coupled NAND or NOR latches.

level clocking A type of triggering in which the output of a flip-flop responds to the level (high or low) of the clock signal. With positive level clocking, for example, the output can change at any time during the positive half cycle.

master-slave triggering A type of triggering using two cascaded latches called the master and the slave. The master is cocked during the positive half cycle of the clock, and the slave is triggered during the negative half cycle.

propagation delay time The time it takes for the output of a gate or flip-flop to change after the inputs have changed.

race condition An undesirable condition which may exist in a system when two or more inputs change simultaneously. If the final output depends on which input changes first, a race condition exists.

setup time The minimum amount of time the inputs to a flip-flop must be present before the clock edge arrives.

toggle Change of the output to the opposite state in a *JK* flip-flop.

SELF-TESTING REVIEW

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

1. A flip-flop is a ________ element that stores a binary digit as a low or high voltage. With an *RS* latch a high *S* and a low *R* sets the output to ________; a low *S* and a high *R* ________ the output to low.

2. (*memory, high, reset*) With a NAND latch a low *R* and a low *S* produce a ________ condition. This is why *R* and *S* are kept high when inactive. One use for latches is switch debouncers; they eliminate the effects of ________ bounce.

3. (*race, contact*) Computers use thousands of flip-flops. To coordinate the overall action, a common signal called the ________ is sent to each flip-flop. With positive clocking the clock signal must be ________ for the flip-flop to respond. Positive and negative clocking are also called level clocking because the flip-flop responds to the ________ of the clock, either high or low.

4. (*clock, high, level*) In a *D* latch, data bit *D* drives the *S* input of a latch, and the complement $\overline{D}$ drives the *R* input; therefore, a high *D* ________ the latch and a low *D* resets it. Since *R* and *S* are always in opposite states in a *D* latch, the ________ condition is impossible.

5. (*sets, race*) With a positive-edge-triggered *D* flip-flop, the data bit is sampled and stored on the ________ edge of the clock pulse. Preset and clear inputs are often called ________ set and ________ reset. These inputs override the other inputs; they have first priority. When preset goes low, the *Q* output goes ________ and stays there no matter what the *D* and *CLK* inputs are doing.

6. (*rising, direct, direct, high*) In a flip-flop, propagation delay time is the amount of time it takes for the ________ to change after the clock edge has struck. Setup time is the amount of time an input signal must be present ________ the clock edge strikes. Hold time is the amount of time an input signal must be present ________ the clock edge strikes.

7. (*output, before, after*) In a positive-edge-triggered *JK* flip-flop, a low *J* and a low *K* produce the ________ state. A high *J* and a high *K* mean that the output will ________ on the rising edge of the clock.

8. (*inactive, toggle*) With a *JK* master-slave flip-flop the master is cocked when the clock is ________, and the slave is triggered when the clock is ________. This type of flip-flop is usually level-clocked instead of edge-triggered. For this reason, *J* and *K* are normally kept ________ while the clock is high.

9. (*high, low, constant*) Since capacitors are too difficult to fabricate on an IC chip, manufacturers rely on various direct-coupled designs for *D* flip-flops and *JK* flip-flops.

PROBLEMS

7-1. The waveforms of Fig. 7-17 drive a clocked *RS* latch (Fig. 7-5*a*). If *Q* is low before time *A*,
a. At what point does *Q* become a 1?
b. When does *Q* reset to 0?

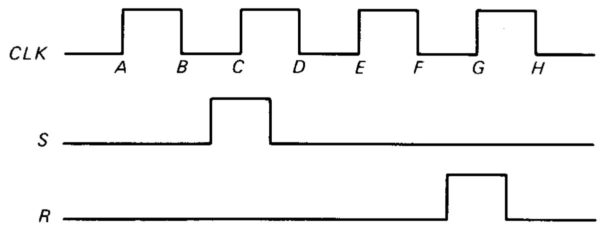


Fig. 7-17

7-2. A *D* flip-flop has these specifications:

$$t_{\text{setup}} = 10 \text{ ns}$$
$$t_{\text{hold}} = 5 \text{ ns}$$
$$t_p = 30 \text{ ns}$$

a. How far ahead of the rising clock edge must the data bit be applied to the *D* input to ensure correct storage?
b. After the rising clock edge, how long must you wait before letting the data bit change?
c. How long after the rising clock edge will *Q* change?

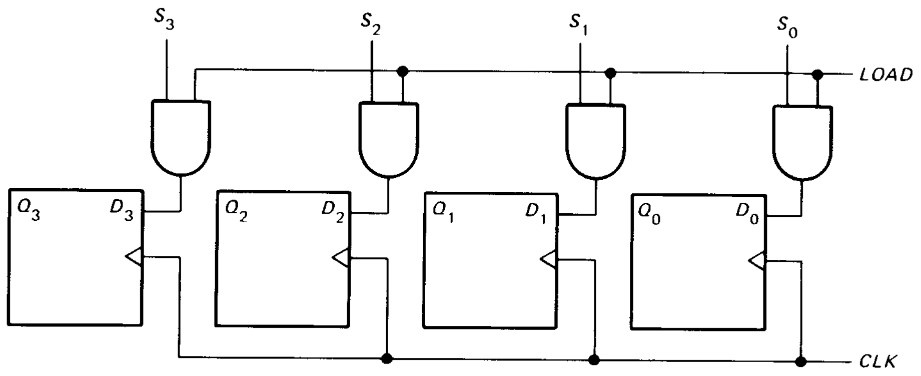


Fig. 7-18

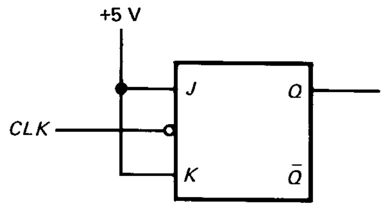


Fig. 7-19

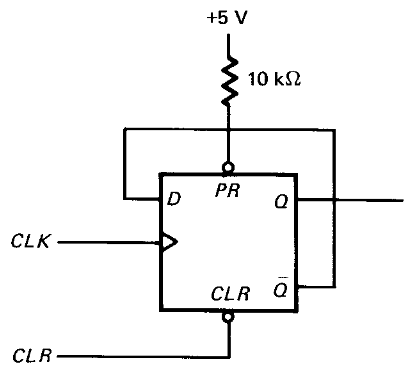


Fig. 7-20

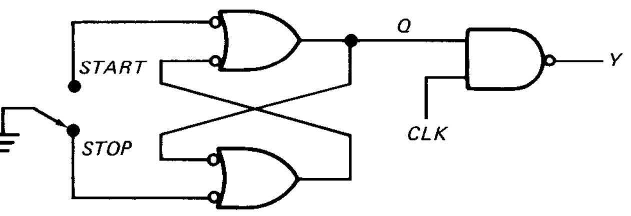


Fig. 7-21

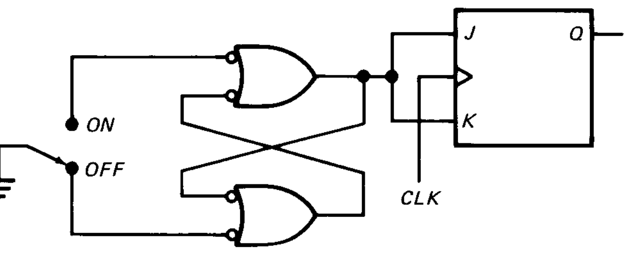


Fig. 7-22

7-3. In Fig. 7-18, the data word to be stored is

$$S = 1001$$

a. If *LOAD* is low, what does *Q* equal after the positive clock edge?
b. If *LOAD* is high, what does *Q* equal after the positive clock edge.

7-4. The clock of Fig. 7-19 has a frequency of 1 MHz, and the flip-flop has a propagation delay time of 25 ns.

a. What is the period of the clock?
b. The frequency of the *Q* output? Its period?
c. How long after the negative clock edge does the *Q* output change?

7-5. The clock has a frequency of 6 MHz in Fig. 7-19. What is the frequency of the *Q* output? This circuit is sometimes called a divide-by-2 circuit. Explain why.

7-6. In Fig. 7-20, *CLR* is taken low temporarily, then high. Draw the timing diagram. If the clock has a frequency of 1 MHz, what is the frequency of the *Q* output? Is this a divide-by-2 circuit?

7-7. Figure 7-21 shows a NAND latch used as a switch debouncer. With the switch in the STOP position, what do *Q* and *Y* equal? If the switch is thrown to the START position, what do *Q* and *Y* equal?

7-8. The clock has a frequency of 1 MHz in Fig. 7-22. With the switch in the OFF position, what is the frequency of the *Q* output? If the switch is thrown to the ON position, what is the frequency of the *Q* output?

8 REGISTERS AND COUNTERS

A *register* is a group of memory elements that work together as a unit. The simplest registers do nothing more than store a binary word; others modify the stored word by shifting its bits left or right or by performing other operations to be discussed in this chapter. A *counter* is a special kind of register, designed to count the number of clock pulses arriving at its input. This chapter discusses some basic registers and counters used in microcomputers.

8-1 BUFFER REGISTERS

A *buffer register* is the simplest kind of register; all it does is store a digital word.

Basic Idea

Figure 8-1 shows a buffer register built with positive-edge-triggered *D* flip-flops. The *X* bits set up the flip-flops for loading. Therefore, when the first positive clock edge arrives, the stored word becomes $Q_3Q_2Q_1Q_0 = X_3X_2X_1X_0$. In chunked notation,

$$\mathbf{Q} = \mathbf{X}$$

The circuit is too primitive to be of any use. What it needs is some control over the *X* bits, some way of holding them off until we're ready to store them.

Controlled

Figure 8-2 is more like it. This is a controlled buffer register with an active-high *CLR*. Therefore, when *CLR* goes high, all flip-flops reset and the stored word becomes

$$\mathbf{Q} = 0000$$

When *CLR* returns low, the register is ready for action.

LOAD is a control input; it determines what the circuit does. When *LOAD* is low, the *X* bits cannot reach the flip-flops. At the same time, the inverted signal $\overline{LOAD}$ is high; this forces each flip-flop output to feed back to its data input. When each rising clock edge arrives, data is circulated or retained. In other words, the register contents are unchanged when *LOAD* is low.

When *LOAD* goes high, the *X* bits are transmitted to the data inputs. After a short setup time, the flip-flops are ready for loading. With the arrival of the positive clock edge, the *X* bits are loaded and the stored word becomes

$$Q_3Q_2Q_1Q_0 = X_3X_2X_1X_0$$

If *LOAD* returns to low, the foregoing word is stored indefinitely; this means that the *X* bits can change without affecting the stored word.

EXAMPLE 8-1

Chapter 10 discusses the *SAP* (simple-as-possible) computer. This educational computer has three generations, SAP-1, SAP-2, and SAP-3. Figure 8-3 shows the output register of the SAP-1 computer. The 74LS173 chips are controlled buffer registers, similar to Fig. 8-2. What does the circuit do?

SOLUTION

To begin with, it is an 8-bit buffer register built with TTL chips. Each chip handles 4 bits of input word **X**. The upper nibble $X_7X_6X_5X_4$ goes to pins 14, 13, 12, and 11 of C22; the lower nibble $X_3X_2X_1X_0$ goes to pins 14, 13, 12, and 11 of the C23.

Output word **Q** drives an 8-bit LED display. The upper nibble $Q_7Q_6Q_5Q_4$ comes out of pins 3, 4, 5, and 6 of C22; the lower nibble $Q_3Q_2Q_1Q_0$ comes out of pins 3, 4, 5, and 6 of C23. The typical high-state output of a 74LS173 is 3.5 V, and the typical LED drop is 1.5 V. Since each current-limiting resistance is 1 kΩ, the high-state current is approximately 2 mA for each output pin.

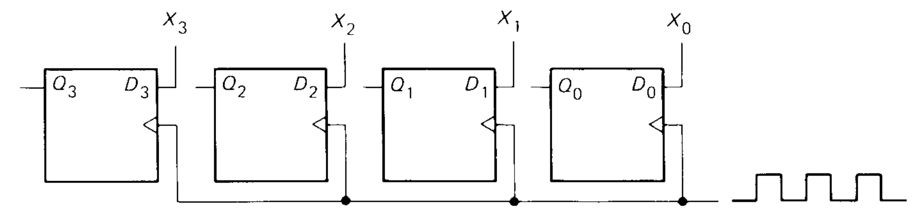


Fig. 8-1 Buffer register.

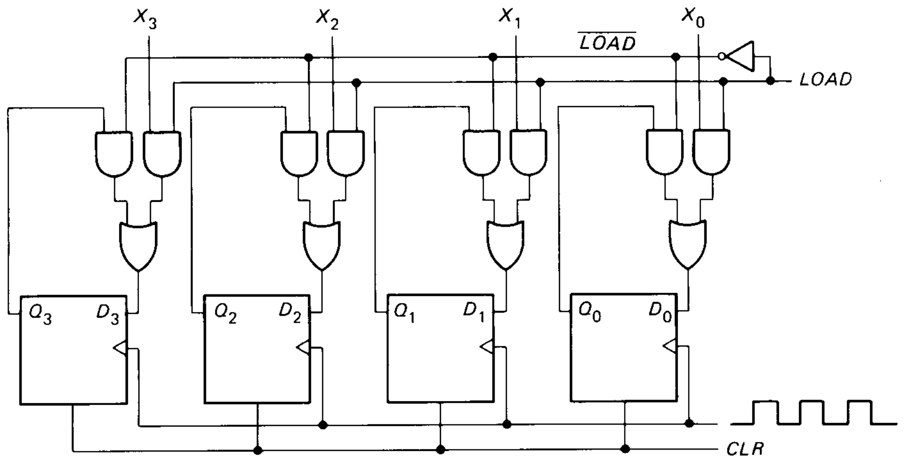
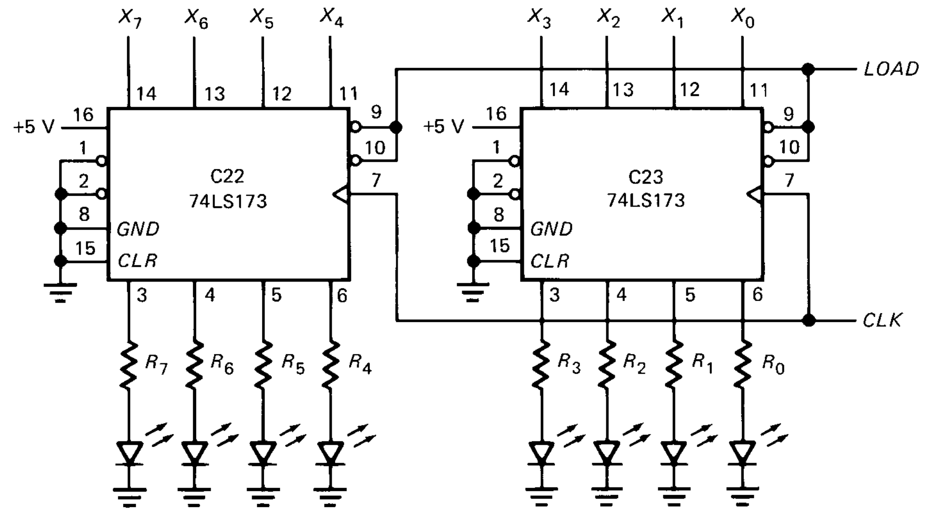


Fig. 8-2 Controlled buffer register.



Note: All resistors are 1 kΩ.

Fig. 8-3 SAP-1 output register.

The 74LS173 requires a 5-V supply for pin 16 and a ground return on pin 8. The SAP-1 output register never needs clearing; this is why the CLR input (pin 15) is made inactive by tying it to ground. In a 74LS173, pins 9 and 10 are separate *LOAD* controls. Because SAP-1 needs only a single *LOAD* control, pins 9 and 10 are tied together. The bubbles on pins 9 and 10 indicate an active low state; this means that *LOAD* must be low for the positive clock edge to store the input word. See Appendix 4 for a more detailed description of the 74LS173.

The action of the circuit is straightforward. While *LOAD* is high, the register contents are unchanged even though the clock is running. To change the stored word, *LOAD* must go low. Then the next rising clock edge loads the *X* bits into the register. As soon as this happens, the LED display shows the new contents.

8-2 SHIFT REGISTERS

A *shift register* moves the stored bits left or right. This bit shifting is essential for certain arithmetic and logic operations used in microcomputers.

Shift Left

Figure 8-4 is a *shift-left* register. As shown, D_{in} sets up the right flip-flop, Q_0 sets up the second flip-flop, Q_1 the third, and so on. When the next positive clock edge strikes, therefore, the stored bits move one position to the left.

As an example, here's what happens with $D_{in} = 1$ and

$$Q = 0000$$

All data inputs except the one on the right are 0s. The arrival of the first rising clock edge sets the right flip-flop, and the stored word becomes

$$Q = 0001$$

This new word means D_1 now equals 1, as well as D_0. When the next positive clock edge hits, the Q_1 flip-flop sets and the register contents become

$$Q = 0011$$

The third positive clock edge results in

$$Q = 0111$$

and the fourth rising clock edge gives

$$Q = 1111$$

Hereafter, the stored word is unchanged as long as $D_{in} = 1$.

Suppose D_{in} is now changed to 0. Then, successive clock pulses produce these register contents:

$$Q = 1110$$
$$Q = 1100$$
$$Q = 1000$$
$$Q = 0000$$

As long as $D_{in} = 0$, subsequent clock pulses have no further effect.

The timing diagram of Fig. 8-5 summarizes the foregoing discussion.

Shift Right

Figure 8-6 is a *shift-right register*. As shown, each Q output sets up the D input of the preceding flip-flop. When the rising clock edge arrives, the stored bits move one position to the right.

Here's an example with $D_{in} = 1$ and

$$Q = 0000$$

All data inputs except the one on the left are 0s. The first positive clock edge sets the left flip-flop and the stored word becomes

$$Q = 1000$$

With the appearance of this word, D_3 and D_2 are 1s. The second rising clock edge gives

$$Q = 1100$$

The third clock pulse gives

$$Q = 1110$$

and the fourth clock pulse gives

$$Q = 1111$$

8-3 CONTROLLED SHIFT REGISTERS

A *controlled shift register* has control inputs that determine what it does on the next clock pulse.

SHL Control

Figure 8-7 shows how the shift-left operation can be controlled. *SHL* is the control signal. When *SHL* is low, the inverted signal $\overline{SHL}$ is high. This forces each flip-flop output to feed back to its data input. Therefore, the data is retained in each flip-flop as the clock pulses arrive. In this way, a digital word can be stored indefinitely.

When *SHL* goes high, D_{in} sets up the right flip-flop, Q_0 sets up the second flip-flop, Q_1 the third flip-flop, and so on. In this mode, the circuit acts like a shift-left register. Each positive clock edge shifts the stored bits one position to the left.

Serial Loading

Serial loading means storing a word in the shift register by entering 1 bit per clock pulse. To store a 4-bit word, we need four clock pulses. For instance, here's how to serially store the word

$$X = 1010$$

With *SHL* high in Fig. 8-7, make $D_{in} = 1$ for the first clock pulse, $D_{in} = 0$ for the second clock pulse, $D_{in} = 1$

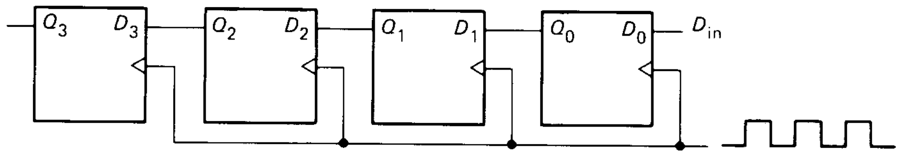


Fig. 8-4 Shift-left register.

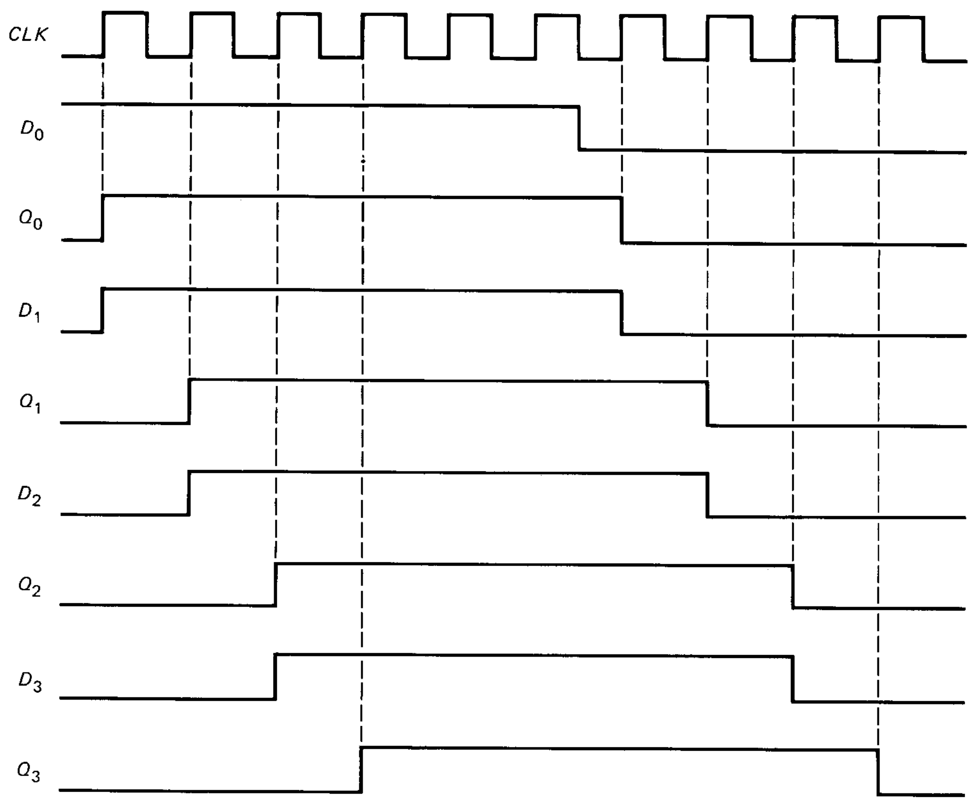


Fig. 8-5 Shift-left timing diagram.

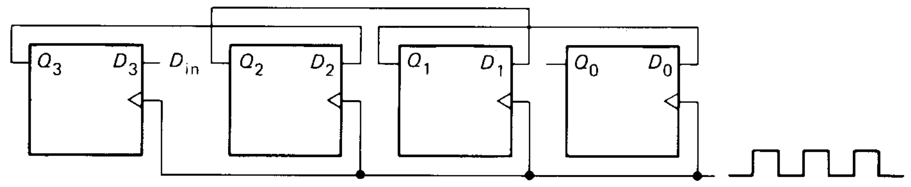


Fig. 8-6 Shift-right register.

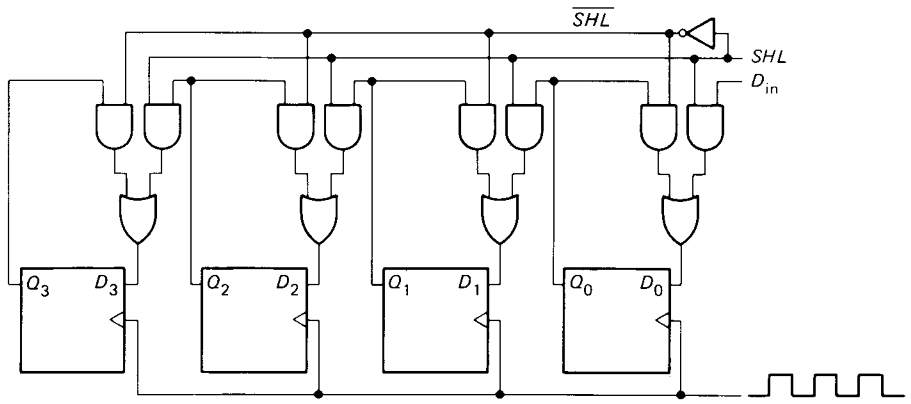


Fig. 8-7 Controlled shift register.

for the third clock pulse, and $D_{in} = 0$ for the fourth clock pulse. If the register is clear before the first clock pulse, the successive register contents look like this:

$$
\begin{array}{ll}
Q = 0001 & (D_{in} = 1\text{: first clock pulse}) \\
Q = 0010 & (D_{in} = 0\text{: second clock pulse}) \\
Q = 0101 & (D_{in} = 1\text{: third clock pulse}) \\
Q = 1010 & (D_{in} = 0\text{: fourth clock pulse})
\end{array}
$$

In this way, data is entered serially into the right end of the register and shifted left until all 4 bits have been stored. After the last bit is entered, *SHL* is taken low to freeze the register contents.

Parallel Loading

Figure 8-8 is another step in the evolution of shift registers. The circuit can load *X* bits directly into the flip-flops, the same as a buffer register. This kind of entry is called *parallel* or *broadside loading;* it takes only one clock pulse to store a digital word.

If *LOAD* and *SHL* are low, the output of the NOR gate is high and flip-flop outputs return to their data inputs. This forces the data to be retained in each flip-flop as the positive clock edges arrive. In other words, the register is inactive when *LOAD* and *SHL* are low, and the contents are stored indefinitely.

When *LOAD* is low and *SHL* is high, the circuit acts like a shift-left register, as previously described. On the other hand, when *LOAD* is high and *SHL* is low, the circuit acts like a buffer register because the *X* bits set up the flip-flops for broadside loading. (Having *LOAD* and *SHL* simultaneously high is forbidden because it's impossible to do both operations on a single clock edge.)

By adding more flip-flops we can build a controlled shift register of any length. And with more gates, the shift-right operation can be included. As an example, the 74198 is a TTL 8-bit bidirectional shift register. It can broadside load, shift left, or shift right.

8-4 RIPPLE COUNTERS

A *counter* is a register capable of counting the number of clock pulses that have arrived at its clock input. In its simplest form it is the electronic equivalent of a binary odometer.

The Circuit

Figure 8-9*a* shows a counter built with *JK* flip-flops. Since the *J* and *K* inputs are returned to a high voltage, each flip-flop will toggle when its clock input receives a negative edge.

Here's how the counter works. Visualize the *Q* outputs as a binary word

$$Q = Q_3Q_2Q_1Q_0$$

Q_3 is the most significant bit (MSB), and Q_0 is the least significant bit (LSB). When *CLR* goes low; all flip-flops reset. This results in a digital word of

$$Q = 0000$$

When *CLR* returns to high, the counter is ready to go. Since the LSB flip-flop receives each clock pulse, Q_0 toggles once per negative clock edge, as shown in the timing diagram of Fig. 8-9*b*. The remaining flip-flops toggle less often because they receive their negative edges from the preceding flip-flops.

For instance, when Q_0 goes from 1 back to 0, the Q_1 flip-flop receives a negative edge and toggles. Likewise,

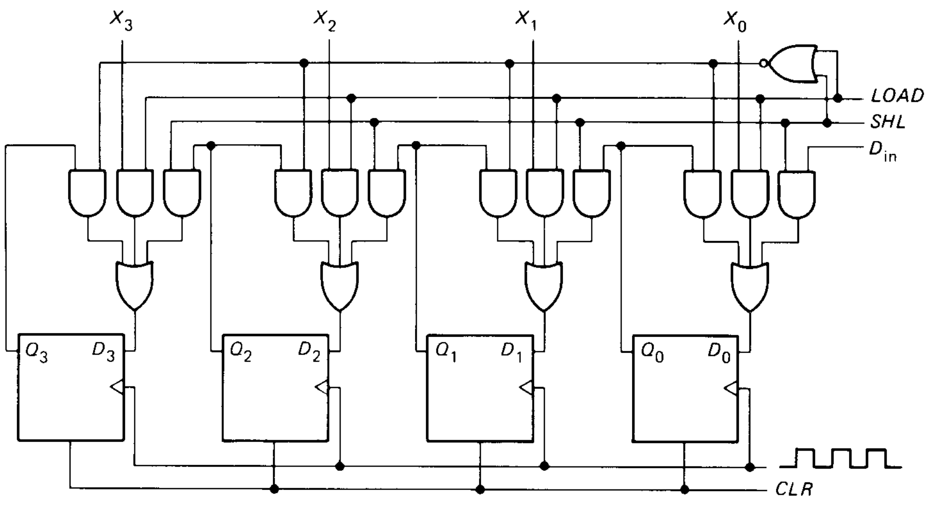
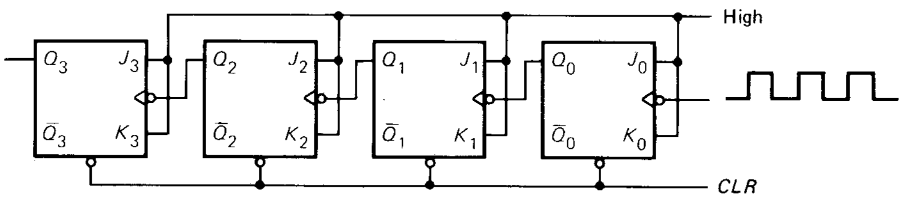


Fig. 8-8 Shift register with broadside load.


High
CLR
Q_3 J_3 $\bar{Q}_3$ K_3
Q_2 J_2 $\bar{Q}_2$ K_2
Q_1 J_1 $\bar{Q}_1$ K_1
Q_0 J_0 $\bar{Q}_0$ K_0
(a)

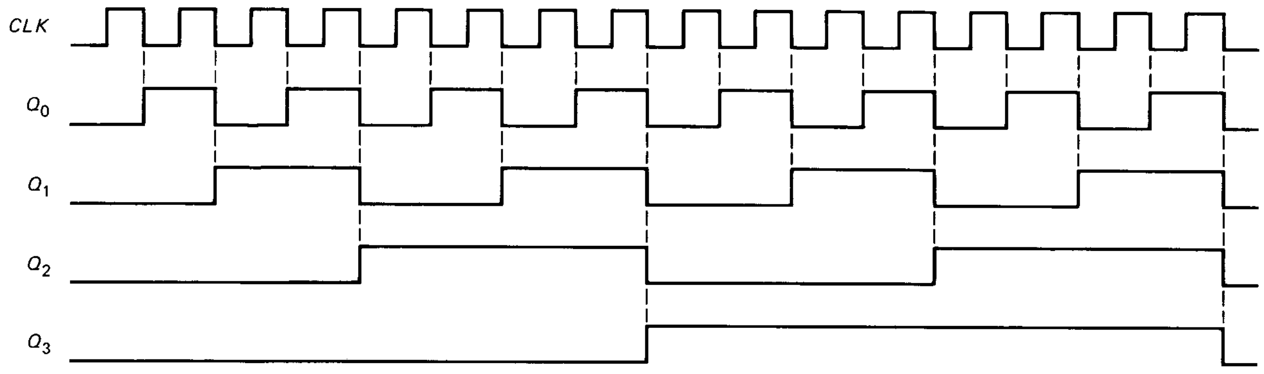


Fig. 8-9 (*a*) Ripple counter; (*b*) timing diagram.

when Q_1 changes from 1 back to 0, the Q_2 flip-flop gets a negative edge and toggles. And when Q_2 goes from 1 to 0, the Q_3 flip-flop toggles. In other words, whenever a flip-flop resets to 0, the next higher flip-flop toggles (see Fig. 8-9*b*).

What does this remind you of? Reset and carry! Each flip-flop acts like a wheel in a binary odometer; whenever it resets to 0, it sends a carry to the next higher flip-flop. Therefore, the counter of Fig. 8-9*a* is the electronic equivalent of a binary odometer.

Counting

If *CLR* goes low then high, the register contents of Fig. 8-9*a* become

$$Q = 0000$$

When the first clock pulse hits the LSB flip-flop, Q_0 becomes a 1. So the first output word is

$$Q = 0001$$

When the second clock pulse arrives, Q_0 resets and carries; therefore, the next output word is

$$Q = 0010$$

The third clock pulse advances Q_0 to 1; this gives

$$Q = 0011$$

The fourth clock pulse forces the Q_0 flip-flop to reset and carry. In turn, the Q_1 flip-flop resets and carries. The resulting output word is

$$Q = 0100$$

The fifth clock pulse gives

$$Q = 0101$$

The sixth gives

$$Q = 0110$$

and the seventh gives

$$Q = 0111$$

On the eighth clock pulse, Q_0 resets and carries, Q_1 resets and carries, Q_2 resets and carries, and Q_3 advances to 1. So the output word becomes

$$Q = 1000$$

The ninth clock pulse gives

$$Q = 1001$$

The tenth gives

$$Q = 1010$$

and so on.

TABLE 8-1. RIPPLE COUNTER

Count	$Q_3Q_2Q_1Q_0$
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
10	1 0 1 0
11	1 0 1 1
12	1 1 0 0
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

The last word is

$$\mathbf{Q} = 1111$$

corresponding to the fifteenth clock pulse. The next clock pulse resets all flip-flops. Therefore, the counter resets to

$$\mathbf{Q} = 0000$$

and the cycle repeats.

Table 8-1 summarizes the operation of the counter. *Count* represents the number of clock pulses that have arrived. As you see, the counter output is the binary equivalent of the decimal count.

Frequency Division

Each flip-flop in Fig. 8-9*a* divides the clock frequency by a factor of 2. This is why a flip-flop is sometimes called a *divide-by-2 circuit*. Since each flip-flop divides the clock frequency by 2, *n* flip-flops divide the clock frequency by 2^n.

The timing diagram of Fig. 8-9*b* illustrates the divide-by-2 action. Q_0 is one-half the clock frequency, Q_1 is one-fourth the clock frequency, Q_2 is one-eighth the clock frequency, and Q_3 is one-sixteenth of the clock frequency. In other words,

1 flip-flop divides by 2
2 flip-flops divide by 4
3 flip-flops divide by 8
4 flip-flops divide by 16

and

n flip-flops divide by 2^n

Ripple Counter

The counter of Fig. 8-9*a* is known as a *ripple counter* because the carry moves through the flip-flops like a ripple on water. In other words, the Q_0 flip-flop must toggle before the Q_1 flip-flop, which in turn must toggle before the Q_2 flip-flop, which in turn must toggle before the Q_3 flip-flop. The worst case occurs when the stored word changes from 0111 to 1000, or from 1111 to 0000. In either case, the carry has to move all the way to the MSB flip-flop. Given a t_p of 10 ns per flip-flop, it takes 40 ns for the MSB to change.

By adding more flip-flops to the left end of Fig. 8-9*a* we can build a ripple counter of any length. Eight flip-flops give an 8-bit ripple counter, twelve flip-flops result in a 12-bit ripple counter, and so on.

Controlled Counter

A controlled counter counts clock pulses only when commanded to do so. Figure 8-10 shows how it's done. The *COUNT* signal can be low or high. Since it conditions the *J* and *K* inputs, *COUNT* controls the action of the counter, forcing it to either do nothing or to count clock pulses.

When *COUNT* is low, the *J* and *K* inputs are low; therefore, all flip-flops remain latched in spite of the clock pulses driving the counter.

On the other hand, when *COUNT* is high, the *J* and *K* inputs are high. In this case, the counter works as previously described; each negative clock edge increments the stored count by 1.

EXAMPLE 8-2

As mentioned earlier, the program and data are stored in the memory before a computer run. The program is a list of instructions telling the computer how to process the data.

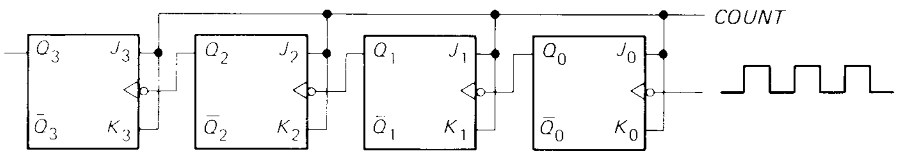


Fig. 8-10 Controlled ripple counter.

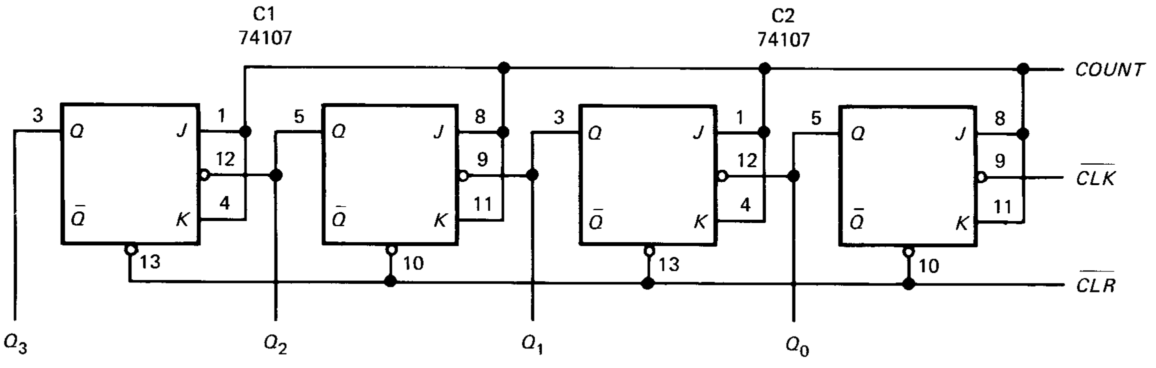


Fig. 8-11 SAP-1 program counter.

Every microcomputer has a *program counter* to keep track of the instruction being executed.

Figure 8-11 shows part of the program counter used in SAP-1. What does it do?

SOLUTION

To begin with, let's find out why the $\overline{CLR}$ and $\overline{CLK}$ signals are shown as complements. Signals are often available in complemented and uncomplemented form. The switch debouncer of Fig. 7-4*a* has two outputs, *CLR* and $\overline{CLR}$. In SAP-1 the *CLR* signal goes to any circuit that uses an active high clear and the $\overline{CLR}$ signal to any circuit with an active low clear. This is why $\overline{CLR}$ goes to the counter of Fig. 8-11; it has an active low clear. A similar idea applies to the clock signal.

The 74107 is a dual *JK* master-slave flip-flop. The SAP-1 program counter uses two 74107s. Although not shown, pin 14 ties to the 5-V supply, and pin 7 is the chip ground. Because master-slave flip-flops are used, a high $\overline{CLK}$ cocks the master and a low $\overline{CLK}$ triggers the slave.

Before a computer run, the operator pushes a clear button that sends a low $\overline{CLR}$ to the program counter. This resets its count to

$$Q = 0000$$

When the operator releases the button, $\overline{CLR}$ goes high and the computer run begins.

After the first instruction has been fetched from the memory, *COUNT* goes high for one clock pulse and the count becomes

$$Q = 0001$$

This count indicates that the first instruction has been fetched from the memory. (Later you will see how the computer executes the first instruction.)

After the first instruction has been executed, the computer fetches the second instruction in the memory. Once again, *COUNT* goes high for one clock pulse, producing a new count of

$$Q = 0010$$

The program counter now indicates that the second instruction has been fetched from the memory.

Each time a new instruction is fetched from the memory, the program counter is incremented to produce the next higher count. In this way, the computer can keep track of which instruction it's working on.

8-5 SYNCHRONOUS COUNTERS

When the carry has to propagate through a chain of *n* flip-flops, the overall propagation delay time is nt_p. For this reason ripple counters are too slow for some applications. To get around the ripple-delay problem, we can use a *synchronous counter*.

The Circuit

Figure 8-12 shows one way to build a synchronous counter with positive-edge-triggered flip-flops. This time, clock pulses drive all flip-flops in parallel. Because of the simultaneous clocking, the correct binary word appears after one propagation delay time rather than four.

The least significant flip-flop has its *J* and *K* inputs tied to a high voltage; therefore, it responds to each positive clock edge. But the remaining flip-flops can respond to the positive clock edge only under certain conditions. As shown in Fig. 8-12, the Q_1 flip-flop toggles on the positive clock edge only when Q_0 is a 1. The Q_2 flip-flop toggles only when Q_1 and Q_0 are 1s. And the Q_3 flip-flop toggles only when Q_2, Q_1, and Q_0 are 1s. In other words, a flip-flop toggles on the next positive clock edge if all lower bits are 1s.

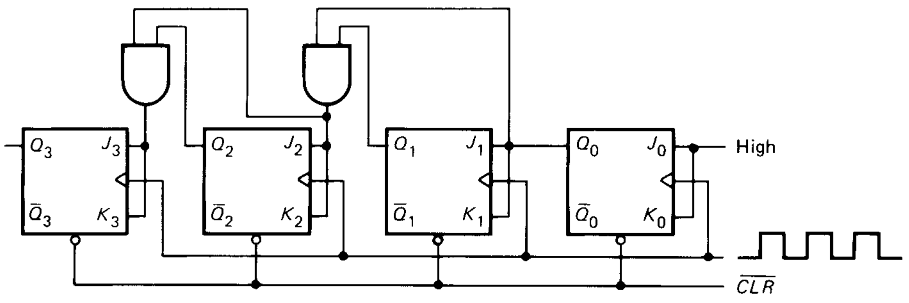


Fig. 8-12 Synchronous counter.

Here's the counting action. A low $\overline{CLR}$ resets the counter to

$$\mathbf{Q} = 0000$$

When the $\overline{CLR}$ line goes high, the counter is ready to go. The first positive clock edge sets Q_0 to get

$$\mathbf{Q} = 0001$$

Since Q_0 is now 1, the Q_1 flip-flop is conditioned to toggle on the next positive clock edge.

When the second positive clock edge arrives, Q_1 and Q_0 simultaneously toggle and the output word becomes

$$\mathbf{Q} = 0010$$

The third positive clock edge advances the count by 1:

$$\mathbf{Q} = 0011$$

Because Q_1 and Q_0 are now 1s, the Q_2, Q_1, and Q_0 flip-flops are conditioned to toggle on the next positive clock edge. When the fourth positive clock edge arrives, Q_2, Q_1, and Q_0 toggle simultaneously, and after one propagation delay time the output word becomes

$$\mathbf{Q} = 0100$$

The successive **Q** words are 0101, 0110, 0111, and so on up to 1111 (equivalent to decimal 15). The next positive clock edge resets the counter, and the cycle repeats.

By adding more flip-flops and gates we can build synchronous counters of any length. The advantage of a synchronous counter is its speed; it takes only one propagation delay time for the correct binary count to appear after the clock edge hits.

Controlled Counter

Figure 8-13 shows how to build a *controlled synchronous counter*. A low *COUNT* disables all flip-flops. When *COUNT* is high, the circuit becomes a synchronous counter; each positive clock edge advances the count by 1.

8-6 RING COUNTERS

Instead of counting with binary numbers, a *ring counter* uses words that have only a single high bit.

Circuit

Figure 8-14 is a *ring counter* built with *D* flip-flops. The Q_0 output sets up the D_1 input, the Q_1 output sets up the D_2 input, and so on. Therefore, a ring counter resembles a

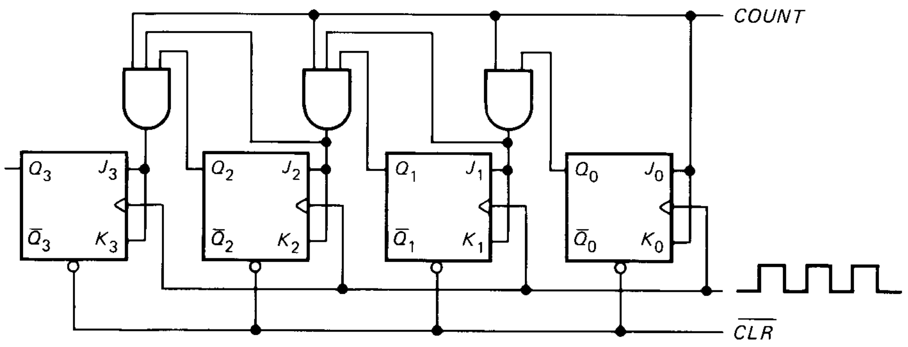


Fig. 8-13 Controlled synchronous counter.

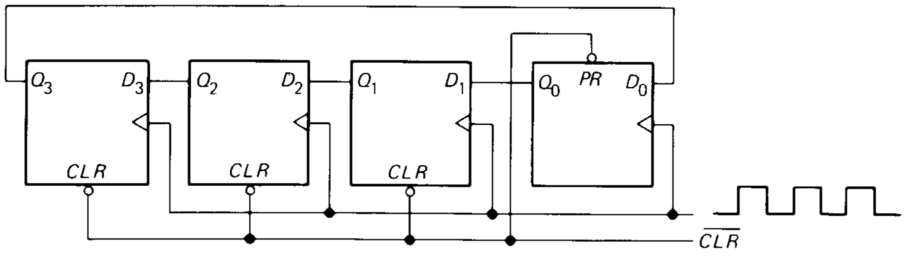


Fig. 8-14 Ring counter.

shift-left register because the bits are shifted left one position per positive clock edge. But the circuit differs because the final output is fed back to the D_0 input. This kind of action is called *rotate left;* bits are shifted left and fed back to the input.

When $\overline{CLR}$ goes low then back to high, the initial output word is

$$\mathbf{Q} = 0001$$

The first positive clock edge shifts the MSB into the LSB position; the other bits shift left one position. Therefore, the output word becomes

$$\mathbf{Q} = 0010$$

The second positive clock edge causes another rotate left and the output word changes to

$$\mathbf{Q} = 0100$$

After the third positive clock edge, the output word is

$$\mathbf{Q} = 1000$$

The fourth positive clock edge starts the cycle over because the rotate left produces

$$\mathbf{Q} = 0001$$

The stored 1 bit follows a circular path, moving left through the flip-flops until the final flip-flop sends it back to the first flip-flop. This is why the circuit is called a ring counter.

More Bits

Add more flip-flops and you can build a ring counter of any length. With six flip-flops we get a 6-bit ring counter. Again, the $\overline{CLR}$ signal resets all flip-flops except the LSB flip-flop. Therefore, the successive ring words are

$$\begin{aligned} \mathbf{Q} &= 000001 \quad (0)\\ \mathbf{Q} &= 000010 \quad (1)\\ \mathbf{Q} &= 000100 \quad (2)\\ \mathbf{Q} &= 001000 \quad (3)\\ \mathbf{Q} &= 010000 \quad (4)\\ \mathbf{Q} &= 100000 \quad (5) \end{aligned}$$

Each of the foregoing words has only 1 high bit. The initial word stands for decimal 0 and the final word for decimal 5. If a ring counter has n flip-flops, therefore, the final ring word represents decimal $n - 1$.

Applications

Ring counters cannot compete with ripple and synchronous counters when it comes to ordinary counting, but they are invaluable when it's necessary to control a sequence of operations. Because each ring word has only 1 high bit, you can activate one of several devices.

For instance, suppose the six small boxes (A to F) of Fig. 8-15 are digital circuits that can be turned on by a high Q bit. When $\overline{CLR}$ goes low, Q_0 goes high and activates device A. After $\overline{CLR}$ returns to high, successive clock pulses turn on each device for a short time. In other words, as the stored 1 bit shifts left, it turns on B to F in sequence, and then the cycle starts over.

Many digital circuits participate during a computer run. To fetch and execute instructions, a computer has to activate

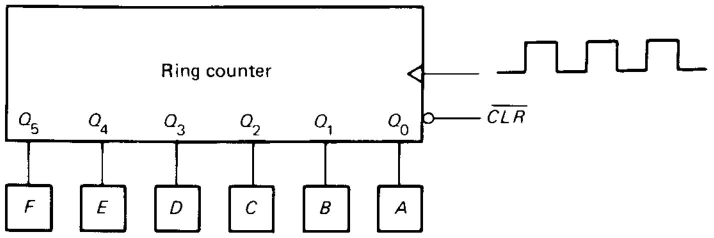
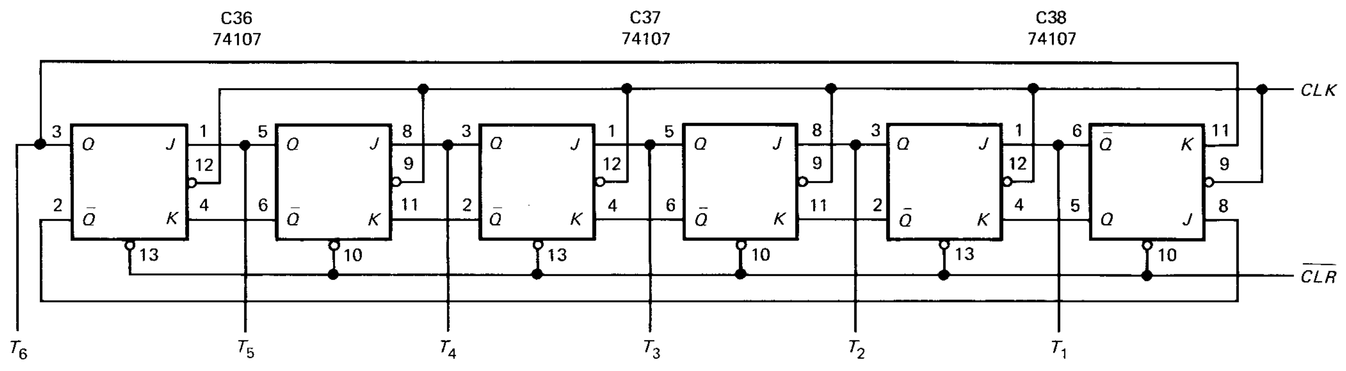


Fig. 8-15 Controlling a sequence of operations



Note: Pin 14 is connected to +5 V, and pin 7 is grounded.

Fig. 8-16 SAP-1 ring counter.

these circuits at precisely the right time and in the right sequence. This is where ring counters shine; they produce the ring words for timing different operations during a computer run.

EXAMPLE 8-3

Figure 8-16 shows the ring counter used in the SAP-1 computer. T_6 to T_1 are called *timing* signals because they control a sequence of digital operations. What does this ring counter do?

SOLUTION

The 74107 is a dual *JK* master-slave flip-flop, previously used in the SAP-1 program counter (Example 8-2). The flip-flops are connected in a rotate-left mode. Since the 74107 does not have a preset input, the Q_0 flip-flop is inverted so that its $\overline{Q}$ output drives the *J* input of the Q_1 flip-flop. In this way, a low $\overline{CLR}$ produces the initial timing word

$$T_6T_5T_4T_3T_2T_1 = 000001$$

In chunked form

$$T = 000001$$

Because of the master-slave action, a complete clock pulse is needed to produce the next ring word. After $\overline{CLR}$ returns high, the successive clock pulses produce the timing words

$$T = 000010$$
$$T = 000100$$
$$T = 001000$$
$$T = 010000$$
$$T = 100000$$

Then the cycle repeats.

EXAMPLE 8-4

The clock frequency in Fig. 8-16 is 1 kHz. $\overline{CLR}$ goes low then high. Show the timing diagram.

SOLUTION

Figure 8-17 is the timing diagram. Since the clock has a frequency of 1 kHz, it has a period of 1 ms. This is the amount of time between successive negative clock edges. Each negative clock edge produces the next ring word. When its turn comes, each timing signal goes high for 1 ms.

Notice that the *CLK* signal of Fig. 8-17 is the input to the ring counter of Fig. 8-16, whereas the complement $\overline{CLK}$ is the input to the program counter of Fig. 8-11. This half-cycle difference is deliberate. The reason is given in Chap. 10, which explains how the timing signals of Fig. 8-17 control circuits that fetch and execute each program instruction.

8-7 OTHER COUNTERS

The *modulus* of a counter is the number of output states it has. A 4-bit ripple counter has a modulus of 16 because it has 16 distinct states numbered from 0000 to 1111. By changing the design we can produce a counter with any desired modulus.

Mod-10 Counter

Figure 8-18*a* shows a way to build a modulus-10 (or mod-10) counter. The circuit counts from 0000 to 1001, as before. However, on the tenth clock pulse, the counter

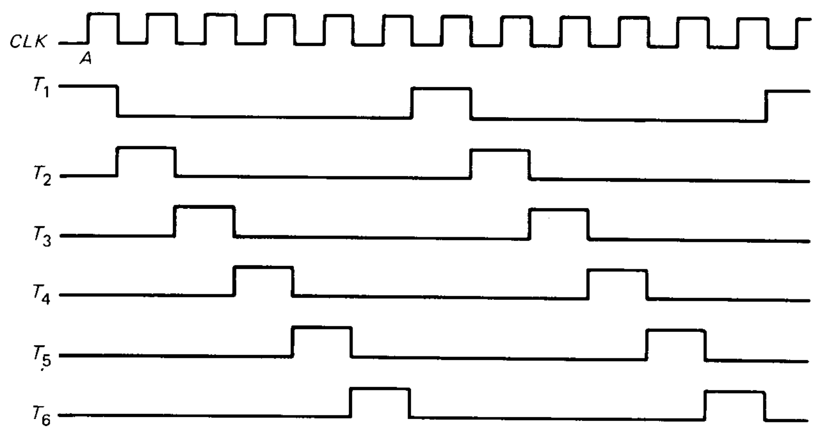


Fig. 8-17 SAP-1 clock and timing pulses.

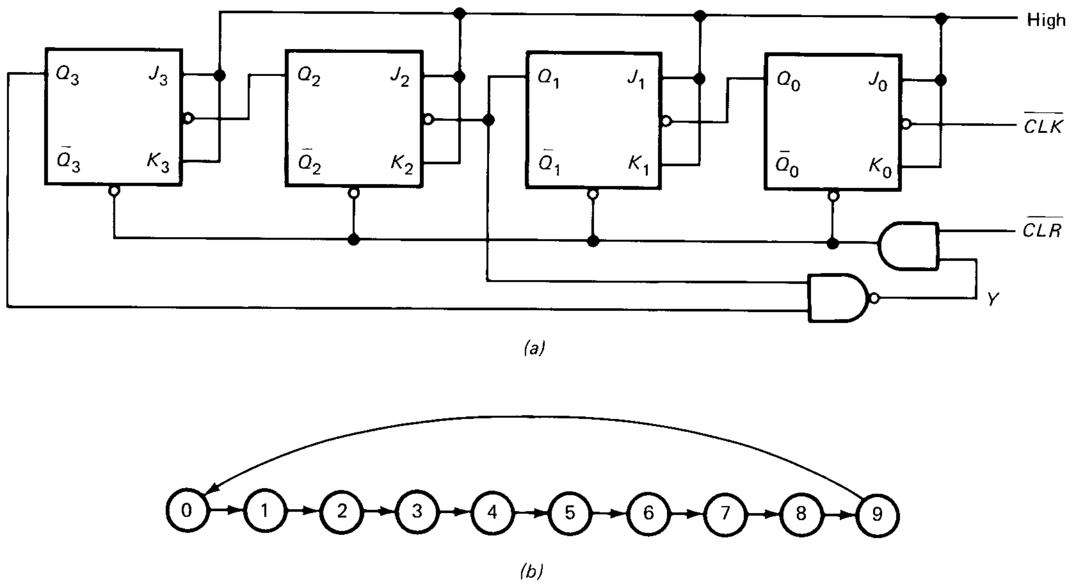


Fig. 8-18 Mod-10 counter.

generates its own clear signal and the count jumps back to 0000. In other words, the count sequence is

Q = 0000	(0)
Q = 0001	(1)
Q = 0010	(2)
Q = 0011	(3)
Q = 0100	(4)
Q = 0101	(5)
Q = 0110	(6)
Q = 0111	(7)
Q = 1000	(8)
Q = 1001	(9)
Q = 0000	(0)

As you see, the circuit skips states 10 to 15 (1010 through 1111). The counting sequence is summarized by the *state diagram* of Fig. 8-18*b*.

Why does the counter skip the states from 10 to 15? Because of the AND gate, the counter can be reset by a low $\overline{CLR}$ or a low Y. Initially, $\overline{CLR}$ goes low to produce

$$\mathbf{Q} = 0000$$

When $\overline{CLR}$ returns to high, the counter is ready for action.

The output of the NAND gate is

$$Y = \overline{Q_3 Q_1}$$

This output is high for the first nine states (0000 to 1001). Nothing unusual happens when the circuit is counting from 0 to 9. On the tenth clock pulse, however, the **Q** word becomes

$$\mathbf{Q} = 1010$$

which means that Q_3 and Q_1 are high. Almost immediately, Y goes low, forcing the counter to reset to

$$Q = 0000$$

Y then goes high, and the counter is ready to start over.

Since it takes 10 clock pulses to reset the counter, the output frequency of the Q_3 flip-flop is one-tenth of the clock frequency. This is why a mod-10 counter is also known as a *divide-by-10 circuit.*

A mod-10 counter like Fig. 8-18*a* is often called a *decade counter*. Because it counts from 0 to 9, it is a natural choice in BCD applications like frequency counters, digital voltmeters, and electronic wristwatches.

To get any other modulus, we can use the same basic idea. For instance, to get a mod-12 counter, we can drive the NAND gate of Fig. 8-18*a* with Q_3 and Q_2. Then the circuit counts from 0 to 11 (0000 to 1011). On the next clock pulse, Q_3 and Q_2 are high, which clears the counter. (What is the modulus if Q_3 and Q_0 drive the NAND gate?)

Down Counter

All the counters discussed so far have counted upward, toward higher numbers. Figure 8-19 shows a *down counter;* it counts from 1111 to 0000. Each flip-flop toggles when its clock input goes from 1 to 0. This is equivalent to an uncomplemented output going from 0 to 1. For instance, the Q_1 flip-flop toggles when $\overline{Q}_0$ goes from 1 to 0; this is equivalent to Q_0 going from 0 to 1.

A preset signal generated elsewhere is available in either uncomplemented or complemented form; *PRE* goes to all circuits with an active-high preset; $\overline{PRE}$ goes to all circuits with an active-low preset. Initially, the preset signal $\overline{PRE}$ goes low in Fig. 8-19, producing an output word of

$$Q = 1111 \qquad (15)$$

When $\overline{PRE}$ goes high, the action starts. Notice that Q_0 toggles once per clock pulse. In the following discussion, a *positive toggle* means a change from 0 to 1, a *negative toggle* means a change from 1 to 0.

The first clock pulse produces a negative toggle in Q_0; nothing else happens:

$$Q = 1110 \qquad (14)$$

The second clock pulse produces a positive toggle in Q_0, which produces a negative toggle in Q_1:

$$Q = 1101 \qquad (13)$$

On the third clock pulse, Q_0 toggles negatively, and

$$Q = 1100 \qquad (12)$$

On the fourth clock pulse, Q_0 toggles positively, Q_1 toggles positively, and Q_2 toggles negatively:

$$Q = 1011 \qquad (11)$$

You should have the idea by now. The circuit is counting down, from 15 to 0. When it reaches 0,

$$Q = 0000$$

On the next clock pulse, all flip-flops toggle positively to get

$$Q = 1111$$

and the cycle repeats.

Up-Down Counter

Figure 8-20 shows how to build an *up-down counter*. The flip-flop outputs are connected to *steering* networks. An *UP* control signal produces either down counting or up counting. If the *UP* signal is low, $\overline{Q}_2$, $\overline{Q}_1$, and $\overline{Q}_0$ are transmitted to the clock inputs; this results in a down counter. On the other hand, when *UP* is high, Q_2, Q_1, and Q_0 drive the clock inputs and the circuit becomes an up counter.

Presettable Counter

In a *presettable counter,* the count starts at a number greater than zero. Figure 8-21*a* shows a presettable counter; the count begins with $P_3P_2P_1P_0$, a number between 0000 and 1111.

To start the analysis, look at the *LOAD* control line. When it is low, all NAND gates have high outputs; therefore,

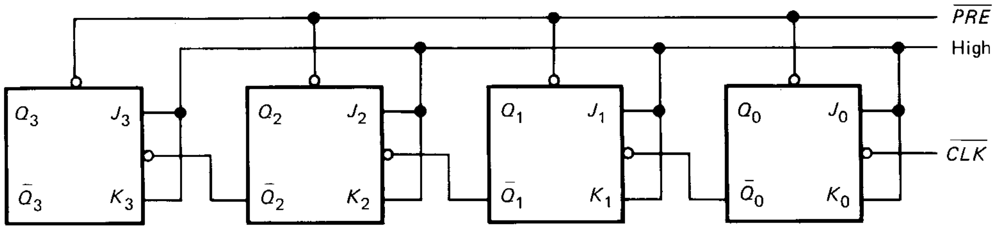


Fig. 8-19 Down counter.

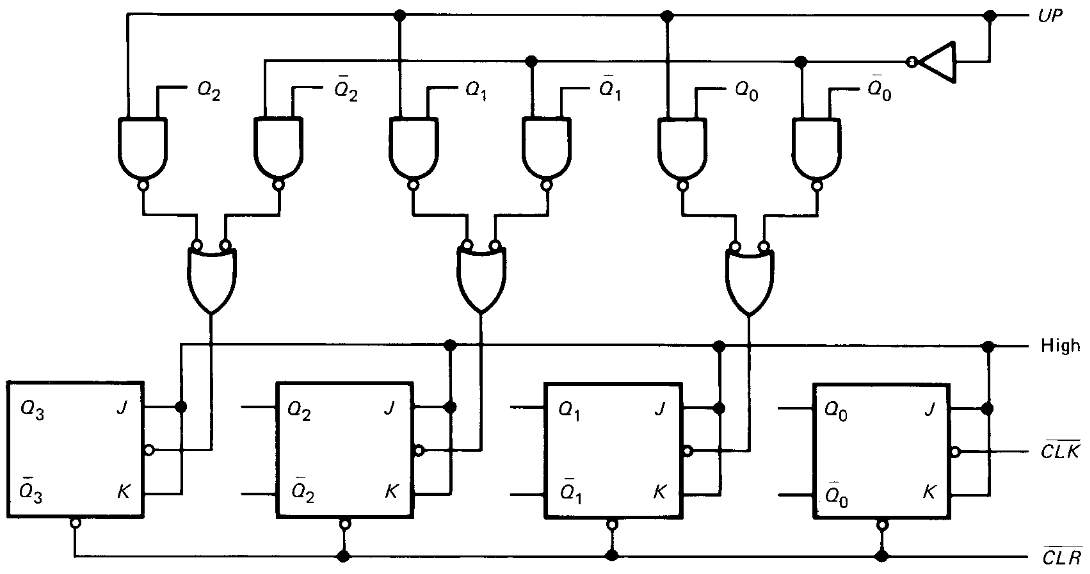


Fig. 8-20 Up-down counter.

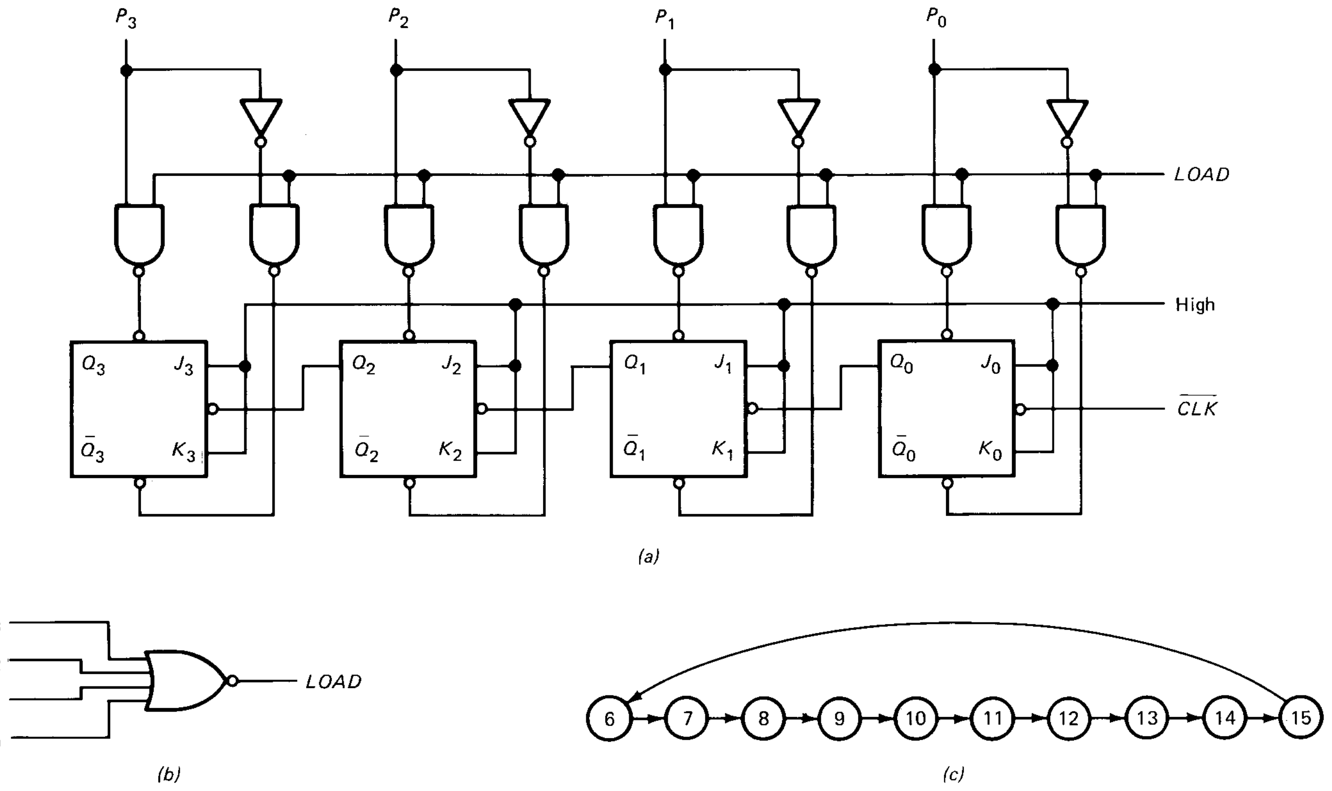


Fig. 8-21 Presettable counter.

the preset and clear inputs of all flip-flops are inactive. In this case, the circuit counts upward, as previously described. The data inputs P_3 to P_0 have no effect because the NAND gates are disabled.

When the *LOAD* line is high, the data inputs and their complements pass through the NAND gates and preset the counter to $P_3P_2P_1P_0$. As an example, suppose the preset input is

$$P_3P_2P_1P_0 = 0110$$

Because of the two left NAND gates, the low P_3 produces a high preset and a low clear for the Q_3 flip-flop; this clears

Q_3 to a 0. By a similar argument, the high P_2 sets Q_2, the high P_1 sets Q_1, and the low P_0 clears Q_0. Therefore, the counter is preset to

$$\mathbf{Q} = 0110$$

When *LOAD* returns to low, the circuit reverts to a counter. Successive clock pulses produce

$$\mathbf{Q} = 0111$$
$$\mathbf{Q} = 1000$$
$$\mathbf{Q} = 1001$$

up to a maximum count of

$$\mathbf{Q} = 1111$$

The next clock pulse resets the counter to

$$\mathbf{Q} = 0000$$

In summary,

1. When *LOAD* is low, the circuit counts.
2. When *LOAD* is high, the counter presets to $P_3P_2P_1P_0$.

Programmable Modulus

The most important use of a presettable counter is *programming a modulus.* Here's the idea. Let's add the NOR gate of Fig. 8-21*b* to the presettable counter of Fig. 8-21*a*. Then the *Q* outputs drive the NOR gate, and the NOR gate controls the *LOAD* line of the presettable counter. Because a NOR gate recognizes a word with all 0s and disregards all others, *LOAD* is high for **Q** = 0000 and low for all other words. This means that the circuit presets when **Q** = 0000 and counts when **Q** is 0001 to 1111.

If the preset input is 0110, successive clock pulses produce 0111, 1000, 1001, . . . , reaching a maximum value of

$$\mathbf{Q} = 1111$$

The next clock pulse resets the count to

$$\mathbf{Q} = 0000$$

Almost immediately, however, the NOR-gate outputs goes high, and the data inputs preset the counter to

$$\mathbf{Q} = 0110$$

In other words, the counter effectively skips states 0 to 5, illustrated by the state diagram of Fig. 8-21*c*.

Figure 8-21*c* shows 10 distinct states; by presetting 0110, we have programmed the counter to become a mod-10 counter. If we change the preset input, we get a different modulus. In general,

$$M = N - P \tag{8-1}$$

where M = modulus of preset counter
N = natural modulus
P = preset count

The natural modulus equals 2^n where n is the number of flip-flops in the counter. So four flip-flops give a natural modulus of 16, eight give a natural modulus of 256, and so on.

As an example, if you preset 82 into a preset counter with eight flip-flops, the modulus is

$$M = 256 - 82 = 174$$

In other words, this preset counter is equivalent to a divide-by-174 circuit.

TTL Counters

Table 8-2 lists some TTL counters. The 7490 is an industry standard, a widely used decade counter. This ripple counter has two sections, a divide-by-2 and a divide-by-5. This allows you to divide by 2, to divide by 5, or to cascade both sections to divide by 10.

The 7492 is a mod-12 ripple counter, organized in two sections by divide-by-2 and divide-by-6. This allows you to divide by 2, divide by 6, or cascade to divide by 12. The 7493 is a mod-16 ripple counter, with two sections of divide-by-2 and divide-by-8.

The 74160 and 74161 are presettable synchronous counters, the first being a decade counter and the second a divide-by-16 counter. Finally, the 74190 and 74191 are up-down presettable counters.

This is a sample of basic TTL counters; others are listed in Appendix 3.

TABLE 8-2. TTL COUNTERS

Number	Type
7490	Decade
7492	Divide-by-12
7493	Divide-by-16
74160	Presettable decade
74161	Presettable divide-by-16
74190	Up-down presettable decade
74191	Up-down presettable divide-by-16

8-8 THREE-STATE REGISTERS

The *three-state switch,* a development of the early 1970s, has greatly simplified computer wiring and design because it's ideal for *bus-organized computers* (the common type nowadays).

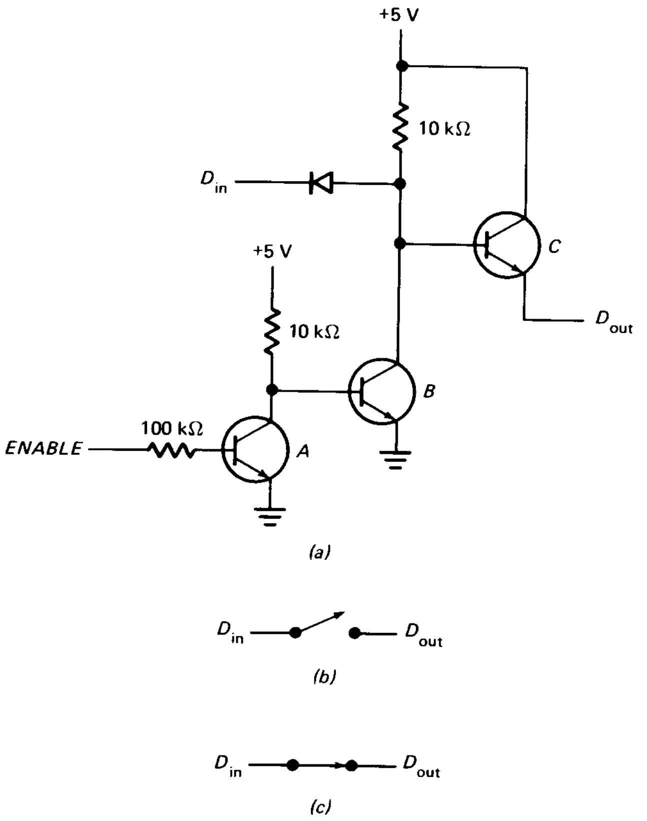


Fig. 8-22 (*a*) Three-state switch; (*b*) floating or high-impedance state; (*c*) output equals input.

Three-State Switch

Figure 8-22*a* is an example of a three-state switch. The *ENABLE* input can be low or high. When it's low, transistor A cuts off and transistor B saturates. This pulls the base of transistor C down to ground, opening its base-emitter diode. As a result, D_{out} floats. This floating state is equivalent to an *open* switch (Fig. 8-22*b*).

On the other hand, when *ENABLE* is high, transistor A saturates and transistor B cuts off. Now, the transistor C acts like an emitter follower, and the overall circuit is equivalent to a *closed* switch (Fig. 8-22*c*). In this case,

$$D_{out} = D_{in}$$

This means that D_{out} is low or high, the same as D_{in}.

Table 8-3 summarizes the action. When *ENABLE* is low, D_{in} is a don't care and D_{out} is open or floating. When *ENABLE* is high, the circuit acts like a noninverting buffer because D_{out} equals D_{in}.

TABLE 8-3. NORMALLY OPEN

ENABLE	D_{in}	D_{out}
0	X	Open
1	0	0
1	1	1

Commercial three-state switches are much more complicated than Fig. 8-22*a* (a totem-pole output and other enhancements are added). But simple as it is, Fig. 8-22*a* captures the key idea of a three-state switch; the output can be in any of three states: low, high, or floating (sometimes called the *high-impedance state* because the Thevenin impedance is high).

Three-state switches are also known as *Tri-state switches.* (Tri-state is a trademark name used by National Semiconductor, the originator of three-state TTL logic.)

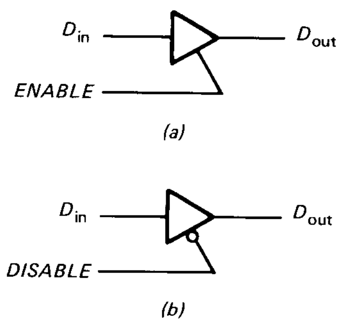


Fig. 8-23 (*a*) Normally open switch; (*b*) normally closed switch.

Normally Open Switch

Figure 8-23*a* is the symbol for a three-state noninverting buffer. When you see this symbol, remember the action: a low *ENABLE* means that the output is floating; a high *ENABLE* means that the output is 0 or 1, the same as the input. Think of this switch as *normally open;* to close it, you have to apply a high *ENABLE.*

In the 7400 series, the 74126 is a quad three-state normally open switch. This means four switches like Fig. 8-23*a* in one package. The SAP-1 computer uses five 74126s.

Normally Closed Switch

Figure 8-23*b* is different. This is the symbol for a *normally closed switch* because the control input *DISABLE* is active low. In other words, the switch is closed when *DISABLE* is low, and open when *DISABLE* is high. Table 8-4 summarizes the operation.

The 74125 is a quad three-state normally closed switch (four switches like Fig. 8-23*b* in one package).

TABLE 8-4. NORMALLY CLOSED

DISABLE	D_{in}	D_{out}
0	0	0
0	1	1
1	X	Open

Three-State Buffer Register

The main application of three-state switches is to convert the two-state output of a register to a three-state output. For instance, Fig. 8-24 shows a three-state buffer register, so called because of the three-state switches on the output lines. When *ENABLE* is low, the *Y* outputs float. But when *ENABLE* is high, the *Y* outputs equal the *Q* outputs; therefore,

$$Y = Q$$

You already know how the rest of the circuit works; it's the controlled buffer register discussed earlier. When *LOAD* is low, the contents of the register are unchanged. When *LOAD* is high, the next positive clock edge loads $X_3X_2X_1X_0$ into the register.

8-9 BUS-ORGANIZED COMPUTERS

A *bus* is a group of wires that transmit a binary word. In Fig. 8-25, vertical wires W_3, W_2, W_1, and W_0 are a bus; these wires are a common transmission path between the three-state registers. The input data bits for register A come from the **W** bus; at the same time, the three-state output of register A connects back to the **W** bus. Similarly, the other registers have their inputs and outputs connected to the **W** bus.

In Fig. 8-25 all control signals are in uncomplemented form; this means that the registers have active high inputs. In other words, a load input (L_A to L_D) must be high to set up for loading, and an enable signal (E_A to E_D) must be high to connect an output to the bus.

Register Transfers

The beauty of bus organization is the ease of transferring a word from one register to another. To begin with, the same clock signal drives all registers, but nothing happens until you apply high control inputs. In other words, as long as all *LOAD* and *ENABLE* inputs are low, the registers are isolated from the bus.

To transfer a word from one register to another, make the appropriate control inputs high. For instance, here's how to transfer the contents of register A to the register D. Make E_A and L_D high; then the contents of register A appear on the bus and register D is set up for loading. When the next positive clock edge arrives, word **A** is stored in register D.

Here is another example. Suppose the following words are stored in the registers:

A = 0011
B = 0110
C = 1001
D = 1100

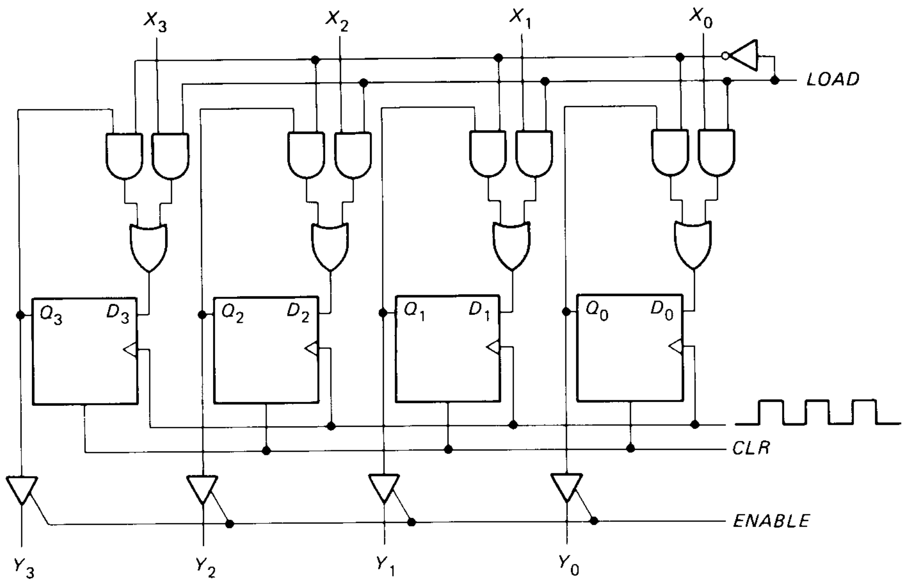


Fig. 8-24 Three-state buffer register.

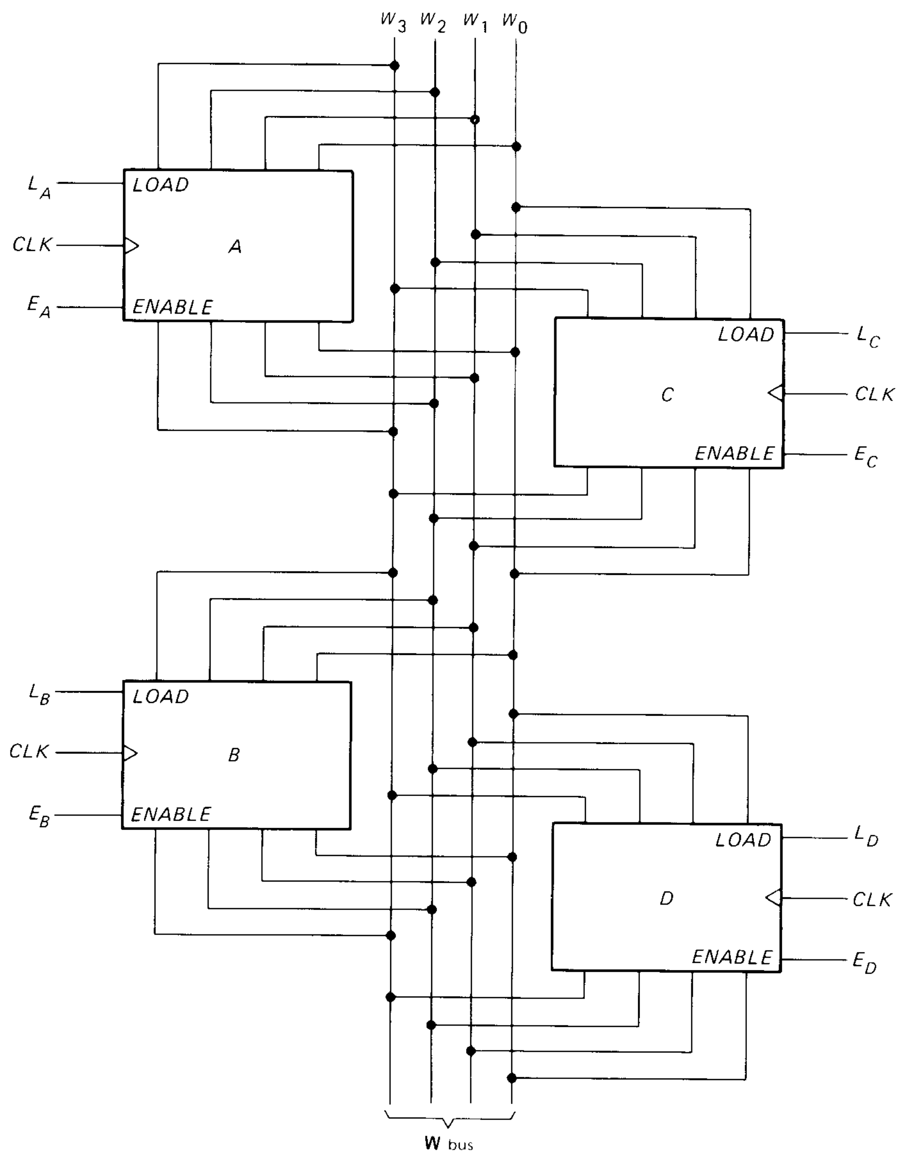


Fig. 8-25 Registers connected to bus.

To transfer word **C** into register B, make E_C and L_B high. The high E_C closes the three-state switches of register C, placing word **C** on the bus. The high L_B sets up register B for loading. When the next positive clock edge arrives, word **C** is stored in register B, and the new words are

$$\mathbf{A} = 0011$$
$$\mathbf{B} = 1001$$
$$\mathbf{C} = 1001$$
$$\mathbf{D} = 1100$$

The whole point of bus organization (connecting the registers to a common word path) is to simplify the wiring and operation of computers. As you will see in Chap. 10, SAP-1 is a bus-organized computer of incredible simplicity made possible by the three-state switch.

Simplified Drawings

Figure 8-25 shows a 4-bit bus. The same idea applies to any number of bits. For example, a 16-bit bus has 16 wires, each carrying 1 bit of a word. By connecting the inputs and outputs of 16-bit registers to this bus, we can transfer 16-bit words from one register to another.

Drawings get very messy unless we simplify the appearance of the bus. Figure 8-26 shows an abbreviated form of Fig. 8-25. The solid arrows represents words going into and out of registers. The solid bar represents the W bus.

EXAMPLE 8-5

Figure 8-27 shows part of the SAP-1 computer. Describe the circuitry.

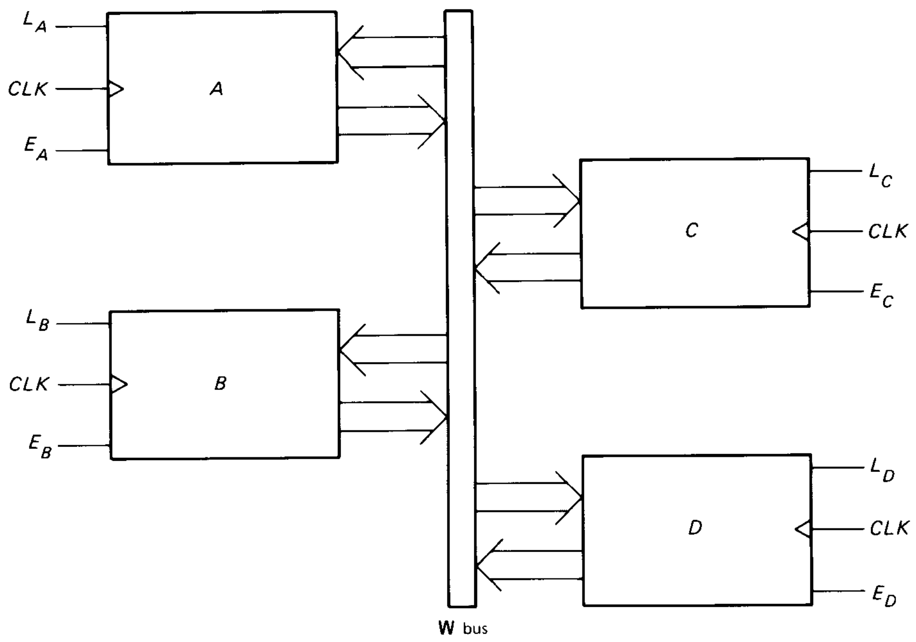


Fig. 8-26 Simplified bus diagram.

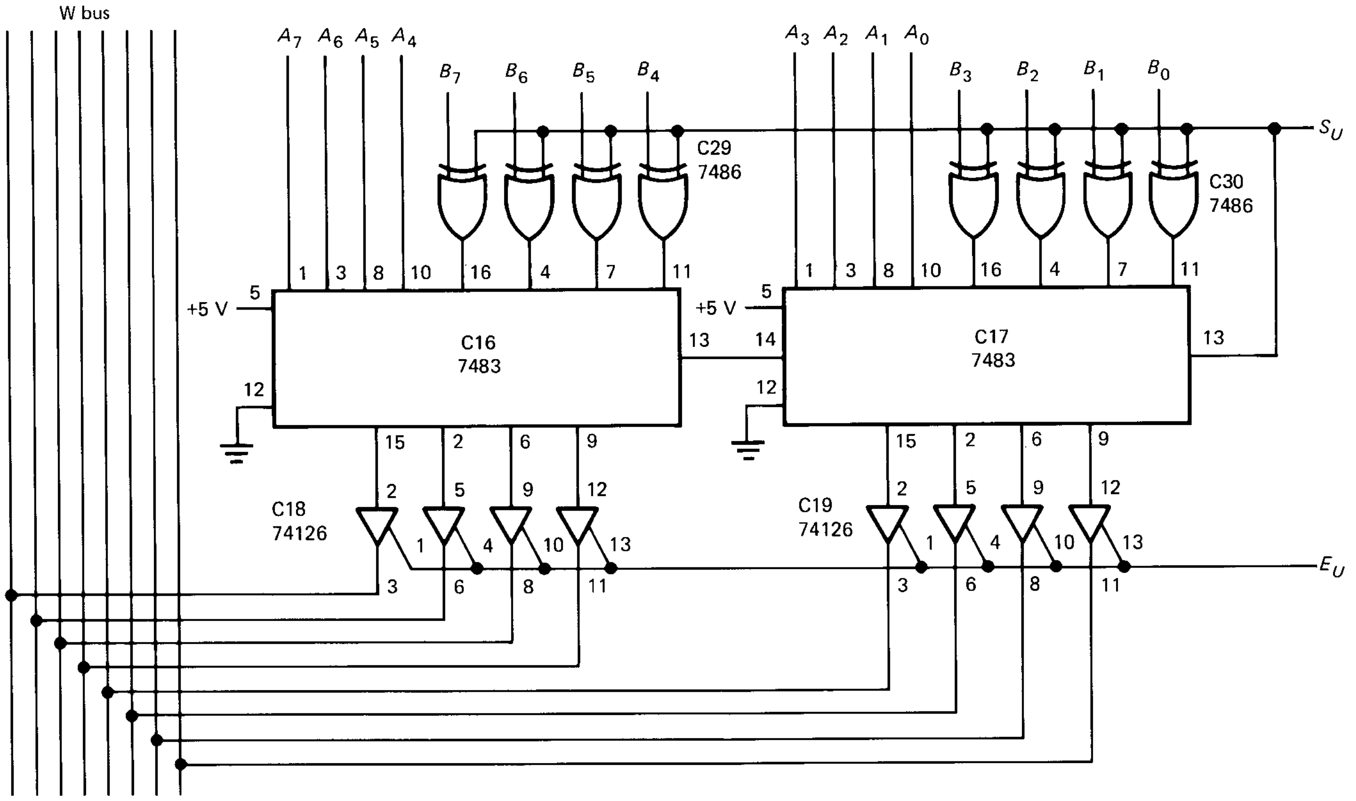


Fig. 8-27 SAP-1 ALU connected to bus.

SOLUTION

As discussed in Sec. 6-8, the 7483 is a 4-bit adder. The two 7483s of Fig. 8-27 are the ALU of the SAP-1 computer. The inputs to this ALU are the words

$$\mathbf{A} = A_7A_6A_5A_4A_3A_2A_1A_0$$
$$\mathbf{B} = B_7B_6B_5B_4B_3B_2B_1B_0$$

A pair of 7486s allow us to complement the **B** input for subtraction.

The sum (S_U low) or difference (S_U high) appears at the output (pins 15, 2, 6, 9 of C16 and pins 15, 2, 6, 9 of C17). Three-state switches (C18 and C19) connect the ALU output to the W bus when E_U is high. If E_U is low, the 74126s are open and the ALU output is isolated from the bus.

EXAMPLE 8-6

Figure 8-28 shows the *instruction register* (C8 and C9) of the SAP-1 computer. What does this 8-bit register do?

SOLUTION

Example 8-1 introduced the 74LS173. As you may recall, pins 9 and 10 are tied together and control the *LOAD* function. Because of the bubble, a low $\overline{L}_I$ is needed to set up the registers for loading. When $\overline{L}_I$ is low, the next positive clock edge loads the data on the bus into the instruction register.

The output of the instruction register is split; the upper nibble $I_7I_6I_5I_4$ goes to the *instruction decoder,* a circuit that will be discussed in Chap. 10. The lower nibble out of the instruction register goes back to the W bus.

The 74LS173 is a 4-bit three-state buffer register; it has internal three-state switches controlled by pins 1 and 2. The bubbles on pins 1 and 2 indicate active-low inputs; therefore, the output of C9 is connected to the bus when $\overline{E}_I$ is low and disconnected when $\overline{E}_I$ is high.

Notice that pins 1 and 2 of C8 are grounded; this means that the upper nibble is always a two-state output. In other words, the 74LS173 can be used as an ordinary two-state register by grounding pins 1 and 2. (This was done in Example 8-1, where we used two 74LS173s for the output register to drive an 8-bit LED display.)

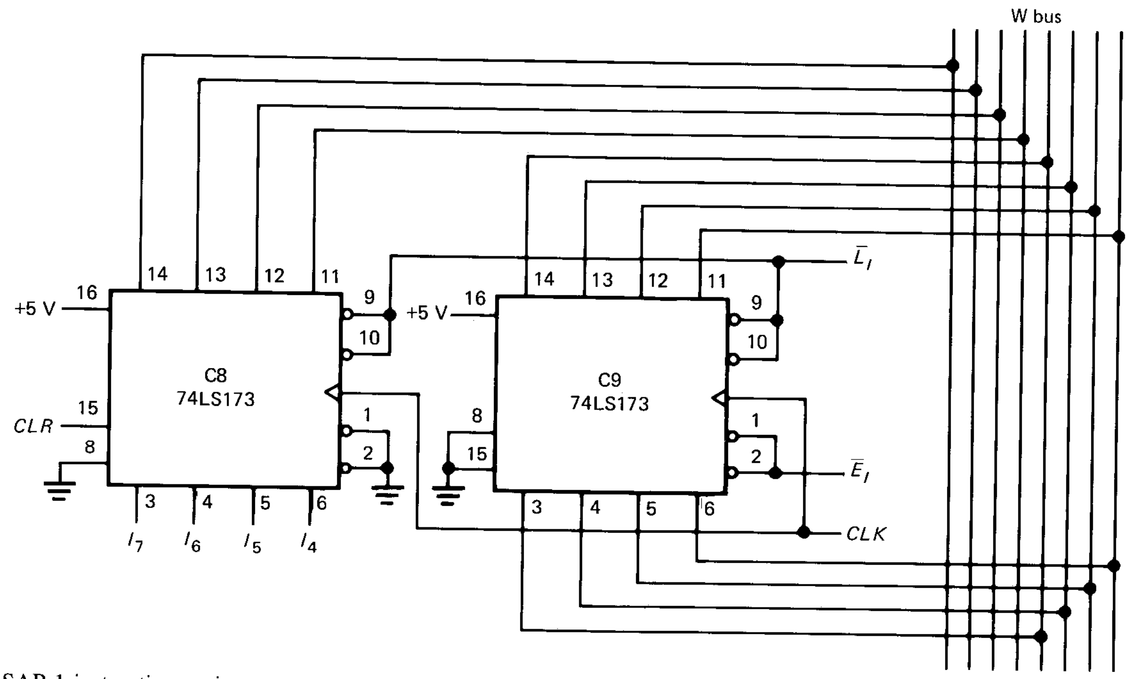


Fig. 8-28 SAP-1 instruction register.

GLOSSARY

buffer register A register that temporarily stores a word during data processing.

bus A group of wires used as a common word path by several registers.

modulus The number of stable states a counter has.

parallel entry Loading all bits of a word in parallel during one clock pulse. Also called broadside loading.

presettable counter A counter that allows you to preset a

number from which the count begins. Sometimes called a programmable counter.
register A group of memory elements that store a word.
ring counter A counter producing words with 1 high bit, which shifts one position per clock pulse.
ripple counter A counter with cascaded flip-flops. This means that the carry has to propagate in series through the flip-flops.
serial entry Loading a word into a shift register 1 bit per clock pulse
shift register A register that can shift the stored bits one position to the left or right.
synchronous counter A counter in which the clock drives each flip-flop to eliminate the ripple delay.
three-state switch A noninverting buffer that can be closed or opened by a control signal. Also called a Tri-state switch.

SELF-TESTING REVIEW

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

1. When the *LOAD* input of a buffer register is active, the input word is stored on the next positive ________ edge. If *LOAD* then becomes inactive, the input word can change without effecting the ________ word.
2. (*clock, stored*) A shift register moves the ________ left or right. Serial loading means storing a word in a shift register by entering ________ bit per clock pulse. With parallel or broadside loading, it takes only one ________ pulse to load the input word.
3. (*bits, 1, clock*) One flip-flop divides the clock frequency by a factor of ________. Two flip-flops divide by 4, three flip-flops by 8, and four flip-flops by ________. In general, n flip-flops divide by 2^n.
4. (*2, 16*) In a ripple counter, the carry has to propagate through all the flip-flops to reach the MSB flip-flop. The overall propagation delay time is ________. A controlled counter counts ________ pulses only when the *COUNT* signal is active. The clock signal drives each flip-flop of a ________ counter.
5. (*nt_p, clock, synchronous*) Instead of counting with binary numbers, a ring counter uses words that have a single high ________. A ring counter is ideal for timing a sequence of digital operations.
6. (*bit*) The modulus of a counter is the number of stable output ________ it has. A mod-10 counter can divide the clock frequency by a factor of ________.
7. (*states, 10*) An up-down counter can count up or down. A presettable counter starts the count from a ________ number. This allows us to program the ________. If the modulus is M, a presettable counter is equivalent to a divide-by-M circuit.
8. (*preset, modulus*) A three-state switch has an output that is either low, high, or ________. Two types are available; normally open and normally closed. The main use of three-state switches is to convert the ________ output of a register to a three-state output.
9. (*floating, two-state*) A bus is a group of wires used by three-state registers as a common word path. Bus-organized computers, the common type nowadays, have several registers connected to one or more buses. Instructions and data travel along these buses as they move from one register to another.

PROBLEMS

8-1. Figure 8-29 shows an output register. Before time *A* the data word to be loaded is

$$\mathbf{X} = 1000\ 1101$$

and the LED display is

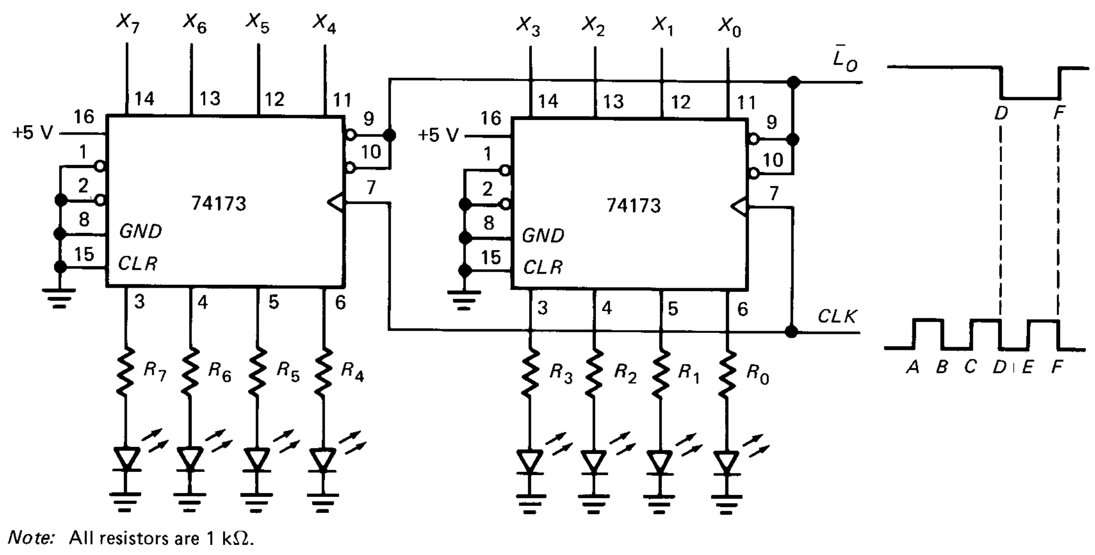
$$\mathbf{Q} = 0001\ 0111$$

a. What is the LED display at time *D*?
b. What is the LED display at time *F*?

8-2. The data sheet of a 74173 gives these values:

$$t_{\text{setup}} = 17 \text{ ns} \quad (L_O \text{ input})$$
$$t_{\text{setup}} = 10 \text{ ns} \quad (\text{Data})$$
$$t_{\text{hold}} = 2 \text{ ns} \quad (L_O \text{ input})$$
$$t_{\text{hold}} = 10 \text{ ns} \quad (\text{Data})$$

a. In Fig. 8-29, how far ahead of point *E* must the *X* bits be applied to ensure accurate loading?
b. Suppose the clock has a frequency of 1 MHz



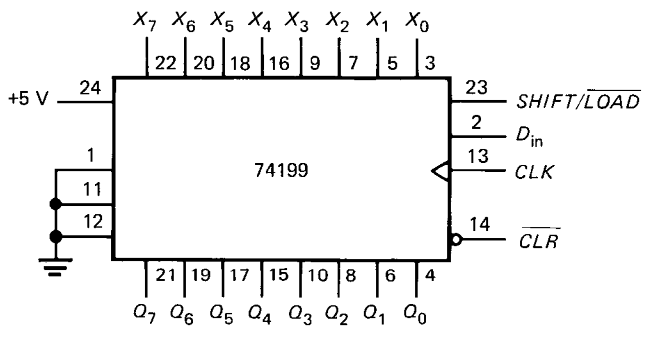
X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0
+5 V
74173
GND
CLR
$\bar{L}_O$
CLK
R_7 R_6 R_5 R_4 R_3 R_2 R_1 R_0
D F
A B C D E F
Note: All resistors are 1 kΩ.

Fig. 8-29

and the X bits are applied at the point D. Is the setup time sufficient for the data inputs?

c. How long must you wait after point E before removing the X bits or letting them change?

8-3. Each output pin of a 74173 can source up to 5.2 mA. In Fig. 8-29 suppose the high output voltage is 3.5 V and the LED drop is 1.5 V. To get more light out of the LEDs, we want to reduce the current-limiting resistors. What is the minimum allowable resistance?



X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0
+5 V
74199
SHIFT/$\overline{LOAD}$
D_{in}
CLK
$\overline{CLR}$
Q_7 Q_6 Q_5 Q_4 Q_3 Q_2 Q_1 Q_0

Fig. 8-30

8-4. A 74199 is an 8-bit shift-left register with a single control signal, as shown in Fig. 8-30. When $SHIFT/\overline{LOAD}$ is low, the circuit loads the **X** word on the next positive clock edge. When $SHIFT/\overline{LOAD}$ is high, the register shifts the bits to the left.

a. To clear the register, should $\overline{CLR}$ be low or high? When you are ready to run, what should $\overline{CLR}$ be?

b. Is the **X** word loaded on the positive or negative edge of the clock?

c. If **X** = 0100 1011, D_{in} = 0, and $SHIFT/\overline{LOAD}$ = 0, what does the **Q** output word equal after two positive clock edges?

d. If **X** = 0100 1011, D_{in} = 0, and $SHIFT/\overline{LOAD}$ = 1, what does the **Q** output word equal after two positive clock edges?

8-5. The clock frequency is 2 MHz. How long will it take to serially load the shift register of Fig. 8-30?

8-6. In Fig. 8-30, **Q** = 0001 0110. If $SHIFT/\overline{LOAD}$ is high and D_{in} is high, what does **Q** equal after three clock pulses?

8-7. Data from a satellite is received in serial form (1 bit after another). If this data is coming at a 5-MHz rate and if the clock frequency is 5 MHz, how long will it take to serially load a word in a 32-bit shift register?

8-8. A ripple counter has 16 flip-flops, each with a propagation delay time of 25 ns. If the count is

$$\mathbf{Q} = 0111\ 1111\ 1111\ 1111$$

how long after the next active clock edge before

$$\mathbf{Q} = 1000\ 0000\ 0000\ 0000$$

8-9. What is the maximum decimal count for the counter of the preceding problem?

8-10. When pins 1 and 12 of a 7490 are tied together as shown in Fig. 8-31, the divide-by-2 and divide-by-5 sections are cascaded to get a mod-10 counter. Pin 14 is the input and pin 11 is the output of each 7490. As a result, each 7490 acts like a divide-by-10 circuit and the overall circuit divides by 1,000.

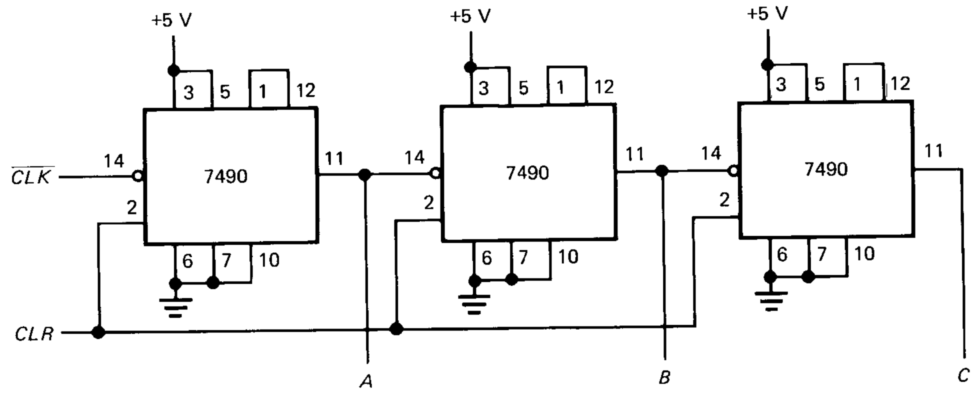


Fig. 8-31

If the clock has a frequency of 5 MHz, what is the frequency of *A*? Of *B*? Of *C*?

8-11. The clock signal driving a 6-bit ring counter has a frequency of 1 MHz. How long is each timing bit high? How long does it take to cycle through all the ring words?

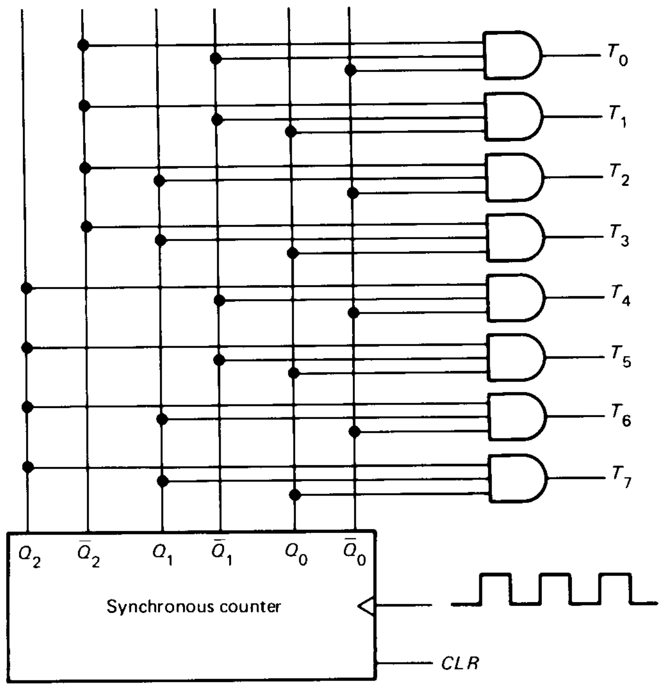


Fig. 8-32

8-12. Figure 8-32 shows another way to produce ring words. After the circuit is cleared,

$$\mathbf{Q} = Q_2Q_1Q_0 = 000$$

Since the AND gates are a 1-of-8 decoder, the first timing word is

$$\mathbf{T} = 0000\ 0001$$

What does **T** equal for each of the following:

a. **Q** = 001
b. **Q** = 010
c. **Q** = 101
d. **Q** = 111

8-13. If the clock frequency is 5 MHz in Fig. 8-32, how long does it take to produce all the ring words? How long is each timing bit high?

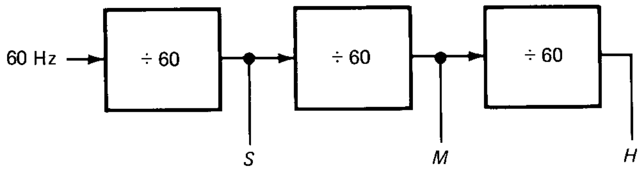


Fig. 8-33

8-14. In a digital clock, the 60-Hz line frequency is divided down to lower frequencies, as shown in Fig. 8-33. What are the frequency and period of the *S* output? Of the *M* output? Of the *H* output?

8-15. You have an unlimited number of the following ICs to work with: 7490, 7492, and 7493. Which of these would you use to build the divide-by-60 circuits of Fig. 8-33?

8-16. A presettable counter has eight flip-flops. If the preset number is 125, what is the modulus?

8-17. Given a presettable 8-bit counter, what number would you preset to get a divide-by-120 circuit?

8-18. In Fig. 8-34, we want to transfer the contents of register D to register C. Which are the *ENABLE* and *LOAD* inputs you should make high?

8-19. Look at Fig. 8-35 and answer each of these questions.

a. To add the inputs and put the answer on the bus, what should S_U and E_U be?
b. To subtract the inputs and put the answer on the bus, what should S_U and E_U be?
c. To isolate the ALU from the bus, what should E_U be?