Computer System Architecture

ECAP268

Edited by
Rishi Chopra

LOVELY PROFESSIONAL UNIVERSITY

LOVELY PROFESSIONAL UNIVERSITY

Computer System Architecture

Edited By:
Rishi Chopra

CONTENT

Unit 01: Binary Systems

CONTENTS

Objectives

After studying this unit, you will be able to

- Learn about the number systems
- Convert one number with a base to a number having different base
- Find out the complement of a number
- Do the subtraction operation using the complement of the number
- Understand the position of a binary point in a number

Introduction

The number system is the system of naming and representing the numbers which humans and computer systems can understand. The number systems are of various types:base-10 number system/decimal number system, base-2 number system/binary number system, base-8 number system/octal number system, base-16 number system/hexadecimal number system. The numbers which we use in our daily life belongs to the decimal number system. But our computer systems cannot understand the numbers of decimal number system as they use two states or binary logic. In binary logic, the two logic states are '0' and '1'. The other decimal number systems like octal and hexadecimal number systems are a compact way of representing the numbers of binary number systems. So, we can define a number differently in these number systems.

Example The numbers $(2A)_{16}$ and $(52)_8$have the same value $(42)_{10.}$

1.1 Number Systems

The number system is the numerical notation in which we use the digits or some other symbols for representation. It gives a distinctive representation of every number. It provides us with the privilege to do simple operations of arithmetic like addition, subtraction, and division.

Types of Number Systems:

The four most common number systems are:

- Base-10 Number System/Decimal Number System
- Base-2 Number System/Binary Number System
- Base-8 Number System/Octal Number System
- Base-16 Number System/Hexadecimal Number System

So, in every number system, we have to determine the value of a digit in that number. We can find out the value of any digit by using three things: the digit, its position in the number and the base of the number system.

Decimal Number System:

When digits form a string, we call it a number. These decimal numbers have a base of D or 10 or Dec or radix. In a base-10 number,we have the values which move from zero to nine: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 values.

Example A number like "4321" has four places, and each place can have the digits 0-9.

Every digit in the number has some power of 10 and it has some weight coupled to it. These two things depend upon the place of the digit in the number. We start processing the numbers starting from right, the powers of 10 will start from 0 (at the rightmost bit), and every time it will increment, like 0, 1, 2… etc. So, the procedure for calculation of a number's value is:

Number's value = a weighted sum of the digits.

Number' value = digit * 10^x + digit * 10^x

where x = (position number - 1).

Example 123410 = 1x 103 + 2x 102 + 3x101 +4x100 = 1000 + 200 + 30 + 4 = 123410

Example 9876_D = 9x 103 + 8x 102 + 7x101 +6x100 = 9000 + 800 + 70 + 6 = 9876_D

Task Find the number's value of $(6715)_D$

Binary Number Systems:

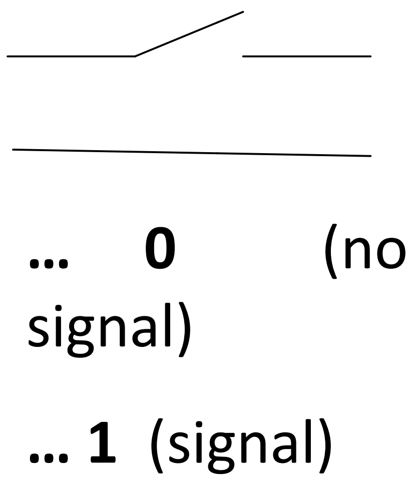
The term binary numbering formats corresponds to the system implemented in digital computers to represent numbers. The base in binary number systems is 2 or 'b' or 'B' or 'Bin'. This uses two symbols: 0 and 1.Each bit in the number is weighted by the power of 2.

Did You Know? The basic unit in digital computers is a binary digit which is abbreviated as a bit.

Binary numbers are made up of bits that can be represented electronically as shown in figure 1.The other units by which the binary information is processed are: nibble, byte, and word. A unit of four words is known as a nibble, unit of eight bits is known as a byte or an octet. Unit of sixteen bits is known as a word. All the information in the digital computer is represented as bit patterns. The important thing in the bit pattern is the number of bits. The number of bits can be calculated by counting the number of 0s and 1s.

Example 01010101: The number of bits in this pattern is 8.

Task Count the number of bits in 1010101010101010



Bit7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

Figure 1: Electronic Representation of Bits

Representation of Bit Pattern: Bit Positions and Their Significance

Table 1: Bit Positions and their Significance

Table 1 represents the bit positions in a number and their significance. There are 8 bits in the above table, Bit 0 is at the rightmost position, and it is the first bit to be processed. It is the bit which has the least weight so that is why it is known as the least significant bit. As we process the bits starting from the right and we keep on moving to the left. So, the last bit in the number has the most significance. That is why it is known as the most significant bit.

Octal Number System:

Computer mechanics often need to write out binary quantities, but in practice writing out a binary number such as 1001001101010001 is tedious and prone to errors. Therefore, binary quantities are written in a base-8 which is known as the octal representation. The base of octal numbers is 8 or 'o'

Example 123, 567, 7654

or 'Oct'. In this 8 symbols are used: 0, 1, 2, 3, 4, 5, 6, 7. Each number is weighted by the power of 8.

Hexadecimal Number System:

The hexadecimal number system base is 16 or 'H' or 'Hex'.16 symbols are used: 10 digits and 6 letters { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10=A, 11=B, 12=C, 13=D, 14=E, 15= F}.Each symbol is weighted bythe power of 16.

Example AB12, 876F, FFFF

1.2 Number System Conversion

For understanding, the numbers are converted from one system to another system. So, the types of conversions are shown in figures 2-5.

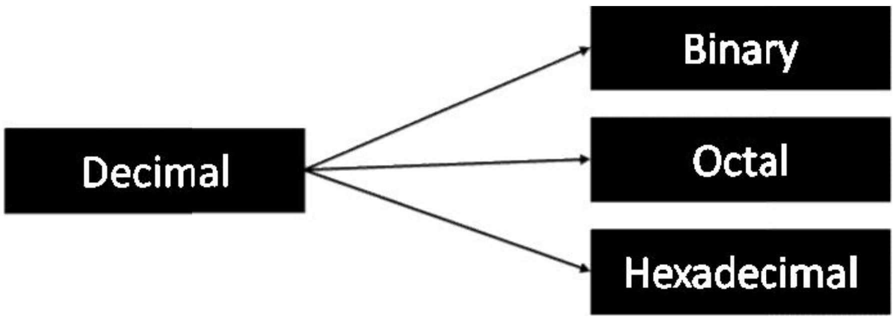


Figure 2: Conversions: Decimal number system to other number systems

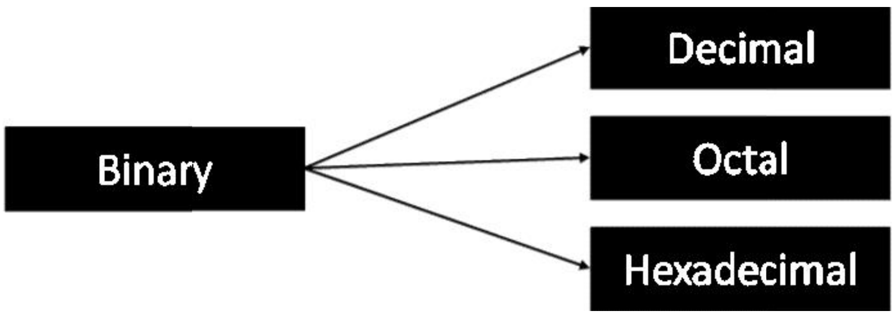


Figure 3: Conversion: Binary number system to other number systems

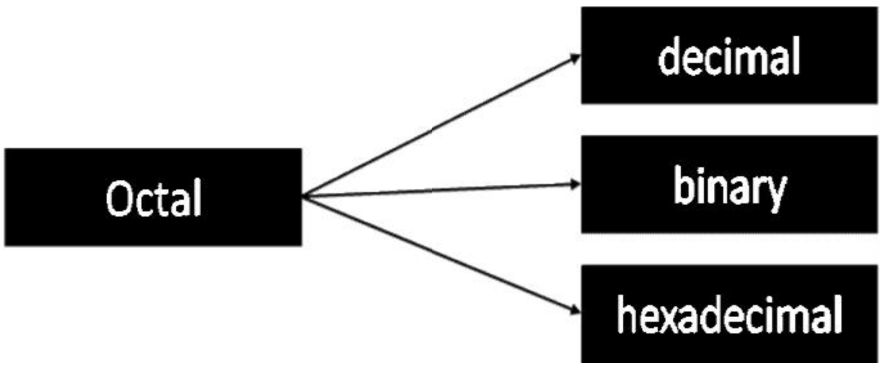


Figure 4: Conversion:Octal number system to other number systems

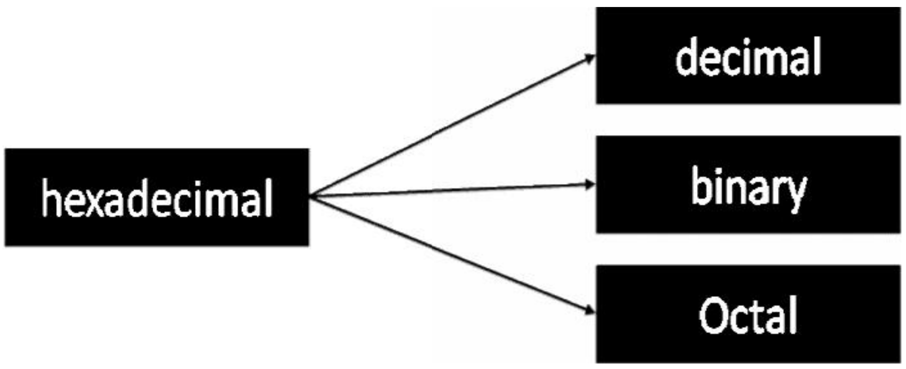


Figure 5: Conversion:Hexadecimal number system to other number systems

Decimal to Binary Conversion:

The procedure for converting a decimal number to a binary number is as:

Step 1: Take the number and divide it by 2,

Step 2: Get the integer quotient for the next repetition,

Step 3: Get the remainder for the binary digit,

Step 4: Repeat steps 1-3 until the quotient becomes 0.

Example

Convert $(41)_{10}$ to binary representation

	Quotient	Remainder
41/2	20	1
20/2	10	0
10/2	5	0
5/2	2	1
2/2	1	0
1/2	0	1

$(41)_{10} = (101001)_2$

Example

Convert $(35)_{10}$ to binary representation

	Quotient	Remainder

35/2	17	1
17/2	8	1
8/2	4	0
4/2	2	0
2/2	1	0
1/2	0	1

$(35)_{10} = (100011)_2$

Task Convert $(145)_{10}$ to binary representation

Decimal to Octal Conversion

The procedure for converting a decimal number to an octal number is:

Step 1: Take the number and divide it by 8,

Step 2: Findfor the next iteration, the integer quotient,

Step 3: Get the remainder for the binary digit,

Step 4: Repeat steps 1-3 until the quotient becomes 0.

Example Convert $(153)_{10}$ to octal representation

	Quotient	Remainder
153/8	19	1
19/8	2	3
2/8	0	2

$(153)_{10} = (231)_8$

Example Convert $(670)_{10}$ to octal representation

	Quotient	Remainder
670/8	83	6
83/8	10	3
10/8	1	2
1/8	0	1

$(670)_{10} = (1236)_8$

Task Convert $(3521)_{10}$ to octal representation

Decimal to Hexadecimal Conversion:

The procedure for decimal to hexadecimal conversion is:

Step 1: Divide the number by 16,

Step 2: Get the integer quotient for the next iteration,

Step 3: Get the remainder for the binary digit,

Step 4: Repeat the steps until the quotient becomes 0.

Example

Convert $(4735)_{10}$ to hexadecimal representation

	Quotient	Remainder
4735/16	295	15 (F)
295/16	18	7
18/16	1	2
1/16	0	1

$(4735)_{10} = (127F)_{16}$

Example

Convert $(2020)_{10}$to hexadecimal representation

	Quotient	Remainder
2020/16	126	4
126/16	7	14 (E)
7/16	0	7

$(2020)_{10} = (7E4)_{16}$

Task Convert $(9845)_{10}$ to hexadecimal representation.

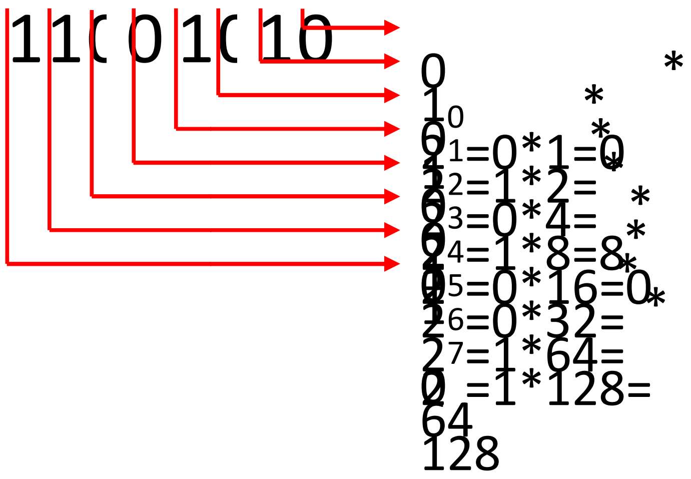
Binary to Decimal Conversion

The procedure for conversion from binary to decimal is as follows:

Step 1: Multiply each bit with the power of 2 (from LSB to MSB) (0 to n).

Example

Convert $(11001010)_2$ to decimal representation



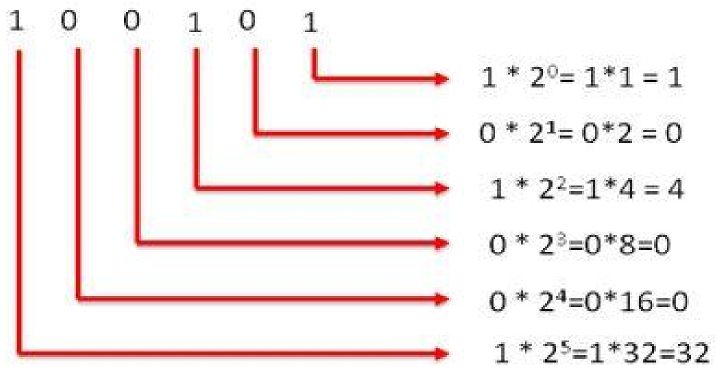
0 + 2 + 0 + 8 + 0 + 0 +64 +128 = 202

$(11001010)_2 = (202)_{10}$

Step 2: Add all the product values.

Example

Convert $(100101)_2$ to decimal representation



1 + 0 + 4 + 0 + 0 + 32 = 37

$(100101)_2 = (37)_{10}$

Task

Convert $(100101011)_2$ to decimal representation

Binary to Octal Conversion:

The procedure from octal to binary conversion is as follows:

Step 1: Multiply each bit with the power of 2 (from LSB to MSB) (0 to n).

Step 2: Add all the product values.

Step 3: Divide the number by 8.

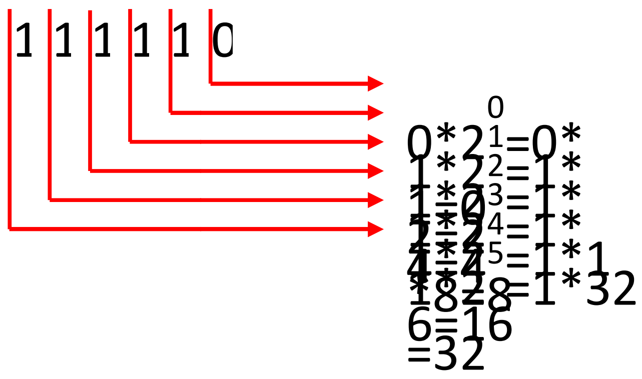
Step 4: Get the integer quotient for the next iteration.

Step 5: Get the remainder for the binary digit.

Step 6: Repeat the steps until the quotient becomes 0.

Example

Convert $(111110)_2$ to octal representation



0 + 2 + 4 + 8 + 16 + 32 = 62

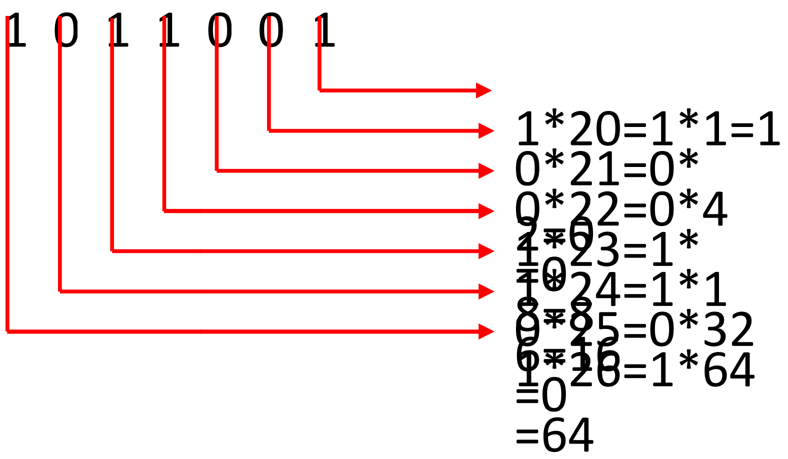
$(111110)_2 = (62)_{10}$

	Quotient	Remainder
62/8	7	6
7/8	0	7

$(111110)_2 = (62)_{10} = (76)_8$

Example

Convert $(1011001)_2$ to octal representation



1 + 0 + 0 + 8 + 16 + 0 + 64 = 89

$(1011001)_2 = (89)_{10}$

	Quotient	Remainder
89/8	11	1
11/8	1	3
1/8	0	1

$(1011001)_2 = (89)_{10} = (131)_8$

1.3 Binary to Hexadecimal Conversion

The procedure for conversion from binary to hexadecimal is:

Step 1: Multiply each bit with the power of 2 (from LSB to MSB) (0 to n).

Step 2: Add all the product values.

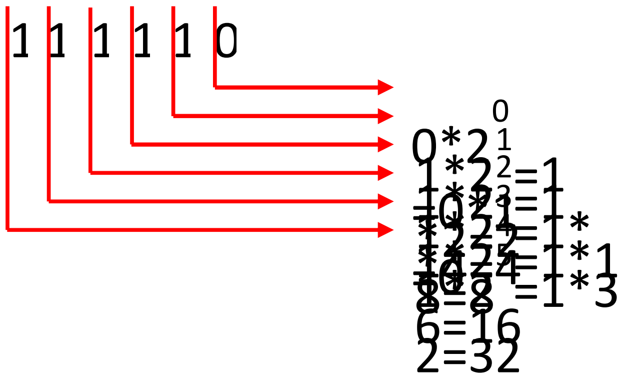
Step 3: Divide the number by 16.

Step 4: Get the integer quotient for the next iteration.

Step 5: Get the remainder for the binary digit.

Step 6: Repeat the steps until the quotient becomes 0.

Example Convert $(111110)_2$ to hexadecimal representation.



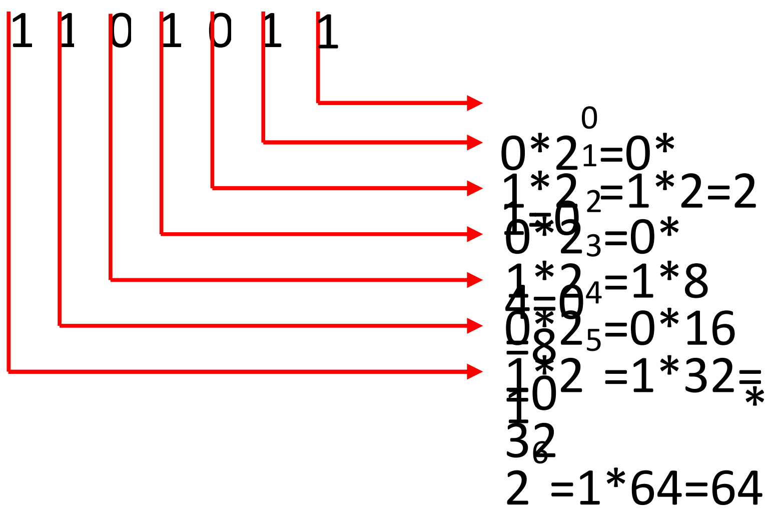
0 + 2 + 4 + 8 + 16 + 32 = 62

$(111110)_2 = (62)_{10}$

	Quotient	Remainder
62/16	3	14(14=E)
3/16	0	3

$(111110)_2 = (62)_{10} = (3E)_{16}$

Example **Convert $(1101011)_2$ to hexadecimal representation**



0 + 2 + 0 + 8 + 0 + 32 + 64 = 106

$(1101011)_2 = (106)_{10}$

	Quotient	Remainder
106/16	6	10 (10=A)
6/10	0	6

$(1101011)_2 = (106)_{10} = (6A)_{16}$

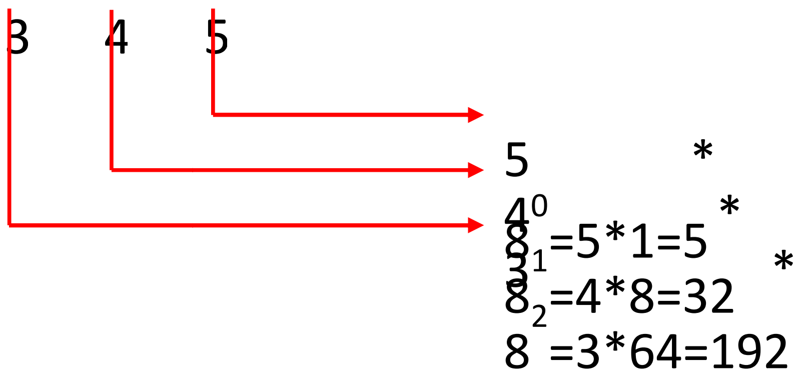
1.4 Octal to Decimal Conversion

The procedure for converting octal to decimal number is:

Step 1: Multiply each bit with the power of 8 (from LSB to MSB) (0 to n).

Step 2: Add all the product values to get the number.

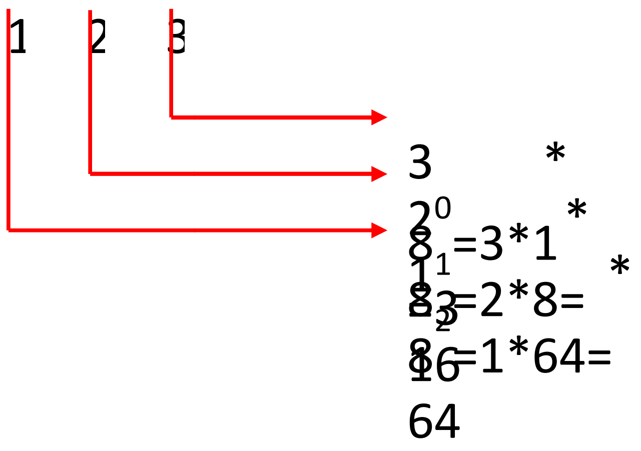
Example **Convert $(345)_8$ to decimal representation**



5 + 32 + 192 = 229

$(345)_8 = (229)_{10}$

Example **Convert $(123)_8$ to decimal representation.**



3+ 16 + 64 = 38

$(123)_8= (38)_{10}$

1.5 Octal to Binary Conversion

The procedure for conversion from octal to binary is as follows:

Step 1: Multiply each bit with the power of 8 (from LSB to MSB) (0 to n).

Step 2: Add all the product values and get the number.

Step 3: Divide the number by 2.

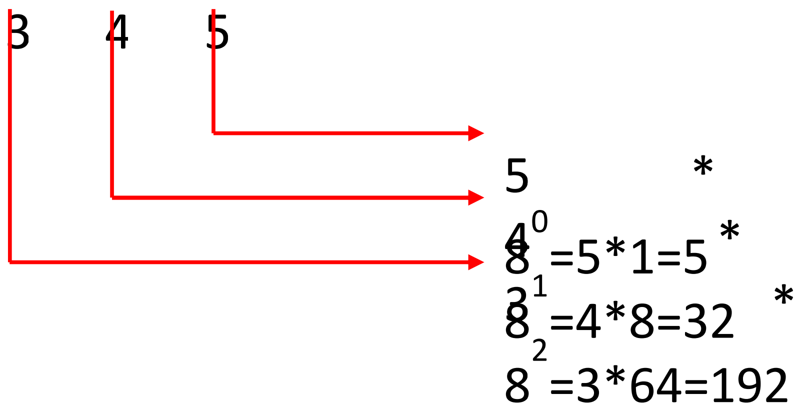
Step 4: Get the integer quotient for the next iteration.

Step 5: Get the remainder for the binary digit.

Step 6: Repeat the steps until the quotient becomes 0.

Example

Convert $(345)_8$ to binary representation



5 + 32 + 192 = 229

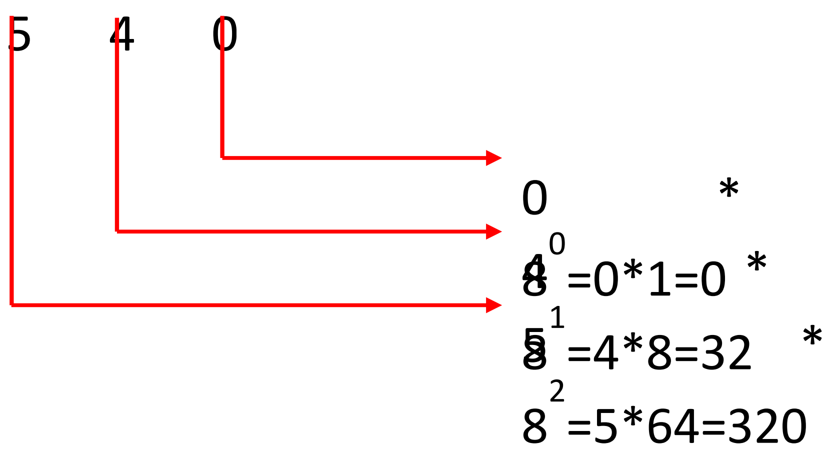
$(345)_8 = (229)_{10}$

	Quotient	**Remainder**
229/2	114	1
114/2	57	0
57/2	28	1
28/2	14	0
14/2	7	0
7/2	3	1
3/2	1	1
1/2	0	1

$(345)_8 = (229)_{10} = (11100101)_2$

Example

Convert $(540)_8$ to binary representation



0 + 32 + 320 = 352

$(540)_8 = (352)_{10}$

	Quotient	**Remainder**
352/2	176	0
176/2	88	0
88/2	44	0
44/2	22	0
22/2	11	0
11/2	5	1
5/2	2	1
2/2	1	0
1/2	0	1

$(540)_8 = (352)_{10} = (101100000)_2$

Octal to Hexadecimal Conversion

The procedure for converting octal to hexadecimal is as follows:

Step 1: Multiply each bit with the power of 8 (from LSB to MSB) (0 to n).

Step 2: Add all the product values to get the number.

Step 3: Divide the number by 16.

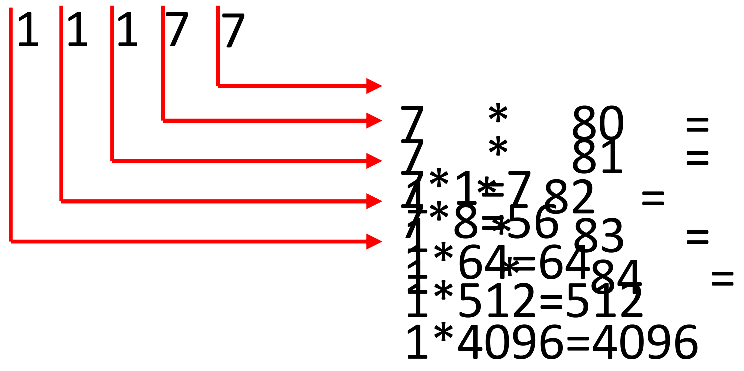
Step 4: Get the integer quotient for the next iteration.

Step 5: Get the remainder for the binary digit.

Step 6: Repeat the steps until the quotient becomes 0.

Example

Convert $(11177)_8$ to hexadecimal



representation

7 + 56 + 64 + 512 + 4096 = 4735

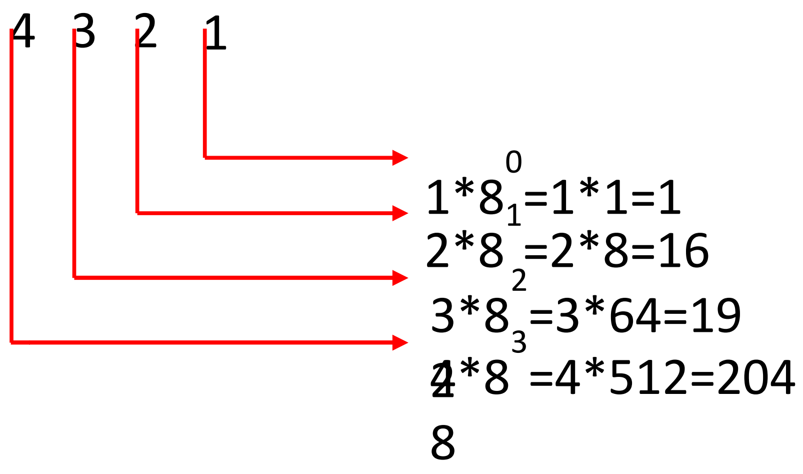
$(11177)_8 = (4735)_{10}$

	Quotient	Remainder
4735/16	295	15 (F)
295/16	18	7
18/16	1	2
1/16	0	1

$(11177)_8 = (4735)_{10} = (127F)_{16}$

Example

Convert $(4321)_8$ to hexadecimal representation



1 + 16 + 192 + 2048 = 2257

$(4321)_8 = (2257)_{10}$

	Quotient	Remainder
2257/16	141	1
141/16	8	13 (13=D)
8/16	0	8

$(4321)_8 = (2257)_{10} = (8D1)_{16}$

Hexadecimal to Decimal Conversion

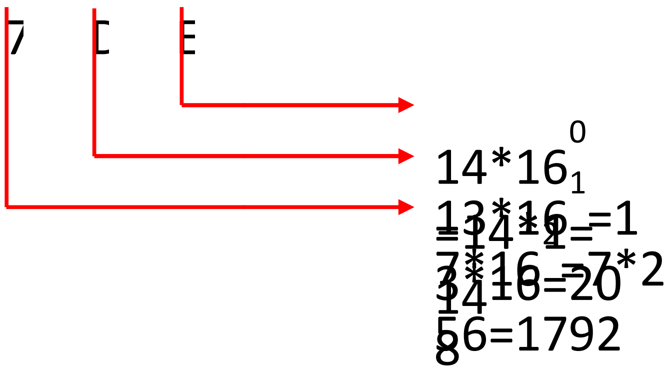
The procedure for converting hexadecimal to decimal conversion is as follows:

Step 1: Multiply each bit with the power of 16 (from LSB to MSB) (0 to n).

Step 2: Add all the product values to get the number.

Example

Convert $(7DE)_{16}$ to decimal representation

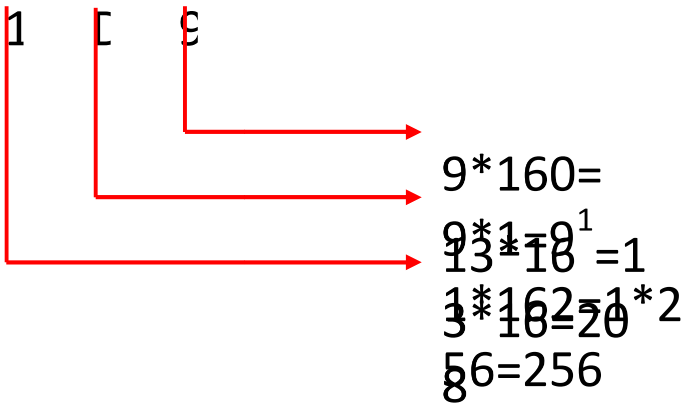


14 + 208 + 1792 = 2014

$(7DE)_{16} = (2014)_{10}$

Example

Convert $(1D9)_{16}$ to decimal representation



9 + 208 + 256 = 473

$(1D9)_{16} = (473)_{10}$

Hexadecimal to Binary Conversion

The procedure for converting from hexadecimal to binary conversion is as follows:

Step 1: Multiply each bit with the power of 16 (from LSB to MSB) (0 to n).

Step 2: Add all the product values to get the number.

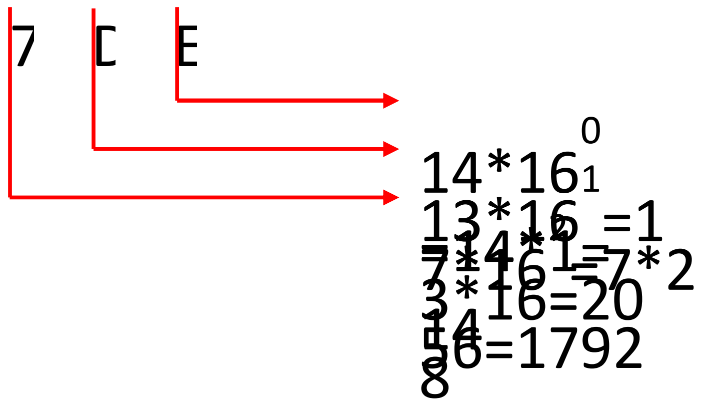
Step 3: Divide the number by 2.

Step 4: Get the integer quotient for the next iteration.

Step 5: Get the remainder for the binary digit.

Step 6: Repeat the steps until the quotient becomes 0.

Example **Convert $(7DE)_{16}$ to binary representation.**



14 + 208 + 1792 = 2014

$(7DE)_{16} = (2014)_{10}$

	Quotient	Remainder
2014/2	1007	0
1007/2	503	1
503/2	251	1
251/2	125	1
125/2	62	1
62/2	31	0
31/2	15	1
15/2	7	1
7/2	3	1
3/2	1	1
1/2	0	1

$(7DE)_{16} = (2014)_{10} = (11111011110)_2$

Hexadecimal to Octal

The procedure for converting from hexadecimal to octal is as follows:

Step 1: Multiply each bit with the power of 16 (from LSB to MSB) (0 to n).

Step 2: Add all the product values to get the number.

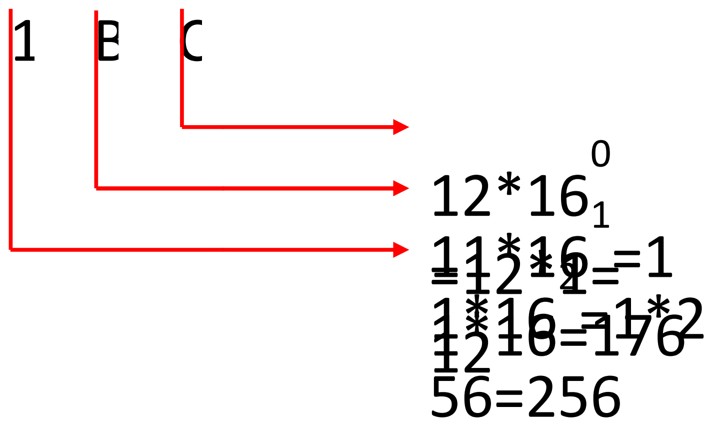
Step 3: Divide the number by 8.

Step 4: Get the integer quotient for the next iteration.

Step 5: Get the remainder for the binary digit.

Step 6: Repeat the steps until the quotient becomes 0.

Example **Convert $(1BC)_{16}$ to octal representation**



12 + 176 + 256 = 444

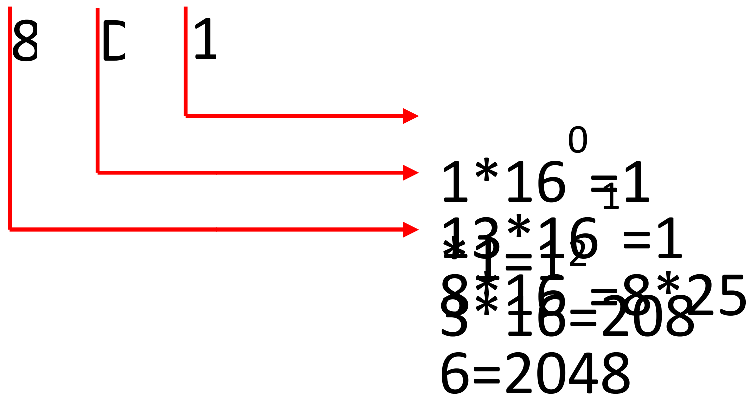
$(1BC)_{16} = (444)_{10}$

	Quotient	Remainder
444/8	55	4
55/8	6	7
6/8	0	6

$(1BC)_{16} = (444)_{10} = (674)_8$

Example

Convert $(8D1)_{16}$ to octal representation



1 + 208 + 2048 = 2272

$(8D1)_{16} = (2272)_{10}$

	Quotient	**Remainder**
2257/8	282	1
282/8	35	2
35/8	4	3
4/8	0	4

$(8D1)_{16} = (2272)_{10} = (4321)_8$

Decimal	Binary	Octal	Hexadecimal
0	00	0	0
1	01	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1.6 Complements

Complement is obtained by interchanging 1's and 0's in the values of F. The logic behind the complement is: When you are taking the complement, just involve the elements which are not present in the set. Like in Boolean algebra, the values can be 0 or 1. So, U = {0, 1}. If A= {0}, then the complement will be A′ = {1} and if A= {1}, then the complement will be A′ = {0}.

U ={a, e, i, o, u}.

A={a, e, i}.

Complement of A, i.e., A′ = U-A = {o, u}.

Types of Complements

There are two types of complements for each r-base system:

- Diminished radix complement ((r-1)'s complement)
- Radix complement (r's complement)

When we substitute the value of base here, these are referred to as:

- 2's complement and 1's complement (For binary numbers).
- 10's complement and 9's complement (For decimal numbers).

Diminished Radix Complement

Given a number Nin base r havingn digits, the complement of N = (r^n-1)-N

For decimal numbers r=10 and (r-1)=9,So, 9's complement of number is N = (10^n-1)-N

10^n represents a number that consists of a single 1 followed by n 0s. 10^n -1 is a number represented by n 9s.

Example If n=4,

10^4 =10,000

10,000-1=9,999.

For binary numbers r=2 and (r-1)=1, So, 2's complement of a number N= $(2^n – 1)$-N.

2^n represents a binary number that consists of a 1 followed by n 0s. 2^n -1 is a number represented by n 1s

Example If n=4

$2^4 = (10000)_2$

$2^4-1=(1111)_2$.

Example Find the 9's complement of 546700

999999-546700=453299.

Example Find the 9's complement of 012938

999999-012938=987061.

Example Find the 1's complement of 1011000

1111111-1011000=0100111.

Example Find the 1's complement of 0101101

1111111-0101101=1010010.

Radix Complement

The r's complement of n - digit number N in base r is defined as

R^n-N	for	N≠0
0	for	N=0

The r's complement can also be obtained by adding 1 to the (r-1)'s complement since

$R^n – N = [(r^n- 1) - N] + 1$

Example

Find 10's complement of decimal 2389

It is obtained by adding 1 to its 9's complement

7610+1=7611.

Example

Find 2's complement of binary 101100

It is obtained by adding 1 to its 1's complement

010011+1=010100.

Example

Find the 10's complement of 012398

9's complement - 999999-012398=987601

10's complement – 987601+1=987602.

Example

Find the 10's complement of 246700

9's complement – 999999-246700=753299

10's complement – 753299+1=753300.

Example

Find the 2's complement of 1101100

1's complement – 1111111-1101100=0010011

2's complement – 0010011+1=0010100.

Example

Find the 2's complement of 0110111

1's complement – 1111111-0110111=1001000

2's complement – 1001000+1=1001001.

Subtraction of Complements:

The simple operation subtraction can be easily done using the complements.

Example

Using 10's complement, subtract 72532-3250

M= 72532

10's complement of N= 96750

Sum=169282

So, discard end carry here, answer=69282

Example

Using 10's complement, subtract 3250-72532

M=03250

10's complement of N=27468

Sum=30718

There is no end carry. So, answer=-(10's complement of 30718)=-69282

Example

Using 2's complement, subtract 1010100-1000011

M=1010100

2's complement of N=0111101

Sum=10010001

So, discard the end carry, answer=0010001

Example

Using 2's complement, subtract 1000011-1010100

M=1000011

2's complement of N=0101100

Sum=1101111

There is no end carry, aanswer=-(2's complement of 1101111)=-0010001

1.7 Fixed-Point and Floating-Point Representation

Positive integers, including 0, can be represented as unsigned numbers. However, to represent negative integers, we need a notation for negative values. In computer systems, the hardware is limited, so everything must be represented using 1's and 0's, including the sign of a number. So, it will be better if we represent the sign in the leftmost position bit of the number. The convention is to make the sign bit 0 for positive and 1 for negative.

In addition to the sign, a number may have a decimal (or binary) point. The position of binary point be needed to represent the fractions, integers, or mixed-integer fraction numbers.There are two ways to specify the position of a binary point in a register:

A) By giving it a fixed position

B) By employing a floating-point representation

Fixed-Point Representation:

This method assumes that the binary point is always fixed in one position. The two positions most widely used are:

A) The binary point in the extreme left of the register to make the stored number a fraction

B) The binary point in the extreme right of the register to make the stored number an integer.

In either case, the binary point is not present, but its presence is assumed from the fact that number stored in the register is treated as a fraction or an integer.When the integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number.When the number is negative, the sign is represented by 1 but the rest of the number may be represented in one of three possible ways:

A) Signed magnitude representation

B) Signed 1's complement representation

C) Signed 2's complement representation

Example

Consider the signed number 14 stored in a 14-bit register

+14 is represented by a sign bit of 0 in the leftmost position followed by the binary equivalent of 14: 00001110. There is only one way to represent the positive signed numbers.

-14 can be represented by three ways with eight bits:

A) Signed magnitude representation : 1 0001110

B) Signed 1's complement representation: 1 1110001

C) Signed 2's complement representation: 1 1110010

Floating-Point Representation

In this method, it uses a second register to store a number that designates the position of the decimal point in the first register. The floating point representation of a number has two parts:

A) The first part represents a signed, fixed point number called the mantissa.

B) The second part contains the position of the decimal (or binary) point is called the exponent.

Example

The fixed point mantissa may be a fraction or an integer. For example: the decimal number +6132.789 is represented in floating point with a fraction and an exponent as:

fraction +0.6132789

exponent +04

The value of the exponent indicates the actual position of the decimal point. This can also be represented as $+0.6132789 * 10^4$

Summary

- The number system is used to represent any number in the world.
- The decimal number system can be easily understood by human.
- But the digital computer systems can understand only binary number systems.
- Binary numbers are used to represent 0s and 1s information in the computers.
- The other systems, i.e., octal and hexadecimal number systems are compact representation of binary number systems.
- Fixed-point numbers are used to represent factorial numbers.
- Floating-point numbers are used to represent a decimal point which is multiplied by a base value and it is scaled up with some exponent value.
- There are three ways to represent the magnitude of the signed binary numbers namely, the sign and magnitude representing, the signed and 1's complement representation, and the signed and 2's complement representation.

Keywords

Number System: It is a system of naming the numbers.
Base: The base of a number is the number of digits or the combination of digits that a system of counting uses to represent the numbers.
Conversion: The conversion of a number from one number system to another number system.
1's Complement: It is a method for the representation of negative numbers in the computers.
2's Complement: It is a method for the representation of negative binary numbers in computers.

SelfAssessment

1. What is the weight associated with digit 4 in 4531?

A. 10^2

B. 10^3
C. 10^4
D. 10^5

2. A unit of 8 bits is known as
A. A byte
B. A word
C. A nibble
D. A sentence

3. The hexadecimal number A is equivalent to ________ octal number.
A. 9
B. 10
C. 11
D. 12

4. What is the compact way of call this whole unit of bits - 10101111?
A. A byte
B. A word
C. A nibble
D. None of these

5. What is MSB in 10101110?
A. 1
B. 0

6. Convert the number $(74)_{10}$ into binary representation.
A. 0101001
B. 1001010
C. 1101010
D. 1100110

7. Convert the number $(537)_{10}$ to octal representation
A. 1141
B. 1411

8. Convert the number $(11001)_2$to decimal representation
A. 48
B. 49

9. Convert the number $(436)_8$to binary representation
A. 100100010
B. 110110010
C. 10001001
D. 101000100

10. Convert the number $(C76E)_{16}$ to octal representation
A. 51040
B. 51054

11. In which type of complement, we convert 0 to 1 or 1 to 0?
A. 1's complement
B. 2's complement
C. 3's complement
D. None of these

12. What is the 9's complement of 189023?
A. 810976
B. 023189
C. 180023
D. 189024

13. How can we represent -15 using signed 1's complement representation?
A. 111101001
B. 100010111
C. 111101000
D. None of these

14. In fixed-point representation of numbers, the binary point is always fixed in which position.
A. Extreme left
B. Extreme right
C. Either Extreme Left or Extreme Right
D. None of the above

15. When the sign is represented by 0, then the integer binary number is
A. Positive
B. Negative
C. 0
D. Any of the above

Answers for SelfAssessment

1.	B	2.	A	3.	D	4.	A	5.	A
6.	B	7.	A	8.	B	9.	A	10.	B
11.	A	12.	A	13.	C	14.	C	15.	A

Review Questions

Q 1: Write the procedure for finding out the value of each digit in a number

Q 2: What are the different steps involved in converting a binary number to other number systems?

Q 3: What to find out the 9's complement and 10's complement of a decimal number?

Q 4: What are the two ways to specify the position of a binary point in a register? Explain in detail.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education Asia, 2002.

M. Chandni, Lecture Notes on Digital Circuits and Systems, Chadalawada Ramanamma Engineering College, Department of Electrical and Electronics and Engineering, 2018-2019.

Web Links

1. https://www.tutorialspoint.com/computer_fundamentals/computer_number_system.htm
2. https://www.splashlearn.com/math-vocabulary/number-sense/number-system
3. https://www.vedantu.com/maths/number-system
4. https://www.geeksforgeeks.org/number-system-in-maths/

Unit 02: Boolean Algebra

CONTENTS

Objectives

After studying this unit, you will be able to

- Learn about the basic and axiomatic definitions of Boolean algebra
- Use basic theorems and properties in Boolean algebra
- Simplify the Boolean functions
- Understand the standard forms of an expression

Introduction

Boolean algebra is the branch of algebra in which the values of the variables are the truth values, i.e., true and false, usually denoted by 1 and 0 respectively. It is like any other mathematical system, which may be defined with a set of elements, a set of operators and a number of postulates.

A set of elements is any collection of objects having a common property. If S is a set, x and y are certain objects, then $x \in S$ denotes that x is a member of set S and $y \notin S$ denotes that y is not an element of S.Consider the relation a*b=c, we say that * is a binary operator if it specifies a rule for finding c from a pair (a,b) and also if $a,b,c \in S$. However, * is not a binary operator of $a,b \in S$ but $c \notin S$.

2.1 Postulates of Algebraic Structures

The postulates of a mathematical system form the basic assumptions from which it is possible to deduce the rules, theorems and properties of the system.The most common postulates used to formulate various algebraic structures are:

1) Closure
2) Associative law
3) Commutative law
4) Identity element
5) Inverse
6) Distributive law
7) Double inversion law
8) Annulment law
9) Idempotent law
10) Complement law
11) Absorptive law

Closure

A set is closed with respect to a binary operator if, for every pair of elements of S, the binary operator specifies a rule for obtaining a unique element of S.For example: The set of natural numbers N= {1,2,3,4........} is closed with respect to the binary operator + by the rules of arithmetic addition, since for any a,b∈N we obtain a unique c∈N by the operation of a+b=c. But the set of natural numbers is not closed with respect to binary operator minus (-) by the rules of arithmetic subtraction because 2-3=-1 and 2,3∈N while (-1) ∉ N.

Associative Law

This law allows the removal of brackets from an expression and regrouping of the variables.

- A + (B + C) = (A + B) + C = A + B + C (OR Associate Law)
- A(B.C) = (A.B) C = A .B. C (AND Associate Law)

Commutative Law

The order of application of two separate terms is not important

- A . B = B . A The order in which two variables are AND'ed makes no difference
- A + B = B + A The order in which two variables are OR'ed makes no difference

Identity element

A term OR´ed with a "0" or AND´ed with a "1" will always equal that term

- A + 0 = A A variable OR'ed with 0 is always equal to the variable
- A . 1 = A A variable AND'ed with 1 is always equal to the variable

Distributive Law

This law permits the multiplying or factoring out of an expression.

- A(B + C) = A.B + A.C (OR Distributive Law)
- A + (B.C) = (A + B).(A + C) (AND Distributive Law)

Double Inversion Law

A term that is inverted twice is equal to the original term

- (A')' = A A double complement of a variable is always equal to the variable

Annulment law

A term AND´ed with a "0" equals 0 or OR´ed with a "1" will equal 1

- A . 0 = 0 A variable AND'ed with 0 is always equal to 0
- A + 1 = 1 A variable OR'ed with 1 is always equal to 1

Idempotent law

An input that is AND´ed or OR´ed with itself is equal to that input

- A + A = A A variable OR'ed with itself is always equal to the variable
- A . A = A A variable AND'ed with itself is always equal to the variable

Complement law

A term AND´ed with its complement equals "0" and a term OR´ed with its complement equals "1"

- A . A = 0 A variable AND'ed with its complement is always equal to 0
- A + A = 1 A variable OR'ed with its complement is always equal to 1

Absorptive law

This law enables a reduction in a complicated expression to a simpler one by absorbing like terms.

- A + (A.B) = A (OR Absorption Law)
- A(A + B) = A (AND Absorption Law)

Axiomatic Definition of Boolean algebra

For the treatment of logic, the Boolean Algebraic system was developed in 1854. The switching algebra which is a two-valued Boolean algebra was developed in 1983. So, this is an algebraic structure which uses a set of elements B having two binary operators + and . has provided a number of postulates

1) Closure w.r.t. +
2) Closure w.r.t. .
3) x+0 = 0+x = 0
4) x+y = y+x
5) x.(y+z) =x.y+x.z
6) x+(y.z)=(x+y)(x+z)
7) x+x'=1
8) x.x'=0

2.2 Basic Theorems and Properties

Duality principle

If a statement is true, then also its dual statement is true. We can obtain the dual statement by changing + for., . for +, 1 for 0 and 0 for 1. Examples:

- 0.1=0: is a true statement asserting that "false and true evaluates to false".
- 1+0=1: is a true statement asserting that "true or false evaluates true".

Basic Theorems

1. A+0=A
2. A+1=1
3. A.0=0
4. A.1=A
5. A+A=A
6. A+A′=1
7. A.A=A
8. A.A′=0
9. (A′)′=A
10. A+AB=A
11. A+A′B=A+B
12. (A+B)(A+C)=A+BC

2.3 De-Morgan's Theorem

De-Morgan's theorems are basically two sets of rules or laws developed from the Boolean expressions for AND, OR and NOT using two input variables, A and B. These two rules or theorems allow the input variables to be negated and converted from one form of a Boolean function into an opposite form.

Inputs		**Outputs (Theorem 1)**				
B	A	A.B	(A.B)′	A′	B′	A′+B′
0	0	0	1	1	1	1
0	1	0	1	0	1	1
1	0	0	1	1	0	1
1	1	1	0	0	0	0

Inputs		**Outputs (Theorem 2)**				
B	A	A+B	(A+B)′	A′	B′	A′.B′
0	0	0	1	1	1	1
0	1	1	0	0	1	0
1	0	1	0	1	0	0
1	1	1	0	0	0	0

De-Morgan's First Theorem:

De-Morgan's First Theorem: When two (or more) input variables are AND'ed and negated, they are equivalent to the OR of the complements of the individual variables. Thus the equivalent of the NAND function and is a negative-OR function proving that A.B = A+B. Complementing the result of AND'ing variables together is equivalent to OR'ing the complements of the individual variables.

De-Morgan's Second Theorem:

DeMorgan's Second Theorem: It proves that when two (or more) input variables are OR'ed and negated, they are equivalent to the AND of the complements of the individual variables. Thus the equivalent of the NOR function and is a negative-AND function proving that (A+B)′ = A′.B′ and again we can show this using the following truth table.

2.4 Operator Precedence

The order of operatorsfor evaluation of Boolean expressions is:

1. Parentheses
2. NOT
3. AND
4. OR

It is described as the expression we have in parentheses will be evaluated first. Here the NOT is described in the terms of complement, so will be evaluated next. Then AND and finally the OR.

Boolean Function

A binary variable can take the value either 0 or 1. A Boolean function is an expression formed with binary variables, two binary operators AND, OR and one unary operator NOT, parentheses and an equal sign.For the given value of variables, the function can be either 0 or 1.

Example:F1=xyz′. The function F1 will be 1 if x, y and z′ are 1, otherwise F1=0.

Thus in this example, the Boolean function is represented as an algebraic example; we can also represent these in truth table.If there are n variables, then there will be 2^n combinations of 1's and 0's.

F1=xyz′,   F2=x+y′z,   F3=x′y′z+x′yz+xy′,   F4=xy′+x′z

x	y	z	F1	F2	F3	F4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1

1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Did you Know?

It is possible to find two algebraic expressions that specify the same functions.

Two functions of n binary variables are said to be equal if they have the same value for all possible 2^n combinations of the n variables.

2.5 Algebraic Manipulation

A literal is the use of the variable or its complement form in the expression.The minimization of number of literals and the number of terms results in a circuit with less equipment.The number of literals can be minimized by the algebraic manipulations.

Example: $x+x'y$

$=(x+x')(x+y)$

$=(1)(x+y)$

$=x+y$

Example: $x(x'+y)$

$=xx'+xy$

$=0+xy$

$=xy$

Example: Q: $x'y'z+x'yz+xy'$

$=x'z(y'+y)+xy'$

$=x'z(1)+xy'$

$=x'z+xy'$

Example: $xy+x'z+yz$

$=xy+x'z+yz(x+x')$

$=xy+x'z+xyz+x'yz$

$=xy+xyz+x'z+x'yz$

$=xy(1+z)+x'z(1+y)$

$=xy+x'z$

Task: Simplify the Boolean Function: BC +B′C+ BA

Task: Simplify the Boolean Function: A(A+B)+B(A+AA)(A+B)

Task: Simplify the Boolean Function: A+A′B+A′B′C+A′B′C′D+A′B′C′D′E

Canonical and standard forms

All Boolean expressions regardless of their form can be converted to either of these two forms:

- SOP
- POS

2.6 Sum of Products

A product term is a term consisting of the Boolean multiplication of literals. It is also known as min-term. When two or more min-terms are summed by Boolean addition, the resulting expression is a sum of products (SOP) form.

- AB+ABC
- ABC+CDE+B′CD′
- A′B+A′BC′+AC

SOP can also contain a single variable term such as A′+A′B′C+BCD′.In SOP form, single bar cannot extend over more than one variable however more than one variable in a term can have an over-bar. Example: SOP expression can have A′B′C′ but it cannot have (ABC)′.

2.7 Product of Sums

A sum is defined as a term consisting of Boolean addition of the literals. It is also known as max-term. When two or more sum terms are multiplied, the resulting expression is a product of sum (POS) form.

- (A′+B)(A+B′+C)
- (A′+B′+C′)(C+D′+E)(B′+C+D)
- (A+B)(A+B′+C)(A′+C)

It can also contain a single variable such as A(A′+B′+C′)(C+D′+E)(B′+C+D). In POS form, single bar cannot extend over more than one variable however more than one variable in a term can have an overbar. For example: a POS expression can have the term A′+B′+C′ but not (A+B+C)′.

K-Map

It is a systematic method for simplifying the Boolean expressions.It produces simplest POS or SOP which is known as minimum expression.It is similar to TT because it represents all the possible values of IP variables and the resulting OP for each value.K-Map is an array of cells in which each cell represents a binary value of IP variables.It can be used for expressions with two, three, four and five variables.

Number of Cells

- For 2 variables, no of cells=2^2=4

- For 3 variables, no of cells=2^3=8
- For 4 variables, no of cells=2^4=16
- For 5 variables, no of cells=2^5=32
-

Cell adjacency

The cells in a k-map are arranged so that there is only a single variable change between adjacent cells.Adjacency: It is defined by a single variable change. Cells that differ by only one variable are adjacent cells.

Example: For 3 variable k-map

- 010 is adjacent to 000, 011,110
- 010 is not adjacent to 001, 111

So, physically each cell is adjacent to the cells that are immediately next to it on any of its four sides.A cell is not adjacent to the cells that diagonally touch any of its corner.

The cells in the top row are adjacent to the corresponding cells in the bottom row and the cells in the outer left column are adjacent to the corresponding cells in the outer right column. You can think of the map as wrapping around from top to bottom to form a cylinder or left to right to form a cylinder.

2-variable k-map

- For two variables, there are four min-terms.

3-variable k-map

- For three variables, there are eight min-terms.

4-variable k-map

- For four variables, there are sixteen min-terms.

Task: How many min-terms will be there for five variables?

2.8 K-Map SOP Minimization

A minimized SOP contains the fewest possible terms with the fewest possible variables per term. After SOP expression is mapped, there are three steps in obtaining a minimum SOP expression.

1) Grouping of 1's.

2) Determine the product term of each group.

3) Summing the resulting product term.

Example: Simplify the boolean function: A'B'C+A'BC'+ABC'+ABC

A/BC	00	01	11	10
0		1		1
1			1	1

Simplified Boolean function: A′B′C + AB + BC′

Example: Simplify the boolean function:
A′B′CD+A′BC′D′+ABC′D+ABCD+ABC′D′+A′B′C′D+AB′CD′

AB/CD	00	01	11	10
00		1	1	
01	1			
11	1	1		
10				1

Simplified Boolean Function: BC′D′+ABC′+A′B′D+ AB′CD′

Example: Simplify the boolean function: A′+AB′+ABC′

It is a non standard expression, so we need to convert it into standard expression.

A′BC+A′BC′+A′B′C+A′B′C′+AB′C+AB′C′+ABC′

Simplified expression = C + AB′ + A′B

A/BC	00	01	11	10
0		1	1	1
1	1	1	1	

Task: Simplify the boolean function: B′C′+AB′+ABC′+AB′CD′+A′B′C′D+AB′CD

2.9 K-Map POS Minimization

After POS expression is mapped, there are three steps in obtaining a minimum POS expression.

1) Grouping of 0′s.

2) Determine the sum term of each group.

3) Multiplying the resulting sum terms.

For a standard POS expression, 0 is placed on k-map for each sum term in the expression.

(A+B+C)(A+B′+C)(A′+B′+C)(A′+B+C)

Task: Simplify the Boolean function: (A+B+C)(A+B+C′)(A+B′+C)(A+B′+C′)(A′+B′+C)

Task: Simplify the Boolean function:
(B+C+D)(A+B+C′+D)(A′+B′+C+D′)(A+B′+C+D)(A′+B′+C+D)

Task: Simplify the Boolean function:
(A′+B′+C+D)(A+B′+C+D)(A+B+C+D′)(A+B+C′+D′)(A′+B+C+D′)(A+B+C′+D)

5-variable K-map-Example

Example: Simplify the boolean function: A′B′C′D′E′+A′B′CD′E′+A′BCD′E′+ A′BC′D′E′+A′B′C′D′E+A′BCD′E + A′BCDE + AB′C′D′E′ + AB′C′D′E + ABCD′E + AB′CDE+ABCDE

For A = 0 (A′B′C′D′E′, A′B′CD′E′, A′BCD′E′, A′BC′D′E′, A′B′C′D′E, A′BCD′E, A′BCDE

BC/DE	00	01	11	10
00	1	1		
01	1			
11		1	1	
10	1			

For A = 1

Example: Simplify the boolean function: F(A,B,C,D,E)={0,2,4,6,9,13,21,23,25,29,31}.

The k-map method of simplification is convenient as long as the variables does not exceed five or six. As the number of variables increases, the excessive number of squares prevents a reasonable selection of adjacent squares. The k-map method relies on the ability of human user to recognize certain patterns. For functions of five or more variables, it is difficult to be sure that the best solution has been made.

The tabulation method overcomes this difficulty. It is a step by step procedure that is guaranteed to produce a simplified standard form expression for a function.It can be applied to problems with many variables.It is also known as Quine- McCluskey method.

The tabulation method

This simplification method consists of two parts:

A) The first is to find by an exhaustive search all the terms that are candidates for inclusion in the simplified function. These terms are known as prime implicants.

B) The second operation is to choose among the prime implicants those that give an expression with the least number of literals.

2.10 Determination of Prime Implicants

1) Write down the list of minterms that specify the function.

2) Compare each minterm with every other minterm. If two minterms differ in only one variable, that variable is removed and a term with one less literal is found.

3) This process is repeated for every minterm until the exhaustive search is completed.

4) The matching process cycle is repeated for those new terms just found.

5) Further cycles are continued until a single pass through a cycle yields no further elimination of literals. The remaining terms and all the terms that does not match during the process comprise the prime implicants.

Example: Q: Simplify the following Boolean function by using the tabulation method: F=Σ(0,1,2,8,10,11,14,15)

	w	x	y	z	
0	0	0	0	0	✔
1	0	0	0	1	✔
2	0	0	1	0	✔
8	1	0	0	0	✔
10	1	0	1	0	✔
11	1	0	1	1	✔
14	1	1	1	0	✔
15	1	1	1	1	✔

	w	x	y	z	
0, 1	0	0	0	-	
0, 2	0	0	-	0	✔
0, 8	-	0	0	0	✔
2, 10	-	0	1	0	✔
8, 10	1	0	-	0	✔
10, 11	1	0	1	0	✔
10, 14	1	-	1	-	✔
11, 15	1	-	1	1	✔

14, 15	1	1	1	-	✓

	w	x	y	z	
0, 2, 8, 10	-	0	-	0	
0, 8, 2, 10	-	0	-	0	
10, 11, 14, 15	1	-	1	-	
10, 14, 11, 15	1	-	1	-	

Summary

- Boolean algebra holds two values, i.e., 0 and 1.
- Binary numbers are used to represent 0s and 1s information in the computers.
- The various laws and postulates associated with this are used to solve various Boolean functions.
- De-Morgan's theorems are used to reduce the Boolean functions.
- K-Map is also used to reduce the Boolean functions.
- K-Map SOP and POS minimization methods are used to find out the minimal forms.
- Tabulation method is also used for finding out the minimal forms.
- Truth table is also one way of the way for simplification of Boolean expressions.

Keywords

Boolean algebra: It is the branch of algebra in which the values of the variables are the truth values, i.e., true and false, usually denoted by 1 and 0 respectively.

Self Assessment

Q 1: Which term belongs to Idempotent law?

A. A.A=A

B. (B+C) =A.B+A.C

C. (A')'=A

D. A+B=B+A

Q 2: A(A+B)=A shows ____________ law.

A. Idempotent

B. Annulment

C. Distributive

D. Absorptive

Q 3: For evaluation of Boolean expressions, what will be order of?

AND

OR

NOT

Parentheses

A. 1,2,3,4

B. 2,1,4,3

C. 4,3,1,2

D. None of the above

Q 4: According to De-Morgan's theorem (A+B)′ = A′.B′, if A=1, B=0, then what is the output?

A. 1

B. 0

C. Either 1 or 0

D. None of the above

Q 5: In which form, a single variable is allowed?

A. SOP

B. POS

C. Both SOP and POS

D. None of the above

Q 6: In a Boolean function, if there are *n* variables, then the number of combinations of 1s and 0s will be

A. 2^n

B. 2^{n+1}

C. 2^{n-1}

D. $2^{n+1}-2$

Q 7: In 3 variable k-maps, which of the following is not adjacent to 011?

A. 101

B. 001

C. 111

D. 010

Q 8: In 4 variable k-maps, which of the following is not adjacent to 0010?

A. 0000

B. 0011

C. 0111

D. 0110

Q 9: After mapping of POS expression, for minimization we group together

A. 0

B. 1

C. Both 0 and 1

D. Either 0 or 1

Q 10: After mapping of SOP expression, for minimization we group together

A. 0

B. 1

C. Both 0 and 1

D. Either 0 or 1

Q 11: What are the steps of simplification in tabulation method?

A. Search for prime implicants

B. Choosing of prime implicants for expression with least number of literals

C. Both searching of prime implicants and choosing of them for expression

D. None of the above

Q 12: The _______ can be minimized.

A. POS

B. SOP

C. Both POS and SOP

D. None of the above

Q 13: After mapping of POS expression, for minimization we group together

A. 0

B. 1

C. Both 0 and 1

D. Either 0 or 1

Q 14: Simplify the Boolean function, A′+AB′+ABC′

A. A+B+C

B. A′+B′+C′

C. A′+B+C

D. A′+B′+C

Q 15: According to De-Morgan's theorem (A+B)′ = A′.B′, if A=1, B=0, then what is the output?

A. 1

B. 0

C. Either 1 or 0

D. None of the above

Answers for Self Assessment

1.	A	2.	D	3.	C	4.	B	5.	C
6.	A	7.	A	8.	C	9.	A	10.	B
11.	C	12.	C	13.	A	14.	B	15.	B

Review Questions

Q 1: What is a Boolean function? Write its laws and postulates.

Q 2: Simplify the given 5-variable Boolean equation by using k-map.

f (A, B, C, D, E) = ∑ m (0, 5, 6, 8, 9, 10, 11, 16, 20, 42, 25, 26, 27).

Q 3: Minimize the following Boolean function using sum of products (SOP):

f(a,b,c,d) = ∑m(3,7,11,12,13,14,15) .

Q 4: Explain how to find out the prime implicants using tabulation method.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education Asia, 2002.

Web Links

https://web.iit.edu/sites/web/files/departments/academic-affairs/academic-resource-center/pdfs/kmaps.pdf

https://www.geeksforgeeks.org/5-variable-k-map-in-digital-logic/

Unit 03: Implementation of Combinational Logic Design

CONTENTS

Objectives

After studying this unit, you will be able to

- Understand various logic gates.
- Understand various types of combinational circuits and their implementation.
- Know the functions of various combinational circuits.
- Understand the logic symbols of adders, subtractors, encoders, decoders, multiplexers and de-multiplexers.

Introduction

Logic gates are the building blocks of any digital computer. Logic gates are electronic circuit having one or more than one input and only one output. The relationship between the input and the output is based on certain logic. The inputs any gate can have either 0 or 1. Based upon the concept of logic gate, we get the output also 0 or 1.

3.1 Types of Logic Gates

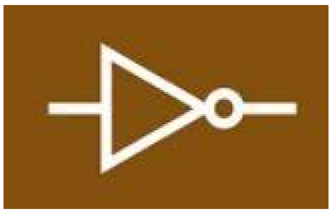
There are various types of logic gates and every logic gate has its own functionalities and working. These types are:

- NOT gate
- OR gate
- AND gate
- NOR gate

- NAND gate
- XOR gate
- XNOR gate

NOT gate

The logic symbol for NOT gate is

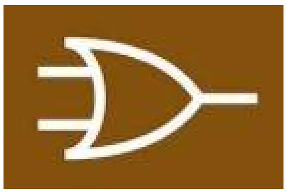


The expression for NOT gate is Output = (A)′. Here A is designated as the input provided

A	OUTPUT
0	1
1	0

OR gate

The logic symbol for OR gate is

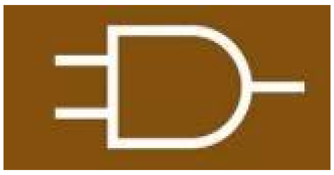


The expression for OR gate is OUTPUT = (A+B). Here A and B are inputs provided.

A	B	OUTPUT
0	0	0
0	1	1
1	0	1
1	1	1

AND gate

The logic symbol for AND gate is

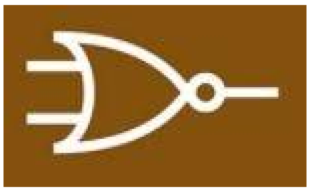


The expression for AND gate is OUTPUT = (A * B). Here A and B are inputs provided.

A	B	OUTPUT
0	0	0
0	1	0
1	0	0
1	1	1

NOR gate

The logic symbol for NOR gate is

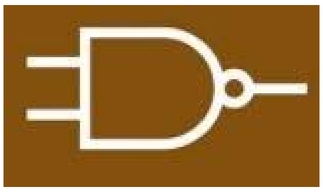


The expression for NOR gate is OUTPUT = (A + B)′.

A	B	(A+B)	OUTPUT
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

NAND gate

The logic symbol for NAND gate is

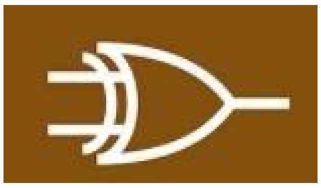


The expression for NAND gate is OUTPUT = (A * B)′. Here A and B are the inputs provided.

A	B	(A.B)	OUTPUT
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

XOR gate

The logic symbol for XOR gate is

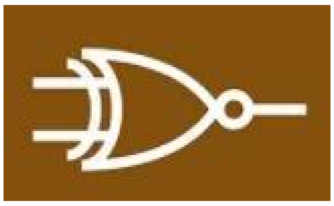


The expression for XOR gate is OUTPUT = A * B′ + A′ * B. Here A and B are the inputs provided.

A	B	A.B’	A’.B	OUTPUT
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	0	0	0

XNOR gate

The logic symbol for XNOR gate is



The expression for XNOR gate is OUTPUT = A * B + A′ * B′. Here A and B are the inputs provided to the gate.

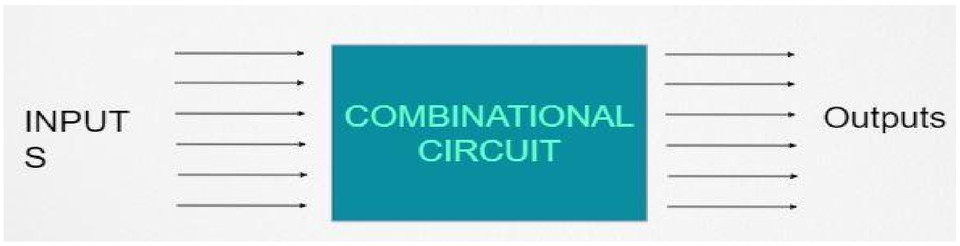
A	B	A.B	A'.B'	OUTPUT
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

Combinational logic:

The logic circuits for digital system can be either sequential or combinational. A sequential circuit employs storage elements in addition to logic gates. Their output area is a function of the inputs and the state of storage elements. The state of storage elements, in turn, is a function of previous inputs.

A combinational circuit consists of logic gates whose output is determined from the present combination of inputs. A combinational circuit consists of input variables, logic gates and output variables. The logic gates accept signals from the inputs and generate signals to the outputs. This process transforms binary information from the given input data to a required output data.

3.2 Block Diagram of Combinational Logic



The n input binary variables come from an external source and the m output variables go to an external destination.Each input and output variable exists physically as a binary signal that represents logic 1 and logic 0.For n input variables, there are 2^n possible binary input combinations. For each input combination there is one possible output value.

Types of combinational circuits

Adders

Subtractors

Encoders

Decoders

Multiplexers

De-multiplexers

3.3 Adders and Subtractors

Digital computers perform a variety of information processing tasks. Among all these, the most basic arithmetic operation is the addition of two binary digits.

The simple addition consists of four possible elementary operations:

A	B	C
0	0	0
0	1	1
1	0	1
1	1	10

The first three operations produce a sum of one digit, but when augend and addend bits are equal to 1, the binary sum consists of 2 digits. The higher significant bit of this result is a CARRY. When the augend and addend number contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits.

A combinational circuit that performs the addition of two bits is called a half adder. A combinational circuit that performs the addition of three bits (two significant bits and one previous carry) is a full adder. (Two half adders can be employed to make the full adder). The half adder design is carried out first from which we will develop the full adder. Connecting n full adders in cascade produce the binary adder for two n-bit numbers.

Half adder

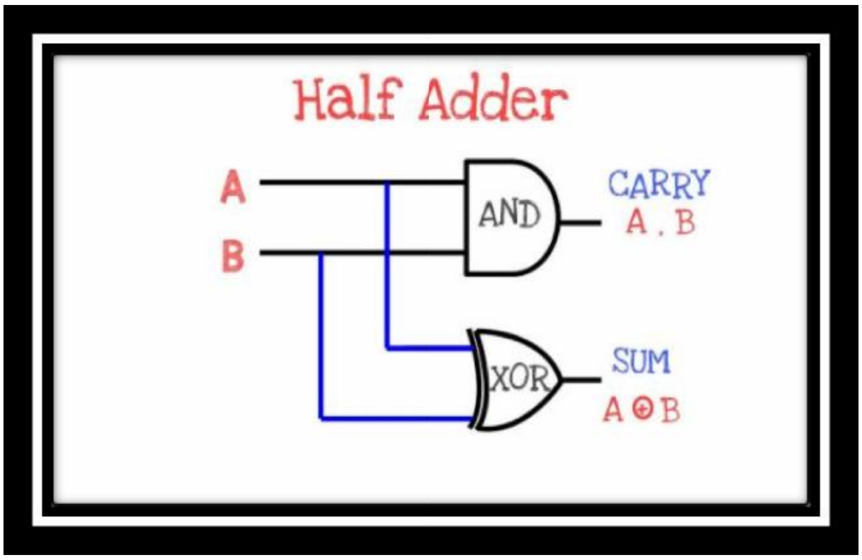
The half adder needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits (x and y). The output variables produce the sum and carry (S and C).

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

C is 1 only when both of the inputs are 1. The S output represents the least significant bit of the sum. The Boolean expression for this:

$S = X' * Y + X * Y'$ and $C = X * Y$.

The combinational circuit for half adder is



Full Adder

A full adder is the combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs. Inputs (Two of the input variables denoted by x and y), represent the two significant bits to be added and third input, z represents the carry from the previous lower significant positions) and the outputs (S and C; the binary variable S gives the value of the least significant bit of the sum and the binary variable C given the output carry).

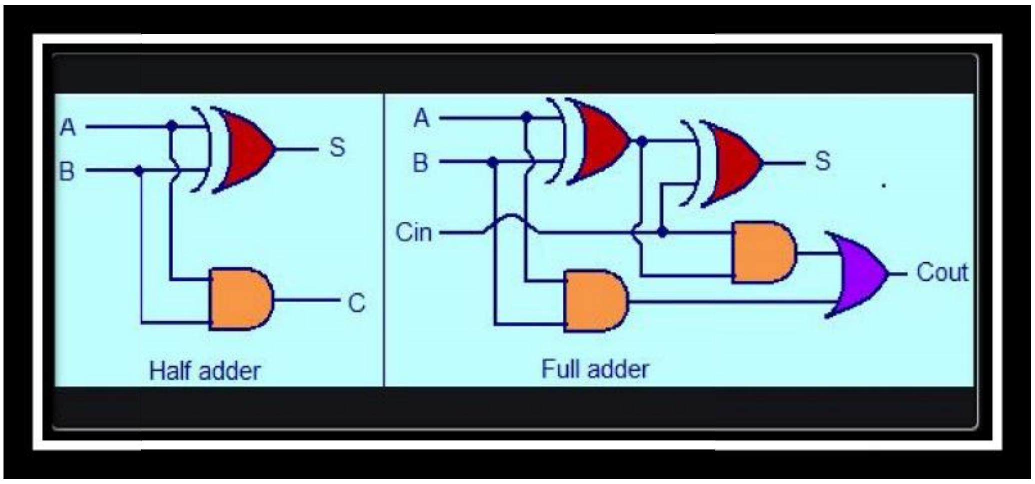
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The Boolean expressions for full adder are

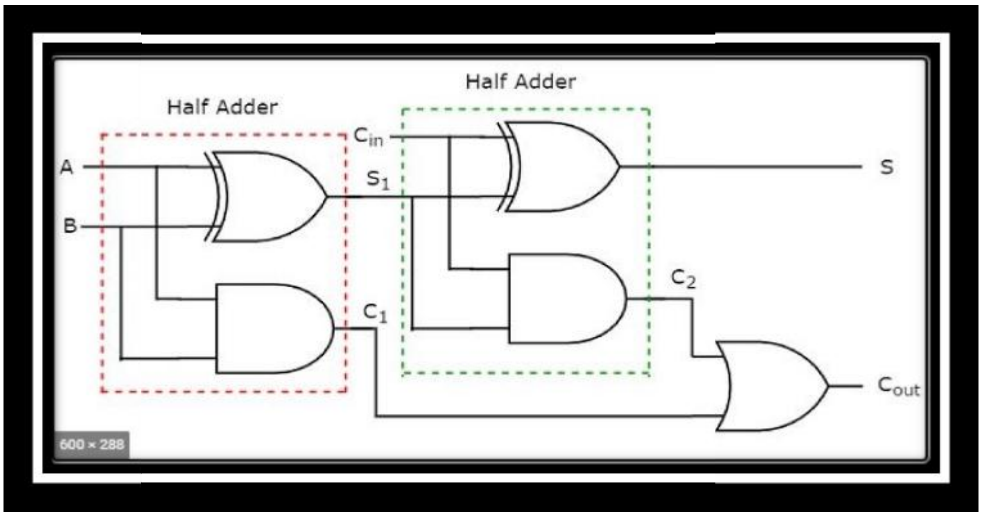
$S=x'y'z+x'yz'+xy'z'+xyz$

$C=xy+xz+yz$

The combinational circuit for full adder is



The full adder is made up of serial connection of two half adders as shown in the picture.



Subtractors

The subtraction A-B can be done by taking the 2's complement of B and adding it to A.The 2's complement can be obtained by taking the 1's complement and adding 1 to the LSB pair of bits. The 1's complement can be implemented with inverters and a one can be added to the sum through the input carry. So the circuit for A-B consists of an adder with inverters placed with each data input B.The initial input carry must be equal to 1 when performing subtraction.So, the operation becomes A, plus the 1's complement of B, plus 1 which is equal to A plus the 2's complement of B.

The initial input carry must be equal to 1 when performing subtraction. So, the operation becomes A, plus the 1's complement of B, plus 1 which is equal to A plus the 2's complement of B.

Overflow

For unsigned numbers, this gives A-B if A>=B or 2's complement of (B-A) if A<B.For signed numbers, the result is A-B, provided that there is no overflow. When two numbers of n digits each are added and the sum occupies n+1 digits, then the overflow occurs.Overflow is a problem in digital computers because the number of bits that hold the number is finite and the result that contains n+1 bits cannot be accommodated.The overflow is detected after the addition of two binary numbers depends upon whether the numbers are considered to be signed or unsigned. When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position. In the case of signed numbers, the left most bit always represents the sign.

An overflow cannot occur after the addition if one number is positive and the other number is negative. It can occur if the two numbers added are either positive or both negative.

3.4 Decimal Adder

Computers or calculators that perform arithmetic operations directly in the decimal number system represent the decimal numbers in binary coded form. An adder for such a computer must employ an arithmetic circuit that accepts coded decimal numbers and present results in the same code.For binary addition it is sufficient to consider a pair of significant bits together with a previous carry.A decimal adder requires a minimum of 9 inputs and 5 outputs. Since 4 bits are required to code each decimal digit and the circuits must have an input and output carry.Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from previous stage. Since each input digits does not exceed 9, the output sum cannot be greater than (9+9+1=19).Suppose we apply two BCD digits to a 4-bit binary adder. The adder will form the sum in binary and produces the results that range from 0 to 19.

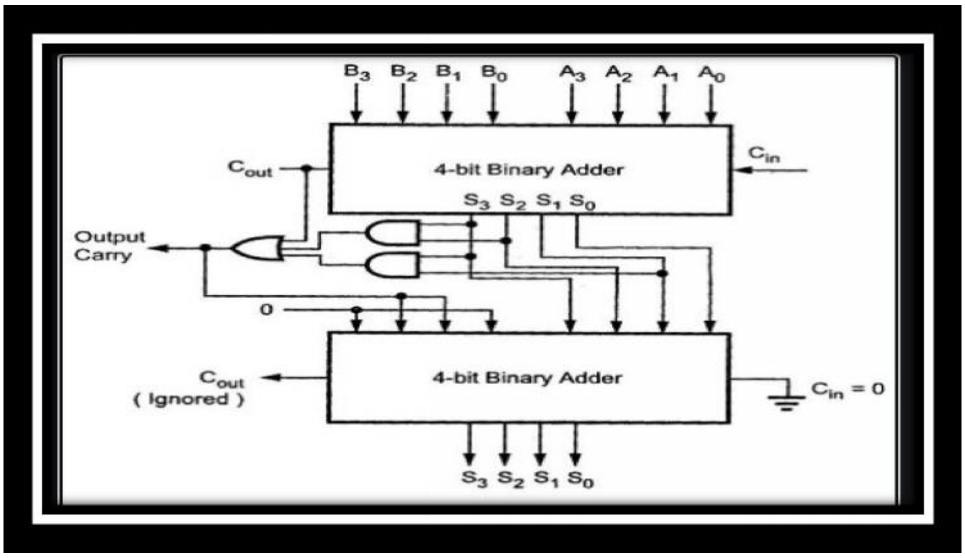
Binary Sum					BCD Sum					Decimal
K	Z8	Z4	Z2	Z1	C	S8	S4	S2	S1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8

0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

K is the carry and the subscripts under the letter Z represents the weights 8,4,2 and 1 that can be assigned to four bits in BCD code.The columns under the binary sum list the binary value that appears in the output of 4-bit binary adder.The problem is to find a rule by which the binary sum is to be converted to the correct BCD digit representation of the number in BCD sum.It is apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical and therefore no conversion is required. When the binary sum is greater than 1001, we obtain a non valid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.It is obvious that the correction is needed when the binary sum has an output carry K=1. The other 6 combinations from 1010 through 1111 that need a correction have a 1 in position Z8. To distinguish from binary 1000 and 1001, which also have a 1 in8, we move further to either Z4 or Z2 which must have a 1.

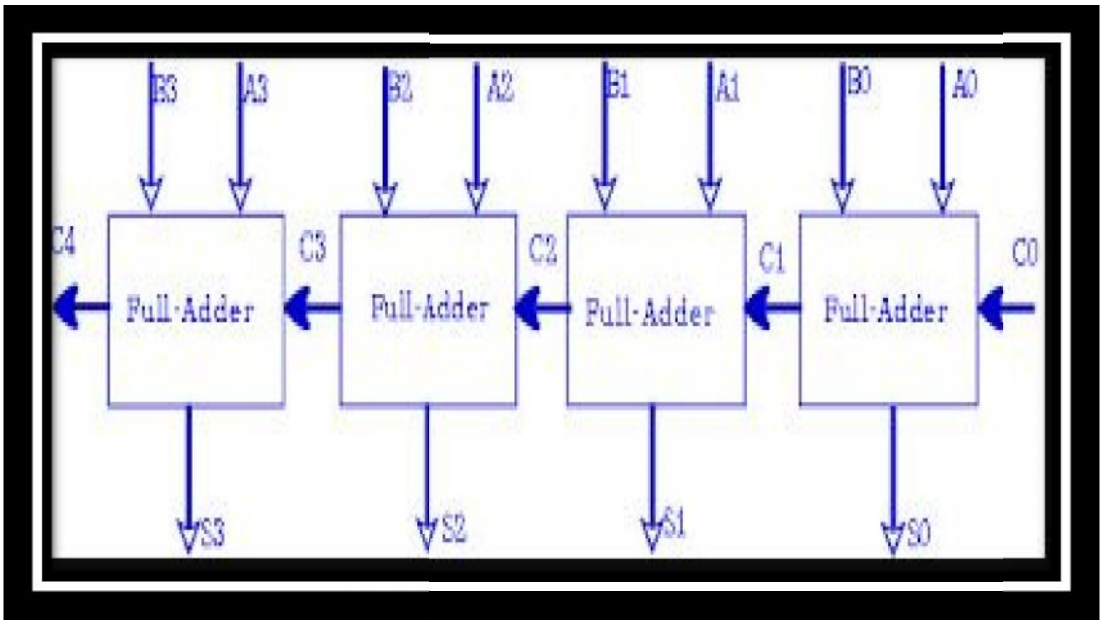
C=K+Z8*Z4+Z8*Z2

The combinational circuit for BCD adder is



3.5 Binary Parallel Adder

A binary parallel adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with connecting full adders; with the output carry from each full adder is connected to the input carry of next full adder in the chain.



Here the augend bits of A and addend bits of B are designated by the subscript numbers from right to left with subscript 0 denoting LSB. The carries are connected in a chain through the full adder. The input carry to the adder is C0 and it ripples through the full adders to the output carry C4.The S outputs generate the required sum bits. A n-bit adder requires n full adders with each output carry connected to the input carry of the next higher order full adder.

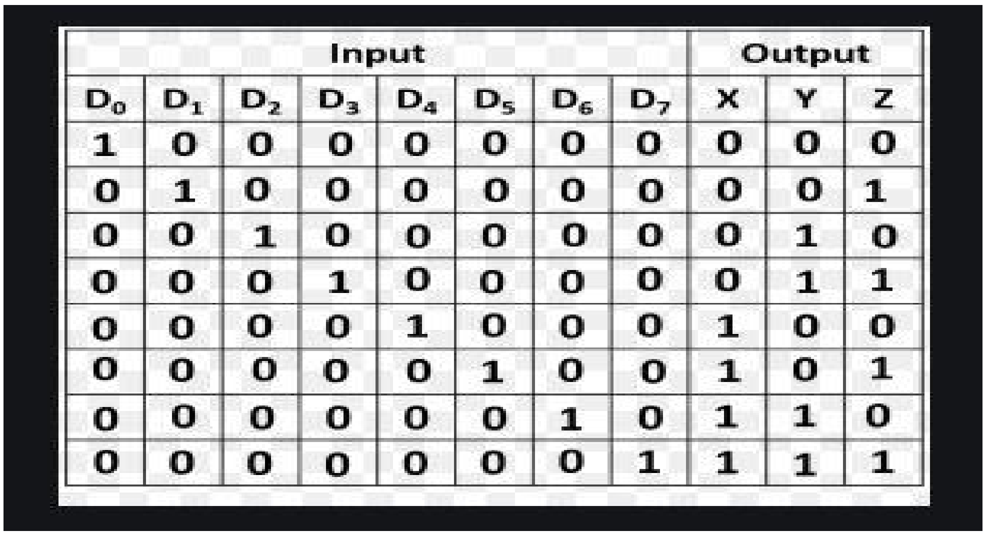
Carry propagation

The addition of two binary numbers in parallel implies that all the bits of augend and addend are available for computation at the same time. As in any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals.The total propagation time is equal to the propagation delay of a gate multiplied by the number of gate levels in the circuit. The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders.

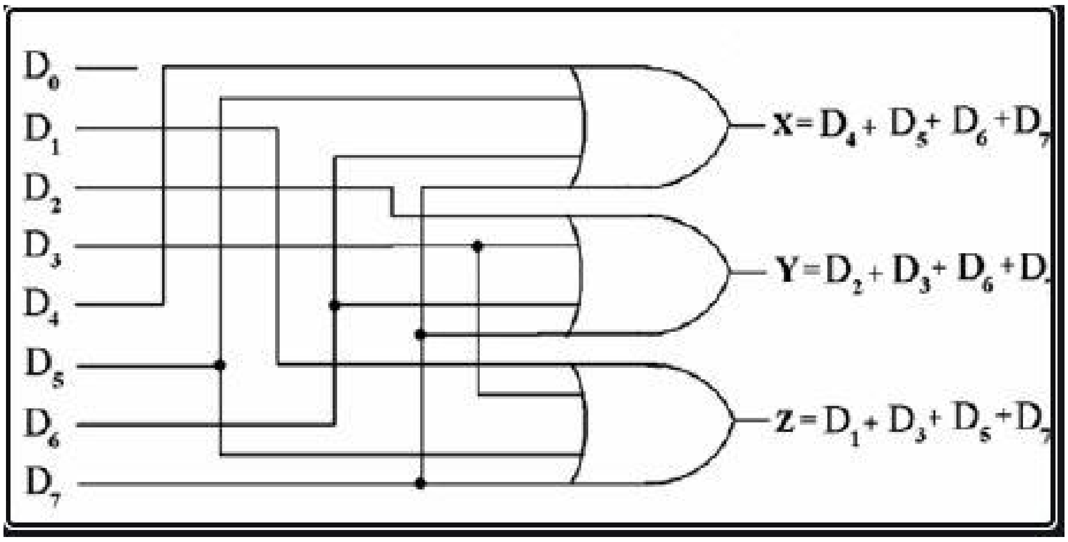
3.6 Encoders and Decoders

An encoder has 2^n input lines and n output lines.Example: octal to binary encoder. It has eight inputs (one for each of the octal digits) and three outputs that generate the corresponding binary number. The output lines generate the binary code corresponding to the input value.It is assumed that only one input has a value of 1 at any given time.

Octal to Binary Encoder



Input								Output		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1


D_0
D_1
D_2
D_3
D_4
D_5
D_6
D_7
$X = D_4 + D_5 + D_6 + D_7$
$Y = D_2 + D_3 + D_6 + D_7$
$Z = D_1 + D_3 + D_5 + D_7$

The encoder can be implemented with OR gates whose inputs are determined directly from the truth table.

Output z is equal to 1 when the input octal digit is 1,3,5 or 7. $Z= D_1+ D_3+ D_5 +D_7$

Output y is equal to 1 when the input octal digit is 2,3,6 or 7. $Y= D_2+D_3+D_6+D_7$

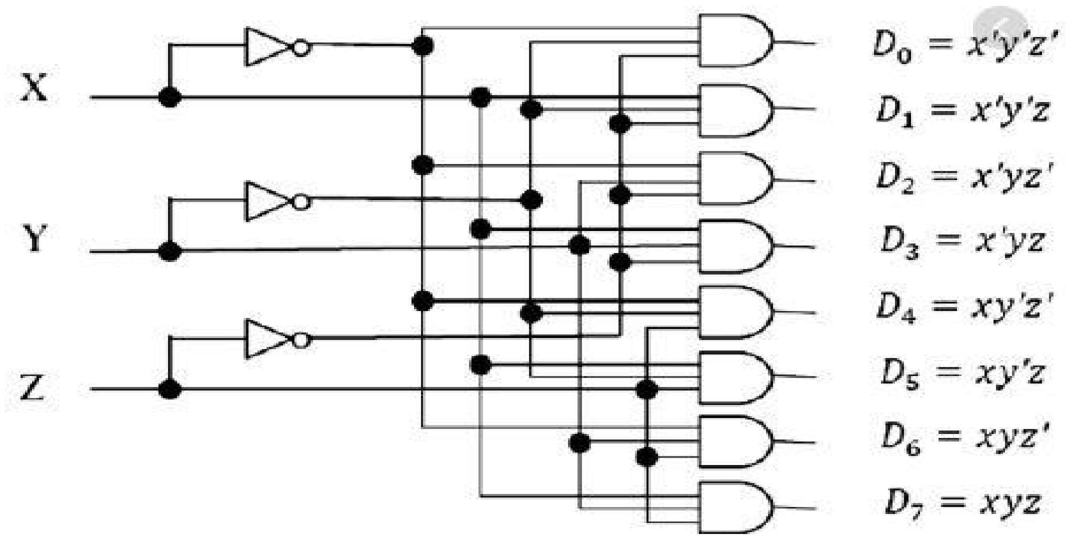
Output x is equal to 1 when the input octal digit is 4,5,6 or 7. $X= D_4+D_5+D_6+D_7$

The encoder has a limitation that only one input can be active at any given time.

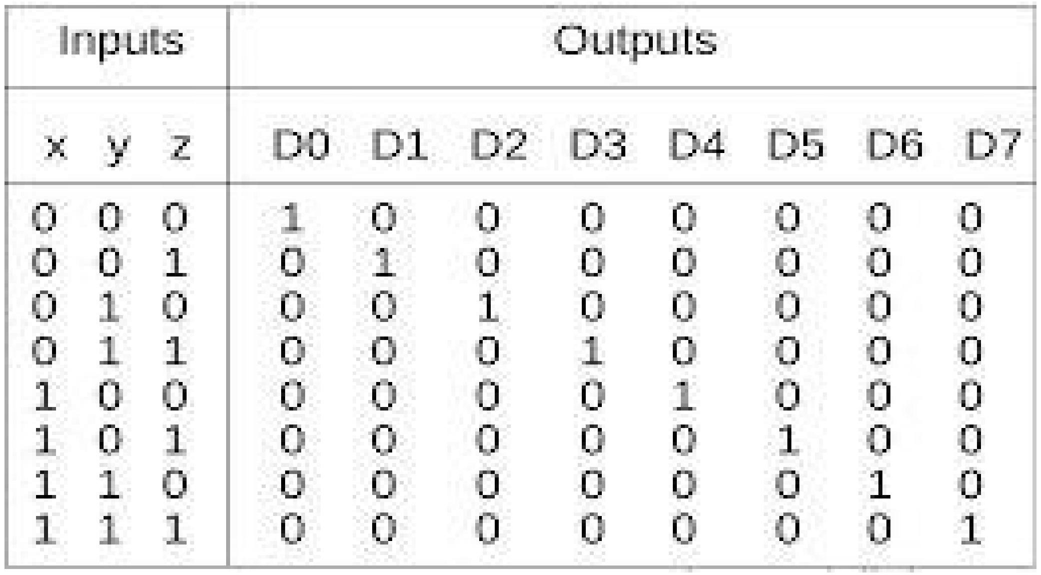
If two inputs are active simultaneously, the output produces an undefined combination. For example, if D3 and D6 are 1 simultaneously, the output of the encoder will be 111 because all three outputs are equal to 1. This does not represent either binary 3 or binary 6. To resolve this ambiguity, encoder circuits must establish an input priority to ensure that only one input is encoded.If we establish a higher priority for inputs with higher subscript numbers and if both D3 and D6 are 1 at the same time, the output will be 110 because D6 has higher priority than D3.

Decoders are opposite of encoders.Discrete quantities of information are represented in digital systems by binary codes. A binary code of n bits is capable of representing 2^n distinct elements of coded information. A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n-bit coded information has unused combinations, the decoder may have fewer than 2^n outputs.

3-to-8 line decoder



This decoder is known as n-to-m line decoder, where m<=2^n. The purpose is to generate the 2^n or fewer min-terms of n input variables. Here 3 inputs are decoded into 8 outputs, each representing one of the min-terms of the three input variables.Here the three inverters provide the complement of the inputs and each one of the eight AND gates generate one of the min-terms. The application of this is to decode any 3-bit code to provide eight outputs, one for each element of the code.



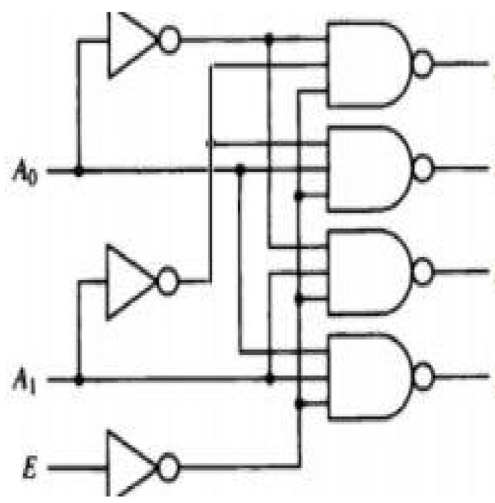
Inputs			Outputs							
x	y	z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

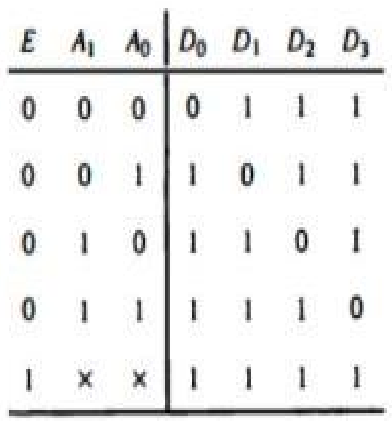
For each possible input combination, there are 7 outputs that are equal to 0 and only one that is equal to 1. The output whose value is 1 represents the min-term equivalent of the binary number presently available in the input lines.

Decoder with NAND gate

Since a NAND gate produces the AND operation with an inverted output, it becomes more economical to generate the decoder min-terms in their complemented form. Decoder includes one or more enable inputs to control the circuit operation.

3.7 2-to-4 lineDecoder with Enable input



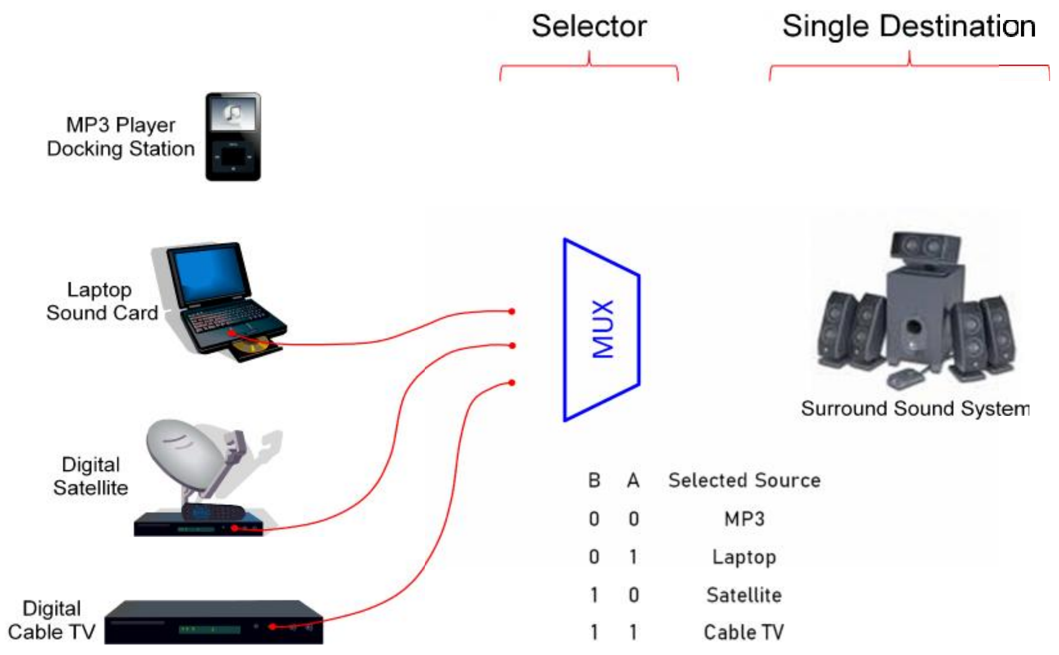


E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	×	×	1	1	1	1

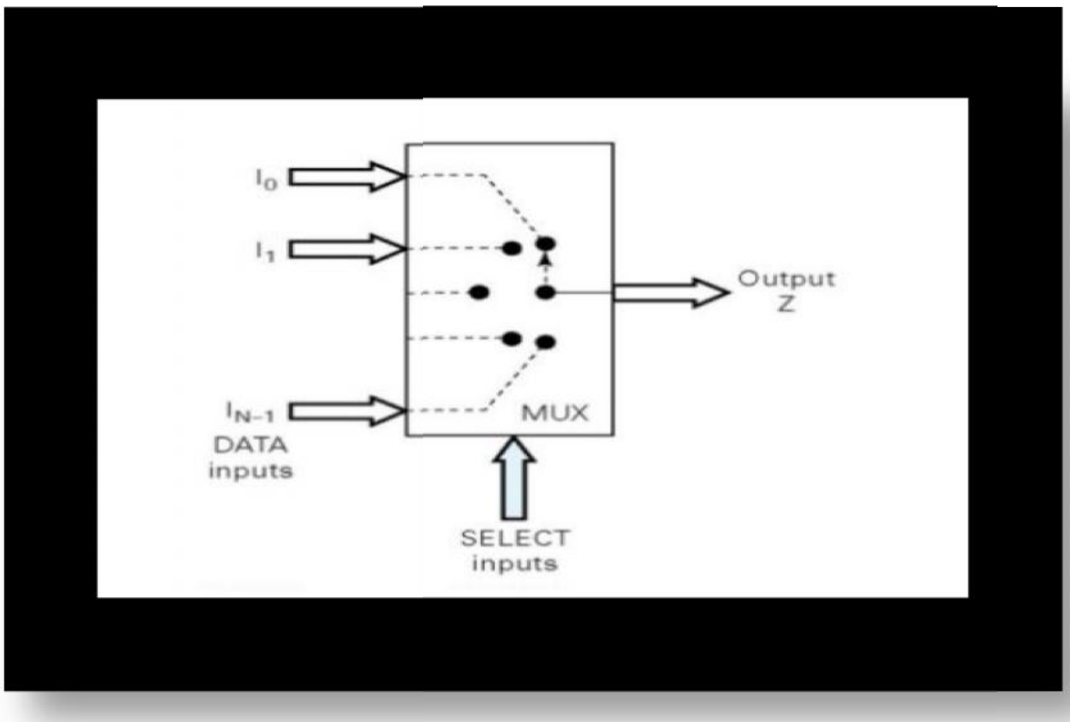
The decoder is enabled when E=0, regardless of the value of 2 other inputs.Only one output can be equal to 0 at any given time, all other outputs are equal to 1. The output whose value is equal to 0 represents the min-term selected by inputs A and B. The circuit is disabled when E=1, regardless of the values of other two inputs. When the circuit is disabled, none of the outputs are equal to 0 and none of the min-terms are selected. A decoder with enable input can function as a demultiplexer. It is a circuit that receives information from a single line and directs it to one of the 2^n possible output lines. The selection of a specific output is controlled by the bit combination of n selection lines.

Multiplexers and De-multiplexers

A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.The multiplexer is also known as the data selector



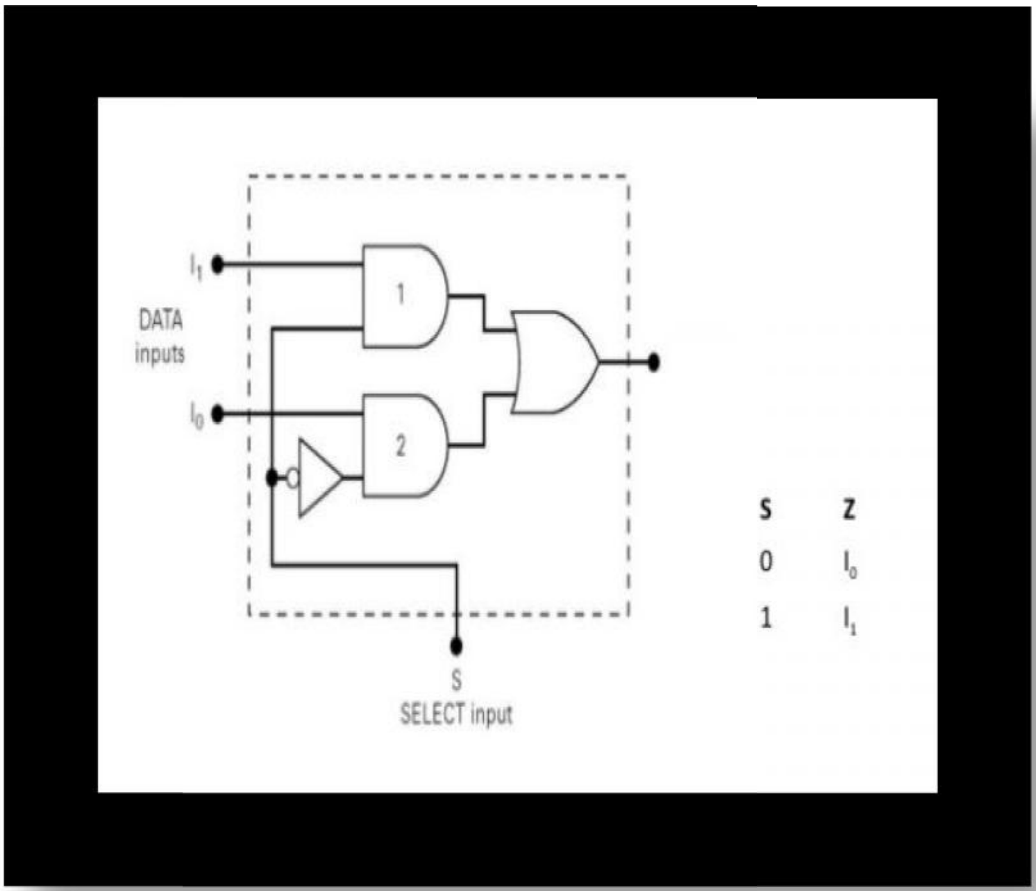
B	A	Selected Source
0	0	MP3
0	1	Laptop
1	0	Satellite
1	1	Cable TV



Types of multiplexers

- 2–to-1 line multiplexer.
- 4-to-1 line multiplexer.

3.8 2-to-1 lineMultiplexer



A 2-to-1 line multiplexer connects one of two 1-bit sources to a common destination. The circuit has two input lines, one output line and one selection line. When S=0, the upper AND gate is enabled and I1 has a path to the output.When S=1, the lower AND gate is enabled and I0 has a path to the output.The multiplexer acts like an electronic switch, which selects one out of two sources.

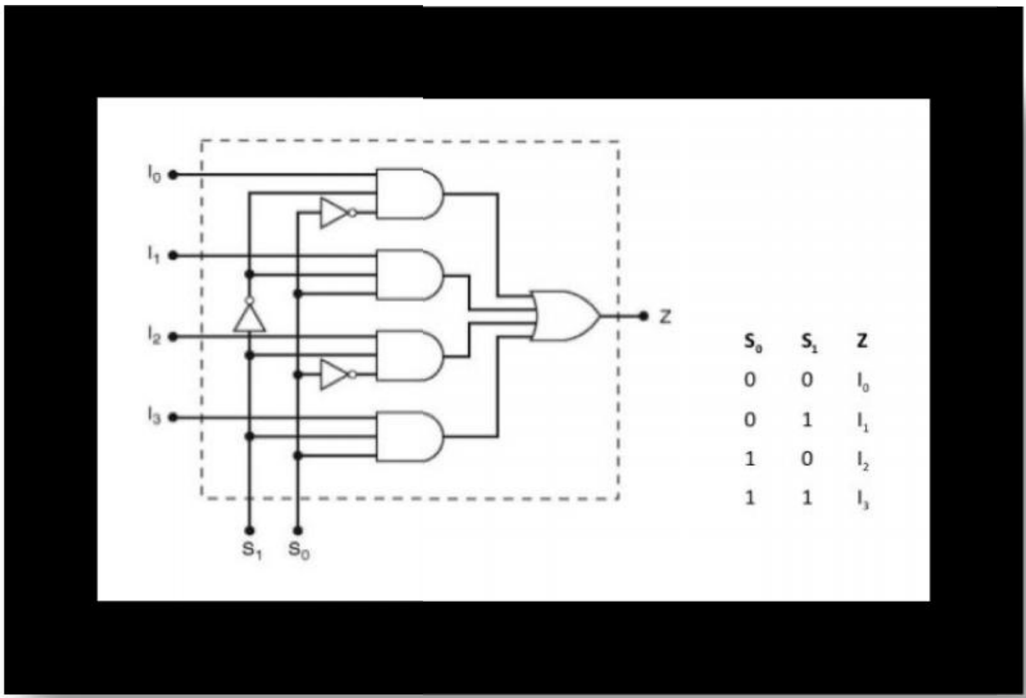
S	Z
0	I0
1	I1

Output=i0 when S=0, Y=i0.S′,

Output=i1 when S=1, Y=i1.S.

Y=i0.S′+i1.S.

3.9 4-to-1 Line Multiplexer



S_0	S_1	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Each of the four inputs, I0 through I3, is applied to one input of AND gate.Selection lines S0 and S1 are decoded to select a particular AND gate. The outputs of AND gates are applied to single OR gate that provides the 1-line output.Example: S1S0=10: The AND gate associated with input I2 has two of its inputs equal to 1 and third input connected to I2. The other three AND gates have at least one input equal to 0, which makes their output equal to 0. The OR gate output is equal to the value of I2, providing a path from the selected input to the output.It is working as a data selector since it selects one of many inputs and steers the binary information to the output line.

S0	S1	Y
0	0	I0
0	1	I1
1	0	I2
1	1	i3

O/P is i0 only if S0=0 and S1=0, Y=i0*s0′*s1′.

O/P is i1 only if S0=0 and S1=1, Y=i1*s0′*s1.

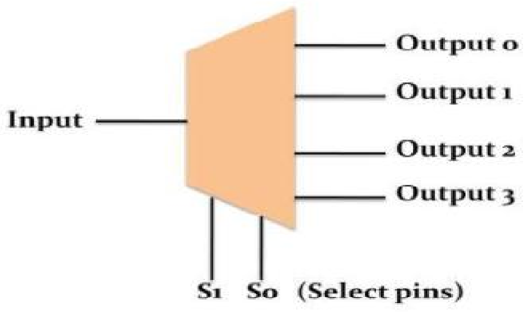
O/P is i2 only if S0=1 and S1=0, Y=i2*s0*s1′.

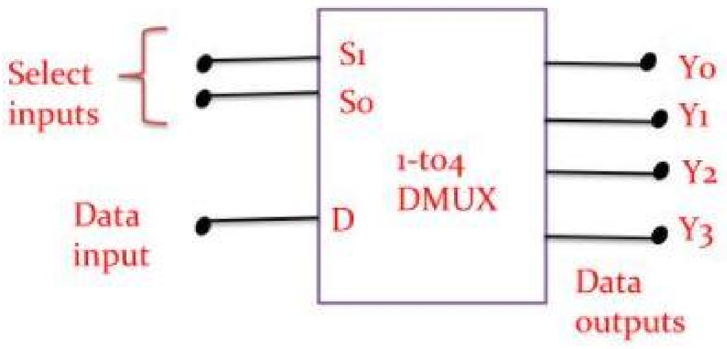
O/P is i3 only if S0=1 and S1=1, Y=i3*s0*s1.

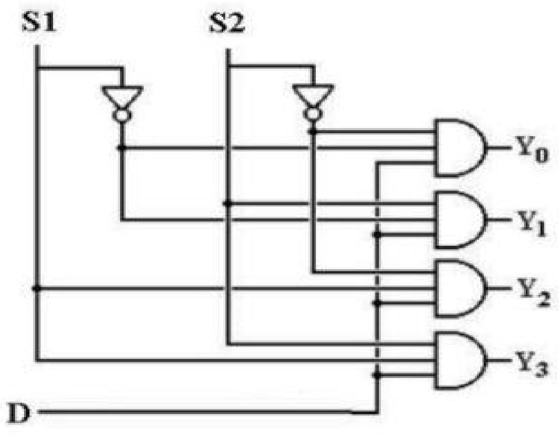
Y=i0*s0′*s1′ + i1*s0′*s1 + i2*s0*s1′ + i3*s0*s1.

De-multiplexer

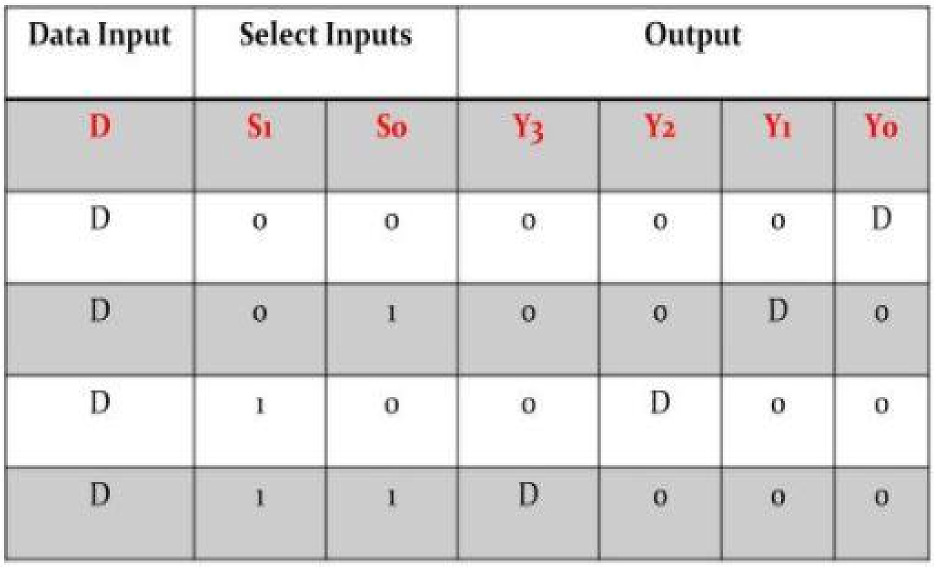
It reverses the multiplexing function.The word de-multiplex means one into many. It takes digital information from one line and distributes it to a given number of output lines. It is the process of taking information from one input and transmitting the same over one of the several outputs.







A 1-to-4 de-multiplexer has a single input, four outputs and two select lines.



Data Input	Select Inputs		Output			
D	S_1	S_0	Y_3	Y_2	Y_1	Y_0
D	0	0	0	0	0	D
D	0	1	0	0	D	0
D	1	0	0	D	0	0
D	1	1	D	0	0	0

From the truth table it is clear that the data input is connected to output Y0 when S1=0 and S0=0 and the data input is connected to output Y1 when S1=0 ad S1.Similarly, the data input is connected to outputs Y2 and Y3 when S1=1 and S0=0 and when S1=1 and S0=1 respectively.

Y0=S1'S0'D,

Y1=S1'S0D,

Y2=S1S0'D,

Y3=S1S0D.

Now using these expressions, a 1-to-4 line De-multiplexer can be implemented using four 3 input AND gates and two NOT gates.Here, the input data is connected to all the AND gates.

Applications of De-Multiplexer

- Communication System
- ALU
- Serial to parallel converter

Summary

- Logic gates are the building blocks of all the combinational circuits.
- Every combinational circuit requires the implementation.
- Adders are required for the implementation of addition mathematical operation. Various adders are available.
- Encoders are required to encode the inputs. The most famous encoder is octal to binary encoder. This one encodes the eight bits input to three bits output.
- Decoders do the reverse function of encoders.
- The multiplexers takes many input lines

Keywords

OR gate: This gate performs the operation of addition.

AND gate: This gate performs the operation of multiplication.

NOT gate: This gate inverts the input.

NAND gate: This gate performs the inverse of multiplication of inputs.

NOR gate: This gate performs the inverse of addition of inputs.

Self Assessment

1.If A=0, B=1 and C=0, what will be the output in (A OR B) AND C?

A. 0

B. 1

C. 2

D. None of the above

2.If X=1,Y=1 and Z=1, what will be the output in (X AND Y) XOR Z?

A. 0

B. 1

C. 2

D. None of the above

3.What will be the output, if two inputs, R=0 and S=0 are applied to XNOR logic.

A. 0

B. 1

C. 2

D. None of the above

4.A combinational circuit that performs the addition of three bits is known as.

A. Full Adder

B. Half Adder

5.Which combinational circuit is represented by these Boolean expressions S=x'y+xy', C=xy?

A. Full Subtractors

B. Half Adder

C. Full Adder

D. None of the above

6.When an overflow occurs, the sign is represented by ________ most bit in signed numbers.

A. Left

B. Right

C. Middle

D. None of the above

7.In case of full adder, if X=1, Y=1 and Z=1, then what will be the values of C and S?

A. 0, 0

B. 1, 1

C. 0, 1

D. 1, 0

8.A decimal adder requires a minimum of _____ inputs and ______ outputs.

A. 9,9

B. 5,9

C. 9,5

D. 5,5

9.If we apply two BCD digits to a 4-bit binary adder. The adder will form the sum in binary and produces the result that ranges from___ to _____.

A. 0,9

B. 1,9

C. 1,13

D. 0,19

10. The addition of _________ to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

A. 0100

B. 0101

C. 0110

D. 0111

11. In Octal to Binary encoder, we have 8 inputs and 3 outputs. At a particular time, how many inputs can be one?

A. 1

B. 2

C. 4

D. 8

12. Which of the following combinational circuit has n inputs and 2^n outputs?

A. Adder

B. Subtractors

C. Encoder

D. Decoder

13. Which of the following combinational circuit distributes the information taken from one line to a given number of output lines?

A. Encoder

B. Decoder

C. Multiplexer

D. De-multiplexer

14. In a de-multiplexer, if we have only 1 input lines and 4 output lines, then how many selection lines will be there?

A. 0

B. 1

C. 2

D. 4

15. Which of the following combinational circuit has 2^n inputs and n outputs?

A. Adder

B. Subtractor

C. Encoder

D. Decoder

Answers for Self Assessment

1.	A	2.	A	3.	B	4.	A	5.	B
6.	A	7.	B	8.	C	9.	D	10.	C
11.	A	12.	D	13.	D	14.	C	15.	C

Review Questions

1. What are logic gates? Explain its functionalities, truth table and logic symbol.
2. What are adders? Explain half, full and decimal adders.
3. Explain encoders and decoders with their logic symbol and their functionalities.
4. Explain multiplexers, its use and variants.
5. Explain de-multiplexers, its use and variant.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education Asia, 2002.

Web Links

https://vardhaman.org/wp-content/uploads/2018/03/CAO%20Unit-I%20part-3.pdf

http://www.nou.ac.in/econtent/BCA%20Part%20I/Paper%207/BCA%20Paper-VII%20Block-2%20Unit-6.pdf

Unit 04: Design of Synchronous Sequential Circuits

CONTENTS

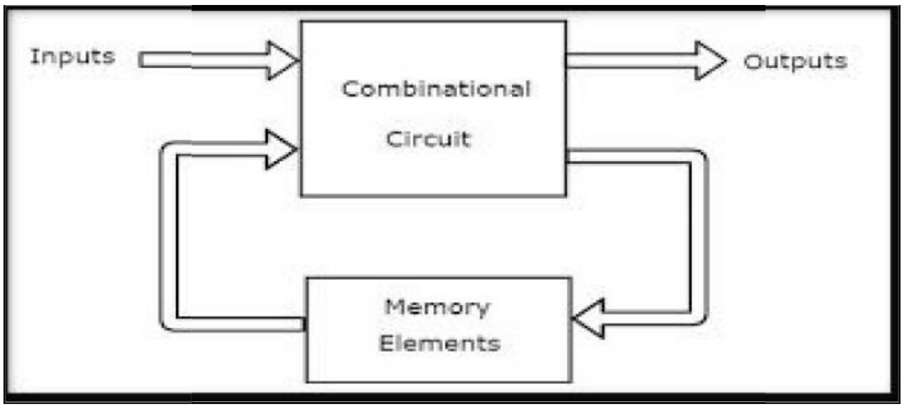
Objectives

After studying this unit, you will be able to:

- understand the sequential circuit
- understand the latches and different types of flip flops
- analyse the clocked sequential circuits
- understand state reduction and state assignment
- understand the design process of sequential circuit

Introduction

When the system requires the storage elements, then the system can be described in the terms of sequential circuits.The sequential circuit is made up of combinational circuit to which storage elements are connected to form a feedback path.The storage elements are devices capable of storing binary information. Sequential circuits are used to construct the finite state machines, which are basic building blocks in all digital circuitry. The binary information stored in these elements at any given time defines the state of the sequential circuit at that time. The sequential circuit receives binary information from external inputs. These inputs together with the present state of the storage elements determine the binary value of the outputs.They also determine the condition for changing the state in the storage elements.Thus, the outputs in the sequential circuit are a function not only of the inputs but also of the present state of the storage elements. The next state of the storage elements is also a function of external inputs and present state. Thus, a sequential circuit is specified by a time sequence of inputs, outputs and external states.



Flip-flop

A flip-flop is a binary storage device which can store one bit of information.A sequential circuit may use many flip flops to store as many bits as necessary. Examples of sequential circuits are:

Example: Flip flops, clocks, registers and counters

4.1 Types of Sequential circuits

There are two types of sequential circuits. These are:

- Asynchronous sequential circuits
- Synchronous sequential circuits

Asynchronous sequential circuits

This is a system whose outputs depend upon the order in which its input variables change and can be affected at any instant of time. Gate-type asynchronous systems are basically combinational circuits with feedback paths. Because of the feedback among logic gates, the system may, at times, become unstable. Consequently, they are not often used.

Synchronous sequential circuits

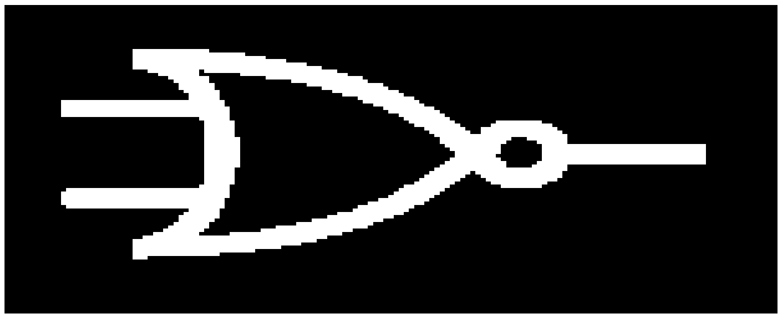
This type of system uses storage elements called flip-flops that are employed to change their binary value only at discrete instants of time. Synchronous sequential circuits use logic gates and flip-flop storage devices. Sequential circuits have a clock signal as one of their inputs. All state transitions in such circuits occur only when the clock value is either 0 or 1 or happen at the rising or falling edges of the clock depending on the type of memory elements used in the circuit. Synchronization is achieved by a timing device called a clock pulse generator. Clock pulses are distributed throughout the system in such a way that the flip-flops are affected only with the arrival of the synchronization pulse.

The basic storage element is called a latch. It latches 0 or 1.A flip flop can maintain a binary state indefinitely until directed by an input signal to switch states.

4.2 Basic Flip-Flop Circuit

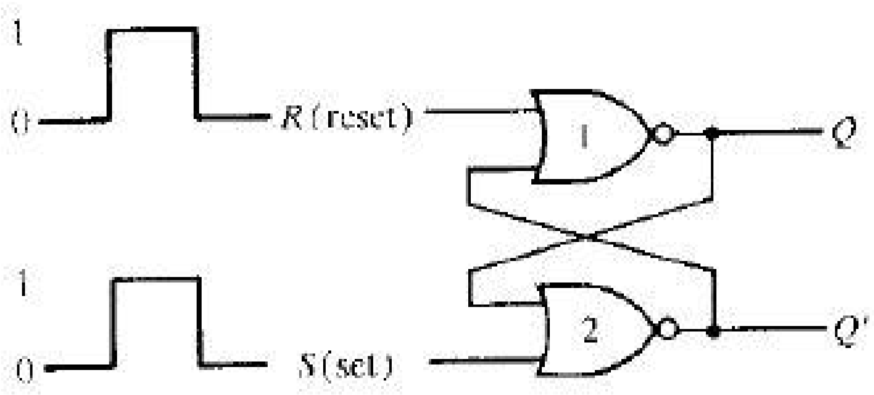
A flip-flop circuit can be constructed from two NOR gates or two NAND gates. First the operation of NOR and NAND gates is explained individually.

Basic Operation of NOR gate



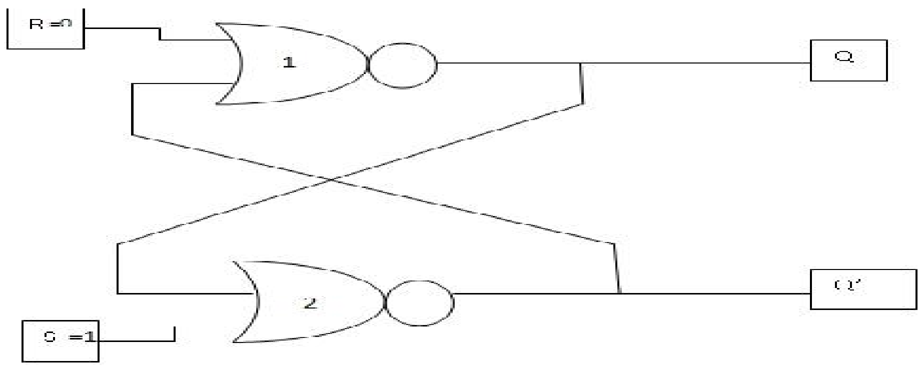
A	B	O/P
0	0	1
0	1	0
1	0	0
1	1	0

Basic flip-flop circuit using NOR gates



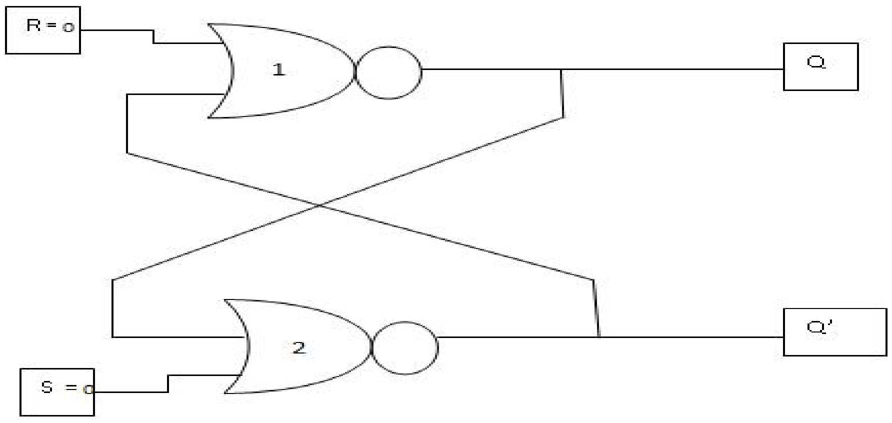
The cross coupled connection from the output of one gate to the input of other gate constitutes a feedback path. The inputs are S and R and the outputs are Q and Q′.

Case 1: S=1 and R=0



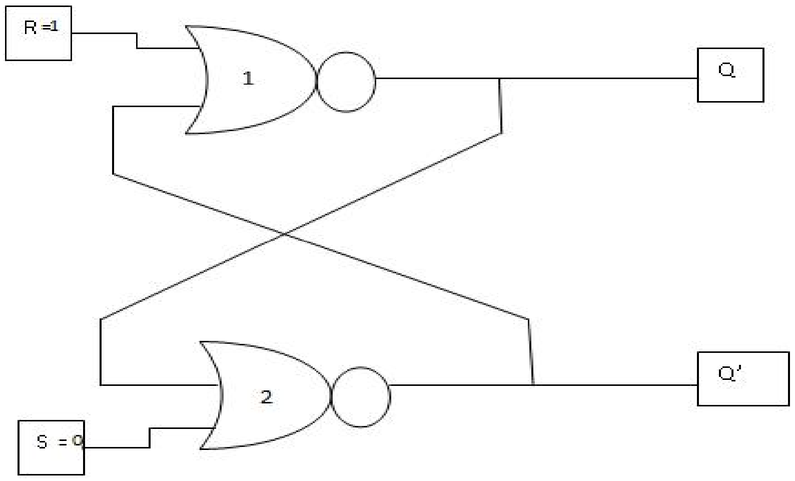
Case 2: S=0 and R=0

When the set input returns to 0. So, S=0, R=0, then Q=1, Q′=0



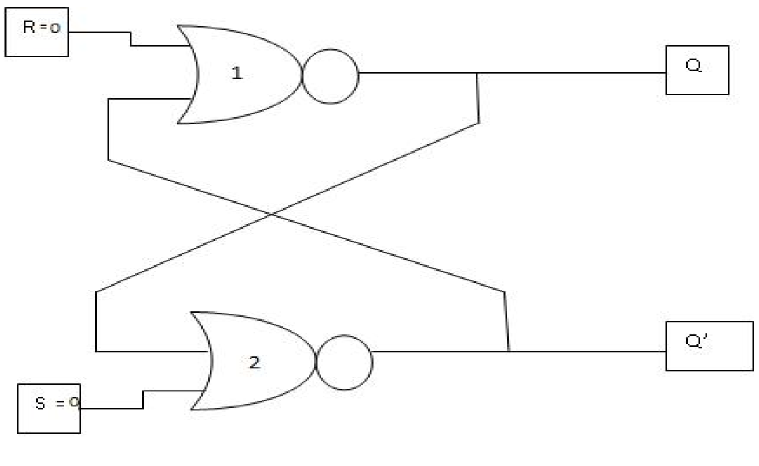
Case 3: S=0 and R=1

When a 1 is there in the reset input.So, S=0, R=1, then Q=0, Q′=1



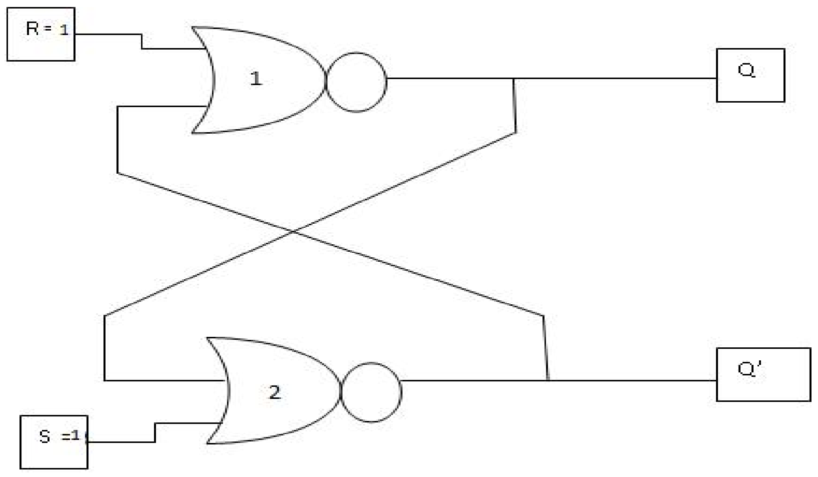
Case 4: S=0 and R=0

When reset input returns to 0. So, S=0, R=0, then Q=0, Q′=1



Case 5: S=1 and R=1

When S=1, R=1, then Q=1, Q′=1



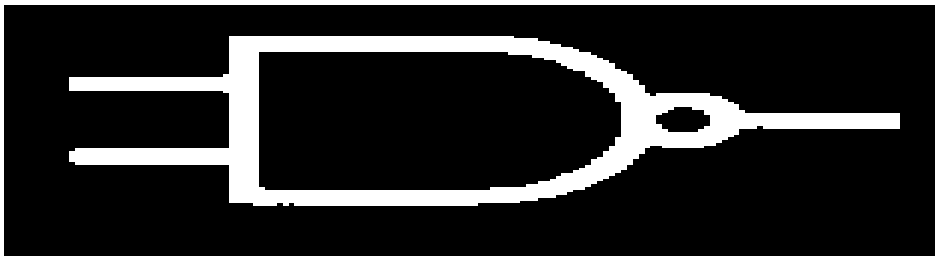
So, this flip-flop has two useful states:

- When Q=1 & Q′=0, it is in the set state or 1-state.
- When Q=0 & Q′=1, it is in clear state or 0-state.

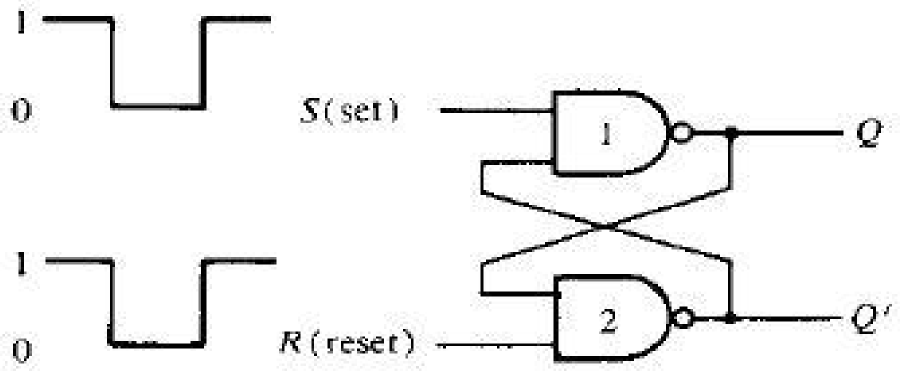
S	R	Q	Q′	
1	0	1	0	
0	0	1	0	(After S=1, R=0)
0	1	0	1	
0	0	0	1	(After S=0, R=1)
1	1	0	0	Undefined

Basic Operation of NAND gate

A	B	O/P
0	0	1
0	1	1
1	0	1
1	1	0

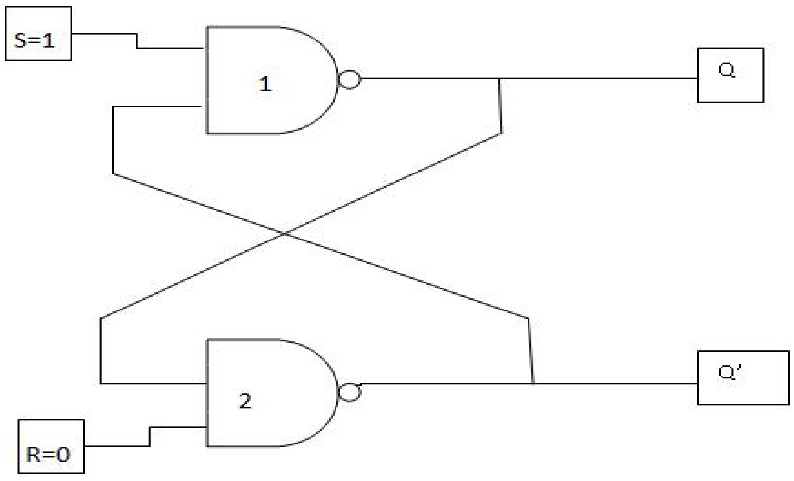


Basic flip-flop circuit using NAND gates



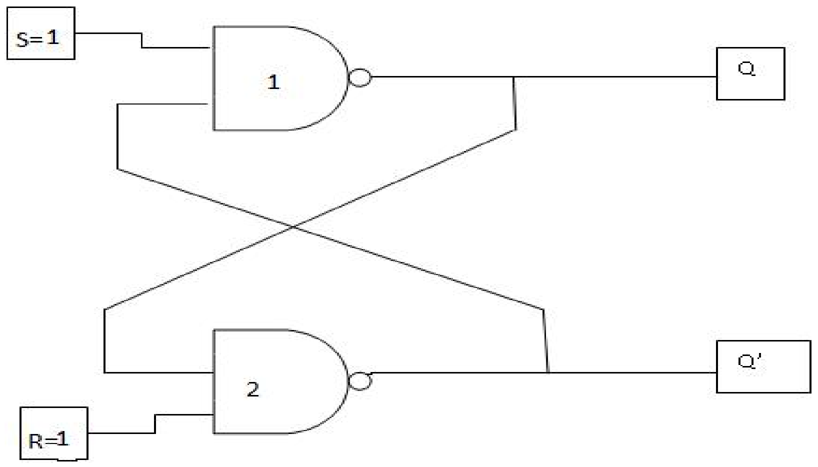
Case 1: S=1, R=0

When S=1, R=0, then Q=0, Q'=1



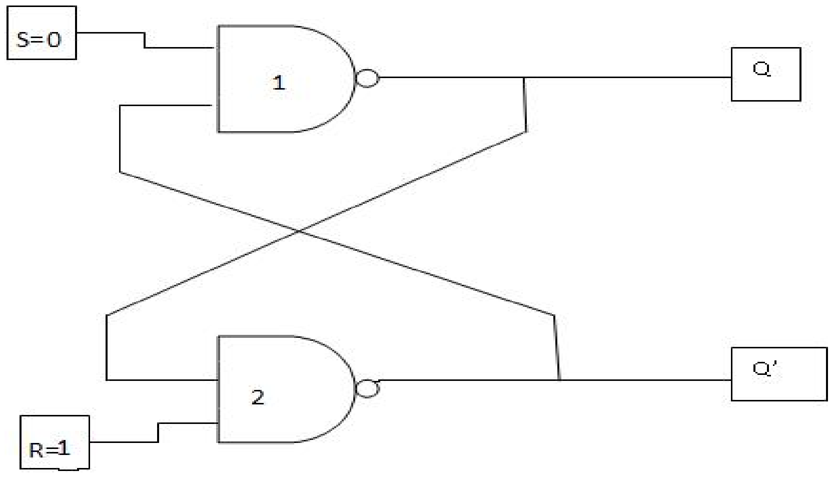
Case 2: S=1, R=1

When S=1, R=1, then Q=0, Q'=1



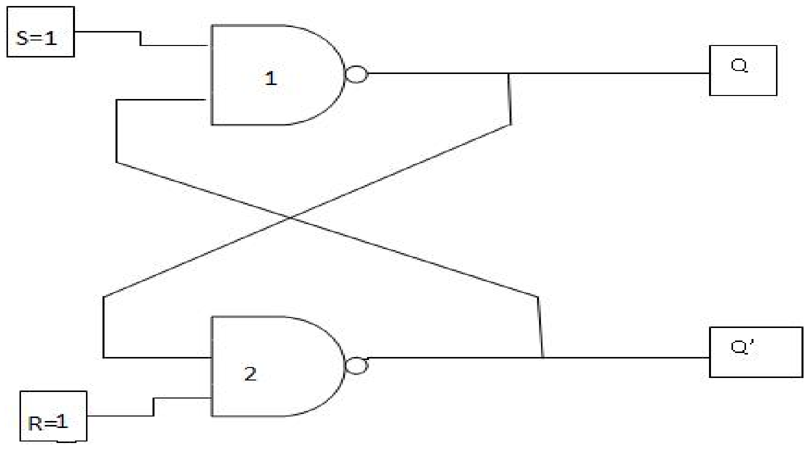
Case 3: S=0, R=1

When S=0, R=1, then Q=0, Q'=1



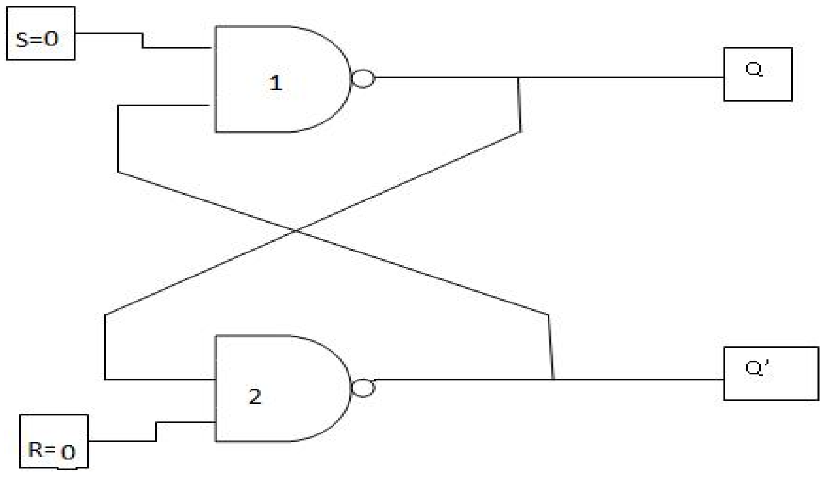
Case 4: S=1, R=1

When S=0, R=1, then Q=0, Q'=1



Case 5: S=0, R=0

When S=0, R=0, then Q=1, Q'=1

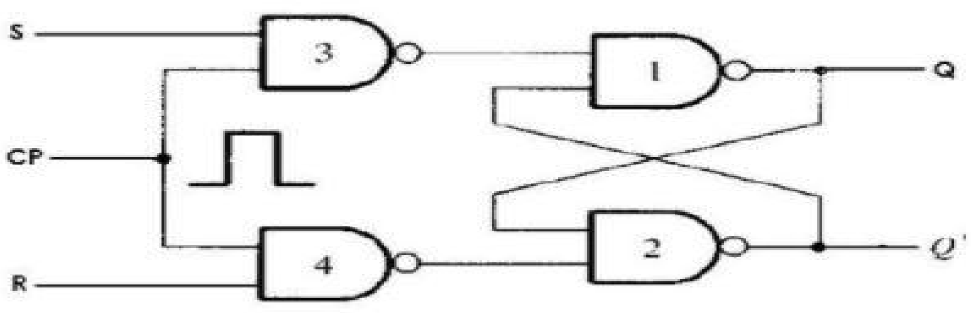


S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(After S=1, R=0)
0	1	1	0	
1	1	1	0	(After S=0, R=1)
0	0	1	1	

4.3 Introduction of Control Input

The operation of basic flip-flop can be modified by providing an additional control input, i.e., clock pulse which determines when the state of the clock needs to be changed. The best example of this includes RS flip-flop.

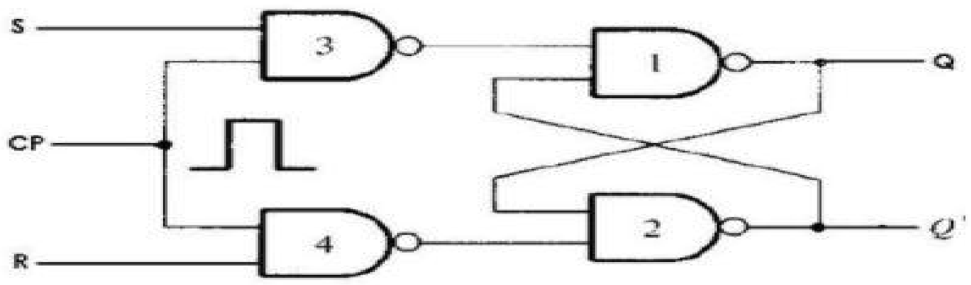
RS Flip-Flop



When CP=0, it does not matter what are the values of S and R, the outputs of gates 3 and 4 will stay at logic 1.When CP=1, the information from S and R is allowed to reach the output. So, here only those cases are shown when the CP=1.

Case 1: CP=1, S=1, R=0 (Set State)

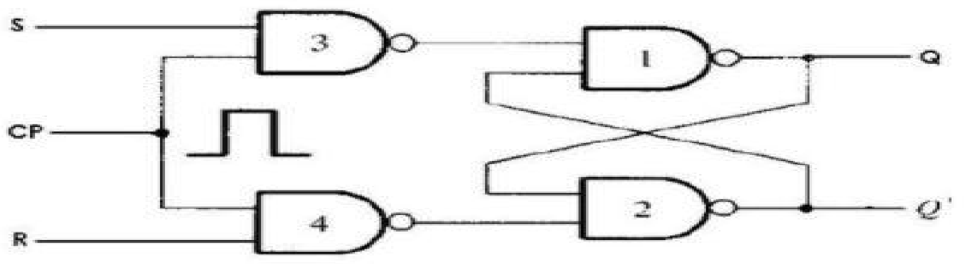
When CP=1, S=1, R=0, then Q=1, Q’=0



So, the set state is reached with S=1, R=0 and CP=1.

Case 2: CP=1, S=0, R=1 (Reset State)

When CP=1, S=0, R=1, then Q=0, Q’=1



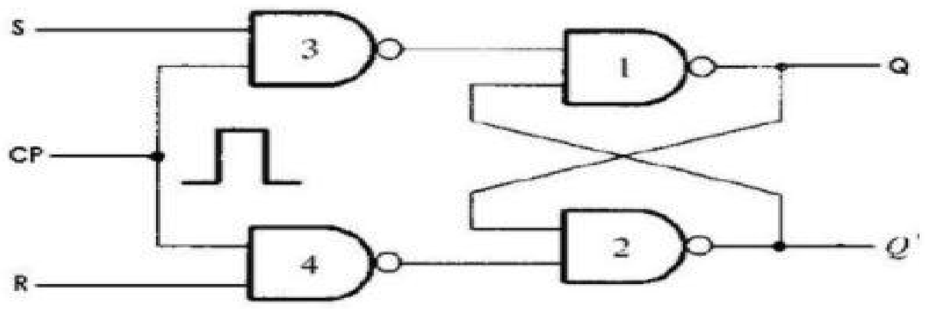
To change to reset state, the inputs must be S=0, R=1 and CP=1.In either case, the CP returns to 0 and the circuit remains in its previous state.

Case 3: CP=1, S=0, R=0

When CP=1 and both the S and R inputs equal to 0, the state of circuit does not change.

Case 4: CP=1, S=1, R=1 (Intermediate condition)

When CP=1, S=1, R=1, then Q=1, Q’=1



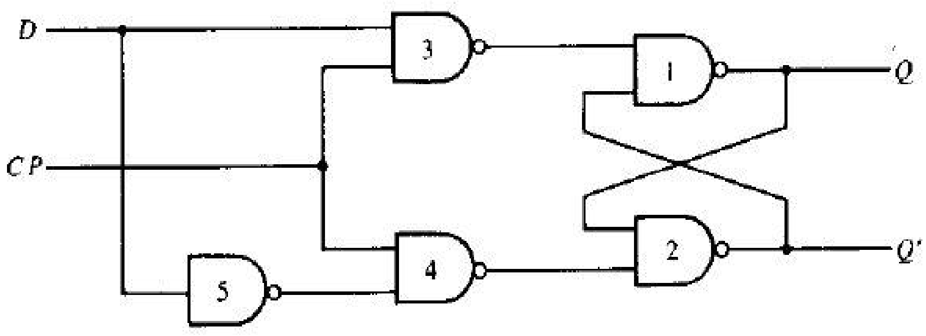
When CP input goes back to 0 (while S and Rare maintained at 1), it is not possible to determine the next state, as it depends on whether the output of gate 3 or gate 4 goes to 1 first. This intermediate condition makes the circuit difficult to manage and it is seldom used in practice. But it is an important circuit because all the other flip flops are constructed from this.

CP	S	R	Q	Q'	
0	0	0			No change
0	0	1			No change
0	1	0			No change
0	1	1			No change
1	0	0			No change
1	0	1	0	1	Reset state
1	1	0	1	0	Set state
1	1	1	1	1	Intermediate Condition

This control input disables the circuit by applying 0 to C, so that the state of the output does not change regardless of the values of S and R. When C=1, and both the S and r inputs are equal to 0, then the state of the circuit does not change.An intermediate condition occurs when all three inputs are equal to 1. This will pass 0s in both inputs of the basic SR latch. When the control input goes back to 0, one cannot conclusively determine the next state. This intermediate condition makes the circuit difficult to manage.

D Flip Flop

The D flip flop has two inputs: D (Data) and C (Enable/Control).The D input directly goes to S input and its complement goes to R input.



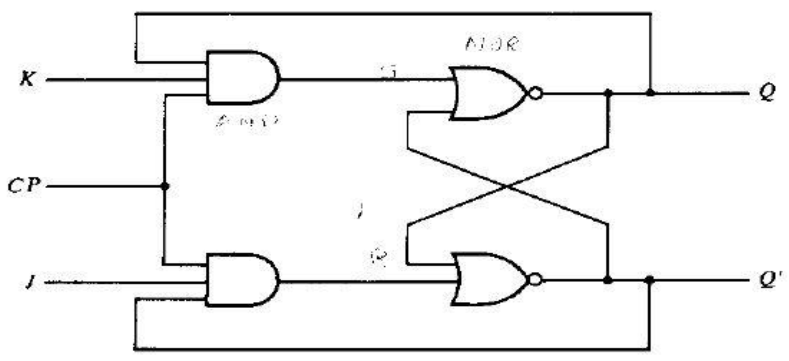
CP=0: If the pulse input is at 0, the outputs of gates 3 and 4 are at the 1 level and the circuit cannot change state regardless of the value of D.

CP=1: The D input is sampled when CP=1.If D=1, the Q output goes to 1, placing the circuit in the set state.If D=0, the output Q goes to 0 and the circuit switches to the clear state.

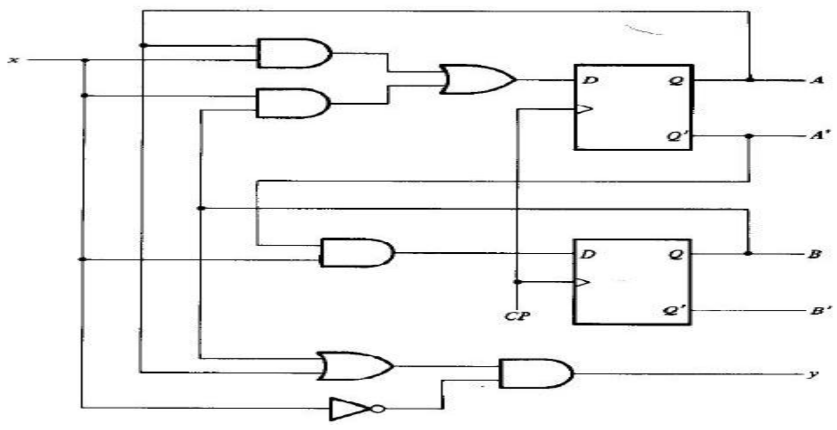
Clock	D	Q
0	0	No change
0	1	No change
1	0	0 (Clear State)
1	1	1 (Set State)

JK Flip Flop

It is a refinement of RS flip-flop in that the intermediate state of the RS type is defined in JK type.Inputs J and K behave like S and R to set and clear the flip flop respectively.The input marked J is for set and input marked K is for reset.When both inputs J and K are equal to 1, the flip flop switches to its complement state, that is, if Q=1, it switches to Q=0 and vice-versa.So, here two cross coupled NOR gates and two AND gates are used.Output Q is ANDed with K and CP inputs so that the flip-flop is cleared during a clock pulse only if Q was previously 1.Similarly, output Q′ is ANDed with J and CP inputs so that the flip flop is set with a clock pulse only when Q′ was previously 1.When both J and K are 1, the input pulse is transmitted through one AND gate only: the one whose input is connected to the flip flop output that is presently equal to 1.Thus, if Q=1, the output of the upper AND gate becomes 1 upon application of the clock pulse and the flip-flop is cleared.If Q′=1, the output of lower AND gate becomes 1 and the flip-flop is set.In either case, the output state of the flip-flop is complemented.



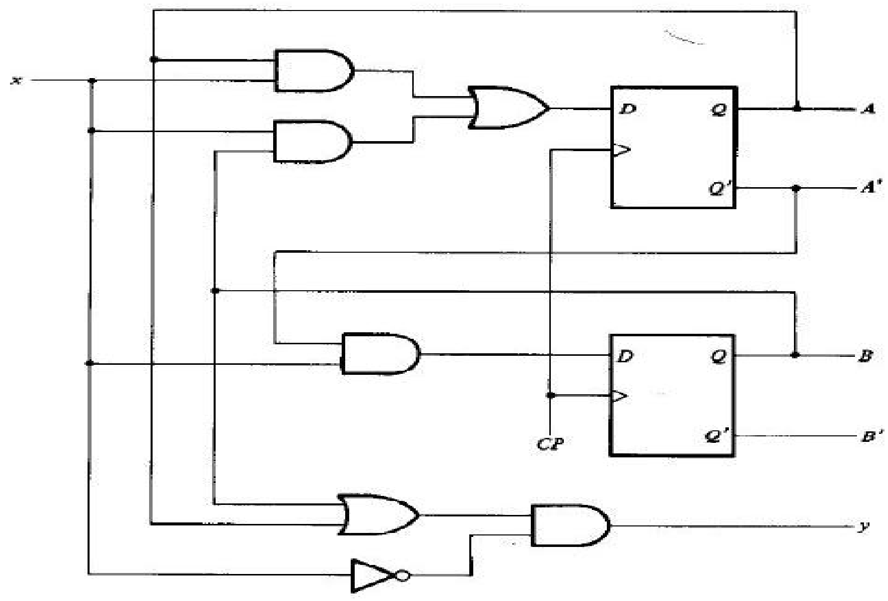
The analysis of a sequential circuit consists of obtaining a table or a diagram for the time sequence of inputs, outputs and internal states.A logic diagram is recognized as a clocked sequential circuit if it includes flip flops.



The analysis of a sequential circuit consists of obtaining a table or a diagram for the time sequence of inputs, outputs and internal states.

Logic Diagram

A logic diagram is recognized as a clocked sequential circuit if it includes flip flops. Next state equations for the circuit: A(t+1) = A(t)x(t) + B(t)x(t), B(t+1) = A′(t)x(t). The previous equations can be expressed in more compact form as follows:A(t+I) = Ax + Bx, B(t+1) = A′x.



The present-state value of the output can be expressed algebraically as follows:y(t)=[A(t) + B(t)]x'(t). Removing the symbol (t) for the present state, we obtain the output Boolean function:y=(A+ B)x'

State Table

The time sequence of inputs, outputs, and flip-flop states can be enumerated in a state table. The table consists of four sections labeled present state, input, next state, and output.

Present State		Input	Next State		Output
A	B	X	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

In general, a sequential circuit with m flip-flops and n inputs needs 2^{m+n} rows in the state table. The binary numbers from 0 through 2^{m+n} -1 are listed under the present-state and input columns. The next-state section has m columns, one for each flip-flop. The binary values for the next state are derived directly from the state equations. The output section has as many columns as there are output variables. We have eight binary combinations from 000 to 111. The next state of flip-flop A must satisfy the state equation:A (t + I) = Ax + Bx. Similarly, the next state of flip-flop B is derived from the state equationB(t + 1) = A'x. The output column is derived from the output equation y = Ax' + Bx.

Second Form of State Table

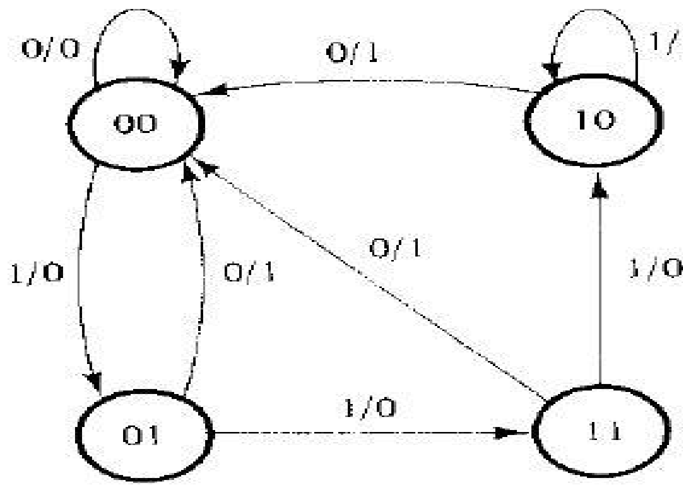
	Next State		Output	
Present State	X=0	X=1	X=0	x=1
AB	AB	AB	y	Y
00	00	01	0	0
01	00	11	1	0

10	00	10	1	0
11	00	10	1	0

In this configuration, the state table has only three sections: present state, next state, and output. The input conditions are enumerated under the next-state and output sections. For each present state, there are two possible next states and outputs, depending on the value of the input.

State Diagram

The information available in a state table can be represented graphically in a state diagram. In this type of diagram, a state is represented by a circle, and the transition between states is indicated by directed lines connecting the circles.



State table vs. State diagram

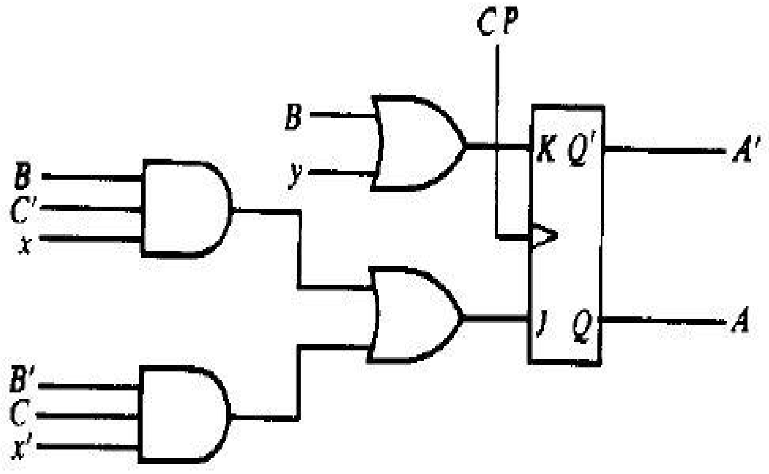
There is no difference between a state table and a state diagram except in the manner of representation. The state table is easier to derive from a given logic diagram and the state diagram follows directly from the state table. The state diagram gives a pictorial view of state transitions and is the form suitable for human interpretation of the circuit operation.

Flip-Flop Input Functions

The part of the circuit that generates the inputs to flip-flops are described algebraically by a set of Boolean functions called flip-flop input functions, or sometimes input equations.As an example, consider the following flip-flop input functions: JA = BC'x + B'Cx' &KA=B+y.

JA and KA designate two Boolean variables. The first letter in each denotes the J and K input, respectively, of a JK flip-flop. The second letter, A, is the symbol name of the flip-flop. The right side of each equation is a Boolean function for the corresponding flip-flop input variable.

Implementation of Flip-Flop Input Functions



The JK flip-flop has an output symbol A and two inputs labeled J and K. This combinational circuit is the implementation of the algebraic expression given by the input functions. The outputs of the combinational circuit are denoted by JA and KA in the input functions and go to the J and K inputs, respectively, of flip-flop A.The sequential circuit has one input x, one output y, and two D flip-flops A and B.

Flip-Flop Input Functions

The logic diagram can be expressed algebraically with two flip-flop input functions and one output-circuit function:

DA= Ax+Bx

DB= A'x

y =(A+ B)x'

This set of Boolean functions provides all the necessary information for drawing the logic diagram of the sequential circuit. The symbol DA specifies a D flip-flop labeled A. DB specifies a second D flip-flop labeled B. The flip-flop input functions constitute a convenient algebraic form for specifying a logic diagram of a sequential circuit. They imply the type of flip-flop from the first letter of the input variable, and they fully specify the combinational circuit that drives the flip-flop.

4.4 Characteristic Table

The analysis of a sequential circuit with flip-flops other than the D type is complicated because the relationship between the inputs of the flip-flop and the next state is not straightforward. They define the next state as a function of the inputs and present state. Q(t) refers to the present state prior to the application of a pulse. Q (t + I) is the next state one clock period later.

JK Flip-Flop Characteristic Table

J	K	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Reset

1	0	1	Set
1	1	Q′(t)	Complement

RS Flip Flop Characteristic Table

S	R	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Unpredictable

D Flip Flop Characteristic Table

D	Q(t+1)	
0	0	Reset
1	1	Set

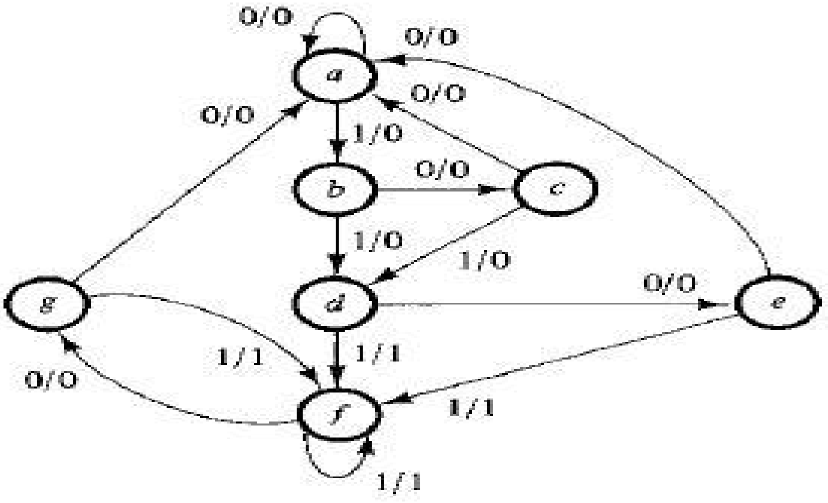
T Flip Flop Characteristic Table

T	Q(t+1)	
0	Q(t)	No change
1	Q′(t)	Complement

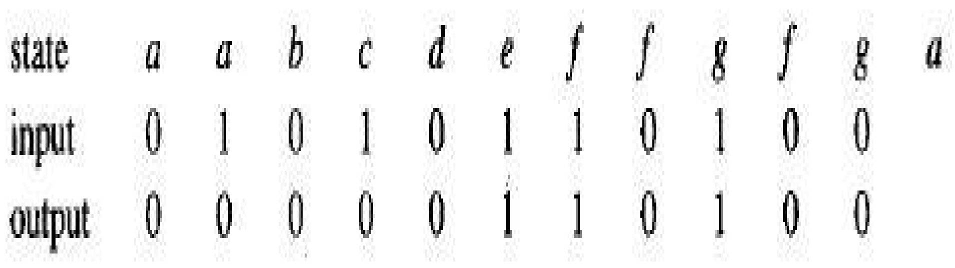
The analysis of sequential circuits starts from a circuit diagram and culminates in a state table or diagram.

State Reduction

Any design process must consider the problem of minimizing the cost of the final circuit.The reduction of the number of flip-flops in a sequential circuit is referred to as state reduction problem.



There are infinite number of input sequences that may be applied to the circuit, each results in a unique output sequence.Example: Consider the input sequence 01010110100 starting from the initial state a. Each input of 0 or 1 produces an output of 0 or 1 and causes the circuit to go to the next state.



state	a	a	b	c	d	e	f	f	g	f	g	a
input	0	1	0	1	0	1	1	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	

4.5 State Table

Let us assume that we have found a sequential circuit whose state diagram has less than 7 states and we wish to compare it with the circuit whose state diagram is just shown.If identical inputs sequences are applied to the two circuits and identical outputs occur for all input sequences, then the two circuits are said to be equivalent, and one may be replaced by the other.The problem of state reduction is to find ways of reducing the number of states in a sequential circuit without altering the input-output relationships.

	Next state		Output	
Present state	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0

c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

Algorithm for state reduction

Two states are said to be equivalent if, for each member of the set of inputs, they give the same output and send the circuit either to the same state or to an equivalent state.When two states are equivalent, one of them can be removed without altering the input-output relationships.

Reducing the state table

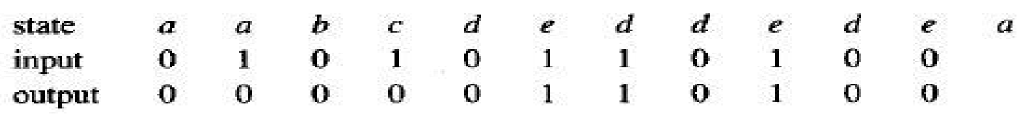
	Next state		Output	
Present state	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g e	f	0	1
g	a	f	0	1

	Next state		Output	
Present state	X=0	X=1	X=0	X=1

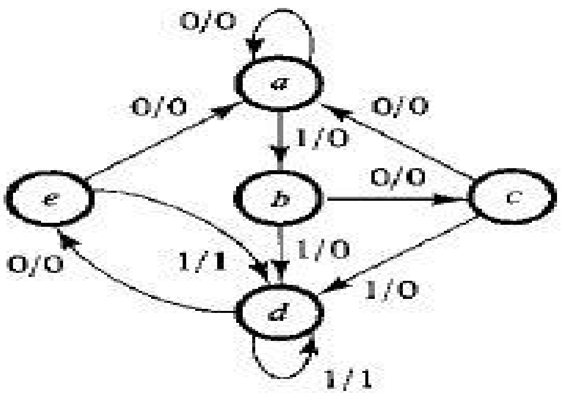
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f d	0	1
e	a	f d	0	1
f	e	f d	0	1

	Next state		Output	
Present state	X=0	X=1	X=0	X=1
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

Considering again the same input sequence 01010110100 starting from the initial state a.It results in the same output sequence results although the state sequence is different.



state	*a*	*a*	*b*	*c*	*d*	*e*	*d*	*d*	*e*	*d*	*e*	*a*
input	0	1	0	1	0	1	1	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	



In this example of sequential circuit, the number of states are reduced from seven to five.In either case, the representation of the states with physical components require that we use three flip flops, we can formulate up to 8 binary states denoted by binary numbers 000 through 111, with each bit designating the state of one flip-flop.In general, reducing the number of states in a state table is likely to result in a circuit with less equipment

State Assignment

The cost of the combinational circuit part of a sequential circuit can be reduced by using the known simplification methods for combinational circuits.The factor state assignment problem that comes into play in minimizing the combinational gates.State assignment procedures are concerned with methods for assigning binary values to states in such a way as to reduce the cost of the combinational circuit that derives the flip flops.

State	Assignment 1	Assignment 2	Assignment 3
a	001	000	000
b	010	010	100
c	011	011	010
d	100	101	101
e	101	111	011

Binary State Assignment

Assignment 1 is a straight binary assignment for the sequence of states from a through e. The other two assignments for the sequence of states from a through e. The other two assignments are chosen arbitrarily.

Reduced State Table with Binary Assignment 1

	Next Stage		Output	
Present State	x=0	x=1	x=0	x=1
001	001	010	0	0
010	011	100	0	0
011	001	100	0	0
100	101	100	0	1
101	001	100	0	1

The binary form of the state table is used to derive the combinational circuit part of sequential circuit. The complexity of the combinational circuit obtained depends on the binary state assignment chosen.

4.6 Characteristic Table and Excitation Table

The characteristic table is useful for analysis and for defining the operation of the flip-flop. It specifies the next state when the inputs and present state are known. During the design process, we usually know the transition from present state to next state and wish to find the flip-flop input conditions that will cause the required transition. For this reason, we need a table that lists the required inputs for a given change of state. Such a list is called an excitation table.

RS Flip Flop Characteristic Table

S	R	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Unpredictable

RS Flip Flop Excitation Table

Q(t)	Q(t+1)	S	R
0	0	0	X

0	1	1	0
1	0	0	1
1	1	X	0

JK Flip-Flop Characteristic Table

J	K	Q(t+1)	
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

JK Flip Flop Excitation Table

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

D Flip Flop Characteristic Table

D	Q(t+1)	
0	0	Reset

1	1	Set

D Flip Flop Excitation Table

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

T Flip Flop Characteristic Table

T	Q(t+1)	
0	Q(t)	No change
1	Q′(t)	Complement

T Flip Flop Excitation Table

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

4.7 Design Procedure

The design of a clocked sequential circuit starts from a set of specifications and culminates in a logic diagram or a list of Boolean functions from which the logic diagram can be obtained. In contrast to a combinational circuit, which is fully specified by a truth table, a sequential circuit requires a state table for its specification. The first step in the design of sequential circuits is to obtain a state table or an equivalent representation, such as a state diagram.A synchronous sequential circuit is made up of flip-flops and combinational gates. The design of the circuit consists of choosing the flip-flops and then finding a combinational gate structure that, together with the flip-flops, produces a circuit that fulfills the stated specifications.The number of flip-flops is determined from the number of states needed in the circuit. The design process involves a transformation from the sequential-circuit problem into a combinational-circuit problem.

Step 1: The word description of the circuit behavior is stated. This may be accompanied by a state diagram, a timing diagram, or other pertinent information.

Step 2: From the given information about the circuit, obtain the state table.

Step 3: The number of states may be reduced by state-reduction methods if the sequential circuit can be characterized by input-output relationships independent of the number of states.

Step 4: Assign binary values to each state if the state table obtained in step 2 or 3 contains letter symbols.

Step 5: Determine the number of flip-flops needed and assign a letter symbol to each.

Step 6: Choose the type of flip-flop to be used.

Step 7: From the state table, derive the circuit excitation and output tables.

Step 8: Using the map or any other simplification method, derive the circuit output functions and the flip-flop input functions.

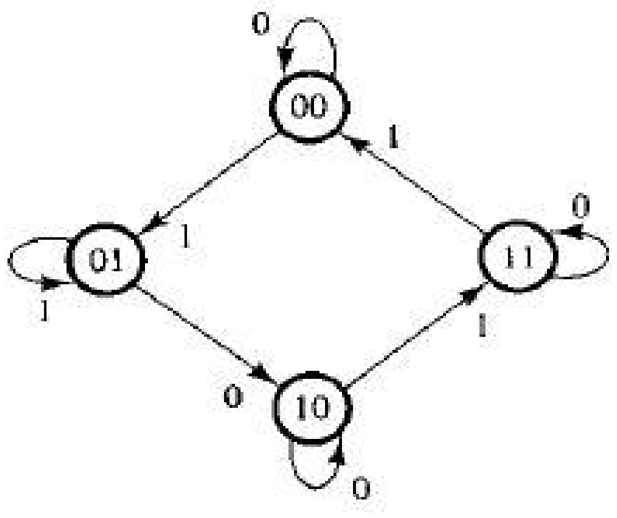
Step 9: Draw the logic diagram.

Representation of states

The m flip-flops can represent up to 2^m distinct states.A circuit may have unused binary states if the total number of states is less than 2^m. The unused states are taken as don't-care conditions during the design of the combinational circuit part of the circuit.

Type of flip-flop to be used

The type of flip-flop to be used may be included in the design specifications or may depend on what is available to the designer. Many digital systems are constructed entirely with JK flip-flops because they are the most versatile available.When many types of flip-flops are available, it is advisable to use the D flip-flop for applications requiring transfer of data (such as shift registers), the T type for applications involving complementation (such as binary counters), and the JK type for general applications.



Flip-Flop specified: JK Flip-Flop

State Table

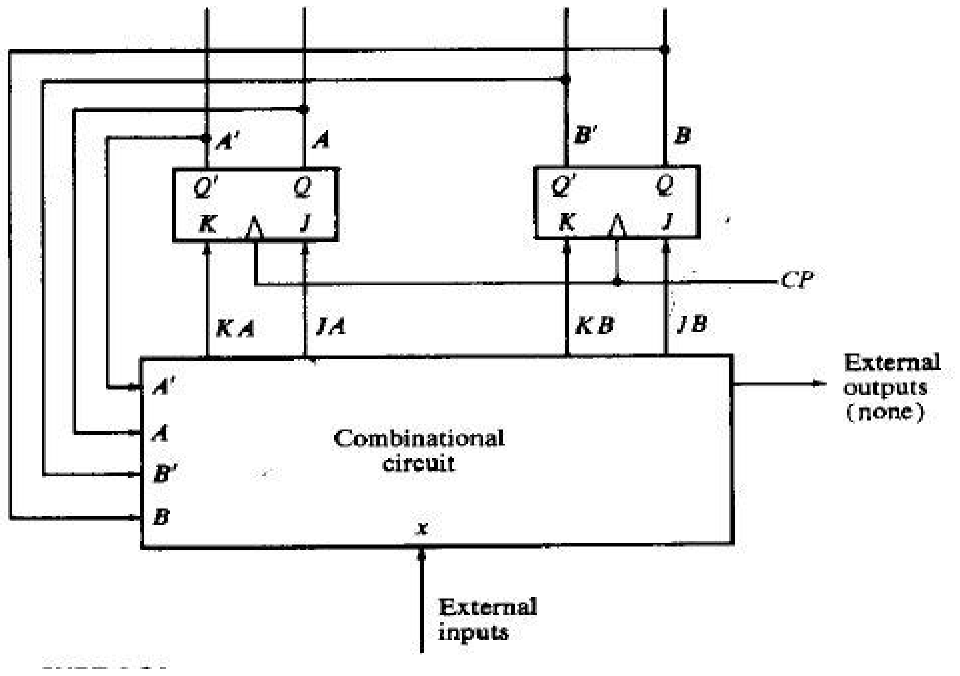
		Next State			
Present State		X=0		X=1	
A	B	A	B	A	B
0	0	0	0	0	1
0	1	1	0	0	1
1	0	1	0	1	1
1	1	1	1	0	0

Excitation Table

Inputs of Combinational Circuit					Outputs of Combinational Circuit			
Present State		Input	Next State		Flip-Flop Inputs			
A	B	x	A	B	JA	KA	JB	KB
0	0	0	0	0	0	X	0	X
0	0	1	0	1	0	X	1	X
0	1	0	1	0	1	X	X	1

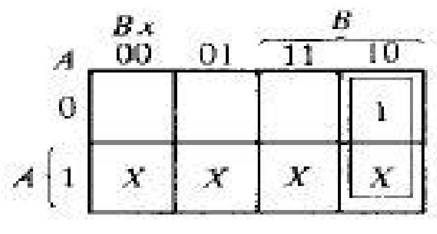
0	1	1	0	1	0	X	X	0
1	0	0	1	0	X	0	0	X
1	0	1	1	1	X	0	1	X
1	1	0	1	1	X	0	X	0
1	1	1	0	0	X	1	X	1

Block Diagram of Sequential Circuit

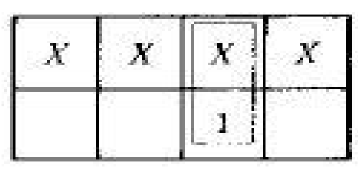


Derivation of simplified Boolean function

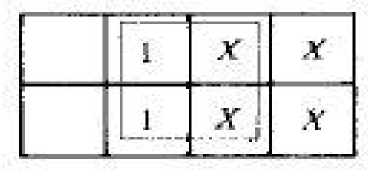
The information from the truth table is transferred into the maps.



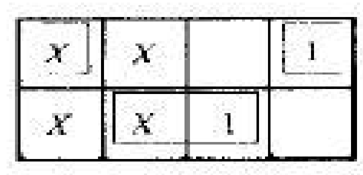
$JA = Bx'$



$KA = Bx$



$JB = x$

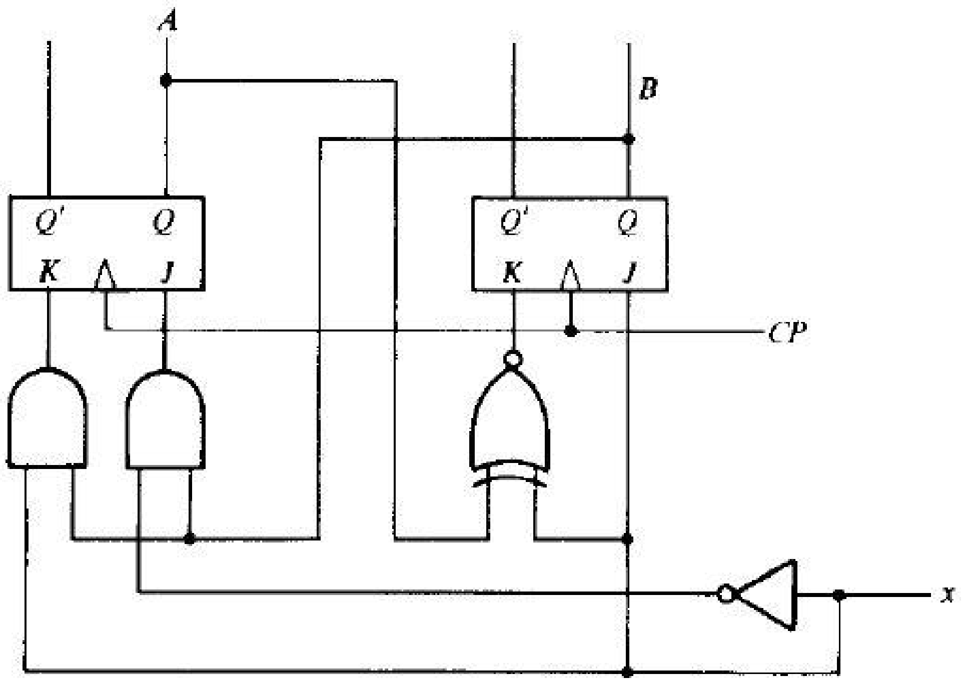


$KB = (A \oplus x)'$

Where the four simplified flip-flop input functions are derived:

- JA = Bx′ KA= Bx
- JB = x KB = (A ⊕ x)'

Logic Diagram



Design with D flip flop

The time it takes to design a sequential circuit that uses D flip-flops can be shortened if we utilize the fact that the next state of the flip-flop is equal to its D input prior to the application of a clock pulse.

Excitation Table

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

State Table

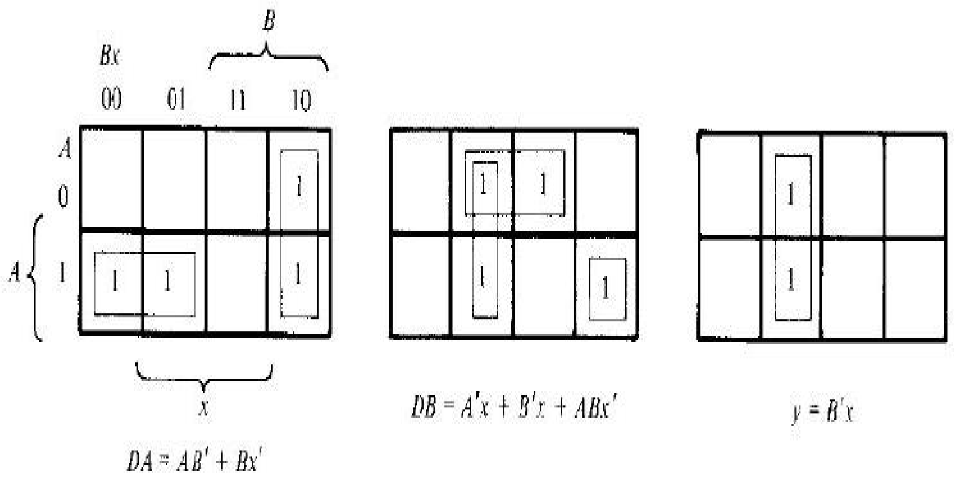
Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	0	0

It is not necessary to include the excitation table for flip-flop inputs DA and DB since DA = A (t + 1) and DB = B(t + 1). The sum of min-terms is as follows:

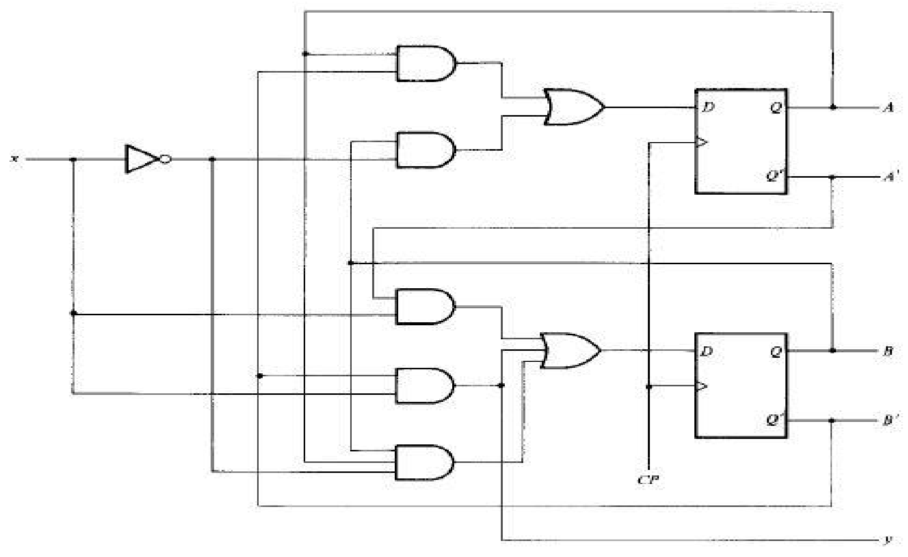
- DA(A, B, x) = Σ (2, 4, 5, 6)
- DB(A, B, x) = Σ (I, 3, 5, 6)
- y(A, B, x) = Σ (1, 5)

Maps for Input Functions and Output y

The Boolean functions are simplified by means of the maps.



Logic diagram of sequential circuit



Design with unused states

A circuit with m flip-flops would have 2^m states. There are occasions when a sequential circuit may use less than this maximum number of states. When simplifying the input functions to flip-flops, the unused states can be treated as don't-care conditions.

State Table

Present State			Input	Next State			Flip-Flop Inputs						Output
A	B	C	X	A	B	C	SA	RA	SB	RB	SC	RC	Y
0	0	1	0	0	0	1	0	X	0	X	X	0	0
0	0	1	1	0	1	0	0	X	1	0	0	1	0

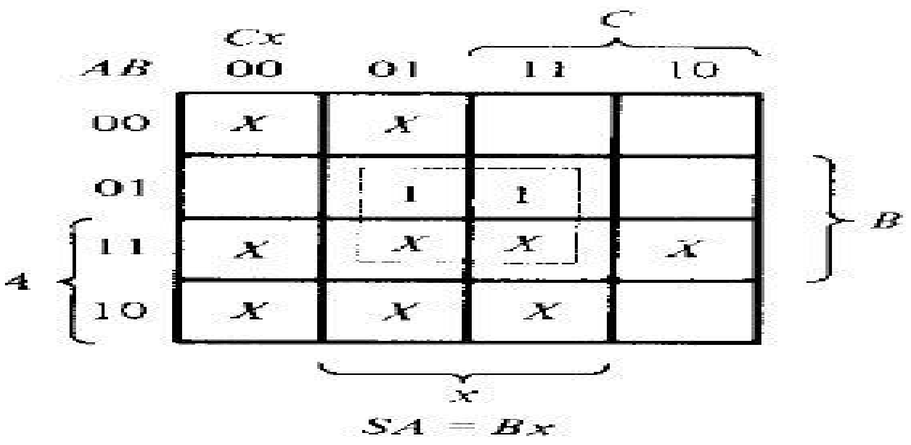
0	1	0	0	0	1	1	0	X	X	0	1	0	0
0	1	0	1	1	0	0	1	0	0	1	0	X	0
0	1	1	0	0	0	1	0	X	0	1	X	0	0
0	1	1	1	1	0	0	1	0	0	1	0	1	0
1	0	0	0	1	0	1	X	0	0	X	1	0	0
1	0	0	1	1	0	0	X	0	0	X	0	X	1
1	0	1	0	0	0	1	0	1	0	X	X	0	0
1	0	1	1	1	0	0	X	0	0	X	0	1	1

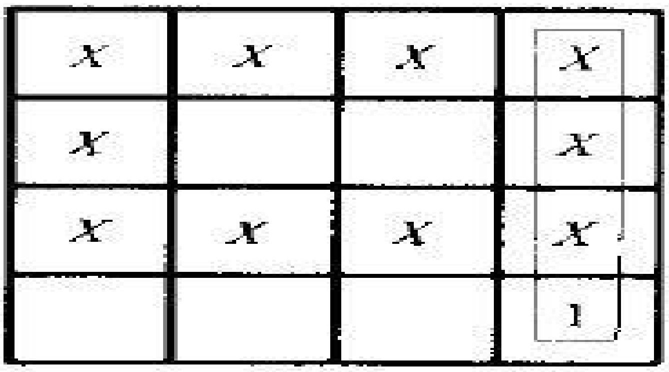
Flip-flop specified: RS Flip Flop

There are five states listed in the table: 001, 010, O11, 100, and 101. The other three states, 000, 110, and 111, are not used. When an input of 0 or 1 is included with these unused states, we obtain six min-terms: 0, 1, 12, 13, 14, and 15.

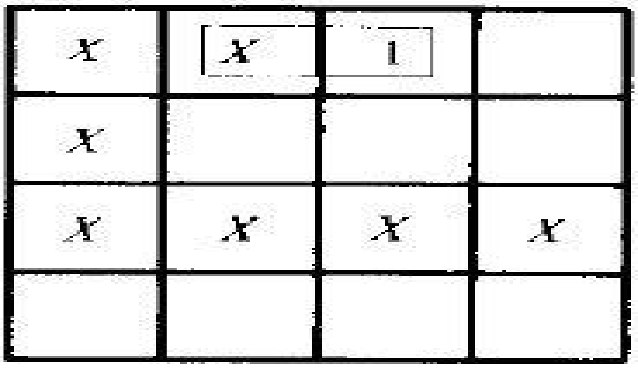
Maps for simplifying the sequential circuit

Six maps are for simplifying the input functions for the three RS flip-flops. The seventh map is for simplifying the output y. Each map has six X's in the squares of the don't-care min-terms 0, 1, 2, 13, 14, and 15.The other don't-care terms in the maps come from the X's in the flip-flop input columns of the table.

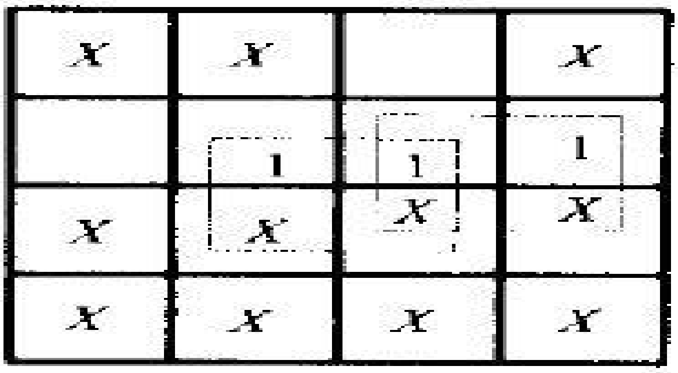




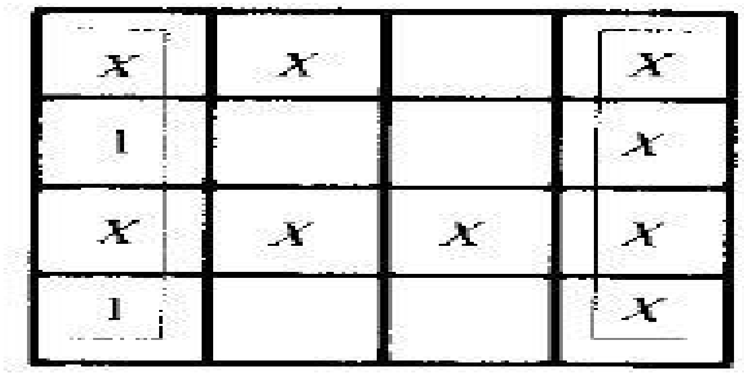
$RA = Cx'$



$SB = A'B'x$

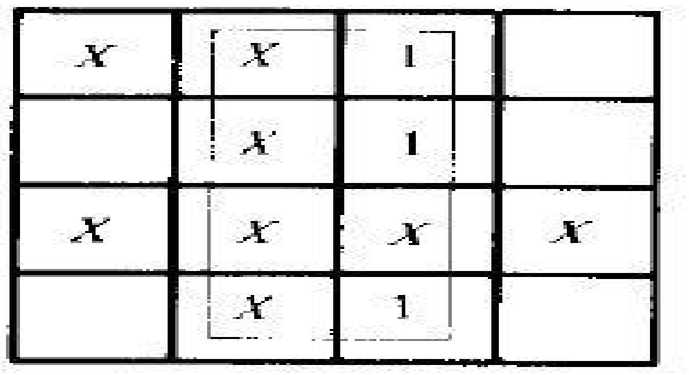


$RB = BC + Bx$



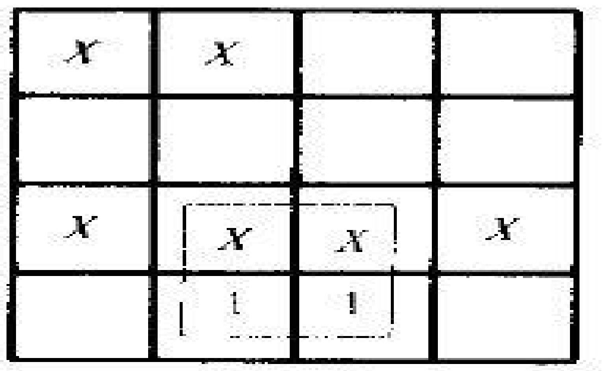
X	X		X
1			X
X	X	X	X
1			X

$SC = x'$



X	X	1	
	X	1	
X	X	X	X
	X	1	

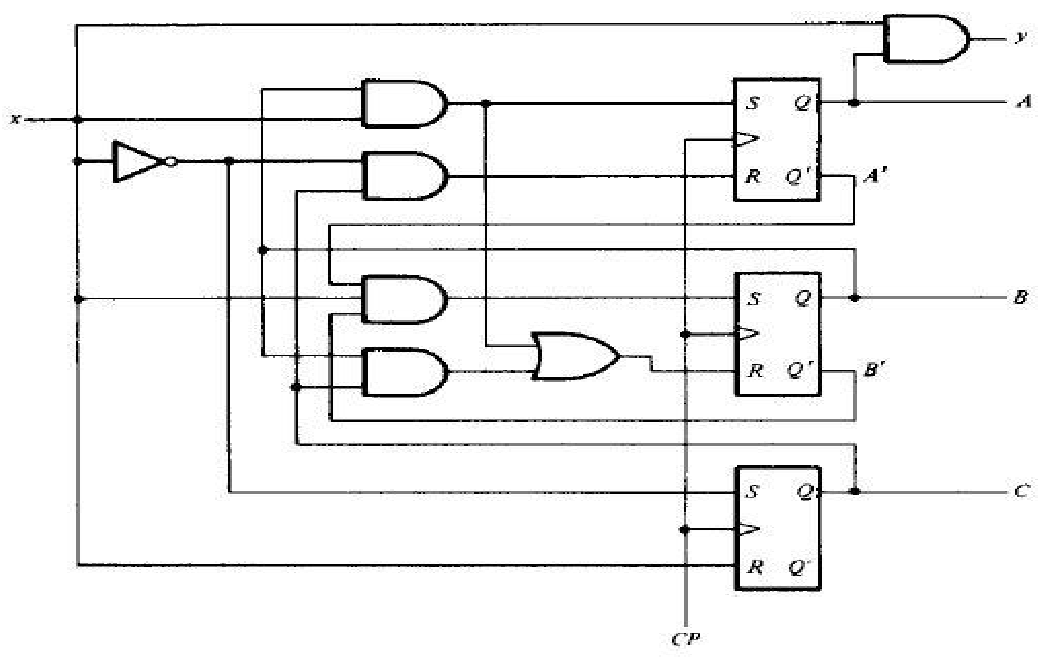
$RC = x$



X	X		
X	X	X	X
	1	1	

$y = Ax$

Logic Diagram with RS Flip-Flop



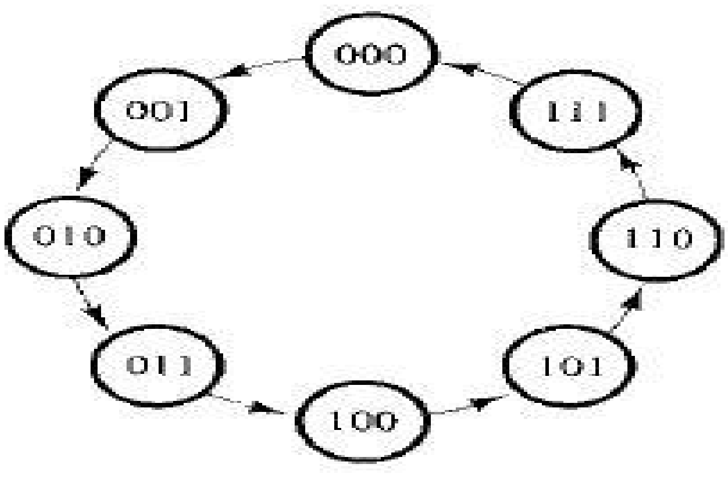
4.8 Counters

A sequential circuit that goes through a prescribed sequence of states upon the application of input pulses is called a counter. The input pulses, called count pulses, may be clock pulses or they may originate from an external source and may occur at prescribed intervals of time or at random. In a counter, the sequence of states may follow a binary count or any other sequence of states. Counters are found in almost all equipment containing digital logic. They are used for counting the number of occurrences of an event and are useful for generating timing sequences to control operations in a digital system.

Binary Counter

Of the various sequences a counter may follow, the straight binary sequence is the simplest and most straightforward. A counter that follows the binary sequence is called a binary counter. An n-bit binary ripple counter consists of n flip-flops and can count in binary from O to $2^n - 1$.

State Diagram of a 3-bit binary ripple counter

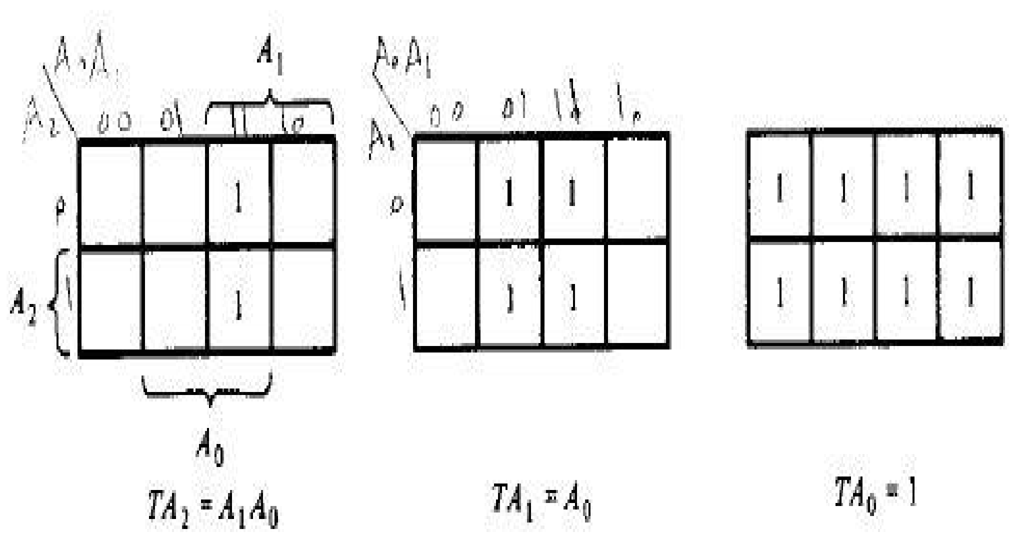


The next state of a counter depends entirely on its present state, and the state transition occurs every time the pulse occurs.

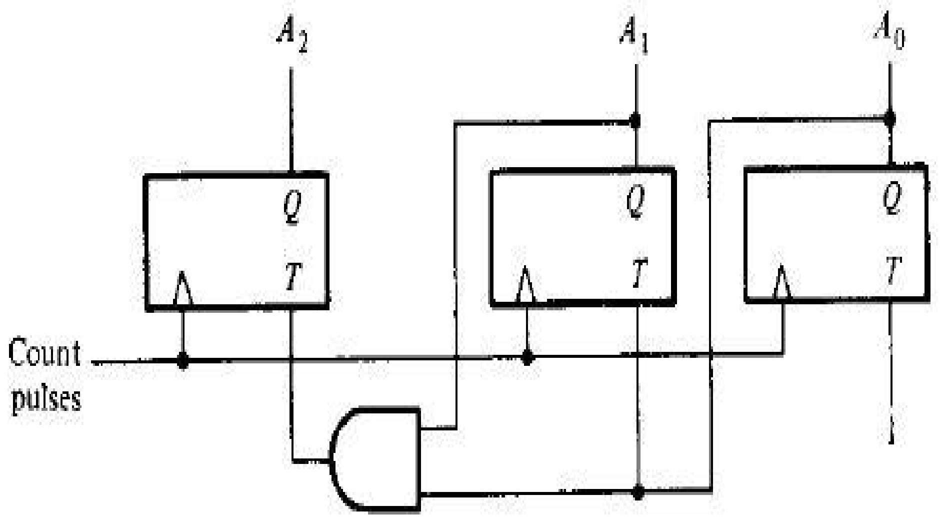
Excitation Table for 3-bit ripple Counter

Present State			Next State			Flip-Flop Inputs		
A2	A1	A0	A2	A1	A0	TA2	TA1	TA0
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

Maps for 3-bit ripple Counter



Logic Diagram of 3-bit ripple Counter



Counter with non-binary sequence-Decade Counters

A counter with n flip-flops may have a binary sequence of less than 2^n states. A BCD counter counts the binary states from 0000 to 100I and returns to 0000 to repeat the sequence. Other counters may follow an arbitrary sequence that may not be the straight binary sequence.In any case, the design procedure is the same.The count has a repeated sequence of six states, with flip-flops B and C repeating the binary count 00, 01, 10, while flip-flop A alternates between 0 and 1 every three counts. The count sequence is not straight binary and two states, 011 and 11I, are not included in the count.

Excitation table for counter

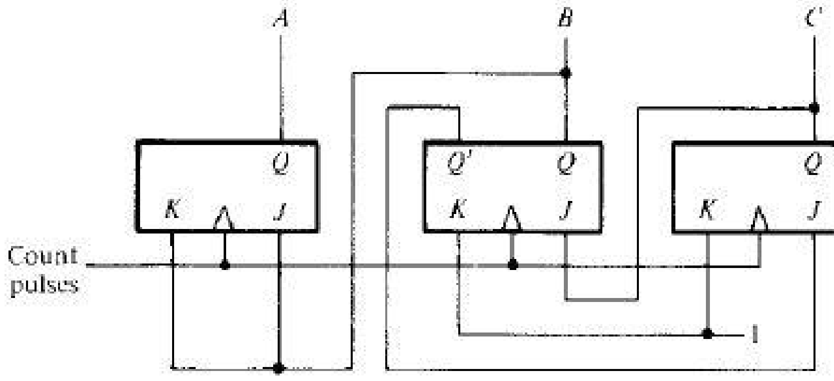
Present State			Next State			Flip-Flop Inputs					
A	B	C	A	B	C	JA	KA	JB	KB	JC	KC
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	1	0	0	1	X	X	1	0	X
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	0	0	0	X	1	X	1	0	X

Inputs KB and KC have only 1's and X's in their columns, so these inputs are always equal to 1. The other flip-flop input functions can be simplified using min-terms 3 and 7 as don't-care conditions. The simplified functions are:

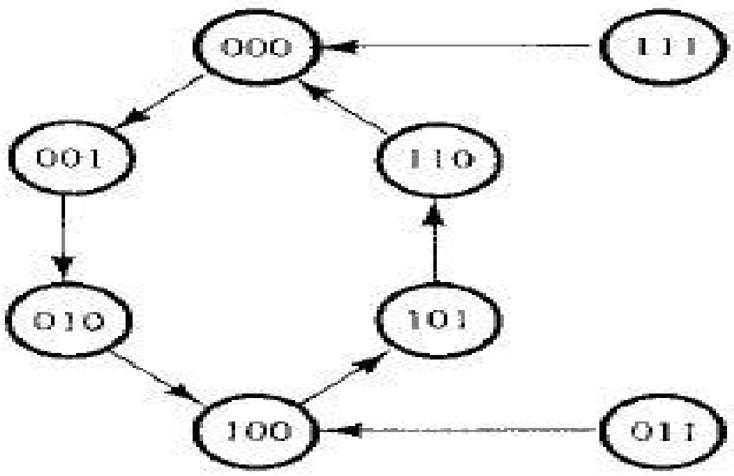
JA = B	JB = C	JC= B′
KA= B	KB= 1	KC= 1

Logic Diagram of Counter with Non-Binary Sequence

If the circuit happens to be in state 011 because of an error signal, the circuit goes to state 100 after the application of clock pulse. This is obtained by noting that while the circuit is in present state 011, the outputs of the flip-flops are A = 0, B = 1, and C = 1.From the flip-flop input functions, we obtain JA = KA = 1, JB = KB = 1, JC = 0, and KC = 1. Therefore, flip-flop A is complemented and goes to 1.Flip-flop B is also complemented and goes to O. Flip-flop C is reset to O because KC = 1.This results in next state 100. In a similar manner, we can evaluate the next state from present state 111 to be 000.



State Diagram of Counter with Non-Binary Sequence



If the circuit ever goes to one of the unused states because of an error, the next count pulse transfers it to one of the valid states and the circuit continues to count correctly. Thus, the counter is self-correcting. A self-correcting counter is one that if it happens to be in one of the unused states, it eventually reaches the normal count sequence after one or more clock pulses.

Summary

- The sequential circuit is made up of combinational circuit to which storage elements are connected to form a feedback path.

- A flip-flop is a binary storage device which is capable of storing one bit of information.
- A flip flop can maintain a binary state indefinitely until directed by an input signal to switch states.
- The cross coupled connection from the output of one gate to the input of other gate constitutes a feedback path.
- The D flip flop has two inputs: D (Data) and C (Enable/Control).
- The analysis of a sequential circuit consists of obtaining a table or a diagram for the time sequence of inputs, outputs and internal states.
- The state table consists of four sections labeled present state, input, next state, and output.
- The information available in a state table can be represented graphically in a state diagram.
- There are infinite number of input sequences that may be applied to the circuit; each results in a unique output sequence.
- State assignment procedures are concerned with methods for assigning binary values to states in such a way as to reduce the cost of the combinational circuit that derives the flip flops.
- The design of a clocked sequential circuit starts from a set of specifications and culminates in a logic diagram or a list of Boolean functions from which the logic diagram can be obtained.
- The design of a combinational circuit, which is fully specified by a truth table, a sequential circuit requires a state table for its specification.
- In a counter, the sequence of states may follow a binary count or any other sequence of states.

Keywords

- **Flip-flop:**A flip-flop is a binary storage device which can store one bit of information.
- **Latch:**The basic storage element is called a latch. It latches 0 or 1.
- **State Table:**The time sequence of inputs, outputs, and flip-flop states can be enumerated in a state table.
- **State Diagram:**A state is represented by a circle, and the transition between states is indicated by directed lines connecting the circles.
- **Characteristic Table:** The characteristic table is useful for analysis and for defining the operation of the flip-flop. It specifies the next state when the inputs and present state are known.
- **Counter:** A sequential circuit that goes through a prescribed sequence of states upon the application of input pulses is called a counter.
- **Binary Counter:**A counter that follows the binary sequence is called a binary counter.
- **Self-correcting Counter:**A self-correcting counter is one that if it happens to be in one of the unused states, it eventually reaches the normal count sequence after one or more clock pulses.

Self Assessment

1. In a state table, which of these things are included?

A. Present state, Input

B. Next state, Output

C. Present state and next state

D. All present state, input, next state and output

2. A sequential circuit with m flip-flops and n inputs needs ________ rows in the state table.
A. 2^m
B. 2^n
C. 2^{m+n}
D. 2^{m-n}

3. In a state diagram, the states are represented by
A. Rectangles
B. Squares
C. Lines
D. Circles

4. In a basic flip flop circuit using NAND gates, if S=0 and R=0 are provided, then what will be values of Q and Q'?
A. 0,0
B. 0,1
C. 1,0
D. 1,1

5. In RS flip flop, when CP=1, S=1, and R=0, what do we call this state?
A. Set state
B. Reset state
C. Memory state
D. Invalid state

6. In a basic flip flop circuit using NOR gates, what values of S and R describe the undefined condition?
A. 0, 0
B. 0, 1
C. 1, 0
D. 1, 1

7. A basic flip-flop circuit can be made up of
A. Two NAND gates
B. Two NOR gates
C. Either two NAND gates or two NOR gates
D. None of the above

8. In D flip-flop, when CP=1 and D=0, the circuit will be in
A. Set state
B. Clear state

9. In D flip-flop, D input goes directly to _______ input and its complement goes to ______ input.
A. S, R
B. R, S

10. Which table lists the required inputs for a given change of state?
A. Input table
B. Characteristic table
C. Excitation table
D. None of the above

11. During the design of combinational circuit part, the unused states are taken as
A. 0
B. 1
C. X
D. None of the above

12. In applications which require the transfer the data, what kind of flip-flop is advisable to use?
A. D flip-flop
B. T flip-flop
C. JK flip-flop
D. None of the above

13. Which of the following circuit use the clock signal as one of the inputs for synchronization?
A. Asynchronous sequential circuit
B. Synchronous sequential circuit
C. Combinational circuit
D. None of the above

14. A flip-flop is a binary storage device which can store ______ bit of information.
A. Zero
B. One
C. Two
D. None of the above

15. If the circuit requires the storage elements, then the circuit can be described in terms of
A. Sequential circuit
B. Combinational circuit
C. Either sequential or combinational circuit
D. Both sequential and combinational circuits

Answers for Self Assessment

1.	D	2.	C	3.	D	4.	D	5.	A
6.	D	7.	C	8.	B	9.	A	10.	C
11.	C	12.	A	13.	B	14.	B	15.	A

Review Questions:

1. What is a sequential circuit? Explain its diagram and types.
2. What is a basic flip-flop circuit? How it can be constructed using different ways?
3. What is RS flip-flop? Explain its cases. Draw its truth table, excitation table and characteristic table.
4. What is JK flip-flop? Explain its cases. Draw its truth table, excitation table and characteristic table.
5. What is a state table? Explain its components with examples.
6. What is a characteristic table? Draw the characteristic table of JK, RS, D and T flip-flops.
7. Explain the design procedure of sequential circuits.
8. What is a counter? Explain 3-bit ripple binary counter.
9. Explain the counter with non-binary sequences.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education Asia, 2002.

Web Links

https://www.electronicsforu.com/technology-trends/learn-electronics/flip-flop-rs-jk-t-d

https://www.tutorialspoint.com/digital_circuits/digital_circuits_flip_flops.htm

Unit 05: Register Transfer and Micro-Operations

CONTENTS

Objectives

After studying this unit, you will be able to

- Understand the register transfer.
- Understand the register transfer language.
- Understand the bus line construction using multiplexers and three state buffers.
- Understand read and write operations.
- Understand the different micro-operations.

Introduction

Digital system design invariably uses a modular approach. Digital modules are defined by the registers they contain and the operations that are performed on the data stored in them. The internal hardware organization of a digital computer is best defined by specifying:Set of registers, sequence of micro-operations and control to initiate the sequence of micro-operations.

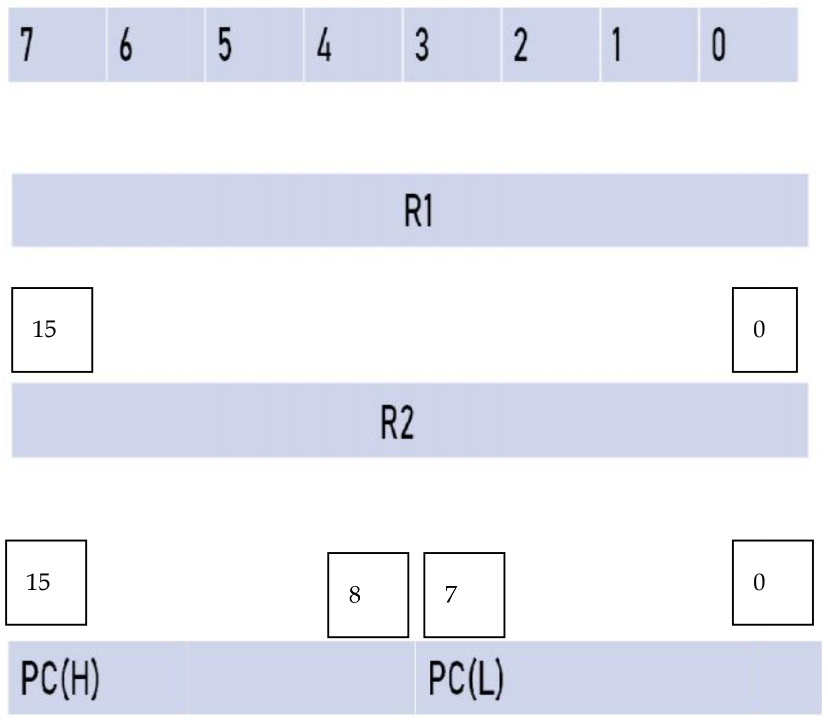
The symbolic notation used to describe the micro-operation transfers among registers is called a register transfer language. A register transfer language is a system for expressing in symbolic form the micro-operation sequences among the registers of a digital module.

5.1 Register Transfer

Computer registers are designated by capital letters to denote the function of a register. It is always better to use the capital letters for the functioning of registers.

MAR, PC, IR.

Block diagram of registers



Information transfers using registers

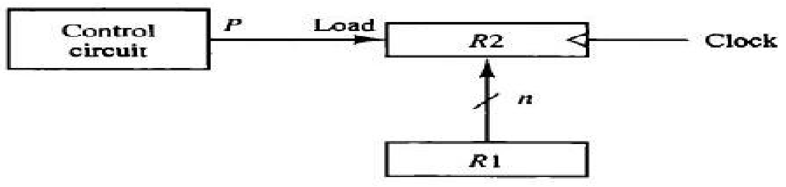
The information can be transferred from one register to another register. This is represented as:

R1 ← R2. The date is transferred from register R2 to R1.

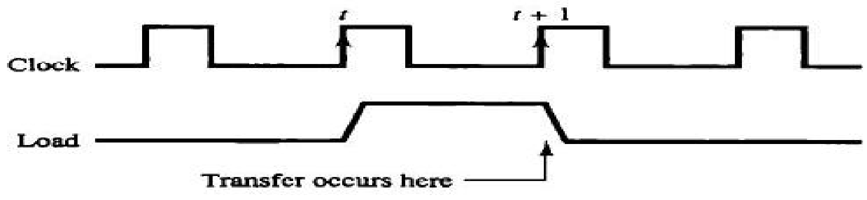
5.2 Transfer under a Predetermined Control Condition

If (P=1) then (R2 ‹– R1) **P: R2 ‹– R1**

Here the predetermined control condition is if P = 1, then only the data of register R1 should be transferred to register R2. Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.



(a) Block diagram



5.3 Basic Symbols for Register Transfers

T: R2<-R1,R1<-R2

Symbol	Description	Examples
Letters(and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of the register	R2(0=7), R2(L)
Arrow <-	Denotes transfer of information	R2<-R1
Comma ,	Separates two micro-operations	R2<-R1,R1<-R2

Bus and memory transfer

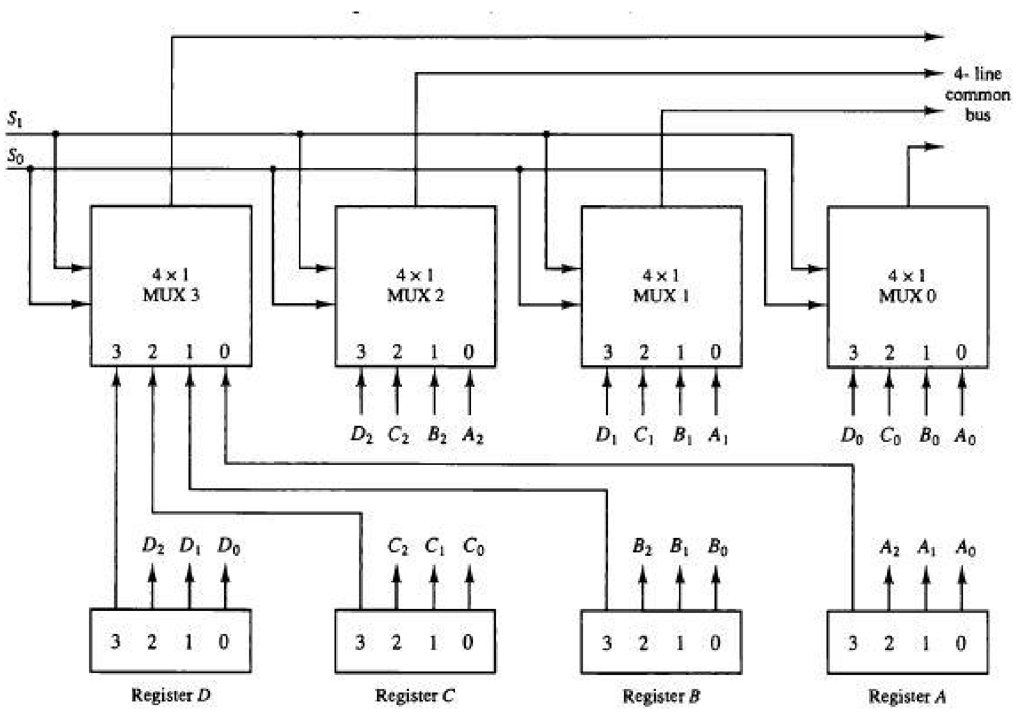
A more efficient scheme for transferring information between registers in a multiple register configuration is a common bus system. A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.

Construction of a common bus system

There are two ways to construct a common bus system:

1) With multiplexers
2) With three-state bus buffers

5.4 Construction using Multiplexers



Bus selection

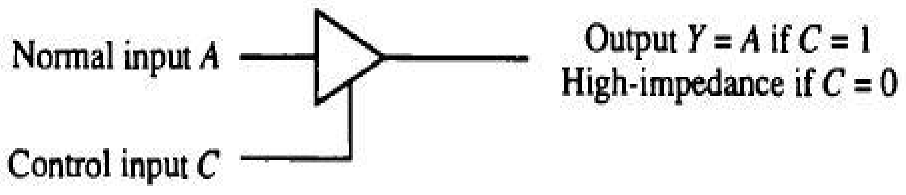
S1	S0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

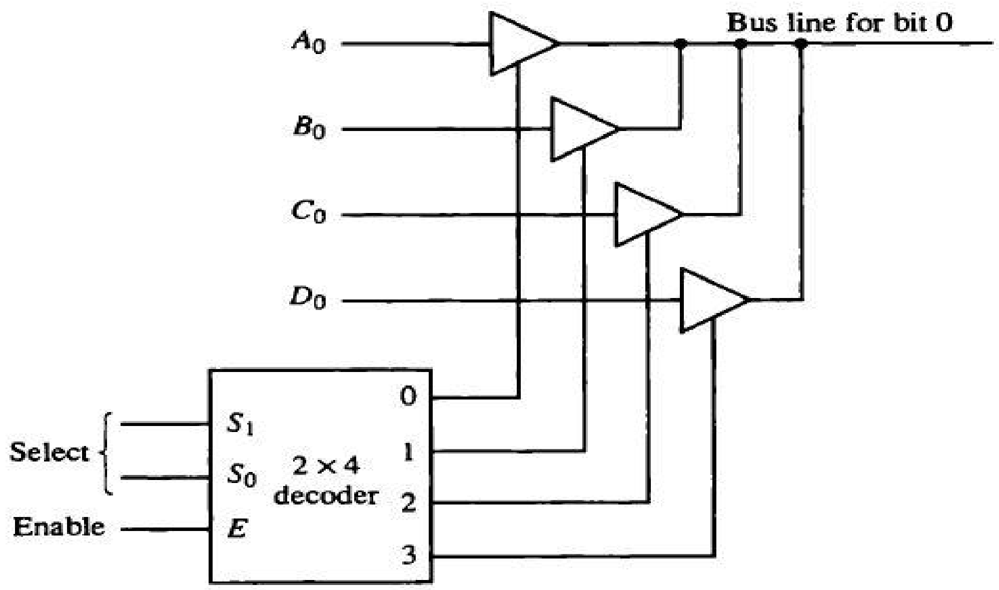
Information transfer

BUS <- C, R1 <- BUS.

5.5 Construction using Three State Bus Buffers

A three state bus buffer contains three states, i.e., logic 1, logic 0 and high impedance state.





5.6 Memory Transfer

The memory transfer contains read and write operations.Amemory word will be symbolized by the letter M.It is necessary to specify the address of M when writing memory transfer operations.

Memory read

The memory read operation is represented using the letter M.

Read: DR <-M[AR]. This representation indicates that the information from memory address register is read.

Memory write

The memory write operation is also represented using the letter M.

Write: M [AR] <- R1. This representation indicates that the information from R1 register is written to memory.

Types of Micro-operations

There are various types of micro-operations like

- Register Transfer Micro-operations
- Arithmetic Micro-operations
- Logical Micro-operations

- Shift Micro-operations

Register transfer micro-operation

These kinds of micro-operations don't change the information content.Other three types of micro-operations change the information change the information content during the transfer.

5.7 Arithmetic Micro-Operations

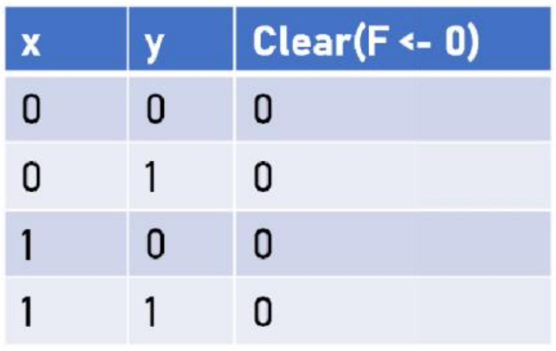
This set contains the basic arithmetic micro-operations, i.e.,

- Addition
- Subtraction
- Increment
- Decrement

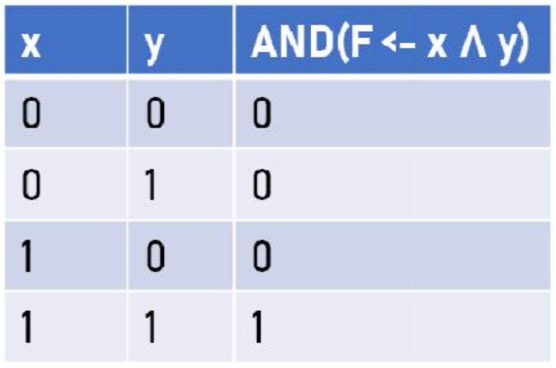
Arithmetic micro-operations	**Description**
R3 <- R1 + R2	Contents of R1 plus R2 are transferred to R3
R3 <- R1 - R2	Contents of R1 minus R2 are transferred to R3
R2 <- R2′	1′s complement the contents of R2
R2 <- R2′ + 1	2′s complement the contents of R2
R3 <- R1 + R2′ +1	Subtraction operation
R1 <- R1 + 1	Increments the contents of R1 by 1
R1 <- R1 - 1	Decrements the contents of R1 by 1

5.8 Logic Micro-Operations

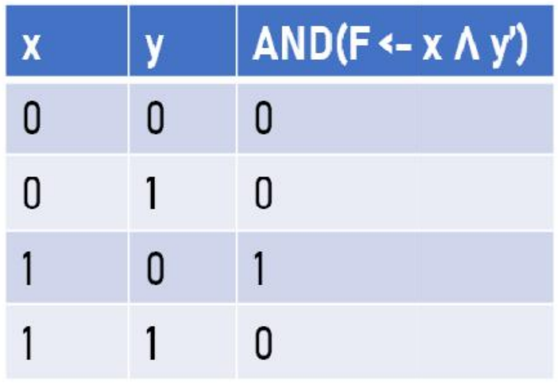
This set of micro-operations specifies binary operations for strings of bits stored in registers. For example, P:R1 <- R1 ⊕ R2. It specifies a logic micro-operation to be executed on the individual bits of the registers provided that the control variable P = 1.



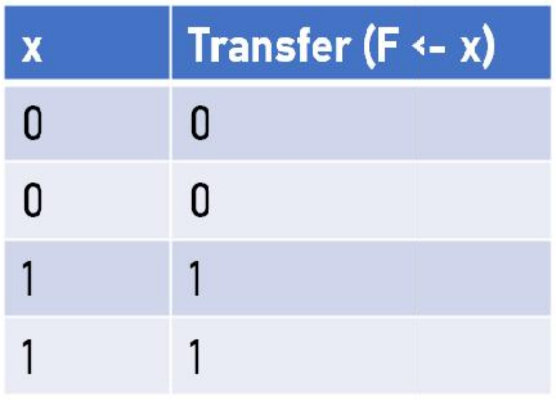
x	y	Clear(F <- 0)
0	0	0
0	1	0
1	0	0
1	1	0



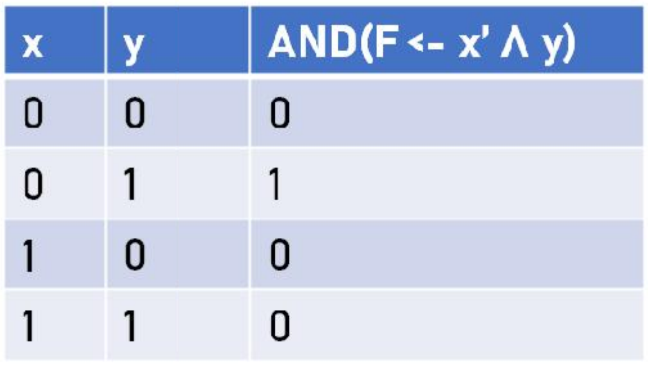
x	y	AND(F <- x ∧ y)
0	0	0
0	1	0
1	0	0
1	1	1



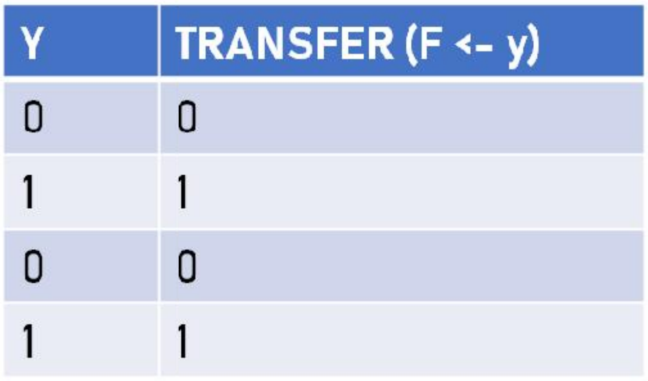
x	y	AND(F <- x ∧ y')
0	0	0
0	1	0
1	0	1
1	1	0



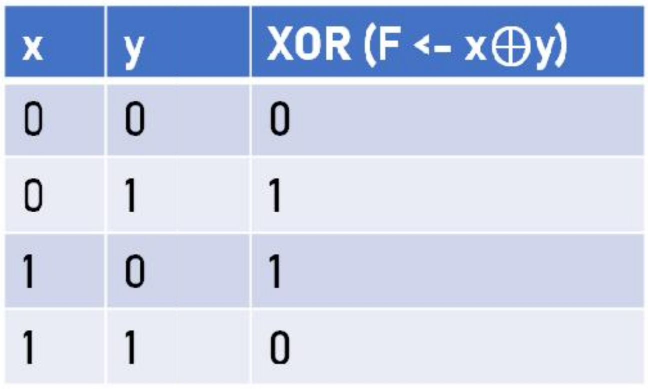
x	Transfer (F <- x)
0	0
0	0
1	1
1	1



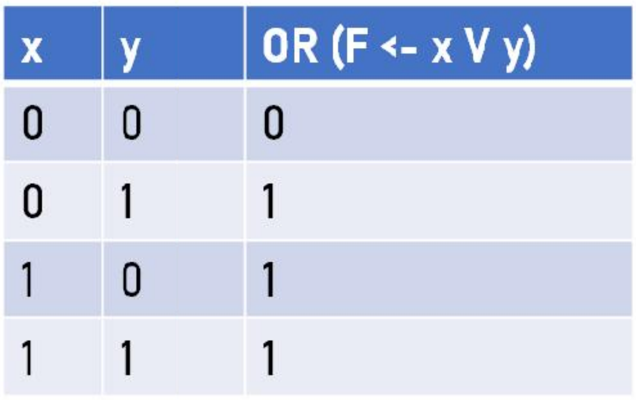
x	y	AND(F <- x' Λ y)
0	0	0
0	1	1
1	0	0
1	1	0



Y	TRANSFER (F <- y)
0	0
1	1
0	0
1	1



x	y	XOR (F <- x⊕y)
0	0	0
0	1	1
1	0	1
1	1	0



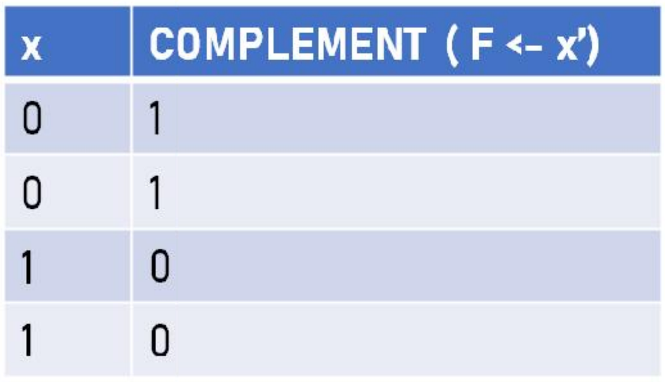
x	y	OR (F <- x V y)
0	0	0
0	1	1
1	0	1
1	1	1

x	y	NOR (F <- (x V y)’)
0	0	1
0	1	0
1	0	0
1	1	0

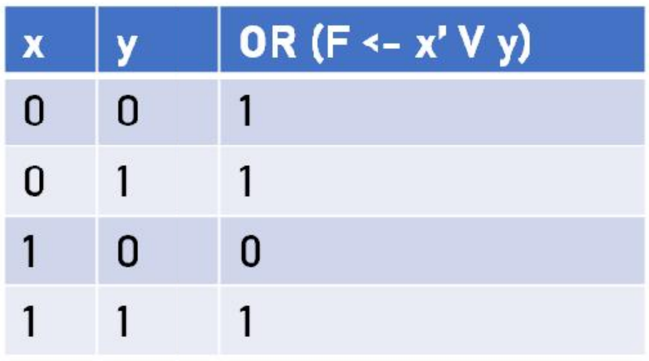
x	y	XNOR (F <- (x ⊕ y)’)
0	0	1
0	1	0
1	0	0
1	1	1

y	COMPLEMENT (F <- y’)
0	1
1	0
0	1
1	0

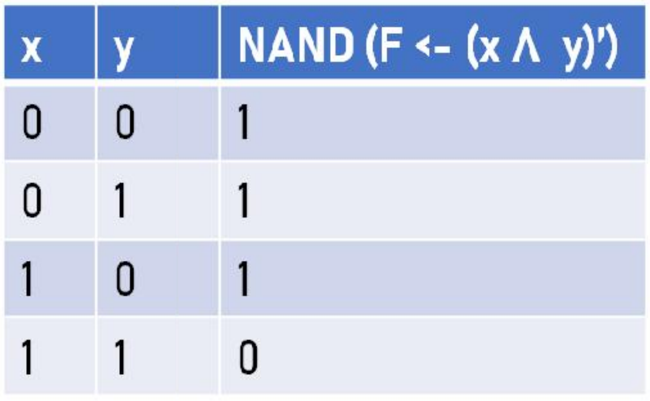
x	y	OR (F <- x V y’)
0	0	1
0	1	0
1	0	1
1	1	1



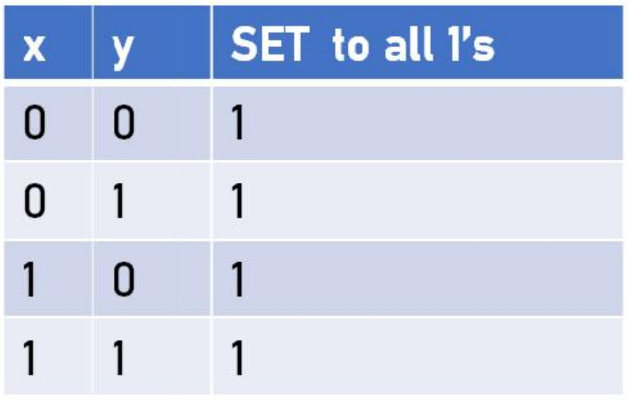
x	COMPLEMENT (F <- x')
0	1
0	1
1	0
1	0



x	y	OR (F <- x' V y)
0	0	1
0	1	1
1	0	0
1	1	1



x	y	NAND (F <- (x Λ y)')
0	0	1
0	1	1
1	0	1
1	1	0



x	y	SET to all 1's
0	0	1
0	1	1
1	0	1
1	1	1

5.9 Applications of Logic Micro-Operations

How the bits of one register (designated by A) are manipulated by logic micro-operations as a function of the bits of another register (designated by B).

Selective set

1010	A(Before)
1100	B(Logic Operand)
1110	A(After)

Selective complement

1010	A(Before)
1100	B(Logic Operand)
0110	A(After)

Selective Clear

1010	A(Before)
1100	B(Logic Operand)
0010	A(After)

Mask

0110 1010	A (Before)
0000 1111	B (Mask)
0000 1010	A (After masking)

5.10 Shift Micro-Operations

Shift micro-operations are used for serial transfer of data. There are three types of shifts: logical, circular, and arithmetic.

Logical shift

R1 <- shl R1

R2 <- shr R2

Circular shift

R <- cil R

R <- cir R

Arithmetic shift

R <- ashl R

R <- ashr R

Summary

- A register transfer language is a system for expressing in symbolic form the micro-operation sequences among the registers of a digital module.
- Computer registers are designated by capital letters to denote the function of a register.
- There are two ways to construct a common bus system with multiplexers and three-state bus buffers.
- There are various types of micro-operations like register Transfer Micro-operations, arithmetic Micro-operations, logical Micro-operations and shift Micro-operations.
- Applications of logic micro-operations are selective set, selective complement, selective clear and mask.

Keywords

Register transfer language:A system for expressing in symbolic form the micro-operation sequences among the registers of a digital module.

Three state bus buffers: It contains three states, i.e., logic 1, logic 0 and high impedance state.

Shift micro-operations: These are used for serial transfer of data. There are three types of shifts: logical, circular, and arithmetic.

Self Assessment

1. The internal hardware organization of a digital computer is best defined by specifying:

A. Set of registers

B. Sequence of micro-operations

C. A control to initiate the sequence of micro-operations

D. All registers, sequence of micro-operations and a control for initializing

2. Is it allowed to use the lower case letters for representation of computer registers?

A. Yes

B. No

3. The separation of two micro-operations is done using

A. Comma

B. Semi-colon

C. Colon

D. Asterisk

4. We can construct a common bus system for transferring of information between registers by using

A. Multiplexers

B. Three state bus buffers

C. Both multiplexers and three state bus buffers

D. None of the above

5. A three state gate exhibits

A. Logic 0 state, logic 1 state and logic 2 state

B. Logic 1 state, high-impedance state and logic 2 state

C. Logic 0 state, logic 1 state and high impedance state

D. None of the above

6. Which of the state behaves like an open circuit in three state gate?

A. Logic 0

B. Logic 1

C. High impedance state

D. None of the above

7. When the control input is ______, then the gate goes to high-impedance state.

A. 0

B. 1

C. 2

D. 3

8. We can employ _____________ to ensure that no more than one control input is active at any given time.

A. Adder

B. Multiplexer

C. Demultiplexer

D. Decoder

9. Which operation is defined in R3 ‹– R1 + R2′ + 1

A. Addition

B. Subtraction

C. Negation

D. None of the above

10. Which logical micro-operation is defined F‹–x ∧ y

A. OR

B. AND

C. XOR

D. XNOR

11. Which logical micro-operation is defined F‹–(x ⊕ y)′

A. XOR

B. XNOR

C. NAND

D. NOR

12. Which logical micro-operation is defined F‹–(x ∧ y)′

A. XOR

B. XNOR

C. NAND

D. NOR

13. Apply selective clear on A, A=1010, B=1100 (logic operand)

A. 1010

B. 1100

C. 0010

D. 0011

14. Apply selective complement on A, A=1010, B=1100 (logic operand)

A. 1010

B. 0110

C. 0011

D. 1111

15. M [AR] <- R1 is

A. Memory read operation

B. Memory write operation

Answer for Self Assessment

1.	D	2.	B	3.	A	4.	C	5.	C
6.	C	7.	A	8.	D	9.	B	10.	B
11.	B	12.	C	13.	C	14.	B	15.	B

Review Questions

1. How can we specify the internal hardware organization of a digital computer?
2. How do we represent the registers?
3. What are the two ways to construct a common bus system? Explain.
4. What are different types of micro-operations?
5. Write all the applications of logical micro-operations.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education Asia, 2002.

Web Links

https://vardhaman.org/wp-content/uploads/2018/03/CAO%20Unit-I%20part-3.pdf

http://www.nou.ac.in/econtent/BCA%20Part%20I/Paper%207/BCA%20Paper-VII%20Block-2%20Unit-6.pdf

Unit 06: Instruction Codes and Instruction Cycles

CONTENTS

Objectives: After studying this unit, you will be able to

- Understand the instruction codes and instruction cycles.
- Understand the use of registers in computers.
- Understand the common bus system for the connection of registers and memory of basic computers.
- Understand the control unit and control timing signals.
- Understand the instruction cycle and its phases.
- Understand the different types of instructions.

Introduction: The organization of the computer is defined by

1) Its internal organization,

2) The timing and control structure,

3) The set of instructions that it uses.

The internal organization of a digital system is defined by the sequence of micro-operations. These micro-operations are performed by the means of a program on a computer.

Program

The general purpose digital computer is capable of executing various micro-operations. A program is a set of instructions that specify the operations, operands, and the sequence by which processing has to occur.

6.1 Computer Instruction

A computer instruction is a binary code that specifies a sequence of micro-operations for the computer. The computer reads each instruction from memory and places it in a control register. Every computer has its own unique instruction set to perform various tasks.

Instruction Code

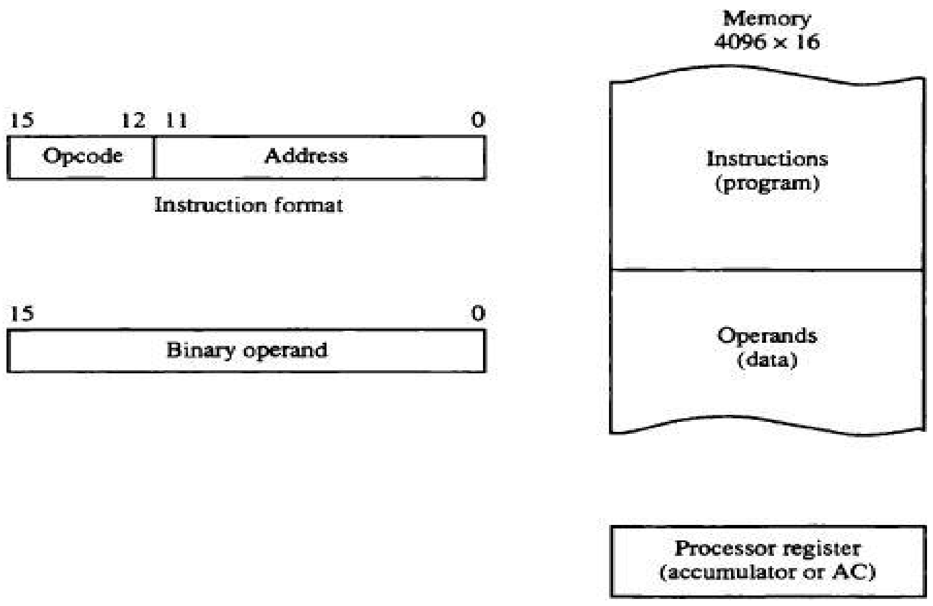
An instruction code is a group of bits that instruct the computer to perform a specific operation. The most basic part of an instruction code is its operation part.The operation code must consist of atleast *n* bits for a given 2^n (or less) distinct operations. This operation must be performed on some data stored in processor registers or in memory.

6.2 Stored Program Organization

It has one processor register and an instruction code format with two parts.

1st part: The operation to be performed

2nd part: An address.

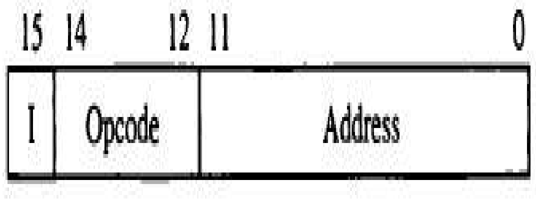


Accumulator

This is represented as AC. Computers that have a single-processor register. The operation is performed with the memory operand and the content ofAC.If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction can be used for other purposes.

Instruction code format

The memory unit has a capacity of 4096 words and each word contains 16 bits. Here, 12 bits represents the address of an operand, 3 bits indicates operation of the instruction and 1 bit differentiates between the direct or indirect address.



Addresses

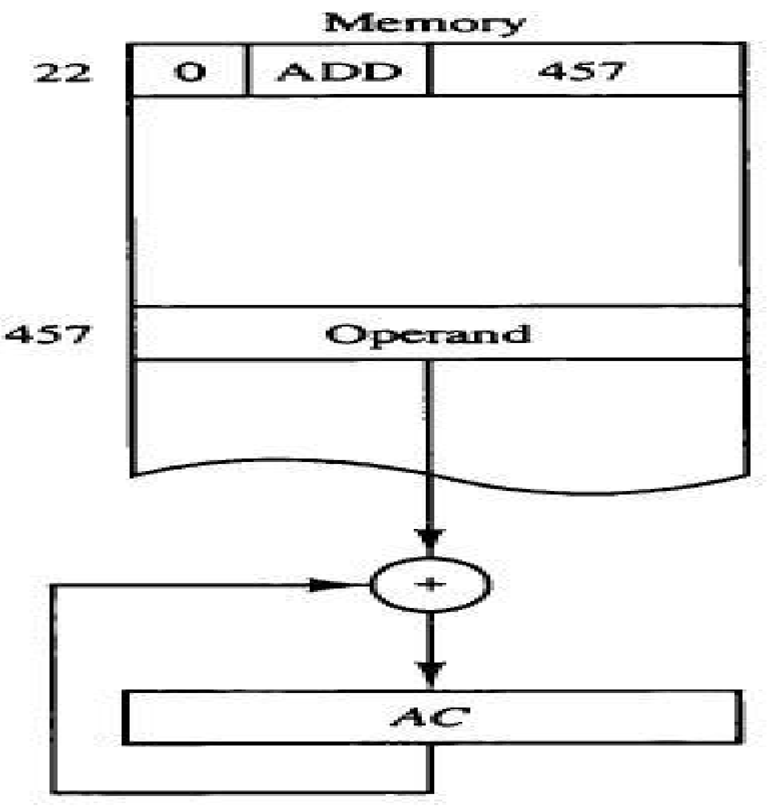
Address bits can contain actual operand, address of the operand or address of some memory word.

1) Immediate operand
2) Direct address
3) Indirect address

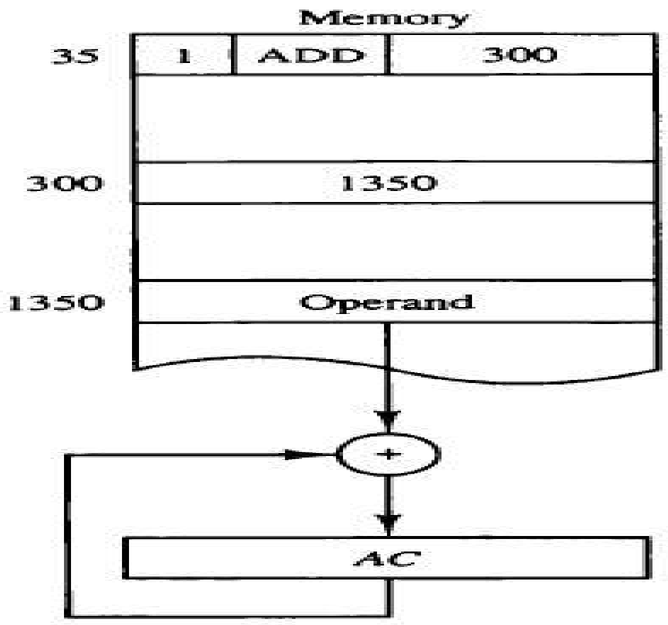
Direct and indirect address

- One bit of the instruction code can be used to distinguish between a direct and an indirect address, i.e., I.

Direct address



Indirect address



Effective address

- The address of the operand in a computation-type instruction or the target address in a branch-type instruction.

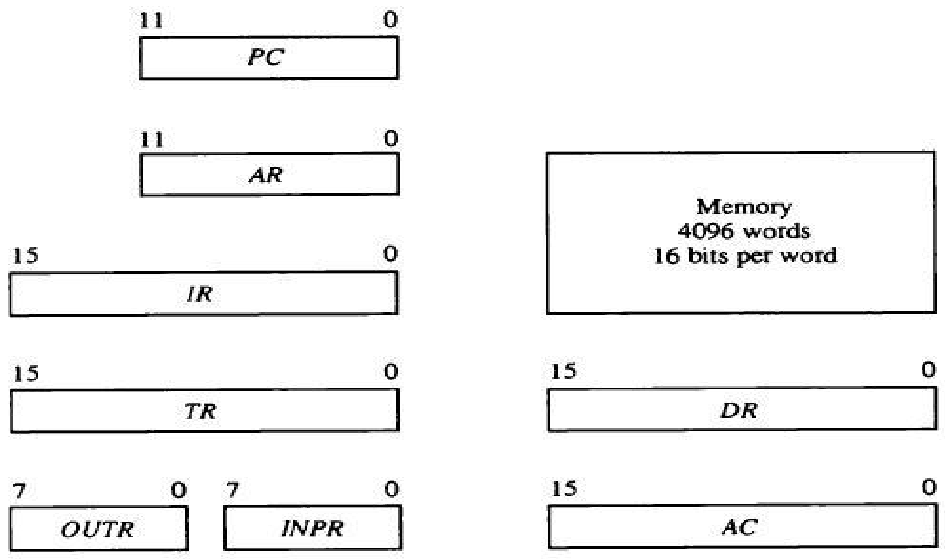
6.3 Computer Registers

These registers are stored in consecutive memory locations.The control reads an instruction from a specific address in memory and executes it. A counter is needed to calculate the address of the next instruction.

List of registers

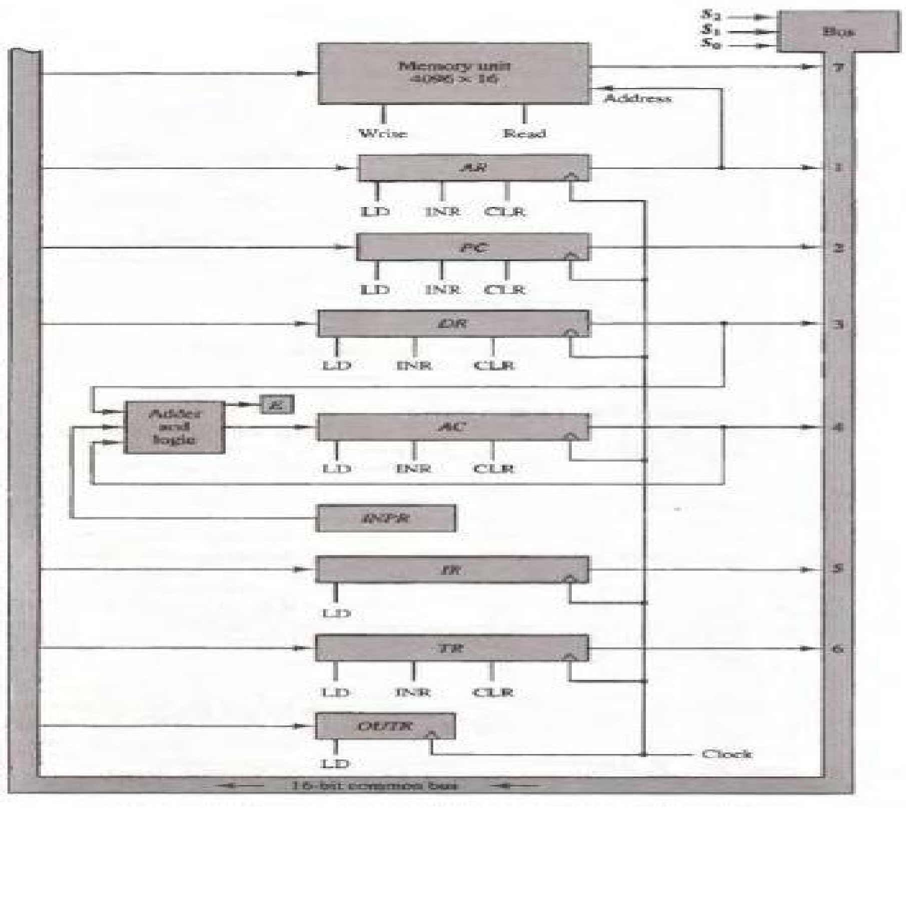
Register symbol	Number of bits	Register name	Function
DR	16	Data register	Holds memory operand
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

Basic computer registers and memory



Common bus system

The basic computer has eight registers, a memory unit, and a control unit. A more efficient scheme for transferring information in a system with many registers is to use a common bus. The connection of the registers and the memory of basic computer to a common bus system are shown. Five registers have three control inputs: LO (load), INR (increment), and CLR (clear). Two registers have only a LO input.



Execution of instructions

The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into *AC*. For example, the two Microoperations: *DR* <-*AC* and *AC* <-*DR* can be executed at the same time.

Timing and Control

The timing for all registers in the basic computer is controlled by a master clock generator.The clock pulses do not change the state of a register unless the register is enabled by a control signal.

6.4 Types of control organization

There are two major types of control organization:

1) Hardwired control

2) Micro-programmed control.

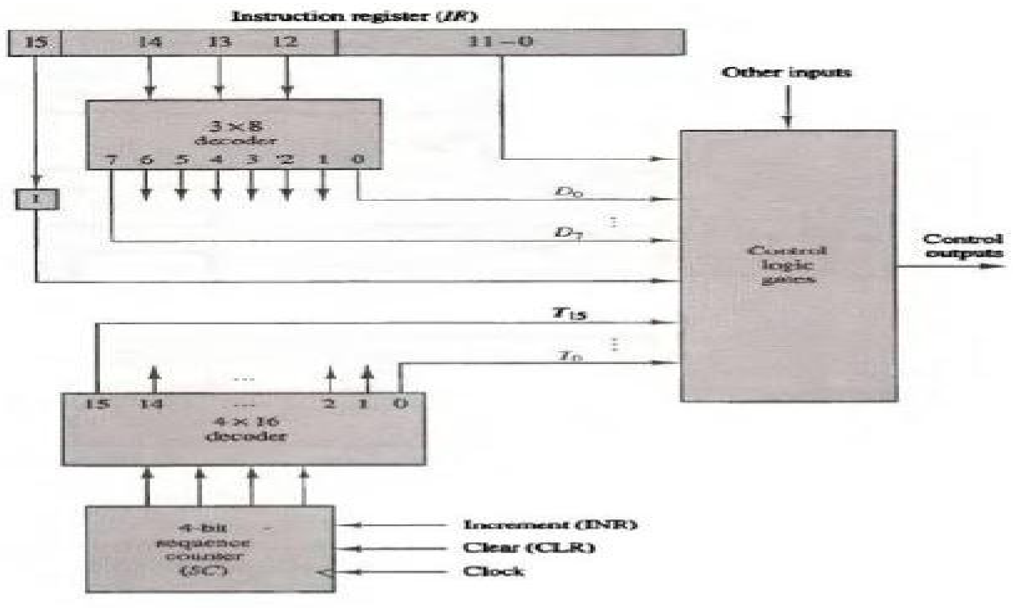
Hardwired Organization:

The control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It can be optimized to produce a fast mode of operation.

Micro-programmed Control

The control information is stored in a control memory.The control memory is programmed to initiate the required sequence of micro-operations.

Control unit of basic computer



Components:

A) Two decoders (4 * 16 decoder and 3 * 8 decoder),

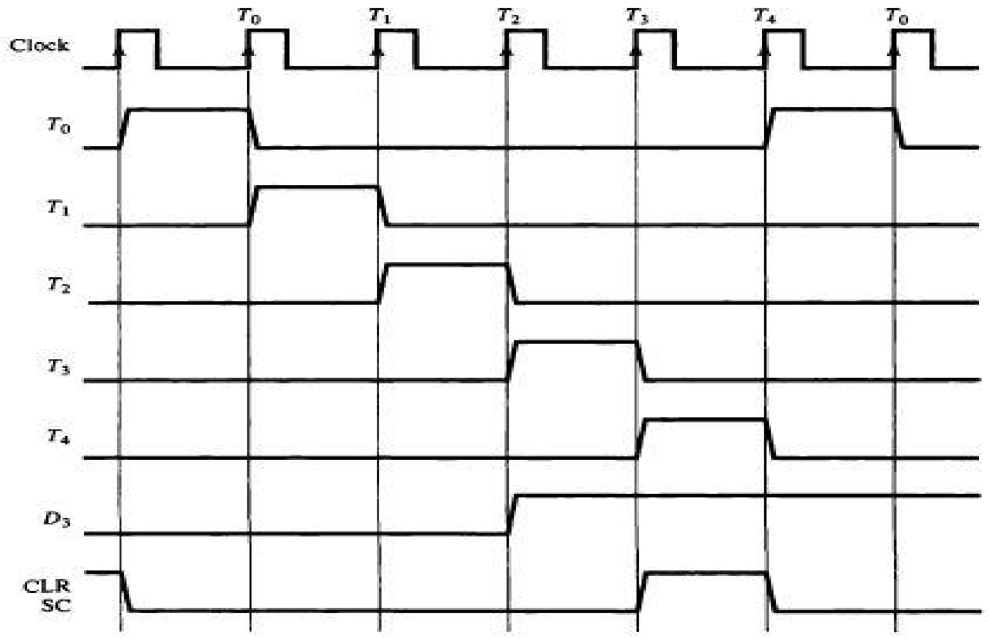
B) A 4 bit sequence counter,

C) A number of control logic gates.

An instruction read from memory is placed in the IR. It is divided into three parts: the *I* bit, the operation code, and bits 0 through 11. The 4-bit SC can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}. The SC can be incremented or cleared synchronously.Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in a while, the counter is cleared to 0, causing the next active timing signal to be T_0.As an example, consider the case where SC is incremented to provide timing signals T_0, T_1, T_2, T_3, and T_4 in sequence.At time T_4, SC is cleared to O if decoder output D_3 is active. This is expressed symbolically by the statement

D_3T_4: SC <- O

Timing control signals



The last three waveforms show how SC is cleared when D_3T_4 = I. Output D_3 from the operation decoder becomes active at the end of timing signal T_2. When timing signal T_4 becomes active, the output of the AND gate that implements the control function D_3T_4 becomes active. This signal is applied to the CLR input of SC. On the next positive clock transition (the one marked T_4 in the diagram) the counter is cleared to 0. This causes the timing signal T_0 to become active instead of T_5 that would have been active if SC were incremented instead of cleared.

Memory read and write

A memory read or writes cycle will be initiated with the rising edge of a timing signal.It will be assumed that a memory cycle time is less than the clock cycle time. According to this assumption, a memory read or writes cycle initiated by a timing signal will be completed by the time the next clock goes through its positive transition. The clock transition will then be used to load the memory word into a register. This timing relationship is not valid in many computers because the memory cycle time is usually longer than the processor clock cycle. In such a case it is necessary to provide wait cycles in the processor until the memory word is available. For example, the register transfer statementT_0: AR <- PCspecifies a transfer of the content of PC into AR if timing signal T_0 is active. T_0 is active during an entire clock cycle interval.

During this time the content of PC is placed onto the bus (with $S_2S_1S_0 = 010$) and the LD (load) input of AR is enabled. The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition.This same positive clock transition increments the sequence counter SC from 0000 to 0001. The next clock cycle has T_1 active and T_0 inactive.

Instruction Cycle

The program is executed in the computer by going through a cycle for each instruction.

6.5 Phases of Instruction Cycle

In the basic computer each instruction cycle consists of the following phases:

- Fetch an instruction from memory.
- Decode the instruction.
- Read the effective address from memory if the instruction has an indirect address.
- Execute the instruction.

Fetch and Decode

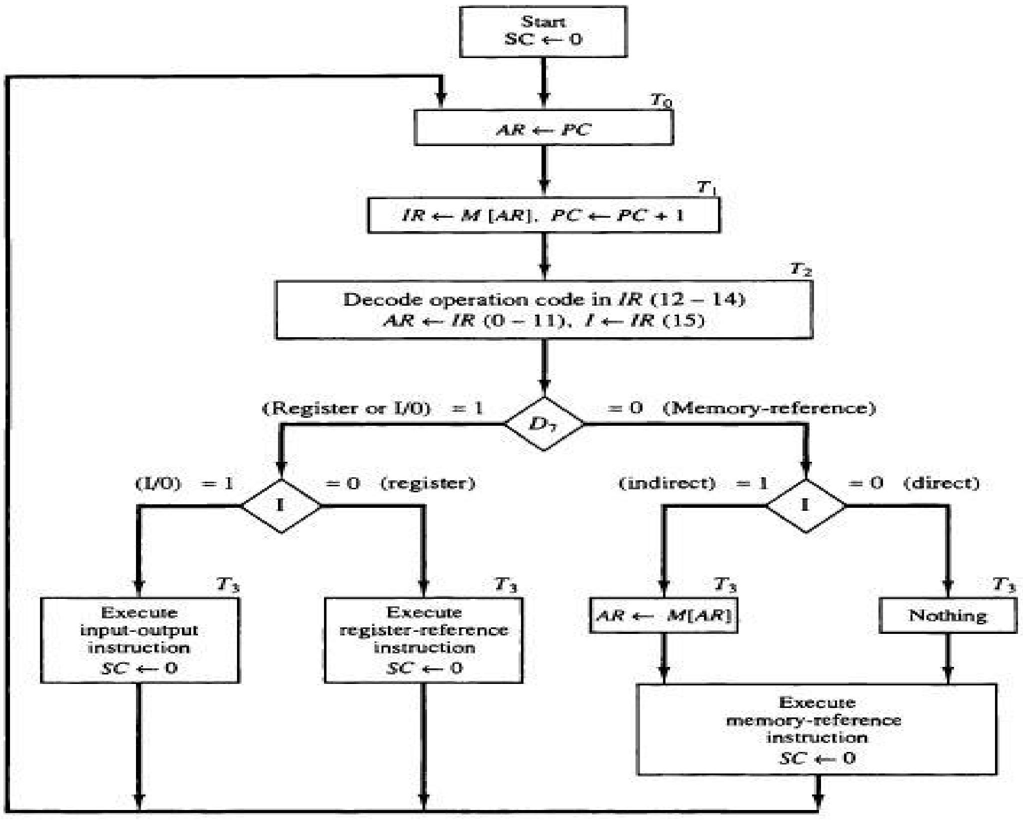
Initially, the PC is loaded with the address of the first instruction in the program. The sequence counter SC is cleared to 0, providing a decoded timing signal T_0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T_0, T_1, T_2, and so on. The micro-operations for the fetch and decode phases can be specified by the following register transfer statements.

T_0:	AR<-PC
T_1:	IR<-M[AR], PC<-PC + 1
T_2:	D0,………. D7 <- Decode IR(12-14),
	AR<-IR(0-11), I<-IR(15)

After Decoding

The timing signal that is active after the decoding is T_3. During time T_3, the control unit determines the type of instruction that was just read from the memory.

6.6 Flowchart for Instruction Cycle


Start
SC ← 0
T0
AR ← PC
T1
IR ← M [AR], PC ← PC + 1
T2
Decode operation code in IR (12 – 14)
AR ← IR (0 – 11), I ← IR (15)
(Register or I/O) = 1
D7
= 0 (Memory-reference)
(I/O) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T3
Execute input-output instruction SC ← 0
T3
Execute register-reference instruction SC ← 0
T3
AR ← M[AR]
T3
Nothing
Execute memory-reference instruction SC ← 0

6.7 Determine the type of instruction

Decoder output D7 is equal to 1 if the operation code is equal to binary 111.If D7=1, then it can be register-reference or input-output type instruction.If D7 = 0, then it is memory-reference instruction. Control then inspects the value of the first bit of the instruction, which is now available in flip-flop I.If D_7 = 0 and I = 1, we have a memory reference instruction with an indirect address. It is then necessary to read the effective address from memory.The micro-operation for the indirect address condition can be symbolized by the register transfer statement AR <- M[AR].

The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T_3.This can be symbolized as follows:

D_7' IT_3: AR <- M[AR], D_7'I'T_3: Nothing

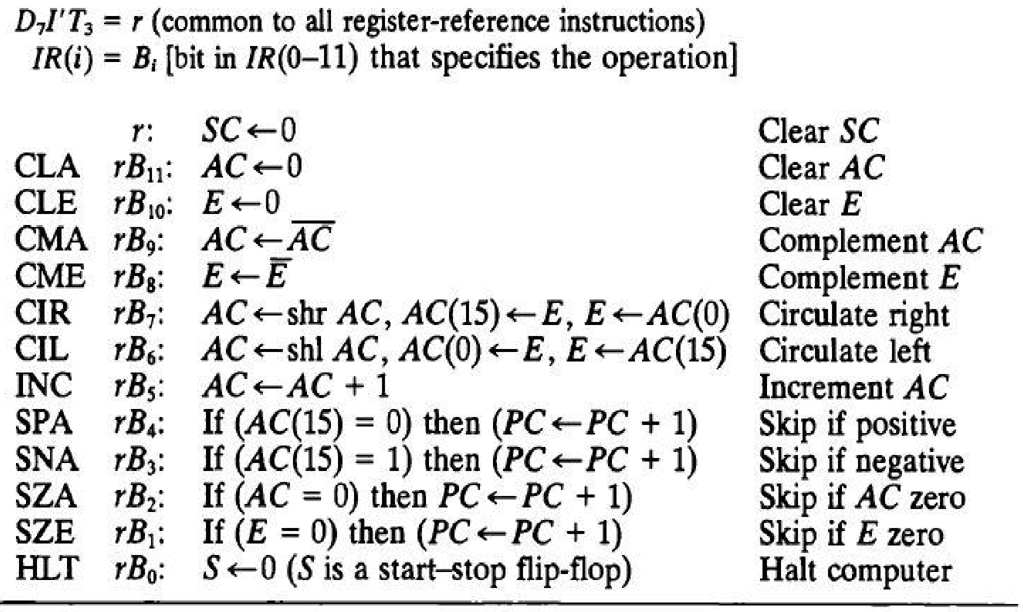
D_7I'T_3: Execute a register-reference instruction

D_7IT_3: Execute an input-output instruction

When a MRI with I = 0 is encountered, it is not necessary to do anything since the effective address is already in AR.However, SC must be incremented when D_7' T3 = 1, so that the execution of the MRI can be continued with timing variable T_4.A register-reference or I/O instruction can be executed with the clock associated with timing signal T_3. After the instruction is executed, SC is cleared to O and control returns to the fetch phase with T_0 =1.SC is either incremented or cleared to 0 with every positive clock transition.

Register-Reference Instructions

Register-reference instructions are recognized by the control when D_7= 1 and I=0.These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions. These 12 bits are available in IR(0-11). They were also transferred to AR during time T_2.

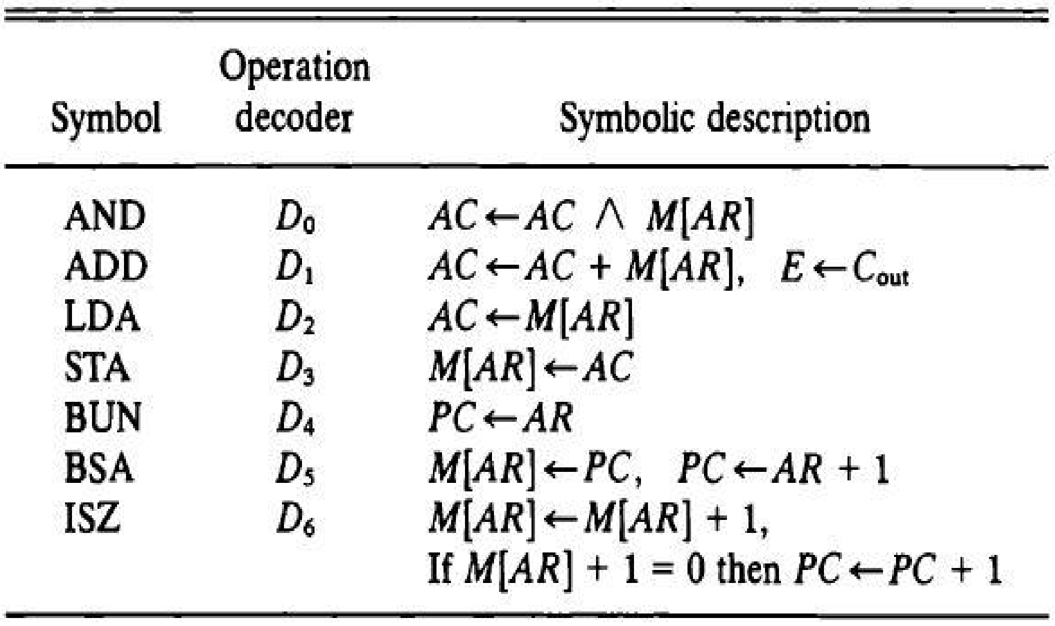


$D_7I'T_3 = r$ (common to all register-reference instructions)

$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

	r:	$SC \leftarrow 0$	Clear SC
CLA	rB_{11}:	$AC \leftarrow 0$	Clear AC
CLE	rB_{10}:	$E \leftarrow 0$	Clear E
CMA	rB_9:	$AC \leftarrow \overline{AC}$	Complement AC
CME	rB_8:	$E \leftarrow \overline{E}$	Complement E
CIR	rB_7:	$AC \leftarrow \text{shr } AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$	Circulate right
CIL	rB_6:	$AC \leftarrow \text{shl } AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$	Circulate left
INC	rB_5:	$AC \leftarrow AC + 1$	Increment AC
SPA	rB_4:	If $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	rB_3:	If $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative
SZA	rB_2:	If $(AC = 0)$ then $PC \leftarrow PC + 1)$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start–stop flip-flop)	Halt computer

Memory Reference Instructions



Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR],\quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC,\quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$, If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

Memory Reference Instructions - Effective Address

The decoded output D_i for i = 0, 1, 2, 3, **4,** 5, and 6 from the operation decoder that belongs to each instruction.The effective address of the instruction is in the address register AR and was placed there during timing signal T_2 when I = 0, or during timing signal T_3, when I = 1. The execution of the memory-reference instructions starts with timing signal T_4.

Memory Reference Instructions- AND to AC

D_0T_4: DR<-M[AR]

D_0T_5 : AC<-AC/\DR, SC<-0

Memory Reference Instructions- ADD to AC

D_1T_4: DR <- M[AR]

D_1T_5: AC <- AC + DR, E <- C_{out}, SC <- 0

Memory Reference Instructions- Load to AC

D_2T_4: DR <- M[AR]

D_2T_5: AC <- DR, SC <- 0

Memory Reference Instructions- Store AC

D_3T4: M[AR] <- AC, SC <- 0

Memory Reference Instructions- Branch Unconditionally

This instruction transfers the program to the instruction specified by the effective address.

D_4T_4: PC <- AR, SC <-0

Memory Reference Instructions- Branch and Save Return Address

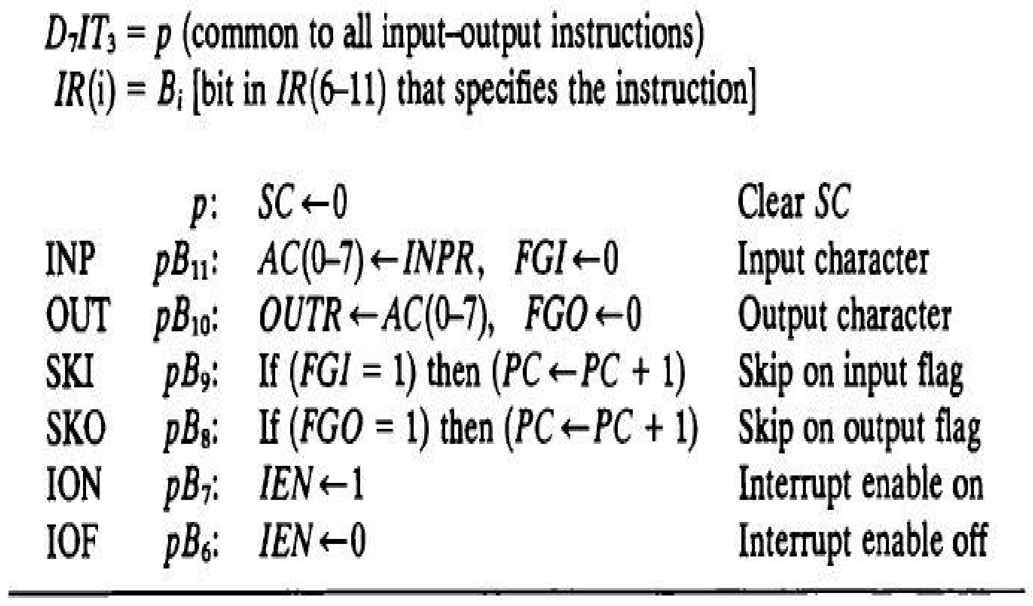
M[AR] <- PC, PC <- AR + I

Memory Reference Instructions-Increment and Skip if 0

This instruction increment the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1.

Input and Output Instructions

Input and output instructions are needed for transferring information to and from AC register, for checking the flag bits, and for controlling the interrupt facility. Input-output instructions have an operation code 1111 and are recognized by the control when D_7= 1 and I = 1. The remaining bits of the instruction specify the particular operation.



$D_7IT_3 = p$ (common to all input–output instructions)

$IR(i) = B_i$ [bit in $IR(6–11)$ that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$	Output character
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

Summary

- The internal organization of a digital system is defined by the sequence of micro-operations.
- A program is a set of instructions that specify the operations, operands, and the sequence by which processing has to occur.
- The computer reads each instruction from memory and places it in a control register.
- An instruction code is a group of bits that instruct the computer to perform a specific operation.
- A more efficient scheme for transferring information in a system with many registers is to use a common bus.
- The clock pulses do not change the state of a register unless the register is enabled by a control signal.
- An instruction read from memory is placed in the IR.
- The SC can be incremented or cleared synchronously.

Keywords

- Program: It is a set of instructions that specify the operations, operands, and the sequence by which processing has to occur.
- Clock pulses: These do not change the state of a register unless the register is enabled by a control signal.
- Indirect address: The micro-operation for the indirect address condition can be symbolized by the register transfer statement: AR <- M[AR].
- Branch unconditionally: This instruction transfers the program to the instruction specified by the effective address.
- Input and output instructions: These are needed for transferring information to and from AC register, for checking the flag bits, and for controlling the interrupt facility.

Self Assessment

1. The computer reads each instruction from memory and places it in a ______ register.

A. Control

B. Processor

C. Either control or processor

D. None of the above

2. The operation code must consist of atleast *n* bits for a given 2^n (or less) distinct operations.

A. n-1, 2^{n-1}

B. n, 2^n

C. n+1, 2^{n+1}

D. None of the above

3. The instruction code format have two parts, the first part represents.

A. The operation to be performed

B. An address of operand

C. Either operation or address

D. None of the above

4. What is the total number of bits in an instruction?

A. 8

B. 12

C. 16

D. 24

5. How many bits are used to distinguish between direct and indirect address?

   A. 1

   B. 2

   C. 4

   D. 8

6. What is the total number of bits in PC?

A. 8

B. 12

C. 16

D. None of the above

7. What is the total number of bits in TR?

A. 8

B. 12

C. 16

D. None of the above

8. What is the total number of bits in OUTR?

A. 8

B. 12

C. 16

D. None of the above

9. Which of these register holds the address of instruction?

A. PC

B. AC

C. AR

D. DR

10. In a common bus system, how many registers just have LO input?

A. 1

B. 2

C. 3

D. 4

11. A ____ bit sequence counter can count in binary from 0 to 15.

A. 2

B. 4

C. 6

D. 8

12. Memory read/write cycle will be initiated with the __________ edge of a timing signal.

A. Falling

B. Rising

13. During each instruction cycle, what is the next phase after decoding of instruction is

A. Fetching of instruction from memory

B. Reading of effective address from memory

C. Execution of instruction

D. None of the above

14. If D7=0, then what type of instruction it specifies?

A. Register reference

B. Input-output

C. Memory Reference

D. None of the above

15. If D7=1 and I=1, then what type of instruction it specifies?

A. Register reference

B. Input-output

C. Memory Reference

D. None of the above

Answers for Self Assessment

1.	A	2.	B	3.	A	4.	C	5.	A
6.	B	7.	C	8.	A	9.	A	10.	B
11.	B	12.	B	13.	B	14.	C	15.	B

Review Questions:

1. What is stored program organization? Explain every component of it.
2. What are the kinds of addresses used in computer organization?
3. Differentiate between direct and indirect addresses with examples.
4. List all the computer registers.
5. What is common bus system? Explain its components.
6. What are the major types of control organizations?
7. What is an instruction cycle? Write about its phases.
8. What are memory reference and register reference instructions?
9. What are input-output instructions?

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education Asia, 2002.

Unit 07: Machine Language

CONTENTS

Objectives: **After studying this unit, you will be able to:**

- understand the machine language
- understand the assembly language and
- understand the translation process in two passes
- understand the program loops
- understand the pointer counter

Introduction:

A computer system includes both hardware and software. The hardware components are composed of all the physical components of a computer whereas the softwares are the programs for the computer. Both of these components have a great influence on each other.

Those concerned with computer architecture should have knowledge of both hardware and software because of the impact of these two branches. The computers use alphanumeric character set for machine instructions. Machine instructions inside the computer form a binary pattern which is difficult for people to work with and understand. It is preferable to work with the more familiar symbols of the alphanumeric character set.

Machine dependence/independence: A program written by a user may be either dependent or independent of the physical computer that runs this program.

Machine Language

There are various types of programming languages that one may write for a computer, but the computer can execute programs when they are represented internally in binary form.

7.1 Categories of machine language

Programs written for a computer may be in one of the following categories:

A) Binary code: A sequence of instructions and operands in binary form.

B) Octal or hexadecimal code: An equivalent translation of the binary code to octal or hexadecimal representation.

C) Symbolic code: The user employs symbols (letters, numerals or special characters) for the operation part, the address part and other parts of the instruction code.

D) High-level programming languages: An example of a high level language is Fortran. It employs problem oriented symbols or formats. The program is written in a sequence of statements in a form that people prefer to think in when solving a problem.

7.2 Computer Instructions

There are 25 instructions of the basic computer. Each instruction provides a three letter symbol to facilitate writing symbolic programs.

Symbol	Hexadecimal code	Description
AND	0 or 8	AND M to AC
ADD	1 or 9	ADD M to AC, carry to E
LDA	2 or A	Load AC from M
STA	3 or B	Store AC in M
BUN	4 or C	Branch unconditionally to m
BSA	5 or D	Save return address in m

		and branch to m+1
ISZ	6 or E	Increment M and skip if zero
CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right E and AC
CIL	7040	Circulate left E and AC
INC	7020	Increment AC
SPA	7010	Skip if AC is positive
SNA	7008	Skip if AC is negative
SZA	7004	Skip if AC is zero
SZE	7002	Skip if E is zero
HLT	7001	Halt computer
INP	F800	Input information and clear flag
OUT	F400	Output information and clear flag
SKI	F200	Skip if input flag is on

SKO	F100	Skip if output flag is on
ION	F080	Turn interrupt on
IOF	F040	Turn interrupt off

A Binary program to add two numbers

Location	Instruction code			
0	0010	0000	0000	0100
1	0001	0000	0000	0101
10	0011	0000	0000	0110
11	0111	0000	0000	0001
100	0000	0000	0101	0011
101	1111	1111	1110	1001
110	0000	0000	0000	0000

Hexadecimal Program to add two numbers

Location	**Instruction**
000	2004
001	1005
002	3006
003	7001

004	0053
005	FFE9
006	0000

7.3 Program with Symbolic Operation Codes

Location	Instruction	Comments
000	LDA 004	Load first operand into AC
001	ADD 005	Add second operand to AC
002	STA 006	Store sum in location 006
003	HLT	Halt computer
004	0053	First operand
005	FFE9	Second operand (negative)
006	0000	Store sum here

7.4 Assembly Language Program to Add Two Numbers

ORG 0	/Origin of program is location 0
LDA A	/Load operand from location A
ADD B	/Add operand from location B

STA C	/Store sum in location C
HLT	/Halt computer
A, DEC 83	/Decimal operand
B, DEC -23	/Decimal operand
C, DEC 0	/Sum stored in location C
END	/End of symbolic symbol

Here we replaced each hexadecimal address by the symbolic address and each hexadecimal operand by a decimal operand. This is convenient because one usually does not know exactly the numeric memory location of operands while writing the program. The decimal numbers are more familiar than their hexadecimal equivalents.

7.5 Fortran program to add two numbers

INTEGER A, B, C
DATA A,83   B,-23
C=A+B
END

Assembly language

A programming language is defined by the set of rules. Users must conform with all the format rules of the language if they want their program to be translated correctly.

Almost every computer has its own particular assembly language. The basic unit of an assembly language program is a line of code. The specific language is defined by a set of rules that specify the symbols that can be used and how they may be combined to form a line of code.

7.6 Rules of the Language

Each line of the assembly language program is arranged in three columns called fields. The field specify the following information:

A) Label

B) Instruction

C) Comment

A) Label: This field may be empty or it may specify a symbolic address.

B) Instruction: This field specifies a machine instruction or pseudo-instruction.

C) Comment: This field may be empty or it may include a comment.

7.7 Label field-Symbolic address

A symbolic address consists of one, two, or three, but not more than three alphanumeric characters. The first character must be a letter and the next two may be letters or numerals. The symbol can be chosen arbitrarily by the programmer.

Recognition of Label field:

A symbolic address in the label field is terminated by a comma so that it will be recognized as a label by the assembler.

7.8 Instruction field

The instruction field in an assembly language program may specify one of the following items:

1. Memory reference instruction (MRI)
2. A register reference or input-output instruction (Non-MRI)
3. A pseudo-instruction with or without operand

7.9 Memory reference instruction (MRI)

A memory reference instruction occupies two or three symbols separated by spaces. The first must be a three letter symbol defining an MRI operation code. The second is a symbolic address. The third symbol is I which may or may not be present.

I (Present) – Indirect address instruction.

I (Absent) – Direct address instruction.

7.10 Non-MRI

This instruction does not have an address part.

CLA	Non-MRI
ADD OPR	Direct address MRI
ADD PTR I	Indirect address MRI

First three letter symbols in each line must be an instruction symbol of the computer.

7.11 Instruction field-Pseudo-instruction

It is not a machine instruction but rather an instruction to the assembler giving information about some phase of the translation.

Symbol	Information for the assembler
ORG N	Hexadecimal number N is the memory location for the instruction or operand
END	Denotes the end of the symbolic program
DEC N	Signed decimal number N to be converted to binary
HEX N	Hexadecimal number N to be converted to binary

- ORG N: The ORG informs the assembler that the instruction or operand in the following line is to be placed in a memory location specified by the number next to ORG.
- It is possible to use ORG more than once in a program to specify more than one segment of the memory.
- END: It is placed at the end of the program to inform the assembler that the program has been terminated.

Comment

These are helpful in understanding the step by step procedure taken by the program. These are inserted for explanation purpose. These are neglected during the binary translation purposes.

Recognition of a comment:

A comment must be preceded by a slash for the assembler to recognize the beginning of the comment field.

Assembly Language Program to Subtract Two Numbers

	ORG 100	/origin of the program is location 100
	LDA SUB	/Load subtrahend to AC
	CMA	/Complement AC
	INC	/Increment AC
	ADD MIN	/Add minuend to AC
	STA DIF	/Store difference
	HLT	/Halt computer
MIN,	DEC 83	/Minuend
SUB,	DEC -23	/Subtrahend
DIF,	HEX 0	/Difference stored here
	END	/End of symbolic program

ORG: Define the origin of the program at memory location $(100)_{16.}$ Next 6 lines define machine instructions. Last 4 lines have pseudo-instructions. Three symbolic addresses have been used and each is listed in column 1 as a label and in column 2 as an address of a MRI. Three of pseudo-instruction use operands and last one signifies the END of the program.

Procedure of Subtraction:

The subtraction is performed by adding the minuend to the 2's complement of the subtrahend. The subtrahend is a negative number. It is converted into a binary number in signed-2's complement representation because we dictate that all negative numbers be in their 2's complement form. When the 2's complement of the subtrahend is taken (by complementing and incrementing the AC), -23 converts to +23 and the difference is 83 + (2's complement of -23) = 83+ 23 = 106.

7.12 Translation to binary

The translation into binary from the symbolic program is done by a special program called an assembler. The input symbolic program is called the source program and the resulting binary program is called the object program. It is program that operates on character strings and produces an equivalent binary interpretation. The tasks performed by the assembler will be better understood if we first perform the translation on paper. The translation may be done by scanning the program and replacing the symbols by their machine code binary equivalent. Starting from the first line, we encounter an ORG pseudo-instruction which tells us to start the binary program at hexadecimal location 100. Second line, i.e., LDA SUB: It has two symbols. It must be an MRI to be placed in location 100. Since the letter I is not here, the first bit of the instruction code must be 0. The symbolic name of the operation is LDA. The first hexadecimal digit of the instruction should be 2.

Symbol	Hexadecimal code	Description
AND	0 or 8	AND *M* to *AC*
ADD	1 or 9	Add *M* to *AC*, carry to *E*
LDA	2 or A	Load *AC* from *M*
STA	3 or B	Store *AC* in *M*
BUN	4 or C	Branch unconditionally to *m*
BSA	5 or D	Save return address in *m* and branch to $m + 1$
ISZ	6 or E	Increment *M* and skip if zero
CLA	7800	Clear *AC*
CLE	7400	Clear *E*
CMA	7200	Complement *AC*
CME	7100	Complement *E*
CIR	7080	Circulate right *E* and *AC*
CIL	7040	Circulate left *E* and *AC*
INC	7020	Increment *AC*,
SPA	7010	Skip if *AC* is positive

The binary value of the address part must be obtained from the address symbol SUB. We scan this label column and find this symbol in line 9. To determine its hexadecimal value we note that the line 2 contains an instruction for location 100 and every other line specifies a machine instruction or an operand for sequence memory locations. Label SUB in line 9 corresponds to memory location 107. When the two parts of the instruction are assembled, we obtain the hexadecimal code 2107. In the same way, we translate the other lines representing the machine instructions. Two lines in the symbolic program specify the decimal operands with the pseudo-instruction DEC. A third specifies a zero by means of a HEX pseudo-instruction (DEC could be used as well). Decimal 83 is converted to binary and placed in location 106 in its hexadecimal equivalent. Decimal -23 is a negative number and must be converted into binary in signed-2's complement form.

Translation to binary-Address Symbol Table

The translation process can be simplified if we scan the entire symbolic program twice. No translation is done during the first scan. We assign a memory location to each machine instruction and operand. This will facilitate the translation process during the second scan. We assign location 100 to the first instruction after ORG. We then assign sequential locations for each line of the code that has a machine instruction or operand up to the end of the program. ORG and END are not assigned a numerical location because they do not represent an instruction or an operand. When the first scan is completed, we associate with each label its location number and form a table that defines the hexadecimal value of each symbolic address.

For this program, the address symbol table is as follows:

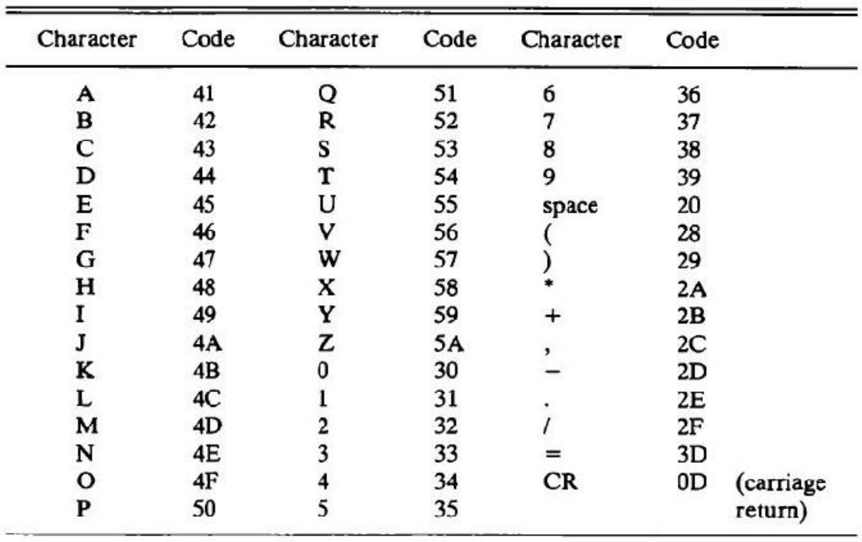
Address Symbol	Hexadecimal address
MIN	106
SUB	107
DIF	108

During the second scan of the symbolic program we refer to the address symbol table to determine the address value of a memory reference instruction. For example, the line of code LDA SUB is translated during the second scan by getting the hexadecimal value of LDA and the hexadecimal value of SUB from the address-symbol table. We then assemble the two parts into a four-digit hexadecimal instruction. The hexadecimal code can be easily converted to binary. When the translation from symbols to binary is done by an assembler program, the first scan is called the first pass, and the second is called the second pass.

7.13 Representation of Symbolic Program in Memory

Prior to starting the assembly process, the symbolic program must be stored in memory. The user types the symbolic program on a terminal. A loader program is used to input the characters of the symbolic program into memory. Since the program consists of symbols, its representation in memory must use an alphanumeric character code. In the basic computer, each character is represented by an 8-bit code. The high-order bit is always 0 and the other seven bits are as specified by ASCII.

7.14 Hexadecimal Character Code



Character	Code	Character	Code	Character	Code
A	41	Q	51	6	36
B	42	R	52	7	37
C	43	S	53	8	38
D	44	T	54	9	39
E	45	U	55	space	20
F	46	V	56	(	28
G	47	W	57	)	29
H	48	X	58	*	2A
I	49	Y	59	+	2B
J	4A	Z	5A	,	2C
K	4B	0	30	−	2D
L	4C	1	31	.	2E
M	4D	2	32	/	2F
N	4E	3	33	=	3D
O	4F	4	34	CR	0D (carriage return)
P	50	5	35		

7.15 Representation of Symbolic Program in Memory

Each character is assigned two hexadecimal digits which can be easily converted to their equivalent 8-bit code. The last entry in the table does not print a character but is associated with the physical movement of the cursor in the terminal. The code for CR is produced when the return key is depressed. This causes the "carriage" to return to its initial position to start typing a new line. The assembler recognizes a CR code as the end of a line of code.

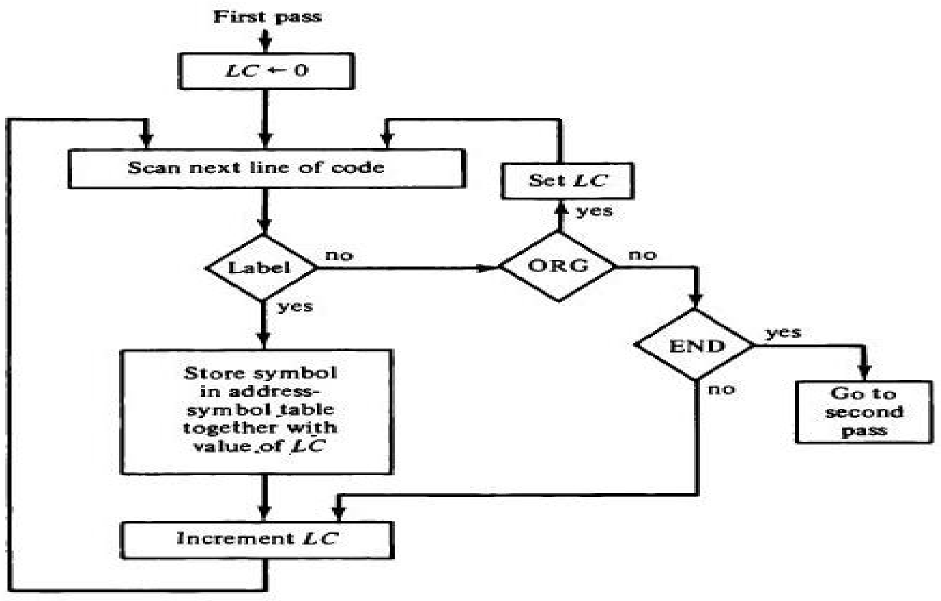
A line of code is stored in consecutive memory locations with two characters in each location, i.e., a word, since a memory word has a capacity of 16 bits. A label symbol is terminated with a comma. Operation and address symbols are terminated with a space. The end of the line is recognized by the CR code. For example, the following line of code: PL3,LDA SUBI

Memory word	Symbol	Hexadecimal code	Binary representation
1	P L	50 4C	0101 0000 0100 1100
2	3 ,	33 2C	0011 0011 0010 1100
3	L D	4C 44	0100 1100 0100 0100
4	A	41 20	0100 0001 0010 0000
5	S U	53 55	0101 0011 0101 0101
6	B	42 20	0100 0010 0010 0000
7	I CR	49 0D	0100 1001 0000 1101

The label PL3 occupies two words and is terminated by the code for comma (2C). The instruction field in the line of code may have one or more symbols. Each symbol is terminated by the code for space (20) except for the last symbol, which is terminated by the code of carriage return (OD). If the line of code has a comment, the assembler recognizes it by the code for a slash (2F). The assembler neglects all characters in the comment field and keeps checking for a CR code. When this code is encountered, it replaces the space code after the last symbol in the line of code. The input for the assembler program is the user's symbolic language program in ASCII. This input is scanned by the assembler twice to produce the equivalent binary program. The binary program constitutes the output generated by the assembler.

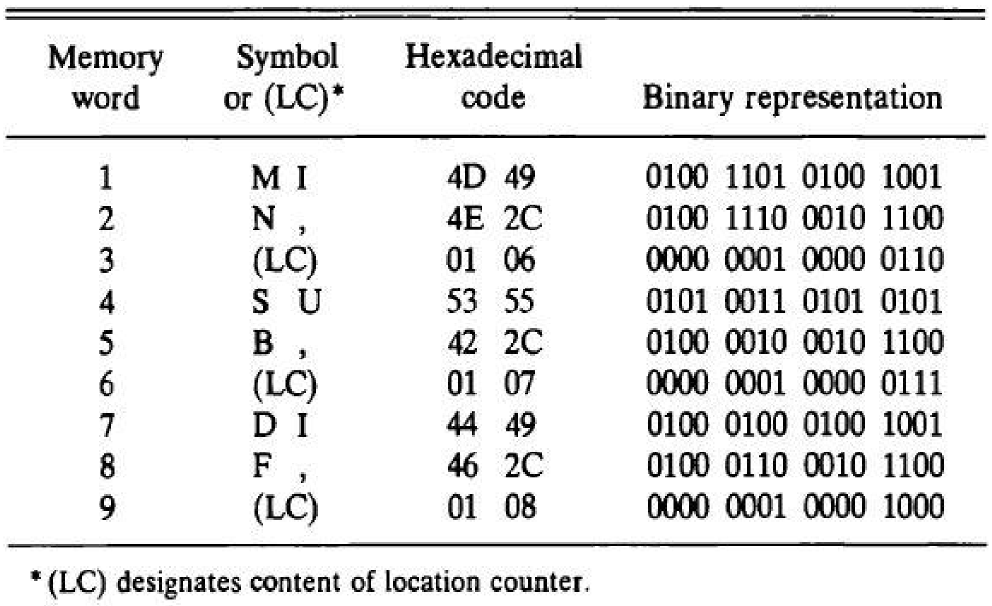
First Pass

A two-pass assembler scans the entire symbolic program twice. During the first pass, it generates a table that correlates all user-defined address symbols with their binary equivalent value. The binary translation is done during the second pass. To keep track of the location of instructions, the assembler uses a memory word called a LC. The content of LC stores the value of the memory location assigned to the instruction or operand presently being processed. The ORG pseudo-instruction initializes the LC to the value of the first location. Since instructions are stored in sequential locations, the content of LC is incremented by 1 after processing each line of code. To avoid ambiguity in case ORG is missing, the assembler sets the location counter to O initially.



LC is initially set to 0. A line of symbolic code is analyzed to determine if it has a label (by the presence of a comma). If the line of code has no label, the assembler checks the symbol in the instruction field. If it contains an ORG pseudo-instruction, the assembler sets LC to the number that follows ORG and goes back to process the next line. If the line has an END pseudo-instruction, the assembler terminates the first pass and goes to the second pass. Note that a line with ORG or END should not have a label. If the line of code contains a label, it is stored in the address symbol table together with its binary equivalent number specified by the content of LC. Nothing is stored in the table if no label is encountered. LC is then incremented by 1 and a new line of code is processed.

Address Symbol Table



Memory word	Symbol or (LC)*	Hexadecimal code	Binary representation
1	M I	4D 49	0100 1101 0100 1001
2	N ,	4E 2C	0100 1110 0010 1100
3	(LC)	01 06	0000 0001 0000 0110
4	S U	53 55	0101 0011 0101 0101
5	B ,	42 2C	0100 0010 0010 1100
6	(LC)	01 07	0000 0001 0000 0111
7	D I	44 49	0100 0100 0100 1001
8	F ,	46 2C	0100 0110 0010 1100
9	(LC)	01 08	0000 0001 0000 1000

* (LC) designates content of location counter.

Each label symbol is stored in two memory locations and is terminated by a comma. If the label contains less than three characters, the memory locations are filled with the code for space. The value found in LC while the line was processed is stored in the next sequential memory location. The program has three symbolic addresses: MIN, SUB, and DIF. These symbols represent 12-bit addresses equivalent to hexadecimal 106, 107, and 108, respectively. The address symbol table occupies three words for each label symbol encountered and constitutes the output data that the assembler generates during the first pass.

Second Pass

Machine instructions are translated during the second pass by means of table lookup procedures. A table-lookup procedure is a search of table entries to determine whether a specific item matches one of the items stored in the table. The assembler uses four tables. Any symbol that is encountered in the program must be available as an entry in one of these tables; otherwise, the symbol cannot be interpreted.

7.16 Tables in second pass

Four tables are:

1) Pseudo-instruction table.

2) MRI table.

3) Non-MRI table.

4) Address symbol table.

1) Pseudo-instruction Table:

   The entries of the pseudo-instruction table are the four symbols ORG,END, DEC, and HEX. Each entry refers the assembler to a subroutine that processes the pseudo-instruction when encountered in the program.

2) MRI Table:

   The MRI table contains the seven symbols of the memory-reference instructions and their 3-bit operation code equivalent.

3) Non-MRI Table:

   The non-MRI table contains the symbols for the 18 register-reference and input-output instructions and their 16-bitbinary code equivalent.

4) Address Symbol Table:

   The address symbol table is generated during the first pass of the assembly process.

The assembler searches these tables to find the symbol that it is currently processing in order to determine its binary value.

LC is initially set to 0. Lines of code are then analyzed one at a time. Labels are neglected during the second pass, so the assembler goes immediately to the instruction field and proceeds to check the first symbol encountered. It first checks the pseudo-instruction table. A match with ORG sends the assembler to a subroutine that sets LC to an initial value. A match with END terminates the translation process. An operand pseudo-instruction causes a conversion of the operand into binary. This operand is placed in the memory location specified by the content of LC. The LC is then incremented by 1 and the assembler continues to analyze the next line of code.

If the symbol encountered is not a pseudo-instruction, the assembler refers to the MRI table. If the symbol is not found in this table, the assembler refers to the non-MRI table. A symbol found in the non-MRI table corresponds to a register reference or input-0utput instruction. The assembler stores the 16-bit instruction code into the memory word specified by LC. The LC is incremented and a new line analyzed.

When a symbol is found in the MRI table, the assembler extracts its equivalent 3-bit code and inserts it in bits 2 through 4 of a word. A memory reference instruction is specified by two or three symbols. The second symbol is a symbolic address and the third, which may or may not be present, is the letter I. The symbolic address is converted to binary by searching the address symbol table.

The first bit of the instruction is set to O or 1, depending on whether the letter I is absent or present. The three parts of the binary instruction code are assembled and then stored in the memory location specified by the content of LC. The LC is incremented and the assembler continues to process the next line.

7.17 Error Diagnostics

One important task of an assembler is to check for possible errors in the symbolic program. This is called error diagnostics. One such error may be an invalid machine code symbol which is detected by its being absent in the MRI and non-MRI tables. The assembler cannot translate such a symbol because it does not know its binary equivalent value. In such a case, the assembler prints an error message to inform the programmer that his symbolic program has an error at a specific line of code. Another possible error may occur if the program has a symbolic address that did not appear also as a label. The assembler cannot translate the line of code properly because the binary equivalent of the symbol will not be found in the address symbol table generated during the first pass. Other errors may occur and a practical assembler should detect all such errors and print an error message for each.

Program Loops

A program loop is a sequence of instructions that are executed many times, each time with a different set of data. Program loops are specified in Fortran by a DO statement. The following is an example of a FORTRAN program that forms the sum of 100 integer numbers.

```
          DIMENSION A(100)
          INTEGER SUM, A
          SUM= 0
          DO  Ǝ   J= 1,    100
Ǝ         SUM=SUM+A(J)
```

Statement number 3 is executed 100 times, each time with a different operand AG) for **J** = 1, 2, ... , 100. A system program that translates a program written in a high-level programming language to **a** machine language program is called a compiler. A compiler is a more complicated program than an assembler and requires knowledge of systems programming to fully understand its operation. Nevertheless, we can demonstrate the basic functions of a compiler by going through the process of translating the program above to an assembly language program. A compiler may use an assembly language as an intermediate step in the translation or may translate the program directly to binary.

Line			
1		ORG 100	/Origin of program is HEX 100
2		LDA ADS	/Load first address of operands
3		STA PTR	/Store in pointer
4		LDA NBR	/Load minus 100
5		STA CTR	/Store in counter
6		CLA	/Clear accumulator
7	LOP,	ADD PTR I	/Add an operand to AC
8		ISZ PTR	/Increment pointer
9		ISZ CTR	/Increment counter
10		BUN LOP	/Repeat loop again
11		STA SUM	/Store sum
12		HLT	/Halt
13	ADS,	HEX 150	/First address of operands
14	PTR,	HEX 0	/This location reserved for a pointer
15	NBR,	DEC −100	/Constant to initialized counter
16	CTR,	HEX 0	/This location reserved for a counter
17	SUM,	HEX 0	/Sum is stored here
18		ORG 150	/Origin of operands is HEX 150
19		DEC 75	/First operand
•			
•			
•			
118		DEC 23	/Last operand
119		END	/End of symbolic program

The first statement in the Fortran program is a DIMENSION statement. This statement instructs the compiler to reserve 100 words of memory for 100 operands. The value of the operands is determined from an input statement. The second statement informs the compiler that the numbers are integers. If they were of the real type, the compiler would have to reserve locations for floating-point numbers and generate instructions that perform the subsequent arithmetic with floating-point data. These two statements are non executable and are similar to the pseudo-instructions in an assembly language. Suppose that the compiler reserves locations $(150)_{16}$ to $(1B3)_{16}$ for the 100 operands. These reserved memory words are listed in lines 19 to 118 in the translated program. This is done by the ORG pseudo-instruction in line 18, which specifies the origin of the operands. The first and last operands are listed with a specific decimal number, although these values are not known during compilation. The compiler just reserves the data space in memory and the values are inserted later when an input data statement is executed. The line numbers in the symbolic program are for reference only and are not part of the translated symbolic program. The indexing of the DO statement is translated into the instructions in lines 2 through 5 and the constants in lines 13 through 16. The address of the first operand (150) is stored in location ADS in line 13. The number of times that Fortran statement number 3 must be executed is 100. So -100 is stored in location NBR. The compiler then generates the instructions in lines 2 through 5 to initialize the program loop.

The address of the first operand is transferred to location PTR. This corresponds to setting AO) to A(l). The number -100 is then transferred to location CTR. This location acts as a counter with its content incremented by one every time the program loop is executed. When the value of the counter reaches zero, the 100 operations will be completed and the program will exit from the loop. Some compilers will translate the statement SUM = 0 into a machine instruction that initializes location SUM to zero. A reference to this location is then made every time Fortran statement number 3 is executed. A more intelligent compiler will realize that the sum can be formed in the accumulator and only the final result stored in location SUM. This compiler will produce an instruction in line 6 to clear the *AC*. It will also reserve a memory location symbolized by SUM (in line 17) for storing the value of this variable at the termination of the loop. The program loop specified by the DO statement is translated to the sequence of instructions listed in lines 7 through 10. Line 7 specifies an indirect ADD instruction because it has the symbol I. The address of the current operand is stored in location PTR.

When this location is addressed indirectly the computer takes the content of PTR to be the address of the operand. As a result, the operand in location 150 is added to the accumulator. Location PTR is then incremented with the ISZ instruction in line 8, so its value changes to the value of the address of the next sequential operand. Location CTR is incremented in line 9, and if it is not zero, the computer does not skip the next instruction. The next instruction is a branch (BUN) instruction to the beginning of the loop, so the computer returns to repeat the loop once again. When location CTR reaches zero (after the loop is executed 100 times), the next instruction is skipped and the computer executes the instructions in lines 11 and 12. The sum formed in the accumulator is stored in SUM and the computer halts. The halt instruction is inserted here for clarity; actually, the program will branch to a location where it will continue to execute the rest of the program or branch to the beginning of another program. Note that ISZ in line 8 is used merely to add 1 to the address pointer PTR. Since the address is **a** positive number, a skip will never occur.

7.18 Pointer Counter

The program introduces the idea of a pointer and a counter which can be used, together with the indirect address operation, to form a program loop. The pointer points to the address of the current operand and the counter counts the number of times that the program loop is executed. In this example we use two memory locations for these functions. In computers with more than one processor register, it is possible to use one processor register as a pointer, another as a counter, and a third as an accumulator. When processor registers are used as pointers and counters they are called index registers.

Summary:

- Machine instructions inside the computer form a binary pattern which is difficult for people to work with and understand.
- A program written by a user may be either dependent or independent of the physical computer that runs this program.
- A sequence of instructions and operands in binary form is binary code.
- An equivalent translation of the binary code to octal or hexadecimal representation is octal code.
- The user employs symbols (letters, numerals or special characters) for the operation part, the address part and other parts of the instruction code is symbolic code.
- High level programming languages employs problem oriented symbols or formats.
- There are 25 instructions of the basic computer.

Keywords:

Programming language: It is defined by the set of rules.

Label: This field may be empty or it may specify a symbolic address.

Instruction: This field specifies a machine instruction or pseudo-instruction.

Comment: This field may be empty or it may include a comment.

Symbolic address: It consists of one, two, or three, but not more than three alphanumeric characters.

Memory reference instruction: It occupies two or three symbols separated by spaces.

Self Assessment

1. In a basic computer, each instruction has a ______ letter symbol in programs.

A. 2

B. 3

C. 4

D. 5

2. In BUN computer instruction, the hexadecimal code is

A. 2 or A

B. 3 or B

C. 4 or C

D. 5 or D

3. In BSA computer instruction, the hexadecimal code is

A. 2 or A

B. 3 or B

C. 4 or C

D. 5 or D

4. In which category, the problem oriented symbols or formats are used?

A. Binary code

B. Octal/hexadecimal code

C. Symbolic code

D. High level programming languages

5. The fields, i.e., label, instruction and comment specify

A. Assembly language

B. High level programming language

C. Octal programming language

D. None of the above

6. A symbolic address must consist of __________ alphanumeric characters.

A. 1

B. 2

C. 3

D. Either 1, 2 or 3

7. The first character in the label field must be a

A. Letter
B. Numeral
C. Symbol
D. None of the above

8. A symbolic address in the label field is terminated by a _________ so that it will be recognized as a label by the assembler.
A. Comma
B. Semi-colon
C. Dot
D. Slash

9. A comment must be preceded by a ________ for the assembler to recognize the beginning of the comment field.
A. Comma
B. Semi-colon
C. Dot
D. Slash

10. The instruction CLA specifies
A. Non-MRI instruction
B. Direct address instruction
C. Indirect address instruction
D. None of the above

11. The pseudo-instruction *ORG N* gives the memory location for the instruction; here N is a ________ number.
A. Decimal
B. Binary
C. Octal
D. Hexadecimal

12. The translation into binary from the symbolic program is done by a special program called
A. Linker
B. Assembler
C. Loader
D. None of the above

13. The assignment of a memory location to each machine instruction and operand is done in _______ scan.
A. First
B. Second

14. The address symbol table creation is done in _______ scan.
A. First

B. Second

15. The table look up procedures is implemented in _________ pass.

A. First

B. Second

Answer for Self Assessment

1.	B	2.	C	3.	D	4.	D	5.	A
6.	D	7.	A	8.	A	9.	D	10.	B
11.	D	12.	B	13.	A	14.	A	15.	B

Review Questions:

1. What is machine language? Explain its categories.
2. Write a binary and hexadecimal program to add two numbers.
3. Write a program with symbolic operation codes and Fortran program for addition of two numbers.
4. Explain what the rules of assembly language are.
5. Explain two passes of assembler.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education Asia, 2002.

Web Links

https://www.britannica.com/technology/machine-language

Unit 08: Machine Programming

CONTENTS

Objectives

After studying this unit, you will be able to understand

- understand the arithmetic operationprogramming
- understand the logic operation programming
- understand the subroutines
- understand input-output programming

Introduction

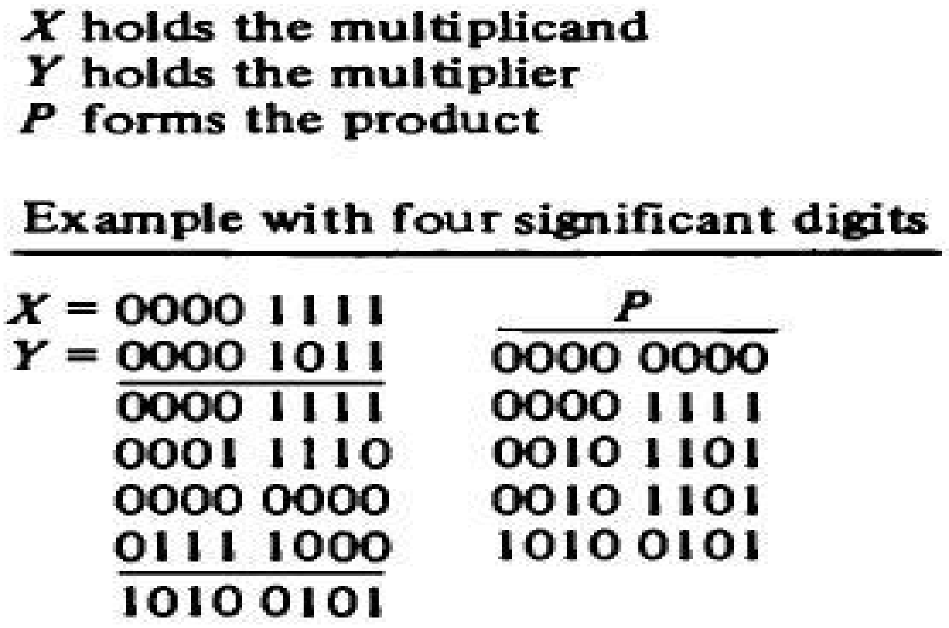
Consider the four basic arithmetic operations. Some computers have machine instructions to add, subtract, multiply, and divide. Others, such as the basic computer, have only one arithmetic instruction, such as ADD. Operations not included in the set of machine instructions must be implemented by a program.Operations with one machine instruction represents and is related to the hardware component of a computer.Operations with a set of instructions represents and is related to the software component of a computer.

Some computers have a variety of hardware instructions. Others contain a smaller set of hardware instructions and depend upon the software implementation of many operations.Both of these have their own pros and cons.Depending upon the situation, either of these configurations can be chosen.

For doing a multiplication program, we need a multiplicand and a multiplier. The result of the multiplication operation is called a product. The assumptions for this are:

A) Neglect the sign bit and assume positive numbers.

B)The two binary numbers have no more than eight significant bits so their product cannot exceed the word capacity of 16 bits.

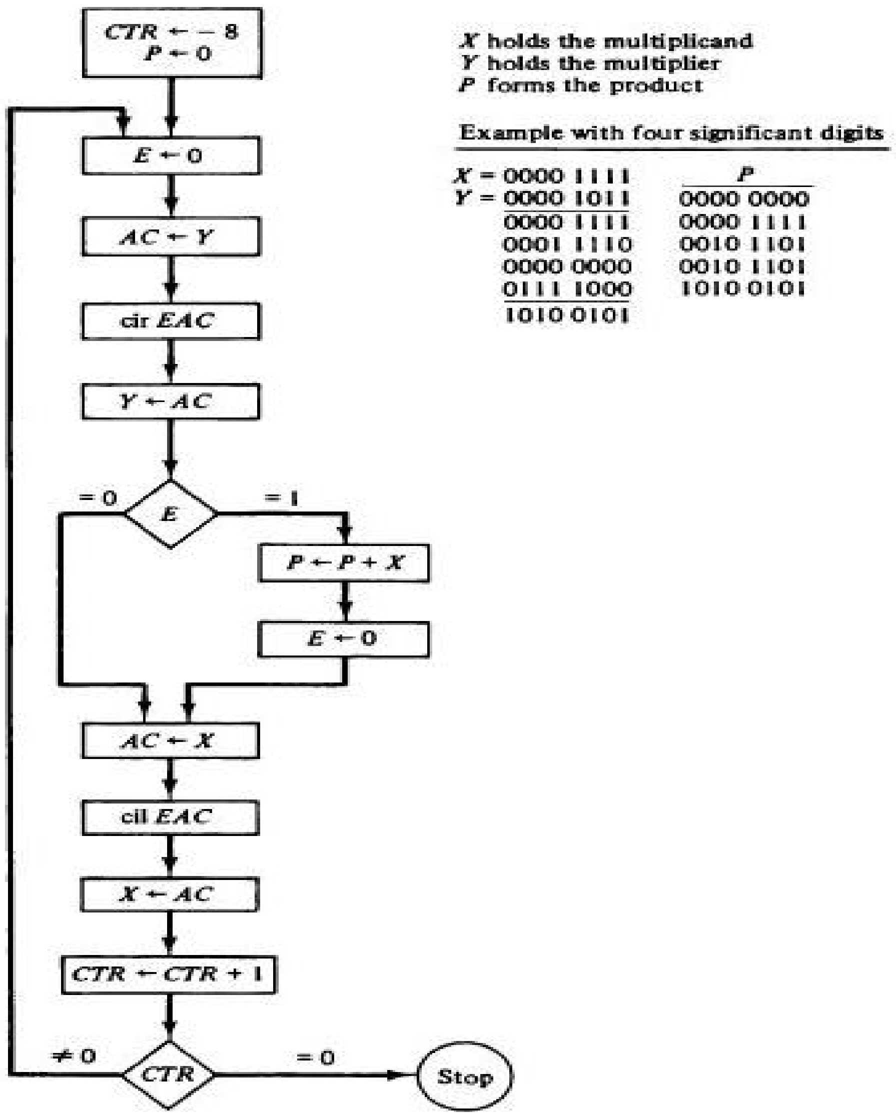


X holds the multiplicand
Y holds the multiplier
P forms the product

Example with four significant digits

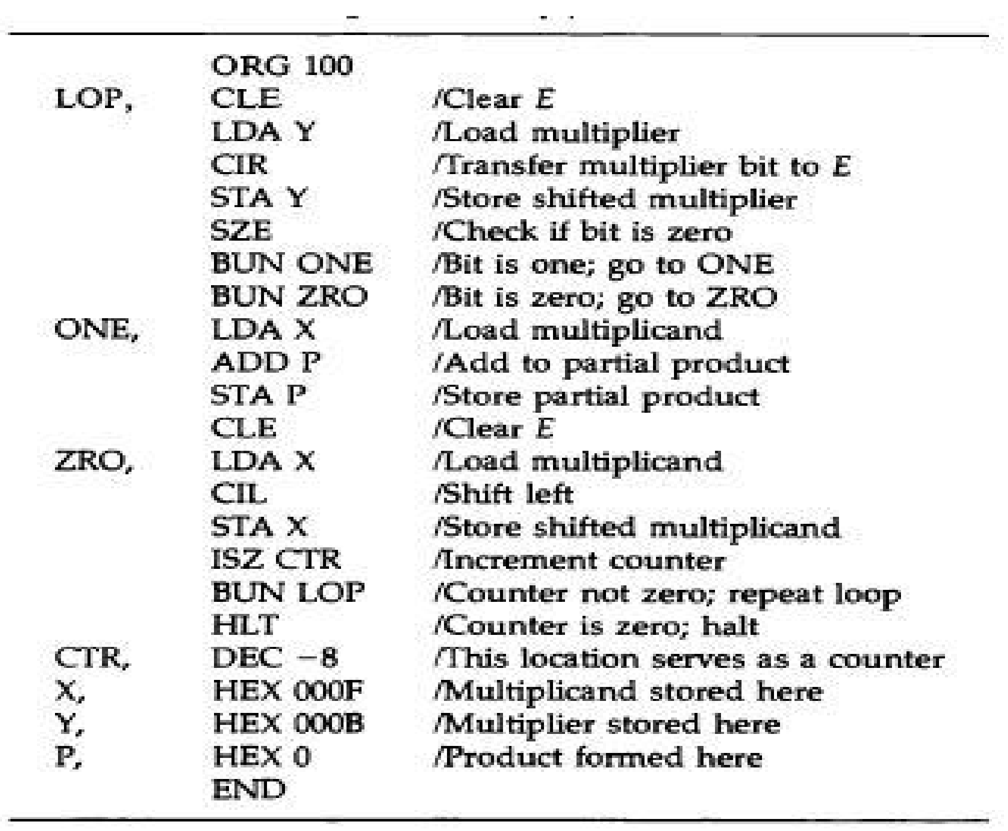
```
X = 0000 1111          P
Y = 0000 1011      0000 0000
    0000 1111      0000 1111
    0001 1110      0010 1101
    0000 0000      0010 1101
    0111 1000      1010 0101
    1010 0101
```

The program has a loop that is traversed eight times, once for each significant bit of the multiplier. Initially, location X holds the multiplicand and location Y holds the multiplier.



The initialization is not listed here but should be included when the program is loaded into the computer. Initialization of the Program:

- Multiplicand into location X
- Multiplier into locations Y,
- Counter to -8;
- Initializing location P to O.



```
         ORG 100
LOP,     CLE          /Clear E
         LDA Y        /Load multiplier
         CIR          /Transfer multiplier bit to E
         STA Y        /Store shifted multiplier
         SZE          /Check if bit is zero
         BUN ONE      /Bit is one; go to ONE
         BUN ZRO      /Bit is zero; go to ZRO
ONE,     LDA X        /Load multiplicand
         ADD P        /Add to partial product
         STA P        /Store partial product
         CLE          /Clear E
ZRO,     LDA X        /Load multiplicand
         CIL          /Shift left
         STA X        /Store shifted multiplicand
         ISZ CTR      /Increment counter
         BUN LOP      /Counter not zero; repeat loop
         HLT          /Counter is zero; halt
CTR,     DEC -8       /This location serves as a counter
X,       HEX 000F     /Multiplicand stored here
Y,       HEX 000B     /Multiplier stored here
P,       HEX 0        /Product formed here
         END
```

8.1 Logic Operations

The basic computer has three machine instructions that perform logic operations: AND, CMA, and CLA.The LOA instruction may be considered as a logic operation that transfers a logic operand into the AC. Any logic function can be implemented using the AND and complement operations. For example, the OR operation is not available as a machine instruction in the basic computer. From De-Morgan's theorem we recognize the relationx + y = (x'y')'. The second expression contains only AND and complement operations. A program that forms the OR operation of two logic operands A and B is as follows:

- LDAA — Load first operand A
- CMA — Complement to get A
- STA TMP — Store in a temporary location
- LDAB — Load second operand B
- CMA — Complement to get B
- AND TMP — AND with A′ to get A′ Λ B′
- CMA — Complement again to get A v B

8.2 Shift Operations

These two shifts can be programmed with a small number of instructions.The logical shift requires that zeros be added to the extreme positions. This is easily accomplished by clearing *E* and circulating the *AC* and E. For a logical shift-right operation we need the two instructions: CLE and CIR.For a logical shift-left operation we need the two instructions: CLE and CIL.The arithmetic shifts depend on the type of representation of negative numbers.For the basic computer we have adopted the signed-2's complement representation. For an arithmetic right-shift it is necessary that the sign bit in the left most position remain unchanged. But the sign bit itself is shifted into the high-order bit position of the number. The program for the arithmetic right-shift requires that we set E to the same value as the sign bit and circulate right, thus:

- CLE /Clear E t o □
- SPA /Skip if AC is positive; E remains □
- CME /AC is negative; set E to 1
- CIR /Circulate E and AC

For arithmetic shift-left, it is necessary that the added bit in the least significant position be O. This is easily done by clearing E prior to the circulate-left operation. The sign bit must not change during this shift. With a circulate instruction, the sign bit moves into E.

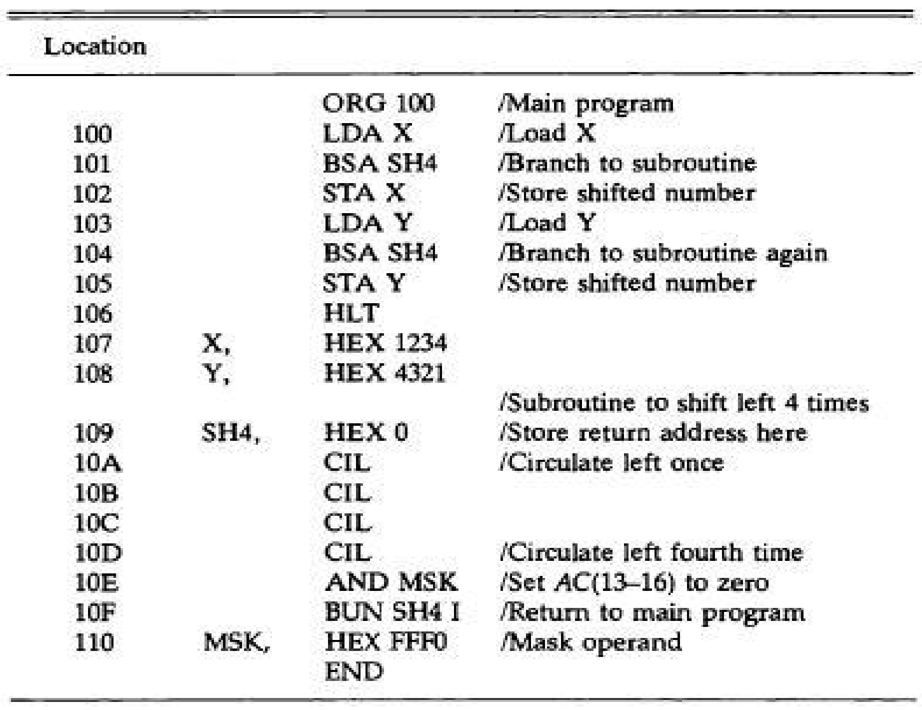
8.3 Subroutines

A set of common instructions that can be used in a program many times is called a subroutine. Branching is used for the concept of subroutine. A subroutine consists of a self-contained sequence of instructions that carries out a given task. A branch can be made to the subroutine from any part of the main program. It is necessary to store the return address somewhere in the computer for the subroutine to know where to return.

8.4 BSA Instruction

In the basic computer, the link between the main program and a subroutine is the BSA instruction. BSA is branch and save return address.

A subroutine that shifts the content of the accumulator four times to the left. Shifting a word four times is a useful operation for processing BCD numbers or alphanumeric characters.



```
Location

                 ORG 100       /Main program
100              LDA X         /Load X
101              BSA SH4       /Branch to subroutine
102              STA X         /Store shifted number
103              LDA Y         /Load Y
104              BSA SH4       /Branch to subroutine again
105              STA Y         /Store shifted number
106              HLT
107     X,       HEX 1234
108     Y,       HEX 4321
                               /Subroutine to shift left 4 times
109     SH4,     HEX 0         /Store return address here
10A              CIL           /Circulate left once
10B              CIL
10C              CIL
10D              CIL           /Circulate left fourth time
10E              AND MSK       /Set AC(13–16) to zero
10F              BUN SH4 I     /Return to main program
110     MSK,     HEX FFF0      /Mask operand
                 END
```

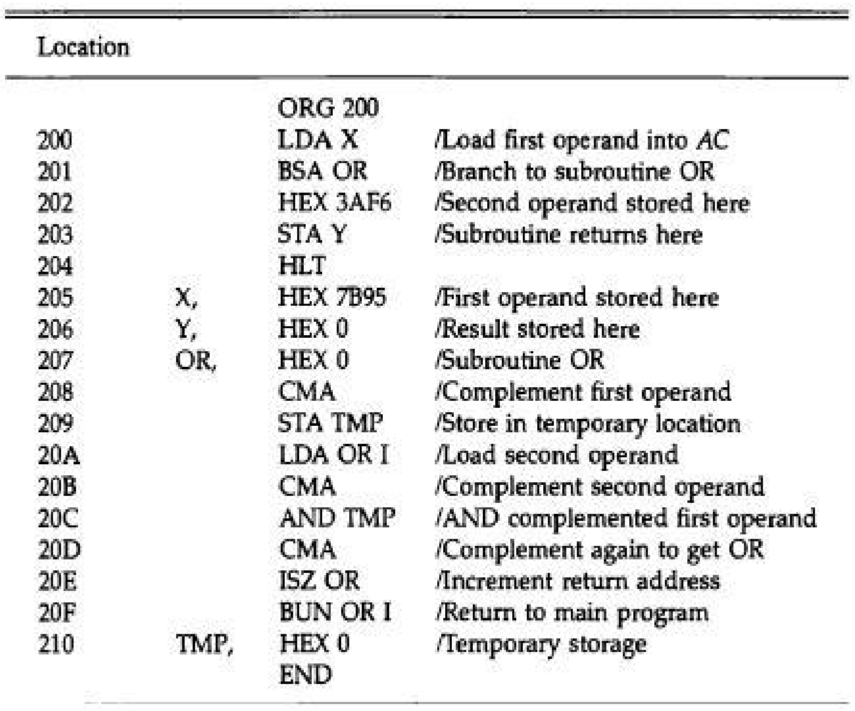
8.5 Subroutine Linkage

The first memory location of each subroutine serves as a link between the main program and the subroutine. The procedure for branching to a subroutine and returning to the main program is referred to as a subroutine linkage.The BSA instruction performs an operation commonly called subroutine call. The last instruction of the subroutine performs an operation commonly called subroutine return.The procedure used in the basic computer for subroutine linkage is commonly found in computers with only one processor register. Many computers have multiple processor registers and some of them are assigned the name index registers.In such computers, an index register is usually employed to implement the subroutine linkage. A branch-to-subroutine instruction stores the return address in an index register. A return-from-subroutine instruction is effected by branching to the address presently stored in the index register.

8.6 Subroutine Parameters and Data Linkage

When a subroutine is called, the main program must transfer the data it wishes the subroutine to work with. In the previous example, the data were transferred through the accumulator. The operand was loaded into the AC prior to the branch. The subroutine shifted the number and left it there to be accepted by the main program. The accumulator can be used for a single input parameter and a single output parameter. In computers with multiple processor registers, more parameters can be transferred this way. Another way to transfer data to a subroutine is through the memory. Data are often placed in memory locations following the call. They can also be placed in a block of storage.The first address of the block is then placed in the memory location following the call.In any case, the return address always gives the link information for transferring data between the main program and the subroutine.

Consider a subroutine that performs the logic OR operation. Two operands must be transferred to the subroutine and the subroutine must return the result of the operation.



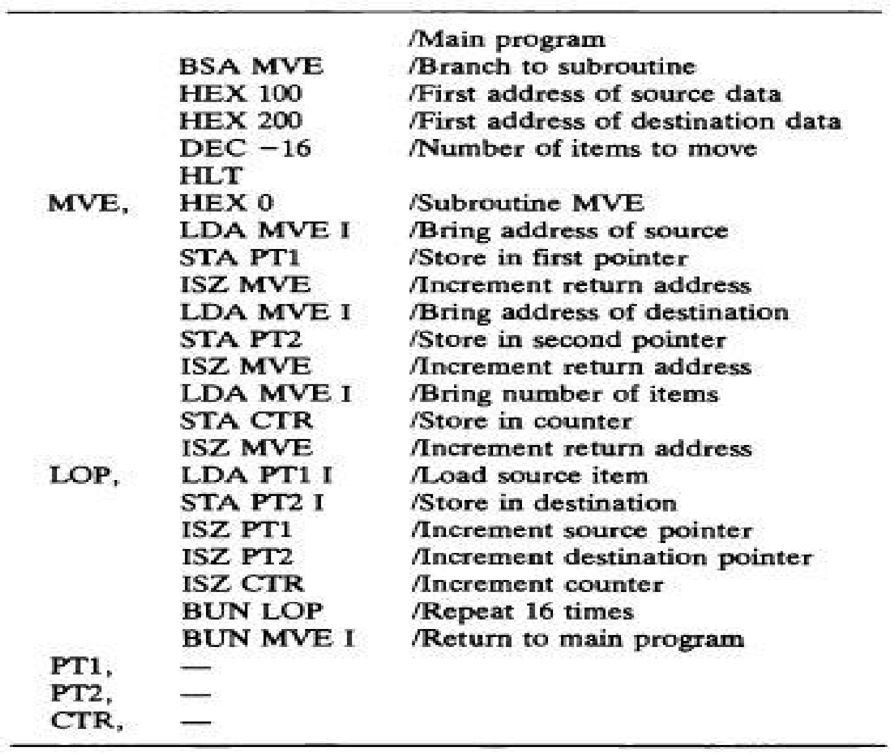
```
Location

                ORG 200
200             LDA X       /Load first operand into AC
201             BSA OR      /Branch to subroutine OR
202             HEX 3AF6    /Second operand stored here
203             STA Y       /Subroutine returns here
204             HLT
205     X,      HEX 7B95    /First operand stored here
206     Y,      HEX 0       /Result stored here
207     OR,     HEX 0       /Subroutine OR
208             CMA         /Complement first operand
209             STA TMP     /Store in temporary location
20A             LDA OR I    /Load second operand
20B             CMA         /Complement second operand
20C             AND TMP     /AND complemented first operand
20D             CMA         /Complement again to get OR
20E             ISZ OR      /Increment return address
20F             BUN OR I    /Return to main program
210     TMP,    HEX 0       /Temporary storage
                END
```

The accumulator can be used to transfer one operand and to receive the result. The other operand is inserted in the location following the BSA instruction. The first operand in location X is loaded into the AC. The second operand is stored in location 202 following the BSA instruction. After the branch, the first location in the subroutine holds the number 202. Note that in this case, 202 is not the return address but the address of the second operand. The subroutine starts performing the OR operation by complementing the first operand in the AC and storing it in a temporary location TMP. The second operand is loaded into the AC by an indirect instruction at location OR. Remember that location OR contains the number 202. When the instruction refers to it indirectly, the operand at location 202 is loaded into the AC.

This operand is complemented and then ANDed with the operand stored in TMP. Complementing the result forms the OR operation.The return from the subroutine must be manipulated so that the main program continues from location 203 where the next instruction is located.This is accomplished by incrementing location OR with the ISZ instruction. Now location OR holds the number 203 and an indirect BUN instruction causes a return to the proper place.It is possible to have more than one operand following the BSA instruction. The subroutine must increment the return address stored in its first location for each operand that it extracts from the calling program. Moreover, the calling program can reserve one or more locations for the subroutine to return results that are computed. The first location in the subroutine must be incremented for these locations as well, before the return. If there is a large amount of data to be transferred, the data can be placed in a block of storage and the address of the first item in the block is then used as the linking parameter.

8.7 Subroutine to Move a Block of Data



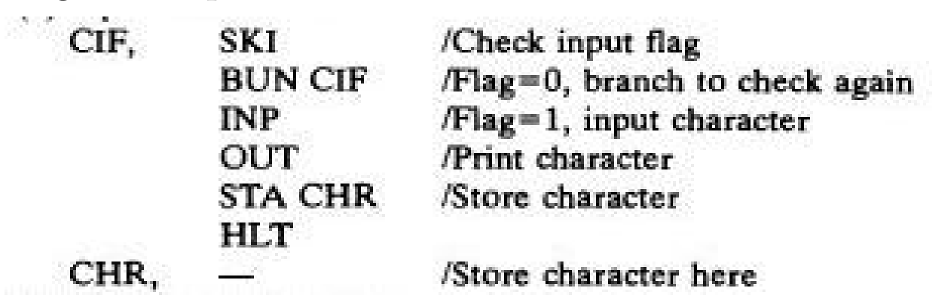
```
                        /Main program
        BSA MVE         /Branch to subroutine
        HEX 100         /First address of source data
        HEX 200         /First address of destination data
        DEC −16         /Number of items to move
        HLT
MVE,    HEX 0           /Subroutine MVE
        LDA MVE I       /Bring address of source
        STA PT1         /Store in first pointer
        ISZ MVE         /Increment return address
        LDA MVE I       /Bring address of destination
        STA PT2         /Store in second pointer
        ISZ MVE         /Increment return address
        LDA MVE I       /Bring number of items
        STA CTR         /Store in counter
        ISZ MVE         /Increment return address
LOP,    LDA PT1 I       /Load source item
        STA PT2 I       /Store in destination
        ISZ PT1         /Increment source pointer
        ISZ PT2         /Increment destination pointer
        ISZ CTR         /Increment counter
        BUN LOP         /Repeat 16 times
        BUN MVE I       /Return to main program
PT1,    —
PT2,    —
CTR,    —
```

A subroutine that moves a block of data starting at address 100 into a block starting with address 200 is listed in table.The length of the block is 16 words. The first introduction is a branch to subroutine MVE. The first part of the subroutine transfers the three parameters 100, 200 and -16 from the main program and places them in its own storage location. The items are retrieved from their blocks by the use of two pointers. The counter ensures that only 16 items are moved. When the subroutine completes its operation, the data required is in the block starting from the location 200. The return to the main program is to the HLT instruction.

8.8 Input-Output Programming

Symbols is astrings of characters and the character is a 8-bit code. INP is a binary-coded character enters the computer. OUT is a binary-coded character is transferred to the output device.

8.9 Program to Input one Character



```
CIF,    SKI          /Check input flag
        BUN CIF      /Flag=0, branch to check again
        INP          /Flag=1, input character
        OUT          /Print character
        STA CHR      /Store character
        HLT
CHR,    —            /Store character here
```

- SKI: Checks the input flag for the availability of character.
- The next instruction is skipped if the input flag bit is 1.
- INP: Transfers the binary-coded character into AC (0-7).
- OUT: Prints the character.

OUT Instruction

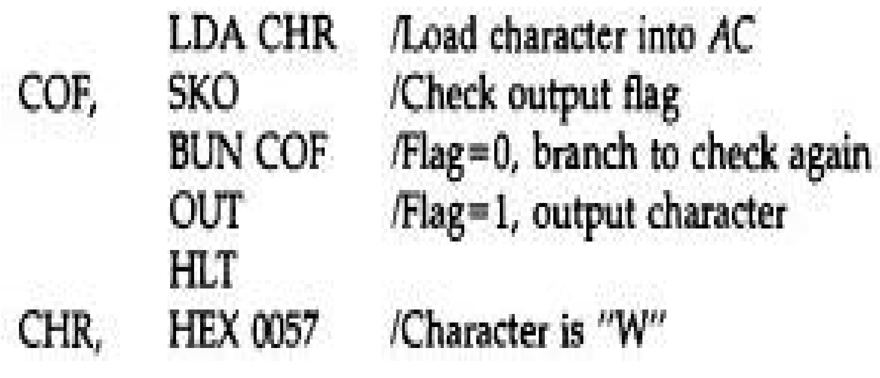
A terminal unit that communicates directly with a computer does not print the character when a key is depressed. To type it, it is necessary to send an OUT instruction for the printer. In this way, the user is ensured that the correct transfer has occurred.

Flag=0

What happens when the flag is 0:

- Then, the next instruction in sequence is executed.
- This instruction is a branch to return and check the flag bit again.

8.10 Program to Output one Character



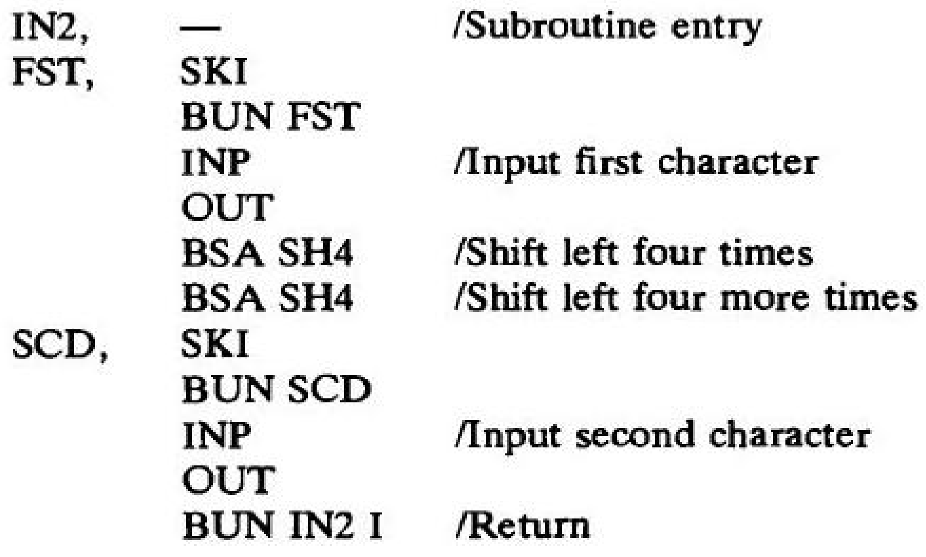
```
        LDA CHR      /Load character into AC
COF,    SKO          /Check output flag
        BUN COF      /Flag=0, branch to check again
        OUT          /Flag=1, output character
        HLT
CHR,    HEX 0057     /Character is "W"
```

The character is first loaded into the *AC*. The output flag is then checked. If it is 0, the computer remains in a two-instruction loop checking the flag bit. When the flag changes to 1, the character is transferred from the accumulator to the printer.

8.11 Character Manipulation

A computer can also work as a symbol manipulator. One character manipulation task is to pack two characters in one word. This is convenient because each character occupies 8 bits and a memory word contains 16 bits.

8.12 Subroutine to Input and Pack two Characters

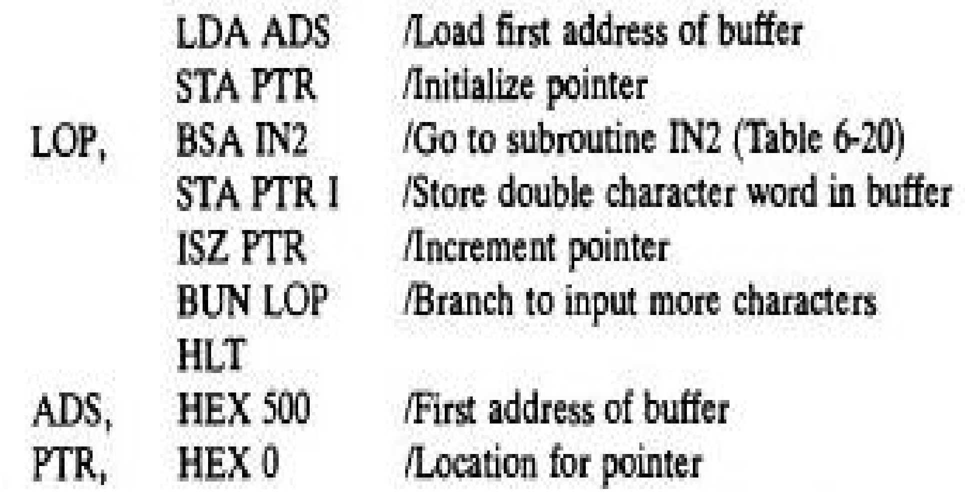


```
IN2,    —               /Subroutine entry
FST,    SKI
        BUN FST
        INP             /Input first character
        OUT
        BSA SH4         /Shift left four times
        BSA SH4         /Shift left four more times
SCD,    SKI
        BUN SCD
        INP             /Input second character
        OUT
        BUN IN2 I       /Return
```

The program lists a subroutine named IN2 that inputs two characters and packs them into one 16-bit word. The packed word remains in the accumulator.

8.13 A program to Store Input Characters in a Buffer

In assembler, the symbolic program is stored in a section of memory which is sometimes called a buffer. The symbolic program being typed enters through the input device and is stored in consecutive memory locations in the buffer.



```
        LDA ADS         /Load first address of buffer
        STA PTR         /Initialize pointer
LOP,    BSA IN2         /Go to subroutine IN2 (Table 6-20)
        STA PTR I       /Store double character word in buffer
        ISZ PTR         /Increment pointer
        BUN LOP         /Branch to input more characters
        HLT
ADS,    HEX 500         /First address of buffer
PTR,    HEX 0           /Location for pointer
```

The first address of the buffer is 500. The first double character is stored in location 500 and all characters are stored in sequential locations.No counter is used in the program, so characters will be read as long as they are available or until the buffer reaches location O (after location FFFF). This is an operation that searches a table to find out if it contains a given symbol. The search may be done by comparing the given symbol with each of the symbols stored in the table.

8.14 A Program to Compare two Words

```
          LDA WD1   /Load first word
          CMA
          INC       /Form 2's complement
          ADD WD2   /Add second word
          SZA       /Skip if AC is zero
          BUN UEQ   /Branch to "unequal" routine
          BUN EQL   /Branch to "equal" routine
WD1,      —
WD2,      —
```

The comparison is accomplished by forming the 2's complement of a word (as if it were a number) and arithmetically adding it to the second word. If the result is zero, the two words are equal and a match occurs. If the result is not zero, the words are not the same.

8.15 Program Interrupt

The running time of input and output programs is made up primarily of the time spent by the computer in waiting for the external device to set its flag.The waiting loop that checks the flag keeps the computer occupied with a task that wastes a large amount of time. This waiting time can be eliminated if the interrupt facility is used to notify the computer when a flag is set. The advantage of using the interrupt is that the information transfer is initiated upon request from the external device. It is useful in a multi-program environment.The program currently being executed is referred to as the running program. The other programs are usually waiting for input or output data. The function of the interrupt facility is to take care of the data transfer of one (or more) program while another program is currently being executed. The running program must include an ION instruction to turn the interrupt on. If the interrupt facility is not used, the program must include an IOF instruction to turn it off.

The interrupt facility allows the running program to proceed until the input or output device sets its ready flag.Whenever a flag is set to 1, the computer completes the execution of the instruction in progress and then acknowledges the interrupt. The result of this action is that the return address is stored in location 0. The instruction in location 1 is then performed; this initiates a service routine for the input or output transfer.The service routine can be stored anywhere in memory provided a branch to the start of the routine is stored in location 1. The service routine must have instructions to perform the following tasks:

- Save contents of processor registers.
- Check which flag is set.
- Service the device whose flag is set.
- Restore contents of processor registers.
- Turn the interrupt facility on.
- Return to the running program.

The contents of processor registers before the interrupt and after the return to the running program must be the same; otherwise, the running program may be in error. Since the service routine may use these registers, it is necessary to save their contents at the beginning of the routine and restore them at the end. The sequence by which the flags are checked dictates the priority assigned to each device. The device with higher priority is serviced first followed by the one with lower priority. The sequence by which the flags are checked dictates the priority assigned to each device. The device with higher priority is serviced first followed by the one with lower priority.

8.16 Program to Service an Interrupt

```
Location

  0      ZRO,    —            /Return address stored here
  1              BUN SRV      /Branch to service routine
100              CLA          /Portion of running program
101              ION          /Turn on interrupt facility
102              LDA X
103              ADD Y        /Interrupt occurs here
104              STA Z        /Program returns here after interrupt
  .               .
  .               .
  .                           /Interrupt service routine
200      SRV,    STA SAC      /Store content of AC
                 CIR          /Move E into AC(1)
                 STA SE       /Store content of E
                 SKI          /Check input flag
                 BUN NXT      /Flag is off, check next flag
                 INP          /Flag is on, input character
                 OUT          /Print character
                 STA PT1 I    /Store it in input buffer
                 ISZ PT1      /Increment input pointer
         NXT,    SKO          /Check output flag
                 BUN EXT      /Flag is off, exit
                 LDA PT2 I    /Load character from output buffer
                 OUT          /Output character
                 ISZ PT2      /Increment output pointer
         EXT,    LDA SE       /Restore value of AC(1)
                 CIL          /Shift it to E
                 LDA SAC      /Restore content of AC
                 ION          /Turn interrupt on
                 BUN ZRO I    /Return to running program
         SAC,    —            /AC is stored here
         SE,     —            /E is stored here
         PT1,    —            /Pointer of input buffer
         PT2,    —            /Pointer of output buffer
```

The contents of AC and E are stored in special locations. (These are the only processor registers in the basic computer.) The flags are checked sequentially, the input flag first and the output flag second.If any or both flags are set, an item of data is transferred to or from the corresponding memory buffer.Before returning to the running program the previous contents of E and AC are restored and the interrupt facility is turned on.The last instruction causes a branch to the address stored in location 0. This is the return address stored there previously during the interrupt cycle. Hence the running program will continue from location 104, where it was interrupted.A typical computer (may have many more input and output devices connected to the interrupt facility.Furthermore, interrupt sources are not limited to input and output transfers. Interrupts can be used for other purposes, such as internal processing errors or special alarm conditions.

Summary

- The basic computer, have only one arithmetic instruction, such as ADD.
- Operations not included in the set of machine instructions must be implemented by a program.
- Any logic function can be implemented using the AND and complement operations.

- The logical shift requires that zeros be added to the extreme positions.
- In the basic computer, the link between the main program and a subroutine is the BSA instruction.
- The accumulator can be used to transfer one operand and to receive the result.

Keywords

- Logical shift-right:For this operation, we need the two instructions: CLE and CIR.
- Logical shift-left: For this operation, we need the two instructions: CLE and CIL.
- Subroutine: A set of common instructions that can be used in a program many times.
- Subroutine Linkage: The procedure for branching to a subroutine and returning to the main program.
- Buffer: In assembler, the symbolic program is stored in a section of memory.

SelfAssessment

1. Which of the operation is not available as a machine instruction in basic computer?

A. OR
B. AND
C. NOT
D. None of the above

2. For arithmetic shift-left, it is necessary that the added bit in the least significant position should be _______.

A. -1
B. 0
C. 1
D. 2

3. Which of these instructions is considered as a logic operation that transfers a logic operand into the AC?

A. LOA
B. CMA
C. AND
D. CLA

4. Any logic function can be implemented using ______ and ___________ operations.

A. OR, XOR
B. NOR, XNOR
C. AND, complement
D. OR, complement

5. CMA instruction is used for

A. Complementation

B. Accumulation
C. Correlation
D. None of the above

6. In the basic computer, the link between the main program and a subroutine is the ____________ instruction.
A. LDA
B. BSA
C. BAS
D. None of the above

7. Which memory location of each subroutine serves as a link between the main program and the subroutine?
A. First
B. Second
C. Fifth
D. Last

8. Which instruction of the subroutine performs an operation commonly called subroutine return?
A. First
B. Second
C. Fifth
D. Last

9. When computers have multiple processor registers then which register is usually employed to implement the subroutine linkage?
A. Index register
B. Input register
C. Output Register
D. None of the above

10. The return to the main program after execution of subroutine is done by using ___________.
A. HLT
B. Exit
C. Back
D. None of the above

11. Which of the instruction checks the input flag for the availability of character?
A. SKI
B. INP
C. OUT
D. None of the above
12. A binary-coded character is transferred to the output device using _____ instruction?

A. SKI
B. INP
C. OUT
D. None of the above

13. To turn the interrupt on, which instruction is used?
A. INP
B. ION
C. IOF
D. None of the above

14. Interrupt sources can be used for
A. Input transfers
B. Output transfers
C. Internal processing errors
D. All input and output transfers and internal processing errors

15. A binary-coded character enters the computer using __________ instruction
A. INP
B. ION
C. IOF
D. None of the above

Answer for Self Assessment

1. A	2. B	3. A	4. C	5. A
6. B	7. A	8. D	9. A	10. A
11. A	12. C	13. B	14. D	15. A

Review Questions

1. Explain the instructions for multiplying two numbers.
2. Explain the shift operations.
3. What is a subroutine and its linkage.
4. What are subroutine parameters and data linkage?
5. Write a subroutine to move a block of data.
6. Write a program to input and output one character.
7. Write a subroutine to input and pack two characters.
8. Explain program interrupt. Write a program to service an interrupt.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education Asia, 2002.

Web Links

https://vardhaman.org/wp-content/uploads/2018/03/CAO%20Unit-I%20part-3.pdf

http://www.nou.ac.in/econtent/BCA%20Part%20I/Paper%207/BCA%20Paper-VII%20Block-2%20Unit-6.pdf

Unit 09: Register Organization

CONTENTS

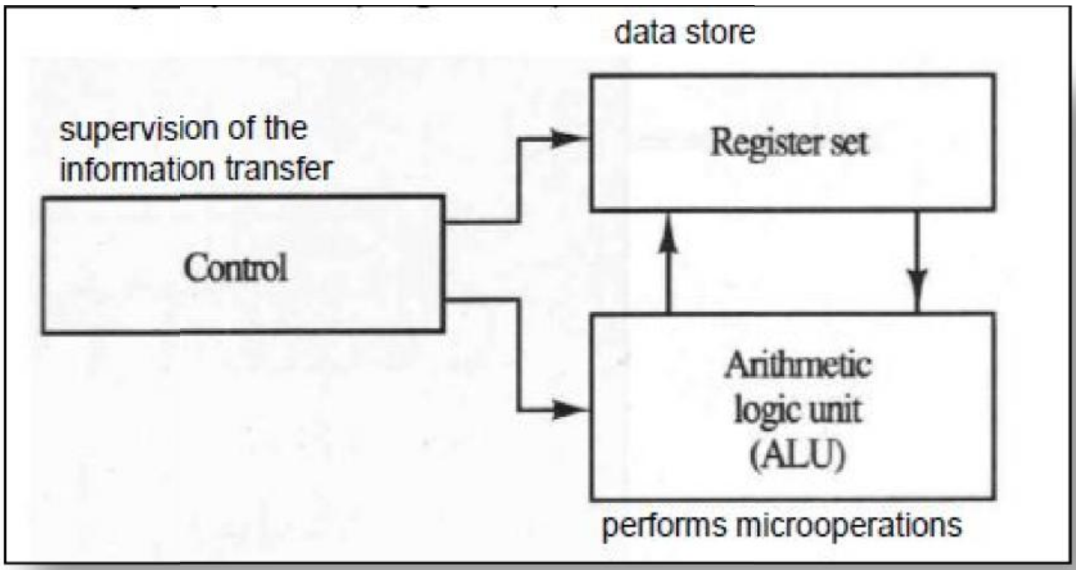
Objectives

After studying this unit, you will be

- Know the major components of CPU and their role.
- Understand the general register organization in CPU.
- Understand the basic data structure, i.e., stack
- Understand the basic operations on stack
- understand the infix, prefix and postfix notation

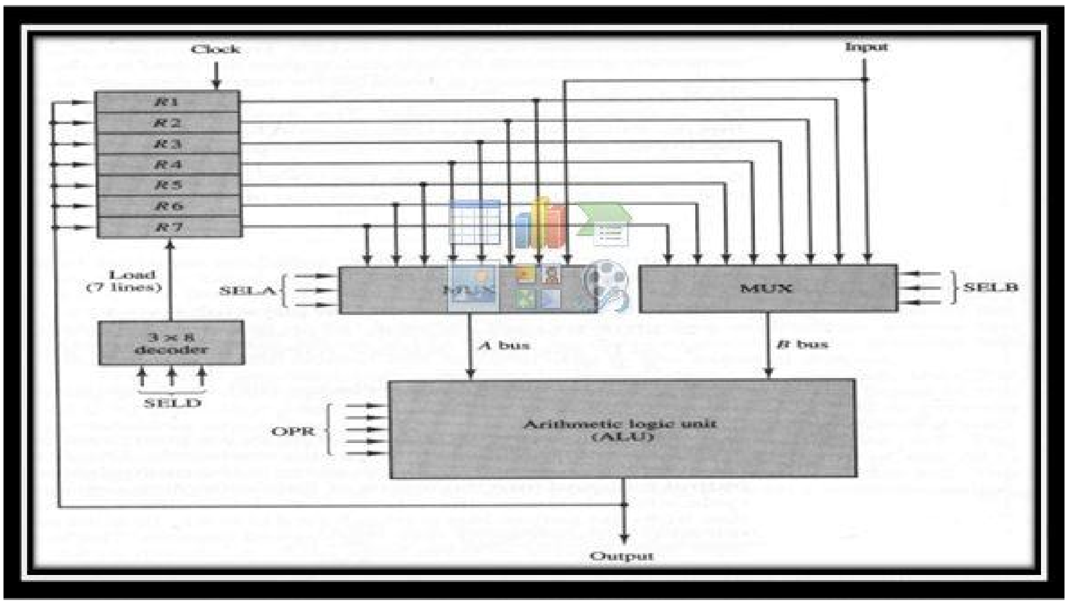
Introduction

CPU stands for Central processing unit. It performs the data processing operations in a computer. The major components of a CPU are register sets, arithmetic logic unit (ALU) and the control. Every component has its own functionalities. The register set is used to store and transfer the data. The ALU is responsible for performing various micro-operations. The control, i.e., the head is the supervisor of the information transfer.


data store
supervision of the information transfer
Register set
Control
Arithmetic logic unit (ALU)
performs microoperations

9.1 <u>General Register Organization</u>

The memory locations are needed for storing pointers, counters, return addresses, temporary results and partial products during any kind of operation. Having to refer to memory locations for such applications is time consuming because memory access is the most time consuming operation in a computer.It is more convenient and efficient to use registers to store these immediate values. If CPU has a large number of registers, a common bus is used to connect the registers.The registers communicate with each other not only for direct data transfer, but also while performing the micro-operations. So, a common unit is necessary that can perform all the arithmetic, logic and shift micro-operations in the processor.The general register organization contains registers, decoders, multiplexers and the arithmetic logic unit in which each component works in a responsible way for the management of data. This management involves the storage, retrieval, transfer and exchange of the data.



The output of each register is connected to two multiplexers (MUX) to form the two buses A and B. The selection lines in each multiplexer select one register or the input data for the particular bus.The A and B buses form the inputs to a common ALU.The operation selected in ALU determines the arithmetic or logic operation that is to be performed. The results of the micro-operations are available for output data and also go into the inputs of all the registers. The register that receives the information from the output bus is selected by the decoder. The decoder activates one of the register load inputs, thus providing a transfer path between the data in the output bus and the inputs of the selected destination register. The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the system.

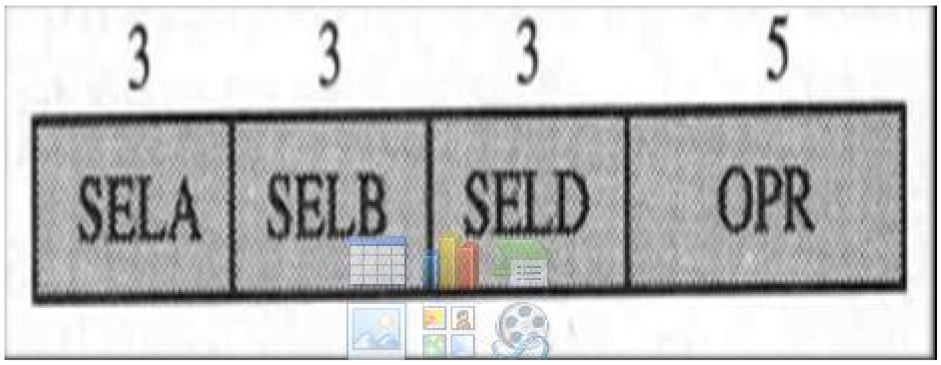
For example: R1🡐R2+R3 To perform this operation, the control must provide binary selection variables as:

1. MUX A selector SELA: is used to place the content of R2 into bus A. (SEL A): BUS A 🡐R2

2. MUX B selector SELB: is used to place the content of R3 into bus B. (SEL B): BUS B🡐R3

3. ALU operation selector (OPR): selects the arithmetic addition. (OPR): A+B

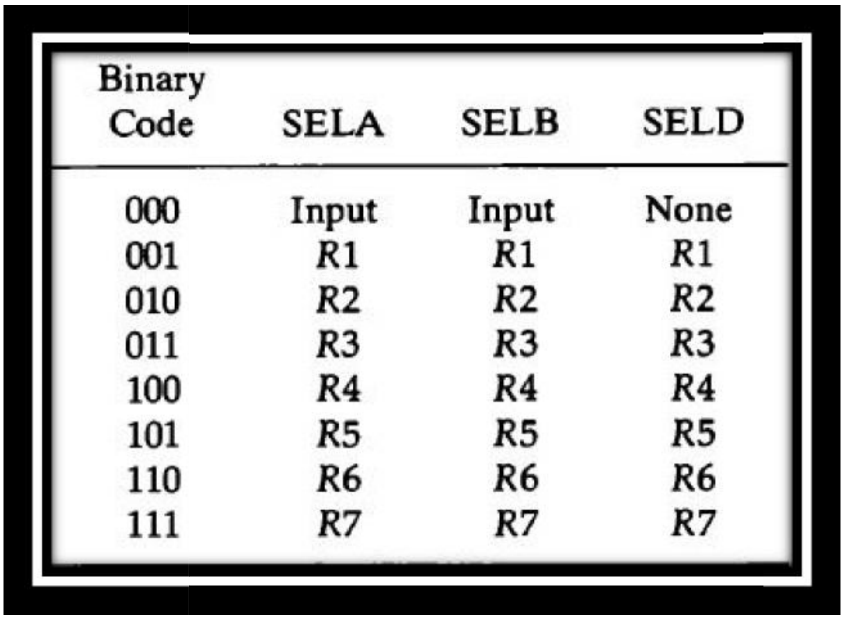
4. Decoder destination selector (SELD): is used to transfer the content of output bus into R1.



Control word

There are 14 binary selection inputs in the unit and their combined value specifies a control word.It consists of 4 fields.Three fields contain three bits each and one field has five bits.The three bits of SELA select a source register for the A input of ALU. The three bits of SELB select a source register for the B input of ALU. The three bits of SELD select a destination register using the decoder and its seven load outputs. The five bits of OPR selects one of the operations in ALU.The 14 bit control word when applied to selection inputs specifies a particular micro-operation.

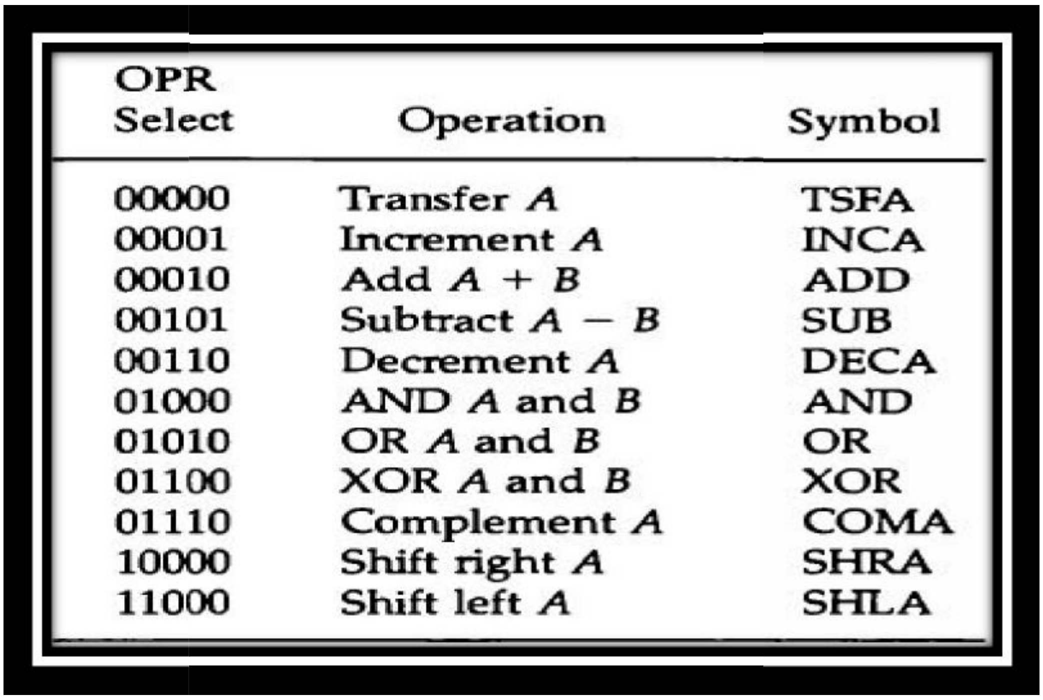
9.2 Encoding of register selection fields



Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

The three bit binary code listed in the first column of the table specifies the binary code for each of the three fields.The register selected by fields SELA, SELB and SELD is the one whose decimal number is equivalent to the binary number in the code.When SELA or SELB is 000, the corresponding multiplexer selects the external input data.When SELD=000, no destination register is selected but the contents of the output bus are available in the external output.

9.3 Encoding of ALU operations



OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add $A + B$	ADD
00101	Subtract $A - B$	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

The ALU provides arithmetic and logic micro-operations.The OPR field has five bits and each operation is designated with a symbolic name.

Stack

A useful feature that is included in the CPU of most computer systems is a stack.A stack is a storage device for storing information in such a manner that the item stored last is the first item retrieved (LIFO – last-in, first-out).The operation of a stack can be compared to a stack of trays. The last tray placed on the top of the stack is the first to be taken off.The stack in digital computers is essentially a memory unit with an address register that can count only (after an initial element is loaded into it).The register that holds the address for stack is called a stack pointer (SP) because its value always points out its top item in the stack.

9.4 Operations of a stack

There are two main operations which can be performed on a stack:

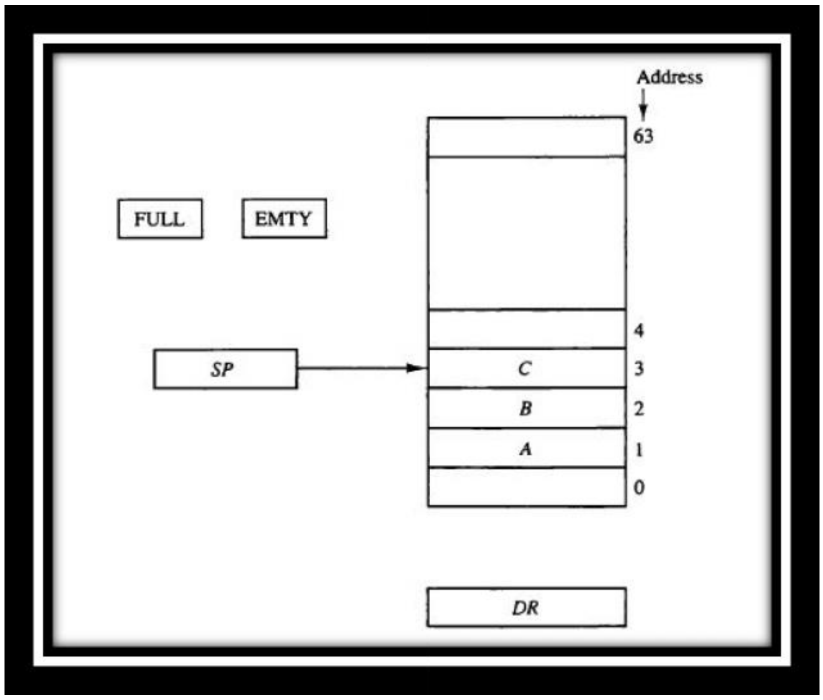
1) Push
2) Pop

The push operation deals with the insertion of elements in the stack and the pop operation deals with the deletion of elements from the stack. Both of these highly used operations deals with the stack pointer, i.e., top of the stack which always incremented and decremented depending upon the operation performed. These operations are simulated by incrementing or decrementing the stack pointer register.

9.5 Register stack

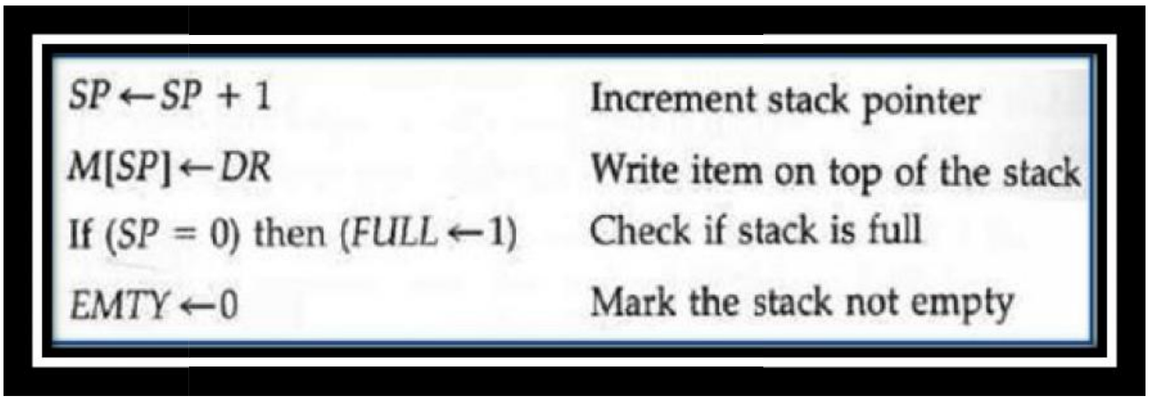
A stack can be placed in a portion of large memory or it can be organized as a collection of a finite number of memory words or registers.The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on the top of it.

9.6 64 word stack



Three items are placed in the stack: A, B and C, in that order. Item C is on the top of the stack so that the content of SP is now 3. To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now at the top of the stack since SP holds address 2. To insert a new item, the stack is pushed by incrementing SP and writing a word in the next higher location in the stack.

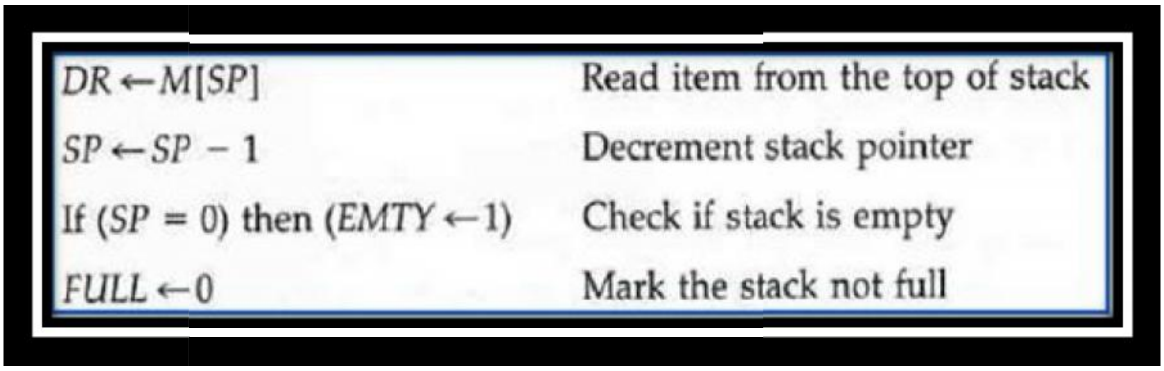
9.7 PUSH operation



$SP \leftarrow SP + 1$	Increment stack pointer
$M[SP] \leftarrow DR$	Write item on top of the stack
If $(SP = 0)$ then $(FULL \leftarrow 1)$	Check if stack is full
$EMTY \leftarrow 0$	Mark the stack not empty

The stack pointer is incremented so that it points to the address of the next higher word. The memory write operation inserts the word from DR into the top of the stack. The first item stored in the stack is at address 1.The last item is stored at address 0. If SP reaches 0, the stack is full of items, so FULL is set to 1. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 1.

9.8 POP operation



$DR \leftarrow M[SP]$	Read item from the top of stack
$SP \leftarrow SP - 1$	Decrement stack pointer
If $(SP = 0)$ then $(EMTY \leftarrow 1)$	Check if stack is empty
$FULL \leftarrow 0$	Mark the stack not full

The top item is read from the stack into DR. The stack pointer is then decremented. If its value reaches 0, then the stack is empty and then EMTY is set to 1. This condition is reached if the item read was in location 1.

9.9 Memory stack

Stack can also be implemented with a RAM attached to a CPU:

A portion of memory is assigned to a stack operation

A processor register is used as a stack pointer

A portion of memory is partitioned into three segments: program, data, and stack.Most computer do not provide hardware for checking stack overflow or underflow.If registers are used to store the upper limit (e.g. 3000) and the lower limit (e.g. 4000), then after push SP can be compared against the upper limit register, and after pull against the lower limit register. The advantage of the memory stack is that CPU can refer it without having to specify an address: the address in always in SP and automatically updated during a push or pop instruction.

Notations

Stack organization is very effective for evaluating arithmetic expressions.The common mathematical method of writing arithmetic expressions imposes difficulties when evaluated by a computer.The common arithmetic expressions are written in INFIX notation, with each operator written between the operands: A*B+C*D.To evaluate this expression, it is necessary to compute the product A*B, store the product while computing C*D, and then sum the two products. Two other notations used are: PREFIX and POSTFIX.

PREFIX notation: This representation is also known as polish notation which places the operator before the operands.

POSTFIX notation: It is referred to as reverse polish notation, places the operator after the operands.

9.10 Demonstration of representations

A+B	INFIX notation
+AB	PREFIX or POLISH notation
AB+	POSTFIX or REVERSE POLISH notation

9.11 Procedure of evaluation

The reverse polish notation is in a form suitable for stack manipulation, the expression A*B +C*D can be written as AB*CD*+.Scan the expression from left to right. When an operator is reached, perform the operation with the two operands found on the left side of the operator. Remove the two operands and the operator and replace them by the number obtained from the result of the operation. Continue to scan the expression and repeat the procedure for every operator encountered until there are no more operators.

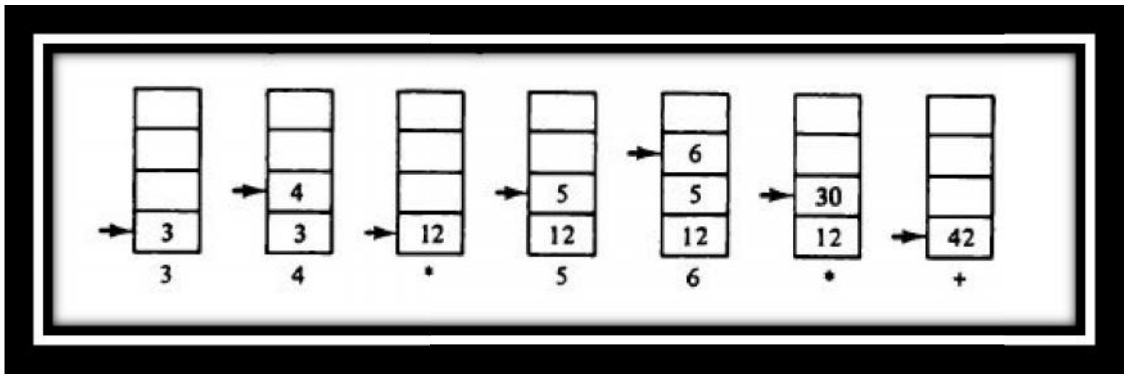
9.12 Evaluation of arithmetic expression

The postfix notation, combined with a stack arrangement of registers is the most efficient way known for evaluating arithmetic expressions.This procedure is employed in some electronic calculators and also in some computers.It is based on the fact that any arithmetic expression can be expressed in parentheses-free Polish notation

Consider the arithmetic expression: (3 * 4) + (5 * 6)

In reverse polish notation, it is expressed as: 3 4 * 5 6 * +

9.13 Representation in stack



Each box represents one stack operation and the arrow always points to the top of the stack.Scanning the expression from left to right, we encounter two operands. First the number 3 is pushed into the stack, then the number 4. Then *. This causes a multiplication of the two topmost items in the stack. The stack is then popped and the product is placed on the top of the stack, replacing the original operands.Next we encounter the two operands 5 and 6, so they are pushed into the stack. The stack operation that results from the next * replaces these two numbers by their product. The last operation causes an arithmetic addition to produce the final result of 42.

Summary

- CPU performs the data processing operations in a computer.
- If CPU has a large number of registers, a common bus is used to connect the registers.
- The registers communicate with each other not only for direct data transfer, but also while performing the micro-operations.
- The registers communicate with each other not only for direct data transfer, but also while performing the micro-operations.
- There are 14 binary selection inputs in the unit and their combined value specifies a control word.
- A stack is a storage device for storing information in such a manner that the item stored last is the first item retrieved (LIFO – last-in, first-out).
- Stack organization is very effective for evaluating arithmetic expressions.

Keywords

- Stack: It is a useful feature that is included in the CPU of most computer systems.
- Stack pointer: The register that holds the address for stack.
- PREFIX notation: This representation is also known as polish notation which places the operator before the operands.
- POSTFIX notation: It is referred to as reverse polish notation, places the operator after the operands.

Self Assessment Questions:

1. Which of the following component of CPU performs micro-operations?

A. Control
B. ALU
C. Register set
D. None of the above

2. The register that receives the information from the output bus is selected by the ___________

A. Encoder
B. Multiplexer
C. Decoder
D. Demultiplexer

3. How many fields are included in a control word?

A. 2
B. 3
C. 4

D. 5

4. Which of the following field is of 5 bits?

A. SELA

B. SELB

C. SELD

D. OPR

5. The fields SELA, SELB and SELD each consists of

A. 2

B. 3

C. 4

D. 5

6. A stack follows the rule of

A. FIFO

B. LIFO

C. LILO

D. None of the above

7. The register that holds the address for stack is called a _________

A. Stack Pointer

B. Stack Register

C. Stack Pop

D. Stack Push

8. The value of stack pointer always points towards the _____ of the stack

A. Bottom

B. Left

C. Right

D. Top

9. When we are removing any item from the stack, the content in SP will be

A. Incremented

B. Decremented

C. Remains same

D. None of the above

10. When we are inserting any item to the stack, the content in SP will be

A. Incremented

B. Decremented

C. Remains same

D. None of the above

11. The arithmetic expressions can be effectively calculated by _______ organization

A. Stack

B. Queue

C. Graph

D. Tree

12. In polished notation, we place the operator is placed _______

A. Between the operands

B. Before the operands

C. After the operands

D. The operator is not placed

13. Which of the following referred to as reverse polished notation?

A. Infix notation

B. Prefix notation

C. Postfix notation

D. None of the above

14. The expression AB+ refers to as ____________

A. Infix notation expression

B. Prefix notation expression

C. Postfix notation expression

D. None of the above

15. Which of the following form is best suitable for stack manipulation?

A. Polished notation

B. Reverse Polished notation

C. Accurate Polished notation

D. None of the above

Answers for Self Assessment

1.	B	2.	C	3.	C	4.	D	5.	B
6.	B	7.	A	8.	D	9.	B	10.	A
11.	A	12.	B	13.	C	14.	C	15.	B

Review Questions:

1. What is a CPU? Explain all its components.
2. What is general register organization? Explain its components.

3. What is a control word? Explain the encoding of register selection fields and ALU operations.
4. What is a stack? Explain its operations.
5. What are notations of a stack? Explain infix, prefix and postfix notations with examples.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education

Asia, 2002.

Web Links

https://www.geeksforgeeks.org/introduction-of-general-register-based-cpu-organization/

Unit 10: Addressing Modes

CONTENTS

Objectives

After studying this unit, you will be able to

- Understand the format of an instruction.
- Understand different types of instructions.
- Understand the addressing modes.
- Understand the different modes of an instruction.
- Understand the use of effective address

Introduction

A computer will usually have a variety of instruction code formats.It is the function of the control unit within the CPU to interpret each instruction code and provide the necessary control functions needed to process the instruction.The format of an instruction is usually depicted in a rectangular box symbolizing the bits of the instruction as they appear in memory words or in a control register.

10.1 Fields of Instruction

The bits of the instruction are divided into groups called fields. The most common fields found in instruction formats are:

- Operation code: An operation code field that specifies the operation to be performed.
- Address field: An address field that designates a memory address or a processor register.
- Mode field: A mode field that specifies the way the operand or the effective address is determined.

Other special fields are sometimes employed under certain circumstances, as for example a field that gives the number of shifts in a shift-type instruction. The operation code field of an instruction is a group of bits that define various processor operations, such as add, subtract, complement, and shift. The bits that define the mode field of an instruction code specify a variety of alternatives for choosing the operands from the given address.

Register Address

Operations specified by computer instructions are executed on some data stored in memory or processor registers. Operands residing in memory are specified by their memory address. Operands residing in processor registers are specified with a register address. A register address is a binary number of k bits that defines one of 2^k registers in the CPU. Thus a CPU with 16 processor registers R0through R15 will have a register address field of four bits. The binary number 0101, for example, will designate register R5.

10.2 Types of CPU Organization

Computers may have instructions of several different lengths containing varying number of addresses.The number of address fields in the instruction format of a computer depends on the internal organization of its registers.Most computers fall into one of three types of CPU organizations:

- Single accumulator organization.
- General register organization.
- Stack organization.

Single Accumulator Organization:

An example of an accumulator-type organization is the basic computer.All operations are performed with an implied accumulator register.The instruction format in this type of computer uses one address field.

The instruction that specifies an arithmetic addition is defined by an assembly language instruction as

ADD Xwhere X is the address of the operand.

The ADD instruction in this case results in the operation AC <- AC + M[X]. AC is the accumulator register and M[X] symbolizes the memory word located at address X.

10.3 General Register Type Organization

The instruction format in this type of computer needs three register address fields. Thus the instruction for an arithmetic addition may be written in an assembly language as

ADD R1, R2,R3to denote the operation R1 <- R2 + R3.

The number of address fields in the instruction can be reduced from three to two if the destination register is the same as one of the source registers. Thus the instruction ADD R1,R2would denote the operation R1 <- R1+R2.Only register addresses for R1 and R2 need be specified in this instruction.

Computers with multiple processor registers use the move instruction with a mnemonic MOV to symbolize a transfer instruction. Thus the instruction MOVR1, R2denotes the transfer R1 <- R2 (or R2<-R1, depending on the particular computer). Thus transfer-type instructions need two address fields to specify the source and the destination.

General register-type computers employ two or three address fields in their instruction format. Each address field may specify a processor register or a memory word. An instruction symbolized byADD R1, Xwould specify the operation R1 <-R1 + M[X]. It has two address fields, one for register R1 and the other for the memory address X.

10.4 Stack Organization

Computers with stack organization would have PUSH and POP instructions which require an address field.

PUSH X

Thus the instruction will push the word at address X to the top of the stack. The stack pointer is updated automatically. Operation-type instructions do not need an address field in stack-organized computers. This is because the operation is performed on the two items that are on top of the stack. The instructionADDin a stack computer consists of an operation code only with no address field. This operation has the effect of popping the two top numbers from the stack, adding the numbers, and pushing the sum into the stack. There is no need to specify operands with an address field since all operands are implied to be in the stack.Most computers fall into one of the three types of organizations that have just been described.

Some computers combine features from more than one organizational structure. For example, the Intel 8080 microprocessor has seven CPU registers, one of which is an accumulator register.As a consequence, the processor has some of the characteristics of a general register type and some of the characteristics of an accumulator type.

All arithmetic and logic instructions, as well as the load and store instructions, use the accumulator register, so these instructions have only one address field.On the other hand, instructions that transfer data among the seven processor registers have a format that contains two register address fields.Moreover, the Intel 8080 processor has a stack pointer and instructions to push and pop from a memory stack. The processor, however, does not have the zero-address-type instructions which are characteristic of a stack-organized CPU.

Evaluation of Arithmetic Expression

To illustrate the influence of the number of addresses on computer programs, we will evaluate the arithmetic statement X = (A + B) * (C + D)using zero, one, two, or three address instructions. We will use the symbols

1) ADD, SUB, MUL, and DIV for the four arithmetic operations;

2) MOV for the transfer-type operation;

3) LOAD and STORE for transfers to and from memory and AC register.

We will assume that the operands are in memory addresses A, B, C, and D, and the result must be stored in memory at address X.

10.5 Three-Address Instruction

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates

X = (A + B) * (C + D) is

ADD	R1, A, B	R1 <- M[A] + M[B]
ADD	R2, C, D	R2 <- M[C] + M[D]
MUL	X, R1, R2	M [X] <- R1 * R2

It is assumed that the computer has two processor registers, R1 and R2. The symbol M[A] denotes the operand at memory address symbolized by A.The advantage of the three-address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.

A commercial computer that uses three-address instructions is the Cyber 170.

10.6 Two-Address Instruction

Two-address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word. The program to evaluate X = (A + B) * (C + D) is as follows:

MOV	R1, A	R1 <- M[A]
ADD	R1, B	R1 <- R1 + M[B]
MOV	R2, C	R2 <- M[C]
ADD	R2, D	R2 <- R2 + M[D]
MUL	R1, R2	R1 <- R1 * R2
MOV	X, R1	M[X] <- R1

The MOV instruction moves or transfers the operands to and from memory and processor registers.The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

10.7 One Address Instruction

One-address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division there is a need for a second register. However, here we will neglect the second register and assume that the AC contains the result of all operations.

The program to evaluate X = (A + B) * (C + D) is

LOAD	A	AC <- M[A]
ADD	B	AC <- AC + M[B]
STORE	T	M[T] <- AC
LOAD	C	AC <- M[C]
ADD	D	AC <- AC + M[D]
MUL	T	AC <- AC * M[T]

```
STORE           X                    M[X] <- AC
```

All operations are done between the AC register and a memory operand. T is the address of a temporary memory location required for storing the intermediate result.

10.8 Zero Address Instruction

A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack. The following program shows how X = (A + B) * (C + D) will be written for a stack organized computer. TOS stands for top of stack.

```
PUSH        A           TOS <- A
PUSH        B           TOS <- B
ADD                     TOS <- (A+B)
PUSH        C           TOS <- C
PUSH        D           TOS <- D
ADD                     TOS <- (C+D)
MUL                     TOS <- (C+D) * (A+B)
POP         X           M[X] <- TOS
```

To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse Polish notation. The name "zero-address" is given to this type of computer because of the absence of an address field in the computational instructions.

10.9 RISC Instruction

The instruction set of a typical RISC processor is restricted
to the use of load and store instructions when communicating between memory and CPU. All other instructions are executed within the registers of the CPU without referring to memory. A program for a RISC-type CPU consists of LOAD and STORE instructions that have one memory and one register address, and computational-type instructions that have three addresses with all three specifying processor registers. The following is a program to evaluate

```
X = (A + B) * (C + D)
LOAD        R1, A           R1 <- M[A]
LOAD        R2, B           R2 <- M[B]
LOAD        R3, C           R3 <- M[C]
LOAD        R4, D           R4 <- M[D]
ADD         R1, R1, R2      R1 <- R1 + R2
ADD         R3, R3, R2      R3 <- R3 + R4
MUL         R1, R1, R3      R1 <- R1 * R3
STORE       X, R1           M[X] <- R1
```

The load instructions transfer the operands from memory to CPU registers. The add and multiply operations are executed with data in the registers without accessing memory.The result of the computations is then stored in memory with a store instruction.

Addressing Modes

The operation field of an instruction specifies the operation to be performed. This operation must be executed on some data stored in computer registers or memory words. The way the operands are chosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

Computers use addressing mode techniques for the purpose of accommodating one or both of the following provisions:

1. To give programming versatility to the user by providing such facilities as pointers to memory, counters for loop control, indexing of data, and program relocation.

2. To reduce the number of bits in the addressing field of the instruction.

The availability of the addressing modes gives the experienced assembly language programmer flexibility for writing programs that are more efficient with respect to the number of instructions and execution time.To understand the various addressing modes to be presented in this section, it is imperative that we understand the basic operation cycle of the computer.The control unit of a computer is designed to go through an instruction cycle that is divided into three major phases:

- Fetch the instruction from memory.
- Decode the instruction.
- Execute the instruction.

Program Counter

There is one register in the computer called the program counter or PC that keeps track of the instructions in the program stored in memory. PC holds the address of the instruction to be executed next and is incremented each time an instruction is fetched from memory. The decoding done in step 2 determines the operation to be performed, the addressing mode of the instruction, and the location of the operands. The computer then executes the instruction and returns to step 1 to fetch the next instruction in sequence.In some computers the addressing mode of the instruction is specified with a distinct binary code, just like the operation code is specified. Other computers use a single binary code that designates both the operation and the mode of the instruction. Instructions may be defined with a variety of addressing modes, and sometimes, two or more addressing modes are combined in one instruction.

Mode field

The operation code specifies the operation to be per-formed. The mode field is used to locate the operands needed for the operation. There may or may not be an address field in the instruction. If there is an address field, it may designate a memory address or a processor register. Moreover, as discussed in the preceding section, the instruction may have more than one address field, and each address field may be associated with its own particular addressing mode.Although most addressing modes modify the address field of the instruction, there are two modes that need no address field at all. These are the implied and immediate modes.

10.10 Implied Mode

In this mode the operands are specified implicitly in the definition of the instruction. For example, the instruction "complement accumulator" is an implied-mode instruction because the operand in the accumulator register is implied in the definition of the instruction. In fact, all register reference instructions that use an accumulator are implied-mode instructions. Zero-address instructions in a stack-organized computer are implied-mode instructions since the operands are implied to be on top of the stack.

Opcode	Mode	Address

10.11 Immediate Mode

In this mode the operand is specified in the instruction itself. In other words, an immediate-mode instruction has an operand field rather than an address field.The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.Immediate-mode instructions are useful for initializing registers to a constant value. It was mentioned previously that the address field of an instruction may specify either a memory word or a processor register.When the address field specifies a processor register, the instruction is said to be in the register mode.

10.12 Register Mode

In this mode the operands are in registers that reside within the CPU. The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2^k registers.

10.13 Register Indirect Mode

In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory. In other words, the selected register contains the address of the operand rather than the operand itself.Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction. A reference to the register is then equivalent to specifying a memory address. The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

10.14 Auto-increment or auto-decrement mode

This is similar to the register indirect mode except that the register is incremented or decremented after (or before) its value is used to access memory. When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction. However, because it is such a common requirement, some computers incorporate a special mode that automatically increments or decrements the content of the register after data access.

Effective Address

The address field of an instruction is used by the control unit in the CPU to obtain the operand from memory.Sometimes the value given in the address field is the address of the operand, but sometimes it is just an address from which the address of the operand is calculated. To differentiate among the various addressing modes it is necessary to distinguish between the address part of the instruction and the effective address used by the control when executing the instruction.The *effective address* is defined to be the memory address obtained from the computation dictated by the given addressing mode. The effective address is the address of the operand in a computational - type instruction. It is the address where control branches in response to a branch - type instruction.

10.15 Direct Address Mode

In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of the instruction.In a branch-type instruction the address field specifies the actual branch address.

10.16 Indirect Address Mode

In this mode the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.**A** few addressing modes require that the address field of the instruction be added to the content of **a** specific register in the CPU.

The effective address in these modes is obtained from the following computation:

Effective address = address part of instruction + content of CPU register

The CPU register used in the computation may be the program counter, an index register, or a base register. In either case we have a different addressing mode which is used for a different application.

10.17 Relative Address Mode

In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address. The address part of the instruction is usually a signed number (in 2's complement representation) which can be either positive or negative. When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction. To clarify with an example, assume that the program counter contains the number 825 and the address part of the instruction contains the number 24.

The instruction at location 825 is read from memory during the fetch phase and the program counter is then incremented by one to 826. The effective address computation for the relative address mode is826 + 24 = 850. This is 24 memory locations forward from the address of the next instruction. Relative addressing is often used with branch-type instructions when the branch address is in the area surrounding the instruction word itself.It results in a shorter address field in the instruction format since the relative address can be specified with a smaller number of bits compared to the number of bits required to designate the entire memory address.

10.18 Index Address Mode

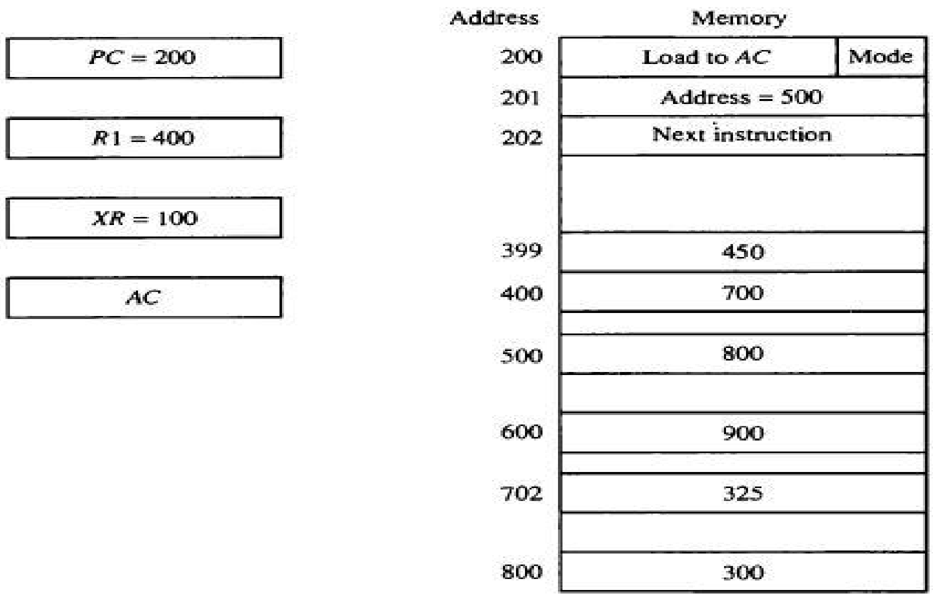
In this mode the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory. Each operand in the array is stored in memory relative to the beginning address. The distance between the beginning address and the address of the operand is the index value stored in the index register. Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value. The index register can be incremented to facilitate access to consecutive operands. Note that if an index type instruction does not include an address field in its format, the instruction converts to the register indirect mode of operation.Some computers dedicate one CPU register to function solely as an index register. This register is involved implicitly when the index-mode instruction is used. In computers with many processor registers, any one of the CPU registers can contain the index number. In such a case the register must be specified explicitly in a register field within the instruction format.

10.19 Base Register Addressing Mode

In this mode the content of a base register is added to the address part of the instruction to obtain the effective address. This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.The difference between the two modes is in the way they are used rather than in the way that they are computed. An index register is assumed to hold an index number that is relative to the address part of the instruction.A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address. The base register addressing mode is used in computers to facilitate the relocation of programs in memory.When programs and data are moved from one segment of memory to another, as required in multiprogramming systems, the address values of instructions must reflect this change of position. With a base register, the displacement values of instructions do not have to change.Only the value of the base register requires updating to reflect the beginning of a new memory segment.

To show the differences between the various modes, we will show the effect of the addressing modes on the instruction. The two-word instruction at address 200 and 201 is a “load to AC” instruction with an address field equal to 500.The first word of the instruction specifies the operation code and mode, and the second word specifies the address part. PC has the value 200 for fetching this instruction.

The content of processor register R1 is 400, and the content of an index register XR is 100. AC receives the operand after the instruction is executed.



The mode field of the instruction can specify any one of a number of modes.

For each possible mode we calculate the effective address and the operand that must be loaded into AC.

In the direct address mode the effective address is the address part of the instruction 500 and the operand to be loaded into AC is 800.

In the immediate mode the second word of the instruction is taken as the operand rather than an address, so 500 is loaded into AC.

(The effective address in this case is 201.)

In the indirect mode the effective address is stored in memory at address 500.

Therefore, the effective address is 800 and the operand is 300.

In the relative mode the effective address is 500 + 202 = 702 and the operand is 325.

Note that the value in PC after the fetch phase and during the execute phase is 202.)

In the index mode the effective address is

XR + 500 = 100 + 500 = 600 and the operand is 900. In the register mode the operand is in R1 and 400 is loaded into AC.

There is no effective address in this case.

In the register indirect mode the effective address is 400, equal to the content of R1 and the operand loaded into AC is 700.

The auto-increment mode is the same as the register in direct mode except that R1 is incremented to 401 after the execution of the instruction.

The auto-decrement mode decrements R1 to 399 prior to the execution of the instruction.

The operand loaded into AC is now 450.

Addressing Mode	Effective Address	Content of AC
Direct Address	500	800

Immediate Operand	201	500
Indirect Address	800	300
Relative Address	702	325
Indexed Address	600	900
Register	--	400
Register Indirect	400	700
Auto increment	400	700
Auto decrement	399	450

Summary

- The bits of the instruction are divided into groups called fields.
- An operation code field that specifies the operation to be performed.
- An address field that designates a memory address or a processor register.
- A mode field that specifies the way the operand or the effective address is determined.
- Operands residing in memory are specified by their memory address.
- Operands residing in processor registers are specified with a register address.
- Most computers fall into one of three types of CPU organizations: single accumulator organization, general register organization and stack organization.

Keywords

Implied Mode: In this mode the operands are specified implicitly in the definition of the instruction.

Immediate Mode: The immediate-mode instruction has an operand field rather than an address field.

Register Mode: In this mode the operands are in registers that reside within the CPU.

Register indirect mode: In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in memory.

Effective address: The effective addressis defined to be the memory address obtained from the computation dictated by the given addressing mode.

Indirect address mode: In this mode the address field of the instruction gives the address where the effective address is stored in memory.

Self Assessment

1. What all contains in a common instruction format?

A. Field of operation

B. Field of address

C. Field of mode

D. All the fields mentioned above

2. The instruction of kind MUL X defines

A. General register organization
B. Stack organization
C. Stack Accumulator organization
D. None of the above

3. The instruction MUL R1, R2 represents
A. Single accumulator organization
B. General register type organization
C. Stack organization
D. None of the above

4. The operations are performed between a memory operand and AC register. What kind of instruction is discussed here?
A. Zero address instruction
B. One address instruction
C. Two address instruction
D. Three address instruction

5. When the communication between CPU and memory occurs, which instruction set is just bound to use store and load instructions?
A. RISC instructions
B. CISC instructions
C. MISC instructions
D. None of the above

6. The PUSH and POP operations are performed in what kind of organization?
A. General Register Organization
B. Queue Organization
C. Stack Organization
D. Single Accumulator Organization

7. The register reference instructions which employs accumulator are known as ______.
A. Relative mode
B. Implied mode
C. Immediate mode
D. None of the above

8. What kind of instructions does not require any kind of address field?
A. Implied and immediate mode instructions
B. Implied and relative mode instructions
C. Relative and immediate mode instructions
D. None of the above

9. The effective address is calculated by adding the content of base register with the address part of instruction. What kind of addressing mode is discussed here?
A. Indexed addressing mode
B. Relative addressing mode

C. Base register addressing mode

D. None of the above

10. In indirect address mode, the effective address is calculated as:

A. Effective address = address part of instruction - content of CPU register

B. Effective address = address part of instruction + content of CPU register

C. Effective address = address part of instruction * content of CPU register

D. Effective address = address part of instruction / content of CPU register

11. The operand is specified in the instruction itself. What kind of mode is specified here?

A. Implied mode

B. Register mode

C. Immediate mode

D. Relative address mode

12. THE CLA or CME represents

A. Implied mode

B. Immediate mode

C. Register mode

D. Relative address mode

13. In an operation, which field is used to locate the operands?

A. Whole instruction

B. Operation field

C. Mode field

D. None of the above

14. What increments every time an instruction is fetched from the memory?

A. Instruction register

B. Input register

C. Output register

D. Program Counter

15. What holds the address of instruction which is to be executed next?

A. Instruction register

B. Input register

C. Output register

D. Program Counter

Answer for Self Assessment

1.	A	2.	C	3.	B	4.	B	5.	A
6.	C	7.	B	8.	A	9.	B	10.	B
11.	C	12.	A	13.	C	14.	D	15.	D

Review Questions

1. What is an instruction? Explain its fields.
2. What are the different types of CPU organization? Explain.
3. What is three address instructions? Explain with example.
4. What are two address and one address instructions? Explain with examples.
5. What are zero address and RISC instructions? Explain with examples.
6. What is a mode field? Explain different types of mode fields.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education, Asia, 2002.

Web Links

https://www.geeksforgeeks.org/addressing-modes/

Unit 11: Pipeline Processing

CONTENTS

Objectives:

After studying this unit, you will be able to

- understand the parallel processing
- know the classification of parallel processing
- understand the pipeline
- understand the different types of pipeline
- know the pipeline hazards and their possible solutions

Introduction:

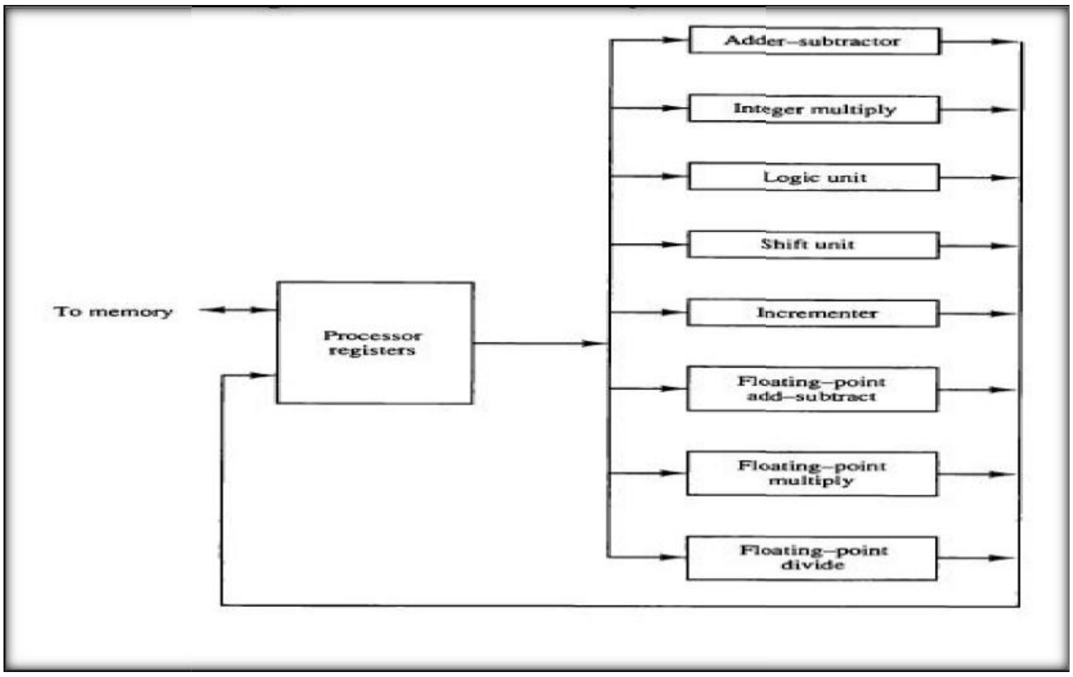
Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneous data-processing task for the purpose of increasing the computational speed of a computer system.Parallel processing is used to provide simultaneous data-processing task. It also performs concurrent data processing. A system may have two or more ALUs and two or more processors. The purpose of employing parallel processing is speeding up the computer processing capability and increasing its throughput.

11.1 Different levels of complexity

Parallel processing can be viewed from various levels of complexity:

1) Between parallel and serial operations.
2) Having a multiplicity of functional units.

In the image shown below, a processor with multiple functional parts is shown. There are various modules which performs the different functionalities. These are adder-subtractors, integer multiply, logic unit, shift unit, floating point add-subtract, floating point multiply and floating point divide.



11.2 Classification of parallel processing

The classification introduced by M.J.Flynn considers the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.

Instruction stream: The sequence of instructions read from memory.

Data stream: The operations performed on the data in the processor.

Parallel processing may occur in the instruction stream, in the data stream, or in both.

Flynn's classification divides computers into four major groups as follows:

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data stream (SIMD)
- Multiple instruction stream, single data stream (MISD)
- Multiple instruction stream, multiple data stream(MIMD)

Single instruction stream, single data stream

The organization of a single computer contains a control unit, a processor unit, and a memory unit. Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities.

Single instruction stream, multiple data stream

Thisorganization includes many processing units under the supervision of a common control unit. All processors receive the same instruction from the control unit but operate on different items of data.

Multiple instruction stream, single data stream

This kind of organization is just of theoretical interest, there is no practical application found on this classification.

Multiple instruction stream, multiple data stream

It refers to a computer system capable of processing several programs at the same time.

11.3 Pipelining

It is a technique of decomposing a sequential process into sub-operations. It is a collection of processing segments.The final result is obtained after the data have passed through all segments. The term pipeline implies the flow of information.Several computations can be seen in progress in distinct segments at the same time.The overlapping of computation is there but registers provide isolation.

Flow of information

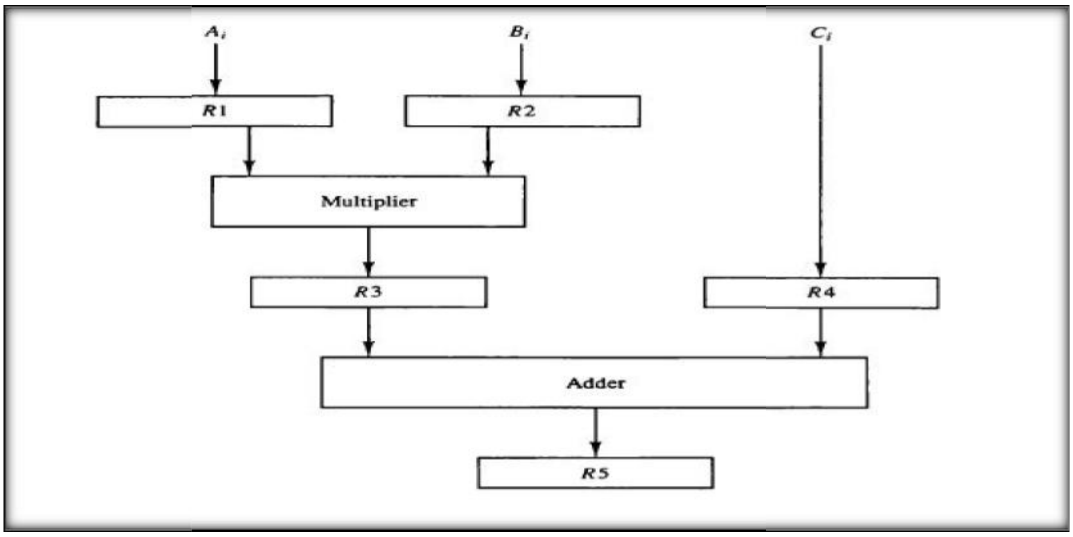
Each segment consists of an input register followed by a combinational circuit. The output of the combinational circuit in a given segment is applied to the input register of the next segment. The information flows through the pipeline one step at a time. Suppose we want to apply the combined multiply and add operations with a stream of numbers.

$A_i * B_i + C_i$ for i=1,2,3,…….7

R1 through R5 are registers that receive new data with every clock pulse. The multiplier and adder are combinational circuits.

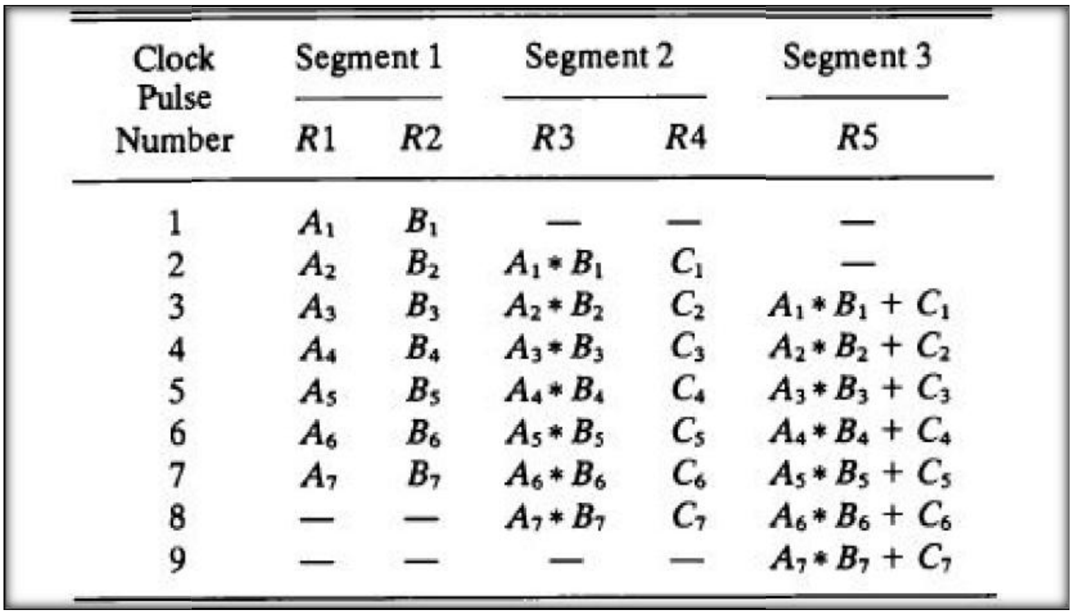
The suboperations performed in each segment of the pipeline are as follows:

Rl<-A_i, R2<-B_i	Input $A_i$and B_i
R3<- Rl*R2, R4<-C_i	Multiply and input C_i
R5<-R3+R4	Add C_i to product



Effect of each clock pulse

Every operation is done with the clock pulses.



Clock Pulse Number	Segment 1		Segment 2		Segment 3
	$R1$	$R2$	$R3$	$R4$	$R5$
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

11.4 Applicability of the pipelining

There are two areas where the pipeline organization is applicable:

- Arithmetic pipeline
- Instruction pipeline

Arithmetic pipeline

It is used to implement:

A) Floating-point operations,

B) Multiplication of fixed-point numbers,

C) Similar computations encountered in scientific problems.

The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.

X=A * 2a, Y=B * 2b.

The sub-operations that are performed in the four segments are:

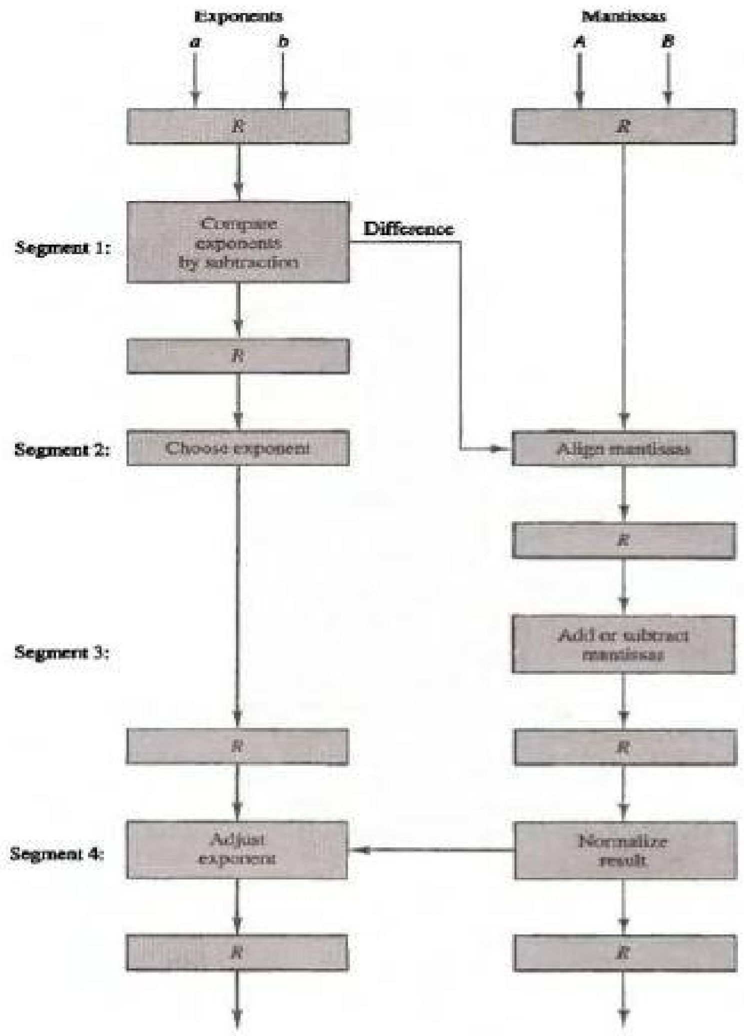
- Compare the exponents.
- Align the mantissas.
- Add or subtract the mantissas.
- Normalize the result.

There are two conditions which can occur here. These are overflow and underflow.

A) **Overflow:** The mantissa of the sum or difference is shifted right and the exponent incremented by one.

B) **Underflow:** The number of leading zeros in the mantissa determines the number of left shifts in the mantissa and the number that must be subtracted from the exponent.

Pipeline for floating point addition and subtraction



Consider the two normalized floating point numbers:

$X= 0.9504 * 10^3$, $Y=0.8200 * 10^2$

1st segment: Two exponents are subtracted, 3-2=1. The larger exponent 3 is chosen as the exponent of the result.

2nd segment: It shifts the mantissa of Y to the right to obtain: $X= 0.9504 * 10^3$, $Y=0.08200 * 10^{3.}$

3rd segment: Adds the two mantissa: $Z= 1.0324 * 10^3$

4th segment: Normalization of values, $Z=0.10324 * 10^4$

Instruction pipeline

Pipeline processing can occur not only in the data stream but in the instruction stream as well. The instruction's fetch and execute phases to overlap and perform simultaneous operations.It may cause a branch out of sequence - in that case the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction must be discarded. It follows the FIFO rule. Consider a computer with an instruction fetch unit and an instruction execution unit designed to provide **a** two-segment pipeline. The instruction fetch segment can be implemented by means of a first-in, first-out (FIFO) buffer. Whenever the execution unit is not using memory, the control increments the program counter and uses its address value to read consecutive instructions from memory.

11.5 Processing of each instruction

The computer needs to process each instruction with the following sequence of steps:

- Fetch the instruction from memory.
- Decode the instruction.
- Calculate the effective address.
- Fetch the operands from memory.
- Execute the instruction.
- Store the result in the proper place.

11.6 Difficulties in instruction pipeline

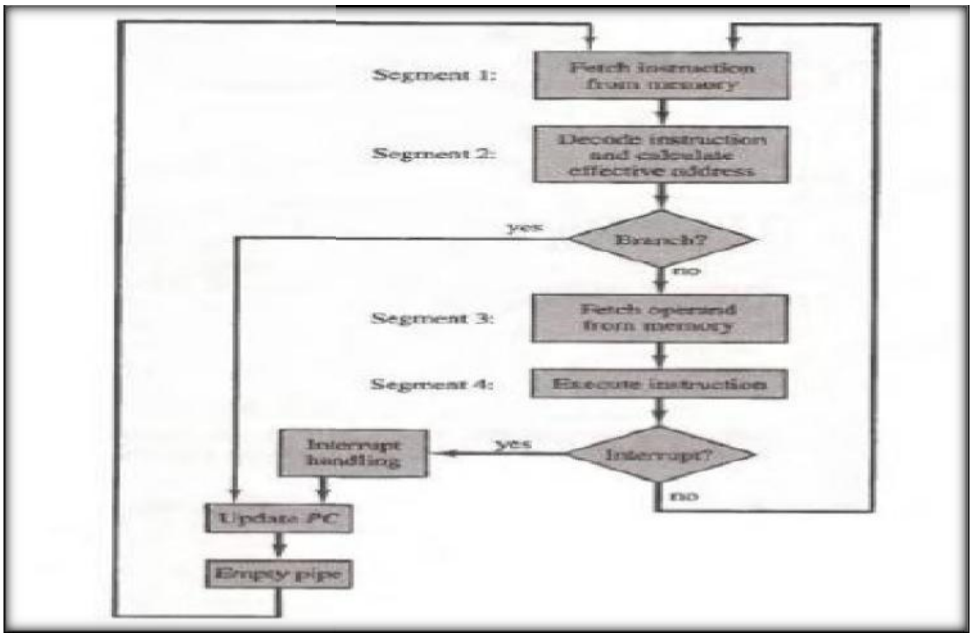
1) Different segments may take different times to operate on the incoming information.

2) Some segments are skipped for certain operations.

3) Two or more segments may require memory access at the same time, causing one segment to wait until another is finished with the memory.

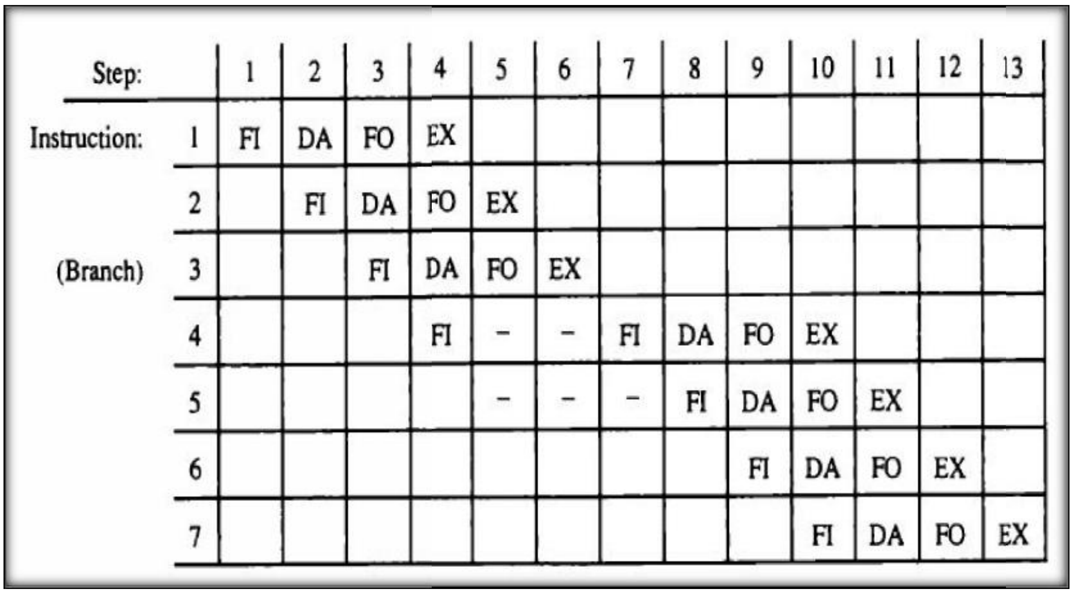
11.7 Four-Segment Instruction Pipeline

Assume that the decoding of the instruction can be combined with the calculation of the effective address into one segment.



Once in a while, an instruction in the sequence may be a program control type that causes a branch out of normal sequence - in that case the pending operations in the last two segments are completed and all information stored in the instruction buffer is deleted. The pipeline then restarts from the new address stored in the program counter. Similarly, an interrupt request, when acknowledged, will cause the pipeline to empty and start again from a new address value.

11.8 <u>Timing of instruction pipeline</u>



Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	–	–	FI	DA	FO	EX			
	5					–	–	–	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Pipeline hazards

The problems that occur in the pipeline are called hazards. Hazards that arise in the pipeline prevent the next instruction from executing during its designated clock cycle.

1) Data dependency

2) Branching of instructions

11.9 Data dependency

A data dependency occurs when an instruction needs data that are not yet available. Similarly, an address dependency may occur when an operand address cannot be calculated because the information needed by the addressing mode is not available.

Dealing with data dependency

Pipelined computers deal with such conflicts between data dependencies in a variety of ways:

1) Hardware interlocks
2) Operand forwarding
3) Delayed load

11.10 Branching of instructions

This is the Occurrence of branch instructions. An unconditional branch always alters the sequential program flow by loading the program counter with the target address.In a conditional branch, the control selects the target instruction if the condition is satisfied or the next sequential instruction if the condition is not satisfied.

Dealing with branching of instructions

Pipelined computers employ various hardware techniques for this:

1) Pre-fetch target instruction
2) Branch target buffer
3) Loop buffer
4) Branch prediction
5) Delayed branch

Summary:

- Pipelining is a technique of decomposing a sequential process into sub-operations.
- There are two areas where the pipeline organization is applicable: arithmetic pipeline and instruction pipeline.
- The sub-operations are per formed in the four segments. These are compare the exponents, align the mantissas, add or subtract the mantissas and normalize the result.
- Pipeline processing can occur not only in the data stream but in the instruction stream as well.
- The instructions fetch and execute phases to overlap and perform simultaneous operations.

Keywords:

Overflow: The mantissa of the sum or difference is shifted right and the exponent incremented by one.

Underflow: The number of leading zeros in the mantissa determines the number of left shifts in the mantissa and the number that must be subtracted from the exponent.

Parallel Processing: Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneous data-processing task for the purpose of increasing the computational speed of a computer system.

SISD:SISD represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.

SIMD: SIMD represents an organization that includes many processing units under the supervision of a common control unit.

MIMD:MIMD organization refers to a computer system capable of processingseveralprogramsatthesametime.

Delayed load: A procedure employed in some computers is to give the responsibility for solving data conflicts problems to the compiler that translates the high-level programming language into a machine language program.

Self Assessment

1. Where can we apply the concept of pipeline organization?

A. Arithmetic pipeline

B. Instruction pipeline

C. Both arithmetic and instruction pipelines

D. None of the above

2. In which condition, the mantissa of the sum or difference is shifted right and the exponent incremented by one?

A. Overflow

B. Underflow

C. Baseflow

D. None of the above

3. In which condition, the number of leading zeros in the mantissa determines the number of left shifts in the mantissa and the number that must be subtracted from the exponent?

A. Overflow

B. Underflow

C. Baseflow

D. None of the above

4. The instruction fetch segment can be implemented by means of _______ buffer.

A. FIFO

B. LIFO

C. FILO

D. LILO

5. ________ occurs when an operand address cannot be calculated

A. Data dependency

B. Address dependency

C. Calculation dependency

D. None of the above

6. ________ occurs when an instruction needs data that are not available yet.

A. Data dependency

B. Address dependency

C. Calculation dependency

D. None of the above

7. Which of the following defines a way for dealing conflicts of data dependencies?
A. Hardware interlocks
B. Operand forwarding
C. Delayed load
D. All hardware interlocks, operand forwarding and delayed load

8. In ____________ the control selects the target instruction if the condition is satisfied or the next sequential instruction if the condition is not satisfied.
A. Conditional branch
B. Unconditional branch
C. Semiconditional branch
D. None of the above

9. An ____________ always alters the sequential program flow by loading the program counter with the target address.
A. Conditional branch
B. Unconditional branch
C. Uniconditional branch
D. None of the above

10. Which of the following is a way for dealing with branching of instructions?
A. Loop buffer
B. Branch prediction
C. Delayed branch
D. All loop buffer, branch prediction and delayed branch

11. Why it is preferred to do parallel processing?
A. For increasing the computer processing capabilities
B. For increasing the throughput
C. For both increasing the throughput and processing capabilities
D. None of the above

12. Parallel processing can occur in
A. Data stream
B. Instruction stream
C. Both data and instruction stream
D. None of the above

13. In which group, all processors receive the same instruction from the control unit but operate on different items of data?
A. SISD
B. SIMD
C. MISD
D. MIMD

14. Of which group, as such there is no practical application. It is studied due to the theoretical interest?

A. SISD
B. SIMD
C. MISD
D. MIMD

15. In which group, the computer systems are capable of processing several programs at the same time?
A. SISD
B. SIMD
C. MISD
D. MIMD

Answers for Self Assessment

1.	C	2.	A	3.	B	4.	A	5.	B
6.	A	7.	D	8.	A	9.	B	10.	D
11.	C	12.	C	13.	B	14.	C	15.	D

Review Questions:

1. What is parallel processing? Explain its purpose and levels of complexity.
2. What is the classification of parallel processing?
3. What is pipeline and its flow of information? Write about its applicability.
4. Explain the instruction pipeline.
5. Explain pipeline hazards.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education, Asia, 2002.

Web Links

https://www.geeksforgeeks.org/computer-organization-and-architecture-pipelining-set-1-execution-stages-and-throughput/

Unit 12: Memory Technology

CONTENTS

Objectives

After studying this unit, you will be able to

- understand what is memory
- understand the different types of memory
- understand the associative memory
- understand the cache memory
- understand the concept of virtual memory
- understand the memory management unit

Introduction

The memory unit is an essential component in any digital computer since it is needed for storing programs and data. It is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by CPU. The memory unit that communicates directly with the CPU is called the main memory. Devices that provide backup storage are called auxiliary memory.Only programs and data currently needed by the processor reside in main memory. All other information is stored in auxiliary memory and transferred to main memory when needed.

12.1 Memory Hierarchy

The memory hierarchy system consists of all storage devices employed in a computer system:

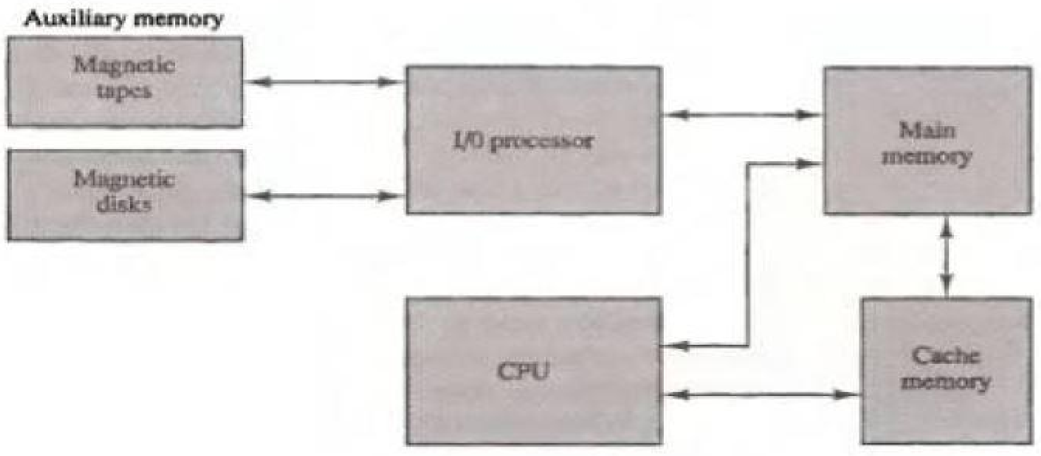
1) Slow but high-capacity auxiliary memory,

2) Relatively faster main memory,

3) Smaller and faster cache memory.

Components in Memory Hierarchy

At the bottom of the hierarchy, are the relatively slow magnetic tapes used to store removable files.Next there are the magnetic disks used as backup storage.The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an IOprocessor. When programs not residing in main memory are needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space for currently used programs and data.A special very-high-speed memory called a cacheis sometimes used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.



12.2 Use of cache memory

The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic. CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory. A technique used to compensate for the mismatch in operating speed is to employ an extremely fast, small cache between the CPU and main memory whose access time is close to processor logic clock cycle time.The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations.

Levels in Memory Hierarchy

The reason for having two or three levels of memory hierarchy is economics. As the storage capacity of the memory increases, the cost per bit for storing binary information decreases and the access time of the memory becomes longer. The auxiliary memory has a large storage capacity, is relatively inexpensive, but has low access speed compared to main memory. The cache memory is very small, relatively expensive, and has very high access speed. Thus as the memory access speed increases, so does its relative cost. The overall goal of using a memory hierarchy is to obtain the highest-possible average access speed while minimizing the total cost of the entire memory system.

Multiprogramming

Many operating systems are designed to enable the CPU to process a number of independent programs concurrently. This concept is called multiprogramming. It refers to the existence of two or more programs in different partsof the memory hierarchy at the same time. In this way it is possible to keep all parts of the computer busy by working with several programs in sequence. For example, suppose that a program is being executed in the CPU and an I/Otransfer is required. The CPU initiates the VO processor to start executing the transfer. This leaves the CPU free to execute another program. In a multiprogramming system, when one program is waiting for input or output transfer, there is another program ready to utilize the CPU.

12.3 Main Memory

The main memory is the central storage unit in a computer system. It is a relatively large and fast memory used to store programs and data during the computer operation. The principal technology used for the main memory is based on semiconductor integrated circuits.

RAM

Integrated circuit RAM chips are available in two possible operating modes:

A) Static

B) Dynamic.

Static RAM: The static RAM consists essentially of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit. The static RAM is easier to use and has shorter read and write cycles.

Dynamic RAM: The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip.

ROM

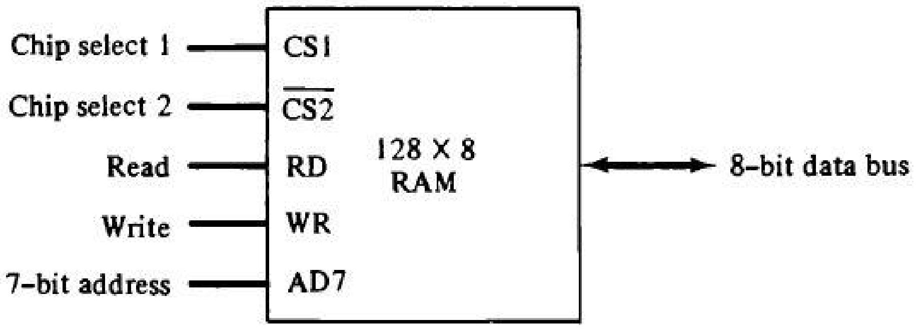
ROM is used for storing programs that are permanently resident in the computer and for tables of constants that do not change in value once the production of the computer is completed.The ROM portion of main memory is needed for storing an initial program called a bootstrap loader.The bootstrap loader is a program whose function is to start the computer software operating when power is turned on.

Since RAM is volatile, its contents are destroyed when power is turned off. But the contents of ROM remain unchanged after power is turned off and on again. The startup of a computer consists of turning the power on and starting the execution of an initial program. Thus when power is turned on, the hardware of the computer sets the program counter to the first address of the bootstrap loader. The bootstrap program loads a portion of the operating system from disk to main memory and control is then transferred to the operating system, which prepares the computer for general use.

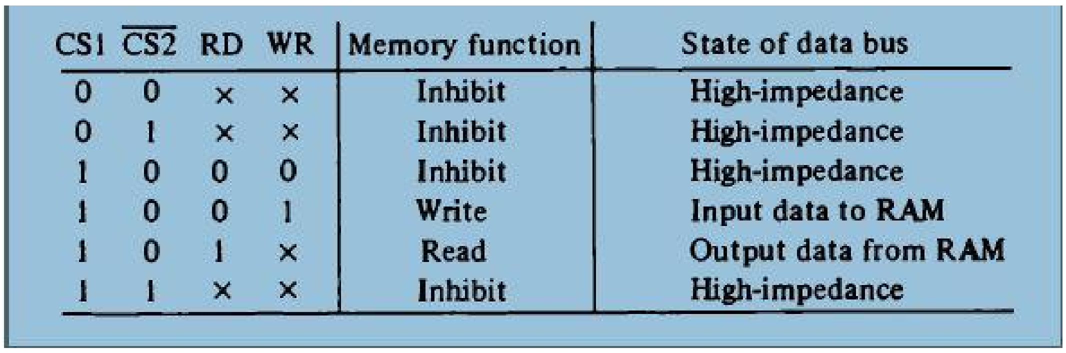
12.4 RAM and ROM chips

RAM chips

A RAM chip is better suited for communication with the CPU if it has one or more control inputs that select the chip only when needed. Another common feature is a bidirectional data bus that allows the transfer of data either from memory to CPU during a read operation or from CPU to memory during a write operation. A bidirectional bus can be constructed with three-state buffers. A three-state buffer output can be placed in one of three possible states: a signal equivalent to logic 1, a signal equivalent to logic 0, or a high impedance state. The logic 1 and O are normal digital signals. The high impedance state behaves like an open circuit, which means that the output does not carry a signal and has no logic significance.



The capacity of the memory is 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8-bit bidirectional data bus. The read and write inputs specifies, the memory operation and the two chips select (CS) control inputs are fee enabling the chip only when it is selected by the microprocessor. The availability of more than one control input to select the chip facilitates the decoding ofthe address lines when multiple chips are used in the microcomputer. The read and write inputs are sometimes combined into one line labeled R/W. When the chip is selected, the two binary states in this line specify the two operations of read or write.

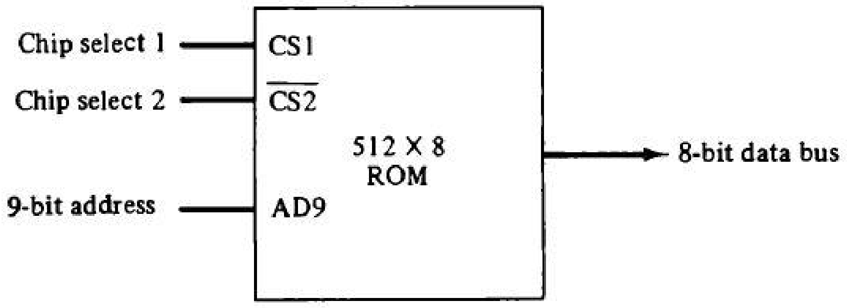


CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	×	×	Inhibit	High-impedance
0	1	×	×	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	×	Read	Output data from RAM
1	1	×	×	Inhibit	High-impedance

The unit is in operation only when CSl = 1 and CS2′= 0. The bar on top ofthesecondselectvariableindicatesthatthisinputisenabledwhenitis equal to0.Ifthechipselectinputsarenotenabled, oriftheyareenabled but theread or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedance state. When CSl = 1 and CS2′= 0, the memory can be placed in a write or read mode. When the WR input is enabled,the memorystores a byte from the data bus into a location specified by the address input lines. When the RD input is enabled, the content of the selected byte is placed intothedatabus.TheRDandWRsignalscontrolthememoryoperationaswell as the bus buffers associated with the bidirectional databus.

ROM chips

A ROM chip is organized externally in a similar manner. However, since a ROM can only read, the data bus can only be in an output mode.



For the same-size chip, it is possible to have more bits of ROM than of RAM, because the internal binary cells in ROM occupy less space than in RAM. For this reason, the diagram specifies a 512-byte ROM, while the RAM has only 128 bytes. The nine address lines in the ROM chip specify any one of the 512 bytes stored in it. The two chip select inputs must be CSl = 1 and CS2= O for the unit to operate. Otherwise, the data bus is in a high-impedance state. There is no need for a read or write control because the unit can only read. Thus when the chip is enabled by the two select inputs, the byte selected by the address lines appears on the data bus.

12.5 Memory Address Map

The designer of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM. The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available. The addressing of memory can be established by means of a table that specifies the memory address assigned to each chip. The table, called a memory address map, is a pictorial representation of assigned address space for each chip in the system.

Memory Address Map for Microcomputer

Assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM. The component column specifies whether a RAM or a ROM chip is used. The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip. The address bus lines are listed in the third column. The small x's under the address bus lines designate those lines that must be connected to the address inputs in each chip. The RAM chips have 128 bytes and need seven address lines. The ROM chip has 512 bytes and needs 9 address lines. The x's are always assigned to the low-order bus lines: lines 1 through 7 for the RAM and lines 1 through 9 for the ROM. It is now necessary to distinguish between four RAM chips by assigning to each a different address.

Component	Hexadecimal address	Address bus									
		10	9	8	7	6	5	4	3	2	1
RAM 1	0000–007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080–00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100–017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180–01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200–03FF	1	x	x	x	x	x	x	x	x	x

For this particular example we choose bus lines 8and 9 to represent four distinct binary combinations. Note that any other pair of unused bus lines can be chosen for this purpose. The table clearly shows that the nine low-order bus lines constitute a memory space for RAM equal to 2^9 = 512 bytes. The distinction between a RAM and ROM address is done with another bus line. Here we choose line 10 for this purpose. When line 10 is O, the CPU selects a RAM, and when this line is equal to 1, it selects the ROM. The equivalent hexadecimal address for each chip is obtained from the information under the address bus assignment. The address bus linesare subdivided into groups of four bits each so that each group can be represented with a hexadecimal digit. The first hexadecimal digit represents lines 13 to 16 and is always 0. The next hexadecimal digit represents lines 9 to 12, but lines 11 and 12 are always 0. The range of hexadecimal addresses for each component is determined from the x's associated with it. These x's represent a binary number that can range from an all-O's to an all-l's value.

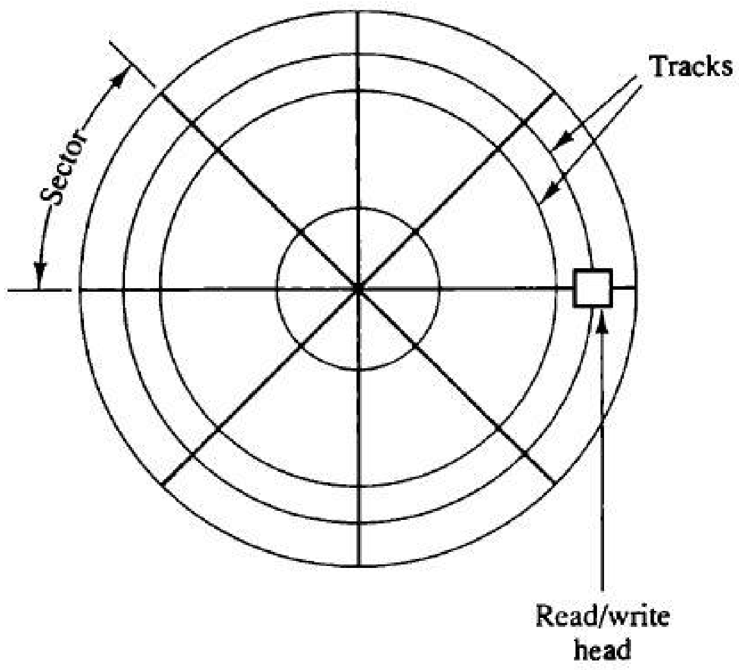
12.6 Auxiliary memory

The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. Other components used, but not as frequently, are magnetic drums, magnetic bubble memory, and optical disks. To understand fully the physical mechanism of auxiliary memory devices one must have knowledge of magnetic, electronics, and electromechanical systems. Although the physical properties of these storage devices can be quite complex, their logical properties can be characterized and compared by a few parameters. The important characteristics of any device are its access mode, access time, transfer rate, capacity, and cost.

The average time required to reach a storage location in memory and obtain its contents is called the access time. In electromechanical devices with moving parts such as disks and tapes, the access time consists of a seek time required to position the read-write head to a location and a transfer time required to transfer data to or from the device.Because the seek time is usually much longer than the transfer time, auxiliary storage is organized in records or blocks. A record is a specified number of characters or words. Reading or writing is always done on entire records. The transfer rate is the number of characters or words that the device can transfer per second, after it has been positioned at the beginning of the record.

Magnetic drums and disks are quite similar in operation. Both consist of high-speed rotating surfaces coated with a magnetic recording medium. The rotating surface of the drum is a cylinder and that of the disk, around flat plate. The recording surface rotates at uniform speed and is not started or stopped during access operations. Bits are recorded as magnetic spots on the surface as it passes a stationary mechanism called a write head. Stored bits are detected by a change in magnetic field produced by a recorded spot on the surface as it passes through a read head. The amount of surface available for recording in a disk is greater than in a drum of equal physical size.Therefore, more information can be stored on a disk than on a drum of comparable size. For this reason, disks have replaced drums in more recent computers.

Magnetic Disks



A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of the disk are used and several disks may be stacked on one spindle with read/ write heads available on each surface. All disks rotate together at high speed and are not stopped or started for access purposes. Bits are stored in the magnetized surface in spots along concentric circles called tracks. The tracks are commonly divided into sections called sectors. In most systems, the minimum quantity of information which can be transferred is a sector.

Some units use a single read/write head for each disk surface. In this type of unit, the track address bits are used by a mechanical assembly to move the head into the specified track position before reading or writing. In other disk systems, separate read/write heads are provided for each track in each surface. The address bits can then select a particular track electronically through a decoder circuit. This type of unit is more expensive and is found only in very large computer systems.

Permanent timing tracks are used in disks to synchronize the bits and recognize the sectors. A disk system is addressed by address bits that specify the disk number, the disk surface, the sector number and the track within the sector. After the read/write heads are positioned in the specified track, the system has to wait until the rotating disk reaches the specified sector under the read/ write head. Information transfer is very fast once the beginning of a sector has been reached. Disks may have multiple heads and simultaneous transfer of bits from several tracks at the same time.

A track in a given sector near the circumference is longer than a track near the center of the disk.If bits are recorded with equal density, some tracks will contain more recorded bits than others. To make all the records in a sector of equal length, some disks use a variable recording density with higher density on tracks near the center than on tracks near the circumference. This equalizes the number of bits on all tracks of a given sector.Disks that are permanently attached to the unit assembly and cannot be removed by the occasional user are called hard disks.A disk drive with removable disks is called a floppy disk. The disks used with a floppy disk drive are small removable disks made of plastic coated with magnetic recording material. There are two sizes commonly used, with diameters of 5.25 and 3.5 inches.The 3.5-inch disks are smaller and can store more data than can the 5.25-inch disks. Floppy disks are extensively used in personal computers as a medium for distributing software to computer users.

Magnetic Tape

A magnetic tape transport consists of the electrical, mechanical, and electronic components to provide the parts and control mechanism for a magnetic-tape unit. The tape itself is a strip of plastic coated with a magnetic recording medium. Bits are recorded as magnetic spots on the tape along several tracks. Usually, seven or nine bits are recorded simultaneously to form a character together with a parity bit.Read/ write heads are mounted one in each track so that data can be recorded and read as a sequence of characters.

Magnetic tape units can be stopped, started to move forward or in re verse, or can be rewound. However, they cannot be started or stopped fast enough between individual characters.For this reason, information is recorded in blocks referred to as records. Gaps of unrecorded tape are inserted between records where the tape can be stopped.The tape starts moving while in a gap and attains its constant speed by the time it reaches the next record.Each record on tape has an identification bit pattern at the beginning and end.By reading the bit pattern at the beginning, the tape control identifies the record number. By reading the bit pattern at the end of the record, the control recognizes the beginning of a gap.A tape unit is addressed by specifying the record number and the number of characters in the record.Records may be of fixed or variable length.

Many data-processing applications require the search of items in a table stored in memory. Examples:

A) An assembler program.

B) An account number.

The search procedure is a strategy for choosing a sequence of addresses.The number of accesses to memory depends on the location of the item and the efficiency of the search algorithm.

12.7 Associative Memory

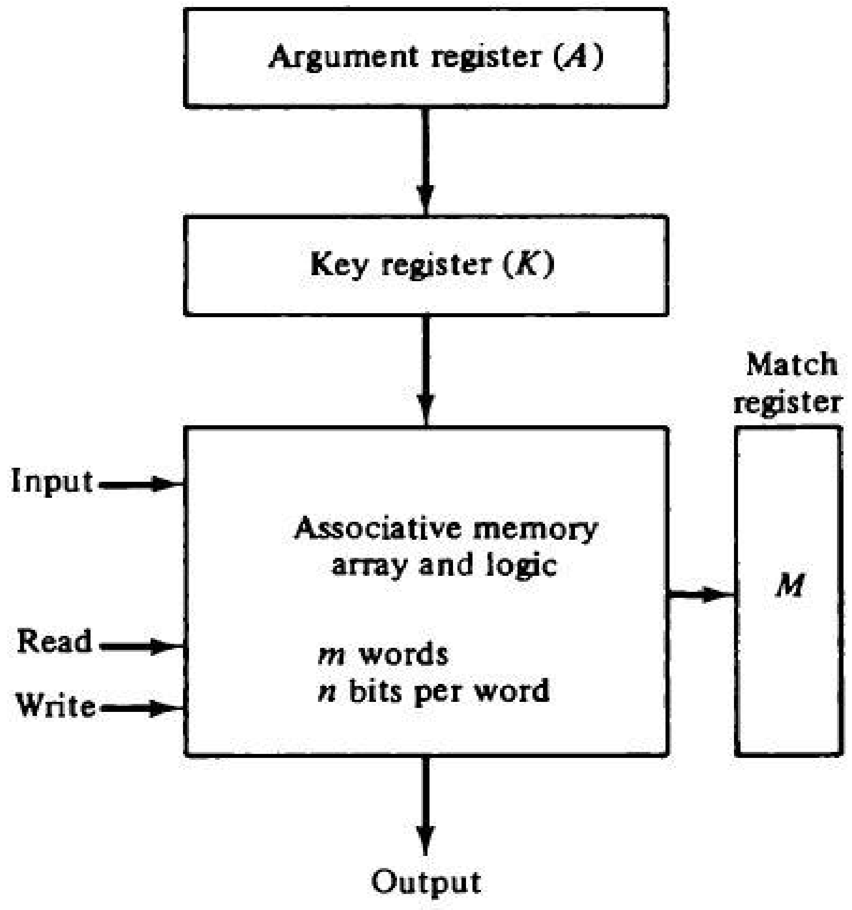
A memory unit accessed by content is called an associative memory or content addressable memory (CAM).

Characteristics of Associative Memory

Memory is accessed simultaneously and in parallel on the basis of data content.No address is given when writing. It works by finding an empty unused location.The content of the word, or part of the word, is specified. The memory locates all words which match the specified content and mark them for reading.The associative memory is expensive because of storage capabilities and logic circuits for matching.

Applications

It performs parallel searches by data association. It is applicable where the search time is very critical and must be very short.



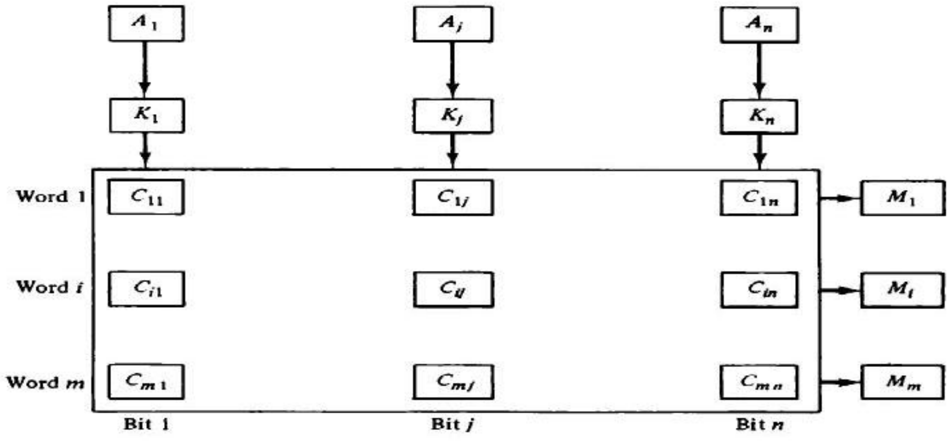
It consists of a memory array and logic for m words with n bits per word. The argument register A and key register K each have n bits, one for each bit of a word. The match register M has m bits, one for each memory word.

Matching process:

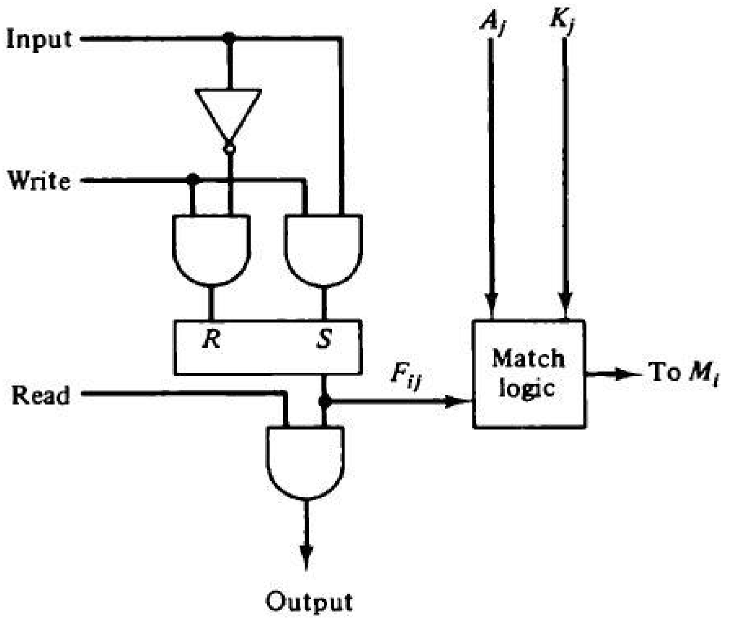
1) Each word in memory is compared in parallel with the content of the argument register.
2) The words that match the bits of the argument register set a corresponding bit in the match register.
3) After the matching process, those bits in the match register that have been set indicate the fact that their corresponding words have been matched.

Example:

A	101 111100	
K	111 000000	
Word 1	100 111100	**No match**
Word 2	101 000001	**Match**



This shows the relationship between the memory array and external registers in an associative memory.This is how the cell is internally organized.



The match logic for each word can be derived from the comparison algorithm for two binary numbers. First, we neglect the key bits and compare the argument in A with the bits stored in the cells of the words. Word i is equal to the argument in A if $A_j = F_{ij}$ for j = 1, 2,…n. Two bits are equal if they are both1 or both 0. The equality of two bits can be expressed logically by the Boolean function

$$x_j = A_j F_{ij} + A_j' F_{ij}'$$

Where $x_j = 1$ if the pair of bits in position j are equal; otherwise, $x_j = 0$.

12.8 Cache Memory:

It is placed between the CPU and main memory.The cache memory access time is less than the access time of main memory by a factor of 5 to 10.It is the fastest component in the memory hierarchy and approaches the speed of CPU components. The fundamental idea is to keep the most frequently accessed instructions and data, the average memory access time will approach the access time of the cache. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word. A block of words containing the one just accessed is then transferred from main memory to cache memory.The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio.The high ratio verifies the validity of the locality of reference property.The average memory access time of a computer system can be improved considerably by use of a cache.

12.9 Mapping Process

The basic characteristic of cache memory is its fast access time. Therefore, very little or no time must be wasted when searching for words in the cache. The transformation of data from main memory to cache memory is referred to as a mapping process.

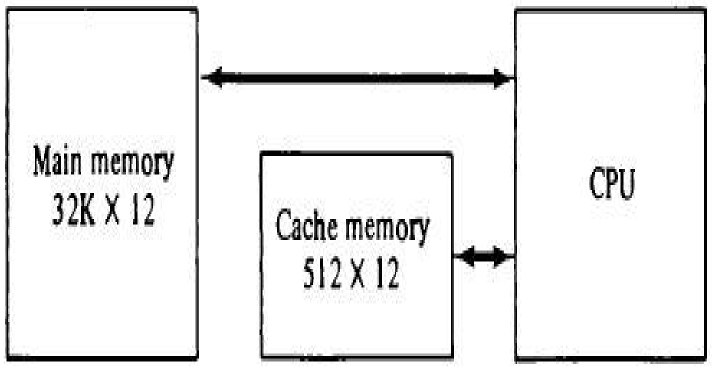
Types of mapping

There are three types of mapping procedures:

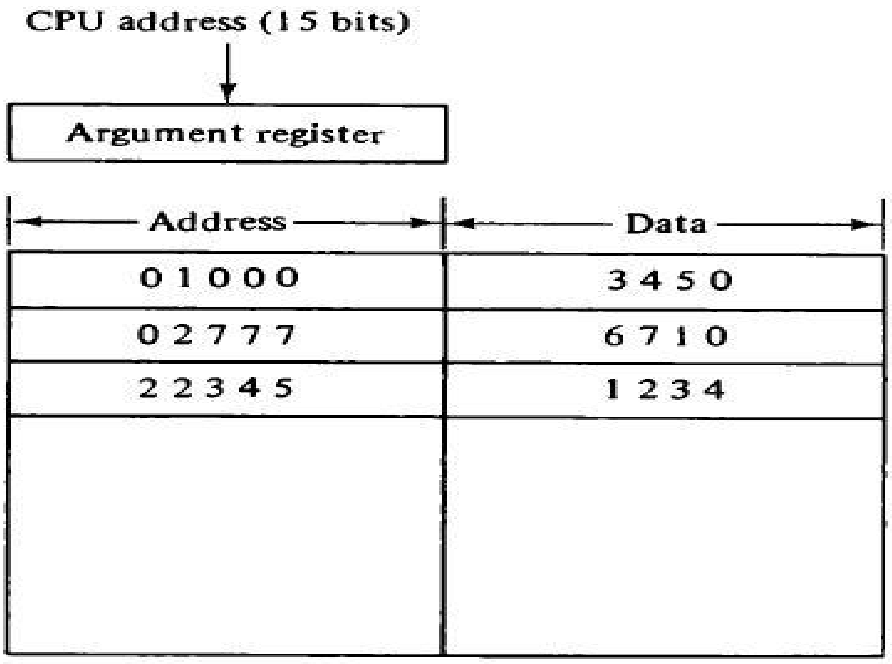
1) Associative mapping

2) Direct mapping

3) Set-associative mapping



Associative Mapping:

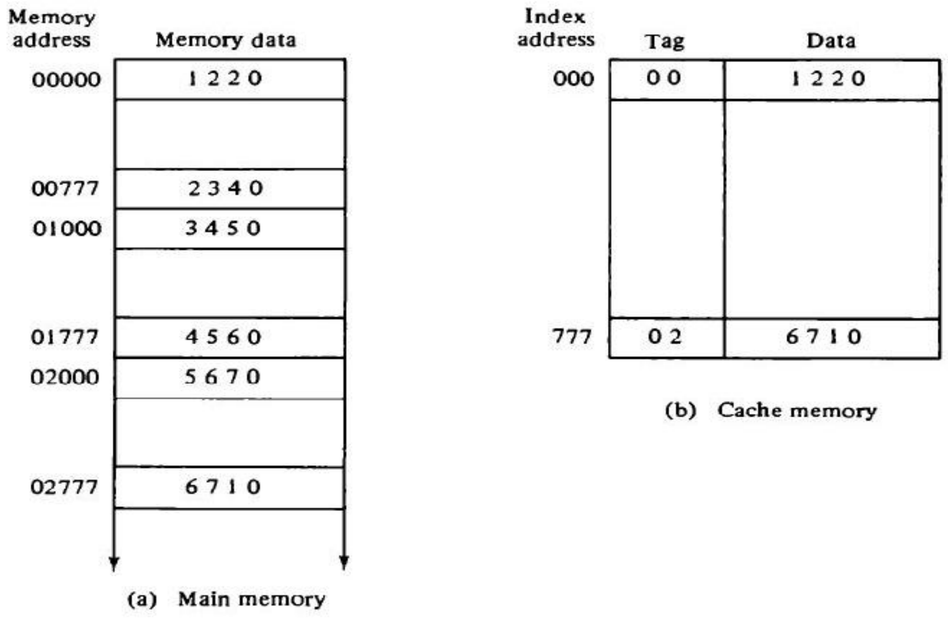


Address	Data
0 1 0 0 0	3 4 5 0
0 2 7 7 7	6 7 1 0
2 2 3 4 5	1 2 3 4

Here three words presently stored in the cache. The address value of 15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number. A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address. If the address is found, the corresponding 12-bit data is read and sent to the CPU. If no match occurs, the main memory is accessed for the word. The address-data pair is then transferred to the associative cache memory. For replacement it uses round-robin order.

Direct Mapping

In this, the CPU address of 15 bits is divided into two fields: 9 bits: index field and 6 bits: tag field.The main memory needs an address that includes both the tag and the index bits.



In the general case, there are 2^k words in cache memory and 2^n words in main memory. The direct mapping cache organization uses the n-bit address to access the main memory and the k-bit index to access the cache.

Disadvantage of direct mapping:

Direct mapping is that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly.

Set Associative Mapping

It is an improvement over the direct mapping organization in that each word of cache can store two or more words of memory under the same index address. Each data word is stored together with its tag and the number of tag-data items in one word of cache is said to form a set. Each index address refers to two data words and their associated tags. Each tag requires 6 bits and each data word has 12 bits, so the word length is 2(6 + 12) = 36 bits. An index address of 9 bits can accommodate 512 words. Thus the size of cache memory is 512x36. It can accommodate 1024 words of main memory since each word of cache contains two data words.In general, a set-associative cache of set size k will accommodate k words of main memory in each word of cache.When the CPU generates a memory request, the index value of the address is used to access the cache. The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs. The comparison logic is done by an associative search of the tags in the set similar to an associative memory search: thus the name "set-associative.

Index	Tag	Data	Tag	Data
000	01	3450	02	5670
777	02	6710	00	2340

Replacement algorithms

When a miss occurs in a set-associative cache and the set is full, it is necessary to replace one of the tag-data items with a new value. The most common replacement algorithms used are:

1) Random replacement

2) First-in-first out (FIFO)

3) Least recently used (LRU)

Both FIFO and LRU can be implemented by adding a few extra bits in each word of cache.

Writing into Cache

An important aspect of cache organization is concerned with memory write requests. When the CPU finds a word in cache during a read operation, the main memory is not involved in the transfer. However, if the operation is a write, there are two ways that the system can precede:

1) Write through method

2) Write back method

Cache Initialization

The cache is initialized when power is applied to the computer or when the main memory is loaded with a complete set of programs from auxiliary memory. After initialization the cache is considered to be empty, but in effect it contains some non valid data. It have an inclusion of valid bit.The cache is initialized by clearing all the valid bits to 0. The valid bit of a particular cache word is set to 1 the first time this word is loaded from main memory and stays set unless the cache has to be initialized again. The introduction of the valid bit means that a word in cache is not replaced by another word unless the valid bit is set to 1 and a mismatch of tags occurs. If the valid bit happens to be 0, the new word automatically replaces the invalid data.Thus the initialization condition has the effect of forcing misses from the cache until it fills with valid data.

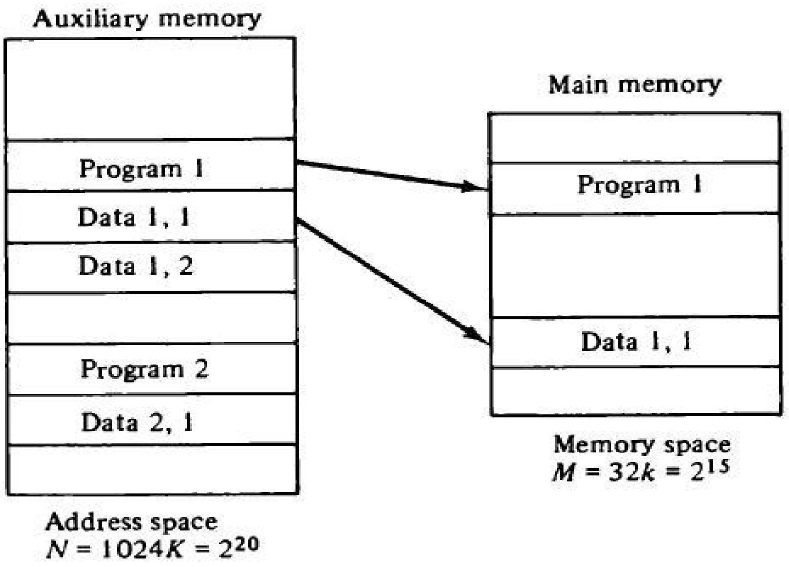
12.10 Virtual Memory

Virtual memory is a concept used in some large computer systems that permit the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory.Each address that is referenced by the CPU goes through an address mapping from the so-called virtual address to a physical address in main memory. A virtual memory system provides a mechanism for translating program generated addresses into correct main memory locations. The translation or mapping is handled automatically by the hardware by means of a mapping table.

Address Space and Memory Space

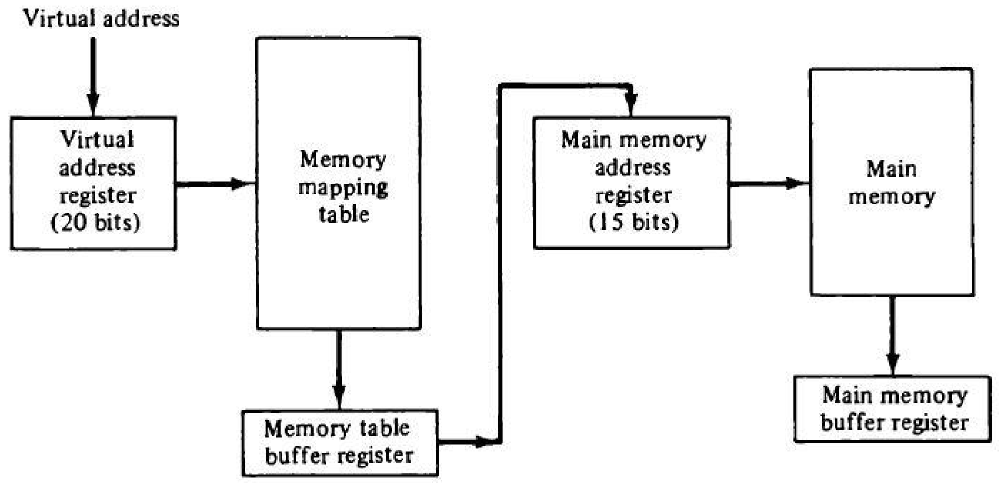
An address used by a programmer will be called a virtual address, and the set of such addresses the address space. An address in main memory is called a location or physical address. The set of such locations is called the memory space. In a multi-program computer system, programs and data are transferred to and from auxiliary memory and main memory based on demands imposed by the CPU. Suppose that program 1 is currently being executed in the CPU. Program 1 and a portion of its associated data are moved from auxiliary memory into main memory.

In a virtual memory system, programmers are told that they have the total address space at their disposal. Moreover, the address field of the instruction code has a sufficient number of bits to specify all virtual addresses. For efficient transfers, auxiliary storage moves an entire record to the main memory. A table is then needed, to map a virtual address of 20 bits to a physical address of 15 bits. The mapping is a dynamic operation, which means that every address is translated immediately as a word is referenced by CPU.



Memory Table for Mapping a Virtual Address

The mapping table may be stored in a separate memory or in main memory:In the first case, an additional memory unit is required as well as one extra memory access time and in the second case, the table takes space from main memory and two accesses to memory are required with the program running at half speed.

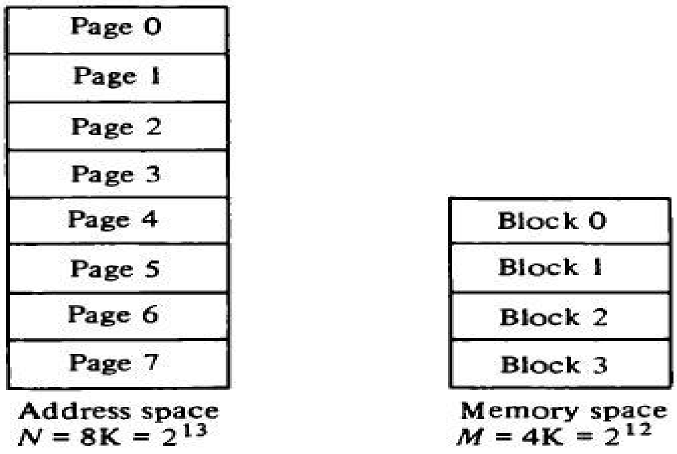

Virtual address
Virtual address register (20 bits)
Memory mapping table
Memory table buffer register
Main memory address register (15 bits)
Main memory
Main memory buffer register

Address Mapping Using Pages

The physical memory is broken down into groups of equal size called blocks, which may range from 64 to 4096 words each.The term page refers to groups of address space of the same size. For example, if a page or block consists of 1K words, then, using the previous example, address space is divided into 1024 pages and main memory is divided into 32 blocks. Although both a page and a block are split into groups of 1K words, a page refers to the organization of address space, while a block refers to the organization of memory space.The programs are also considered to be split into pages. Portions of programs are moved from auxiliary memory to main memory in records equal to the size of a page. The term "page frame" is sometimes used to denote a block.

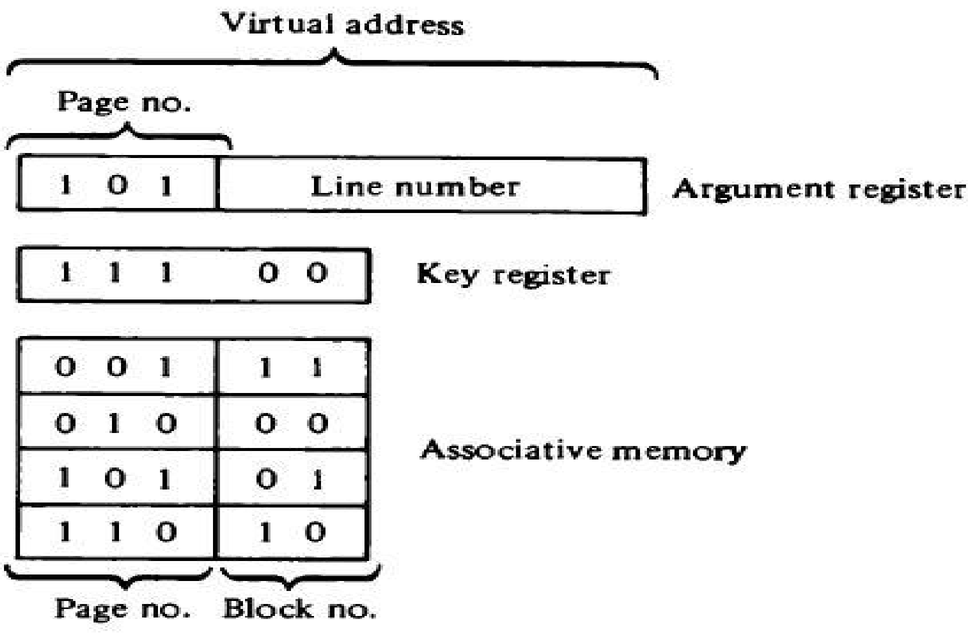
Example: Consider a computer with an address space of 8K and a memory space of 4K.

If we split each into groups of 1K words we obtain eight pages and four blocks. At any given time, up to four pages of address space may reside in main memory in any one of the four blocks.



Associative Memory Page Table

A random-access memory page table is inefficient with respect to storage utilization.In general, a system with n pages and m blocks would require a memory-page table of n locations of which up to m blocks will be marked with block numbers and all others will be empty. As a numerical example, consider an address space of 1024K words and memory space of 32K words.If each page or block contains 1K words, the number of pages is 1024 and the number of blocks 32.The capacity of the memory-page table must be 1024 words and only 32 locations may have a presence bit equal to 1.At any given time, at least 992 locations will be empty and not in use.A more efficient way to organize the page table would be to construct it with a number of words equal to the number of blocks in main memory.In this way the size of the memory is reduced and each location is fully utilized. This method can be implemented by means of an associative memory with each word in memory containing a page number together with its corresponding block number. The page field in each word is compared with the page number in the virtual address. If a match occurs, the word is read from memory and its corresponding block number is extracted.



Each entry in associative memory array consists of two fields.First three bits specify a field for storing the page number. Last two fields constitute a field for storing the block number.Virtual address is placed in the argument register.

Page Replacement

A virtual memory system is a combination of hardware and software techniques. When a program starts execution, one or more pages are transferred into main memory and the page table is set to indicate their position. The program is executed from main memory until it attempts to reference a page that is still in auxiliary memory. This condition is called page fault.When a page fault occurs in a virtual memory system, it signifies that the page referenced by the CPU is not in main memory.A new page is then transferred from auxiliary memory to main memory. The policy for choosing pages to remove is determined from the replacement algorithm that is used.Two of the most common replacement algorithms used are the FIFO and LRU.

12.11 Memory Management Hardware

The demands on computer memory brought about by multiprogramming have created the need for a memory management system.A memory management system is a collection of hardware and software procedures for managing the various programs residing in memory.

Basic Components of a Memory Management Unit

The basic components of a memory management unit are:

- A facility for dynamic storage relocation that maps logical memory references into physical memory addresses.

- A provision for sharing common programs stored in memory by different users.
- Protection of information against unauthorized access between users and preventing users from changing operating system functions.

A facility for dynamic storage relocation

The dynamic storage relocation hardware is a mapping process similar to the paging system.The fixed page size used in the virtual memory system causes certain difficulties with respect to program size and the logical structure of programs. It is more convenient to divide programs and data into logical parts called segments. Examples of segments are a subroutine, an array of data, a table of symbols, or a user's program.The sharing of common programs is an integral part of a multi-programming system.

Protection of information against unauthorized access

The issue in multiprogramming is protecting one program from unwanted interaction with another. The secrecy of certain programs must be kept from unauthorized personnel to prevent abuses in the confidential activities of an organization.

12.12 Logical address

The address generated by a segmented program is called a logical address. This is similar to a virtual address except that logical address space is associated with variable-length segments rather than fixed-length pages. In addition to relocation information, each segment has protection information associated with it. Shared programs are placed in a unique segment in each user's logical address space so that a single physical copy can be shared. The function of the memory management unit is to map logical addresses into physical addresses similar to the virtual memory mapping concept.

Segmented-Page Mapping

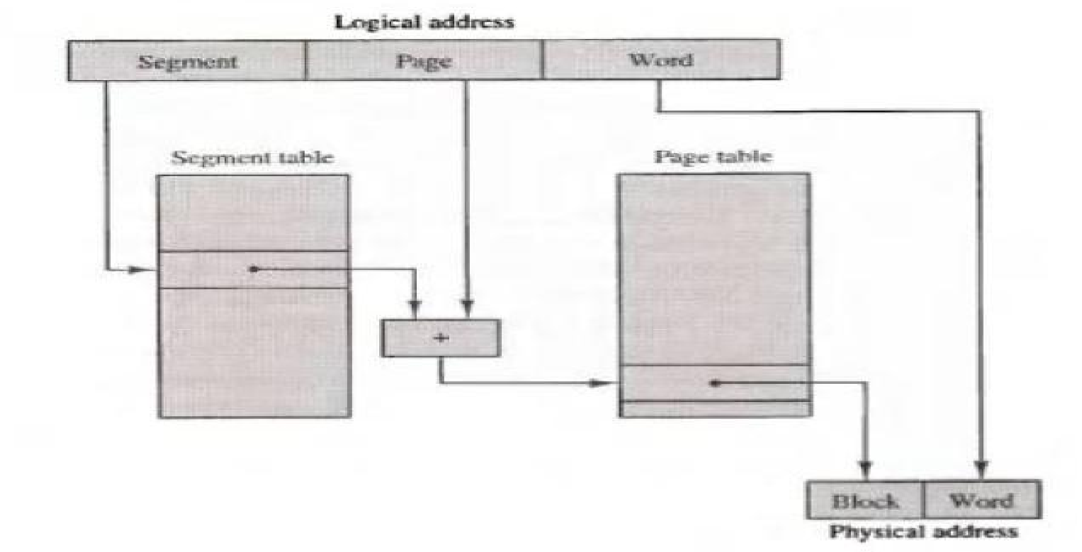
The property of logical space is that it uses variable-length segments. The length of each segment is allowed to grow and contract according to the needs of the program being executed. One way of specifying the length of a segment is by associating it with a number of equal-size pages. The logical address is partitioned into three fields:

A) The segment field specifies a segment number.

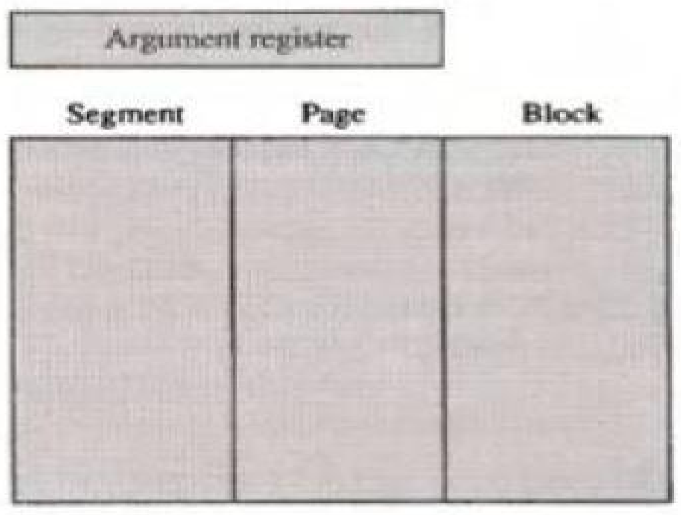
B) The page field specifies the page within the segment.

C) The word field gives the specific word within the page. A page field of k bits can specify up to 2^k pages.

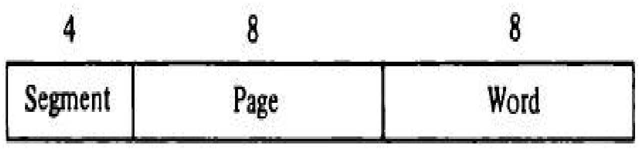
Logical to physical address mapping



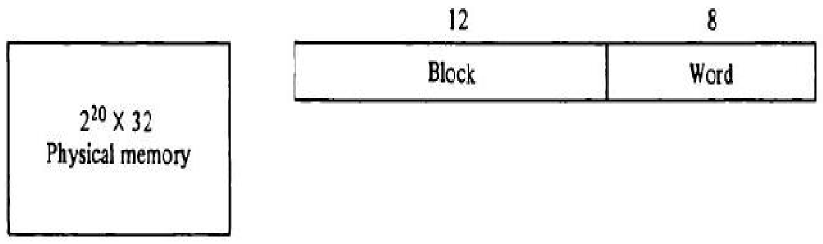
TLB



Example: Consider the 20-bit logical address.



(a) Logical address format: 16 segments of 256 pages each, each page has 256 words



(b) Physical address format: 4096 blocks of 256 words each, each word has 32 bits

Summary

- The memory hierarchy system consists of all storage devices employed in **a** computer system:auxiliary memory, main memory and cache memory.
- A cacheis sometimes used to make current programs and data available to the CPU at a rapid rate.
- The overall goal is to obtain the highest-possible average access speed while minimizing the total cost of the entire memory system.
- RAM is volatile and ROM is non-volatile in nature.
- Some examples of auxiliary memory aremagnetic disks, tapes, magnetic drums, magnetic bubble memory, and optical disks.
- Cache memory is placed between the CPU and main memory.
- The most common replacement algorithms used are: random replacement, first-in-first out (FIFO) and least recently used (LRU).

Keywords

Access time: The average time required to reach a storage location in memory and obtain its contents is called the access time.

Seek time: In electromechanical devices with moving parts such as disks and tapes, the access time consists of a seek time required to position the read-write head to a location and a transfer time required to transfer data to or from the device.

Transfer rate: It is the number of characters or words that the device can transfer per second, after it has been positioned at the beginning of the record.

Write head: Bits are recorded as magnetic spots on the surface as it passes a stationary mechanism called a write head.

Read head: Stored bits are detected by a change in magnetic field produced by a recorded spot on the surface as it passes through a read head.

Associative memory: A memory unit accessed by content is called an associative memory or content addressable memory (CAM).

Hit ratio:The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio.

Self Assessment

1. Memory space belongs to

A. Auxiliary memory

B. Main memory

C. Cache memory

D. None of the above

2. Address space belongs to

A. Auxiliary memory

B. Main memory

C. Cache memory

D. None of the above

3. Which of the following word is used to denote a block?

A. Page frame

B. Word frame

C. Line frame

D. Sentence frame

4. Which of the following is page replacement algorithm?

A. LRU

B. FIFO

C. Both LRU and FIFO

D. None of the above

5. Which memory is placed between CPU and main memory?

A. Cache memory

B. Associative memory

C. Random memory

D. None of the above

6. In which memory, no address is specified when writing?

A. Cache memory

B. Associative memory

C. Random memory

D. None of the above

7. Which memory requires the logic circuit for matching the content?

A. Cache memory

B. Associative memory

C. Random memory

D. Main memory

8. Which of the following is fastest kind of memory?

A. Auxiliary memory

B. Cache memory

C. Main memory

D. None of the above

9. Which of the following is slowest kind of memory?
A. Auxiliary memory
B. Cache memory
C. Main memory
D. None of the above

10. Magnetic tapes are a kind of
A. Auxiliary memory
B. Cache memory
C. Main memory
D. None of the above

11. Which of the following memory does not deal directly with the CPU?
A. Auxiliary memory
B. Cache memory
C. Main memory
D. None of the above

12. Which of the following memory is employed to compensate for speed differential between main memory access time and processor logic?
A. Magnetic tapes
B. Magnetic disks
C. Cache memory
D. None of the above

13. In which of the following memory, the electric charges are applied to the capacitors?
A. Static RAM
B. Dynamic RAM
C. Both static and dynamic RAMs
D. None of the above

14. Which of the following is an important characteristic of any device?
A. Access time
B. Transfer rate
C. Cost
D. All access time, transfer rate and cost

15. Which of the following memory is non volatile in nature?
A. RAM
B. ROM
C. Both ROM and RAM

D. None of the above

Answers for Self Assessment

1.	B	2.	A	3.	A	4.	C	5.	A
6.	B	7.	B	8.	B	9.	A	10.	A
11.	A	12.	C	13.	B	14.	D	15.	B

Review Questions

1. What is memory hierarchy? Explain its components.
2. What are RAM and ROM? Explain its chips also.
3. What is memory address map? Explain for microcomputer.
4. What is auxiliary memory? Explain its types also.
5. What is associative memory? Explain its characteristics and

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education, Asia, 2002.

Web Links

https://www.britannica.com/technology/computer-memory

Unit 13: I/O Subsystems

CONTENTS

Objectives

After studying this unit, you will be able to understand:

- understand the peripheral devices
- understand the I/O interface
- understand the I/O bus and interface modules
- understand the synchronous and asynchronous methods of transfer
- understand the strobe and handshaking process in asynchronous data transfer
- understand the three modes of data transfer

Introduction

The I/O devices Provides an efficient mode of communication between the central system and the outside environment. Programs and data must be entered into computer memory for processing and results obtained from computations must be recorded or displayed for the user. The most familiar means of entering information into a computer is through a typewriter-like keyboard that allows a person to enter alphanumeric information directly.The fastest possible speed for entering information this way depends on the person's typing speed. On the other hand, the CPU is an extremely fast device capable of performing operations at very high speed. To increase the efficiency, the data must be prepared in advance and transmitted into a storage medium such as magnetic tapes or disks. The information in the disk is then transferred into computer memory at a rapid rate.

13.1 Peripheral Devices

The devices that are under the direct control of the computer are said to be connected on-line. These devices are designed to read and write information. Input or output devices attached to the computer are also called peripherals.

Monitor and keyboard

Video monitors are the most commonly used peripherals. They consist of a keyboard as the input device and a display unit as the output device. There are different types of video monitors, but the most popular use a CRT.A characteristic feature of display devices is a cursor. These devices can operate in different modes: single-character mode and block mode.

Printer

Printers provide a permanent record on paper of computer output data or text. There are three basic types of character printers: daisywheel, dot matrix, and laser printers.

Magnetic tapes

These magnetic tapes are used to store files of data. Here the access is sequential and the tape moves along a stationary read-write mechanism. It is one of the cheapest and slowest methods for storage. It can also be removed when not in use.

Magnetic disks

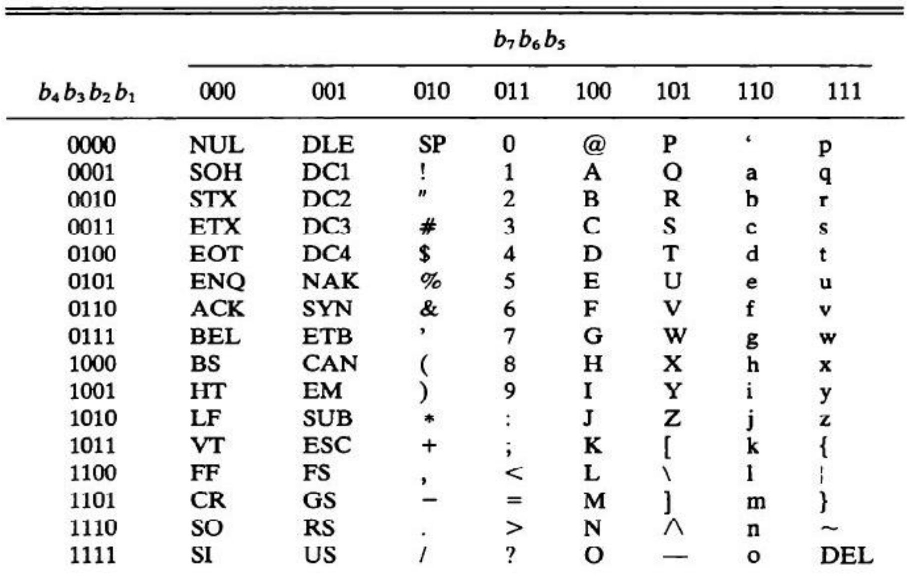
These are high-speed rotational surfaces coated with magnetic material. Here the access is achieved by moving a read-write mechanism to a track in the magnetized surface. It is used for bulk storage of programs and data.

Other input and output devices:

The other input and output devices are digital incremental plotters, optical and magnetic character readers, analog-todigital converters, and various data acquisition equipment. Computers are not just intended for humans but also used to control various processes in real time.The input-output organization of a computer is a function of the size of the computer and the devices connected to it.

13.2 ASCII Alphanumeric Characters

The standard binary code is ASCII. It uses 7 bits to code 128 characters. The bits are designated by b1 through b7,with b, being the most significant bit.The letter A, for example, is represented in ASCII as 100 0001 (column 100, row 0001). The ASCII code contains 94 characters that can be printed and 34 nonprinting characters used for various control functions. The printing characters consist of the 26 uppercase letters A through Z, the 26 lowercase letters, the 10 numerals O through 9, and 32 special printable characters such as %, *, and$. The ASCII characters are:



	$b_7 b_6 b_5$							
$b_4 b_3 b_2 b_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	‘	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	’	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	¦
1101	CR	GS	−	=	M	]	m	}
1110	SO	RS	.	>	N	∧	n	~
1111	SI	US	/	?	O	—	o	DEL

ASCII is a 7-bit code, but most computers manipulate an 8-bit quantity as a single unit called a byte. Therefore, ASCII characters most often are stored one per byte. The extra bit is sometimes used for other purposes, depending on the application.The control characters are used for routing data and arranging the printed text into a prescribed format. There are three types of control characters:

1) Format effectors: Controls the layout of printing.

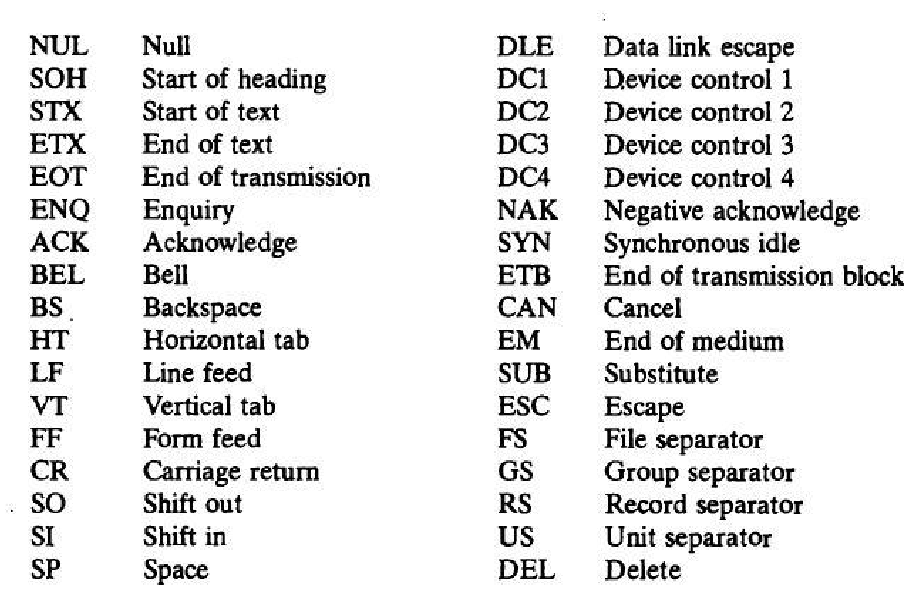
Example: BS, HT and CR.

2) Information separators: Separates the data into divisions like paragraphs and pages.

Example: RS and FS.

3) Communication control characters: Used during the transmission of text between remote terminals.

Example: RETXS and FS.



NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

13.3 Input-Output Interface

This provides a method for transferring information between internal storage and external I/0 devices. Peripherals connected to a computer need special communication links for interfacing them with the CPU to resolve the differences that exist between the central computer and each peripheral. The major differences are:

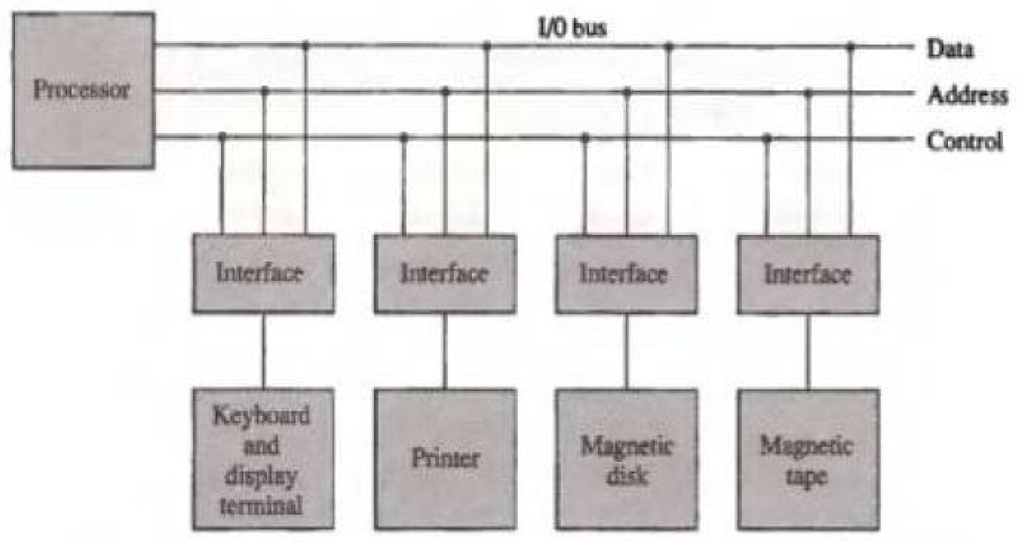
- A conversion of signal values may be required because of the differences in devices and its operation.
- A synchronization mechanism may be needed because the data transfer rate is different.
- Data codes and formats in peripherals differ from the word format in the CPU and memory.
- The operating modes of peripherals are different from each other.

The solution for these differences can also be found:

- To resolve these differences, computer systems include interfaceunits because they interface between the processor bus and the peripheral device.
- Each device may have its own controller that supervises the operations of the particular mechanism in the peripheral.

13.4 I/0 Bus and Interface Modules

It shows a communication link between the processor and several peripherals. The *I/O* bus consists of data lines, address lines, and control lines. The magnetic disk, printer, terminal and magnetic tape are employed.Each peripheral device has associated with it an interface unit. Each peripheral has its own controller that operates the particular electromechanical device. The I/O bus from the processor is attached to all peripheral interfaces. To communicate with a particular device, the processor places a device address on the address lines.When the interface detects its own address, it activates the path between the bus lines and the device that it controls.All peripherals whose address does not correspond to the address in the bus are disabled by their interface.



13.5 Input-Output command

At the same time that the address is made available in the address lines, the processor provides a function code in the control lines.The interface selected responds to the function code and proceeds to execute it. The function code is referred to as an I/O command and is an instruction that is executed in the interface and its attached peripheral unit. The interpretation of the command depends on the peripheral that the processor is addressing.There are four types of commands that an interface may receive. They are classified as control, status, data output, and data input.

Control command

A control command is issued to activate the peripheral and to inform it what to do. For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction.

Status Command

A status command is used to test various status conditions in the interface and the peripheral.For example, the computer may wish to check the status of the peripheral before a transfer is initiated.

Output data

A data output command causes the interface to respond by transferring data from the bus into one of its registers.

Input data

The data input command is the opposite of the data output. In this case the interface receives an item of data from the peripheral and places it in its buffer register.

13.6 I/O versus Memory Bus

In addition to communicating with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address, and read/write control lines. There are three ways that computer buses can be used to communicate with memory and I/O:

1) Use two separate buses, one for memory and the other for I/O.
2) Use one common bus for both memory and I/O but have separate control lines for each.
3) Use one common bus for memory and IO with common control lines.

IOP

The computer has independent sets of data, address, and control buses, one for accessing memory and the other for I/O. The memory communicates with both the CPU and the IOP through a memory bus. The IOP communicates also with the input and output devices through a separate I/0 bus with its own address, data and control lines. It provides an independent pathway for the transfer of information between external devices and internal memory.

Isolated I/O Method

Many computers use one common bus to transfer information between memory or I/0 and the CPU.The distinction between a memory transfer and I/0 transfer is made through separate read and write lines.The CPU specifies whether the address on the address lines is for a memory word or for an interface register by enabling one of two possible read or write lines.The I/0 read and I/0 write control lines are enabled during an I/0 transfer. The memory read and memory write control lines are enabled during a memory transfer. This configuration isolates all I/0 interface addresses from the addresses assigned to memory and is referred to as the isolated I/O method for assigning addresses in a common bus.

Memory-mapped I/O Method

The other alternative is to use the same address space for memory and I/0. This configuration is referred to as memory-mapped I/0. The computer treats an interface register as being part of the memory system. The assigned addresses for interface registers cannot be used for memory words, which reduce the memory address range available.In a memory-mapped I/0 organization there are no specific input or output instructions. Each interface is organized as a set of registers that respond to read and write requests in the normal address space. Computers with memory-mapped I/0 can use memory-type instructions to access I/0 data.The advantage is that the load and store instructions used for reading and writing from memory can be used to input and output data from I/0 registers. Two units, such as a CPU and an I/*O* interface, are designed independently of each other. If they share a common clock, then it is known as synchronous transfer of data. If they share a private clock, then it is known as asynchronous transfer of data.

13.7 Strobe and handshaking

Strobe: A control signal (strobe pulse) for indicating the time at which data is being transmitted. One of the units indicates to the other unit when the transfer has to occur.

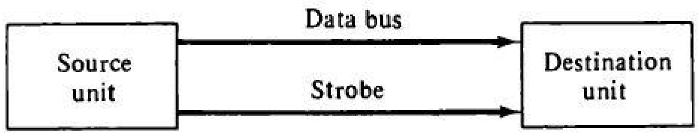
Handshaking: A control signal being transferred accompanying the data item from the sender's side. From the receiver's side, another control signal is sent to acknowledge.

These two methods are not restricted to I/O transfers but they are also used which requires the transfer of data between two independent units.The better way to specify the asynchronous transfer by means of a timing diagram that shows the timing relationship that must exist between the control signals and the data in the buses.

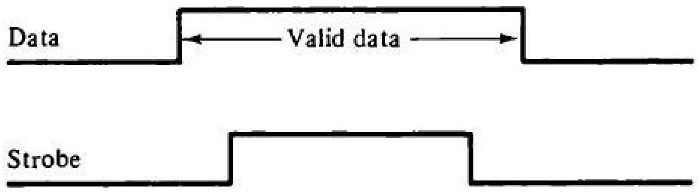
13.8 Strobe Control

This method employs a single control line to time each transfer. The strobe pulse may be activated by either the source or the destination unit. It is a single line that informs the destination unit when a valid data word is available in the bus.

Source Initiated Strobe for Data Transfer

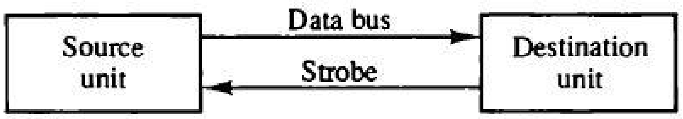


(a) Block diagram

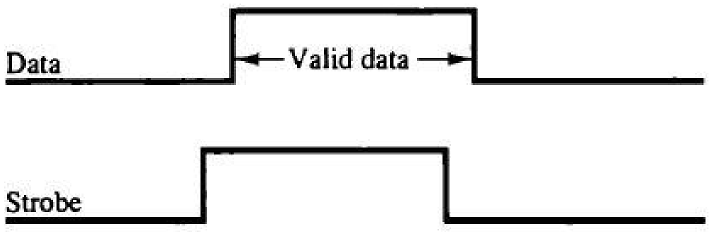


(b) Timing diagram

Destination Initiated Strobe for Data Transfer



(a) Block diagram



(b) Timing diagram

Disadvantages of Strobe Method

There is no way of knowing whether the other unit has actually received/ placed the data. There is no way of acknowledgement.

13.9 Handshaking

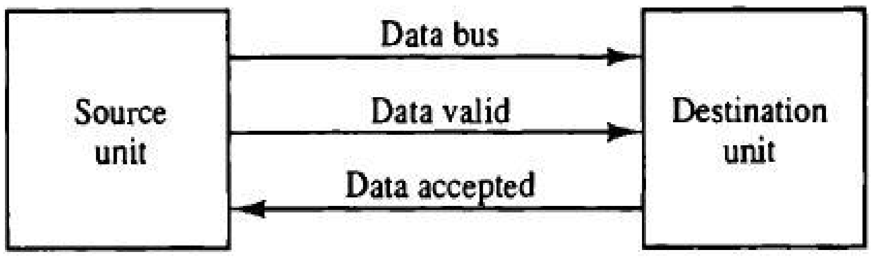
This method gives a second control signal that provides a reply to the unit that initiates the transfer. The pprinciple for this is:1st control line is in the same direction as the data flows and 2nd control line is in the other direction from the destination unit to inform the source whether it can accept data.

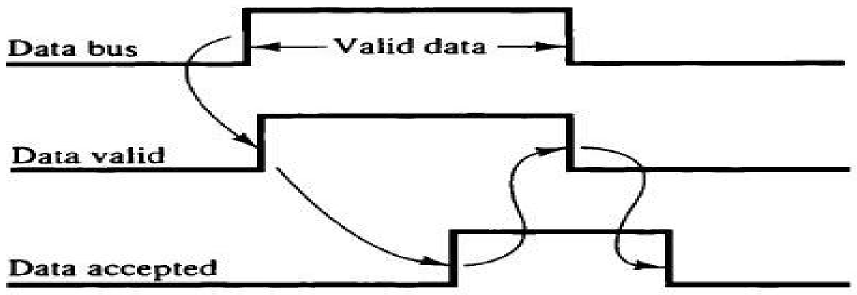
Source Initiated Transfer using Handshaking

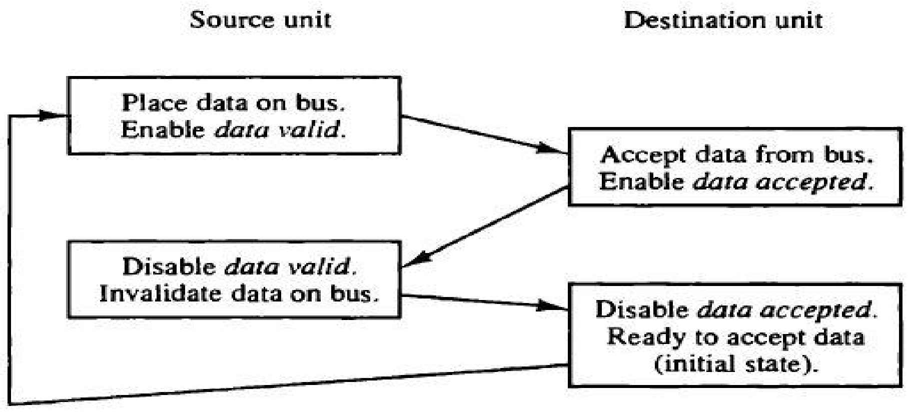
The two handshaking lines used in the process are data valid and data accepted.

Data valid: The data valid handshaking line is generated by the source unit.

Data accepted: The data accepted handshaking line is generated by the destination unit.

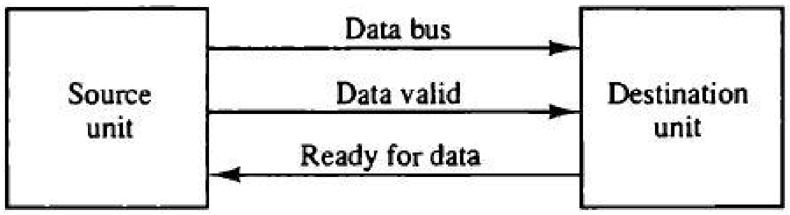


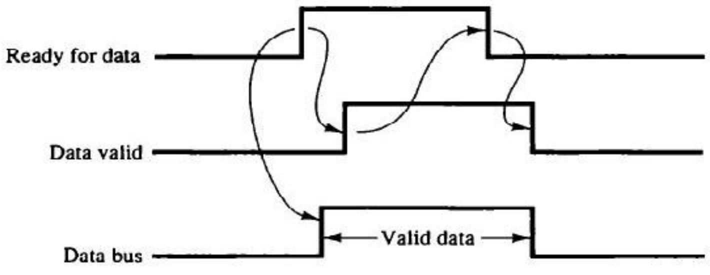


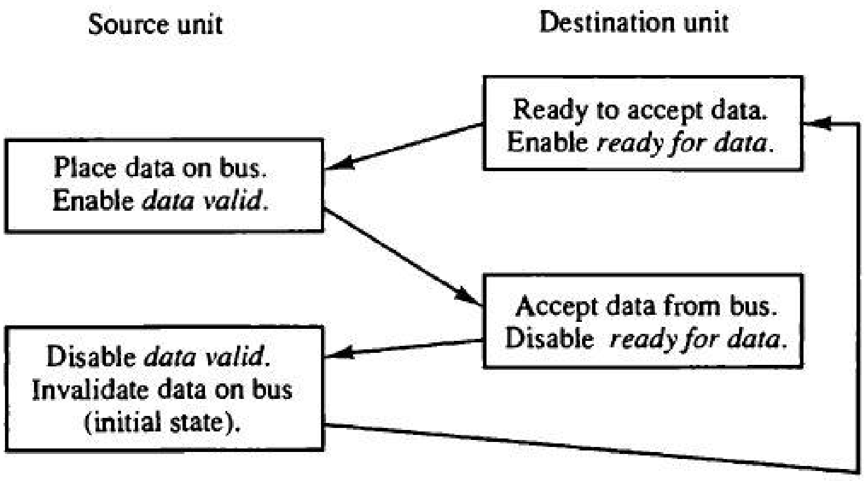

Source unit
Destination unit
Place data on bus. Enable *data valid*.
Accept data from bus. Enable *data accepted*.
Disable *data valid*. Invalidate data on bus.
Disable *data accepted*. Ready to accept data (initial state).

Destination Initiated Transfer using Handshaking

The name of the signal generated by the destination unit is ready for data to reflect its new meaning. The source unit in this case does not place data on the bus until after it receives the ready for data signal from the destination unit.






Source unit
Destination unit
Ready to accept data. Enable *ready for data*.
Place data on bus. Enable *data valid*.
Accept data from bus. Disable *ready for data*.
Disable *data valid*. Invalidate data on bus (initial state).

Timeout

If one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a timeout mechanism.This is implemented by means of an internal clock that starts counting time when the unit enables one of its handshaking control signals. If the return handshake signal does not respond within a given time period, the unit assumes that an error has occurred.

13.10 Asynchronous Serial Transfer

The transfer of data between two units may be done in parallel or serial.

Parallel Data transmission: Each bit of the message has its own path and the total message is transmitted at the same time.

Serial data transmission: Each bit in the message is sent in sequence one at a time.

Parallel transmission is faster but requires many wires. It is used for short distances and where speed is important. Serial transmission is slower but is less expensive since it requires only one pair of conductors.

13.11 Serial Synchronous Transmission

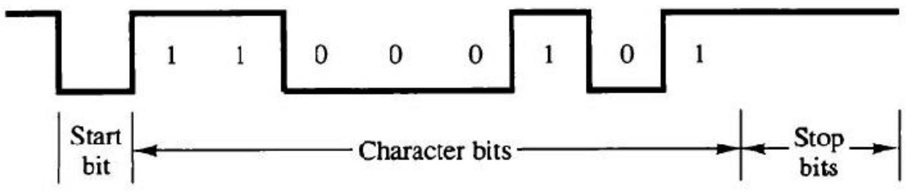
Serial transmission can be synchronous or asynchronous.

Synchronous transmission: In this, a common clock frequency and bits are transmitted continuously at the rate dictated by the clock pulses. In long distant serial transmission, each unit is driven by a separate clock of the same frequency.

Asynchronous transmission: In this, the binary information is sent only when it is available and the line remains idle when there is no information to be transmitted.

13.12 Serial Asynchronous Transmission:

A serial asynchronous data transmission technique employs special bits that are inserted at both ends of the character code.With this technique, each character consists of three parts: a start bit (always 0), the character bits, and stop bits (always 1).The transmitter rests at the 1-state when no characters are transmitted.



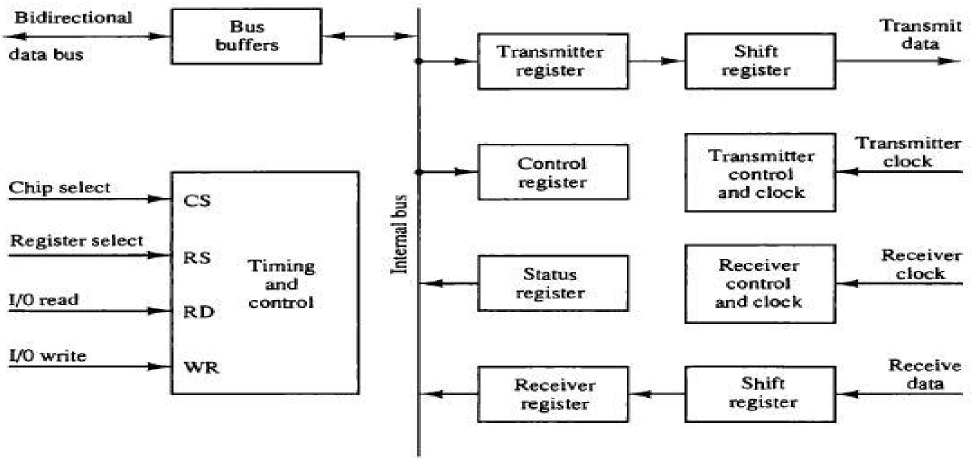
A transmitted character can be detected by the receiver from knowledge of the transmission rules.Using these rules, the receiver can detect the start bit when the line goes from 1 to 0. A clock in the receiver examines the line at proper bit times. The receiver knows the transfer rate of the bits and the number of character bits to accept. After the character bits are transmitted, one or two stop bits are sent. The stop bits are always in the I-state and frame the end of the character to signify the idle or wait state.This helps in resynchronization

Baud Rate

The baud rate is defined as the rate at which serial information is transmitted and is equivalent to the data transfer in bits per second.

Asynchronous communication interface

The terminal has a keyboard and a printer. Every time a key is depressed, the terminal sends 11 bits serially along a wire. To print a character in the printer, an 11-bit message must be received along another wire. The terminal interface consists of a transmitter and a receiver. Integrated circuits are available which are specifically designed to provide the interface between computer and similar interactive terminals. Such a circuit is called an asynchronous communication interface



CS	RS	Operation	Register selected
0	×	×	None: data bus in high-impedance
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register

13.13 Modes of Transfer

The memory unit is the essential part when dealing with data transfers. Data transfer between the central computer and I/0 devices may be handled in a variety of modes. Some modes use the CPU as an intermediate path; others transfer the data directly to and from the memory unit.

Data transfer to and from peripherals may be handled in one of three possible modes:

- Programmed I/0
- Interrupt-initiated I/0
- Direct memory access(DMA)

Programmed I/O

Each data item transfer is initiated by an instruction in the program. Transferring data under program control requires constant monitoring of the peripheral by the CPU.Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made. It is up to the programmed instructions executed in the CPU to keep close tabs on everything that is taking place in the interface unit and the I/0device.

Interrupt-Initiated I/O

The problem, i.e., CPU remaining in the same can be avoided by using an interrupt facility.The interface meanwhile keeps monitoring the device. When the interface determines that the device is ready for data transfer, it generates an interrupt request to the computer. The CPU stop the processing and branches to process the I/O transfer.

DMA

In direct memory access (DMA), the interface transfers data into and out of the memory unit through the memory bus. CPU initiates supplies the interface with the starting address and the number of words.When the transfer is made, the DMA requests memory cycles through the memory bus. When the request is granted by the memory controller, the DMA transfers the data directly into memory.

Summary

- I/O devices provide an efficient mode of communication between the central system and the outside environment.
- The CPU is an extremely fast device capable of performing operations at very high speed.
- The ASCII code contains 94 characters that can be printed and 34 nonprinting characters used for various control functions.
- Control characters are used for routing data and arranging the printed text into a prescribed format.
- Input / Output interface provides a method for transferring information between internal storage and external I/0 devices.
- There are four types of commands that an interface may receive. They are classified as control, status, data output, and data input.
- If one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a timeout mechanism.
- Parallel transmission is faster but requires many wires. It is used for short distances and where speed is important. Serial transmission is slower but is less expensive since it requires only one pair of conductors.

Keywords

Format effectors: Controls the layout of printing. Egs: BS, HT and CR.

Information separators: Separates the data into divisions like paragraphs and pages. Egs: RS and FS.

Communication control characters: Used during the transmission of text between remote terminals. Egs: ETX.

Control command: A control command is issued to activate the peripheral and to inform it what to do.

Status command: A status command is used to test various status conditions in the interface and the peripheral.

Strobe: A control signal (strobe pulse) for indicating the time at which data is being transmitted. One of the units indicates to the other unit when the transfer has to occur.

Handshaking: A control signal being transferred accompanying the data item from the sender's side. From the receiver's side, another control signal is sent to acknowledge.

Baud rate: The baud rate is defined as the rate at which serial information is transmitted and is equivalent to the data transfer in bits per second.

Self Assessment

1. In serial asynchronous transmission, the start bit is ____ and stop bit is ______.

A. 1,1

B. 0,0

C. 0,1

D. 1,0

2. When interface receives an item of data from the peripheral and places it in its buffer register. Then what kind of command is issued?
A. Control
B. Status
C. Data input
D. Data output

3. Which of these is requires many wires for connection and is usually faster?
A. Parallel transmission
B. Serial transmission
C. Equal transmission
D. None of the above

4. A _____________ command causes the interface to respond by transferring data from the bus into one of its registers.
A. Control
B. Status
C. Data input
D. Data output

5. The line data valid in source initiated data transfer using handshaking is generated by ________ unit.
A. Source
B. Destination
C. Either source or destination
D. Both source and destination

6. Which of the following command is issued to activate the peripheral?
A. Control
B. Status
C. Data input
D. Data output

7. Which of the method provides a reply in the form of control signal to the unit that initiates the transfer?
A. Strobe method
B. Handshaking method
C. Both strobe and handshaking methods
D. None of the above

8. Which of the following commands can be received by an interface?
A. Control
B. Status

C. Data input and output

D. All control, status, data input and output

9. In which method of data transfer, there is no way of knowing whether the other unit has actually placed or received the data?

A. Strobe method

B. Handshaking method

C. Both strobe and handshaking methods

D. None of the above

10. ASCII is a _____ bit code.

A. 6

B. 7

C. 8

D. 9

11. If two units, such as CPU and I/O interface which are designed independently of each other and they share a common clock, then what mode of transfer is this?

A. Synchronous mode

B. Asynchronous mode

C. Many-synchronous mode

D. None of the above

12. Which of the following control characters are used during the transmission of text between remote terminals?

A. Format effectors

B. Information separators

C. Communication control characters

D. None of the above

13. If two units, such as CPU and I/O interface which are designed independently of each other and they share a private clock, then what mode of transfer is this?

A. Synchronous mode

B. Asynchronous mode

C. Many-synchronous mode

D. None of the above

14. Which of the following control characters control the layout of printing?

A. Format effectors

B. Information separators

C. Communication control characters

D. None of the above

15. Which of the following control characters separates the data into divisions like pages and paragraphs?
A. Format effectors
B. Information separators
C. Communication control characters
D. None of the above

Answers for Self Assessment

1.	C	2.	C	3.	A	4.	D	5.	A
6.	A	7.	B	8.	D	9.	A	10.	B
11.	A	12.	C	13.	B	14.	A	15.	B

Review Questions

1. What are peripheral devices? Explain few examples of it.
2. What are control characters? Explain its types.
3. Explain input-output interface.
4. What is an Input-output bus and interface module?
5. What are the different ways that computer buses can be used to communicate with memory and I/O? Explain in detail.
6. What is strobe control? Explain the source initiated strobe for data transfer.
7. Explain destination initiated strobe for data transfer. What are the disadvantages of it?
8. Explain the process of handshaking? How the transfer is made when it is initiated by destination?
9. What are the modes of transfer? Explain in detail.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education, Asia, 2002.

Web Links

https://byjus.com/govt-exams/input-output-devices/

Unit 14: Hardware Description Logic

CONTENTS

Objectives

After studying this unit, you will be able to

- Understand the hardware description language
- Know various kinds of HDLs

- Know how to write the Verilog codes for various logic gates
- Know how to write the Verilog codes for Boolean functions and combinational circuits

Introduction

The hardware description logics (HDLs) are shorthand for describing the digital hardware. Begin the designing process by planning, on paper or in your mind, the hardware you want. Then write the HDL code that implies that hardware to a synthesis tool. There are two types of HDLs which are widely used: Verilog and VHDL.

Verilog

Verilog stands for Verify Logic. It provides a concept of module. It is a basic building block of hardware with inputs and outputs is called a module. It is a thing between the keywords module and endmodule.

AND gate, a multiplexer, and a priority circuit.

14.1 Characteristics of Verilog

Case sensitive

Vendor independent

Supports simulation

Supports synthesis

14.2 Program Structure in Verilog

module <module_name> (input, output);

…………………..

………………….

<logic of program>

………………………

Endmodule

14.3 Declaration of Input and Output & Comments

The inputs can be declared by using the keyword input.

input a, b. This representation is for two inputs each of one bit.

input [3:0]a, b. This representation is for four bit inputs.

The comments are used for helping the reader for better understanding. // are used for comments in a single line, whereas /*…………………………………..*/ are used for multiple lines.

Operators in Verilog

An operator, in many ways, is similar to a simple mathematical operator.

Arithmetic operator

Logical operator

Bitwise operator

Reduction operator

Relational operator

Equity operator

Conditional operator

14.4 Arithmetic Operators

Expression	Operator	Operation
a + b	+	Add
a – b	-	Subtract
a * b	*	Multiply
a / b	/	Divide
a % b	%	Modulus
a ** b	**	Power

14.5 Logical Operators

Expression	Operator	Operation
a && b	&&	Logical AND
a || b	||	Logical OR
!a	!	Logical Negation

14.6 Bitwise Operators

Expression	Operator	Operation
~a	~	Negation
a & b	&	Bitwise AND
a | b	|	Bitwise OR
a ^ b	^	Bitwise XOR

a ^~ b	^~	Bitwise XNOR

14.7 Reduction Operators

Expression	Operator	Operation
&A	&	Performs bitwise AND operation on A
|A	|	Performs bitwise OR operation on A
^A	^	Performs bitwise XOR operation on A

14.8 Relational Operators

Expression	Operator	Operation
a>b	>	Greater than
a<b	<	Less than
a<=b	<=	Greater than or equal to
a>=b	>=	Less than or equal to

14.9 Equity Operator

Expression	Operator	Operation
a==b	==	Equal to
a!=b	!=	Not equal to

14.10 Conditional Operator

condition? true_expression : false_expression

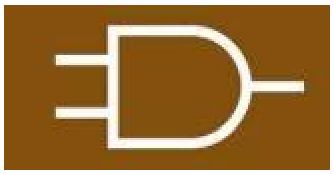
Verilog Code for various gates

14.11 Verilog code for Inverter

The truth table for Not gate or Inverter is

A (Input)	B (Output)
0	1
1	0

The logic symbol is



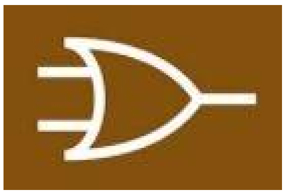
```
module not_gate ( input a, output c );
assign c=~a;
endmodule
```

14.12 Verilog code for OR gate

The truth table for OR gate is

A (Input)	B (Input)	C (Output)
0	0	0
0	1	1
1	0	1
1	1	1

The logic symbol is



```
module and_gate ( input a, input b, output c );
assign c=a & b;
endmodule
```

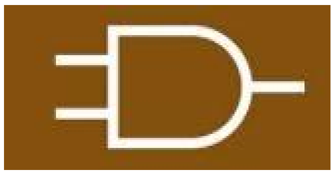
14.13 Verilog code for AND gate

The truth table for AND gate is

A (Input)	B (Input)	C (Output)
0	0	0
0	1	0

1	0	0
1	1	1

The logic symbol is



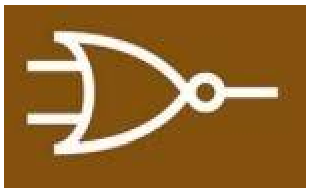
```
module and_gate ( input a, input b, output c );
assign c=a & b;
endmodule
```

14.14 Verilog Code for NOR gate

The truth table for NOR gate is

A (Input)	B (Input)	C (Output)
0	0	1
0	1	0
1	0	0
1	1	0

The logic symbol is



```
module nor_gate ( input a, input b, output c );
        assign c=~(a | b);
        endmodule
```

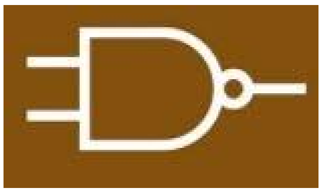
14.15 Verilog code for NAND gate

The truth table for NAND gate is

A (Input)	B (Input)	C (Output)

0	0	1
0	1	1
1	0	1
1	1	0

The logic symbol is



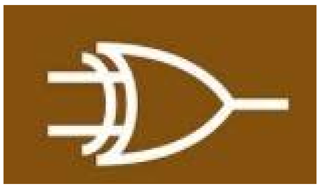
```
module nand_gate ( input a, input b, output c );
          assign c=~(a & b);
          endmodule
```

14.16 <u>Verilog code for XOR gate</u>

The truth table for XOR gate is

A (Input)	B (Input)	C (Output)
0	0	0
0	1	1
1	0	1
1	1	0

The logic symbol is



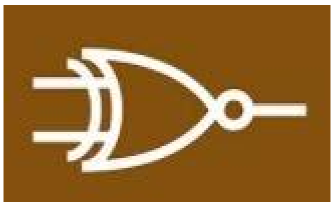
```
module xor_gate ( input a, input b, output c );
          assign c=a ^ b;
          endmodule
```

14.17 Verilog code for XNOR gate

The truth table for XNOR gate

A (Input)	B (Input)	C (Output)
0	0	1
0	1	0
1	0	0
1	1	1

The logic symbol is



```
module xnor_gate ( input a, input b, output c );
        assign c=~(a ^ b);
        endmodule
```

```
Y=A'B'C'+AB'C'+AB'C,
module evaluate (input logic a, b, c, output logic y);
assign y = ~a & ~b & ~c |a & ~b & ~c |a & ~b & c;
endmodule
```

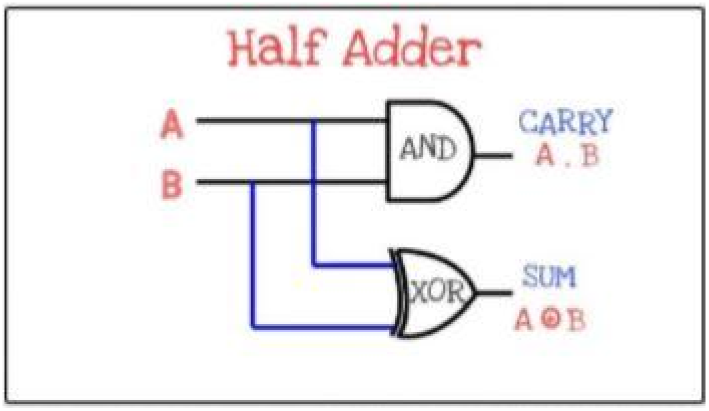
Verilog codes for combinational circuits

14.18 Verilog code for half adder

The truth table for half adder is

a	b	Sum	Carry
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The circuit for half adder is



The expressions for sum and carry are

Sum=a'b + ab'

Carry=ab

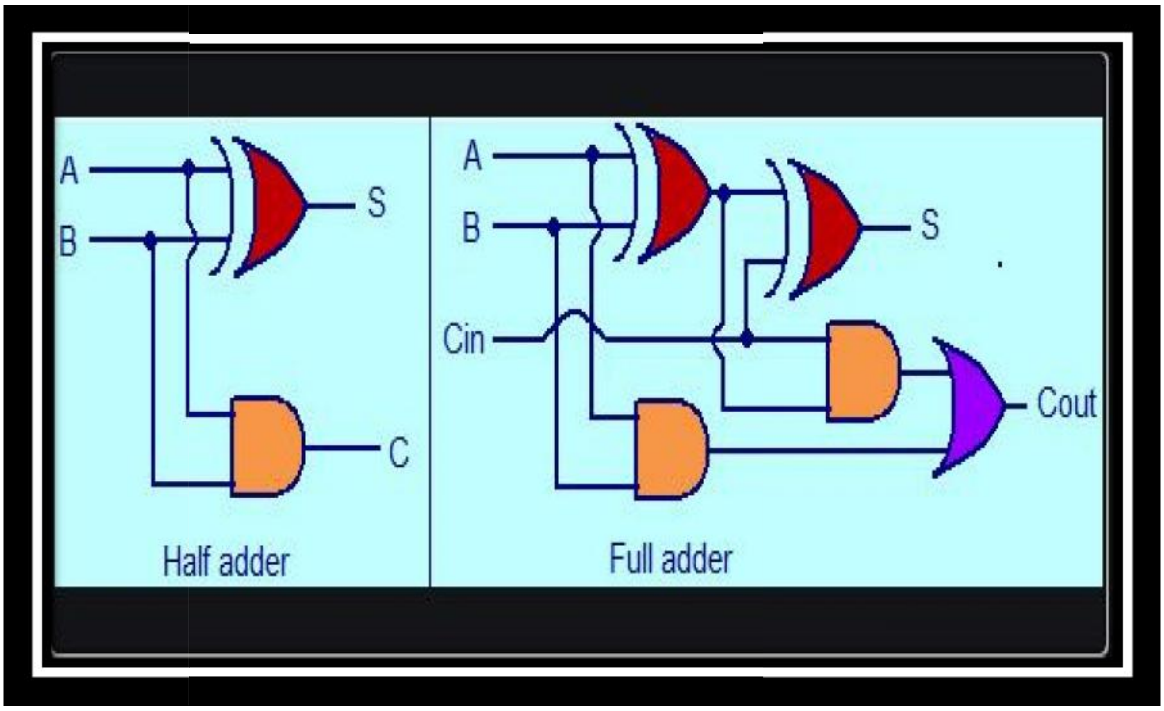
```
module test_halfadder(input logic a,b, output logic sum,carry);
assign sum=a^b, carry=a&b;
endmodule;
```

14.19 Verilog code for full adder

The truth table for full adder is

a	b	C_{in}	C_{out}	S
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

The circuit for full adder is



The expressions for sum and carry are:

$S=a'b'C_{in}+a'bC_{in}'+ab'C_{in}'+abC_{in}$

$C=ab+aC_{in}+bC_{in}$

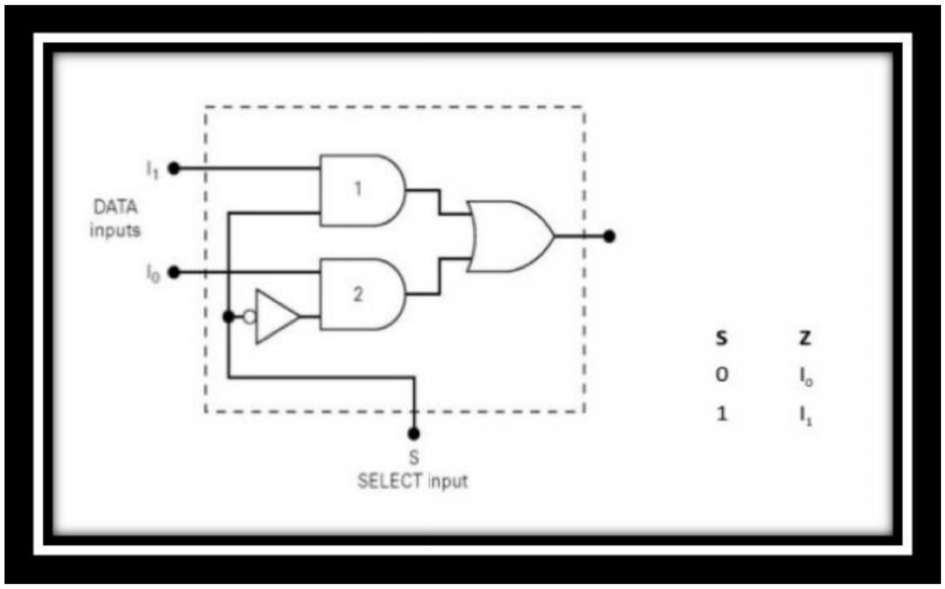
```
module fullAdder ( input logic a, b, Cin, output logic S, Cout);
assign S = (a ^ b) ^ Cin ,
Cout = (a & b) | (b & Cin) | (Cin& a);
Endmodule
```

14.20 <u>Verilog code for 2-to-1 multiplexer</u>

The truth table for 2-to-1 multiplexer

S	Z
0	I0
1	I1

The circuit for 2-to-1 multiplexer is



The expressions for 2-to-1 multiplexer is

Output=i0 when S=0, Y=i0.S’

Output=i1 when S=1, Y=i1.S.

Z=i0.S’+i1.S

module Mux_2_To_1(input logic S, i0, i1, output logic Z);

assign Z = S ? i0 : i1;

endmodule

14.21 Verilog code for 4-to-1 multiplexer

The truth table for 4-to-1 multiplexer is:

S0	S1	Z
0	0	I0
0	1	I1
1	0	I2
1	1	i3

The expression for 4-to-1 multiplexer is

O/P is i0 only if S0=0 and S1=0, Z=i0*s0’*s1’

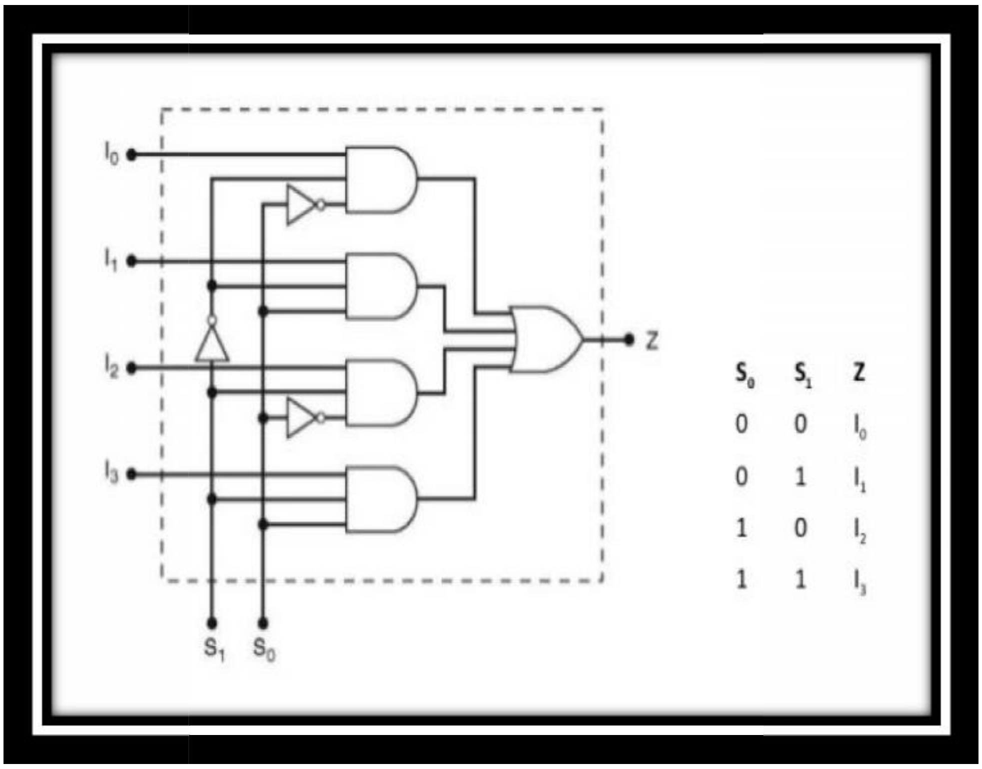
O/P is i1 only if S0=0 and S1=1, Z=i1*s0′*s1

O/P is i2 only if S0=1 and S1=0, Z=i2*s0*s1′

O/P is i3 only if S0=1 and S1=1, Z=i3*s0*s1

Z=i0*s0′*s1′ + i1*s0′*s1 + i2*s0*s1′ + i3*s0*s

The circuit for 4-to-1 multiplexer is



```
module mux_4x1 (output logic Z, input logic i0, i1, i2, i3, s0, s1);
assign out = (~s0 & ~s1 & i0) | (s0 & ~s1 & i1) | (~s0 & s1 & i2) | (s0 & s1 & i0);
endmodule
```

14.22 Verilog code for 1-to-4 Demultiplexer

The truth table for 1-to-4 demultiplexer is

Data Inputs		Select inputs		Outputs		
D	S1	S2	Y3	Y2	Y1	Y0
D	0	0	0	0	0	D
D	0	1	0	0	D	0

D	1	0	0	D	0	0
D	1	1	D	0	0	0

The expressions for 1-to-4 demultiplexer are:

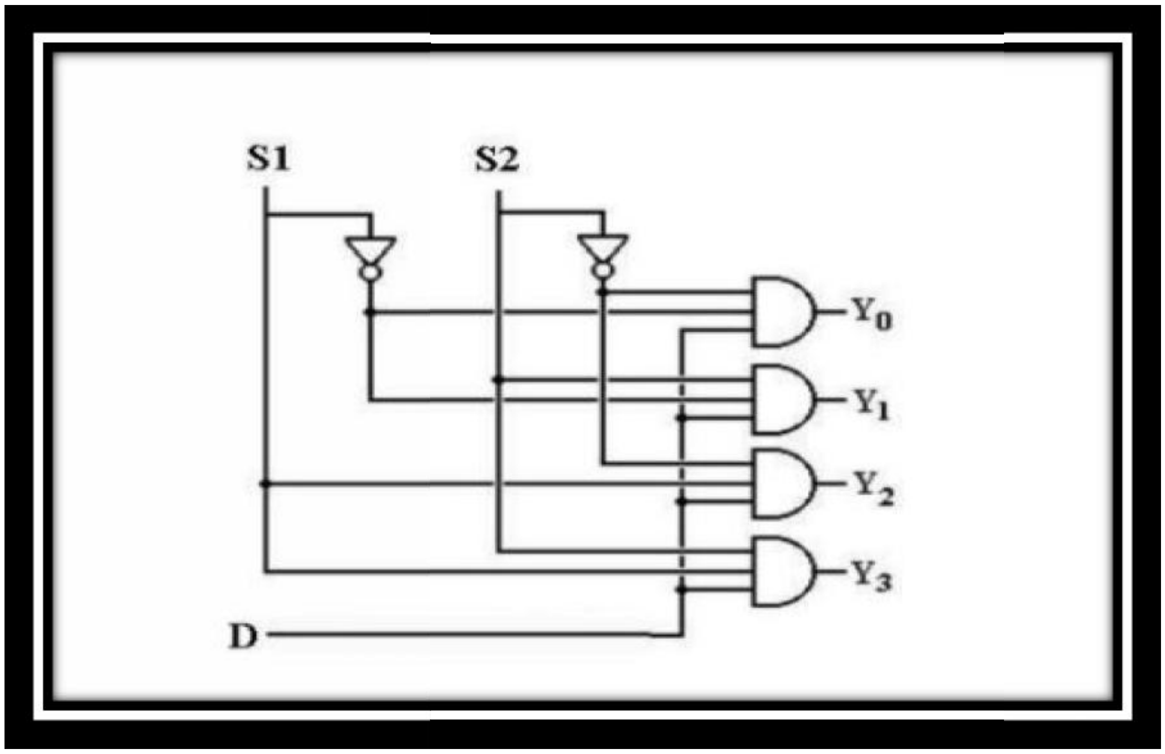
Y0=S1'S2'D

Y1=S1'S2D

Y2=S1S2'D

Y3=S1S2D

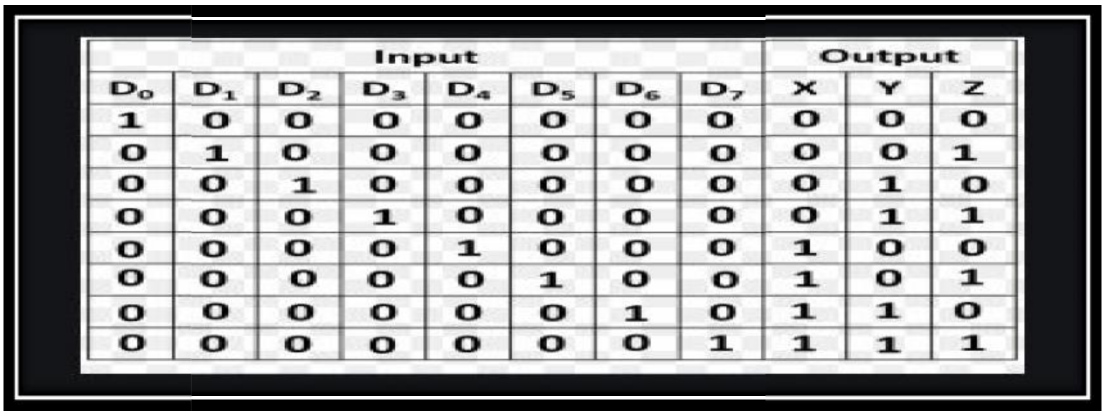
The logic circuit for 1-to-4 demultiplexer is



```
module demultiplexer(input logic d,s1,s2,output logic y0,y1,y2,y3);
assign y0=(d & ~s2 & ~s1),
y1=(d & s2 & ~s1),
y2=(d & ~s2 & s1),
y3=(d & s2 & s1);
endmodule
```

14.23 Verilog code for 8-to-3 encoder

The truth table for 8-to-3 encoder is



Input								Output		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	Y	Z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

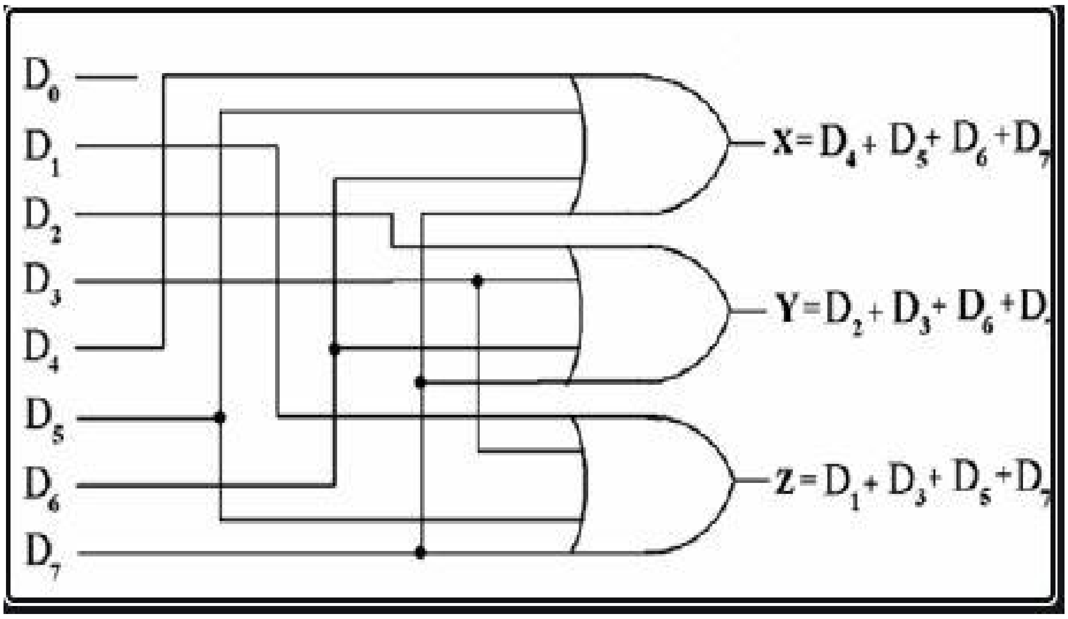
The expressions for 8-to-3 encoder is

X= $D_4+D_5+D_6+D_7$

Y= $D_2+D_3+D_6+D_7$

Z= $D_1+ D_3+ D_5 +D_7$

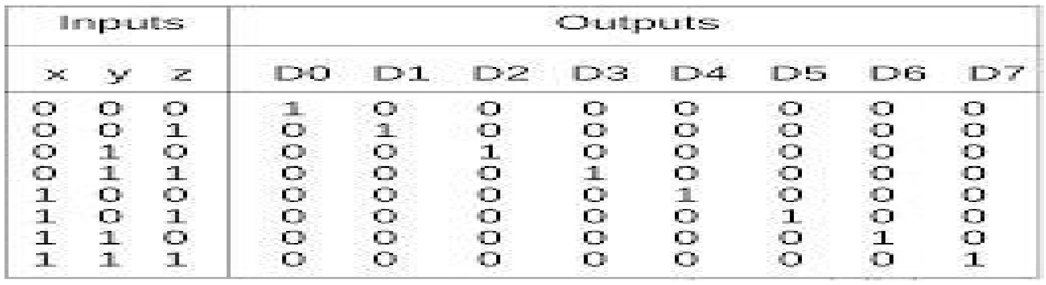
The circuit is



```
module Encoder(input logic d0,d1,d2,d3,d4,d5,d6,d7, output logic x,y,z);
assign x=(d4 | d5 | d6 | d7),
y=(d2 | d3 | d6 | d7),
z=(d1 | d3 | d5 | d7);
endmodule
```

14.24 Verilog code for 3-to-8 decoder

The truth table for 3-to-8 decoder is



Inputs			Outputs							
x	y	z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

The expressions for 3-to-8 decoder are

D0=x′y′z′

D1=x′y′z

D2=x′yz′

D3=x′yz

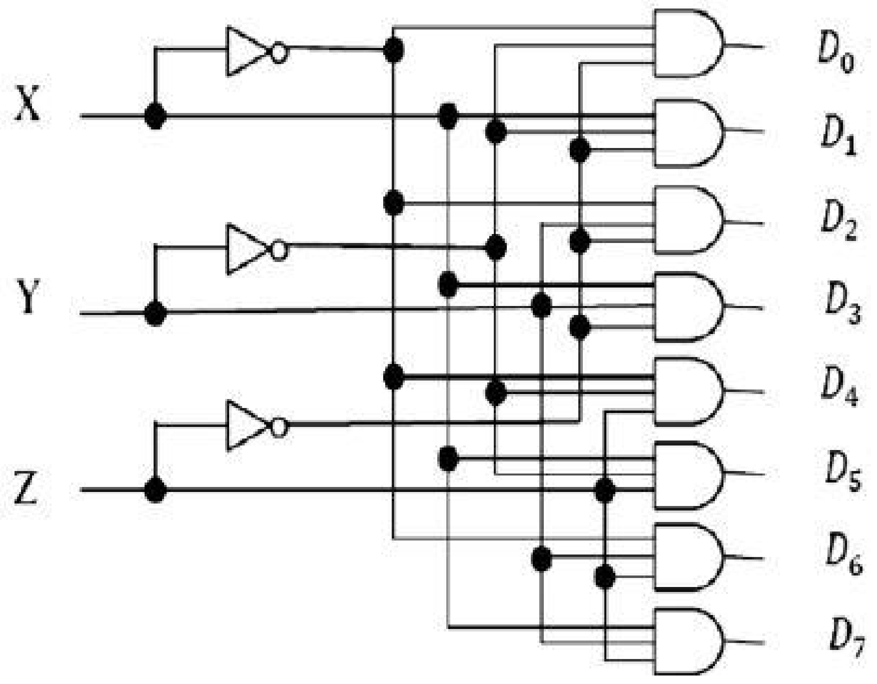
D4=xy′z′

D5=xy′z

D6=xyz′

D7=xyz

The circuit is



module Decoder(input logic x,y,z, output logic d0,d1,d2,d3,d4,d5,d6,d7);

assign d0=(~x&~y&~z), d1=(~x&~y&z), d2=(~x&y&~z),d3=(~x&y&z), d4=(x&~y&~z), d5=(x&~y&z), d6=(x&y&~z),d7=(x&y&z);

endmodule

VHDL

VHDL stands for VHSIC hardware description language. VHSIC stands for very high speed integrated circuit. It is languages which can replica the behavior and organization of systems at multiple intensity of generalization; it can be from level of simple basic gates to the level of systems, for the purpose of verification and documentation.

Summary:

- HDL is used as shorthand for describing the digital hardware.
- Two HDLs are widely used: VHDL and Verilog.
- The HDLs are used to meet the decided hardware to synthesis tool.
- We can write the Verilog code for various logic gates.
- We can write the Verilog codes for Boolean functions.
- We can write the Verilog codes for combinational circuits.

Keywords:

Module: The Verilog provides the concept of module.

Verilog: The Verilog is a case sensitive and vendor independent language. It supports simulation and synthesis.

VHDL: It stands for VHSIC hardware description logic.

Self Assessment:

Q 1: In verify logic, we have keywords.

A. module

B. endmodule

C. Both module and endmodule

D. None of the above

Q 2: In verilog, how do we provide four bits input?

A. input [3:0] a;

B. input [0:3] a;

C. input [0-3] a;

D. input [3-0] a;

Q 3: && represents

A. Logical OR

B. Logical AND

C. Logical AND

D. None of the above

Q 4: X ^ Y represents

A. Bitwise AND

B. Bitwise OR

C. Bitwise XOR

D. Bitwise XNOR

Q 5: The conditional operator is defined by

A. ?:

B. :?

C. ?@

D. @?

Q 6: Which of the following is an HDL?

A. Verilog

B. VHDL

C. Both Verilog and VHDL

D. None of the above

Q 7: The modulus operation is performed using

A. +

B. -

C. %

D. *

Q 8: Out=~X, if X=0, then what will be the value of Out?

A. 0

B. 1

C. 2

D. None of the above

Q 9: module abc (input a, output c);

assign c=~a;

endmodule

A. NOT gate

B. AND gate

C. OR gate

D. XOR gate

Q 10: module abc_gate (input a, input b, output c);

assign c=~(a | b);

endmodule

A. NOR gate

B. AND gate

C. OR gate

D. XOR gate

Q 11: module abc_gate (input a, input b, output c);

```
assign c=~(a & b);
endmodule
```

A. NAND gate

B. AND gate

C. OR gate

D. XOR gate

Q 12: module abc_gate (input a, input b, output c);

```
assign c=~(a ^ b);
endmodule
```

A. NAND gate

B. AND gate

C. XNOR gate

D. XOR gate

Q 13: module xyz(input logic a, b, output logic sum, carry);

```
assign sum =a^b, carry =a&b'
endmodule
```

A. Half adder

B. Full adder

C. Half Subtractor

D. Full Subtractor

Q 14: module xyz (input logic S, i0, i1, output logic Z);

```
assign Z=S?i0:i1
endmodule
```

A. Half adder

B. Full adder

C. Multiplexer

D. De-multiplexer

Q 15: Which of these defines the capabilities of Verilog?

A. Case sensitive

B. Vendor independence

C. Both case sensitivity and independence from vendor

D. None of the above

Answers for Self assessment

1.	C	2.	A	3.	B	4.	C	5.	A
6.	C	7.	C	8.	B	9.	A	10.	A
11.	A	12.	C	13.	A	14.	C	15.	C

Review Questions

Q 1: What is an HDL? Write the Verilog code for fundamental gates.

Q 2: Write the Verilog code for Boolean function F = A′BC + AB′C + ABC′ + ABC.

Q 3: Write the Verilog code for Boolean function F = X′Y′Z′ + X′Y′Z + X′YZ + X′YZ′ + XY′Z′ + XYZ′.

Q 4: Explain two types of HDLs, i.e., Verilog and VHDL.

Further Readings

M. Morris Mano, Digital Design, Third Edition, Pearson Education Asia, 2002.

Web Links

https://www.allaboutcircuits.com/technical-articles/what-is-a-hardware-description-language-hdl/

https://www.sciencedirect.com/topics/computer-science/hardware-description-languages

LOVELY PROFESSIONAL UNIVERSITY
Jalandhar-Delhi G.T. Road (NH-1)
Phagwara, Punjab (India)-144411
For Enquiry: +91-1824-521360
Fax.: +91-1824-506111
Email: odl@lpu.co.in