AMBO UNIVERSITY

WOLISO CAMPUS

DEPARTMENT OF COMPUTER SCIENCE

COMPUTER OGANIZATION AND ARCHITECTURE HANDOUT FOR 2ND YEAR EXTENTION STUDENT

PREPARED BY CHALTU M.

MAY 25,2020

AMBO,ETHIOPAI

Table of Contents

Chapter 1: Digital Circuitry Logic

1.1 Introduction

A digital implies that the information in the computer represented by variables that take limited number of discrete values.

The digital computer is a digital system that performs various computational tasks. Digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a bit.

Computer system is subdivided in to two functional entities are hard ware and software of the computer.

Hardware of the computer is divided in to 3 major parts: they includes cpu, RAM and input and output processor (IOP).

Computer architecture: refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program. **Examples** of architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and techniques for addressing memory.

Computer organization refers to the operational units and their interconnections that realize the architectural specifications. Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer and peripherals; and the memory technology used.

The computer designer is concerned with structure and function characteristics of computer system:

- **Structure:** The way in which the components are interrelated
- **Function:** The operation of each individual component as part of the structure

Computers can perform four basic functions:

- Data processing
- Data storage
- Data movement
- Control

There are four main structural components:

- **Central processing unit (CPU):** Controls the operation of the computer and performs its data processing functions; often simply referred to as **processor**.
- **Main memory:** Stores data.
- **I/O:** Moves data between the computer and its external environment.
- **System interconnection:** Some mechanism that provides for communication among CPU, main memory, and I/O.

A computer system can be subdivided into two functional entities: hardware and software.

Computer **Hardware** consists all of the electronic component and electromechanical devices that comprises the physical entities of the devicese.

Computer **software** consists of the instructions and data that the computer manipulates to perform various data-processing tasks. A sequence of instructions for the computer is called a program.

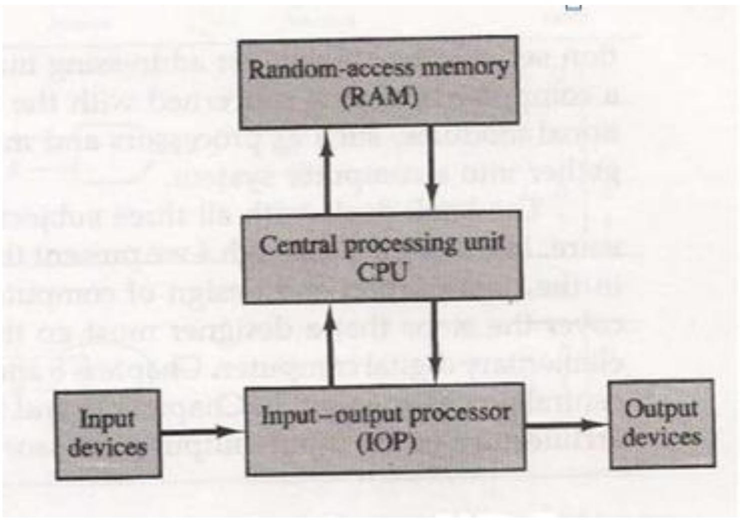


Figure 1.1: Block Diagram of Digital Computer.

1.2 Logic Gates

Binary information in digital computers represented by physical quantities called signal.
The fundamental building block of all digital logic circuits is the gate. Logical functions are implemented by the interconnection of gates. A gate is an electronic circuit that produces an output signal that is a simple Boolean operation on its input signals. The basic gates used in digital logic are AND, OR, NOT, NAND, NOR, and XOR. Each gate is defined in three ways: graphic symbol, algebraic notation, and truth table. Note that the inversion (NOT) operation is indicated by a circle.

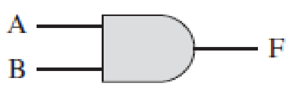
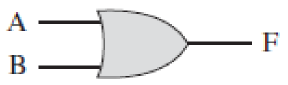
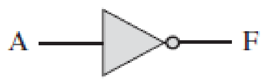
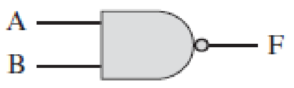
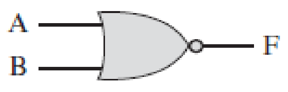
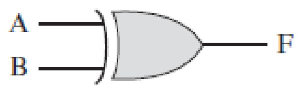
Name	Graphical Symbol	Algebraic Function	Truth Table
AND		$F = A \bullet B$ or $F = AB$	A B | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = A + B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
NOT		$F = \overline{A}$ or $F = A'$	A | F 0 | 1 1 | 0
NAND		$F = \overline{AB}$	A B | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = \overline{A + B}$	A B | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
XOR		$F = A \oplus B$	A B | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0

Figure 1.2: Basic logic gates

1.3 Boolean Algebra

The digital circuitry in digital computers and other digital systems is designed, and its behavior is analysed, with the use of a mathematical discipline known as *Boolean algebra*. The name is in honor of an English mathematician George Boole, who proposed the basic principles of this algebra in 1854 in his essay.

Boolean algebra makes use of logical variables and logical operators. The possible values for a logical variable are either TRUE or FALSE. For ease of use, these values are, conventionally, represented by 1 and 0 respectively. A system in which the only possible values are 0 and 1 is the binary number system. Likewise, it is similar to the binary states of digital electronics and that is why Boolean algebra is used to analyse digital circuits. The logical operators of Boolean algebra are AND, OR, and NOT, which are symbolically represented by dot (·), plus sign (+), and over bar (¯). Often the dot is omitted in Boolean expression. Hence, A·B is written as AB without the dot.

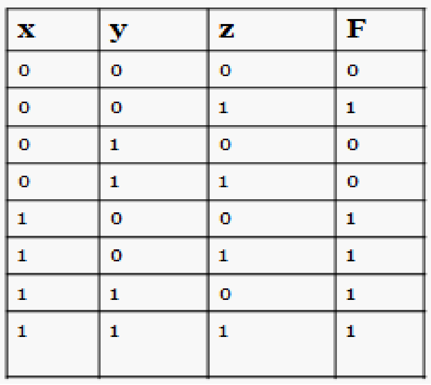
X	Y	$\overline{X}$	X+Y	X·Y	$\overline{X+Y}$	$\overline{X \cdot Y}$	X⊕Y
0	0	1	0	0	1	1	0
0	1	1	1	0	0	1	1
1	0	0	1	0	0	1	1
1	1	0	1	1	0	0	0

Table 1.1: Truth table for Boolean operators

A Boolean function can be expressed algebraically with logic variables and logic operators.

Example, ***F= x +y'z***

A Boolean function can be represented by a truth table and a logic diagram. Truth table and logic diagram of Boolean function F is shown below.



x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

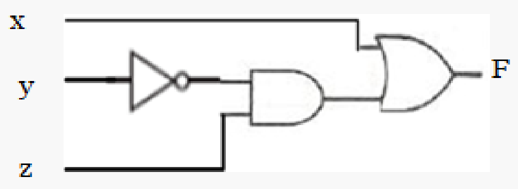
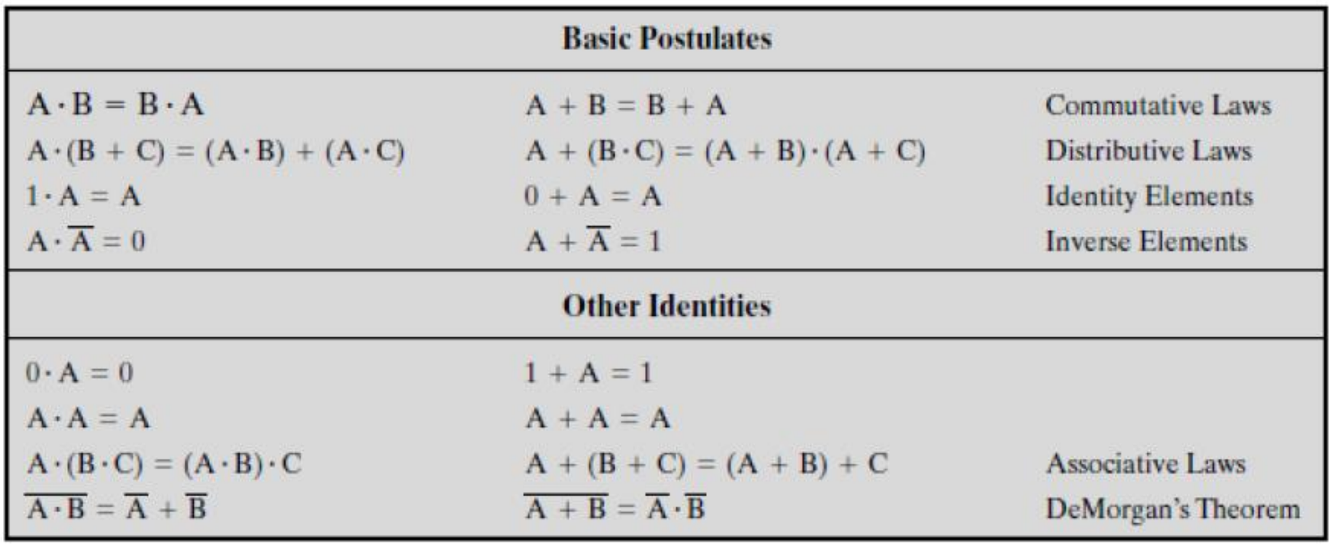


Table 1.2: important identities of Boolean algebra.

These identities are useful in simplifying Boolean functions in order to find simple circuit designs.



Basic Postulates		
$A \cdot B = B \cdot A$	$A + B = B + A$	Commutative Laws
$A \cdot (B + C) = (A \cdot B) + (A \cdot C)$	$A + (B \cdot C) = (A + B) \cdot (A + C)$	Distributive Laws
$1 \cdot A = A$	$0 + A = A$	Identity Elements
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	Inverse Elements
Other Identities		
$0 \cdot A = 0$	$1 + A = 1$	
$A \cdot A = A$	$A + A = A$	
$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	$A + (B + C) = (A + B) + C$	Associative Laws
$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$	DeMorgan's Theorem

Table 1.3: Basic identities of Boolean algebra

For example, consider the following Boolean algebra expression:

AB’ + C’D + AB’ + C’D

By letting A= AB’ + C’D the expression be written as A + A. From identity element of Table 1-3, we find that A+A =A. Thus, the expression can be simplified as:

AB’ + C’D + AB’ + C’D = AB’ + C’D

Example:1. Simplify the expression x’y’z+yz+xz

Solution x’y’z+yz+xz

=z(x’y’+y+x) (distributive property)

=z(x’+y+x) (adsorption)

=z(x’+x+y) (commutativity)

=z(1+y) (complementation)

=z(1)=z (Dominance law)

2. Simplify the expression: F=ABC+ABC’+A’C

K-MAP Simplification

Rules for K-map simplification

1. Group may not contain zero.
2. We can group by 1, 2, 4and8 grouping:2^n cells.
3. Each group should be large as possible.
4. Cell contain 1 must be grouped.
5. Groups may be overlap.
6. Opposite grouping and corner grouping are allowed.
7. There should be as few groups as possible.

Example: 1.simlify the expression A’B’C’+A’BC’+A’BC+ABC’

=A’C’+A’B+BC’

2.simplify F(A,B,C)=(0,2,4,5,6)

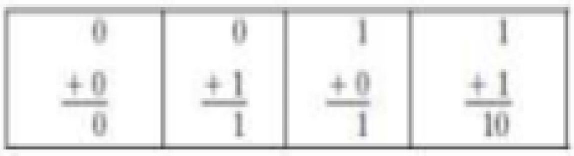
1.4 Combinational Circuits

- A combinational circuit is an interconnected set of gates whose output at any time is a function only of the input at that time. As with a single gate, the appearance of the input is followed almost immediately by the appearance of the output, with only gate delays. In general terms, a combinational circuit consists of n binary inputs and m binary outputs.

- Combinational circuits have no feedback (a feedback is an output that is given back to one of the inputs of the circuit)
- Adders are simple examples of arithmetic circuits.
- They serve as a basic building blocks for the construction of more complicated arithmetic circuits.
- A combinational circuit can be defined in three ways:
  - ✓ **Truth table** : For each of the 2n possible combinations of input signals, the binary value of each of the m output signals is listed.
  - ✓ **Graphical symbols** : The interconnected layout of gates is depicted.
  - ✓ **Boolean equations**: Each output signal is expressed as a Boolean function of its input signals.

Common Combinational Circuits are:

Adders (Half Adder & Full Adder): Binary addition differs from Boolean algebra in that the result includes a carry term. Thus,

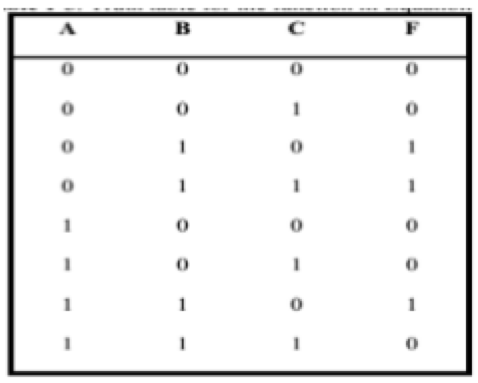


0	0	1	1
+0	+1	+0	+1
0	1	1	10

Implementation of Boolean Functions : Any Boolean function can be implemented in electronic form as a network of gates. For any given function, there are a number of alternative realizations. Consider the Boolean function represented by the truth table .We can express this function by simply itemizing the combinations of values of A, B, and C that cause F to be 1:

$$F = \overline{A}B\overline{C} + \overline{A}BC + AB\overline{C}$$

There are three combinations of input values that cause F to be 1, and if any one of these combinations occurs, the result is 1. This form of expression, for self-evident reasons, is known as the sum of products (SOP) form. Figure 1-3 shows a straight forward implementation with AND, OR, and NOT gates.



A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Figure 1.3 Truth table for function in equation

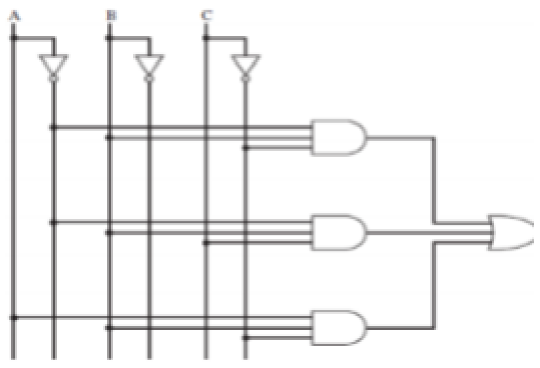


Figure 1.4 Sum of production implementation

Half Adder

- A digital arithmetic circuit that carries out the addition of two bits is called a **half adder**.
- It has two input variables and two outputs variables (sum & carry).

$S = X \oplus Y$, $C = xy$

X	Y	S(sum)	C(carry)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 1.4: Truth Table of half adder.

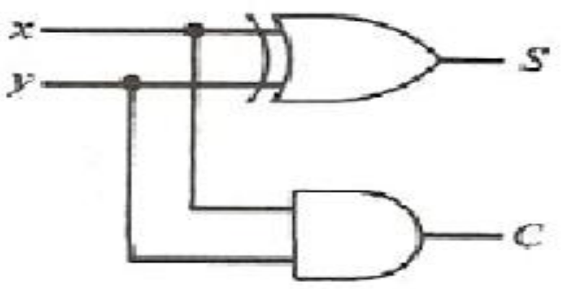


Figure 1.5: Logic Diagram

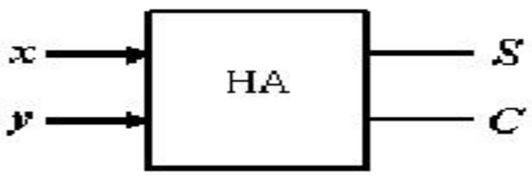


Figure 1.6: Block Diagram

Full Adder

- We are not only interested in performing addition on just a single pair of bits.
- Rather, we wish to add two n-bit numbers along with a carry from a previous bitwise addition (performs addition of three bits).
- Such digital circuit is called a **full adder**.

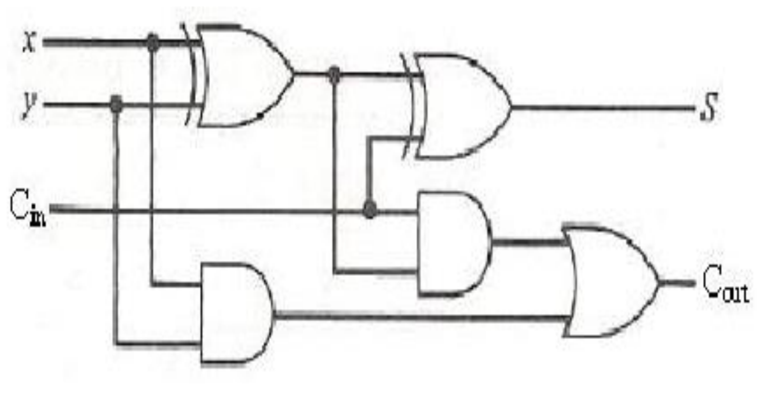
  A combination of two half adders creates a ***full adder.***

Cin	X	Y	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1.5: Truth Table of full adder.

$S = X \oplus Y \oplus Cin$

$C = XY + (X \oplus Y)Cin$



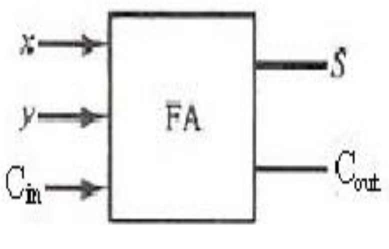


Figure 1.7 Logic Diagram Figure 1.8 Block Diagram

Multiple-Bit Adder

- By combining a number of full adders, we can have the necessary logic to implement a multiple-bit adder.
- The output from each adder depends on the carry from the previous adder.
- E.g Construction of a 32-bit adder using 8-bit adders

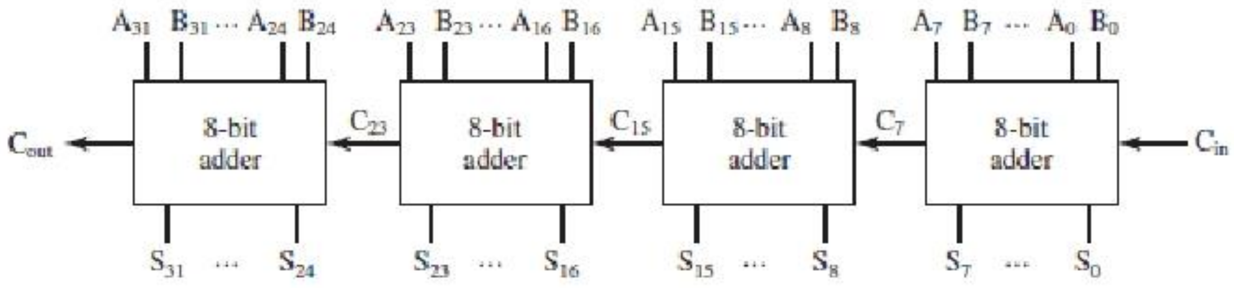


Figure 1.9 Block Diagram

1.5 Sequential Circuits

- In case of combinational circuits, the value of each output depends on the values of signals applied to the inputs.
- However, in case of Sequential Circuits, the values of the outputs depend not only on the present values of the inputs but also on the past behavior of the circuit.
- Sequential circuits consist of a combinational circuit and some memory elements. The memory elements are used to store information about the past.
- Such circuits include storage elements that store the values of logic signals.
- Flip-flops can store a one-bit data (information).
- The value stored in a flip-flop is called the **state** of the flip-flop, and is designated by the letter Q.

- A second output, Q’ is also usually provided, which is the complement of the state of the flip-flop.

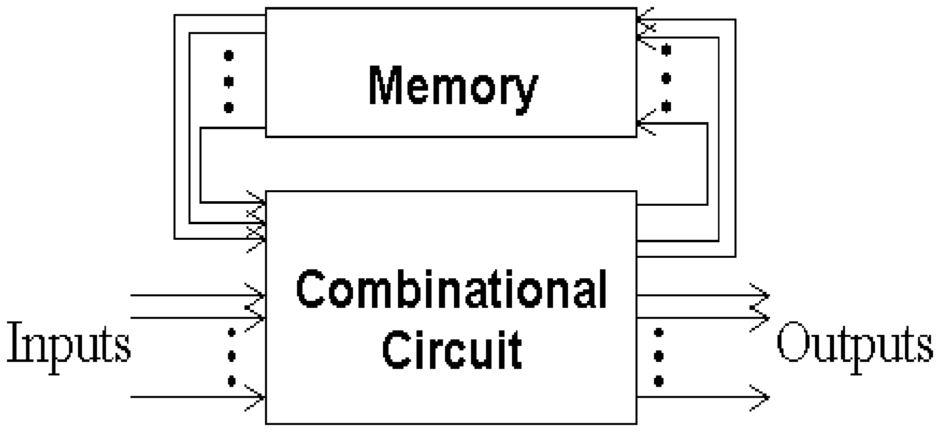


Figure 1.10: diagram of sequential circuit

1.6 Flip-Flops

The memory elements used in sequential circuits are called flip-flops. Flip-flops can store or remember a 0 or a 1 (a Boolean value). We use the term **bit** (**bi**nary digi**t**) to refer to a Boolean value.

- So, flip-flops can store a one-bit data (information).
- flip-flops is The simplest form of sequential circuit.
- There are *a variety of flip-flops,* all of which share two properties:
- The flip-flop is a bistable device, i.e. has two stable states.
- It exists in one of two states and, in the absence of input- function as a 1-bit memory.
- The flip-flop has two *outputs, Q and* the *complement* of Q.
- E.g S-R, J-K & D flip-flops

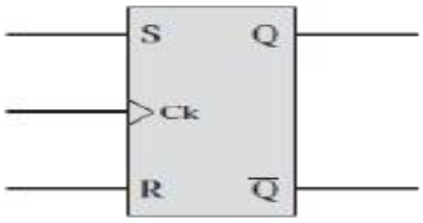
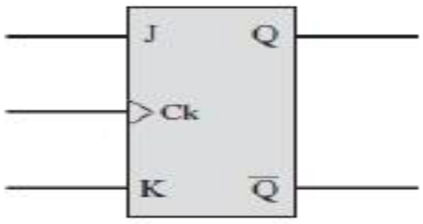
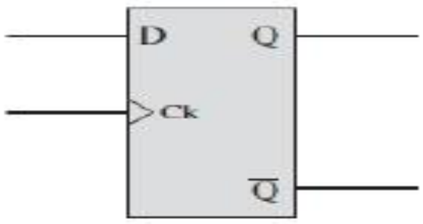
Name	Graphical Symbol	Truth Table
S–R		$\begin{array}{ccc} S & R & Q_{n+1} \\ \hline 0 & 0 & Q_n \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & — \end{array}$
J–K		$\begin{array}{ccc} J & K & Q_{n+1} \\ \hline 0 & 0 & Q_n \\ 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & \overline{Q_n} \end{array}$
D		$\begin{array}{cc} D & Q_{n+1} \\ \hline 0 & 0 \\ 1 & 1 \end{array}$

Figure 1.10 basic types of Flip Flops

1.6.1 S-R Flip-Flops

- The circuit has two inputs, S (Set) and R (Reset), and two outputs, Q & complement of Q.
- Mostly, events in the digital computer are synchronized to a clock pulse, so that changes occur only when a clock pulse occurs.
- the clock signal is a Boolean variable that alternatingly takes the values 0 and 1.
- The S and R inputs are passed to the NOR gates only during the clock pulse.
- Only when the clock signal changes [0-1] can output affected according to the values of input S and R.

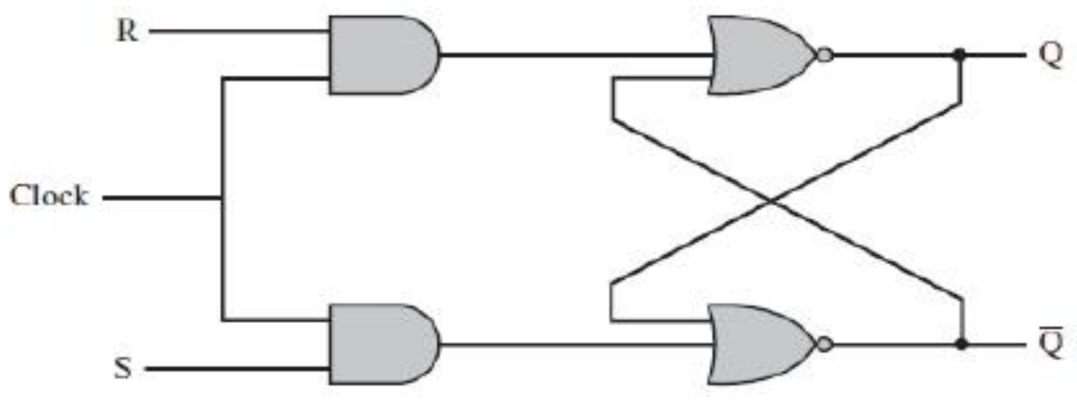


Figure 1.11 S-R Flip-Flops

1.6.2 J-K Flip-Flops

- Like S–R flip-flops, it has two inputs.
- However, in this case all possible combinations of input values are valid.
- Intermediate state of S-R type is defined in J-K flip- flops.

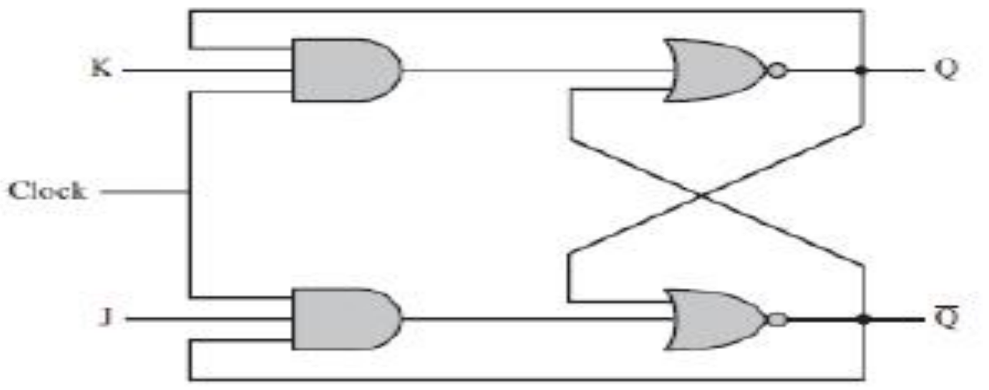


Figure 1.12:J-K Flip-flop

1.6.3 D Flip Flop

- The D(data) flip-flop -data flip-flop uses for storage of one bit of data.
- The output of the D flip-flop is always equal to the *most recent value* applied to the input.
- Hence, it remembers and produces the last input.

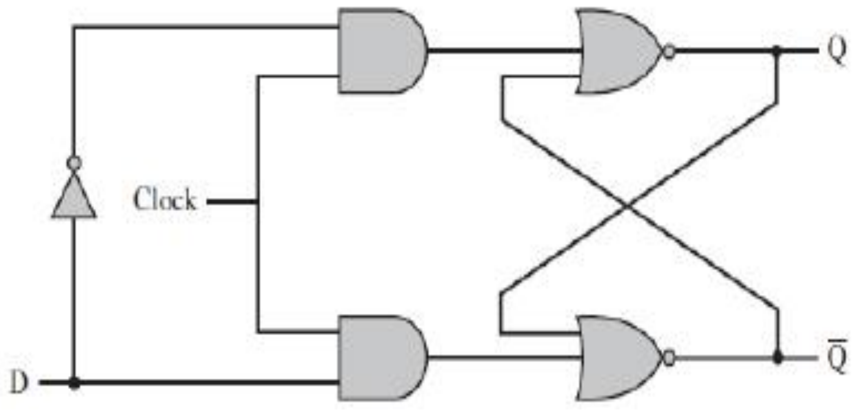


Figure 1.13 D Flip-Flop

Chapter 2

2.1 Number systems

Basically, there are two types of number systems.

I. Non-positional number system: The value of a symbol (digit) in a number does not depend on the position of the digit in number.

II. **Positional number system**: The value of a symbol in the number is determined by its position, the symbol and the base of the number system.

Decimal number system

The decimal number system, also called the base 10 number system, is the number system we use in our day-to-day life. The decimal number system has 10 different symbols identified as 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

Binary number system

The binary number system, also known as base 2 number system, has two digits 0 and 1. The two digits of the binary number system correspond to the two distinct states of the digital electronics.

Data Representation

Data in computers is represented in binary form. The represented data can be number, text, movie, color (picture), sound, or anything else. We enter data into a computer using letters, digits & special symbols. But inside the computer, there is no color, letter, digit or any other character inside the computer system unit. Computers understand and respond to only the flow of electrical charge. They also have storage devices that work based on magnetism. This shows that the overall structure of computers work only in binary conditions (the semi-conductors are conducting or not conducting, a switch is closed or opened, a magnetic spot is magnetized or demagnetized). Hence, data must be represented in the form of binary code that has a corresponding electrical signal. The form of binary data representation system we are seeking is similar to the binary number system in mathematics.

The number systems (bases) we will discuss are: decimal, binary, octal, and hexadecimal

Data representation using the binary number system results in a large string of 0s and 1s. This makes the represented data large and difficult to read. For the sake of writing the binary strings in a short hand form and make them readable, the *octal* and the *hexadecimal* number systems are used.

Octal number system

Octal number system, also called base 8 number system, has 8 different symbols: 0, 1, 2, 3, 4, 5, 6, and 7.

Hexadecimal number system

The hexadecimal number system, also called base 16 number system, has 16 different symbols 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F. It is used to write binary numbers in short form. Memory addresses and MAC addresses are usually written in hex.

2.1.1 Converting from one base to another

I. Conversion from Decimal to Base m

Step 1: Divide the given decimal number by m (the desired base). The result will have a quotient and a remainder.

Step 2: Repeat step 1 until the quotient becomes 0, the quotient is 0 whenever the number < m.

Step 3: Collect and arrange the remainders in such a way that the first remainder is the least significant digit and the last remainder is the most significant digit.

Example: Convert the following decimal number 47 into binary, octal, and hexadecimal.

a. Conversion to binary In order to convert the given decimal numbers into binary (base 2), they are divided by 2.

	Quotient	Remainder
47 ÷ 2	23	1
23 ÷ 2	11	1
11 ÷ 2	5	1
5 ÷ 2	2	1
2 ÷ 2	1	0
1 ÷ 2	0	1

Hence the result is 101111_2.

b. Conversion to octal: Here the numbers are divided by 8 because the required base is octal (base 8).

	Quotient	Remainder
47 ÷ 8	5	7
5 ÷ 8	0	5

c. Conversion to hexadecimal Since the conversion now is into hexadecimal (base 16) the given decimal numbers are divided by 16.

	Quotient	Remainder
47 ÷ 16	2	15
2 ÷ 16	0	2

The hexadecimal equivalent for the decimal 15 is F and that of 2 is 2. Therefore, 47 = $2F_{16}$

II. Conversion from Base m to Decimal

Step 1: Multiply each digit by its positional value.

Step 2: Calculate the sum of the products you get in step 1.

Example 1: Convert the binary number 110001 and 1011.1101 into decimal.

a.$110001_2 = (1 \times 2^5) + (1 \times 2^4) + (0 \times 2^3) + (0 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = (1 \times 32) + (1 \times 16) + (0 \times 8) + (0 \times 4) + (0 \times 2) + (1 \times 1) = 32 + 16 + 0 + 0 + 0 + 1 = 49$

Therefore, $110001_2 = \underline{49}$.

b. $1011.1101_2 =(1 \text{ x } 2^3) + (0 \text{ x } 2^2) + (1 \text{ x } 2^1) \text{ x } (1 \text{ x } 2^0) + (1 \text{ x } 2^{-1}) + (1 \text{ x } 2^{-2}) + (0 \text{ x } 2^{-3}) (+ 1 \text{ x } 2^{-4})$ = (1x 8) + (0 x 4) + (1 x 2) + (1 x1) + (1 x ½) + (1 x ¼) + (0 x1/8) + (1 x 1/16) = 8 + 0 + 2 + 1+ ½ + ¼ + 0 + 1/16 = 11 + 13/16 =11.8

There fore, $1011.1101_2 = \underline{11.8}$.

Example 2: Convert the octal number 22 into decimal.

$22_8 = (2 \times 8^1) + (2 \times 8^0) = (2 \times 8) + (2 \times 1) = 16 + 2 = 18$

Therefore, $22_8 = 18$

Example 3: Convert the hexadecimal number D1 into decimal.

$D1_{16} = (13 \times 16^1) + (1 \times 16^0)$

Therefore, $D1_{16} = \underline{209}$

III. Conversion from Binary to Octal

It is possible to use decimal number system as an intermediate base to convert from any base to any other base. However, for conversion from binary to octal or vice versa, there is a very simple method.

Step 1: Group the binary digits (bits) starting from the rightmost dig into 3 bits. If the remaining bits at the leftmost position are fewer than 3, add 0s at the front.

Step 2: For each 3-bit binary string, find the corresponding octal number.

Example: Convert the binary numbers 110011 and 1101111 to octal.

A. 110011

$$\underset{6}{\underline{110}}\ \underset{3}{\underline{011}}$$

The bits are grouped in three with the equivalent octal digit given below the three bit group. Thus, $110011_2 = 63_8$

B. 1101111

$$\underset{1}{\underline{001}}\ \underset{5}{\underline{101}}\ \underset{7}{\underline{111}}$$

The result is $1101111_2 = 157_8$.

IV. Conversion from Octal to Binary

Step 1: For each octal digit, find the equivalent three digit binary number.

Step 2: If there are leading 0s for the binary equivalent of the leftmost octal digit, remove them.

Example: Find the binary equivalent for the octal numbers 73 and 160.

A. 73

$$\overset{7}{111}\qquad \overset{3}{011}$$

Therefore, $73_8 = 111011_2$

B. 160

$$\overset{1}{001}\qquad \overset{6}{110}\qquad \overset{0}{000}$$

Thus, 1608 = 11100002

From Binary to Hexadecimal

One possible way to convert a binary number to hexadecimal, is first to convert the binary number to decimal and then from decimal to hex. The simple steps states are stated below.

Step 1: Starting from the rightmost bit, group the bits in 4. If the remaining bits at the leftmost position are fewer than 4, add 0s at the front.

Step 2: For each 4-bit group, find the corresponding hexadecimal number.

Example: Convert the binary numbers 1110110001 and 10011110 to hexadecimal.

A. 1110110001

$$\underset{2}{\underline{0011}}\ \underset{B}{\underline{1011}}\ \underset{1}{\underline{0001}}$$

Therefore, $1110110001_2 = 2B1_{16}$

B. 10011110

$$\underset{9}{\underline{1001}}\ \underset{E}{\underline{1110}}$$

Therefore, $10011110_2 = 9E_{16}$

VI. Conversion from Hexadecimal to Binary

Step 1: For each hexadecimal digit, find the equivalent four digit binary number.

Step 2: If there are leading 0s for the binary equivalent of the leftmost hexadecimal digit, remove them.

Example: Find the binary equivalents for the hexadecimal numbers 1C and 823.

A. 1C

1	C
0001	1100

Thus, $1C_{16} = 11100_2$

B. 823

8	2	3
1000	0010	0011

Hence, $823_{16} = 100000100011_2$.

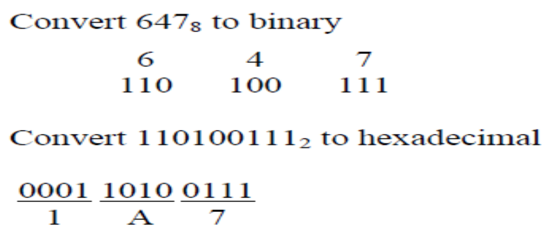
VII. Conversion from Octal to Hexadecimal of Vice Versa

The decimal number system can be used as an intermediate conversion base. second alternative is using the binary number system as an intermediate base.

Step 1: Convert the given number into binary.

Step 2: Convert the binary number you got in step 1 into the required base.

Example : Convert the octal number 647 to hexadecimal.



Convert 647_8 to binary

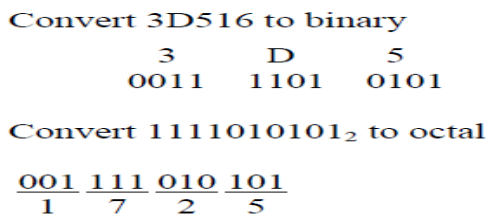
```
      6        4        7
     110      100      111
```

Convert 110100111_2 to hexadecimal

$$\underline{0001}\ \underline{1010}\ \underline{0111}$$
```
  1       A      7
```

Therefore, $647_8 = 1A7_{16}$

Example 2: Find the octal equivalent for the hexadecimal number 3D5



Convert 3D516 to binary

```
       3        D        5
     0011     1101     0101
```

Convert 1111010101_2 to octal

$$\underline{001}\ \underline{111}\ \underline{010}\ \underline{101}$$
```
  1     7     2     5
```

Therefore, $3D5_{16} = 1725_8$

2.2 Representation of Integers (Fixed point Representation)

If the numbers we want to represent are only positive (unsigned) integers, the solution is straightforward; simply represent the unsigned integer with its binary value. For example, 34 is represented as 00100010 in 8 bits.

Signed integer representations are sign magnitude, 1's complement, 2's and complement.

2.2.1 Sign-Magnitude Representation

In mathematics, positive integers are indicated by a preceding + sign (although usually it is avoided) a preceding – sign identify the integer as a negative number. In computers, there is no place for a + or a – sign; there are only 0s and 1s. A similar way of representing + and – signs is to treat the most significant bit as a sign bit the remaining bits are used to represent the magnitude of the integer. By convention a 0 on the sign bit indicates the integer is positive and a 1 indicates the integer is a negative.

As example, the sign-magnitude representation of 79 and -79 in 8 bits are 01001111 and 11001111 respectively.

2.2.2 One’s Complement Integer Representation

Every number system has two complement systems. For a given base n the complements are n’s complement and (n-1)’s complement. Thus, in decimal numbers system (base 10), the complement systems are 10’s complement and 9’s complement. Similarly, in binary number system, the complements are 2’s complement and 1’s complement. Using complements in binary number systems makes subtraction and logical negation very simple. Furthermore, using complements makes arithmetic operations simple. The one’s complement of a binary integer is found by inverting all 0s to 1s and all 1s to 0s. In one’s complement integer representation, the negative of an integer is represented by its complement.

For example, the one’s complement representation of 16 and -16 in 8 bits are 00010000 and 11101111 respectively.

2.2.3 Two’s Complement Integer Representation

The two’s complement of an integer is found by adding 1 to its one’s complement. As a reminder, in binary arithmetic, 0+1 = 1 and 1+1 = 0 with a carry of 1 to the next higher significant bit. A shortcut method to find the two’s complement of a number is to keep all the bits up to and including the first 1 from the right and invert all the others.
Example, two’s complement representation of 19 and -19 in 8 bits are 00010011 and 11101101 respectively.

2.3 Floating Point Numbers

Floating Point representation of a number has two parts. The first part represents a signed, fixed-point number called the mantissa. The second part designates the position of the decimal (or binary) point and is called the exponent. Floating point is always represented as follows:

$\boldsymbol{m \times r^{e}}$ where, m (mantissa) r(radix) and e(exponent)

For example, the decimal, *+6132.789.*

	Fraction	***Exponent***	***Scientific Notation (floating point)***
e.g	*+0.6132789*	*+04*	$+0.6132789 \times 10^{+4}$

Only the mantissa *m* and the exponent *e* are physically represented in the register (including their signs).

e.g + 1001.11

Fraction	***Exponent***
01001110	*000100*

The fraction has 0 in the leftmost position to denote positive the binary point of the fraction follows the sign bit but is not shown in register. The floating- point number is equivalent to

$m \times r^{e} = +(.1001110)_2 \times 2^{+4}$

2.4 Codes

2.4.1 Binary Coded Decimal (BCD)

The BCD (Binary Coded Decimal), also called packed decimal, in order to have representations for the ten digits of the decimal number system, we need a four bit string. Thus 0 = 0000, 1 = 0001, 2 = 0010, …, and 9 = 1001. In BCD, multiples of 8 bits, in which the bits are grouped in 4, are used to represent decimal numbers. Thus the decimal number 461 is represented as 0000 0100 0110 0001.

2.4.2 Characters

Text documents contain strings of characters. Characters refer to letters of the alphabet, the ten digits (0 through 9), punctuation marks, characters that are used to format the layout of text on pages such as the newline, space, and tab characters, and other characters that are useful for communication. The American version of IRA is called American Standard Code for Information Interchange (ASCII). Even though IRA used 8 bits, each character is represented by 7 bits; hence a total of 128 characters are represented. The eighth bit is used for parity (error detection).

Another character encoding system is the EBCDIC (Extended Binary Coded Decimal for Interchange Code). It uses 8 bits per character (and a ninth parity bit), thus represents 256 characters. As with IRA, EBCDIC is compatible with BCD. In the case of EBCDIC, the codes 11110000 through 11111001 represent the digits 0 through 9. ASCII is a standard for use in the United States. Many countries adapted their own versions of ASCII. There are also 8-bit versions of ASCII.

To allow encoding of characters of all the languages in the world, a character set known as the **Unicode** is devised. The Unicode character has variants known as UTF-8, UTF-16, and UTF-32. UTF-8 is the same as ASCII. UTF-16 and UTF-32 use 16-bit and 32-bit per character respectively.

Chapter 3: Common Digital Components

3.1 Integrated Circuits

- Integrated circuit (IC) is the basic building block of digital circuits.
- An integrated circuit is a small silicon semiconductor crystal, called a *chip*.
- The various gates are interconnected inside the chip to form the required circuit.
- As the technology of ICs has improved, the number of gates that can be put in a single chip has increased.
  - ✓ **Small-scale integration (SSI)** devices contain several (usually less than 10) independent gates in a single package.
  - ✓ **Medium-scale integration (MSI)** devices contain approximately 10 to 200 gates in a single package.

    E.g To form decoders, adders, and registers.
  - ✓ **Large-scale integration (LSI)** devices contain between 200 and a few thousands gates in a single package.

    Eg. processors, memory chips, and programmable modules.
  - ✓ **Very-large-scale integration (VLSI)** devices contain thousands of gates in a single package.

  E.g large memory arrays and complex microcomputer chips.

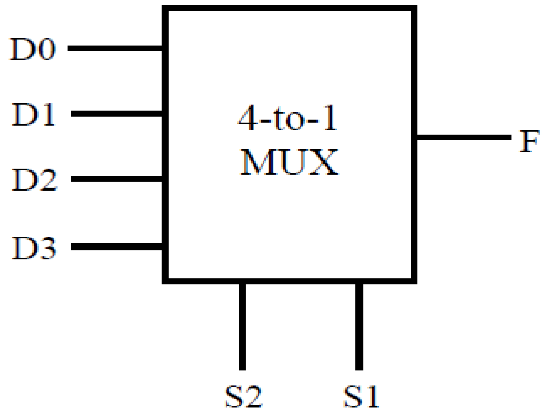
Digital integrated circuits are also classified based on the specific circuit technology to which they belong.

- The basic circuit in each technology is either a NAND, a NOR, or an inverter gate.
- The most popular logic families of integrated circuits are:
  - ✓ TTL Transistor-transistor logic
    - has been in operation for many years and is considered as standard.
  - ✓ ECL Emitter-coupled logic
    - has an advantage in systems requiring high-speed operation.
  - ✓ MOS Metal-oxide semiconductor
    - is suitable for circuits that need high component density.
- CMOS Complementary metal-oxide semiconductor
  - Preferable in systems requiring low power consumption.

3.5 Multiplexers

- Multiplexer is a combinational circuit that receives binary information from one of 2^n input data & directs to one output.
- The multiplexer connects multiple inputs to a single output.
- At any time, one of the inputs is selected to be passed to the output.

  E.g a 4-to-1 multiplexer
- There are four input lines, labelled D0, D1, D2, and D3.
- One of these lines is selected to provide the output signal F.
- To select one of the four possible inputs, a 2-bit selection code is needed, and this is implemented as two select lines S1 and S2.



S2	S1	F
0	0	D0
0	1	D1
1	0	D2
1	1	D3

Figure 3.1: Block diagram Figure 3.2: Truth table of a 4-to-1 multiplexer

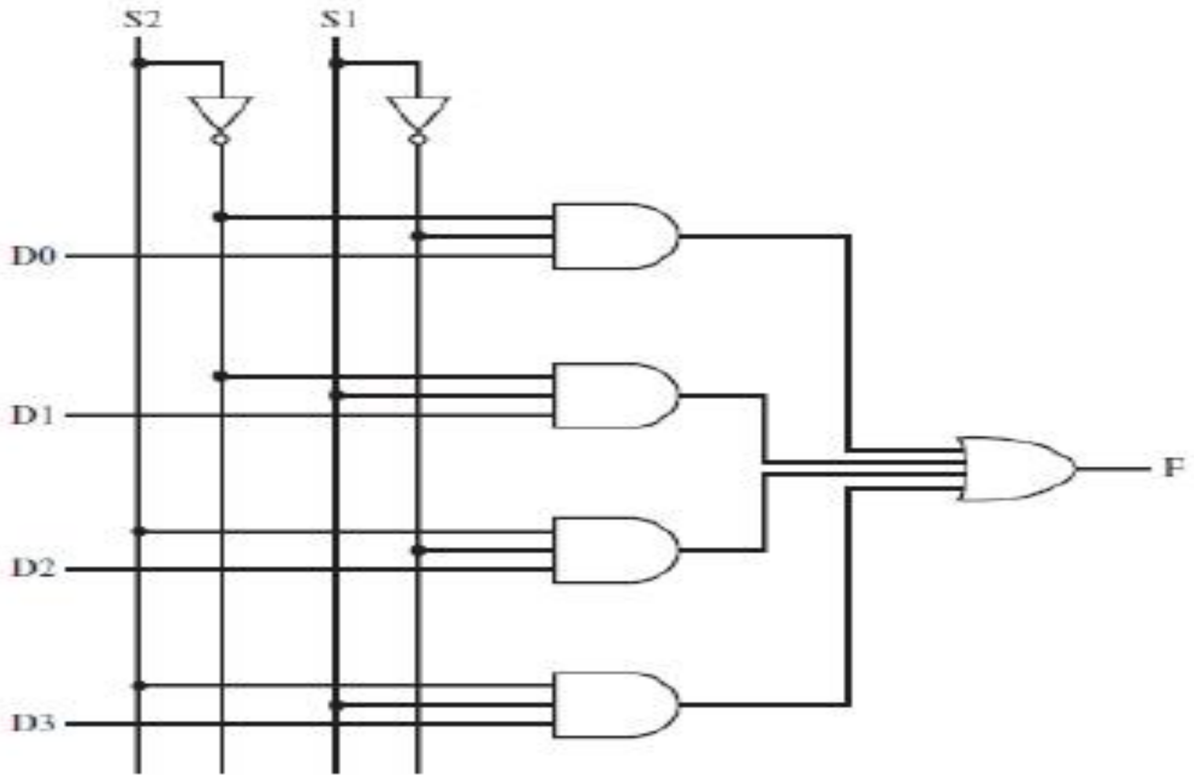


Figure 3.3: An implementation of a 4-to-1 multiplexer using AND, OR, and NOT gates

- Multiplexers are used in digital circuits to *control signal* and *data routing*.
- An example is the loading of the program counter (PC).

3.6 Demultiplexer

- The demultiplexer performs the inverse function of a multiplexer.
- It connects a single input to one of several outputs.

3.7 Decoders

- A decoder is a combinational circuit with a number of output lines, only one of which is selected at any time, depending on the pattern of input lines.
- In general, a decoder has n inputs and 2^n outputs.
- Decoders find many uses in *digital computers*.
- One example is address decoding.
- The other is binary-to- octal conversion.

Decoder with 3 inputs and $2^3 = 8$ outputs

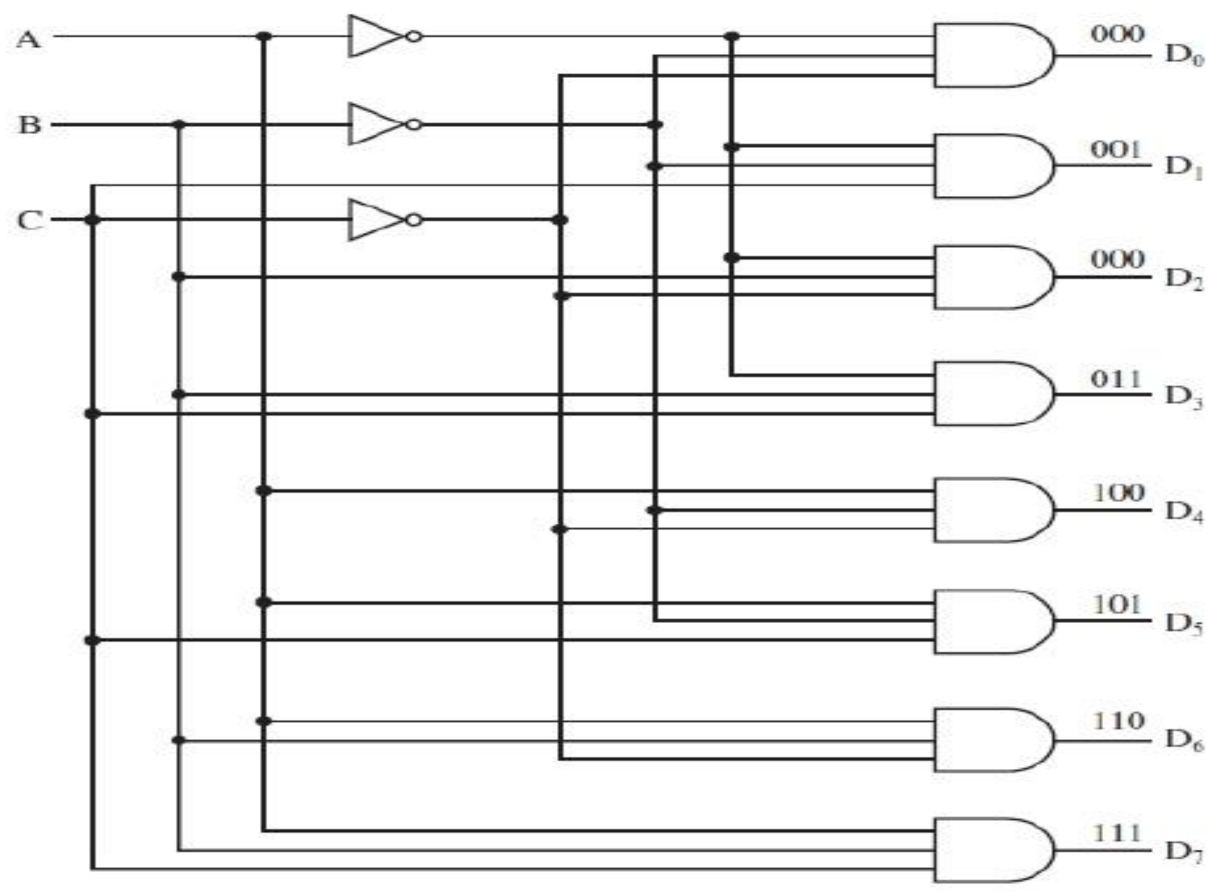


Figure 3.4:3-to 8 line Decoder

3.8 Encoders

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n (or less) input lines and n output lines. The output lines generate the binary code corresponding to the input value.
- An example of an encoder is the octal-to-binary encoder.

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

Figure 3.5: Truth Table Octal to binary Encoder.

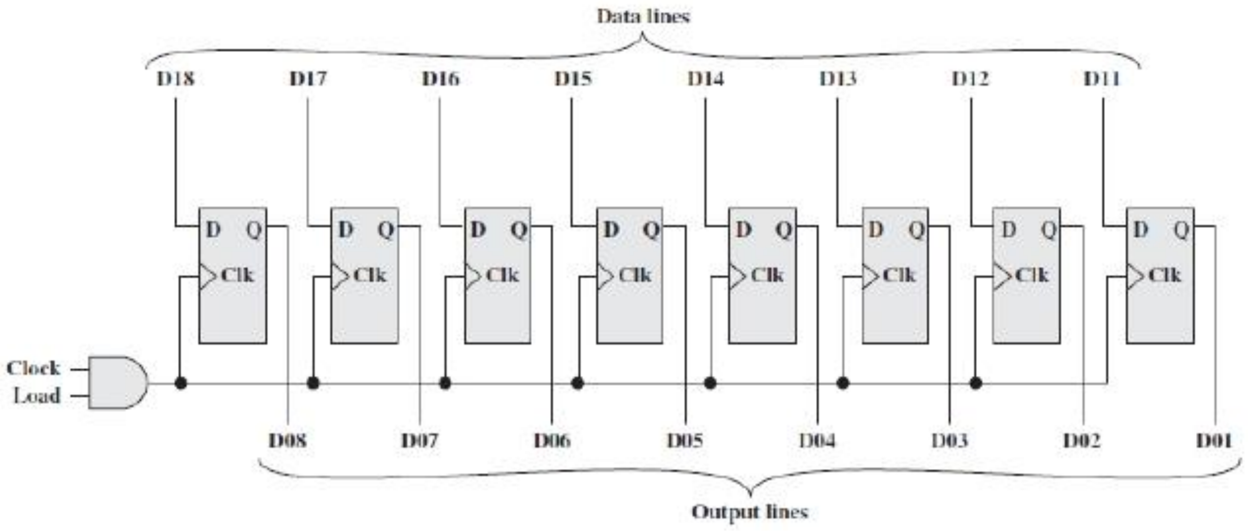
3.9 Registers

- A register is a group of flip-flops with each flip- flop capable of storing one bit of information.
- A register is a digital circuit used within the CPU to store one or more bits of data.
- An n-bit register has a group of n flip-flops and is capable of storing n bits of binary information.
- In addition to Flip-Flops, it may have combinational gates that perform certain data processing.
- Two basic types of registers are commonly used: parallel registers and shift registers.

Parallel Registers

- A parallel register consists of a set of 1-bit memories that can be read or written simultaneously. Is used to store data.

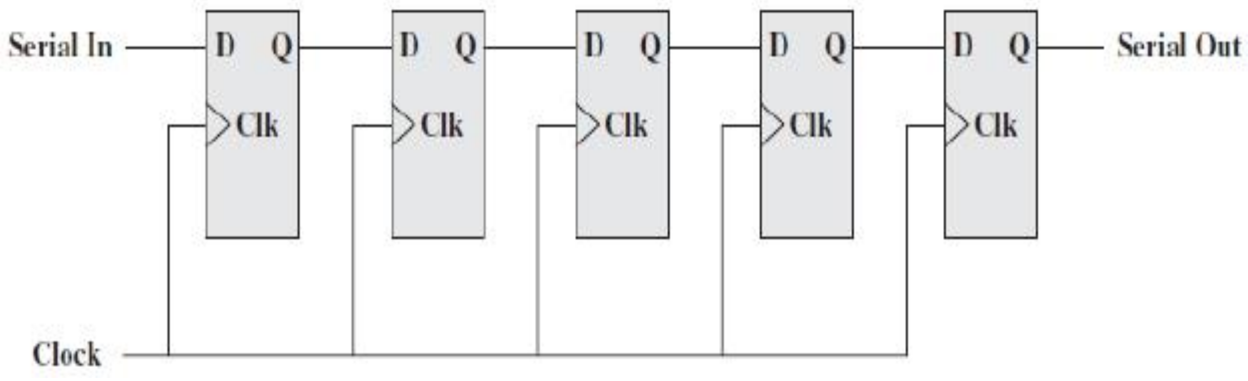
  e.g The 8-bit register- D Flip-Flops



Shift Register

- A shift register accepts and/or transfers information serially.
- Data are input only to the leftmost flip-flop.
- Shift registers can be used to interface to serial I/O devices.
- With each clock pulse, data are shifted to the right, one position, and the rightmost bit is transferred out.
- In addition, they can be used within the ALU to perform logical shift and rotate functions.

5-bit Shift Register



3.10 Binary Counters

- Another useful category of sequential circuit is the counter.
- A counter is a register whose value is easily incremented by 1 modulo the capacity of the register; that is, after the maximum value is achieved the next increment sets the counter value to 0.
- Thus, a register made up of n flip-flops can count up to 2^n-1.
- An example of a counter in the CPU is the program counter.
- Counters can be designated as *asynchronous* or *synchronous*, depending on the way in which they operate.
- Asynchronous counters are relatively slow because the output of one flip-flop causes a change in the status of the next flip-flop.
- In a synchronous counter, all of the flip-flops change state at the same time.
- Because the latter type is much faster, it is the kind used in CPUs.

Chapter 4: Register Transfer and Micro operation

A digital system is an interconnection of digital hardware modules that accomplish a specific information-processing task. Digital systems vary in size and complexity from a few integrated circuits to a complex form of interconnected and interacting digital computers. Digital system design invariably uses a modular approach. The modular are constructed from such digital components as register, decoders, arithmetic elements, and control logic.

Digital modules are best defined by the registers they contain and the operations that are performed on the data stored in them. The operations executed on data stored in registers are called micro operations. A micro operation is an elementary operation performed on the information stored in one or more registers.

Examples of micro operation are shift, count, clear, and load.

Register Transfer Language

The symbolic notation used to describe the micro operation transfers among registers is called a ***register transfer language***. The term “register transfer” implies the availability of hardware logic circuits that can perform a micro operation and transfer the result of operation to some other register. A programming language is a procedure for writing symbols to specify a given computational process.

4.1 Register Transfer

Computer registers are designed by capital letters (some time followed by numerals) to denote the functions of the registers. For example, Memory address register is designated by MAR, PC (program counter), IR (instruction register) and R1 (process register).

The *individual flip-flops* in an n-bit register are numbered in sequence from 0 to n-1, starting from 0 in the rightmost position and increasing toward the left.

Information transfer from one register to another is designed in symbolic form by means of a replacement operator.

Example, **R2←R1**,

Denotes the transfer of the content of register R1 into register R2.

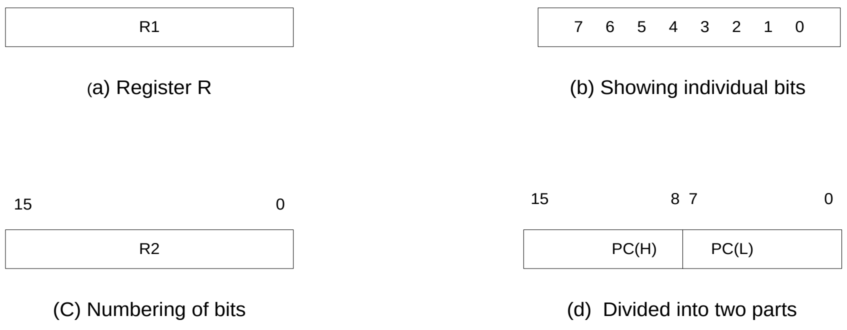


Figure 4.1: Block diagram of registers.

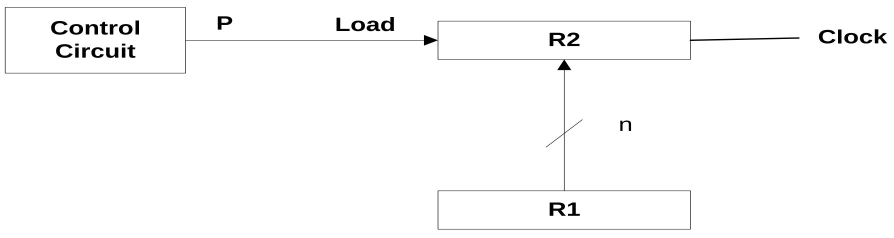
Normally, we want the transfer to occur only under a predefined control condition. This can be shown by means of *if-then* statement.

If (P=1) then (R2←R1)

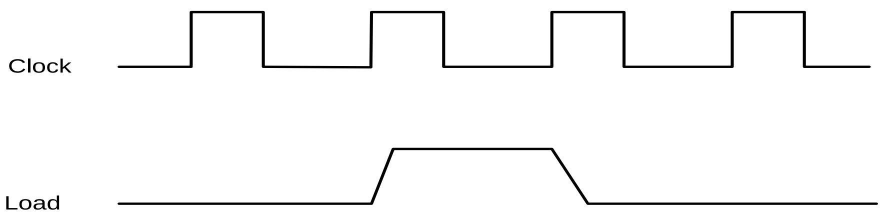
where P is a control signal generated in the control section. But, sometimes it is convenient to separate the control variables from the register transfer operation by specifying a *control function*. A control function is a Boolean variable that is equals to 1or 0.

P: R2←R1

The control condition terminated with a colon and denotes that the transfer operation be executed by the hardware only if P=1. Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.



(a) Block diagram that shows transfer of any n bits from R1 to R2



(b) Timing diagram

Figure 4.2 Shows transfer of content of R1 to R2.

Register R2 has a load input that is activated by the control variable P. It is assumed that the control variable is synchronized with the same clock as the one applied to the register. P is activated in the control section by the rising edge of a clock pulse *t*. The next positive transmission of clock at time *t+1* finds the load input active and the inputs of R2 are then loaded into the register. P may go back to 0 at time *t+1*, otherwise, the transfer will occur with every clock pulse transmission.

A comma is used to separate two or more operations that are executed at the same time.

T: R2←R1, R1←R2, exchanges of the contents of two registers during common clock pulse (**T=1**).

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

Table 4.1: Basic Symbols for Register Transfers

4.2 Bus and Memory Transfers

A typical computer has many registers, and paths must be provided to transfer information from one register to another. An efficient way 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. Control signals determine which register the bus selects during each particular register transfer.

Two ways of constructing, a common bus system is with **multiplexers & three-State Bus Buffers**

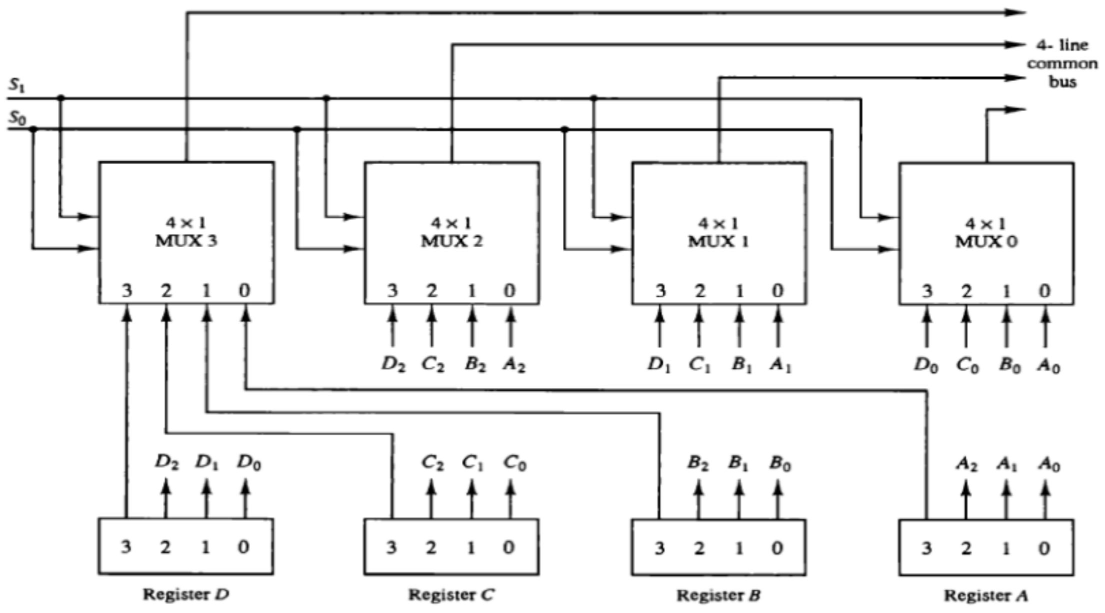


Figure 4.3: Bus system for four registers.

The transfer of information from a bus into one of many destination registers can be accomplished by connecting the bus lines to all of the destination registers. The symbolic statement that includes *bus transfer* is:

BUS ←C,   R1←BUS

The content of register C is placed on the bus, and the content of the bus is loaded into register R1 by activating control input. If the bus is known to exist, only the register transfer can be shown:

R1←C

Three-State Bus Buffers AND, NAND & Buffer gates

The bus system can be constructed with three-state gates instead of multiplexers. A three-state gate is a digital circuit that exhibits three states. Two of the states are signals equivalent to logic 1 and 0 as in a conventional gate. The third state is a high- impedance state. The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have logic significance.

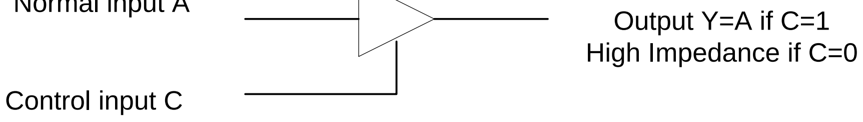


Figure 4.4: Graphic symbols for three-state buffer

To construct a common bus for four registers of n bits each using three state buffers.

4.3 Memory Transfer

The transfer of information from a memory word to the outside environment is called ***read*** operation. The transfer of new information to be stored into the memory is called a ***write*** operation. A memory word is symbolized by **M**. It is important to specify the address of M when writing memory transfer operations.

Read: DR←M[AR]

This causes a transfer of information into DR (Data Register) from the memory word M selected by the address in AR (Address Register).

Write: M[AR]←R1

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

4.4 Arithmetic and Logic Micro operations

4.4.1 Arithmetic Micro operations

The basic arithmetic micro operations are ***addition, subtraction, increment*** and ***decrement***. The arithmetic micro operation defined by the statement

R3←R1 + R2

specifies an add micro operation. It states that contents of register R1 are added to the contents of register R2 and the sum transferred to register R3.

Substruction is most often implemented through ***complementation*** and ***addition***.

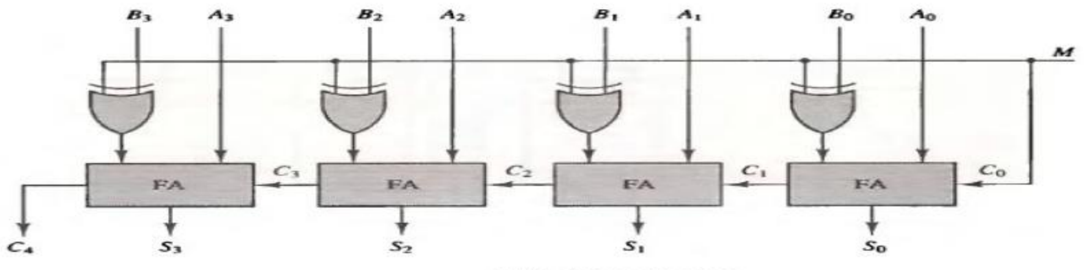
Example, $R3 \leftarrow R1 + \overline{R2} + 1$

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

Symbolic Designation	Description
R3←R1 + R2	Contents of R1 plus R2 transferred to R3
R3←R1 –R2	Contents of R1 minus R2 transferred to R3
R2←$\overline{R2}$	Complement the contents of R2 (1's complement)
R2←$\overline{R2}$ + 1	2's complement the contents of R2 (negative)
R3←R1 +$\overline{R2}$ + 1	R1 plus the 2's complement of R2 (substruction)
R1←R1 + 1	Increment the contents of R1 by one
R1←R1- 1	Decrement the contents of R1 by one

Table 4.2: Arithmetic Micro operation

The increment and decrement micro operations are symbolized by plus one and minus- one operation, respectively.



4-bit adder-subtractor.

4.4.2 Logic micro operations

Logic micro operations specify binary operations for strings of bits stored in registers. These operations consider each bit of the register separately and treat them as a binary variables.

For Example, the exclusive-OR microoperation,

P: R1←R1 $\oplus$ R2

1010	Contents of R1
<u>1100</u>	Contests of R2
0110	Contents of R1 after P=1

The contents of R1, after the execution of the micro operation, is equals to bit-by-bit exclusive- OR operation on pairs of bits in R2 and previous values of R1.

Special symbols will be adopted for the logic micro operation OR (v), AND (∧), and complement to distinguish them from Boolean functions symbols. The complement micro operation is the same as 1's complement and over bar is used.

For example,

P + Q: R1←R2 + R3, R4 V R5

the + between P and Q is an OR operation between two binary variables of a control function. The + between R1 and R2 specifies an add micro operation. The OR micro operation is designated by V between registers R4 and R5

Some Applications

Logic micro operations are very useful for manipulating individual bits or a portion of a word stored in a register. They can be used to change bit values, delete a group of bits, or insert new bit values into a register.
Selective-set: The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B. It does not affect bit positions that have 0's in B.

e.g	1010	A Before
	<u>1100</u>	B (logic operand)
	1110	A after, the corresponding micro operation is A←A V B

Selective-complement: The selective-complement operation complements bits in A where there are corresponding 1's in B. It does not affect bit positions that have 0's in B.

e.g	1010	A before
	<u>1100</u>	B (logic operand)
	0110	A after , the corresponding logic micro operation A←A $\oplus$ B

Selective-clear: The selective-clear operation clears to 0 the bits in A only where there are corresponding 1's in B.

e.g	1010	A before
	<u>1100</u>	B (logic operand)
	0010	A after

The corresponding logic micro operation is A← A ∧ $\bar{B}$

4.5 Shift micro operations

Shift micro operation are used for serial transfer of data. They are also used in conjunction with *arithmetic, logic,* and other *data processing*. The content of a register is shifted to the left or the right. At the same time that the bits are shifted, the first flip-flop receives its binary information from the serial input. During a shift-left operation the serial input transfers a bit into the rightmost position. During a shift-right operation the serial input transfers a bit into the leftmost position. There are three types of shifts: *logical*, *circular,* and *arithmetic*.

A logical shift is one that transfers 0 through the serial input.

Example, **R1←shl R1**

R2←shr R2

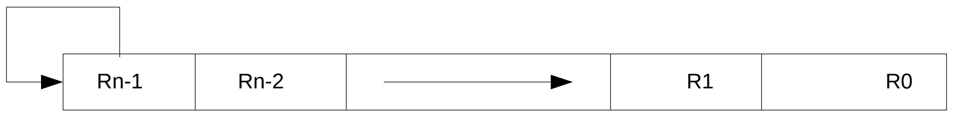
are two micro operations that specifies a 1-bit shift to the left of the content of register R1 and a shift to the right of the content of register R2. The register symbol must be the same on both sides of the arrow.

The *circular shift (also known as a rotate operation)* circulates the bits of the register around the two ends without loss of information. This is accomplished by connecting the serial output of the shift register to its serial input.

Symbolic Destination	Description
R←shl R	Shift-left register R
R←shr R	Shift-right register R
R←cil R	Circular shift-left register R
R←cir R	Circular shift-right register R
R←ashl R	Arithmetic shift-left register R
R←ashr R	Arithmetic shift-right register R

Table 4.3: Shift micro operations

An arithmetic shift is a microoperation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed bit binary number by 2. An arithmetic shift-right divides the number by 2. Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2.



Arithmetic shift right

Sign bit

The leftmost bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is 0 for positive and 1 for negative. Negative numbers are in 2's complement form. Bit R_{n-1} in the leftmost position holds the sign bit. R_{n-2} is the most significant bit of the number and R_0 is the least significant bit. The arithmetic shift-right leaves the sign bit unchanged and shifts the number (including the sign bit) to the right. Thus, R_{n-1} remains the same, R_{n-2} receives the bit from R_{n-1} , and so on for the other bits in the register. The bit in R_0 is lost.

The arithmetic shift-left inserts a 0 into $R_{0,}$ and shifts all other bits to the left. The initial bit of R_{n-1} is lost and replaced by the bit from R_{n-2}. A sign reversal occurs if the bit in R_{n-1} changes in value after the shift. This happens if the multiplication by 2 causes an overflow. An overflow occurs after an arithmetic shift left if initially, before the shift, R_{n-1} is not equal to R_{n-2}. An overflow flip-flop V_s can be used to detect an arithmetic shift-left overflow.

$$\mathbf{V_s = R_{n-1} \oplus R_{n-2}}$$

If Vs=0, there is no overflow, but if Vs=1, there is an overflow and a sign reversal after the shift. V_s must be transferred into the overflow flip-flop with the same clock pulse that shifts the register.

Chapter 5: Basic Computer Organization and Design

5.1 Instruction Codes

The internal organization of a digital system is defined by the sequence of micro-operations it performs on data stored in its registers. The user of the computer can control the process by means of a program. 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. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations.

An **Instruction Code** is a group of bits that instructs the computer to perform a specific operation. It is usually divided into **parts**, each having its own particular interpretation. The most basic part of an instruction code is its **operation part**. The operation code of an instruction is a group of bits that defines such operations as **Add**, **Subtract**, **Multiply**, **Shift** and **Complement**. The operation part of an instruction code specifies the operation to be performed. This operation must be performed on some data stored in **processor registers** or in **memory**. An instruction must specify not only the operation but also the registers or the memory words where the operands are to be found.

- Stored Program Organization

The simplest way to organize a computer is to have *one processor register* and *an instruction code* format with two parts. The first part of instruction code specifies the *operation* to be performed and the second specifies *an address*. The memory address tells the control where to find an operand in memory. This operand is read from memory and used as the data to be operated on together with the data stored in processor register. Instructions are stored in one section of memory and data in another. The operation is performed with memory operands and the content of AC. Figure 5-1 below depicts this type of organization.

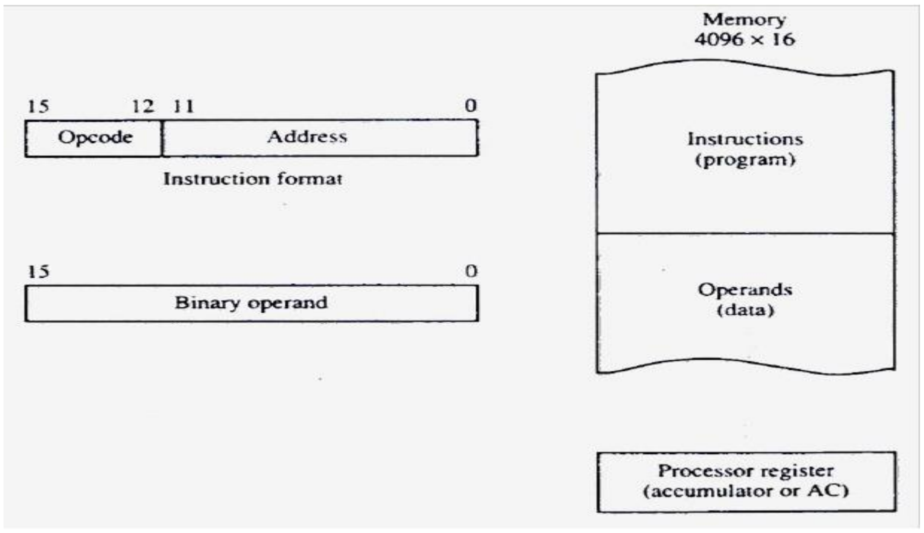


Figure 5-1: Stored program organization

Note: Computers that have a single processor register usually assign to the term Accumulator (AC).

Example: For a memory unit with 4096 words we need 12 bits to specify an address (Since $2^{12} = 4096$). If we store each instruction in one 16 bit memory word, we have available 4-bits for the operation code OP-Code. To specify one out of 16 possible operations and 12 bits specify the address of an operand. The control reads a 16-bit instruction from the program portion of memory. It uses the 12-bit address part of the instruction to read 16-bit operand from the data portion of memory. It then executes the operation specified by the operation code.

5.2 Computer Registers

Computer instructions are usually stored in consecutive memory locations and executed sequentially one at a time. The control reads an instruction from specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it and so on. This type of instruction sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed. It is also necessary to provide a register in a control unit for storing the instruction code after it is read from the memory. The data register (DR) holds the operand read from memory. The accumulator (AC) register is a general purpose processing register. The instructions read from memory are placed in instruction register (IR). The temporary register (TR) is used for holding temporary data during the processing.

Register Symbol	Number of Bits	Register Name	Function
DR	16	Data register	Holds memory operand
AR	12	Address Register	Holds address of 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

Table 5-1: List of registers for the basic computer

The memory address register (AR) has 12 bits since this is the width of a memory address. The program counter (PC) also has 12 bits and it holds the address of the next instruction to read from memory after the current instruction is executed. The PC goes through a counting sequence and causes the computer to read sequential instructions previously stored in memory.

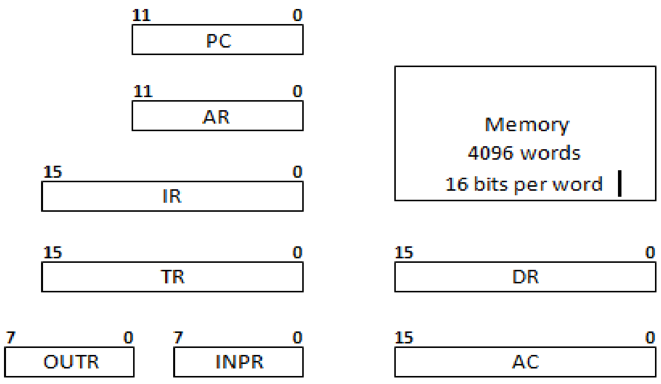


Figure 5-2: Basic Computer Registers and Memory

5.3 Common Bus Systems

The basic computer has eight registers, a memory unit, and a control unit. Paths must be provided to transfer information from one register to another and between memory and registers. Path between each and every register would cause too many wires running around the registers. A better solution is the use of a common bus.

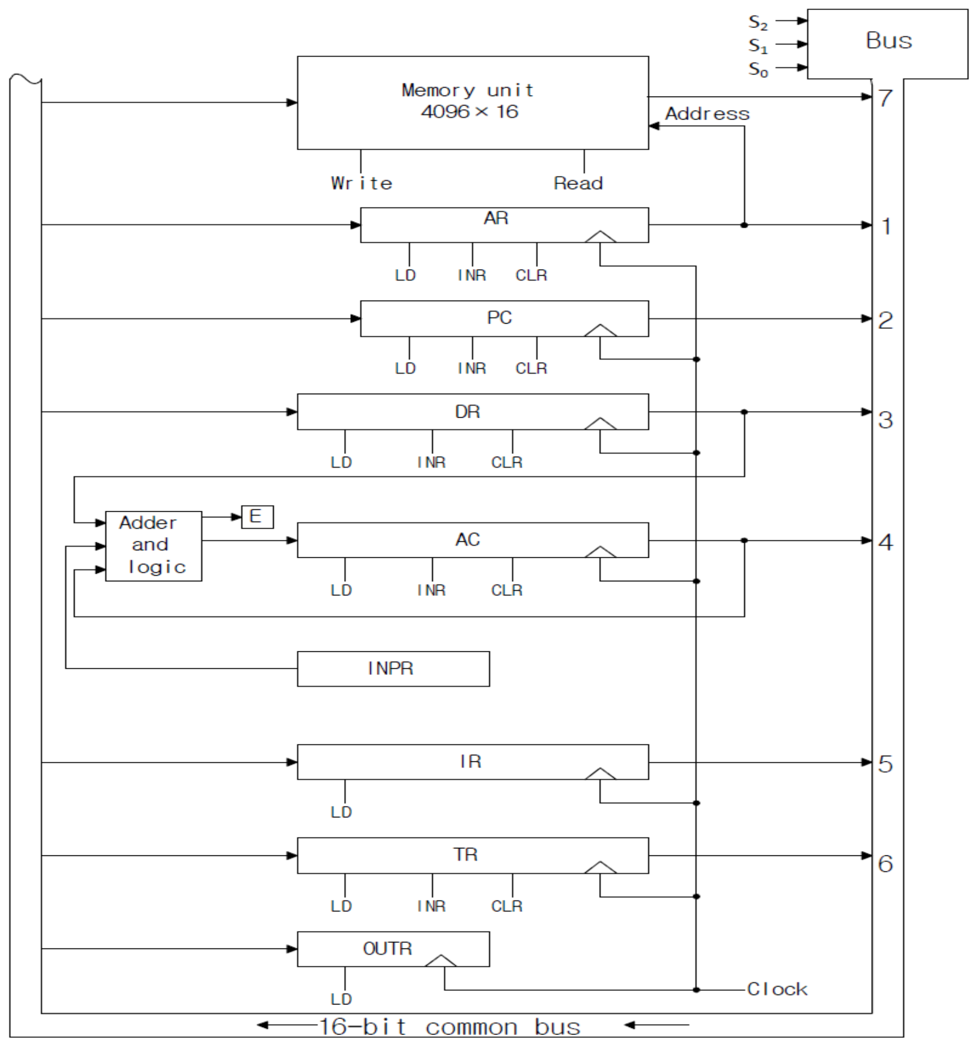


Figure 5-3: Basic computers connected to a common bus

The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S_2, S_1, and S_0. The number along each output shows the decimal equivalent of the required binary selection. For example, the number along the output DR is 3. The 16-bit output of DR are placed on the bus line when $S_2S_1S_0$=011 since this is the binary value of decimal number 3. The lines from the common bus are connected to the inputs of each register and the data inputs of the memory. The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition. The memory receives the content of the bus when its write input is activated. The memory places its 16-bit output onto the bus when the read input is activated and $S_2S_1S_0$=111.

5.4 Computer Instructions

The basic computer has three instruction code formats. Each format has 16 bits. The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining depends on the operation code encountered. A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I (I = 0 for direct address and I = 1 for Indirect address).The register-reference instructions are recognized by the operation code 111 with a 0 in the left most bit (bit 15) of the instruction. A register-reference instruction specifies an operation on or a test of the AC register. Input-output instructions do not need to refer to the memory and are recognized by operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operations or test performed.

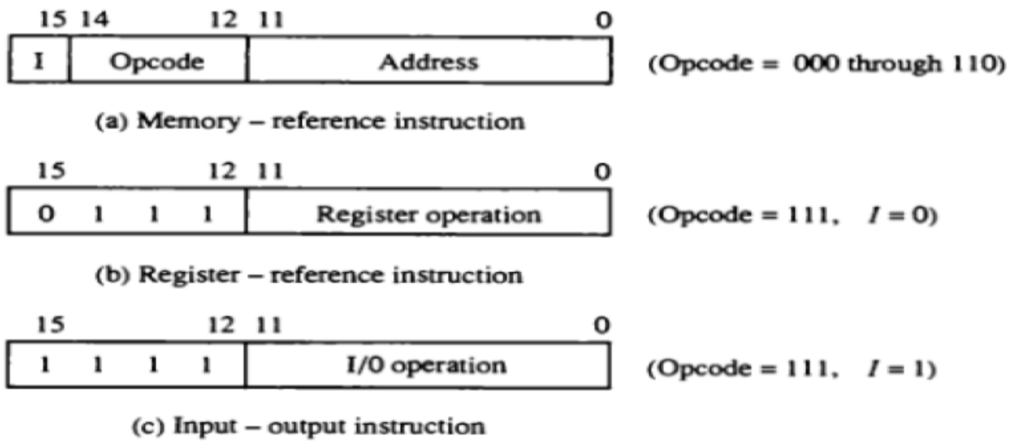
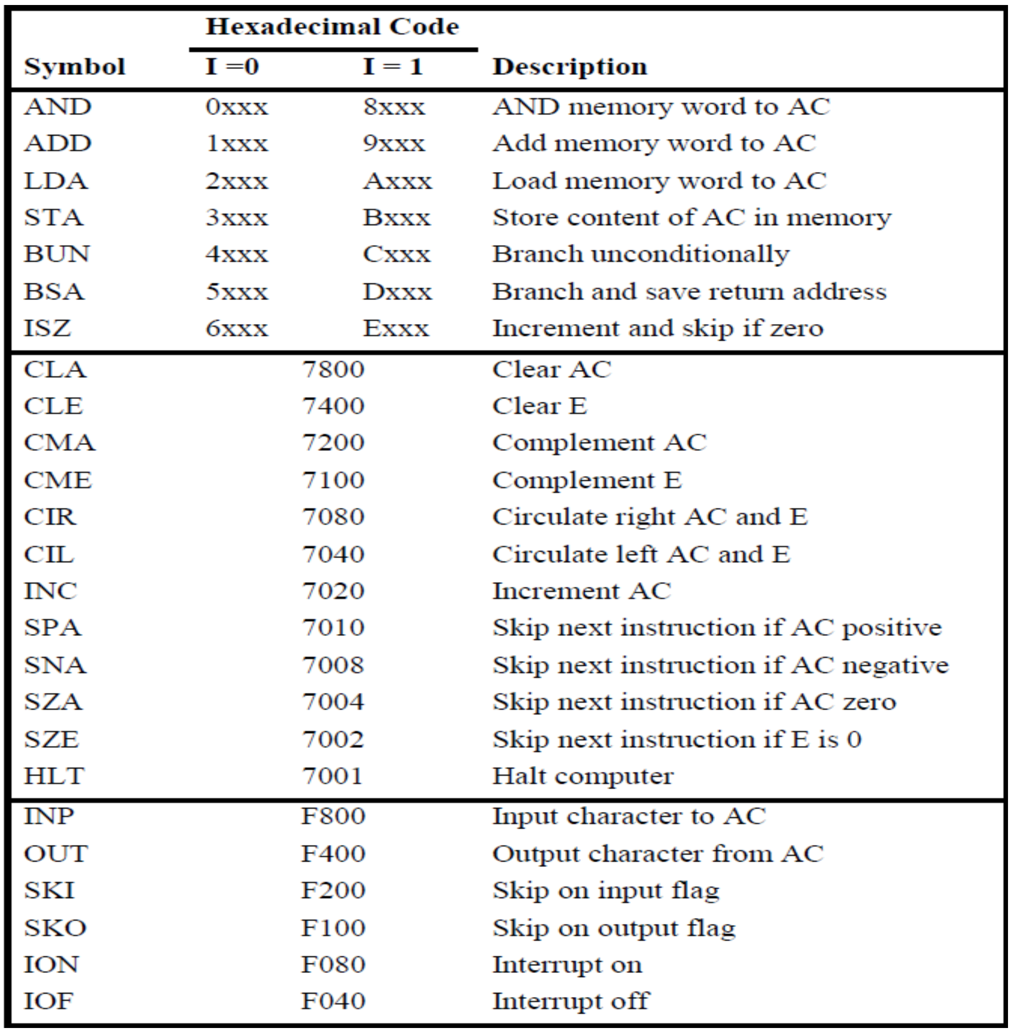


Figure 5-4: Instruction format of basic computer.

Instruction Set Completeness

A computer should have a set of instructions so that the user can construct machine language program to evaluate any function that is known to be computable. The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

1. Arithmetic, Logical and Shift Instructions.
2. Instructions for moving information to and from memory and processor registers.
3. Program control instructions together with instructions that check status conditions.
4. Input and Output instructions.



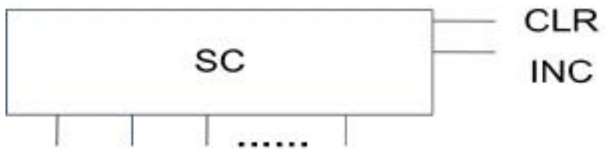
Symbol	Hexadecimal Code I =0	I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Table 5-2: Basic computer instructions

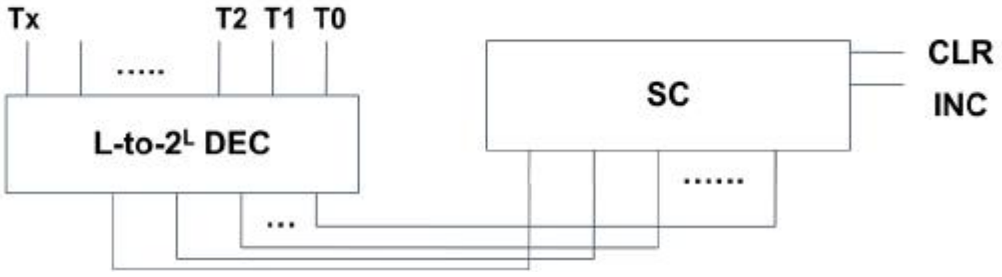
Arithmetic, logical and shift instructions provide computational capabilities for processing the type of data that the user may wish to employ. The bulk of the binary information in a digital computer is stored in memory, but all computations are done in processor registers. Therefore, the user must have the capability of moving information between these two units

5.5. Timing and Control

In order to control the steps of the instruction cycle, it is necessary to introduce a counter, whose output is used as input to the control logic. Sequence Counter Register (SC) is a register that holds a count value, can be reset/cleared to zero and can be incremented (or decremented).



by one. Each instruction will require a specified number of time steps to complete a sequence of micro-operations. Each step of the sequence is marked by a count value in SC. The SC outputs a string of bits whose value is in the range from 0 to $2^{L-}1$. Eg. for L=3, from 0 to 7. The counter is incremented to provide the sequence of timing signals T0, T1, T3….Tx out of the $Lx2^L$ decoder, (where x = $2^{L-}1$).



The timing and control unit is the component that determines what the ALU should do at a given instant. There are two kinds of control organization:
1. Hardwired Control
2. Micro programmed Control

Hardwired control: The control logic is implemented with digital circuits (decoders, flip-flops, etc.). It has the advantage that it can be optimized to produce a fast mode of operation but requires changes in the wiring among the various components if the design has to be modified or changed.
Micro programmed control: Control information is stored in a control memory. Required modifications can be done by updating the micro-program in control memory.

5.6 memory reference instruction

in order to specify he micro operation needed for the execution of each Instructions, it is necessary that the function they are intended to perform be defined precisely. We will know show that the function of the memory reference Instruction can be defined precisely by means of Register Transfer notation .they are a number of memory reference instruction, the Decoded Output Di for i=0,1,2,3.4.5 and 6 for the operation Decoders that belongs to each instruction is described bellow. The actual execution of the instruction in the bus system will require a sequence of micro operation. This is because data stored in the memory cannot processed directly. The data must be read from memory to register where they can be operated on with logical circuit.examples of some basic memory reference Instruction are:

AND to AC

it is an instruction that performs an AND logic operation on pairs of bits.the result of operation is eransferred to AC the micro opration that execuete these Instruction are,

$$\begin{aligned} D_0T_4&: \quad DR \leftarrow M[AR] \\ D_0T_5&: \quad AC \leftarrow AC \wedge DR, \quad SC \leftarrow 0 \end{aligned}$$

ADD to AC

This Instruction adds the content of the memory word specified by the effective address to the value of the AC.

The sum is transferred to the AC and the output carry Cout is transferred to E (Extended Accumulator) flip flop.

The micro operation needed to Execute this Instruction are

$$\begin{aligned} D_1T_4&: \quad DR \leftarrow M[AR] \\ D_1T_5&: \quad AC \leftarrow AC + DR, \quad E \leftarrow C_{out}, \quad SC \leftarrow 0 \end{aligned}$$

LDA Load to AC

this Instruction transfer the memory word specified by the effective address to AC.the micro operation needed to execute this Instruction are

$$\begin{aligned} D_2T_4&: \quad DR \leftarrow M[AR] \\ D_2T_5&: \quad AC \leftarrow DR, \quad SC \leftarrow 0 \end{aligned}$$

At the bus system there is no direct path from the bus into the AC.the adder and logic circuit receive information from DR which can be transferred in to AC.therefore,it is necessary to read the memory word in to DR first and then transfer the content of DR in to AC.

STA Store AC

this Instruction store the content of AC in to the memory word specified by the effective address since the out put of the AC is applied to the bus and the data in put of memory is connected to the bus, we can execute this Instruction with one micro operation.

$$D_3T_4: \quad M[AR] \leftarrow AC, \quad SC \leftarrow 0$$

5.7 Instruction Cycle

A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed by going through a cycle for each instruction; the so called fetch-decode-execute cycle. Each instruction cycle in turn is subdivided into a sequence of phases. In the basic computer each instruction cycle consists of the following phases:

1. Fetch an instruction from memory
2. Decode the instruction
3. Read the effective address from memory if the instruction has an indirect address
4. Execute the instruction

Upon the completion of step 4, the control goes back to step 1 to fetch, decode and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

I. Fetch the instruction
Initially, the program counter PC is loaded with the address of the first instruction in the program and SC is cleared to 0, providing a decoded timing signal T0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2 and so on. Since the address lines of the memory unit are hardwired to AR, address of the instruction to be fetched must first be placed in AR. Thus the first register transfer in an instruction cycle should be transferring the content of PC (address of the instruction to be brought in for execution) to AR
T_0: AR ← PC
Bus selects PC, LD(AR) = 1. Next the current instruction is brought from memory into IR and PC is incremented
T_1: IR ← M[AR], PC ← PC + 1. Bus selects RAM, Memory read is set, LD(IR) = 1, INR(PC) = 1
II. Decode the instruction
T_2: D_0, ... , D_7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)
Therefore, 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: D_0, …, D_7 ← Decode IR(12-14), AR ← IR(0-11), I ← IR(15)

5.8 Input-Output

A computer instruction can serve no useful purpose unless it communicates with the external environment. Instructions and data stored in memory must come from some input device. Computational results must be transmitted to the user through some output device.
Input-Output Configuration
The terminal sends and receives serial information. Each quantity of information has eight bits of an alphanumeric code. The serial information from the keyboard is shifted into the input register INPR. The serial information for the printer is stored in the output register OUTR. These two register communicate with a communication interface serially and with the AC in parallel.

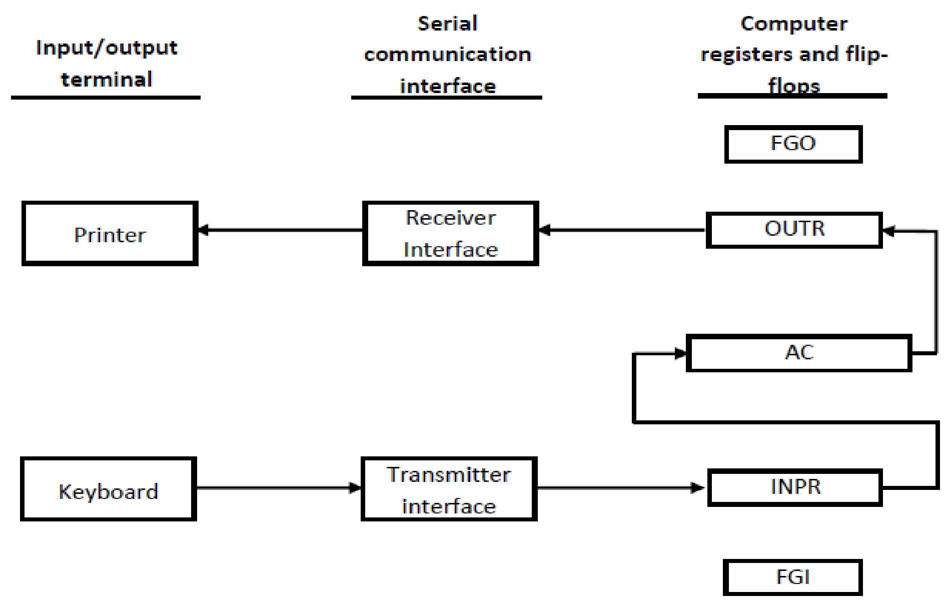


Figure 5.5: Input-Output Configurations

The input register INPR consists of eight bits and holds alphanumeric input information. The 1-bit input flag FGI is a control flip-flop. The flag is needed to synchronize the timing rate difference between the input devices and the computer. Initially, the input flag (FGI) is cleared to 0. When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and FGI is set to 1. The FGI is cleared to 0 when the information is accepted by the computer.

The output register OUTR works similarly but the direction of information flow is reversed. Initially, FGO is set to 1. The computer checks the flag bit, if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The output device accepts the coded information, prints the corresponding character and when the operation is completed, it sets FGO to 1.

5.9 Design of Basic Computer

The basic computer consists of the following hardware components;

- A memory unit with 4096 words of 16-bits each
- Nine registers: AR, PC, DR, AC, IR, TR, OUTR, INPR, and SC
- Seven flip-flops: I, S, E, R, IEN, FGI, and FGO (to hold 1-bit of information)
- Two decoders: a 3x8 operation decoder and a 4x16 timing decoder
- A 16-bit common bus
- Control logic gates
- Adder and logic circuit connected to the input of AC.

5.10 Design of accumulator logic

The adder and logic circuit has three sets of input. One set of 16 input comes from output of AC another set of 16 input comes from the data register DR a third set of input eight input comes from the input register INPR the output of the adder and logic circuit provides the data input for the register, in addition, it is necessary to include logic gate controlling the LD,INR and CLR in the register and for controlling the operation of the adder and logic circuit.in order to design the logic associated with AC,it is necessary to go over the register transfer statements

5.11. Central Processing Unit.

The part of the computer that performs the bulk of data processing operations is called the Central processing unit and is referred to as the CPU. The CPU is made up of three major parts The register set stores intermediate data used during the execution of the instructions. The Arithmetic and Logic unit (ALU) performs the required micro-operations for executing the instructions. The control unit supervises the transfer of information among the registers and instructs the arithmetic and logic units as to which operation to perform. The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer. Computer architecture is sometimes defined as the computer structure and behavior as seen by the programmer that uses machine language instructions. This includes the instruction formats, addressing modes, the instruction sets and the general organizations of the CPU registers

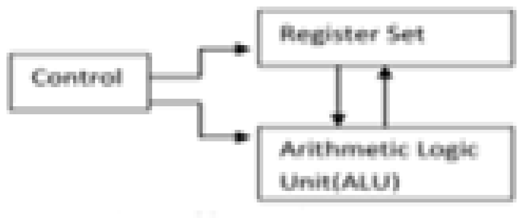


Figure5.6: major components of CPU

5.12 General Register Organizations .

In the programming examples of the previous chapter, we have shown that memory locations are needed for storing pointers, counters, return addresses, temporary results and partial product during multiplication. 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 more efficient to store these intermediate values in processor register. When a large number of registers are included in the CPU, It is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfers, but also while performing various micro operations.

A bus organization for 7-CPU registers is shown Figure 5-2. The output of each register is connected to 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 busses form the inputs to common arithmetic logic unit.

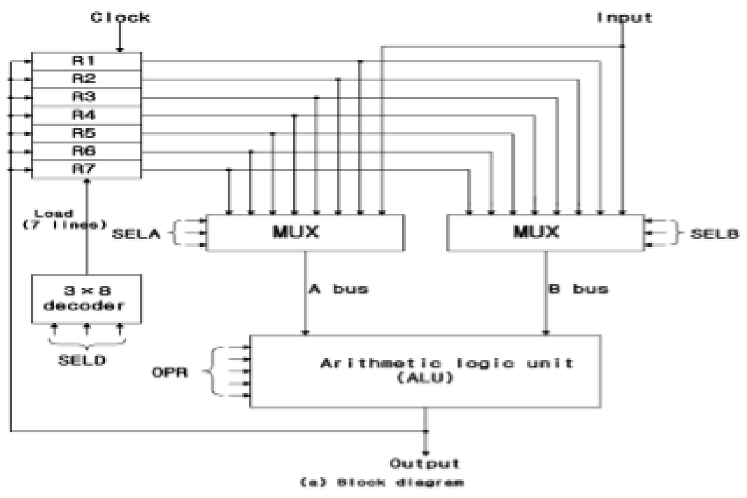
The operation selected in the ALU determines the arithmetic or Logic micro-operation that is to be performed. The result of micro-operation is available for output data and also goes in to the inputs of all the registers. The register that receives the information from the output bus is selected by a 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 register and ALU by selecting the various components in the system. For example, to perform the operation

R1 R2 + R3, the control must provide binary selection variables to the following selector inputs:

1. MUX A selector (SELA): to place the content of R2 into bus A.
2. MUX selector (SELB): to place the content of R3 into bus B.
3. ALU operation selector (OPR): to provide the arithmetic add A +B.
4. Decoder destination selector (SELD): to transfer the content of output bus in to R1.

The four control selection variables are generated in the control unit and must be available at the beginning of a clock cycle. The data from the two source registers propagates through the gates in the multiplexers and the ALU, to the output bus, and into the inputs of the destination register, all during the clock cycle interval. Then, when the next clock transition occurs, the binary information from the output bus is transferred into R1. To achieve a fast response time, the ALU is constructed with high speed circuits. The busses are implemented with multiplexers or three state gates.



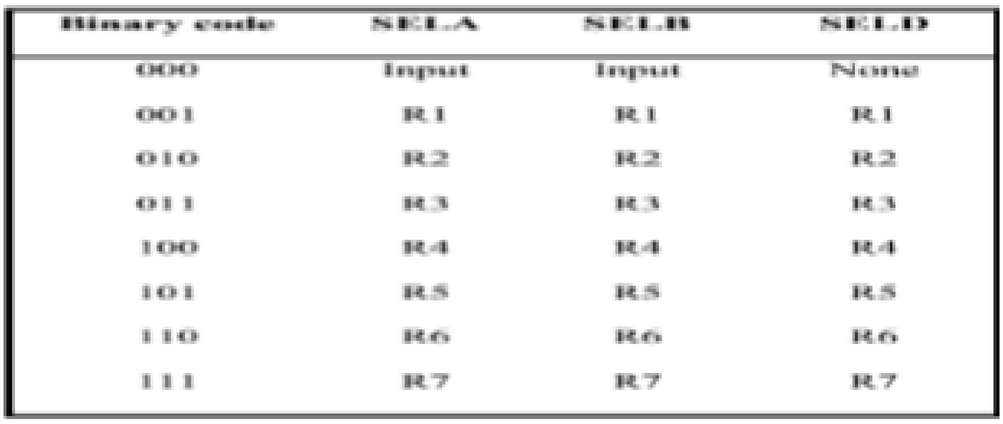
(a) Block diagram

3	3	3	5
SELA	SELB	SELD	OPR

(b) Control word

5.13 Control Word

There are 14 binary selection inputs in the unit, and their combined value specifies a control word. The 14-bit control word is illustrated in Figure 5-2 (b). It consists of four fields. Three fields contain three bits each, and one field has five bits. The three bits SELA select a source register for the A input of the ALU. The three bits of SELB select a register for the B input of the ALU. The Three bits of SELD selects a destination register using the decoder and its seven load outputs. The five bits of the OPR select one of the operations in the ALU. The 14-bit control word when applied to the selection inputs specify a particular micro operation. The encoding of the register selection is specified in the following Table.



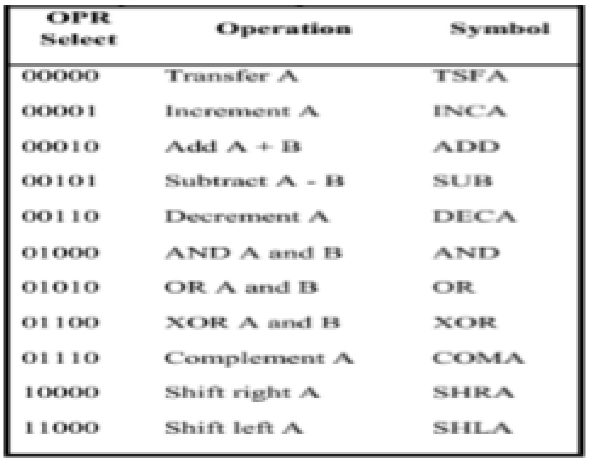
Binary code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5
110	R6	R6	R6
111	R7	R7	R7

Table 5.3 Encoding of Register selection Fields

The 3-bit binary code listed in the first column of Table 5-1 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.

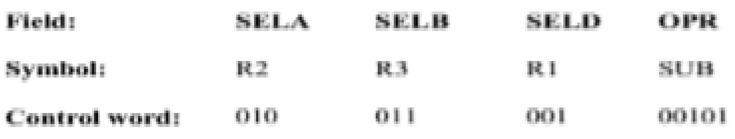
The ALU provides arithmetic and logic operation. In addition, the CPU must provide shift operations. The shifter may be placed in the input of the ALU to provide a pre-shift capability, or at the output of the ALU to provide post-shifting capability. In some cases, the shift operations are included with the ALU.

A control word of 14-bits is needed to specify a micro-operation in the CPU. The control word for a given micro-operation can be derived from the selection variables. For example, the subtract micro operation given by the statement R ← R2 - R3 specifies R2 for the input A of the ALU, R3 for the B inputs of the ALU, R1 for the destination register, and an ALU operation to subtract A - B. Thus, the control word is specified by the four fields and the corresponding binary value for each field is obtained from the encoding list in Table 5-1 and Table 5-2. The binary control word for the subtraction micro operation is 010 011 001 00101 and is obtained as follows:



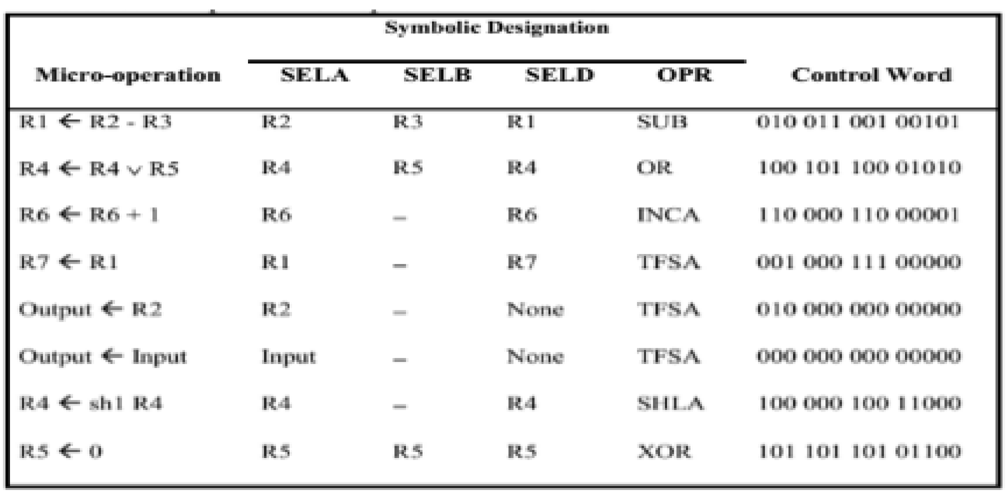
OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Table 5.4 Examples of micro operation



Field:	**SELA**	**SELB**	**SELD**	**OPR**
Symbol:	R2	R3	R1	SUB
Control word:	010	011	001	00101

The control word for this micro-operation and a few others are listed in Table 5-3. The increment and transfer micro-operations do not use the B input of the ALU. For these cases, the B field is marked with a dash.



	Symbolic Designation				
Micro-operation	SELA	SELB	SELD	OPR	Control Word
R1 ← R2 - R3	R2	R3	R1	SUB	010 011 001 00101
R4 ← R4 ∨ R5	R4	R5	R4	OR	100 101 100 01010
R6 ← R6 + 1	R6	–	R6	INCA	110 000 110 00001
R7 ← R1	R1	–	R7	TFSA	001 000 111 00000
Output ← R2	R2	–	None	TFSA	010 000 000 00000
Output ← Input	Input	–	None	TFSA	000 000 000 00000
R4 ← shl R4	R4	–	R4	SHLA	100 000 100 11000
R5 ← 0	R5	R5	R5	XOR	101 101 101 01100

Table 5.5 examples of micro operation for the cpu

5.14 Stack Organization.

A useful feature that is included in the CPU of most computers is a stack or last- in, first-out (LIFO) list. A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. The stack in digital computers is essentially a memory unit with an address register that can count only (after an initial value is loaded into it). The register that holds the address for the stack is called a stack pointer (SP) because its value always points at the top in the stack.

The two operations of a stack are the insertion and deletion of items. The operation of the insertion is called PUSH (or push-down) because it can be thought of as the result of pushing a new item on top. The operation of deletion is called POP (or pop-up) because it can be thought as the result of removing one item so that the stack pops up. However, nothing is pushed or popped in a computer stack. In computers, these operations are simulated by incrementing or decrementing the stack pointer register.

5.14.1 Register Stack.

A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure 5-6 shows the organizations of a 64-word register stack. The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in the stack: A, B and C in that order. Item C is on 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 on 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. Note that item C has been read out but not physically removed. This does not matter because when the stack is pushed, a new item is written in its place.

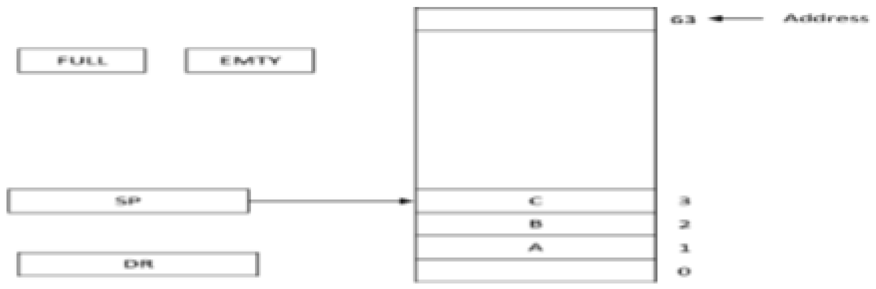


Figure 5.6 block diagram of 64 word stack

In a 64-word stack, the stack pointer contains 6-bits because 26 = 64. Since SP has only 6-bits it cannot exceed a number greater than 63 (111111 in binary). When 63 is incremented by 1, the result is 0 since 111111+ 1 = 1000000 in binary, but SP can accommodate only the 6 list significant bits. Similarly, when 000000 is decremented by 1, the result is 111111. The 1-bit register FULL is set to 1 when the stack is full, and the 1-bit register EMTY is set to 1 when the stack is empty of items. DR is the data register that holds the binary data to be written in to or read out of the stack.

Initially, SP is cleared to 0, EMTY is set to 1 and FULL is cleared to 0, so that points to the word at address 0 and the stack is marked empty and not full. If the stack is not full (if FULL = 0), a new item is inserted with the PUSH operation. The PUSH operation is implemented with the following sequence of micro-operations SP SP + 1 Increment stack pointer M[SP] DR Write item on top of the stack If (SP = 0) then (FULL 1) Check if stack is full EMTY 0 Mark the stack not empty

The stack pointer is incremented so that it points to the address the next higher word. A memory write operation inserts the word from DR in to the top of the stack. Note that SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available in SP. 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. This condition is reached if the top item prior to the last push was in location 63 and after incrementing SP, the last item is stored in location 0. Once an item is stored in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0.

A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequences of microoperations: DR M[SP] Read item from the top of the stack SP SP - 1 Decrement stack pointer If (SP = 0) then (EMTY 1) Check if stack is empty FULL 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, the stack is empty, so EMTY is set to 1. This condition is reached if the item read was in location 1. Once this item is read out, SP is decremented and reaches the value of 0, which is the initial value of SP. Note that if a pop operation reads the

item from location 0 and then SP is decremented, SP changes to 111111, which is equivalent to decimal 63. In this configuration the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL=1 or popped when EMTY=1.

5.14.2 Memory Stack

A stack can exist as standalone unit or can be implemented in random access memory attached to a CPU. The implementation of a stack in the CPU is done by assigning a portion of memory to stack operation and using a processor register as a stack pointer. Figure 5-7 shows a portion of computer memory partitioned into three segments: Program, Data, and Stack.

The program counter PC points at the address of the next instruction in the program. The address register ER points at an array of data. The stack pointer SP points at the top of the stack. The three registers are connected to a common address bus, and either one can provide an address for memory. PC is used during the fetch phase to read an instruction. AR is used during the execute phase to read an operand. SP is used to push or POP items into or from the stack.

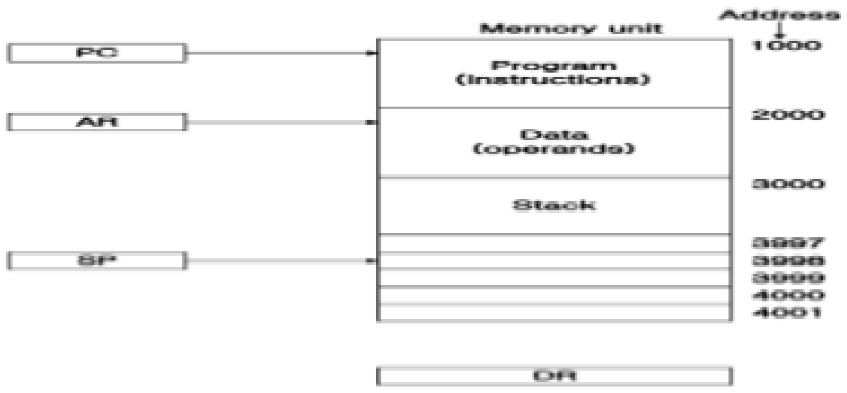


Figure5.7 computer memory

the initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack is 3000. No provisions are available for stack limit checks. We assume that the items in the stack communicate with a data register DR. A new item is inserted with the push operation as follows: SP ← SP - 1 M[SP] DR

The stack pointer is decremented so that it points at the address of the next word. A memory write operation inserts the word from DR into the top of the stack. A new item is deleted with the pop operation as follows:

DR M[SP] SP SP + 1

The top item is read from the stack into DR. The stack pointer is then incremented to point at the next item in the stack.

Most computers do not provide hardware to check for stack overflow (full stack) or underflow (empty stack). The stack limits can be checked by using to processor registers: one to hold the upper limit (3000 in our example), and the other to hold the lower limit (4001 in the example). After a push operation, SP is compared with the upper limit register and after a pop operation SP is compared with the lower limit register. The two micro-operations

needed for either the push or pop are: 1. An access to memory through SP and 2. Updating SP.

5.15 Instruction Formats

A computer will usually have a variety of instruction code formats. The bits of the instruction are divided into groups called Fields. The most common fields found in instruction formats are: 1. An operation code field that specifies the operation to be performed 2. An address field that designate a memory address or a processor register 3. A mode field that specifies the way the operand or the effective address is determined

Computers may have instruction of several lengths containing varying number of addresses. The number of address field in the instruction format of a computer depends on the internal organization of its registers.

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 Most computers fall into one of three types CPU organizations. 1. Single accumulator organization 2. General register organization 3. Stack organization

In an accumulator-type organization, all operations are performed with an implied accumulator register. The instruction format in this type of computer uses one address field. For example, the instruction that specifies an arithmetic addition is defined by an assembly language:

ADD X, where X is the address of the operand.

Which results AC ←AC + M[X]

AC is the accumulator register and M[X] symbolizes the memory word located at address X.

The instruction format in a computer with a general register organization type needs two or three register address fields.

e.g ADD R1, R2, R3   R1←R2 + R3.

The number of address field in the instruction can be reduced from three to two if the destination register is the same as one of the source register.

e.g ADD R1, R2 denotes R1←R1 + R2. Only register addresses for R1 and R2 need be specified in this instruction.

e.g ADD R1, X denotes R1←R1 + M[x]. It has two address fields, one for register R1 and the other for the memory address X.

Computers with multiple processor registers use the move instruction with a mnemonics MOV to symbolize a transfer instruction.

e.g MOV R1, R2 denotes 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.

Computers with stack-organization would have PUSH and POP instructions which require an address field.

e.g PUSH X will push the word at address X to the top of the stack.

Operation-type (e.g ADD) 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.

symbols ADD, SUB, MUL and DIV for four arithmetic operations; and 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.

5.15.1 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 shown below, together with comments that explain the register transfer operation of each instruction.

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 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.

5.15.2 Two- Address Register

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 shown 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.

5.15.3 One-Address Instruction

One-address instructions use an implied accumulator (AC) register for all data manipulations. 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.

5.15.4 Zero-Address Instruction

A stack-organized computer does not use an address field for 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.

5.16 Addressing Mode

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 give 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 mode to be presented in this section, it is important 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. 1. Fetch the instruction from memory 2. Decode the instruction 3. Execute the instruction

There is one register in the computer called the program counter (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 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 operand.

5.16.1 Direct and Indirect Addressing Modes

It is sometimes convenient to use the address bits of an instruction code not as an address but as an actual operand. When the second part of an instruction code specifies an operand, the instruction is said to have an **immediate operand**.

When the second part specifies the address of an operand, the instruction is said to have **direct address**. This is in contrast to a third possibility called **indirect address**, where the bits in the second part of the instruction designate an address of memory word in which the address of the operand is found.

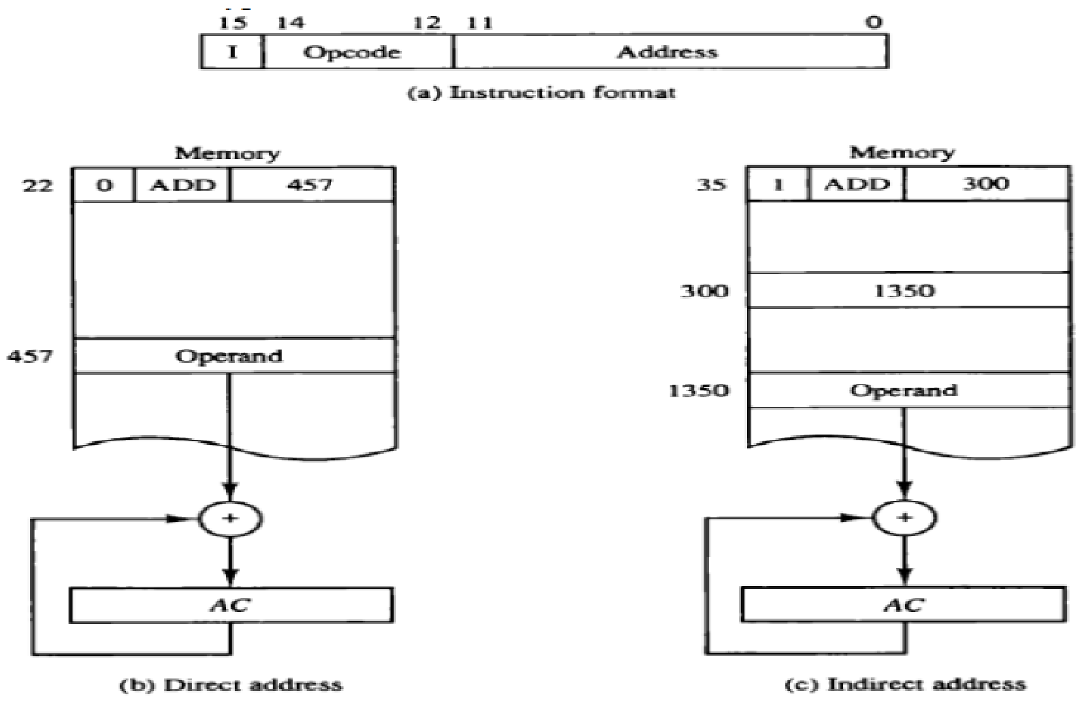


Figure 5.8: Direct and indirect addressing modes

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 stack organized are implied mode instructions since the operands are implied to be on top of the stack.

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.

When the address field specifies a processor register, the instruction is said to be in the register mode.

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 of the 2k registers.

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 the 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.

Autoincrement and Autodecrement 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 a data in memory, it is necessary to increment or decrement the register after every access to the table. The address field of an instruction is used by the control unit in CPU to obtain the operand from memory.

Direct Addressing 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 of instruction the address field specifies the actual branch address.

Indirect Addressing 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.

Relative Addressing mode: In this mode the content of 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 negative or positive.

5.17 Data Transfer and Manipulation.

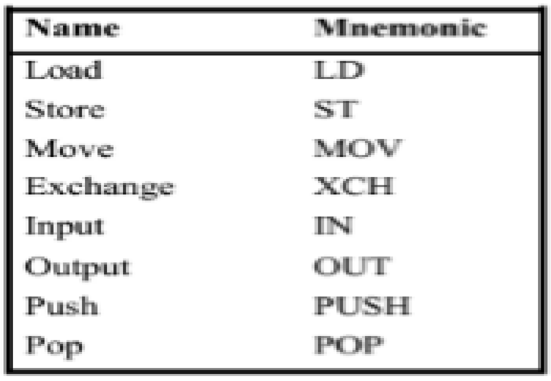
Computers provide an extensive set of instruction to give the user the flexibility to carry out various computational tasks. The instruction set of different computers differ from each other mostly in the way the operands are determined from the address and mode fields. The actual operations available in the instruction set are not very different from one computer to another. It so happens that the binary code assignments in the operation code field is different in different computers, even for the same operation. It may also happen that the

symbolic name given to instructions in the assembly language notation is different in different computers, even for the same instruction. Most computer instructions can be classified in to three categories.

1. Data transfer instruction.
2. Data manipulation instruction
3. Program control instruction

Data transfer instruction.

Data transfer instructions move data from place to place in the computer to another without changing the data content. The most common transfers are between memory and processor registers, between processor register and input or output, and between the processor registers themselves. Table 5-5 gives lists of eight data transfer instructions used in many computers. Accompanying each instruction is mnemonic symbol.



Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Table 5.6 Lists of data transfer Instruction

Data Manipulation

Data manipulation instructions perform operations on data and provide the computational capabilities for the computer. The data manipulation instructions in a typical computer are usually divided into three basic types.

1. Arithmetic instruction

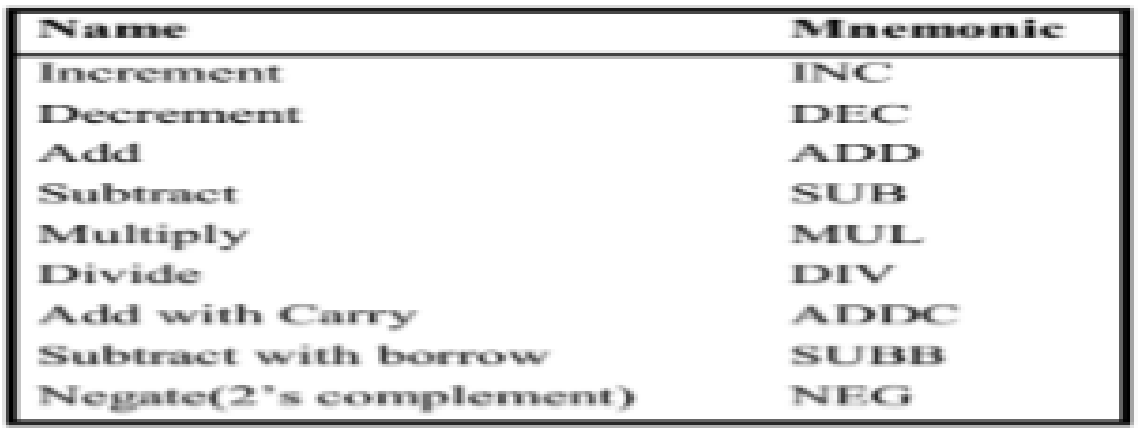
2. Logical and bit manipulation

3. Shift instruction

a. Arithmetic Instruction

The four basic arithmetic operations are addition, subtraction, multiplication and division. Most computers provide instructions for all four operations. Some small computers have only addition and possibly subtraction instructions. The multiplication and division must then be generated by means of software subroutines.

The four basic arithmetic operations are sufficient for formulating solution to scientific problems. when expressed in terms of numeri analysis methods.

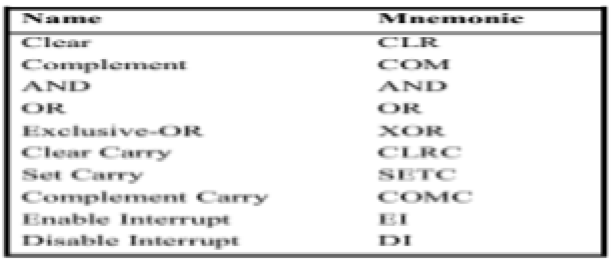


Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with borrow	SUBB
Negate(2's complement)	NEG

Table 5.7 Basic Arithmetic operations

b. Logical and Bit manipulation Instructions

Logical instructions perform binary operations on strings of bits stored in registers. They are useful for manipulating individual bits or a group of bits that represent binary–coded information. The logical instructions consider each bit of the operands separately and treat it as a Boolean variable. By proper application of the logical instructions it is possible to change bit values, to clear a group of bits or to insert new bit values into the operands stored in registers or memory words. Some logical and bit manipulation instructions are listed in Table 5-8 as follows.

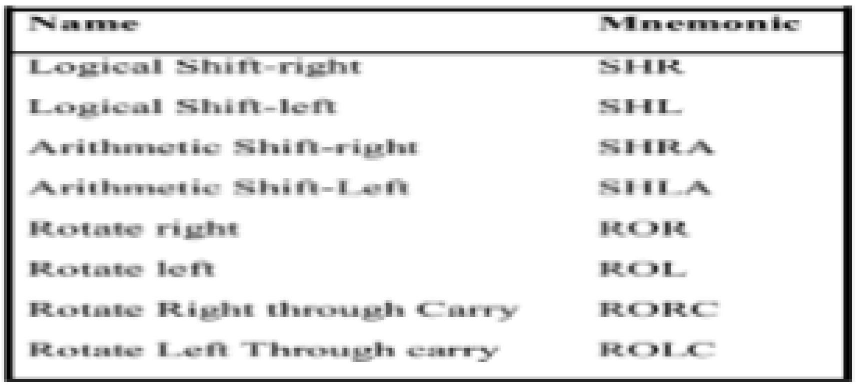


Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear Carry	CLRC
Set Carry	SETC
Complement Carry	COMC
Enable Interrupt	EI
Disable Interrupt	DI

Table 5-8 Some logical and bit manipulation instructions

c. Shift Instruction

Instructions to shift the content of an operand are quite useful and are often provided in several variations. Shifts are operations in which the bits of a word are moved to the left or right. The bit shifted in at the end of the word determines the type of shift used. Shift instructions may specify either logical shifts, arithmetic shifts, or rotate type operations. In either case the shift may be to the right or to the left.



Name	Mnemonic
Logical Shift-right	SHR
Logical Shift-left	SHL
Arithmetic Shift-right	SHRA
Arithmetic Shift-Left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate Right through Carry	RORC
Rotate Left Through carry	ROLC

Table 5.9 Lists of shift operation

5.18 Reduced Instruction Set computer (RISC).

An important aspect of computer architecture is the design of the instruction set for the processor. The instruction set chosen for a particular computer determines the way that machine language programs are constructed. Early computers had small and simple instruction sets, forced mainly by the need to minimize the hardware to implement them. As digital hardware became cheaper with the advent of integrated circuits, computer instructions tended to increase both in number and complexity. Many computers have instruction sets that include more than 100 and sometimes even more than 200 instructions. These computers also employ a variety of data types and a large number of addressing modes. A computer with a large number of instructions is classified as a complex instruction set computer, abbreviated as CISC. In the early 1980s, a number of computer designers recommended that computers use fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often. This type of computer is classified as a Reduced Instruction Set Computer (RISC).

RISC characteristics

• Relatively few instructions

• Relatively few addressing modes

• Memory access limited to load and store instructions

• All operations done within the register of the CPU

• Fixed length, easily decoded instruction format

• Single cycle instruction execution

• Hard-wired rather than micro-programmed control

• A relatively large number of registers in the processor

• Use overlapped register windows to speed up procedure call and return

• Efficient instruction pipeline

• Compiler support for efficient transmission high-level language programs into machine language programs

5.19 CISC Instruction:

CISC refers to Complex Instruction Set Computer

CISC characteristic

• Large number of instructions – typically from 100 to 250 instructions

• Some instructions that perform specialized tasks and are used infrequently

• A large variety of addressing modes – typically from 5 to 20 different modes

• Variable length instruction formats

• Instructions that manipulate operands in memory RISC characteristics

Chapter 6: Memory Organization

6.1 Memory Hierarchy:

A computer system is equipped with a hierarchy of memory subsystems. There are several memory types with very different physical properties. The important characteristics of memory devices are cost per bit, access time, data transfer rate, alterability and compatibility with processor technologies. Figure shows the hierarchy of memory in a typical memory with a trend in access time, amount of storage, and cost per byte.

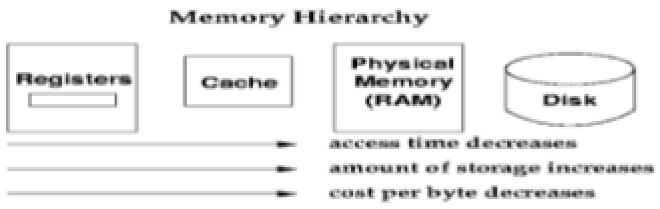


Figure 6.1 memory hierarchy

- Design constraints: How much? How fast? How expensive?
- Faster access time, greater cost per bit
- Greater capacity, smaller cost per bit,
- Greater capacity, slower access time.

6.2 Main Memory

The main memory (RAM) stores data and instructions. the capacity of the memory is 128 words of 8 bits (one byte) per word.

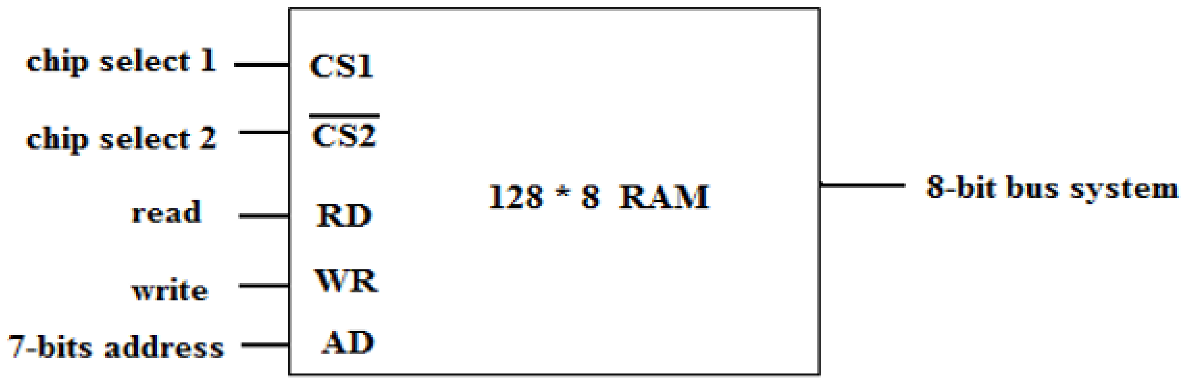


Figure 6.2 Block diagram of RAM

RAMs are built from semiconductor materials. Semiconductor memories fall into two categories, SRAMs (static RAMs) and DRAMs (dynamic RAMs).

DYNAMIC RAM (DRAM)

is made with cells that store data as charge on capacitors. The presence or absence of charge in a capacitor is interpreted as a binary 1 or 0. Because capacitors have a natural tendency to discharge, DRAMs require periodic charge refreshing to maintain data storage.

STATIC RAM (SRAM)

In a SRAM, binary values are stored using traditional flip-flop logic-gate. A static RAM will hold its data as long as power is supplied to it. Static RAM's are faster than dynamic RAM's.

Dynamic Verses Static RAM

Dynamic RAM:

- ♦ It requires periodic refreshing.
- ♦ Each cell stores bit with a capacitor and transistor.
- ♦ Large storage capacity
- ♦ Needs to be refreshed frequently.

- Used to create main memory.
- Slower and cheaper than SRAM.
- is simpler and hence smaller than the static RAM.
- it is denser and less expensive.

Static RAM:

- a bit of data is stored using the state of a flip-flop.
- Applying power is enough (no need for refreshing).
- Retains value indefinitely, as long as it is kept powered.
- Mostly uses to create cache memory of CPU
- Faster and more expensive than DRAM.

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 bootstrap loader, which is to start the computer software operating when power is turned on.

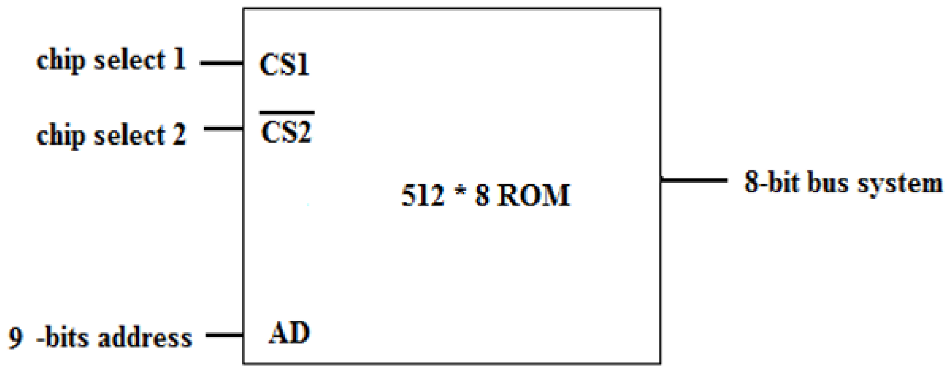


Figure 6.3 Block diagram of ROM

Types of ROM

ROM: The data is actually wired in the factory. The data can never be altered.

PROM: Programmable ROM. It can only be programmed once after its fabrication. It requires special device to program.

EPROM: Erasable Programmable ROM. It can be programmed multiple times. Whole capacity need to be erased by ultraviolet radiation before a new programming activity. It cannot be partially programmed.

EEPROM: Electrically Erasable Programmable ROM. Erased and programmed electrically. It can be partially programmed. Write operation takes considerably longer time compared to read operation

6.2.1 Connection of RAM and CPU

Data transfer between the main memory and the CPU register takes place through two registers namely MAR (memory address register) and MDR (memory data register). If MAR is k bits long and MDR is n bits long, the main memory unit can contain up to 2k addressable locations.

During a “memory cycle‟ n bits of data are transferred between main memory and CPU. This transfer takes place over the processor bus, which has k address lines and n data lines.

The CPU initiates a memory operation by loading the appropriate data into registers MDR and MAR, and setting either Read or Write memory control line to 1. When the required operation is completed the memory control circuitry sends Memory Function Completed (MFC) signal to CPU.

Memory Address Map is a pictorial representation of assigned address space for each chip in the system.

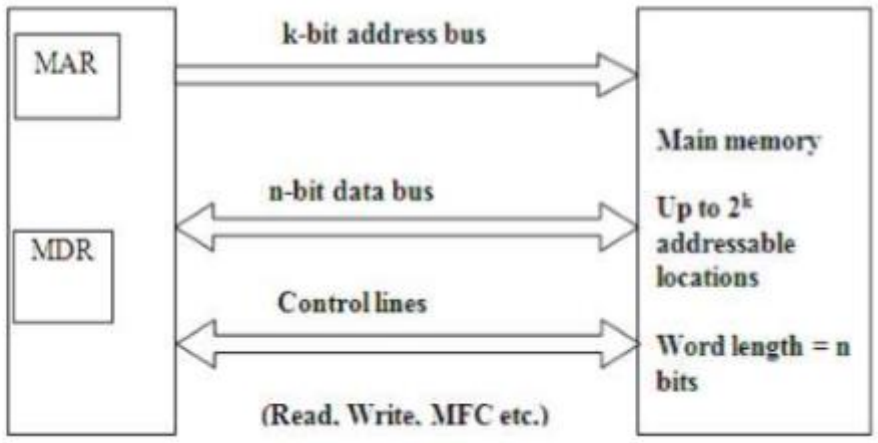


Figure 6.4 Connection of RAM to CPU

The time that elapses between the initiation of an operation and completion of that operation is called memory access time. The minimum time delay between two successive memory operations is called memory cycle time. The cycle time is usually slightly lower than the access time.

6.3 Cache Memory

Cache memory is a small, high-speed RAM buffer located between the CPU and main memory.

Cache memory holds a copy of the instructions (instruction cache) or data (operand or data cache) currently being used by the CPU.

The main purpose of a cache is to accelerate your computer while keeping the price of the computer low.

• The cache is the fastest component in the memory hierarchy and approaches the speed of CPU component

• When CPU needs to access memory, the cache is examined

• If the word is found in the cache, it is read from the fast memory

• If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word

• When the CPU refers to memory and finds the word in cache, it is said to produce a hit

• Otherwise, it is a miss

• The performance of cache memory is frequently measured in terms of a quantity called hit ratio

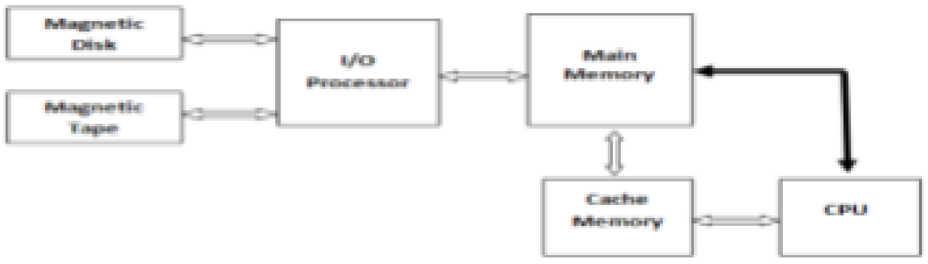
Hit ratio = hit / (hit+miss)

• The basic characteristic of cache memory is its fast access time

• Therefore, very little or no time must be wasted when searching the words in the cache

• The transformation of data from main memory to cache memory is referred to as a mapping process.

6.3.1 Placement of Cache memory in the computer



Types of Cache Mapping

1. Direct Mapping

2. Associative Mapping

3. Set Associative Mapping

Direct Mapping

The direct mapping technique is simple and inexpensive to implement. In general case, there are 2^k words in cache memory and 2^n words in main memory (in our case, k=9, n=15)

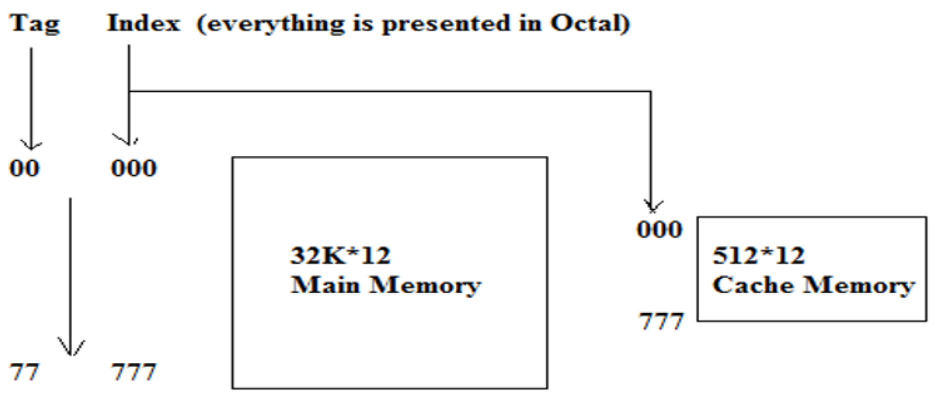
The n bit memory address is divided into two fields: k-bits for the index and n-k bits for the tag field

When the CPU wants to access data from memory, it places an address. The **index field of CPU** address is used to access address.

The **tag field of CPU** address is compared with the associated **tag in the word read from the cache**.

If the **tag-bits of CPU address** is matched with the **tag-bits of cache**, then there is a hit and the required data word is read from cache.

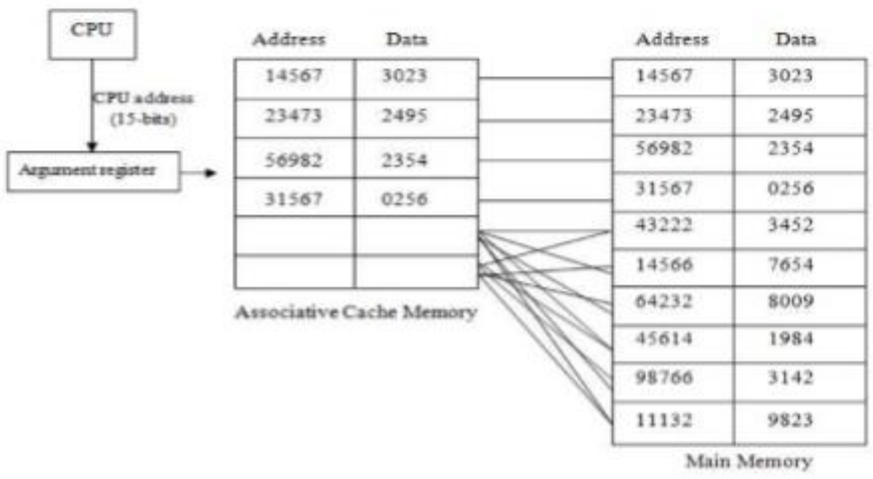
If there is **no match,** then there is a miss and the required data word is stored in main memory. It is then transferred from main memory to cache memory with the new tag.



Addressing relationships between main and cache memories

Associative Mapping

- An associative mapping uses an associative memory.
- This memory is being accessed using its contents.
- Associative memory is expensive compared to RAM
- Each line of cache memory will accommodate **the address** (main memory) and **the contents of that address** from the main memory.
- That is why this memory is also called Content Addressable Memory (CAM). It allows each block of main memory to be stored in the cache.



A CPU address of 15 bits is places in the argument register and the associative memory us searched for a matching address

• If the address is found, the corresponding 12bits data is read and sent to the CPU

• If not, the main memory is accessed for the word

If the cache is full, an address-data pair must be displaced to make room for a pair that is needed and not presently in the cache

Set Associative Mapping

That is the easy control of the direct mapping cache and the more flexible mapping of the fully associative cache.

In set associative mapping, each cache location can have more than one pair of tag + data items.

That is more than one pair of tag and data are residing at the same location of cache memory. If one cache location is holding two pair of tag + data items, that is called 2-way set associative mapping.

The disadvantage of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time

Set-Associative Mapping is an improvement over the direct-mapping in that each word of cache can store two or more word of memory under the same index address

6.3.2 Replacement Algorithms of Cache Memory

Replacement algorithms are used when there are no available space in a cache in which to place a data. Four of the most common cache replacement algorithms are described below:

Least Recently Used (LRU):

The LRU algorithm selects for replacement the item that has been least recently used by the CPU.

First-In-First-Out (FIFO):

The FIFO algorithm selects for replacement the item that has been in the cache from the longest time.

Least Frequently Used (LRU):

The LRU algorithm selects for replacement the item that has been least frequently used by the CPU.

Random:

The random algorithm selects for replacement the item randomly.

6.3.3 Writing into Cache

When memory write operations are performed, CPU first writes into the cache memory. These modifications made by CPU during a write operations, on the data saved in cache; need to be written back to main memory or to auxiliary memory.

These two popular cache write policies (schemes) are: Write-Through and Write-Back.

Write-Through

In a write through cache, the main memory is updated each time the CPU writes into cache.

The advantage of the write-through cache is that the main memory always contains the same data as the cache contains.

This characteristic is desirable in a system which uses direct memory access scheme of data transfer. The I/O devices communicating through DMA receive the most recent data.

Write-Back

In a write back scheme, only the cache memory is updated. During the updating, locations in the cache memory are marked by a flag so that later on, when the word is removed from the cache, it is copied into the main memory.

The words are removed from the cache time to time to make room for a new block of words.

Virtual Memory

The term virtual memory refers to something which appears to be present but actually it is not.

The virtual memory technique allows users to use more memory for a program than the real memory of a computer.

So, virtual memory is the concept that gives the illusion to the user that they will have main memory equal to the capacity of secondary storage media.

Concept of Virtual Memory

A programmer can write a program which requires more memory space than the capacity of the main memory. Such a program is executed by virtual memory technique.

The program is stored in the secondary memory. The memory management unit (MMU) transfers the currently needed part of the program from the secondary memory to the main memory for execution.

This to and fro movement of instructions and data (parts of a program) between the main memory and the secondary memory is called Swapping.

Address Space And Memory Space.

Virtual address is the address used by the programmer and the set of such addresses is called the address space or virtual memory.

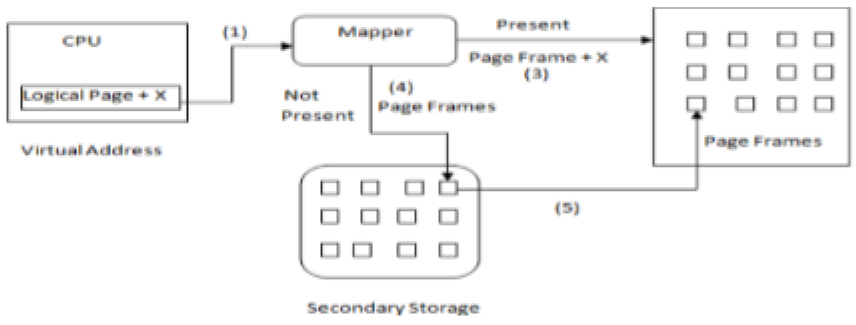
An address in main memory is called a location or physical address. The set of such locations in main memory is called the memory space or physical memory.

CPU generated logical address consisting of a logical page number plus the location within that page (x).

It must be mapped onto an actual (physical) main memory address by the operating system using mapper.

If the page is present in the main memory, CPU gets the required data from the main memory.

If the mapper detects that the requested page is not present in main memory, a page fault occurs and the page must be read from secondary storage into a page frame in main memory.



6.4 Secondary memory

Devises that provides backup storage are called an Auxiliary memory (**Secondary memory)**

- It stores information that is not necessarily in current use.
- It is slower and having higher capacity than primary memory.
- This kind of memory is large, slow and inexpensive.
- It is non-volatile storage media i.e. the contents are not erased when the power is switched off.
- Magnetic disk, Magnetic tape and optical disk are the examples of secondary storage.

6.4.1 Magnetic Disk

- Magnetic disks are the foundation of external memory on virtually all computer systems.
- A disk is a circular platter constructed of nonmagnetic material called the substrate coated with a magnetisable material.
- More suitable than magnetic tapes for a wider range of applications such as supporting direct access of data
- It is a thin, circular plate made of metal & plastic, which is coated with iron-oxide.
- We can randomly access the data.
- They must be stored in dust free environment.
- It stores large amount of data.
- The magnetic disks come in different sizes.
- Due to large storage capacity of magnetic disks and lesser failures the use of these devices increasing day by day.
- Due to their low cost and high data recording densities, the cost per bit of storage is low for magnetic disks.
- An additional cost benefit is that magnetic disks can be erased and reused many times
- Suitable for both on-line and off-line storage of data
- Data transfer rate for a magnetic disk system is normally higher than a magnetic tape system.

There are two types of Magnetic Disks:

- FLOPPY DISK
- HARD DISK

a. Floppy disks are:

- Very large amount of data can be stored in a small storage space.
- It is a portable, inexpensive storage medium that consists of thin, circular, flexible plastic Mylar film.
- It was introduced by IBM in 1972.
- Standard floppy disk has storage capacity up to 1.44MB.
- Floppy disks are compact, lightweight and easily portable from one place to another.
- Most popular secondary storage medium used in small computers.
- Also known as floppies or diskettes

Types of Floppy disks:

5¼-inch diskette, whose diameter is 5¼-inch. It is encased in a square, flexible vinyl jacket.

3½-inch diskette, whose diameter is 3½-inch. It is encased in a square, hard plastic jacket.

Advantages

- Reusable, portable, Handy.
- Very low price
- Provide random access of data

Disadvantages

- Not Durable and Prone to damage
- Very low Capacities

b. Hard Disk

- Round, flat piece of rigid metal (frequently aluminium) disks coated with magnetic oxide.
- It is a storage device that contains one or more inflexible, circular patterns that store data, instructions & information.
- We can store documents, presentation, database, e-mails, messages, music, video, software etc.
- Come in many sizes, ranging from 1 to 14-inch diameter.

Hard disk of capacities 10GB, 20GB, 40GB and even more are easily available.

RAID (Redundant Array of Independent Disks)

To achieve greater performance and higher availability, servers and larger systems use RAID disk technology.

RAID is a family of techniques for using multiple disks as a parallel array of data storage devices, with redundancy built in to compensate for disk failure.

- RAID is a set of physical disk drives viewed by the operating system as a single logical drive.
- Data are distributed across the physical drives of an array.
- Redundant disk capacity is used to store parity information, which guarantees data recoverability in case of a disk failure.
- the RAID array creates significant performance and reliability gains.

4.2 Optical Disk

- Is a Laser beam technology for recording and reading of data on the disk.Laser beam technology is used for recording/reading of data on the disk. Also known as laser disk / optical laser disk, due to the use of laser beam technology. Proved to be a promising random access medium for high capacity secondary storage because it can store extremely large amounts of data in a limited space.
- Access times for optical disks are typically in the range of 100 to 300 milliseconds and that of hard disks are in the range of 10 to 30 milliseconds.
- The most popular optical disk uses a disk of 5.25 inch diameter with storage capacity of around 650 Megabytes.

The optical disk became the preferred medium for music, movies and software programs because, its

- Compact
- lightweight
- durable and digital
- the optical disk also provides a minimum of 650 MB of data storage.

Advantage of Optical Disk

- The cost-per-bit of storage for optical disks is very low because of their low cost and enormous storage density.
- Optical disk drives do not have any mechanical read/write heads to rub against or crash into the disk surface. This makes optical disks a more reliable storage medium than magnetic tapes or magnetic disks.
- Optical disks have a data storage life in excess of 30 years. This makes them a better storage medium for data archiving as compared to magnetic tapes or magnetic disks.

Limitation of Optical disk

- Data once recorded, cannot be erased and hence the optical disks cannot be reused.
- The data access speed for optical disks is slower than magnetic disks.
- Optical disks require a complicated drive mechanism.

Optical Disk Types:

Compact Disk(CD):

Is A non-erasable disk that stores digitized audio information. The standard system uses 12-cm disks and can record more than 60 minutes of uninterrupted playing time.

Compact Disk Read-Only Memory (CD-ROM):

A non-erasable disk used for storing computer data. The standard system uses 12-cm disks and can hold more than 650 Mbytes.

Recordable CD (CD-R):

The user can write to the disk only once.

CD-RW:

The user can erase and rewrite to the disk multiple times.

Digital Versatile Disk (DVD):

A technology for producing digitized, compressed representation of video information, as well as large volumes of other digital data

DVD Rewritable (DVD-R)

the user can write to the disk only once. Only one-sided disks can be used.

DVD Rewritable (DVD-RW):

The user can erase and rewrite to the disk multiple times. Only one-sided disks can be used.

The Blu-ray Disc: is a technology plat form that can store sound and video while maintaining high quality and also access the stored content in an easy way to-use.

Advantage of Blu-ray Disc are:.

- Large recording capacity up to 27 GB.
- High-speed data transfer rate 36 Mbps.
- Easy to use disc cartridge.

Both the Compact Disk CD and the CD-ROM (compact disk read-only memory) share a similar technology. The main difference is that CD-ROM players are more rugged and have error correction devices to ensure that data are properly transferred from disk to computer.

• The optical disk is removable, allowing the disk itself to be used for archival storage. Most magnetic disks are no removable.

Disadvantages of CD-ROM are as follows:

• It is read-only and cannot be updated.

• It has an access time much longer than that of a magnetic disk drive, as much as half a second.

- The CD-RW has the obvious advantage over CD-ROM and CD-R that it can be rewritten and thus used as a true secondary storage.
- A key advantage of the optical disk is that the engineering tolerances for optical disks are much less severe than for high-capacity magnetic disks.Thus, they exhibit higher reliability and longer life,as much as a CD-ROM

6.4.2 Magnetic Tape

- Magnetic Tape is a plastic ribbon which is usually ½ inch or ¼ inch wide & 50 to 2400 feet long.
- It is coated with iron-oxide material.
- It is similar to the tape of audio cassettes of tape recorders. Data is stored as binary digits.
- Data is accessed sequentially so searching becomes difficult.
- Tape systems use the same reading and recording techniques as disk systems..
- Magnetic tape was the first kind of secondary memory.It is still widely used as the lowest-cost, slowest-speed member of the memory hierarchy.

Advantages:

- Store data up to few gigabytes and Low cost
- Magnetic tape used by both mainframes and microcomputers

Disadvantages:

- Sequential access so searching becomes difficult.
- We can either read or write data at one time

Chapter 7: Input / Output Organization

7.1 Peripheral (External) Devices

Input or output devices attached to the computer are called peripherals. Among the most known peripherals some of them are, keyboard display unit and printers.

Video monitors are the most commonly used peripheral. They consists keyboard as the input device and display unit as an output devices.

The input and output organization of a computer is a function of a size of computer and devices connected to it.

The difference between a small and large system is mostly depends on the amount of hardware the computer has available for communicating with peripheral unit and the number of peripheral connected to the system.

The input/output subsystem of a computer, referred to as I/O, 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 users. A computer serves no useful purpose without the ability to receive information from an outside source and to transmit results in a meaningful form. I/O operations are accomplished through a wide assortment of external devices that provide a means of exchanging data between the external environment and the computer. An external device attaches to the computer by a link to an I/O module. The link is used to exchange control, status, and data between the I/O module and the external device. An external device connected to an I/O module also called Interface is often referred to as a peripheral device or simply a peripheral.

7.2 Input output Interfaces

Input output Interfaces provides method for transferring information between internal storage and external IO devices

Peripheral connected to the computer needs special communication link for interacting them with the central processing unit. The purpose of communication link is to resolve the difference exist between the central computer and each peripheral. These components are called peripheral units.

7.3 Classification of External devices

External devices broadly can be classified into three categories:

Human readable: suitable for communicating with the computer user. Examples: Screen, keyboard, video display terminals (VDT) and printers.

Machine readable: suitable for communicating with equipment. Examples: magnetic disk & tapes systems, Monitoring and control, sensors and actuators which are used in robotics.

Communication: These devices allow a computer to exchange data with remote devices, which may be machine readable or human readable. Examples: Modem, Network Interface Card (NIC)

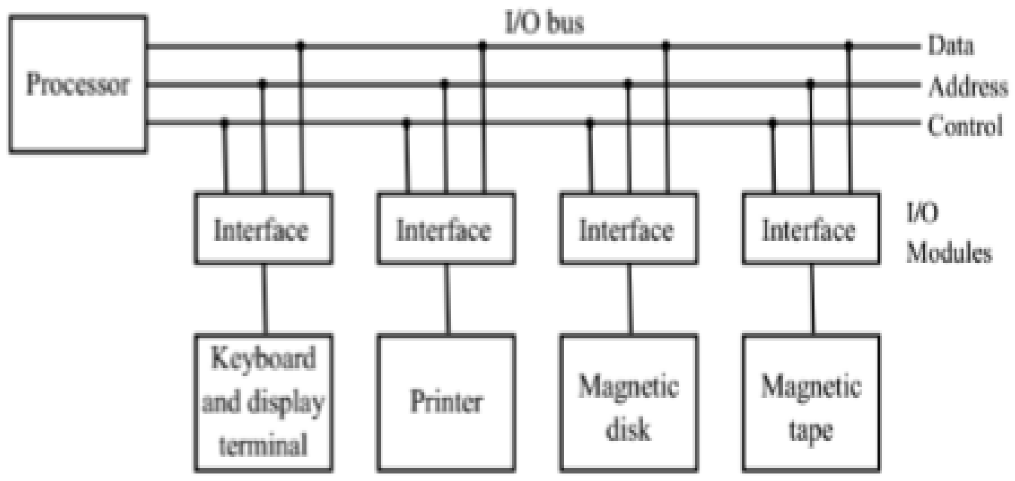


Figure 7-1: Connection of CPU, I/O module and peripheral devices

Asynchronous data transfer

The internal operation in digital system is synchronized by means of clock pulse supplied by a common pulse generator. Two units such as cpu and an I/O are designed independently of each other, if the register in the interface share a common clock with the cpu registers, the transfer between the two units is called to be synchronous.in most case the internal timing in each unit is independent from the other in that each uses its own private clock for internal register.in such cases the two units are said to be asynchronous to each other. This approach is widely used in most computers systems.

7.4 Input/output Problems

Wide variety of peripherals and delivering different amounts of data per second Work at different speeds Send/receive data in different formats all slower than CPU and RAM. Hence I/O modules are used as a solution.

Input/output Module: It is the entity within a computer that is responsible for the control of one or more external devices and for the exchange of data between those devices and main memory and/or CPU.

I/O Module Function The major functions or requirements for an I/O module fall into the following five categories.

- Control & Timing
- CPU Communication
- Device Communication
- Data Buffering and
- Error Detection

During any period of time, the CPU may communicate with one or more external devices in unpredictable patterns on the program's need for I/O. The internal resources, main memory and the CPU must be shared among number of activities including handling data I/O. Thus the I/O device includes a control and timing requirement to coordinate the flow of traffic between internal resources and external devices to the CPU. Thus CPU might involve in sequence of operations like:

- CPU checks I/O module device status
- I/O module returns device status
- If ready, CPU requests data transfer
- I/O module gets data from device
- I/O module transfers data to CPU

I/O module must have the capability to engage in communication with the CPU and external device. Thus CPU communication involves

Command decoding: The I/O module accepts commands from the CPU carried on the control bus.

Data: data are exchanged between the CPU and the I/O module over data bus

Status reporting:. I/O module can report with the status signals.common used status signals are BUSY or READY. Various other status signals may be used to report various error conditions.

Address recognition:

just as each memory word has an address, there is address associated with every I/O device. Thus I/O module must be recognized with a unique address for each peripheral it controls. The I/O module must also be able to perform device communication. This communication involves commands,status information, and data. Some of the essentials tasks are listed below:

- **Error detection**: I/O module is often responsible for error detection and subsequently reporting errors to the CPU.
- **Data buffering:** the transfer rate into and out of main memory or CPU is quite high, and the rate is much lower for most of the peripherals. The data is buffered in the I/O module and then sent to the peripheral device at its rate.

7.5 Input / Output Techniques (Data transfer mode) :

Three techniques are possible for I/O operations or data transfer mode. They are:

- Programmed I/O
- Interrupt driven
- Direct Memory Access (DMA)

Programmed I/O :With Programmed I/O, data are exchanged between the CPU and the I/O module. The CPU executes a program that gives it direct control of the I/O operation, including sensing device status, sending a read or write command and transferring data. When CPU issues a command to I/O module, it must wait until I/O operation is complete. If the CPU is faster than I/O module, there is wastage of CPU time. The I/O module does not take any further action to alert CPU. That is it doesn't interrupt CPU. Hence it is the responsibility of the CPU to periodically check the status of the I/O module until it finds that the operation is complete. The sequences of actions that take place with programmed I/O are:

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits
- CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU

7.6 I/O commands

To execute an I/O related instruction, the CPU issues an address, specifying the particular I/O module and external device and an I/O command. Four types of I/O commands can be received by the I/O module when it is addressed by the CPU. They are

- **A control command**: is used to activate a peripheral and tell what to do. Example: a magnetic tape may be directed to rewind or move forward a record.
- **A test command**: is used to test various status conditions associated with an I/O module and its peripherals. The CPU wants to know the interested peripheral for use. It also wants to know the most recent I/O operation is completed and if any errors have occurred.

- **A read command**: it causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer. The CPU then gets the data items by requesting I/O module to place it on the data bus.
- **A write command**: it causes the I/O module to take an item of data from the data bus and subsequently transmit the data item to the peripheral.

7.8 I/O Mapping

When the CPU, main memory, and I/O module share a common bus two modes of addressing are possible.

- **Memory mapped I/O**
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
  - Large selection of memory access commands available
- **Isolated I/O**
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O and Limited set

7.9 Priority interrupt

Data transfer between CPU and I/O device is initiated by the CPU.how ever the CPU cannot start the transfer unless the device is ready to communicate with the cpu.

A priority interrupt is a system that establish a priority over various sources to determine which condition is to be serviced first when two or more requests arrives simultaneously.

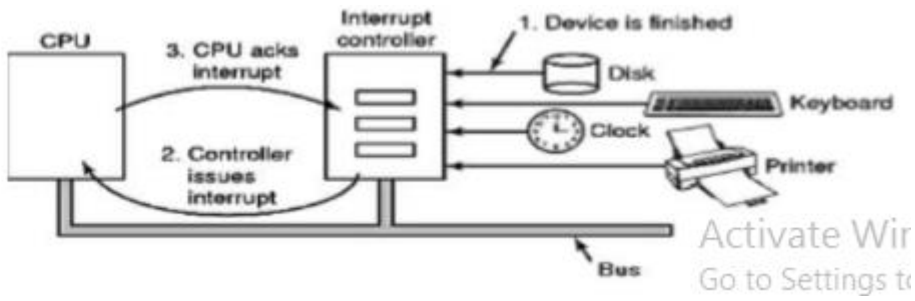
When two devices interrupt the computer at the same time,the computer services the devices with the higher priority first.

Interrupt Driven I/O.

Using Program-controlled I/O requires continuous involvement of the processor in the I/O activities. It is desirable to avoid wasting processor execution time. An alternative is for the CPU to issue an I/O command to a module and then go on other work. The I/O module will then interrupt the CPU requesting service when it is ready to exchange data with the CPU. The CPU will then execute the data transfer and then resumes its former processing. Based on the use of interrupts, this technique improves the utilization of the processor. With Interrupt driven I/O, the CPU issues a command to I/O module and it does not wait until I/O operation is complete but instead continues to execute other instructions. When I/O module has completed its work it interrupts the CPU. An interrupt is more than a simple mechanism for coordinating I/O transfers. In a general sense, interrupts enable transfer of control from one program to another to be initiated by an event that is external to a computer. Execution of the interrupted program resumes after completion of execution of the interrupt service routine. The concept of interrupts is useful in operating systems and in many control applications where processing of certain routines has to be accurately timed relative to the external events. Using Interrupt Driven I/O technique CPU issues read command.

I/O module gets data from peripheral while CPU does other work and I/O module interrupts CPU checks the status if no error that is the device is ready then CPU requests data and I/O module transfers data. Thus CPU reads the data and stores it in the main memory. Basic concepts of an Interrupt

An interrupt is an exception condition in a computer system caused by an event external to the CPU. Interrupts are commonly used in I/O operations by a device interface (or controller) to notify the CPU that it has completed an I/O operation. An interrupt is indicated by a signal sent by the device interface to the CPU via an interrupt request line (on an external bus).



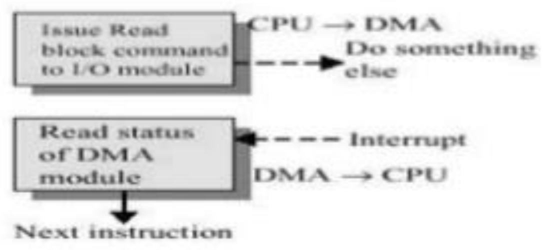
7.10 Direct Memory Access (DMA)

Interrupt driven and programmed I/O require active CPU intervention.

Transfer rate is limited

CPU is tied up DMA is the solution for these problems

Direct Memory Access is capabilities provided by some computer bus architectures that allow data to be sent directly from an attached device (such as a disk drive) to the memory on the computer’s motherboard. The microprocessor (CPU) is freed from involvement with the data transfer, thus speeding up overall computer operation.


Issue Read block command to I/O module
CPU → DMA
Do something else
Read status of DMA module
Interrupt
DMA → CPU
Next instruction

When the CPU wishes to read or write a block of data, it issues a command to the DMA module and gives following information: CPU tells DMA controller:

- Whether to read or write
- Device address
- Starting address of memory block for data
- Amount of data to be transferred The CPU carries on with other work.
- Thus DMA controller steals the CPU‟s work of I/O operation.
- The DMA module transfers the entire block of data
- One word at a time, directly to or from memory, without going through CPU.

When the transfer is complete, DMA controller sends interrupt when finished Thus CPU is involved only at the beginning and at the end of the transfer.

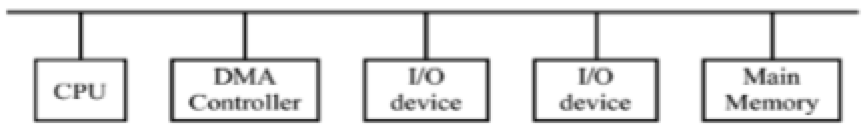
DMA Configurations

The DMA mechanism can be configured in variety of ways. Some of the common configurations are discussed here.

Single Bus Detached DMA

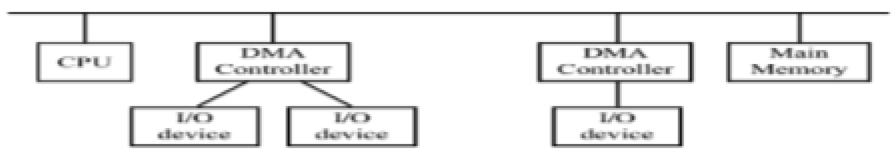
In this configuration all modules share the same system bus. The DMA module that is mimicking the CPU uses the programmed I/O to exchange the data between the memory and the I/O module through the DMA module. This scheme may be inexpensive but is clearly inefficient. The features of this configuration are:

- Single Bus, Detached DMA controller
- Each transfer uses bus twice
- I/O to DMA then DMA to memory
- CPU is suspended twice



Single Bus, integrated DMA

Here, there is a path between DMA module and one or more I/O modules that do not include the system bus. The block diagram of single bus Integrated DMA is as shown in Figure 7-5. The DMA logic can actually be considered as a part of an I/O module or there may be a separate module that controls one more I/O modules.

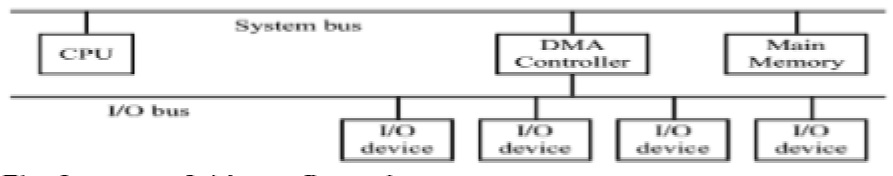


The features of this configuration can be considered as: Single Bus, Integrated DMA controller

- Controller may support >1 device
- Each transfer uses the bus once
- DMA to memory CPU is suspended once

DMA using an I/O bus

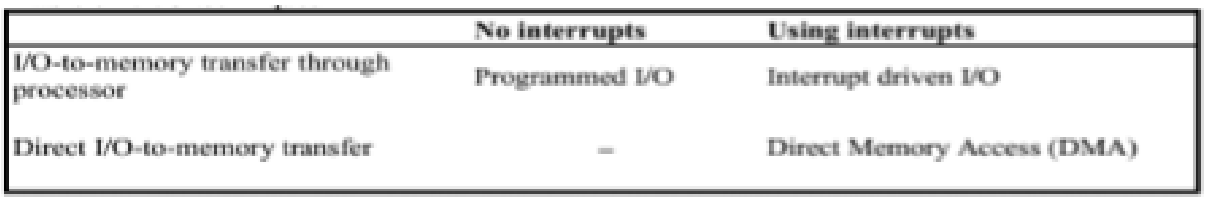
one further step of the concept of integrated DMA is to connect I/O modules to DMA controller using a separate bus called I/O bus. This reduces the number of I/O interfaces in the DMA module to one and provides for an easily expandable configuration. The block diagram of DMA using I/O bus is as shown in Figure below. Here the system bus that the DMA shares with CPU and main memory is used by DMA module only to exchange data with memory. And the exchange of data between the DMA module and the I/O modules takes place off the system bus that is through the I/O bus.



The features of this configuration are:

- Separate I/O Bus Bus supports all DMA enabled devices
- Each transfer uses bus once
- DMA to memory

CPU is suspended once With both Programmed I/O and Interrupt driven I/O the CPU is responsible for extracting data from main memory for output and storing data in main memory for input. Table indicates the relationship among the three techniques.



	No interrupts	Using interrupts
I/O-to-memory transfer through processor	Programmed I/O	Interrupt driven I/O
Direct I/O-to-memory transfer	–	Direct Memory Access (DMA)

7.10.1 Advantages of DMA

DMA has several advantages over polling and interrupts. DMA is fast because a dedicated piece of hardware transfers data from one computer location to another and only one or two bus read/write cycles are required per piece of data transferred. In addition, DMA is usually required to achieve maximum data transfer speed, and thus is useful for high speed data acquisition devices. DMA also minimizes latency in servicing a data acquisition device because the dedicated hardware responds more quickly than interrupts and transfer time is short. Minimizing latency reduces the amount of temporary storage (memory) required on an I/O device. DMA also off-loads the processor, which means the processor does not have to execute any instructions to transfer data. Therefore, the processor is not used for handling the data transfer activity and is available for other processing activity. Also, in systems where the processor primarily operates out of its cache, data transfer is actually occurring in parallel, thus increasing overall system utilization.

7.11 Serial communication

A data communication processor is an I/O processor that distribute and collect data from many remote terminals connected through telephone and other communication lines it is a specialized I/O processor designed to communicate directly with data communication network. The most striking difference between I/O processor and data communication processor is in the way processor communicate with the I/O devices.an I/O processor communicate with the peripherals through common I/O bus that is comprised of many data and control lines. All peripherals share common bus and use it to transfer information to and from I/O processor. A data communication processor communicates with each terminal through a single pairs of wires.

Data can be transmitted between two points in three different modes: simplex, half duplex and full duplex.

A simplex line carries information in one direction only. This mode is seldom used in data communication because the receiver cannot communicate to the transmitter with indicate the occurrence of error examples of simplex transmission are radio and television broadcasting.

Half duplex transmission system is one that have capable of transmitting in both direction but, data can transmitted in only one direction at a time. A pair of wire is needed for mode.

A full duplex transmission can send and receive data in both directions simultaneously. This can be achieved by means of a four wire link, with a different pair of wires dedicated to each direction of transmission. The communication lines, modems and other equipment used in the transmission of information between two more stations is called data link.

Chapter 8: Introduction to Parallel Processing

A traditional way to increase computer system performance is to use multiple processors that can execute in parallel to support a given workload. The most common multiple-processor organizations is symmetric multiprocessors (SMPs) SMP consists of multiple similar processors within the same computer, interconnected by a bus or some sort of switching arrangement. Each processor has its own cache and so it is possible for a given line of data to be present in more than one cache. When more than one processor is implemented on a single chip, the configuration is referred to as multiprocessor system. This design scheme is used to replicate some of the components of a single processor so that the processor can execute multiple threads/processes/tasks concurrently. Instruction pipelining, at least to the extent of overlapping fetch and execute operations, has been around for a long time. Both of these are examples of performing functions in parallel. As computer technology has evolved, and as the cost of computer hardware has dropped, computer designers have sought more and more opportunities for parallelism.

8.1 Parallel processing'

Parallel processing/computing, uses multiple processing elements simultaneously to solve a problem.

Parallel processing or parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel").

The following are important points about parallel processing:

- A parallel processing system is able to perform concurrent data processing to achieve faster execution time.
- The system may have two or more ALUs and be able to execute two or more instructions at the same time
- Also, the system may have two or more processors operating concurrently
- Goal is to increase the throughput – the amount of processing that can be accomplished during a given interval of time

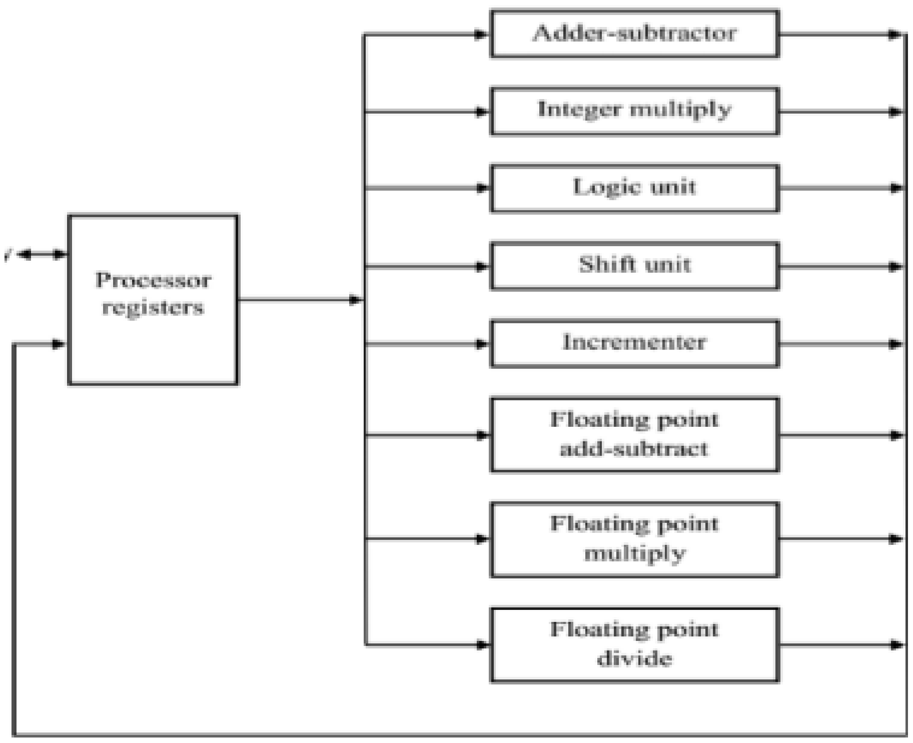
Parallel processing can be classified from:

- The internal organization of the processors
- The interconnection structure between processors
- The flow of information through the system
- The number of instructions and data items that are manipulated simultaneously

The sequence of instructions read from memory is the instruction stream

The operations performed on the data in the processor is the data stream

Parallel processing may occur in the instruction stream, the data stream, or both



Computer can be classified as:

- ❖ 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

- SISD – Instructions are executed sequentially. Parallel processing may be achieved by means of multiple functional units or by pipeline processing
- SIMD – Includes multiple processing units with a single control unit. All processors receive the same instruction, but operate on different data.
- MISD-- The same data stream flows through a linear array of processors executing different instruction streams.
- MIMD – A computer system capable of processing several programs at the same time.

8.3 Pipelining

Pipelining is an implementation technique where multiple instructions are overlapped in execution. Pipelining refers to the technique in which a given task is divided into a number of subtasks that need to be performed in sequence. Each subtask is performed by a given functional unit

Arithmetic Pipeline

Pipeline arithmetic units are usually found in very high speed computers. They are used to implement floating-point operations, multiplication of fixed-point numbers, and similar computations encountered in scientific problems. Example for floating-point addition and subtraction is shown below. The inputs are two normalized floating-point binary numbers X and Y where X = A x 2a and Y = B x 2b

A and B are two fractions that represent the mantissas

A and b are the exponents

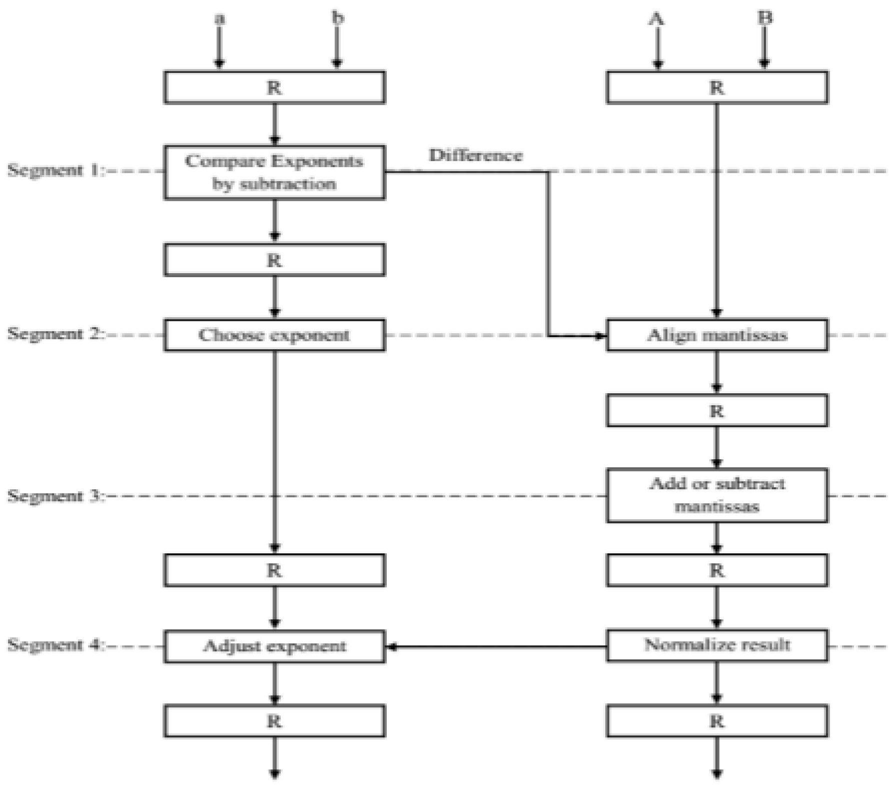
Four segments are used to perform the following sub operations:

1. Compare the exponents

2. Align the mantissas

3. Add or subtract the mantissas

4. Normalize the result



Instruction Pipeline:

Pipeline processing can occur not only in the data stream but also in the instruction stream as well. An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments. Whenever there is space in the buffer, the control unit initiates the next instruction fetch phase. The following steps are needed to process each instruction:

1. Fetch the instruction from memory
2. Decode the instruction
3. Calculate the effective address
4. Fetch the operands from memory
5. Execute the instruction
6. Store the result in the proper place

RISC Pipelines:

A RISC (Reduced Instruction Set Computer) processor pipeline operates in much the same way, although the stages in the pipeline are different. While different processors have different numbers of steps, they have basically variations of these five steps:

1. fetch instructions from memory

2. read registers and decode the instruction

3. execute the instruction or calculate an address

4. access an operand in data memory

5. write the result into a register

The length of the pipeline is dependent on the length of the longest step. Because RISC instructions are simpler than those used in pre-RISC processors (now called CISC, or Complex Instruction Set Computer), they are more conducive to pipelining. While CISC instructions varied in length, RISC instructions are all the same length and can be fetched in a single operation. Ideally, each of the stages in a RISC processor pipeline should take 1 clock cycle so that the processor finishes an instruction each clock cycle and averages one cycle per instruction (CPI). Among the characteristics attributed to RISC is its ability to use an efficient instruction pipeline.

8.4 Vector Processing

- The part of a computer that carries out the instructions of various programs is the central processing unit (CPU).
- The CPU, also called a processor, receives a program's instructions; decodes those instructions, breaking them into individual parts; executes those instructions; and reports the results, writing them back into memory.
- The format for the processor comes in one of two primary types: vector and scalar.
- The difference between the two is that scalar processors operate on only one data point at a time, while vector processors operate on an array of data.
- Many scientific problems require arithmetic operations on large arrays of numbers.
- These numbers are usually formulated as vectors and matrices of floating point numbers.
- A vector is an ordered set of a one dimensional array of data items. A vector V of length n is represented as a row vector by V= [V1, V2, V3,...,Vn].

  .