University of Mumbai

INSTITUTE OF DISTANCE AND OPEN LEARNING

MCA103

COMPUTER ORGANIZATION AND ARCHITECTURE

F.Y. MCA (CBCS)

SEMESTER - I

Computer Organization and Architecture
F.Y. MCA (CBCS) Semester I

Prof. Suhas Pednekar
Vice Chancellor
University of Mumbai, Mumbai

Prof. Prakash Mahanwar
Director Incharge
Institute of Distance & Open Learning,
University of Mumbai, Mumbai

Prof. Madhura Kulkarni
Asst. Director & Incharge
Study Material Section,
University of Mumbai, Mumbai

Programme Co-ordinator, I/c Faculty of Science & Technology

Prof. Mandar Bhanushe
Department of Mathematics, University of Mumbai, IDOL

Course Co-ordinator

Asst. Prof. Reshma Kurkute
Department - MCA, University of Mumbai, IDOL

Asst. Prof. Shardul Gavande
Department - MCA, University of Mumbai, IDOL

Course Writer

Asst. Prof. Amit Kukreja
K. J. Somaiya Institute of Engineering & Information Technology, Sion. Mumbai

Asst. Prof. Umesh Koyande
Vidyalankar School of Information Technology, Wadala (East), Mumbai.

Asst. Prof. Aarati Sahitya
K. J. Somaiya Institute of Engineering & Information Technology, Sion. Mumbai

Asst. Prof. Prachi Surve
Department - I.T., Ramniranjan Jhunjunwala College, Ghatkopar (W), Mumbai.

Published by
Dr. Prakash Mahanwar, Director Incharge
on behalf of Institute of Distance & Open Learning,University of Mumbai, Mumbai
and Printed at
Name & Address.

DTP Composed by 7skills

CONTENTS

Computer Organization and Architecture

F.Y. MCA (CBCS) Semester I

SYLLABUS

Sr. No.	Module	Detailed Contents	Hrs
1	**Fundamentalsof Digital Logic**	Boolean Algebra, Logic Gates, Simplification of Logic Circuits: Algebraic Simplification, Karnaugh Maps. Combinational Circuits : Adders, Mux, De-Mux, Sequential Circuits : Flip-Flops (SR, JK & D), Counters : synchronous and asynchronous Counter	**12**
2	**ComputerSystem**	Comparison of Computer Organization &Architecture, Computer Components and Functions, Interconnection Structures. Bus Interconnections, Input / Output: I/O Module, Programmed I/O, Interrupt Driven I/O, Direct Memory Access	**06**
3	**Memory System Organization**	Classification and design parameters, Memory Hierarchy, Internal Memory: RAM, SRAM and DRAM, Interleaved andAssociative Memory. Cache Memory: Design Principles, Memory mappings, Replacement Algorithms, Cache performance, Cache Coherence. Virtual Memory, External Memory : Magnetic Discs, Optical Memory, Flash Memories,RAID Levels	**08**
4	**Processor Organization**	Instruction Formats, Instruction Sets, Addressing Modes, Addressing Modes Examples with Assembly Language [8085/8086 CPU] , Processor Organization, Structure and Function. Register Organization, Instruction Cycle, Instruction Pipelining. Introduction to RISC and CISC Architecture, Instruction Level Parallelism and Superscalar Processors: Design Issues.	**12**

Sr. No.	Module	Detailed Contents	Hrs
5	**Control Unit**	Micro-Operations, Functional Requirements, Processor Control, Hardwired Implementation, Micro-programmed Control	**04**
6	**Fundamentals of Advanced Computer Architecture**	Parallel Architecture: Classification of Parallel Systems, Flynn's Taxonomy, Array Processors, Clusters, and NUMAComputers. Multiprocessor Systems : Structure & InterconnectionNetworks, Multi-Core Computers: Introduction, Organization and Performance.	**08**
7	**Case Study**	Case study : Pentium 4 processor Organization and Architecture	**02**

1

FUNDAMENTALS OF DIGITAL LOGIC

Unit Structure

1.1 Boolean Algebra

- Boolean algebra was invented in 1854 by George boole.
- It uses only the binary numbers 0 1nd 1.
- It is used to analyze and simplify the digital (logic) circuits.
- It is also called Binary Algebra or logical Algebra.
- It is mathematics of digital logic
- A variable is a symbol usually represented with an uppercase letter
- Complement is the inverse of a variable; it can be denoted by bar above variable.
- A literal ia a variable or the complement of a variable.
- Boolean expressions are created by performing operations on Boolean variables. – Common Boolean operators include AND, OR, and NOT
- OR operation between two variables denoted using plus (+) symbol (A OR B as A + B)
- AND operation between two variables denoted using dot (.) symbol (A AND B as A . B)
- NOT operation is a unary operation i.e. complement of a variable (NOT A as $\overline{A}$ or A')

Following are Boolean Algebra Law:

Sr. No.	Law	OR operation	AND operation
1	Commutative Law	x + y = y + x	x • y = y • x
2	Associative Law	x+(y + z)=(x + y)+z	x(y • z) = (x • y) • z
3	Distributive Law	x •(y + z) =x • y + x • z	x + y • z=(x + y)•(x + z)
4	Identity Law	x + 0 = x	x • 1 = x
5	Null Law	x + 1 = 1	x • 0 = 0
6	Complement Law	x + x' = 1	x • x' = 0
7	Idempotent Law	x + x = x	x • x = x
8	De Morgan's Law	$\overline{x + y} = \overline{x}\ \overline{y}$	$\overline{x \cdot y} = \overline{x} + \overline{y}$

Duality Principle:

According to this principle every valid boolean expression (equality) remains valid if the operators and identity elements are interchanged.

In this principle,

- if we have theorems of Boolean Algebra for one type of operation then that operation can be converted into another type of operation
- i.e., AND can be converted to OR and vice-versa
- interchange

  '0 with 1',

  '1 with 0',

  '(+) sign with (.) sign' and

  '(.) sign with (+) sign'.
- This principle ensures if a theorem is proved using the theorem of Boolean algebra, then the dual of this theorem automatically holds and we need not prove it again separately. This is an advantage of dual principle.

- Some Boolean expressions and their corresponding duals are given below,

  Example :

  If boolean expression is
  A.(B+C)=(A.B) + (A.C)

  Then its dual expression is ,
  A+(B.C)=(A+B).(A+C)

Boolean expression :

It is an algebraic statement which contains variables and operators.

Theorems/axioms/postulates can also be proved using the truth table method.

The other method is by an algebraic manipulation using axioms/postulates or other basic theorems.

Few application of dual principle are as follows:

Idempotency Law

A) $x + x = x$

B) $x \cdot x = x$

We will prove part A)

LHS $= (x + x). 1$ ---------- Identity law
$= (x + x).(x + x')$ ---------- Complement law
$= x + x.x'$ ---------- Distributive law
$= x + 0$ ---------- Complement law
$= x$ ---------- Identity law
$= RHS$

As part A is proved, according to dual principle we need not to prove part B.

Absorption Law

A) $x + (x \cdot y) = x$

B) $x \cdot (x + y) = x$

We will prove part A)

LHS = x . 1 + x . y ---------- Identity law

= x . (1 + y) ---------- Distributive law

= x . (y + 1) ---------- Commutative law

= x . 1 ----------*? (Identify the law)* _____

= x ---------- Identity law

= RHS

As part A is proved, according to dual principle we need not to prove part B.

Check your progress:

1. What values of the boolean variables x and y satisfy xy = x + y?

2. What is the Boolean variable and what is it used for?

3. What is the complement of a Boolean number? How do we represent the complement of a Boolean variable, and what logic circuit function performs the complementation function?

4. Prove DeMorgan's Law using dual principle.

1.2 Logic gate

Logic gates are the basic building blocks of digital systems.

This electronic circuit has one or more than one input and only one output.

Basic logic gates are AND gate, OR gate, NOT gate etc.

AND Gate

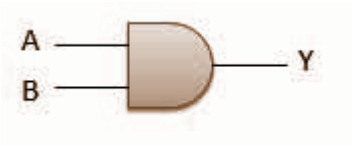
It is a binary operation, it requires at least two inputs and generates one output.

The output of AND gate is High or On or 1 only if both the inputs are High or On or 1, else for the rest of all cases the output is Low or off or 0.

Symbol

$Y = A \cdot B$

Logic diagram



Truth Table

Input		**Output**
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR Gate

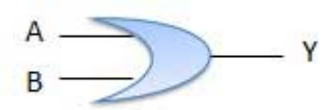
It is a binary operation, it requires at least two inputs and generates one output.

The output of OR gate is Low or Off or 0 only if both the inputs are Low or Off or 0 , else for the rest of all cases the output is High or On or 1.

Symbol

Y = A + B

Logic diagram



Truth Table

Input		**Output**
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOT Gate

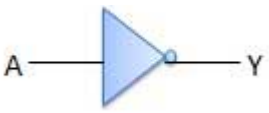
It is a unary operation, it requires only one input and generates one output.

The output of the NOT gate is Low or Off or 0 only if input is High or On or 1, and vice versa.

Symbol

Y = A ’

Logic diagram



Truth Table

Input	Output
A	Y
0	1
1	0

Special type of gates:

XOR Gate

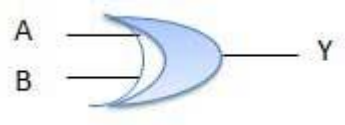
XOR or Ex-OR gate is a special type of gate. It is a binary operation required at least two inputs and only one output. If all the input have same value i.e. Low or Off or 0 or High or On or 1 then the output is Low or Off or 0 , else for the rest of all cases the output is High or On or 1.

It can be used in the half adder, full adder and subtractor.

Symbol

$Y = A \oplus B$

diagram



Truth Table

Input		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

XNOR Gate

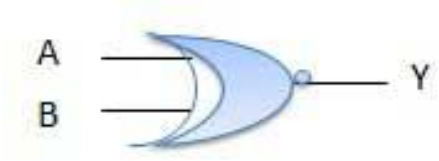
XNOR gate is a special type of gate. It is a binary operation with at least two inputs and only one output. If all the input have same value i.e. Low or Off or 0 / High or On or 1 then the output is High or On or 1 , else for the rest of all cases the output is Low or Off or 0.

It can be used in the half adder, full adder and subtractor.

Symbol

$Y = A \ominus B$

Logic diagram



Truth Table

Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

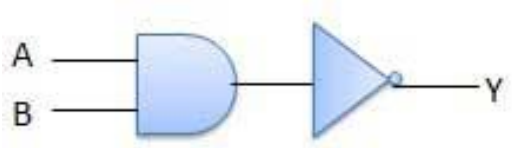
NAND Gate

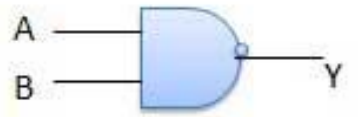
A NOT-AND operation is known as NAND operation. It has at least two or more than two inputs and one output. The output is Low or off or 0 only if all the inputs are high or On or 1. For rest of the cases output is High or On or 1.

Symbol

$Y = \overline{AB}$

Logic diagram





Truth Table

Input		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

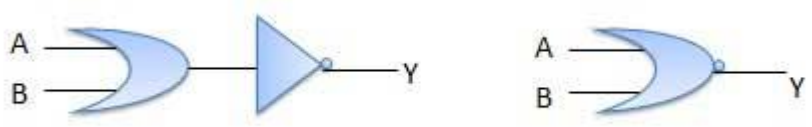
NOR Gate

A NOT-OR operation is known as NOR operation. It has at least two or more than two inputs and one output. The output is high or On or 1only if all the inputs are Low or off or 0 . For rest of the cases output is Low or off or 0.

Symbol

Y= $\overline{A+B}$

Logic diagram

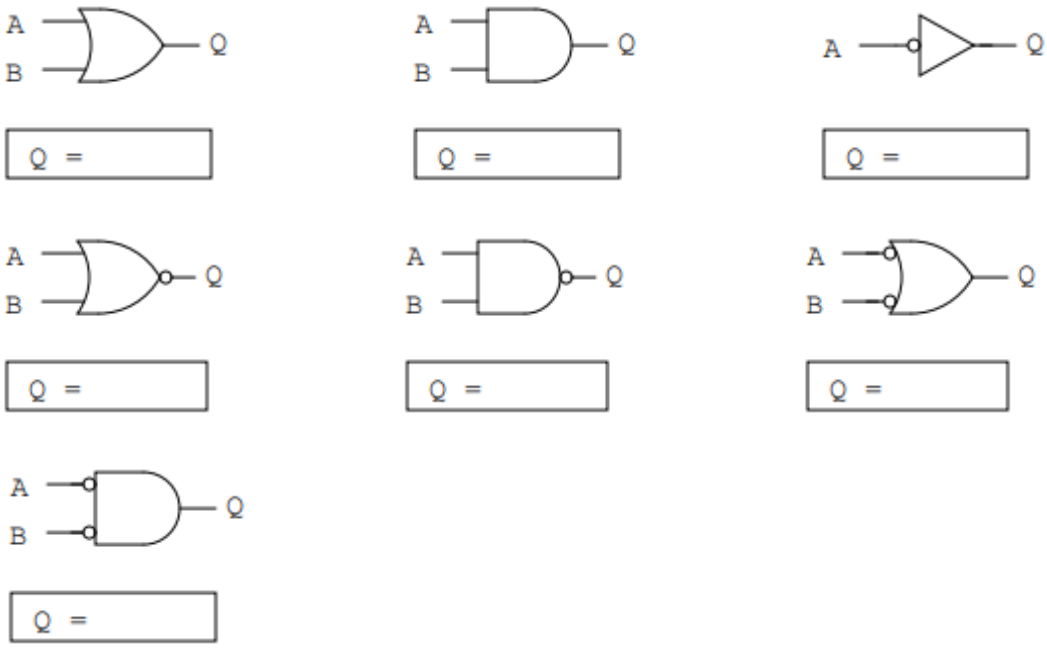


Truth Table

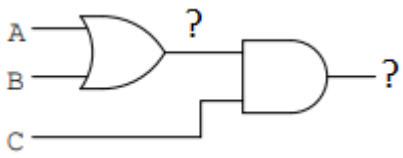
Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Check your progress

Write the Boolean expression for each of these logic gates, showing how the output (Q) algebraically relates to the inputs (A and B):



Convert the following logic gate circuit into a Boolean expression, writing Boolean sub-expressions next to each gate output in the diagram:



1.3 Simplification of Logic Circuits

Algebraic Simplification, Karnaugh Maps

IMPLEMENTATION OF BOOLEAN FUNCTIONS:

- Operator precedence from highest to lowest is : Bracket , NOT (’), AND(.) and OR (+)
- Using the truth table we can write boolean expressions for the output.
- Then we can define reduced expressions using logic gates.
- For any output signal is to be written from the truth table only those input combinations are written for which the output is high.

Example : Write boolean expression for following truth table:

x	y	z	F_1	F_2	F_3
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	1	0
1	1	1	0	1	0

F1 output have high high value only when input xyz has value 1 1 0.

Hence boolean expression for F1 is,

F1 = x . y. z'

Similarly we can write boolean expression for rest of the output as follow:

F2 = x' . y' . z + x . y' . z' + x . y' . z + x . y . z' + x . y . z

F3 = x' . y' . z + x' . y . z + x . y' . z' + x . y' . z

Now we can reduce the above expression using boolean algebra.

F2 = x' . y' . z + x . y' . z' + x . y' . z + x . y . z' + x . y . z

= x' . y' . z + x . y' . (z' + z) + x . y .(z' + z)

= x' . y' . z + x . y' + x . y

= x' . y' . z + x . (y' + y)

= x' . y' . z + x

= (x' + x) . (y' . z + x) ……. Absorption law

= 1. (y' . z + x)

= (y' . z + x)

For F3 do it yourself :

Complement of a function : For function F , complement of function is F'.

Example :

If F = x . y . z'

Then F' = (x . y . z')'

= x' . y' . (z')'

= x' . y' . z

Standard Form

The following are two standard canonical forms for writing Boolean expression:

1. Sum-Of-Product (SOP) - It is a product term or logical sum OR operation of several product terms which produce high output.

   Write the input variables if the value is 1, and write the complement of the variable if its value is 0.

2. Product-Of-Sum (POS) - It is a sum term or logical product AND operation of several sum terms which produce low output.

   Writing the input variables if the value is 0, and write the complement of the variable if its value is 1.

Example :

Minterm and Maxterm:

Minterm of n variables is the product of n literals from the different variables.

n variables give maximum 2^n minterms.

It is denoted by small m.

Maxterm of n variables is the sum of n literals from the different variables.

n variables give maximum 2^n minterms.

It is denoted by capital M.

Following table represent minterm and maxterm for 2 variables:

Variables		Minterm		Maxterm	
x	y	Term	Notation	Term	Notation
0	0	x'.y'	m_o	x+y	M_o
0	1	x'.y	m_1	x+y'	M_1
1	0	x.y'	m_2	x'+y	M_2
1	1	x.y	m_3	x'+y'	M_3

Observing the above table , maxterm is the sum of terms of the corresponding minterm with its literal complement.

m_0 = x'.y'

= (x'.y')'

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

= x + y

= M_0

CANONICAL FORM :

It is a unique way of representing boolean expressions.Any boolean expression can be represented in the form of sum of minterm and sum of maxterm. ∑ symbol is used for showing the sum of minterm and ☐ symbol is used for showing product of maxterm.

Example: Consider the given truth table where x,y are input variables and F1,F2, F3 are output.

x	y	F1	F2	F3
0	0	0	0	1
0	1	0	1	0
1	0	1	1	0
1	1	0	0	1

Sum of minterms can be obtained by summing the minterms of the function where result is a 1 as follows:

F1 =x.y' $= \sum m(2)$

F2 =x'.y+x.y' $= \sum m(1,2)$

F3 =x'.y'+x.y $= \sum m(0,3)$

Product of maxterms can be obtained by the maxterms of the function where result is a 0 as follows:

F1 =(x'+y').(x'+y).(x+y) $= \prod M(0,1,3)$

F2 =(x'+y').(x+y) $= \prod M(0,3)$

F3 =(x'+y).(x+y') $= \prod M(1,2)$

Steps for conversion of canonical forms:

Sum of minterms to product of maxterms

- Rewrite minterm to maxterm
- Replace minterm indices with indices not already used.

Product of maxterms to sum of minterms

- Rewrite maxterm using minterm
- Replace maxterm indices with indices not already used.

Example :

From the above truth table ,

F1 $= \sum m(2) = \square M(0,1,3)$

F3 $= \square M(1,2) = \sum m(0,3)$

Using Algebraic manipulation:

Use a+a'=1 for missing variables in each term.

Example:

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

F = x.y'+x'.z

= x.y'.(z+z')+x'.z.(y+y')

= x.y'.z + x.y'.z' + x'.z.y + x'.z.y'

= x.y'z + x.y'.z' + x'.y.z + x'.y'.z (rewriting above term)

= $m_5 + m_4 + m_3 + m_1$

= $m_1 + m_3 + m_4 + m_5$

= $\sum m(1,3,4,5)$

Boolean expressions can be manipulated into many forms.

Boolean algebra is used in the simplification of logic circuits.

There are several rules of Boolean algebra that can be used in reducing expressions to the simplest form.

There are basically two types of simplification techniques:

- Algebraic Simplification
- Karnaugh Maps (K map)

1.3.1 Algebraic Simplification

Algebraic simplification aime to minimise the number of literals and terms.

Some instructions for reducing the given Boolean expression are listed below,

1. Remove all the parenthesis by multiplying all the terms if present.
2. Group all similar terms which are more than one, then remove all other terms by just keeping one.

   Example:

   ABC + AB +ABC + AB = ABC +ABC + AB +AB

   = ABC +AB
3. A variable and its negation together in the same term always results in a 0 value, it can be dropped.

   Example:

   A. BCC' + BC = A.(B.0) + BC

   = A . 0 + BC

   = BC
4. Look for the pair of terms that are identical except for one variable which may be missing in one of the terms. The larger term can be dropped. Example:

$$\begin{aligned} A\bar{B}\bar{C}D + A\bar{B}\bar{C} &= A\bar{B}\bar{C}(\bar{D} + 1) \\ &= (A\bar{B}\bar{C}).1 \quad \text{------- Null Law} \\ &= A\bar{B}\bar{C} \end{aligned}$$

5. Look for the pair of terms that have the same variables, with one or more variables complimented. If a variable in one term of such a variable is complimented while in the second term it is not, then such terms can be combined into a single term with that variable dropped.

Example

$$\begin{aligned} A\bar{B}\bar{C}D + A\bar{B}\bar{C}D &= A\bar{B}\bar{C}(\bar{D} + D) \\ &= (A\bar{B}\bar{C}).1 \\ &= A\bar{B}\bar{C} \end{aligned}$$

OR

$$AB(C+D) + AB\overline{(C+D)} = AB\,[(C+D) + \overline{(C+D)}]$$

$$= AB.\ 1$$

$$= AB$$

Examples:

- $AB + \overline{A}C + BC = AB + \overline{A}C$ (Consensus Theorem)

Proof Steps	Justification
$AB + \overline{A}C + BC$	
$= AB + \overline{A}C + 1 \cdot BC$	Identity element
$= AB + \overline{A}C + (A + \overline{A}) \cdot BC$	Complement
$= AB + \overline{A}C + ABC + \overline{A}BC$	Distributive
$= AB + ABC + \overline{A}C + \overline{A}CB$	Commutative
$= AB \cdot 1 + ABC + \overline{A}C \cdot 1 + \overline{A}CB$	Identity element
$= AB\ (1+C) + \overline{A}C\ (1 + B)$	Distributive
$= AB\ .\ 1 + \overline{A}C\ .\ 1$	$1+X = 1$
$= AB + \overline{A}C$	Identity element

Our boolean expression has 6 literals which we reduced to 4 literals.

Using the above laws, simplify the following expression:

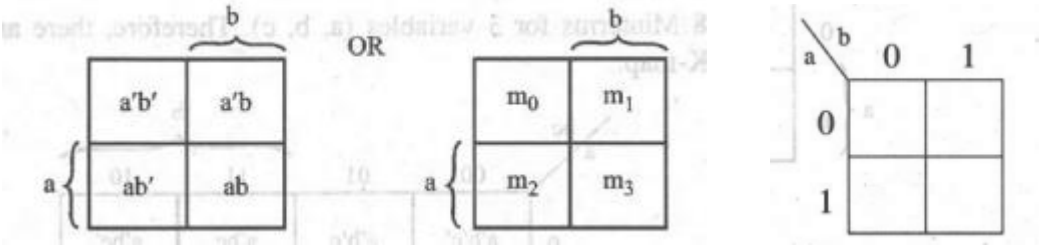
$(A + B)(A + C)$	$= A.A + A.C + A.B + B.C$	Distributive law
	$= A + A.C + A.B + B.C$	Idempotent AND law ($A.A = A$)
	$= A(1 + C) + A.B + B.C$	Distributive law
	$= A.1 + A.B + B.C$	Identity OR law ($1 + C = 1$)
	$= A(1 + B) + B.C$	Distributive law
	$= A.1 + B.C$	Identity OR law ($1 + B = 1$)
	$= A + (B.C)$	Identity AND law ($A.1 = A$)

1.3.2 Karnaugh Maps (K map)

Karnaugh maps also known as K-map minimize Boolean expressions of 3, 4 variables very easily without using any Boolean algebra theorems.

- K-map can take two forms Sum of Product (SOP) and Product of Sum (POS) according to the need of the problem.
- It is a graphical method, which consists of 2n cells for 'n' variables.
- It gives more information than TRUTH TABLE.
- Fill a grid of K-map with 0's and 1's then solve it by making groups.

2 Variable K-Map

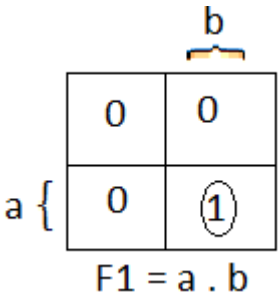


The K-map for a function which is the Sum of minterms is specified by putting:

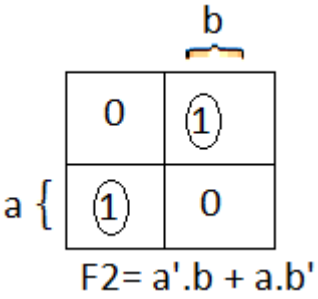
1 in the square corresponding to a minterm and 0 otherwise.

Example:

F1=a.b



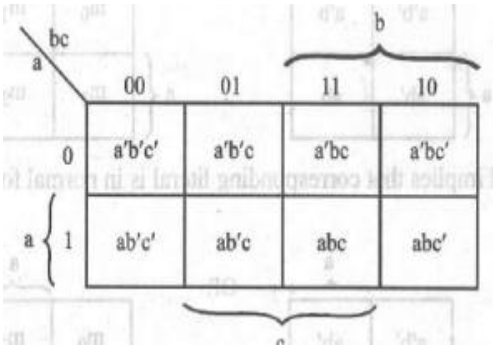
F2= a'.b + a.b'



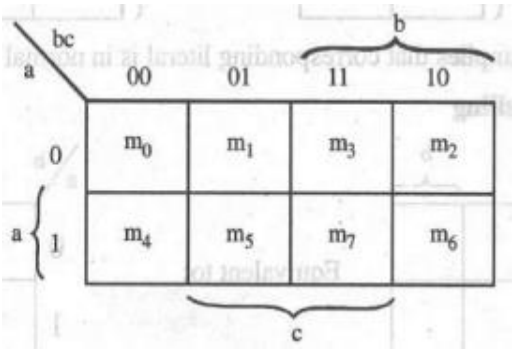
3 Variable K-Map

There are $2^3 = 8$ minterms for 3 variables a,b,c. Therefore there are 8 cells in a 3 variable k-map.

a	**b**	**c**	**y**	
0	0	0	a'.b'.c'	m_0
0	0	1	a'.b'.c	m_1
0	1	0	a'.b.c'	m_2
0	1	1	a'.b.c	m_3
1	0	0	a.b'.c'	m_4
1	0	1	a.b'.c	m_5
1	1	0	a.b.c'	m_6
1	1	1	a.b.c	m_7



Or



Wrap around in K-map:

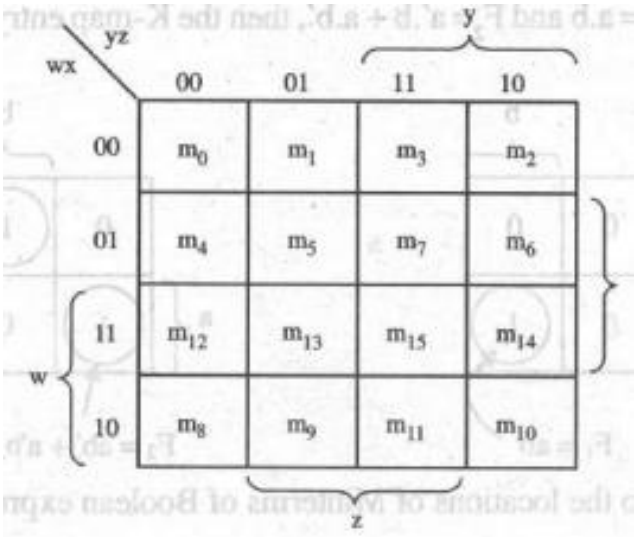
m_0 is adjacent to m_2 since only literal b is different.

m_4 is adjacent to m_6 since only literal b is different

Each cell in 3 variable K-map has 3 adjacent neighbors.

Each cell in an n-variable K-map has n adjacent neighbours.

4 Variable K-map:



<u>Steps to solve expression using K-map-</u>

1. Select K-map according to the number of variables.
2. Identify minterms or maxterms as given in problem.
3. For SOP put 1's in blocks of K-map respective to the minterms (0's elsewhere).

4. For POS put 0's in blocks of K-map respective to the maxterms(1's elsewhere).

5. Make rectangular groups containing total terms in power of two like 2,4,8 ..(except 1) and try to cover as many elements as you can in one group.

6. From the groups made in step 5 find the product terms and sum them up for SOP form.

Check your progress

Example

Simplify the Boolean expressions:

1- $A\overline{B} + A\overline{(B + C)} + B\overline{(B + C)}$.

2- $[A\overline{B}(C + BD) + \overline{A}\,\overline{B}]C$

3- $\overline{A}BC + A\overline{B}\,\overline{C} + \overline{A}\,\overline{B}\,\overline{C} + A\overline{B}C + ABC$

❖❖❖

2

COMBINATIONAL CIRCUITS : ADDERS, MUX, DE-MUX,

Unit Structure

2.1 Combinational Circuits

Combinational circuit is a circuit in which we combine the different gates in the circuit.

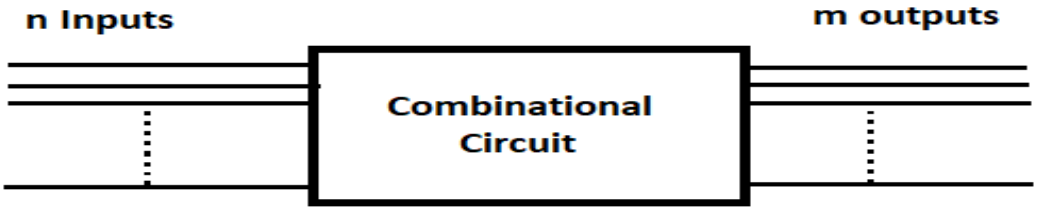
A logic gate is a basic building block of any electronic circuit. The output of the combinational circuit depends on the values at the input at any given time. The circuits do not make use of any memory or storage device

Examples are : encoder, decoder, multiplexer and demultiplexer.

Combinational logic is used in computer circuits to perform Boolean algebra on input signals and on stored data.Other circuits used in computers, such as half adders, full adders, half subtractors, full subtractors, multiplexers, demultiplexers, encoders and decoders are also made by using combinational logic.

Some of the characteristics of combinational circuits are following :

- The output of a combinational circuit at any instant of time, depends only on the levels present at input terminals.
- It does not use any memory. The previous state of input does not have any effect on the present state of the circuit.
- It can have an n number of inputs and m number of outputs.



Why do we use combinational circuits?

_____

_____

_____

_____

_____

_____

2.2 Adder

What is an Adder?

An adder is a circuit that can be integrated with many other circuits for a wide range of applications. It is a kind of calculator used to add two binary numbers. There are two kinds of adders;

1. Half adder
2. Full adder

Half Adder

With the help of half adder, we can design circuits that are capable of performing simple addition with the help of logic gates.

Example of the addition of single bits.

0+0 = 0

0+1 = 1

1+0 = 1

1+1 = 10

These are the possible single-bit combinations. But the result for 1+1 is 10. Though this problem can be solved with the help of an EXOR Gate, the sum result must be re-written as a 2-bit output.

Thus the above equations can be written as:

0+0 = 00

0+1 = 01

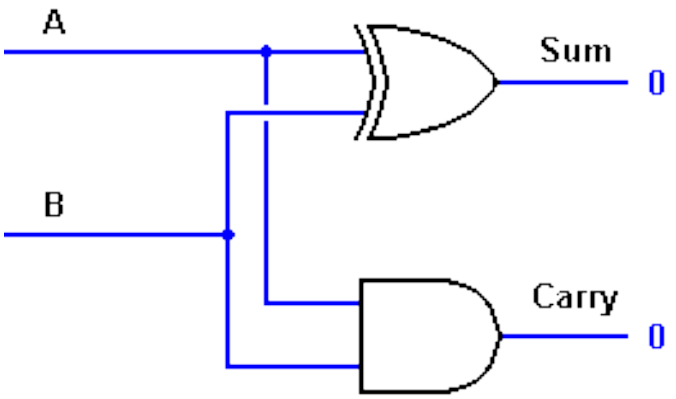
1+0 = 01

1+1 = 10

Here the output ‘1’of ‘10’ becomes the carry-out. The result is shown in a truth-table below. ‘SUM’ is the normal output and ‘CARRY’ is the carry-out.

INPUTS		OUTPUTS	
A	**B**	**SUM**	**CARRY**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

From the equation, it is clear that this 1-bit adder can be easily implemented with the help of EXOR Gate for the output ‘SUM’ and an AND Gate for the carry. Take a look at the implementation below.



Half Adder Circuit

Full Adder

This type of adder is used for complex addition. The main difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs. The first two inputs are A and B and the third input is an input carry designated as CIN. When a full adder logic is designed we will be able to string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next.

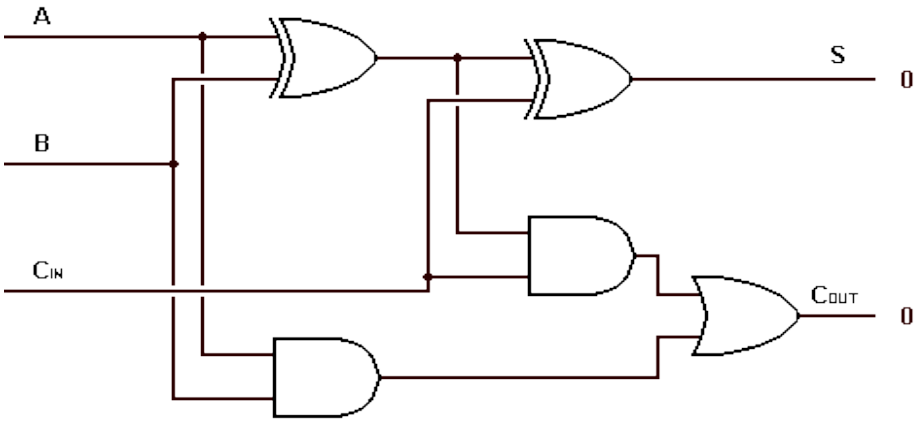
The output carry is designated as COUT and the normal output is designated as S. Take a look at the truth-table.

Full Adder logic truth table as given below:

INPUTS			OUTPUTs	
A	B	CIN	COUT	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

We can see that the output S is an EXOR between the input A and the half-adder SUM output with B and CIN inputs. We must also note that the COUT will only be true if any of the two inputs out of the three are HIGH.

We can implement a full adder circuit with the help of two half adder circuits. The first half adder will be used to add A and B to produce a partial Sum. The second half adder logic can be used to add CIN to the Sum produced by the first half adder to get the final S output. If any of the half adder logic produces a carry, there will be an output carry. Thus, COUT will be an OR function of the half-adder Carry outputs. The implementation of the full adder circuit shown below:



Full Adder Circuit

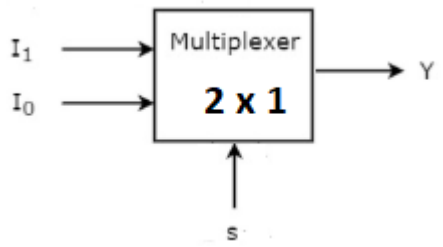
2.3 Mux, De-Mux

Multiplexer is a combinational circuit that has a maximum of 2^n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.

Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination will select only one data input. Multiplexer is also called as Mux.

2x1 Multiplexer

2x1 Multiplexer has two data inputs I1 & I0, one selection line S and one output Y. The block diagram of 2x1 Multiplexer is shown in the following figure.



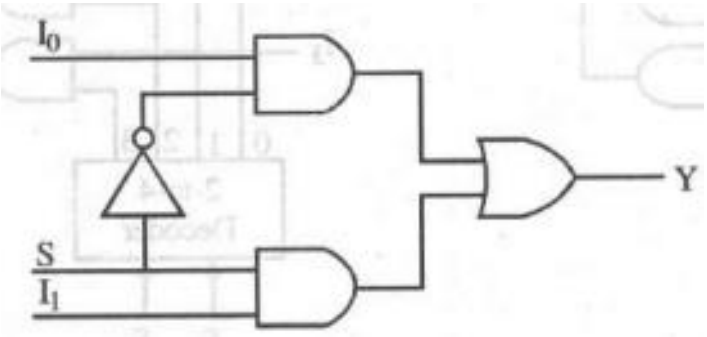
Truth table of 4x1 Multiplexer is shown below.

Input	Selection Lines (S)	OUTPUT (Y)
I_0	0	I_0
I_1	1	I_1

The Boolean function for output, Y as

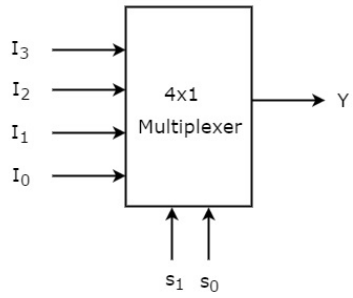
$Y= S'I_0 + SI_1$

The circuit diagram of 2x1 multiplexer is as below :



4x1 Multiplexer

4x1 Multiplexer has four data inputs I3, I2, I1 & I0, two selection lines s1 & s0 and one output Y. The block diagram of 4x1 Multiplexer is shown in the following figure.



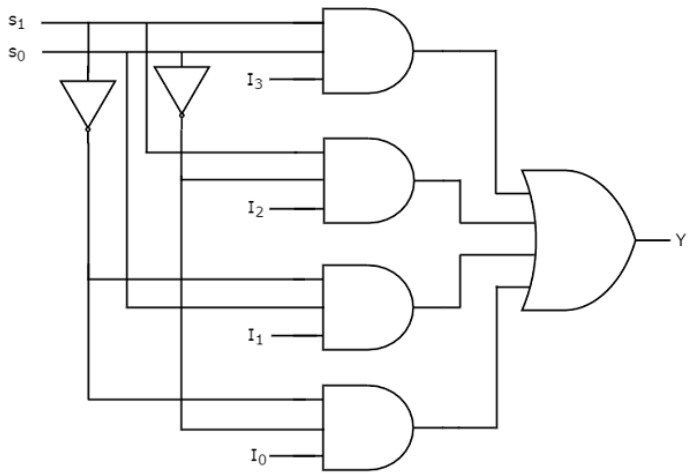
Truth table of 4x1 Multiplexer is shown below.

Selection Lines		OUTPUT
S_0	S_1	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

The Boolean function for output, Y as

$Y= S_0'S_1'I_0 + S_0'S_1I_1 + S_0S_1'I_2 + S_0S_1I_3$

The circuit diagram of 4x1 multiplexer is as below :



Higher Multiplexers:

Higher multiplexers can be constructed from smaller ones. An 8x1 multiplexer can be constructed from smaller multiplexers.

Important is placement of selector lines to get desired output.

For the 8x1 multiplexer we have eight input lines, I_7,I_6,I_5,I_4,I_3,I_2,I_1 and I_0 and three selection lines S_2,S_1 and S_0 (2^3=8).

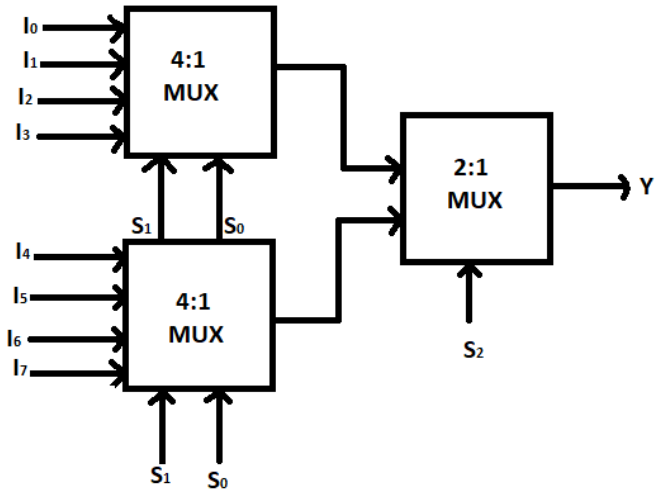
The truth table for the 8x1 multiplexer is as follows:

Selection Lines			Output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

Do it yourself:

Find the Boolean function for 8x1 multiplexer output, Y as

The block diagram of 4x1 multiplexer is as below :



The same selection lines, s1 & s0 are applied to both 4x1 Multiplexers.

The outputs of the first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in the second stage.

The other selection line, s2 is applied to 2x1 Multiplexer.

- If s2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I3 to I0 based on the values of selection lines s1 & s0.
- If s2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I7 to I4 based on the values of selection lines s1 & s0.

Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

Do it yourself

Draw circuit diagram for 8x1 multiplexer:

De-Multiplexer

De-Multiplexer or demux is a combinational circuit that performs the reverse operation of a Multiplexer. It has single input,and selects one of many data output lines,which is connected to the single input. It has 'n' selection lines and maximum of 2^n outputs. Since there are 'n' selection lines, there will be 2^n possible combinations of zeros and ones. So, each combination can select only one output.

The process of getting information from one input and transmitting the same over one of many outputs is called demultiplexing. There are several types of demultiplexers based on the output configurations such as 1:2, 1:4, 1:8 and 1:16.

1:2 Demultiplexer

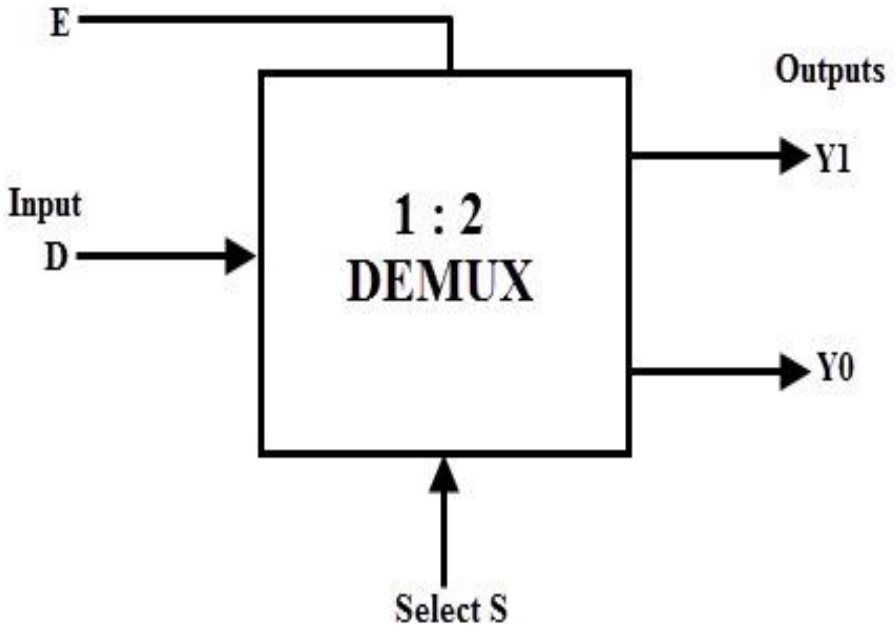
It has one input line and two output lines, one select line.

(Take note : 2^n = Number of output lines

Where as n is the number of select lines .

Therefore $2^1 = 2$)

In the figure, there are only two possible ways to connect the input to output lines, thus only one select signal is enough to do the demultiplexing operation. When the select input is low, then the input will be passed to Y0 and if the select input is high then the input will be passed to Y1.



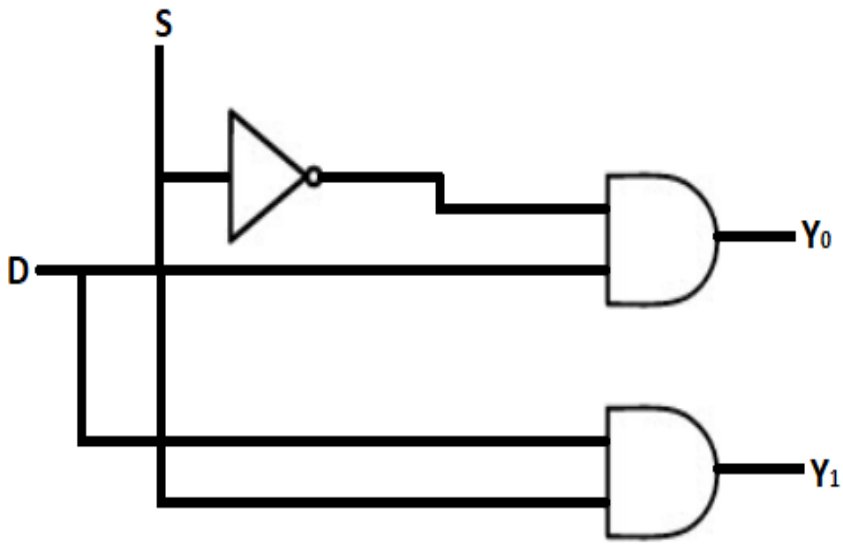
Truth table :

Select	Input	Output	
S	D	Y1	Y0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

From the above truth table, the logic diagram of this demultiplexer can be designed by using two AND gates and one NOT gate as shown in the figure below. When the select lines S=0, AND gate A1 is enabled while A2 is disabled.

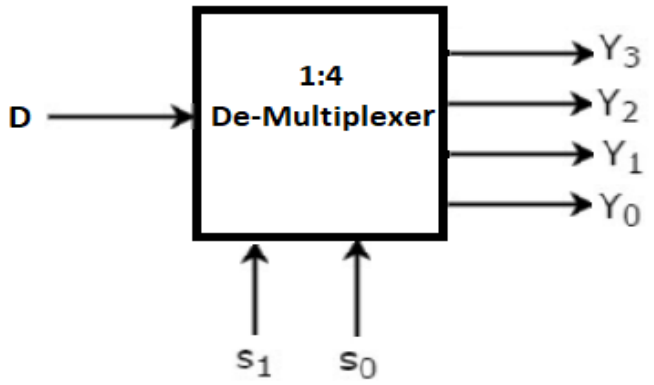
Then, the data from the input flows to the output line Y1. Similarly, when S=1, AND gate A2 is enabled and AND gate A1 is disabled, thus data is passed to the Y0 output.

Circuit diagram:



1:4 De-Multiplexer :

1:4 De-Multiplexer has one input D, two selection lines, s_1 & s_0 and four outputs Y_3, Y_2, Y_1 &Y_0. The block diagram of 1:4 De-Multiplexer is shown in the following figure.



The Truth table of 1:4 De-Multiplexer is shown below.

Selection Inputs		Outputs			
S_1	S_2	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Boolean functions for each output as :

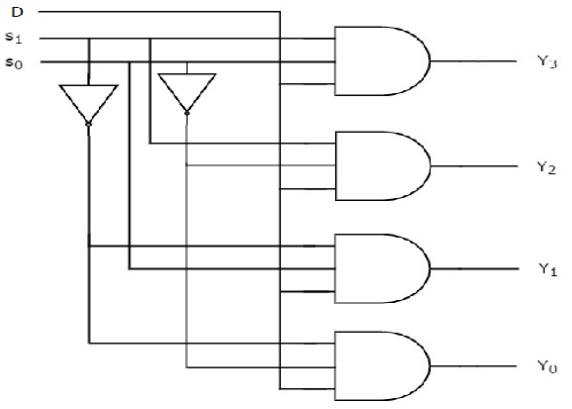
$Y_3 = S_1S_0D$

$Y_2 = S_1S_0'D$

$Y_1 = S_1'S_0D$

$Y_0 = S_1'S_0'D$

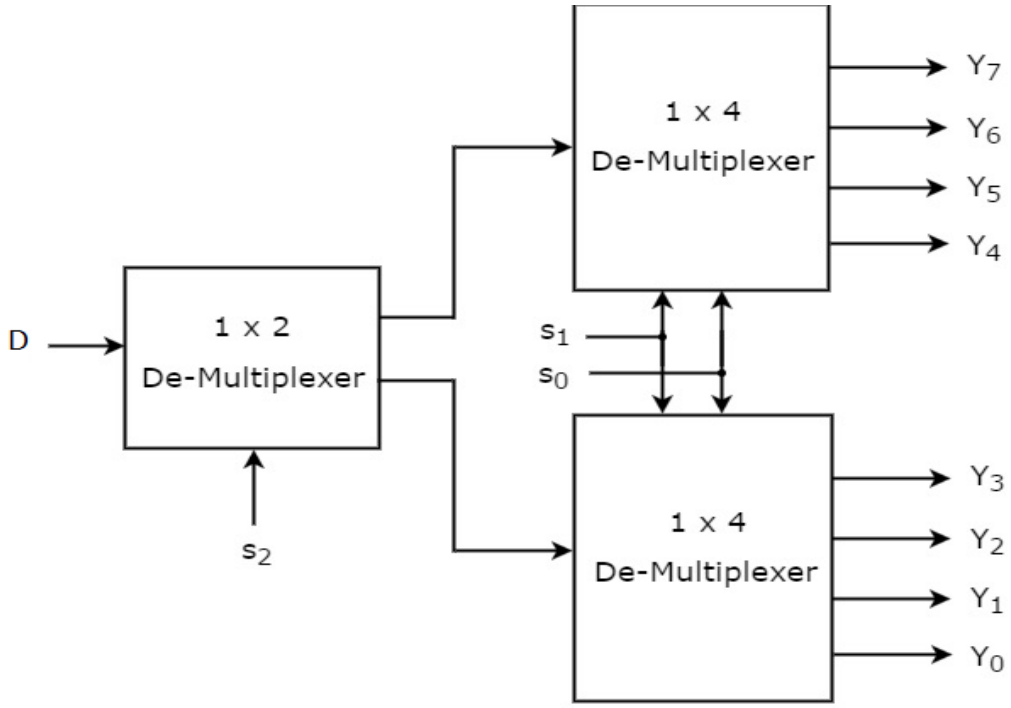
The circuit diagram of 1:4 De-Multiplexer using Inverters & 3-input AND gates is shown in the following figure:



Higher-order De-Multiplexers

We can implement 1:8 De-Multiplexer and 1:16 De-Multiplexer in similar way as we did for 1:2 and 1:4 De-Multiplexer.

The block diagram of 1x8 De-Multiplexer is shown in the following figure:



Do it yourself:

Draw block diagram for 1:16 De-Multiplexer with brief explanation:

Exercise:

1 Explain working of full adder circuit with its logic diagram and truth table

2 With the neat logic diagram explain the working of 4 bit parallel adder.

3 With the neat logic diagram explain the working of 4 bit parallel adder.

4 Draw the circuit diagram of full adder and explain it with the help of truth table .

5 Describe half adder with the help of the logic diagram and truth table.

6 What is multiplexer ? Draw circuit diagram of 8 : 1 multiplexer. explain its working in brief.

7 what is multiplexer ? Draw the circuit for a 2 : 1 nibble multiplexer and explain in brief its working

8 what is demultiplexer ? Explain working of a 1:4 demultiplexer with logic diagram.

9 Explain demultiplexer and its functions with the help of its illustrations.

3

SEQUENTIAL CIRCUITS

Unit Structure

3.1 Sequential Circuits

We have discussed combinational circuit in previous unit. The combinational circuit does not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit. But sequential circuit has memory so output can vary based on input. This type of circuits uses previous input, output, clock and a memory element.

Combinational circuit vs Sequential circuit :

	Combinational circuit	**Sequential circuit**
1	In this output depends only upon present input.	In this output depends upon present as well as past input.
2	Speed is fast. It is designed easy.	Speed is slow.It is designed tough as compared to combinational circuits.
3	It is easy to use and handle.	It is not easy to use and handle.
4	These circuits do not have any memory element	These circuits have memory element.

	Combinational circuit	**Sequential circuit**
5	Elementary building blocks: Logic gates	Elementary building blocks: Flip-flops
6	This is time independent.	This is time dependent.
7	Combinational circuits don't have capability to store any state.	Sequential circuits have capability to store any state or to retain earlier state.
8	As combinational circuits don't have clock, they don't require triggering.	As sequential circuits are clock dependent they need triggering.
9	There is no feedback between input and output.	There exists a feedback path between input and output.
10	Used for arithmetic as well as boolean operations.	Mainly used for storing data.
	Examples – Encoder, Decoder, Multiplexer, Demultiplexer	**Examples –** Flip-flops, Counters

3.1.1 Flip-Flop SR

Flip-flop is a circuit which has two stable states and can be used to store state information.Flip-flops are used as data storage elements.Such data storage can be used for storage of state and such a circuit is described as sequential logic. Flip-flops can be either simple or clocked. State can change either at positive transition,when the clock is changing from 0 to 1 or at negative transition, when the clock is changing from 1 to 0.

Four basic types of flip-flops based on clock trigger are as follow:

1. S-R Flip-flop
2. D Flip-flop
3. T Flip-flop
4. J-K Flip-flopS-R Flip-flop :

   S-R flip-flop is a set-reset flip-flop.

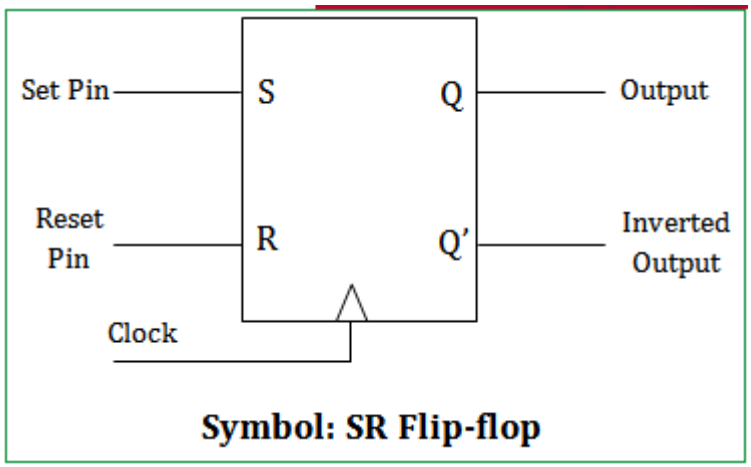
It is used in MP3 players, Home theatres, Portable audio docks, and etc.

S-R can be built with NAND gate or with NOR gate. Either of them will have the input and output complemented to each other.

We are using NAND gates for demonstrating the SR flip flop.

Whenever the clock signal is LOW, the inputs S and R are never going to affect the output. The clock has to be high for the inputs to get active.

Thus, SR flip-flop is a controlled Bi-stable latch where the clock signal is the control signal. Again, this gets divided into positive edge triggered SR flip flop and negative edge triggered SR flip-flop.



Symbol: SR Flip-flop

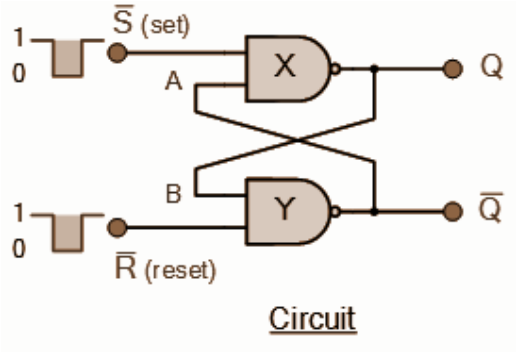
Truth table and circuit diagram :

CLK State	INPUT		OUTPUT	
Clock	S'	R'	Q	Q'
LOW	x	x	0	1
HIGH	0	0	0	1
HIGH	1	0	1	0
HIGH	0	1	0	1
HIGH	1	1	1	0

The memory size of SR flip flop is one bit.

The S and R are the input states for the SR flip-flop.

The Q and Q' represents the output states of the flip-flop.



3.1.2 D Flip-flop:

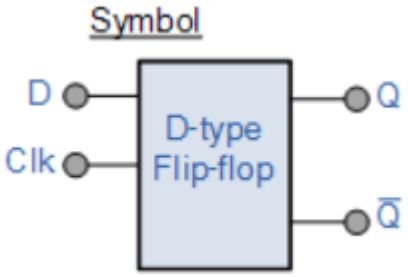
D Flip-flops are used as a part of memory storage elements and data processors as well.

It can be built using NAND gate or with NOR gate.

It is mainly used to introduce delay in timing circuit, as a buffer, sampling data at specific intervals.

It is simpler in terms of wiring connection compared to JK flip-flop.

We are using NAND gates for demonstrating the D flip flop.



The flip flop is a basic building block of sequential logic circuits. It is a circuit that has two stable states and can store one bit of state information. The output changes state by signals applied to one or more control inputs.

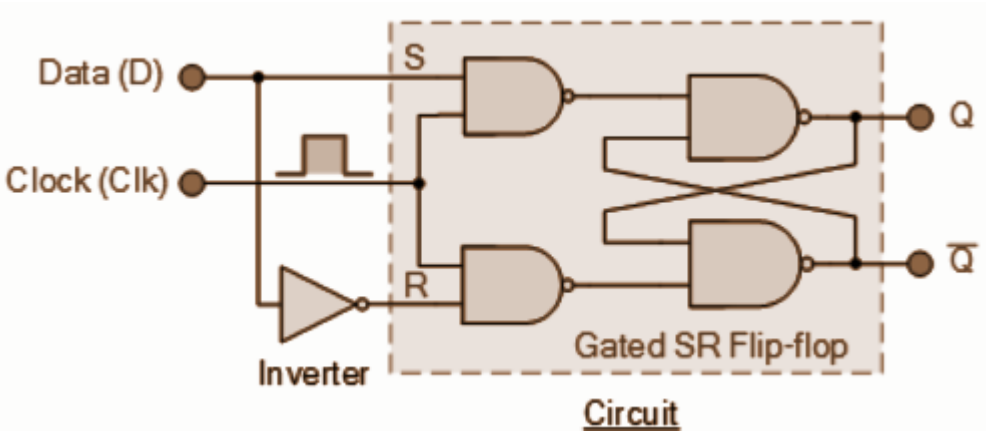
The basic D Flip Flop has a D (data) input and a clock input and outputs Q and Q' (the inverse of Q).

When the clock signal is LOW, the input is never going to affect the output state. The clock has to be high for the inputs to get active. Thus, D flip-flop is a controlled Bi-stable latch where the clock signal is the control signal. Again, this gets divided into positive edge triggered D flip flop and negative edge triggered D flip-flop.

The truth table and circuit diagram:

Clock	INPUT	OUTPUT	
	D	Q	Q'
LOW	x	0	1
HIGH	0	0	1
HIGH	1	1	0

The D(Data) is the input state for the D flip-flop. The Q and Q' represents the output states of the flip-flop. According to the table, based on the inputs the output changes its state. All these can occur only in the presence of the clock signal. This works exactly like SR flip-flop for the complementary inputs alone.



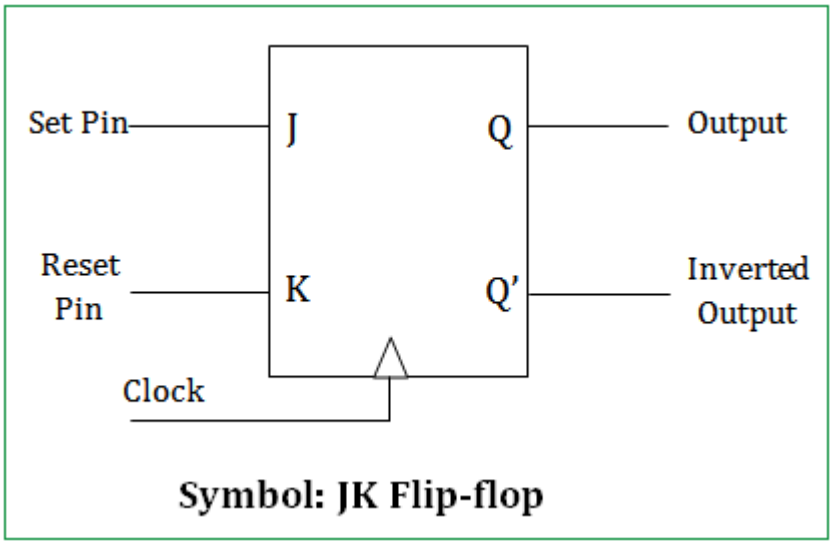
Circuit

3.1.3 J-K Flip-flop:

The name JK flip-flop is termed from the inventor Jack Kilby from texas instruments. JK flip-flop are used in Shift registers, storage registers, counters and control circuits. Similar to D type flip-flop, JK flip-flop has a toggling nature.

We are using NAND gates for demonstrating the JK flip flop.

Whenever the clock signal is LOW, the input is never going to affect the output state. The clock has to be high for the inputs to get active. Thus, JK flip-flop is a controlled Bi-stable latch where the clock signal is the control signal.



Symbol: JK Flip-flop

Truth table and circuit diagram :

***Table 1.27** Truth Table for J-K Flip-Flop*

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

***Table 1.26** Characteristic Table for J-K Flip-Flop*

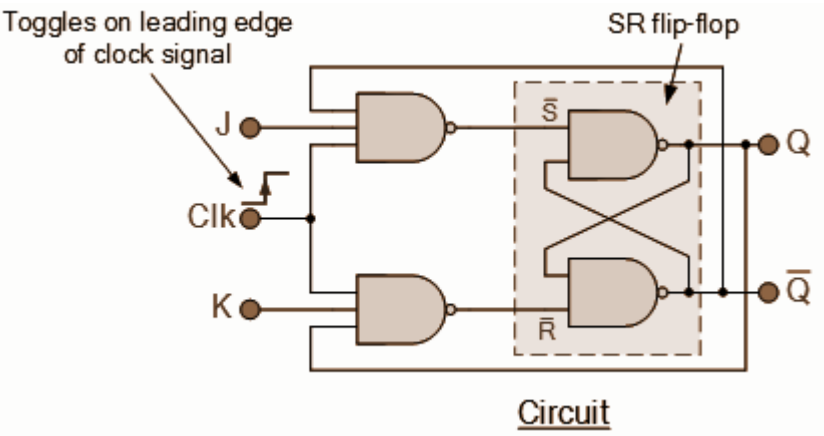
J	K	CLK	Q(t+1)	Comments
0	0	↑	Q(t)	No Change
0	1	↑	0	Reset
1	0	↑	1	Set
1	1	↑	Q(t)	Toggle

The J and K are the input states for the JK flip-flop.

The Q and Q' represents the output states of the flip-flop.

According to the table, based on the inputs, the output changes its state.

This works like SR flip-flop for the complementary inputs and the advantage is that this has toggling function.



Circuit

Check your progress:

Explain the working of JK flip-flop. How can you convert the flip-flop into D flip-flop?

T Flip-Flop :

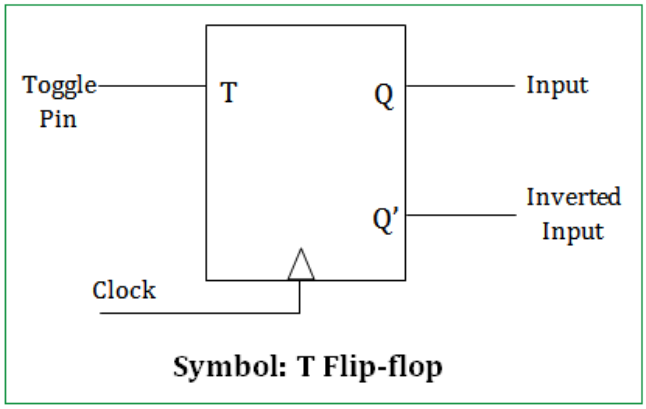
T flip-flop is termed from the nature of toggling operation.

It is used in counters and control circuits.

T flip flop is a modified form of JK flip-flop making it to operate in the toggling region.

Whenever the clock signal is LOW, the input is never going to affect the output state. The clock has to be high for the inputs to get active.

Thus, T flip-flop is a controlled Bi-stable latch where the clock signal is the control signal.



Symbol: T Flip-flop

Truth table and circuit diagram :

***Table 1.28** Truth Table for T Flip-Flop*

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

***Table 1.29** Characteristic Table for T Flip-Flop*

T	CLK	Q(t+1)	Comments
0	↑	Q(t)	No change
1	↑	Q(t)′	Toggle

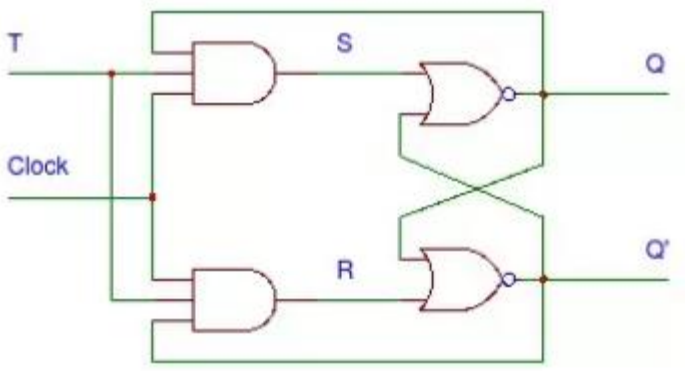
The T flip flop is the modified form of JK flip flop.

The Q and Q’ represents the output states of the flip-flop.

According to the table, based on the input the output changes its state.

This works unlike SR flip Flop & JK flip-flop for the complementary inputs.

This only has the toggling function.



Check your progress :

Define following terms:

1) CLOCK 2) CLEAR 3) PRESET 4) Positive edge trigger 5) Negative edge trigger

3.2 Counters

Asynchronous Counters and sequential circuits:

Flip-flop can be clocked in asynchronous sequential circuits.

A counter is nothing but a sequential circuit that experiences a specific sequence of states when certain input pulses are applied to the circuit.

The input pulses, also known as count pulses, can be pulses from a clock or an external source, which can occur at random or at specific intervals of time.

The sequence of states in a counter can conform to a binary or any other sequence.

A counter can be of two types:

3.2.1 Synchronous Counters change their states, and thereby their output values, at specified points of time and all at the same time.

3.2.2 Asynchronous Counters can be set or cleared when an external event occurs.

Asynchronous Sequential Circuits

A flip-flop is a basic memory element that can store one bit of information. Asynchronous sequential circuits change their states and output values whenever there is a change in input values.

In asynchronous sequential circuits, the inputs are levels and there are no clock pulses.

The input events drive the circuit. In other words, the circuit is said to be asynchronous if it is not driven by a periodic clock signal to synchronize its internal states.

For example, consider a ripple counter, which is asynchronous. In the ripple counter:

- An external clock drives the first flip-flop.
- For the remaining flip-flops, the clock for each flip-flop is driven by the output from the previous flip-flops.

Asynchronous sequential circuits used when :

- Speed of operation is important
- Response required quickly without waiting for a clock pulse
- Applied in small independent systems
- Only a few components are required
- The input signals may change independently of internal clock
- Asynchronous in nature
- Used in the communication between two units that have their own independent clocks
- Must be done in an asynchronous fashion

Exercise :

1 Write the characteristic equations for Jk and D Flip Flops.

2 Explain how it is avoided in JK master slave FF

3 How can a D flip flop be converted into T flip-flop

4 What is meant by the term edge triggered

5 Define synchronous sequential circuit

6 Define flip flop.Explain R-S flip-flop using NAND gates

7 What is race around condition?

8 "RS flip-flop cannot be converted to T flip-flop but JK flip-flop can be converted to T flip-flop". Justify this statement.

9 What is the difference between synchronous & asynchronous sequential circuits?

10 What is flip-flop? What are the different types of flip-flops?

11 Explain working of JK flip-flop with logic diagram and truth table.

12 What are T and D types of flip flops ?

13 Draw the logic diagram of clocked RS flip flop and explain its working.

14 Draw the logic diagram of clocked RS flip flop using NAND gates.

15 Explain D flop flip with proper logic diagram .

❖❖❖

4

COMPUTER SYSTEM

Unit Structure

4.0 Objective

- Understand the difference between computer architecture and organization.
- Describe the different types of computer.
- Understand the organization of computer and its various units.
- Describe the bus structures in detail and their interconnections.

4.1 Introduction to Computer Systems

The purpose of computer organization and architecture is to prepare clear and complete understanding of the nature and characteristics of modern-day computer systems. We begin this text with the overview of computer organization and architecture and structural /functional view of a computer.

It is necessary to make distinction between computer organization and architecture. Although it is difficult to give precise definitions for these terms we define them as follows:

- **Computer architecture** refers to those attributes of a system visible to a programmer. In other words, we can also say that the computer architecture refers to the attributes that have a direct impact on the logical execution of the program.
- **Computer organization** refers to the operational units and their interconnections that realize the architectural specifications.

The architectural attributes include the instruction set, data types, number of bits used to represent data types, I/O mechanism, and techniques for addressing memory. On the other hand, the organizational attributes include those hardware details transparent to programmer, such as control signals, interfaces between the computer, memory and I/O peripherals. For example, it is an architectural issue whether a computer will have a multiply and division instructions. It is an organizational issue whether to implement multiplication and division by special unit or by a mechanism that makes repeated use of the add and subtract unit to perform multiplication and division, respectively. The organizational issue may be based on which approach to be used depending on the speed of operation, cost and size of the hardware and the memory required to perform the operation.

Many computer manufactures offer a family of computer models with different price and performance characteristics. These computers have the same architecture

but different organizations. Their architectures have survived for many years, but their organizations change with changing technology. For example, IBM has introduced many new computer models with improved technology to replaced older models, offering the customer greater speed, lower cost or both. These newer models have the same architecture with advance computer organizations. The common architecture between the computer model maintains the software compatibility between them and hence protects the software investments of the customers.

In microcomputers, the relationship between computer architecture and organization is very close. In which, changes in technology not only influence organization but also result in the introduction of new powerful architecture. The RISC (Reduced Instruction Set) machine is good example of this.

4.2 Computer Types

A digital computer in its simplest form is a fast electronic calculating machine that accepts digitized information from the user, processes it according to a sequence of instructions stored in the internal storage, and provides the processed information to the user. The sequence of instructions stored in the internal storage is called **computer program**and internal storage is called **computer memory.**

According to size, cost computational power and application computers are classified as:

- Microcomputers
- Minicomputers
- Desktop computers
- Personal computers
- Portable notebook computers
- Workstations
- Mainframes or enterprise system
- Serves
- Super computers

Microcomputers: As the name implies micro-computers are smaller computers. They contain only one Central Processing Unit. One distinguishing feature of a

microcomputer is that the CPU is usually a single integrated circuit called a microprocessor.

Minicomputers: Minicomputers are the scaled up version of the microcomputers with the moderate speed and storage capacity. These are designed to process smaller data words, typically 32 bit words. These type of computers are used for scientific calculations, research, data processing application and many other.

Desktop computers: The desktop computers are the computers which are usually found on a home or office desk. They consist of processing unit, storage unit, visual display and audio as output units, and keyboard and mouse as input units. Usually storage unit of such computer consists of hard disks, CD-ROMs, and diskettes.

Personal computers: The personal computers are the most common form of desktop computers. They found wide use in homes, schools and business offices.

Portable Notebook Computers: Portable notebook computers are the compact version of personal computers. The lap top computers are the good example of portable notebook computer.

Workstations: Workstations have higher computation power than personal computers. They have high resolution graphics terminals and improved input/output capabilities. Workstations are used in engineering applications and in interactive graphics applications.

Mainframes or Enterprise System: Mainframe computers are implemented using two or more central processing units (CPU). These are designed to work at very high speeds with large data word lengths, typically 64 bits or greater. The data storage capacity of these computers is very high. This type of computers are used for complex scientific calculations, large data processing (e.g. : For creating walkthroughs with the help of animation softwares).

Servers: These computers have large storage unit and faster communication links. The large storage unit allows to store sizable database and fast communication links allow faster communication of data blocks with computers connected in the network. These computers serve major role in internet communication.

Supercomputers: These computers are basically multiprocessor computers used for the large-scale numerical calculations required in applications such as weather forecasting, robotic engineering aircraft design simulation.

4.3 Basic Organization of Computer

The computer consists of five functionally independent unit: input, memory, arithmetic and logic, output and control units. The Fig.4.1 shows thesefive functional units of a computer, and its physical locations in the computer.

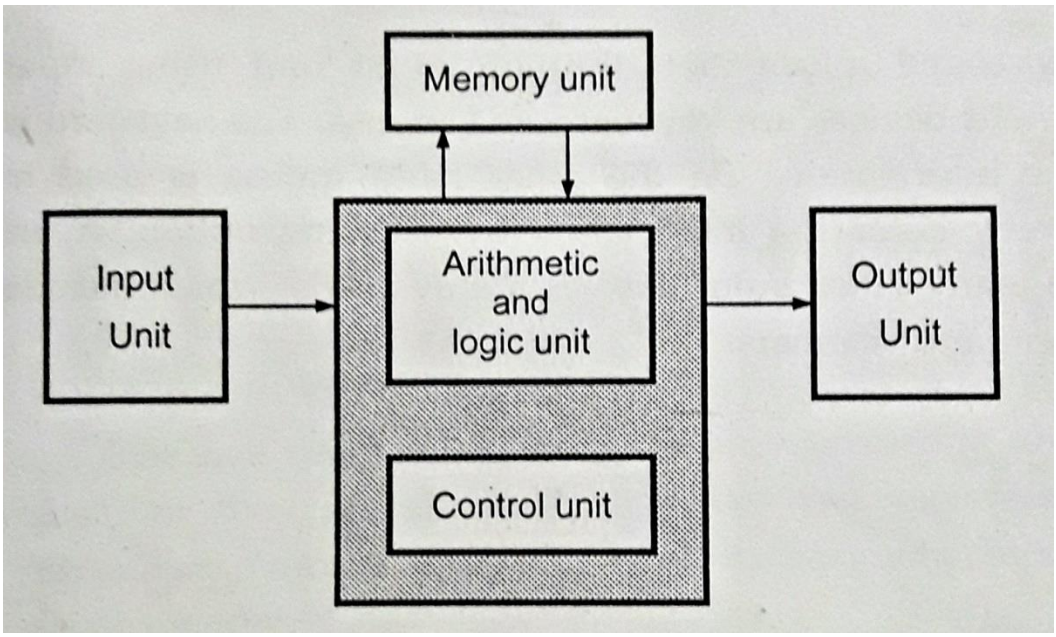


Figure 4.1 Basic functional units of a computer

The input unit accepts the digital information from user with the help of input devices such as keyboard, mouse, microphone etc. The information received from the arithmetic and logic unit to perform the desired operation. The program stored in the memory decides the processing steps and the processed output is sent to the user with the help of output devices or it is stored in the memory for later reference. All the above mentioned activities are coordinated and controlled by the control unit. The arithmetic and logic unit in conjunction with control unit is commonly called C**entral Processing Unit (CPU)**. Let us discuss the functional units in detail.

4.3.1 Input Unit

A computer accepts a digitally coded information through input unit using input devices. The most commonly used input devices are keyboard and mouse. The keyboard is used for entering text and numeric information. On the other hand mouse is used to position the screen cursor and thereby enter the information by selecting option. Apart from keyboard and mouse there are many other input devices are available, which include joysticks, trackball, space ball, digitizers and scanners.

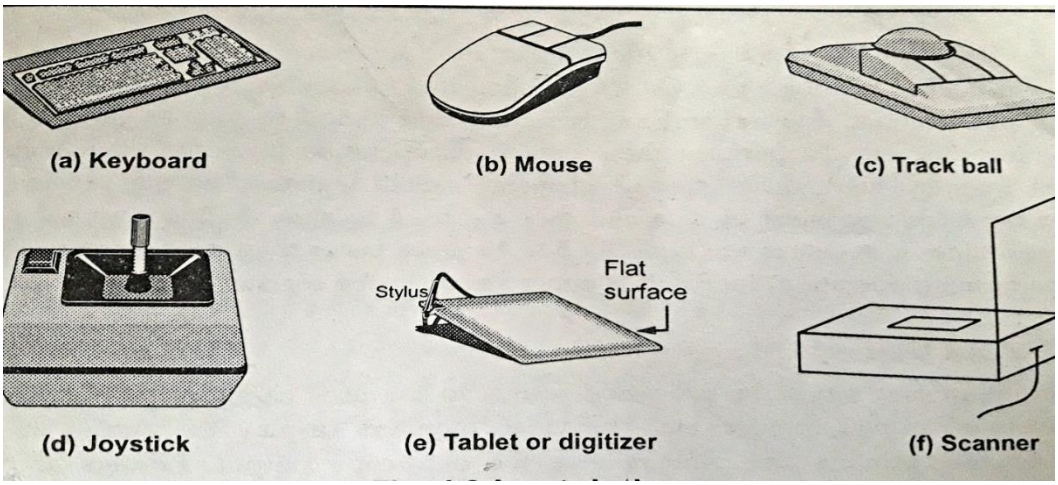


Figure 4.2 Input Devices

4.3.2 Memory Unit

The memory unit is used to store programs and data. Usually, two types of memory devices are used to from a memory unit : primary storage memory memory device and secondary storage memory device. The primary memory, commonly called main memory is a fast memory used for the storage of programs and active data (the data currently in process). The main memit consists of a large number of semiconductor storage cells, each capable of storage cells, each capable of storing one bit of information. These cells are read or written by the central processing unit in a group of fixed size called **word**. The main memory is organized such that the content of one word, contents of one word, containing n bits, can be stored or retrievedin one write or read operation, respectively

To access data from a particular word from main memory each word in the main memory has a distinct address. This allows to access any word from the main memory by specifying corresponding address. The number of bits in each word is referred to as the **word length** of the computer. Typically, the word length varies from 8 to 64 bits. The number of such word in the main memory decides the **size of memory** or **capacity of the memory**. This is one of specification of the computer. The size of computer main memory varies from few million words to tens of million words.

An important characteristic of a memory is an **access time** (the time required to access one word). The access time for main memory should be as small as possible. Typically, it is of the order of 10 to 100 nanoseconds. This accessed time also depend on the type of memory. In randomly accessed memories (**RAMs**), fixed time is required to access any word in the memory. However, In sequential access memories this time is not fixed.

The main memory consists of only randomly accessed memories. These memories are fast but they are small in the capacities and expensive. Therefore, the computer user the secondary storage memories such a magnetic tapes, magnetic disks for the storage of large amount of data.

4.3.3 Arithmetic and Logic Unit

The arithmetic and logic unit (ALU)is responsible for performing arithmetic operations such as add, subtract, division and multiplication, and logical operations such as ANDing, ORing, Inverting etc. To perform these operations operands from the main memory are brought into high speed storage element called registers of the processor.Each register can store one word of data and they are used to store frequently used operands. The access times to registers are typically 5 to 10 times faster than access times to memory. After performing operation the is either stored in the register or memory location.

4.3.4. Output Unit

The output unit sends the processed results to the user using output devices such as video monitor, printer, plotter, etc. The video monitors display the output on the CRT screen whereas printers and plotters, give the hard-copy output. Printers are classified according to their printing method: **Impact printers** and **non-impact printers.** Impact printers press formed character faces against an inked printers. Non impact printers and plotters use laser techniques, ink-jet sprays, xerographic processes, electrostatic methods, and electro thermal methods to get images onto the paper. A ink-jet printer is an example of non-impact printers.

4.3.5. Control Unit

As mentioned earlier, the control unit co-ordinates and controls the activities amongst the functional units. The basic function of control unit is to fetch the instructions stored in the main memory, identify the operations and the devices involved in it, and accordingly generate control signals to execute the desired operations.

The control unit uses control signals to determine when a given action is to take place. It controls input and output operations, data transfers between the processor, memory and input/output devices using timing signals.

The control and the arithmetic and logic units of a computer are usually many times faster than other devices connected to a computer system. This enables them to control a number of external input/output devices.

4.4 Basic Operational Concept

The basic function of computer is to execute program, sequence of instructions. These instructions are stored in the computer memory. The instructions are executed to process data which already loaded into the computer memory through input unit. After processing the data, the result is either stored back into the computer memory for further reference or it is sent to theoutside world through the output port. Therefore all functional units of the computer contribute to execute program. Let us summarize the functions of different computer units

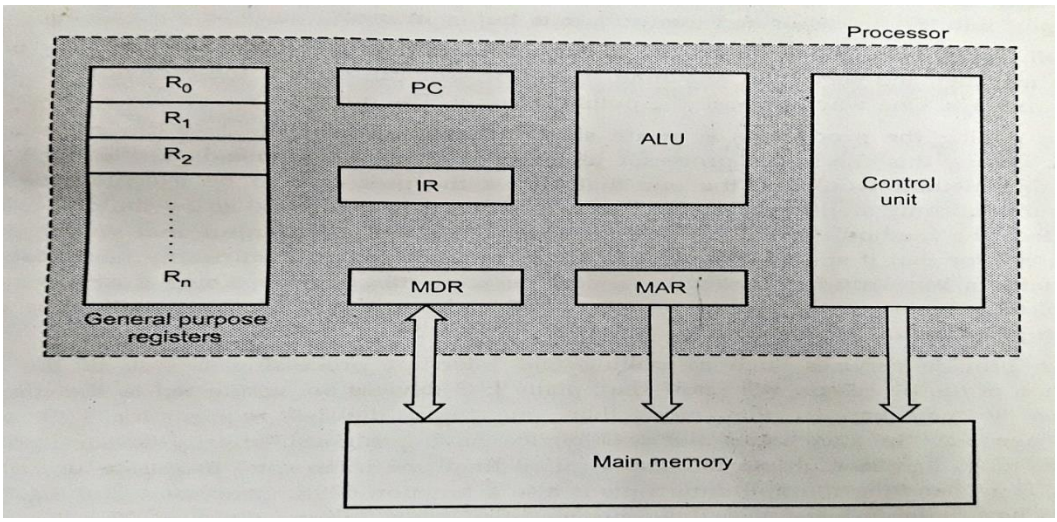


Figure 4.3 Connections between the processor and the main memory

- The input unit accepts data and instructions from the outside world to machine. It is operate by control unit.
- The memory unit stores both, data and instructions.
- The arithmetic-logic unit performs arithmetic and logical operations.
- The control unit fetches and interprets the instructions in memory and causes them to be executed.
- The output unit transmits final results and messages to the outside world.

To perform execution of instruction, in addition to the arithmetic logic unit, and data and some special function registers, as shown in Figure 4.3. The special function registers include program counter (PC), instruction register (IR), memory address register (MAR) and memory data register (MDR).

The program **counter** is one of the most important registers in the CPU.As mentioned earlier; a program is a series of instructions stored in the memory. These

instructions tell CPU exactly how to get the desired result. It is important that these instructions must be executed in a proper order to get the correct result. This sequence of instruction execution is monitored by the program counter. It keeps track of which instruction is being executed and what the next instruction will be.

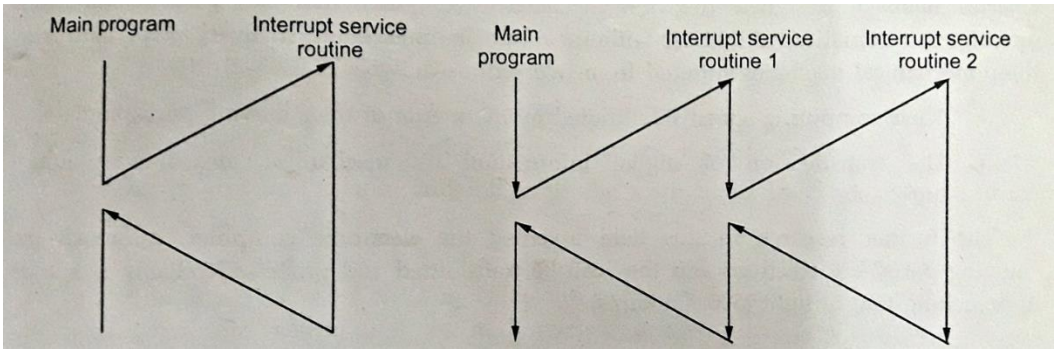
The instruction register (IR) is used to hold the instruction that is currently being executed. The contents of IR are available to the control unit, which generate the timing signals that control the various processing elements involved in executing the instruction.

The two registers MAR and MDR are used to handle the data transfer between the main memory and the processor. The MAR holds the address of the main memory to or from which data is to transferred. The MDR contains the data to be written into or read from the addressed word of the main memory.

Sometimes it is necessary to have computer system which can automatically execute one of the collections of special routines whenever certain conditions exist within a program or in the computer system. E.g. It is necessary that computer system should give response to devices such as keyboard, sensor and other components when they request for service. When the processor is asked to communicate with devices, we say that the processor is servicing the devices. For example, each time when we type a character on a keyboard, a keyboard service routine is called. It transfers the character we type from the keyboard I/O port into the processor and then to a data buffer in memory.

When you have one or more I/O devices connected to a computer system, any one of them may demand service at ant time. The processor can service these devices in one of the two ways. One way is to use the polling routine. The other way is to use an interrupt.

In polling the processor's software simply checks each of the I/O devices every so often. During this check, the processor tests to see if any device needs servicing. A more desirable method would be the one that allows the processor to be executing its main program and only stop to service I/O devices when it is told do so by the device itself. In effect, the method, would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed and fetch a new routine that will service the requesting device. Once this servicing is completed, the processor would resume exactly where it left off. This method is called **interrupt method.**



(a) Program flow for single interrupt (b)Program for nested interrupt

Figure 4.4 : Program for interrupts

To provide services such as polling and interrupt processing is one of the major functions of the processor. We known that, many I/O devices are connected to the computer system. It may happen that more than one input devices request for I/O service simultaneously. In such cases the I/O device having highest priority is serviced first. Therefore, to handle multiple interrupts (more than one interrupts) processor use priority logic. Thus, handling multiple interrupts is also a function of the processor. The Figure 4.4.(a) shows how program execution flow gets modified when interrupt occurs. The Figure 4.4 (b) shows that interrupt service routine itself can be interrupted by higher priority interrupt. Processing of such interrupt is called **nested interrupt processing** and interrupt is called **nested interrupt.**

We have seen that the processor provides the requested service by executing an appropriate interrupt service routine. However, due to change in the program sequence, the internal state of the processor may change and therefore, it is necessary to save it in the memory before servicing the interrupt. Normally, contents of PC, the general registers, and some control information are stored in memory. When the interrupt service routine is completed, the state of the processor is restored so that the interrupted program may continue. Therefore, saving the state of the processor at the time of interrupt is also one of the function of the computer system.

Let us see few examples which will make you more easy to understand the basic operations of a computer.

Example 4.1 : State the operations involved in the execution of ADD R_1,R_0 instruction.

Solution: The instruction Add R_1,R_0 adds the operand in R_1 register to the operand in R_0 register and stores the sum into R_0 register. Let us see the steps involved in the execution of this instruction.

1. Fetch the instruction from the memory into IR register of the processor.
2. Decode the instruction.
3. Add the contents of R_1 and R_0 and store the result in the R_0.

Example 4.2: state the operations involved in the execution of Add LOCA, R_0.

Solution: The instruction Add LOCA, R_0 and stores result in the register R_0. The steps involved in the execution of this instruction are:

1. Fetch the instruction from the memory into the IR register of the processor.
2. Decode the instruction.
3. Fetch the second operand from memory location LOCA and add the contents of LOCA and the contents of register R_0.
4. Store the result in the R_0.

4.5 Bus Structures

We knowthat the central processing unit, memory unit and I/O unit are the hardware components/modules of the computer. They work together with communicating each other various modules is called the **interconnection structure**. The design of this interconnection structure will depend on the exchanges that must be made between modules. A group of wires, called **bus** is used to provide necessary signals for communication between modules. A bus that connects major computer components/ modules (CPU, memory I/O) ia called a **system bus.** The system bus is a set of conductors that connects the CPU, memory and I/O modules. Usually, the system bus is separated into three functional groups:

- Data Bus
- Address Bus
- Control Bus

1. Data Bus:

The data bus consists of 8, 16, 32 or more parallel signal lines. The data bus lines are bi-directional. This means that CPU can read data on these lines from memory

or from a ports, as well as send data out on these lines to a memory location or to a port. The data bus is connected in parallel to all peripherals. The communication between peripheral and CPU is activated by giving output enable pulse to the peripheral. Outputs of peripherals are floated when they are not in use.

2. Address Bus:

It is an unidirectional bus. The address bus consists of 16, 20, 24 or more parallel signal lines. On these lines the CPU sends out the address of the memory location or I/O port that is to be written to or read from.

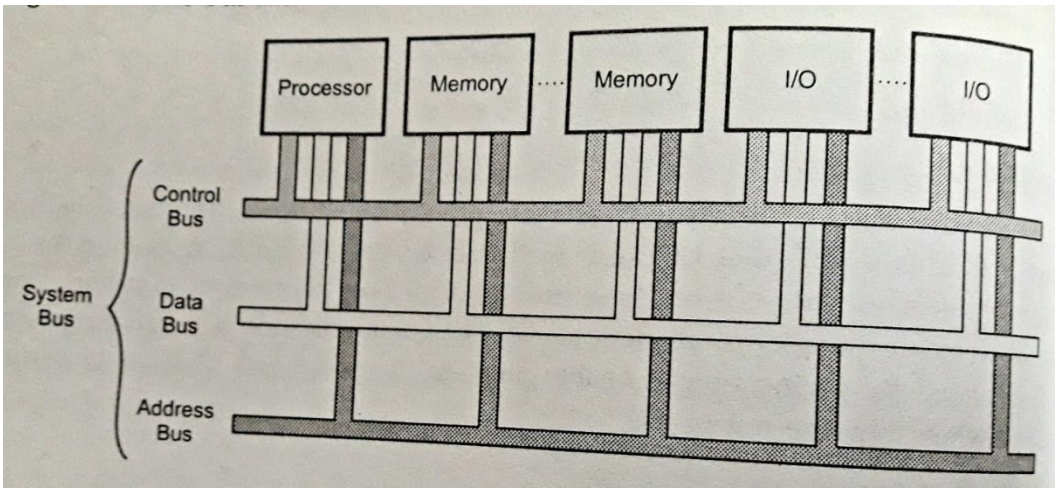


Figure 4.5 Bus Interconnection scheme

3. Control Bus:

The control lines regulate the activity on the CPU sends on the control bus to enable the outputs of addressed memory devices or port devices.

Typical control bus signals are:

- Memory Read (MEMR)
- Memory Write (MEMW)
- I/O Read (IOR)
- I/O Write (IOW)
- Bus Request (BR)
- Bus Grant (BG)
- Interrupt Request (INTR)
- Interrupt Acknowledge (INTA)

- Clock (CLK)
- Reset
- Ready
- Hold
- Hold Acknowledge (HLDA)

4.5.1 Single Bus Structure

Another way to represent the same bus connection scheme is shown in Fig. Here, address bus, data bus and control bus are shown by single bus called system bus. Hence such interconnection bus structure is called **single bus structure.**

In a single bus structure all units are connected to common bus called system bus. However, with single bus only two units can communicate with each other at a time. The bus control lines are used to arbitrate multiple requests for use of the bus. The main advantage of single bus structure is its low cost and its flexibility for attaching peripheral devices.

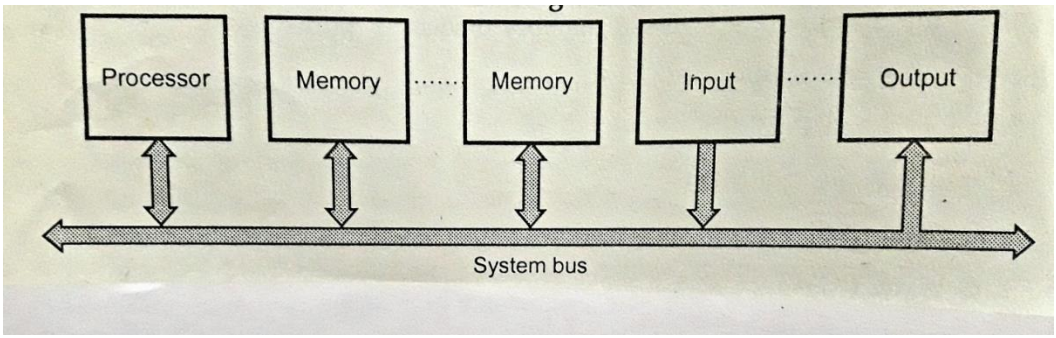


Figure 4.6 Single Bus Structure

The complexity of bus control logic depends on the amount of translation needed between the system bus and CPU, the timing requirements, whether or not interrupt management is included and the size of the overall system. For a small system, control signals of the CPU could be used directly to reduce handshaking logic. Also, drives and receivers would not be needed for the data and address lines. But large system, with several interfaces would need bus driver and receiver circuits connected to the bus in order to maintain adequate signal quality. In most of the processors, multiplexed address and data buses are used to reduce the number of pins. During first part of bus cycle, address is present on this bus. Afterwards, the same bus is used for data transfer purpose.

So latches are required to hold the address sent by the CPU initially. Interrupt priority management is optional in a system. It is not required in systems which use software priority management. The complex system includes hardware for managing the I/O interrupts to increase efficiency of a system. Many manufacturers have made priority management devices. Programmable interrupt (PIC) is the IC designed to fulfill the same task.

4.5.2 Multiple Bus Structures

The performance of computer system suffers when large numbers of devices are connected to the bus. This is because of the two major reasons:

- When more devices are connected to the common bus we need to share the bus amongst these devices. The sharing mechanism coordinates the use of bus to different devices. This coordination requires finite time called **propagation delay.** When control of the bus passes from one device to another frequently, these propagation delays are noticeable and affect the performance of computer system.

- When the aggregate data transfer demand approaches the capacity of the bus, the bus may become a bottleneck. In such situations we have to increase the data rate of the bus or we have to use wider bus.

Now a-days the data transfer rate for video controllers and network interfaces are growing rapidly. The need of high speed shared bus is impractical to satisfy with a single bus. Thus, most computer system uses the multiple buses. These buses have the hierarchical structure.

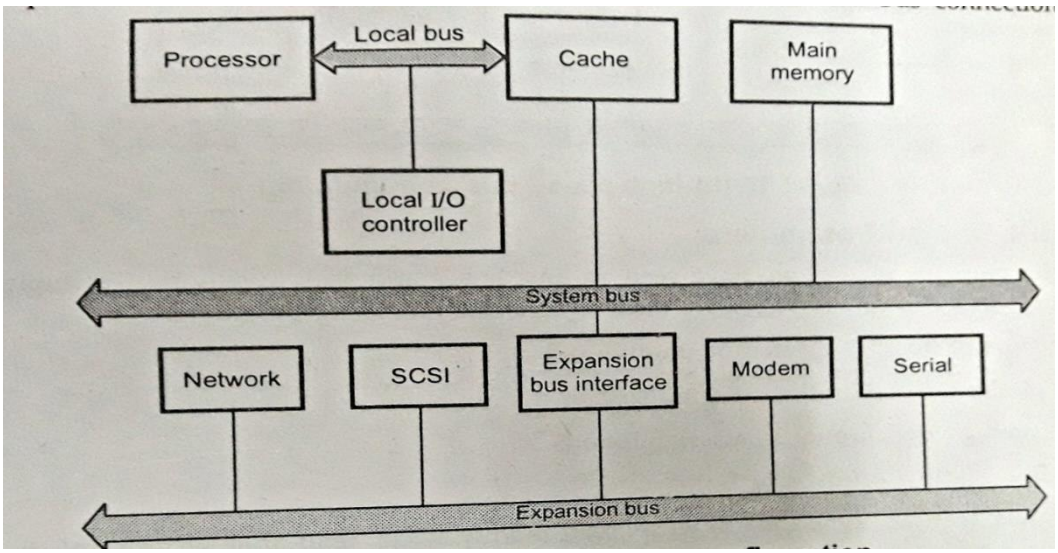


Figure 4.7 (a) Traditional Bus Configuration

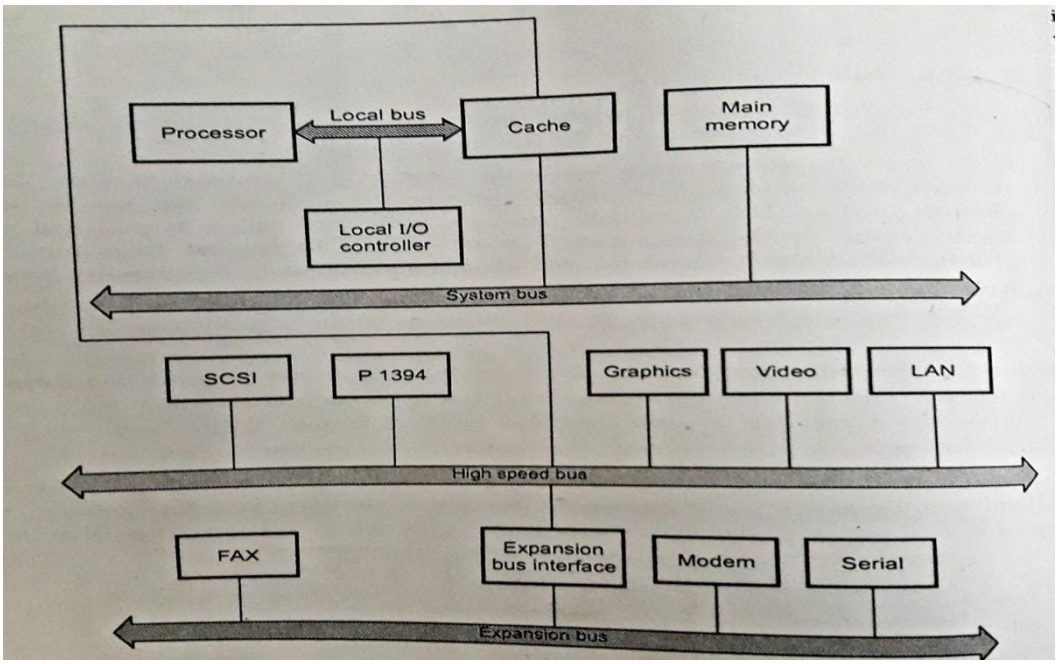


Figure 4.7(b) High-speed Bus Configuration

Figure 4.7 (a) shows two bus configurations. The traditional bus connection uses three buses: local bus, system bus and expanded bus. The high speed bus configuration uses high-speed bus along with the three buses used in the traditional bus connection. Here, cache controller is connected to high-speed bus. This bus supports connection high-speed LANs, such as Fiber Distributed Data Interface (FDDI), video and graphics workstation controllers, as well as interface controllers to local peripheral buses including SCSI and P1394.

4.5.3 Bus Design parameters

In the previous section we have seen various buses and their functions. To implement these buses in a computer system we have to consider following parameters:

- Type of Bus : Dedicated or multiplexed
- Method of Arbitration : Centralized or distributed
- Timing : Synchronous or asynchronous
- Bus Width : Address or Data
- Data Transfer Type: Read, write, read modify write, read-after write or block.

4.5.3.1 Bus Types

When a particular bus is permanently bus is permanently assigned to one function or to a physical subset of computer component, it is called **dedicatedbus.** As an example of functional dedication is the use of separate dedicate address and data lines. Physical dedication gives high throughout, because there is less bus contention. A disadvantage is the increased size and cost.

When bus is used for more than one function at different time zones, it is called **multiplexedbus.** An example of multiplexed bus is the use of same bus for address and data. At the beginning of data transfer, the address is placed on the bus along with the address latch enable signal (ALE). At this point module has sufficient time to latch the address with latch ICs. The address is then removed from the bus and the same bus is used for data transfer.

4.5.3.2 Method of Arbitration

In a computer system there may be more than one bus masters such as processor, DMA controller, etc. They share the system bus. In a centralized approach, a single hardware device, referred to as a bus controller or arbiter, is responsible for allocating time on the bus for bus masters. In the distributed approach each bus master contains access control logic and they act together to share the bus.

4.5.3.3 Bus Timings

Bus timing refers to the way in which events are coordinated on the bus. In synchronous timing, the occurrence of events on the bus is determined by the clock whereas in the asynchronous timing, the occurrence of events on the bus is determined by the previous events on the bus.

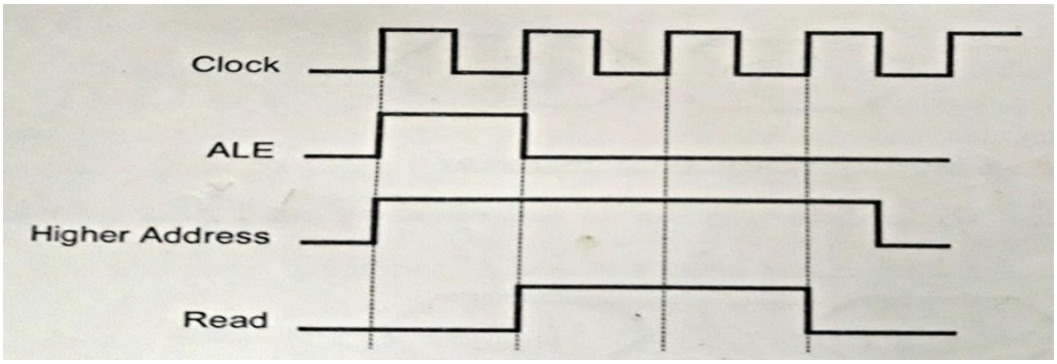


Figure 4.8 (a) Synchronous Timings

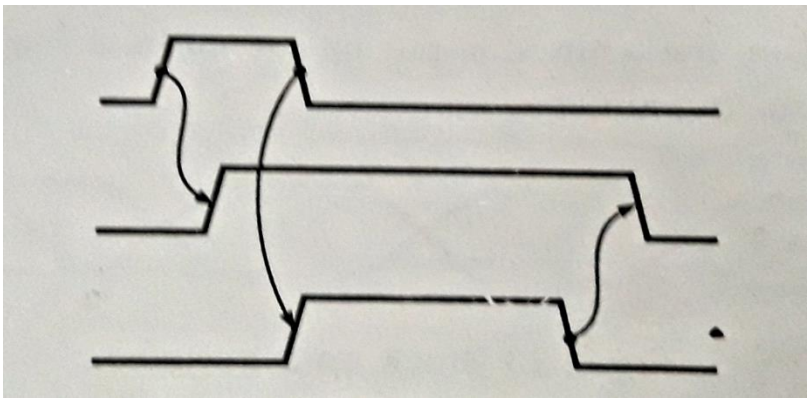


Figure 4.8 (b) Asynchronous Timings

Figure 4.8 shows synchronous and asynchronous timings.

4.5.3.4 Bus Width

Bus width is nothing but the number of lines which are grouped to form a bus. Data bus width decides the number of bits transferred at one time, whereas address bus decides the range of location that can be accessed.

4.5.3.5 Data Transfer type

As mentioned earlier, there are various data transfer types such as read, write, read-modify write, read after write and block.

Read: In this operation address is put on the address bus and after sufficient access time data is available on the data bus. Figure.4.9 shows read data transfer for multiplexed address and data bus.

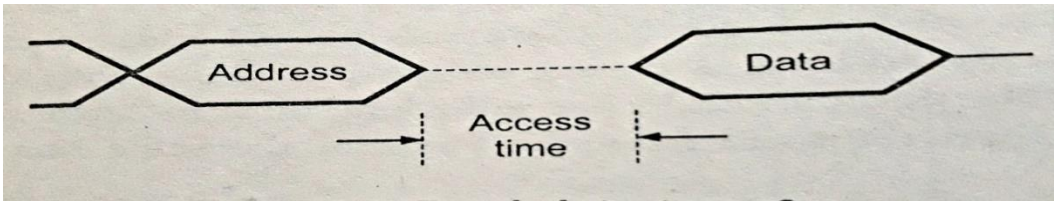


Figure 4.9 Read Data Transfer

Write: In this operation address is put on the multiplexed bus and then immediately after latching period data is put on the multiplexed bus, as shown in the Figure 4.10

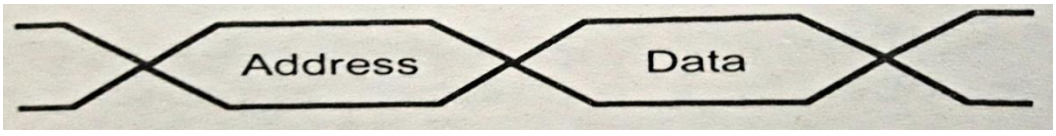


Figure 4.10 Write Data Transfer

Read Modify Write: In this operation, read data transfer is followed by write data transfer at the same address.

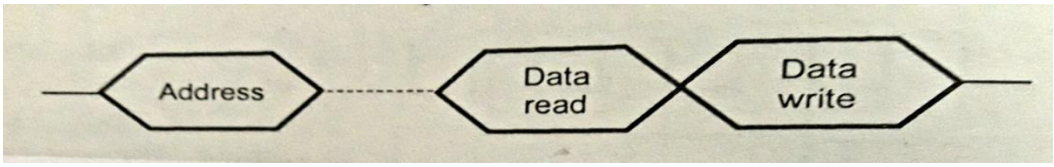


Figure4.11: Read –Modify-Write Data Transfer

Read-After Write: In this operation, write data transfer is followed by read data transfer at the same address.

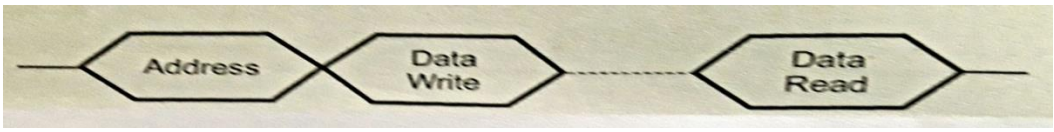


Figure4.12: Read –After-Write Data Transfer

Block: In this operation, the address is put on the address bus and then the n numbers of data transfer cycles are carried out.

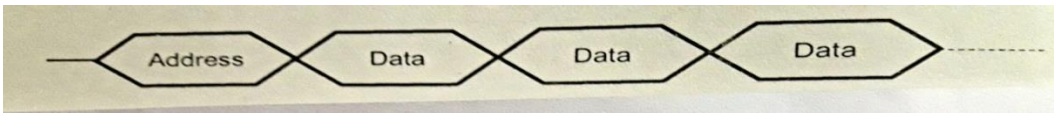


Figure 4.13: Block Data Transfer

4.6 Summary

Computer architecturerefers to those attributes of a system visible to a programmer. In other words, we can also say that the computer architecture refers to the attributes that have a direct impact on the logical execution of the program.

Computer organizationrefers to the operational units and their interconnections that realize the architectural specifications.

The architectural attributes include the instruction set, data types, number of bits used to represent data types, I/O mechanism, and techniques for addressing memory.

The organizational attributes include those hardware details transparent to programmer, such as control signals, interfaces between the computer, memory and I/O peripherals.

According to size, cost computational power and application computers are classified as:Microcomputers, Minicomputers,Desktop computers, Personal computers, Portable notebook computers,Workstations, Mainframes or enterprise system, Serves, Super computers.

The computer consists of five functionally independent unit: input, memory, arithmetic and logic, output and control units.

The basic function of computer is to execute program, sequence of instructions. These instructions are stored in the computer memory. The instructions are executed to process data which already loaded into the computer memory through input unit. After processing the data, the result is either stored back into the computer memory for further reference or it is sent to the outside world through the output port. Therefore all functional units of the computer contribute to execute program.

The central processing unit, memory unit and I/O unit are the hardware components/modules of the computer. They work together with communicating each other various modules is called the interconnection structure.

In a single bus structure all units are connected to common bus called system bus. However, with single bus only two units can communicate with each other at a time.

The traditional bus connection uses three buses: local bus, system bus and expanded bus. The high speed bus configuration uses high-speed bus along with the three buses used in the traditional bus connection. Here, cache controller is connected to high-speed bus. This bus supports connection high-speed LANs, such as Fiber Distributed Data Interface (FDDI), video and graphics workstation controllers.

When a particular bus is permanently bus is permanently assigned to one function or to a physical subset of computer component, it is called dedicated bus.

When bus is used for more than one function at different time zones, it is called multiplexed bus.

Sample Questions

1. Commenton : computer architecture and computer organization.

2. Discuss various functions of a computer.

3. List the structural components of a computer.
4. Draw and explain the block diagram of a simple computer with five functional units.
5. Explain the function of each functional unit in the computer system.
6. Explain the single line bus structure.
7. Explain the multiple bus structure.
8. Draw and explain the timing diagram for synchronous input data transfer.
9. Draw and explain the timing diagram for synchronous output data transfer.

Reference Books

1. Computer Organization & Architecture, William Stallings, 8e, Pearson Education.
2. Computer Architecture& Organization, John P. Hayes, 3e, Tata McGraw Hill.
3. Computer Organization, 5e, Carl Hamacher, ZconkoVranesic&SafwatZaky, Tata McGraw-Hill.
4. Computer Architecture & Organization, Nicholas Carter, McGraw Hill
5. Computer System Architecture, M.MorrisMano ,Pearson Education.

❖❖❖

5

I/O ORGANIZATION

Unit Structure

5.0 Objective

- Describe input / output interface and devices
- Explain the significance of I/O channels and processors

5.1 Introduction

The important components of any computer system are processor, memory, and I/O devices (peripherals). The processor fetches instructions (opcodes and operands/data) from memory, processes them and stores results in memory. The other components of the computer system (I/O devices) may be loosely called the Input/ Output system. The main function of I/O system is to transfer information between processor or memory and the outside world.

The important point to be noted here is, I/O devices (peripherals) cannot be connected directly to the system bus. The reasons are discussed here.

- A variety of peripherals with different methods of operation are available. So it would be impractical to incorporate the necessary logic within the processor to control a range of devices.
- The data transfer rate of peripherals is often much slower than that of the memory or processor. So it is impractical to use the high speed system bus to communicate directly with the peripherals.
- Generally, the peripherals used in a computer system have different data formats and word lengths than that of processor used in it.

So to overcome all these difficulties, it is necessary to use a module in between system bus and peripherals, called **I/O module** or **I/O system.**

The Input /Output system has two major functions:

- Interface to the processor and memory via the system bus,
- Interface to one or more I/O devices by tailored data links

For clear understanding refer Figure 5.1, Links to peripheral devices are used to exchange control, status and data between the I/O system and the external devices.

An important point that is to be noted here is, an I/O system is not simply mechanical connectors for connecting different devices required in the system to the system bus.

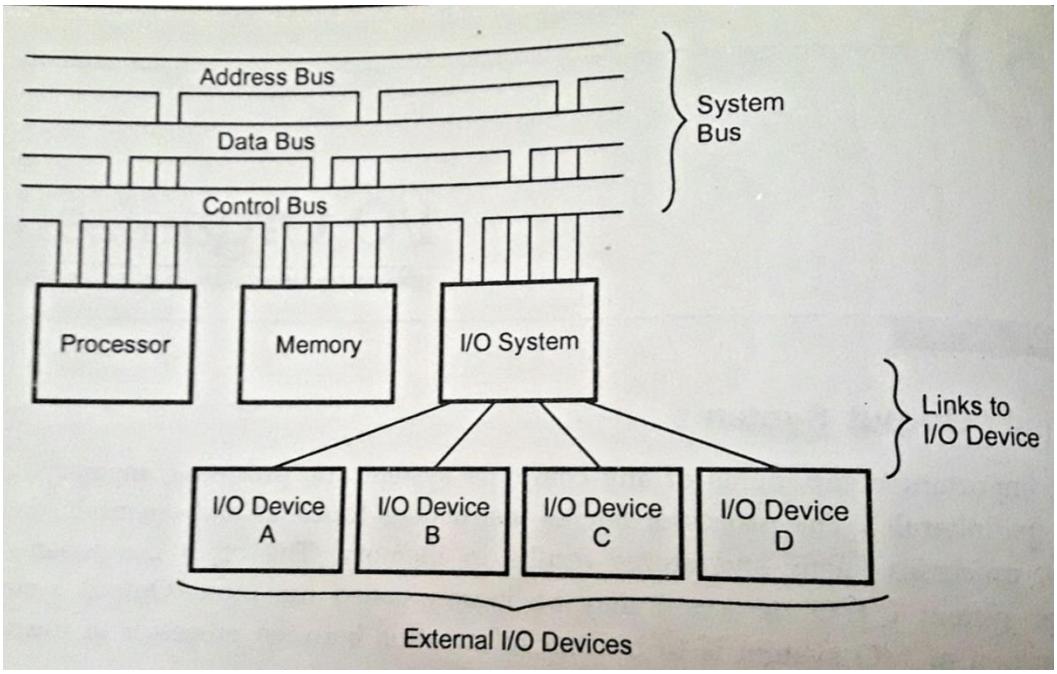


Figure 5.1 Generic Model of computer system

It contains some logic for performing function of communication between the peripheral (I/O device) and the bus. The I/O system must have an interface internal to the computer (To the processor and memory) and an interface external to the computer (to the external I/O devices).

5.1.1 Requirements of I/O System

The I/O system if nothing but the hardware required to connect an I/O device to the bus. It is also called I/O interface. The major requirements of an I/O interface are:

1. Control and timing
2. Processor communication
3. Device communication
4. Data buffering
5. Error detection

All these requirements are explained here. The I/O interface includes a control and timing requirements to co-ordinate the flow of traffic between internal resources (memory system bus) and external devices. Processor communication involves different types of signal transfers such as

- Processor sends commands to the I/O system which are generally the control signals on the bus.
- Exchange of data between the processor and the I/O interface over the data bus.
- The data transfer rate of peripherals is often much slower than that of the processor. So it is necessary to check whether peripheral is ready or not for data transfer. If not, processor must wait. So it is important to know the status of I/O interface. The status signals such as BUSY, READY can be used for this purpose.
- A number of peripheral devices may be connected to the I/O interface. The I/O interface controls the communication of each peripheral with processor. So it must recognize one unique address for each peripheral connected to it.

The I/O interface must able to perform device communication which involves commands,status information and data.

Data buffering is also an essential task of an I/O interface. Data transfer rates of peripheral devices are quite high than that of processor and memory. The data coming from memory or processor are sent to an I/O interface, buffered and then sent to the peripheral device at its data rate. Also, data are buffered in I/O interface so as not to tie up the memory in a slow transfer operation. Thus the I/O interface must be able to operate at both peripheral and memory speeds.

I/O interface is also responsible for error detection and for reporting error detection and for reporting errors to the processor. The different types of errors are mechanical, electrical malfunctions reported by the device such as bad disk track, unintentional changes to the bit pattern, transmission errors etc. To fulfill all these requirements the important blocks necessary in any I/O interface are shown in Figure 5.2.

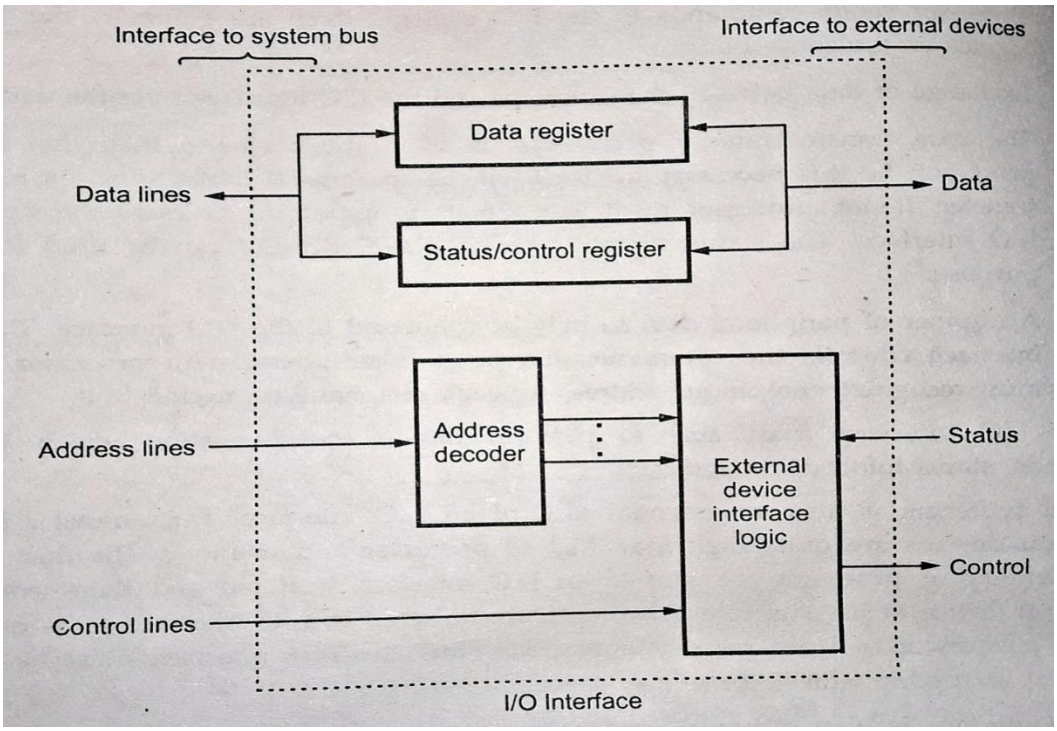
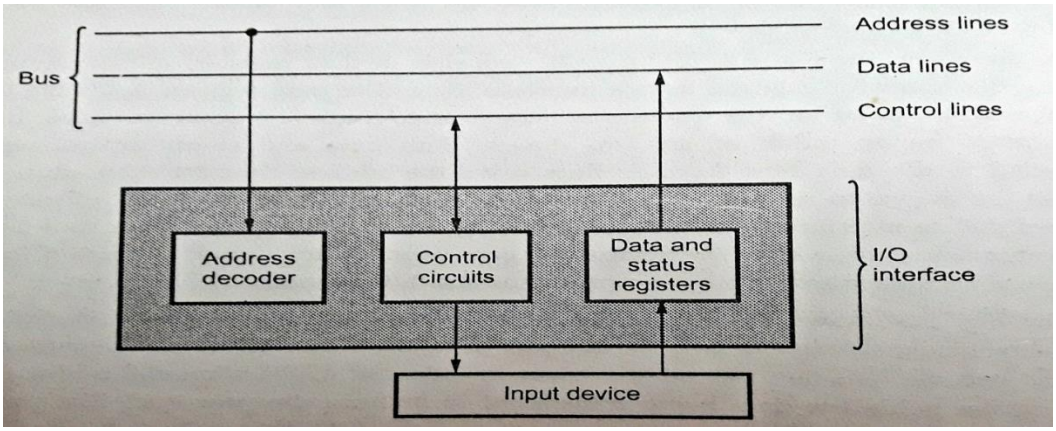
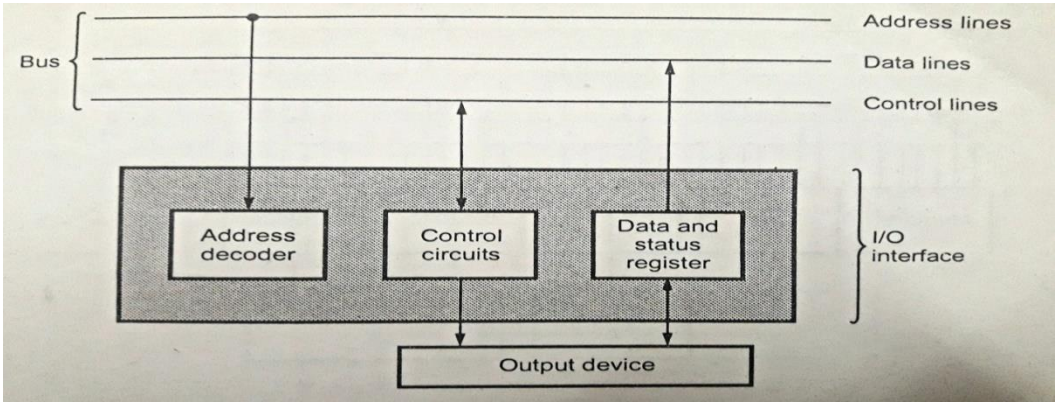


Figure 5.2: Block diagram of I/ O interface

As shown in the Figure 5.2, I/O interface consists of data register, status control register, address decoder and external device interface logic. The data register holds the data being transferred to or from the processor. The status/control register contains information relevant to the operation of the I/O device. Both data and status /control registers are connected to the data bus. Address lines drive the address decoder. The address decoder enables the device to recognize its address when address appears on the address lines. The external device interface logic accepts inputs from address decoder, processor control lines and status signal from the I/O device and generates control signals to control the direction and speed of data transfer between processor and I/O devices.



(a) I/O interface for input device



(b) I/O interface for output device

Figure 5.3 –I /O Interface

The Figure 5.3.shows the I/O interface for input device and output device. Here, for simplicity block schematic of I/O interface is shown instead of detail connections. The address decoder enables the device when its address appears on the address lines. The data register holds the data being transferred to or from the processor. The status register contains information relevant to the operation of the I/O device. Both the data and status registers are assigned with unique addresses and they are connected to the data bus.

5.1.2I/O Interfacing Techniques

The most of the processors support isolated I/O system. It partitions memory from I/O, via software, by having instructions that specifically access (address) memory and others that specifically access I/O. When these instructions are decoded by the processor, an appropriate control signal is generated to active either memory or I/O operation. I/O devices can be interfaced to a computer system I/O in two ways, which are called interfacing techniques,

- Memory mapped I/O
- I/O mapped I/O

Memory mapped I/O

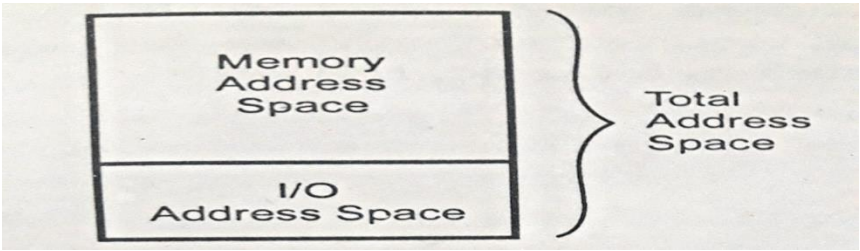


Figure 5.4 Address space

In this technique, the total memory address space is partitioned and part of this space is devoted to I/O addressing as shown in Figure 5.4

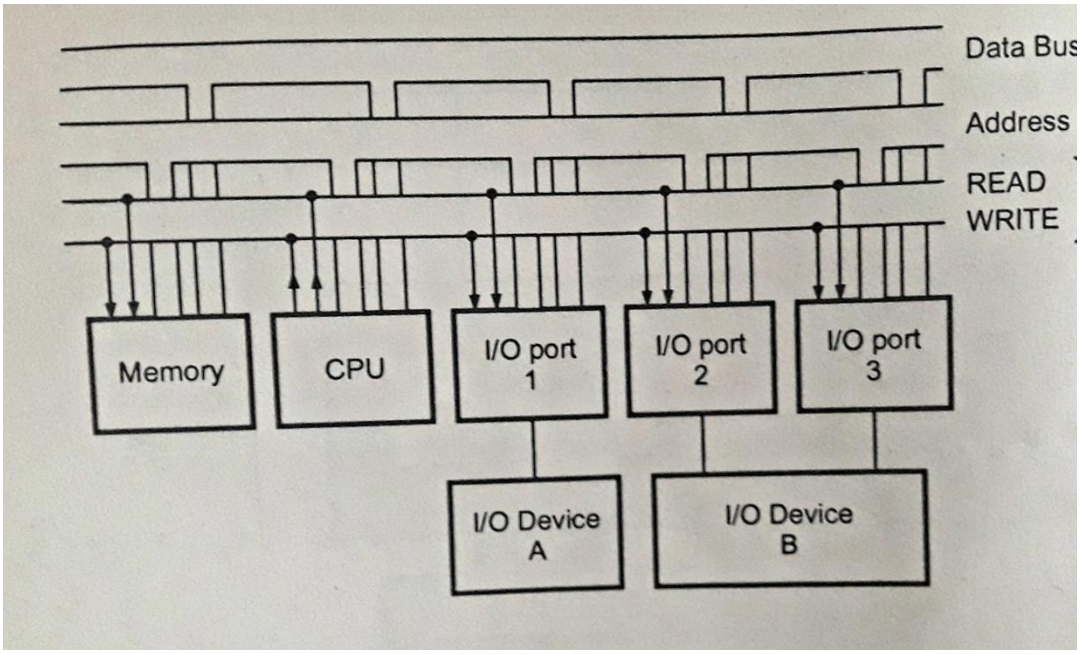


Figure 5.5 Programmed I/ O with memory mapped I /O

When this technique is used, a memory reference instruction that causes data to be fetched from or stored at address specified, automatically becomes an I/O instruction if that address is made the address of an I/O port. The usual memory related instructions are used for I/O instructions are not required Figure 5.5. gives the I/O with memory I/O.

I/O mapped I/O

If we do not want to reduce the memory address space, we allot a different I/O address space, apart from total memory space which is called I/O mapped I/O technique as shown in Figure 5.6

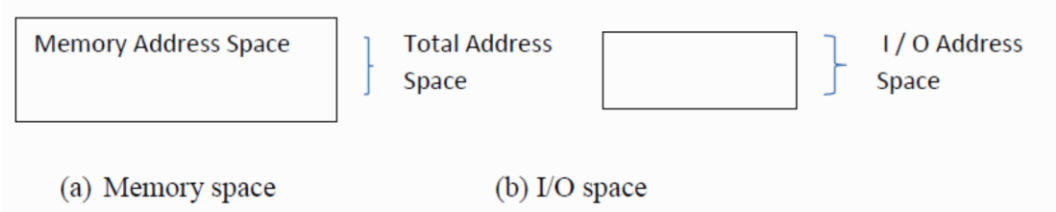


Figure 5.6 Address space

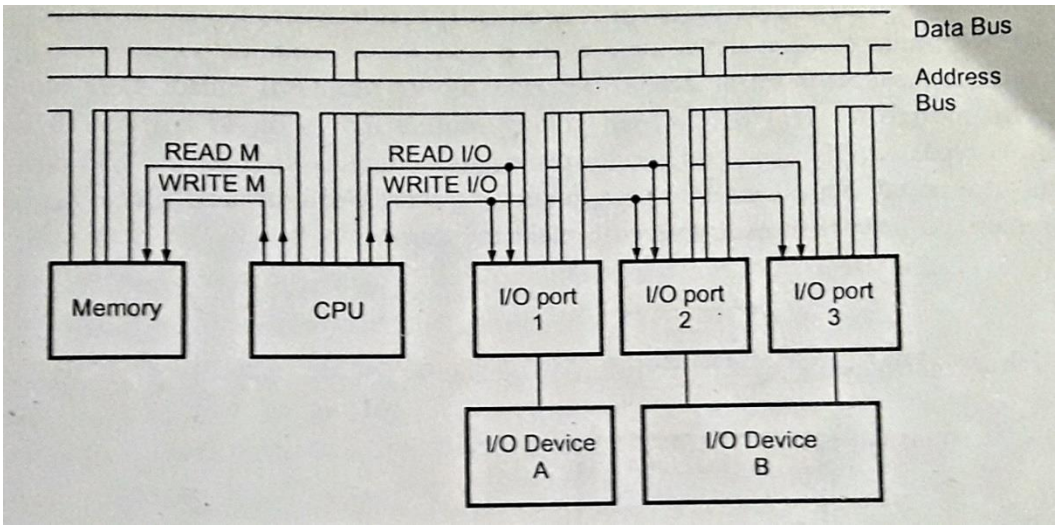


Figure 5.7 gives I/O with I/O mapped I/O technique

Here, the advantage is that the full memory address space is available. But now, the memory related instructions do not work. Therefore, processor can only use this mode if it has special instructions for I/O related operations such as I/O read, I/O write.

Sr.No	Memory mapped I/O	I/O mapped I/O
1	Memory and I/O share the entire address range of processor	Processor provides separate address range for memory and IO devices
2	Usually processor provides more address lines for accessing memory. Therefore more decoding is required control signals.	Usually processor provides less address lines for accessing I/O . Therefore, less decoding is required.
3	control signals are used to control read and write I/O operations.	IO control signals are used to control read and write I / O operations.

5.2 Programmed I/O

I/O operations will mean a data transfer between an I/O device and memory or between an I/O device and the processor. If in any computer system I/O operations are completely controlled by the processor, then that system is said to be using '**Programmed I/O'**. When such a technique is used, processor executes programs

that initiate, direct and terminate the I/O operations, including sensing device status, sending a read or write command and transferring the data. It is the responsibility of the processor to periodically check the status of the I/O system until it finds that the operation is complete. Let us consider the following example.

The processor's software checks each of the I/O devices every so often. During this check, the microprocessor tests to see if any device needs servicing. Figure 5.8 shows the flow chart for this. This is simple program which services I/O ports A,B and C. The routine checks the status of I/O ports in proper sequence. It first transfers the status of I/O port A into the accumulator. Then the routine block checks the contents of accumulator to see if the service request bit is set. If it is, I/O port A service routine is called. After completion of service routine for I/O port A, the polling routine moves on to test port B and the process is repeated. This test and service procedure continues until all the I/O port status registers are tested and all the I/O ports requesting service are serviced. Once this is done, the processor continues to execute the normal programs.

The routine assigns priorities to the different I/O devices. Once the routine is started, the service request bit at port. A is always checked first. Once port A is checked, port B is checked, and then port C. However, the order can be changed by simply changing the routine and thus the priorities.

When programmed I/O technique is used, processor fetches I/O related instructions from memory and issues I/O commands to I/O system to execute the instruction. The form of the instruction depends on the technique used for I/O addressing i.e. memory mapped I/O or I/O mapped I/O.

As seen before, using memory mapped I/O technique, a memory reference instruction that causes data to be fetched from or stored at address specified automatically becomes an I/O instruction if that address is devoted to an I/O port. The usual memory load and store instructions if that address is devoted to an I/O port. The usual memory load and store instructions are used to transfer a word of data to an I/O port.

When I/O mapped I/O technique is used, separate I/O instructions are required to activate READ I/O and WRITE I/O lines, which cause a word to be transferred between the addressed I/O port and the processor. The processor has two separate instructions, IN and OUT for I/O data transfer, e.g. the Intel 8085 microprocessor has two main I/O instructions. The IN port address instruction causes a word to be transferred from I/O port having address specified within the instruction to register A (accumulator) of 8085. The instruction OUT port address transfers a word from register A to I/O port having address specified within the instruction.

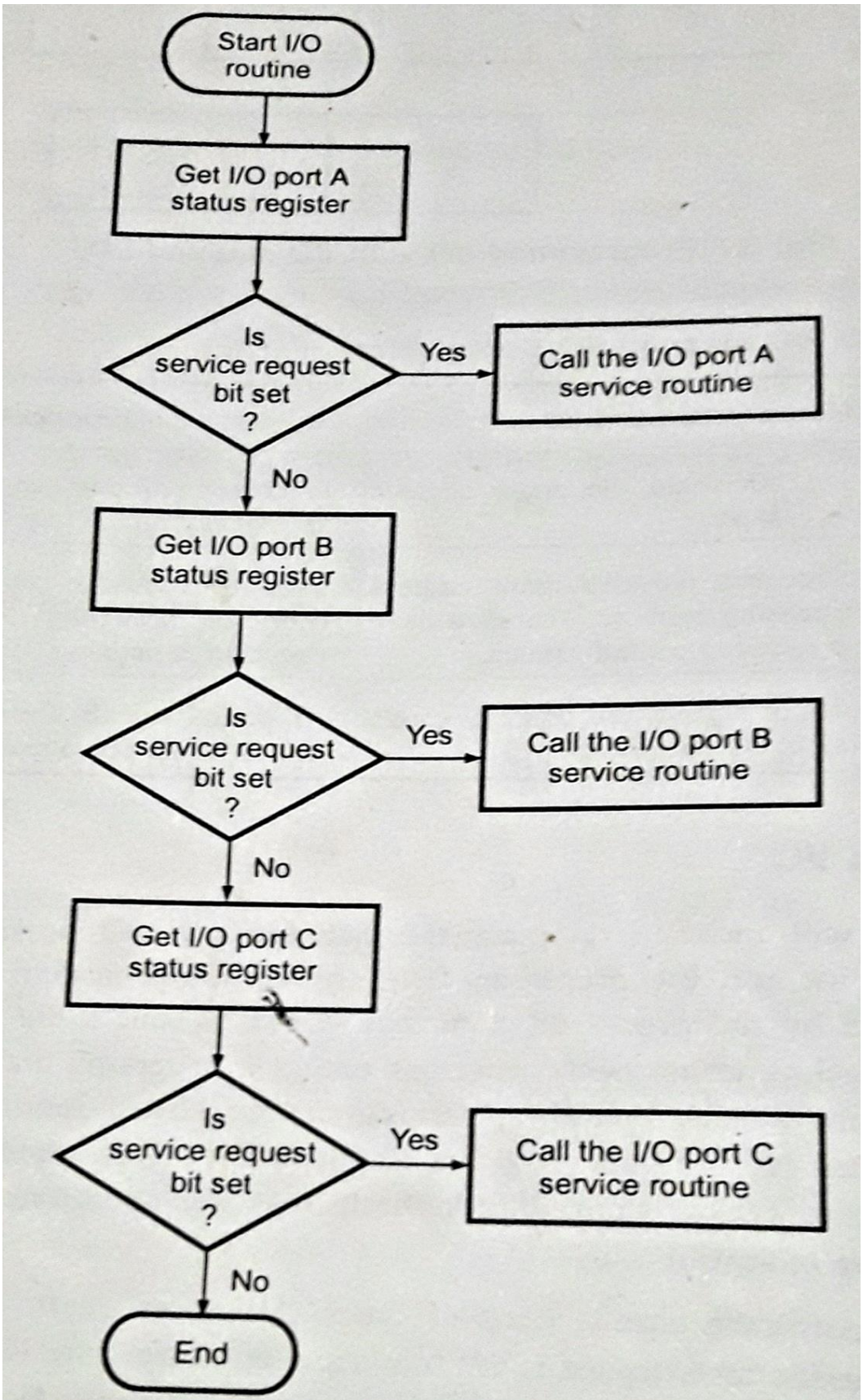


Figure 5.8 Flowchart for I / O service routine

When an I/O instruction is encountered by the processor, the addressed I/O port is expected to be ready to respond the instruction to prevent loss of information. Thus it is desirable for the processor to know the I/O device status (ready or not for I/O data desirable for the processor to know the I/O device status (ready or not for I/O data transfer). In programmed I/O systems, the processor is usually programmed to test the I/O device status before initiating a data transfer.

5.3 Interrupt Driven I/O

Sometimes it is necessary to have the computer automatically execute one of a collection of special routine whenever certain conditions exists within a program or the computer system e.g. It is necessary that computer system should give response to devices such as keyboard, sensor and other components when they request for service.

The most common method of servicing such device is the polled approach. This is where the processor must test each device in sequence and in affect “asks” each one if it needs communication with the processor. It is easy to see that a large portion of the main program is looping through this continuous polling cycle. Such a method would have a serious and decremental effect on system throughput, thus limiting the tasks that could be assumed by the computer and reducing the cost effectiveness of using such devices.

A more desirable method would be the one that allows the processor to execute its main program and only stop to service peripheral devices when it is told to do so by the device itself. In effect, the method would provide an external asynchronous input that would inform the processor that it should complete whatever instruction that is currently being executed, and fetch a new routine that will service the requesting device. Once this of servicing is completed, the processor allows execution of special routines by interrupting normal program execution. When a processor allows executing its current program and calls a special routine which “service” the interrupt. This is illustrated in Figure 5.9. The event that causes the interruption is called interrupt and the special routine executed to service the interrupt is called **interrupt service routine (ISR).** The interrupt service routine is different from subroutine because the address of ISR is predefined or it is available in interrupt vector table (IVT), whereas subroutine address is necessarily to be given in subroutine CALL instruction. IRET instruction is used to return from the ISR whereas RET instruction is used to return from subroutine. IRET instruction restores flag contents along with CS and IP in the IA-32 architecture; however RET instruction only restores CS and IP contents.

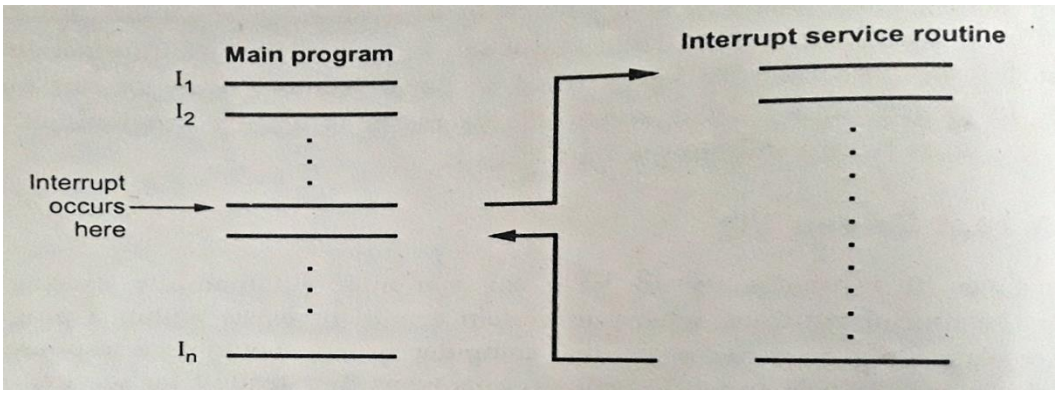


Figure 5.9: Interrupt Operation

Normal program can be interrupted by three ways:

1. By external signal
2. By a special instruction in the program or
3. By the occurrence of some condition.

An interrupt caused by an external signal is referred as a **hardware interrupt**.Conditional interrupts caused by special are called **software interrupts**.

5.3.1 Interrupt Hardware

An I/O device requests an interrupt by activating a bus line called interrupt request or simply interrupt. The interrupts are classified as:

1. Single level interrupts
2. Multi level interrupts

I. Single Level Interrupts

In a single level interrupts there can be many interrupting devices. But all interrupt requests are made via a single input pin of the CPU. When interrupted, CPU has to poll the I/O ports to identify the request device. Polling software routine that checks the logic state of each device. Once the interrupting I/O port is identified, the CPU will service it and then return to task it was performing before the interrupt.

Figure 5.10 shows single level interrupt system, in which the interrupt request from all the devices are logically ORed and connected to the interrupt input of the processor. Thus, the interrupt request from any device is routed to the processor interrupt input. After getting interrupted, processor identifies requesting device by reading interrupt status of each device.

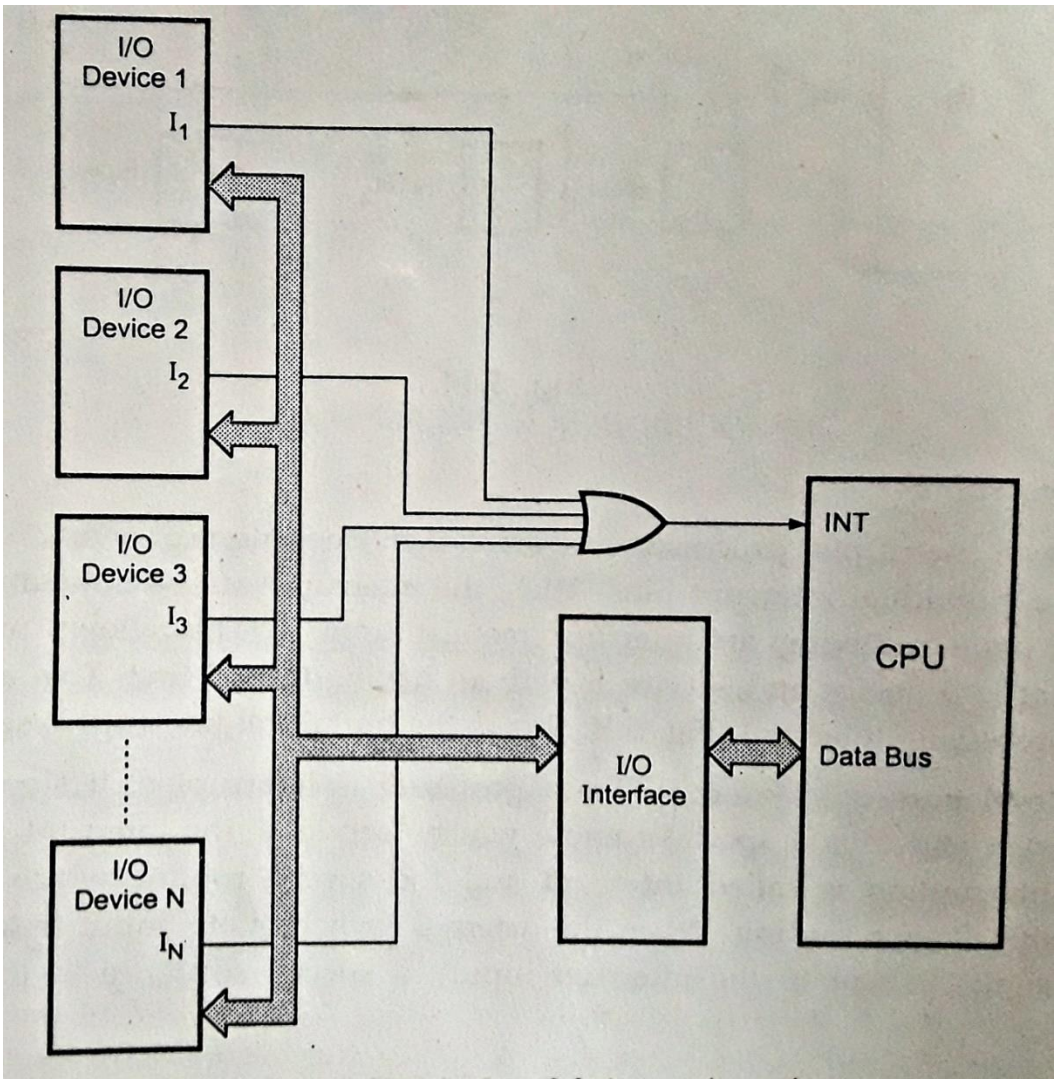


Figure 5.10 Single level interrupt system

The Figure 5.11 shows the equivalent circuit for single interrupt-request line used to serve n-devices. This is another way of representing single level interrupt system. Here, all devices are connected to the INTR line via switches to ground. To request an interrupt, a device closes its associated switch. When all interrupt-request line will be equal to V_{DD}. This is the inactive state of the line. When a device request an interrupt by closing its switch, the voltage on the line drops to 0, causing the interrupt-request signal, INTR, received by the processor to go to 1. Closing of one more switching will cause the line voltage to drop to 0, resulting INTR=1. This is equivalent to logically OR of the requests from individual devices to generate INTR signal.

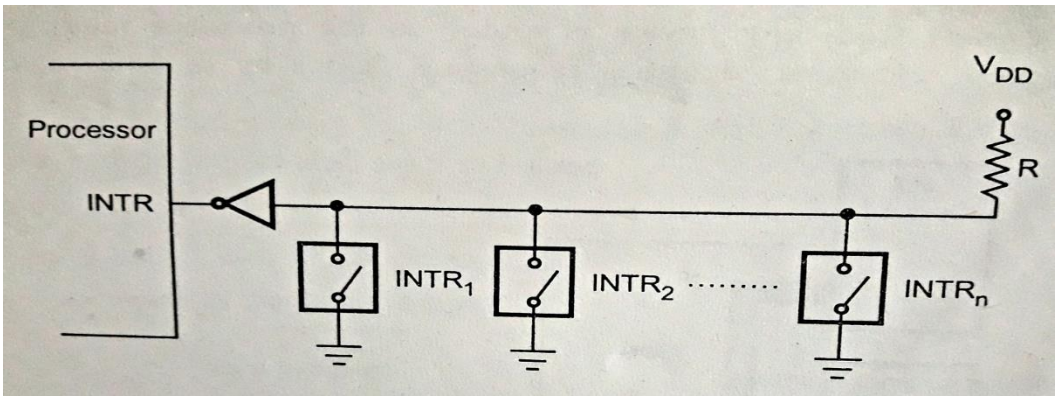


Figure 5.11: Equivalent Circuit

To implement electronic switch shown in the Figure 5.11, we have to use open collector or open drain gates. Because output of an open collector or an open drain gate is equivalent to a switch to ground that is open, and outputs from more than one gate can be connected together and tied to V_{CC} or V_{DD}.

II. Multi-Level Interrupts

In multilevel interrupt system, when a processor is interrupted , it stops executing its current program and calls a special routine which "services" the interrupt. The event that causes the interruption is called interrupt and the special routine which is executed is called interrupt service routine. When the external asynchronous input (interrupt input) is asserted (a signal is sent to the interrupt input), a special sequence in the control logic begins.

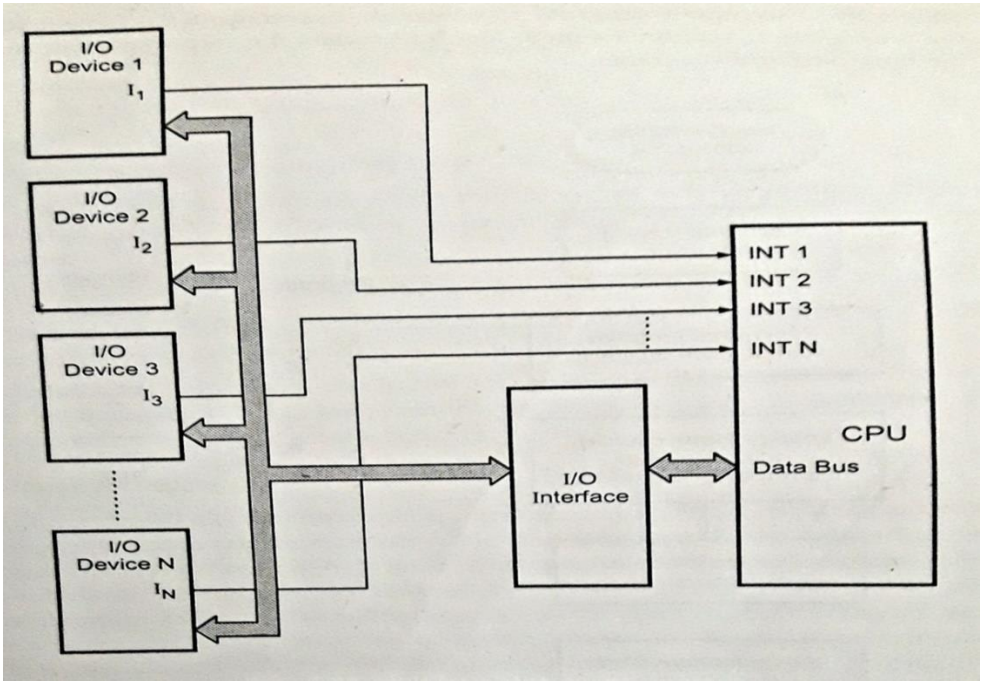


Figure 5.12 Multilevel Interrupt system

1. The processor completes its current its current instruction. No instruction is cut-off in the middle of its execution.
2. The program counter's current contents are stored on the stack. Remember, during the execution of an instruction the program counter is pointing to the memory location for the next instruction.
3. The program counter is loaded with the address of an interrupt service routine.
4. Program execution continues with the instruction taken from the memory location pointed by the new program counter contents.
5. The interrupt program continues to execute until a return instruction is executed.
6. After execution of the RET instruction processor gets the old address (the address of the next instruction from where the interrupt service routine was called.) of the program counter from the stack and puts it back into the program counter. This allows the interrupted program to continue executing at the instruction following the one where it was interrupted. Figure 5.13 shows the response to an interrupt with the flowchart and diagram.

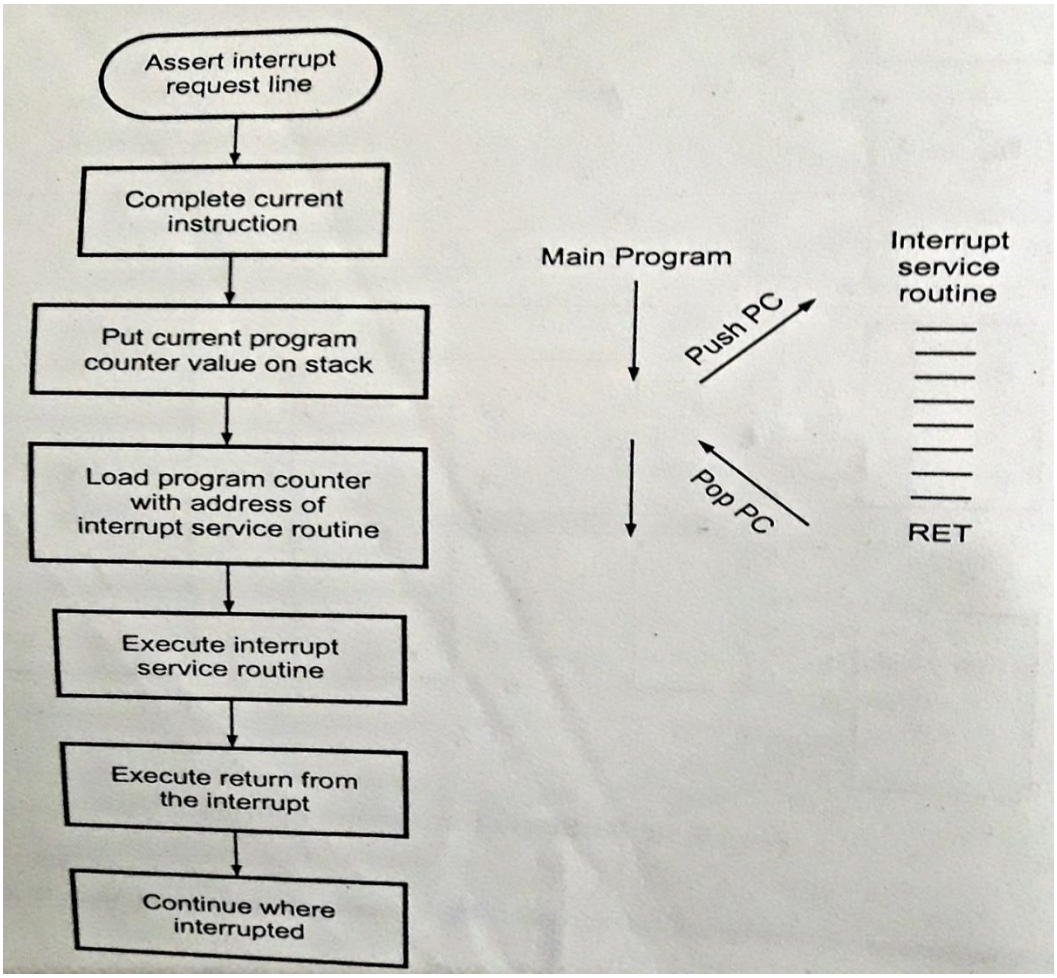


Figure 5.13 Response to an interrupt with the flowchart and diagram

5.3.2 Enabling and Disabling Interrupts

Maskable interrupts are enabled and disable under program control. By setting or resetting particular flip-flops in the processor, interrupts can be masked or unmasked, respectively. When masked, processor does not respond to the interrupt even through the interrupt is activated. Most of the processor provide the masking facility. In the processor those interrupts which can be masked under software control are called maskable interrupts. The interrupts which can not be masked under software control are called non-masked interrupts.

5.3.3 Handling Multiple Devices

In section 5.3.1 we have seen hardware required for handling multiple interrupts. In this section we discuss the issues related to processing of these multiple interrupts from multiple devices. To handle interrupts from multiple devices processor has to perform following tasks.

- It has to recognize the device requesting an interrupt.
- It has to obtain the starting address of interrupt service routine corresponds to interrupt request. Because each interrupt request has its own interrupt service routine.
- It has to allow the device to interrupt while another interrupt is being serviced.
- It has to take decision that which interrupt should be serviced first when there are simultaneous interrupt requests from two different devices.

The modern processors are capable of performing above mentioned tasks. In these processors, such capabilities are implemented by using vectoring of interrupts, by nesting of interrupts and by assigning priorities to the interrupts.

5.3.3.1 Vectored Interrupts

When the external device interrupts the processor (interrupt request), processor has to execute interrupt service routine for servicing that interrupt. If the internal control circuit of the processor produces a CALL to a predetermined memory location which is the starting address of interrupt service routine, then that address is called vector address and such interrupt are called vector interrupts. For vector interrupts fastest and most flexible response is obtained since such an interrupt causes a direct hardware-implemented transition to the correct interrupted, it reads the vector address and loads it in to the PC.

There are two ways to support vector interrupts:

- Fixed vector address
- Programmable vector address

The processors which support fixed vector address approach have default vector address for each interrupt. The programmer cannot change this address. The processor which supports programmable vector address approach maintain the table in memory called the interrupt vector table. The interrupt vector table (IVT) is an array of memory locations which holds the vector addresses of interrupts supported by the processor. When interrupt occurs the processor reads corresponding vector address given in the interrupt vector table and proceed for execution of interrupt service routine. The programmers are allowed to change the vector addresses stored in the interrupt vector table. Thus this approach is known as **programmable vector address**.

5.3.3.2 Interrupt Nesting

For some devices, a long delay in responding to an interrupt request may cause error in the operation of computer. Such interrupts are acknowledged and serviced eventhough processor is executing an interrupt service routine for another device. A system of interrupts that allows an interrupt service routine to be interrupted is known as nested interrupts. Consider, for example, a computer that keeps a track of the time of day using real time clock . This real time clock requests to the processor at regular intervals and processor accordingly updates the counts for seconds, minutes and hours of the day. For the proper operation such an interrupt request from real time clock. This real time clock requests to the processor at regular intervals and processor accordingly updates the counts for seconds, minutes and hours of the day. For the proper operation such an interrupt request from real time clock must be processed eventhough computer is executing an interrupt service for another device.

5.3.3.3 Interrupt Priority

When interrupt requests arrive from two or more devices simultaneously, the processor has to decide which request should be serviced first and which one should be delayed. The processor takes the decision with the help of interrupt priorities. It accepts the request having the highest priority.

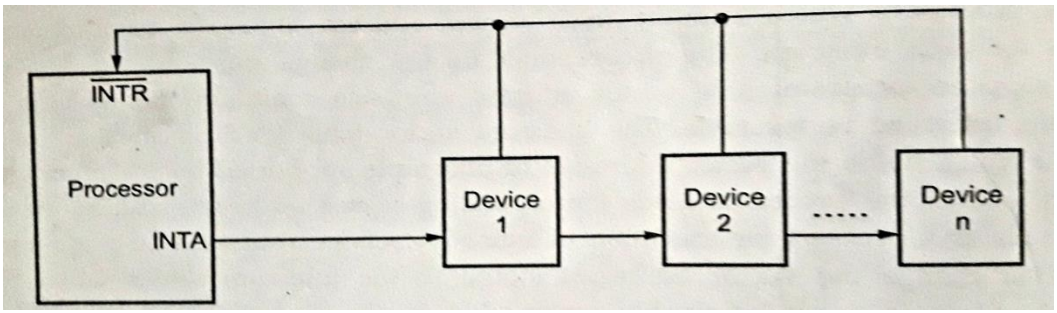


Figure 5.14 Interrupt priority system using daisy chain

In case polling to identify interrupting device, priority is automatically assigned by the order in which devices are polled. Therefore, no further arrangements is required to accommodate simultaneous interrupt requests. However, in case of vectored interrupts, the priority of any device is usually determined by the way the device is connected to the processor. Most common way to connect the devices is to form a daisy chain, as shown in the Fig. As shown in the Fig. the interrupt request line (INTR) is common to all devices and the interrupt acknowledge line (INTA) is connected in a daisy-chain, fashion. In daisy chain fashion the signal is allowed to propagate serially through the devices. When more than one devices issue an interrupt request, the INTR line is activated and processor responds by setting the INTA line. This signal is received by device 1. Device 1 passes the signal to the device 2 only if it does require any service. If device 1 requires service, it blocks the INTA line and puts its identification code on the data lines. Therefore, in daisy-chain arrangement, the device that is electrically closest to the processor has the highest priority.

The Figure 5.15.shows another arrangement for handling priority interrupts. Here, device are organized in groups, and each group is connected at a different priority level. Within a group, devices are connected in a daisy chain.

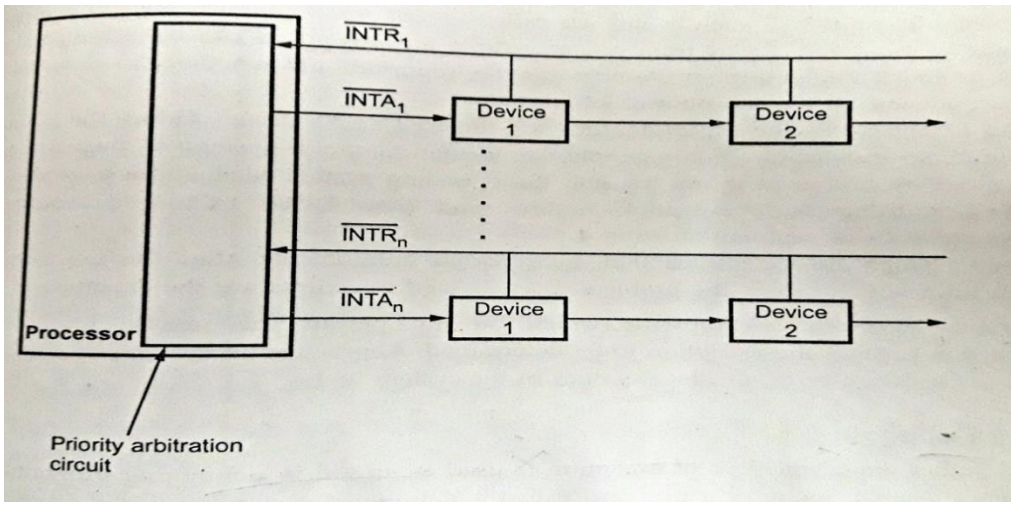


Figure 5.15 Arrangement of priority groups

5.4 Comparison between Programmed I/O and Interrupt Driven I/O

The Table gives the comparison between programmed I/O and interrupt I/O.

Sr. No.	Programmed I/O	Interrupt Driven I/O
1.	In programmed I/O device in sequence and in effect 'ask' each one if it needs communication with the processor. This checking is achieved by continuous polling cycle and hence processor cannot execute other instructions in sequence.	External asynchronous input is used to tell the processor that I/O device needs its service and hence processor does not have to check whether I/O device needs it service or not.
2.	During polling processor is busy and therefore, has serious and decremental effect on system throughput.	In interrupt driven I/O, the processor is allowed to execute its instructions in sequence and only stop to service I/O device when it is told to do so by the device itself. This increase system throughput.
3.	It is implemented without interrupt hardware support.	It is implemented using interrupt hardware support.
4.	It does not depend on interrupt status.	Interrupt must be enabled to process interrupt driven I/O.
5.	It does not need initialization of stack.	It needs initialization of stack.
6.	System throughput decreases as number of I/O devices connected in the system increases.	System throughput does not depend in the system.

5.5 Direct Memory Access (DMA)

In software control data transfer, processor executes a series of instructions to carry out data transfer. For each instruction execution fetch, decode and execute phases are required. Figure 5.16 gives the flow-chart to transfer data from memory to I/O device.

Thus to carry out these tasks processor requires considerable time. So this method of data transfer is not suitable for large data transfers such as data transfer from magnetic disk or optical disk to memory. In such situations hardware controlled data transfer technique is used.

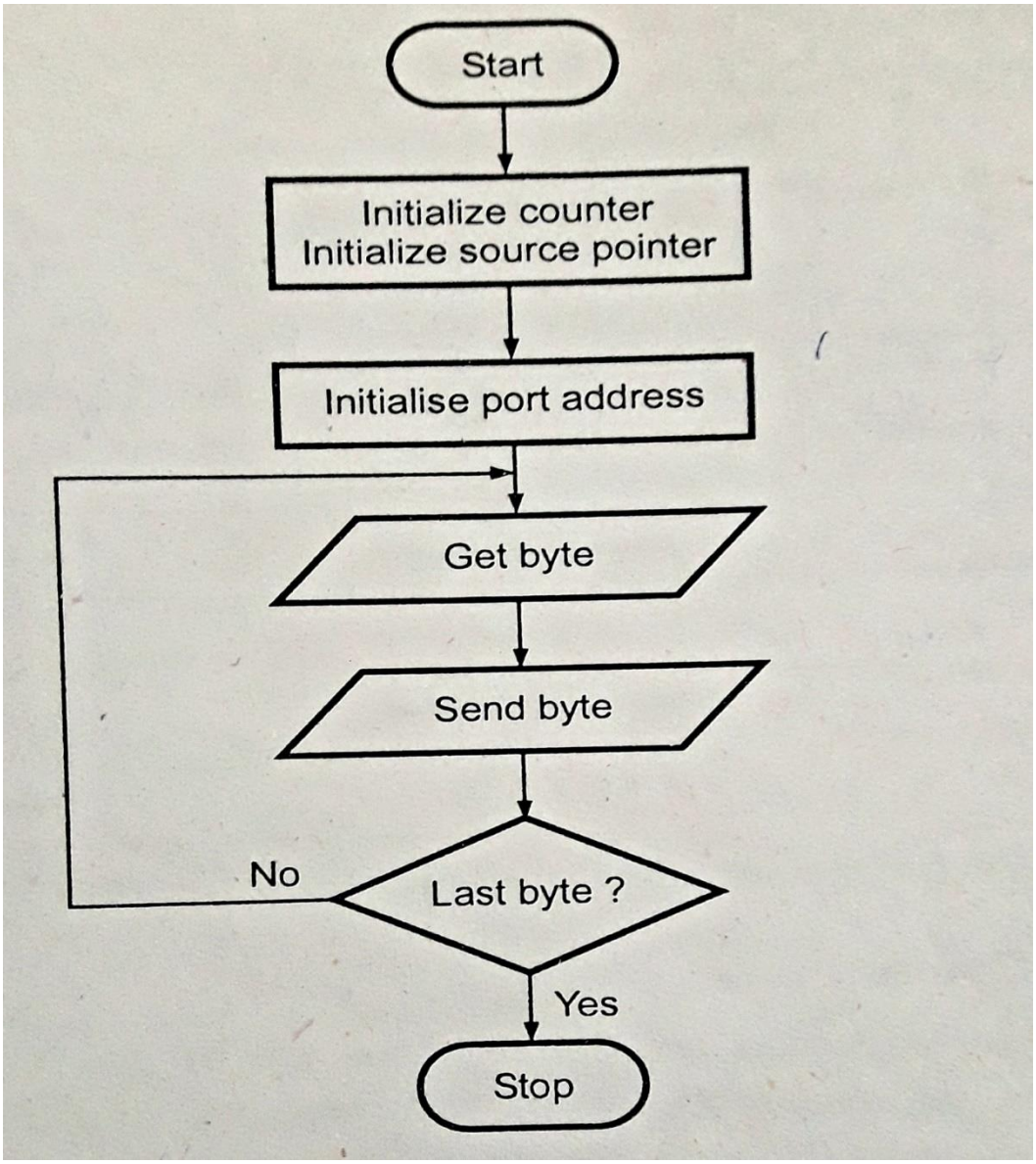


Figure 5.16

There are two main drawbacks in the techniques, programmed I/O and interrupt driven I/O:

- The I/O transfer rate is limited by the speed with which the CPU can test and service a device.
- The time that the CPU spends testing I/O device status and executing a number of instructions for I/O data transfers can often be better spent on other processing tasks.

To overcome above drawbacks an alternative technique, hardware controlled data transfer can be used.

5.5.1 Hardware Controlled Data Transfer

In this technique external device is used to control data transfer. External device generates address and control signals required to control data transfer and allows peripheral device to directly access memory. Hence this technique is referred to as direct memory access (DMA) and external device which controls the data transfer is referred to as DMA controller. Figure 5.17 shows that how DMA controller operates in a computer system.

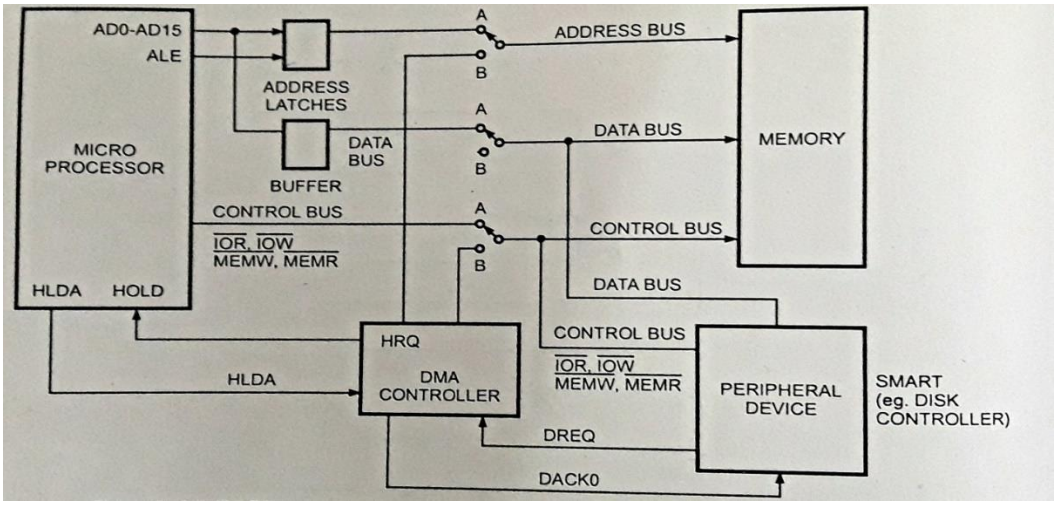


Figure 5.17 DMA controllers operating in a microprocessor system

5.5.2 DMA Idle Cycle

When the system is turned on, the switches are in the A position, so the buses are connected from the processor to the system memory and peripherals. Processor then executes the program until it needs to read a block of data from the disk. To read a block of data from the disk processor sends a series of commands to the disk

controller device telling it to search and read the desired block of data from the disk. When disk controller is ready to transfer first byte of data from disk, it sends DMA request DRQ signal to the DMA controller. Then DMA controller sends a hold request HRQ signal to the processor HOLD input. The processor responds this HOLD signal by floating its buses and sending out a hold acknowledge signal HLDA, to the DMA controller. When the DMA controller receives the HLDA signal, it sends a control signal to change switch position from A to B. This disconnects the processor from the buses and connects DMA controller to the buses.

5.5.3 DMA Active Cycle

When DMA controller gets control of the buses, it sends the memory address where the first byte of data from the disk is to be written. It also sends a DMA acknowledge, DACK signal to the disk controller device telling it to get ready to output the byte. Finally,it asserts both the IOR and MEWM signals on the controls bus. Asserting the IOR signal enables the disk controller to output the byte of data from the disk on the data bus and asserting the MEMW signal enables the addressed memory to accept data from the data bus. In this technique data is transferred directly from the disk controller to the memory location without passing through the processor or the DMA controller.

Thus, the CPU is involved only at the beginning and end of the transfer. The data transfer is monitored by DMA controller, which is also called DMA channel. When the CPU wishes to read or write a block of data, it issues a command to the DMA module or DMA channel by sending the following information to the DMA channel/controller:

1. A read or write operation.
2. The address of I/O device involved.
3. The starting address in memory to read from or write to.
4. The number of words to be read or written.

5.5.4 DMA Channels

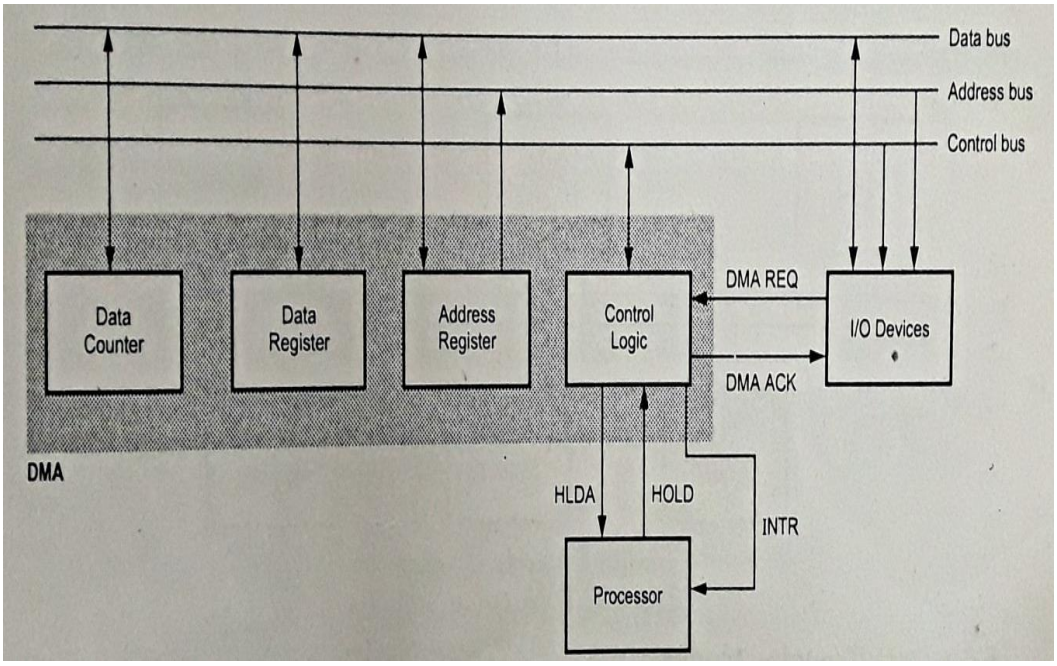


Figure 5.18 Typical DMA block diagram

For performing the above functions, the basic blocks required in a DMA channel/controller are shown in Figure 5.16. It consist of data count, data register, address register and control logic. Data counter register stores the number which gives the number data transfers to be done in one DMA cycle. It is automatically decremented after each word transfer. Data register acts as buffer whereas address register initially holds the starting address of the device. Actually, it stores the address of the next word to be transferred. It is automatically incremented or decremented after each word transfer. After each transfer data counter is tested for zero. When the data count reaches zero, the DMA transfer halts. The DMA controller is normally provided with an interrupts capability, in which case it sends an interrupt to processor to signal the end of the I/O data transfer. Figure 5.19 gives the possible DMA configurations.

As shown in Figure 5.19 (a) all modules, CPU, DMA module, I/O system and memory share the same system bus. In system this programmed I/O technique is used. The data is exchanged between memory and I/O system through DMA module. Each transfer of a word consumes two bus cycles.

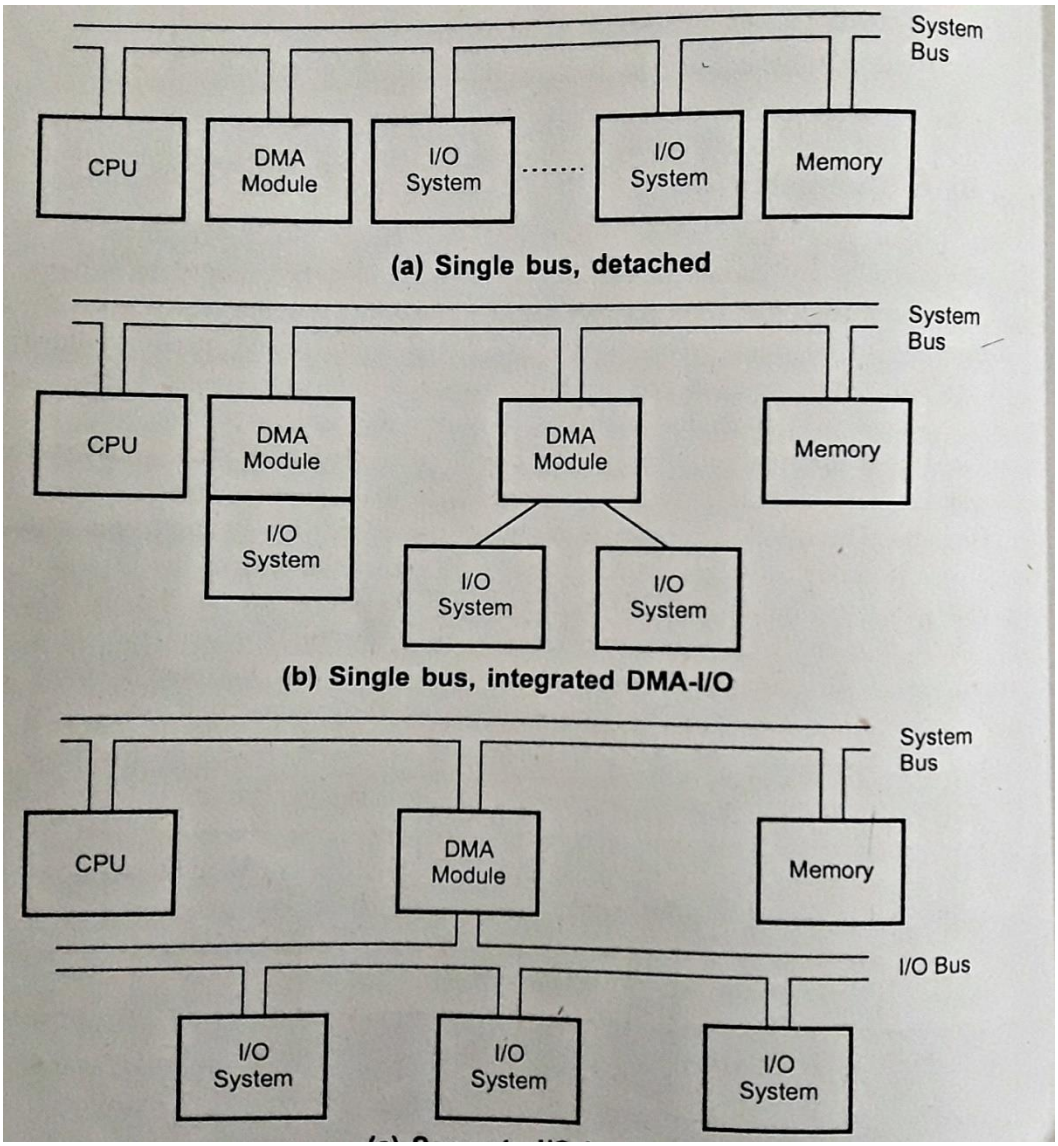


Figure 5.19 DMA configurations

Figure 5.19 (b) shows the another type of DMA configuration in which there is different path between DMA module and I/O system which does not include the system bus. The DMA logic may actually be a part of an I/O system, or it may be separate module that controls one or more I/O systems.

A one step ahead is the third type of configuration as shown in Figure 5.19 (c). In this I/O system are connected to module using an I/O system are connected to module using an I/O bus. This reduces the number of I/O interfaces in the DMA module to one and provides an easy expandable configuration.

5.5.5 Data Transfer Modes

DMA controller transfer data in one of the following three modes:

- Single transfer mode (cycle stealing)
- Block transfer mode
- Demand or burst transfer mode

Single Transfer Mode

In this mode device can make only one transfer (byte or word). After each transfer DMAC gives the control of all buses to the processor. Due to this processor can have access to the buses on a regular basis.

It allows the DMAC in a single transfer mode is as given below:

The operation of the DMA in a single transfer mode is as given below:

1. I/O device asserts DRQ line when it is ready to transfer data.
2. The DMAC asserts HLDA line to request use of the buses from the processor.
3. The processor asserts HLDA, granting the control of buses to the DMAC.
4. The DMAC asserts DACK to the requesting I/O device and executes DMA bus cycle, resulting data transfer.
5. I/O device deasserts its DRQ after transfer of one byte or word.
6. DMA deasserts DACK line.
7. The word/byte transfer count is decremented and the memory address is incremented.
8. The HOLD line is deasserted to give control of all buses back to the processor.
9. HOLD signal is reasserted to request the use of buses when I/O device is ready to transfer another byte or word. The same processor is then repeated until the last transfer.
10. When the transfer count is exhausted, terminal count is generated to indicate the end of the transfer.

Block Transfer Mode

In this mode device can make number of transfers as programmed in the word count register. After each transfer word count is decremented by 1 and the address is decremented or incremented by 1. The DMA transfer is continued until the word count "rollsover" from zero to FFFFH, a Terminal Count (TC) or an external END of Process (EOP) is encountered. Block transfer mode is used when the DMAC needs to transfer a block of data.

The operation of DMA in block transfer mode is as given below:

1. I/O device asserts DRQ line when it is ready to transfer data.
2. The DMAC asserts HLDA line to request use of the buses from the microprocessor.
3. The microprocessor asserts HLDA, granting the control of buses to the DMAC.
4. The DMAC asserts DACK to the requesting I/O device and executes DMA bus cycle, resulting data transfer.
5. I/O device deasserts its DRQ after data transfer of one byte or word.
6. DMA deasserts DACK line.
7. The word/byte transfer count is decremented and the memory address is incremented.
8. When the transfer count is exhausted, the data transfer is not complete and the DMAC waits for another DMA request from the I/O device, indicating that it has another byte or word to transfer. When DMAC receives DMA request steps through are repeated.
9. If the transfer count is not exhausted, the data transfer is complete then DMAC deasserts the HOLD to tell the microprocessor that it no longer needs the buses.
10. Microprocessor then deasserts the HLDA signal to tell the DMAC that it has resumed control of the buses.

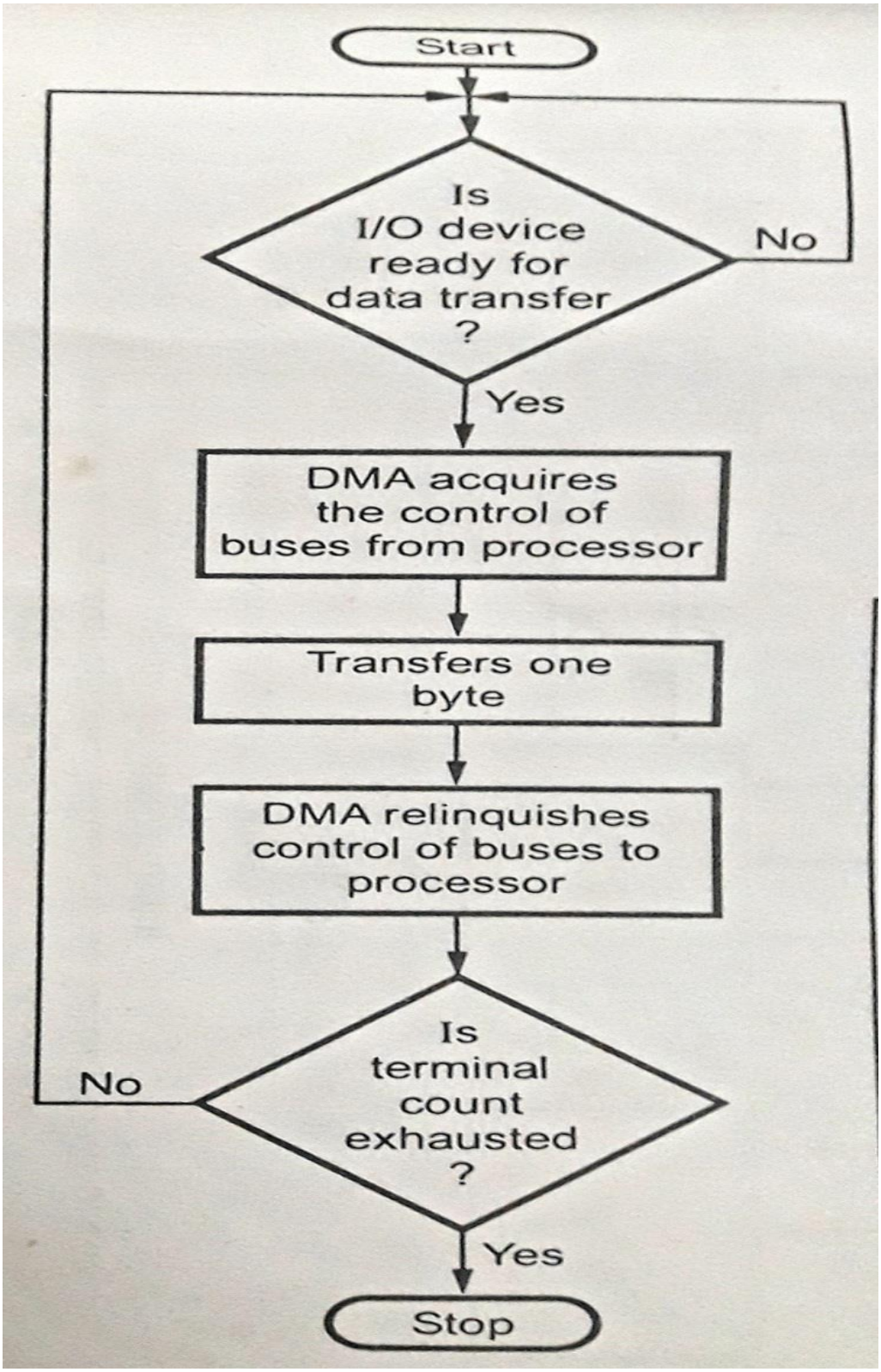
Demand Transfer mode

In this mode the device is programmed to continue making transfers until a TC or external EOP is encountered or until DREQ goes inactive.

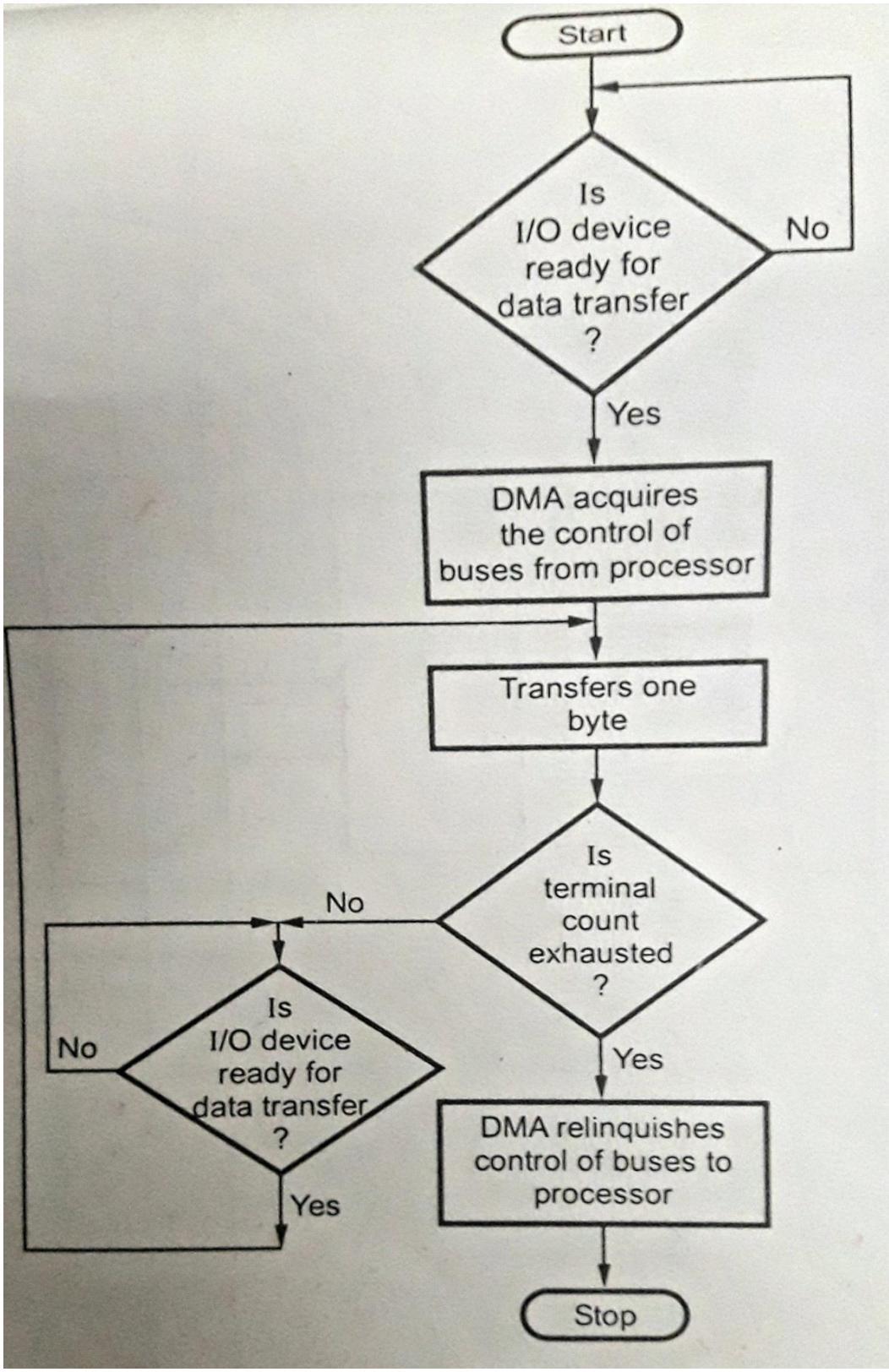
The operation of DMA in demand transfer mode is as given below:

1. I/O device asserts DRQ line when it is ready to transfer data.
2. The DMAC asserts HLDA line to request use of the buses from the microprocessor.
3. The microprocessor asserts HLDA, granting the control of buses to the DMAC.
4. The DMAC asserts DACK to the requesting I/O device and executes DMA bus cycle, resulting data transfer.
5. I/O device deasserts its DRQ after data transfer of one byte or word.
6. DMA deasserts DACK line.
7. The word/byte transfer count is decremented and the memory address is incremented.
8. The DMAC continues to execute transfer cycles until the I/O device deasserts DRQ indicating its inability to continue delivering data. The DMAC deasserts HOLD signal, giving the buses back to microprocessor. It also deasserts DACK.
9. I/O device can re-initiate demand transfer by reasserting DRQ signal.
10. Transfer continues in this way until the transfer count has been exhausted.

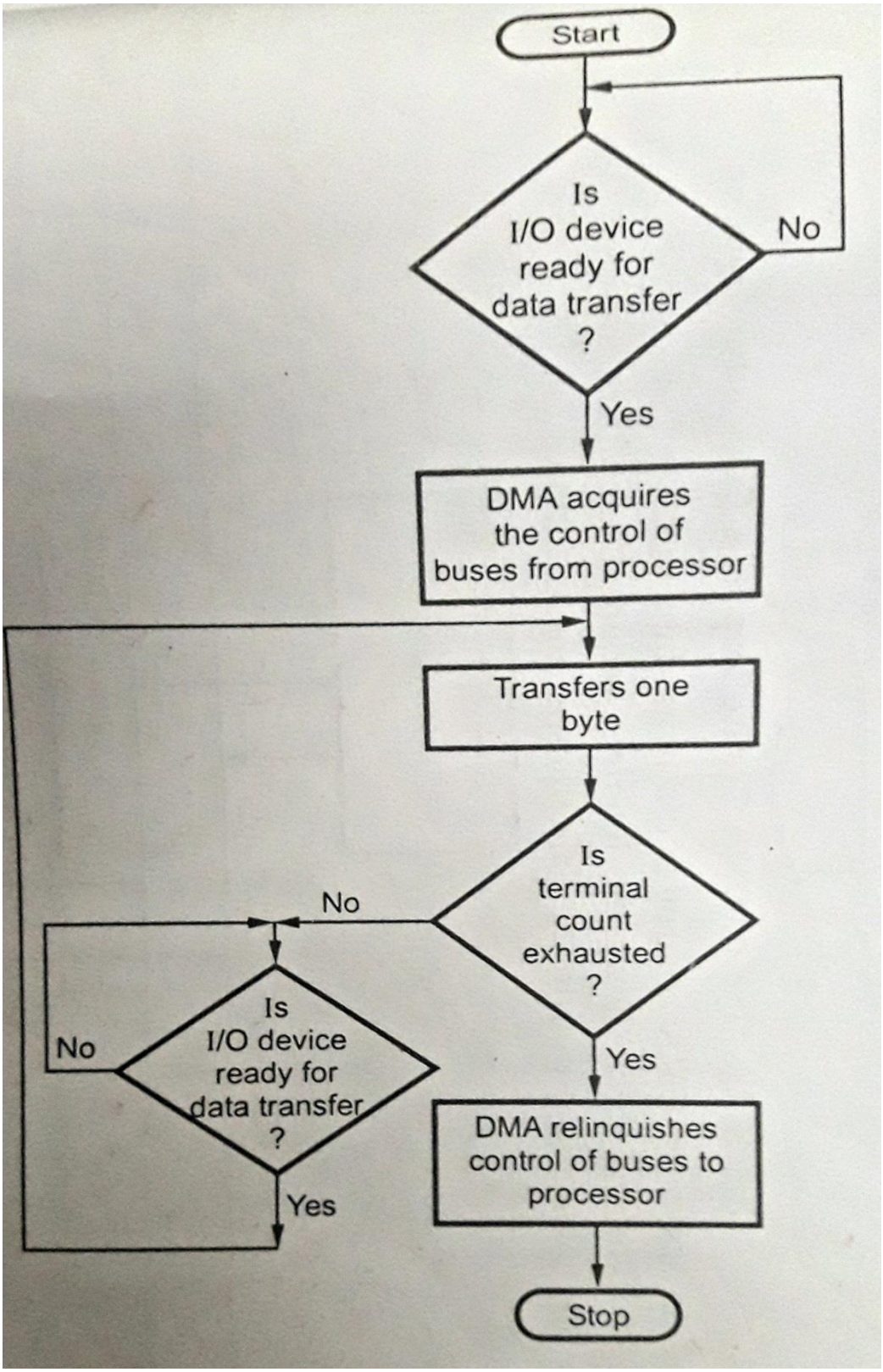
The flowcharts in the Figure 5.20 summarized the three data transfer modes of DMA.



5.20 (a) Single Transfer



5.20 (b) Block transfer



5.20 (c) Demand transfer

5.6 Summary

Programmed I / O is the simplest method of I/O data transfer and it executes a program that contains dedicated I/O instructions. When the processor encounters an I/O instruction, it issues a command for the appropriate I / O module that execute the given instruction.

Channels are used to handle the I / O operation of computer system. The term channel is also referred for I/O processor that is used to manage the processing of I/ O operations. The number of channels connected to the computer system is determined on the basis of application running on the system.

Sample Questions:

1. Why I /O devices be connected directly the systembus?
2. Write short note on Programmed I/O
3. Write short note on Interrupt driven I/O
4. Write short note on DMA controlled I/O
5. Write short note on DMA .
6. Give the comparison between programmed I/O and interrupt driven I/O.
7. Draw and explain generic model of computer showing I/O system.
8. Explain DMA idle and active cycles.
9. Explain various DMA transfer modes.
10. What are the important functions of I/O system?

Reference Books

1. Computer Organization & Architecture, William Stallings, 8e, Pearson Education.
2. Computer Architecture&Organization, John P. Hayes, 3e, Tata McGraw Hill.
3. Computer Organization, 5e, Carl Hamacher, ZconkoVranesic&SafwatZaky, Tata McGraw-Hill.
4. Computer Architecture&Organization, Nicholas Carter, McGraw Hill
5. Computer System Architecture, M.MorrisMano ,Pearson Education.

❖❖❖

6

MEMORY SYSTEM ORGANIZATION

Unit Structure

6.0 Objective

- Understand the characteristics of memory systems
- Explain memory system design and hierarchy
- Understand Associative memory and its organization.
- Discuss Interleaved memory

6.1 Introduction

Programs and the data that processor operate are held in the main memory of the computer during execution, and the data with high storage requirement is stored in the secondary memories such as floppy disk, etc. In this chapter, we discuss how this vital part of the computer operates.

This chapter begins with the characteristics of memory system. It describes organization of multilevel memory system in a computer, main memory organization semiconductor RAM and DRAM organizations and access method.

6.2 Characteristics Of Memory Systems

The key characteristics of memory systems are:

- **Location**: The computer memory is placed in three different locations:

  I. **CPU** It is in the form of CPU registers and its internal cache memory (16 K bytes in case of Pentium)

  II. **Internal**:It is in the main memory of the system which CPU can access directly.

III. **External:** It is in the form of secondary storage devices such as magnetic disk, tapes, etc. The CPU accesses this memory with the help of I/O controllers.

- **Capacity:**It is expressed using two terms: Word size and number of words.

  I. **Word size**: It is expressed in bytes (8-bit). The common word sizes are 8.16 and 32 bits.

  II. **Number of word:**This term specifies the number of words available in the particular memory device. For example, if memory capacity is 4K X 8, then its word size is g and number of words are 4K=4096.

- **Unit of Transfer:**It is the maximum number of bits that can be read or written into the memory at a time. In case of main memory, most of the times it is equal to word size. In case of external memory, unit of transfer is not limited to word size, it is often larger than a word and it is referred to as blocks.

- **Access Method:** There are two different method generally used for memory access.

  I. **Sequential access** Here, memory is organized into units of data, called records. If current record is 1, then in order to read record N, it is necessary to read physical records 1 through N -1. A tape drive is an example of sequential access memory.

  II. **Random access**Here, each addressable location in memory has an unique address. It is possible to access any memory location at random.

- **Performance:** The performance of the memory system is determined using three parameters:

  I. **Access time**: In case of random access memory, it is the time taken by memory to complete read/ write operation from the instant that an address is sent to the memory. On the otherhand, for nonrandom access memory, access time is the time it takes to position the read-write mechanism at the desired location.

  II. **Memory cycle time**: This term is used only in concern with random access memory and it is defined as access time plus additional time required before a second access can commence.

  III. **Transfer rate**: It is defined as the rate at which data can be transferred into or out of a memory unit.

- **Physical Type :** Two most common physical types used today are semiconductor memory and magnetic surface memory.

- **Physical characteristics:**

  I. **Volatile/Nonvolatile**: If memory can hold data even if power is turned off, it is called as nonvolatile memory; otherwise it is called as volatile memory.

  II. **Erasable/Nonerasable**: The memories in which data is once programmed cannot be erased are called as nonerasable memories. On the other hand, if data in the memory is erasable then memory is called as erasable memory.

The Table shows the characteristics of some common memory technologies.

Table 6.1 : Characteristics of some common memory technologies

Technology	**Storage**	**Access Method**	**Alterability**	**performance**	**Typical access time t_A**
Semiconductor RAM	Electronic	Random	Read/write	Volatile	10 ns
Semiconductor ROM	Electronic	Random	Read only	Non Volatile	10 ns
Magnetic (Hard) disk	Magnetic	Semi-random	Read/write	Non Volatile	50 ns
Optical disk CD-ROM	optical	Semi-random	Read only	Non Volatile	100 ms
Erasable optical disk	optical	Semi-random	Read/write	Non Volatile	100 ms
Magnetic tape	Magnetic	Serial	Read/write	Non Volatile	1 s Depend on access location

6.3 Memory Hierarchy / Multilevel Memory

Ideally, computer memory should be fast, large and inexpensive. Unfortunately, it is impossible to meet all the three of these requirements simultaneously. Increased speed and size are achieved at increased cost. Very fast memory system can be achieved if SRAM chips are used. These chips are expensive and for the cost reason it is impracticable to build a large main memory using SRAM chips. The only alternative is to use DRAM chips for large main memories.

Processor fetches the code and data from the main memory to execute the program. The DRAMs which from the main memory are slower devices. So it is necessary to insert wait states wait states in memory read/ write cycles. This reduces the speed of execution. The solution for this problem is come out with the fact that most of the computer programs work with only small sections of code and data at a particular time. In the memory system small section of SRAM is added along with main memory, referred to as cache memory. The program (code) and data that work at a particular time is usually accessed from the cache memory. This is accomplished by loading the active part of code and data from main memory and cache memory. The cache controller looks after this swapping between main memory and cache memory with the help of DMA controlled. The cache memory just discussed is called secondary cache. Recent processors have the built-in cache memory called primary cache.

DRAMs along with allow main memories in the range of tens of megabytes to be implemented at a reasonable cost, the size better speed the performance. But the size of memory is still small compared to the demands of large programs with voluminous data. A solution is provided by using secondary storage, mainly magnetic disk and magnetic tapes to implement large memory spaces. Very large disk are available at a reasonable price, sacrificing the speed.

From the above discussion, we can realize that to make efficient computer system it not possible to rely on a single memory component, but to employ a memory hierarchy. Using memory hierarchy all of different types of memory units are employed to give efficient computer system. A typical memory hierarchy is illustrated in figure 6.1

In summary, we can say that a huge amount of cost-effective storage can be provided by magnetic disks. A large, yet affordable, main memory can be built with DRAM technology along with the cache memory to achieve better speed performance.

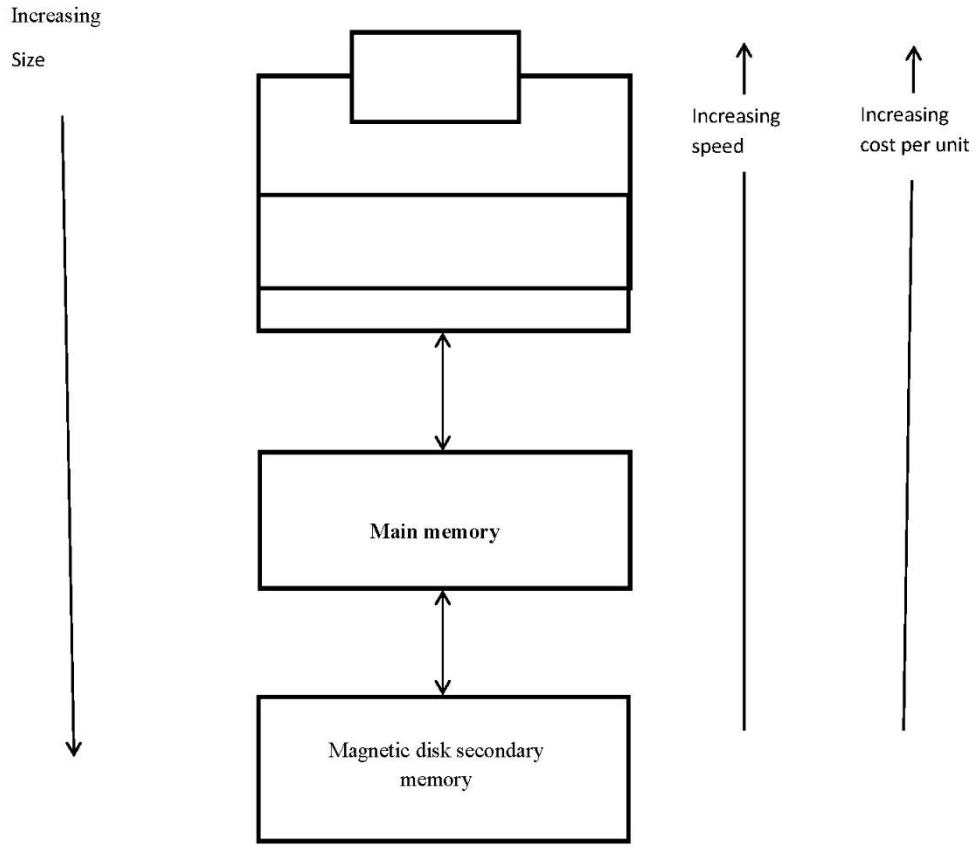


Figure 6.1 Memory Hierarchy

6.4 Semiconductor Main Memory Organization

As mentioned earlier, main memory consists of DRAMs supported with SRAM cache. These are semiconductor members. The semiconductor memories are classified as shown in fig In this section we study various types of semiconductor memories.

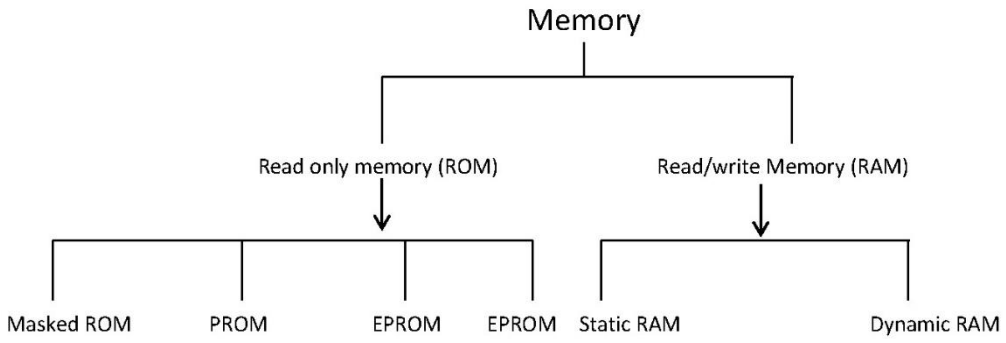


Figure 6.2 Classification of semiconductor memories

6.4.1 ROM (Read Only Memory)

It is a read only memory. We can't write data in this memory. It is non-volatile memory i.e. it can hold data even if power is turned off. Generally, ROM is used to store the binary codes for the sequence of instructions you want the computer to carry out and data such as look up tables. This is because this type of information does not change.

It is important to note that although we give the name RAM to static and dynamic read/write memory devices that does not mean that the ROMs that we are using are also not random access devices. In fact, most ROMs are accessed randomly with unique addresses.

There are four types of ROM : Masked ROM, PROM, EPROM and EEPROM or E^2PROM.

6.4.1.1 ROM and its organization

First, we will see the simple ROM.Figure 6.3 shows four byte diode ROM

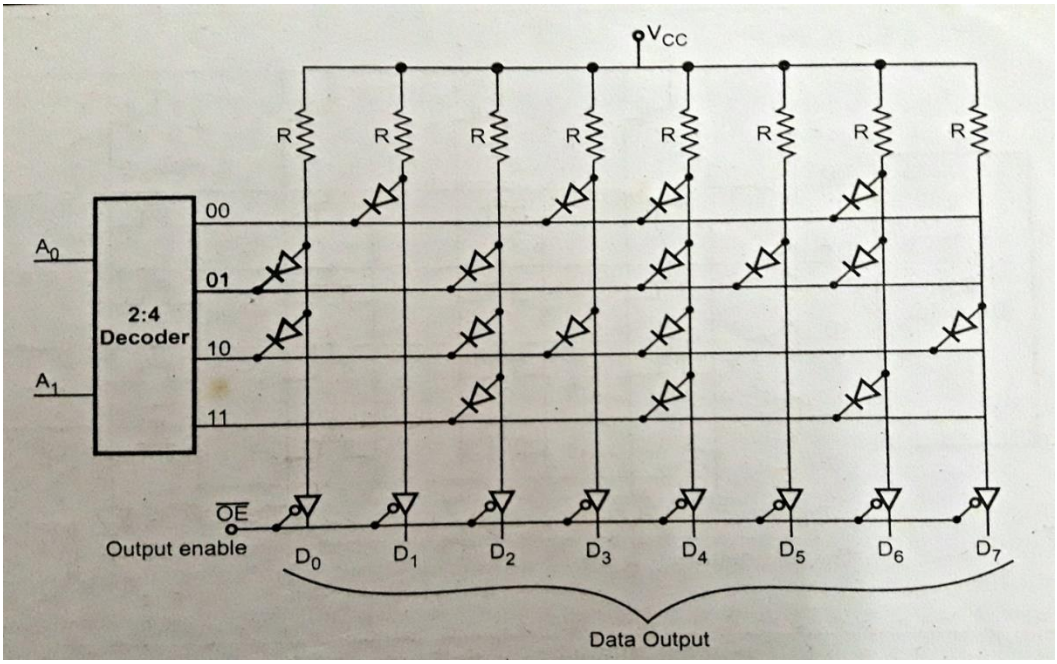


Figure 6.3 : Simple four byte diode ROM

Diode ROM consists of only diodes and decoder. As shown in the Figure 6.3 address lines A_0 and A_1 are decoded by 2:4 decoder and used to select one of the four rows. As decoder output is active low, it places logic 0 on the selected row. Each output data line goes to logic 0 if a diode connects the output data column to

the selected row. Data is available on the output data lines only when output enable (OE) signal is low.

Table 6.2 Contents of ROM

Address in binary	Binary Data								Data in Hex
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	
00	1	0	1	0	0	1	0	1	A5
01	0	1	0	1	0	0	0	1	51
10	0	1	0	0	0	1	1	0	46
11	1	1	0	1	0	1	0	1	D5

Now a days ROMs use MOS technology instead of diode. Figure 6..4Shows four nibble (half-byte) ROM using MOS transistors of diodes and pull up resistors are replaced by MOS transistors.

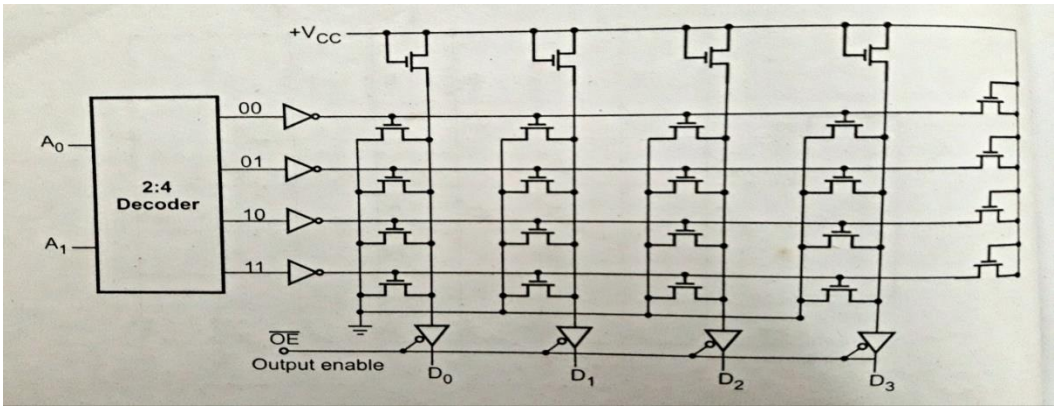


Figure 6.4 Simple four half-byte ROM

The address on the address lines (A_0 and $A_{1)}$ is decoded by 2 : 4 decoder. Decoder selects one of the four rows making it logic 0. The inverter connected at the output of decoder inverts the state of selected row (i.e. logic 1). Therefore, each output data line goes to logic 0 if a gate of MOS transistor is connected to row select lines. When gate of the MOS transistor is connected to the selected row, MOS transistor is turned on. This pulls the corresponding column data line to logic 0.

In integrated circuits, a thin metallized layer connects the gates of some transistors to the row select lines. The gate connections of MOS transistors depend on the data to be stored in the ROM. Therefore, according to the user truth table, manufacturer can deposit thin layer of metal to connect gates of the transistors. Once the pattern/

mask is decided, it is possible to make thousands of such ROMs. Such ROMs are called Mask-programmed ROMs. Masked ROMs are used in microprocessor based toys, TV games, home computers and other such high volume consumer products.

6.4.1.2 PROM (Programmable Read Only memory)

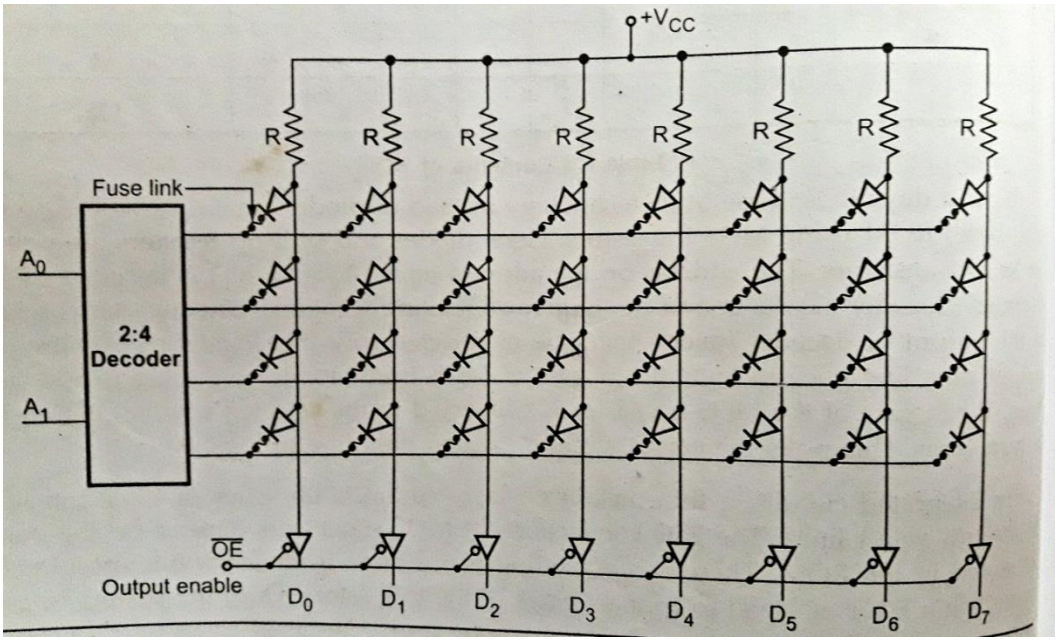


Figure 6.5 : Four byte PROM

Figure 6.5 shows four byte PROM. It has diodes in every bit position; therefore, the output is initially all 0s. Each diode, however has a fusible link in series with it. By addressing bit applying proper current pulse at the corresponding output, we can blow out the fuse, storing logic 1 at the bit the fuse it is necessary to pass around 20 to 50 mA of current for period 5 to 20 us. Theblowing of fuses according to the truth table is called programming of ROM. The user can program PROMs with special PROM programmer. The PROM programmer selectively burns the fuses according to the bit pattern to be stored. This process is also known as burning of PROM. The PROM programmer selectively burns the fuses according to the bit pattern to be stored. This process is also known as burning of PROM. The PROMs are one time programmable. Once programmed, the information stored is permanent.

6.4.1.3 EPROM (Erasable Programmable Read Only Memory)

Erasable Programmable ROMs use MOS circuitry. They store 1s and 0s as a packet of charge in a buried layer of the IC chip. EPROMs can be programmed by the user with a special EPROM programmer. The important point is that we can erase the stored data in the EPROMs by exposing the chip to ultraviolet light through its quartz window for 15 to 20 minutes, as shown in the figure 6.6

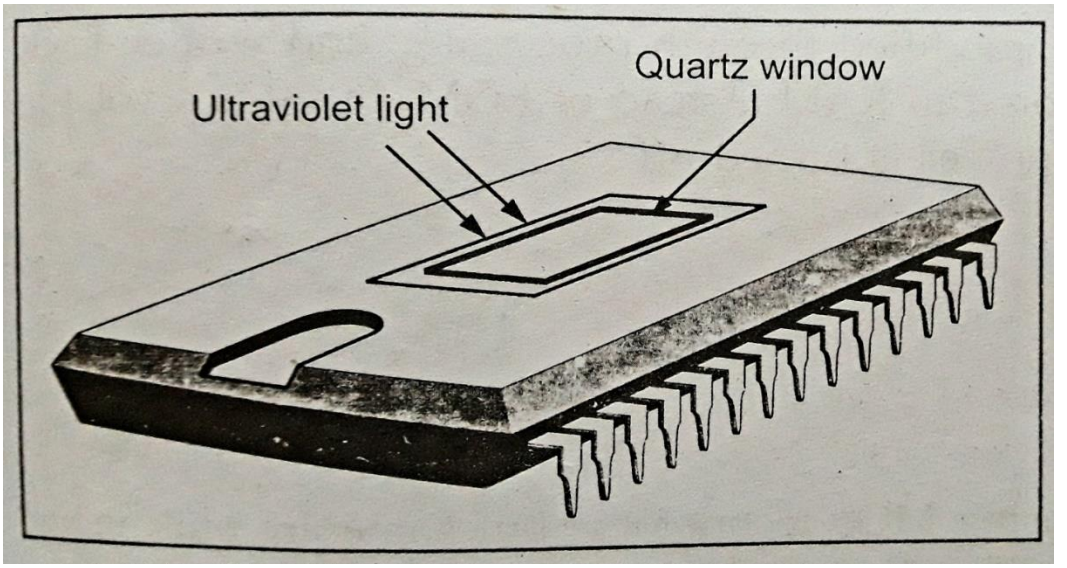


Figure 6.6: EPROM

It is not possible to erase selective information, when erased the entire information is lost. The chip cab be reprogrammed. This memory is ideally suitable for product development, experiment projects, and college laboratories, since this chip can be reused many times, over.

EPROM Programming:

When erased each cell in the EPROM contains 1. Data is introduced by selectively programming 0s into the desired bit locations. Although only 0s will be programmed, both 1s and 0s can be presented in the data.

During programming address and data are stable, program pulse is applied to the program input of the address and data are stable, program pulse is applied to the program input of the EPROM. The program pulse duration is around 50 ms and its amplitude depends on EPROM IC. It is typically 5.5 V to 25 V. In EPROM, it is possible to program any location at any location at any time- either individually, sequentially, or at random.

6.4.1.4 EEPROM (Electrical Erasable Programmable Read Only Memory)

Electrically erasable programmable ROMs also use MOS circuitry very similar to that of EPROM. Data is stored as charge or no charge on an insulated layer or an insulating floating gate in the device. The insulating layer is made very thin (<200 A). Therefore, a voltage as low as 20 to 25 V can be to move charge across the thin barrier in the either direction for programming or erasing. EEPROM allows selective erasing at the register level rather than erasing all the information since the information can be changed by using electrical signals. The EEPROM memory also has a special chip erase mode by which entire chip can be erased in 10 ms.This

time is quite small as compared to time required to erase EPROM, and it can be erased and reprogrammed with device right in the circuit. However, EEPROMs expensive and the least dense ROMs.

6.5 Ram and its Organization

Unlike ROM, we can read from or write into the RAM, so it is often called read/write memory. The numerical and character data that are to be processed by the computer change frequently. These data must be stored in type of memory from which they can be read by the microprocessor, modified through processing and written back for storage. For this reason, they are stored in RAM instead of ROM. But it is a volatile memory, i.e. it cannot hold data when power is turned off.

There are two types of RAMs:

- Static RAM
- Dynamic RAM

6.5.1 Static RAM

Most of the static RAMs are built using MOS technology, but some are built using bipolar technology. In this section we will see both the types.

6.5.2 TTL RAM CELL

Figure 6.7 shows a simplified schematic of a bipolar memory cell. The memory cell is implemented using TLL (Transistor – Transistor-Logic) multiple emitter technology. It stores 1 bit of information. It is nothing but a flip-flop. It can store either 0 or 1 as long as power is applied, and it can set or reset to store either 1 or 0, respectively.

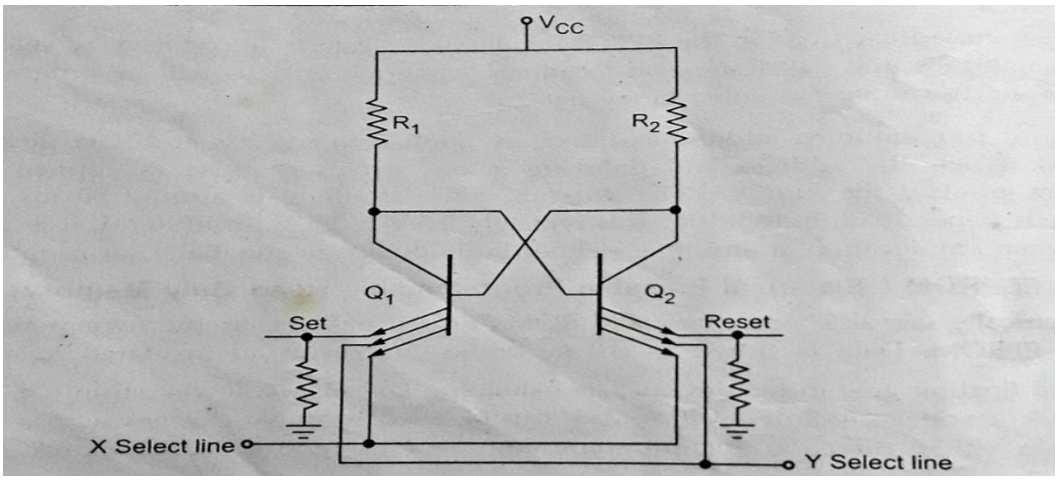


Figure 6.7 TTL RAM CELL

Operation: The X select and Y select input lines select a cell from matrix. The Q_1 and Q_2 are cross coupled inverters, hence one is always OFF while the other is ON. A "1" is stored in the cell if Q_1 is conducting and Q_2 is OFF. A "0" is stored in the cell if Q_2 is conducting and Q_1 is OFF. The state of the cell is changed to a "0" by pulsing a HIGH on the Q_1 (SET) emitter. This turns OFFQ_1. When Q_1 is turned OFF, $Q_2$is turned ON. As long as Q_2 is ON, its collector is LOW and Q_1 is held OFF.

A 1 can be rewritten by pulsing the Q_2 (reset) emitter high. Large number of these memory cells are organized on a row and column basis to form a memory chip. Fig. 6.8 shows the row and column organization of 8192 bit memory chip.

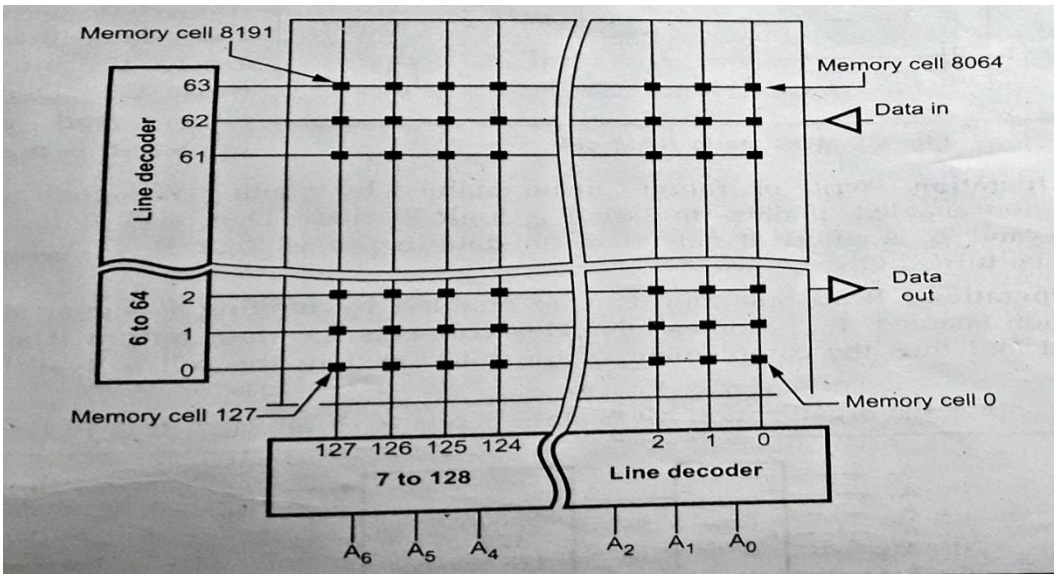


Figure 6.8 Row and column organization for 8192 bit static RAM

The chip has 13 address lines. The first seven address lines are connected to the column decoder to indicate one of the 128 columns. The remaining 6 address lines are connected to the row decoder to indicate one of 64 rows. Where the decoded row and column cross, they select the desired individual memory cell. Simple arithmetic shows that there are 64*128=8192, crossing. Therefore, this memory 8192 memory cells.

6.5.3 MOS Static RAM CELL

Figure 6.9 shows a simplified schematic of MOS static RAM cell. Enhancement mode MOSFET transistors are used to make this RAM cell. It is very similar to TTL cell discussed earlier.

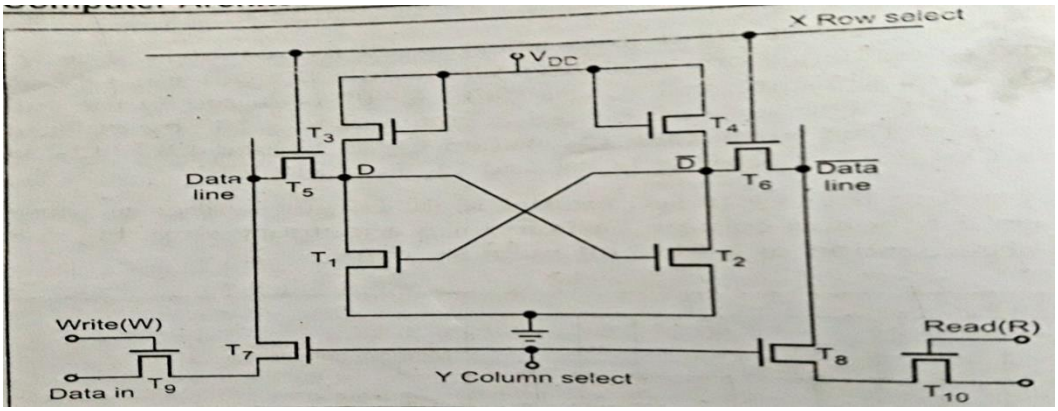


Figure 6.9 MOS static RAM cell

Here, T_1 and T_2 from the basic cross coupled inverters and T_3 and T_4 act as load resistors for T_1 and T_2 . X and Y lines are used for addressing the cell. When X and Y both are high, cell is selected. When X=1, T_5 and T_6 get ON and the cell is connected to the data and data line. When Y=1, T_7 and T_8 are made ON. Due to this, either read or write operation is possible.

Write Operation: Write operation can be enabled by making W signal high. With write operation enable, if data-in signal is logic 1, node D is also at logic 1. This turns ON T_2 and T_1 is cutoff. If new data on data-in pin is logic 0, T_2 will be cutoff and T_1 will be turned ON.

Read Operation: ReadOperation can be enabled by mistake R signal high. With read operation enabled, T_{10} becomes ON. This connects the data output (Data) line to the data out and thus the complement of the bit stored in the cell is available at the output.

Figure 6.10 shows organization MOS static RAM IC Intel 2167. It is 16 K X 1 RAM

To enable Read and write Operation CS must be low. If CS and WE both are low, it is a write operation. In write operation, data on the D_{in} input is written into the addressed cell while the output will remain in the high impedance state.

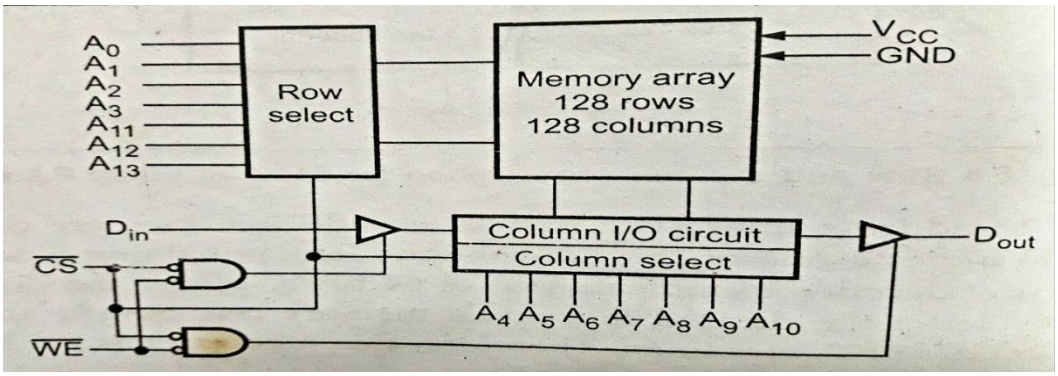


Figure 6.10 Memory organization for IC 2167 (16 K X 1)

For Read operation CS signal must be low and WE signal must be high. In read operation, data from the addressed cell appears at the output while the input buffer is disabled.

When CS is high, both input and output buffers are disable and the chip is effectively electrically disconnected. This makes the IC to operate in power down mode. In power down mode it takes only 40 mA current as compared to 125 mA in the active mode.

6.5.4 Dynamic RAM

Dynamic RAM stores the data as a charge on the capacitor. Figure 6.11 shows the dynamic RAM cell. A dynamic RAM contains thousands of such memory cells. When COLUMN (Sence) and ROW (Control) lines go high, the MOSFET conducts and charges the capacitor. When the COLUMN and ROW lines go low, the MOSFET opens and the capacitor retains its charge. In this way, it stores 1 bit. Since only a single MOSFET and capacitor are needed, the dynamic RAM contains more memory cells as compared to static RAM per unit area.

The disadvantage of dynamic RAM is that it need refreshing of charge on the capacitor after every few milliseconds. This complicates the system design, since it requires the extra hardware to control refreshing of dynamic RAMs.

In this type of cell, the transistor acts as a switch. The basic simplified operation is illustrated in figure 6.12. As shown in the figure 6.12 circuit consists of three tri-state buffers: input buffer, output buffer and refresh buffer. Input and output buffers are enable disable by controlling R/W line. When R/W line is low, input buffer is disabled and output buffer is enabled. With this basic information let us see the read, write and refresh operations.

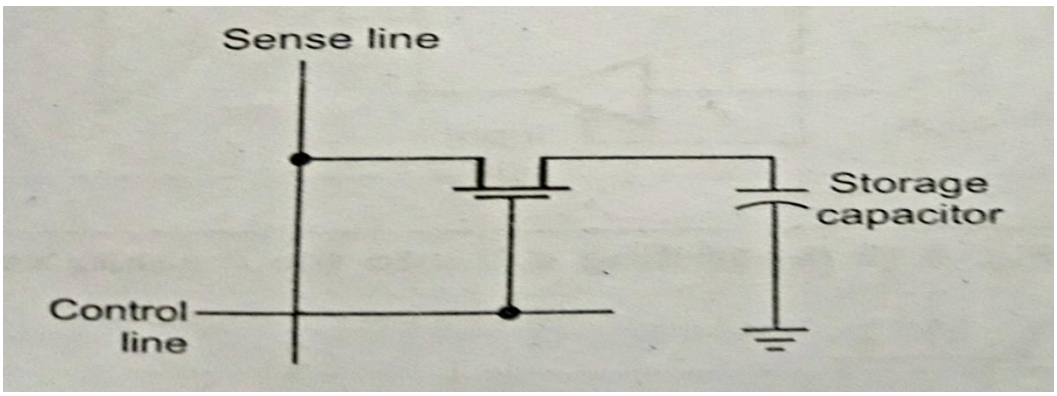


Figure 6.11 Dynamic RAM

6.5.5 Write Operation

To enable write operation R/W line is made LOW which enables input buffer and disables output buffers, as shown in the figure 6.12(a).

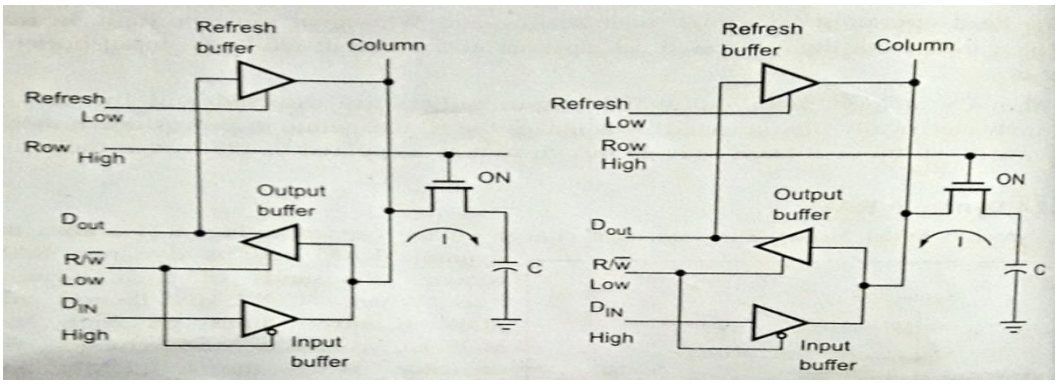


Figure 6.12 (a) Writing a 1 into memory cell (b) Writing a 0 into the memory cell

To write a 1 into the cell, the D_{IN} line is HIGH on the ROW line. This allows the capacitor to charge to a positive voltage. When 0is to be stored, a LOW is applied to the D_{IN} line. The capacitor remains uncharged, or if it is storing a 1, it discharges as indicated in figure 6.12(b). When the ROW line is made LOW, the transistor turns off and disconnects the capacitor from the data line, thus storing the charge (either 1 or 0) on the capacitor.

6.5.6 Read Operation

To read data from the cell the R/W line is made HIGH, which enables output buffer and disable input buffer. Then ROW line is made HIGH. It turns transistors ON and connects the capacitor to the D_{OUT} line through output buffer. This is illustrated in figure 6.12(c) .

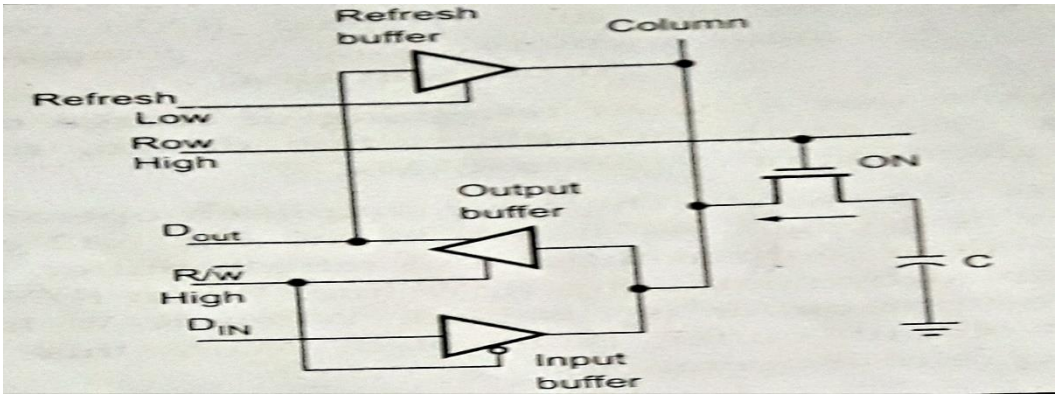


Figure 6.12 (c) Reading a 1 from the memory cell

6.5.7 Refresh Operation

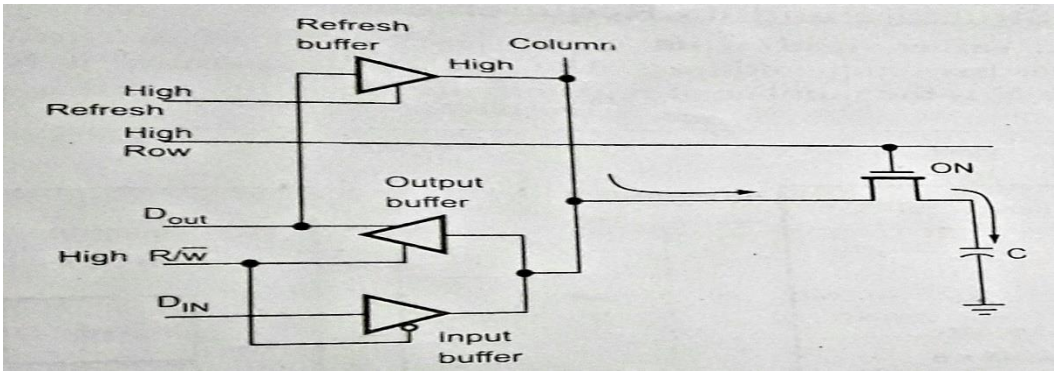


Figure 6.12 (d) Refreshing a stored 1

Basic operations of a dynamic memory cell

To enable refresh operation R/W line, ROW line and REFRESH line are made HIGH. This turns ON transistor and connects capacitor to COLUMN line. As R/W is HIGH, output buffer is enabled and the stored data bit is applied to the input of refresh buffer. The enable refresh buffer then produces a voltage on COLUMN line corresponding to the stored bit and thus replenishing the capacitor as shown in figure 6.12(d) .

6.5.8 Comparison Between SRAM and DRAM

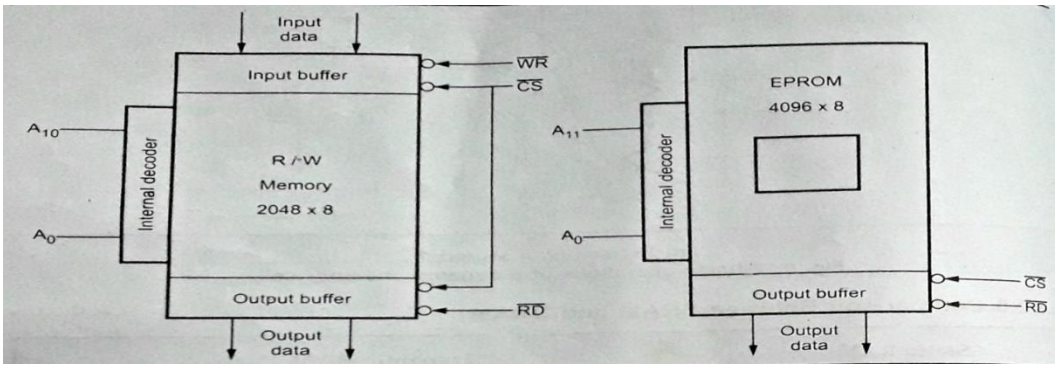
Table 6.3 Comparison Between SRAM and DRAM

Static RAM		**Dynamic RAM**
1	Static RAM contains less memory cells per unit area.	Dynamic RAM contains more memory cells as compared to static RAM per unit area.
2	It less access time hence faster memories	Its access time is greater than static RAMs.
3	Static RAM consists of number of flip-flops. Each flip-flop stores one bit.	Dynamic RAM stores the data as a charge on the capacitor. It consists of MOSFET and the capacitor for each cell.
4	Refreshing circuitry is not required.	Refreshing circuitry is required to maintain the charge on the capacitors after every few milliseconds. Extra hardware is required to control refreshing. This makes system design complicated.
5	Cost is more.	Cost is less.

In above discussion we have seen memory organization for ROM and static RAM. In this organization, data is organized as a single bit data word. However, in most memory ICs the data is organized as eight bit data word. In these ICs eight memory cells are connected in parallel and enabled at a time to read or write eight bit data word. The eight memory cells connected in parallel forms a register and memory is nothing but an array of registers.

6.6 Memory structure and its Requirements

As mentioned earlier, read/write memories consists of an array of registers, in which each register has unique address. The size of the memory is N*M as shown in figure 6..13(a) where N is the number of registers and M is the word length, in number of bits.



6.13 (a) Logic diagram for RAM (b) Logic diagram for EPROM

Example 1: If memory is having 12 address lines and 8 data lines, then

Number of registers/memory locations= $2^N = 2^{12}$

= 4096

Word length = M bit = 8 bit

Example 2 : If memory has 8192 memory locations, then it has 13 address lines.

The table 6.4 summarizes the memory capacity and address lines required for memory interfacing.

Figure 6.13 (b) shows the logic diagram of a typical EPROM (Erasable Programmable Read-Only Memory) with 4096 (4K) registers. It has 12 address lines A_0-A_{11}, one chip select (CS), one Read control signal. Since EPROM is a read only memory, it does not require the (WR) signal.

Table 6.4 : Memory capacity and address lines required

Memory capacity	Address Lines Required
1K = 1024 memory locations	10
2K = 2048 memory locations	11
4K=4096 memory locations	12
8K = 8192 memory locations	13
16K = 16384 memory locations	14
32K = 32768 memory locations	15
64K = 65536 memory locations	16

6.6.1 Memory Cycles

Let us study read and write memory cycle with their timing parameters.

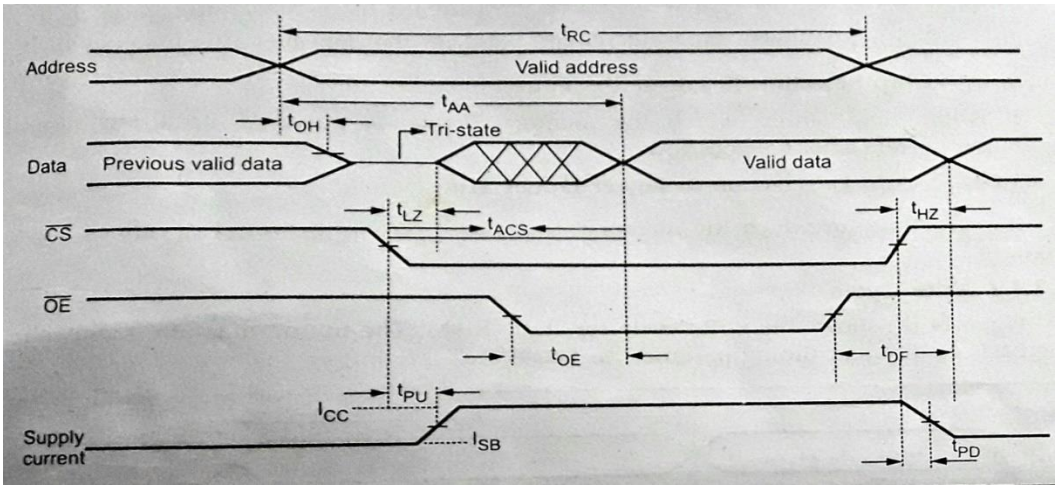


Figure 6.14(a) Read cycle timing waveforms

Figure 6.14(a) shows the read cycle for static RAM. The timing diagram is drawn is drawn on the basis of different timing parameters. These are as follows:

1. t_{RC}: **Read cycle time**

   It is the minimum time for which an address must be held stable on the addressbus, in read cycle.

2. **t_{AA}:Address Access Time**

   It is the maximum specified time within which a valid new data is put on the data bus after an address is applied.

3. **t_{OH}:Output Hold From Address Change**

   It is the time for which a data from previously read address will be present on the output after a new address is applied.

4. **t_{lz} : ChipSelection to Output in Tri-state**

   It is the minimum time for which the data output will remain in tri-state after CS goes low.

5. **t_{ACS}: Chip select access time**

   It is the minimum timerequired for memory device to put valid data on the data bus after CS goes low.

6. **t_{HZ}: Chip Deselection to Output in Tri-State**

   It is the time within which output returns to its high impedance state (tri-state) after cs goes high.

7. **t_{OE}: Output Enable to Output Delay**

   It is the minimum time required for the memory device to put valid data on the data bus after OE goes low.

8. **t_{DF}: Output Enable to Output Delay**

   It is the for which data will remain valid on the data bus after OE goes high

9. **t_{PU}: Chip Selection to Power Up Time**

   It is the time within which the memory device is powered up to its normal operating current after CS goes low.

10. **t_{PD}: Chip Deselection to Power Down Time**

    It is the time for which the memory device will remain powered up after CS goes high.

6.6.2 Write cycle

Figure 6.14(b) shows the write cycle for static RAM. The timing diagram is drawn on the basis of different timing parameters. These are as follow:

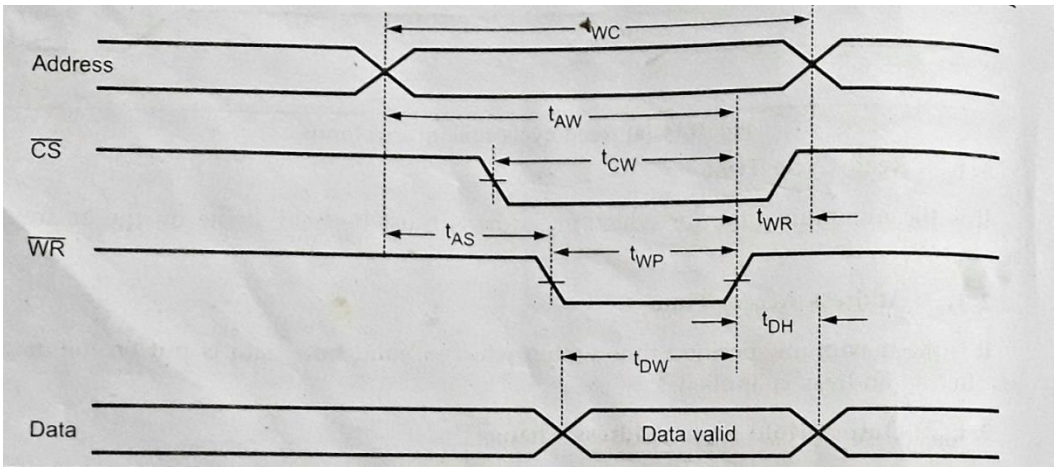


Figure 6.14 (b) Write cycle timing waveforms

1. t_{WC} : **Write cycle Time:**

   It is the minimum time for which an address must be held stable on the address bus, in write cycle.

2. t_{AW} : **Address Valid to End of Write**

   It is the time at which address must be applied on the address bus before WR goes high.

3. t_{WR}: **Write Recovery Time**

   It is the time for which address will remain on address bus after WR goes high.

4. t_{AS}: **Address setup Time**

   When address is applied, it is the time after which WR can be made low.

5. t_{CW}: **Chip Selection to the End of Write**

   It is the time at which the CE must be made low to select the device before WR goes high.

6. t_{WP}: **Write pulse width**

   It is the time for which WR goes low.

7. t_{DW}: **Data valid to the End of write**

   It is the minimum time for which data must be valid on the data bus before WR goes high.

8. t_{DH}: **Data Hold Time**

   It is the time for which data must be held valid after WR goes high.

6.6.3 Memory Chip Organization

In the previous section we have seen various memory cells, such as TTL RAM cell MOS static RAM cell. Large numbers of these cells are organized on a row and column basis chip to form a memory chip.

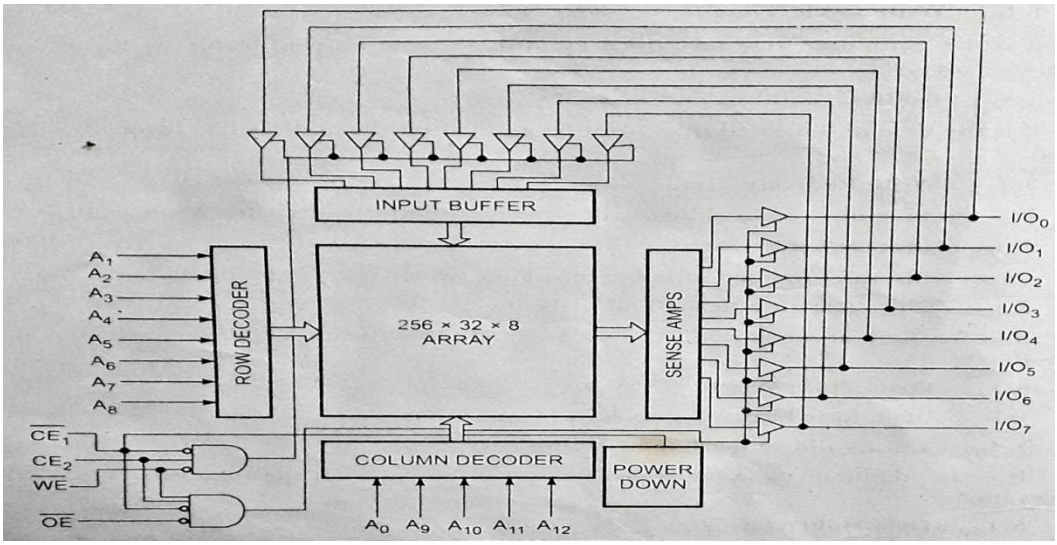


Figure 6.15(a) shows the row and column organization for 6264

Figure 6.15(a) shows the row and column organization for 6264 a high performance CMOS static RAM (SRAM). The chip has 13 address lines. The eight address lines are connected to the row decoder to indicate one of the 256 rows, and remaining five address lines are connected to the column decoder to indicate one of 32 columns. Where the decoded row and column cross, they select the desired individual memory cell. Simple, arithmetic shows that there are 256* 32,8192 crossings. The entire memory has eight such arrays. Therefore this memory has 8192* 8 memory cells.

The integrated circuits for this SRAM is mounted in package have 28 pins. Figure 6.15 (b) shows the pin configuration for IC 6264 (8 K* 8 SRAM).

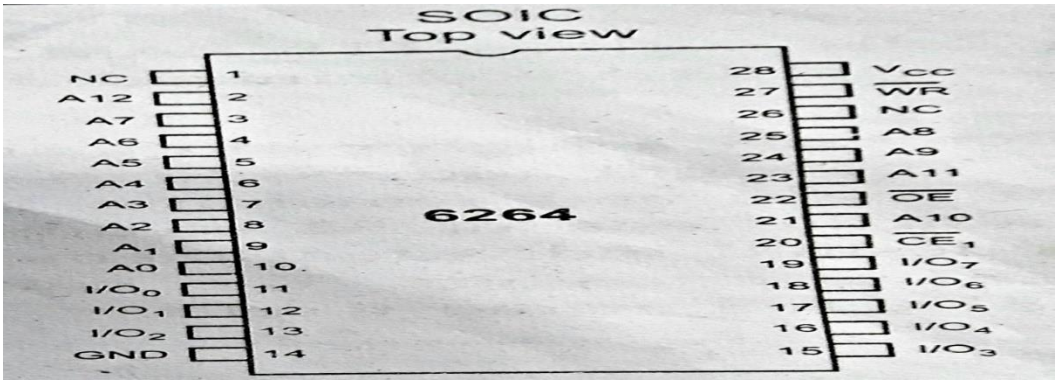


Figure 6.15 (b) Pin diagram of 6264

Figure 6.16 shows typical DRAM memory organization. In DRAMs row address and column address lines are multiplexed to reduce number of pins. Thus for a given memory a DRAM chip has less address pins than SRAM chip. The elements of memory array are connected by both horizontal (row) and vertical (column) lines. Each horizontal line connects to the select terminal of each cell in its row; each vertical line connects to the Data-In/ sence terminal of each cell in its column.

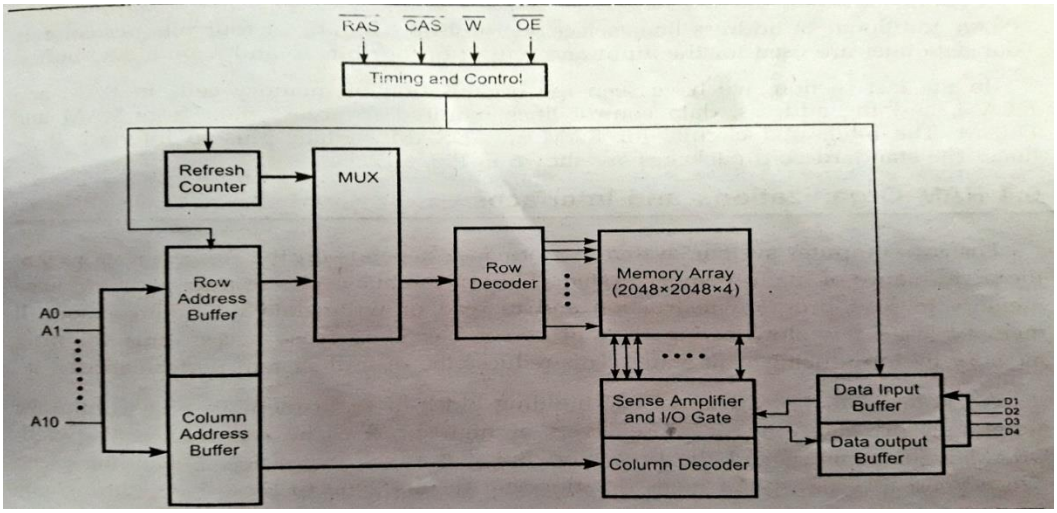


Figure 6.16 DRAM Organization

The DRAM organization shown in figure 6.16 has four arrays of 2048 by 2048 elements. The 11 address lines are required to select one 2048 rows. These 11 lines are fed into a row decoder, which has 11 lines of input and 2048 lines for output. The logic of the decoder activates a single one of the 2048 outputs depending on the bit pattern on the 11 input lines ($2^{11} = 2048$).

An additional 11 address lines select one of 2048 columns of four bits per column. Four data lines are used for the input and output of four bits to and from a data buffer.

In the last section, we have seen the organization of memory cells in RAM and DRAM, and the address, data control lines required to access data from RAM and DRAM. The integrated circuits for RAM and DRAM contain pins to interface these lines. The standard chip packages are shown in figure 6.17

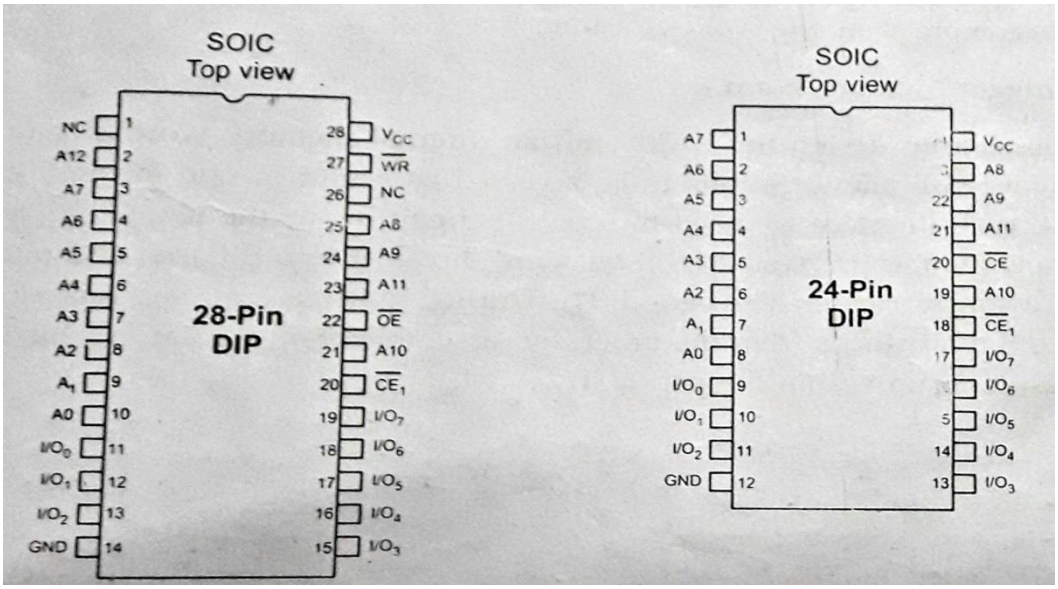


Figure 6.17

6.7 RAM Organization and Interfaces

We know that, DRAM is a basic building block of main memory and to improve system performance one or more levels of high-speed cache are inserted between DRAM main memory and the processor. But it is much expensive. Expanding cache size beyond a certain point gives deteriorating results. Due to these facts, throughout the past two decades, various trends in memory designs have become apparent. In this section we will see a number of enhancements to the basic RAM interface and architecture.

There are two basic ways we can increase the data-transfer rate across its external interface.

There are two basic ways we can increase the data-transfer rate across its external interface.

- Use a bigger memory word and
- Access more than one word at a time.
- **Use a bigger Memory Word:** It is possible to design the RAM with an internal memory words size of W=Sn bits. This word size allows sn bits to be accessed as a unit in one memory cycle time T_M. To have such interface we need the control circuit inside the RAM that, in the case of a read operation, can access an S n-bit word, break it into S parts, and output them to the processor, all within the period T_M. During write operations, this circuit must accept up to S n-bit words from the processor, assemble them into an n s-bits word, and store the result, again within the period T_M.

- **Access More than one Word at a time:** It is possible to partition the RAM into S separate banks B_0, B_1 …., B_{S-1}, each covering part and the memory address space and each provide with its own addressing circuitry. Such architecture allows to carry out S independent access simultaneously in one memory clock period T_M. Here also we need the control circuit inside the RAM to assemble and disassemble the words being accessed.

6.8 Associative Memory

Many data-processing applications require the search of items in a table stored in memory. They use object names or number to identify the location of the named or numbered object within a memory space. For example, an account number may be searched in a file to determine the holder's name and account status. To search an object, the number of accesses to memory depends on the location of the object and the efficiency of the search algorithm.

The time required to find an object stored in memory can be reduced considerably if objects are selected based on their contents, not on their locations. A memory unit accessed by the content is called an **associative memory** or **content addressable memory (CAM).** This type of memory accessed simultaneously and in parallel on the basis of data content rather than by specific address or location.

6.8.1 Hardware organization

The figure 6.18 shows the block diagram of an associative memory. It consists of memory array with match logic for m n-bit words and associated registers. The argument register (A) and key register (K) each have n-bits per word. Each word in memory is compared in parallel with the contents of the argument register. The words that match with the word stored in the argument register set a corresponding bits in the match register. Therefore, reading can be accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.

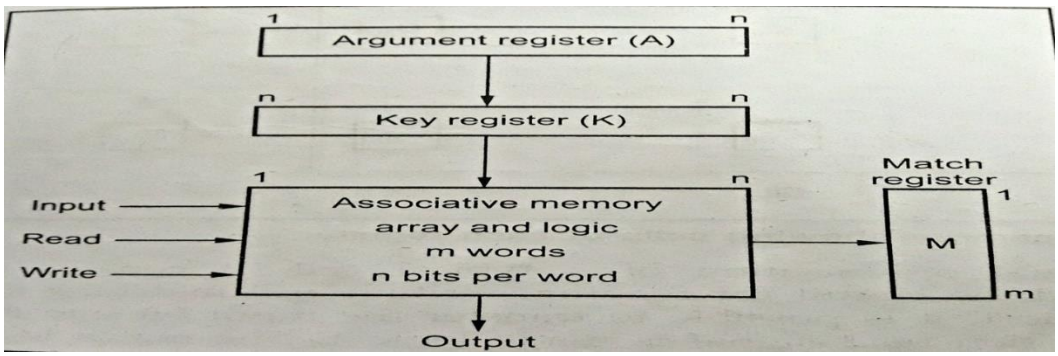


Figure 6.18 Block diagram of associative memory

The key register provides a mask for choosing a particular field or bits in the argument word. Only those bits in the argument register having 1's in their corresponding position of the key register are compared. For example, if argument register. A and the key register K have the bit configuration shown below. Only the three rightmost bits of A are compared with memory words because K has 1's in these positions.

A 11011 O1O

K 00000 111

Word 1 01010 010 match

Word 2 11011 100 no match

The Figure 6.19 shows the associated memory with cells of each register. The cells in the memory array are marked by the letter C with two subscripts. The cells in the memory array are marked by the letter C with two subscripts. The first subscript gives the bit position in the word.

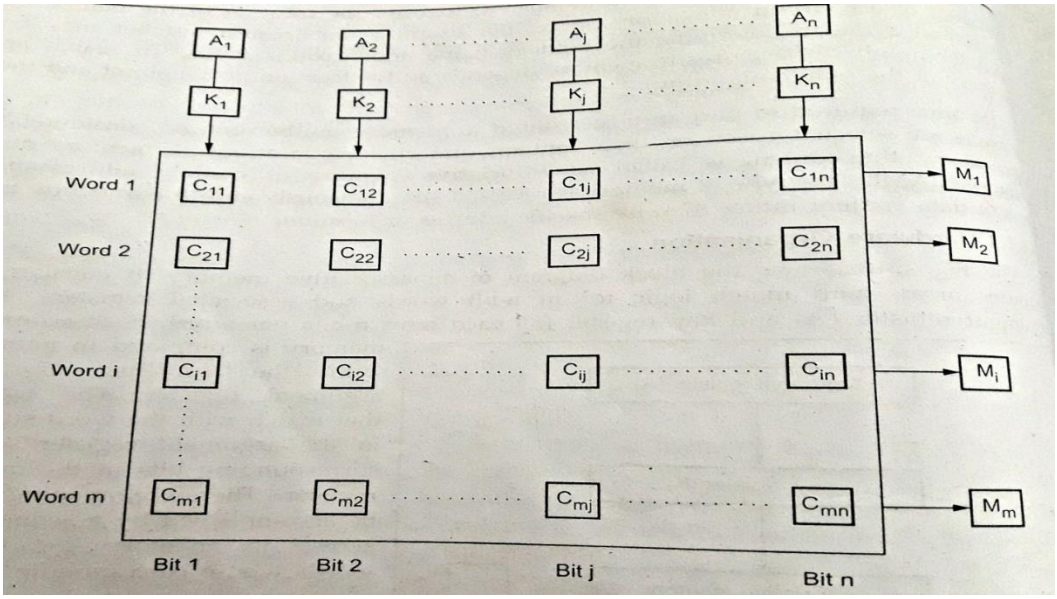


Figure 6.19 shows the associated memory with cells of each register

The Figure 6.20 shows the internal organization of a typical cell. It consists of RS flip-flop as a storage element and the circuit for reading, writing and matching the cell. By making the write signal logic 1, it is possible to transfer the input bit into the storage cell.

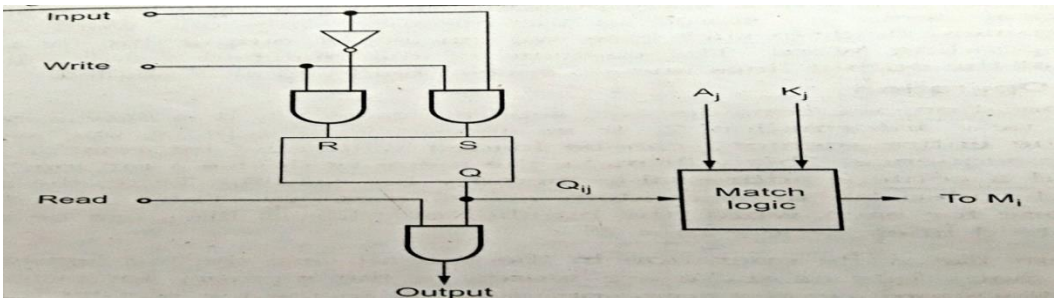


Figure 6.20 Internal organization of typical cell of associative memory

To carry-out read operation read signal is made logic 1. The match logic compares the bit in the storage cell with the corresponding unmasked bit of the argument and provides an output for the decision logic that sets the corresponding bit in the match register.

The Figure 6.21 shows the match logic for each word. The cell bit (Q_{ij}) is compared with the corresponding argument bit with Ex-NOR gate. Ex-NOR gate gives output logic 1 only when its inputs are same. The output Ex-NOR gate is valid only when the corresponding bit in key register is logic 1. This condition is implemented using 2-input OR gate. When corresponding bit in key register is logic 1, the inverter gives output 0 which forces the output of OR gate to follow the second input, i.e. the comparison output; otherwise output of OR-gate is logic 1. The outputs of all OR-gates are then ANDed with n-input AND gate to check whether all bits in the word are matched with the bits in the argument register.

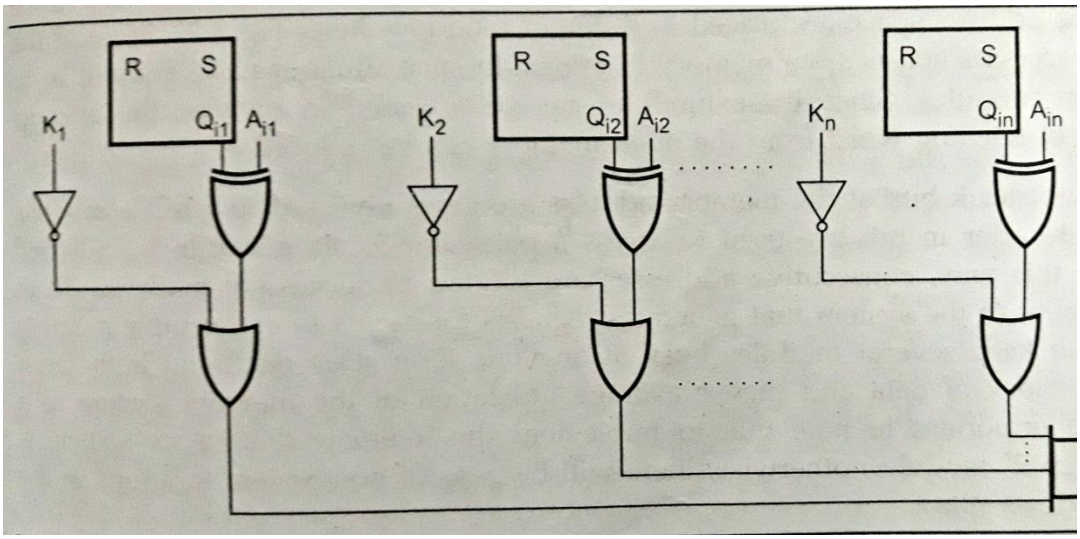


Figure 6.21 Match logic for one word of associative memory

6.8.2 Read Operation

In read operation ,all matched words are read in sequence by applying a read signal to each word line whose corresponding $M_i$bit is a logic 1. In applications where no two identical items are stored in the memory, only one word may match the unmasked argument field. In such case, we use $M_i$output directly as a read signal for the corresponding word. The contents of the matched word will be presented automatically at the output lines and no special read signal is needed.

6.8.3 Write Operation

In write operation , we consider two separate cases : 1. It is necessary to load entire memory with new information . 2. It is necessary to replace one word with new information. The entire memory can be loaded with new information by addressing each location in sequence. This will make the memory device a random access memory for writing and a content addressable memory for reading. Here, the address for the input can be decoded as in a random access memory. Thus instead of having m address lines, one for each word, the number of address lines can be reduced by the use of decoder to d lines, where m 2^d .

To implement the write operation in the second case the tag register is used. This register has as many bits as there are words in the memory for every active (valid) word stored in the memory, the corresponding bit in the tag register is set to 1.When word is deleted from the memory the corresponding tag bit is cleared , i.e. it is set to logic 0. The word is then stored in the memory by scanning the tag register until the first 0 bit is encountered. This gives the first available inactive word and a position for writing a new word. After the new word is stored in the memory it is made active by setting the corresponding tag bit in the register.

6.9 Interleaved Memory

We know that, the memory access time is the bottleneck in the system and one way to reduce it is to use cache memory. Alternative technique to reduce memory access time is **memory interleaving**. In this technique, to reduce memory is divided into a number of memory modules and the address are arranged such that the successive words in the address space are placed in different modules. Refer Figure Most of the times CPU accesses consecutive memory locations. In such situations address will be to the different modules. Since these modules can be accessed in parallel, the average access time of fetching word from the main memory can be reduced.

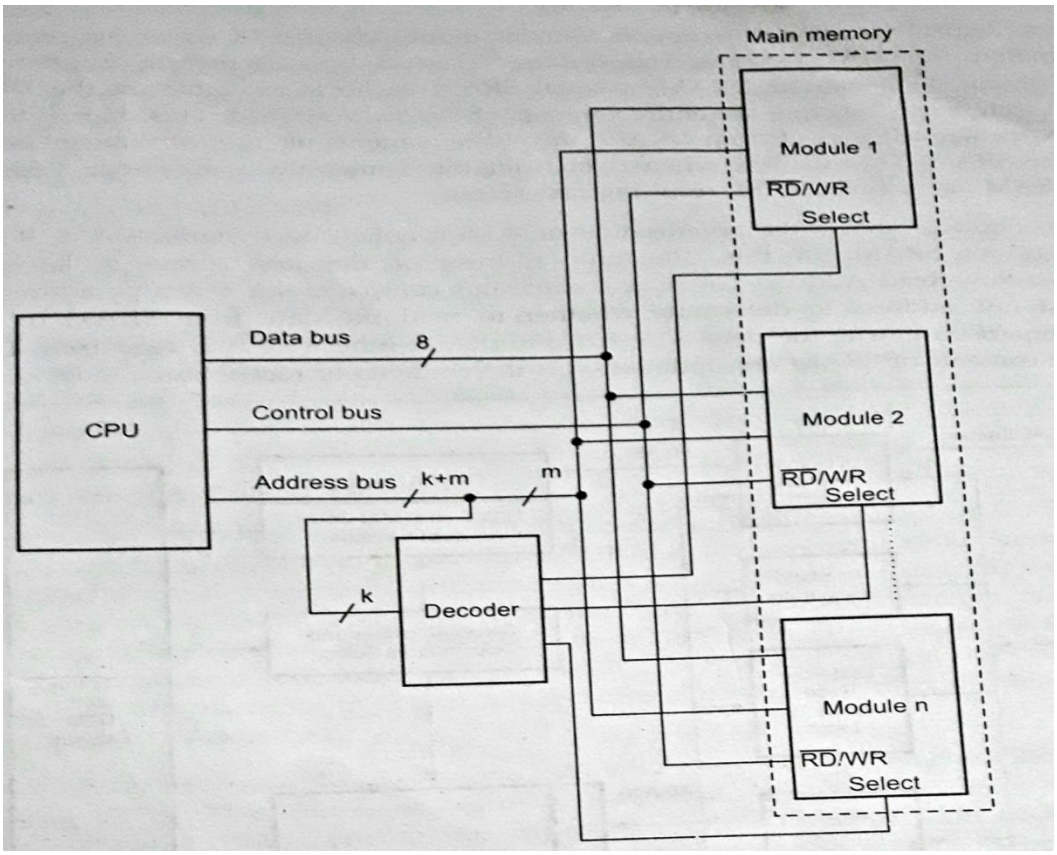


Figure 6.22 Interleaved Memory Organization

The low-order k bits of the memory address are generally used to select a module, and the high-order m bits are used to access a particular location within the selected module. In this way, consecutive address is located in successive modules.Thus,anycomponent of the system that generates requests for access to consecutive memory system as a whole. Itis important to note that implement the interleaved memory structure, there must be 2 modules; otherwise, there will be gaps of nonsexist locations in the memory address space.

The effect of interleaving is substantial. However, it does not speed up memory operation by a factor equal to the number of the module. It reduces the effective memory cycle time by a factor close to the number of modules.

Pipeline and vector processors often require simultaneous access to memory from two or more source. This need can be satisfied by interleaved memory. A CPU with instruction pipeline can also take the advantage of interleaved memory modules so that each segment in the pipeline can access memory independent of memory access from other segments.

6.10 Summary

Programs and the data that processor operate are held in the main memory of the computer during execution, and the data with high storage requirement is stored in the secondary memories such as floppy disk, etc.

The key characteristics of memory systems are: location, capacity, number of word, unit of transfer, access method, performance, physical characteristics.

. The program (code) and data that work at a particular time is usually accessed from the cache memory. This is accomplished by loading the active part of code and data from main memory and cache memory. The cache controller looks after this swapping between main memory and cache memory with the help of DMA controlled. The cache memory just discussed is called secondary cache.

ROM is a read only memory. We can't write data in this memory. It is non-volatile memory i.e. it can hold data even if power is turned off.

There are four types of ROM: Masked ROM, PROM, EPROM and EEPROM or E^2PROM.

There are two types of RAMs:Static RAM,Dynamic RAM.

There are two basic ways we can increase the data-transfer rate across its external interface use a bigger memory word and access more than one word at a time.

The time required to find an object stored in memory can be reduced considerably if objects are selected based on their contents, not on their locations. A memory unit accessed by the content is called an associative memory or content addressablememory (CAM).

Alternative technique to reduce memory access time is memory interleaving. In this technique, to reduce memory is divided into a number of memory modules and the addressare arranged such that the successive words in the address space are placed in different modules.

Sample Questions:

1. List the different characteristics of memory system.
2. Write a note on memory hierarchy
3. Describe the various types of semiconductor memories.
4. Draw and explain the organizations for a) RAM b) PROM

5. What are the types of RAMs ? Explain them in detail.
6. Write short note on Associative Memory**.**
7. Write short note on Interleaved Memory.

Reference Books

1. Computer Organization & Architecture, William Stallings, 8e, Pearson Education.
2. Computer Architecture& Organization, John P. Hayes, 3e, Tata McGraw Hill.
3. Computer Organization, 5e, Carl Hamacher, ZconkoVranesic&SafwatZaky, Tata McGraw-Hill.
4. Computer Architecture & Organization, Nicholas Carter, McGraw Hill
5. Computer System Architecture, M.MorrisMano ,Pearson Education.

❖❖❖

7

CACHE MEMORY SYSTEM

Unit Structure

7.0 Objective

- Discuss cache memory and its functions.
- Understand different memory access.
- Explain floppy disk memory, magnetic tape, Redundant Array of Inexpensive Disk (RAID) , optical memory, CD-ROM

7.1 Introduction

7.1.1 Random Access

If storage locations can be accessed in any order and access time is independent of the location being accessed, the access method is known as random access. The memory that provides such access is known random access memory (RAM) IC (semiconductor) memories are generally of this type.

7.1.2 Serial Access

In serial access memories data must be accessed in a predetermined order via read-write circuitry that is shared by different storage locations. Large serial access memories typically store information in a fixed set of tracks, each consisting of a sequence of 1-bit storage cells. A track has one or more access points at which a read-write head can read/write information to or from the track. A stored item is then accessed serially by moving either the stored information or the read-write heads or both.

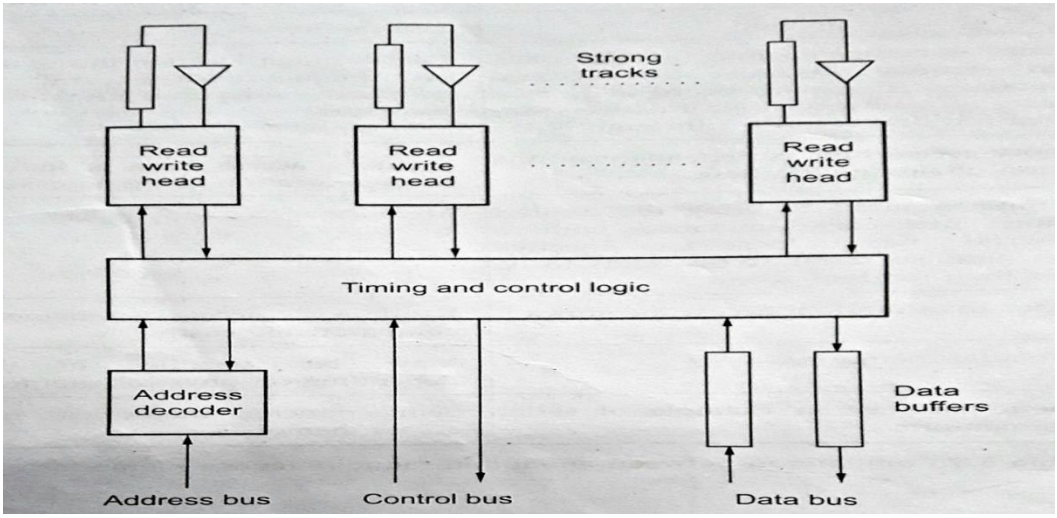


Figure 7.1 Typical organization of serial – access memory unit

The Figure 7.1 shows the typical organization of a memory unit. It is assumed that each word is stored along a single track and that each access results in the transfer of a block of words. The address of the data to be accessed is sent to the address decoder, whose output determines the track to be used (the address of track) an the location of the desired block of information within the track (address of block). The address of track determines the particular read-write head to be selected. The read-write head is then moved to the position from where it can access the desired track.

The read-write head is moved along the track to search for the desired block. A track position indictor generates the address of the block that is currently passing the read-write head. The generated address is compared with the block address produced by the address decoder. When they match, the selected read-write head is enable and data transfer takes place.

7.1.3 Semi-random Access

Memory devices such as magnetic hand disk and CD-ROMs contain many rotating storage tracks. If each track has its own read-write head, the tracks can be accessed randomly, but access within each track is serial. In such cases the access method is semi-random.

Table 7.1 shows the comparison between serial and random access memories.

Table 7.1 Comparison between serial and random access memories

Sr. No.	Serial Access Memories	Random Access Memories
1.	Use serial access method.	Use random access method.
2.	Memory is organized into units of data, called records. Records are accessed sequentially. If current records is 1, then in order to read record N, it is necessary to read physical records 1 through N-1.	Each storage location in the memory has an unique address and it can be accessed independently of the other locations.
3.	Memory access time is dependent on the position of storage location.	Memory access time is independent of storage location being accessed.
4.	The time required to bring the desired location into correspondence with a read-write head increase effective access time, so serial access tends to be slower than random access.	Memory access time is less.
5.	Cheaper than random access memories.	Random access memories are comparatively costly.
6.	Nonvolatile memories	May be volatile or nonvolatile depending on physical characteristics.
7.	Magnetic tape is an example of serial access memories.	Semi-conductor memories are random access memories.

7.2 Cache Memory System

A cache memory system includes a small amount of fast memory (SRAM) and a large amount of slow memory (DRAM). This system is configured to simulate a large amount of fast memory. Figure7.2 shows a cache memory system.

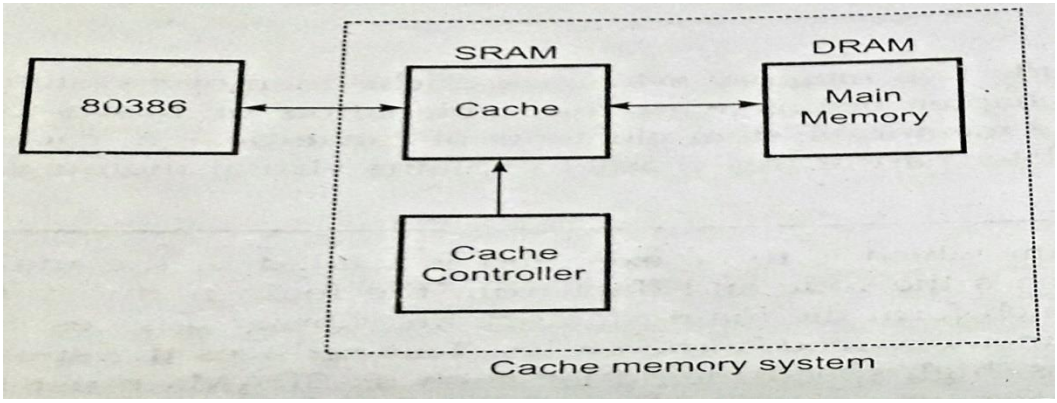


Figure 7.2 Cache memory system

It consists of following sections:

- Cache: This block consists of static RAM (SRAM).
- Main Memory: This block consists of Dynamic RAM (DRAM).
- Cache Controller: This block implements the cache logic.

It decides which block of memory should be moved in or out of the cache block and into or out of main memory, based on the requirements.

7.2.1 Program Locality

In cache memory system, prediction of memory location for the next access is essential. This is possible because computer systems usually access memory from the consecutive locations. This prediction of next memory address from the current memory address is known as program Locality. Program locality enables cache controller to get a block of memory instead of getting just a single address.

The principle of program locality may not work properly when program executes JUMP, and CALL instructions. In case of these instructions, program code is not in sequence.

7.2.2 Locality of Reference

We know that program may contain a simple loop, nested loops, or a few procedures that repeatedly call each other. The point is that many instructions in localized area of the program are executed repeatedly during some time period, and the reminder of the program is accessed relatively infrequently. This is referred to as locality of reference. It manifests itself in two ways: temporal and spatial. The temporal means that recently executed instruction is likely to be executed again very soon. The spatial means that instructions stored nearby to the recently executed instruction are also likely to be executed soon.

The temporal aspect of the locality of reference suggest that whenever an instruction or data is first needed, it should be brought into the cache and it should remain there until it is needed again. The spatial aspect suggests that instead of bringing just one instruction or data from the main memory to the cache, it is wise to bring several instructions and data items that reside at adjacent address as well. We use the term block to refer to a set of contiguous addresses of some size.

7.2.3 Block Fetch

Block fetch technique is used to increase the hit rate of cache. A block fetch can retrieve the data located before the requested byte (look behind) or data located after the requested byte (look ahead), or both. When CPU needs to access any byte from the block, entire block that contains the needed byte is copied from main memory into cache.

The size of the block is one of the most important parameters in the design of a cache memory system. The following section describes the pors and cons of small block size and large block size.

7.2.4 PROS and CONS

1. If the block size is too small, the look ahead and look-behind are reduced, and therefore the hit rate is reduced.
2. Larger blocks reduce the number of blocks that fit into a cache. As the number of blocks decrease, block rewrites from main memory becomes more likely.
3. Due to large size of block the ratio of required data and useless data is less.
4. Bus size between the cache and the main memory increases with block size to accommodate larger data transfers between main memory and the cache, which increases the cost of cache memory system.

7.3 Elements of Cache Design

The cache design elements include cache size, mapping function, replacement algorithm write policy, block size and number of caches.

Cache Size: The size of the cache should be small enough so that the overall average cost per bit is close to that of main memory alone and large enough so that the overall average access time is close to that of the cache alone.

Mapping function: The cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory. Thus we have to use mapping functions to relate the main memory blocks and cache blocks. There are two mapping functions commonly used: direct mapping and associative mapping. Detail description of mapping functions is given in section

Replacement Algorithm: When a new block is brought into the cache, one of the existing blocks must be replaced, by a new block.

There are four most common replacement algorithms:

- Least-Recently Used (LRU)
- First-in-First-out (FIFO)
- Least-Frequently-Used (LFU)
- Random

Cache design change according to the choice of replacement algorithm.

Write Policy: It is also known as cache updating policy. In cache system, two copies of the same data can exist at a time, one in cache and one in main memory. If one copy is altered and other is not, two different sets of data become associated with the same address. To prevent this the cache system has updating system such as : write through system, buffered write through system and write-back system. The choice of cache write policy also change the design of cache. Detail description of write policy is given in section

Block size: It should be optimum for cache memory system. Its importance is already discussed in the section

Number of Caches: When on-chip cache is insufficient the secondary cache is used. The cache design changes as number of caches used in the system changes.

7.4 Cache Organizations

Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

There are two main mapping techniques which decide the cache organization:

1. Direct-mapping technique

2. Associative-mapping technique

The associative mapping technique is further classified as fully associative and set associative techniques.

To discuss these techniques of cache mapping we consider a cache consisting of 128 block of 32 words each, for a total of 4096(4K) words, and assume that the main memory has 64K words. This 64K words of main memory is addressable by a 16-bit address and it can be viewed as 2 K blocks of 16 words each. The group of 128 blocks of 32 words each in main memory from a page.

7.4.1 Direct Mapping

It is the simplest mapping technique. In this technique, each block from the main memory has only one possible location in the cache organization. In this example, the block I of the main memory maps on to block I module 128 of the cache, as shown in figure 7.3 Therefore, whenever one of the main memory blocks 0, 128,256,…… is loaded in the cache, it is stored in cache block 0. Blocks 1, 129, 257,…… are stored in cache block 1, and so on.

To implement such cache system, the address is divided into three fields, as shown in Figure 7.3 .The lower order 5-bits select one of the 32 words in a block. This field is known as word field. The second field known as block field is used to distinguish a block from other blocks. When a new block enters the cache, the 7-bit cache block field determiners the cache position in which this block must be stored. The third field is a tag field. It is used to store the high-order 4-bits of memory address of the block. These 4-bit (tag bits) are used to identify which of the 16 blocks (pages) that are mapped into the cache.

When memory is accessed, the 7-bit cache block field of each address generated by CPU points to a particular block location in the cache. The high-order 4-bits of the address are compared with the tag bits associated with that cache location. If they match, then the desired word is in that block of the cache. If there is no match, then the block containing the required word must first be read from the main memory and loaded into the cache. This means that to determine whether requested word is in the cache, only tag field is necessary to be compared. This needs only one comparison.

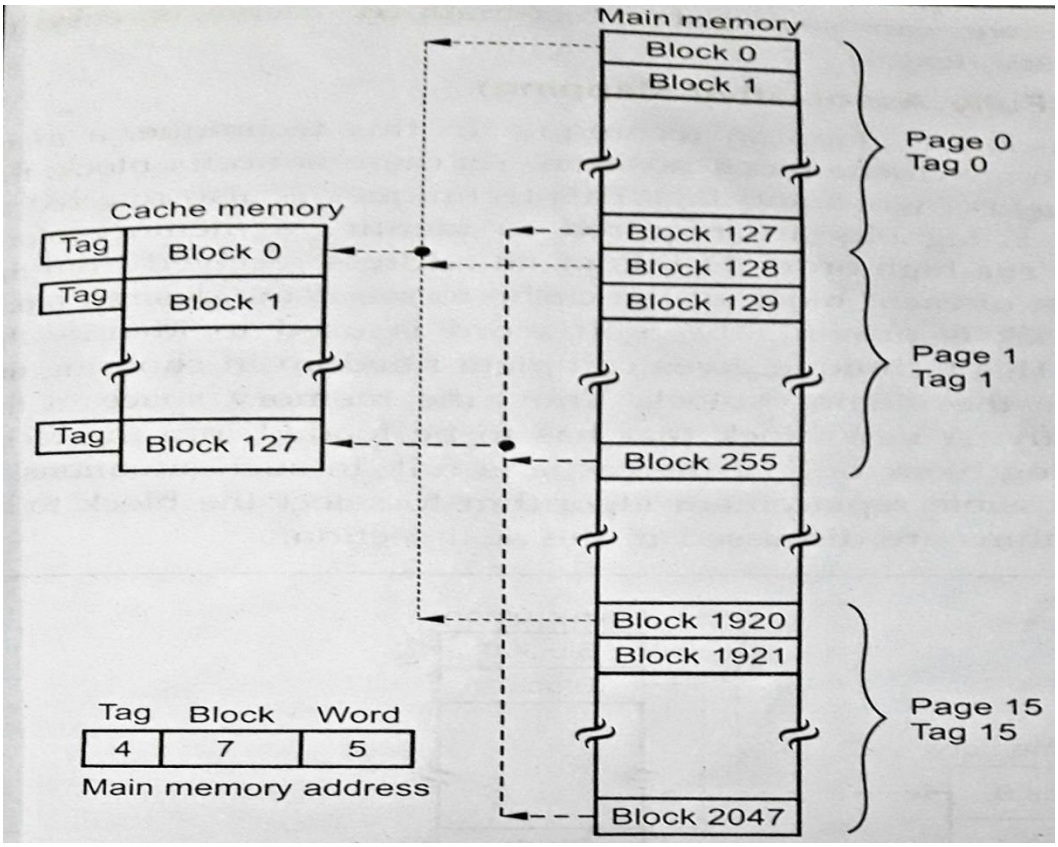


Figure 7.3 Direct mapped cache

The main drawback of direct mapped cache is that if processor needs to access same memory locations from two different pages of the main memory frequently, the controller has to access main memory frequently. Since only one of these locations can be in the cache at a time. For example , if processor want to access memory location 100H from page 0 and then from page 2, the cache controller has to access page 2 of the main memory. Therefore, we can say that direct-mapped cache is easy to implement, however, it is not very flexible.

7.4.2 Associative Mapping (Fully Associative Mapping)

The figure 7.4 shows the associative mapping technique. In this technique, a main memory block can be placed into any cache block position. As there is no fix block, the memory address has only two fields: word and tag. This technique is also referred to as fully-associative cache. The 11-tag bits are required to identify a memory block when it is resident in the tag bits of each block of the cache to see if the desired block is present. Once the desired block is present, the 5-bit word is used to identify the necessary word from the cache. This technique gives complete freedom in choosing the cache location in which to place the memory block. Thus, the memory space in the cache can be used more efficiently. A new block that has

to be loaded into the cache has to replaced (remove) an existing block only if the cache is full. In such situations, it is necessary to use one of the possible replacement algorithm to select the block to be replaced.

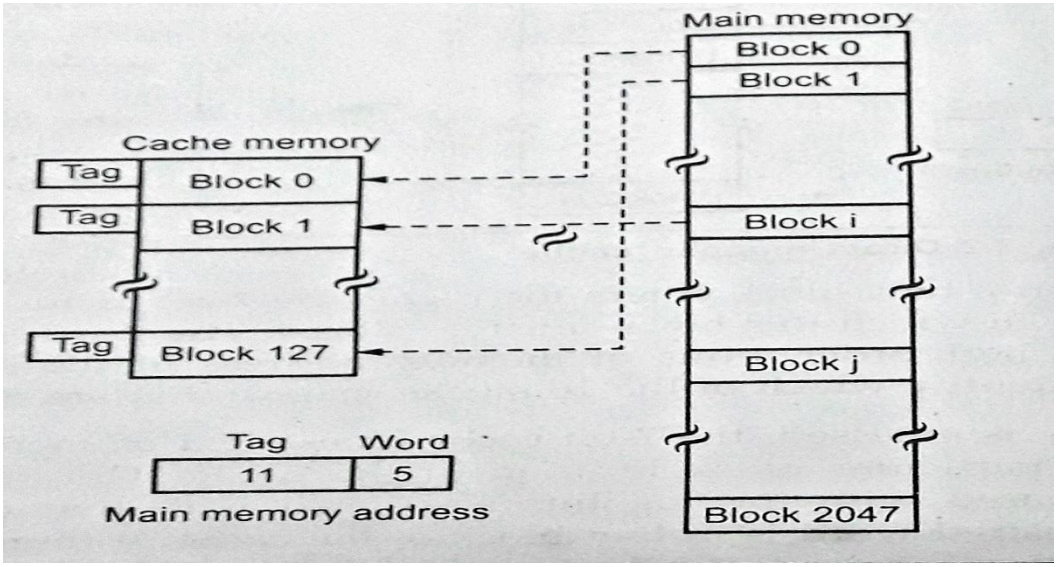


Figure 7.4 Associative-mapped cache

In associative-mapped cache, it is necessary to compare the higher-order bits of address of the main memory with all 128 tag corresponding to each block to determine whether a given is in the cache. This is the main disadvantage of associative-mapped cache.

7.4.3 Set-Associative Mapping

The set-associative mapping is a combination of both direct and associative mapping. It contains several groups of direct mapped blocks that operate as several direct mapped caches in parallel. A block of data from any page in the main memory can go into a particular block location of any direct-mapped cache. Hence the contention problem of the direct-mapped technique is eased by having a few choices for block placement. The required address comparisons depend on the number of direct mapped caches in the cache system. These comparisons are always less than the comparisons required in the fully-associative mapping.

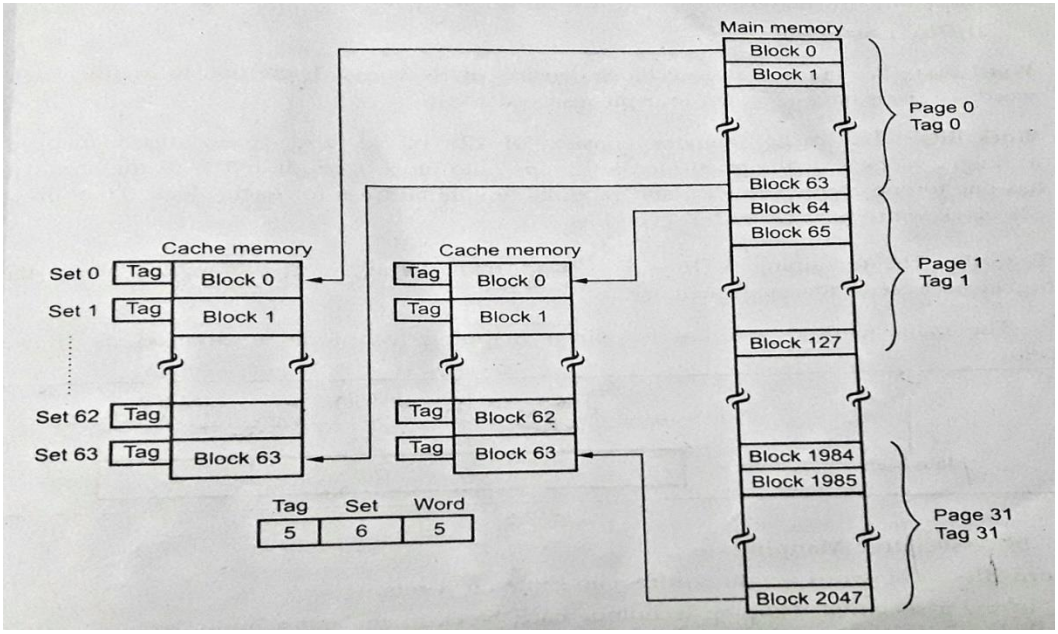


Figure 7.5 Two-way set associative cache

Figure 7.5.shows two way set associative cache. Each page in the main memory is organized in such a way that the size of each page is same as the size of one directly mapped cache. It is called two-way set associative cache because each block from main memory has two choices for block placement. In this technique, block 0, 64,128,…….,1984 of main memory can map into any of the two (block 0) blocks of set 0, block1, 65,129,……., 1985 of main memory can map into any of the two (block 1) blocks of set 1 and so on.

As there are two choices, it is necessary to compare address of memory with the tag bits of corresponding two block locations of particular set. Thus for two-way set associative cache we required two comparisons to determine whether a given block is in the cache. Since from there are two direct mapped caches, any two bytes having same offset from different pages can be in the cache at a time. This improves the hit rate of the cache system.

To implement set-associative cache system, the address is divided into three fields, as shown in figure. The 5-bits word field selects one of the 32 words in a block. The set field needs 6-bits to determine the desired block from 64 sets. However, there are now 31 pages. To identify which of the 32 blocks (pages) that are mapped into the particular set of cache, five tag bits are required.

7.5 Virtual Memory

In most modern computers, the physical main memory is not as large as the address space spanned by an address issued by the processor. Here, the virtual memory technique is used to extend the apparent size of the physical memory. It uses secondary storage such as disks, to extend the apparent size of the physical memory. It uses secondary storage such as disks, to extend the apparent size of the physical memory. Let us see how this technique works.

When a program does not completely fit into the main memory, it is divided into segments. The segments which are currently being executed are kept in the main memory and remaining segments are stored in the secondary storage devices, such as a magnetic disk. If an executing program needs a segment which is not currently in the main memory, the required segment is copied from the secondary storage device. When new segment of a program is to be copied into a main memory, it must replace another segment already in the memory. In modern computers, the operating system moves program and data automatically between the main memory and secondary storage. Techniques that automatically swaps program and data blocks between main memory and secondary storage device are called virtual memory.The address that processor issues to access either instruction or data are called virtual or logical address.These addresses are translated into physical addresses by a combination of hardware and software components. If a virtual address refers to a part of the program or data space that is currently in the main memory, then the contents of the appropriate location in the main memory are accessed immediately. On the other hand, if the reference address is not in the main memory, its contents must be brought into a suitable location in the main memory before they can be used.

We have seen that, how virtual memory removes the programming burdens of a small, limited amount of main memory. Along with this it also allows efficient and safe sharing of memory among multiple programs. Consider a number of programs running at once on a computer. The total memory required by all the programs may be much larger than the amount of main memory available on the computer, but only a fraction of this memory is actively being used at any point in time. The main memory need to contain only the active portions of the many programs. This allows us to efficiently share the processor as well as the main memory.

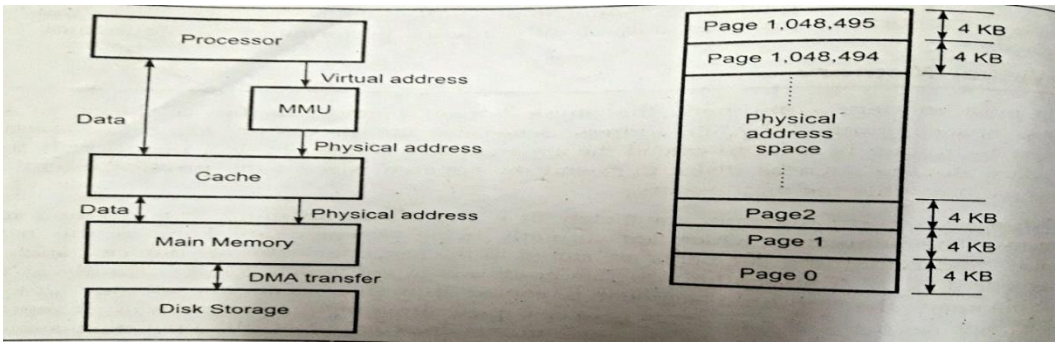


Figure 7.6 Virtual memory organization

Figure 7.7 Paged organization of the physical address space

Figure 7.6 Shows a typical memory organization that implements virtual memory. The memory management unit controls this virtual memory system. It translates virtual address into physical address is to assume that all programs and data are composed of fixed length unit called pages, as shown in the pages constitute the basic unit of information that is moved between the main memory and the disk whenever the page translation mechanism determines that a swaping is required.

7.5.1 Address Translation

In virtual memory, the address is broken into a virtual page number and a page offset. Figure 7.8 shows the translation of the virtual page number to a physical page number. The physical page number constitutes the upper portion of the physical address, while the page offset, which is not changed, constitutes the lower portion. The number of bits in the page offset field decides the page size.

The page table is used to keep the information about the main memory location of each page. This information includes the main memory address where the page is stored and the current status of the page. To obtain the address of the corresponding entry in the page table the virtual page number is added with the contents of page table base register, in which the starting address of the page table is stored. The entry in the page table gives the physical page number, in which offset is added to get the physical address of the main memory.

If the page required by the processor is not in the main memory, the page fault occurs and the required page is loaded into the main memory from the secondary storage memory by special routine called page fault routine. This technique of getting the desired page in the main memory is called demand paging.

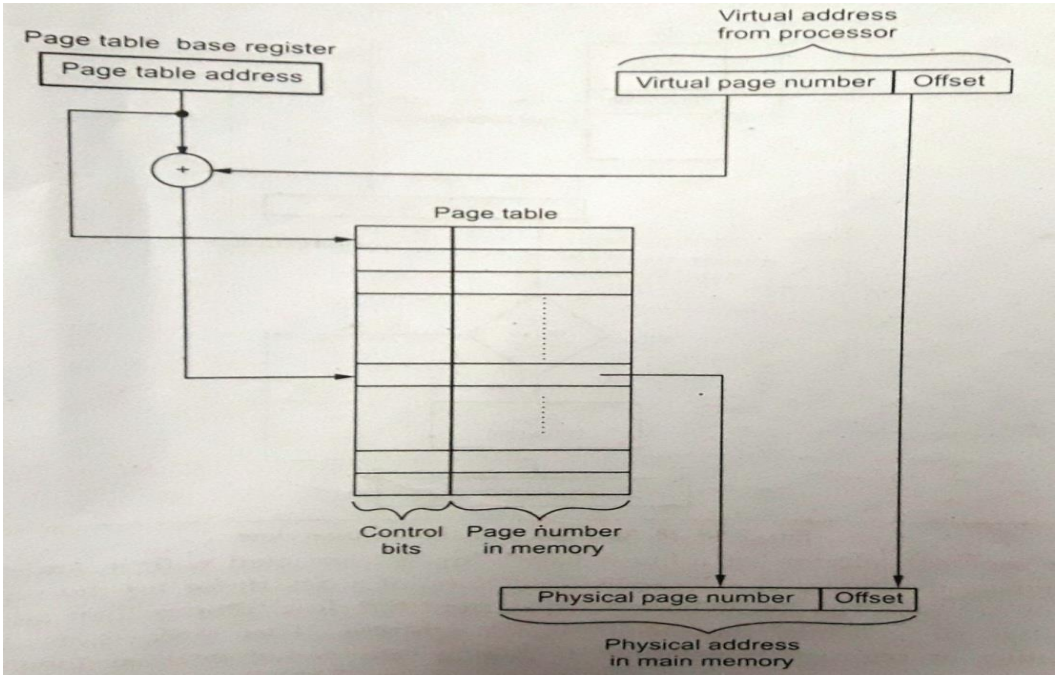


Figure 7.8 Virtual to physical address translation

To support demand paging and virtual memory processor has to access page table which is kept in the main memory. To avoid the access time and degradation of performance, a small portion of the page table is accommodated in the memory management unit. This portion is called translation look aside buffer (TLB) and it is used to hold the page table entries that corresponds to the most recently accessed pages. When processor finds the page table entries in the TLB it does not have to access page table and saves substantial access time.

7.6 Magnetic Disk Memory

7.6.1 Magnetic Surface Recording

A magnetic disk is a thin, circular metal plate. It is coated with a thin magnetic film, usually on both sides. Digital information is stored on the magnetic disk by magnetizing the magnetic surface in a particular direction as shown in the Figure 7.9 (a)

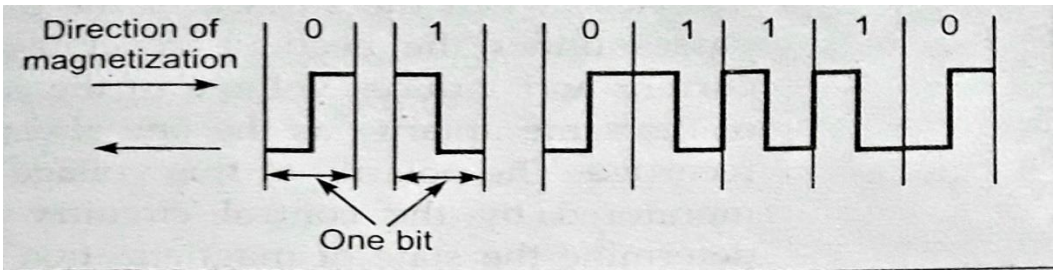


Figure 7.9(a) Bit representation of magnetic disk

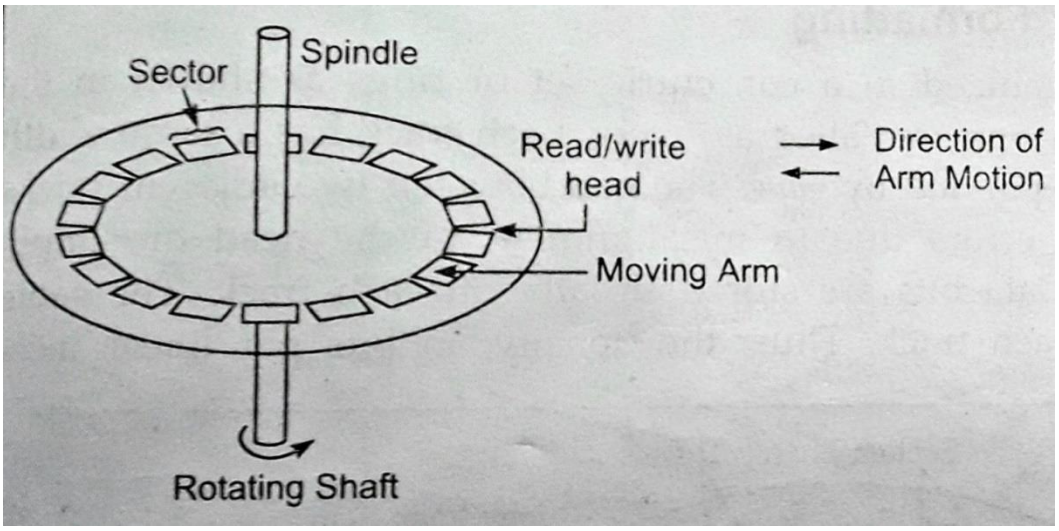


Figure 7.9 (b) Rotating shaft

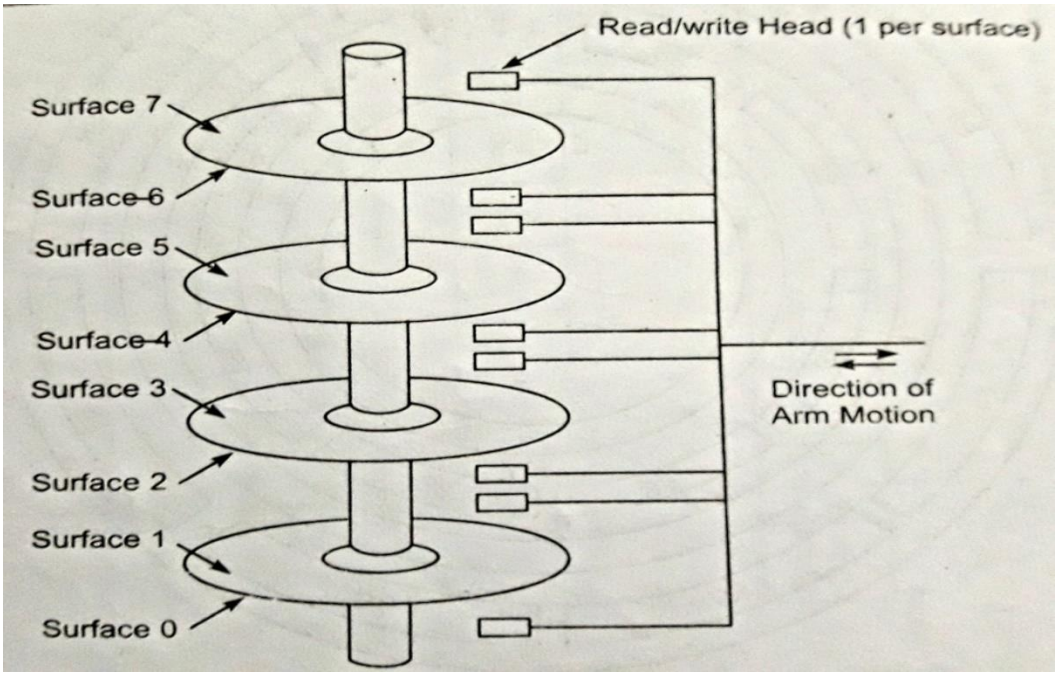


Figure 7.9 (c) Mechanical structure

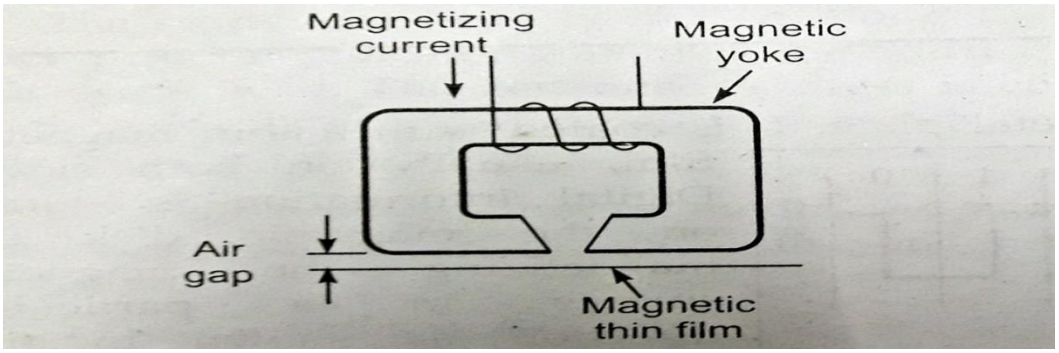


Figure 7.9 (d) Read / Write head detail

The disk are mounted`on a rotary drive so that the magnetized surface moves in close proximity to magnetizing coil or head as shown in Figure 7.9(b). The head consist of a magnetic yoke and the magnetic can be stored on the magnetic film by applying current pulses of suitable polarity to the magnetizing coil. This causes the magnetization of the film in the area immediately below the head. The same head is used for reading the stored information. In this case, when the surface of the disk passes under the head, it generates a current and induces voltage in the coil of the same polarity of this voltage is monitored by the control circuitry to determine the state of magnetization of the film.

7.6.2 Data Organization and Formatting

The data on the disks is organized in a concentric set of rings as shown in the Figure 7.10. These concentric set of rings are called as tracks. Each track has a same width as head and adjacent tracks are separated by gaps. The gap between two adjacent tracks prevents, or at least minimizes, errors due to misalignment of the head or simply interference of magnetic fields. Data bits are stored serially on each tracks. The same numbers of bits are stored on each track. Thus, the density, in bits per linear inch, increase as we move from the outermost track to the inner most track. The tracks are further divided into sectors, as shown in the Figure 7.10Each sector stores a block of data which can be transferred to or from the disk. To avoid magnetic interference between two adjacent sector the gap is introduced between two adjacent sectors.

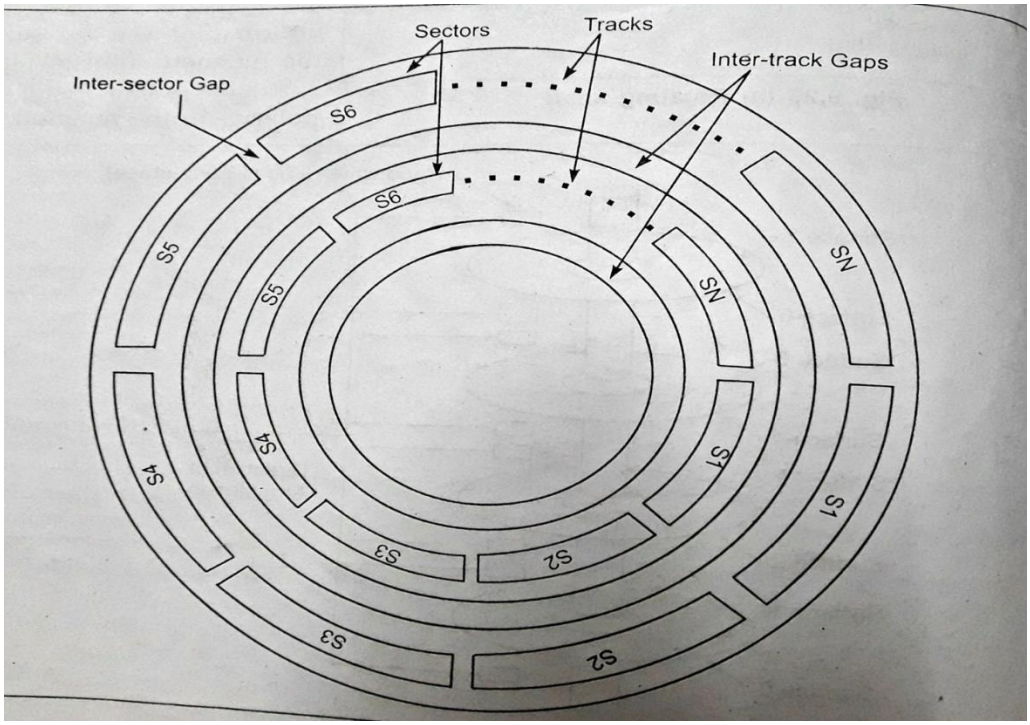


Figure7.10 Disk Data Layout

Data on disk is addressed by specifying the disk number or head number, track number and the sector number. The start and end of each sector is determined by the control data stored on each sector.

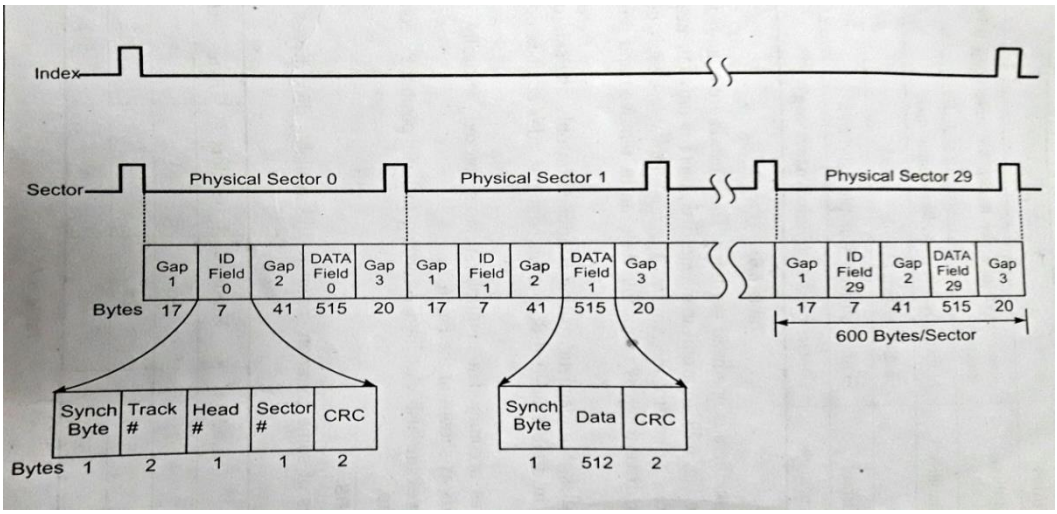


Figure 7.11 Winchester disk track format (Seagate ST506)

Figure 7.11shows disk format for disk. As shown in the Fig. each track contains 30 fixed length sectors of 600 bytes each. Each sector holds512 bytes of data plus control information useful to the disk controller. The control information is represented by ID field, which is used as an unique identifier to locate a particular sector.

7.6.3 Characteristics

The Table 7.2 lists the important characteristics of a magnetic disk.

Table 7.2

Attributes	**Type**
Head Motion	Fixed head (one per track), Movable head (one per surface)
Disk Portability	Nonremovable disk, Removable disk
Sides	Single - sided, Double – sided
Disk/surface	Single -surface, Multiple – surfaces
Head Mechanism	Contact fixed gap, Aerodynamic gap

In fixed head disk, there is one head per track. These heads are mounted on a rigid arm as shown in the Figure 7.9. In a movable head disk, there is only one head and it is moved between tracks by control mechanism. The nonremovable disk is permanently mounted in the disk drive whereas removable disk can be removed and replaced with another disk.

In single sided disk, the magnetic coating coating is applied on only one side of the disk. On the other hand, in double sided disk it is on both the sides of disk, doubling the storage capacity.

Some disk drives accommodate multiple surfaces stacked vertically apart with separate pair of heads as shown in the Figure 7.9

Finally, disk drives are also classified according to head positions : contact, fixed gap or aerodymic gap.

7.6.4 Specifications

The specifications of some common commercially available disk drives are shown in Table

Table 7.3

Capacity	Cylinders	Heads	Sectors	Bytes/Sector
4.302 GB	523	255	63	512
2.159 GB	523	128	63	512
1.08 GB	523	64	63	512

7.7 Floppy Disk Memory

The magnetic disks that we described above are called hard-disks. These disks, in the form of disk – packs are suitable for storage of large amount of data. Floppy disks are smaller, simpler and cheaper disk units that consist of a flexible, removable, plastic diskettes coated with magnetic material. The diskette is enclosed in a plastic jacket, which has an opening where the head makes contact with the diskette. A hole in the diskette allows a spindle mechanism in the disk drive to position and rotate the diskette as shown in the Figure 7.12

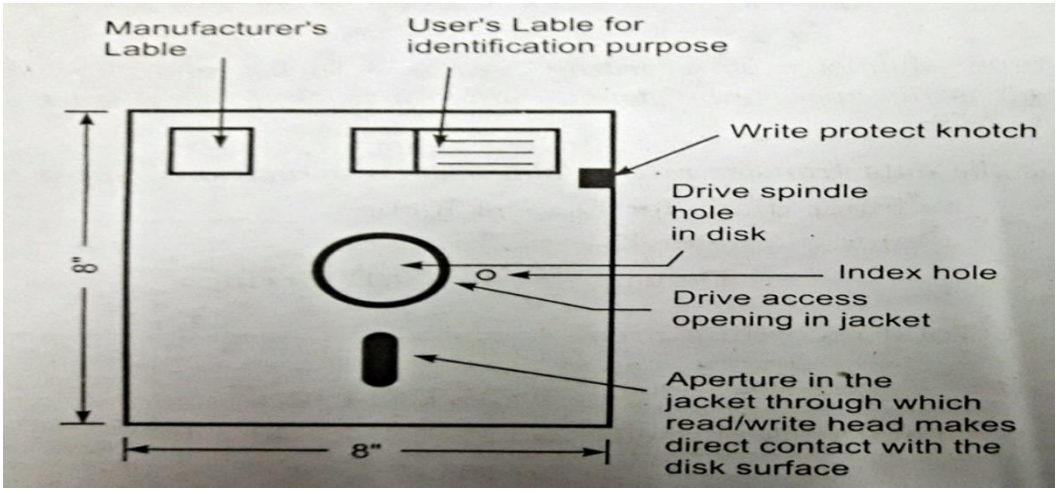


Figure 7.12: Floppy Disk

7.8 Magnetic Tape

Magnetic tape is one of the most popular storage medium for large data that are a very thin and ½ inch or ¼ inch wide plastic tape. Usually, iron oxide is used as a magnetizing material. The tape ribbon itself is stored in reels similar to the tape used on a tape recorder except that it is of high quality and more durable. Like audio tape, computer tape can be erased and reused indefinitely. Old data on a tape are automatically erased as new data are recorded in the same area.

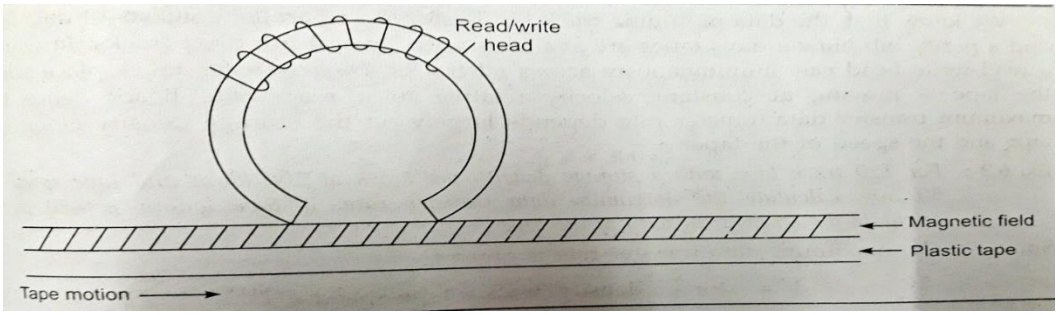


Figure 7.13 Magnetic recording with read / write head

The information is recorded on the tape with the help of read/write head. It magnetizes or nonmagnetizes tiny invisible spots (representing 1's and 0's) on the iron oxide side of the tape, as shown in the Figure

Usually seven or nine bits (corresponding to one character) are recorded in parallel across the width of the tape, perpendicular to the direction of motion. A separate read/write head is provided for each bit position on the tape, so that all bits of characters can be read or written in parallel. One of the character bit is used as a parity bit. This is illustrated in Figure

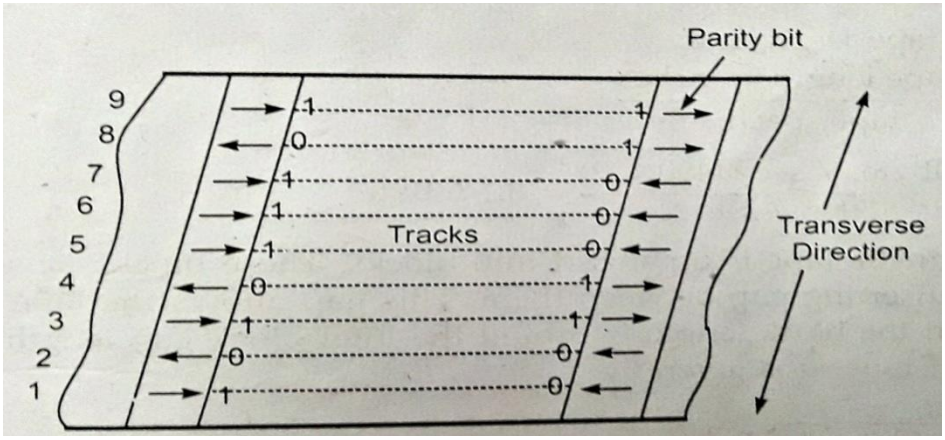


Figure 7.14 Magnetic recorded tape

Data on the tape are organized in the form of separated by gaps, as shown in the Figure. A set related records constitutes a file. A file mark is used to identify the beginning of the file. The file mark is a special single or multiple character record, file mark (first record) can be used as a header or identifier or for this file. This allows user to identify a particular file from number of files recorded on the tape.

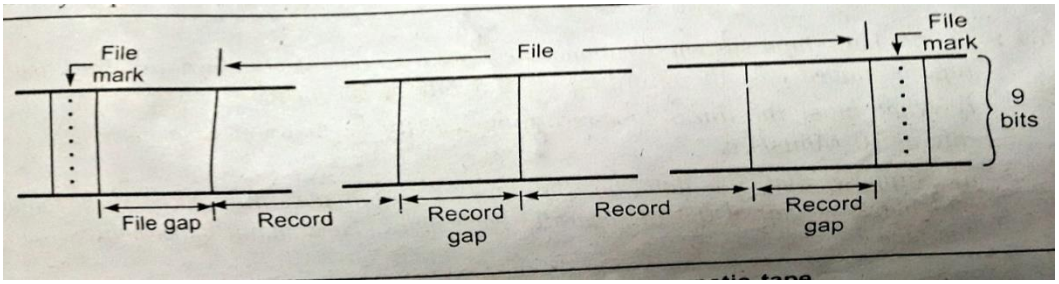


Figure 7.15 : Organization of data on magnetic tape

We know that, the data on a nine track tap is stored in parallel consists of data byte and a parity bit. Now-a-days tapes are available with several hundred tracks. In which a read-write head can simultaneously access all tracks. Data transfer takes place when the tape is moving at constant velocity relative to a read-write head; hence the maximum transfer data-transfer data- transfer rate depends largely on the storage density along the tape and the speed of the tape.

7.8.1 Advantages of Magnetic Tape

1. **Unlimited storage:** The storage capacity of magnetic tape is virtually unlimited because we can use as many tapes as required for recording.
2. **High data density:** A typical 10.5 inch reel of magnetic tape is 28800 inches long and is able to hold upto 6250 characters per inch. Therefore, the maximum capacity of magnetic tape is about 180 million characters.

3. **Low cost:** The cost of magnetic tape is much less as compared to other storage devices.
4. **Repaid transfer rate:** Data transfer rate for magnetic tape can be in excess or 1 million bytes per second.
5. **Ease of handling:** Since it is compact and light weighted it is easier to use it.
6. **Portability:** The tape is wound on a reel and it is packed in a plastic package, hence it is very convenient way of carrying information from one place to another.

7.8.2 Disadvantages of magnetic tapes

1. **Sequential access:** Magnetic tape is a sequential access device and hence data recorded on tape cannot be accessed directly. Thus more time is required for accessing the record.
2. **Environmental restriction:** The magnetic tapes suffer from dust and uncontrolled humidity or temperature levels. This may cause data reading errors.

7.8.3 Comparison between magnetic disk and magnetic tape

Attribute	Magnetic tape	Magnetic disk
Access	Sequential	Direct or sequential
Access time	More	Less
Environmental Restrictions	More	Less
Data transfer rate	Comparatively	More
Reliability	Less	More
Portability	More	Less
Cost	Less	High
Application	Used for backups	Used as a secondary storage device in computer systems

7.9 Redundant Array of Inexpensive Disk (RAID)

The magnetic disk system just described provide a single, serial bit stream during data transfer. Recent disk- system developments have been directed at decreasing access time and increasing bandwidth during data flow through the use of multiple disk operating parallel. In multiple disk system, array of inexpensive disks operate

independently and in parallel to improve the speed of data transfer. This system is referred to as RAID (Redundant Arrays of Inexpensive Disk). In a RAID system, a single large file is stored in several separate disk units by breaking the file up into a number of smaller pieces and storing these pieces on different disks. This is called data stripping. When the file is accessed for a read, all disk deliver their data in parallel. The total file transfer time can be given as, the transfer time that would be required in a single disk system divided by the number of disks used in the array. However, the time required to locate the beginning of the data on each disk, access time, is not reduced.

We have seen in RAID systems, a single file modules can be accessed simultaneously. This approach improves the speed of data transfer. The another approach in RAID system is to add redundancy to improve reliability. This can be achieved by duplicating all pieces of the distributed, or striped, file on addition disk units, thus doubling the total number of disks used. With the use of multiple disks, there is a wide variety of ways in which the data can be organized and in which redundancy can be added to improve reliability. These are called as RAID levels. RAID levels do not imply a hierarchical relationship but designate different design architectures.

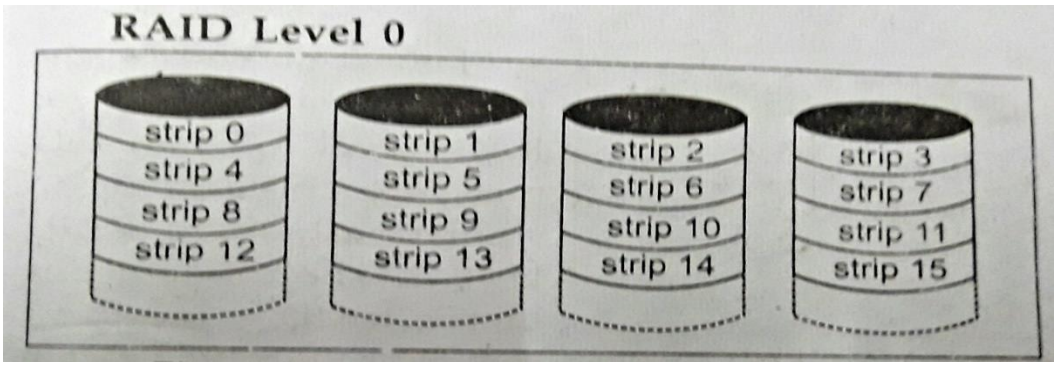


Figure 7.16 RAID level 0 (Non –Redundant)

Figure shows the data stripping used in the RAID level 0. The strips are mapped round-robin to consecutive array members. A set of logically consecutive strips that maps exactly one strips that maps exactly one strip to each array member is referred to as a stripe. In an n-disk array, the first n logical strips are physically stored as the first strip on each of the n disks, the second n strips are distributed as the second strips on each disk, and so on. With this design architecture, two different I/O requests for two different blocks of data can be processed in parallel, reducing the I/O queuing time. The RAID level 0 architecture achieves the parallism but level 0 is not a true member of the RAID family.

RAID Level 1

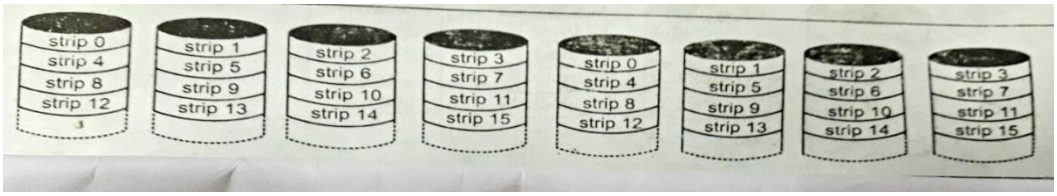


Figure 7.17 RAID level 1 (Mirrored)

Figure shows how the data stripping is used and redundancy is achieved in RAID level 1. In RAID level 1, the redundancy is achieved by just duplicating all the data. The data stripping is used, same as in RAID level 0. But in this case, each logical strip is mapped to two separate physical disks so that every disk in the array has a mirror disk that contains the same data.

RAID level 2

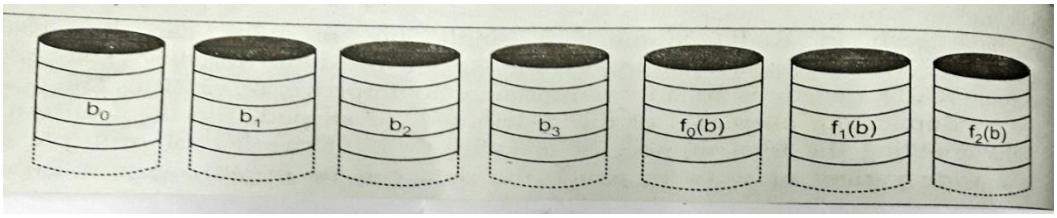


Figure 7.18 RAID level 2 (Redundancy through hamming code)

Figure shows the design architecture of RAID level 2. Here, data stripping used, same way as in RAID level 0 and RAID level 1. But to achieve reliability, an error correcting code is calculated across corresponding bits on each data disks, and the bits of the code are stroed in the corresponding bit position on multiple parity disks. Usually hamming code is used as single error corrector and double error detector (SEC- DED).

RAID Level 3

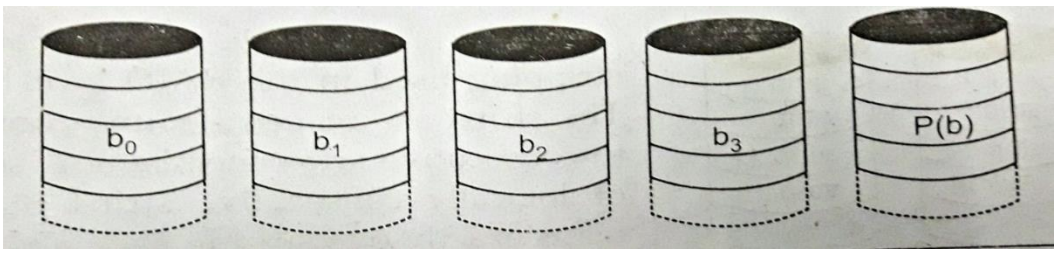


Figure 7.19 : RAID level 3 (Bit – Interleaved Parity)

Figure shows the design architecture for RAID level 3. Here, data bits are organized in similar fashion to RAID level 2. The difference is that RAID level 3 requires

only a single redundant disk, no matter how large the disk array. The data stripping is used, similar to the other RAID levels. In RAID level 3 instead of an error correcting code, a simple parity bit is computed for the set of individual bits in the same position on all of the data disks.

RAID level 4

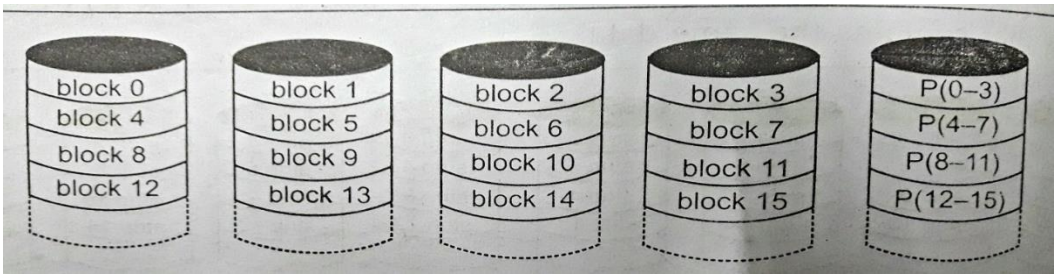


Figure 7.20 RAID level 4 (Block level parity)

Figure shows the design architecture for RAID level 4. Here, data stripping used is same as used in other levels. But the size of strip is large which represents one data block. In RAID level 4, a bit by bit parity strip is calculated across corresponding data blocks on each data disk, and the parity bits are stored in the corresponding strip on the parity disk. For each read/write operation parity bits are checked for data reliability. As a result, parity disk becomes a bottleneck.

RAID Level 5

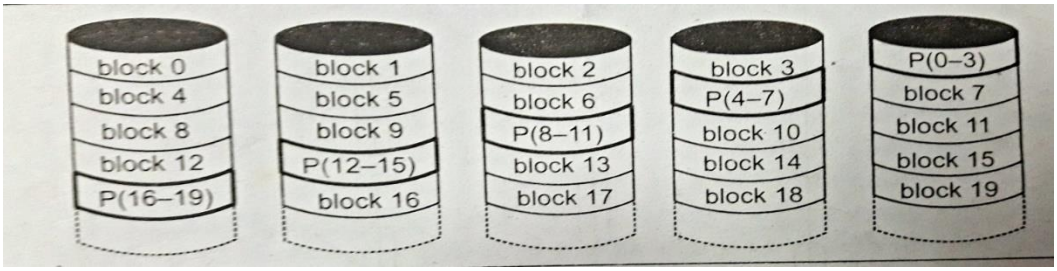


Figure 7.21 RAID level 5 (Block – Level distributed parity)

Figure 7.21shows thedesign architecture for level 5. Here, the data bits are organized in a similar fashion to RAID level 4. The difference is that RAID5 distributes the parity strips across all disks. Due to the distribution of parity strips, the bottleneck found in RAID level 4 can be avoided.

In RAID levels from 1 to 5, the redundant disk capacity is used to either duplicate all data or to store parity information which guarantees data recoverability in case of a disk failure.

7.10 Optical Memory

In 1983, the first optical memory device, compact disk (CD) was successfully launched. The CD is a nonerasable disk that can store more than 60 minutes of audio information on one read-only memory (CD-ROM)

- Compact disk read-only memory (CD-ROM)
- Write-once read-many (WORM) and
- Erasable optical disk

7.10.1 Compact disk read-only memory (CD-ROM)

The compact disk (CD) and compact disk read only memory (CD-ROM) uses the similar technology. However, the CD-ROM players are more rugged and have error correction devices to ensure that data are properly transferred from disk to computer. Both disks consists of a rotating disk which is coated with a thin highly reflective material. In this system, the data recording is done by focusing a laser beam on the surface of the spinning disk, as shown in the Figure 7.22

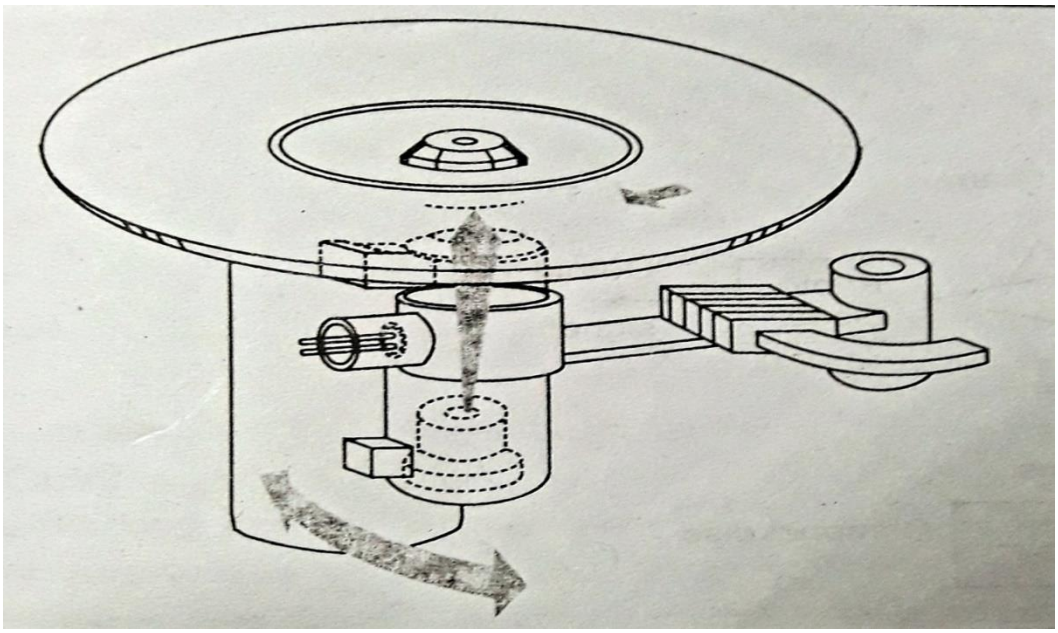


Figure 7.22 Compact disk with laser beam control mechanism

The laser beam is turned ON and OFF according to digital signal. Because of which tiny holes (pits) are brunt into the metal coating of the disk along its tracks, as shown in the Figure 7.23

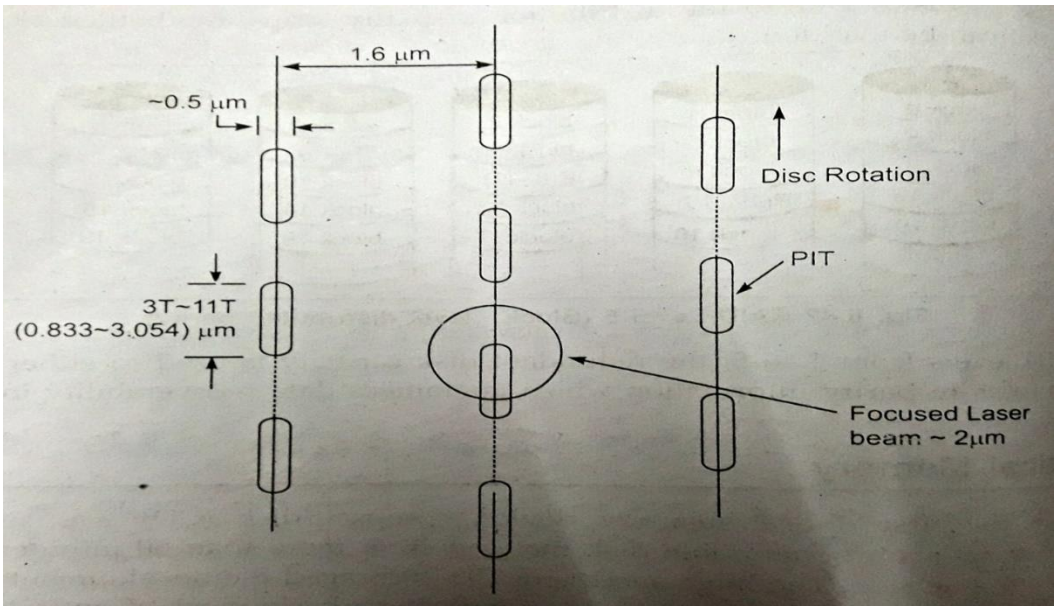


Figure 7.23 Recorded portion of CD showing pits along its tracks

The data is recovered from the compact disc with an optical pickup, which moves across the surface of the rotating disc. Figure 7.24 shows optical pickup system.

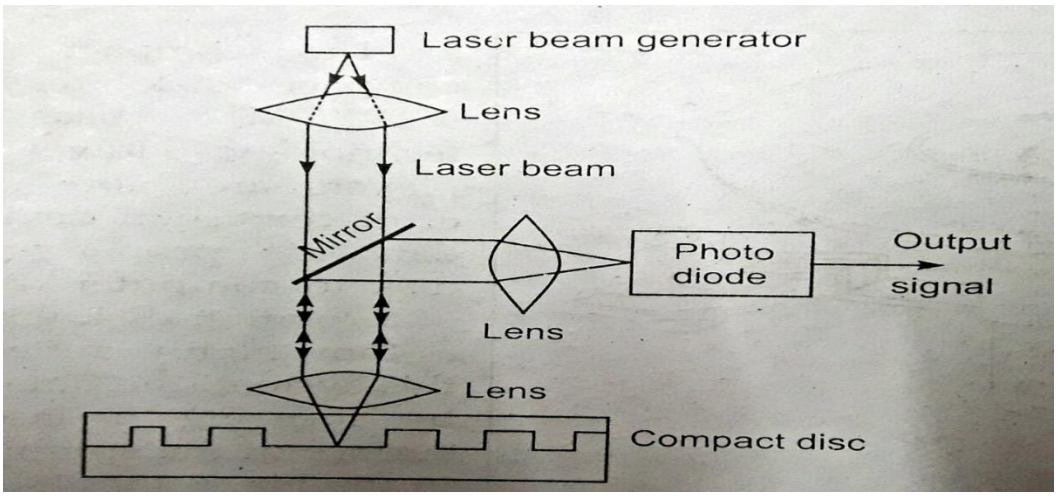


Figure 7.24 Optical pickup system

Optical pickup system uses a semiconductor laser with approximately 5 milliwatt optical output irradiating a coherent beam with a 790 nanometer wavelength. This beam is incident on the compact disc through a half silver mirror. The mirror allows the beam to pass through itself but does not allow the returing beam to pass. When beam strikes a band interval between two pits, the light is almost totally reflected. When it strikes a pit a lower intensity light is returned. The change in intensity represent 1 and unchanged intensity represents 0.

A pit near the center of a rotating disk travels slower than a pit on the outside. It is necessary to Compensate for the variation in speed so that the laser can read all the pits at the same rate. This can be done by increasing the spacing between bits of information recorded in segments of the disk. Thus the information can be read at the same rate by rotating the disk at a fixed speed, known as the **constantangular velocity** (CAV). The Figure7.25 shows the layout of a disk using CAV. As shown in the Fig. the disk is divided into a number of pie-shaped sectors and into a series of concentric tracks. This arrangement makes access to CD-ROM very easy. It is possible to access CD-ROM by specifying particular track and sector. However, due to this arrangement a lot of space in the outer track is wasted.

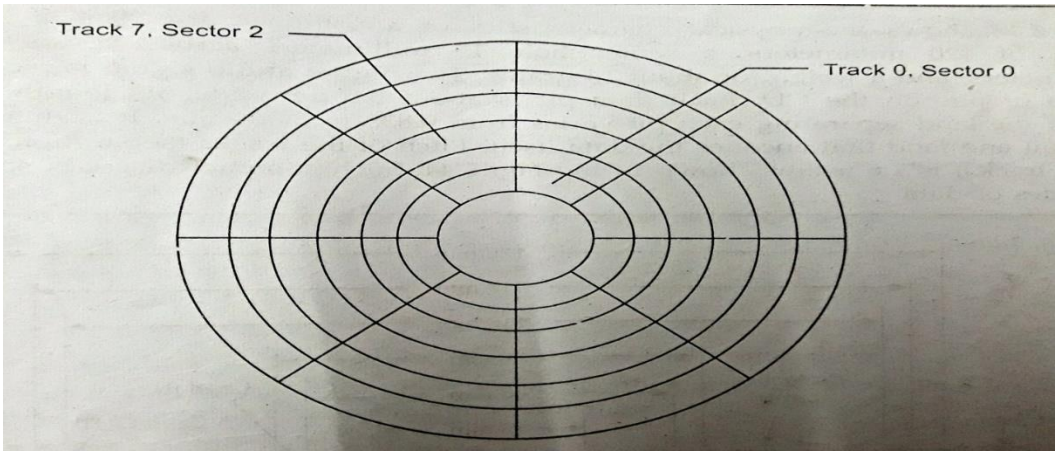


Figure 7.25 Disk layout using constant angular velocity

Due to this disadvantage of CAV method, now-a-days **constant linear velocity (CLV)** method is used to put information on the disk. In this method, the information is packed evenly across the disk in segments of the same size, and these are scanned at the same rate by rotating the disk at a variable speed. The Figure 7.26 shows the layout of a disk using CLV.

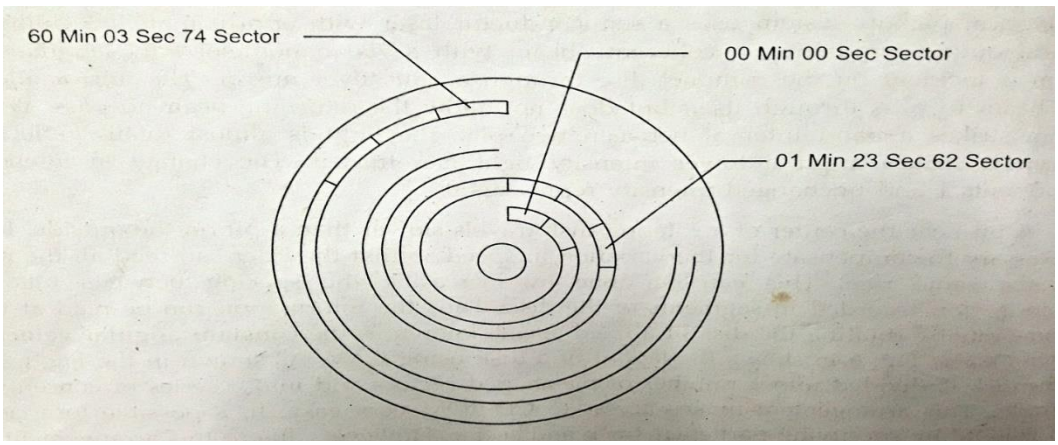


Figure 7.26 Disk layout using constant linear velocity

Figure 7.27shows the physical characteristics of a compact disc. It has an outer diameter of 120 millimeters, a center hole 15 millimeters across, a thickness of 1.2 millimeters, and a weight of about 14 grams. The digital audio signal is recorded in the form of pits on the CD. Each data pit varies from 0.833 to 3.054 It is that varying ratio of pit and land that encodes the data itself. Pitch of the tracks (separation between adjacent tracks) is 1.6µm, as shown in the Fig. such compact disc can store upto 650 Mbytes of data.

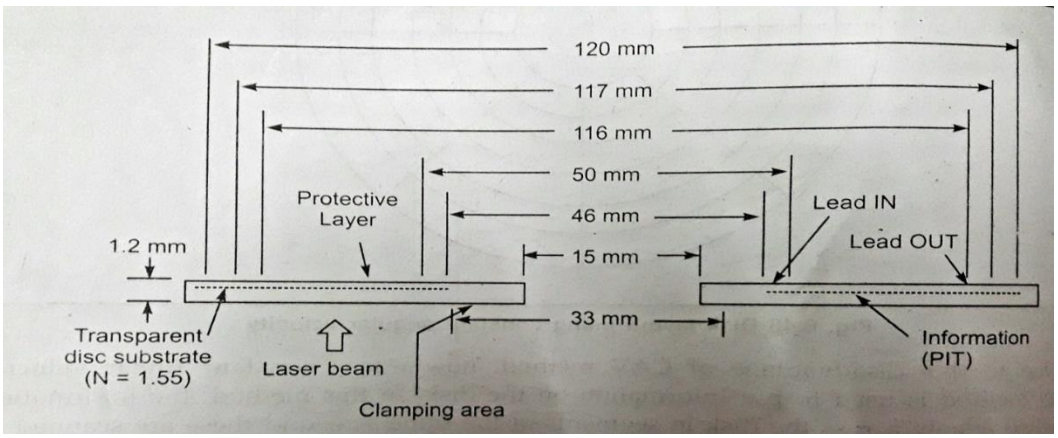


Figure 7.27 physical characteristics of a compact disc

The Figure 7.28shows the format for typical data block on the CD-ROM. Such blocks are sequentially stored on the CD-ROM. Each data block consist of following fields:

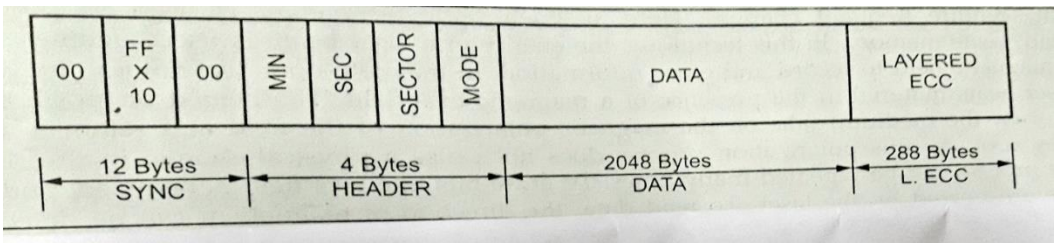


Figure 7.28 Block format for CD- ROM

SYNC : This field identifies the beginning of a block. It consists of one byte of all zeros, 10 bytes of all ones and then one byte of all zeroes.

HEADER : This field consist of address and mode information. Mode 1 indicates the blank data field, Mode 2 specifies the use of an error-correcting code and 2048 bytes of data and Mode 2 specifies 2336 bytes of user data with no error correcting code.

DATA : This field stores user data.

LECC : This field is used to store error correcting code in Mode 1. In Mode 2 this field stores additional user data.

Advantages of CD-ROM

- Provides high capacity read-only storage memory.
- The CD-ROM together with information stored on it can be mass replicated quite economically.
- It is highly reliable and efficient information storage system.
- CD-ROM is removable, light in weight and can easily be carried from one computer to another.

Disadvantages of CD-ROM

- It is read-only memory and cannot be updated.
- Its access time is much higher than the magnetic disk drive.
- It needs careful handling. Because dust, finger prints or crashes on the reading surface may affect the reproduction of data.

7.10.2 WORM

In applications where one or a small number of copies of a set of data is needed, the write-once read-many (WORM) CD is used. The WORM storage provides one-time writing but unlimited reading of data. Data cannot be overwritten or erased but can be updated by writing new information into a file at another location on the disk. The new file is then linked to original file through software.

The WORM disk can be subsequently written once with a laser beam of modest intensity. This requires somewhat more expensive disk controller than for CD-ROM.

The WORM uses constant angular velocity (CAV) method to provide more rapid access scarifying its storage capacity. During preparation of WORM disk a high-power laser is used to produce a series of blisters on the disk. The is then placed in a WORM drive to burst the prerecorded blisters with a low powered laser. During a read operation, a laser in the WORM drive illuminates the disk's surface. Since the brust blisters provide higher contrast than the surrounding area, there are easily recognized by simple circuitry

WORM optical disks are used to handle data that must be occasionally updated and changed. It provides a permanent storage of large volumes of user data.

WORM optical disks are used to handle data that must be occasionally updated and changed. It provides a permanent storage of large volumes of user data.

7.10.3 Erasable Optical Disk

Erasable optical storage media is an emerging area where improvements are continually being made. Erasable optical media are used in applications where stored data require frequent changes. Here, magneto-optic technology is used to produce read/write memory. In this technique the energy of a laser beam of a laser beam is used together with a magnetic field to record and erase information. In magneto-optic technology the write laser heats material in the presence of a magnetic bias field. The heated bit position (a spot on the medium) take on the magnetic polarization of the field and retain it after they cool. As this polarization process does not cause a physical change in the disk, the process can be repeated many times. To erase bits, the bias field is reversed, and all bits are heated by the laser. To read data, the direction of magnetism can be detected by polarized laser light. The polarization of the light striking the written bit positions will be opposite that striking the rest of the media.

Advantages of Erasable Optical Disk

- Erasable optical disk can be erased and rewritten and thus can be used as a true secondary storage.
- Provides high storage capacity about 650 Mbytes of data on 5.25 inch disk.
- It is portable and can easily be carried from one computer to another.
- It is highly reliable and has longer life.

Disadvantages of Erasable Optical Disk

- It uses constant angular velocity method, therefore lot of storage space is wasted in the outer tracks.
- Overwriting data on magneto-optic media is slower than for magnetic media, since one revolution, is required to erase a bit and second is required to write back to that location.

7.11 Summary

If storage locations can be accessed in any order and access time is independent of the location being accessed, the access method is known as random access.

A cache memory system includes a small amount of fast memory (SRAM) and a large amount of slow memory (DRAM).

The cache design elements include cache size, mapping function, replacement algorithm write policy, block size and number of caches.

There are four most common replacement algorithms:Least-Recently Used (LRU),First-in-First-out (FIFO),Least-Frequently-Used (LFU),Random.

There are two main mapping techniques which decide the cache organization: Direct-mapping technique, Associative-mapping technique.

The virtual memory technique is used to extend the apparent size of the physical memory. It uses secondary storage such as disks, to extend the apparent size of the physical memory. It uses secondary storage such as disks, to extend the apparent size of the physical memory.

A magnetic disk is a thin, circular metal plate. It is coated with a thin magnetic film, usually on both sides.

Floppy disks are smaller, simpler and cheaper disk units that consist of a flexible, removable, plastic diskettes coated with magnetic material. The diskette is enclosed in a plastic jacket, which has an opening where the head makes contact with the diskette. A hole in the diskette allows a spindle mechanism in the disk drive to position and rotate the diskette

RAID system, a single large file is stored in several separate disk units by breaking the file up into a number of smaller pieces and storing these pieces on different disks. This is called data stripping. When the file is accessed for a read, all disk deliver their data in parallel. The total file transfer time can be given as, the transfer time that would be required in a single disk system divided by the number of disks used in the array. However, the time required to locate the beginning of the data on each disk, access time, is not reduced.

Sample Questions:

1. Give the characteristics of magnetic disk?
2. List the specifications for magnetic disks.
3. Write short note on floppy disks
4. Write short note on magnetic tapes.
5. Write short note on RAID.

6. Explain various RAID levels.
7. What is cache memory? Why it is implemented?
8. Draw and explain fully associative cache organization.
9. Draw and explain direct mapped cache organization.
10. Draw and explain set associative cache organization.
11. What is virtual memory?

Reference Books

1. Computer Organization & Architecture, William Stallings, 8e, Pearson Education.
2. Computer Architecture& Organization, John P. Hayes, 3e, Tata McGraw Hill.
3. Computer Organization, 5e, Carl Hamacher, ZconkoVranesic&SafwatZaky, Tata McGraw-Hill.
4. Computer Architecture & Organization, Nicholas Carter, McGraw Hill
5. Computer System Architecture, M.MorrisMano, Pearson Education.

❖❖❖

8

PROCESSOR ORGANIZATION

Unit Structure

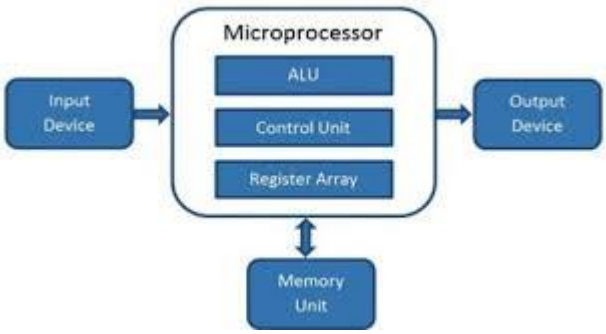
8.1 8085 Microprocessor

8.2 Registers in 8085

8.3 Restart Interrupts (Input)

8.1 8085 Microprocessor

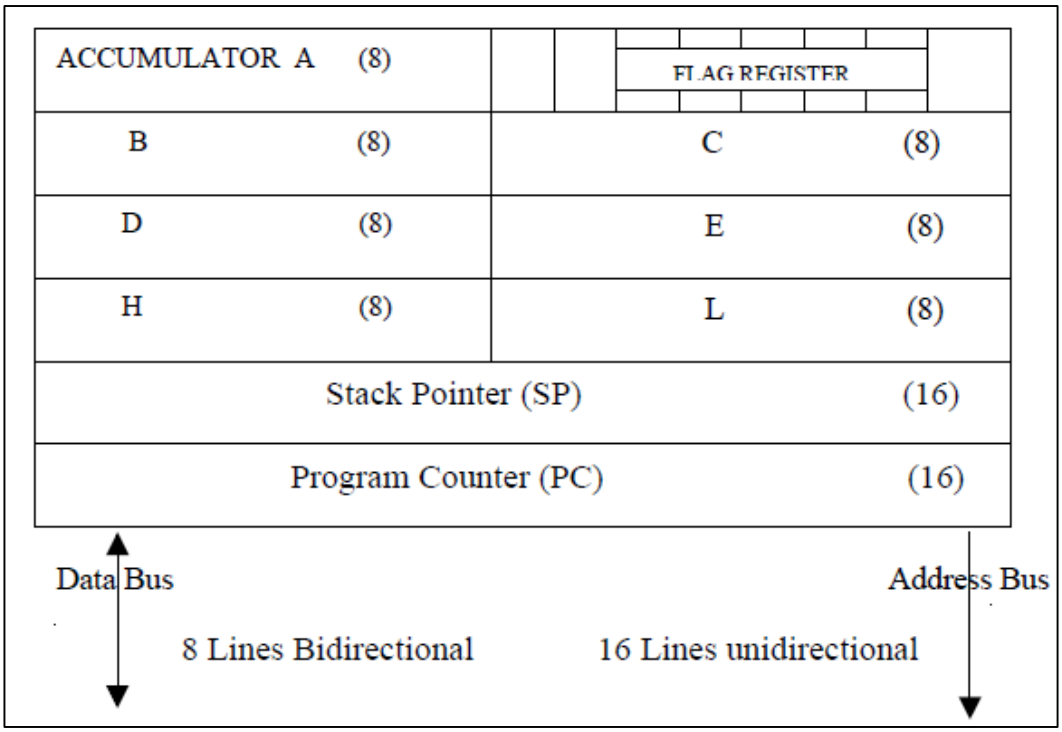
8085 is an 8-bit microprocessor produced by Intel and introduced in March 1976. It is a programmable logic device that reads binary instruction from storage device called memory.It accepts binary data as input, then processes the data according to the instructions and provides result as output.



Features of 8085 microprocessor

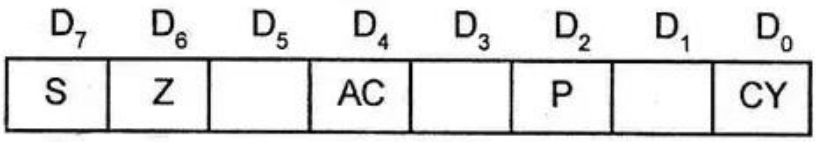
- It is 8-bit microprocessor.
- It has 16-bit address bus ($A_0 – A_{15}$), hence can address up to 2^{16} = 64 KB of memory.
- It has 8-bit data bus multiplexed with lower order address bus lines ($AD_0 – AD_7$)
- It has 16-bit Program Counter (PC)
- It has 16-bit Stack Pointer (SP)
- Six 8-bit general purpose registers (B,C,D,E,H,L) along with an Accumulator and a Status (flag) register.
- It requires a single +5V power supply
- It operates at clock frequency of 3.2 MHz.

8.2 Registers in 8085



Accumulator : The accumulator is an 8-bit register. This register is used to store 8-bit data and to perform arithmetic and logical operations. The result of an operation is stored in the accumulator. The accumulator is also identified as register A.

Flags : There are five flags in 8085. Depending upon the value of result after any arithmetic and logical operation, the flag bits become set (1) or reset (0).The microprocessor uses these flags to test data conditions.



- Sign (S) : Sign flag is set (=1) when the result of operation is negative.
- Zero(Z) : Zero flag is set (=1) when the result of operation is zero (0).
- Auxiliary Carry (AC) : Auxiliary Carry flag is set (=1) when a carry is generated by digit D_3 and passed on to digit D_4.
- Parity (P) : Parity flag is set (=1) when result contains even number of 1's.
- Carry (CY) : Carry flag is set (=1) when a carry is generated by an operation.

General Purpose Registers : The 8085 has six general-purpose registers to store 8-bit data; B,C,D,E,H, and L. They can be combined as register pairs - BC, DE, and HL - to perform some 16-bit operations..

Program Counter (PC) : This 16-bit register deals with sequencing the execution of instructions. This register is a memory pointer. Memory locations have 16-bit addresses, and that is why this is a 16-bit register. The microprocessor uses this register to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.

Stack Pointer (SP): The stack pointer is also a 16-bit register used as a memory pointer. It points to a memory location in R/W memory, called the stack. The beginning of the stack is defined by loading 16-bit address in the stack pointer.

8085 System Bus : Typical system uses a number of busses, collection of wires, which transmit binary numbers, one bit per wire. A typical microprocessor communicates with memory and other devices (input and output) using three busses: Address Bus, Data Bus and Control Bus.

Address Bus : One wire for each bit, therefore 16 bits = 16 wires. Binary number carried alerts memory to ‘open’ the designated box. Data (binary) can then be put in or taken out. The Address Bus consists of 16 wires, therefore 16 bits. Its "width" is 16 bits.

A 16 bit binary number allows 2^{16} different numbers, or 65535 different numbers, i.e. 0000000000000000 up to 1111111111111111.

Data Bus : Data Bus: carries ‘data’, in binary form, between μP and other external units, such as memory. Typical size is 8 or 16 bits. The Data Bus typically consists of 8 wires. Therefore, 2^8 combinations of binary digits. Data bus used to transmit "data", i.e. information, results of arithmetic, etc, between memory and the microprocessor. Data bus is bi-directional. Size of the data bus determines what arithmetic can be done.

Control Bus : Control Bus are various lines which have specific functions for coordinating and controlling μP operations. E.g. : Read/Write line, single binary digit. Control whether memory is being ‘written to’ (data stored in memory) or ‘read from’ (data taken out of memory) Read = 0, Write = 1. May also include clock line(s) for timing/ synchronising, ‘interrupts’, ‘reset’ etc.

Pin Diagram:

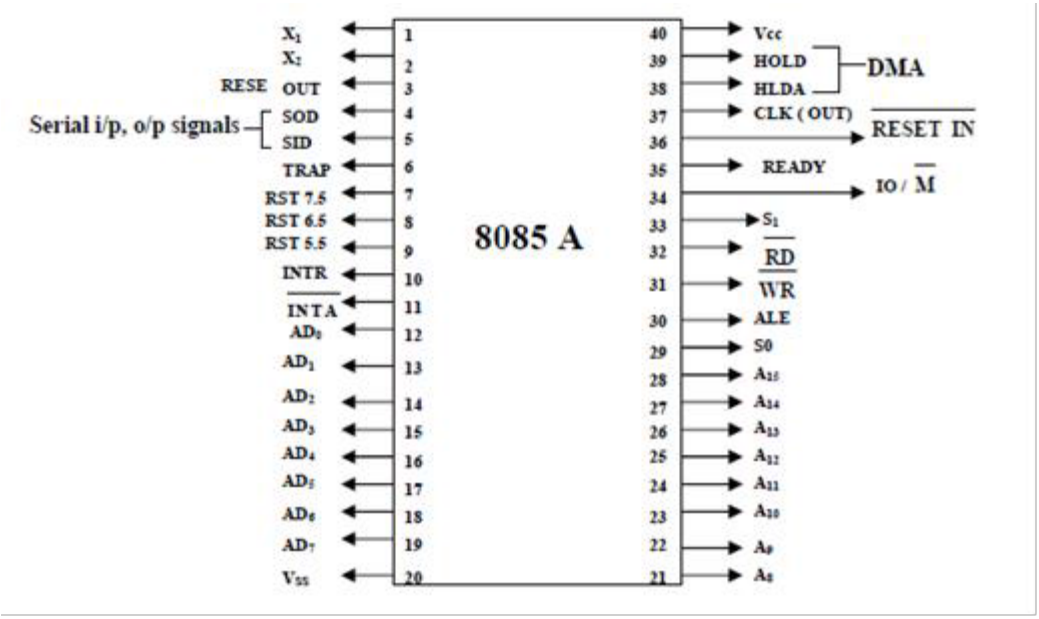
Pin Description : The following describes the function of each pin:

A8 - A16 (Output 3 State) : Address Bus; The most significant 8 bits of the memory address or the 8 bits of the I/O address,3 stated during Hold and Halt modes.

AD0 - 7 (Input/Output 3state) : Multiplexed Address/Data Bus; Lower 8 bits of the memory address (or I/O address) appear on the bus during the first clock cycle of a machine state. It then becomes the data bus during the second and third clock cycles. 3 stated during Hold and Halt modes.

ALE (Output) : This is a positive going pulse generated every time 8085 begins an operation, it indicates that the bits AD_7–AD_0 are address bits. This signal is used to latch the low-order address from multiplexed bus and generate separate set of 8 address line A_7–A_0.

SO, S1 (Output) : Data Bus Status. Encoded status of the bus cycle :



Machine Cycle	Status			Control Signals
	IO / $\overline{M}$	S1	S0	
Opcode Fetch	0	1	1	$\overline{RD} = 0$
Memory Read	0	1	0	$\overline{RD} = 0$
Memory Write	0	0	1	$\overline{WR} = 0$
I/O Read	1	1	0	$\overline{RD} = 0$

Machine Cycle	Status			Control Signals
	$IO/\overline{M}$	S1	S0	
I/O Write	1	0	1	$\overline{WR} = 0$
Interrupt Acknowledge	1	1	1	$\overline{INTA} = 0$
Halt	Z	0	0	
Hold	Z	Z	X	$\overline{RD}, \overline{WR} = Z$ and $\overline{INTA} = 1$
Reset	Z	X	X	

Note : Z = Tri-state (high impedance) X = Unspecified

RD (Output 3state) :READ; indicates the selected memory or 1/0 device is to be read and that the Data Bus is available for the data transfer.

WR (Output 3state) :WRITE; indicates the data on the Data Bus is to be written into the selected memory or 1/0 location. Data is set up at the trailing edge of WR. 3stated during Hold and Halt modes.

READY (Input) : If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If Ready is low, the CPU will wait for Ready to go high before completing the read or write cycle.

HOLD (Input) :HOLD; indicates that another Master is requesting the use of the Address and Data Buses. The CPU, upon receiving the Hold request. will relinquish the use of buses as soon as the completion of the current machine cycle. Internal processing can continue. The processor can regain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3stated.

HLDA (Output) :HOLD ACKNOWLEDGE; indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The CPU takes the buses one half clock cycle after HLDA goes low.

INTR (Input) :INTERRUPT REQUEST; is used as a general purpose interrupt. It is sampled only during the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a RESTART or CALL instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

INTA (Output) :INTERRUPT ACKNOWLEDGE; is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted. It can be used to activate the 8259 Interrupt chip or some other interrupt port.

8.3 Restart Interrupts (Input)

These three inputs have the same timing as INTR except they cause an internal RESTART to be automatically inserted.

RST 7.5 ~~ Highest Priority

RST 6.5

RST 5.5 Lowest Priority

The priority of these interrupts is ordered as shown above. These interrupts have a higher priority than the INTR.

RST5.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 2CH (hexadecimal) address.

RST6.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 34H (hexadecimal) address.

RST7.5 is a maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 3CH (hexadecimal) address.

TRAP is a non-maskable interrupt. When this interrupt is received the processor saves the contents of the PC register into stack and branches to 24H (hexadecimal) address.

All maskable interrupts can be enabled or disabled using EI and DI instructions. RST 5.5, RST6.5 and RST7.5 interrupts can be enabled or disabled individually using SIM instruction.

RESET IN (Input) : Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flipflops. None of the other flags or registers (except the instruction register) are affected The CPU is held in the reset condition as long as Reset is applied.

RESET OUT (Output) : Indicates CPU is being reset. Can be used as a system RESET. The signal is synchronized to the processor clock.

X1, X2 (Input) : Crystal or RC network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to give the internal operating frequency.

CLK (Output) : Clock Output for use as a system clock when a crystal or R/ C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

IO/M (Output) :IO/M indicates whether the Read/Write is to memory or l/O Tristated during Hold and Halt modes.

SID (Input) : Serial input data line The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

SOD (output) : Serial output data line. The output SOD is set or reset as specified by the SIM instruction.

Vcc
+5 volt supply.

Vss
Ground reference

❖❖❖

9

INSTRUCTION FORMAT

Unit Structure

9.1 Instructions

9.2 Addressing Modes of 8085

9.3 Instruction Set

9.1 Instructions

An instruction is a command to the microprocessor to perform certain task on a given data. The programming using these instructions of the microprocessor is called as “assembly language programming”. Each instruction consists of opcode(Operation Code)and operand. **Opcode** specifies the operation to be performed and **operand** can be any register or data.

For example :

ADI 45 H

The opcode is ADI instruction and operands is the value 45 hex.

Instruction Format

The 8085 instruction set is classified into the following three groups according to word size:

- One-word or 1-byte instructions
- Two-word or 2-byte instructions
- Three-word or 3-byte instructions

1. **One–Byte Instructions :**A 1–byte instruction includes the opcode and the operand in the same byte. For example

Task	**Opcode**	**Operand**	**Binary Code**	**Hex Code**
Add the contents of register C to the contents of the accumulator.	ADD	C	1000 0001	81H

2. **Two–Byte Instructions :** In a 2–byte instruction, the first byte specifies the operation code and the second byte specifies the operand. For example

Task	Opcode	Operand	Binary Code	Hex Code	Bytes
The 8–bit data (operand) are added to the contents of the accumulator	ADI	8 bit Data	11000110 8 bit Data	C6 H 8 bit Data	First Byte Second Byte

3. **Three–Byte Instructions :**In a 3–byte instruction, the first byte specifies the opcode, and the following two bytes specify the 16–bit address. Note that the second byte is the low–order address and the third byte is the high–order address. For example :

Task	Opcode	Operand	Binary Code	Hex Code	Bytes
Transfer the program sequence to the memory location 2050H.	JMP	2050H	1100 0011 0101 0000 0010 0000	C3 50 20	First Byte Second Byte Third Byte

9.2 Addressing Modes of 8085

We have seen that every instruction contains op-code and operand. The method of specifying operand determines addressing mode of the instruction. The various formats for specifying operands are called the addressing modes. For 8085, they are:

i. Register addressing mode

ii. Immediate addressing mode

iii. Direct addressing mode

iv. Indirect addressing mode

v. Implied or implicit addressing mode

i) Register addressing Mode :

Operand is register i.e. data is provided through the registers.

Format: MOV Rd, Rs

Here Rd is destination register and Rs is a source register. Source and destination both are registers.

Example:

MOV C,A – Copy content of register A to register C

DCRC -Decrement content of register C by 1. Here only one operand C which is register

ii) Immediate addressing Mode:

Data is specified within the instruction only. The instruction loads the immediate data to the destination provided.

Example: MVI Rd, 8 bit data-

Move 8-bit data specified in the instruction to thedestination register Rd.

LXI D, 16 bit data- Move 16 bit data specified within instruction to register pair DE

iii) Direct addressing :

8 bit or 16 bit direct address of the operand is specified within the instruction.

Example: LDA 16 bit address-

Move data stored in specified address to the destination register A.

LDA 7400H – Move data stored in 7400 to accumulator

IN 90H – Read data from port whose address is 90 and load it in A

iv) Indirect Addressing :

In this addressing mode,the address of operand is indirectly specified in the register pair or memory location M.

Example: MOV C, M-

Move data stored in memory location whose address is specified in HL pair to register C.

STAX D – Store the data into A from the memory location whose address is specified in register pair DE.

iv) Implied or implicit Addressing :

The operand is not specified within the instruction. Mostly operand is accumulator.

Example: DAA – Decimal adjust accumulator

RAR– Rotate content of accumulator to right

9.3 Instruction Set

Data Transfer Instructions:

Instruction	Explanation	Example
MOV Rd, Rs	Move the content of the one register to another	MOV C, A
MOV Rd, M	Move the content of memory to register	MOV C, M
MOV M, Rs	Move the content of register to memory	MOV M, B
MVI Rd, data	Move immediate data to register	MVI M, 10H
LXI Rp, data 16	Load Register pair immediate	LXI H, 2100H
LDA addr	Load Accumulator direct	LDA 3100H
STA Addr	Store accumulator direct	STA 4100H
LHLD addr	Load H-L pair direct	LHLD 5100H
SHLD addr	Store H-L pair direct	SHLD 6100H
LDAX Rp	Load accumulator indirect	LDAX B
STAX Rp	Store accumulator indirect	STAX D
XCHG	Change the contents of H-L with D-E pair	XCHG

Arithmetic Instructions

Instruction	Explanation	Example
ADD R	Add register to accumulator	ADD B
ADD M	Add memory to accumulator	ADD M
ADC R	Add register with carry to accumulator	ADC B
ADC M	Add memory with carry to accumulator	ADC M
ADI data	Add immediate data to accumulator	ADI 15H
ACI data	Add with carry immediate data to accumulator	ACI 50H
DAD Rp	Add register pair to H-L pair	DAD B
SUB R	Subtract register from accumulator	SUB B
SUB M	Subtract memory from accumulator	SUB M
SBB R	Subtract memory from accumulator with borrow	SBB B
SUI data	Subtract immediate data from accumulator	SUI 60H
SBI data	Subtract immediate data from accumulator with borrow	SBI 50H
INR R	Increment register content by 1	INR C
INR M	Increment memory content by 1	INR M
DCR R	Decrement register content by 1	DCR D
DCR M	Decrement memory content by 1	DCR M
INX Rp	Increment register pair by 1	INX H
DCX Rp	Decrement register pair by 1	DCX H
DAA	Decimal adjust accumulator	DAA

Logical Instructions:

Instruction Set	Explanation	Example
ANA R	AND register with accumulator	ANA B
ANA M	AND memory with accumulator	ANA M
ANI data	AND immediate data with accumulator	ANI 55H
ORA R	OR-register with accumulator	ORA C
ORA M	OR-memory with accumulator	ORA M
ORI data	OR -immediate data with accumulator	ORI 45H
XRA R	XOR register with accumulator	XRA B
XRA M	XOR memory with accumulator	XRA M
XRI data	XOR immediate data with accumulator	XRI 35H
CMA	Complement the accumulator	CMA
CMC	Complement the carry status	CMC
STC	Set carry status	STC
CMP R	Compare register with accumulator	CMP C
CMP M	Compare memory with accumulator	CMP M
CPI data	Compare immediate data with accumulator	CPI 25H
RLC	Rotate accumulator left	RLC
RRC	Rotate accumulator right	RRC
RAL	Rotate accumulator left through carry	RAL
RAR	Rotate accumulator right through carry	RAR

Branch Instruction :

Instruction	Explanation	Example
JMP address	Unconditional jump: jump to the instruction specified by the address	JMP 2050

Conditional Jump

Instruction	Explanation	Condition	Example
JZ address	Jump, if the result is zero	Jump if Z=1	JZ 2060H
JNZ address	Jump if the result is not zero	Jump if Z=0	JNZ 2070H
JC address	Jump if there is a carry	Jump if CS =1	JC 3050H
JNC address	Jump if there is no carry	Jump if CS =0	JNC 3060H
JP address	Jump if result is plus	Jump if S=0	JP 4050H
JM address	Jump if result is minus	Jump if S=1	JM 4060
JPE address	Jump if even parity	The parity status P =1	JPE 5050H
JPO address	Jump if odd parity	The parity status P =0	JPO 5060

Conditional CALL

Instruction	Explanation	Example
CALL Address	Unconditional CALL: Call the subroutine identified by the address if the specified condition is fulfilled	CALL 5050

Unconditional Return

Instruction	Explanation	Flag	Example
CC address	Call subroutine if carry status CS=1	C=1	CC
CNC address	Call subroutine if carry status CS=0	C=0	
CZ address	Call Subroutine if the result is zero	Z=1	
CNZ address	Call Subroutine if the result is not zero	Z=0	
CP address	Call Subroutine if the result is plus	S=0	
CM address	Call Subroutine if the result is minus	S= 1	
CPE address	Call subroutine if even parity	P=1	
CPO address	Call subroutine if odd parity	P= 0	

Instruction	Explanation
RET	Unconditional RET: Return from subroutine

Conditional Return

Instruction Set	Explanation
IN port - address	Input to accumulator from I/O port
OUT port-address	Output from accumulator to I/O port
PUSH Rp	Push the content of register pair to stack
POP Rp	Pop the content of register pair, which was saved, from the stack
PUSH PSW	Push processor word
HLT	Halt

Instruction Set	Explanation
XTHL	Exchange top stack with H-L
SPHL	Moves the contents of H-L pair to stack pointer
EI	Enable Interrupts
SIM	Set Interrupts Masks
RIM	Read Interrupts Masks
NOP	No Operation

Assembly language Programs:

1. Subtract two 8-bit numbers. 33H – 22H

Address	Label	Mnemonics	Hexcode	Bytes
0000		MVI A,33	3E	2
0001			33	
0002		SUI 22	D6	2
0003			22	
0004		HLT	76	1

2. **Subtract the contents of memory location 4001H from the memory location 2000H and place the result in memory location 4002H.**

Address	Label	Mnemonics	Hexcode	Bytes
0000		LXI H,4001	21	3
0001			01	
0002			40	
0003		LDA 2000	3A	3
0004			00	
0005			20	
0006		SUB M	96	1
0007		INX H	23	1
0008		MOV M,A	77	1
0009		HLT	76	1

3.**Add the contents of memory locations 4000H and 4001H and place the result in the memory locations 4002Hand 4003H.**

Address	Label	Mnemonics	Hexcode	Bytes
0000		MVI D,00	16	2
0001			00	
0002		LXI H,4000	21	3
0003			00	
0004			40	
0005		MOV A,M	7E	1
0006		INX H	23	1
0007		ADD M	86	1
0008		JNC SKIP	D2	3
0009			0C	
000A			00	
000B		INR D	14	1
000C	SKIP	INX H	23	1
000D		MOV M,A	77	1
000E		INX H	23	1
000F		MOV M,D	72	1
0010		HLT	76	1

4. **Find the 1's complement of the number stored at memory location 4400H and store the complemented number at memory location 4300H.**

Address	Label	Mnemonics	Hexcode
0000		LDA 4400	3A
0001			00
0002			44
0003		CMA	2F
0004		STA 4300	32
0005			00
0006			43
0007		HLT	76

5. **Write a program to shift an eight bit data four bits right. Assume that data is in register C.**

Address	Label	Mnemonics	Hexcode	Bytes
0000		MOV A,C	79	1
0001		RRC	0F	1
0002		RRC	0F	1
0003		RRC	0F	1
0004		RRC	0F	1
0005		MOV C,A	4F	1
0006		HLT	76	1

❖❖❖

10

PROCESSOR STRUCTURE AND FUNCTION

Unit Structure

10.0 Objective

After studying this chapter, you should be able to:

Distinguish between **user-visible**and **control/status registers**, and discuss thepurposes of registers in each category.

Summarize the **instruction cycle**.

Discuss the principle behind **instruction pipelining** and how it works inpractice.

10.1 Processor Organization

Fetch instruction: The processor reads an instruction from memory (register,cache, main memory).

Interpret instruction: The instruction is decoded to determine what action isrequired.

Fetch data: The execution of an instruction may require reading data frommemory or an I/O module.

Process data: The execution of an instruction may require performing somearithmetic or logical operation on data.

Write data: The results of an execution may require writing data to memoryor an I/O module.

To do these things, it should be clear that the processor needs to store some data temporarily. It must remember the location of the last instruction so that it canknow where to get the next instruction. It needs to store instructions and data temporarilywhile an instruction is being executed. In other words, the processor needsa small internal memory.

Figure 14.1 is a simplified view of a processor, indicating its connection to the

rest of the system via the system bus.

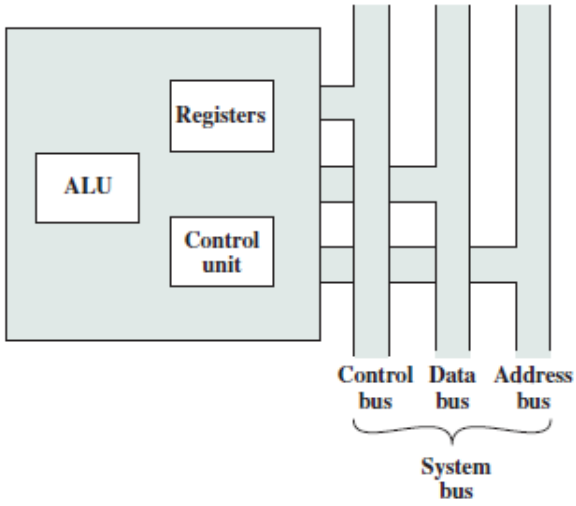


Figure 14.1 The CPU with the System Bus

The major components of a processor are an Arithmetic Logic Unit (ALU) and a Control Unit(CU). The data processing and all computations are done by ALU; whereas CU controls the movement of data, instructions (into and out of the processor) and also controls the operation of ALU. Minimum internal memory is available in the form of a set of storage locations called registers.

Figure 14.2 is a more detailed view of the processor. The data control and logic control paths are shown along with internal CPU bus. This bus is required since the ALU operates only on the data in internal memory. The figure also shows typical elements contained in an ALU.

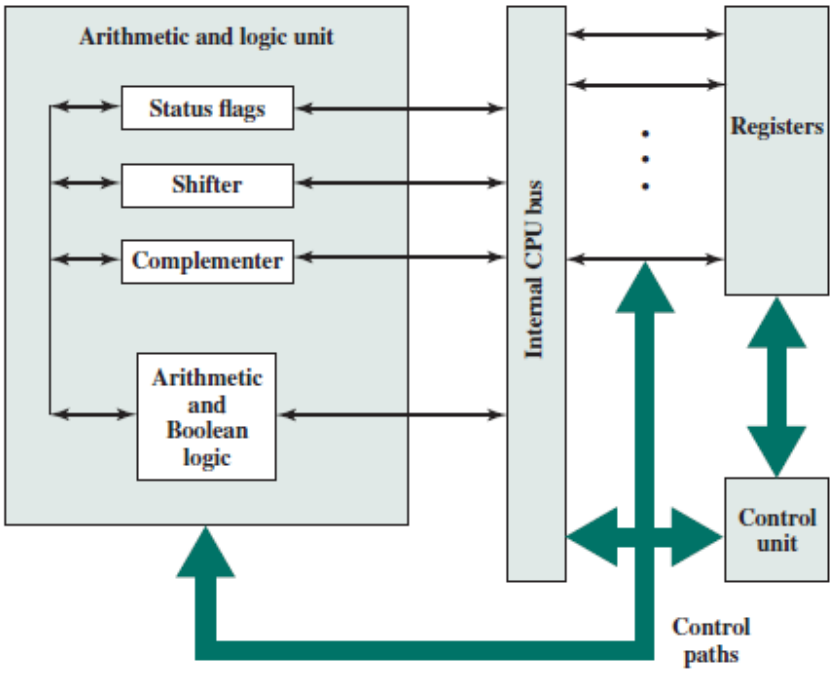


Figure 14.2 Internal Structure of the CPU

10.3 Registers Structure

A computer system follows a memory hierarchy. At higher levels, the memory is more fast, small and more expensive in terms of cost per bit. Within the processor, we have a set of registers which function at a higher level than main memory and cache, in terms of hierarchy. The registers within the processor perform following two main functions:

- **User-visible registers**: These registers can be accessed and optimized for memory usage by the machine or assembly language programmer.
- **Control/Status registers**: These registers are used by the control unit for controlling the operation of the processor.

There is no clear separation between these two types of registers. For example, in some machines, the program counter is user-visible whereas it is not so in some machines.

User-visible registers: User visible registers can be broadly classified as:

- General purpose
- Data
- Address
- Condition codes

We discuss each of them in brief.

General purpose registers: The programmer can assign these registers various functions. Any general purpose register can contain the operand for any opcode. In some cases, these general purpose registers can be used for addressing function. In other cases, there exists a clear separation between data and address registers. **Data registers** can hold only data and cannot be used for address purpose.Registers which are used as **addressregisters** must be able to hold the largest address. Similarly, data registers must be able to hold values of most data types.

Condition codes(or flags) are registers which are partially visible to the user. These condition codes are bits set by the processor hardware as a result of some operation. Condition code bits are collected into one or more registers; and they usually form part of a control register. Generally, they can be read but cannot be altered by the user/programmer.

Control and Status registers:

Most of these registers are not visible to the users. Four registers are necessary for executing an instruction. They are:

1. **Program Counter(PC):**This contains the address of an instruction to be fetched.
2. **Instruction Register(IR):** This contains the instruction most recently fetched.
3. **Memory Address Register(MAR):** It contains the address of a location in the memory.
4. **Memory Buffer Register(MBR):** It contains a word of data to be written to memory or the word most recently read.

These four registers are used for the movement of data between the processor and memory. Within the processor, data must be presented to ALU for processing. ALU generally has direct access to MBR and other user-visible registers.

Many processor designs include a register or a set of registers, known as the **program status word (PSW),** which contains the status information. Generally, a PSW consists the following flags:

- **Sign:**It contains the sign bit of the result of the last arithmetic instruction.
- **Zero:** It is Set when the result of an operation is zero.
- **Carry:** It is Set if result of an operation has generated a carry during addition, or a borrow out of the higher bit during subtraction results.

- **Equal:** It is Set if the result of a logical comparison is equality**.**
- **Overflow:** It indicates arithmetic overflow, generally into the sign bit.
- **Interrupt Enable/Disable:** It is used to enable or disable interrupts.
- **Supervisor:** It indicates whether the processor is executing in supervisor or user mode.

Every register in 8086 is special-purpose, however some registers can also be used as general-purpose. The 8086 contains four 16-bit data registers which are addressable on a byte or 16-bit. It also contains four 16-bit pointer and index registers. There are also four 16-bit segment registers.

10.4 Instruction Cycle

An instruction cycle consists of following stages**:**

Fetch: Fetching means reading the next instruction from memory into the processor**.**

Execute: Execution means interpreting the opcode and performing the indicated operation.

Interrupt: If interrupts are enabled and an interrupt has occurred, then the processor saves the current process state and services(attends) the interrupt.

The execution of an instruction may involve one or more operands in memory. Each operand will require memory access. Also, if indirect addressing is used, then additional memory accesses are required. Thus, fetching of indirect addresses can be viewed as one more instruction stage as shown in Figure 14.4

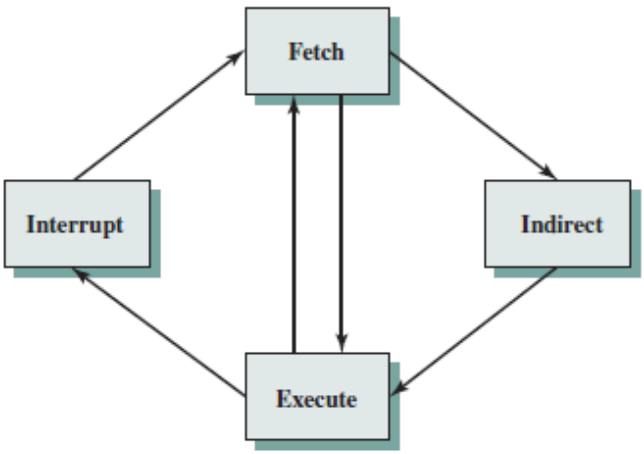


Figure 14.4 The Instruction Cycle

To understand the data flow, let us assume that the processor uses a memory address register(MAR), a memory buffer register (MBR), a program counter (PC); and an instruction register (IR).

During the fetch cycle, an instruction is read from the memory. Figure 14.6 shows the flow of data during this cycle. The PC contains the address of the next instruction to be fetched. This address is moved to the MAR and placed on the address bus.

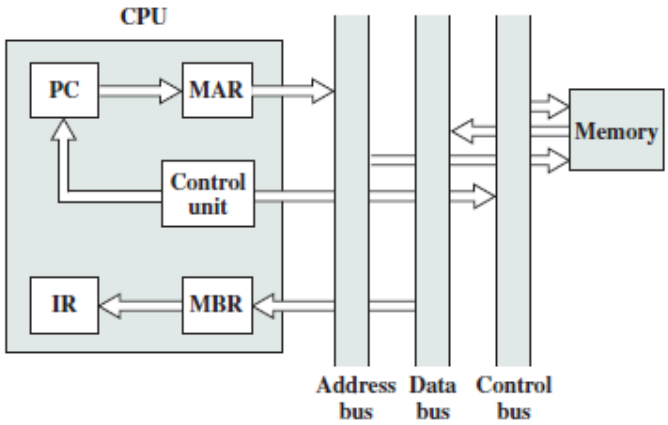


Figure 14.6 Data Flow, Fetch Cycle

The control unit requests a memory read, and the result is placed on the data bus and copied into the MBR and then moved to IR. Meanwhile, the PC is incremented by 1, preparing for the next fetch.

Once the fetch cycle is over, the control unit examines the contents of the IR to determine if it contains an operand specifier using indirect addressing. If it is so, then an indirect cycle is performed as shown in Figure 14.7

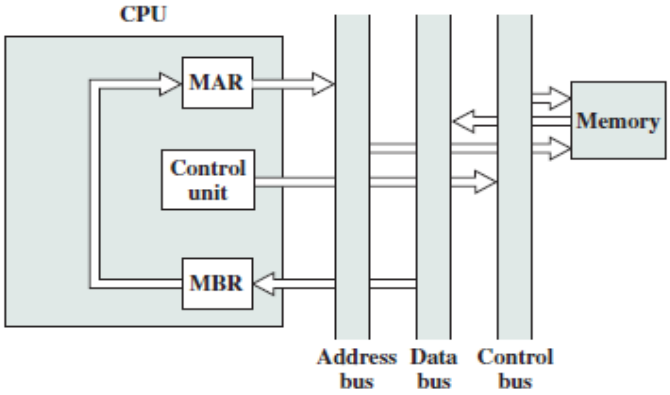


Figure 14.7 Data Flow, Indirect Cycle

The right-most N bits of the MBR are transferred to the MAR. Then the control unit requests a memory read, to get the desired address of the operand into the MBR.

The fetch and indirect cycles are simple and predictable. The execute cycle has many forms and it depends on which instructions are present in the IR. This execute cycle may involve transferring data among the registers, read or write from memory or I/O; and/or calling the ALU.

Just like the fetch and indirect cycles, the interrupt cycle is predictable as shown in Figure 14.8. The current contents of the PC must be saved so that the processor can resume normal activity after the interrupt. Thus, the contents of the PC are transferred to the MBR to be written into memory. The special memory location reserved for this purpose is loaded into the MAR from the control unit. It may be a stack pointer, for example. The PC is loaded with the address of the interrupt routine. As a result, the next instruction cycle will begin by fetching the appropriate instruction.

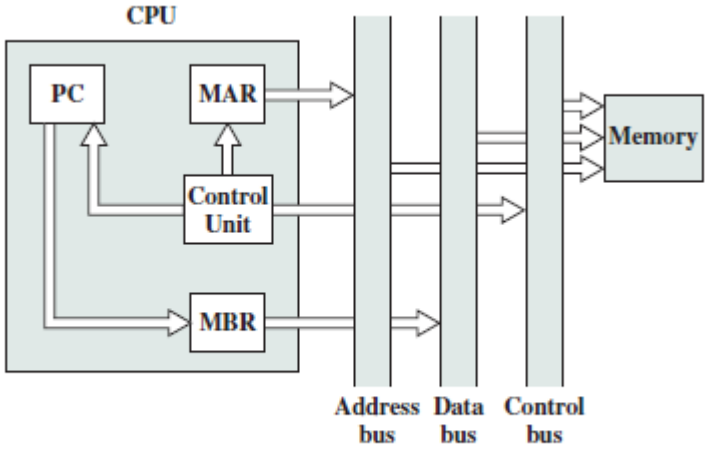


Figure 14.8 Data Flow, Interrupt Cycle

10.5 Instruction Pipelining

Instruction Pipelining is similar to the use of an assembly line in a manufacturing plant**.**

In an assembly line, the product goes through various stages of production. The advantage of such a system is that products at various stages can be worked on simultaneously. This process is known as **pipelining**since just like a pipeline, new inputs are accepted at one end before previously accepted inputs appear as outputs at the other end.

We can consider the instruction processing consisting of two stages: fetch instruction and execute instruction. There are times during the execution of an instruction when main memory is not being accessed. This time could be used to fetch the next instruction in parallel with the execution of the current one. Figure 14.9(a) depicts this where the pipeline has two independent stages. The first stage fetches an instruction and buffers it. When the second stage is free, the first stage passes the buffered instruction. While the second stage is executing the instruction, the first stage fetches and buffers the next instruction. This is called as **instruction prefetch** or **fetch overlap**.

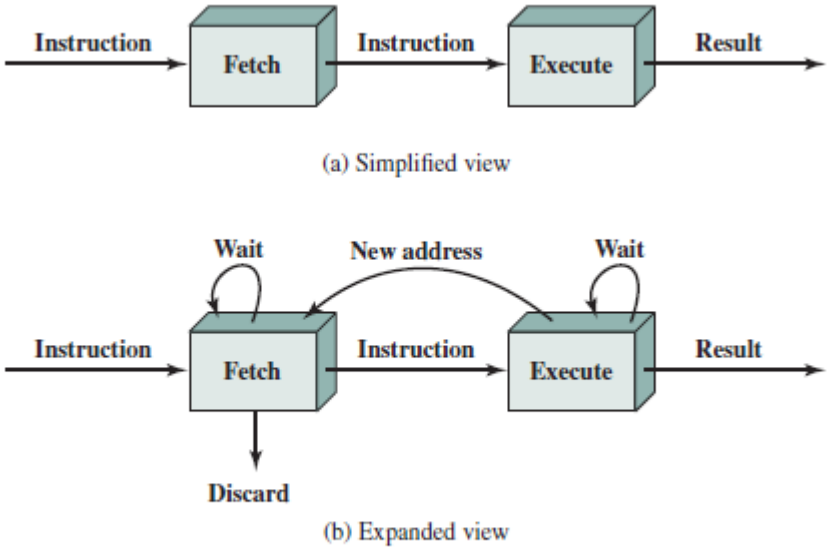


Figure 14.9 Two-Stage Instruction Pipeline

In general, pipelining requires more registers to store data between the stages. Figure 14.9(b) shows an expanded view of this concept. Obviously, to speed up the instruction cycle processing, the pipeline should have more stages.

❖❖❖

11

INTRODUCTION TO RISC AND CISC ARCHITECTURE

Unit Structure

11.0 Objectives

After reading this unit,student will be able to

- Classify the different types of architectures.
- Compare and contrast between the RISC and CISC architectures
- Outline the concept of Instruction Level Parallelism
- Summarize the concept of Super scalar processors:Design Issues

11.1 Introduction

1. The dominant architecture in the PC market, was the Intel IA-32, belongs to the complex instruction set(CISC) design.
2. The CISC instruction set architecture is too complex in nature and developers develop it very complex so to use with higher level languages which supports complex data structures.
3. But the side effects of this is very serious to ignore.
4. The systems which are designed with Reduced Instruction set required, some sort of parallelism in form of instruction. So the concept of Instruction level parallelism has evolved.

5. So, ILP is a family of processor and compiler design techniques that speed up execution by causing individual machine operations such as memory loads and stores, integer additions and floating point multiplications, to execute in parallel.

6. The operations involved are normal RISC-style operations and the system is handed a single program written with a sequential processor in mind.

7. The term superscalar refers to a processor that is designed to a)improve the performance of the execution of scalar instructions.b)Represents the next evolution step.

8. The simple idea to develop a super scalar processor to increase the number of pipelines.

9. So, this unit is all about RISC-CISC architecture, Instruction level parallelism and super scalar processor with its design issues.

11.2 Introduction to Risc-Cisc Architecture

1. The CISC processor designers provide a variety of addressing modes,leads to variable-length instructions. For example, instruction length increases if an operand is in memory as opposed to in a register.

   A. This is because we have to specify the memory address as part of instruction encoding, which takes many more bits.

   B. This complicates instruction decoding and scheduling. The side effect of providing a wide range of instruction types is that the number of clocks required to execute instructions varies widely.

   C. This again leads to problems in instructions scheduling and pipelining.

2. Evolution of RISC

2.1 For variety of reasons, in the early 1980's designers started looking at simple Instruction set architectures, as these ISAs tend to produce instruction sets with less number of instructions known as Reduced Instruction Set Computer(RISC).

2.2 Even though the main goal was not to reduce the number of instructions, but the complexity, the term has stuck.

2.3 There is no precise definition to refer to a RISC processor design. But there are typical characteristics which are present in most RISC based systems.

2.4 CISC and RISC have their advantages and disadvantages, modern processor take features from both classes. For example, the PowerPC, which follows the RISC terminology has very few complex instructions also.

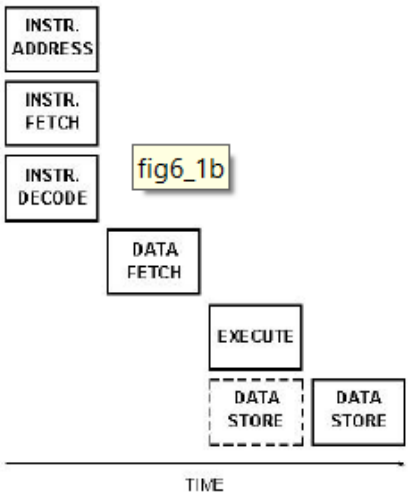


Fig 1 Typical RISC Architecture based Machine level Instruction

2.5 RISC or Reduced Instruction set computer is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions, rather than a more specialized set of instructions often found in other types of architectures.

2.5.1 Evolution- The first RISC projects was developed by IBM, standford, and UC-Berkeley in the late 70's and early 80's.

2.5.2 The IBM 801, standford , MIPS and Berkeley RISC 1 and 2 were all designed with a similar philosophy which has become known as RISC. Certain design features have been characteristics of most RISC processor.

A. One cycle execution time- RISC processors have a (Clock per instruction) of one cycle. This is due to the optimization of each instruction on the CPU and a technique called.

B. Pipelining- A technique that allows for simultaneous execution of parts or stages of instructions to more efficiently process instructions.

C. Large number of registers- The RISC design philosophy generally incorporates a large number of registers to prevent in large amounts of interactions with memory.

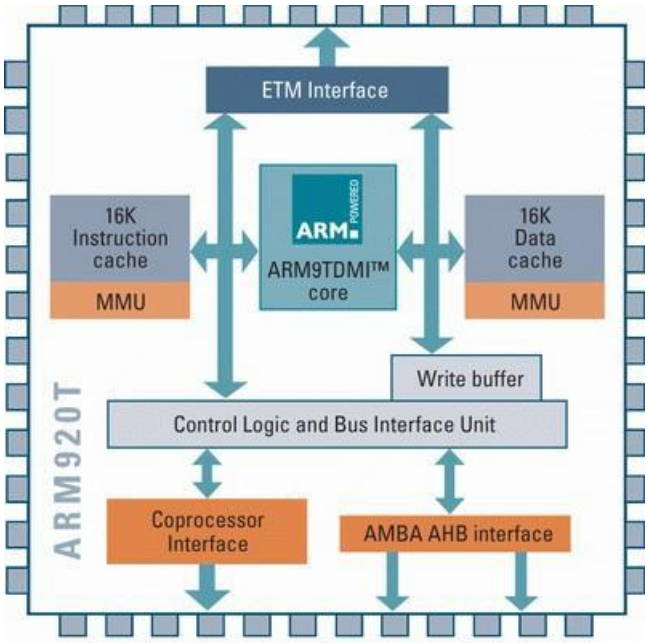


Fig 2 Advanced RISC machine ARM

3 Non RISC design or Pre RISC design/ CISC design

3.1 In the early days of the computer industry, programming was done in assembly language or machine code, which encouraged powerful and easy to use instructions.

3.2 CPU designers therefore tried to make instructions that would do as much work as possible.

3.3 With the advent of higher level languages, computer architects also started to create dedicated instructions to directly implement certain central mechanism of such languages.

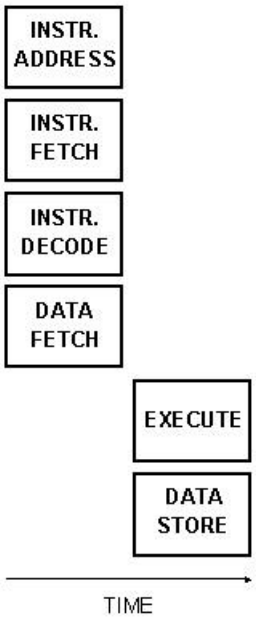


Fig 3 CISC Architecture

3.4 Another general goal was to provide every possible addressing mode for every instruction, known as orthogonality to ease compiler implementation.

3.5 So to design hardware in a better way, there is also a requirement to add functionality in hardware or in microcode rather than in a memory constrained compiler or its generated code alone. So this design is coined with term known as Complex instruction set.

3.6 Complexities is more in CISC instruction set due to limited amount of main memories which is in the order of kilobytes, so is the reason to add variable length of instruction in computer programs itself.

3.7 The problem of packaging together the instruction with computer program is that it reduces the frequency of CPU for accessing the resources.

3.8 Modern computers face similar limiting factors: main memories are slow compared to the CPU and the fast cache memories employed to overcome this are instead limited in size and because of these reasons RISC came in to existence.

4. RISC typical characteristics

4.1 RISC designers take help from CISC designing framework in order to improve the performance of hardware and software. So some of the characteristics are listed below as follows

4.2 Simple instructions- RISC instruction set uses high-level languages for launching program in main memory along with different semantics of these languages. For example, the semantics of the C for loop is not exactly the same as that in other languages. Thus, compilers tend to synthesize the code using simpler instructions.

4.3 Few Data Types- CISC ISA tends to support a variety of data structures from simple data types such as integers and characters to complex data structures are used relatively infrequently. Thus, it is beneficial to design a system that supports a few simple data types efficiently and from which the missing complex data types can be synthesized.

4.4 Simple Addressing Modes- CISC designs provide a large number of addressing modes. The main motivations are

4.4.1 To support complex data structures and

4.4.2 To provide flexibility to access operands

A) Problems caused- This also enhances flexibility, it also introduces problems. First it cause variable instruction execution times, depending on the location of the operands.

B) Second it leads to variable-length instructions which lead to inefficient instruction decoding and scheduling.

4.5 Identical General Purpose registers- Allowing any register to be used in any context, simplifying compiler design.

4.6 Harvard Architecture Based- RISC designs are also more likely to feature a Harvard memory model, where the instruction stream and the data stream are conceptually separated; this means that modifying the memory where code is held might not have any effect on the instructions executed by the processor

5. RISC and CISC example

5.1 Multiplying Two numbers in memory- The main memory is divided in to locations numbered from row 1 to row 6 and column 1 to column 4 with operations and operand loaded on six registers (A,B,C,D,E or F).

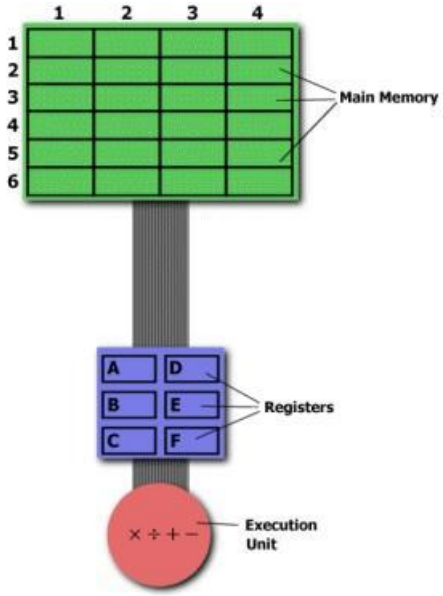


Fig 4 Representation of Storage scheme for a generic computer

5.2 Suppose one has to find multiplication of two numbers of data stored in row (2) and column 3 with data stored in row(5) and column(2). This example will solve by CISC and RISC approaches

5.2.1. CISC Approach- The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible. This is achieved by building processor.hardware that is capable of understanding and executing a series of operations. For this particular task, a CISC processor would come prepared with a specific instruction (say "MUL").

a. When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register.

b. Thus, the entire task of multiplying two numbers can be completed with one instruction:

MUL 2:3, 5:2

c. MUL is what is known as a "complex instruction."

d. It operates directly on the computer's memory banks and does not require the programmer to explicitly call any loading or storing functions.

e. It closely resembles a command in a higher level language. For instance, if we let "a" represent the value of 2:3 and "b" represent the value of 5:2, then this command is identical to the C statement "a = a x b."

5.2.2 Advantage- one of the primary advantages of this system is that the compiler has to do very little work to translate a high-level language statement in to assembly.As length of the code is relatively short, very little RAM is required to store instructions. The emphasis is put on building complex instructions directly in to the hardware.

5.3 RISC Approach- RISC processors only use simple instructions that can be executed within one clock cycle. Thus, the "MUL" command described above could be divided into three separate commands:

a. "LOAD," which moves data from the memory bank to a register,

b. "PROD," which finds the product of two operands located within the registers, and

c. "STORE," which moves data from a register to the memory banks.

d. In order to perform the exact series of steps described in the CISC approach, a programmer would need to code four lines of assembly:

LOAD A,2:3

LOAD B, 5:2

PROD A, B

STORE 2:3, A

5.4 Analysis- At first, this may seem like a much less efficient way of completing the operation. As there are more lines of code, more RAM

is needed to store the assembly level instructions. The compiler must also perform more work to convert a high-level language statement in to code for this form.

5.4.1 Advantage- As each instruction in RISC strategy requires only clock cycle to execute, the entire program will execute in approximately the same amount of time as the multi-cycle "MUL" command and also the pipelining is possible due to separating the LOAD and STORE instructions which actually reduces the amount of work that the computer system must perform. In RISC the operand will remain in the register until another value is loaded in its place.

6. Differentiate between RISC and CISC

Sr. No	**Point of discussion**	**CISC**	**RISC**
1	Emphasis	Emphasis on hardware	Emphasis on Software
2	Clock Cardinality	Includes multi-clock complex instructions	Single clock,reduced instruction only
3	Devices to Devices incorporation	Memory-to-Memory	Register to Register
4	About LOAD and STORE instructions	LOAD and STORE incorporated in instructions	LOAD and STORE are independent instructions
5	Code Size	Small code sizes, high cycles per second	Low cycles per second, large code sizes
6	About usage of transistors	Transistors used for storing complex instructions	Spends more transistors on memory registers

7. The performance equation- The following equation is commonly used for expressing a computers performance ability

$$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \text{ x } \frac{\text{cycles}}{\text{instruction}} \text{ x } \frac{\text{instructions}}{\text{program}}$$

7.1 CISC Approach- The CISC approach attempts to minimize the number of instructions per program.

7.2 RISC Approach- It also reduces the cycles per instruction at the cost of the number of the instruction per program.

8 RISC Advantages

8.1 The price of RAM has decreased dramatically. In 1977, 1MB of DRAM cost about $5,000.

8.2 By 1994, the same amount of memory cost only $6 (when adjusted for inflation). Compiler technology has also become more sophisticated, so that the RISC use of RAM and emphasis on software has become ideal.

9. RISC Processor example

9.1 Digital equipment corporation (DEC)- Alpha

9.2 Advanced Micro devices (AMD) 29000

9.3 Advanced RISC Machine(ARM)

9.4 Atmel AVR

9.5 Microprocessor without interlocked Pipeline stages(MIPS)

9.6 Precision Architecture-Reduced instruction set Computer (PA-RISC)

9.7 POWER-PC

10 CISC Example- Today used by intel x86

11.3 Instruction Level Parallelism

1. Instruction level parallelism is a family of processor and compiler design techniques that speed up execution by causing individual machine operations such as memory loads and stores, integer additions and floating point multiplications to execute in parallel.

2. The operations involved are normal RISC-style operations, and the system is handed a single program written with a sequential processor in mind.

3. Thus an important feature of these techniques is that like circuit speed improvements, but unlike traditional multiprocessor parallelism and massive parallel processing, they are largely transparent to users.

4. VLIWs and superscalars are examples of processors that derive their benefit from instruction-level parallelism, and software pipelining and trace scheduling are example software techniques that expose the parallelism that these processors can use

5. Several systems were built and sold commercially which pushed ILP far beyond in terms of the amount of ILP offered and in the central role ILP played in the design of the system.

6. Nowadays every processor manufacturing company uses ILP techniques.

7. A typical ILP processor has the same type of execution hardware as a normal RISC machine. The differences between a machine with ILP and one without is that there are may be more of that hardware for adding simple integers numbers.

8. ILP execution hardware allows multiple-cycle operations to be pipelined, so we may assume that a total of four operations can be initiated each cycle.

9. Table 1 Execution hardware for a simplified ILP processor

Functional Unit	Operations Performed	Latency
Integer Unit 1	Integer ALU operations Integer multiplication Loads Stores	1 2 2 1
Integer Unit 2 / Branch Unit	Integer ALU operations Integer multiplication Loads Stores Test-and-branch	1 2 2 1 1
Floating-point Unit 1 Floating-point Unit 2	Floating-point operations	3

10. Modern Instruction-Level Parallelism

    10.1 The beginnings of a new style of ILP called VLIW(Very Long Instruction word) emerged in the late 1970's can be more incorporated on machines from 1980's by doing some changes in semiconductor technology that has a large impact upon the entire computer industry.

10.2 Because of these technologies like Combination of RISC and ILP, Superscalar computers were developed which could issue up to two instructions for each cycle.

10.3 VLIW CPU's were offered commercially in the form of capable, general-purpose machines. Three computer start-ups - culler, multiflow and cyndrome built VLIW with varying degrees of parallelism.

11 ILP Architectures

11.1 The end result of instruction-level parallelism execution is that multiple operations are simultaneously in execution, either as a result of having been issued simultaneously or because the time to execute an operation is grater than the interval between the issuance of successive operations.

11.2 ILP architectures are classified are as follows

11.2.1 Sequential Architecture- Architectures for which the program is not expected to convey any explicit information regarding parallelism. Superscalar processors are representative of ILP processor implementation for sequential architectures.

11.2.2 Dependence Architecture-Architectures for which the program explicitly indicates the dependencies that exist between operations. Data flow computers follow this architecture.

11.2.3 Independence Architecture- Architectures for which the program provides information as to which operations are independent of one another. VLIW processors is following this architecture.

11.3 If ILP is to be achieved, between the compiler and the run time hardware, the following functions must be performed.

11.3.1 The dependencies between operations must be determined.

11.3.2 The operations that are independent of any operation that has not yet completed must be determined.

11.3.3 These independent operations must be scheduled to execute at some particular time, on some specific functional unit and must be assigned a register in to which the result may be deposited.

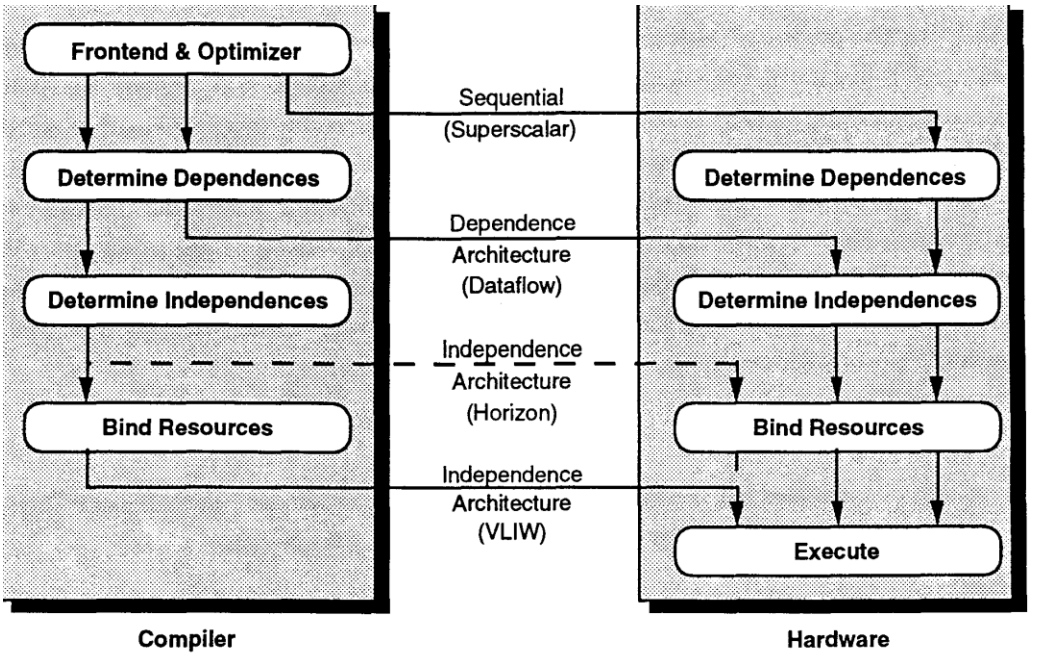


Fig 5 Division of responsibilities between the compiler and the hardware for the three classes of architecture.

11.4 Superscalar Processor-Design Issues

1. The first processors were known as a scalar which has a single pipelined functional unit exist for integer operations and one for floating point operations.

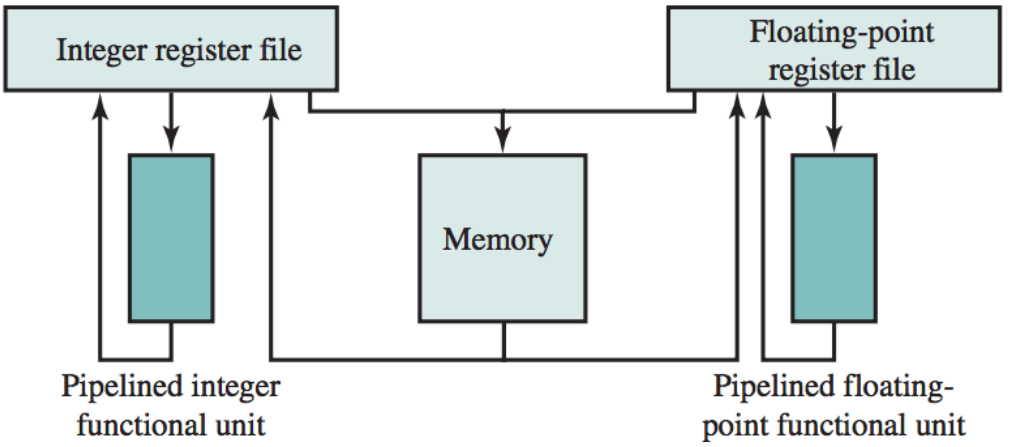


Fig 6 Scalar organization for integer and float point

2. Parallelism is not achieved with scalar processor, but need parallelism which enable multiple instructions at the different stages of pipelines. For that reasons Super Scalar processor came in to existence.

3. So Super Scalar processor has the ability to execute instructions in different pipelines in an independently and concurrently manner.

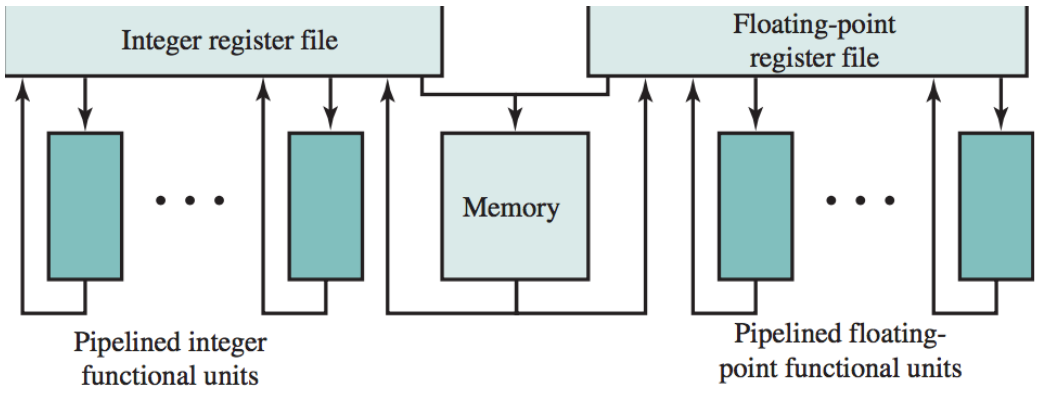


Fig 6 SuperScalar Organization

4. Pipeline concept has some problems like Resource hazard, Data Hazard (RAW(Read After Write),WAR(Write After Read),WAW(Write After Write)) and control hazards.

5. Different design issues need to be considered while designing Superscalar processor

5.1 Instruction-Level Parallelism and Machine Parallelism-

5.1.1 Instruction-level Parallelism- exists when instructions are in a sequence and are independent of each other and can be executed in a parallel.

```
Load R1 ← R2        Add R3 ← R3, "1"
Add R3 ← R3, "1"    Add R4 ← R3, R2
Add R4 ← R4, R2     Store [R4] ← R0
```

5.1.2 For eg as shown in above snippet instructions on the left are independent and can be executed in parallel, whereas instructions on the right are dependent and can be executed in parallel.

5.1.3 Machine parallelism- is a measure of the ability of the processor to take advantage of instruction-level parallelism which is determined by number of instructions that can be fetched at the same time,number of instructions that can be executed at the same time and ability to find independent instructions.

5.1.4 Three types of orderings are important in this regard: a)Order in which instructions are fetched,b)order in which instructions are

executed,c)order in which instructions update register/memory contents. So processor may alter one or more orderings to give final result correct.

5.2 Instruction Issue Policies-

5.2.1 It fall in to following categories like In-order issue with in-order completion, In-order issue with out-of-order completion, out-of-order with out-of-order completion.

5.2.2 This instruction policy can be used as a baseline for comparing more sophisticated approaches.

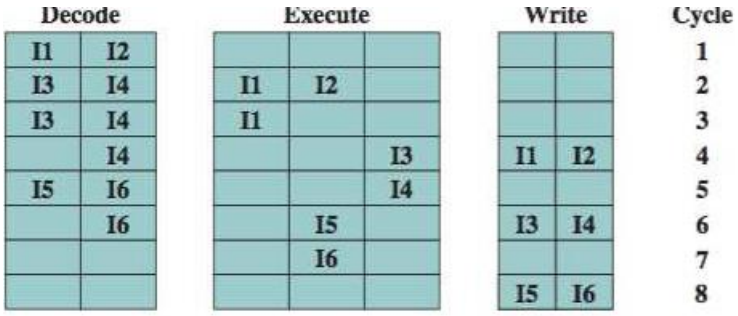


Fig 7 In-order issue with in-order completion

5.2.3 Consider the following example shown in above figure where superscalar pipeline is used for fetching and decoding two instructions at a time with separate functional units where I1 requires two cycles to execute, I3 and I4 conflict for a functional unit , I5 depends on the value produced by I4, I5 and I6 conflict for a functional unit.

5.2.4 So to avoid this conflict between instructions processor are need to design with look-ahead capability where independent instructions can be brought in to the execution stage.

5.3 Register Renaming-

5.3.1 Allocated dynamically by the processor hardware, associated with the values needed by instructions at points in time.

5.3.2 When an instruction executes that has a register as a destination so new register is allocated for that value, subsequent instructions that access the value in the register, need to refer to the allocated register.

```
I1: R3  ← R3  op R5
I2: R4  ← R3  + 1
I3: R3  ← R5  + 1
I4: R7  ← R3  op R4
```

Figure 8 Register Renaming(a)

5.3.3 There exists a problem in above shown snippet, this can be solved by renaming as shown below

$$
\begin{aligned}
&I1: R3_b \leftarrow R3_a \ op \ R5_a \\
&I2: R4_a \leftarrow R3_b + 1 \\
&I3: R3_c \leftarrow R5_a + 1 \\
&I4: R7_a \leftarrow R3_c \ op \ R4_a
\end{aligned}
$$

Figure 8 Register Renaming(b)

5.4 Branch Prediction

5.5 Superscalar Execution

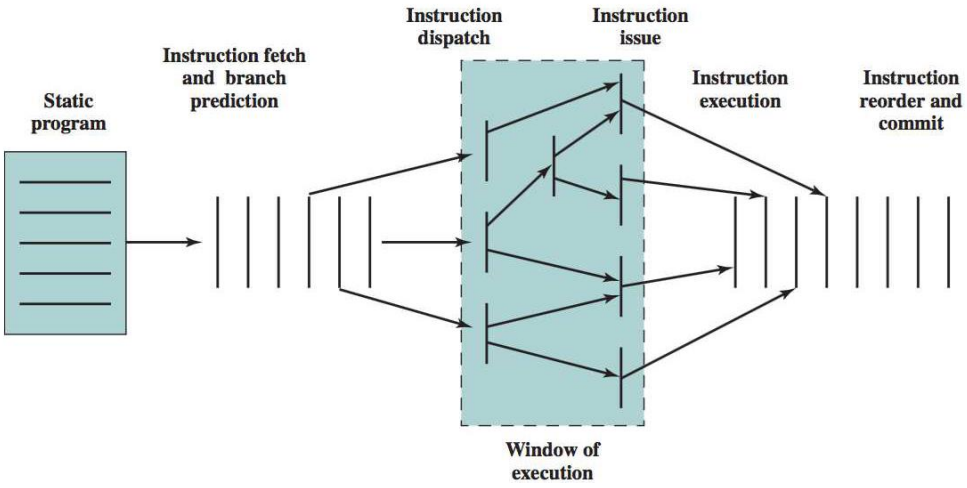


Fig 8 Conceptual depiction of superscalar processing

5.6 Superscalar Implementation

Miscellaneous Questions

Q1. Explain Design issues of Super scalar processor

Q2. Differentiate between RISC V/S CISC

Q3. Write a short note on RISC Processor

Q4. Write a short note on CISC processor

Q5. Depict the concept of Instruction level parallelism

12

CONTROL UNIT

Unit Structure

12.1 Introduction

Generally ,a digital system is separated into two parts : Data processing unit and Control unit .The data processing unit is a collection of functional units capable of performing certain operations on data , whereas control unit issues control signals to the data processing part to perform operations on data. These control signals select the functions to be performed at specific times and route the data through the appropriate functional units. In other words , the control unit decides the sequence in which certain sets of micro- operations to be performed to carry out particular operation on data .Hence ,we can say that, the data processing unit is logically reconfigured by the control unit and it is intimately involved in the sequencing and timing of the data processing unit.

In this chapter, we are going to study the implementation of the control unit part of a processor using two basic approaches: hardwired and micro programmed .

Before going to study these design approaches we will see functional requirements of the central processing unit and how various elements of the CPU are controlled by control unit.

12.2 Micro-operations

The primary function of a CPU unit is to execute sequence or instructions stored in a memory, which is external to the processor unit. The sequence of operations involved in processing an instruction constitutes an instruction cycle, which can be subdivided into three major phases: Fetch cycle, decode cycle and execute cycle. To perform fetch, decode and execute cycles the CPU unit has to perform set of operations called micro- operations.

In this section, we are going to see how events of any instruction cycle can be described as a sequence of such micro-operations and how the concept of micro-operations serves as a guide to the design of the CPU unit.

12.2.1 Micro-operations for Fetch Cycle

The fetch cycle occurs at the beginning of each instruction cycle and it causes an instruction to be fetched from memory. To find the micro-operations involved in the fetch cycle, look at the typical processor structure shown in the Figure12.1

The four registers AR, PC, DR and IR are involved in the fetch cycle. The address lines of the system bus are connected to the AR (Address Register) is connected to the data lines of the system bus and it contains the value to be stored in memory or the value read from memory . The PC (Program counter) holds the address of the next instruction to be fetched and IR (Instruction Register) holds the last instruction fetched.

Let us look at the sequence of micro-operations to be performed for fetch cycles.

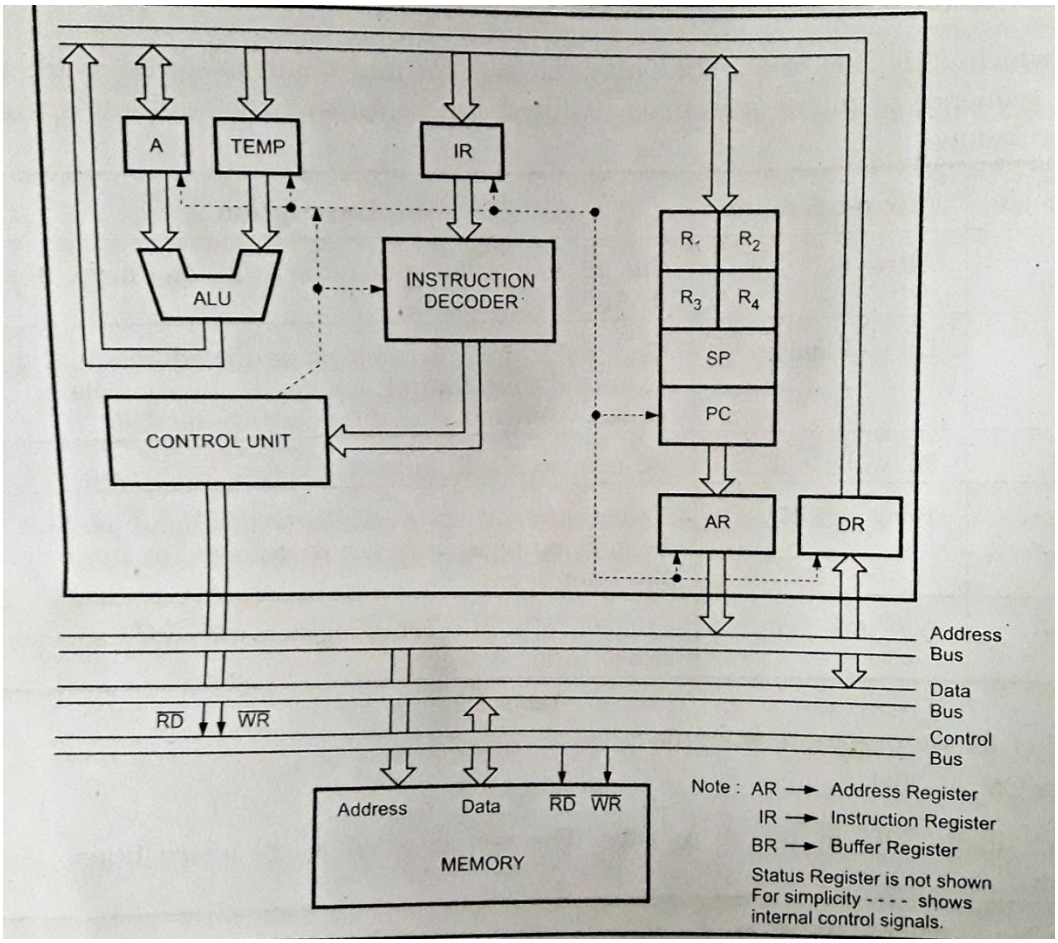


Figure 12.1 : Typical CPU structure

Table 12.1

Sr.No.	Micro-operation	Description
1	AR ← PC	The address of the next instruction to be fetched is in PC and it is necessary to load this address in AR, since this is the only register which gives address to the memory.
2	DR ← Memory	Once the address is available on the address bus, the read command from control unit copies the contents of addressed memory location to the IR.
3	PC ← PC +1	PC is incremented to point the next instruction or the operand of the current instruction.
4	IR ← DR	The contents of DR are copied to the IR for decoding.

12.2.2 Micro-operations for Execute Cycle

The instruction in the IR is decoded and CPU has to perform a particular set of micro-operation depending on the instruction, in the execution cycle.

Let us consider an add instruction ADD A, 20H

Which adds 20H to the contents of A register and result is stored in the A register.

The sequence of micro-operations required for performed for addition operation is as given below.

Table 12.2

Sr. No.	Micro-operation	Description
1	AR ← PC	The address of the operand is in PC and hence it is loaded in AR.
2	DR ← Memory	Once the address is available on the address bus, the read command from control unit copies the contents of addressed memory location (20H) to the DR.
3	PC ← PC +1	PC is incremented to point the next instruction.
4	Temp ← DR	To perform add operation the second input for ALU must be in Temp register. Hence contents of DR are copied in the Temp register.
5	A ← A+Temp	Contents A and Temp registers are added and result is stored in the A register.

Let us see one more example

MOV A, 20H

which loads 20H in the register. The sequence of micro-instructions is as given below:

Table 12.3

Sr. No.	Micro-operation	Description
1	AR ← PC	The address of the operand is in PC and hence it is loaded in AR.
2	DR ← Memory	Once the address is available on the address bus, the read command from control unit copies the contents of addressed memory location (20H) to the DR.
3	PC ← PC +1	PC is incremented to point the next instruction.
4	A ← DR	The contents of DR are copied into A register.

12.2.3 Micro-operations for Interrupt Cycle

At the completion of each execution cycle, a test is made to determine whether any enabled interrupts have occurred. If so, the interrupt cycle occurs. The nature of this cycles varies greatly from one computer to another. We present a vary simple sequence of events, as illustrated in Table 12.4

Table 12.4

Sr.No.	Micro-operation	Description
1	AR ← PC	The address of the operand is in PC and hence it is loaded in AR.
2	DR ← Memory	Once the address is available on the address bus, the read command from control unit copies the contents of addressed memory location address of ISR to the DR.
3	AR ← SP	Address of the stack memory (top of stack) is available in the SP. It is loaded in AR.
4	PC ←PC + 1	PC is incremented to point the next instruction.
5	Memory ← PC	The contents of PC are saved in the stack memory

6	PC ⟵ DR	Load the address of ISR (Interrupt Service Routine) in PC.
7	SP ⟵ SP - 1	SP contents are decremented to point the next memory location.

12.2.4 Fetching a Word from Memory

With this understanding now we see how various functions are performed by CPU by executing specific sequences of operations. For this consider the single bus organisation of CPU, as shown in Figure 12.2

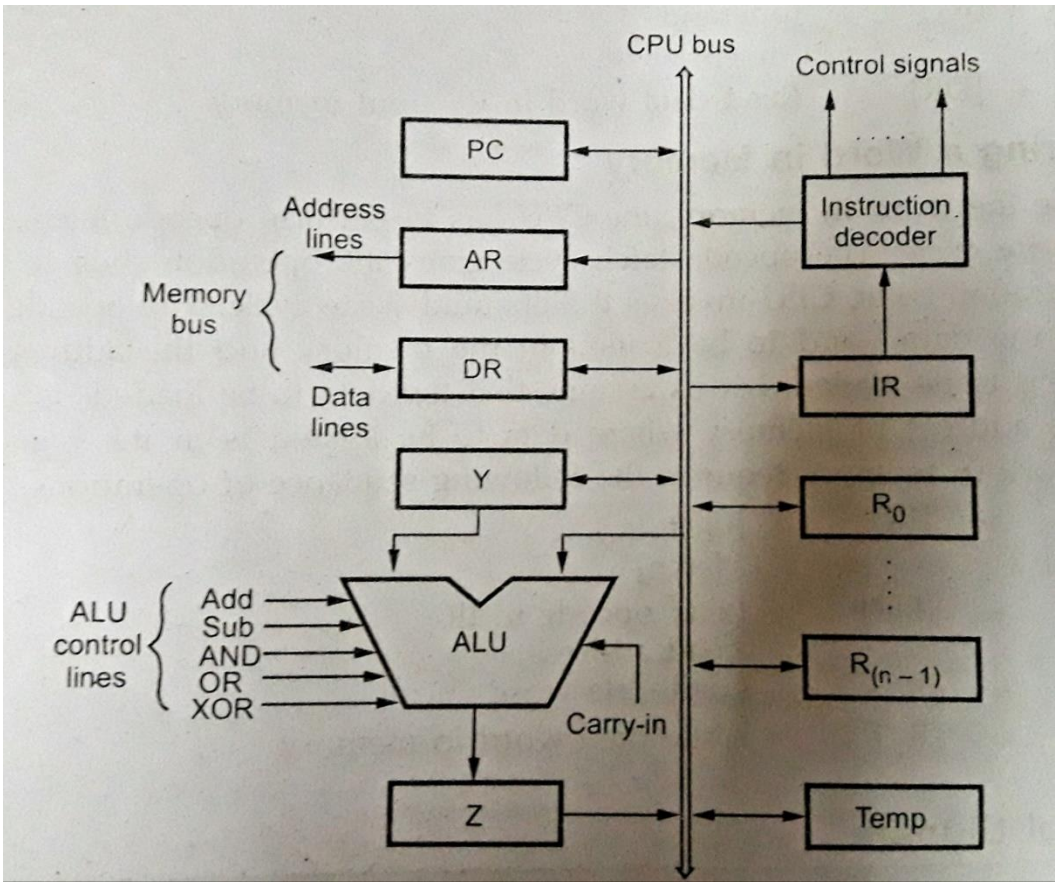


Figure 12.2 : Single Bus Organisation of CPU

In single bus organisation the arithmetic and logic unit and all CPU registers are connected through a single common bus. It also shows the external memory bus connected to the address (AR) and data register(DR).

To fetch the word from memory, the CPU has to opcode fetch cycle and then operand fetch cycle. The opcode fetch cycle gives operation code of fetching a word from memory to the CPU .The CPU then invokes the operand fetch cycle.

The opcode specifies the address of the memory location where information is stored .

Let us assume that the address of memory word to be read is in register R_2 .Then CPU transfers the address (the contents of register R_2) to the AR, which is connected to the address lines of the memory bus. Hence, the address is transferred to the memory. Meanwhile, the CPU activates the read signal of the memory to indicate the read operation is needed. As a result, memory copies data from the addressed register on the data bus. The CPU then reads this data from data register and loads it in the specified register.

As an example, assume that the address of the memory location to be accessed is in register R_2 and that the memory data are to be loaded into register R_3 .This is done by the following sequence of operations.

1. AR ⟵ [PC] ; opcode
2. Read ; fetch
3. IR ⟵ [DR] ; Load opcode in IR
4. AR ⟵ [R_2] ; opcode
5. Read ;fetch
6. R_3 ⟵ [DR] ; read data word in R_3 from memory

12.2.5 Storing a Word in Memory

To store the word in memory, the CPU has to perform opcode fetch cycle and the operand write cycle. The opcode fetch cycle gives the operation code to the CPU .As per the operation code, CPU invokes the operand write cycle. The opcode specifies the address of the data word to be loaded in the memory and the address of the memory where data is to be loaded.

Let us assume that the data to be loaded is in the register R_2 and the address of memory where data to be loaded is in the register R_3 . Then storing a word in memory requires the following sequence of operations:

1. AR ⟵ [PC] ; opcode
2. Read ; fetch
3. IR ⟵ [DR] ; Load opcode in IR Load address
4. AR ⟵ [R_3] ; Load data word
5. DR ⟵ [R_2] ; Load data word in memory
6. Write

12.3 Functional Requirements

We have seen the micro-operations for fetch, execute and interrupt cycles. Fetch and interrupt cycles have a fixed sequence of micro- operations whereas micro-operations in the execute cycle depend on the instruction. With this exercise we can defined operations of CPU as a sequence of micro-operations. Now we will see how control unit controls the sequence of micro- operations. To do this, control unit performs two basic tasks:

Sequencing: The control unit causes the CPU to step through a series of micro-operations in the proper sequence, based on the instructions being executed.

Execution: The control unit causes the CPU to perform each micro-instruction.

Let us see the control signals required to perform sequencing and execution of micro-operations. Figure12.3 shows the internal CPU structure with control signals.

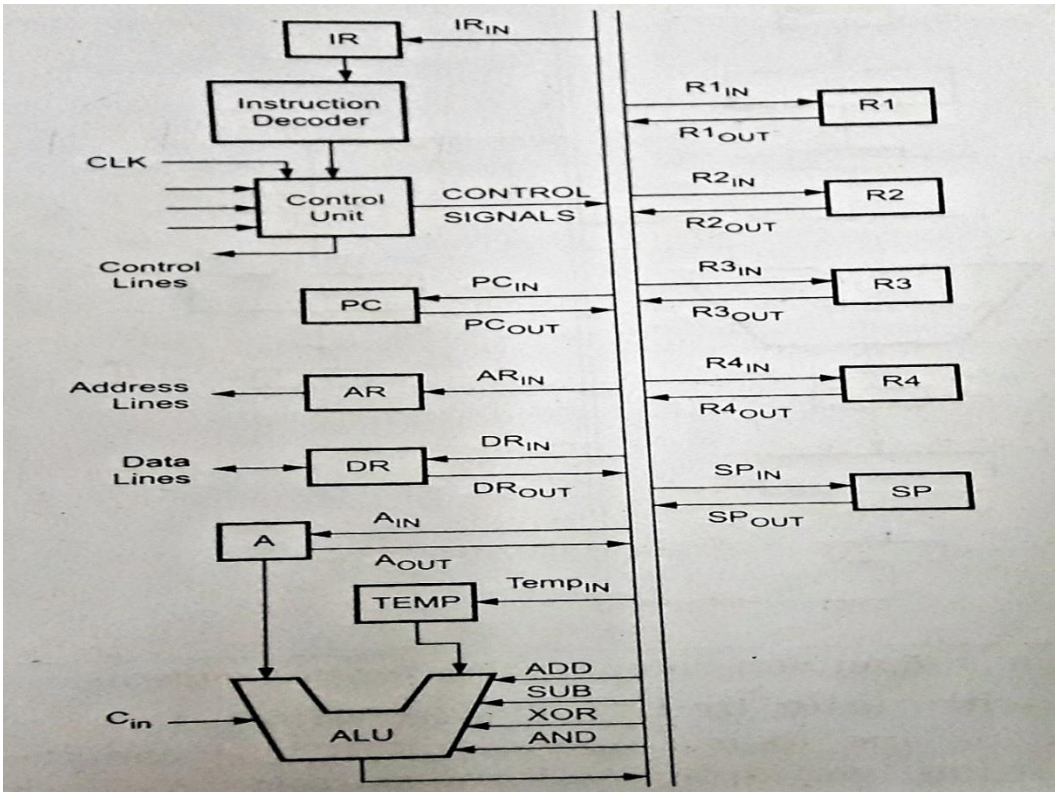


Figure 12.3 Internal CPU structure with control signals

12.3.1 Register Transfers

In figure12.3, the data transfer between registers and common bus is shown by a line with arrow heads. But in actual practice each register has input and output

gating and these gates are controlled by corresponding control signals. This is illustrated in figure 12.4.

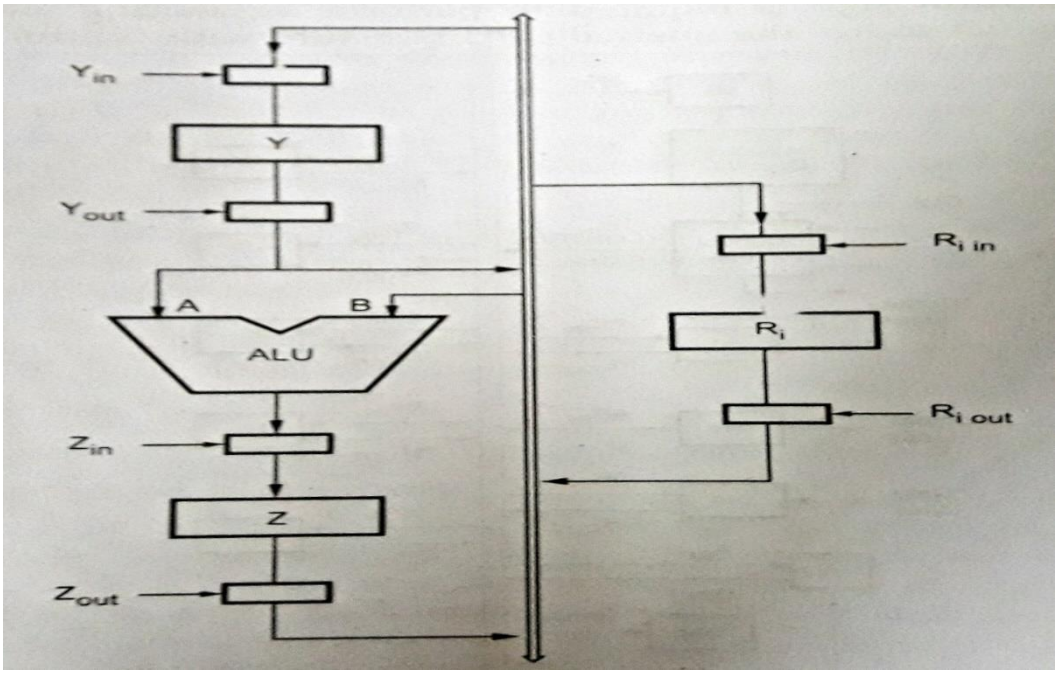


Figure 12.4 : Input and output gating for the register

As shown in Figure12.4, control signals. $R_{i\ in}$ and $R_{i\ out}$ control the input and output gating of register R_i . When $R_{i\ in}$ is set to 1, the data available on the common data bus is loaded into register R_i . Similarly, when $R_{i\ out}$ is set to 1 , the contents of register R_i are placed on the common data bus.

The input and output gates are nothing but the electronic switches which can be controlled by the control signals. When signal is 1, the switch is ON and when signal is 0, the switch is OFF.

12.3.2 Performing an Arithmetic or Logic Operation

ALU performs arithmetic and logical operations. It is a combinational circuit that has no internal memory. It has two inputs and one output. One input for ALU is from register Y and other input for ALU is from the common bus. The result of the ALU is stored temporarily in register Z. Let us find the sequence of operations required to subtract contents of register R_2 from register R_1 , and store the result in register R_3 . The sequence can be given as follows:

1. $R_{1\ out}$, y_{in}

2. $R_{2\ out}$, sub , z_{in}

3.Z_{out} , $R_{3\ in}$

Example :- Find the complete control sequence for execution of the instruction

Add (R_3) , R_1 for the single-bus CPU.

Solution :- This instruction adds the contents of register R_1 to the memory location whose address is specified by the register R_3 . The result is then stored in the register R_1 .The execution of this instruction requires following sequence of operations.

1. PC_{out} , AR_{in} ; [opcode

2. Read DR_{out} , IR_{in} ; fetch]

3. $R_{1\ out}$, $Y_{\ in}$; Load contents of R_1 in Y

4. $R_{3\ out}$, AR_{in} ; [load the contents pointed by R_3

5. Read ,DR_{out} ; on common data bus

6. Y_{out} , Sub, Z_{in} ; subtract B from A

7. Z_{out} ,$R_{1\ in}$; store the result inR_1

Until now we have seen inputs, outputs and general functions of control unit. In this section we are going to see implementation of control unit. As mentioned earlier, there are two general approaches to control unit design.

Hardwired control

Micro programmed Control

We discuss each of these techniques in detail, starting with Hardwired control

12.4 Hardwired Implementation

In the hardwired implementation, the control units use fixed logic circuits to interpret instructions and generate control signals from them .The Fgure12.5 shows the typical hardwired implementation unit.

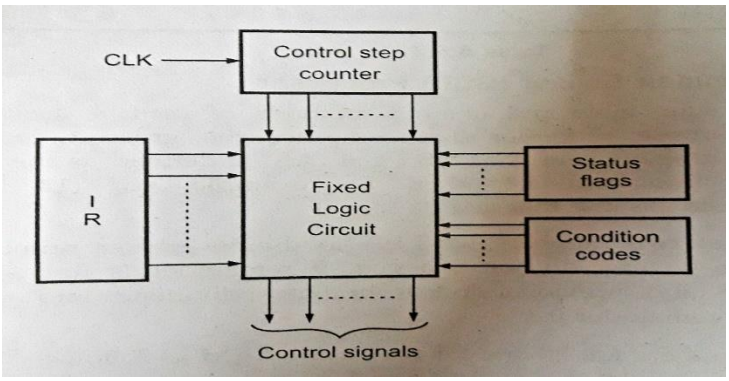


Figure 12.5 Typical Hardwired Implementation Unit

The hardwired control unit design involves various trade-offs between the amount of hardwired used, the speed of operation, and the cost of the design process itself. To illustrate these issues, we discuss two simplified and systematic methods for the design of hardwired controllers.

State-table Method or Classical Method: It is standard algorithmic approach to sequential circuit design. It attempts to minimize the amount of hardware required to implement control unit.

One-hot Method: It is a heuristic method based on the use of one D flip-flop per state to generate control signals.

12.4.1 State Tables

The behaviour of a sequential circuit for the design of control unit can be represented by a state table. We know that state table gives the relationship between inputs, present state, next state and outputs. For Moore sequential circuit the output of sequential circuit is a function of present state only. On the other hand, for Mealy sequential circuit, the output is a function of present state as well as present input. The Table12.5 shows the general form of state table. The rows of the state table correspond to set of internal states. These states are determined by the information stored in the machine at discrete points of time (clock cycles).

Table 12.5 : State Table

Present state	Next state		Output Y
	X=0	X=1	
a(00)	a	c	0
b(01)	b	a	0
c(10)	d	c	1
d(11)	b	d	0

12.4.1.1 Greatest Common Divisor (GCD) Processor

To illustrate the state table and one- hot methods of control design we see the design of special purpose processor that computes the greatest common divisor, gcd (A,B), of two positive integers A and B: gcd (AB) is defined as the largest integer that divides exactly into both A and B .

For example , gcd (12,20) = 4 , and gcd (13,21) = 1 .

We can assume that gcd (0,0)=0

To find the gcd of two numbers we have to divide greater number by smaller number, if remainder is zero, divisor is a gcd. If remainder is not zero, remainder becomes new divisor and previous divisor becomes dividend and this process is repeated until we get remainder 0.

For example , if numbers are 20 and 15 we can find gcd as follows :

20 ÷ 15 =1 Remainder 5 i.e. ≠ 0

15 ÷ 5 = 3 Remainder 0

Gcd = 5

The hardware description language (HDL) algorithm of the above method is as given below. In this program division of two numbers is achieved by repetitive subtraction. In this algorithm the smaller number is subtracted from greater number and result is stored as a new greater number. If greater number is still greater than the smaller number subtraction is repeated; otherwise smaller number becomes a new greater number and last remainder becomes a new smaller number. This process is repeated until remainder is zero.

Algorithm

```
gcd ( in: A,B ; out : Y)
register  RegA , RegB. RegTemp;
RegA = A;     {Read first number}
RegB = B;   { Read second number }
while RegA  >  0  do
      begin
   if RegA  ≤  RegB then  { swap two numbers }
  begin
 RegTemp := RegB
RegB := RegA ;
RegA := RegTemp ;
end
RegA := RegA  - RegB  ;    {Subtract smaller number from greater number }
  end
 Y := RegB  ;  { Write the result  }
```

end gcd;

The above algorithm goes through following steps when A= 12 and B= 20.

RegA := 12 , RegB := 20

Loop 1 : RegA > 0 true

RegA ≤ RegB true

RegA := 20;

RegB :=12

RegA := RegA – RegB = 8 ;

Loop 2 : RegA > 0 true

RegA ≤ RegB true

RegA := 12;

RegB := 8 ;

RegA := RegA – RegB = 4;

Loop 3 : RegA > 0 true

RegA ≤ RegB true

RegA := 8;

RegB :=4

RegA := RegA – RegB = 4;

Loop 4 : RegA > 0 true

RegA ≤ RegB true

RegA := 4;

RegB := 4;

RegA := RegA – RegB = 0;

Loop 5 : RegA > 0 false

Y:= RegB =12

Therefore ,gcd(12,20) = 4

To implement the gcd processor we need pair of register RegA and RegB to store the corresponding variables, to perform subtraction , two magnitude comparator : one to compare two numbers and other to compare greater number with zero and multiplexer for data routing. This is illustrated in Figure 12.6 below:

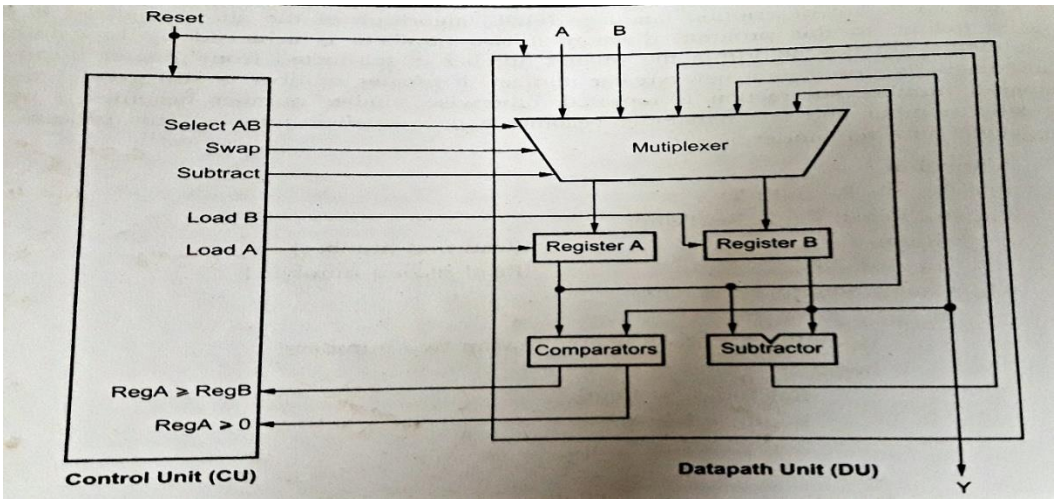


Figure 12.6 Hardware for gcd processor

As shown in figure 12.6 , control unit generate control signals Load A and Load B to load data in Register A and Register B independently. The multiplexer with select inputs (select AB , swap and subtract) decides the inputs for register A and B. With this hardware it is possible to read from and write to a register in the same clock cycle. This allows us to swap the contents of register A and register B without the need of temporary register.

The control unit provides select AB signal to route A and B, to register A and B , respectively. The swap signal from the control unit, controls the swap operation, i.e. A:=B, B:=A. Finally, the subtract signal from the control unit, controls the subtraction RegA := RegA – RegB by routing the output of the subtraction to RegA through multiplexer. The input signals to the control unit are an asynchronous Reset signal, two comparison signals (RegA $\leq$ RegB) and (RegA > 0) generated by data path unit and internal clock signal.

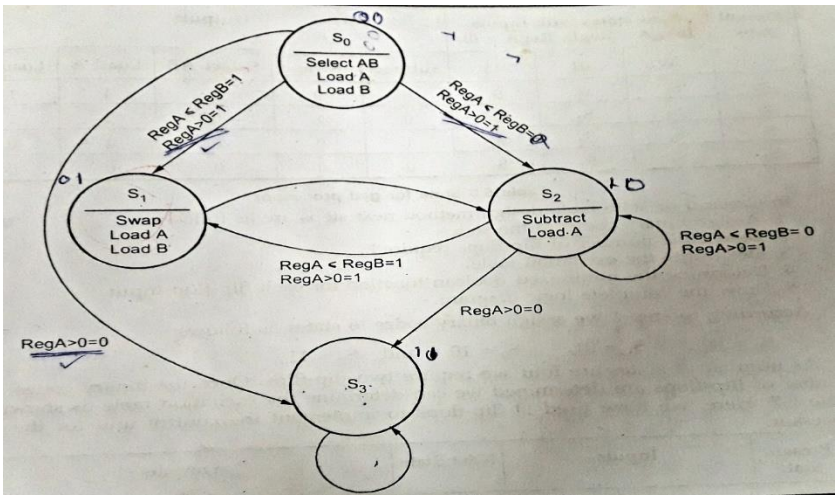


Figure 12.7 State diagram for gcd processor

We can easily identify the set of states for control unit by examining the behaviour defined in algorithm. These states are indicated in the state diagram shown in figure

A state S_0 is entered when Reset becomes 1; this state also loada A and B into the register A and B , respectively. The subsequent actions of gcd processor are either a swap or a subtraction, for which we assign the states S_1 and S_2 , respectively .A final state S_3 indicates that the gcd (A,B) has been computed.

From the state diagram shown in Figure12.7 , we can determine the state table for gcd processor as shown in table12.6. In state S_0 , if input RegA $>$ 0 is not true (RegA $>$ 0) transaction S_0 S_3 is made irrespective of other input

RegA $\leq$ RegB.

On the other hand if input RegA $>$ 0 is true (RegA $>$ 0=1) the while loop is entered , and a transaction is made to S_1 to perform a swap if RegA $\leq$ RegB is true (RegA $\leq$ RegB =1) ; otherwise the transition is made to S_2 to perform a subtraction.

Since a subtraction always follows swap, all next sate entries in the present state S_1 are S_2 . The next state entries for state S_2 are same as state S_0 . The outputs are activated as shown in the state diagram. The state S_3 is dead state; it is not affected by any input accept reset.

Table 12.6 State for gcd processor

Present state	**Next states with inputs (RegA ≤ RegB, RegA > 0)**			**Outputs**				
	X0	**01**	**11**	**Subtract**	**Swap**	**Select AB**	**Load A**	**Load B**
S_0	S_3	S_2	S_1	0	0	1	1	1
S_1	S_2	S_2	S_2	1	0	0	1	1
S_2	S_3	S_2	S_1	1	0	0	1	0
S_3	S_3	S_3	S_3	0	0	0	0	0

In classical or state table design method next steps are as follows :

1. Determine binary codes to the states.
2. Determine number of flip-flops required.
3. Determine the excitation table.
4. Determine the minimized Boolean functions for each flip-flop input.
5. Draw the complete logic diagram.

According to step 1 we assign binary codes to states as follows :

$S_0 = 00$, $S_1 = 01$, $S_2 = 10$ and $S_3 = 11$

As number of states are four we require two flip-flops. Once the binary codes and number of flip-flops are determined we can determine the excitation table as shown in Table12.7 . Here we have used D- flip- flops to implement the control unit for the gcd processor.

Table 12.7 : Excitation table for gcd processor

Present State	**Inputs**		**Next State**	**Outputs**				
D_1 D_0	**RegA ≤ RegB**	**Reg > 0**	**D_1^+ D_0^+**	**Subtract**	**Swap**	**Select AB**	**Load A**	**Load B**
S_0 0 0	X	0	1 1	0	0	1	1	1
0 0	0	1	1 0	0	0	1	1	1
0 0	1	1	0 1	0	0	1	1	1
S_1 0 1	X	0	1 0	0	1	0	1	1
0 1	0	1	1 0	0	1	0	1	1
0 1	1	1	1 0	0	1	0	1	1
S_2 1 0	X	0	1 1	1	0	0	1	0
1 0	0	1	1 0	1	0	0	1	0
1 0	1	1	0 1	1	0	0	1	0
S_3 1 1	X	0	1 1	0	0	0	0	0
1 1	0	1	1 1	0	0	0	0	0
1 1	1	1	1 1	0	0	0	0	0

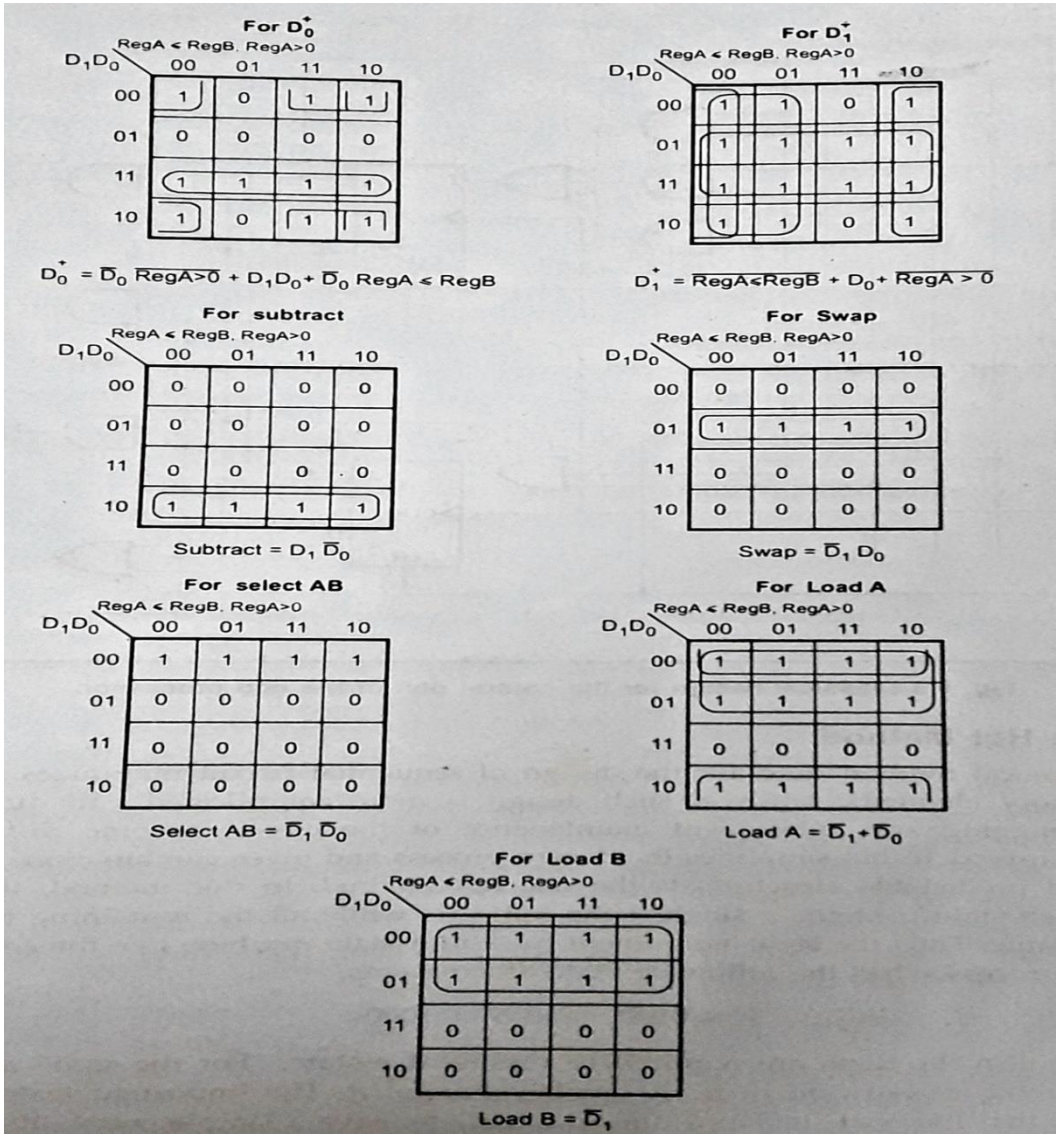


Figure 12.8 : K- map Simplification

Logic Diagram

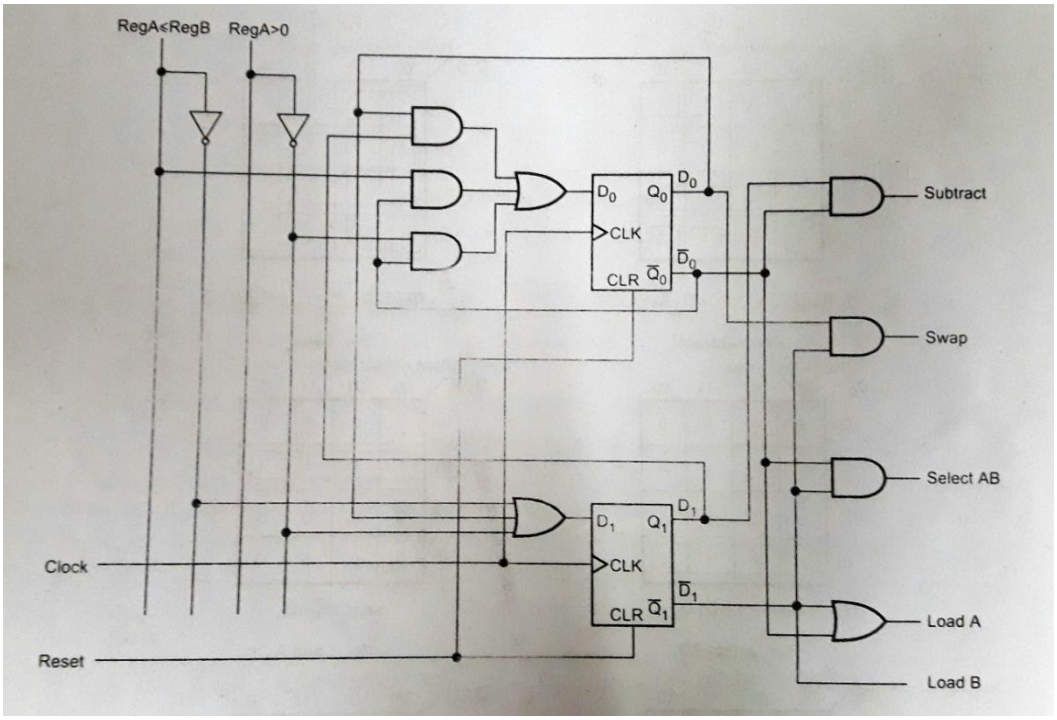


Figure 12.9 : Classical design for the control unit of the gcd processor

12.4.2 One Hot Method

The classical method used for the design of sequential circuit minimizes a control unit's memory elements; however such design is more complicated and due to that design debugging and subsequent maintenance of the circuit become difficult. An alternative approach that simplifies the design process and gives combinational circuit a regular and predictable structure, is the one hot method. In this method, the binary value of each state contains a single 1- hot bit while all the remaining bits are 0 , hence the name. Thus the state assignment for a four-state machine like the control unit of the gcd processor has the following state assignments.

$S_0 = 0001$, $S_1 = 0010$, $S_2 = 0100$ and $S_3 = 1000$

In general , n flip-flops are required to represent n-states. For the same reason, the one-hot method is restricted to fairly small values of n .The important feature of this technique is that the next-state and output equations have a simple, symbolic form and can be written down directly from the control units state table.

In one hot design we know that n flip-flops are required to represent n-states.If we use D flip-flops then state S_i corresponds to hot variable D_i . If we assume that $I_{j,1}$, $I_{j,2}$, $I_{j,n}$ denote all input combinations that cause transition from S_j to S_i .Then each

AND combination of the form D_j , I_{jk} must make D_i= 1 .Hence, considering all such combinations that cause transition to S_i , we can write

$D_i^+ = D_1 (I_{1,1} + I_{1,2} + \ldots\ldots\ldots + I_{1,n1}) + D_2 (I_{2,1} + I_{2,2} + \ldots.. + I_{2,n2}) +$

This gives the Boolean expression in the SOP form

$D_i^+ = D_1 I_{1,1} + D_1 I_{1,2} + \ldots\ldots\ldots\ldots\ldots + D_1 I_{1,n1} + D_2 I_{2,1} + D_2 I_{2,2} + \ldots..+ D_2 I_{2,n2} + \ldots..$

Which can be implemented using an AND-OR or NAND-NAND logic. Let us see the implementation of the control unit of the gcd processor using one-hot method.

Refer the state diagram for the control unit of the gcd processor given in Figure .State S_1 appears as a next state only for S_0 and S_2 in each case input combination (RegA ≥ RegB) . (RegA > 0). Hence equation becomes

$D_1^+ = D_0$. (RegA ≤ RegB). (RegA > 0) + D_2 . (RegA ≤ RegB). (RegA > 0)

Applying similar method we can write the entire set of next- state and output equations for the gcd processor as given below

$D_1^+ = 0$

$D_1^+ = D_0$. (RegA ≤ RegB). (RegA > 0) + D_2 . (RegA ≤ RegB). (RegA > 0)

$D_2^+ = D_0$. ($\overline{\text{RegA} \leq \text{RegB}}$). (RegA > 0) + D_1 + D_2 .($\overline{\text{RegA} \leq \text{RegB}}$). (RegA >0)

$D_3^+ = D_0$.(RegA > 0) + D_2 + (RegA > 0) + D_3

Subtract = D_2

Swap = D_1

Select AB= D_0

Load A = D_0 + D_1 + D_2

Load B = D_0 + D_1

The figure 12.10 shows the logic diagram for the control unit of the gcd processor using one hot design method.

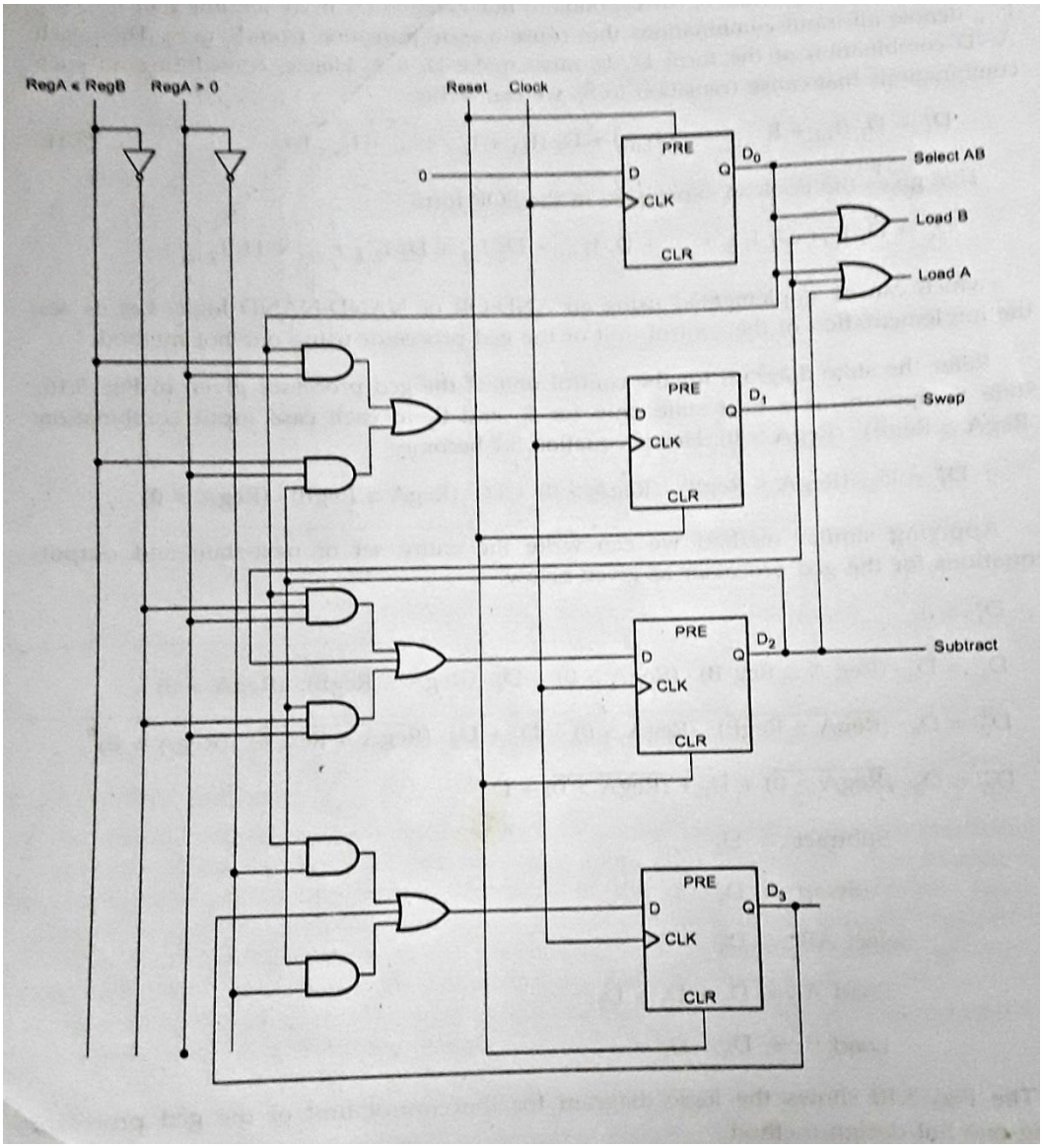


Figure 12.10 : One hot design for the control unit of the gcd processor

12.5 Processor Control

Consider the accumulator based CPU as shown in figure .It consists of a data path unit DPU designed to execute the set of 10 basic single address instructions listed in table .It is assumed that the instructions are of fixed length and they act on data words of the same fixed length , say 32 bits. The program control unit PCU is responsible for managing the control signals linking the PCU to the DPU , as well as the control signals between the CPU and the external memory M.

Table 12.8: Instruction set

Instruction Type	Instruction	Comment
Data Transfer	LD X	AC ⟵ M(X) : Load data from location X in the memory into AC
	ST X	M(X) ⟵ AC : Store contents ofAC in location X of memory
	MOV DR, AC	DR ⟵ AC : Copy contents of AC to DR
	MOV AC, DR	AC ⟵ DR : Copy contents of DR to AC
Data processing	ADD	AC ⟵ AC +DR : Add AC to DR and store result in AC
	SUB	AC ⟵ AC -DR : Subtract DR from AC and store result in AC
	AND	AC ⟵ AC and DR : AND DR to AC and store result in AC
	NOT	AC ⟵ not AC : Complement contents of AC
Program control	BRA addr	PC ⟵ M(addr) : Jump to instruction with address addr.
	BZ addr	If AC= 0 then : Jump to instruction with address PC ⟵ M(addr) addr if AC = 0.

To design the circuit for PCU, it is necessary to identify the relevant control actions (micro- operations) needed to process the given instruction set using the hardware from figure12.11 The figure12.12 shows the flowchart which gives the behavioural description of the CPU.

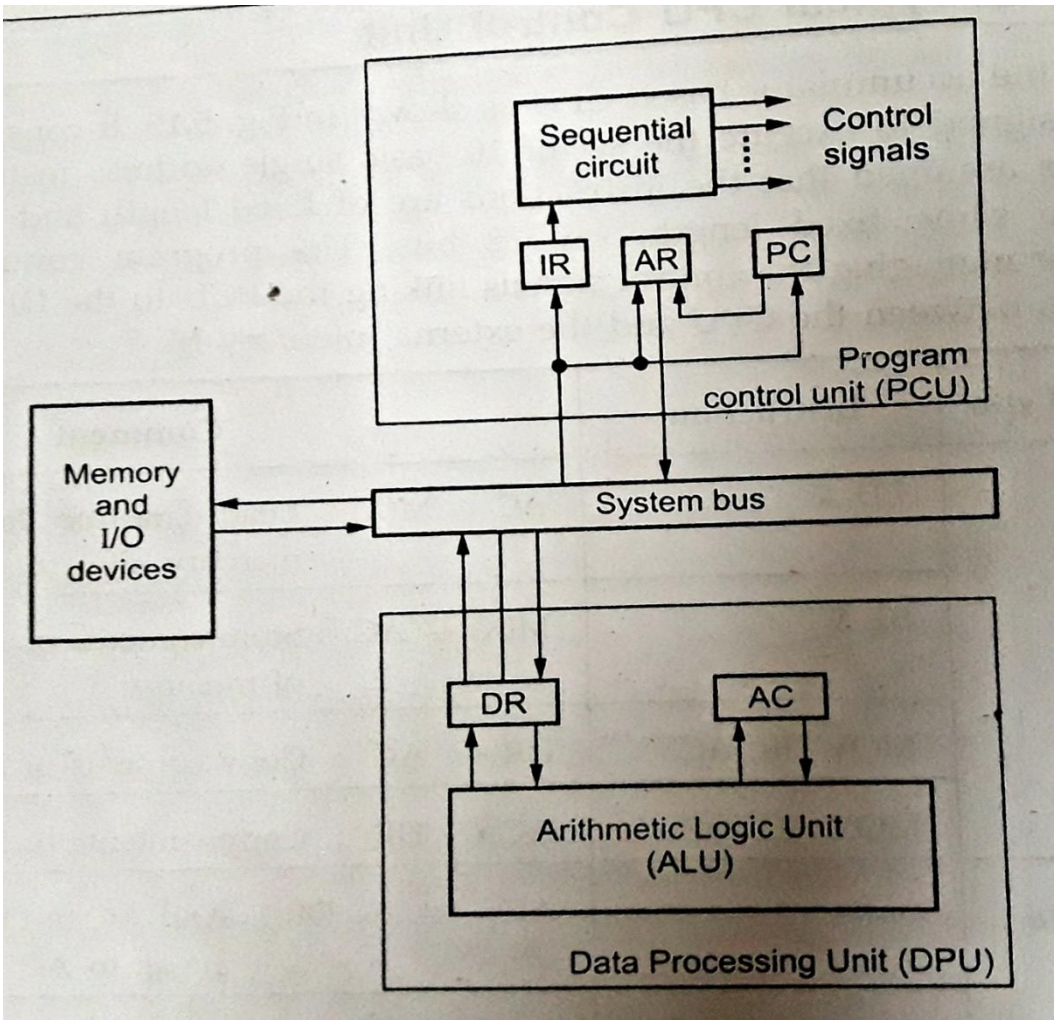


Figure 12.11: Accumulator based CPU organisation

We know that, all instructions require a common instruction- fetch step. It is followed by an execution step that varies with instruction to instruction. In the fetch step, the contents of the PC are copied into memory address register AR. A memory read operation is then executed, which transfers the instruction word I (opcode) to memory data register DR. This is expressed in the flowchart as DR M(AR).

The opcode is then transferred to instruction register IR, where it is decoded ; at the same time PC is incremented to point to the next consecutive instruction in the memory.

The operations required to execute instruction will depend on the opcode. For example, load instruction LD X is executed in three steps : the address field of LD X is transferred to AR , the contents of the addressed location are transferred to DR by performing memory read operation [DR M(AR)] , and finally, the contents of DR are transferred to the accumulator AC. As shown in the figure12.12, instruction require 1 to 3 steps for execution.

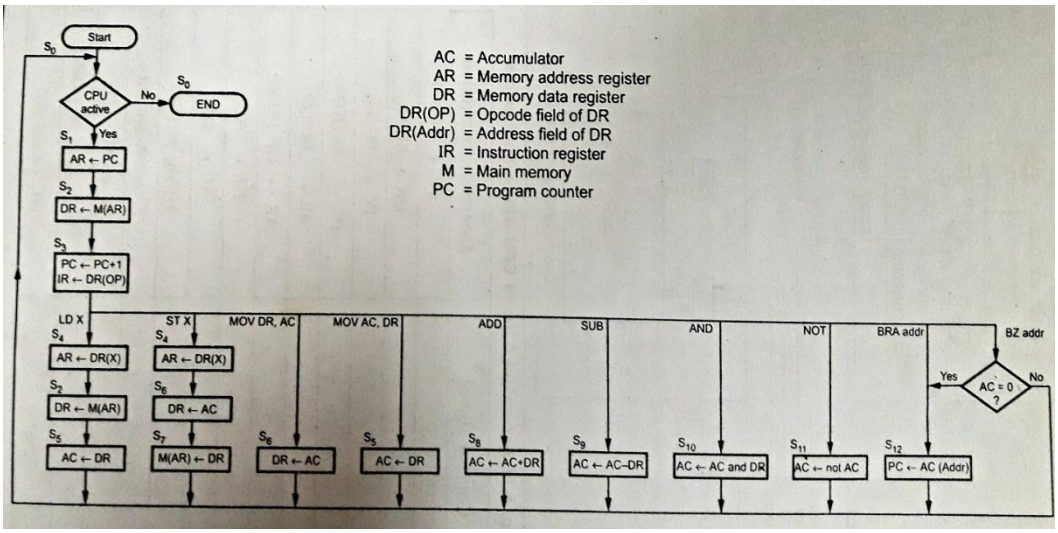


Figure 12.12 Flowchart of the accumulator based CPU

The micro operations shown the flowchart implicitly determine the control signals and control points needed by the CPU. The Table12.9 shows the set of the control signals for the CPU and their functions, while Figure12.13 shows the approximate positions of the control points in both the PCU and DPU.

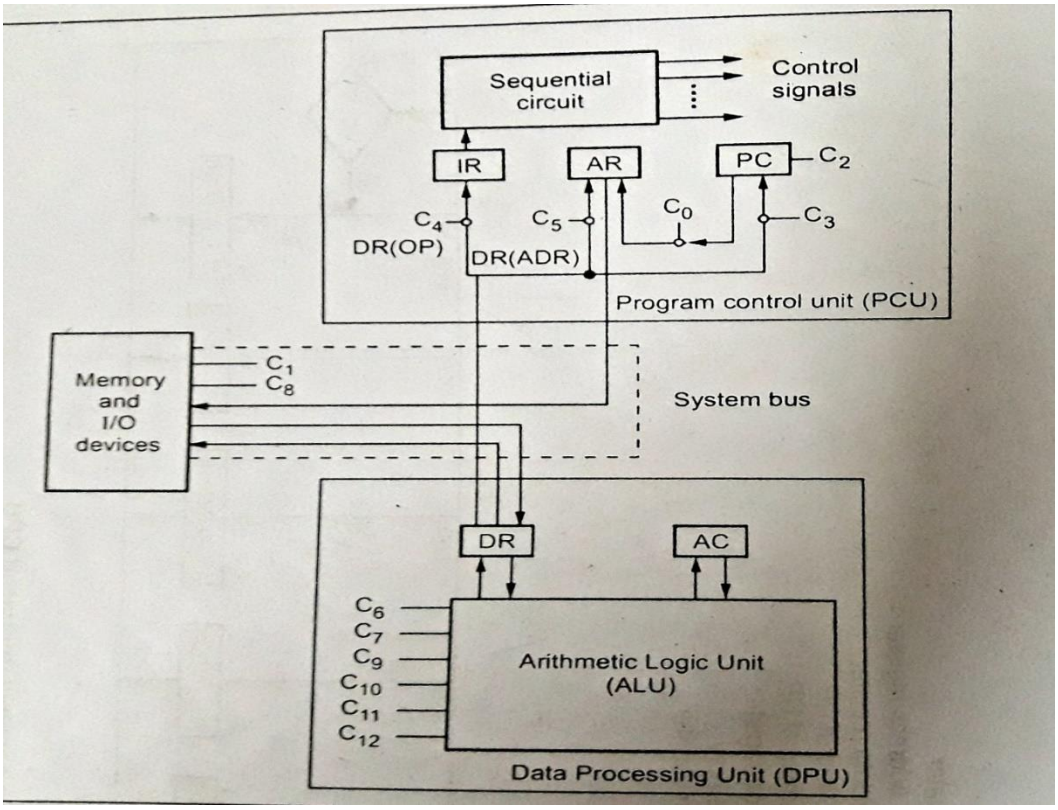


Figure 12.13 : Accumulator based CPU with control points

Table 12.9 : - Control signal definitions for the accumulator based CPU

Control signal	Operation controlled
C_0	AR ← PC
C_1	DR ← M (AR)
C_2	PC ← PC+1
C_3	PC ← DR (ADR)
C_4	IR ← DR (OP)
C_5	AR ← DR (ADR)
C_6	DR ← AC
C_7	AC ← DR
C_8	M (AR) ← DR
C_9	AC ← AC+DR
C_{10}	AC ← AC - DR
C_{11}	AC ← AC and DR
C_{12}	AC ← not AC

The control signals shown in table can be generated using sequential circuit. This sequential circuit will have 10 inputs signals from instruction decoder, one per instruction type , one input status signal (AC = 0) and one input start signal. Thus, the sequential circuit will have 12 inputs and 13 outputs. As shown in Figure12.13, each distinct action box is assigned to a different state. Therefore, the circuit has 13 states S_0 : S_{12} .Once the inputs, outputs and internal states are known the circuit can bee implemented using standard procedure for sequential machine design.

12.6 Microprogrammed Control

In this section we discuss the design of control units that use micro-programs to interpret and execute instructions.

12.6.1 Basic Concepts

Every instruction in a CPU is implemented by a sequence of one or more sets of concurrent micro-operations. Each micro-operation is associated with a specific set of control lines which, when activated, causes that micro-operation to take place.

Since the number of instructions and control lines is often in the hundreds , the complexity of hardwired control units is very high. Thus, it is costly and difficult to design. Furthermore, the hardwired control unit is relatively inflexible because it is difficult to change the design, if one wishes to correct design error or modify the instruction set.

Microprogramming is a method of control unit design in which the control signal selection and sequencing information is stored in a ROM or RAM called a control memory CM. The control signals to be activated at any time are specified by a microinstruction, which is fetched from CM in much similar way an instruction is fetched from main memory. Each microinstruction also explicitly or implicitly specifies the next microinstruction to be used, thereby providing the necessary information for sequencing.

A sequence of one or more micro-operations designed to control specific operation, such as addition, multiplication is called a microprogram. The microprograms for all instructions are stored in the control memory.

The address where these microinstructions are stored in CM is generated by microprogram sequencer / microprogram controller. The microprogram sequencer generates the address for microinstruction according to the instruction stored in the IR.

12.6.2 Control Unit Organisation

The Figure12.14 shows the organisation of microprogrammed control unit. It consists of control memory, control address register, micro instruction register and micro program sequencer. The components of control unit work together as follows:

- The control address register holds the address of next microinstruction to be read.
- When address is available in control address register, the sequencer issues READ command to the control memory.
- After issue of READ command , the word from the addressed location is read into the microinstruction register.
- Now the content of the micro instruction register generates control signals and next address information for the sequencer.
- The sequencer loads a new address into the control address register based on the next address information.

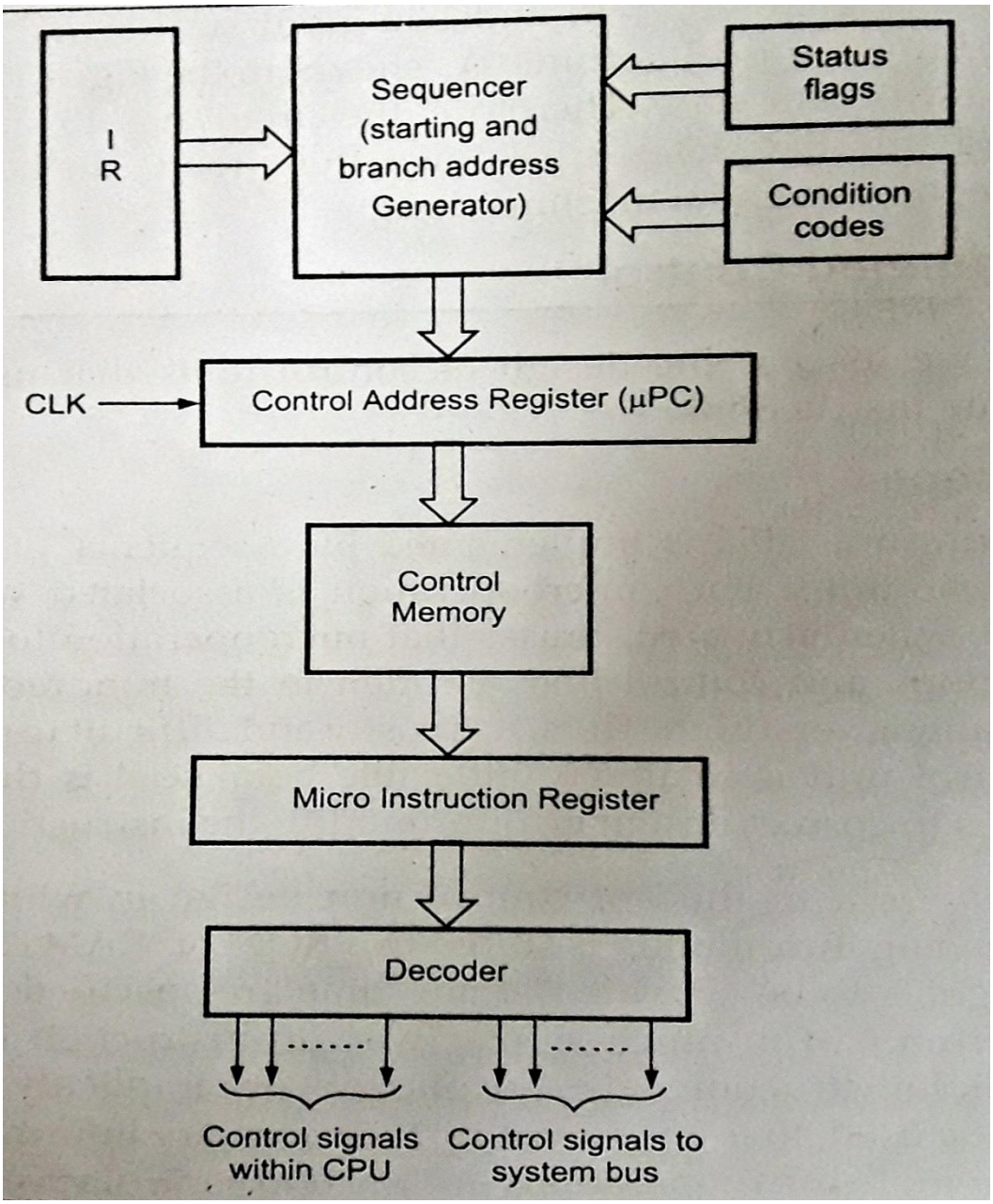


Figure 12.14 Micro programmed control unit

12.6.3 Advantages of Micro programmed Control

- It simplifies the design of control unit. Thus it is both, cheaper and less error prone to implement.
- Control functions are implemented in software rather than hardware.
- The design process is orderly and systematic.
- More flexible, can be changed to accommodate new system specifications or to correct the design errors quickly and cheaply.

- Complex functions such as floating point arithmetic can be realised efficiently.

12.6.4 Disadvantages of Micro programmed Control

- A micro programmed control unit is somewhat slower than the hardwired control unit, because time is required to access the microinstructions from CM.
- The flexibility is achieved at some extra hardware cost due to the control memory and its access circuitry.

Besides these disadvantages, the microprogramming is the dominant technique for implementing control units.

12.6.5 Alternatives to Increase Speed in Microprogramming

- Faster available control memory can be selected and use can be made of long microinstructions to simultaneously generate as many as control signals as possible.
- Pre fetching of micro instructions or a technique called pipelining can be used for microprogramming to increase speed of operation.

12.6.6 Microinstruction Sequencing

We have seen that microprogrammed control unit which performs two basic tasks

Microinstruction sequencing: Get the next microinstruction from the control memory.

Microinstruction execution: Generate the control signals needed to execute the microinstruction.

In this section, we discuss the design consideration and techniques to implement micro program sequencer.

12.6.7 Microinstruction Addressing and Timing

The two important factors must be considered while designing the micro program sequencer:

- The size of the microinstruction and
- The address generation time

The size of the microinstruction should be minimum so that the size of control memory required to store microinstructions is also less. This reduces the cost of control memory.

With less address generation time, microinstructions can be executed in less time , resulting better throughout.

During execution of a microprogram, the address of the next microinstruction to be executed has three sources:

- Determining by instruction register
- Next sequential Address
- Branch

Out of these three address sources, first occurs only once per instruction cycle. The second source is most commonly used. However , if we store separate microinstructions for each machine instruction , there will be large number of microinstructions. As a result, large space in CM is required to store these microinstructions .We know that, machine instructions involve several addressing modes and there can be many instructions and addressing mode combinations. A separate microinstructions for each of these combinations would produce considerable duplication of common microinstructions. We want to organise the microprograms such that they share as many microinstructions as possible.

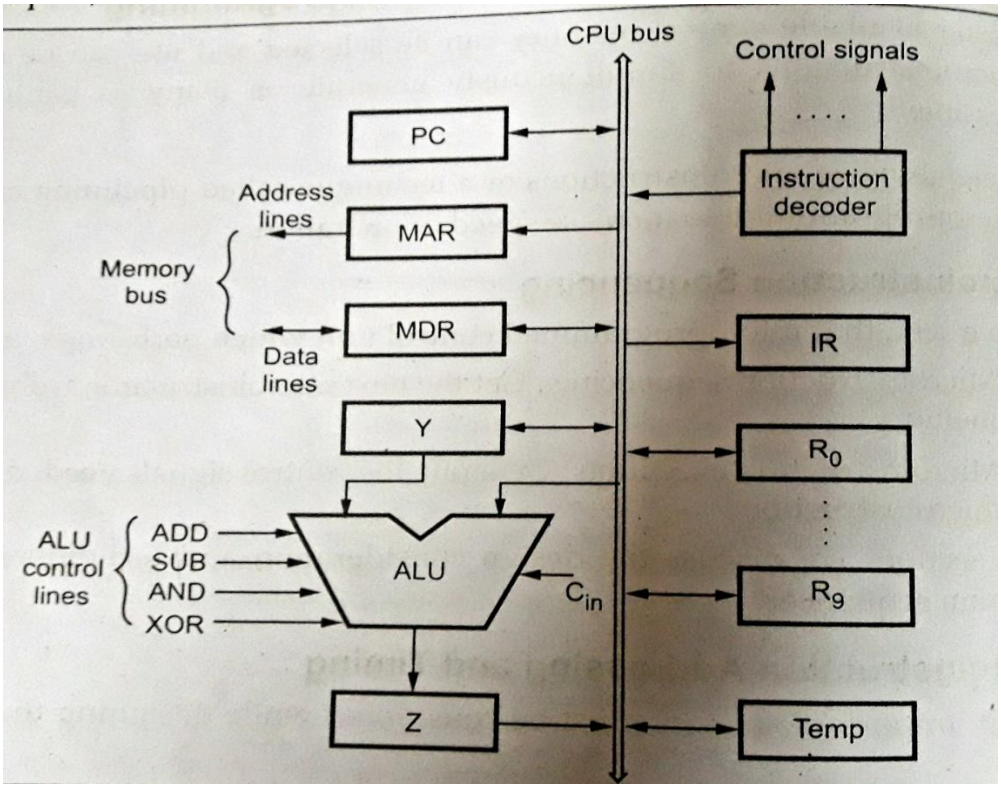


Figure 12.15 : CPU structure

This requires many branch microinstructions, both unconditional and conditional to transfer control among the various microinstructions. Thus it is important to design compact, time- efficient techniques for microinstruction branching.

Let us see how microinstructions can be shared using microinstruction branching.

Considering instruction ADD src, Rdst . The instruction adds the source operand to the contents of register Rdst and places the sum in Rdst, the destination register.

Let us assume that the source operand can be specified in the following addressing modes : Indexed , auto-increment, auto-decrement , register indirect and register direct. We now use this instruction in conjunction with the CPU structure shown in Figure to demonstrate a possible micro-programmed.

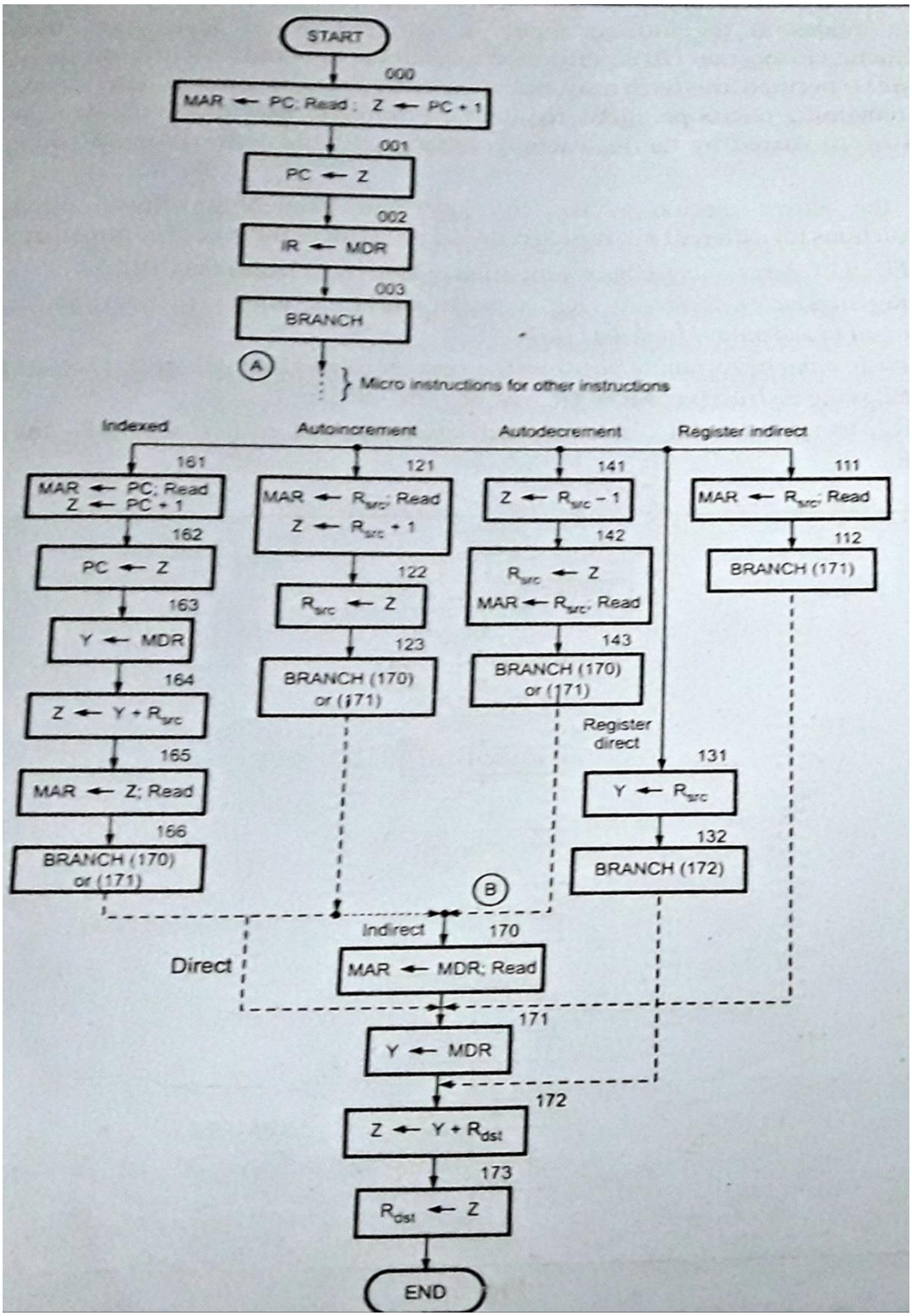


Figure 12.16 : Flowchart of a micro program for the ADD src , Rdst instruction

implementation. Figure12.16 shows a flowchart of a microprogram for ADD src, Rdst instruction

Each box in the flowchart corresponds to a microinstruction that controls the transfers and operations indicated within the box. The microinstruction is located at the address indicated by the number above the upper right- hand corner of the box .

During the execution of the microinstruction, the branching takes place at point A. The branching address is determined by the addressing mode used in the instruction.

At point B, it is necessary to choose between actions required by direct and indirect addressing modes. If the indirect mode is specified in the instruction, then the microinstruction in location 170 is performed to fetch the operand from the memory. If the direct mode is specified, this fetch must be bypassed by branching immediately to location 171.

The remaining microoperations required to complete execution of instruction are same and hence shared by the instructions having operand with different addressing modes.

From the above discussion we can say that branching allows sharing of microinstructions for different microprograms and it reduces the size of control memory.

12.7 Branching Techniques

Special Address Field:-

In this technique, the branch address is stored in the special address field within the microinstruction. The branch address is loaded in CM address register when a branch condition is satisfied. The special address field within the microinstruction increases the size of the microinstruction. The size of the microinstruction can be reduced by storing part of the address (low order bits) in the microinstruction. This restricts the range of branch instructions to a small region of the CM, and may therefore increase the difficulty of writing some microprograms.

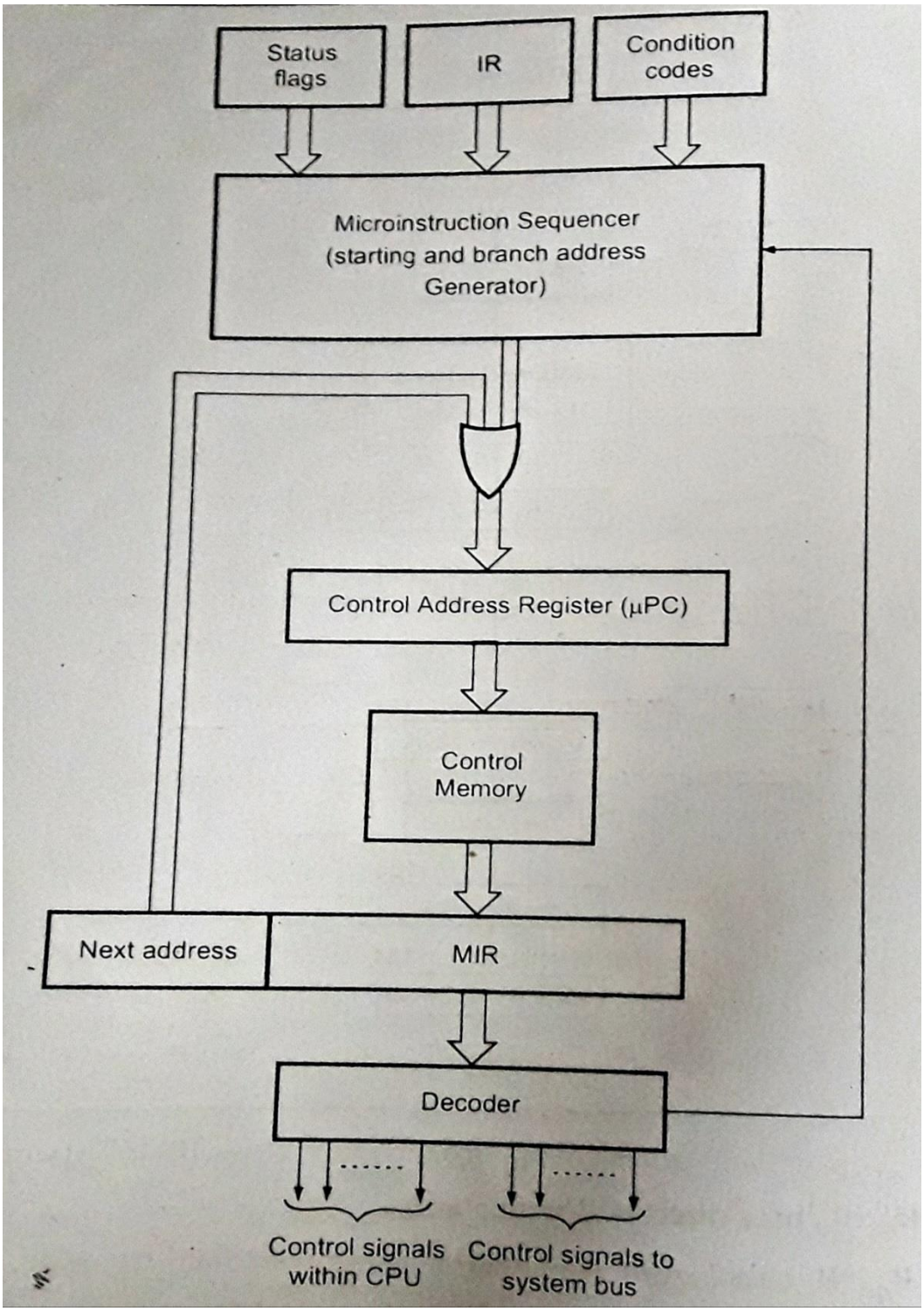


Figure 12.17 : Microinstruction sequencing organisation

Bit- ORing

In this technique, the branch address is determined by ORing particular bit or bits with the current address of the microinstruction. For example, if the current address

is 170 and the branch address is 172 then the branch address can be generated by ORing 02 (bit 1), with the current address.

Using Condition Variables

In this technique the condition variables are used to modify the contents of the CM address register directly, thus eliminating whole or in part the need for branch addresses in microinstructions. For example, let the condition variable CY indicate occurrence of CY=1 , and no carry when CY=0. Suppose that we want to execute a SKIP_ON_CARRY microinstruction. This can be done by logically connecting CY to the count enable input of μpc at on appropriate point in the microinstruction cycle. This allows the overflow condition to increment μpc an extra time, thus performing the desired skip operation.

Figure shows typical microinstruction sequencing organisation. As shown in the figure the next address is determined on the basis of the data in the IR, next address in the microinstruction, status flags, and condition codes.

Wide-Branch Addressing

Generating branch addresses becomes more difficult as the number of branches increases. In such situations Programmable Logic Array can be used to generate the required branch addresses. This simple and inexpensive way of generating branch addresses is known as wide-branch addressing. Here, the opcode of a machine instruction is translated into the starting address of the corresponding micro-routine. This is achieved by connecting the opcode bits of the instruction register as inputs to the PLA, which acts as a decoder. The output of the PLA is the address of the desired micro-routine.

12.8 Microinstruction Execution

A micro-programmed computer has two distinct levels of control.

i) The instruction level and

ii) The microinstruction level

At the instruction level, the CPU continuously executes instruction cycles that involve the following steps.

1. The CPU fetches an instruction from main memory , whose address is stored in the program counter PC.

2. The opcode part of I is placed in an instruction register IR, the operation specified by IR is then decoded and executed.

3. PC is altered to point to the next instruction to be fetched from M.

A similar sequence of operations takes place at the lower micro-instruction level, where the control- unit continuously executes microinstruction cycles as follows:

1. The addressing portion (micro-program sequencer) of the control-unit fetches a microinstruction MI from the control memory CM, whose address is stored in the micro-program counter μpc.

2. MI is loaded into the micro-instruction register MIR and is decoded to produce the required control signals.

3. μpc is altered to point the next microinstruction to b fetched from CM.

A microinstruction cycle can be executed faster than an instruction cycle, since micro-instructions are stored within the CPU, whereas instructions must be fetched from an external memory. Micro-instruction also normally requires less decoding than instructions.

12.9 Microinstruction Format

Having described a scheme for sequencing and execution of microinstructions, we now take a closer look at the format of individual microinstructions. A simple way to structure microinstructions is to assign one bit position to each control signal required in the CPU. However, this scheme has one serious drawback- assigning individual bits to each control signal results in long micro-instructions, because the number of required signals is usually large. Moreover, only a few bits are used in any given instruction.

The solution of this problem is to group the control signals.

12.9.1 Grouping of Control Signals

Grouping technique is used to reduce the number of bits in the micro-instruction .This technique is explained in the following section.

Let us consider single bus CPU having different control signals, as shown in figure

Gating signals: (IN and OUT Signals)

Control signals: Read, write, clear A, set carry in, continue operation, end , etc.

ALU Signals : Add , sub , etc . There are in all 46 signals and hence each microinstruction will have 46 bits. It is not at all necessary to use all 46 bits for every microinstruction because by using grouping of control signals we minimise number of bits for microinstruction.

Ways to reduce number of bits in microinstruction

1. Most signals are not needed simultaneously.
2. Many signals are mutually exclusive e.g. only one function of ALU can be activated at a time.
3. A source for data transfers must be unique which means that it should not be possible to get the contents of two different registers on to the bus at the same time.
4. Read and write signals to the memory can't be activated simultaneously.

This suggests the possibility of grouping the control signals so that all the signals that are mutually exclusive are placed in the same group. Thus a group can specify one micro-instruction at a time. So with these suggestions 46 control signals can be grouped in 7 different groups.

G1 (4 Bits) : IN grouping

00 00 No Transfer

0001 IR_{IN}

0010 PC_{IN}

0011 DR_{IN}

0100 AR_{IN}

0101 A_{IN}

0110 $Temp_{IN}$

0111 R_{1IN}

1001 R_{3IN}

1010 R_{4IN}

1011 SP_{IN}

G_2 (4 Bits) : OUT grouping

0000 No Transfer

0001 PC_{OUT}

0010 DR_{OUT}

0011 A_{OUT}

0100 R_{1OUT}

0101 R_{2OUT}

0110 R_{3OUT}

0111 R_{4OUT}

1000 SP_{OUT}

G_3 (4 Bits): ALU Functions

0000 ADD

0001 SUB

.

.

.

1111 XOR

16 ALU Functions

G_4 (2 Bits) : RD / WR Control Signals

00 No Action

01 Read

10 Write

G_5 (1 Bit) A Register

0- No action

1- Clear A

G_6 (1 Bit): Carry

0 – 1 Carry in to ALU =0

1 – Carry in to ALU =1

G7 (1 Bit): Operation

0 : Continue Operation

1 : End

The total number of grouping bits is 17. Therefore, we minimized 46 bits microinstruction to 17 bit microinstruction. Grouping of control signals result in relatively small increase in the required hardware as it becomes necessary to use decoding circuits to translate the bit patterns of each group into actual control signals. Since the number of bits required is less for microinstruction, less space is required for some instruction.

Two Techniques of Grouping of Control Signals

Highly encoded schemes that use compact codes to specify only a small number of control functions in each microinstruction are referred to as a vertical organisation. On the other hand, the minimally encoded scheme, in which resources can be controlled with a single instruction, is called a horizontal organisation.

12.9.2 Comparison Between Horizontal And Vertical Organisation

Horizontal	**Vertical**
1) Long formats	1) Short formats
2) Ability to express a high degree of parallelism	2) Limited ability to express parallel microopertions
3) Little encoding of the control information	3) Considerable encoding of the control information
4) Useful when higher operating speed is desired	4) Slower operating speeds

Advantages and Disadvantages

1. The horizontal organisation approach is suitable when operating speed of computer is a critical factor and where the machine structure allows parallel usage of a number of resources.
2. Vertical approach results in slower operations but less bits are required in the microinstruction.
3. In vertical approach the significant factor is the reduced requirement for the parallel hardware required to handle the execution of microinstruction.

12.9.3 Comparison Between Hardwired and Microprogrammed Control

Attribute	**Hardwired Control**	**Microprogrammed Control**
Speed	Fast	Slow
Control functions	Implemented in hardware	Implemented in software
Flexibility	Not flexible , to accommodate new system specifications or new instructions	More flexible, to accommodate new system specifications or new instructions redesign is required
Ability to handle large / complex instruction sets	Some what difficult	Easier
Ability to support operating systems and diagnostic features	Very difficult (unless anticipated during design)	Easy
Design process	Somewhat complicated	Orderly and systematic
Applications	Mostly RISC microprocessors	Mainframes, some microprocessors
Instruction set Size	Usually under 100 instructions	Usually over 100 instructions
ROM size	---------	2K to 10K by 20- 400 bits microinstructions
Chip area efficiency	Uses least area	Uses more area

12.9.4 Applications of Microprogramming

The various microprogramming applications are:

- **Realization of Computers:** Microprograms are used to implement the control unit of the computer.
- **Emulation:** Microprograms are used in emulation process. Emulation refers to the use of a microprogram on one machine to execute programs originally written for another. The emulation process is used as an aid to users in migrating from one computer to another.
- **Operating System Support:** Microprograms can be used to implement some of the primitives of the operating system. This simplifies the task of the operating system implementation. The implementation of primitives using microprogram also improves the performance of operating system.
- **High- Level Language Support:** In high- level language various sub functions and data types can be implemented using microprogramming. This makes it easy to compile the program into an efficient machine language form.
- **Micro diagnostics:** Microprogramming can be used to detection, isolation, monitoring and repair of system errors. These features are known as micro diagnostics and they significantly enhance the system maintenance facility.
- **User Tailoring:** By implementing control memory using RAM it is possible to tailor the machine to the desired application.

Questions:

1. What do you mean by micro operation?
2. Give the micro operations for fetch cycle.
3. Give the micro operations for interrupt cycle.
4. What are basic tasks of control unit ?
5. Draw the internal structure of CPU with control signals.
6. Write short note on hardwired control.
7. Draw and explain typical hardwired control unit.

8. Draw and explain the hardware required for gcd processor.
9. Explain the control unit design for gcd processor using state table method.
10. Explain the control unit design for gcd processor using one hot design method.
11. Write a short note on CPU control unit.
12. What is micro programming and micro programmed control unit ?
13. Write short note on micro programmed control unit.
14. Draw and explain the working of control unit.
15. List the advantages and disadvantages of micro programmed control .
16. Compare hardwired control unit and micro programmed control unit.
17. Write a short note on micro-instruction format .
18. What is micro program sequencing ?
19. Explain various branching techniques used in micro-programmed control unit?
20. Write short note on bit-ORing and wide –branch addressing.

Reference Books

1. Computer Organization & Architecture, William Stallings, 8e, Pearson Education.
2. Computer Architecture & Organization, John P. Hayes, 3e, Tata McGraw Hill.
3. Computer Organization, 5e, Carl Hamacher, Zconko Vranesic & Safwat Zaky,Tata McGraw-Hill.
4. Computer Architecture & Organization, Nicholas Carter, McGraw Hill
5. Computer System Architecture, M.Morris Mano ,Pearson Education.

❖❖❖

13

FUNDAMENTALS OF ADVANCED COMPUTER ARCHITECURE

Unit Structure

13.0 Objectives

After reading this unit,student will be able to

- Classify the different types of parallel systems.
- Compare and contrast between the types of Flynn's Taxonomy
- Outline the concept of Array processor parallel system
- Summarize the concept of NUMA computers

13.1 Introduction

1 As we have seen the architecture of computer system is structured upon vonNeumann model is depicted below , the model has component like Memory unit, Input unit, Arithmetic logic unit, output unit and Control unit. These five basic units are work in an coordinated manner to produce the accurate output.

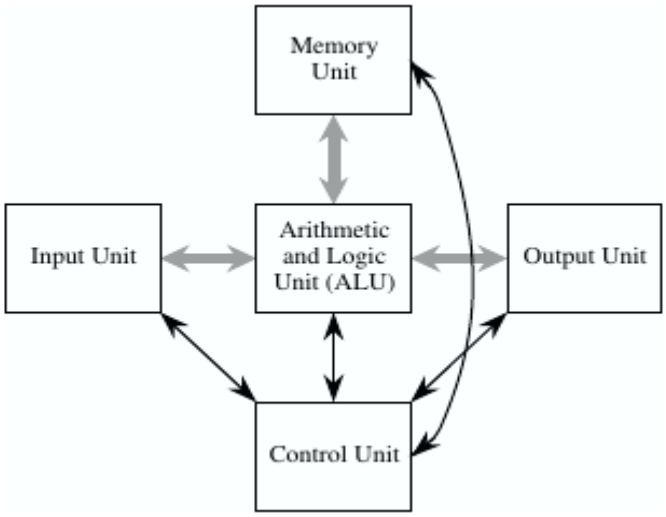


Figure 1 Architecture of von Neumann model.

2. So question over here, is if above diagram depicts about computer architecture then what is parallel computer architecture?, so developers follow this basic design to construct the parallel computer but double the units like memory unit, processors etc.

3. So Parallel Computer Architecture is defined as the way of efficiently utilizing of computer resouces like procesors ,memory, in order to inrease the performance of the system that was not possible with single processor system.

4. Parallel computing is defined as the simultaneous use of many computing resources at a single moment of time where a computation problem is broken in to a series of instructions, that series of instructions are executed on different processors simultaneously.

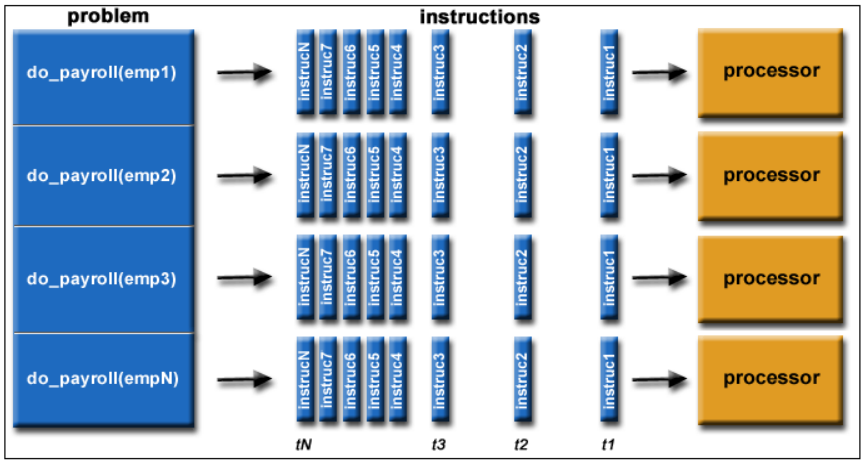


Figure 2 Example of Parallel computing

Here in above eg problem is broken down in to a number of instructions and simultaneously executing on different processors.

13.2 Classifcation of Parallel Systems

Classification based on

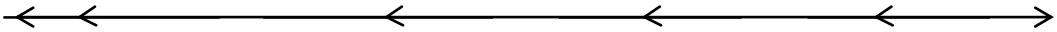


Figure 3 Classification of Parallel Systems

13.2.1 Instruction and Data Streams

1. The instruction cycle consists of a sequence of steps needed for the execution of an instruction in a program.
2. Program consist of intruction which in turn comprised of two parts one is opcode and another is operand. For eg instruction like c=A+B in that variables A and B are operand and '+' operator is nothing but opcode.
3. The operand part is divided in to two parts as a)addressing mode b)operand.
4. The addressing mode specifies the method of determining the addresses of the actual data on which operation is to be performed and the operand part is the actual data.
5. The control unit of the CPU of the computer fetches instructions in the program, one at a time.
6. The fetched instruction is then decoded by the decoder which is a part of the control unit and the processor executes that instruction.
7. Hence result is stored in Memory Buffer Register(MBR)(also called Memory Data Register).

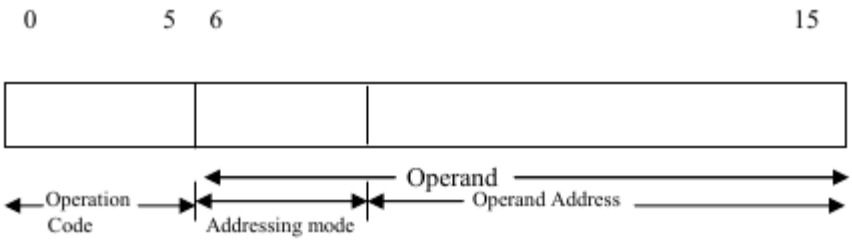


Figure 3 Opcode and Operand

8. The normal execution of a program is depicted in figure below. Flowchart explains that when the program is loaded in to the computational device, the address of the instruction is calculated, then CPU fetches and decodes the intruction and execute the instruction one at a time.

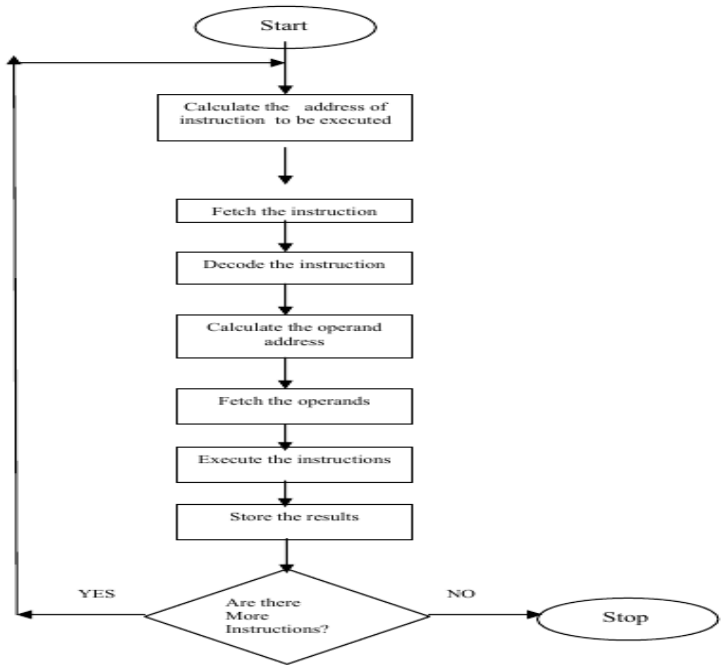


Figure 4 Instruction execution steps

9. Instruction Stream and data stream

9.1 The term 'stream' refers to a sequnce or flow of instruction or data operated by te computer.

9.2 In the complete cycle of instruction execution, a flow of instructions is passing from main memory to the CPU, so this flow of instruction is called as instruction stream.

9.3 On that basis, there is flow of operands between processor and memory and from memory to processor.

9.4 These two types of streams shown below

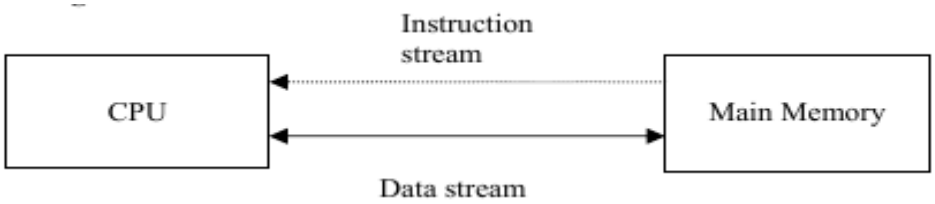


Figure 5 Instruction and data stream

9.5 The sequence of instructions executed by CPU forms the Instruction stream and sequence of data required for execution of instruction form the data stream.

13.2.2 Structure of the Computers

1. When multiprocessors communicate through the global shared memory moduls then this orgaisation is called shared memory computer or tightly coupled systems.

2. Similarly when every processor in a multiprocessor system, has its own local memory and the processors communicate via messages pass between their local memories, then this organisation is called Distributed memory computer or Loosely coupled system.

3. The processors and memory in both organisations are interconnected via an interconnection network.

4. This interconnection network may be different forms like crossbar, switch, multistage network.

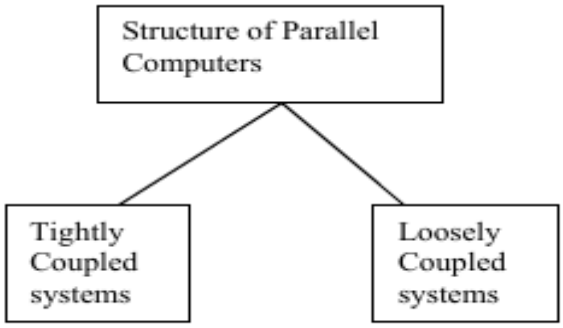


Figure 6 Structural classification

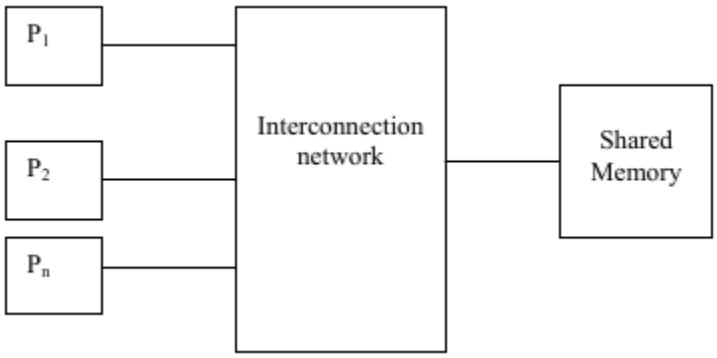


Figure 7 Tightly coupled system

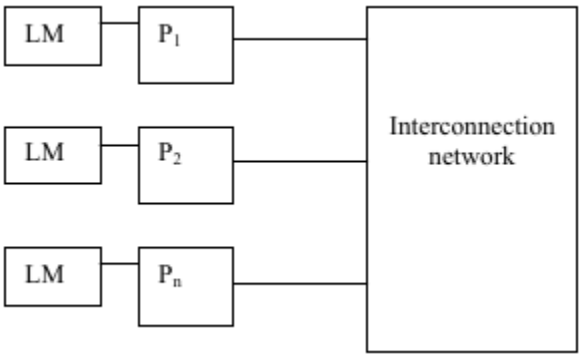


Figure 8 Loosely coupled system

5. Shared Memory system/ Tightly Coupled system

5.1 Shared memorymultiprocessors have the following characteristics: a)Every processor communicates through a shared global memory. b)For high speed real time processing, these systems are preferable as their throughput is high as compared toloosely coupld systems.

5.2 In tighly coupled ystem organization, multiple processors share a global main memory, which may have many modules as shown in figure below.

5.3 The inter-communication between processors, memory and other devices are implemented through various interconnection networks.

5.4 Processor-Memory Interconnection network(PMIN)

5.4.1 This is a switch that connects various processors to diferent memory modules.

5.5 Input-Output Processor Interconnction Network(IOPN)

5.5.1 Thi interconnection network is used for communication between processors and I/O channels.

5.6 Interrupt Signal Interconnction Network(ISIN)

5.6.1 When a processor wants to send an interruption to another processor, then this interrupt first goes to ISIN, through which is passed to the destination processor.

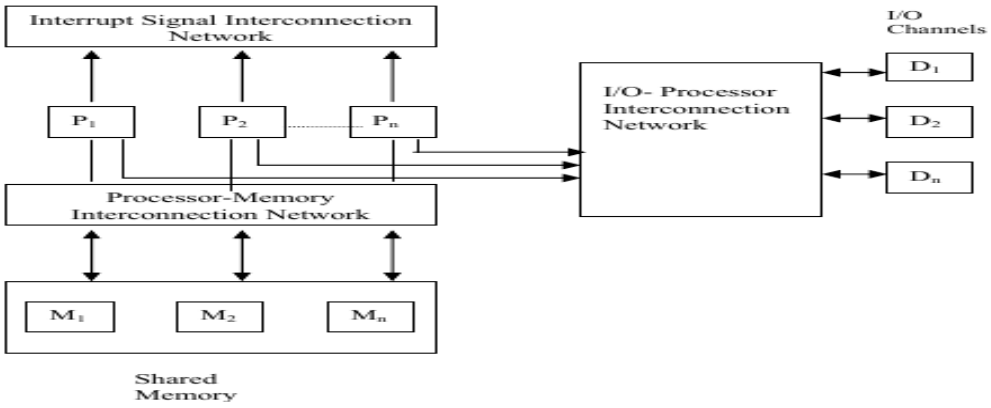


Figure 9 Tightly coupled system organization

6. Since, every reference to the memory in tightly coupled systems is via interconnection network, there is delay in executing the instruction. To reduce this delay, every processor may use cache memory for the frequent rference made by the processor as shown in figure below

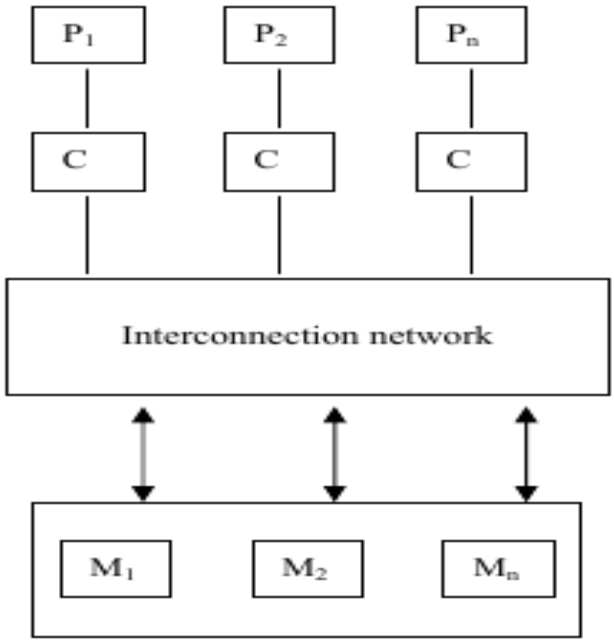


Figure 10 Tightly coupled system with cache memory

7. Loosely coupled systems

7.1 These systems do not share the global memory because shared memory concept has the disadvantage like due to the memory conflict problem which slows down the execution of the instructions.

7.2 Therefore to solve this problem, each processor in loosely coupled systems is having a large local memory(LM), which is not shared by any other processor.

7.3 The set of processor,memory and I/O devices makes a computer system, such a system is called as multi-computer system or called as distributed multicomputer systems.

7.4 In these systems, processors are communicate via message passing interconnection network.

7.5 Since local memories are accessible to the attached processor only, no processor can access remote memory, so called as no-remote memory access(NORMA) systems.

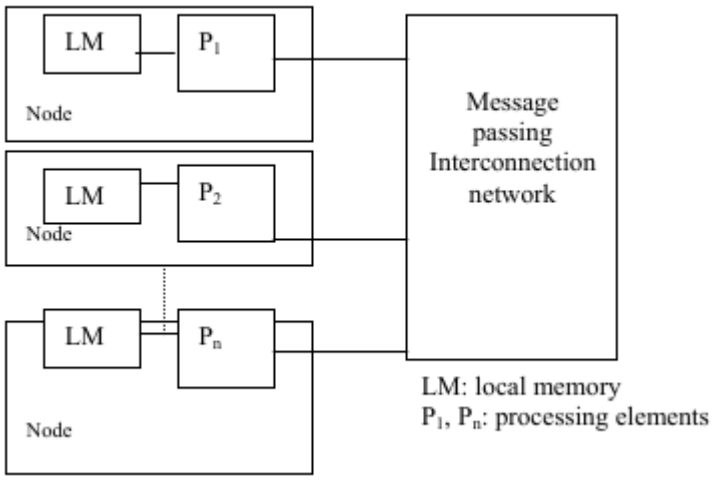


Figure 11 Loosely coupled system organization

13.2.3 Classification based on how the memory is accessed

1. The shared memory multiprocessor systems is further divided in to three modes based on the way the shared memory is accessed.

2. Uniform Memory Access Model(UMA)

   2.1 In this model, main memory is uniformly shared by all processors in multiprocessor systems and each processor has equal access time to shared memory.

   2.2 This model is used for time-sharing applications in a multi user environment.

3. Non-Uniform Memory Access Model (NUMA)

   3.1 In shared memory multiprocessor systems, local memories can be connected with every processor, so processor are accessing memory at an equal interval of time.

   3.2 But in this case, processor are not accessing local memories at an equal interval of time, so accessing is not uniform hence called as non-uniform memory access

4. Cache-Only Memory Access Model(COMA)

   4.1 Shared memory multiprocessor systems may use cache memories with every processor for reducing the execution time of an instruction.

   4.2 Thus in NUMA model, if we use cache memories instead of local memories, then it becomes COMA model.

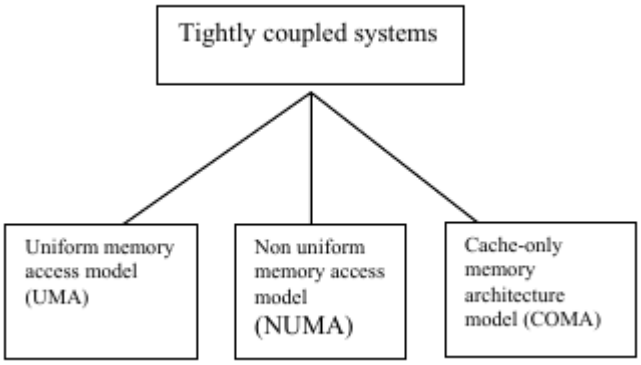


Figure 12 Modes of Tightly Coupled Systems

13.2.4 Classification based on Gran Size

1 Grain size or Granularity is a measure whic determines how much computation is involved in a process.

2 Grain size is dtermined by counting the number of instructions in a program segment.

3. There are three types of grain size availble as a)Fine Grain: This type contains approximately less than 20 instructions .b) Medium Gain: This type contains approximately less than 500 instructions.c) Coarse Grain: This type contains approximately greater than or equal to one thousand instrutions.

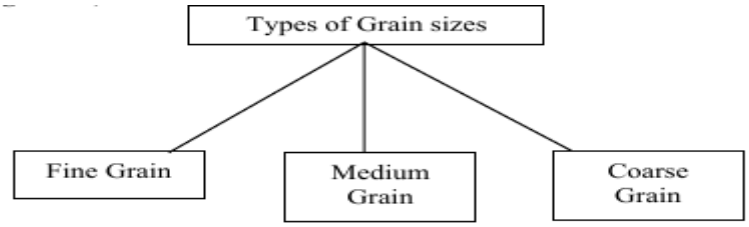


Figure 13 Types of Grain sizes

13.3 Flynn's Taxonomy

1. There are different ways to classify parallel computers. One of the more widely classifications used since 1966 is called Flynn's Taxonomy.

2. This classification was first studied and proposed by Michael Flynn in 1972. Flynn did not consider the machine architecture for classification of parallel computers, but he introduced the concept of instruction and data streams for categorizing of computers.

3. The matrix given below define the 4 possible classifications according to Flynn

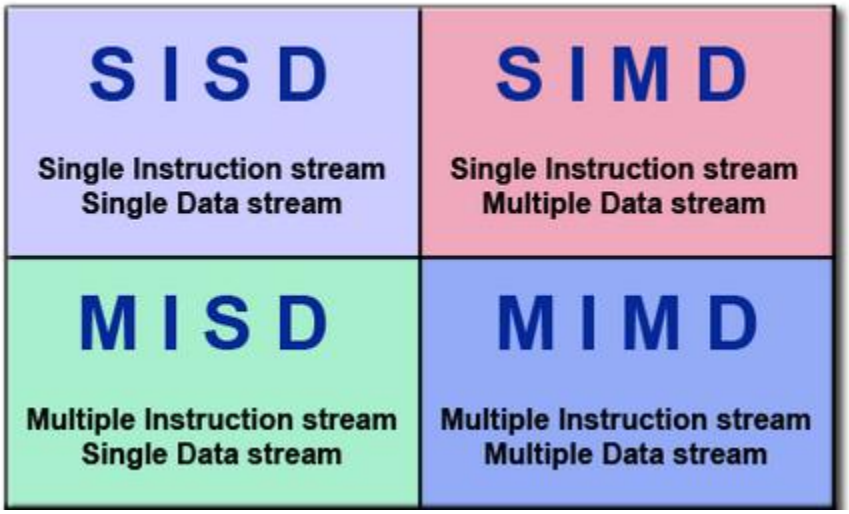


Figure 14 Classification of Flynn's Taxonomy

4 Single Instruction, Single Data(SISD)

4.1 A serial (non-parallel) computer.

4.2 Single Instruction: Only one instruction stream is being acted on by the CPU during any one clock cycle.

4.3 Single Data: Only one data stream is being used as input during any one clock cycle

4.4 This is the oldest type of computer . For eg older generation mainframes, minicomputers, workstations and single processor/core PCs.

4.5 Diagram given below is of SISD which depict that SISD has only one Instruction and Data stream.

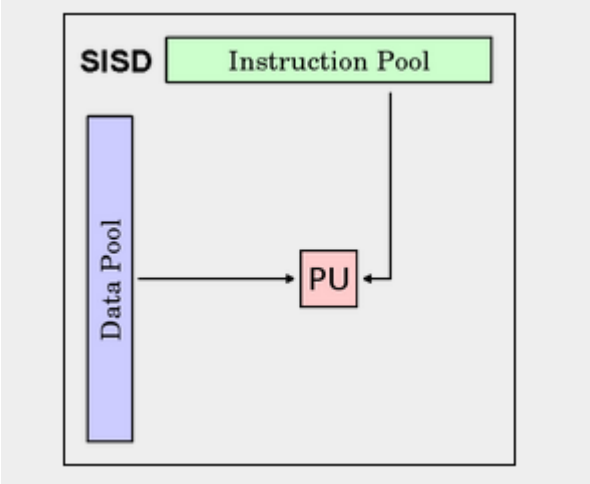
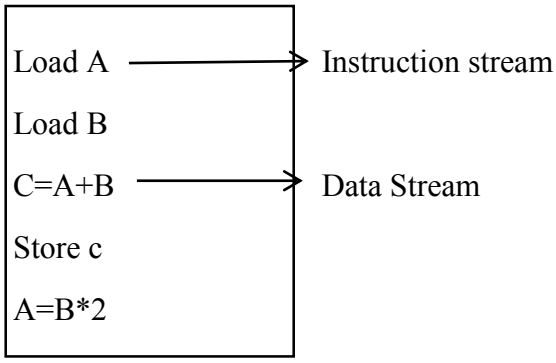


Figure 15: SISD

4.6 Example

Instruction and data stream like



Store A

4.7. Real time examples of SISD

5. Single Instruction, Multiple Data(SIMD)

5.1 This is a type of parallel computer

5.2 Single instruction: All processing unit can execute the same instruction at any given clock cycle.

5.3 Multipl Data: Each processing unit can operate on a different data element

5.4 Greatly used for specialized problems such as graphics/image processing.

5.5 Two varieties of it are Processor Arrays and vector pipelines

5.6 Examples

5.6.1 Processor Arrays: Thinkinmachines CM-2, Maspar MP-1 & Mp-2, LLIAC IV

5.6.2 Vector pipelines: IBM 9000, cray X-Mp etc

5.7 Most modern comouters, particularly those with graphics processor units(GPUS) emply SIMD instructions and execution units

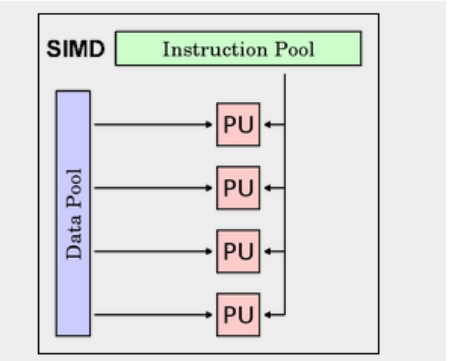


Figure 16 SIMD

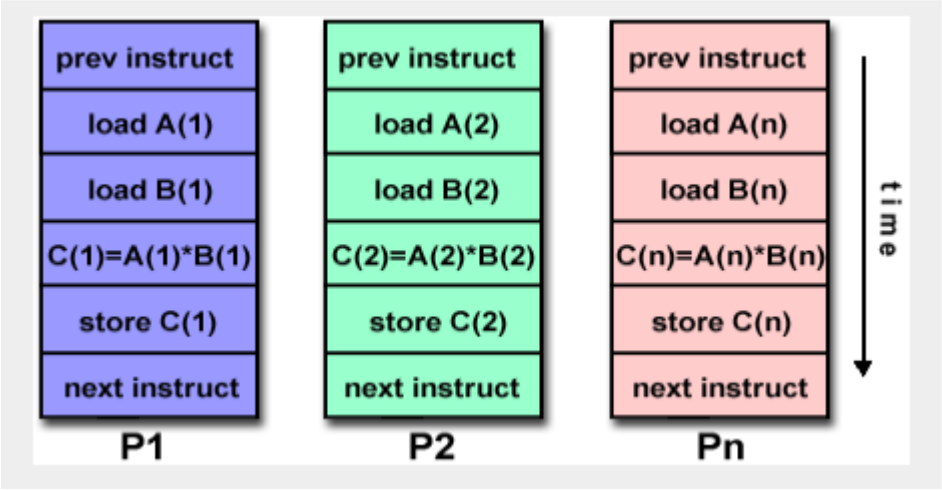


Figure 17 Example of SIMD

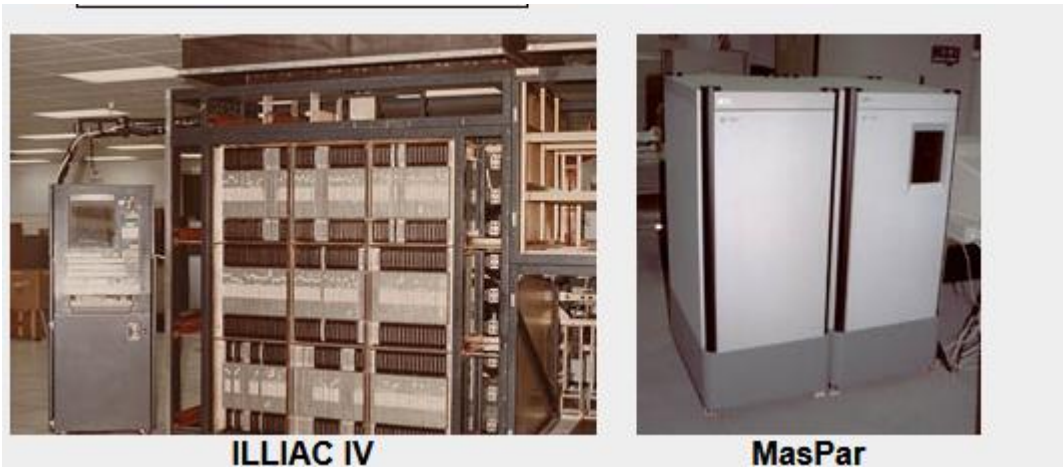


Figure 18 Real world applications of SIMD

6. Multiple Instruction, Single Data(MISD)

6.1 This is a type of parallel computer.

6.2 Multiple Instruction: Each processing unit operates on the data independently via separate instruction streams.

6.3 Single Data: A single data stream is fed in to multiple processing units.

6.4 For eg Multiple frequency filters operating on a single signal stream and multiple cryptography algorithms attempting to crack a single coded message.

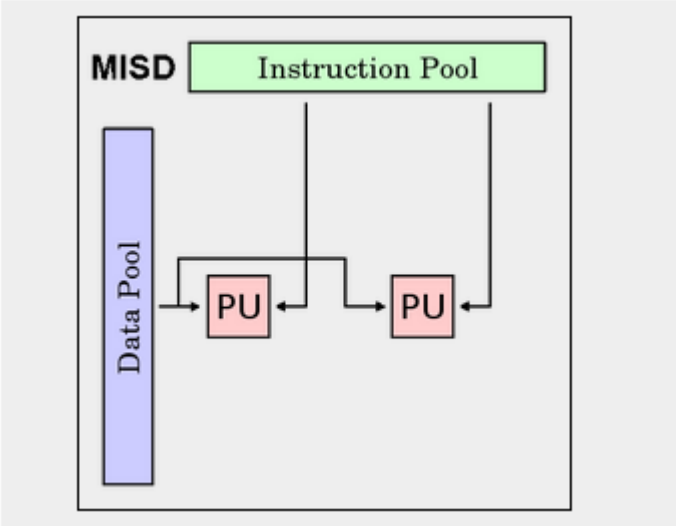


Figure 19 MISD Organization

The above figure explains that There is multiple instruction set which is connected to two processing units(PU) and only one data stream.

6.5 Example

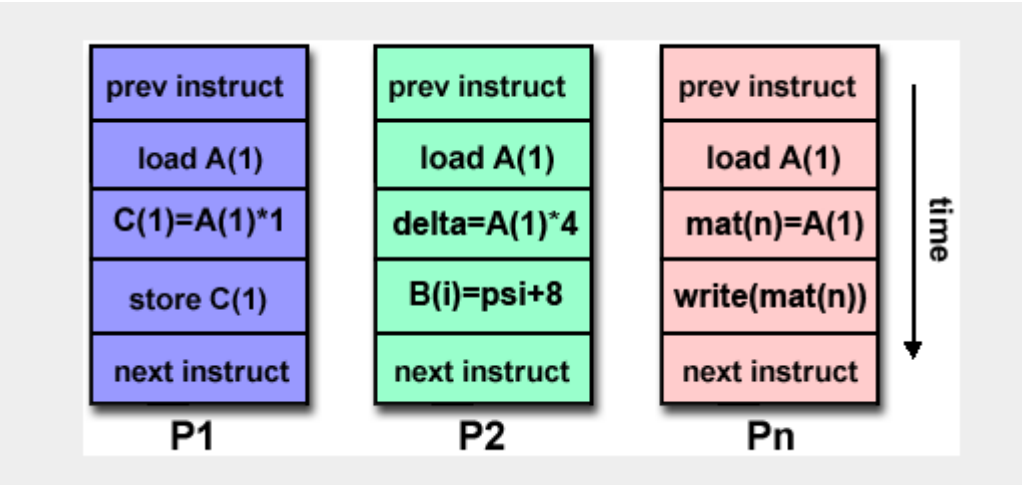


Figure 20 MISD example

The above figure explains that there is one data stream indicated as ‘A’ but there are multiple operations are performed on one data item as ‘A’.

7. Multiple Instruction, Multiple Data(MIMD)

7.1 This is also a type of Parallel computer

7.2 Multiple Instrution: Every processor may be executing a different instructon stream.

7.3 Multiple Data: Every processor may be working with a different data stream

7.4 Execution can be synchronous or asynchronous.

7.5 For eg Supercomputer are of MIMD type

7.6 Diagram given below depicts about MIMD architecture which signifies that Data pool and instruction pool are connected to different processing unit(PU)

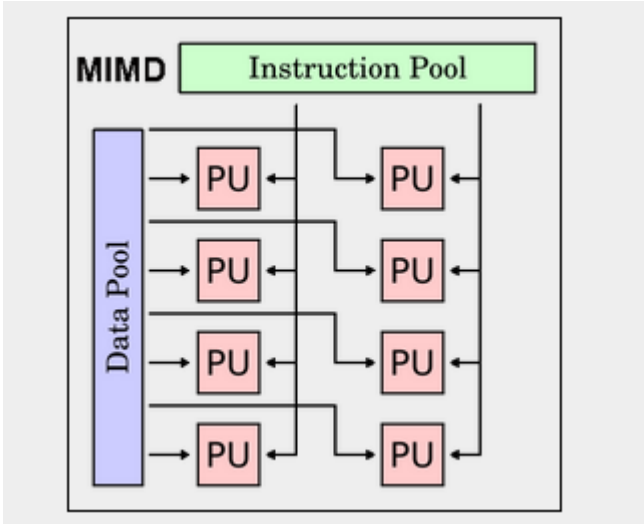


Figure 21 MIMD Organisation

7.7 Example given below indicates that there are more than one instruction and more than one data stream are simultaneously executing on processing unit

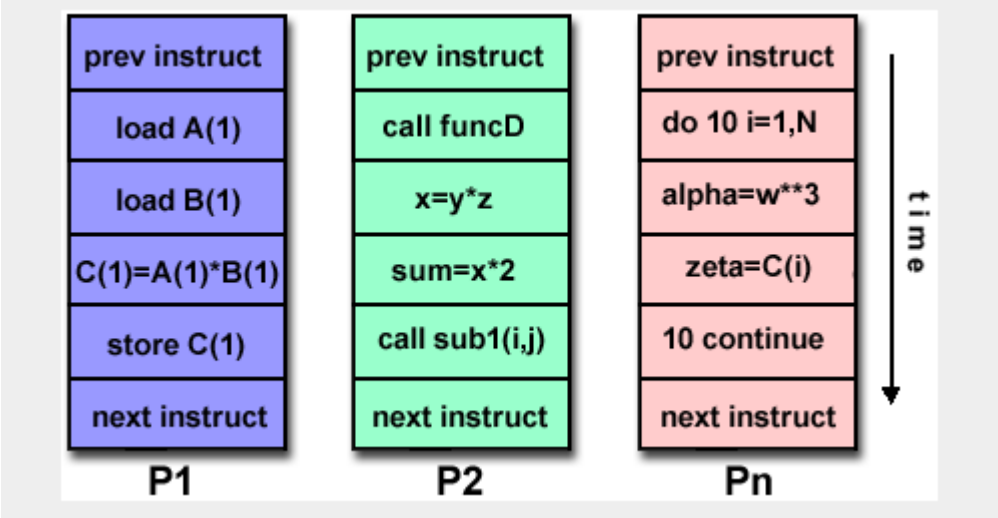


Figure 22 MIMD Example

13.4 Array Processors

1. They are also known as multiprocessors or vector processors.
2. They perform computations on large arrays of data.
3. Hence they are used to improve the performance of the computer.
4. There are basically two types of array processors as depicted below

Figure 23 Classification of Array processors

5. Attached Array processors

   5.1 An attached array processor is type of processor connected to a general purpose computer in order to enchance the performance of numerical computational tasks.

   5.2 Diagram given below depicted for Attached array processor which signifies that attached array processor is connected to a general purpose computer via an I/O interface where general purpose computer is connected to main memory and attached array processor is connected to local memory and local memory is communicated with main memory through high speed memory to memory bus.

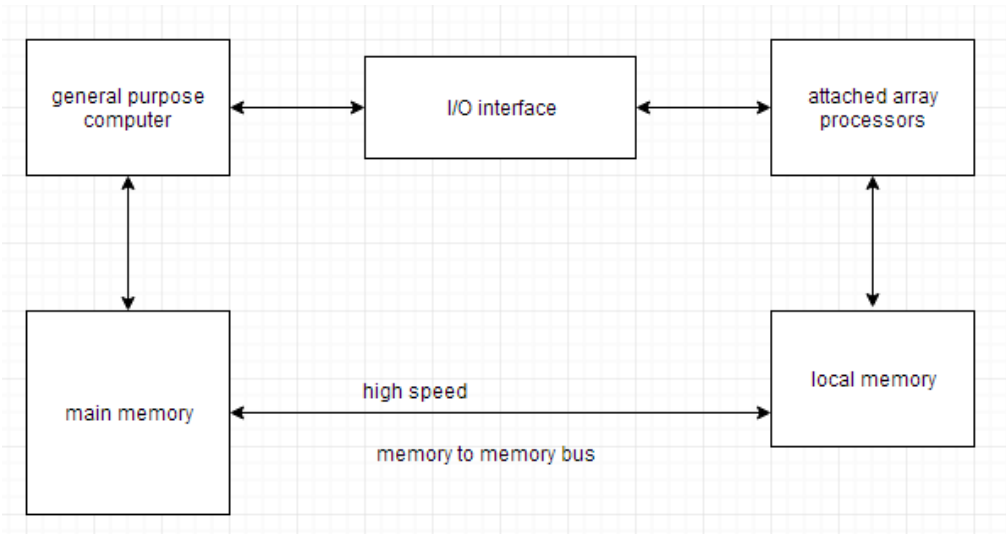


Figure 24 Attached Array Processor Orgaization

6. SIMD Array processor

1 SIMD I the organization of a single computer containing multiple processors operating in parallel.

2. The processing units are made to operate under the control of a common control unit thus able to provide a single instruction and multiple data streams.

3. Figure given below depicts the block diagram of an array processors which explains that it consist of a group of procesing elements(PES) whih has a local memory, ALU and registers. The work of master control unit is to decode the instructions and also decide which instruction is going to be executed.

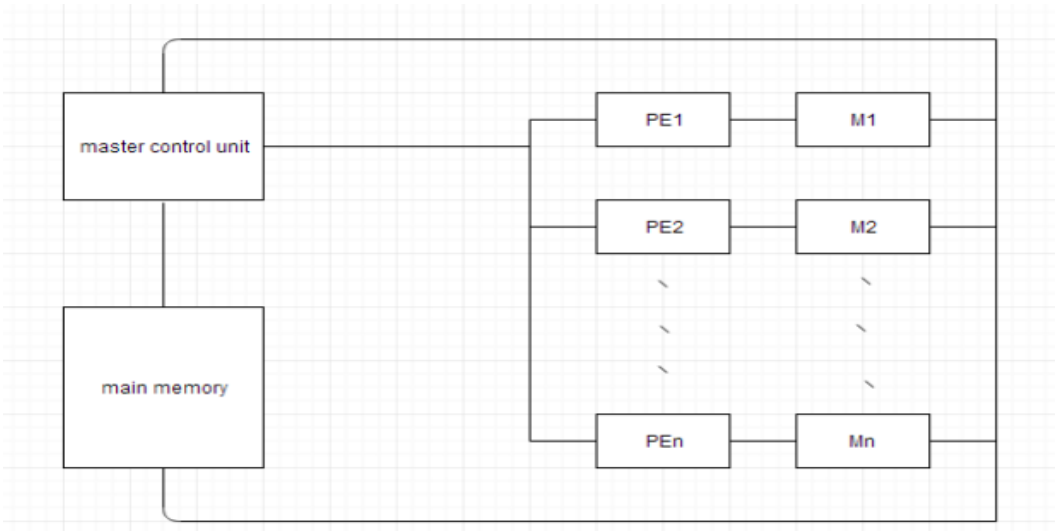


Figure 25 Array Processor

4. For eg ILLIAC IV computer are based on SIMD array processors. These computers are suitable for numerical problems that are represented in matrix form.

5. Advantages

5.1 It expands the overall instruction pocessing speed.

5.2 As array processor are working in an asnchronous mode so improves the overall capaciy of the system.

5.3 As each processing element of array processor has its own local memory which provides extra memory for systems with low memory.

13.5 Clusters Computers

1. Cluster iis just a bunch of interconnected machines, where the each machine decide which role to play and how to connect with other machines in an structured way.

2. The simplest approach is a symmetric cluster. Figure given below repreresents the architecture of symmetric cluster computer which has the nodes connected with each others and creating a network of machines with a servers attached to machines.

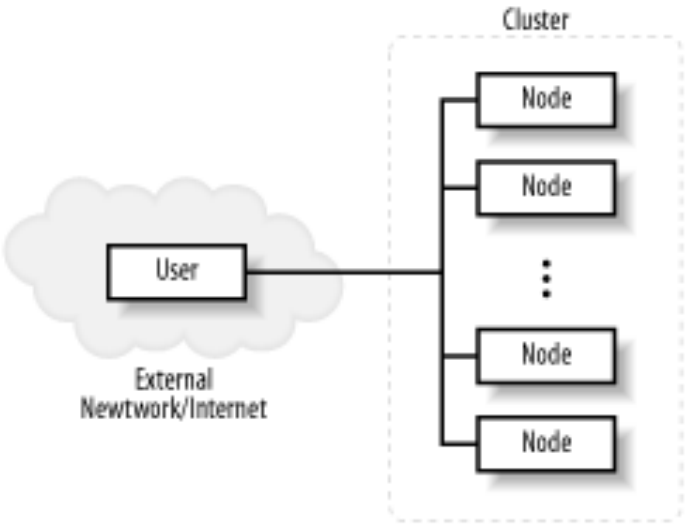


Figure 26 Symmetric clusters

3. Disadvantages of symmetric clusters are as follows

   3.1 Cluster management and securty can be more difficult.

   3.2 Workload distribution can become a problem, making it more difficult to achieve optimal performance.

4. For dedicated cluster, an asymmetric architecture is more common.

   4.1 Figure given below shows the architecture of asymmetric cluster which signifies that between user and cluster nodes there is a head node between them which is nothing but gateway to provide high level of security between the remaining nodes and the users.

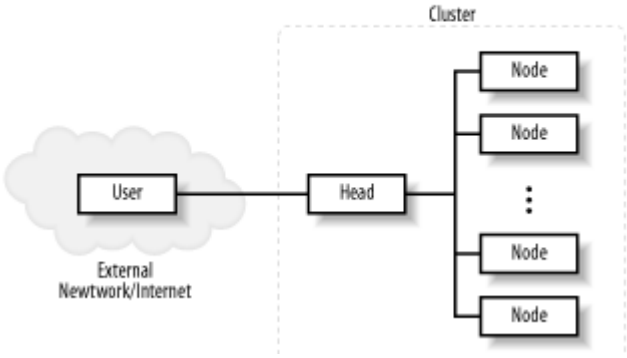


Figure 27 Asymmetric clusters

4.2 Disadvantage

4.2.1 Cluster head requires a high computation node to improve the performance of the system.

4.2.2 As the cluster size expands diffiult to manage the network only with one head / gateway node.

5. To solve issue of asymmetric cluster , an additional servers are required to monitor the health of cluster like NFS server, I/O server etc. Hence this is called as expandable clusters.

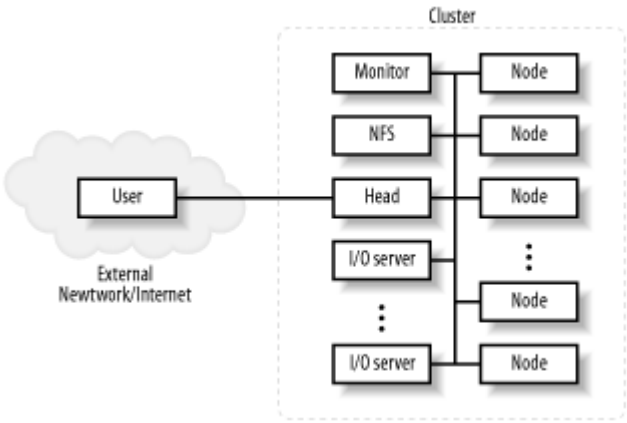


Figure 28 Expanded clusters

13.6 NUMA Computers

1. In Non uniform Memory Access(NUMA) each processor has its own local memory.

2. Intel's Quick path Interconnect(QPI) procesor follows NUMA architecture which has built-in memory controller.

3. A processor usually uses its local memory to store the data required for its processing. Accessing a local memory has least latency.

4. NUMA provides better scalability than UMA when number of processors is very large.

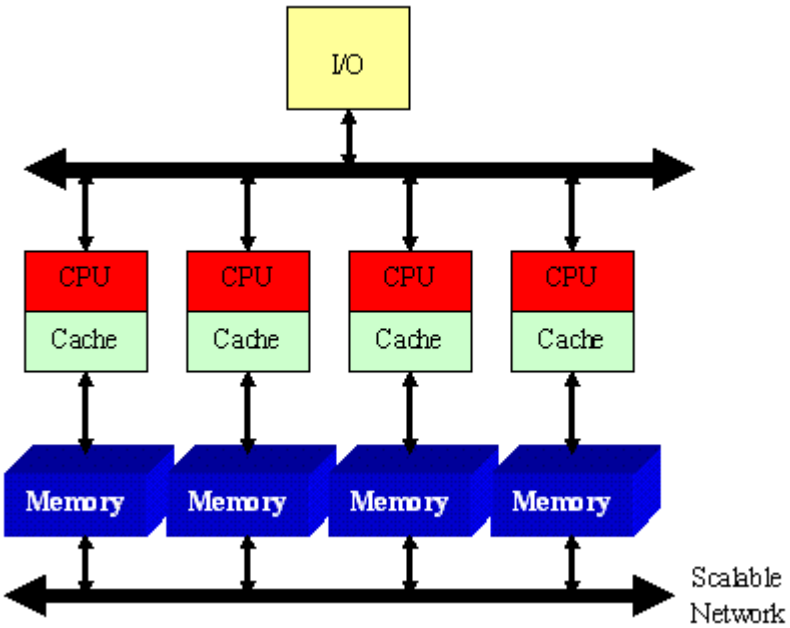


Figure 29 Non UniformMemory Access

6. The hardware trend is to use NUMA systems with several NUMA nodes as shown in given figure below, which explains that NUMA has many processors with shared memory. Multiple NUMA nodes can be bunched together to form a SMP.

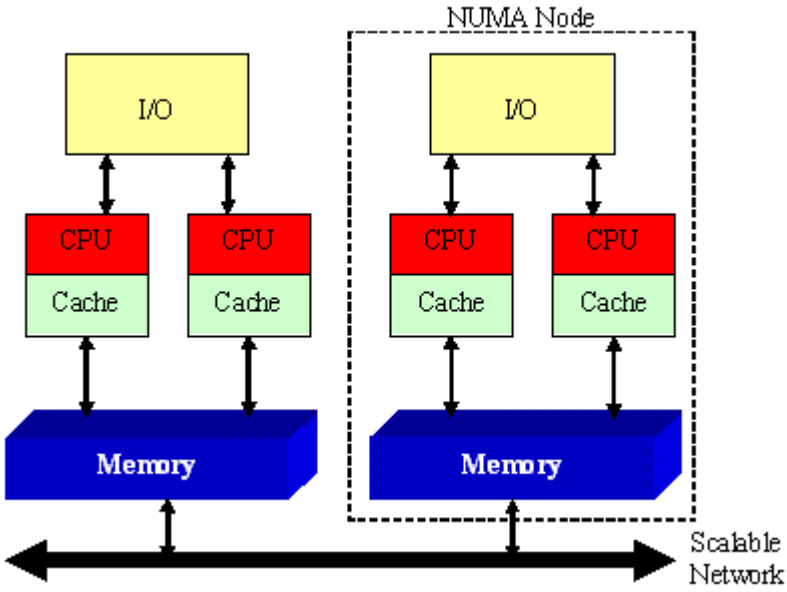


Figure 30 NUMA SMP

Questions to be Revised

Q1 What are various criteria for classification of parallel computers?

Q2 Define Instructions and data streams

Q3 Define Loosely coupled systems and tightly coupled systems

Q4 Differentiate between UMA,NUMA and COMA

Q5 Describe in short about Array Proccessors

Q6 Describe in short about NUMA arcitecture

Q7 Clssify the Flynn's Taxonomy

❖❖❖

14

MULTIPROCESSOR AND MULTICORE COMPUTERS

Unit Structure

14.0 Objectives

At the end of this unit, Student will be able to

- Construct the structure of multiprocessor system.
- Describe the interconnection network of multiprocessor system.
- Build the architecture of multi-core computers.
- Illustrate the performance,organization of multi-core computer.

14.1 Introduction

1. A multi-processor is a computer system in which two or more CPUs share full access to a common RAM.
2. The problem with this type of system is as memory over here is sharable between different CPU's. So if one CPU has written some value in to the

memory and if that CPU want to access that value from memory then its not able to get back the original value as other CPU might changed the value.

3. The above problem can be solved if proper organization of all the CPU's has done by using the property called Inter-process communication where one CPU writes some data in to memory and another one reads the data out.

4. Multiprocessor system can handle system calls, memory management, process synchronization, resource management.

5. A Multi-core computer, combines two or more processors on a single computer chip.

6. The main variables in a multicore organization are the number of processors on the chip,the number of levels of cache memory and the extend to which cache memory is shared.

7. A multicore computer commonly known as a chip based multiprocessor which combines two or more processors (called cores) on a single piece of die called a (silicon).

8. Each core of an independent processor consist of following component

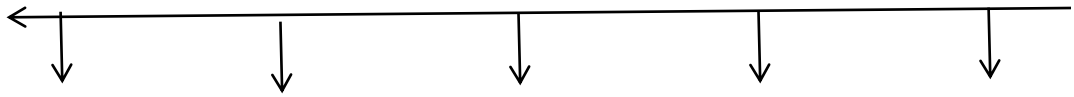


Fig 1 Components of an Independent processor

14.2 Multiprocessor

1. Multiprocessor hardware and software decides the structure and interconnection between the components of multiprocessor.

2. In hardware part of multiprocessor comprises of different interconnection network like crossbar switches,multistage switching networks etc.

3. Whereas software part consist of different types of operating systems in multiprocessor.

14.2.1 Structure & Interconnection network

This topic will going to cover points as

Table 1 Structure and Interconnection network to be cover under this heading

Sr.No	Structure to be learn
1	UMA bus based structure
2	UMA multiprocessors using crossbar switches
3	UMA multiprocessors using multistage switching networks
4	NUMA Multiprocessors

1. Uniform Memory Access (UMA) Bus based architecture.

1. With multiprocessor operating systems various organizations are possible, but simplest multiprocessor are based on a single bus as shown in figure below which is nothing but **Uniform Memory Access (UMA)** Bus based architecture.

2. Two or more CPUs and one or more memory modules all use the same bus for communication.

3. When a CPU wants to read a memory word, it first checks to see if the bus is busy. If the bus is idle then the CPU signals the bus with the address of the word it want to access and in turn memory put word on bus and send to CPU.

4. If the bus is busy, then CPU will wait until bus becomes free. But the problem with this architecture is then it works well for two or more CPU, but when more number (32-64) of CPUs is connected to Bus, its performance degrades.

5. The solution to above problem is cache is connected to every CPU connected to bus, depicted in figure given below. Advantage of using cache is, it reduces traffic on the bus as many read operation can done through cache.

6. In general, caching is not done on an individual word basis but on the basis of 32 or 64 byte blocks. When a word is referenced, its entire block is fetched in to the cache of the CPU asking for a byte to read.

7. One more solution for single bus based problem is also possible that along with cache, private memory is connected to CPU to faster access the data as depicted in figure below. To use this configuration optimally, the compiler

should place all the programs,text,strings,constants and other read-only data,stacks and local variables in to the private memories.

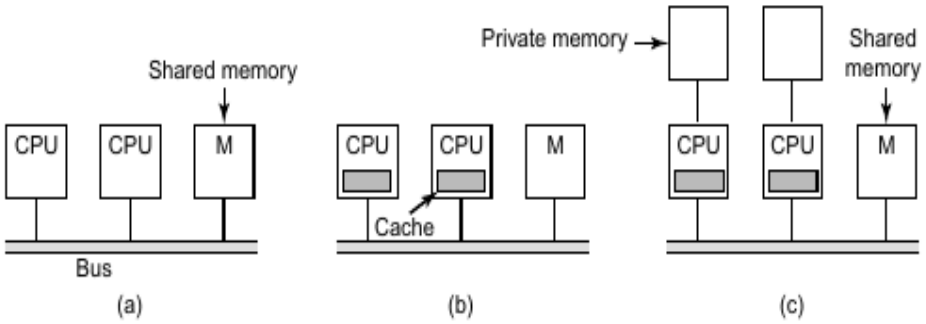


Figure 2 - Three bus-based multiprocessors a)Without caching b)With caching c)Wih caching and private memories.

2. UMA multiprocessors using Crossbar switches

1. If a wider network of CPU is required not just connection of 32-64 CPU on a single bus, then the development of UMA using crossbar switches had come.

2. Crossbar switches have been used for decades within telephone switching exchanges to connect a group of incoming lines to a set of outgoing lines in an arbitrary way.

3. Crossbar switches has architecture which symbolises two way connection consist of Row(Horizontal line) used for incoming purpose and Column(Vertical line) used for outgoing purpose commonly known as crosspoint.

4. A crosspoint is a small switch that can be electrically opened or closed, depending on whether the horizontal and vertical lines are to be connected or not.

5. Figure depicts below the, 8*8 cross-point switch with open and close crosspoint. This figure explains that two types of circles one dark circle which indicates closed switch and darkless circle indicates open switch. So here three crosspoints closed simultaneously as (010,000),(101,101) and (110,010)

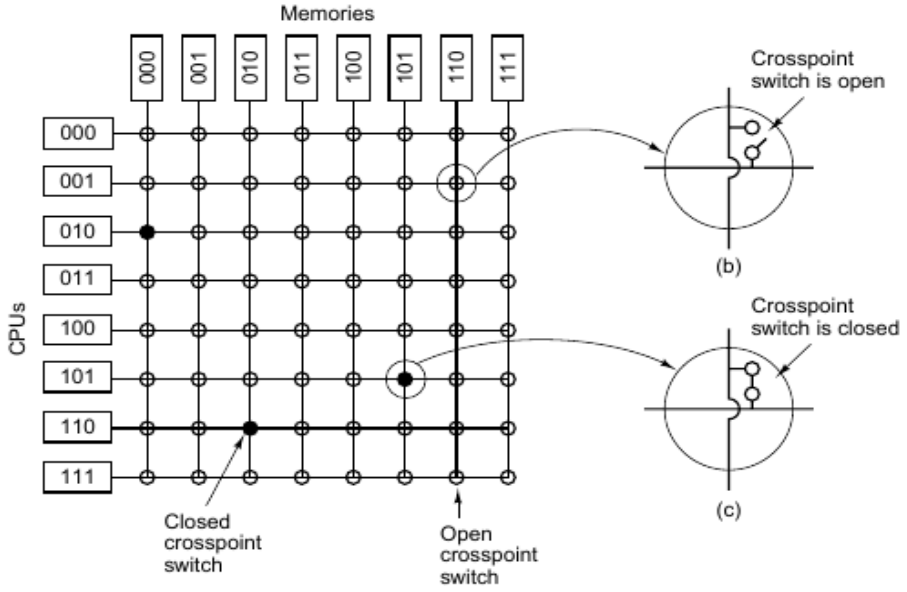


Figure 3 An 8*8 crossbar switch with open and closed crosspoint

6. One of the nicest properties of the crossbar switch is that it is a nonblocking network, meaning that no CPU is ever denied the connection it needs because some crosspoint or line is already occupied (assuming the memory module itself is available).

7. Also no advance planning is needed, that is if seven arbitrary connection are alreay set up, it is always possible to connect the remaining CPU to the remaining memory.

8. One of the disadvantage of the crossbar switch is the fact that the number of crosspoints grows as $n^{2.}$. So with 1000 CPUs and 1000 memory modules we need a million crosspoints. Such a large crossbar switch is not feasible.

3. UMA Multiprocessors Using Multistage Switching Networks

1. This multiprocessor design consist of 2 switches for input an 2 switches for output where messages arriving on either input line can be switched to either output line. The designing is shown in figure given below(a).

2. A message format consist of four field described below as follows

Table 2 Message format description

Sr.No	Field Name	Field Description
1	Module Field	Indicates which memory to use
2	Address field	Specifies an address within a module
3	Opcode	It gives the operation such as read or write
4	Value(optional field)	It consist of exact value written to a particular memory.

3. Switches can be connected using Omega network. For eg if we connect 8 CPUs to eight memories so there is only requirement of 12 switches as to connect n CPUs and n memories only there is need of $\log 2^n$ stages, which is better than n^2

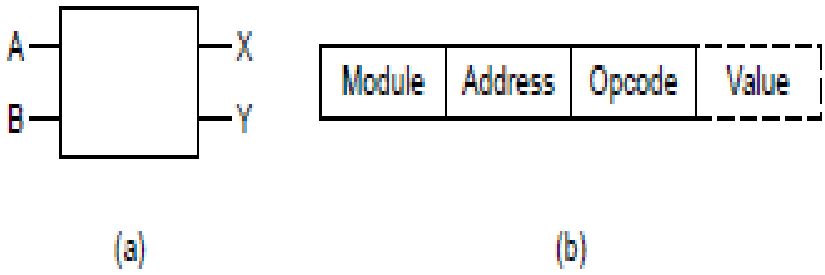


Figure 4 a) A 2X2 Switch b) A Message Format

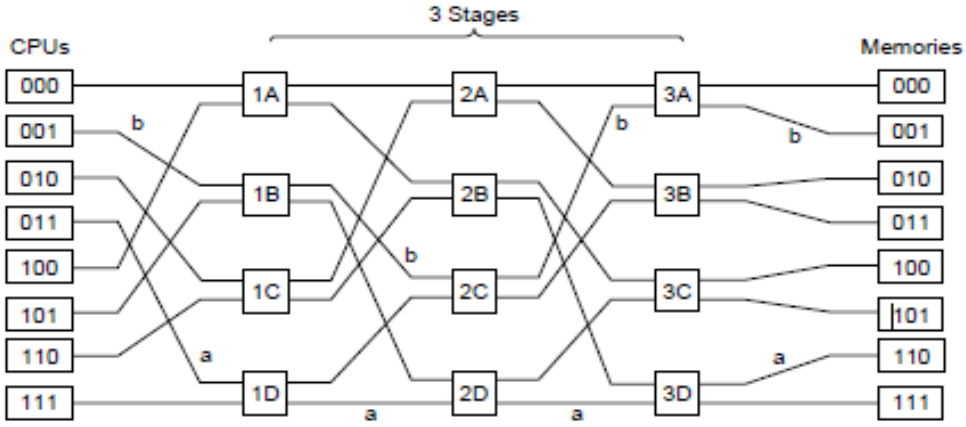


Figure 5 An Omega Switching network

4. The Omega netowork works like as described here suppose that CPU 011 wants to read a word from memory module 110. The CPU sends a READ message to switch to cell 110 in the module field. If the leftmost bit is 1 then message is routed via the lower output and if the left most bit is 0 then the message is routed via upper output.

5. All the second stage switches, will route the message to third stage via the lower output and finally the message is reached at destination place as 110.

6. As the messages moves through the switching network, the bits at the left hand of the module number are no longer needed. So they can be used for recording the number of lines comes as incoming line.

7. At the same time if request comes for same memory module on which currently word writing operation is going on , then omega network has a

facility of blocking network , that is only one request is handled at one time and one has to wait for sometime till the request is not finished.

8. A memory system in which consecutive words are in different modules is said to be interleaved. Interleaved memories maximize parallelism because most memory references are to consecutive addresses. It is also possible to design switching networks that are nonblocking and which offer multiple paths from each CPU to each memory module, to spread the traffic better.

4. NUMA Multiprocessors

1. We have studied till yet UMA based multiprocessor with swicthing technology but the crossbar or switched methodology need a lot of expensive hardware.

2. So alternate solution for accessing memory module at same time is nothing but the concept of NUMA(Non uniform memory Access) is introduced.

3. NUMA machines have three key characterisitcs that all of them possess and which together distinguish them from other multiprocessors

   3.1 There is a single address space visible to all CPUs.

   3.2 Access to remote memory is via LOAD and STORE instruction.

   3.3 Access to remote memory is slower than access to local memory.

4. When the access time to remote memory is not known as there is no caching available then that system is called as No Cache-NUMA(NC-NUMA).

5. When the cache system is available in NUMA for remote access to a memory then that system is called Cache coherent NUMA.

6. The most popular approach for building large CC-NUMA multiprocessors currently available is the directory based multiprocessor. The idea is to maintain a database telling where each cache line is and what its status is.

7. When a cache line is referred, the database is queried to find out whether it is clean or dirty (modified).

8. The structure of NUMA is depicted in figure given below where nodes are connected by an interconnection network.

9. Each node also holds the directory entries for the 64 byte cache lines. For eg Suppose a LOAD instruction from CPU 20 that references a cached line. First

the CPU issuing the instruction presents it to its MMU, which translates it to a physical address. The MMU splits this address in to the three parts shown in Fig below. The MMU sees that the memory with referenced is from 36, not 20, so it sends a request message through the interconnection network to the node 36.

10. When the request arrives at node 36 over the interconnection network,it is routed to the directory hardware. So the hardware check its index table and search for cache line 4 and the data of node 36 , is now cached at node 20 instead of line 4 and node 36.

11. Disadvantage of this design is

11.1 That a line can be cached at only one node. To allow lines to be cached at multiple nodes, need some way of locating all of them, for eg to invalidate or update them on a write.

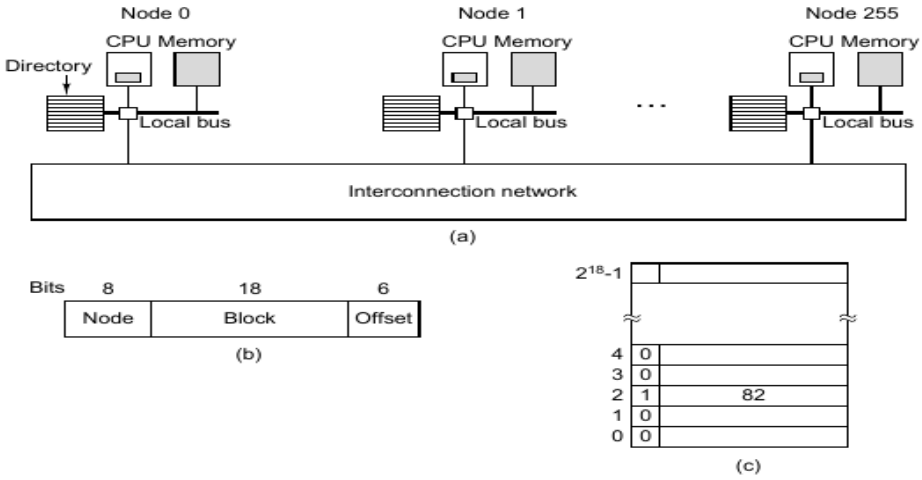


Fig 6 Interconnection network of Multiprocessor

14.3 Multi-Core Computers

14.3.1 Introduction

1. A multicore computer, combines two or more processors on a single computer chip.

2. The use of more complex single-processor chips has reached a limit due to hardware performance issues.

3. The multicore architecture poses challenges to software developers to exploit the capability for multithreading across multiple cores.

4. The main variables in a multicore organization are the number of processors on the chip, the number of evels of cache memory and the extend to which cache memory is shared.

5. A multicore computer also known as a chip multiprocessor combines two or more processors (called cores) on a single piece of silicon(called a die).

6. Each core consists of all the components of an independent processor, such as registers,ALU, pipeline hardware and control unit, L1 instruction and data caches.

7. In addition to the multiple cores, contemporary multicore chips also include L2 cache and in some cases L3 cache.

14.3.2 Organization

1. Figure given below depicts an organization of multi-core computer chips.

2. In this, organization, the only on-chip cache is L1 cache, each core having its own dedicated L1 cache.

3. The L1 cache is divided in to instruction and data caches.

4. An example of this organization is the ARM11 MpCore.

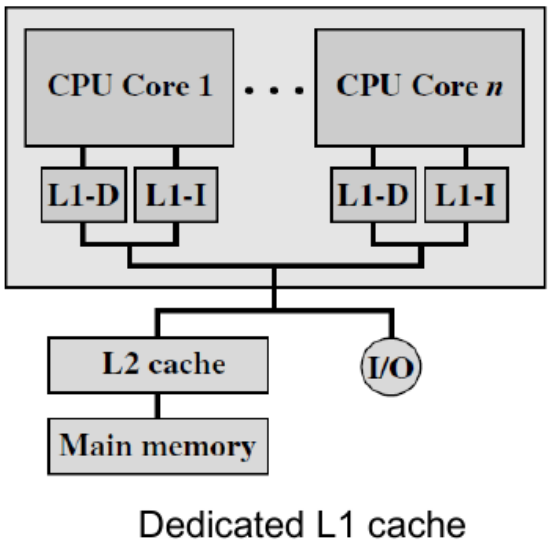


Figure 7

5. The organization is also one in which there is no on-chip cache sharing.

6. In this, there is enough area available on the chip to allow for L2 cache.

7. An example of this organization is the AMD opteron.

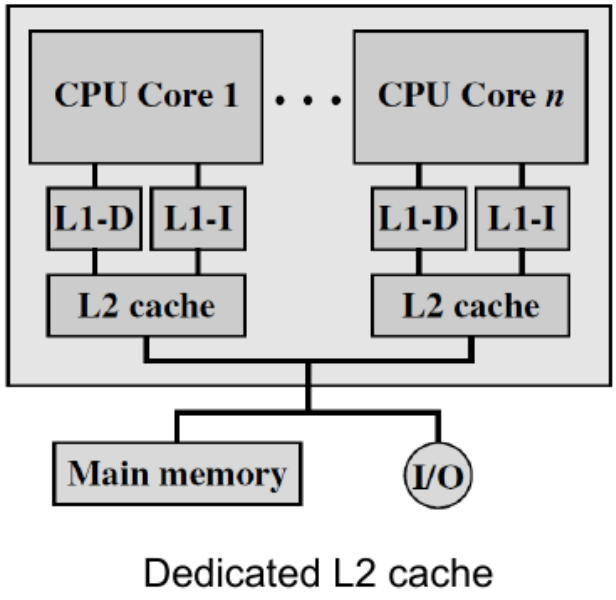


Figure 8

9. Figure given below shows allocation of chip space to memory, but with the use of a shared L2 cache.

10. The intel core duo has this organization.

11. The amount of cache memory available on the chip continues to grow, performance considerations dictate splitting off a separate, shared L3 cache, with dedicated L1 and L2 caches for each core processor.

12. The intel core i7 is an example of this organization

13. Advantages of shared L2/L3 cache on the chip

    13.1 Constructive interference can reduce overall miss rates:- a)If a thread on one core accesses a main memory location, this brings the frame conaining the referened location in to the shared cache.b)If a thread on another core soon thereafter accesses the same memory block, the memoy locations will already be available in te shared chip cache.

    13.2 Data shared by multiple cores is not replicated at the shared cache.

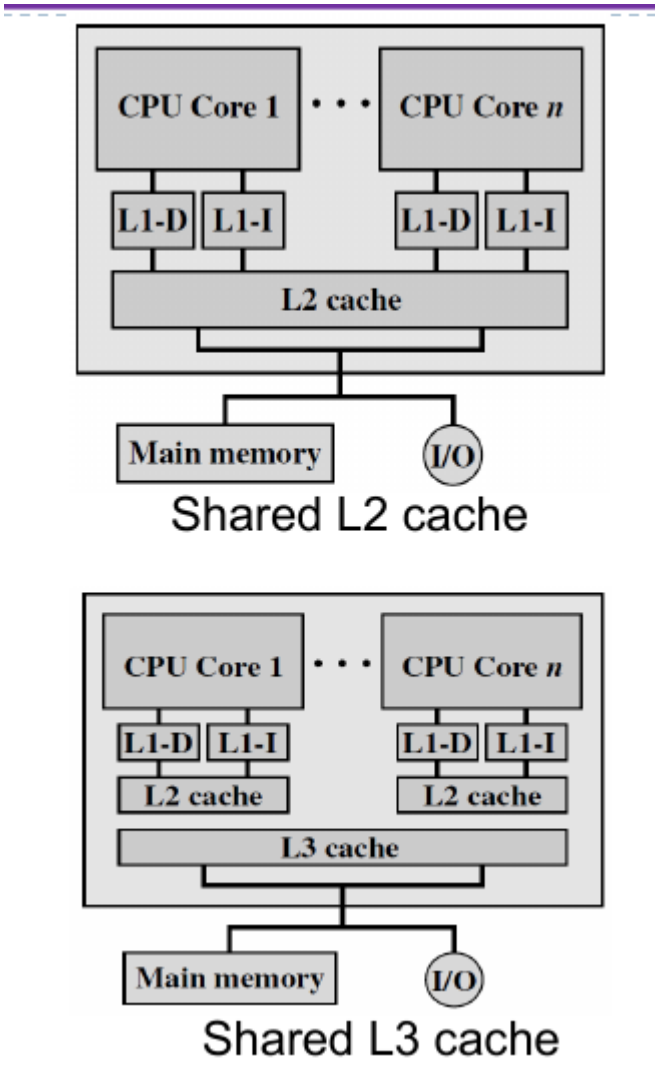


Figure 9

13.3 With proper frame replacement algorithms, the amount of shared cache allocated to each core is dynamic, so that theads that have a less locality can employ more cache.

13.4 Interprcessor communicaton is easy to implement, via shared memory locations.

13.5 The use of a shred L2 cache confines the cache coherency problem to the L1 cache level which may provide some additional performance advantage.

13.6 An advatage of having only dedicated L2 caches on the chip is that each core enjoys more rapid access to its private L2 cache.

14. Intel has introducted a number of mulicore products in recent years.

    14.1 The Intel core Duo

    14.2 The intel core i7

14.3.3 Performance

1.Hardware Issues

1. Microprocessor systems have experienced a steady, exponential increase in execution performance for decades.

2. This increase is due to refinements in the organization of the processor on the chip, and the increase in the clock frequency.

3. Increase in parallelism:- The processor design have primarily been focused on increasing instruction-level parallelism, so that more work could be done in each clock cycle with the implementing of facility like Super scalar cache, simultanoeus multi-threading and multi-core processors.

4. Pipelining- The instruction which are saved in stack are executed through a pipeline of stages so that while one instruction is executing in one stage of the pipeline, another instruction is executing in another stage of the pipeline. Pipelining can be range from three stages to five stages.

5. Superscalar:- Multiple pipelines are constructed by replicating execution resources. This enables parallel execution of instructions in parallel pipelines, so long as hazards are avoided. But on the other hand there is limitation also as more logic is required to manage hazards and to stage instruction resources.

6. Simulteneous multithreading(SMT):- Register banks are replicated so that multiple threads can share the use of pipeline resources. In order to increase the register banks means to increase the amounts of the chip area is occupied with coordinating and signal transfer logic, which increases the difficulty of designing, fabricating and debugging the chips.

7. Power Consumption:- To maintain the higher performance, the number of transistors per chip rise and high clock frequencies, as power requirements, have grown exponentially as chip density and clock frequency have risen.

2. Software Performance Issues

1. A detailed examination of the software performance issues related to multicore organization is huge task.
2. Software on multicore:- The performance benefits of a multicore organization depend on the ability to effectively exploit the parallel resources. Consider single application running on a multicore system as speed is defined by law

   Speedup= time to execute program on a single processor / time to execute program on N parallel processors.
3. A number of classes of applications benefit directly from the ability to scale throughput with the number of cores.
4. Multithreaded native applications:- Multithreaded applications are characterized by having a small number of highly threaded processes. Eg siebel Customer Relationship Manager.
5. Multiprocess applications:- Multiprocess applications are characterized by the presence of many single-threaded processes.eg oracle database, SAP and People Soft.
6. Java applications:- Java language greatly facilitate multithreaded applications. Java applications that can benefit directly from multicore resources include application servers such as Sun's Java applications server, IBM'S Websphere etc.
7. All applications that use a Java 2 platform Enterprise Edition (J2EE platform) application server can immediately benefit from multicore technology.
8. MultiInstance applications :- Even if an individual application does not scale to take advantage of a large number of threads, it is still possible to gain advantage from multicore architecture by running multiple instances of the application in parallel.

Miscalleneous Questions

Q1. Explain the Structure of Multiprocessors Systems?

Q2. Describe about the interconnection network of Multiprocessor Systems?

Q3. Explain the organization of multicore systems?

Q4. Illustrate in detail the performance issues of multicore systems?

15

CASE STUDY

UNIT 7: PENTIUM 4 PROCESSOR ORGANIZATION AND ARCHITECTURE

Unit Structure

15.1 Objectives

At the end of this unit, Student will be able to

- Illustrate the organization of Pentium 4 Processors
- Draw the architecture of P4 Processors
- Describe the Cache of P4 Processors

15.2 Introduction

1. Pentium 4 was a series of single-core central processing units(CPU) for desktop PCs and laptops.
2. The series was designed by intel and launched in November 2000.
3. P4 clock speeds were 2.0 GHZ.
4. Intel shipped Pentium 4 processors until August 2008. Pentium 4 variants included code named willamette, Northwood, Prescott and Cedar Mill with clock speeds that varied from 1.3-3.8 GHZ.
5. The P4 processor replaced the Pentium III via an embedded seventh-generation x86 microarchitecture, known as Netburst Microarchitecture, which was the first new chip architecture launched after the P6 microarchitecture in the 1995 pentium Pro CPU model.

6. The Pentium 4 architecture enchanced chip processing in the following ways:

   6.1 Performance was boosted by increased processor frequency.

   6.2 A rapid-execution engine allowed each instruction execution to occur in a half-clock cycle.

   6.3 The 400 MHZ system bus had data transfer rates of 3.2 GBPS.

   6.4 Execution trace cache optimized cache memory and improved multimedia units and floating points.

   6.5 Advanced dynamic execution enabled faster processing, which was especially critical for voice recognition, video and gaming.

15.3 Organization

1. The pentium 4 has three levels of cache.
2. The level 1 cache is a split cache and is 8 KB in size and is four-way set associative. This means that each set is made up of four lines in cache. The replacement algorithm used for this is a "least recently used" algorithm. The line size is 64 bytes.
3. The Pentium 4 makes use of "Hyper-Pipelined Technology" for performance. The pipeline is a 20-stage pipeline, meaning that 20 instructions can be run simultaneously. This is an improvement from the Pentium III pipeline which only allowed 10.
4. With the longer pipeline, less actual work is being done as more time is dedicated to filling the pipeline. Therefore, the pentium 4 pipeline has to run at a higher frequency in order to do the same amount of work as the shorter pentium III pipeline.
5. Intel use an enhanced out-of-order speculative execution engine with the pentium 4 using advanced prediction algorithms to obtain more instructions to execute using deeper out-of-order resources, up to 126 instructions.
6. The level 1 cache is small to reduce latency, taking 2 cycles for an integer data cache hit and 6 cycles for a floating point.
7. The pentium 4 uses a trace cache which takes advantage of the advanced branch prediction algorithms rather than using classic level 1 instruction cache.

8. Pentium 4 cache organization consist of

8.1 80386- no on chip cache

8.2 80486- 8k bytes using 16 bytes/lines and 4-way set associative organization

8.3 All version of Pentium consist of two L1 caches chip

8.4 L1 caches are 8k bytes, having 64 bytes/line and 4 way set associative

8.5 L2 cache consist of L1 cache with 256 Kbytes, having 128 bytes/line and 8.6-8-way set associative

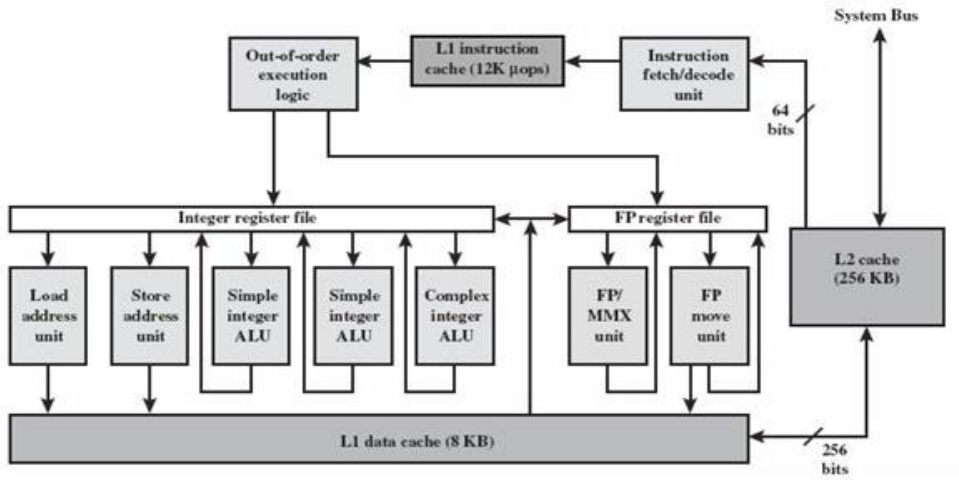


Fig 1 Pentium 4 block diagram

9. The above diagram explains about the placement of the three caches, the processor core consists of four major components

9.1 Fetch/decode unit:- Fetches program instructions in order from the L2 cache, decodes these in to a series of micro-operations and stores the results in the L1 instruction cache.

9.2 Out-of-order execution logic:- Schedules execution of the micro-operations subject to data dependencies and resource availability, thus micro operations may be scheduled for execution in a different order than they were fetched from the instruction stream.

9.3 Execution units:- These units execute micro-operations, fetching required data from the L1 data cache and temporarily storing results in registers.

9.4 Memory subsystem:- This unit includes the L2 and L3 cache and system bus, which is used to access main memory when the L1 and L2 cache have a cache miss, and to access the system I/O resources.

10. A fast processor requires balancing and tuning of many microarchitectural features that complete for processor die cost and for design and validation efforts. Figure below shows the basic intel netburst microarchitecture of the pentium 4 processor.

11. There are four main sections: the in-order front end, the out-of-order execution engine, the integer and the floating-point execution units and the memory subsystem.

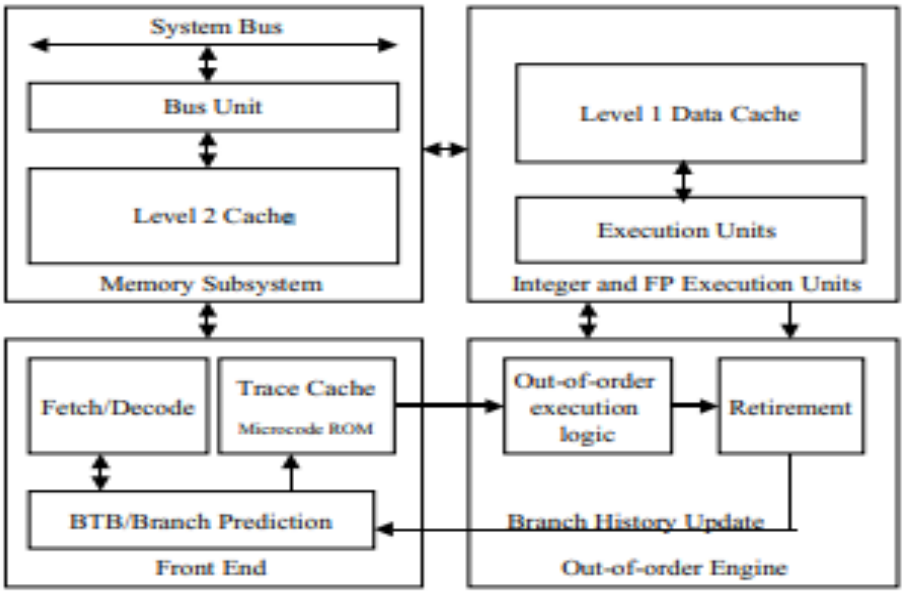


Fig 2 P4 Microarchitecture

12. In-Order Front end:- a) The in-order front end is the part of the machine that fetches the instructions to be executed next in the program and prepares them to be used later in the machine pipeline. b)Its job to supply a high-bandwidth stream of decoded instructions to the out-of order execution core, which will do the actual completion of the instructions.. c)The front end has highly accurate branch prediction logic that uses the past history accurate branch prediction logic that uses the past history of program execution to speculate where the program is going to execute next.

13. Out-of-order execution logic:- a)The out-of-order execution engine is where the instructions are prepared for execution. b)It executes logic which has several buffers that it uses to smooth and re-order the flow of instructions to optimize performance as they go down the pipeline and get scheduled for execution. c)It allows the execution resources such as the ALUs and the cache to be kept as busy as possible executing independent instructions that are

ready to execute. d)The retirement logic is what reorders the instructions executed in an out-of-order manner, back to the original program order.

14. Integer and Floating-point execution units:- a)The execution units are where the instructions are actually executed. b)This section includes the register files that store the integer and floating-point data operand values that the instructions need to execute. c)The execution units include several types of integer and floating point execution units that compute the results and also the L1 data cache that is used for most load and store operations.

15. Memory subsystem:- a)This includes the L2 cache and the system bus. b)The L2 cache stores data that cannot fit in the level1 (L1) caches. c) The external system bus is used to access main memory and also the system I/O devices, when the L2 cache has a flag of cache miss d)With the help of memory subsystem, it is easy to use high bandwidth stream-oriented applications such as 3D, video and content creation.

16. Performance:- a)The Pentium P4 processor delivers the highest SPECint_base performance of any processor in the world. b)It also delivers world-class SPECfp2000 performance. c) These are industry standard benchmarks that evaluate general integer and floating-point application performance. d) Figure given below shows the performance of P4 compared to PIII

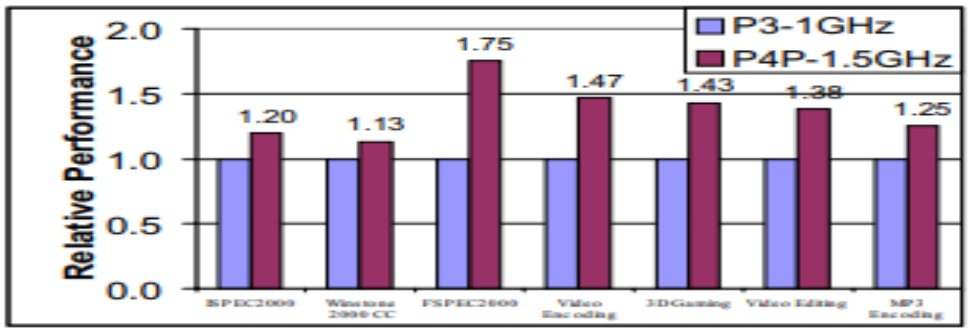


Fig 3 Performance Comparison

15.4 Architecture

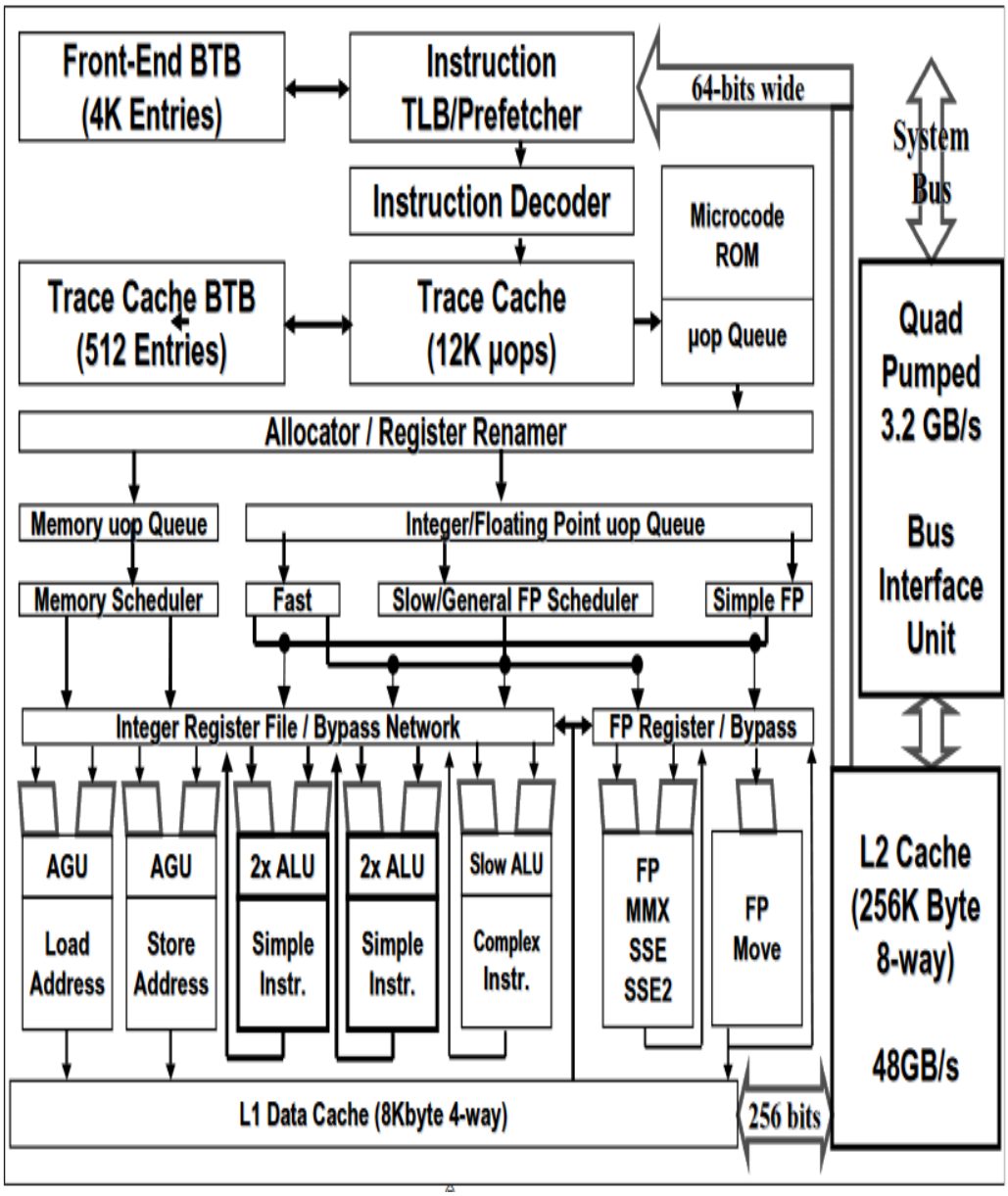


Figure 4 Pentium 4 Processor Architecture

Architecture Description

1. Above figure shows a more detailed block of the net burst micro-architecture of the pentium 4 processor.

2. The top-left diagram shows the front end of the machine, whereas as the middle of the diagram illustrates the out-of-order buffering logic and the bottom of the diagram shows the integer and floating-point execution units and the L1 data cache and the right of the diagram is the memory subsystem.

3. Front End- The front end of the pentium 4 processor consists of several units as shown in top of the figure. It consists of following components 1)Instruction TLB(ITLB) 2)The front-end branch predictor 3)The IA-32 instruction decoder 4)The Trace cache 5) Microcode ROM

4. Trace cache- a)The trace cache is the primary or level 1 (L1) instruction cache of the P4 processors and delivers up to three micro ops per clock to the out-of-order execution. b)IA-32 instructions are cumbersome to decode. The instructions have a variable number of bytes and have many different options. c)The Execution trace cache takes the already-decoded microops from the IA-32 instruction decoder and builds them in to program-ordered sequences of microops called traces.d)Conventional instruction caches typically provide instructions up to and including a taken branch instruction but none after it during that clock cycle. e)The Trace cache predictor is smaller than the front-end predictor as its main purpose is to predict the branches in the subset of the program that is currently in the Trace cache.

5. Microcode ROM- a)Its near the Trace cache. This ROM is used for complex IA-32 instructions such as string move, and for fault and interrupt handling. b)After the microcode ROM finishes sequencing micro ops for the current IA-32 instruction, the front end of the machine resumes fetching micro ops from the trace cache. c)The micro ops that come from the trace cache and the microcode ROM are buffered in a simple in-order micro ops queue that helps smooth the flow of micro ops going to the out of order execution logic.

6. ITLB and Front-end BTB- a)The ITLB translates the linear instruction pointer addresses given to it into physical addresses needed to access the L2 cache. The ITLB also performs page-level protection checking. b)The front-end branch predictor is quite large–4K branch target entries–to capture most of the branch history information for the program. If a branch is not found in the BTB, the branch prediction hardware statically predicts the outcome of the branch based on the direction of the branch displacement (forward or backward).

7. IA-32 Instruction Decoder- a) The instruction decoder receives IA-32 instruction bytes from the L2 cache 64-bits at a time and decodes them into primitives, called uops, that the machine knows how to execute. b)This single instruction decoder can decode at a maximum rate of one IA-32 instruction per clock cycle.

8. Out-of-order Execution logic- a)The out-of-order execution engine consists of the allocation, renaming, and scheduling functions. b)The out-of-order execution engine will execute as many ready instructions as possible each clock cycle, even if they are not in the original program order. c)The out-of-order execution engine has several buffers to perform its re-ordering, tracking, and sequencing operations. The Allocator logic allocates many of the key machine buffers needed by each micro ops to execute.

9.

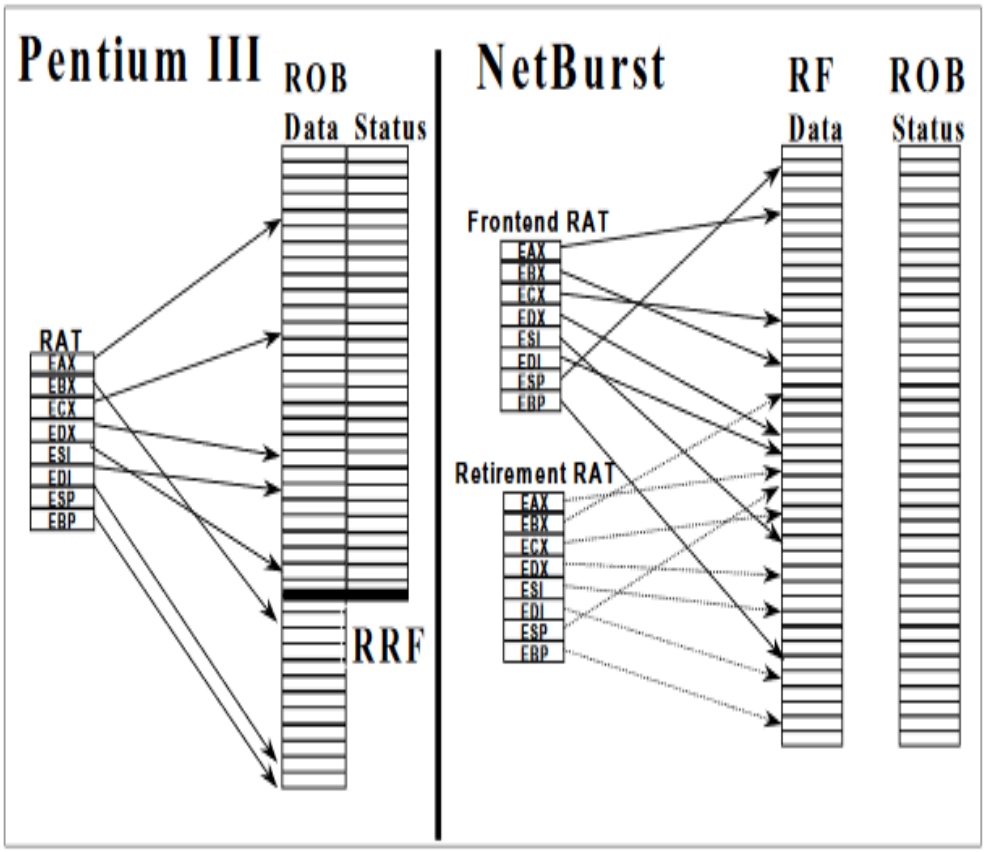


Figure 5 P3 V/s P4 processor register allocation

9.1 As shown in above figure the NetBurst microarchitecture allocates and renames the registers somewhat differently than the P6 microarchitecture. It allocates the data result registers and the ROB(Recorded Buffer) entries as a single, wide entity with a data and a status field. The ROB data field is used to store the data result value of the uop, and the ROB status field is used to track the status of the uop as it is executing in the machine. These ROB entries are allocated and deallocated sequentially and are pointed to by a sequence number that indicates the relative age of these entries. Upon retirement, the result data is physically copied from the ROB data result field into the

separate Retirement Register File (RRF). The RAT(Register Alias Table) points to the current version of each of the architectural registers such as EAX. This current register could be in the ROB or in the RRF.

10. Micro ops (Uop) Scheduling- a)The uop schedulers determine when a uop is ready to execute by tracking its input register operands. This is the heart of the out-of-order execution engine. b)The NetBurst microarchitecture has two sets of structures to aid in uop scheduling: the uop queues and the actual uop schedulers. c)There are two uop queues–one for memory operations (loads and stores) and one for non-memory operations. Each of these queues stores the uops in strict FIFO (first-in, first-out) order with respect to the uops in its own queue, but each queue is allowed to be read out-of-order with respect to the other queue. d)There are several individual uop schedulers that are used to schedule different types of uops for the various execution units on the Pentium 4 processor as shown in given figure below.e)These schedulers are tied to four different dispatch ports. There are two execution unit dispatch ports labeled port 0 and port 1.e)Multiple schedulers share each of these two dispatch ports. The fast ALU schedulers can schedule on each half of the main clock cycle while the other schedulers can only schedule once per main processor clock cycle.

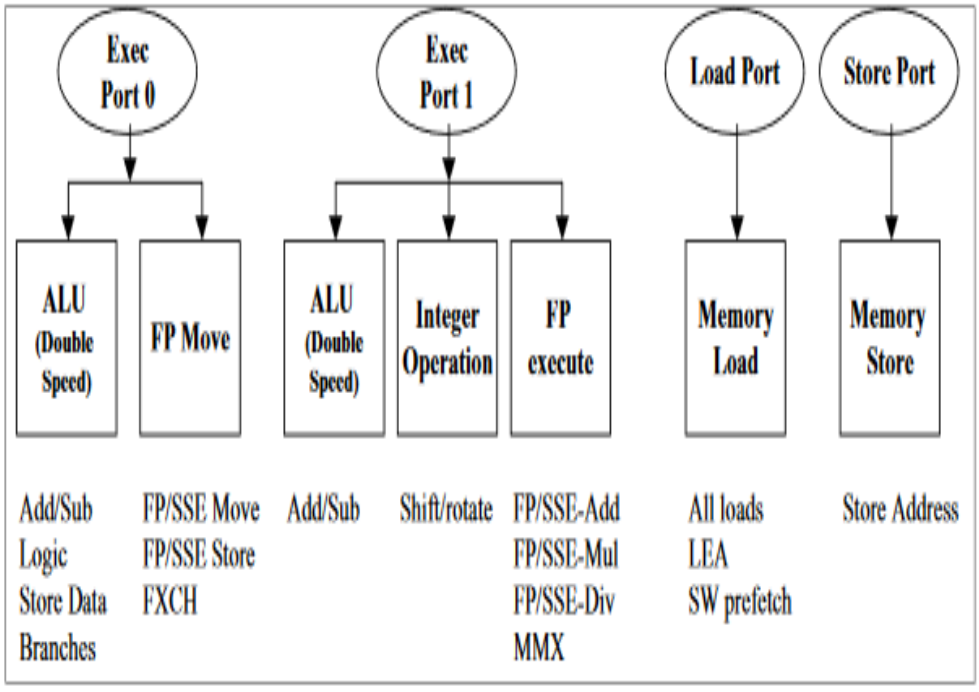


Figure 6 Dispatch ports in the pentium 4 processor

Miscellaneous Questions

Q1. Discuss the architecture of Pentium 4 processor

Q2 Why P4 is better than the P3 Processor

Q3. Name the Dispatch ports of micro ops (Uops) Schedulers

Q4. Give the performance measures of P4 processors