## Preface

Dear students,

I am extremely happy to present the book of **" Computer Organization and Architecture"** for you. I have divided the subject into small chapters so that the topics can be arranged and understood properly. The topics within the chapters have been arranged in a proper sequence to ensure smooth flow of the subject.

A large number of solved examples have been included. So, I am sure that this book will cater all your needs for this subject.

I am thankful to Shri. Pradeep Lunawat and Shri. Sachin Shah for the encouragement and support that they have extended. I am also thankful to the staff members of Tech-Max Publications and others for their efforts to make this book as good as it is. We have jointly made every possible efforts to eliminate all the errors in this book. However if you find any, please let us know, because that will help me to improve further.

I am also thankful to my family members and friends for patience and encouragement.

I (Harish Narula) would also like to thank my Guruji Shree Swami Satyanandji Maharaj for his blessings and Mrs. Khushboo Narula for her encouragement.

**For any queries please mail on : harish@harishnarula.com**

**Author**

□□□

## Syllabus

**Mumbai University**
**Revised syllabus (Rev-2016) from Academic Year 2017-18**

### Computer Organization and Architecture

| Course Code | Course Name | Credit |
|---|---|---|
| CSC403 | Computer Organization and Architecture | 4 |

**Course Objectives :**

1. To have a thorough understanding of the basic structure and operation of a digital computer.
2. To discuss in detail the operation of the arithmetic unit including the algorithms & implementation of fixed-point and floating-point addition, subtraction, multiplication & division.
3. To study the different ways of communicating with I/O devices and standard I/O interfaces.
4. To study the hierarchical memory system including cache memories and virtual memory.

**Course Outcomes :** At the end of the course student should be able-

1. To describe basic structure of the computer system.
2. To demonstrate the arithmetic algorithms for solving ALU operations.
3. To describe instruction level parallelism and hazards in typical processor pipelines.
4. To describe superscalar architectures, multi-core architecture and their advantages
5. To demonstrate the memory mapping techniques.
6. To Identify various types of buses, interrupts and I/O operations in a computer system

**Prerequisite :** Digital Logic Design and Application

| Sr. No. | Module | Detailed Content | Hours |
|---|---|---|---|
| 1. | **Introduction** | **Overview of Computer Architecture & Organization**<br>• Introduction<br>• Basic organization of computer<br>• Block level description of the functional units.<br>**Data Representation and Arithmetic Algorithms :**<br>• Integer Data computation: Addition, Subtraction. Multiplication: unsigned multiplication, Booth" s algorithm.<br>• Division of integers: Restoring and non restoring division<br>• Floating point representation. IEEE 754 floating point number representation.<br>• Floating point arithmetic : Addition, Subtraction, Multiplication, Division. **(Refer chapter 1)** | 08 |
| 2. | **Processor Organization and Architecture** | • Von Neumann model, Harvard Architecture<br>• Register Organization, Instruction formats, addressing modes, instruction cycle. Instruction interpretation and sequencing.<br>• ALU and Shifters<br>• Basic pipelined datapath and control, Data dependences, data hazards, Branch hazards, delayed branches, branch prediction | 10 |

| Sr. No. | Module | Detailed Content | Hours |
|---|---|---|---|
| | | • Performance measures – CPI, speedup, efficiency, throughput and Amdahl's law. **(Refer chapter 2)** | |
| 3. | **Control Unit Design** | • Hardwired control unit design methods: State table, delay element, sequence counter with examples like control unit for multiplication and division<br>• Microprogrammed control Unit: Microinstruction sequencing and execution. Micro operations, Wilkie's microprogrammed Control Unit, Examples on microprograms. **(Refer chapter 3)** | 08 |
| 4. | **Memory Organization** | • Classifications of primary and secondary memories. Types of RAM (SRAM, DRAM, SDRAM, DDR, SSD) and ROM, Characteristics of memory, Memory hierarchy: cost and performance measurement.<br>• Virtual Memory: Concept, Segmentation and Paging, Address translation mechanism.<br>• Interleaved and Associative memory.<br>• Cache memory Concepts, Locality of reference, design problems based on mapping techniques. Cache Coherency, Write Policies . **(Refer chapter 4)** | 12 |
| 5. | **I/O Organization and Peripherals** | • Common I/O device types and characteristics<br>• Types of data transfer techniques: Programmed I/O, Interrupt driven I/O and DMA.<br>• Introduction to buses, Bus arbitration and multiple bus hierarchy<br>• Interrupt types, Interrupts handling **(Refer chapter 5)** | 06 |
| 6. | **Advanced Processor Principles** | • Introduction to parallel processing, Flynn's Classification<br>• Concepts of superscalar architecture, out-of-order execution, speculative execution, multithreaded processor, VLIW, data flow computing.<br>• Introduction to Multi-core processor architecture. **(Refer chapter 6)** | 08 |

□□□

A

ARTICLE

# Pre-requisites

**Syllabus**

Basic combinational and sequential logic circuits, binary numbers and arithmetic, basic computer organizations.

**Syllabus Topic : Binary Numbers, Basic Computer Organizations**

## A.1 Binary Number System

- Binary number system is used in digital systems. The major advantage of digital system over analog systems is that, there are less chances of errors in a digital system.
- Since there are only two levels in a digital system (as it uses binary number system) i.e. '0' and '1', the chances of errors are less. The voltage levels, selected for the logic '0' is 0 volts and for the logic '1', voltage level is 5 volts. Hence if noise changes the voltage level from let us say 5 volts to 4.5 volts, still it can be considered as logic '1', as the voltage level for logic '0' is very far from 4.5 volts.
- While in case in decimal number system or analog systems, the slight change in voltage can cause errors.
- But the signals available from most of the systems is analog in nature. If we want to work with digital signals, then there must be some mechanism to convert a decimal value into binary and vice-versa. In this section we will see the conversions amongst all the above discussed number system i.e. decimal, binary, octal and hexadecimal.

### A.1.1 Conversion from Decimal to Binary

The steps to be followed for converting a decimal number to a binary number are :

- Divide the decimal number by 2.
- Note down the quotient and remainder as shown in the example below.
- Repeat the above procedure till the quotient is zero.
- The last remainder is the MSB (Most Significant Bit i.e. the bit with highest weight) and the first remainder is the LSB (Least Significant Bit).

**Ex. 1**

$(5)_{10} = (?)_2$.

**Soln. :**

| | | |
|---|---|---|
| 2 | 5 | Quotient |
| 2 | 2 | 1 ← Remainder |
| 2 | 1 | 0 |
| | 0 | 1 |

$(5)_{10} = (101)_2$

**Ex. 2**

$(52)_{10} = (?)_2$.

**Soln. :**

| | | |
|---|---|---|
| 2 | 52 | Quotient |
| 2 | 26 | 0 ← Remainder |
| 2 | 13 | 0 |
| 2 | 6 | 1 |
| 2 | 3 | 0 |
| 2 | 1 | 1 |
| | 0 | 1 |

$(52)_{10} = (110100)_2$

In these examples the number is only integer, i.e. no fraction part the above method works. In case of decimal fraction number, the procedure to convert it to binary number is as follows :

- Multiply the fraction decimal number by 2.
- Note down the product, and separate the integer part and fraction part as shown in the example below.
- Repeat the above procedure till the fraction part is zero (or upto some 5-6 times).
- The first integer part is the MSB (Most Significant Bit i.e. the bit with highest weight) and the last integer part is the LSB (Least Significant Bit).

**Ex. 3**

$(0.875)_{10} = (?)_2$

**Soln. :** $0.875 \times 2 = 1.75$ — 1 (Integer Part)
$0.75 \times 2 = 1.5$ — 1
$0.5 \times 2 = 1.0$ — 1

$\therefore (0.875)_{10} = (0.111)_2$

Convert the following decimal numbers to their binary equivalents.

**Ex. 4**

$(524.31)_{10} = (?)_2$

**Soln. :**

| | | |
|---|---|---|
| 2 | 524 | |
| 2 | 262 | 0 |
| 2 | 131 | 0 |
| 2 | 65 | 1 |
| 2 | 32 | 1 |
| 2 | 16 | 0 |
| 2 | 8 | 0 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| 2 | 1 | 0 |
| | 0 | 1 |

$(524)_{10} = (1000001100)_2$

$0.31 \times 2 = 0.62$ — 0
$0.62 \times 2 = 1.24$ — 1
$0.24 \times 2 = 0.48$ — 0
$0.48 \times 2 = 0.96$ — 0
$0.96 \times 2 = 1.92$ — 1
$0.92 \times 2 = 1.84$ — 1
$0.84 \times 2 = 1.68$ — 1

$(0.31)_{10} = (0.0100111)_2$

$\therefore (524.31)_{10} = (1000001100.0100111)_2$

### A.1.2 Binary to Decimal

The steps to be followed for converting a binary number to a decimal number are :

- Multiply the binary digits (bits) with powers of 2 according to their positional weight.

**Ex. 5**

$(11100)_2 = (?)_{10}$

**Soln. :**

$= 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 0 \times 2^0$

$= 16 + 8 + 4 + 0 + 0$

$= (28)_{10}$

- In case number is only integer, i.e. no fraction, the above method works. In case of decimal numbers, the procedure to convert it into decimal number is as follows :
- Multiply the binary digits (bits) with powers of 2 according to their positional weight.

**Ex. 6**

$(0.1011)_2 = (?)_{10}$

**Soln. :**

$= 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 1 \times 2^{-4}$

$= 0.5 + 0 + 0.125 + 0.0625$

$= (0.6875)_{10}$

Convert the following binary numbers to their decimal equivalents

**Ex. 7**

$(11001011.01101)_2 = (?)_{10}$

**Soln. :**

$(11001011)_2 = 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$

$= 128 + 64 + 8 + 2 + 1$

$= (203)_{10}$

$(0.01101)_2 = 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5}$

$= 0.25 + 0.125 + 0.03125$

$= (0.40625)_{10}$

$\therefore (11001011.01101)_2 = (203.40625)_{10}$

**Ex. 8**

$(111011.111001)_2 = (?)_{10}$.

**Soln. :**

$(111011.111001)_2 = 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 0 \times 2^{-5} + 1 \times 2^{-6}$

$= 32 + 16 + 8 + 2 + 1 + 0.5 + 0.25 + 0.125 + 0.015625$

$= (59.890625)_{10}$

**Syllabus Topic : Binary Arithmetic**

## A.2 Binary Arithmetic

- The way we perform basic arithmetic operations decimal numbers, we also can perform operations binary numbers. In fact, the computers work only



binary number systems; all digital systems work with binary number systems.

- Although a computer accepts decimal input provides decimal output, but internally it works on binary. It converts the decimal input to binary as studied earlier and processes it in binary and again converts the result to decimal to give the output.
- Let us see how the operations can be done on binary data.

### A.2.1 Binary Addition

- To add two binary numbers we follow the truth-table given in Table A.2.1. Truth table is a table that shows how the behaviour of a system is.

**Table A.2.1 : Truth Table for binary addition**

| A | B | Sum | Carry |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

- The columns 'A' and 'B' in the Table A.2.1 are the inputs or the bits to be added. The "Sum" and "Carry" are the outputs or the result i.e. after adding the inputs we get a sum and the carry generated.
- 0 + 0 is 0, 0 + 1 is 1 and 1 + 0 is also 1. But, when we add 1 + 1, we get $(2)_{10} = (10)_2$. Thus sum is 0 and carry is 1, as seen in the Table A.2.1.
- Similarly when we add 1 + 1 + 1, we get $(3)_{10} = (11)_2$ i.e. sum is 1 and carry is also 1.
- When we add 1 + 1 + 1 + 1, we get $(4)_{10} = (100)_2$ where we have $(10)_2$ as carry and $(0)_2$ as sum and so on.

**Ex. 9**

Add (10110101) + (11101110)

**Soln. :**

| | | |
|---|---|---|
| | 1 1 1 1 1 | Carry |
| | 1 0 1 1 0 1 0 1 | Number 1 = $(181)_{10}$ |
| + | 1 1 1 0 1 1 1 0 | Number 2 = $(238)_{10}$ |
| | 1 1 0 1 0 0 0 1 1 | Sum = $(419)_{10}$ |

**Ex. 10**

Add (1001100) + (1011110)

**Soln. :**

| | | |
|---|---|---|
| | 1 1 | Carry |
| | 1 0 0 1 1 0 0 | Number 1 = $(36)_{10}$ |
| + | 1 0 1 1 1 1 0 | Number 2 = $(46)_{10}$ |
| | 1 0 1 0 0 1 0 | Sum = $(82)_{10}$ |

### A.2.2 Binary Subtraction

- To subtract two binary numbers we follow the truth-table given in Table A.2.2.

**Table A.2.2 : Truth table for binary subtraction**

| A | B | Difference | Borrow |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 |

- Again in Table A.2.1, 'A' and 'B' are inputs for subtraction while the "Difference" and "Borrow" are the outputs i.e. A – B is performed and the result difference and borrow required are generated as outputs.
- Thus we will see in the Table A.2.2; 0 – 0 is 0, 1 – 0 is 1 and 1 – 1 is 0. But, when we subtract 0 - 1, we need to take a borrow 1 (either we can subtract that one from the preceding bit or we actually borrow 1).
- Now when we take a borrow 1, the subtraction becomes, $(10)_2 - (1)_2$ (i.e. 2 – 1) = $(1)_2$. Hence we have difference as 1 as well as borrow as 1.

**Ex. 11**

Subtract (10110101) – (1101110)

**Soln. :**

| | | |
|---|---|---|
| | 1 0 1 1 0 1 0 1 | Number 1 = $(181)_{10}$ |
| – | 0 1 1 0 1 1 1 0 | Number 2 = $(110)_{10}$ |
| | 1 1 1 1 . | Borrow |
| | 0 1 0 0 0 1 1 1 | Difference = $(71)_{10}$ |

**Note :** As seen in the above example, in the fourth bit from right hand we have 0 – 1, which is difference 1 with borrow 1; but there is also a borrow to be subtracted from the difference, so we finally get difference 0 and borrow 1.

**Ex. 12**

Subtract (10000001) – (101110)

**Soln. :**

```
  1 0 0 0 0 0 0 1     Number 1 = (129)10
- 0 0 1 0 1 1 1 0     Number 2 = (46)10
  1 1 1 1 1 1 1 -     Borrow
  0 1 0 1 0 0 1 1     Difference = (83)10
```

### A.2.3 Positive and Negative Numbers

#### A.2.3.1 Signed Magnitude Representation

- We have seen the operations for unsigned numbers (i.e. only positive numbers) in the previous sections. But we know that in mathematics we also have negative numbers.
- There are various ways of representing negative numbers. We will see one of those methods i.e. Signed magnitude representation in this subsection and some more methods in the subsequent subsections.
- In the signed magnitude representation the MSB (Most Significant Bit or the first bit from left that has maximum weight) is used for sign, '1' indicating negative number and '0' indicating positive number.
- The remaining bits indicate magnitude. But in this form of representation '0' has two distinct representations of + 0 and – 0, which is wrong according to mathematics. So we go for other methods of negative numbers representation namely the 1's and 2's complement methods as seen in the subsequent subsections.

### A.2.4 Use of Complements to Represent Negative Numbers

#### A.2.4.1 1's Complement Method of Subtraction

- To implement addition and subtraction according to the Sections A.2.1 and A.2.2, different circuits are required. Hence to reduce the circuitry, we go for one's complement method of subtraction where we take one's complement and then add the two numbers. Also the advantage of this method is that it supports negative as well as positive numbers.
- To find ones complement in any base numbered system, we subtract it from the maximum number. For example in decimal subtract each digit by 9 and it is called as 9's complement; in octal, subtract each digit by 7 and it is called as 7's complement; in hexadecimal, subtract each digit by 15 (or F) and it is called as 15's complement; in binary, subtract each digit by 1 and it is called as 1's complement; but in binary number system this can be simply done by reversing the bits from 0 to 1 and vice-versa.
- Although different names according to the number system but same function.
- After getting one's complement of the negative number we can directly add it to the positive number, or if both numbers are negative we take one's complement of both the numbers and then add their complements of numbers to get the difference.
- But in this method also we have '0' with two representation of + 0 and – 0, which is wrong. So we go for yet another method of negative numbers representation to be seen in next subsection.

Follow the following rules to get the difference using one's complement :

**Step 1 :** First take the 1's complement of the negative number(s).

**Step 2 :** Add the complemented number with the positive number(in case of both negative numbers add both complemented numbers)

**Step 3 :** If there is a carry generated then the number is in its TRUE FORM and only the carry is to be added to the difference; else if there is no carry generated then the number is in its 1's complement form, so to get the true form take its 1's complement. And the result is negative. In case both numbers were negative the number is always in its 1's complement form so we have to take 1's complement of the result after adding the carry generated. And the number is negative.

Let us see some examples for this.

**Ex. 13**

Subtract $(35)_{10} - (23)_{10}$ using 1's complement method.

**Soln. :**

| | | |
|---|---|---|
| 2 | 35 | |
| 2 | 17 | 1 |
| 2 | 8 | 1 |
| 2 | 4 | 0 |
| 2 | 2 | 0 |
| 2 | 1 | 0 |
| | 0 | 1 |

| | | |
|---|---|---|
| 2 | 23 | |
| 2 | 11 | 1 |
| 2 | 5 | 1 |
| 2 | 2 | 1 |
| 2 | 1 | 0 |
| | 0 | 1 |

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

$(23)_{10} = (10111)_2$

**Step 1 :** Take 1's complement of negative number

$\therefore$ Taking 1's C of $(23)_{10} = (10111)_2$ by subtracting each digit from 1.



```
  1 1 1 1 1 1
-   1 0 1 1 1
  1 0 1 0 0 0
```

**Note : 1.** The number of digits must be according to the number with greater number of digits.
**2.** We can also find 1's complement by simply inverting each bit in binary.

**Step 2 :** Add the positive number with 1's complement of negative number.

```
∴      1 0 0 0 1 1
     + 1 0 1 0 0 0
   [1] 0 0 1 0 1 1
       ↑
     Carry
```

**Step 3 :** Since carry is generated, add the carry to result and the result is in true form and also the result is positive.

```
∴ [1] 0 0 1 0 1 1
      └──────────→ 1
      +
  (0 0 1 1 0 0)2 = (12)10
```

$\therefore$ Result $= +(1100)_2 = (12)_{10}$

**Ex. 14**

Subtract $(23)_{10} - (35)_{10}$ using 1's complement method.

**Soln. :**

$(23)_{10} = (10111)_2$

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

**Step 1 :** Take 1's complement of negative number.

$\therefore \quad (35)_{10} = (100011)_2$

1's complement of $(35)_{10} = -(35)_{10}$

$\therefore \quad -(35)_{10} = (011100)_2$

**Note : Replace 1's with 0's and 0's with 1's.**

**Step 2 :** Add the 1's complemented with positive number.

```
∴      0 1 0 1 1 1
     + 0 1 1 1 0 0
   [0] 1 1 0 0 1 1
       ↑
     Carry
```

**Step 3 :** Since no carry is generated, the result is negative and is in its 1's complement form. Hence to get the decimal value, we take 1's complement of the answer :

```
                1 1 0 0 1 1
1's complement = (0 0 1 1 0 0)2
```

$\therefore$ The answer $= -(1100)_2 = -(12)_{10}$

#### A.2.4.2 2's Complement Method for Subtraction

- The third method of representing signed numbers is 2's complement method. The major advantage of the 2's complement method over the signed magnitude method and the 1's complement method is that there are not two representations for '0' i.e. no separate representation for + 0 and – 0. Hence it is the most suited method in the microprocessors and digital computers for storing signed numbers.
- Also the operations like subtraction can be easily performed using the 2's complement method, almost similar to 1's complement method.
- To get the 2's complement of a number we need to first take 1's complement and then add 1 or subtract the number from the smallest number of one digit greater than the number whose 2's complement is to be found.
- The following steps are used to perform subtraction using 2's complement method :

**Step 1 :** First take the 2's complement of the negative number(s).

**Step 2 :** Add the complemented number with the positive number(in case of both negative numbers add both complemented numbers)

**Step 3 :** If there is a carry generated then the number is in its TRUE FORM and only the carry is to be discarded; else if there is no carry generated then the number is in its 2's complement form, so to get the true form take its 2's complement. And the number is negative.

In case both numbers were negative the number is always in its 2's complement form so we have to take 2's complement of the result after adding the carry generated. And the result is negative.

Let us see some examples for this.

**Ex. 15**
Subtract $(23)_{10} - (35)_{10}$ using 2's complement method.

**Soln. :**

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

$(23)_{10} = (10111)_2$

**Step 1 :** Take 2's complement of negative number

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

1's complement is $\rightarrow (011100)_2$

$+1$

2's complement is $\rightarrow (011101)_2$

**Step 2 :** Add the complement with positive number

$$\begin{array}{r} \therefore \quad 0\ 1\ 0\ 1\ 1\ 1 \\ +\ 0\ 1\ 1\ 1\ 0\ 1 \\ \hline \boxed{0}\ (1\ 1\ 0\ 1\ 0\ 0)_2 \end{array}$$

↑ Carry

**Step 3 :** Since no carry is generated, the result is negative and to get its actual value we need to take 2's complement of result.

$\therefore$ Result $= (110100)_2$

1's complement $= (001011)_2$

$+1$

$\therefore$ 2's complement $= (001100)_2$

Thus result $= -(1100)_2 = -(12)_{10}$

**Ex. 16**
Subtract $(35)_{10} - (23)_{10}$ using 2's complement method.

**Soln. :**

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

$(23)_{10} = (10111)_2$

**Step 1 :** Take 2's complement of negative number.

$\therefore (23)_{10} = (010111)_2$

1's complement of $(23)_{10} = (101000)_2$

$+1$

2's complement of $(23)_{10} = (101001)_2$

**Step 2 :** Add the complement with the positive number.

$$\begin{array}{r} \therefore \quad 1\ 0\ 1\ 0\ 0\ 1 \\ +\ 1\ 0\ 0\ 0\ 1\ 1 \\ \hline \boxed{1}\ 0\ 0\ 1\ 1\ 0\ 0 \end{array}$$

↑ Carry

**Step 3 :** Since carry is generated, the result is correct and positive. We need to just discard the carry generated.

Thus result $= (1100)_2 = (12)_{10}$

## A.3 Binary Multiplication

- To multiply two binary numbers the same rules apply as in decimal i.e. $0 \times 1 = 0$, $1 \times 0 = 0$, $0 \times 0 = 0$ and $1 \times 1 = 1$.
- While adding all the rows use the rules of binary addition.

**Ex. 17**
181 ×14. Multiply by converting to binary.

**Soln. :**

$$\begin{array}{rll} & 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1 & \text{Number 1} \\ \times & \quad 1\ 1\ 1\ 0 & \text{Number 2} \\ \hline & 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 & \\ & 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0 & \\ & 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0 & \\ & 1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 0\ 0 & \\ \hline & 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 1\ 1\ 0 & \end{array}$$

**Ex. 18**
$(20)_{10} \times (5)_{10}$. Multiply by converting to binary.

**Soln. :**

$(20)_{10} = (10100)_2$

$(5)_{10} = (101)_2$

$$\begin{array}{r} 1\ 0\ 1\ 0\ 0 \\ \times \quad 1\ 0\ 1 \\ \hline 1\ 0\ 1\ 0\ 0 \\ +\ 0\ 0\ 0\ 0\ 0\ 0 \\ \hline 1\ 0\ 1\ 0\ 0\ 0\ 0 \\ \hline 1\ 1\ 0\ 0\ 1\ 0\ 0 \end{array}$$

Thus result $= (1100100)_2 = (100)_{10}$



### A.3.1 Binary Division

To divide two binary numbers the same rules apply as in decimal. While subtracting all use the rules of binary subtraction.

**Ex. 19**
105/5. Divide using binary.

**Soln. :**

$$\begin{array}{l} 101\overline{)1101001}\,(10101 \\ \underline{-101} \\ 00110 \\ \underline{-101} \\ 00101 \\ \underline{-101} \\ 000 \end{array}$$

**Ex. 20**
$(20)_{10} / (7)_{10}$. Divide using binary.

**Soln. :**

$(20)_{10} = (10100)_2$

$(7)_{10} = (111)_2$

$$\begin{array}{l} 111\overline{)10100}\,(10 \\ \underline{-111\downarrow} \\ 00110 \end{array}$$

Quotient $= (10)_2 = (2)_{10}$

Remainder $= (110)_2 = (6)_{10}$

## A.4 Basic Logical Operations

To solve or simplify the logical expressions, used in digital circuits we need to use "logical operators". The three main logic operators are :

1. AND operator.
2. OR operator.
3. NOT operator (Invert).

Let us understand them one by one.

### A.4.1 NOT Operator (Inversion)

- The NOT operation represents a logical inversion or complementing. This operation changes one logic level to the opposite logic level as shown in Fig. A.4.1.
- When the input is HIGH (1), the output will be LOW (0) and when the input is LOW (0), the output will be HIGH (1).

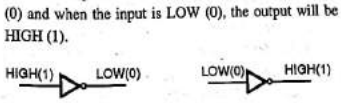



**Fig. A.4.1 : The NOT operator**

- The NOT operation is implemented by a logical circuit. called inverter. It's symbol is as shown in Fig. A.4.1.
- The "NOT" operator represents a logical inversion or complementing.

The NOT operation is denoted by a bar (–) over the variable to be inverted. For example, if A is to be inverted then the process of inversion is denoted by $\bar{A}$ and it is to be read as NOT A.

$$\bar{A} = \text{NOT A}$$

### A.4.2 AND Operator or Logical Multiplication

- The AND operation produces a high (1) output only if all the inputs are high (1), as shown in Fig. A.4.2.
- If any one or both inputs are Low (0), then output will be Low (0).
- The AND operation is implemented by a logic circuit called AND gate. Its symbol is shown in Fig. A.4.2.

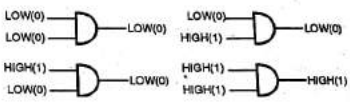



**Fig. A.4.2 : AND operation**

- The "AND" operator represents logical multiplication. It is denoted by a dot between the two variables to be multiplied. i.e.

**A · B .... Logical multiplication**

- However sometimes this dot is not used and we denote the logical multiplication of A and B as AB.
- Thus if A is to be multiplied with B in a logic circuit then the multiplication is represented as A · B, AB and it is to be read as "A AND B".

### A.4.3 OR Operator

- The OR-operation produces a HIGH (1) output when any one or all the inputs are HIGH(1).
- It will produce a LOW (0) output only if all the inputs are LOW (0).

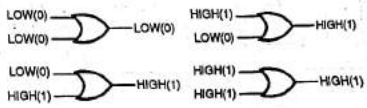



**Fig. A.4.3 : The OR operation**

- The OR operation is implemented by a logic circuit, called as OR-gate. Its symbol is shown in Fig. A.4.3.
- The "OR" operator represents logical addition. It is denoted by a (+) sign between the two variables to be added.
- If A and B are to be added in a logic circuit then it is represented as,

**A + B ... Logical addition**

And it is to be read as "A OR B".

### A.4.4 Logic Gates

- Logic gates are the basic building blocks of any digital system. It is an electronic circuit having one or more than one inputs and only one output.
- The relationship between the input and the output is based on a "certain logic". Based on this logic, the gates are named as NOT gate, AND gate, OR, NAND, NOR etc.

**Truth table :** The operation of a logic gate can be best understood with the help of a table called "Truth Table". The truth table consists of all the possible combinations of the inputs and the corresponding state of output of a logic gate.

**Boolean expression :** The relation between the inputs and the outputs of a gate can be expressed mathematically by means of the **Boolean Expression**. Let us now discuss the operation of various logic gates.

### A.4.5 Gates, Symbols and Boolean Expression

In order to understand Boolean algebra , we need to use the gates. So the symbols and Boolean expressions should be known to us. Table A.4.1 gives information.

**Table. A.4.1 : Various logic gates**

| Sr. No. | Name of gate | Boolean Expression | Logical Operation |
|---|---|---|---|
| 1. | NOT gate or inverter (A → Y) | $Y = \overline{A}$ | Inversion |
| 2. | AND gate (A, B → Y) | $Y = AB$ | Logical Multiplication |
| 3. | OR gate (A, B → Y) | $Y = A + B$ | Logical addition |
| 4. | NAND gate (A, B → Y) | $Y = \overline{AB}$ | NOT AND |
| 5. | NOR gate (A, B → Y) | $Y = \overline{A + B}$ | NOT OR |
| 6. | Exclusive OR (A, B → Y) | $Y = A \oplus B$ | Addition/ Subtraction |
| 7. | Exclusive NOR (A, B → Y) | $Y = \overline{A \oplus B}$ | NOT EXOR |

**Note :** A and B are the inputs whereas Y denotes the output.

**Syllabus Topic : Basic Combinational Logic Circuits**

## A.5 Introduction to Combinational Circuits

**Types of digital systems :**

The digital systems in general are classified into two categories namely :

1. Combinational logic circuits
2. Sequential logic circuits.

**Combinational circuits**

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuits do not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit.
- The sequence in which the inputs are being applied has no effect on the output of a combinational circuit.
- A combinational circuit is a logic circuit the output of which depends only on the combination of the inputs. The output does not depend on the past value of inputs or outputs. Hence combinational circuits do not require any memory.




**Fig. A.5.1 : Block diagram of a combinational circuit**

- The block diagram of a combinational circuit is shown in Fig. A.5.1.
- A combinational circuit can have a number of inputs and a number of outputs. The circuit of Fig. A.5.1 has "n" inputs and "m" outputs.
- Between the inputs and outputs logic gates are connected so combinational circuit basically consists of logic gates.
- A combinational circuit operates in three steps,
  a) It accepts n-different inputs.
  b) The combination of gates operates on the inputs.
  c) "m" different outputs are produced as per requirement.

**Examples of combinational circuits**

Following are the examples of some combinational circuits :

1. Adders, subtractors
2. Comparator
3. Code converters
4. Encoders, decoders
5. Multiplexers, demultiplexers

### A.5.1 Combinational Circuit Design

- In this section we are going to discuss the design of combinational circuits.
- To design a combinational circuit we have to specify the requirements of combinational circuits. Such requirements can be specified in the following ways :
  1. A set of statements.
  2. Boolean expressions or
  3. Truth table.
- From the specified requirements we have design a combinational circuit using a combination of gates.
- There are two approaches to the combinational circuit design.
  1. Traditional methods
  2. Use of Medium Scale Integration (MSI)

**Steps in traditional method**

1. A truth table is given.
2. Obtain the Boolean expression from the truth table.
3. Simplify the Boolean equation using standard methods.
4. Realize the simplified equation using gates.

**Methods to simplify the boolean functions**

- The methods used for simplifying the Boolean functions are as follows :
  1. Algebraic method.
  2. Karnaugh-map simplification.
  3. Quine-Mc Cluskey method and
  4. Variable entered mapping (VEM) technique.
- Out of these, the method of algebraic reduction using Boolean algebra has been already discussed in previous section.
- The Boolean theorems and De-Morgan's theorems are useful in manipulating the logic expressions. We can then realize the logical expressions using gates.
- The number of logic gates required for the realization of a logical expression should be reduced to the minimum possible value.
- This is possible if we can simplify the logical expressions. In this chapter we will discuss one of the simplification techniques called **Karnaugh map or K-map.**

**Ex. 21**

A circuit has four inputs and two outputs. One of the outputs is high when majority of inputs are high. The second output is high only when all inputs are of same type. Design the combinational circuit.

**Soln. :**

**Step 1 :** Assign symbols to input and output variables :

Let the four inputs be A, B, C, D and the two outputs be $Y_1$ and $Y_2$.

**Step 2 :** Write the truth table :

The truth table is as given in Table Ex. 21.

**Table Ex. 21 : Truth table relating the inputs and outputs**

| | Inputs | | | | Outputs | |
|---|---|---|---|---|---|---|
| Decimal | A | B | C | D | $Y_1$ | $Y_2$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 1 | 0 | 1 | 0 | 0 |
| 6 | 0 | 1 | 1 | 0 | 0 | 0 |
| 7 | 0 | 1 | 1 | 1 | 1 | 0 |
| 8 | 1 | 0 | 0 | 0 | 0 | 0 |

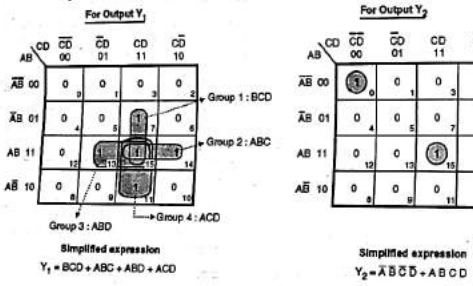
| Decimal | Inputs A | B | C | D | Outputs $Y_1$ | $Y_2$ |
|---|---|---|---|---|---|---|
| 9 | 1 | 0 | 0 | 1 | 0 | 0 |
| 10 | 1 | 0 | 1 | 0 | 0 | 0 |
| 11 | 1 | 0 | 1 | 1 | 1 | 0 |
| 12 | 1 | 1 | 0 | 0 | 0 | 0 |
| 13 | 1 | 1 | 0 | 1 | 1 | 0 |
| 14 | 1 | 1 | 1 | 0 | 1 | 0 |
| 15 | 1 | 1 | 1 | 1 | 1 | 1 |

From the truth table we note the following things :

- $Y_1 = 1$ when number of 1 inputs is higher than number of 0 inputs.
- $Y_2 = 1$ when A = B = C = D.

**Step 3 : Write K-map for each output and get simplified expression**

K-maps for the two outputs and the corresponding simplified Boolean expressions are given in Fig. Ex.21 (a) and (b).




(a) K-map and simplification for $Y_1$ (b) K-map and simplification for $Y_2$

Fig. Ex. 21

**Step 4 : Draw the logic diagram**

The logic diagram is as shown in Fig. Ex. 21 (c).

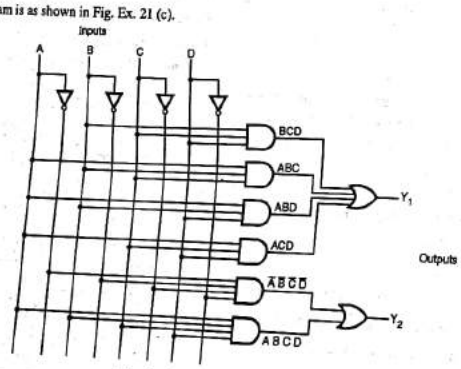



Fig. Ex. 21 (c) : Logic diagram



**Syllabus Topic : Basic Sequential Logic Circuits**

## A.6 Introduction to Sequential Circuits

**1. Combinational circuits**

- Till now we have discussed only the combinational circuits.
- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- The combinational circuits do not use any memory. Hence the previous state of input does not have any effect on the present state of the circuit.
- The sequence in which the inputs are being applied has no effect on the output of a combinational circuit.

**2. Sequential circuits**

- In the sequential circuit, the **timing** parameter comes into picture.
- The output of a sequential circuit depends on the present time inputs, the previous output and the sequence in which the inputs are applied.
- In order to provide the previous input or output a memory element is required to be used. Thus a sequential circuit needs a memory element.

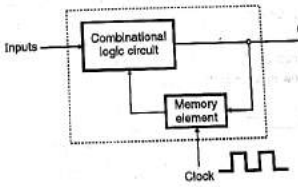



Fig. A.6.1 : Block diagram of a sequential circuit

- Fig. A.6.1 shows the block diagram of a sequential circuit which includes the memory element in the feedback path.
- **Present state of sequential circuit :** The data stored by the memory element at any given instant of time is called as the present state of the sequential circuit.
- **Next state :** The combinational circuit operates on the external inputs and the present state to produce new outputs. Some of these new outputs are stored in the memory element and called as the next state of the sequential circuit.
- The most important part of the sequential circuit seems to be the memory element. The memory element of Fig. A.6.1 is known as flip flop (FF). It is the basic memory element.

### A.6.1 Clock Signal

- The clock signal shown in Fig. A.6.2 is a timing signal. Every sequential signal will have this timing signal applied.
- Clock is a rectangular signal as shown in Fig. A.6.2; with a duty cycle equal to 50%.
- The clock signal repeats itself after every T seconds. Hence the clock frequency is $f = 1/T$.

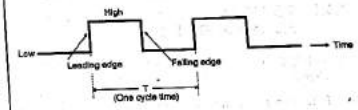



Fig. A.6.2 : Clock signal

### A.6.2 Clock Skew

- Clock skew is defined as the difference in time between the clock edges arriving at a pair of clock inputs.
- In a perfect system, all clock signals arrive at all the various clock input pins of the system at exactly the same time and the skew is zero.
- But in real time systems, the edges do not arrive at exactly the same time and there is some skew.
- This clock skew occurs due to different delays on different paths from the clock generator to various circuits. The major reasons for this are :
  - (i) Different length of wires (wires introduce delay)
  - (ii) Gates (buffers) on the paths
  - (iii) Flip-flops that clock on different edges (need for inverting clock for some flip-flops)
  - (iv) Gating the clock to control loading of registers
- The **maximum allowable clock skew** for the system is the difference between the shortest and longest path delays.
- Clock skew is an important design parameter in high-speed clock systems.

### A.6.3 Comparison of Combinational and Sequential Circuits

| Sr. No. | Parameter | Combinational circuits | Sequential circuits |
|---|---|---|---|
| 1 | Output depend on | Inputs present at that instant of time. | Present inputs and past inputs / outputs. |
| 2 | Memory | Not necessary | Necessary |
| 3 | Clock input | Not necessary | Necessary |
| 4 | Examples | Adders, subtractors, code converters | Flip flops, shift registers, counters |

### A.6.4 1-Bit Memory Cell (Basic Bistable Element or Flip Flop)

- Flip-flop is also known as the basic digital memory circuit.
- It has two stable states namely logic 1 state and logic 0 state. We can design it either using NOR gates or NAND gates.
- A flip-flop can be designed by using the fundamental circuit shown in Fig. A.6.3. NAND gates 1 and 2 are basically acting as inverters. Hence this circuit is called as a cross coupled inverter.
- Output of gate 1 is connected to the input of gate-2 and output of gate 2 is connected to input of gate-1 as shown in Fig. A.6.3.

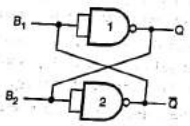



**Fig. A.6.3 : A cross coupled inverter as memory element**

**Operation :**

- Assume that output of gate-1 i.e. Q = 1. Hence $B_2 = 1$.
- As $B_2 = 1$, output of gate-2 i.e. $\bar{Q} = 0$. This makes $B_1 = 0$.
- Hence Q continues to be equal to 1.
- Similarly we can demonstrate that if we start with Q = 0, then we end up obtaining Q = 0 and $\bar{Q} = 1$.

**Conclusions :** From the above discussion we can draw the following conclusions :

- The outputs of the circuit (Q and $\bar{Q}$) will always be complementary. That means if Q = 0 then $\bar{Q} = 1$ and vice versa. They will never be equal Q = $\bar{Q}$ = 0 or 1 which is an invalid state.
- This circuit has two stable states. One of them corresponds to Q = 1, $\bar{Q}$ = 0 and it is called as 1 state or **set state**. Whereas the other corresponds to Q = 0, $\bar{Q}$ = 1 and it is called as 0 state or **reset state**.
- If the circuit is in the reset state (Q = 0, $\bar{Q}$ = 1), then it will continue to be in the reset state and if it is in the set state (Q = 1, $\bar{Q}$ = 0) then it will continue to remain in the set state.
- This property of the circuit shows that it can store 1 bit of digital information. Therefore it is called as a **1-bit memory cell**.

### A.6.5 SR Flip Flop or a Latch

- The cross coupled inverter of Fig. A.6.3 is capable of locking or latching the information. Hence this circuit is also called as a latch.
- The disadvantage of this circuit is that we cannot enter the desired digital data into it.
- This disadvantage can be overcome by modifying the circuit as shown in Fig. A.6.4. This modification will allow us to enter the desired digital data into the circuit.

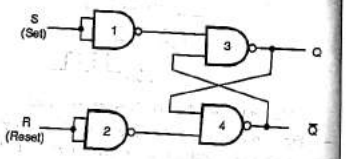



**Fig. A.6.4 : Modified memory cell**

## A.7 Two Bit Asynchronous Up Counter using JK Flip-Flops

- A 2 bit asynchronous counter up using JK flip-flops is shown in Fig. A.7.1. Note that the J and K inputs of both the flipflops are connected to logic 1 so actually they are converted into T flip-flops.
- The operation of this circuit is exactly same as that of the counter using the T flip-flops.



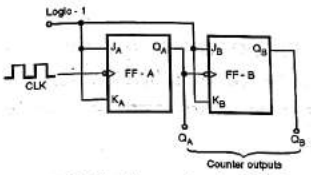



**Fig. A.7.1 : Two bit ripple up counter using JK flip-flops**

## A.8 Classification of Registers

- Registers are classified based on the way in which data are entered and taken out from a register. There are four possible modes as follows :

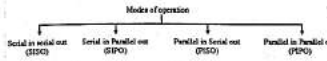



**Fig. A.8.1 : Modes of operation of registers**

- It is possible to design registers using discrete flip-flops such as SR, JK or D flip-flops.
- But the registers are also available in MSI devices.

They are available in 54/74 TTL series as listed in Table A.8.1.

**Table A.8.1 : Shift registers available in 74/54 series**

| IC number | Description |
|---|---|
| 7491, 7491A | 8 bit serial in, serial out |
| 7494 | 4 bit parallel in, serial out |
| 7495 | 4 bit serial/parallel in, parallel out (shift right shift left) |
| 7496 | 5 bit parallel in / parallel out, serial in / serial out. |

## A.9 Buffer Registers

- The simplest type of register constructed using four D flip-flops is shown in Fig. A.9.1. This is a 4 bit register, but we can construct an n-bit register by following the same principle.
- This register is also called as the **buffer register**.
- Each D-flip-flop is negative edge triggered and all the flip-flops are connected to a common clock signal. Hence all of them are triggered at the same instant of time.
- Buffer registers are used for temporary storage of digital words.

**Operation :**

- The word to be stored is $B_3 B_2 B_1 B_0 = 1010$.
- These bits are connected to the D inputs of the four D flip-flops as shown in Fig. A.9.1.

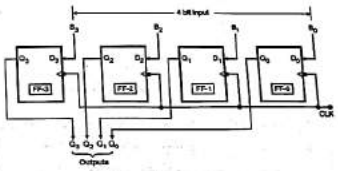
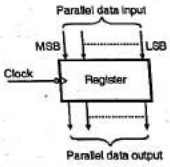



**Fig. A.9.1 : A four bit buffer register using D flip-flops**

- Then the clock pulse is applied.
- Corresponding to the first negative edge of the clock pulse, the outputs of all the D flip-flops will follow their respective inputs.

$$\therefore \quad Q_3 Q_2 Q_1 Q_0 = B_3 B_2 B_1 B_0 = 1010$$

- Even if the inputs are now changed, the output remains latched to 1 0 1 0 till the next negative edge of the clock.
- Thus the buffer register is capable of storing the digital data.




**Schematic diagram :**

The schematic diagram of a buffer register is as shown in Fig. A.9.2.

**Fig. A.9.2 : Schematic diagram of buffer register**

**Conclusions :**

Some of the important conclusions from the discussion till now are as follows :

1. There must be 1-FF for each bit to be stored. Hence to store a 4 bit number we need four flip-flops.
2. Note that all the four input bits $B_3 B_2 B_1 B_0$ are loaded into the register simultaneously, i.e. at the same instant of time.
3. Hence this way of applying the input and taking the output is called as parallel input parallel output and the mode of operation is called as **parallel shifting**.

CHAPTER 1

Module I

# Introduction to Computer Organization and Architecture

**Syllabus**

**Overview of Computer Architecture and Organization**

Introduction, Basic organization of computer, Block level description of the functional units.

**Data Representation and Arithmetic Algorithms**

Integer Data computation : Addition, Subtraction. Multiplication: unsigned multiplication, Booth's algorithm. Division of integers : Restoring and non restoring division, Floating point representation. IEEE 754 floating point number representation. Floating point arithmetic: Addition, Subtraction, Multiplication, Division.

**Syllabus Topic : Introduction**

## 1.1 Introduction

→ **(MU - Dec. 2015, May 2016, Dec. 2016)**

**Q.** Differentiate between Computer Organization and Computer Architecture. **Dec. 15, May 16, Dec. 16. 5 Marks**

**Q.** List different memory organization characteristics. **Dec. 15. 5 Marks**

**Q.** Compare Computer Architecture and organization. **(5 Marks)**

- There are various people involved in making of a computer. The chip designer who designs and manufactures the chip, the system designer who designs the system using the manufactured chip or microprocessor and the programmer who makes software using the system.
- Those attributes of a computer that are necessary to be known to a system designer or a programmer are called as the architectural features of the computer. Hence the people manufacturing the chip have to reveal certain things about the processor to the system designer and the programmer using the datasheets for their processor chips.
- Those attribute of a computer or moreover the processor, that are just used for the designing purposes of the processor and are not revealed are called as the organizational features of the processor.

| Sr. No. | Computer architecture | Computer organization |
|---|---|---|
| 1. | It refers to those attributes of a system visible to the programmer. | It refers to the implementation of the features and is mostly not known to the user. |
| 2. | Instruction set, number of bits used for data representation, addressing techniques etc. form the part of computer architecture. | Control signals, interfaces, memory technology etc. form the part of the computer organization. |
| 3. | For example, is there a multiply instruction? | For example, is there a dedicated hardware multiply unit or it is done by repeated addition? |
| 4. | All INTEL 80x86 microprocessors share the same basic architecture. | All INTEL 80x86 microprocessors differ in their organization. |



**Syllabus Topic : Basic Organization of Computer and Block Level Description of Functional Units**

## 1.2 Basic Organization of Computer and Block Level Description of Functional Units

### 1.2.1 Structural Components of a Computer

**Q.** Explain the structural overview of a computer. **(5 Marks)**

It is the way in which components related to each other. Fig. 1.2.1 shows the structure of a digital computer. It is made up of three main components namely the Central Processing Unit (CPU) or the processor, the memory to store the programs and data, and the Input / Output (I/O) devices. The functions of these are explained below :

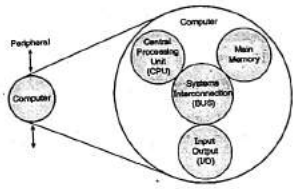



**Fig. 1.2.1 : Structure of a computer**

(i) The components of a computer are central processing unit, Input and Output (I/O) devices and memory.

(ii) The three units are connected with the help of system interconnection i.e. buses.

(iii) Memory is used to store code (programs) and data. It can be various kinds of like semiconductor memory using ICs, magnetic memory or optical memory etc.

(iv) I/O devices are used to accept a input or give an output by the CPU. There are various input devices like keyboard, mouse, scanner; and various output devices like CRT, printer etc.

(v) The CPU is further divided into three units as shown in the Fig. 1.2.2.

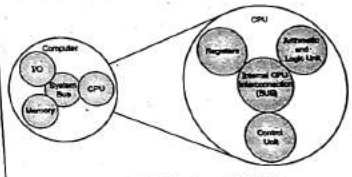



**Fig. 1.2.2 : Structure of a CPU**

(vi) The components of the Central Processing Unit (CPU) are Arithmetic and Logic Unit (ALU), Control Unit(CU) and CPU Registers, which are also connected with the internal buses.

(vii) ALU is used to perform arithmetic operations like addition, subtraction etc. and logical operations such as AND, OR etc.

(viii) CPU Registers are used to store the data temporarily in the CPU to save memory access time.

(ix) The CU is further divided in three parts as shown in Fig. 1.2.3.

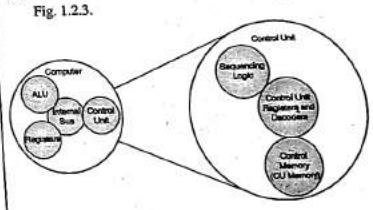



**Fig. 1.2.3 : Structure of Control Unit**

(x) Control Unit comprises of control memory, control unit register and sequencing logic.

(xi) The control memory stores the microinstructions loads it into the control unit register and the sequencing logic gives these signals in a proper sequence to execute a instruction.

### 1.2.2 Functional View of a Computer

(i) It is the operation of the individual component as the part of the structure.

(ii) The functions of a computer are data processing, data storage, data movement and control.

(iii) There can be various paths followed by the data as shown in the Fig. 1.2.4. They can be: the data may be taken into the processor from the input device, processed and then the result may be given at the output device; the data may be taken into the processor from the input device, processed and the result stored in memory; the data may be taken from memory, processed and the result given to output device; the data taken from an input device and given to an output device etc.

(iv) In Fig. 1.2.4, we can see that the computer is divided into three main components namely data storage facility, data movement facility and the data processing facility. These are the different functions can perform a computer.

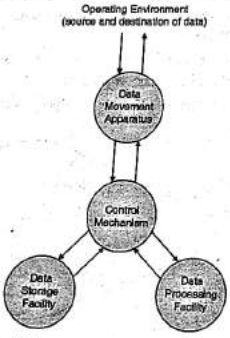



**Fig. 1.2.4 : Functional view of a Computer**

## 1.3 Evolution of Computers

- The evolution of computers seen and used by us these days have gone through many major eras. The mechanical era and the electronic era.
- The electronic era further has many generations namely the vacuum tube, semiconductor and IC. These are discussed below.

### 1.3.1 Mechanical Era (1600s-1940s)

Some of the people involved and their major breakthroughs in this era are listed below :

(i) **Wilhelm Schickhard** in 1623 designed a system that could automatically add, subtract, multiply, and divide.

(ii) **Blaise Pascal** in 1642 sis mass production of machine that could add and subtract. He manufa... 50 such machines.

(iii) **Gottfried Liebniz** in 1673 improved on P... machine so that it could add, subtract, multiply, ... divide.

(iv) **Charles Babbage** in 1822 designed the structure ... computer which is used even in the modern comp... He is called as the "Father of modern computer". ... the Modern structure: I/O, storage, ALU. His ma... could add in 1 second, multiply in 1 minute.

(v) **Herman Hollerith** in 1889 formed Tabu... Machine Company which later became IBM. He ... designed a tabulating machine to tabulate the ce... data for his country.

(vi) **Konrad Zuse** in 1938 built first working mechan... computer, called as the Z1. The specialty of ... computer was that it was a binary machine.

### 1.3.2 The Electronic Era

- As discussed earlier there were various generations ... this era based on the material used to manufacture ... CPU.
- These are discussed below.

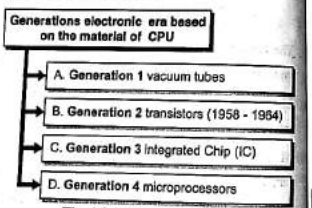



**Fig. 1.3.1 : Electronic Era**

→ **(A) Generation 1 Vacuum Tubes**

- This generation was between the 1945 and 1958.
- The vacuum tubes are of two types namely vacuu... diodes and triodes that function similar ... semiconductor diode and transistors respectively. Th... difference being that vacuum tubes consume a lot ... power, and are bulky (almost of the size of a small ... bulb).
- There were two major computers developed using th... vacuum tubes as listed below.

(i) ENIAC (Electronic Numerical Integrator and Computer)



- This computer is regarded as the first electronic computer.
- It used decimal number system.

(ii) IAS (Institute for Advanced Studies)

This architecture later came to be known as the "Von Neumann" architecture and has been the basis for almost all the machines designed since then.

→ **(B) Generation 2 Transistors (1958 - 1964)**

- This generation was during the period 1958 to 1964.
- The semiconductor transistors replaced vacuum tubes. As already discussed the advantages of transistors over the vacuum tubes are as below :
  1. Transistors are smaller in size than vacuum tubes.
  2. Transistors are also cheaper than vacuum tubes.
  3. Transistors have very less heat dissipation and power consumption.

→ **(C) Generation 3 Integrated Chip (IC)**

- This was the generations from 1964 to 1974.
- Here multiple ICs were used that replaced huge transistor circuits into a single IC. Multiple ICs of this kind were required to make a CPU.

→ **(D) Generation 4 Microprocessors**

- This is the generation of electronic computers which had a single chip CPU.
- This reduced the size of the computer drastically as the entire CPU came on a single chip.
- Table 1.3.1 shows some of the Intel microprocessors with their special features.

**Table 1.3.1 : History of Intel processors**

| PROCESSOR | FEATURES | | | |
|---|---|---|---|---|
| | CLK SPEED | BUS WIDTH DATA | BUS WIDTH ADDRESS | OTHERS |
| 4004 | 740kHz | 4 bits | 4 bits | World's first microprocessor |
| 4040 | | 4 bits | 4 bits | Interrupts were introduced |
| 8008 | | 8 bits | 8 bits | 8-bit processor |
| 8080 | 2MHz | 8 bits | 16 bits | First general purpose processor |
| 8085 | 3MHz, 5Mhz | 8 bits | 8 bits | Serial Communication was introduced |
| 8086 | 5MHz, 8MHz, 10MHz | 16 bits | 20 bits | 1. Multi-bus compatible (multiprocessor configuration)<br>2. Introduction of pipelining<br>3. Concept of Memory Segmentation. |
| 8088 | | 8 bits | 20 bits | |
| 80186 | | | | It has two on-chip Timers, DMA Controller and Interrupt Controller |
| 80286 | 6MHz | 16 bits | 24 bits | Memory protection to support multi-tasking |
| 80386 | 16, 20, 25, 33 MHz | 32 bits | 32 bits | Concept of Virtual Mode |
| 80486 | 25, 33, 50 MHz | 32 bits | 32 bits | Level 1 cache on-chip |
| Pentium 1 | Upto 133 MHz | 64 bits | 32 bits | Superscalar architecture, etc. |
| Pentium 2 | Upto 300 MHz | 64 bits | 32 bits | Level 2 cache on chip |
| Pentium 3 | Upto 1.1 GHz | 64 bits | 32 bits | Faster cache |
| Pentium 4 | Upto 3.7 GHz | 64 bits | 32 bits | Hyper threading, etc. |
| Multi-core processors | Upto 3.7 GHz | 64 bits | 32 bits | Better performance for multiple task execution simultaneously. |

- Table 1.3.1 shows the list of microprocessors developed by Intel. Many other companies have manufactured microprocessors. Also the features listed are very few of them. Some of terminologies may not be understood by you, and it is out of the scope of this subject, but they are just listed for your reference.

## 1.4 Number Representation : Binary Data Representation, Two's Complement Representation and Floating - Point Representation

### 1.4.1 Simple Integer Representation

- Since it is binary only 0's and 1's are used to represent everything.
- Only positive numbers can be stored in binary using this method.

  **For example :** 41 = 00101001
- In this case no minus sign can be represented and hence we go for the other methods like signed magnitude, one's complement and two's compliment methods of representation.

### 1.4.2 Signed Magnitude Representation

- In the signed magnitude representation the MSB (Most Significant Bit or the first bit from left that has maximum weight) is used for sign, '1' indicating negative number and '0' indicating positive number.
- The remaining bits indicate magnitude. But in this form of representation '0' has two distinct representations of +0 and – 0, which is wrong according to mathematics.
- So we go for other methods of negative numbers representation namely the 1's and 2's complement methods as seen in the subsequent subsections.

### 1.4.3 One's Complement Method of Representation

We use 1's complement method to reduce the circuitry for addition/subtraction. We go for one's complement method of subtraction where we take one's complement and then add the two numbers. Also the advantage of this method is that it supports negative as well as positive numbers.

- To find ones complement in any base numbered system, we subtract it from the maximum number. For example in decimal subtract each digit by 9 and it is called as 9's complement; in octal, subtract each digit by 7 and it is called as 7's complement; in hexadecimal, subtract each digit by 15 (or F) and it is called as 15's complement; in binary, subtract each digit by 1 and it is called as 1's complement; but in binary number system this can be simply done by reversing the bits from 0 to 1 and vice-versa. Although different names according to the number system but same function.
- After getting one's complement of the negative numbers we can directly add it to the positive number, or if both numbers are negative we take one's complement of both the numbers and then add their complements to get the difference.
- The range of numbers that can be represented by a 1's complement method is given by : $(2^{n-1} - 1)$ to $+(2^{n-1} - 1)$.
- But in this method also we have '0' with two representation of + 0 and – 0, which is wrong. So we go for yet another method of negative number representation to be seen in next subsection.

### 1.4.4 Two's Complement Method of Representation

- The third method of representing signed numbers is 2's complement method. The major advantage of the 2's complement method over the signed magnitude method and the 1's complement method is that there are not two representations for '0' i.e. no separate representation for + 0 and – 0. Hence it is the most suited method in the microprocessors and digital computers for storing signed numbers.
- Also the operations like subtraction can be easily performed using the 2's complement method, almost similar to 1's complement method.
- To get the 2's complement of a number we need to first take 1's complement and then add 1 or subtract the number from the smallest number of one digit greater than the number whose 2's complement is to be found.
- Fig. 1.4.1 shows the 4-bit representations of 2's complement method.
- The range of numbers that can be represented by the 2's complement method is given by : $-(2^{n-1})$ to $+(2^{n-1} - 1)$.



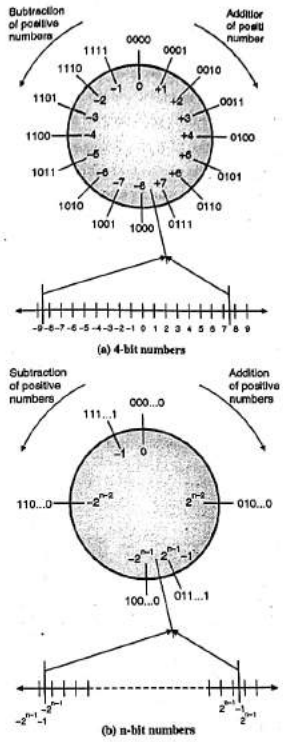


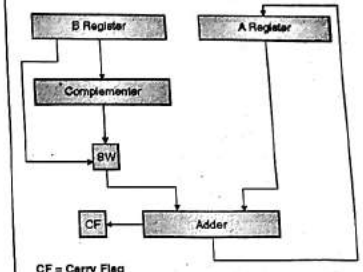

**Fig. 1.4.1 : 2's complement method of representation**

**Syllabus Topic : Integer Data Computation : Addition, Subtraction**

## 1.5 Integer Data Computation : Addition, Subtraction

**Q.** Explain various signed and unsigned number representations for integers. **(10 Marks)**

### 1.5.1 Integer Addition and Subtraction




CF = Carry Flag
SW = Switch (select addition subtraction)

**Fig. 1.5.1 : Adder / Subtractor circuit**

- For performing addition the normal binary addition is done as seen in chapter 3 using the adders seen in chapter 4. The adder circuit used is shown in Fig. 1.5.1. The carry is to be checked.
- For subtraction, we need to take 2's complement of subtrahend and add to minuend i.e. a – b = a + (– b), since 2's complement of a number gives its negative equivalent. So we only need addition and complement circuits to get the subtraction also implemented. In this case we need to monitor the borrow.
- Fig. 1.5.1 shows the hardware for addition and subtraction. It has an adder circuit, which gets one input from register 'A' and the result is also stored in this register. The other input comes either directly from register 'B' (in case of addition) or through the complement circuit (in case of subtraction) and hence both the operations are implemented by the same circuit. The switch is used to select the operation i.e. addition or subtraction and accordingly the correct operand (data on which the operation is to be performed) is passed to the adder circuit.
- The 'CF' in the Fig. 1.5.1 works as carry flag for addition and borrow flag for subtraction.

**Syllabus Topic : Multiplication : Unsigned Multiplication**

## 1.5.2 Multiplication : Unsigned Multiplication

**Q. Explain shift and add method of multiplication with hardware and flowchart. (10 Marks)**

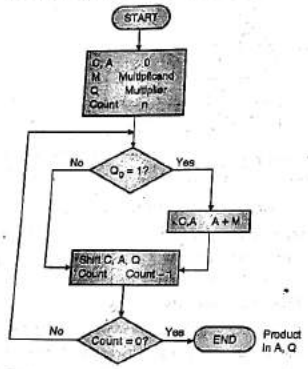



**Fig. 1.5.2 : Flowchart for shift and add method of multiplication**

- Multiplication can be done for the binary numbers using a special method called as shift and add method.
- In this case, each bit of the multiplier is checked and accordingly the multiplicand is added or not added. If a particular bit of the multiplier is '1', then the multiplicand is added to the MSB and then shifted.
- The checking of the multiplier bits begins from the LSB (Last Significant Bit) and hence the multiplicand added for this data is shifted and hence reaches to the last bit places in the result.
- This happens because the result is shifted after every addition of the multiplicand to the result, hence the multiplicand added for the first time i.e. for the LSB is shifted for maximum times and hence reaches to the last place in the result.
- The flowchart for this is shown in Fig. 1.5.2.
- This system requires some registers to store data like 'M' for storing multiplicand, 'Q' to store multiplier, 'A' which is initialized as zero (the result count that keeps the count of number of bits in input data to be multiplied and the single bit called 'C' which is the carry generated after addition.
- $Q_0$ refers to the LSB of the multiplier. This is checked to decide whether the multiplier is to be added or not. Addition is done because, if the checked bit the multiplier is '1' then when multiplied with multiplicand we get the multiplicand itself, and if bit is '0', then the result is zero, hence multiplicand not added in that case.
- After this the result is shifted right by one time so as arrange the bit position of the multiplicand added.
- The above process as shown in Fig. 1.5.2 is repeated for the count number of times. The result is finally available in the registers 'Q' and 'A'.
- The implementation of this can be implemented using the circuit as shown in Fig. 1.5.3.

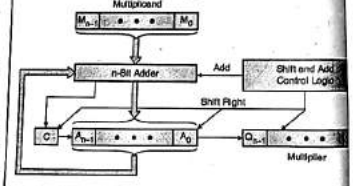



**Fig. 1.5.3 : Circuit implementation for shift and add method for multiplication**

- The registers 'A' and 'Q' along with the single bit 'C' are connected together for right shift operation as shown in the Fig. 1.5.3. The adder adds the current value in register 'A' with that of the multiplicand and the result is given back to the register 'A'.
- The control unit checks the value of the LSB bit of the register 'Q' i.e. the multiplier. Accordingly the addition and shift operation or simply the shift operation is controlled by the control unit.



- Some examples of shift and add method with the values of the registers after each step are given below :

**Ex. 1.5.1**

Multiply 13 × 11 using the shift and add method and give the values of all the registers after each step.

**Soln. :**

| C | A | Q | M | | |
|---|---|---|---|---|---|
| 0 | 0000 | 1101 | 1011 | Initial | Values |
| 0 | 1011 | 1101 | 1011 | Add | First Cycle |
| 0 | 0101 | 1110 | 1011 | Shift | |
| 0 | 0010 | 1111 | 1011 | Shift | Second Cycle |
| 0 | 1101 | 1111 | 1011 | Add | Third Cycle |
| 0 | 0110 | 1111 | 1011 | Shift | |
| 1 | 0001 | 1111 | 1011 | Add | Fourth Cycle |
| 0 | 1000 | 1111 | 1011 | Shift | |

$(10001111)_2 = (143)_{10}$ ...Ans.

**Ex. 1.5.2**

Explain how to multiply 1001 with 1101 in the register. (All are 5-bit registers)

**Soln. :**

| C | A | Q | M | | |
|---|---|---|---|---|---|
| 0 | 00000 | 01001 | 01101 | Initial values | |
| 0 | 01101 | 01001 | 01101 | First Cycle | |
| 0 | 00110 | 10100 | 01101 | | |
| 0 | 00011 | 01010 | 01101 | Second Cycle | |
| 0 | 00001 | 10101 | 01101 | Third Cycle | |
| 0 | 01110 | 10101 | 01101 | Fourth Cycle | |
| 0 | 00111 | 01010 | 01101 | | |
| 0 | 00011 | 10101 | 01101 | Fifth Cycle | |

$(0001110101)_2 = (117)_{10}$ ...Ans.

**Ex. 1.5.3**

Explain how to multiply 13 with 5 in the register.

**Soln. :**

| C | A | Q | M | |
|---|---|---|---|---|
| 0 | 0000 | 0101 | 1101 | Initial values |
| 0 | 1101 | 0101 | 1101 | First Cycle |
| 0 | 0110 | 1010 | 1101 | |
| 0 | 0011 | 0101 | 1101 | Second Cycle |
| 1 | 0000 | 0101 | 1101 | Third Cycle |
| 0 | 1000 | 0010 | 1101 | |
| 0 | 0100 | 0001 | 1101 | Fourth Cycle |

$(01000001)_2 = (65)_{10}$ ...Ans.

**Ex. 1.5.4**

Show multiplication of two numbers 13 and 14 using 4-bit registers.

**Soln. :**

M = 1101 (13)

| Carry | AC | Q | | |
|---|---|---|---|---|
| 0 | 0000 | 1110 | | Initial values |
| 0 | 0000 | 0111 | shift | 1st cycle since $Q_0$=0, no addition is required |
| 0 | 1101 | 0111 | AC = AC+M | Add(M) |
| 0 | 0110 | 1011 | shift | 2nd cycle since $Q_0$=1, at the end of first cycle, M is added |
| 1 | 0011 | 1011 | AC = AC+M | Add(M) |
| 0 | 0110 | 1101 | shift | 3rd cycle since $Q_0$=1, at the end of 2nd cycle, M is added |
| 1 | 0110 | 1101 | AC = AC+M | Add(M) |
| 0 | 1011 | 0110 | shift | 4th cycle since $Q_0$=1, at the end of 3rd cycle, M is added |

**Syllabus Topic : Booth's Algorithm**

## 1.5.3 Multiplication : Signed Multiplication: Booth's Algorithm

→ **(MU - May 2014, Dec. 2014, May 2015, Dec. 2015, May 2017)**

**Q. Draw the flow chart for Booth's Algorithm for Two's Complement Multiplication.**

**May 14, Dec. 14, May 15, Dec. 15, May 17. 5 Marks**

Q. Explain booth's principle (5 Marks)

Q. Explain the Booth's method of multiplying signed numbers with hardware and flowchart. (10 Marks)

- **Booth's principle states that** "The value of a series of 1's of binary can be given as the weight of the bit preceding the series minus the weight of the last bit in the series".

**For Example :**

1. $(011111111110)_2 = 2^{11} - 2^1$
$= 048 - 2$
$= (2046)_{10}$

- If this is the multiplier of multiplication, then in the shift and add method we would have to do 10 addition operations i.e. $2^{10} + 2^9 + 2^8 \ldots 2^1 = 2046$. But in case of Booth's algorithm we will have to do only 2 operations i.e. $2^{11} - 2^1$. Hence this is called as the best case condition of Booth's algorithm when the series of 1s is very large.

2. $(01100110)_2 = 2^7 - 2^5 + 2^3 - 2^1$
$= 128 - 32 + 8 - 2$
$= (102)_{10}$

- If this is the multiplier of multiplication, then in the shift and add method we would have to do 4 addition operations i.e. $2^6 + 2^5 + 2^4 + 2^1 = 102$. Also in case of Booth's algorithm we will have to do 4 operations i.e. $2^7 - 2^5 - 2^3 - 2^1$

3. $(01010101)_2 = 2^7 - 2^6 + 2^5 - 2^4 + 2^3 - 2^2 + 2^1 - 2^0$
$= 128 - 64 + 32 - 16 + 8 - 4 + 2 - 1$
$= (83)_{10}$

- If this is the multiplier of multiplication, then in the shift and add method we would have to do 4 addition operations i.e. $2^6 + 2^4 + 2^2 + 2^0 = 102$. But in case of Booth's algorithm we will have to do 8 operations.

i.e. $2^7 - 2^6 + 2^5 - 2^4 + 2^3 - 2^2 + 2^1 - 2^0$. Hence this is called as the worst case condition of Booth's algorithm wherein there are alternate zeros and ones.

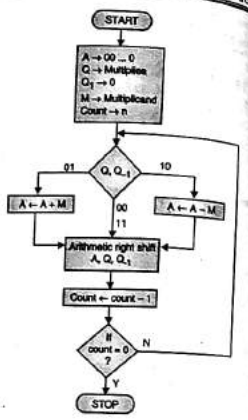



**Fig. 1.5.4**

- Fig. 1.5.4 shows the flowchart of Booth's algorithm. According to the booth's principle the bit preceding the series of 1s can be known if a zero followed by one (i.e. 01), hence we add when this is the case. Similarly the last bit in the series is when the one is followed by zero (i.e. 10), hence we subtract in this case. While for consecutive zeros or ones we have not to do any arithmetic operation.

**Ex. 1.5.5**

Multiply : $7 \times -3$, using Booth's Algorithm.

| A | Q | $Q_{-1}$ | m | Count |
|---|---|---|---|---|
| 0000 | 1101 | 0 | 0111 | 4 |
| +1001 | | | | |
| 1001 | 1101 | 0 | | |
| 1100 | 1110 | 1 | | 3 |
| +0111 | | | | |
| 0011 | 1110 | 1 | | |
| 0001 | 1111 | 0 | | 2 |
| 1001 | | | | |
| 1010 | 1111 | 0 | | |
| 1101 | 0111 | 1 | | 1 |
| 1110 | 1011 | 1 | | 0 |



**Soln. :**

$11101011 = -(00010101)_2$
$= -(21)_{10}$

**Ex. 1.5.6**

Multiply : $-7 \times -3$, using Booth's Algorithm.

| A | Q | $Q_{-1}$ | m | Count |
|---|---|---|---|---|
| 0000 | 1101 | 0 | 1001 | 4 |
| 0111 | | | | |
| 0111 | 1101 | 0 | | |
| 0011 | 1110 | 1 | | 3 |
| +1001 | | | | |
| 1100 | 1110 | 1 | | |
| 1110 | 0111 | 0 | | 2 |
| +0111 | | | | |
| 0101 | 0111 | 0 | | 0 |
| 0010 | 1011 | 1 | | 1 |
| 0001 | 0101 | 1 | | 0 |

**Soln. :**

$(00010101)_2 = (21)_{10}$
$13 = (01101)_2$
$11 = (01011)_2$
$-11 = (10101)_2$

**Ex. 1.5.7**

Multiply : $13 \times -11$, using Booth's Algorithm.

| A | Q | $Q_{-1}$ | m | Count |
|---|---|---|---|---|
| 00000 | 01101 | 0 | 10101 | 5 |
| + 01011 | | | | |
| 01011 | 01101 | 0 | | |
| 00101 | 10110 | 1 | | 4 |
| + 10101 | | | | |
| 11010 | 10110 | 1 | | |
| 11101 | 01011 | 0 | | 3 |
| + 01011 | | | | |
| 01000 | 01011 | 0 | | |
| 00100 | 00101 | 1 | | 2 |
| 00010 | 00010 | 1 | | 1 |
| + 10101 | | | | |
| 10111 | 00010 | 1 | | |
| 11011 | 10001 | 0 | | 0 |

**Soln. :**

$(11011\ 10001)_2$
$= -(00100\ 01111)_2$
$= 1 + 2 + 4 + 8 + 128 = -(143)_{10}$

**Ex. 1.5.8**

Multiply : $16 \times 15$, using Booth's Algorithm.

$16 = (010000)_2$
$15 = (001111)_2$
$-m = (110000)_2$

| A | Q | $Q_{-1}$ | m | Count |
|---|---|---|---|---|
| 000000 | 001111 | 0 | 010000 | 6 |
| + 110000 | | | | |
| 110000 | 001111 | 0 | | |
| 111000 | 000111 | 1 | | 5 |
| 111100 | 000011 | 1 | | 4 |
| 111110 | 000001 | 1 | | 3 |
| 111111 | 000000 | 1 | | 2 |
| + 010000 | 000000 | 1 | | |
| 001111 | 000000 | 1 | | |
| 000111 | 100000 | 0 | | 1 |
| 000011 | 110000 | 0 | | 0 |

**Soln. :**

$(000011110000)_2$
$= 16 + 32 + 64 + 128$
$= (240)_{10}$

**Ex. 1.5.9**

Explain Booth's algorithm to multiply the following pair of signed 2's complement numbers :

A = 110011 multiplicand

B = 101100 multiplier

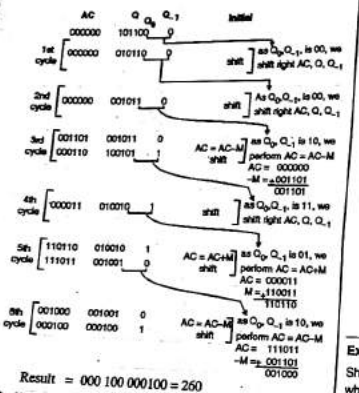
**Soln. :**

Multiplicand (M) = 110011

2's complement of multiplicand = 001101 = (– M)

Multiplier Q = 101100

Multiplication will require 6 cycles as the register size n = 6-bits.




Result = 000 100 000100 = 260

**Verification of result :**

$A = 110011, -A$

$= (2\text{'s complement of } A) = 001101$

$= (13)_{10}$

$\therefore A = 110011 = (-13)_{10}$

$B = 101100, -13$

$= (2\text{'s complement of } 13) = 010100$

$= (20)_{10}$

$\therefore B = 101100 = (-20)$

$\therefore A \times B = (-13) \times (-20) = 260$

**Ex. 1.5.10**

Using Booth's algorithm multiply the followings :

Multiplicand = + 15

Multiplier = − 6

**Soln. :**

Since, the representation of signed 15, requires 5 bits to represent (4 bits for magnitude, 1 bit for sign) :

Register size, n = 5

$(15)_{10} = (01111)_2$ Multiplicand (M)

$(-15)_{10} = (10001)_2$

2's complement of M = (− M)

$(6)_{10} = (00110)_2$

$\therefore (-6)_{10} = (11010)_2$ Multiplier (Q)

| AC | Q | $Q_{-1}$ | | |
|---|---|---|---|---|
| 00000 | 11010 | 0 | Initial | |
| 00000 | 01101 | 0 | Shift | 1st cycle |
| 10001 | 01101 | 0 | AC = AC − M | |
| 11000 | 10110 | 1 | Shift | 2nd cycle |
| 00111 | 10110 | 1 | AC = AC + M | |
| 00011 | 11011 | 0 | Shift | 3rd cycle |
| 10100 | 11011 | 0 | AC = AC − M | |
| 11010 | 01101 | 1 | Shift | 4th cycle |
| 11101 | 00110 | 1 | Shift | 5th cycle |

Result = 11101 00110

[1110100110 = − 0001011010 = − 90]

**Ex. 1.5.11**

Show the multiplication process using Booth's algorithm when the following binary numbers are multiplied.

(− 12) × (− 18)

**Soln. :**

$(12)_{10} = 001100$ [6 bit representation is assumed, − 18 requires 6 bits to represent]

$\therefore (-12)_{10} = 110100$ 2's complement of $(12)_{10}$

Multiplicand (m) = 110100

And − m = 001100

$(18)_{10} = 010010$

$\therefore (-18)_{10} = 101110$ 2's complement of $(18)_{10}$

Multiplier (Q) = 101110

| AC | Q | $Q_{-1}$ | | |
|---|---|---|---|---|
| 000000 | 101110 | 0 | Initial | |
| 000000 | 010111 | 0 | Shift | 1st cycle |
| 001100 | 010111 | 0 | AC=AC-M | |
| 000110 | 001011 | 1 | Shift | 2nd cycle |
| 000011 | 000101 | 1 | Shift | 3rd cycle |
| 000001 | 100010 | 1 | Shift | 4th cycle |
| 110101 | 100010 | 1 | AC=AC+M | |
| 111010 | 110001 | 0 | Shift | 5th cycle |
| 000110 | 110001 | 0 | AC=AC-M | |
| 000011 | 011000 | 1 | Shift | 6th cycle |

Product = 000011011000 = $(216)_{10}$



**Ex. 1.5.12**

Iterate Booth's multiplication algorithm for the product (+15) × (− 15).

**Soln. :**

$(15)_{10} = 01111$

$(-15)_{10} = 10001$

2's complements of $(15)_{10}$

Multiplicand (m) = 01111

And − m = 10001

Multiplier (Q) (−15) = 10001

| AC | Q | $Q_{-1}$ | | |
|---|---|---|---|---|
| 00000 | 10001 | 0 | Initial | |
| 10001 | 10001 | 0 | AC←AC−M | |
| 11000 | 11000 | 1 | Shift | 1st cycle |
| 00111 | 11000 | 1 | AC←AC+M | |
| 00011 | 11100 | 0 | Shift | 2nd cycle |
| 00001 | 11110 | 0 | Shift | 3rd cycle |
| 00000 | 11111 | 0 | Shift | 4th cycle |
| 10001 | 11111 | 0 | AC=AC−M | |
| 11000 | 11111 | 1 | Shift | 5th cycle |

Product = 1100011111 = (−225)$_{10}$

**Ex. 1.5.13**

Multiply - 9 and 7 using Booth's algorithm.

**Soln. :**

Multiplicand (M) = − 9

$(9)_{10} = (01001)_2$

2's complement of 9 = 10111

$\therefore$ M = 10111

and − M = 01001

Multiplier (Q) = $(7)_{10}$ = 00111

Multiplication will require 5 cycles as the register size n = 5 bits.

| AC | Q | $Q_{-1}$ | | |
|---|---|---|---|---|
| 00000 | 00111 | 0 | Initial | |
| 01001 | 00111 | 0 | AC = AC − M | as $Q_0 Q_{-1}$ is 10 |
| 00100 | 10011 | 1 | Shift | AC = 00000 |
| | | | | − M = 01001 |
| | | | | + |
| | | | | 01001 |
| 00010 | 01001 | 1 | Shift | as $Q_0 Q_{-1}$ is 11 |
| 00001 | 00100 | 1 | Shift | |
| 11000 | 00100 | 1 | AC = AC + M | AC = 00001 |
| 11100 | 00010 | 0 | Shift | M = 10111 |
| | | | | $Q_0 Q_{-1}$ = 01 |
| | | | | + |
| | | | | 11000 |
| 11110 | 00001 | 0 | Shift | $Q_0 Q_{-1}$ = 00 |

$\therefore$ Product = 1111000001

= − (0000111110) = − 63

**Ex. 1.5.14**

Show the multiplication process using Booth's algorithm when the following binary numbers are multiplied (− 13) × (− 6).

**Soln. :**

$(13)_{10} = 01101$

$\therefore (-13)_{10}$ = 2's complement of (01101) = 10011

$(6)_{10} = 00110$

$\therefore (-6)_{10}$ = 2's complement of (00110) = 11010

Multiplicand (M) = 10011

and − M = 01101

Multiplier (Q) = 11010

| AC | Q | $Q_{-1}$ | | |
|---|---|---|---|---|
| 00000 | 11010 | 0 | Initial | |
| 00000 | 01101 | 0 | Shift | $Q_0 Q_{-1}$ = 00 |
| 01101 | 01101 | 0 | AC = AC − M | $Q_0 Q_{-1}$ = 10 |
| 00110 | 10110 | 1 | Shift | AC = 00000 |
| | | | | − M = 01101 |
| | | | | + |
| | | | | 01101 |
| 11001 | 10110 | 1 | AC = AC + M | $Q_0 Q_{-1}$ = 01 |
| 11100 | 11011 | 0 | Shift | AC = 00110 |
| | | | | + M = 10011 |
| | | | | 11001 |
| 01001 | 11011 | 0 | AC = AC − M | AC = 11100 |
| 00100 | 11101 | 1 | Shift | − M = 01101 |
| | | | | 01001 |
| 00010 | 01110 | 1 | Shift | $Q_0 Q_{-1}$ = 11 |

$\therefore$ Result = 00010 01110 = 78

**Ex. 1.5.15**

Multiply : 7 and 14 using Booth's algorithm.

**Soln. :**

$(7)_{10} = 00111$

$\therefore (-7)_{10} = 11001$

$(14)_{10} = 01110$

Multiplicand (M) $= -7 = 11001$

and $-M = 00111$

Multiplier (Q) $= 01110$

| AC | Q | $Q_{-1}$ | | |
|---|---|---|---|---|
| 00000 | 01110 | 0 | Initial | |
| 00000 | 00111 | 0 | Shift | $Q_0, Q_{-1} = 00$ |
| 00111 | 00111 | 0 | AC = AC − M | $Q_0, Q_{-1} = 10$ |
| 00011 | 10011 | 1 | Shift | AC = 00000 |
| | | | | − M = 00111 |
| | | | | + |
| | | | | 00111 |
| 00001 | 11001 | 1 | Shift | $Q_0, Q_{-1} = 11$ |
| 00000 | 11100 | 1 | Shift | $Q_0, Q_{-1} = 11$ |
| 11001 | 11100 | 1 | AC = AC + M | $Q_0, Q_{-1} = 01$ |
| 11100 | 11110 | | Shift | AC = 00000 |
| | | | | M = 11001 |
| | | | | + |
| | | | | 11001 |

Product $= 11100\ 11110 = -(0001100010) = -98$

**Ex. 1.5.16** May 2014. 7 Marks

Using booth's algorithm shows the multiplication of 7 and 5.

**Soln. :**

| A | Q | $Q_{-1}$ | M | Count | Remark |
|---|---|---|---|---|---|
| 0000 | 0101 | 0 | 0111 | 4 | Initialization |
| + 1001 | | | | | |
| 1001 | 0101 | 0 | | | A ← A − M |
| 1100 | 1010 | 1 | | 3 | Arithmetic Right shift |
| + 0111 | | | | | |
| 0011 | 1010 | | | | A ← A + M |
| 0001 | 1101 | 0 | | 2 | Arithmetic Right shift |
| + 1001 | | | | | |
| 1010 | 1101 | | | | A ← A − M |
| 1101 | 0110 | 1 | | 1 | Arithmetic Right shift |
| + 0111 | | | | | |
| 0100 | 0110 | | | | A ← A + M |
| 0010 | 0011 | 0 | | 0 | Arithmetic right shift |

$(00100011)_2 = (35)_{10}$ ...Ans.

**Ex. 1.5.17** Dec. 2014. 10 Marks

Multiply $(-2)_{10}$ and $(-5)_{10}$ using Booth's Algorithm.

**Soln. :**

| A | Q | $Q_{-1}$ | M | Count | Remark |
|---|---|---|---|---|---|
| 0000 | 1011 | 0 | 1110 | 4 | Initialization |
| + 0010 | | | | | |
| 0010 | 1011 | | | | A ← A − M |
| 0001 | 0101 | 1 | | 3 | Arithmetic right shift |
| 0000 | 1010 | 1 | | 2 | Arithmetic right shift |
| + 1110 | | | | | |
| 1110 | 1010 | | | | A ← A + M |
| 1111 | 0101 | 0 | | 1 | Arithmetic right shift |
| + 0010 | | | | | |
| 0001 | 0101 | 0 | | | A ← A − M |
| 0000 | 1010 | 1 | | 0 | Arithmetic right shift |

$(00001010)_2 = (10)_{10}$ ...Ans.

**Ex. 1.5.18** Dec. 2015. 5 Marks

Using Booth's algorithm show the multiplication of and − 7.

**Soln. :**

| A | Q | $Q_{-1}$ | M | Count | Remark |
|---|---|---|---|---|---|
| 0000 | 1001 | 0 | 1101 | 4 | Initialization |
| 0011 | | | | | A ← A − M |
| 0011 | 1001 | | | | |
| 0001 | 1100 | 1 | | 3 | Arithmetic right shift |
| + 1101 | | | | | |
| 1110 | 1100 | | | | A ← A + M |
| 1111 | 0110 | 0 | | 2 | Arithmetic right shift |
| 1111 | 1011 | 0 | | 1 | Arithmetic right shift |
| + 0011 | | | | | |
| 0010 | 1011 | | | | A ← A − M |
| 0001 | 0101 | 1 | | 0 | Arithmetic right shift |

$(00010101)_2 = (21)_{10}$ ...Ans.

**Ex. 1.5.19** May 2016. 10 Marks

Multiply (− 10) and (− 4) using Booth's algorithm.



**Soln. :**

| A | Q | $Q_{-1}$ | M | Count | Remark |
|---|---|---|---|---|---|
| 00000 | 11100 | 0 | 10110 | 5 | Initialization |
| 00000 | 01110 | 0 | | 4 | Arithmetic Right shift |
| 00000 | 00111 | 0 | | 3 | Arithmetic Right shift |
| + 01010 | | | | | |
| 01010 | 00111 | | | | A ← A − M |
| 00101 | 00011 | 1 | | 2 | Arithmetic Right shift |
| 00010 | 10001 | 1 | | 1 | Arithmetic Right shift |
| 00001 | 01000 | 1 | | 0 | Arithmetic Right shift |

$(0000101000)_2 = (40)_{10}$ ...Ans.

**Ex. 1.5.20** Dec. 2016. 10 Marks

Multiply (− 7) with (4) by using Booth's algorithm of Multiplication.

**Soln. :**

$-7 \times 4$

| A | Q | $Q_{-1}$ | M | Count | Remark |
|---|---|---|---|---|---|
| 0000 | 0100 | 0 | 1001 | 4 | Initialization |
| 0000 | 0010 | 0 | | 3 | Arithmetic Right shift |
| 0000 | 0001 | 0 | | 2 | Arithmetic Right shift |
| + 0111 | | | | | |
| 0111 | 0001 | | | | A ← A − M |
| 0011 | 1000 | 1 | | 1 | Arithmetic Right shift |
| + 1001 | | | | | |
| 1100 | 1000 | | | | A ← A − M |
| + 1110 | 0100 | 0 | | 0 | Arithmetic Right shift |

$(11100100)$

$= -(00011100)_2$

$= -(28)_{10}$ ...Ans.

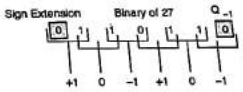
### 1.5.4 Bit-pair Recoding of Multipliers (A Fast Multiplication Method)

Above technique halves the maximum number of summands. Bit-pair recording technique is derived directly from the Booth's algorithm. In Booth's algorithm, multiplier is examined two bits at a time, starting from the right. Table given below gives, operation based on pair of bits.

| $Q_0$ | $Q_{-1}$ | Operation | Notation | |
|---|---|---|---|---|
| 0 | 0 | No add/subtract | 0 | '0' stands for no operation. + 1 stands for addition and − 1 stands for subtraction. |
| 1 | 1 | No add/subtract | 0 | |
| 0 | 1 | Add | + 1 | |
| 1 | 0 | Subtract | − 1 | |

**Fig. 1.5.5 : Operations as per Booth's algorithm**

Multiplier (Q) = 27 = $(11011)_2$




(oo 2.21)**Fig. 1.5.6 : Operations for the multiplier = 27, using 0, + 1 and − 1 notation**

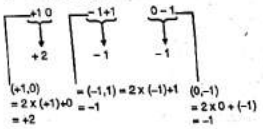
- In bit-pair recording, two adjacent operations of Booth's algorithm are combined together.
- The weight of the $i^{th}$ bit of a binary number is 2 times the weight of the $(i-1)^{th}$ bit.
- Let us try to understand operations + 2, − 2, + 1, − 1, 0 on the data M = 9 and register size n = 5 bits.

$M = 9 = (01001)_2 =$ 00000 01001

10 bit representation of '9'

as the register size = 5

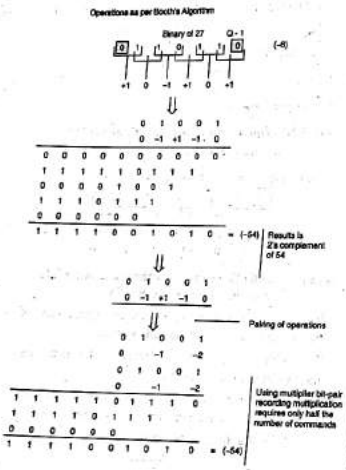
$-9$ = 2's complement of 9 = 11111 10111




| Operation | Value (9) | Multiplicand |
|---|---|---|
| 0 | 00000 00000 | |
| + 1 | 00000 01001 | |
| − 1 | 11111 10111 | |
| + 2 | 00000 10010 | ← data is left shifted |
| − 2 | 11111 01110 | ← − 9 is left shifted |

$01001\ (+9)$ $(6)_{10} = (00110)_2$

$\times\ 11010\ (-6)$ $\therefore (-6)_{10} = (11010)_2$




### 1.5.5 Hardware Implementation of Booth Algorithm

**Q. Explain with hardware requirement how to add and subtract integer numbers. (5 Marks)**

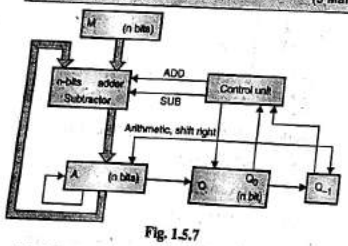



**Fig. 1.5.7**

- The Fig 1.5.7 shows the hardware implementation of the Booth's algorithm. The accumulator or the A register and the Q register along with the $Q_{-1}$ bit are combined together as a right shift register. The control unit checks the $Q_0$ and $Q_{-1}$ bits to decide whether to add, subtract or directly shift. Arithmetic shift right is implemented to maintain the sign. The register A holds the multiplicand while the register Q holds the multiplier. Register A and $Q_{-1}$ are initialized to zero.

**Syllabus Topic : Division of Integers : Restoring**

## 1.6 Division of Integers : Restoring Method

### 1.6.1 Restoring Division Method

➔ **(MU - May 20**

**Q. Draw the flow chart for restore division algorithm. May 17, 4 Mar**

**Q. Divide using restore division method 7/3. May 17, 6 Mar**

**Q. Explain restoring method of division with flowchart. (5 Mar**

- Similar to the multiplication algorithm to multiply binary numbers, we also have a method for division called as the restoring method of division for binary numbers.
- Fig. 1.6.1 shows the flowchart for the restoring method of division. Here also we have the registers namely 'A', 'M', 'Q' and count to store the result, dividend, divisor and the count respectively.
- In this case we shift left the registers 'A' and 'Q' to their left, and then check whether the value in 'A' is greater than the divisor or not. This is done by subtracting the divisor from the value of register 'A'.
- To find out whether greater or not we check the result is positive or not. If yes then we put '1' in the LSB of the Q register, which was initially left blank while shifting. If no, then we put a '0' in the LSB of the Q register and add the divisor back to the value of register 'A' to restore the previous value of register 'A', hence the name "Restoring Division method".
- The count is decremented and the above process is repeated until the count is not equal to zero.
- Let us see some examples based on restoring method for division.



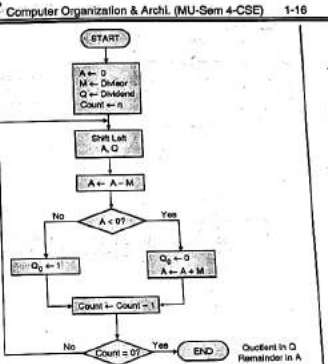



**Fig. 1.6.1 : Flowchart for the restoring method of division**

**Ex. 1.6.1**

Divide 13 / 5 using the restoring method of division and give the values of all the registers after each step.

**Soln. :**

**Note :**
1. For subtraction of divisor, we will add the 2's complement of the divisor. Since the divisor is 00101 in this case, the 2's complement will be 11011.
2. From the MSB of the result we will come to know whether the result is positive or negative. If the MSB is '1', the result is negative and if the MSB is '0', the result is positive.

| A | Q | M | |
|---|---|---|---|
| 00000 | 01101 | 00101 | Initial values |
| 00000 | 1101_ | 00101 | |
| + 11011 | | | |
| 11011 | 11010 | 00101 | First cycle |
| + 00101 | | | |
| 00000 | | | |
| 00001 | 1010_ | 00101 | |
| + 11011 | | | |
| 11100 | 10100 | 00101 | Second cycle |
| + 00101 | | | |
| 00001 | | | |
| 00011 | 0100_ | 00101 | |
| + 11011 | | | |
| 11110 | 01000 | 00101 | Third cycle |
| + 00101 | | | |
| 00011 | | | |
| 00110 | 1000_ | 00101 | |
| + 11011 | | | |
| 00001 | 10001 | 00101 | Fourth cycle |
| 00011 | 0001_ | 00101 | |
| + 11011 | | | |
| 11110 | 00010 | 00101 | |
| + 00101 | | | Fifth cycle |
| 00011 | | | |

Answer = > Quotient = $(00010)_2 = (2)_{10}$ (from register 'Q')

Remainder = $(00011)_2 = (3)_{10}$ (from register 'A')

**Ex. 1.6.2**

Explain how to divide 13 by 3 in the registers and showing how the quotient and remainder are placed after the division (all are 5 bit registers).

**Soln. :**

**Note :**
1. For subtraction of divisor, we will add the 2's complement of the divisor. Since the divisor is 00011 in this case, the 2's complement will be 11101.
2. From the MSB of the result we will come to know whether the result is positive or negative. If the MSB is '1', the result is negative and if the MSB is '0', the result is positive.

| A | Q | M | |
|---|---|---|---|
| 00000 | 01101 | 00111 | Initial values |
| 00000 | 1101_ | 00011 | |
| + 11101 | | | |
| 11101 | 11010 | 00011 | First cycle |
| + 00011 | | | |
| 00000 | | | |
| 00001 | 1010_ | 00011 | |
| + 11101 | | | |
| 11110 | 10100 | 00011 | Second cycle |
| + 00011 | | | |
| 00001 | | | |

```
  00011  0100_  00011
+ 11101                 } Third cycle
  00000  01001  00011

  00000  1001_  00011
+ 11101
  11101  10010  00011   } Fourth cycle
+ 00011
  00000

  00001  0010_  00011
+ 11101
  11110  00100  00011   } Fifth cycle
+ 00101
  00001
```

Answer => Quotient = $(00100)_2 = (4)_{10}$ (from register 'Q')

Remainder = $(00001)_2 = (1)_{10}$ (from register 'A')

**Ex. 1.6.3**

Perform division of the following positive numbers using restoring division.

1000 + 11, register size, n = 4 bits.

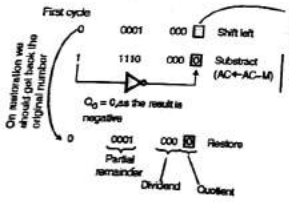
**Soln. :**

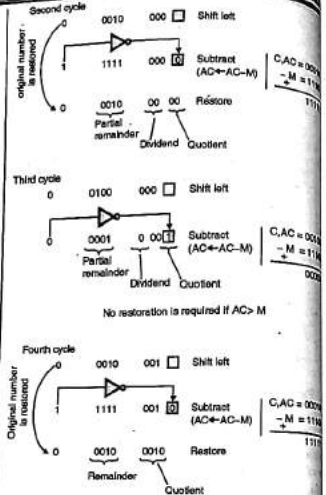
Divisor M = 0 0 0 1 1 and −M = 1 1 1 0 1

(n + 1 bits) (n + 1 bits)

One additional bit has been taken to handle borrow. Borrow (C) will be set whenever M > AC and we perform AC = AC − M.

| C | AC | Q (Dividend) | |
|---|---|---|---|
| 0 | 0000 | 1000 | Initial values |




**Ex. 1.6.4**

Perform division of the following numbers using restoring division,

Dividend = 17; Divisor = 03

**Soln. :**

Dividend Q = 17 = $(10001)_2$,

n = 5 as 17 requires 5 bits of storage

Divisor M = 03 = 00011

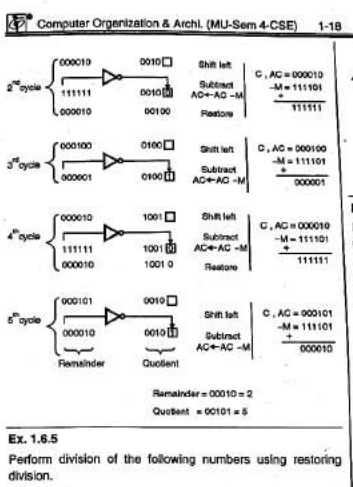
n + 1 bits – one additional bit to handle borrow

−M = 11101

| | C | AC | Q | |
|---|---|---|---|---|
| | 0 | 00000 | 10001 | ← Initial |
| 1st cycle | 0 | 00001 | 0001 | Shift left |
| | 1 | 11110 | 0001 | Subtract AC←AC −M |
| | 0 | 00001 | 00010 | Restore |

C, AC = 00001 / −M = 11101 / 11110






**Ex. 1.6.5**

Perform division of the following numbers using restoring division.

1100 ÷ 11

**Soln. :**

DividendQ = 1100

Divisor M = 00011

n + 1 bits ← one additional bit to handle borrow

−M = 11101

−M = 11101

| | C | AC | Q | |
|---|---|---|---|---|
| | 0 | 0000 | 1100 | - Intial |
| 1st cycle | 0 | 0001 | 100 | Shift left |
| | 1 | 1110 | 100 | Subtract AC←AC − M |
| | 0 | 0001 | 100 | Restore |
| 2nd cycle | 0 | 0011 | 000 | Shift left |
| | 0 | 0000 | 000 | subtract AC←AC − M |
| 3rd cycle | 0 | 0000 | 001 | Shift left |
| | 1 | 1101 | 001 | Subtract AC←AC − M |
| | 0 | 0000 | 001 | Restore |
| 4th cycle | 0 | 0000 | 010 | Shift left |
| | 1 | 1101 | 010 | Subtract AC←AC − M |
| | 0 | 0000 | 010 | Restore |

C, AC = 00001 / −M = 11101 / 11110

C, AC = 00011 / −M = 11101 / 00000

C, AC = 00000 / −M = 11101 / 11101

Remainder = 0000

Quotient = 0011

**Ex. 1.6.6**

Perform division of the following numbers using restoring division algorithm.

A : 1100 B : 0100

**Soln. :**

Dividend Q = 1100

Divisor M = 0 0100

n + 1 bits – one additional bit to handle borrow

−M = 11100

| | C | AC | Q | |
|---|---|---|---|---|
| | 0 | 0000 | 1100 | - Intial |
| 1st cycle | 0 | 0001 | 100 | Shift left |
| | 1 | 1101 | 100 | Subtract AC←AC − M |
| | 0 | 0001 | 100 | Restore AC←AC −M |
| 2nd cycle | 0 | 0011 | 000 | Shift left |
| | 1 | 1111 | 000 | Subtract AC←AC −M |
| | 0 | 0011 | 000 | Restore |
| 3rd cycle | 0 | 0110 | 000 | Shift left |
| | 0 | 0010 | 000 | Subtract AC←AC −M |
| 4th cycle | 0 | 0100 | 001 | Shift left |
| | 1 | 0000 | 001 | Subtract AC←AC −M |
| | Remainder | | Quotient | |

C, AC = 00001 / −M = 11100 / 11101

C, AC = 00011 / −M = 11100 / 11111

C, AC = 00110 / −M = 11100 / 00010

C, AC = 00100 / −M = 11100 / 00000

Remainder = 0000 = 0

Quotient = 0100 = 4

**Ex. 1.6.7**

Perform division of the following numbers using restoring division.

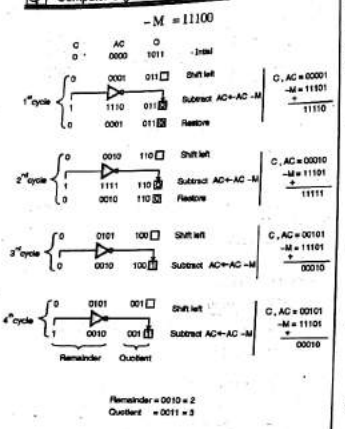
Dividend 1011

Divisor 0011

**Soln. :** Dividend Q = 1011

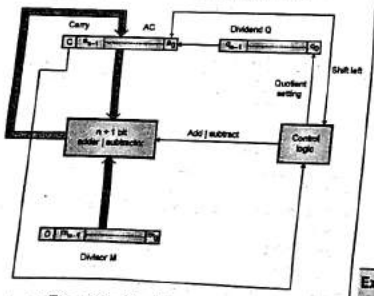
Divisor M = 0 0011,

n + 1 bits – one additional bit to handle borrow




### Hardware implementation of binary division

- An n-bit positive divisor is loaded into register M.
- An n-bit positive dividend is loaded into register Q.
- Register AC(A) is set to 0.
- Initial carry C is set to 0.
- After the division is complete, the n-bit quotient is in register Q and the remainder is in register AC.




(oo 2.40)**Fig. 1.6.2 : Circuit for binary division [Both restoring and non-restoring]**

**Ex. 1.6.8**

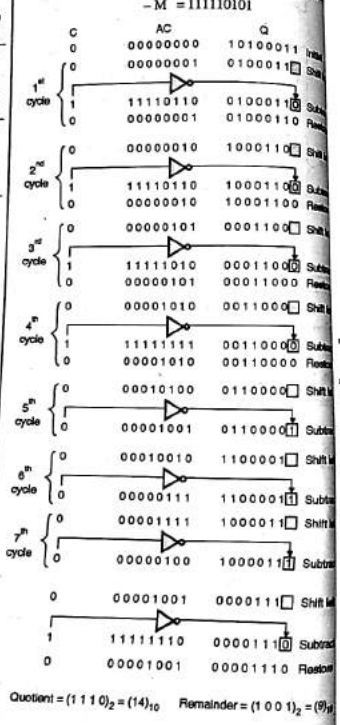
Explain and solve the following problem using by ... division algorithm ? Hence divide $(163)_{10}$ with $(11)_{10}$.

**Soln. :**

Dividend Q $= (163)_{10} = 10100011$

Divisor M $= (11)_{10} = \underbrace{00000}_{\text{n+1 bits} \leftarrow \text{one additional bit to handle b...}}1011$

$-M = 111110101$




Quotient = $(1\ 1\ 1\ 0)_2 = (14)_{10}$ Remainder = $(1\ 0\ 0\ 1)_2 = (9)_{10}$

**Ex. 1.6.9** May 2015. 6 Marks

Using unsigned Binary Division method, divide 7 by 3.



**Soln. :**

| A | Q | M | Count | Remark |
|---|---|---|---|---|
| 0000 | 0111 | 0011 | 4 | Initialization |
| 0000 | 1110 | | | Left shift |
| + 1101 | 1011 | | | |
| 1101 | | | | A ← A − M |
| + 0011 | | | | |
| 0000 | 1110 | | 3 | A ← A + M |
| 0001 | 1100 | | | Left shift |
| + 1101 | | | | |
| 1110 | | | | A ← A − M |
| + 0011 | | | | |
| 0001 | 1100 | | 2 | A ← A + M |
| 0011 | 1001 | | | Left shift |
| + 1101 | | | | |
| 0000 | 1001 | | 1 | A ← A − M |
| 0001 | 0010 | | | Left shift |
| + 1101 | | | | |
| 1110 | | | | A ← A − M |
| + 0011 | | | | |
| 0001 | 0010 | | 0 | A ← A + M |

Quotient : $(0010)_2 = (2)_{10}$
Remainder : $(0001)_2 = (1)_{10}$ ...Ans.

**Syllabus Topic : Division of Integers : Non-restoring Method**

## 1.7 Division of Integers: Non-restoring Method

**Q.** Explain non-restoring method of division with flowchart. **(5 Marks)**

The algorithm of restoring division can be improved by avoiding restoring after an unsuccessful subtraction. Subtraction is said to be unsuccessful if the result is negative.

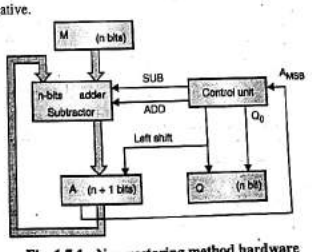



**Fig. 1.7.1 : Non-restoring method hardware implantation**

**Flowchart - non restoring method**

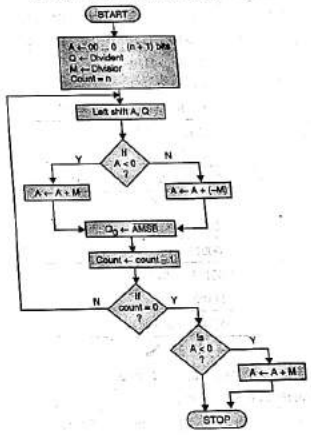



**Fig. 1.7.2 : Flowchart of non restoring method**

Flowchart for non-restoring division is given in the Fig. 1.7.2 Logic circuit for both restoring and non-restoring division can be handled on the circuit given in Fig. 1.7.1.

### Algorithm for non-restoring division

**Step 1 : Do the following n times :**

If the sign of AC is positive (C = 0), then shift C, AC and Q left one bit position and subtract M from AC.

else

Shift C, AC and Q left one bit position and add M to AC.

If the sign of AC is positive then

Set $q_0$ to 1

else

Set $q_0$ to 0

**Step 2 : If the sign of AC is negative then add M to AC.**

Step 2 is needed to leave the proper positive remainder in AC at the end of n cycles.

**Example 1 :** Perform 15/4.

| A | Q | m | Count |
|---|---|---|---|
| 00000 | 1111 | 00100 | 4 |
| 00001 | 111$\underline{0}$ | | |
| 11100 | | | |
| 11101 | 1110 | | 3 |
| 11011 | 110$\underline{0}$ | | |
| 00100 | | | |
| 11111 | 1100 | | 2 |
| 11111 | 100$\underline{1}$ | | |
| 00100 | | | |
| 00011 | 1001 | | 1 |
| 00111 | 001$\underline{1}$ | | |
| 11100 | | | |
| 00011 | 0011 | | 0 |

Remainder Quotient

**Example 2 :** Perform 10/4.

| A | Q | m | Count |
|---|---|---|---|
| 00000 | 1010 | 00100 | 4 |
| 00001 | 010$\underline{0}$ | | |
| 11100 | | | |
| 11101 | 0100 | | 3 |
| 11010 | 1000 | | |
| 00100 | | | |
| 11110 | 1000 | | 2 |
| 11101 | 000$\underline{1}$ | | |
| 00100 | | | |
| 00001 | 0001 | | 1 |
| 00010 | 001$\underline{0}$ | | |
| 11100 | | | |
| 11110 | 0010 | | |
| − 00100 | | | |
| 00010 | 0010 | | 0 |

Remainder Quotient

**Example 3 :** Perform 12/3.

$-m = 11101$

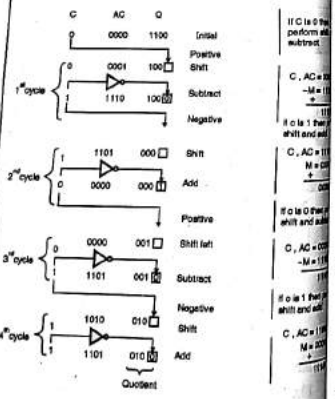
| A | Q | m | Count |
|---|---|---|---|
| 00000 | 1100 | 00011 | 4 |
| 00001 | 100$\underline{0}$ | | |
| 11101 | | | |
| 11110 | 1000 | | 3 |
| 11101 | 000$\underline{1}$ | | |
| 00011 | | | |
| 00000 | 0001 | | 2 |
| 00000 | 001$\underline{0}$ | | |
| 11101 | | | |
| 11101 | 0010 | | 1 |
| 11010 | 010$\underline{0}$ | | |
| 00011 | | | |
| 11101 | 0100 | | 0 |
| 00011 | | | |
| 00000 | 0100 | | |

Remainder Quotient

**Ex. 1.7.1**

Perform division of the following numbers using restoring division. 1100 ÷ 11

**Soln. :**






Dividend Q = 1100

Divisor M = 0 0011,

n + 1 bits – one additional bit to handle carry/borrow/sign

$-M = 11101$

Since, at the end of 4(a) cycles, AC is negative. We perform the operation :

$AC \leftarrow AC + M$

C, AC = 11101

M = 00011

00000 ← Remainder

∴ Remainder = 0000

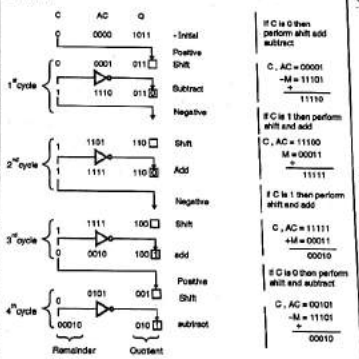
Quotient = 0100 = 4

**Ex. 1.7.2**

Perform division of the following numbers using non-restoring division algorithm.

Dividend = 1011; Divisor = 0011

**Soln. :**




Dividend (Q) = 1011

Divisor (M) = 00011

n + 1 bits – one additional bit to handle carry/borrow/sign

$-M = 11101$

Quotient = 0011 = 3

Remainder = 0010 = 2

**Ex. 1.7.3**

Explain and solve the following problem using non-restoring division algorithm. Hence divide $(10)_{10}$ with $(3)_{10}$.

**Soln. :**

Non-restoring division technique has been explained in the section 1.7.

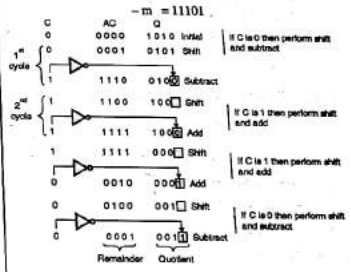
**Division of $(10)_{10}$ with $(3)_{10}$ :**

Dividend (Q) = 1010

Divisor (M) = 00011

n + 1 bits – one additional bit to handle carry/borrow/sign

$-m = 11101$




Quotient = 0011 = 3 Remainder = 0001 = $(1)_{10}$

**Syllabus Topic : Floating-Point Representation : IEEE 754 Floating Point Number Representation**

## 1.8 Floating-Point Representation : Basics of Floating Point Representation IEEE 754 Floating Point (Single and Double Precision) Number Representation

→ **(MU - May 14, May 15,Dec. 15, May 17)**

Q. Show IEEE 754 standards for binary floating-point representation for 32 bit single format and 64 bit double format. **May 14, Dec 15, 5 Marks**

Q. Explain IEEE 754 standards for Floating Point number representation. **May 15, 6 Marks**

Q. Show IEEE 754 standards for binary floating point representation for 32 bit single format and 64 bit double format. **May 17, 10 Marks**

The range of numbers that can be represented by a fixed-point number is insufficient for many applications. In scientific applications, very large and very small numbers are encountered. Scientific notation permits us to represent such numbers using relatively few digits.

For example,

$2.5 \times 10^{10}$

Represents a fixed point integer 25000000000.

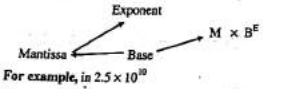
The floating-point codes used in computers are represented in binary.

### Format of representation

Three numbers are associated with a floating point number :

(1) A mantissa (M)
(2) An exponent (E)
(3) Base (B)

The mantissa M is also referred to as the significant or fraction. These three components together represent the real number.




**For example,** in $2.5 \times 10^{10}$

Mantissa = 2.5
Exponent = 10
Base = 10

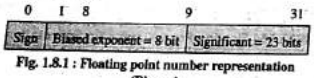



**Fig. 1.8.1 : Floating point number representation (Binary)**

In a typical representation of a floating point number :

(1) Exponent is biased.
(2) Mantissa or significant is normalized.

### (1) Biased exponent

Exponent is represented using an excess -128 code. An 8 bit binary number represents values from 0 to 255. However, as we are adding 128 in the biased exponent, the actual exponent values represented will be – 128 to 127.

| Exponent value | Exponent value after biasing (+128) |
|---|---|
| – 128 | $0\ (000\ 00000)_2$ |
| – 127 | $1\ (0000\ 0001)_2$ |
| 0 | $128\ (1000\ 0000)_2$ |
| 127 | $255\ (1111\ 1111)_2$ |

### Advantages of normalization

Very large or small exponents can easily represented.

**For example,**

If the exponents are in the range – 1050 to – 900 can select the biasing factor as 1050. With 1050 as bias

– 1050 will be represented as – 1050 + 1050 = 0
– 900 will be represented as – 900 + 1050 = 150

### (2) Normalized mantissa

A binary floating point number is represented normalized form, that is, the number is of the form (Significant starting with non-zero bit) X 2 ± (exp value).

**For example,**

| Binary number | Its normal form |
|---|---|
| 11.101 | $.11101 \times 2^2$ |
| .00101 | $.101 \times 2^{-2}$ |
| $1.01 \times 2^1$ | $.101 \times 2^6$ |

### Advantages of normalization

As for a normalized mantissa, the left most bit ca be zero, therefore, it has to be 1. Thus, it is not necessary store this first bit and it is assumed implicitly the number. Therefore, a 23 bit mantissa can repre 23 + 1 = 24 bit significant.

**Ex. 1.8.1**

Represent $(13.54)_{10}$ in 32 bit register with

Exponent = 8 bits

Mantissa = 23 bits

Exponent is biased with biasing of 128 and Mantissa normalised.

**Soln. :**

**Step 1 : Converting 13.54 to its equivalent binary form**

13 = 1101
.54 × 2 = 1.08
.08 × 2 = 0.16
.16 × 2 = 0.32
.32 × 2 = 0.64



.64 × 2 = 1.28
.28 × 2 = 0.56
.56 × 2 = 1.12
.12 × 2 = 0.24
.24 × 2 = 0.48
.48 × 2 = 0.96
.96 × 2 = 1.92
.92 × 2 = 1.84
.84 × 2 = 1.68
.68 × 2 = 1.36
.36 × 2 = 0.72
.72 × 2 = 1.44
.44 × 2 = 0.88
.88 × 2 = 1.76
.76 × 2 = 1.52
.52 × 2 = 1.04
.04 × 2 = 0.08
.08 × 2 = 0.16
.16 × 2 = 0.32
.32 × 2 = 0.64

**Step 2 : Biasing :**

$\therefore$ 13.54 = 1101.10001010001111010111000000
= .11011000101000111101011100000 × $2^4$

Exponent = 4

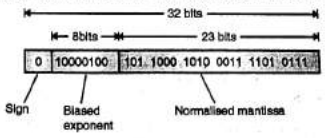
Exponent after biasing = 128 + 4
= 132
$(132)_{10} = (10000100)_2$

**Step 3 : Normalization :**

Mantissa = .1101 1000 1010 0011 1101 0111

First bit is not stored as it is assumed to be implicit (hidden).

Mantissa after normalization = .101 1000 1010 0011 1101 0111




Representation of $(13.54)_{10}$ in floating point format.

Minimum value of significant :

The implicit first bit as 1 followed by 23 0's.

.1000 0000 0000 0000 0000 0000 (1 is hidden)

Decimal equivalent = $1 \times 2^{-1} = 0.5$

Maximum value of significant :

The implicit first bit 1 followed by 23 1's.

.1111 1111 1111 1111 1111 1111

Decimal equivalent :

Binary : 0.1111 1111 1111 1111 1111 1111
+ 0.0000 0000 0000 0000 0000 0001 = $2^{-24}$
1.0000 0000 0000 0000 0000 0000

$\therefore$ Maximum value of significant = $1 - 2^{-24}$

Therefore, in normalized mantissa and biased exponent form, the format of Fig. Ex. 1.8.1 can represent binary floating point number in the range.

Lowest negative number : Maximum significant and maximum exponent

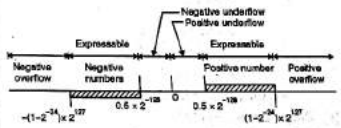
$= -(1 - 2^{-24}) \times 2^{127}$

Highest negative numbers = Minimum significant and minimum exponent

$= -0.5 \times 2^{-128}$

Lowest positive number : $0.5 \times 10^{-128}$

Highest positive number : $(1 - 2^{-24}) \times 10^{127}$




(oo 2.50) **Fig. Ex. 1.8.1 : Binary floating point number range for 32-bit format**

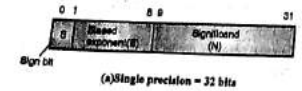
### Disadvantages of normalization

(1) From the Fig. Ex. 1.8.1, it is clear that 0 cannot be represented.
(2) A number in the range $0.5 \times 2^{-128} <> -0.5 \times 2^{-128}$ cannot be represented as it will cause an underflow.
(3) $\infty$ can not be represented.

### 1.8.1 IEEE-754 Standard for Representing Floating Point Numbers

**Q. Explain the IEEE 754 standard for representing floating point numbers. (10 Marks)**

- Representation of floating point number discussed in section 1.8 has many subtle problems.
- IEEE floating point standards addresses a number of such problems.
- Zero has definite representation in IEEE format.
- ± ∞ has been represented in IEEE format. A + ∞ indicated that the result of an arithmetic expression is too large to be stored.
- If an underflow occurs, implying that a result is too small to be represented as a normalized number, it is encoded in a denormalized scale.
- Fig. 1.8.2 gives the representation of floating point numbers.




(a)Single precision = 32 bits


(b)Double precision = 64 bits

- base = 2
- Significand is in normalized form i.e the first bit is 1 and it is hidden.
- S is sign bit.

(∞ 2.11) **Fig. 1.8.2 : IEEE standard format**

**(a) Single precision (32 bits)**

| | Exponent (E) | Significant (N) | Value/comments |
|---|---|---|---|
| | 255 | Not equal to 0 | Does not represent a number |
| | 255 | 0 | − ∞ or + ∞ depending on sign bit |
| Normalized scale | 0 < E < 255 | Any | ± (1.N) $2^{E-127}$ |
| Denormalized scale | 0 | Not equal to 0 | ± (0.N) $2^{-126}$ |
| | 0 | 0 | ± 0 depending on sign bit |

**(b) Double precision (64 bits)**

| | Exponent (E) | Significant (N) | Value/comments |
|---|---|---|---|
| | 2047 | Not equal to 0 | Does not represent a number |
| | 2047 | 0 | − ∞ or + ∞ depending on sign bit |
| Normalized scale | 0 < E < 2047 | Any | ± (1.N) $2^{E-1023}$ |
| Denormalized scale | 0 | Not equal to 0 | ± (0.N) $2^{-1022}$ |
| | 0 | 0 | ± 0 depending on sign bit |

**Fig. 1.8.3 : Values of floating point numbers as per IEEE format**

**Example (1) :** Represent $(200.625)_{10}$ in both single precision as well as double precision

**Step 1 : Convert to binary**

16 | 200
16 | 12 ↑ $(200)_{10} = (C8)_{16}$
0 $= (11001000)_2$

$0.625 \times 16 = 10 = A$

$\therefore (0.625)_{10} = (0.A)_{16} = (01010)_2$

$(200.625)_{10} = (11001000.1010)_2$

**Step 2 : Normalization**

$11001000101 \times 2^7$

**Step 3 : Calculate biased exponent**

For single precision

Biased exp = exp + 127
$= (134)_{10}$
$= (10000110)_2$

**For double precision**

Biased exp = exp + 1023 = $(1030)_{10}$
$= (10000000110)_2$

**Step 4 : Find representation**

Single price

| 31 | 30 ... 23 | 22 ... 0 |
|---|---|---|
| S | Biased exponent | mantissa |
| 0 | 10000110 | 10010001010 |



Double price

| 63 | 62 ... 52 | 51 ... 0 |
|---|---|---|
| S | Biased exponent | mantissa |
| 0 | 10000000110 | 100100010100 |

**Example (2) :** Represent $-(0.125)_{10}$ in both single precision as well as double precision

**Step 1 :**

$(0.125)_{10} = 2 = (0.0010)_2$

**Step 2 : Normalization**

$10 \times 2^{-3}$

**Step 3 : Single precision**

Biased exp = −3 + 127
$= (124)_{10}$
$= (0111100)_2$

Double precision
$= -3 + 1023$
$= 1020$
$= (01111111100)_2$

**Single precision**

| 31 | 30 ... 23 | 22 ... 0 |
|---|---|---|
| S | Biased exp | Mantissa |
| 1 | 01111100 | 00... |

**Double precision**

| 63 | 62 ... 52 | 51 ... 0 |
|---|---|---|
| S | Biased exp | Mantissa |
| 1 | 01111100 | 00... |

**Note :** The biased exponent can be in the range 1-254 for single precision (exponent range is − 126 to + 127) and 1 − 2046 for double precision (exp range − 1022 to + 1023). The biased exp values 0 and 255 for single precision && (0 and 2047 for double precision) are used to represent zero, ∞, denormalized form, NaN (Not a Number.)

**Ex. 1.8.2**

Represent $(178.1875)_{10}$ in single and double precision floating point format.

**Soln. :**

Convert given decimal number into its equivalent binary

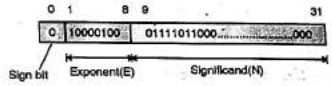
178 = 10111100

.1875 × 2 = 0.3750
.3750 × 2 = 0.7500
.7500 × 2 = 1.500
.500 × 2 = 1.000
$\therefore (178.1875)_{10} = (10111100.0011)_2$

**(a) Single precision format**

In IEEE format, the value of a number for given exponent (E) and significant (N) is given by (1.N) $2^{E-127}$ in order to represent $(10111100.0011)_2$, we must convert it into the form (1.N) $2^{E-127}$.

$101111.011 = 1.01111011 \times 2^5$
$5 = E - 127$
$\therefore E = 127 + 5 = 132$
$(132)_{10} = (10000100)_2$

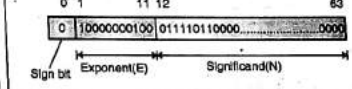



**(b) Double precision format**

In IEEE double precision format, the value of a number for given exponent (E) and significant (N) is given by (1.N) $2^{E-1023}$.

In order to represent $(10111100.0011)_2$, we must convert it into the form (1.N) $2^{E-1023}$.

$101111.011 = 1.01111011 \times 2^5$
$5 = E - 1023$
$\therefore E = 1023 + 5 = 1028$
$(1028)_{10} = (10000000100)_2$




**Ex. 1.8.3**

Represent $(309.1875)_{10}$ in single precision and double precision format.

**Soln. :** Convert given decimal number into its equivalent binary.

$(309)_{10} = (100110101)_2$
.1875 × 2 = 0.3750
.3750 × 2 = 0.7500

$.7500 \times 2 = 1.5000$

$.5000 \times 2 = 1.0000$

$\therefore 309.1875 = 100110101.0011$

**(a) Single precision format**

In IEEE single precision format, the value of a number for given exponent (E) and significant (N) is given by $(1.N) \times 2^{E-127}$.

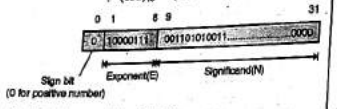
In order to represent 100110101.0011, we must convert it into the form $(1.N) \times 2^{E-127}$.

$$100110101.0011 = 1.00110101001 1 \times 2^{8}$$

$$8 = E - 127$$

$$\therefore E = 127 + 8 = 135$$

$$(135)_{10} = (10000111)_2$$




**(b) Double precision format**

In IEEE double precision format, the value of a number for given exponent (E) and significant (N) is given by,

$(1.N) \times 2^{E-1023}$

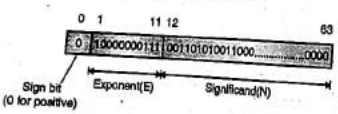
In order to represent 100110101.0011, we must convert it into the form $(1.N) \times 2^{E-1023}$.

$$100110101.0011 = 1.00110101001 1 \times 2^{8}$$

$$5 = E - 1023$$

$$\therefore E = 1023 + 5 = 1031$$

$$(1031)_{10} = (10000000111)_2$$




**Ex. 1.8.4** May 2016 5 Marks

Express $(35.25)_{10}$ in the IEEE single precision standard of floating point representation.

**Soln. :**

$(35.25)_{10}$

**Step 1 :** $(35)_{10} = (23)_{16} = (00100011)_2$

$(0.25)_{10} = (0.01)_2$

$\therefore (35.25)_{10} = (100011.01)_2$

**Step 2 :** $(35.25)_{10} = (1.0001101)_2 \times 2^5$

**Step 3 :**

Biased exponent = $127 + 5 = (132)_{10} = (10$

5 Biased exponent mantissa

| 0 | 10000100 | 0001101000..... |
|---|---|---|

31 30 23 22 0

**Ex. 1.8.5** Dec. 2016. 10 Marks

Convert $(127.125)_{10}$ in IEEE-754 single and dou floating point representation.

**Soln. :**

**Step 1 :**

| | | | |
|---|---|---|---|
| 16 | 127 | | |
| 16 | 7 | (15) F | $\therefore (127)_{10} = (7F)_{16} = (01$ |
| 0 | 7 | | |

$0.125 \times 16 = 2.000$

$\therefore (0.125)_{10} = (2)_{16} = (0.001)_2$

$\therefore (0.125)_{10} = (2)_{16} = (0.000)$

**Step 2 :** $(127 - 125)_{10} = (0111\ 1111.0010)_2$

$= (1.111111001)_2 \times 2^6$

**Step 3 : Biased exponent**

(i) For single Precision $\Rightarrow 127 + 6 = (133)_{10} = (12$

(ii) For double Precision $\Rightarrow 1023 + 6 - (1029)_{10}$

$= (10000000101)$

**(a) Single Precision**

31 30 23 22 0

| S | Biased Exponent | Mantissa |
|---|---|---|
| 0 | 10000101 | 11111100100..... |

**(b) Double Precision**

63 62 52 51 0

| S | Biased Exponent | Mantissa |
|---|---|---|
| 0 | 10000000101 | 11111100100..... |

**Syllabus Topic : Floating Point Arithmetic Addition, Subtraction, Multiplication, Divisi**

## 1.9 Floating Point Arithmetic : Addition, Subtraction, Multiplication, Division

Q. Explain with flowchart addition and sub floating point numbers.



In floating point arithmetic, addition and subtraction are more complex than multiplication and division. Addition and subtraction operations are carried out in four basic phases :

(1) Check for zeros.

(2) Align the significant

(3) Add or subtract the significant

(4) Normalize the result

First number $X = M \times B^x$

Second-number $Y = N \times B^y$

**Arithmetic operations :**

$X + Y = (M \times B^{x-y} + N) B^y$ } Exponent $x \le$ exponent $y$

$X - Y = (M \times B^{x-y} - N) B^y$

$X * Y = (M * N) B^{x+y}$

$X \div Y = (M \div N) \times B^{x-y}$

- In the next phase, exponents of the two numbers X and Y are made equal. Alignment is achieved by shifting either the smaller number to its right (increasing its exponent) or shifting the larger number to the left.
- Since either operation may result in loss of digits, it is the smaller number that is shifted. The alignment is achieved by repeatedly shifting the magnitude portion of the significant right 1 digit and incrementing the exponent until the two digits exponents are equal.
- Next, the two significant are added together, taking into account their signs. Since the sign may differ, the result may be 0. There is also the possibility of significant over flow.
- Next, result is normalized. Normalization consists of shifting significant digits left until the most significant digit is nonzero. Each shift causes a decrement of the exponent and thus could cause an exponent underflow.

**Addition and subtraction in float type data**

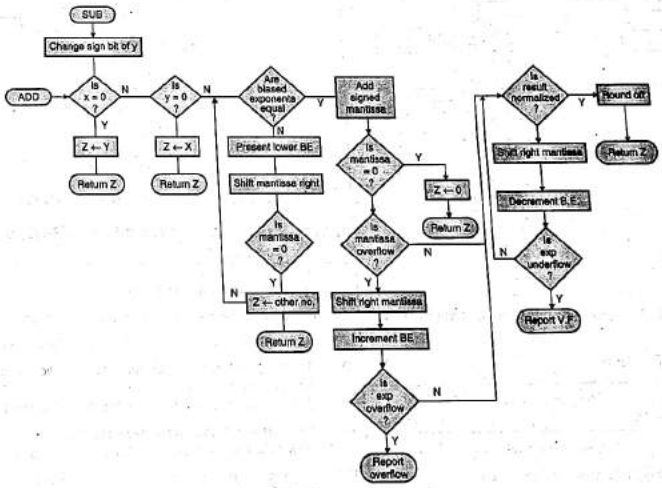



Fig 1.9.1

$.7500 \times 2 = 1.5000$

$.5000 \times 2 = 1.0000$

$\therefore 309.1875 = 100110101.0011$

**(a) Single precision format**

In IEEE single precision format, the value of a number for given exponent (E) and significant (N) is given by $(1.N) \times 2^{E-127}$.

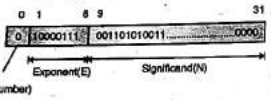
In order to represent 100110101.0011, we must convert it into the form $(1.N) \times 2^{E-127}$.

$$100110101.0011 = 1.001101010011 \times 2^{8}$$

$$8 = E - 127$$

$$\therefore E = 127 + 8 = 135$$

$$(135)_{10} = (10000111)_2$$




**(b) Double precision format**

In IEEE double precision format, the value of a number for given exponent (E) and significant (N) is given by,

$$(1.N) \times 2^{E-1023}$$

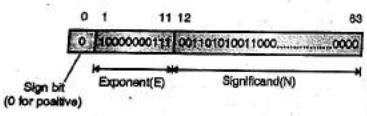
In order to represent 100110101.0011, we must convert it into the form $(1.N) \times 2^{E-1023}$.

$$100110101.0011 = 1.001101010011 \times 2^{8}$$

$$5 = E - 1023$$

$$\therefore E = 1023 + 5 = 1031$$

$$(1031)_{10} = (10000000111)_2$$




**Ex. 1.8.4** May 2016. 5 Marks

Express $(35.25)_{10}$ in the IEEE single precision standard of floating point representation.

**Soln. :**

$(35.25)_{10}$

**Step 1 :** $(35)_{10} = (23)_{16} = (00100011)_2$

$(0.25)_{10} = (0.01)_2$

$\therefore (35.25)_{10} = (100011.01)_2$

**Step 2 :** $(35.25)_{10} = (1.0001101)_2 \times 2^5$

**Step 3 :**

Biased exponent = $127 + 5 = (132)_{10} = (1000\ 0100)_2$

5 Biased exponent mantissa

| 0 | 10000100 | 0001101000..... |
|---|---|---|

3130 23 22 0

**Ex. 1.8.5** Dec. 2016. 10 Marks

Convert $(127.125)_{10}$ in IEEE-754 single and double precision floating point representation.

**Soln. :**

**Step 1 :**

| 16 | 127 | |
|---|---|---|
| 16 | 7 | (15) F |
| | 0 | 7 |

$\therefore (127)_{10} = (7F)_{16} = (01111111)_2$

$0.125 \times 16 = 2.000$

$\therefore (0.125)_{10} = (2)_{16} = (0.001)_2$

$\therefore (0.125)_{10} = (2)_{16} = (0.000)$

**Step 2 :** $(127 - 125)_{10} = (0111\ 1111.0010)_2$

$= (1.111111001)_2 \times 2^6$

**Step 3 : Biased exponent**

(i) For single Precision $\Rightarrow 127 + 6 = (133)_{10} = (1000010$

(ii) For double Precision $\Rightarrow 1023 + 6 - (1029)_{10}$

$= (10000000101)$

**(a) Single Precision**

31 30 23 20 0

| S | Biased Exponent | Mantissa |
|---|---|---|
| 0 | 10000101 | 11111100100..... |

**(b) Double Precision**

63 62 52 51 0

| S | Biased Exponent | Mantissa |
|---|---|---|
| 0 | 10000000101 | 11111100100..... |

**Syllabus Topic : Floating Point Arithmetic : Addition, Subtraction, Multiplication, Division**

## 1.9 Floating Point Arithmetic : Addition, Subtraction, Multiplication, Division

**Q.** Explain with flowchart addition and subtraction of floating point numbers. (10 Marks)



In floating point arithmetic, addition and subtraction are more complex than multiplication and division. Addition and subtraction operations are carried out in four basic phases :

(1) Check for zeros.

(2) Align the significant

(3) Add or subtract the significant

(4) Normalize the result

First number $X = M \times B^x$

Second number $Y = N \times B^y$

**Arithmetic operations :**

$X + Y = (M \times B^{x-y} + N) B^y$ } Exponent $x \leq$ exponent $y$

$X - Y = (M \times B^{x-y} - N) B^y$ }

$X * Y = (M * N) B^{x+y}$

$X \div Y = (M \div N) \times B^{x-y}$

- In the next phase, exponents of the two numbers X and Y are made equal. Alignment is achieved by shifting either the smaller number to its right (increasing its exponent) or shifting the larger number to the left.
- Since either operation may result in loss of digits, it is the smaller number that is shifted. The alignment is achieved by repeatedly shifting the magnitude portion of the significant right 1 digit and incrementing the exponent until the two digits exponents are equal.
- Next, the two significant are added together, taking into account their signs. Since the sign may differ, the result may be 0. There is also the possibility of significant over flow.
- Next, result is normalized. Normalization consists of shifting significant digits left until the most significant digit is nonzero. Each shift causes a decrement of the exponent and thus could cause an exponent underflow.

**Addition and subtraction in float type data**

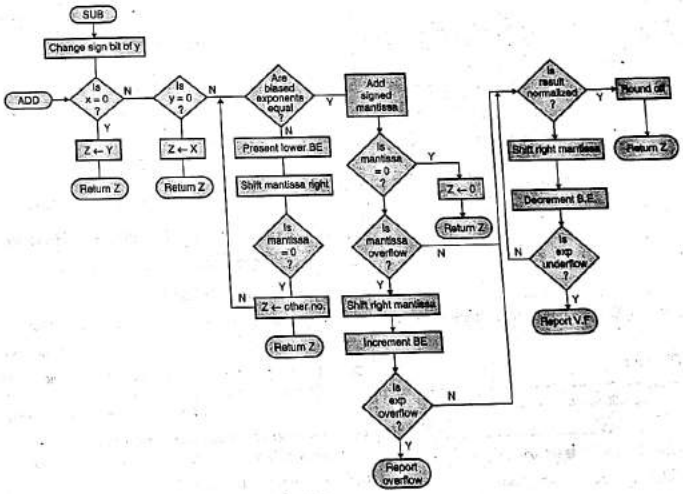
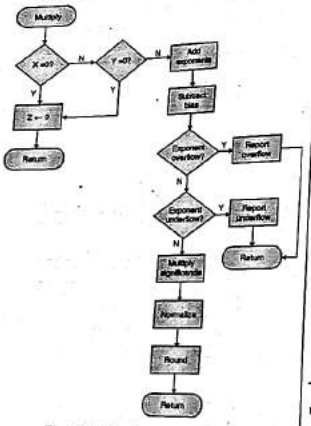



Fig 1.9.1

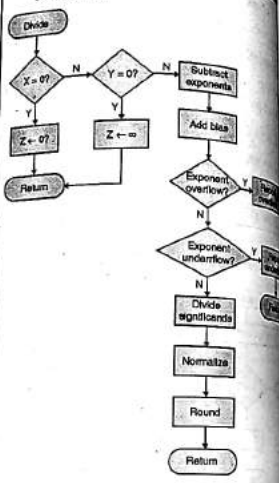
### 1.9.1 Multiplication

**Q. Explain with flowchart multiplication of floating point numbers. (10 Marks)**

- If either of the operand is 0, 0 is reported as the result.
- Next, exponents are added together. If the exponents are stored in biased form, the exponent sum would have doubled the bias. Thus, the bias must be subtracted from the sum. The result could cause either an exponent overflow or underflow.
- Next, if the exponent of the product is within the proper range, significant are multiplied together.
- After the product is calculated, the result is then normalized. Normalization may result in exponent underflow.
- Next, the divisor exponent is subtracted from the dividend exponent. This removes the bias, which must be added back.
- Exponent is checked for underflow or overflow.
- Next, significant are divided.
- Normalization and rounding are performed subsequently.




(∞ 2.60)**Fig. 1.9.2 : Floating point multiplication (Z = X * Y)**




(∞ 2.65)**Fig. 1.9.3 : Floating point division (Z = X**

### 1.9.2 Division

**Q. Explain with flowchart division of floating point numbers. (10 Marks)**

- If the divisor is 0, result is set to infinity.
- If the dividend is 0, result is set to zero.

## 1.10 Exam Pack (University and Re Questions)

☞ **Syllabus Topic : Introduction**

**Q.** Compare Computer Architecture and organiza
(Ans. : Refer section 1.1) (5

**Q.** Differentiate between Computer Organiza Computer Architecture.
*(Ans. : Refer section 1.1)* (Dec. 2015, 5

**Q.** List different memory organization characteris
*(Ans. : Refer section 1.1)* (Dec. 2015, 5

**Q.** Differentiate between Computer Architectu Computer organization.
*(Ans. : Refer section 1.1)* (May 2016, 5



**Q.** Differentiate between Computer Organization and Architecture. *(Ans. : Refer section 1.1)* **(Dec. 2016, 5 Marks)**

☞ **Syllabus Topic : Basic Organization of Computer and Block Level Description of Functional Units**

**Q.** Explain the structural overview of a computer. *(Ans. : Refer section 1.2.1)* **(5 Marks)**

☞ **Syllabus Topic : Integer Data Computation : Addition, Subtraction**

**Q.** Explain various signed and unsigned number representations for integers. *(Ans. : Refer section 1.5)* **(10 Marks)**

☞ **Syllabus Topic : Multiplication : Unsigned Multiplication**

**Q.** Explain shift and add method of multiplication with hardware and flowchart. (Ans. : Refer section 1.5.2) **(10 Marks)**

☞ **Syllabus Topic : Booth's Algorithm**

**Q.** Explain booth's principle. *(Ans. : Refer section 1.5.3)* **(5 Marks)**

**Q.** Explain the Booth's method of multiplying signed numbers with hardware and flowchart. *(Ans. : Refer section 1.5.3)* **(10 Marks)**

**Q.** Draw the flow chart for Booth's Algorithm for Two's Complement Multiplication. *(Ans. : Refer section 1.5.3)* **(May 2014, 5 Marks)**

**Q.** Using booth's algorithm show the multiplication of $7 \times 5$. *(Ans. : Refer Ex. 1.5.16)* **(May 2014, 7 Marks)**

**Q.** Draw flow chart of Booth's algorithm. *(Ans. : Refer section 1.5.3)* **(Dec. 2014, 5 Marks)**

**Q.** Multiply $(-2)_{10}$ and $(-5)_{10}$ using Booth's Algorithm. *(Ans. : Refer Ex. 1.5.17)* **(Dec. 2014, 10 Marks)**

**Q.** Draw flowchart for Booth's Algorithm for Two's Complement Multiplication. *(Ans. : Refer section 1.5.3)* **(May 2015, 3 Marks)**

**Q.** Draw the flow chart for Booth's Algorithm for two's complement multiplication. *(Ans. : Refer section 1.5.3)* **(Dec. 2015, 4 Marks)**

**Q.** Using Booth's algorithm show the multiplication of $-3$ and $-7$. *(Ans. : Refer Ex. 1.5.18)* **(Dec. 2015, 6 Marks)**

**Q.** Multiply (–10) and (– 4) using Booth's algorithm. *(Ans. : Refer Ex. 1.5.19)* **(May 2016, 10 Marks)**

**Q.** Multiply (– 7) with (4) by using Booth's algorithm of Multiplication. *(Ans. : Refer Ex. 1.5.20)* **(Dec. 2016, 10 Marks)**

**Q.** Multiply (– 5) and (2) using Booth's algorithm. *(Ans. : Similar to Ex.1.5.17)* **(May 2017, 10 Marks)**

**Q.** Explain with hardware requirement how to add and subtract integer numbers. *(Ans. : Refer section 1.5.5)* **(5 Marks)**

☞ **Syllabus Topic: Division of Integers: Restoring**

**Q.** Using unsigned Binary Division method, divide 7 by 3. *(Ans. : Refer Ex. 1.6.9)* **(May 2015, 6 Marks)**

**Q.** Draw the flow chart for restore division algorithm. *(Ans. : Refer section 1.6.1)* **(May 2017, 4 Marks)**

**Q.** Divide using restore division method 7/3. *(Ans. : Refer section 1.6.1)* **(May 2017, 6 Marks)**

**Q.** Explain restoring method of division with flowchart. *(Ans. : Refer section 1.6.1)* **(5 Marks)**

☞ **Syllabus Topic : Division of Integers : Non-Restoring Methods**

**Q.** Explain non-restoring method of division with flowchart. *(Ans. : Refer section 1.7)* **(5 Marks)**

☞ **Syllabus Topic : Floating-Point Representation : IEEE 754 Floating Point Number Representation**

**Q.** Explain the IEEE 754 standard for representing floating point numbers. *(Ans. : Refer section 1.8.1)* **(May 2015, 6 Marks)**

**Q.** Show IEEE 754 standards for binary floating-point representation for 32 bit single format and 64 bit double format. *(Ans. : Refer section 1.8)* **(May 2014, 3 Marks)**

**Q.** Explain IEEE 754 standards for Floating Point number representation. *(Ans. : Refer section 1.8)* **(May 2015, 6 Marks)**

**Q.** Show IEEE 754 standards for Binary Floating Point Representation for 32 bit single format and 64 bit double format. *(Ans. : Refer section 1.8)* **(Dec. 2015, 5 Marks)**

**Q.** Express $(35.25)_{10}$ in the IEEE single precision standard of floating point representation. *(Ans. : Refer Ex. 1.8.4)* **(May 2016, 5 Marks)**

**Q.** Convert $(127.125)_{10}$ in IEEE-754 single and double precision floating point representation. *(Ans. : Refer Ex. 1.8.5)* **(Dec. 2016, 10 Marks)**

**Q.** Show IEEE 754 standards for binary floating point representation for 32 bit single format and 64 bit double format. *(Ans. : Refer section 1.8)* **(May 2017, 10 Marks)**

**Syllabus Topic : Floating Point Arithmetic : Addition, Subtraction, Multiplication, Division**

Q. Explain with flowchart addition and subtraction of floating point numbers. **(10 Marks)**
*(Ans. : Refer section 1.9)*

Q. Explain with flowchart multiplication of numbers. *(Ans. : Refer section 1.9.1)*

Q. Explain with flowchart division of numbers. *(Ans. : Refer section 1.9.2)*

CHAPTER 2

# Processor Organization and Architecture

Module II

**Syllabus**

Von Neumann model, Harvard Architecture, Register Organization, Instruction formats, addressing modes, instruction cycle. Instruction interpretation and sequencing. ALU and Shifters, Basic pipelined datapath and control, Data dependences, data hazards, Branch hazards, delayed branches, branch prediction, Performance measures – CPI, speedup, efficiency, throughput and Amdahl's law.

**Syllabus Topic : Von Neumann Model**

## 2.1 Von Neumann and Harvard Architecture

→ **(MU - May 2014, Dec. 2014, May 2015, Dec. 2015, Dec. 2016)**

- What is stored program concept ? **May 14, 3 Marks**
- Define stored program concept and draw Von Neumann's Architecture. **Dec. 14, 5 Marks**
- What is stored program concept in digital computer? **May 15, 3 Marks**
- Explain role of different registers like IR, PC, SP, AC, MAR and MDR used in Von Neumann model. **Dec. 15, 5 Marks**
- Explain Von Neumann architecture in detail. **Dec. 16, 5 Marks**

There are two ways of memory interfacing architectures for a processor depending on the processor design. The first one is called Von Neumann architecture and later Harvard architecture.

### 2.1.1 Von Neumann Architecture

- Explain von Neumann's system **(5 Marks)**

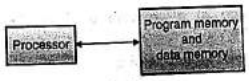



Fig. 2.1.1

- Fig. 2.1.1 shows the connection for Von Neumann architecture of computer.
- The name is derived from the mathematician and early computer scientist John Von Neumann.

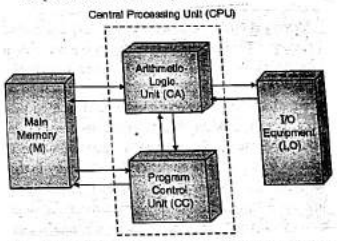



**Fig. 2.1.2 : Von Neumann Architecture of a computer**

- The computer has a common memory for data as well as code to be executed.
- The processor needs two clock cycles to complete an instruction, first to get an instruction and second to get the data.
- This system has three units CPU, Memory and I/O devices. The CPU has two units Arithmetic Unit and Control unit. Let us discuss these units in detail.

→ **1. Input unit**

A computer accepts inputs from the user through these devices i.e. input devices. The commonly used input devices are keyboard and mouse. Besides that, there are

**Syllabus Topic : Floating Point Arithmetic : Addition, Subtraction, Multiplication, Division**

Q. Explain with flowchart addition and subtraction of floating point numbers. **(10 Marks)**
*(Ans. : Refer section 1.9)*

Q. ... numbers. *(Ans. : Refer section 1.9.1)*

Q. Explain with flowchart division of ... numbers. *(Ans. : Refer section 1.9.2)*

CHAPTER 2

# Processor Organization and Architecture

Module II

**Syllabus**

Von Neumann model, Harvard Architecture, Register Organization, Instruction formats, addressing modes, instruction cycle. Instruction interpretation and sequencing. ALU and Shifters, Basic pipelined datapath and control, Data dependences, data hazards, Branch hazards, delayed branches, branch prediction, Performance measures – CPI, speedup, efficiency, throughput and Amdahl's law.

**Syllabus Topic : Von Neumann Model**

## 2.1 Von Neumann and Harvard Architecture

→ **(MU - May 2014, Dec. 2014, May 2015, Dec. 2015, Dec. 2016)**

Q. What is stored program concept ? **May 14, 3 Marks**
Q. Define stored program concept and draw Von Neumann's Architecture. **Dec. 14, 5 Marks**
Q. What is stored program concept in digital computer? **May 15, 3 Marks**
Q. Explain role of different registers like IR, PC, SP, AC, MAR and MDR used in Von Neumann model. **Dec. 15, 5 Marks**
Q. Explain Von Neumann architecture in detail. **Dec. 16, 5 Marks**

There are two ways of memory interfacing architectures for a processor depending on the processor design. The first one is called Von Neumann architecture and later Harvard architecture.

### 2.1.1 Von Neumann Architecture

Q. Explain von Neumann's system. **(5 Marks)**

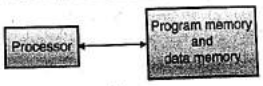



Fig. 2.1.1

- Fig. 2.1.1 shows the connection for Von Neumann architecture of computer.
- The name is derived from the mathematician and early computer scientist John Von Neumann.

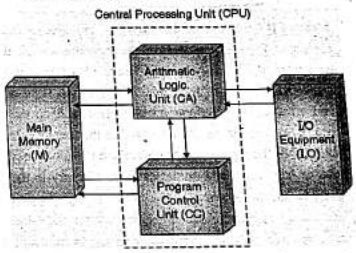



**Fig. 2.1.2 : Von Neumann Architecture of a computer**

- The computer has a common memory for data as well as code to be executed.
- The processor needs two clock cycles to complete an instruction, first to get an instruction and second to get the data.
- This system has three units CPU, Memory and I/O devices. The CPU has two units Arithmetic Unit and Control unit. Let us discuss these units in detail.

→ **1. Input unit**

A computer accepts inputs from the user through thes devices i.e. input devices. The commonly used inp devices are keyboard and mouse. Besides that, there a

devices like joystick, camera, scanner etc. which are also input devices. The devices input the data accepted from the user in a proper coded form understood by the computer.

➜ **2. Output unit**

The result is given back by the computer to the user through an output device. Input devices and output devices are also called as human interface devices, because they are used to interface the human to the computer. The mainly used output devices are monitor and printer. But there are many other output devices like plotter, speaker etc.

➜ **3. Arithmetic and Logic Unit (ALU)**

Arithmetic or logic operations like multiplication, addition, division, AND, OR, EXOR etc. are performed by ALU. Operands are brought into the ALU, where the necessary operation is performed.

➜ **4. Control unit**

The control unit as we know is the main unit that controls all the operations of the system, inside and outside the processor. The memory or I/O devices have to be controlled by the computer to perform the operation according to the instruction given to it.

➜ **5. Memory unit**

Memory is used to store the programs and data for the computer. The instructions from the programs are taken by the processor, decoded and executed accordingly. The data is also stored in the memory. The data is taken from memory and the operation is performed on that data, as well as the results are stored in the memory. In some cases the input to an operation and the result may also be from input and output devices. Memory in the Von Neumann system has a special organization wherein the data and instructions are stored in the same memory. We will see about this in the subsequent section.

### ☞ Key features of a Von Neumann machine

- The Von Neumann machine uses stored program concept. The program and data are stored in the same memory unit.
- Each location of the memory has a unique address i.e. no two locations have the same memory address.
- Execution of instruction in Von Neumann machine is carried out in a sequential manner (unless explicitly altered by the program itself) from one instruction to the next.

### ☞ Detailed structure of the CPU

The block diagram of the computer pro... Neumann have a minimal number of registers ... above blocks. This computer has a small set ... and an instruction was allowed to contain only ... address. Fig. 2.1.3 gives the detailed structure ... The structure shown in Fig. 2.1.3 consists of ... registers.

➜ **Accumulator (AC)**

It normally provides one of the operand to ... stores the result.

➜ **Data Register (DR)**

It acts as buffer storage between the CPU, ... memory or I/O devices.

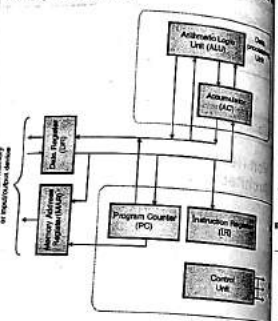



**Fig. 2.1.3 : Structure of the CPU**

➜ **Program Counter (PC)**

It always contains the address of the next instruc... be executed.

➜ **Instruction Register (IR)**

It holds the current instruction i.e. the opco... operand of the instruction to be executed.

➜ **Memory Address Register (MAR)**

The address from which the data or instruction ... fetched is provided by the processor through MAR. I... used to forward the address of memory location wh... is to be stored.

### ☞ Execution of a program by Von Neu... machine

- The program to be executed is stored in memory ...



A register, PC (Program counter) always points to the first instruction of the program when the computer starts. CPU fetches the instruction pointed by PC. PC contents are automatically incremented to point to the next instruction.

If the initial value of PC = 0000 (in binary), first instruction be fetched for execution. After fetching "Instruction number 1", PC will be incremented by one. It is assumed that the size of each instruction is 1 byte.

PC = PC + 1 = 0 + 1 = 1

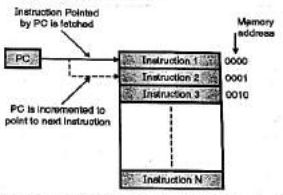



**Fig. 2.1.4 : Instruction pointed by PC is fetched by the CPU for execution. Subsequently PC is made to point to the next instruction**

### ☞ Fetching an instruction

- CPU interacts with memory through two special registers :
  - (1) **MAR (Memory Address Register) :** It provides address of memory location from where data or instruction is to be retrieved or to which data is to be stored.
  - (2) **DR (Data Register) :** It acts as buffer storage between the main memory and the CPU.
- The function and operation of these registers will be understood by the example below.
- The instruction to be executed is brought from the memory to the CPU, through the following steps :
  - (1) The address of the instruction is transferred from PC to MAR.

    MAR ← PC
  - (2) MAR puts this address on the address bus for selection of the required location of the memory.
  - (3) Control Unit generates the $\overline{RD}$ (read control signal) signal to perform read operation on memory. Required instruction is given on data bus by the memory. Instruction on data bus is accepted in DR (Data Register).

### ☞ Execution of instruction

- The fetched instruction is in the form of binary code and is loaded into instruction register (IR) from DR (Data Register). The instruction specifies what action the CPU has to take.
- The CPU interprets the instruction and performs the required action. The action could be :
  - (1) Data transfer between CPU and memory.
  - (2) Data transfer between CPU and I/O.
  - (3) The CPU may perform an arithmetic or logic operation on data.

**Syllabus Topic : Harvard Architecture**

## 2.1.2 Harvard Architecture


**Fig. 2.1.5**

- Fig. 2.1.5 shows the connection for Harvard computer architecture.
- The name is originated from Harvard's, "Harvard Mark I" a relay based old computer.
- In this case there are two separate memories for storing data and program.
- In this case the processor can simultaneously access instruction as well as the data and hence can complete an instruction execution in one cycle.

**Syllabus Topic : Register Organization**

## 2.2 CPU Architecture and Register Organization

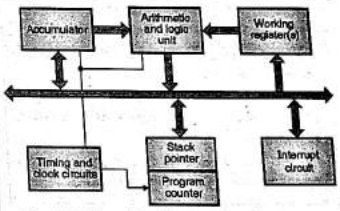



**Fig. 2.2.1 : General architecture of a microprocessor**

Fig 2.2.1 shows the architecture of microprocessor. This architecture is divided in different groups as follows :

(i) Registers

(ii) Arithmetic and logic unit
(iii) Interrupt control
(iv) Timing and control circuitry.

- It consists of PIPO (Parallel in parallel out) register as shown in Fig. 2.2.2.
- This section is also called as scratch pad memory. It stores data and address of memory.
- The register organization affects the length of program, the execution time of program and simplification of the program. To achieve better performance, the number of registers should be large.
- The architecture of microcomputer depends upon the number and type of the registers used in microprocessor. It consists 8-bit registers or 16 bit registers.
- The register section varies from microprocessor to microprocessor.
- The registers are used to store the data and address.
- These registers are classified as :
  - Temporary registers
  - General purpose registers
  - Special purpose registers.

➜ **(i) Register section**

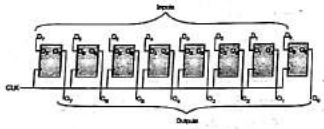



**Fig. 2.2.2 : 8 bit register**

➜ **(ii) Arithmetic and logical unit**

- This section processes data i.e. it performs arithmetic and logical operations.
- It performs arithmetic operations like addition, subtraction and logical operations like ANDing, ORing, EX-ORing, etc.
- The ALU is not available to the user. Its word length depends upon the width of an internal data bus.
- The ALU is controlled by timing and control circuits.
- It accepts operands from memory or register. It stores result of arithmetic and logic operations in register or memory.
- It provides status of result to the flag register. Flag register shows status of result.
- ALU looks after the branching decisions.

➜ **(iii) Interrupt control**

This block accepts different interrupt request inputs. When a valid interrupt request is present it informs control logic to take action in response to each signal.

➜ **(iv) Timing and control unit**

- This is a control section of microprocessor made up of synchronous sequential logic circuit.
- It controls all internal and external circuits.
- It operates with reference to clock signal.
- This accepts information from instruction decoder and generates micro steps to perform it. In addition to this the block accepts clock inputs, performs sequencing and synchronising operations. The synchronization is required for communication between microprocessor and peripheral devices. To implement this it uses different status and control signals.
- The basic operation of a microprocessor is regulated by this unit.
- It synchronizes all the data transfers.
- This unit takes appropriate actions in response to external control signals.

## Syllabus Topic : Instruction formats

### 2.2.1 Instruction Formats

**Q. Write a short note on instruction formats. (5 Marks)**

- The Control Unit and the ALU (Arithmetic and Logic Unit) along with some registers constitute the Central Processing Unit.
- Fig. 2.2.3 shows the basic components of the computer and their interconnection. Also the internal components of the CPU are shown in the Fig. 2.2.3. The computer consists of three basic components namely the CPU, memory and I/O devices connected with each other via the buses.
- Input devices are required to give the instructions and data to the system. The output devices are used to give the output devices.
- The instructions and the data given by the input device are to be stored, and for storage we require memory.



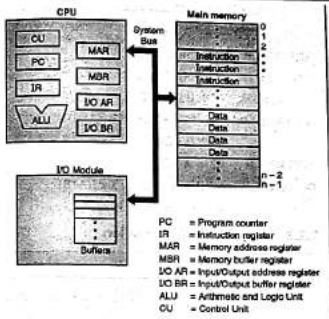



**Fig. 2.2.3 : Basic Components of the computer and the CPU**

- The memory module receives address and control signals (like read, write, timing etc) and also receives and sends data.
- The memory is assumed to be of 'n' locations with the addresses from 0 to n – 1.
- CPU module reads instruction and data, writes the data and also sends the control signals. It also receives interrupts from the I/O devices and acts accordingly.
- The CPU has the control unit that provides control signals to all the resources inside and outside the CPU. The CPU also has the ALU that performs the arithmetic and logical operations.
- The CPU also has some registers as seen in the Fig. 2.2.3. The register PC (Program Counter) is a register that always points to the next instruction to be executed. Hence the word "Program" in the name. It has to increment always to point to the next instruction, which is automatically done. Hence the word "Counter" in the name.
- The MAR (Memory Address Register) is used to store the address to be provided to the memory. And MBR (Memory Buffer Register) is used to store the data to be given or taken from the memory. Similarly for I/O devices we have IOAR and IOBR.
- Another register named as IR (Instruction Register) is used to store the instruction to be executed by the CPU.
- The use of these registers will be further seen in the next section named as Instruction Cycle.

### 2.2.2 Instruction Word Format - Number of Addresses

**Q. Explain single address, two address and three address instructions. (5 Marks)**

☞ **Elements of an Instruction**

1. Operation code (Opcode) is that part of the instruction which gives the code for the operation to be performed.
2. Source Operand reference or address 1, gives the reference of the data on which the operation is to be performed. This address could be a register, memory or an input device.
3. Source Operand reference or address 2, gives the reference of the second data on which the operation is to be performed. This address could again be a register, memory or an input device.
4. Result Operand reference gives the reference where the result after performing operation is to be stored. The result could be stored in the register, memory or given to an output device.
5. An instruction may have only one address with the other two fixed, or may have two addresses with one of the source operand address as the result operand address. Hence the instruction can have one, two or three addresses.

The Fig. 2.2.4 shows an example of a simple instruction format with one and two addresses.

| Opcode | Operand Address 1 |
|---|---|

**(a) Single address instruction format**

| Opcode | Operand Address 1 | Operand Address 2 |
|---|---|---|

**(b) Two address instruction format**

**Fig. 2.2.4 : Instruction Word Formats**

- Three address, One address and Zero address instructions.

☞ **Zero address instructions**

- PUSH A ; ToS <- A
- PUSH B ; ToS <- B
- ADD ; ToS <- A + B
- PUSH C ; ToS <- C
- PUSH D ; ToS <- D

- ADD ; ToS <- C + D
- MUL ; ToS <- (A + B)*(C + D)
- PUSH E ; ToS <- E
- PUSH F ; ToS <- F
- SUB ; ToS <- (E – F)
- DIV ; ToS <- (A + B)*(C + D)/(E – F)
- POP X ; M[X] <- ToS

**One address instructions**

- LOAD A ; AC <- M[A]
- ADD B ; AC <- AC + M[B]
- STORE P ; M[P]<- AC
- LOAD C ; AC<- M[C]
- ADD D ; AC <- AC + M[D]
- MUL P ; AC <- AC * M[P]
  i.e. AC <- (A + B) * (B + C)
- STORE P ; M[P] <- AC
- LOAD E ; AC <- M[E]
- SUB F ; AC<- AC – M[F]
- STORE Q ; M[Q] <- AC
- LOAD P ; AC <- M[P]
- DIV Q ; AC <- AC / M[Q]
  i.e. AC <- (A + B)*(C+D)/(E – F)
- STORE X ; M[X] <- X
  i.e. X<- (A + B)*(C + D)/(E – F)

**Accumulator type one address format**

LOAD A ; AC <- M[A]
MUL B ; AC <- AC * M[B]
STORE P ; M[P]<- AC
LOAD C ; AC<- M[C]
MUL D ; AC <- AC * M[D]
SUB E ; AC<- AC – M[E]
ADD P ; AC <- AC + M[P]
i.e. AC <- (A*B) + (C * D – E)
STORE P ; M[P] <- AC
LOAD A ; AC <- M[A]
ADD B ; AC<- AC + M[B]
STORE Q; M[Q] <- AC
LOAD P ; AC <- M[P]
DIV Q ; AC <- AC / M[Q]
i.e. AC <- ((A*B) + (C*D – E)) / (A + B)
STORE X; M[X] <- X I.E. X<- ((A*B) + (C*D – E)) / (A + B)

**Three address instructions**

((A*B) + (C*D – E)) / (A + B)
ADD R1,A,B ; R1<- M[A] + M[B]
MUL R2,C,D ; R2<- M[C] * M[D]
SUB R2,R2,E ; R2<- R2 – M[E]
ADD R1,R1,R2 ; R1< R1 + R2 i.e. R1< ((A*B) + (C*D – E))
ADD R2,A,B ; R2 <- A + B
DIV X,R1,R2 ; M[X] <- R1/ R2 i.e. X < ((A*B) + (C*D – E))/ (A + B)

### 2.2.3 Reverse Polish Notation

**Q. Explain polish notation.**

- Reverse Polish Notation (RPN) is a mathe... notation wherein the operator follows all ... operands. The RPN is also known as Postfix ... and is parenthesis-free.
- The polish notation use to follow the sequence ... operator followed by its operands, and hence ... reverse polish notation gets its name. The name ... notation is derived from its origin country.
- The Reverse Polish notation was proposed in ... Burks, Warren, and Wright and was independ... reinvented by F. L. Bauer and E. W. Dijkstra ... early 1960s. This notation helps to reduce co... memory access and to utilize the stack to ev... expressions.
- During the 1970s and 1980s, RPN was even know... the general public, as it was widely used in calcula...
- In Reverse Polish notation the operators follow ... operands. This means that the operands are mentio... first and then their operator, for example, to add ... 4, one would write "3 4 +" rather than "3 + 4".
- If there are multiple operations, then the operat... given immediately after its second operand. ... example the expression written "3 – 4 + ... conventional infix notation would be wr... "3 4 – 5 +" in RPN.
- An advantage of RPN is that it removes the need ... parentheses.



**Syllabus Topic : Instruction Cycle**

### 2.2.4 Basic Instruction Cycle

**Q. Explain basic instruction cycle. (5 Marks)**

- The instruction cycle is a representation of the states that the computer or the microprocessor performs when executing an instruction.
- The instruction cycle comprises of two main steps to be followed to execute the instruction, namely the fetch operation in the fetch cycle and the execution operation during the execute cycle.

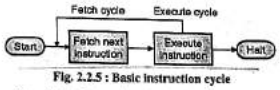



**Fig. 2.2.5 : Basic instruction cycle**

- Fig. 2.2.5 shows the basic instruction cycle. It comprises of the fetch and executes cycle in a loop to execute huge number of instructions, until it reaches the halt instruction.
- The fetch cycle comprises of the following operations :
  1. Program Counter (PC) holds address of next instruction to fetch; hence the CPU (Processor) fetches instruction from memory location pointed to by PC. This is done by providing the value of the PC to the MAR and giving the Read control signal to the memory. On this the memory provides the value in the given address (which is the instruction) to MBR.
  2. The PC value has to be incremented to point to the next instruction (Sometimes the value of PC may have be completely changed in case of some special instructions called as branching instructions).
  3. The instruction is loaded into Instruction Register (IR) from the MBR.
  4. Finally the processor interprets or decodes the instruction. The processor performs required operations in the execute cycle.
- In the execute cycle the operation asked to be performed by the instruction is done. It may comprise of one or more of the following operations :
  1. Transfer of data between processor and memory or between processor and IO module.
  2. Processing of data like some arithmetic or logical operations on data.
  3. Change of the sequence of operation i.e. branching instructions.

### 2.2.5 Interrupt Cycle

**Q. Explain the instruction cycle with interrupts. (5 Marks)**

- Fetch and execute are not the only two states in the instruction cycle. There is one more state i.e. Interrupt cycle.
- In this subsection we will see the concept of interrupt in short and the interrupt cycle.
- Interrupt is a mechanism by which I/O modules can interrupt normal sequence of processing. Interrupt can be because of some request from an I/O device to service that particular device. This service may be take or give data or some control operation. It may also because of some unexpected operation in the program execution by the CPU itself.
- Interrupt cycle as discussed earlier is added to instruction cycle. During this cycle the processor checks for interrupt, and if present and enabled services the same.
- If no interrupt is present then it fetches the next instruction else if interrupt pending then it performs the following operations :
  1. Suspend the execution of current program.
  2. Save the context of the current program under execution.
  3. Set the PC value to start address of interrupt handler routine also called as interrupt service routine. Interrupt service routine is a small program which when executed, services the interrupting source.
  4. Process the interrupt service routine (ISR) and then.
  5. Restore the context and continue execution of the interrupted program.
- Thus the complete basic instruction cycle with Interrupts can be as shown in the Fig. 2.2.6.

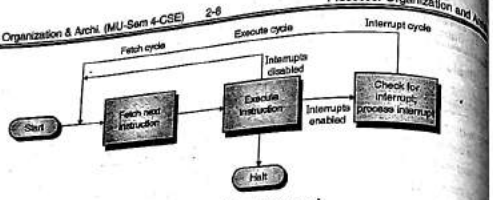



**Fig. 2.2.6 : Complete basic instruction cycle**

- You will notice in Fig. 2.2.6, the interrupts are checked for, after the execute cycle and processed if enabled and else, it fetches the next instruction.
- The detailed instruction cycle is shown in Fig. 2.2.7.

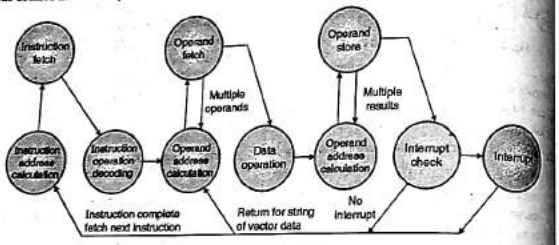



**Fig. 2.2.7 : Detailed instruction cycle**

- In Fig. 2.2.7, there are some states drawn on the upper side, while some on the lower side. The ones on the upper side are the operations carried out on the buses or are external operations, while the ones at the lower level are the operations carried out inside the CPU or are internal operations.
- The instruction cycle begins from the "Instruction address calculation" state, wherein the address of the next instruction is calculated or the value of the PC is updated. Then the instruction is fetched, which requires the operation on the buses.
- The instruction fetched is then decoded. Until this state, it is the fetch cycle.
- In the execute cycle, the operand address is calculated and the operands are fetched from the calculated address. Again to fetch the operands, we require the buses. After fetching the operand, if more operands are required for multiple operand instructions, then the next state is again calculate the operand address i.e. the address of the next operand. Once all the operands are fetched, the data operation is carried out as per operation indicated in the instruction.
- Now for the result storage again the address of operand is calculated and the result is stored in the specified location of the memory. In case of multiple operands again the calculation and storage process for operand continues until all the operands are stored.
- Now begins the interrupt cycle, wherein the first state is to check the presence of an enabled interrupt. If there is none, then the next state as seen in the Fig. 2.2.7, is calculation of next instruction address i.e. execute next sequential instruction.
- But in case the interrupt is present and enabled then servicing of the same is done as discussed earlier in section.
- In the Fig. 2.2.7, you will also notice that there are paths from the end of the previous instruction. The that goes to the state "Instruction address calcu



for the next instruction; and the one that goes to the "Operand address calculation" for vector instructions.

- Vector instructions are those instructions wherein the operation is same but the data on which the operation is to be performed in a huge block of data or an array of data. Hence in the second case, the instruction is already fetched and decoded i.e. the operation is already know, and the operation is to be performed on a block of data.
- After completing the operation on one set of operands, the CPU returns to the next operand address calculation state, wherein it calculates and fetches the next operand. Then it performs the operation, stores the result and again for the next set of operand, until all the operands in the array are completed.

**Syllabus Topic : Addressing Modes**

## 2.3 Addressing Modes

→ **(MU - Dec. 2014)**

**Q. Explain addressing modes of a processor. (10 Marks)**
**Q. Explain in detail different types of addressing modes. Dec. 14, 10 Marks**

- We have seen in the previous section, that an instruction has various components among the source operand reference. For the source operand reference we have various methods to give the same. Based on these different methods we have different addressing modes. In other words, addressing modes are the different methods (modes) to select (address) the operand for an instruction.
- The different addressing modes are :

| | | | |
|---|---|---|---|
| 1. | Immediate | 2. | Direct |
| 3. | Indirect | 4. | Register |
| 5. | Register Indirect | 6. | Displacement (Indexed) |
| 7. | Relative | 8. | Stack |

### → 1. Immediate addressing mode

- In this case the operand is part of instruction. The operand is in the immediate next location of the opcode, hence the name.
- For example ADD AX, 0005H. This instruction adds the contents of register AX with the value 5. In this case 5 is an operand and is given in the instruction itself. AX register is also called as the accumulator register.
- The advantage of this addressing mode is that it is fast.
- The disadvantage is that it has a limited range.
- Fig. 2.3.1 shows the structure of an instruction and operand access technique for the immediate addressing mode.

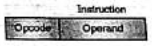



**Fig. 2.3.1 : Immediate addressing mode**

### → 2. Direct addressing mode

- In this case the address field contains memory address of the operand.
- For example ADD AX,[0005H]. This instruction adds the contents of memory location 0005H to accumulator. The operand is taken from the memory location specified in the instruction.
- In this case there is only a single memory reference to access data.
- The advantage is that there are no calculations to work out effective address.
- The disadvantage is that this addressing mode can be used for a limited address space.

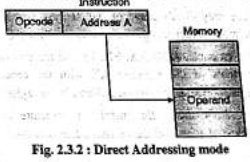



**Fig. 2.3.2 : Direct Addressing mode**

- Fig 2.3.2 shows the structure of the instruction and operand access technique for the direct addressing mode.

### → 3. Indirect addressing mode

- In this case a memory location has the address of the operand in another memory location i.e. a memory operand is pointed to by address field contains the address of (pointer to) the operand.
- For example ADD AX, [[1000]]. This instruction adds the contents of memory location pointed to by contents of memory location 1000, with the contents of accumulator and stores the result in accumulator.
- The disadvantage is that this method is slower as multiple memory locations are to be accessed to get the operand.

- Fig. 2.3.3 shows the structure of an instruction and operand access technique for the indirect addressing mode.

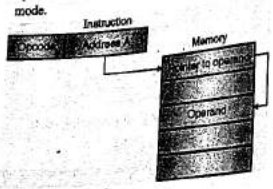



Fig. 2.3.3

### ➜ 4. Register addressing mode

- In this case the operand is held in the register named in operand address field.
- There are limited numbers of registers, hence very small address field is required. Hence shorter instructions and faster instruction fetch.
- Also another advantage is that there is no memory access. We can say that this is the best addressing mode in terms of time required to access the operand.
- The only disadvantage of this method is the limited number of registers available in most of the processors.
- For example, ADD AX, BX. This instruction adds the contents of the registers AX with the contents of registers BX and the result is stored in the register AX.
- Fig. 2.3.4 shows the instruction structure and the method of access for the register addressing mode.
- As already discussed the major advantage of this addressing mode is that it has very fast execution but limited address space.
- Thus the processors that have multiple registers helps in improving the performance of the processor.

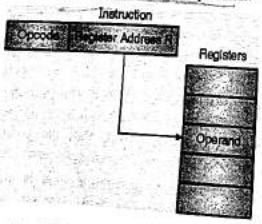



Fig. 2.3.4 : Register addressing mode

### ➜ 5 Register indirect addressing

- In this case the operand memory address is by contents of register R.
- It requires one less memory access than addressing mode as seen in above point number
- Fig. 2.3.5 shows the structure of the instruction way to access the operand for the register addressing mode.

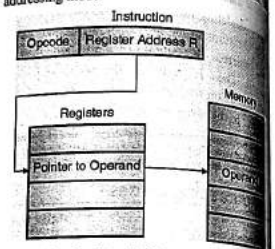



Fig. 2.3.5 : Register Indirect addressing mode

- As seen in Fig. 2.3.5, the instruction contains a address field that selects the register that memory address of the operand to be accessed.

### ➜ 6. Displacement addressing mode

- In this type of addressing mode, there are two fields that hold the base address and the displ The base address is held by the register addres in the instruction.
- The register address field in the instruction sele of the register that has the address. This addres added with the displacement and hence give address of the memory location that has the opera
- The major advantage of this type of this addr mode is that the pointer need not be contin updated; instead the displacement can be given w instruction itself.
- The disadvantage is that there is extra compu required for the address calculation.
- The structure of the instruction and the access re for the displacement addressing mode is show Fig. 2.3.6.



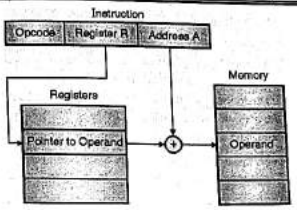



Fig. 2.3.6 : Displacement addressing mode

### ➜ 7. Relative addressing mode

- It is a version of displacement addressing where the register is the program counter, PC. Program counter is a register that always points to the next instruction to be executed by the processor and hence tells the processor about which next instruction it has to execute.
- It is used for branching or transfer of control instructions. In this case the current value of the program counter is updated with the relative address specified in the instruction. This relative address is added to the current value of program counter and hence the name as relative addressing mode.

### ➜ 8. Stack addressing mode

- This addressing mode is used to access the data from the top of the stack.
- It used PUSH and POP instructions to access the stack. In this case the operand is implicitly on top of stack.
- For example, POP AX. ; Pop top two items bytes from the top of stack.

## 2.3.1 Examples on Addressing Modes

**Ex. 2.3.1**

An instruction is stored at location 300 with its address field at location 301, the address field has the value 400. A processor register R1 contains value 200. Evaluate effective address if addressing mode of instruction is :

(a) Direct (b) Immediate
(c) Relative (d) Register Indirect
(e) Index with R1 as index register

**Soln. :**

**(a) Direct**

In this case the instruction has the address field as 400, hence the effective address is 400.

**(b) Immediate**

In this case the instruction has the address field as 400, hence the operand itself is 400. This operand is stored in the immediate next location of the instruction. Since the instruction is stored at location 300, the operand is at location 301.

**(c) Relative**

In this case the instruction has the address field as 400, which will be added with the register R1's value i.e. 200 and hence the effective address will be 600.

**(d) Register Indirect**

In this case the register provides the address of the operand. Since the register R1 has a value 200, the effective address in this case will be also 200.

**(e) Index with R1 as index register**

In this case the address field i.e. 400 will have an address that will be added with the value of the register R1 which has 200. Hence the effective address in this case will be the address at location 400 + the value of register R1 i.e. 200.

**Ex. 2.3.2**

A two address instruction is stored in memory at an address designated with the symbol W. The address field of the instruction (stored at W+1) is designated by Y. The operand used during execution of instruction is stored at address symbolized Z. An index register contains value X. State how Z is calculated from other address if addressing mode of instruction is :

(a) Direct (b) Indirect (c) Relative (d) Indexed

**Soln. :**

**(a) Direct**

In this case the instruction has the address field as Y, hence the effective address is Y. Thus Z = Y.

**(b) Indirect**

In this case the instruction has the address field as Y, hence the operand is at the address which is stored at location Y, i.e. the address field at W + 1, that has the value Y, is actually the address of the address of operand. Thus Z = [Y].

**(c) Relative**

In this case the instruction has the address field as Y, which will be added with the value of program counter. Thus Z = PC + Y.

**(d) Indexed**

In this case the address field i.e. Y will have an address that will be added with the value of the register R1 which has X. Hence the effective address in this case will be the address at location Y + the value of register R1 i.e. X. Thus Z = [Y] + X

**Syllabus Topic : Instruction Interpretation and Sequencing**

## 2.4 Instruction Interpretation and Sequencing and Micro-Operations with their Sequencing

- The structure of the CPU seen in section 2.1.1 is shown in details in Fig. 2.4.1. This structure has a speciality that all the control signals are shown in it.
- Programs are executed as a sequence of instructions. As seen in the previous sections of this chapter, each instruction consists of a series of steps that make up the instruction cycle i.e. fetch, decode, etc. Each of these steps are, in turn, made up of a smaller series of steps called micro-operations or micro-instructions.
- Control signals are issued to perform these micro-operations and micro-instructions are these control signals.
- Fig. 2.4.1 shows the structure of the CPU with these micro-instructions or the control signals.
- It also shows those register as already seen in section 2.2.1 like PC, MAR, MBR, etc.
- There are some registers like the register 'Y' to provide one of the operand to the ALU as shown in the Fig. 2.4.1.
- Another register is the 'Z' register, which is used to store the result given by the ALU.
- A "temp" register or the temporary register to store some temporary data.
- The set of registers R0 to Rn (the value of 'n' depends on the registers in the CPU) for general purpose operations.
- There is also an instruction decoder for decoding the instructions stored in the instruction register and in turn provides the micro-instructions or the control signals for the resources inside and outside the CPU. The ALU also gets the control signals from this decoder indicating the operation to be performed like Add, Sub, etc.
- The ALU also has an extra input called as C carry input as required for adder.
- To execute any instruction as seen earlier divided into three cycles viz. fetch, interrupt cycles. The execute cycle will differ the operation to be carried out in the instruction fetch and interrupt cycle will be common cycles.
- Let us see the micro-instructions to be given these cycles.

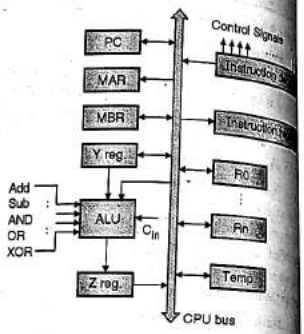



**Fig. 2.4.1 : Data path structure with control sign**

### 2.4.1 Fetch Cycle

- Fetch cycle is concerned to fetch (i.e. read memory) the instruction. It involves follo operations in different t-states (t-state is a time and is equal to one clock pulse) and hence mentioned microinstructions in Table 2.4.1.

**Table 2.4.1 : Microinstructions for the fetch cycle**

| | Operation | Microinstructions |
|---|---|---|
| t1 | PC → MAR | $PC_{out}$, $MAR_{in}$, Read, Clear y, Set $C_{in}$, Add, $Z_{in}$ |
| t2 | M → MBR<br>PC ← PC + 1 | $Z_{out}$, $PC_{in}$, Wait for memory cycle |
| t3 | MBR → IR | $MBR_{out}$, $IR_{in}$ |

- As seen in the table, three clock pulses or t-states required for the fetch cycle. Note, the control unit organizational part of the CPU, hence the design vary from processor to processor.



- In the first t-state, the address of the instruction to be executed is given to the MAR register from the PC register. To perform this operation the control signals given are $PC_{out}$ and $MAR_{in}$. This will make the PC register give out its data and the MAR register accept this data. Also the memory is indicated to perform a read operation from memory hence the signal "Read". To increment the value of PC, the various operations are performed on ALU signals i.e. Clear Y, Set $C_{in}$, Add, $Z_{in}$. The 'Y' register is cleared and the carry flag is set. Now when the ALU is said to perform the "ADD" operation it will add the contents of the 'Y' register, carry flag and the contents of the internal data bus. The contents of the internal data bus are nothing but the value given out by the PC register. Hence the PC is added with '1' i.e. the carry flag and hence incremented value of PC is given to the 'Z' register.
- In the second clock pulse the CPU has to wait for the memory operation, but in the same time it can transfer the result in 'Z' register to the PC register with the control signals namely $Z_{out}$ and $PC_{in}$. This could not be done in the previous t-state, as two data cannot be given simultaneously on the data bus, else it will get mixed up. Only one data can be given on the data bus in any clock pulse, but as many as required can accept the data.
- In the final t-state, the contents received from the memory i.e. the instruction is transferred to its correct place i.e. the instruction register. This is done by the control signals namely $MBR_{out}$ and $IR_{in}$. This also completes the entire fetch operation of the instruction.

### 2.4.2 Execute Cycle

- Execute cycle as discussed can be of various types based on the operation to be performed in the instruction and the location of the operand. We will see some examples in this subsection.
- The first example we will take for the execution of a direct addressed operand. In this case the address of the operand is directly given in the instruction. It involves different operations in various t-states as shown in Table 2.4.2 assuming the instruction ADD R1, [X].

**Table 2.4.2 : Microinstructions for the execute cycle of direct addressed mode of operand access**

| | Operation | Microinstructions |
|---|---|---|
| t1 | IR → MAR | $IR_{out}$(address), $MAR_{in}$, Read, Clear $C_{in}$ |
| t2 | M → MBR | $R1_{out}$, $Y_{in}$, Wait for memory read cycle |
| t3 | MBR + R1 → R1 | $MBR_{out}$, Add, $Z_{in}$ |
| t4 | | $Z_{out}$, $R1_{in}$ |

- In this case of direct addressing mode, the address of the memory operand is in the instruction itself. The instruction as we have seen in the fetch cycle reaches the IR register. Hence the IR register is given a signal to give out the address part and the MAR register to accept this address input value by giving the control signals $IR_{out}$(address) and $MAR_{in}$. At the same time, since the memory is to be read from the control signal is given to the memory i.e. "Read". Also the carry flag is cleared to get ready for the addition operation.
- Since the instruction expects addition of the register 'R1' and the data at memory location with address 'X', the contents of register 'R1' are transferred to the 'Y' register, which is one of the operands for any ALU operation. To perform this transfer operation the control signals given are $R1_{out}$ and $Y_{in}$. Also by the end of the second t-state, the data operand required from the memory will be available in the MBR register.
- In the third t-state the contents of the MBR, which is the content of memory location with the address 'X', is placed on the internal data bus and the ALU is indicated to perform the addition operation. It adds the contents of the 'Y' register and the contents of the internal data bus, and the result is given to the 'Z' register. An extra t-state is required to send the data from the 'Z' register to the register R1, as seen earlier two data cannot be given simultaneously on the data bus in the same t-state. And the contents of memory location with the address 'X' are already put on the data bus in the third t-state.
- The fourth t-state is thus required to transfer the data from register 'Z' to register R1 using the signals $Z_{out}$, $R1_{in}$.
- Another execute cycle we will be studying in the sub-section is for the indirect addressed operand. In this case, the address given in the instruction is the memory location that contains the address of the operand.
- The Table 2.4.3 shows the micro-operations required for such an execute cycle for an example instruction ADD R1, [[X]]

– Table 2.4.3 shows the control signals to be given exactly similar to that of the Table 2.4.2, with a minor difference i.e. the value received in the MBR on first memory read is the operand address and hence is to be given back to the memory to fetch the actual operand.

**Table 2.4.3 : Microinstructions of the execute cycle of an indirect addressed operand instruction**

| | Operation | Microinstructions |
|---|---|---|
| t1 | IR → MAR | $IR_{out}$(address), $MAR_{in}$, Read, Clear $C_{in}$ |
| t2 | M → MBR | $RI_{out}$, $Y_{in}$, Wait for memory read cycle |
| t3 | MBR → MAR | $MBR_{out}$(address), $MAR_{in}$, Read |
| t4 | M → MBR | Wait for memory read cycle |
| t5 | MBR + R1 → R1 | $MBR_{out}$, Add, $Z_{in}$ |
| t6 | | $Z_{out}$, $R1_{in}$ |

### 2.4.3 Interrupt Cycle

– It is concerned to perform the test for any pending interrupts at the end of every instruction execution and if an interrupt occurs.

– It involves the different micro-operations for various t-states as shown in Table 2.4.4.

– Here you will notice a special register used called as the stack pointer (SP), which always points to the top of the stack. This stack is used to store the return address of the interrupted program.

**Table 2.4.4 : Microinstructions for the interrupt cycle**

| | Operation | Microinstructions |
|---|---|---|
| t1 | SP ← SP – 1 | $SP_{out}$(address), Decrement, $Z_{in}$ |
| t2 | SP → MAR | $Z_{out}$, $MAR_{in}$, $SP_{in}$ |
| t3 | PC → MBR | $PC_{out}$(return address), $MBR_{in}$, Write |
| t4 | ISR address → PC | ISR address out, $PC_{in}$ (new address), Wait for memory write cycle |

– The control signals are to be generated using the control unit. The design of this control unit can be done in two ways namely : Hardwired Control Unit and Microprogrammed Control Unit. We will see these two methods in the subsequent sections.

### 2.4.4 Applications of Microprogramming

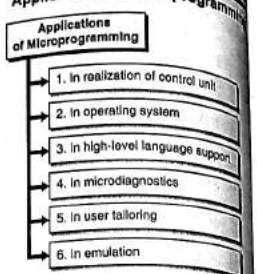



**Fig. 2.4.2 : Applications of Microprogramming**

The applications of microprogramming are :

→ **1. In realization of control unit :** Microprogramming is used widely in implementing the control unit of computers.

→ **2. In operating system :** Microprogramming is used to implement some of the primitives of operating system. This simplifies the operating system implementation and also improves performance of the operating system.

→ **3. In high-level language support :** In high-level language various sub functions and data types can be implemented using microprogramming. This makes compilation into an efficient machine language from possible.

→ **4. In microdiagnostics :** Microprogramming is used for detection isolation monitoring and correction of system errors. This known as microdiagnostics and they significant enhance system maintenance.

→ **5. In user tailoring :** By using RAM for implementing control memory (CM), it is possible to tailor the machine to different applications.

→ **6. In emulation :** Emulation refers to the use of a microprogram on one machine to execute programs originally written for another machine. This is used widely as an aid for users migrating from one computer to another.



## 2.5 Pipeline Processing

**Q. Compare pipelined vs non-pipelined system. (5 Marks)**

A processor has many resources like the ALU, buses, registers, etc. An attempt to utilize all these attributes to their fullest or continuously can be achieved by pipelining. In a pipelined system the instructions flow through the processor as if the processor was a pipe. The instructions move from one stage to another to accomplish the assigned operation. Hence at most of the times each unit of the processor is busy handling one or the other instruction, making the attribute of the processor being used continuously. This chapter deals with advanced pipelining and superscalar design in the processor development. We will go through the concepts and design issues of the linear and non-linear pipelining. We will also discuss collision-free scheduling techniques for performing dynamic functions. Techniques to design instruction pipelines, arithmetic pipelines are also discussed.

### 2.5.1 Non-Pipelined System vs. Two Stage Pipelining

– In a non-pipelined system, the processor fetches an instruction from memory, decodes it to determine what the instruction was, read the instructions inputs from the register file, performs the computation required by the instruction and writes the result back into the register file. This approach is also called as un pipelined approach.

– The problem with this approach is that, the hardware needed to perform each of these steps (Instruction fetch, instruction decode, register read, instruction execution and register write-back) is different and most of the hardware is idle at any given moment. Waiting for the other parts of the processor to complete their part of executing an instruction.

– Pipelining is a technique for overlapping the execution of several instructions to reduce the execution time of a set of instructions.

– Two stage pipelining includes two stages i.e. Fetch instruction and Execute instruction.

– These two operations are performed for one instruction and the next instruction overlapping i.e. when the first instruction is being executed the next is fetched and when this instruction is executed the next is fetched and so on.

– This method of execution the instructions in pipeline speeds up the processor operation.

– This also makes sure that all the units of the processor are busy operating and none of them is starving.

– Thus with the help of pipelining the operation speed of the processor increases i.e. more the number of pipeline stages, faster becomes the processor, but complex in design.

– Two stage instruction pipeline stage is as shown in Fig. 2.5.1(a)

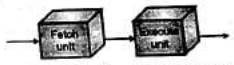
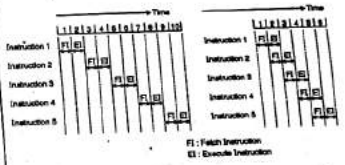



**Fig. 2.5.1 (a) : Two stage pipeline architecture**




**(b) Timing diagram of execution of instructions in non-pipelined system.** **(c) Timing diagram of execution of instructions in pipelined systems.**

**Fig. 2.5.1**

– In case of a system without pipelining the time required for executing a set of instructions is much more than the time required executing the same set of instructions in a pipelined system.

– The comparison of the execution of five instructions in a system with and without pipeline is shown in Figs. 2.5.1 (b) and 2.5.1(c).

– You will notice that the time required for executing five instructions on a non-pipelined system is 10 clock pulses while that on two stage pipelined processor is 6 clock pulses. Thus the number off clock pulses required in two stage processor will always be $x/2 + 1$, where 'x' is the number of clock pulses in non-pipelined instructions and '2' is the number of stages.

– If we increase the number of instructions, we can make the expression as x/2 (since '1' is negligible for huge number of instructions) clock pulses for a two stage pipelined processor, wherein 'x' clock pulses in case of non-pipelined processor.

- If we try to generalize this expression, we can write it as x/n, where x is number of clock pulses required for non-pipelined instructions and 'n' is the number of stages of a pipelined processor.
- Thus we can say that the speed-up achieved by a pipelined processor can be maximum 'n' times of the non-pipelined processor.

**Syllabus Topic : Basic pipelined Datapath and Control**

## 2.5.2 Basic pipelined Datapath and Control for a Six Stage CPU Instruction Pipeline

**(MU - May 2014, Dec. 2014, May 2015, Dec. 2015)**

Q. Write a short note on six stage pipelined system. (8 Marks)

Q. What is instruction pipelining? May 14, 6 Marks

Q. Explain six stage instruction pipelines with suitable diagram. Dec. 14, 10 Marks

Q. What is instruction pipelining? What are advantages of pipelining? May 15, Dec. 15, 6 Marks

The flowchart given in Fig. 2.5.2 gives an instruction flow through the six stage pipelined processor.

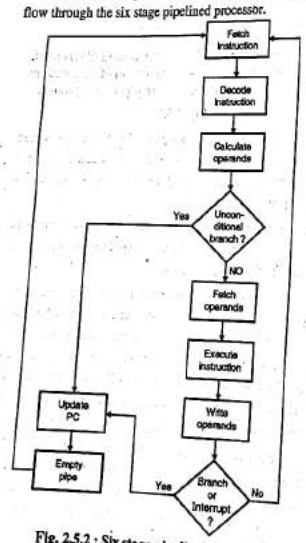
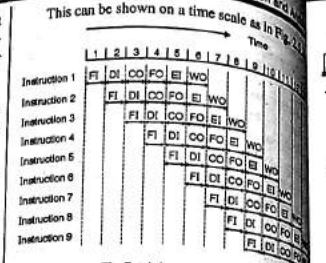



**Fig. 2.5.2 : Six stage pipeline flowchart**

This can be shown on a time scale as in Fig. 2.5.3




Where,

FI : Fetch Instruction
DI : Decode Instruction
CO : Calculate Operand address
FO : Fetch Operand
EI : Execute Instruction
WO : Write Operand result

**Fig. 2.5.3 : Six stage pipelined processor timing diagram**

- Pipelining is termed as overlapped parallelism. In case of pipelining the instructions are overlapped in execution of the instructions is overlapped in a manner that there are several instructions in different process in the pipelined processor as shown in Fig. 2.5.3.
- As shown in the Fig. 2.5.3, for e.g. during the 6th pulse, there are six instructions in the pipeline. Instruction 1 has its result being written in instruction 2 is being executed, instruction 3 has operands being fetched, instruction 4 has the address operands being calculated, instruction 5 is decoded and instruction 6 is being fetched. This how pipelining is a overlap parallelism of instruction
- This six stage pipeline system can be implemented six units as shown in Fig. 2.5.4

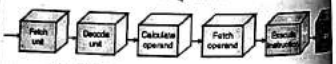



**Fig. 2.5.4 : Six stage pipelined architecture**

- In pipelining, when a branch instruction is executed system causes a huge waste of time i.e. processor starving. The instructions in the pipeline are sequential instructions.
- If a branching instruction is given the next instruction to be executed is not the sequential one, instead it instruction at target instruction.



- Hence the sequential instructions in the pipeline are to be cleared and instructions from target are to be fetched. Clearing the sequential instructions from the pipeline is called as flushing of the pipeline.
- These problems are discussed in detail in section 2.6. Also the solutions to the same are discussed in that section. This is as shown in the timing diagram in Fig. 2.5.5.

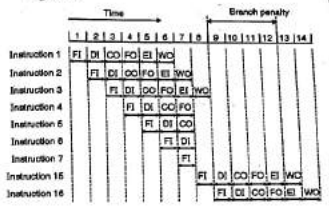



**Fig. 2.5.5 : Branch in a six-stage pipeline**

**Ex. 2.5.1**

Draw the space-time diagram for a six-segment pipeline showing the time it takes to process eight tasks.

**Soln. :**

| Clock cycles | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Segment : 1 | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | - | - | - | - | - |
| 2 | - | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | - | - | - | - |
| 3 | - | - | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | - | - | - |
| 4 | - | - | - | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | - | - |
| 5 | - | - | - | - | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | - |
| 6 | - | - | - | - | - | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 |

It takes 13 clock cycles to process 8 tasks.

## 2.5.3 Linear Pipeline Procesors

- In a linear pipelined processor there are n stages connected linearly to perform different operations. These may perform different operations to execute an instruction, perform arithmetic operations or memory access operations.
- In a linear pipelined processor with suppose n stages, the partially processed instructions are passed from stage i to stage i + 1, where i varies from 1 to k – 1. The linear pipeline can either be a synchronous system or asynchronous system.

### 2.5.3.1 Asynchronous and Synchronous Linear Pipelining

- In case of an asynchronous linear pipeline system, there is a set of handshaking signals between the two stages. Whenever a stage (say stage i) completes its operation, it places the result on the input lines of next stage (i.e. stage i + 1) and enables the ready (or strobe) signal. The next stage (i.e. stage i + 1) on completing its operation, accepts the data from its input lines and indicates this to the previous stage (i.e. stage i) by giving an acknowledgement signal. On this, the stage which had placed the data (i.e. stage i), also checks its input if it has previous stage (i.e. stage i – 1) completed its operation and is ready with the result.
- It also repeats the same process as explained with stage i and i + 1. This can be explained as shown in the Fig. 2.5.6.


**Fig. 2.5.6 : Asynchronous linear pipelining system**

- Hence the asynchronous linear pipelined system will have variable throughput rate and will experience different amount of delay at each stage.
- In case of a synchronous linear pipelined system the stages are separated by latches. Whenever a stage completes its part of operation it stores the result in the latch. The clock signal is synchronously given to all the latches, such that on reception of the clock signal each stage takes the output of the latch connected to its input. This system is shown in Fig. 2.5.7.

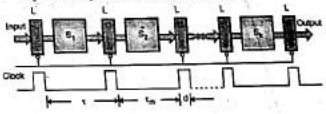



**Fig. 2.5.7 : Synchronous linear pipelining system**

- The latches are infact master-slave flipflops. The time required by each stage is expected to be equal; and it is this time that determines the clock period as well as the speed of the pipelined system.
- The utilization of the stages or the utilization pattern of stages in a synchronous pipeline can be represented by the reservation table. The reservation table follows a diagonal line for synchronous linear pipeline as shown in Fig. 2.5.8.

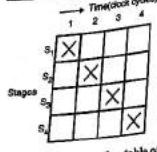



**Fig. 2.5.8 : Reservation table of a synchronous linear pipeline**

- Reservation table is a space-time diagram showing streamline pattern. Hence as seen in Fig. 2.5.8, for an n-stage pipeline, n clock pulses are required to execute the instruction.
- Once the pipeline is filled up completely, the processor completes one instruction execution every clock pulse.

### 2.5.3.2 Clocking and Timing Control

- The clock cycle, $\tau$ as shown in Fig. 2.5.7, can be calculated as discussed below. Let $\tau_i$ be the delay time for stage $S_i$. Hence the **clock cycle** time can be given as :

$$\tau = \max \left[ \tau_i \right]_1^k + d = \tau_m + d$$

where $\tau_m$ the maximum stage delay and d is, as shown in the Fig. 2.5.7, the 'on' period of the clock pulse.

- The data from each stage is latched in the master flipflop of latch register during the rising edge and given to the slave flipflop during the falling edge. In fact, $\tau_m >> d$; hence we can say that, $\tau \equiv \tau_m$.
- Hence the pipeline frequency can be given as $= 1 / \tau$. This frequency f, is also termed as the throughput of the system as it gives the time required for one instruction to come out of the pipeline.
- The actual **throughput** of the pipeline may be less than the maximum throughput given by f, which may be because of more than one clock pulses, may be required for the initiation of successive instructions.
- The initiation of successive instructions may take more clock pulses because of their data or control dependency. The clock pulse at each stage is expected to arrive simultaneously. But, because of the time delay of the path, different stages get the pulse at different time offset $s$; this problem is referred to as **clock skewing**. Assuming the shortest logic path to get the clock at a delay of $t_{min}$ and the longest logic path to get the clock pulse at delay of $t_{max}$ : to avoid [illegible] the $\tau_m \geq t_{max} + s$ and $d \leq t_{min} - s$. Thus the [illegible] with skew is :

$$d + t_{max} + s \leq \tau \leq \tau_m + t_{min} - s$$

Hence in ideal case, $s = 0$, $t_{max} = \tau_m$ and $t_{min}$ [illegible] even with the clock skewing $\tau = \tau_m + d$

### 2.5.3.3 Speedup, Efficiency and Throu[illegible]

- A linear pipeline of k stages will take $k + (n$ [illegible] cycles to execute n instructions; the first [illegible] will take k clock cycles and the remaining [illegible] instructions will take one clock cycle each, [illegible] there are no dependency of the instructions [illegible] with the clock cycle width being $\tau$, the [illegible] required to execute these n instructions will be

$$T_k = \tau [k + (n-1)]$$

An equivalent non-pipelined system the time [illegible] to execute n instructions will be

$$T_1 = nk\tau$$

Thus the speedup factor of a k-stage pipeline can be given as :

$$S_k = \frac{T_1}{T_k} = \frac{nk\tau}{k\tau + (n-1)\tau} = \frac{nk}{k + (n-1)}$$

- Hence as the number of instructions n increa[illegible] tends to k. Thus, **'k' stage pipeline processor at most 'k' times faster than the corresp[illegible] non-pipelined processor.** The speed up fac[illegible] function of the number of instructions is shown Fig. 2.5.9.

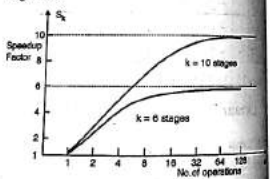



**Fig. 2.5.9 : Relationship of speedup factor with the [illegible] operations**

- Also there is a limit to the number of stages, [illegible] number of stages increases the delay and [illegible] increases.
- Hence the finest pipelining or micro-pipelining [illegible] the subdivision of the stages at almost gate level [illegible] consider this optimal number of stages.



- Most of the systems have the number of stages varying from 2 to 15. Very few systems are designed to have the number of stages beyond 10.
- This is because the increase in the number of stages should take care that the PCR (performance to cost ratio) has also increased. But beyond a particular number of stages, termed $k_0$ (optimal no. of stages), the PCR starts reducing as shown in the Fig. 2.5.10.

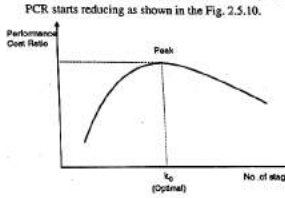



**Fig. 2.5.10 : Graph of PCR vs. k**

- To execute a program on sequential non-pipelined system, say the time required is 't'. Thus to execute this program on a k-stage pipeline with equal flow-through the time required $= \frac{t}{k} + d$, where d is latch delay.

Hence, $$f = \frac{1}{\frac{t}{k} + d}$$

The performance / cost ratio (PCR) can be defined as :

$$PCR = \frac{f}{(c + kh)} = \frac{1}{\left(\frac{t}{k} + d\right)(c + kh)} \quad ...(2.5.4)$$

where h is the cost of each latch, c is the cost of all logic gates of a stage, d is the latch delay, f is the pipeline frequency, k is the number of stages and t is the flow-through delay of each stage. The optimal number of stages can be given as

$$k_0 = \sqrt{\frac{t \cdot c}{d \cdot h}} \quad ...(2.5.5)$$

The efficiency of such a system is defined as (using Equation (2.5.3))

$$E_k = \frac{S_k}{k} = \frac{n}{k + (n-1)} \quad ...(2.5.6)$$

The pipeline throughput $H_k$ is defined as number of operations performed per unit time.

$$H_k = \frac{\text{Efficiency}}{\text{One unit time } (\tau)} = \frac{n}{[k + (n-1)]\tau}$$

$$= \frac{nf}{k + (n-1)} \quad ...(2.5.7)$$

Hence the maximum throughput as discussed earlier will be equal to f, when the number of instructions n tends to $\infty$.

**Ex. 2.5.2**

Consider the execution of a program of 15,000 instructions by a linear pipeline processor with a clock rate of 25 MHz. Assume that the instruction pipeline has five stages and that one instruction is issued per clock cycle. The penalties due to branch instructions and out-of-sequence executions are ignored.

(a) Calculate the speedup factor in using this pipeline to execute the program as compared with the use of an equivalent non-pipelined processor with an equal amount of flow-through delay.

(b) What are the efficiency and throughput of this pipelined processor ?

**Soln. :**

**Given :** n = 15000, f = 25 MHz, k = 5

Speed up factor $(S_k) = \frac{nk}{k + (n-1)}$

$$= \frac{15000 \times 5}{5 + (15000 - 1)} = 4.9986$$

Efficiency $(E_k) = \frac{S_k}{k} = 0.99973$

Throughput $(H_k) = \frac{E_k}{k} = f E_k = 25 \text{ MHz} \times 0.99973$

$= 24.9933$ MIPS

**Ex. 2.5.3**

A non-pipeline system takes 50 ns to process a task. The same task can be processed in a six stage pipeline with a clock cycle of 10 ns. Determine the speedup and the efficiency of the pipeline for 100 tasks. What is the maximum speedup and efficiency that can be achieved ?

**Soln. :**

**Given :** For the non-pipeline system : $t_n = 50$ ns

For the pipeline system : k = 6, $t_p = 10$ ns

Number of tasks n = 100

Speed up $(S_k) = \frac{T_1}{T_k} = \frac{nk\tau_1}{k\tau + (n-1)\tau}$

$$= \frac{100 \times 50}{6 \times 10 + (100 - 1) \times 10} = 4.7619$$

Maximum speedup will occur when the number of tasks (n) is very large (n >> k). Hence neglecting the term k – 1, we have Max Speedup $= \frac{n\tau_1}{n\tau} = \frac{50}{10} = 5$

Efficiency $(E_k) = \frac{S_k}{k} = \frac{4.7619}{6} = 0.7936$

Cosidering the max speedup the max efficiency $= \frac{5}{6}$

$= 0.8333$

## 2.5.4 Non Linear Pipeline Processors

- A dynamic or multi-function pipeline is called as non-linear pipeline. In a linear pipeline the operations that are being performed are fixed; each stage as a fixed operation.
- But in a non-linear pipeline allows feed forward and feedback connections in addition to the streamline connection. It may also have more than one output i.e. the output need not be from the last stage. An example of three stage non-linear pipeline system is shown in Fig. 2.5.11.

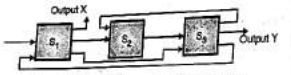



**Fig. 2.5.11 : A 3-stage non linear pipeline**

- In the Fig. 2.5.11 besides the three stages connected in streamline, there are also some feedback and feedforward connections.
- The feedforward connection is from $S_1$ to $S_3$ and feedback connections from $S_2$ to $S_2$ and $S_3$ to $S_1$. The reservation table for such connections may be different for different operations.
- There are two examples of different operations say, X and Y, for which the reservation table is shown in Fig. 2.5.11.

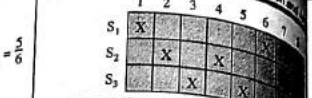



**Fig. 2.5.12(a) : Reservation table for**

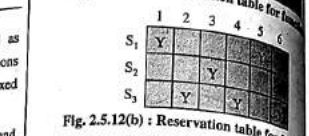
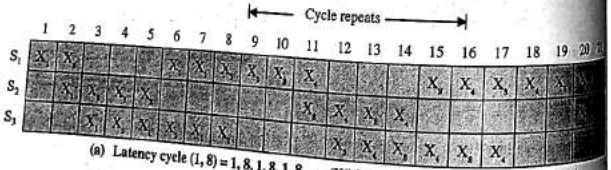



**Fig. 2.5.12(b) : Reservation table for**

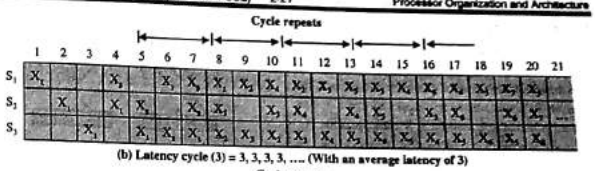
- The Fig. 2.5.12 shows the requirement of stages at different times for doing the operation. For e.g. the function X, has first to $S_1$, then $S_2$, then $S_3$, then $S_2$, then $S_3$, and finally to $S_3$ which will give the output.
- Similarly the function Y is first given to $S_1$, then $S_2$, then $S_3$, then $S_1$ and finally to $S_3$ give the output. The number of reservation table corresponds to the that function. Hence from Fig. 2.5.12, an evaluation time of 8 clock cycles has an evaluation time of 6 clock cycles.
- A pipeline initiation table consists of when the next time the same function can be The number of time units (clock cycle between the two initiations of a function is **latency period** between them.
- Some valid latencies or latency sequences cause any collision are shown in Fig. latencies that do not cause collision are **latency sequence** while latencies that are called as **forbidden latencies**. forbidden latencies are shown in Fig. 2.5.14.



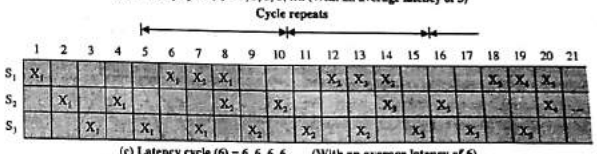

(a) Latency cycle (1, 8) = 1, 8, 1, 8, 1, 8, ..., (With an average latency of 4.5)

**Fig. 2.5.13 Conti...**



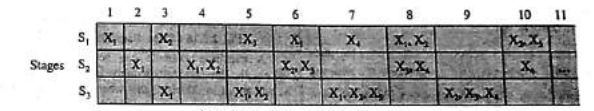



(b) Latency cycle (3) = 3, 3, 3, 3, .... (With an average latency of 3)

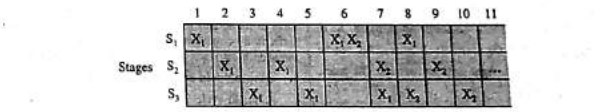



(c) Latency cycle (6) = 6, 6, 6, 6, ..., (With an average latency of 6)

**Fig. 2.5.13 : Example latencies of function X that do not cause collision**




(a) Collision with scheduling latency 2




(b) Collision with scheduling latency 5

**Fig. 2.5.14 : Example forbidden latencies i.e. latencies that cause collision of function X**

- As shown in Fig. 2.5.13, forbidden latencies are 2 and 5. Besides, 4 and 7 are also forbidden latencies. To detect the forbidden latency, you need to check the distance between the two marks on the reservation table in a row.
- For e.g. in case of function X, as shown in Fig. 2.5.12(a), the distance between two marks in $S_1$ is 5 or 2. Average latency of a latency cycle is defined as the ratio of sum of all latencies to the number of latencies along the cycle. For e.g. (1,8) latency as shown in Fig. 2.5.13(a), has an average latency of (1+8)/2 = 4.5.
- The average latency of a constant cycle, i.e. latency cycle that contains only one latency value, is same as the constant value. For e.g. the average latency of the cycle 3 and 6, as shown in Figs. 2.5.13(b) and (c) are 3 and 6 respectively.

### 2.5.4.1 Collision Free Scheduling or Job Sequencing

- As seen in non-linear pipelining, we cannot sequence the instructions (job) as in linear pipelining; else there would be collision.
- Hence we need to have optimal job sequencing or scheduling technique. In a non-linear pipelining while scheduling, the main aim is to obtain smallest average latency without any collision.
- This pipeline design theory requires the concepts of collision vectors, state diagrams, single cycles, greedy cycles and minimal average latency (MAL).

### 1. Collision vectors :

As seen in the previous section, we can separate the permissible latencies and the forbidden latencies using the reservation table. For a reservation table with n columns, the maximum forbidden latency (m) should be $\leq n - 1$. The permissible latency (p), must be as small as possible. An ideal case would be p = 1. This smallest latency i.e. p = 1 is possible in linear pipelining, but in non-linear pipelining, it is difficult to achieve.

A **collision vector** is an 'm' bit binary vector ($C = C_m\ C_{m-1}.....C_2\ C_1$), that shows the set of permissible and forbidden latency. In a collision vector, a bit $C_i$ is '1' if the latency i causes a collision, else it is '0'. For e.g., the reservation tables seen in Fig. 2.5.12, will have the collision vectors as $C_x$ = (1011010) and $C_y$ = (1010). Thus for $C_x$, there is a permissible latency 1, 3 and 6 while forbidden latencies of 7, 5, 4 and 2.

### 2. State Diagrams :

From the collision vector, we can make the state diagram for the pipeline. The collision vector $C_x$, achieved above is called as the initial collision vector. When loaded in a register and shifted right, each bit at the output corresponds to an increase in latency. A '1' at the output indicates collision, while a '0' indicates no collision. A '0' is inserted from the left for every clock cycle. This can be implemented as said by a right shift register and OR gates, as shown in Fig. 2.5.15.

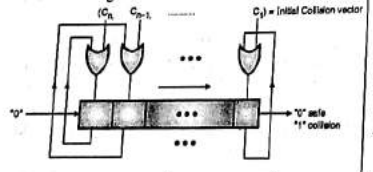
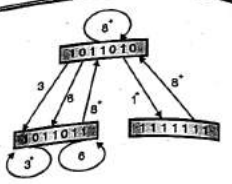
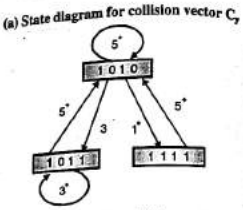



**Fig. 2.5.15 : n-bit right shift register for state transition**

The state transition diagram can be constructed using this state register. The next state at time t + p, where p is the permissible latency and t is some no. of clock pulses, obtained by shifting the register for p times and ORing it with the initial collision vector. The state diagrams for the collision vectors $C_x$ and $C_y$ are as shown in Fig. 2.5.16.




**(a) State diagram for collision vector $C_7$**




**(b) State diagram for collision vector $C_7$**

**Fig. 2.5.16 : State diagrams for collision**

A transition example can be explained as below. For e.g. the three bit shifts with the initial vector of function X, will result in 0001011; this when ORed with the initial collision vector results in 1011011. The state diagram (Refer Fig. 2.5.16(a)) shows this transition for the function X. If the number of shifts is greater than m, the next state is same as the initial collision vector. For e.g. if the number of shifts is 8 or more in Fig. 2.5.16(a), it comes back to the initial collision state. This transition is denoted by 8*.

### 3. Single cycle and Greedy cycle

A single cycle is one in which any state appears not more than once. For example for the X function state diagram shown in Fig. 2.5.16(a), the different single cycles are (3), (6), (8), (1,8), (3,8) and (6,8). A cycle that travels for more than one time through the same state is a greedy cycle. some greedy cycles in the Fig. 2.5.16(a) are (1,8,3,8), (1,8,6,8), (3,6,3,8,6) etc.

### 4. Minimal Average Latency (MAL)

We have already studied the minimal average latency in the previous sections. There are some bounds on this value of MAL. These bounds are listed below :

(i) The lower bound of MAL is the maximum number of checkmarks in any row of the reservation table

(ii) The MAL should be lower than or equal to the latency of any greedy cycle in a reservation table.

(iii) The upper bound of MAL is equal to the number of 1's in the initial collision vector plus 1.



**Ex. 2.5.4**

Consider following pipeline reservation table.

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| $S_1$ | X | | | X |
| $S_2$ | | X | | |
| $S_3$ | | | X | |

(a) What are the forbidden latencies ?

(b) Draw the state transition diagram.

(c) List all simple cycles and greedy cycles.

(d) Determine the optimal constant latency cycle and the minimal average latency.

(e) Let the pipeline clock period be $\tau$ = 20 ns. Determine the throughput of this pipeline.

**Soln. :**

(a) Collision Vector = (100)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $x_1$ | $x_2$ | $x_3$ | $x_1x_4$ | $x_2x_5$ | $x_3$ | $x_4$ | $x_5$ |
| | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | ... | ... |
| | | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | ... |

Collision with latency 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $x_1$ | | | $x_1x_2$ | | | $x_2x_3$ | | $x_3$ |
| | $x_1$ | | | $x_2$ | | | $x_3$ | ... |
| | | $x_1$ | | | $x_2$ | | | $x_3$ ... |

Collision with latency 3

Hence latencies (1) and (3) are forbidden latencies.

(b) State transition shift register

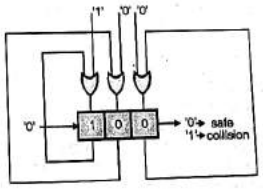



**Fig. Ex. 2.5.4**

Using the above shift register, we can generate the following state transition diagram.

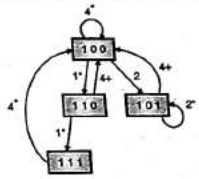
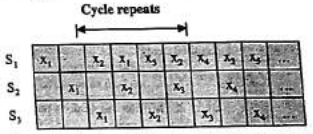



**Fig. Ex. 2.5.4(a)**

(c) As seen in the state transition diagram,
Simple cycles : (2), (4), (1,4), (1,1,4) (2,4) etc.
Greedy cycles : (1,4,2,4) , (1,1,4,2,4) etc.

(d) Optimal constant latency (2)
∴ Minimum average latency (MAL) = 2

(e) Throughput




As seen in the above table, one instruction is executed every two cycles.

$$\therefore \text{Throughput} = \frac{1}{2} \times f = \frac{1}{2} \times \frac{1}{20 \text{ nsec}}$$

$$= 25 \text{ MIPS}$$

**Ex. 2.5.5**

A non-pipelined processor X has a clock rate of 25 MHz and an average CPI (cycles per instruction) of 4. Processor Y, an improved sucessor of X, is designed with a five-stage linear instruction pipeline. However, due to latch delay and clock skew effects, the clock rate of Y is only 20 MHz.

(a) It is a program containing 100 instructions is executed on both processors, what is the speedup of processor Y compared with that of processor X?

(b) Calculate the MIPS rate of each processor.

**Soln. :**

**(a) Program has 100 Instruction**

i.e. n = 100

Time taken by a non-pipelined (X) processor to execute this program ($T_1$) = nkτ.

where n = number of instruction = 100

k = number of cycles per instruction = 4

$\tau$ = clock width = $\frac{1}{25\ MHz}$

$I_1 = nk\tau = 100 * 4 * \frac{1}{25\ MHz} = 16\ \mu sec.$

For a 5-stage pipelined (Y) processor, the time required to execute n-instruction $(T_k) = [k + (n-1)]\ \tau$.

where k = 5 stages

n = 100 instructions

$\tau = \frac{1}{f} = \frac{1}{20MHz} = 0.05\ \mu sec$

$T_k = [5 + (100 - 1)] * 0.05\ \mu sec$

$= 5.2\ \mu sec$

$\therefore$ Speedup = $\frac{T_1}{T_k} = \frac{16\ \mu sec}{5.2\ \mu sec} = 3.07$

**(b) Non-pipelined processor (X)**

| Time taken | Instructions |
|---|---|
| 16 μsec | 100 |
| 1 sec | x |

$\therefore$ MIPS rate, (x) = $\frac{100\ Instruction}{16\ \mu sec}$

= 6.25 MIPS (Million instructions per second)

**Pipelined processor (Y)**

| Time taken | Instructions |
|---|---|
| 5.2 μsec | 100 |
| 1 sec | x |

$\therefore$ MIPS rate, (x) = $\frac{100\ Instruction}{5.2\ \mu sec}$ = 19.23 MIPS

**Ex. 2.5.6**

*Consider the following pipeline reservation table*

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | × | | | | | | | | × |
| 2 | | × | × | | | | | × | |
| 3 | | | | × | | | | | |
| 4 | | | | | × | × | | | |
| 5 | | | | | | | × | × | |

*(i) Determine latencies in the forbidden list F and collision vector C*

*(ii) Draw state Transition diagram*

*(iii) List all simple cycles and greedy cycles*

*(iv) Determine MAL*

**Soln. :**

**(i) Forbidden latencies F = (1, 5, 6, 8)**

$\therefore$ Collision vector C = (10110001)

(Since the total number of tasks are 9, there are 8 bits in collision vector i.e. $C_8$ to $C_1$)

**(ii) State transition diagram**

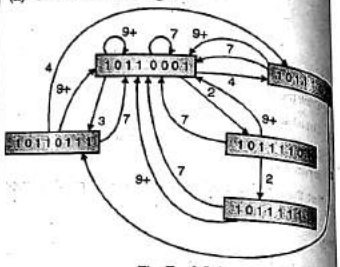



Fig. Ex. 2.5.6

**(iii) Latency cycles :**

(7), (2, 7), (2, 2, 7), (4, 7), (4, 3), (3, 7), (4, 3, 4, 7), (9), (2, 9), (4, 9) (3, 9)

**Simple cycles :**

(7), (2, 7), (4, 7), (4, 3), (3, 7), (9), (2, 9), (4, 9), (3, 9)

**Greedy cycles :** (2, 2, 7), (4, 3, 4, 7)

**(iv) Optimal control latency : (4, 3)**

$\therefore$ Minimum average latency (MAL) = $\frac{4+3}{2} = 3.5$

**Ex. 2.5.7**

Consider the following pipeline reservation table.

| Clock / cycle / Stage | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| $S_1$ | × | | × | | | | × |
| $S_2$ | | | | × | | × | |
| $S_3$ | | | × | | × | | |

(i) Determine latencies in Forbidden list F and collision vector C

(ii) Draw the state transistor diagram

(iii) List all simple cycles and greedy cycles

(iv) Determine minimum average latency (MAL)

(v) For a pipeline clock period $\tau$ = 20 ns. Determine maximum throughput of the pipeline.

**Soln. :**

**(i) Forbidden latencies = (2, 4, 6)**

Collision vector C = (101010)



(Since the total number of tasks are 7, there are 6 bits in the collision vector i.e. $C_6$ to $C_1$)

**(ii) State transition diagram**

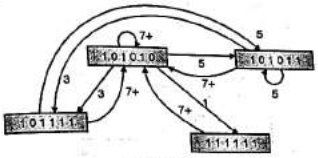



Fig. Ex. 2.5.7

**(iii) Latency cycles :**

(7), (1, 7), (3, 7), (3, 5), (5), (3, 7, 5, 3, 7), (5, 3, 7)

**Simple cycles :**

(7), (1, 7), (3, 7), (3, 5), (5), (5, 3, 7)

**Greedy cycles :** (3, 7, 5, 3, 7)

**(iv) Optimal constant latency :** (1, 7) or (3, 5)

$\therefore$ Minimal average latency (MAL) = $\frac{1+7}{2} = 4$

(v) Since MAL = 4, 1 instruction will be executed every 4 cycles.

$\therefore$ Throughput = $\frac{1}{4} * f = \frac{1}{4} * \frac{1}{20\ nsecs}$

($\because$ frequency = $\frac{1}{\tau} = \frac{1}{20\ nsecs}$)

= $\frac{1}{70\ nsecs}$ = 25 MIPS

**Ex. 2.5.8**

For a unifunction pipeline, the forbidden set of latencies is as given below.

F = {1, 3, 6} with the largest forbidden latency = 6

(i) Obtain collision vector

(ii) Draw the state diagram

(iii) State all simple and greedy cycles

(iv) Obtain MAL

**Soln. :**

(i) Collision vector (C) = (100101)

(ii)

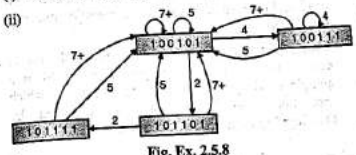



Fig. Ex. 2.5.8

(iii) Latencies : (5), (4), (4, 5), (2, 5), (2, 2, 5), (7), (2, 7), (4, 7)

Simple latencies : (5), (4), (4, 5), (2, 5), (7), (2, 7)

Greedy latency : (2, 2, 5)

(iv) Optimal latency : (2, 2, 5)

$\therefore$ Minimal average latency (MAL) = $\frac{2+2+5}{3} = 3$

**Note : Optimal latency is the latency that gives minimal average i.e. the average latency minimum.**

**Ex. 2.5.9**

For the following reservation table, determine collision vector, state transition diagram and MAL.

| Clock cycle / Stage | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| $S_1$ | × | | | | | × | | |
| $S_2$ | | × | | | | × | × | |
| $S_3$ | | | × | | × | | | × |
| $S_4$ | | | | × | | | | |

Also find the throughput for $\tau$ = 10 nsec

**Soln. :**


Fig. Ex. 2.5.9

(i) Collision vector (C) = (0011111)

(ii) State transition diagram : (Refer Fig. Ex. 2.5.9)

(iii) Latencies : (6), (7)

Simple latencies : (6), (7)

Greedy latencies : NIL

(iv) Optimal latency : (6)

Minimal average latency (MAL) = 6

(v) Since MAL = 6, 1 instruction takes 6 clock pulses

$\therefore$ Throughput = $\frac{1}{6} * f = \frac{1}{6} * \frac{1}{10\ nsec}$

= 16.67 MIPS

**Ex. 2.5.10**

For the following reservation table, determine collision vector state transition diagram and MAL.

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $S_1$ | × | | | × | |
| $S_2$ | | | × | | × |
| $S_3$ | | × | | | |

Also find the throughput for $\tau$ = 25 nsecs

**Soln. :**

(i) Collision vector (C) = (0110)

(ii) State transition diagram : (Refer Fig. Ex. 2.5.10)

(iii) Latencies : (4), (1,4)

Simple latencies : (4), (1,4)

Greedy latencies : NIL

(iv) Optimal latency : (1,4)

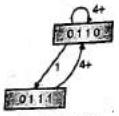



Fig. Ex. 2.5.10

Minimal average latency (MAL) = $\frac{1+4}{2} = 2.5$

(v) Since MAL = 2.5, 1 instruction takes 2.5 clock pulses

$\therefore$ Throughput = $\frac{1}{2.5} * f = \frac{1}{2.5} * \frac{1}{25 \text{ nsec}}$

= 16 MIPS

**Ex. 2.5.11**

A certain pipeline with the four stages S1, S2, S3 and S4 is characterized by the following Table Ex. 2.5.11.

**Table Ex. 2.5.11**

| | $t_0$ | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ |
|---|---|---|---|---|---|---|---|
| $S_1$ | X | | | | | X | |
| $S_2$ | | | X | | | | X |
| $S_3$ | | X | | X | | | |
| $S_4$ | | | X | | X | | |

(i) Determine the latencies in the forbidden list F and the collision vector C.

(ii) Determine the minimum constant latency L by checking the forbidden list

(iii) Draw the state diagram for this pipeline and determine MAL.

**Soln. :**

(i) Forbidden latencies = (2, 4, 5)

Collision vector C = (011010)

(ii) State transition diagram

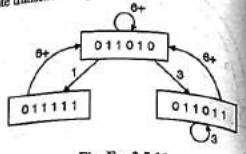



Fig. Ex. 2.5.11

(iii) Latency cycle : (6), (1, 6), (3), (3, 6), (3, 3, 6), (6, 1, 6), (3, 6, 6)

Simple cycles : (6), (1, 6), (3), (3, 6)

Greedy cycles : (3, 3, 6), (6, 1, 6), (3, 6, 6)

(iv) Optimal constant latency : (3)

Minimum average latency (MAL) = 3

## 2.6 Instruction Pipelining and Pipelining Stages

- Instruction pipelining is a technique for overlapping the execution of several instructions to reduce the execution time of a set of instructions.
- Generally, the processor fetches an instruction from memory, decodes it to determine what the instruction was, read the instructions inputs from the register file, performs the computation required by the instruction and writes the result back into the register file. This approach is called unpipelined approach.
- The problem with this approach is that, the hardware needed to perform each of these steps (Instruction fetch, instruction decode, register read, instruction execution and register write-back) is different and most of the hardware is idle at any given moment. Waiting for the other parts of the processor to complete their part of executing an instruction.
- Pipelining is a technique for overlapping the execution of several instructions to reduce the execution time of a set of instructions.
- Each instruction takes the same amount of time to execute in a pipelined processor as it would in a non-pipelined processor, but the rate at which instructions can be executed is increased by **Overlapping Instruction Execution.**



**Latency** is the amount of time that a single operation takes to execute.

**Throughput** is the rate at which operations get executed.

In a non-pipelined processor,

$$\text{Throughput} = \frac{1}{\text{latency}}$$

[expressed as operations/second or operations/cycles]

In a pipelined processor,

$$\text{Throughput} > \frac{1}{\text{latency}}$$

**Pipelining :** To implement pipelining, designers divide a processor's data path into sections called stages and place pipeline latches between each section.

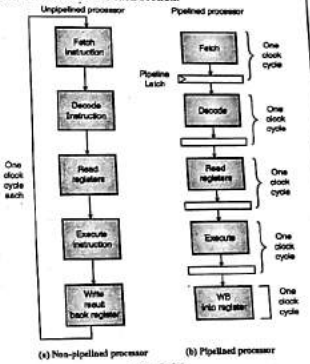



Fig. 2.6.1

As shown in Fig. 2.6.1, at start of each cycle, the pipeline latches read their inputs and copy them to their outputs.

The amount of data path that a signal travels through in one cycle is called a stage of the pipeline.

A **five-stage pipeline** is shown in Fig. 2.6.1(b).

Stage 1 : Fetch block

Stage 2 : Decode block

Stage 3, 4, 5 are subsequent blocks in execution process.

Fig. 2.6.2 shows the Instruction flow in a pipelined processor.

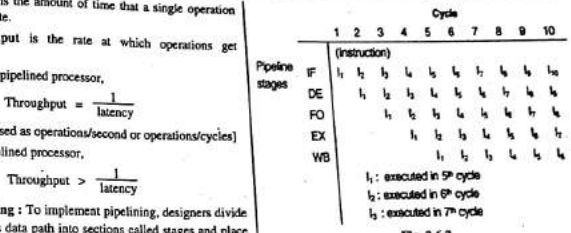



Fig. 2.6.2

Cycle time of a pipelined processor is calculated as

$$\text{Cycle time (pipelined)} = \frac{\text{Cycle time (unpipelined)}}{\text{Number of pipeline stages}} + \text{Pipeline latch latency}$$

As, the number of pipeline stages increases, the pipeline latch latency increases which in turn limits the benefit of dividing a processor into a very large number of pipelining stages.

**Ex. 2.6.1**

An unpipelined processor has a cycle time of 25 ns. What is the cycle time of a pipelined version of the processor with 5 evenly divided pipeline stages, if each pipeline latch has a latency of 1 ns?

**Soln. :**

$$\text{Cycle time pipelined} = \frac{\text{Cycle time unpipelined}}{\text{Number of stages of pipeline}} + \text{Pipeline latch latency}$$

$$= \frac{25 \text{ ns}}{5} + 1 \text{ ns} = 6 \text{ ns}.$$

To find the speedup of the execution process in a pipelined processor,

Execution time pipelined = $(K + n - 1)\tau$

Execution time unpipelined = $(K\tau) \times n$

Where, n = number of instructions

$\tau$ = time taken for each stage

K = number of stages in pipeline

**Ex. 2.6.2**

If a processor executes 100 instructions in a pipelined (5 stage) processor and unpipelined processor. What is the speedup achieved by pipelining technique if the time taken for each stage is 20 ns.

**Soln. :** n = 100 instruction, K = 5, $\tau$ = 20 ns

Execution time pipelined = $(5 + 100 - 1) \times \tau$

$= (5 + 99) \times 20 \text{ ns} = 2080 \text{ ns}.$

Execution time unpipelined = $(K \tau) n = 5 \times 20 \text{ ns} \times 100$

$= 10000 \text{ ns}.$

Speedup ratio is $= \frac{10000}{2080} = 4.80$ times.

**Syllabus Topic : Pipeline Hazards ; Data Dependences, Data Hazards, Branch Hazards**

## 2.7 Pipeline Hazards

**Q. Explain various pipeline hazards with their solutions. (5 Marks)**

Pipelining increases processor performance by increasing instruction throughput, because several instructions are overlapped in the pipeline, cycle time can be reduced, increasing the rate at which instructions execute.

Instruction Hazards (dependencies) occur when instructions read or write registers that are used by other instructions. The type of conflicts are divided into three categories :

(1) Structural hazards (resource conflicts)
(2) Data hazards (Data dependency conflicts)
(3) Branch difficulties (Control hazards)

➜ **(1) Structural hazards (Resource conflicts)**

- These hazards are caused by access to memory by two instructions at the same time. These conflicts can be slightly resolved by using separate instruction and data memories.
- Structural hazards occur when the processor's hardware is not capable of executing all the instructions in the pipeline simultaneously.
- Structural hazards within a single pipeline are rare on modern processors because the Instruction Set architecture is designed to support pipelining.

➜ **(2) Data hazards (Data dependency)**

- This hazard arises when an instruction depends on the result of a previous instruction, but this result is not yet available.
- These are divided into four categories :
  - (i) RAW – Hazard (Read after write Hazard)
  - (ii) RAR – Hazard (Read after read Hazard)
  - (iii) WAW – Hazard (Write after write Hazard)
  - (iv) WAR – Hazard (Write after read Hazard)

☞ **RAR Hazard**

RAR Hazard occurs when two instructions both read from the same register. This hazard does not cause problem for the processor because reading a register does not change the register's value. Therefore, two instructions that have RAR Hazard can execute on successive cycles.

**Example 1** : Instructions having RAR Hazard.

ADD $r_1, r_2, r_3$
SUB $r_4, r_5, r_3$ ← Both instructions read $r_3$, creating RAR

☞ **RAW Hazard**

This hazard occurs when an instruction reads a register that was written by a previous instruction. These are also called as **data dependencies** (or) **true dependencies**.

**Example 2** : Instructions having RAW - Hazard.

ADD $r_1, r_2, r_3$
Subtract reads the output of the addition creating — RAW hazard
SUB $r_4, r_5, r_1$

**WAR and WAW** are also called as name dependencies.

These hazards occur when the output register of an instruction has been either read or written by a previous instruction.

If the processor executes instructions in the order that they appear in the program and uses the same pipeline for all instructions, WAR and WAW hazards do not cause any problem in execution process.

**Example 3** : Instruction having WAR Hazard.

ADD $r_1, r_2, r_3$
SUB $r_2, r_5, r_6$ — WAR Hazard

**Example 4** : Instructions having WAW Hazard

ADD $r_1, r_2, r_3$
SUB $r_1, r_5, r_6$
WAW Hazard

➜ **(3) Branch Hazards**

- Branch instructions, particularly conditional branch instructions, create data dependencies between the branch instruction and the previous instruction, fetch stage of the pipeline.



- Since the branch instruction computes the address of the next instruction that the instruction fetch stage should fetch from, it consumes some time and also some time is required to flush the pipeline and fetch instructions from target location. This time wasted is called as branch penalty.

### 2.7.1 Methods to Resolve the Data Hazards and Advances in Pipelining

➜ **(MU - Dec. 2014, May 2015, May 2016, Dec. 2016, May 2017)**

**Q. Explain various pipeline hazards with their solutions. (5 Marks)**

**Q. What are the types of pipeline hazards ? Dec. 14. 5 Marks**

**Q. Pipeline hazards. May 15, May 16, Dec. 16, May 17. 7 Marks**

The methods used to resolve the data hazards are discussed in the following sub sections.

#### 2.7.1.1 Pipeline Stalls

- The hardware inserts a special instruction called (NOP) i.e. no operation instruction known as a bubble into the flow of execution stage of pipeline to resolve the RAW hazard between two instructions.
- This method is also called as hardware interlocks.
- This approach detects the hazard and maintains the program sequence by introducing delays to resolve the data hazards (RAW).

#### 2.7.1.2 Operand Forwarding (or) Bypassing

- This technique uses a special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline stages.
- Example of a RAW dependency exists between two instructions, instead of transferring an ALU result into a destination register, the hardware checks the destination operand, and if it is needed as a source in the next instructions, it passes the result directly into the ALU input; bypassing the register file.
- This method requires additional hardware paths through MUX (Multiplexers).
- In this case there is a multiplexer between the two stages, wherein the data required by the next instruction is forwarded.
- Let us see this with a program example.
- If we take the following sequence of instructions,

  ADD $r_1, r_2, r_3$
  SUB $r_4, r_1, r_3$

- We will notice that the destination of instruction 1 is the source for instruction 2. In this case the ALU stage or the execution stage of the pipeline will forward the data to the next instruction as shown in the Fig. 2.7.1. The Fig. 2.7.1 assumes that the system is 6 stage pipelined system.
- As shown in Fig. 2.7.1 the data i.e. the value of register r1 is passed from the first instruction to the second instruction. Actually the value of the register r1 is updated by the write operand stage of the first instruction. But, before that the same is required by the execute stage of second instruction. Hence the value of this register is passed to the second instruction.

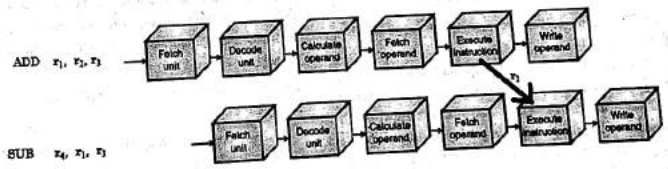



**Fig. 2.7.1 : Operand forwarding mechanism in pipelining for resolving a data hazard**

### 2.7.1.3 Dynamic Instruction Scheduling (or) Out-Of-Order (OOO) Execution

- This is another interesting and very widely used technique because of the speed up given by it. It is used in Pentium IV processor.
- Here the execution of the instructions of a program is done out-of-order i.e. not in the sequence as the instructions were written by the programmer. As and when the resources of an instruction are available, the execution of that instruction is done. If, for an instruction the resources are not available, it is kept in waiting state and the further instructions whose resources are available will be executed.
- But, you would think that this approach will have a problem. The logic implemented by the programmer will not be followed properly i.e. wrong sequence of instructions will be executed. The answer to this is that, although the instructions are executed out-of-order, but the write-back is done in order, and hence the final result of the program is in sequence.
- The compiler is designed in such a way that, while translating from high-level language to machine language program, it detects the data dependencies and re-orders the instructions.
- If necessary to delay the loading of the conflicting data it inserts no-operation instruction (NOP).

## 2.7.2 Handling of Branch Instructions to Resolve Control Hazards

The methods used to resolve the control hazards are discussed in the following sub sections.

### 2.7.2.1 Pre-Fetch Target Instruction

- One way of handling a conditional branch is to prefetch the target instruction in addition to the instruction following the branch. Both are saved until the branch is executed.
- If the branch condition is successful, the pipeline continues from the branch target instructions else sequential instructions are executed.

### 2.7.2.2 Branch Target Buffer (BTB)

- The BTB is an associative memory included in the fetch segment of the pipeline.
- Each entry in BTB consists of the address of a previously executed branch instructions and the target instruction for that branch. It also stores the next few instructions after the branch target instructions. When the pipeline decodes a branch instruction, it searches the associative memory BTB for the address of the instruction.
- If it is in the BTB, the instruction is available directly and prefetch continues from the new path.
- If the instruction is not in BTB, pipeline shifts to the instruction streams and stores the target instruction.

### 2.7.2.3 Loop Buffer

- This is like a BTB, but is a high speed register maintained by the instruction fetch segment of the pipeline. When a program loop is detected in a program, it is stored in the loop buffer.
- The program loop can be executed directly without having to access memory until loop mode is removed by final branching out.

### 2.7.2.4 Branch Prediction

- A pipeline with branch prediction uses additional logic to speculate the outcome of a conditional branch instruction before it is executed.
- The pipeline then begins pre-fetching the instruction stream from the predicted path.
- A correct prediction eliminates the wasted time caused by branch penalties.
- A detailed operation of branch prediction logic will be discussed in the later sections of this chapter.

**Syllabus Topic : Delayed Branches**

### 2.7.2.5 Pipeline Stall (Delayed Branch)

Complier detects branch instruction and rearranges the machine language code sequence by inserting useful instructions and rearranges the code sequence to reduce the delays incurred by Branch Instruction.

### 2.7.2.6 Loop Unrolling Technique

- This is a very superb solution to handle the stalls due to branching in loops.
- In this case a code which has a loop that has to be executed multiple times, will be actually stored multiple times (or unrolled) so as to remove the need of branching.
- Let us see how this can be implemented with an example.
- If there is a code for adding an array of 5 numbers, the loop can be written as shown in the code below (using processor 8086) :



| Label | Instructions |
|---|---|
| | MOV AX,0000H |
| | MOV CX,0005H |
| ACAIN : | ADD AX,[SI] |
| | INC SI |
| | DEC CX |
| | JNZ ACAIN |

- This loop can be unrolled to avoid stalling as shown below :

| Label | Instructions |
|---|---|
| | MOV AX,0000H |
| | ADD AX,[SI] |
| | INC SI |
| | ADD AX,[SI] |
| | INC SI |
| | ADD AX,[SI] |
| | INC SI |
| | ADD AX,[SI] |
| | INC SI |
| | ADD AX,[SI] |

- Thus you will notice that we have unrolled the loop, and written the loop for the number of times it was to be repeated. This totally removes the pipeline stalls due to the loops.
- The **advantage** clearly seen of this method is that there is no scope for pipeline stall and hence the performance will increase.
- The major **disadvantage** of this method is that the memory required will be more as the loop has to be unrolled and stored in memory. In our example the loop was to be repeated for only 5 times, but if the loop was larger and had to be repeated for say 100 or 1000 times, the memory consumed would be very huge.

### 2.7.2.7 Software Scheduling or Software Pipelining

- In case of software pipelining, the iterations of a loop of the source program are continuously initiated at regular intervals, before the earlier iterations complete. Thus taking advantage of the parallelism in data path.
- It can be said that software scheduling, schedules the operations within a loop, such that an iteration of the loop can be pipelined to yield optimal performance.
- The sequences of the instructions before steady state are called as **PROLOG,** while the ones after the steady state are called as **EPILOG.**
- Let us see this with an example. Suppose the source code is

```
for(i=0;i<=n-1;i++)
a[i]=a[i]+10;
```

- When this loop is executed by a processor, the processor will do the following:

```
for(i=0;i<=n-1;i++)
{
  Load a[i];
  Add a[i]+10;
  Store a[i];
}
```

- Here you will notice that the three instructions inside the loop (in each iteration) are the same i.e. each of the three instructions have to operate on the data a[i].
- When this is converted to pipeline, it will look as shown in the Fig. 2.7.2. But the three instructions, one below the other are dependent and hence cannot be pipelined. But the instructions that are circled can be pipelined.

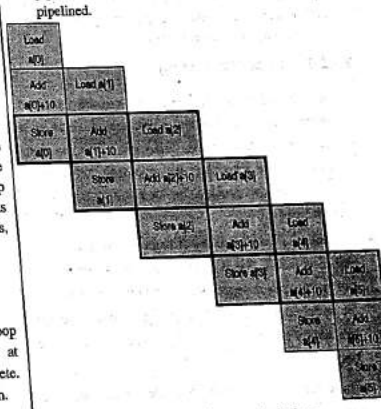



**Fig. 2.7.2 : Software pipelining**

- You will notice in the Fig. 2.7.2, that the th instructions that are circled are store, add and lo

These instructions are always independent i.e. they have different data to operate on.

- For example in the first circle: Store a[0], Add a[1]+10 and Load a[2] is performed. Each of these instructions is using different data.
- Thus the code can be changed to the following :

```
Load a[0]
Add a[0]+10
Load a[1]
for(i=2;i<n-3;i++)
{
  Store a[i-2];
  Add a[i-1]+10;
  Load a[i]
}
Store a[n-2];
Add a[n-1]+10;
Store a[n-1]
```

- Thus, you will notice inside the for loop i.e. for each iteration, each of the three instructions are working on different data and hence are not dependent on each other and hence allowing pipelining of the three instructions without any hazards.

### 2.7.2.8 Trace Scheduling

- In a general pipelining, the instructions are scheduled in sequence. This results in a problem or hazard on a branching instruction as discussed in the previous section. Trace scheduling is a good solution to avoid hazard due to branching. Let us see how this can be implemented.
- In this case the probability of branch to be taken or not taken is found. Based on this the code is written with all instructions in sequence, such that no branching will be required for most of the times according to the probability calculated earlier. This code is called as the trace.
- The other blocks of code are made for less probable cases i.e. if branching is taken. Hence this trace code and the other blocks of code are written, with minimizing branches. Let us see this with a program example. Suppose the source code is:

```
if(a[i]==0)
a[i]=a[i]+10;
else a[i]=a[i]+1;
x[i]=x[i]*x[i];
```

- The number is to be squared in the above code number is zero then 10 is to be squared there, while if it is any other number its value is to be squared and stored in the When this is converted to assembly program look as shown below :

| Label | Instruction |
|---|---|
| | Load a[i] into say AL |
| | Compare AL with 0 |
| | If not equal to zero then branch to label over |
| | Add AL with 10 |
| | Branch to label next |
| over: | Increment AL |
| next: | Multiply AL with itself |
| | Store the result in a[i] |

This can be divided into four blocks as shown in Fig. 2.7.3.

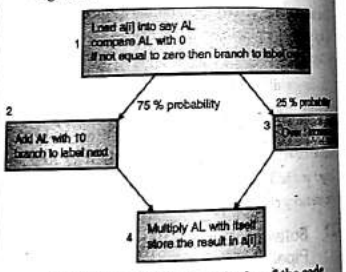



**Fig. 2.7.3 : Division of Blocks of the code**

- Since the path 1-2-4 is the most probable path, this make the trace. The path 1-3-4 will be a separate This is shown in the Fig. 2.7.4.
- Hence in most of the cases i.e. 75% cases the trace be executed and hence no branching will be Although in 25% cases we will need to Block 1, but there would be only one branching multiple branching as required in the previous



Trace

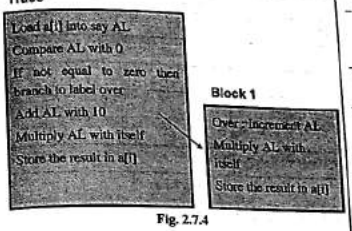



**Fig. 2.7.4**

### 2.7.2.9 Predicated Execution

- This is also a method that removes the branches. Here each instruction has a predicate that decides whether the instruction is to be executed or not. If the predicate is true then the instruction is executed, else it is not executed. The predicate is a condition bit. If the bit is '1' then the instruction is to be executed else it is not to be executed.
- Each instruction has the operands and a predicate. This removes the branching instructions and hence the stall of pipeline.
- An example of predicate instruction is given below,

  CMOVZ AX, BX, CX.
- This instruction copies the contents of register BX into register AX, if the predicate register CX is zero. Else the contents of BX are not copied into AX.
- Predication mainly implements the if-else statement and hence the branching required for if-else is removed. It can remove the branching required for all the instructions.
- Hence we can say that predication totally removes the need of handling the branches in a pipelined system. The only disadvantage of predication is that the instruction size increases.
- Predication is used in IA-64 processors of Intel, ARM processor.

### 2.7.2.10 Speculative Loading

- This is a process implemented in EPIC processors discussed in chapter 1. In this case the data is brought from the memory, well before it is needed.
- The compiler indicates the data that will be required in the later parts of the program and the corresponding data is brought and kept in the processor.
- This removes the latency of memory accesses required for the data to be brought from the memory.
- As the data required later is speculated and brought in advance it is called as speculative loading of data.

### 2.7.2.11 Register Tagging

- Register tagging is normally done by a unit called as Reservation Station (RS) in a processor.
- This reservation station is used in order to resolve the data or resource conflicts amongst the multiple instructions entering the processor.
- The operands are made to wait in the reservation station until their data dependencies are resolved.
- A tag is used to identify each reservation station, and the tag unit keeps on monitoring these reservation stations.
- This tag unit also monitors all the registers used currently or the reservation stations. This technique is called as register tagging.
- This mechanism allows to resolve the register conflicts and hence the resultant data hazards.
- The reservation stations can also be used as buffers between the various stages of pipeline in the processor. These stages can work simultaneously once the conflict is resolved.

**Syllabus Topic : Branch Prediction**

## 2.7.3 Branch Prediction

Q. Explain branch prediction. (5 Marks)

Branch prediction foretells the outcome of conditional branch instructions. Excellent branch handling techniques are essential for today's and for future microprocessors. Requirements of high performance branch handling:

- An early determination of the branch outcome (the so-called branch resolution),
- Buffering of the branch target address in a BTAC (Branch Target Address Cache),
- An excellent branch predictor (i.e. branch prediction technique) and speculative execution mechanism.

- Often another branch is predicted while a previous branch is still unresolved; so the processor must be able to pursue two or more speculation levels, and
- An efficient rerolling mechanism when a branch is mispredicted (minimizing the branch misprediction penalty).

### 2.7.3.1 Misprediction Penalty

The performance of branch prediction depends on the prediction accuracy and the cost of misprediction. Misprediction penalty depends on many organizational features:

- The pipeline length (favouring shorter pipelines over longer pipelines),
- The overall organization of the pipeline,
- The fact if miss speculated, instructions can be removed from internal buffers, or have to be executed and can only be removed in the retire stage,
- The number of speculative instructions in the instruction window or the reorder buffer. Typically only a limited number of instructions can be removed each cycle.
- Misprediction is expensive (11 or more cycles in the Pentium II).

### 2.7.3.2 Static Branch Prediction

- Static Branch Prediction predicts always the same direction for the same branch during the whole program execution.
- It comprises hardware-fixed prediction and compiler-directed prediction.
- Simple hardware-fixed direction mechanisms can be :
  (a) Predict always not taken
  (b) Predict always taken
  (c) Backward branch predict to be taken, forward branch predict not to be taken
- Sometimes a bit in the branch opcode allows the compiler to decide the prediction direction.

### 2.7.3.3 Branch-Target Buffer or Branch-Target Address Cache

The Branch Target Buffer (BTB) or Branch-Target Address Cache (BTAC) stores branch and jump addresses, their target addresses, and optionally prediction information. The BTB is accessed during the IF stage.

| Branch address | Target address | Predict |
|---|---|---|
| | | |
| | | |
| | | |
| | | |
| ... | ... | ... |
| | | |

Fig. 2.7.5

### 2.7.3.4 Dynamic Branch Prediction

The hardware influences the prediction while execution proceeds. Prediction is decided on the computation history of the program. During the start-up phase of the program execution, where a static branch prediction might be more effective, the history information is gathered and dynamic branch prediction gets more effective. In general, dynamic branch prediction gives better results than static branch prediction, but at the cost of increased hardware complexity.

### 2.7.3.5 One-bit Dynamic Branch Predictor

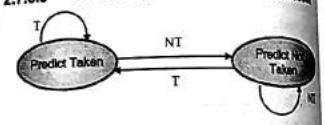



Fig. 2.7.6

- A one-bit predictor correctly predicts a branch at the end of loop iteration, as long as the loop does not exit.
- In nested loops, a one-bit prediction scheme will cause two misprediction for the inner loop :
- One at the end of the loop, when the iteration exits the loop instead of looping again, and one when executing the first loop iteration, when it predicts exit instead of looping.
- Such a double misprediction in nested loops is avoided by a two-bit predictor scheme.

### 2.7.3.6 Two-bit Prediction

A prediction must miss twice before it is changed when a two-bit prediction scheme is applied.



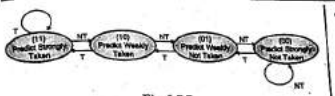



Fig. 2.7.7

**Syllabus Topic : Performance Measures of Computer Architecture CPI, Speedup, Efficiency, Throughput**

## 2.8 Performance Measures of Computer Architecture

**Q. Explain various performance metrics of CPU (10 Marks)**

There are various parameters that are used to measure the performance of a parallel system. We will see these parameters in this section.

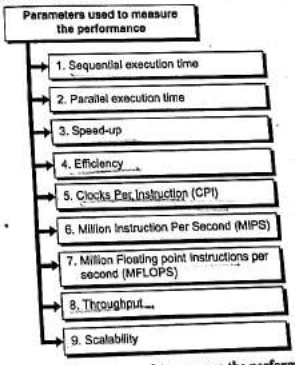



**Fig. 2.8.1 : Parameters used to measure the performance**

**1. Sequential execution time**

The time required for a program to be executed on a sequential (uni-processor) system is called as the sequential execution time with respect to parallel processors. It is represented by T(1).

**2. Parallel execution time**

The time required for a program to be executed on a n-parallel processor system is called as parallel execution time for 'n' processors. It is represented as T(n), where 'n' is the number of processors.

**3. Speed-up**

The speed increase because of the parallel system compared to the uni-processor system is called as the speed up. It is the ratio of the speed of parallel system to that of the sequential system. It can also be given as the ratio of time required to execute a program on sequential system to that of the parallel system (since time is inverse of frequency or speed). It is a very important metric to measure the performance of a parallel system. It is represented as S(n) and given as below :

$$S(n) = \frac{T(1)}{T(n)} \qquad ...(2.8.1)$$

**4. Efficiency**

Efficiency of a parallel system is the ratio of the actual speed-up obtained by a system to the ideal-speed-up that should be achieved according to the number of processors in the parallel system. The ideal time required to execute a program using 'n' processors should be T(1)/n i.e. the time required should be 1/n of the time required on a sequential or single processor system. Thus the efficiency (η) can be given as below :

$$\eta \text{ or } E(n) = \frac{\text{Actual speed-up}}{\text{Ideal speed-up}} = \frac{T(1)}{nT(n)}. \qquad ...(2.8.2)$$

Since actual speed-up will be given as 1/T(n) and the ideal speed-up will be given as n/T(1).

**5. Clocks Per Instruction (CPI)**

This is as the name says a measure of the clock pulses require per instruction. It is the ratio of the clock cycles required for a program to the number of instructions in the program. The time for one clock pulse is given as 'τ' and is the inverse of frequency (f). Let the number of instructions in the program be '$I_c$', thus the time required to execute a program (T) can be given as:

$$T = I_c \times CPI \times \tau \qquad ...(2.8.3)$$

This is because there are $I_c$ instructions and CPI is the clock cycles for one instruction. Thus $I_c \times CPI$ is the total number of clock pulses required to execute the program. The time for one clock pulse is τ, thus the total time required to execute the program will be as given in Equation (2.8.3).

**6. Million Instruction Per Second (MIPS)**

This is a very widely used performance measure. As the name says it is the count of instructions executed per second in millions. For example, if a system has 5

MIPS, it means it can execute 5 million instructions in a second.

Let us get an equation to find the value of MIPS. The time required to execute one instruction is CPI × τ. Thus the number of instructions executed in one second is :

$$\text{Instructions per second} = \frac{1}{CPI \times \tau} \quad ...(2.8.4)$$

Thus, the MIPS count of instruction can be given as:

$$MIPS = \frac{1}{CPI \times \tau \times 10^6} \quad ...(2.8.5)$$

We have simply divided the instructions per second in Equation (2.8.4) by $10^6$ i.e. 1 Million to get Million Instructions per second (MIPS). This equation can be written by replacing τ by 1/f. Also if 'C' is equal to total clock pulses to execute a program i.e. $C = CPI \times I_c$; then CPI can be given as $CPI = C/I_c$. With these replacements, the new expression for MIPS will be:

$$MIPS = \frac{f \times I_c}{C \times 10^6} \quad ...(2.8.6)$$

**→ 7. Million Floating point Instructions per second (MFLOPS)**

This is similar to the MIPS, only the difference being here floating point instructions are taken into account. The same equations will work for MFLOPS, if the instruction count and CPI are replaced according to floating point instructions.

**→ 8. Throughput**

The throughput of a system is defined as the number programs executed per unit time. This is represented as $W_s$, and is given as below :

$$W_s = \frac{\text{Number of programs}}{\text{Time in seconds}} \quad ...(2.8.7)$$

**→ 9. Scalability**

A parallel system is said to be scalable if the ... efficiency is obtained by increasing the ... processor. Since the efficiency is dependent ... number of processors, and it normally ... decreasing with the increase in the ... processors. In Equation (2.8.2), there is a term ... the number of processors in the denominator. ... in this value of 'n' should not reduce the efficiency ... do this the value T(n) should reduce in the ... proportion as that of the increase in 'n'. This will ... the system to be scalable.

In the following section we will see the scalable ... performance measure using various laws.

## 2.9 Principles of Scalable Performance

There are some laws that govern the performance of a parallel processing system. These laws will be studied in this section.

**Syllabus Topic : Amdahl's law**

### 2.9.1 Amdahl's Law

**Q. Explain Amdahl's Law.** (4)

This law is used for a fixed workload parallel system. There are systems that have fixed computational work. Hence as the number of processing elements or processors increase, the time required for execution must reduce.

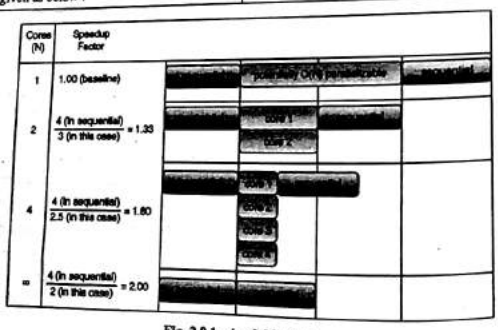



Fig. 2.9.1 : Amdahl's law



- Let the time requires for executing a task on a sequential system is $t_s$. If the 'f' fraction of the task is serial and (1 – f) is non-serial or parallel, then we can make the second part work in parallel on multiple processors but the first part has to work serially. For example if a part of code is serial, it has to always work serially on whatever number of processors be there.
- The part that is parallelizable will require lesser and lesser time when the number of processor increases. Thus the Speed-up also keeps on increasing. This can be explained with the Fig. 2.9.1. In this figure, four time units as shown are required to execute the program in case of a sequential system.
- When there are two processors, the sequential time remains the same, while the parallelizable time requires half the time i.e. one time unit, as there are two processors. When the number of processors are 4, the time required further halves to just a half time unit.

Thus the speed-up for n-processors according to the Amdahl's law can be given as below :

$$S(n) = \frac{t_s}{ft_s + (1-f)t_s/n} \quad ...(2.9.1)$$

where, $t_s$ is the total time required on sequential system

f is that fraction of the code which is sequential, hence 1-f is the parallelizable part of the task and 'n' is the number of processors.

- Equation (2.9.1) is called as Amdahl's law. The limitation of Amdahl's law is shown in Fig. 2.9.1, the last case. If we keep on increasing the number of processors to infinity, the speed-up cannot go beyond 1/f (2 in this case), as 'f' is the fraction of the code which is serial.
- The graph for the speed-up factor vs. number of processors 'n' for Amdahl's law is as shown in Fig. 2.9.2.

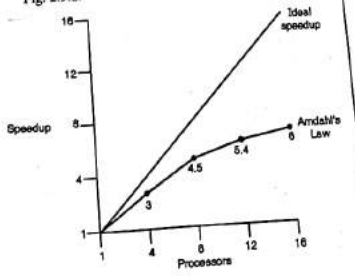



Fig. 2.9.2 : Speed-up vs number of processors for Amdahl's law

### 2.9.2 Gustafson's Law

- Gustafson law overcame the drawback of the Amdahl's law. Gustafson relaxed the problem size from being fixed to be of any size. Gustafson said that instead of having a fixed problem size or fixed workload we must assume that we have a fixed execution time. This is because in case of huge problem size, we will need to increase our system size i.e. number of processors instead of the execution time.
- The time for execution for whatever huge the workload is, the execution time must be fixed. According to this the speed-up factor will be numerically different as compared to the Amdahl's speed-up factor, and hence this is termed as scaled speed-up factor (S'(n)).
- Thus in case of Gustafson's law the execution time is fixed. Thus let the serial execution time be 's' and the parallel execution time be 'p' for 'n' processor system. Thus total execution time is s + p.
- If the execution is to be done on a sequential system, the execution time will be hence s + np. Thus the scaled speedup factor can be given as below :

$$S'(n) = \frac{s+np}{s+p} = \frac{s+np}{1} \quad ...(2.9.2)$$

assuming the time required on parallel system is 1 i.e. s + p = 1.

$$\text{Thus, } S'(n) = \frac{s+n(1-s)}{s+(1-s)}$$

(since, s + p = 1, therefore p = 1 – s)

$$= \frac{n+s(1-n)}{1}$$

$$S'(n) = n + s(1-n) \quad ...(2.9.3)$$

- This equation is called as Scaled speed-up factor of Gustafson's law.
- The graph of speedup factor vs. sequential part of the task can be shown as in Fig. 2.9.3. Fig. 2.9.3 shows that there is no effect of the sequential part on the speedup factor of a system.

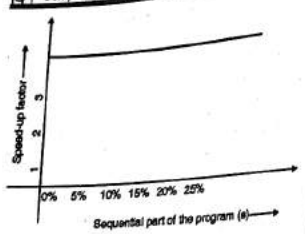



**Fig 2.9.3 : Speedup vs. Sequential part in the task**

**Syllabus Topic : ALU and Shifters**

## 2.10 Arithmetic Logic Unit and Shifters

- The various circuits used to execute arithmetic and logic operations are usually combined in a single circuit called an arithmetic logic unit or ALU.
- Simple ALUs performing fixed point addition and subtraction can be realized by combinational circuits. ALUs that also perform multiplication and division can be constructed using circuits for addition and subtraction.

### 2.10.1 Combinational ALUs

ALU performs some basic arithmetic operations on the numeric data stored in the registers. These basic operations may be.

1. Addition
2. Subtraction
3. Incrementing a number
4. Decrementing a number
5. Arithmetic shift operation

An **add** operation can be specified as :

$$R_3 \leftarrow R_1 + R_2$$

It implies: add the contents of registers $R_1$ and $R_2$ and store them in Register $R_3$.

The add operation mentioned above requires three registers along with the addition circuit in the ALU.

Subtraction in many machines, is implemented through 2's complement arithmetic operation as :

$$R_3 \leftarrow R_1 - R_2$$

$$\Rightarrow R_3 \leftarrow R_1 + 2\text{'s complement of } R_2$$

$$\Rightarrow R_3 \leftarrow R_1 + (1\text{'s complement of } R_2) + 1$$

$$\Rightarrow R_3 \leftarrow R_1 + R_2' + 1$$

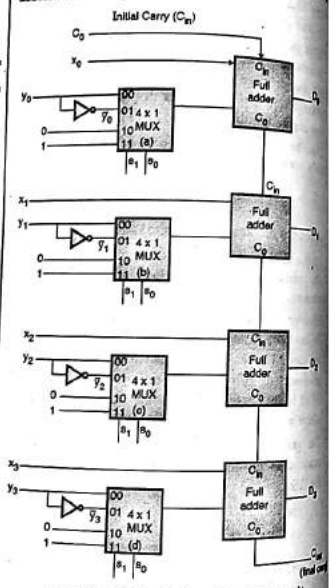
An increment operation can be written as :

$$R_1 \leftarrow R_1 \times 1$$

While a decrement operation can be written as :

$$R_1 \leftarrow R_1 - 1 \text{ [or, } R1 \leftarrow R1 + 2\text{'s complement}$$

An arithmetic circuit can be implemented using adders. Fig. 2.10.1 shows an implementation of 4 arithmetic circuit. The circuit is constructed by using 4 adders and 4 multiplexers.




(co 2.2)**Fig. 2.10.1 : 4-bit arithmetic circuit**

- Each multiplexer (MUX) of the given circuit has two select inputs $S_0$ and $S_1$. These selection lines have been shown separately for each MUX to simplify the circuit.
- The two selection lines to the MUX along with the carry ($C_{in}$) determine the type of operation to be performed by the circuit.
- This 4-bit circuit takes input from 2 4-bit registers with an initial carry ($C_{in}$) and outputs the four resultant bits ($D_0$, $D_1$, $D_2$, $D_3$) and a carry out bit.



- One of the inputs x ($x_0$, $x_1$, $x_2$, $x_3$) is fed directly to the full adder, while the second input y is fed through the multiplexer. The $S_0$, $S_1$ inputs to the multiplexer determines what type of input is selected ($y_i$, $\bar{y}_i$, 0, 1) for full adder. Let us see how $S_0$ and $S_1$ determine the output of multiplexer :

| $S_1$ | $S_0$ | Output of 4 × 1 MUX | | | | Comment |
|---|---|---|---|---|---|---|
| | | MUX (a) | MUX (b) | MUX (c) | MUX (d) | |
| 0 | 0 | $y_0$ | $y_1$ | $y_2$ | $y_3$ | Data word y is input to full adder. |
| 0 | 1 | $\bar{y}_0$ | $\bar{y}_1$ | $\bar{y}_2$ | $\bar{y}_3$ | 1's complement of y is input to full adder. |
| 1 | 0 | 0 | 0 | 0 | 0 | Data word (0, 0, 0, 0) is input to full adder. |
| 1 | 1 | 1 | 1 | 1 | 1 | Data word (1, 1, 1, 1) is input to full adder. |

Now let us see how by coupling $C_{in}$ input bit with the selection values we can obtain various operations.

| $S_1$ | $S_0$ | $C_{in}$ | Equivalent function | Equivalent operation | Name of the operation |
|---|---|---|---|---|---|
| 0 | 0 | 0 | x + y | $R \leftarrow R_1 + R_2$ | Add |
| 0 | 0 | 1 | x + y + 1 | $R \leftarrow R_1 + R_2 + 1$ | Add with carry |
| 0 | 1 | 0 | $x + \bar{y}$ | $R \leftarrow R_1 + \bar{R}_2$ | Subtract with borrow |
| 0 | 1 | 1 | $x + (\bar{y} + 1)$ | $R \leftarrow R_1 + 2$'s complement of $R_2$ | Subtract |
| 1 | 0 | 0 | x | $R \leftarrow R_1$ | Transfer |
| 1 | 0 | 1 | x + 1 | $R \leftarrow R_1 + 1$ | Increment |
| 1 | 1 | 0 | x + (1111) | $R \leftarrow R_1 + 2$'s complement of decimal 1 | Decrement |
| 1 | 1 | 1 | x | $R \leftarrow R_1$ | Transfer |

Some of the operations given above need no explanation but let us discuss subtract with borrow.

In subtract with borrow, the equivalent function is given as :

$$x + \bar{y}$$

or $(x - 1) + (\bar{y} + 1)$

or, x + (2's complement of y) – 1

This implies that we are taking a borrow out of x before subtraction of y.

☛ **Logic operations**

Logic operations are basically the binary operations which are performed on the string of bits stored in registers. For a logic operation each bit of a register is treated as a variable.

A logic operation :

$R_1 \leftarrow R_1$ AND $R_2$ specifies AND operation to be performed on the contents of $R_1$ and $R_2$ store the result in $R_1$. For example, if $R_1$ and $R_2$ are 8 bit registers and

$R_1$ contains 11010110 and

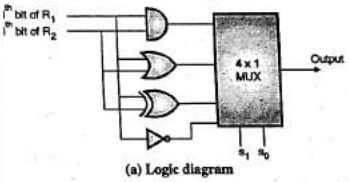
$R_2$ contains 01110011

then $R_1$ will contain 01010010 after AND operation.

Some of the common logic micro-operations are AND, OR, NOT or complement, Exclusive OR, NOR, NAND.

#### 2.10.1.1 Implementation of Logic Operations

- Fig. 2.10.2 shows one bit, that is $i^{th}$ bit stage of the four logic operations.
- The $i^{th}$ bits of Registers $R_1$ and $R_2$ are passed through the circuit. On the basis of selection inputs $S_0$ and $S_1$ the desired operation is obtained.
- For a logic operation on n-bit words, we must have n-stages of the circuit shown in Fig. 2.10.2.



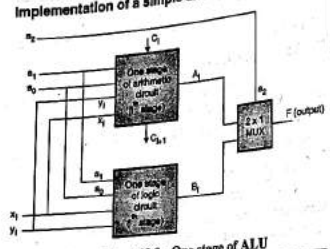

**(a) Logic diagram**

| $S_1$ | $S_0$ | Output | Operation |
|---|---|---|---|
| 0 | 0 | $R_1 \wedge R_2$ | AND |
| 0 | 1 | $R_1 \vee R_2$ | OR |
| 1 | 0 | $R_1 \oplus R_2$ | XOR |
| 1 | 1 | $\bar{R}_1$ | NOT |

**(b) Functional representation**

(co 2.3)**Fig. 2.10.2 : Logic diagram of one stage of logic circuit**

### Implementation of a simple arithmetic, logic unit :




(co 3.10) **Fig. 2.10.3 : One stage of ALU**

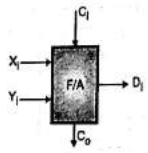
**Ex. 2.10.1**

Design an arithmetic circuit with one selection variable and two n-bit data inputs A and B. The circuit generates the following four arithmetic operations in conjunction with the input carry $c_{in}$. Draw the logic diagram for the first two stages.

| S | $C_{in} = 0$ | $C_{in} = 1$ |
|---|---|---|
| 0 | D = A + B | D = A + 1 |
| 1 | D = A − 1 | D = A + $\overline{B}$ + 1 |

**Soln. :**

Let us design the above arithmetic circuit with the help of full adders.



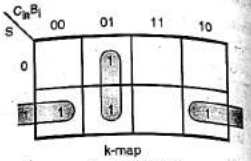

$X_i$ and $Y_i$ should be expressed using the Boolean functions to obtain the desired value of D.

**Table Ex. 2.10.1 : Function table for $X_i$, $Y_i$ and $D_i$**

| S | $C_{in}$ | $A_i$ | $B_i$ | $D_i$ | |
|---|---|---|---|---|---|
| 0 | 0 | $A_i$ | $B_i$ | $A_i + B_i$ | |
| 0 | 1 | $A_i$ | 0 | $A_i + 1$ | |
| 1 | 0 | $A_i$ | 1 | $A_i - 1$ | |
| 1 | 1 | $A_i$ | $\overline{B}_i$ | $A_i + \overline{B}_i + 1$ | [ A − 1 = A + 2's complement of 1] |
| | | | | | |

$\therefore X_i = A_i$

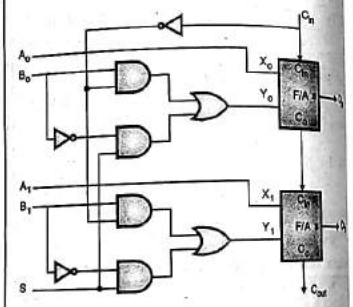
$Y_i = \overline{S} \cdot \overline{C}_{in} \cdot B_i + S \cdot \overline{C}_{in} + S \cdot C_{in} \cdot \overline{B}_i$




k-map

$\therefore Y_i = \overline{C}_{in} B_i + S\overline{B}_i$

$\therefore Y_i = \overline{C}_{in} B_i + S \cdot \overline{B}_i$




(co 2.71) **Fig. Ex. 2.10.1**

**Ex. 2.10.2**

An 8-bit CPU has the register R input to 2's complement ALU. The current value of R is hexadecimal $(82)_{16}$. For each of the following instructions determine the content of the status register having bits V, Z, S, C (V = overflow, Z = zero, S = sign C = carry) and interconnected to the ALU.

(i) ADD immediate operand $(B9)_{16}$ to R.

(ii) SUB immediate operand $(6E)_{16}$ from R.

**Soln. :**

**(i)**

$R = (82)_{16} = 1000\ 0010$

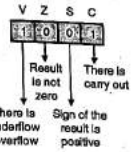
Operand = $(B9)_{16} = +\ 1011\ 1001$

$0011\ 1011$ (1 ↰)



Since, the sign of the result indicates that the sum of two negative numbers is a positive number, underflow has occurred.

Contents of status register =



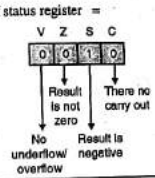

**(ii)** Since $(6E)_{16}$ is subtracted from R, we will find its 2's complement.

$(6E)_{16} = 01101110$

$\therefore$ 2's complement of $(6E)_{16} = 10010010$

$\therefore R - (6E)_{16} = \begin{vmatrix} R = & 1000\ 0010 \\ -(6E)_{16} = & 0110\ 1110 \\ & 1111\ 0000 \end{vmatrix}$

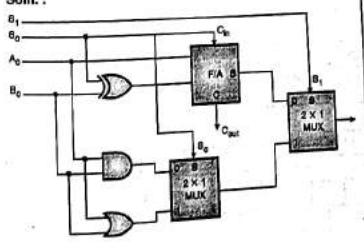
Contents of status register =




**Ex. 2.10.3**

Draw logic diagram of ALU that performs AND, OR logic operations and ADD, SUB arithmetic operation.

**Soln. :**

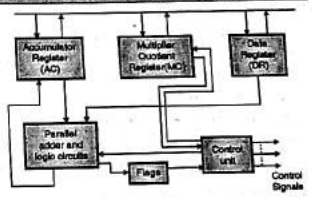



(co 2.72) **Fig. Ex. 2.10.3 : One stage of ALU**

| Control lines | | Output |
|---|---|---|
| $S_1$ | $S_0$ | |
| 0 | 0 | A + B |
| 0 | 1 | A − B |
| 1 | 0 | A ∧ B |
| 1 | 1 | A ∨ B |

## 2.10.2 Sequential ALU

**Q. Design a sequential ALU (5 Marks)**




(co 3.11) **Fig. 2.10.4 : Structure of a basic sequential ALU**

- Although, both multiplication and division can be implemented by combinational logic, it is very impractical. Combinational multiplier and dividers are costly in terms of hardware. Such circuits are much slower than addition and subtraction circuits. Fig. 2.10.4 shows a very common ALU design that aims at minimizing hardware cost.
- The Organization has three one word registers AC, MQ and DR which are used for data storage.
- In case of arithmetic (Add, Subtract) and logic operations, two inputs are in AC and DR registers, while output is AC register. AC and MQ are generally organized as a single ACMQ register.
- This register is capable of left or right shift operation. Some of the operations which can be defined on this unit are :

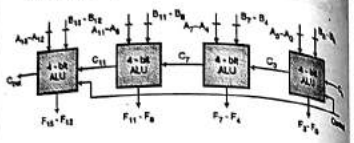
| | |
|---|---|
| Addition | AC = AC + DR |
| Subtraction | AC = AC − DR |
| Multiplication | AC · MQ = DR × MQ |
| Division | AC · MQ = MQ/DR |
| AND | AC = AC and DR |
| OR | AC = AC or DR |
| Exclusive-OR | AC = AC X or DR |
| NOT | AC = not (AC) |

- The MQ register stores the multiplier if multiplication is to be performed. Multiplication can be performed using add and shift (right shift) operations. MQ register stores the quotient if division is to be performed.
- The result of multiplication or division can finally be obtained in AC - MQ register combination.
- DR is another important register which is used for storing second operand. In fact it acts as a buffer register which stores the data brought from the memory for an instruction.

### 2.10.3 ALU Expansion

It is quite feasible to manufacture an entire sequential ALU for fixed point m-bit number on a single IC chip. Moreover, the ALU can easily be designed for expansion to handle operands of size n = km. It can be done in two ways :

1. Spatial expansion (using bit slice ALU) : Connect k-copies of the m-bit ALU in the manner of a ripple-carry adder to form a single ALU capable of processing km-bit words directly. The resulting array-like circuit is said to be bit sliced because each component ALU concurrently processes a separate slice of m-bits from each km bit operand.




(co 2.12) **Fig. 2.10.5 : A 16 bit ALU composed of four 4-bit ALUs linked by carry propagation**

2. Temporal expansion: Use one copy of the m-bit ALU chip in the manner of a serial adder to perform an operation on km-bit words in k-consecutive steps. In each step the ALU processes a separate m-bit slice of each operand. The processing is called multicycle processing.

## 2.11 Shift Registers and Shift Operations

- The binary data in a register can be moved within the register from one flip-flop to the other or outside it with application of clock pulses.
- The registers that allow such data transfers are called as **shift registers.**
- Shift registers are used for data storage, data transfer and certain arithmetic and logic operations.

**Modes of operation of a shift register :**

The various modes in which a shift register can operate (also called as register transfer operations) are as follows :

1. Serial input serial output.
2. Parallel in serial out.
3. Serial input parallel output.
4. Parallel in parallel out.

These modes are explained in brief in Table 2.11.1.

**Table 2.11.1 : Brief explanation of various modes of shift register**

| Sr. No. | Mode | Illustrative diagram | Comments |
|---|---|---|---|
| 1. | Serial input serial output (serial shift right) | I/P → FF3 FF2 FF1 FF0 → O/P | Data bits shift from left to right by 1 position per clock cycle. |
| 2. | Serial input serial output (serial shift left) | O/P ← FF3 FF2 FF1 FF0 ← I/P | Data bits shift from right to left by 1 position per clock. |
| 3. | Serial input parallel output | I/P → FF3 FF2 FF1 FF0; Outputs | All o/p bits are made available simultaneously after 4-clock pulses. |



| Sr. No. | Mode | Illustrative diagram | Comments |
|---|---|---|---|
| 4. | Parallel input serial output | Inputs; FF3 FF2 FF1 FF0 → O/P | All inputs are loaded simultaneously but output bit by bit. |

### 2.11.1 Serial Input Serial Output (Shift Left Mode)

- The serial input serial output type shift register with shift left mode is shown in Fig. 2.11.1.
- Let all the flip-flops be initially in the reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0 = 0$.
- We are going to illustrate the entry of a four bit binary number 1 1 1 1 into the register.
- When this is to be done, this number should be applied to "$D_{in}$" bit by bit with the MSB bit applied first.

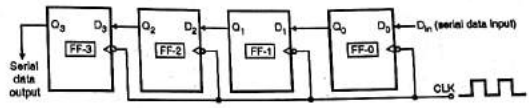



**Fig. 2.11.1 : Serial shift left register**

- The D input of FF-0 i.e. $D_0$ is connected to serial data input ($D_{in}$). Output of FF-0 i.e. $Q_0$ is connected to the input of the next flip-flop i.e. $D_1$ and so on.

**Operation :**

- Before application of clock signal let $Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 0\ 0$ and apply MSB bit of the number to be entered to $D_{in}$. So $D_{in} = D_0 = 1$.
- Apply the clock. On the first falling edge of clock, the FF-0 is set and the stored word in the register is

$Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 0\ 1$

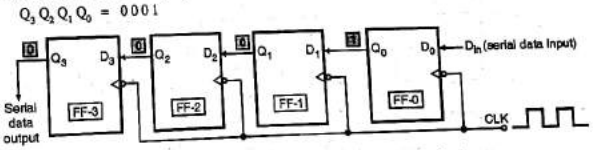



**Fig. 2.11.2 : Shift register status after first falling clock edge**

- Apply the next bit to $D_{in}$. So $D_{in} = 1$.
- As soon as the next negative edge of the clock hits, FF-1 will set and the stored word changes to,

$Q_3\ Q_2\ Q_1\ Q_0 = 0\ 0\ 1\ 1$

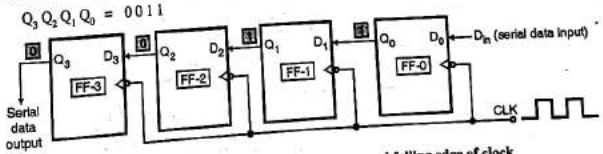



**Fig. 2.11.3 : Shift register status after the second falling edge of clock**

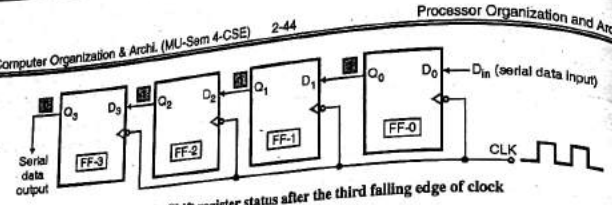



**Fig. 2.11.4 : Shift register status after the third falling edge of clock**

- Apply the next bit to be stored i.e. 1 to $D_{in}$.
- Apply the clock pulse. As soon as the third negative clock edge hits, FF-2 will be set and the output get modified to,

  $Q_3 Q_2 Q_1 Q_0 = 0111$
- Similarly with $D_{in} = 1$, and with the fourth negative clock edge arriving, the stored word in the register is

  $Q_3 Q_2 Q_1 Q_0 = 1111$

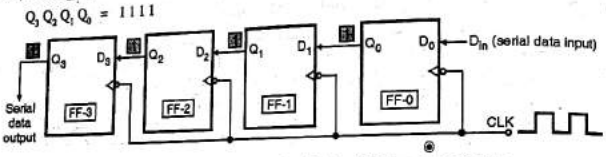


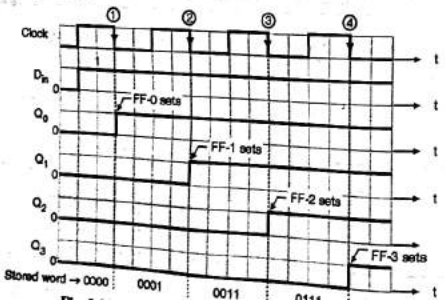

**Fig. 2.11.5 : Shift register status after the fourth falling edge of clock**

**Table 2.11.2 : Summary of shift left operation**

| | CLK | $Q_3$ | $Q_2 = D_3$ | $Q_1 = D_2$ | $Q_0 = D_1$ | Serial input $D_{in} = D_0$ |
|---|---|---|---|---|---|---|
| Initially | | 0 | 0 | 0 | 0 | |
| 1st | ↓ | 0 | 0 | 0 | 1 | 1 |
| 2nd | ↓ | 0 | 0 | 1 | 1 | 1 |
| 3rd | ↓ | 0 | 1 | 1 | 1 | 1 |
| 4th | ↓ | 1 | 1 | 1 | 1 | 1 |

Direction of data travel ←

**Waveforms for shift left operation :**






The waveforms for the shift left operation are shown in Fig. 2.11.6.

**Important Note :** Using the SISO mode, we needed 4 clock pulses to store a 4-bit word. So in general we can conclude that it requires n number of clock pulses to store an n bit word using SISO mode.

### 2.11.2 Serial In Serial Out (Shift Right Mode)

- The serial input serial output type shift register with shift right mode is shown in Fig. 2.11.7.
- Let all the flip-flops be initially in the reset condition i.e. $Q_3 = Q_2 = Q_1 = Q_0 = 0$.
- We are going to illustrate the entry of a four bit binary number 1 1 1 1 into the register.
- When this is to be done, this number should be applied to "$D_{in}$" bit by bit with the LSB bit applied first.

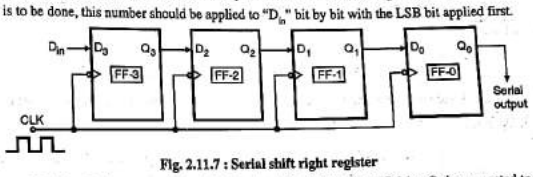



**Fig. 2.11.7 : Serial shift right register**

- The D input of FF-3 i.e. $D_3$ is connected to serial data input ($D_{in}$). Output of FF-3 i.e. $Q_3$ is connected to the input of the next flip-flop i.e. $D_2$ and so on.

**Operation :**

- Before application of clock signal let $Q_3 Q_2 Q_1 Q_0 = 0\ 0\ 0\ 0$ and apply LSB bit of the number to be entered to $D_{in}$. So $D_{in} = D_3 = 1$.
- Apply the clock. On the first falling edge of clock, the FF-3 is set, and the stored word in the register is

  $Q_3 Q_2 Q_1 Q_0 = 1000$

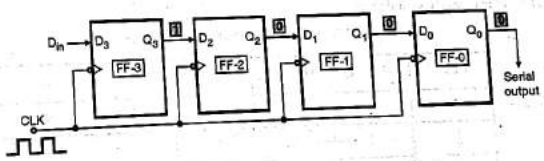



**Fig. 2.11.8 : Shift register status after first falling clock edge**

- Apply the next bit to $D_{in}$. So $D_{in} = 1$.
- As soon as the next negative edge of the clock hits, FF-2 will set and the stored word changes to,

  $Q_3 Q_2 Q_1 Q_0 = 1100$

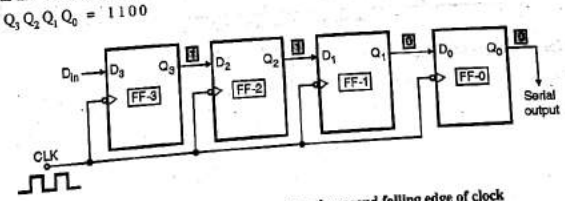



**Fig. 2.11.9 : Shift register status after the second falling edge of clock**

- Apply the next bit to be stored i.e. 1 to $D_{in}$.
- Apply the clock pulse. As soon as the third negative clock edge hits, FF-1 will be set and the output get modified to,

$Q_3\ Q_2\ Q_1\ Q_0 = 1\ 1\ 1\ 0$

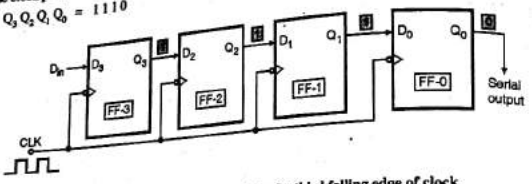



**Fig. 2.11.10 : Shift register status after the third falling edge of clock**

- Similarly with $D_{in} = 1$ and with the fourth negative clock edge arriving, the stored word in the register is

$Q_3\ Q_2\ Q_1\ Q_0 = 1\ 1\ 1\ 1$

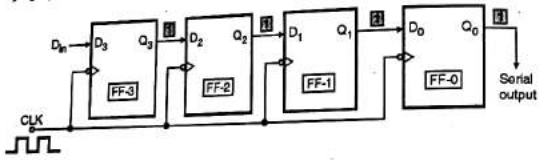



**Fig. 2.11.11 : Shift register status after the fourth falling edge of clock**

Table 2.11.3 summarizes the shift right operation.

**Table 2.11.3 : Summary of shift right operation**

| | CLK | $D_{in} = Q_3$ | $Q_3 = D_2$ | $Q_2 = D_1$ | $Q_1 = D_0$ | $Q_0$ |
|---|---|---|---|---|---|---|
| Initially | · | | 0 | 0 | 0 | 0 |
| 1st | ↓ | 1 | 1 | 0 | 0 | 0 |
| 2nd | ↓ | 1 | 1 | 1 | 0 | 0 |
| 3rd | ↓ | 1 | 1 | 1 | 1 | 0 |
| 4th | ↓ | 1 | 1 | 1 | 1 | 1 |

Direction of data travel

**Waveforms for shift right operation :**

The waveforms for shift right operation are as shown in Fig. 2.11.12.



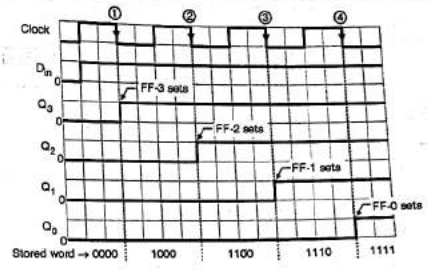



**Fig. 2.11.12 : Waveforms for the shift right operation**

**Important note :** Using the SISO mode, we needed 4 clock pulses to store a 4-bit word. So in general we can conclude that it requires n number of clock pulses to store an n bit word using SISO mode.

### 2.11.3 Applications of Serial Operation

- The transmission of data from one place to the other takes place in serial manner as shown in Fig. 2.11.13.
- It takes a longer time for serial transmission, because the time required to transmit one bit is equal to the time corresponding to one clock cycle.
- However for long distance communication where the distances are in kilometres, serial communication has an advantage that only one conductor is required to be used.


**Fig. 2.11.13 : Application of serial operation**

## 2.12 Serial In Parallel Out (SIPO)

- In this operations the data is entered serially and taken out in parallel.
- That means first the data is loaded bit by bit. The outputs are disabled as long as the loading is taking place.
- As soon as the loading is complete, and all the flip-flops contain their required data, the outputs are enabled so that all the loaded data is made available over all the output lines simultaneously.
- Number of clock cycles required to load a four bit word is 4. Hence the speed of operation of SIPO mode is same as that of SISO mode.

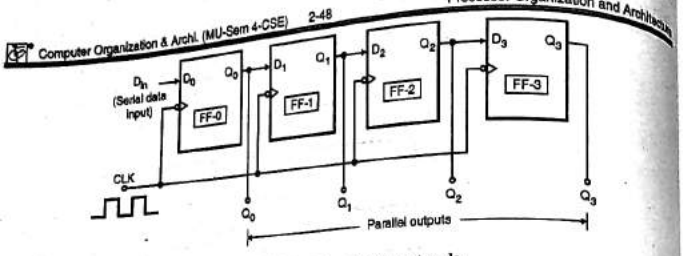



**Fig. 2.12.1 : Serial input parallel output mode**

## 2.13 Parallel In Serial Out Mode (PISO)

- In this mode, the bits are entered in parallel i.e. simultaneously as shown in Fig. 2.13.1.
- The circuit shown in Fig. 2.13.1 is a four bit parallel input serial output register.
- Output of previous FF is connected to the input of the next one via a combinational circuit.
- The binary input word $B_0$, $B_1$, $B_2$, $B_3$ is applied through the same combinational circuit.
- There are two modes in which this circuit can work namely shift mode or load mode.

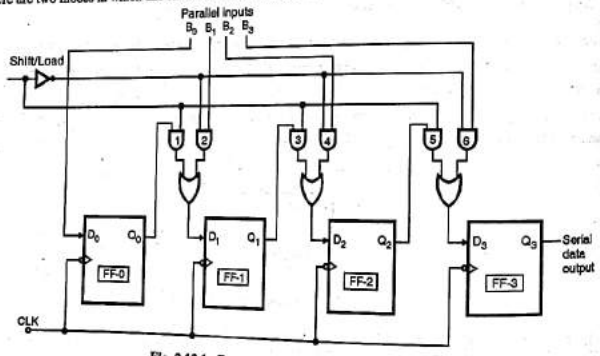



**Fig. 2.13.1 : Parallel in serial out shift register**

**Load mode :**

- When the shift / $\overline{\text{load}}$ line is low (0), the AND gates 2, 4 and 6 become active. They will pass $B_1$, $B_2$ and $B_3$ bits to the corresponding flip-flops.
- On the low going edge of clock, the binary inputs $B_0$ $B_1$ $B_2$ $B_3$ will get loaded into the corresponding flip-flops. Thus parallel loading takes place.

**Shift mode :**

- When the shift / $\overline{\text{load}}$ line is high (1), the AND gates 2, 4, 6 become inactive. Hence the parallel loading of the data becomes impossible.
- But the AND gates 1, 3 and 5 become active. Therefore the shifting of data from left to right bit by bit on application of clock pulses.
- Thus the parallel in serial out operation takes place.



## 2.14 Parallel In Parallel Out (PIPO)

- Fig. 2.14.1 demonstrates the parallel in parallel out mode of operation.
- The 4 bit binary input $B_0$, $B_1$, $B_2$, $B_3$ is applied to the data inputs $D_0$, $D_1$, $D_2$ and $D_3$ respectively of the four flip-flops.
- As soon as a negative clock edge is applied, the input binary bits will be loaded into the flip-flops simultaneously.
- The loaded bits will appear simultaneously to the output side. Only one clock pulse is essential to load all the bits.

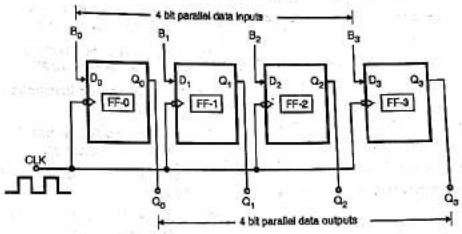



**Fig. 2.14.1 : Parallel in parallel out shift register**

## 2.15 Universal Shift Register

- A shift register which can shift the data in only one direction is called as a unidirectional shift register.
- A shift register which can shift the data in both the directions is called as a bi-directional shift register.
- Applying the same logic, a shift register which can shift the data in both the directions (shift right or left) as well as load it parallely, then it is called as a **universal shift register.**
- Fig. 2.15.1 shows the logic diagram of a universal shift register.

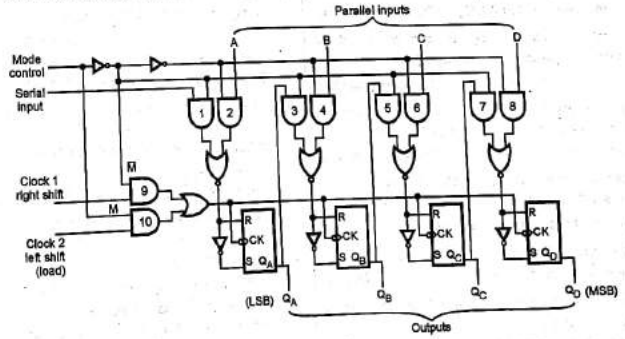



**Fig. 2.15.1 : Logic diagram of a universal shift register**

- This shift register is capable of performing the following operations :
  1. Parallel loading (parallel input parallel output)
  2. Left shifting
  3. Right shifting
- The Mode control input is connected to Logic 1 for parallel loading operation whereas it is connected to 0 for serial shifting.
- With mode control pin connected to ground, the universal shift register acts as a bi-directional register.
- For serial left operation, the input is applied to the serial input which goes to AND gate-1 in Fig. 2.15.1.
- Whereas for the shift right operation, the serial input is applied to D input (input of AND gate 8).
- The well known example of universal shift register in the IC form is IC7495.

## 2.16 Exam Pack (University and Review Questions)

### Syllabus Topic : Von Neumann Model

**Q.** Explain von Neumann's system. *(Ans. : Refer section 2.1.1)* **(5 Marks)**

**Q.** What is stored program concept? *(Ans. : Refer section 2.1.1(5))* **(May 2014, 3 Marks)**

**Q.** Define stored program concept and draw Von Neumann's Architecture. *(Ans. : Refer section 2.1.1)* **(Dec. 2014, 5 Marks)**

**Q.** What is stored program concept in digital computer ? *(Ans. : Refer section 2.1.1)* **(May 2015, 3 Marks)**

**Q.** Explain role of different registers like IR, PC, SP, AC, MAR and MDR used in Von Neumann model. *(Ans. : Refer section 2.1)* **(Dec. 2015, 5 Marks)**

**Q.** Explain Von Neumann architecture in detail. *(Ans. : Refer section 2.1)* **(Dec. 2016, 5 Marks)**

### Syllabus Topic : Instruction Formats

**Q.** Write a short note on instruction formats. *(Ans. : Refer section 2.2.1)* **(5 Marks)**

**Q.** Explain single address, two address and three address instructions. *(Ans. : Refer section 2.2.2)* **(5 Marks)**

**Q.** Explain polish notation *(Ans. : Refer section 2.2.3)* **(5 Marks)**

### Syllabus Topic : Basic Instruction Cycle

**Q.** Explain basic instruction cycle. *(Ans. : Refer section 2.2.4)* **(5 Marks)**

**Q.** Explain the instruction cycle with interrupts. *(Ans. : Refer section 2.2.5)* **(5 Marks)**

### Syllabus Topic : Addressing Modes

**Q.** Explain addressing modes of a processor. *(Ans. : Refer section 2.3)* **(10 Marks)**

**Q.** Explain in detail different types of addressing modes. *(Ans. : Refer section 2.3)* **(Dec. 2014, 10 Marks)**

### Syllabus Topic : Instruction Interpretation and Sequencing

**Q.** Compare pipelined vs non-pipelined system. *(Ans. : Refer section 2.5.1)* **(5 Marks)**

### Syllabus Topic : Basic pipelined datapath and control

**Q.** Write a short note on six stage pipelined system. *(Ans. : Refer section 2.5.1)* **(5 Marks)**

**Q.** What is instruction pipelining ? *(Ans. : Refer section 2.5.2)* **(May 2014, 6 Marks)**

**Q.** Explain six stage instruction pipelines with suitable diagram. *(Ans. : Refer section 2.5.2)* **(Dec. 2014, 10 Marks)**

**Q.** What is instruction pipelining ? What are advantages of pipelining ? *(Ans. : Refer section 2.5.2)* **(May 2015, 6 Marks)**

**Q.** Explain six stage instruction pipeline with suitable diagram. *(Ans. : Refer section 2.5.2)* **(Dec. 2015, 10 Marks)**

### Syllabus Topic :Pipeline Hazards : Data Dependences, Data Hazards, Branch Hazards

**Q.** Explain various pipeline hazards with their solutions. *(Ans. : Refer sections 2.7 and 2.7.1)* **(5 Marks)**

### Syllabus Topic : Branch Prediction

**Q.** Explain branch prediction . *(Ans. : Refer sections 2.7.2.4 and 2.7.3)* **(10 Marks)**

**Q.** What are the types of pipeline hazards ? *(Ans. : Refer section 2.7)* **(Dec. 2014, 5 Marks)**

**Q.** Pipeline Hazards. *(Ans. : Refer section 2.7)* **(May 2015, 7 Marks)**

**Q.** Explain various pipeline hazards. *(Ans. : Refer section 2.7)* **(May 2016, 5 Marks)**

**Q.** Explain various pipeline hazards with example. *(Ans. : Refer section 2.7)* **(Dec. 2016, 5 Marks)**



**Q.** Explain different pipelining hazards. *(Ans. : Refer section 2.7)* **(May 2017, 10 Marks)**

### Syllabus Topic : Performance Measures of Computer Architecture CPI, Speedup, Efficiency, Throughput

**Q.** Explain various performance metrics of CPU. *(Ans. : Refer section 2.8)* **(10 Marks)**

### Syllabus Topic : Amdahl's law

**Q.** Explain Admahl's Law. *(Ans. : Refer section 2.9.1)* **(10 Marks)**

### Syllabus Topic : ALU and Shifters

**Q.** Design a sequential ALU. *(Ans. : Refer section 2.10.2)* **(5 Marks)**

□□□

CHAPTER 3

Module III

# Control Unit Design

**Syllabus**

Hardwired control unit design methods : State table, delay element, sequence counter with examples like control unit for multiplication and division, Microprogrammed control Unit: Microinstruction sequencing and execution. Micro operations, Wilkie's microprogrammed Control Unit, Examples on microprograms.

## 3.1 CPU Architecture and Register Organization

→ (MU - May 2016)

**Q.** Describe the register organization within the CPU. **May 16, 10 Marks**

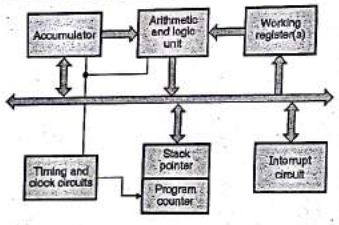



***Fig. 3.1.1 : General architecture of a microprocessor***

Fig 3.1.1 shows the architecture of microprocessor. This architecture is divided in different groups as follows :

1. Registers
2. Arithmetic and logic unit
3. Interrupt control
4. Timing and control circuitry

- It consists of PIPO (Parallel in parallel out) register as shown in Fig. 3.1.2.
- This section is also called as scratch pad memory. It stores data and address of memory.
- The register organization affects the length of program, the execution time of program and simplification of the program. To achieve better performance, the number of registers should be large.
- The architecture of microcomputer depends upon the number and type of the registers used in microprocessor. It consists 8-bit registers or 16 bit registers.
- The register section varies from microprocessor to microprocessor.
- The registers are used to store the data and address.
- These registers are classified as :
  - Temporary registers
  - General purpose registers
  - Special purpose registers.



### 3.1.1 Register Section

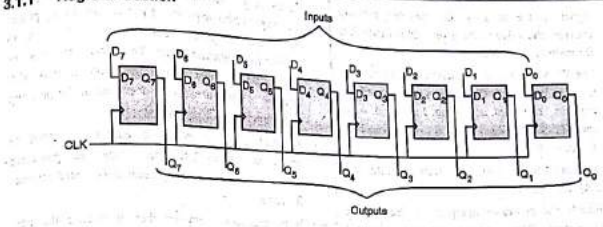



**Fig. 3.1.2 : 8 bit register**

### 3.1.2 Arithmetic and Logical Unit

- This section processes data i.e. it performs arithmetic and logical operations.
- It performs arithmetic operations like addition, subtraction and logical operations like ANDing, ORing, EX-ORing, etc.
- The ALU is not available to the user. Its word length depends upon the width of an internal data bus.
- The ALU is controlled by timing and control circuits.
- It accepts operands from memory or register. It stores result of arithmetic and logic operations in register or memory.
- It provides status of result to the flag register. Flag register shows status of result.
- ALU looks after the branching decisions.

### 3.1.3 Interrupt Control

This block accepts different interrupt request inputs. When a valid interrupt request is present it informs control logic to take action in response to each signal.

### 3.1.4 Timing and Control Unit

- This is a control section of microprocessor made up of synchronous sequential logic circuit.
- It controls all internal and external circuits.
- It operates with reference to clock signal.
- This accepts information from instruction decoder and generates microsteps to perform it. In addition to this, the block accepts clock inputs, performs sequencing and synchronising operations. The synchronization is required for communication between microprocessor and peripheral devices. To implement this it uses different status and control signals.
- The basic operation of a microprocessor is regulated by this unit.
- It synchronizes all the data transfers.
- This unit takes appropriate actions in response to external control signals.

## 3.2 Basic Instruction Cycle

- The instruction cycle is a representation of the states that the computer or the microprocessor performs when executing an instruction.
- The instruction cycle comprises of two main steps to be followed to execute the instruction, namely the fetch operation in the fetch cycle and the execution operation during the execute cycle.

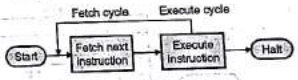



**Fig. 3.2.1 : Basic instruction cycle**

- Fig. 3.2.1 shows the basic instruction cycle. It comprises of the fetch and executes cycle in a loop to execute huge number of instructions, until it reaches the halt instruction.
- The fetch cycle comprises of the following operations :
  1. Program Counter (PC) holds address of next instruction to fetch; hence the CPU (Processor) fetches instruction from memory location pointed to by PC. This is done by providing the value of the PC to the MAR and giving the Read control

signal to the memory. On this the memory provides the value in the given address (which is the instruction) to MBR.

2. The PC value has to be incremented to point to the next instruction (Sometimes the value of PC may have be completely changed in case of some special instructions called as branching instructions).
3. The instruction is loaded into Instruction Register (IR) from the MBR.
4. Finally the processor interprets or decodes the instruction. The processor performs required operations in the execute cycle.

- In the execute cycle the operation asked to be performed by the instruction is done. It may comprise of one or more of the following operations :
  1. Transfer of data between processor and memory or between processor and IO module.
  2. Processing of data like some arithmetic or logical operations on data.
  3. Change of the sequence of operation i.e. branching instructions.

### 3.2.1 Interrupt Cycle

- Fetch and execute are not the only two states in the instruction cycle. There is one more state i.e. Interrupt cycle.
- In this subsection we will see the concept of interrupt in short and the interrupt cycle.
- Interrupt is a mechanism by which I/O modules can interrupt normal sequence of processing. Interrupt can be because of some request from an I/O device to service that particular device. This service may take or gives data or some control operation. It may also because of some unexpected operation in the program execution by the CPU itself.
- Interrupt cycle as discussed earlier is added to instruction cycle. During this cycle the processor checks for interrupt, and if present and enabled services the same.
- If no interrupt is present then it fetches the next instruction else if interrupt pending then it performs the following operations :
  1. Suspend the execution of current program.
  2. Save the context of the current program under execution.
  3. Set the PC value to start address of interrupt handler routine also called as interrupt service routine. Interrupt service routine is a small program which when executed, services the interrupting source.
  4. Process the interrupt service routine (ISR) and then.
  5. Restore the context and continue execution of the interrupted program.
- Thus the complete basic instruction cycle with Interrupts can be as shown in the Fig. 3.2.2.

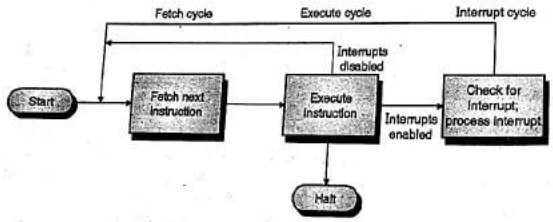



**Fig. 3.2.2 : Complete basic instruction cycle**

- You will notice in Fig. 3.2.2, the interrupts are checked for, after the execute cycle and processed if enabled and exist; else, it fetches the next instruction.
- The detailed instruction cycle is shown in Fig. 3.2.3.



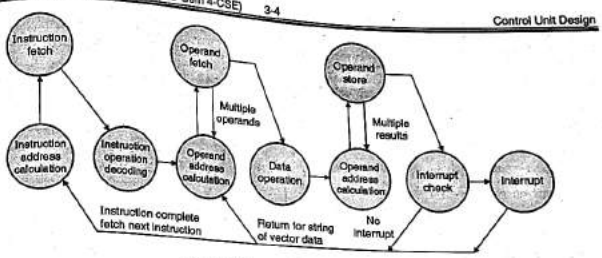



**Fig. 3.2.3 : Detailed instruction cycle**

- In Fig. 3.2.3, there are some states drawn on the upper side, while some on the lower side. The ones on the upper side are the operations carried out on the buses or are external operations, while the ones at the lower level are the operations carried out inside the CPU or are internal operations.
- The instruction cycle begins from the "Instruction address calculation" state, wherein the address of the next instruction is calculated or the value of the PC is updated. Then the instruction is fetched, which requires the operation on the buses.
- The instruction fetched is then decoded. Until this state, it is the fetch cycle.
- In the execute cycle, the operand address is calculated and the operands are fetched from the calculated address. Again to fetch the operands, we require the buses. After fetching the operand, if more operands are required for multiple operand instructions, then the next state is again calculate the operand address i.e. the address of the next operand. Once all the operands are fetched, the data operation is carried out as per the operation indicated in the instruction.
- Now for the result storage again the address of operand is calculated and the result is stored in the specified location of the memory. In case of multiple operands again the calculation and storage process for the operand continues until all the operands are stored.
- Now begins the interrupt cycle, wherein the first step is to check the presence of an enabled interrupt. If there is none, then the next state as seen in the Fig. 3.2.3 is the calculation of next instruction address i.e. executes the next sequential instruction.
- But in case the interrupt is present and enabled then the servicing of the same is done as discussed earlier in this section.
- In the Fig. 3.2.3, you will also notice that there are two paths from the end of the previous instruction. The one that goes to the state "Instruction address calculation" for the next instruction; and the one that goes to the "Operand address calculation" for vector instructions.
- Vector instructions are those instructions wherein the operation is same but the data on which the operation is to be performed in a huge block of data or an array of data. Hence in the second case, the instruction is already fetched and decoded i.e. the operation is already know, and the operation is to be performed on a block of data.
- After completing the operation on one set of operands, the CPU returns to the next operand address calculation state, wherein it calculates and fetches the next operand. Then it performs the operation, stores the result and again for the next set of operand, until all the operands in the array are completed.

**Syllabus Topic : Micro programmed control Unit : Microinstruction sequencing and execution, Micro Operations**

## 3.3 Instruction, Micro-Instructions and Micro-operations: Interpretation and Sequencing

→ **(MU - May 2015, May 2016, Dec 2016)**

**Q.** Explain Microinstruction sequencing and execution. **May 15, 7 Marks**

Q. Microinstructions to execute an instruction MOV (R1), R2. May 16, 6 Marks

Q. Explain micro instruction sequencing and execution. Dec 16, 10 Marks

- The structure of the CPU seen in section 3.2 is shown in details in Fig. 3.3.1. This structure has a speciality that all the control signals are shown in it.
- Programs are executed as a sequence of instructions. As seen in the previous sections of this chapter, each instruction consists of a series of steps that make up the instruction cycle i.e. fetch, decode, etc. Each of these steps is, in turn, made up of a smaller series of steps called micro-operations or micro-instructions.
- Control signals are issued to perform these micro-operations and micro-instructions are these control signals.
- Fig. 3.3.1 shows the structure of the CPU with these micro-instructions or the control signals.
- It also shows those registers as already seen in section 3.2 like PC, MAR, MBR, etc.
- There are some registers like the register 'Y' to provide one of the operand to the ALU as shown in the Fig. 3.3.1.
- Another register is the 'Z' register, which is used to store the result given by the ALU.
- A "temp" register or the temporary register to store some temporary data.
- The set of registers *R0* to *Rn* (the value of 'n' depends on the registers in the CPU) for general purpose operations.
- There is also an instruction decoder for decoding the instructions stored in the instruction register and in turn provides the micro-instructions or the control signals for the resources inside and outside the CPU.
- *The ALU also gets the control signals from this decoder indicating the operation to be performed like Add, Sub, and AND etc.*
- *The ALU also has an extra input called as $C_{in}$ i.e. the carry input as required for adder.*

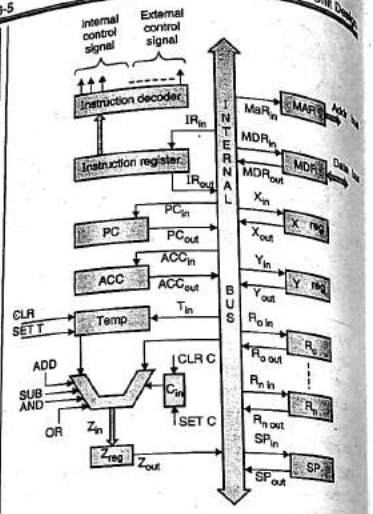



**Fig. 3.3.1 : Data path structure with control signals**

- To execute any instruction as seen earlier it is to be divided into three cycles viz. fetch, execute and interrupt cycles. The execute cycle will differ based on the operation to be carried out in the instruction, but the fetch and interrupt cycle will be common for all the cycles.
- Let us see the micro-instructions to be given for each of these cycles.

### 3.3.1 Fetch Cycle

Q. Explain fetch cycle with the corresponding micro-program. 5 Marks

- Fetch cycle is concerned to fetch (i.e. read from memory) the instruction. It involves following operations in different t-states (t-state is a time unit and is equal to one clock pulse) and hence the mentioned microinstructions in Table 3.3.1.

**Table 3.3.1 : Microinstructions for the fetch cycle**

| | Operation | Microinstructions |
|---|---|---|
| T1 | PC → MAR | $PC_{out}$, $MAR_{in}$, Read, Clear $Y_{in}$, Set $C_{in}$, Add, $Z_{in}$ |
| T2 | M → MBR<br>PC ← PC + 1 | $Z_{out}$, $PC_{in}$, Wait for memory fetch cycle |
| T3 | MBR → IR | $MBR_{out}$, $IR_{in}$ |



- As seen in the table, three clock pulses or t-states are required for the fetch cycle. Note, the control unit is an organizational part of the CPU; hence the design can vary from processor to processor.
- In the first t-state, the address of the instruction to be executed is given to the MAR register from the PC register. To perform this operation the control signals given are $PC_{out}$ and $MAR_{in}$. This will make the PC register give out its data and the MAR register accept this data. Also the memory is indicated to perform a read operation from memory hence the signal "Read".
- To increment the value of PC, the various operations are performed on ALU signals i.e. Clear Y, Set $C_{in}$, Add, $Z_{in}$. The 'Y' register is cleared and the carry flag is set. Now when the ALU is said to perform the "ADD" operation it will add the contents of the 'Y' register, carry flag and the contents of the internal data bus.
- The contents of the internal data bus are nothing but the value given out by the PC register. Hence the PC is added with '1' i.e. the carry flag and hence incremented value of PC is given to the 'Z' register.
- In the second clock pulse the CPU has to wait for the memory operation, but in the same time it can transfer the result in 'Z' register to the PC register with the control signals namely $Z_{out}$ and $PC_{in}$. This could not be done in the previous t-state, as two data cannot be given simultaneously on the data bus, else it will get mixed up. Only one data can be given on the data bus in any clock pulse, but as many as required can accept the data.
- In the final t-state, the contents received from the memory i.e. the instruction is transferred to its correct place i.e. the instruction register. This is done by the control signals namely $MBR_{out}$ and $IR_{in}$. This also completes the entire fetch operation of the instruction.

### 3.3.2 Execute Cycle

- Execute cycle as discussed can be of various types based on the operation to be performed in the instruction and the location of the operand. We will see some examples in this subsection.
- The first example we will take for the execution of a direct addressed operand. In this case the address of the operand is directly given in the instruction. It involves different operations in various t-states as shown in Table 3.3.2 assuming the instruction ADD R1, [X].

**Table 3.3.2 : Microinstructions for the execute cycle of direct addressed mode of operand access**

| | Operation | Microinstructions |
|---|---|---|
| T1 | IR → MAR | $IR_{out}$(address), $MAR_{in}$, Read, Clear $C_{in}$ |
| T2 | M → MBR | $R1_{out}$, $Y_{in}$, Wait for memory read cycle |
| T3 | MBR + R1 → R1 | $MBR_{out}$, Add, $Z_{in}$ |
| T4 | | $Z_{out}$, $R1_{in}$ |

- In this case of direct addressing mode, the address of the memory operand is in the instruction itself. The instruction as we have seen in the fetch cycle reaches the IR register. Hence the IR register is given a signal to give out the address part and the MAR register to accept this address input value by giving the control signals $IR_{out}$(address) and $MAR_{in}$. At the same time, since the memory is to be read from the control signal is given to the memory i.e. "Read". Also the carry flag is cleared to get ready for the addition operation.
- Since the instruction expects addition of the register 'R1' and the data at memory location with address 'X', the contents of register 'R1' are transferred to the 'Y' register, which is one of the operands for any ALU operation. To perform this transfer operation the control signals given are $R1_{out}$ and $Y_{in}$. Also by the end of the second t-state, the data operand required from the memory will be available in the MBR register.
- In the third t-state the contents of the MBR, which is the content of memory location with the address 'X', is placed on the internal data bus and the ALU is indicated to perform the addition operation. It adds the contents of the 'Y' register and the contents of the internal data bus, and the result is given to the 'Z' register.
- An extra t-state is required to send the data from the 'Z' register to the register R1, as seen earlier two data cannot be given simultaneously on the data bus in the

same t-state. And the contents of memory location with the address 'X' are already put on the data bus in the third t-state.

- The fourth t-state is thus required to transfer the data from register 'Z' to register R1 using the signals $Z_{out}$, $R1_{in}$.
- Another execute cycle we will be studying in this sub-section is for the indirect addressed operand. In this case, the address given in the instruction is the memory location that contains the address of the operand.
- The Table 3.3.3 shows the micro-operations required for such an execute cycle for an example instruction ADD R1, [[X]]
- Table 3.3.3 shows the control signals to be given exactly similar to that of the Table 3.3.2, with a minor difference i.e. the value received in the MBR on first memory read is the operand address and hence is to be given back to the memory to fetch the actual operand.

**Table 3.3.3 : Microinstructions of the execute cycle of an indirect addressed operand instruction**

| | Operation | Microinstructions |
|---|---|---|
| T1 | IR → MAR | $IR_{out}$(address), $MAR_{in}$, Read, Clear $C_{in}$ |
| T2 | M → MBR | $R1_{out}$, $Y_{in}$, Wait for memory read cycle |
| T3 | MBR → MAR | $MBR_{out}$(address), $MAR_{in}$, Read |
| T4 | M → MBR | Wait for memory read cycle |
| T5 | MBR + R1 → R1 | $MBR_{out}$, Add, $Z_{in}$ |
| T6 | | $Z_{out}$, $R1_{in}$ |

### 3.3.3 Interrupt Cycle

- It is concerned to perform the test for any pending interrupts at the end of every instruction execution and if an interrupt occurs.
- It involves the different micro-operations for various t-states as shown in Table 3.3.4.
- Here you will notice a special register used called as *the stack pointer* (SP), which always points to the top *of the stack*. This stack is used to store the return *address of the interrupted* program.

**Table 3.3.4 : Microinstructions for the interrupt cycle**

| | Operation | Microinstructions |
|---|---|---|
| T1 | SP ← SP – 1 | $SP_{out}$(address), Decrement, $Z_{in}$ |
| T2 | SP → MAR | $Z_{out}$, $MAR_{in}$, $SP_{in}$ |
| T3 | PC → MBR | $PC_{out}$(return address), $MBR_{in}$, Write |
| T4 | ISR address → PC | ISR address out, $PC_{in}$ (new address), Wait for memory write cycle |

- The control signals are to be generated using the control unit. The design of this control unit can be done in two ways namely: Hardwired Control Unit and Microprogrammed Control Unit. We will see these two methods in the subsequent sections.

**Syllabus Topic : Examples on Microprograms**

### 3.3.4 Examples of Microprograms

1. Write a microprograms for the instruction : MOV $R_3$, $R_4$

| T-state | Operation | Microinstructions |
|---|---|---|
| T1 | PC → MAR | PCout, MARin, Read, Clear y, Set Cin, Add, Zin |
| T2 | M → MBR<br>PC ← PC + 1 | Zout, PCin, Wait for memory fetch cycle |
| T3 | MBR → IR | MBRout, IRin |
| T4 | $R_3$ ← $R_4$ | $R_{4\,out}$, $R_{3\,in}$ |
| T5 | Check for intr | Assumption enabled intr pending<br>CLRX, SETC, SPout, SUB, Zin, |
| T6 | SP ← Sp – 1 | Zout, SPin, MARin |
| T7 | PC → MDR | PCout, MDR in, WRITE |
| T8 | MDR → [SP] | Wait for mem access |
| T9 | PC ← IS Raddr | PCin IS Raddr out |



2. Write a microprogram for the instruction : ADD $R_3$, $R_4$

Q. Write microprogram for : ADD [R1],[R2] **(5 Marks)**

| T-state | Operation | Microinstructions |
|---|---|---|
| T1 | PC → MAR | PCout, MARin, Read, Clear y, Set Cin, Add, Zin |
| T2 | M → MBR<br>PC ← PC + 1 | Zout, PCin, Wait for memory fetch cycle |
| T3 | MBR → IR | MBRout, IRin |
| T4 | $R_3$ → x | $R_{3\,out}$, $X_{in}$, CLRC |
| T5 | $R_4$ → ALU | $R_{4\,out}$, ADD, $Z_{in}$ |
| T6 | Z → $R_3$ | $Z_{out}$, $R_{3\,in}$ |
| T7 | Check for intr | Assumption enabled intr pending<br>CLRX, SETC, SPout, SUB, Zin, |
| T8 | SP ← Sp – 1 | Zout, SPin, MARin |
| T9 | PC → MDR | PCout, MDR in, WRITE |
| T10 | MDR → [SP] | Wait for mem access |
| T11 | PC ← IS Raddr | PCin IS Raddr out |

3. Write a microprogram for the instruction: MOV $R_3$ , [$R_4$] OR LOAD $R_3$ , [$R_4$]

| T-state | Operation | Microinstructions |
|---|---|---|
| T1 | PC → MAR | PCout, MARin, Read, Clear y, Set Cin, Add, Zin |
| T2 | M → MBR<br>PC ← PC + 1 | Zout, PCin, Wait for memory fetch cycle |
| T3 | MBR → IR | MBRout, IRin |
| T4 | $R_4$ → MAR | $R_{4\,out}$, $MAR_{in}$, READ |
| T5 | Mem → MDR | Wait for mem access |
| T6 | MDR → $R_3$ | $MDR_{out}$ , $R_{3\,in}$ |
| T7 | Check for intr | Assumption enabled intr pending<br>CLRX, SETC, SPout, SUB, Zin, |
| T8 | SP ← Sp – 1 | Zout, SPin, MARin |
| T9 | PC → MDR | PCout, MDR in, WRITE |
| T10 | MDR → [SP] | Wait for mem access |
| T11 | PC ← IS Raddr | PCin IS Raddr out |

4. Write a microprogram for the instruction : ADD $R_3$, [$R_4$]

| T-state | Operation | Microinstructions |
|---|---|---|
| T1 | PC → MAR | PCout, MARin, Read, Clear y, Set Cin, Add, Zin |
| T2 | M → MBR<br>PC ← PC + 1 | Zout, PCin, Wait for memory fetch cycle |
| T3 | MBR → IR | MBRout, IRin |
| T4 | $R_4$ → MAR | $R_{4\,out}$, $MAR_{in}$, READ, CLRC |
| T5 | Mem → MDR | Wait for men access |
| T5 | MDR → ALU | MD $R_{out}$ , $Z_{in}$ ADD |
| T6 | $R_3$ → $X_1$ | $R_{3out}$, $X_{in}$ |
| T7 | Z → $R_3$ | $Z_{out}$, $R_{3\,in}$ |
| T8 | Check for intr | Assumption enabled Intr pending<br>CLRX, SETC, SPout, SUB, Zin, |
| T9 | SP ← Sp – 1 | Zout, SPin, MARin |
| T10 | PC → MDR | PCout, MDR in, WRITE |
| T11 | MDR → [SP] | Wait for mem access |
| T12 | PC ← IS Raddr | PCin IS Raddr out |

5. Write a microprogram for the instruction : ADD $R_3$, [ [$R_4$] ]

| T-state | Operation | Microinstructions |
|---|---|---|
| T1 | PC → MAR | PCout, MARin, Read, Clear y, Set Cin, Add, Zin |
| T2 | M → MBR<br>PC ← PC + 1 | Zout, PCin, Wait for memory fetch cycle |
| T3 | MBR → IR | MBRout, IRin |
| T4 | mem → MDR | Wait for mem access |
| T5 | MDR → ALU | $MDR_{out}$, $Z_{in}$, ADD |
| T6 | Z → $R_3$ | $Z_{out}$, $R_{3\,in}$ |
| T7 | Check for intr | Assumption enabled pending<br>CLRX, SETC, SP SUB, Zin, |

| T-state | Operation | Microinstructions |
|---|---|---|
| T8 | SP ← Sp – 1 | Zout, SPin, MARin |
| T9 | PC → MDR | PCout, MDR in, WRITE |
| T10 | MDR → [SP] | Wait for mem access |
| T11 | PC ← IS Raddr | PCin IS Raddr out |

6. Write a microprogram for the instruction : ADD $R_3$, 45H

| T-state | Operation | Microinstructions |
|---|---|---|
| T1 | PC → MAR | PCout, MARin, Read, Clear y, Set Cin, Add, Zin |
| T2 | M → MBR<br>PC ← PC + 1 | Zout, PCin, Wait for memory fetch cycle |
| T3 | MBR → IR | MBRout, IRin |
| $T_4$ | $R_3$ → X | $R_{out}$, $X_{in}$, CLRC |
| $T_5$ | IRdata → ALU | $IR_{out}$, ADD, $Z_{in}$ |
| $T_6$ | Z → $R_3$ | $Z_{out}$, $R_{3\,in}$ |
| T7 | Check for intr | Assumption enabled intr pending<br>CLRX, SETC, SPout, SUB, Zin, |
| T8 | SP ← Sp – 1 | Zout, SPin, MARin |
| T9 | PC → MDR | PCout, MDR in, WRITE |
| T10 | MDR → [SP] | Wait for mem access |
| T11 | PC ← IS Raddr | PCin IS Raddr out |

7. Write a microprogram for the instruction : ADD $R_3$, [45H]

| T-state | Operation | Microinstructions |
|---|---|---|
| T1 | PC → MAR | PCout, MARin, Read, Clear y, Set Cin, Add, Zin |
| T2 | M → MBR<br>PC ← PC + 1 | Zout, PCin, Wait for memory fetch cycle |
| T3 | MBR → IR | MBRout, IRin |
| T4 | IR addr → MAR | $IR_{out}$, $MAR_{in}$, D, CLRC |
| | $R_3$ → X | $R_3$ ... X |



| T-state | Operation | Microinstructions |
|---|---|---|
| T5 | mem → MDR | Wait for mem access |
| T6 | MDR → ALU<br>[$R_3$ + 45] | $MDR_{out}$, $Z_{in}$, ADD |
| T7 | Z → $R_3$ | $Z_{out}$, $R_{3\,in}$ |
| T8 | Check for intr | Assumption enabled intr pending<br>CLRX, SETC, $SP_{out}$, SUB, $Z_{in}$, |
| T9 | SP ← Sp – 1 | $Z_{out}$, $SP_{in}$, MARin |
| T10 | PC → MDR | $PC_{out}$, MDR in, WRITE |
| T11 | MDR → [SP] | Wait for mem access |
| T12 | PC ← IS Raddr | $PC_{in}$ IS Raddr out |

8. Write a microprogram for the instruction ADDX, [Y]

| T-state | Operation | Microinstruction |
|---|---|---|
| T1 | PC → MAR | $PC_{out}$, $MAR_{in}$, READ, CLRT, STC, ADD, Z |
| T2 | mem → MDR<br>PC ← PC + 1 | Wait for mem access<br>$Z_{out}$, $PC_{in}$ |
| T3 | MDR → IR | $MDR_{out}$, $IR_{in}$ |
| T4 | Y → MAR | $y_{out}$, $MAR_{in}$, READ, CLRC |
| T5 | mem → MDR<br>X → Temp | Wait for mem access<br>$X_{out}$, $T_{in}$ |
| T6 | MDR → ALU | $MDR_{out}$, $Z_{in}$, ADD |
| T7 | Z → X | $Z_{out}$, $X_{in}$ |
| T8 | Check for intr | Assumption enabled intr pending<br>CLRX, SETC, SPout, SUB, Zin, |
| T9 | SP ← Sp – 1 | Zout, SPin, MARin |
| T10 | PC → MDR | PCout, MDR in, WRITE |
| T11 | MDR → [SP] | Wait for mem access |
| T12 | PC ← IS Raddr | PCin IS Raddr out |





9. Write a microprogram for the instruction : ADD X, [[400]]

| T-state | Symbolic operations | Microinstruction |
|---|---|---|
| T1 | PC → MAR | $PC_{out}$, $MAR_{in}$, READ, CLRT, SETC, ADD, Z |
| T2 | mem → MDR<br>PC ← PC + 1 | Wait for mem access<br>$Z_{out}$, $PC_{in}$ |
| T3 | MDR → IR | $MDR_{out}$, $IR_{in}$ |
| T4 | IRaddr → MAR | $IR_{out}$, $MAR_{in}$, READ, CLRC |
| T5 | mem → MDR<br>X → Temp | Wait for mem access<br>$X_{out}$, $T_{in}$ |
| T6 | MDR → MAR | $MDR_{out}$, $MAR_{in}$, READ |
| T7 | mem → MDR | Wait for mem access |
| T8 | MDR → ALU | $MDR_{out}$, ADD, $Z_{in}$ |
| T9 | Z → X | $Z_{out}$, $X_{in}$ |
| T8 | Check for intr | Assumption enabled intr pending<br>CLRX, SETC, SPout, SUB, Zin, |
| T9 | SP ← Sp – 1 | Zout, SPin, MARin |
| T10 | PC → MDR | PCout, MDR in, WRITE |
| T11 | MDR → [SP] | Wait for mem access |
| T12 | PC ← IS Raddr | PCin IS Raddr out |

### 3.3.5 Applications of Microprogramming

→ (MU - May 2014, May 2015)

Q. What are applications of microprogramming? May 14, May 15, 3 Marks

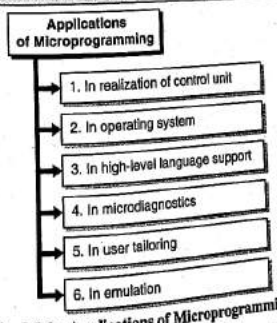



**Fig. 3.3.2 : Applications of Microprogramming**

The applications of microprogramming are :

→ 1. **In Realization of control unit :** Microprogramming is used widely now for implementing the control unit of computers.

→ 2. **In Operating system :** Microprograms can be used to implement some of the primitives of operating system. This simplifies operation system implementation and also improves the performance of the operating system.

→ 3. **In High-Level Language support :** In High-Level language various sub functions and data types can be implemented using microprogramming. This makes compilation into an efficient machine language from possible.

→ 4. **In Micro diagnostics :** Microprogramming can be used for detection isolation monitoring and repair of system errors. This known as micro diagnostics and they significant enhance system maintenance.

→ 5. **In User Tailoring :** By using RAM for implementing control memory (CM), it is possible to tailor the machine to different applications.

→ 6. **In Emulation :** Emulation refers to the use of a microprograms on one machine to execute programs originally written for another machine. This is used widely as an aid for users in migrating from one computer to another.

**Syllabus Topic : Hardwired Control Unit Design Methods: State table, delay element, sequence counter with examples like control unit for multiplication and division**

## 3.4 Control Unit : Hardwired Control Unit Design Methods

→ (MU - May 2014, May 2015, Dec 2015, Dec 2016, May 2017

Q. Explain with diagram functioning of Hardwired Contro unit. May 15, 8 Marks

Q. Describe hardwire control unit and specify advantages. May 14, Dec. 15, Dec. 16, May 17, 10 Marks

– The hardwired Control unit is viewed as a sequential combinational logic circuit. It is used to generate a of fixed sequences of control signals. It is implemen using any of a variety of "standard" digital lo circuits.

- The major advantages of hardwired control units are higher speed of operation and smaller space required for implementation on silicon wafer i.e. the IC (Integrated Circuit), since the components required are lesser.
- The only disadvantage is that modifications to the design are slightly difficult.
- The use of hardwired control unit is majorly found in the RISC designs.
- There are different methods to implement hardwired control unit :

1. State table method.
2. Delay-Element method.
3. Sequence counter method.
4. PLA method.

### 1. State table method

**Q. Write short note on state table method of control unit design (5 Marks)**

- In this method state transition for each instruction is made and hence a state table is obtained.
- This state table is then combine to form a instruction set state table, where all the instructions (OPCODE) are considered as inputs and according to this the next state is being determined. Each state with a set of microinstructions to be issued to various components of the processor as well as external control signals.
- This state table is then implemented using flip-flops and combinational circuit to generate different control signals.
- An example state table implementation is shown in Fig. 3.4.1.

| | Inputs | | | | |
|---|---|---|---|---|---|
| State | $I_1$ | $I_2$ | ..... | $I_m$ | |
| $S_1$ | $S_{1,1}$, $O_{1,1}$ | $S_{1,2}$, $O_{1,2}$ | ..... | $S_{1,m}$, $O_{1,m}$ | |
| $S_2$ | $S_{2,1}$, $O_{2,1}$ | $S_{2,2}$, $O_{2,2}$ | ..... | $S_{2,m}$, $O_{2,m}$ | |
| : | | | ..... | | |
| $S_n$ | $S_{n,1}$, $O_{n,1}$ | $S_{n,2}$, $O_{n,2}$ | ..... | $S_{n,m}$, $O_{n,m}$ | |

*Fig. 3.4.1(a) : Mealy*

| | Inputs | | | | |
|---|---|---|---|---|---|
| State | $I_1$ | $I_2$ | ..... | $I_m$ | Outputs |
| $S_1$ | $S_{1,1}$ | $S_{1,2}$ | ..... | $S_{1,m}$ | $O_1$ |
| $S_2$ | $S_{2,1}$ | $S_{2,2}$ | ..... | $S_{2,m}$ | $O_2$ |
| : | | | ..... | | |
| $S_n$ | $S_{n,1}$ | $S_{n,2}$ | ..... | $S_{n,m}$ | $O_n$ |

*(b) Moore Type*

*Fig. 3.4.1 : State tables for a finite-state machine*

### 2. Delay element method

- This method is implemented using delay elements i.e. D-flipflops.
- A flipflop is made to give output logic '1' after the specific event or in a t-state in sequence and the outputs of these flipflops are used to generate control signals for the micro-instructions i.e. two operations that require a delay of 1 t-state between them are separated by a D flipflop between them. Fig. 3.4.2 shown this implementation.

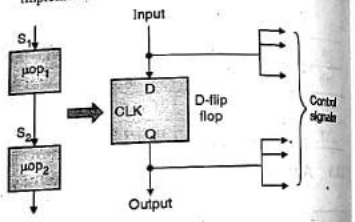



*Fig. 3.4.2 : Use of D flip flop as a delay element between two sets of control signals*

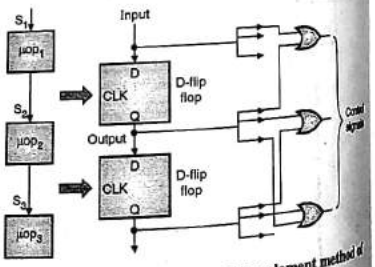



*Fig. 3.4.3 : Use of OR gate in delay element method of Hardwired control unit*



- The signals that activate the same control signal are ORed together i.e. if a signal has to be activated from the outputs of multiple flipflops then an OR gate is used as shown in Fig. 3.4.3.
- In case if a decision is to be made then it is implemented using a If-Then-Else circuit i.e. two AND gates coupled to a OR gate. This is shown in Fig. 3.4.4.

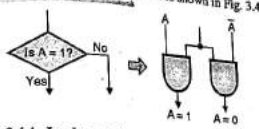



*Fig. 3.4.4 : Implementation of If-Then-Else in delay element method of Hardwired control unit*

### 3. Sequence counter method

**Q. Explain the sequence counter method of hardwired control unit (5 Marks)**

- In this method, multiple clock signals are derived from the master clock using a standard counter-decoder approach as shown in the Fig. 3.4.5. These signals are applied to the combinational portion of the circuit.
- As shown in Fig. 3.4.5, the counter keeps on incrementing and generating different counts. The counts are decoded using a decoder and the decoder outputs are given to various components as control signals in the CPU.

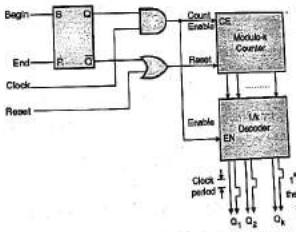



*Fig. 3.4.5 : Sequential counter method of Hardwired control unit implementation*

### 4. PLA method

- In this method a PLA (Programmable Logic Array) is used to generate the control signals. PLA is an array of AND gates at input and the OR gates at output.
- The inputs are to be given to the AND gates, which can be connected to the specific OR gates as required.
- The OR gates outputs are the outputs of the overall PLA and are used as control signals in the system i.e. the inputs to the AND array is from various control signals generated and the output of the OR array is given as control signals to various components of the processor as well as the external control signals required.
- Fig 3.4.6 shows the implementation of the PLA method of implementation of control unit.

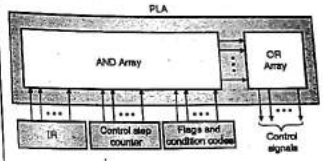



*Fig. 3.4.6 : PLA Technique*

## 3.5 Control Unit : Soft Wired (Micro programmed) Control Unit Design Methods

→ (MU - May 2014)

**Q. Explain with block diagram the micro-programmed control unit (5 Marks)**

**Q. Explain with diagram functioning of Microprogrammed Control Unit. May 14, 8 Marks**

- Micro programmed control unit generates control signals based on the microinstructions stored in a special memory called as the control memory.
- Each instruction points to a corresponding location in the control memory that loads the control signals in the control register.
- The control register is then read by a sequencing logic that issues the control signals in a proper sequence.
- The implementation of the micro programmed is shown in the Fig. 3.5.1.
- The Instruction Register (IR), Status flag and condition codes are read by the sequencer that generates the address of the control memory location for the corresponding instruction in the IR.
- This address is stored in the Control address register that selects one of the locations in the control memory having the corresponding control signals.

- These control signals are given to the microinstruction register, decoded and then given to the individual components of the processor and the external devices.

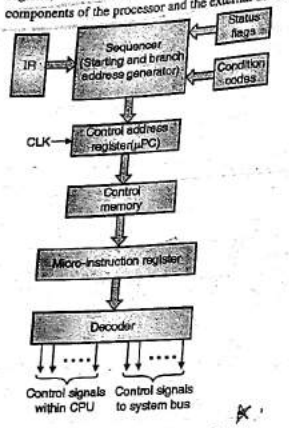



**Fig. 3.5.1 : Micro programmed control unit**

**Syllabus Topic : Wilkie's Microprogrammed Control Unit**

### 3.5.1 Wilkie's Microprogrammed Control Unit → (MU - Dec. 2014)

**Q. Explain Wilkie's microprogrammed control unit. (5 Marks)**

**Q. Explain Wilkie's Engine (Hardwired Control Unit) in detail. Dec. 14, 10 Marks**

First working model of a micro-programmed control unit was proposed by Wilkie's in 1952. In the above design, a microinstruction has two major components :

- *Control field*
- *Address field*

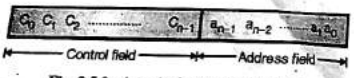



***Fig. 3.5.2 : A typical microinstruction***

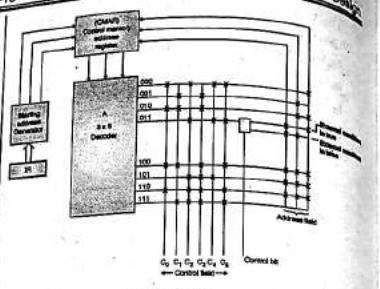


**Fig. 3.5.3 : Wilkes control**

If a microinstruction is encoded as given below :

| $C_0$ | $C_1$ | $C_2$ | $C_3$ | $C_4$ | $C_5$ | $C_6$ | $A_2$ | $A_1$ | $A_0$ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

- Then the control information 0100110 indicates that on execution of above microinstruction, control signals $C_1$, $C_4$ and $C_5$ will be activated. Address field contains the address of the next microinstruction.
- Thus, after execution of the above instruction, the next instruction to be executed is one which is at the address 010.
- The control field tells the control signals which are to be activated and the address field provide the address of the next microinstruction to be executed.
- In Wilkie's control, control memory is organized as a program logic array.
- The Control Memory Access Register (CMAR) can be loaded from an external source (instruction register) as well as from the address field of a microinstruction. A machine instruction typically provides the starting address of a micro-program in control memory.
- On the basis of starting address from instruction register, decoder activates one of the eight output lines.
- This activated line, in turn, generates control signals and the address of the next microinstruction to be executed.
- This address is once again fed to the CMAR resulting in activation of another control line and address field.
- This cycle is repeated till the execution of the instruction is achieved.



For example, as shown below, if the machine instruction under execution causes the decoder to have an entry address for a machine instruction in control memory at line 000. The decoder activates the lines in the sequence given below :

| Line activated | Control signal generated | Address of next microinstruction |
|---|---|---|
| 000 | $C_0, C_2, C_4, C_5$ | 001 |
| 001 | $C_1, C_3$ | 010 |
| 010 | $C_0, C_1, C_3$ | 011 |
| 011 | $C_2, C_4, C_5$ | 2 |

On execution of microinstruction at address 011, address of the next microinstruction depends on the external condition. If the condition is true then the address 101 will be selected else the address 110 will be selected.

### 3.5.2 Comparison between Hardwired and Micro-programmed Control

| Attribute | Hardwired Control | Micro-programmed Control |
|---|---|---|
| Speed | Fast | Slow |
| Cost of implementation | More | Cheaper |
| Implementation approach | Sequential circuit | Programming |
| Flexibility | Not flexible, difficult to modify for new instruction. | Flexible, new machine instructions can easily be added. |
| Ability to handle Complex instructions | Difficult | Easier |
| Design process | Complicated | Systematic |
| Decoding and sequencing logic | Complex | Easy |
| Applications | RISC μp | CISC μp |
| Instruction set size | Small | Large |
| Control memory | Absent | Present |
| Chip area required | Less | More |

## 3.6 Concepts of Nano Programming

→ **(MU - May 2014, Dec. 2014, May 2015, Dec. 2016)**

**Q. Explain concepts of nanoprogramming. May 14, Dec. 14, May 15, Dec. 16, 6 Marks**

The following block diagram of Fig. 3.6.1 explains the flow in a nanoprogram concept.

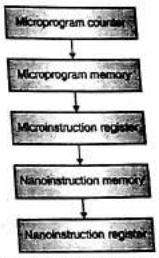



**Fig 3.6.1 : Nano programming**

The microprogram memory (control ROM) is much narrower because it contains pointers to the actual microinstructions. The microinstruction register contains a short pointer that points to the nanoinstruction memory. The nanoinstruction memory contains the actual micro-instructions which is very wide.

## 3.7 Exam Pack (University and Review Questions)

☞ **Syllabus Topic : Micro programmed control Unit: Microinstruction sequencing and execution**

**Q.** Explain fetch cycle with the corresponding micro-program *(Ans. : Refer section 3.3.1)* **(5 Marks)**

**Q.** Describe the register organization within the CPU. *(Ans. : Refer section 3.1)* **(May 2016, 10 Marks)**

**Q.** Explain Microinstruction sequencing and execution. *(Ans. : Refer section 3.3)* **(May 2015, 7 Marks)**

**Q.** Microinstructions to execute an instruction MOV [R1], R2. *(Ans. : Refer section 3.3)* **(May 2016, 6 Marks)**

**Q.** Explain micro instruction sequencing and execution. *(Ans. : Refer section 3.3)* **(Dec. 2016, 10 Marks)**

**Syllabus Topic : Examples on microprograms**

Q. What are applications of microprogramming? *(Ans. : Refer section 3.3.5)* **(May 2014, May 2015, 3 Marks)**

**Syllabus Topic : Hardwired Control Unit Design Methods : State table, delay element, sequence counter with examples like control unit for multiplication and division**

Q. Write short note on state table method of control unit design *(Ans. : Refer section 3.4.(1))* **(5 Marks)**

Q. Explain the sequence counter method of hardwired control unit *(Ans. : Refer section 3.4(3))* **(5 Marks)**

Q. Explain with block diagram the micro-programmed control unit *(Ans. : Refer section 3.5)* **(5 Marks)**

Q. Describe hardwired control unit and specify its advantages. *(Ans. : Refer section 3.4)* **(May 2014, 7 Marks)**

Q. Explain with diagram functioning of Hardwired Control unit. *(Ans. : Refer section 3.4)* **(May 2015, 8 Marks)**

Q. Describe hardwire control unit and specify its advantages. *(Ans. : Refer section 3.4)* **(Dec. 2015, Dec. 2016, May 2017, 10 Marks)**

Q. Explain with diagram functioning of Microprogrammed Control Unit. *(Ans. : Refer section 3.5)* **(May 2014, 8 Marks)**

**Syllabus Topic : Wilkie's Microprogrammed Control Unit**

Q. Explain Wilkie's microprogrammed control unit *(Ans. : Refer section 3.5.1)* **(5 Marks)**

Q. Explain Wilkie's Engine (Hardwired Control Unit) in detail. *(Ans. : Refer section 3.5.1)* **(Dec. 2014, 10 Marks)**

Q. Explain concepts of nanoprogramming. *(Ans. : Refer section 3.6)* **(May 2014, 6 Marks)**

Q. Nano-programming. *(Ans. : Refer section 3.6)* **(Dec 2014, 5 Marks)**

Q. Explain concepts of Nano programming. *(Ans. : Refer section 3.6)* **(May 2015, Dec 2016, 6 Marks)**

000

CHAPTER 4

# Memory Organization

Module IV

**Syllabus**

Classifications of primary and secondary memories. Types of RAM (SRAM, DRAM, SDRAM, DDR, SSD) and ROM, Characteristics of memory, Memory hierarchy: cost and performance measurement. Virtual Memory: Concept, Segmentation and Paging, Address translation mechanism. Interleaved and Associative memory. Cache memory Concepts, Locality of reference, design problems based on mapping techniques. Cache Coherency, Write Policies.

**Syllabus Topic : Characteristics of Memory**

## 4.1 Introduction to Memory and Memory Parameters

→ **(MU - May 2014, May 2016, Dec. 2016)**

Q. What are characteristics of memory devices ? **May 14, 8 Marks**

Q. Describe the characteristics of Memory. **May 16, Dec. 16, 10 Marks**

When a memory is taken then there are various characteristics of this memory that are considered. The characteristics of memory are based on the following :

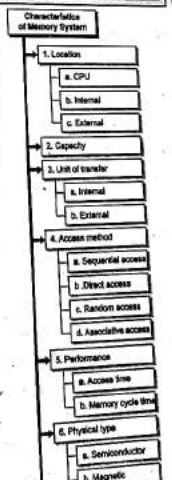



**Fig. 4.1.1 : Characteristics of memory system**

→ **1. Location :** The memory can be located in one of the following :

(a) **CPU :** This includes CPU registers and on-chip cache memory.

(b) **Internal :** This includes the memory that the processor can directly access.

(c) **External :** This is normally removable or virtual memory and hence access is slower.

→ **2. Capacity :** It is measured in terms of the word size and the number of words. Word size is the size of each location. Number of words is the number of locations.

→ **3. Unit of transfer :** This refers to the size of the data that is transferred in one clock cycle. It mainly depends on the data bus size. The data as discussed earlier may be internal or external and accordingly will be the data to be transferred in one clock pulse :

(a) **Internal :** It is related to the communication o data with the memory directly accessible. It i usually governed by data bus width.

(b) **External :** This is the data communication wi the external removable memory or virtu memory. It is usually a block which is mu larger than a word.

→ **4. Access method :** There are various methods accessing the memory based on the mem organization. These methods are listed be with examples :

(a) **Sequential access :** The sequential access m start from the beginning and read throug order until the byte to be read is rea Hence the access time depends on locati data and previous location acc

**Syllabus Topic : Examples on microprograms**

Q. What are applications of microprogramming? *(Ans. : Refer section 3.3.5)* **(May 2014, May 2015, 3 Marks)**

**Syllabus Topic : Hardwired Control Unit Design Methods : State table, delay element, sequence counter with examples like control unit for multiplication and division**

Q. Write short note on state table method of control unit design *(Ans. : Refer section 3.4.(1))* **(5 Marks)**

Q. Explain the sequence counter method of hardwired control unit *(Ans. : Refer section 3.4(3))* **(5 Marks)**

Q. Explain with block diagram the micro-programmed control unit *(Ans. : Refer section 3.5)* **(5 Marks)**

Q. Describe hardwired control unit and specify its advantages. *(Ans. : Refer section 3.4)* **(May 2014, 7 Marks)**

Q. Explain with diagram functioning of Hardwired Control unit. *(Ans. : Refer section 3.4)* **(May 2015, 8 Marks)**

Q. Describe hardwire control unit and specify its advantages. *(Ans. : Refer section 3.4)* **(Dec. 2015, Dec. 2016, May 2017, 10 Marks)**

Q. Explain with diagram functioning of Microprogrammed Control Unit. *(Ans. : Refer section 3.5)* **(May 2014, 8 Marks)**

**Syllabus Topic : Wilkie's Microprogrammed Control Unit**

Q. Explain Wilkie's microprogrammed control unit *(Ans. : Refer section 3.5.1)* **(5 Marks)**

Q. Explain Wilkie's Engine (Hardwired Control Unit) in detail. *(Ans. : Refer section 3.5.1)* **(Dec. 2014, 10 Marks)**

Q. Explain concepts of nanoprogramming. *(Ans. : Refer section 3.6)* **(May 2014, 6 Marks)**

Q. Nano-programming. *(Ans. : Refer section 3.6)* **(Dec 2014, 5 Marks)**

Q. Explain concepts of Nano programming. *(Ans. : Refer section 3.6)* **(May 2015, Dec 2016, 6 Marks)**

□□□

CHAPTER 4

Module IV

# Memory Organization

**Syllabus**

Classifications of primary and secondary memories. Types of RAM (SRAM, DRAM, SDRAM, DDR, SSD) and ROM, Characteristics of memory, Memory hierarchy: cost and performance measurement. Virtual Memory: Concept, Segmentation and Paging, Address translation mechanism. Interleaved and Associative memory. Cache memory Concepts, Locality of reference, design problems based on mapping techniques. Cache Coherency, Write Policies.

**Syllabus Topic : Characteristics of Memory**

## 4.1 Introduction to Memory and Memory Parameters

→ **(MU - May 2014, May 2016, Dec. 2016)**

Q. What are characteristics of memory devices ? **May 14, 8 Marks**

Q. Describe the characteristics of Memory. **May 16, Dec. 16, 10 Marks**

When a memory is taken then there are various characteristics of this memory that are considered. The characteristics of memory are based on the following :

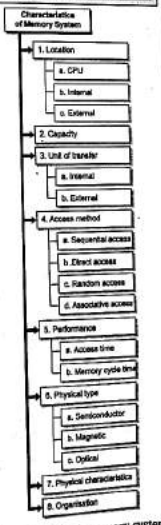



**Fig. 4.1.1 : Characteristics of memory system**

→ **1. Location :** The memory can be located in one of the following :

(a) **CPU :** This includes CPU registers and on-chip cache memory.

(b) **Internal :** This includes the memory that the processor can directly access.

(c) **External :** This is normally removable or virtual memory and hence access is slower.

→ **2. Capacity :** It is measured in terms of the word size and the number of words. Word size is the size of each location. Number of words is the number of locations.

→ **3. Unit of transfer :** This refers to the size of the data that is transferred in one clock cycle. It mainly depends on the data bus size. The data as discussed earlier may be internal or external and accordingly will be the data to be transferred in one clock pulse :

(a) **Internal :** It is related to the communication of data with the memory directly accessible. It is usually governed by data bus width.

(b) **External :** This is the data communication with the external removable memory or virtual memory. It is usually a block which is much larger than a word.

→ **4. Access method :** There are various methods of accessing the memory based on the memory organization. These methods are listed below with examples :

(a) **Sequential access :** The sequential access means start from the beginning and read through in order until the byte to be read is reached. Hence the access time depends on location of data and previous location accessed.

For Example, magnetic tape. If one wants to listen to the third stanza of the fourth song stored in an audio cassette, he has to go through the entire first song second and the third song, and then the first stanza, second stanza of the third song and then reaches to the third stanza of that song.

(b) **Direct access :** Here individual blocks have unique address and the access is done by jumping to vicinity plus sequential search. Hence access time depends on location and previous location. For Example magnetic or optical disk. Let take the same example that a person wants to listen to the third stanza of the fourth song on a CD, then he can directly reach to the fourth song, but thereafter he has to access the stanzas of the song sequentially to reach to the third stanza.

(c) **Random access :** In case of random access individual addresses identify locations exactly. Hence the access time is independent of location or previous access. For example RAM. In case of a RAM, any location to be accessed can be directly reached to without going through the locations sequentially.

(d) **Associative access :** Here the data is located by a comparison with contents of a portion of the stored data(address). Hence the access time is independent of location or previous access. For example cache. In case of cache memory, each location has a tag associated with it, and to reach to the required location the tags are to be compared with the location to be accessed. There are techniques used to reach to the required tagged location at a faster speed.

→ **5. Performance :** The performance of the memory depends on its speed of operation or the data transfer rate. The data transfer rate is the rate at which the data is transferred. The speed of operation depends on two things :

(a) **Access time :** The time between providing the address and getting the valid data from memory is called as its access time i.e. the address to data time.

(b) **Memory cycle time :** The time that is required for the memory to "recover" before next access i.e. the time between two addresses is called as memory cycle time.

→ **6. Physical type :** The physical material using which the memory is made can be different like :

(a) **Semiconductor :** Memory can be made using semiconductor material i.e. ICs, for example RAM.

(b) **Magnetic :** Memory can also be made using magnetic read and write mechanism, for example Magnetic disk and Magnetic tape.

(c) **Optical :** Optical memories i.e. memories that use optical methods to read and write have become famous these days, for example CD and DVD.

(d) There are some other methods using which data was stored in early days like Bubble and Hologram.

→ **7. Physical characteristics :** The physical characteristic of memory is also an important aspect to be considered. This includes the volatility, power consumption, erasable / not erasable, etc.

→ **8. Organisation :** It is not that always the memory will be organized sequentially. There are some other types of memory organization like interleaved memory, etc. Interleaved memory is used in microprocessor 8086.

### 4.1.1 Bytes and Bits

- The byte is a unit of digital information that mostly consists of eight bits.
- Infact, a byte was the number of bits used to encode a single character of text in a computer and for this reason it has become the basic addressable element in many computer architectures. The size of the byte has been hardware dependent and no definition exists.
- The fact is that standard of eight bits is also convenient power of two permitting the values from 0 to 255 for one byte. With ISO/IEC 80000-13, this common meaning was codified in a formal standard. Many types of applications use variables representable in eight bits or multiple of eight bits.



**Syllabus Topic : Memory Hierarchy: Classifications of Primary and Secondary Memories**

## 4.2 Memory Hierarchy : Classifications of Primary and Secondary Memories

→ (MU - May 2014)

Q. Explain in details Memory Hierarchy with examples. May 14, 6 Marks

Q. Explain memory hierarchy. (5 Marks)

- Memory Hierarchy explains that the nearer the memory to the processor, faster is its access. But costlier the memory becomes as it goes closer to the processor. The following sequence is in faster to slower or costlier to cheaper memory :

1. Registers i.e. inside the CPU.
2. Internal memory that includes one or more levels of cache and the main memory. Internal memory is always RAM, SRAM for cache and DRAM for main memory. The differences between the SRAM and DRAM will be seen in a later section in this chapter. This is also called as the primary memory.
3. External memory or removable memory includes the hard disk, CDs, DVDs etc. This is the secondary memory.

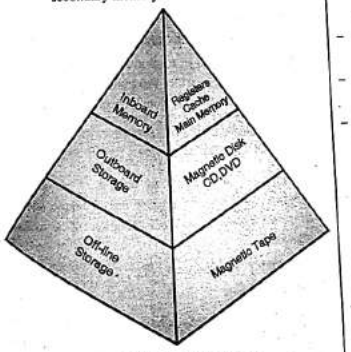



**Fig. 4.2.1 : Memory Hierarchy**

- Fig. 4.2.1 shows the memory hierarchy based on the closeness to the processor. The registers as discussed are the closest to the processor and hence are the fastest while off-line storage like magnetic tape are the farthest and also the slowest. The list of memories from closest to the processor to the farthest is given as below :

1. Registers
2. L1 Cache
3. L2 Cache
4. Main memory
5. Magnetic Disk
6. Optical
7. Tape

To have a large faster memory is very costly and hence the different memory at different levels gives the memory hierarchy. How does this memory hierarchy give faster operation and some other terms like cache etc. will be understood in the subsequent sections.

**Syllabus Topic : Types of RAM (SRAM, DRAM, SDRAM, DDR, SSD)**

## 4.3 Types of RAM and ROM

### 4.3.1 SRAM and DRAM

Q. Compare SRAM and DRAM. (5 Marks)

- RAM (Random Access Memory) is called so because any memory location in this IC can be accessed randomly.
- There are two types of RAM, namely, SRAM (Static RAM) and DRAM (Dynamic RAM).
- SRAM is made up of flip flops while the DRAM is made up of capacitors.
- Since DRAM is made using capacitors, it requires less number of components to make a one bit cell, hence also requires less space on the silicon wafer. Thus it is also comparatively cheaper. But it is slower than SRAM, because capacitors require time for charging and discharging. Also the capacitors loose charge in some time and hence need to be recharged according to the data, this is called as refreshing the DRAM. Fig. 4.3.1(a) shows the structure of a DRAM cell.
- The address line selects the particular location, it enables the MOSFET that connects the capacitor to the data bus and hence if the data is to be read, simply the data line gets the data to be read; while if the data is to be written the data is to be given on the data line and will be written on the capacitor.

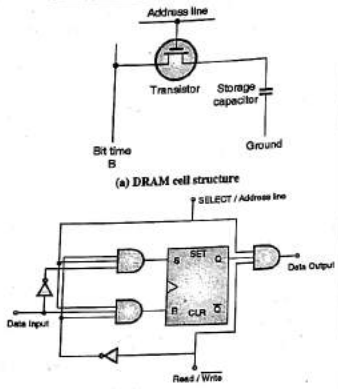



Fig. 4.3.1

- On the other hand, the SRAM has each cell made of a flip-flop, thus requires more components as compared to the DRAM cell. Hence it occupies more space on the silicon wafer, and is costlier. Thus it is also costlier. But the advantage is that it doesn't require any refreshing and is also very fast compare to DRAM. Fig. 4.3.1 (b) shows the structure of the DRAM cell.
- The flip-flop is used to store the data. There is a AND gate at the output of the flip-flop, which will be enabled by the select line (that works as an address line) and the Read / $\overline{\text{Write}}$ operation (when data has to be read) when it is logic '1'. Thus when both i.e. the select line and the Read / $\overline{\text{Write}}$ lines are '1', the output of the flip-flop will be available on the data line.
- Similarly for the write operation, the select line must be enabled by making it logic '1' and the Read / $\overline{\text{Write}}$ line must be logic '0'. Thus the data from the input line will be stored in the flip-flop.
- Table 4.3.1 shows the differences between the SRAM and DRAM.

Table 4.3.1

| Sr. No. | SRAM | DRAM |
|---|---|---|
| 1. | No refreshing required. | Continuous refreshing required (disadvantage). |
| 2. | It is faster for accessing data. | It is slower in accessing data. |
| 3. | It takes more space on chip as more number of components are required per bit. | It takes less space on chip as less number of components are required per bit. |
| 4. | Hence is also costly. | Hence is cheaper. |
| 5. | Bit density is lesser. | Bit density is more. |
| 6. | The bit is stored in a flip-flop. | The bit is stored as a charged or discharged capacitor. |
| 7. | SRAM is mainly used or selected for cache memory. | DRAM is mainly used or selected for semiconductor main memory. |

### 4.3.2 Types of Memory

- We will also see how to place the memory in different locations in the system.
- To begin with we will revise the different types of memory that can be interfaced with the processor.
- The different types of memories used with processors are EPROM (Erasable Programmable Read Only Memory) and RAM (Random Access Memory).

#### 4.3.2.1 Memory Map, Structure and its Requirements

- The read / write memories consist of an array of registers wherein each register has a unique address. Fig. 4.3.2 shows the block diagram of a memory device.

  N : number of registers

  M : word length.
- **Example :** If a memory is having 13 address lines and 8 data lines, then number of registers / memory locations = $2^N = 2^{13} = 8192$ word length = M bit = 8 bit.
- The number of address lines of a microprocessor depends on the size of the memory.



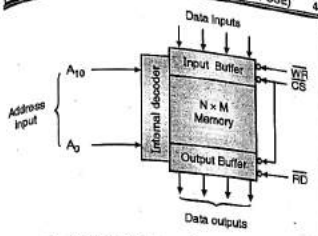



Fig. 4.3.2 : Block diagram of a memory device

| Number of address lines required | Size of memory in bytes |
|---|---|
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |
| 5 | 32 |
| 6 | 64 |
| 7 | 128 |
| 8 | 256 |
| 9 | 512 |
| 10 | 1024 = 1 k |
| 11 | 2048 = 2 k |
| 12 | 4096 = 4 k |
| 13 | 8192 = 8 k |
| 14 | 16384 = 16 k |
| 15 | 32768 = 32 k |
| 16 | 65536 = 64 k |

### 4.3.3 Memory Chip Size and Numbers

**Q.** Interface 8 KB EPROM and 4 KB RAM to a processor with 16-bit address and 8-bit data bus. **(5 Marks)**

Table 4.3.2 : EPROM ICs available in the market

| IC number | Memory size Address data | Number of pins |
|---|---|---|
| 2716 | 2 k × 8 | 24 |
| 2732 | 4 k × 8 | 24 |
| 2764 | 8 k × 8 | 28 |
| 27128 | 16 k × 8 | 28 |
| 27256 | 32 k × 8 | 28 |
| 27512 | 64 k × 8 | 28 |

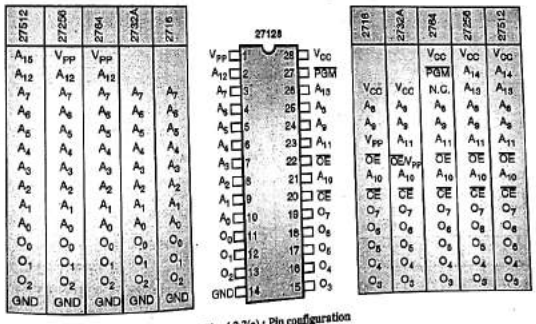


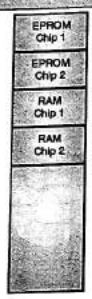

Fig. 4.3.3(a) : Pin configuration

**Ex. 4.3.2**

Interface 8K of EPROM and 8 KB of RAM using 4KB devices.

**Soln. :**

**Note :** Let us assume the processor has 16 address lines.




**Step 1 :** Total EPROM required = 8 kB

Chip size available = 4 kB (IC 2732)

$\therefore$ No of chips required = $\frac{8\text{ kB}}{4\text{ kB}} = 2$

**Chip 1 :** Starting address = 0000H

Chip size = 4 kB = 0FFFH

$\therefore$ Ending address = 0FFFH

**Chip 2 :** Starting address = Previous ending + 1

= 0FFFH + 1 = 1000H

Chip size = 0FFFH

$\therefore$ Ending address = 1FFFH

**Step 2 :** Total RAM required = 8 kB

Chip size available = 4 kB ( IC 6232 )

$\therefore$ No of chips required = $\frac{8\text{ kB}}{4\text{ kB}} = 2$

**Chip 1 :** Starting address = EPROM ending address + 1

= 1FFFH + 1 = 2000H

Chip size = 4 kB = 0FFFH

$\therefore$ Ending address = 2FFFH

**Chip 2 :** Starting address = previous ending + 1

= 2FFFH + 1 = 3000H

Chip size = 4 kB = 0FFFH

$\therefore$ Ending address = 3FFFH

**Step 3 :** Memory Map :

| | | $A_{15}$ $A_{14}$ $A_{13}$ $A_{12}$ | $A_{11}$ $A_{10}$ $A_9$ $A_8$ $A_7$ $A_6$ $A_5$ $A_4$ $A_3$ $A_2$ $A_1$ $A_0$ |
|---|---|---|---|
| EPROM chip 1 $\overline{y}_0$ | SA = 0000H | 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0 |
| | EA = 0FFFH | 0 0 0 0 | 1 1 1 1 1 1 1 1 1 1 1 1 |
| EPROM chip 2 $\overline{y}_1$ | SA = 1000H | 0 0 0 1 | 0 0 0 0 0 0 0 0 0 0 0 0 |
| | EA = 1FFFH | 0 0 0 1 | 1 1 1 1 1 1 1 1 1 1 1 1 |
| RAM chip 1 $\overline{y}_2$ | SA = 2000H | 0 0 1 0 | 0 0 0 0 0 0 0 0 0 0 0 0 |
| | EA = 2FFFH | 0 0 1 0 | 1 1 1 1 1 1 1 1 1 1 1 1 |
| RAM chip 2 $\overline{y}_3$ | SA = 3000H | 0 0 1 1 | 0 0 0 0 0 0 0 0 0 0 0 0 |
| | EA = 3FFFH | 0 0 1 1 | 1 1 1 1 1 1 1 1 1 1 1 1 |



**ex Full decoding logic**

EPROM chip size = 4kB and RAM chip size = 4 kB.

$\therefore$ Smaller chip size (Both are same in this case) = 4 kB = $2^{12}$

$\therefore$ Neglect lower 12 address lines i.e. $A_0$ to $A_{11}$ and consider remaining address lines i.e. $A_{12}$ to $A_{15}$ for decoding. Hence 4:16 decoder is required. Hence circle the four bits as shown in the memory map. EPROM chip one has 0000b, thus it requires $\overline{y}_0$ ; while EPROM chip two has 0001b, thus it requires $\overline{y}_1$ and so on.

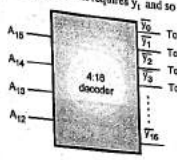



Fig. Ex. 4.3.2

**Step 4 :** Final implementation.

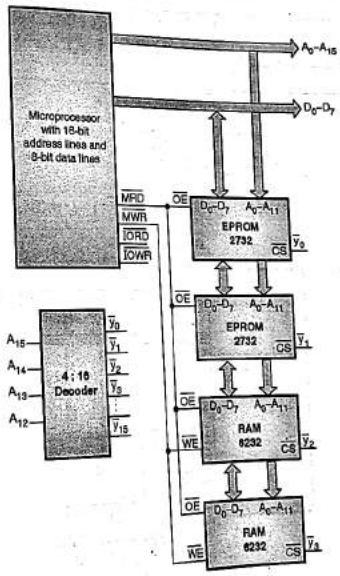



Fig. Ex. 4.3.3(a)

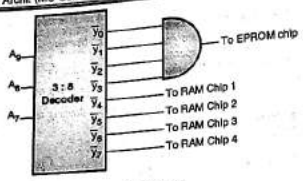



Fig. Ex. 4.3.4

**Absolute (full) decoding logic**

EPROM chip size = 512B while RAM chip size = 128B. Thus smaller chip size = 128B = $2^7$. Therefore neglect lower 7 address lines i.e. $A_0$ to $A_6$. Now since three address lines are remaining, we need a 3 : 8 decoder. The remaining lines i.e. $A_7$ to $A_9$ will be inputs to the decoder, as shown by circles in memory map.

**Step 4 :** Final Implementation

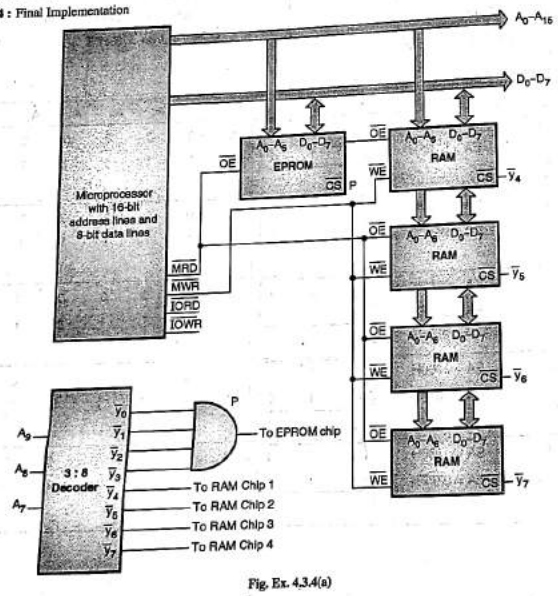



Fig. Ex. 4.3.4(a)



**Syllabus Topic : Types of ROM**

## 4.4 ROM (Read Only Memory)

→ (MU - Dec. 2016)

**Q.** Write a short notes on Types of ROM.
Dec. 16, 5 Marks

- ROM or the read only memory is quite cheaper compared to RAM and is mainly used for implementation of the secondary or the virtual memory. Thus the application of ROM is for virtual or secondary memories like hard disks, external storage like CD / DVD and floppy disks etc.
- We will see different types of ROM in the subsequent sub-section and thereafter some ROM memories used in computers like magnetic disk, CD, DVD etc.

### 4.4.1 Types of ROM

**Q.** Explain various types of ROM : Magnetic as well as optical. (5 Marks)

- There are various types of ROM available based on whether or not it can be re-written; they are called as ROM, PROM, EPROM and EEPROM. These types of memories will be studied in this section.
- There are some more ROMs available these days like flash memory, OTP etc.
- The ROM is a memory wherein, the user cannot write anything. The data to be stored in the ROM is to be given to the ROM manufacturer, who writes this data on the ROM and provides the same.
- The PROM (Programmable Read Only Memory) or also sometimes referred to as OTP (One Time Programmable) memory, as it can be written onto only once. When manufactured, it is blank, once written on it, it cannot be re-written. There are diodes that are used to store data, and they are fused or kept as it is to store the data in them. The internal diagram of the PROM is shown in Fig. 4.4.1.
- The AND array is used as address lines and the OR array as data lines. The AND array (on the left in Fig. 4.4.1) comes as predefined connections as shown in the Fig. 4.4.1 in sequence of binary, in this case from "000" to "111", as there are three bit address.
- The OR array (on the right hand in Fig. 4.4.1) comes with programmable link, the ones to be retained can be retained while the remaining fused or opened.
- Hence whenever a memory address is given to the address lines (a, b and c in Fig. 4.4.1) the specified location will be selected and according to the fused links, the data will be available on the OR gates output lines.

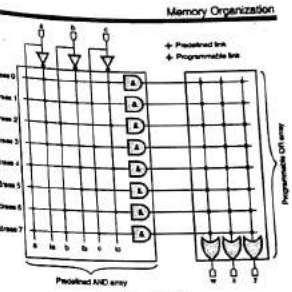



Fig. 4.4.1 : PROM

- The EPROM (Erasable Programmable Read Only Memory) although extinct today and is replaced by EEPROM or $E^2$PROM, but it used to be the only erasable memory available earlier. In case of EPROM the data written can be erased by keeping the EPROM IC in the UV box, as the UV rays erase the previously written data on the EPROM.
- The EPROM as discussed earlier are these days replace with EEPROM (Electrically Erasable Programmable Read Only Memory). The EEPROMs are erased by giving an extra supply voltage.

### 4.4.2 Magnetic Memory

- Magnetic disks are very cheap and widely used as external storage and as hard disks. When used as hard disks, they are called as Winchester Disk.
- Initially magnetic tapes were used for storage. Magnetic tapes are used even today in some places because of its low cost and ease of data storage. When huge amount of data is to be stored, magnetic tape is used.
- Let us see the construction of these magnetic memories.
- The magnetic disk substrate is coated with magnetisable material.
- The aluminium substrate was used earlier but now glass is used because of the following :
  1. Improved surface uniformity
  2. Increase reliability
  3. Reduction in surface defects and read/write errors
  4. Better stiffness and shock/damage resistance.

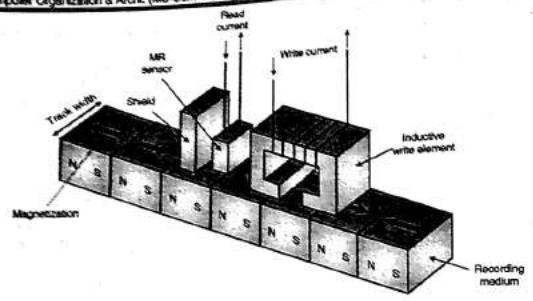
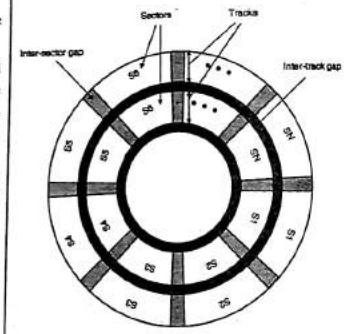


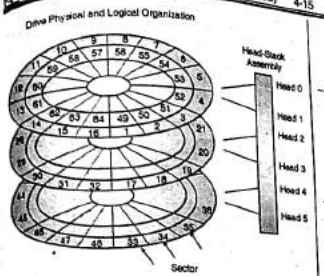

**Fig. 4.4.2 : Read-Write mechanism**

- The reading and writing mechanism of the magnetic memory is shown in the Fig. 4.4.2. Writing and reading of data is done with the help of a conductive coil called as head. The head may be single for both read and write operations or separate ones.
- During read/write operation, head is stationary while the platter (disk) rotates.
- Write operation is done by passing current through coil that produces magnetic field and then the pulses are sent to head. Thus the magnetic pattern i.e. NS (North-South) or SN (South-North) is recorded on surface below.
- Read operation is done by magnetic field moving relative to coil that produces current. According to the magnetic pattern the data is read by the head.
- The organization of data on the platter is in a special manner with concentric circles called as tracks as shown in Fig. 4.4.4(a). Further the tracks are divided into sectors.
- The data is also stored in a special manner such that first the data is stored in the first track of first platter (upper and lower sides) and then in the first track of the second platter(upper and lower sides), then of the third (upper and lower sides) and so on. This is shown in the Fig. 4.4.3(b). Thus when reading from one track of a platter, the head mechanism may not be moved and the other head will start reading from the same track of another platter.




**(a) Data organization on a disk**
**Fig. 4.4.3 contd...**






**(b) Data organization on multiple platters**
**Fig. 4.4.3**

- A floppy disk is single platter, while a hard disk or winchester disk is multi platter as shown in the Fig. 4.4.3(b). In this case one head for each side of the multiple platters are mounted to form a head stack assembly.
- It is called as Winchester hard disk because it was developed by IBM in Winchester (USA). It is a sealed unit with the platters and the heads fly on boundary layer of air as disk spins. Also, there is very small head to disk gap making it more robust. Winchester hard disk is cheap and the fastest external storage.
- There is gap between the two sectors as well as between two tracks as shown in Fig. 4.4.3(a).
- The disk moves at constant angular velocity and hence the data is read at the same speed, may be the innermost track or the outermost track. Each data stored on the disk is stored in a special manner with some ID information as shown in the Fig. 4.4.4.

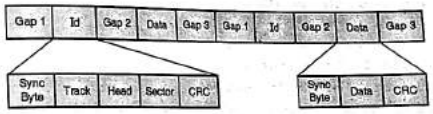



**Fig. 4.4.4 : Data storage format in magnetic memory**

### 4.4.3 Optical Memory

- The memory devices like Compact Disc (CD) and Digital Versatile Disc or Digital Video Disc (DVD) use the optical method to read the data written on them.
- The following sub-sections discuss about the CD and the DVD data storage and reading.

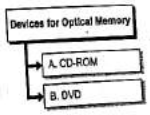



**Fig. 4.4.5 : Devices for optical memory**

#### A. CD-ROM

- CD-ROM was originally built for audio and was of the capacity of 650Mbytes giving over 70 minutes audio.
- The construction of the CD was such that it used polycarbonate coating with highly reflective coat, usually aluminium.
- In the CD-ROM the data stored as pits and lands as shown in Fig. 4.4.6(a).
- These pits and lands are read by reflecting laser. The CD has a constant packing density hence constant linear velocity across a track is required as against the constant angular velocity in case of magnetic discs.

- The Fig. 4.4.6(a) shows that the CD is made up of three layers, namely the polycarbonate plastic, aluminium and the protective material like acrylic. The laser beam incident on the highly reflective substance like aluminium, returns back in some amount of time. Based on this time gap, the optical disk reader can realize that there was a land or a pit. In case ifit is a land it will take more time to return while less time in case if it is a pit as seen in the Fig. 4.4.6(a).

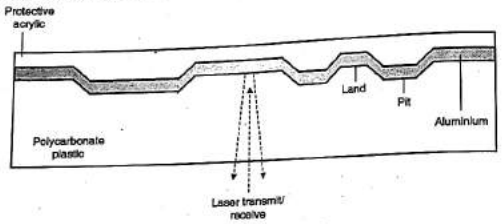



**Fig. 4.4.6(a) : Construction of CD**

- The data format on a CD-ROM is shown in the Fig. 4.4.6(b). Initially a data 00H is stored, followed by 10 bytes of FFH and again a 00H, called as the synchronous 12 bytes. The next is the 4 bytes ID (IDentity) about the time required for this data to be played (in MINutes and SEConds), the sector in which the data is placed and the mode. There are three modes,
  - Mode 0 indicates blank data field
  - Mode 1 indicates 2048 byte data + error correction
  - Mode 2 indicates 2336 byte data and no correction data
- Thus the remaining two fields contain data and error correction code (ECC) as defined by the mode bits.

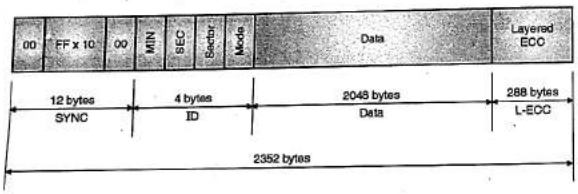
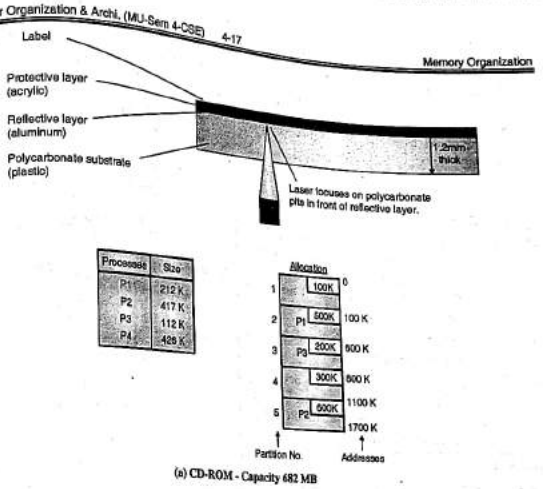



**Fig. 4.4.6(b) : Data Format on CD**

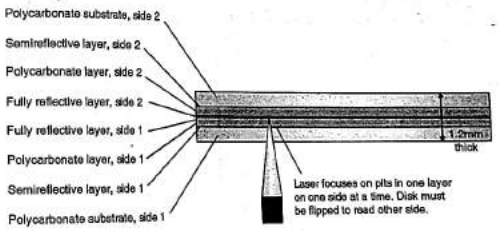
### ➜ B. DVD

- The major difference between a CD and DVD is that a DVD has multiple layers and hence very high capacity. Another major difference in a DVD with respect to CD is that the DVD has more denser data storage mechanism which results in the data storage capacity of around 4.7G per layer of a DVD.
- There are DVDs available with single layer as well as multiple layers.
- The Fig. 4.4.7 shows the constructional differences of a CD and DVD.






**(a) CD-ROM - Capacity 682 MB**




**(b) DVD-ROM, double-sided, dual-layer - Capacity 17 GB**

**Fig. 4.4.7 : Construction of (a) CD and (b) DVD**

- As seen in the Fig. 4.4.7(b), the double sided, two layers DVD, has a reflective and semi-reflective layers on both the sides. Hence in this case, the laser beam and receiver have to be on both the sides of the disc.
- Also there have to be two types of beam with low and high intensity, the low intensity beam is reflected by the semi-reflective substance, while the high intensity beam is reflected by the highly reflective substance. Thus giving a mechanism to read the data written on both the layers of each of the side.

**Syllabus Topic : Allocation Policies**

## 4.5 Allocation Policies

Q. Explain the different allocation policies. **(5 Marks)**

- Partitioning refers to logical division of the memory into subparts so that they can be accessed individually by tasks fragmentation generally happens when memory blocks have been allocated and are freed randomly.
- This results in splitting of partition memory into small non-continuous fragments.

There are 3 memory allocation policies :

→ **(1) Best fit :** In this case the smallest available fragment is searched and the required data is stored in that fragment. The smallest fragment searched for should be greater than of equal to the size of data to be stored.

→ **(2) Worst fit :** In this case the largest available block is used to store the data.

→ **(3) Next fit :** In this case immediate next empty block of a size equal to or greater than the size of data to be stored is searched sequentially and the required data is stored there.

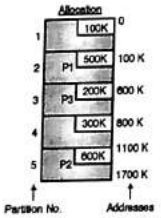
**Ex. 4.5.1 :**

Given the memory partitions of size 100 K, 500 K, 200 K, 300 K and 600 K (in order). How would each of the first fit, best-fit, worst fit algorithms place the processes of 212 K, 417 K, 112 K and 426 K (in order) ? Which algorithm makes the most efficient use of memory ?

**Soln. :**

**i] First-fit :**

| Processes | Size |
|---|---|
| P1 | 212 K |
| P2 | 417 K |
| P3 | 112 K |
| P4 | 426 K |




(ss.s)Fig. Ex. 4.5.1

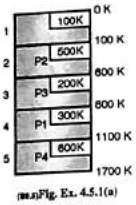
- Partition number 2 of size 500 K is assigned to P1 (size = 212 K). It is the first partition that can accommodate P1.
- Partition number 5 of size 600 K is assigned to P2 (size = 417 K). It is the first empty partition that can accommodate P2.
- P3 is assigned to partition 3.
- P4 cannot be executed.

$$\text{Memory utilization} = \frac{\text{Memory utilized}}{\text{Total memory}}$$

$$= \frac{\text{Memory utilized by P1, P2 and P3}}{\text{Total memory}}$$

$$= \frac{212\text{ K} + 417\text{ K} + 112\text{ K}}{1700\text{ K}} = \frac{741}{1700} = 0.436$$

**ii] Best-fit :**




(ss.s)Fig. Ex. 4.5.1(a)

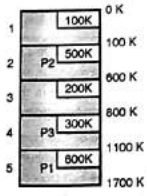
- Partition no. 4 of size 300 K is allocated to P1 (212 K). It is the smallest free partition that can accommodate P1.
- Partition no. 2 of size 500 K is allocated to P2 of size 417 K. It is the smallest free partition that can accommodate P2.
- Similarly, partition no.3 is allocated to P3 and the partition no.5 is allocated to P4.

$$\text{Memory utilization} = \frac{\text{Memory utilized by P1, P2, P3 and P4}}{\text{Total memory}}$$

$$= \frac{212\text{ K} + 417\text{ K} + 112\text{ K} + 426\text{ K}}{1700\text{ K}}$$

$$= \frac{1167\text{ K}}{1700\text{ K}} = 0.686$$

**Worst-Fit :**




(ss.s)Fig. Ex. 4.5.1(b)



- The largest free partition no.5 of size 600 K is allocated to P1 (212 K).
- P2 (size 417 K) is assigned to partition no.2. Partition no. 2 is the largest free partition and it can accommodate P2.
- P3 (size 112 K) is assigned to partition no.4. Partition no. 4 is the largest free partition.
- P4 cannot be executed as there is no free partition that can accommodate P4.

$$\text{Memory utilization} = \frac{\text{Memory utilized by P1, P2, P3}}{\text{Total memory}}$$

$$= \frac{212\text{ K} + 417\text{ K} + 112\text{ K}}{1700\text{ K}} = \frac{741}{1700} = 0.436$$

**Syllabus Topic : Cache Memory : Concept, Architecture (L1, L2, L3) and Cache Consistency**

## 4.6 Cache Memory : Concept, Architecture (L1, L2, L3) and Cache Consistency

→ **(MU - May 2014, Dec. 2014, May 2015, May 2016)**

Q. What are elements of cache design? Explain in details. May 14, 8 Marks

a. L1, L2 and L3 Cache memory. May 14, 7 Marks

b. Explain various high speed memories such as interleaved memories and caches. Dec. 14, 10 Marks

c. Describe what are the features of cache design? May 15, May 16, 8 Marks

Before going to the cache of Pentium processor, we will see some basics of the cache like its operation, advantage, principles of locality of reference, cache architectures, write policies etc.

### 4.6.1 Cache Operation

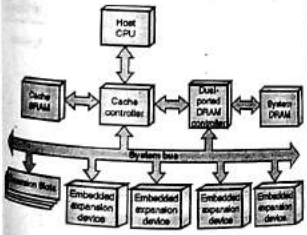



Fig. 4.6.1

1. Implementation of cache memory subsystem is an attempt to achieve almost all accesses with zero wait state while accessing memory, but with an acceptable system cost.
2. The cache controller maintains a directory to keep a track of the information and it has copied into the cache memory.
3. When the processor initiates a memory read bus cycle, the cache controller checks the directory to determine if it has a copy of the requested information in cache memory.
4. If the copy is present, the cache controller reads the information from the cache, sends it to the processors data bus, and asserts the processor's ready signal. This is called as **READ HIT**.
5. If the cache controller determines that it does not have a copy of the requested information in its cache, the information is now read from main memory (DRAM). This is known as **READ MISS** and causes wait states due to slow access time of DRAM.
6. The requested information is from the DRAM given to the processor. The information is also copied into the cache memory by cache controller and it updates its directory to track the information stored in cache memory.

Assume the cache memory is empty, in the beginning (after reset). The following sequence takes place :

1. The processor performs a memory read cycle to fetch the first instruction from memory.
2. The cache controller uses the address issued by the processor to determine if a copy of the requested information is already in the cache memory. But a cache miss occurs as the cache memory is empty.
3. The cache controller initiates a memory read cycle to fetch the requested information from DRAM memory. This will consume some wait states.
4. The information from DRAM memory is sent to the processor. It is also copied into the cache memory and the cache controller updates its directory to reflect the presence of the new information. The information being sent is not just the required instruction, but a block (line) of data is sent to the cache. No

performance gain is achieved due to absence of information in cache memory.

5. After executing the first instruction, the processor's prefetched requests a series of memory read bus cycles to fetch the remaining instructions in program loop. But since an entire line was brought earlier, most of the required instructions will be in cache and hence resulting in faster access and performance gain. If the cache is sufficiently large, all instructions in the program loop can become resident in cache memory.
6. The program has loop instruction to jump to the beginning of the loop start over again. The processor then requires the same program again.
7. When the processor requests for the first instruction in the loop, cache controller detects the presence of the instruction in the cache memory and hence provides it to the processor with zero wait states.

**Syllabus Topic : Locality of Reference**

### 4.6.2 Principles of Locality of Reference

**Q. What are the principles of locality of reference ? (5 Marks)**

1. **Locality of reference** is the term used to explain the characteristics of programs that run in relatively small loops in consecutive memory locations.
2. The locality of reference principle comprises of two components :

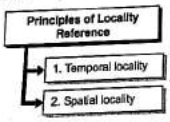



**Fig. 4.6.2 : Two components of locality of reference**

→ **(1) Temporal locality**

(a) Since the programs have loops, the same instructions are required frequently, i.e. the programs tend to use the most recently used information again and again.

(b) If for a long time a information in cache is not used, then it is less likely to be used again.

(c) This is known as the principle of temporal locality.

→ **(2) Spatial locality**

(a) Programs and the data accessed by the processor mostly reside in consecutive memory locations.

(b) This means that processor is likely to need code or data that are close to locations already accessed.

(c) This is known as the Principle of Spatial Locality.

3. The performance gains are realized by the use of cache memory subsystem are because of most of the memory accesses that require zero wait states due to principles of locality.

### 4.6.3 Cache Performance

1. Performance of cache subsystems depends on the frequency of cache hits, usually termed as **hit rate.**

$$\% \text{ HIT RATE} = \frac{\text{Cache Hits}}{\text{Total Memory Accesses}} \times 100\ \%$$

2. If a program requires a small area of memory and consists of loops, then maximum cache hits are possible.
3. On the other hand, if the program has non-looping code, many accesses will result in cache misses.
4. Fortunately, most programs run in loops and hence a high percentage of cache hit (85 – 95%) is experienced.
5. Besides locality of reference various other factors contribute to cache hit rates like Cache's architecture, Size of cache memory and Cache memory organization.

### 4.6.4 Cache Architectures

**Q. Write a short note on Look through and look aside cache architectures. (10 Marks)**

Two basic architectures are found in today's systems:

1. Look-through cache design
2. Look-aside cache design

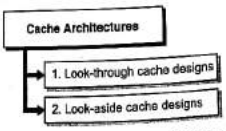



**Fig. 4.6.3 : Two basic cache architectures**



→ **1. Look-through cache designs**

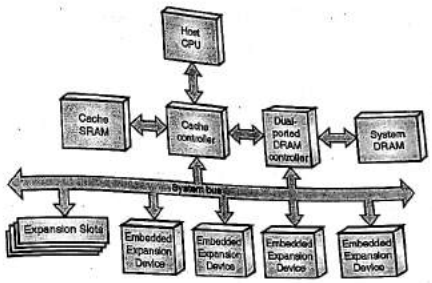



**Fig. 4.6.4**

(a) The performance of systems incorporating Look Through Cache is typically higher than that of systems incorporating Look Aside Cache.

(b) Data from main memory (DRAM) is not transferred to the processor using system bus hence system bus is free for other bus masters (like DMAC) to access the main memory.

(c) This system isolates the processor's local bus from the system bus hence achieving bus concurrency.

(d) The major advantage is that two bus masters can operate simultaneously. One processor accesses look through cache while another bus master such as DMA can access the system bus is possible.

(e) To expansion devices, a look-through cache controller is like a system processor.

(f) During memory writes, look-through cache provides zero wait state operation (using posted writes) for write misses.

☞ **Advantages**

(a) It reduces the system and memory bus utilization, leaving them available for use by other bus master.

(b) It allows bus concurrency, where both the processor and another bus master can perform bus cycles at the same time.

(c) It also completes write operations in zero wait states using posted writes.

☞ **Disadvantages**

(a) In the event that the memory request is a cache miss, the lookup process delays the request to memory. This delay is called as lookup penalty.

(b) It is more complex, costly and difficult to design and implement.

2. **Look-aside cache designs**

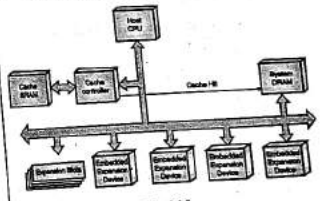



**Fig. 4.6.5**

(a) In this case the processor is directly connected to the system bus or memory bus.

(b) When the processor initiates a bus access, cache controller as well as main memory detects the bus access address.

(c) The cache controller sits aside and monitors each processor memory request to determine if the cache contains the copy of the requested information.

(d) If it is a cache hit, the cache controller terminates the bus cycle by instructing memory subsystem to ignore the request. If it is a cache miss, the bus cycle completes in normal fashion from memory (and wait states are required).

**Advantages**

(a) Cache miss cycles complete faster in Look Aside Cache as the bus cycle is already in progress to memory and hence no look up penalty is incurred.

(b) Simplicity of designs because only one address is to be monitored by cache controller form processor and not from I/O devices.

(c) Lower cost of implementation due to their simplicity.

**Disadvantages**

(a) The processor requires system bus utilization for its every access, to access both cache subsystem and memory.

(b) Concurrent operations are not possible as all masters reside on the same bus.

**Syllabus Topic : Cache Coherency**

## 4.6.5 Cache Consistency (Also Known as Cache Coherency)

→ (MU - Dec. 2014, May 2015, Dec. 2016)

Q. Explain in detail cache coherence. Dec. 14, May 15, Dec. 16, 5 Marks

Q. What is cache coherency ? (5 Marks)

1. In order to work properly for the cache subsystems, the CPU and the other bus masters must be getting the most updated copy of the requested information.
2. There are several cases wherein the data stored in cache or in main memory may be altered whereas the duplicate copy remains unchanged.

**Causes of cache consistency problems**

1. When the copy of line in cache, no longer matches the contents of line stored in memory, there is loss of cache consistency. It can be either due to cache line being updated while the memory line is not, or the memory line being updated while the cache line is not.
2. In each of these instances the stale data must be updated. It can be a result of cache write hit and hence the caches write policy has to handle this problem for the first case.
3. For the second case the coherency problem is due to some other bus master changing the data in memory. This change is to be updated in cache line by the cache controller, hence the cache controller has to monitor the system bus.

**Syllabus Topic : Write Policies**

## 4.6.6 Write Policy

Q. Explain different write policies. (10 Marks)

1. When the write hit occurs, the cache memory is updated and it contains the latest data while memory contains stale data.
2. Such a cache line is called as dirty or 'modified' because it has no longer mirrors of its corresponding line in memory.
3. In order to correct this cache consistency problem, the corresponding memory line must be updated to reflect the change made in the cache; else another bus master may get stale data if it reads from these lines.
4. Three write policies are used to prevent this type of consistency problem: Write-through, Buffered or posted write-through and Write-back.

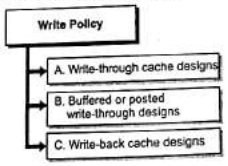



Fig. 4.6.6 : Write policy

### → A. Write-Through Cache Designs

1. In this write policy, the data is passed to the memory immediately, so that the memory has the updated data.
2. Even on write hit operation, the cache controller updates the line in the cache and the corresponding line in memory, and hence ensuring that consistency is maintained between cache and memory.
3. Very simple and effective implementation.
4. But poor performance due to slow main memory writes operation.
5. Also it doesn't allow bus concurrency.

### → B. Buffered or Posted Write-Through Designs

1. It has an advantage of providing zero wait state write operation for cache hits as well as cache misses.
2. When a write occurs, buffered write through caches tricks the processor into thinking that the information was written to memory in zero wait states. In fact, the write to main memory has not been performed yet.
3. The look-through cache controller stores the entire write operation in a buffer, and writes to the main memory later. Hence the processor need not perform slow write operation with wait states and hence doesn't impact processor's performance. This is assuming that the posted write buffer is only one transaction deep.
4. But, if there are two back-to-back memory write bus cycles, the cache controller will insert wait states into second bus cycle until the first write to memory has actually been completed. The bus controller will then post the second bus cycle and assert the processor's ready line. But since processor typically writes only one write operation, the memory writes are completed in zero wait states.
5. With this policy, another bus master is not permitted to use the bus until the write-through is completed, thereby ensuring that the bus master will receive the latest information from memory.
   - The write-through operations use either system or memory bus. Hence when write-through to memory is in progress, bus masters are prevented from accessing memory.
   - But actual cache consistency problem occurs only when the bus master reads from a location in memory that is stale. The frequency of this type of occurrence is very less. In fact, the memory line is likely to be updated many times by the processor before another bus master reads from that particular line.
   - As a result the write-through and buffered write-through designs, update memory each time a memory write is performed, although the need for such action may not be required immediately.

### → C. Write-Back Cache Designs

1. Write-back designs improve the overall system performance by updating a line in main memory only when necessary, thereby keeping the system bus free for use by other processors and bus masters and hence ensuring bus concurrency.
2. The memory is updated only when:
   (a) Another bus master initiates a read operation from a memory line that contains stale data.
   (b) Another bus master initiates a write operation from a memory line that contains stale data.
   (c) The cache line that contains modified information is about to be overwritten in order to store a line newly acquired from memory i.e. during line replacement.
3. Cache controller marks the cache lines as 'modified' in the cache directory when the processor updates them. Hence when read by another master or written into the memory, the cache subsystem checks whether it is marked as 'modified' in cache.
4. They design of such cache controller is COMPLICATED to implement because they must MAKE DECISIONS on when to write 'modified' lines back to memory to ensure consistency.

## 4.6.7 Bus Master/Cache Interaction for Cache Coherency

1. When another device in system uses the buses, it must become bus master.
2. In case of look-through cache design, the cache controller is requested for bus; while in case of look-aside cache design, the processor is requested for same. In both cases HOLD and HLDA logic is used.
3. In some cases, the request is to be given to bus arbiter like 8289.

Since bus masters can write to and read from memory, cache consistency problems may happen under three circumstances:

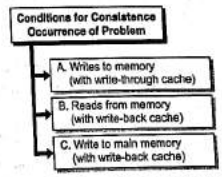



Fig. 4.6.7 : Conditions for consistence occurrence of problem

**A. Writes to memory (with write-through cache)**

1. When the bus masters write through memory, they update locations that may also be cached by the cache controller.
2. In these cases, memory is updated and the line in the cache becomes stale. Hence cache controller must monitor the memory writes to avoid this coherency problems. When the write is detected, the cache line is invalidated because it will contain stale data after the write to memory completes. Hence the cache controller has to monitor the system bus to find out what the other bus master is doing on the system bus. So that if another master is updating a line of the main memory, the cache has to invalidate this line. This monitoring of the system bus is called as snooping.

**B. Reads from memory (with write-back cache)**

1. When the bus master reads from memory in a system that has a write-back cache, it may read from a line containing stale data i.e. the location has been updated in cache but not in memory.
2. To detect this coherency problem, write-back caches must also snoop reads from memory.
3. The system can be designed to back-off the bus master and write the cache line to memory, before releasing back off and allowing the read continue.

**C. Write to main memory (with write-back cache)**

1. This problem occurs when another bus-master is performing a memory write to a line containing stale data. The bus master updates one or more locations in memory that are also contained within the cache.
2. Even if the cache line is not capable of data snarfing, it could invalidate that cache line, causing a mistake.
3. Since the line has been marked 'modified', it indicates that some or all of the information in the line is more current than the corresponding data in the memory.
4. The memory write being performed by another bus master will update some item within the memory line. By invalidating the line in cache it would quite probably discard some data that is more current than that within the memory line.
5. If the cache permits the bus master to complete the write, and then flushes the cache line to memory, the data just written by the bus master may be over-written by stale data in the cache line. The correct action would be to back-off the bus master, before it is able to complete the write to memory.
6. The cache controller then seizes the bus and performs a memory write to update this stale line in the main memory. In the cache directory, the cache line is now invalidated because the bus master will update the memory cache line immediately after the line is flushed. The cache then removes back-off signal, permitting the bus master to reinitiate the memory write operation. When the bus master completes the write to memory, the memory line will contain the most updated data.

### 4.6.8 Bus Snooping/Snarfing

**Q. List and explain different replacement policies. (5 Marks)**

1. Bus snooping is the method used by cache controller to monitor memory accesses performed by other bus master.
2. The cache controller monitors, or snoops the system bus when another bus master is performing an access to memory so as to maintain cache consistency.
3. A line that is accessed from memory and is also resident in the cache, it is referred to as snoop hit. In such cases either cache or memory line may contain stale data.



There are two possibilities that may create the need to snoop the bus:

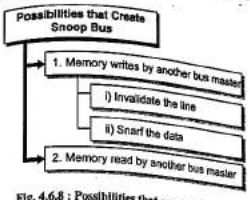



**Fig. 4.6.8 : Possibilities that create snoop bus**

**(1) Memory writes by another bus master**

(a) Assume that the cache controller has given the control of the system bus to the DMA controller so that it may transfer a block of data from a disk controller into memory.

(b) It is possible that the DMA controller will alter the contents of memory lines that have already been copied into the cache.

(c) Cache controllers handle this situation differently depending on how they are designed.

(i) **Invalidate the line** in cache that would otherwise contain stale data. The next time that this particular line is requested by the processor, it will result in cache miss, forcing cache subsystem to read from memory.

(ii) **Snarf the data :** The cache controller snoops the bus during another bus master's write operation and if a snoop hit is detected, it will capture (snarf) the data from the system bus while it's being written to memory by the bus master. In this way, both the memory and cache lines will have the updated data.

**(2) Memory read by another bus master**

(a) Only if the cache subsystems use write-back policy, they must snoop the system bus during memory reads initiated by other bus master.

(b) The cache controller snoops memory reads by another bus master to determine if the line being read from his being updated in cache, but memory has a stale data (i.e. the cached location is marked as 'modified').

(c) If yes, this would result in snoop read hit to a 'modified' line. Hence the cache controller must force the bus master attempting the memory read to suspend the bus cycle until it has updated memory.

(d) Once the memory line has been updated by the cache controller, the bus master is allowed to complete its memory read operation.

(e) Alternatively the cache controller may find a snoop hit and instruct the system memory not to supply the data and instead it will supply the data from the cache. 4.7.9 Replacement Algorithms

Replacement algorithm is required to replace a line from the cache memory with the new line as discussed earlier.

There are various replacement polices available. The widely used ones are LRU, FIFO, LFU and random as discussed below :

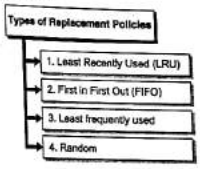



**Fig. 4.6.9 : Types of replacement polices**

1. **Least Recently used (LRU) :** In this case the line which is least recently used is replaced with the new line. Thus the line which has not been used for longest time is replaced with the new line.
2. **First in first out (FIFO) :** In this case the line which was brought into the cache first is replaced first. Thus the line which has stayed the longest in the cache is replaced.
3. **Least frequently used :** In this case the line which is used for the least number of times is replaced first.
4. **Random :** In this case randomly any line is replaced.

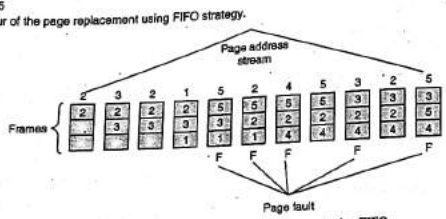
**Ex. 4.6.1**

Assume that memory consists of three frames and during execution of a program, following pages are referenced in the sequence :

2 3 2 1 5 2 4 5 3 2 5

Show that behaviour of the page replacement using FIFO strategy.

**Soln. :**




(ee s.44)**Fig. Ex. 4.6.1 : Behaviour of page replacement using FIFO**

**Ex. 4.6.2**

Find out page fault for following string using LRU and FIFO method.6 0 12 0 30 4 2 30 32 1 20 15

(Consider page frame size = 3)

**Soln. :**

Page address stream

| | 6 | 0 | 12 | 0 | 30 | 4 | 2 | 30 | 32 | 1 | 20 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| FIFO | 6 | 6 | 6 | 6 | 30 | 30 | 30 | 30 | 32 | 32 | 32 | 15 |
| | | 0 | 0 | 0 | 0 | 4 | 4 | 4 | 4 | 1 | 1 | 1 |
| | | | 12 | 12 | 12 | 12 | 2 | 2 | 2 | 2 | 20 | 20 |
| | | | | | F | F | F | | F | F | F | F |
| LRU | 6 | 6 | 6 | 6 | 30 | 30 | 30 | 30 | 30 | 30 | 20 | 20 |
| | | 0 | 0 | 0 | 0 | 0 | 2 | 2 | 2 | 1 | 1 | 1 |
| | | | 12 | 12 | 12 | 4 | 4 | 4 | 32 | 32 | 32 | 15 |
| | | | | | F | F | F | | F | F | F | F |

Page faults are indicated by 'F'.

**Ex. 4.6.3**

Consider a paging system in which M1 has a capacity of three frames. The page address stream formed by executing a program is

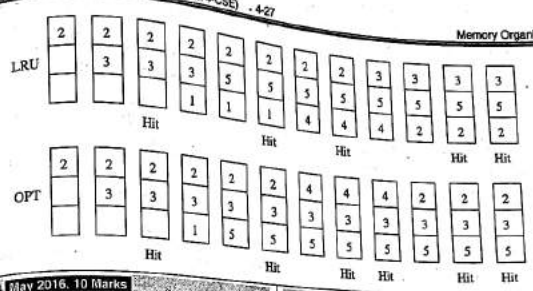
2 3 2 1 5 2 4 5 3 2 5 2

Find the page hit using FIFO, LRU and OPT.

**Soln. :**

| Time | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Address space | 2 | 3 | 2 | 1 | 5 | 2 | 4 | 5 | 3 | 2 | 5 | 2 |
| FIFO | 2 | 2 | 2 | 2 | 5 | 5 | 5 | 2 | 2 | 3 | 3 | 3 |
| | | 3 | 3 | 3 | 3 | 2 | 2 | 2 | 2 | 2 | 5 | 5 |
| | | | 2 | 1 | 1 | 1 | 4 | 4 | 4 | 4 | 4 | 2 |
| | | | Hit | | | | | Hit | | Hit | | |






**Ex. 4.6.4** **May 2016. 10 Marks**

Find out page fault for following string using LRU method. Consider page frame size = 3

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

**Soln. :**

| Pages accessed | Frames | | | |
|---|---|---|---|---|
| 7 | 7 | – | – | |
| 0 | 7 | 0 | – | |
| 1 | 7 | 0 | 1 | |
| 2 | 2 | 0 | 1 | F |
| 0 | 2 | 0 | 1 | |
| 3 | 2 | 0 | 3 | F |
| 0 | 2 | 0 | 3 | |
| 4 | 4 | 0 | 3 | F |
| 2 | 4 | 0 | 2 | F |
| 3 | 4 | 3 | 2 | F |
| 0 | 0 | 3 | 2 | F |
| 3 | 0 | 3 | 2 | |
| 2 | 0 | 3 | 2 | |
| 1 | 1 | 3 | 2 | F |
| 2 | 1 | 3 | 2 | |
| 0 | 1 | 0 | 2 | F |
| 1 | 1 | 0 | 2 | |
| 7 | 1 | 0 | 7 | F |
| 0 | 1 | 0 | 7 | |
| 1 | 1 | 0 | 7 | |

**Ex. 4.6.5** **Dec. 2015, Dec. 2016. 10 Marks**

Calculate the hit and miss using various page replacement policies LRU, OPT, FIFO for following sequence (page frame size 3) 4, 7, 3, 0, 1, 7, 3, 8, 5, 4, 5, 3, 4, 7. State which one is best for above example ?

**Soln. :**

**LRU**

| | 4 | 7 | 3 | 0 | 1 | 7 | 3 | 8 | 5 | 4 | 5 | 3 | 4 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4 | 4 | 4 | 0 | 0 | 0 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 |
| | | 7 | 7 | 7 | 1 | 1 | 1 | 8 | 8 | 8 | 8 | 3 | 3 | 3 |
| | | | | 3 | 3 | 3 | 7 | 7 | 7 | 5 | 5 | 5 | 5 | 7 |
| | M | M | M | M | M | M | M | M | M | M | H | M | H | H |

$$\% \text{ Hit} = \frac{2}{14} \times 100 = 14.3\ \%$$

$$\% \text{ Miss} = \frac{12}{14} \times 100 = 85.7\ \%$$

**(II) FIFO**

| | 4 | 7 | 3 | 0 | 1 | 7 | 3 | 8 | 5 | 4 | 5 | 3 | 4 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4 | 4 | 4 | 0 | 0 | 0 | 3 | 3 | 3 | 4 | 4 | 4 | 4 | 4 |
| | | 7 | 7 | 7 | 1 | 1 | 1 | 8 | 8 | 8 | 8 | 3 | 3 | 3 |
| | | | 3 | 3 | 3 | 7 | 7 | 7 | 5 | 5 | 5 | 5 | 5 | 7 |
| | | | | | | | | | | | | | H | H |

$$\% \text{ Hit} = \frac{2}{14} \times 100 = 14.3\ \%$$

$$\% \text{ Miss} = \frac{12}{4} \times 100 = 85.7\ \%$$

**(III) OPT**

| | 4 | 7 | 3 | 0 | 1 | 7 | 3 | 8 | 5 | 4 | 5 | 3 | 4 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 4 | 4 | 4 | 0 | 1 | 1 | 1 | 1 | 5 | 5 | 5 | 5 | 5 | 5 |
| | | 7 | 7 | 7 | 7 | 7 | 7 | 8 | 8 | 4 | 4 | 4 | 4 | 4 |
| | | | | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 7 |
| | | | | | | | | | | | H | H | | |

$$\% \text{ Hit} = \frac{5}{14} \times 100 = 35.7\ \%$$

$$\% \text{ Miss} = \frac{9}{14} \times 100 = 64.3\ \%$$

**Syllabus Topic : Memory Hierarchy : Cost and Performance Measurement**

### 4.6.9 Cost and Performance Measurement of Two Level Memory Hierarchy

→ (MU - Dec. 2014)

Q. Explain LRU page replacement policy with suitable example. Dec. 14, 10 Marks

Q. List and explain the different performance characteristics of two level memory. (5 Marks)

- Any two level memory has to be analysed with its performance characteristics as per the following set of characteristics.
- The different group of two level memories can be cache memory and main memory, main memory and virtual memory, internal and external cache memory etc.
- Let us see the various parameters to be considered during the performance analysis.

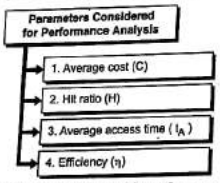



**Fig. 4.6.10 : Parameters considered for performance analysis**

→ **1. Average cost (C)** $= \frac{C1S1 + C2S2}{S1 + S2}$

where, C1 and C2 are the costs per bit of memory 1 (faster memory) and memory 2 (slower memory) respectively.

S1 and S2 are the sizes of memory 1 and memory 2 respectively.

→ **2. Hit Ratio (H)** $= \frac{N1}{N1 + N2}$

where N1 is number of hits and N2 is number of misses.

→ **3. Average access time** $( t_A ) = H\, t_{A1} + (1 - H)\, t_{A2}$

where $t_{A1}$ and $t_{A2}$ are the time taken to access memory 1 (faster memory) and memory 2 (slower memory) respectively.

$$t_A = H\, t_{A1} + (1 - H)\, t_{A2}$$
$$= H\, t_{A1} + (1 - H)\, (t_{A1} + t_B)$$

where $t_{A2} = t_{A1} + t_B = t_{A1} + (1 - H)\, t_B$

→ **4. Efficiency** $(\eta) = \frac{t_{A1}}{t_A}$

$$= \frac{t_{A1}}{H\, t_{A1} + (1 - H)\, t_{A2}} = \frac{1}{H + (1 - H)\, r}$$

where $r = \frac{t_{A2}}{t_{A1}}$ = **Speed Ratio**

**Syllabus Topic : Mapping Techniques**

## 4.7 Cache Mapping Techniques

- Mapping Function and replacement algorithm together decides where a line from the main memory can reside in the cache.
- The different mapping functions are Direct mapping, Fully Associative mapping and Set associative mapping.

### 4.7.1 Direct Mapping Technique

Q. Write a short note on Direct mapping technique. (5 Marks)

- In this case each block of main memory can map to only one cache line.
- A given block maps to any **line (i mod j)**, where i is the line number of the main memory to be mapped and j is the total number of lines in the cache memory.
- The address is divided into three parts i.e. the word selector, line selector and the tag.
- Least Significant w bits identify unique word of a particular line
- Most Significant s bits specify one memory block to which the cache line corresponds.
- The MSBs are split into a cache line field r and a tag of s-r (most significant)
- **Example**: Let Cache be of 64kByte that is divided into blocks of 4 bytes hence cache is 16k ($2^{14}$) lines of 4 bytes. And let the main memory size be 16MBytes that requires 24 bit address lines ($2^{24}$=16M).

| Tag (s - r) (8 bits) | Line (r) (14 bits) | Word (w) (2 bits) |
|---|---|---|

- Hence the 24 bit address is divided as 2 bit word identifier (4 byte block), 22 bit block identifier i.e. 8 bit tag 14 bit slot or line(16K lines=$2^{14}$)



- Fig. 4.7.1 shows the organization of Direct Mapping Cache.
- Fig. 4.7.1 shows the method the access the data from the cache implementing direct mapping technique

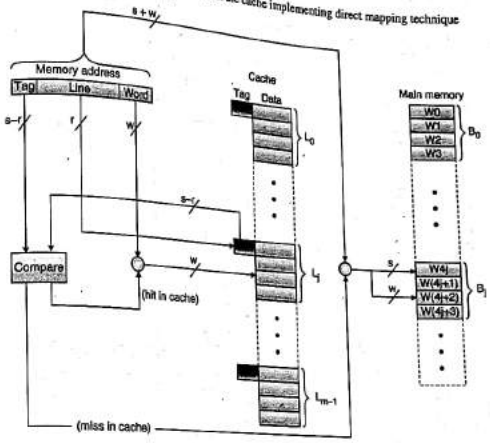



**Fig. 4.7.1**

- In this case to search a line from the cache memory, the line field selects the particular line, whose tag is to be compared with the tag of the address specified by the processor.
- The advantages of Direct Mapping are:
  1. Simple implementation
  2. Inexpensive
- The disadvantages of Direct mapping are :
  1. Fixed location for given block hence if a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high.

### 4.7.2 Fully Associative Mapping

Q. Explain fully associative mapping technique. (5 Marks)

- In this case a main memory block can load into any line of cache.
- There are only two fields in the address as tag and word
- The tag uniquely identifies block of memory from where the line has been copied into the cache memory.
- To search a particular data the tag of every line is to be examined for a match. Thus cache searching gets expensive in terms of time required.
- **Example**: Let Cache be of 64k Byte that is divided into blocks of 4 bytes hence cache is 16k ($2^{14}$) lines of 4 bytes. And let the main memory size be 16M Bytes that requires 24 bit address lines ($2^{24}$ = 16M).

- The associative Mapping Address Structure for this example considered: 22 bit tag stored with each 4-word block of data.
- Compare tag field with tag entry in cache to check for hit. Least significant 2 bits of address identify which word is required from 4-word data block

| Tag (22 bits) | Word (2 bits) |
|---|---|

- The organization of Fully Associative Cache mapping is shown in the Fig. 4.7.2.

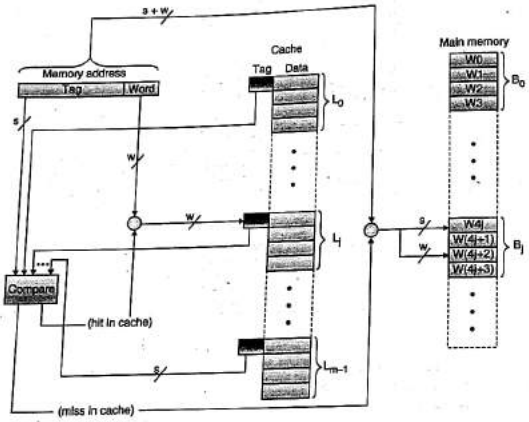



Fig. 4.7.2

- The advantages of Associative Mapping are
  1. If a program accesses 2 blocks (that would map to the same line in case of Direct mapping) repeatedly, cache misses will not occur
- The disadvantages of Associative Mapping are
  1. Complex design for many parallel comparisons of tag.
  2. Expensive due to implementation of parallel comparator.

### 4.7.3 Set Associative Mapping

**Q. Explain with example two way set associative mapping technique. (5 Marks)**

- In this case cache is divided into a number of sets. Each set contains a number of lines.
- A given block maps to any **line in a given set (i mod j)**, where i is the line number of the main memory to be mapped and j is the total number of sets in the cache memory.
- For example, if there are 2 lines per set, it is called as 2 way associative mapping i.e. a given block can be in one of 2 lines in only one set.



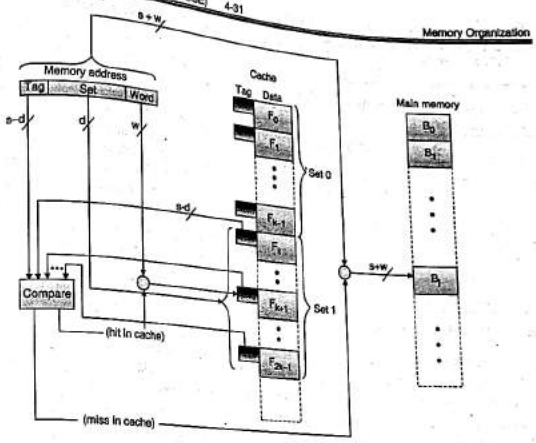



Fig. 4.7.3

- **Example:** Let Cache be of 64kByte that is divided into blocks of 4 bytes hence cache is 16k ($2^{14}$) lines of 4 bytes. And let the main memory size be 16MBytes that requires 24 bit address lines ($2^{24}$=16M).
- For this example for set associative mapping address structure: 2 bits for one of the 4 words, 8K lines in each of the 2 sets hence 13 bits to select a set ($2^{13}$=8K) and remaining (24 – 13 – 2=) 9 bits for tag.

| Tag (9 bits) | Set (13 bits) | Word (2 bits) |
|---|---|---|

- In this case the set field is used to determine cache set to look in and Compare tag field to see if we have a hit.
- Fig. 4.7.3 shows an example of Two Way Set Associative Cache Organization
- The advantages of Set Associative Mapping are:
  1. If a program accesses 2 blocks that map to the same set repeatedly, cache misses will not occur because they would go into different lines of the set.
  2. Not very complex because of just 2, 4 or 8 parallel comparisons.
  3. Not much expensive again because of simple implementation.

**Ex. 4.7.1 May 2016, 10 Marks**

A block set associative cache consists of 64 blocks divided in 4 block sets. The main memory contains 4096 blocks, each 128 words of 16 bit length

(1) How many bits are there in main memory address ?

(2) How many bits are there in cache memory address (tag, set and word fields) ?

**Soln. :**

(1) Main memory size = 4096 blocks x 128 word = $2^{12} \times 2^7 = 2^{19}$

Thus main memory address lines required is equal to 19.

(2) Cache memory has 64 blocks divided in 4 block sets, thus each set has 16 blocks. Hence $16 = 2^4$; 4 address lines for set

Each block has 128 words; hence $128 = 2^7$, 7 address lines for word field

Remaining lines i.e. 19 – 4 – 7 = 8 address lines for tag

| Tag (7 bits) | Set (4 bits) | Word (7 bits) |
|---|---|---|

**Syllabus Topic : Interleaved and Associative Memory**

## 4.8 Interleaved and Associative Memory

→ **(MU - May 2015,Dec. 2015,May 2016)**

Q. What is Associative memory ? **May 15. 4 Marks**

Q. Explain set associative and associative cache mapping techniques. **Dec. 15. 10 Marks**

Q. Explain the interleaved memory. **May 16. 10 Marks**

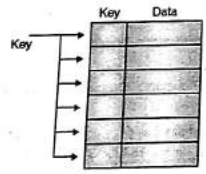
Q. Write short notes on interleaved memory and associative memory. **(5 Marks)**

### 4.8.1 Associative Memory

- In associative memory any stored item can be accessed by using the contents of the item. The subfield chosen to address the memory is called the key. Items stored in an associative memory can be viewed as having the two field format :

  KEY, DATA

  Where KEY is the stored address and DATA is the information to be accessed.
- Associative searching is based on simultaneous matching of the key to be searched with the stored key associated with each line of data. A word is retrieved based on a portion of its contents rather than its address.




(co 5.42)**Fig. 4.8.1 : A key to be searched is matched simultaneously**

### 4.8.2 Interleaved Memory

- Interleaved memory implements the concept of accessing more words in single memory access cycle. Memory can be partitioned into N separate memory modules. Thus N accesses can be carried out to the memory simultaneously.
- Once presented with a memory address, each memory module returns one word per cycle. It is possible to present different addresses to different memory modules so that parallel access to multiple words can be done simultaneously or in a pipelined fashion.
- The maximum processor bandwidth in interleaved memory can be equal to the number of modules i.e. N words per cycle.
- To achieve the address interleaving consecutive addresses are distributed among the N interleaved modules. For example, if we have consecutive addresses and 4 interleaved memory modules then $0^{th}$, $4^{th}$, $8^{th}$, ... addresses will be assigned to the first memory module and so on.

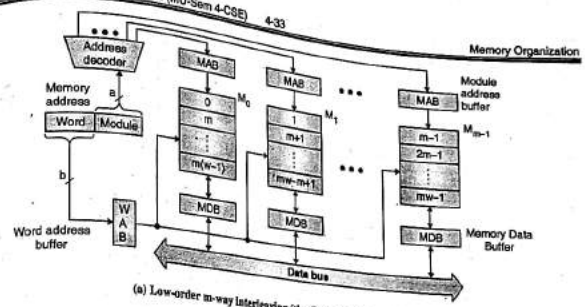
  0, 4, 8, 12, 16 ... Addresses to memory module 0

  1, 5, 9, 13, 17 ... Addresses to memory module 1

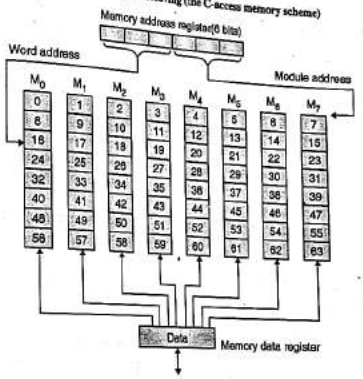
  2, 6, 10, 14, 18 ... Addresses to memory module 2

  3, 7, 11, 15, 19 ... Addresses to memory module 3
- Consider a main memory formed with $m = 2^a$ memory modules, each containing $w = 2^b$ words of memory cells. The total memory capacity is $m \cdot w = 2^{a+b}$ words. Fig. 4.8.2(a) shows memory format for memory interleaving.
- Interleaving spreads contiguous memory locations across m modules horizontally. This implies that the low-order a bits of the memory address are used to identify the memory module.
- The high-order b bits are used to address a word inside a module. Same word address is applied to all memory modules simultaneously. A module address decoder is used to distribute module addresses.
- Access of the m modules can be overlapped in a pipelined fashion. For this purpose, the memory cycle is subdivided into m sub cycles. An eight-way interleaved memory is shown in Fig. 4.8.2(b).






(a) Low-order m-way interleaving (the C-access memory scheme)




(b) Eight-way low-order interleaving (absolute address shown in each memory word)

**Fig. 4.8.2 : Interleaved Memory**

**Syllabus Topic : Virtual Memory : Concept, Segmentation and Paging**

## 4.9 Virtual Memory

→ **(MU - May 2014, Dec. 2014, May 2015, Dec. 2015, May 2017)**

Q. What is virtual memory ? **May 14. Dec. 15. 4 Marks**

Q. Explain virtual memory with reference to memory hierarchy, segments and pages. **Dec. 14. May 15. 10 Marks**

Q. What is TLB ? Explain working of TLB. **Dec. 15. 10 Marks**

Q. What is Segmentation ? **May 17. May 17. 5 Marks**

- Virtual memory is a concept wherein the applications are made to feel that a huge main memory (fast semiconductor memory) is interfaced to the processor, whereas actually a small amount of main memory and a huge external memory (typically slow ROM like magnetic disk) is interfaced.

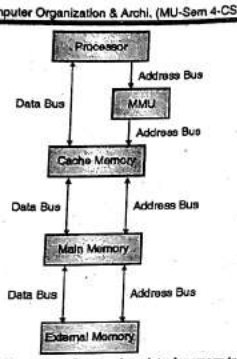



**Fig. 4.9.1 : Connection of external or virtual memory to the processor**

- The data required by the application is brought from the external slow memory to main memory in blocks (also called as pages) by the mechanism called as Paging.
- Hence now we can say that the entire memory system interfaced to the system looks something as shown in Fig. 4.9.1. The CPU or the processor is connected to the fast memory i.e. cache memory or SRAM which is then connected to the main memory or DRAM and then to the virtual memory or the external memory.
- The memory management unit (MMU) connected to the processor converts the virtual address to the physical address and take care of bringing the pages (block of data) to the main memory from the external memory.

### 4.9.1 Paging Mechanism or the Memory Management Unit

**Q. Explain the paging mechanism. (5 Marks)**
**Q. What is the use of Translational look aside buffer ? (5 Marks)**

- The memory management unit or the paging unit is responsible to convert the virtual or the linear address to physical address.
- Fig. 4.9.2 shows how the address translation takes place.
- The address given by the processor i.e. the linear or virtual address, is broken into the page number and the word number in that page.

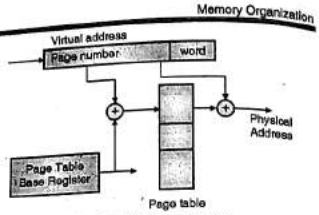



**Fig. 4.9.2 : Paging mechanism**

- The page number is checked for its presence in the main memory by comparing the entry for each page in the page table. If the page is present the word required is read from the page in the main memory.
- If the page required by the processor is not in the main memory, the page fault (similar to cache miss) occurs and the required page is loaded into the main memory by a special routine called as page fault routine. This technique is called as **Demand Paging** i.e. the page is brought from the external memory to the main memory only when required.
- A Translational Look aside Buffer (TLB) is implemented in the memory management system, which reduces the memory access time, by translating the linear to physical address without undergoing the paging mechanism.
- The structure of memory management with TLB is as shown in Fig. 4.9.3.

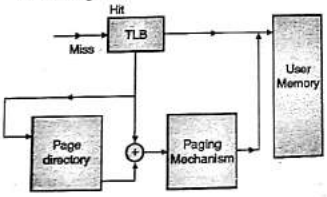



**Fig. 4.9.3 : Translation Look aside Buffer**

- As shown in the Fig. 4.9.3, TLB is placed parallel with the paging mechanism and hence if the TLB gives a hit, the paging mechanism doesn't perform the address translation, else the paging mechanism performs the address translation.

### 4.9.2 Segmentation

Segmentation refers to logical division of the main memory so as to give modular storage mechanism and multitasking.



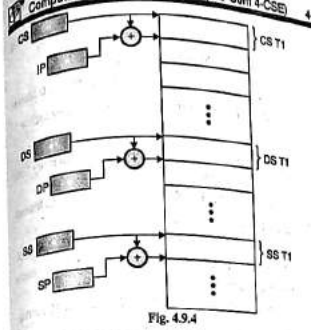



**Fig. 4.9.4**

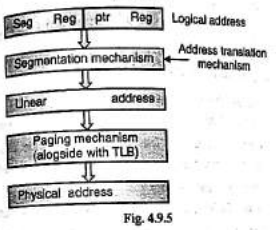



**Fig. 4.9.5**

## 4.10 Exam Pack (University and Review Questions)

**Syllabus Topic : Characteristics of memory**

Q. What are characteristics of memory devices? *(Ans. : Refer section 4.1)* **(May 2014, 8 Marks)**

Q. Describe the characteristics of Memory. *(Ans. : Refer section 4.1)* **(May 2016, Dec. 2016, 10 Marks)**

Q. List and explain the characteristics of memory. *(Ans. : Refer section 4.1)* **(5 Marks)**

**Syllabus Topic : Classifications of Primary and Secondary Memories**

Q. Explain in details Memory Hierarchy with examples. *(Ans. : Refer section 4.2)* **(May 2014, 6 Marks)**

Q. Explain memory hierarchy. *(Ans. : Refer section 4.2)* **(5 Marks)**

**Syllabus Topic : Types of RAM (SRAM, DRAM, SDRAM, DDR, SSD)**

Q. Compare SRAM and DRAM. *(Ans. : Refer section 4.3.1)* **(5 Marks)**

Q. Interface 8 KB EPROM and 4 KB RAM to a processor with 16-bit address and 8-bit data bus. *(Ans. : Refer Example 4.3.3)* **(5 Marks)**

**Syllabus Topic : Types of ROM**

Q. Write a short notes on Types of ROM. *(Ans. : Refer section 4.4.1)* **(Dec. 2016, 5 Marks)**

Q. Explain various types of ROM : Magnetic as well as optical. *(Ans. : Refer section 4.4.1)* **(5 Marks)**

**Syllabus Topic : Allocation Policies**

Q. Explain the different allocation policies. *(Ans. : Refer section 4.5)* **(5 Marks)**

**Syllabus Topic : Cache Memory : Concept, Architecture (L1, L2, L3) and Cache Consistency**

Q. What are elements of cache design? Explain in details. *(Ans. : Refer section 4.6)* **(May 2014, 8 Marks)**

Q. L1, L2 and L3 Cache memory. *(Ans. : Refer section 4.6)* **(May 2014, 7 Marks)**

Q. Explain various high speed memories such as interleaved memories and caches. *(Ans. : Refer section 4.6)* **(Dec. 2014, 10 Marks)**

Q. Describe what are the features of cache design ? *(Ans. : Refer section 4.6)* **(May 2015, 8 Marks)**

Q. What are the features of cache memory design? *(Ans. : Refer section 4.6)* **(May 2016, 10 Marks)**

**Syllabus Topic : Locality of Reference**

Q. What are the principles of locality of reference ? *(Ans. : Refer section 4.6.2)* **(5 Marks)**

Q. Calculate number of page faults and page hits for the page replacement policies FIFO, Optimal and LRU for given reference string, 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1 (assuming three frame size). *(Ans. : Refer example 4.6.4)* **(May 2016, 10 Marks)**

Q. Write a short note on Look through and look aside cache architectures. *(Ans. : Refer section 4.6.4)* **(10 Marks)**

**Syllabus Topic : Cache Coherency**

Q. Explain in detail cache coherence. *(Ans. : Refer section 4.6.5)* **(Dec. 2014, 5 Marks)**

Q. Explain in details Cache Coherency. *(Ans. : Refer section 4.6.5)* **(May 2015, 7 Marks)**

**Q.** Calculate the hit and miss using various page replacement policies LRU, OPT, FIFO for following sequence (page frame size 3) 4, 7, 3, 0, 1, 7, 3, 8, 5, 4, 5, 3, 4, 7. State which one is best for above example ?
*(Ans. : Refer example 4.6.5)* **(Dec. 2015, 10 Marks)**

**Q.** Calculate the hit and miss using various page replacement policies LRU, OPTIMAL, FIFO for following sequence (page frame size = 3) 4, 7, 3, 0, 1, 7, 3, 8, 5, 4, 5, 3, 4, 7. State which one is best for above example ?
*(Ans. : Refer example 4.6.5)* **(Dec. 2016, 10 Marks)**

**Q.** Cache Coherency.
*(Ans. : Refer section 4.6.5)* **(Dec. 2016, 5 Marks)**

**Q.** What is cache coherency ?
*(Ans. : Refer section 4.6.5)* **(5 Marks)**

### Syllabus Topic : Write Policies

**Q.** Explain different write policies.
*(Ans. : Refer section 4.6.6)* **(10 Marks)**

**Q.** List and explain different replacement policies.
*(Ans. : Refer section 4.6.8)* **(5 Marks)**

### Syllabus Topic : Memory hierarchy: cost and performance measurement

**Q.** Explain LRU page replacement policy with suitable example.
*(Ans. : Refer section 4.6.9)* **(Dec. 2014, 10 Marks)**

**Q.** List and explain the different performance characteristics of two level memory.
*(Ans. : Refer section 4.6.9)* **(5 Marks)**

### Syllabus Topic : Mapping Techniques

**Q.** Write a short note on Direct mapping technique.
*(Ans. : Refer section 4.7.1)* **(5 Marks)**

**Q.** Explain fully associative mapping technique.
*(Ans. : Refer section 4.7.2)* **(5 Marks)**

**Q.** Explain with example two way set associative mapping technique. *(Ans. : Refer section 4.7.3)* **(5 Marks)**

**Q.** A block set associative cache consists of 64 blocks divided in 4 block sets. The main memory contains 4096 blocks, each 128 words of 16 bit length

(1) How many bits are there in main memory address ?

(2) How many bits are there in cache memory address (tag, set and word fields) ?
*(Ans. : Refer example 4.7.3.1)* **(May 2016, 10 Marks)**

### Syllabus Topic : Interleaved and Associative Memory

**Q.** Write short notes on interleaved memory and associative memory.
*(Ans. : Refer section 4.8)* **(5 Marks)**

**Q.** What is Associative memory ?
*(Ans. : Refer section 4.8.1)* **(May 2015, 4 Marks)**

**Q.** Explain set associative and associative cache mapping techniques.
*(Ans. : Refer section 4.8.1)* **(Dec. 2015, 10 Marks)**

**Q.** Explain the interleaved memory.
*(Ans. : Refer section 4.8.2)* **(May 2016, 10 Marks)**

### Syllabus Topic : Virtual Memory: Concept, Segmentation and Paging

**Q.** Explain the paging mechanism.
*(Ans. : Refer section 4.9.1)* **(5 Marks)**

**Q.** What is the use of Translational look aside buffer ?
*(Ans. : Refer section 4.9.1)* **(5 Marks)**

**Q.** What is virtual memory ?
*(Ans. : Refer section 4.9)* **(May 2014, 4 Marks)**

**Q.** Explain virtual memory with reference to memory hierarchy, segments and pages.
*(Ans. : Refer section 4.9)* **(Dec. 2014, 10 Marks)**

**Q.** Explain in details Virtual Memory, Segmentation and Paging.
*(Ans. : Refer section 4.9)* **(May 2015, 7 Marks)**

**Q.** What is virtual memory ?
*(Ans. : Refer section 4.9)* **(Dec. 2015, 5 Marks)**

**Q.** What is TLB? Explain working of TLB.
*(Ans. : Refer section 4.9.1)* **(Dec. 2015, 10 Marks)**

**Q.** Explain Virtual Memory.
*(Ans. : Refer section 4.9)* **(May 2017, 5 Marks)**

**Q.** What is TLB ?
*(Ans. : Refer section 4.9.1)* **(May 2017, 8 Marks)**

**Q.** What is Segmentation ?
*(Ans. : Refer section 4.9.2)* **(May 2017, 5 Marks)**

□□□

CHAPTER 5

Module V

# I/O Organization and Peripherals

**Syllabus**

Common I/O device types and characteristics, Types of data transfer techniques : Programmed I/O, Interrupt driven I/O and DMA, Introduction to buses, Bus arbitration and multiple bus hierarchy, Interrupt types, Interrupts handling.

### Syllabus Topic : Common Input/Output Device Types and Characteristics

## 5.1 Input / Output System

**Q.** Explain the need of I/O module. **(5 Marks)**

- There are a wide variety of peripherals or I/O devices that deliver different amounts of data at different speeds and in different formats. All these devices are slower than CPU and RAM and hence to interface these devices to the CPU there is a need of I/O modules.
- Input/output module is interface to CPU and memory with one or more peripherals.
- The general model of I/O-module interfacing with system bus is shown in Fig. 5.1.1.

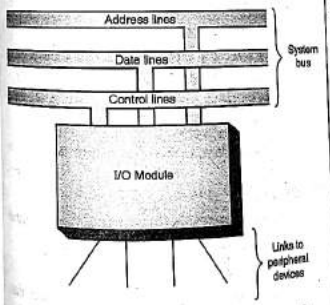



**Fig. 5.1.1 : General model of I/O module interface**

- The various functions of the I/O module involve :
  1. Issue of control and timing signals
  2. Communication with CPU
  3. Communication with peripheral
  4. Buffering of data between the CPU and peripheral and
  5. Detection of errors
- The internal block diagram of I/O module is shown in Fig. 5.1.2.

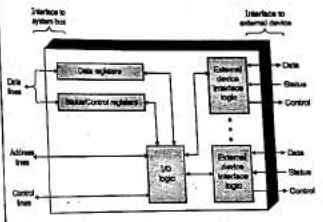



**Fig. 5.1.2 : Internal block diagram of an I/O module**

### 5.1.1 Parallel vs. Serial Interface

- The word, ***communication*** specifies, data transfer between two points.
- The data may be a digital or analog in nature.
- We will consider only digital data transfer because microprocessor is digital circuit. Suppose you want to transfer data from Point A to Point B. There are two possible ways of doing it :

(1) Parallel data transfer

**Q.** Calculate the hit and miss using various page replacement policies LRU, OPT, FIFO for following sequence (page frame size 3) 4, 7, 3, 0, 1, 7, 3, 8, 5, 4, 5, 3, 4, 7. State which one is best for above example ?
*(Ans. : Refer example 4.6.5)* **(Dec. 2015, 10 Marks)**

**Q.** Calculate the hit and miss using various page replacement policies LRU, OPTIMAL, FIFO for following sequence (page frame size = 3) 4, 7, 3, 0, 1, 7, 3, 8, 5, 4, 5, 3, 4, 7. State which one is best for above example ?
*(Ans. : Refer example 4.6.5)* **(Dec. 2016, 10 Marks)**

**Q.** Cache Coherency.
*(Ans. : Refer section 4.6.5)* **(Dec. 2016, 5 Marks)**

**Q.** What is cache coherency ?
*(Ans. : Refer section 4.6.5)* **(5 Marks)**

### Syllabus Topic : Write Policies

**Q.** Explain different write policies.
*(Ans. : Refer section 4.6.6)* **(10 Marks)**

**Q.** List and explain different replacement policies.
*(Ans. : Refer section 4.6.8)* **(5 Marks)**

### Syllabus Topic : Memory hierarchy: cost and performance measurement

**Q.** Explain LRU page replacement policy with suitable example.
*(Ans. : Refer section 4.6.9)* **(Dec. 2014, 10 Marks)**

**Q.** List and explain the different performance characteristics of two level memory.
*(Ans. : Refer section 4.6.9)* **(5 Marks)**

### Syllabus Topic : Mapping Techniques

**Q.** Write a short note on Direct mapping technique.
*(Ans. : Refer section 4.7.1)* **(5 Marks)**

**Q.** Explain fully associative mapping technique.
*(Ans. : Refer section 4.7.2)* **(5 Marks)**

**Q.** Explain with example two way set associative mapping technique. *(Ans. : Refer section 4.7.3)* **(5 Marks)**

**Q.** A block set associative cache consists of 64 blocks divided in 4 block sets. The main memory contains 4096 blocks, each 128 words of 16 bit length
(1) How many bits are there in main memory address ?
(2) How many bits are there in cache memory address (tag, set and word fields) ?
*(Ans. : Refer example 4.7.3.1)* **(May 2016, 10 Marks)**

### Syllabus Topic : Interleaved and Associative Memory

**Q.** Write short notes on interleaved memory and associative memory.
*(Ans. : Refer section 4.8)* **(5 Marks)**

**Q.** What is Associative memory ?
*(Ans. : Refer section 4.8.1)* **(May 2015, 4 Marks)**

**Q.** Explain set associative and associative cache mapping techniques.
*(Ans. : Refer section 4.8.1)* **(Dec. 2015, 10 Marks)**

**Q.** Explain the Interleaved memory.
*(Ans. : Refer section 4.8.2)* **(May 2016, 10 Marks)**

### Syllabus Topic : Virtual Memory: Concept, Segmentation and Paging

**Q.** Explain the paging mechanism.
*(Ans. : Refer section 4.9.1)* **(5 Marks)**

**Q.** What is the use of Translational look aside buffer ?
*(Ans. : Refer section 4.9.1)* **(5 Marks)**

**Q.** What is virtual memory ?
*(Ans. : Refer section 4.9)* **(May 2014, 4 Marks)**

**Q.** Explain virtual memory with reference to memory hierarchy, segments and pages.
*(Ans. : Refer section 4.9)* **(Dec. 2014, 10 Marks)**

**Q.** Explain in details Virtual Memory, Segmentation and Paging.
*(Ans. : Refer section 4.9)* **(May 2015, 7 Marks)**

**Q.** What is virtual memory ?
*(Ans. : Refer section 4.9)* **(Dec. 2015, 5 Marks)**

**Q.** What is TLB? Explain working of TLB.
*(Ans. : Refer section 4.9.1)* **(Dec. 2015, 10 Marks)**

**Q.** Explain Virtual Memory.
*(Ans. : Refer section 4.9)* **(May 2017, 5 Marks)**

**Q.** What is TLB ?
*(Ans. : Refer section 4.9.1)* **(May 2017, 8 Marks)**

**Q.** What is Segmentation ?
*(Ans. : Refer section 4.9.2)* **(May 2017, 5 Marks)**

□□□

CHAPTER 5

Module V

# I/O Organization and Peripherals

**Syllabus**

Common I/O device types and characteristics, Types of data transfer techniques : Programmed I/O, Interrupt driven I/O and DMA. Introduction to buses, Bus arbitration and multiple bus hierarchy, Interrupt types, Interrupts handling.

### Syllabus Topic : Common Input/Output Device Types and Characteristics

## 5.1 Input / Output System

**Q.** Explain the need of I/O module. **(5 Marks)**

- There are a wide variety of peripherals or I/O devices that deliver different amounts of data at different speeds and in different formats. All these devices are slower than CPU and RAM and hence to interface these devices to the CPU there is a need of I/O modules.
- Input/output module is interface to CPU and memory with one or more peripherals.
- The general model of I/O module interfacing with system bus is shown in Fig. 5.1.1.

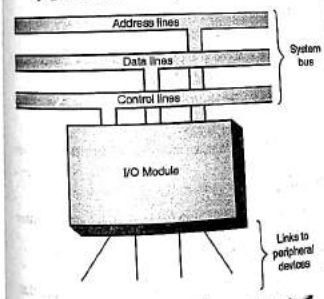



**Fig. 5.1.1 : General model of I/O module interface**

- The various functions of the I/O module involve :
  1. Issue of control and timing signals
  2. Communication with CPU
  3. Communication with peripheral
  4. Buffering of data between the CPU and peripheral and
  5. Detection of errors
- The internal block diagram of I/O module is shown in Fig. 5.1.2.

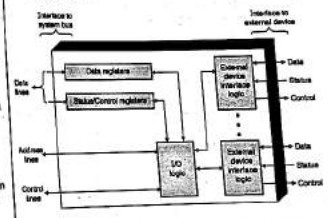



**Fig. 5.1.2 : Internal block diagram of an I/O module**

### 5.1.1 Parallel vs. Serial Interface

- The word, ***communication*** specifies, data transfer between two points.
- The data may be a digital or analog in nature.
- We will consider only digital data transfer because microprocessor is digital circuit. Suppose you want to transfer data from Point A to Point B. There are two possible ways of doing it :
  (1) Parallel data transfer

(2) Serial data transfer.

- For parallel data transfer, we can use 8255. Two 8255's are connected, one at each side.
- The Port A of 8255.
  (1) At point A is connected to Port A of 8255.
  (2) At point B. So the data transferred is of 8 bits at a time. For implementing this communication, we want 8 lines of PA interconnected and line will be the common ground between two points.
- In serial data transfer the data is transferred serially on a single line, the same hardware used for parallel data can also be used to implement this. Instead of connecting all 8 lines connect single line from Port A of 8255 :
  (1) To Port A of 8255.
  (2) To implement above communication we require one line of Port A interconnected and second line i.e. common ground between two points.

Now let's compare the specified 2 methods of data transfer.

| Sr. No. | Parallel | Serial |
|---|---|---|
| 1. | Parallel lines of 8/16/32 bits. Hence 8/16/32 bits can be transmitted simultaneously. | Only 1 bit is transmitted at a time. |
| 2. | The data transfer is comparatively faster. | The data transfer is comparatively slower. |
| 3. | Due to so many parallel paths 'crosstalk' among different bits is possible. | No 'crosstalk' possible. |
| 4. | This cannot be used for distant communication. | This can be used for distant communication. |
| 5. | More parallel hardware is required. | Less parallel hardware required. |
| 6. | It is comparatively costlier. | It is comparatively cheaper. |

- *In these two methods the cost of connecting two distant points, is the main factor. So though the parallel data transfer is faster, it is preferred for small distances only. But for long distances, serial data transfer is preferred.*
- In serial data transfer the 8 bits of data is converted into serial 8 bits; using shift register (parallel in serial out mode). These serial bits are transferred on single line using serial I/O data transfer.
- To transfer 8 bits of data, it will require 8 clock pulses. On the other side exactly opposite process is done. These serial 8 bits are accepted and converted to parallel form to get 8 bits of data. This process is shown in Fig. 5.1.3.

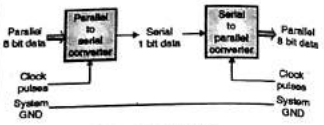



**Fig. 5.1.3 : Serial I/O**

### 5.1.2 Types of Communication Systems

The communication systems are classified on the basis of transmission :

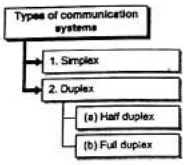



**Fig. 5.1.4 : Types of communication systems**

➜ **(1) Simplex**

- The simplex is one way transmission.
- The connection exists such that data transfer takes place only in one direction.
- There is no possibility of data transfer in the other direction.
- System A is transmitter and system B is receiver only.

➜ **(2) Duplex**

The duplex is two way transmissions. It is further divided in 2 groups :

**(a) Half duplex**

It is a connection between two terminals such that, data may travel in both the directions, but transmission activated in one direction at a time. This indicates that the line has to turn around after communication is complete in one direction.

**(b) Full duplex**

It is a connection between two terminals such that, data may travel in both the directions simultaneously. So it will contain one way transmission or two way transmission at a time.

**Syllabus Topic : Input Output Modules and 8089 IO Processor**

## 5.2 I/O Modules and 8089 IO Processor

An Input/output device can never be connected directly to the processor. It always has to be interfaced using an I/O module.

I/O module is required for the following reasons :

1. I/O devices are normally slower than the processor and also have different speeds. Hence if there is no I/O module, the processor will have to wait for long time for the I/O devices. Hence I/O module works as a buffer between the processor and I/O device to hold the data for the required time
2. Each I/O device has different data bus width. I/O module does the required width conversion.
3. Each I/O device has different protocol to be followed. Some use serial communication, some use parallel, some have handshaking signals etc. Hence I/O module communicates according to the protocol required by the I/O devices.

### 5.2.1 I/O Module

Q. Explain with block diagram the structure of I/O module. **(10 Marks)**

**(1) Need of input module :** each output device operates at a different speed, has different data format and different protocol.

Also, most of the I/O devices are slower than the speed of the processor. Hence, an IO module is used to interface the IO device to the processor.

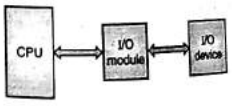



**Fig. 5.2.1 : Input output module**



**Block diagram of I/O module**

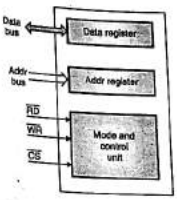



**Fig. 5.2.2 : Block diagram of I/O module**

- Data register is used to store the data given by the input device to be forwarded to the processor OR given by the processor to be forwarded to an output device.
- Address register is used to provide address of the IO device to be accessed.
- Mode and control unit indicates the mode of operation for the I/O module, as well as controls the transfer of data between the IO module and IO device, as well as IO module and CPU.

### 5.2.2 8089 I/O Processor

➜ **(MU - May 2014, May 2015, May 2016, May 2017)**

Q. Explain in brief function of 8089 I/O processor. **May 14, May 16, 4 marks**

Q. What are major requirements for an I/O module? **May 14, May 15, 6 Marks**

Q. Discuss the functions of 8089 I/O processor. **May 17, 10 Marks**

Q. Explain the operation of 8089 with 8086. **(10 Marks)**

Again, for interfacing 8086 with 8089, let us first see the pin diagram of 8089. The pin diagram of 8089 is shows in Fig. 5.2.3, which has almost the same pins to be interfaced to 8086 as that were in case of 8087.

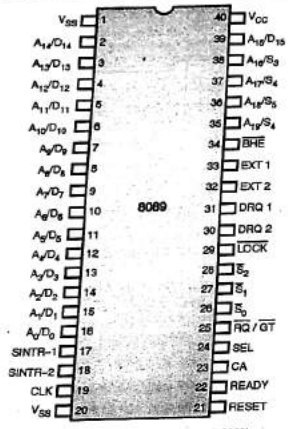



Fig. 5.2.3 : Pin Configuration of (IOP 8089)

### Interfacing 8086 and 8089 : Local Configuration

Refer Fig. 5.2.4.

- CLK, Reset and ready given to 8086/8088 as well as IOP 8089.
- $\overline{S_0}$ – $\overline{S_2}$ from 8086 given to 8089 as well as bus controller 8288.
- $AD_0$ – $AD_{15}$ and A16/S3 to A19/S7 lines of 8086 given to 8282 latch for demultiplexing address and data bus.
- $A_1$ to $A_{15}$ lines of 8086 also given to 8286 (data bus buffer). Output from 8282 is $A_0$ to $A_{19}$ with $\overline{BHE}$ signal.
- $A_1$ to $A_{15}$ lines given to address decoder for generating chip select for IOP.
- INT pin of IOP is given to 8259, for generating interrupt for 8086, whenever required.
- $\overline{RQ}/\overline{GT}$ of IOP 8089 connected to $\overline{RQ}/\overline{GT}$ of microprocessor.
- When 8089 is directly connected to 8086/8088, the $\overline{RQ}/\overline{GT}$ lines built into all these processors are used to arbitrate use of a local bus.
- First we will see how $\overline{RQ}/\overline{GT}$ of CPU operates.
- An external processor sends a pulse to the CPU to request use of the bus.
- The CPU finishes its current bus cycle, if one is in progress, and sends a pulse to the processor to indicate that it has been granted the bus.
- When the external processor is finished with the bus, it sends a final pulse to the CPU, to indicate that it is releasing the bus.

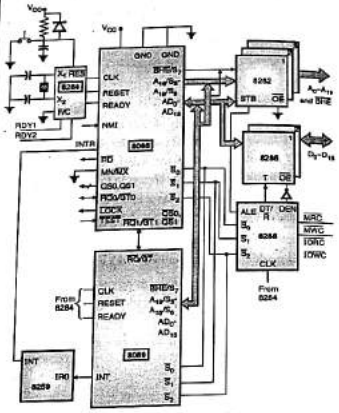



Fig. 5.2.4

(1) The host sets up message in memory and then wakes up the independent processor by sending a command to one of the independent processor's ports.

(2) The independent processor then accesses the shared memory to get the assigned task and executes the task in parallel with the host.

(3) After the task is completed, the external processor notifies its host of the completion by using either a status bit or an interrupt request.

(4) The message format, totally depends upon independent processor and the application. The message should specify which operation is to be performed the Input parameters and the addresses of the locations in which to store the result.



Similarly, the third configuration of closely coupled i.e. 8086 - 8087 and 8089, can be implemented together as seen in Section 5.2.2.

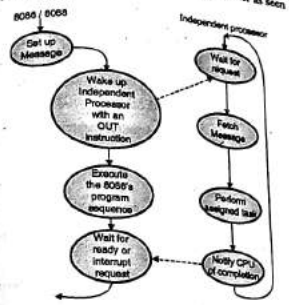



Fig. 5.2.5 : Interprocessor communication through shared memory

**Syllabus Topic : Types of Data Transfer Techniques - Programmed Input Output**

## 5.3 Types of Data Transfer Techniques : Programmed I/O, Interrupt Driven I/O and DMA.

→ (MU - May 2014, May 2015)

Q. Programmed I/O. May 14, 6 Marks
Q. Explain in brief Programmed I/O. May 15, 4 Marks
Q. Write short notes on: Programmed I/O, Interrupt Driven I/O and DMA based I/O. (10 Marks)

- There is yet another method of classifying the interfacing of I/O devices based on how and when the data is transferred between the processor and I/O devices.
- There are three types under this method of classification namely programmed I/O, interrupt driven I/O and DMA (Direct Memory Access).

### Definition of polling

Polling is a mechanism, wherein the processor checks each and every device for it needs a service or not.

### 5.3.1 Programmed I/O

Q. Write short notes on: Programmed I/O, Interrupt Driven I/O and DMA based I/O. (10 Marks)

- In the programmed I/O method of interfacing, CPU has direct control over I/O.
- The processor checks the status of the devices and issues read or write commands and then transfers data. During the data transfer, CPU waits for I/O module to complete operation and hence this system wastes the CPU time.
- The sequence of operations to be carried out in programmed I/O operation are :
  1. CPU requests for I/O operation.
  2. I/O module performs the said operation.
  3. I/O module updates the status bits.
  4. CPU checks these status bits periodically. Neither the I/O module can inform CPU directly nor can I/O module interrupt CPU.
  5. CPU may wait for the operation to complete or may continue the operation later.
- IC 8255 is generally used as a I/O module for programmed I/O method of interfacing.

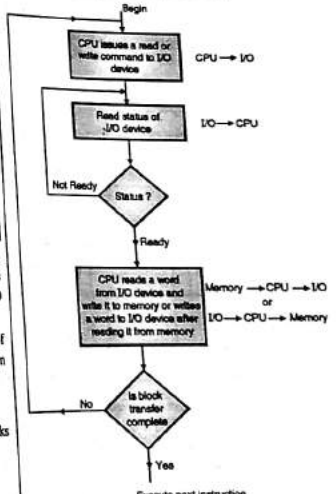



Fig. 5.3.1 : Transferring a block of data using programmed Input/output

- A common programming task is the transfer of a block of words between an Input/output device and memory.
- Fig. 5.3.1 gives a flowchart for transferring a block of data.

### 5.3.1.1 Input/Output Addressing

When the processor, main memory, and Input/output share a common bus, two modes of addressing are possible :

1. Memory-mapped Input/output
2. Input/output-mapped Input/output.

**1. Memory-Mapped Input / Output :**

General structure of a memory mapped Input/output is shown in Fig. 5.3.2. With memory-mapped Input/output, there is a single address space for memory locations and Input/output devices.

- Processor treats status and data registers as separate memory locations. Status and data registers are part of an Input/output device.
- Processor uses same memory instructions to access both memory and Input/output devices.
- With memory-mapped Input/output, a single read line and a single write line are needed on the bus.

**READ** line is activated during transfer of data from memory to CPU.

**Example :**

MOV AX, x [AX ← x]

8086, assembly instruction 'MOV Ax, x' will transfer a word of data from memory location x into CPU register Ax. This will activate READ line.

**WRITE** line is activated during transfer of data from CPU to memory.

**Example : MOV x, AX [x ← Ax]**

- With memory-mapped Input/output, no special commands (like IN, OUT) are needed for Input/output operations.
- Powerful addressing modes, available for accessing memory variables can also be used to address an Input/output device.
- A large set of instructions (meant for memory operands) can be used for Input/output. This allows more efficient programming.
- Interfacing circuit for memory-mapped Input/output is complex. Device has to behave like a set of memory locations to CPU.

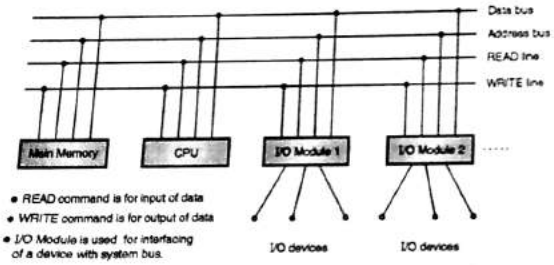



**Fig. 5.3.2 : Structure of memory mapped Input/output**



**2. Input/Output-Mapped Input/Output :**

Structure of an Input/output-mapped Input-output is shown in Fig. 5.3.3.

- There are separate control lines for memory and Input/output devices.
- A memory reference instruction does not effect an Input/output device.
- There are separate address spaces for memory and Input/output devices. An Input/output device and a memory location can have the same address.

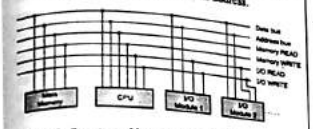



**Fig. 5.3.3 : Structure of Input/output-mapped Input/output**

- There are separate instructions for Input/output read and Input/output write.
- A memory reference instruction for memory read will cause generation of 'Memory READ' control signal.

  MOV Ax, x [Ax ← x]
- A memory reference instruction for memory write will cause generation of 'Memory WRITE' control signal.

  MOV x, Ax [x ← AX]
- An Input/output reference instruction for Input/output read will cause generation of 'Input/output READ' control signal.

  IN AL, 300H (AL ← (data from port with address 300H))
- An Input/output reference instruction for Input/output write will cause generation of 'Input output WRITE' control signal.

  OUT 300H, AL (Contents of AL register is written to port with address 300H)

**Q. Compare Interrupts driven I/O and DMA.** Dec. 15, 10 Marks

**Q. Write short notes on : Programmed I/O, Interrupt Driven I/O and DMA based I/O.** (10 Marks)

- Interrupt Driven I/O overcomes the disadvantage of programmed I/O i.e. the CPU waiting for I/O device.
- This disadvantage is overcome by CPU not repeatedly checking for the device being ready or not instead the I/O module interrupts when ready.
- The sequence of operations for interrupt Driven I/O is as below :
  1. CPU issues the read command to I/O device.
  2. I/O module gets data from peripheral while CPU does other work.
  3. Once the I/O module completes the data transfer from I/O device, it interrupts CPU.
  4. On getting the interrupt, CPU requests data from the I/O module.
  5. I/O module transfers the data to CPU.
- After issuing the read command the CPU performs its work, but checks for the interrupt after every instruction cycle as seen earlier in this chapter.
- When CPU gets an interrupt, it performs the following operation in sequence
  - Save context i.e. the contents of the registers on the stack
  - Processes interrupt by executing the corresponding ISR
  - Restore the register context from the stack.
- IC 8259 has 8 interrupt lines and is used as a I/O module when Interrupt driven I/O is used.
- The interrupt driven Input/output mechanism for transferring a block of data is shown in Fig. 5.3.4.

**Syllabus Topic : Interrupt Driven Input Output**

## 5.3.2 Interrupt Driven I/O

→ (MU - May 2015, Dec. 2015, Dec. 2016)

Interrupt driven I/O. May 15, Dec 16, 6 Marks

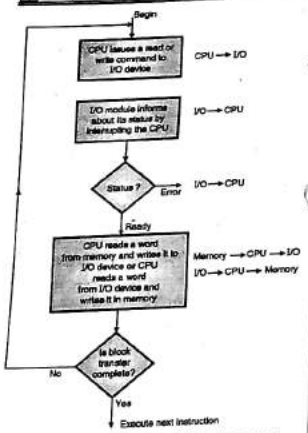



**Fig. 5.3.4 : Transferring a block of data using interrupt driven Input/output**

**Transferring a word of data**

- CPU issues a 'READ' command to Input/output device and then switches to some other program. CPU may be working on different programs.
- Once the Input/output device is ready with the data in its data register, Input/output device signals an interrupt to the CPU.
- When the interrupt from Input/output device occurs, it suspends execution of the current program, reads data from the port and then resumes execution of the suspended program.

### 5.3.2.1 Comparison between Programmed and Interrupt Driven Input/Output

- Programmed Input/output can be implemented with the help of software without any additional hardware cost. Whereas in interrupt driven, addition hardware is required to handle interrupt.
- Programmed Input/output is simple to implement and it is used in low end system where cost is a very important factor. Most of the contemporary computer systems are based on interrupt driven Input/output.
- Programmed Input/output is based on busy waiting. CPU keeps checking the status of the Input/output device.
- Since, Input/output devices are very slow, CPU will have to waste lot of its time waiting for the device to become ready.
- In interrupt driven Input/output, CPU switches to some other program without waiting for the Input/output device to complete or to become free.
- Only one Input/output activity can be handled using programmed Input/output.
- Whereas, multiple Input/output activities can be carried out in overlapped fashion, with interrupt driven Input/output.

### 5.3.2.2 Interrupt Processing

Interrupts can be generated by various sources both internal and external. An interrupt or exception causes CPU to temporarily transfer control from its current program to another program-an interrupt handle.

Interrupt handler services the interrupt. Input/output devices receive service from CPU primarily through interrupt. This mechanism significantly improves a computer's Input/output performance :

- Multiple Input/output activities can be handled.
- Provides rapid access to CPU.
- Frees the CPU from the need to check the status of the Input/output device.

The basic method of interrupting the CPU is by activating a control line that connects the interrupt source to the CPU. On recognizing the presence of interrupt, the CPU executes a specific interrupt-handling program.

Each interrupt source requires execution of a different interrupt-handling program. CPU must determine the source of interrupt and the address of the interrupt-handling program to be used.

**CPU takes following steps in response to an interrupt**

1. The CPU identifies the source of interrupts.
2. CPU finds the address of the interrupt-handling program.
3. The Program Counter (PC) and the status word PSW is saved on stack.
4. PC is loaded with interrupt handler. This will transfer control to interrupt handler program. Execution of interrupt handler proceeds until a return instruction is encountered, which transfers control back to the interrupted program.



A flowchart for interrupt processing is shown in Fig. 5.3.5.

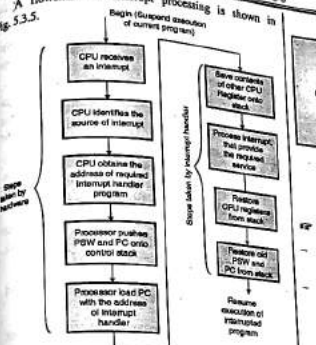



**Fig. 5.3.5 : Interrupt processing**

### 5.3.2.3 Interrupt Selection (Multiple Interrupts)

If several devices are connected to the CPU, CPU must know the interrupting device. Multiple devices may generate interrupt at the same time. In case of multiple interrupts, CPU will have to use some arbitration technique to select one Input/output device to service.

- Each device may have an independent interrupt request line going upto CPU (Fig. 5.3.7).
- Multiple devices may share the single interrupt request line (Fig. 5.3.6).
- CPU may be using vectored interrupt using bus arbitration technique or daisy chaining.

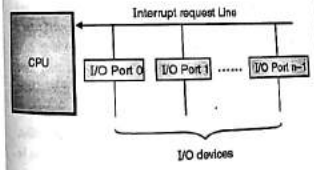



**Fig. 5.3.6 : Single line interrupt system**

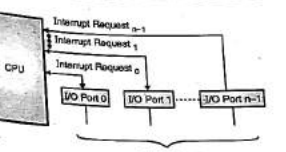



**Fig. 5.3.7 : Multiple interrupts using independent interrupt request lines**

**Independent request line :**

- The straight forward solution of finding the interrupting device is to provide multiple interrupt request lines.
- This results in immediate recognition of the interrupting device. Priority mechanism can be used to select one with highest priority, in case of multiple interrupts at the same time.

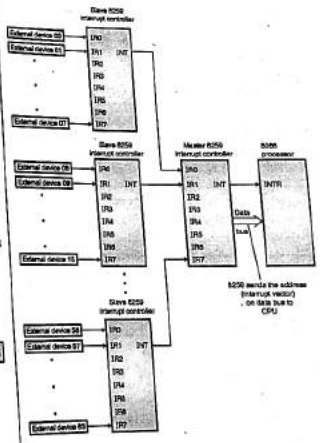



**Fig. 5.3.8 : Simple vectored interrupt system using 8259, interrupt controller**

- Providing multiple interrupt lines is an impractical approach. Only a few lines of the system bus can be devoted for interrupt.

**Software poll**

- The interrupt selection method requiring minimum hardware is the single interrupt request line method (Fig. 5.3.8).
- On receiving an interrupt, CPU can scan all the Input/output devices to determine the source of interrupt.
- Alternately, CPU starts executing a software routine which polls to each Input/output port to determine which Input/output device has caused the interrupt. This may be achieved by reading the status register of the port.
- Priority can be implemented easily by defining the polling sequence.

**Vectored interrupts using daisy chaining**

- In vectored interrupts, the interrupting device must supply the CPU with the starting address of the interrupt handler or interrupt vector (interrupt vector table contains addresses of the corresponding service routines).
- In daisy chaining, we have one interrupt acknowledge-line, which is chained through various devices. There is just one interrupt line.
- On receiving an interrupt request, the interrupt acknowledge line is activated which in turn passes this signal device by device.
- The first device which has made the interrupt request grabs the interrupt acknowledgement signal and blocks its further propogation.
- Interrupting device, holding the interrupt acknowledgement signal responds by putting a word which is normally an address of interrupt servicing program or an interrupt vector.
- The daisy chaining has an in-built priority scheme, which is determined by the sequence of devices on interrupt acknowledgement line.

**Bus arbitration (vectored) using 8259**

- With bus arbitration, an Input/output module must first gain control of the bus before it can raise the interrupt request line.
- Thus, only one module can raise the line at a time. When the CPU detects the interrupt, it responds on the interrupt acknowledgement line.
- The requesting Input/output module then places its vector on the data bus.

The intel processor 8086 provides a single Interrupt Request Line (INTR) and a single interrupt acknowledgement Line (INTA).

Fig. 5.3.8 shows the use of 8259 to connect multiple Input/output devices. 8259 is a general purpose interrupt handling IC. In cascade mode, it can handle up to 64 Input/output devices.

8259 accepts interrupt requests from attached devices, determines which interrupt has the highest priority, and then requests the CPU by raising INTR line.

The CPU acknowledges via the INTA line. In response to acknowledgement, 8259 places the appropriate vector information on the data bus.

### 5.3.2.4 Difference between Subroutine and Interrupt Service Routine

The routine executed in response to an interrupt request is called the interrupt service routine. Subroutines are written to modularly structure a big program. An interrupt service routine is treated much like a subroutine.

- When a call to a subroutine instruction is executed, the current program counter value is saved on top of the stack and the address of the subroutine is loaded in program counter.
- After execution of the current subroutine, the return address is popped in counter.
- When an interrupt comes, the processor first completes execution of current instruction.
- Then it loads the program counter with the address of the first instruction of the interrupt-handler-program. To facilitate return from the interrupt-service-routine, return address is pushed on top of the stack.
- An important difference between a subroutine and interrupt service routine is that a subroutine performs a function required by the program from which it is called, whereas the interrupt-service routine may have nothing in common with the program being executed at the time the interrupt request is received.
- Before starting execution of interrupt-service routine, any information (CPU registers, flag register etc.) to be altered must be saved.
- This information must be restored before execution of the interrupted program is resumed.



- In this way, the original program can continue execution without being affected in any way by interrupt.
- Typically, the processor saves only the contents of program counter and the processor status register when an interrupt is generated. In case of subroutine call, the process saves only the contents of program counter.
- Any additional information to be saved must be saved by program instructions at the beginning of subroutine and restored at the end.

**Subroutine call**

The instruction BSB is used for branching to a subroutine portion of a program.

- On execution of the above instruction, the return address which is currently in PC is stored at the beginning of the subroutine. The actual code of the subroutine starts from the next instruction.

$M \leftarrow PC + 5000$

$PC \leftarrow m + 1$

- After the subroutine is executed, control is transferred back to the calling program by means of a BUN instruction placed at the end of the subroutine.

$PC \leftarrow m$

### 5.3.2.5 Types of Interrupts

There are various sources of interrupts. These sources could be both internal and external.

- Input/output requests are external requests. They are used to initiate or terminate an Input/output operation.
- A virtual memory management unit can generate a page fault (type of interrupt) to swap a page from a secondary storage.
- Hardware or software errors can activate an interrupt.
- A power-supply failure can generate an interrupt to save critical data.
- An attempt by an instruction to divide by zero can raise an interrupt.
- Execution of a privileged instruction when not in privileged mode will generate an interrupt.
- Multiprogramming with pre-emptive scheduling is implemented using interrupts.

We can classify interrupts in following categories :

1. Program interrupts (s/w interrupts).
2. Timer interrupts.
3. Input/output interrupts.
4. Hardware failure.

1. **Program interrupts** are generated by some condition that occurs as a result of an instruction execution, such as :
   (a) Arithmetic overflow.
   (b) Division by zero.
   (c) Execution of an illegal machine instruction.
   (d) Segment limit violation.
   (e) Execution of privileged instruction.
2. **Timer interrupts** are generated within the processor. This allows the operating system to perform certain operations on regular basis.
3. **Input/output interrupts** are generation for initiation or completion of Input/output operation. Input/output failure or Input/output error too can generate an interrupt.
4. **Hardware failure interrupts** are generated by a failure, such as power failure or memory parity error.

**Syllabus Topic : Types of Data Transfer Techniques - DMA**

### 5.3.3 DMA

→ (MU - May 2014, Dec. 2014, May 2015, May 2016, Dec. 2016, May 2017)

Q. Explain the DMA based data transfer techniques for I/O devices. May 14, May 15, Dec. 16, 8 Marks

Q. DMA (Direct Memory Access) Dec. 14, 5 Marks

Q. What is the need of DMA ? Explain its various techniques of data transfer. May 16, 10 Marks

Q. Discuss the functions of 8089 I/O processor. May 17, 10 Marks

Q. Explain different data transfer techniques of DMA. (5 Marks)

Q. Write short notes on : Programmed I/O, Interrupt Driven I/O and DMA based I/O. (10 Marks)

Q. Explain different data transfer techniques of DMA. (5 Marks)

- DMA stands for Direct Memory Access. The I/O module can directly access (read or write) the memory using this method.
- Interrupt driven and programmed I/O require active operation of the CPU, hence transfer rate is limited and CPU is also busy doing the transfer operation. DMA is the solution for these problems.
- DMA controller takes over the control of the bus from CPU for I/O transfer.
- The internal block diagram of a DMA controller of the I/O module for DMA method of I/O interfacing is shown in the Fig. 5.3.1.

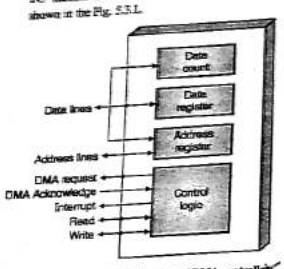



**Fig. 5.3.1 : Internal block diagram of DMA controller**

- In Fig. 5.3.1, you will notice that there are various registers like data count, data register and address register.
- The address register is used to hold the address of the memory location from which the data is to be transferred. There may be multiple address registers to hold multiple addresses.
- The address may be incremented or decremented after every transfer based on the mode of operation.
- The data count register is used to keep a track of the number of bytes to be transferred. The counter register is decremented after every transfer.
- The data register is used in a special case i.e. when the transfer of a block is to be done from one memory location to another memory location.
- Also you will note in the Fig. 5.3.1 the read and write signals are bidirectional.
- The DMA controller is initially programmed by the CPU, for the count of bytes to be transferred, address of the memory block for the data to be transferred etc.
- During this programming of the DMAC (DMA controller), the read and write lines work as inputs for the DMAC.
- This is because the CPU has to tell the DMAC whether it is reading or writing from the DMAC.
- Once the DMAC takes the control of the system bus i.e. transfers the data between the memory and I/O device, these read and write signals work as output signals.
- They are used to tell the memory that the DMAC wants to read or write from the memory according to the operation being data transfer from memory to I/O or from I/O to memory.
- The speciality of DMA is that the CPU carries on with other work while the DMA controller deals with transfer of data. DMA controller sends a signal when finished.

### 5.3.4 DMA Transfer Modes

- There are various modes of operation used to transfer the data between the memory and I/O device by the DMA controller.
- The four major methods used are discussed below :

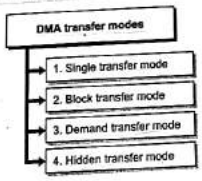



**Fig. 5.3.2 : DMA transfer modes**

**1. Single transfer mode**

- In single transfer mode, the device is programmed to make one byte transfer only after getting the control of the system bus.
- After transferring one byte the control of the bus will be returned back to the CPU.
- The word count will be decremented and the address decremented or incremented following each transfer.



- The disadvantage in this method is that the I/O device has to wait for a long time after every transfer for the extra request grant signals.
- The advantage is that the CPU has not to remain out of the system or not having the access of system bus for longer time, instead only for one transfer.

**2. Block transfer mode**

- In block transfer mode, the device is activated by DREQ (DMA Request) or software request and continues making transfers during the service until a Terminal Count (i.e. the counter becomes zero), or an external End of Process (EOP) is encountered.
- The disadvantage is that the CPU has to remain out of the system or not having the access of system bus for longer time, until all the bytes in the block are transferred.
- The problem further increases in case if the I/O device is slower and the system is waiting for the I/O device to complete its operation, thus the CPU has to wait for very long period in this case.
- The advantage is that the I/O device gets the transfer of data at a very faster speed.

**3. Demand transfer mode**

- In demand transfer mode, the device continues making transfers until a Terminal Count or external EOP is encountered, or until DREQ goes inactive.
- Thus, transfer may continue until the I/O device has exhausted its data handling capacity.
- Thus this method is said to be a trade off between the earlier two methods. If the I/O device is fast enough it will keep on getting data and need not wait for extra time for the request grant signals as in the single transfer method.
- Also the CPU has not to wait for longer time in case if the I/O device is slower, because if the I/O device is slower the transfer terminate.

**4. Hidden transfer mode**

- In hidden transfer mode, the DMA controller takes over the charge on the system bus and transfers data when processor does not needs system bus.
- The processor does not even realize of this transfer being taken place.
- The processor does not needs the system bus when it is performing some execution of an instruction in the ALU or certain instructions that do not need the system bus access at all. It happens mostly between the machine cycles.
- Hence these transfers are hidden from the processor.

**Syllabus Topic : Introduction to Buses**

## 5.4 Introduction to Buses

**BUS**

It is a group of wires, pins, signals, connection at having common function is called as a bus. The bus or connections that do the function of carrying data is called as **DATA bus.** The bus or connections that do the function of carrying address is called as **ADDRESS bus.**

The bus or connections that do the function of carrying control signals is called as **CONTROL bus.** The three busses together are called as system bus. The figure shown below shows the symbols used for single signal and the bus.

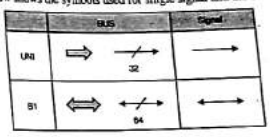



**Fig. 5.4.1**

### 5.4.1 Single-Bus Structure

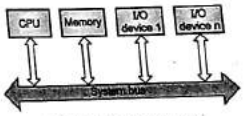



**Fig. 5.4.2 : A single bus structure**

The simplest way to interconnect functional units of a computer is to use a single system bus.

System bus consists of :

(i) Data bus
(ii) Address bus
(iii) Control bus

This bus is time shared. Because the bus can be used for only one transfer at a time, only two units can communicate at any given instant.

**1. Data bus**

- The data lines provide a path for moving data between system modules. These lines, collectively are called the 'Data bus'.
- The data lines are bi-directional, so that the data can be sent or received by the processor.

**2. Address bus**

- Every device connected to bus has an address. A memory unit is given a block of addresses, depending on number of words in it.
- For example, if the CPU wishes to read a word of data from memory, it puts the address of the desired word on the address lines.
- The address lines are always unidirectional i.e. the address is transmitted by the processor to different modules.

**3. Control bus**

- The control lines are used to control the various units like memory and I/O. Processor uses control signals to control various modules.
- Control signals transmit both command and timing information. Control signals specify operation to be performed. Typical control signals include :
  - Memory read
  - Memory write
  - I/O read
  - I/O write
  - Bus request
  - Bus grant

**The operation of the bus**

If one module wishes to send data to mother, it must do the followings :

(1) Obtain control of bus.

(2) Transfer data via the bus.

Similarly, if one module wishes to request data from another module, it must do the followings :

(1) Obtain control of bus.

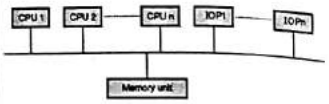
(2) Makes a request to the other module over the appropriate control lines and address lines.

(3) It must then wait for the requested module to send data.

The principle use of the system bus is high-speed data transfer between the CPU and memory.

Most I/O devices are slower than memory and they are put on the local bus. These devices are connected to the system bus via interface circuit called I/O controller. A single I/O controller can interface many I/O devices to the system bus.

## 5.5 Bus Contentions




(see s.m)Fig. 5.5.1 : Time shared common bus organization

In a bus system, processors, memory modules and peripheral devices are attached to the bus. The bus can handle only one transaction at a time between a master and slave. In case of multiple requests, the bus arbitration logic must be able to allocate or deallocate and it should service request one at a time.

Thus, such a bus is a time sharing or contention bus among multiple functional modules. As only one transfer can take place at any one time on the bus, the overall performance of the system is limited by the bandwidth of the bus.

- When number or processors (masters) contending to acquire a bus exceeds the limit then a single bus architecture may become a major bottleneck. This may cause a serious delay in servicing a transaction.
- Aggregate data transfer demand should never exceed the capacity of the bus. This problem can be countered to some extent by increasing the data rate of the bus and by using a wider bus (increasing data bus from 32 bits to 64 bits).
- Method of avoiding (reducing) contention is multiple bus hierarchy.

**Syllabus Topic : Bus Hierarchy**

### 5.5.1 Multiple-Bus Hierarchies

→ (MU - Dec. 2015)

**Q. Explain the importance of multiple bus hierarchies with the help of suitable diagram.**

**Dec. 15, 10 Marks**



If a greater number of devices are connected to the bus, performance will suffer due to following reasons :

In general, the more devices attached to the bus, the greater will be the propagation delay.

The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus.

This problem can be countered to some extent by increasing the data rate that the bus can carry and by using wider buses.

Most computer systems enjoy the use of multiple buses. These buses are arranged in a hierarchy.

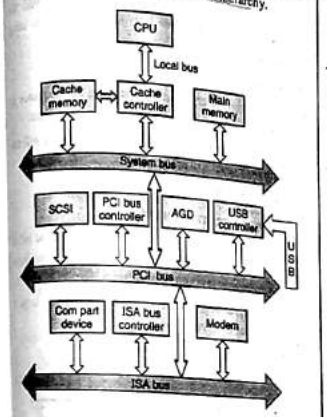



Fig. 5.5.2 : A multiple bus structure

**Syllabus Topic : Bus Arbitration**

## 5.6 Bus Arbitration

→ (MU - Dec. 2014, Dec. 2015, May 2016)

**Q. What is Bus Arbitration? Explain any two techniques of Bus Arbitration.**

**Dec. 14, Dec. 15, May 16, 10 Marks**

When many bus masters are connected to a single bus, there is bus congestion. To avoid or reduce the problem of bus congestion we use bus arbitration.

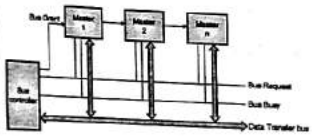
- The arbitration procedure comes into picture whenever there are more than one processor requesting the services of bus.
- The process of selecting a processor (master) among requesting processors is known as arbitration.
- A selection mechanism must be based on fairness or priority basis.

There are three representative arbitration schemes :

(1) Daisy-chaining
(2) Polling
(3) Independent requesting

- Arbitration process could be centralized or distributed. In the centralized scheme a hardware circuit device that is referred to as bus controller or bus arbiter decides about processor to be granted access of the bus among requesting processors.
- The bus controller may be a separate module or can be constructed as the part of the CPU. For example, I/O processor may need control of the bus for transferring data to memory.
- Similarly, CPU also needs the bus for various activities. Therefore, the system buses have I/O processor and CPU that need control of bus for data transfer.
- Now, this is upto the bus controller to resolve the simultaneous data transfer requests on the bus.

**1. Daisy Chaining**




(see s.m)Fig. 5.6.1 : Bus arbitration using daisy chaining

- Daisy chaining is characterized by **Bus Grant** signal connected serially from master to master as shown in the Fig. 5.6.1. The process involves three control signals :

(1) Bus Busy (2) Bus Request (3) Bus Grant.

- The bus controller responds to Bus request signal only if Bus busy is inactive. Bus busy signal remains active during the period it is being used by any of the processors.

- All processors are connected to the Bus request line. A processor makes a request to controller for bus grant by activating Bus request line.
- The bus controller responds to the Bus request signal by placing the Bus grant signal on the Bus grant line.
- When the first unit requesting access to the bus receives Bus grant signal, it blocks further propagation of signal, activates Bus busy signal, and starts using the Bus.
- When a non-requesting processor receives the Bus grant signal, it forwards the signal to next processor. Thus, if two processors simultaneously request bus access, the one closer to the bus controller gains access to the bus.
- Selection priority is determined by the proximity of the requesting processor from the controller.

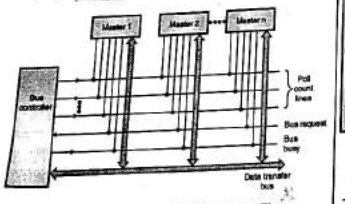
**Advantages**

- It is a very simple arbitration scheme.
- It requires very few control lines.
- Additional device can easily be added.

**Disadvantages**

- In this scheme the priority is wired in and connect be changed.
- There could be a problem of starvation. If the master 1 is generating Bus request at a high rate than rest of the masters may not get the bus for quite sometimes.
- If a master (say $i^{th}$ master) is not working then all processors ahead of it will never get bus grant line.

→ **2. Polling**




(GGG 8.88)**Fig. 5.6.2 : Bus arbitration using polling**

- Polling is process of calling each master turn by turn. A master is called by its address. Address of a master is generated on poll count lines.
- Poll count lines are connected to each device. Bus request and Bus busy line has the same meaning as in the context of daisy chaining.
- A request to use the bus is made on the Bus request line. Bus request will not be responded to till the Bus busy line is active.
- The bus controller responds to a signal on Bus request line by generating addresses in sequence on poll count lines. Each master (device or processor) is assigned a unique address.
- When the poll count matches, the address of a particular master that is requesting for the bus, the master activates the Bus busy signal and starts using the bus.

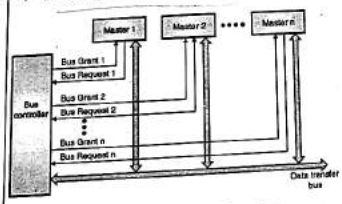
**Advantages**

- Priority can be changed by changing the sequence of the generation of addresses on the poll count lines.
- Failure of one master (a device or processor) will not affect any other master.

**Disadvantages**

- Polling requires more control lines.
- Maximum number of masters to be connected to bus is restricted by poll count lines. With n poll count lines, we can have maximum of $2^n$ masters.
- Delay in granting control of bus could become large if the number of devices to be polled is large.

→ **3. Independent requesting**




(GGG 8.89)**Fig. 5.6.3 : Bus arbitration using independent requesting**

- In this scheme, each master has its independent Bus request and Bus grant line. In this scheme the identification of requesting master is almost immediate and the request can be responded quickly.



- The arbitration among masters is still carried out by a central arbiter. In case of multiple requests, a requesting master can be selected on the basis of priority.
- Normally, we use priority-based policy for I/O transactions and fairness-based policy among the processors.

### 5.6.1 Elements of Bus Design

Buses are classified on the basis of the following parameters :

1. Bus type :
   (a) Dedicated (b) Multiplexed
2. Method of Arbitration :
   (a) Centralized (b) Distributed
3. Timing :
   (a) Synchronous (b) Asynchronous
4. Bus width :
   (a) Address bus width (b) Data bus width
5. Data Transfer Type :
   (a) Read (b) Write
   (c) Read-modify-write (d) Read-after-write
   (e) Block

**1. Bus type**

Bus lines could be dedicated or multiplexed. Dedicated lines could be dedicated **functionally** or **physically**.

An example of functional dedication is use of separate dedicated address and data lines. Physical dedication refers to the use of multiple buses, each of which connects only a subset of modules. A typical example is the use of I/O bus to connect I/O devices. Physical dedication of bus improves throughput as there is less bus contention.

Buses can be time multiplexed by using the same lines for multiple purposes. For example, address and data information may be transmitted over the same set of lines using an address valid control signal. At the beginning of data transfer, address is placed on the bus and the address valid control signal is activated.

After an interval, address is removed from the bus, and the same bus lines are used for subsequent read or write data transfer. The advantage of time multiplexing is the use of fewer lines. But, there is reduction in performance as it restricts parallelism.

**2. Method of Arbitration**

In a centralized scheme, a bus controller is responsible for allocating time on the bus. In a shared bus system, a number of processors may need to access the bus simultaneously.

Because, only one unit at a time can successfully transmit over the bus, some method of arbitration is needed. In a distributed scheme, there is no central controller. Modules contain access control logic and they act together to share the bus.

**3. Timing**

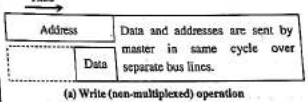
Buses use either synchronous timing or asynchronous timing. In synchronous bus, the occurrence of events on the bus is determined by a clock. In asynchronous timing, bus clock signal is replaced with control signals like ready and accept. These signals are generated by communicating units. These units are self timed and units with different data transfer rates can communicate with each other.

Synchronous timing is simpler to implement with asynchronous timing, a mixture of slow and fast devices can share the bus efficiently.

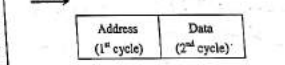
**4. Bus width**

The width of data has an effect on system performance. A wider bus will be able to carry greater number of bits in a data transfer operation. The width of the address bus has an effect on address space.

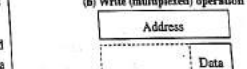
**5. Data transfer type**



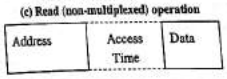

(a) Write (non-multiplexed) operation




(b) Write (multiplexed) operation




(c) Read (non-multiplexed) operation



**(d) Read (multiplexed) operation**

| Address | Access Time | Data Read | Data Write |
|---|---|---|---|

**(e) Read-modify-write operation**

| Address | Data write | Access Time | Data Read |
|---|---|---|---|

**(f) Read-after-write operation**

| Address | Data | Data |
|---|---|---|

**(g) Block data transfer**

**Fig. 5.6.4**

A bus can support various data transfer types as shown above.

- In case of a multiplexed bus, the bus is first used for specifying the address and then for transferring the data.
- For a read operation, there is a wait while the data is being accessed from the slave.
- In the case of dedicated buses, the address is put on the address bus and remains there while the data are put on the data bus.
- Some buses allow a read-modify-write operation. The whole operation is indivisible to prevent any access to the data element by other masters on the bus. This is used for protecting shared memory resources in a multiprogramming system.
- In read-after-write operation, the read operation is performed for checking purposes.
- Some bus systems also support a block data transfer. In this, one address cycle is followed by n data cycles.

**Syllabus Topic : Interrupt types, Interrupts Handling**

## 5.7 Interrupts types

***Definitions***

**1. Interrupt**

*It is a mechanism by which an I/O device (Hardware interrupt) or an instruction (Software interrupt) can suspend the normal execution of the processor and get itself serviced.*

**2. Interrupt service routine (ISR)**

A small program or a routine that when executed services the corresponding interrupting source is called as an ISR.

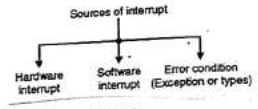
**3. Vectored/Non-vectored Interrupt**

If the ISR address of an interrupt is to be taken from the interrupting source itself, it is called as a non-vectored interrupt; else it is a vectored interrupt.

**4. Maskable/Non-maskable Interrupt**

Interrupt that can be masked (disabled) or unmasked (enabled) by the programmer is called as maskable interrupt else it is a non maskable interrupt.

In 8086, we have three sources of interrupt.




**1. Hardware Interrupt**

In this type of interrupt, physical pins are provided in the chip. In 8086 we have two pins :

(i) NMI (Non maskable interrupt).

(ii) INTR.

To interrupt the processor we have to apply signal to these pins. As name suggests, NMI is non maskable i.e. microprocessor has to service this interrupt, it cannot avoid it. Whereas INTR is maskable, if IF flag in flag register is '0', microprocessor will not recognize interrupt available on the pin.

**2. Software Interrupt**

Software interrupt, in 8086 we have INT instruction. When INT instruction is executed interrupt will occur.

**3. Error Conditions (Exception Or Types)**

- We know that 8086 supports division, multiplication, addition etc.
- Suppose by mistake if user asks microprocessor to divide any number by ZERO, then you know that dividing any number by ZERO produces answer '∞ *(infinity)*'.
- So in this case microprocessor will generate an interrupt *"Automatically"* and interrupt current execution.



- In ISR, user can display message *"Divide by zero error"*. (Possibly you may have come across this error). Instead of showing the answer as '∞ *(infinity)*'.
- Thus, internally generated errors produces an interrupt for microprocessor, normally referred as *"TYPE"* by Intel engineer and referred as *"Exception"* by Motorola engineer.
- Thus we conclude that 8086 has a simple and versatile interrupt system. Every interrupt is assigned a *"type code"* that identifies it to the CPU. The 8086 can handle upto 256 different interrupt types.
- Interrupts may be initiated by devices external to the CPU; in addition, they also may be triggered by software interrupt introductions and under certain condition, by the CPU itself. Fig. 5.7.1 shows interrupt sources for 8086.
- As shown in Fig. 5.7.1 8086 have two lines that external device may use to signal interrupts. The INTR line is usually driven by an Intel 8259A (PIC), which in turn connected to the devices that need interrupt services.

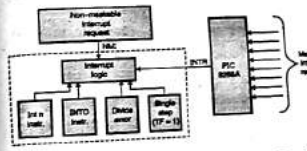



**Fig. 5.7.1 : Interrupt sources (INTO-Interrupt on Overflow)**

| Sr. No. | Hardware Interrupts | Software Interrupts |
|---|---|---|
| 1. | To implement a hardware interrupt a pin is given on the processor | To implement a software interrupt an instruction is given in the instruction set of the processor. |
| 2. | Hardware interrupts are mostly maskable or non-maskable interrupts | Software interrupts are mostly non-maskable although may have lower priority. |
| 3. | Hardware interrupts may be vectored or non-vectored | Software interrupts are mostly vectored. |
| 4. | Hardware interrupts of 8086 are NMI and INTR | Software interrupts of 8086 are INT n (where n is any value from 00H to FFH) |

## 5.8 Software Interrupts

- The INT instruction of the 8086 can be used to do one of the 256 interrupts (Type 0-255).
- The interrupt type is specified by the number as a part of the instruction. Example INT 0 instruction can be used to send execution to a divide by zero interrupts service routine.
- With the help of these software interrupts we can call the routines from different programs in the system example BIOS. The BIOS routines are called with INT instructions.

**Syllabus Topic : Interrupt Handling**

## 5.9 Interrupt Handling

At the end of each instruction cycle, the 8086 checks to see if any interrupts have been requested. Therefore whenever interrupt occurs, it won't be immediately checked by microprocessor. Microprocessor first completes execution of current instruction and then checks for an interrupt.

Now the main important point is, how 8086 acknowledges it.

(1) First, microprocessor will complete execution of current instruction.

(2) It checks for any internal interrupt, suppose the same is not present.

(3) Then it checks for NMI i.e. hardware interrupt.

## 5.10 Exam Pack (University and Review Questions)

**Syllabus Topic : Common input/output device types and characteristics**

Q. Explain the need of I/O module. **(5 Marks)**
*(Ans. : Refer section 5.1)*

Q. Explain with block diagram the structure of I/O module. *(Ans. : Refer section 5.2.1)* **(10 Marks)**

Q. Explain the operation of 8089 with 8086. *(Ans. : Refer section 5.2.2)* **(10 Marks)**

☞ **Syllabus Topic : Input Output Modules and 8089 IO Processor**

Q. Explain in brief function of 8089 I/O processor. *(Ans. : Refer section 5.2.2)* **(May 2014, 4 Marks)**

Q. What are major requirements for an I/O module ? *(Ans. : Refer section 5.2)* **(May 2014, 6 Marks)**

Q. What are major requirements for an I/O module ? *(Ans. : Refer section 5.2)* **(May 2015, 6 Marks)**

Q. Explain in brief the function of 8089 I/O processor. *(Ans. : Refer section 5.2.2)* **(May 2016, 5 Marks)**

Q. Discuss the functions of 8089 I/O processor. *(Ans. : Refer section 5.2.2)* **(May 2017, 10 Marks)**

☞ **Syllabus Topic : Types of Data Transfer Techniques - Programmed Input Output**

Q. Programmed I/O. *(Ans. : Refer section 5.3)* **(May 2014, 6 Marks)**

Q. Explain in brief Programmed I/O. *(Ans. : Refer section 5.3)* **(May 2015, 4 Marks)**

☞ **Syllabus Topic : Interrupt Driven Input Output**

Q. Interrupt driven I/O. *(Ans. : Refer section 5.3.2)* **(May 2015, 6 Marks)**

Q. Interrupt driven I/O. *(Ans. : Refer section 5.3.2)* **(Dec. 2016, 5 Marks)**

Q. Compare Interrupt driven I/O and DMA. *(Ans. : Refer section 5.3.2)* **(Dec. 2015, 10 Marks)**

☞ **Syllabus Topic : Types of Data Transfer Techniques - DMA**

Q. Explain the DMA based data transfer techniques for I/O devices. *(Ans. : Refer section 5.3.3)* **(May 2014, 8 Marks)**

Q. DMA (Direct Memory Access) *(Ans. : Refer section 5.3.3)* **(Dec. 2014, 5 Marks)**

Q. Explain DMA based data transfer technique for I/O devices. *(Ans. : Refer section 5.3.3)* **(May 2015, 7 Marks)**

Q. What is the need of DMA ? Explain its various techniques of data transfer. *(Ans. : Refer section 5.3.3)* **(May 2016, 10 Marks)**

Q. Explain DMA based data transfer technique for I.O. devices. *(Ans. : Refer section 5.3.3)* **(Dec. 2016, 10 Marks)**

Q. Discuss the functions of 8089 I/O processor. *(Ans. : Refer section 5.3.3)* **(May 2017, 10 Marks)**

Q. Write short notes on : Programmed I/O, Interrupt Driven I/O and DMA based I/O. **(10 Marks)** *(Ans. : Refer sections 5.3, 5.3.1, 5.3.2 and 5.3.3)*

Q. Explain different data transfer techniques of DMA. *(Ans. : Refer section 5.3.3)* **(5 Marks)**

☞ **Syllabus Topic : Bus Hierarchy**

Q. Explain the importance of multiple bus hierarchies with the help of suitable diagram. *(Ans. : Refer section 5.5.1)* **(Dec. 2016, 10 Marks)**

☞ **Syllabus Topic : Bus Arbitration**

Q. What is Bus Arbitration ? Explain any two techniques of Bus Arbitration. *(Ans. : Refer section 5.6)* **(Dec. 2014, 10 Marks)**

Q. What is bus arbitration ? Explain any two techniques of bus arbitration. *(Ans. : Refer section 5.6)* **(Dec. 2015, 5 Marks)**

Q. What is bus arbitration ? Explain its techniques. *(Ans. : Refer section 5.6)* **(May 2016, 5 Marks)**

□□□

CHAPTER 6

Module VI

# Advance Processor Principles

**Syllabus**

Introduction to parallel processing, Flynn's Classification, Concepts of superscalar architecture, out-of-order execution, speculative execution, multithreaded processor, VLIW, data flow computing. Introduction to Multi-core processor

**Syllabus Topic : Introduction to Parallel Processing Concepts**

## 6.1 Introduction to Parallel Processing Concepts

### 6.1.1 Overlapping the CPU and Memory or I/O Operations

- This is a very basic parallelism implemented in the Intel's 8086, wherein we had instructions prefetched from the memory before they are to be executed. Also for I/O operations a special dedicated I/O processor can be connected.
- Hence all the operations i.e. accessing the data or instructions from memory, accessing I/O devices and internal ALU operations can be done simultaneously. In the 8086 processor, there are two separate units to perform the memory accesses and the ALU operations named as Bus Interface Unit (BIU) and the Execution Unit (EU).

**Syllabus Topic : Flynn's Classifications**

## 6.2 Flynn's Classifications

→ **(MU - May 2014, May 2015, Dec. 2015, May 2016, Dec. 2016, May 2017)**

Q. List the Flynn's Classification of Parallel Processing Systems. **May 14, May 2015, 3 Marks**

Q. Explain Flynn's classification. **Dec. 15, May 16, Dec. 16, May 17, 10 Marks**

Q. Write a short note on Flynn's classification of parallel computing. **(5 Marks)**

### 6.2.1 Flynn's Classification of Parallel Computing

A method introduced by Flynn, for classification of parallel processors is most common. This classification is based on the number of Instruction Streams (IS) and Data Streams (DS) in the system. There may be single or multiple streams of each of these. Hence accordingly, Flynn classified the parallel processing into four categories :

1. Single Instruction Single Data (SISD)
2. Single Instruction Multiple Data (SIMD)
3. Multiple Instruction Single Data (MISD)
4. Multiple Instruction Multiple Data (MIMD)

→ **1. Single Instruction Single Data (SISD)**

- In this case there is a single processor that executes one instruction at a time on single data stored in the memory.
- In fact, this type of processing can be said to be unit processing, hence unit processors fall into this category.
- Fig. 6.2.1 shows this type of system. You will notice there is a Control Unit (CU) that accepts the instruction from the processor and decodes it.
- The Processing Element (PE) accesses the data from the memory and performs the operation on this data as per the signal given by control unit.

Q. Explain the operation of 8089 with 8086. *(Ans. : Refer section 5.2.2)* **(10 Marks)**

☞ **Syllabus Topic : Input Output Modules and 8089 IO Processor**

Q. Explain in brief function of 8089 I/O processor. *(Ans. : Refer section 5.2.2)* **(May 2014, 4 Marks)**

Q. What are major requirements for an I/O module ? *(Ans. : Refer section 5.2)* **(May 2014, 6 Marks)**

Q. What are major requirements for an I/O module ? *(Ans. : Refer section 5.2)* **(May 2015, 6 Marks)**

Q. Explain in brief the function of 8089 I/O processor. *(Ans. : Refer section 5.2.2)* **(May 2016, 5 Marks)**

Q. Discuss the functions of 8089 I/O processor. *(Ans. : Refer section 5.2.2)* **(May 2017, 10 Marks)**

☞ **Syllabus Topic : Types of Data Transfer Techniques - Programmed Input Output**

Q. Programmed I/O. *(Ans. : Refer section 5.3)* **(May 2014, 6 Marks)**

Q. Explain in brief Programmed I/O. *(Ans. : Refer section 5.3)* **(May 2015, 4 Marks)**

☞ **Syllabus Topic : Interrupt Driven Input Output**

Q. Interrupt driven I/O. *(Ans. : Refer section 5.3.2)* **(May 2015, 6 Marks)**

Q. Interrupt driven I/O. *(Ans. : Refer section 5.3.2)* **(Dec. 2016, 5 Marks)**

Q. Compare Interrupt driven I/O and DMA. *(Ans. : Refer section 5.3.2)* **(Dec. 2015, 10 Marks)**

☞ **Syllabus Topic : Types of Data Transfer Techniques - DMA**

Q. Explain the DMA based data transfer techniques for I/O devices. *(Ans. : Refer section 5.3.3)* **(May 2014, 8 Marks)**

Q. DMA (Direct Memory Access) *(Ans. : Refer section 5.3.3)* **(Dec. 2014, 5 Marks)**

Q. Explain DMA based data transfer technique for I/O devices. *(Ans. : Refer section 5.3.3)* **(May 2015, 7 Marks)**

Q. What is the need of DMA ? Explain its various techniques of data transfer. *(Ans. : Refer section 5.3.3)* **(May 2016, 10 Marks)**

Q. Explain DMA based data transfer technique for I.O. devices. *(Ans. : Refer section 5.3.3)* **(Dec. 2016, 10 Marks)**

Q. Discuss the functions of 8089 I/O processor. *(Ans. : Refer section 5.3.3)* **(May 2017, 10 Marks)**

Q. Write short notes on : Programmed I/O, Interrupt Driven I/O and DMA based I/O. **(10 Marks)** *(Ans. : Refer sections 5.3, 5.3.1, 5.3.2 and 5.3.3)*

Q. Explain different data transfer techniques of DMA. *(Ans. : Refer section 5.3.3)* **(5 Marks)**

☞ **Syllabus Topic : Bus Hierarchy**

Q. Explain the importance of multiple bus hierarchies with the help of suitable diagram. *(Ans. : Refer section 5.5.1)* **(Dec. 2016, 10 Marks)**

☞ **Syllabus Topic : Bus Arbitration**

Q. What is Bus Arbitration ? Explain any two techniques of Bus Arbitration. *(Ans. : Refer section 5.6)* **(Dec. 2014, 10 Marks)**

Q. What is bus arbitration ? Explain any two techniques of bus arbitration. *(Ans. : Refer section 5.6)* **(Dec. 2015, 5 Marks)**

Q. What is bus arbitration ? Explain its techniques. *(Ans. : Refer section 5.6)* **(May 2016, 5 Marks)**

□□□

CHAPTER 6

Module VI

# Advance Processor Principles

**Syllabus**

Introduction to parallel processing, Flynn's Classification, Concepts of superscalar architecture, out-of-order execution, speculative execution, multithreaded processor, VLIW, data flow computing. Introduction to Multi-core processor architecture.

**Syllabus Topic : Introduction to Parallel Processing Concepts**

## 6.1 Introduction to Parallel Processing Concepts

### 6.1.1 Overlapping the CPU and Memory or I/O Operations

- This is a very basic parallelism implemented in the Intel's 8086, wherein we had instructions prefetched from the memory before they are to be executed. Also for I/O operations a special dedicated I/O processor can be connected.
- Hence all the operations i.e. accessing the data or instructions from memory, accessing I/O devices and internal ALU operations can be done simultaneously. In the 8086 processor, there are two separate units to perform the memory accesses and the ALU operations named as Bus Interface Unit (BIU) and the Execution Unit (EU).

**Syllabus Topic : Flynn's Classifications**

## 6.2 Flynn's Classifications

➔ **(MU - May 2014, May 2015, Dec. 2015, May 2016, Dec. 2016, May 2017)**

Q. List the Flynn's Classification of Parallel Processing Systems. **May 14, May 2015, 3 Marks**

Q. Explain Flynn's classification. **Dec. 15, May 16, Dec. 16, May 17, 10 Marks**

Q. Write a short note on Flynn's classification of parallel computing. **(5 Marks)**

### 6.2.1 Flynn's Classification of Parallel Computing

A method introduced by Flynn, for classification of parallel processors is most common. This classification is based on the number of Instruction Streams (IS) and Data Streams (DS) in the system. There may be single or multiple streams of each of these. Hence accordingly, Flynn classified the parallel processing into four categories :

1. Single Instruction Single Data (SISD)
2. Single Instruction Multiple Data (SIMD)
3. Multiple Instruction Single Data (MISD)
4. Multiple Instruction Multiple Data (MIMD)

➔ **1. Single Instruction Single Data (SISD)**

- In this case there is a single processor that executes one instruction at a time on single data stored in the memory.
- In fact, this type of processing can be said to be unit processing, hence unit processors fall into this category.
- Fig. 6.2.1 shows this type of system. You will notice there is a Control Unit (CU) that accepts the instruction from the processor and decodes it.
- The Processing Element (PE) accesses the data from the memory and performs the operation on this data as per the signal given by control unit.

- The Memory Module (MM) is connected to the PE and the CU for the data and the instruction streams respectively.

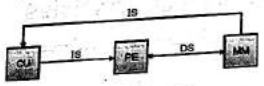



**Fig. 6.2.1 : SISD computer**

**➜ 2. Single Instruction Multiple Data (SIMD)**

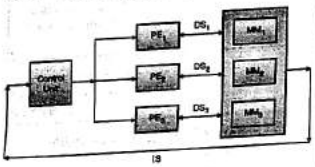



**Fig. 6.2.2 : SIMD organization**

- In this case the same instruction is given to multiple processing elements, but different data.
- This kind of system is mainly used when many data (array of data) have to be operated with same operation. Vector processors and array processors fall into this category.
- Fig. 6.2.2 shows the structure of a SIMD system

**➜ 3. Multiple Instruction Single Data (MISD)**

- In case of MISD, there are multiple instruction streams and hence multiple control units to decode these instructions.
- Each control unit takes a different instruction from the different memory module in the same memory.
- The data stream is single. In this case the data is taken by the first processing element.
- This processing element performs an operation on the data given to it and forwards the result to the next processing element for further operation.
- This processing element performs a similar operation and so on the final result reaches back to the same memory module.

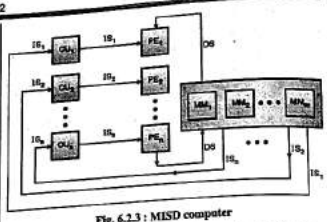



**Fig. 6.2.3 : MISD computer**

- This system is not used much, but can be used in cases where in a data has to undergo many computations to get the result for e.g. to add two floating point numbers. Fig. 6.2.3 shows the implementation of such a system.

**➜ 4. Multiple Instruction Multiple Data (MIMD)**

- This is a complete parallel processing example. Here each processing element is having a different set of data and different instructions.
- Examples of this kind of systems are SMPs (Symmetric Multiprocessors), clusters and NUMA (Non-Uniform Memory Access). Fig. 6.2.4 shows the structure of such a system.

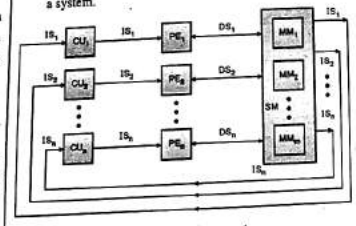



**Fig. 6.2.4 : MIMD computer**

**Syllabus Topic : Concepts of Superscalar Architecture**

## 6.3 Superscalar Processors

**Q. Write a short note on Superscalar architecture. (5 Marks)**

- Superscalar processors are those processors that have multiple execution units.



- Hence these processors can execute the independent instructions simultaneously and hence with the help of this parallelism it increases the speed of the processor.
- It has been seen that the number of independent consecutive instructions is around 2 to 5. Hence the instruction issue degree in a superscalar processor is restricted from 2 to 5.

### 6.3.1 Pipelining in Superscalar Processors

- The pipelining is the most important representation of demonstrating the speed increase by the superscalar feature of the processors. Fig. 6.3.1 shows the timing diagram of a two issue superscalar and 4-stage pipeline processor.

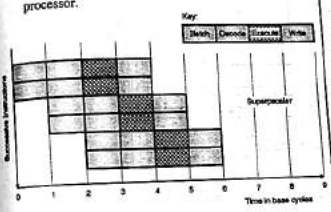



**Fig. 6.3.1 : Timing diagram of the superscalar processor with degree m=2**

- Hence to implement multiple operations simultaneously, we need to have multiple execution units to execute each instruction independently.

Fig. 6.3.2 shows a block diagram of a typical superscalar processor. As shown in the Fig. 6.3.1, the instruction fetch unit fetches m instructions ( where m is the degree of the processor superscalar).

- The ID (instruction decode) and rename unit, decodes the instruction and then by the use of register renaming avoids instruction dependency. The instruction window then takes the decoded instructions and based on some pair ability rules, issues them to the respective execution units.
- The instructions once executed move to the Retire and write back unit, wherein the instructions retire and the result is written back to the corresponding destination.

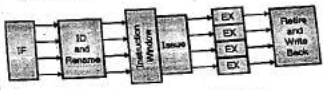



**Fig. 6.3.2 : Block diagram of a typical superscalar processor**

A RISC or CISC processors execute one instruction per cycle. Their performance can be improved with superscalar architecture. In a superscalar architecture :

1. Multiple instruction pipelines are used.
2. Multiple instructions are issued for execution per cycle.
3. Multiple results are generated per cycle.

Superscalar processors can exploit more instruction-level parallelism in user programs.

**☞ Pipelining in superscalar processor**

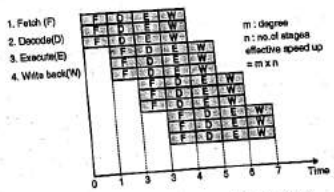



**Fig. 6.3.3 : A superscalar processor of degree = 3 (m)**

The basic structure of a superscalar pipeline is shown in Fig. 6.3.3. There are three instruction pipeline in parallel. A superscalar processor of degree m can issue m instructions per cycle. To fully utilize the capability of a superscalar architecture, m instructions must be executed parallely. A typical superscalar architecture for RISC processor is shown in Fig. 6.3.4.

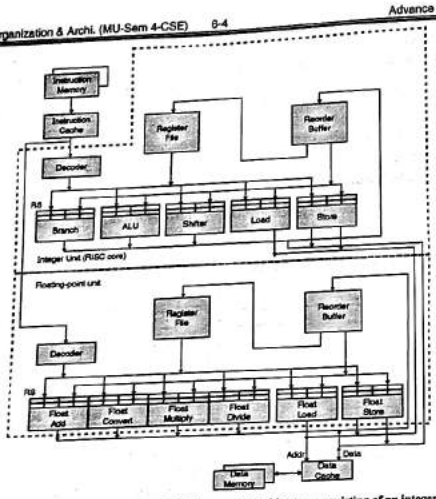



**Fig. 6.3.4 : A typical superscalar RISC processor architecture consisting of an integer unit and a floating-point unit**

- Multiple instruction pipelines are used.
- The instruction cache supplies multiple instructions per fetch.
- The number of instructions issued may be constrained by data dependency and resource conflict.
- Multiple functional units are built into the integer unit and into the floating point unit.
- Multiple data buses exist among the functional units.
- All functional units can be simultaneously used if conflict and dependencies do not exist among them during any cycle.

## 6.4 Vector Processor

- Vector processors are capable of executing hundreds of millions of floating point operations per second. Main task of a vector processor is to perform arithmetic operations on arrays of vectors of floating point numbers.
- Approach of computation involves a high degree of parallelism in vector operations. There are three approaches in having high degree of parallelism. These are :

1. Pipelined ALU
2. Parallel ALU
3. Parallel processor

### ➜ 1. Pipelined ALU

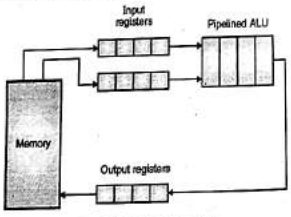



**Fig. 6.4.1 : Pipelined ALU**



- Concept of pipelining is extended to pipelined ALU. Floating point operations are complex. They can be broken into sub operations.
- A floating point addition can be broken up into four stages; compare, shift, add and normalize.
- When we decompose an operation into four stages, these stages can operate on different set of data concurrently. This is shown in Fig. 6.4.1. A vector number is presented to the first stage.
- As the operation proceeds, four different sets of number will be operated on concurrently in the pipeline.
- Input registers have been added to enhance pipeline operation. This keeps the pipelined ALU busy as the vector element can be fetched form input registers.

### ➜ 2. Parallel ALU

- Vector processing can be done with the help of multiple ALUs. These ALUs are in a single processor, under the control of single control unit. Data elements are routed to ALUs so that they can function in parallel.
- As with pipelined organization, a parallel ALU organization is suitable for vector processing. The control unit routes vector elements to ALUs in round-robin fashion until all elements are processed.

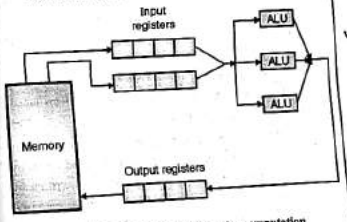



**Fig. 6.4.2 : Parallel ALUs for vector computation**

### ➜ 3. Parallel processor

- Vector processing can be done using multiple processors. The main task is broken in sub tasks or multiple processes to be executed in parallel.
- This technique requires complex operating system and parallelising compiler. Interaction latency is higher in multiple processor environment.
- Each process has its own control unit and may or may not have its local memory. In multiple processor, these processors may co-operate effectively to execute a task.

**Ex. 6.4.1**

**Explain : (i) Vector Processing (ii) Vector Operations. Explain how matrix multiplication is carried out on a computer supporting Vector Computations.**

**Soln. :**

**(I) Vector processing**

A vector is a set of scalar data items. All the items of a vector are of the same type. Vector processing involves arithmetic or logical operations on vectors.

Vector processing is faster and more efficient than scalar processing. Vector processing is often carried out by pipelined processors.

**(II) Vector operations**

1. Vector-vector instructions.
2. Vector-scalar instructions.
3. Vector-memory instructions.
4. Vector-reduction instructions.
5. Gather and scatter instructions.
6. Masking instructions.

**☞ Multiplication of arrays (using systolic arrays)**

Systolic processors can be designed to implement various complex arithmetic operations such as :

1. Matrix multiplication
2. Solution of linear equations
3. Matrix inversion

Let, X and Y are two $3 \times 3$ matrices

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \end{bmatrix} \qquad Y = \begin{bmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ y_{31} & y_{32} & y_{33} \end{bmatrix}$$

Product of matrices X and Y gives the matrix XZ

$$Z = XY$$

An element $z_{ij}$ of the matrix Z is given by,

$$z_{ij} = \sum_{k=1}^{a} x_{ik} \times y_{kj}$$

A systolic array for matrix multiplication can be constructed from a cell (Fig. Ex. 6.4.1) that executes the following multiply-and-add operation.

$$z = z' + x \times y$$

**Fig. Ex. 6.4.1 : A cell for $z = z' + x \times y$**

Each M cell has three inputs and three outputs.

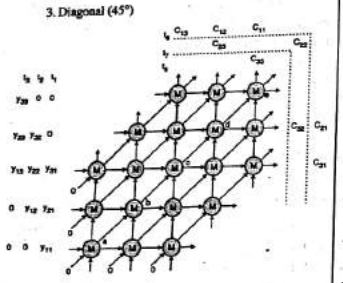
Inputs : x, y, z'

Output : x, y, z

Where $z = z' + x \times y$

Adjacency between cells is defined in three directions.

1. Horizontal
2. Vertical
3. Diagonal (45°)



$$\begin{matrix} x_{11} & x_{12} & x_{13} & 0 & 0 & t_1 \\ 0 & x_{21} & x_{22} & x_{23} & 0 & t_2 \\ 0 & 0 & x_{31} & x_{32} & x_{33} & t_3 \end{matrix}$$

**Fig. Ex. 6.4.1(a) : Pipelined multiplication of two 3 × 3 matrices using systolic arrays**

- The input matrices are fed into the array in the horizontal and vertical directions.
- Three clock periods are needed in feeding the matrices.

To illustrate the operations of the matrix multiplier, we can consider the computation of $z_{11}$.

$$z_{11} = x_{11} y_{11} + x_{12} y_{21} + x_{13} y_{31}$$

1. $x_{11}$ flows upwards through the cell a.
   $y_{11}$ flows right through the cell a.
   $x_{11} y_{11}$ flows diagonally from the cell a to cell b.
2. Cell b computes the value
   $x_{11} y_{11}$ (from the cell a) + $x_{12} y_{21}$ and sends it to the cell c.
3. Cell c computes the value
   $(x_{11} y_{11} + x_{12} y_{21})$ (from the cell b) + $x_{13} y_{31}$
4. Only 0 is added to $x_{11} y_{11} + x_{12} y_{21} + x_{13} y_{31}$ in cells d and e.

### 6.4.1 Issues in Vector Architecture

- As discussed earlier, the lengths of the vectors are most of the times different then the size of the vector registers.
- In case if the vector size is smaller than the vector register size, we can use the register of that length as the vector size.
- But in case if the vector size is greater than the size of the vector register size, the long vector is divided into small parts equal to the size of the vector registers. This is called as strip-mining.
- The block of vector data in the memory that will be taken to the vector registers, forms a stride. It is easy to handle a single stride at a given time.
- Caches also handle the stride together and hence bringing the entire stride block together into the cache.
- For data that is not of one stride, special stride registers are used to load such data of the vector register.
- Another issue in Vector processors is the implementation of the cache.
- Many supercomputer using vector processors have no cache in the system because of various problems faced. Let us discuss these problems.
  1. Most of the vector programs have very huge data vectors. By the time a data will be used again by the processor, the data is already over-written by another part of the data from the main memory. This is because the cache size is smaller than the vector data size, or the multiple vectors required to perform the operation.
  2. Also as just seen, the data is stored in strides in case of vector data, if the stride size does not matches the cache line size then the required data and the available data in cache may be different.
  3. The huge number of banks i.e. interleaved memories have been a better solution to give the multiple data simultaneously without any miss.



- There are three types of misses in case of a cache viz. Compulsory miss, capacity miss and conflict miss. Compulsory misses are due to the initial loading of the data into the cache, which is definitely going to happen i.e. there is no solution to this.
- The capacity miss is because of the overflow of the cache i.e. when the cache size is full. The solution to this is to have a huge size of cache memory. But as discussed earlier the vector data size is very huge, the cache size increase may not serve the purpose.
- Cache size increase causes a very huge increase in the cost.
- The conflict miss is when the two or more strides map into the same line of the cache. In such case there will be a conflict amongst which line must remain in the cache and hence result in a cache miss on every access if the two strides have to be accessed alternately.
- Every miss in the cache results in a huge number of processor cycles to stall the processor and this is equivalent to the memory access time.
- Some special types of mapping called as prime-mapped cache have shown some improvement in the performance of the vector processor systems.
- The addresses to be generated for accessing this kind of cache can be done in parallel and also it takes shorter time for the address to be generated. Hence the penalty reduces and performance increases.

### 6.4.2 Vector Performance Modelling

- There are two parameters to describe the performance of the vector processors, viz.
  1. Asymptotic performance or the theoretical peak performance ($r_\infty$)
  2. Half performance length ($n_{1/2}$)
- Theoretical peak performance is the maximum possible rate of computation that can be achieved by the processor and is expressed in FLOPS (Floating Point instructions per second).
- This parameter can be used to measure the performance of a single vector processor as well as multiple vector processors.
- For example, the $r_\infty$ of a single Cray Y-MP processor is 167 MFLOPS and that of a 8-processor system of Cray Y-MP is 2.6 GFLOPS.
- The half performance length is as the name says is the vector length for which the performance is half the peak performance.
- The performance of a vector processor depends on the vector start-up time and the pipeline depth. If these start-up time and pipeline depth keep on increasing, it becomes very difficult to attain the peak performance. So it is expected to reach atleast half the peak performance or the $n_{1/2}$ value.

Besides these parameters the basic performance measure for any multiprocessor system is the same that is the speed up factor.

- The speed up factor is given as the ratio of the execution time for one processor to that of the 'P' processors. It can also be said as the ratio of the speed of 'P' processors executing simultaneously to that of the single processor.

$$S_P = \frac{\text{Execution time using one processor}}{\text{Execution time using P processors}}$$

- The speciality of this performance parameter is that it considers the execution time and hence all the overhead of the parallel system are already taken into account.
- A very important point to be considered is that the same program is not to be tested for the parallel processors and single processor. This is because the algorithm to perform a task on a single processor and parallel processors will be different.
- Also when comparing the times required to execute the problem in single and parallel processor, the time to be considered on sequential processor must be the best algorithm time required.
- Hence we can say the speed-up ratio can be given as:

$$S_P = \frac{\text{Execution time for the best serial algorithm on one processor}}{\text{Execution time for parallel algorithm on P processors}}$$

## 6.5 Vectorizers and Optimizers

- Vectorization is done by a vectorizer. Vectorizer is nothing but a vector compiler. It converts the high level language program into a object code and vector instructions that can be executed in parallel.
- Thus the compiled code of an high level language can be converted to a program for vector processor. Hence, we can also say the vectorizer as a post compiler tool.

- The performance depends on the vectorization ratio. Vectorization ratio depends on the number of loops that can be converted to vector instructions to be executed in parallel.
- The main job of a vectorizer is to identify the "Do" loops, that are SIMD in function so that they can be allocated simultaneously to the different functional units.
- This process is called a vectorization and a process of efficient allocation of the same is called as optimization.
- Vectorization and then the optimization of the code mainly affects the performance of the vector processor.

### 6.5.1 Vectorization

- There are some important tasks that a vectorizer must be capable of performing. This list is given below :
  1. Locate the "Do" loops and identify the flow of the operation in that loop.
  2. Find out the loop variables and locate their independence.
  3. Locate the sequence of operation and their precedence.
  4. Finally, replace the loop with the vector instruction.
- Also in some cases the size of the vector is not decided during the compilation of the program. It is known only during the execution of the program. For example :

```
Do i=1:N
a[i] = b[i] + c[i]
```

- In this example the value of 'n' is not known to the compiler. It will be known to the system only on the execution of the program. There are two solutions to this as discussed earlier in section 6.4.1.
- The first solution is strip mining if the vector size is greater than the register size. The other is stride, which is required when all the elements in the vector are not placed nearby in the memory.
- The stride is the distance that separates the elements from each other in the vector stored in memory. This parameter helps getting the proper elements into the register, of the vector as required for a particular operation.

*Let us see some examples of conversion of the code into the vector form.*

**Ex. 6.5.1**

Convert the following code into vector form :

```
Do 10 I =1, N
Load R1, X(I)
Load R2,Y(I)
Multiply R1,S
Add R1,R2
Store Z(I),R1
10 Continue
```

**Soln. :**

This code can be interpreted as addition of two arrays in memory named as 'X' and 'Y'. The array 'X' is first multiplied by a constant 'C' and then added with the array 'Y'. Finally the result is stored in another array in the memory called as 'Z'. The vector instructions for this code can be written as shown below :

| Vector Instruction | Comments |
|---|---|
| M(x : x + N – 1) → V1 | Vector load from memory of the vector 'X' |
| M(y : y + N – 1) → V2 | Vector load from memory of the vector 'Y' |
| S × V1 → V1 | Multiply a scalar value 'S' with all the elements of vector V1 |
| V2 + V1 → V1 | Add the corresponding elements of the two vectors and store the result in one of those vectors |
| V2 → M(z : z + N – 1) | Store the result vector in the memory vector named 'Z' |

Here, the letter 'M' indicates memory load or store instruction. 'x', 'y' and 'z' indicate the starting index of the vectors 'X', 'Y' and 'Z' respectively.

In the first two instruction the vector registers are loaded with the memory vectors namely 'X' and 'Y'. Here the first task is to load the entire vector into the processor's vector registers, unlike the scalar processor wherein the first element of the vectors will be brought into the processor, operated on and then corresponding result stores, similarly for next element of the vector and so on.

The third instruction multiplies each element of the vector with a scalar value. Finally the two vectors are added and the result stored in one of the vector registers in the processor.



After all the computations are done the result is stored back in to the memory from the vector register that was holding the result.

For simplicity the above program can be written in a single statement as shown below. This format is called as compound vector functions.

$$Z(I) = C \times X(I) + Y(I)$$

where the index 'I' in the expression indicates that the expression involves 'N' elements.

Here, the index 'I' indicates that the value of this index has to change from 1 to 'N'. Hence the constant 'C' has to be multiplied with each of the element of the vector X(I) and then the product is to be added with the corresponding element of the 'Y' vector.

Finally the result is to be stored in the vector 'Z'. Some of these common vector functions are given in the Table Ex.6.5.1. These functions are to perform various basic operations like add, multiply, divide, shift etc.

**Table Ex. 6.5.1 : Standard vector instructions**

| Sr. No. | 1-D Vector functions |
|---|---|
| 1. | V1(I) = V2 (I) + V3(I) × V4(I) |
| 2. | V1(I) = B(I) + C(I) |
| 3 | A(I) = V1(I) × S + B(I) |
| 4. | A(I) = V1(I) + C(I) + B(I) |
| 5. | A(I) = B(I) × S + C(I) |
| 6. | A(I) = B(I) + C(I) + D(I) |
| 7. | A(I) = S × V1(I) × (Q × B(I) + C(I)) |
| 8. | A(I) = B(I) × C(I) + V1(I) × D(I) |
| 9. | A(I) = V1(I) + (1/A(I) + 1/B(I)) + Log(V2(I)) |

In this table, V(I) and V1(I) are vectors register A(I), B(I), C(I) and D(I) are vector stored in memory S and Q are scalar values.

**Ex. 6.5.2**

```
Do 10 I = 1,100,1
10 C(I) = A(I + 2) + B(I + 3)
```

**Soln. :**

| Vector Instruction | Comments |
|---|---|
| M(a + 2 : a + 101) → V1 | Vector load operation of vector A. |
| M(b + 3 : b + 102) → V2 | Vector load operation of vector B. |
| V1 + V2 → V3 | Add the vector registers V1 and V2. |
| V3 → M(c : c + 99) | Vector store V3 in memory vector C. |

**Ex. 6.5.3**

```
Do 10 I = 1,N
10 IF (L(1) . NE . 0) A(1 : N) = A(I : N) + 1
```

**Soln. :**

C(1 : 100) = A(3 : 102) + B(4 : 103)

| Vector Instruction | Comments |
|---|---|
| M(a : a + N – 1) → V1 | Vector load operation of vector A. |
| (L(1).NE . 0) V1 = V1 + 1 | Add 1 to each element of the vector register V1, if the value of L(1) is not equal to zero. |
| V1 → M(a : a + N – 1) | Vector store V1, back in memory vector A. |

**Ex. 6.5.4**

```
Do 20 I = 1, N
20 A(I) = B(I) + C(I)
```

**Soln. :**

| Vector Instruction | Comments |
|---|---|
| M(c : c + N – 1) → V1 | Vector load operation of vector C. |
| M(b : b + N – 1) → V2 | Vector load operation of vector B. |
| V1 + V2 → V3 | Add the vector registers V1 and V2. |
| V3 → M(a : a + N – 1) | Vector store V3 in memory vector A. |

**Ex. 6.5.5**

DIMENSION A (100), B (50), C (50)

C (1: 50) = A (1: 99: 2)* B (1: 50) + A (1: 99: 2)* C(1 : 50)

Obtain possible set of intermediate code and simplified code for vector processor.

**Soln. :**

| Vector Instruction | Comments |
|---|---|
| M(a : a + 99 : 2) → V1 | Vector load operation of alternate elements of vector A. The 'a' indicates index of first element, a+99, will obviously be the index of last element. The third argument i.e. 2, indicates that the vector is to be accessed in steps 2. Thus it will load the elements 1,3,5,...and so on it will load 50 elements. |
| M(b : b + 49) → V2 | Vector load operation of vector B. |
| M(c : c + 49) → V3 | Vector load operation of vector C. |
| V1 × V2 → V2 | Multiply the two vectors V1 and V2 and the result is stored in vector V2. |
| V1 × V3 → V3 | Multiply the two vectors V1 and V3 and the result is stored in vector V3. |
| V2 + V3 → V3 | Sum of the two product vectors is stored in the vector V3. |
| V3 → M(c : c + 49) | Vector store V3 in memory vector C. |

This can also be written in simple manner using the compound vector function as shown below :

$V1(I) = A((I \times 2) - 1)$

$C(I) = V1(I) \times B(I) + V1(I) \times C(I)$

***Ex. 6.5.6***

*Explain implementation of following loop in conventional scalar processor and vector processor.*

```
A(0) = X
Do 20 I = 1, N
20 A (I) = A (I – 1)* B (I) + C(I + 1)
```

***Soln. :***

*The first statement i.e. A(0) = X, will be executed sequentially in vector processor. Hence for this instruction there is no time difference in case of a scalar or vector processor.*

The next is a Do loop, which will be executed for 'N' times in a scalar processor, and hence the time taken is as required for 'N' operations given inside the loop i.e. 'N' multiplications and 'N' additions.

Besides this the element by element loading for the three vectors and the storage of one vector element by element will be required. Hence in the loop we will have three load operations, one multiple, one add and one store operation. All these six operations will be executed for 'N' times. This loop can be executed by the vector processor, by just one instruction to load all the elements of a vector.

Similarly the add and multiply operations can be done in a single instruction. All these operations will be done simultaneously, for example, all the additions will be done together by the multiple functional units in the ALU. Similarly for multiplications and load. Finally, once all the operations are done the resultant vector will also be stored together in a single instruction.

Hence the program will be executed without any branching, resulting in no branch penalty. Also all the ALU operations will be performed simultaneously and hence further saving of time. Let us see the vector instructions to perform the above task.

| Vector Instruction | Comments |
|---|---|
| X → A(0) | Store the constant 'X' in the vector 'A' as its first element. |
| M(a : a + N) → V1 | Vector load operation of vector A starting from index 0. |
| M(b : b + N – 1) → V2 | Vector load operation of vector B. |
| M(c + 1 : c + N – 2) → V3 | Vector load operation of vector C. |
| V1 × V2 → V2 | Multiply the two vectors V1 and V2 and the result is stored in vector V2. |
| V2 + V3 → V1 | Add the two vectors V2 (result of previous step) and V3 and the result is stored in vector V1. |
| V3 → M(a : a + N) | Vector store V3 in memory vector A. |



***Ex. 6.5.7***

*Explain implementation of following loop in conventional scalar processor and vector processor.*

```
Do 10 I = 1 : N
A (I) = B (I) + C (I)
B (I) = 2 * A (I + 1)
10 Continue
```

**Soln. :** The Do loop will be executed for 'N' times in a scalar processor, and hence the time taken is as required for 'N' operations given inside the loop i.e. 'N' multiplications and 'N' additions. Besides this the element by element loading for the three vectors and the storage of one vector element by element will be required. Hence in the loop we will have three load operations, one multiple, one add and one store operation. All these six operations will be executed for 'N' times.

This loop can be executed by the vector processor, by just one instruction to load all the elements of a vector. Similarly the add and multiply operations can be done in a single instruction. All these operations will be done simultaneously, for example, all the additions will be done together by the multiple functional units in the ALU. Similarly for multiplications and load.

Finally, once all the operations are done the resultant vector will also be stored together in a single instruction. Hence the program will be executed without any branching, resulting in no branch penalty. Also all the ALU operations will be performed simultaneously and hence further saving of time. Let us see the vector instructions to perform the above task.

| Vector Instruction | Comments |
|---|---|
| M(a : a + N – 1) → V1 | Vector load operation of vector A |
| M(b : b + N – 1) → V2 | Vector load operation of vector B |
| M(c : c + N – 1) → V3 | Vector load operation of vector C |
| M(a + 1 : a + N – 1) → V4 | Vector load operation of vector A from 2nd element |
| V2 + V3 → V1 | Add the two vectors V3 and V2 and the result is stored in vector V1. |
| V4 × 2 → V2 | Multiply the scalar quantity 2 to each of the element of the vector V4 and the result is stored in vector V2 |
| V1 → M(a : a + N – 1) | Vector store V1 in memory vector A |
| V2 → M(b : b + N – 1) | Vector store V2 in memory vector B |

### 6.5.2 Optimization

Optimization as discussed earlier is the process of efficient allocation of the vectors. There are various methods of optimization, which are broadly classified as General optimization, extended optimization and vector-extended optimization. Fig. 6.5.1 shows this classification chart.

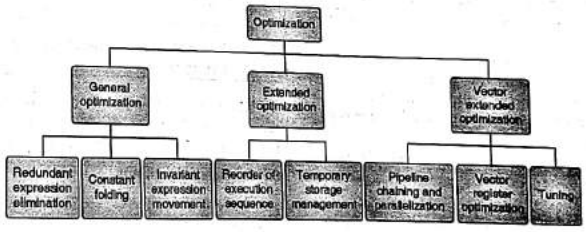



**Fig. 6.5.1 : Classification of different types of optimization**

Let us discuss these types of optimization in detail.

### 6.5.2.1 Redundant Expression Elimination

As the name says, the redundant expressions are eliminated from the code. This reduces not only the number of operations for the functional units but also removes the memory accesses.

## 6.6 Array Processor

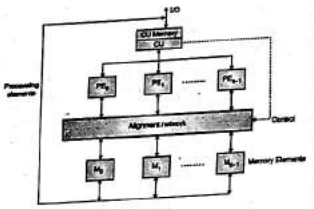



**Fig. 6.6.1 : Configuration of SMD array processor**

- An array processor is similar to a multiple ALU processor. It has an array of multiple Processing Elements (PEs).
- These processing element work under the supervision of one control unit. An array processor can handle multiple data streams and this is reason that it is known as SIMD processors.
- The control unit has its own memory for storage of programs. User program is loaded inside the CU memory. Scalar instructions are executed locally by CU, whereas vector instructions are executed with the help of array processor.
- A vector instruction is broadcasted to all PEs and vector elements are distributed to the memory elements. Memory elements are connected to PEs through an alignment network. PEs operate in parallel to perform a vector operation.

Before seeing the differences between the array processors and vector processors, let us see the similarities between the two processors.

- The basic principle of both array processor and the vector processor is that they operate on the SIMD system i.e. single instruction on multiple data.
- Both of them use the data parallelism and are designed to perform the operation on the same type of problems.

Let us see the differences between the two types of processor.

| Sr. No. | Vector Processor | Array Processor |
|---|---|---|
| 1. | In this case parallelism is achieved by multiple functional units in the ALU. | In case of array processor, parallelism is achieved by multiple processing elements or multiple ALUs. |
| 2. | Besides multiple functional units in the ALU, parallelism is also achieved by pipelining these functional units. | Here, no pipelining of processing elements or functional units is done. |
| 3. | The functional units need not communicate amongst themselves. | The processing elements need to communicate among each other. |
| 4. | At a time only single data can be accessed between the processor and memory. | There are multiple data streams to access the data between the processor and memory. |
| 5. | All the data is accessed from the memory which is shared by all the functional units of the processor. | Here communication among multiple processing elements is done by either shared memory or direct communication between the processing elements. |

## 6.7 Parallel Algorithms for Array Processors

As seen the array processors can operate on an array of data simultaneously and hence provide parallel processing. Let us see some algorithms implemented on array processor.

### 6.7.1 Scan Algorithms

These algorithms are as the name says that scan an array and perform the required operation. We will see the various applications of these scanning algorithms in the following sections.



### 6.7.1.1 Adding a Set of Elements of an Array

If we want to add a set of elements in an array on a single processor system, then it will take us O(n) time. The same thing can be ideally done on an array processor in O(n/p) time.

Since there are 'p' processors, each processor working on one element of the 'n' sized array simultaneously, the time required will be 1/p of what it is required on a single processor. A code to perform this operation of a array processors can be as given below:

```
total = segment[0];          //each processor takes the first
                                element in its segment as the initial
                             //total for(i = 1; i < segment.length; i++)
{
  total += segment[i];       //all the remaining elements of
                                the segment given to a processor are
                             // added and stored in the variable total
}
if(pid > 0)                  // for all the processors except for the
                                processor '0', send their
                             //individual total to the processor '0'
{
  send(0, total);
}
else
{
  for(int k = 1; k < procs; k++) //for processor '0' receive
                                   the total from each processor and add them.
  {
    Total += receive(k);
  }
}
```

In this program the variable "pid" is used to indicate the processor number or the identity of the processor number. The variable "procs" is the total number of processors in the parallel processing system.

The method or the function send() is used to send a value to another processor. The parameters passed with this function are the "pid" of the processor to whom the value is to be passed and the value to be passed.

Each processor is given a section of array to operate on called as "segment". The numbers written outside the circles are the "pid" in Fig. 6.7.1. Also in this Fig. 6.7.1, the elements of the arrays are taken as a, b, c and so on. And each array is given two elements of the array i.e. the segment size is two elements.

Initially the variable "total" is initialized to first element of the segment in each processor.

Then each processing element adds the elements in its segment (a part of the total array) using the first "for" loop seen in the program. The time taken for this is O(n / p) time, as expected by us. But then each processor has to forward its total to the processor "zero". This will require extra time for p - 1 transfers and their additions in the processor '0'. his is done in the second "for" loop and it will require O(p) time. Thus, the total time taken is O(n / p + p) i.e. more than what was expected. A diagrammatic representation of this method is shown in the Fig. 6.7.1.

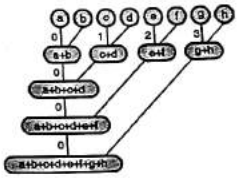



**Fig. 6.7.1 : A method for parallel scan**

Hence you will notice that the time for communication is much more than the time for actual computation. In this case the first "for" loop is for computation and the second "for" loop is for communication.

The first part i.e. the computation will take very less time as 'p' processors are working together, but the next "for" loop requires a lot of time. The second loop i.e. for communication, although for each processor is done simultaneously, but the processor '0' cannot accept or receive messages simultaneously.

This process of message receiving is done as one by one. For a small array, where the value of 'p' is small this time required for communication is not an issue. But for long arrays with huge value of 'p', the extra time is definitely a big concern.

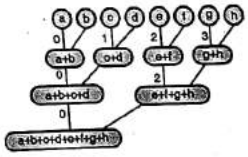



**Fig. 6.7.2**

Let us see another implementation to avoid the second loop. Fig. 6.7.2 shows how we can reduce the time required for communication. Here each alternate processor takes the element from the next processor, and adds them. Then in the next time unit, four alternate processors add, then eight alternate processors and so on.

Thus in the example shown in Fig. 6.7.2 which has the array of 4 processors, after adding the elements in the individual segments, the processors '0' and '2' perform the partial additions of the total from processors '0' with '1' and '2' with '3' respectively. In the next round the processor '0' performs the addition of the total from the processor '0' with '2'. Let us see the code for implementation of this logic.

```
total = segment[0];          //each processor takes the first
                             element in its segment as the
for(i = 1; i < segment length; i++)
{
    total += segment[i];     //all the remaining elements
                             of the segment given to a processor
                             // are added and stored in the variable total
}
for(int k = 1; k < procs; k *= 2)  // 'k' is always multiplied
                             by 2, hence its values will be 0,2,4,8
                             // and so on. This is done so that
                             we can select the processors
                             //at these interval for performing the
                             addition operation as
                             //discussed above
{
  if((pid & k) != 0)         // processors that are not supposed to
                             add in a given round
                             // are supposed to send their total and break,
                             because there after
                             // they are not involved in the further rounds
  {
    send(pid - k, total);
    break;
  }
  else if (pid + k < p)      // For processors that are still a
                             multiple of 'k' are supposed
                             // to add their own total with that of the
                             total received by them
                             // from their  total += receive(pid + k);
                             neighbour processor
  {

  }
}
```

*This algorithm will take $O(n/p + \log p)$ time, which is much better compared to the previous algorithm. Here the communication time is $O(\log p)$, where p is the number of processors.*

In our case, since there are 4 processors the time for communication is just 2. In case of huge arrays, this algorithm can give high performance.

We have seen in these algorithms how parallel addition is done in array processor and hence increase the speed of operation. All operations that follow the law of associativity can use this algorithm, for example, multiplication, finding minimum or maximum value from an array etc.

**Syllabus Topic : Multithreaded Processor**

## 6.8 Multi-threading

**Q. Write a short note on multithreaded processor. (5 Marks)**

- This is another very important feature supported by many Operating Systems (OS) that allows multiple threads or processes to be executed by the processor simultaneously on a time sharing basis.
- This also sometimes gives parallel processing and hence increasing the speed; especially because a task may be waiting for an operation to be completed and this time can be utilized by another task or thread.

**Syllabus Topic : Out-Of-Order Execution**

### 6.8.1 Dynamic Instruction Scheduling (or) Out-Of-Order (OOO) Execution

**Q. Explain how out-of order execution of instructions works. (5 Marks)**

**Q. Write short note on speculative execution. (5 Marks)**

- This is another interesting and very widely used technique because of the speed up given by it. It is used in Pentium IV processor.
- Here the execution of the instructions of a program is done out-of-order i.e. not in the sequence as the instructions were written by the programmer. As and when the resources of an instruction is available, the execution of that instruction is done. If, for an instruction the resources are not available, it is kept in waiting state and the further instructions whose resources are available will be executed.
- But, you would think that this approach will have a problem. The logic implemented by the programmer will not be followed properly i.e. wrong sequence of instructions will be executed. The answer to this is that, although the instructions are executed out-of-order, but the write-back is done in order, and hence the final result of the program is in sequence.
- The compiler is designed in such a way that, while translating from high-level language to machine language program, it detects the data dependencies and re-orders the instructions.
- If necessary to delay the loading of the conflicting data it inserts No-Operation instruction (NOP).



**Syllabus Topic : Speculative Execution**

### 6.8.2 Speculative Loading

- This is a process implemented in EPIC processors discussed in chapter 1. In this case the data is brought from the memory, well before it is needed.
- The compiler indicates the data that will be required in the later parts of the program and the corresponding data is brought and kept in the processor.
- This removes the latency of memory accesses required for the data to be brought from the memory.
- As the data required later is speculated and brought in advance it is called as speculative loading of data.

Multithreaded system is one that can make Massively Parallel Processor (MPP) system. There are various other architectures that are also capable of developing MPP. The major research in this area is on the latency hiding techniques. We will see these techniques in the next section. We will also see the principles and architecture of multithreaded architectures.

## 6.9 Latency Hiding Techniques

Latency means extra time or the time delay. The extra time is required to access the memory because the memories are comparatively slower than the processors is one major cause of latency. Also in a multiprocessor system many a times we have shared memory which has more latency. Hence it is necessary to either reduce this latency or atleast hide it from the processor. There are various latency hiding mechanisms we will be studying in this section. They are as listed below :

1. Pre-fetching technique i.e. bringing the instructions and the data before they are actually needed. Hence reducing the memory access time, or hiding the latency from the processors.
2. Multiple coherent caches that will reduce the cache misses.
3. Relaxed memory consistency models that allow buffering as well as pipelined access for memory accesses.
4. Multiple context support processors that allow switching from one context to another whenever the first one has a long latency. This one is nothing but multithreading.

We will see the first three in the subsections in this section, while the fourth one in the next section.

### 6.9.1 Pre-fetching Techniques

This method of data hiding brings the data and instructions before they are needed by the execution unit of the processor. But for this technique it is necessary to have a knowledge of the data and instructions that will be expected by the processor. Pre-fetching can be classified in two manners. The first classification is bounded and non-bounded pre-fetching. Another classification is on the control of this pre-fetching i.e. whether the control is hardware or software. We will see all these methods in the following sub sections.

#### 6.9.1.1 Bounded and Non-bounded Pre-fetching

- In this case as the name says there is some binding between the pre-fetching. For example, if an instruction accesses a data from a memory location whose address is given by a register pointer. And this register pointer is initialized in one of the instruction. This address or pointer is also to be pre-fetched.
- Here, if another processor modifies the memory location that initializes the reference or pointer during the delay between the pre-fetch of this address and the actual initialization of the reference (i.e. register pointer), then the pointer reference will be initialized with wrong value and hence wrong data will be accessed.
- This will also be true if the data that may not be a reference or pointer, but is pre-fetched before the actual execution of that instruction. Similar to the above case if that data is modified by another processor before the actual execution of this data then there will be a wrong operation i.e. a stale data (old data) will be considered instead of the most updated version of the data.

- A major disadvantage in such cases i.e. bounded pre-fetching, is getting a stale data.
- In this case also the data is brought into the processor before the actual execution of the instruction i.e. the data is pre-fetched. But another major care taken here to avoid the disadvantage of the bounded pre-fetching is that the cache coherency protocol keeps an eye on this data.
- Thus if another processor alters the value of this memory location which is pre-fetched, the cache coherency protocol monitors this and also updates the pre-fetched data by the processor. This totally removes the problem of stale data being pre-fetched.

### 6.9.1.2 Hardware and Software Controlled Pre-fetching

- Hardware controlled pre-fetch is automatic hardware system that fetches the instructions and data automatically into the cache based on the principles of locality of reference.
- But this type of pre-fetch has the limitation i.e. the data and instructions fetched are wrong in case of a branching.
- In case of software controlled pre-fetching there are instructions that are given to the processor to fetch the data in advance, so as to keep the buses occupied during the execution of ALU instructions.
- We know that most of the processors have pipelining implemented in them. In such cases, many a times the buses of the processor are free while the internal operations are carried out by the ALU and the internal cache.
- As we know principles of locality of reference assure 90% of the accesses from the cache memory. This makes the system bus free for most of the time, hence if the instructions are given for pre-fetching the data that will be required later, the operations will be faster.
- An important thing for software pre-fetching is that the processor must have instructions to support the instructions for pre-fetching.
- There must be special instructions in the instruction set of such processors, that allow pre-fetching. This can be said to be a disadvantage of software controlled pre-fetching i.e. the extra instructions and the circuit implementation for the same inside the processor or the overhead included in adding these instructions.
- The most important advantage of any type of pre-fetching is that the code is always available in the pipeline and hence the stages of the pipeline never starves.
- The experiments have been carried out that show that there is a drastic improvement in the performance when the pre-fetching is done compared with respect to when no pre-fetching is done.

### 6.9.2 Multiple Coherent Caches

- In case when we have a single cache for each processor, then the maintenance of coherency is very complicated. Hence many multiprocessor supporting processors do not have any cache.
- But there is another better solution that allows the use of cache and still give a higher performance.
- The multiple coherent system says cache the private and shared data. Since there are multiple processors operating on a task divided into multiple parts, they may access the same data.
- If the data is once cached for one processor, since this shared data is cached in all the caches, the processor need not face the miss. Hence no latency for the access of the shared data.
- We can have the shared and private data cacheable. This is seen to increase the performance substantially.
- The main reason being that this reduces the number of read and write misses in case of the shared data being cached. The statistical data shows the performance increase.
- One such implementation is called as Scalable Coherent Interface (SCI) specified as a standard by IEEE. This cache coherency model protocol maintains distributed directories in the individual processor cache module.
- Fig 6.9.1 shows the implementation of SCI protocol.

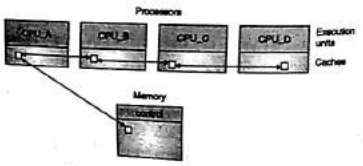



**Fig. 6.9.1 : SCI cache coherence protocol**



- In each processor cache there is a directory for the local or private data and a separate directory for the shared data. There are additional bits in the tag for each address to indicate the CPU. Hence, one can say that there is a doubly linked list maintained.
- Whenever a data is required by suppose CPU_A in the Fig. 6.9.1, it checks for the availability of this data in the caches of the other three processors besides its own cache. If it doesn't gets this data then it accesses it from the main memory.
- All other processors maintain the presence of this data in their coherent cache directories. Now if another processor requires the same data, it need not access the memory, instead it will be available from the cache as indicated in the coherent cache directory.

**Syllabus Topic : VLIW**

## 6.10 VLIW Processors

Q. Explain VLIW computing. **(5 Marks)**

VLIW (Very Long Instruction Word) has around 256 to 1024 bits per instruction. It is a combination of horizontal micro-coding and superscalar. It also has a large register file.

### 6.10.1 Horizontal vs. Vertical Micro-coding

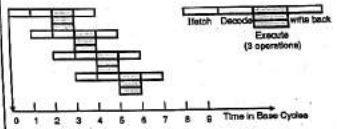
VLIW processor uses horizontal micro-coding. Let us understand how the horizontal micro-coding is different from vertical micro-coding.

| Sr. No. | Vertical Programming | Horizontal Programming |
|---|---|---|
| 1. | Decodes the instructions one by one as they are available | All the instructions are given simultaneously |
| 2. | This is comparatively slower. | This is comparatively faster |
| 3. | Less no. of bits for each instruction are used | More no. of bits for each instruction |
| 4. | Low degree of parallelism | High degree of parallelism |

### 6.10.2 VLIW Instruction and Pipelining

- The instruction format and the pipelining of VLIW processors is as shown in Fig. 6.10.1.
- As seen in Fig. 6.10.1(a), the VLIW instruction consists of many operations in a single instruction.
- Some operations like FP Add (Floating Point Addition), FP Multiply (Floating Point Multiplication), Branching instruction, Integer ALU operation etc.
- Hence on issuing one instruction itself many operations are done simultaneously during the execute cycle of the pipelining. This multiple operations in the execution stage of the VLIW processors is shown in the Fig. 6.10.1(b).


**(a) A typical VLIW processor and instruction format**




**(b) VLIW execution with degree m = 3**

**Fig. 6.10.1 : Instruction structure and pipelining of VLIW processors**

### 6.10.3 VLIW Processor Structure

- To perform multiple operations in a single execution stage, we need to have separate units to perform each of these operations.
- To perform the different operations like floating point add, multiply, branching, integer ALU etc, we need separate units for each of these operations this is shown in the Fig. 6.10.2.
- Fig. 6.10.2 shows the architecture of a typical VLIW processor with all the above mentioned units that can follow the instruction format and pipelining given in the Fig. 6.10.1.

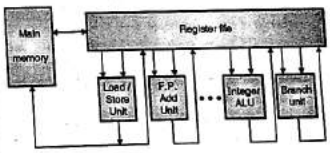



**Fig. 6.10.2 : Typical VLIW architecture**

The special characteristic of a traditional VLIW Processor is that the instruction has multiple operations. The multiple operations given in the instruction are independent operations and have no flow dependences between these operations.

**Advantages**

(a) No runtime dependence checks against previously or simultaneously issued operations.

(b) No runtime scheduling decisions.

(c) No need for register renaming.

**Disadvantages**

(a) No tolerance for any difference in the types of functional units.

(b) No object code compatibility.

In Superscalar and VLIW processors, more than a single instruction can be issued to the execution units per cycle.

- Superscalar machines are able to dynamically issue multiple instructions each clock cycle from a conventional linear instruction stream.
- VLIW processors use a long instruction word that contains a usually fixed number of instructions that are fetched, decoded, issued, and executed synchronously.
- Hence, Superscalar has dynamic issue, while VLIW has static issue.

**Syllabus Topic : Data Flow Computing**

## 6.11 Data Flow Computers

**Q. Explain Data flow computing. (5 Marks)**

- A data flow computer has the instructions executed according to the availability of data.
- Any instruction should be ready for execution whenever the data is available. Instruction execution is independent of the physical location of that instruction in the program memory.
- Since the instructions need not be ordered in the sequence of execution, there is no need of the Program Counter (PC). The control-flow computers use shared memory to store the data and program, which may cause errors in other instruction or data while executing a particular instruction.
- In case of data flow computers, there is no need of shared memory as the instructions are stored in the instruction itself. Hence there are no problems related to the shared memory.

### 6.11.1 Data Flow Graphs

- Data flow graphs are used to represent programs for data driven computations. An example of data flow graph to perform the operation $z = (x - y) * 2$, is shown in Fig. 6.11.1.

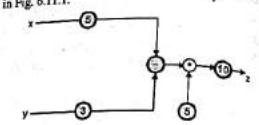



**Fig. 6.11.1 : Data flow graph of $z = (x - y) * 5$**

- As shown in the Fig. 6.11.1, the inputs x and y are given to the subtract instruction. When the result operand is ready, the instruction for multiplication is being executed.
- The template implementation of the above data flow graph can be as shown in Fig. 6.11.2.

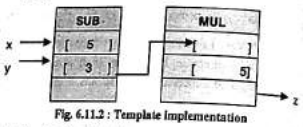
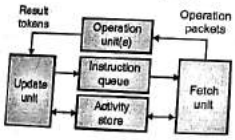


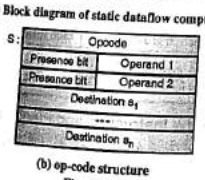

**Fig. 6.11.2 : Template implementation**

### 6.11.2 Static Dataflow

- It combines control and data into a template like a reservation station, except that they are held in memory. They can inhibit parallelism among loop iterations and also re-use of template.




**(a) Block diagram of static dataflow computer**




**(b) op-code structure**

**Fig. 6.11.3**



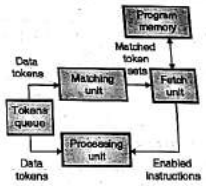
- The block diagram of the static dataflow computer is shown in Fig. 6.11.3(a). The different blocks of the same are explained below.

1. **Instruction Queue :** When the instruction becomes ready for execution, the address of the activity template is entered in the instruction queue. Each activity template has unique address.
2. **Fetch and Update Units :** Instruction fetching and data accessing operations are performed by fetch and update units.
3. **Operation Unit :** The specified operation is performed by the operation unit. The generated result is passed to each destination field in the template.
4. **Activity store :** This stores the activity templates.

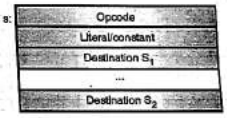
- The Fig. 6.11.3(b) shows the structure of an opcode. The opcode has the operands and the corresponding presence bits to indicate if the operands are presented (ready) or not.
- As shown in Fig. 6.11.3(b), an instruction consists of opcode, operands and their presence bits and the destinations of the result. Thus when an instruction is executed, all those instruction's operand fields are updated which are the destination for the executed instruction.
- Also, the corresponding presence bits are set, to indicate if the operands are available or not.
- The fetch unit fetches the instruction from the instruction queue. If the instruction has both the presence bit set, it is forwarded to operation unit. If the operands are not ready (i.e. presence bit is clear), the instruction is given back to the queue.
- The operation unit operates on the instructions whose operands are available and then forwards it to the update unit. The update unit, updates those instruction's operand fields which are the destination for the executed instruction.

### 6.11.3 Dynamic Dataflow

- Dynamic dataflow computers have separate data tokens and control. It has a tagged token i.e. a labeled packet of information.
- The data token is tagged with the context descriptor and is then called as a tagged token.
- The tagging is achieved by attaching a label with each token. The label identifies the context of that token.
- This allows multiple iterations to be simultaneously active with shared control (instruction) and separate data tokens. A data token can carry a loop iteration number.
- The operation can be described as below :
  1. Match token's tags in matching store via associative search. If match not found, make entry and wait for partner. This requires large associative search to match tags.
  2. When there is a match, fetch corresponding instruction from program memory and execute the instruction.
- The block diagram and the opcode structure of the dynamic dataflow computer is shown in Fig. 6.11.4.




**(a) Block Diagram of Dynamic Data Flow Computer**




**(b) Opcode structure**

**Fig. 6.11.4**

**Advantages**

1. No program counter
2. Data-driven
3. Execution inhibited only by true data dependences
4. Stateless / side-effect free
5. Further enhances parallelism

**Disadvantages**

1. No program counter leads to very long fetch/execute latency
2. Spatial locality in instruction-fetch is hard to exploit
3. Requires matching (Example, via associative compares)
4. No shared data structures
5. No pointers into data structures (implies state)

**Syllabus Topic : Introduction to Multi-core Processor Architecture**

## 6.12 Comparative Study of Multi-core Processors I3, I5 and I7

Q. Compare I3 and I5 multi-core processors. **(5 Marks)**

The second generation microprocessors of the Intel core i3, i5 and i7 processors are the ones we normally seen in the computers today. The comparison of the same is given in the Table 6.12.1.

**Table 6.12.1 : Comparison of I3, I5 and I7 processors**

| Sr. No. | Feature | I3 | I5 | I7 | I7 Extreme |
|---|---|---|---|---|---|
| 1. | Number of cores | 2 for desktop as well as for laptop | 4 for Desktop<br>2 for Laptop | 4 or 6 for Desktop<br>2 or 4 for Laptop | 6 for desktop<br>4 for mobile |
| 2. | Processing threads | 4 for desktop as well as laptop | 8 threads for desktop 4 threads for Laptop | 8 or 12 threads for desktop 4 or 8 threads for laptop | 12 threads for Desktop<br>8 threads for laptop |
| 3. | Maximum base clock frequency | 3.4GHz | 3.4GHz | 3.2GHz | 3.3GHz |
| 4. | Maximum turbo boost frequency | Not Applicable | 3.8GHz | 3.8GHz | 3.9GHz |
| 5. | Maximum smart cache size | 3MB | 6MB | 12MB | 15MB |
| 6. | Intel turbo boost 2.0 | Not present | Present | Present | Present |
| 7. | Intel Hyperthreading | Present | Present only in Laptop processors | Present | Present |
| 8. | Best Desktop processor | Intel Core i3-2130 (3.4GHz, 3MB) | Intel Core i5-2550K (3.4GHz, 6MB) | Intel Core i7-3930 (3.2GHz, 12MB) | Intel Core i7-3960 (3.3GHz, 15MB) |
| 9. | Best Mobile (Laptop) processor | Intel Core i3-2370 (2.4GHz, 3MB) | Intel Core i5-2540M (2.6GHz, 3MB) | Intel Core i7-2860 (2.5GHz, 8MB) | Intel Core i7-2960XM (2.7GHz, 8MB) |

## 6.13 Exam Pack (University and Review Questions)

**Syllabus Topic : Flynn's Classifications**

Q. Name the Flynn's classification of parallel processing systems. *(Ans. : Refer section 6.2)* **(May 2014, 3 Marks)**

Q. List the Flynn's Classification of Parallel Processing Systems. *(Ans. : Refer section 6.2)* **(May 2015, 3 Marks)**

Q. Explain Flynn's classification. *(Ans. : Refer section 6.2)* **(Dec. 2015, 10 Marks)**

Q. Explain Flynn's classification in detail. *(Ans. : Refer section 6.2)* **(May 2016, 10 Marks)**

Q. Explain Flynn's Classification. *(Ans. : Refer section 6.2)* **(Dec. 2016, 5 Marks)**

Q. Explain Flynn's classification. *(Ans. : Refer section 6.2)* **(May 2017, 10 Marks)**

Q. Write a short note on Flynn's classification of parallel computing. *(Ans. : Refer section 6.2)* **(5 Marks)**

**Syllabus Topic : Concepts of superscalar architecture**

Q. Write a short note on Superscalar architecture. *(Ans. : Refer section 6.3)* **(5 Marks)**



**Syllabus Topic : Multithreaded processor**

Q. Write a short note on multithreaded processor. *(Ans. : Refer section 6.8)* **(5 Marks)**

**Syllabus Topic : Out-Of-Order Execution**

Q. Explain how out-of order execution of instructions works. *(Ans. : Refer section 6.8.1)* **(5 Marks)**

Q. Write short note on speculative execution. *(Ans. : Refer section 6.8.1)* **(5 Marks)**

**Syllabus Topic : VLIW**

Q. Explain VLIW computing. *(Ans. : Refer section 6.10)* **(5 Marks)**

**Syllabus Topic : Data flow computing**

Q. Explain Data flow computing. *(Ans. : Refer section 6.11)* **(5 Marks)**

**Syllabus Topic : Introduction to Multi-core processor architecture**

Q. Compare I3 and I5 multi-core processors. *(Ans. : Refer section 6.12)* **(5 Marks)**

□□□