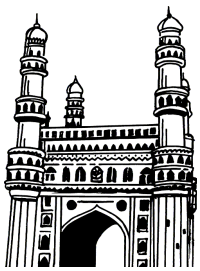
Rahul's ✔
Topper's Voice



M.C.A.

I Year I Sem

(Osmania University)

Latest 2024 Edition

COMPUTER ARCHITECTURE

- ☞ **Study Manual**
- ☞ **FAQ's and Important Questions**
- ☞ **Solved Model Papers**
- ☞ **Solved Previous Question Papers**

- by -

WELL EXPERIENCED LECTURER

Price 209-00

Rahul Publications™

Hyderabad. Cell : 9391018098, 9505799122

M.C.A.

I Year I Sem

(Osmania University)

COMPUTER ARCHITECTURE

Inspite of many efforts taken to present this book without errors, some errors might have crept in. Therefore we do not take any legal responsibility for such errors and omissions. However, if they are brought to our notice, they will be corrected in the next edition.

Price `. 209 -00

Sole Distributors :

Cell : 9391018098, 9505799122

VASU BOOK CENTRE

Shop No. 2, Beside Gokul Chat, Koti, Hyderabad.

Maternity Hospital Opp. Lane, Narayan Naik Complex, Koti, Hyderabad.

Near Andhra Bank, Subway, Sultan Bazar, Koti, Hyderabad -195.

CONTENTS

COMPUTER ARCHITECTURE

STUDY MANUAL

SOLVED MODEL PAPERS

SOLVED PREVIOUS QUESTION PAPERS

SYLLABUS

UNIT - I

Data Representation: Data types, Complements, Fixed and Floating Point representations, and Binary codes.

Overview of Computer Function and Interconnections: Computer components, Interconnection structures, Bus interconnection, Bus structure, and Data transfer.

UNIT - II

Register Transfer Micro operations: Register Transfer Language, Register Transfer, Bus and Memory Transfers, Arithmetic, Logic and Shift micro operations, Arithmetic Logic Shift Unit.

Basic Computer Organization and Design: Instruction Codes, Computer Registers, Computer Instructions, Timing and Control, Instruction Cycle, Memory reference instruction, Input-Output and Interrupt.

UNIT - III

Micro programmed Control: Control memory, Address Sequencing, Micro program example, Design of Control Unit.

Central Processing Unit: General Register Organization, Stack Organization, Instruction formats, Addressing modes, Data Transfer and Manipulation, and Program control.

Computer Arithmetic: Addition and Subtraction, Multiplication, Division, and Floating Point Arithmetic Operations.

UNIT - IV

Memory Organization: Memory Hierarchy, Main Memory, RAM and ROM, Auxiliary memory, Associative memory, Cache memory, Virtual memory, Memory Management hardware.

UNIT - V

Input-Output Organization: Peripheral Devices, Input-Output Interface, Asynchronous data transfer, Modes of Transfer, Priority Interrupt, Direct Memory Access (DMA), I/O Processor, Serial Communication.

Pipeline Processing: Arithmetic, Instruction and RISC Pipelines.

Assessing and Understanding Performance: CPU performance and its factors, Evaluating performance.

Contents

Topic No. **Page No.**

UNIT - V

Frequently Asked & Important Questions

UNIT - I

1. Describe how data is stored in the digital computers.

Ans : **(Imp.)**

Refer Unit-I, Page No. 1, Q.No. 1.

---

2. What is the complement number system? Explain the complement systems with examples.

Ans : **(Nov.-23, Aug.-21, Imp.)**

Refer Unit-I, Page No. 8, Q.No. 4.

---

3. Explain about fixed and floating point representation of numbers.

Ans : **(May-23, Imp.)**

Refer Unit-I, Page No. 10, Q.No. 5.

---

4. Write about various types of binary codes.

Ans : **(Nov.-23, Aug.-21, Imp.)**

Refer Unit-I, Page No. 13, Q.No. 6.

---

5. What is digital computer? Explain the block diagram of it.

Ans : **(Imp.)**

Refer Unit-I, Page No. 17, Q.No. 7.

---

6. What is the use of bus interconnection? Explain the types of bus organization.

Ans : **(May-23, Aug.-21, Imp.)**

Refer Unit-I, Page No. 20, Q.No. 9.

---

7. Explain three modes of data transfer between I/O devices and a computer system.

Ans : **(Nov.-23, Aug.-21, Imp.)**

Refer Unit-I, Page No. 24, Q.No. 11.

UNIT - II

1. Explain the Register Transfer Language.

Ans : **(Imp.)**

Refer Unit-II, Page No. 27, Q.No. 1.

2. Design and explain a common bus system for four register.

Ans : **(Nov.-23, Imp.)**

Refer Unit-II, Page No. 29, Q.No. 3.

3. Write various Arithmetic Micro-operation.

Ans : **(Imp.)**

Refer Unit-II, Page No. 34, Q.No. 7.

4. Explain briefly about Binary Incrementer

Ans : **(Imp.)**

Refer Unit-II, Page No. 36, Q.No. 10.

5. Draw and explain Logic Micro-operations in detail.

Ans : **(Imp.)**

Refer Unit-II, Page No. 39, Q.No. 12.

6. Draw and explain one stage of arithmetic logic shift unit.

Ans : **(Imp.)**

Refer Unit-II, Page No. 45, Q.No. 15.

7. Explain Stored Program Organization in detail.

Ans : **(Imp.)**

Refer Unit-II, Page No. 48, Q.No. 17.

8. Explain briefly about Registers of basic computer.

Ans : **(Imp.)**

Refer Unit-II, Page No. 50, Q.No. 19.

9. Explain different types of Computer Instructions formats.

Ans : **(Imp.)**

Refer Unit-II, Page No. 53, Q.No. 21.

10. Draw and explain input-output configuration of basic computer.

Ans : **(Imp.)**

Refer Unit-II, Page No. 61, Q.No. 26.

11. List and explain memory reference instructions.

Ans : **(Aug.-21, Imp.)**

Refer Unit-II, Page No. 58, Q.No. 25.

UNIT - III

1. Explain the steps of Address Sequencing in detail.

Ans : **(Imp.)**

Refer Unit-III, Page No. 68, Q.No. 4.

2. Draw the diagram of Micro programmed sequencer for a control memory and explain it.

Ans : **(Imp.)**

Refer Unit-III, Page No. 71, Q.No. 7.

3. What is central processing unit. Explain about it.

Ans : **(May-23, Imp.)**

Refer Unit-III, Page No. 73, Q.No. 8.

4. What is stack? Give the organization of register stack with all necessary elements and explain the working of push and pop operations.

Ans : **(Nov.-23, May-23, Imp.)**

Refer Unit-III, Page No. 77, Q.No. 10.

5. What are status register bits? Draw and explain the block diagram showing all status registers.

Ans : **(Imp.)**

Refer Unit-III, Page No. 91, Q.No. 16.

6. Explain the procedure for Addition and Subtraction with signed-magnitude data with the help of flowchart.

Ans : **(Aug.-21, Imp.)**

Refer Unit-III, Page No. 94, Q.No. 19.

UNIT - IV

1. How main memory is useful in computer system?

Ans : **(Aug.-21, Imp.)**

Refer Unit-IV, Page No. 108, Q.No. 2.

2. Explain the memory address map of RAM and ROM.

Ans : **(Imp.)**

Refer Unit-IV, Page No. 110, Q.No. 3.

3. Explain briefly about Content Addressable Memory (CAM).

Ans : **(Nov.-23, May-23, Imp.)**

Refer Unit-IV, Page No. 113, Q.No. 5.

4. Define Cache memory. Discuss associative mapping in organization of cache memory.

Ans : **(Imp.)**

Refer Unit-IV, Page No. 115, Q.No. 6.

5. What is segment? What is logical address? Explain segmented page mapping.

Ans : **(May-23, Aug.-21, Imp.)**

Refer Unit-IV, Page No. 122, Q.No. 12.

6. Write about memory management hardware.

Ans : **(Imp.)**

Refer Unit-IV, Page No. 123, Q.No. 13.

UNIT - V

1. Define Peripherals. Explain I/O Bus and Interface Modules.

Ans : **(Imp.)**

Refer Unit-V, Page No. 126, Q.No. 1.

2. What do you mean by Asynchronous data transfer? Explain Strobe control in detail.

Ans : **(Nov.-23, Aug.-21, Imp.)**

Refer Unit-V, Page No. 129, Q.No. 3.

3. What is priority interrupt? Explain briefly about Daisy Chaining Priority.

Ans : **(Imp.)**

Refer Unit-V, Page No. 134, Q.No. 7.

4. Write a detailed note on Direct Memory Access (DMA).

Ans : **(Nov.-23, May-23, Aug.-21, Imp.)**

Refer Unit-V, Page No. 136, Q.No. 8.

5. What is Serial Communication?

Ans : **(Imp.)**

Refer Unit-V, Page No. 140, Q.No. 11.

6. Explain Flynn's classification for computers.

Ans : **(Imp.)**

Refer Unit-V, Page No. 144, Q.No. 12.

7. Draw and explain Arithmetic Pipeline.

Ans : **(Nov.-23, Imp.)**

Refer Unit-V, Page No. 147, Q.No. 14.

8. Explain the Instruction Pipelining with example.

Ans : **(May-23, Aug.-21, Imp.)**

Refer Unit-V, Page No. 148, Q.No. 15.

9. What are the factors affecting the performance of the CPU ?

Ans : **(Aug.-21, Imp.)**

Refer Unit-V, Page No. 153, Q.No. 17.

UNIT I

Data Representation:

Data types, Complements, Fixed and Floating Point representations, and Binary codes.

Overview of Computer Function and Interconnections:

Computer components, Interconnection structures, Bus interconnection, Bus structure, and Data transfer.

1.1 Data Representation

1.1.1 Data Types

Q1. Describe how data is stored in the digital computers.

Ans : **(Imp.)**

- Binary information in digital computers is stored in memory or processor registers.
- Registers contain either data or control information.
- Control information is a bit or a group of bits used to specify the sequence of command signals needed for manipulation of the data in other registers.
- Data are numbers and other binary coded information that are operated on to achieve required computational results.
- The data types found in the registers of digital computers may be classified as one of the following categories:
  1. numbers used in arithmetic computations,
  2. letters of the alphabet used in data processing. and
  3. other discrete symbols used for specific purposes.

All types of data, except binary numbers, are represented in computer registers in binary. This is because registers are made up of flip-flops and flip-flops are two-state devises that can store only l's and O's.

Bits, bytes, nibble and word

- The terms bits, bytes, nibble and word are used widely in reference to computer memory and data size.
- **Bits:** can be defined as either a binary, which can be 0, or 1.It is the basic unit of data or information in digital computers.
- **Byte:** a group of bits (8 bits) used to represent a character. A byte is considered as the basic unit of measuring memory size in computer.
- **A nibble**: is half a byte, which is usually a grouping of 4 bytes.
- **Word:** two or more bits make a word. The term word length is used as the measure of the number of bits in each word. For example, a word can have a length of 16 bits, 32 bits, 64 bits etc.

Q2. What are the various number systems used in Computer?

Ans :

Number systems are the technique to represent numbers in the computer system architecture, every value that you are saving or getting into/from computer memory has a defined number system.

Computer architecture supports following number systems.

- Binary number system
- Octal number system
- Decimal number system
- Hexadecimal (hex) number system

1. Binary Number System

A Binary number system has only two digits that are 0 and 1. Every number (value) represents with 0 and 1 in this number system. The base of binary number system is 2, because it has only two digits.

2. Octal number system

Octal number system has only eight (8) digits from 0 to 7. Every number (value) represents with 0,1,2,3,4,5,6 and 7 in this number system. The base of octal number system is 8, because it has only 8 digits.

3. Decimal number system

Decimal number system has only ten (10) digits from 0 to 9. Every number (value) represents with 0,1,2,3,4,5,6, 7,8 and 9 in this number system. The base of decimal number system is 10, because it has only 10 digits.

4. Hexadecimal number system

Number system	Base	Used digits	Example	C Language assignment
Binary	2	0,1	$(11110000)_2$	int val=0b11110000;
Octal	8	0,1,2,3,4,5,6,7	$(360)_8$	int val=0360;
Decimal	10	0,1,2,3,4,5,6,7,8,9	$(240)_{10}$	int val=240;
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9, A,B,C,D,E,F	$(F0)_{16}$	int val=0xF0;

A Hexadecimal number system has sixteen (16) alphanumeric values from 0 to 9 and A to F. Every number (value) represents with 0,1,2,3,4,5,6, 7,8,9,A,B,C,D,E and F in this number system. The base of hexadecimal number system is 16, because it has 16 alphanumeric values. Here A is 10, B is 11, C is 12, D is 13, E is 14 and F is 15.

Table of the Numbers Systems with Base used Digits, Representation C language representation

Number System Conversions

There are three types of conversion:

- **Decimal Number System to Other Base**

  [for example: Decimal Number System to Binary Number System]

- **Other Base to Decimal Number System**

  [for example: Binary Number System to Decimal Number System]

- **Other Base to Other Base**

  [for example: Binary Number System to Hexadecimal Number System]

Decimal Number System to Other Base

To convert Number system from Decimal Number System to Any Other Base is quite easy; you have to follow just two steps:

A) Divide the Number (Decimal Number) by the base of target base system (in which you want to convert the number: Binary (2), octal (8) and Hexadecimal (16)).

B) Write the remainder from step 1 as a Least Signification Bit (LSB) to Step last as a Most Significant Bit (MSB).

Decimal to Binary Conversion	**Result**
Decimal Number is : $(12345)_{10}$	Binary Number is $(11000000111001)_2$

2	12345	1	**LSB**
2	6172	0	
2	3086	0	
2	1543	1	
2	771	1	
2	385	1	
2	192	0	
2	96	0	
2	48	0	
2	24	0	
2	12	0	
2	6	0	
2	3	1	
	1	1	**MSB**

Decimal to Octal Conversion	**Result**
Decimal Number is : **$(12345)_{10}$**	Octal Number is **$(30071)_8$**

8	12345	1	LSB
8	1543	7	
8	192	0	
8	24	0	
	3	3	MSB

<table>
<tr><th>Decimal to Hexadecimal Conversion</th><th>Result</th></tr>
<tr><td>Example 1
Decimal Number is : $(12345)_{10}$
<table><tr><td>16</td><td>12345</td><td></td><td>9</td><td>LSB</td></tr><tr><td>16</td><td>771</td><td></td><td>3</td><td></td></tr><tr><td>16</td><td>48</td><td></td><td>0</td><td></td></tr><tr><td>8</td><td>3</td><td></td><td>3</td><td>MSB</td></tr></table></td><td>Hexadecimal Number is
$(3039)_{16}$</td></tr>
<tr><td>Example 2
Decimal Number is : $(725)_{10}$
<table><tr><td>16</td><td>725</td><td></td><td>5</td><td>5</td><td>LSB</td></tr><tr><td>16</td><td>45</td><td></td><td>13</td><td>D</td><td></td></tr><tr><td></td><td>2</td><td></td><td>2</td><td>2</td><td>MSB</td></tr></table></td><td>Hexadecimal Number is
$(2D5)_{16}$
Convert
10, 11, 12, 13, 14, 15
to its equivalent...
A, B, C, D, E, F</td></tr>
</table>

Other Base System to Decimal Number Base

To convert Number System from Any Other Base System to Decimal Number System, you have to follow just three steps:

A) Determine the base value of source Number System (that you want to convert), and also determine the position of digits from LSB (first digit's position – 0, second digit's position – 1 and so on).

B) Multiply each digit with its corresponding multiplication of position value and Base of Source Number System's Base.

C) Add the resulted value in step-B.

Explanation regarding examples:

Below given exams contains the following rows:

A) Row 1 contains the **DIGITs** of number (that is going to be converted).

B) Row 2 contains the **POSITION** of each digit in the number system.

C) Row 3 contains the multiplication: **DIGIT* BASE ^ POSITION**.

D) Row 4 contains the calculated result of **step C**.

E) And then add each value of **step D**, resulted value is the Decimal Number.

Binary to Decimal Conversion

1	1	0	0	0	0	0	0	1	1	1	0	0	1
13	12	11	10	9	8	7	6	5	4	3	2	1	0
$1x2^{13}$	$1x2^{12}$	$0x2^{11}$	$0x2^{10}$	$0x2^{9}$	$0x2^{8}$	$0x2^{7}$	$0x2^{6}$	$1x2^{5}$	$1x2^{4}$	$1x2^{3}$	$0x2^{2}$	$0x2^{1}$	$1x2^{0}$
8192	4096	0	0	0	0	0	0	32	16	8	0	0	1

=8192+4096+32+16+8+1

=12345

Octal to Decimal Conversion	Result
3 0 0 7 1 4 3 2 1 0 $3*8^4$ $0*8^3$ $0*8^2$ $7*8^1$ $1*8^0$ 12288 0 0 56 1	**= 12288+0+0+56+1** **= 12345** Decimal Number is: **$(12345)_{10}$**

Hexadecimal to Decimal Conversion	Result
Hexadecimal Number is : **$(2D5)_{16}$** 2 D (13) 5 2 1 0 $2*16^2$ $13*16^1$ $5*16^0$ 512 208 5	**= 512+208+5** **= 725** Decimal Number is: **$(725)_{10}$**

Other Base System to Non-Decimal System

Steps

- **Step 1:** Convert the original number to a decimal number (base 10).
- **Step 2:** Convert the decimal number so obtained to the new base number.

Example

Octal Number – 25_8

Calculating Binary Equivalent –

Step 1 – Convert to Decimal

Step	Octal Number	Decimal Number
Step 1	25_8	$((2 \times 8^1) + (5 \times 8^0))_{10}$
Step 2	25_8	$(16 + 5)_{10}$
Step 3	25_8	21_{10}

Octal Number – 25_8 = Decimal Number – 21_{10}

Step 2 – Convert Decimal to Binary

Step	Operation	Result	Remainder
Step 1	21 / 2	10	1
Step 2	10 / 2	5	0
Step 3	5 / 2	2	1
Step 4	2 / 2	1	0
Step 5	1 / 2	0	1

Decimal Number – 21_{10} = Binary Number " 10101_2

Octal Number – 25_8 = Binary Number " 10101_2

Shortcut Method - Binary to Octal

Steps

- **Step 1** – Divide the binary digits into groups of three (starting from the right).
- **Step 2** – Convert each group of three binary digits to one octal digit.

Example

Binary Number – 10101_2

Calculating Octal Equivalent –

Step	Binary Number	Octal Number
Step 1	10101_2	010 101
Step 2	10101_2	2_8 5_8
Step 3	10101_2	25_8

Binary Number – 10101_2 = Octal Number – 25_8

Shortcut Method - Octal to Binary

Steps

- **Step 1** – Convert each octal digit to a 3 digit binary number (the octal digits may be treated as decimal for this conversion).
- **Step 2** – Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Example

Octal Number – 25_8

Calculating Binary Equivalent –

Step	Octal Number	Binary Number
Step 1	25_8	$2_{10}\ 5_{10}$
Step 2	25_8	$010_2\ 101_2$
Step 3	25_8	010101_2

Octal Number – 25_8 = Binary Number – 10101_2

Shortcut Method - Binary to Hexadecimal

Steps

- **Step 1** – Divide the binary digits into groups of four (starting from the right).
- **Step 2** – Convert each group of four binary digits to one hexadecimal symbol.

Example

Binary Number – 10101_2

Calculating hexadecimal Equivalent –

Step	Binary Number	Hexadecimal Number
Step 1	10101_2	$0001\ 0101$
Step 2	10101_2	$1_{10}\ 5_{10}$
Step 3	10101_2	15_{16}

Binary Number – 10101_2 = Hexadecimal Number – 15_{16}

Shortcut Method - Hexadecimal to Binary

Steps

- **Step 1** – Convert each hexadecimal digit to a 4 digit binary number (the hexadecimal digits may be treated as decimal for this conversion).
- **Step 2** – Combine all the resulting binary groups (of 4 digits each) into a single binary number.

Example

Hexadecimal Number – 15_{16}

Calculating Binary Equivalent –

Step	Hexadecimal Number	Binary Number
Step 1	15_{16}	$1_{10}\ 5_{10}$
Step 2	15_{16}	$0001_2\ 0101_2$
Step 3	15_{16}	00010101_2

Hexadecimal Number " 15_{16} = Binary Number " 10101_2

Q3. Write about other coding schemes in number system.

Ans :

Symbolic representation using coding schemes

- In computing, a single character such as a letter, a number or a symbol is represented by a group of bits. The number of bits per character depends on the *coding* scheme used.
- The most common coding schemes are:
- Binary Coded Decimal (BCD),
- Extended Binary Coded Decimal Interchange Code (EBCDIC) and
- American Standard Code for Information Interchange (ASCII).

Binary Coded Decimal

- Binary Coded Decimal is a 4-bit code used to represent numeric data only. For example, a number like 9 can be represented using Binary Coded Decimal as 1001_2 .
- Binary Coded Decimal is mostly used in simple electronic devices like calculators and microwaves. This is because it makes it easier to process and display individual numbers on their Liquid Crystal Display (LCD) screens.
- **A standard Binary Coded Decimal**, an enhanced format of Binary Coded Decimal, is a 6-bit representation scheme which can represent non-numeric characters. This allows 64 characters to be represented. For letter A can be represented as 110001_2 using standard Binary Coded Decimal

Extended Binary Coded Decimal Interchange code (EBCDIC)

- Extended Binary Coded Decimal Interchange code (EBCDIC) is an 8-bit character-coding scheme used primarily on IBM computers. A total of 256 (2^8) characters can be coded using this scheme. For example, the symbolic representation of letter A using Extended Binary Coded Decimal Interchange code is 11000001_2.

American standard code for information interchange (ASCII)

- American standard code for information interchange (ASCII) is a 7-bit code, which means that only 128 characters i.e. 2^7 can be represented. However, manufactures have added an eight bit to this coding scheme, which can now provide for 256 characters.
- This 8-bit coding scheme is referred to as an 8-bit American standard code for information interchange. The symbolic representation of letter A using this scheme is 1000001_2.

1.1.2 Complements

Q4. What is the complement number system? Explain the complement systems with examples.

Ans :

Ones Compliment

- The term compliment refers to *a part* which together with another makes up a *whole*. For example in geometry two complimentary angle (90^0).
- The idea of compliment is used to address the problem of signed numbers i.e. positive and negative.
- In decimal numbers (0 to 9), we talk of nine's compliment. For example the nines compliment
- Of 9 is 0, that of 5 is 4 while 3 is 6.
- However, in binary numbers, the ones compliment is the bitwise NOT applied to the number. Bitwise NOT is a unary operator (operation on only one operand) that performs logical negation on each bit. For example the bitwise NOT of 1100_2 is 0011_2e.
- 0s are negated to 1s while 1s are negated to 0s.

Twos compliment

- Twos compliment, equivalent to tens compliment in decimal numbers, is the most popular way of representing negative numbers in computer systems. The advantages of using this method are:

(i) There are no two ways of representing a zero as in the case with other two methods.

(ii) Effective addition and subtraction can be done even with numbers that are represented with a sign bit without a need for circuitries to examine the sign of an operand.

- The twos compliment of a number is obtained by getting the ones compliment then adding a 1. For example, to get the twos compliment of a decimal number $45_{10,}$
- First convert it to its binary equivalent then find its ones compliment. Add a 1 to ones compliment i.e

$45_{10} = 00101101_2$

Bitwise NOT (00101101) = 11010010

Two's compliment = $11010010_2 + 1_2$

$= 11010011_2$

Binary Addition

The five possible additions in binary are

1. $0 + 0 = 0$
2. $0 + 1_2 = 1_2$
3. $1_2 + 0 = 1_2$
4. $1_2 + 1_2 = 10_2$ (read as 0, carry 1)
5. $1_2 + 1_2 + 1_2 = 11_2$ (read as 1, carry 1)

Example 1

Find the sum of $111_2 + 011_2$

Sol :

Arrange the bits vertically. 111

Working from the right to the left, we proceed as follows: + 011

Step 1: $1_2 + 1_2 = 10_2$, (write down 0 and carry 1) 1010_2

Step 2: $1_2 + 1_2 + 1_2 = 11_2$, (add and carry over digit to 1 + 1 in order to get 1 + 1 +1. From the sum, write down digit one the carry Forward)

Step 3: $1_2 + 1_2 + 0_2 = 10_2$, (add the carry over digit to 1 + 0 in order to get 1 + 1 + 0. since this is the last step, write down 10)

Therefore $111_2 + 011_2 = 1010_2$

This can be summarized in the table

1st number	1	1	1
2nd number	0	1	1
Carry digit	–	1	1
Sum	10	1	0

Example 2

Add the following binary number

10110_2

1011_2

$+ 111_2$

Sol :

Add the first two numbers and then add the sum to the third number a follows:

Step 1	**Step 2**
10110_2	100001_2
$+ 1011_2$	$+ 111_2$
100001_2	101000_2

Binary Subtraction

Direct subtraction

The four possible subtractions in binary are:

1. $0 - 0 = 0$
2. $1_2 - 0 = 1_2$
3. $1_2 - 1_2 = 0$
4. $10_2 - 1_2 = 1_2$ (borrow 1 from the next most significant digit to make 0 become 10_2, hence $10_2 - 1_2 = 1_2$)

Subtraction using ones compliment

The main purpose of using ones compliment in computers is to perform binary subtraction. For example to get the difference in 5 – 3, using the ones compliment, we proceed as follows:

1. Rewrite the problem as 5 + (-3) to show that he computer binary subtraction by adding the binary equivalent of 5 to ones compliment of 3.
2. Convert the absolute value of 3 into 8-bits equivalent i.e. 00000011_2.
3. Take the ones compliment of 00000011_2 e. 11111100_2 which is the binary representation of -3_{10}.
4. Add the binary equivalent of 5 to ones compliment of 3 i.e.

   00000101

   + 11111000

   (1)00000001

Subtraction using twos compliments

Like in ones compliment, the twos compliment of a number is obtained by negating a positive number to is negative counterpart. For example to get the difference in 5-3, using twos compliment, we proceed as follow:

1. rewrite the problem as 5 + (-3)
2. Convert the absolute value of 3 into 8-bit binary equivalent i.e. 00000011.
3. Take the ones compliment of 00000011 i.e. 11111100.
4. add a 1 to the ones compliment i.e. 11111100 to get 11111101
5. add he binary equivalent of 5 to the twos compliment of 3 i.e.

   00000101

   + 11111001

   (1) 00000010 Ignoring the overflow bit, the resulting number is 00000010, which is directly read as a binary equivalent of +2.

Example

Using twos compliment

$31_{10} - 17_{10}$ in binary form.

Sol :

17_{10} in binary 00010001

1's compliment 11101110

2's compliment 11101111

$31_{10} = 00011111_2$

$00011111 + 11101111 = (1)00001110_2$

1.1.3 Fixed and Floating Point Representations

Q5. Explain about fixed and floating point representation of numbers.

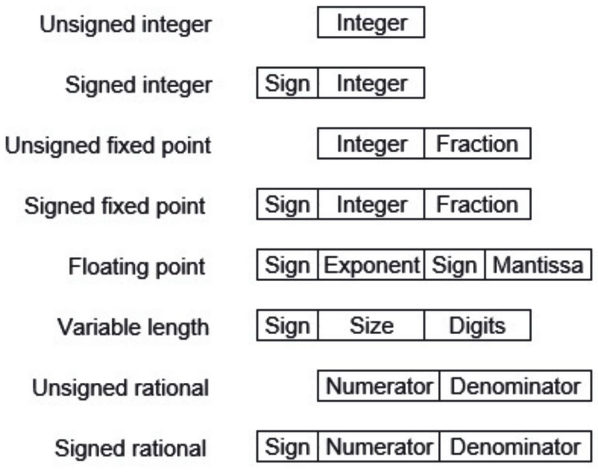
Ans : **(Imp.)**

- Digital Computers use Binary number system to represent all types of information inside the computers.
- Alphanumeric characters are represented using binary bits (i.e., 0 and 1).
- Digital representations are easier to design, storage is easy, accuracy and precision are greater.
- There are various types of number representation techniques for digital number representation,

For example: Binary number system, octal number system, decimal number system, and hexadecimal number system etc. But Binary number system is most relevant and popular for representing numbers in digital computer system.

Storing Real Number:

These are structures as following below:



There are two major approaches to store real numbers (i.e., numbers with fractional component) in modern computing. These are (i) Fixed Point Notation and (ii) Floating Point Notation. In fixed point notation, there are a fixed number of digits after the decimal point, whereas floating point number allows for a varying number of digits after the decimal point.

A) Fixed-Point Representation:

This representation has fixed number of bits for integer part and for fractional part. For example, if given fixed-point representation is IIII.FFFF, then you can store minimum value is 0000.0001 and maximum value is 9999.9999. There are three parts of a fixed-point number representation: the sign field, integer field, and fractional field.

We can represent these numbers using:

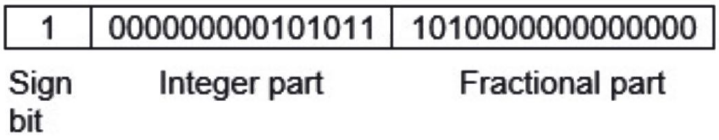
- Signed representation: range from $-(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 1's complement representation: range from $-(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 2's complementation representation: range from $-(2^{(k-1)})$ to $(2^{(k-1)}-1)$, for k bits.

2's complementation representation is preferred in computer system because of unambiguous property and easier for arithmetic operations.

Example:

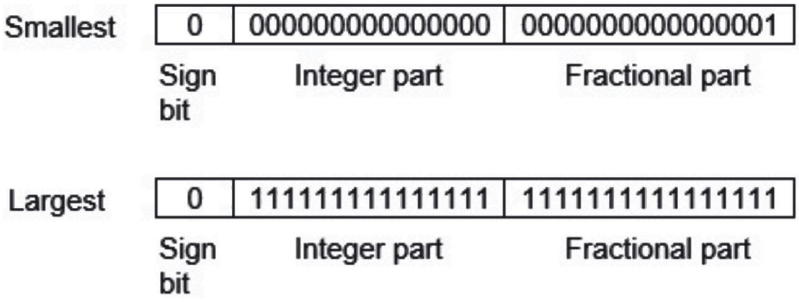
Assume number is using 32-bit format which reserve 1 bit for the sign, 15 bits for the integer part and 16 bits for the fractional part.

Then, -43.625 is represented as following:



Where, 0 is used to represent + and 1 is used to represent. 000000000101011 is 15 bit binary value for decimal 43 and 1010000000000000 is 16 bit binary value for fractional 0.625.

The advantage of using a fixed-point representation is performance and disadvantage is relatively limited range of values that they can represent. So, it is usually inadequate for numerical analysis as it does not allow enough numbers and accuracy. A number whose representation exceeds 32 bits would have to be stored inexactly.



These are above smallest positive number and largest positive number which can be store in 32-bit representation as given above format. Therefore, the smallest positive number is 2^{-16} H" 0.000015 approximate and the largest positive number is $(2^{15}-1)+(1-2^{-16})=2^{15}(1-2^{-16})$ =32768, and gap between these numbers is 2^{-16}.

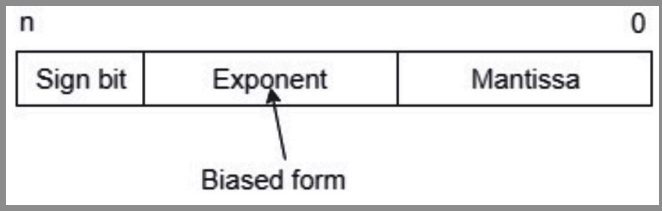
We can move the radix point either left or right with the help of only integer field is 1.

B) Floating-Point Representation:

This representation does not reserve a specific number of bits for the integer part or the fractional part. Instead it reserves a certain number of bits for the number (called the mantissa or significand) and a certain number of bits to say where within that number the decimal place sits (called the exponent).

The floating number representation of a number has two part: the first part represents a signed fixed point number called mantissa. The second part of designates the position of the decimal (or binary) point and is called the exponent. The fixed point mantissa may be fraction or an integer. Floating -point is always interpreted to represent a number in the following form: Mxr^e.

Only the mantissa m and the exponent e are physically represented in the register (including their sign). A floating-point binary number is represented in a similar manner except that is uses base 2 for the exponent. A floating-point number is said to be normalized if the most significant digit of the mantissa is 1.



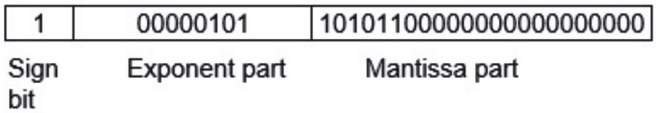
So, actual number is $(-1)^s (1+m) \times 2^{(e-Bias)}$, where s is the sign bit, m is the mantissa, e is the exponent value, and Bias is the bias number.

Note that signed integers and exponent are represented by either sign representation, or one's complement representation, or two's complement representation.

The floating point representation is more flexible. Any non-zero number can be represented in the normalized form of $\pm (1.b_1b_2b_3 \ldots)_2 \times 2^n$ This is normalized form of a number x.

Example: Suppose number is using 32-bit format: the 1 bit sign bit, 8 bits for signed exponent, and 23 bits for the fractional part. The leading bit 1 is not stored (as it is always 1 for a normalized number) and is referred to as a "hidden bit".

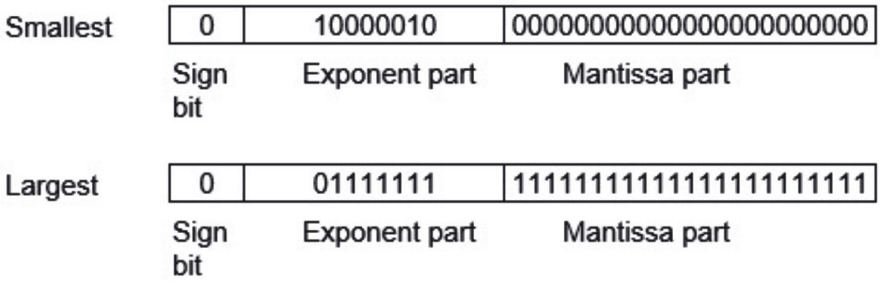
Then –53.5 is normalized as $-53.5=(-110101.1)_2=(-1.101011) \times 2^5$, which is represented as following below,



Where 00000101 is the 8-bit binary value of exponent value +5.

Note that 8-bit exponent ûeld is used to store integer exponents -126 d" n d" 127.

The smallest normalized positive number that ûts into 32 bits is $(1.00000000000000000000000)_2 \times 2^{-126}=2^{-126}$H"$1.18\times10^{-38}$, and largest normalized positive number that ûts into 32 bits is $(1.11111111111111111111111)_2 \times 2^{127} = (2^{24}-1) \times 2^{104} \sim 3.40 \times 10^{38}$. These numbers are represented as following below,



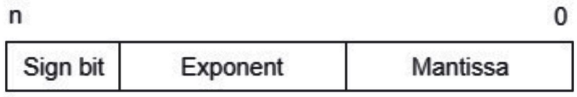
The precision of a floating-point format is the number of positions reserved for binary digits plus one (for the hidden bit). In the examples considered here the precision is 23+1 = 24.

The gap between 1 and the next normalized floating-point number is known as machine epsilon. the gap is $(1+2^{-23})-1 = 2^{-23}$ for above example, but this is same as the smallest positive floating-point number because of non-uniform spacing unlike in the fixed-point scenario.

Note that non-terminating binary numbers can be represented in floating point representation, e.g., 1/3 = $(0.010101 ...)_2$ cannot be a ûoating-point number as its binary representation is non-terminating.

IEEE Floating point Number Representation:

IEEE (Institute of Electrical and Electronics Engineers) has standardized Floating-Point Representation as following diagram.



So, actual number is $(-1)^s(1+m)x2^{(e-Bias)}$, where s is the sign bit, m is the mantissa, e is the exponent value, and Bias is the bias number. The sign bit is 0 for positive number and 1 for negative number. Exponents are represented by or two's complement representation.

According to IEEE 754 standard, the floating-point number is represented in following ways:

- Half Precision (16 bit): 1 sign bit, 5 bit exponent, and 10 bit mantissa
- Single Precision (32 bit): 1 sign bit, 8 bit exponent, and 23 bit mantissa
- Double Precision (64 bit): 1 sign bit, 11 bit exponent, and 52 bit mantissa
- Quadruple Precision (128 bit): 1 sign bit, 15 bit exponent, and 112 bit mantissa

1.1.4 Binary Codes

Q6. Write about various types of binary codes.

Ans :

In the coding, when numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as a code. The digital data is represented, stored and transmitted as group of binary bits. This group is also called as binary code. The binary code is represented by the number as well as alphanumeric letter.

Advantages of Binary Code

Following is the list of advantages that binary code offers.

- Binary codes are suitable for the computer applications.
- Binary codes are suitable for the digital communications.
- Binary codes make the analysis and designing of digital circuits if we use the binary codes.
- Since only 0 & 1 are being used, implementation becomes easy.

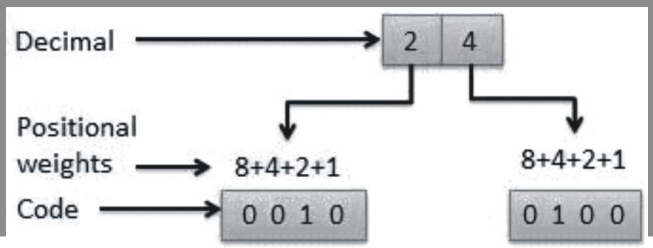
Classification of Binary Codes

The codes are broadly categorized into following four categories.

1. Weighted Codes
2. Non-Weighted Codes
3. Gray Codes
4. Binary Coded Decimal Code
5. Alphanumeric Codes
6. Error Codes

1) Weighted Codes

Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits

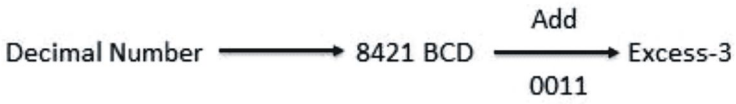


2) Non-Weighted Codes

In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are Excess-3 code and Gray code.

Excess-3 code

The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding (0011)2 or (3)10 to each code word in 8421. The excess-3 codes are obtained as follows –



Example

Decimal	BCD 8 4 2 1	Excess-3 BCD + 0011
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

3) Gray Code

It is the non-weighted code and it is not arithmetic codes. That means there are no specific weights assigned to the bit position. It has a very special feature that, only one bit will change each time the decimal number is incremented as shown in fig. As only one bit changes at a time, the gray code is called as a unit distance code. The gray code is a cyclic code. Gray code cannot be used for arithmetic operation.

Decimal	BCD	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1

Application of Gray code

- Gray code is popularly used in the shaft position encoders.
- A shaft position encoder produces a code word which represents the angular position of the shaft.

4) Binary Coded Decimal (BCD) code

In this code each decimal digit is represented by a 4-bit binary number. BCD is a way to express each of the decimal digits with a binary code. In the BCD, with four bits we can represent sixteen numbers (0000 to 1111). But in BCD code only first ten of these are used (0000 to 1001). The remaining six code combinations i.e. 1010 to 1111 are invalid in BCD.

Decimal	0	1	2	3	4	5	6	7	8	9
BCD	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Advantages of BCD Codes

- It is very similar to decimal system.
- We need to remember binary equivalent of decimal numbers 0 to 9 only.

Disadvantages of BCD Codes

- The addition and subtraction of BCD have different rules.
- The BCD arithmetic is little more complicated.
- BCD needs more number of bits than binary to represent the decimal number. So BCD is less efficient than binary.

5) Alphanumeric Codes

A binary digit or bit can represent only two symbols as it has only two states '0' or '1'. But this is not enough for communication between two computers because there we need many more symbols for communication. These symbols are required to represent 26 alphabets with capital and small letters, numbers from 0 to 9, punctuation marks and other symbols.

The alphanumeric codes are the codes that represent numbers and alphabetic characters. Mostly such codes also represent other characters such as symbol and various instructions necessary for conveying information. An alphanumeric code should at least represent 10 digits and 26 letters of alphabet i.e. total 36 items. The following three alphanumeric codes are very commonly used for the data representation.

- American Standard Code for Information Interchange (ASCII).
- Extended Binary Coded Decimal Interchange Code (EBCDIC).
- Five bit Baudot Code.

ASCII code is a 7-bit code whereas EBCDIC is an 8-bit code. ASCII code is more commonly used worldwide while EBCDIC is used primarily in large IBM computers.

6) Error Codes

There are binary code techniques available to detect and correct data during data transmission.

Error Code	Description
Error Detection and Correction	Error detection and correction code techniques

1.2 Overview of Computer Function and Interconnections

1.2.1 Computer Components

Q7. What is digital computer? Explain the block diagram of it.

(OR)

Explain the basic components of a computer.

Ans : **(Imp.)**

It is an electronic computer in which the input is discrete rather than continuous, consisting of combinations of numbers, letters, and other characters written in an appropriate programming language and represented internally in binary notation, i.e., using only the two digits 0 and 1. By counting, comparing, and manipulating these digits or their combinations according to a set of instructions held in its memory, a digital computer can perform such tasks as to control industrial processes and regulate the operations of machines; analyze and organize vast amounts of business data; and simulate the behaviour of dynamic systems.

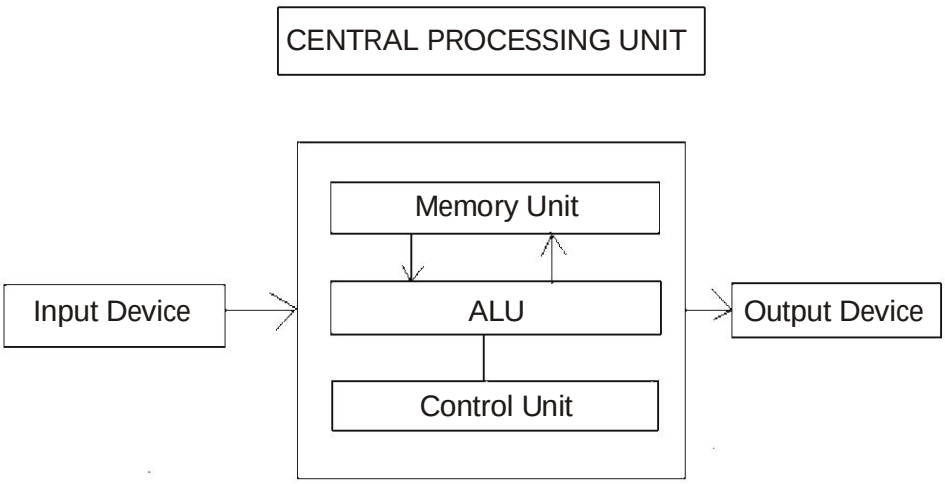
Block Diagram of a Computer

All these major Operations of the computer are performed by using the five basic components of the computer, which are interconnected each other, known as block diagram of the computer.

The following are the five major components of the block diagram of a computer.

- Input Unit
- Output Unit
- Arithmetic and Logic Unit
- Control Unit
- Memory Unit

The following figure shows the block diagram of the computer with five major components.



(a) Input Unit

The Input Unit accepts the instructions and data from outside world through input devices like keyboard, mouse etc. Then it converts these instructions and data from user understandable form to electronic signals, which are understood by the computer. Then it supplies these signals to the Central Processing Unit for further processing.

(b) Central Processing unit

Once the data is accepted from the input unit, it fed to the CPU for further processing before the output is generated. This is known as an electronic brain.

CPU carries out instructions and tells the rest of the computer system what to do. This is done by the Control Unit of the CPU which sends command signals to the other components of the computer

It performs Arithmetical and Logical Calculations with the help of ALU, which is known as Computer calculator. Holds data and instructions which are in current use. These are kept in the Main Memory.

All these operations are done by the following three major components of the CPU.

i) Memory Unit

ii) Control Unit

iii) Arithmetic and Logic Unit

(i) Memory Unit

It is also known as Primary Memory. It can store the data in terms of programs and files. The data stored can be accessed and processed whenever is needed by the CPU is known as Primary memory. The memory is measured as BYTES, KB, MB , GB etc.

The Primary memory is sub divided into,

- **RAM** : This is the main store and is the place where the programs and software we load gets stored.When the Central Processing Unit runs a program, it fetches the program instructions from RAM
- **ROM** : The CPU can only fetch or read instructions from ROM. ROM comes with instructions permanently stored in it.

(ii) Arithmetic and Logic Unit (ALU)

Whenever calculations are required, the control unit transfers the data from storage unit to ALU . Once the computations are done, the results are transferred to the storage unit by the control unit and then it is send to the output unit for displaying results.

The ALU is again sub divided into two functional units

- **Arithmetic Unit** : The arithmetic unit executes arithmetic operations. Which includes addition, subtraction, multiplication, division
- **Logical Unit** : The logic unit executes logical operations. Which includes comparisons like greater than, less than, equals to etc. It also performs logical operations like AND, OR and NOT.

(iii) Control Unit

- It controls all other units in the computer. The control unit must communicate with both the arithmetic logic unit and main memory. The control unit instructs the arithmetic logic unit which arithmetic operations or logical is to be performed.

- The control unit instructs the input unit, where to store the data after receiving it from the user.It controls the flow of data and instructions from the storage unit to ALU.
- It also controls the flow of results from the ALU to the storage unit.

c) Out Put Unit

- It produces the results or the information after the computation to the outside world through the output devices like monitor, printer etc.

It can also store the output to the secondary storage devices like floppy disks, hard disks, CD etc. The control Unit instructs the output unit to produce the output when ever it is necessary to the user.

1.2.2 Interconnection Structures

Q8. Explain about the interconnection structure of the computer.

Ans :

- A computer consists of a set of components or modules of three basic types (processor, memory, I/O) that communicate with each other.
- In effect, a computer is a network of basic modules. Thus, there must be paths for connecting the modules.
- The collection of paths connecting the various modules is called the interconnection structure. The design of this structure will depend on the exchanges that must be made among modules.
- Figure below suggests the types of exchanges that are needed by indicating the major forms of input and output for each module type.

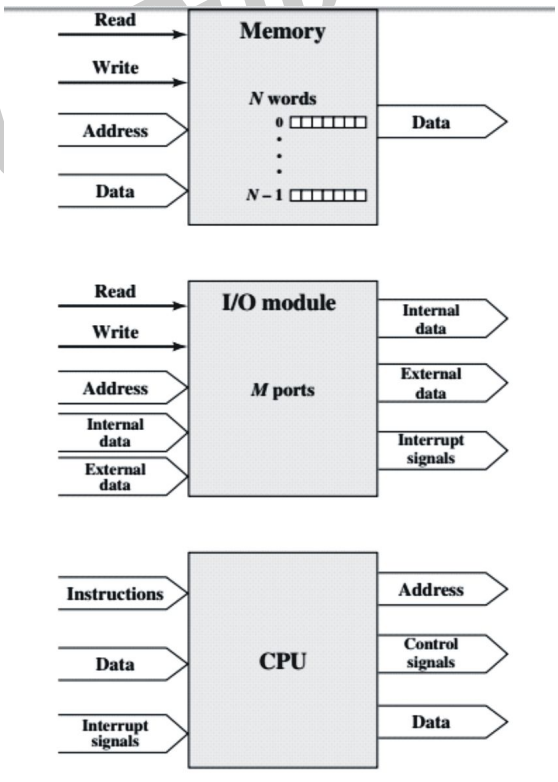


Figure Computer Modules

- **Memory:** Typically, a memory module will consist of N words of equal length. Each word is assigned a unique numerical address (0, 1, . . . , N – 1). A word of data can be read from or written into the memory.The nature of the operation is indicated by read and write control signals. The location for the operation is specified by an address.
- **I/O module:** From an internal (to the computer system) point of view, I/O is functionally similar to memory. There are two operations, read and write. Further, an I/O module may control more than one external device. We can refer to each of the interfaces to an external device as a port and give each a unique address (e.g., 0, 1,…, M – 1). In addition, there are external data paths for the input and output of data with an external device. Finally, an I/O module may be able to send interrupt signals to the processor.
- **Processor:** The processor reads in instructions and data, writes out data after processing, and uses control signals to control the overall operation of the system. It also receives interrupt signals. The preceding list defines the data to be exchanged. The interconnection structure must support the following types of transfers:
- **Memory to processor:** The processor reads an instruction or a unit of data from memory.
- **Processor to memory:** The processor writes a unit of data to memory.
- **I/O to processor:** The processor reads data from an I/O device via an I/O module.
- **Processor to I/O:** The processor sends data to the I/O device.
- **I/O to or from memory:** For these two cases, an I/O module is allowed to exchange data directly with memory, without going through the processor, using direct memory access (DMA).

1.2.3 Bus Interconnection

Q9. What is the use of bus interconnection? Explain the types of bus organization.

Ans :

- A bus is a communication pathway connecting two or more devices.
- A key characteristic of a bus is that it is a shared transmission medium. Multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus.
- If two devices transmit during the same time period, their signals will overlap and become garbled. Thus, only one device at a time can successfully transmit.
- Typically, a bus consists of multiple communication pathways, or lines. Each line is capable of transmitting signals representing binary 1 and binary 0. Over time, a sequence of binary digits can be transmitted across a single line.
- Taken together, several lines of a bus can be used to transmit binary digits simultaneously (in parallel).
- For example, an 8-bit unit of data can be transmitted over eight bus lines. Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy.
- A bus that connects major computer components (processor, memory, I/O) is called a **system bus**. The most common computer interconnection structures are based on the use of one or more system buses.

System bus - This consists of data bus, address bus and control bus

Data bus - A bus which carries data to and from memory/IO is called as data bus

Address bus - This is used to carry the address of data in the memory and its width is equal to the number of bits in the MAR of the memory.

For ex. If comp. memory of 64K has 32 bit words then the computer will have a data bus of 32 bits wide and the address bus of 16 bits wide

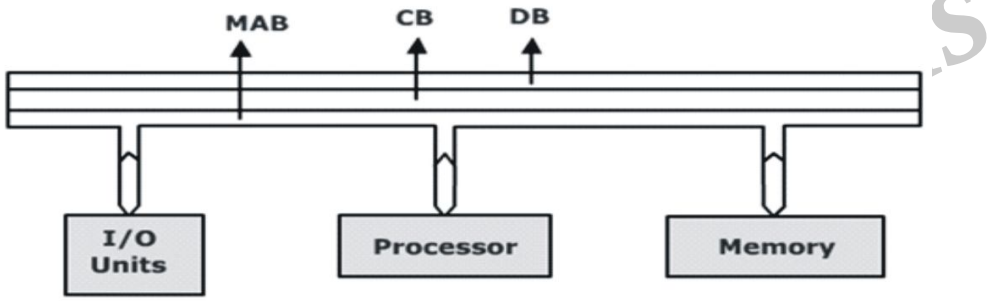
Control Bus - Carries the control signals between the various units of the computer. **Ex: Memory Read/write, I/O Read/write**

Two types of Bus organizations:

(i) Single Bus organization

(ii) Two bus Organization

(i) Single Bus Architecture



Single - Bus Organization

Three units share the single bus. At any given point of time, information can be transferred between any two units

- Here I/O units use the same memory address space (Memory mapped I/O)
- So no special instructions are required to address the I/O, it can be accessed like a memory location
- Since all the devices do not operate at the same speed, it is necessary to smooth out the differences in timings among all the devices A common approach used is to include buffer registers with the devices to hold the information during transfers

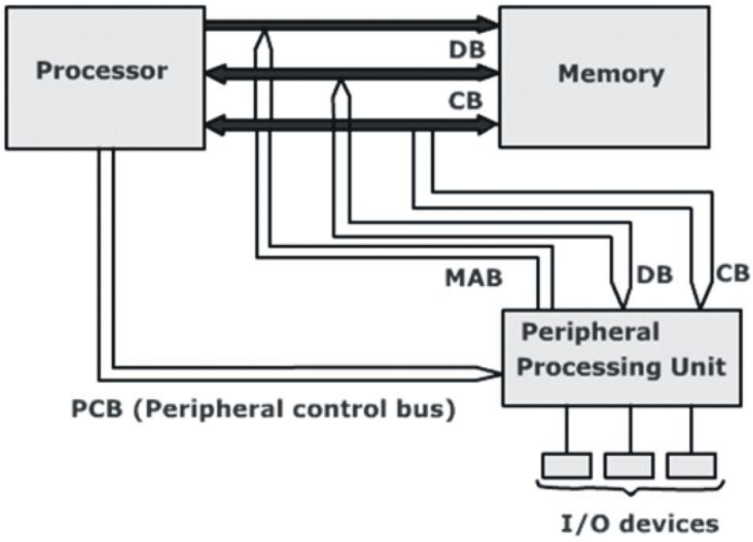
Ex: Communication between the processor and printer

(ii) Two Bus Architecture

Two Bus Organization

- Various units are connected through two independent buses.
- I/O units are connected to the processor though an I/O bus and Memory is connected to the processor through the memory bus.
- I/O bus consists of address, data and control bus Memory bus also consists of address, data and control bus In this type of arrangements processor completely supervises the transfer of information to and from I/O units. All the information is first taken to processor and from there to the memory. Such kind of transfers are called as program controlled transfer.

Alternative Two Bus Architecture



Alternative Two-Bus Organization

- In this I/O units are directly connected to the memory and not to the processor.

  The I/O units are connected to special interface logic known as Direct Memory Access (DMA) or an I/O channel.
- This is also called as Peripheral Processor Unit (PPU)

  In this the data from the I/O device is directly sent to memory bypassing the processor.

1.2.4 Bus structure

Q10. Explain about the structure of typical bus.

Ans :

- A system bus consists, typically, of from about 50 to hundreds of separate lines. Each line is assigned a particular meaning or function.
- Although there are many different bus designs, on any bus the lines can be classified into three functional groups : data, address, and control lines.

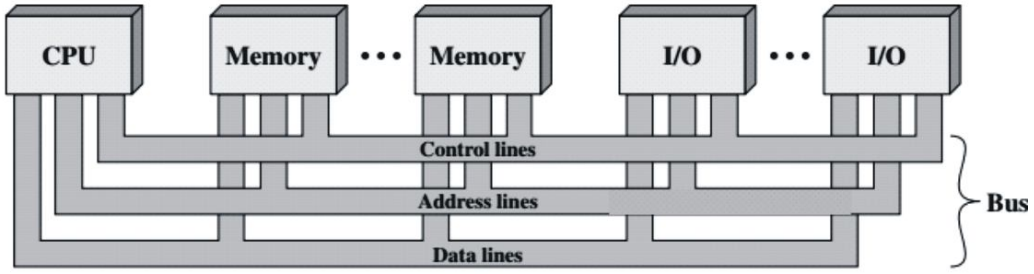


Figure Bus Interconnection Scheme

- In addition, there may be power distribution lines that supply power to the attached modules.
- The data lines provide a path for moving data among system modules. These lines, collectively, are called the **data bus**.
- The data bus may consist of 32, 64, 128, or even more separate lines, the number of lines being referred to as the width of the data bus. Because each line can carry only 1 bit at a time, the number of lines determines how many bits can be transferred at a time.
- The width of the data bus is a key factor in determining overall system performance.
- For example, if the data bus is 32 bits wide and each instruction is 64 bits long, then the processor must access the memory module twice during each instruction cycle.

Data line

- Provide a path for moving data between system modules.
- These lines, collectively, are called the**data bus**
- Data bus may consist of from 32 to 100 separated line
- Each line can carry only one bit at a time
- Number of line in the bus determine the data rate and overall the system performance

Address Line

- The **address lines** are used to designate the source or destination of the data on the data bus.
- For example, if the processor wishes to read a word (8, 16, or 32 bits) of data from memory, it puts the address of the desired word on the address lines. Clearly, the width of the address bus determines the maximum possible memory capacity of the system.
- Furthermore, the address lines are generally also used to address I/O ports.
- Typically, the higher-order bits are used to select a particular module on the bus, and the lower-order bits select a memory location or I/O port within the module.
- For example, on an 8-bit address bus, address 01111111 and below might reference locations in a memory module (module 0) with 128 words of memory, and address 10000000 and above refer to devices attached to an I/O module (module 1).

Control Lines

- The **control lines** are used to control the access to and the use of the data and address lines. Because the data and address lines are shared by all components, there must be a means of controlling their use.
- Control signals transmit both command and timing information among system modules. Timing signals indicate the validity of data and address information. Command signals specify operations to be performed.

Typical control lines include:

- **Memory write:** Causes data on the bus to be written into the addressed location.
  - **Memory read:** Causes data from the addressed location to be placed on the bus.
  - **I/O write:** Causes data on the bus to be output to the addressed I/O port.
  - **I/O read:** Causes data from the addressed I/O port to be placed on the bus.
  - **Transfer ACK:** Indicates that data have been accepted from or placed on the bus.
  - **Bus request:** Indicates that a module needs to gain control of the bus.
  - **Bus grant:** Indicates that a requesting module has been granted control of the bus.
  - **Interrupt request:** Indicates that an interrupt is pending.
  - **Interrupt ACK:** Acknowledges that the pending interrupt has been recognized.
  - **Clock:** Is used to synchronize operations.
  - **Reset:** Initializes all modules

The operation of the bus is as follows. If one module wishes to send data to another, it must do two things:

(1) obtain the use of the bus, and

(2) transfer data via the bus.

If one module wishes to request data from another module, it must

(1) obtain the use of the bus, and

(2) transfer a request to the other module over the appropriate control and address lines. It must then wait for that second module to send the data.

Physically, the system bus is actually a number of parallel electrical conductors. In the classic bus arrangement, these conductors are metal lines etched in a card or board (printed circuit board).

The bus extends across all of the system components, each of which taps into some or all of the bus lines.

1.2.5 Data Transfer

Q11. Explain three modes of data transfer between I/O devices and a computer system.

Ans : **(Imp.)**

The three modes of data transfer or three possible techniques of I/O operations are as follows.

1. Programmed I/O
2. Interrupt Driven I/O
3. Direct Memory Access (DMA)

1. Programmed I/O

It is a technique of organizing an I/O activity such that the process gives full access to a seperate unit called I/O model to take charge of the entire I/O activity, which (I/O model) inturn does not take any responsibility to intimate the processor about completion of it's task.

In case of programmed I/O whenever a processor indulges itself in the program execution on, (say) encountering an I/O activity, it calls the described I/O module and authorize it. The I/O module after taking charge executes the required I/O activity depending on the availability of the processor. After completion of a task, the I/O module sets the contents of I/O status register and switches to other activities. It neither intimates the processor nor interrupts it to provide the status of execution, rather it is the responsibility of the processor to check the status of the I/O module periodically. Also, if there is any requirement of data to be collected from the main memory, the processor collects it and provides it to the given I/O module. Also here the processor performs various other operations involving checking of the I/O, device, Status, transferring data as well as initiating READ/WRITE command etc.

The processor is ineffective since the time of the processor wasted in handling various meagre activities rather than utilizing it in completing the high value task. Hence, the usage of program driven I/O has declined to greater extent.

2. Interrupt driven I/O

This technique is used to overcome the limitations of programmed I/O. In short, in interrupt driven I/O instead of making the processor to check the status of the I/O module, it is the responsibility of the I/O module to intimate the processor by issuing an interrupt signal. The possible cases are as follows.

Case 1:

Here the interrupt driven I/O process is analyzed from the I/O module view i.e the I/O module receives a READ signal from the processor. After receiving the signal, the I/O module gets activated and receives the data from the corresponding I/O unit meanwhile, the processor resumes to some other task. As the I/O module completes its task, it intimates the processor about it's status by sending an interrupt through control signals and wait until it receives the response from the processor. The processor responds back to the I/O module by sending the address of the memory(where the required data has to be stored). The I/O module places it's data to the lines and reverts back to the same address.

Case 2:

Here the interrupt driven I/O is analyzed from the processors view. Whenever the processor indulges itself in executing the given program and encounter an I/O activity it temporarily sustain it's activity (by placing it's status in the PC and stack pointer), transfer a read command to the I/O module and resumes back to its execution by popping the data from PC and SP registers. The I/O module on receiving command collects the required data from the I/O devices, interrupts the processor, and the entire activity goes on as it was observed in case 1.

3. Direct memory access (DMA)

This technique is more efficient than the programmed I/O and interrupt driven techniques. DMA or Direct Memory Access is granted to an independent block which resides either on the system bus or near the I/O module. Whenever the processor analyzes the requirement of performing an I/O activity, it activates the given DMA module by transmitting special information consisting of data such as, the address of the given I/O devices, status of READ/WRITE operations, address of the memory location, amount of data to be READ/WRITE etc. and the processor resume back to its execution. The DMA module based on this information performs the given task and interrupts the processor about it's status. Hence, in this process, the processor is involved only at the beginning and at the end. As the I/O devices can now transmit their data or interact with the memory directly, the process is called DMA.

UNIT II

Register Transfer Micro operations

Register Transfer Language, Register Transfer, Bus and Memory Transfers, Arithmetic, Logic and Shift micro operations, Arithmetic Logic Shift Unit.

Basic Computer Organization and Design

Instruction Codes, Computer Registers, Computer Instructions, Timing and Control, Instruction Cycle, Memory reference instruction, Input-Output and Interrupt.

2.1 Register Transfer Micro Operations

2.1.1 Register Transfer Language

Q1. Explain the Register Transfer Language.

Ans : **(Imp.)**

Definition: The symbolic notation used to describe the microoperation transfers among registers is called a register transfer language.

- The term "register transfer" implies the availability of hardware logic circuits that can perform a stated microoperation and transfer the result of the operation to the same or another register.
- The word "language" is borrowed from programmers, who apply this term to programming languages.
- A register transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.
- It is a convenient tool for describing the internal organization of digital computers in concise and precise manner.
- It can also be used to facilitate the design process of digital systems.
- Information transfer from one register to another is designated in symbolic form by means of a replacement operator.
- The statement below denotes a transfer of the content of register R1 into register R2. R2 – R1
- A statement that specifies a register transfer implies that circuits are available from the outputs of the destination register has a parallel load capability.
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer.

2.1.2 Register Transfer

Q2. Explain the Register Transfer in detail with block diagram and timing diagram.

Ans :

Definition: Information transfer from one register to another is designated in symbolic form by means of a replacement operator is known as Register Transfer.

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

Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.

For example:

MAR	Holds address of memory unit
PC	Program Counter
IR	Instruction Register
R_1	Processor Register

Below figure (a) shows the representation of registers in block diagram form.

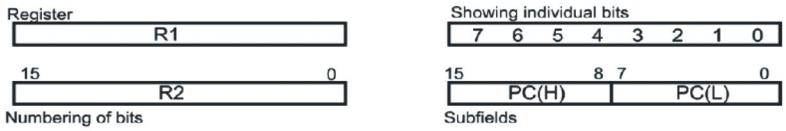


Figure (a) : Block diagram of register

The most common way to represent a register is by a rectangular box with the name of the register inside, as shown in figure.

- Bits 0 through 7 are assigned the symbol L (for low byte) and bits 8 through 15 are assigned the symbol H (for high byte). The name of the 16-bit register is PC. The symbol PC(0-7) or PC(L) refers to the low-order byte and PC(8-15) or PC(H) to the high-order byte.
- The statement that specifies a register transfer implies that circuits are available from the outputs of the source register to the inputs of the destination register and that the destination register has a parallel load capability.

Register Transfer with Control Function

- If we want the transfer to occur only under a predetermined control condition. This can be shown by means of an if-then statement.
- If (P = 1) then (R2 $\rightarrow$ R1) where P is a control signal.
- It is sometimes convenient to separate the control variables from the register transfer operation control function by specifying a control function.
- A **control function** is a Boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

  P: R2 – R1

- The control condition is terminated with a colon. It symbolizes the requirement that the transfer operation be executed by the hardware only if P = 1.
- Every statement written in a register transfer notation implies a hardware construction for implementing the transfer. Below figure shows the block diagram that depicts the transfer from R1 to R2.

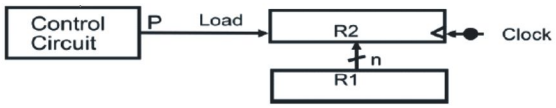


Figure (b) Transfer from R1 to R2 when P = 1

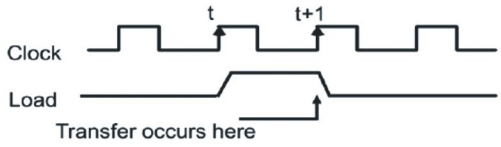


Figure (c) : Timing diagram

- The n outputs of register R1 are connected to the n inputs of register R2. The letter n will be used to indicate any number of bits for the register.
- In the timing diagram, P is activated in the control section by the rising edge of a clock pulse at time t.
- The next positive transition of the clock at time t + 1 finds the load input active and the data inputs of R2 are then loaded into the register in parallel.
- P may go back to 0 at time t + 1; otherwise, the transfer will occur with every clock pulse transition while P remains active.

The basic symbols of the register transfer notation are listed in Table below:

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	MAR, R2
Parentheses ( )	Denotes a part of a register	R 2(0-7), R2(L)
Arrow ←	Denotes transfer of information	R2←R1
Comma ,	Separates two micro operations	R2←R1, R1←R2

Table : Basic Symbols for Register Transfers

Registers are denoted by capital letters, and numerals may follow the letters.

- Parentheses are used to denote a part of a register by specifying the range of bits or by giving a symbol name to a portion of a register.
- The arrow denotes a transfer of information and the direction of transfer.
- A comma is used to separate two or more operations that are executed at the same time.
- The statement below, denotes an operation that exchanges the contents of two registers during one common clock pulse provided that T = 1.

  T: R2 – R1, R1 – R2

- This simultaneous operation is possible with registers that have edge-triggered flipflops.

2.1.3 Bus and Memory Transfers

Q3. Design and explain a common bus system for four register.

Ans :

- A typical digital computer has many registers, and paths must be provided to transfer information from one register to another.

- The number of wires will be excessive if separate lines are used between each register and all other registers in the system.
- A more efficient scheme for transferring information between registers in a multiple register configuration is a common bus system.
- A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.
- Control signals determine which register is selected by the bus during each particular register transfer.
- One way of constructing a common bus system is with multiplexers.
- The multiplexers select the source register whose binary information is then placed on the bus.
- The construction of a bus system for four registers is shown in figure below.
- Each register has four bits, numbered 0 through 3.
- The bus consists of four 4 × 1 multiplexers each having four data inputs, 0 through 3, and two selection inputs, S1 and S0.
- The diagram shows that the bits in the same significant position in each register are connected to the data inputs of one multiplexer to form one line of the bus

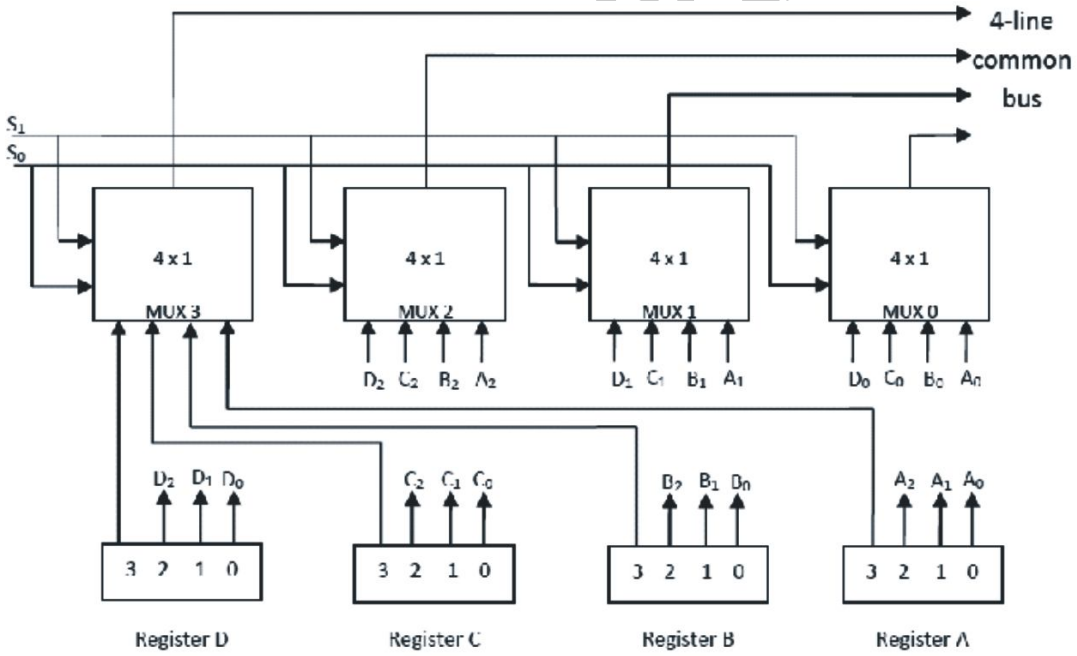


Figure: Bus system for four registers

The two selection lines S1 and S0 are connected to the selection inputs of all four multiplexers.

- The selection lines choose the four bits of one register and transfer them into the fourline common bus.

S_1	S_0	Register selected
0	0	A
0	1	B
1	0	C
1	1	D

Table : Function Table for Bus

- When S1 S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that form the bus.
- This causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.
- Similarly, register B is selected if S1S0 = 01, and so on.
- Table shows the register that is selected by the bus for each of the four possible binary values of the selection lines.
- In general, a bus system will multiplex k registers of n bits each to produce an n-line common bus.
- The number of multiplexers needed to construct the bus is equal to n, the number of bits in each register.
- The size of each multiplexer must be K x 1 since it multiplexes K data lines.

For example, a common bus for eight registers of 16 bits each requires 16 multiplexers, one for each line in the bus. Each multiplexer must have eight data input lines and three selection lines to multiplex one significant bit in the eight registers.

Q4. A digital computer has a common bus system for 16 registers of 32 bits each. (i) How many selection input are there in each multiplexer? (ii) What size of multiplexers is needed? (iii) How many multiplexers are there in a bus?

Ans :

(i) How many selection input are there in each multiplexer?

2n = No. of Registers; n = selection input of multiplexer

2n = 16; here n = 4

Therefore 4 selection input lines should be there in each multiplexer.

(ii) What size of multiplexers is needed?

size of multiplexers = Total number of register X 1

= 16 X 1

Multiplexer of 16 x 1 size is needed to design the above defined common bus.

(iii) How many multiplexers are there in a bus?

No. of multiplexers = bits of register

= 32

32 multiplexers are needed in a bus.

Q5. Explain three-state bus buffer.

OR

Explain the operation of three state bus buffers and show its use in design of common bus.

Ans :

- A bus system can be constructed with three-state gates instead of multiplexers.
- A three-state gate is a digital circuit that exhibits three states.
  - State 1: Signal equivalent to Logic 1
  - State 2: Signal equivalent to Logic 0
  - State 3: High Impedance State (behaves as open circuit)
- The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have logic significance.
- The most commonly used design of a bus system is the buffer gate.
- The graphic symbol of a three-state buffer gate is shown in figure below:

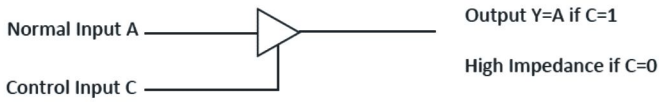
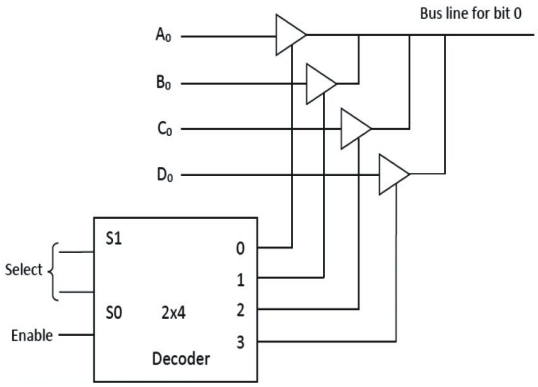


Figure : Graphic symbols for three - state buffer

- It is distinguished from a normal buffer by having both a normal input and a control input.
- The control input determines the output state. When the control input C is equal to 1, the output is enabled and the gate behaves like any conventional buffer, with the output equal to the normal input.
- When the control input C is 0, the output is disabled and the gate goes to a high impedance state, regardless of the value in the normal input.
- The high-impedance state of a three-state gate provides a special feature not available in other gates.
- Because of this feature, a large number of three-state gate outputs can be connected with wires to form a common bus line without endangering loading effects.
- The construction of a bus system with three-state buffers is demonstrated in figure 1.6 below:



- The outputs of four buffers are connected together to form a single bus line.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- No more than one buffer may be in the active state at any given time.
- The connected buffers must be controlled so that only one three-state buffer has access to the bus line while all other buffers are maintained in a high impedance state.
- One way to ensure that no more than one control input is active at any given time is to use a decoder, as shown in the figure: Bus line with three state-buffers.
- When the enable input of the decoder is 0, all of its four outputs are 0, and the bus line is in a high-impedance state because all four buffers are disabled.
- When the enable input is active, one of the three-state buffers will be active, depending on the binary value in the select inputs of the decoder.
- To construct a common bus for four registers of n bits each using three- state buffers, we need n circuits with four buffers in each as shown in figure: Bus line with three state-buffers,
- Each group of four buffers receives one significant bit from the four registers.
- Each common output produces one of the lines for the common bus for a total of n lines.
- Only one decoder is necessary to select between the four registers.

Q6. Explain Memory Transfer.

Ans :

Read Operation: The transfer of information from a memory word to the outside environment is called a read operation.

Write Operation: The transfer of new information to be stored into the memory is called a write operation.

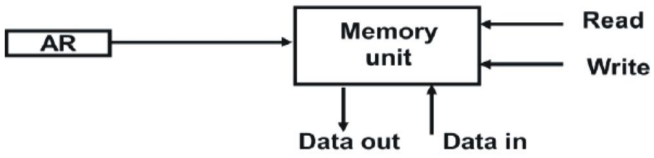
- A memory word will be symbolized by the letter M.
- It is necessary to specify the address of M when writing memory transfer operations.
- This will be done by enclosing the address in square brackets following the letter M.
- Consider a memory unit that receives the address from a register, called the address register, symbolized by AR.
- The data are transferred to another register, called the data register, symbolized by
- DR. The read operation can be stated as follows:

Read: DR → M[AR]

- This causes a transfer of information into DR from the memory word M selected by the address in AR.
- The write operation transfers the content of a data register to a memory word M selected by the address. Assume that the input data are in register R1 and the address is in AR.
- Write operation can be stated symbolically as follows:

Write: M[AR] → R1

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



2.1.4 Arithmetic

Q7. Write various Arithmetic Micro-operation.

Ans :

The basic arithmetic micro-operations are:

1. Addition
2. Subtraction
3. Increment
4. Decrement
5. Shift

The additional arithmetic micro operations are:

1. Add with carry
2. Subtract with borrow
3. Transfer/Load , etc.

Summary of Typical Arithmetic Micro-Operations:

Addition	$R3 \leftarrow R1 + R2$	Contents of R1 plus R2 transferred to R3
Subtraction	$R3 \leftarrow R1 - R2$	Contents of R1 minus R2 transferred to R3
Complement	$R2 \leftarrow R2'$	Complement the contents of R2
Add with carry	$R2 \leftarrow R2' + 1$	2's complement the contents of R2 (negate)
Subtract with born	$R3 \leftarrow R1 + R2' + 1$	Subtraction
Increment	$R1 \leftarrow R1 + 1$	Increment
Decrement	$R1 \leftarrow R1 - 1$	Decrement
Transfer	$F = A$	Transfer A
Shift	$R_1 = ShlRt$	Shift R_1 one bit left

Q8. Explain Binary Adder in detail

Ans :

To implement the add micro operation with hardware, we need :

1. Registers : that hold the data
2. Digital component: that performs the arithmetic addition.

Full-adder

The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a full-adder.

Binary adder

The digital circuit that generates the arithmetic sum of two binary numbers of any lengths is called a binary adder.

The binary adder is constructed with full-adder circuits connected in cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.

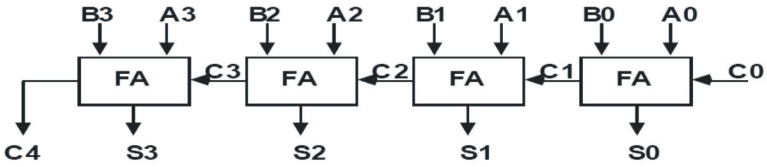


Figure : 4-bit binary adder

- Above figure shows the interconnections of four full-adders (FA) to provide a 4-bit binary adder.
- The augends bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the low-order bit.
- The carries are connected in a chain through the full-adders.
- The input carry to the binary adder is C0 and the output carry is C4.
- The S outputs of the full-adders generate the required sum bits.
- An n-bit binary adder requires n full-adders.
- The output carry from each full-adder is connected to the input carry of the next-highorder full-adder.
- The n data bits for the A inputs come from one register (such as R1), and the n data bits for the B inputs come from another register (such as R2). The sum can be transferred to a third register or to one of the source registers (R1 or R2), replacing its previous content.

Q9. Explain Binary Adder-Subtractor in detail.

Ans :

- The subtraction of binary numbers can be done most conveniently by means of complements.
- Remember that the subtraction A - B can be done by taking the 2's complement of B and adding it to A.
- The 2's complement can be obtained by taking the I's complement and adding one to the least significant pair of bits. The I's complement can be implemented with inverters and a one can be added to the sum through the input carry.
- The addition and subtraction operations can be combined into one common circuit by including an exclusive-OR gate with each full-adder.
- The mode input M controls the operation.
  - When M = 0 the circuit is an Adder
  - When M = 1 the circuit becomes a Subtractor

- Each exclusive-OR gate receives input M and one of the inputs of B.
  - When M = 0,
  - We have C0=0
  - B + 0 = B
- The full-adders receive the value of B, the input carry is 0, and the circuit performs A plus B.
  - When M = 1,
  - We have C0=1
  - B + 1 = B' ; B complement
- The B inputs are all complemented and 1is added through the input carry. The circuit performs the operation A plus the 2's complement of B.
  - A + 2's compliment of B
- A 4-bit adder-subtractor circuit is shown as follows:

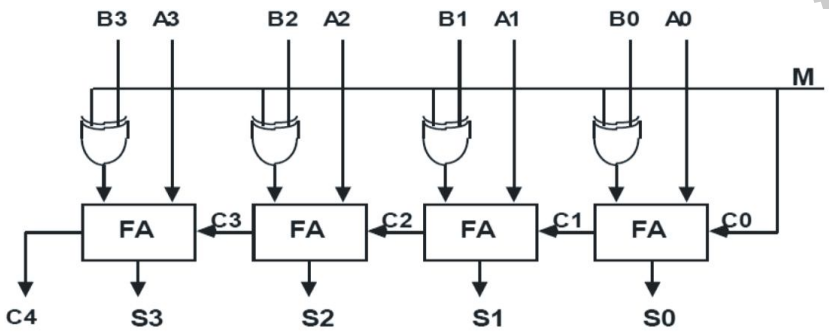


Figure : 4-bit Adder -Subtractor

- For unsigned numbers,
  - If A > = B, then A–B
  - If A < B, then B–A
- For signed numbers,
  - Result is A–B, provided that there is no overflow.

Q10. Explain briefly about Binary Incrementer

Ans :

- The increment micro operation adds one to a number in a register.
- For example, if a 4-bit register has a binary value 0110, it will go to 0111 after it is incremented.

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

- The diagram of a 4-bit combinational circuit incrementer is shown above. One of the inputs to the least significant half-adder (HA) is connected to logic-1 and the other input is connected to the least significant bit of the number to be incremented.
- The output carry from one half-adder is connected to one of the inputs of the next higher- order half-adder.
- The circuit receives the four bits from A0 through A3, adds one to it, and generates the incremented output in S0 through S3.
- The output carry C4 will be 1 only after incrementing binary 1111. This also causes

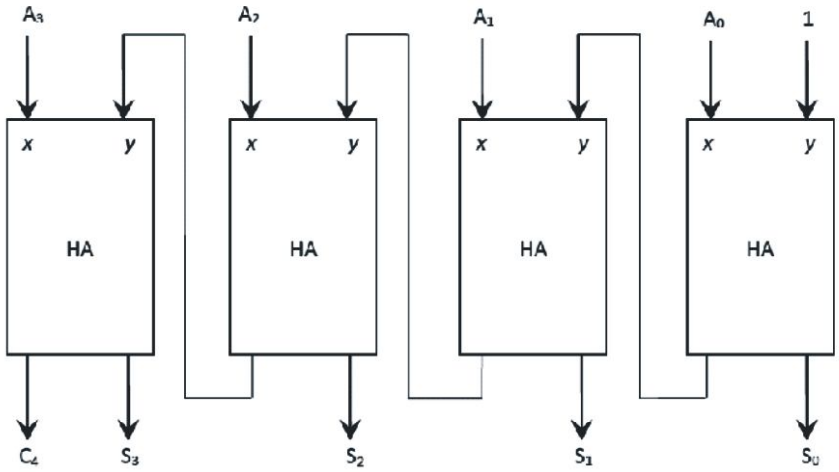


Figure : 4-bit binary incrementer

Q11. Draw block diagram of 4-bit arithmetic circuit and explain it in detail.

Ans :

- The arithmetic micro operations can be implemented in one composite arithmetic circuit.
- The basic component of an arithmetic circuit is the parallel adder.
- By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.
- Hardware implementation consists of:

1. 4 full-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations.
2. There are two 4-bit inputs A and B

   The four inputs from A go directly to the X inputs of the binary adder. Each of the four inputs from B is connected to the data inputs of the multiplexers. The multiplexer's data inputs also receive the complement of B.
3. The other two data inputs are connected to logic-0 and logic-1. Logic-0 is a fixed voltage value (0 volts for TTL integrated circuits) and the logic-1 signal can be generated through an inverter whose input is 0.
4. The four multiplexers are controlled by two selection inputs, S1 and S0.

5. The input carry Cin goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.
6. 4-bit output D0…D3

The diagram of a 4-bit arithmetic circuit is shown below figure 1.10:

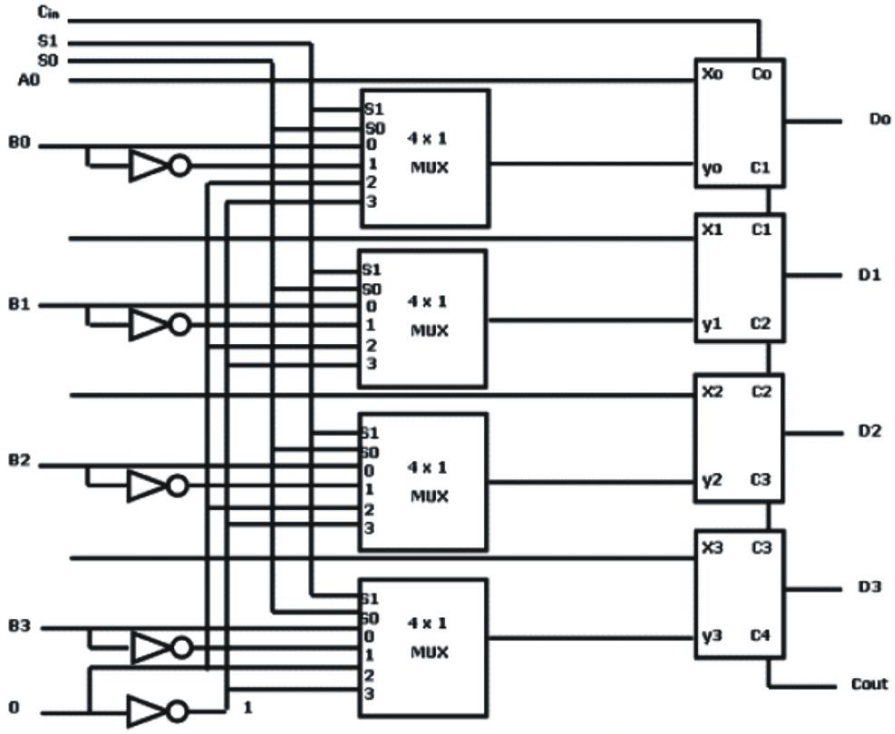


Figure : 4-bit arithmetic circuit

The output of binary adder is calculated from arithmetic sum.

D = A + Y + Cin

Select S_1 S_0 C_{in}			Input Y	Output D = A + Y + Cin	Microoperation
0	0	0	B	D = A + B	Add
0	0	1	B	D = A + B + 1	Add with Carry
0	1	0	B'	D = A + B'	Subtract with Borrow
0	1	1	B'	D = A + B' + 1	Subtract
1	0	0	0	D = A	Transfer A
1	0	1	0	D = A + 1	Increment A
1	1	0	1	D = A – 1	Decrement A
1	1	1	1	D = A	Transfer A

Table : 4 – 4 Arithmetic Circuit Function Table

- When S1 S0 = 0 0
  - If Cin = 0, D = A + B; Add
  - If Cin = 1, D = A + B + 1; Add with carry
- When S1 S0 = 0 1
  - If Cin = 0, D = A + $\underline{B}$; Subtract with borrow
  - If Cin = 1, D = A + $\underline{B}$ + 1; A + 2's compliment of B i.e. A–B
- When S1 S0 = 1 0
  - Input B is neglected and Y = > logic '0'
  - D = A + 0 + Cin
    - If Cin = 0, D = A; Transfer A
    - If Cin = 1, D = A + 1;Increment A
- When S1 S0 = 11
  - Input B is neglected and Y = > logic '1'
  - D = A – 1 + Cin
    - If Cin = 0, D = A–1; 2's compliment
    - If Cin = 1, D = A; Transfer A
- Note that the micro-operation D = A is generated twice, so there are only seven distinct micro-operations in the arithmetic circuit.

2.1.5 Logic and Shift micro operations

Q12. Draw and explain Logic Micro-operations in detail.

Ans : **(Imp.)**

- Logic micro operations specify binary operations for strings of bits stored in registers.
- These operations consider each bit of the register separately and treat them as binary variables. For example, the exclusive-OR micro-operation with the contents of two registers R1 and R2 is symbolized by the statement:

  P: R1 ⟶ R1 + R2

  1 0 1 0 Content of R1

  + 1 1 0 0 Content of R2

  ---

  0 1 1 0 Content of R1 after P = 1

- The logic micro-operations are seldom used in scientific computations, but they are very useful for bit manipulation of binary data and for making logical decisions.

Notation:

- The symbol v will be used to denote an **OR** microoperation and the symbol ^ to denote an **AND** microoperation. The complement microoperation is the same as the 1's complement and uses a bar on top of the symbol that denotes the register name.

- Although the + symbol has two meanings, it will be possible to distinguish between them by noting where the symbol occurs. When the symbol + occurs in a micro operation, it will denote an arithmetic plus. When it occurs in a control (or Boolean) function, it will denote an OR operation.

  P + Q: R1 → R2 + R3, R4 → R5 V R6

- The + between P and Q is an OR operation between two binary variables of a control function. The + between R2 and R3 specifies an add microoperation. The OR micro operation is designated by the symbol V between registers R5and R6.

List of Logic Micro Operations

- There are 16 different logic operations that can be performed with two binary variables.
- They can be determined from all possible truth tables obtained with two binary variables as shown in table below.

x	Y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table : Truth Tables for 16 Functions of Two Variables

Boolean function	Microoperation	Name
$F_0 = 0$	F←0	Clear
$F_1 = xy$	F←A ∧ B	AND
$F_2 = xy'$	F←A ∧ $\overline{B}$	
$F_3 = x$	F←A	Transfer A
$F_4 = x'y$	F←$\overline{A}$ ∧ B	
$F_5 = y$	F←B	Transfer B
$F_6 = x \oplus y$	F←A⊕B	Exclusive-OR
$F_7 = x + y$	F←AvB	OR
$F_8 = (x+ y)'$	F←$\overline{A \vee B}$	NOR
$f_9 = (x \oplus y)'$	F←$\overline{A \oplus B}$	Exclusive-NOR
$F_{10} = y'$	F←$\overline{B}$	Complement B
$F_{11} = x + y'$	F←A V $\overline{B}$	
$F_{12} = x'$	F←$\overline{A}$	Complement A
$F_{13} = x' + y$	F←$\overline{A}$ V B	
$F_{14} = (xy)'$	F←$\overline{A \wedge B}$	NAND
$F_{15} = 1$	F ←all 1's	Set to all I's

Table : Sixteen Logic Microoperation

Hardware Implementation

- The hardware implementation of logic microoperation requires that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic function.
- Although there are 16 logic microoperation, most computers use only four—AND, OR, XOR (exclusive-OR), and complement from which all others can be derived.
- Below figure shows one stage of a circuit that generates the four basic logic micro operations.

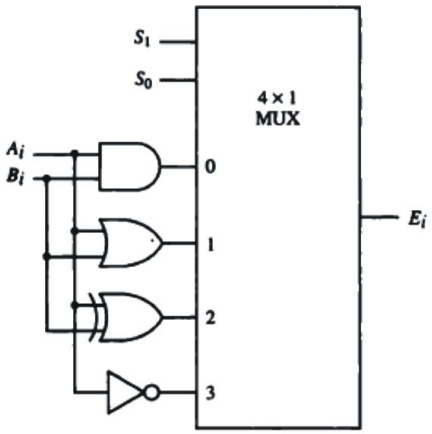


Figure 1.11 : One stage of logic circuit

S_1	S_0	Output	Operation
0	0	E = A ∧ B	AND
0	1	E = A V B	OR
1	0	E = A ⊕ B	XOR
1	1	E = $\bar{A}$	Compliment

Table : Function table

- Hardware implementation consists of four gates and a multiplexer.
- Each of the four logic operations is generated through a gate that performs the required logic.
- The outputs of the gates are applied to the data inputs of the multiplexer.
- The two selection inputs S1 and S0 choose one of the data inputs of the multiplexer and direct its value to the output.
- The diagram shows one typical stage with subscript i. For a logic circuit with n bits, the diagram must be repeated n times for i = 0, 1, 2, . . . n - 1. The selection variables are applied to all stages.

Q13. Explain selective set, selective complement and selective clear.

Ans :

Selective-Set operation:

- The selective-set operation sets to 1 the bits in register A where there are corresponding 1's in register B. It does not affect bit positions that have 0's in B. The following numerical example clarifies this operation:

1010 A before

1100 B (logical operand)

1110 A after

- The two leftmost bits of B are 1's, so the corresponding bits of A are set to 1.
- One of these two bits was already set and the other has been changed from 0 to 1.
- The two bits of A with corresponding 0's in B remain unchanged. The example above serves as a truth table since it has all four possible combinations of two binary variables.
- The OR micro operation can be used to selectively set bits of a register.

Selective-Complement operation:

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

1010 A before

1100 B (logical operand)

0110 A after

- Again the two leftmost bits of B are 1's, so the corresponding bits of A are complemented.
- The exclusive-OR microoperation can be used to selectively complement bits of a register.

Selective-Clear operation:

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

1010 A before

1100 B (logical operand)

0010 A after

- Again the two leftmost bits of B are 1's, so the corresponding bits of A are cleared to 0.
- One can deduce that the Boolean operation performed on the individual bits is AB'.
- The corresponding logic microoperation is A ← A ^ B'.

Q14. Explain shift micro operations and draw 4-bit combinational circuit shifter.

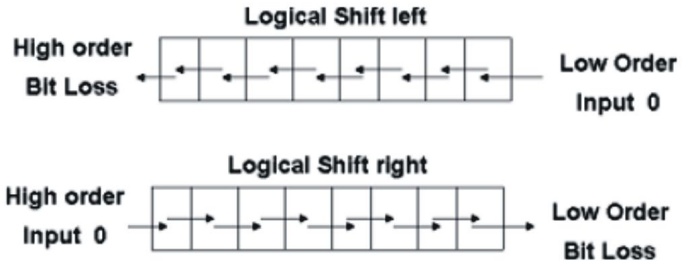
Ans :

There are 3 types of shift micro-operations:

1. Logical Shift:

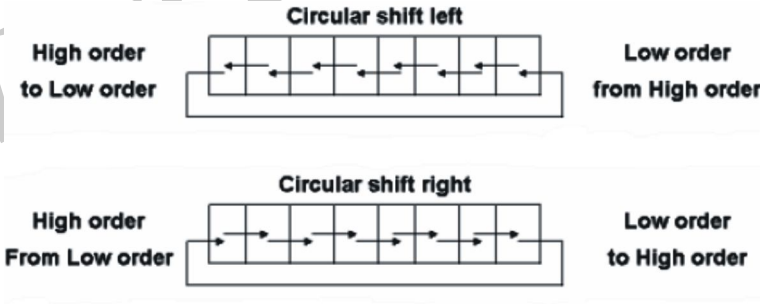
- A logical shift is one that transfers 0 through the serial input. We will adopt the symbols shl and shr for logical shift-left and shift-right micro-operations.
- For example:
  - R1 ← shl R1
  - R2 ← shr R2

- are two micro-operations that specify a 1-bit shift to the left of the content of register R1 and a 1-bit shift to the right of the content of register R2.
- The register symbol must be the same on both sides of the arrow.
- The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.



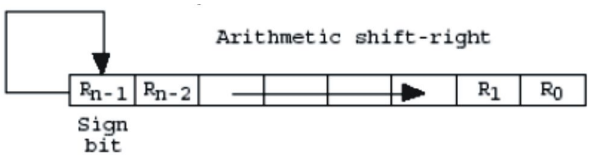
2. Circular Shift:

- The circular shift (also known as a rotate operation) circulates the bits of the register around the two ends without loss of information.
- This is accomplished by connecting the serial output of the shift register to its serial input. We will use the symbols cil and cir for the circular shift left and right, respectively.
  - o R1 ← Cil R1
  - o R2 ← Cir R2

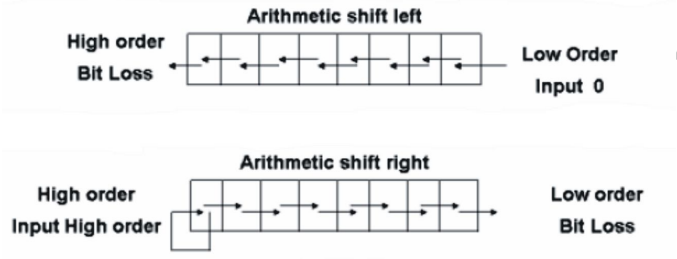


3. Arithmetic Shift:

- An arithmetic shift is a micro-operation that shifts a signed binary number to the left or right.
- An arithmetic shift-left multiplies a signed binary number by 2.
- An arithmetic shift-right divides the number by 2.
- Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2.
- The leftmost bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is 0 for positive and 1 for negative.
- Negative numbers are in 2's complement form.



- Figure shows a typical register of n bits. Bit Rn-1 in the leftmost position holds the sign bit.
- Rn-2 is the most significant bit of the number and R0 is the least significant bit.
- The arithmetic shift-right leaves the sign bit unchanged and shifts the number (including the sign bit) to the right.
- Thus Rn-1 remains the same; Rn-2 receives the bit from Rn-1, and so on for the other bits in the register.
- The bit in R0 is lost.
- The arithmetic shift-left inserts a 0 into R0, and shifts all other bits to the left.
- The initial bit of Rn-1 is lost and replaced by the bit from Rn-2.
- A sign reversal occurs if the bit in Rn-1 changes in value after the shift. This happens if the multiplication by 2 causes an overflow.



4-bit Combinational Circuit Shifter

A combinational circuit shifter can be constructed with multiplexers as shown in Figure below.

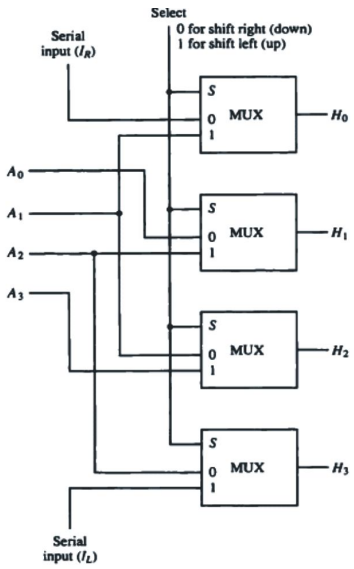


Figure : 4-bit combinational circuit shifter

- The 4-bit shifter has four data inputs, A0 through A3 and four data outputs, H0 through H3.
- There are two serial inputs, one for shift left (IL) and the other for shift right (IL).
- When the selection input S = 0, the input data are shifted right (down in the diagram).
- When S = 1, the input data are shifted left (up in the diagram).
- The function table in Figure shows which input goes to each output after the shift.

Function table

Select	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

- A shifter with n data inputs and outputs requires n multiplexers.
- The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.

2.1.6 Arithmetic Logic Shift Unit.

Q15. Draw and explain one stage of arithmetic logic shift unit.

Ans : **(Imp.)**

- Instead of having individual registers performing the micro operations directly, computer systems employ a number of storage registers connected to a common operational unit called an arithmetic logic unit, abbreviated ALU.
- To perform a microoperation, the contents of specified registers are placed in the inputs of the common ALU.
- The ALU performs an operation and the result of the operation is then transferred to a destination register.
- The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one dock pulse period.
- The arithmetic, logic, and shift circuits introduced in previous sections can be combined into one ALU with common selection variables.
- One stage of an arithmetic logic shift unit is shown in figure below:

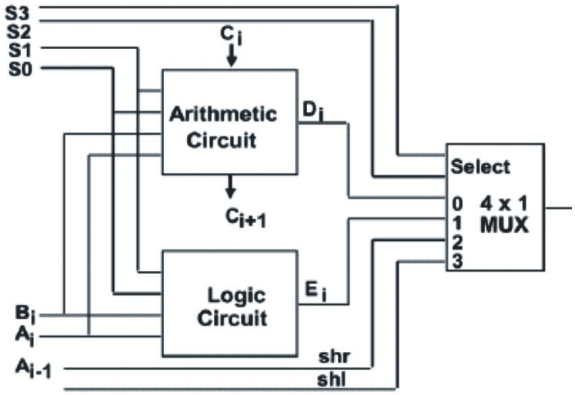


Figure : One stage of arithmetic logic shift unit

- The subscript i designates a typical stage. Inputs Ai and Bi are applied to both the arithmetic and logic units.
- A particular microoperation is selected with inputs S1 and S0.
- A 4 x 1 multiplexer at the output chooses between an arithmetic output in Di and a Logic output in Ei .
- The data in the multiplexer are selected with inputs S3 and S2.
- The other two data inputs to the multiplexer receive inputs Ai-1 for the shift-right operation and Ai + 1 for the shift-left operation.
- Note that the diagram shows just one typical stage. The circuit shown in figure must be repeated n times for an n-bit ALU.
- The outputs carry Ci + 1 of a given arithmetic stage must be connected to the input carry Cin of the next stage in sequence.
- The input carry to the first stage is the input carry On, which provides a selection variable for the arithmetic operations.
- The circuit whose one stage is specified in figure provides
  - 8 arithmetic operation
  - 4 logic operations
  - 2 shift operations
- Each operation is selected with the five variables S3, S2, Si, S0, and Cin. The input carry Cin is used for selecting an arithmetic operation only.
- Table below lists the 14 operations of the ALU.

Operation Select					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Increment A
0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	F = A + B + 1	Add with carry
0	0	1	0	0	F = A + $\overline{B}$	Subtract with borrow
0	0	1	0	1	F = A + $\overline{B}$ + 1	Subtraction
0	0	1	1	0	F = A − 1	Decrement A
0	0	1	1	1	F = A	Transfer A
0	1	0	0	X	F = A ∧ B	AND
0	1	0	1	X	F = A ∨ B	OR
0	1	1	0	X	F = A ⊕ B	XOR
0	1	1	1	X	F = $\overline{A}$	Complement A
1	0	X	X	X	I = shr A	Shift right A into I
1	1	X	X	X	I = shl A	Shift left A into I

Figure : Function Table for Arithmetic logic Shift Unit

- The first eight are arithmetic operations and are selected with S3S2 = 00.
- The next four are logic operations are selected with S3S2 = 01. The input carry has no effect during the logic operations and is marked with don't-care X's.
- The last two operations are shift operations and are selected with S3S2 = 10 and 11.
- The other three selection inputs have no effect on the shift.

2.2 Basic Computer Organization and Design

2.2.1 Instruction Codes

Q16. Define the following:

Ans :

i) Instruction Codes

ii) Operation Codes

iii) Accumulator

i) Instruction Code

An instruction code is a group of bits that instruct the computer to perform a specific operation.

ii) Operation Code

The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement. an instruction depends on the total number of operations available in the computer. The operation code must consist of at least n bits for a given 2n (or less) distinct operations.

iii) Accumulator (AC)

Computers that have a single-processor register usually assign to it the name accumulator (AC) accumulator and label it AC. The operation is performed with the memory operand and the content of AC.

Q17. Explain Stored Program Organization in detail.

Ans :

- The simplest way to organize a computer is to have one processor register and an instruction code format with two parts.
- The first part specifies the operation to be performed and the second specifies an address.
- The memory address tells the control where to find an operand in memory.
- This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.
- The following figure shows this type of organization.

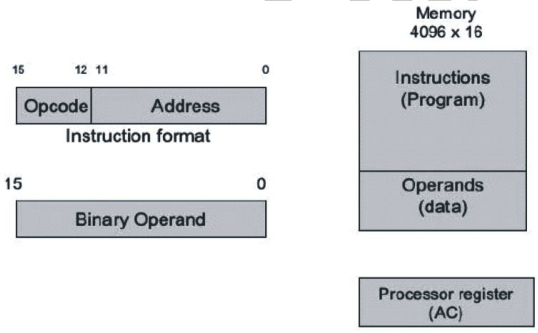


Figure : Stored Program Organization

- Instructions are stored in one section of memory and data in another.
- For a memory unit with 4096 words, we need 12 bits to specify an address since 212 = 4096.
- If we store each instruction code in one 16-bit memory word, we have available four bits for operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.
- The control reads a 16-bit instruction from the program portion of memory.
- It uses the 12-bit address part of the instruction to read a 16-bit operand from the data portion of memory.
- It then executes the operation specified by the operation code.
- Computers that have a single-processor register usually assign to it the name accumulator and label it AC.

- If an operation in an instruction code does not need an operand from memory, the rest of the bits in the instruction can be used for other purposes.
- For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register. They do not need an operand from memory. For these types of operations, the second part of the instruction code (bits 0 through 11) is not needed for specifying a memory address and can be used to specify other operations for the computer.

Q18. Explain Direct and Indirect addressing of basic computer.

Ans :

- The second part of an instruction format specifies the address of an operand, the instruction is said to have a **direct address**.
- In **Indirect address**, the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found.
- One bit of the instruction code can be used to distinguish between a direct and an indirect address.
- It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I.
- The mode bit is 0 for a direct address and 1 for an indirect address.
- A direct address instruction is shown in Figure (a). It is placed in address 22 in memory.
- The I bit is 0, so the instruction is recognized as a direct address instruction.
- The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457.
- The control finds the operand in memory at address 457 and adds it to the content of AC.
- The instruction in address 35 shown in Figure (b) has a mode bit I = 1, recognized as an indirect address instruction.
- The address part is the binary equivalent of 300.
- The control goes to address 300 to find the address of the operand. The address of the operand in this case is 1350. The operand found in address 1350 is then added to the content of AC.

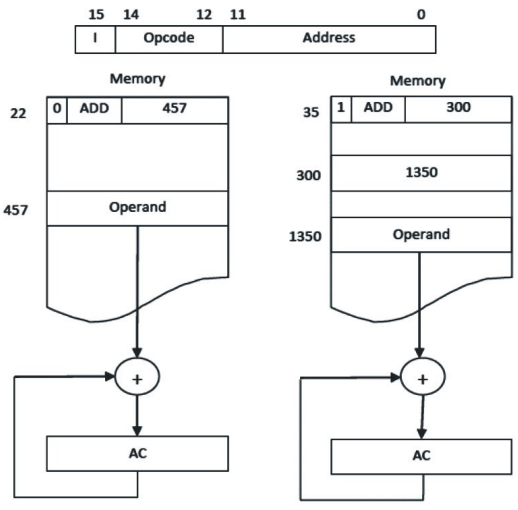


Figure (a) Direct Address Figure (b) Indirect Address

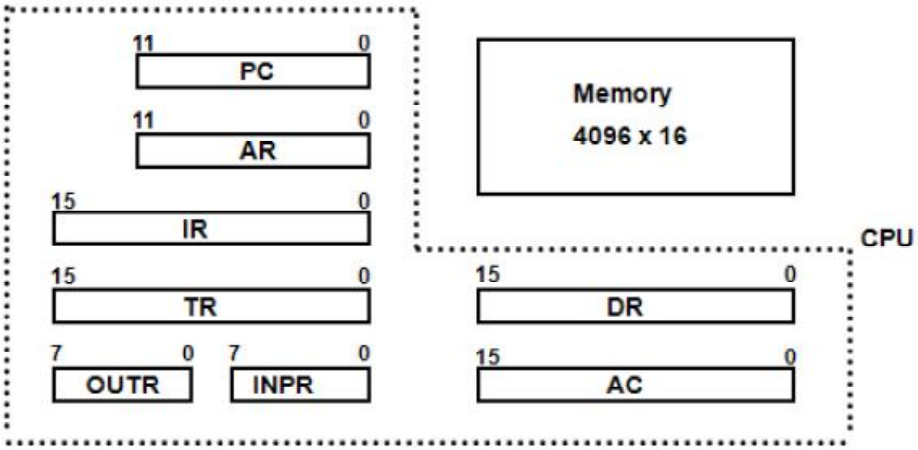
Direct Address	Indirect Address
When the second part of an instruction code specifies the address of an operand, the instruction is said to have a direct address.	When the second part of an instruction code specifies the address of a memory word in which the address of the operand, the instruction is said to have a direct address.
For instance the instruction MOV R0 00H. R0, when converted to machine language is the physical address of register R0. The instruction moves 0 to R0.	For instance the instruction MOV @R0 00H, when converted to machine language, @R0 becomes whatever is stored in R0, and that is the address used to move 0 to. It can be whatever is stored in R0.

2.2.2 Computer Registers

Q19. Explain briefly about Registers of basic computer.

Ans : **(Imp.)**

- It is necessary to provide a register in the control unit for storing the instruction code after it is read from memory.
- The computer needs processor registers for manipulating data and a register for holding a memory address.
- These requirements dictate the register configuration shown in Figure 2.4.



- The data register (DR) holds the operand read from memory.
- The accumulator (AC) register is a general purpose processing register.
- The instruction read from memory is placed in the instruction register (IR).
- The temporary register (TR) is used for holding temporary data during the processing.
- The memory address register (AR) has 12 bits.
- The program counter (PC) also has 12 bits and it holds the address of the next instruction to be read from memory after the current instruction is executed.

- Instruction words are read and executed in sequence unless a branch instruction is encountered. A branch instruction calls for a transfer to a nonconsecutive instruction in the program.
- Two registers are used for input and output. The input register (INPR) receives an 8-bit character from an input device. The output register (OUTR) holds an 8-bit character for an output device.

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

Table 2.1: List of Registers for Basic Computer

Q20. Draw and explain Common Bus System for basic computer register. What is the requirement of common bus System?

Ans :

- The basic computer has eight registers, a memory unit and a control unit.
- Paths must be provided to transfer information from one register to another and between memory and register.
- The number of wires will be excessive if connections are between the outputs of each register and the inputs of the other registers. An efficient scheme for transferring information in a system with many register is to use a common bus.
- The connection of the registers and memory of the basic computer to a common bus system is shown in figure.
- The outputs of seven registers and memory are connected to the common bus. The specific output that is selected for the bus lines at any given time is determined from the binary value of the selection variables S2, S1, and S0.
- The number along each output shows the decimal equivalent of the required binary selection.
- The particular register whose LD (load) input is enabled receives the data from the bus during the next clock pulse transition. The memory receives the contents of the bus when its write input is activated. The memory places its 16-bit output onto the bus when the read input is activated and S2 S1 S0 = 1 1 1.
- Four registers, DR, AC, IR, and TR have 16 bits each.
- Two registers, AR and PC, have 12 bits each since they hold a memory address.
- When the contents of AR or PC are applied to the 16-bit common bus, the four most
- Significant bits are set to 0's. When AR and PC receive information from the bus, only the 12 least significant bits are transferred into the register.

- The input register INPR and the output register OUTR have 8 bits each and communicate with the eight least significant bits in the bus. INPR is connected to provide information to the bus but OUTR can only receive information from the bus.

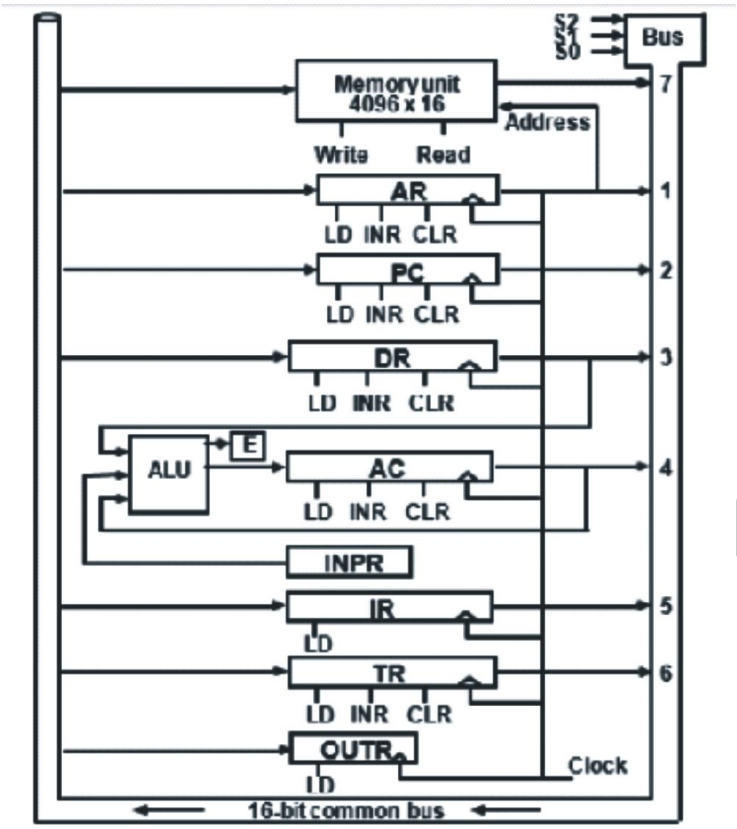


Figure : Basic computer registers connected to a common bus

- Five registers have three control inputs: LD (load), INR (increment), and CLR (clear). Two registers have only a LD input.
- AR must always be used to specify a memory address; therefore memory address is connected to AR.
- The 16 inputs of AC come from an adder and logic circuit. This circuit has three sets of inputs.
  - Set of 16-bit inputs come from the outputs of AC.
  - Set of 16-bits come from the data register DR.
  - Set of 8-bit inputs come from the input register INPR.
- The result of an addition is transferred to AC and the end carry-out of the addition is transferred to flip-flop E (extended AC bit).
- The clock transition at the end of the cycle transfers the content of the bus into the designated destination register and the output of the adder and logic circuit into AC.

2.2.3 Computer Instructions

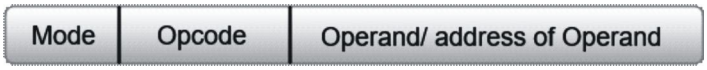
Q21. Explain different types of Computer Instructions formats.

Ans :

Computer instructions are a set of machine language instructions that a particular processor understands and executes. A computer performs tasks on the basis of the instruction provided.

An instruction comprises of groups called fields. These fields include:

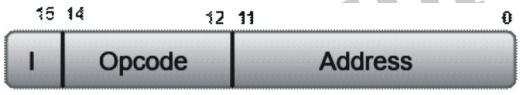
- o The Operation code (Opcode) field which specifies the operation to be performed.
- o The Address field which contains the location of the operand, i.e., register or memory location.
- o The Mode field which specifies how the operand will be located.



A basic computer has three instruction code formats which are:

1. Memory - Reference instruction
2. Register - Reference instruction
3. Input-Output instruction

1) Memory - Reference Instruction



(Opcode = 000 through 110)

In Memory-reference instruction, 12 bits of memory is used to specify an address and one bit to specify the addressing mode 'I'.

2) Register - Reference Instruction

(Opcode = 111, I = 0)

The Register-reference instructions are represented by the Opcode 111 with a 0 in the leftmost bit (bit 15) of the instruction.

Note: The Operation code (Opcode) of an instruction refers to a group of bits that define arithmetic and logic operations such as add, subtract, multiply, shift, and compliment.

A Register-reference instruction specifies an operation on or a test of the AC (Accumulator) register.

3) Input-Output Instruction

(Opcode = 111, I = 1)

Just like the Register-reference instruction, an Input-Output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of the input-output operation or test performed.

Note :

- The three operation code bits in positions 12 through 14 should be equal to 111. Otherwise, the instruction is a memory-reference type, and the bit in position 15 is taken as the addressing mode I.
- When the three operation code bits are equal to 111, control unit inspects the bit in position 15. If the bit is 0, the instruction is a register-reference type. Otherwise, the instruction is an input-output type having bit 1 at position 15.

Instruction Set Completeness

A set of instructions is said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

- Arithmetic, logical and shift instructions
- A set of instructions for moving information to and from memory and processor registers.
- Instructions which controls the program together with instructions that check status conditions.
- Input and Output instructions
- Arithmetic, logic and shift instructions provide computational capabilities for processing the type of data the user may wish to employ.
- A huge amount of binary information is stored in the memory unit, but all computations are done in processor registers. Therefore, one must possess the capability of moving information between these two units.
- Program control instructions such as branch instructions are used change the sequence in which the program is executed.
- Input and Output instructions act as an interface between the computer and the user. Programs and data must be transferred into memory, and the results of computations must be transferred back to the user.

2.2.4 Timing and Control

Q22. Explain the basic working principle of the Control Unit with timing diagram.

Ans :

The block diagram of the control unit is shown in figure

Components of Control unit are

1. Two decoders
2. A sequence counter
3. Control logic gates

- An instruction read from memory is placed in the instruction register (IR). In control unit the IR is divided into three parts: I bit, the operation code (12-14)bit, and bits 0 through 11.
- The operation code in bits 12 through 14 are decoded with a 3 X 8 decoder.

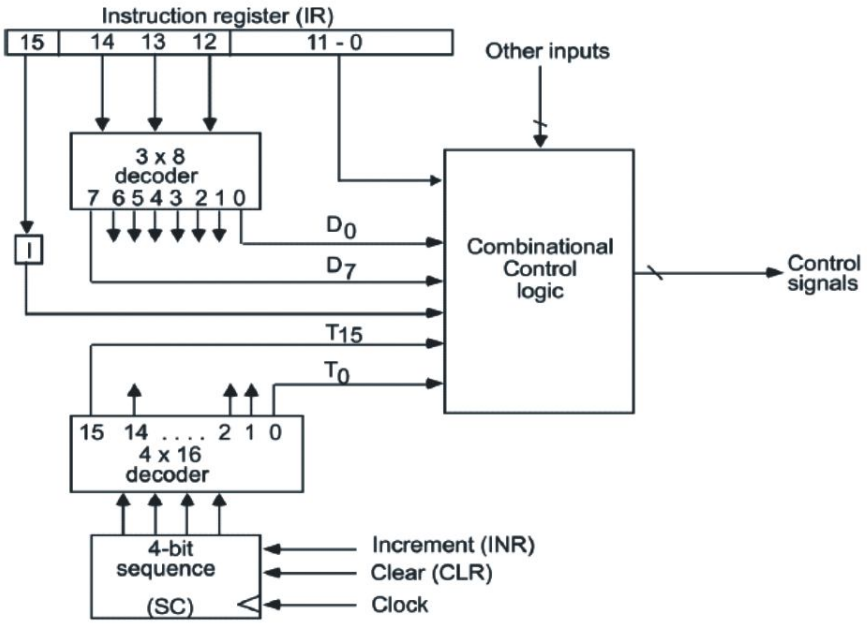


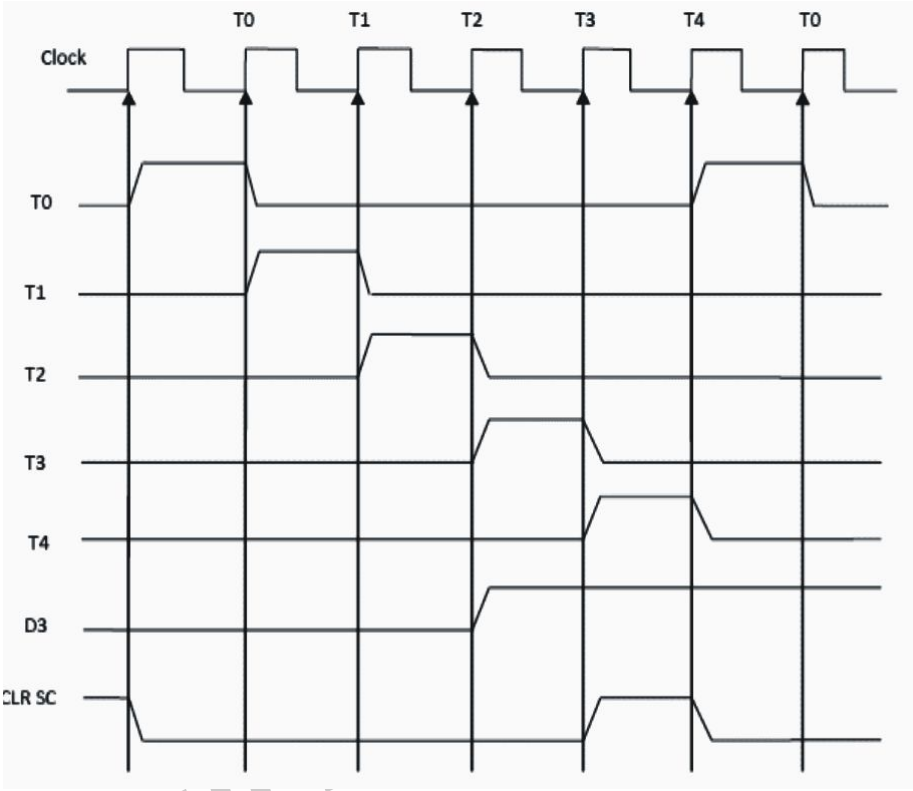
Figure : Control unit of basic computer

- Bit-15 of the instruction is transferred to a flip-flop designated by the symbol I.
- The eight outputs of the decoder are designated by the symbols D0 through D7. Bits 0 through 11 are applied to the control logic gates. The 4†bit sequence counter can count in binary from 0 through 15.The outputs of counter are decoded into 16 timing signals T0 through T15.
- The sequence counter SC can be incremented or cleared synchronously. Most of the
- Time, the counter is incremented to provide the sequence of timing signals out of 4 X 16 decoder. Once in awhile, the counter is cleared to 0, causing the next timing signal to be T0.
- As an example, consider the case where SC is incremented to provide timing signals T0, T1, T2, T3 and T4 in sequence. At time T4, SC is cleared to 0 if decoder output D3 is active.
- This is expressed symbolically by the statement D3T4: SC ‡ 0

Timing Diagram:

- The timing diagram figure2.8 shows the time relationship of the control signals.
- The sequence counter SC responds to the positive transition of the clock.
- Initially, the CLR input of SC is active.
- The first positive transition of the clock clears SC to 0, which in turn activates the timing T0 out of the decoder. T0 is active during one clock cycle. The positive clock transition labelled T0 in the diagram will trigger only those registers whose control inputs are connected to timing signal T0.

- SC is incremented with every positive clock transition, unless its CLR input is active.
- This procedures the sequence of timing signals T0, T1, T2, T3 and T4, and so on. If SC is not cleared, the timing signals will continue with T5, T6, up to T15 and back to T0.



- The last three waveforms shows how SC is cleared when D3T4 = 1. Output D3 from the operation decoder becomes active at the end of timing signal T2. When timing signal T4 becomes active, the output of the AND gate that implements the control function D3T4 becomes active.
- This signal is applied to the CLR input of SC. On the next positive clock transition the counter is cleared to 0. This causes the timing signal T0 to become active instead of T5 that would have been active if SC were incremented instead of cleared.

2.2.5 Instruction Cycle

Q23. Draw and explain the flowchart for instruction cycle.

Ans :

- A program residing in the memory unit of the computer consists of a sequence of instructions. In the basic computer each instruction cycle consists of the following phases:

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

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

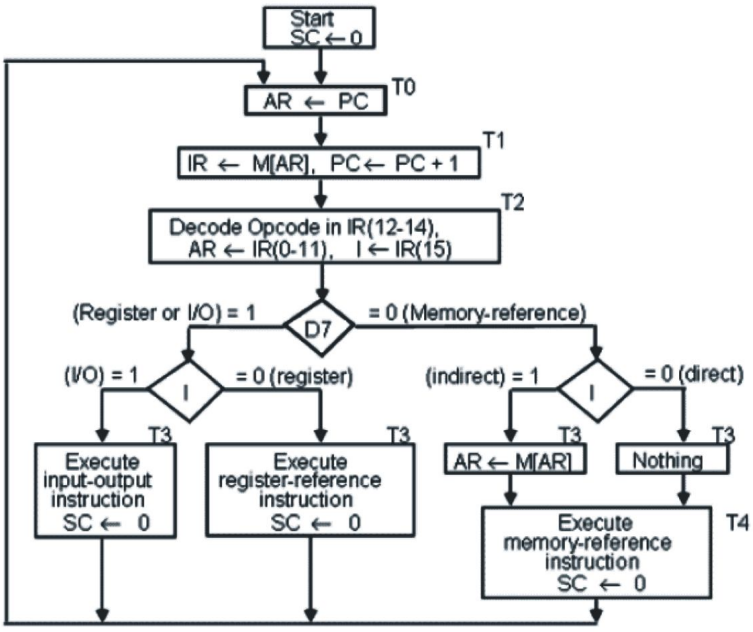


Figure : Flowchart for instruction cycle (initial configuration)

- The flowchart presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding.
- If D7 = 1, the instruction must be register-reference or input-output type. If D7 = 0, the operation code must be one of the other seven values 110, specifying a memory reference instruction. Control then inspects the value of the first bit of the instruction, which now available in flip-flop I.
- If D7 = 0 and I = 1, we have a memory-reference instruction with an indirect address. It is then necessary to read the effective address from memory.
- The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T3.This can be symbolized as follows:
  - D'7 I T3: AR M [AR]
  - D'7 I' T3: Nothing
  - D7 I' T3: Execute a register-reference instruction
  - D7 I T3: Execute an input-output instruction
- When a memory-reference instruction with I = 0 is encountered, it is not necessary to do anything since the effective address is already in AR.
- However, the sequence counter SC must be incremented when D'7 I T3 = 1, so that the execution of the memory-reference instruction can be continued with timing variable T4.
- A register-reference or input-output instruction can be executed with the click associated with timing signal T3. After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with T0 =1. SC is either incremented or cleared to 0 with every positive clock transition.

Q24. List and explain register reference instruction.

Ans :

- When the register-reference instruction is decoded, D7 bit is set to 1.
- Each control function needs the Boolean relation D7 I' T3

- There are 12 register-reference instructions listed below:

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

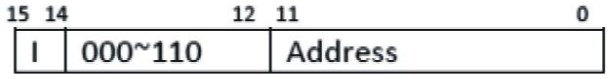
- These 12 bits are available in IR (0-11). They were also transferred to AR during time T2.
- These instructions are executed at timing cycle T3.
- The first seven register-reference instructions perform clear, complement, circular shift, and increment microoperations on the AC or E registers.
- The next four instructions cause a skip of the next instruction in sequence when condition is satisfied. The skipping of the instruction is achieved by incrementing PC.
- The condition control statements must be recognized as part of the control conditions.
- The AC is positive when the sign bit in AC(15) = 0; it is negative when AC(15) = 1. The content of AC is zero (AC = 0) if all the flip-flops of the register are zero.
- The HLT instruction clears a start-stop flip-flop S and stops the sequence counter from counting. To restore the operation of the computer, the start-stop flip-flop must be set manually.

2.2.6 Memory reference instruction

Q25. List and explain memory reference instructions.

Ans :

When the memory-reference instruction is decoded, D7 bit is set to 0.



- The following table lists seven memory-reference instructions.

Symbol	Operation Decoder	Symbolic Description
AND	D_0	AC ← AC ∧ M[AR]
ADD	D_1	AC ← AC + M[AR], E ← C_{out}
LDA	D_2	AC ← M[AR]
STA	D_3	M[AR] ← AC
BUN	D_4	PC ← AR
BSA	D_5	M[AR] ← PC, PC ← AR + 1
ISZ	D_6	M[AR] ← M[AR] + 1, if M[AR] + 1 = 0 then PC ← PC+1

- The effective address of the instruction is in the address register AR and was placed there during timing signal T2 when I = 0, or during timing signal T3 when I = 1.
- The execution of the memory-reference instructions starts with timing signal T4.

AND to AC

- This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC.
  - D0T4: DR → M[AR]
  - D0T5: AC → AC + DR, SC → 0

ADD to AC

- This instruction adds the content of the memory word specified by the effective address to the value of AC. The sum is transferred into AC and the output carry Cout is transferred to the E (extended accumulator) flip-flop.
  - D1T4: DR → M[AR]
  - D1T5: AC → AC + DR, E → Cout, SC → 0

LDA: Load to AC

- This instruction transfers the memory word specified by the effective address to AC.
  - D2T4: DR → M[AR]
  - D2T5: AC → DR, SC → 0

STA: Store AC

- This instruction stores the content of AC into the memory word specified by the effective address.
  - D3T4: M[AR] → AC, SC → 0

BUN: Branch Unconditionally

- This instruction transfers the program to instruction specified by the effective address.
- The BUN instruction allows the programmer to specify an instruction out of sequence and the program branches (or jumps) unconditionally.
- D4T4: PC ⟶ AR, SC ⟶ 0

BSA: Branch and Save Return Address

- This instruction is useful for branching to a portion of the program called a subroutine or procedure. When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address.
  - M[AR] ⟶ PC, PC ⟶ AR + 1
  - M[135] ⟶ 21, PC ⟶ 135 + 1 = 136

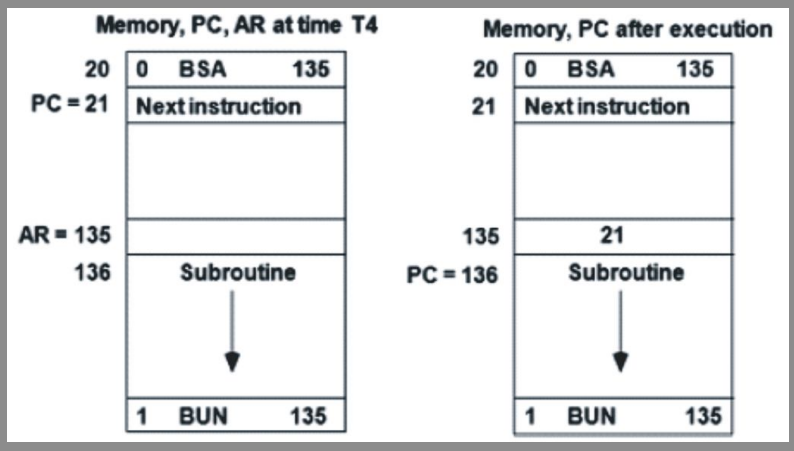


Figure : Example of BSA instruction exectution

- It is not possible to perform the operation of the BSA instruction in one clock cycle when we use the bus system of the basic computer. To use the memory and the bus properly, the BSA instruction must be executed with a sequence of two micro operations:
  - D5T4: M[AR] ⟶ PC, AR ⟶ AR + 1
  - D5T5: PC ⟶ AR, SC ⟶ 0

ISZ: Increment and Skip if Zero

- These instruction increments the word specified by the effective address, and if the
- incremented value is equal to 0, PC is incremented by 1. Since it is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory.
  - D6T4: DR ⟶ M[AR]
  - D6T5: DR ⟶ DR + 1
  - D6T4: M[AR] ⟶ DR, if (DR = 0) then (PC ⟶ PC + 1), SC ⟶ 0

Control Flowchart

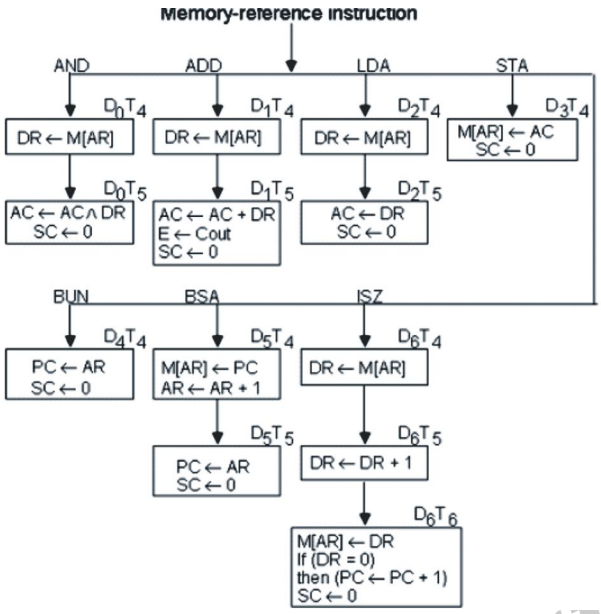


Figure : Flowchart for memory-reference instructions

2.2.7 Input-Output and Interrupt

Q26. Draw and explain input-output configuration of basic computer.

Ans : **(Imp.)**

- A computer can serve no useful purpose unless it communicates with the external environment.
- To exhibit the most basic requirements for input and output communication, we will use a terminal unit with a keyboard and printer.

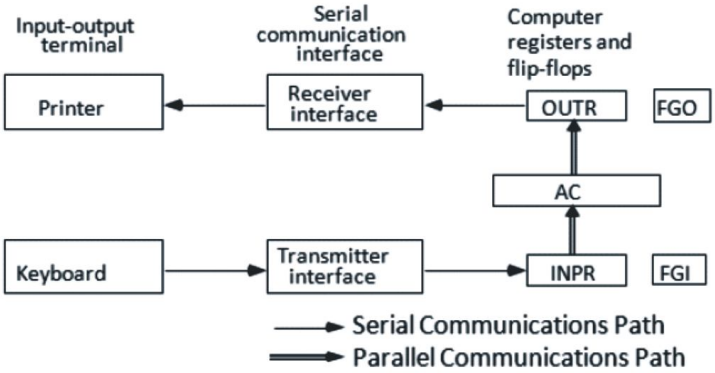


Figure : Input -output configuration

- The terminal sends and receives serial information and each quantity of information has eight bits of an alphanumeric code.
- The serial information from the keyboard is shifted into the input register INPR.

- The serial information for the printer is stored in the output register OUTR.
- These two registers communicate with a communication interface serially and with the AC in parallel.
- The transmitter interface receives serial information from the keyboard and transmits it to INPR. The receiver interface receives information from OUTR and sends it to the printer serially.
- The 1-bit input flag FGI is a control flip-flop. It is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.
- The flag is needed to synchronize the timing rate difference between the input device and the computer.
- The process of information transfer is as follows:

The process of input information transfer:

- Initially, the input flag FGI is cleared to 0. When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and the input flag FGI is set to 1.
- As long as the flag is set, the information in INPR cannot be changed by striking another key. The computer checks the flag bit; if it is 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0.
- Once the flag is cleared, new information can be shifted into INPR by striking another key.

The process of outputting information:

- The output register OUTR works similarly but the direction of information flow is reversed.
- Initially, the output flag FGO is set to 1. The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.
- The computer does not load a new character into OUTR when FGO is 0 because this condition indicates that the output device is in the process of printing the character.

Q27. Explain Input-Output instructions.

Ans :

- Input and output instructions are needed for transferring information to and from AC register, for checking the flag bits, and for controlling the interrupt facility.
- Input-output instructions have an operation code 1111 and are recognized by the control when D7 = 1 and I = 1.
- The remaining bits of the instruction specify the particular operation.
- The control functions and microoperations for the input-output instructions are listed below.

INP	AC(0-7) ← INPR, FGI ← 0	Input char. to AC
OUT	OUTR ← AC(0-7), FGO ← 0	Output char. from AC
SKI	if(FGI = 1) then (PC ← PC + 1)	Skip on input flag
SKO	if(FGO = 1) then (PC ← PC + 1)	Skip on output flag
ION	IEN ← 1	Interrupt enable on
IOF	IEN ← 0	Interrupt enable off

Table : Input Output Instructions

- The INP instruction transfers the input information from INPR into the eight low-order bits of AC and also clears the input flag to 0.
- The OUT instruction transfers the eight least significant bits of AC into the output register OUTR and clears the output flag to 0.
- The next two instructions in Table check the status of the flags and cause a skip of the next instruction if the flag is 1.
- The instruction that is skipped will normally be a branch instruction to return and check the flag again.
- The branch instruction is not skipped if the flag is 0. If the flag is 1, the branch instruction is skipped and an input or output instruction is executed.
- The last two instructions set and clear an interrupt enable flip-flop IEN. The purpose of IEN is explained in conjunction with the interrupt operation.

Q28. What is an Interrupt Cycle? Draw and explain flow chart of it.

Ans :

- The way that the interrupt is handled by the computer can be explained by means of the flowchart shown in figure 2.13.
- An interrupt flip-flop R is included in the computer.
- When R = 0, the computer goes through an instruction cycle.
- During the execute phase of the instruction cycle IEN is checked by the control.
- If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle.
- If IEN is 1, control checks the flag bits.
- If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information.
- In this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN = 1, flip-flop R is set to 1.
- At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

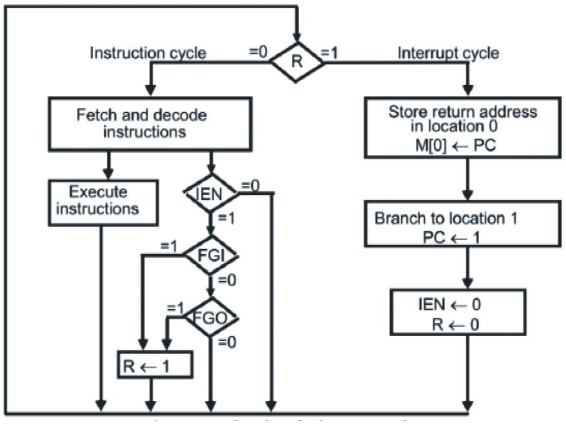


Figure : Flowchart for interrupt cycle

Interrupt Cycle

- The interrupt cycle is a hardware implementation of a branch and save return address operation.
- The return address available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted. This location may be a processor register, a memory stack, or a specific memory location.
- Here we choose the memory location at address 0 as the place for storing the return address.
- Control then inserts address 1 into PC and clears IEN and R so that no more interruptions can occur until the interrupt request from the flag has been serviced.
- An example that shows what happens during the interrupt cycle is shown in Figure .

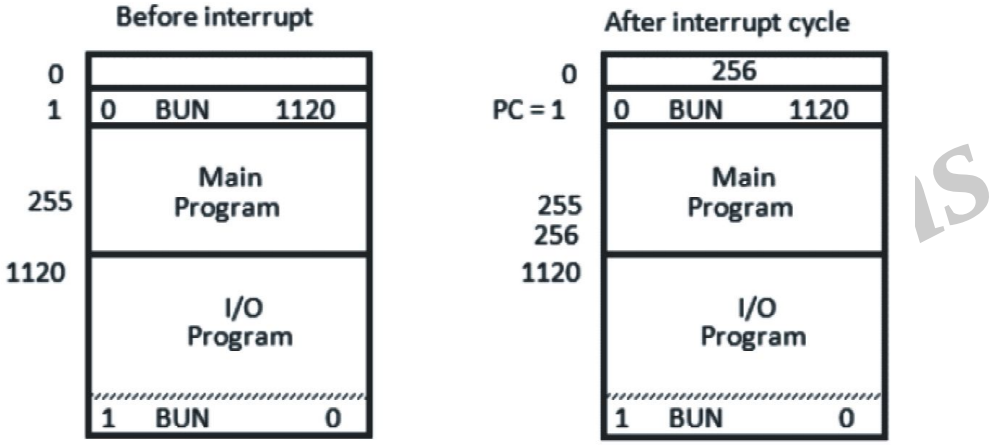


Figure : Demonstration of the interrupt cycle

- Suppose that an interrupt occurs and R = 1, while the control is executing the instruction at address 255. At this time, the return address 256 is in PC.
- The programmer has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1.
- The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0.
- At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC. The branch instruction at address 1 causes the program to transfer to the input-output service program at address 1120.
- This program checks the flags, determines which flag is set, and then transfers the required input or output information. Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted.
- The instruction that returns the computer to the original place in the main program is a branch indirect instruction with an address part of 0. This instruction is placed at the end of the I/O service program.
- The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

Register transfer statements for the interrupt cycle

- The flip-flop is set to 1 if IEN = 1 and either FGI or FGO are equal to 1. This can happen with any clock transition except when timing signals T0, T1 or T2 are active.

- The condition for setting flip-flop R = 1 can be expressed with the following register
- transfer statement:
  - $T0 \rightarrow T1 \rightarrow T2 \rightarrow IEN) (FGI + FGO): R \rightarrow 1$
- The symbol + between FGI and FGO in the control function designates a logic OR operation. This is AND with IEN and $T0 \rightarrow T1 \rightarrow T2$.
- The fetch and decode phases of the instruction cycle must be modified and Replace T0, T1, T2 with R'T0, R'T1, R'T2
- Therefore the interrupt cycle statements are :
  - $RT0: AR \rightarrow 0, TR \rightarrow PC$
  - $RT1: M[AR] \rightarrow TR, PC \rightarrow 0$
  - $RT2: PC \rightarrow PC + 1, IEN \rightarrow 0, R \rightarrow 0, SC \rightarrow 0$
- During the first timing signal AR is cleared to 0, and the content of PC is transferred to the temporary register TR.
- With the second timing signal, the return address is stored in memory at location 0 and PC is cleared to 0.
- The third timing signal increments PC to 1, clears IEN and R, and control goes back to T0 by clearing SC to 0.
- The beginning of the next instruction cycle has the condition RT0 and the content of PC is equal to 1. The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.

UNIT III

Micro programmed Control: Control memory, Address Sequencing, Micro program example, Design of Control Unit.

Central Processing Unit: General Register Organization, Stack Organization, Instruction formats, Addressing modes, Data Transfer and Manipulation, and Program control. Computer Arithmetic: Addition and Subtraction, Multiplication, Division, and Floating Point Arithmetic Operations.

3.1 Micro Programmed Control

3.1.1 Control memory

Q1. Define the following.

Ans :

a) Hardwired Control Unit

b) Micro programmed control unit

c) Dynamic microprogramming

d) Control Memory

e) Writeable Control Memory

f) Control Word

(a) Hardwired Control Unit

When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired.

(b) Micro programmed control unit

A control unit whose binary control variables are stored in memory is called a micro programmed control unit.

(c) Dynamic microprogramming

A more advanced development known as dynamic microprogramming permits a micro-program to be loaded initially from an auxiliary memory such as a magnetic disk. Control units that use dynamic microprogramming employ a writable control memory. This type of memory can be used for writing.

(d) Control Memory

Control Memory is the storage in the micro-programmed control unit to store the micro-program.

(e) Writeable Control Memory

Control Storage whose contents can be modified, allow the change in microprogram and Instruction set can be changed or modified is referred as Writeable Control Memory**.**

(f) Control Word

The control variables at any given time can be represented by a control word string of 1 ‘s and 0’s called a control word.

Q2. Describe the following terms: Micro-operation, Micro-instruction, Micro-program, Micro-code.

Ans :

i) Micro-operations:

- In computer central processing units, micro-operations (also known as a microops or ¼ops) are detailed low-level instructions used in some designs to implement complex machine instructions (sometimes termed macro-instructions in this context).

ii) Micro instruction:

- A symbolic microprogram can be translated into its binary equivalent by means of an assembler.
- Each line of the assembly language microprogram defines a symbolic microinstruction.
- Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD.

iii) Micro program:

- A sequence of microinstructions constitutes a microprogram.
- Since alterations of the microprogram are not needed once the control unit is in operation, the control memory can be a read-only memory (ROM).
- ROM words are made permanent during the hardware production of the unit.
- The use of a micro program involves placing all control variables in words of ROM for use by the control unit through successive read operations.
- The content of the word in ROM at a given address specifies a microinstruction.

iv) Microcode:

- Microinstructions can be saved by employing subroutines that use common sections of microcode.
- For example, the sequence of micro operations needed to generate the effective
- address of the operand for an instruction is common to all memory reference instructions.
- This sequence could be a subroutine that is called from within many other routines to execute the effective address computation.

Q3. Draw and explain the organization of micro programmed control unit.

Ans :

- The general configuration of a micro-programmed control unit is demonstrated in the block diagram of Figure .
- The control memory is assumed to be a ROM, within which all control information is permanently stored.

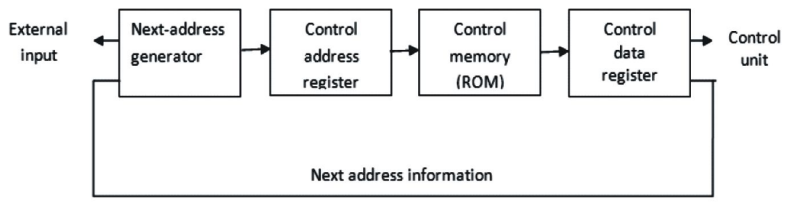


Figure : Micro programmed control organization

- The control memory address register specifies the address of the microinstruction, and the control data register holds the microinstruction read from memory.
- The microinstruction contains a control word that specifies one or more micro-operations for the data processor. Once these operations are executed, the control must determine the next address.
- The location of the next microinstruction may be the one next in sequence, or it may be located somewhere else in the control memory.
- While the microoperations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction.
- Thus a microinstruction contains bits for initiating microoperations in the data processor part and bits that determine the address sequence for the control memory.
- The next address generator is sometimes called a micro-program sequencer, as it determines the address sequence that is read from control memory.
- Typical functions of a micro-program sequencer are incrementing the control address register by one, loading into the control address register an address from control memory, transferring an external address, or loading an initial address to start the control operations.
- The control data register holds the present microinstruction while the next address is computed and read from memory.
- The data register is sometimes called a pipeline register.
- It allows the execution of the microoperations specified by the control word simultaneously with the generation of the next microinstruction.
- This configuration requires a two-phase clock, with one clock applied to the address register and the other to the data register.
- The main advantage of the micro programmed control is the fact that once the hardware configuration is established; there should be no need for further hardware or wiring changes.
- If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory.

3.1.2 Address Sequencing

Q4. Explain the steps of Address Sequencing in detail.

Ans : **(Imp.)**

- Microinstructions are stored in control memory in groups, with each group specifying a routine.
- To appreciate the address sequencing in a micro-program control unit, let us specify the steps that the control must undergo during the execution of a single computer instruction.

Step-1:

- An initial address is loaded into the control address register when power is turned on in the computer.
- This address is usually the address of the first microinstruction that activates the instruction fetch routine.
- The fetch routine may be sequenced by incrementing the control address register through the rest of its microinstructions.

- At the end of the fetch routine, the instruction is in the instruction register of the computer.

Step-2:

- The control memory next must go through the routine that determines the effective address of the operand.
- A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers.
- The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on the status of the mode bits of the instruction.
- When the effective address computation routine is completed, the address of the operand is available in the memory address register.

Step-3:

- The next step is to generate the microoperations that execute the instruction fetched from memory.
- The microoperation steps to be generated in processor registers depend on the operation code part of the instruction.
- Each instruction has its own micro-program routine stored in a given location of control memory.
- The transformation from the instruction code bits to an address in control memory where the routine is located is referred to as a mapping process.
- A mapping procedure is a rule that transforms the instruction code into a control memory address.

Step-4:

- Once the required routine is reached, the microinstructions that execute the instruction may be sequenced by incrementing the control address register.
- Micro-programs that employ subroutines will require an external register for storing thereturn address.
- Return addresses cannot be stored in ROM because the unit has no writing capability.
- When the execution of the instruction is completed, control must return to the fetch routine.
- This is accomplished by executing an unconditional branch microinstruction to the first address of the fetch routine.

In summary, the address sequencing capabilities required in a control memory are:

1. Incrementing of the control address register.
2. Unconditional branch or conditional branch, depending on status bit conditions.
3. A mapping process from the bits of the instruction to an address for control memory.
4. A facility for subroutine call and return.

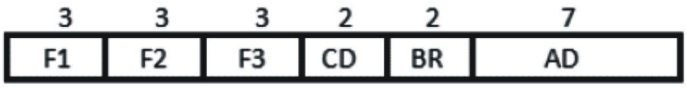
3.1.3 Micro Program Example

Q5. Explain Microinstruction Format in detail.

Ans :

The microinstruction format for the control memory is shown in figure. The 20 bits of the microinstruction are divided into four functional parts as follows:

1. The three fields F1, F2, and F3 specify microoperations for the computer. The microoperations are subdivided into three fields of three bits each. The three bits in each field are encoded to specify seven distinct microoperations. This gives a total of 21 microoperations.
2. The CD field selects status bit conditions.
3. The BR field specifies the type of branch to be used.
4. The AD field contains a branch address. The address field is seven bits wide, since the control memory has 128 = 27 words.



F1, F2, F3: Microoperation fields
CD: Condition for branching
BR: Branch field
AD: Address field

Figure : Microinstruction Format

- As an example, a microinstruction can specify two simultaneous microoperations from
  - F2 and F3 and none from F1.
  - DR → M[AR] with F2 = 100
  - PC → PC + 1 with F3 = 101
- The nine bits of the microoperation fields will then be 000 100 101.
- The CD (condition) field consists of two bits which are encoded to specify four status bit conditions as listed in Table 4.1.

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of AC
11	AC = 0	Z	Zero value in AC

Figure : Condition Field

- The BR (branch) field consists of two bits. It is used, in conjunction with the address field AD, to choose the address of the next microinstruction shown in Table .

BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1
		CAR ← CAR + 1 if condition = 0
01	CALL	CAR ← AD, SBR ← CAR + 1 if condition = 1
		CAR ← CAR + 1 if condition = 0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Table : Branch Field

Q6. Explain briefly about Symbolic Microinstruction.

Ans :

- Each line of the assembly language microprogram defines a symbolic microinstruction.
- Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR, and AD. The fields specify the following Table.

1.	Label	The label field may be empty or it may specify a symbolic address. A label is terminated with a colon (:).
2.	Microoperations	It consists of one, two, or three symbols, separated by commas, from those defined in Table 5.3. There may be no more than one symbol from each F field. The NOP symbol is used when the microinstruction has no microoperations. This will be translated by the assembler to nine zeros.
3.	CD	The CD field has one of the letters U, I, S, or Z.
4.	BR	The BR field contains one of the four symbols
5.	AD	The AD field specifies a value for the address field of the microinstruction in one of three possible ways: i. With a symbolic address, this must also appear as a label. ii. With the symbol NEXT to designate the next address in sequence. iii. When the BR field contains a RET or MAP symbol, the AD field is left empty and is converted to seven zeros by the assembler.

Figure : Symbolic Microinstruction

3.1.4 Design of Control Unit

Q7. Draw the diagram of Micro programmed sequencer for a control memory and explain it.

Ans : **(Imp.)**

Micro-program Sequencer

- The basic components of a micro-programmed control unit are the control memory and the circuits that select the next address.
- The address selection part is called a micro-program sequencer.
- A micro-program sequencer can be constructed with digital functions to suit a particular application.
- To guarantee a wide range of acceptability, an integrated circuit sequencer must provide an internal organization that can be adapted to a wide range of applications.
- The purpose of a microprogram sequencer is to present an address to the control memory so that a microinstruction may be read and executed.
- Commercial sequencers include within the unit an internal register stack used for temporary storage of addresses during microprogram looping and subroutine calls.

- Some sequencers provide an output register which can function as the address register for the control memory.
- The block diagram of the microprogram sequencer is shown in figure 4.6.
- There are two multiplexers in the circuit.
- The first multiplexer selects an address from one of four sources and routes it into a control address register CAR.
- The second multiplexer tests the value of a selected status bit and the result of the test is applied to an input logic circuit.
- The output from CAR provides the address for the control memory.
- The content of CAR is incremented and applied to one of the multiplexer inputs and to the subroutine registers SBR.
- The other three inputs to multiplexer 1 come from the address field of the presentmicroinstruction, from the output of SBR, and from an external source that maps the instruction.
- Although the figure shows a single subroutine register, a typical sequencer will have a register stack about four to eight levels deep. In this way, a number of subroutines can be active at the same time.
- The CD (condition) field of the microinstruction selects one of the status bits in the second multiplexer.
- If the bit selected is equal to 1, the T (test) variable is equal to 1; otherwise, it is equal to 0.
- The T value together with the two bits from the BR (branch) field goes to an input logic circuit.
- The input logic in a particular sequencer will determine the type of operations that are available in the unit.

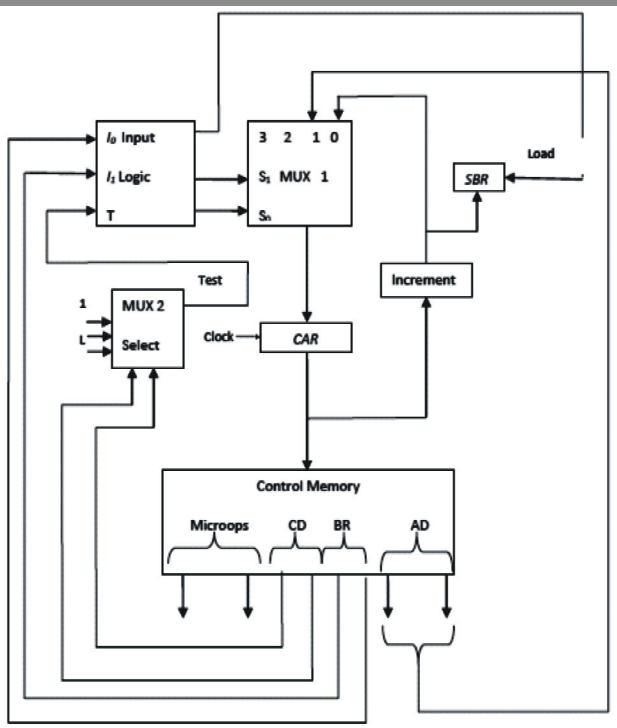


Figure : Microprogram Sequencer for a control memory

Input Logic : Truth Table

BR	Input			MUX 1		Load SBR
	I_1	I_0	T	S_1	S_0	L
0 0	0	0	0	0	0	0
0 0	0	0	1	0	1	0
0 1	0	1	0	0	0	0
0 1	0	1	1	0	1	1
1 0	1	0	X	1	0	0
1 1	1	1	X	1	1	0

Table 4.4: Input Logic Truth Table for Microprogram Sequencer

Boolean Function:

S0 = I0

S1 = I0I1 + I0'T

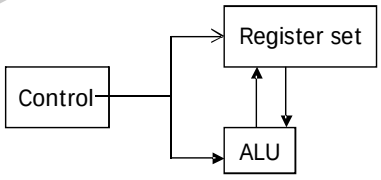
L = I0'I1T

- Typical sequencer operations are: increment, branch or jump, call and return from subroutine, load an external address, push or pop the stack, and other address sequencing operations.
- With three inputs, the sequencer can provide up to eight address sequencing operations.
- Some commercial sequencers have three or four inputs in addition to the T input and thus provide a wider range of operations.

3.2 CENTRAL PROCESSING UNIT

Q8. What is central processing unit. Explain about it.

Ans : **(Imp.)**



- The CPU is made up of 3 major Components.
- The CPU performs a variety of functions dictated by the type of instructions that are incorporated in the computer
- In programming, memory locations are needed for storing pointers, counters, return addresses, temporary result , etc. Memory access is most time consuming operation in a computer.
- It is then more convenient and more efficient to store these intermediate values in processor registers, which are connected through common bus system.

1. Accumulator based CPU

- **Characteristics** :
  - Initially, computers had accumulator based CPUs.
  - It is a simple CPU in which the accumulator contains an operand for the Instruction.

- The instruction leaves the result in the accumulator.
- These CPUs have zero address & single address instruction.

The Advantages :

- Short Instruction & less memory space.
- Instruction cycle takes less time because it saves time in Instruction fetching due to the absence of operand fetch.

The Disadvantages :

- Program Size increases, memory size increases.
- Program execution time increases due to increase in program size.

2. Register based CPU

Characteristics:

- Multiple registers are used.
- The use of registers result in short programs with limited instructions.

The Advantages :

- Shorter Program size
- Increase in the number of registers, increases CPU efficiency.

The Disadvantages :

- Additional memory accesses are needed.
- Compilers need to be more efficient in this aspect.

3.2.1 General Register Organization

Q9. Explain about general register organization.

Ans :

General Register Organization:

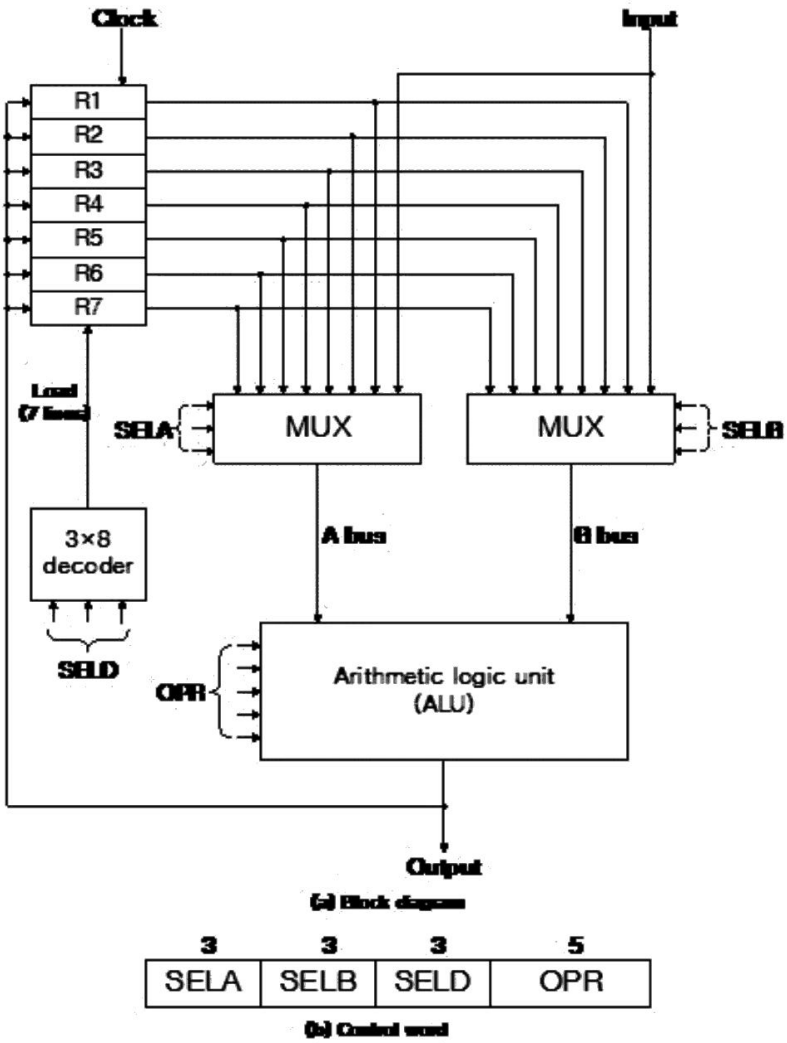
The number of registers in a processor unit may vary from just one processor register to as many as 64 registers or more.

1. One of the CPU registers is called as an accumulator AC or 'A' register. It is the main operand register of the ALU.
2. The data register (DR) acts as a buffer between the CPU and main memory. It is used as an input operand register with the accumulator.
3. The instruction register (IR) holds the opcode of the current instruction.
4. The address register (AR) holds the address of the memory in which the operand resides.
5. The program counter (PC) holds the address of the next instruction to be fetched for execution.

- Additional addressable registers can be provided for storing operands and address. This can be viewed as replacing the single accumulator by a set of registers. If the registers are used for many purpose, the resulting computer is said to have general register organization. In the case of processor registers, a registers is selected by the multiplexers that form the buses.

- When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfers, but also while performing various micro-operations. Hence it is necessary to provide a common unit that can perform all the arithmetic, logic and shift micro-operation in the processor.

A Bus Organization for Seven CPU Registers



The output of each register is connected to true multiplexer (mux) to form the two buses A & B.

- The selection lines in each multiplexer select one register or the input data for the particular bus.
- The A and B buses forms the input to a common ALU. The operation selected in the ALU determines the arithmetic or logic micro-operation that is to be performed.
- The result of the micro-operation is available for output and also goes into the inputs of the registers.
- The register that receives the information from the output bus is selected by a decoder.
- The decoder activates one of the register load inputs, thus providing a transfer both between the data in the output bus and the inputs of the selected destination register.

OPERATION OF CONTROL UNIT

- The control unit that operates the CPU bus system directs the information flow through the registers and ALU by selecting the various components in the systems.

R1 ® R2 + R3

(1) MUX A selection (SEC A): to place the content of R2 into bus A

(2) MUX B selection (sec B): to place the content of R3 into bus B

(3) ALU operation selection (OPR): to provide the arithmetic addition (A + B)

(4) Decoder destination selection (SEC D): to transfer the content of the output bus into R1

- These form the control selection variables are generated in the control unit and must be available at the beginning of a clock cycle. The data from the two source registers propagate through the gates in the multiplexer and the ALU, to the output bus, and into the into of the destination registers, all during the clock cycle intervals.

Example: R1 ⟸ R2 + R3

[1] MUX A selector (SELA): BUS A ⟸ R2

[2] MUX B selector (SELB): BUS B ⟸ R3

[3] ALU operation selector (OPR): ALU to ADD

[4] Decoder destination selector (SELD): R1 ⟸ Out Bus

Control Word

Encoding of register selection fields

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

ALU Control

Encoding of ALU operations

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

Examples of ALU Microoperations

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

3.2.2 Stack Organization

Q10. What is stack? Give the organization of register stack with all necessary elements and explain the working of push and pop operations.

Ans : **(Imp.)**

Stack Organization:

- A stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved.
- The stack in digital computers is essentially a memory unit with an address register that can count only. The register that holds the address for the stack is called a stack pointer
- (SP) because its value always points at the top item in the stack.
- The physical registers of a stack are always available for reading or writing. It is the content of the word that is inserted or deleted.

Register Stack:

Register stack:

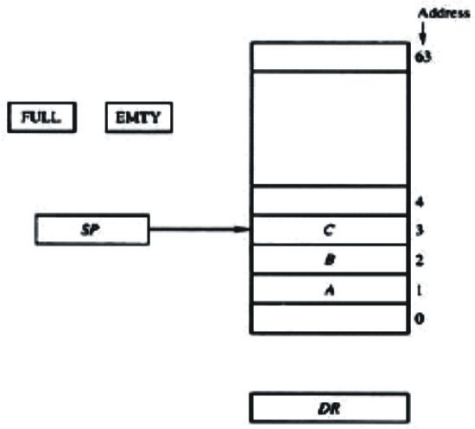


Figure : Block diagram of a 64-word stack

- A stack can be placed in a portion of a large memory or it can be organized as a collection of a finite number of memory words or registers. Figure shows the organization of a 64-word register stack.

- The stack pointer register SP contains a binary number whose value is equal to the address of the word that is currently on top of the stack. Three items are placed in thestack: A, B, and C, in that order. Item C is on top of the stack so that the content of SP is now 3.
- To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP. Item B is now on top of the stack since SP holds address 2.
- To insert a new item, the stack is pushed by incrementing SP and writing a word in the next-higher location in the stack.
- In a 64-word stack, the stack pointer contains 6 bits because 26 = 64.
- Since SP has only six bits, it cannot exceed a number greater than 63 (111111 in binary).
- When 63 are incremented by 1, the result is 0 since 111111 + 1 = 1000000 in binary, but
- SP can accommodate only the six least significant bits.
- Similarly, when 000000 is decremented by 1, the result is 111111. The one-bit register
- FULL is set to 1 when the stack is full, and the one-bit register EMTY is set to 1 when the stack is empty of items.
- DR is the data register that holds the binary data to be written into or read out of the stack.

PUSH:

- If the stack is not full (FULL =0), a new item is inserted with a push operation. The push operation consists of the following sequences of microoperations:

  SP ← SP + 1 Increment stack pointer

  M [SP] ← DR WRITE ITEM ON TOP OF THE STACK

  IF (SP = 0) then (FULL ← 1) Check is stack is full

  EMTY ← 0 Mark the stack not empty
- The stack pointer is incremented so that it points to the address of next-higher word. A memory write operation inserts the word from DR into the top of the stack.
- SP holds the address of the top of the stack and that M[SP] denotes the memory word specified by the address presently available in SP.
- The first item stored in the stack is at address 1. The last item is stored at address 0. If SP reaches 0, the stack is full of items, so FULL is set to 1. This condition is reached if the top item prior to the last push was in location 63 and, after incrementing SP, the last item is stored in location 0.
- Once an item is stored in location 0, there are no more empty registers in the stack. If an item is written in the stack, obviously the stack cannot be empty, so EMTY is cleared to 0.

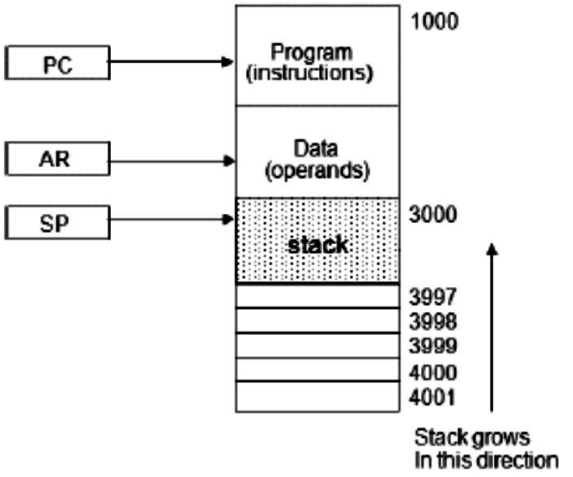
POP:

- A new item is deleted from the stack if the stack is not empty (if EMTY = 0). The pop operation consists of the following sequences of microoperations:
  - DR ← M [SP] Read item on top of the stack
  - SP ← SP – 1 Decrement stack pointer
  - IF (SP = 0) then (EMTY ← 1) Check if stack is empty
  - FULL ← 0 Mark the stack not full

- The top item is read from the stack into DR. The stack pointer is then decremented. If its value reaches zero, the stack is empty, so EMTY is set to 1.
- This condition is reached if the item read was in location 1.
- Once this item is read out, SP is decremented and reaches the value 0, which is the initial value of SP. If a pop operation reads the item from location 0 and then SP is decremented, SP is changes to 111111, which is equivalent to decimal 63.
- In this configuration, the word in address 0 receives the last item in the stack. Note also that an erroneous operation will result if the stack is pushed when FULL = 1 or popped when EMTY = 1. 10. Explain Memory Stack.

Q11. Explain briefly about Memory Stack.

Ans :



ter memory with program, data, and stack segments

- The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a stack pointer.
- Figure shows a portion of computer memory partitioned into three segments: program, data, and stack.
- The program counter PC points at the address of the next instruction in the program which is used during the fetch phase to read an instruction.
- The address registers AR points at an array of data which is used during the execute phase to read an operand.
- The stack pointer SP points at the top of the stack which is used to push or pop items into or from the stack.
- The three registers are connected to a common address bus, and either one can provide an address for memory.
- As shown in Figure 5.2, the initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at address 3999, and the last address that can be used for the stack is 3000.
- We assume that the items in the stack communicate with a data register DR.

PUSH

- A new item is inserted with the push operation as follows:
  - SP ← SP - 1
  - M[SP] ← DR
- The stack pointer is decremented so that it points at the address of the next word.
- A memory write operation inserts the word from DR into the top of the stack.

POP

- A new item is deleted with a pop operation as follows:
  - DR ← M[SP]
  - SP ← SP + 1
- The top item is read from the stack into DR.
- The stack pointer is then incremented to point at the next item in the stack.
- The two microoperations needed for either the push or pop are (1) an access to memory through SP, and (2) updating SP.
- Which of the two microoperations is done first and whether SP is updated by incrementing or decrementing depends on the organization of the stack.
- In figure. 5.2 the stack grows by decreasing the memory address. The stack may be constructed to grow by increasing the memory also.
- The advantage of a memory stack is that the CPU can refer to it without having to specify an address, since the address is always available and automatically updated in the stack pointer.

3.2.3 Instruction Formats

Q12. Explain four types of instruction formats.

Ans :

Three Address Instructions:

Computers with three-address instruction formats can use each address field to specify either a processor register or a memory operand. The program in assembly language that evaluates X = (A + B) * (C + D) is shown below.

```
ADD R1, A, B R1      M [A] + M [B]
ADD R2, C, D R2      M[C] + M [D]
MUL X, R1, R2 M[X]   R1 * R2
```

- The advantage of three-address format is that it results in short programs when evaluating arithmetic expressions.
- The disadvantage is that the binary-coded instructions require too many bits to specify three addresses.
- An example of a commercial computer that uses three-address instruction is the Cyber 170.

Two Address Instructions:

Two address instructions are the most common in commercial computers. Here again each address field can specify either a processor register or a memory word. The program to evaluate X = (A + B) * (C + D) is as follows:

MOV R1, A R1 ⟶ M [A]

ADD R1, B R1 ⟶ R1 + M [B]

MOV R2, C R2 ⟶ M [C]

ADD R2, D R2 ⟶ R2 + M [D]

MUL R1, R2 R1 ⟶ R1 * R2

MOV X, R1 M [X] ⟶ R1

- The MOV instruction moves or transfers the operands to and from memory and processor registers. The first symbol listed in an instruction is assumed to be both a source and the destination where the result of the operation is transferred.

One Address Instructions:

- One address instructions use an implied accumulator (AC) register for all data manipulation. For multiplication and division these is a need for a second register.
- However, here we will neglect the second register and assume that the AC contains the result of all operations. The program to evaluate X = (A + B) * (C + D) is

  LOAD A AC ⟶ M [A]

  ADD B AC ⟶ AC + M [B]

  STORE T M [T] ⟶ AC

  LOAD C AC ⟶ M [C]

  ADD D AC ⟶ AC + M [D]

  MUL T AC ⟶ AC * M [T]

  STORE X M [X] ⟶ AC

All the operations are done between the AC register and a memory operand. T is the address of the temporary memory location required for storing the intermediate result.

Zero Address Instructions:

- A stack-organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.
- The program to evaluate X = (A + B) * (C + D) will be written for a stack-organized computer.
  PUSH A TOS ⟶ A

  PUSH B TOS ⟶ B

  ADD TOS ⟶ A + B)

  PUSH C TOS ⟶ B

  PUSH D TOS ⟶ D

ADD TOS ⟶ C + D)

MUL TOS ⟶ C + D) * (A + B)

POP X M [X] ⟶ TOS

- To evaluate arithmetic expressions in a stack computer, it is necessary to convert the expression into reverse polish notation.

RISC Instructions:

- All other instructions are executed within the registers of the CPU without referring to memory. A program for a RISC type CPU consists of LOAD and STORE instructions that have one memory and one register address, and computational-type instructions that have three addresses with all three specifying processor registers.
- The following is a program to evaluate X = (A + B) * (C + D).

LOAD R1, A R1 ⟶ M [A]

LOAD R1, B R1 ⟶ M [B]

LOAD R1, C R1 ⟶ M [C]

LOAD R1, D R1 ⟶ M [D]

ADD R1, R1, R2 R1 ⟶ R1 + R2

ADD R3, R3, R2 R3 ⟶ R3 + R4

MUL R1, R1, R3 R1 ⟶ R1 * R3

STORE X, R1 M [X] ⟶ R1

- The load instructions transfer the operands from memory to CPU register.
- Add and multiply operations are executed with data in the registers without accessing memory.
- The result of the computations is then stored in memory with a store instruction.

3.2.4 Addressing Modes

Q13. Write a note on different Addressing Modes.

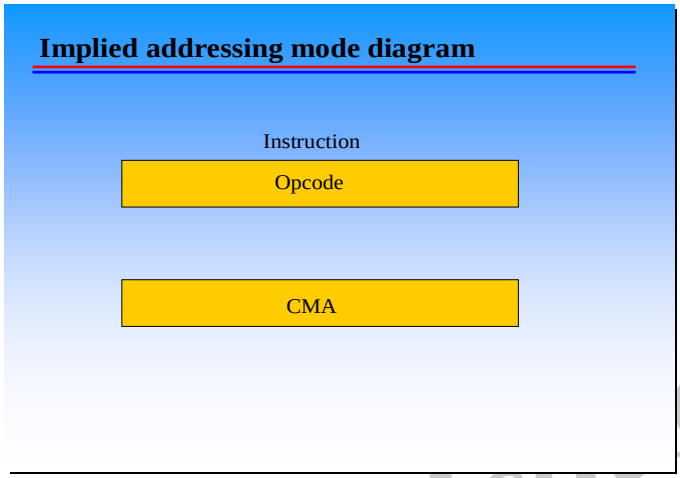
Ans :

The general addressing modes supported by the computer processor are as follows:

- Implied addressing mode
- Immediate addressing mode
- Direct addressing mode
- Indirect addressing mode
- Register addressing mode
- Register Indirect addressing mode
- Autoincrement or Autodecrement addressing mode
- Relative addressing mode
- Indexed addressing mode
- Base register addressing mode

➢ Implied addressing mode

In this mode the operands are specified implicitly in the definition of the instruction. For example the 'complement accumulator' instruction is an implied mode instruction because the operand in the accumulator register is implied in the definition of the instruction itself. All register reference instructions that use an accumulator are implied mode instructions. Zero address instructions in a stack organized computer are implied mode instructions since the operands are implied to be on the top of the stack.



➢ Immediate Addressing Mode

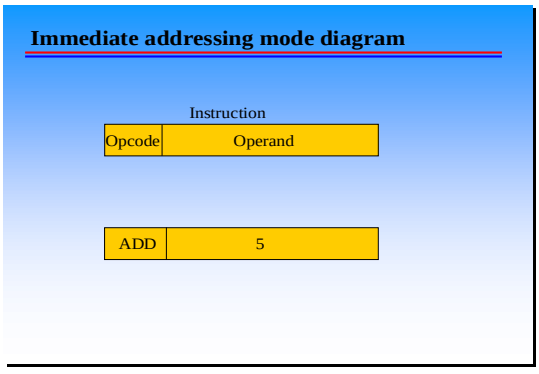
In this mode the operand is specified in the instruction itself. In other words, an immediate mode instruction has a operand field rather than an address field. The operand field contains the actual operand to be used in conjunction with the operation specified in the instruction. Immediate mode instructions are useful for initializing registers to a constant value.

Example ADD 5

- Add 5 to contents of accumulator
- 5 is operand

Advantages and Disadvantages

- No memory reference to fetch data
- Fast
- Limited range



- **Direct Addressing Mode**

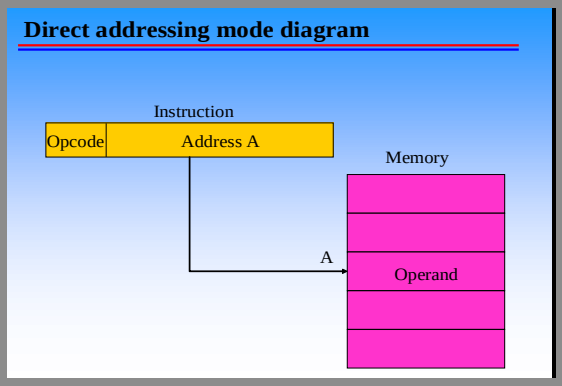
In this mode the effective address is equal to the address part of the instruction. The operand resides in memory and its address is given directly by the address field of instruction. In a branch type instruction the address field specifies the actual branch address.

Effective address (EA) = address field (A)

e.g. LDA A

- Look in memory at address A for operand which is to be loaded in the accumulator.
- Load contents of cell A to accumulator

- **Advantages and Disadvantages**
  - Single memory reference to access data
  - No additional calculations to work out effective address
  - Limited address space



- **Indirect Addressing Mode**

In this mode the address field of the instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

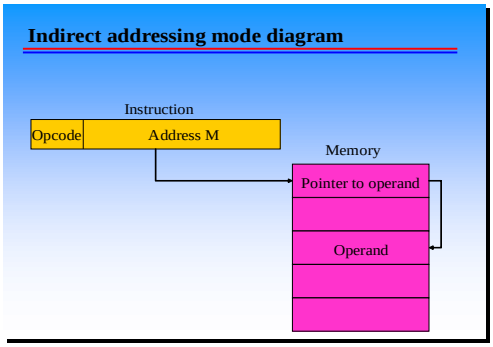
EA = address contained in memory location M

- Look in M, find address contained in M and look there for operand

For example

ADD @M

- Add contents of memory location pointed to by contents of M to accumulator



➢ **Register Addressing Mode**

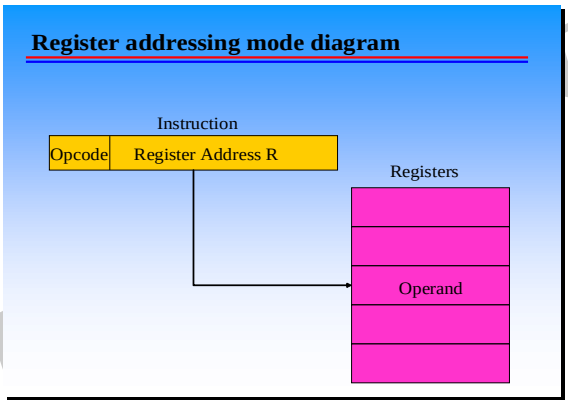
In this mode the operands are in the registers that reside within the CPU.

EA = R

Example ADD B

Advantages and Disadvantages

- ➢ No memory access. So very fast execution.
- ➢ Very small address field needed.
  - Shorter instructions
  - Faster instruction fetch
- ➢ Limited number of registers.
- ➢ Multiple registers helps performance.
  - Requires good assembly programming or compiler writing



➢ **Register Indirect Addressing Mode**

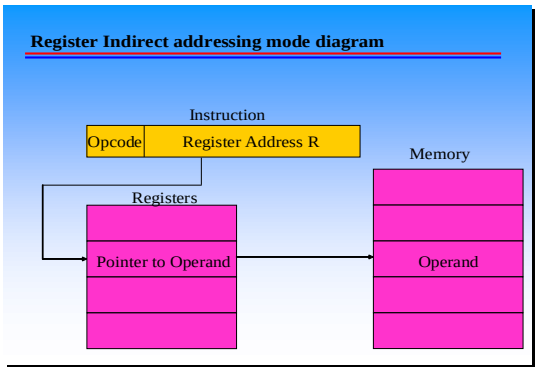
In this mode the instruction specifies a register in the CPU whose contents give the address of the operand in the memory. In other words, the selected register contains the address of the operand rather than the operand itself. Before using a register indirect mode instruction, the programmer must ensure that the memory address of the operand is placed in the processor register with a previous instruction. The advantage of a register indirect mode instruction is that the address field of the instruction uses fewer bits to select a register than would have been required to specify a memory address directly.

Therefore EA = the address stored in the register R

- ➢ Operand is in memory cell pointed to by contents of register R
- ➢ Example LDAX B

Advantage

- ➢ Less number of bits are required to specify the register.
- ➢ One fewer memory access than indirect addressing.



➢ **Autoincrement or Autodecrement Addressing Mode**

This is similar to the register indirect mode except that the register is incremented or decremented after or before its value is used to access memory. When the address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using the increment or decrement instruction.

However, because it is such a common requirement that the computers have incorporated special mode that automatically increments or decrements the contents of the register after data access.

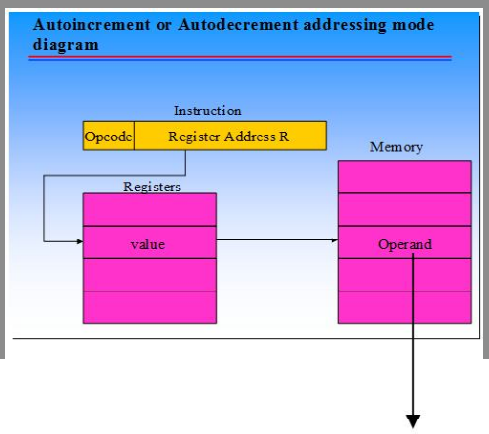
- This mode is specially useful when we want to access a table of data.
- For example

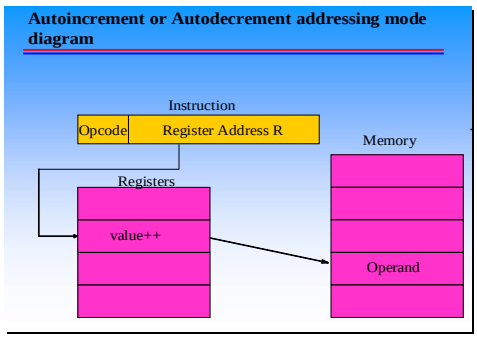
  INR R1

  Will increment the register R1.

  DCR R2

  Will decrement the register R2.





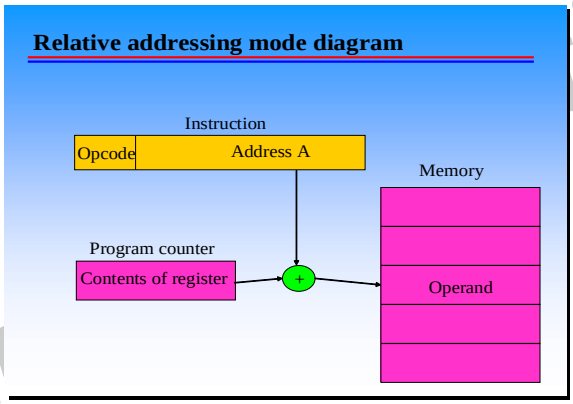
➢ Relative addressing mode

In this mode the content of the program counter is added to the address part of the instruction in order to obtain the effective address. Effective address is defined as the memory address obtained from the computation dictated by the given addressing mode. The address part of the instruction is usually a signed number (in 2's complement representation) which can be either positive or negative. When this number is added to the content of the program counter, the result produces an effective address whose position in memory is relative to the address of the next instruction. Relative addressing is often used with branch type instructions when the branch address is in the area surrounding the instruction word itself. It results in a shorter address field in the instruction format since the relative address can be specified with a smaller number of bits compared to the bits required to designate the entire memory address.

EA = A + contents of PC

Example: PC contains 825 and address part of instruction contains 24.

After the instruction is read from location 825, the PC is incremented to 826. So EA = 826 + 24 = 850. The operand will be found at location 850 i.e. 24 memory locations forward from the address of the next instruction.



➢ Indexed Addressing Mode

In this mode the content of an index register is added to the address part of the instruction to obtain the effective address. The index register is a special CPU register that contains an index value. The address field of the instruction defines the beginning address of a data array in memory. Each operand in the array is store din memory relative to the beginning address. The distance between the beginning address and the address of the operand is the index value stored in the index register. Any operand in the array can be accessed with the same instruction provided that the index register contains the correct index value. The index register can be incremented to facilitate access to consecutive operands. Note that if an index type instruction does not include an address field in its format, then the instruction converts to the register indirect mode of operation.

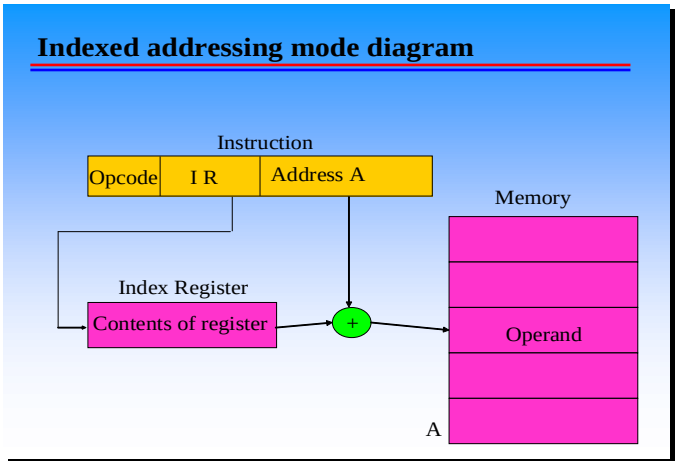
- Therefore

  EA = A + IR

- Example MOV AL , DS: disp [SI]

Advantage

- Good for accessing arrays.



➢ **Base Register Addressing Mode**

In this mode the content of base register is added to the address part of the instruction to obtain the effective address. This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register. The difference between the two modes is in the way they are used rather than in the way that they are computed. An index register is assumed to hold an index number that is relative to the address part of the instruction. A base register is assumed to hold a base address and the address field of the instruction gives a displacement relative to this base address. The base register addressing mode is used in computers to facilitate the relocation of the programs in memory. When programs and data are moved from one segment of memory to another, as required in multiprogramming systems, the address values of instructions must reflect this change of position. With a base register, the displacement values of instructions do not have to change. Only the value of the base register requires updating to reflect the beginning of a new memory segment.

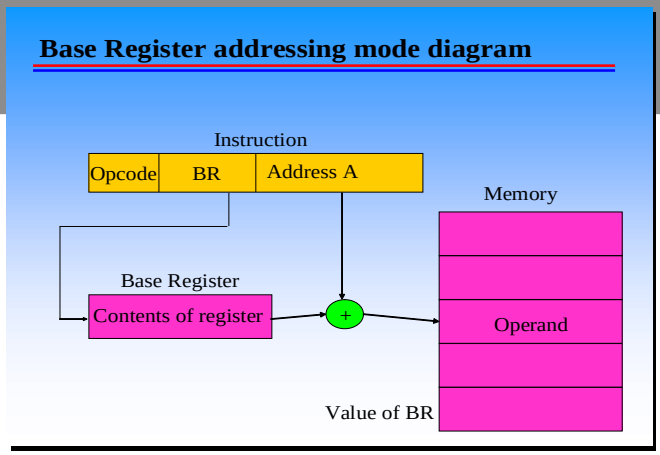
➢ Therefore

EA = A + BR

➢ For example:

MOV AL, disp [BX]

Segment registers in 8086



3.2.5 Data Transfer and Manipulation

Q14. Explain Data Transfer Instructions.

Ans :

Typical data transfer instructions are as follows:

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

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.
- The **load** instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator.
- The **store** instruction designates a transfer from a processor register into memory.
- The **move** instruction has been used in computers with multiple CPU registers to designate a transfer from one register to another. It has also been used for data transfers between CPU registers and memory or between two memory words.
- The **exchange** instruction swaps information between two registers or a register and a memory word.
- The **input and output** instructions transfer data among processor registers and input or output terminals.
- The **push and pop** instructions transfer data between processor registers and a memory stack.

Q15. Explain about data manipulation instructions.

Ans :

Data Manipulation Instruction

It performs operations on data and provides the computational capabilities for the computer. The data manipulation instructions in a typical computer are usually divided into three basic types.

(a) Arithmetic Instruction

(b) Logical bit manipulation Instruction

(c) Shift Instruction.

(a) Arithmetic Instruction

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	Add
Subtract	Sub
Multiply	MUL
Divide	DIV
Add with Carry	ADDC
Subtract with Basses	SUBB
Negate (2's Complement)	NEG

(b) Logical & Bit Manipulation Instruction

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-Or	XOR
Clear Carry	CLRC
Set Carry	SETC
Complement Carry	COMC
Enable Interrupt	ET
Disable Interrupt	OI

(c) Shift Instruction

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

Name	Mnemonic
Logical Shift right	SHR
Logical Shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate mgmt through carry	RORC
Rotate left through carry	ROLC

3.2.6 Program Control

Q16. What are status register bits? Draw and explain the block diagram showing all status registers.

Ans : **(Imp.)**

- It is sometimes convenient to supplement the ALU circuit in the CPU with a status register where status bit conditions be stored for further analysis. Status bits are also called condition-code bits or flag bits.
- Figure shows the block diagram of an 8-bit ALU with a 4-bit status register. The four status bits are symbolized by C, S, Z, and V. The bits are set or cleared as a result of an operation performed in the ALU.

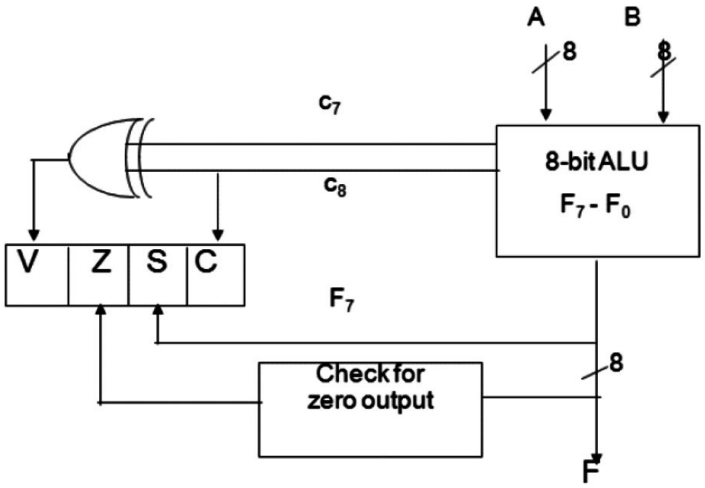


Figure 5.3: Status Register Bits

1. Bit C (carry) is set to 1 if the end carry C8 is 1. It is cleared to 0 if the carry is 0.
2. Bit S (sign) is set to 1 if the highest-order bit F7 is 1. It is set to 0 if set to 0 if the bit is 0.
3. Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. it is cleared to 0 otherwise. In other words, Z = 1 if the output is zero and Z = 0 if the output is not zero.
4. Bit V (overflow) is set to 1 if the exclusives-OR of the last two carries is equal to 1, and cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement. For the 8-bit ALU, V = 1 if the output is greater than + 127 or less than -128.

   - The status bits can be checked after an ALU operation to determine certain relationships that exist between the vales of A and B.
   - If bit V is set after the addition of two signed numbers, it indicates an overflow condition.
   - If Z is set after an exclusive-OR operation, it indicates that A = B.
   - A single bit in A can be checked to determine if it is 0 or 1 by masking all bits except the bit in question and then checking the Z status bit.

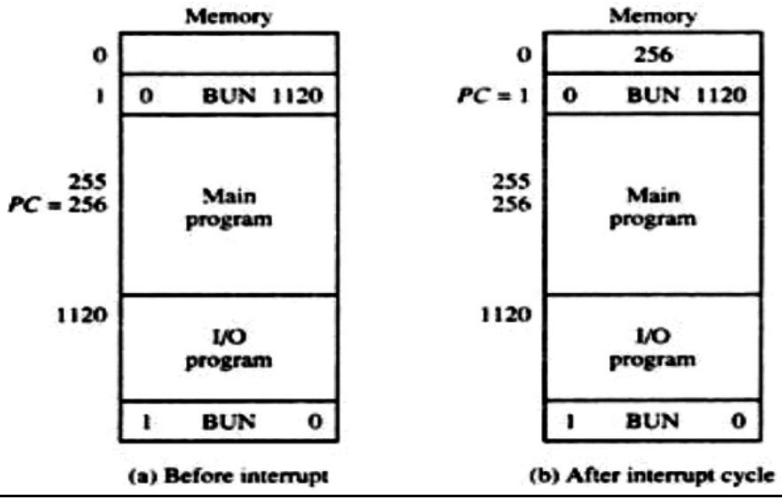
Q17. What is program interrupt? What happens when it Comes? What are the tasks to be performed by service routine?

OR

Explain Program Interrupts. Explain clearly, discussing the role of stack, PSW and return from interrupt instruction, how interrupts are implemented on computers.

Ans :

- The concept of program interrupt is used to handle a variety of problems that arise out of normal program sequence.
- Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request.
- Control returns to the original program after the service program is executed.
- After a program has been interrupted and the service routine been executed, the CPU must return to exactly the same state that it was when the interrupt occurred.
- Only if this happens will the interrupted program be able to resume exactly as if nothing had happened.
- The state of the CPU at the end of the execute cycle (when the interrupt is recognized) is determined from:
    1. The content of the program counter
    2. The content of all processor registers
    3. The content of certain status conditions
- The interrupt facility allows the running program to proceed until the input or output device sets its ready flag. Whenever a flag is set to 1, the computer completes the execution of the instruction in progress and then acknowledges the interrupt.
- The result of this action is that the retune address is stared in location 0. The instruction in location 1 is then performed; this initiates a service routine for the input or output transfer.
- The service routine can be stored in location 1.
- The service routine must have instructions to perform the following tasks:
    1. Save contents of processor registers.
    2. Check which flag is set.
    3. Service the device whose flag is set.
    4. Restore contents of processor registers.
    5. Turn the interrupt facility on.
    6. Return to the running program.



Q18. Explain various types of interrupts.

Ans :

There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

1. External interrupts
2. Internal interrupts
3. Software interrupts

1. External interrupts

- External interrupts come from input-output (I/0) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source.
- Examples that cause external interrupts are I/0 device requesting transfer of data, I/o device finished transfer of data, elapsed time of an event, or power failure. Timeout interrupt may result from a program that is in an endless loop and thus exceeded its time allocation.
- Power failure interrupt may have as its service routine a program that transfers the
- Complete state of the CPU into a nondestructive memory in the few milliseconds before power ceases.
- External interrupts are asynchronous. External interrupts depend on external conditions that are independent of the program being executed at the time.

2. Internal Interrupts

- Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps.
- Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation. These error conditions usually occur as a result of a premature termination of the instruction execution. The service program that processes the internal interrupt determines the corrective measure to be taken.
- Internal interrupts are synchronous with the program. . If the program is rerun, the internal interrupts will occur in the same place each time.

3. Software Interrupts

- A software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program.
- The most common use of software interrupt is associated with a supervisor call instruction.
- This instruction provides means for switching from a CPU user mode to the supervisor mode.
- Certain operations in the computer may be assigned to the supervisor mode only, as for example, a complex input or output transfer procedure. A program written by a user must run in the user mode.
- When an input or output transfer is required, the supervisor mode is requested by means of a supervisor call instruction. This instruction causes a software interrupt that stores the old CPU state and brings in a new PSW that belongs to the supervisor mode.
- The calling program must pass information to the operating system in order to specify the particular task requested.

3.3 COMPUTER ARITHMETIC

3.3.1 Addition and Subtraction

Q19. Explain the procedure for Addition and Subtraction with signed-magnitude data with the help of flowchart.

Ans :

- The two signs A, and B, are compared by an exclusive-OR gate.
  - o If the output of the gate is 0 the signs are identical;
  - o If it is 1, the signs are different.
- For an add operation, identical signs dictate that the magnitudes be added. For a subtract operation, different signs dictate that the magnitudes be added.
- The magnitudes are added with a microoperation EA A + B, where EA is a register that combines E and A. The carry in E after the addition constitutes an overflow if it is equal to 1. The value of E is transferred into the add-overflow flip-flop AVF.
- The two magnitudes are subtracted if the signs are different for an add operation or identical for a subtract operation. The magnitudes are subtracted by adding A to the 2's complemented B. No overflow can occur if the numbers are subtracted so AVF is cleared to 0.
- 1 in E indicates that A > = B and the number in A is the correct result. If this numbs is zero, the sign A must be made positive to avoid a negative zero.
- 0 in E indicates that A < B. For this case it is necessary to take the 2's complement of the value in A. The operation can be done with one microoperation A A' +1.
- However, we assume that the A register has circuits for microoperations complement and increment, so the 2's complement is obtained from these two microoperations.
- In other paths of the flowchart, the sign of the result is the same as the sign of A. so no change in A is required. However, when A < B, the sign of the result is the complement of the original sign of A. It is then necessary to complement A, to obtain the correct sign.

- The final result is found in register A and its sign in As. The value in AVF provides an overflow indication. The final value of E is immaterial.
- Figure 7.2 shows a block diagram of the hardware for implementing the addition and subtraction operations.
- It consists of registers A and B and sign flip-flops As and Bs.
- Subtraction is done by adding A to the 2's complement of B.
- The output carry is transferred to flip-flop E , where it can be checked to determine the relative magnitudes of two numbers.
- The add-overflow flip-flop *AVF* holds the overflow bit when A and B are added.
- The A register provides other microoperations that may be needed when we specify the sequence of steps in the algorithm.

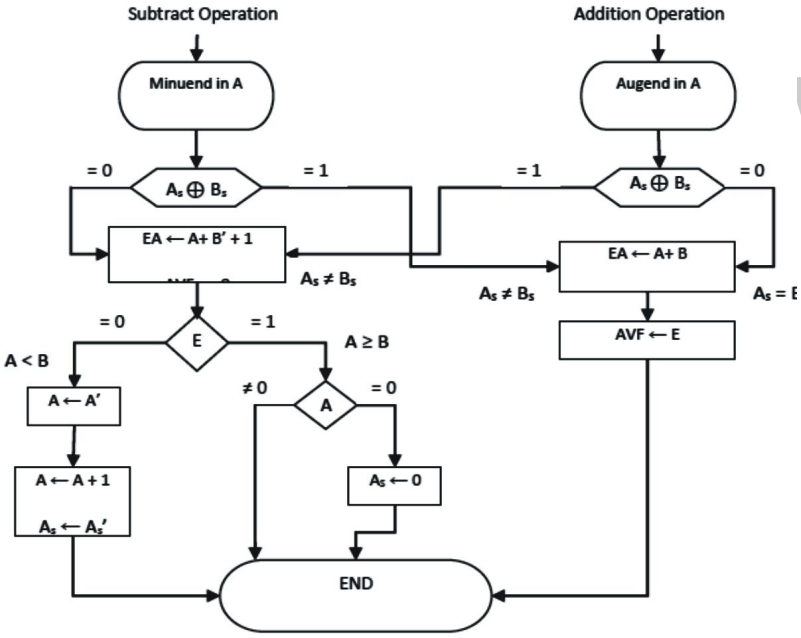


Figure. Flowchart for add and subtract operations

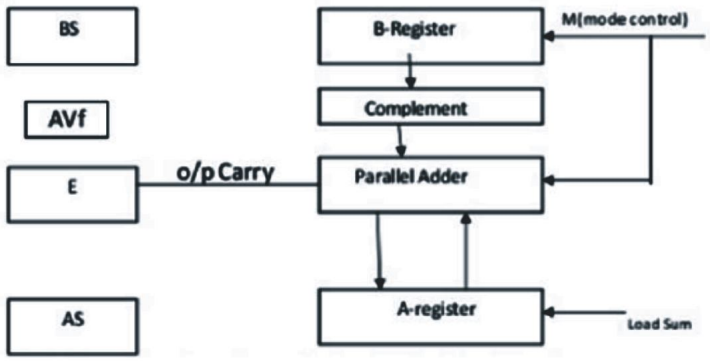


Figure. Hardware for signed -magnitude addition and subtraction

3.3.2 Multiplication

Q20. Explain the Booth's algorithm with the help of flowchart.

Ans :

- Booth algorithm gives a procedure for multiplying binary integers in signed - 2's complement representation.
- It operates on the fact that strings of 0's in the multiplier require no addition but just shifting, and a string of 1's in the multiplier from bit weight 2k to weight 2m can be treated as 2k+1 – 2m.
- For example, the binary number 001110 (+14) has a string 1's from 23 to 21 (k=3, m=1).
- The number can be represented as 2k+1 – 2m. = 24 – 21 = 16 – 2 = 14. Therefore, the multiplication M X 14, where M is the multiplicand and 14 the multiplier, can be done as M X 24 – M X 21.
- Thus the product can be obtained by shifting the binary multiplicand M four times to the left and subtracting M shifted left once.

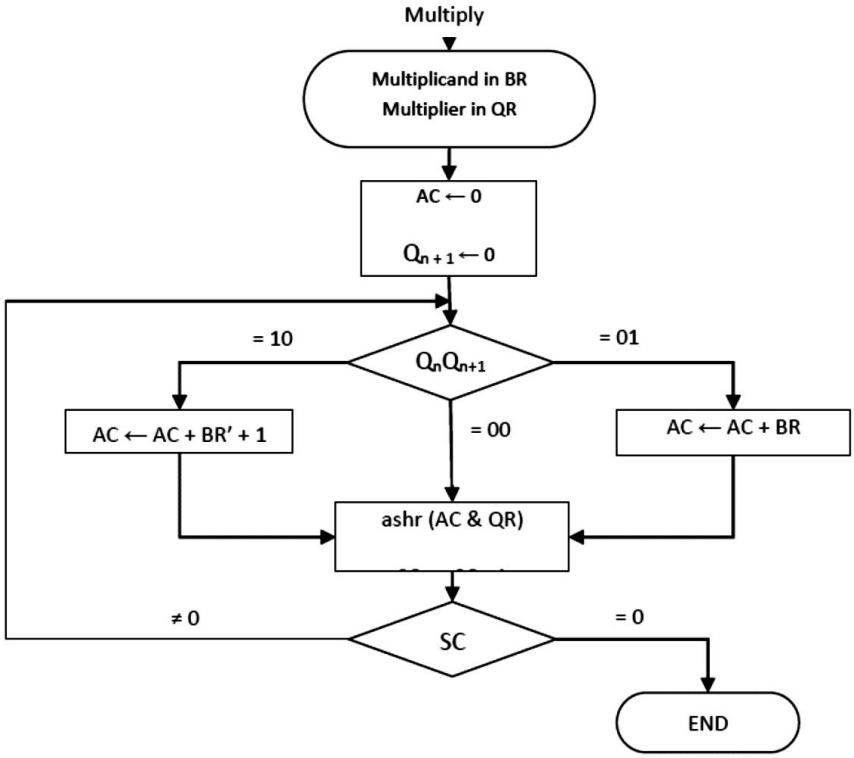


Fig. Booth algorithm for multiplication of signed-2's complement numbers

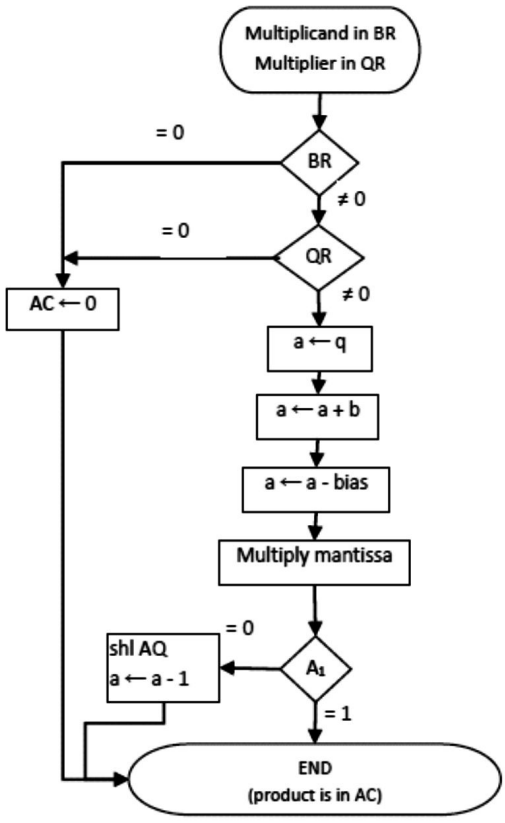
- As in all multiplication schemes, booth algorithm requires examination of the multiplier bits and shifting of partial product.

Q21. Explain with proper block diagram the Multiplication Operation on two floating point numbers.

Ans :

- The multiplication of two floating-point numbers requires that we multiply the mantissas and add the exponents.

- No comparison of exponents or alignment of mantissas is necessary.
- The multiplication of the mantissas is performed in the same way as in fixed-point to provide a double-precision product.
- The double-precision answer is used in fixed-point numbers to increase the accuracy of the product.
- In floating-point, the range of a single-precision mantissa combined with the exponent is usually accurate enough so that only single-precision numbers are maintained.
- Thus the half most significant bits of the mantissa product and the exponent will be taken together to form a single-precision floating-point product.
- The multiplication algorithm can be subdivided into four parts:
  1. Check for zeros.
  2. Add the exponents.
  3. Multiply the mantissas.
  4. Normalize the product.
- The flowchart for floating-point multiplication is shown in Figure . The two operands are checked to determine if they contain a zero.
- If either operand is equal to zero, the product in the AC is set to zero and the operation is terminated.
- If neither of the operands is equal to zero, the process continues with the exponent addition.
- The exponent of the multiplier is in q and the adder is between exponents a and b.
- It is necessary to transfer the exponents from q to a, add the two exponents, and transfer the sum into a.



Q22. Multiply the (-9) with (-13) using Booth's algorithm. Give each step.

Ans :

A numerical example of booth algorithm is shown for n = 5. It shows the step-by-step multiplication of (-9) X (-13) = +117.

9:	01001		13:	01101	
1's complement of 9:	10110		1's complement of 13:	10010	
	+ 1			+ 1	
2's complement of 9:	10111	(-9)	2's complement of 13:	10011	(-13)

AC	QR(-13)	Q_{n+1}	M(BR)(-9)	SC	Comments
00000	10011	0	10111	5	Initial value
01001	10011	0	10111	4	Subtraction: AC=AC+BR'+1
00100	11001	1	10111		Arithmetic Shift Right
00010	01100	1	10111	3	Arithmetic Shift Right
11001	01100	1	10111	2	Subtraction: AC=AC+BR'+1
11100	10110	0	10111		Arithmetic Shift Right
11110	01011	0	10111	1	Arithmetic Shift Right
00111	01011	0	10111	0	Subtraction: AC=AC+BR'+1
00011	**10101**	1	10111		Arithmetic Shift Right

Answer: –9 X –13 = 117 = > 001110101

Q23. Multiply the (7) with (3) using Booth's algorithm. Give each step.

Ans :

7: 0111 | 3: 0011

AC	QR(3)	Q_{n+1}	M(BR)(7)	SC	Comments
0000	0011	0	0111	4	Initial value
1001	0011	0	0111	3	Subtraction: AC=AC+BR'+1
1100	1001	1	0111		Arithmetic Shift Right
1110	0100	1	0111	2	Arithmetic Shift Right
0101	0100	1	0111	1	Addition: AC=AC+BR
0010	1010	0	0111		Arithmetic Shift Right
0001	**0101**	0	0111	0	Arithmetic Shift Right

Answer: 7 X 3 = 21 = > 00010101

Q24. Multiply the (15) with (13) using Booth's algorithm. Give each step.

Ans :

15: 01111 | 13: 01101

15X13=195

AC	QR(15)	Q_{n+1}	M(BR)(13)	SC	Comments
00000	01111	0	01101	5	Initial value
10011	01111	0	01101	4	Subtraction: AC=AC+BR'+1
11001	10111	1	01101		Arithmetic Shift Right
11100	11011	1	01101	3	Arithmetic Shift Right
11110	01101	1	01101	2	Arithmetic Shift Right
11111	00110	1	01101	1	Arithmetic Shift Right
01100	00110	1	01101	0	Addition: AC=AC+BR
00110	**00011**	1	01101		Arithmetic Shift Right

Answer: 15X13=195 => 0011000011

Q25. Multiply the (+15) with (-13) using Booth's algorithm. Give each step.

Ans :

15: 01111

13: 01101
1's complement of 13: 10010
+ 1
2's complement of 13: 10011 (-13)

AC	QR(-13)	Q_{n+1}	M(BR)(+15)	SC	Comments
00000	10011	0	01111	5	Initial value
10001	10011	0	01111	4	Subtraction: AC=AC+BR'+1
11000	11001	1	01111		Arithmetic Shift Right
11100	01100	1	01111	3	Arithmetic Shift Right
01011	01100	1	01111	2	Addition: AC=AC+BR
00101	10110	0	01111		Arithmetic Shift Right
00010	11011	0	01111	1	Arithmetic Shift Right
10011	11011	0	01111	0	Subtraction: AC=AC+BR'+1
11001	**11101**	1	01111		Arithmetic Shift Right

Answer: (+15) X (-13) = -195 = > **1100111101**

To verify 0011000010
+ 1
+195=> 0011000011

3.3.3 Division

Q26. Explain about division algorithm.

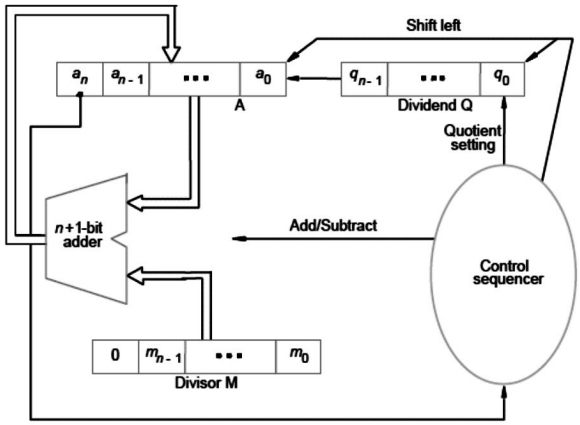
Ans :

Division Algorithm

Position the divisor appropriately with respect to the dividend and performs a subtraction.

- If the remainder is zero or positive, a quotient bit of 1 is determined, the remainder is extended by another bit of the dividend, the divisor is repositioned, and another subtraction is performed.
- If the remainder is negative, a quotient bit of 0 is determined, the dividend is restored by adding back the divisor, and the divisor is repositioned for another subtraction.

Circuit Arrangement



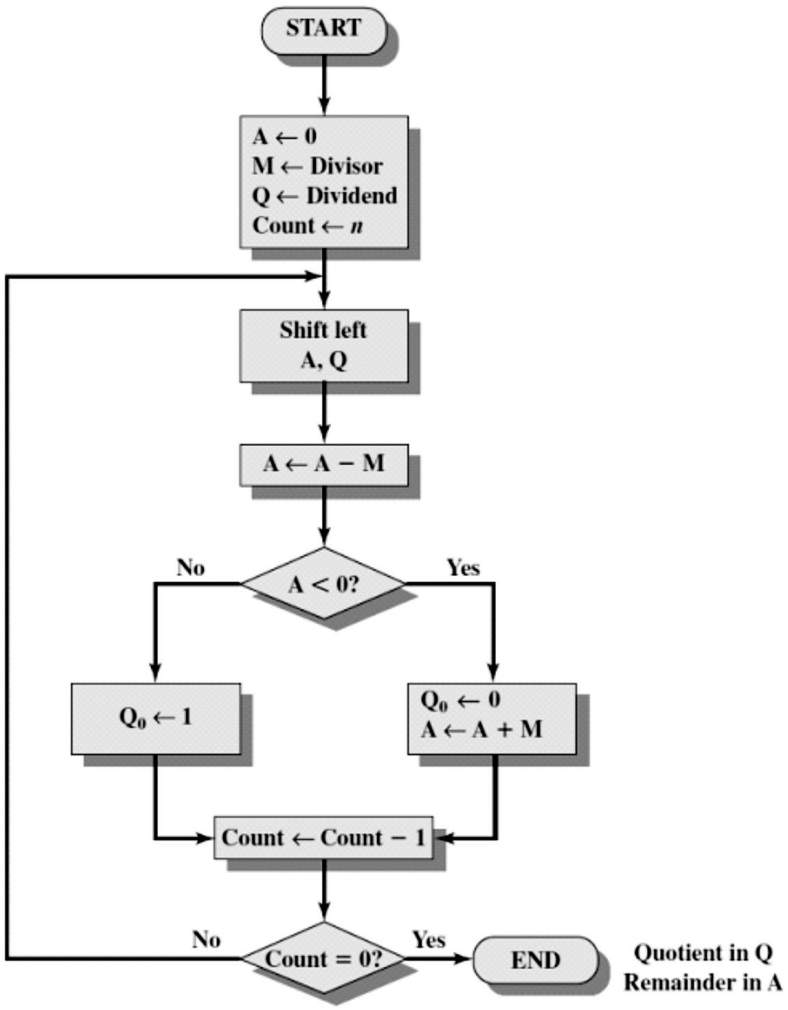
Restoring Division

- Shift A and Q left one binary position
- Subtract M from A, and place the answer back in A
- If the sign of A is 1, set q0to 0 and add M back to A (restore A); otherwise, set q0to 1
- Repeat these steps ntimes

Non restoring division

1. Load the divisor into the M register. Load the dividend into the A, Q registers. (The dividend is expressed as a 2n-bit positive number.)
2. Shift A, Q left 1 bit position.
3. Subtract M from A.
   - If the result is positive, then add 1 to Q.
   - If the result is negative, restore the previous value of A.
4. This process repeated for each bit of the original dividend.
5. The remainder is in A and the quotient is in Q.

A	Q	
0000	0111	Initial value
0000	1110	Shift
1101		Use twos complement of 0011 for subtraction
1101		Subtract
0000	1110	Restore, set $Q_0 = 0$
0001	1100	Shift
1101		
1110		Subtract
0001	1100	Restore, set $Q_0 = 0$
0011	1000	Shift
1101		
0000	1001	Subtract, set $Q_0 = 1$
0001	0010	Shift
1101		
1110		Subtract
0001	0010	Restore, set $Q_0 = 0$



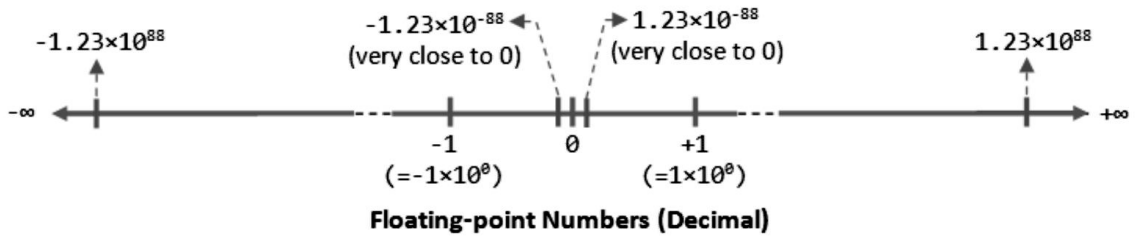
3.3.4 Floating Point Arithmetic Operations.

Q27. Write about the representation of floating point numbers

Ans :

Floating-Point Representation (Scientific Notation)

A floating-point number (or real number) can represent a very large (1.23 × 10 ^ 88) or a very small (1.23 × 10 ^ –88) value. It could also represent very large negative number (–1.23 × 10 ^ 88) and very small negative number (–1.23 × 10 ^ 88), as well as zero, as illustrated:



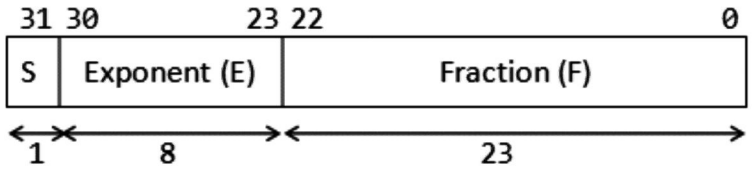
Floating-point Numbers (Decimal)

- A floating-point number is typically expressed in the scientific notation, with a fraction (F), and an exponent (E) of a certain radix (r), in the form of F×r^E. Decimal numbers use radix of 10 (F×10^E); while binary numbers use radix of 2 (F×2^E).
- Representation of floating point number is not unique. For example, the number 55.66 can be represented as 5.566×10^1, 0.5566×10^2, 0.05566×10^3, and so on. The fractional part can be normalized. In the normalized form, there is only a single non-zero digit before the radix point. For example, decimal number 123.4567 can be normalized as 1.234567×10^2; binary number 1010.1011B can be normalized as 1.0101011B×2^3.
- It is important to note that floating-point numbers suffer from loss of precision when represented with a fixed number of bits (e.g., 32-bit or 64-bit). This is because there are infinite number of real numbers (even within a small range of says 0.0 to 0.1). On the other hand, a n-bit binary pattern can represent a finite 2^n distinct numbers. Hence, not all the real numbers can be represented. The nearest approximation will be used instead, resulted in loss of accuracy.
- It is also important to note that floating number arithmetic is very much less efficient than integer arithmetic. It could be speed up with a so-called dedicated floating-point co-processor. Hence, use integers if your application does not require floating-point numbers.
- In computers, floating-point numbers are represented in scientific notation of fraction (F) and exponent (E) with a radix of 2, in the form of F×2^E. Both E and F can be positive as well as negative. Modern computers adopt IEEE 754 standard for representing floating-point numbers. There are two representation schemes: 32-bit single-precision and 64-bit double-precision.

IEEE-754 32-bit Single-Precision Floating-Point Numbers

In 32-bit single-precision floating-point representation:

- The most significant bit is the sign bit (S), with 0 for positive numbers and 1 for negative numbers.
- The following 8 bits represent exponent (E).
- The remaining 23 bits represents fraction (F).



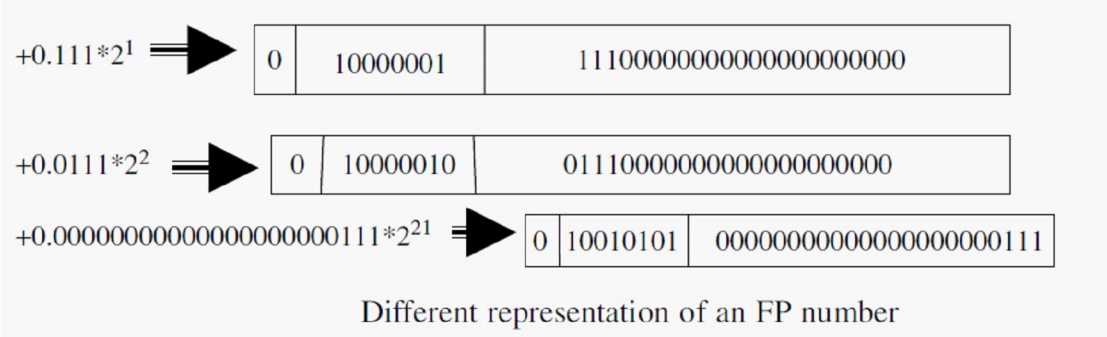
32-bit Single-Precision Floating-point Number

Q28. Explain, how to do floating point arithmetic addition and subtraction

Ans :

Floating-Point Arithmetic Addition/Subtraction

- The difficulty in adding two FP numbers stems from the fact that they may have different exponents.
- Therefore, before adding two FP numbers, their exponents must be equalized, that is, the mantissa of the number that has smaller magnitude of exponent must be aligned.



Different representation of an FP number

Steps Required to Add/Subtract Two Floating-Point Numbers

1. Compare the magnitude of the two exponents and make suitable alignment to the number with the smaller magnitude of exponent.
2. Perform the addition/subtraction.
3. Perform normalization by shifting the resulting mantissa and adjusting the resulting exponent.

Example Consider adding the two FP numbers $1.1100 * 2^4$ and $1.1000 * 2^2$.

1. **Alignment:** $1.1000 * 2^2$ has to be aligned to $0.0110 * 2^4$
2. **Addition:** Add the two numbers to get $10.0010 * 2^4$.
3. **Normalization:** The final normalized result is $0.1000 * 2^6$ (assuming 4 bits are allowed after the radix point).

Addition/subtraction of two FP numbers can be illustrated using the schematic shown in Figure.

Q29. Explain, floating point multiplication and division.

Ans :

Multiplication Multiplication of a pair of FP numbers $X = mx * 2^a$ and $Y = m_y * 2^b$ is represented as $X * Y = (mx * my) * 2^{a+b}$

A general algorithm for multiplication of FP numbers consists of three basic steps. These are:

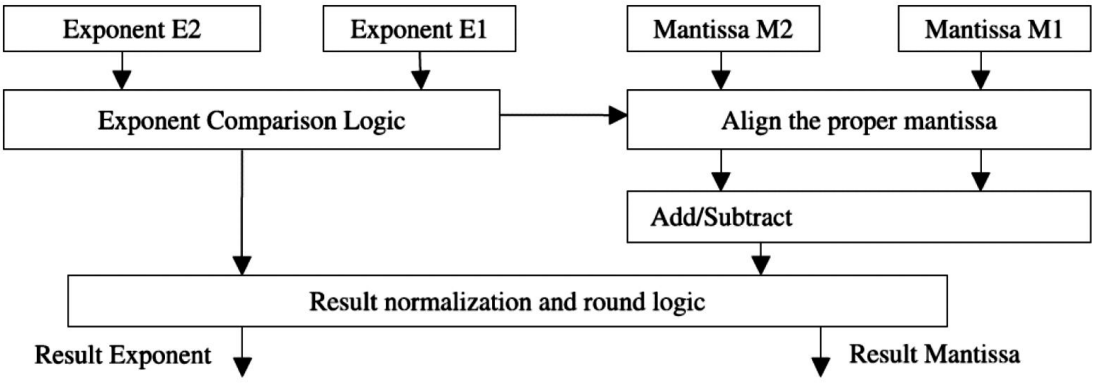
1. Compute the exponent of the product by adding the exponents together.
2. Multiply the two mantissas.
3. Normalize and round the final product.

Example Consider multiplying the two FP numbers X ¼ $1.000 * 2^{-2}$ and

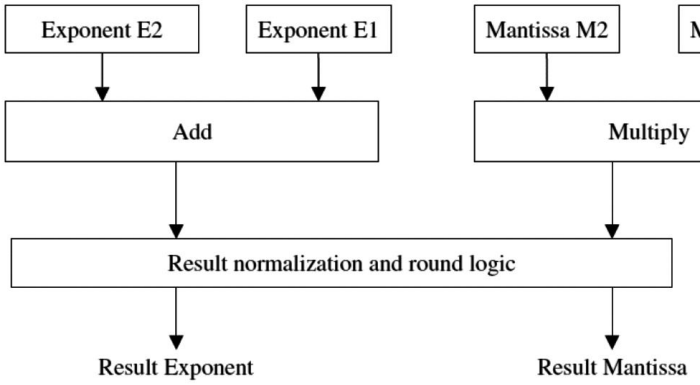
$Y = -1.010 * 2^{-1.}$

1. Add exponents: $-2 + (-1) = -3$.
2. Multiply mantissas: $1.000 * -1.010 = -1.010000$.

The product is $-1.0100 * 2^{-3}$.



Addition/subtraction of FP numbers



FP multiplication

Multiplication of two FP numbers can be illustrated using the schematic shown in Figure

Division of a pair of FP numbers X = mx * 2^a and Y = my * 2^b is represented as X/Y = (mx /my)* 2a_b.

A general algorithm for division of FP numbers consists of three basic steps:

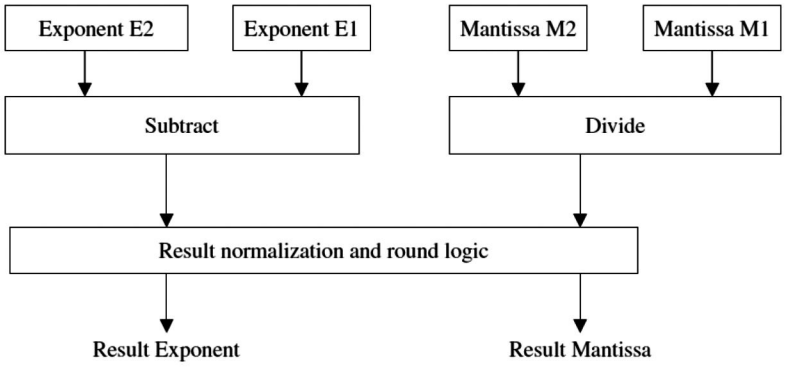
1. Compute the exponent of the result by subtracting the exponents.
2. Divide the mantissa and determine the sign of the result.
3. Normalize and round the resulting value, if necessary.

Example Consider the division of the two FP numbers X = 1.0000 * 2^{-2} and

Y = -1.0100 * 2^{-1}.

1. Subtract exponents: -2 - (-1) = -1.
2. Divide the mantissas: 1.0000 / -1.0100 = - 0.1101.
3. The result is -0.1101* 2 $^{-1}$.

Division of two FP numbers can be illustrated using the schematic shown in Figure



FP division

UNIT IV

Memory Organization

Memory Hierarchy, Main Memory, RAM and ROM, Auxiliary memory, Associative memory, Cache memory, Virtual memory, Memory Management hardware.

4.1 MEMORY ORGANIZATION

4.1.1 Memory Hierarchy

Q1. Explain briefly about memory hierarchy.

Ans :

- Memory hierarchy is the hierarchy of memory and storage devices found in a computer system.
- It ranges from the slowest but high capacity auxiliary memory to the fastest but low capacity cache memory.

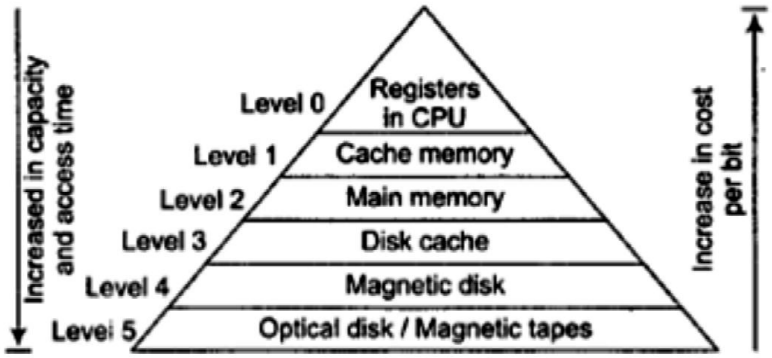
Need

There is a trade-off among the three key characteristics of memory namely-

- Cost
- Capacity
- Access time

Memory hierarchy is employed to balance this trade-off.

Memory Hierarchy Diagram-



Level-0

- At level-0, registers are present which are contained inside the CPU.
- Since they are present inside the CPU, they have least access time.
- They are most expensive and therefore smallest in size (in KB).
- Registers are implemented using Flip-Flops.

Level-1

- At level-1, Cache Memory is present.
- It stores the segments of program that are frequently accessed by the processor.
- It is expensive and therefore smaller in size (in MB).
- Cache memory is implemented using static RAM.

Level-2

- At level-2, main memory is present.
- It can communicate directly with the CPU and with auxiliary memory devices through an I/O processor.
- It is less expensive than cache memory and therefore larger in size (in few GB).
- Main memory is implemented using dynamic RAM.

Level-3

- At level-3, secondary storage devices like Magnetic Disk are present.
- They are used as back up storage.
- They are cheaper than main memory and therefore much larger in size (in few TB).

Level-4

- At level-4, tertiary storage devices like magnetic tape are present.
- They are used to store removable files.
- They are cheapest and largest in size (1-20 TB).

Observations

The following observations can be made when going down in the memory hierarchy-

- Cost / bit decreases
- Frequency of access decreases
- Capacity increases
- Access time increases

Goals of Memory Hierarchy

The goals of memory hierarchy are-

- To obtain the highest possible average access speed
- To minimize the total cost of the entire memory system

4.1.2 Main Memory

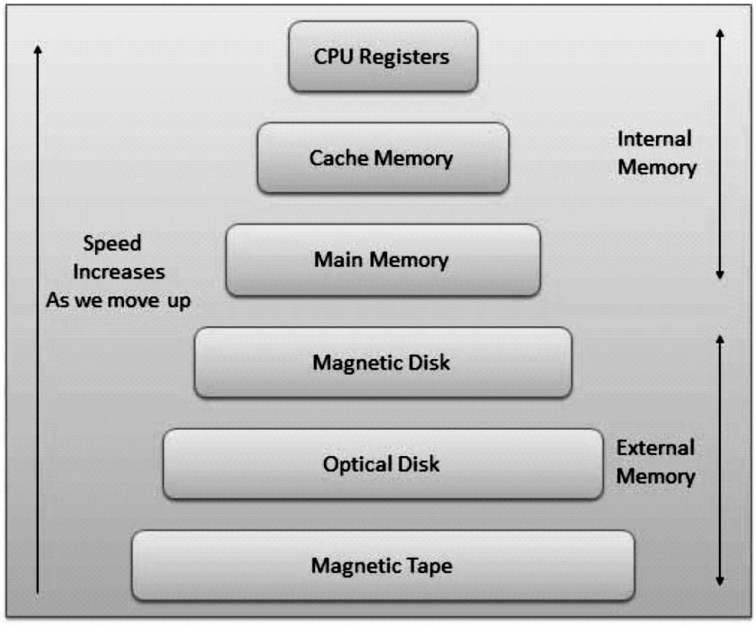
Q2. How main memory is useful in computer system?

Ans : **(Imp.)**

- A memory is just like a human brain. It is used to store data and instruction. Computer memory is the storage space in computer where data is to be processed and instructions required for processing are stored.
- The memory is divided into large number of small parts. Each part is called a cell. Each location or cell has a unique address which varies from zero to memory size minus one.
- For example if computer has 64k words, then this memory unit has 64 * 1024 = 65536 memory location. The address of these locations varies from 0 to 65535.

Memory is primarily of two types

- **Internal Memory** – cache memory and primary/main memory
- **External Memory** – magnetic disk / optical disk etc.



Characteristics of Memory Hierarchy are following when we go from top to bottom.

- Capacity in terms of storage increases.
- Cost per bit of storage decreases.
- Frequency of access of the memory by the CPU decreases.
- Access time by the CPU increases.

1. RAM

- A RAM constitutes the internal memory of the CPU for storing data, program and program result. It is read/write memory. It is called random access memory (RAM).

- Since access time in RAM is independent of the address to the word that is, each storage location inside the memory is as easy to reach as other location & takes the same amount of time. We can reach into the memory at random & extremely fast but can also be quite expensive.
- RAM is volatile, i.e. data stored in it is lost when we switch off the computer or if there is a power failure. Hence, a backup uninterruptible power system (UPS) is often used with computers. RAM is small, both in terms of its physical size and in the amount of data it can hold.
- RAM is of two types
  - Static RAM (SRAM)
  - Dynamic RAM (DRAM)

i) Static RAM (SRAM)

The word static indicates that the memory retains its contents as long as power remains applied. However, data is lost when the power gets down due to volatile nature. SRAM chips use a matrix of 6-transistors and no capacitors. Transistors do not require power to prevent leakage, so SRAM need not have to be refreshed on a regular basis.

Because of the extra space in the matrix, SRAM uses more chips than DRAM for the same amount of storage space, thus making the manufacturing costs higher.

Static RAM is used as cache memory needs to be very fast and small.

ii) Dynamic RAM (DRAM)

DRAM, unlike SRAM, must be continually refreshed in order for it to maintain the data. This is done by placing the memory on a refresh circuit that rewrites the data several hundred times per second. DRAM is used for most system memory because it is cheap and small. All DRAMs are made up of memory cells. These cells are composed of one capacitor and one transistor.

2. ROM

- ROM stands for Read Only Memory. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture.
- A ROM, stores such instruction as are required to start computer when electricity is first turned on, this operation is referred to as bootstrap. ROM chip are not only used in the computer but also in other electronic items like washing machine and microwave oven.
- Following are the various types of ROM –

i) MROM (Masked ROM)

The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs. It is inexpensive ROM.

ii) PROM (Programmable Read Only Memory)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM programmer. Inside the PROM chip there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

iii) EPROM (Erasable and Programmable Read Only Memory)

The EPROM can be erased by exposing it to ultra-violet light for a duration of upto 40 minutes. Usually, an EPROM eraser achieves this function. During programming an electrical charge is trapped in an insulated gate region. The charge is retained for more than ten years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window (lid). This exposure to ultra-violet light dissipates the charge. During normal use the quartz lid is sealed with a sticker.

iv) EEPROM (Electrically Erasable and Programmable Read Only Memory)

The EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms (millisecond). In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of re-programming is flexible but slow.

v) Serial Access Memory

Sequential access means the system must search the storage device from the beginning of the memory address until it finds the required piece of data. Memory device which supports such access is called a Sequential Access Memory or Serial Access Memory. Magnetic tape is an example of serial access memory.

vi) Direct Access Memory

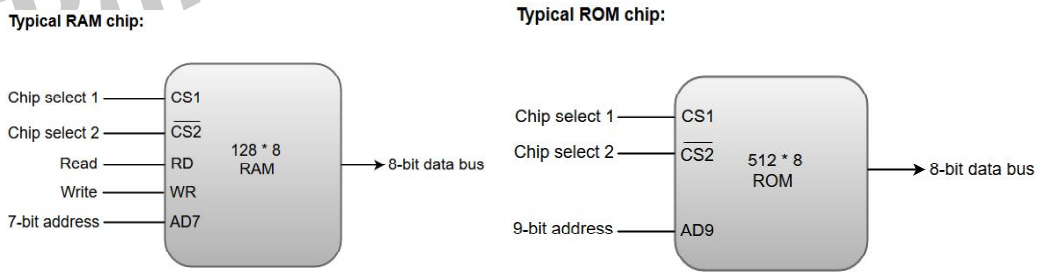
Direct access memory or Random Access Memory, refers to conditions in which a system can go directly to the information that the user wants. Memory device which supports such access is called a Direct Access Memory. Magnetic disks, optical disks are examples of direct access memory.

4.1.3 RAM and ROM

Q3. Explain the memory address map of RAM and ROM.

Ans : **(Imp.)**

Memory address map of RAM and ROM



- The designer of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM.
- The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available.
- The addressing of memory can be established by means of a table that specifies the memory address assigned to each chip.
- The table, called a **memory address map**, is a pictorial representation of assigned address space for each chip in the system, shown in table.

- To demonstrate with a particular example, assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM.
- The RAM and ROM chips to be used are specified in figure 9.1

Component	Hexa address	Address bus
		10 9 8 7 6 5 4 3 2 1
RAM 1	0000 - 007F	0 0 0 x x x x x x x
RAM 2	0080 - 00FF	0 0 1 x x x x x x x
RAM 3	0100 - 017F	0 1 0 x x x x x x x
RAM 4	0180 - 01FF	0 1 1 x x x x x x x
ROM	0200 - 03FF	1 x x x x x x x x x

Table Memory Address Map for Micro-procomputer

- The component column specifies whether a RAM or a ROM chip is used.
- The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip.
- The address bus lines are listed in the third column.
- Although there are 16 lines in the address bus, the table shows only 10 lines because the other 6 are not used in this example and are assumed to be zero.
- The small x's under the address bus lines designate those lines that must be connected to the address inputs in each chip.
- The RAM chips have 128 bytes and need seven address lines. The ROM chip has 512 bytes and needs 9 address lines.
- The x's are always assigned to the low-order bus lines: lines 1 through 7 for the RAM and lines 1 through 9 for the ROM.
- It is now necessary to distinguish between four RAM chips by assigning to each a different address. For this particular example we choose bus lines 8 and 9 to represent four distinct binary combinations.
- The table clearly shows that the nine low-order bus lines constitute a memory space for RAM equal to 29 = 512 bytes.
- The distinction between a RAM and ROM address is done with another bus line. Here we choose line 10 for this purpose.
- When line 10 is 0, the CPU selects a RAM, and when this line is equal to 1, it selects the ROM.

4.1.4 Auxiliary Memory

Q4. Explain briefly about Auxiliary memory.

Ans :

- Auxiliary memory is much larger in size than main memory but is slower. It normally stores system programs, instruction and data files.
- It is also known as secondary memory. It can also be used as an overflow/virtual memory in case the main memory capacity has been exceeded.

- Secondary memories cannot be accessed directly by a processor. First the data/information of auxiliary memory is transferred to the main memory and then that information can be accessed by the CPU. Characteristics of Auxiliary Memory are following -
  - **Non-volatile memory** - Data is not lost when power is cut off.
  - **Reusable** - The data stays in the secondary storage on permanent basis until it is not overwritten or deleted by the user.
  - **Reliable** - Data in secondary storage is safe because of high physical stability of secondary storage device.
  - **Convenience** - With the help of a computer software, authorised people can locate and access the data quickly.
  - **Capacity** - Secondary storage can store large volumes of data in sets of multiple disks.
  - **Cost** - It is much lesser expensive to store data on a tape or disk than primary memory.

Fixed and Removable Storage

i) Fixed Storage

A Fixed storage is an internal media device that is used by a computer system to store data, and usually these are referred to as the Fixed Disks drives or the Hard Drives.

- Fixed storage devices are literally not fixed, obviously these can be removed from the system for repairing work, maintenance purpose, and also for upgrade etc.
- But in general, this can't be done without a proper toolkit to open up the computer system to provide physical access, and that needs to be done by an engineer.
- Technically, almost all of the data i.e. being processed on a computer system is stored on some type of a built-in fixed storage device.

Types of fixed storage

- Internal flash memory (rare)
- SSD (solid-state disk) units
- Hard disk drives (HDD)

ii) Removable Storage

- A Removable storage is an external media device that is used by a computer system to store data, and usually these are referred to as the Removable Disks drives or the External Drives.
- Removable storage is any type of storage device that can be removed/ejected from a computer system while the system is running. Examples of external devices include CDs, DVDs and Blu-Ray disk drives, as well as diskettes and USB drives.
- Removable storage makes it easier for a user to transfer data from one computer system to another.
- In a storage factors, the main benefit of removable disks is that they can provide the fast data transfer rates associated with storage area networks (SANs)

Types of Removable Storage:

- Optical discs (CDs, DVDs, Blu-ray discs)
- Memory cards

- Floppy disks
- Magnetic tapes
- Disk packs
- Paper storage (punched tapes , punched cards)

Secondary Storage Media

There are the following main types of storage media:

1. Magnetic storage media:

Magnetic media is coated with a magnetic layer which is magnetized in clockwise or anticlockwise directions. When the disk moves, the head interprets the data stored at a specific location in binary 1s and 0s at reading.

Examples: hard disks, floppy disks and magnetic tapes.

- **Floppy Disk:** A floppy disk is a flexible disk with a magnetic coating on it. It is packaged inside a protective plastic envelope. These are one of the oldest type of portable storage devices that could store up to 1.44 MB of data but now they are not used due to very less memory storage.
- **Hard disk:** A hard disk consists of one or more circular disks called platters which are mounted on a common spindle. Each surface of a platter is coated with a magnetic material. Both surfaces of each disk are capable of storing data except the top and bottom disk where only the inner surface is used. The information is recorded on the surface of the rotating disk by magnetic read/write heads. These heads are joined to a common arm known as access arm.

2. Optical storage media

In optical storage media information is stored and read using a laser beam. The data is stored as a spiral pattern of pits and ridges denoting binary 0 and binary 1.

Examples: CDs and DVDs

4.1.5 Associative Memory

Q5. Explain briefly about Content Addressable Memory (CAM).

OR

Explain Associative memory.

Ans : **(Imp.)**

- The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address.
- A memory unit accessed by content is called an associative memory or content addressable memory (CAM).
- This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location.
- The block diagram of an associative memory is shown in figure.

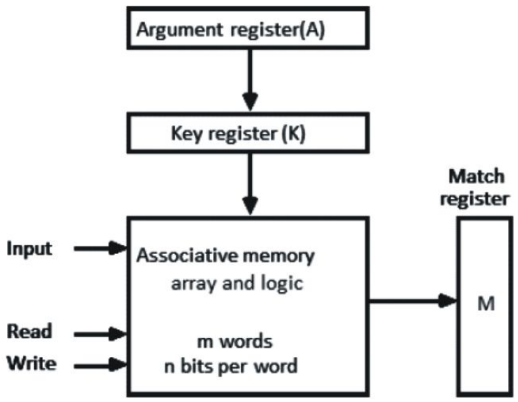


Figure. Block diagram of associative memory

- It consists of a memory array and logic form words with n bits per word.
- The argument register A and key register K each have n bits, one for each bit of a word.
- The match register M has m bits, one for each memory word.
- Each word in memory is compared in parallel with the content of the argument register.
- The words that match the bits of the argument register set a corresponding bit in the match register.
- After the matching process, those bits in the match register that have been set indicate the fact that their corresponding words have been matched.
- Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.

Hardware Organization

- The key register provides a mask for choosing a particular field or key in the argument word.
- The entire argument is compared with each memory word if the key register contains all 1's.
- Otherwise, only those bits in the argument that have 1st in their corresponding position of the key register are compared.
- Thus the key provides a mask or identifying piece of information which specifies how the reference to memory is made.
- To illustrate with a numerical example, suppose that the argument register A and the key register K have the bit configuration shown below.
- Only the three leftmost bits of A are compared with memory words because K has 1's in these position.

A	101 111100	
K	111 000000	
Word1	100 111100	no match
Word2	101 000001	match

Word 2 matches the unmasked argument field because the three leftmost bits of the argument and the word are equal.

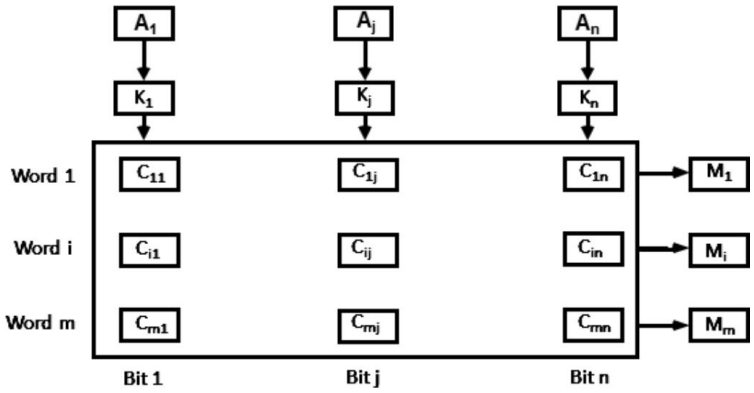


Fig. Associative memory of *m* word, n cells per word

- The relation between the memory array and external registers in an associative memory is shown in figure.
- The cells in the array are marked by the letter C with two subscripts.
- The first subscript gives the word number and the second specifies the bit position in the word.
- Thus cell Cij is the cell for bit j in words i.
- A bit Aj in the argument register is compared with all the bits in column j of the array provided that Kj = 1.
- This is done for all columns j = 1, 2... n.
- If a match occurs between all the unmasked bits of the argument and the bits in word i, the corresponding bit Mi in the match register is set to 1.
- If one or more unmasked bits of the argument and the word do not match, Mi is cleared to 0.

4.1.6 Cache memory

Q6. Define Cache memory. Discuss associative mapping in organization of cache memory.

Ans : **(Imp.)**

Cache Memory

- Cache is a fast small capacity memory that should hold those information which are most likely to be accessed.
- The basic operation of the cache is, when the CPU needs to access memory, the cache is examined.
- If the word is found in the cache, it is read from the fast memory. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word.
- The transformation of data from main memory to cache memory is referred to as a mapping process.

Associative Mapping

- Consider the main memory can store 32K words of 12 bits each.
- The cache is capable of storing 512 of these words at any given time.

- For every word stored in cache, there is a duplicate copy in main memory.
- The CPU communicates with both memories.
- It first sends a 15-bit address to cache. If there is a hit, the CPU accepts the 12-bit data from cache, if there is miss, the CPU reads the word from main memory and the word is then transferred to cache.

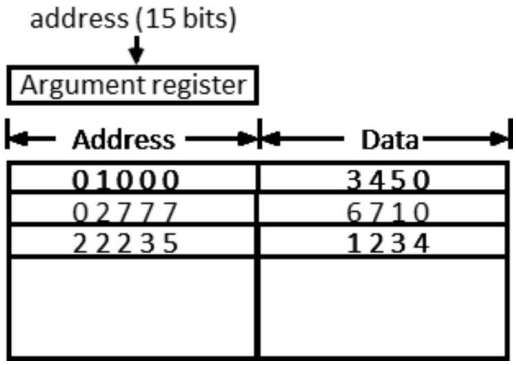


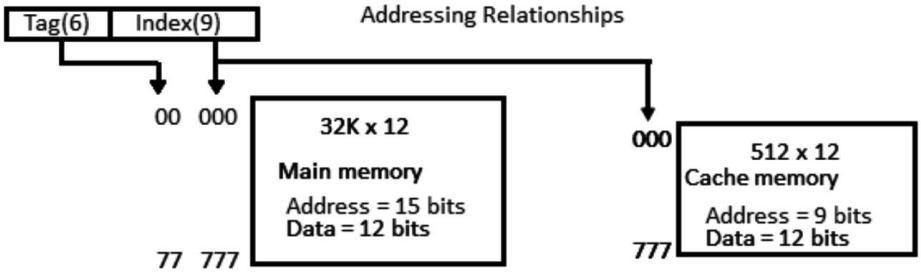
Fig. Associative mapping cache (all numbers in octal)

- The associative memory stores both the address and content (data) of the memory word.
- This permits any location in cache to store any word from main memory.
- The figure shows three words presently stored in the cache. The address value of 15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number.
- A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address.
- If the address is found the corresponding 12-bit data is read and sent to CPU.
- If no match occurs, the main memory is accessed for the word.
- The address data pairs then transferred to the associative cache memory.
- If the cache is full, an address data pair must be displaced to make room for a pair that is needed and not presently in the cache.
- This constitutes a first-in first-one (FIFO) replacement policy.

Q7. Discuss direct mapping in organization of cache memory.

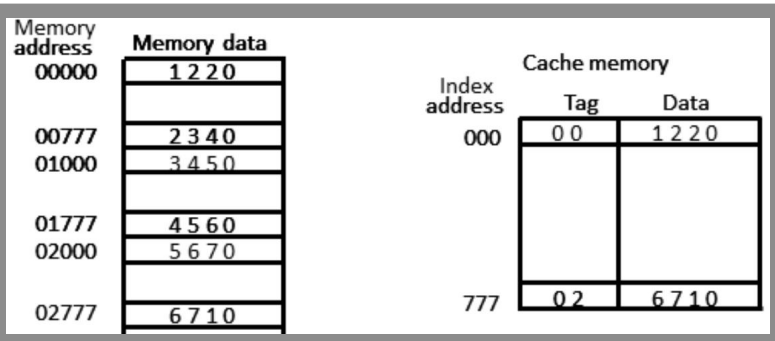
Ans :

- The CPU address of 15 bits is divided into two fields.
- The nine least significant bits constitute the index field and the remaining six bits from the tag field.
- The figure (a) shows that main memory needs an address that includes both the tag and the index.



Fig(a) : Addressing relationships between main and cache memories

- The number of bits in the index field is equal to the number of address bits required to access the cache memory.
- The internal organization of the words in the cache memory is as shown in figure .



Fig(b) : Direct mapping cache organization

- Each word in cache consists of the data word and its associated tag.
- When a new word is first brought into the cache, the tag bits are stored alongside the data bits.
- When the CPU generates a memory request the index field is used for the address to access the cache.
- The tag field of the CPU address is compared with the tag in the word read from the cache.
- If the two tags match, there is a hit and the desired data word is in cache.
- If there is no match, there is a miss and the required word is read from main memory.
- It is then stored in the cache together with the new tag, replacing the previous value.
- The word at address zero is presently stored in the cache (index = 000, tag = 00, data = 1220).

Suppose that the CPU now wants to access the word at address 02000.

- The index address is 000, so it is used to access the cache. The two tags are then compared.
- The cache tag is 00 but the address tag is 02, which does not produce a match.
- Therefore, the main memory is accessed and the data word 5670 is transferred to the CPU.
- The cache word at index address 000 is then replaced with a tag of 02 and data of 5670.
- The disadvantage of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time.

Q8. Discuss set-associative mapping in organization of cache memory.

Ans :

- A third type of cache organization, called set associative mapping in that each word of cache can store two or more words of memory under the same index address.
- Each data word is stored together with its tag and the number of tag-data items in one word of cache is said to form a set.
- An example one set-associative cache organization for a set size of two is shown in figure.

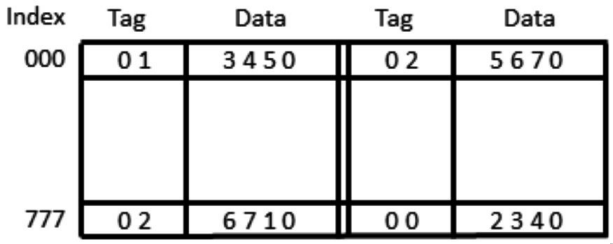


Fig. Two-way set-associative mapping cache

- Each index address refers to two data words and their associated terms.
- Each tag required six bits and each data word has 12 bits, so the word length is 2 (6 + 12) = 36 bits.
- An index address of nine bits can accommodate 512 words.
- Thus the size of cache memory is 512 × 36. It can accommodate 1024 words or main memory since each word of cache contains two data words.
- In generation a set-associative cache of set size k will accommodate K word of main memory in each word of cache.
- The octal numbers listed in figure 9.8 are with reference to the main memory contents.
- The words stored at addresses 01000 and 02000 of main memory are stored in cache memory at index address 000.
- Similarly, the words at addresses 02777 and 00777 are stored in cache at index address 777.
- When the CPU generates a memory request, the index value of the address is used to access the cache.
- The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs.
- The comparison logic is done by an associative search of the tags in the set similar to an associative memory search: thus the name “set-associative”.
- When a miss occurs in a set-associative cache and the set is full, it is necessary to replace one of the tag-data items with a new value.
- The most common replacement algorithms used are: random replacement, first-in firstout (FIFO), and least recently used (LRU).

Q9. Explain Write-through and Write-back cache write method.

Ans :

Write Through

- The simplest and most commonly used procedure is to update main memory with every memory write operation.
- The cache memory being updated in parallel if it contains the word at the specified address. This is called the *write-through* method.
- This method has the advantage that main memory always contains the same data as the cache.
- This characteristic is important in systems with direct memory access transfers.
- It ensures that the data residing in main memory are valid at all times so that an I/O device communicating through DMA would receive the most recent updated data.

Write-Back (Copy-Back)

- The second procedure is called the write-back method.
- In this method only the cache location is updated during a write operation.
- The location is then marked by a flag so that later when the word is removed from the cache it is copied into main memory.
- The reason for the write-back method is that during the time a word resides in the cache, it may be updated several times.
- However, as long as the word remains in the cache, it does not matter whether the copy in main memory is out of date, since requests from the word are filled from the cache.
- It is only when the word is displaced from the cache that an accurate copy need be rewritten into main memory.

4.1.7 Virtual Memory

Q10. What do you mean by address space and memory space in virtual memory? Also explain the relation between address space and memory space in virtual memory.

Ans :

Virtual Memory

- Virtual memory is used to give programmers the illusion that they have a very large memory at their disposal, even though the computer actually has a relatively small main memory.
- A virtual memory system provides a mechanism for translating program-generated addresses into correct main memory locations.

Address space

- An address used by a programmer will be called a virtual address, and the set of such addresses is known as address space.

Memory space

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

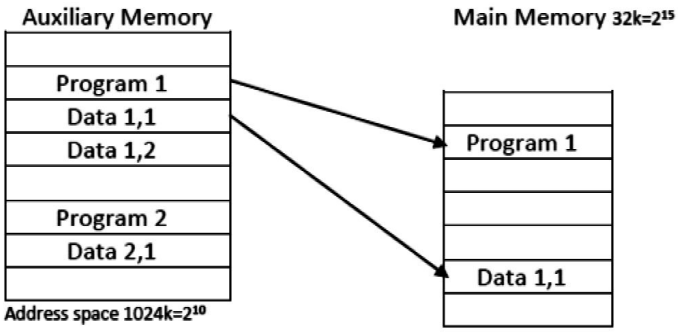


Fig. Relation between address and memory space in a virtual memory system

- As an illustration, consider a computer with a main-memory capacity of 32K words (K = 1024). Fifteen bits are needed to specify a physical address in memory since 32K = 215.
- Suppose that the computer has available auxiliary memory for storing 220 = 1024K words.
- Thus auxiliary memory has a capacity for storing information equivalent to the capacity of 32 main memories.
- Denoting the address space by N and the memory space by M, we then have for this example N = 1024K and M = 32K.
- In a multiprogramming computer system, programs and data are transferred to and from auxiliary memory and main memory based on demands imposed by the CPU.
- Suppose that program 1 is currently being executed in the CPU. Program 1 and a portion of its associated data are moved from auxiliary memory into main memory as shown in figure.
- Portions of programs and data need not be in contiguous locations in memory since information is being moved in and out, and empty spaces may be available in scattered locations in memory.
- In our example, the address field of an instruction code will consist of 20 bits but physical memory addresses must be specified with only 15 bits.
- Thus CPU will reference instructions and data with a 20-bit address, but the information at this address must be taken from physical memory because access to auxiliary storage for individual words will be too long.

Q11. Explain address mapping using pages.

Ans :

- The table implementation of the address mapping is simplified if the information in the address space and the memory space are each divided into groups of fixed size.
- The physical memory is broken down into groups of equal size called blocks, which may range from 64 to 4096 words each.
- The term page refers to groups of address space of the same size.
- Consider a computer with an address space of 8K and a memory space of 4K.
- If we split each into groups of 1K words we obtain eight pages and four blocks as shown in figure
- At any given time, up to four pages of address space may reside in main memory in any one of the four blocks.

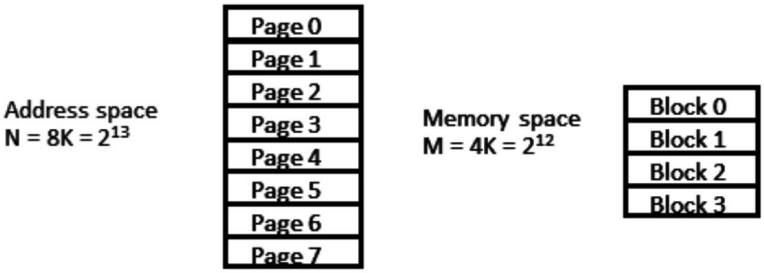


Fig. Address and Memory space split into group of 1K words

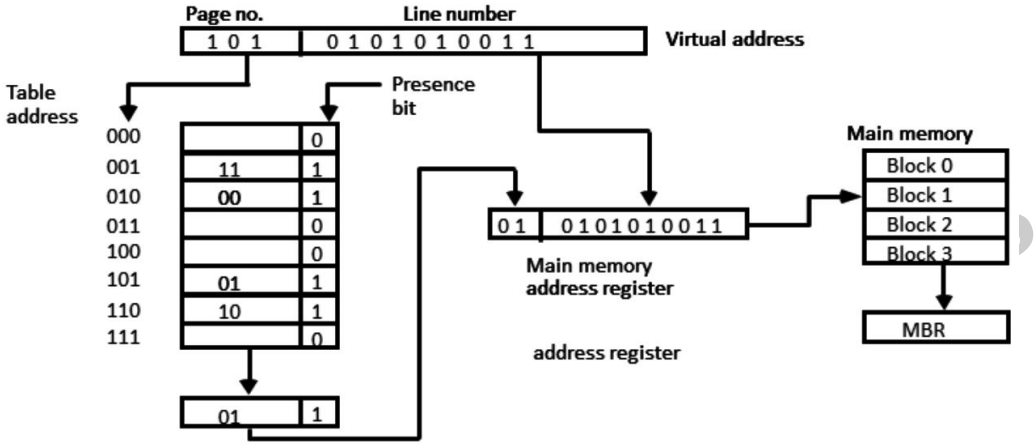


Fig. Memory table in paged system

- The organization of the memory mapping table in a paged system is shown in figure 9.10.
- The memory-page table consists of eight words, one for each page.
- The address in the page table denotes the page number and the content of the word give the block number where that page is stored in main memory.
- The table shows that pages 1, 2, 5, and 6 are now available in main memory in blocks 3, 0, 1, and 2, respectively.
- A presence bit in each location indicates whether the page has been transferred from auxiliary memory into main memory.
- A 0 in the presence bit indicates that this page is not available in main memory.
- The CPU references a word in memory with a virtual address of 13 bits.
- The three high-order bits of the virtual address specify a page number and also an address for the memory-page table.
- The content of the word in the memory page table at the page number address is read out into the memory table buffer register.
- If the presence bit is a 1, the block number thus read is transferred to the two high-order bits of the main memory address register.
- The line number from the virtual address is transferred into the 10 low-order bits of the memory address register.

- A read signal to main memory transfers the content of the word to the main memory buffer register ready to be used by the CPU.
- If the presence bit in the word read from the page table is 0, it signifies that the content of the word referenced by the virtual address does not reside in main memory.

Q12. What is segment? What is logical address? Explain segmented page mapping.

Ans :

Segment

A segment is a set of logically related instructions or data elements associated with a given name.

Logical address

The address generated by segmented program is called a logical address.

Segmented page mapping

The length of each segment is allowed to grow and contract according to the needs of the program being executed. Consider logical address shown in figure 9.12.

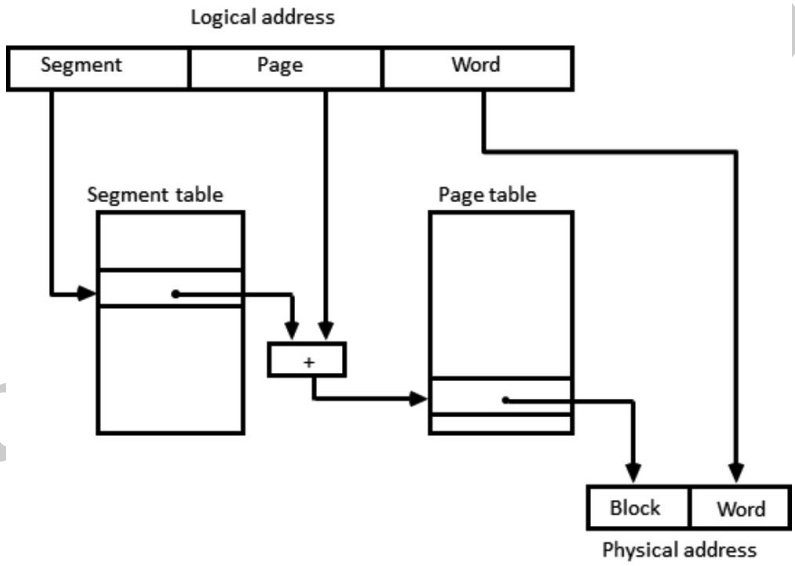


Fig. Logical to physical address mapping

- The logical address is partitioned into three fields.
- The segment field specifies a segment number.
- The page field specifies the page within the segment and word field gives specific word within the page.
- A page field of k bits can specify up to 2k pages.
- A segment number may be associated with just one page or with as many as 2k pages.
- Thus the length of a segment would vary according to the number of pages that are assigned to it.
- The mapping of the logical address into a physical address is done by means of two tables, as shown in figure.

- The segment number of the logical address specifies the address for the segment table.
- The entry in the segment table is a pointer address for a page table base.
- The page table base is added to the page number given in the logical address.
- The sum produces a pointer address to an entry in the page table.
- The concatenation of the block field with the word field produces the final physical mapped address.
- The two mapping tables may be stored in two separate small memories or in main memory.
- In either case, memory reference from the CPU will require three accesses to memory:
- one from the segment table, one from the page table and the third from main memory.
- This would slow the system significantly when compared to a conventional system that requires only one reference to memory.

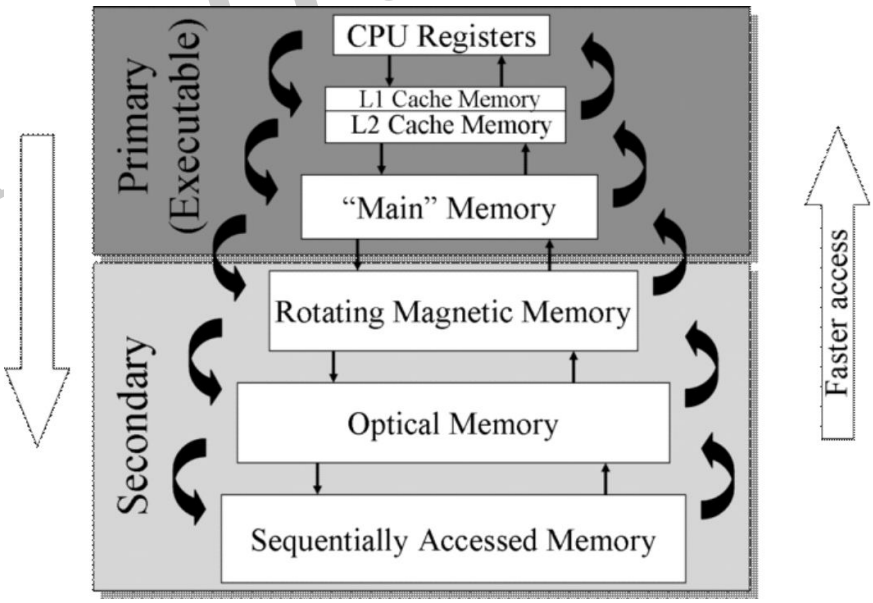
4.1.8 Memory Management hardware.

Q13. Write about memory management hardware.

Ans : **(Imp.)**

Memory Hierarchy

Computers have several different types of memory. This memory is often viewed as a hierarchy as shown below.



Our main concern here will be the computer's main or RAM memory. The cache memory is important because it boost's the speed of accessing memory, but it is managed entirely by the hardware. The rotating magnetic memory or disk memory is used by the Virtual Memory Management.

The Memory Management Unit (MMU) performs translations.

The MMU contains the following:

- The table walk unit, which contains logic that reads the translation tables from memory.
- *Translation Lookaside Buffers* (TLBs), which cache recently used translations.

All memory addresses that are issued by software are virtual. These memory addresses are passed to the MMU, which checks the TLBs for a recently used cached translation. If the MMU does not find a recently cached translation, the table walk unit reads the appropriate table entry, or entries, from memory, as shown here:

A virtual address must be translated to a physical address before a memory access can take place (because we must know which physical memory location we are accessing). This need for translation also applies to cached data, because on Armv6 and later processors, the data caches store data using the physical address (addresses that are physically tagged). Therefore, the address must be translated before a cache lookup can complete.

Note: Architecture is a behavioural specification. The caches must behave as if they are physically tagged. An implementation might do something different, as long as this is not software-visible.

Table Entry

The translation tables work by dividing the virtual address space into equal-sized blocks and by providing one entry in the table per block.

Entry 0 in the table provides the mapping for block 0, entry 1 provides the mapping for block 1, and so on. Each entry contains the address of a corresponding block of physical memory and the attributes to use when accessing the physical address.

Table Lookup

A table lookup occurs when a translation takes place. When a translation happens, the virtual address that is issued by the software is split in two, as shown in this diagram:

This diagram shows a single-level lookup.

The upper-order bits, which are labelled 'Which entry' in the diagram, tell you which block entry to look in and they are used as an index into the table. This entry block contains the physical address for the virtual address.

The lower-order bits, which are labelled 'Offset in block' in the diagram, are an offset within that block and are not changed by the translation.

Multilevel Translation

In a single-level lookup, the virtual address space is split into equal-sized blocks. In practice, a hierarchy of tables is used.

The first table (Level 1 table) divides the virtual address space into large blocks. Each entry in this table can point to an equal-sized block of physical memory or it can point to another table which subdivides the block into smaller blocks. We call this type of table a 'multilevel table'. Here we can see an example of a multilevel table that has three levels:

In Armv8-A, the maximum number of levels is four, and the levels are numbered 0 to 3. This multilevel approach allows both larger blocks and smaller blocks to be described. The characteristics of large and small blocks are as follows:

- Large blocks require fewer levels of reads to translate than small blocks. Plus, large blocks are more efficient to cache in the TLBs.
- Small blocks give software fine-grain control over memory allocation. However, small blocks are less efficient to cache in the TLBs. Caching is less efficient because small blocks require multiple reads through the levels to translate.

To manage this trade-off, an OS must balance the efficiency of using large mappings against the flexibility of using smaller mappings for optimum performance.

Note: The processor does not know the size of the translation when it starts the table lookup. The processor works out the size of the block that is being translated by performing the table walk.

UNIT V

Input-Output Organization

Peripheral Devices, Input-Output Interface, Asynchronous data transfer, Modes of Transfer, Priority Interrupt, Direct Memory Access (DMA), I/O Processor, Serial Communication.

Pipeline Processing

Arithmetic, Instruction and RISC Pipelines. Assessing and Understanding Performance: CPU performance and its factors, Evaluating performance.

5.1 INPUT-OUTPUT ORGANIZATION

5.1.1 Peripheral Devices

Q1. Define Peripherals. Explain I/O Bus and Interface Modules.

Ans : **(Imp.)**

Peripherals:

Input-output device attached to the computer are also called peripherals.

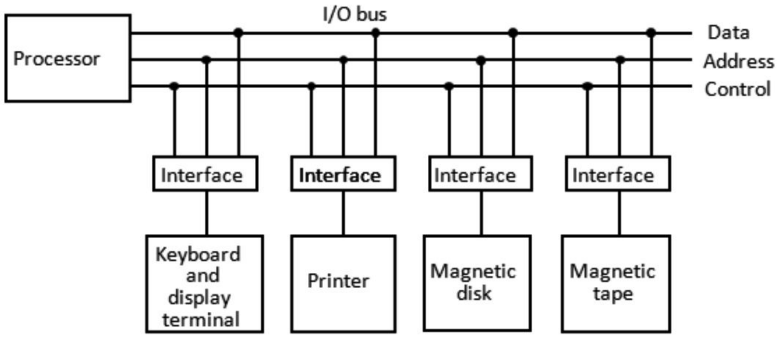


Fig. Connection of I/O bus to input-output device

- A typical communication link between the processor and several peripherals is shown in figure.
- The I/O bus consists of data lines, address lines, and control lines.
- The magnetic disk, printer, and terminal are employed in practically any general purpose computer.
- Each peripheral device has associated with it an interface unit.
- Each interface decodes the address and control received from the I/O bus, interprets them for the peripheral, and provides signals for the peripheral controller.
- It also synchronizes the data flow and supervises the transfer between peripheral and processor.
- Each peripheral has its own controller that operates the particular electromechanical device.
- For example, the printer controller controls the paper motion, the print timing, and the selection of printing characters.
- The I/O bus from the processor is attached to all peripheral interfaces.

- To communicate with a particular device, the processor places a device address on the address lines.
- Each interface attached to the I/O bus contains an address decoder that monitors the address lines.
- When the interface detects its own address, it activates the path between the bus lines and the device that it controls.
- All peripherals whose address does not correspond to the address in the bus are disabled by their interface selected responds to the function code and proceeds to execute it.
- The function code is referred to as an I/O command.
- There are four types of commands that an interface may receive. They are classified as control, status, data output, and data input.
- A control command is issued to activate the peripheral and to inform it what to do.
- For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction.
- A status command is used to test various status conditions in the interface and the peripheral.
- For example, the computer may wish to check the status of the peripheral before a transfer is initiated.
- During the transfer, one or more errors may occur which are detected by the interface.
- These errors are designated by setting bits in a status register that the processor can read at certain intervals.
- A data output command causes the interface to respond by transferring data from the bus into one of its registers.
- The computer starts the tape moving by issuing a control command.
- The processor then monitors the status of the tape by means of a status command.
- When the tape is in the correct position, the processor issues a data output command.
- The interface responds to the address and command and transfers the information from the data lines in the bus to its buffer register.
- The interface that communicates with the tape controller and sends the data to be stored on tape.
- The data input command is the opposite of the data output.
- In this case the interface receives an item of data from the peripheral and places it in its buffer register.
- The processor checks if data are available by means of a status command and then issues a data input command.
- The interface places the data on the data lines, where they are accepted by the processor.

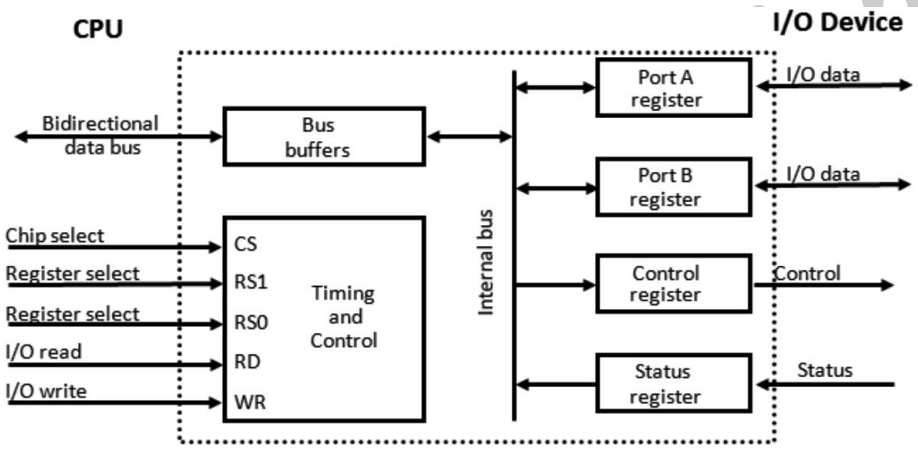
5.1.2 Input-Output Interface

Q2. Explain I/O interface with example.

Ans :

- It consists of two data registers called ports, a control register, a status register, bus buffers, and timing and control circuit.

- The interface communicates with the CPU through the data bus.
- The chip select and register select inputs determine the address assigned to the interface.
- The I/O read and writes are two control lines that specify an input or output, respectively.
- The four registers communicate directly with the I/O device attached to the interface.
- The I/O data to and from the device can be transferred into either port A or port B.
- If the interface is connected to a printer, it will only output data, and if it services a character reader, it will only input data.
- A magnetic disk unit transfers data in both directions but not at the same time, so the interface can use bidirectional lines.
- A command is passed to the I/O device by sending a word to the appropriate interface register.
- The control register receives control information from the CPU. By loading appropriate bits into the control register, the interface and the I/O device attached to it can be placed in a variety of operating modes.



CS	RS1	RS0	Register Selected
0	X	X	None: data bus in high impedance
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Figure : Example of I/O interface unit

- For example, port A may be defined as an input port and port B as an output port.
- A magnetic tape unit may be instructed to rewind the tape or to start the tape moving in the forward direction.
- The bits in the status register are used for status conditions and for recording errors that may occur during the data transfer.
- For example, a status bit may indicate that port A has received a new data item from the I/O device.

- Another bit in the status register may indicate that a parity error has occurred during the transfer.
- The interface registers communicate with the CPU through the bidirectional data bus.
- The address bus selects the interface unit through the chip select and the two register select inputs.
- A circuit must be provided externally (usually, a decoder) to detect the address assigned to the interface registers.
- This circuit enables the chip select (CS) input when the interface is selected by the address bus.
- The two register select inputs RS1 and RS0 are usually connected to the two least significant lines of the address bus.
- These two inputs select one of the four registers in the interface as specified in the table accompanying the diagram.
- The content of the selected register is transfer into the CPU via the data bus when the I/O read signal is enables.
- The CPU transfers binary information into the selected register via the data bus when the I/O write input is enabled.

5.1.3 Asynchronous Data Transfer

Q3. What do you mean by Asynchronous data transfer? Explain Strobe control in detail.

Ans : **(Imp.)**

Asynchronous Data Transfer

Data transfer between two independent units, where internal timing in each unit is independent from the other. Such two units are said to be asynchronous to each other.

Strobe Control

The Strobe control method of asynchronous data transfer employs a single control line to time each transfer.

Source-initiated strobe for data transfer

- The strobe may be activated by either the source or the destination unit. Figure 8.3 shows a source-initiated transfer.
- The data bus carries the binary information from source unit to the destination unit.
- The strobe is a single line that informs the destination unit when a valid data word is available in the bus.
- The source unit first places the data on the data bus.
- After a delay to ensure that the data settle to a steady value, the source activates the strobe pulse.
- The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data.
- The source removes the data from the bus a brief period after it disables its strobe pulse.

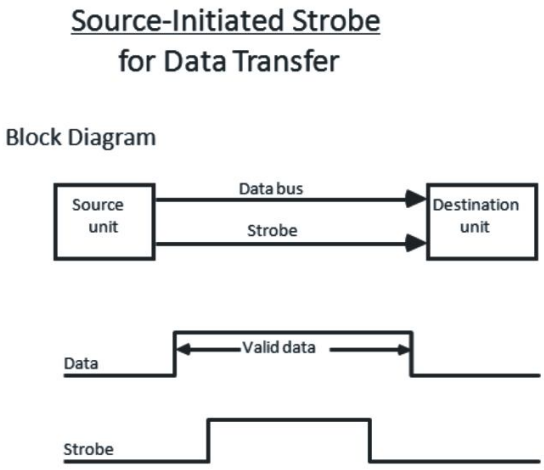


Fig. Source-initiated strobe for data transfer

Destination-initiated strobe for data transfer

- It shows a data transfer initiated by the destination unit. In this case the destination unit activates the strobe pulse, informing the source to provide the data.
- The source unit responds by placing the requested binary information on the data bus.
- The data must be valid and remain in the bus long enough for the destination unit to accept it.
- The falling edge of the strobe pulse can be used again to trigger a destination register.
- The destination unit then disables the strobe. The source removes the data from the bus after a predetermined time interval.
- The transfer of data between the CPU and an interface unit is similar to the strobe transfer just described.

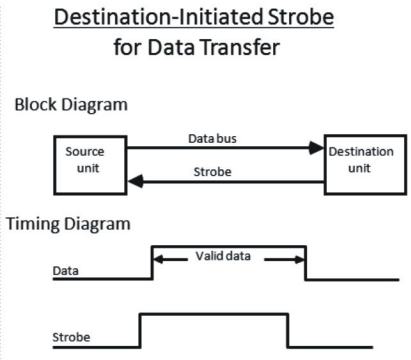


Figure. Destination-initiated strobe for data transfer

Disadvantage of Strobe method

- The disadvantage of the strobe method is that the source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was placed in the bus
- Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on the bus.

Q4. Explain Asynchronous data transfer with Handshaking method.

Ans :

The handshake method solves the problem of Strobe method by introducing a second control signal that provides a reply to the unit that initiates the transfer.

Source-initiated transfer using handshaking

- One control line is in the same direction as the data flow in the bus from the source to the destination.
- It is used by the source unit to inform the destination unit whether there are valid data in the bus.

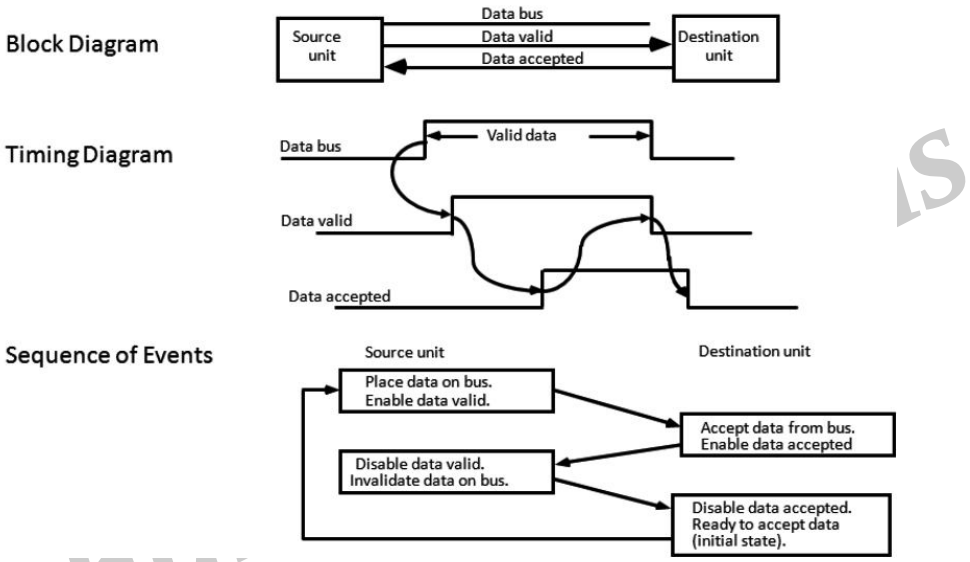


Fig. Source-initiated transfer using handshaking

- The other control line is in the other direction from the destination to the source.
- It is used by the destination unit to inform the source whether it can accept data.
- The sequence of control during the transfer depends on the unit that initiates the transfer.
- Figure (a) shows the data transfer procedure initiated by the source.
- The two handshaking lines the data valid, which is generated by the source unit, and data accepted, generated by the destination unit, the timing diagram shows the exchange of signals between the two units.
- The sequence of events listed in figure (a) shows the four possible states that the system can be at any given time.
- The source unit initiates the transfer by placing the data on the bus and enabling its data valid signal.
- The data accepted signal is activated by the destination unit after it accepts the data from the bus.
- The source unit then disables its data valid signal, which invalidates the data on the bus.
- The destination unit then disables its data accepted signal and the system goes into its initial state.

- The source does not send the next data item until after the destination unit shows its readiness to accept new data by disabling its data accepted signal.
- This scheme allows arbitrary delays from one state to the next and permits each unit to respond at its own data transfer rate.

Destination-initiated transfer using handshaking

- The destination-initiated transfer using handshaking lines is shown in figure (b)
- Note that the name of the signal generated by the destination unit has been changed to ready for data to reflect its new meaning.
- The source unit in this case does not place data on the bus until after it receives the ready for data signal from the destination unit.
- From there on, the handshaking procedure follows the same pattern as in the source initiated case.
- Note that the sequence of events in both cases would be identical if we consider the ready for data signal as the complement of data accepted.
- In fact, the only difference between the source-initiated and the destination-initiated transfer is in their choice of initial state.

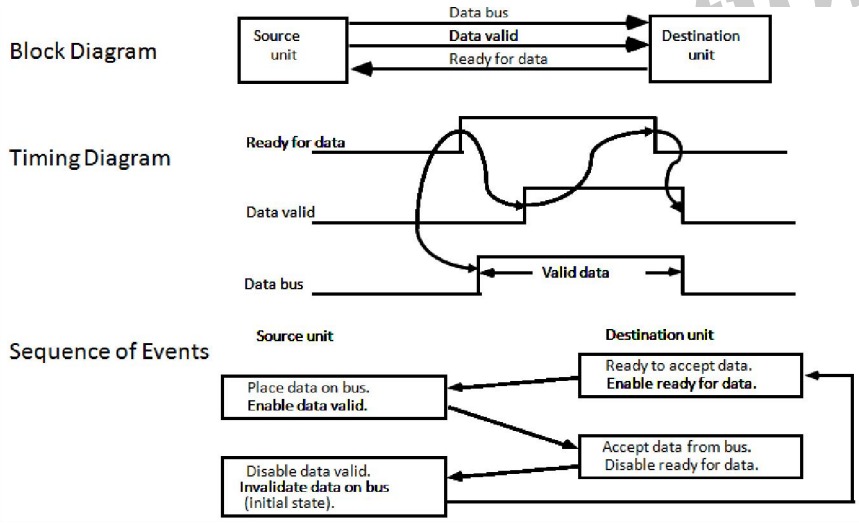


Fig (b). Destination-initiated transfer using handshaking

5.1.4 Modes of Transfer

Q5. Explain Programmed I/O with example.

Ans :

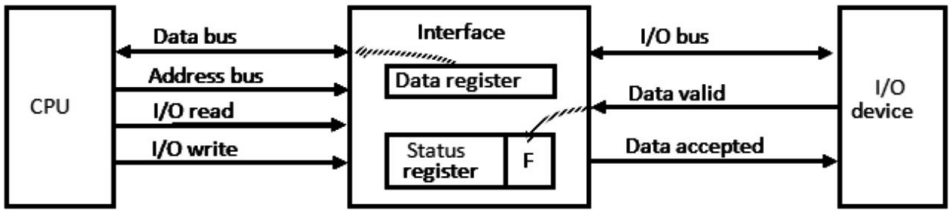


Fig (a): Data transfer from I/O device to CPU

- In the programmed I/O method, the I/O device does not have direct access to memory.
- An example of data transfer from an I/O device through an interface into the CPU is shown in figure (a)
- When a byte of data is available, the device places it in the I/O bus and enables its data valid line.
- The interface accepts the byte into its data register and enables the data accepted line.
- The interface sets a bit in the status register that we will refer to as an F or "flag" bit.
- The device can now disables the data valid line, but it will not transfer another byte until the data accepted line is disables by the interface.
- A program is written for the computer to check the flag in the status register to determine if a byte has been placed in the data register by the I/O device.
- This is done by reading the status register into a CPU register and checking the value of the flag bit.
- Once the flag is cleared, the interface disables the data accepted line and the device can then transfer the next data byte.

Example of Programmed I/O

- A flowchart of the program that must be written for the CPU is shown in figure (b)
- It is assumed that the device is sending a sequence of bytes that must be stored in memory.
- The transfer of each byte requires three instructions :
- Read the status register.
- Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
- Read the data register.
- Each byte is read into a CPU register and then transferred to memory with a store instruction.
- A common I/O programming task is to transfer a block of words from an I/O device and store them in a memory buffer.

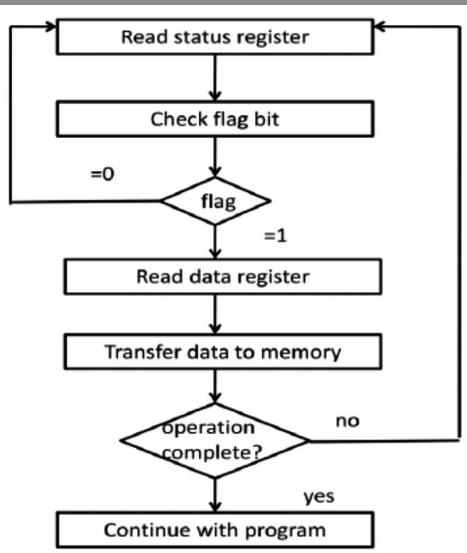


Fig (b). Flowchart for CPU program to input data

Q6. Write a note on Interrupt Initiated I/O

Ans :

- In programmed initiated, CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer.
- This is a time consuming process since it keeps the processor busy needlessly.
- It can be avoided by using an interrupt facility and a special command to inform the interface to issue an interrupt request signal when data are available from the device.
- In the meantime CPU can proceed to execute another program.
- The interface meanwhile keeps monitoring the device.
- When the interface determines that the device is ready for data transfer, it generates an interrupt request to the computer.
- While the CPU is running a program, it does not check the flag. However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that the flag has been set.
- The CPU deviates from what it is doing to take care of the input or output transfer.
- After the transfer is completed, the computer returns to the previous program to continue what it was doing before the interrupt.
- The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer.
- The way that the processor chooses the branch address of the service routine varies from one unit to another.
- In non-vectored interrupt, branch address is assigned to a fixed location in memory.
- In a vectored interrupt, the source that interrupts supplies the branch information to the computer. The information is called vector interrupt.
- In some computers the interrupt vector is the first address of the I/O service routine.
- In other computers the interrupt vector is an address that points to a location in memory where the beginning address of the I/O service routine is stored.

5.1.5 Priority Interrupt

Q7. What is priority interrupt? Explain briefly about Daisy Chaining Priority.

Ans : **(Imp.)**

- Determines which interrupt is to be served first when two or more requests are made simultaneously
- Also determines which interrupts are permitted to interrupt the computer while another is being serviced
- Higher priority interrupts can make requests while servicing a lower priority interrupt.

Daisy Chaining Priority

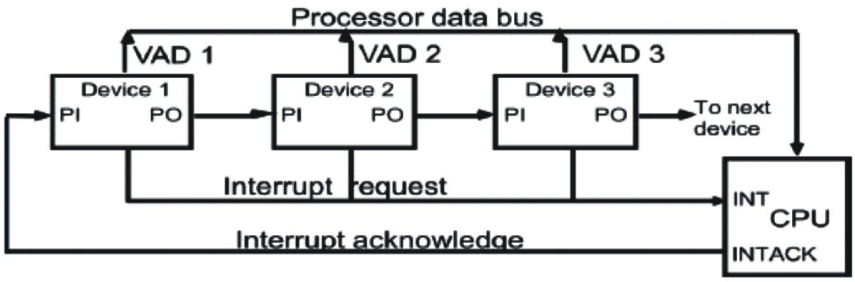


Fig. Daisy-chain priority interrupt

- The daisy-chaining method of establishing priority consists of a serial connection of all devices that request an interrupt.
- The device with the highest priority is placed in the first position, followed by lower priority devices up to the device with the lowest priority, which is placed last in the chain.
- This method of connection between three devices and the CPU is shown in figure .
- If any device has its interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU.
- When no interrupts are pending, the interrupt line stays in the high-level state and no interrupts are recognized by the CPU.
- The CPU responds to an interrupt request by enabling the interrupt acknowledge line.
- This signal passes on to the next device through the PO (priority out) output only if device 1 is not requesting an interrupt.
- If device 1 has a pending interrupt, it blocks the acknowledge signal from the next device by placing a 0 in the PO output.
- It then proceeds to insert its own interrupt vector address (VAD) into the data bus for the CPU to use during the interrupt cycle.
- A device with a 0 in its PI input generates a 0 in its PO output to inform the next-lower priority device that the acknowledge signal has been blocked.
- A device that is requesting an interrupt and has a 1 in its PI input will intercept the acknowledge signal by placing a 0 in its PO output.
- If the device does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output.
- Thus the device with PI = 1 and PO = 0 is the one with the highest priority that is requesting an interrupt, and this device places its VAD on the data bus.
- The daisy chain arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU.
- The farther the device is from the first position; the lower is its priority.

5.1.6 Direct Memory Access (DMA)

Q8. Write a detailed note on Direct Memory Access (DMA).

Ans :

Direct Memory Access

- Transfer of data under programmed I/O is between CPU and peripheral.
- In direct memory access (DMA), Interface transfers data into and out of memory through the memory bus.
- The CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks.
- When the transfer is made, the DMA requests memory cycles through the memory bus.
- When the request is granted by the memory controller, DMA transfers the data directly into memory.

DMA Controller

- DMA controller - Interface which allows I/O transfer directly between Memory and Device, freeing CPU for other tasks
- CPU initializes DMA Controller by sending memory address and the block size (number of words).

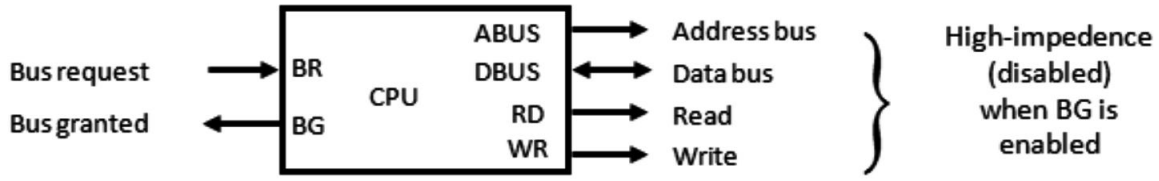


Fig. CPU bus signals for DMA transfer

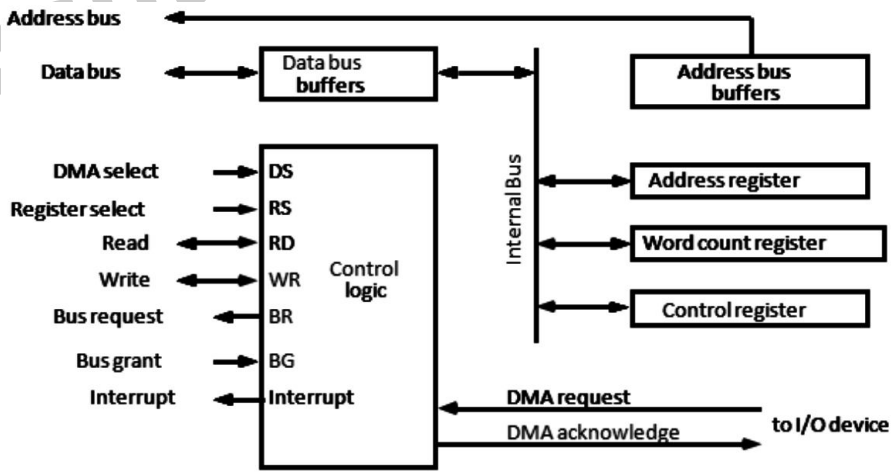


Fig. Block diagram of DMA controller

- The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device.
- In addition, it needs an address register, a word count register, and a set of address lines.

- The address register and address lines are used for direct communication with the memory.
- The word count register specifies the number of words that must be transferred.
- The data transfer may be done directly between the device and memory under control of the DMA.
- Figure shows the block diagram of a typical DMA controller.
- The unit communicates with the CPU via the data bus and control lines.
- The register in the DMA are selected by the CPU through the address bus by enabling the
- DS (DMA select) and RS (register select) inputs.
- The RD (read) and WR (write) inputs are bidirectional.
- When the BG (bus grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers.
- When BG = 1, the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address but and activating the RD or WR control.
- The DMA communicates with the external peripheral through the request and acknowledge lines by using a prescribed handshaking procedure.
- The DMA controller has three registers: an address register, a word count register, and a control register.
- The address register contains an address to specify the desired location in memory.
- The word count register holds the number of words to be transferred.
- This register is decremented by one after each word transfer and internally tested for zero.
- The control register specifies the mode of transfer.
- All registers in the DMA appear to the CPU as I/O interface registers.
- Thus the CPU can read from or write into the DMA register under program control via the data bus.
- The DMA is first initialized by the CPU.
- After that, the DMA starts and continues to transfer data between memory and peripheral unit until an entire block is transferred.
- The CPU initializes the DMA by sending the following information through the data bus
- The staring address of the memory block where data are available (for read) or where data are to be stored (for write)
- The word count, which is the number of words in the memory block.
- Control to specify the mode of transfer such as read or write.
- The starting address is stored in the address register.

5.1.7 I/O Processor

Q9. Explain Input- Output Processor (IOP)

Ans :

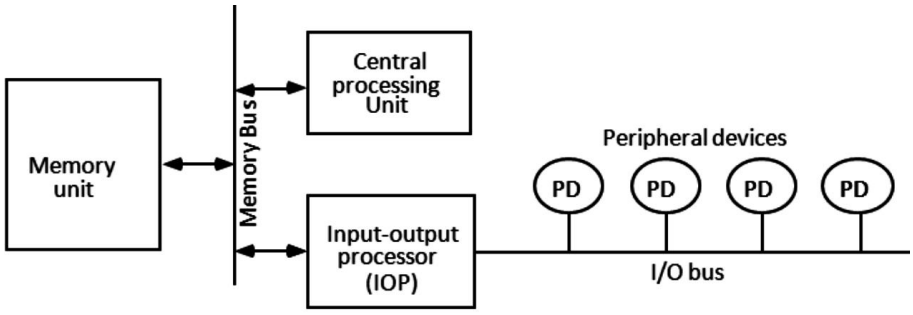


Fig. Block diagram of a computer with I/O processor

- IOP is similar to a CPU except that it is designed to handle the details of I/O processing.
- Unlike the DMA controller that must be setup entirely by the CPU, the IOP can fetch and execute its own instruction.
- IOP instructions are specifically designed to facilitate I/O transfers.
- In addition, IOP can perform other processing tasks, such as arithmetic, logic branching, and code translation.
- The block diagram of a computer with two processors is shown in figure 8.12.
- The memory unit occupies central position and can communicate with each processor by means of direct memory access.
- The CPU is responsible for processing data needed in the solution of computational tasks.
- The IOP provides a path of for transfer of data between various peripheral devices and memory unit.
- The CPU is usually assigned the task of initiating the I/O program.
- From then, IOP operates independent of the CPU and continues to transfer data from external devices and memory.
- The data formats of peripheral devices differ from memory and CPU data formats. The IOP must structure data words from many different sources.
- For example, it may be necessary to take four bytes from an input device and pack them into one 32-bit word before the transfer to memory.
- Data are gathered in the IOP at the device rate and bit capacity while the CPU is executing its own program.
- After the input data are assembled into a memory word, they are transferred from IOP directly into memory by “stealing” one memory cycle from the CPU.
- similarly, an output word transferred from memory to the IOP is directed from the IOP to the output word transferred from memory to the IOP.

- In most computer systems, the CPU is the master while the IOP is a slave processor.
- The CPU is assigned the task of initiating all operations, but I/O instructions are executed in the IOP.
- CPU instructions provide operations to start an I/O transfer and also to test I/O status conditions needed for making decisions on various I/O activities.
- The IOP, in turn, typically asks for CPU attention by means of an interrupt.
- Instructions that are read from memory by an IOP are sometimes called commands, to distinguish them from instructions that are read by the CPU.

Q10. Explain CPU-IOP Communication.

Ans :

- The communication between CPU and IOP may take different forms, depending on the particular computer considered.
- In most cases the memory unit acts.
- The sequence of operations may be carried out as shown in the flowchart of figure
- The CPU sends an instruction to test the IOP path.
- The IOP responds by inserting a status word in memory for the CPU to check.
- The bits of the status word indicate the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer, or device ready for I/O transfer.
- The CPU refers to the status word in memory to device what do next.
- If all is in order, the CPU sends the instruction to start I/O transfer.
- The memory address received with this instruction tells the IOP where to find its program.
- The CPU can now continue with another program while the IOP is busy with the I/O program.
- Both programs refer to memory by means of DMA transfer.
- When the IOP terminates the execution of its program, it sends an interrupt request to the CPU.
- The CPU responds to the interrupt by issuing an instruction to read the status from the IOP.
- The IOP responds by placing the contents of its status report into a specified memory location.
- The status word indicates whether the transfer has been completed or if any errors occurred during the transfer.
- From inspection of the bits in the status word, the CPU determines if the I/O operation was completed satisfactorily without errors.
- The IOP takes care of all data transfers between several I/O units and the memory while the CPU is processing another program.
- The IOP and CPU are competing for the use of memory, so the number of devices that can be in operation is limited by the access time of the memory.

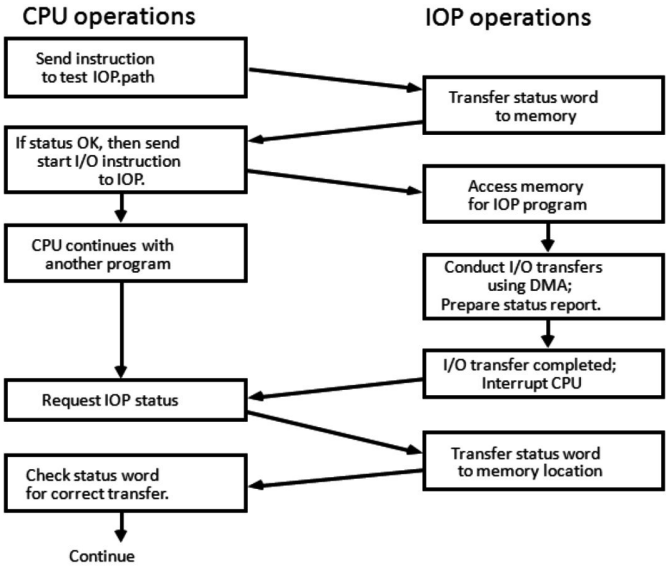


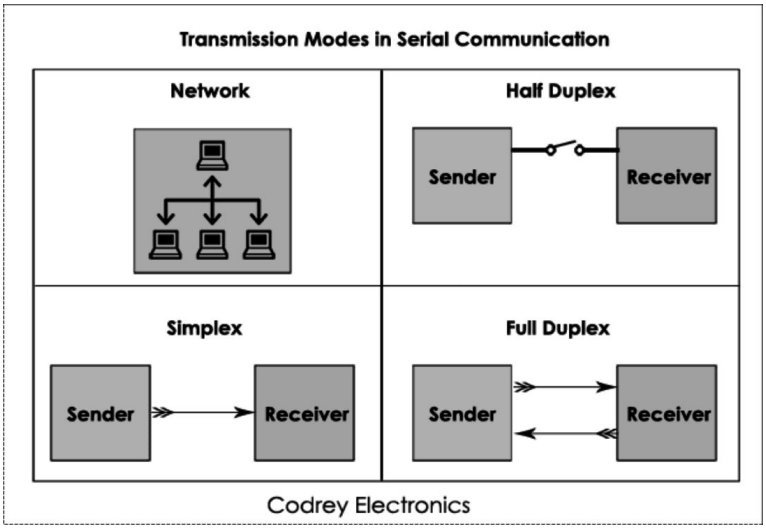
Fig. CPU-IOP communication

5.1.8 Serial Communication

Q11. What is Serial communication?

Ans : **(Imp.)**

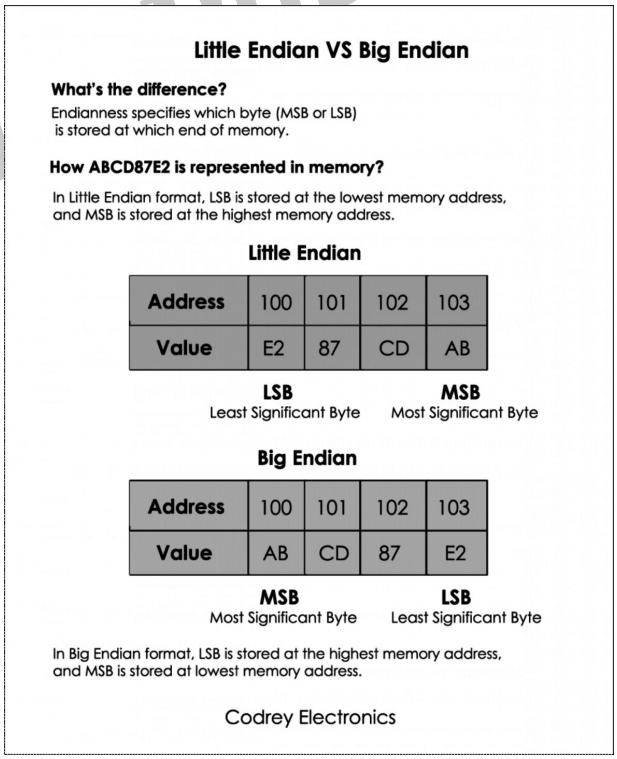
- In serial communication, data is in the form of binary pulses. In other words, we can say Binary One represents a logic HIGH or 5 Volts, and zero represents a logic LOW or 0 Volts.
- Serial communication can take many forms depending on the type of transmission mode and data transfer.
- The transmission modes are classified as Simplex, Half Duplex, and Full Duplex. There will be a source (also known as a sender) and destination (also called a receiver) for each transmission mode.



Transmission Modes – Serial Communication

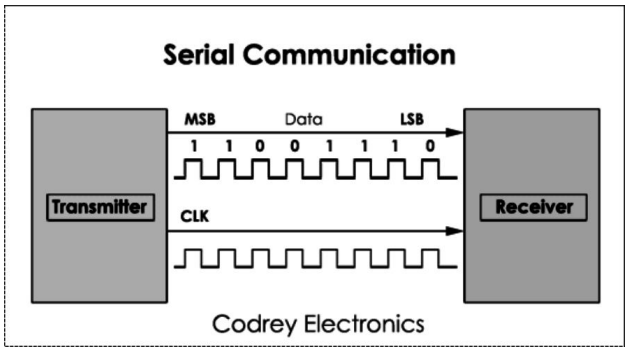
- The **Simplex method** is a one-way communication technique. Only one client (either the sender or receiver is active at a time). If a sender transmits, the receiver can only accept. Radio and Television transmission are the examples of simplex mode.
- In **Half Duplex mode**, both sender and receiver are active but not at a time, i.e. if a sender transmits, the receiver can accept but cannot send and vice versa. A good example is an internet. If a client (laptop) sends a request for a web page, the web server processes the application and sends back the information.
- The **Full Duplex mode** is widely used communication in the world. Here both sender and receiver can transmit and receive at the same time. An example is your smartphone. Beyond the transmission modes, we have to consider the endianness and protocol design of the host computer (sender or receiver). **Endianness** is the way of storing the data at a particular memory address. Depending on the data alignment endian is classified as
  - Little Endian and
  - Big Endian.

Take this example to understand the concept of endianness. Suppose, we have a 32-bit hexadecimal data ABCD87E2. How is this data stored in memory? To have a clear idea, I have explained the difference between Little Endian and Big Endian.



Little Endian Vs Big Endian

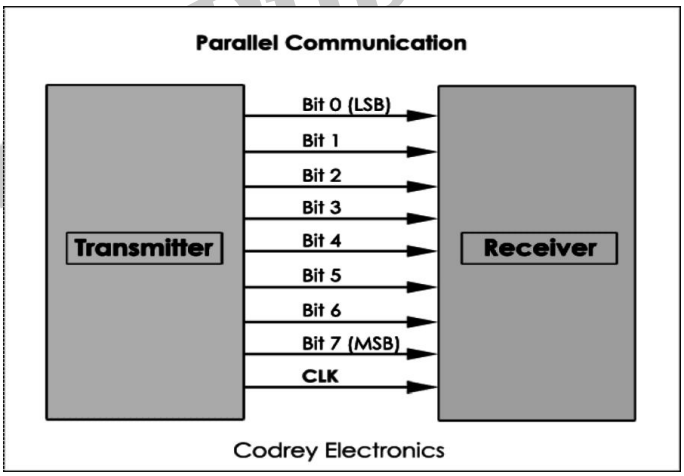
- Data transfer can happen in two ways. They are serial communication and parallel communication. Serial communication is a technique used to send data bit by bit using a two-wires i.e. transmitter (sender) and receiver.
- For example, I want to send an 8-bit binary data 11001110 from the transmitter to the receiver. But, which bit goes out first? Most Significant Bit – MSB (7th bit) or Least Significant Bit- LSB (0th Bit). We cannot say. Here I am considering LSB is moving first (for little Endian).



Serial Communication

From the above diagram, for every clock pulse; the transmitter sends a single bit of data to the receiver.

Parallel communication moves 8,16, or 32 bits of data at a time. Printers and Xerox machines use parallel communication for faster data transfer.



RS232 Parallel Communication

Difference between Serial and Parallel communication

- Serial communication sends only one bit at a time. so, these require fewer I/O (input-output) lines. Hence, occupying less space and more resistant to cross-talk.
- The main advantage of serial communication is, the cost of the entire embedded system becomes cheap and transmits the information over a long distance.
- Serial transfer is used in DCE (Data communication Equipment) devices like a modem.

- In parallel communication, a chunk of data (8,16 or 32 bit) is sent at a time. So, each bit of data requires a separate physical I/O line.
- The advantage of parallel communication is it is fast but its drawback is it use more number of I/O (input-output) lines.
- Parallel transfer is used in PC (personal computer) for interconnecting CPU (central processing unit), RAM (random access memory), modems, audio, video and network hardware.

Serial Communication	Parallel Communication
Sends data bit by bit at one clock pulse	Transfers a chunk of data at a time
Requires one wire to transmit the data	Requires 'n' number of lines for transmitting 'n' bits
Communication speed is slow	Communication speed is fast
Installation cost is low	Installation cost is high
Preferred for long distance communication	Used for short distance communication
Example: Computer to Computer	Computer to multi function printer

Clock Synchronization

For efficient working of serial devices, the clock is the primary source. Malfunction of the clock may lead to unexpected results. The clock signal is different for each serial device, and it is categorized as synchronous protocol and asynchronous protocol.

Synchronous Serial Interface

All the devices on *Synchronous* serial interface use the single CPU bus to share both clock and data. Due to this fact, data transfer is faster. The advantage is there will be no mismatch in baud rate. Moreover, fewer I/O (input-output) lines are required to interface components. Examples are I2C, SPI etc.

Asynchronous Serial Interface

The *asynchronous* interface does not have an external clock signal, and it relies on four parameters namely

1. Baud rate control
2. Data flow control
3. Transmission and reception control
4. Error control.

Asynchronous protocols are suitable for stable communication. These are used for long distance applications. Examples of asynchronous protocols are RS-232, RS-422, and RS-485.

How Serial communication Works?

Advanced CPU such as microcontroller and Microprocessor make use of serial communication to communicate with the external world as well as on the chip peripherals. To get familiar, let us take a simple example. For suppose, you want to send a file present in your laptop to smartphone. How would you send? Probably using Bluetooth or WiFi protocol, Right.

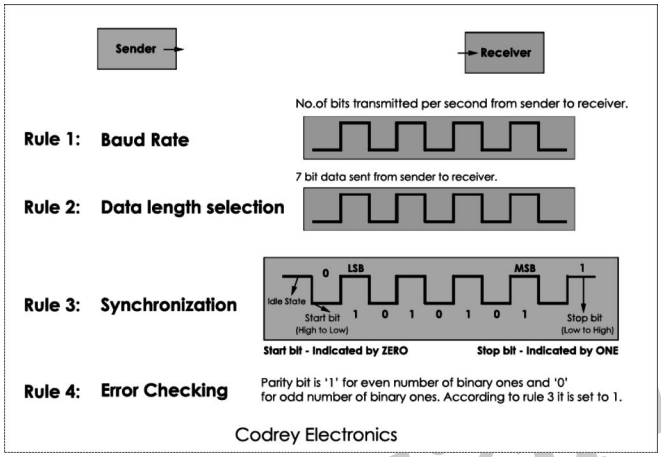
So, here are the steps to establish the serial communication

1. Add the connection

In the first step, your laptop will search for devices nearby 100m and will list out the devices found. This process is often called roaming.

2. Select the device you want to communicate

To connect to your mobile, the pairing has to be done. The default configuration is already present in the software. So no need to configure the baud rate manually. Beyond this, there are four unknown rules. They are baud rate, data bit selection (framing), start-stop bit, and parity.



5.2 Pipeline Processing

5.2.1 Arithmetic

Q12. Explain Flynn's classification for computers.

Ans : **(Imp.)**

Flynn's Classification

It is based on the multiplicity of Instruction Streams and Data Streams.

Instruction Stream

Sequence of Instructions read from memory.

Data Stream

Operations performed on the data in the processor.

		Number of Data Streams	
		Single	**Multiple**
Number of Instruction Streams	**Single**	SISD	SIMD
	Multiple	MISD	MIMD

Fig. Flynn's Classification

SISD:

- Single instruction stream, single data stream.
- SISD represents the organization of a single computer containing a control unit, a processor unit, and a memory unit.
- Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities.

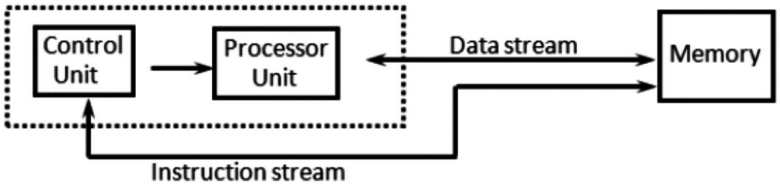


Fig. SISD Organization

SIMD:

- SIMD represents an organization that includes many processing units under the
- supervision of a common control unit.
- All processors receive the same instruction from the control unit but operate on different items of data.

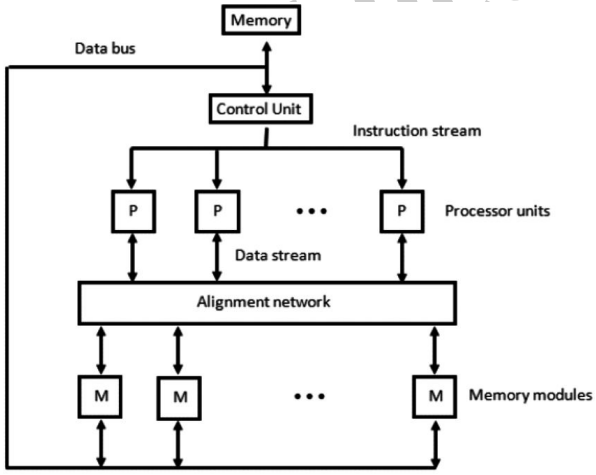


Fig. SIMD Organization

MISD:

- There is no computer at present that can be classified as MISD.
- MISD structure is only of theoretical interest since no practical system has been constructed using this organization.

MIMD:

- MIMD organization refers to a computer system capable of processing several programs at the same time.
- Most multiprocessor and multicomputer systems can be classified in this category.
- Contains multiple processing units.

- Execution of multiple instructions on multiple data.

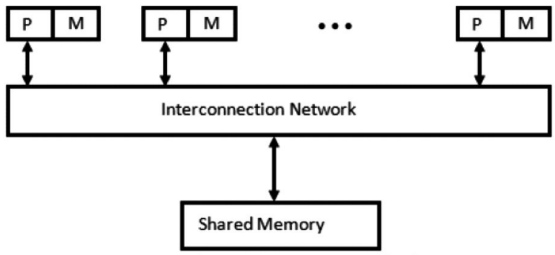


Fig. MIMD Organization

Q13. Explain pipelining technique. Draw the general structure of four segment pipeline.

Ans :

- Pipeline is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.
- A pipeline can be visualized as a collection of processing segments through which binary information flows.
- Each segment performs partial processing dictated by the way the task is partitioned.
- The result obtained from the computation in each segment is transferred to the next segment in the pipeline.
- It is characteristic of pipelines that several computations can be in progress in distinct segments at the same time.
- The overlapping of computation is made possible by associating a register with each segment in the pipeline.
- The registers provide isolation between each segment so that each can operate on distinct data simultaneously.
- Any operation that can be decomposed into a sequence of sub operations of about the same complexity can be implemented by a pipeline processor.
- The technique is efficient for those applications that need to repeat the same task many times with different sets of data.
- The general structure of a four-segment pipeline is illustrated in Figure 6.5.

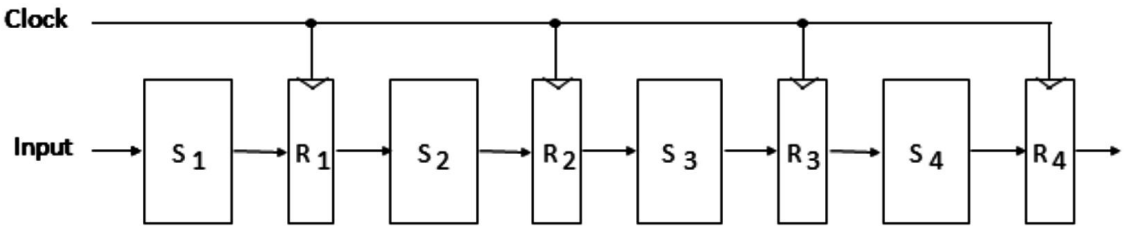


Fig. General Structure of Four-Segment Pipeline

- The operands pass through all four segments in a fixed sequence.
- Each segment consists of a combinational circuit S, which performs a sub operation over the data stream flowing through the pipe.
- The segments are separated by registers R, which hold the intermediate results between the stages.
- Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously.
- We define a task as the total operation performed going through all the segments in the pipeline.

Q14. Draw and explain Arithmetic Pipeline.

Ans :

The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers.

X = A x 2a

Y = B x 2b

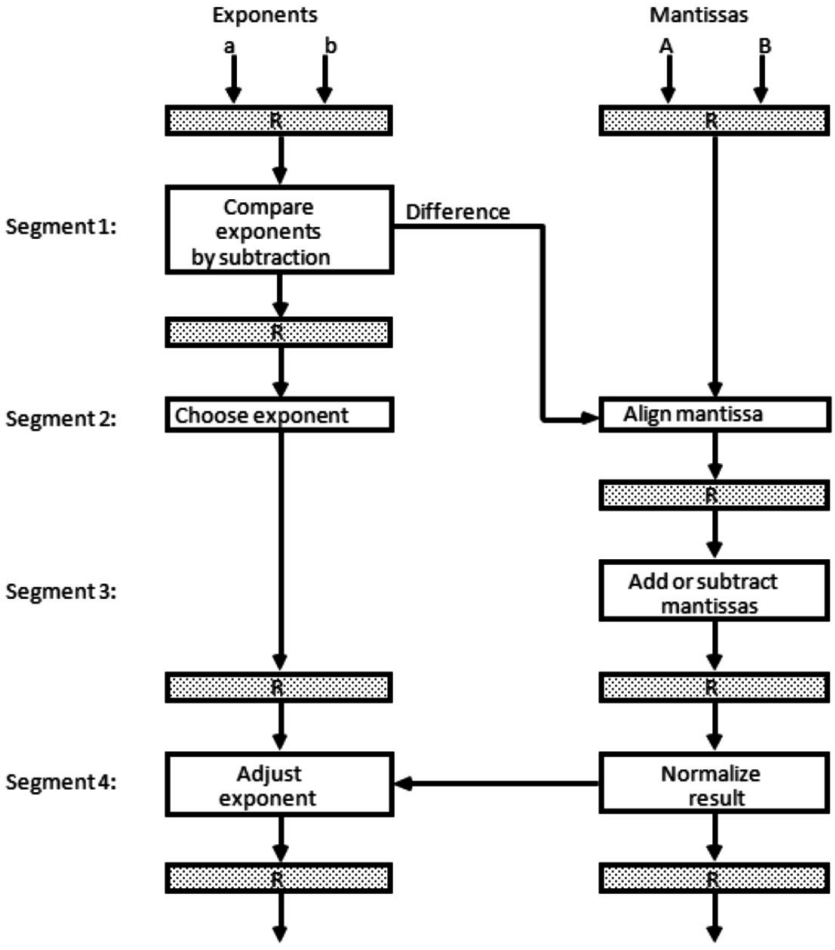


Fig. Pipeline for floating-point addition and subtraction

- A and B are two fractions that represent the mantissas and a and b are the exponents.
- The floating-point addition and subtraction can be performed in four segments, as shown in Figure
- The registers labeled R are placed between the segments to store intermediate results.
- The sub-operations that are performed in the four segments are:
  1. Compare the exponents
  2. Align the mantissas
  3. Add or subtract the mantissas
  4. Normalize the result
- The following numerical example may clarify the sub-operations performed in each segment.
- For simplicity, we use decimal numbers, although Figure 6.6 refers to binary numbers.
- Consider the two normalized floating-point numbers:

  X = 0.9504 x 103

  Y = 0.8200 x 102
- The two exponents are subtracted in the first segment to obtain 3 – 2 = 1.
- The larger exponent 3 is chosen as the exponent of the result.
- The next segment shifts the mantissa of Y to the right to obtain

  X = 0.9504 x 103

  Y = 0.0820 x 103

  This aligns the two mantissas under the same exponent. The addition of the two mantissas in segment 3 produces the sum

  Z = 1.0324 x 103

5.2.2 Instruction and RISC Pipelines

Q15. Explain the Instruction Pipelining with example.

OR

Discuss four-segment instruction pipeline with diagram(s).

Ans :

- Pipeline processing can occur in data stream as well as in instruction stream.
- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments.
- This causes the instruction fetch and executes phases to overlap and perform simultaneous operations.
- One possible digression associated with such a scheme is that an instruction may cause a branch out of sequence.

- In that case the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction must be discarded.
- Consider a computer with an instruction fetch unit and an instruction execution unit designed to provide a two-segment pipeline.
- The instruction fetch segment can be implemented by means of a first-in, first-out (FIFO) buffer.
- The buffer acts as a queue from which control then extracts the instructions for the execution unit.

Instruction Cycle

- The fetch and execute to process an instruction completely.
- In the most general case, the computer needs to process each instruction with the following sequence of steps
  1. Fetch the instruction from memory.
  2. Decode the instruction.
  3. Calculate the effective address.
  4. Fetch the operands from memory.
  5. Execute the instruction.
  6. Store the result in the proper place.
- There are certain difficulties that will prevent the instruction pipeline from operating at its maximum rate.
- Different segments may take different times to operate on the incoming information.
- Some segments are skipped for certain operations.
- The design of an instruction pipeline will be most efficient if the instruction cycle is divided into segments of equal duration.
- The time that each step takes to fulfill its function depends on the instruction and the way it is executed.

Example: Four-Segment Instruction Pipeline

- Assume that the decoding of the instruction can be combined with the calculation of the effective address into one segment.
- Assume further that most of the instructions place the result into a processor registers so that the instruction execution and storing of the result can be combined into one segment.
- This reduces the instruction pipeline into four segments.
  1. FI: Fetch an instruction from memory
  2. DA: Decode the instruction and calculate the effective address of the operand
  3. FO: Fetch the operand
  4. EX: Execute the operation

- Figure (a) shows, how the instruction cycle in the CPU can be processed with a four segment pipeline.
- While an instruction is being executed in segment 4, the next instruction in sequence is busy fetching an operand from memory in segment 3.
- The effective address may be calculated in a separate arithmetic circuit for the third instruction, and whenever the memory is available, the fourth and all subsequent instructions can be fetched and placed in an instruction FIFO.
- Thus up to four sub operations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time.

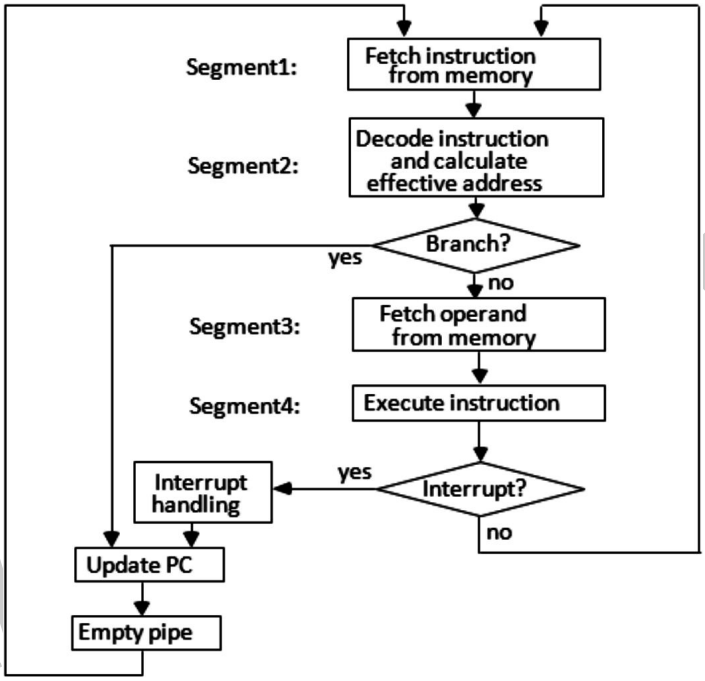


Fig (a). Four-segment CPU pipeline

Figure (b) shows the operation of the instruction pipeline. The time in the horizontal axis is divided into steps of equal duration. The four segments are represented in the diagram with an abbreviated symbol.

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	-	-	FI	DA	FO	EX			
	5					-	-	-	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Fig. Timing of Instruction Pipeline

- It is assumed that the processor has separate instruction and data memories so that the operation in FI and FO can proceed at the same time.
- Thus, in step 4, instruction 1 is being executed in segment EX; the operand for instruction 2 is being fetched in segment FO; instruction 3 is being decoded in segment
- DA; and instruction 4 is being fetched from memory in segment FI.
- Assume now that instruction 3 is a branch instruction.
- As soon as this instruction is decoded in segment DA in step 4, the transfer from FI to DA of the other instructions is halted until the branch instruction is executed in step 6.
- If the branch is taken, a new instruction is fetched in step 7. If the branch is not taken, the instruction fetched previously in step 4 can be used.
- The pipeline then continues until a new branch instruction is encountered.
- Another delay may occur in the pipeline if the EX segment needs to store the result of the operation in the data memory while the FO segment needs to fetch an operand.
- In that case, segment FO must wait until segment EX has finished its operation.

Q16. What is pipeline conflict? Explain data dependency and handling of branch instruction in detail.

Ans :

Pipeline Conflict:

There are three major difficulties that cause the instruction pipeline conflicts.

1. Resource conflicts caused by access to memory by two segments at the same time.
2. Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
3. Branch difficulties arise from branch and other instructions that change the value of PC.

Data Dependency

- A collision occurs when an instruction cannot proceed because previous instructions did not complete certain operations.
- A data dependency occurs when an instruction needs data that are not yet available.
- Similarly, an address dependency may occur when an operand address cannot be calculated because the information needed by the addressing mode is not available.
- Pipelined computers deal with such conflicts between data dependencies in a variety of ways as follows.

Hardware Interlocks

- An interlock is a circuit that detects instructions whose source operands are destinations of instructions farther up in the pipeline.
- Detection of this situation causes the instruction whose source is not available to be delayed by enough clock cycles to resolve the conflict.
- This approach maintains the program sequence by using hardware to insert the required delays.

Operand Forwarding

- It uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments.
- This method requires additional hardware paths through multiplexers as well as the circuit that detects the conflict.

Delayed Load

- Sometimes compiler has the responsibility for solving data conflicts problems.
- The compiler for such computers is designed to detect a data conflict and reorder the instructions as necessary to delay the loading of the conflicting data by inserting no operation instruction, this method is called delayed load.

Handling of Branch Instructions

- One of the major problems in operating an instruction pipeline is the occurrence of branch instructions.
- A branch instruction can be conditional or unconditional.
- The branch instruction breaks the normal sequence of the instruction stream, causing difficulties in the operation of the instruction pipeline.
- Various hardware techniques are available to minimize the performance degradation caused by instruction branching.

Pre-fetch Target

- One way of handling a conditional branch is to prefetch the target instruction in addition to the instruction following the branch.
- If the branch condition is successful, the pipeline continues from the branch target instruction.
- An extension of this procedure is to continue fetching instructions from both places until the branch decision is made.

Branch Target Buffer

- Another possibility is the use of a branch target buffer or BTB.
- The BTB is an associative memory included in the fetch segment of the pipeline.
- Each entry in the BTB consists of the address of a previously executed branch instruction and the target instruction for that branch.
- It also stores the next few instructions after the branch target instruction.
- The advantage of this scheme is that branch instructions that have occurred previously are readily available in the pipeline without interruption.

Loop Buffer

- A variation of the BTB is the loop buffer. This is a small very high speed register file maintained by the instruction fetch segment of the pipeline.
- When a program loop is detected in the program, it is stored in the loop buffer in its entirety, including all branches.

Branch Prediction

- A pipeline with branch prediction uses some additional logic to guess 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.

Delayed Branch

- A procedure employed in most RISC processors is the delayed branch.
- In this procedure, the compiler detects the branch instructions and rearranges the machine language code sequence by inserting useful instructions that keep the pipeline operating without interruptions.

5.3 Assessing and Understanding Performance

5.3.1 CPU Performance and Its Factors

Q17. What are the factors affecting the performance of the CPU ?

Ans : **(Imp.)**

1. The materials the processor is made from

Different processors, like any product, are made using different materials and different qualities of materials. There are 'good' quality materials and not-so-good ones! The materials used in a processor will affect the reliability, speed and performance of that processor. Importantly, the speed of the slowest component might slow right down an otherwise fast CPU.

2. Clock speed

Another reason why different processors perform differently is clock speed. Every computer has a 'clock'. This is a crystal that vibrates, generating 'pulses' that are used to control how the different components of a computer system work together. If you have a 800Mhz machine, it means the system clock is generating approximately 800 million pulses every second! The faster the clock, the more fetch-decode-execute cycles the CPU can perform in a second, and (very broadly) the faster your programs should go! (There are lots of other factors that may mean this is not the case, however).

3. Cores

A CPU traditionally had one 'core' but processors these days might be dual-core or quad-core, for example. A core is actually a processor with its own cache. So a dual-core CPU has not one but two processors. A quad core CPU has four processors. Two brains (or four brains) are better than one! Each brain can be working on different parts of a program at the same time and so this speeds up the overall CPU's performance.

4. Cache

Although CPUs fetch instructions from RAM, there is another place in can get instructions from, called 'cache'. Cache is just like RAM but much faster to read from and write to compared to RAM. The computer cleverly puts data into cache that it needs again and again. It is a lot quicker for the CPU to get data from cache than RAM. The more cache a computer has, the better the CPU will perform.

5. Word size

An important characteristic of a processor is its word size. This is the number of bits that the CPU can work with in any one clock cycle. The more bits it can work with in one clock cycle, the faster the computer will go. The early commercial computers like the Spectrum ZX were 8-bit computers (in another words, the processor could work on 8 bits of data at a time). Things have moved on since the early 1980s and you are likely to be using either a 32-bit computer or a 64-bit computer now. No doubt we will all be buying 128-bit computers in the near future.

5.3.2 Evaluating performance.

Q18. How, CPU performance can be evaluated. Explain

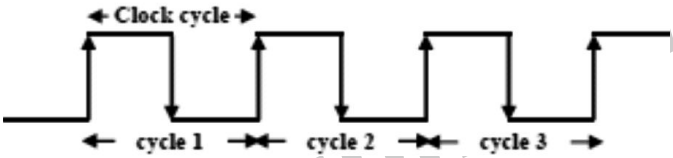
Ans :

Cycles Per Instruction (CPI)

- Most computers run synchronously utilizing a CPU clock running at a constant clock rate:

  Or clock frequency: f where: Clock rate = 1 / clock cycle

  f = 1 /C



- The CPU clock rate depends on the specific CPU organization (design) and hardware implementation technology (VLSI) used.
- A computer machine (ISA) instruction is comprised of a number of elementary or micro operations which vary in number and complexity depending on the the instruction and the exact CPU organization (Design).
- A micro operation is an elementary hardware operation that can be performed during one CPU clock cycle.
- This corresponds to one micro-instruction in microprogrammed CPUs.
- Examples: register operations: shift, load, clear, increment, ALU operations: add , subtract, etc.
- Thus: A single machine instruction may take one or more CPU cycles to complete termed as the Cycles Per Instruction (CPI).

  Instructions Per Cycle = IPC = 1/CPI

- Average (or effective) CPI of a program: The average CPI of all instructions executed in the program on a given CPU design.

  Cycles/sec = Hertz = Hz

  MHz = 106 Hz GHz = 109 Hz

Program Execution Time

For a specific program compiled to run on a specific machine (CPU) "A", has the following parameters:

- The total executed instruction count of the program $\longrightarrow$ I

The average number of cycles per instruction (average CPI) ⟶ CPI

- Clock cycle of machine "A" ⟶ C
  - How can one measure the performance of this machine (CPU) running this program?
- Intuitively the machine (or CPU) is said to be faster or has better performance running this program if the total execution time is shorter.
- Thus the inverse of the total measured program execution time is a possible performance measure or metric:

Programs/second — $\text{Performance}_A = 1 / \text{Execution Time}_A$ — Seconds/program

Comparing Computer Performance Using Execution Time

- To compare the performance of two machines (or CPUs) "A", "B" running a given specific program:

  Performance A = 1 / Execution Time A

  Performance B = 1 / Execution Time B
- Machine A is n times faster than machine B means (or slower? if n < 1) :

$$\text{Speedup} = n = \frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution Time}_B}{\text{Execution Time}_A}$$

Example:

For a given program:

Execution time on machine A: ExecutionA = 1 second

Execution time on machine B: ExecutionB = 10 seconds

Performance A / Performance B = Execution Time B / Execution Time A = 10 / 1 = 10

The performance of machine A is 10 times the performance of machine B when running this program, or: Machine A is said to be 10 times faster than machine B when running this program.

Note: The two CPUs may target different ISAs provided the program is written in a high level language (HLL)

CPU Execution Time: The CPU Equation

A program is comprised of a number of instructions executed ⟶ I

- Measured in: instructions/program
  - The average instruction executed takes a number of *cycles per instruction (CPI)* to be completed.
- Measured in: cycles/instruction, CPI

Or Instructions Per Cycle (IPC):

IPC = 1/CPI

- CPU has a fixed clock cycle time C = 1/clock rate

➢ Measured in: seconds/cycle

C = 1 / *f*

- CPU execution time is the product of the above three parameters as follows:

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

$$T = I \times CPI \times C$$

T: execution Time per program in seconds; I: Number of instructions executed; CPI: Average CPI for program; C: CPU Clock Cycle

This equation is commonly known as the CPU performance equation

CPU Average CPI/Execution Time

For a given program executed on a given machine (CPU):

$$CPI = \frac{\text{Total program execution cycles}}{\text{Instructions count Executed}}$$

(i.e average or effective CPI)

$$\rightarrow \text{CPU clock cycles} = \text{Instruction count} \times CPI$$

(executed)

$$\text{CPU execution time} = \text{CPU clock cycles} \times \text{Clock cycle}$$

$$= \text{Instruction count} \times CPI \times \text{Clock cycle}$$

$$T = I \times CPI \times C$$

T: execution Time per program in seconds; I: Number of instructions executed; CPI: Average or effective CPI for program; C: CPU Clock Cycle

CPU Execution Time: Example

A Program is running on a specific machine (CPU) with the following parameters:

➢ Total executed instruction count: 10,000,000 instructions ⟶ I

➢ Average CPI for the program: 2.5 cycles/ instruction.

➢ CPU clock rate: 200 MHz. (clock cycle = C = 5 × 10 – 9 seconds)

- What is the execution time for this program:

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

CPU time = Instruction count × CPI × Clock cycle

T = I × CPI × C

= 10,000,000 × 2.5 × 1 / clock rate

= 10,000,000 × 2.5 × 5 × 10 – 9

= 0.125 seconds

FACULTY OF INFORMATICS

M.C.A. I Year I Semester Examination

Model Paper - I

COMPUTER ARCHITECTURE

Time : 3 Hours] Max. Marks : 70

Answer all the question according to the internal choice **(5 × 14 = 70)**

ANSWERS

1. (a) What are the various number systems used in Computer? **(Unit-I, Q.No. 2)**
   (b) Explain about the structure of typical bus. **(Unit-I, Q.No. 10)**

(OR)

2. (a) Explain about fixed and floating point representation of numbers. **(Unit-I, Q.No. 5)**
   (b) What is the complement number system ? Explain the complement systems with examples. **(Unit-I, Q.No. 4)**
3. (a) Design and explain a common bus system for four register. **(Unit-II, Q.No. 3)**
   (b) A digital computer has a common bus system for 16 registers of 32 bits each. (i) How many selection input are there in each multiplexer? (ii) What size of multiplexers is needed? (iii) How many multiplexers are there in a bus? **(Unit-II, Q.No. 4)**

(OR)

4. (a) Draw block diagram of 4-bit arithmetic circuit and explain it in detail. **(Unit-II, Q.No. 11)**
   (b) List and explain register reference instruction. and memory reference instructions. **(Unit-II, Q.No. 24, 25)**
5. (a) What is stack? Give the organization of register stack with all necessary elements and explain the working of push and pop operations. **(Unit-III, Q.No. 10)**
   (b) Draw the diagram of Micro programmed sequencer for a control memory and explain it. **(Unit-III, Q.No. 7)**

(OR)

6. (a) Explain four types of instruction formats. **(Unit-III, Q.No. 12)**
   (b) (i) Explain the procedure for Addition and Subtraction with signed-magnitude data with the help of flowchart. **(Unit-III, Q.No. 19)**
   (ii) Multiply the (-9) with (-13) using Booth's algorithm. Give each step. **(Unit-III, Q.No. 22)**
7. (a) How main memory is useful in computer system? **(Unit-IV, Q.No. 2)**
   (b) Discuss set-associative mapping in organization of cache memory. **(Unit-IV, Q.No. 8)**

(OR)

8. (a) Explain briefly about auxiliary memory. **(Unit-IV, Q.No. 4)**
   (b) What is segment? What is logical address? Explain segmented page mapping. **(Unit-IV, Q.No. 12)**
9. (a) Define Peripherals. Explain I/O Bus and Interface Modules. **(Unit-V, Q.No. 1)**
   (b) What is pipeline conflict? Explain data dependency and handling of branch instruction in detail. **(Unit-V, Q.No. 16)**

(OR)

10. (a) What do you mean by Asynchronous data transfer? Explain Strobe control in detail. **(Unit-V, Q.No. 3)**
    (b) Explain Input- Output Processor (IOP). **(Unit-V, Q.No. 9)**

FACULTY OF INFORMATICS

M.C.A. I Year I Semester Examination

Model Paper - II

COMPUTER ARCHITECTURE

Time 3 Hrs] [Max Marks : 70

Answer all the question according to the internal choice **(5 × 14 = 70)**

ANSWERS

1. (a) Describe how data is stored in the digital computers. **(Unit-I, Q.No. 1)**
   (b) Explain about the interconnection structure of the computer. **(Unit-I, Q.No. 8)**

(OR)

2. (a) Write about various types of binary codes. **(Unit-I, Q.No. 6)**
   (b) Explain the basic components of a computer. **(Unit-I, Q.No. 7)**
3. (a) Explain the Register Transfer Language. **(Unit-II, Q.No. 1)**
   (b) Explain shift micro operations and draw 4-bit combinational circuit shifter. **(Unit-II, Q.No. 14)**

(OR)

4. (a) Explain Direct and Indirect addressing of basic computer. **(Unit-II, Q.No. 18)**
   (b) Explain Instruction Format with its types. **(Unit-II, Q.No. 21)**
5. (a) Explain Registers of basic computer. **(Unit-III, Q.No. 19)**
   (b) Explain the Booth's algorithm with the help of flowchart. Multiply the (7) with (3) using Booth's algorithm. Give each step. **(Unit-III, Q.No. 20, 23)**

(OR)

6. (a) Write about the representation of floating point numbers. **(Unit-III, Q.No. 27)**
   (b) Explain Program Interrupts. Explain clearly, discussing the role of stack, PSW and return from interrupt instruction, how interrupts are implemented on computers. **(Unit-III, Q.No. 17)**
7. (a) Explain the memory address map of RAM and ROM. **(Unit-IV, Q.No. 3)**
   (b) Discuss direct mapping in organization of cache memory. **(Unit-IV, Q.No. 7)**

(OR)

8. (a) Explain Content Addressable Memory (CAM). **(Unit-IV, Q.No. 5)**
   (b) Explain Write-through and Write-back cache write method. **(Unit-IV, Q.No. 9)**
9. (a) Draw and explain Arithmetic Pipeline. **(Unit-V, Q.No. 13)**
   (b) Explain pipelining technique. Draw the general structure of four segment pipeline. **(Unit-V, Q.No. 13)**

(OR)

10. (a) Explain I/O interface with example. **(Unit-V, Q.No. 2)**
    (b) What are the factors affecting the performance of the CPU. **(Unit-V, Q.No. 17)**

FACULTY OF INFORMATICS

M.C.A. I Year I Semester Examination

Model Paper - III

COMPUTER ARCHITECTURE

Time 3 Hrs] [Max Marks : 70

Answer all the question according to the internal choice **(5 × 14 = 70)**

ANSWERS

1. (a) Write about other coding schemes in number system. **(Unit-I, Q.No. 3)**
   (b) Explain three modes of data transfer between I/O devices and a computer system. **(Unit-I, Q.No. 11)**

(OR)

2. (a) What are the various number systems used in Computer? **(Unit-I, Q.No. 2)**
   (b) What is the use of bus interconnection? Explain the types of bus organization. **(Unit-I, Q.No. 9)**
3. (a) Explain Registers of basic computer. **(Unit-II, Q.No. 19)**
   (b) Draw and explain the flowchart for instruction cycle. **(Unit-II, Q.No. 23)**

(OR)

4. (a) Explain the basic working principle of the Control Unit with timing diagram. **(Unit-II, Q.No. 22)**
   (b) What is an Interrupt Cycle? Draw and explain flow chart of it. **(Unit-II, Q.No. 28)**
5. (a) Explain Microinstruction Format in detail. Explain Symbolic Microinstruction. **(Unit-III, Q.No. 5,6)**
   (b) (i) Explain with proper block diagram the Multiplication Operation on two floating point numbers. **(Unit-III, Q.No. 21)**
   (ii) Multiply the (15) with (13) using Booth's algorithm. Give each step. **(Unit-III, Q.No. 24)**

(OR)

6. (a) What are status register bits? Draw and explain the block diagram showing all status registers. **(Unit-III, Q.No. 16)**
   (b) Explain about general register organization. **(Unit-III, Q.No. 9)**
7. (a) Define Cache memory. Discuss associative mapping in organization of cache memory. **(Unit-IV, Q.No. 6)**
   (b) Write about memory management hardware. **(Unit-IV, Q.No. 13)**

(OR)

8. (a) Explain address mapping using pages. **(Unit-IV, Q.No. 11)**

(b) What do you mean by address space and memory space in virtual memory? Also explain the relation between address space and memory space in virtual memory. **(Unit-IV, Q.No. 10)**

9. (a) Write a detailed note on Direct Memory Access (DMA). **(Unit-V, Q.No. 8)**

(b) Write a note on Interrupt Initiated I/O. **(Unit-V, Q.No. 6)**

(OR)

10. (a) What is priority interrupt? Explain Daisy Chaining. **(Unit-V, Q.No. 7)**

(b) Explain Flynn's classification for computers. **(Unit-V, Q.No. 12)**

FACULTY OF INFORMATICS

M.C.A (CBCS) I - Semester Examination

August - 2021

COMPUTER ARCHITECTURE

Time : 2 Hours] **[Max. Marks : 70**

PART - A - ($4 \times 17^{1/2} = 70$ Marks)

Note : Answer any four Questions.

			ANSWERS
1.	(a)	Explain the procedure for subtractionof unsigned numbers with an example of binary and decimal numbers.	**(Unit-I, Q.No. 4)**
	(b)	Elaborate on interrupts and its cycle.	**(Unit-I, Q.No. 11)**
2.	(a)	Explain the gray code.	**(Unit-I, Q.No. 6)**
	(b)	Explain about bus interconnection and its types.	**(Unit-I, Q.No. 9)**
3.	(a)	Construct a bus line with three state buffers.	**(Unit-II, Q.No. 5)**
	(b)	With an example, explain BSA instruction execution.	**(Unit-II, Q.No. 25)**
4.	(a)	Construct 4-bit combinational circuit shifter and explain its function table	**(Unit-II, Q.No. 14)**
	(b)	Demonstrate the illustration of direct — indirect address.	**(Unit-II, Q.No. 18)**
5.	(a)	Explain about the symbolic microprogram.	**(Unit-III, Q.No. 6)**
	(b)	What are the three types of CPU organizations ? Explain each with an example.	**(Unit-II, Q.No. 8)**
6.	(a)	Write and explain the flowchart for the selection of address for control memory.	**(April/May-2023, Q.No. 5(b))**
	(b)	Explain the flowchart for add-subtract operation done in hardware.	**(Unit-III, Q.No. 19)**
7.	(a)	Illustrate memory connection to CPU.	**(Unit-IV, Q.No. 2)**
	(b)	Explain memory table for mapping an initial address.	**(Unit-IV, Q.No. 11)**
8.	(a)	Illustrate direct mapping with an example.	**(Unit-IV, Q.No. 7)**
	(b)	Describe segmented - page mapping with a figure.	**(Unit-IV, Q.No. 12)**
9.	(a)	Explain Strore control method of asynchronous data transfer	**(Unit-V, Q.No. 3)**
	(b)	Illustrate the use of DMA	**(Unit-V, Q.No. 8)**
10.	(a)	Elaborate on the steps for instruction pipeline.	**(Unit-V, Q.No. 15)**
	(b)	Explain the factors of CPU performance.	**(Unit-V, Q.No. 17)**

FACULTY OF INFORMATICS

MCA I - Semester (CBCS) (Main & Backlog) Examinations

April / May - 2023

COMPUTER ARCHITECTURE

Time : 3 Hours] **[Max. Marks : 70**

Note : I. Answer one questions from each unit. All questions carry equal marks.

II. Missing data, if any, may be suitably assumed.

ANSWERS

Unit - I

1. a) Illustrate the procedures with examples for conversion of Binary to Octal, Decimal and Hexa Decimal. **(Unit-I, Q.No. 2)**

b) Explain about the instruction cycle with a program execution illustration.

Ans:

A program consisting of the memory unit of the computer includes a series of instructions. The program is implemented on the computer by going through a cycle for each instruction.

In the basic computer, each instruction cycle includes the following procedures -

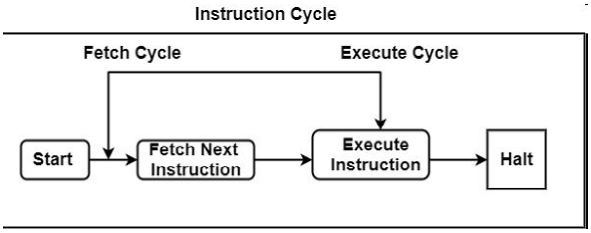
i) It can fetch instruction from memory.

ii) It is used to decode the instruction.

iii) It can read the effective address from memory if the instruction has an indirect address.

iv) It can execute the instruction.

The control switches back to the first step and repeats the similar process for the next instruction. Therefore, the cycle continues until a Halt condition is met. The figure shows the phases contained in the instruction cycle.



As display in the figure, the halt condition appears when the device receive turned off, on the circumstance of unrecoverable errors, etc.

Fetch Cycle

The address instruction to be implemented is held at the program counter. The processor fetches the instruction from the memory that is pointed by the PC.

Next, the PC is incremented to display the address of the next instruction. This instruction is loaded onto the instruction register. The processor reads the instruction and executes the important procedures.

Execute Cycle

The data transfer for implementation takes place in two methods are as follows -

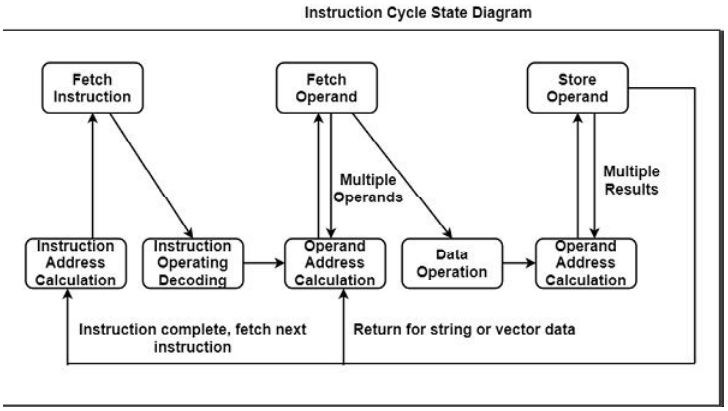
i) **Processor-memory** - The data sent from the processor to memory or from memory to processor.

ii) **Processor-Input/Output** - The data can be transferred to or from a peripheral device by the transfer between a processor and an I/O device.

iii) **In the execute cycle,** the processor implements the important operations on the information, and consistently the control calls for the modification in the sequence of data implementation. These two methods associate and complete the execute cycle.

State Diagram for Instruction Cycle

The figure provides a large aspect of the instruction cycle of a basic computer, which is in the design of a state diagram. For an instruction cycle, various states can be null, while others can be visited more than once.



i) **Instruction Address Calculation:** The address of the next instruction is computed. A permanent number is inserted to the address of the earlier instruction.

ii) **Instruction Fetch:** The instruction is read from its specific memory location to the processor.

iii) **Instruction Operation Decoding:** The instruction is interpreted and the type of operation to be implemented and the operand(s) to be used are decided.

iv) **Operand Address Calculation:** The address of the operand is evaluated if it has a reference to an operand in memory or is applicable through the Input/Output.

v) **Operand Fetch:** The operand is read from the memory or the I/O.

vi) **Data Operation:** The actual operation that the instruction contains is executed.

vii) **Store Operands:** It can store the result acquired in the memory or transfer it to the I/O.

(OR)

2. a) Discuss about floating point representation with illustrations **(Unit-I, Q.No. 5)**

   b) Explain about bus interconnection and its types. **(Unit-I, Q.No. 9)**

Unit - II

3. a) Write notes on how Register transfer takes place, **(Unit-II, Q.No. 2)**

   b) Describe three instruction code formats. **(Unit-II, Q.No. 5)**

(OR)

4. a) Construct a 4-bit adder - subtractor. **(Unit-II, Q.No. 9)**

   b) Explain the flowchart for the fetch phase. **(Unit-II, Q.No. 23)**

Unit - III

5. a) What are the three types of CPU organizations? Explain each with an example. **(Unit-III, Q.No. 8)**

b) Write and explain the flowchart for the selection of address for control memory.

Ans:

The control memory is used to store the microinstructions in groups. Here each group is used to specify a routine. The control memory of each computer has the instructions which contain their micro-programs routine. These micro-programs are used to generate the micro-operations that will be used to execute the instructions. Suppose the address sequencing of control memory is controlled by the hardware. In that case, that hardware must be capable to branch from one routine to another routine and also able to apply sequencing of microinstructions within a routine. When we try to execute a single instruction of computer, the control must undergo the following steps:

i) When the power of a computer is turned on, we have to first load an initial address into the CAR (control address register). This address can be described as the first microinstruction address. With the help of this address, we are able to activate the instruction fetch routine.

ii) Then, the control memory will go through the routine, which will be used to find out the effective address of operand.

iii) In the next step, a micro-operation will be generated, which will be used to execute the instruction fetched from memory.

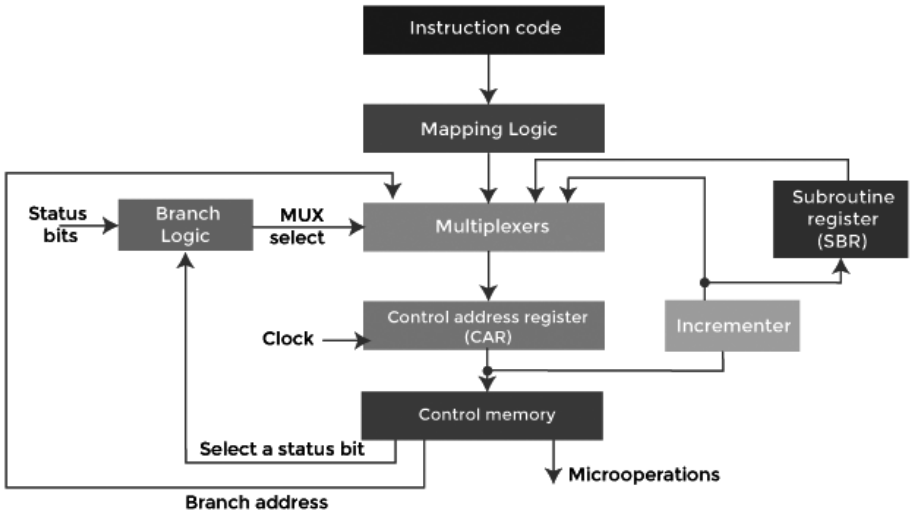
iv) We are able to transform the bits of instruction code into an address with the help of control memory where routine is located. This process can be called the mapping process. The control memory required the capabilities of address sequencing, which is described as follows

v) On the basis of the status bit conditions, the address sequencing selects the conditional branch or unconditional branch.

vi) Addressing sequence is able to increment the CAR (Control address register).

vii) It provides the facility for subroutine calls and returns.

viii) A mappings process is provided by the addressing sequence from the instructions bits to a control memory address.



Selection of Address for Control Memory

In the above diagram, we can see a block diagram of a control memory and associative hardware, which is required for selecting the address of next microinstruction. The microinstruction is used to contain a set of bits in the control memory. With the help of some bits, we are able to start the micro-operations in a computer register. The remaining bits of microinstruction are used to specify the method by which we are able to obtain the next address.

(OR)

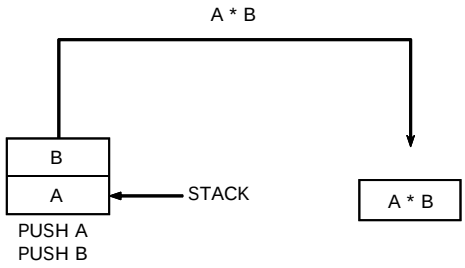
6. a) Evaluate the arithmetic statement X = (A+B) * (C+D) using Three-address instruction, Two- address instruction, One-address instruction, Zero-address instruction, RISC instruction.

Ans:

Based on the number of addresses, instructions are classified as:

i) Zero Address Instructions

These instructions do not specify any operands or addresses. Instead, they operate on data stored in registers or memory locations implicitly defined by the instruction. For example, a zero-address instruction might simply add the contents of two registers together without specifying the register names.



A stack-based computer does not use the address field in the instruction. To evaluate an expression first it is converted to reverse Polish Notation i.e. Postfix Notation.

Expression: X = (A+B)*(C+D)

Postfixed : X = AB+CD+*

TOP means top of stack

M[X] is any memory location

PUSH	A	TOP = A
PUSH	B	TOP = B
ADD		TOP = A+B
PUSH	C	TOP = C
PUSH	D	TOP = D
ADD		TOP = C+D
MUL		TOP = (C+D)*(A+B)
POP	X	M[X] = TOP

ii) One Address Instructions

These instructions specify one operand or address, which typically refers to a memory location or register. The instruction operates on the contents of that operand, and the result may be stored in the same or a different location. For example, a one-address instruction might load the contents of a memory location into a register.

This uses an implied ACCUMULATOR register for data manipulation. One operand is in the accumulator and the other is in the register or memory location. Implied means that the CPU already knows that one operand is in the accumulator so there is no need to specify it.

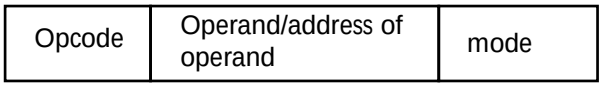


Fig.: One Address Instruction

Expression: X = (A+B)*(C+D)

AC is accumulator

M[] is any memory location

M[T] is temporary location

```
LOAD    A    AC = M[A]
ADD B        AC = AC + M[B]
STORE   T    M[T] = AC
LOAD    C    AC = M[C]
ADD D        AC = AC + M[D]
MUL T        AC = AC * M[T]
STORE   X    M[X] = AC
```

iii) Two Address Instructions

These instructions specify two operands or addresses, which may be memory locations or registers. The instruction operates on the contents of both operands, and the result may be stored in the same or a different location. For example, a two-address instruction might add the contents of two registers together and store the result in one of the registers.

This is common in commercial computers. Here two addresses can be specified in the instruction. Unlike earlier in one address instruction, the result was stored in the accumulator, here the result can be stored at different locations rather than just accumulators, but require more number of bit to represent the address.

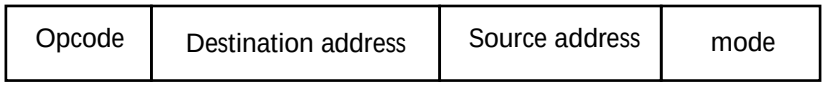


Fig. Two Address Instruction

Here destination address can also contain an operand.

Expression: X = (A+B)*(C+D)

R1, R2 are registers

M[] is any memory location

```
MOV     R1, A     R1 = M[A]
ADD     R1, B     R1 = R1 + M[B]
MOV     R2, C     R2 = M[C]
ADD     R2, D     R2 = R2 + M[D]
MUL     R1, R2    R1 = R1 * R2
MOV     X, R1     M[X] = R1
```

iv) Three Address Instructions

These instructions specify three operands or addresses, which may be memory locations or registers. The instruction operates on the contents of all three operands, and the result may be stored in the same or a different location. For example, a three-address instruction might multiply the contents of two registers together and add the contents of a third register, storing the result in a fourth register.

This has three address fields to specify a register or a memory location. Programs created are much short in size but number of bits per instruction increases. These instructions make the creation of the program much easier but it does not mean that program will run much faster because now instructions only contain more information but each micro-operation (changing the content of the register, loading address in the address bus etc.) will be performed in one cycle only.

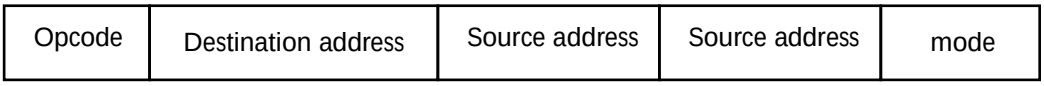


Fig. Three Address Instruction

Expression: X = (A+B)*(C+D)

R1, R2 are registers

M[] is any memory location

ADD R1, A, B R1 = M[A] + M[B]

ADD R2, C, D R2 = M[C] + M[D]

MUL X, R1, R2 M[X] = R1 * R2

b) Describe the block diagram of a 64-word stack. **(Unit-III, Q.No. 10)**

Unit - IV

7. a) Demonstrate associative memory with example. **(Unit-IV, Q.No. 5)**

   b) Illustrate set-associative mapping with an example. **(Unit-IV, Q.No. 8)**

(OR)

8. a) Illustrate direct mapping with an example. **(Unit-IV, Q.No. 7)**

   b) Explain segmentation with a numerical example. **(Unit-IV, Q.No. 12)**

Unit - V

9. a) Discuss source and destination initiated asynchronous transfer using handshaking. **(Unit-V, Q.No. 4)**

   b) With a figure, explain DMA transfer. **(Unit-V, Q.No. 8)**

(OR)

10. a) Explain programmed I/O with an example. **(Unit-V, Q.No. 9)**

    b) Write notes on RISC pipeline. **(Unit-V, Q.No. 15)**

FACULTY OF INFORMATICS

MCA I - Semester (CBCS) (Main & Backlog) Examinations

October / November - 2023

COMPUTER ARCHITECTURE

Time : 3 Hours] **[Max. Marks : 70**

Note : I. Answer one questions from each unit. All questions carry equal marks.

II. Missing data, if any, may be suitably assumed.

ANSWERS

Unit - I

1. a) Explain the procedure to perform subtraction of unsigned numbers. **(Unit-I, Q.No. 4)**
   b) Illustrate the steps to obtain 1's and 2's complement. **(Unit-I, Q.No. 4)**

(OR)

2. a) Describe the process through which data is transferred. **(Unit-I, Q.No. 11)**
   b) Write the sequence of gray codes. **(Unit-I, Q.No. 6)**

Unit-II

3. a) Construct a binary adder for 4 bits. **(Unit-II, Q.No. 8)**
   b) Illustrate direct and indirect addresses. **(Unit-II, Q.No. 18)**

(OR)

4. a) Draw and explain the circuit for 4 registers bus system. **(Unit-II, Q.No. 3)**
   b) Explain the steps of instruction cycle. **(Unit-II, Q.No. 23)**

Unit-III

5. a) Write about the various addressing modes. **(Unit-III, Q.No. 13)**
   b) Explain stack organization in memory. **(Unit-III, Q.No. 10)**

(OR)

6. a) Discuss about microprogramed control organization. **(Unit-III, Q.No. 3)**
   b) Elaborate the format of microinstruction. **(Unit-III, Q.No. 5)**

Unit-IV

7. a) Describe the hierarchy of memory. **(Unit-IV, Q.No. 1)**
   b) Explain about the auxiliary memory. **(Unit-IV, Q.No. 4)**

(OR)

8. a) Discuss the working of associative memory. **(Unit-IV, Q.No. 5)**
   b) Explain the process of mapping addresses using pages. **(Unit-IV, Q.No. 11)**

Unit - V

9. a) Differentiate between source and destination initiated strobe for data transfer. **(Unit-V, Q.No. 3)**
   b) Explain the steps of DMA. **(Unit-V, Q.No. 8)**

(OR)

10. a) With the help of an example, describe arithmetic pipeline. **(Unit-V, Q.No. 14)**
    b) Discuss the steps of interrupt initiated I/O mode of transfer. **(Unit-V, Q.No. 2)**