CSE 2nd Year KCS302 (Computer Organization) Question Bank (Unitwise)

Q.No.	Question Description	Course Outcome	Marks	Difficulty Level
1	Describe the digital computer generation in brief.	CO1	2	Low
2	Define bus arbitration with suitable diagram.	CO1	5	Medium
3	Explain different functions of a computer system.	CO1	5	Medium
4	Write short note on the following: (i)Common Bus system (ii)Bus Arbitration	CO1	5	Medium
5	Draw a diagram of a bus system for four registers that uses three state buffer and a decoder instead of the multiplexers	CO1	5	High
6	Explain ALU (Arithmetic Logic Unit) ? Draw logic diagram of ALU that performs AND, OR logic operations and ADD, SUB arithmetic operations	CO1	10	Medium
7	Compute the following arithmetic operations in 8-bit registers. Use signed 2's complement notation. Indicate overflow/underflow, if any :— (a) - 28 - (- 100) (b) - 28 - 100 (c) 78 - (- 49) (d) +50-5	CO1	10	Medium
8	Elaborate the following conditional control statement by two register transfer statements with control functions.If (P = 1) then (Rl<-R2) else if (Q = l) then(Rl^R3) 11.A computer has 16 registers, an ALU with	CO1	5	Medium
9	Explain various addressing modes with suitable examples.	CO1	10	Medium
10	Describe stack organization ? Compare Register stack and Memory stack.	CO1	10	Medium
11	Describe use of registers in computer? Explain general Register organization with multiple bus	CO1	5	High
12	Explain the need of having many addressing modes in your machine ?Discuss Indirect and displacement addressing in detail.	CO1	5	Medium
13	The following statements specify a memory. Explain the memory operation in each case (a)R2←M[AR] (b) M[AR]←R3 (c) R5 ←M[R5]	CO1	5	Medium
14	An 8-bit register contains the binary value 10011100.Calculate the register value after an arithmetic shift right? Starting from the initial number 10011100, determine the register value after the arithmetic shift left and state whether, there is an overflow.	CO1	2	Medium

CSE 2nd Year KCS302 (Computer Organization) Question Bank (Unitwise)

15	Review about wrong with the following register transfer statements? a) xT:AR←AR,AR←0 b) yT:R1←R2,R1←R3	CO1	2	Medium
16	Write a program to evaluate arithmetic expressions using three, zero address instruction: X= (A-B)+C*(D*E-F)/G+H*K	CO1	10	High
17	The memory unit of a computer has 256 words of 32 bits each .the computer has an instruction format with four fields : an operation code, a mode field to specify one of seven addressing modes, a register address field to specify one of 60 processor registers and a memory address. state the instruction format and the number of bits in each field if the instruction in one memory word.	CO1	10	High
18	An instruction is stored at location 300 with its address field at location 301. The address field has the value 400. A processor register R1 contains the number 200.Evaluate the effective address if the addressing mode of the instruction is a) direct b)immediate c) relative d) register indirect e) index with R1 as the index register.	CO1	10	Medium
19	Evaluate the no. of address and data line in a main memory of size 64K*16	CO1	2	Low
20	A digital computer has a common bus system for 32 registers of 16 bits each. The bus is constructed with multiplexers. How many selection inputs are there in each multiplexer? How many multiplexers are there in the bus? What is the size of each MUX.	CO1	5	Medium
21	Explain Booth's Algorithm in detail with flow chart. Implement multiplication of two operands (-12) and (+4) with the help of Booth's algorithm.	CO2	10	High
22	Explain Unsigned multiplication algorithm with flow chart. Implement multiplication of 12 and 4 using Unsigned multiplication algorithm	CO2	10	Medium
23	Design a 3*3 bit array multiplier with a suitable diagram	CO2	5	Medium
24	Discuss IEEE standard for floating point numbers.	CO2	2	Medium
25	Discuss and Differentiate between fixed point representation and floating point representation. Explain with suitable examples	CO2	5	Medium
26	Explain the following with reference to floating point representation: — (i)IEEE standard (ii) Normalization (iii) Biasing (iv) Overflow & Underflow	CO2	10	Low
27	State Booth Algorithm for multiplication of two numbers. Draw a logic diagram for the implementation of the Booth Algorithm for determining the product of two 8-bit signed numbers	CO2	10	Medium
28	State the division of the following positive number 1000/11. using 4-bit register.	CO2	10	High
29	Design the carry look ahead adder.	CO2	10	Medium
30	Design carry look ahead generator.	CO2	5	High

CSE 2nd Year KCS302 (Computer Organization) Question Bank (Unitwise)

31	Explain floating point addition and subtraction algorithm.	CO2	10	High
32	Define Floating point overflow and underflow conditions.	CO2	2	Medium
33	Explain the various steps required to design ALU.	CO2	10	High
34	Explain steps to divide two signed numbers.	CO2	5	Medium
35	Explain the process of dividing +7/-3 using restoring method.	CO2	5	High
36	Explain signed multiplication algorithm with the help of flow chart. Implement multiplication of (+12)*(-4) using Unsigned multiplication algorithm	CO2	10	High
37	Design a 4 bit arithmetic circuit using a suitable block diagram.	CO2	5	Medium
38	Design a 4 bit Bidirectional shifter with the help of multiplexer.	CO2	5	Medium
39	Explain divide overflow.	CO2	2	Medium
40	Describe carry look ahead in digital circuits.	CO2	2	Low
41	Define CISC ? Explain it with its characteristics.	CO3	5	High
42	A computer has 16 registers, an ALU with 32 operations, and a shifter with eight operations, all connected to a common bus system . (i) Formulate a Control Word for a micro-operation. (ii) Specify die number of bits in each field of the control word and give a general encoding scheme. (iii) Show the bits of the control word that specify the micro-operation R4 <- R5 + R6.	CO3	10	High
43	Define microoperation? How can microoperation be used for execution of an instruction? Explain with the help of an example.	CO3	5	Medium
44	Instruction cycle is divided into subcycles. With the help of diagram, explain the sequence in which subcycles are executed	CO3	5	Medium
45	Discuss how address sequencing is carried out in microprogrammed organization.	CO3	5	High
46	Describe how control unit of a computer function? Explain with the help of a block diagram	CO3	10	Medium
47	Define the following : (i) Microoperation (ii) Microinstruction (iii) Microprogram (iv) Microcode	CO3	2	Medium
48	Describe the basic differences between a branch instruction, a call subroutine instruction, and program interrupt?	CO3	5	Medium
49	Describe the differences between CISC & RISC architecture.	CO3	5	Medium
50	State the meaning of hard wired control unit? Give various methods to design hardwired control unit. Describe one of the design methods for hardwired control unit with suitable diagrams.	CO3	10	High

CSE 2nd Year KCS302 (Computer Organization) Question Bank (Unitwise)

51	Explain wide-branch addressing? Explain with example.	CO3	5	High
52	Define Microinstruction ? How is it different from microprogram sequencer ? Explain with the help of example.	CO3	5	Medium
53	Describe how a processor execute instructions? Define the internal functional units of a processor and how they are interconnected	CO3	10	Medium
54	A system uses a control memory of 1024 words of 32 bits each..The micro-instruction has three fields:select, address and microoperation fields. The microoperation's field has 16 bits. (a) How many bits are there in the branch address field and select field?	CO3	10	Medium
55	Using the mapping procedure, determine the first micro-instruction address for the following operation code: a)0010 b)1011 c)1111	CO3	5	High
56	Describe the difference between a microprocessor and a microprogram? Is it possible to design a microprocessor without a microprogram? Explain.	CO3	5	High
57	Define interrupt? Explain various types of interrupt.	CO3	5	High
58	Explain the hardwired control unit organization explaining each component clearly	CO3	10	Medium
59	Explain advantages of pipelining with a suitable example.	CO3	5	High
60	Define program control. Explain its types.	CO3	5	Medium
61	Write short notes on any two of the following : (i) Auxiliary memory (ii) Memory Hierarchy (iii) Virtual Memory (iv) Cache Memory	CO4	5	Medium
62	Define semiconductor RAM memories? Show the read operation and write operation in static memories with examples	CO4	5	Medium
63	Explain the concept of Virtual memory. How address mapping is performed in virtual memory ?	CO4	5	High
64	Define the difference between 2D and 2]2 D memory organization ? Explain it with the help of suitable examples	CO4	5	Medium
65	A ROM chip of 1024 x 8 bits has four select inputs and operates from a 5-volt power supply. How many pins are needed for the IC package? Draw a block diagram and label all inputs and output terminals in the ROM.	CO4	2	Low
66	A two-way set associative cache memory uses blocks of four words. The cache can accommodate a total of 2048 words from the main memory. The main memory size is 128 K*32. (i) Formulate all pertinent information required to construct the cache	CO4	10	High

	memory. (ii) What is the size of the cache memory			
67	Define the terms address space and memory space. An address space is specified by 24 bits and the corresponding memory space by 16 bits. Find the following : (i) How many words are there in the address space? (ii) How many words are there in the memory space	CO4	5	Medium
68	Discuss the various organization of RAM. A computer uses RAM chips of 1024x1 capacity. How many chips are needed and how should their address lines be connected to provide a memory capacity of 1024 bytes ?	CO4	5	Medium
69	Explain various cache mapping techniques. A computer system has a 4k word cache organised in block set associative manner with 4 blocks per set, 64 words per block. The main memory contains 65536 blocks. How many bits are there in each of the TAG, SET & WORD fields ?	CO4	10	Medium
70	Describe the difference between Direct mapping and Associative mapping procedures for organisation of cache memory with example. Give merits and demerits of both mapping procedures	CO4	10	High
71	Evaluate how many 128 bytes RAM chips are required to provide a memory of 2048 bytes ? Show details of connection clearly indicating address, data and decoder configuration	CO4	5	High
72	A virtual memory system has a page size of 1K words.There are eight pages and four blocks. The associative memory page table contains the following entries: Page Block 0 3 1 1 4 2 6 0 Make a list of all virtual address (in decimal) that will cause a page fault, if used by the processor.	CO4	5	Medium
73	The logical address space in a computer system consists of 129 segments. Each segment can have up to 32 pages of 4K word in each.Physical memory consists of 4K Blocks of 4K words in each. Formulate the logical and physical address formats.	CO4	5	Medium
74	computer employs RAM chips of 1024*8 and ROM chips of 512*8.The computer system needs 2KB of RAM 4KB of ROM and four interface units, each with four registers.,A memory mapped I/O configuration is used The two highest order bits of the address bus assigned 00 for RAM,01 for ROM and 10 for interface registers.. a). determine RAM and ROM chips are needed? b) .Draw a memory-address map for the system	CO4	10	High
75	Describe memory hierarchy	CO4	2	Low
76	Explain various types of ROM ?	CO4	5	Medium

CSE 2nd Year KCS302 (Computer Organization) Question Bank (Unitwise)

77	Write the difference between SRAM & DRAM?	CO4	2	Low
78	Define locality of reference. Explain with an example?	CO4	5	Medium
79	Describe Cache memory. How it is constructed.	CO4	2	Medium
80	Describe why cache memory mapping is best and why?	CO4	2	Medium
81	Explain Direct Memory Access (DMA).	CO5	5	Medium
82	Explain Synchronous and Asynchronous communication.	CO5	5	Medium
83	Define Interrupts with their types and exceptions	CO5	5	High
84	Write short note on the following together with their importance: (i) Input-output processor (ii) Serial Communication.	CO5	5	Medium
85	Describe why Input-Output interface is required? Describe various methods for I/O interface together with their merits and demerits	CO5	10	Medium
86	Define what programming steps are required to check when a source interrupts the computer while it is still serviced by a previous interrupt request from the same source	CO5	5	High
87	Draw a flow chart that describes the CPU-I/O channel communication in the IBM 370.	CO5	10	Medium
88	Describe what is basic advantage of using interrupt initiated data transfer over transfer under program control without an interrupt ?	CO5	5	Medium
89	Describe Strobe control and Hand shaking for Asynchronous Data Transfer.	CO5	10	Medium
90	Define programmed I/O.	CO5	2	Medium
91	Explain various types of peripheral devices with a suitable example.	CO5	5	Low
92	Determine the different modes of data transfer. Explain any one of them with a suitable example.	CO5	5	Medium
93	With the help of block diagram, explain working of Direct Memory Access (DMA). Give a brief comparison of programmed I/O & Interrupt I/O.	CO5	10	Medium
94	Explain strobe methods with suitable diagram.	CO5	5	High
95	Explain standard communication interface with a suitable diagram.	CO5	5	Medium
96	Describe the advantage of using input output interface ?	CO5	2	Medium
97	Write the difference between I/O Channel and I/O processors.	CO5	5	Medium
98	Explain the difference between external and internal interrupt.	CO5	5	Medium
99	Explain structure of Magnetic hard disk along with read and write operations.	CO5	5	Low
100	Explain the exceptions in interrupt handling process ?.Explain in detail ?.	CO5	5	Medium

(1.) Describe the digital computer generation in brief?

Ans: In the electronic computer word we measure technological advancement by generation.
A specific system is said to belong to a specific "generation".
Currently we are moving twoords the Fourth generatio

FIRST GENERATION!

The computer of first generation (1951-1958) were physically very large machines characterized by the vacum tube. They are very unreliable and required lot of Power and generate heat.

SECOND GENERATION!

The computer of second generation (1959-1963), were characterized by transistor. transistor were smaller, less expensive, generate almost no heat and required very less power.

THIRD GENERATION!

The computer of this generation (1964-1970), are characterized by miniaturized circuits.
This incress the durability and internal processing speed by reducing the physical size of computer.

FOURTH GENERATION AND BEYOND!

In Forth generation, the manufacturing of integrated circu has advanced to the points where thousand of active

component can be placed on a silicon wafer only a fraction of an inch in size. As result of this computer are significantly smaller in physical size and lower in cost.

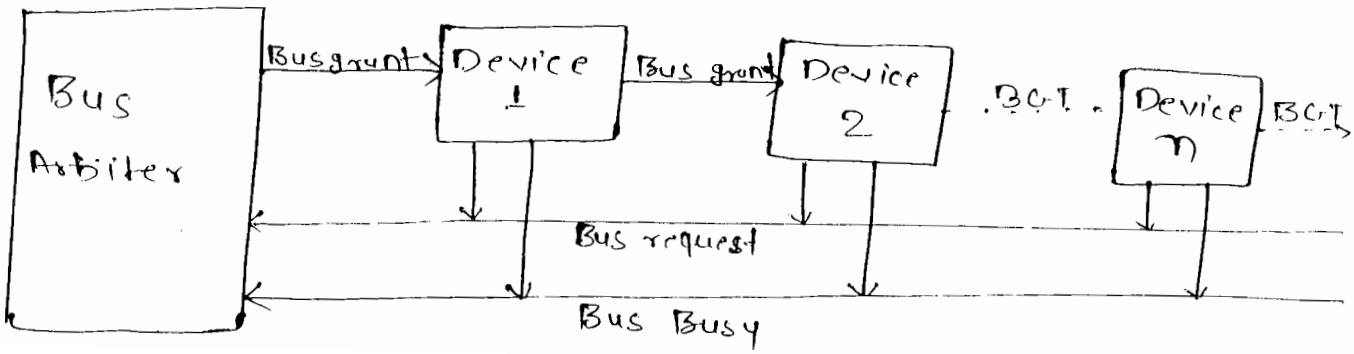
(2.) Define bus arbitration with suitable digram?

Ans: Bus arbitration is a mechanism which decide the slection of current master to access bus.
A conflict may cries if number of process try to access the comman bus at same time, but access can given only one of those.
To resolve this confilict, Bus Arbitration implemente

→ Three Different machanism is commonly used for this

(i) Daisy chaining method:

It is a simple and cheaper method where all the master use the same line for making bus request.



Advantage:

-> Simplicity and scalability

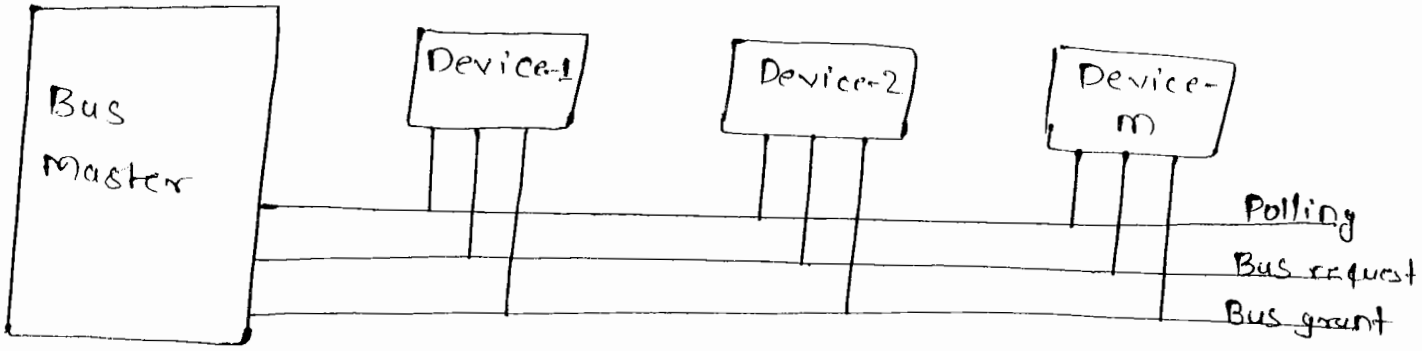
-> The user can add more device along the chain.

Dis- Advantage:

-> If one device fail then entier system stop working.

-> Propogation dealy aries in this method.

(ii) Polling method



-> In this method if Bus is not busy, make bus request ~~by~~, master polls by placing device ID on polling line. if device gets bus grant mark bus busy.

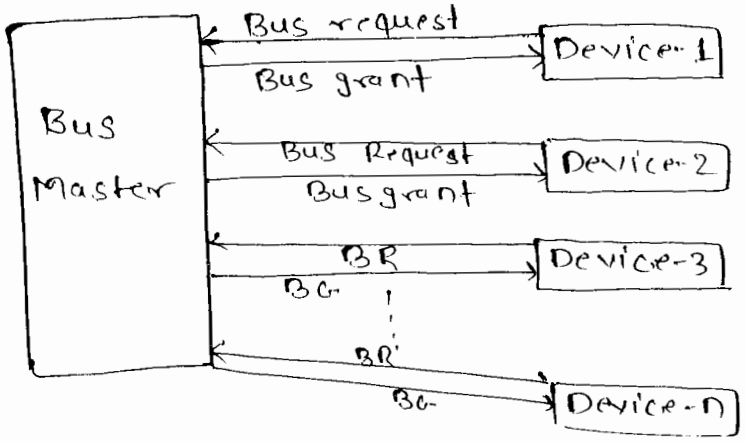
Advantage:

-> If one device fails then entier system will not stop working.

Disadvantage:

Extra poll line required.

3) Independent request:



-> Here each device request independent for bus master, and master decide who to grant access of bus.

Advan:-

-> This method generate fast response.

Dis- Adv:

- Hardware cost is high as large no. of control line are required.

Explain different functions of computer system?

Ans At a fundamental level computer operates through these four functions:

-> Input unit
-> Storage unit
-> Processing unit
-> Output unit

Input unit:

Input unit accept instruction and data from outside and convert it into computer understable Form and supply to computer for the further Process.

ii) Storage unit

A storage unit store data and instruction required for process.
it also store intermediat result of processing, outpu result befor they are relused to output device.

(iii) Central processing unit (CPU)

CPU is brain OF computer.
it take data From outside ie input device and Process them according to set of instruction calle Program and output of Program is send to outsid through output unit.

CPU is categries into two part.

-> ALU (Arithmetic and logic unit)

ALU is responsible for arthmetical operation like addition, substraction multiplication, devision etc. and logical operation like. AND, OR, NOT ex-OR etc.

-> Control unit

This unit is mainaly used for generating the electronic control signal for the synchronizatic of various operations.

(iv) Output unit

Basicaly output unit accept the result produced by computer which are in coded form, and convert them into human understable form and supply them into outside.

4. Write Sort note on Following

(i) Common Bus System.

-> The basic computer has eight registers, memory un and control unit path must be provided to transfer information from one register to another register. The number of wires is very high if connection are made between the output of each registers with input of each registers, Therefor we use common bus in place of excessive wire.

(ii) Bus Arbitration

Bus arbitration is the process by which the next device to becom the bus master.

The slection of Bus m...

⑥ Explain ALU (Arithmetic and Logic unit)? Draw the Logical digram of ALU that perform AND, OR, logic operation and ADD, SUB arithmetic operation?

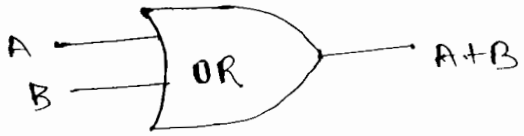
→ An arithmetic logic unit (ALU) is a digital circuit used to perform arithmetic and logic operation such as addition, substraction, multiplication and AND, OR, ex-OR operations.

- AND operation

AND

A	B	AB
0	0	0
1	0	0
0	1	0
1	1	1

- OR operation

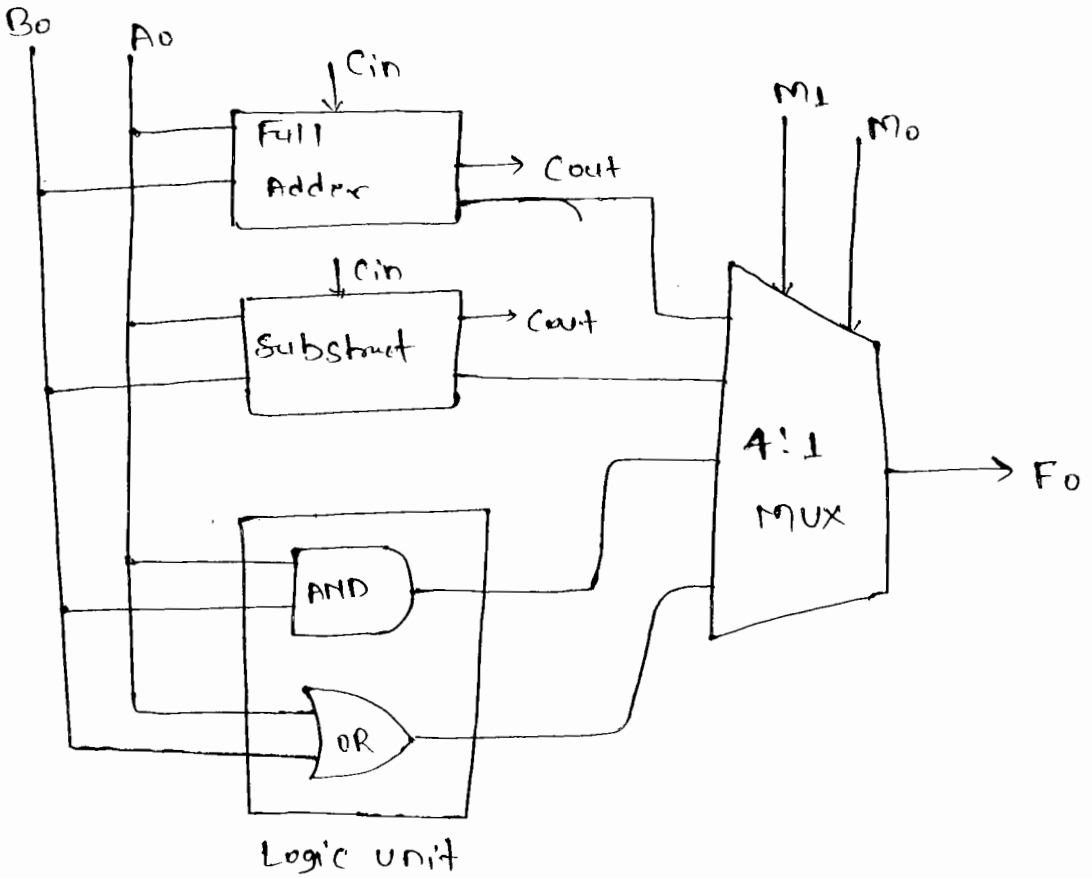


OR

A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

→ Consider a simple ALU the perform a arithmetic operation (1 bit addition) and ~~does~~ substraction and logic operation namely AND, OR, ~~NOR~~ as shown below.

The operation slected depends on the slection line of the multiplexer as show-



1- bit ALU

input		output
M_1	M_0	operation
0	0	ADD
1	0	SUB
0	1	AND
1	1	OR

(7-7) Compute the following arithmetic operation in 8-bit register, use 2's complement notation indicate overFlow / underFlow if any. ?

(a) $-28 - (-1\infty)$

$-28 + (-(-1\infty))$

$-28 + 1\infty$

$-28 = 11100100$

$+1\infty = 01100100$

$+28 = 00011100$

1's ↓ 11100011

$+1$

$-28 \Rightarrow 11100100$

Carry out of Sign bit ↓ carry into sign bit ↓

$$\begin{array}{r} 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0 \longrightarrow \text{Carry} \\ 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0 \rightarrow -28 \\ +\ 0\ 1\ 1\ 0\ 0\ 1\ 0\ 0 \rightarrow +1\infty \\ \hline 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0 \\ \hline \text{Answer} \end{array}$$

OR $(72)_{10}$

Here,

Carry into Sign bit = 1

Carry out of Sign bit = 1

∴ No overFlow

(b) $-28 - 1\infty$

$-28 + (-1\infty)$

$-28 = 11100100$

$-1\infty = 10011100$

$+1\infty = 01100100$

1's ↓ 10011011

$+1$

$-1\infty \Rightarrow 10011100$

$$
\begin{array}{rl}
 & \boxed{1\ 1}\ 1\ 1\ 1\ 1\ 0\ 0 \longrightarrow \text{carry} \\
 & 1\ 1\ 1\ 0\ 0\ 1\ 0\ 0 \leftarrow -28 \\
+ & 1\ 0\ 0\ 1\ 1\ 1\ 0\ 0 \leftarrow -100 \\
\hline
 & 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\
\hline
\end{array}
$$

2's

Here,

Carry into sign bit = 1

Carry out of sign bit = 1

∴ No overflow

(c) $74 - (-49)$

$74 + (49)$

$+74 = 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0$

$+49 = 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1$

$$
\begin{array}{rl}
 & \boxed{0\ 0}\ 0\ 0\ 0\ 0\ 0\ 0 \longrightarrow \text{carry} \\
 & 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0 \rightarrow +74 \\
+ & 0\ 0\ 1\ 1\ 0\ 0\ 0\ 1 \longrightarrow +49 \\
\hline
 & 0\ 1\ 1\ 1\ 1\ 0\ 1\ 1 \\
\hline
\end{array}
$$

Ans

or $(+123)_{10}$

Here, Carry into sign bit = 0

Carry out of sign bit = 0

∴ No overflow

(d) $+50 - 5$

⇒ $+50 + (-5)$

$+50 = 00110010$

$-5 = 11111011$

$$\begin{array}{r} \boxed{1\ 1}\ 1\ 1\ 0\ 0\ 1\ 0 \leftarrow \text{carry} \\ 0\ 0\ 1\ 1\ 0\ 0\ 1\ 0 \leftarrow +50 \\ +\ 1\ 1\ 1\ 1\ 1\ 0\ 1\ 1 \leftarrow -5 \\ \hline 0\ 0\ 1\ 0\ 0\ 1\ 0\ 1 \\ \hline \end{array}$$ Ans

OR $(+45_{10})$

Here, Carry into sign bit = 1

Carry out of sign bit = 1

∴ No overflow.

(8) Elebrate the following conditional control statemer by two register statement with control function

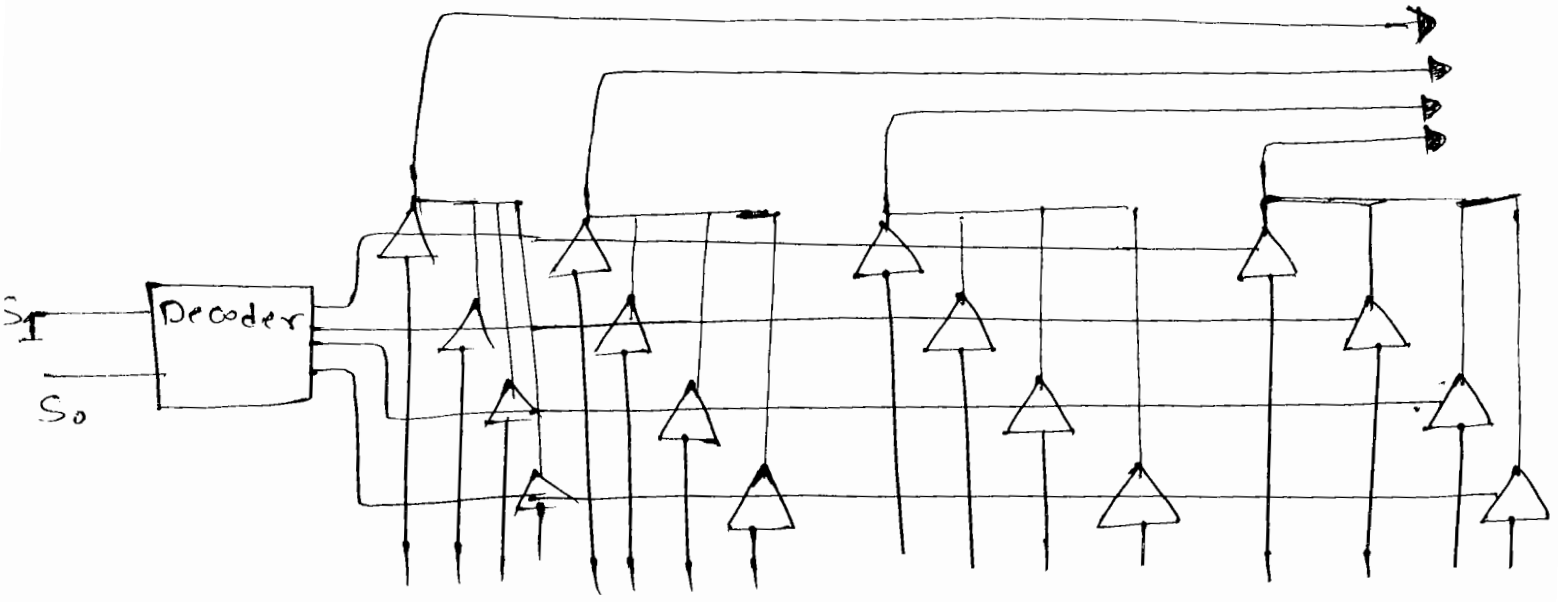
IF (P=1) then ($R1 \leftarrow R_2$) else if (Q=1) then ($R1 \wedge R3$)

→ Here if (P=1) then ($R1 \leftarrow R_2$) operation is done

ie P: $R1 \leftarrow R_2$; if(P=1) then ($R1 \leftarrow P_2$)

P'Q: $R1 \leftarrow R3$; else if (Q=1) then ($P1 \leftarrow P3$)

(5) Draw the digram of a bus system for four registers that uses three state buffer and a decoder instead of the multiplexers.



→ The S1 and S2 control signal are decoded so that only one of the output of decoder is activated at a time.

(9) Explain various addressing mode with example?

Ans Type of Addressing mode!

1) Immediate mode!

In this mode, the operand is specified in the instructi itself.

For example-

ADD 7

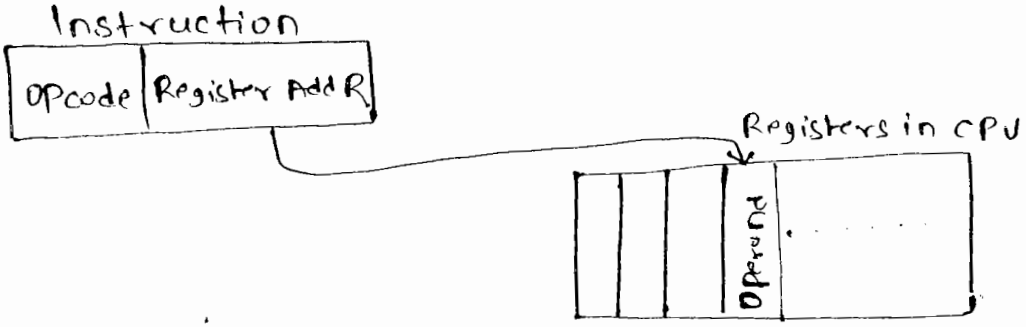
which say that Add 7 to content of accumulator.

AC ← AC + 7

2- Register mode

In this mode the operand is stored in the register and this register is present in CPU.

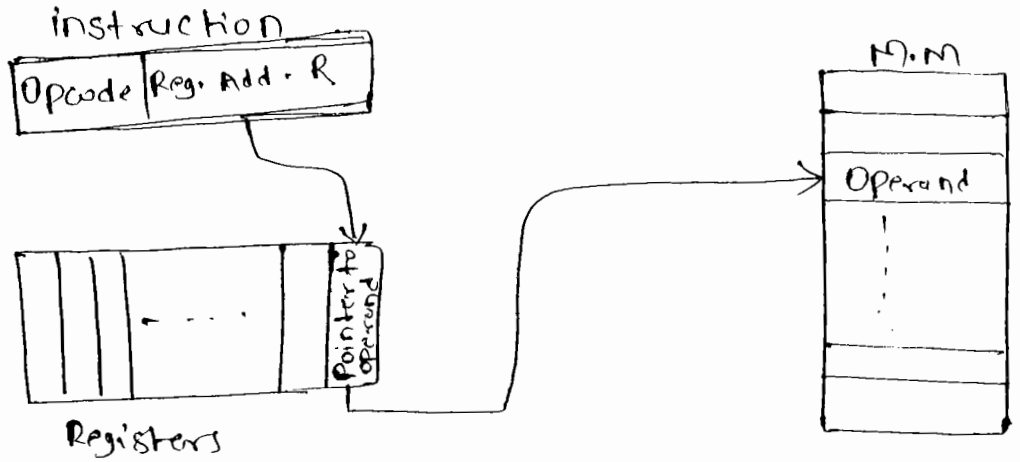
For example-



3. Register Indirect mode

In this mode, the instruction specify the register whose contents give us the address of operand which in the memory, Thus register contain address of operand rather then operand itself.

Ex-

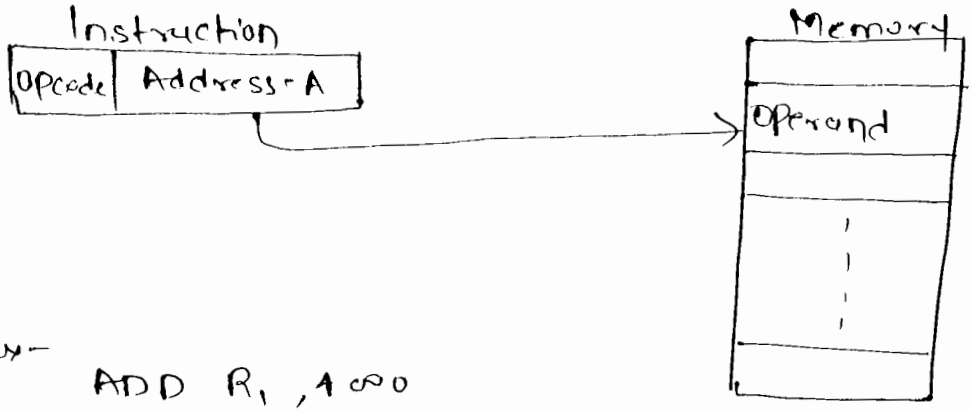


Auto Increment / Decrement mode:

→ In this mode the register is increment or decrement after or before its value used.

Direct Addressing mode

→ In this mode effective address of operand is present in instruction itself.
ie no additional calculation to find Effective add.



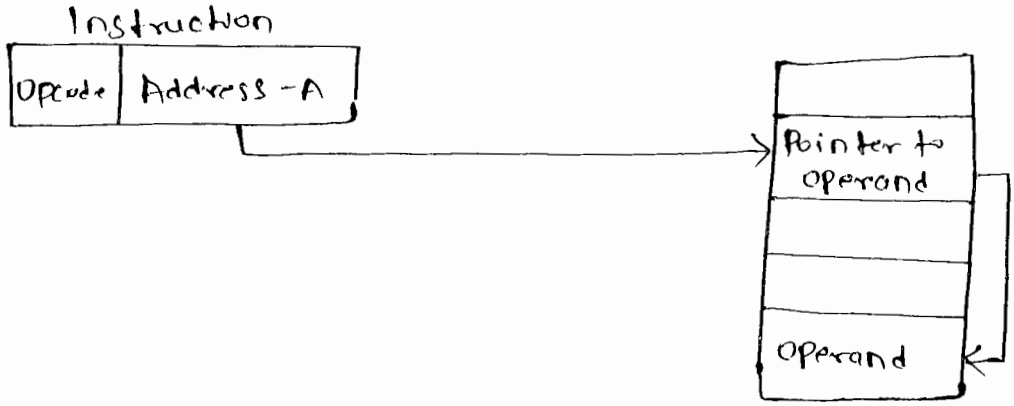
for Ex-

ADD R1, 1000

~~R1 ← R1 + M[1000]~~

-> Indirect Addressing mode

In this, the address field of instruction gives the Address where the effective address is stored in memory.



~~Stack Addressing mode~~

~~In this mode, operand is at the top of the stack.~~

~~for example :~~

~~ADD,~~

~~this instruction will pop top two items from the stack, add them and will push the result to the top of stack.~~

(10) Describe stack organization ? Compare Register stack and memory stack ?

Ans The computer which use stack-based CPU organization are based on a data structure called stack.

A stack is list of data words, which uses last in first out (LIFO) access method.

the two operation of stack

1. PUSH insert an item on top of stack.
2. POP Delete an item on top of stack.

⇒ Implementation of stack organization

in digital computer stack can be implemented in two way.

-> Register stack

-> memory stack

⇒ Register stack

Full Empty

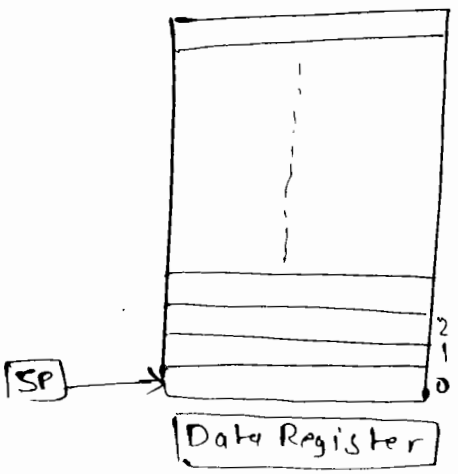
~~POP~~ PUSH

SP ← SP+1

M[SP] ← DR

IF(SP=0) then (Full ← 1)

Empty ← 0



POP

DR ← M[SP]

SP ← SP-1

IF(SP=0) then

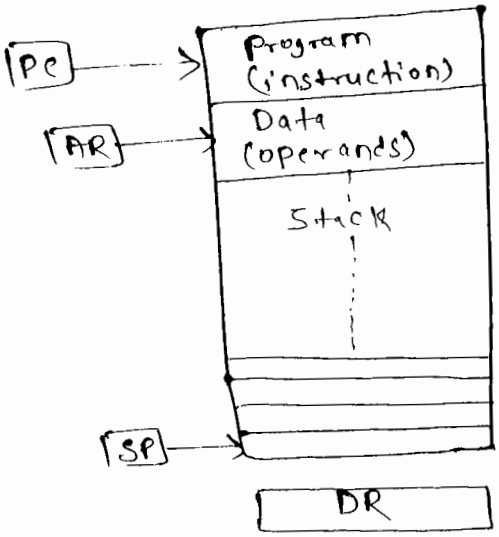
Empty ← 1

Full ← 0

→ A ~~Stack~~ ~~c~~

A Register stack can be organized as a collection of finite number of Register that are used to store temporary information during execution of Program. The stack pointer (SP) is a register that hold the address of top of element of the stack.

② memory stack



PUSH

SP ← SP-1

M[SR] ← DR

POP

DR ← M[SP]

SP ← SP+1

→ A stack can be implemented in a random access memory (RAM) attached to CPU.

→ Difference between the memory stack and register stack

→ The memory stack is implemented using computer memory (RAM) but register stack is implemented using Registers.

→ The memory stack is slower than the Register stack.

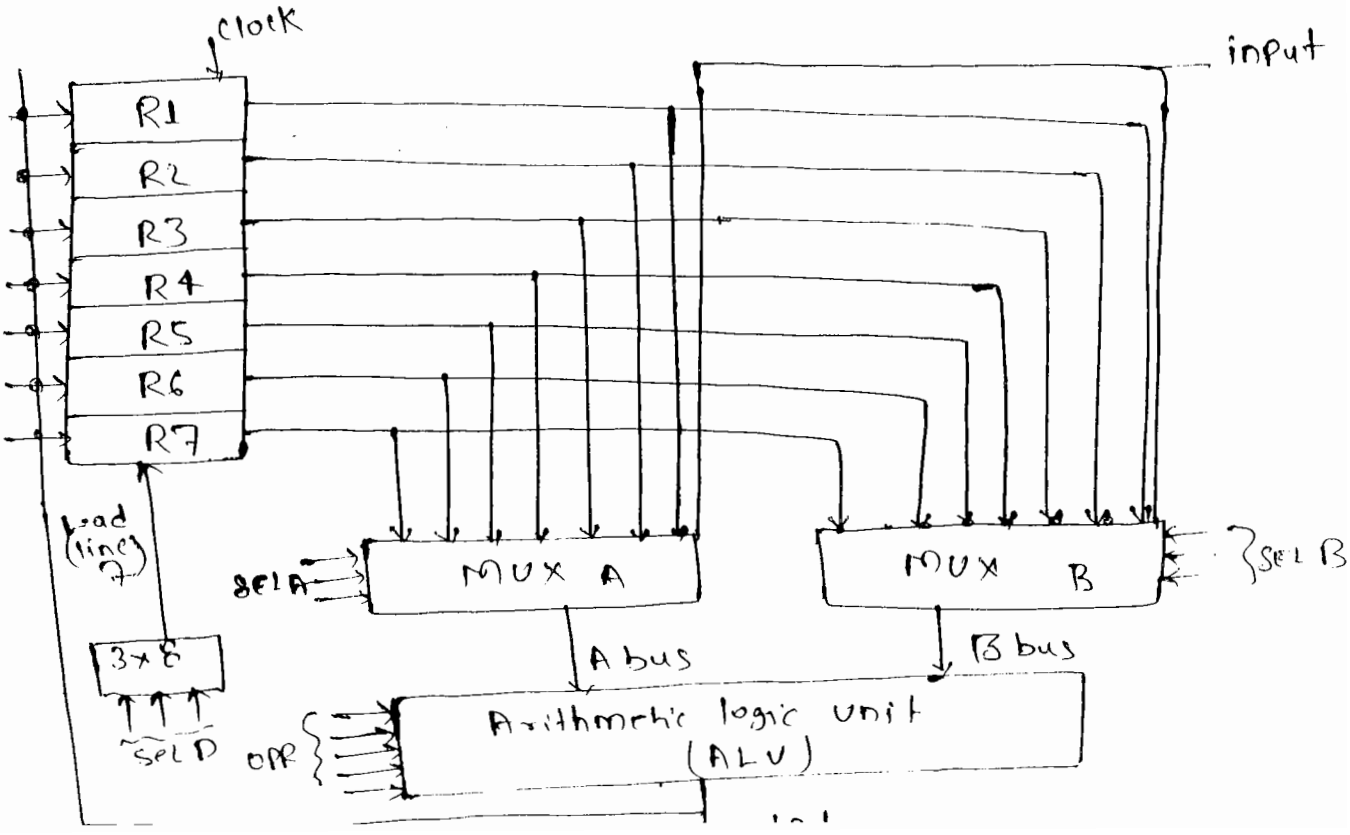
→ A Register stack is Fairly Small in Size but memory stack is large in size as compair to Register Stack.

→ The implementation cost of Register stack is greater than the memory stack.

(11) Describe use of register in computer ?
Explain general Register organization with multiple bus.

→ Registers are a type of computer memory used to quickly accept, store, and transfer data and instruction that are being used immediately by the CPU.

→ General Register organization



→ Generally CPU has seven general register. Register organization show how register are slected and how data Flow between Register and ALU.

A decoder is use to slect perticuler register.

The output of each register is connected to two multiplexers to form two buses A and B. The slection line in each multiplexer slect the input data for the perticuler bus.

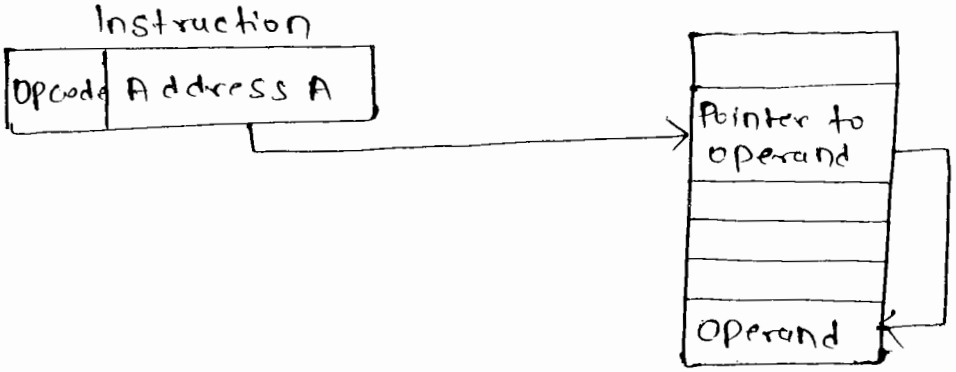
The A and B buses form the two inputs of an ALU. The operation slect line decide the micro operation to be performed by ALU. The Result of ~~ALU~~ is micro operation is available at output bus. the output bus is connected to the input of all the register, thus by slecting the destination register it is possible to store the result in it.

(12) Explain the need of having many addressing mode in your computer.? Discuss Indirect and displacement.

→ The way any operand is slected during the program execution is dependent on the addressing mode. of the instruction.

The Purpose of using addressing mode is to reduce the number of bits in addressing field of instructio

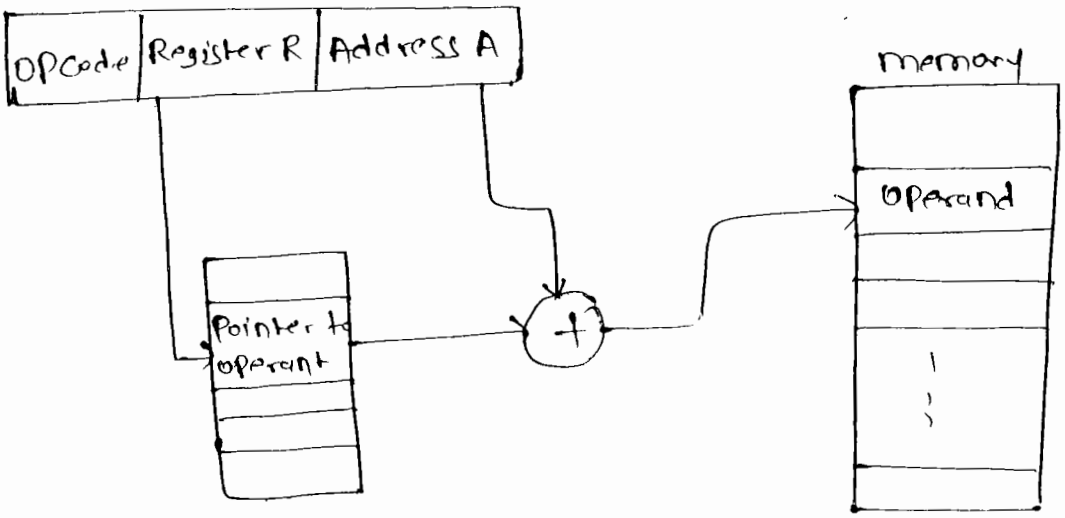
→ Indirect Addressing mode



In this, the address field of instruction gives the address where the effective address is stored in memory. This slow down the execution, as multiple memory search to find effective address.

→ Displacement Addressing mode

In this, the contents of the indexed register is added to the address part of the instruction, to obtain the effective address of operand.



(13) The following statement specify a memory. Explain the memory operation in each case?

→ (a) $R_2 \leftarrow M[AR]$

→ This statement would transfer the contents of memory word that has the address specified by AR into R2 register.

→ (b) $M[AR] \leftarrow R3$

→ WRITE the value in register R3 into the memory word that has the address specified in AR.

→ (c) $R5 \leftarrow M[R5]$

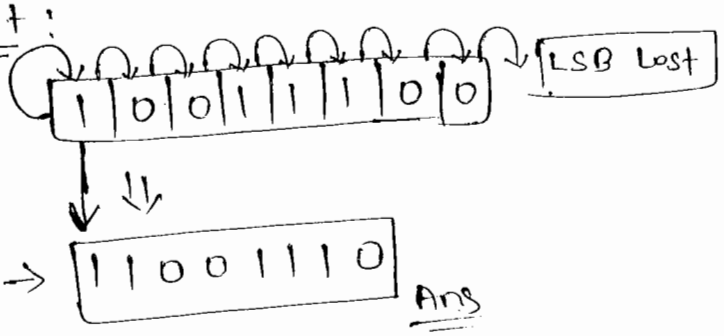
It will Firstly READ the memory word specified by R5 and then transfer the value into same Register R5.

(14) An 8-bit register contain the binary value 10011100. What is the register value after arithmetic Shift Right.? Starting from the initial number 10011100 determine the register value after an arithmetic shift left, and state weather there is an overFlow.

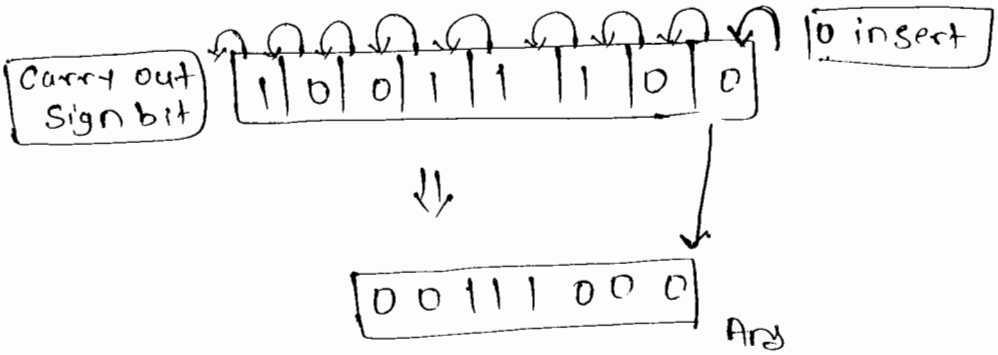
Ans

R = 10011100 (Given)

Arithmetic Shift right:



Arithmetic Shift left:



→ overFlow because a negative number ~~ch~~ change to positive.

(15) Review about wrong with the following register transfer statement ?

a) $xT : AR \leftarrow AR, AR \leftarrow 0$

b) $yT : R1 \leftarrow R2, R1 \leftarrow R3$

a $xT : AR \leftarrow AR, AR \leftarrow 0$

This statement is ambiguous ; it doesn't determine what to be load into AR.

ie two microoperation done the same time on same Register.

(b) $yT : R1 \leftarrow R2, R1 \leftarrow R3$

This micro operation make no sence, because two value are to be loaded into the same register at same time.

piyusfcs@
rkgit.edu.in

COA Unit 2 Solutions

Q.21. Explain Booth's Algorithm in detail with flow chart. Implement multiplication of two operands (-12) and (+4) with the help of Booth's algorithm.

Ans.21: Booth's algorithm is a powerful algorithm that is used for signed multiplication. It generates a 2n bit product for two n bit signed numbers. Booth algorithm gives a procedure for,multiplying binary integers in signed 2's complement representation in efficient way, i.e., less number of additions/subtractions required. 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 2^k to weight 2^m can be treated as 2^(k+1) to 2^m.

The steps in Booth's algorithm are as follow:

1) Initialize A,Q_{-1} to 0 and count to n

2) Based on the values of Q_0 and Q_{-1} do the following:

a. if Q_0,Q_{-1}=0,0 then Right shift A,Q,Q_{-1} and finally decrement count by 1

b. If Q_0,Q_{-1}=0,1 then Add A and B store in A, Right shift A,Q,Q_{-1} and finally decrement count by 1

c. If Q_0,Q_{-1}=1,0 then Subtract A and B store in A, Right shift A,Q,Q_{-1} and finally decrement count by 1

d. If Q_0,Q_{-1}=1,1 then Right shift A,Q,Q_{-1} and finally decrement count by 1

3) Repeat step 2 till count does not equal 0.

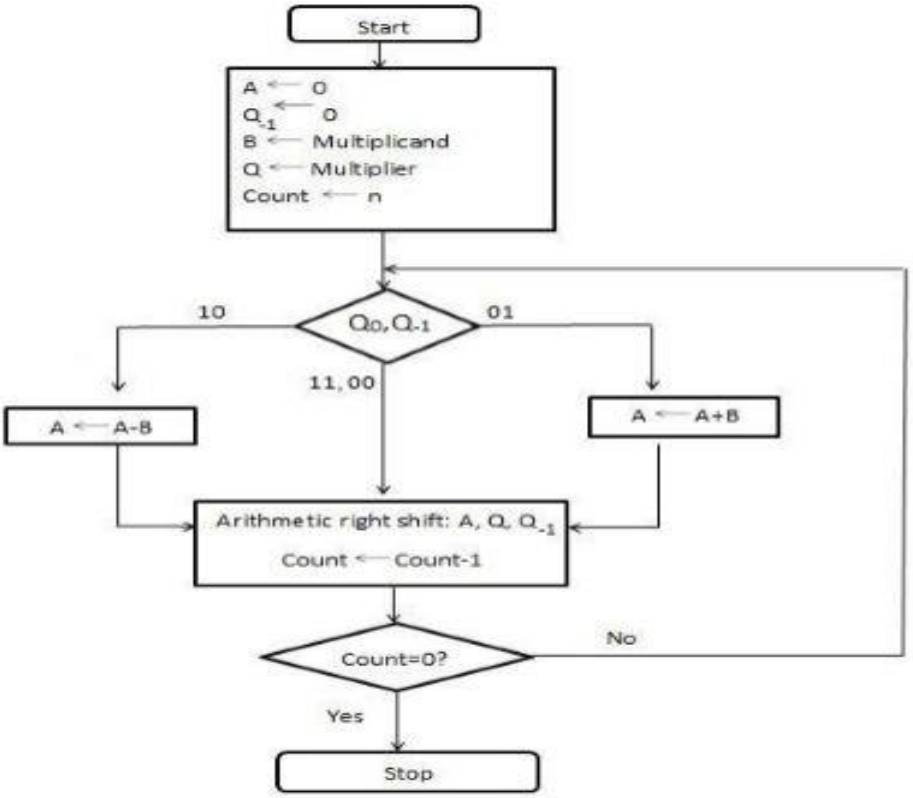
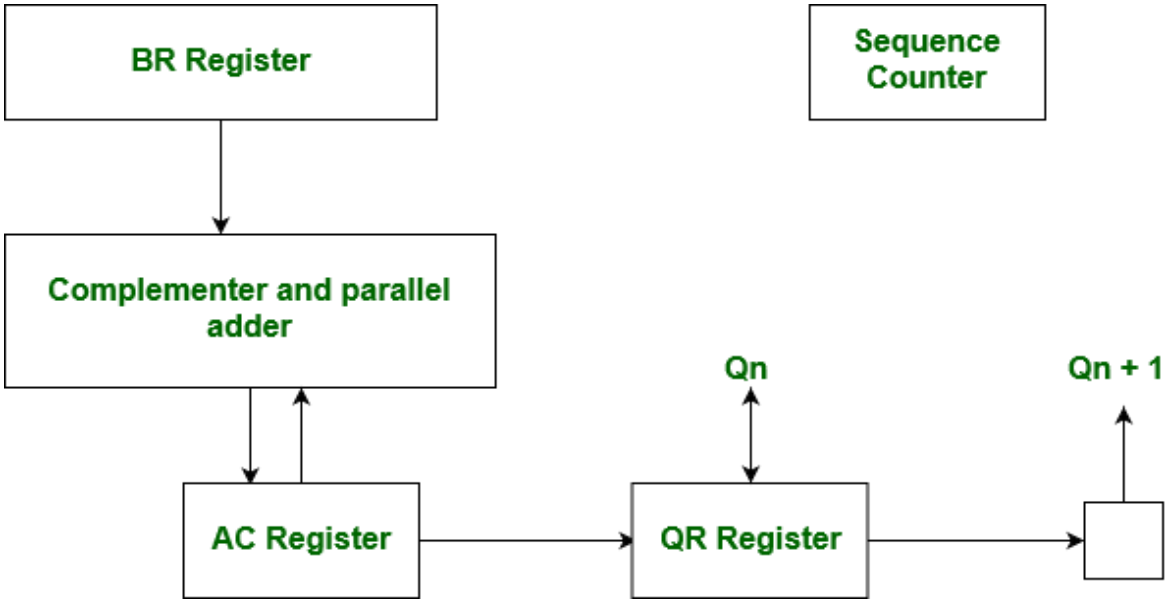


Figure 1

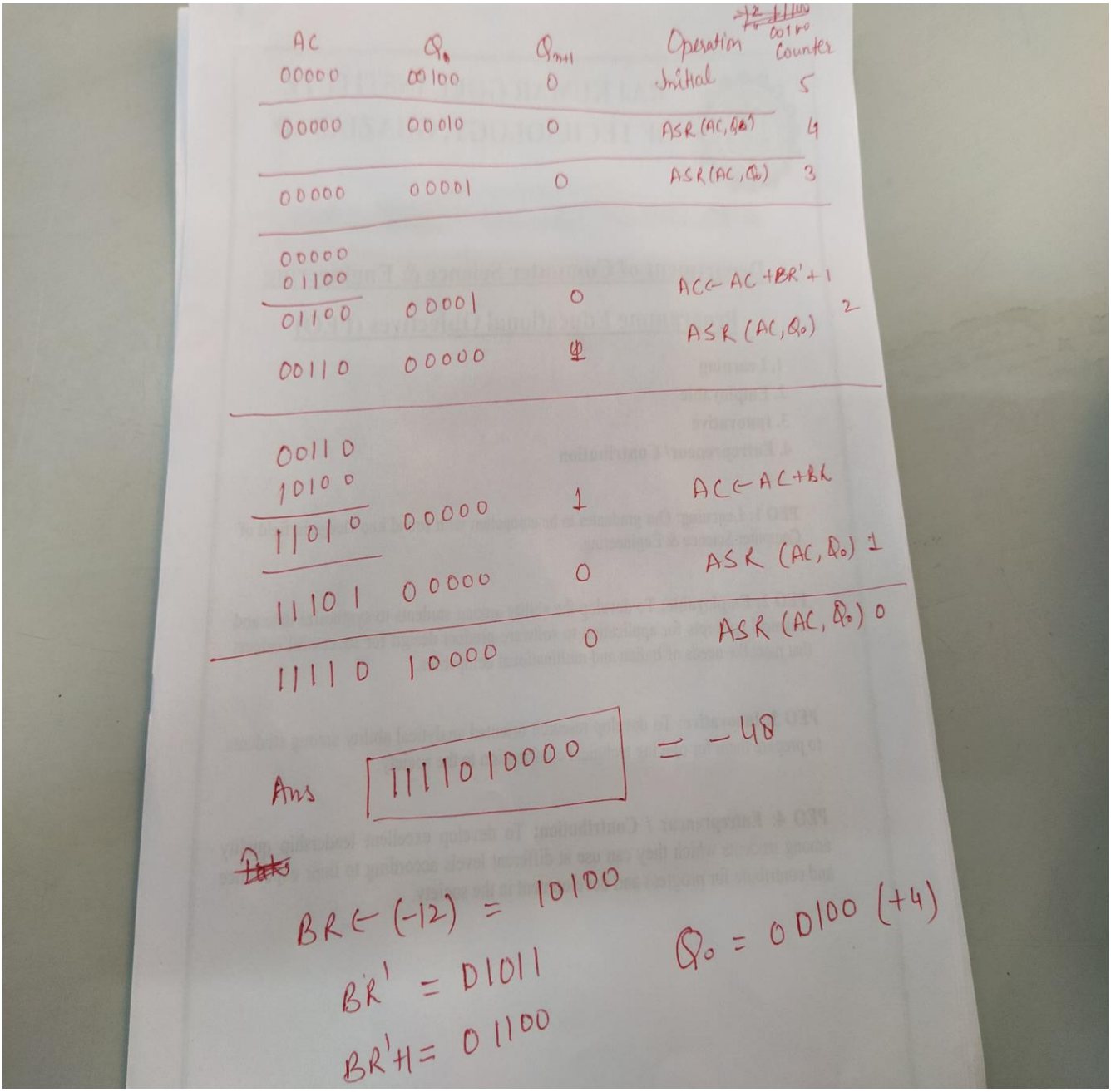
AC and the appended bit Qn+1 are initially cleared to 0 and the sequence SC is set to a number n equal to the number of bits in the multiplier. The two bits of the multiplier in Qn and Qn+1are inspected. If the two bits are equal to 10, it means that the first 1 in a string has been encountered. This requires subtraction of the multiplicand from the partial product in AC. If the 2 bits are equal to 01, it means that the first 0 in a string of 0's has been encountered. This requires the addition of the multiplicand to the partial product in AC.

When the two bits are equal, the partial product does not change. An overflow cannot occur because the addition and subtraction of the multiplicand follow each other. As a consequence, the 2 numbers that are added always have a opposite signs, a condition that excludes an overflow. The next step is to shift right the partial product and the multiplier (including Qn+1). This is an arithmetic shift right (ashr) operation which AC and QR ti the right and leaves the sign bit in AC unchanged. The sequence counter is decremented and the computational loop is repeated n times

Hardware Implementation of Booths Algorithm – The hardware implementation of the booth algorithm requires the register configuration shown in the figure below.



We name the register as A, B and Q, AC, BR and QR respectively. Qn designates the least significant bit of multiplier in the register QR. An extra flip-flop Qn+1is appended to QR to facilitate a double inspection of the multiplier.
Solution of numerical



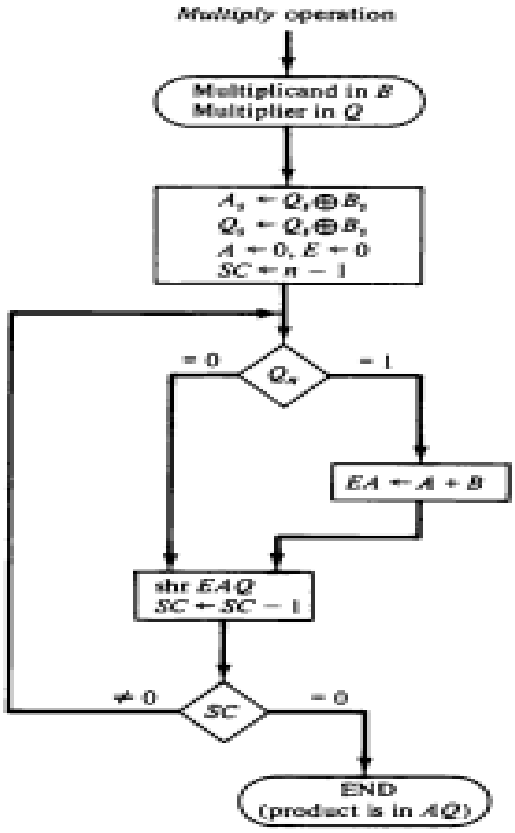
Q.22: Explain Unsigned multiplication algorithm with flow chart. Implement multiplication of 12 and 4 using Unsigned multiplication algorithm

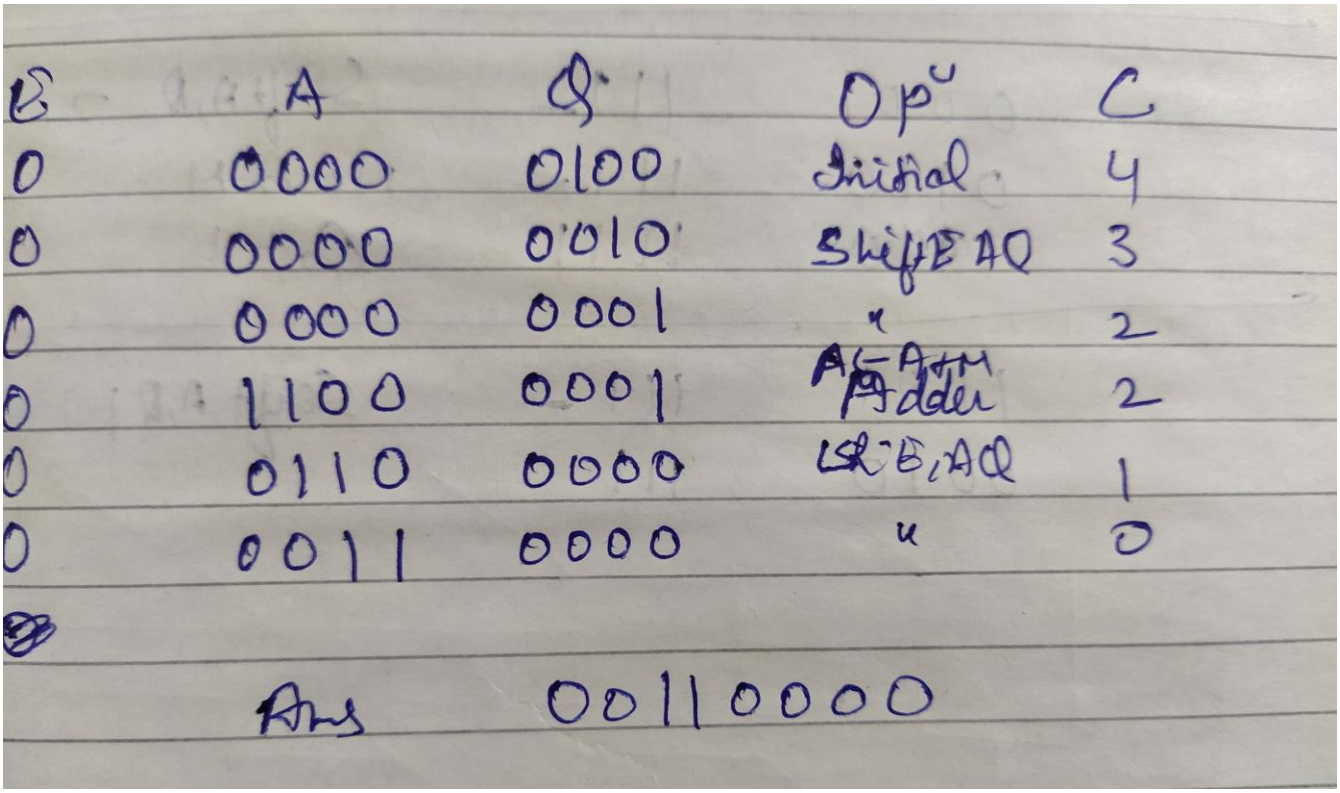
Ans.22: Initially the multiplicand in M and the multiplier in Q. Their coreesponding sign in M_s and Q_s , respectively. The signs are compared and both A and Q are set to

correspond to the sign of the product since a double-length product will be stored in registers A and Q. Registers A and E are cleared and the sequence counter SC is set to a number equal to the number of bits of the multiplier. We are assuming here that operands are transferred to registers from a memory unit that has words of n bits. Since an operand must be stored with its sign, one bit of the word will be occupied by the sign and the magnitude will consist of $n - 1$ bits.

After the initialization, the low-order bit of the multiplier in Q_n is tested. If it is a 1, the multiplicand in B is added to the present partial product in A. If it is a 0, nothing is done. Register EAQ is then shifted once to the right to form the new partial product. The sequence counter is decremented by 1 and its new value checked. If it is not equal to zero, the process is repeated and a new partial product is formed. The process stops when $SC = 0$. Note that the partial product formed in A is shifted into Q one bit at a time and eventually replaces the multiplier. The final product is available in both A and Q, with A holding the most significant bits and Q holding the least significant bits.

The previous numerical example is repeated in Table 10-2 to clarify the hardware multiplication process. The procedure follows the steps outlined in the flowchart.





E	A	Q	Op^u	C
0	0000	0100	Initial	4
0	0000	0010	Shift EAQ	3
0	0000	0001	"	2
0	1100	0001	A ← A+M Adder	2
0	0110	0000	LSR E, AQ	1
0	0011	0000	"	0

Ans 00110000

Q.23: Design a 3*3 bit array multiplier with a suitable diagram
Ans.23:

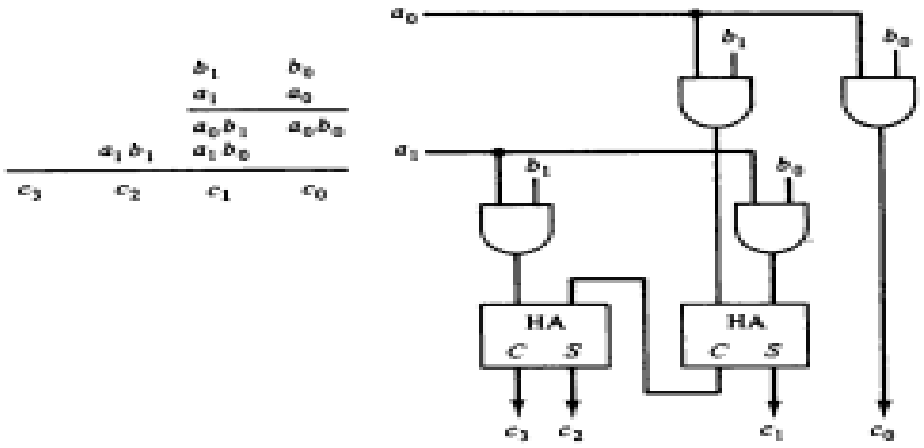
Array Multiplier

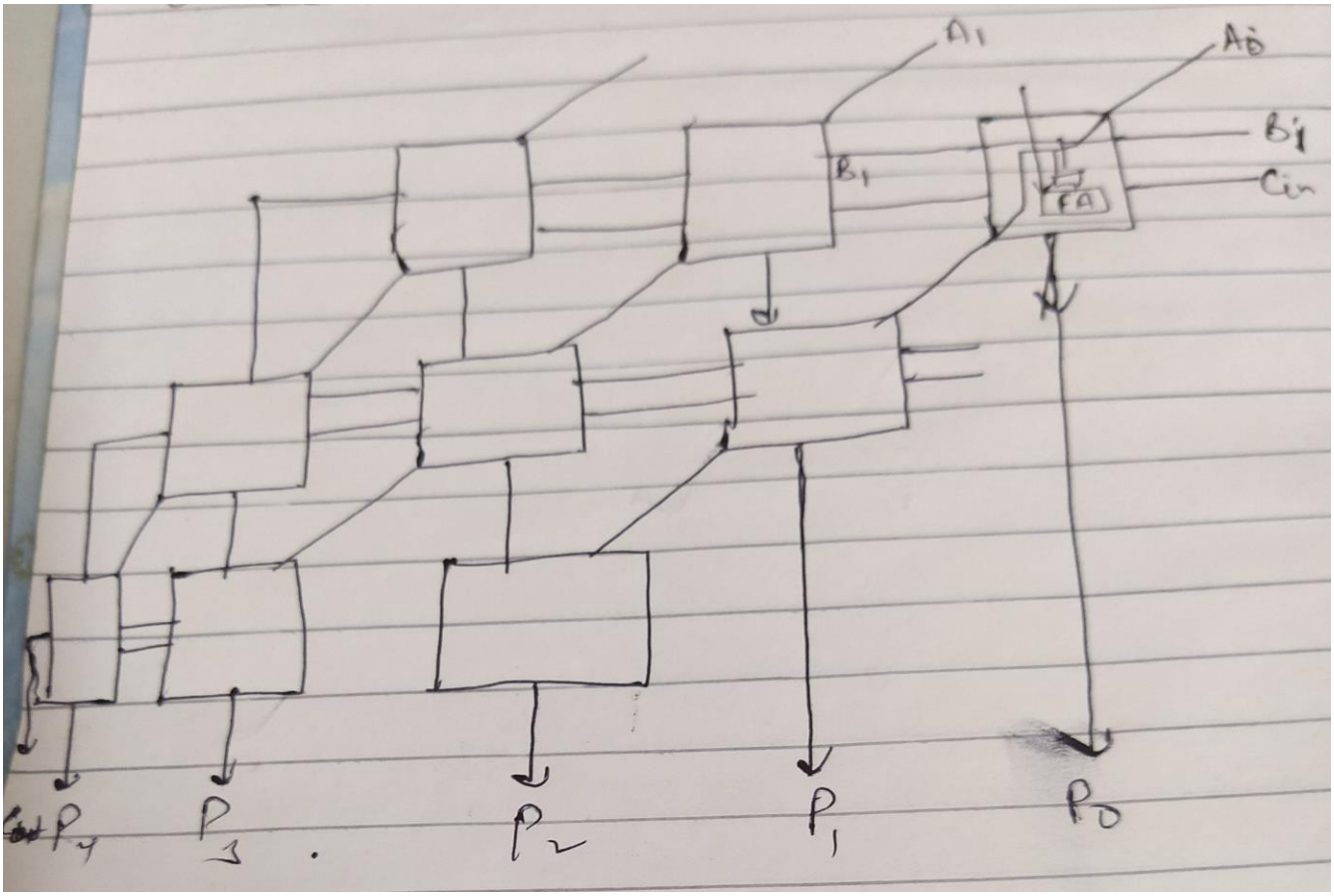
Checking the bits of the multiplier one at a time and forming partial products is a sequential operation that requires a sequence of add and shift microoperations. The multiplication of two binary numbers can be done with one microoperation by means of a combinational circuit that forms the product bits all

at once. This is a fast way of multiplying two numbers since all it takes is the time for the signals to propagate through the gates that form the multiplication array. However, an array multiplier requires a large number of gates, and for this reason it was not economical until the development of integrated circuits.

To see how an array multiplier can be implemented with a combinational circuit, consider the multiplication of two 2-bit numbers as shown in Fig. 10-9. The multiplicand bits are b_1 and b_0, the multiplier bits are a_1 and a_0, and the product is $c_3 c_2 c_1 c_0$. The first partial product is formed by multiplying a_0 by $b_1 b_0$. The multiplication of two bits such as a_0 and b_0 produces a 1 if both bits are 1; otherwise, it produces a 0. This is identical to an AND operation and can be implemented with an AND gate. As shown in the diagram, the first partial product is formed by means of two AND gates. The second partial product is formed by multiplying a_1 by $b_1 b_0$ and is shifted one position to the left. The two partial products are added with two half-adder (HA) circuits. Usually, there are more bits in the partial products and it will be necessary to use full-adders to produce the sum. Note that the least significant bit of the product does not have to go through an adder since it is formed by the output of the first AND gate.

A combinational circuit binary multiplier with more bits can be constructed in a similar fashion. A bit of the multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier. The binary output in each level of AND gates is added in parallel with the partial product of the previous level to form a new partial product. The last level produces the product. For j multiplier bits and k multiplicand bits we need $j \times k$ AND gates and $(j - 1)$ k-bit adders to produce a product of $j + k$ bits.





Q.24: Discuss IEEE standard for floating point numbers.

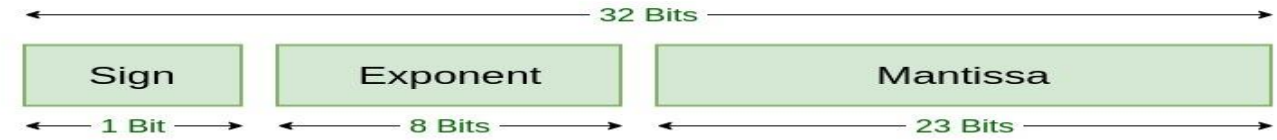
Ans.24: The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation which was established in 1985 by the **Institute of Electrical and Electronics Engineers (IEEE)**. The standard addressed many problems found in the diverse floating point implementations that made them difficult to use reliably and reduced their portability. IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC's, Macs, and most Unix platforms.

There are several ways to represent floating point number but IEEE 754 is the most efficient in most cases. IEEE 754 has 3 basic components:

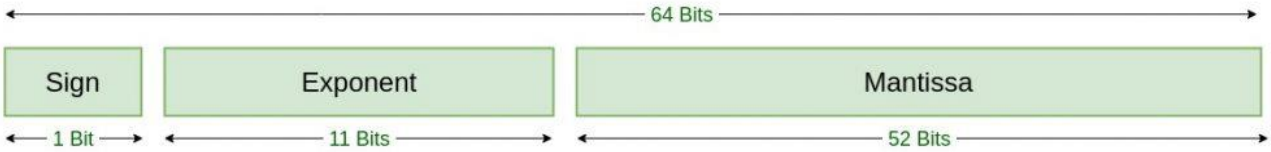
1. **The Sign of Mantissa –**
   This is as simple as the name. 0 represents a positive number while 1 represents a negative number.
2. **The Biased exponent –**
   The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.
3. **The Normalised Mantisa –**
   The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits. Here we have only 2 digits, i.e. O and 1. So a normalised mantissa is one with only one 1 to the left of the decimal.

IEEE numbers are divided into two based on the above three components:

1. single precision
2. double precision.



Single Precision
IEEE 754 Floating-Point Standard



Double Precision
IEEE 754 Floating-Point Standard

Special Values: IEEE has reserved some values that can ambiguity.

- **Zero –**
  Zero is a special value denoted with an exponent and mantissa of 0. -0 and +0 are distinct values, though they both are equal.
- **Denormalised –**
  If the exponent is all zeros, but the mantissa is not then the value is a denormalized number. This means this number does not have an assumed leading one before the binary point.
- **Infinity –**
  The values +infinity and -infinity are denoted with an exponent of all ones and a mantissa of all zeros. The sign bit distinguishes between negative infinity and positive infinity. Operations with infinite values are well defined in IEEE.
- **Not A Number (NAN) –**
  The value NAN is used to represent a value that is an error. This is represented when exponent field is all ones with a zero sign bit or a mantissa that it not 1 followed by zeros. This is a special value that might be used to denote a variable that doesn't yet hold a value.

Q.25: Discuss and Differentiate between fixed point representation and floating point representation. Explain with suitable examples

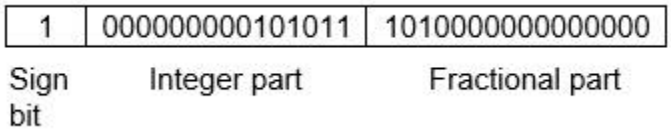
Ans.25: **Fixed-Point Representation:** This representation has fixed number of bits for integer part and for fractional part. For example, if given fixed-point representation is IIII.FFFF, then you can store minimum value is 0000.0001 and maximum value is 9999.9999. There are three parts of a fixed-point number representation: the sign field, integer field, and fractional field.

We can represent these numbers using:

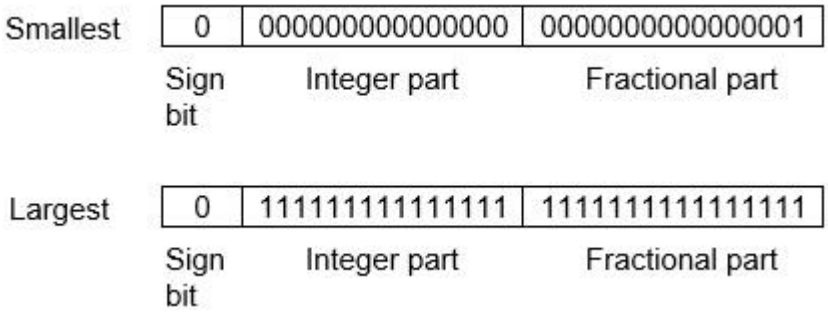
- Signed representation: range from $-(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 1's complement representation: range from $-(2^{(k-1)}-1)$ to $(2^{(k-1)}-1)$, for k bits.
- 2's complementation representation: range from $-(2^{(k-1)})$ to $(2^{(k-1)}-1)$, for k bits.

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.



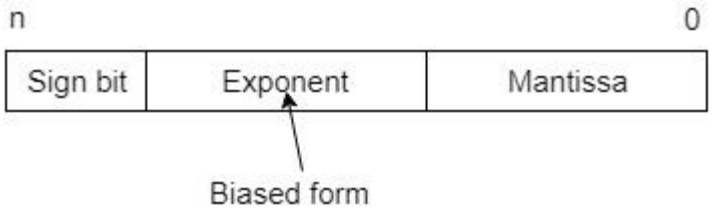
These are above smallest positive number and largest positive number which can be store in 32-bit representation as given above format. Therefore, the smallest positive number is $2^{-16} \approx 0.000015$ approximate and the largest positive number is $(2^{15}-1)+(1-2^{-16})=2^{15}(1-2^{-16}) =32768$,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

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:

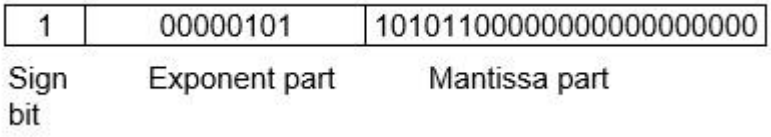
$$Mxr^e.$$

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.

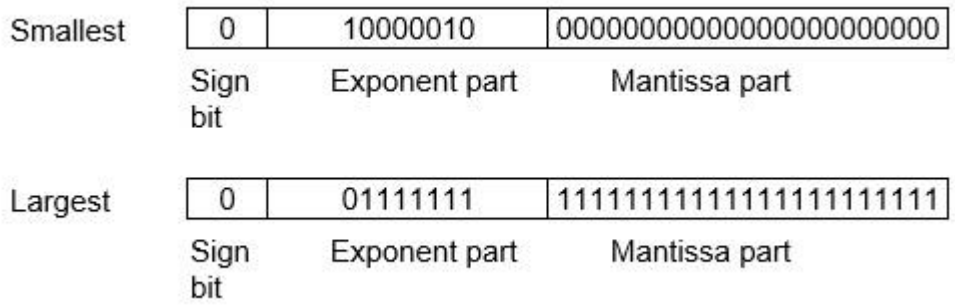


So, actual number is $(-1)^s(1+m)x2^{(e-Bias)}$, where *s* is the sign bit, *m* is the mantissa, *e* is the exponent value, and *Bias* is the bias number. Note that signed integers and exponent are represented by either sign representation, or one's complement representation, or two's complement representation. The floating point representation is more flexible. Any non-zero number can be represented in the normalized form of $\pm(1.b_1b_2b_3 ...)_2x2^n$ 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 $-53.5=(-110101.1)_2=(-1.101011)x2^5$, which is represented as following below,



Where 00000101 is the 8-bit binary value of exponent value +5. Note that 8-bit exponent field is used to store integer exponents $-126 \leq n \leq 127$. The smallest normalized positive number that fits into 32 bits is $(1.00000000000000000000000)_2x2^{-126}=2^{-126}\approx1.18x10^{-38}$, and largest normalized positive number that fits into 32 bits is $(1.11111111111111111111111)_2x2^{127}=(2^{24}-1)x2^{104} \approx 3.40x10^{38}$. 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 floating-point number as its binary representation is non-terminating.

Q.26: Explain the following with reference to floating point representation: —

(i)IEEE standard

(ii) Normalization

(iii) Biasing

(iv) Overflow & Underflow

Ans.26: (i) **IEEE standard**: The IEEE Standard for Floating-Point Arithmetic (IEEE 754) is a technical standard for floating-point computation which was established in 1985 by the **Institute of Electrical and Electronics Engineers (IEEE)**. The standard addressed many problems found in the diverse floating point implementations that made them difficult to use reliably and reduced their portability. IEEE Standard 754 floating point is the most common representation today for real numbers on computers, including Intel-based PC's, Macs, and most Unix platforms.
There are several ways to represent floating point number but IEEE 754 is the most efficient in most cases. IEEE 754 has 3 basic components:

4. **The Sign of Mantissa –**
   This is as simple as the name. 0 represents a positive number while 1 represents a negative number.
5. **The Biased exponent –**
   The exponent field needs to represent both positive and negative exponents. A bias is added to the actual exponent in order to get the stored exponent.
6. **The Normalised Mantisa –**
   The mantissa is part of a number in scientific notation or a floating-point number, consisting of its significant digits. Here we have only 2 digits, i.e. O and 1. So a normalised mantissa is one with only one 1 to the left of the decimal.

IEEE numbers are divided into two based on the above three components:

3. single precision
4. double precision.

(ii) Normalization: Normalisation is the process of moving the binary point so that the first digit after the point is a significant digit. This maximises precision in a given number of bits.

- To maximise the precision of a positive number you should have a mantissa with no leading zeros.
- To maximise the precision of a negative number you should have a mantissa with no leading ones.
- This means that the first two digits will always be different in a normalised floating point binary number.

The precision of a floating point number depends on the number of digits stored in the mantissa but also on how these digits are used:

$0.0035 \times 10^8 = 350{,}000$

$0.3457 \times 10^6 = 345{,}700$ is more precise

In binary:

```
 0  0 0 0 1 1 0 1 0 1      0 0 0 1 1 0          becomes
                              0   1 1 0 1 0 1 0 0 0      0 0 0 0 1 1
+  . 0 0 0 1 1 0 1 0 1    e x p = 6            becomes
                              +  . 1 1 0 1 0 1 0 0 0      e x p = 3
```

(iii) Biasing: Where the exponent field is supposed to be 2, yet encoded as 129 (127+2) called *biased exponent*. The exponent field is in plain binary format which also represents negative exponents with an encoding (like sign magnitude, 1's complement, 2's complement, etc.). The biased exponent is used for the representation of negative exponents. The biased exponent has advantages over other negative representations in performing bitwise comparing of two floating point numbers for equality.
A *bias* of $(2^{n-1} - 1)$, where **n** is # of bits used in exponent, is added to the exponent (e) to get biased exponent (*E*). So, the biased exponent (*E*) of *single precision* number can be obtained as

$$E = e + 127$$

The range of exponent in single precision format is -128 to +127. Other values are used for special symbols.

(iv) **Overflow and Underflow:**
Overflow is said to occur when the true result of an arithmetic operation is finite but larger in magnitude than the largest floating point number which can be stored using the given precision. nce overflow has occurred, an infinity value can be represented and propagated through a calculation.
Underflow is said to occur when the true result of an arithmetic operation is smaller in magnitude (infinitesimal) than the smallest normalized floating point number which can be stored. Overflow can't be ignored in calculations whereas underflow can effectively be replaced by zero. Underflow occurs in fl. pt. representations when a number is to small (close to 0) to be represented. (show number line!)

Q.27: State Booth Algorithm for multiplication of two numbers. Draw a logic diagram for the implementation of the Booth Algorithm for determining the product of two 8-bit signed numbers

Ans. 27: Booth's algorithm is a powerful algorithm that is used for signed multiplication. It generates a 2n bit product for two n bit signed numbers. Booth algorithm gives a procedure for, multiplying binary integers in signed 2's complement representation in efficient way, i.e., less number of additions/subtractions required. 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 2^k to weight 2^m can be treated as 2^(k+1) to 2^m.
The steps in Booth's algorithm are as follow:

1) Initialize A,Q_{-1} to 0 and count to n

2) Based on the values of Q_0 and Q_{-1} do the following:

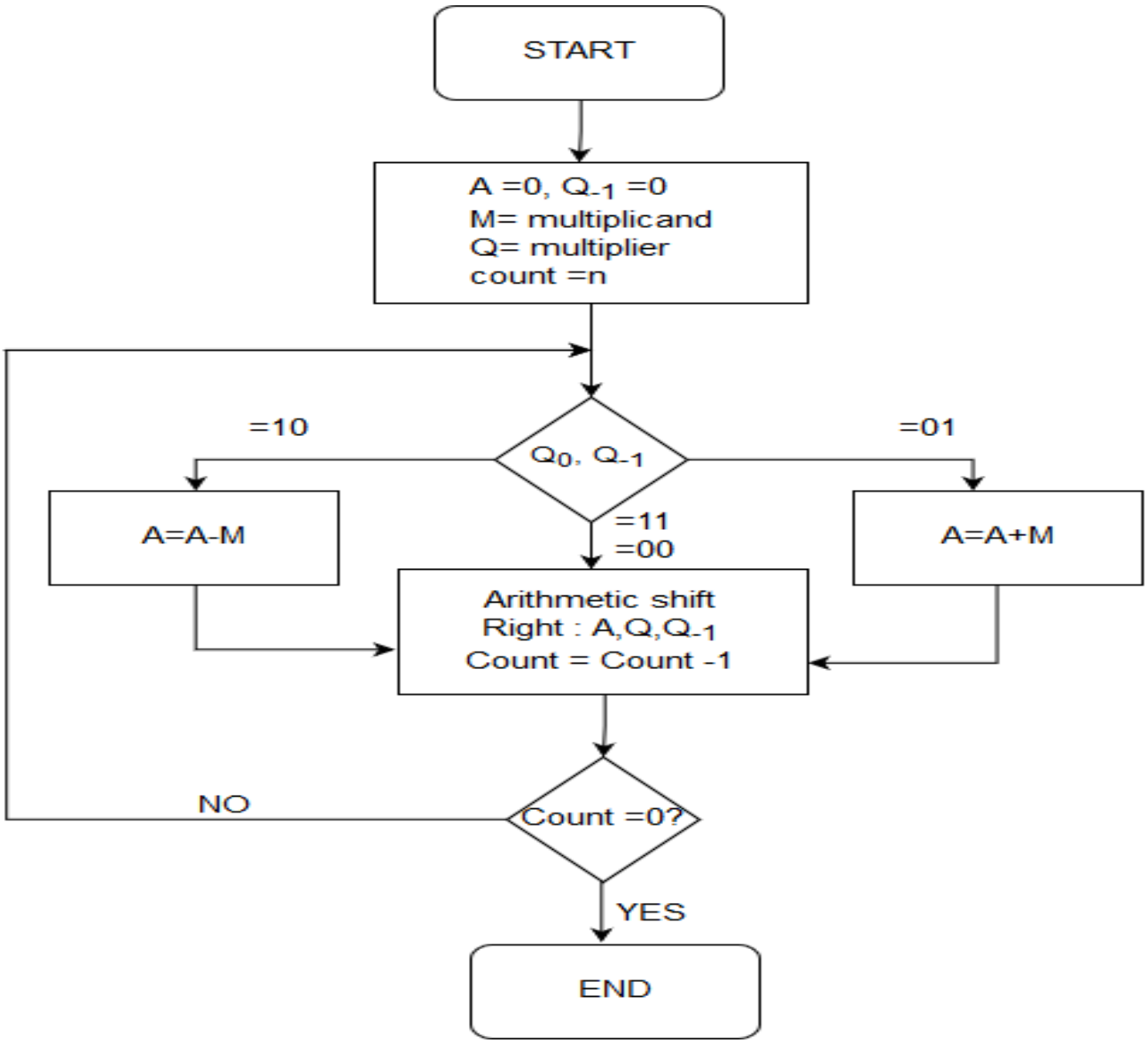
a. if Q_0,Q_{-1}=0,0 then Right shift A,Q,Q_{-1} and finally decrement count by 1

b. If Q_0,Q_{-1}=0,1 then Add A and B store in A, Right shift A,Q,Q_{-1} and finally decrement count by 1

c. If Q_0,Q_{-1}=1,0 then Subtract A and B store in A, Right shift A,Q,Q_{-1} and finally decrement count by 1

d. If Q_0,Q_{-1}=1,1 then Right shift A,Q,Q_{-1} and finally decrement count by 1

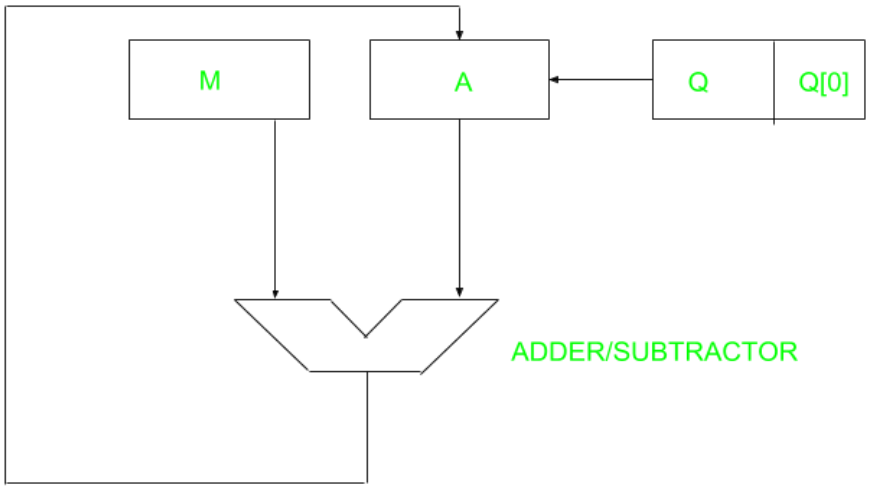
3) Repeat step 2 till count does not equal 0.



The Multiplier and Multiplicand are placed in the QQ and MM registers, respectively. There is also a 11-bit register placed logically to the right of the least significant bit(Q_0Q_0) of the QQ register and designated $Q_{-1}Q_{-1}$. The results of the multiplication will appear in the AA and QQ registers. AA and $Q_{-1}Q_{-1}$ are initialized to 0o. The control logic scans the bits of the multiplier one at a time. Now, as each bit is examined, the bit to its right is also examined. If the two bits are the same (1111 or 0000), then all of the bits of the A,Q,A,Q, and $Q_{-1}Q_{-1}$ registers are shifted to the right 11 bit. If the two bits differ, then the multiplicand is added to or subtracted from the AA register, depending on whether the two bits are 0101 or 1010. Following, the addition or subtraction, the right shift occurs. In either case, the right shift is such that the leftmost bit of AA, namely $A_{n-1}A_{n-1}$, not only is shifted into $A_{n-2}A_{n-2}$,but also remains in $A_{n-1}A_{n-1}$. This is required to preserve the sign of the number in AA and QQ. It is known as an arithmetic shift, because it preserves the sign bit.

Q.28: State the division of the following positive number 1000/11. using 4-bit register.

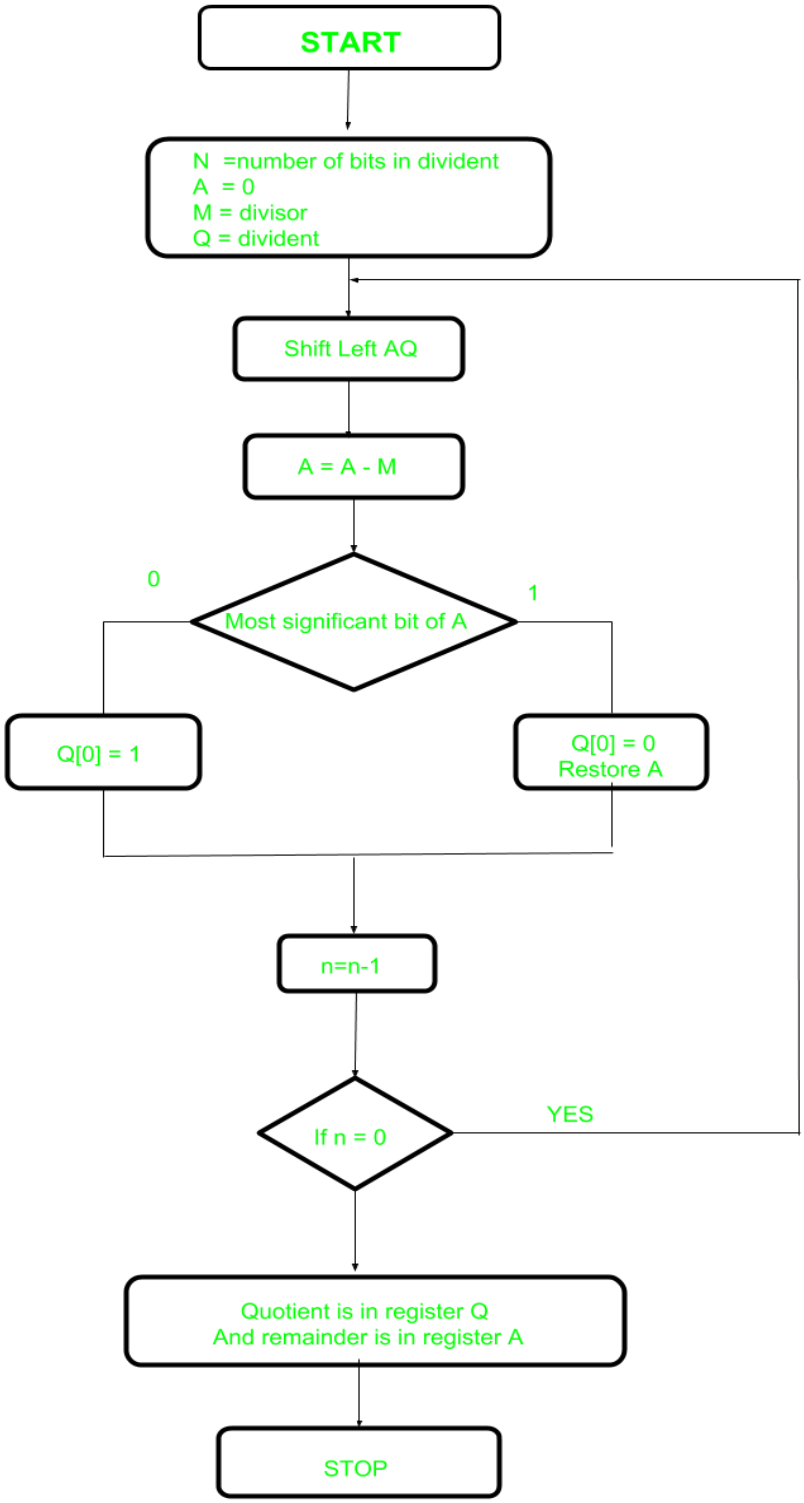
Ans.28: A division algorithm provides a quotient and a remainder when we divide two number. They are generally of two type **slow algorithm and fast algorithm**. Slow division algorithm are restoring, non-restoring, non-performing restoring, SRT algorithm and under fast comes Newton–Raphson and Goldschmidt. Restoring term is due to fact that value of register A is restored after each iteration.

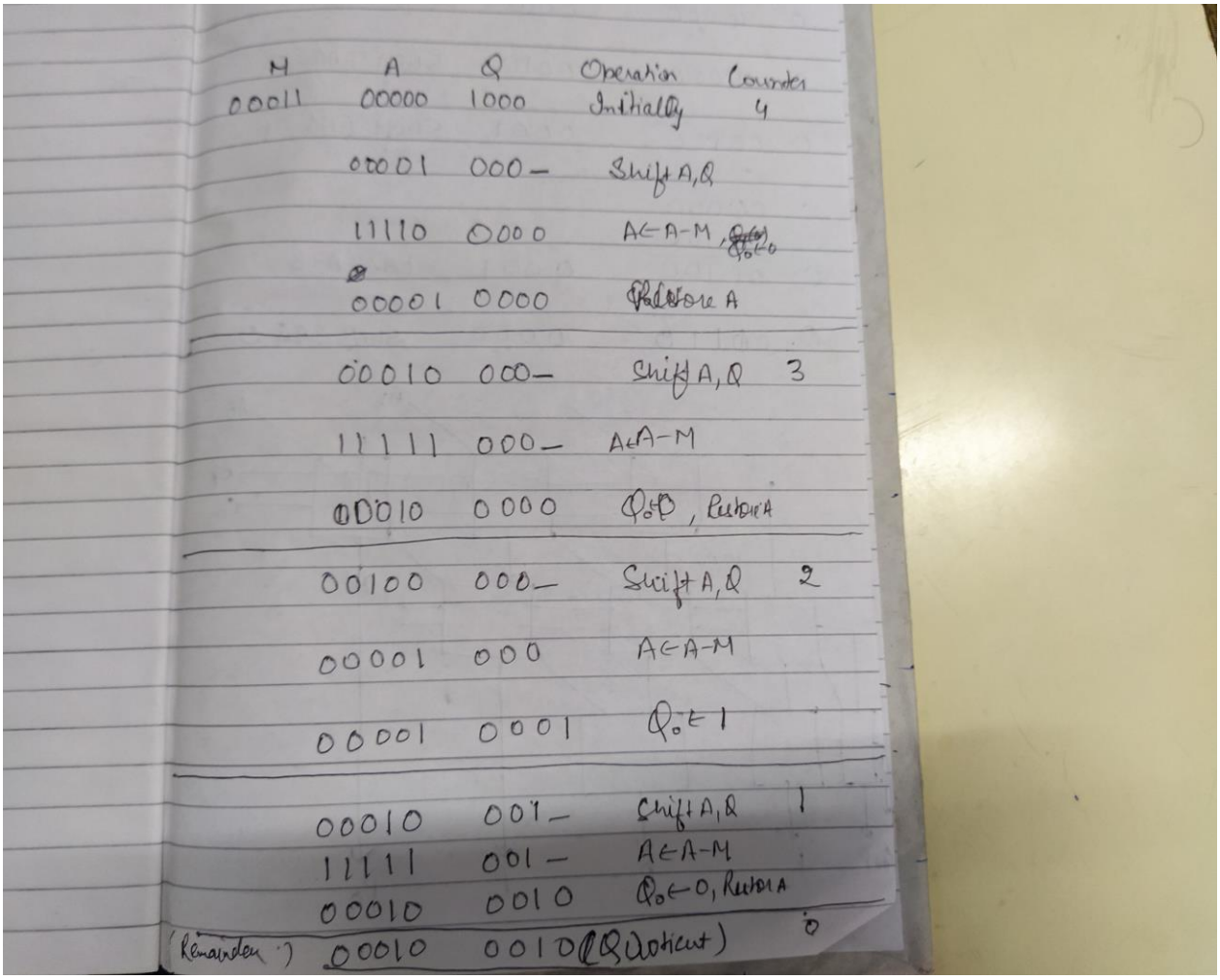


Here, register Q contain quotient and register A contain remainder. Here, n-bit dividend is loaded in Q and divisor is loaded in M. Value of Register is initially kept 0 and this is the register whose value is restored during iteration due to which it is named Restoring.

- **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)
- **Step-2:** Then the content of register A and Q is shifted right as if they are a single unit
- **Step-3:** Then content of register M is subtracted from A and result is stored in A

- **Step-4:** Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e the value of A before the subtraction with M
- **Step-5:** The value of counter n is decremented
- **Step-6:** If the value of n becomes zero we get of the loop otherwise we repeat from step 2
- **Step-7:** Finally, the register Q contain the quotient and A contain remainder

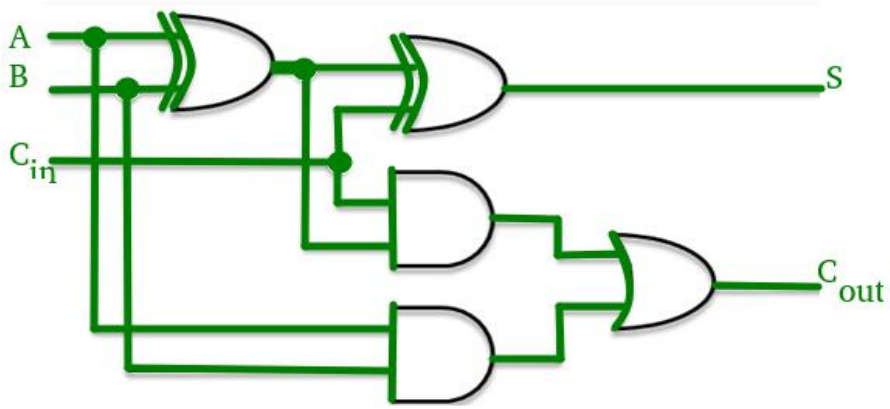

START
N =number of bits in divident
A = 0
M = divisor
Q = divident
Shift Left AQ
A = A - M
0
Most significant bit of A
1
Q[0] = 1
Q[0] = 0
Restore A
n=n-1
If n = 0
YES
Quotient is in register Q
And remainder is in register A
STOP



M	A	Q	Operation	Counter
00011	00000	1000	Initially	4
	00001	000–	Shift A,Q	
	11110	0000	A←A-M, Q_0←0	
	00001	0000	Restore A	
	00010	000–	Shift A,Q	3
	11111	000–	A←A-M	
	00010	0000	Q_0←0, Restore A	
	00100	000–	Shift A,Q	2
	00001	000	A←A-M	
	00001	0001	Q_0←1	
	00010	001–	Shift A,Q	1
	11111	001–	A←A-M	
	00010	0010	Q_0←0, Restore A	
(Remainder)	00010	0010 (Quotient)		0

Q.29: Design the carry look ahead adder.

Ans.29: Carry Look-ahead Adder: A carry look-ahead adder reduces the propagation delay by introducing more complex hardware. In this design, the ripple carry design is suitably transformed such that the carry logic over fixed groups of bits of the adder is reduced to two-level logic. Let us discuss the design in detail.



A	B	C	C +1	Condition
0	0	0	0	
0	0	1	0	No Carry
0	1	0	0	Generate
0	1	1	1	
1	0	0	0	No Carry
1	0	1	1	Propogate
1	1	0	1	Carry
1	1	1	1	Generate

Consider the full adder circuit shown above with corresponding truth table. We define two variables as **'carry generate' G_i** and **'carry propagate'P_i** then,

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

The sum output and carry output can be expressed in terms of carry generate G_i and carry propagate P_i as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

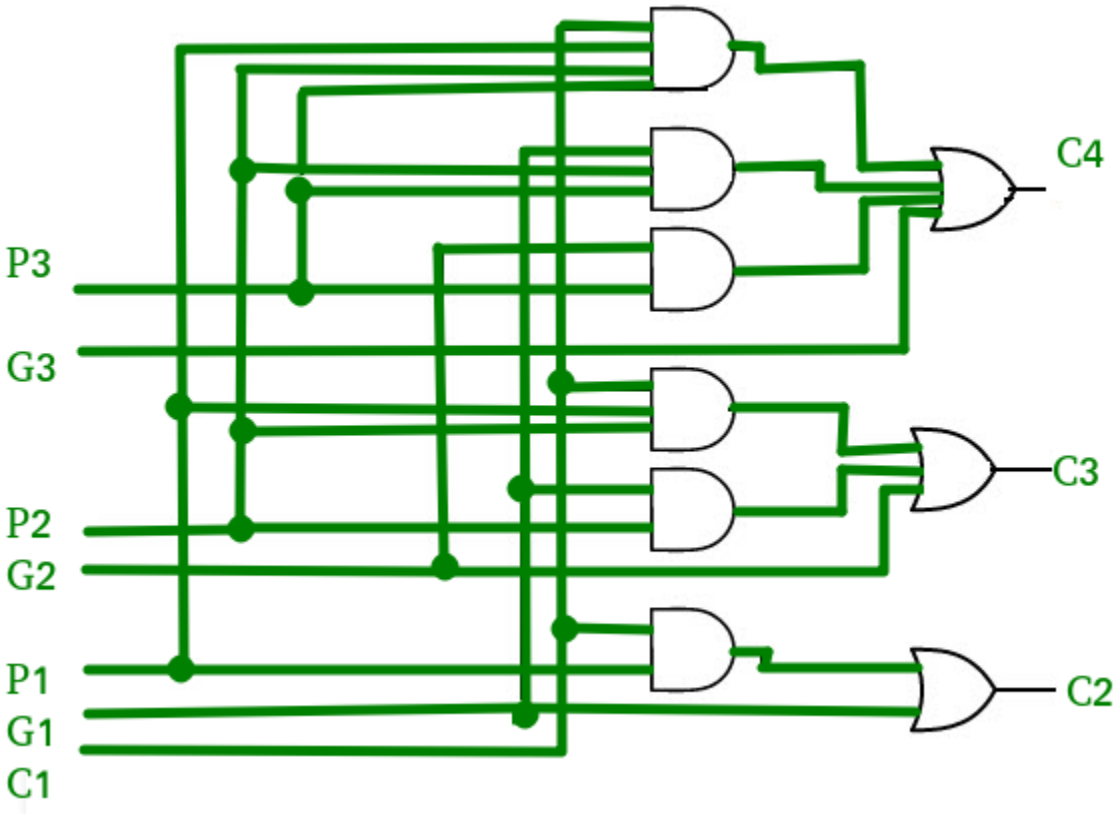
where G_i produces the carry when both A_i, B_i are 1 regardless of the input carry. P_i is associated with the propagation of carry from C_i to C_{i+1}.

The carry output Boolean function of each stage in a 4 stage carry look-ahead adder can be expressed as

$$C_1 = G_0 + P_0 C_{in}$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

From the above Boolean equations we can observe that C_4 does not have to wait for C_3 and C_2 to propagate but actually C_4 is propagated at the same time as C_3 and C_2 . Since the Boolean expression for each carry output is the sum of products so these can be implemented with one level of AND gates followed by an OR gate.

The implementation of three Boolean functions for each carry output (C_3, C_2 and C_4) for a carry look-ahead carry generator shown in below figure.



Advantages and Disadvantages of Carry Look-Ahead Adder:

Advantages –

- The propagation delay is reduced.
- It provides the fastest addition logic.

Disadvantages –

- The Carry Look-ahead adder circuit gets complicated as the number of variables increase.
- The circuit is costlier as it involves more number of hardware.

Q.30: Design carry look ahead generator.

Ans.30: Carry Look-ahead Generator:

Using a look-ahead carry generator we can easily construct a 4-bit parallel adder with a look-ahead carry scheme. Fig. 4.2.27 shows a 4-bit parallel adder with a look-ahead carry scheme. As shown in the Fig. 4.2.27, each sum output requires two exclusive-OR gates. The output of first exclusive-OR gate generates P_i , and the AND gate generates G_i. The carries are generated using look-ahead carry generator and applied as inputs to the second exclusive-OR gate. Other inputs to exclusive-OR gate is P_i . Thus second exclusive-OR gate generates sum outputs. Each output each generated after a delay of two levels of gate. Thus outputs S_2 through S_4 have equal propagation delay times.

C_2, C_3 and C_4 using AND-OI

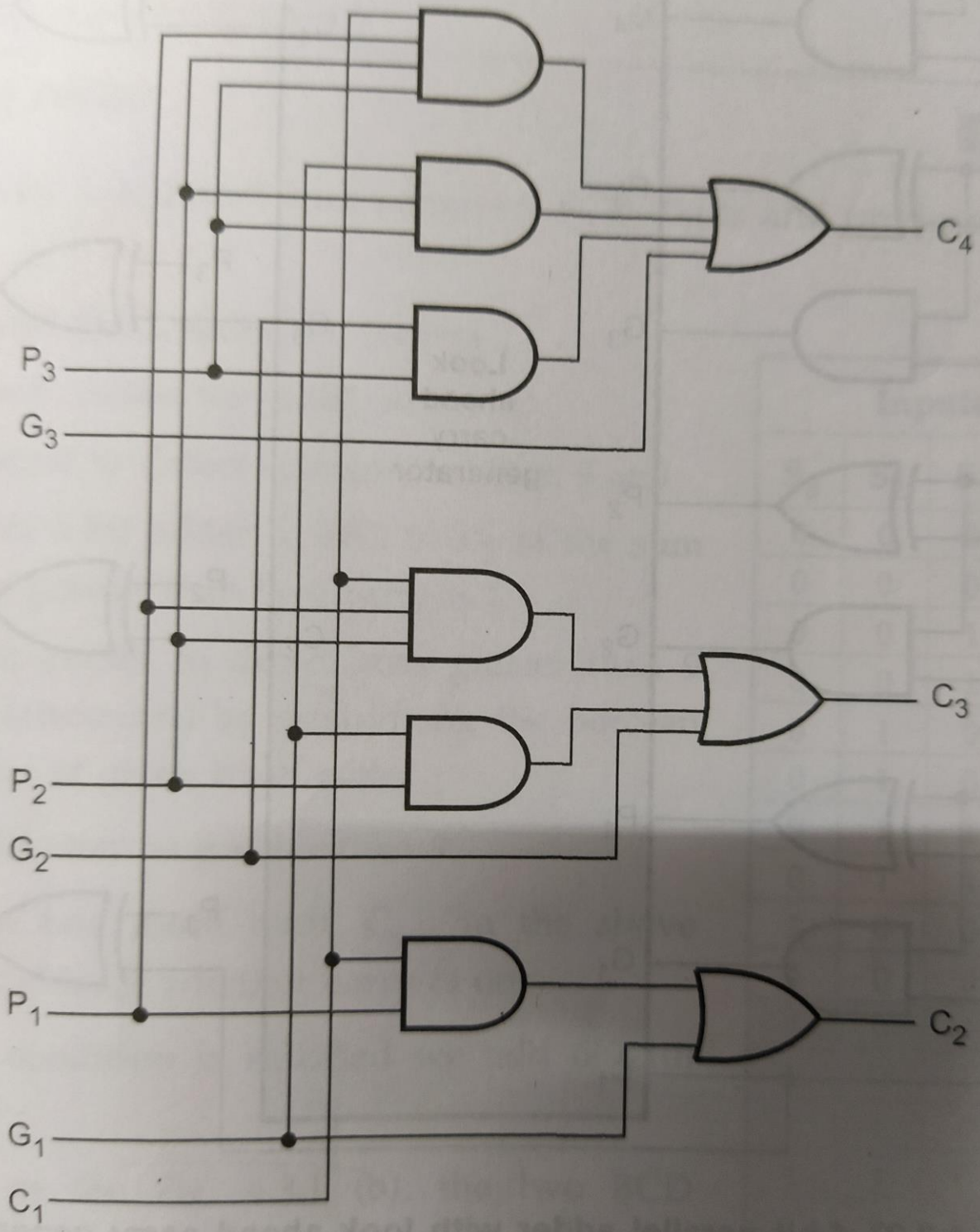


Fig. 4.2.26 Logic diagram of a look-ahead carry generator

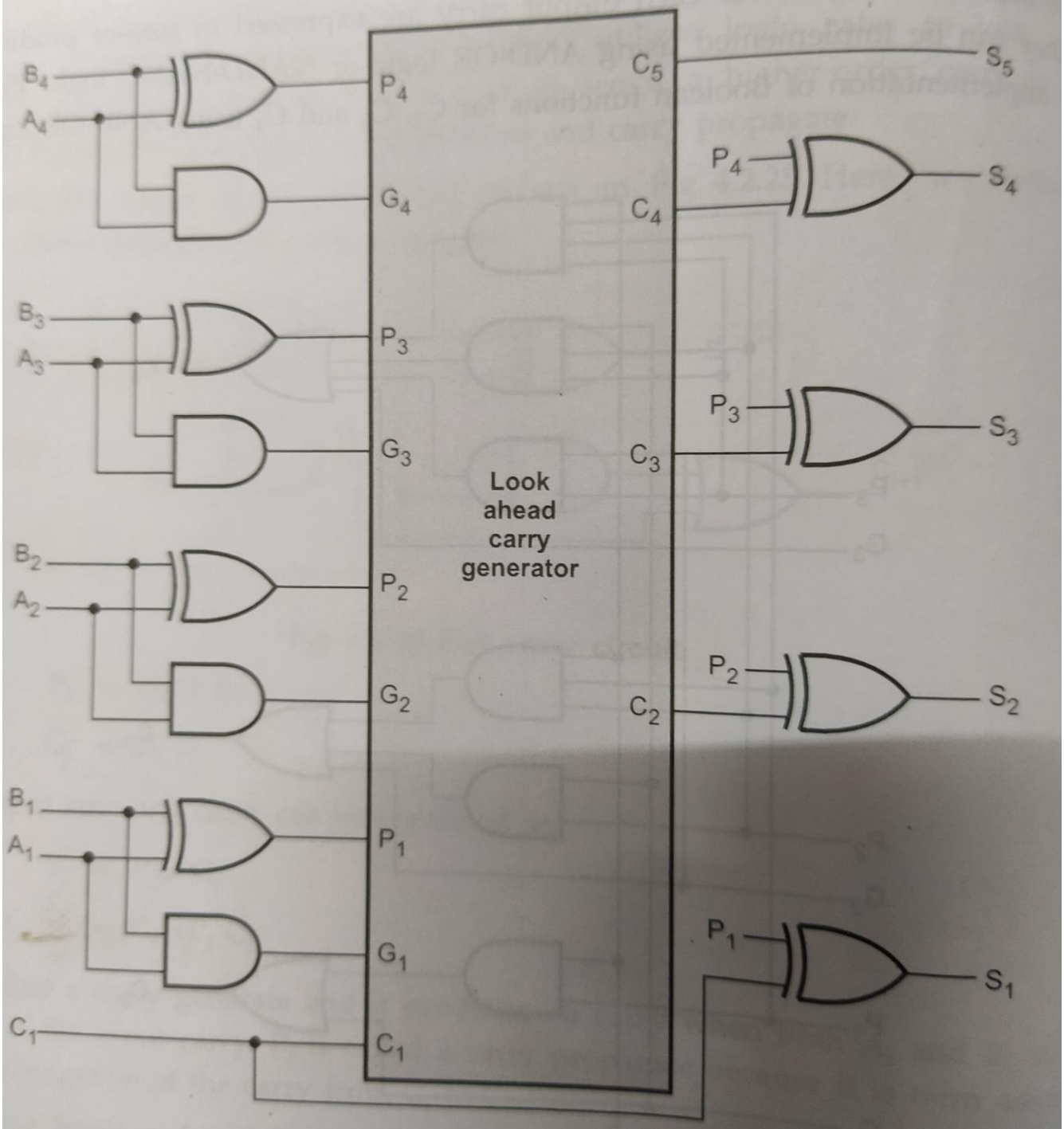


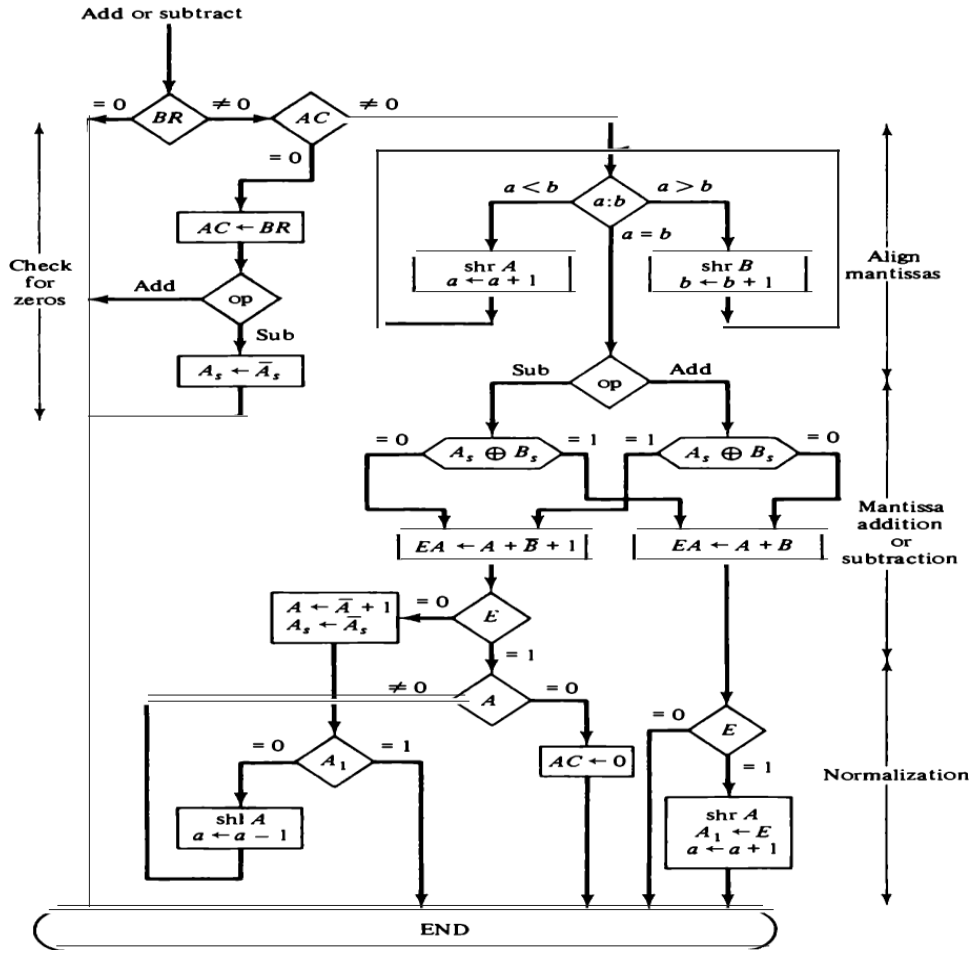
Fig. 4.2.27 4-bit parallel adder with look ahead carry generator

Q.31: Explain floating point addition and subtraction algorithm.

Ans.31: During addition or subtraction, the two floating-point operands are in *AC* and BP. The sum or difference is formed in the ztC. The algorithm can be divided into four consecutive parts:

- Check for zeros.
- Align the mantissas.
- Add or subsact the mantissas.
- Normalize the result.

A floating-point number that is zero cannot be normalized. If this number is used during the compuration, the result may also be zero. Instead of check- ing for zeros during the normalization process we check for zeros at the beginning and terminate the process if necessary. The alignment



The magnitude comparator attached to exponents 4 and 6 provides three outputs that indicate their relaDve magnitude. If the two exponents are equal, we go to perform the arithmetic operation. If the exponents are not equal, the mantissa having the smaller exponent is shifted to the right and its exponent incremented. This process is repeated until the two exponents are equal. The addition and subDaction of the two mantissas is idenhcal to the fixed-point addition and subDacfion algorithm

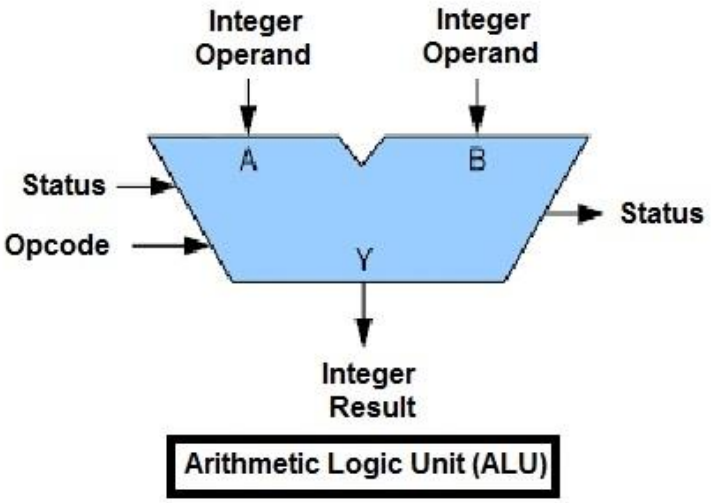
presented in Fig. 102. The magnitude part is added or subtracted depending on the operahon and the signs of the two mantissas. If an overflow occurs when the magnitudes are added, it is transferred into flip-flop E. If E is equal to 1, the bit is transferred into zti and all other bits of zl are shitted right. The exponent must be incre- mented to maintain the correct number. No underflow may occur in this case because the original mantissa that was not shifted during the aligrunent was already in a normalized position. If the magnitudes were subtracted, the result may be zero or may have an underflow. If the mantissa is zero, the entire floating-point number in the *AC* is made zero. Otherwise, the mantissa must have at least one bit that is equal to 1. The mantissa has an underflow if the most significant bit in position A, is 0. In that case, the mantissa is shifted left and the exponent decremented. The bit in zl; is checked again and the process is repeated until it is equal to 1 When zl; **– I,** the mantissa is normalized and the operation is completed.

Q.32: Define Floating point overflow and underflow conditions.

Ans.32: ***Overflow*** is said to occur when the true result of an arithmetic operation is finite but larger in magnitude than the largest floating point number which can be stored using the given precision. nce overflow has occurred, an infinity value can be represented and propagated through a calculation.
Underflow is said to occur when the true result of an arithmetic operation is smaller in magnitude (infinitesimal) than the smallest normalized floating point number which can be stored. Overflow can't be ignored in calculations whereas underflow can effectively be replaced by zero. Underflow occurs in fl. pt. representations when a number is to small (close to 0) to be represented. (show number line!)

Q.33: Explain the various steps required to design ALU.

Ans.33: Inside a computer, there is an Arithmetic Logic Unit (ALU), which is capable of performing logical operations (e.g. AND, OR, Ex-OR, Invert etc.) in addition to the arithmetic operations (e.g. Addition, Subtraction etc.). The control unit supplies the data required by the ALU from memory, or from input devices, and directs the ALU to perform a specific operation based on the instruction fetched from the memory. ALU is the "calculator" portion of the computer.



An arithmetic logic unit(ALU) is a major component of the central processing unit of the a computer system. It does all processes related to arithmetic and logic operations that need to be done on instruction words. In some microprocessor architectures, the ALU is divided into the arithmetic unit (AU) and the logic unit (LU). An ALU can be designed by engineers to calculate many different operations. When the operations become more and more complex, then the ALU will also become more and more expensive and also takes up more space in the CPU and dissipates more heat. That is why engineers make the ALU powerful enough to ensure that the CPU is also powerful and fast, but not so complex as to become prohibitive in terms of cost and other disadvantages. ALU is also known as an Integer Unit (IU). The arithmetic logic unit is that part of the CPU that handles all the calculations the CPU may need. Most of these operations are logical in nature. Depending on how the ALU is designed, it can make the CPU more powerful, but it also consumes more energy and creates more heat. Therefore, there must be a balance between how powerful and complex the ALU is and how expensive the whole unit becomes. This is why faster CPUs are more expensive, consume more power and dissipate more heat.

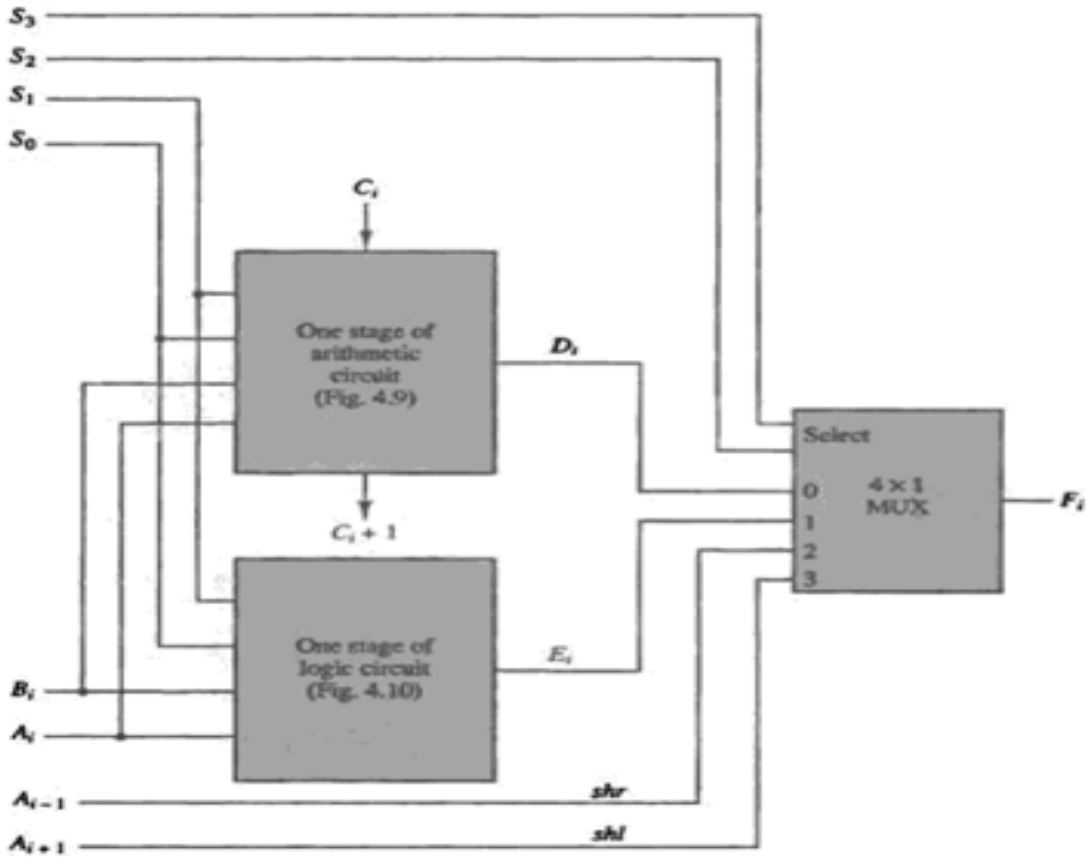
Different operation as carried out by ALU can be categorized as follows –

- **logical operations**: These include operations like AND, OR, NOT, XOR, NOR, NAND, etc.
- **Bit-Shifting Operations**: This pertains to shifting the positions of the bits by a certain number of places either towards the right or left, which is considered a multiplication or division operations.
- **Arithmetic operations**: This refers to bit addition and subtraction. Although multiplication and division are sometimes used, these operations are more expensive to make. Multiplication and subtraction can also be done by repetitive additions and subtractions respectively.

The circuit whose one stage is specified in Fig. provides eight arithmetic operation, four logic operations, and two shift operations. Each opera- non is selected with the five vanables S_3, S_2, S_1, $S_0$and C_{in}. The input carry C_{in} is used for selecting an arithmetic operation only.
Table lists the 14 operations of the ALU. The first eight are arithmetic operations and are selected with S_3S_2= 00. The next four are logic operations and are selected with S_3S_2 = 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 S_3S_2=10 and 11. The other three selection inputs have no effect on the shift.

Operation select						
S_3	S_2	S_1	S_0	C_{in}	Operation	Function
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	×	$F = A \wedge B$	AND
0	1	0	1	×	$F = A \vee B$	OR
0	1	1	0	×	$F = A \oplus B$	XOR
0	1	1	1	×	$F = \overline{A}$	Complement A
1	0	×	×	×	F = shr A	Shift right A into F
1	1	×	×	×	F = shl A	Shift left A into F

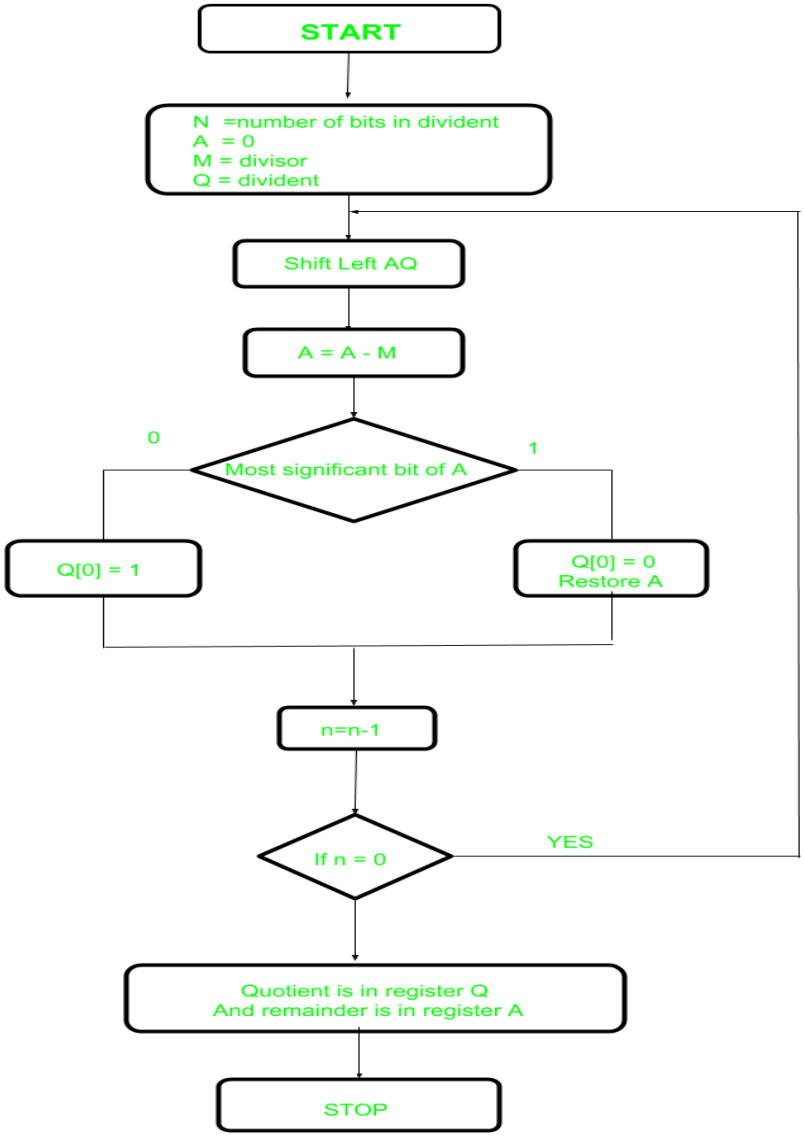


Q.34: Explain steps to divide two signed numbers.

Ans.34: Here, register Q contain quotient and register A contain remainder. Here, n-bit dividend is loaded in Q and divisor is loaded in M. Value of Register is initially kept 0 and this is the register whose value is restored during iteration due to which it is named Restoring.

- **Step-1:** First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)
- **Step-2:** Then the content of register A and Q is shifted right as if they are a single unit

- **Step-3:** Then content of register M is subtracted from A and result is stored in A
- **Step-4:** Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e the value of A before the subtraction with M
- **Step-5:** The value of counter n is decremented
- **Step-6:** If the value of n becomes zero we get of the loop otherwise we repeat from step 2
- **Step-7:** Finally, the register Q contain the quotient and A contain remainder

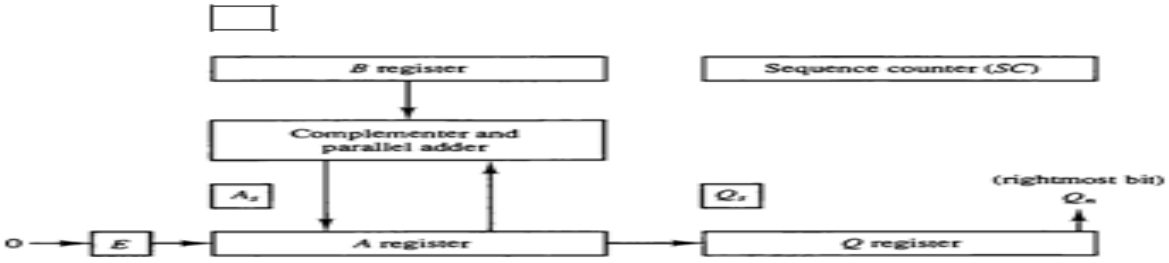


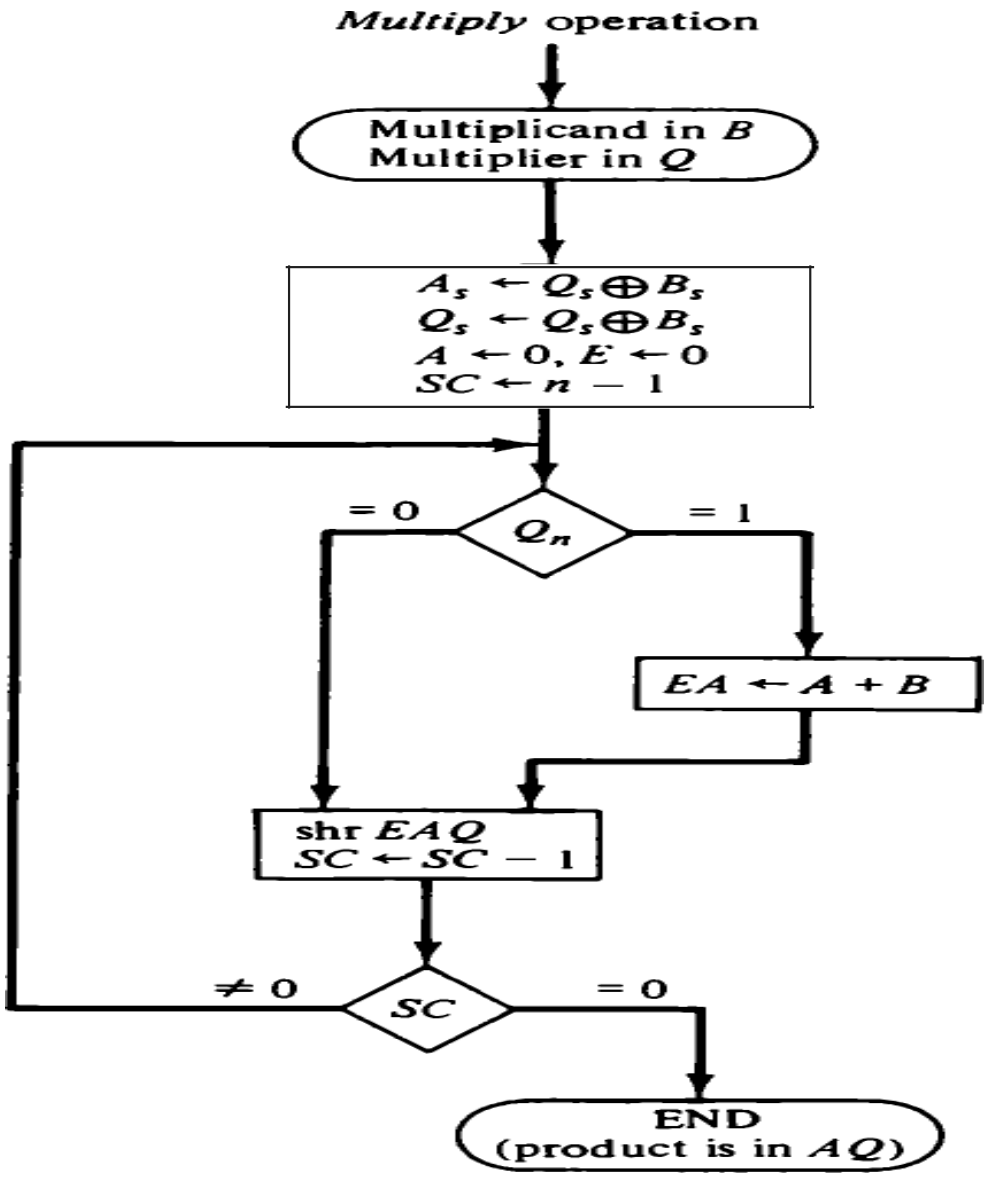
Q.35: Explain the process of dividing 7/3 using restoring method.
Ans.35:

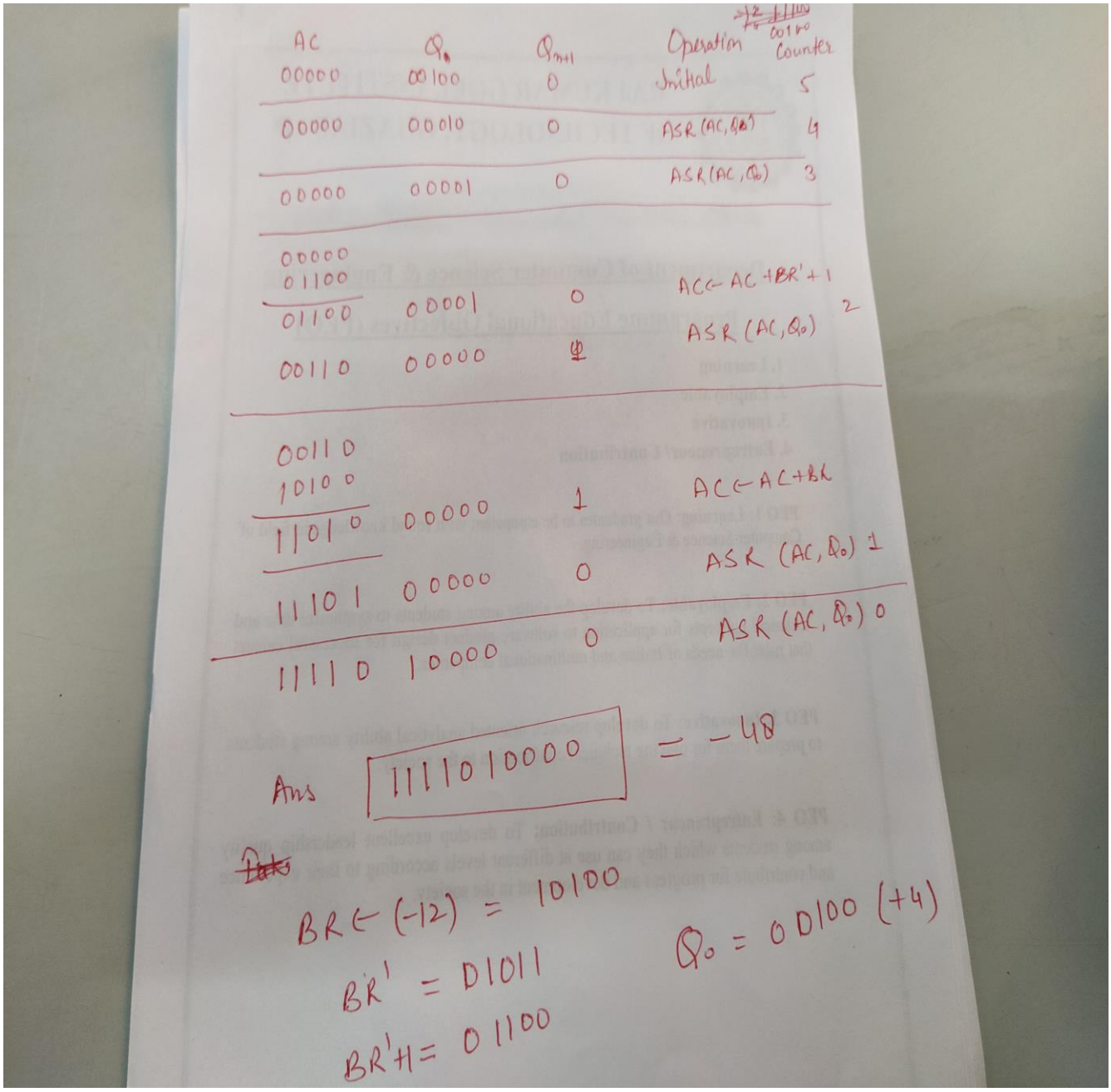
A	Q	
0000	0111	Initial value
0000	1110	Shift
1101		Use twos complement of 0011 for subtraction
1101		Subtract
0000	1110	Restore, set $Q_0 = 0$
0001	1100	Shift
1101		
1110		Subtract
0001	1100	Restore, set $Q_0 = 0$
0011	1000	Shift
1101		
0000	1001	Subtract, set $Q_0 = 1$
0001	0010	Shift
1101		
1110		Subtract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

Q.36: Explain signed multiplication algorithm with the help of flow chart. Implement multiplication of (+12)*(-4) using Unsigned multiplication algorithm
Ans.36: When multiplication is implemented in a digital computer, it is convenient to change the process slightly. First, instead of providing registers to store and add simultaneously as many binary numbers as there are bits in the multiplier, it is convenient to provide an adder for the summation of only two binary numbers and successively accumulate the partial products in a register. Second, instead of shifting the multiplicand to the left, the partial product is shifted to the right, which results in leaving the partial product and the multiplicand in the required relative positions. Third, when the corresponding bit of the multiplier is 0, there is no need to add all zeros to the partial product since it will not alter its value. The hardware for multiplication consists of the equipment shown in Fig. 10-1 plus two more registers. These registers together with registers A and 8 are shown in Fig. 105. The multiplier is stored in the Q register and its sign in Q,. The sequence counter SC is initially set to a number equal to the number of bits in the multiplier. The counter is decremented by 1 after forming each partial product. When the content of the counter reaches zero, the product is formed and the process stops.
Initially, the multiplicand is in register *B* and the multiplier in Q. The sum *ot A* and B forms a partial product which is transferred to the *£A* register. Both partial product and multiplier are shifted to the right. This shift will be denoted by the statement shr *EAQ to* designate the right shift depicted in Fig. 10-5. The least significant bit of *A is s M:te6 into* the most significant position of Q, the bit from £ is shifted into the most significant position of A, and 0 is shifted into After the shift, one bit of the partial product is shifted into Q, pushing the multiplier bits one position to the right. In this manner, the rightmost flip-flop in register Q, designated by Q,, will hold the bit of the multiplier, which must be inspected next.

Figure 10-5 Hardware for multiply operation.







Q.37: Design a 4 bit arithmetic circuit using a suitable block diagram.

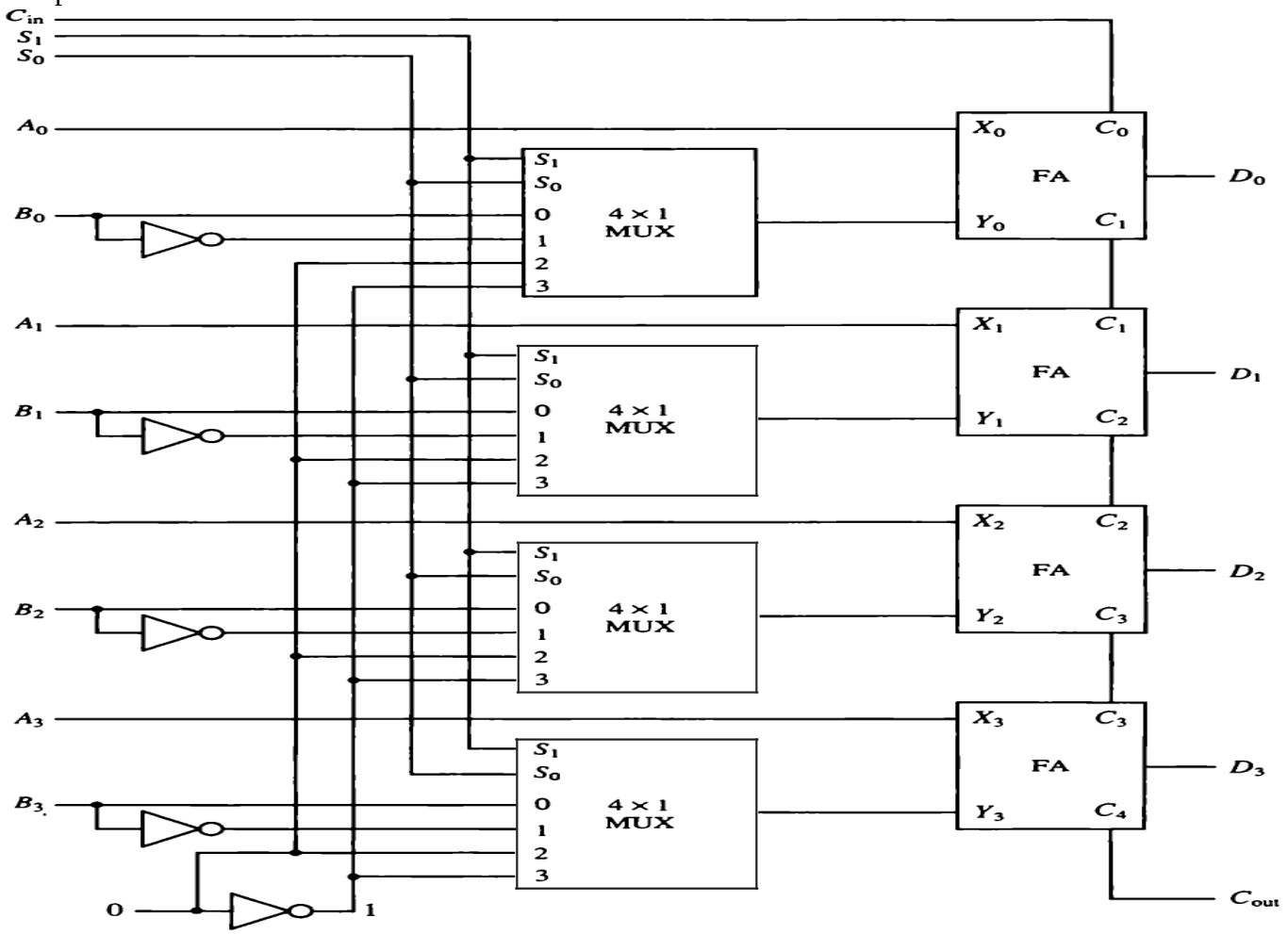
Ans.37: : The arithmetic micro-operations listed in Table 4-3 can be implemented in one composite arithmetic circuit. The basic component of an arithmetic circuit is the parallel adder. By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.

The diagram of a 4-bit arithmetic circuit is shown in Fig. 4-9. It has four hdl-adder circuits that constitute the 4-bit adder and four multiplexers for choosing different operations. There are two 4-bit inputs A and B and a 4-bit output D. The four inputs from Ago directly to the X inputs of the binary adder. Each of the four inputs from B are connected to the data inputs of the multiplexers. The multiplexers data inputs also receive the complement of B. 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. The four multiplexers are controlled by two selection inputs, S_1; and S_0 . The input carry C_{in} goes to the carry input of the FA in the least significant position. The other carries are connected from one stage to the next.
The output of the binary adder is calculated from the following arithmetic

$$D = A+Y+ Cin$$

where *A is* the 4-bit binary number at the X inputs and Y is the 4-bit binary number at the 1' inputs of the binary adder. C_{in} is the input carry, which can be equal to 0 or 1. Note that the symbol + in the equation above denotes an arithmetic plus. By controlling the value of Y with the two selection inputs , S_1; and S_0 and making Cin equal to 0 or 1, it is possible to generate the eight arithmetic micro-operations listed in Table 4-4.



Q.38: Design a 4 bit Bidirectional shifter with the help of multiplexer.

Ans.38: Bidirectional Shift Register with Parallel Load

A register capable of shifting in one direction only is called a unidirectional shift register. A register that can shift in both directions is called a bidirectional shift register. Some shift registers provide the necessary input and output terminals for parahel transfer. The most general shift register has all the capabilities listed below. Others may have some of these capabilities, with at least one shift operation.

1. An input for clock pulses to synchronize all operations.
2. A shift-right operation and a serial input line associated with the shift- right.
3. A shift-left operation and a serial input line associated with the shift-left.
4. A parallel load operation and n input lines associated with the parahel transfer.
5. ii parallel output lines.
6. A control state that leaves the information in the register unchanged even though clock pulses are applied continuously.

A 4-bit bidirectional shift register with parallel load is shown in Fig. 2-9. Each stage consists of a D flip-flop and a 4 x 1 multiplexer. The two selection inputs 5, and S select one of the multiplexer data inputs for the D flip-flop. The selection lines control the mode of operation of the register according to the function table shown in Table 2-4. When the mode control S,S - 00, data input 0 of each multiplexer is selected. This condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock transition transfers into each flip-flop the binary value it held previously, and no change of state occurs. When S;S - 01, the terminal marked 1 in each mulfiplexer has a path to the D input of the corresponding flip-flop. This causes a shift-right operation, with the serial input data transferred into flip-flop A and the content of each flip-flop zl, .; transferred into flip-flop A_i for i = 1, 2, 3. When SiS = 10 a shift-left operation results, with the other serial input data going into flip-flop *A* and the content of flip-flop *A*, + *i* transferred into flip-flop zl, for i = 0, 1, 2. When SiS - 11, the binary information from each input I through I is transferred into the corresponding flip-flop, resulting in a parallel load operation. Note that the way the diagram is drawn, the shirt-right operafion shifts the contents of the register in the down direction while the shift left operation causes the contents of the register to shift in the upward direction.

TABLE 2H Function Table for Register of Fig. 2-9

14odecontel

$S_{\ldots}$	S	Re Meropera fon
0	0	No change
0	1	Shift right (down)
1	0	Shift left (up)
1	1	Parallel load

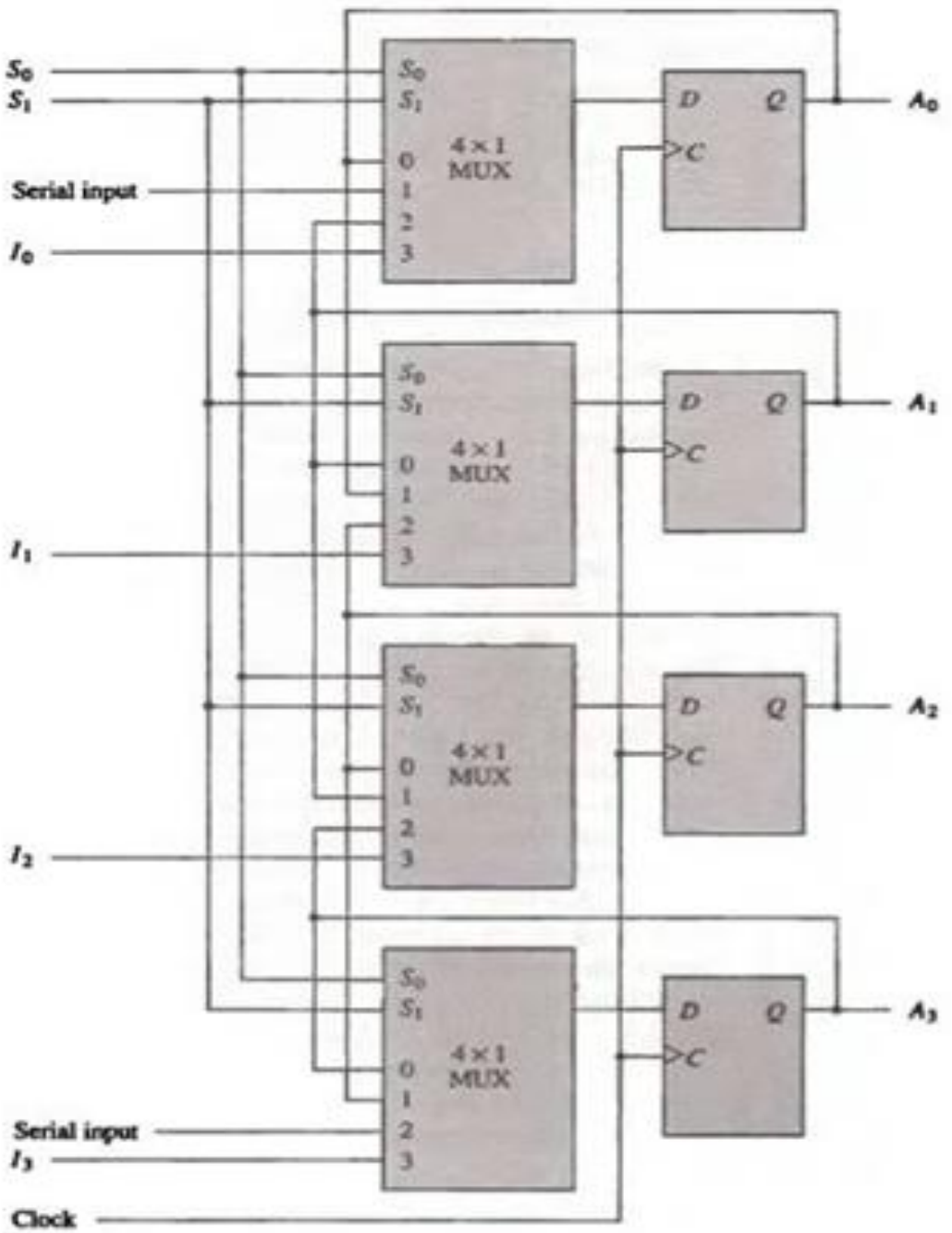


Figure 2-9 Bidirectional shift register with parallel load.

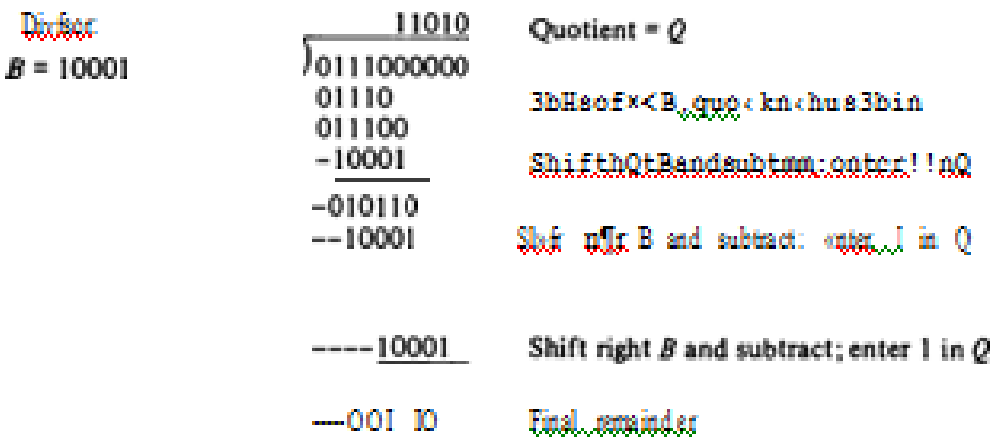
Q.39: Explain divide overflow.

Ans.39: Divide Overflow

The division operation may result in a quotient with an overflow. This is not a problem when working with paper and pencil but is critical when the operation is implemented with hardware. This is because the length of regis- ters is finite and will not hold a number that exceeds the standard length. To see this, consider a system that has 5-bit registers. We use one register to hold the divisor and two registers to hold the dividend. From the example of Fig. 10-11 we note that the quotient will consist of six bits if the five most significant bits of the dividend constitute a number greater than the divisor. The quotient is to be stored in a standard

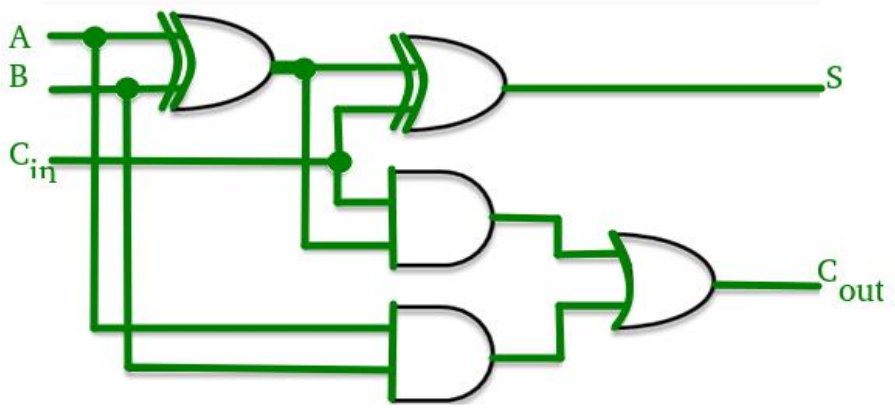
5-bit register, so the overflow bit will require one more flip-flop for storing the sixth bit. This divide-overflow condition must be avoided in normal computer operations because the entire quo- tient will be too long for transfer into a memory unit that has words of standard length, that is, the same as the length of registers. Provisions to ensure that this condition is detected must be included in either the hardware or the software of the computer, or in a combination of the two. When the dividend is twice as long as the divisor, the condition for overflow can be stated as follows: A divide-overflow condition occurs if the high-order half bits of the dividend constitute a number greater than or equal to the divisor. Another problem associated with division is the fact that a division by zero must be avoided. The divide-overflow condition takes care of this condition as well. This occurs because any dividend will be greater than or equal to a divisor which is equal to zero. Overflow condition is usually detected when a special flip-flop is set. We will call it a divide-overflow flip-flop and label it DVF.

Figure 10-1 Example of binary division.



Q.40: Describe carry look ahead in digital circuits.

Ans.40: **Carry Look-ahead Adder:** A carry look-ahead adder reduces the propagation delay by introducing more complex hardware. In this design, the ripple carry design is suitably transformed such that the carry logic over fixed groups of bits of the adder is reduced to two-level logic. Let us discuss the design in detail.



A	B	C	C +1	Condition
0	0	0	0	
0	0	1	0	No Carry
0	1	0	0	Generate
0	1	1	1	
1	0	0	0	No Carry
1	0	1	1	Propogate
1	1	0	1	Carry
1	1	1	1	Generate

Consider the full adder circuit shown above with corresponding truth table. We define two variables as **'carry generate' G_i** and **'carry propagate'P_i** then,

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

The sum output and carry output can be expressed in terms of carry generate G_i and carry propagate P_i as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

where G_i produces the carry when both A_i, B_i are 1 regardless of the input carry. P_i is associated with the propagation of carry from C_i to C_{i+1}.

The carry output Boolean function of each stage in a 4 stage carry look-ahead adder can be expressed as

$$C_1 = G_0 + P_0 C_{in}$$
$$C_2 = G_1 + P_1 C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$
$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

From the above Boolean equations we can observe that C_4 does not have to wait for C_3 and C_2 to propagate but actually C_4 is propagated at the same time as C_3 and C_2 . Since the Boolean expression for each carry output is the sum of products so these can be implemented with one level of AND gates followed by an OR gate.

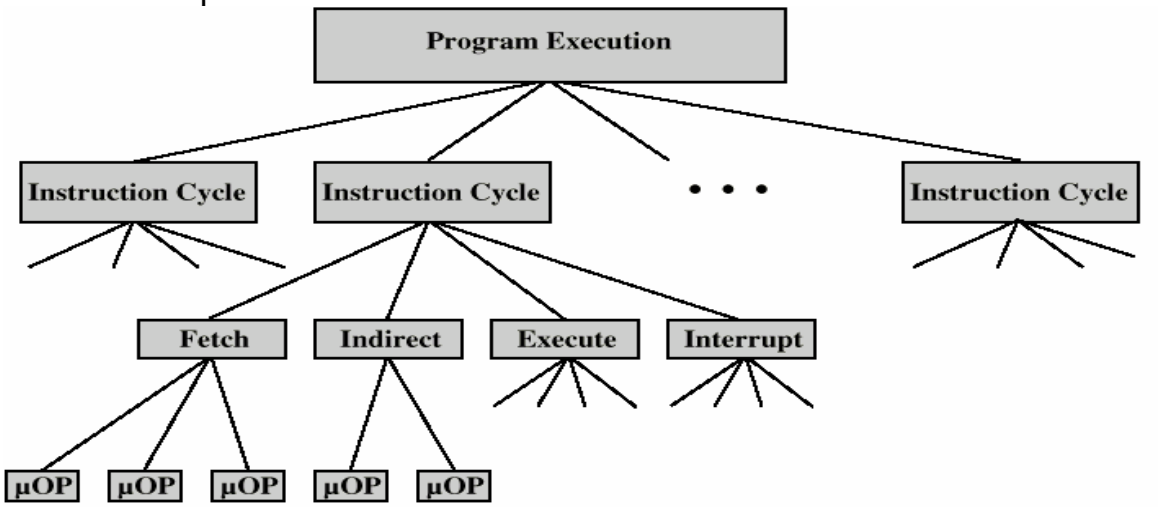
CSE 2nd Year KCS302 (Computer Organization) Unit 3 Solutions

Q.No.	Question Description
41	**Define CISC ? Explain it with its characteristics.** **Complex Instruction Set Architecture (CISC) –** The main idea is to make hardware complex as a single instruction will do all loading, evaluating and storing operations just like a multiplication command will do stuff like loading data, evaluating and storing it. **Characteristic of CISC –** 1. Complex instruction, hence complex instruction decoding. 2. Instruction are larger than one word size. 3. Instruction may take more than single clock cycle to get executed. 4. Less number of general purpose register as operation get performed in memory itself. 5. Complex Addressing Modes. 6. More Data types. Reduced Set Instruction Set Architecture (RISC) – The main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating and storing operations just like a load command will load data, store command will store the data. Characteristic of RISC – Simpler instruction, hence simple instruction decoding. Instruction come under size of one word. Instruction take single clock cycle to get executed. More number of general purpose register. Simple Addressing Modes. Less Data types. Pipeling can be achieved.
42	**A computer has 16 registers, an ALU with 32 operations, and a shifter with eight operations,**

	all connected to a common bus system . **(i) Formulate a Control Word for a micro-operation.** **(ii) Specify the number of bits in each field of the control word and give a general encoding scheme.** **(iii) Show the bits of the control word that specify the micro-operation R4 <- R5 + R6.** a. 16 registers need 4 bits, ALU operations need 5 bits, the shifter needs 3 bits. To encode all operations 20 bits are needed. b. SRC1 (4 bits) | SRC2 (4 bits) | DEST (4 bits) | ALU (5 bits) | SHIFT (3 bits) c. 0101 (R5) | 0110 (R6) | 0100 (R4) | 00100 (ADD) | 000 (SHIFT)
43	**Define microoperation? How can microoperation be used for execution of an instruction? Explain with the help of an example.** **Micro-Operations** The operations executed on data stored in registers are called micro-operations. A micro-operation is an elementary operation performed on the information stored in one or more registers. **Example:** Shift, count, clear and load. **Types of Micro-Operations** The micro-operations in digital computers are of 4 types: 1. Register transfer micro-operations transfer binary information from one register to another. 2. Arithmetic micro-operations perform arithmetic operations on numeric data stored in registers. 3. Logic micro-operations perform bit manipulation operation on non-numeric data stored in registers. 4. Shift micro-operations perform shift micro-operations performed on data.

The execution of a program consists of the sequential execution of instructions. Each instruction is executed during an instruction cycle made up of shorter sub-cycles (e.g., fetch, indirect, execute, interrupt). The performance of each sub-cycle involves one or more shorter operations, that is, micro-operations.

Micro-operations are the functional, or atomic, operations of a processor. In this section, we will examine micro-operations to gain an understanding of how the events of any instruction cycle can be described as a sequence of such micro-operations .



1.1 The Fetch Cycle

We begin by looking at the fetch cycle, which occurs at the beginning of each instruction cycle and causes an instruction to be fetched from memory. Four registers are involved:

- Memory address register (MAR): Is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.
- Memory buffer register (MBR): Is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from memory.
- Program counter (PC): Holds the address of the next instruction to be fetched.

- Instruction register (IR): Holds the last instruction fetched.

Let us look at the sequence of events for the fetch cycle from the point of view of its effect on the processor registers

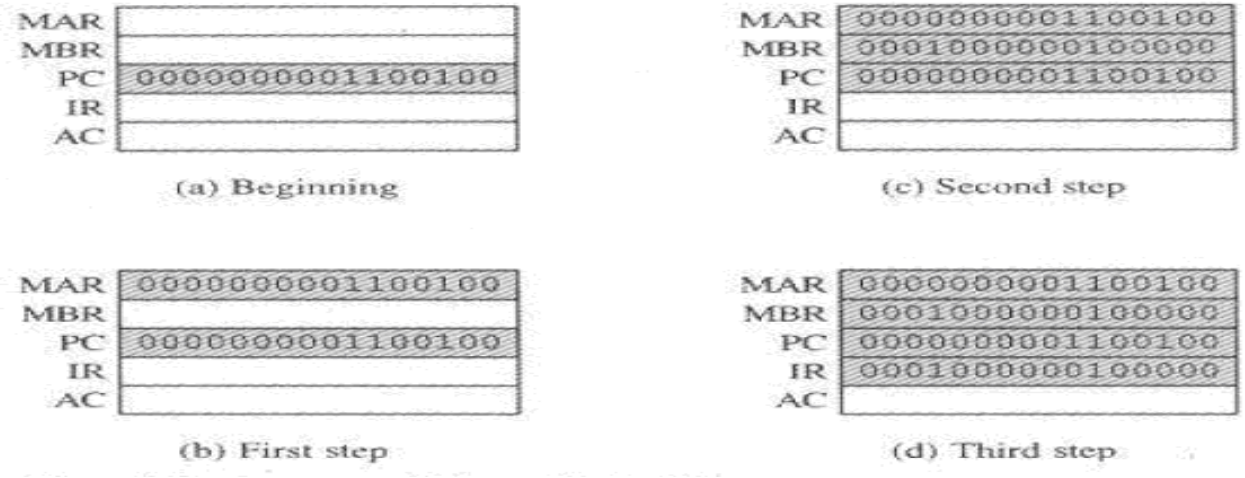


Figure 6.2 Sequence of Events, Fetch Cycle

- At the beginning of the fetch cycle, the address of the next instruction to be executed is in the program counter (PC); in this case, the address is 1100100.
- The first step is to move that address to the memory address register (MAR) because this is the only register connected lo the address lines of the system bus.
- The second step is to bring in the instruction. The desired address (in the MAR) is placed on the address bus, the control unit issues a READ command on the control bus, and the result appears on the data bus and is copied into the memory buffer register (MBR). We also need to increment the PC by 1 to get ready for the next instruction. Because these two actions (read word from memory, add 1 to PC) do not interfere with each other, we can do them simultaneously to save time.
- The third step is to move the contents of the MBR to the instruction register (IR). This frees up the MBR for use during a possible indirect cycle.

Thus, the simple fetch cycle actually consists of three steps and four micro-operations. Each micro-operation involves the movement of data into or out of a register. So long as these movements do not interfere with one

another, several of them can take place during one step, saving lime. Symbolically, we can write this sequence of events as follows:
t1: MAR <= (PC)
t2: MBR <= Memory
PC <= (PC) + l
t3: IR <= (MBR)
where l is the instruction length. We need to make several comments about this sequence. We assume that a clock is available for timing purposes and that it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit. The notation (t1, t2, t3) represents successive time units. In words, we have

- First time unit: Move contents of PC to MAR.
- Second time unit:
  - Move contents of memory location specified by MAR to MBR.
  - Increment by l the contents of the PC.
- Third time unit: Move contents of MBR to IR.

Note that the second and third micro-operations both take place during the second time unit. The third micro-operation could have been grouped with the fourth without affecting the fetch operation:
t1: MAR <= (PC)
t2: MBR <= Memory
t3: PC <= (PC) + l
IR <= (MBR)
The groupings of micro-operations must follow two simple rules:
1. The proper sequence of events must be followed. Thus (MAR <= (PC)) must precede (MBR <= Memory) because the memory read operation makes use of the address in the MAR.
2. Conflicts must be avoided. One should not attempt to read to and write from the same register in one time unit, because the results would be unpredictable. For example, the micro-operations (MBR <= Memory) and (IR <= MBR) should not occur during the same time unit.

A final point worth noting is that one of the micro-operations involves an addition. To avoid duplication of circuitry, this addition could be performed by the ALU. The use of the ALU may involve additional micro-operations, depending on the functionality of the ALU and the organization of the processor.

1.2 The Indirect Cycle

Once an instruction is fetched, the next step is to fetch source operands. Continuing our simple example, let us assume a one-address instruction format, with direct and indirect addressing allowed. If the instruction specifies an indirect address, then an indirect cycle must precede the execute cycle. The data flow includes the following micro-operations:

t1: MAR <= (IR (Address))
t2: MBR <= Memory
t3: IR(Address) <= (MBR(Address))

The address field of the instruction is transferred to the MAR. This is then used to fetch the address of the operand. Finally, the address field of the IR is updated from the MBR, so that it now contains a direct rather than an indirect address.

The IR is now in the same state as if indirect addressing had not been used, and it is ready for the execute cycle. We skip that cycle for a moment, to consider the interrupt cycle.

1.3 The Interrupt Cycle

At the completion of the execute cycle, a test is made to determine whether any enabled interrupts have occurred. If so, the interrupt cycle occurs. The nature of this cycle varies greatly from one machine to another. We present a very simple sequence of events, we have

t1 : MBR <= (PC)
t2 : MAR <= Save_Address
PC <= Routine_Address
t3: Memory <= (MBR)

In the first step, the contents of the PC are transferred to the MBR, so that they can be saved for return from the interrupt. Then the MAR is loaded with the address at which the contents of the PC are to be saved, and the PC is loaded with the address of the start of the interrupt-processing routine. These two actions may each be a single

micro-operation. However, because most processors provide multiple types and/or levels of interrupts, it may lake one or more additional micro-operations to obtain the save_address and the routine_address before they can be transferred to the MAR and PC, respectively. In any case, once this is done, the final step is to store the MBR, which contains the old value of the PC, into memory. The processor is now ready to begin the next instruction cycle.

1.4 The Execute Cycle

The fetch, indirect, and interrupt cycles are simple and predictable. Each involves a small, fixed sequence of micro-operations and, in each ease, the same micro-operations are repealed each time around. This is not true of the execute cycle. For a machine with N different opcodes, there are N different sequences of micro-operations that can occur. Let us consider several hypothetical examples.

First, consider an add instruction:

ADD R1, X

which adds the contents of the location X to register Rl. The following sequence of micro-operations might occur:

t1: MAR <= (IR(address))
t2: MBR <= Memory
t3: Rl <= (Rl) + (MBR)

We begin with the IR containing the ADD instruction. In the first step, the address portion of the IR is loaded into the MAR. Then the referenced memory location is read. Finally, the contents of R1 and MBR are added by the ALU. Again, this is a simplified example. Additional micro-operations may be required to extract the register reference from the IR and perhaps to stage the ALU inputs or outputs in some intermediate registers.Let us look at two more complex examples. A common instruction is increment and skip if zero:

ISZ X

The content of location X is incremented by 1. If the result is 0, the next instruction is skipped. A possible sequence of micro-operations is

t1: MAR <= (CR(address))
t2: MBR <= Memory
t3: MBR <= (MBR) - 1

	t4: Memory <= (MBR) If ((MBR) = 0) then (PC <= (PC) + I) The new feature introduced here is the conditional action. The PC is incremented if (MBR) = 0; this test and action can be implemented as one micro-operation. Note also that this micro-operation can be performed during the same time unit during which the updated value in MBR is stored back to memory. Finally, consider a subroutine call instruction. As an example, consider a branch-and-save-address instruction: BSA X The address of the instruction that follows the BSA instruction is saved in location X, and execution continues al location X - l. The saved address will later be used for return. This is a straightforward technique for providing subroutine calls. the following micro-operations suffice: t1 : MAR <= (IR(address)) MBR <= (PC) t2: PC <= (IR(address)) Memory <= (MBR) t3: PC <= (PC) + I The address in the PC at the start of the instruction is the address of the next instruction in sequence. This is saved at the address designated in Ihe IK. The latter address is also incremented to provide the address of the instruction for the next instruction cycle.
44	**Instruction cycle is divided into subcycles. With the help of diagram, explain the sequence in which subcycles are executed** **The Instruction Cycle –** Different Instruction Cycles: 1. **The Fetch Cycle –** At the beginning of the fetch cycle, the address of the next instruction to be executed is in the *Program Counter*(PC).

Step 1: The address in the program counter is moved to the memory address register(MAR), as this is the only register which is connected to address lines of the system bus.

Step 2: The address in MAR is placed on the address bus, now the control unit issues a READ command on the control bus, and the result appears on the data bus and is then copied into the memory buffer register(MBR). Program counter is incremented by one, to get ready for the next instruction.(These two action can be performed simultaneously to save time)

Step 3: The content of the MBR is moved to the instruction register(IR).

Here 'I' is the instruction length. The notations (t1, t2, t3) represents successive time units. We assume that a clock is available for timing purposes and it emits regularly spaced clock pulses. Each clock pulse defines a time unit. Thus, all time units are of equal duration. Each micro-operation can be performed within the time of a single time unit.
First time unit: Move the contents of the PC to MAR.
Second time unit: Move contents of memory location specified by MAR to MBR. Increment content of PC by I.
Third time unit: Move contents of MBR to IR.
Note: Second and third micro-operations both take place during the second time unit.

2. **The Indirect Cycles –**
   Once an instruction is fetched, the next step is to fetch source operands. *Source Operand* is being fetched by indirect addressing(it can be fetched by any addressing mode, here its done by indirect addressing). Register-based operands need not be fetched. Once the opcode is executed, a similar process may be needed to store the result in main memory.

Step 1: The address field of the instruction is transferred to the MAR. This is used to fetch the address of the operand.
Step 2: The address field of the IR is updated from the MBR.(So that it now contains a direct addressing rather than indirect addressing)
Step 3: The IR is now in the state, as if indirect addressing has not been occurred.

Note: Now IR is ready for the execute cycle, but it skips that cycle for a moment to consider the *Interrupt Cycle* .

3. **The Execute Cycle**

The other three cycles(*Fetch, Indirect and Interrupt*) are simple and predictable. Each of them requires simple, small and fixed sequence of micro-operation. In each case same micro-operation are repeated each time around.
Execute Cycle is different from them. Like, for a machine with N different opcodes there are N different sequence of micro-operations that can occur.
Lets take an hypothetical example :-
consider an add instruction:

We begin with the IR containing the ADD instruction.
Step 1: The address portion of IR is loaded into the MAR.
Step 2: The address field of the IR is updated from the MBR, so the reference memory location is read.
Step 3: Now, the contents of R and MBR are added by the ALU.

Here, the PC is incremented if (MBR) = 0. This test (is MBR equal to zero or not) and action (PC is incremented by 1) can be implemented as one micro-operation.
Note : This test and action micro-operation can be performed during the same time unit during which the updated value MBR is stored back to memory.

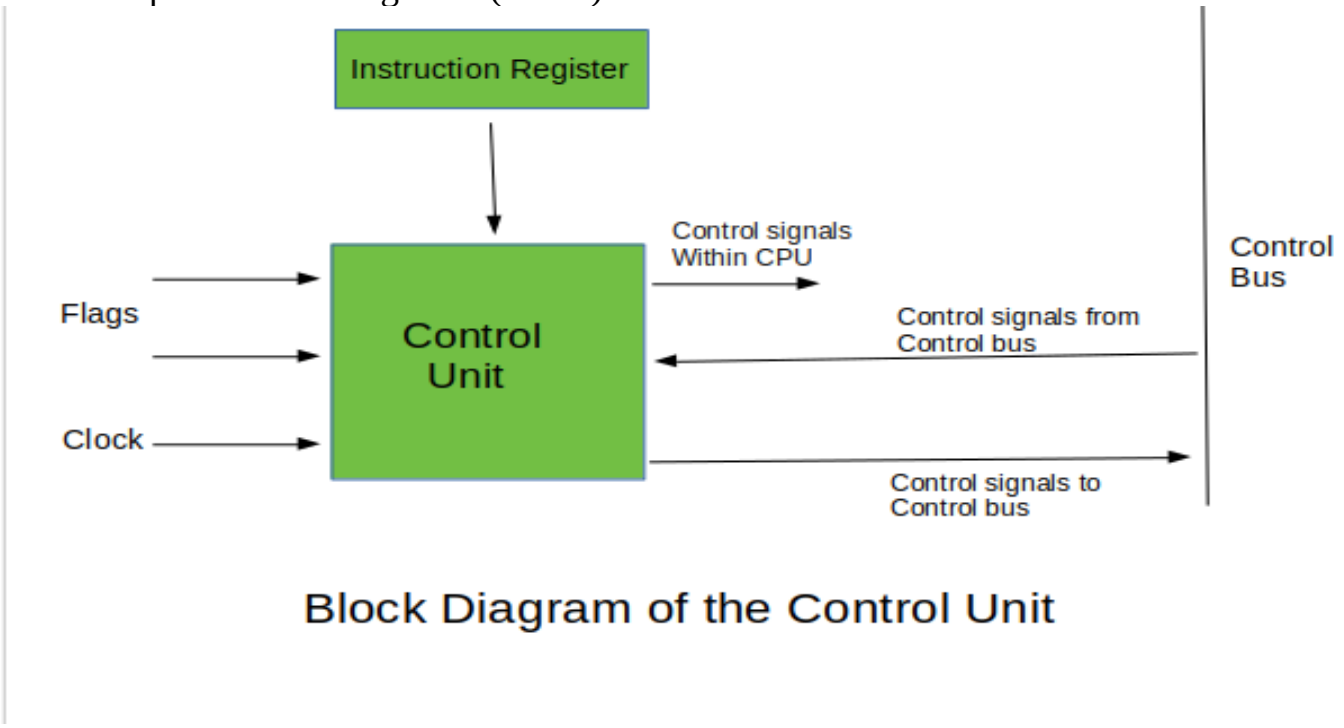
	4. **The Interrupt Cycle**: At the completion of the Execute Cycle, a test is made to determine whether any enabled interrupt has occurred or not. If an enabled interrupt has occurred then Interrupt Cycle occurs. The natare of this cycle varies greatly from one machine to another. Lets take a sequence of micro-operation:- Step 1: Contents of the PC is transferred to the MBR, so that they can be saved for return. Step 2: MAR is loaded with the address at which the contents of the PC are to be saved. PC is loaded with the address of the start of the interrupt-processing routine. Step 3: MBR, containing the old value of PC, is stored in memory. **Note:** In step 2, two actions are implemented as one micro-operation. However, most processor provide multiple types of interrupts, it may take one or more micro-operation to obtain the save_address and the routine_address before they are transferred to the MAR and PC respectively.
45	**Discuss how address sequencing is carried out in microprogrammed organization.** **Address Sequencing •** Microinstructions are stored in control memory in groups, with each group specifying a routine • Each computer instruction has its own microprogram routine to generate the microoperations • The hardware that controls the address sequencing of the control memory must be capable of sequencing the microinstructions within a routine and be able to branch from one routine to another • Steps the control must undergo during the execution of a single computer instruction: o Load an initial address into the CAR when power is turned on in the computer. This address is usually the address of the first microinstruction that activates the instruction fetch routine – IR holds instruction o The control memory then goes through the routine to determine the effective address of the operand – AR holds operand address o The next step is to generate the microoperations that execute the instruction by considering the opcode and applying a mapping o After execution, control must return to the fetch routine by executing an unconditional branch •

	The microinstruction in control memory contains a set of bits to initiate microoperations in computer registers and other bits to specify the method by which the next address is obtained • Conditional branching is obtained by using part of the microinstruction to select a specific status bit in order to determine its condition The status conditions are special bits in the system that provide parameter information such as the carry-out of an adder, the sign bit of a number, the mode bits of an instruction, and i/o status conditions • The status bits, together with the field in the microinstruction that specifies a branch address, control the branch logic • The branch logic tests the condition, if met then branches, otherwise, increments the CAR • If there are 8 status bit conditions, then 3 bits in the microinstruction are used to specify the condition and provide the selection variables for the multiplexer • For unconditional branching, fix the value of one status bit to be one load the branch address from control memory into the CAR • A special type of branch exists when a microinstruction specifies a branch to the first word in control memory where a microprogram routine is located • The status bits for this type of branch are the bits in the opcode • Assume an opcode of four bits and a control memory of 128 locations • The mapping process converts the 4-bit opcode to a 7-bit address for control memory • This provides for each computer instruction a microprogram routine with a capacity of four microinstructions • Subroutines are programs that are used by other routines to accomplish a particular task and can be called from any point within the main body of the microprogram • Frequently many microprograms contain identical section of code • Microinstructions can be saved by employing subroutines that use common sections of microcode • Microprograms that use subroutines must have a provisions for storing the return address during a subroutine call and restoring the address during a subroutine return • A subroutine register is used as the source and destination for the addresses
46	**Describe how control unit of a computer function? Explain with the help of a block diagram** **Control Unit** is the part of the computer's central processing unit (CPU), which directs the operation of the processor. It was included as part of the Von Neumann Architecture by John von Neumann. It is the responsibility of the Control Unit to tell the computer's memory, arithmetic/logic unit and input and output devices how to respond to the

instructions that have been sent to the processor. It fetches internal instructions of the programs from the main memory to the processor instruction register, and based on this register contents, the control unit generates a control signal that supervises the execution of these instructions.

A control unit works by receiving input information to which it converts into control signals, which are then sent to the central processor. The computer's processor then tells the attached hardware what operations to perform. The functions that a control unit performs are dependent on the type of CPU because the architecture of CPU varies from manufacturer to manufacturer. Examples of devices that require a CU are:

- Control Processing Units(CPUs)
- Graphics Processing Units(GPUs)



Block Diagram of the Control Unit

Functions of the Control Unit –

1. It coordinates the sequence of data movements into, out of, and between a processor's many sub-units.

	2. It interprets instructions. 3. It controls data flow inside the processor. 4. It receives external instructions or commands to which it converts to sequence of control signals. 5. It controls many execution units(i.e. ALU, data buffers and registers) contained within a CPU. 6. It also handles multiple tasks, such as fetching, decoding, execution handling and storing results.
47	**Define the following :** **(i) Microoperation** **(ii) Microinstruction** **(iii) Microprogram** **(iv) Microcode** Microoperation - an elementary digital computer operation. Microinstruction - an instruction stored in control memory. Microprogram - a sequence of microinstructions. Microcode - same an microprogram.

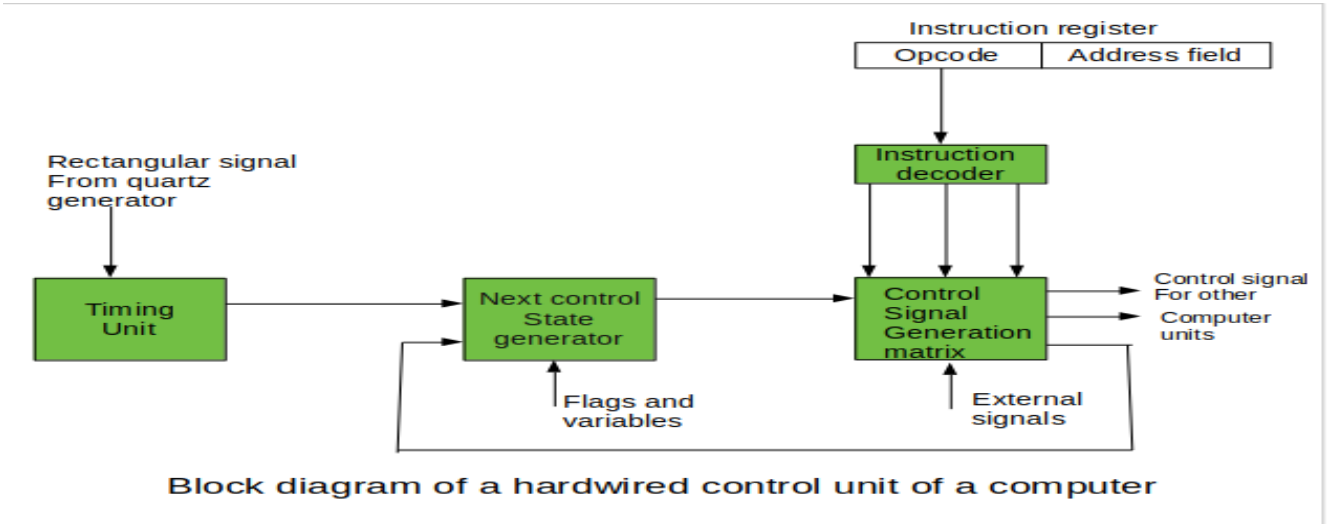
48	**Describe the basic differences between a branch instruction, a call subroutine instruction, and program interrupt?**
	Branch instruction:
	Every time it is not possible to use line-sequence of instruction, sometime we need breakup in our program according to some condition this is nothing but branching and to perform these branching we require some instruction that is called branch instruction
	Eg: MOV R0, num1
	MOV R2, num2
	CMP R0, R1
	JB . next
	Sub R0, R1

	MOV R2, R1
	Next: Sub R1, R0
	MOV R2, R0
	Here the task of IB. is it compairs the two Numbers i.e. R0, and R1, if R0, < R1, then it goes to next and execute the instruction otherwise it execute the instructions just next to it.
	The IB is your branch instruction.
	e.g. : IC, INC, IZ, INZ, IB, INB etc.
	
	Interrupt procedure is similar to Subroutine call, however there are some differences:
	1. The interrupt is usually initiated by external or internal signal rather than from the execution of an instruction.(except for software interrupt)
	2. The address of the interrupt service program is determined by the hardware rather than by the address field of the instruction.
	3. Interrupt procedure stores all information necessary to define the state of the CPU rather than storing only the program counter.
	4. Subroutine call is called by the user through instructions, whereas Interrupt is called by the hardware or any external signal.

49	**Define CISC ? Explain it with its characteristics.** **Complex Instruction Set Architecture (CISC) –** The main idea is to make hardware complex as a single instruction will do all loading, evaluating and storing operations just like a multiplication command will do stuff like loading data, evaluating and storing it. **Characteristic of CISC –** 7. Complex instruction, hence complex instruction decoding. 8. Instruction are larger than one word size. 9. Instruction may take more than single clock cycle to get executed.

	10.Less number of general purpose register as operation get performed in memory itself. 11.Complex Addressing Modes. 12.More Data types. **Reduced Set Instruction Set Architecture (RISC)** – The main idea behind is to make hardware simpler by using an instruction set composed of a few basic steps for loading, evaluating and storing operations just like a load command will load data, store command will store the data. **Characteristic of RISC –** Simpler instruction, hence simple instruction decoding. Instruction come under size of one word. Instruction take single clock cycle to get executed. More number of general purpose register. Simple Addressing Modes. Less Data types. Pipeling can be achieved.
50	**State the meaning of hard-wired control unit? Give various methods to design hardwired control unit. Describe one of the design methods for hardwired control unit with suitable diagrams.** **Hardwired Control Unit –** In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we can not modify the signal generation method without physical change of the circuit structure. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode.

	As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for executive units of the computer. This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals. The matrices are built in a similar way as a programmable logic arrays. Control signals for an instruction execution have to be generated not in a single time point but during the entire time interval that corresponds to the instruction execution cycle. Following the structure of this cycle, the suitable sequence of internal states is organized in the control unit. A number of signals generated by the control signal generator matrix are sent back to inputs of the next control state generator matrix. This matrix combines these signals with the timing signals, which are generated by the timing unit based on the rectangular patterns usually supplied by the quartz generator. When a new instruction arrives at the control unit, the control units is in the initial state of new instruction fetching. Instruction decoding allows the control unit enters the first state relating execution of the new instruction, which lasts as long as the timing signals and other input signals as flags and state information of the computer remain unaltered. A change of any of the earlier mentioned signals stimulates the change of the control unit state. This causes that a new respective input is generated for the control signal generator matrix. When an external signal appears, (e.g. an interrupt) the control unit takes entry into a next control state that is the state concerned with the reaction to this external signal (e.g. interrupt processing). The values of flags and state variables of the computer are used to select suitable states for the instruction execution cycle. The last states in the cycle are control states that commence fetching the next instruction of the program: sending the program counter content to the main memory address buffer register and next, reading the instruction word to the instruction register of computer. When the ongoing instruction is the stop instruction that ends program execution, the control unit enters an operating system state, in which it waits for a next user directive.



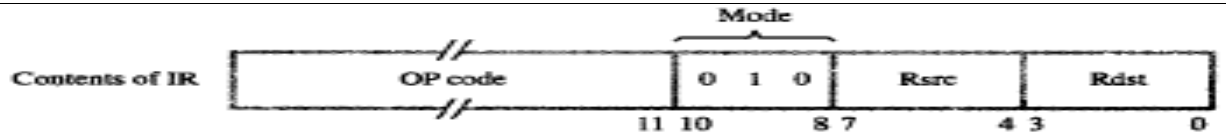
Block diagram of a hardwired control unit of a computer

51 **Explain wide-branch addressing? Explain with example.**

Wide Branch Addressing •

The instruction-decoder(InstDec) generates the starting-address of the microroutine that implements the instruction that has just been loaded into the IR. • Here, register IR contains the Add instruction, for which the instruction decoder generates the microinstruction address 101. (However, this address cannot be loaded as is into the μPC). • The source-operand can be specified in any of several addressing-modes. The bit-Oring technique can be used to modify the starting-address generated by the instruction-decoder to reach the appropriate path. Use of WMFC • WMFC signal is issued at location 112 which causes a branch to the microinstruction in location 171. •

	WMFC signal means that the microinstruction may take several clock cycles to complete. If the branch is allowed to happen in the first clock cycle, the microinstruction at location 171 would be fetched and executed prematurely. To avoid this problem, WMFC signal must inhibit any change in the contents of the Mpc during the waiting-period. Detailed Examination • Consider Add (Rsrc)+,Rdst; which adds Rsrc content to Rdst content, then stores the sum in Rdst and finally increments Rsrc by 4 (i.e. auto-increment mode). • In bit 10 and 9, bit-patterns 11, 10, 01 and 00 denote indexed, auto-decrement, auto-increment and register modes respectively. For each of these modes, bit 8 is used to specify the indirect version. • The processor has 16 registers that can be used for addressing purposes; each specified using a 4-bit-code. • There are 2 stages of decoding: 1) The microinstruction field must be decoded to determine that an Rsrc or Rdst register is involved. 2) The decoded output is then used to gate the contents of the Rsrc or Rdst fields in the IR into a second decoder, which produces the gating-signals for the actual registers R0 to R15. The microprogram requires several branch microinstructions which perform no useful operation. Thus, they detract from the operating speed of the computer. • Solution: Include an address-field as a part of every microinstruction to indicate the location of the next microinstruction to be fetched. (This means every microinstruction becomes a branch microinstruction). • The flexibility of this approach comes at the expense of additional bits for the address-field. • Advantage: Separate branch microinstructions are virtually eliminated. There are few limitations in assigning addresses to microinstructions. There is no need for a counter to keep track of sequential addresse. Hence, the Mpc is replaced with a Mar (Microinstruction Address Register). {which is loaded from the next-address field in each microinstruction}. • The next-address bits are fed through the OR gate to the Mar, so that the address can be modified on the basis of the data in the IR, external inputs and condition-codes. • The decoding circuits generate the starting-address of a given microroutine on the basis of the opcode in the IR.



Address (octal)	Microinstruction
000	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
001	Z_{out}, PC_{in}, Y_{in}, WMFC
002	MDR_{out}, IR_{in}
003	μBranch {μPC ← 101 (from Instruction decoder); $\mu PC_{5,4} \leftarrow [IR_{10,9}]$; $\mu PC_3 \leftarrow [\overline{IR_{10}}] \cdot [\overline{IR_9}] \cdot [IR_8]$}
121	$Rsrc_{out}$, MAR_{in}, Read, Select4, Add, Z_{in}
122	Z_{out}, $Rsrc_{in}$
123	μBranch {μPC ← 170; $\mu PC_0 \leftarrow [\overline{IR_8}]$}, WMFC
170	MDR_{out}, MAR_{in}, Read, WMFC
171	MDR_{out}, Y_{in}
172	$Rdst_{out}$, SelectY, Add, Z_{in}
173	Z_{out}, $Rdst_{in}$, End

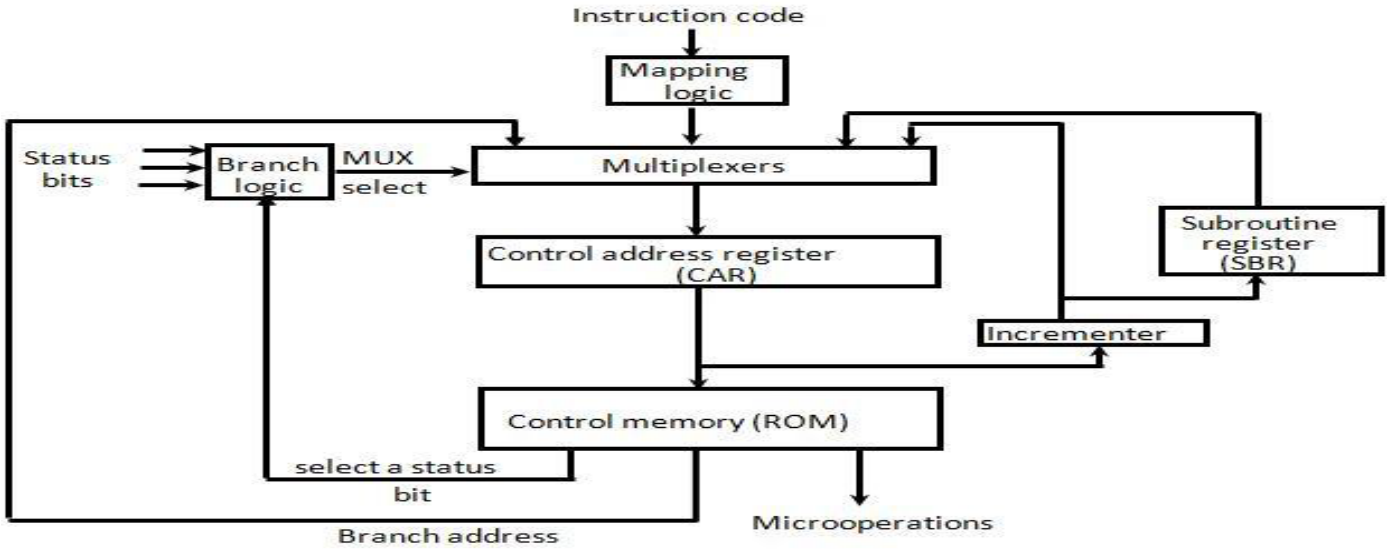
Figure 7.21 Microinstruction for Add (Rsrc)+,Rdst.

52 **Define Microinstruction ? How is it different from microprogram sequencer ? Explain with the help of example.**

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.

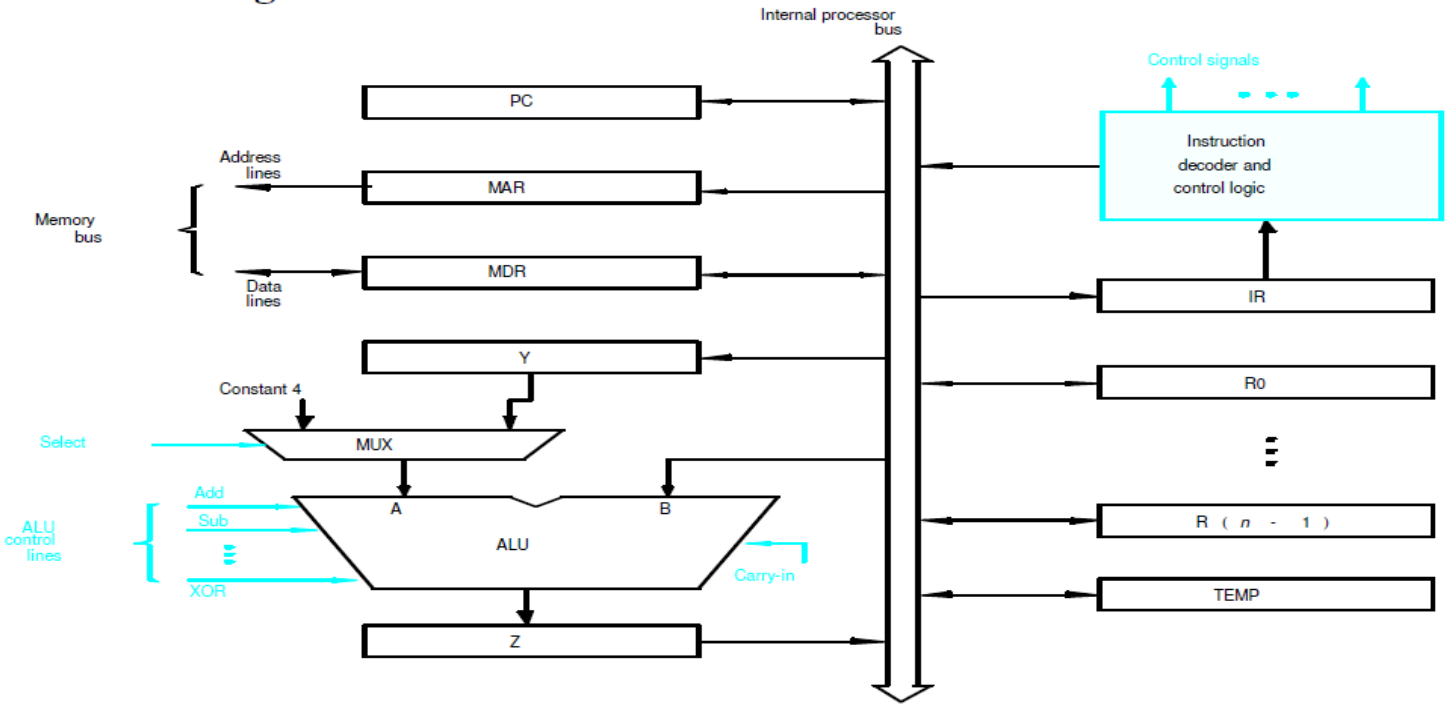
Micro program sequencer

	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. **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 the return 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. 
53	**Describe how a processor execute instructions? Define the internal functional units of a processor and how they are interconnected** **Executing an Instruction •** Fetch the contents of the memory location pointed to by the PC. The contents of this location are loaded into the IR (fetch phase). IR ← [[PC]] • Assuming that the memory is byte addressable, increment the contents of the PC by 4 (fetch phase). PC ← [PC] + 4

• Carry out the actions specified by the instruction in the IR (execution phase).

Processor Organization



ALU and all the registers are interconnected via a single common bus.

• The data and address lines of the external memory bus connected to the internal processor bus via the memory data register, MDR, and the memory address register,MAR respectively.

• Register MDR has two inputs and two outputs.

• Data may be loaded into MDR either from the memory bus or from the internal processor bus.

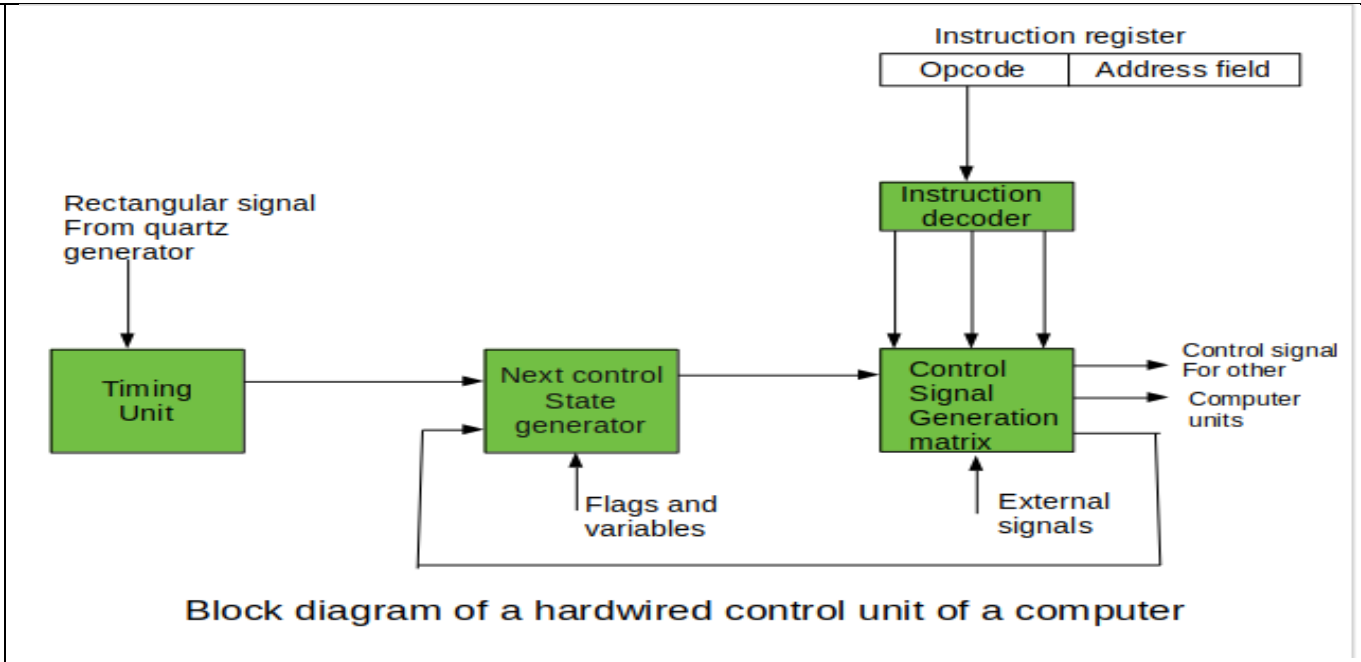
• The data stored in MDR may be placed on either bus.

• The input of MAR is connected to the internal bus, and its output is connected to the external bus.

• The control lines of the memory bus are connected to the instruction decoder and control logic.

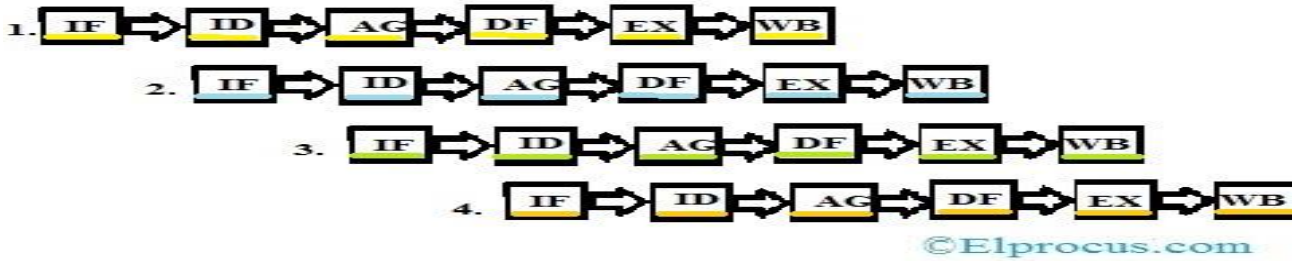
	• This unit is responsible for issuing the signals that control the operation of all the units inside the processor and for increasing with the memory bus. • The MUX selects either the output of register Y or a constant value 4 to be provided as input A of the ALU. • The constant 4 is used to increment the contents of the program counter.
54	**A system uses a control memory of 1024 words of 32 bits each..The micro-instruction has three fields:select, address and microoperation fields. The microoperation's field has 16 bits. (a) How many bits are there in the branch address field and select field?** a. How many bits are there in the branch address field and the select field? b. If there are 16 status bits in the system, how many bits of the branch logic are used to select a status bit? c. How many bits are left to select an input for the multiplexers? a. The branch address field = 10 bits and the select field = 32-16-10 = 6 bits b. 4 bits c. 2 bits
55	**Using the mapping procedure, determine the first micro-instruction address for the following operation code:** **a)0010 b)1011 c)1111** **Solution:** (a) 0001000= 8 (b) 0101100= 44 (c) 0111100= 60
56	**Describe the difference between a microprocessor and a microprogram? Is it possible to design a microprocessor without a microprogram? Explain.** A microprocessor is a small size CPU (computer on a chip) Microprogram is a program for a sequence of microoperations. The control unit of a microprocessor can be hardwired or microprogrammed, depending on the specific design. A microprogrammed computer does not have to be a microprocessor.

57	**Define interrupt? Explain various types of interrupt.** A **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. There are mainly three types of *interrupts*: 1. **External interrupts:** It arises due to external call from I/O devices. For e.g. I/O devices requesting transfer of data, power failure, etc. 2. **Internal interrupts**: It arises due to illegal and erroneous use of an instruction or data. For e.g. stack overflow, division by zero, invalid opcode, etc. These are also called *traps*. 3. **Software interrupts**: It is initiated by executing an instruction. It can be used by the programmer to initiate an interrupt at the desired point in the program. External and internal interrupts are initiated from signals that occur in the hardware of the CPU whereas Software interrupts occur from the instructions.
58	**Explain the hardwired control unit organization explaining each component clearly** **Hardwired Control Unit –** In the Hardwired control unit, the control signals that are important for instruction execution control are generated by specially designed hardware logical circuits, in which we can not modify the signal generation method without physical change of the circuit structure. The operation code of an instruction contains the basic data for control signal generation. In the instruction decoder, the operation code is decoded. The instruction decoder constitutes a set of many decoders that decode different fields of the instruction opcode. As a result, few output lines going out from the instruction decoder obtains active signal values. These output lines are connected to the inputs of the matrix that generates control signals for executive units of the computer. This matrix implements logical combinations of the decoded signals from the instruction opcode with the outputs from the matrix that generates signals representing consecutive control unit states and with signals coming from the outside of the processor, e.g. interrupt signals. The matrices are built in a similar way as a programmable logic arrays.



Block diagram of a hardwired control unit of a computer

59 **Explain advantages of pipelining with a suitable example.**



Advantages of Pipelining

- Instruction throughput increases.
- Increase in the number of pipeline stages increases the number of instructions executed simultaneously.

<table>
<tr><td></td><td>

Faster ALU can be designed when pipelining is used.
Pipelined CPU's works at higher clock frequencies than the RAM.
Pipelining increases the overall performance of the CPU.

Disadvantages of Pipelining

Designing of the pipelined processor is complex.
Instruction latency increases in pipelined processors.
The throughput of a pipelined processor is difficult to predict.
The longer the pipeline, worse the problem of hazard for branch instructions.

</td></tr>
<tr><td>60</td><td>
Define program control. Explain its types.
A program control instruction changes address value in the PC and hence the normal flow of execution. • Change in PC causes a break in the execution of instructions. • It is an important feature of the computers since it provides the control over the flow of the program and provides the capability to branch to different program segments.

Typical Program Control Instructions
<table>
<tr><th>Name</th><th>Mnemonic</th></tr>
<tr><td>Branch</td><td>BR</td></tr>
<tr><td>Jump</td><td>JMP</td></tr>
<tr><td>Skip next instruction</td><td>SKP</td></tr>
<tr><td>Call procedure</td><td>CALL</td></tr>
<tr><td>Return from procedure</td><td>RET</td></tr>
<tr><td>Compare (by subtraction)</td><td>CMP</td></tr>
<tr><td>Test (by ANDing)</td><td>TEST</td></tr>
</table>
Branch (BR) and Jump (JMP) instructions are used sometimes interchangeably but, they are different.
• Branch and Jump instructions usually differ in addressing modes.
• Usually Jump is used to refer to unconditional version of branch.
• Skip (SKP) instructions is used to skip one(next) instruction. It can be conditional or unconditional. It does not need an address field.
</td></tr>
</table>

	• In case of conditional skip instruction, the combination of conditional skip and a unconditional branch can be used an alternative of conditional branch. But, storing two instructions will take extra space. • In skip instruction we increment the PC in execution stage, effectively incrementing it by 2. **Compare (CMP) instruction** performs a comparison via a subtraction, with difference not retained. • The comparison causes one of the three following operations a. A conditional Branch (3ree addresses [2 registers and 1 memory]) b. Change in the contents of a register (3ree addresses) c. Sets or resets stored status bits (2 addresses), this type of instruction is usually followed by a branch instruction to conditionally check the status bit and perform a branch. • Similarly test (TEST) instructions performs the AND of two operands without retaining the result. • It also causes one the above three functions. **A conditional branch instruction** is a branch instruction that may or may not cause a transfer of control depending on the value of stored bits in the PSR (processor status register). • Each conditional branch instruction tests a different combination of Status bits for a condition. • If the condition is true, control is transferred to the effective address (PC←Add). If the condition is false, the program continues with the next instruction (PC←PC+1). **Procedure Call and Return Instructions** • A procedure is a self contained sequence of instructions that performs a given task. • It is also called a subroutine. • When a procedure is called, the starting address of the procedure is stored in the PC and the instruction following the current instruction is temporarily stored elsewhere. When the procedure (block of code) is executed, the return is made to the main program by loading the PC with its old value.

	• Instruction following the procedure call is called continuation point and the corresponding address is called the return address. • It is actually a low level form of functions in C++. (e.g., square(22);) • Procedure can also be called within another procedure. • The final instruction of every procedure must be return to the calling program. The return address can be stored in memory, register or stack. • Stack is preferred because of its ease of access when we need to call a procedure inside another procedure. In that case that the return address at the TOS (top of stack) is always to the program which called the current procedure. **Program Interrupts** An interrupt transfers control from a program that is currently running to another program as a result of externally or internally generated request. Types of Interrupts An interrupt has three types 1. External Interrupts 2. Internal Interrupts 3. Software Interrupts • External and Internal interrupts are both called hardware interrupts. • External interrupts come from input or output devices, from timing devices, from a circuit monitoring the power supply. Or from any other external source. • Conditions that cause external interrupts are an input or output device requesting a transfer of data, the external device completing a transfer of data, the time-out of an event, or an impending power failure.

Sol :- 61)Short Notes on:-

1)Auxiliary memory (also referred to as *secondary storage*) is the non-volatile memory lowest-cost, highest-capacity, and slowest-access storage in a computer system. It is where programs and data kept for long-term storage or when not in immediate use.

Such memories tend to occur in two types-sequential access (data must access in a linear sequence) and direct access (data may access in any sequence). The most common sequential storage device is the hard disk drives, whereas direct-access devices include rotating drums, disks, CD-ROMs, and DVD-ROMs.It used as permanent storage of data in mainframes and supercomputers.

Auxiliary memory may also refer to as auxiliary storage, secondary storage, secondary memory, external storage or external memory. Auxiliary memory is not directly accessible by the CPU; instead, it stores noncritical system data like large data files, documents, programs and other back up information that supplied to primary memory from auxiliary memory over a high-bandwidth channel, which will use whenever necessary. Auxiliary memory holds data for future use, and that retains information even the power fails.

What is Memory Hierarchy?

The memory in a computer can be divided into five hierarchies based on the speed as well as use. The processor can move from one level to another based on its requirements. The five hierarchies in the memory are registers, cache, main memory, magnetic discs, and magnetic tapes. The first three hierarchies are volatile memories which mean when there is no power, and then automatically they lose their stored data. Whereas the last two hierarchies are not volatile which means they store the data permanently.

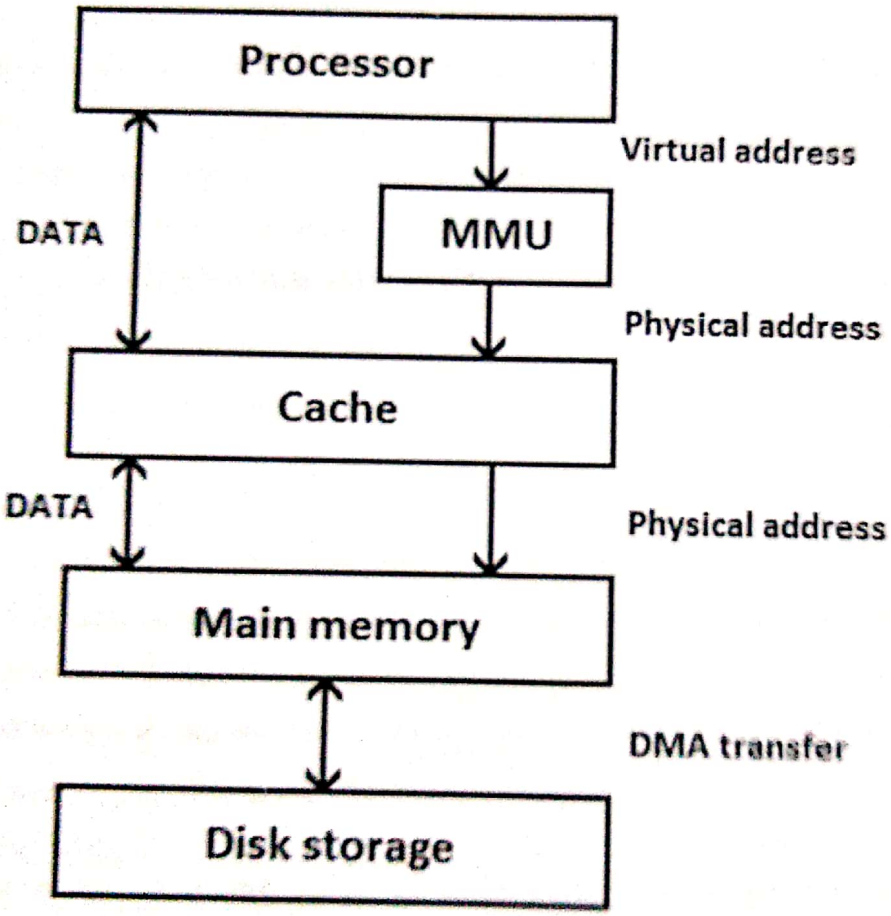
A memory element is the set of storage devices which stores the binary data in the type of bits. In general, the storage of memory can be classified into two categories such as volatile as well as non-volatile.

2)Memory Hierarchy in Computer Architecture

The **memory hierarchy design** in a computer system mainly includes different storage devices. Most of the computers were inbuilt with extra storage to run more powerfully beyond the main memory

capacity. The following **memory hierarchy diagramis** a hierarchical pyramid for computer memory. The designing of the memory hierarchy is divided into two types such as primary (Internal) memory and secondary (External) memory.

3)**Virtual memory** is a valuable concept in **computer architecture** that allows you to run large, sophisticated programs on a **computer** even if it has a relatively small amount of RAM. A **computer** with **virtual memory** artfully juggles the conflicting demands of multiple programs within a fixed amount of physical **memory**.



4) **Cache Memory** is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

Sol:- 62)

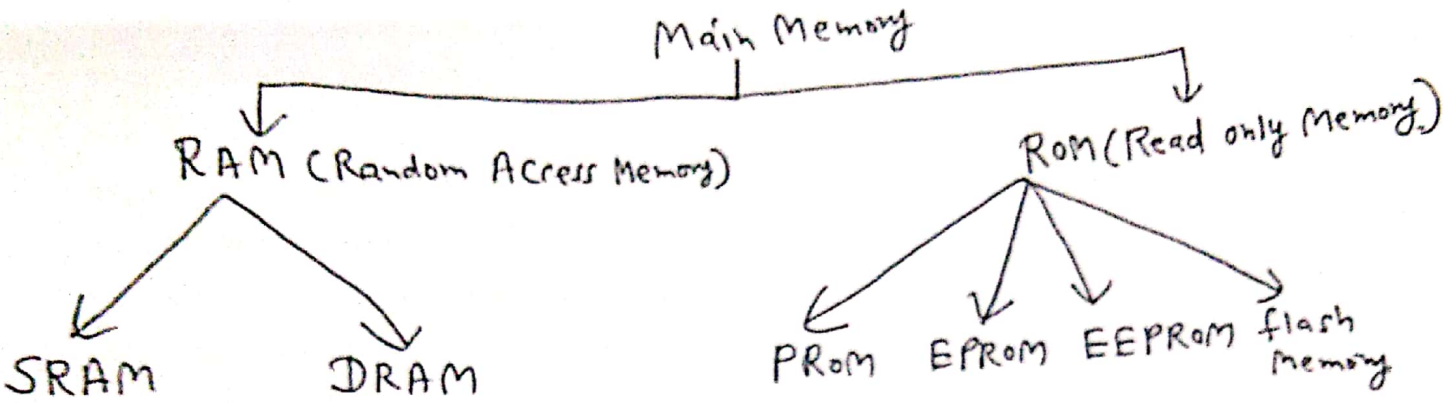
Main Memory

(1) It is the central storage unit in a computer system.

(2) It is used to store program and data during the computer operation.

(3) Based on semiconductor ~~IC (Integrated circuits)~~ Integrated circuits.

(4) Types of Main Memory:



RAM (Random Access Memory) : (i) It is a volatile memory.

(ii) Read/write Access is random.

(iii) Both read and write operation can be performed.

(iv) RAM has two types:-

(a) **SRAM (Static RAM)**: It makes use of flip-flops to store binary information. It is called static RAM because data once stored will remain same (does not change) until power is on. The memory density (memory cell/Area) is relatively less. So less access time is taken in read/write operation. So it is faster, due to this, it is used in Cache memory.

(b) **DRAM (Dynamic RAM)**: It makes use of capacitors for storage. The presence of electric charge on capacitor indicates value (1). The stored charge on capacitors decrease with time. So value '1' may change to '0' if electrical charge finishes. Due to this, it is called dynamic RAM. The memory density (memory cell per unit Area) is relatively is high. Due to this, it has larger access time but more

Scanned by CamScanner

Sol:-63) Virtual memory is a valuable concept in **computer architecture** that allows you to run large, sophisticated programs on a **computer** even if it has a relatively small amount of **RAM**. A **computer** with **virtual memory** artfully juggles the conflicting demands of multiple programs within a fixed amount of physical **memory**.

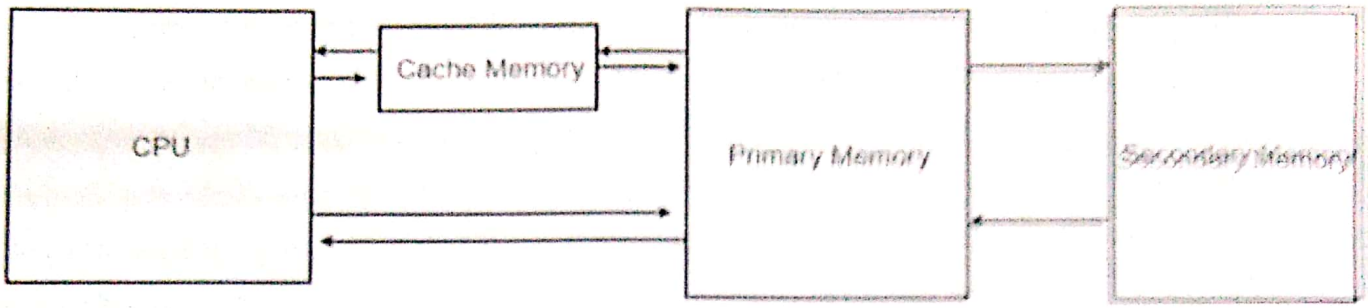
Basically, virtual memory provides an illusion to the users that the PC has enough primary memory left to run the programs. Though the size of programs i.e. to be executed may sometimes very bigger than the size of primary memory left, the user never feels that he needs a bigger primary storage to run that program. When the RAM is full, but program execution needs more space in RAM, then the operating system occupies a portion of the hard disk and uses it as a RAM. In that part of the secondary storage, the part of the program which not currently being executed is stored and all the parts of the program that are eventually executed are first brought into the main memory.

The concept of virtual memory in computer organisation is allocating memory from the hard disk and making that part of the hard disk as a temporary RAM. In the earlier days, when the concept of virtual memory was not introduced, there was a big troubleshooting that when RAM is already full but program execution needs more space in RAM. The computers became unresponsive in such type of situation since processor forced the program to be in RAM but RAM can't hold them because it is already running out of space. To deal with this type problem the concept of virtual memory was introduced.

So, more precisely, we can define virtual memory as the technique which automatically moves programs and data blocks into the main memory when they are required for execution. The processor generates a virtual or logical address for accessing the data from secondary memory and then it is converted into the physical address with some paging algorithm. Set of all such virtual address is called address space and the set of all physical address is called memory space.

If a virtual address pointing to a specific part of a program, which is currently in the main memory, then it is accessed immediately. But if the address pointing is not in the main memory i.e. when the referenced address is in the secondary memory, then the data from this address must be brought into the main memory before using that. In this situation processor actually performs a swapping operation between primary and secondary memory. The program or the part of the program which has the highest

Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.



priority of execution are moved on to the main memory and lower priority programs are moved to the secondary storage.

The conversion of the virtual or logical address into the physical address is done by a special hardware unit known as MMU (Memory Management Unit). When the required data are found in the main memory, then they are fetched to the cache memory for further processing. If required data are not found in the main memory, the MMU with the help of operating system brought the data from secondary storage to main memory. This transfer of data from secondary memory to primary memory is performed using DMA (Direct Memory Access) scheme, which is a feature of a computer system that allows accessing the main memory by certain hardware system. This whole process is independent of the central processing unit.

Now, let see the whole concept of virtual memory with a small example. Suppose the capacity of the main memory of a computer system is 32K while the secondary memory can store 1024K. That means the secondary memory can store data equal to the capacity of 32 main memory. If we denote address space by N and memory space by M, then we can have N = 1024K and M = 32K.

Here, 15 bits are required to specify each physical address since 32K = 2_{15} and 20 bits are required for each virtual address because, 1024K = 2_{20}. So, here CPU is referring an address of 20-bit long for accessing the data from main memory but in reality, our main memory has the only 15-bit long address for accessing data. So, in this case, to access the data from main memory CPU has to perform a mapping of 20-bit virtual address to a 15-bit physical address.

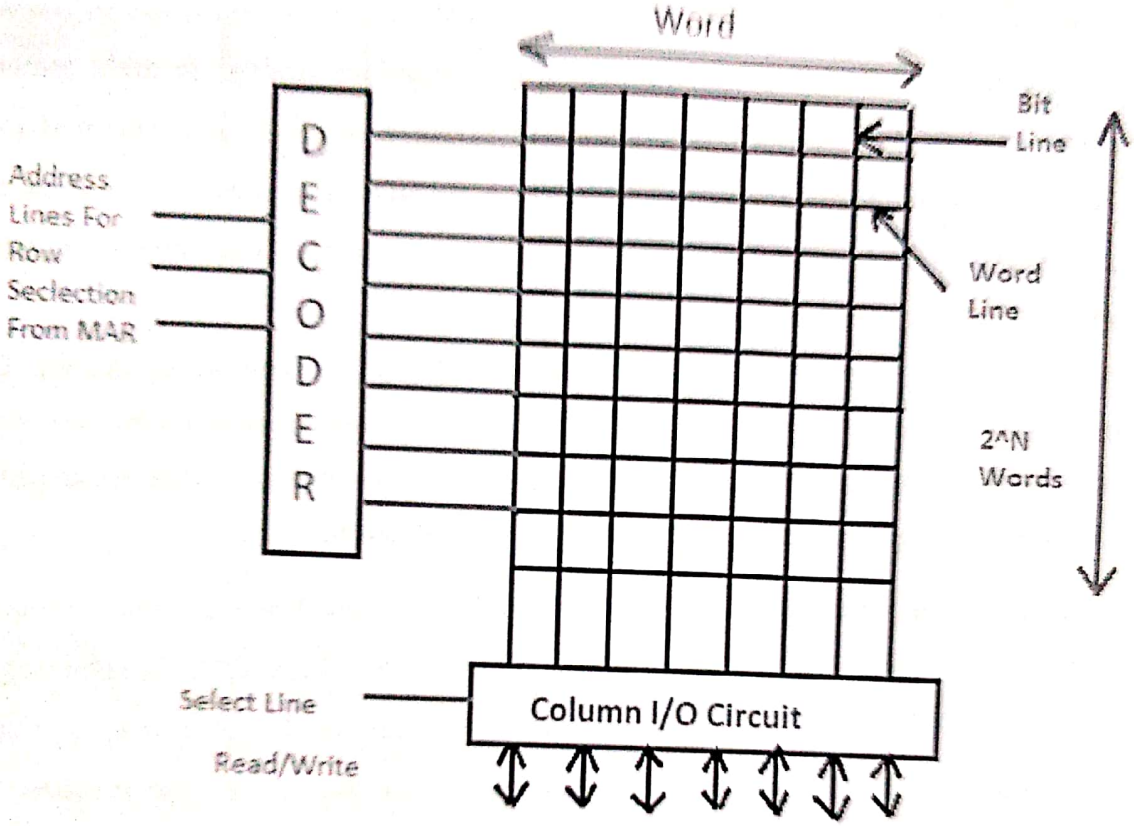
Sol:64) Internal structure of Memory either RAM or ROM is made of memory cells which contains a memory bit. Basically group of 8 bits makes a word. Now the memory is formed in multidimensional array of rows and columns. In which each cell stores a bit and a complete row contains a word. A memory simply can be divided in this below form.

$2^n = N$

where,n is the no. of address lines and N is the total memory in bytes. There will be 2^n words.

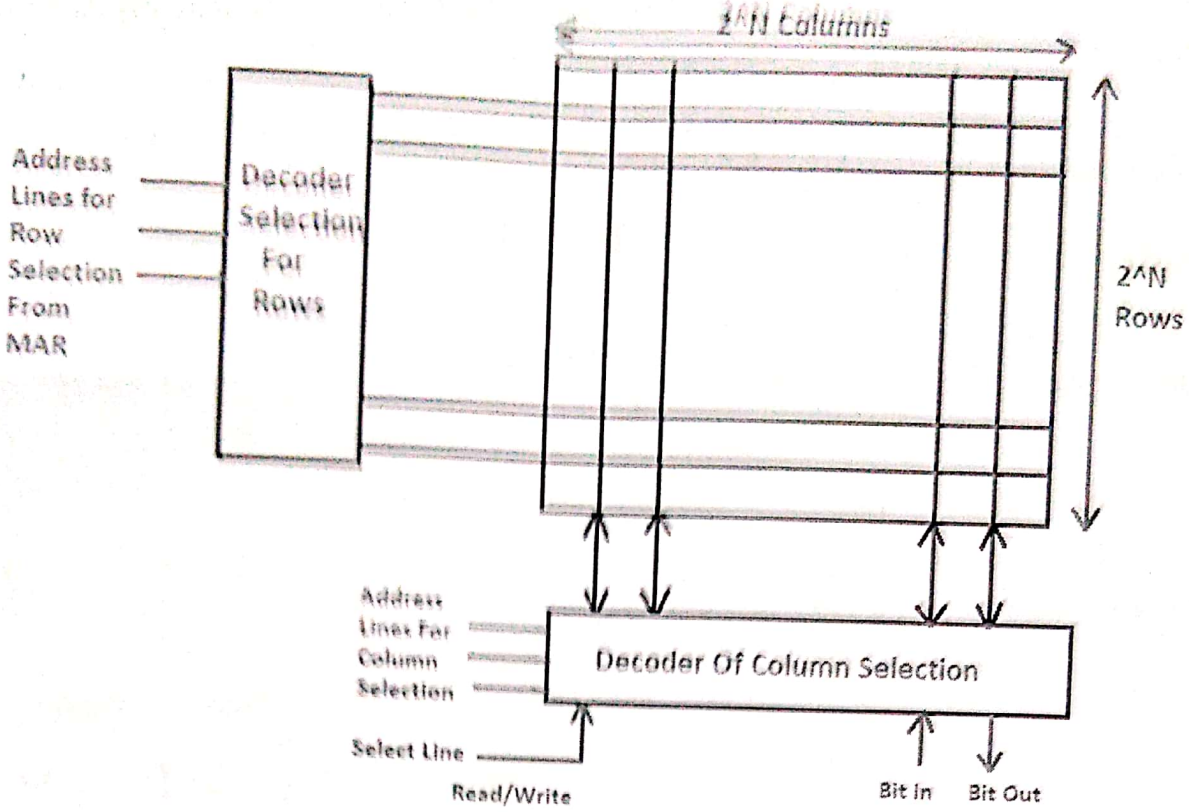
2D Memory Organization

Basically in 2D organization memory is divides in the form of rows and columns. Each row contains a word now in this memory organization there is a decoder. A decoder is a combinational circuit which contains n input lines and 2_n output lines. One of the output line will select the row which address is contained in the MAR. And the word which is represented by the row that will get selected and either read or write through the data lines.



2D Memory Organization

In 2.5D Organization the scenario is the same but we have two different decoders one is column decoder and another is row decoder. Column decoder used to select the column and row decoder is used to select the row. Address from the MAR will go in decoders' input. Decoders will select the respective cell. Through the bit outline, the data from that location will be read or through the bit in line data will be written at that memory location.



2.5D Memory Organization

Read and Write Operations –

1.If the select line is in Read mode then the Word/bit which is represented by the MAR that will be coming out to the data lines and get read.

2.If the select line is in write mode then the data from memory data register (MDR) will go to the respective cell which is addressed by the memory address register (MAR).

3.With the help of the select line the data will get selected where the read and write operations will take place.

Comparison between 2D & 2.5D Organizations –

1.In 2D organization hardware is fixed but in 2.5D hardware changes.

2.2D Organization requires more no. of Gates while 2.5D requires less no. of Gates.

3.2D is more complex in comparison to the 2.5D Organization.

4.Error correction is not possible in the 2D organization but In 2.5D error correction is easy.

5.2D is more difficult to fabricate in comparison to the 2.5D organization.

Sol:-66) 128 K = 217; For a set size of 2, the index: address has 10 bits to accomodate 2048/2 = 1024 words of cache.

(a) 7 bits 10 bits

TAG INDEX

← Block → Words ←

8 bits 2 bits

(b) Tag 1 Data 1 Tag 2 Data 2

← 7 → ← 32 bits → ← 7 → ← 32 bits →

Size of cache memory is 1024 x 2 (7 + 32) = 1024 × 78

Memory Address Map

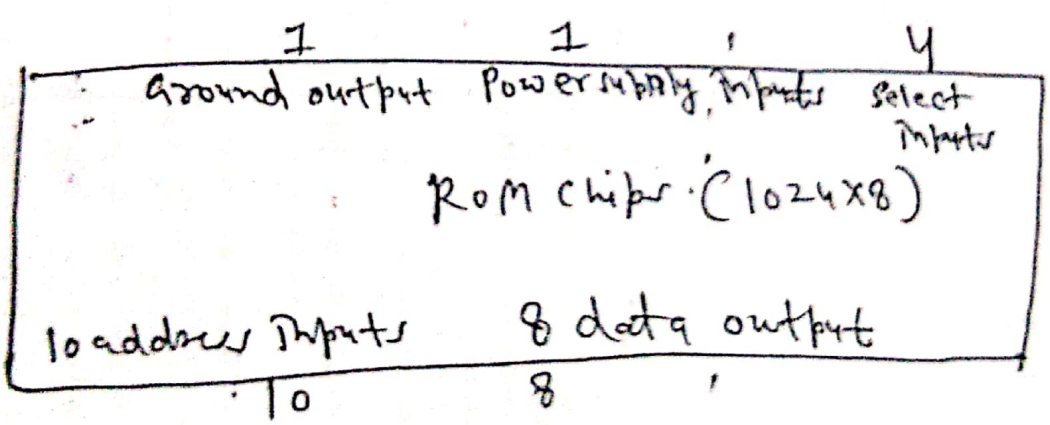
(10)

Components	Memory Addresses	$A_{15}A_{14}A_{13}A_{12}A_{11}A_{10}A_9A_8A_7A_6A_5A_4A_3A_2A_1$
RAM 8	0000-0FFFH	0000 $\xleftarrow[\text{decoder}]{5\times32}$ XXXXXXX
ROM	1000H-1FFFH	0001 $\xleftarrow[\text{decoder}]{3\times8}$ XXXXXXXXX

Sol:- (6s)

A ROM chip of 1024 X 8 bits has four select inputs and operates from a 5V power supply. How many pins are needed for the IC package. Draw a block diagram and label all inputs & output terminals

(Or)



No. of address lines. = 1024 X 8

= 2^{10} X 8

10 address lines, data lines

No. of data lines = 8

Power input = +5V, Select Inputs = 4

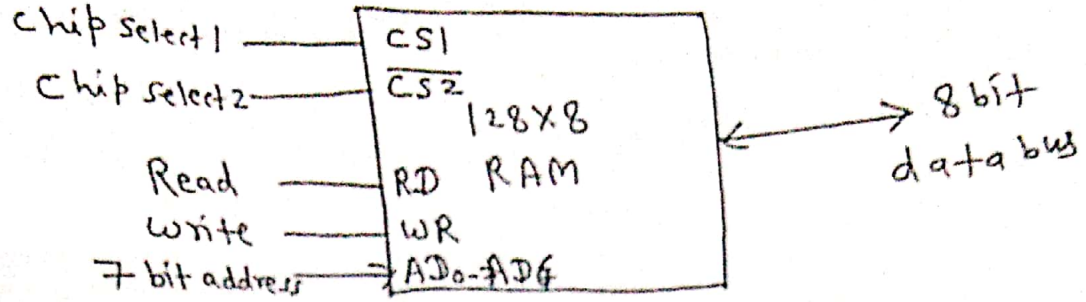
Ground pin = GND,

Total pins = 10+8+1+1+4 = 24 pins

RAM chips & ROM chips Configuration

RAM chip:

Sal:-68



Typical RAM chip of size 128X8

$CS_1, \overline{CS_2}$ → Two chip select Control inputs
RD → Read control input
WR → write control input
AD_0-AD6 → 7 bit address inputs
8 bit data bus

Function table for RAM chips

CS_1	$\overline{CS_2}$	RD	WR	Memory function	State of the data bus
0	0	X	X	Inhibit	High Impedance
0	1	X	X	Inhibit	High Impedance
1	0	0	0	Inhibit	High Impedance
1	0	0	1	write	Input data to RAM
1	0	1	X	Read	output data from RAM
1	1	X	X	Inhibit	High-impedance

Scanned by CamScanner

(4) 3 address line (A8, A9, A10) must be dec... for chip select.

Decoder size = 3 X 8 decoder

(5) Detailed Connection are already shown on Ans (3).

Sol:- 12.2

12.2 A Computer uses RAM chips of 1024X1 Capacity

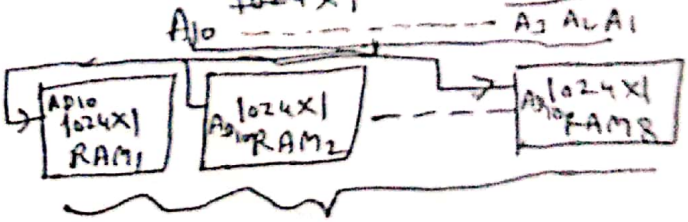
(a) How many chips are needed, and how should their address lines be connected to provide a memory capacity of 1024 byte.

(b) How many chips are needed to provide a memory Capacity of 16 K bytes. How the chips are to be connected to the address bus.

(a) No. of chips needed = $\frac{\text{Total memory}}{\text{one RAM size}} = \frac{1024 \text{ byte}}{1024 \times 1}$

$= \frac{1024 \times 8}{1024 \times 1} = 8$ chips

Memory Connection diagram

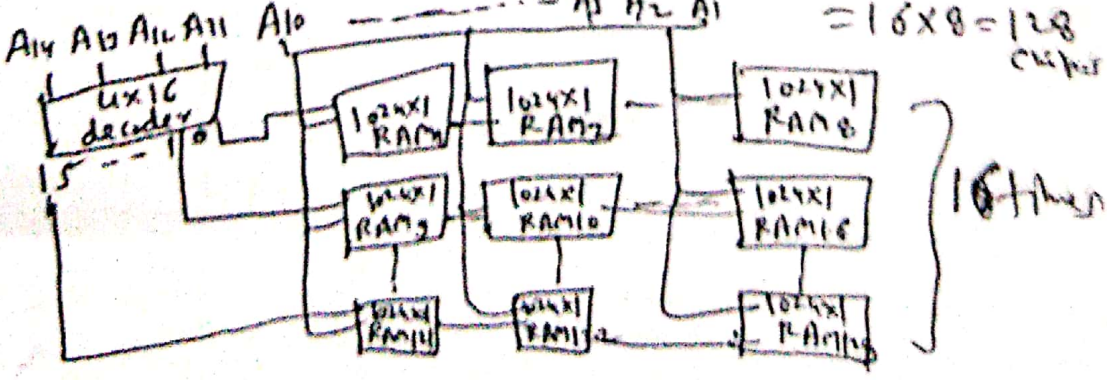


8 chips

Total memory = 1024 X 8

(b) No. of chips needed = $\frac{16 \text{ K byte}}{1024 \times 1} = \frac{16K \times 8}{1024 \times 1} = \frac{16K \times 8}{1K \times 1} = 16 \times 8 = 128$ chips

Memory Connection



Scanned by CamScanner

If the particular address is found in the cache, the block of data is sent to the CPU, and the CPU goes about its operation until it requires something else from memory. When the CPU finds what it needs in the cache, a hit has occurred. When the address requested by the CPU is not in the cache, a miss has occurred and the required address along with its block of data is brought into the cache according to how it is mapped.

Cache processing in some computers is divided into two sections: **main** cache and **eavesdrop** cache. Main cache is initiated by the CPU within. Eavesdrop is done when a write to memory is performed by another requestor (other CPU or IOC). Eavesdrop searches have no impact on CPU performances.

CACHE MAPPING TECHNIQUES.— Cache mapping is the method by which the contents of main memory are brought into the cache and referenced by the CPU. The mapping method used directly affects the performance of the entire computer system.

•Direct mapping —Main memory locations can only be copied into one location in the cache. This is accomplished by dividing main memory into pages that correspond in size with the cache (fig. 5-10).

•Fully associative mapping —Fully associative cache mapping is the most complex, but it is most flexible with regards to where data can reside. A newly read block of main memory can be placed anywhere in a fully associative cache. If the cache is full, a replacement algorithm is used to determine which block in the cache gets replaced by the new data (fig. 5-11).

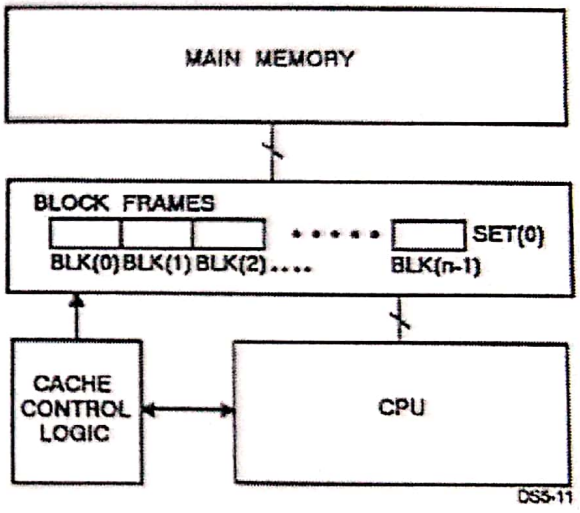


Figure 5-11.—Example of fully associated mapping used in cache memory.

•Set associative mapping —Set associative cache mapping combines the best of direct and associative cache mapping techniques. As with a direct mapped cache, blocks of main memory data will still map into as specific set, but they can now be in any N-cache block frames within each set (fig. 5-12).

CACHE READ.— The two primary methods used to read data from cache and main memory are as follows:

CACHE MAPPING TECHNIQUES

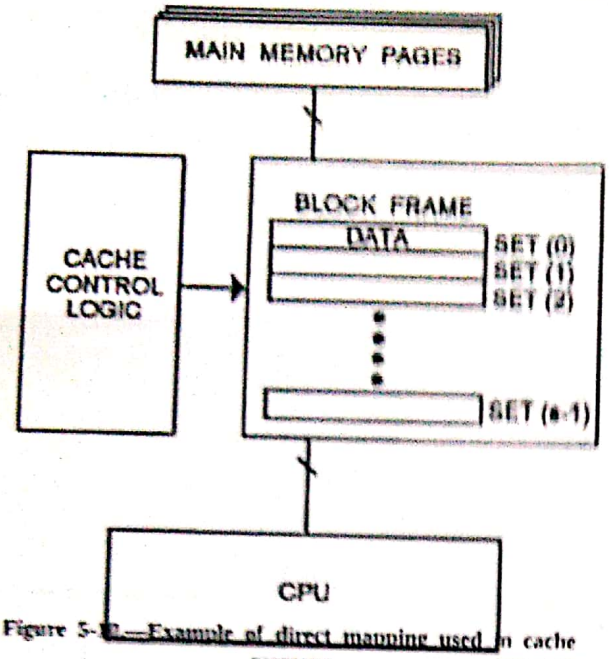
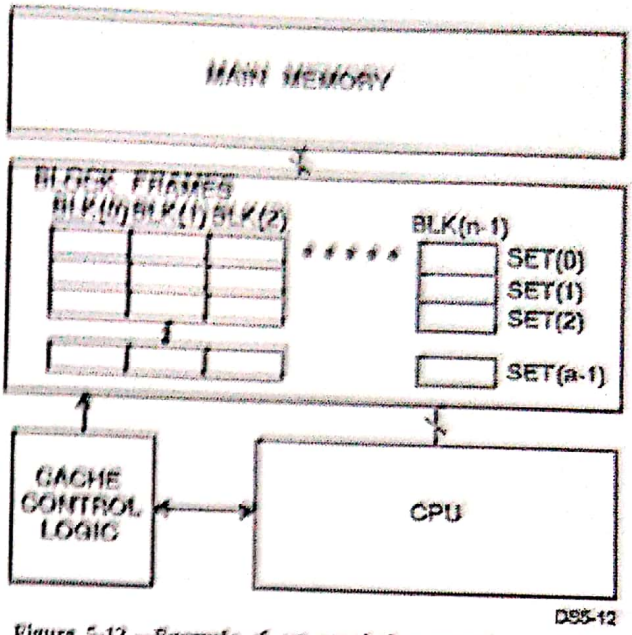


Figure 5-[illegible]—Example of direct mapping used in cache memory. ETFC0053



D95-12

Figure 5-12.—Example of set association mapping used in cache memory.

Sol:70)Direct Mapping

The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line.

In Direct mapping, assigne each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping`s performance is directly proportional to the Hit ratio.

i = j modulo m

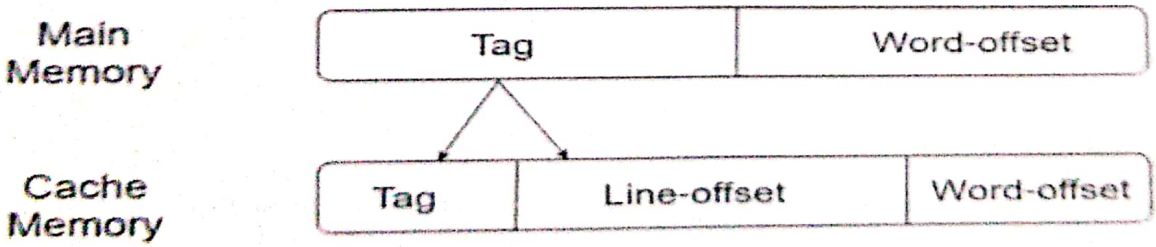
where

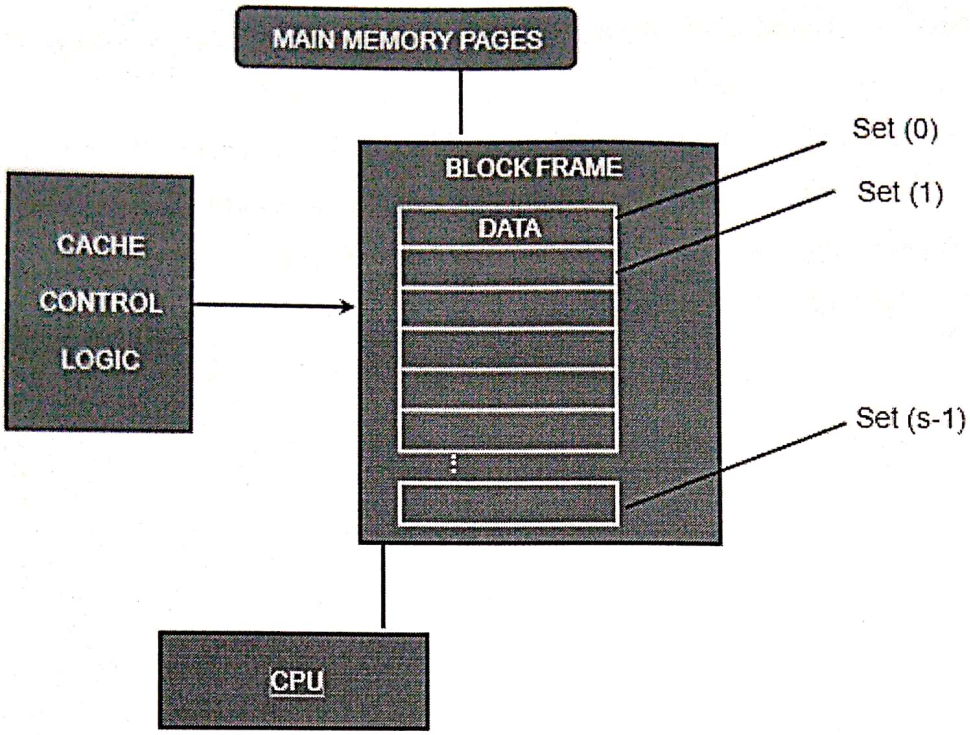
i=cache line number

j= main memory block number

m=number of lines in the cache

For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^s blocks of main memory. The cache logic interprets these s bits as a tag of s-r bits (most significant portion) and a line field of r bits. This latter field identifies one of the $m=2^r$ lines of the cache.





1.**Associative Mapping** –

In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form.

Numerical on RAM & ROM chips

Q① How many 128×8 RAM chips are needed to provide a memory capacity of 2048 bytes

Ans

Size of one RAM chip = 128×8

Total memory capacity = 2048 bytes

= 2048×8

No. of RAM chips = Total memory size / one RAM chip size

$= \frac{2048 \times 8}{128 \times 8} = \frac{2^{11}}{2^7}$

$= 2^{11-7} \Rightarrow 2^4 = 16$

Ans = 16 chips

② How many ROM chips of 1024×8 size are required for Total ROM of 4K bytes

Ans

No. of ROM chips = Total ROM size / one ROM chip size

$= \frac{4K \text{ bytes}}{1024 \times 8}$

$= \frac{4K \times 8}{1024 \times 8}$

$= \frac{4 \times 1024 \times 8}{1024 \times 8}$

Ans = 4 ROM chips

Sol:-72) The pages that are not in main memory are: Page Address address that will cause fault 2 2K 2048 – 3071 3 3K 3072 – 4095 5 5K 5120 – 6143 7 7K 7168 – 8191

sol:-73) Logical address: 7 bits 5 bits 12 bits = 24 bits Segment Page Word Physical address: 12 bits 12 bits Block Word

Part 1: No. of RAM chips = $\frac{\text{Total RAM size}}{\text{one RAM size}}$

$= \frac{2\text{KBytes}}{256 \times 8} =$ **8 RAM chips**

No. of ROM chips = $\frac{\text{Total ROM size}}{\text{one ROM size}}$

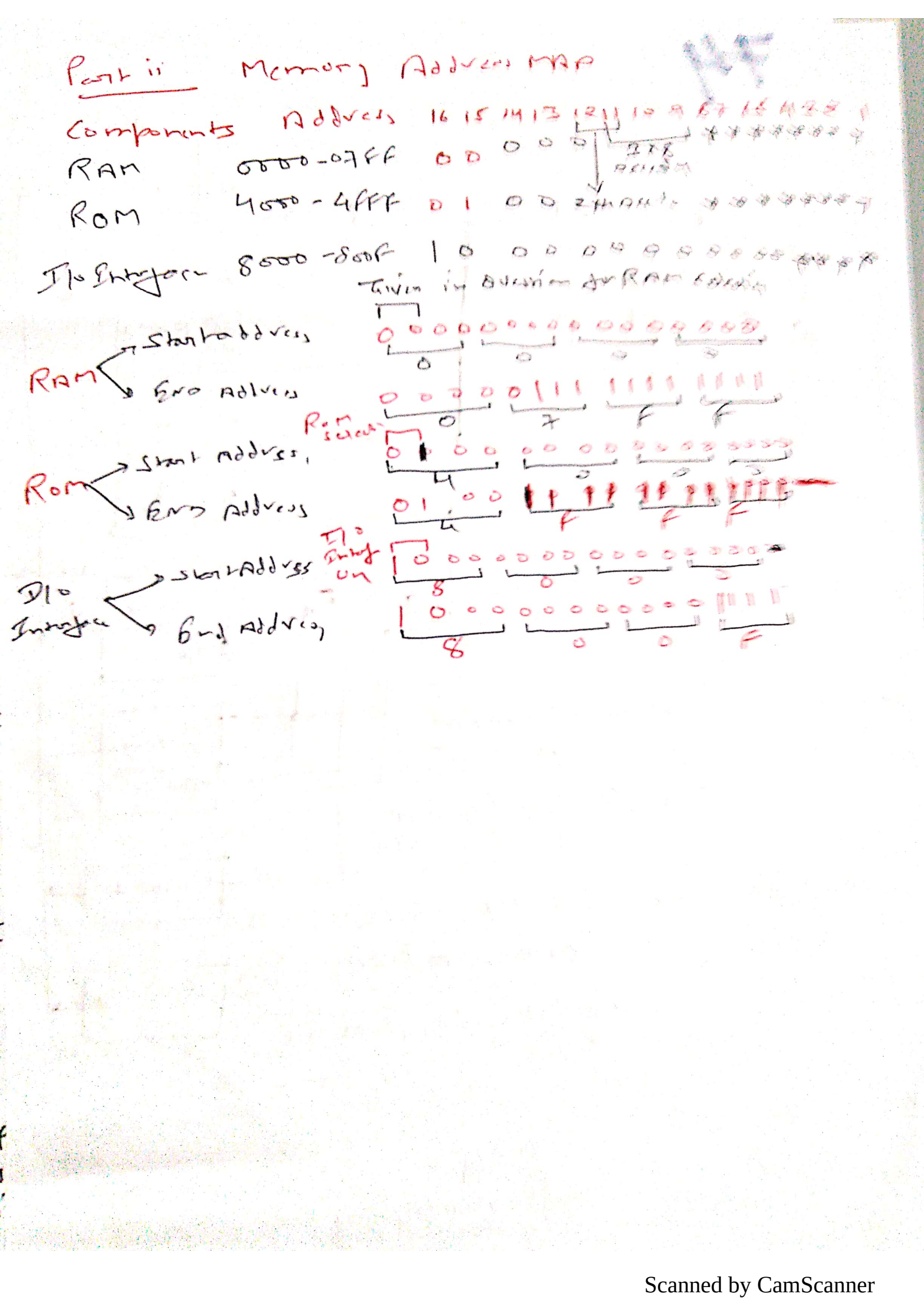
$= \frac{4\text{K Bytes}}{1024 \times 8} =$ **4 ROM chips**

No. of I/O Registers = 4 × 4

Registers = **16 Registers**

Part ii Memory Address MAP

Components	Address	16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
RAM	0000 - 07FF	0 0 0 0 0
ROM	4000 - 4FFF	0 1 0 0
I/O Interface	8000 - 800F	1 0 0 0



Set. - 2s

Memory Hierarchy:

The memory hierarchy system consists of all storage devices employed in a computer system from the slow (but high storage) auxiliary memory to very fast (but low storage) registers.

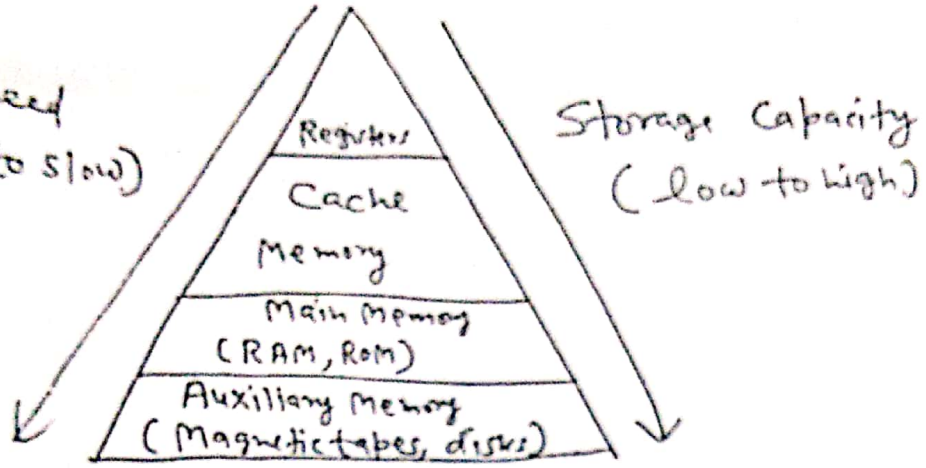


fig: 1 Memory Hierarchy

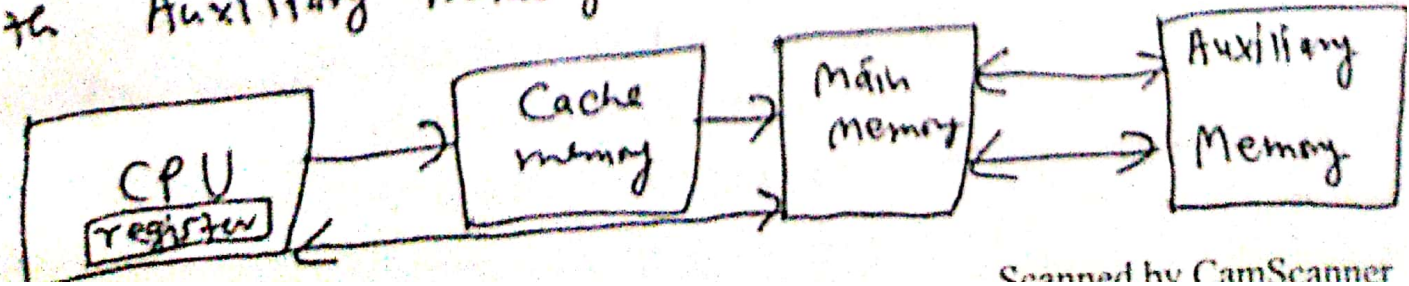
1) **Registers**: are fastest memory with very low storage and most expensive.

2) **Cache Memory**: Slower than registers, but faster than main memory & has more storage than registers used to hold program & data that are heavily used.

3) **Main Memory**: Storage more than Cache Memory but less than Auxiliary memory. Speed less than Cache memory but more than auxiliary memory

4) **Auxiliary Memory**: It has maximum Storage Capacity but slowest among all devices. It is also cheapest among all devices

CPU can directly communicate with main memory and also via cache memory. CPU can not directly communicate with Auxiliary memory.



large capacity. Due to this, it is used in main memory.

(ii) ROM (Read Only Memory): It is also a type of main memory. In this, data/program both are read only. No write operation can be performed in this (But versions of ROM now performs write operation also). ~~Read/write access~~ ROM is also random access. It is also non-volatile (No data lost, if power off) in nature. The types of ROMs are:-

(i) PROM (Programmable ROM): This ROM chip is one time programmable. once programmed, it can not be erased.

(ii) EPROM (Erasable Programmable ROM): This ROM chip is multi time programmable. once programmed, user can erase data using Ultra-violet rays to erase whole chip & then programme again. Any change require, whole chip to be erased.

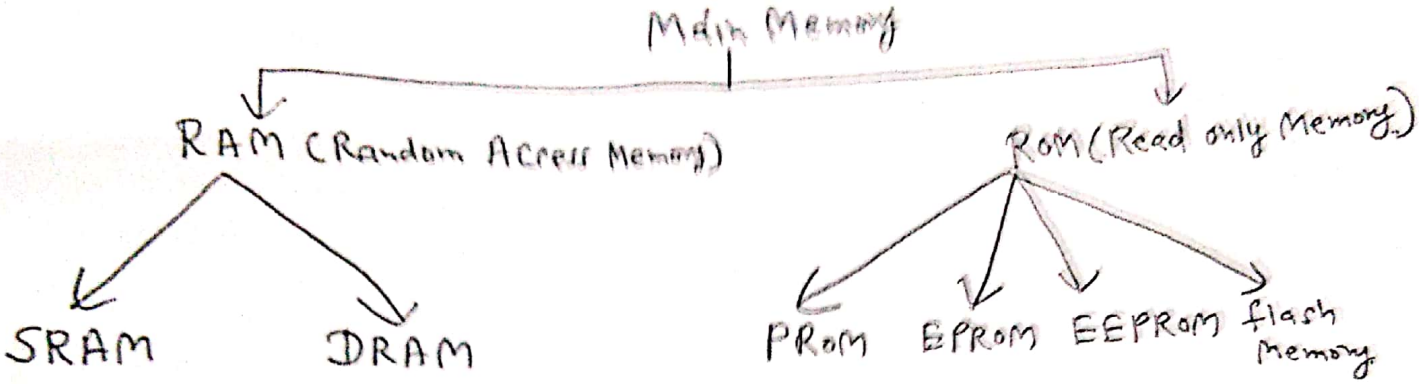
(iii) EEPROM (Electrical Erasable PROM): This ROM chip is also multiple time programmable. In this chip, electrical signals are used to erase data at byte or block level rather than erasing whole chip & program again. faster than EPROM.

(iv) Flash Memory: It is almost similiar to EEPROM. The main difference from EEPROM is that in this, a bit level erasing & rewriting of data is possible which make it very fast. Due to this, it is called flash memory.

Sol. Q 7

Main Memory

(1) It is the central storage unit in a computer system.
(2) It is used to store program and data during the computer operation.
(3) Based on semiconductor integrated circuits.
(4) Types of Main Memory:



RAM (Random Access Memory) (i) It is a volatile memory.
(ii) Read/write Access is random.
(iii) Both read and write operation can be performed.
(iv) RAM has two types:-

(a) **SRAM (Static RAM)**: It makes use of flip-flops to store binary information. It is called static RAM because data once stored will remain same (does not change) until power is on. The memory density (memory cell/Area) is relatively less. So less access time is taken in read/write operation. So it is faster, due to this, it is used in Cache memory.

(b) **DRAM (Dynamic RAM)**: It makes use of capacitors for storage. The presence of electric charge on capacitor indicates value (1). The stored charge on capacitor decreases with time. So value '1' may change to '0' if electrical charge finishes. Due to this, it is called dynamic RAM. The memory density (memory cell per unit Area) is relatively is high. Due to this, it has larger access time but more

Sol :-78) How locality of reference used in cache memory :

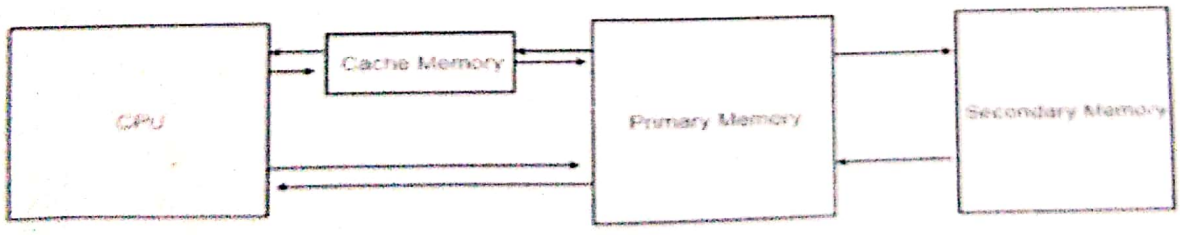
The locality of reference holds good in computer architecture which is utilized to enhance the speed of the cache memory system. If we consider the execution of 5 instructions, with the implemented cache memory and locality of reference, We can bring those **5** instructions simultaneously from primary memory to cache memory only in **90ns**.

As a CPU can hold only a single instruction at a time, the time required to bring these 5 instructions from cache to CPU, one after another is **5*20 = 100ns**. So, the total time required is **90+100 = 190ns**.

This way the cache memory with the locality of reference can enhance the speed of computer system. It is due to the locality of reference, that almost all the instructions fetched from primary memory to cache memory will get executed by the CPU.

Sol:79) Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.

Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory which stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.



The locality of reference is implemented to utilize the full benefit of cache memory in computer organization. It indicates that all the instructions referred by the processor are localized in nature. If the processor searches an instruction 'i', the probability is very high that after execution of instruction 'i', it will search for 'i+1'. That is a processor request instructions from memory which are residing in memory in nearby locations.

Sol:-80) With cache design , you always have to balance hit rate (the likelihood the cache contains the data you want) vs hit time/latency (how long it takes your cache to respond to a request). Complexity is also a factor.

That matters a lot when asking "which is faster". A slow but accurate cache can prove to be more effective than a simpler , faster one , or vice versa. The architecture of the CPU behind it matters a lot. A direct-mapped cache theoretically provides a shorter critical path , meaning it can be run faster than a set associative cache. But in practice , a lot of the tag checking and other work associated with a set-associative cache can be done in parallel , meaning the actual speed difference is likely to be much less significant.

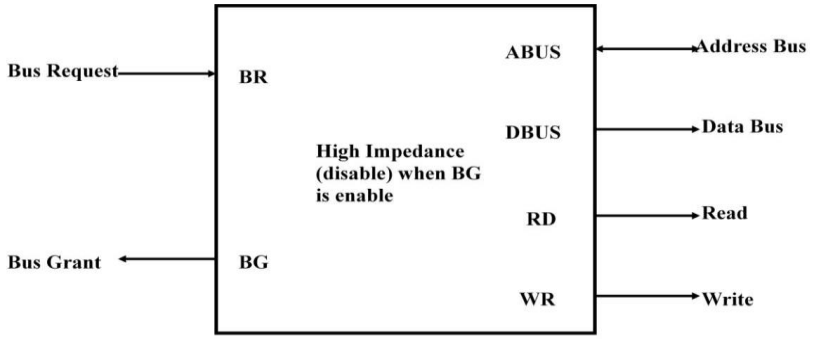
Set associative caches generally have a higher hit rate than direct mapped caches because they suffer from conflict misses a lot less than direct mapped ones.

81 **Explain Direct Memory Access (DMA).**

Direct Memory Access (DMA):

In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA).

During the DMA transfer, the CPU is idle and has no control of the memory buses. A DMA Controller takes over the buses to manage the transfer directly between the I/O device and memory.



CPU bus Signals for DMA Transfer

The transfer can be made in several ways that are:

i. DMA Burst

ii. Cycle Stealing

i) DMA Burst :- In DMA Burst transfer, a block sequence consisting of a number of memory words is transferred in continuous burst while the DMA controller is master of the memory buses.

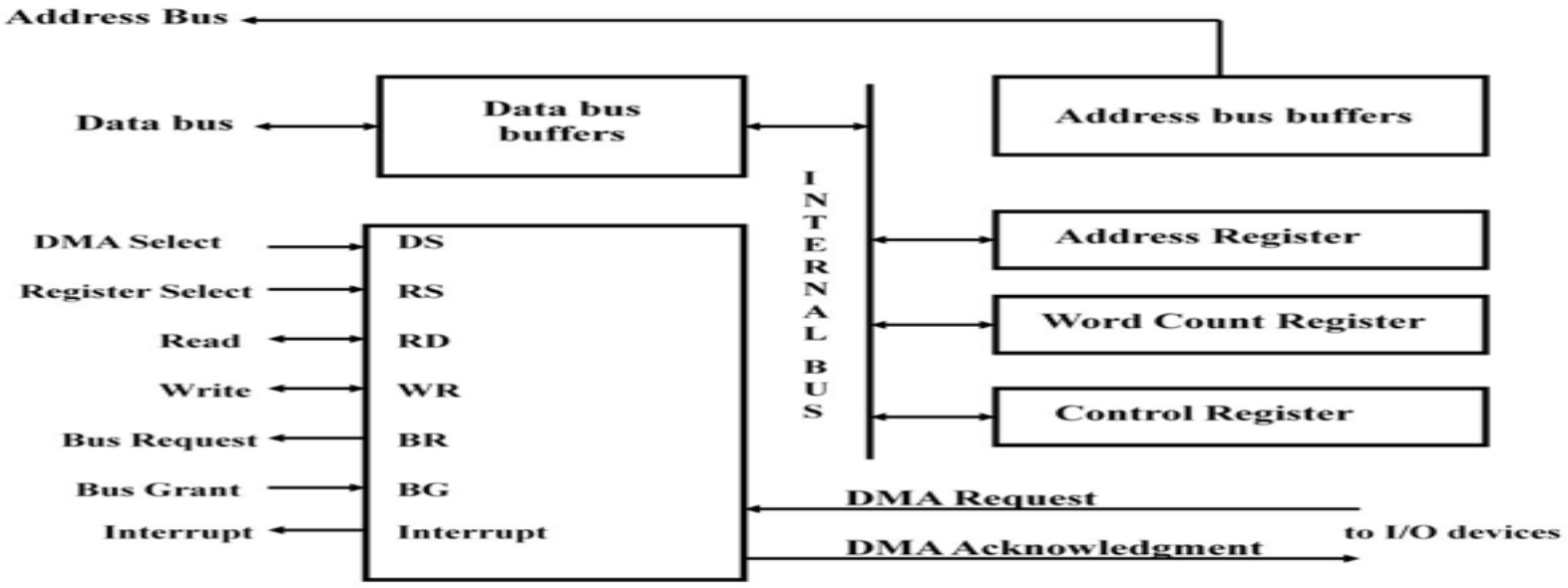
ii) Cycle Stealing :- Cycle stealing allows the DMA controller to transfer one data word at a time, after which it must returns control of the buses to the CPU.

The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. The DMA controller has three registers:

i. Address Register

ii. Word Count Register

iii. Control Register



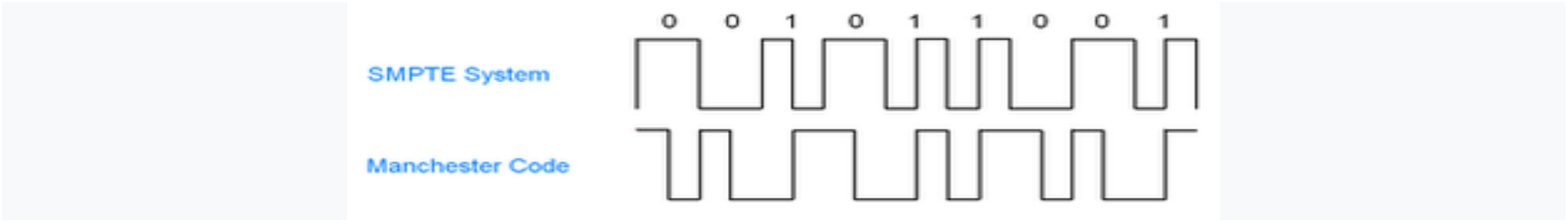
Block Diagram of DMA Controller

82 **Explain Synchronous and Asynchronous communication.**

Synchronous transmission

In synchronous communications, the stream of data to be transferred is encoded as fluctuating voltage levels in one wire (the 'DATA'), and a periodic pulse of voltage on a separate wire (called the "CLOCK" or "STROBE") which tells the receiver "the current DATA bit is 'valid' at this moment in time".

Practically all parallel communications protocols use synchronous transmission. For example, in a computer, address information is transmitted synchronously—the address bits over the address bus, and the read or write 'strobe's of the control bus.



A logical one is indicated when there are two transitions in the same time frame as a zero. In the Manchester coding a transition from low to high indicates a one and a transition from high to low indicates a zero. When there are successive ones or zeros, an opposite transition is required on the edge of the time frame to prepare for the next transition and signal.

Asynchronous transmission

The most common asynchronous signalling, asynchronous start-stop signalling, uses a near-constant 'bit' timing (+/- 5% local oscillator required at both end of the connection). Using this method, the receiver detects the 'first' edge transition... (the START BIT), waits 'half a bit duration' and then reads the value of the signal. A further delay of one 'whole bit duration' is executed before the next data bit is 'read' - repeating for length of the whole serial word (typically 7/8-data bits). Finally, an optional STOP bit is appended to identify the end of the data word.

The word structure used in typical asynchronous serial communications is START-DATA[0:7]-STOP[0:1] (followed by an optional PARITY bit). These formatting variables are specified when configuring the transmit and receive nodes before communications take place. The bit duration is determined from the nominated 'bit rate' in bps... 300, 1200, 9600, 19200, 115200 etc. The use of the word BAUD is not strictly correct in the modern application of serial channels.

Special level & timing conditions are detected to identify an open-circuit condition (BREAK)

CSE 2nd Year KCS302 (Computer Organization) Unit 5 Solution

83	**Define Interrupts with their types and exceptions** Interrupts and Exceptions both alter program flow. The difference being, interrupts are used to handle external events (serial ports, keyboard) and exceptions are used to handle instruction faults, (division by zero, undefined opcode). 1) **Interrupts** are handled by the processor after finishing the current instruction. 1) **Exceptions** on the other hand are divided into three kinds. • Faults o Faults are detected and serviced by the processor before the faulting instructions. • Traps o Traps are serviced after the instruction causing the trap. User defined interrupts go into this category and can be said to be traps. • Aborts o Aborts are used only to signal severe system problems, when operation is no longer possible. 2) Interrupt is an as asynchronous event that is normally(not always) generated by hardware(Ex, I/O) not in sync with processor instruction execution. 2) Exceptions are synchronous events generated when processor detects any predefined condition while executing instructions. Lastly, An interrupt handler may defer an exception handler, but an exception handler never defers an interrupt handler.
84	**Write short note on the following together with their importance:** **(i) Input-output processor** **(ii) Serial Communication.** **(i) Input-Output Processor:** □ It is a processor with direct memory access capability that communicates with IO devices. □ IOP is similar to CPU except that it is designed to handle the details of IO operation.

- Unlike DMA which is initialized by CPU, IOP can fetch and execute its own instructions.

IOP instruction are specially designed to handle IO operation

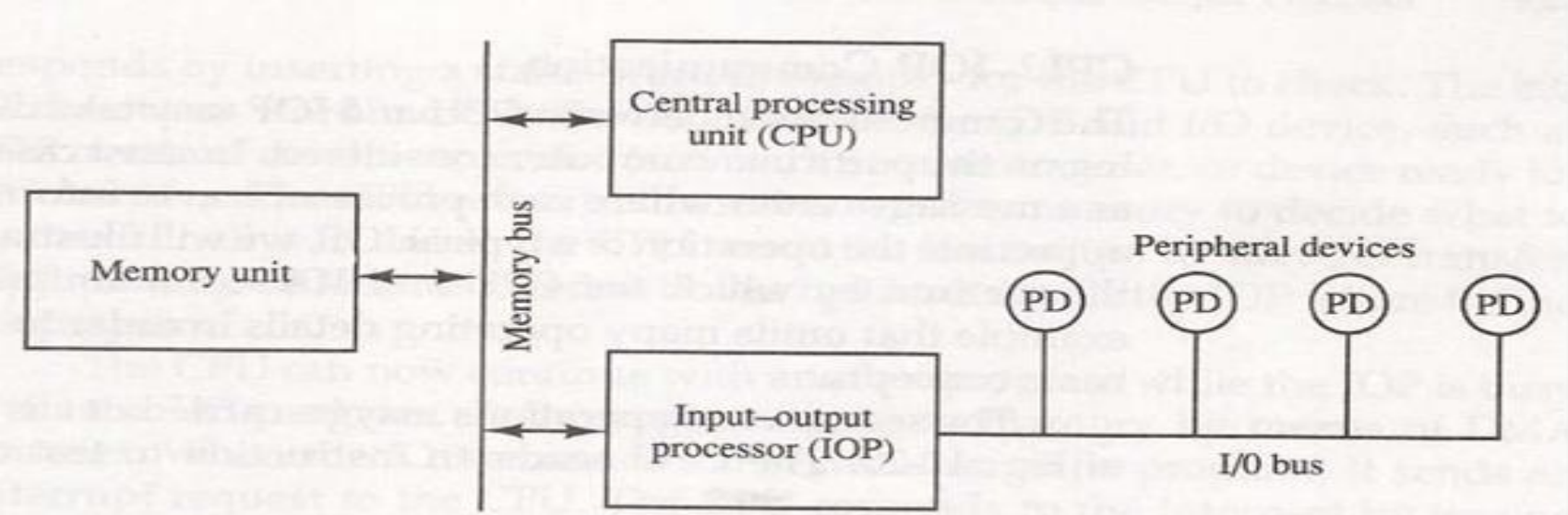


Figure Block diagram of a computer with I/O processor.

- Memory occupies the central position and can communicate with each processor by DMA.
- CPU is responsible for processing data.
- IOP provides the path for transfer of data between various peripheral devices and memory.
- Data formats of peripherals differ from CPU and memory. IOP maintain such problems.
- Data are transfer from IOP to memory by stealing one memory cycle.
- Instructions that are read from memory by IOP are called commands to distinguish them from instructions that are read by the CPU.

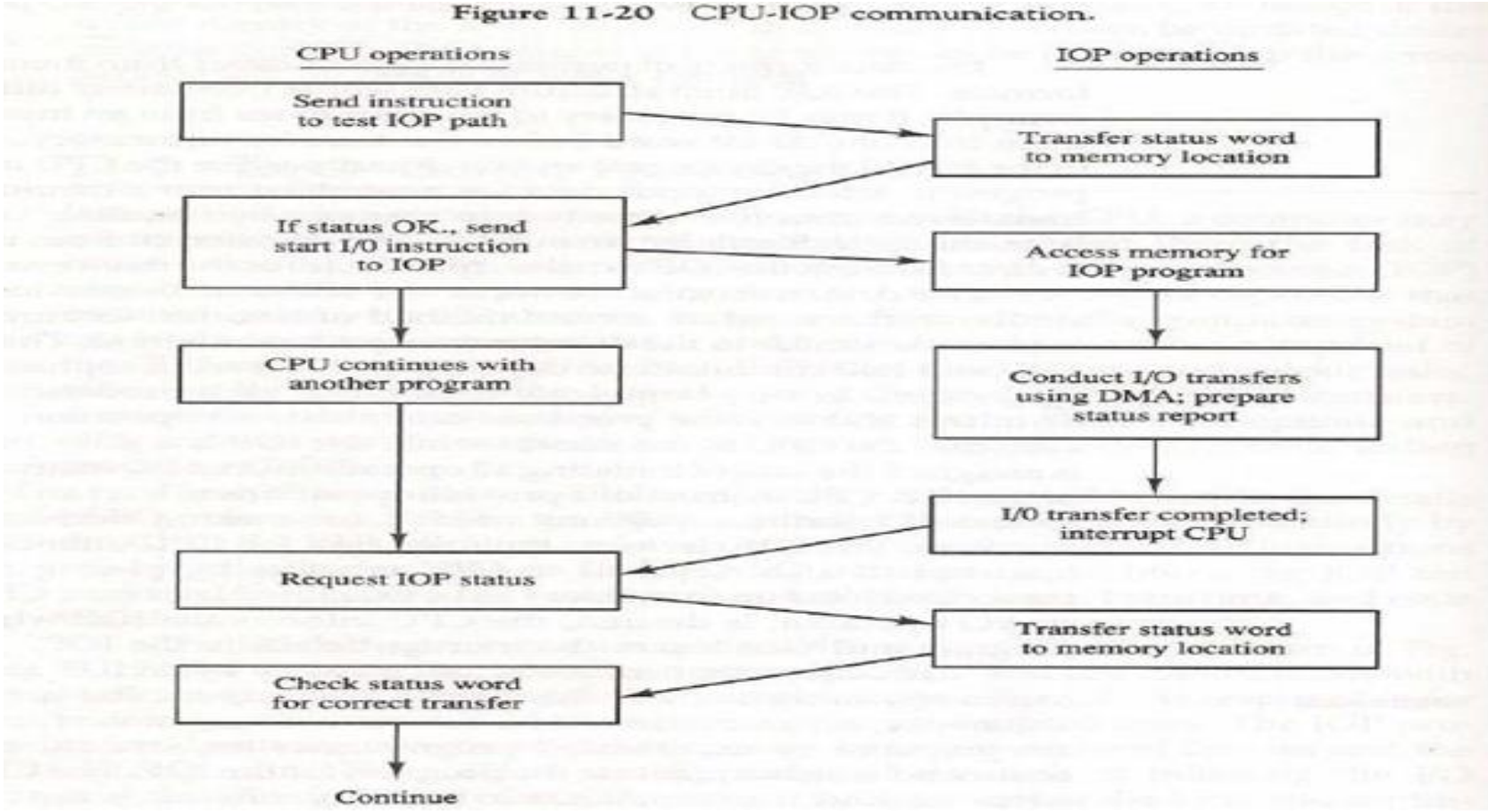


Figure 11-20 CPU-IOP communication.

(ii) **Serial communication** is a communication technique used in telecommunications wherein data transfer occurs by transmitting data one bit at a time in a sequential order over a computer bus or a communication channel. It is the simplest form of communication between a sender and a receiver. Because of the synchronization difficulties involved in parallel communication, along with cable cost, serial communication is considered best for long-distance communication.

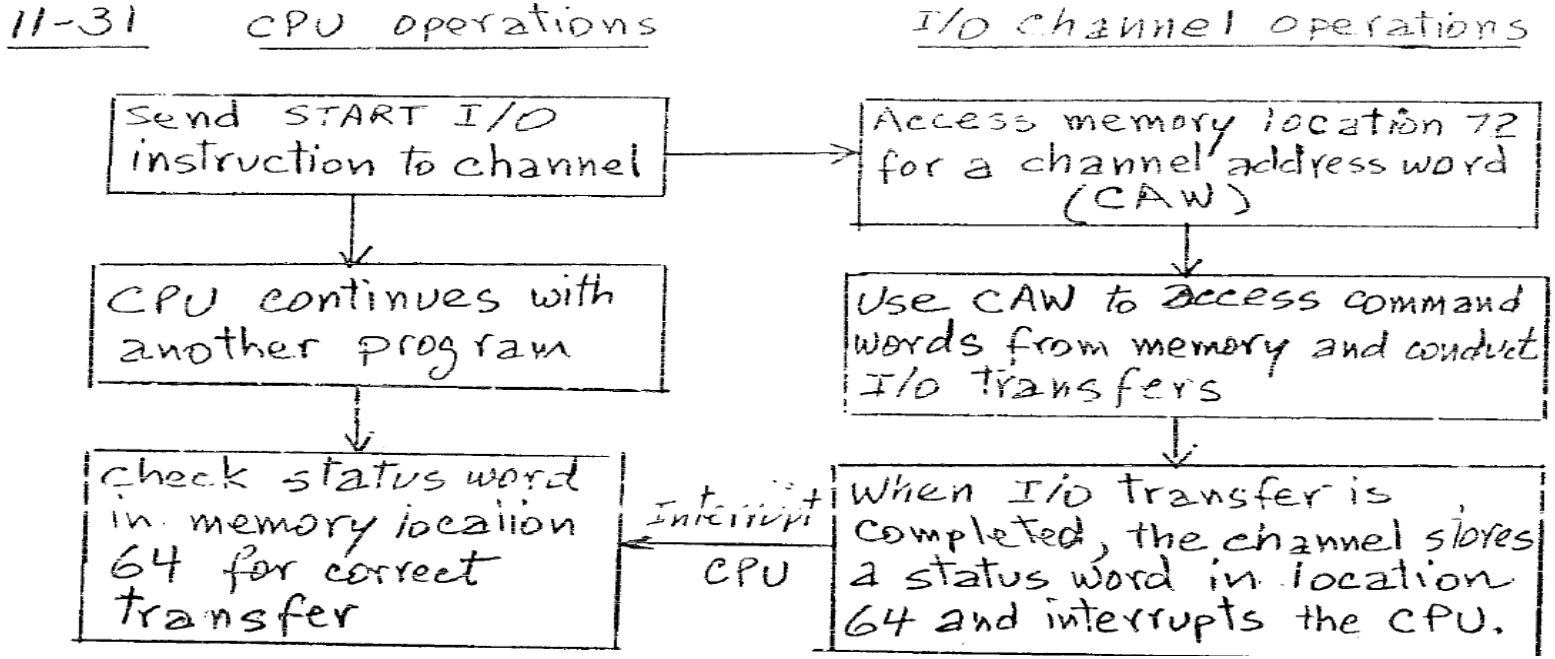
In contrast to parallel communication, which is half duplex, serial communication is full duplex, i.e., transmission and receipt of signals can occur simultaneously. It is the most popular mode of communication protocol for most instrumentation devices. It is also popular in computer devices, peripheral devices and integrated circuits, which are provided with one or more serial ports, resulting in no

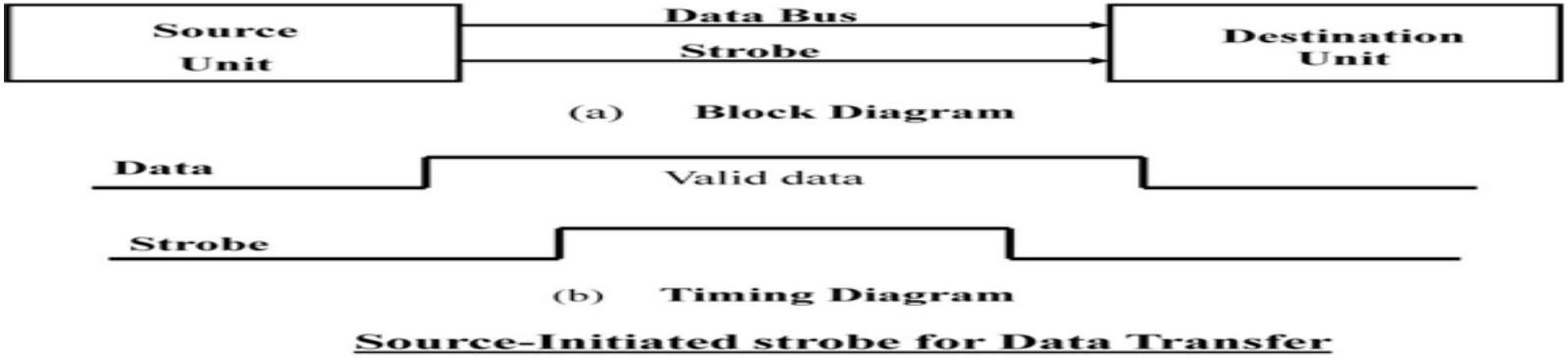
	additional hardware requirements for serial communication. There are several advantages with serial communication. As there are fewer conductors in contrast to parallel communication, the cross-talk issue is significantly less. Interconnecting cables are fewer, and there is no need for a serializer/deserializer in any case. The data transfer rate, however, may be low in comparison to parallel communication. Nevertheless, the clock skew problem that often happens between different channels of communication is not an issue with serial communication. Compared to parallel communication, serial communication has better signal integrity. In addition, serial communication is one of the cheapest modes of communication that can be implemented, and over long-haul communication, it can provide numerous benefits.
85	Describe why Input-Output interface is required? Describe various methods for I/O interface together with their merits and demerits **Input - Output Interface** **Input Output Interface** provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of communication link is to resolve the differences that exist between the central computer and each peripheral. The Major Differences are:- 1. Peripherals are electromechnical and electromagnetic devices and CPU and memory are electronic devices. Therefore, a conversion of signal values may be needed. 2. The data transfer rate of peripherals is usually slower than the transfer rate of CPU and consequently, a synchronization mechanism may be needed. 3. Data codes and formats in the peripherals differ from the word format in the CPU and memory. 4. The operating modes of peripherals are different from each other and must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

	To Resolve these differences, computer systems include special hardware components between the CPU and Peripherals to supervises and synchronizes all input and out transfers These components are called Interface Units because they interface between the processor bus and the peripheral devices. **Memory mapped I/O and Isolated I/O** As a CPU needs to communicate with the various memory and input-output devices (I/O) as we know data between the processor and these devices flow with the help of the system bus. There are three ways in which system bus can be allotted to them : 1. Separate set of address, control and data bus to I/O and memory. 2. Have common bus (data and address) for I/O and memory but separate control lines. 3. Have common bus (data, address, and control) for I/O and memory. In first case it is simple because both have different set of address space and instruction but require more buses. **Isolated I/O –** Then we have Isolated I/O in which we Have common bus(data and address) for I/O and memory but separate read and write control lines for I/O. So when CPU decode instruction then if data is for I/O then it places the address on the address line and set I/O read or write control line on due to which data transfer occurs between CPU and I/O. As the address space of memory and I/O is isolated and the name is so. The address for I/O here is called ports. Here we have different read-write instruction for both I/O and memory. **Memory Mapped I/O –** In this case every bus in common due to which the same set of instructions work for memory and I/O. Hence we manipulate I/O same as memory and both have same address space, due to which addressing capability of memory become less because some part is occupied by the I/O.
86	**Define what programming steps are required to check when a source interrupts the computer while it is still serviced by a previous interrupt request from the same source**

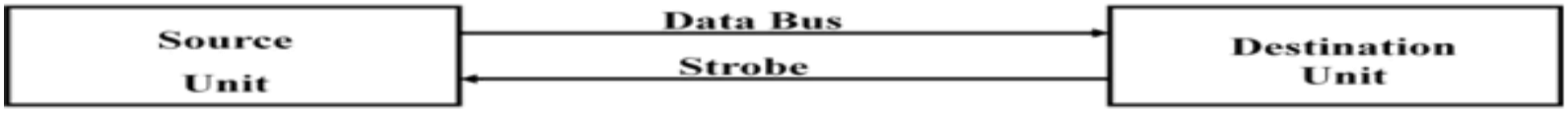
Set the mask bit belonging to the interrupt source so it can interrupt again.
At the beginning of the service routine, check the value of the return address in the stack. If it is an address within the source service program, then the same source has interrupted again while being serviced.

87 **Draw a flow chart that describes the CPU-I/O channel communication in the IBM 370.**

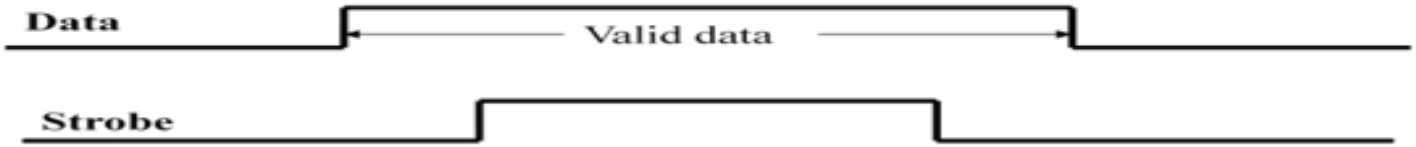


88	Describe what is basic advantage of using interrupt initiated data transfer over transfer under program control without an interrupt ?
89	Describe Strobe control and Hand shaking for Asynchronous Data Transfer. **Strobe Signal :** The strobe control method of Asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source or the destination unit. Data Transfer Initiated by Source Unit:  **Source-Initiated strobe for Data Transfer** In the block diagram fig. (a), the data bus carries the binary information from source to destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available. The timing diagram fig. (b) the source unit first places the data on the data bus. The information on the data bus and strobe signal remain in the active state to allow the destination unit to receive the data. Data Transfer Initiated by Destination Unit: In this method, the destination unit activates the strobe pulse, to informing the source to provide the data. The source will respond 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. When accepted the destination unit then disables the strobe and the source unit removes the data from the bus.



(a) Block Diagram



(b) Timing Diagram

Destination-Initiated strobe for Data Transfer

Handshaking:

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

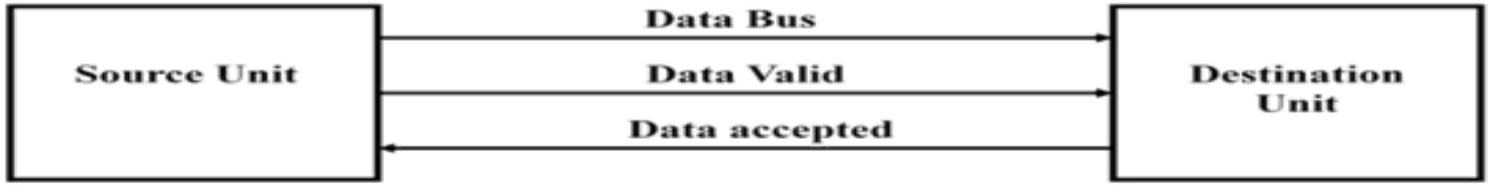
Principle of Handshaking:

The basic principle of the two-wire handshaking method of data transfer is as follow:

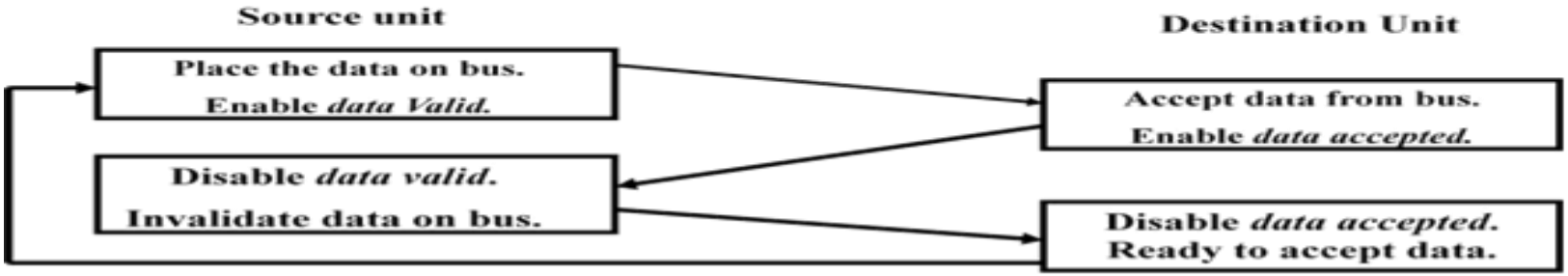
One control line is in the same direction as the data flows in the bus from the source to destination. It is used by source unit to inform the destination unit whether there a valid data in the bus. The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept the data. The sequence of control during the transfer depends on the unit that initiates the transfer.

Source Initiated Transfer using Handshaking:

The sequence of events shows 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 accepted* signal and the system goes into its initial state.



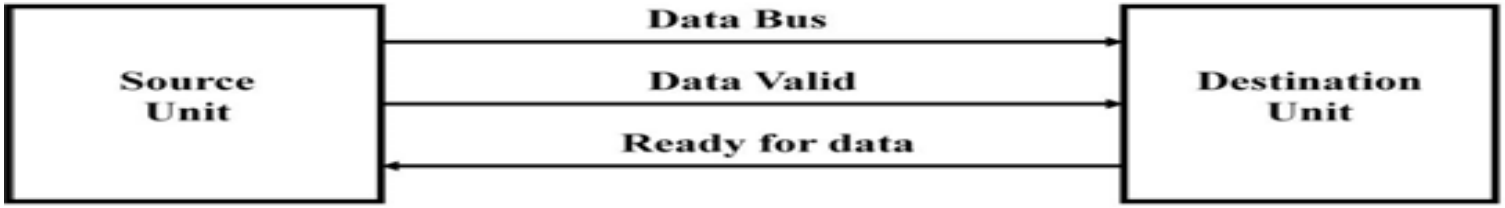
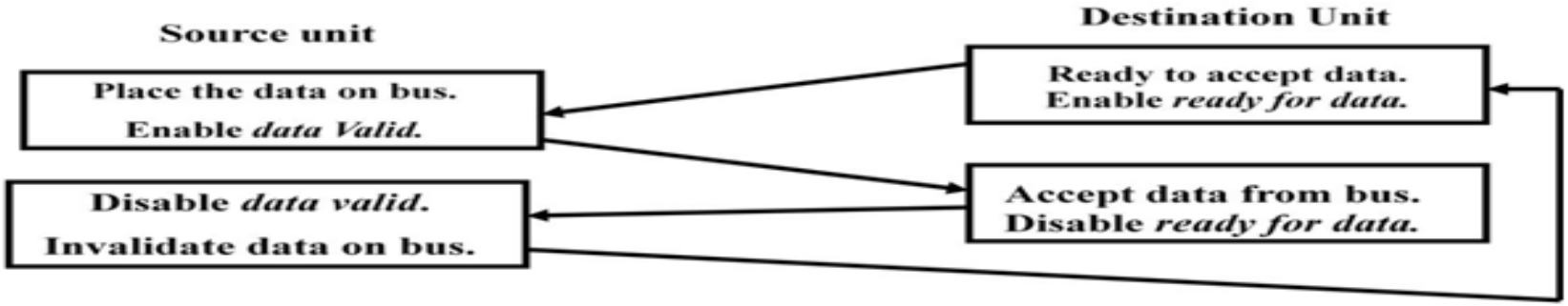
(a) Block Diagram



(b) Sequence of events

Destination Initiated Transfer Using Handshaking:

The name of the signal generated by the destination unit has been changed to *ready for data* to reflects 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.

	The only difference between the Source Initiated and the Destination Initiated transfer is in their choice of Initial sate.  (a) Block Diagram  (b) Sequence of events **Destination-Initiated transfer using Handshaking**
90	Define programmed I/O. **Programmed I/O Mode:** In this mode of data transfer the operations are the results in I/O instructions which is a part of computer program. Each data transfer is initiated by a instruction in the program. Normally the transfer is from a CPU register to peripheral device or vice-versa.

Once the data is initiated the CPU starts monitoring the interface to see when next transfer can made. The instructions of the program keep close tabs on everything that takes place in the interface unit and the I/O devices.

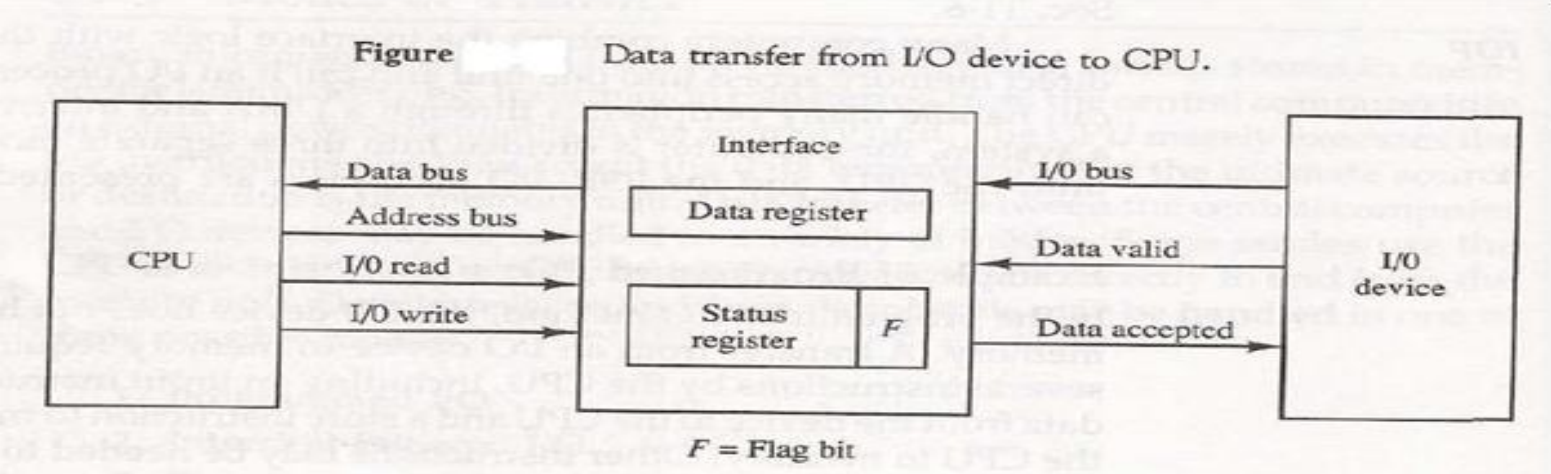
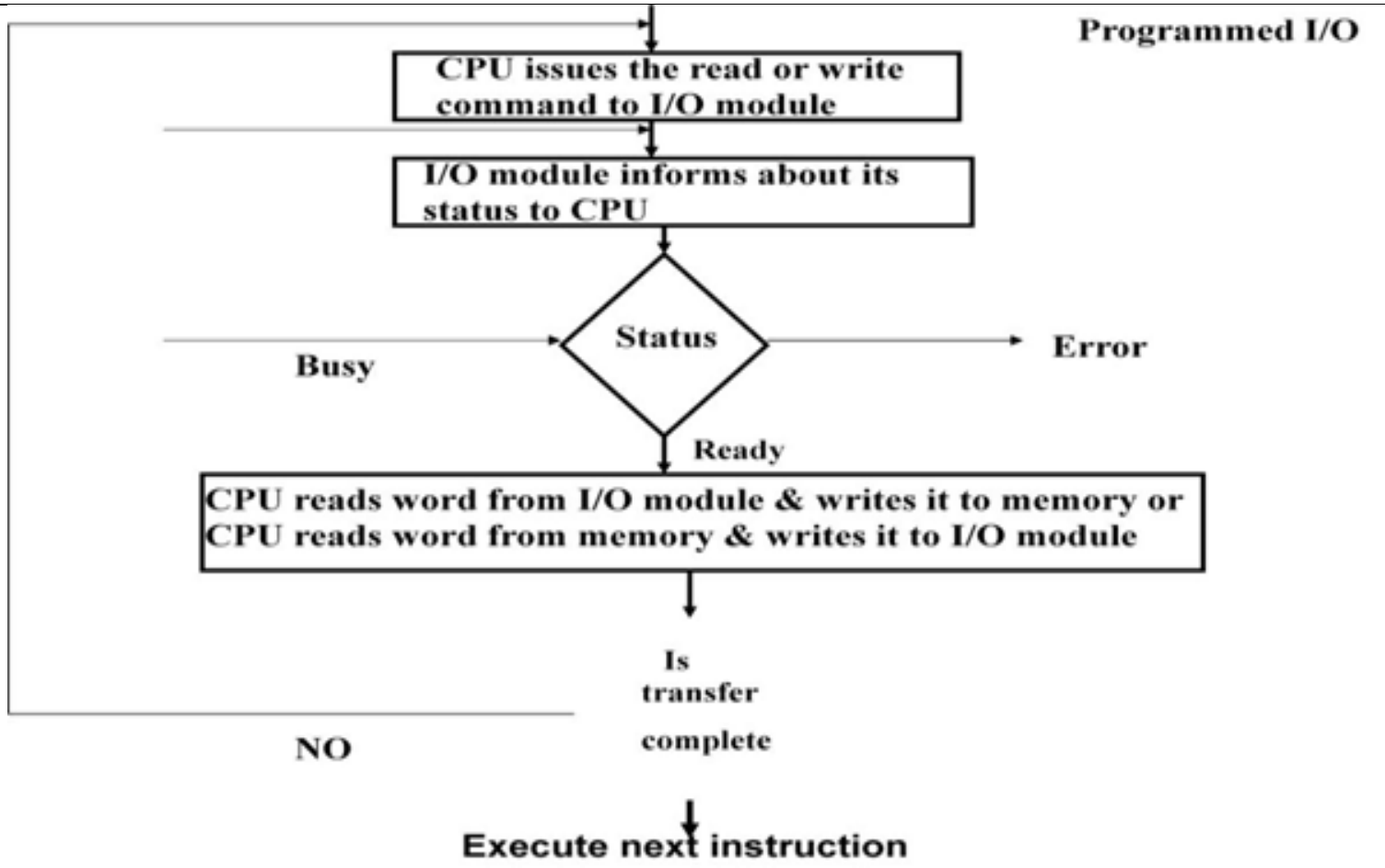


Figure Data transfer from I/O device to CPU.

- The transfer of data requires three instructions:

1. Read the status register.
2. Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
3. Read the data register.

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>91</td><td>Explain various types of peripheral devices with a suitable example.
Peripheral Devices:

The Input / output organization of computer depends upon the size of computer and the peripherals connected to it. The I/O Subsystem of the computer, provides an efficient mode of communication between the central system and the outside environment

The most common input output devices are:</td></tr>
</table>

	i) Monitor ii) Keyboard iii) Mouse iv) Printer v) Magnetic tapes The devices that are under the direct control of the computer are said to be connected online.
92	Determine the different modes of data transfer. Explain any one of them with a suitable example. **Modes of Data Transfer :** Transfer of data is required between CPU and peripherals or memory or sometimes between any two devices or units of your computer system. To transfer a data from one unit to another one should be sure that both units have proper connection and at the time of data transfer the receiving unit is not busy. This data transfer with the computer is Internal Operation. All the internal operations in a digital system are synchronized by means of clock pulses supplied by a common clock pulse Generator. The data transfer can be i. Synchronous or ii. Asynchronous When both the transmitting and receiving units use same clock pulse then such a data transfer is called Synchronous process. On the other hand, if the there is not concept of clock pulses The data transfer can be handled by various modes. some of the modes use CPU as an intermediate path, others transfer the data directly to and from the memory unit and this can be handled by 3 following ways:

i. Programmed I/O

ii. Interrupt-Initiated I/O

iii. Direct Memory Access (DMA)

Interrupt-Initiated I/O :

In this method an interrupt facility an interrupt command is used to inform the device about the start and end of transfer. In the meantime the CPU executes other program. When the interface determines that the device is ready for data transfer it generates an Interrupt Request and sends it to the computer.

When the CPU receives such an signal, it temporarily stops the execution of the program and branches to a service program to process the I/O transfer and after completing it returns back to task, what it was originally performing.

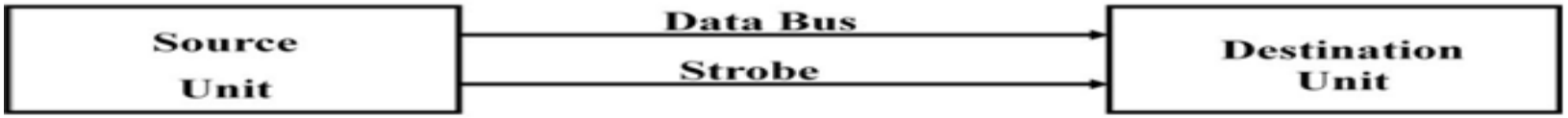
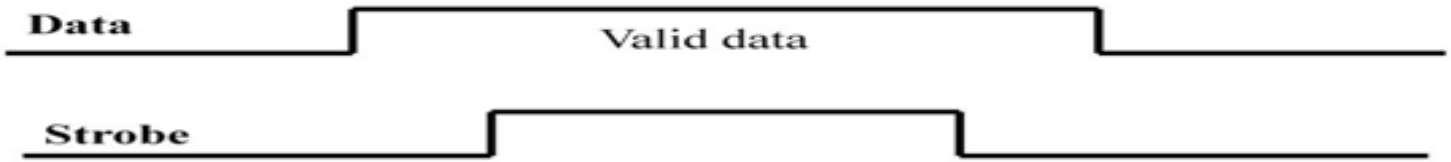
- In this type of IO, computer does not check the flag. It continue to perform its task.
- Whenever any device wants the attention, it sends the interrupt signal to the CPU.
- CPU then deviates from what it was doing, store the return address from PC and branch to the address of the subroutine.
- There are two ways of choosing the branch address:
- Vectored Interrupt
- Non-vectored Interrupt
- In vectored interrupt the source that interrupt the CPU provides the branch information. This information is called interrupt vectored.
- In non-vectored interrupt, the branch address is assigned to the fixed address in the memory.

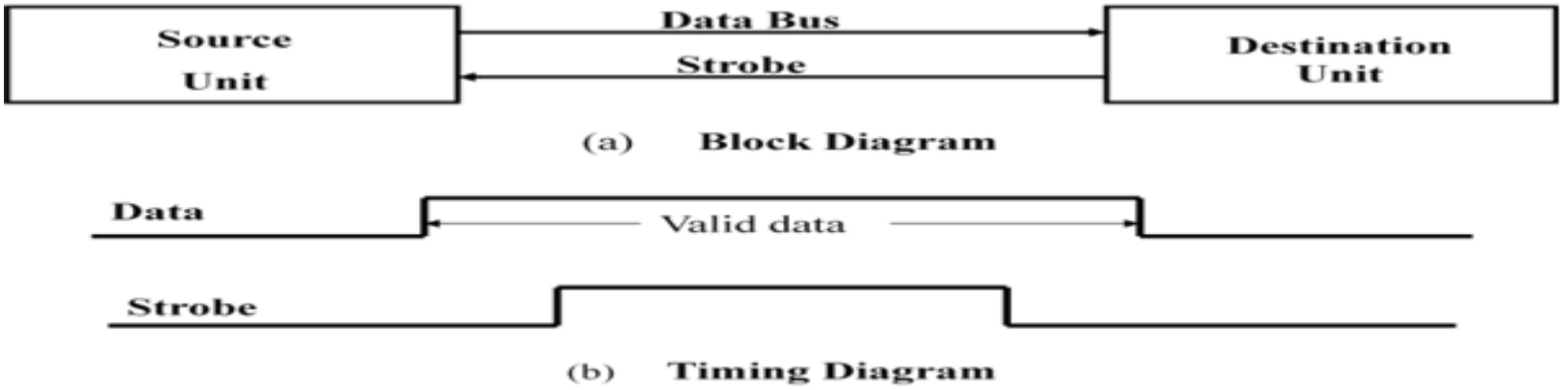
93 With the help of block diagram, explain working of Direct Memory Access (DMA). Give a brief comparison of programmed I/O & Interrupt I/O.

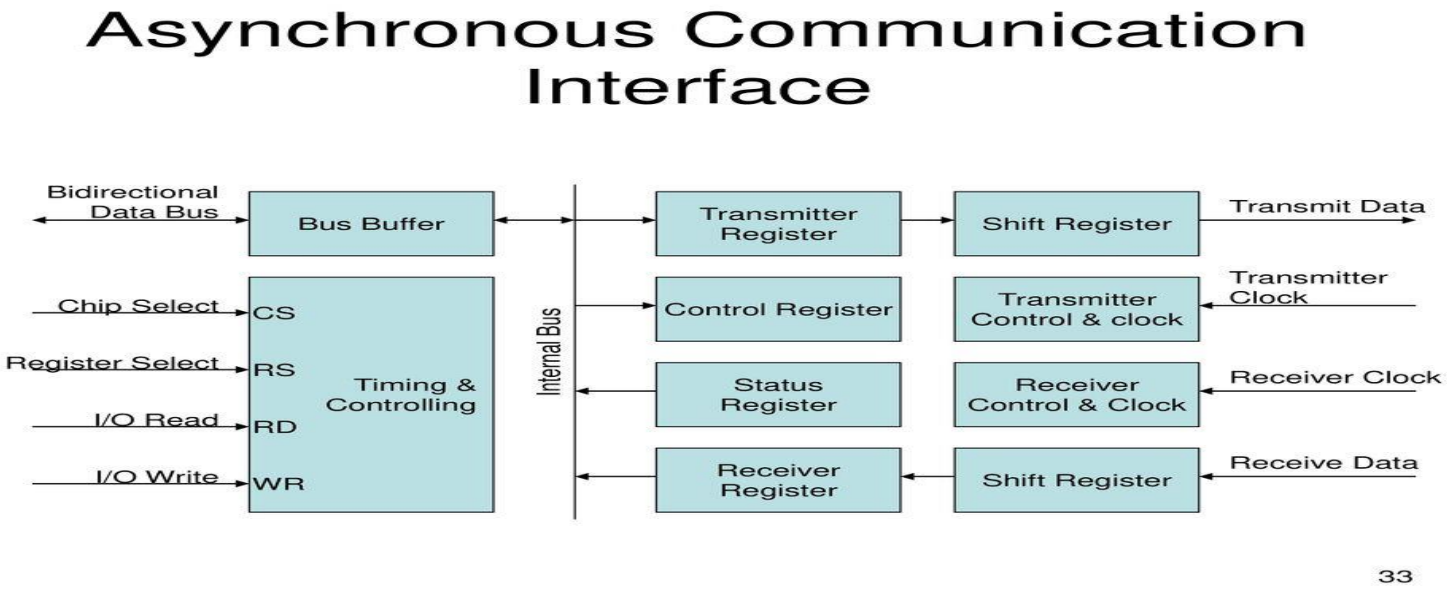
Direct Memory Access (DMA):

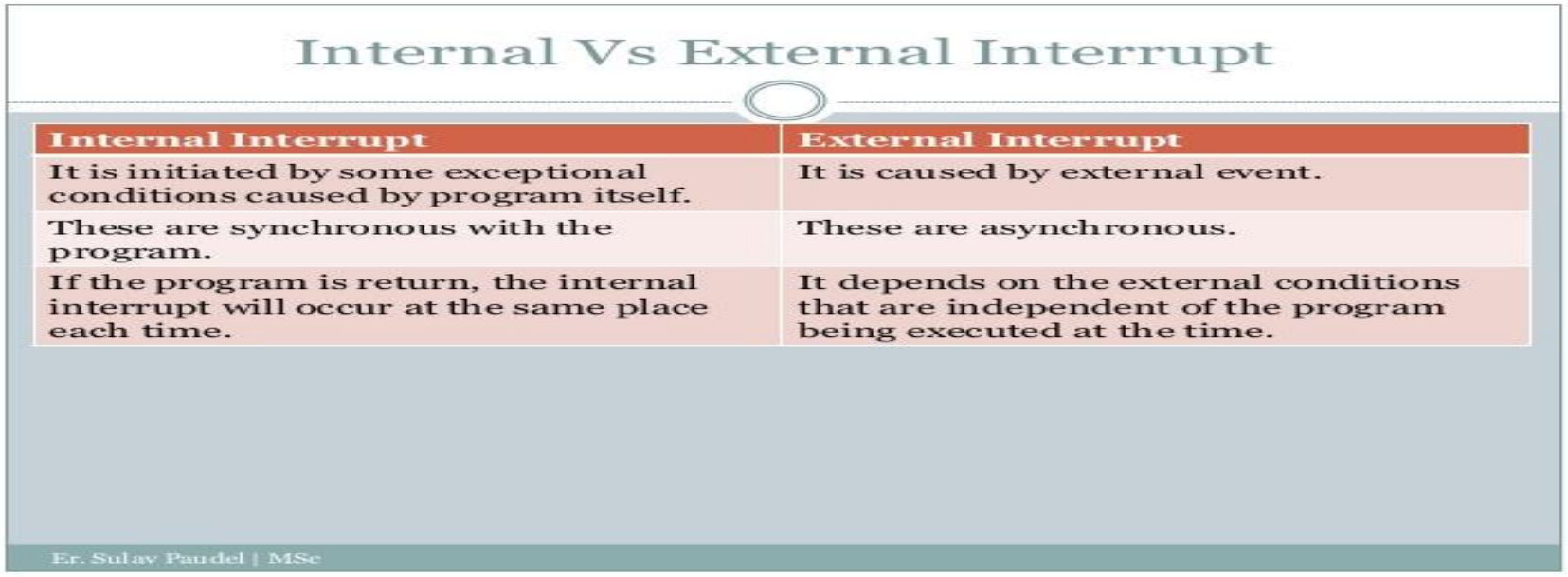
In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA).

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

	During the DMA transfer, the CPU is idle and has no control of the memory buses. A DMA Controller takes over the buses to manage the transfer directly between the I/O device and memory.
94	**Explain strobe methods with suitable diagram.** **Strobe Signal :** The strobe control method of Asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source or the destination unit. Data Transfer Initiated by Source Unit:  (a) Block Diagram  (b) Timing Diagram **Source-Initiated strobe for Data Transfer**

	 **Destination-Initiated strobe for Data Transfer**
95	Explain stan Asynchronous Communication Interface: It works as both a receiver and a transmitter. Its operation is initialized by CPU by sending a byte to the control register. The transmitter register accepts a data byte from CPU through the data bus and transferred to a shift register for serial transmission. The receive portion receives information into another shift register, and when a complete data byte is received it is transferred to receiver register. CPU can select the receiver register to read the byte through the data bus. Data in the status register is used for input and output flags. First In First Out Buffer (FIFO): A First In First Out (FIFO) Buffer is a memory unit that stores information in such a manner that the first item is in the item first out. A FIFO buffer comes with separate input and output terminals. The important feature of this buffer is that it can input data and output data at two different rates. When placed between two units, the FIFO can accept data from the source unit at one rate, rate of transfer and deliver the data to the destination unit at another rate.

	If the source is faster than the destination, the FIFO is useful for source data arrive in bursts that fills out the buffer. FIFO is useful in some applications when data are transferred asynchronously. 
96	**Describe the advantage of using input output interface ?** **Input - Output Interface** Input Output Interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of communication link is to resolve the differences that exist between the central computer and each peripheral. The Major Differences are:- 1. Peripherals are electromechnical and electromagnetic devices and CPU and memory are electronic devices. Therefore, a conversion of signal values may be needed. 2. The data transfer rate of peripherals is usually slower than the transfer rate of CPU and consequently, a synchronization mechanism may be needed.

	3. Data codes and formats in the peripherals differ from the word format in the CPU and memory
97	**Write the difference between I/O Channel and I/O processors.** The concept of DMA operation can be extended to relieve the CPU further from getting involved with the execution of I/O operations. This gives rises to the development of special purpose processor called **Input-Output Processor (IOP) or IO channel**. The Input Output Processor (IOP) is just like a CPU that handles the details of I/O operations. It is more equipped with facilities than those are available in typical DMA controller. The IOP can fetch and execute its own instructions that are specifically designed to characterize I/O transfers. In addition to the I/O – related tasks, it can perform other processing tasks like arithmetic, logic, branching and code translation. The main memory unit takes the pivotal role. It communicates with processor by the means of DMA.
98	**Explain the difference between external and internal interrupt.** 
99	Explain structure of Magnetic hard disk along with read and write operations. Magnetic Disks • Traditional magnetic disks have the following basic structure:

One or more *platters* in the form of disks covered with magnetic media. *Hard disk* platters are made of rigid metal, while "*floppy*" disks are made of more flexible plastic.

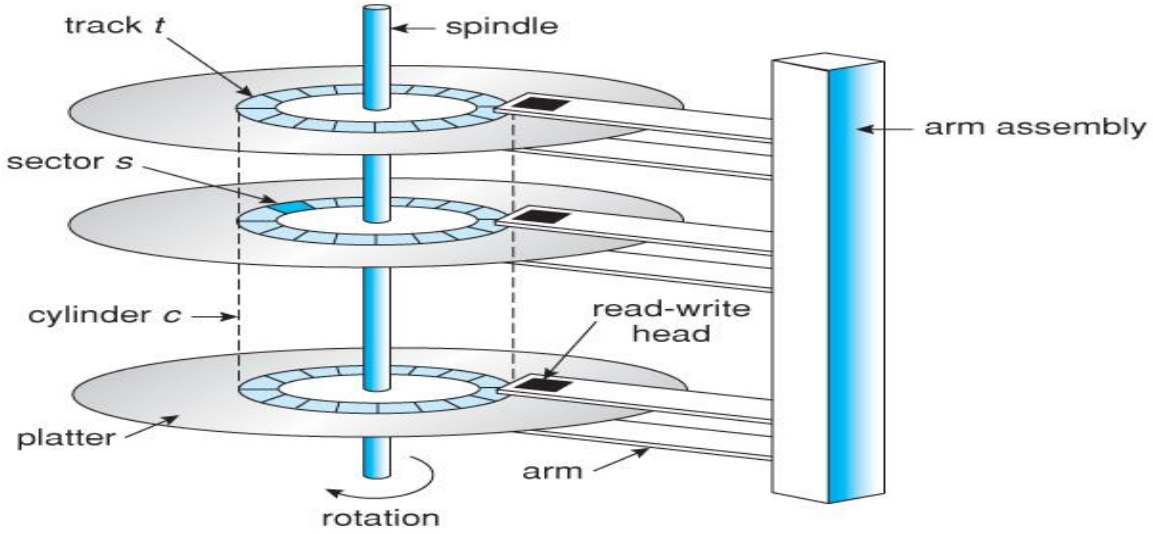
Each platter has two working *surfaces.* Older hard disk drives would sometimes not use the very top or bottom surface of a stack of platters, as these surfaces were more susceptible to potential damage.

Each working surface is divided into a number of concentric rings called *tracks.* The collection of all tracks that are the same distance from the edge of the platter, (i.e. all tracks immediately above one another in the following diagram) is called a *cylinder*.

Each track is further divided into *sectors,* traditionally containing 512 bytes of data each, although some modern disks occasionally use larger sector sizes. (Sectors also include a header and a trailer, including checksum information among other things. Larger sector sizes reduce the fraction of the disk consumed by headers and trailers, but increase internal fragmentation and the amount of disk that must be marked bad in the case of errors.)

The data on a hard drive is read by read-write *heads.* The standard configuration (shown below) uses one head per surface, each on a separate *arm,* and controlled by a common *arm assembly* which moves all heads simultaneously from one cylinder to another. (Other configurations, including independent read-write heads, may speed up disk access, but involve serious technical difficulties.)

The storage capacity of a traditional disk drive is equal to the number of heads (i.e. the number of working surfaces), times the number of tracks per surface, times the number of sectors per track, times the number of bytes per sector. A particular physical block of data is specified by providing the head-sector-cylinder number at which it is located



CSE 2nd Year KCS302 (Computer Organization) Unit 5 Solution

100	**Explain the exceptions in interrupt handling process ?.Explain in detail ?.** **Exceptions and Interrupts defined** ***Exceptions* and *interrupts* are unexpected events that disrupt the normal flow of instruction execution. An exception is an unexpected event from within the processor. An interrupt is an unexpected event from outside the processor. You are to implement exception and interrupt handling in your multicycle CPU design.** When an exception or interrupt occurs, the hardware begins executing code that performs an action in response to the exception. This action may involve killing a process, outputting a error message, communicating with an external device, or horribly crashing the entire computer system by initiating a "Blue Screen of Death" and halting the CPU. The instructions responsible for this action reside in the *operating system kernel*, and the code that performs this action is called the *interrupt handler code*. You can think of handler code as an operating system subroutine. After the handler code is executed, it may be possible to continue execution after the instruction where the execution or interrupt occurred . **Exceptions: Types** For your project, there are three events that will trigger an exception: *arithmetic overflow*, *undefined instruction*, and *system call*. *Arithmetic overflow* occurs during the execution of an add, addi, or sub instruction. If the result of the computation is too large or too small to hold in the result register, the *Overflow* output of the ALU will become high during the execute state. This event triggers an exception. *Undefined instruction* occurs when an unknown instruction is fetched. This exception is caused by an instruction in the IR that has an unknown opcode or an R-type instruction that has an unknown function code. *System call* occurs when the processor executes a syscall instruction. Syscall instructions are used to implement operating system services (functions). **Interrupts** We also want to have to ability to service external interrupts. This is useful if a device external to the processor needs attention. To do this, we'll add 2 pins to the processor. The first pin, called IRQ (interrupt request), will be an input that will allow an external device to interrupt the processor. Since we don't want the processor to service any external interrupts before it is finished executing the current instruction, we may have to make the external device wait for several clock cycles. Because of this, we need a way to tell the external device that we've serviced its interrupt. We'll solve this problem by adding the second pin, called IACK (interrupt acknowledge), that will be an output. The following waveform defines the timing for external interrupt requests.

	Dealing with multiple types of exceptions and interrupts In a situation where multiple types of exceptions and interrupts can occur, there must be a mechanism in place where different handler code can be executed for different types of events. In general, there are two methods for handling this problem: *polled interrupts* and *vectored interrupts*. 1. The processor can branch to a certain address that begins a sequence of instructions that check the cause of the exception and branch to handler code for the type of exception encountered. This is called *polled exception handling*. 2. The processor can branch to a different address for each type of exception. Each exception address is separated by only one word. A jump instruction is placed at each of these addresses that forces the processor to jump to the handler code for each type of exception. This method is called *vectored exception handling*.