1. **Explain the basic operational concepts between processor and memory (08 marks) Jan 2015**

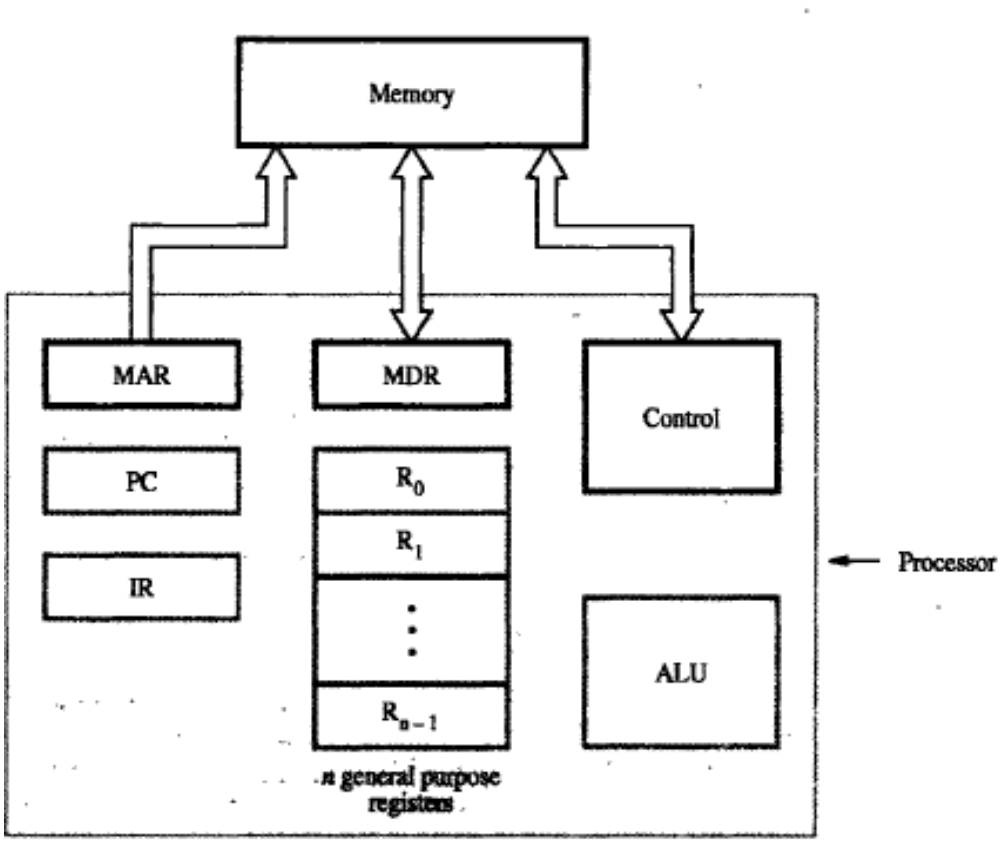
Or

With the neat block diagram, Explain the basic operational concepts of computer system (06 marks) Jul 2013

1. CONNECTION BETWEEN MEMORY AND PROCESSOR

The connection between Memory and Processor is as shown in the figure.
The Processor consists of different types of registers.

1. MAR (Memory Address Register)
2. MDR (Memory Data Register)
3. Control Unit
4. PC (Program Counter)
5. Register Array or Register File
6. IR (Instruction Register)
7. ALU (Arithmetic and Logic Unit



The functions of these registers are as follows

1. MAR

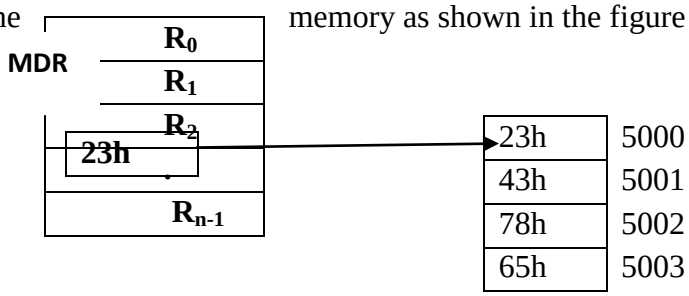
- It establishes communication between Memory and Processor

- It stores the address of the Memory Location as shown in the figure.

2. MDR

- It also establishes communication between Memory and the Processor.
- It stores the contents of the memory location (data or operand) to be

Read or write into the memory as shown in the figure



3. CONTROL UNIT

- It controls the data transfer operations between memory and the processor.
- It controls the data transfer operations between I/O and processor.

It generates control signals for Memory and I/O devices.

4. PC (PROGRAM COUNTER)

- It is a special purpose register used to hold the address of the next instruction to be executed.
- The contents of PC are incremented by 1 or 2 or 4 after either instruction or data fetched from memory
- The contents of pc are incremented by 1 for 8 bit CPU, 2 for 16 bit CPU and for 4 for 32 bit CPU

4. REGISTER ARRAY

The structure of register file is as shown in the figure

- It consists of set of registers.
- A register is defined as group of flip flops. Each flip flop is designed to store 1 bit of data.
- It is a storage element.
- It is used to store the data temporarily during the execution of the program.

- It can be used as a pointer to Memory.
- The Register size depends on the processing speed of the CPU
- EX: Register size = 8 bits for 8 bit CPU
-

5. IR (INSTRUCTION REGISTER

It holds the instruction to be executed. It output is available to the control units.

6. ALU (ARITHMETIC and LOGIC UNIT)

- It performs arithmetic and logical operations on given data.

2.1 *STEPS FOR INSTRUCTION EXECUTION*

Consider the following instruction

Ex: 1 Add LOCA, R_0

This instruction is in the form of the following instruction format

Opcode, Source, Destination

Where Add is the *operation code,* LOCA is the Memory operand and R_0 is Register operand

This instruction adds the contents of memory location LOCA with the contents of Register R_0 and the result is stored in R_0 Register.

The symbolic representation of this instruction is

[LOCA] + [R_0] → R_0

The contents of memory location LOCA and Register R_0 before and after the execution of this instruction is as follows

Before instruction execution	After instruction execution
LOCA = 23H	**LOCA = 23H**
R_0 = 22H	**R_0 = 45H**

The steps for instruction execution are as follows

1. Fetch the instruction from memory into the IR.
2. Decode the instruction
3. Access the Memory Operand
4. Access the Register Operand
5. Perform the operation according to the Operation Code.
6. Store the result into the Destination Memory location or Destination Register.

Ex:2 Add R_1, R_2, R_3

This instruction is in the form of the following instruction format

Opcode, Source-1, Source-2, Destination

Where R1 is Source Operand-1, R2 is the Source Operand-2 and R3 is the Destination. This instruction adds the contents of Register R1 with the contents of R2 and the result is placed in R3 Register.
The symbolic representation of this instruction is

[R1] + [R2] → R3

The contents of Registers R1,R2,R3 before and after the execution of this instruction is as follows.

Before instruction execution	After instruction execution
R1 = 24H	**R1 = 24H**
R2 = 34H	**R2 = 34H**
R3 = 38H	**R3 = 58H**

The steps for instruction execution is as follows

1. Fetch the instruction from memory into the IR.
2. Decode the instruction
3. Access the First Register Operand R1
4. Access the Second Register Operand R2
5. Perform the operation according to the Operation Code.
6. Store the result into the Destination Register R3.

2. How to measure the performance of a computer? Explain(06 marks) Jan 2015

- The performance of a Computer System is based on hardware design of the processor and the instruction set of the processors.
- To obtain high performance of computer system it is necessary to reduce the execution time of the processor.
- Execution time: It is defined as total time required executing one complete program.
- The performance of the processor is inversely proportional to execution time of the processor.
- More performance = Less Execution time.
- Less Performance = More Execution time.

The Performance of the Computer System is based on the following factors

1. Cache Memory
2. Processor clock
3. Basic Performance Equation
4. Pipelining and Super Scalar operation
5. Instruction set
6. Compiler

THE PERFORMANCE OF A COMPUTER CAN BE MEASURED USED BASIC PERFORMANCE EQUATION

7. Let ' T ' be *total time* required to execute the program.

8. Let 'N ' be the *number of instructions* contained in the program. To execute one instruction there are 3 steps namely 1. Fetch 2. Decode 3. Execute
9. Let ' S ' be the *average number of steps* required to one instruction.
10. Let ' R' be number of clock cycles per second generated by the processor to execute one program. Hence Processor Execution Time is given by
11. **T = N * S / R**
12. This equation is called as Basic Performance Equation.
13. For the programmer the value of T is important. To obtain high performance it is necessary to reduce the values of N and S and to increase the value of R

3. Write a note on byte addressability, Little endian and big endian (06 marks) Jan 2015
Explain Little endian and big endian assignments with necessary diagrams (05 marks) Jan 2014

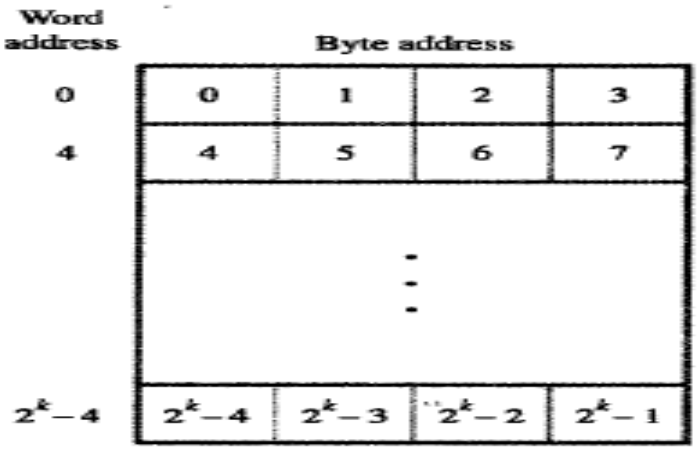
The computer performs ALU operations on 3 quantities namely bit, byte and word. It is impractical to assign addresses for 1 bit of information. Hence for practical reasons it is necessary to assign the addresses for successive bytes.

- Hence Byte Addressability is the process of assignment of address to successive bytes of the memory. The successive bytes have the addresses 1, 2, 3, 4…….2^n-1.

BIG ENDIAN ASSIGNMENT

In this technique lower byte of data is assigned to higher address of the memory and higher byte of data is assigned to lower address of the memory.

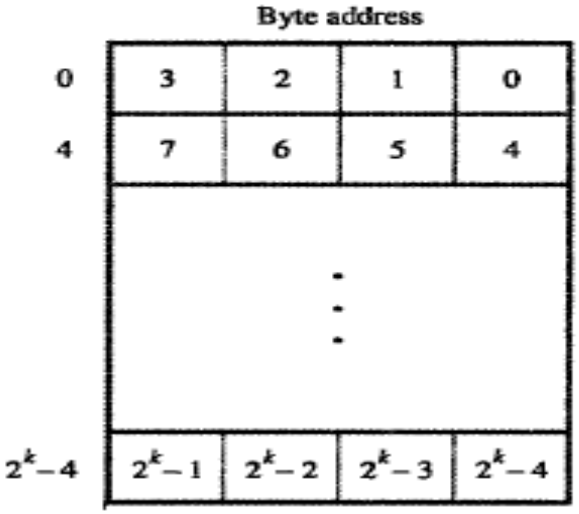
The structure of memory to represent 32 bit number for big endian assignment is as shown in the fig.



LITTLE ENDIAN ASSIGNMENT

In this technique lower byte of data is assigned to lower address of the memory and higher byte of data is assigned to higher address of the memory.

The structure of memory to represent 32 bit number for little endian assignment is as shown in the fig.



4. What is an addressing mode? Explain any four addressing modes with an example for each

(08 marks) Jan 2015

List the name, assembler syntax and addressing functions used in different types of addressing modes (10 marks) Jan 2014

Define addressing mode. Explain the following addressing modes with an example for each

i) Index addressing mode

ii) Indirect addressing mode

iii) Relative addressing mode

iv) Auto decrement addressing mode (10 marks) Jul 2013

1. Addressing Modes

The various formats of representing operand of an instruction or location of an operand is called as "Addressing Mode". The different types of Addressing Modes are

a) Register Addressing
b) Direct Addressing
c) Immediate Addressing
d) Indirect Addressing
e) Index Addressing
f) Relative Addressing
g) Auto Increment Addressing
h) Auto Decrement Addressing

a. REGISTER ADDRESSING:

In this mode operands are stored in the registers of CPU. The name of the register is directly specified in the instruction.

Ex: MOVE R_1,R_2 Where R1 and R2 are the Source and Destination registers respectively. This instruction transfers 32 bits of data from R1 register into R2 register. This instruction does not refer memory for operands. The operands are directly available in the registers.

b. DIRECT ADDRESSING

It is also called as Absolute Addressing Mode. In this addressing mode operands are stored in the memory locations. The name of the memory location is directly specified in the instruction.

Ex: MOVE X, R_1 : Where X is the memory location and R1 is the Register.

This instruction transfers 32 bits of data from memory location X into the General Purpose Register R1.

c. IMMEDIATE ADDRESSING

In this Addressing Mode operands are directly specified in the instruction. The source field is used to represent the operands. The operands are represented by # (hash) sign.

Ex: MOVE #23, R0

d. INDIRECT ADDRESSING

In this Addressing Mode effective address of an operand is stored in the memory location or General Purpose Register.

The memory locations or GPR^S are used as the memory pointers.

Memory pointer: It stores the address of the memory location.

There are two types Indirect Addressing

i) Indirect through GPR^S

ii) Indirect through memory location

i) Indirect Addressing Mode through GPR^S.

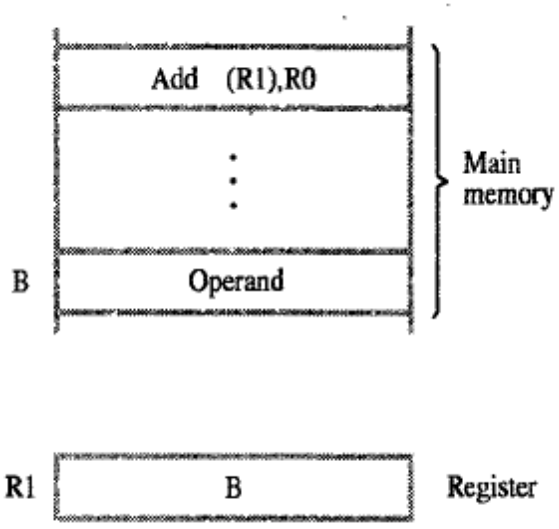
In this Addressing Mode the effective address of an operand is stored in the one of the General

Purpose Register of the CPU.

Ex: ADD (R_1), R_0 ; Where $R_{1\ and}$ $R_{0\ are}$ GPR^s.

This instruction adds the data from the memory location whose address is stored in R_1 with the contents of R_0 Register and the result is stored in R_0 register as shown in the fig.

The diagrammatic representation of this addressing mode is as shown in the fig.



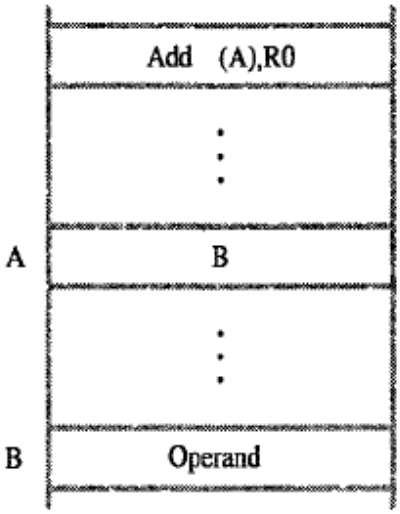
ii) Indirect Addressing Mode through Memory Location.

In this Addressing Mode, effective address of an operand is stored in the memory location.

Ex: ADD (X), R_0

This instruction adds the data from the memory location whose address is stored in 'X' memory location with the contents of R_0 and result is stored in R_0 register.

The diagrammatic representation of this addressing mode is as shown in the fig.



e. INDEX ADDRESSING MODE

In this addressing mode, the effective address of an operand is computed by adding constant value with the contents of Index Register and any one of the General Purpose Register namely R_0 to R_{n-1} can be used as the Index Register. The constant value is directly specified in the instruction.

The symbolic representations of this mode are as follows

1. X (R_i) where X is the Constant value and R_j is the GPR.

It can be represented as

EA of an operand = X + (R_i)

2. (R_i , R_J) Where R_i and R_j are the General Purpose Registers used to store addresses of an operand and constant value respectively. It can be represented as

The EA of an operand is given by

EA = (R_i) + (R_j)

3. X (R_i , R_j) Where X is the constant value and R_I and R_J are the General Purpose Registers used to store the addresses of the operands.It can be represented as

The EA of an operand is given by

EA = (R_i) + (R_j) + X

There are two types of Index Addressing Modes

i) Offset is given as constant.

ii) Offset is in Index Register.

Note : Offset : It is the difference between the starting effective address of the memory location and the effective address of the operand fetched from memory.

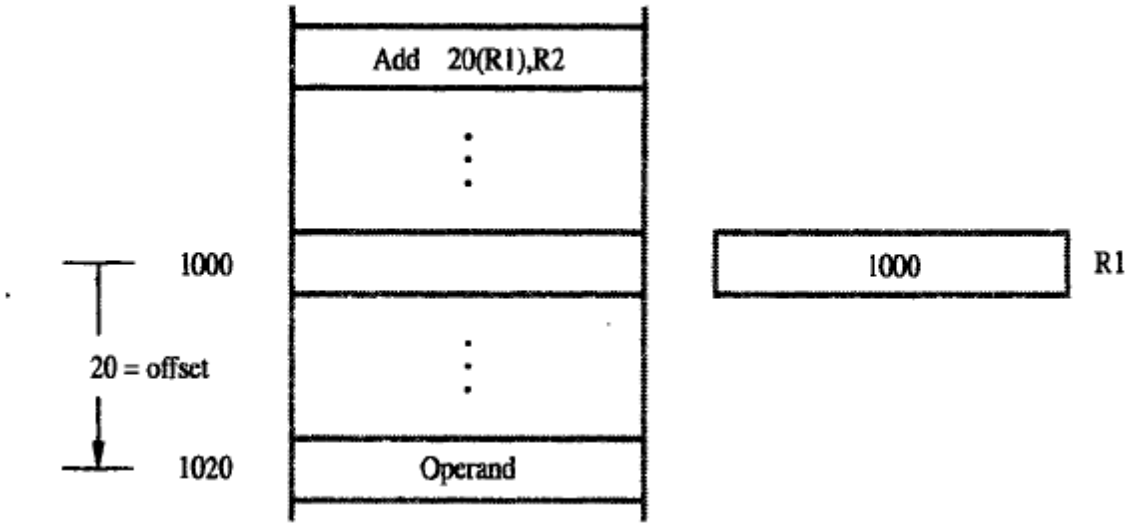
i)Offset is given as constant

Ex: ADD $20(R_1)$, R_2

The EA of an operand is given by

$EA = 20 + [R_1]$

This instruction adds the contents of memory location whose EA is the sum of contents of R_1 with 20 and with the contents of R_2 and result is placed in R_2 register. The diagrammatic representation of this mode is as shown in the fig.



ii) Offset is in Index Register

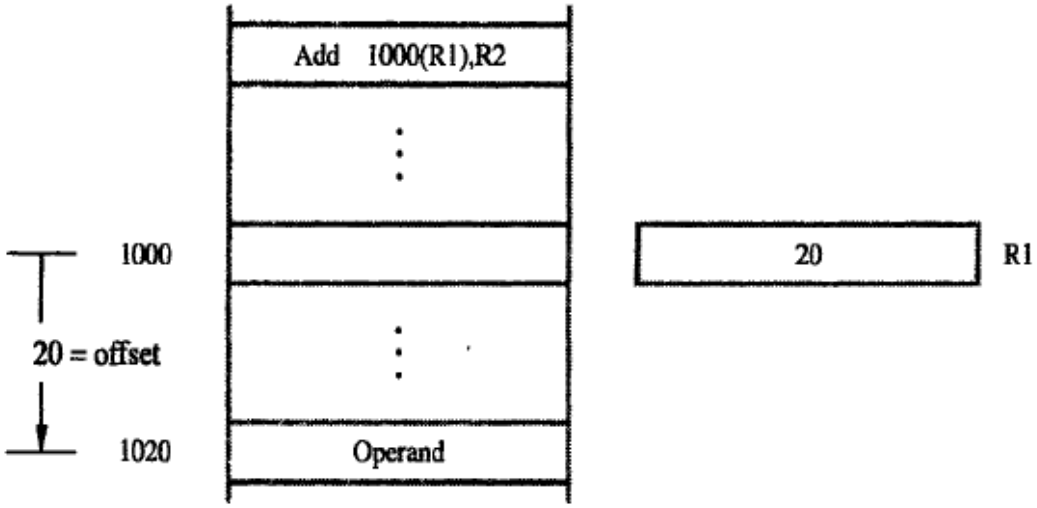
Ex: ADD $1000(R_1)$, R_2 R_1 holds the offset address of an operand.

The EA of an operand is given by

$EA = 1000 + [R_1]$

This instruction adds the data from the memory location whose address is given by [1000 + [R1] with the contents of R_2 and result is placed in R_2 register.

The diagrammatic representation of this mode is as shown in the fig.



f. RELATIVE ADDRESSING MODE:

In this Addressing Mode EA of an operand is computed by the Index Addressing Mode. This Addressing Mode uses PC (Program Counter)to store the EA of the next instruction instead of GPR.

The symbolic representation of this mode is X (PC).Where X is the offset value and PC is the Program Counter to store the address of the next instruction to be executed.

It can be represented as

EA of an operand = X + (PC).

This Addressing Mode is useful to calculate the EA of the target memory location.

g. AUTO INCREMENT ADDRESSING MODE

In this Addressing Mode , EA of an operand is stored in the one of the GPR^s of the CPU. This Addressing Mode increment the contents of memory register by 4 memory locations after operand access.

The symbolic representation is

$(R_I)+$ Where R_i is the one of the GPR.

Ex: MOVE $(R_1)+$, R_2

This instruction transfer's data from the memory location whose address is stored in R_1 into R_3 register and then it increments the contents of R_1 by 4 memory locations.

h. AUTO DECREMENT ADDRESSING MODE

In this Addressing Mode , EA of an operand is stored in the one of the GPR^s of the CPU. This Addressing Mode decrements the contents of memory register by 4 memory locations and then transfers the data to destination.

The symbolic representation is

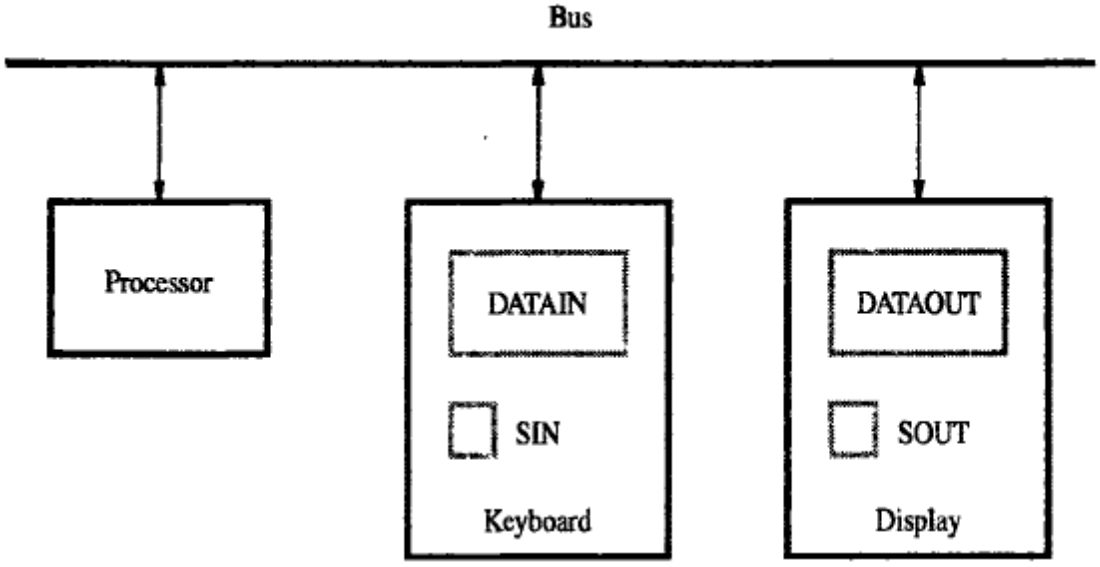
-(R_I) Where R_i is the one of the GPR.

Ex: MOVE - (R_1) , R_2

This instruction first decrements the contents of R_1 by 4 memory locations and then transfer's data to destination register.

5. With the neat block diagram, explain Basic I/O operations (05 marks) Jan 2013

The simple arrangement of connecting i/p and o/p devices into the processor is as shown in the fig



The Processor performs two operations with respect to i/o device namely

i) Input operation and

ii)Output operation

i)**Input operation:** It is the process of reading the data or instructions from the input device. The I/O subsystem consists of block of instructions to perform i/p operation.

ii)**Output operation:** It is the process of writing the data or instructions into the input device. The I/O subsystem consists of block of instructions to perform o/p operation

Consider a problem of transferring 1 byte of data from i/p device to o/p device. The i/p device transfers few characters/sec. The data transfer rate of i/p device is expressed in terms of few characters/sec. Similarly o/p device transfers thousands of characters to o/p device for display.The processor is capable of executing millions of instructions per second.

From the above analysis it is clear that there is difference in data transfer rate

To provide the synchronization between Processor, i/p device and o/p device it is necessary to follow the several steps are as follows

Steps to provide synchronization between Processor and i/p device

1.When a character is pressed on the keyboard, the ASCII value of a character is stored in DATAIN register and hence SIN flag is set to 1.

2.When SIN = 1, the processor reads the ASCII value of a character from DATAIN register into the Processor register.

3.When the character code is read from the DATAIN register into the Processor register, the SIN=0.

Steps to provide synchronization between Processor and O/p device

1.When the o/p device is ready to display the character ,the Processor transfers the character code from the processor register into the DATAOUT register.

2.The SOUT flag is set to 1 when DATAOUT register holds the character code.

3.The SOUT flag is cleared when the character code is transferred to o/p device

The i/p operation can be implemented as follows

Let R_0 be the Processor register and DATAIN be the internal register of the i/p device.

MOVE DATAIN, R0

The o/p operation can be implemented as follows

Let R_0 be the Processor register and DATAOUT be the internal register of the O/p device.

MOVE $R_{0,}$ DATAOUT

6. Explain how to encode the instructions into 32 bit words (06 marks) Jan 2015
Discuss briefly encoding of machine instructions (05 marks) Jul 2013

A list of instructions is called as program. To execute a program in processor instructions must be encoded into a compact binary form. Such encoded instructions are called as Machine instructions. The instructions that use symbolic names are called as "Assembly Language ".The Assembler converts Assembly Language instructions into Machine Language instructions.

Consider few instructions to perform several operations such as add, sub, multiply, shift, branch. These instructions msy use operands of different size such as 32 bit , 8 bit or 16 bit. The type of operation to be performed and type of operand may be specified by encoded binary pattern and is called as " opcode ".

There are 3 types of instruction formats.

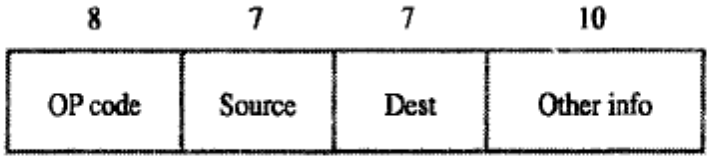
a) One word instruction format.

b) Two word instruction format.

c) Three operand instruction format.

a) One word instruction format

The one word instruction format is as shown in the fig



Ex:1 ADD R0, R1

This instruction is an example for Register Addressing mode.

This instruction transfers 32 bit data from R0 to R1. Where R0 is the Source Register and R1 is the Destination Register. The encoding of this instruction according to the above instruction format is as follows.

8 bits → opcode

4 bits → Source Register.

3 bits → Source Register addressing mode.

4 bits → Destination Register.

3 bits → Destination Register addressing mode.

10 bits → Index value or immediate operand.

Ex:2 MOVE 24(R0), R5

This instruction is an example for Register Indirect Addressing mode.

This instruction transfers 32 bit data from Memory to Register. The effective address of the operand is stored in the Register R0.

The EA of an operand is given by

EA = [R0] + 24.

The encoding of this instruction according to the above instruction format is as follows.

8 bits → opcode

4 bits → Source Register.

3 bits → Source Register addressing mode.

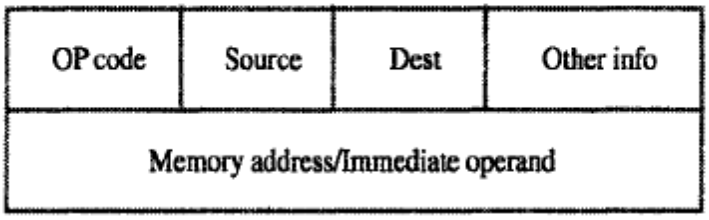
4 bits → Destination Register.

3 bits → Destination Register addressing mode.

10 bits → Index value or immediate operand.

b) Two word instruction format

The two instruction format is as shown in the fig.



Ex :1 MOVE R2, LOCA

This instruction is an example for Direct Addressing Mode.

This instruction transfers 32 bit data from Register R2 to Memory location whose symbolic name is given by LOCA.

The encoding of this instruction according to above instruction format is as follows

- This instruction format consists of 2 words.

- The 1st word is used to specify the opcode, Source register, Addressing Mode for Source,Destination Register,Addressing Mode for Destination and index value or immediate operand as follows

  8 bits → opcode

  4 bits → Source Register.

  3 bits → Source Register addressing mode.

  4 bits → Destination Memory location.

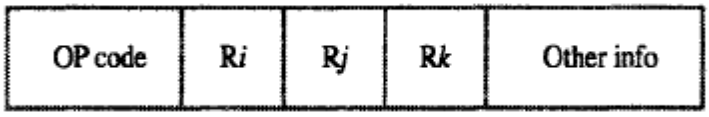
  3 bits → Destination Memory location addressing mode.

  10 bits → Index value or immediate operand.

- The 2nd word is used to specify the 32 bit Memory address or 32 bit operand.

c) **Three operand instruction format**

The three operand instruction format is as shown in the fig.



Ex: ADD R1, R2, R3

This instruction is an example for Three operand Register Addressing Mode.

This instruction adds the contents of Register R1 with the contents of R2 and result is placed in R3 Register.

The mathematical representation of this is as follows

[R1] + [R2] → [R3].

7. Explain logical and arithmetic shift instructions with an example (05 marks) Jan 2014

a. Shift instructions

The shift instructions are designed to shift the contents of processor register or memory location to left or right according to the number of bits specified in the count.

There are 2 types of shift instructions.

1. Logical Shift Left.
2. Logical Shift Right.
3. Arithmetic Shift Right

1. Logical Shift Left

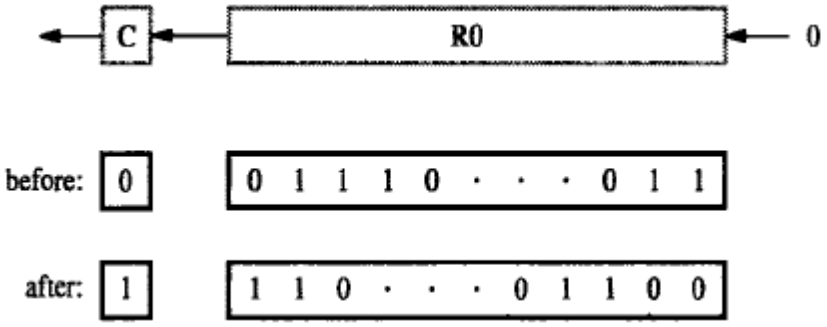
Format: opcode count, destination

The opcode indicates operation to be performed. The count can be either immediate operand or the contents of processor register. The destination can be either register operand or memory operand.

Ex: LshiftL #2, R0

This instruction shifts the contents of register R0 to left through carry by 2 bits. The count value directly specified in the instruction as an immediate operand.

The contents of R0 before and after the execution of this instruction are as shown in the fig. The shifted positions are filled with zeros from right side as shown in the fig.



2.Logical Shift Right

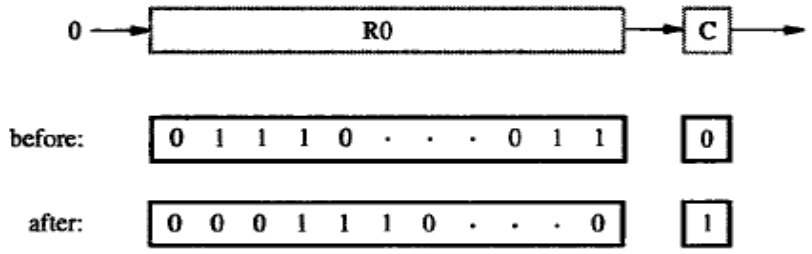
Format: opcode count, destination

The opcode indicates operation to be performed. The count can be either immediate operand or the contents of processor register. The destination can be either register operand or memory operand.

Ex: LshiftR #2, R0

This instruction shifts the contents of register R0 to right through carry by 2 bits. The count value directly specified in the instruction as an immediate operand.

The contents of R0 before and after the execution of this instruction are as shown in the fig. The shifted positions are filled with zeros from left side as shown in the fig.



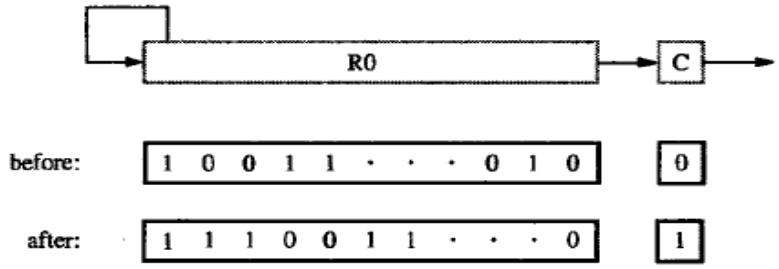
3.Arithmetic Shift Right

Format: opcode count, destination

The opcode indicates operation to be performed. The count can be either immediate operand or the contents of processor register. The destination can be either register operand or memory operand

Ex: AShiftR #2, R0

This instruction is designed to preserve the sign bit. This instruction shifts the contents of register or memory location to right through carry by number of bits specified in the count. After each shift it copies leftmost bit to Most Significant Bit. The contents of R0 before and after the execution of this instruction are as shown in the fig.



8.. Draw arrangement of single bus structure (05 marks) Jan 2014

BUS STRUCTURE

Bus: It is defined as set of parallel wires used for data communication. Each wire carries 1 bit of data. There are 3 of buses, namely

1. Address bus
2. Data bus and
3. Control bus1.

1. *Address bus :*

- It is unidirectional.
- The CPU sends the address of an I/O device or Memory device by means of this bus.

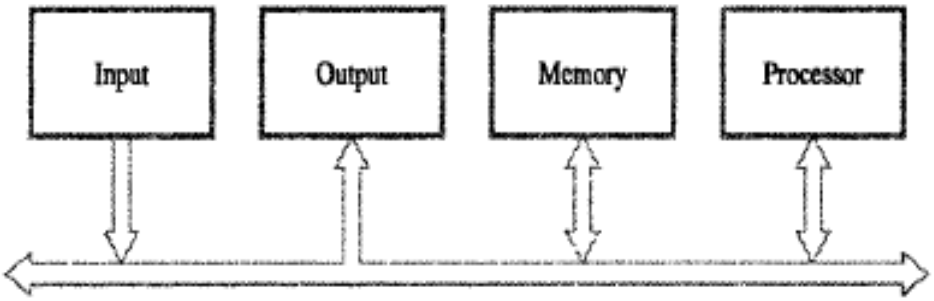
2. Data bus

- It is a bidirectional bus.
- The CPU sends data from Memory to CPU and vice versa as well as from I/O to CPU and vice versa by means of this bus.

3. *Control bus:*

- This bus carries control signals for Memory and I/O devices. It generates control signals for Memory namely MEMRD and MEMWR and control signals for I/O devices namely IORD and IOWR.
- It also generates special control signal to differentiate between Memory and I/O device. This control signal is called as M/IO.
- M/IO = 1 for Memory operations and M/IO = 0 for I/O operations

The structure of single bus organization is as shown in the figure.



- The I/O devices, Memory and CPU are connected to this bus is as shown in the figure.
- It establishes communication between two devices.

Features of Single bus organization are

- Less Expensive
- Flexible to connect I/O devices.
- Poor performance due to single bus.

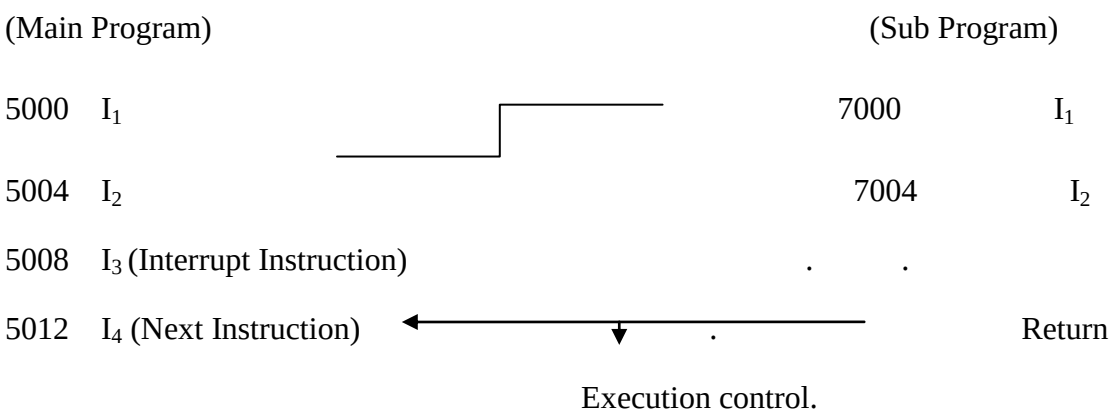
There is a variation in the devices connected to this bus in terms of speed of operation. To provide the synchronization between two devices, a buffer register is attached to each device. It holds the data temporarily during the data transfer between two devices.

8. What is an interrupt? With an example illustrate the concept of interrupts
(06arks) Jan 2015

It is an event which suspends the execution of one program and begins the execution of another program.

The arrival of interrupt causes the processor to transfer the execution control from main program to sub program.

To explain the concept of interrupt, consider the following two programs namely:



Execution control.

- **Most processor uses stack segment memory to store the address of the next instruction and is also called as return address.**

  The following steps are takes place when the interrupt related instruction is executed:

- It suspends the execution of current instruction.
- Transfer the execution control to sub program from main program.
- Increments the content of PC by 4 memory location.
- It decrements SP by 4 memory locations.
- Pushes the contents of PC into the stack segment memory whose address is stored in SP.
- It loads PC with the address of the first instruction of the sub program.

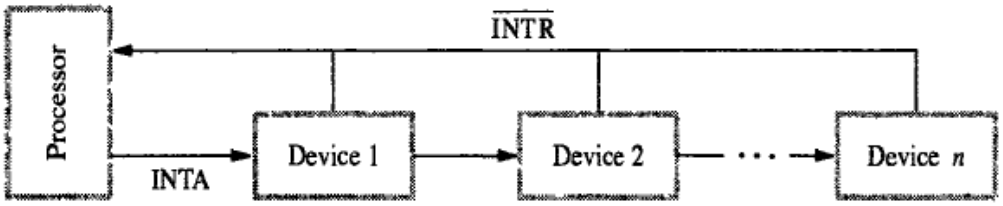
9. Explain in detail, the situations where a number of devices capable of initiating interrupts are connected to the processor? How to resolve the problems

(07marks) Jan 2015

With neat sketches, Explain various methods of handling multiple interrupts requests

(12 marks) Jul 2013

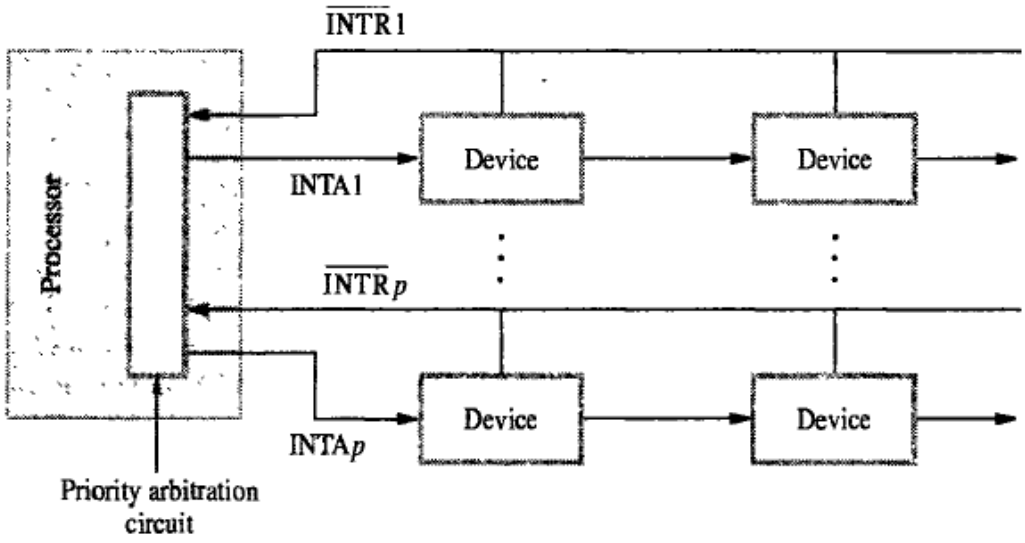
1. The daisy chain with multiple priority levels is as shown in the figure.



- The interrupt request line INTR is common to all devices as shown in the fig.
- The interrupt acknowledge line is connected in a daisy fashion as shown in the figure.
- This signal propagates serially from one device to another device.
- The several devices raise an interrupt by activating INTR signal. In response to the signal, processor transfers its device by activating INTA signal.
- This signal is received by device 1. The device-1 blocks the propagation of INTA signal to device-2, when it needs processor service.
- The device-1 transfers the INTA signal to next device when it does not require the processor service.

In daisy chain arrangement device-1 has the highest priority.

2. The second method of handling multiple devices is the multiple devices in each level as shown in the figure.



In this technique, devices are organizes in a group and each group is connected to the processor at a different priority level.

With in a group devices are connected in a daisy chain fashion as shown in the figure.

Vectored interrupt:

- To reduce the time involved in the polling process, a divice requesting an interrupt may indentify itself to the processor.
- A device requesting an interrupt may indentify itself by sending a special code to the processor over the bus.
- Then the processor can immediately transfer its service to interrupt service routine. Such interrupts are known as vectored interrupts.

10. Define bus arbitration. Explain any one of approach of bus arbitration

(08marks) Jul 2013

Explain two approaches of bus arbitration (06 marks) Jan 2015

Draw the arrangement for bus arbitration using daisy chain and explain in brief

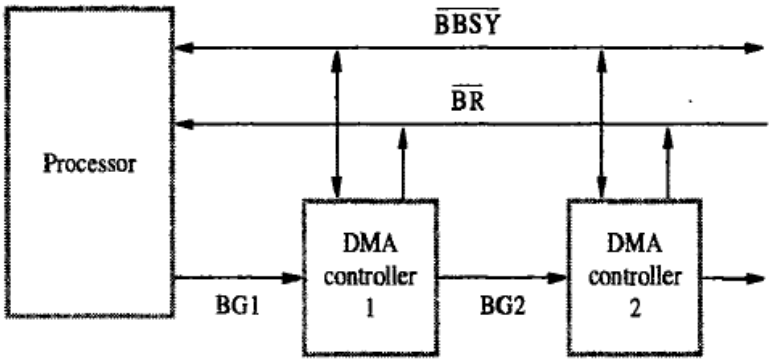
(05 marks) Jan 2014

Bus – Arbitration:

- Any device which initiates data transfer operation on bus at any instant of time is called as Bus-Master.
- When the bus mastership is transferred from one device to another device, the next device is ready to obtain the bus mastership.
- The bus-mastership is transferred from one device to another device based on the principle of priority system. There are two types of bus-arbitration technique:

a**). Centralized bus arbitration:(Daisy Chain arrangement of bus arbitration)**

In this technique CPU acts as a bus-master or any control unit connected to bus can be acts as a bus master.



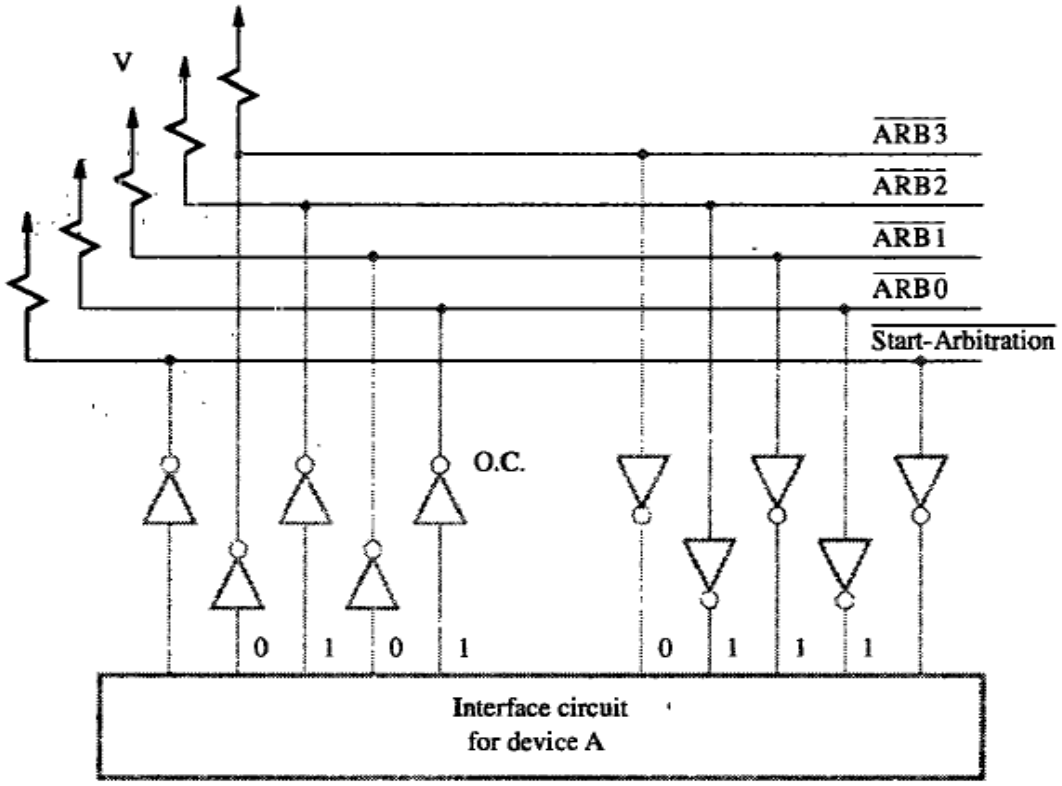
The schematic diagram of centralized bus arbitration is as shown in the fig.:

- The following steps are necessary to transfer the bus mastership from CPU to one of the DMA controller:). The DMA controller request the processor to obtain the bus mastership by activating BR (Bus request) signal
- . In response to this signal the CPU transfers the bus mastership to requested devices DMA controller1 in the form of BG (Bus grant).
- When the bus mastership is obtained from CPU the DMA controller1 blocks the propagation of bus grant signal from one device to another device.
- The BG signal is connected to DMA controller2 from DMA controller1 in as daisy fashion style is as shown in the figure.

- When the DMA controller1 transfers the bus mastership to DMA controller2 by unblocking bus grant signal.
- . When the DMA controller1 receives the bus grant signal it enables BBSY signal. When BBSY signal is set to 1 the set of devices connected to system bus doesn't have any rights to obtain the bus mastership from the CPU.

b). **Distributed bus arbitration:**

- In this technique 2 or more devices trying to access system bus at the same time may participate in bus arbitration process.
- The schematic diagram of distributed bus arbitration is as shown in the figure:

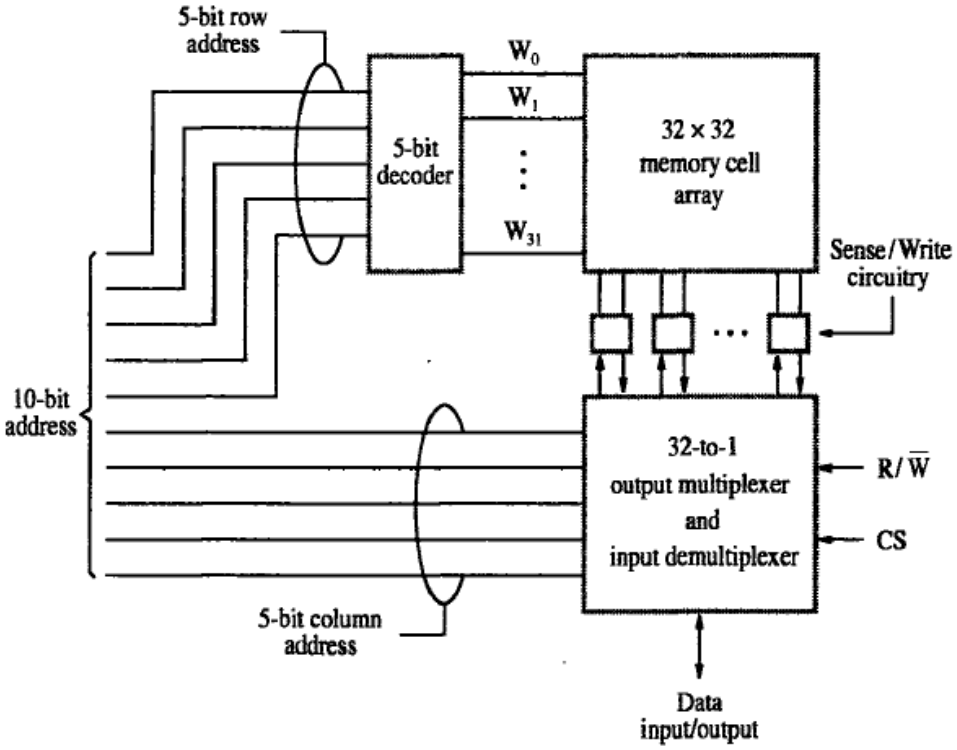


- The external device requests the processor to obtain bus mastership by enabling start arbitration signal.
- In this technique 4 bit code is assigned to each device to request the CPU in order to obtain bus mastership.
- Two or more devices request the bus by placing 4 bit code over the system bus.
- The signals on the bus interpret the 4 bit code and produces winner as a result from the CPU.

- When the input to the one driver = 1, and input to the another driver = 0, on the same bus line, this state is called as "Low level voltage state of bus".
- Consider 2 devices namely A & B trying to access bus mastership at the same time.
  Let assigned code for devices A & B are 5 (0101) & 6 (0110) respectively.
- The device A sends the pattern (0101) and device B sends its pattern (0110) to master. The signals on the system bus interpret the 4 bit code for devices A & B produces device B as a winner.
- The device B can obtain the bus mastership to initiate direct data transfer between external devices and main memory.

13. Draw 1K X 1 memory chip with neat diagram (05 Marks) Jan 2014

- ***The organization of (1Kx1) memory chip is as shown in fig:***



- The 10 bit address is divided into two groups. 5 bits for row and 5 bits for column.
- A 5 bit decoder selects a row of 32 cells.

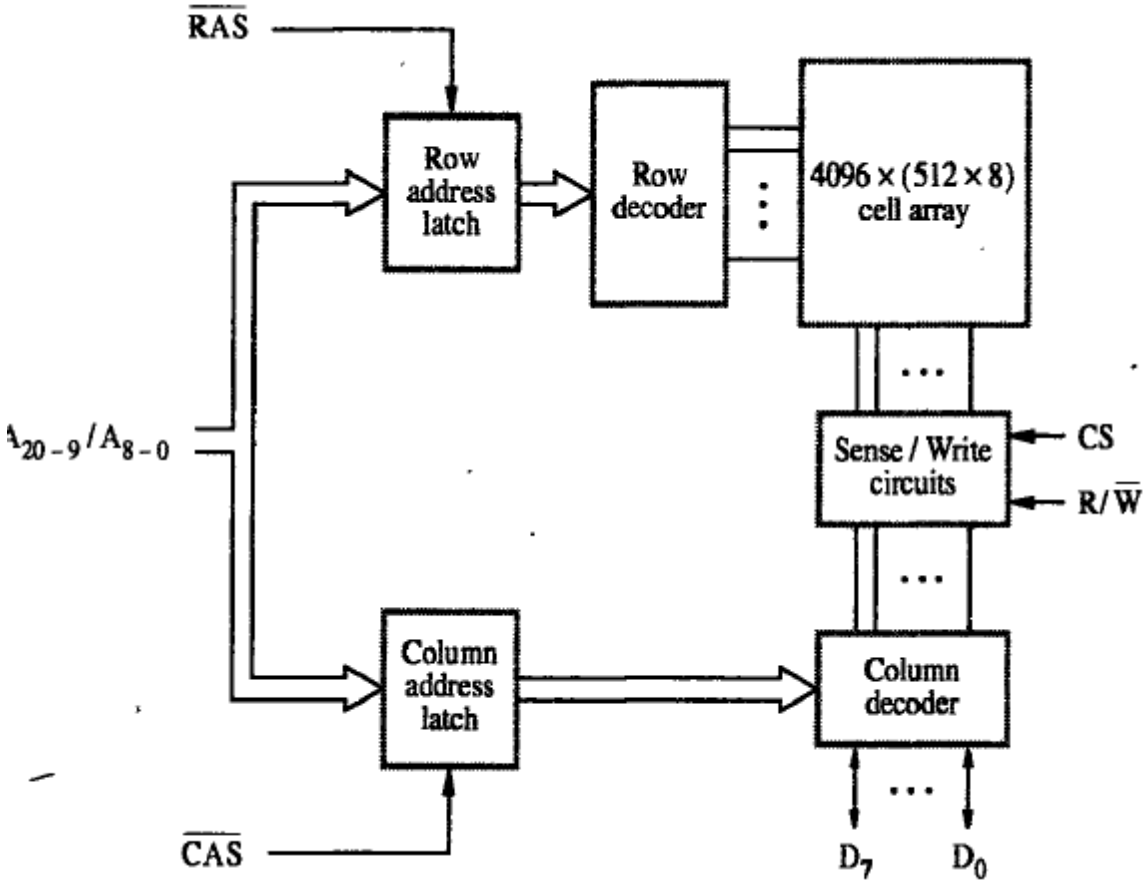
- According to the column address, only one of these cells is connected to the data line by means of input demultiplexer and output multiplexer.

14. Discuss the internal organization of 2MX8 asynchronous DRAM Chip

(09marks) Jan 2015

Internal Organization of 2Mx8 dynamic memory

- The internal organization of 2Mx8 dynamic memory chip is as shown in the fig



- 2Mx8 is equal to 2 * 1024 KB x 8

  = 2 * 1024 * 1024 x 8
- The cells are organized in the form of 4K x 4K array.
- The 4 x 1024 = 4096 cells in each row are divided into 512 groups of 8.
- Hence each row can store 512 bytes.

- Each row consists of 8 groups.
- Each row consists of 512 bits.
- Hence to select a row 12 address bits are necessary and 9 address bits are necessary to select a group.
- Hence 12 + 9 = 21 bits of address is needed to access a byte from memory.
- The higher order 12 bits are called as row address and lower order 9 bits are called as column address.
- To reduce the number of pins needed for external connections, multiplex the row address and column address on 12 pins.
- During a read or write operation. Load the column address into a column address latch by asserting CAS signal and row address into row address latch by asserting RAS signal.
- The column decoder decodes 9 bit address to select a group of 8 bits and row decoder decodes 12 bit address to select a row.
- The sense/write circuit reads or writes 1 byte of data from the selected row and selected group.

  The R/W = 1 for read operation.

  R/$\overline{W}$ = 0 for write operation.
- The CS signal is used by sense/write circuit to activate the memory chip for read and write operation

Fast page mode:

- It is the process of transferring successive bytes of data at high transfer data by applying successive CAS signals.
- This scheme allows transferring a block of data called as fast page mode.

16. Describe the different mapping techniques of cache (10 marks) Jan 2015

Briefly explain any two cache mapping techniques (06 marks) Jul 2013

With neat figure, explain direct mapping techniques (05 marks) Jan 2014

There are 3 techniques to map main memory blocks into cache memory.

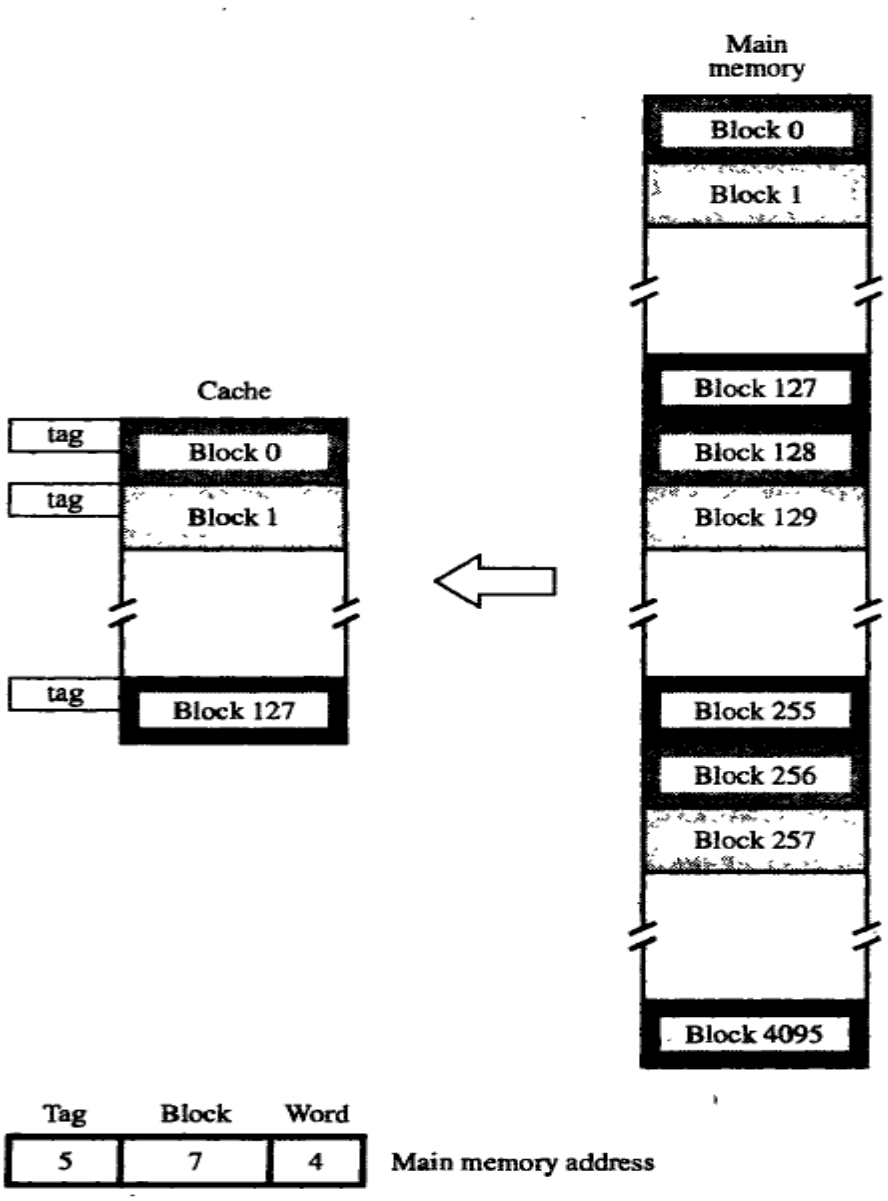
Direct mapped cache:

- The simplest way to determine cache locations in which to store memory blocks is the direct mapping technique as shown in the figure.
- The block 0 , 128 and block 256 are mapped into block-0 cache location.

Similarly blocks 1,129 and 257 from main memory are loaded into cache block-1.

- It is note that contention may arise when more than one memory blocks are loaded into single cache block, even when the cache is not full.
- The main memory block is loaded into cache block by means of memory address. The main memory address consists of 3 fields as shown in the figure.
- Each block consists of 16 words. Hence least significant 4 bits are used to select one of the 16 words.
- The 7bits of memory address are used to specify the position of the cache block, location. The most significant 5 bits of the memory address are stored in the tag bits. The tag bits are used to map one of $2^5 = 32$ blocks into cache block location.
- The higher order 5 bits of memory address are compared with the tag bits. If they match, then the desired word is in that block of the cache.

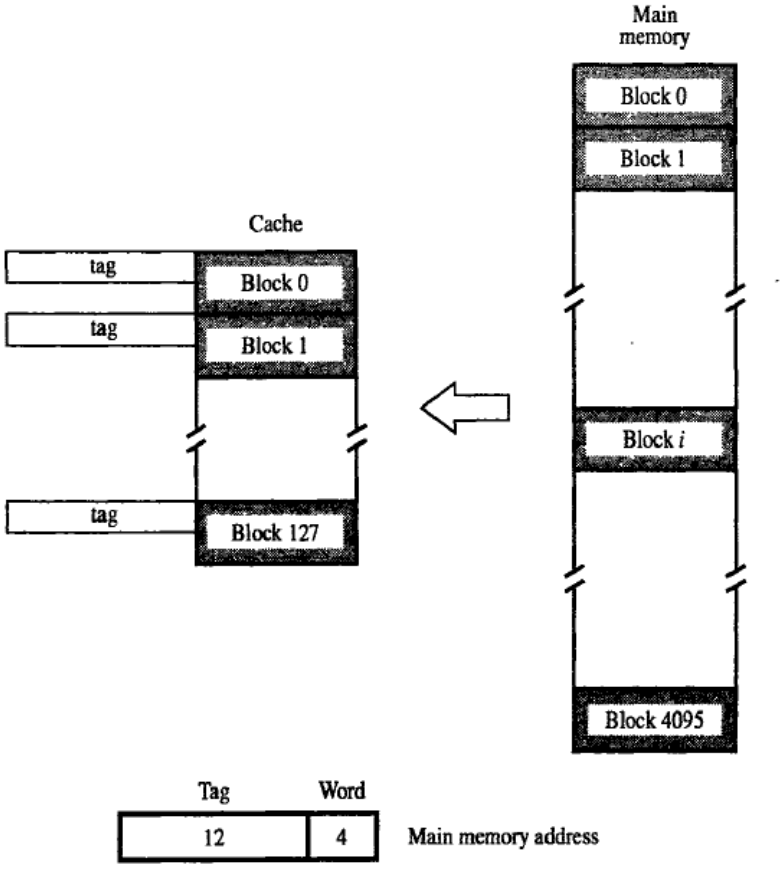
If there is no match, then the block containing the required word must first be read from the main memory and loaded into the cache. It is very easy to implement, but not flexible.



Associative Mapping:

- It is also called as associative mapped cache. It is much more flexible.
- In this technique main memory block can be placed into any cache block position.

- In this case , 12 tag bits are required to identify a memory block when it is resident of the cache memory.
- The Associative Mapping technique is illustrated as shown in the fig.

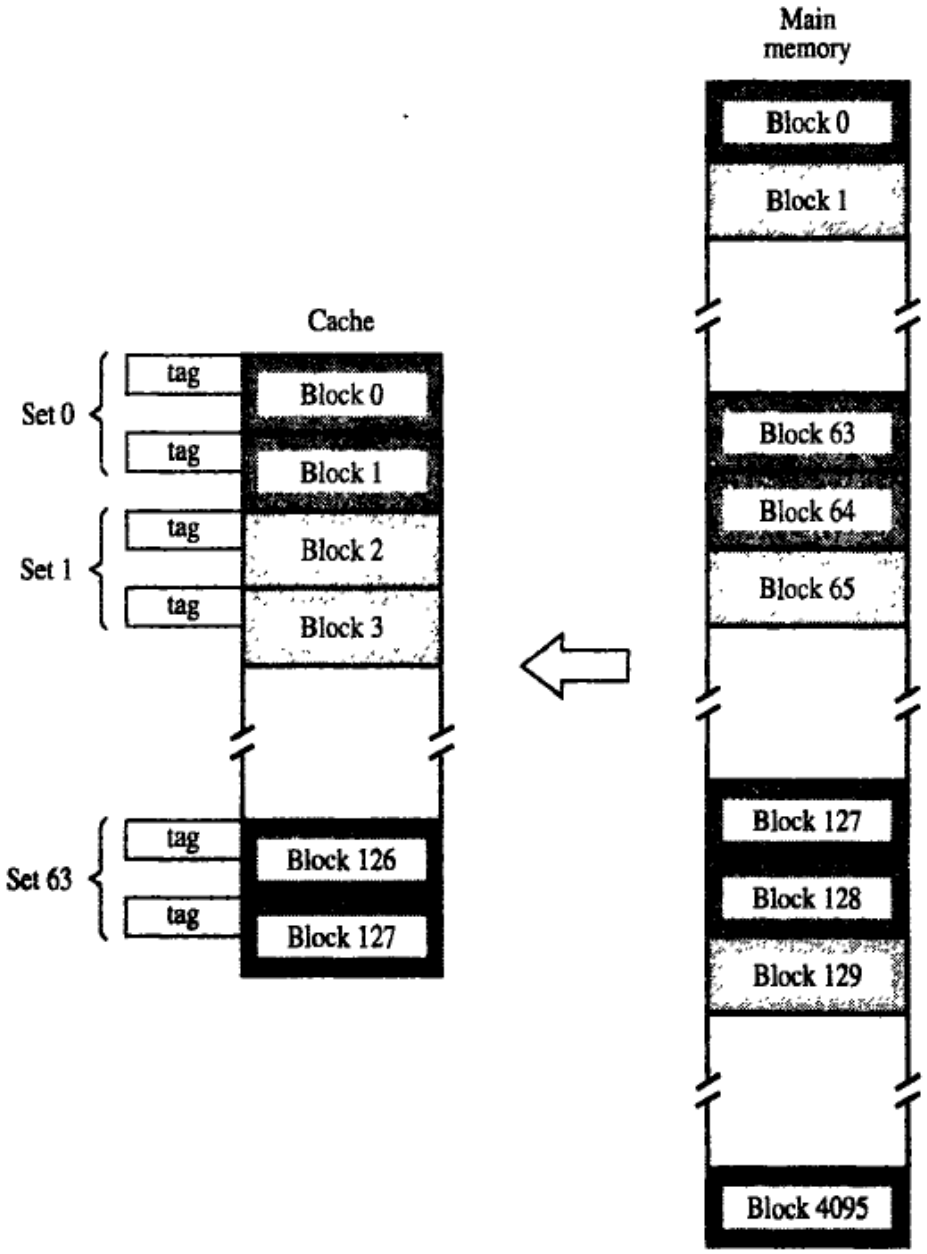


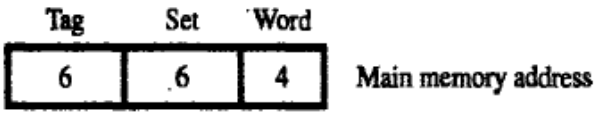
- In this technique 12 bits of address generated by the processor are compared with the tag bits of each block of the cache to see if the desired block is present. This is called as associative mapping technique.

3.Set Associative Mapping:

- It is the combination of direct and associative mapping techniques.
- The blocks of cache are divided into several groups. Such a groups are called as sets.
- Each set consists of two cache blocks. A memory block is loaded into one of the cache sets.
- The main memory address consists of three fields, as shown in the figure.

- The lower 4 bits of memory address are used to select a word from a 16 words.
- A cache consists of 64 sets as shown in the figure. Hence 6 bit set field is used to select a cache set from 64 sets.
- The tag field (6 bits) of memory address is compared with the tag fields of each set to determine whether memory block is available or not.
- The following figure clearly describes the working principle of Set Associative Mapping technique.

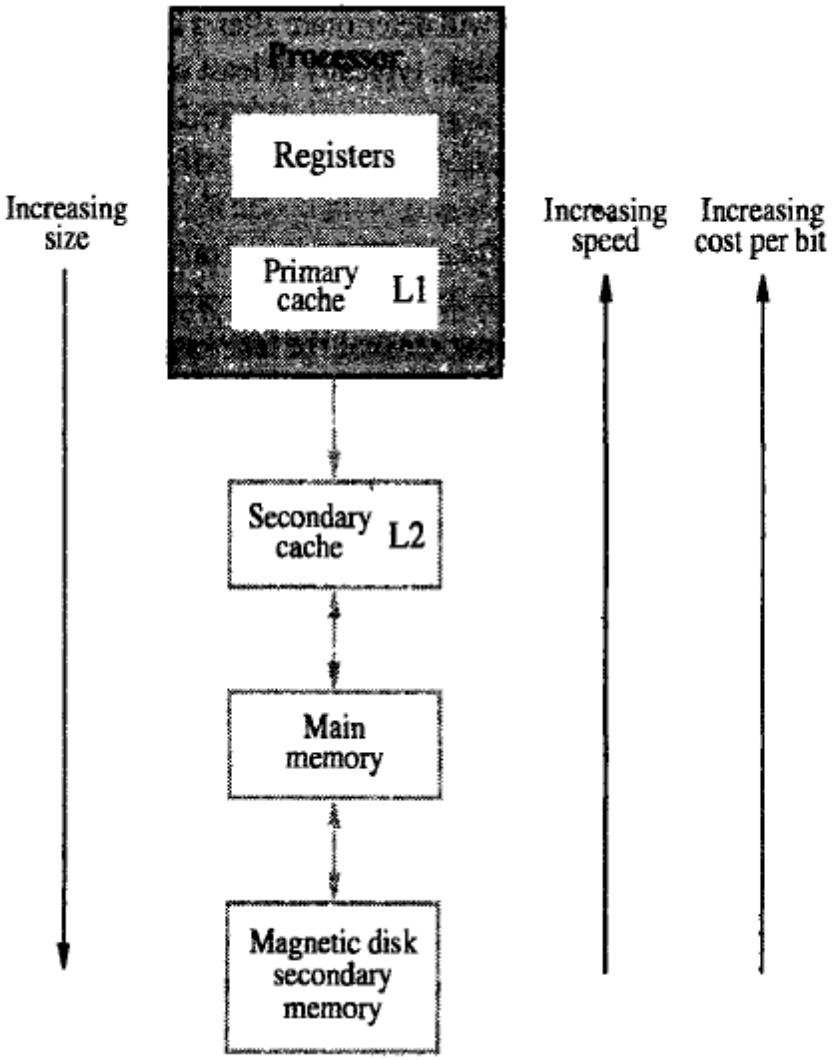

Main memory
Block 0
Block 1
Block 63
Block 64
Block 65
Block 127
Block 128
Block 129
Block 4095
Cache
Set 0
tag
Block 0
tag
Block 1
Set 1
tag
Block 2
tag
Block 3
Set 63
tag
Block 126
tag
Block 127


Tag
Set
Word
6
6
4
Main memory address

17. Show with diagram, memory hierarchy with respect to speed, size and cost

(05 marks) Jan 2014

The B.D of memory hierarchy is as shown in the fig:



- Registers: The fastest access is to data held in registers. Hence registers are part of the memory hierarchy.

  More speed, small size and cost per bit is also more.

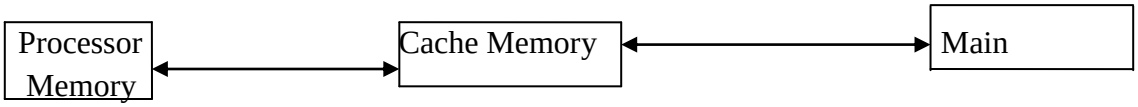
- At the next level of hierarchy, small amount of memory can be directly implemented on the processor chip.
- This memory is called as processor cache. It holds the copy of data and instructions.
- There are 2 levels of caches viz level-1 and level-2. Level-1 cache is part of the processor and level-2 cache is placed in between level-1 cache and main memory.
- The level-2 cache is implemented using SRAM chips.

- The next level in the memory hierarchy is called as main memory. It is implemented using dynamic memory components. The main memory is larger but slower than cache memory. The access time for main memory is ten times longer than the cache memory

- The level next in the memory hierarchy is called as secondary memory. It holds huge amount of data.

- **Cache Memory:**

  It is the fast access time located in between processor and main memory as shown in the fig. It is designed to reduce the access time.



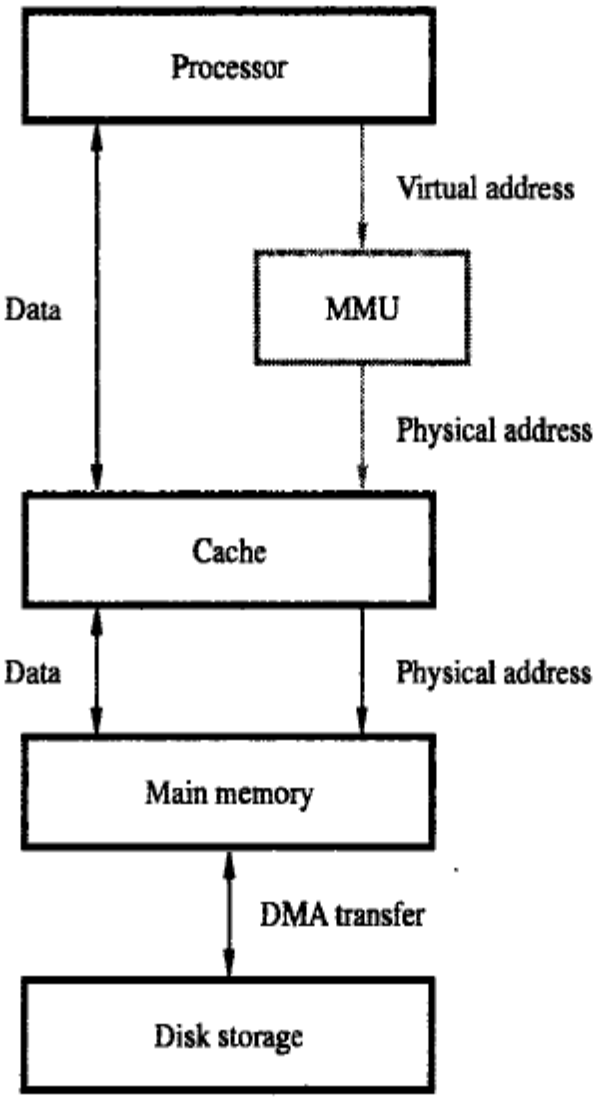
- The cache memory holds the copy of data and instructions.
- The processor needs less access time to read the data and instructions from the cache memory as compared to main memory.
- Hence by incorporating cache memory, in between processor and main memory, it is possible to enhance the performance of the system.

18. With the neat diagram, explain the translation of a virtual address into physical address

(08 marks) Jul 2013

- In case of modern computers, the address space specified by the processor, is larger than the size of the main memory.
- Ex: If the processor consists of 32 address lines, capable of addressing 2^{32} = 4GB of memory.
- The size of the main memory is in the range of hundred megabytes to 1G bytes.
- When the program is too large and hence part of this program is stored in secondary storage devices such as magnetic disks.
- Techniques that automatically transfers program and data blocks from secondary memory into the physical memory for execution are called as "virtual memory techniques".
- Hence the processor reference an instruction and data are completely independent from the physical main memory.
- The processor issues binary address for instructions and data are called as "virtual address or logical address". The memory management unit converts logical address. into physical address

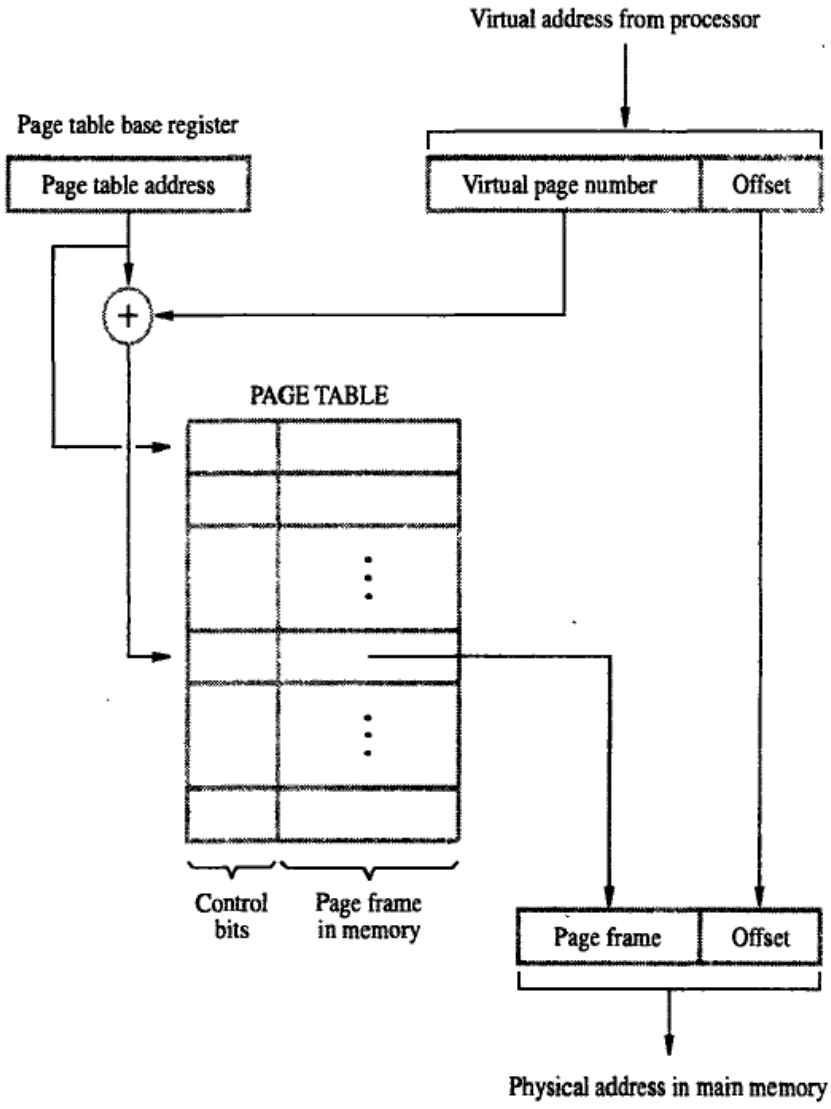
- **The virtual memory organization is as shown in the figure:**



- The MMU converts virtual address into physical address.
- If the data and programs are stored in the main memory. CPU directly fetches data and programs from the main memory.
- If the data or instructions are not in the main memory, the MMU instructs the operating system to bring the data into the main memory from the secondary memory.
- Transfer of data between the disk and main memory by means of DMA scheme.

Address Translation:

- It is the process of converting virtual address into physical address. The MMU converts virtual address into physical address.



- The virtual memory address translation mechanism is as shown in the figure.
-
- To explain the concept of address translation, assume that all programs and data are composed of fixed length units called as pages. Each page consists of block of words. The size of page ranges from 2k to 16 kbytes.

- The virtual address generated by the processor is either for instruction fetch or for operand fetch.

- The virtual address can be reffered as two fields viz virtual page number (higher order bits) and offset (lower order bits).
- The offset specifies the location of byte within the page.
- The information about the main memory location of each page is stored in page table.
- It consists of 2 fields,namely control bits and page frame in memory.
- The page table holds the main memory address of the page and current status of the page.
- A set of main memory locations to hold one page is called as "Page Frame".
- The page table base register holds the starting address of the page table.
- The page table information must be stored in MMU. But this information is not stored in MMU, because MMU is implemented as part of the processor chip
- **The MMU converts virtual address into physical address by means of page table information and offset.**
- The page table information is stored in small cache memory and is called as " Translation Lookaside Buffer".

19. Perform signed multiplication of number (-12) and (-11) using booths algorithm

(08 marks) Jul 2013

Perform multiplication for -13 and +9 using Booths algorithm (06 marks) Jan 2015

Refer Class Notes for the problems

20. Explain with figure 4 bit carry look ahead adder (10 marks) Jan 2014

Write the logic diagram for 4 bit carry look ahead adder. Explain the operation

(06 marks) Jan 2015

Carry-Look ahead Addition:

As it is clear from the previous discussion that a parallel adder is considerably slow & a fast adder circuit must speed up the generation of the carry signals, it is necessary to make the carry input to each stage readily available along with the input bits. This can be achieved either by propagating the previous carry or by generating a carry depending on the input bits & previous carry. The logic expressions for s_i (sum)

and c_{i+1} (carry-out) of stage **i**th are

$$s_i = x_i \oplus y_i \oplus c_i$$

and

$$c_{i+1} = x_i y_i + x_i c_i + y_i c_i$$

Factoring the second equation into

$$c_{i+1} = x_i y_i + (x_i + y_i) c_i$$

we can write

$$c_{i+1} = G_i + P_i c_i$$

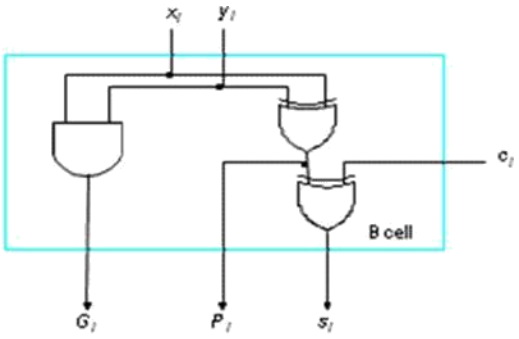
where

$$G_i = x_i y_i \quad \text{and} \quad P_i = x_i + y_i$$

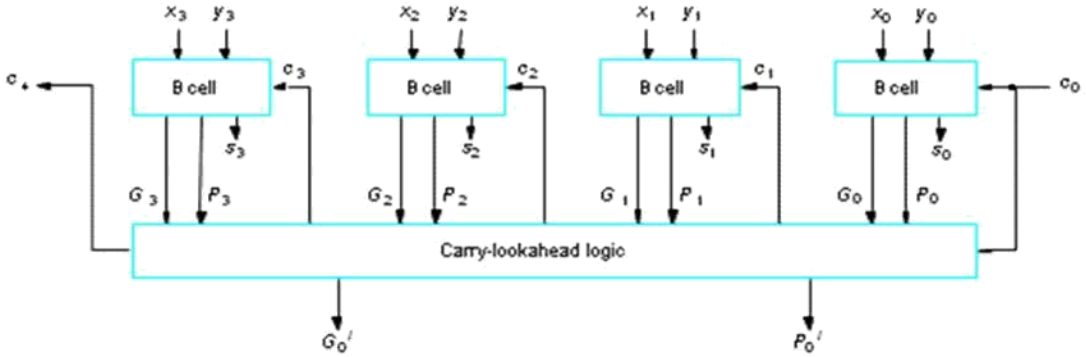
The above expressions G_i and P_i are called carry generate and propagate functions for stage **i**. If the generate function for stage **i** is equal to 1, then $c_{i+1} = 1$, independent of the input carry, c_i. This occurs when both x_i and y_i are 1. The propagate function means that an input carry will produce an output carry when either x_i **or** y_i **or** both equal to 1. Now, using Gi & Pi functions we can decide carry for ith stage even before its previous stages have completed their addition operations. All G_i and P_i functions can be formed independently and in parallel in only one gate delay after the Xi and Yi inputs are applied to an n-bit adder. Each bit stage contains an AND gate to form G_i, an OR gate to form P_i and a three-input XOR gate to form s_i. However, a much simpler circuit can be derived by considering the propagate function as $P_i = x_i \square y_i$, which differs from $P_i = x_i + y_i$ only when $x_i = y_i = 1$ where $G_i = 1$ (so it does not matter whether P_i is 0 or 1). Then, the basic diagram in Figure-5 can be used in each bit stage to predict carry ahead of any stage completing its addition.

Consider the c_{i+1}expression,

$$c_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} c_{i-1}$$



(a) Bit-stage cell



(b) 4-bit adder

his is because, Ci = (Gi-1 + Pi-1Ci-1).

Further, Ci-1 = (Gi-2 + Pi-2Ci-2) and so on. Expanding in this fashion, the final carry expression can be written as below;

$$C_{i+1} = G_i + P_i G_{i-1} + P_i P_{i-1} G_{i-2} + \ldots + P_i P_{i-1} \ldots P_1 G_0 + P_i P_{i-1} \ldots P_0 C_0$$

Thus, all carries can be obtained in three gate delays after the input signals Xi, Yi and Cin are applied at the inputs. This is because only one gate delay is needed to develop all P_i and G_i signals, followed by two gate delays in the AND-OR circuit (SOP expression) for c_{i+1}. After a further XOR gate delay, all sum bits are available. Therefore, **independent of *n, the number of stages,* the n-bit *addition* process requires only four gate delays**.

FIG-5: 4 bit carry look ahead adder

Now, consider the design of a 4-bit parallel adder. The carries can be implemented as

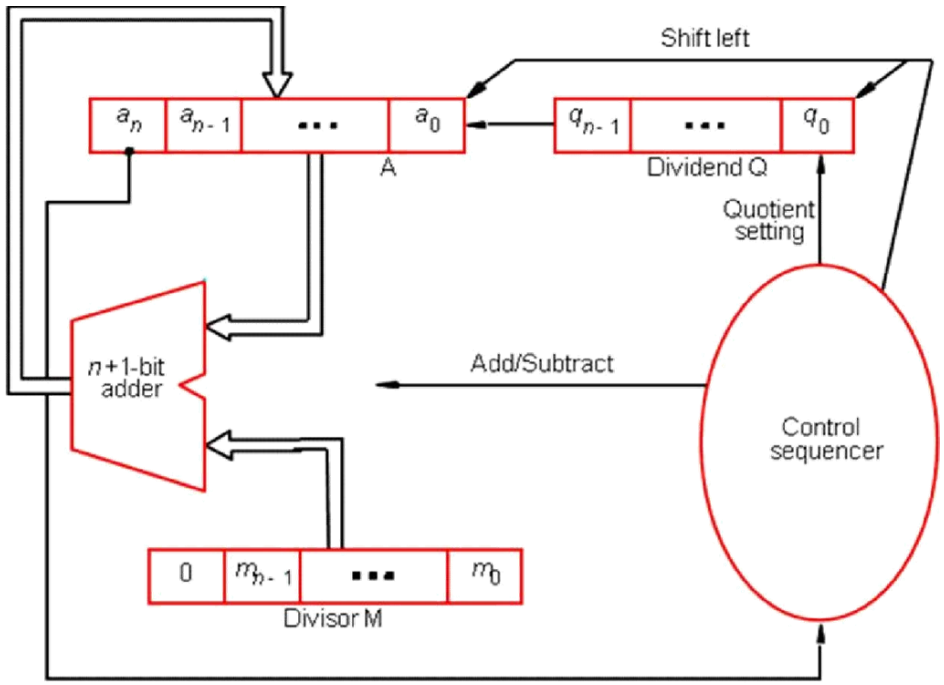
$$c_1 = G_0 + P_0c_0$$
$$c_2 = G_1 + P_1G_0 + P_1P_0c_0$$
$$c_3 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0c_0$$
$$c_4 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0c_0$$

;i = 0 ;i = 1 ;i = 2 ;i = 3

The complete 4-bit adder is shown in Figure 5b where the B cell indicates Gi, Pi & Si generator. The carries are implemented in the block labeled carry look-ahead logic. An adder implemented in this form is called a *carry look ahead adder*. Delay through the adder is 3 gate delays for all carry bits and 4 gate delays for all sum bits. In comparison, note that a 4-bit ripple-carry adder requires 7 gate delays for *S3*(2n-1) and 8

gate delays(2n) for $\mathbf{c_4}$.

21. With figure, explain circuit arrangements for binary division (05 marks) Jan 2014



Circuit arrangement for binary division.

Restoring Division:

Figure above shows a logic circuit arrangement that implements restoring division. Note its similarity to the structure for multiplication that was shown in Figure 8. An n-bit positive divisor is loaded into register M and an n-bit positive dividend is loaded into register Q at the start of the operation. Register A is set to 0. After the division is complete, the n-bit quotient is in register Q and the remainder is in register A. The required subtractions are facilitated by using 2's-complement arithmetic. The extra bit position at the left end of both A and M accommodates the sign bit during subtractions. The following algorithm performs restoring division.

Do the following *n* times:

1. Shift A and Q left one binary position.
2. Subtract M from A, and place the answer back in A.

3. If the sign of A is 1, set q_0 to 0 and add M back to A (that is, restore A); otherwise, set q_0 to 1.

22. Perform the restoring division for the given binary numbers 1000/11, show all the cycles

(08 marks) Jan 2015

Given A=10101 and B=00100 perform A/B using restoring division method

(08 marks) Jan 2013

Refer Class notes

23. Explain IEEE standard for floating point numbers (05 Marks) Jan 2014

We start with a general form and size for floating-point numbers in the decimal system and then relate this form to a comparable binary representation. A useful form is

$$\pm X_1, X_2, X_3, X_4, X_5, X_6, X_7 \times 10 \pm {}^{y1y2}$$

where X_i and F_i are decimal digits. Both the number of significant digits (7) and the exponent range (±99) are sufficient f or a wide range of scientific calculations h is possible to approximate this mantissa precision and scale factor range in a binary representation that occupies 32 bits, which is a standard computer word length. A 24-bit mantissa can approximately represent a 7-digit decimal number, and an 8-bit exponent to an implied base of 2 provides a scale factor with a reasonable range. One bit is needed for the sign of the number. Since the leading nonzero bit of a normalized binary mantissa must be a 1, it does not have to be included explicitly in the representation. Therefore, a total of 32 bits is needed.

This standard for representing floating-point numbers in 32 bits has been developed and specified in detail by the Institute of Electrical and Electronics Engineers (IEEE) [1]. The standard describes both the representation and the way in which the four basic arithmetic operations are to be performed. The 32-bit representation is given in Figure 20a. The sign of the number is given in the first bit,

followed by a representation tor the exponent (to the base 2) of the scale factor. Instead of the signed exponent, E, ±e value actually stored in the exponent field is an unsigned integer E' = E + 127

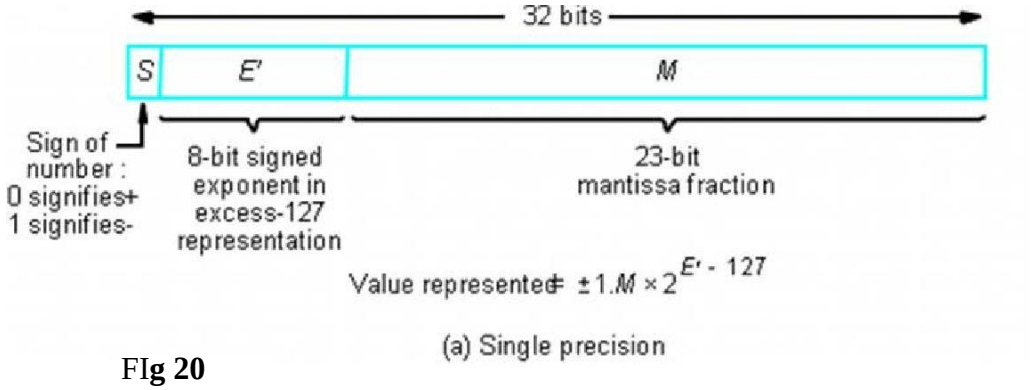
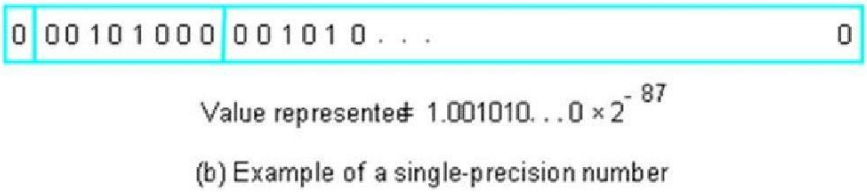
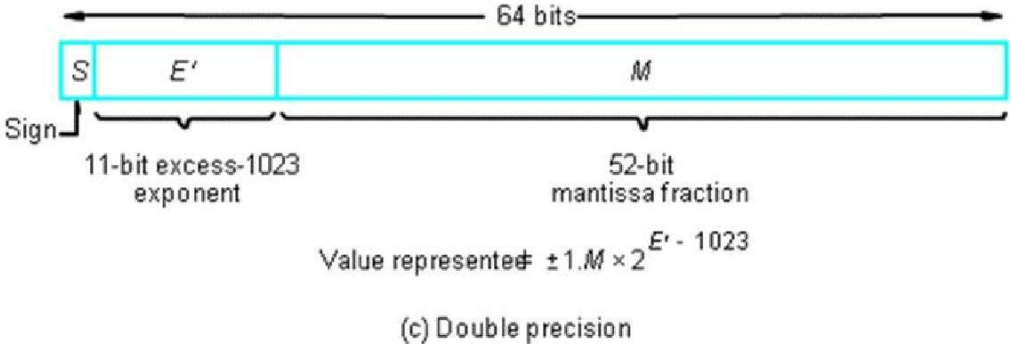


FIg **20**





This is called the excess-127 format. Thus, E' is in the range $0 \leq E' \leq 255$. The

end values of this range, 0 and 255, are used to represent special values, as described below. Therefore, the range of E' for normal values is $1 \leq E' \leq 254$. This means that the

actual exponent, E, is in the range $-126 \leq E \leq 127$. The excess-x representation for exponents enables efficient

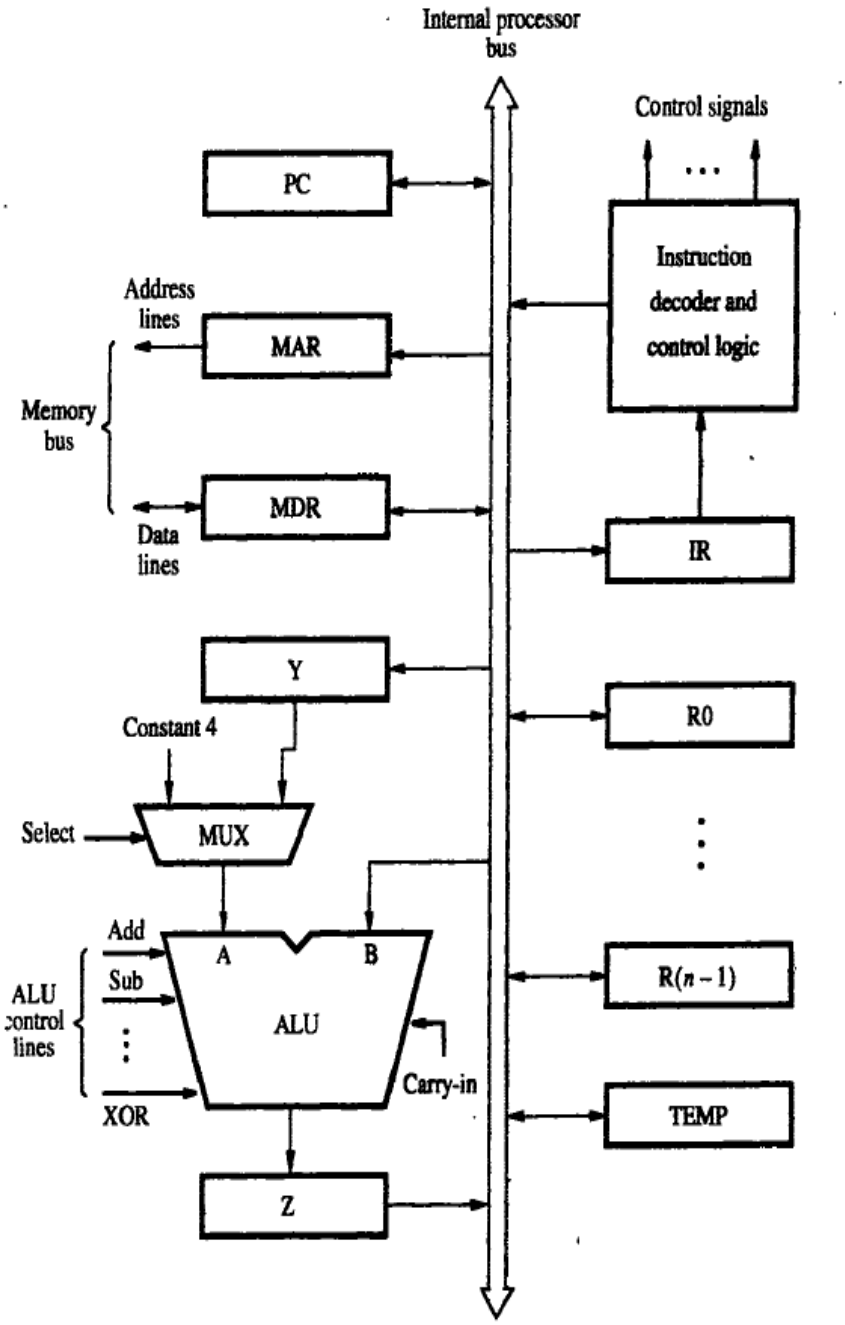
comparison of the relative sizes of two floating-point numbers.

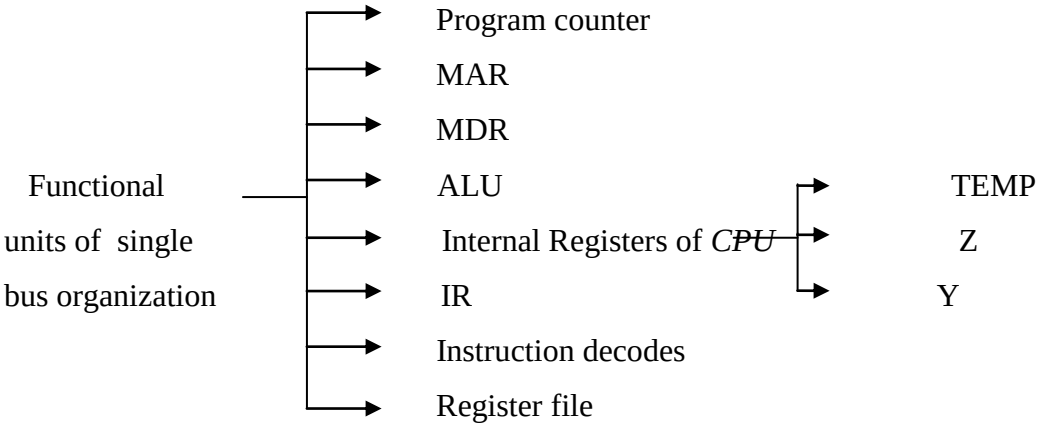
The last 23 bits represent the mantissa. Since binary normalization is used, the mo significant bit of the mantissa is always equal to 1. This bit is not explicitly represented: it is assumed to be to the immediate left of the binary point. Hence, the 23 bits stored in the M field actually represent the fractional part of the mantissa, that is, the bits stored the right of the binary point. An example of a single-precision floating-point number h shown in Figure 20

23. Draw and explain the single bus organization of the data path inside a processor

(10 Marks) Jan 2014

To study the internal operations of CPU the structure of single bus organization is as shown in the fig:


Internal processor bus
Control signals
PC
Instruction decoder and control logic
Address lines
MAR
Memory bus
MDR
Data lines
IR
Y
R0
Constant 4
Select
MUX
Add
Sub
ALU control lines
XOR
A
B
ALU
Carry-in
R(n - 1)
TEMP
Z



- **Program counter:**

❖ It is a special purpose register.
❖ Initially it is pointing to very first memory location of memory.
❖ The contents of PC are incremented by 4 after accessing the op-code of an instruction from memory into the instruction register.
"The program counter holds address of the next instruction to be executed."

- **MAR (Memory Address Register):**

❖ The one input of MAR is connected to internal bus and its output is connected to address lines of memory bus as shown in the fig.
❖ It is a uni-directional used to store the address of the memory location.

- **MDR (Memory Data Register):**

❖ It is a bi-directional and is used to store the contents of memory location.
❖ The content of internal bus and the content of data bus of memory bus can be transferred to MDR by means of bi-directional data lines as shown in the fig.
❖ The content of MDR can be transferred to internal bus or data bus of memory bus by means of bi-directional data lines.

- **ALU (Arithmetic and logic unit):**

❖ It is designed to perform arithmetic as well as logical operations.
❖ It consists of 2 inputs namely A & B as shown in fig.
❖ The 1st input is derived from the output of the multiplexer and 2nd input is derived from the internal bus.
❖ The result is stored in Z temporary register.

- **Internal Registers of CPU:**

❖ The CPU consists of 3 internal registers namely Y, Z and TEMP.
❖ These register are not accessible to programmer.
❖ These 3 registers are used by CPU in-order to hold the intermediate results during the execution of the program.

- **Instruction Register:**

❖ It holds the op-code of an instruction.

- **Instruction Decoder:**

❖ It decodes the instruction and generates control signal necessary for the execution of instruction.

- **Register File:**

❖ It is defined as set of registers used to store the operands temporarily during the execution of the program.

- **Functions of Instruction:**

  The instructions are designed to various types of operations:

❖ Data transfer between general purpose register:

MOV R_1, R_2

❖ ALU operations:

The instructions are designed to perform arithmetic as well as logical operations on operands.

Example:

ADD R_1, R_2, R_3 ; $[R_1] + [R_2] \rightarrow R_3$

This instruction is an example for Register Addressing. Where R1 and R2 are the source operands and R3 is the destination operand

This instruction adds the contents of R1 with the contents R2 and result is placed in R3 register.

XOR R_1, R_2

This instruction is an example for Register Addressing. Where R1 is the source operand and R2is the destination operand

This instruction performs EX_OR operation between the contents of R1 and R2 and result is placed in destination register R2.

❖ The instructions are designed to fetch the operands from memory into the processor register. Such type of operation is called as "Memory read operation."

Example:

MOV (R $_3$), R_2

This instruction transfers the data from memory into the processor register R_2 whose address is stored in R_3 register.

❖ The instruction is designed to transfer data from processor register to memory location of memory. Such type of operations is called as "Memory write operation."

Example:

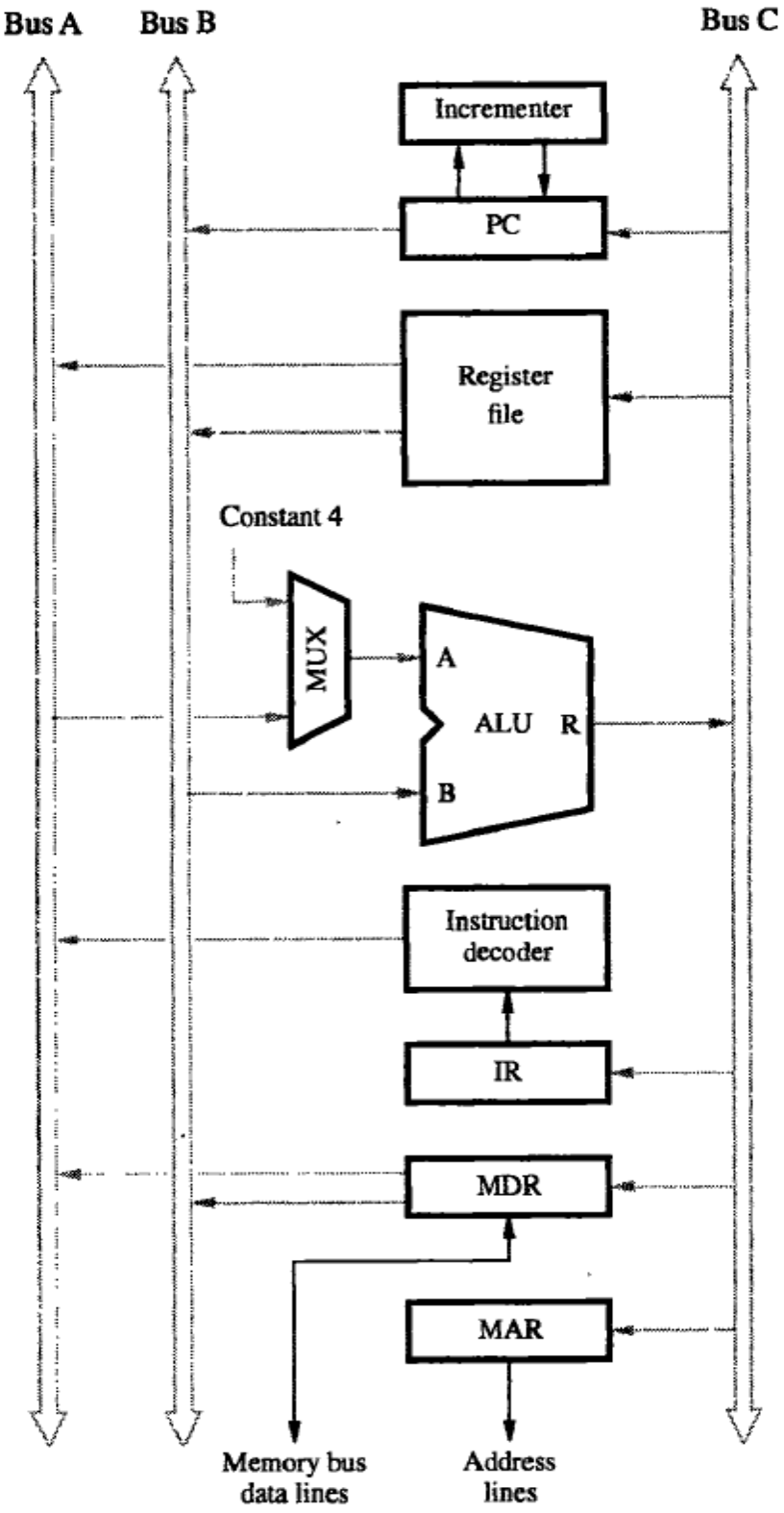
MOV R_3, (R_2)

24. Explain the three bus organization of the processor (08 marks) Jan 2015

Multi-Bus organization:

- The purpose of multi-bus organization is used to reduce the number of control steps required to execute one instruction.
- The structure of multi-bus organization is as shown in the fig.:
- It consists of 3 buses namely, A, B & C used to interconnect register file and ALU of the processor as shown in the figure.

- **Register File:**
- The CPU consists of set of registers and are combined into a single block and is called as register file.
- The register file consists of one input port and 2 output ports.
- The data available on C bus is transferred to one of the register of the register file by means of input port.
- The contents of registers of register file are transferred to either B bus or A bus by means of 2 output ports namely, output port – 1 & output port – 2.
- The increment-or of PC increments the content of PC by 4 and updated value is loaded into PC.


Bus A
Bus B
Bus C
Incrementer
PC
Register file
Constant 4
MUX
A
ALU
R
B
Instruction decoder
IR
MDR
MAR
Memory bus data lines
Address lines

- ALU:
- The ALU consists of 2 inputs namely A & B.
- The output of multiplexer is connected to A input of ALU from bus B and 2 inputs is connected from bus A as shown in the block diagram.
- The ALU performs the operation according to control signal and result is placed on C bus.
- The output of instruction register is connected to instruction decoding circuit to decode the instruction.
- The MDR register is connected in between C, B and A bus as shown in the fig.
- It consists of bidirectional data lines and are connected to data lines of memory bus.
- The MAR register holds the address of the memory or I/O which is connected to address lines of memory bus

Consider the following instruction:

ADD R_4, R_5, R_6 → Destination

↓ ↓

Src-2 Src-1

- This instruction is an example for register addressing. The mathematical representation of this instruction is:

  $[R_4] + [R_5] \rightarrow R_6$

  The sequence of control signals required to execute this instruction are as follows:

1) PC_{OUT}, R=B, MAR_{IN}, read INC PC
2) WMFC
3) MDR_{OUT}, R=B, IR_{IN}
4) $R4_{OUT}$-A, $R5_{OUT}$-B, select A, add, R_{6IN}, end

- During the step 1, the content of PC is transferred to B bus. At the same time the content of B bus are transferred to C bus by means of (R=B) control signal. The content of C bus (content of PC) is transferred to MAR and send a read request to memory

  The CPU waits for memory function complete (WMFC).

- During the step 3, the CPU fetches the op-code of an instruction into MDR by means of B bus. The content of MDR are transferred to instruction register by means of R=B control signal
-

These 3 steps are called as fetch phase.

- During the step 4, the ALU fetches the content of R_4 from A bus and content of R_5 from B bus and performs the addition operation between the contents of R_4 and R_5 and result is placed in R_6.

26. With the neat sketch, Explain the organization of microprogrammed control unit

(08 marks) Jul 2013

Consider the instruction

ADD (R3), (R1)

- This adds the content of location as directed by R3 with the contents of R1 and result is placed in R1.

The steps required to execute this instruction are as follows:

1. Fetch the instruction
2. Fetch the memory operand as directed by R3
3. Perform addition operation
4. Load the result into R1

The sequence of control steps required to execute this instruction are as follows:

1. PC_{out} , MAR_{in} , read ,$Select_4$,add , Z_{in}
2. Z_{out} , PC_{in} , Y_{in} , WMFC
3. MDR_{out} , IR_1
4. $R3_{out}$, MAR_{in} , read
5. $R1_{out}$, Y_{in} , WMFC
6. MDR_{out} , $Select_Y$,add , Z_{in}
7. Z_{out} , $R1_{in}$, end

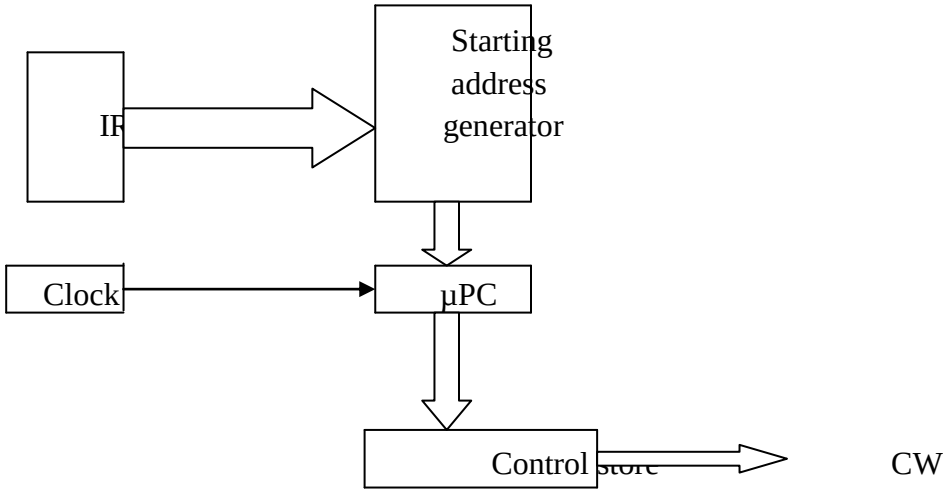
Micro - instruction	..	PC_{in}	PC_{out}	MAR_{in}	Read	MDR_{out}	IR_{in}	Y_{in}	Select	Add	Z_{in}	Z_{out}	$R1_{out}$	$R1_{in}$	$R3_{out}$	WMFC	End	;
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

1. **Control word:** It is group of binary bits , where each bit represents one control signal. From the above figure, it is clear that there are 7 CW's.

2. **Micro routine**: A set of CW's for a given machine instruction is called as micro routine.

3. **Micro instruction**: Each CW in a micro routine is called as micro instruction.

The basic organization of a micro programmed control unit is as shown in the figure:



The various blocks of micro programmed unit

- IR
- Starting address generator
- Micro program counter
- Control store

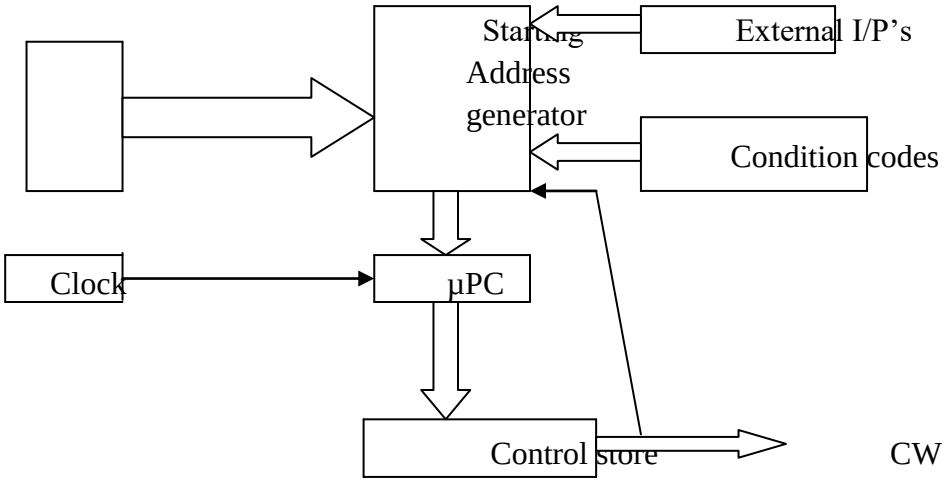
Control store:

- The micro routines of all instructions of a instruction set of a computer are stored in a special memory and is called as “control store” .
- The control unit generates control signals of any instruction by reading Cw’s of micro routines from the control store.
- The micro program counter is used to read the CW’s of micro routine sequentialiy.
- The starting address generator generates address when new instruction is loaded in IR.
- The content of MPC are incremented by clock after reading CW from the control store
-

The branch < 0 instruction can be implemented by a micro routine as shown in the figure:

Address	Microinstruction
0	PC_{out}, MAR_{in}, Read, Select4, Add, Z_{in}
1	Z_{out}, PC_{in}, Y_{in}, WMFC
2	MDR_{out}, IR_{in}
3	Branch to starting address of appropriate microroutine
.....	
25	If N=0, then branch to microinstruction 0
26	Offset-field-of-IR_{out}, SelectY, Add, Z_{in}
27	Z_{out}, PC_{in}, End

The organization of the control unit to allow conditional branch instruction is as shown in the figure:



- After loading the branch instruction into IR. Branch micro instruction transfers control to the corresponding micro routine.
- This address is the output of the starting address generator Clock.
- The micro instruction at location 25 tests the 'N' bit of the condition code.
- If this bit is equal to '0' control transfers to location '0' to fetch the new instruction. Otherwise , micro instruction at location 26 determines the branch target address and is loaded into Z register.

The micro instruction at location 27 transfers branch target address from Z register to PC

28. Describe the organization of Hardwired control unit (08 marks) Jan 2015

Consider the instruction

ADD (R3) , R1

- This instruction adds the contents of memory location as directed by R3 with the contents of register R1 and result is placed in R1 register.

  The sequence of steps required to execute this instruction are as follows:

  1. Fetch the instruction
  2. Fetch the memory operand
  3. Perform the addition
  4. Load the result into R1

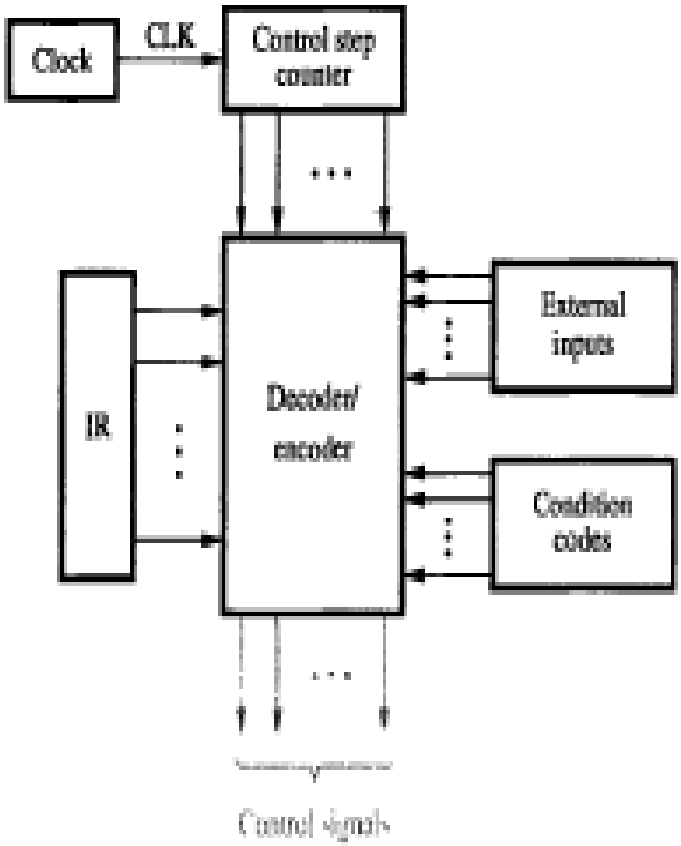
The sequence of control steps required to execute this instruction are as follows:

1. PC_{out} , MAR_{in} , read , $Select_4$, add , Z_{in}
2. Z_{out} , PC_{in} , Y_{in} , WMFC
3. MDR_{out} , IR_1
4. $R3_{out}$, MAR_{in} , read
5. $R1_{out}$, Y_{in} , WMFC
6. MDR_{out} , $Select_Y$, add , Z_{in}
7. Z_{out} , $R1_{in}$, end

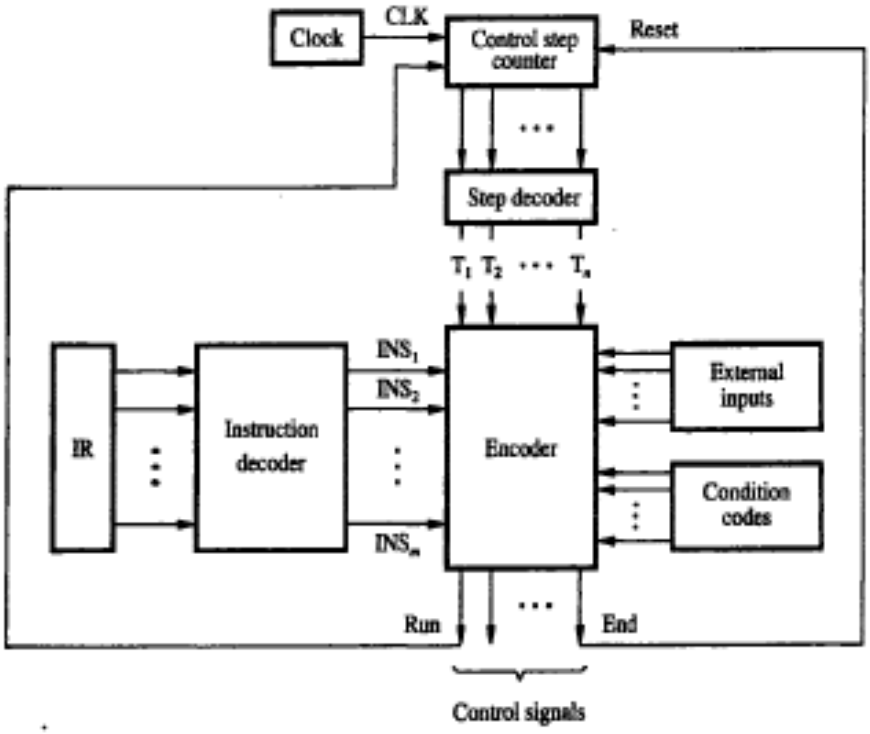
- The processor needs one clock period to complete one control step.
- A counter may be used to keep track of number of steps required one instruction.
- In this example a counter is initialized with 7 steps.
- The contents of counter are decremented by one after the completion of one step.
- The required control signals are determined by the following information:
    1. Contents of control step counter
    2. Contents of IR
    3. Contents of condition codes
    4. External input such as MFC and IR

    The organization of control unit is as shown in the figure:

The decoder/encoder block is a combinational circuit it generates the required control signals.



- **The block diagram that shows the separation of decoding and encoding functions as shown in the figure:**



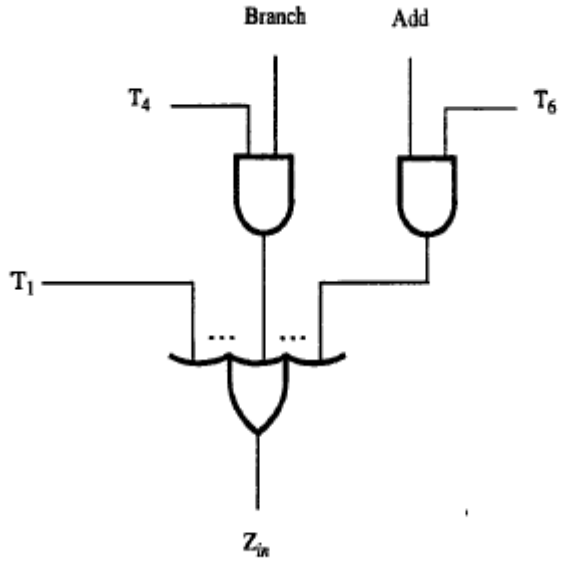
- The step decoder provides separate line for each control step and each control step is completed during one clock cycle as shown in the fig.
- Similarly the instruction decoder provides separate line for each machine instruction and are labeled as INS_1 , INS_2 ……INS_m. When instruction is loaded into IR , any one of these lines are set 1 and reset of the lines are set to '0' .
- The input signals to the encoder block are combined inorder to generate the control signals.

Ex: The generation of control signal Z_{in} by means of encoder implements the logic function

$$Z_{in} = T_1 + T_6 . ADD + T_4 . BR$$

- The Z_{in} signal is at high for all instructions during time slot T_1 , during T_6 for add instruction and during T_4 for branch instruction.

The logic diagram for implementing Z_{in} control signal is as shown in the fig:



27. Write the microroutine for the instruction ADD (R3), R1 (04 marks) Jan 2015

Consider the following instruction

ADD (R3), R2,

→ Destination (Register operand)

→ Source (Memory operand).

- This instruction is an example for register indirect addressing mode .
- According to this instruction it adds the content of memory location whose address is stored in R_3 register with the content of R_1 register and the result is stored in R_1.

The sequence of operation required to execute this instruction are as follow

1) Fetch the instruction (op-code) from memory.
2) Fetch the operand from the memory as directed by R_3.
3) Perform the operation according to op-code of an instruction.
4) Load the result into the R_1 register.

- **The sequence of control steps required to execute this instruction are as follows.**

a. P_{out}, MAR_{IN}, read select 4, add, Z_{IN}
b. Z_{OUT}, P_{CH}, Y_{IN}, WMFC
c. MDR_{OUT}, R_{IN}
d. R_{2OUT}, MAR_{IN}, read
e. R_{1OUT}, Y_{IN}, WMFC
f. MDR_{OUT}, select Y, add, Z_{IN}
g. Z_{OUT}, R_{1IN}, end

- During the step 1 the content of PC are transferred to MAR by activating PC-out & MAR-in, and send a read request to memory. The multiplexer selects constant 4 and is given to A input of ALU. This content of PC are gated to B input of ALU. The ALU adds the content of PC with the content of PC with constant 4 and updated value is stored in Zin register.
- During the step 2, the updated value of PC are transferred from Z register to PC. At the same time the content of PC are transferred to Y register.
- During the step 3, the CPU reads the op-code of an instruction from memory into the MDR. At the same time the content of MDR are transferred to IR.

These 3 steps are called as **fetch phase.**

- During the step 4, the content of R3 are transferred to MAR by activating two control signals namely R3out, MARin and send a real request to memory.
- During the step 5, transfer the content of R1 register to Y register. Therefore multiplexer selects Y input and is connected to B input of ALU.
- During the step 6, memory operand is read into MDR and is gated to B input of ALU. The multiplexer selects Y. The ALU performs the addition operation between register operand and memory operand and result is placed in Zin register.
- During step 7, the result is transferred from register to R1 register.

These 4 steps are called as **execute phase.**