Q.1(a) Explain Lock(bar) and Test(bar) signal. 05

Ans.: **i) Lock prefix :** It is a prefix to 8086 instruction. In closely coupled configuration, other processors request the use of system bus with the help of $\overline{RQ}/\overline{GT}$ signal. If lock prefix is used then 8086 releases system bus at the end of instruction cycle otherwise at the end of current bus cycle.

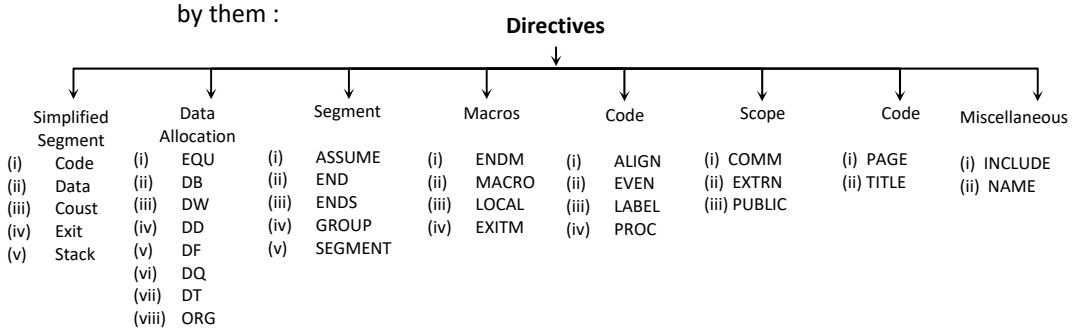
ii) Lock signal : In closely coupled configuration, $\overline{LOCK}$ signal of 8086 is given to $\overline{LOCK}$ of 8289. Whenever 8086 executes an instruction with Lock prefix then $\overline{LOCK}$ signal goes low. Now 8289 will not release the system bus at the end of current bus cycle even if there is request at $\overline{CBRQ}$. Once that instruction is completed, then $\overline{LOCK}$ signal goes high and so, 8289 can release the system bus now.

Q.1(b) Explain assembler directives. 05

Ans.: Assembly Language consist of 2 types of statements :

(1) Executable statements (2) Assemble directives

- Assembler directives are the statements that direct the assembler to do something. It consist of the entire instruction set. As the name says it direct the assembler to do a talk.
- The speciality of these statements that direct and they are effective only during the assembly of a program but they do not generate any code that is machine executable.
- The assembler directives are categorized namely the general purpose directives and the special directives.
- They are classified into the following categories based on the functions performed by them :



Directives (Any 5)

1) Code :

- This assembles directives indicates the beginning of the code segment.
- Its format is as follows :

  CODE [name]
- The name in this format is optional.

2) Data :

- This directive indicates the beginning of the data segment.

3) STACK :

- This directive is used for defining the stack.
- Its format is as follows :
  - STACK [size]
- The size of stack is 1024 bytes by default but this size can be overridden.

4) EQU (EQUATE) :

- It is used to give a name to some value or symbol in the program.
- Its format is :

  [name] EQU initial value.

  e.g.: FACTORIAL EQU 05H.

5) Define Byte [DB] :

- This directive defines the byte types variable.
- The format is as follows :

  [name] DB initial value.
- The initial value can be a numerical value or more than one 8 bit numeric values.

6) Define Word [DW] :

- This directive defines items that are 16 bit in length
- Its format is :

  [name] DW initial value.

  e.g. List DW 2534.

7) Define double word [DD] :

- It defines the data items that are double word in length.
- The format is :

  [name] DD initial value.

  e.g.: Buff DD

8) Define Quad Word [DQ] :

- This directive is used to tell the assembler to declare variables & words in length or to receive 4 words of storage in memory.
- Its formal is :

  [name] DQ initial value, [initial value]

  e.g. where DQ 1 2 3 4 5 6 7 8 9 8 7 6 5 4 3 2 H

9) Define Ten Bytes [DT]

- It is used to define data that are 10 bytes long.
- Its format is :

  [name] DT initial value, [initial value]

  e.g. : userpack DT 1 2 3 4 5 6 7 8 9 0

10) ORG [ORIGINATE]

- It allows the location counter to any desired value at any point in the program.
- Its format is :

  ORG Expression.

  e.g. : ORG 500H; Set the location counter to 500 H.

11) Assume

- It is used for telling the assemble the name of the logical segment which should be used.
- Its format is :

  Assume segment register : Segment-name :

  E.g. : Assume CS code, DS : Data, SS : stack.

12) END

- It is used to neglect any statement after an END directive.
- Its format is :

  END

13) Segment and ENDS

- This segment directive is used to indicate the start of a logical statement.
- Its format is :

  name SEGMENT; Begin segment

  name ENDS; End segment

  e.g. DATA SEGMENT

  DATA ENDS

14) Group

- This directive collects the segments of the same type under one name.
- Its format is

  [name] Group seg-name; [Seg-name]

  e.g.: NAME Group SEG 1, SEG 2.

15) MACRO and ENDM

- The macros in the program can be defined by MACRO directive.
- The ENDM directive is used along with macro directive.
- ENDM defines end of Macro.
- Its format is :

  DISP MACRO

  ; Statement inside the Macro

  ENDM

16) ALIGN :

- It will tell the assembler to align the next inst. on an address which corresponds to the given value.
- Its format is :

  ALIGN Number

  This number should 2, 4, 8, 16, …… power of 2.

  e.g. ALIGN 2.

17) EVEN :

- It tells the assembler to increment counter if required, so that the next defined data item is aligned on an even storage boundary.
  e.g. : Even table DB 10 DUP(0); It declares all array named TABLE of 10 bytes which are starting from an even addr.

18) LABEL :

- This directive assigns name to the current value of the location counter.
- e.g.: Stack Segment
  DW 50 DUP (0); set aside 50 words for back. STACK-TOP LABEL WORD; Give a symbolic name to next location after the last word in stack STACK ENDS.

19) EXTRN

- It indicates that the name or labels that follow the EXTRN directive one in some other assembly module.
- Its format is
  e.g.: Procedure – Here Segment
  EXTRN FACT : FAR
  XTRN SUM : NEAR
  Procedure – Here Ends.

20) PROC - Procedure

- It is used to indicate the start of a procedure
- Its format is

[procedure -name]
e.g.: PROC NEAR

21) PUBLIC

- It informs the assemble and linker that the identified variables in a program are to be referred by other modules linked with current one
- Its format is :
  PUBLIC variable, [variable]
  e.g. PUBLIC MULTIPLIER DIVISOR

22) PAGE

- This directive is used to specify the maximum number of lives on a page and the maximum number of character on a line.
- Its format is :

PAGE [length]. [width]
→ No. of characters on a time
→ No. of lines on a page

e.g. PAGE 55, 102.

23) TITLE

- It is used to give a title to program and print the title on the second line of each page of the program.
- Its format is :
  TITLE text

24) INCLUDE :

- This directive is used to tell the assembler to insert a block of source code from the named file into current source module. This shortens source code.
- Its formal is:

  INCLUDE path : filename

25) NAME :

- This directive assign a specific name to each assembly module when programs consisting of several modules are written.

26) DUP OPERATOR :

- With DUP, we can repeat one or more values while assigning the storage values.
- Its format is :

  [name] Data – type Number DUP (value)

  e.g. List DB 20 DUP [0]; A list of 20 bytes while each byte is zero.

27) GLOBAL – Declare symbols as PUBLIC or EXTRN.

- This directive can be used instead of PUBLIC or EXTRN directive.
- Its format is :

  e.g. Global factor; it makes the variable factor public so that it can be accessed from all other modules that are linked.

Global Factor : Word ; It tells assembles that variable factor of type word which is in another assembly module or EXTRN.

28) LENGTH

- It informs the assembler to find the number of elements in a named data item like a string or are array.
- Its format is :

  e.g. MOV CX, LENGTH STRING; Loads the length of string in CX.

29) OFFSET

- It informs the assembler to determine the offset or displacement of a named data 'item'.
- Its format is :

e.g. MOV AX, OFFSET NUM; It will load the offset of NUM in the AX register.

30) ENDP – End procedure :

- This directive is used along with the name of the procedure to indicate the end of procedure.
- e.g.: Fact PROC FAR; Start procedure named FACT and informs assemble that procedure is FAR

  FACT END P; End of procedure FACT.

31) PTR – Pointer

- It is used to assign a specific type to a variable or to a label.
- e.g.: INC BYTE PTR[BX]; it increments byte pointed to by [BX].

32) SHORT

- It is used to tell the assembler that only 1 – byte displacement is needed to code a jump instruction.
- Its format is :

e.g. JMP SHORT NEAR – LABEL

33) TYPE

- It informs the assembler to find the type of a specified variable.
- It actually finds the number of bytes in the type of variable.
- For a byte type it gives a value 1 and for word type it gives a value 2 and for double word it gives a value 4.

Q.1(c) Explain difference between RISC and CISC. **05**

Ans.: RISC versus CISC characteristics :

Architectural characteristics	CISC	RISC
Instruction set size and instruction formats	Large set of instructions with variable formats (16-64 bits per instruction)	Small set of instructions with fixed (32bit) format and most register based instructions.
Addressing modes	12 – 24	Limited to 3 – 5
GPRs and cache design	8–24 GPRs, mostly with a unified cache for instructions and data, recent designs also use split caches	Large numbers (32–192) of GPRs with mostly split data cache and instruction cache.
Clock rate and CPI	33–50MHz in 1992 with a CPI between 2 and 15	50 – 150MHz in 1993 with one cycle for almost all instructions and on average CPI < 1.5
CPU control	Most microcoded using control memory (ROM), but modern CISC also uses hardwired control.	Most hardwired without control memory.

Q.1(d) Explain nano programming. **05**

Ans.: Computer programming in Nano is one of the newest developments. It was believed that a Nano mechanical computer could run a million times faster than a microprocessor based computer. This is because that one out of the million components of a computer is made of mechanical space.

The Nano computer language is believed to work well with the present day computers systems. The primary use of this programming language is on graphics. With the Nano-X graphics system you could create much fancier graphical programs. To make it work, you have to specifically create the program with the Window, Unix or Macintosh interface in mind.

The Nano computer language primarily came from the nano-technology. Nano technology refers from the fields of applied science that control matter on its molecular and atomic scale.

This program is easy to learn and apply. Texts can be typed immediately into the interface. It is also simple to insert text into the program with the use of some editing configuration. There is also nano editor software that you can use with the main program base so that saving, cutting, pasting and searching becomes fairly straight forward.

Q.2(a) Explain the various addressing modes of 8086 with example. 10

Ans.: Addressing Modes :

1) Register Addressing Mode : In this mode operands are specified by the internal registers of microprocessor. The length of these instructions varies from 2 to 3 bytes. The address for the data memory will not be changed by the execution of these instructions.

e.g. MOV AX, BX
MOV CL, DL

We can transfer from any 8 bit register to 8 bit register or 16 bit register to 16 bit register. We cannot move the contents of 16 bit register to 8 bit register or CS register.

2) Immediate Addressing Mode : In this mode operand is specified by the instruction itself. The instruction length varies from 3 bytes to 6 bytes,

e.g. MOV DL, 80H
MOV BX, 1234H

3) Direct Addressing Mode : In this mode, the address of the operand byte is directly specified in the instruction. The length of these instruction is 4 to 5 bytes.

e.g. MOV AX, LIST:

Transfers a word from the memory location pointed by LIST in the data segment to AX. The effective address is calculated in this case as

$DS \times (10)_{16} + LIST$

MOV[1122H], CL:

Transfers a byte from CL to memory location 1122H in the data segment. The effective address is calculated as

$DS \times (10)_{16} + 1122H.$

4) Register Indirect : In this mode, the base register provides the offset address of the operand indirectly.

e.g. MOV CX, [BX]

Transfers a word from the memory address pointed by BX from data segment to CX. Address is calculated as

$DS \times (10)_{16} + BX$

MOV [BP], DL:

Transfers a byte from DL to the stack memory pointed by BP. Address is calculated as

$SS \times (10)_{16} + BP.$

5) Base Relative Address Mode : In this mode, the operand bytes are addressed by any one of the base registers 16 bit or 8 bit displacement,

e.g. (a) MOV AX, [BX + 4] : Transfers a word from memory location pointed by [BX + 4] in the data memory to AX.

Address : $DS \times (10)_{16} + [BX + 4]$

(b) MOV [BP + 1222H], CL: Transfers a byte from CL, to the stack memory location pointed by BP + 1222H.
Address : $SS \times (10)_{16}$ + [BP +1222H]

6) Base Plus Index : In this mode, the operand byte is addressed by base register plus index register,
e.g. MOV DX, [BX + SI]
Transfers word from the data segment pointed by [BX + SI] to DX.
Address: $DS \times (10)_{16}$ + [BX + SI]
MOV [BP + DI], BX
Transfers a word from BX to stack segment pointed by [Bx + DI].
Address: $SS \times (10)_{16}$ + BP + DI

7) Base Relative Plus Index : In this mode, the operand bytes are addressed by **Base register + Index register + 8 bit or 16 bit displacements.** These type of addressing mode is used to address the complex array elements,

e.g. (a) MOV CX, [BX + DI + 20H].
Address: $DS \times (10)_{16}$ + BX + DI + 20H.
(b) MOV [BP + SI + 2222H], AL
Address: $SS \times (10)_{16}$ + BP + SI + 2222H.

8) Addressing modes for I/O ports : There are 2 types:

a) DIRECT : In this, 8 bit port address is specified in the instruction itself, so we can have at the most 256 I/O ports,
e.g. IN AL, 04H $\Rightarrow$ 8 bit data from input devices address 04H into AL.
IN AX, 04H $\Rightarrow$ 16 bit data from input device address 04H and 05H into AL & AH respectively

b) INDIRECT : In this, 16 bit I/O port address is kept in DX register, so we can have at the most 65536 I/Oports,
e.g. IN AL, DX $\Rightarrow$ move 8 bit data from input device pointed by DX into AL.
IN AX, DX $\Rightarrow$ move 16 bit data from input device pointed by DX & DX + 1 into AX.

9) Implied addressing mode : In this, instruction has no operand,
e.g. STC – set carry flag.
CLD – clear direction flag.
STD – set direction flag.

10) String addressing mode : A string indicates a series of bytes or words stored in a sequential memory locations i.e. they indicate a block of data.
The instructions which can operate on block of data are known as String Instructions. For these instructions, memory source is DS:[SI] & memory destination is ES:[DI]. After performing string operation SI or DI are automatically incremented (or decremented) by 1 or 2 depending on byte operation or word operation. If the direction flag is reset then automatic incrementing & vice-versa.

Q.2(b) Explain IEEE format for floating point representation **10**

Ans.:

- Floating point representation is redundant in the sense that the same number can be represented in more than one way.
  For example : 1.0×10^{18}, 0.1×10^{19}, 1000000×10^{12} and 0.000001×10^{24} are possible representation of a quintillion.
- It is generally desirable to have a unique or normal form for each representable number in a floating–point system. Consider the common case where the mantissa is a sign–magnitude fraction and a base of 'r' is used.
- The mantissa is said to be normalized if the digit to the right of the radix point is not zero, that is no leading zeroes appears in the magnitude part of the number. For example, 0.1×10^{19} is the unique normal form of a quintillion using base 10, a decimal mantissa and a decimal exponential binary fraction in 2's complement code is normalized when the sign bit differs from one bit to its right. Normalization restricts the magnitude |M| of a fractional binary mantissa to the range $1/2 \leq |M| < 1$.
- Normal forms can be defined similarly for other floating–point codes. An un-normalized number is normalized by shifting the mantissa to the right or left and appropriately incrementing or decrementing the exponent to compensate for the mantissa shift.
- The representation of zero posses some special problem. The mantissa must of course, be zero but the exponent can have any value, since $0 \times B^E = 0$ for all the values of E. Round off error result in a mantissa that is nearly, but not exactly zero. For the entire floating point number to be close to zero, its exponent must be a very large negative number – K. This requirement suggests that the exponent used for representing zero should be the negative number with the largest magnitude that can be contained in the exponent field of the number format. If K–bits are allowed for the exponent including its sign, then 2^K exponent bit pattern are available to represent signed integers which can range either from -2^{K-1} to $2^{K-1} - 1$ or from $-2^{K-1} + 1$ to 2^{K-1}, so that, K is 2^{K-1} or $2^{K-1} - 1$.
- The floating point exponent field E contains an integers that is the desired exponent value plus K.
- The quantity K is called the bias and an exponent encoded in this way, is called a biased exponent.

Q.3(a) Explain Booth's Algorithm. **10**

Ans.: Booth's algorithm is depicted as in following figure and can be described as follows :

1) Multiplier and Multiplicand are placed in Q and M registers respectively.
2) There is also a 1 bit register placed logically to the right of the least significant bit (Q_0) of the Q register and designed Q_{-1}.
3) The result of the multiplication will appear in the A and Q register. A and Q_{-1} are initialized to zero.
4) 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.
5) If the two bits are the same (1 – 1 or 0 – 0), then all of the bits of the A, Q and Q_{-1} registers are shifted to the right 1 bit. If two bits differ, then the multiplicand is added to or subtracted from the A register, according as the two bits are 0–1 or 1–0.

6) Following the addition or subtraction, the right shift occurs. In either case, the right shift is such that the leftmost bit of A, namely A_{n-1}, not only shifted into A_{n-2}, but also remains in A_{n-1}. This is required to preserve the sign of the number of A and Q. It is known as an arithmetic shift, since it preserves the sign bit.

e.g. M = 0111 Q = 0011

A	Q	Q_{-1}			
0000	0011	0	Initial		
1001	0011	0	$A \leftarrow A - 1$	}	
1100	1001	1	shift	}	First cycle
1110	0100	1	shift	}	Second cycle
0101	0100	1	$A \leftarrow A + M$	}	
0010	1010	0	shift	}	Third cycle
0001	0101	0	shift	}	Fourth cycle (product in A, Q)

Result = $(00010101)_2 = (143)_{10}$

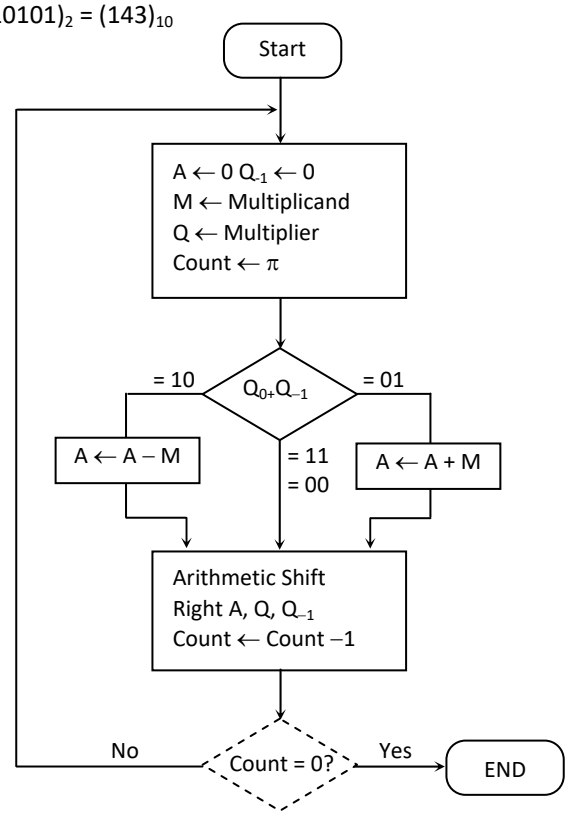


Fig : Booth's algorithm for two's complement multiplication

Algorithm :

Case I : If $Q_i = 0$, $Q_{i-1} = 1$
then add M to partial product

Case II : If $Q_i = 1, Q_{i-1} = 0$
then subtract M from partial product

Case III : If $Q_i = 0, Q_{i-1} = 0$ OR $Q_i = 1, Q_{i-1} = 1$
then, Rshift

Declare : A(7 : 0) Q(7 : 0), M(7 : 0), Count (3 : 0)
Bus : Inbus (7 : 0) Outbus (7 : 0)
Begin : A ← 0 Inbus ← Q(7 : 0)
Count ← n (No. of bits of number)
Input : M ← Inbus, Q ← Inbus
Loop : if (Q_0, Q_{-1}) = (00 V11)
then goto Rshift
if (Q_0, Q_{i-1}) = 01
then, A = A – M or
A (7 : 0) = A(7 : 0) + M(0 : 7)
if (Q_0 , Q_{i-1}) 10
then,
A (7 : 0) = A(7 : 0) – M(7 : 0)
Rshift : A(7 : 1) Q ≤ A . Q (6 : 0)
Test : Count = Count – 1
if Count = 0
then end
else goto loop P
End : stop

Q.3(b) Explain characteristics of memory in detail? 10

Ans.: **Memory Characteristics :**The properties to be considered when evaluating any memory technology are:

i) Cost : The price should include the cost of information storage cells as well as the cost of the peripheral equipment or access circuitry essential to the operation of memory.

cost = price of complete memory system / total bits of storage capacity.

ii) Access time : It is the time required to read or write a fixed amount of information. e.g. one word from the memory. Access time depends upon the physical characteristics of the storage medium and also on the types of access mechanism used. It is usually calculated from the time a read request is received by the memory unit to the time a read request is made available to the memory output terminals. The access time measured in words per second is another widely used performance measure for storage devices. Thus, low cost and high access rates are desirable memory characteristics.

iii) Access modes : It is the order or sequence in which information can be accessed. Memory can be accessed randomly or sequentially. In random access memories each storage location can be accessed independently of the other locations whereas in serial access memory storage locations can be accessed only in a certain predetermined sequence.

iv) **Alterability :** Memories whose contents cannot be altered on line are called Read Only Memories (ROMs) Memories in which reading or writing can be done online are called read write memories. All memories used for temporary purpose are read write memories.

v) **Permanent of storage :** The physical processes involved in storage are sometimes inherently unstable, so that stored information may be lost over a period of time unless appropriate action is taken. There are three important memory characteristics that can destroy information : destructive readout, dynamic storage and volatility. In destructive readout, the memory contents are called as the memory is read. Memories which require periodic refreshing are caused as dynamic memories. Static memories do not require refreshing. If the contents of memory are lost in case of power failure, the memory is termed as volatile memory.

vi) **Cycle time and data transfer rate :** The minimum time that must elapse between the initiation of two different access by memory can be greater than access time, this loosely defined term is called cycle time of the memory. It is generally convenient to assume that cycle time is the time needed to complete any read or write operation in memory.
The maximum amount of data that can be transferred is 1/tm and is called data transfer rate. The access time may be more important in measuring overall computer system performance since it determines the length of time of processor must wait unit initiating a next memory request.

vii) **Physical characteristics :** Many different physical properties of matter are used for information storage. The most important properties used for this purpose, all classified as electronic, magnetic, mechanical and optical. A factor determining the physical size of a memory unit is the storage density measured in bits per unit area. In general, memories with no moving parts have much higher reliability than memories such as magnetic disks which involves considerable mechanical motion.

Q.4(a) What is virtual memory? Explain in detail paging, segmentation and paged segments. 10

Ans.: Main and the secondary memory form another two level hierarchy. This interaction is managed by operating system. However, so it is not transparent to system software but somewhat transparent to the user code. The term **'virtual memory'** is applied when main and secondary memories appears in user program like a single, large and directly addressing memory.

Three reasons for using virtual memory.

- To free user from the need to carry out storage reallocation and permit the efficient sharing of available memory space by different users.
- To make the program independent of the configuration and capacity of the physical memory for execution.
- To achieve the very low cost per bit and low access time that are possible with memory hierarchy.

The program is divided into number of blocks of virtual memory which is known as 'virtual address space'.

A page is a fixed length block which can be assigned to fixed regions of physical memory called as **'page frames'**. Division of physical memory space into equal size block is called **'page frame'**. Size of page frames is equal to size of page.

Dividing the virtual address space into equal size of blocks is known as **'Paging'.**

In pure paging system, each virtual address consists of two parts : a page address (no) and a displacement.

∴ Translation of physical space is

< p . n >
↙ ↘
page address displacement

The memory map, now referred to as a **'page table'** consists of the following information.

Page add	Page frame	Presence bit p	change bit C	Access
A	000000	1	0	R_1X
C	06C7F9	0	1	R_1W_1X

Each virtual page address has corresponding real address of page frame in main or secondary memory. When presence bit p = 1, required page is present in main memory and base address of page frame is stored in page table.

If p = 0, a page fault occurs. The change bit C indicates whether or not the page has been changed since it was last loaded into main memory.
Page table can also contain memory protection data that specifies access rights of current program to read from, write into or execute page.
Since, page frames are contiguous, no external fragmentation access in paging. But, if K–word block is divided into P, n–word pages and k is not multiple of n, the page frame to which the block is assigned will not be filled. Unusable space within the partially filled page frame is 'internal fragmentation'.

Advantages :

1) The chief advantage of paging is that data transfer between memory levels is simplified : an incoming page can be assigned to any available page frame.
2) No external fragmentation problem, as page frames are contiguous.
3) Paging is hidden from the user.
4) This can be used in multiprogramming.

Disadvantages :

1) Protection facility is not available in paging.
2) Internal fragmentation is problem in paging.

Q.4(b) Explain the different mapping techniques of cache memory **10**

Ans.: Various Mapping Methods :

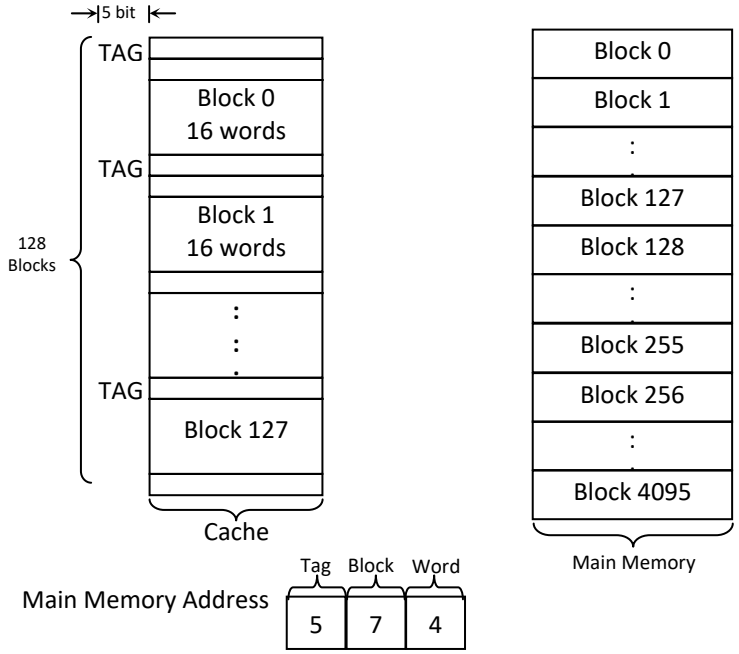
There are possibly three mapping methods to specify where main memory are placed in cache :

(a) Direct mapping method
(b) Associative mapping method
(c) Block-set associative mapping method

To discuss mapping methods, consider a cache consisting of 128 blocks of 16 words each for a total of 2048 (2K) words and assume that the main memory is addressable by a 16–bit address for mapping purpose, the main memory will be viewed as composed of 4k blocks.

(a) Direct Mapping

- This is the simplest way to associate main memory (MM) works with cache blocks. In this technique, block K of main memory maps onto block **K modulo 128** of the cache. Here more than one main memory block is mapped onto a given cache block position.



- As main memory address can be divided into three fields as shown in figure below. When a new block enters the cache, the 7–bit cache block field determines the cache position in which this block must be stored the high order five bits of the main memory address of the block are stored in five tag bits associated with its location in the cache.
- As execution process's the 7–bit cache block field of each address generated by the CPU points to a particular block location in the cache.
- The tag field of that block is compared to the tag field or the address 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. The direct mapping method is easy to implement, but it is not very flexible.

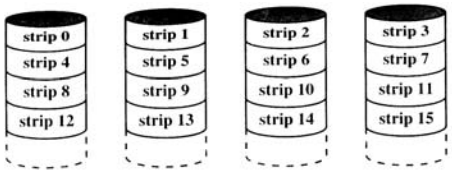
(b) Associative Mapping Method

- The associative mapping technique is much more flexible method in which a main memory block can potentially reside in any cache block position.

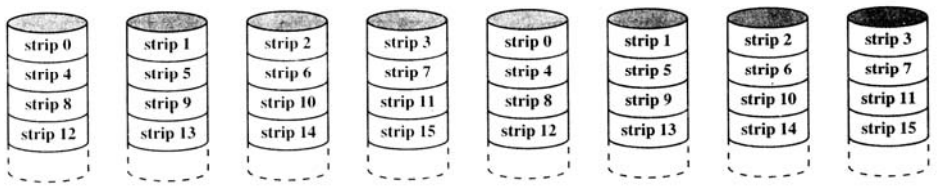
- In this case, 12 tag bits are required to identify a MM block when it is resident in the cache. The tag hits of an address received from the CPU are compared to the tag bits of each block of the cache to see if the desired block is present.
- It's cost of implementation is higher than the cost of the direct mapping scheme because of the need of search all 128 tag patterns to determine whether a given block is in the cache. A search of this kind is called an associative search.

Q.5(a) Explain RAID Levels. **10**

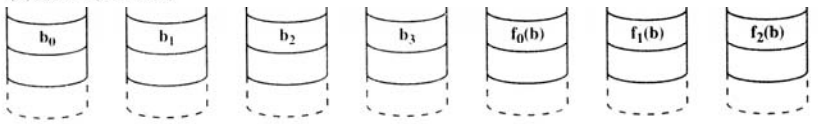
Ans.: Raid : RAID levels

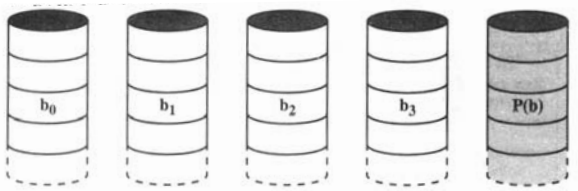


(a) RAID 0 (Nonredundant)

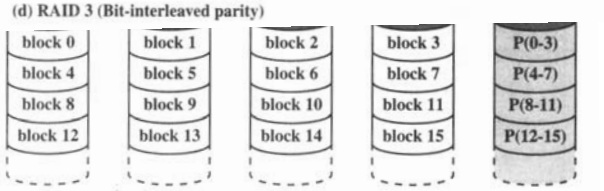


(b) RAID 1 (Mirrored)

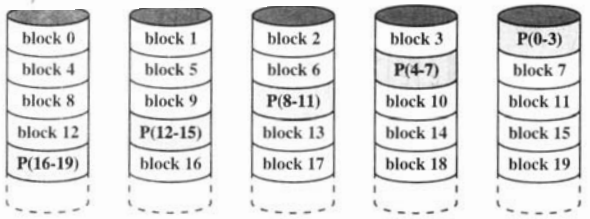




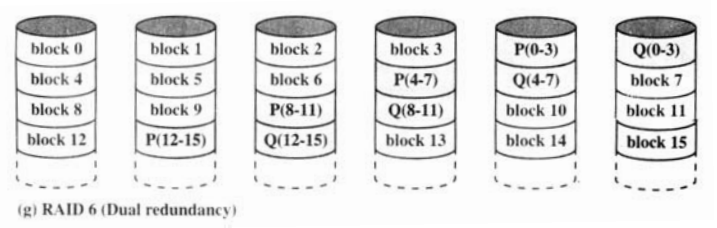
(d) RAID 3 (Bit-interleaved parity)



(e) RAID 4 (Block-level parity)



(f) RAID 5 (Block-level distributed parity)



(g) RAID 6 (Dual redundancy)

These are schemes to provide redundancy at lower cost by using disk stripping combined with parity bits. Different RAID organizations, or RAID levels, have differing cost, performance and reliability characteristics.

RAID level 0 :

- Stripping at the level of blocks ; non − redundant.
- Used in high − performance applications where data loss is not critical.

RAID level 1 :

- Mirrored disks, offers best write performance
- Popular for applications such as storing log files in a database system.

RAID level 2 :

- Memory style Error correcting codes (ECC) with bit stripping.

RAID level 3 :

- Bit interleaved parity; a single parity bit can be used for error corrections not just detection.
- When writing data, parity bit must also be computed and written.
- Faster data transfer than with a single disk, but fewer I/Os per second since every disk has to participate in every I/O.
- Subsumes level 2 (provides all its benefits, at lower cost).

RAID level 4 :

- Block interleaved parity; uses block level striping, and keeps a parity block on a separate disk for corresponding blocks from N other disks.
- Provides higher I/O rates for independent block read than level 3 (block read goes to a single disk, so blocks stored on different disks can be read in parallel.
- Provides high transfer rates for reads of multiple blocks.
- However, parity block becomes a bottleneck for independent block writes since every block write also writes to parity disk.

RAID level 5 :

- Block interleaved distributed parity; partitions data and parity among all N+1 disks, rather than storing data in N disks and parity in 1 disk.
- For eg., with 5 disks, parity block for nth set of blocks is stored on disk (n mod 5) + 1, with data blocks stored on the other 4 disks.
- Higher I/O rates than level 4 (Block writes occur in parallel if the blocks and their parity blocks are on different disks.)
- Subsumes level 4

RAID level 6 :

- P + Q redundancy scheme; similar to level 5, but stores extra redundant information to guard against multiple disk failures.
- Better reliability than level 5 at a higher cost; not used widely.

Table : RAID Levels

Category	Level	Description	Disks required	Data availability	Large I/O data transfer capacity	Small I/O request rate
Striping	0	Nonredundant	N	Lower than single disk.	Very high.	Very high for both read and write.
Mirroring	1	Mirrored	2N, 3N, etc.	Higher than RAID 2, 3, 4 or 5; lower than RAID 6.	Higher than single disk for read similar to single disk for write.	Upto twice that of a single disk for read similar to single disk for write.
Parallel access	2	Redundant via Hamming code	N + m	Much higher than single disk; higher than RAID 3, 4 or 5.	Higher of all listed alternatives.	Approximately twice that of a single disk.
	3	Bit-interleaved parity	N + 1	Much higher than single disk; comparable to RAID 2, 4 or 5.	Highest of all listed alternatives.	Approximately twice that of a single disk.
Independent access	4	Block-interleaved distributed parity	N + 3	Much higher than single disk; comparable to RAID 2, 3 or 5.	Similar to RAID 0 for read; significantly lower than single disk for write.	Similar to RAID 0 for read; generally lower than single disk for write.
	5	Block-interleaved distributed parity	N + 1	Much higher than single disk; comparable to RADID 2, 3 or 4.	Similar to RAID 0 for read; lower than single disk for write.	Similar to RAID 0 for read; generally lower than single disk for write.
	6	Block-interleaved dual distributed parity	N + 2	Highest of all listed alternatives.	Similar to RAID 0 for read; lower than RAID 5 for write.	Similar to RAID 0 for read; significantly lower than RAID 5 for write.

Q.5(b) Explain DMA and different modes of DMA **10**

Ans.: Direct Memory Access

The hardware required to design Direct Memory Access is as shown below :

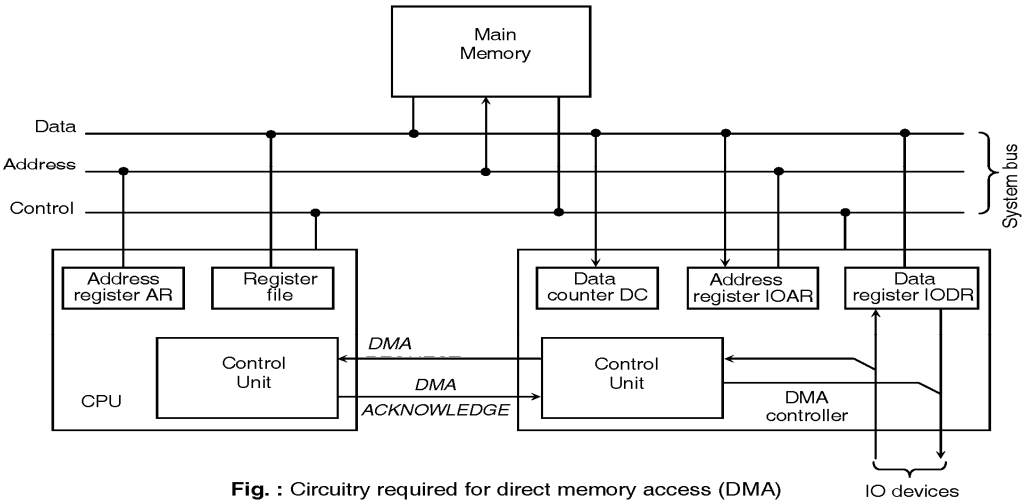


Fig. : Circuitry required for direct memory access (DMA)

- In this structure of DMA, the both DPU & DMA controller have access to main memory via a shared system bus having data, address and control lines.
- The I/O devices are connected to system bus for transferring of data via a special interface circuit shown as **"DMA controller".**
- It contains the data register (IODR) then address register (IOAR) and data counter register (DC) which enables the DMA controller to transfer data to or form the different regions of main memory. The IOAR register of DMA contains the base address of the memory region where transfer operation is to be done. This register is automatically decremented or incremented after each word is transferred.
- The data counters (DC) contain the number of words remain to be transferred. This counter value is automatically decremented after each transfer of word and tested for zero. When his DC reaches to zero, the DMA controller stop the DMA transfer.
- It normally provide with the interrupt capability. It can send interrupt to CPU to indicate end of data transfer.
- DMA transfer can be done by two ways.

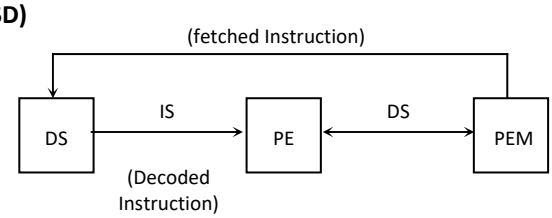
Q.6(a) Explain Flynn classification in detail. **10**

Ans.: Flynn's classification of Computer Organisation :

The classification suggested by Flynn, is based on multiplicity of instruction stream (IS) and data stream (DS).

1. Single Instruction Single Data (SISD)

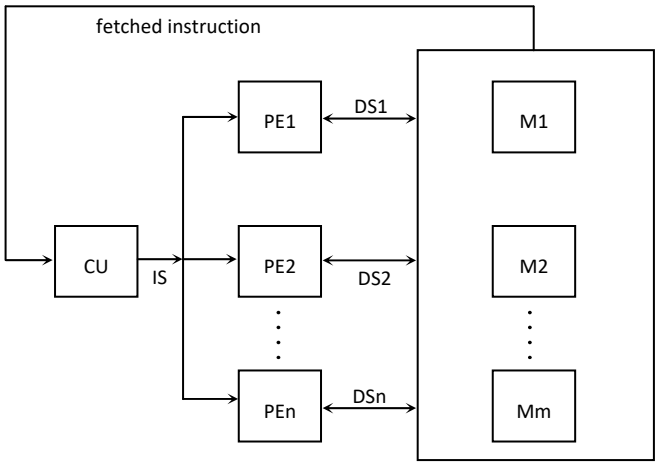
This is normal single processor system (UNI processor) which contains single decoded instruction stream (IS) which operates on a single data stream (DS).



- Here control unit (CU) fetches instruction stream from processing element memory (PEM), which is decoded to generate single decoded instruction stream (IS). This is executed by processing element (PE) using single data stream (DS).
- Once the result data is generated by PE, it is stored back to the PEM.
  Example : of SISD architecture are INTEL, 8085, 8086, 80386.

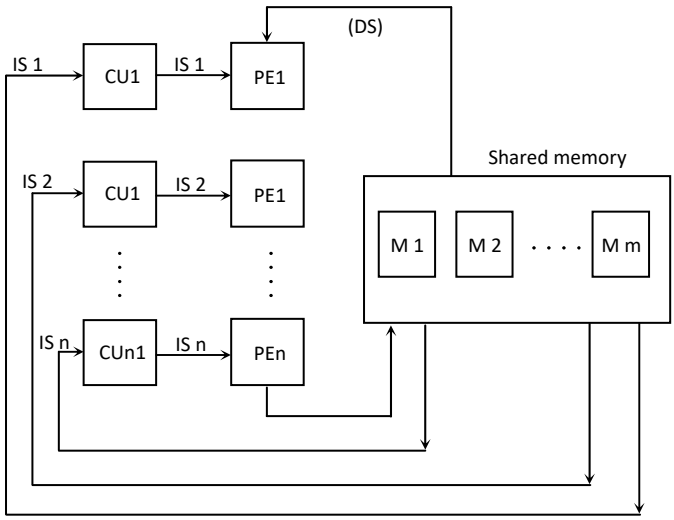
2. Single Instruction Multiple Data (SIMD)

- This is also referred as Array Processor.



Here the single instruction stream is fetched from shared memory modules or taken from external front end system. It is decoded by a single control unit (CU), to generate decoded instruction steak (IS). This instruction is broadcasted to multiple processing elements (PES), which will operate on different data sets. Hence the name given (SIMD). The result generated by processing elements (PEs) are stored back into memory modules.

3. Multiple Instruction Single Data (MISD)

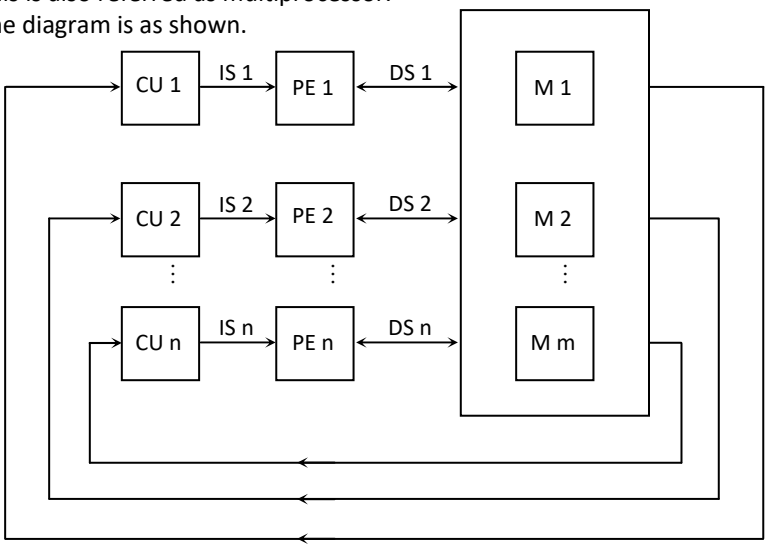


Here multiple instruction streams are fetched form shared memory modules by multiple control units which in turn generates multiple decoded instructions stream (IS). These are operated on single data streak (DS) taken from shared memory modules. MISD architecture is not realized but just a prototype model is designed.

4. Multiple Instruction Multiple Data (MIMD)

This is also referred as multiprocessor.

The diagram is as shown.



Here, multiple instruction streams are fetch by control units. These instruction stream are decoded to get multiple decoded instruction stream (IS), which operates on multiple data stream (DS) taken from shared memory modules. Here, each processing element executes one instruction stream and operate exactly on one data stream.

Q.6(b) Explain instruction cycle state diagram. Explain execution of instruction in detail **10**

Ans.: Instruction Cycle State Diagram

The execution cycle for a particular instruction may involve more than one reference to memory. For any given instruction cycle, some states may be null and others may be visited more than once. In figure shown below, on first line three circles (operations) represent the CPU access to memory or I/O. And on second line five circles (operations) represent the internal CPU operation. The states can be described as :

Instruction Address Calculation (iac) : Determine the address of the next instruction to be executed.

Instruction Fetch (if) : Read instruction from its memory location into the CPU.

Instruction Operation Decoding (iod) : Analyze instruction to determine type of operation to be performed and operand(s) to be used.

Operand Address Calculation (oac) : If the operation involves reference to an operand in memory or variable via I/O then determine the address of the operand.

Operand Fetch (of) : Fetch the operand from memory or read it in from I/O.

Data Operation (do) : Perform the operation indicated in the instruction.

Operand Store (os) : Write the result into memory or out to I/O.

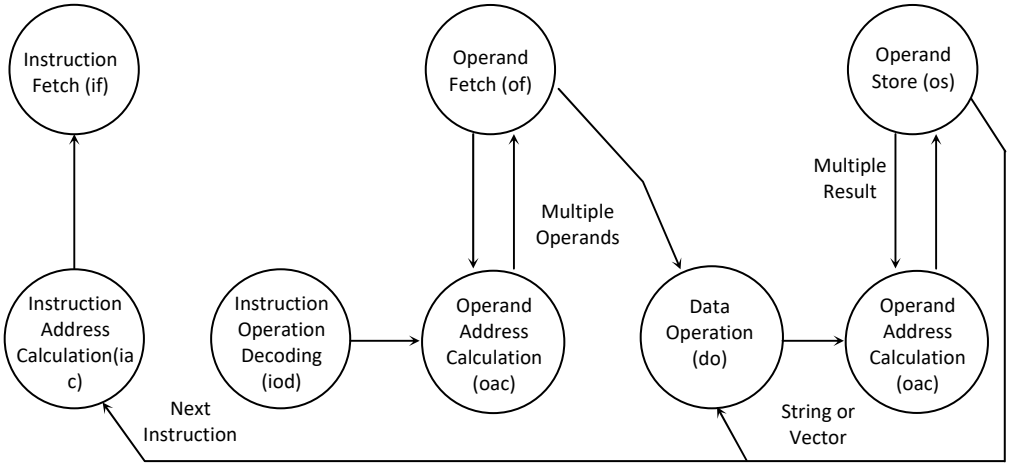


Fig. : Instruction cycle state diagram

Instruction Sequencing :

The basic functions of instruction cycle i.e. fetch and execute. The instructions are prepared in sequence to perform a desired task. The simplest method for instruction sequencing is specify the address of next instruction to be executed in the earlier instruction itself. We can also a special register called as program counter for sequencing.

There are two possible combinations of this program counter implementation.

(1) Straight line sequencing : The instructions are stored in successive memory locations and program counter is used to point to the instruction to be executed. i.e., sequencing is done linearly.

(2) Branching : The branching is also called as program control transfer. There are certain instructions used to select one of several possible addresses for next instruction execution. These instructions are called as branching instructions, these instructions implicitly or explicitly specify the address of next instruction to be executed. Branching may unconditional or conditional.

❑ ❑ ❑ ❑ ❑