UNIT-3

Q) Difference between memory mapped IO & Isolated IO.

→ There are two schemes for connecting I/O device to the processor.

i) Memory Mapped I/O:
In this scheme, I/O device has same address space and addressing scheme as memory.
If there are 50 I/O devices in a systems, the address space available to memory gets reduced by 50. But that also means, no special instructions needed for accessing I/O devices.
Used for some RISC processors where the idea is to reduce no. of instr.

ii) Peripheral or Isolated I/O:
There are some special instructions for input and output devices, and the address space is disjoint and separate from memory address space.
If there are 50 ports, 50 addresses are there. Extra control signals are required.

Q.) Explain two different forms of I/O instructions available in 8086 microprocessor with an example.

→ Two formats available for I/O instruction:

i) Fixed port addressing
This is used only when address of I/O device is 8 bits wide. Here the address of the port is directly mentioned in instruction.

EX:
IN AL, 45H ; mov 8 bit data into AL from input port with address 45H.

ii) Variable port addressing :
It is used when the addressed port has a 16 bit address. Then the port address is to be loaded into DX and then only I/O instruction can be used

Ex:
MOV DX, 9876H; load the add. of port in DX
IN AL, DX ; move 8bit data from the port whose add. is in DX to AL

Q.) What is the relationship b/w EXTERN and PUBLIC directive?

→ PUBLIC :
When a data item or a procedure is to be allowed to be accessed by other modules, it is declared as PUBLIC.
Ex:
PUBLIC num1, num2

EXTERN :
When a module needs to use data & code which have been defined elsewhere, it should use the directive EXTERN.

Syntax: EXTERN name1 : type

Ex:
EXTERN num1 : byte

Q) Why is modular programming important? Discuss in brief.

→ Modular programming is important in following ways:

i) When there is a large programming problem, it is broken into modules, testing the individual module separately, and then integrating the modules together to form the complete solution.

ii) A simple problem can be solved easily by ~~modular programming~~ using program modules which have already been solved & tested.

iii) ~~Another scenario~~
When various teams work on different modules of the same problem & finally integrating it all for a final and complete soln.

→ Issues to be resolved:

i) The diff. modules that constitutes soln may be in diff. code segment.

ii) The data which is to be used by one module may have to be accessed by diff. modules & the permission for this

must be indicated.

iii) Some ~~modules~~ labels used in a module may not be found therein. In that case, there must be some indication that these are defined in some other module to which linking is possible & will be done.

Q.) Write an assembly instruction within C shell for the program to perform addition of two numbers.

```
#include<stdio.h>
#include<iostream.h>
#include<conio.h>
int main()
{
 clrscr();
 char a;

 asm{
  mov ah,01
  int 21h      ; enter first no. with echo
  mov a,al     ; move it to a
  mov dl,'+'   ; display '+'
  mov ah,02
  int 21h
  mov ah,01
  int 21h      ; enter 2nd no. with echo
  mov ah,0     ; ah=0
  add al,a     ; add the two ascii no.
  aaa          ; adjust ascii after addition
  add ax,3030h ; convert sumback to ascii
  mov bx,ax    ; save ascii no. back to ascii
  mov dl,'='   ; display '='
  mov ah,02
  int 21h
```

```
mov dl, bh  ; display upper ascii char
mov ah, 02
int 21h
mov dl, bl  ; display lower ascii character
mov ah, 02
int 21h
}
getch();
return 0;
}
```