Question #1

(i) In real address mode, it stores the base address of assigned memory location (segment)

In protected mode, it stores address to 'segment descriptor table' which contains address information needed to access assigned memory (segment)

(ii) Registers are built in to CPU & hence dont need to be fetched but data from memory has to be fetched by sending signals through control bus which consumes more clock ticks comparitively

(iii) 1. Optimization in game development

2. Developing device drivers

(iv) 1. The instruction pointer (IP) holds the address of the instruction to be executed. The address is placed on address bus & CPU fetches the instruction from instruction queue. It then increments the instruction pointer (IP)

2. The CPU decodes the instruction by looking at its bit pattern. It checks in the bit pattern whether it has operands.

3. If operands are involved, then CPU fetches the operands from memory.

4. Next the ALU performs calculation and its executed then stored in operand. This process is co-ordinated by control bus & clock to ensure flow of data in & out of ALU.

(v) Machine cycle is the clock rate that is fed into the CPU. Instruction cycle is the amount of time it takes to execute a given instruction.

(vi) Java source code is compiled into Java byte code (".class") - a low level language. The bytecode is read by JVM (Java virtual machine) and converted into machine code.
Byte code is always the same on different OS. That makes java program as platform independent.

(vii) The first thing that happens when computer sees the ADD instruction is that it looks up the value in memory with the address 12FCBD10h by sending the address through address bus & bringing the value stored through data bus into the ALU's input register. It then adds the value stored in AL to the ALU's value giving result which is finally sent back to memory (via data bus) and stored at the location 12FCBD10h.

(viii)

(A) Illegal

contents of register cannot be assigned to constant value

(B) Illegal

MOVZX works only when the destination register is larger than the size of source as the instructions extends zeros but since the registers are of same size, it fails

(C) Illegal

since both the operands are not of the same size.

(D) Illegal

Because both the operands cannot be memory locations (one should be reg)

(E) Illegal

An immediate value cannot be incremented.

(ix)

$$\text{var} = \text{word } \overset{2}{\uparrow}\text{"AB"},\ \overset{2}{\uparrow}\text{ABh},\ 20\ \text{DUP}(\overset{20}{\uparrow}10\ \text{DUP}(\text{"AB"}),\ \overset{20}{\uparrow}10\ \text{DUP}(\text{ABh}),\ \overset{2}{\uparrow}\text{"AB"},\ \underset{2}{\text{ABh}})$$

$2 + 2 + 20(20 + 20 + 2 + 2)$

$4 + 20(44)$

$= 884$

(A)

(X) MOV AX, 8F7A H

ADD AX, 7AF8 H

```
   1 1 1
   8 F 7 A
 + 7 A F 8
 ---------
  10 A 7 3
```

CF = 1   OF = 0   SF = 0   ZF = 0

(B)

CF = 0   OF = 0   SF = 1   ZF = 0

Question #2 :

(i) Real Address = Segment x 10 + offset

5 6 0 E 0

\+ 5 3 D 9

⇒ 5 B 4 B 9 h

(ii) offset = Real Address − segment x 10

B 11 E 18 9 3

− 0 8 9 3 0

⇒ B 3 9 6 0 h

(iii) Segment = (Real Address − Offset) / 10

E D 23 12 D

− 5 0 A D

E 8 2 2 0 ÷10

⇒ E 8 2 2 h

Question #3

I)

```
Include Irvine32.inc

.data
A BYTE 20
B BYTE 30

.code
Main proc
    MOV AL, A
    MOV BL, B

    MOV DL, AL
    MOV BL, AL
    MOV AL, DL

    Call DumpRegs
    exit

main ENDP
end main
```

(II)

```
Include Irvine32.inc

.data
X1 WORD 8,5,2,1,10,3,9,6,6,7,9,9,10,11,12,7,1,3,5,2
X2 WORD 15,3,7,2,1,5,6,3,2,1,9,10,2,1,2,3,3,2,1,

.code
Main PROC

    MOV esi , OFFSET X1
    MOV edi , OFFSET X2
    MOV ecx , 20d.

    Target :
        MOV dx , [esi]
        XCHG dx , [edi]
        MOV [esi] , dx
        ADD esi , 2
        ADD edi , 2

        Loop Target

exit
main ENDP
end Main
```

(iii)

```
.code
    main PROC
    MOV esi, OFFSET arr
    MOV ecx, 100
    MOV edx, 0

    Target:
        MOV AL, [esi]
        MOVZX ebx, AL
        ADD EDX, EBX
        inc esi

        Loop Target

    CALL DUMPRegs
    exit
    main ENDP
    end main
```