BMS INSTITUTE OF TECHNOLOGY AND MANAGEMENT
YELAHANKA – BANGALORE - 64

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

II INTERNAL ASSESSMENT TEST SCHEME & SOLUTIONS, APRIL – 2017

Subject: Microprocessors & Microcontrollers	***Subject Code: 15CS44***	***Branch & Semester : CSE - 4 A & B***
Max. Marks : 30 Marks	***Date: 18/04/2017*** ***Time: 2 PM - 3:30 PM***	***Faculty: Mr. Shankar R***

Answer FIVE full questions, selecting ONE full question from each Part.
(Part D & Part E are compulsory)

Q. No	Question	CO, PO, K level	Marks
	PART-A		
1.	**Describe the following instruction with suitable examples:** **AAA XOR CMP DAA CBW LABEL** Each instruction explanation 1 mark **AAA** *ASCII Adjust after Addition.* *Corrects result in AH and AL after addition when working with BCD values.* *It works according to the following Algorithm:* *if low nibble of AL > 9 or AF = 1 then:* *AL = AL + 6* *AH = AH + 1* *AF = 1* *CF = 1* *else* *AF = 0* *CF = 0* *In both cases : clear the high nibble of AL.* **XOR** *Logical XOR (Exclusive OR) between all bits of two operands. Result is stored in first operand.* *These rules apply:* *1 XOR 1 = 0* *1 XOR 0 = 1* *0 XOR 1 = 1* *0 XOR 0 = 0* **CMP** *CMP performs a subtraction of its second operand from its first operand, and affects the flags as if the subtraction had taken place, but does not store the result of the subtraction anywhere.* CMP REG,memory **DAA** *Decimal adjust After Addition.* *Corrects the result of addition of two packed BCD values.* *Algorithm:* *if low nibble of AL > 9 or AF = 1 then:* *AL = AL + 6*	***CO2 (PO1) K1***	06

```
AF = 1
if AL > 9Fh or CF = 1 then:
AL = AL + 60h
CF = 1
```

CBW

Convert byte into word.

Algorithm:

```
if high bit of AL = 1 then:
AH = 255 (0FFh)

else
AH = 0
```

Example:

```
MOV AX, 0   ; AH = 0, AL = 0
MOV AL, -5  ; AX = 000FBh (251)
CBW         ; AX = 0FFFBh (-5)
RET
```

LABEL

As an assembler assembles a section of a data declarations or instruction statements, it uses a location counter to be keep track of how many bytes it is from the start of a segment at any time. The LABEL directive is used to give a name to the current value in the location counter. The LABEL directive must be followed by a term that specifics the type you want to associate with that name. If the label is going to be used as the destination for a jump or a call, then the label must be specified as type near or type far. If the label is going to be used to reference a data item, then the label must be specified as type byte, type word, or type double word. Here's how we use the LABEL directive for a jump address. ¬ ENTRY_POINT LABEL FAR Can jump to here from another segment NEXT: MOV AL, BL Cannot do a far jump directly to a label with a colon

2.

Describe the following instruction with suitable examples:
CLD REPE LODSB SCASB XLAT SAL

Each instruction explanation 1 mark

CO2 (PO1) K1 — 06

CLD

(CLEAR DIRECTION FLAG) This instruction resets the direction flag to 0. It does not affect any other flag.

REPE

REPE and REPZ are two mnemonics for the same prefix. They stand for repeat if equal and repeat if zero, respectively. They are often used with the Compare String instruction or with the Scan String instruction. They will cause the string instruction to be repeated as long as the compared bytes or words are equal (ZF = 1) and CX is not yet counted down to zero. In other words, there are two conditions that will stop the repetition: CX = 0 or string bytes or words not equal.

LODSB

This instruction copies a byte from a string location pointed to by SI to AL, or a word from a string location pointed to by SI to AX. If DF is 0, SI will be automatically incremented (by 1 for a byte string, and 2 for a word string) to point to the next element of the string. If DF is 1, SI will be automatically decremented (by 1 for a byte string, and 2 for a word string) to point to the previous element of the string. LODS does not affect any flag.

```
 CLD                  ;Clear direction flag so that SI is auto-incremented
MOV SI, OFFSET        ;SOURCE Point SI to start of string
LODS SOURCE           ;Copy a byte or a word from string to AL or AX
```

SCASB

SCASB compares a byte in AL or a word in AX with a byte or a word in ES pointed to by DI. Therefore, the string to be scanned must be in the extra segment, and DI must contain the offset of the byte or the word to be compared. If DF is cleared, then DI will be incremented by 1 for byte strings and by 2 for word strings. If DF is set, then DI will be decremented by 1 for byte strings and by 2 for word strings. SCAS affects AF, CF, OF, PF, SF, and ZF, but it does not change either the operand in AL (AX) or the operand in the string.

The following program segment scans a text string of 80 characters for a carriage return, 0DH, and puts the offset of string into DI:

```
 MOV DI, OFFSET STRING
 MOV AL, 0DH                 ; Byte to be scanned for into AL
 MOV CX, 80  ; CX used as element counter CLD Clear DF, so that DI auto increments
 REPNE SCAS STRING          ;Compare byte in string with byte in AL
```

XLAT

The XLAT instruction is used to translate a byte from one code (8 bits or less) to another code (8 bits or less). The instruction replaces a byte in AL register with a byte pointed to by BX in a lookup table in the memory. Before the XLAT instruction can be executed, the lookup table containing the values for a new code must be put in memory, and the offset of the starting address of the lookup table must be loaded in BX. The code byte to be translated is put in AL. The XLAT instruction adds the byte in AL to the offset of the start of the table in BX. It then copies the byte from the address pointed to by (BX + AL) back into AL. XLAT instruction does not affect any flag.

8086 routine to convert ASCII code byte to EBCDIC equivalent: ASCII code byte is in AL at the start, EBCDIC code in AL after conversion.

```
 MOV BX, OFFSET EBCDIC ;Point BX to the start of EBCDIC table in DS
 XLAT                          ; Replace ASCII in AL with EBCDIC from table
```

SAL

SAL and SHL are two mnemonics for the same instruction. This instruction shifts each bit in the specified destination some number of bit positions to the left. As a bit is shifted out of the LSB operation, a 0 is put in the LSB position. The MSB will be shifted into CF. In the case of multi-bit shift, CF will contain the bit most recently shifted out from the MSB. Bits shifted into CF previously will be lost.

The destination operand can be a byte or a word. It can be in a register or in a memory location. If you want to shift the operand by one bit position, you can specify this by putting a 1 in the count position of the instruction. For shifts of more than 1 bit position, load the desired number of shifts into the CL register, and put "CL" in the count position of the instruction.

```
 SAL BX, 1          ;Shift word in BX 1 bit position left, 0 in LSB
MOV CL, 02h        ;Load desired number of shifts in CL
 SAL BP, CL         ;Shift word in BP left CL bit positions, 0 in LSBs
```

PART-B

3a.	**Discuss the differences between INT and CALL instructions.** Any three differences – a mark each. *The INT instruction saves the CS: IP of the following instruction and jumps indirectly to the subroutine associated with the interrupt. A CALL FAR instruction also saves the CS: IP and jumps to the desired subroutine (procedure). The differences can be summarized as follows:*	***CO2 (PO1) K1 CO4 (PO1,PO2) K3***	03 03

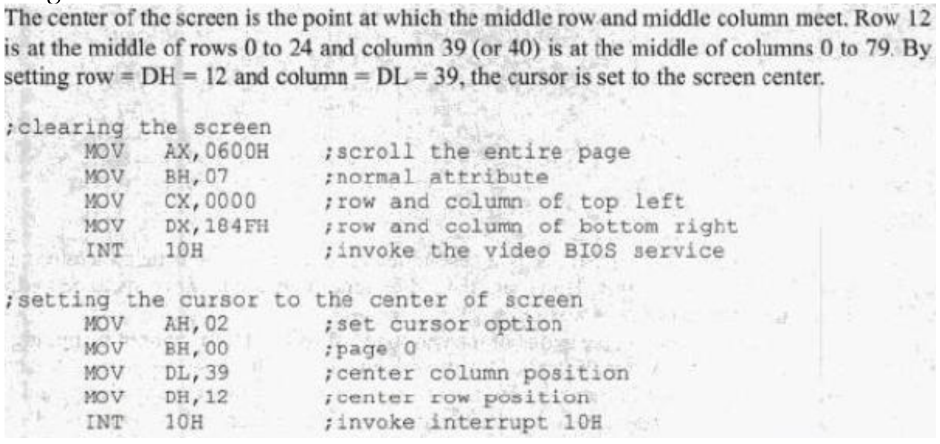
CALL Instruction	*INT instruction*
1. A CALL FAR instruction can jump to any location within the 1M byte address range of the 8088/86 CPU.	*1. INT nn goes to a fixed memory location in the interrupt vector table to get the address of the interrupt service routine.*
2. A CALL FAR instruction is used by the programmer in the sequence of instructions in the program.	*2. An externally activated hardware interrupt can come-in at any time, requesting the attention of the CPU.*
3. A CALL FAR instruction cannot be masked (disabled).	*3. INT nn belonging to externally activated hardware interrupts can be masked.*
4. A CALL FAR instruction automatically saves only CS: IP of the next instruction on the stack.	*4. INT nn saves FR (flag register) in addition to CS: IP of the next instruction.*
5. At the end of the subroutine that has been called by the CALL FAR instruction, the RETF (return FAR) is the last instruction. RETF pops CS and IP off the stack.	*5. The last instruction in the interrupt service routine (ISR) for INT nn is the instruction IRET (interrupt return). IRET pops off the FR (flag register) in addition to CS and IP.*

3b.

Write a program that

- **Clears the screen**
- **Sets the cursor at the center of the screen**

Program –1.5 mark *2 - 3 marks


The center of the screen is the point at which the middle row and middle column meet. Row 12 is at the middle of rows 0 to 24 and column 39 (or 40) is at the middle of columns 0 to 79. By setting row = DH = 12 and column = DL = 39, the cursor is set to the screen center.

```
;clearing the screen
      MOV   AX,0600H     ;scroll the entire page
      MOV   BH,07        ;normal attribute
      MOV   CX,0000      ;row and column of top left
      MOV   DX,184FH     ;row and column of bottom right
      INT   10H          ;invoke the video BIOS service

;setting the cursor to the center of screen
      MOV   AH,02        ;set cursor option
      MOV   BH,00        ;page 0
      MOV   DL,39        ;center column position
      MOV   DH,12        ;center row position
      INT   10H          ;invoke interrupt 10H
```

4a. **Briefly explain the steps taken by a processor to execute an interrupt instruction.**

Steps – 3 marks

CO2 (PO1) K1 03

CO2 (PO1) K1 03

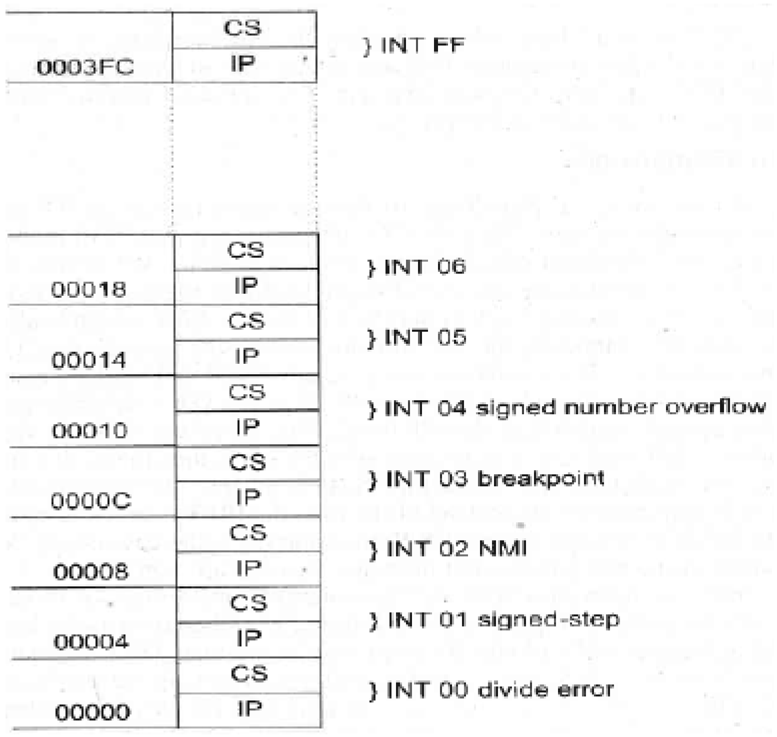
When the 8088/86 processes any interrupt (software or hardware), it goes through the following steps:
1. The flag register (FR) is pushed onto the stack and SP is decremented by 2, since FR is a 2-byte register.
2. IF (interrupt enable flag) and TF (trap flag) are both cleared (IF = 0 and TF = 0). This masks (causes the system to ignore) interrupt requests from the INTR pin and disables single stepping while the CPU is executing the interrupt service routine.

3. The current CS is pushed onto the stack and SP is decremented by 2.
4. The current IP is pushed onto the stack and SP is decremented by 2.
5. The INT number (type) is multiplied by 4 to get the physical address of the location within the vector table to fetch the CS and IP of the interrupt service routine.
6. From the new CS: IP, the CPU starts to fetch and execute instructions belonging to the ISR program.
7. The last instruction of the interrupt service routine must be IRET, to get IP, CS, and FR back from the stack and make the CPU run the code where it left off.

4b. **Explain Interrupt Vector table and Interrupt Service Routine**

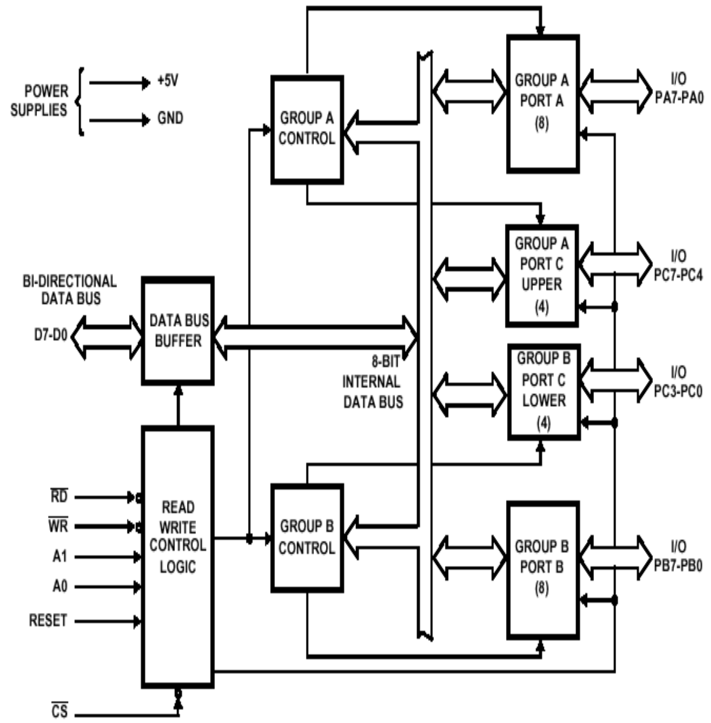
Explanation of both – 3marks

*An interrupt is an external event that informs the CPU that a device needs its service. In 8088/86, there are 256 interrupts: INT 00, INT 01, . . . , INT FF (sometimes called TYPEs). When an interrupt is executed, the microprocessor automatically saves the flag register (FR), the instruction pointer (IP), and the code segment register (CS) on the stack; and goes to a fixed memory location. In x86 PCs, the memory locations to which an interrupt goes is always four times the value of the interrupt number. For example, INT 03 will go to address 0000CH (4 * 3 = 12 = 0CH). The following Table is a partial list of the interrupt vector table.*



Interrupt Service Routine (ISR):

- *For every interrupt there must be a program associated with it.*
- *When an interrupt is invoked, it is asked to run a program to perform a certain service. This program is commonly referred to as an interrupt service routine (ISR). The interrupt service routine is also called the interrupt handler.*
- *When an interrupt is invoked, the CPU runs the interrupt service routine. As shown in the above Table, for every interrupt there are allocated four bytes of memory in the interrupt vector table. Two bytes are for the IP and the other two are for the CS of the ISR.*
- *These four memory locations provide the addresses of the interrupt service routine for which the interrupt was invoked. Thus the lowest 1024 bytes (256 x 4 = 1024) of memory space are set aside for the interrupt vector table and must not be used for any*

	other function.		
PART-C			
5.	**With a neat block diagram explain 82C55 PPI.** Block diagram – 2 marks Explanation – 4 marks  POWER SUPPLIES +5V GND GROUP A CONTROL GROUP A PORT A (8) I/O PA7-PA0 GROUP A PORT C UPPER (4) I/O PC7-PC4 BI-DIRECTIONAL DATA BUS D7-D0 DATA BUS BUFFER 8-BIT INTERNAL DATA BUS GROUP B PORT C LOWER (4) I/O PC3-PC0 $\overline{RD}$ $\overline{WR}$ A1 A0 RESET READ WRITE CONTROL LOGIC GROUP B CONTROL GROUP B PORT B (8) I/O PB7-PB0 $\overline{CS}$ ***Data Bus Buffer*** *This three-state bi-directional 8-bit buffer is used to interface the 8255 to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.* ***Read/Write and Control Logic*** *The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.* ***(CS)*** *Chip Select. A "low" on this input pin enables the communication between the 8255 and the CPU.* ***(RD)*** *Read. A "low" on this input pin enables 8255 to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255.* ***(WR)*** *Write. A "low" on this input pin enables the CPU to write data or control words into the 8255.* ***(A0 and A1)*** *Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word register. They are normally connected to the least significant bits of the address bus (A0 and A1).* ***(RESET)*** *Reset. A "high" on this input initializes the control register to 9Bh and all ports (A, B, C) are set to the input mode.*	***CO2 (PO1) K3***	06

A1	A0	SELECTION
0	0	PORT A
0	1	PORT B
1	0	PORT C
1	1	CONTROL

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 8255. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255. Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Ports A, B, and C

The 8255 contains three 8-bit ports (A, B, and C). All can be configured to a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255.

Port A *One 8-bit data output latch/buffer and one 8-bit data input latch. Both "pull-up" and "pull-down" bus-hold devices are present on Port A.*

Port B *One 8-bit data input/output latch/buffer and one 8-bit data input buffer.*

Port C *One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal output and status signal inputs in conjunction with ports A and B.*

6.

Explain the control word format of 8255 PPI. Write the control words for

- **PORT A as input, PORT B as output, PORT C as output**
- **PORT A as output, PORT B as input, PORT C as input in simple I/O mode.**

*Explanation 3 marks * 2 – 6 marks.*
1. 99h
2. 82h

CO2,CO4 (PO1,PO3) K3

06

PART-D

7.

Discuss the various cases of MUL & DIV instructions with examples.

<u>The MUL/IMUL Instruction</u>

There are two instructions for multiplying binary data. The MUL (Multiply) instruction handles unsigned data and the IMUL (Integer Multiply) handles signed data. Both instructions affect the Carry and Overflow flag.

Syntax

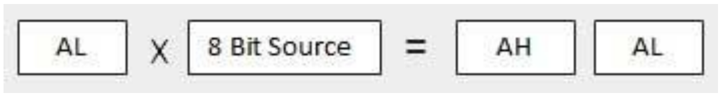
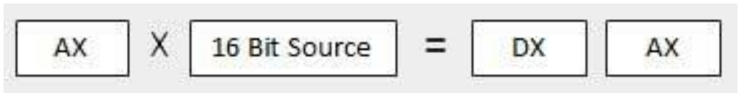
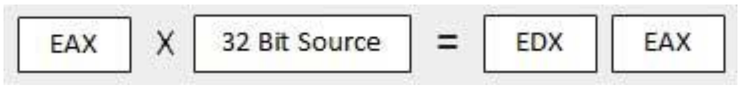
The syntax for the MUL/IMUL instructions is as follows −

MUL/IMUL multiplier

Multiplicand in both cases will be in an accumulator, depending upon the size of the multiplicand and the multiplier and the generated product is also stored in two registers depending upon the size of the operands. Following section explains MUL instructions with three different cases −

CO2 (PO1,PO2) K4

06

SN	Scenarios
1	*When two bytes are multiplied -* *The multiplicand is in the AL register, and the multiplier is a byte in the memory or in another register. The product is in AX. High-order 8 bits of the product is stored in AH and the low-order 8 bits are stored in AL.* 
2	*When two one-word values are multiplied -* *The multiplicand should be in the AX register, and the multiplier is a word in memory or another register. For example, for an instruction like MUL DX, you must store the multiplier in DX and the multiplicand in AX.* *The resultant product is a doubleword, which will need two registers. The high-order (leftmost) portion gets stored in DX and the lower-order (rightmost) portion gets stored in AX.* 
3	*When two doubleword values are multiplied -* *When two doubleword values are multiplied, the multiplicand should be in EAX and the multiplier is a doubleword value stored in memory or in another register. The product generated is stored in the EDX:EAX registers, i.e., the high order 32 bits gets stored in the EDX register and the low order 32-bits are stored in the EAX register.* 

The DIV/IDIV Instructions

The division operation generates two elements - a quotient and a remainder. In case of multiplication, overflow does not occur because double-length registers are used to keep the product. However, in case of division, overflow may occur. The processor generates an interrupt if overflow occurs.

The DIV (Divide) instruction is used for unsigned data and the IDIV (Integer Divide) is used for signed data.

Syntax

The format for the DIV/IDIV instruction –

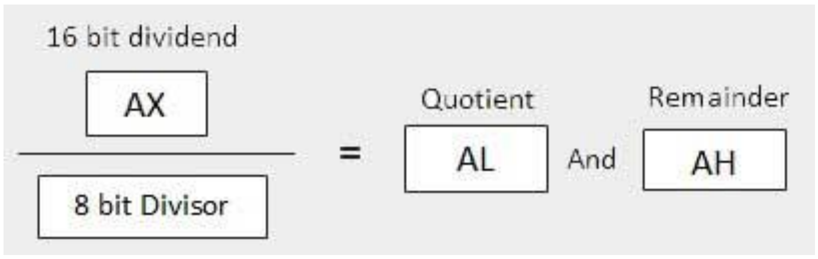
DIV/IDIV divisor

The dividend is in an accumulator. Both the instructions can work with 8-bit, 16-bit or 32-bit operands. The operation affects all six status flags. Following section explains three cases of division with different operand size –

S N   *Scenarios*

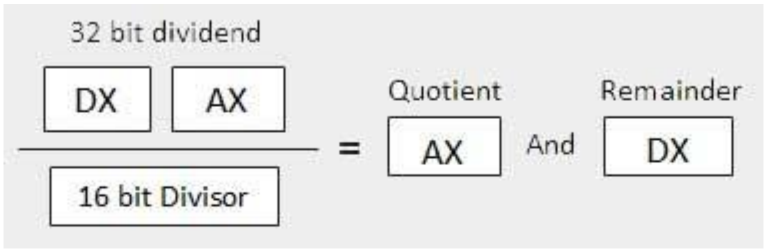
1   *When the divisor is 1 byte -*

The dividend is assumed to be in the AX register (16 bits). After division, the quotient goes to the AL register and the remainder goes to the AH register.



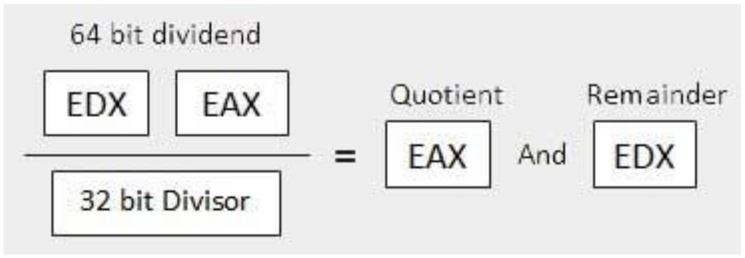
2   *When the divisor is 1 word -*

The dividend is assumed to be 32 bits long and in the DX:AX registers. The high-order 16 bits are in DX and the low-order 16 bits are in AX. After division, the 16-bit quotient goes to the AX register and the 16-bit remainder goes to the DX register.



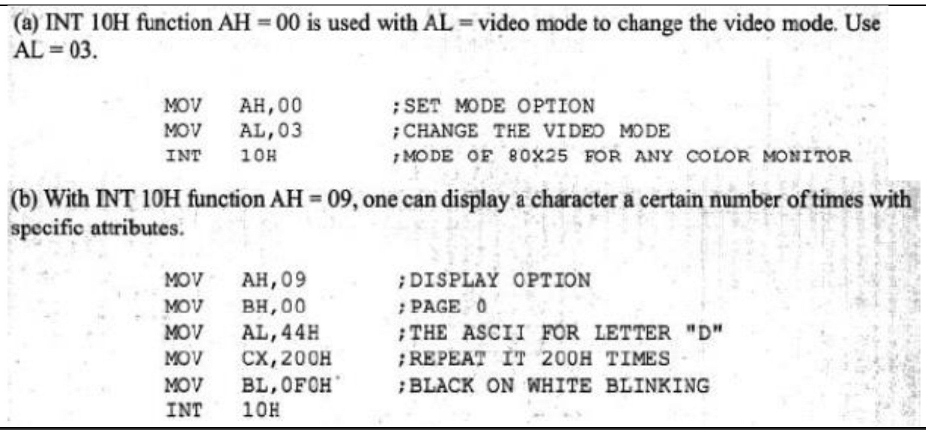
3   *When the divisor is doubleword -*

The dividend is assumed to be 64 bits long and in the EDX:EAX registers. The high-order 32 bits are in EDX and the low-order 32 bits are in EAX. After division, the 32-bit quotient goes to the EAX register and the 32-bit remainder goes to the EDX register.



PART - E

8.	**Write a program using INT 10h to:** ♦ **Change the video mode** ♦ **Display the letter "D" in 200H locations with attributes black on white blinking (blinking letters "D" are black and the screen background is white)** Program 3 marks * 2 – 6 marks.	***CO2,CO5 (PO2,PO3) K6***	06

	
	(a) INT 10H function AH = 00 is used with AL = video mode to change the video mode. Use AL = 03.
	`MOV   AH,00      ;SET MODE OPTION`
	`MOV   AL,03      ;CHANGE THE VIDEO MODE`
	`INT   10H        ;MODE OF 80X25 FOR ANY COLOR MONITOR`
	(b) With INT 10H function AH = 09, one can display a character a certain number of times with specific attributes.
	`MOV   AH,09      ;DISPLAY OPTION`
	`MOV   BH,00      ;PAGE 0`
	`MOV   AL,44H     ;THE ASCII FOR LETTER "D"`
	`MOV   CX,200H    ;REPEAT IT 200H TIMES`
	`MOV   BL,0F0H    ;BLACK ON WHITE BLINKING`
	`INT   10H`		

Course Outcomes: Students will be able to	
CO2	*Describe the architecture of X86 Microprocessors and have an introduction to Assembly Language Programming.*
CO2	*Discuss the Instruction Set of X86 Microprocessors and extend it to interface various devices to X86 families*
CO3	*Understand ARM philosophy and its Instruction Set.*
CO4	*Demonstrate the skills to code in Assembly Language, ARM.*
CO5	*Construct software and hardware programs using Assembly Language Programming, ARM.*

K1: Remember	***K2:Understand***	***K3: Apply***	***K4: Analyze***	***K5: Evaluate***	***K6: Creation***