V2V EDTECH LLP

Online Coaching at an Affordable Price.

OUR SERVICES:

- Diploma in All Branches, All Subjects
- Degree in All Branches, All Subjects
- BSCIT / CS
- Professional Courses

+91 93260 50669

v2vedtech.com

V2V EdTech LLP

v2vedtech

SUMMER – 2023 EXAMINATION
Model Answer – Only for the Use of RAC Assessors

Subject Name: Microprocessors **Subject Code:** 22415

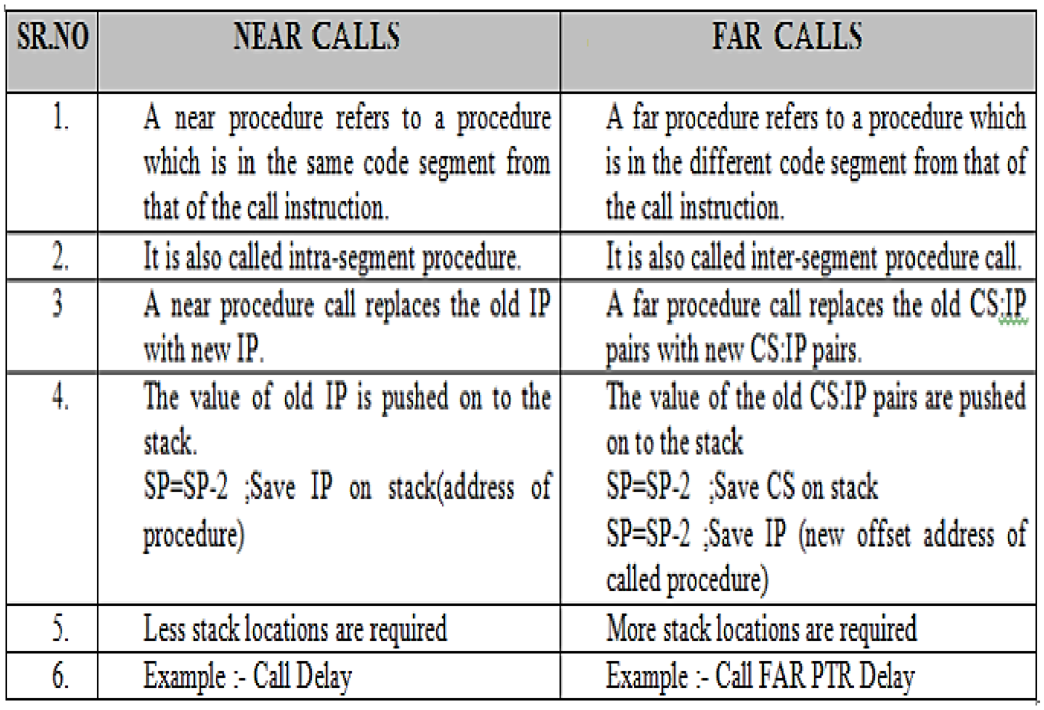
Important Instructions to examiners:
1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.
8) As per the policy decision of Maharashtra State Government, teaching in English/Marathi and Bilingual (English + Marathi) medium is introduced at first year of AICTE diploma Programme from academic year 2021-2022. Hence if the students in first year (first and second semesters) write answers in Marathi or bilingual language (English +Marathi), the Examiner shall consider the same and assess the answer based on matching of concepts with model answer.

Q. No.	Sub Q. N.	Answer	Marking Scheme
1		**Attempt any FIVE of the following:**	**10 M**
	a)	**State the functions of the following pins of 8086 Microprocessor :** **i) ALE** **ii) M/IO**	**2 M**
	Ans	**ALE** - It stands for address enable latch and is available at pin 25. A positive pulse is generated each time the processor begins any operation. This signal indicates the availability of a valid address on the address/data lines. **M/IO -** This signal is used to distinguish between memory and I/O operations. When it is high, it indicates I/O operation and when it is low indicating the memory operation. It is available at pin 28.	1 M 1 M
	b)	**State the function of STC and CMC Instruction of 8086.**	**2 M**
	Ans	**STC** – This instruction is used to Set Carry Flag. CF $\Leftarrow$ 1 **CMC** – This instruction is used to Complement Carry Flag. CF $\Leftarrow$ ~ CF	1 M 1 M

	c)	**List the program development steps for assembly language programming.**	**2 M**
	Ans	**Program Development steps**: 1. Defining the problem 2. Algorithm 3. Flowchart 4. Initialization checklist 5. Choosing instructions 6. Converting algorithms to assembly language program	2 M
	d)	**Define MACRO with its syntax.**	**2 M**
	Ans	Macro: A MACRO is group of small instructions that usually performs one task. It is a reusable section of a software program. A macro can be defined anywhere in a program using directive MACRO &ENDM. **Syntax:** MACRO-name MACRO [ARGUMENT 1,……….ARGUMENT N] ----- ENDM	1 M 1 M
	e)	**Write an ALP to Add two 16-bit numbers.**	**2 M**
	Ans	data segment a dw 0202h b dw 0408h c dw ? data ends code segment assume cs:code,ds:data start: mov ax,data mov ds,ax mov ax,a mov bx,b add ax,bx mov c,ax int 03h code ends **end** start	Any correct program – 2 M

	f)	**State two examples of each, Immediate and based indexed Addressing modes.**	**2 M**
	Ans	Immediate Addressing mode: 1. MOV AX, 2000H 2. MOV CL, 0AH 3. ADD AL, 45H 4. AND AX, 0000H Based indexed Addressing mode: 1. ADD CX, [AX+SI] 2. MOV AX, [AX+DI] 3. MOV AL, [SI+BP+2000]	1 M for any two valid instructions 1 M for any two valid instructions
	g)	**State the use of OF and AF flags in 8086.**	**2 M**
	Ans	**Auxiliary Carry Flag (AF):** This flag is used in BCD (Binary-coded Decimal) operations. This flag is set to 1 if there is a CARRY from the lower nibble or BORROW for the lower nibble in binary representation; else it is set to zero. **Overflow Flag (OF):** This flag will be set (1) if the result of a signed operation is too large to fit in the number of bits available to represent it, otherwise reset (0).	1 M 1 M
2.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Differentiate between NEAR and FAR CALLS.**	**4 M**

Ans



SR.NO	NEAR CALLS	FAR CALLS
1.	A near procedure refers to a procedure which is in the same code segment from that of the call instruction.	A far procedure refers to a procedure which is in the different code segment from that of the call instruction.
2.	It is also called intra-segment procedure.	It is also called inter-segment procedure call.
3	A near procedure call replaces the old IP with new IP.	A far procedure call replaces the old CS:IP pairs with new CS:IP pairs.
4.	The value of old IP is pushed on to the stack. SP=SP-2 ;Save IP on stack(address of procedure)	The value of the old CS:IP pairs are pushed on to the stack SP=SP-2 ;Save CS on stack SP=SP-2 ;Save IP (new offset address of called procedure)
5.	Less stack locations are required	More stack locations are required
6.	Example :- Call Delay	Example :- Call FAR PTR Delay

1 M for each valid point

b) **Explain the concept of memory segmentation in 8086.** **4 M**

Ans

Memory Segmentation: The memory in an 8086 microprocessor is organized as a segmented memory. The physical memory is divided into 4 segments namely, - Data segment, Code Segment, Stack Segment and Extra Segment.

Explanation- 2 M,

Diagram- 2 M

Description:

• Data segment is used to hold data, Code segment for the executable program, Extra segment also holds data specifically in strings and stack segment is used to store stack data.

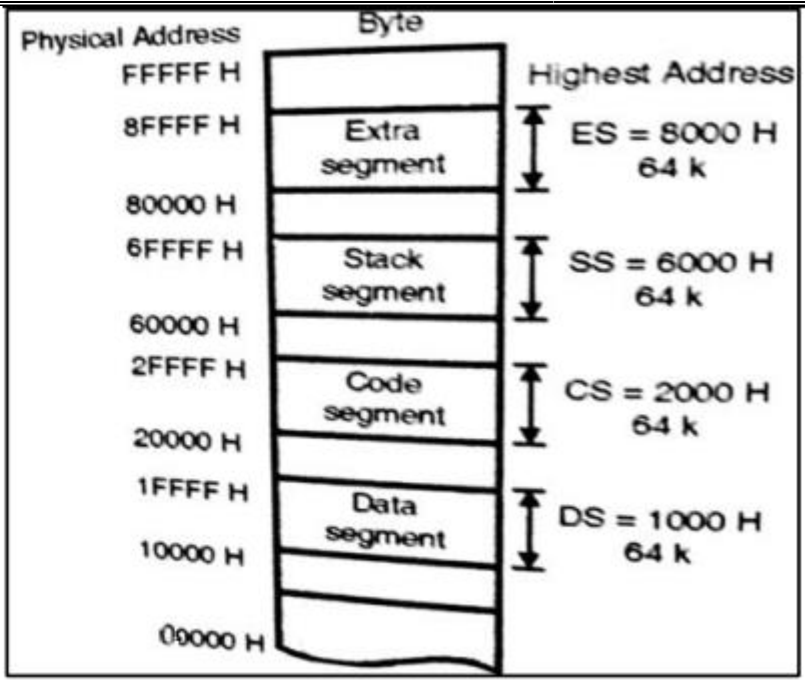
• Each segment is 64Kbytes & addressed by one segment register. i.e. CS, DS, ES or SS

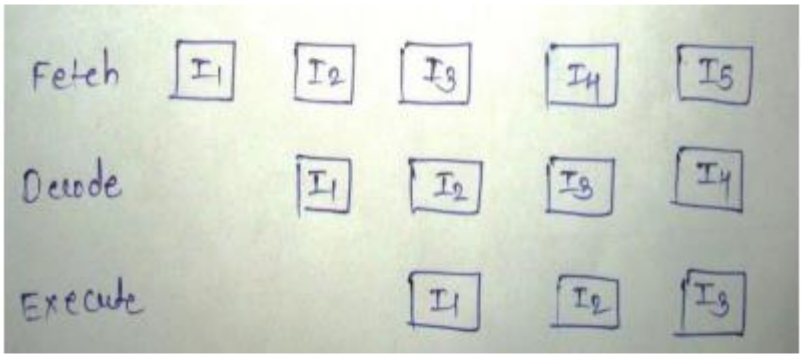
• The 16-bit segment register holds the starting address of the segment.

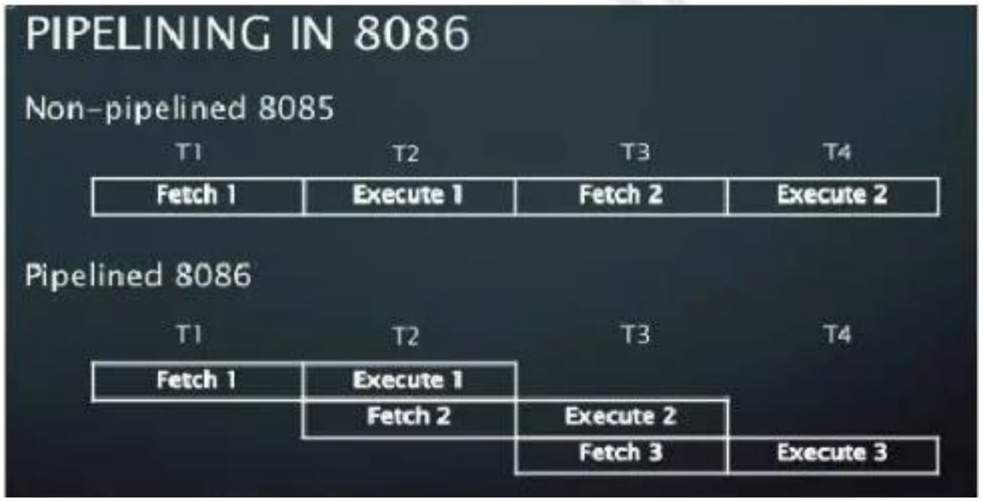
• The offset address to this segment address is specified as a 16-bit displacement (offset) between 0000 to FFFFH. Hence maximum size of any segment is 216=64K locations.

• Since the memory size of 8086 is 1Mbytes, total 16 segments are possible with each having 64Kbytes.

• The offset address values are from 0000H to FFFFH, so the physical address range from 00000H to FFFFFH.

		 Physical Address Byte FFFFF H Highest Address 8FFFF H Extra segment ES = 8000 H 64 k 80000 H 6FFFF H Stack segment SS = 6000 H 64 k 60000 H 2FFFF H Code segment CS = 2000 H 64 k 20000 H 1FFFF H Data segment DS = 1000 H 64 k 10000 H 09000 H	
	c)	**State the Assembler Directives used in 8086 and describe the function of any two.**	**4 M**
	Ans	**Assembler directives:** 1) DW 2) EQU 3) ASSUME 4) OFFSET 5) SEGMENT 6) EVEN **Function of any two:** **1)DW (DEFINE WORD):** The DW directive is used to tell the assembler to define a variable of type word or to reserve storage locations of type word in memory. The statement MULTIPLIER DW 437AH, for example, declares a variable of type word named MULTIPLIER, and initialized with the value 437AH when the program is loaded into memory to be run. **2)EQU (EQUATE):** EQU is used to give a name to some value or symbol. Each time the assembler finds the given name in the program, it replaces the name with the value or symbol you equated with that name. **Example:** **Data SEGMENT** **Num1 EQU 50H** **Num2 EQU 66H** **Data ENDS** Numeric value 50H and 66H are assigned to Num1 and Num2.	List - 2 M Function of each directive 1 M
	d)	**Identify the Addressing Modes for the following instructions:**	**4 M**

		I. MOV CL, 34H **II. MOV BX, [4100H]** **III. MOV DS, AX** **IV. MOV AX, [SI+BX+04]**	
	Ans	I. MOV CL, 34H: Immediate addressing mode. II. MOV BX, [4100H]: Direct addressing mode. III. MOV DS, AX: Resister addressing mode. IV. MOV AX, [SI+BX+04]: Relative Base Index addressing mode.	1 M 1 M 1 M 1 M
3.		**Attempt any THREE of the following:**	**12 M**
	a)	**Explain the concept of pipelining in 8086 microprocessor with diagram.**	**4 M**
	Ans	• In 8086, pipelining is the technique of overlapping instruction fetch and execution mechanism. • To speed up program execution, the BIU fetches as many as six instruction bytes ahead of time from memory. The size of instruction prefetching queue in 8086 is 6 bytes. • While executing one instruction other instruction can be fetched. Thus it avoids the waiting time for execution unit to receive other instruction. • BIU stores the fetched instructions in a 6 level deep FIFO. The BIU can be fetching instructions bytes while the EU is decoding an instruction or executing an instruction which does not require use of the buses • When the EU is ready for its next instruction, it simply reads the instruction from the queue in the BIU • This is much faster than sending out an address to the system memory and waiting for memory to send back the next instruction byte or bytes. • This improves overall speed of the processor. 	Explanation- 3 M, Diagram- 1 M

		OR 	
	b)	**Write an alp to perform block transfer operation of 10 numbers**	**4 M**
	Ans	**WITHOUT STRING INSTRUCTION** .MODEL SMALL .DATA ARR1 DB 00H,01H,02H,03H,04H,05H,06,07H.08H.09H ARR2 DB 10 DUP(00H) ENDS .CODE START: MOV AX, @DATA MOV DS,AX MOV SI, OFFSET ARR1 MOV DI, OFFSET ARR2 MOV CX ,0000A BACK: MOV AL,[SI] MOV [DI],AL INC SI INC DI	Correct program - 4 M

		LOOP BACK MOV AH,4CH INT 21H ENDS END START **OR** **WITH STRING INSTRUCTION** .MODEL SMALL .DATA ARR1 DB 00H, 01H,02H,03H,04H,05H,06,07H.08H.09H ARR2 DB 10 DUP(00H) ENDS .CODE START:MOV AX,@DATA MOV DS,AX MOV SI,OFFSET ARR1 MOV DI, OFFSET ARR2 MOV CX,0000A REP MOVSB MOV AH,4CH INT 21H ENDS END START	
	c)	**Write an ALP to subtract two BCD number's.**	**4 M**
	Ans	.MODEL SMALL .DATA NUM1 DB 86H NUM2 DB 57H	Correct program - 4 M

```
 ENDS
.CODE
START:
MOV AX@,DATA
MOV DS,AX
MOV AL,NUM1
SUB AL,NUM2
 DAS
MOV BL,AL //    STORE FINAL RESULT IN BL REGISTER
MOV AH,4CH
 INT 21H
 ENDS
 END START
```

d) **Compare procedure and macros (4 points).** **4 M**

Ans

Sr.No.	**MACRO**	**PROCEDURE**
1	Macro is a small sequence of code of the same pattern, repeated frequently at different places, which perform the same operation on different data of the same data type	Procedure is a series of instructions is to be executed several times in a program, and called whenever required.
2	The MACRO code is inserted into the program, wherever MACRO is called, by the assembler	Program control is transferred to the procedure, when CALL instruction is executed at run time.
3	Memory required is more, as the code is inserted at each MACRO call	Memory required is less, as the program control is transferred to procedure.
4	Stack is not required at the MACRO call.	Stack is required at Procedure CALL
5.	Less time required for its execution	Extra time is required for linkage between the calling program and called procedure.

One point 1 M each

6	Parameter passed as the part of statement which calls macro.	Parameters passed in registers, memory locations or stack.
7	RET is not used	RET is required at the end of the procedure
8	Macro is called< Macro NAME> [argument list]	Procedure is called using: CALL< procedure name>
9	Directives used: MACRO, ENDM,	Directives used: PROC, ENDP

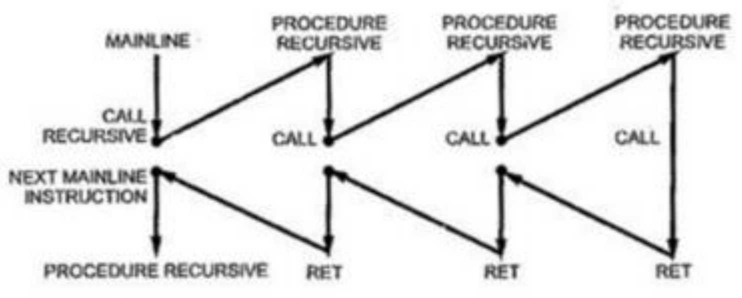
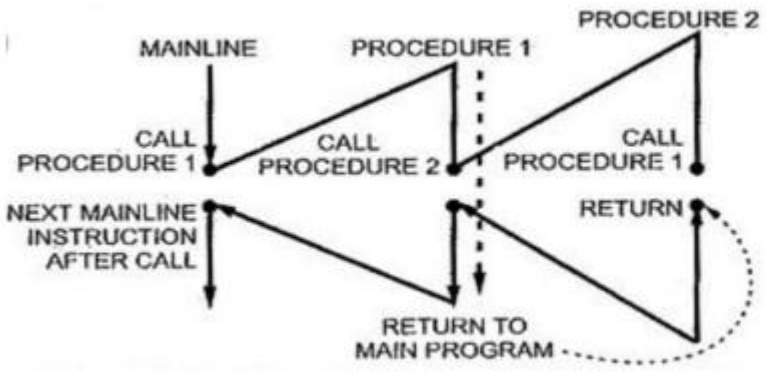
4.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Differentiate between minimum mode and maximum of 8086 microprocessor.**	**4 M**
	Ans		Any four points- 4 M

Sr.No.	**Minimum Mode**	**Maximum Mode**
1	MN/MX’ pin is connected to Vcc. i.e. MN/MX = 1	MN/MX’ pin is connected to ground. i.e. MN/MX = 0
2	Control system M/ IO’ , RD’ , WR’ is available on 8086 directly	Control system M/ IO’ , RD’ , WR’ is not available directly in 8086
3	Single processor in the minimum mode system	Multiprocessor configuration in maximum mode system
4	In this mode, no separate bus controller is required	Separate bus controller (8288) is required in maximum mode
5	Control signals such as IOR’ , IOW’ , MEMW’ , MEMR’ can be generated using control signals M/IO , RD , WR which are available on 8086 directly.	Control signals such as MRDC’ , MWTC’ , AMWC’ , IORC’ , IOWC’ , and AIOWC’ are generated by bus controller 8288.
6	HOLD and HLDA signals are available to interface another master in system such as DMA controller.	RQ / GTQ and RQ / GT 1 signals are available to interface another master in system such as DMA

			controller and coprocessor 8087.
	7	This circuit is simpler	This circuit is complex

	b)	**Write an ALP for sum of series of 05 number's.**	**4 M**
	Ans	.MODEL SMALL .DATA NUM1 DB 10H,20H,30H,40H,50H RESULT DB 00H CARRY DB 00H ENDS .CODE START: MOV AX,@DATA MOV DS, AX MOV CL,05H MOV SI, OFFSET NUM1 UP:MOV AL,[SI] ADD RESULT, AL JNC NEXT INC CARRY NEXT: INC SI LOOP UP MOV AH,4CH INT 21H ENDS END START	Correct program - 4 M
	c)	**Write an ALP to find largest number from array of 10 number's.**	**4 M**
	Ans	.MODEL SMALL .DATA	Correct program - 4 M

		ARRAY DB 02H,04H,06H,01H,05H,09H,0AH,0CH.00H,07H ENDS .CODE START: MOV AX,@DATA MOV DS,AX MOV CL,09H LEA SI,ARRAY MOV AL,[SI] UP : INC SI CMP AL,[SI] JNC NEXT MOV AL[SI] NEXT : DEC CL JNZ UP MOV AH,4CH INT 21H ENDS END START	
	d)	**Describe re-entrant and Recursive procedure with diagram.**	**4 M**
	Ans	A recursive procedure is procedure which calls itself. This results in the procedure call to be generated from within the procedures again and again. The recursive procedures keep on executing until the termination condition is reached. The recursive procedures are very effective to use and to implement but they take a large amount of stack space and the linking of the procedure within the procedure takes more time as well as puts extra load on the processor.	Explanation re-entrant and Recursive- 2M each

		 MAINLINE PROCEDURE RECURSIVE PROCEDURE RECURSIVE PROCEDURE RECURSIVE CALL RECURSIVE CALL CALL CALL NEXT MAINLINE INSTRUCTION PROCEDURE RECURSIVE RET RET RET **2) Re-entrant procedures**: In some situation it may happen that Procedure 1 is called from main program, Procrdure2 is called from procedure1And procedure1 is again called from procdure2. In this situation program execution flow re-enters in the procedure1. These types of procedures are called re-entrant procedures. A procedure is said to be re-entrant, if it can be interrupted, used and re-entered without losing or writing over anything.  MAINLINE PROCEDURE 1 PROCEDURE 2 CALL PROCEDURE 1 CALL PROCEDURE 2 CALL PROCEDURE 1 NEXT MAINLINE INSTRUCTION AFTER CALL RETURN RETURN TO MAIN PROGRAM	
	e)	**Explain MACRO with suitable example. List four advantages of it.**	**4 M**
	Ans	• Macro is a small sequence of code of the same pattern, repeated frequently at different places, which perform the same operation on different data of the same data type • The MACRO code is inserted into the program, wherever MACRO is called, by the assembler • Memory required is more, as the code is inserted at each MACRO call Syntax: Macro_name MACRO [arg1,arg2,.....argN) ..... endM	Macro explanation- 1 M, Example- 1 M, Advantages- 2 M

		Example: .MODEL SMALL PROG MACRO A,B MOV AL,A MUL AL MOV BL,AL MOV AL,B MUL AL ADD AL,BL ENDM .DATA X DB 02H Y DB 03H P DB DUP() ENDS .CODE START: MOV AX,DATA MOV DS,AX PROG X, Y MOV P,AL MOV AH,4CH INT 21H END START ENDS **Advantages of Macro:** 1) Program written with macro is more readable. 2) Macro can be called just writing by its name along with parameters, hence no extra code is required like CALL & RET.	(Any Same Type of Example can be considered)

		3) Execution time is less because of no linking and returning to main program. 4) Finding errors during debugging is easier.	
5.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Define logical and effective address. Describe Physical address generation in 8086. If CS = 2135 H and IP = 3478H, calculate Physical Address.**	**6 M**
	Ans	**<u>A logical address</u>**: A logical address is the address at which an item (memory cell, storage element) appears to reside from the perspective of an executing application program. A logical address may be different from the physical address due to the operation of an address translator or mapping function. **<u>Effective Address or Offset Address</u>**: The offset for a memory operand is called the operand's effective address or EA. It is an unassigned 16-bit number that expresses the operand's distance in bytes from the beginning of the segment in which it resides. In 8086 we have base registers and index registers. <u>Procedure for Generation of 20-bit physical address in 8086: -</u> 1. Segment registers carry 16-bit data, which is also known as base address. 2. BIU appends four 0 bits to LSB of the base address. This address becomes 20-bit address. 3. Any base/pointer or index register carries 16 bits offset. 4. Offset address is added into 20-bit base address which finally forms 20-bit physical address of memory location CS=2135H and IP=3475H Physical address = CS*10H + IP = 2135H * 10H + 3475H = 21350 + 3475 = 247C5H	Defination-3M, Physical address generation-3M
	b)	**Explain the following assembler directives:** **(i) DB (ii) DW (iii) EQU (iv) DUP (v) SEGMENT (vi) END**	**6 M**
	Ans	**(i)** **<u>DB</u>** (Define Byte) – The DB directive is used to declare a BYTE -2-BYTE variable – A BYTE is made up of 8 bits. Declaration Examples:	Each assembler

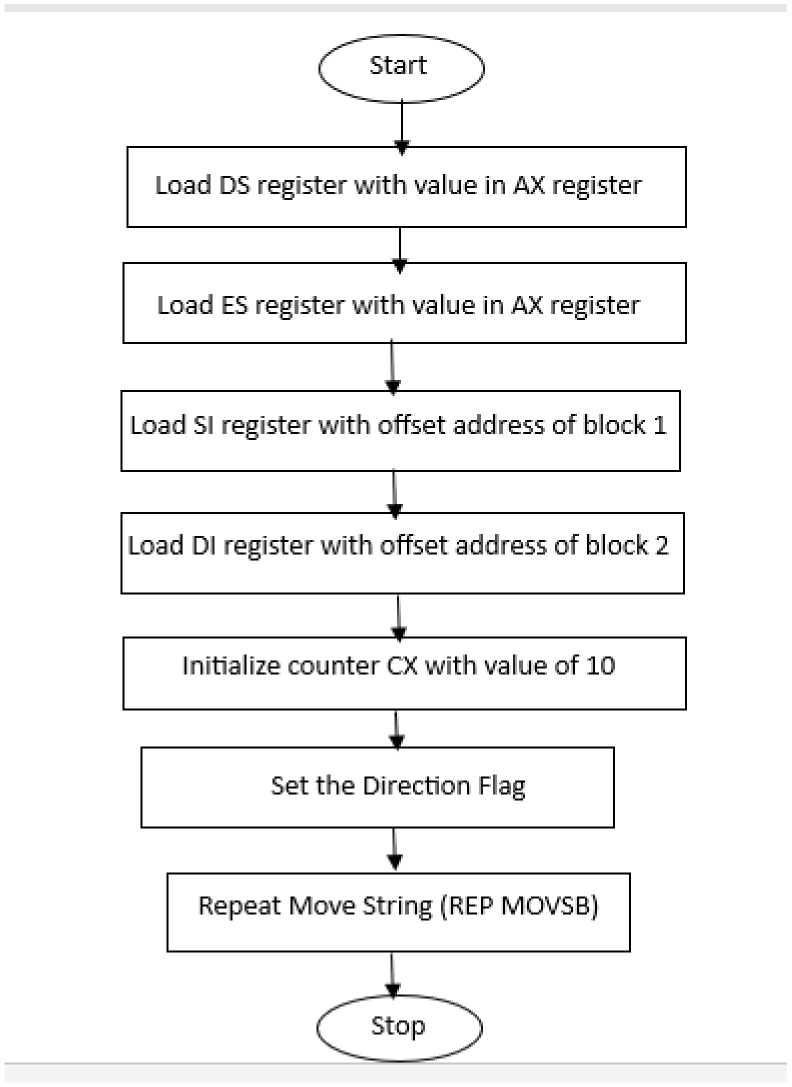
		Byte1 DB 10h Byte2 DB 255; 0FFh, the max. possible for a BYTE CRLF DB 0Dh, 0Ah, 24h; Carriage Return, terminator BYTE **(ii)** **DW (Define Word)**: The DW directive is used to tell the assembler to define a variable of type word or to reserve storage locations of type word in memory. The statement MULTIPLIER DW 437AH. Example, declares a variable of type word named MULTIPLIER, and initialized with the value 437AH when the program is loaded into memory to be run. **(iii)** **EQU (EQUATE)**: EQU is used to give a name to some value or symbol. Each time the assembler finds the given name in the program, it replaces the name with the value or symbol you equated with that name. Example - Data SEGMENT Num1 EQU 50H Num2 EQU 66H Data ENDS Numeric value 50H and 66H are assigned to Num1 and Num2. **(iv)** **DUP**: - It can be used to initialize several locations to zero. e. g. SUM DW 4 DUP(0) - Reserves four words starting at the offset sum in DS and initializes them to Zero. - Also used to reserve several locations that need not be initialized. In this case (?) is used with DUP directives. E. g. PRICE DB 100 DUP(?) - Reserves 100 bytes of uninitialized data space to an offset PRICE. **(v)** **SEGMENT**: - The SEGMENT directive is used to indicate the start of a logical segment. Preceding the SEGMENT directive is the name you want to give the segment. For example, the statement CODE SEGMENT indicates to the assembler the start of a logical segment called CODE. The SEGMENT and ENDS directive are used to "bracket" a logical segment containing code of data. **(vi)** **END**: - An END directive ends the entire program and appears as the last statement. – ENDS directive ends a segment and ENDP directive ends a procedure. END PROC-Name	directives- 1M

	c)	**Explain with suitable example the Instruction given below :** **(i) DAA (ii) AAM**	**6 M**
	Ans	**(i) DAA – Decimal Adjust after BCD Addition:** When two BCD numbers are added, the DAA is used after ADD or ADC instruction to get correct answer in BCD. Syntax- DAA (DAA is Decimal Adjust after BCD Addition) Explanation: This instruction is used to make sure the result of adding two packed BCD numbers is adjusted to be a correct BCD number. The result of the addition must be in AL for DAA instruction to work correctly. If the lower nibble in AL after addition is > 9 or Auxiliary Carry Flag is set, then add 6 to lower nibble of AL. If the upper nibble in AL is > 9H or Carry Flag is set, and then add 6 to upper nibble of AL. Example: - (Any Same Type of Example) AL=99 BCD and BL=99 BCD Then ADD AL, BL 1001 1001 = AL= 99 BCD + 1001 1001 = BL = 99 BCD 0011 0010 = AL =32 H and CF=1, AF=1 After the execution of DAA instruction, the result is CF = 1 0011 0010 =AL =32 H AH =1 + 0110 0110 ------------------------- 1001 1000 =AL =98 in BCD **(ii) AAM - Adjust result of BCD Multiplication:** This instruction is used after the multiplication of two unpacked BCD. The AAM mnemonic stands for ASCII adjust for Multiplication or BCD Adjust after Multiply. This instruction is used in the process of multiplying two ASCII digits. The process begins with masking the upper 4 bits of each digit, leaving an unpacked BCD in each byte. These unpacked BCD digits are then multiplied and the AAM instruction is subsequently used to adjust the product to two unpacked BCD digits in AX. AAM works only after the multiplication of two unpacked BCD bytes, and it works only on an operand in AL. Example Multiply 9 and 5 MOV AL, 00000101 MOV BH, 00001001	Each Instruction-3M

		MUL BH ;Result stored in AX ;AX = 00000000 00101101 = 2DH = 45 in decimals AAM ;AX = 00000100 00000101 = 0405H = 45 in unpacked BCD ; If ASCII values are required an OR operation with 3030H can follow this step.	
6.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Write an appropriate 8086 instruction to perform following operations.** **(i) Rotate the content of BX register towards right by 4 bits.** **(ii) Rotate the content of AX towards left by 2bits.** **(iii) Add 100H to the content of AX register.** **(iv) Transfer 1234H to DX register.** **(v) Multiply AL by 08 H.** **(vi) Signed division of BL and AL**	**6 M**
	Ans	1. Rotate the content of BX register towards right by 4 bits – MOV CL, 04H ROR BX, CL 2. Rotate the content of AX towards left by 2bits – MOV CL, 02H ROL AX, CL 3. Add 100H to the content of AX register – ADD AX,0100H. 4. Transfer 1234H to DX register – MOV DX,1234H 5. Multiply AL by 08H – MOV BL,08h MUL BL 6. Signed division of BL and AL IDIV BL	Each Instruction-1M

	b)	**Explain Addressing modes of 8086 with suitable example.**	**6 M**
	Ans	1. Immediate addressing mode: An instruction in which 8-bit or 16-bit operand (data) is specified in the instruction, then the addressing mode of such instruction is known as immediate addressing mode. Example: MOV AX,67D3H 2. Register addressing mode: An instruction in which an operand (data) is specified in general purpose registers, then the addressing mode is known as register addressing mode. Example: MOV AX, CX 3. Direct addressing mode: An instruction in which 16-bit effective address of an operand is specified in the instruction, then the addressing mode of such instruction is known as direct addressing mode. Example: MOV CL,[2000H] 4. Register Indirect addressing mode: An instruction in which address of an operand is specified in pointer register or in index register or in BX, then the addressing mode is known as register indirect addressing mode. Example: MOV AX,[BX] 5 Indexed addressing mode: An instruction in which the offset address of an operand is stored in index registers (SI or DI) then the addressing mode of such instruction is known as indexed addressing mode. DS is the default segment for SI and DI. For string instructions DS and ES are the default segments for SI and DI resp. this is a special case of register indirect addressing mode. Example: MOV AX,[SI] 6. Based Indexed addressing mode: An instruction in which the address of an operand is obtained by adding the content of base register (BX or BP) to the content of an index register (SI or DI) The default segment register may be DS or ES Example: MOV AX,[BX][SI] 7. Register relative addressing mode: An instruction in which the address of the operand is obtained by adding the displacement (8-bit or 16 bit) with the contents of base registers or index registers (BX, BP, SI, DI). The default segment register is DS or ES.	Each Addressing Mode – 1M

		Example: MOV AX,50H[BX] 8. Relative Based Indexed addressing mode: An instruction in which the address of the operand is obtained by adding the displacement (8 bit or 16 bit) with the base registers (BX or BP) and index registers (SI or DI) to the default segment. Example: MOV AX,50H [BX][SI]	
	c)	**Write an ALP to transfer 10 bytes of data from one memory location to another, also draw the flow chart of the same.**	**6 M**
	Ans	Data Block Transfer Using String Instruction .MODEL SMALL .DATA BLOCK1 DB 01H,02H,03H,04H,05H,06H,07H,08H,09H,0AH BLOCK2 DB 10(?) ENDS .CODE MOV AX, @DATA MOV DS, AX MOV ES, AX LEA SI, BLOCK1 LEA DI, BLOCK2 MOV CX, 000AH ; Initialize counter for 10 data elements CLD REP MOVSB MOV AH, 4CH INT 21H ENDS END	Correct Code-4M, Flowchart-2M


Start
Load DS register with value in AX register
Load ES register with value in AX register
Load SI register with offset address of block 1
Load DI register with offset address of block 2
Initialize counter CX with value of 10
Set the Direction Flag
Repeat Move String (REP MOVSB)
Stop

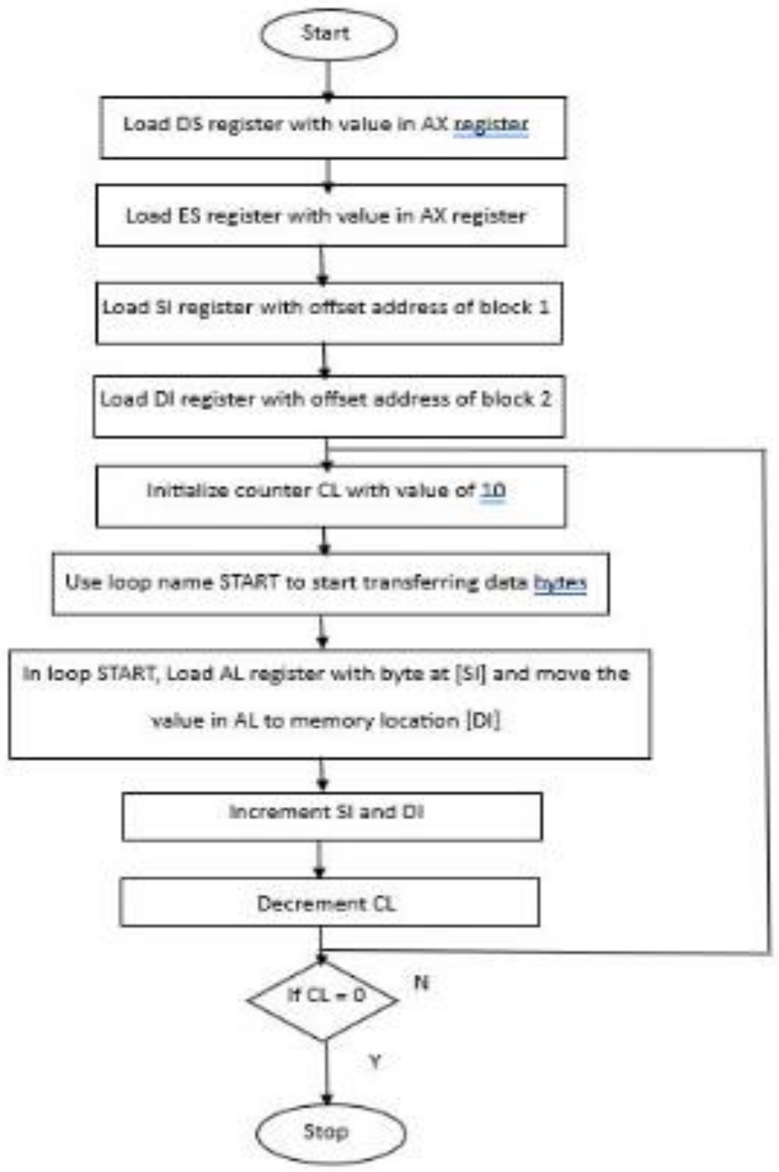
OR

Data Block Transfer Without String Instruction

```
. Model small
. Data
ORG 2000H
Arr1 db 00h,01h,02h,03h,04h,05h,06h,07h,08h,09h
Count Equ 10 Dup
Org 3000H
Arr2 db 10 Dup(00h)
Ends

.code
Start: Mov ax,@data
Mov ds,ax
Mov SI,2000H
Mov DI,3000H
Mov cx, count
Back: Mov al, [SI]
```

		Mov [DI], al Inc SI Inc DI Dec cx Jnc Back Mov ah, 4ch Int 21h Ends End	

		 Start Load DS register with value in AX register Load ES register with value in AX register Load SI register with offset address of block 1 Load DI register with offset address of block 2 Initialize counter CL with value of 10 Use loop name START to start transferring data bytes In loop START, Load AL register with byte at [SI] and move the value in AL to memory location [DI] Increment SI and DI Decrement CL If CL = 0 N Y Stop	