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

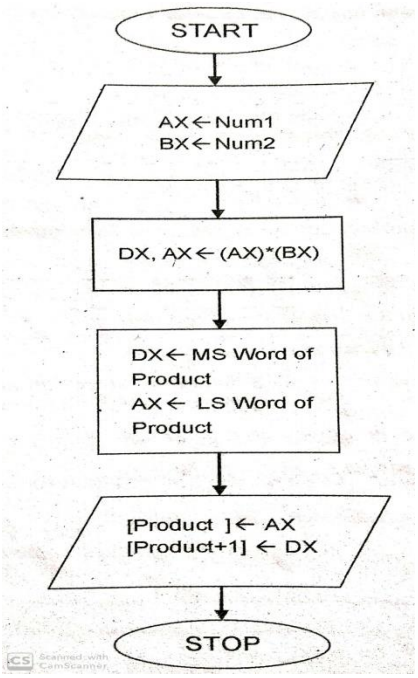
SUMMER – 19 EXAMINATION

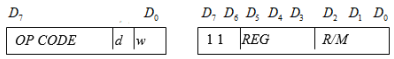
Subject Name: MICROPROCESSOR **Model Answer** **Subject Code: 22415**

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.

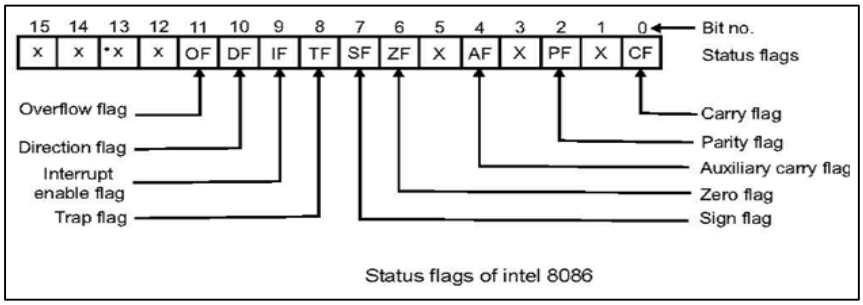
Q. No.	Sub Q. N.	Answer	Marking Scheme
1.		**Attempt any Five of the following:**	**10M**
	a	**State the function of READY and INTR pin of 8086**	**2M**
	Ans	**Ready**: It is used as acknowledgement from slower I/O device or memory. It is Active high signal, when high; it indicates that the peripheral device is ready to transfer data. **INTR** This is a level triggered interrupt request input, checked during last clock cycle of each instruction to determine the availability of request. If any interrupt request is occurred, the processor enters the interrupt acknowledge cycle.	Each correct function 1M
	b	**What is role of XCHG instruction in assembly language program? Give example**	**2M**
	Ans	**Role of XCHG:** This instruction exchanges the contents of a register with the contents of another register or memory location. **Example:** XCHG AX, BX ; Exchange the word in AX with word in BX.	Correct role:1M Correct example : 1M

			(any other example allowed)
	c	**List assembly language programming tools.**	**2M**
	Ans	1. Editors 2. Assembler 3. Linker 4. Debugger.	Each ½ M
	d	**Define Macro.Give syntax.**	**2M**
	Ans	**Macro:** Small sequence of the codes of the same pattern are repeated frequently at different places which perform the same operation on the different data of same data type, such repeated code can be written separately called as Macro. **Syntax:** Macro_name MACRO[arg1,arg2,…..argN) ….. End	Definition1M Syntax 1M
	e	**Draw flowchart for multiplication of two 16 bit numbers.**	**2M**
	Ans		Correct flowchart: 2M(consider any relevant flowchart also)
	f	**Draw machine language instruction format for Register-to-Register transfer.**	**2M**

	Ans		Correct diagram 2M
	g	**State the use of STC and CMC instruction of 8086.**	**2M**
	Ans	STC – This instruction is used to Set Carry Flag. CF←1 CMC – This instruction is used to Complement Carry Flag. CF← ~ CF	Each correct use 1M
2.		**Attempt any Three of the following:**	**12M**
	a	**Give the difference between intersegment and intrasegment CALL**	**4M**
	Ans		Any 4 points 1M each

Sr.no	Intersegment Call	Intrasegment Call
1.	It is also called Far procedure call	It is also called Near procedure call.
2.	A far procedure refers to a procedure which is in the different code segment from that of the call instruction.	A near procedure refers to a procedure which is in the same code segment from that of the call instruction
3	This procedure call replaces the old CS:IP pairs with new CS:IP pairs	This procedure call replaces the old IP with new IP.
4.	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)	The value of old IP is pushed on to the stack. SP=SP-2 ;Save IP on stack(address of procedure)
5.	More stack locations are required	Less stack locations are required

		6.	Example :- Call FAR PTR Delay	Example :- Call Delay

	b	**Draw flag register of 8086 and explain any four flags.**	**4M**
	Ans	**Flag Register of 8086**  Status flags of intel 8086 **Conditional /Status Flags** **C-Carry Flag** : It is set when carry/borrow is generated out of MSB of result. (i.e D_7 bit for 8-bit operation, D_{15} bit for a 16 bit operation). **P-Parity Flag** This flag is set to 1 if the lower byte of the result contains even number of 1's otherwise it is reset. **AC-Auxiliary Carry Flag** This is set if a carry is generated out of the lower nibble, (i.e. From D3 to D4 bit)to the higher nibble **Z-Zero Flag** This flag is set if the result is zero after performing ALU operations. Otherwise it is reset. **S-Sign Flag** This flag is set if the MSB of the result is equal to 1 after performing ALU operation , otherwise it is reset. **O-Overflow Flag** This flag is set if an overflow occurs, i.e. if the result of a signed operation is large enough to be accommodated in destination register. **Control Flags** **T-Trap Flag** If this flag is set ,the processor enters the single step execution mode. **I-Interrupt Flag** it is used to mask(disable) or unmask(enable)the INTR interrupt. When this flag is set,8086 recognizes interrupt INTR. When it is reset INTR is masked.	Correct diagram 2M Any 4 flag explanation :1/2 M each

		D-Direction Flag It selects either increment or decrement mode for DI &/or SI register during string instructions.	
	c	**Explain assembly language program development steps.**	**4M**
	Ans	**1. Defining the problem:** The first step in writing program is to think very carefully about the problem that the program must solve. **2. Algorithm:** The formula or sequence of operations to be performed by the program can be specified as a step in general English is called algorithm. **3. Flowchart:** The flowchart is a graphically representation of the program operation or task. **4. Initialization checklist:** Initialization task is to make the checklist of entire variables, constants, all the registers, flags and programmable ports **5. Choosing instructions**: Choose those instructions that make program smaller in size and more importantly efficient in execution. **6. Converting algorithms to assembly language program:** Every step in the algorithm is converted into program statement using correct and efficient instructions or group of instructions.	Correct steps 4M
	d	**Explain logical instructions of 8086.(Any Four)**	**4M**
	Ans	**Logical instructions.** **1) AND- Logical AND** **Syntax : AND destination, source** **Operation** **Destination ←destination AND source** **Flags Affected :CF=0,OF=0,PF,SF,ZF** This instruction AND's each bit in a source byte or word with the same number bit in a destination byte or word. The result is put in destination. **Example: AND AX, BX** • **AND AL,BL** • **AL 1111 1100** • **BL 0000 0011** **----------** • **AL←0000 0000 (AND AL,BL)** **2) OR – Logical OR** Syntax **:OR destination, source**	Any 4 instruction correct explanation 1M each

Operation

Destination ← OR source

Flags Affected :CF=0,OF=0,PF,SF,ZF

This instruction OR's each bit in a source byte or word with the corresponding bit in a destination byte or word. The result is put in a specified destination.

Example :

- OR AL,BL
- AL 1111 1100
- BL 0000 0011

- AL←1111 1111

3) NOT – Logical Invert

Syntax : NOT destination

Operation: Destination← NOT destination

Flags Affected :None

The NOT instruction inverts each bit of the byte or words at the specified destination.

Example

NOT BL

BL = 0000 0011

NOT BL gives 1111 1100

4) XOR – Logical Exclusive OR

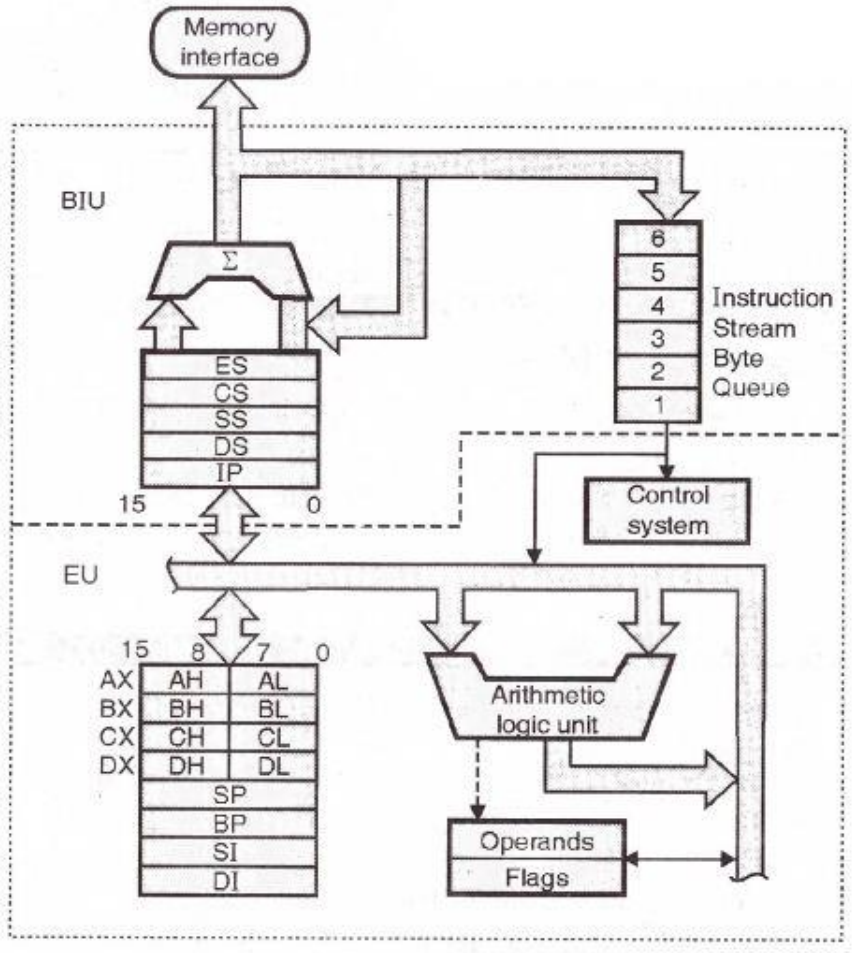
Syntax : **XOR destination, source**

Operation : **Destination Destination XOR source**

Flags Affected :CF=0,OF=0,PF,SF,ZF

This instruction exclusive, OR's each bit in a source byte or word with the same number bit in a destination byte or word.

		Example(optional) **XOR AL,BL** • AL 1111 1100 • BL 0000 0011 ---------- • **AL←1111 1111 (XOR AL,BL)** **5)TEST** **Syntax : TEST Destination, Source** This instruction AND's the contents of a source byte or word with the contents of specified destination byte or word and flags are updated, , flags are updated as result ,but neither operands are changed. **Operation performed:** Flags ⟵ set for result of (destination AND source) **Example: (Any 1)** TEST AL, BL ; AND byte in BL with byte in AL, no result, Update PF, SF, ZF. e.g **MOV AL, 00000101** **TEST AL, 1 ; ZF = 0.** **TEST AL, 10b ; ZF = 1**	
3.		**Attempt any Four of the following:**	
	a	**Draw functional block diagram of 8086 microprocessor.**	**4M**
	Ans		Block diagram 4M



8086 internal architecture

	b	**Write an ALP to add two 16-bit numbers.**	**4M**
	Ans	DATA SEGMENT NUMBER1 DW 6753H NUMBER2 DW 5856H SUM DW 0 DATA ENDS CODE SEGMENT ASSUME CS: CODE, DS: DATA START: MOV AX, DATA	Data segment initialization 1M, Code segment 3M

		MOV DS, AX
		MOV AX, NUMBER1
		MOV BX, NUMBER2
		ADD AX, BX
		MOV SUM, AX
		MOV AH, 4CH
		INT 21H
		CODE ENDS
		END START	

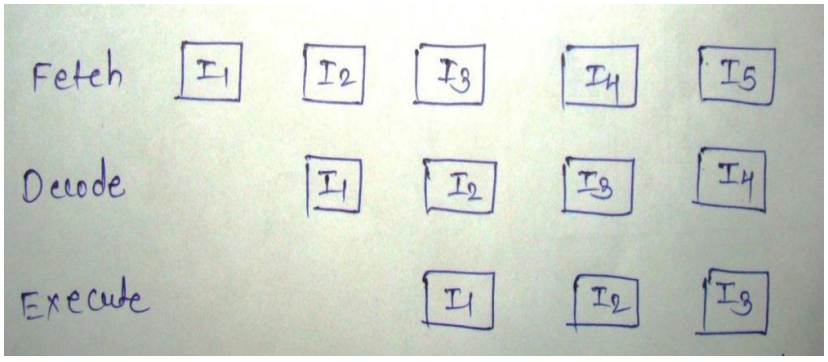
	c	**Write an ALP to find length of string.**	**4M**

	Ans	Data Segment
		STRG DB 'GOOD MORNING$'
		LEN DB ?
		DATA ENDS
		CODE SEGMENT
		START:
		ASSUME CS: CODE, DS : DATA
		MOV DX, DATA
		MOV DS,DX
		LEA SI, STRG
		MOV CL,00H
		MOV AL,'$'
		NEXT: CMP AL,[SI]
		JZ EXIT
		ADD CL,01H
		INC SI
			program - 4 M

		JMP NEXT EXIT: MOV LEN,CL MOV AH,4CH INT 21H CODE ENDS	
	d	**Write an assembly language program to solve p= x^2+y^2 using Macro.(x and y are 8 bit numbers.**	**4M**

	Ans	.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()
		.CODE
		START:
		MOV ax,data
		MOV ds,ax
		PROG x, y
			program - 4 M

		MOV p,al
		MOV ah,4Ch
		Int 21H
		END	

4.		**Attempt any Three of the following:**	
	a	**What is pipelining? How it improves the processing speed.**	
	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
	b	**Write an ALP to count no.of 0's in 16 bit number.**	**4M**
	Ans	DATA SEGMENT N DB 1237H Z DB 0	Program 4 M

		DATA ENDS
		CODE SEGMENT
		ASSUME DS:DATA, CS:CODE
		START:
		MOV DX,DATA
		MOV DS,DX
		MOV AX, N
		MOV CL,08
		NEXT: ROL AX,01
		JC ONE
		INC Z
		ONE: LOOP NEXT
		HLT
		CODE ENDS
		END START	

	c	**Write an ALP to find largest number in array of elements 10H, 24H, 02H, 05H, 17H.**	**4M**

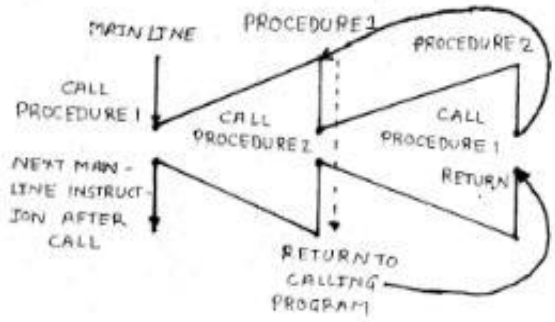
	Ans	DATA SEGMENT
		ARRAY DB 10H,24H,02H,05H,17H
		LARGEST DB 00H
		DATA ENDS
		CODE SEGMENT
		START:
		ASSUME CS:CODE,DS:DATA
		MOV DX,DATA
		MOV DS,DX
		MOV CX,04H
		MOV SI ,OFFSET
		ARRAY MOV AL,[SI]
		UP: INC SI
		CMP AL,[SI]
		JNC NEXT
		MOV AL,[SI]
		NEXT: DEC CX
		JNZ UP
		MOV LARGEST,AL
		MOV AX,4C00H
		INT 21H
		CODE ENDS
		END START
			Program - 4 M

	d	**Write an ALP for addition of series of 8-bit number using procedure.**	**4M**
	Ans	**DATA SEGMENT** NUM1 DB 10H,20H,30H,40H,50H RESULT DB 0H CARRY DB 0H	Program - 4 M

		DATA ENDS
		CODE SEGMENT
		ASSUME CS:CODE, DS:DATA
		START: MOV DX,DATA
		MOV DS, DX
		MOV CL,05H
		MOV SI, OFFSET NUM1
		UP: CALL SUM
		INC SI
		LOOP UP
		MOV AH,4CH
		INT 21H
		
		SUM PROC; Procedure to add two 8 bit numbers
		MOV AL,[SI]
		ADD RESULT, AL
		JNC NEXT
		INC CARRY
		NEXT: RET
		SUM ENDP
		CODE ENDS
		END START	

	e	**Describe re-entrant and recursive procedure with schematic diagram.**	**4M**
	Ans	In some situation it may happen that Procedure 1is called from main program Procrdure2 is called from procedure1And procrdure1 is again called from procdure2. In this situation program execution flow reenters in the procedure1. These types of procedures are called re enterant procedures. The RET instruction at the end of procrdure1 returns to procedure2. The RET instruction at the end of procedure2 will return the execution to procedure1.Procedure1 will again executed from where it had stopped at the time of calling procrdure2 and the RET instruction at the end of this will return the program execution to main program. The flow of program execution for re-entrant procedure is as shown in FIG.	**Re-entrant 2 M, recursive 2 M**

Sketch :



Recursive Procedure

A recursive procedure is a procedure which calls itself. Recursive procedures are used to work with complex data structures called trees. If the procedures is called with N (recursion depth) = 3. Then the n is decremented by one after each procedure CALL and the procedure is called until n = 0. Fig. shows the flow diagram and pseudo-code for recursive procedure.

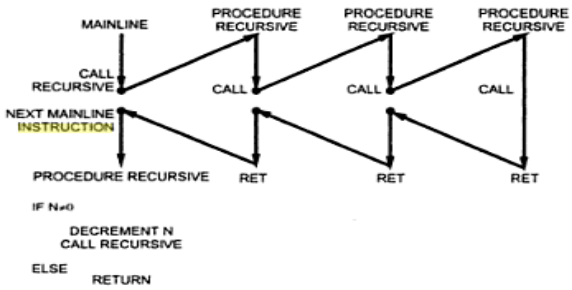
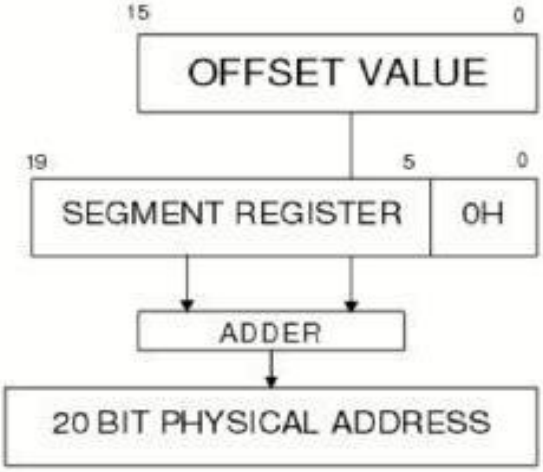


Fig. Flow diagram and pseudo-code for recursive procedure

5.		**Attempt any Two of the following:**	**12 M**
	a	**Define logical and effective address. Describe physical address generation process in 8086. If DS=345AH and SI=13DCH. Calculate physical address.**	**6M**
	Ans	**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. **Effective Address or Offset Address**: 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.	Define each Term :1M. Physical Address Generation. Description : 2 M & Calculation 2 M

		Generation of 20 bit physical address in 8086:- 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 bit offset. 4. Offset address is added into 20-bit base address which finally forms 20 bit physical address of memory location  DS=345AH and SI=13DCH Physical adress = DS*10H + SI = 345AH * 10H + 13DCH = 345A0+13DC = 3597CH	
	b	**Explain the use of assembler directives. 1) DW 2) EQU 3) ASSUME 4) OFFSET 5) SEGMENT 6) EVEN**	**2M**
	Ans	**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. **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.	Each Directive Use : 1M each

		Example **Data SEGMENT** **Num1 EQU 50H** **Num2 EQU 66H** **Data ENDS** Numeric value 50H and 66H are assigned to Num1 and Num2. **ASSUME** ASSUME tells the assembler what names have been chosen for Code, Data Extra and Stack segments. Informs the assembler that the register CS is to be initialized with the address allotted by the loader to the label CODE and DS is similarly initialized with the address of label DATA. **OFFSET** OFFSET is an operator, which tells the assembler to determine the offset or displacement of a named data item (variable), a procedure from the start of the segment, which contains it. **Example** **MOV BX;** **OFFSET PRICES;** It will determine the offset of the variable PRICES from the start of the segment in which PRICES is defined and will load this value into BX. **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 **EVEN (ALIGN ON EVEN MEMORY ADDRESS)** As an assembler assembles a section of data declaration or instruction statements, it uses a location counter to keep track of how many bytes it is from the start of a segment at any time. The EVEN directive tells the assembler to increment the location counter to the next even address, if it is not already at an even address. A NOP instruction will be inserted in the location incremented over.	
	c	**Describe any four string instructions of 8086 assembly language.**	**2M**
	Ans	**1] REP:** REP is a prefix which is written before one of the string instructions. It will cause During length counter CX to be decremented and the string instruction to be repeated until CX becomes 0.	each correct instruction 1½ M each

Two more prefix.

REPE/REPZ: Repeat if Equal /Repeat if Zero.

It will cause string instructions to be repeated as long as the compared bytes or words Are equal and CX≠0.

REPNE/REPNZ: Repeat if not equal/Repeat if not zero.

It repeats the strings instructions as long as compared bytes or words are not equal

And CX≠0.

Example: REP MOVSB

2] MOVS/ MOVSB/ MOVSW - Move String byte or word.

Syntax:

MOVS destination, source

MOVSB destination, source

MOVSW destination, source

Operation: ES:[DI]<----- DS:[SI]

It copies a byte or word a location in data segment to a location in extra segment. The offset of source is pointed by SI and offset of destination is pointed by DI.CX register contain counter and direction flag (DE) will be set or reset to auto increment or auto decrement pointers after one move.

Example

LEA SI, Source

LEA DI, destination

CLD

MOV CX, 04H

REP MOVSB

3] CMPS /CMPSB/CMPSW: Compare string byte or Words.

Syntax:

CMPS destination, source

CMPSB destination, source

CMPSW destination, source

Operation: Flags affected < ----- DS:[SI]- ES:[DI]

It compares a byte or word in one string with a byte or word in another string. SI Holds the offset of source and DI holds offset of destination strings. CS contains counter and DF=0 or 1 to auto increment or auto decrement pointer after comparing one byte/word.

Example

LEA SI, Source

LEA DI, destination

CLD

MOV CX, 100

REPE CMPSB

4] SCAS/SCASB/SCASW: Scan a string byte or word.

Syntax:

SCAS/SCASB/SCASW

Operation: Flags affected < ----- AL/AX-ES: [DI]

It compares a byte or word in AL/AX with a byte /word pointed by ES: DI. The string to be scanned must be in the extra segment and pointed by DI. CX contains counter and DF may be 0 or 1.

When the match is found in the string execution stops and ZF=1 otherwise ZF=0.

Example

LEA DI, destination

MOV Al, 0DH

MOV CX, 80H

CLD

REPNE SCASB

		5] LODS/LODSB/LODSW: Load String byte into AL or Load String word into AX. Syntax: LODS/LODSB/LODSW Operation: AL/AX < ----- DS: [SI] IT copies a byte or word from string pointed by SI in data segment into AL or AX.CX may contain the counter and DF may be either 0 or 1 **Example** LEA SI, destination CLD LODSB **6] STOS/STOSB/STOSW (Store Byte or Word in AL/AX)** Syntax STOS/STOSB/STOSW Operation: ES:[DI] < ----- AL/AX It copies a byte or word from AL or AX to a memory location pointed by DI in extra segment CX may contain the counter and DF may either set or reset	
6.		**Attempt any Two of the following:**	**12M**
	a	**Describe any 6 addressing modes of 8086 with one example each.**	**6M**
	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.	Any 6 mode with example 1 M each

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. **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]	
	b	**Select assembly language for each of the following** **i) rotate register BL right 4 times** **ii) multiply AL by 04H** **iii) Signed division of AX by BL** **iv) Move 2000h in BX register** **v) increment the counter of AX by 1** **vi) compare AX with BX**	**6M**
	Ans	i) MOV CL, 04H RCL AX, CL1 Or MOV CL, 04H ROL AX, CL Or MOV CL, 04H RCR AX, CL1	Each correct instruction 1M

		Or MOV CL, 04H ROR AX, CL ii) MOV BL,04h MUL BL iii) IDIV BL iv) MOV BX,2000h v) INC AX vi) CMP AX,BX	
	c	**Write an ALP to reverse a string. Also draw flowchart for same.**	

	Ans	**Program:**
		DATA SEGMENT
		STRB DB 'GOOD MORNING$'
		REV DB 0FH DUP(?)
		DATA ENDS
		CODE SEGMENT
		START:ASSUME CS:CODE,DS:DATA
		MOV DX,DATA
		MOV DS,DX
		LEA SI,STRB
		MOV CL,0FH
		LEA DI,REV
		ADD DI,0FH
		UP:MOV AL,[SI]
			Program 4 M
			flowchart 2 M

```
MOV [DI],AL
INC SI
DEC DI
LOOP UP
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Flowchart: