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

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

Subject Name: Microprocessor **Subject Code:** 22415

Important Instructions to examiners:

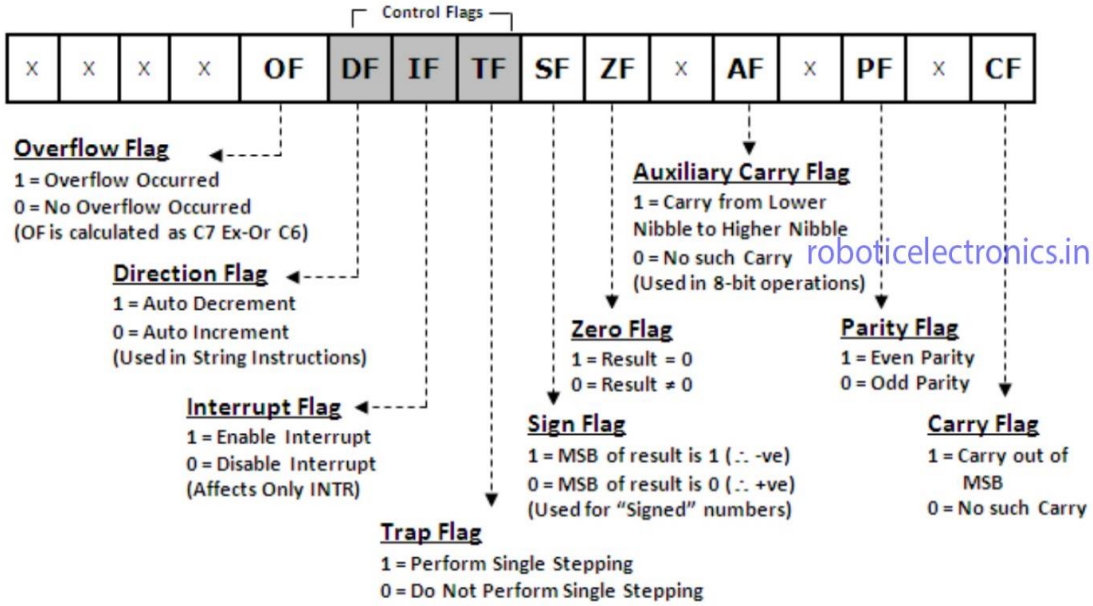
1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling 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 use of MN/ MX and Test signal.**	**2 M**
	Ans	MN/MX: - Minimum/Maximum: - This pin signal indicates what mode the processor will operate in. MN/MX = 1 = 8086 operates in minimum mode. In this mode the 8086 is configured to support small single processor system using a few devices that the system bus. MN/MX = 0 = 8086 is configured to support multiprocessor system. Test: - It is an input pin and is only used by the wait instruction. The 8086 enter a wait state after execution of the wait instruction until a low is seen on the test pin. If the TEST pin is Low, execution continues otherwise the processor waits in an "idle" state. This input is synchronized internally during each clock cycle on the leading edge of CLK.	1 M for each signal
	b)	**List Assembly Language Programming tools.**	**2 M**
	Ans	1. Editors 2. Assembler 3. Linker	1/2 M for each tool

		4. Debugger.	
	c)	**Write any four-bit manipulation instructions of 8086.**	**2 M**
	Ans	**NOT** − Used to invert each bit of a byte or word. **AND** − Used for adding each bit in a byte/word with the corresponding bit in another byte/word. **OR** − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word. **XOR** − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.	1/2 M for each tool
	d)	**What is the use of AAM instruction with suitable example?**	**2 M**

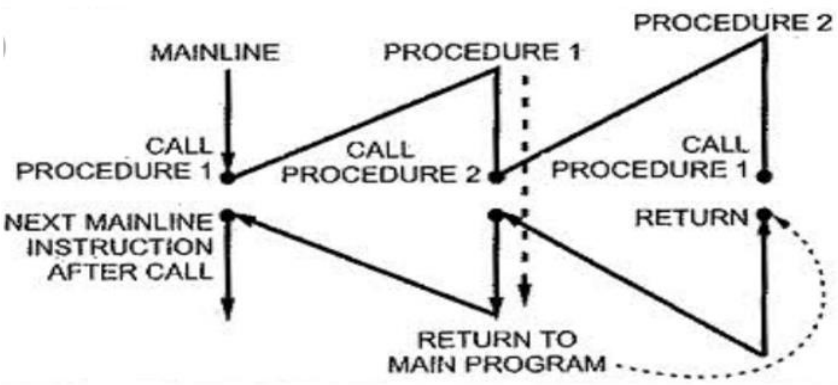
	Ans	The AAM instruction, short for "ASCII Adjust AX After Multiply," is an assembly language instruction in the x86 architecture used for converting the binary result of multiplying two unpacked Binary-Coded Decimal (BCD) values back into a valid unpacked BCD format.
		
		Example:
		
		Let's say we want to multiply the BCD numbers 5 (0101) and 7 (0011) using the x86 architecture.
		
		1. Multiplication:
		MOV AL, 5; Store 5 in AL
		MOV BL, 7; Store 7 in BL
		MUL BL; Multiply AL and BL, result stored in AX
		
		2. Result in AX:
		AH: 0 (carry from overflow)
		AL: 5 (product of lower digits)
		
		3. AAM instruction:
		AAM
		
		4. Adjusted result in AX:
		AH: 1 (quotient from dividing AL by 10)
		AL: 5 (remainder from dividing AL by 10)
		Therefore, the final result of multiplying 5 and 7 in unpacked BCD format is 35 (0011 0101).
			1 M for use
			
			1 M for example

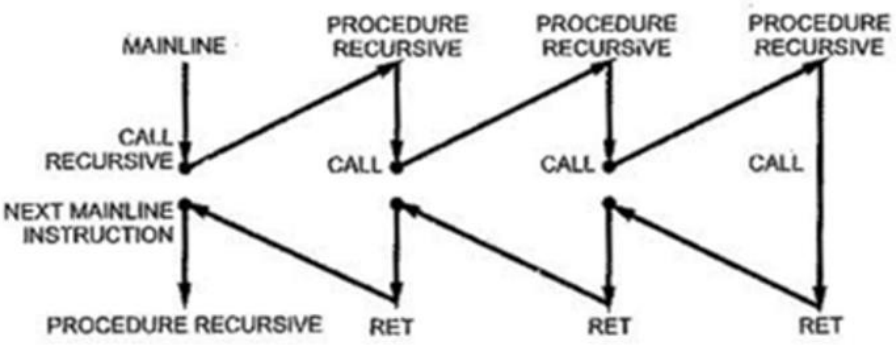
	e)	**Give any two advantages of pipelining in 8086.**	**2 M**
	Ans	1. Increased Instruction Throughput: 2. Improved Efficiency of the Execution Unit (EU)	1 M for each
	f)	**Draw the format of flag register of 8O86.**	**2 M**

	Ans		2 M for format
	g)	**Define procedure and write its syntax**	**2 M**
	Ans	Procedure: A procedure is group of instructions that usually performs one task. It is a reusable section of a software program which is stored in memory once but can be used as often as necessary. A procedure can be of two types. 1) Near Procedure 2) Far Procedure Syntax :- Procedure can be defined as Procedure_name PROC ---- ------ Procedure_name ENDP For Example Addition PROC near ------ Addition ENDP	1 M for Definition 1 M for syntax.
2.		**Attempt any <u>THREE</u> of the following:**	**12 M**

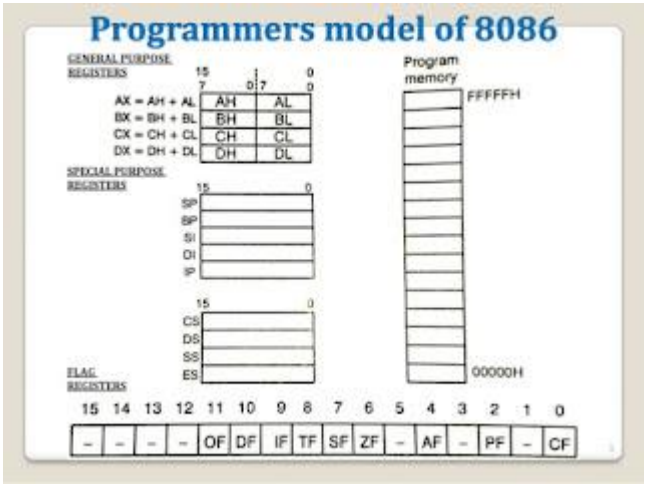
	a)	**Describe the function of the following instructions: (i) DAA (ii) CMP (iii) ADC (iv) JNC**	**4 M**

	Ans	**<u>1) DAA: Decimal adjust after addition</u>**
		This instruction is used to make sure the result of adding two packed BCD numbers is adjusted to be a legal BCD number. The result of the addition must be in AL for DAA to work correctly. If the lower nibble in AL after an addition is greater than 9 or AF was set by the addition, then the DAA instruction will add 6 to the lower nibble in AL. If the result in the upper nibble of AL in now greater than 9 or if the carry flag was set by the addition or correction, then the DAA instruction will add 60H to AL.
		
		Let AL = 59 BCD, and BL = 35 BCD
		ADD AL, BL AL = 8EH; lower nibble > 9, add 06H to AL
		DAA AL = 94 BCD, CF = 0
		
		Let AL = 88 BCD, and BL = 49 BCD
		ADD AL, BL AL = D1H; AF = 1, add 06H to AL
		DAA AL = D7H; upper nibble > 9, add 60H to AL
		AL = 37 BCD, CF = 1
		
		<u>2) CMP: Compare</u>
		This instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location.
		Example: -
		CMP BX, 0100H
		CMP AX, 0100H
		CMP [5000H], 0100H
		CMP BX, [SI]
		CMP BX, CX
		
		<u>3) ADC: Add with Carry</u>
		ADC Destination, Source
		This instruction performs the same operation as ADD instruction, but adds the carry flag to the result.
		Example: -
		ADC 0100H
		ADC AX, BX
		ADC AX, [SI]
		ADC AX, [5000]
		ADC [5000], 0100H
		4) JNC: - Stands for 'Jump if Not Carry'
			1 M for each instruction
			
			(example is not mandatory)

		It checks whether the carry flag is reset or not. If yes, then jump takes place, that is: If CF = 0, then jump. ADD AL, BL Add two bytes JNC NEXT If the result within acceptable range, continue **4) JNC :** Stands for 'Jump if Not Carry' It checks whether the carry flag is reset or not. If yes, then jump takes place, that is: If CF = 0, then jump.	
	b)	**Explain Re-Entrant and Recursive Procedure with diagram.**	**4 M**
	Ans	**<u>1) Re-Entrant Procedure:</u>** The re-entrant procedure is a very special kind of procedure. In such kind of procedure, procedure 1 is called the mainline program, then procedure 2 is called form procedure 1 and then again procedure 1 is called form procedure 2. This can be well understood from the following diagram  This is called a re-entrant procedure because a procedure is re-entering into itself form another procedure which is also present inside its own body. The re-entrant procedure occurs in the following three conditions: when the procedure is undergoing recursion, when multi-threading is being implemented inside a program or when some interruption is being generated. Like the recursive procedures, it is important to have a termination condition for the procedures in the re-entrant procedures also, else we can face machine halts due to infinite procedure calls. **<u>2) Recursive procedures:</u>** A recursive procedure is a procedure which calls itself. This results in the procedure call to be generated from within the procedures again and again. This can be understood as follows:	1 M for explanation 1M for diagram For each

		 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.	
	c)	**Write the function of following pins of 8086:** (i) Ready (ii) ALE iii) $\overline{TEST}$ iv)$\overline{DEN}$	**4 M**
	Ans	(i) Ready: - This is an acknowledgment signal from the slower I/O devices or memory. When high, it indicates that the device is ready to transfer data, else the microprocessor is in the wait state. (ii) ALE:- Address Latch Enable. ALE is provided by the microprocessor to latch the address into the 8282 or 8283 address latch. It is an active high (1) pulse during T1 of any bus cycle. ALE signal is never floated, is always integer. iii) $\overline{TEST}$ - This is an acknowledgment signal from the slower I/O devices or memory. When high, it indicates that the device is ready to transfer data, else the microprocessor is in the wait state. iv)$\overline{DEN}$: Data enable. This pin is provided as an output enable for the 8286/8287 in a minimum system which uses transceiver. DEN is active low (0) during each memory and input-output access and for INTA cycles.	**1 M for each**
	d)	**Draw and explain model of Assembly Language Programming.**	**4 M**
	Ans	The programming model for a microprocessor shows the various internal registers that are accessible to the programmer.	2 M for Diagram

		The Following Figure is a model for the 8086. In general, each register has a special function.	2 M for explanation



In the programming model there are

- 4 General Purpose registers(Data Registers)
- 4 Segment registers
- 2 Pointer registers
- 2 Index registers
- 1 Instruction Pointer register
- 1 Flag register

General purpose registers:

AX Register (Accumulator): This is accumulator register. It gets used in arithmetic, logic and data transfer instructions. In manipulation and division, one of the numbers involved must be in AX or AL.

BX Register (Base Register): This is base register. BX register is an address register. It usually contain a data pointer used for based, based indexed or register indirect addressing.

CX Register (Counter register): This is Count register. This serves as a loop counter. Program loop constructions are facilitated by it. Count register can also be used as a counter in string manipulation and shift/rotate instruction.

DX Register (Data Register): This is data register. Data register can be used as a port number in I/O operations. It is also used in multiplication and division.

Segment Registers:

There are four segment registers in Intel 8086:

1. Code Segment Register (CS),

2. Data Segment Register (DS),

3. Stack Segment Register (SS),

4. Extra Segment Register (ES).

A segment register points to the starting address of a memory segment. Maximum capacity of a segment may be up to 64 KB.

Code segment Register (CS):- It is a 16-bit register containing the starting address of 64 KB segment. The processor uses CS segment for all accesses to instructions referenced by instruction pointer (IP) register.

Stack segment Register (SS):- It is a 16-bit register containing address of 64KB segment with program stack. By default, the processor assumes that all data referenced by the stack pointer (SP) and base pointer (BP) registers is located in the stack segment. SS register can be changed directly using POP instruction.

Data segment Register (DS):- It is a 16-bit register containing address of 64KB segment with program data. By default, the processor assumes that all data referenced by general registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment.

Extra segment Register (ES):- It is a 16-bit register containing address of 64KB segment, usually with program data. By default, the processor assumes that the DI register references the ES segment in string manipulation instructions. It is possible to change default segments used by general and index registers by prefixing instructions with a CS, SS,DS or ES prefix.

Pointer Registers:

SP Register (Stack Pointer): This is stack pointer register pointing to program stack. It is used in conjunction with SS for accessing the stack segment.

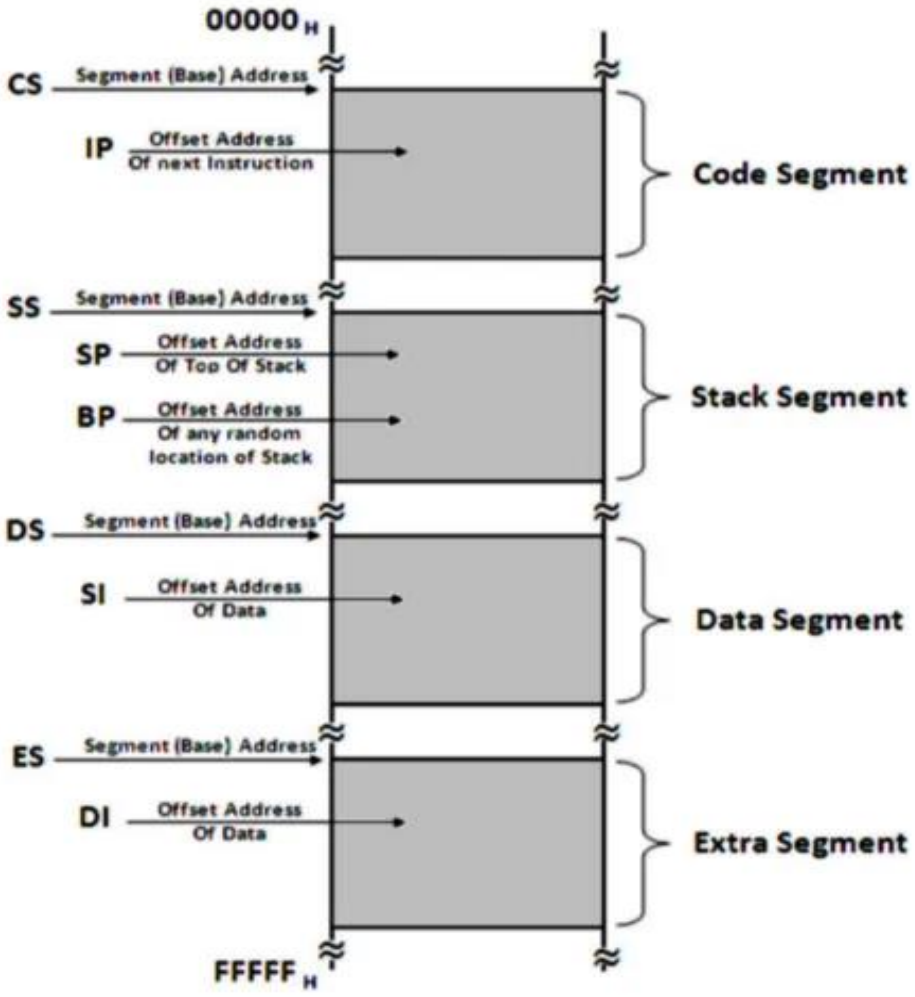
BP Register (Base Pointer): This is base pointer register pointing to data in stack segment. Unlike SP, we can use BP to access data in the other segments.

Index Registers:

SI Register (Source Index): This is used to point to memory locations in the data segment addressed by DS. By incrementing the contents of SI one can easily access consecutive memory locations.

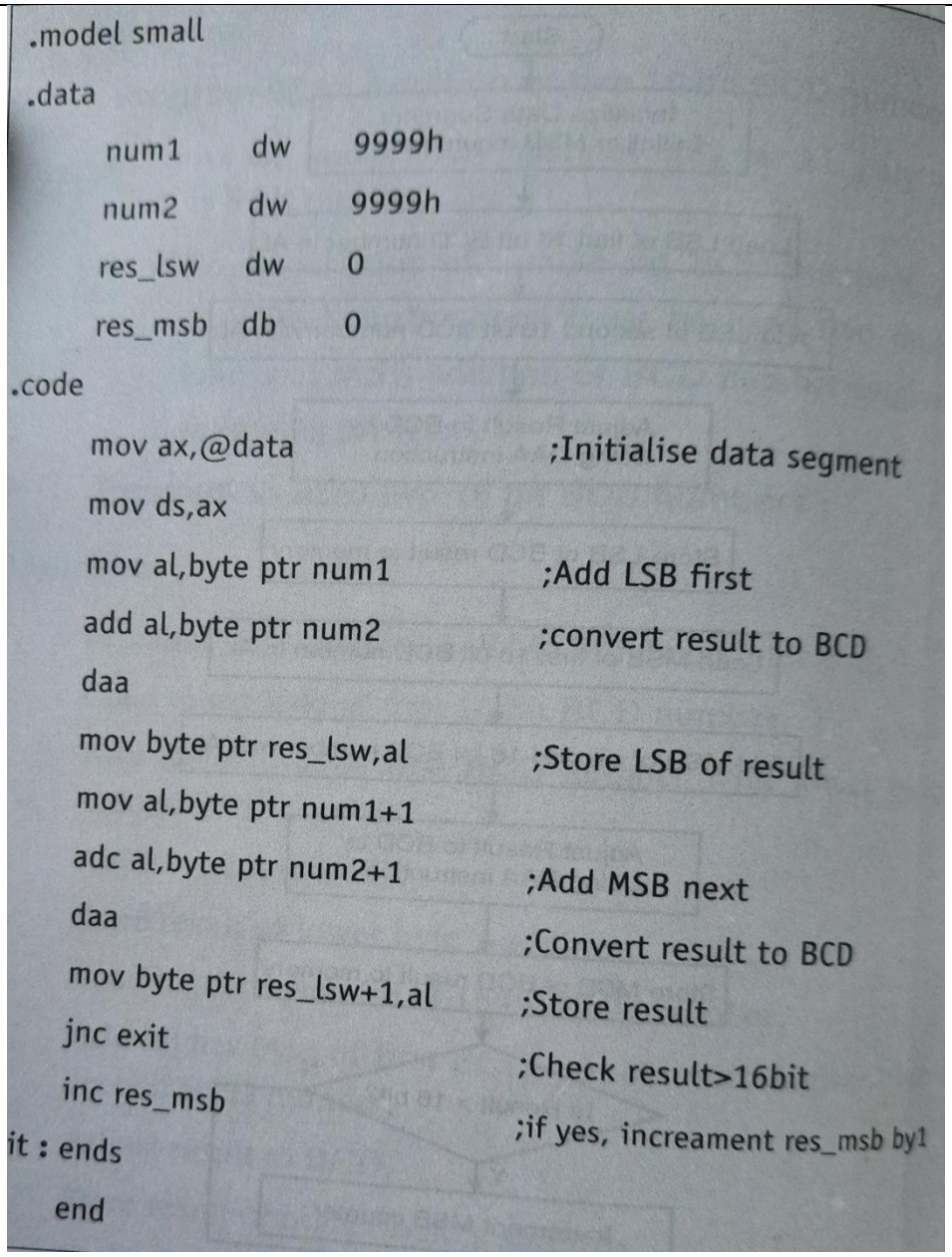
		DI Register (Destination Index): This register performs the same function as SI. There is a class of instructions called string operations, that use DI to access the memory locations addressed by ES. **Instruction Pointer:** The Instruction Pointer (IP) points to the address of the next instruction to be executed. Its content is automatically incremented when the execution of a program proceeds further. The contents of the IP and Code Segment Register are used to compute the memory address of the instruction code to be fetched. This is done during the Fetch Cycle. **Flag Register:** Status Flags determines the current state of the accumulator. They are modified automatically by CPU after mathematical operations. This allows to determine the type of the result. 8086 has 16-bit status register. It is also called Flag Register or Program Status Word (PSW). There are nine status flags and seven bit positions remain unused. 8086 has 16 flag registers among which 9 are active. The purpose of the FLAGS register is to indicate the status of the processor. It does this by setting the individual bits called flags. There are two kinds of FLAGS; Status FLAGS and Control FLAGS. Status FLAGS reflect the result of an operation executed by the processor. The control FLAGS enable or disable certain operations of the processor.	
3.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Describe memory segmentation in 8086 and list its advantages.**	**4 M**
	Ans	**Segmentation** is the process in which the main memory of the computer is logically divided into different segments and each segment has its own base address. It is basically used to enhance the speed of execution of the computer system, so that the processor is able to fetch and execute the data from the memory easily and fast. **Need for Segmentation –** The Bus Interface Unit (BIU) contains four 16 bit special purpose registers (mentioned below) called as Segment Registers. • **Code segment register (CS):** is used for addressing memory location in the code segment of the memory, where the executable program is stored. • **Data segment register (DS):** points to the data segment of the memory where the data is stored. • **Extra Segment Register (ES):** also refers to a segment in the memory which is another data segment in the memory. • **Stack Segment Register (SS):** is used for addressing stack segment of the memory. The stack segment is that segment of memory which is used to store stack data.	1 M for explanation 2 M for diagram 1 M for Advantages

The number of address lines in 8086 is 20, 8086 BIU will send 20bit address, so as to access one of the 1MB memory locations. The four segment registers actually contain the upper 16 bits of the starting addresses of the four memory segments of 64 KB each with which the 8086 is working at that instant of time. A segment is a logical unit of memory that may be up to 64 kilobytes long. Each segment is made up of contiguous memory locations. It is an independent, separately addressable unit. Starting address will always be changing. It will not be fixed.



Advantages of the Segmentation The main advantages of segmentation are as follows:

- It provides a powerful memory management mechanism.
- Data related or stack related operations can be performed in different segments.
- Code related operation can be done in separate code segments.
- It allows to processes to easily share data.
- It allows to extend the address ability of the processor, i.e. segmentation allows the use of 16 bit registers to give an addressing capability of 1 Megabytes. Without segmentation, it would require 20 bit registers.
- It is possible to enhance the memory size of code data or stack segments beyond 64 KB by allotting more than one segment for each area.

	b)	**Write an ALP to perform addition of two 16 bit BCD numbers,**	**4 M**
	Ans		4 Mark for program

```
.model small
.data
        num1    dw      9999h
        num2    dw      9999h
        res_lsw dw      0
        res_msb db      0
.code
        mov ax,@data                ;Initialise data segment
        mov ds,ax
        mov al,byte ptr num1        ;Add LSB first
        add al,byte ptr num2        ;convert result to BCD
        daa
        mov byte ptr res_lsw,al     ;Store LSB of result
        mov al,byte ptr num1+1
        adc al,byte ptr num2+1      ;Add MSB next
        daa                         ;Convert result to BCD
        mov byte ptr res_lsw+1,al   ;Store result
        jnc exit                    ;Check result>16bit
        inc res_msb                 ;if yes, increament res_msb by1
it : ends
    end
```

	c)	**Write an ALP to find largest number in array of 5 elements.**	**4 M**

	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
			4 M for program

		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	

	d)	**Describe CALL and RET instructions with example.**	**4 M**
	Ans	**1. CALL : Unconditional Call** The CALL instruction is used to transfer execution to a subprogram or procedure by storing return address on stack There are two types of calls- NEAR (Inter-segment) and FAR(Intra-segment call). Near call refers to a procedure call which is in the same code segment as the call instruction and far call refers to a procedure call which is in different code segment from that of the call instruction. **Syntax: CALL procedure_name** **2. RET: Return from the Procedure.** At the end of the procedure, the RET instruction must be executed. When it is executed, the previously stored content of IP and CS along with Flags are retrieved into the CS, IP and Flag registers from the stack and execution of the main program continues further. **Syntax :RET**	2 M for each explanation
4.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Differentiate between Procedure and Macros.**	**4 M**
	Ans		1 Mark for each point

Procedure	Macro
Procedures are used for large group of instructions to be repeated	Procedures are used for small group of instructions to be repeated.
Object code is generated only once in memory.	Object code is generated every time the macro is called.
CALL & RET instructions are used to call procedure and return from procedure.	Macro can be called just by writing its name.
Length of the object file is less	Object file becomes lengthy.
Directives PROC & ENDP are used for defining procedure.	MACRO and ENDM are used for defining MACRO
Directives More time is required for its execution	Less time is required for it's execution
Procedure can be defined as Procedure_name PROC ---- ------ Procedure_name ENDP	Macro can be defined as MACRO-name MACRO [ARGUMENT,..... ARGUMENT N] ------ ------- ENDM
For Example Addition PROC near ------ Addition ENDP	For Example Display MACRO msg ------ ENDM

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

	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
			4 M for program

		LEA SI, STRG
		MOV CL,00H
		MOV AL,'$'
		NEXT: CMP AL,[SI]
		JZ EXIT
		ADD CL,01H
		INC SI
		JMP
		NEXT EXIT: MOV LEN,CL
		MOV AH,4CH
		INT 21H
		CODE ENDS	

	c)	**Explain the following assembler directives:** **(i) DB (ii) SEGMENT (iii) DUP (iv) EQU**	**4 M**
	Ans	(i) **DB:** The DB directive is used to declare a BYTE -2-BYTE variable – A BYTE is made up of 8 bits. Declaration examples: (ii) **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 (iii) **DUP:** The DUP directive can be used to generate multiple bytes or words with known as well as un-initialized values. Example : Table dw 100 DUP(0) Stars db 50 dup('*') ARRAY3 DB 30 DUP(?)	1 Mark for each

		(iv) **EQU :** 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 withthe 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.	
	d)	**Write an ALP to count number 1' in 8 bit number.**	**4 M**

	Ans	DATA SEGMENT
		N DB 12H
		Z DB 0
		DATA ENDS
		CODE SEGMENT
		ASSUME DS:DATA, CS:CODE
		START:
		MOV DX,DATA
		MOV DS,DX
		MOV AL, N
		MOV CL,04
		NEXT: ROL AL,01
		JNC ONE
		INC Z
		ONE: LOOP NEXT
		HLT
		CODE ENDS
		END START
			4 M for program

	e)	**Explain any four Addressing Modes of 8086.**	

	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.
			1 Mark for each

		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]	

5.		**Attempt any TWO of the following:**	**12 M**
	a)	**Define Logical and Effective address. Describe how 20 bit Physical address is generated in 8086. If CS = 348AH and IP = 4214H, calculate the Physical Address.**	**6 M**
	Ans	Logical Address:	

- Logical address are also known as virtual address. It is generated by CPU during program execution.
- Logical addresses provide a way for the CPU to access different locations in memory without needing to know the physical organization of the memory.

Definition

2 M

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.

Explanation of generation of Physical address :

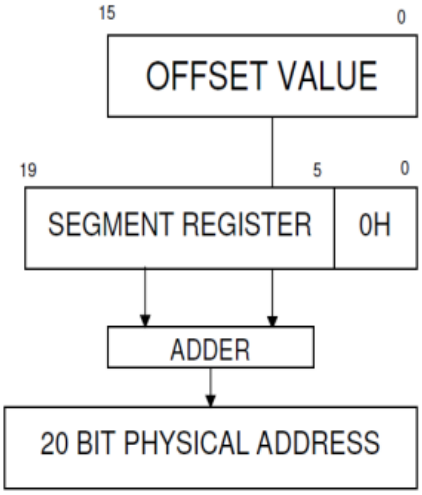
2 M

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

Calculation of Physical addresss :

2 M



if CS 348AH and IP = 4214H,

Physical address=Segment base address*10+Offset (Effective) address

=CS*10 + IP

=348AH*10H+4214H

= 38AB4 H

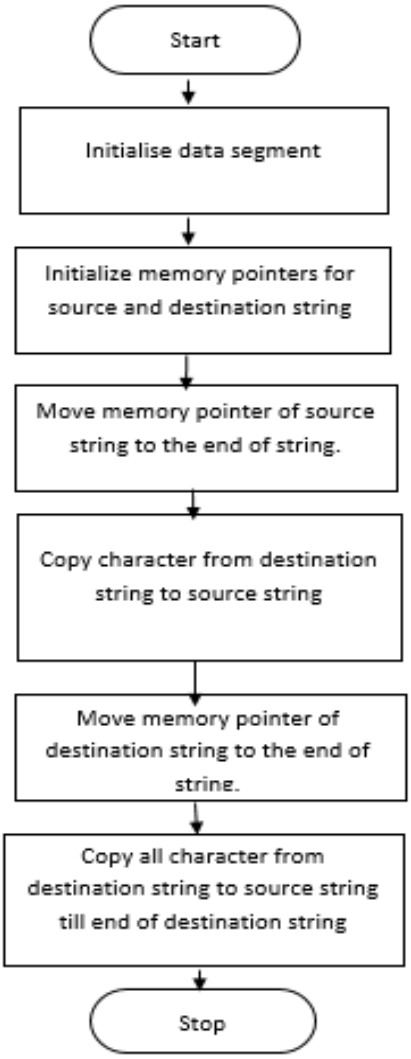
	b)	**Select the instructions for each of the following :** **(i) Multiply AL by 05H**	**6 M**

		(ii) Move 1234H in DS register **(iii) Add AX with BX** **(iv) Signed Division of AX by BL** **(v) Rotate the contents of AX towards left by 4 bits through carry** **(vi) Load SP register with FF00H.**	

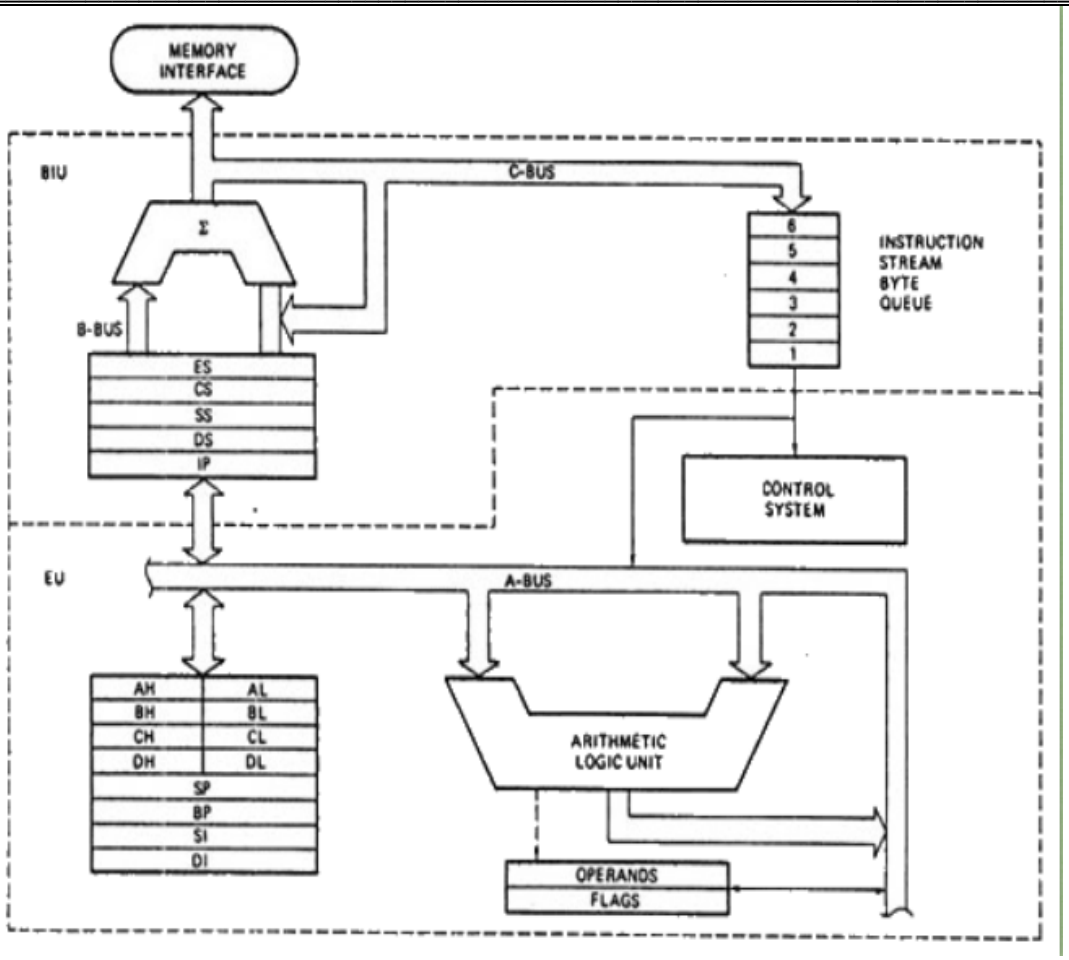
	Ans	**: i)Multiply AL by 05H**.
		MOV BL,05H
		MUL BL
		ii)Move 1234H in DS register
		MOV AX,1234H
		MOV DS,AX
		iii) Add AX with BX
		ADD AX,BX
		iv) Signed Division of AX by BL
		IDIV BL
		v)Rotate the contents of AX towards left by 4 bits through carry
		MOV CL,04
		RCL AX,CL
		vi) **Load SP register with FF00H.**
		MOV SP,FFOOH
			1 M for each instruction

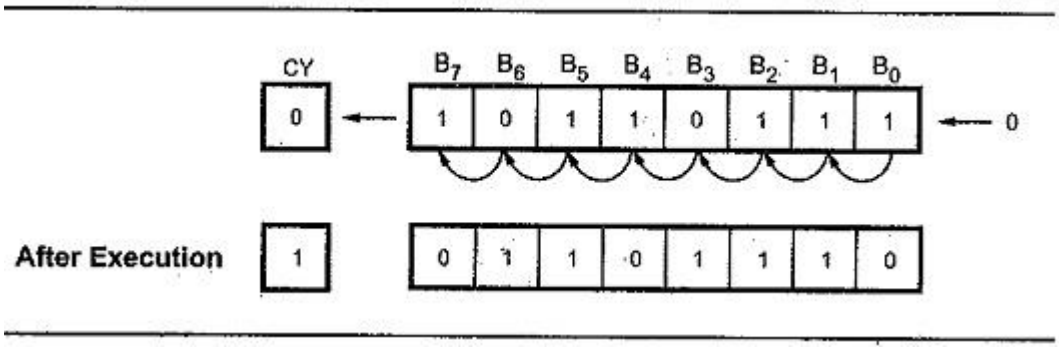
	c)	**Write an ALP for concatenation of two strings. Draw flow chart and assume suitable data.**	**6 M**
	Ans	.MODEL SMALL .DATA STR_S DB 'Hello $' STR_D DB 'World $' .CODE MOV AX, @DATA	Correct program : 4 M Flowchart:

		MOV DS, AX
		MOV SI, OFFSET STR_S
		NEXT:
		MOV AL, [SI]
		CMP AL, '$'
		JE EXIT
		INC SI
		JMP NEXT
		EXIT:
		MOV DI, OFFSET STR_D
		UP: MOV AL, [DI]
		CMP AL, '$'
		JE EXIT1
		MOV [SI], AL
		INC SI
		INC DI
		JMP UP
		EXIT1:
		MOV AL, '$'
		MOV [SI], AL
		MOV AH, 4CH
		INT 21H
		ENDS
		END
			2 M



6.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Draw the functional block diagram of 8086 with all labels.**	**6 M**

	Ans		Correct Diagram: 6 M
	b)	**Explain with example any three Shift and any three Rotate instructions.**	**6 M**
	Ans	**1.SAL/SHL Destination , Count:** Left shifts the bits of destination. MSB is shifted into CARRY LSB gets 0 Bits are shifted count no.of times. IF count =1 ,it is directly specified in instruction If count > 1, it has to be loaded in CL register e.g. SAL BL,1 : Left shift BL bits once	Rotate Instructions :3 M Shift instructions :3 M



More examples:

MOV CL,05H: Load no of shifts in CL register

SHL BL,CL: left shift BL bits CL(5) number of times

2. SHR Destination , Count:

Right shifts the bits of destination.

LSB is shifted into CARRY

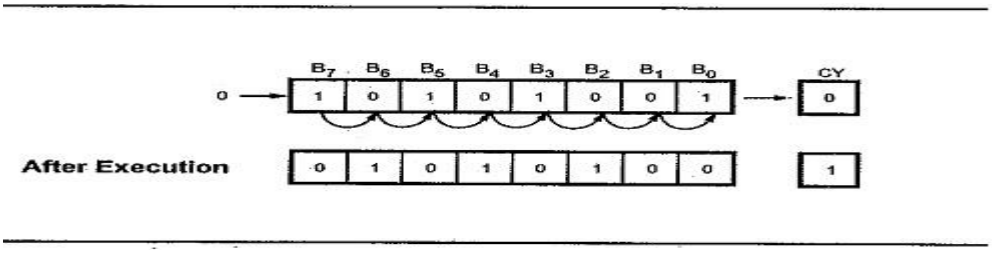
MSB gets 0

Bits are shifted count no.of times.

IF count =1 ,it is directly specified in instruction

If count > 1, it has to be loaded in CL register

e.g. SHR BL,1 right shift BL bits once



More examples:

MOV CL,05H: Load no of shifts in CL register

SHR BL,CL: right shift BL bits CL(5) number of times

3. SAR Destination , Count:

Right shifts the bits of destination.

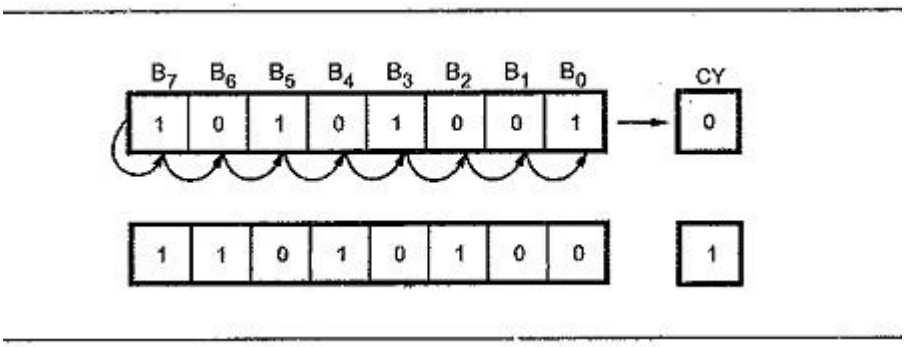
LSB is shifted into CARRY

MSB placed in MSB itself

Bits are shifted count no.of times.

IF count =1 ,it is directly specified in instruction

If count > 1, it has to be loaded in CL register



Rotate Instructions

1.ROL Destination , Count:

Left shifts the bits of destination.

MSB is shifted into CARRY

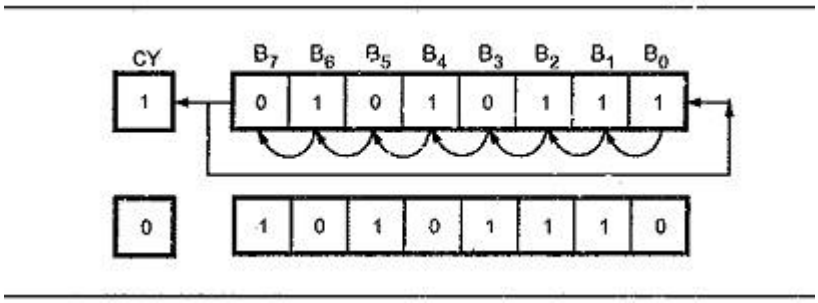
MSB also goes to LSB

Bits are shifted count no.of times.

IF count =1 ,it is directly specified in instruction

If count > 1, it has to be loaded in CL register

e.g ROL BL,1 : Left shift BL bits once



2. ROR Destination , Count:

Right shifts the bits of destination.

LSB is shifted into **CARRY**

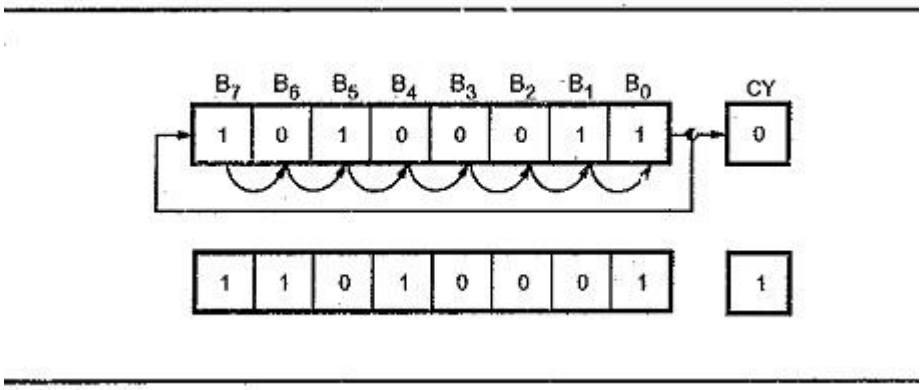
LSB also goes to **MSB**

Bits are shifted count no.of times.

IF count =1 ,it is directly specified in instruction

If count > 1, it has to be loaded in CL register

e.g. ROR BL,1 : Left shift BL bits once



3.RCL Destination , Count:

Left shifts the bits of destination.

MSB is shifted into CARRY

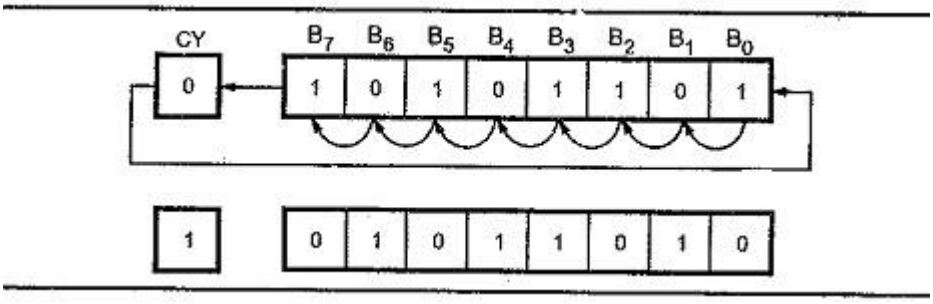
Carry goes **to LSB**

Bits are shifted count no.of times.

IF count =1 ,it is directly specified in instruction

If count > 1, it has to be loaded in CL register

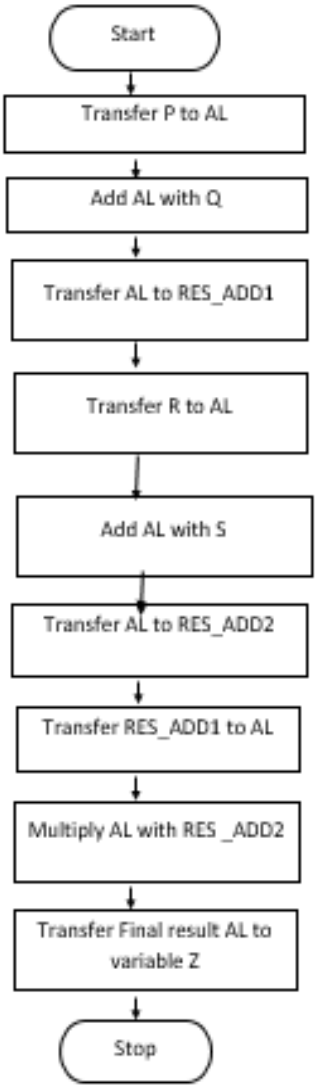
e.G RCL BL,1 : Left shift BL bits once



	c)	**Write an ALP for Z = (P + Q) * (R+ S) using MACRO. Draw flow chart of the same.**	**6 M**

	Ans	MODEL SMALL
		ADD_NO1 MACRO P,Q,RES_ADD1 :MACRO DECLARATION (P+Q)
		MOV AL,P
		ADD AL,Q
		MOV RES_ADD1,AL
		ENDM
		ADD_NO2 MACRO R, S,RES_ADD2 :MACRO DECLARATION (R+S)
		
		MOV AL,R
		ADD AL,S
		MOV RES_ADD2,AL
		ENDM
		MULTIPLY_NUM MACRO RES_ADD1,RES_ADD2,Z
		MOV AL,RES_ADD1
		MUL RES_ADD2
		ENDM
		.DATA
		P DB 02H
		Q DB 03H
		R DB 04H
		S DB 05H
		RES_ADD1 DB ? :RESULT OF P+Q
		RES_ADD2 DB ? : RESULT OF R+S
		
		Z DW? : RESULT OF (P+Q)* (R+S)
		ENDS
		.CODE
		START:
			Correct Program : 4 M
			
			Flowchart: 2 M

```
MOV AX,@DATA
MOV DS,AX
ADD_NO1 MACRO P,Q,RES_ADD1          : CALL MACRO TO ADD
ADD_NO2 MACRO R,S,RES_ADD2          : CALL MACRO TO ADD

MULTIPLY_NUM MACRO RES_ADD1,RES_ADD2,Z          : CALL MACRO
TO MULTIPLY

MOV AX,4C00H
INT   21H
ENDS
END START
```


Start
Transfer P to AL
Add AL with Q
Transfer AL to RES_ADD1
Transfer R to AL
Add AL with S
Transfer AL to RES_ADD2
Transfer RES_ADD1 to AL
Multiply AL with RES _ADD2
Transfer Final result AL to variable Z
Stop

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.

<table>
<tr><th>Q. No.</th><th>Sub Q. N.</th><th>Answer</th><th>Marking Scheme</th></tr>
<tr><td>1</td><td></td><td>Attempt any FIVE :</td><td>10 M</td></tr>
<tr><td></td><td>a</td><td>State the function of BHE and A_0 pins of 8086.</td><td>2 M</td></tr>
<tr><td></td><td>Ans</td><td>BHE: BHE stands for Bus High Enable. It is available at pin 34 and used to indicate the transfer of data using data bus D8-D15. This signal is low during the first clock cycle, thereafter it is active.

A_0: A_0 is analogous to BHE for the lower byte of the data bus, pinsD_0-D_7. A_0 bit is Low during T1 state when a byte is to be transferred on the lower portion of the bus in memory or I/O operations.

<table>
<tr><th>BHE</th><th>A_0</th><th>Word / Byte access</th></tr>
<tr><td>0</td><td>0</td><td>Whole word from even address</td></tr>
<tr><td>0</td><td>1</td><td>Upper byte from / to odd address</td></tr>
<tr><td>1</td><td>0</td><td>Lower byte from / to even address</td></tr>
<tr><td>1</td><td>1</td><td>None</td></tr>
</table></td><td>Explanation: 1 M each</td></tr>
<tr><td></td><td>b</td><td>How single stepping or tracing is implemented in 8086?</td><td>2 M</td></tr>
<tr><td></td><td>Ans</td><td>By setting the Trap Flag (TF) the 8086 goes to single-step mode. In this mode, after the implementation of every instruction s 8086 generates an internal</td><td>Explanation: 2 M</td></tr>
</table>

		interrupt and by writing some interrupt service routine we can show the content of desired registers and memory locations. So it is useful for debugging the program. **OR** **If the trap flag is set, the 8086** will automatically do a type-1 interrupt after each instruction executes. When the 8086 does a type-1 interrupt, it pushes the flag register on the stack. **OR** The instructions to set the trap flag are: **PUSHF** ; *Push flags on stack* **MOV BP,SP** ; *Copy SP to BP for use as index* **OR WORD PTR[BP+0],0100H** ; *Set TF flag* **POPF** ; *Restore flag Register*	
	c	**State the role Debugger in assembly language programming.**	**2 M**
	Ans	**Debugger:** Debugger is the program that allows the extension of program in single step mode under the control of the user. The process of locating & correcting errors using a debugger is known as Debugger. Some examples of debugger are DOS debug command Borland turbo debugger TD, Microsoft debugger known as code view cv, etc…	Explanation: 2 M
	d	**Define Macro & Procedure.**	**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. General Form : MACRO-name MACRO [ARGUMENT 1,……ARGUMENT N] ----- MACRO CODIN GOES HERE ENDM E.G DISPLAY MACRO 12,13 ----------	Definition: 1 M each

MACRO STATEMENTS

ENDM

Procedure: A procedure is group of instructions that usually performs one task. It is a reusable section of a software program which is stored in memory once but can be used as often as necessary. A procedure can be of two types. 1) Near Procedure 2) Far Procedure

Procedure can be defined as

```
Procedure_name PROC
----
------
Procedure_name
ENDP
```

For Example

```
Addition  PROC near
------
Addition ENDP
```

	e	**Write ALP for addition of two 8bit numbers. Assume suitable data.**	**2 M**

	Ans	.Model small
		.Data
		NUM DB 12H
		.Code
		START:
		MOV AX, @DATA
		MOV DS,AX
		MOV AL, NUM
		MOV AH,13H
			Correct Program:2 M

		ADD AL,AH
		MOV AH, 4CH
		INT 21H
		ENDS
		END	

	f	**List any four instructions from the bit manipulation instructions of 8086.**	**2 M**
	Ans	Bit Manipulation Instructions These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc. Following is the list of instructions under this group − Instructions to perform logical operation • **NOT** − Used to invert each bit of a byte or word. • **AND** − Used for adding each bit in a byte/word with the corresponding bit in another byte/word. • **OR** − Used to multiply each bit in a byte/word with the corresponding bit in another byte/word. • **XOR** − Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.	For Each instruction ½ M
	g	**State the use of REP in string related instructions.**	**2 M**
	Ans	• This is an instruction prefix which can be used in string instructions. • It causes the instruction to be repeated CX number of times. • After each execution, the SI and DI registers are incremented/decremented based on the DF (Direction Flag) in the flag register and CX is decremented i.e. DF = 1; SI, DI decrements. E.g. MOV CX, 0023H CLD REP MOVSB The above section of a program will cause the following string operation ES: [DI] ← DS: [SI] SI ← SI + I	Explanation: 2 M

		DI ← DI + I CX ← CX – 1 to be executed 23H times (as CX = 23H) in auto incrementing mode (as DF is cleared). **REPZ/REPE (Repeat while zero/Repeat while equal)** • It is a conditional repeat instruction prefix. • It behaves the same as a REP instruction provided the Zero Flag is set (i.e. ZF = 1). • It is used with CMPS instruction. **REPNZ/REPNE (Repeat while not zero/Repeat while not equal)** • It is a conditional repeat instruction prefix. • It behaves the same as a REP instruction provided the Zero Flag is reset (i.e. ZF = 0). • It is used with SCAS instruction.	
2		**Attempt any THREE of the following :**	**12 M**
	a	**Explain the concept of pipelining in 8086. State the advantages of pipelining (any two).**	**4 M**
	Ans	**Pipelining**: 1. The process of fetching the next instruction when the present instruction is being executed is called as pipelining. 2. Pipelining has become possible due to the use of queue. 3. BIU (Bus Interfacing Unit) fills in the queue until the entire queue is full. 4. BIU restarts filling in the queue when at least two locations of queue are vacant. **Advantages of pipelining:** • The execution unit always reads the next instruction byte from the queue in BIU. This is faster than sending out an address to the memory and waiting for the next instruction byte to come. • More efficient use of processor. • Quicker time of execution of large number of instruction. • In short pipelining eliminates the waiting time of EU and speeds up the processing. -The 8086 BIU will not initiate a fetch unless and until there are two empty bytes in its queue. 8086 BIU normally obtains two	Explanation: 2 M, For any two Advantages: 2 M

instruction bytes per fetch.

b **Compare Procedure and Macros. (4 points).** **4 M**

Ans

Procedure	**Macro**
Procedures are used for large group of instructions to be repeated	Procedures are used for small group of instructions to be repeated.
Object code is generated only once in memory.	Object code is generated every time the macro is called.
CALL & RET instructions are used to call procedure and return from procedure.	Macro can be called just by writing its name.
Length of the object file is less	Object file becomes lengthy.
Directives PROC & ENDP are used for defining procedure.	MACRO and ENDM are used for defining MACRO
Directives More time is required for its execution	Less time is required for it's execution
Procedure can be defined as Procedure_name PROC ---- ------ Procedure_name ENDP	Macro can be defined as MACRO-name MACRO [ARGUMENT,…… ARGUMENT N] ------ ------- ENDM
For Example Addition PROC near ------ Addition ENDP	For Example Display MACRO msg ------ ENDM

Each Point: 1 M (any 4 Points)

c **Explain any two assembler directives of 8086.** **4 M**

Ans **1. DB** – The DB directive is used to declare a BYTE -2-BYTE variable – A BYTE is made up of 8 bits.
Declaration examples:

Explanation for each for any two assembler

		Byte1 DB 10h Byte2 DB 255; 0FFh, the max. possible for a BYTE CRLF DB 0Dh, 0Ah, 24h ;Carriage Return, terminator BYTE **2. DW** – The DW directive is used to declare a WORD type variable – A WORD occupies 16 bits or (2 BYTE). Declaration examples: Word DW 1234h Word2 DW 65535; 0FFFFh, (the max. possible for a WORD) **3. DD** – The DD directive is used to declare a DWORD – A DWORD double word is made up of 32 bits =2 Word's or 4 BYTE. Declaration examples: Dword1 DW 12345678h Dword2 DW 4294967295 ;0FFFFFFFFh. **4. EQU -** The EQU directive is used to give name to some value or symbol. Each time the assembler finds the given names in the program, it will replace the name with the value or a symbol. The value can be in the range 0 through 65535 and it can be another Equate declared anywhere above or below. The following operators can also be used to declare an Equate: THIS BYTE THIS WORD THIS DWORD A variable – declared with a DB, DW, or DD directive – has an address and has space reserved at that address for it in the .COM file. But an Equate does not have an address or space reserved for it in the .COM file. Example: A – Byte EQU THIS BYTE DB 10 A_ word EQU THIS WORD	directives: 2 M

		DW 1000 A_ dword EQU THIS DWORD DD 4294967295 Buffer Size EQU 1024 Buffer DB 1024 DUP (0) Buffed_ ptr EQU $; actually points to the next byte after the; 1024th byte in buffer. **5. SEGMENT:** It is used to indicate the start of a logical segment. It is the name given to the segment. Example: the code segment is used to indicate to the assembler the start of logical segment. **6. PROC:** (PROCEDURE) It is used to identify the start of a procedure. It follows a name we give the procedure. After the procedure the term NEAR and FAR is used to specify the procedure Example: SMART-DIVIDE PROC FAR identifies the start of procedure named SMART-DIVIDE and tells the assembler that the procedure is far.	
	d	**Write classification of instruction set of 8086. Explain any one type out of them.**	**4 M**
	Ans	**classification of instruction set of 8086** • Data Transfer Instructions • Arithmetic Instructions • Bit Manipulation Instructions • String Instructions • Program Execution Transfer Instructions (Branch & Loop Instructions) • Processor Control Instructions • Iteration Control Instructions • Interrupt Instructions 1) **Arithmetic Instructions:** These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc. **ADD:** The add instruction adds the contents of the source operand to the destination operand.	Classification: 2 M, Explanation any one type: 2 M

Eg. ADD AX, 0100H
ADD AX, BX
ADD AX, [SI]
ADD AX, [5000H]
ADD [5000H], 0100H
ADD 0100H

ADC: Add with Carry
This instruction performs the same operation as ADD instruction, but adds the carry
flag to the result.
Eg. ADC 0100H
ADC AX, BX
ADC AX, [SI]
ADC AX, [5000]
ADC [5000], 0100H

SUB: Subtract
The subtract instruction subtracts the source operand from the destination operand
and the result is left in the destination operand.
Eg. SUB AX, 0100H
SUB AX, BX
SUB AX, [5000H]
SUB [5000H], 0100H

SBB: Subtract with Borrow
The subtract with borrow instruction subtracts the source operand and the borrow flag
(CF) which may reflect the result of the previous calculations, from the destination
operand
Eg. SBB AX, 0100H
SBB AX, BX
SBB AX, [5000H]
SBB [5000H], 0100H

INC: Increment
This instruction increases the contents of the specified Register or memory location
by 1. Immediate data cannot be operand of this instruction.
Eg. INC AX
INC [BX]
INC [5000H]

DEC: Decrement
The decrement instruction subtracts 1 from the contents of the specified register or
memory location.
Eg. DEC AX
DEC [5000H]

NEG: Negate
The negate instruction forms 2's complement of the specified destination in the instruction. The destination can be a register or a memory location. This instruction can
be implemented by inverting each bit and adding 1 to it.
Eg. NEG AL
AL = 0011 0101 35H Replace number in AL with its 2's complement
AL = 1100 1011 = CBH

CMP: Compare
This instruction compares the source operand, which may be a register or an immediate data or a memory location, with a destination operand that may be a register or a memory location
Eg. CMP BX, 0100H
CMP AX, 0100H
CMP [5000H], 0100H
CMP BX, [SI]
CMP BX, CX

MUL: Unsigned Multiplication Byte or Word
This instruction multiplies an unsigned byte or word by the contents of AL.
Eg.
MUL BH ; (AX) (AL) x (BH)
MUL CX ; (DX)(AX) (AX) x (CX)
MUL WORD PTR [SI] ; (DX)(AX) (AX) x ([SI])

IMUL: Signed Multiplication
This instruction multiplies a signed byte in source operand by a signed byte in AL or
a signed word in source operand by a signed word in AX.
Eg. IMUL BH
IMUL CX
IMUL [SI]

CBW: Convert Signed Byte to Word
This instruction copies the sign of a byte in AL to all the bits in AH. AH is then said
to be sign extension of AL.

Eg. CBW
AX= 0000 0000 1001 1000 Convert signed byte in AL signed word in AX.
Result in AX = 1111 1111 1001 1000

CWD: Convert Signed Word to Double Word
This instruction copies the sign of a byte in AL to all the bits in AH. AH is then said
to be sign extension of AL.
Eg. CWD
Convert signed word in AX to signed double word in DX : AX
DX= 1111 1111 1111 1111
Result in AX = 1111 0000 1100 0001

DIV: Unsigned division
This instruction is used to divide an unsigned word by a byte or to divide an unsigned
double word by a word.
Eg.
```
DIV CL ; Word in AX / byte in CL
       ; Quotient in AL, remainder in AH
DIV CX ; Double word in DX and AX / word
       ; in CX, and Quotient in AX,
       ; remainder in DX
```

2) Processor Control Instructions
These instructions are used to control the processor action by setting/resetting the flag values.

STC:
It sets the carry flag to 1.

CLC:
It clears the carry flag to 0.

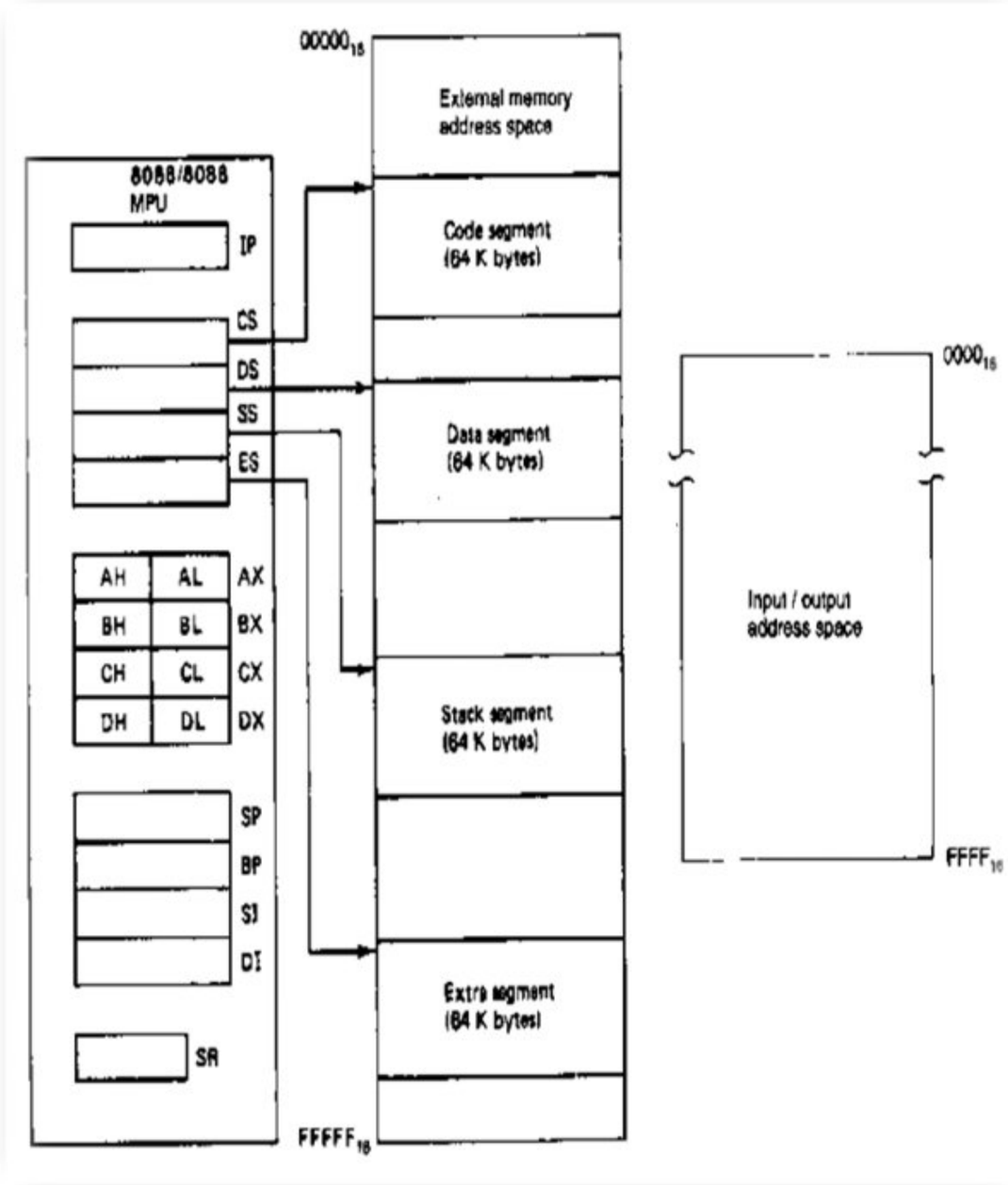
CMC:
It complements the carry flag.

STD:
It sets the direction flag to 1.
If it is set, string bytes are accessed from higher memory address to lower memory address.

CLD:
It clears the direction flag to 0.
If it is reset, the string bytes are accessed from lower memory address to higher

		memory address.	
3		**Attempt any THREE :**	**12 M**
	a	**Explain memory segmentation in 8086 and list its advantages.(any two)**	**4 M**
	Ans	Memory Segmentation: • In 8086 available memory space is 1MByte. • This memory is divided into different logical segments and each segment has its own base address and size of 64 KB. • It can be addressed by one of the segment registers. • There are four segments.	Explanation 2M Any two Advantages 2M

SEGMENT	SEGMENT REGISTER	OFFSET REGISTER
Code Segment	CSR	**Instruction Pointer (IP)**
Data Segment	DSR	**Source Index (SI)**
Extra Segment	ESR	**Destination Index (DI)**
Stack Segment	SSR	**Stack Pointer (SP) / Base Pointer (BP)**



Advantages of Segmentation:

- The size of address bus of 8086 is 20 and is able to address 1 Mbytes ( ) of physical memory.
- The compete 1 Mbytes memory can be divided into 16 segments, each of 64 Kbytes size.
- It allows memory addressing capability to be 1 MB.
- It gives separate space for Data, Code, Stack and Additional Data segment as Extra segment size.
- The addresses of the segment may be assigned as $0000H$ to $F000H$ respectively.
- The offset values are from 00000H to FFFFFH
- Segmentation is used to increase the execution speed of computer system so that processor can able to fetch and execute the data from memory easily and fast.

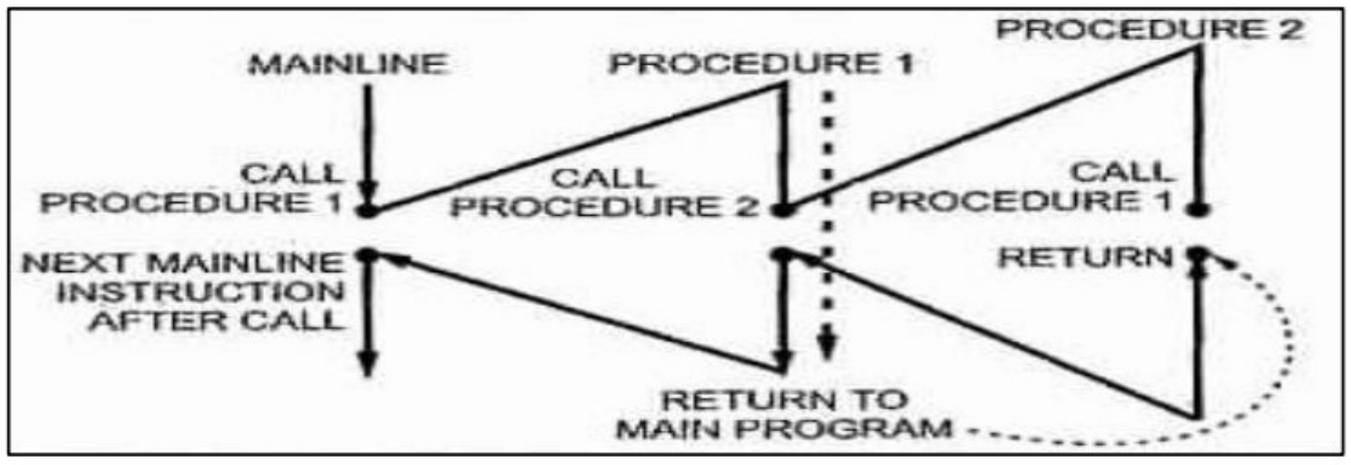
	b	**Write an ALP to count the number of positive and negative numbers in array.**	**4 M**
	Ans	;Count Positive No. And Negative No.S In Given ;Array Of 16 Bit No. **;Assume array of 6 no.s**	Correct program: 4 M

		CODE SEGMENT
		ASSUME CS:CODE,DS:DATA
		START: MOV AX,DATA
		MOV DS,AX
		MOV DX,0000H
		MOV CX,COUNT
		MOV SI, OFFSET ARRAY
		NEXT: MOV AX,[SI]
		ROR AX,01H
		JC NEGATIVE
		INC DL
		JMP COUNT_IT
		NEGATIVE: INC DH
		COUNT_IT: INC SI
		INC SI
		LOOP NEXT
		MOV NEG_COUNT,DL
		MOV POS_COUNT,DH
		MOV AH,4CH
		INT 21H
		CODE ENDS
		
		DATA SEGMENT
		ARRAY DW F423H,6523H,B658H,7612H, 2300H,1559H
		COUNT DW 06H
		POS_COUNT DB ?
		NEG_COUNT DB ?
		DATA ENDS
		END START
			For basic logic may give 1-2 M

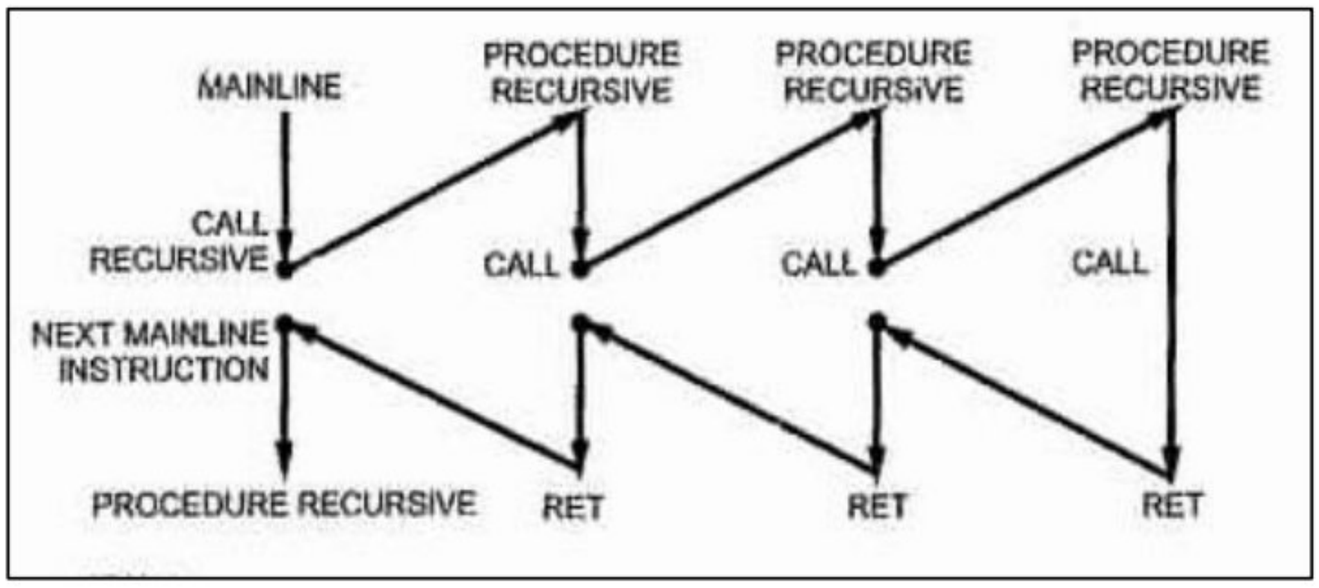
	c	**Write an ALP to find the sum of series. Assume series of 10 numbers.**	**4 M**

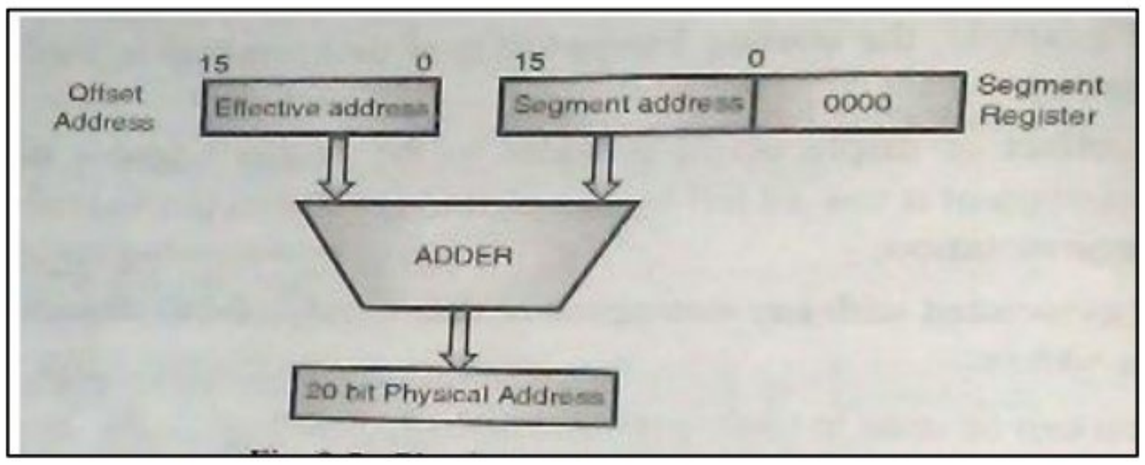
	Ans	; Assume TEN , 8 bit HEX numbers
		CODE SEGMENT
		
		ASSUME CS:CODE,DS:DATA
		
		START: MOV AX,DATA
		
		MOV DS,AX
		
		LEA SI,DATABLOCK
		
		MOV CL,0AH
		
		UP:MOV AL,[SI]
		
		ADD RESULT_LSB,[SI]
			Correct program: 4 M
			For basic logic may give 1-2 M

```
        JNC DOWN
        INC REULT_MSB
DOWN:INC SI
        LOOP UP
CODE ENDS


DATA SEGMENT
DATABLOCK DB 45H,02H,88H,29H,05H,45H,78H,
             95H,62H,30H
RESULT_LSB DB 0
RESULT_MSB DB 0
DATA ENDS


END
```

	d	**With neat sketches demonstrate the use of re-entrant and recursive procedure.**	**4 M**
	Ans	**Reentrant Procedure:** A reentrant procedure is one in which a single copy of the program code can be shared by multiple users during the same period of time. Re-entrance has two key aspects: The program code cannot modify itself and the local data for each user must be stored separately.  **Recursive procedures**: An active **procedure** that is invoked from within itself or from within another	Reentrant: 2 M and recursive procedure explanation With both diagram :2M

active **procedure** is a **recursive procedure**. Such an invocation is called **recursion**. A **procedure** that is invoked **recursively** must have the **RECURSIVE** attribute specified in the **PROCEDURE** statement.



4		**Attempt any THREE :**	**12 M**
	a	**Describe mechanism for generation of physical address in 8086 with suitable example.**	**4 M**
	Ans	 **Fig.: Mechanism used to calculate physical address in 8086** As all registers in 8086 are of 16 bit and the physical address will be in 20 bits. For this reason the above mechanism is helpful. Logical Address is specified as segment: offset Physical address is obtained by shifting the segment address 4 bits to the left and adding the offset address. Thus the physical address of the logical address A4FB:4872 is: **A4FB0** + **4872** ----------	For diagram or computation shown 1M , Explanation 2 M , and for example 1 M

A9822

OR

- i.e. Calculate physical Address for the given
  CS= 3525H, IP= 2450H.

CS		3	5	2	5	0	Implied Zero
IP	+	-	2	4	5	5	
Physical Address		**3**	**7**	**6**	**A**	**5**	**i.e. 376A5H**

b **Write ALP to count ODD and EVEN numbers in an array.** **4 M**

Ans

```
;Count  ODD and EVEN No.S In Given ;Array Of 16 Bit No.
;Assume array of 10 no.s

CODE SEGMENT
ASSUME CS:CODE,DS:DATA
   START:   MOV AX,DATA
                MOV DS,AX
                MOV DX,0000H
                MOV CX,COUNT
                MOV SI, OFFSET ARRAY1
NEXT:           MOV AX,[SI]
                ROR AX,01H
                JC ODD_1
                INC DL
                 JMP COUNT_IT
ODD_1   :       INC DH
COUNT_IT:       INC SI
                 INC SI
                LOOP NEXT
                MOV ODD_COUNT,DH
                MOV EVENCNT,DL
                MOV AH,4CH
                INT 21H
CODE ENDS

DATA SEGMENT
ARRAY1 DW  F423H, 6523H, B658H, 7612H, 9875H,
                  2300H, 1559H, 1000H, 4357H,  2981H
COUNT DW 0AH
ODD_COUNT DB ?
EVENCNT DB ?
```

Correct program: 4 M
For basic logic may give 1-2 M

		DATA ENDS END START	
	c	**Write ALP to perform block transfer operation of 10 numbers.**	**4 M**

	Ans	;Assume block of TEN 16 bit no.s
		;**Data Block Transfer** Using String Instruction
		CODE SEGMENT
		ASSUME CS:CODE,DS:DATA,ES:EXTRA
		MOV AX,DATA
		MOV DS,AX
		MOV AX,EXTRA
		MOV ES,AX
		MOV CX,000AH
		LEA SI,BLOCK1
		LEA DI,ES:BLOCK2
		CLD
		REPNZ MOVSW
		MOV AX,4C00H
		INT 21H
		CODE ENDS
		DATA SEGMENT
		BLOCK1 DW 1001H,4003H,6005H,2307H,4569H, 6123H,
		1865H, 2345H,4000H,8888H
		DATA ENDS
		EXTRA SEGMENT
		BLOCK2 DW ?
		EXTRA ENDS
		END
			Correct program: 4 M
			For basic logic may give 1-2 M

	d	**Write ALP using procedure to solve equation such as** **Z= (A+B)*(C+D)**	**4 M**
	Ans	; **Procedure For Addition** SUM PROC NEAR ADD AL,BL RET SUM ENDP DATA SEGMENT NUM1 DB 10H NUM2 DB 20H NUM3 DB 30H NUM4 DB 40H RESULT DB? DATA ENDS CODE SEGMENT ASSUME CS: CODE,DS:DATA	Correct program: 4 M For basic logic may give 1-2 M

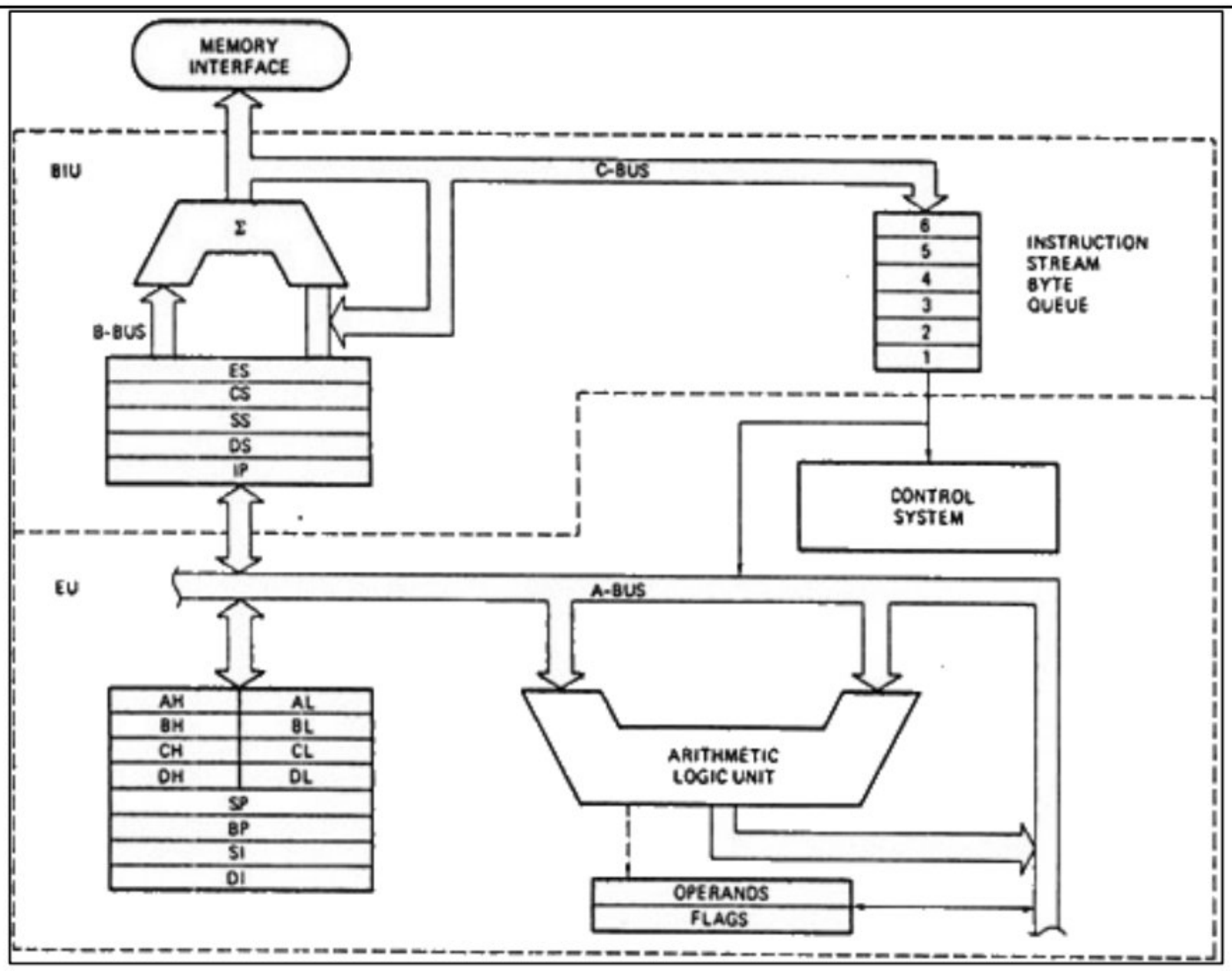
		START:MOV AX,DATA
		MOV DS,AX
		MOV AL,NUM1
		MOV BL,NUM2
		CALL SUM
		MOV CL,AL
		MOV AL, NUM3
		MOV BL,NUM4
		CALL SUM
		MUL CL
		MOV RESULT,AX
		MOV AH,4CH
		INT 21H
		CODE ENDS
		END	

	e	**Write ALP using macro to perform multiplication of two 8 Bit Unsigned numbers.**	**4 M**

	Ans	**; Macro For Multiplication**
		
		PRODUCT MACRO FIRST,SECOND
		MOV AL,FIRST
		MOV BL,SECOND
		MUL BL
		PRODUCT ENDM
		
		DATA SEGMENT
		NO1 DB 05H
		NO2 DB 04H
		MULTIPLE DW ?
		DATA ENDS
		
		CODE SEGMENT
		ASSUME CS: CODE,DS:DATA
		START:MOV AX,DATA
		MOV DS,AX
		PRODUCT NO1,NO2
		MOV MULTIPLE, AX
		MOV AH,4CH
		INT 21H
		CODE ENDS
		END
			Correct program: 4 M
			For basic logic may give 1-2 M

5		**Attempt any TWO :**	**12 M**
	a	**Draw architectural block diagram of 8086 and describe its register organization.**	**6 M**

Ans



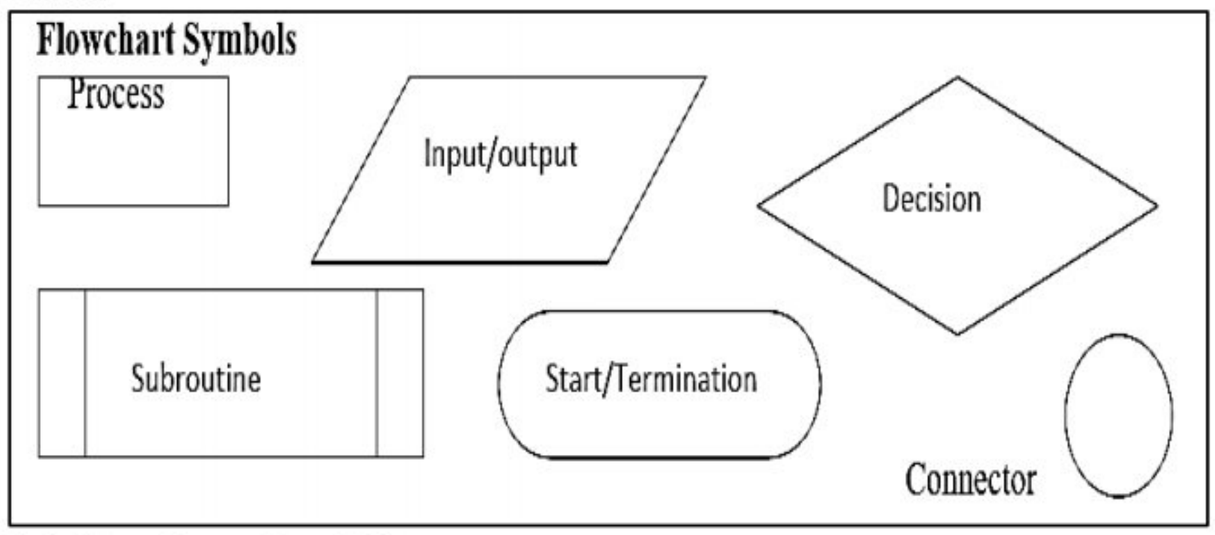
Register Organization of 8086

1. **AX** (Accumulator) – Used to store the result for arithmetic / logical operations
2. **BX** – Base – used to hold the offset address or data
3. **CX** – acts as a counter for repeating or looping instructions.
4. **DX** – holds the high 16 bits of the product in multiply (also handles divide operations)
5. **CS** – Code Segment – holds base address for all executable instructions in a program
6. **SS** - Base address of the stack
7. **DS** – Data Segment – default base address for variables
8. **ES** – Extra Segment – additional base address for memory variables in extra segment.
9. **BP** – Base Pointer – contains an assumed offset from the SS register.
10. **SP** – Stack Pointer – Contains the offset of the top of the stack.

Diagram : 3M

List of Register :1M,

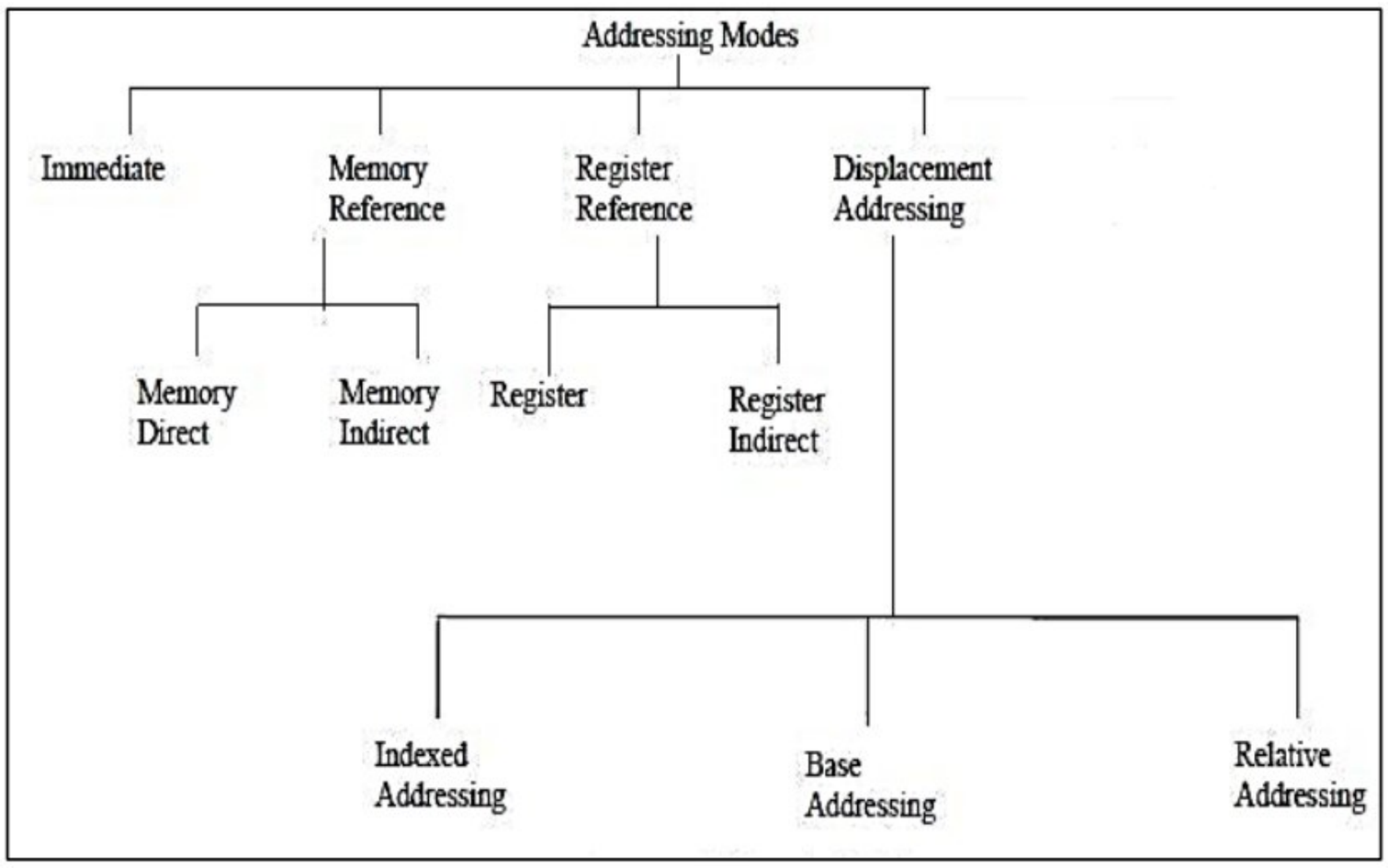
Any 4 registers explanation : ½ M each

		11. **SI** – Source Index – Used in string movement instructions. The source string is pointed to by the SI register. 12. **DI** – Destination Index – acts as the destination for string movement instructions 13. **IP** – Instruction Pointer – contains the offset of the next instruction to be executed. 14. **Flag Register** – individual bit positions within register show status of CPU or results of arithmetic operations.	
	b	**Demonstrate in detail the program development steps in assembly language programming.**	**6 M**
	Ans	**Program Development steps** 1. **Defining the problem** The first step in writing program is to think very carefully about the problem that you want the program to solve. 2. **Algorithm** The formula or sequence of operations or task need to perform by your 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** We should 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.	Each step : 1M (Flowchart symbols are optional)

	c	Illustrate the use of any three branching instructions.	6 M
	Ans	**BRANCH INSTRUCTIONS** Branch instruction transfers the flow of execution of the program to a new address specified in the instruction directly or indirectly. When this type of instruction is executed, the CS and IP registers get loaded with new values of CS and IP corresponding to the location to be transferred. **Unconditional Branch Instructions :** **1. CALL : Unconditional Call** The CALL instruction is used to transfer execution to a subprogram or procedure by storing return address on stack There are two types of calls- NEAR (Inter-segment) and FAR(Intra-segment call). Near call refers to a procedure call which is in the same code segment as the call instruction and far call refers to a procedure call which is in different code segment from that of the call instruction. **Syntax: CALL procedure_name** **2. RET: Return from the Procedure.** At the end of the procedure, the RET instruction must be executed. When it is executed, the previously stored content of IP and CS along with Flags are retrieved into the CS, IP and Flag registers from the stack and execution of the main program continues further. **Syntax :RET** **3. JMP: Unconditional Jump** This instruction unconditionally transfers the control of execution to the specified address using an 8-bit or 16-bit displacement. No Flags are affected by this instruction. **Syntax : JMP Label** **4. IRET: Return from ISR** When it is executed, the values of IP, CS and Flags are retrieved from the stack to continue the execution of the main program. **Syntax: IRET** **Conditional Branch Instructions** When this instruction is executed, execution control is transferred to the address specified relatively in the instruction **1. JZ/JE Label** Transfer execution control to address 'Label', if ZF=1. **2. JNZ/JNE Label** Transfer execution control to address 'Label', if ZF=0 **3. JS Label** Transfer execution control to address 'Label', if SF=1.	Any 3 branch instructions: 2M each

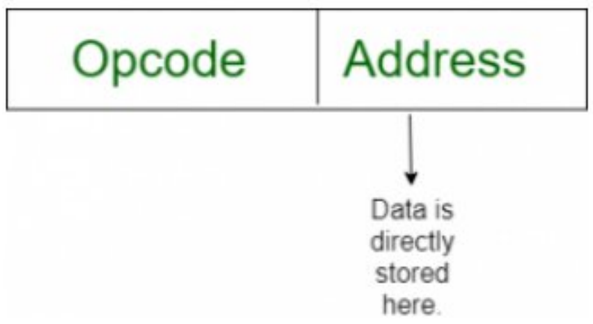
		4. JNS Label Transfer execution control to address 'Label', if SF=0. **5. JO Label** Transfer execution control to address 'Label', if OF=1. **6. JNO Label** Transfer execution control to address 'Label', if OF=0. **7. JNP Label** Transfer execution control to address 'Label', if PF=0. **8. JP Label** Transfer execution control to address 'Label', if PF=1. **9. JB Label** Transfer execution control to address 'Label', if CF=1. **10. JNB Label** Transfer execution control to address 'Label', if CF=0. **11. JCXZ Label** Transfer execution control to address 'Label', if CX=0 **Conditional LOOP Instructions**. **12. LOOP Label :** Decrease CX, jump to label if CX not zero. 13.**LOOPE label** Decrease CX, jump to label if CX not zero and Equal (ZF = 1). **14.LOOPZ label** Decrease CX, jump to label if CX not zero and ZF= 1. **15.LOOPNE label** Decrease CX, jump to label if CX not zero and Not Equal (ZF = 0). **16. LOOPNZ label** Decrease CX, jump to label if CX not zero and ZF=0	
6		**Attempt any TWO :**	**12 M**
	a	**Describe any six addressing modes of 8086 with suitable diagram.**	**6 M**

	Ans	**Different addressing modes of 8086 :**	Any 6 addressing modes correct description: 1M each



1. Immediate: In this addressing mode, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

ex. MOV AX, 0050H

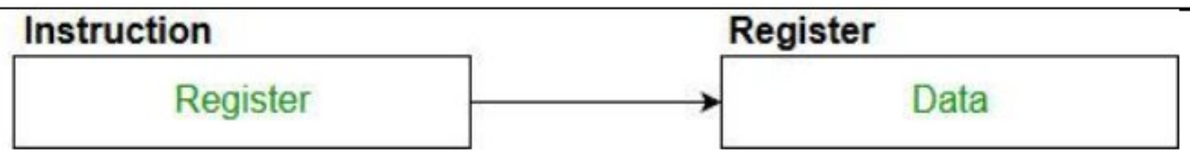


2. Direct: In the direct addressing mode, a 16 bit address (offset) is directly specified in the instruction as a part of it.

ex. MOV AX ,[1 0 0 0 H]

3. Register: In register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers except IP may be used in this mode.

ex. 1)MOV AX,BX



4. Register Indirect: In this addressing mode, the address of the memory location which contains data or operand is determined in an indirect way using offset registers. The offset address of data is in either BX or SI or DI register. The default segment register is either DS or ES.

e.g. MOV AX, $[BX]$

5. Indexed: In this addressing mode offset of the operand is stored in one of the index register. DS and ES are the default segments for index registers SI and DI respectively

e.g. MOV AX, $[SI]$

6. Register Relative: In this addressing mode the data is available at an effective address formed by adding an 8-bit or 16-bit displacement with the content of any one of the registers BX, BP, SI and DI in the default either DS or ES segment.

e.g. MOV AX, 50H$[BX]$

7. Based Indexed: In this addressing mode the effective address of the data is formed by adding the content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

e.g MOV AX, $[BX][SI]$

8. Relative Based Indexed: The effective address is formed by adding an 8-bit or 16-bit displacement with the sum of contents of any one of the base register (BX or BP) and any one of the index registers in a default segment.

e.g. MOV AX, 50H$[BX][SI]$

9 .Implied addressing mode:

		No address is required because the address is implied in the instruction itself. e.g NOP,STC,CLI,CLD,STD Instruction Data	
	b	**Select an appropriate instruction for each of the following & write :** **i)Rotate the content of DX to write 2 times without carry** **ii)Multiply content of AX by 06H** **iii)Load 4000H in SP register** **iv)Copy the contents of BX register to CS** **v)Signed division of BL and AL** **vi) Rotate AX register to right through carry 3 times.**	**6 M**

	Ans	**i)**
		MOV CL,02H
		ROR DX,CL
		(OR)
		ROR DX,03H
		ii)
		MOV BX,06h
		MUL BX
		iii)
		MOV SP,4000H
		iv)
		The contents if CS register cannot be modified directly , Hence no instructions are used However examiner can give marks if question is attempted.
		v)
			Each correct answer : 1 M each

		IDIV BL **vi)** MOV CL,03H RCR AX,CL **(OR)** RCR AX,03H	
	c	**Write an ALP to arrange numbers in array in descending order.**	**6 M**

	Ans	**DATA SEGMENT**
		ARRAY DB 15H,05H,08H,78H,56H
		DATA ENDS
		CODE SEGMENT
		START:ASSUME CS:CODE,DS:DATA
		MOV DX,DATA
		MOV DS,DX
		MOV BL,05H
		
		STEP1: MOV SI,OFFSET ARRAY
		MOV CL,04H
		STEP: MOV AL,[SI]
		CMP AL,[SI+1]
		JNC DOWN
		
		XCHG AL,[SI+1]
		XCHG AL,[SI]
		
		DOWN:ADD SI,1
		LOOP STEP
		DEC BL
		JNZ STEP1
		MOV AH,4CH
		INT 21H
		CODE ENDS
		END START
			Correct Program: 6M (For basic logic may give 2-4 M)

WINTER – 2022 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.
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 function of the following pins of 8086 microprocessor.** **(i) ALE (ii) DT/$\overline{R}$**	**2 M**
	Ans	**(i)ALE (Pin number 25)** – ALE is an abbreviation for address latch enable. Whenever an address is present in the multiplexed address and data bus, then the microprocessor enables this pin. This is done to inform the peripherals and memory devices about fetching of the data or instruction at that memory location. **(ii) DT/$\overline{R}$ (Pin number 27)** – This pin is used to show whether the data is getting transmitted or is received. A high signal at this pin provides the information regarding the transmission of data. While a low indicates reception of data.	Each 1 M
	b)	**Write an assembly language instruction of 8086 microprocessor to** **i) Divide the content of AX register by 50H.** **ii) Rotate the content of BX register by 4-bit towards left.**	**2 M**

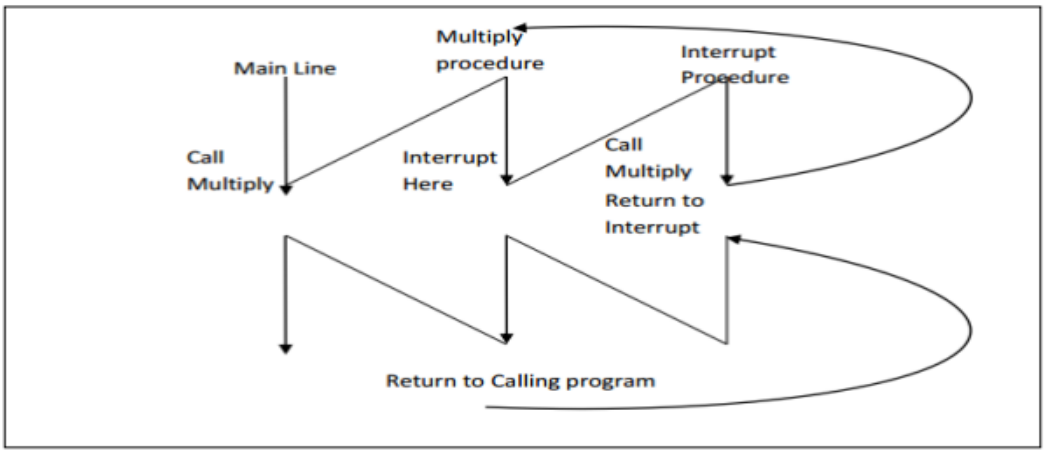
	Ans	(i) Divide the content of AX register by 50H:
		MOV BL,50H
		DIV BL
		(ii) Rotate the content of BX register by 4 bits towards left:
		MOV CL,04H
			Correct Instruction: 1 M each

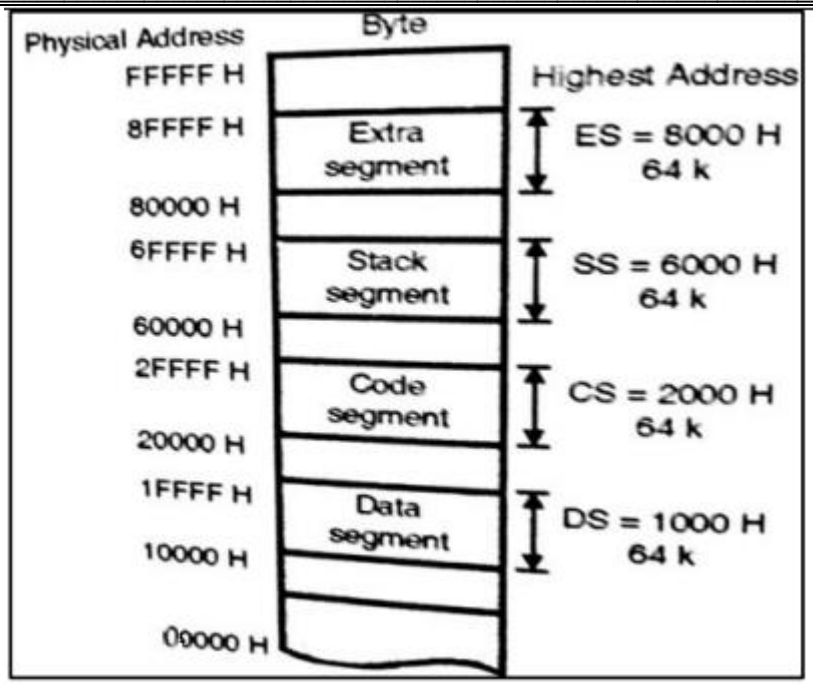
		ROL BX, CL OR MOV CL,04H RCL BX, CL	
	c)	**List directives used for procedure.**	**2 M**
	Ans	The assembler directive that are used for defining a procedure in the 8086 microprocessors are: **PROC and ENDP**	Each 1 M
	d)	**State any two differences between FAR and NEAR procedure.**	**2 M**
	Ans		Any 2 Valid points: each 1 M

SR.NO	NEAR PROCEDURE	FAR PROCEDURE
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

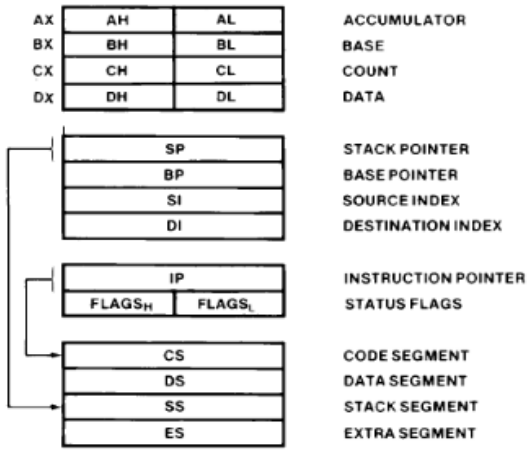
	e)	**Write algorithm to find sum of a series of numbers.**	**2 M**
	Ans	1) Load the count in CX and clear AX and BX. 2) Store the starting address in SI. 3) Move data stored at address pointed by SI in DX. 4) Add AX=AX+DX. 5) If carry exists, increment BX. 6) Increment SI twice. Decrement CX. 7) If CX is not zero, return to step 3. 8) Store the sum (AX) and carry (BX) in memory. 9) Terminate the program.	Any other correct relevant algorithm 2 M
	f)	**What is the use of REP in string related instruction? Explain.**	**2 M**
	Ans	**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.	1M- Definition, 1M-Explanation

		Two more prefix. REPE/REPZ: Repeat if Equal /Repeat if Zero. It will cause string instructions to be repeated as long as the compared by 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	
	g)	**Differentiate between ROL and RCL.**	**2 M**
	Ans	<table><tr><th>ROL</th><th>RCL</th></tr><tr><td>• Rotate left byte or word</td><td>• Rotate through carry left byte or word</td></tr><tr><td>• Syntax: ROL Destination, Count</td><td>• Syntax: RCL Destination, Count Can be used to Swap the nibbles Cannot be used to swap the nibbles</td></tr></table>	1M- For Each Point
2.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**What do you mean by procedure? Explain re-centrant and re-entrant procedure.**	**4 M**
	Ans	**A procedure** is a set of code to be executed several times in a program, and called whenever required. A repeated group of instruction in a program can be organized as subprogram. The subprograms are called as **subroutine or procedures** in assembly language programming which allows reuse of program code. A procedure is a set of the program statements that can be processed independently. **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. To be a re-entrant, • Procedure must first push all the flags and registers used in the procedure. • It should also use only registers or stack to pass parameters. • The flow of re-entrant procedure for a multiply procedure when interrupt procedure is executed, as shown below.	Definition 2 M Explanation 2 M

			
	b)	**What is memory segmentation? Explain it with reference to 8086 microprocessor**.	**4 M**
	Ans	**Memory Segmentation**: Segmentation is the process in which the main memory of the computer is logically divided into different segments and each segment has its own base address. It is basically used to enhance the speed of execution of the computer system, so that the processor is able to fetch and execute the data from the memory easily and fast. 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. **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.	2M -Explanation 2M -Diagram

			
	c)	**Describe following assembler directives:** **(i) DB (i) EQU (ii) Segment (iv) Assume**	**4 M**
	Ans	1) DB: Define Byte The DB directive is used to declare a BYTE -2-BYTE variable – A BYTE is made up of 8 bits. Declaration examples Byte1 DB 10h. 2) EQU: Equate to The EQU directive is used to declare the micro symbols to which some constant value is assigned. Micro assembler will replace every occurrence of the symbol in a program by its value. Syntax: Symbol name EQU expression Example: CORRECTION_FACTOR EQU 100 3) 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 4) ASSUME: Assume directive is used to tell Assembler the name of the logical segment it should use for the specified segment. Example: Assume CS: MAP_CODE, DS: MAP_DATA	1M – For Each
	d)	**What are the functions of CALL and RET instructions? Describe in brief.**	**4 M**

	Ans	**CALL Instruction:** It is used to transfer program control to the sub-program or subroutine. The CALL can be NEAR, where the procedure is in the same segment whereas in FAR CALL, procedure is in a different segment. **Syntax:** CALL procedure name (direct/indirect) **Operation:** Steps executed during CALL **Example:** 1) For Near CALL SP ←SP - 2 Save IP on stack IP address of procedure 2) For Far call SP ← SP-2 Save CS on stack CS New segment base containing procedure SP←SP-2 Save IP on stack IP Starting address of called procedure **RET instruction**: it is used to transfer program execution control from a procedure to the next instruction immediate after the CALL instruction in the calling program. **Syntax:** RET Operation: Steps executed during RET **Example:** 1) For Near Return IP Content from top of stack SP ←SP + 2 2) For Far Return IP Contents from top of stack SP ←SP+2 CS Contents of top of stack SP←SP+2	2M-For Each Instruction
3.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Describe register organization of 8086 microprocessor**.	**4 M**
	Ans	**Register Organization of 8086** 1. AX (Accumulator) - Accumulator register consists of two 8-bit registers AL and AH, which can be combined together and used as a 16- bit register AX. AL in this case contains the low-order byte of the word, and AH contains the high-order byte. Accumulator can be used for I/O operations, rotate and string manipulation. 2. BX –This register is mainly used as a base register. It holds the starting base location of a memory region within a data segment. It is used as offset storage for forming physical address in case of certain addressing mode. 3. CX – It is used as default counter or count register in case of string and loop instructions.	2M-For Diagram,2M-For Explanation



4. DX – Data register can be used as a port number in I/O operations and implicit operand or destination in case of few instructions. In integer 32-bit multiply and divide instruction the DX register contains high-order word of the initial or resulting number.

5. CS – Code Segment – holds base address for all executable instructions in a program

Segment registers:

To complete 1Mbyte memory is divided into 16 logical segments. The complete

1Mbyte memory segmentation is as shown in above figure. Each segment contains 64Kbyte of memory. There are four segment registers.

1.Code segment (CS) is a 16-bit register containing address of 64 KB segment with

Processor instructions. The processor uses CS segment for all accesses to instructions

Referenced by instruction pointer (IP) register.

2.Stack segment (SS) is a 16-bit register containing address of 64KB segment with

Program stack. By default, the processor assumes that all data referenced by the stack

Pointer (SP) and base pointer (BP) registers is located in the stack segment.

3.Data segment (DS) is a 16-bit register containing address of 64KB segment with

Program data. By default, the processor assumes that all data referenced by general

Registers (AX, BX, CX, DX) and index register (SI, DI) is located in the data segment.

4.Extra segment (ES) is a 16-bit register containing address of 64KB segment, usually

with program data.

		Pointers and index registers. The pointers contain within the particular segments. The pointers IP, BP, SP usually contain offsets within the code, data and stack segments respectively. **Stack Pointer (SP)** is a 16-bit register pointing to program stack in stack segment. **Base Pointer (BP)** is a 16-bit register pointing to data in stack segment. **Source Index (SI)** is a 16-bit register. SI is used for indexed, based indexed and register Indirect addressing, as well as a source data addresses in string manipulation instructions. **Destination Index (DI)** is a 16-bit register. DI is used for indexed, based indexed and register indirect addressing, as well as a destination data address in string manipulation Instructions.	
	b)	**Write an assembly language program to add BCD numbers in an array of 10 numbers. Assume suitable array. Store the result at the end of the array.**	**4 M**
	Ans	**Addition of 10 BCD numbers in Series** .MODEL SMALL .STACK 100 .DATA ARRAY DB 1,2,3,4,5,6,7,8,9,10 SUM_LSB DB 0 SUM_MSB DB 0 .CODE MOV AX , @DATA ; Intializing data segment MOV DS , AX MOV CX , 10 ; Initialize byte counter MOV SI , OFFSET ARRAY ; Initialize memory pointer UP: MOV AL , [SI] ; Read byte from memory ADD SUM_LSB , AL ; Add with sum DAA JNC NEXT INC SUM_MSB NEXT: INC SI ; Increment memory pointer LOOP UP ; Decrement byte counter ; If byte counter==0 then exit ; else read next number MOV DL , SUM_MSB MOV AH , 2	4M- For Correct Program

		INT 21H MOV DL , SUM_LSB MOV AH , 2 INT 21H MOV AH , 4CH INT 21H END	
	c)	**Write a procedure to find factorial of given number.**	**4 M**

	Ans	**Procedure to find the factorial.**
		DATA SEGMENT
		NUM DB 04H
		DATA ENDS
		CODE SEGMENT
		START: ASSUME CS:CODE, DS:DATA
		MOV AX,DATA
		MOV DS,AX
		CALL FACTORIAL
		MOV AH,4CH
		INT 21H
		PROC FACTORIAL
		MOV BL,NUM ; TAKE NO IN BL REGISTER
		MOV CL,BL ;TAKE CL AS COUNTER
		DEC CL ;DECREMENT CL BY 1
		MOV AL,BL
		UP: DEC BL ;DECREMENT BL TO GET N-1
		MUL BL ;MULTIPLY CONTENT OF N BY N-1
		DEC CL ;DECREMENT COUNTER
		JNZ UP ;REPEAT TILL ZERO
		RET
		FACTORIAL ENDP
		CODE ENDS
		END START
		(OR)
		DATA SEGMENT
		A DW 0005H
		FACT_LSB DW?
		FACT_MSB DW?
		DATA ENDS
		CODE SEGMENT
		ASSUME DS:DATA,CS:CODE
			4M- For Correct Program

```
START:MOV AX,DATA
 MOV DS,AX
 CALL FACTORIAL
 MOV AH,4CH
 INT 21H
 FACTORIAL PROC
 MOV AX,A
 MOV BX,AX
 DEC BX
 UP: MUL BX ; MULTIPLY AX * BX
 MOV FACT_LSB,AX ;ANS DX:AX PAIR
 MOV FACT_MSB,DX
 DEC BX
 CMP BX,0
 JNZ UP
 RET
 FACTORIAL ENDP
CODE ENDS
END START
```

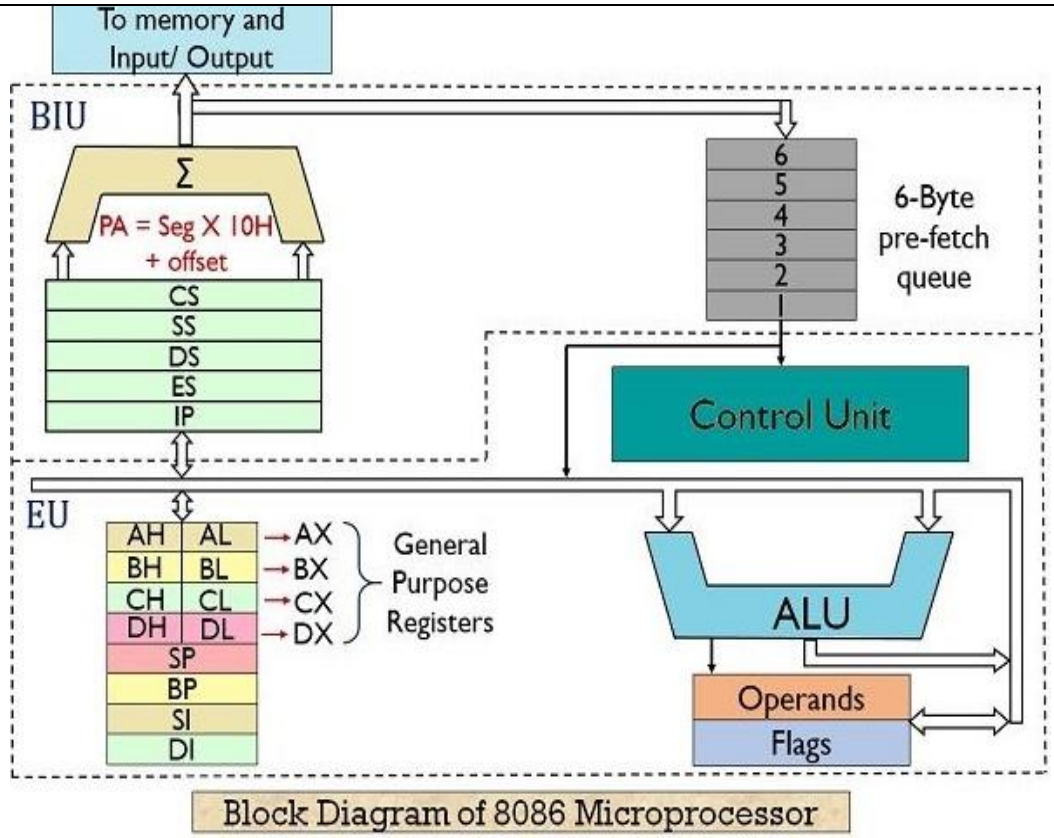
d) Write an assembly language program for conversion of BCD to Hexe number. **4 M**

Ans

4M- For Correct Program

```
Registers used : AL, BL, CL, DX, AH
Procedures used : none
Segments used : Code, Data
DATA SEGMENT
        BCD_NO DB 1 DUP (?)          ; BCD (2 DIGIT packed BCD)
        HEX_NO DB 1 DUP (ϕ)          ; Store hex equivalent here
        DATA ENDS
CODE SEGMENT
        ASSUME CS : CODE, DS : DATA
        MOV DX, DATA                 ; Initialization of Data
        MOV DS, DX                   ; Segment register
        MOV AL, BCD_NO               ; Load the BCD number in AL
```

```
        MOV BL, AL          ; Store it in BL
        AND BL, 0FH         ; Mask the lower BCD digit
        AND AL, 0F0H        ; Mask the upper BCD digit
        MOV CL, 04H         ; Swap the nibbles
        ROR AL. CL
        MOV DL, 0AH         ;
        MUL DL              ; Multiply the upper BCD digit with 0AH
        ADD AL, BL
        MOV HEX_NO, AL      ; Store the Hex equivalent result
        MOV AH, 4CH         ; Program termination with
        INT 21 H            ; return code
CODE    ENDS                ; End of code segment
        END
```

4.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Draw functional block diagram of 8086 microprocessor.**	**4 M**
	Ans	 Block Diagram of 8086 Microprocessor	4M-For Block Diagram

	b)	**Write an assembly language program to arrange the numbers in ascending order (Assume suitable data).**	**4 M**

	Ans	DATA SEGMENT
		ARRAY DB 15h,05h,08h,78h,56h, 60h, 54h, 35h, 24h, 67h
		DATA ENDS
		CODE SEGMENT
		START: ASSUME CS: CODE, DS:DATA
		MOV DX, DATA
		MOV DS, DX
		MOV BL,0AH
		step1: MOV SI,OFFSET ARRAY
		MOV CL,09H
		step: MOV AL,[SI]
		CMP AL,[SI+1]
		JC Down
		XCHG AL,[SI+1]
		XCHG AL,[SI]
		Down : ADD SI,1
		LOOP step
		DEC BL
		JNZ step1
		MOV AH,4CH
		INT 21H
		
		CODE ENDS
		END START
			4M- For Correct Program

	c)	**Write an assembly language program to Count No. of 1's in a 16-bit number.**	**4 M**

	Ans	Assume the number to be stored in BX register. Store the result in CX register.
		MODEL SMALL
		.DATA
		NUM DW 0008H
		ONES DB 00H
		.CODE
		START:
		MOV AX,@DATA
		MOV DS,AX
		MOV CX, 10H ; initialize rotation counter by 16
		MOV BX, NUM ; load number in BX
		UP: ROR BX, 1 ; rotate number by 1 bit right
		JNC DN ; if bit not equal to 1 then go to DN
		INC ONES ; else increment ones by one
		DN: LOOP UP ; decrement rotation counter by 1 and if not zero then go to up
			4M- For Correct Program

```
MOV CX, ONES                ; move result in cx register.
MOV AH, 4CH
INT 21H
ENDS
END                         ; end of program.
```

d) **Write an assembly language program using MACRO to perform following operation.** **4 M**

X = (A +B) * (C +D)

Ans

```
.Model small
add_no1 macro a,b,res_add1
mov al,a
add al,b
mov res_add1,al
endm
add_no2 macro c,d,res_add2
mov al,c
add al,d
mov res_add2,al
endm

multiply_num macro res_add1,res_add2
mov al,res_add1
mul res_add2
endm
.Data
a db 02h
b db 03h
c db 04h
d db 05h
res_add1 db ?
res_add2 db ?
ends
.Code
start :
    mov ax,@data
    mov ds,ax
    mov al,a
    mov bl,b
    mov cl,c
    mov dl,d
    add al,bl
    add cl,dl
```

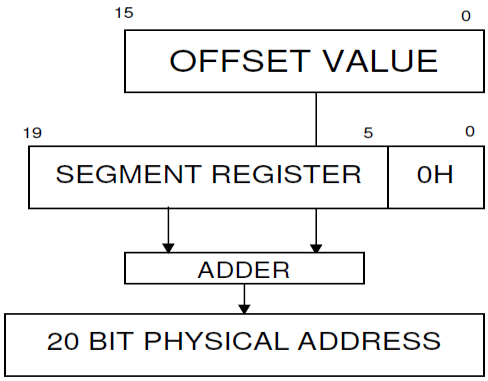
4M- For Correct Program

		mov res_add1,al
		mov res_add2,cl
		multiply_num res_add1,res_add2
		mov ah,4ch
		int 21h
		ends
		end	

	e)	**Describe with suitable example how parameter is passed on the stack in 8086 assembly language procedure.**	**4 M**

	Ans	In order to pass the parameters using stack we push them on the stack before the call for the procedure in the main program. The instructions used in the procedure read these parameters from the stack. Whenever stack is used to pass parameters, it is important to keep a track of what is pushed on the stack and what is popped off the stack in the main program.\
		Example:
		.model small
		.data
		MULTIPLICAND DW 1234H
		MULTIPLIER DW 4232H
		.code
		MOV AX, @data
		MOV DS, AX
		:
		:
		PUSH MULTIPLICAND
		PUSH MULTIPLIER
		CALL MULTI
		:
		:
		MULTI PROC NEAR
		PUSH BP
		MOV BP, SP ; Copies offset of SP into BP
		MOV AX, [BP + 6] ; MULTIPLICAND value is available at
		; [BP + 6] and is passed to AX
		MUL WORD PTR [BP + 4] ; MULTIPLIER value is passed
		POP BP
		RET ; Increments SP by 4 to return address
		MULTI ENDP ; End procedure
		END
			2M-For Explanation,2M-For Example

5.		**Attempt any <u>TWO</u> of the following:**	**12 M**

	a)	**Define logical and effective address, Describe physical address generation process in S086 microprocessor. Calculate physical address by taking suitable DS, CS and IP.**	**6 M**
	Ans	**Logical Address**: It is generated by CPU in perspective of 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. **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  For example if CS = 1000H and IP = 1100H, the microprocessor fetches its next instruction from Physical address=Segment base address*10+Offset (Effective) address =CS*10+IP =1000H*10+1100H =11100H.	Definition-2 M Description-2 M Calculation Example-2 M
	b)	**State the function of following assembly language programing tools:** **(i) Assembler (ii) Linker (ii) Debugger**	**6 M**
	Ans	**(i)Assembler** a) Assembler is a program that translates assembly language program to the correct binary code for each instruction i.e. machine code and generate the file called as object file with extension .obj. b) It also displays syntax errors in the program, if any. c) It can also be used to produce list (.lst) which contains assembly language statements, binary codes, and offset address for each instruction. Example; TASM, MASM. **(ii)Linker** a) It is a programming tool used to convert Object code into executable program. b) It combines ,if requested ,more than one separated assembled modules into one executable	2 M each

		Module such as two or more assembly programs. c) It generates .EXE module Example; TLINK. **(iii)Debugger** a) Debugger is a program that allows the execution of program in single step mode under the control of the user. b) The errors in program can be located and corrected using a debugger. Example; TD.	
	c)	**Describe different addressing modes of 8086 with one suitable example each.**	**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, 3040H **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,BX **3. Direct addressing mode** An instruction in which 16 bit effective address of an operand is specified in the instruction, and then the addressing mode of such instruction is known as direct addressing mode. **Example:** MOV BL,[3000H] **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	Any 6 mode with example 1 M each

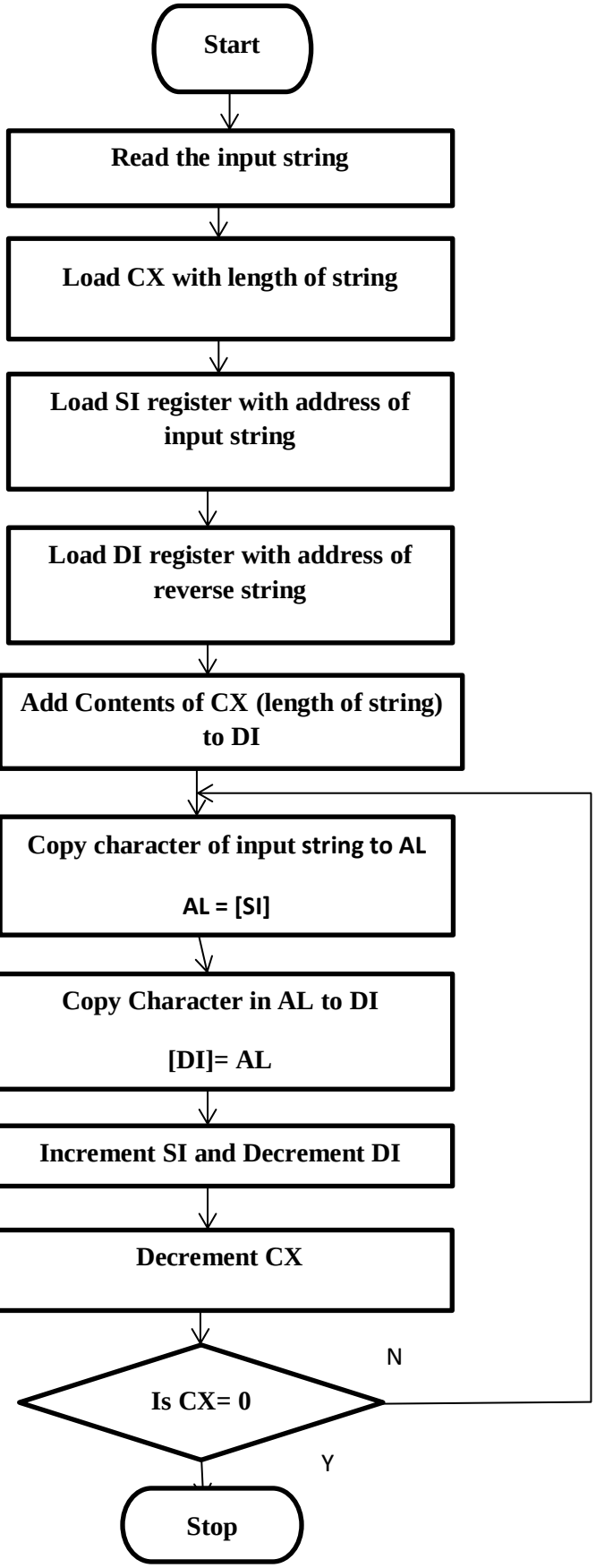
		Example: MOV AX, [BX+SI] **7. Register relative addressing mode:** An instruction in which the address of the operand is obtained by adding the displacement (8-bit or 16 bit) with the contents of base registers or index registers (BX, BP, SI, DI). The default segment register is DS or ES. **Example:** MOV AX, [BX+50H] **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, [BX+SI+50H]	
6.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Describe different branching instructions used in 8086 microprocessor in brief.**	**6 M**
	Ans	Branch instruction transfers the flow of execution of the program to a new address specified in the instruction directly or indirectly. When this type of instruction is executed, the CS and IP registers get loaded with new values of CS and IP corresponding to the location to be transferred Unconditional Branch Instructions: 1. CALL: Unconditional Call The CALL instruction is used to transfer execution to a subprogram or procedure by storing return address on stack There are two types of calls. NEAR (Inter-segment) and FAR(Intra-segment call). Near call refers to a procedure call which is in the same code segment as the call instruction and far call refers to a procedure call which is in different code segment from that of the call instruction. Syntax: CALL procedure name 2. RET: Return from the Procedure. At the end of the procedure, the RET instruction must be executed. When it is executed, the previously stored content of IP and CS along with Flags are retrieved into the CS, IP and Flag registers from the stack and execution of the main program continues further. Syntax :RET 3. JMP: Unconditional Jump This instruction unconditionally transfers the control of execution to the specified address using an 8-bit or 16-bit displacement. No Flags are affected by this instruction. Syntax : JMP Label 4. IRET: Return from ISR When it is executed, the values of IP, CS and Flags are retrieved from the stack to continue the execution of the main program.	Any 3 branch instructions: 2 M each

		Syntax: IRET Conditional Branch Instructions When this instruction is executed, execution control is transferred to the address specified relatively in the instruction 1. JZ/JE Label : Transfer execution control to address 'Label', if ZF=1. 2. JNZ/JNE Label : Transfer execution control to address 'Label', if ZF=0 3. JS Label : Transfer execution control to address 'Label', if SF=1 4. JNS Label Transfer execution control to address 'Label', if SF=0. 5.JO Label Transfer execution control to address 'Label', if OF=1. 6. JNO Label Transfer execution control to address 'Label', if OF=0. 7. JNP Label Transfer execution control to address 'Label', if PF=0. 8. JP Label Transfer execution control to address 'Label', if PF=1. 9. JB Label Transfer execution control to address 'Label', if CF=1. 10. JNB Label Transfer execution control to address 'Label', if CF=0. 11. JCXZ Label Transfer execution control to address 'Label', if CX=0	
	b)	**Explain the following instructions of 8086:** **i) DAA (ii) ADC (ii) XCHG**	**6 M**
	Ans	**i)** DAA − Used to adjust the decimal after the addition operation. It makes the result in Packed BCD from after BCD addition is performed. It works only on AL register.	2 M for each instruction

		All flags are updated; OF becomes Undefined after this instruction. **For AL register ONLY** If D3 – D0 > 9 OR Auxiliary Carry Flag is Set, ADD 06H to AL. If D7 – D4 > 9 OR Carry Flag is Set, ADD 60 H to AL. **Assume :** AL = 14H, CL = 28H Then ADD AL,CL gives AL = 3CH Now DAA gives AL = 42(06 is added to AL as C> 9) ii) ADC − Used to add with carry. ADDs the source to destination with carry and stores the result back into destination e.g. ADC BX,CX will give BX= BX+ CX+ Carry flag iii) XCHG- Used to exchange the data from two locations. 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.	
	c)	**Draw flow chart and write assembly language program to reverse the word in string.**	**6 M**

	Ans	DATA SEGMENT
		STRB DB 'COMPUTER$'
		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]
		MOV [DI],AL
		INC SI
		DEC DI
		LOOP UP
		MOV AH,4CH
		INT 21H
		CODE ENDS
		END START
			Correct program-3 M
			
			Flowchart- 3 M

Ans


Start
Read the input string
Load CX with length of string
Load SI register with address of input string
Load DI register with address of reverse string
Add Contents of CX (length of string) to DI
Copy character of input string to AL
AL = [SI]
Copy Character in AL to DI
[DI]= AL
Increment SI and Decrement DI
Decrement CX
Is CX= 0
N
Y
Stop

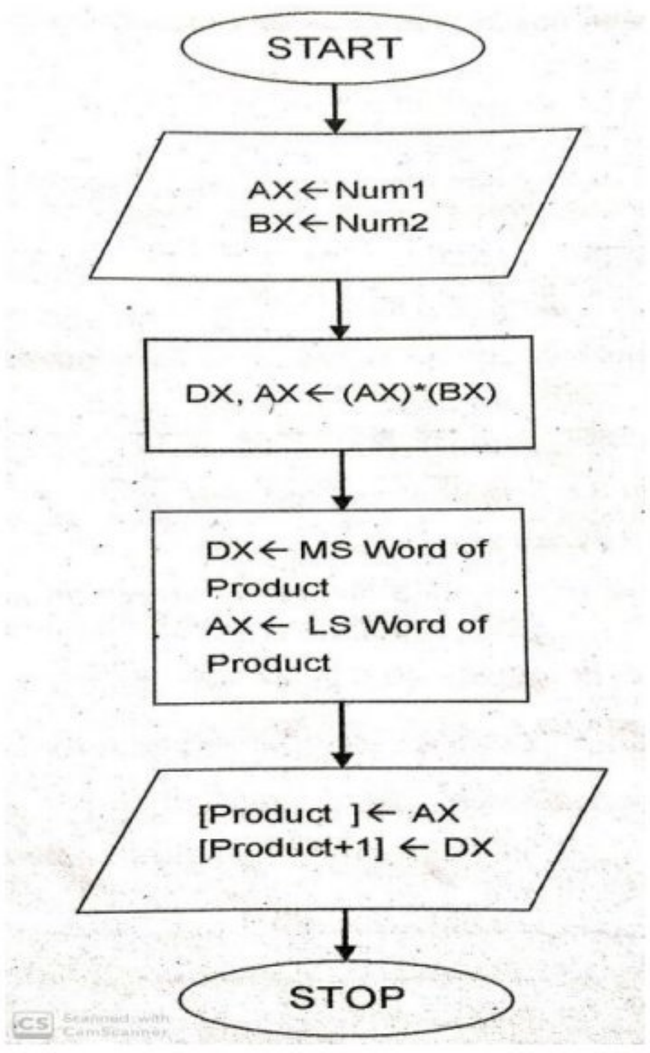
SUMMER – 19 EXAMINATION

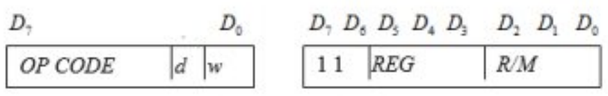
Subject Name: MICROPROCESSOR **Model Answer** **Subject Code: 22415**

Important Instructions to examiners:

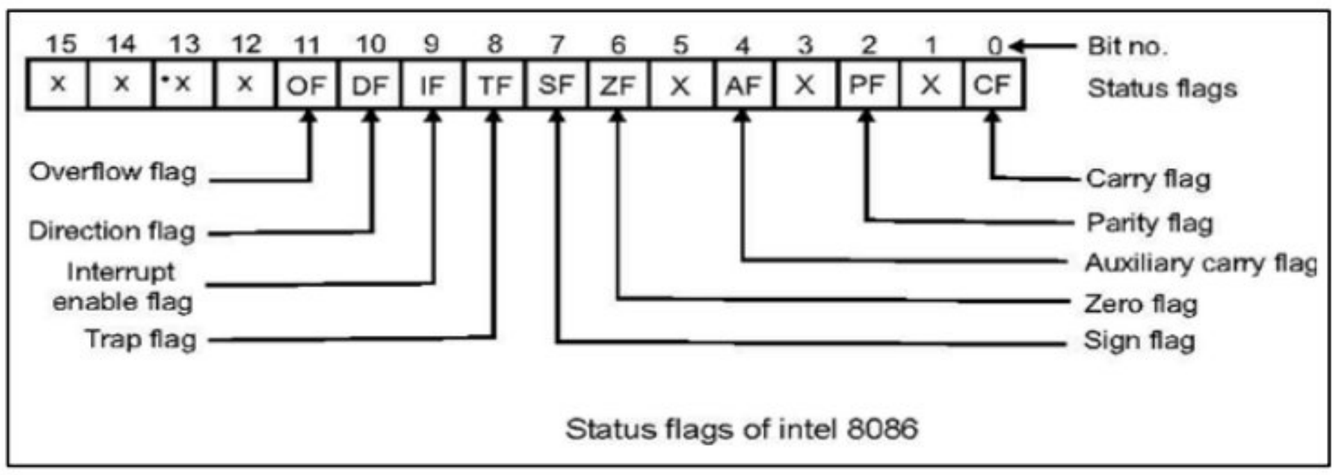
1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling 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)

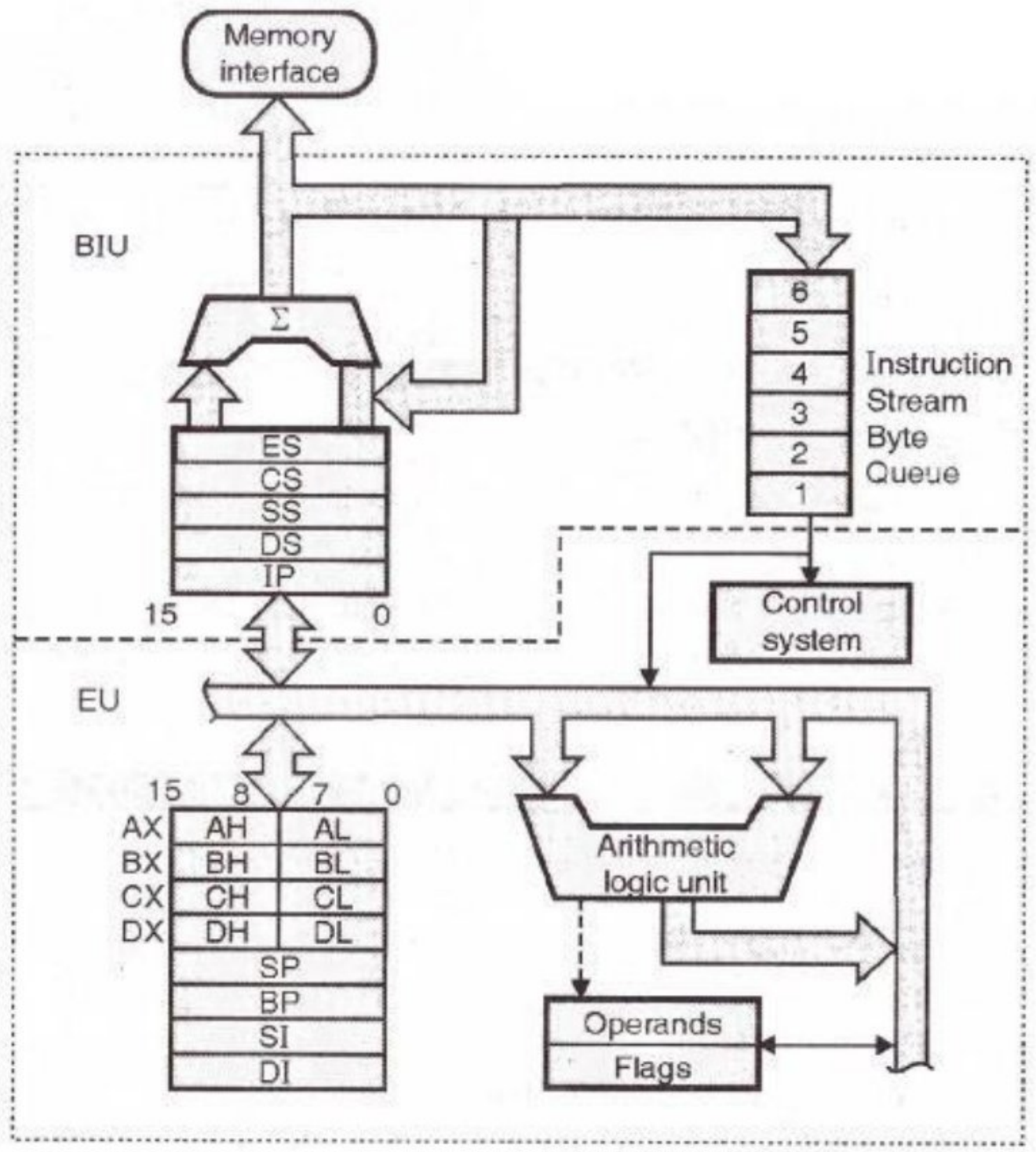
TEST AL, BL ; AND byte in BL with byte in AL, no result, Update PF, SF, ZF.

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	

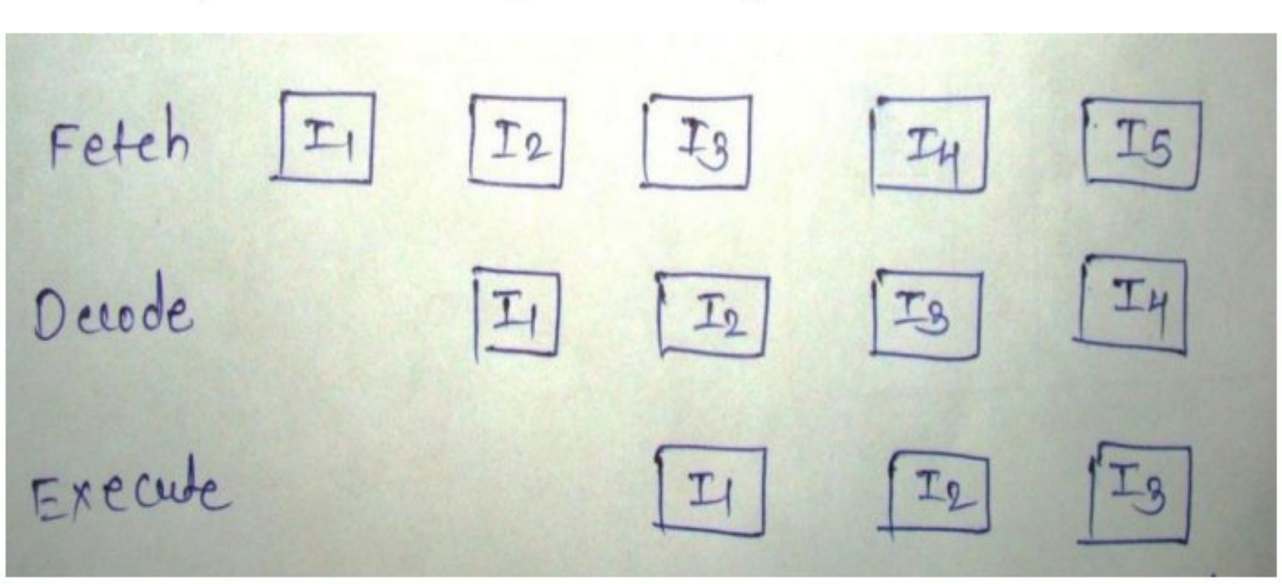
	c	**Write an ALP to find length of string.**	4M

	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**

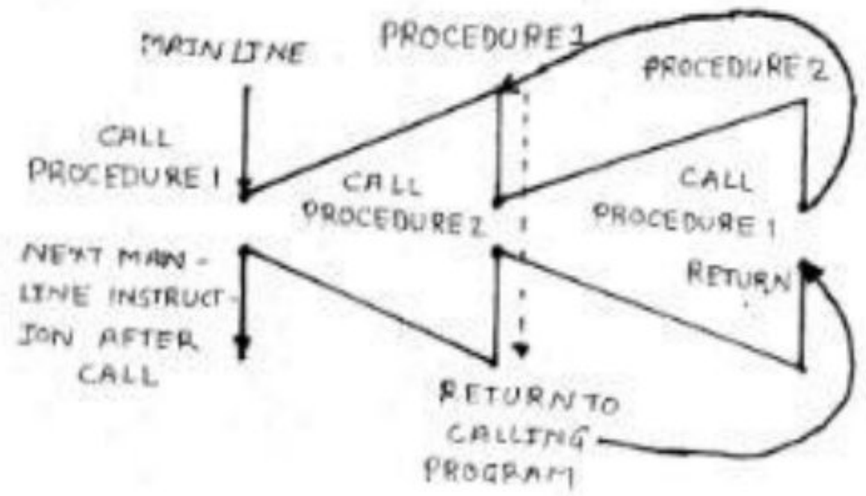
	Ans	DATA SEGMENT
		ARRAY DB 10H,24H,02H,05H,17H
		LARGEST DB 00H
		DATA ENDS
		CODE SEGMENT
		START:
		ASSUME CS:CODE,DS:DATA
		MOV DX,DATA
		MOV DS,DX
		MOV CX,04H
		MOV SI ,OFFSET
		ARRAY MOV AL,[SI]
		UP: INC SI
		CMP AL,[SI]
		JNC NEXT
		MOV AL,[SI]
		NEXT: DEC CX
		JNZ UP
		MOV LARGEST,AL
		MOV 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.

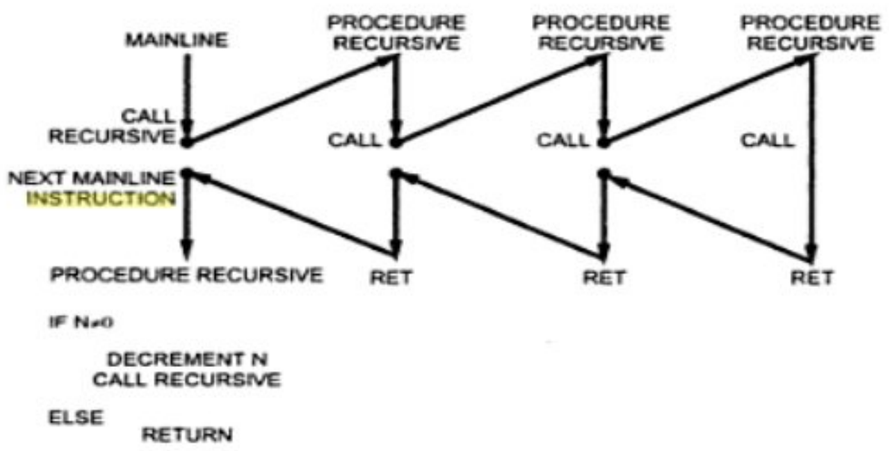


Fig. Flow diagram and pseudo-code for recursive procedure

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

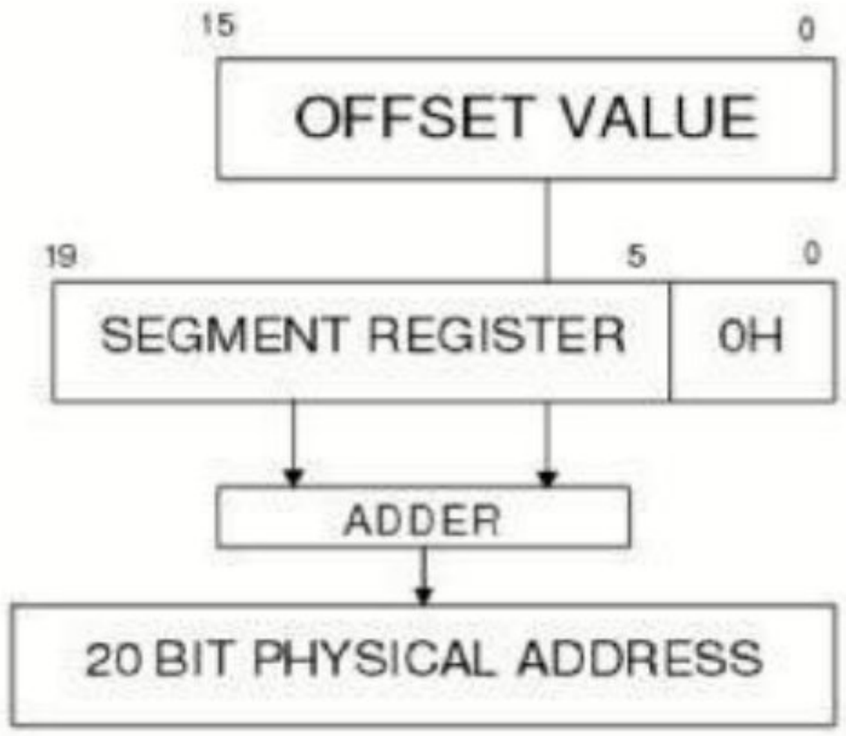
Generation of 20 bit physical address in 8086:-

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

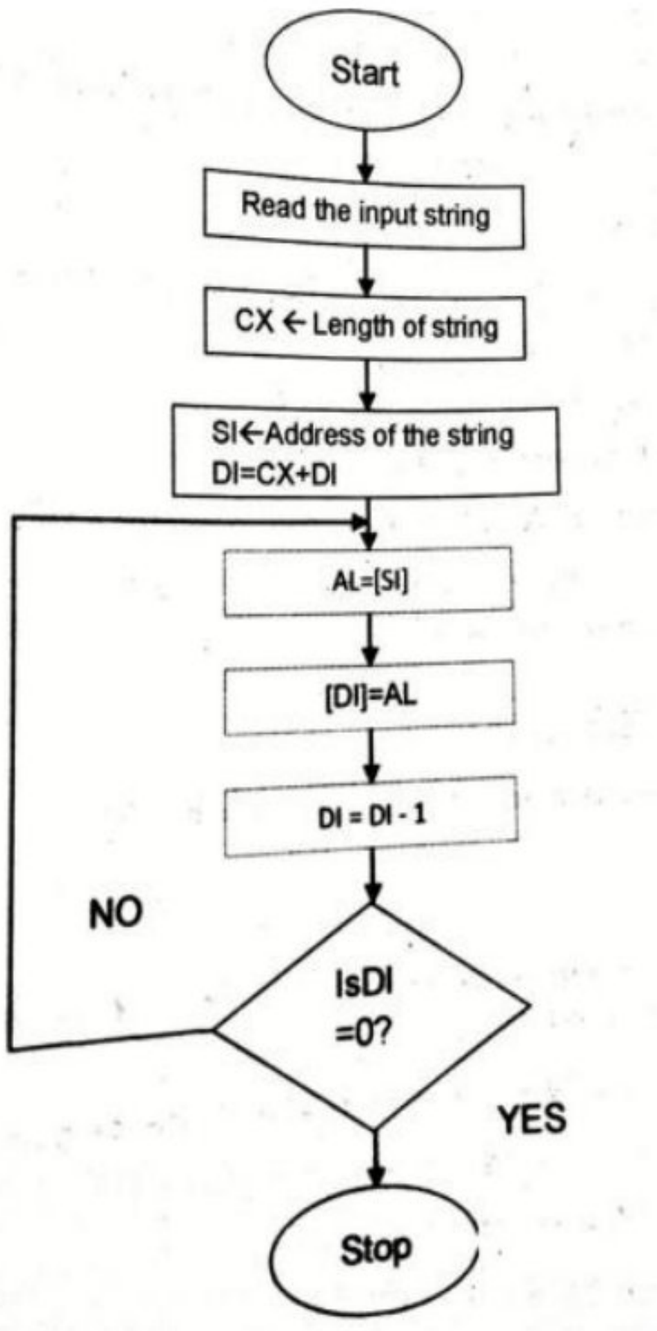
		the contents of base registers or index registers (BX, BP, SI, DI). The default segment register is DS or ES. **Example:** MOV AX, 50H[BX] **8. Relative Based Indexed addressing mode** An instruction in which the address of the operand is obtained by adding the displacement (8 bit or 16 bit) with the 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:

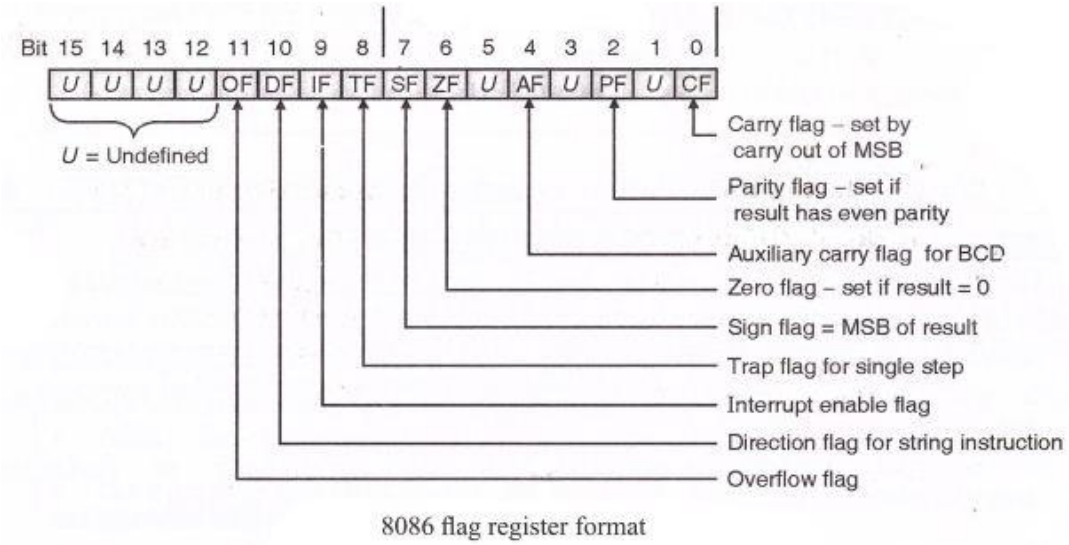


SUMMER – 2022 EXAMINATION

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.
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)	**Draw the labeled format of 8086 flag register**	**2 M**
	Ans	 8086 flag register format	Correct diagram: 2 M

	b)	**State any two difference between TEST and AND instructions.**	**2 M**
	Ans		1 M for each point of comparison

TEST	AND
This instruction logically ANDs the source with the destination but the result is not stored anywhere.	This instruction logically ANDs the source with the destination and stores the result in destination. .
e. g .TEST BL ,CL The result is not saved anywhere.	e.g. AND BL , CL The result is saved in BL register

	c)	**State the function of editor and assembler.**	**2 M**
	Ans	Editor: The editor is a program which allows the user to enter and modify as well as store a group of instructions or text under a file name. Assembler: The assembler is used to convert assembly language written by a user or a program into a machine recognizable format.	1 M for each function
	d)	**Write any two difference between NEAR and FAR procedure.**	**2 M**
	Ans		1 M for each point of comparison

SR.NO	NEAR PROCEDURE	FAR PROCEDURE
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

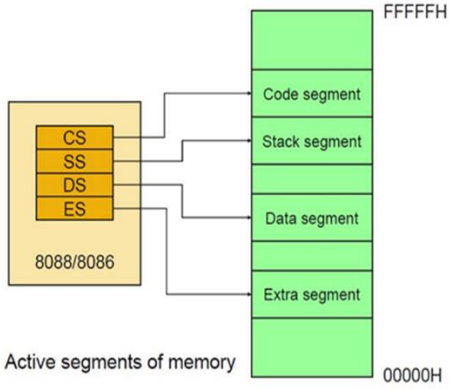
	e)	**Write an ALP to add two 8 bit numbers.**	**2 M**
	Ans	.model small .data	Correct

		a db 06h
		b db 12h
		ends
		.code
		start:
		mov ax,@data
		mov ds,ax
		mov al,a
		mov bl,b
		add al,bl
		int 3
		ends
		end start
			program: 2 M

	f)	**Define immediate addressing mode with suitable example**	**2 M**
	Ans	An instruction in which 8 bit or 16 bit operand (data) is specified in instruction itself then the addressing mode of such instruction is called as immediate addressing mode. Eg. MOV AX,7120H	Definition :1M Example:1M
	g)	**State the use of DAA instruction in BCD addition.**	**2 M**
	Ans	The DAA (Decimal Adjust after Addition) instruction makes the result in Packed BCD from after BCD addition is performed. It works only on AL register.	Explanation: 2 M
2.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Describe the directives used to define the procedure with suitable example**	**4 M**
	Ans	Directives used for procedure: PROC directive: The PROC directive is used to identify the start of a procedure. The PROC directive follows a name given to the procedure. After that the term FAR and NEAR is used to specify the type of the procedure. ENDP Directive: This directive is used along with the name of the procedure to indicate the end of a procedure to the assembler. The PROC and ENDP directive are used in procedure. Example:	Description: 2 M Example: 2 M

		Procedure can be defined as Procedure_name PROC ---- ------ Procedure_name ENDP For Example Addition PROC near ------ Addition ENDP	
	b)	**Write the function of following pins of 8086:** **(i)** $\overline{\text{BHE}}$ **(ii) ALE** **(iii) READY** **(iv) RESET**	**4 M**
	Ans	(i) $\overline{\text{BHE}}$: BHE stands for Bus High Enable. It is available at pin 34 and used to indicate the transfer of data using data bus D8-D15. This signal is low during the first clock cycle, thereafter it is active. (ii) ALE: ALE stands for address Latch Enable, as address and data bus are multiplexed; ALE is used to lock either Address or Data. (iii) 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. (iv) RESET: This pin requires the microprocessor to terminate its present activity immediately	Each pin function 1 M
	c)	**Describe any four assembler directives with suitable example.**	**4 M**
	Ans	1. DB – The DB directive is used to declare a BYTE type variable – A BYTE is made up of 8 bits. Declaration examples: Num1 DB 10h	Each assembler directive 1 M

		Num2 DB 37H 2. DW – The DW directive is used to declare a WORD type variable – A WORD occupies 16 bits or (2 BYTE). Declaration examples: TEMP DW 1234h 3. DD – The DD directive is used to declare a double word which is made up of 32 bits =2 Word's or 4 BYTE. Declaration examples: Dword1 DW 12345678h 4. EQU - This is used to declare symbols to which some constant value is assigned each time the assembler finds the given names in the program, it will replace the name with the value or a symbol. The value can be in the range 0 through 65535 and it can be another Equate declared anywhere above or below. .Num EQU 100 5. SEGMENT: It is used to indicate the start of a logical segment. It is the name given to the segment. Example: the code segment is used to indicate to the assembler the start of logical segment. 6. PROC: (PROCEDURE) It is used to identify the start of a procedure. It follows a name we give the procedure After the procedure the term NEAR and FAR is used to specify the procedure Example: SMART-DIVIDE PROC FAR identifies the start of procedure named SMART-DIVIDE and tells the assembler that the procedure is far.	
	d)	**Describe DAS instruction with suitable example.**	**4 M**
	Ans	**DAS:** Decimal Adjust after Subtraction: - This instruction converts the result of the subtraction operation of 2 packed BCD numbers to a valid BCD number. The subtraction operation has to be only in the AL. If the lower nibble of AL is higher than the value 9, this instruction will subtract 06 from the lower nibble of the AL. If the output of the subtraction operation sets the carry flag or if the upper nibble is higher than value 9, it subtracts 60H from the AL. This instruction modifies the CF, AF, PF, SF, and ZF flags. The OF is not defined after DAS instruction. The instance is following: **Example:** (i) AL = 75 BH = 46 SUB AL, BH ; AL ← 2 F = (AL) – (BH) ; AF = 1 DAS ; AL ← 2 9 (as F> 9, F – 6 = 9)	Description 2 M Example 2 M

3.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Describe memory segmentation in 8086 with suitable diagram.**	**4 M**
	Ans	 **Memory Segmentation**: The memory in 8086 based system is organized as segmented memory. 8086 can access 1Mbyte memory which is divided into number of logical segments. Each segment is 64KB in size and addressed by one of the segment register. The 4 segment register in BIU hold the 16-bit starting address of 4 segments. CS holds program instruction code. Stack segment stores interrupt & subroutine address. Data segment stores data for program. Extra segment is used for string data. ➢ The number of address lines in 8086 is 20, 8086 BIU will send 20bit address, so as to access one of the 1MB memory locations. ➢ The four segment registers actually contain the upper 16 bits of the starting addresses of the four memory segments of 64 KB each with which the 8086 is working at that instant of time ➢ A segment is a logical unit of memory that may be up to 64 kilobytes. Starting address will always be changing. It will not be fixed. Note that the 8086 does not work the whole 1MB memory at any given time. However, it works only with four 64KB segments within the whole 1MB memory.	Diagram: 2 M Explanation: 2 M
	b)	**Write an ALP to multiply two 16 bit signed numbers.**	**4 M**
	Ans	.model small .data A db 2222h B db 1111h	Program Code: 4 M

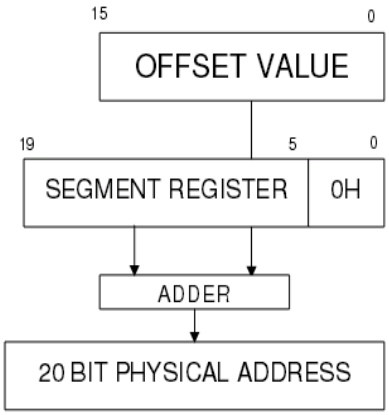
		Ends
		.code
		Mov ax,@data
		Mov ds,ax
		Mov AX,a
		Mov BX,b
		IMul BX
		Int 03h
		Ends
		End	

	c)	**Write an ALP to count odd numbers in the array of 10 numbers**	**4 M**

	Ans	. Model Small
		
		.data
		
		BLK DB 10h,40h,30h,60h
		e db ?h
		o db ?h
		ends
		.code
		mov ax, @data
		mov ds, ax
		lea si, BLK
		mov bl, 00h
		mov bh, 00h
		mov cl, 04h
		up: mov al, [si]
		ror al, 1
		jc go
		inc bl
		jmp next
		go: inc bh
		next: inc si
		dec cl
		jnz up
		mov e,bl
		mov o,bh
		int 3
		ends
		end
			Program Code: 4 M

	d)	**Write a MACRO to perform 32 bit by 16 bit division of unsigned numbers.**	**4 M**
	Ans	.model small Div1 macro no1,no2	Program Code: 4 M

```
mov ax,no1
div no2
endm
.data
num1 dw 12346666h
num2 dw 2222h
.code
mov ax,@data
mov ds,ax
div1 num1,num2
ends
end
```

4.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Describe how 20 bit Physical address is generated in 8086 microprocessor with suitable example.**	**4 M**
	Ans	**Formation of a physical address:-** Segment registers carry 16 bit data, which is also known as base address. BIU attaches 0 as LSB of the base address. So now this address becomes 20-bit address. Any base/pointer or index register carry 16 bit offset. Offset address is added into 20-bit base address which finally forms 20 bit physical address of memory location.  	Describition: 2 M Example: 2 M

		Example Assume DS= 2632H, SI=4567H DS : 26320H ……...0 added by BIU(or Hardwired 0) + SI : 4567H --- 2A887H	
	b)	**Write an ALP to find largest number in the array.**	**4 M**

	Ans	.model small
		.data
		Array db 02h,04h,06h,01h,05h
		Ends
		.code
		Start: Mov ax,@data
		Mov ds,ax
		Mov cl,04h
		Lea si,array
		Mov al,[si]
		Up : inc si
		Cmp al,[si]
		Jnc next
		Mov al,[si]
		Next : dec cl
		Jnz up
		Int 03h
		Ends
			Program Code: 4 M

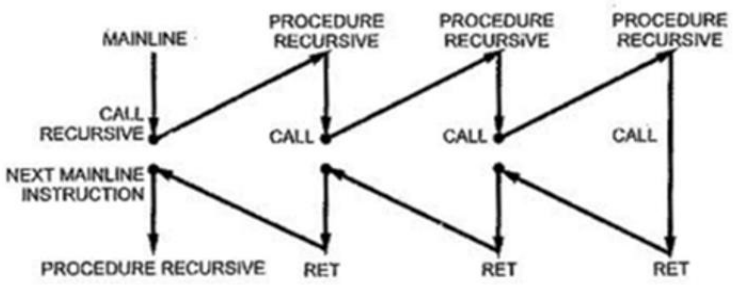
		End start	
	c)	**Write an ALP to count number of 0' in 8 bit number.**	**4 M**

	Ans	.MODEL SMALL
		.DATA
		NUM DB 08H
		ZEROS DB 00H
		.CODE
		START:
		MOV AX,@DATA
		MOV DS,AX
		MOV CX, 08H ; initialize rotation counter by 8
		MOV BX, NUM ;load number in BX
		UP: ROR BX, 1 ; rotate number by 1 bit right
		JC DN ; if bit not equal to 1 then go to DN
		INC ZEROS ; else increment ZEROS by one
		DN: LOOP UP
		;decrement rotation counter by 1 and if not zero then go to up
		MOV CX, ZEROS ;move result in cx register.
		MOV AH, 4CH
		INT 21H
		ENDS
		END ; end of program.
			Program Code: 4 M

	d)	**Write an ALP to subtract two BCD number using procedure.**	**4 M**
	Ans	.model small .data num1 db 13h num2 db 12h	Program Code: 4 M

```
ends
.code
 start:
        mov ax,@data
        mov ds,ax
        call sub1
        sub1 proc near
        mov al,num1
        mov bl,num2
        sub al,bl
         das
        sub1 endp
        mov ah,4ch
        int 21h
        ends
end start
end
```

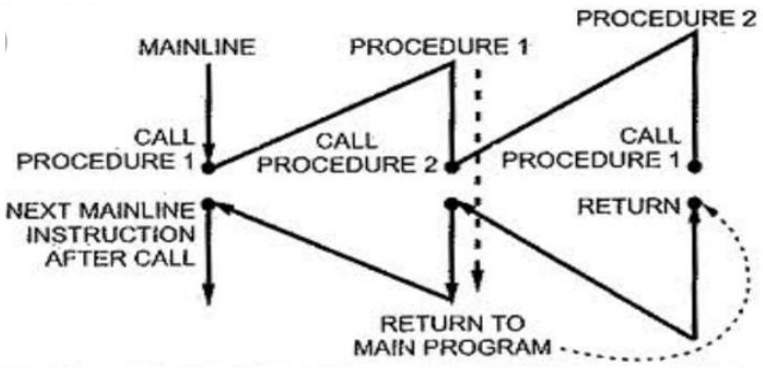
	e)	**Describe re-entrant and recursive procedure with suitable diagram.**	**4 M**
	Ans	**1)Recursive procedure**: A **recursive procedure** is procedure which calls itself. This results in the procedure call to be generated from within the procedures again and again. The **recursive procedures** keep 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.	Recursive procedure: 2 M Re-entrant procedures: 2 M



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.



5.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**(a) Calculate the physical address if:** **(i) CS 1200H and IP = DE00OH** **(ii) SS = FFOOH and SP = 0123H** **(iii) DS 1IFOOH and BX= IA00H for MOV AX, [BX]**	**6 M**
	Ans	Physical address = segment address x 10H + offset address (i) Physical address = CS X 10H + IP = 1200H X 10H + DE00H = 12000H + DE00H	Each correct answer 2 M

		= 1FE00H (ii) Physical address = SS X 10H + SP = FF00H X 10H + 0123H = FF000H + 0123H = FF123H (iii) Physical address = DS X 10H + BX = 1F00H X 10H + 1A00H = 1F000H + 1A00H = 20A00H	
	b)	**Describe how an assembly language program is developed and debugging using program developments tools.**	**6 M**
	Ans	Assembly language development tools: **EDITOR:** It is a program which helps to construct assembly language program with a file extension .asm, in right format so that the assembler will translate it to machine language. It enables one to create, edit, save, copy and make modification in source file. **Assembler:** Assembler is a program that translates assembly language program to the correct binary code. It also generates the file called as object file with extension .obj. It also displays syntax errors in the program, if any. **Linker:** It is a programming tool used to convert Object code (.OBJ) into executable (.EXE) program. It combines, if requested, more than one separated assembled modules into one executable module such as two or more assembly programs or an assembly language with C program. **Debugger:** Debugger is a program that allows the execution of program in single step mode under the control of the user. The errors in program can be located and corrected using a debugger. Debugger generates .exe file.	Each development tool 1.5 M
	c)	**State the addressing mode of following instructions:** **(i) MOV AX, 3456H** **(ii) ADD BX, [2000H]**	**6 M**

		(iii) DAA **(iv) MOV AX, [Si]** **(v) MOV AX, BX** **(vi) SUB AX, [BX +SI +80H]**	
	Ans	(i) MOV AX , 3456H ------- IMMEDIATE ADDRESSING MODE (ii) ADD BX , [2000H] ------- DIRECT ADDRESSING MODE (iii) DAA ------ IMPLIED ADDRESSING MODE (iv) MOV AX , [SI] ------- INDEXED ADDRESSING MODE (v) MOV AX , BX ------- REGISTER ADDRESSING MODE (vi) SUB AX , [BX+SI+80H] ----- BASE RELATIVE INDEX ADDRESSING MODE	Each correct answer 1 M
6.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Describe how string instructions are used to compare two strings with suitable example.**	**6 M**

	Ans	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. CX contains counter and DF=0 or 1 to auto increment or auto decrement pointer after comparing one byte/word.
		e.g.
			Explanation of string compare instruction 4 M
			And
			Example 2 M

Example	**Explanation**
CMPS m8, m8	Compares byte at address DS: SI with byte at address ES: DI and sets the status flags accordingly.
CMPS m16, m16	Compares word at address DS:SI with word at address ES:DI and sets the status flags accordingly.
CMPSB	Compares byte at address DS:SI with byte at address ES:DI accordingly.
CMPSW	Compares word at address DS:SI with word at address ES:DI and sets the status flags accordingly.

	b)	**Write an instructión to perform following operations:** **(i) Multiply BL by 88H** **(ii) Signed division of AL by BL** **(iii) Move 4000H to DS register** **(iv) Rotate content of AX register to left 4 times.** **(v) Shift the content of BX register to right 3 times.** **(vi) Load SS with FF0OH.**	**6 M**
	Ans	(1) Multiply BL by 88h **MOV AL, 88H** **MUL BL** (2) Signed division of AL by BL **IDIV BL** (3) Move 4000H to DS register **MOV DS, 4000H** (4) Rotate content of AX register to left 4 times **MOV CL,04** **ROL AX, CL** (5) Shift the content of BX register to right 3 times **MOV CL,03H** **SHR BX, CL** (6) Load SS with FF00H **MOV AX, FF00H** **MOV SS, AX**	Each correct answer 1 M
	c)	**Write an ALP to concatenate two strings.**	**6 M**
	Ans	DATA SEGMENT STR1 DB "hello$" STR2 DB "world$" DATA ENDS CODE SEGMENT START: ASSUME CS: CODE, DS:DATA MOV AX,@ DATA MOV DS, AX	Correct program 6 M

```
MOV SI, OFFSET STR1
NEXT:    MOV AL, [SI]
CMP AL,'$'
JE   EXIT
INC SI
JMP NEXT
EXIT:   MOV DI, OFFSET STR2
UP:  MOV AL, [DI]
CMP AL, "$"
JE    EXIT1
MOV [SI], AL
INC SI
INC DI
JMP UP
EXIT1: MOV AL,'$'
MOV [SI], AL
MOV AH, 4CH
INT  21H
CODE ENDS
END START
```

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 ⟵ 1 **CMC** – This instruction is used to Complement Carry Flag. CF ⟵ ~ 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**

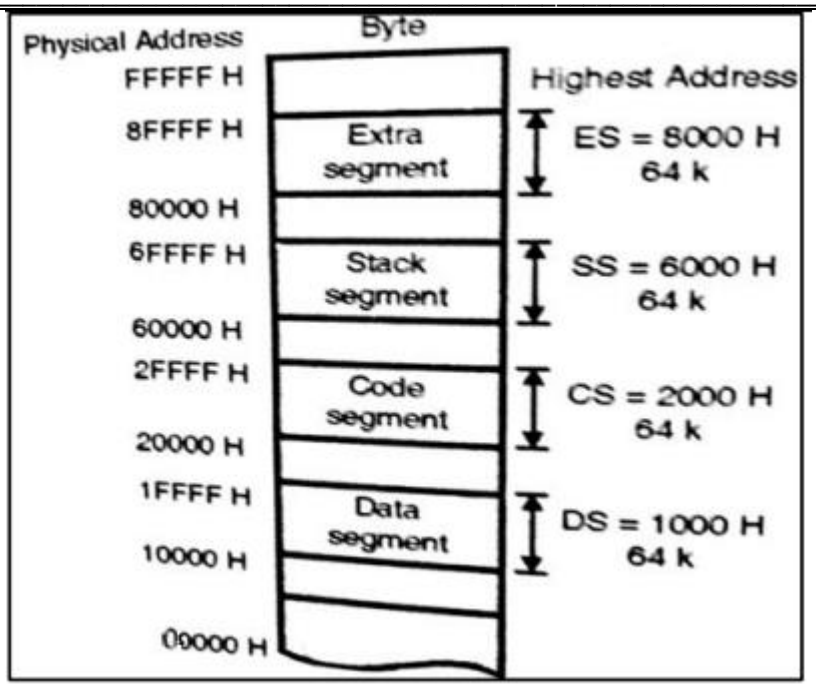
	Ans	data segment
		a dw 0202h
		b dw 0408h
		c dw ?
		data ends
		
		code segment
		assume cs:code,ds:data
		start:
		mov ax,data
		mov ds,ax
		mov ax,a
		mov bx,b
		add ax,bx
		mov c,ax
		int 03h
		code ends
		end start
			Any correct program – 2 M

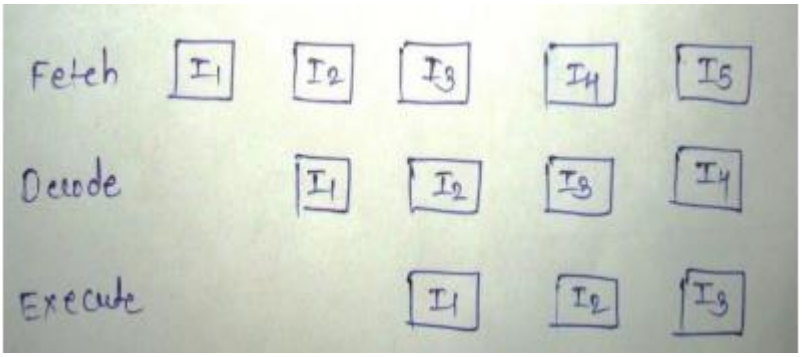
	f)	**State two examples of each, Immediate and based indexed Addressing modes.**	**2 M**
	Ans	Immediate Addressing mode: 1. MOV AX, 2000H 2. MOV CL, 0AH 3. ADD AL, 45H 4. AND AX, 0000H Based indexed Addressing mode: 1. ADD CX, [AX+SI] 2. MOV AX, [AX+DI] 3. MOV AL, [SI+BP+2000]	1 M for any two valid instructions 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**

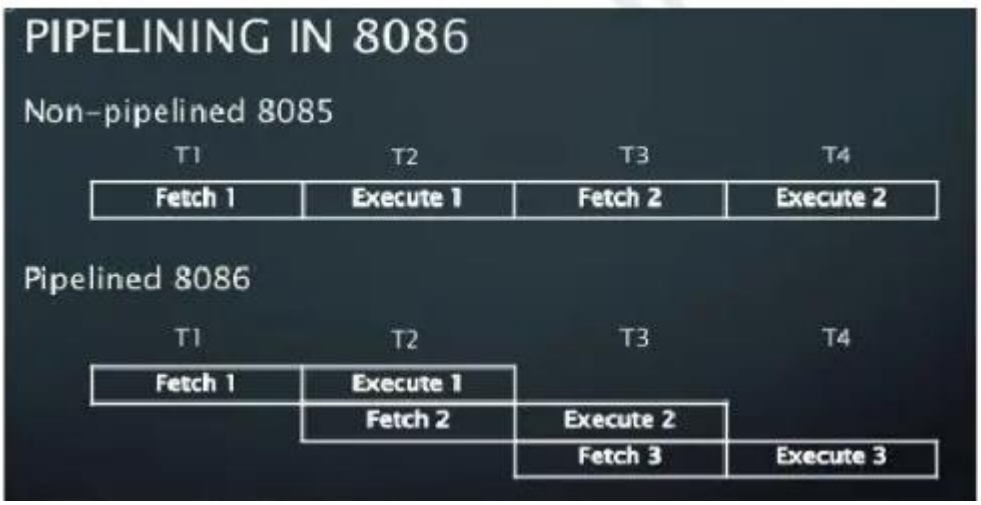
	Ans		1 M for each valid point

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

	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. 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.	Explanation- 2 M, Diagram- 2 M

			
	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 <u>THREE</u> 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		Sr.No.	**Minimum Mode**	**Maximum Mode**	Any four points- 4 M
			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	

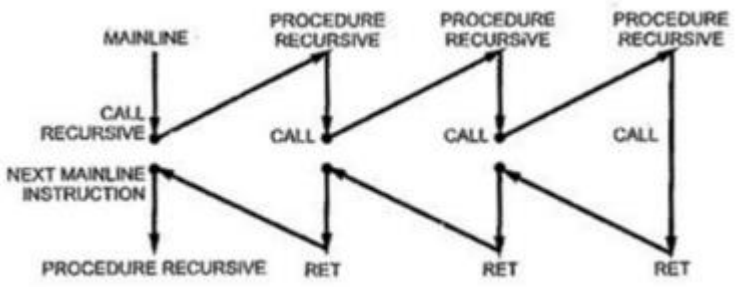
					controller and coprocessor 8087.	
			7	This circuit is simpler	This circuit is complex	
	b)	**Write an ALP for sum of series of 05 number's.**				**4 M**

	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	

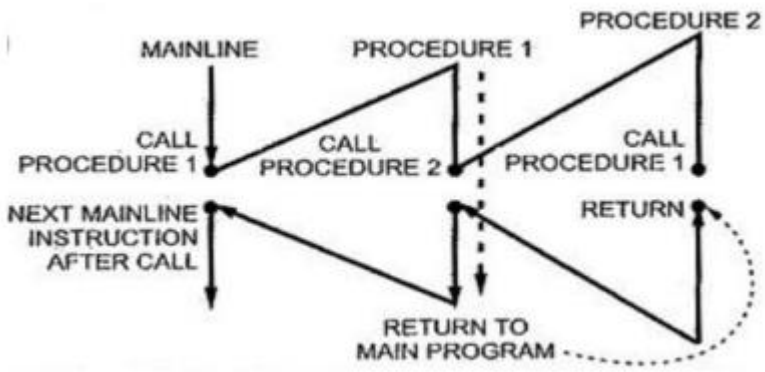
	d)	**Describe re-entrant and Recursive procedure with diagram.**	**4 M**
	Ans	A recursive procedure is procedure which calls itself. This results in the procedure call to be generated from within the procedures again and again. The recursive procedures keep on executing until the termination condition 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



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.



	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
```

(Any Same Type of Example can be considered)

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.

		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

directives- 1M

(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

	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

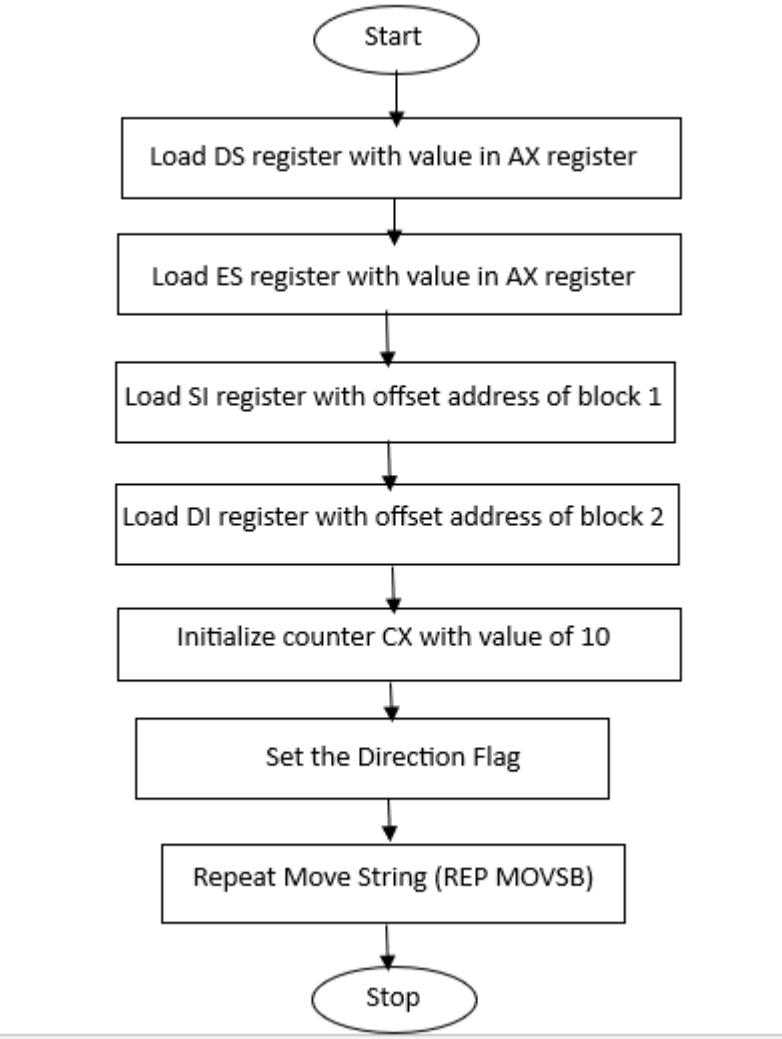
		MUL BH ;Result stored in AX ;AX = 00000000 00101101 = 2DH = 45 in decimals AAM ;AX = 00000100 00000101 = 0405H = 45 in unpacked BCD ; If ASCII values are required an OR operation with 3030H can follow this step.	
6.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Write an appropriate 8086 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

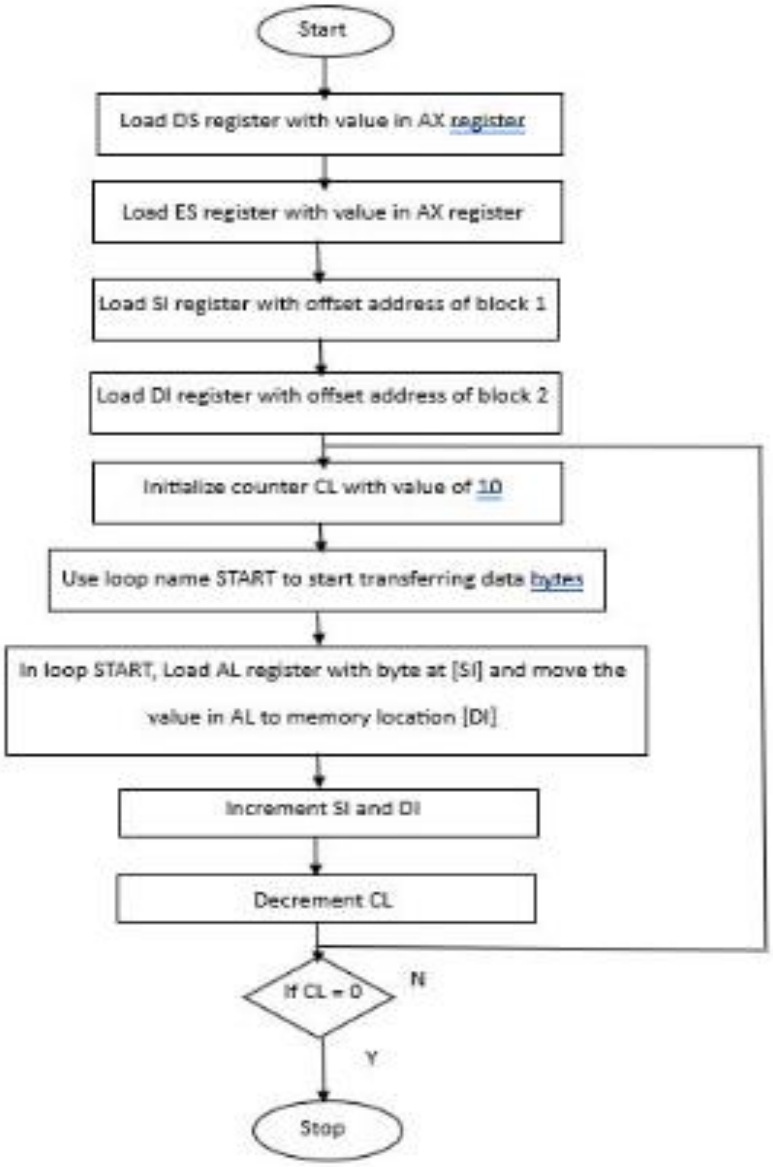

Start
Load DS register with value in AX register
Load ES register with value in AX register
Load SI register with offset address of block 1
Load DI register with offset address of block 2
Initialize counter CX with value of 10
Set the Direction Flag
Repeat Move String (REP MOVSB)
Stop

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