WINTER – 2022 EXAMINATION

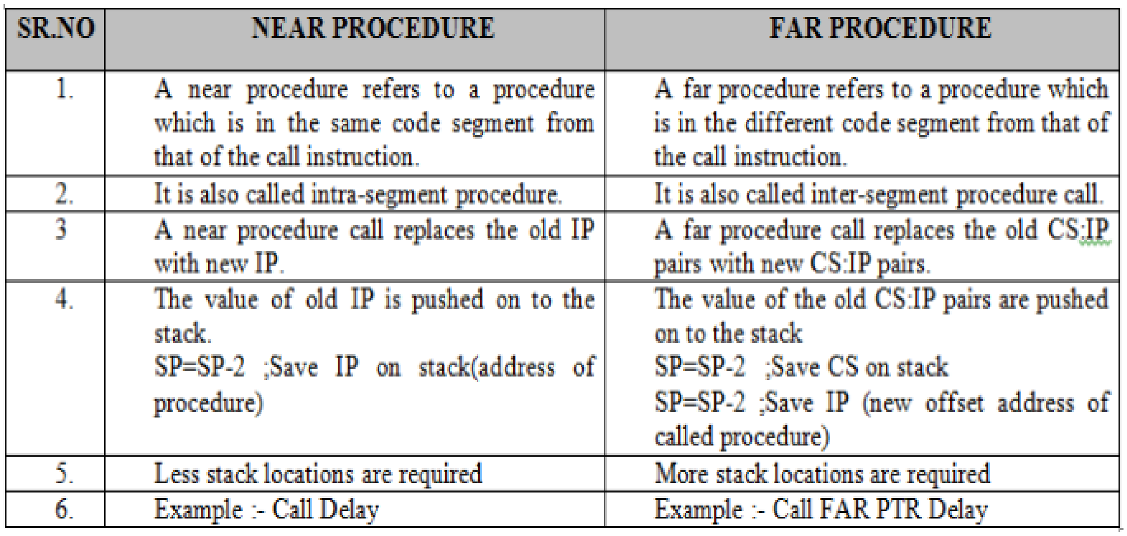
Subject Name: Microprocessor **Model Answer** **Subject Code:** 22415

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

1M- For Each Point

ROL	RCL
• Rotate left byte or word	• Rotate through carry left byte or word
• Syntax: ROL Destination, Count	• Syntax: RCL Destination, Count Can be used to Swap the nibbles Cannot be used to swap the nibbles

2. **Attempt any THREE of the following:** **12 M**

a) **What do you mean by procedure? Explain re-centrant and re-entrant procedure.** **4 M**

Ans

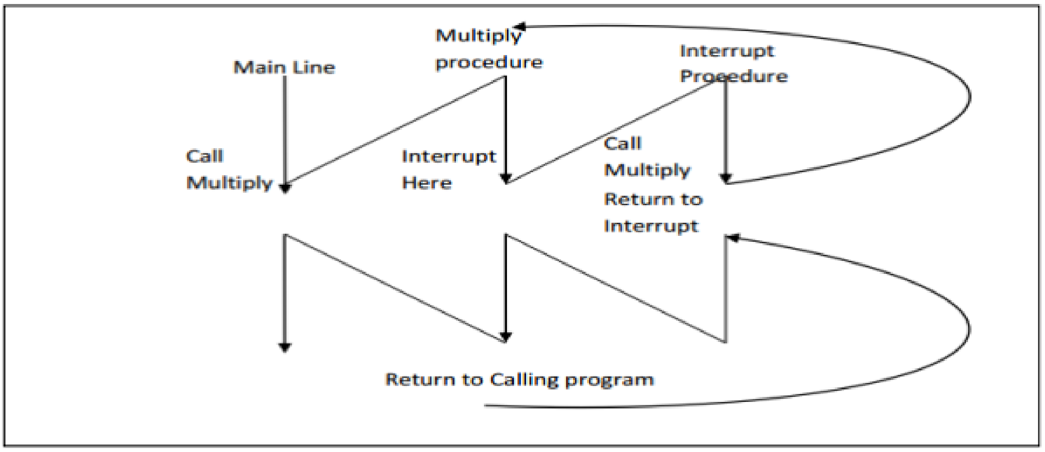
Definition 2 M

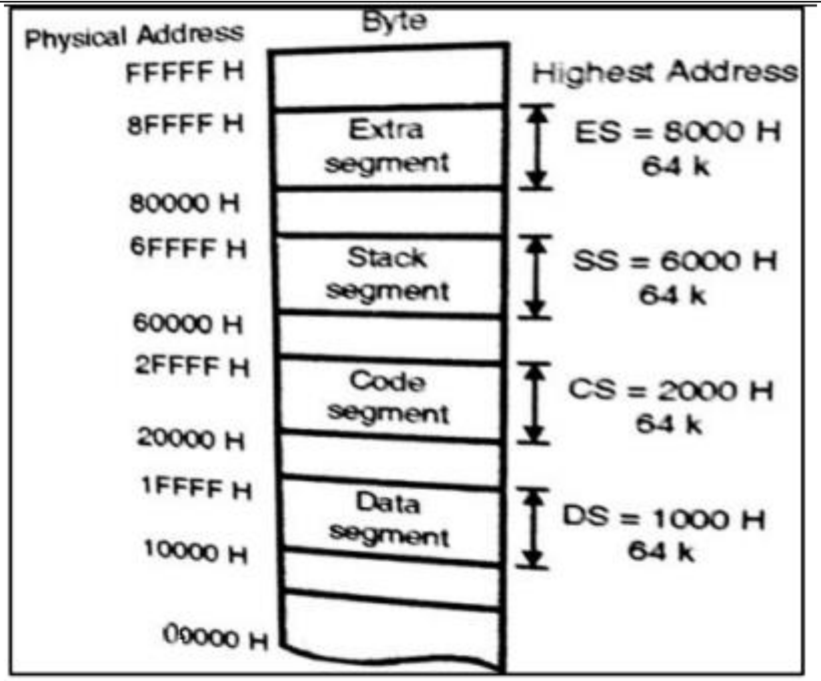
Explanation 2 M

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.

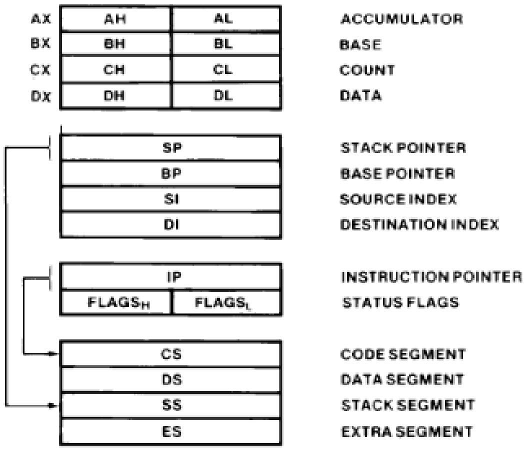
Re-entrant Procedures:

- 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.

		 Main Line Multiply procedure Interrupt Procedure Call Multiply Interrupt Here Call Multiply Return to Interrupt Return to Calling program	
	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

		 Physical Address Byte FFFFF H Highest Address 8FFFF H Extra segment ES = 8000 H 64 k 80000 H 6FFFF H Stack segment SS = 6000 H 64 k 60000 H 2FFFF H Code segment CS = 2000 H 64 k 20000 H 1FFFF H Data segment DS = 1000 H 64 k 10000 H 09000 H	
	c)	**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**

4M- For Correct Program

```
.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
    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**

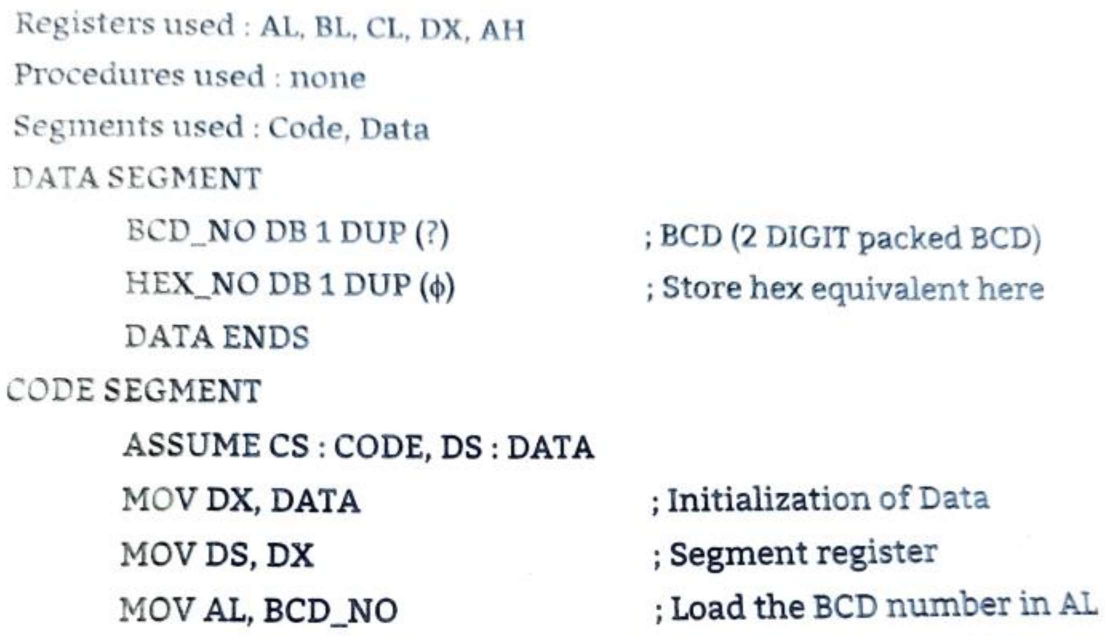
Ans **Procedure to find the factorial.**

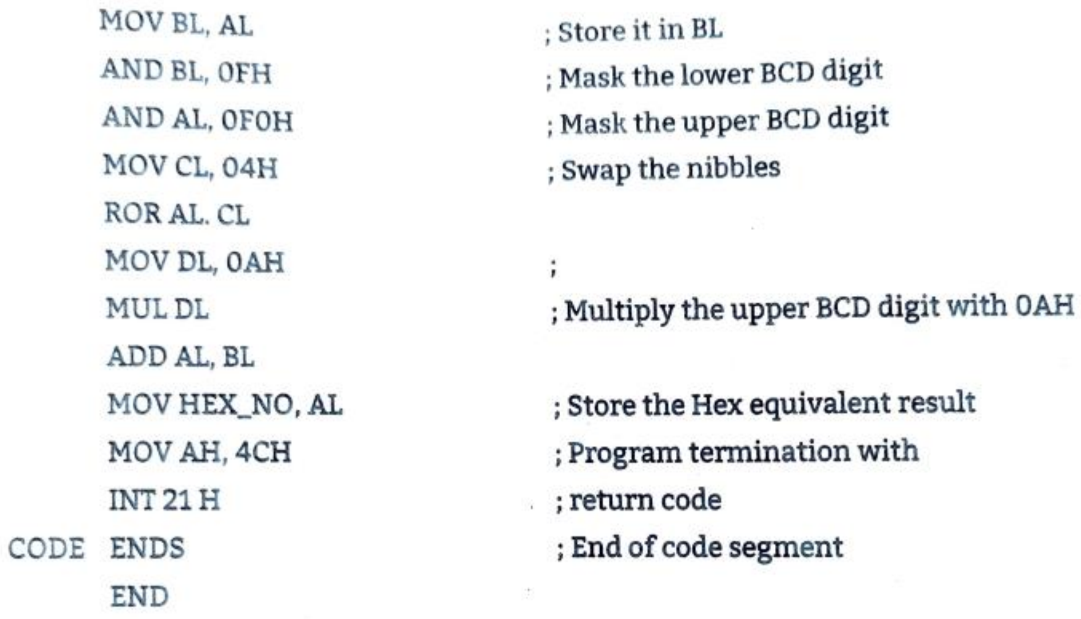
4M- For Correct Program

```
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
```

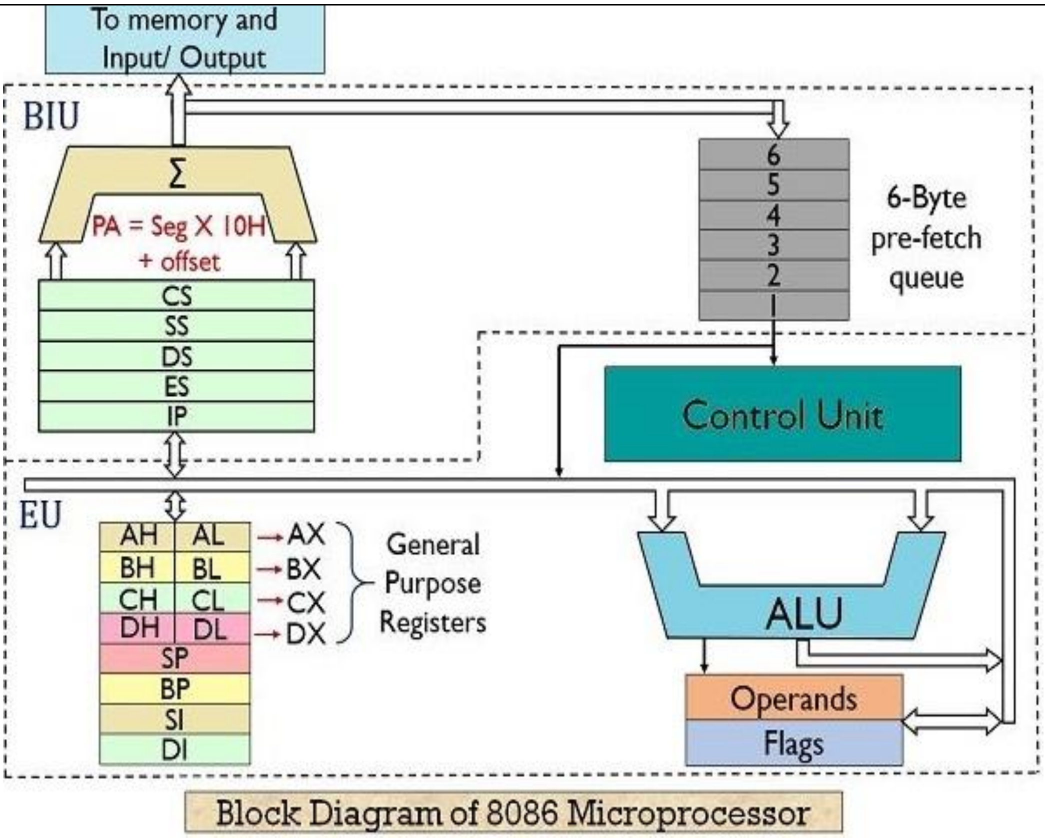
(OR)

```
DATA SEGMENT
 A DW 0005H
 FACT_LSB DW?
 FACT_MSB DW?
DATA ENDS
CODE SEGMENT
 ASSUME DS:DATA,CS:CODE
```

		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	 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	4M- For Correct Program



```
       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 **THREE** of the following:	**12 M**
	a)	**Draw functional block diagram of 8086 microprocessor.**	**4 M**
	Ans		4M-For Block Diagram

Block Diagram of 8086 Microprocessor

	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

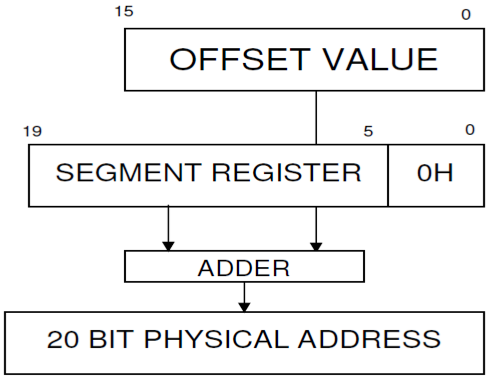
```
MOV CX, ONES                    ; move result in cx register.
MOV AH, 4CH
INT 21H
ENDS
END                             ; end of program.
```

	d)	**Write an assembly language program using MACRO to perform following operation.** **X = (A +B) * (C +D)**	**4 M**
	Ans	See program below	4M- For Correct Program

```
.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
```

		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 TWO 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  15 0 OFFSET VALUE 19 5 0 SEGMENT REGISTER 0H ADDER 20 BIT PHYSICAL ADDRESS 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.

<u>**For AL register ONLY**</u>

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.

<u>**Assume :**</u> 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**

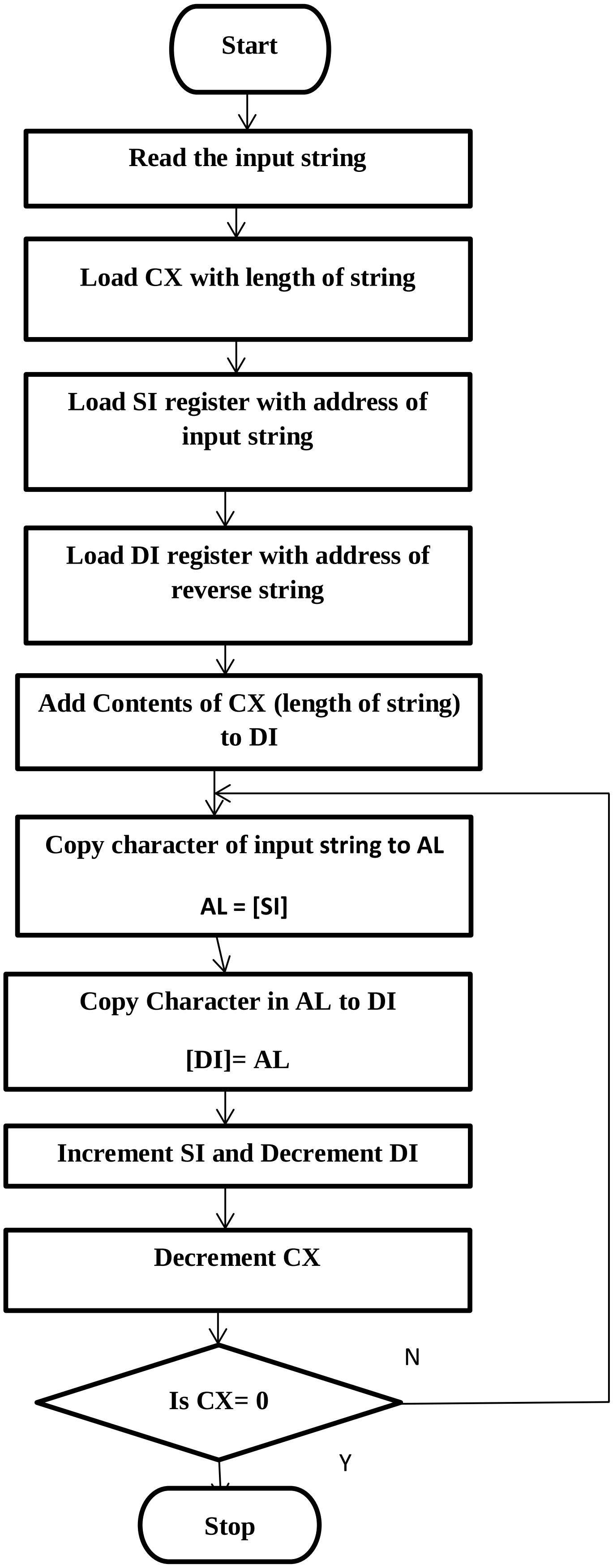
Ans

```
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
N
Is CX= 0
Y
Stop