WINTER – 2019 EXAMINATION
MODEL ANSWER

Subject: Microprocessor and Programming **Subject Code: 17431**

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.	a) (i) Ans.	**Attempt any SIX of the following:** **State the functions of temporary registers of 8085 microprocessor.** **Temp Register (8 bits)** is also called as operand register as it is used by µp for storing one of the operands during an operation and also for storing the result of any execution temproary.	**12** **2M** ***Correct function 2M***
	(ii) **Ans.**	**State the functions of following pins of 8085** **1) SOD** **2) HLDA** **1) SOD:** Serial Output data SOD pin is used to transmit data serially from accumulator to the external devices connected to the pin. **2) HLDA:** Microprocessor generates HLDA signal to acknowledge requesting device after HOLD signal.	**2M** ***Each correct function 1M***
	(iii) **Ans.**	**Define pipelining.** **Pipelining:** Process of fetching the next instruction while the current instruction is executing is called pipelining which will reduce the execution time.	**2M** ***Correct definition 2M***

(iv) **State the use of OF and DF flags of 8086 microprocessor.** **2M**

Ans.

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.

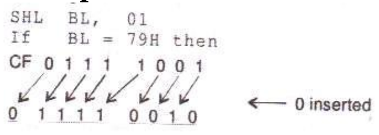
Direction Flag: It selects either increment or decrement mode for DI &/or SI register during string instructions.

Each correct use 1M

(v) **Differentiate between SHL and ROL instructions of 8086. (two points).** **2M**

Ans.

Sr. No.	**SHL**	**ROL**
1	Shift operand bits Left, Put zero in LSB(S)	Rotate left byte or word
2	**Syntax:** SHL destination, count	**Syntax** : ROL Destination, count

3	**Example:**  SHL BL, 01
	If BL = 79H then
	CF 0 1 1 1 1 0 0 1
	0 1 1 1 1 0 0 1 0
	0 inserted
		Example :
		CF=0 BL=1011 1010
		ROL BL, 1 ; Rotate all bits in BL left by one bit position.
		CF=1 BL=0111 0101

Any 2 correct points 1M each

(vi) **Enlist any four addressing modes of 8086 microprocessor.** **2M**

Ans.

Addressing modes of 8086 :

1. Immediate
2. Direct
3. Register
4. Register indirect
5. Indexed
6. Register relative
7. Based indexed
8. Relative based indexed
9. Implied

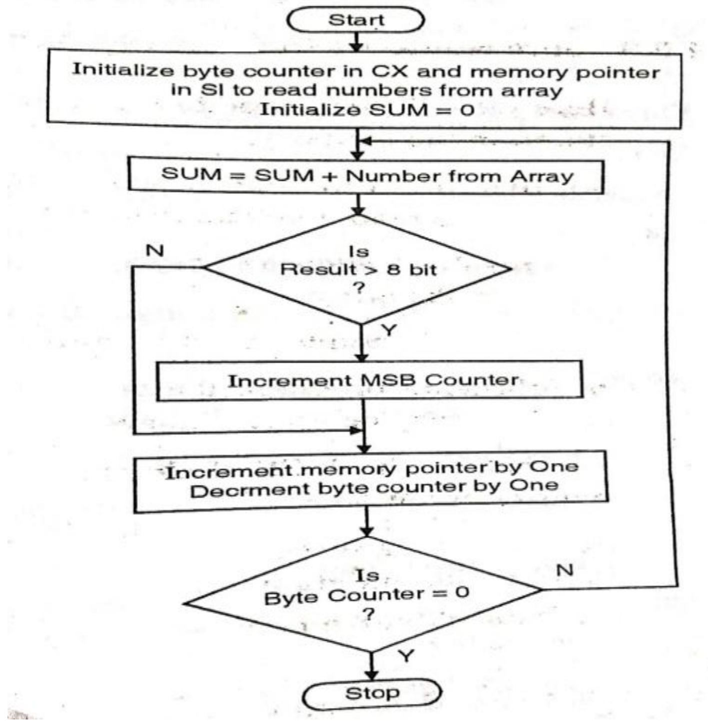
Any 4 modes ½M each

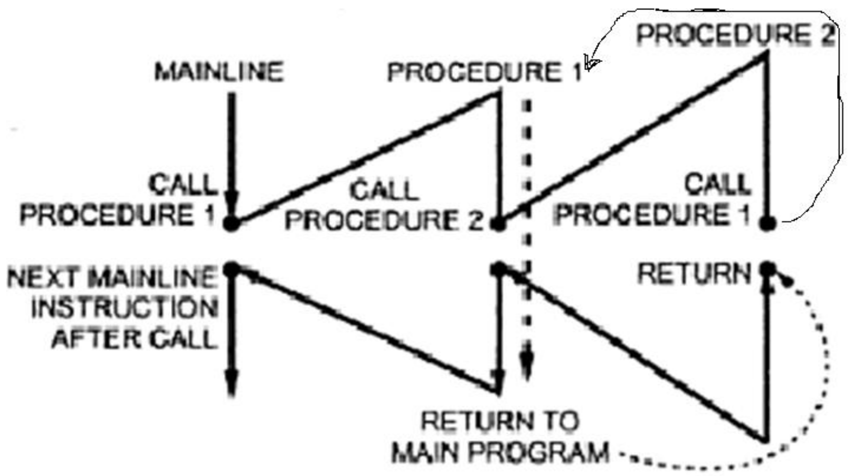
(vii) **Write an algorithm to subtract two 16 bit numbers (With borrow) in 8086 microprocessor.** **2M**

(Note: Any other logic shall be considered)

Ans.

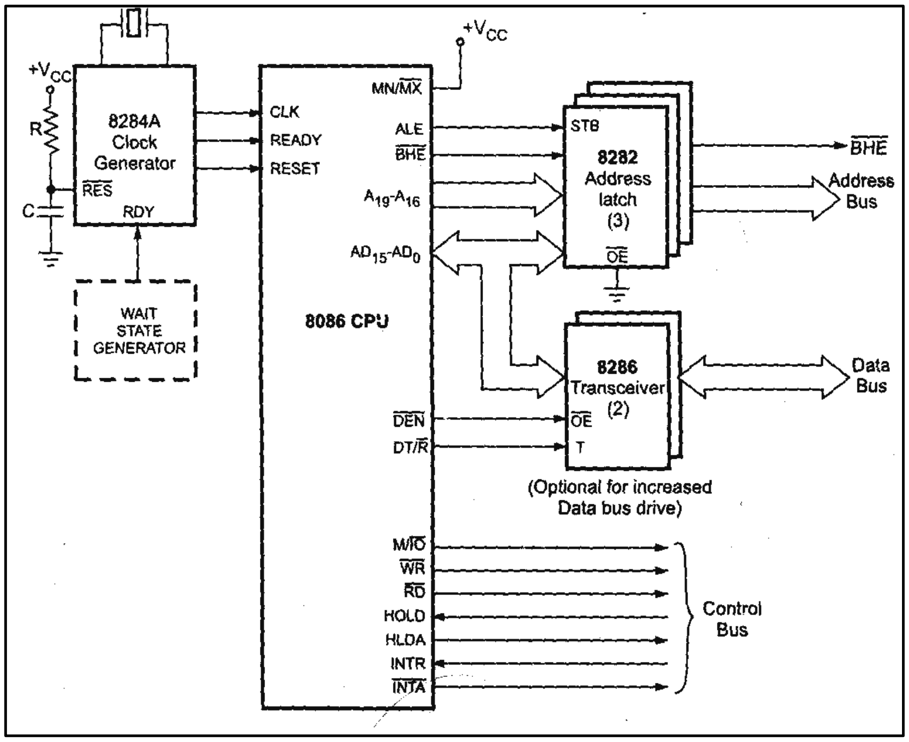
		Algorithm for 16 bit numbers subtraction with borrow: 1. Load 0000H into CX register (for borrow) 2. Load the first number into AX(accumulator) 3. Load the second number into BX register 4. Subtract BX with Accumulator AX using SUB instruction 5. Jump to 7 ,if no borrow 6. Increment CX by 1 7. Move data from AX(accumulator) to memory 8. Move data from CX register to memory 9. Stop	***Correct algorith m 2M***
	(viii) **Ans.**	**Give the syntax for defining Macro.** **Syntax:** Macro_name MACRO[arg1,arg2,…..argN) ….. Endm	**2M** ***Correct syntax 2M***
1.	**b)** **(i)** **Ans.**	**Attempt any <u>TWO</u> of the following:** **Write an algorithm and draw the flowchart to find sum of series of numbers.** ***(Note: Any other logic shall be considered)*** **Algorithim to find sum of series of numbers:** 1. Initialize data segment 2. Initialize byte counter and memory pointer to read number from array. 3. Initialize sum variable to 0 4. sum=sum+number from array 5. If sum> 8 bit then goto step 6 else step 7 6. Increment MSB result counter 7. Increment memory pointer 8. Decrement byte counter 9. If byte counter=0 then step 10 else step 4 10. Stop	**8** **4M** ***Algorith m 2M***

		 Start Initialize byte counter in CX and memory pointer in SI to read numbers from array Initialize SUM = 0 SUM = SUM + Number from Array Is Result > 8 bit ? N Y Increment MSB Counter Increment memory pointer by One Decrment byte counter by One Is Byte Counter = 0 ? N Y Stop	*Flowchart 2M*
	(ii) Ans.	**Explain the following assembler directives.** **1) DB** **2) DUP** **3) EQU** **4) ENDs.** **1) DB (Define Byte or Data Byte):** This is used to define a byte type variable. The range of values : 0 – 255 for unsigned numbers -128 to 127 for signed numbers This can be used to define a single byte or multiple bytes Ex: NUM DB? ; Allocate one memory location **2) DUP (Duplicate memory location):** This directive can be used to generate multiple bytes or words with known as well as un-initialized values. E.g TABLE DW 100 DUP(0) ; Create array of 100 words all contains data 0	**4M** *Each 1M*

		3) 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** **4) ENDs:** This directive informs the assembler the end of the segment The directives SEGMENT, ENDS are always enclosed in data, code, stack and extra segments.	
	(iii) **Ans.**	**Describe re-entrant procedure with the help of schematic diagram.** **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. • In some situation it may happen that procedure1 is called from main program, procedure2 is called from procedure1 is again called from procedure2. In this situation program execution flow reenters in the procedure1. These types of procedures are called reentrant procedures.  MAINLINE PROCEDURE 1 PROCEDURE 2 CALL PROCEDURE 1 CALL PROCEDURE 2 CALL PROCEDURE 1 NEXT MAINLINE INSTRUCTION AFTER CALL RETURN RETURN TO MAIN PROGRAM	**4M** ***Description 2M*** ***Diagram 2M***

2.		**Attempt any <u>FOUR</u> of the following:**	**16**
	a)	**Describe the functions of stack pointer and program counter of 8085.**	**4M**
	Ans.	**Stack pointer:** 1. It is a 16 bit register which is used to store the address of topmost filled memory location of stack memory. 2. SP always points current top of stack. 3. If data is stored in stack memory, the content of stack pointer is auto-decremented by two and if data is picked out from stack memory, the content of SP is auto-incremented by two. **Program counter:** 1. It maintains sequential execution of program written in memory. 2. The PC stores the address of the next instruction which is going to execute. 3. Since program counter stores the address of memory and in 8085 the address of memory is 16 bit. Hence program counter is 16 bit register.	***Any two functions 2M each***
	b) **Ans.**	**Enlist the features of 8085 microprocessor. (eight points).** **Features of 8085 microprocessor:** 1. 16 address line so 2^{16}=64 Kbytes of memory can be addressed. 2. Operating clock frequency is 3MHz and minimum clock frequency is 500 KHz. 3. On chip bus controller. 4. Provide 74 instructions with five addressing modes. 5. 8085 is 8 bit microprocessor. 6. Provides 5 level hardware interrupts and 8 software interrupts. 7. It can generate 8 bit I/O address so 2^8=256 input and 256 output ports can be accessed. 8. Requires a single +5 volt supply 9. Requires 2 phase, 50% duty cycle TTL clock 10. Provide 2 serial I/O lines, so peripheral can be interfaced with 8085 μp	**4M** ***Any eight features ½M each***
	c) **Ans.**	**Define memory segmentation. How memory segmentation is achieved in 8086? State advantages of memory segmentation.** **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.	**4M** ***Definition 1M***

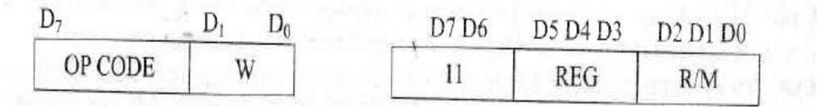
		Description: • Data segment is used to hold data, Code segment for the executable program, Extra segment also holds data specifically in strings and stack segment is used to store stack data. • Each segment is 64Kbytes & addressed by one segment register. i.e CS,DS,ES or SS • The 16 bit segment register 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. • Since the memory size of 8086 is 1Mbytes, total 16 segments are possible with each having 64Kbytes. **Advantages of segmentation:** 1) With the use of segmentation the instruction and data is never overlapped. 2) The major advantage of segmentation is Dynamic relocatability of program which means that a program can easily be transferred from one code memory segment to another code memory segment without changing the effective address. 3) Segmentation can be used in multi-user time shared system. 4) Segmentation allows two processes to share data. 5) Segmentation allows you to extend the addressability of a processor i.e., address up to 1MB although the actual addresses to be handled are of 16 bit size. 6) Programs and data can be stored separately from each other in segmentation.	***Explanation 2M*** ***Any 2 advantages ½M each***
	d) **Ans.**	**Draw typical 8086 minimum mode configuration and explain function of any two signals used in minimum mode.**	**4M**

		 +Vcc R C 8284A Clock Generator RES RDY WAIT STATE GENERATOR CLK READY RESET MN/MX ALE BHE A19-A16 AD15-AD0 8086 CPU DEN DT/R M/IO WR RD HOLD HLDA INTR INTA STB 8282 Address latch (3) OE 8286 Transceiver (2) T (Optional for increased Data bus drive) BHE Address Bus Data Bus Control Bus	*Diagram 2M*
		MN/$\overline{MX}$ • This pin indicates operating mode of 8086, minimum or maximum. • When this pin connected to 1)**Vcc**, the processor operates in **minimum** mode, 2) **ground,** processor operates in **maximum** mode. **$\overline{\text{INTA}}$ (Interrupt acknowledge)** • It is active low output signal. • When processor receive INTR signal, processor complete current machine cycle, and acknowledge interrupt by generating this signal. **ALE(Address Latch Enable)** • It is active high, pulse issued by processor during T1 state of bus cycle to indicate availability of valid address on AD0-AD15. • This pin is connected to latch enable pin of latch 8282 or	*Any 2 signals 1M each*

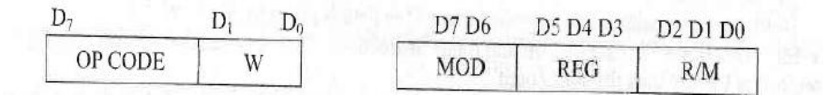
		74LS373. **$\overline{\text{DEN}}$ (Data enable)** • It is an active low signal, issued by processor during middle of T2 until middle of T4, to indicate availability of valid data over AD0-AD15. • This signal is used to enable transreceivers (bi-directional buffers) 8286 or 74LS245 to separate data from multiplexed address/data signal. **DT/ $\overline{\text{R}}$** • This output signal used to decide the direction of data flow through transreceivers (bi-directional buffers) 8286 or 74LS245 • When processor sends data out, this signal is high, when processor receives data, this signal is low. **M/ $\overline{\text{IO}}$** • This signal is issued by processor to distinguish memory access from I/O access. • When this signal high memory is accessed and when this signal is low , an I/O device is accessed • $\overline{WR}$ • It is an active low signal used to write data to memory or I/O device depending on status of M/$\overline{\text{IO}}$. **HLDA:** • This is an active high output signal generated by processor after receiving HOLD signal. **HOLD:** • When another master device needs the use of the address, data, control bus, it sends a HOLD request to the processor through this line. It is an active high input signal.	
	e)	**State the functions of the following pins of 8086.** **(i) MN/$\overline{\text{MX}}$** **(ii) NMI** **(iii) INTR** **(iv) $\overline{\text{LOCK}}$**	**4M**

	Ans.	**(i) MN/$\overline{\text{MX}}$:** This signal indicates operating mode of 8086, minimum or maximum. When this pin connected to 1) Vcc, the processor operates in minimum mode, 2) Ground, processor operates in maximum mode. **(ii) NMI:** An edge triggered signal on this pin causes 8086 to interrupt the program it is executing and execute Interrupt service Procedure corresponding to Type-2 interrupt.NMI is Non-maskable by software **(iii) INTR (Interrupt Request):** 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. **(iv) $\overline{\text{LOCK}}$:** Prevent other processor to take the control of shared resources. Lock the bus attached to lock pin of device while a multicycle instruction completes. The lock prefix this allows a microprocessor to make sure that another processor does not take control of system bus while it is in the middle of a critical instruction.	***Each correct function 1M***
	f) Ans.	**Enlist the instruction formats used in 8086. Describe any one of them.** **Instruction formats of 8086:** 1) One byte Instruction 2) Register to Register 3) Register to/from memory with no displacement 4) Register to/from Memory with Displacement 5) Immediate operand to register. 6) Immediate operand to memory with 16-bit displacement **1) One byte Instruction**: This format is only one byte long and may have the implied data or register operands. The least significant 3 bits of the opcode are used for specifying the register operand, if any. Otherwise, all the eight bits form an opcode and the operands are implied.	**4M** ***List 2M*** ***Description of any one 2M***

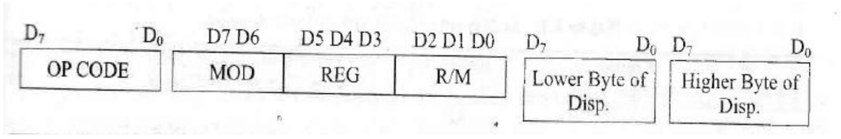
2) Register to Register: This format is 2 bytes long. The first byte of the code specifies the operation code and the width of the operand specifies by *w* bit. The second byte of the opcode shows the register operands and *R/M* field.



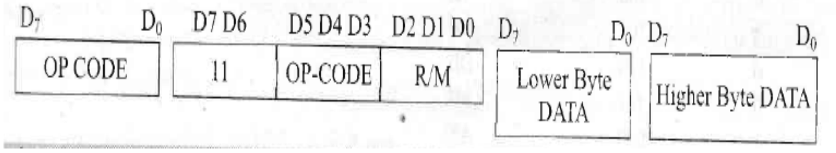
3) Register to/from memory with no displacement: This format is also 2 bytes long and similar to the register to register format except for the MOD field.



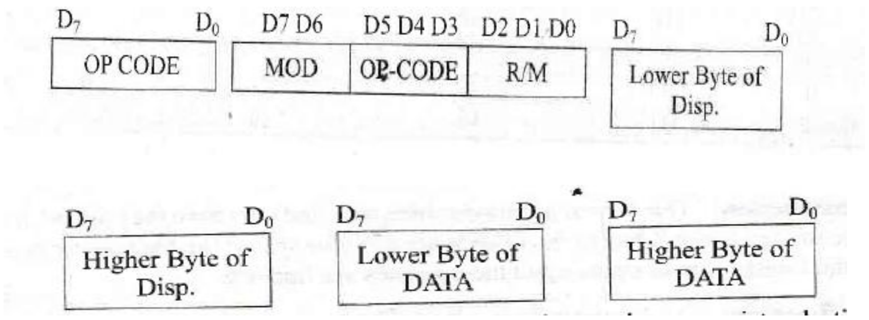
4) Register to/from Memory with Displacement :
This type of instruction format contains one or two additional bytes for displacement along with 2-byte the format of the register to/from memory without displacement.



5) Immediate operand to register
In this format, the first byte as well as the 3 bites from the second byte which are used for *REG* field in case of register to register format are used for opcode. It also contains one or two bytes of immediate data.



6) Immediate operand to memory with 16-bit displacement: This type of instruction format requires 5 to 6 bytes for coding. The first two bytes contain the information regarding OPCODE, MOD and R/M fields. The remaining 4 bytes contain 2 bytes of displacement and 2 bytes of data.



3. **Attempt any <u>FOUR</u> of the following:** **16**

a) **Describe the concept of pipelining in 8086.** **4M**

(Note: Only Explanation OR explanation with diagram shall also be considered).

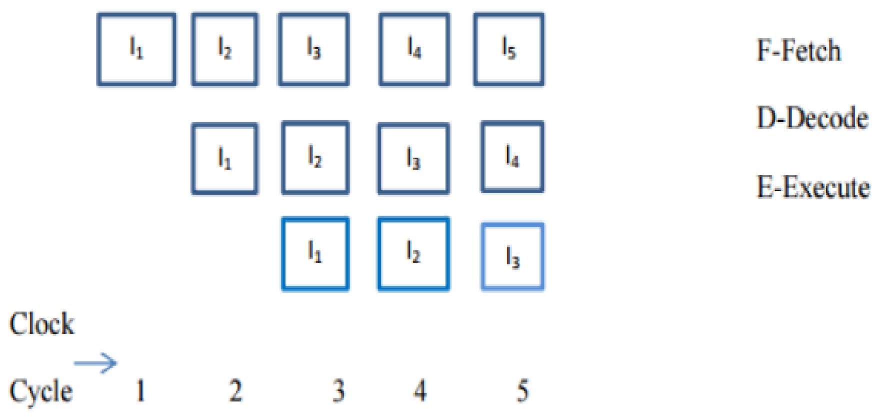
Ans. Concept of pipelining in 8086.

- Process of fetching the next instruction while the current instruction is executing is called pipelining. This reduces the execution time.
- In 8086, pipelining is implemented by providing 6 byte queue in BIU.
- 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.
- So, while executing first instruction in a queue, processor decodes second instruction and fetches 3rd instruction from the memory.
- In this way, 8086 performs fetch, decode and execute operation in parallel i.e. in single clock cycle and it is called pipelining.
- This avoids the waiting time for execution unit to receive other instruction. And increases the speed of operation.

Correct explanation 4M

Concept of pipelining through diagram: **3 instructions** are executed in **5 clock cycles** through pipelining as shown below

Diagram:



b) **Differentiate between minimum and maximum mode of 8086. (four points).** **4M**

Ans.

Sr. No.	Minimum Mode	Maximum mode
1	MN/$\overline{MX}$ pin is connected to Vcc. i.e. MN/$\overline{MX}$ = 1.	MN/$\overline{MX}$ pin is connected to ground. i.e. MN/$\overline{MX}$ = 0.
2	Control system M/$\overline{IO}$, $\overline{RD}$, $\overline{WR}$ is available on 8086 directly.	Control system M/$\overline{IO}$, $\overline{RD}$, $\overline{WR}$ 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 $\overline{IOR}$, $\overline{IOW}$, $\overline{MEMW}$, $\overline{MEMR}$ can be generated using control signals M/$\overline{IO}$, $\overline{RD}$, $\overline{WR}$ which are available on 8086 directly.	Control signals such as $\overline{MRDC}$, $\overline{MWTC}$, $\overline{AMWC}$, $\overline{IORC}$, $\overline{IOWC}$, and $\overline{AIOWC}$ are generated by bus controller 8288.
6	ALE, $\overline{DEN}$, DT/$\overline{R}$ and $\overline{INTA}$ signals are directly available.	ALE, $\overline{DEN}$, DT/$\overline{R}$ and $\overline{INTA}$ signals are not directly available and are generated

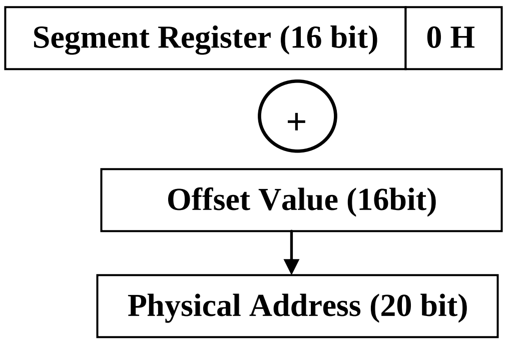
Any four points 1M each

		by bus controller 8288.		
7	HOLD and HLDA signals are available to interface another master in system such as DMA controller.	$\overline{RQ}$ / $\overline{GTQ}$ and $\overline{RQ/GT1}$ signals are available to interface another master in system such as DMA controller and coprocessor 8087.		
8	Status of the instruction queue is not available.	Status of the instruction queue is available on pins QS_0 and QS_1.		

	c)	**Differentiate RCL and RCR instructions on the basis of** **(i) Syntax** **(ii) Operation** **(iii) Example** **(iv) Status of carry flag.** ***(Note: Any other example with any other register shall also be considered).***	**4M**
	Ans.	(table below)	***Each difference 1M***

Factor	**RCL**	**RCR**
Syntax	RCL Register/Memorylocation, Count	RCR Register/memorylocation, Count
Operation	Rotate contents of the register bitwise to the Left 'Count' number of times through carry.	Rotate contents of the register bitwise to the Right 'Count' number of times through carry.
Example	RCL BL, 01	MOV CL,03 RCR BL,CL
Status of Carry Flag	Carry flag will contain the contents of Most Significant Bit (MSB) of the register obtained from the last shift.	Carry flag will contain the contents of Least Significant Bit (LSB) of the register obtained from the last shift.

	d)	**Analyze the content of AL register and status of carry and auxiliary carry flag after the execution of following instructions.**	**4M**

	   Ans.	**MOV AL, 34H** **ADD AL, 12H** **DAA** Given Instructions, MOV AL,34H ADD AL,12H DAA After the instructions, AL = 34 + 12 = 46H Carry Flag = 0, since no carry generated. Auxiliary carry Flag = 0 Since no carry generated from D3 to D4 bit.	***Result 1M*** ***Status of flags with reason 3M***
	e) **Ans.**	**Describe the concept of physical address generation on 8086. If CS = 4312H and IP=5387 H. Calculate physical address.** **Concept of physical address generation in 8086:** **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.  Segment Register (16 bit) 0 H + Offset Value (16bit) Physical Address (20 bit) Given, **CS = 4312H and IP = 5387H** **Appending 4 zero's to CS register,** **4 3 1 2 0 +** **5 3 8 7** **________** **4 8 4 A 7 H** **-------------** **Physical Address = 484A7 H.**	**4M** ***Concept 2M*** ***Calculation 2M***

	f)	**Write an 8086 assembly language program to find smaller of two 8 bit numbers.** ***(Note: Any other logic shall be considered).***	4M

	Ans.	DATA SEGMENT
		        A DB 23H
		        B DB 78H
		        SMA DB ?
		DATA ENDS
		CODE SEGMENT
		ASSUME CS: CODE, DS: DATA
		START : MOV AX, DATA
		                MOV DS, AX
		                MOV AL, A
		                MOV BL, B
		                CMP AL, BL
		                JC SKIP
		                MOV SMA, B
		                JMP EXIT
		        SKIP: MOV SMA, A
		        EXIT: MOV AH,4CH
		                INT 21H
		CODE ENDS
		END START
			Correct program 4M

4.		**Attempt any <u>FOUR</u> of the following:**	**16**
	a)	**State the functions of AAA and AAS instructions of 8086 with example of each.**	**4M**
	Ans.	**AAA Instruction:** Syntax :-- AAA • This instruction is used to convert the result in AL after the addition of ASCII operands to decimal. • Example :-- MOV AH,00H MOV AL,’5’ ; AL ← 35 ADD AL,’7’ ; AL ← 6Ch ←35+37 AAA ; AX ← 0102H **AAS Instruction :** Syntax :-- AAS • This instruction is used to convert the result in AL after the	***Each Instruction: Function 1M*** ***Each example 1M***

		subtraction of ASCII operands to decimal. • If after AAS instruction, the number in the register AH is 00, the result is positive and ASCII code of the result is present in AL. • If after AAS instruction the number in register AH is FFH, then it indicates the result is negative in 10s complement form. • Example :-- MOV AH,00H MOV AL,'8' ; AL ← 38 SUB AL,'2' ; AL ← 06h ←38-32 AAS ;AL ← 06H	
	b) Ans.	**Identify the addressing modes of following 8086 instructions.** **(i) MOV Bx, 0354H** **(ii) ADD AL, [Bx+04]** **(iii) MOV Ax, [Bx+SI]** **(iv) MOV Ax, [Bx+SI+04].** **(i) MOV Bx, 0354H:** Answer: **Immediate Addressing Mode** **(ii) ADD AL, [Bx+04]:** Answer: **Relative Based Addressing Mode/Base addressing mode with displacement.** **(iii) MOV Ax, [Bx+SI]:** Answer: **Based Indexed Addressing Mode** **(iv) MOV Ax, [Bx+SI+04]:** Answer: **Relative Based Indexed Addressing Mode/ Based Indexed addressing mode with displacement.**	**4M** ***Each correct answer 1M***
	c) Ans.	**Write an 8086 assembly language program to find two's complement of 16 bit number.** ***(Note: Any other logic shall be considered)*** **Program for finding 2's complement:** DATA SEGMENT A DW 1234H C DW ? DATA ENDS CODE SEGMENT ASSUME CS:CODE, DS:DATA START: MOV AX, DATA MOV DS, AX MOV AX, A	**4M** ***Correct program 4M***

		```
		NOT AX
		INC AX
		MOV C, AX
		MOV AH, 4CH
		INT 21H
		CODE ENDS
		END START
		```
	d) Ans.	**Write an 8086 assembly language program to count number of 1's in 8 bit number** ***(Note: Any other logic shall be considered)*** **Program for finding number of 1's :** DATA SEGMENT A DB 34H C DB 0H DATA ENDS CODE SEGMENT ASSUME DS:DATA, CS:CODE START: MOV AX, DATA MOV DS, AX MOV AL, A MOV CL, 08H NEXT: SHR AL, 01H JNC SKIP INC C SKIP: LOOP NEXT MOV AH,4CH INT 21H CODE ENDS END START	**4M** ***Correct program 4M***
	e) Ans.	**Write an 8086 assembly language programme to find length of a string.** ***(Note: Any other logic shall be considered)*** **Program for finding length of a string:** DATA SEGMENT S DB 'MSBTE$' L DB 0H DATA ENDS CODE SEGMENT ASSUME CS:CODE, DS:DATA	**4M** ***Correct program 4M***

		```
		START:      MOV AX, DATA
		            MOV DS,AX
		            MOV SI, OFFSET S
		       UP:  MOV AL,[SI]
		            CMP AL, '$'
		            JZ EXIT
		            INC L
		            INC SI
		            JMP UP
		     EXIT:  MOV AH, 4CH
		            INT 21H
		CODE ENDS
		END START
		```

	f)
	
	Ans.
		Describe with suitable example how a parameter is passed in register in 8086 assembly language procedure.
		Parameter passing in Procedure:
		• Procedures may require input data or constants for their execution.
		• Their data or constants may be passed to the procedure by the main program or some procedures may access the readily available data of constants available in memory.
		• The parameter can be passed through the register as given in the following example.
		• Here, registers AX and BX are passed to the procedure, where their contents are added.
		• When the procedure is called using CALL instruction, the instructions in the procedure are executed.
		• The registers AX and BX are added.
		• Thus the data from the main program are used by the procedure using registers.
		```
		CODE SEGMENT
		     START: MOV AX, 5555H
		            MOV BX, 7272H
		                :
		                :
		            CALL PROC1
		                :
		                :
		```
			4M
			
			Description 2M
			
			Example 2M

		PROCEDURE PROC1 : : ADD AX,BX : : RET **PROC1 ENDP** CODE ENDS END START	
5.	**a)** **Ans.**	**Attempt any <u>FOUR</u> of the following:** **State two instructions each for arithmetic multiplication and division with example.** **Instruction to perform multiplication:** • **MUL** − Used to multiply unsigned byte by byte/word by word. Example: MOV AL, 200 ; AL = 0C8h MOV BL, 4 MUL BL ; AX = 0320h (800) RET • **IMUL** − Used to multiply signed byte by byte/word by word. Example: MOV AL, -2 MOV BL, -4 IMUL BL ; AX = 8 RET • **AAM** − Used to adjust ASCII codes after multiplication. Example: MOV AL, 15 ; AL = 0Fh AAM ; AH = 01, AL = 05 RET **Instructions to perform division** • **DIV** − Used to divide the unsigned word by byte or unsigned double word by word. Example: MOV AX, 203 ; AX = 00CBh MOV BL, 4 DIV BL ; AL = 50 (32h), AH = 3 RET • **IDIV** − Used to divide the signed word by byte or signed double word	**16** **4M** ***Any two instructions of multiplication 1M and Example 1M*** ***Any two instructions of division 1M and Example 1M***

		by word.
		Example:
		`MOV AX, -203 ; AX = 0FF35h`
		`MOV BL, 4`
		`IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)`
		`RET`
		
		• **AAD** − Used to adjust ASCII codes after division.
		Example:
		`MOV AX, 0105h ; AH = 01, AL = 05`
		`AAD ; AH = 00, AL = 0Fh (15)`
		`RET`	

	b) Ans.	**Write an 8086 assembly language program to arrange five 8 bit numbers in ascending order.** ***(Note: Any other logic may be used)***	4M ***Correct program 4M***

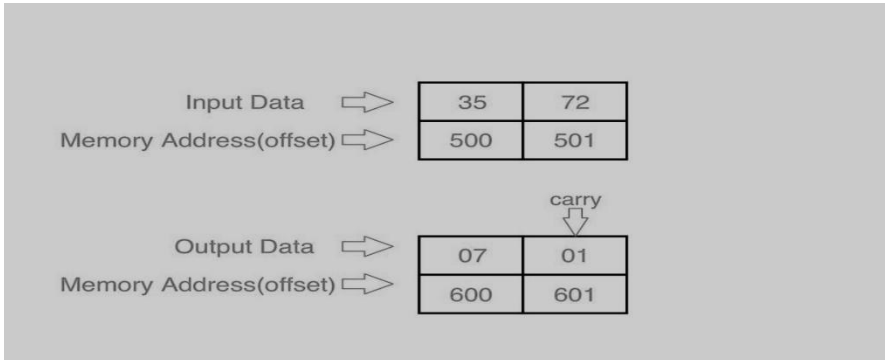
```
        Data segment                ;       start of data segment
                Array db 15h,05h,08h,78h,56h
        Data ends                   ;       end of data segment
        Code segment                        ;       start of code
        segment
        Start:  assume      cs: code,       ds:  data
                mov dx, data    ;    initialize data segment
                mov ds, dx
                mov bl,05h          ;    initialize pass counter to read
                                                   numbers from  array
        step1:      mov si,offset array         ;       initialize memory
                                                pointer to read number
                mov  cl,04h                 ;  initialize byte counter
step:           mov al,[si]
                cmpal,[si+1]            ;  compare two numbers
                 jc      down             ;  if number <next no. Then
                go to down
                xchg al,[si+1]          ;       interchange numbers
                  xchg al,[si]
         Down :         add si,1                        ;       increment
                                         memory pointer to point next
                    loop step            ;       decrement byte counter
                                                if count is ? 0then step
                  dec bl                 ;       decrement pass counter
                                                        if ? 0 then step1
```

```
          jnz step1
    Code ends
    End start
```

c) **Write an 8086 assembly language program to add two BCD numbers.** **4M**

(Note: Program without carry can also be considered)

Ans.



Program –

MEMORY ADDRESS	**MNEMONICS**	**COMMENT**
400	MOV AL, [500]	AL<-[500]
404	MOV BL, [501]	BL<-[501]
408	ADD AL, BL	AL<-AL+BL
40A	DAA	DECIMAL ADJUST AL
40B	MOV [600], AL	AL->[600]
40F	MOV AL, 00	AL<-00
411	ADC AL, AL	AL<-AL+AL+cy(prev)
413	MOV [601], AL	AL->[601]
417	HLT	END

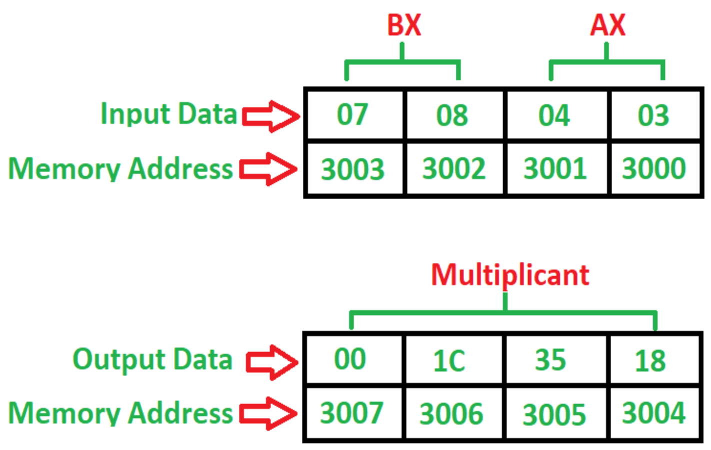
Correct program 4M

OR

```
ASSUME CS: CODE , DS:DATA
DATA SEGMENT
                OP1 EQU 92H
                OP2 EQU 52H
                RESULT DB 02 DUP(00)
DATA ENDS
CODE SEGMENT
START:
                MOV AX,DATA
                MOV DS,AX
                MOV BL,OP1
                XOR AL,AL
                MOV AL,OP2
                ADD AL,BL
                DAA
                MOV RESULT ,AL
                JNC MSBO
                INC [RESULT+1]
MSBO:           MOV AH,4CH
                INT 21H
CODE ENDS
END START
```

d) **Write an 8086 assembly language program to multiply two 16 bit unsigned numbers.** **4M**

Ans.



Correct program 4M

		<table><tr><th>MNEMONICS</th><th>OPERANDS</th><th>COMMENT</th></tr><tr><td>MOV</td><td>AX, [3000]</td><td>[AX] <- [3000]</td></tr><tr><td>MOV</td><td>BX, [3002]</td><td>[BX] <- [3002]</td></tr><tr><td>MUL</td><td>BX</td><td>[AX] <- [AX] *</td></tr><tr><td>MOV</td><td>[3004], AX</td><td>[3004] <- AX</td></tr><tr><td>MOV</td><td>AX, DX</td><td>[AX] <- [DX]</td></tr><tr><td>MOV</td><td>[3006], AX</td><td>[3006] <- AX</td></tr><tr><td>HLT</td><td></td><td>Stop</td></tr></table>	
	e) **Ans.**	**Describe MACRO with suitable example.** **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. • When assembler encounters a Macro name later in the source code, the block of code associated with the Macro name is substituted or expanded at the point of call, known as macro expansion. • Macro called as open subroutine. **Syntax:** Macro_nameMACRO [arg1,arg2,.....argN) ..... endMacro ***Example:*** MyMacro MACRO p1, p2, p3 ; macro definition with arguments MOV AX, p1	**4M** ***Description 2M***

		MOV BX, p2 MOV CX, p3 ENDM ;indicates end of macro. data segment data ends code segment assume cs:code,ds:data start: mov ax,data mov ds,ax MyMacro 1, 2, 3 ; macro call MyMacro 4, 5, DX mov ah,4ch int 21h code ends end start **OR** **(Any Same Type of Example can be considered)**	***Example 2M***
	f) **Ans.**	**State the function of CALL and RET with suitable example.** **CALL :** **Functions of CALL:** **1.** **CALL** pushes the **return** address onto the stack and **2.** Transfers control to a procedure. **OR** The **CALL instruction** is used whenever we need to make a call to some procedure or a subprogram. Whenever a **CALL** is made, the following process takes place inside the microprocessor: The address of the next instruction that exists in the caller program (after the program CALL instruction) is stored in the stack. • The instruction queue is emptied for accommodating the instructions of the procedure. • Then, the contents of the instruction pointer (IP) is changed with	**4M** ***Two functions of CALL and RET 1M each*** ***Example 1M each***

the address of the first instruction of the procedure.
- The subsequent instructions of the procedure are stored in the instruction queue for execution.

Example:

```
        ORG 100h            ;       for COM file.
        CALL p1
        ADD AX, 1

        RET                 ;       return to OS.

        p1      PROC        ;       procedure declaration.
                MOV AX, 1234h
                RET         ;       return to caller.
        p1      ENDP
```

RET :
Functions of RET:

1. **RET** pops the **return** address off the stack and
2. Returns control to **that** location.

OR

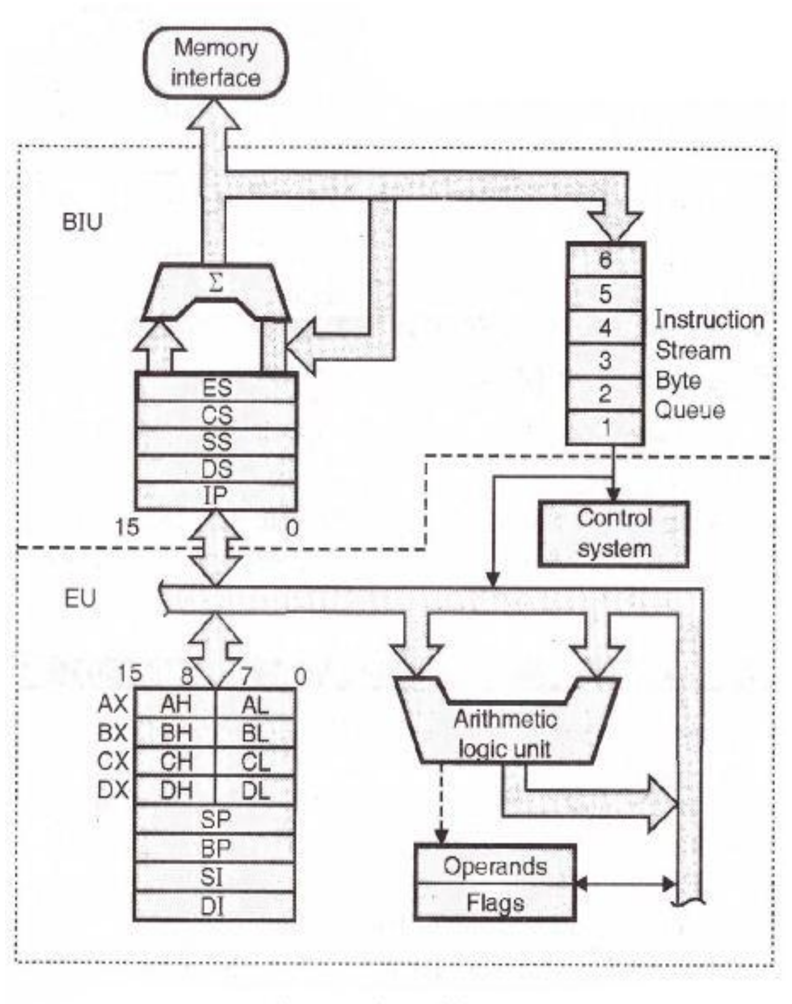
The **RET instruction** stands for return. This instruction is used at the end of the procedures or the subprograms. This instruction transfers the execution to the caller program. Whenever the **RET instruction** is called, the following process takes place inside the microprocessor:
- The address of the next instruction in the mainline program which was previously stored inside the stack is now again fetched and is placed inside the instruction pointer (IP).
- The instruction queue will now again be filled with the subsequent instructions of the mainline program.

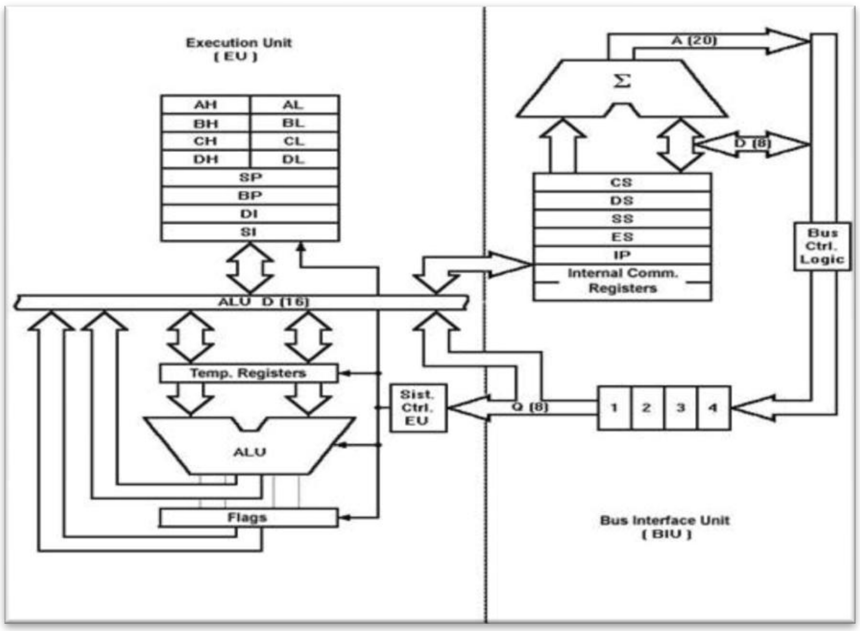
Example:

```
        ORG 100h            ;       for COM file.
        CALL p1
        ADD AX, 1

        RET                 ;       return to OS.

        p1      PROC        ;       procedure declaration.
```

		MOV AX, 1234h RET ; return to caller. p1 ENDP	
6.	**a)** **Ans.**	**Attempt any <u>TWO</u> of the following:** **Draw the functional block diagram of 8086 and describe the functions of any two segment registers.**  Memory interface BIU Σ 6 5 4 3 2 1 Instruction Stream Byte Queue ES CS SS DS IP 15 0 Control system EU 15 8 7 0 AX AH AL BX BH BL CX CH CL DX DH DL SP BP SI DI Arithmetic logic unit Operands Flags 8086 internal architecture	**16** **8M** ***Diagram 4M***

		OR  Execution Unit (EU) AH AL BH BL CH CL DH DL SP BP DI SI ALU D (16) Temp. Registers ALU Flags Sist. Ctrl. EU A (20) Σ D (8) CS DS SS ES IP Internal Comm. Registers Bus Ctrl. Logic Q (8) 1 2 3 4 Bus Interface Unit (BIU) **Functional block diagram of 8086** The 8086 has four special segment registers: cs, ds, es, and ss. These stand for Code Segment, Data Segment, Extra Segment, and Stack Segment, respectively. These registers are all 16 bits wide. They deal with selecting blocks (segments) of main memory. A segment register (e.g., cs) points at the beginning of a segment in memory. Segments of memory on the 8086 can be no larger than 65,536 bytes long. This infamous “64K segment limitation” has disturbed many a programmer. We’ll see some problems with this 64K limitation, and some solutions to those problems, later. 1. **CS-**The CS register points at the segment containing the currently executing machine instructions. Note that, despite the 64K segment limitation, 8086 programs can be longer than 64K. You simply need multiple code segments in memory. Since you can change the value of the cs register, you can switch to a new code segment when you want to execute the code located there. 2. **DS-**The data segment register, DS, generally points at global variables for the program. Again, you’re limited to 65,536 bytes of	***Description of Any two Segment register 2M each***

		data in the data segment; but you can always change the value of the ds register to access additional data in other segments. 3. **ES-**The extra segment register, ES, is exactly that – an extra segment register. 8086 programs often use this segment register to gain access to segments when it is difficult or impossible to modify the other segment registers. 4. **SS-**The SS register points at the segment containing the 8086 stack. The stack is where the 8086 stores important machine state information, subroutine return addresses, procedure parameters, and local variables. In general, you do not modify the stack segment register because too many things in the system depend upon it.	

	b)
	
	Ans.
		Write an 8086 assembly language program to compare two strings using
		(i) String instructions
		(ii) Without using string instructions.
		
		i) ALP With string Instruction
		`section .text`
		`global _start ;must be declared for using gcc`
		
		`_start: ;tell linker entry point`
		`mov esi, s1`
		`mov edi, s2`
		`mov ecx, lens2`
		`cld`
		repe**cmpsb**
		`jecxz equal ;jump when ecx is zero`
		
		;If not equal then the following code
		`mov eax, 4`
		`mov ebx, 1`
		`mov ecx, msg_neq`
		`mov edx, len_neq`
		`int 80h`
		`jmp exit`
		
		`equal:`
		`mov eax, 4`
			8M
			
			Relevant Correct program with string instruction 4M

```
   mov ebx, 1
   mov ecx, msg_eq
   mov edx, len_eq
   int 80h

exit:
   mov eax, 1
   mov ebx, 0
   int 80h

section        .data
s1 db 'Hello, world!',0                  ;our first string
lens1 equ $-s1

s2 db 'Hello, there!', 0                 ;our second string
lens2 equ $-s2

msg_eqdb 'Strings are equal!', 0xa
len_eqequ $-msg_eq

msg_neqdb 'Strings are not equal!'
len_neqequ $-msg_neq
```

ii) ALP Without using string instruction.

```
INCLUDE io.h

Cr EQU 0ah
Lf EQU 0dh

data SEGMENT
p_str1 DB Cr, Lf, 'Enter 1st string: ',0
p_str2 DB Cr, Lf, 'Enter 2nd string: ',0
p_not1 DB Cr, Lf, 'The strings are not same because of different
lengths',0
p_not2 DB Cr, Lf, 'The strings are not same because of different
characters',0
p_same DB Cr, Lf, 'The strings are the same',0
str1 DB 100 DUP (?)
```

Relevant Correct program without string instruction 4M

```
    str2 DB 100 DUP (?)
    data ENDS
code SEGMENT
                ASSUME cs:code, ds:data
    start:      mov ax, data
                mov ds, ax                      ;input str1
                output p_str1
                inputs str1, 100
                mov bx, cx                      ;input str2
                output p_str2
                inputs str2, 100                ;compare lengths
                cmp bx, cx
                jne l_not1                      ;if different length jump

         ;initialize
                lea si, str1
                lea di, str2            ;iterate to compare characters
            nxt_chk:    mov al, [si]    ;copy the two bytes in ax
                mov ah, [di]
                cmp al, ah              ;compare the two bytes
                jne l_not2              ;if not (ah==al)
                incsi                   ;increment the two bytes
                inc di
                dec cx          ;decrement the count(string length)
                jzl_same                ;if count=0 the strings are same
                jmpnxt_chk              ;else jump to check next byte

    l_not1:     output p_not1
                jmp quit
    l_not2:     output p_not2
                jmp quit
    l_same:     output p_same

    quit:mov al, 00h
                mov ah, 4ch
                int 21h
    code ENDS
    END start
```

	c) (i) Ans.	**Define recursive procedure and enlist the directives used in procedure.** **Recursive procedure :** A recursive procedure is a procedure which calls itself. Recursive procedures are used to work with complex data structure called trees. If the procedure is called with N (recursion depth) = 3. Then the n is decremented by one after each procedure is called until n = 0. Fig shows the flow diagram and pseudo-code for recursive procedure. Procedure directives are: i) PROC ii) ENDP i. **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. ii. **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 to bracket a procedure.	**4M** ***Definition 2M*** ***Enlist 2M***
	c) (ii) Ans.	**Write a procedure to find the factorial of a number.** **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	**4M** ***Correct program 4M***

		`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	