



**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.		<b>Attempt any Five of the following:</b>	<b>10M</b>
	a	<b>State the function of READY and INTR pin of 8086</b>	<b>2M</b>
	Ans	<p><b>Ready:</b> 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.</p> <p><b>INTR</b> 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.</p>	Each correct function 1M
	b	<b>What is role of XCHG instruction in assembly language program? Give example</b>	<b>2M</b>
	Ans	<p><b>Role of XCHG:</b> This instruction exchanges the contents of a register with the contents of another register or memory location.</p> <p><b>Example:</b> XCHG AX, BX ; Exchange the word in AX with word in BX.</p>	Correct role: 1M Correct example : 1M



			(any other example allowed)
	<b>c</b>	<b>List assembly language programming tools.</b>	<b>2M</b>
	<b>Ans</b>	<ol style="list-style-type: none"> <li>1. Editors</li> <li>2. Assembler</li> <li>3. Linker</li> <li>4. Debugger.</li> </ol>	Each ½ M
	<b>d</b>	<b>Define Macro.Give syntax.</b>	<b>2M</b>
	<b>Ans</b>	<p><b>Macro:</b> 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.</p> <p><b>Syntax:</b></p> <p>Macro_name      MACRO[arg1,arg2,.....argN)</p> <p>.....</p> <p>End</p>	Definition 1M Syntax 1M
	<b>e</b>	<b>Draw flowchart for multiplication of two 16 bit numbers.</b>	<b>2M</b>
	<b>Ans</b>	<pre> graph TD     Start([START]) --&gt; Input[/AX ← Num1 BX ← Num2/]     Input --&gt; Process[DX, AX ← (AX)*(BX)]     Process --&gt; Process2[DX ← MS Word of Product AX ← LS Word of Product]     Process2 --&gt; Output[/[Product] ← AX [Product+1] ← DX/]     Output --&gt; Stop([STOP])     </pre>	Correct flowchart: 2M(consider any relevant flowchart also)
	<b>f</b>	<b>Draw machine language instruction format for Register-to-Register transfer.</b>	<b>2M</b>



	<b>Ans</b>	$D_7 \quad D_6 \quad D_5 \quad D_4 \quad D_3 \quad D_2 \quad D_1 \quad D_0$ <table border="1" style="margin: auto; border-collapse: collapse;"> <tr> <td style="padding: 2px;"><i>OP CODE</i></td> <td style="padding: 2px;"><i>d</i></td> <td style="padding: 2px;"><i>w</i></td> <td style="padding: 2px;">11</td> <td style="padding: 2px;"><i>REG</i></td> <td style="padding: 2px;"><i>R/M</i></td> </tr> </table>	<i>OP CODE</i>	<i>d</i>	<i>w</i>	11	<i>REG</i>	<i>R/M</i>	Correct diagram 2M												
<i>OP CODE</i>	<i>d</i>	<i>w</i>	11	<i>REG</i>	<i>R/M</i>																
	<b>g</b>	<b>State the use of STC and CMC instruction of 8086.</b>	<b>2M</b>																		
	<b>Ans</b>	<p>STC – This instruction is used to Set Carry Flag. <math>CF \leftarrow 1</math></p> <p>CMC – This instruction is used to Complement Carry Flag.</p> <p><math>CF \leftarrow \sim CF</math></p>	Each correct use 1M																		
<b>2.</b>		<b>Attempt any Three of the following:</b>	<b>12M</b>																		
	<b>a</b>	<b>Give the difference between intersegment and intrasegment CALL</b>	<b>4M</b>																		
	<b>Ans</b>	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 10%;">Sr.no</th> <th style="width: 45%;">Intersegment Call</th> <th style="width: 45%;">Intrasegment Call</th> </tr> </thead> <tbody> <tr> <td style="text-align: center;">1.</td> <td>It is also called Far procedure call</td> <td>It is also called Near procedure call.</td> </tr> <tr> <td style="text-align: center;">2.</td> <td>A far procedure refers to a procedure which is in the different code segment from that of the call instruction.</td> <td>A near procedure refers to a procedure which is in the same code segment from that of the call instruction</td> </tr> <tr> <td style="text-align: center;">3</td> <td>This procedure call replaces the old CS:IP pairs with new CS:IP pairs</td> <td>This procedure call replaces the old IP with new IP.</td> </tr> <tr> <td style="text-align: center;">4.</td> <td>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)</td> <td>The value of old IP is pushed on to the stack.  SP=SP-2 ;Save IP on stack(address of procedure)</td> </tr> <tr> <td style="text-align: center;">5.</td> <td>More stack locations are required</td> <td>Less stack locations are required</td> </tr> </tbody> </table>	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	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																			

		<p>6. Example :- Call FAR PTR Delay</p>	<p>Example :- Call Delay</p>
<b>b</b>	<b>Draw flag register of 8086 and explain any four flags.</b>		<b>4M</b>
<b>Ans</b>	<p><b>Flag Register of 8086</b></p> <div style="text-align: center;"> <p style="text-align: center;">Status flags of intel 8086</p> </div> <p><b>Conditional /Status Flags</b></p> <p><b>C-Carry Flag</b> : It is set when carry/borrow is generated out of MSB of result. (i.e D<sub>7</sub> bit for 8-bit operation, D<sub>15</sub> bit for a 16 bit operation).</p> <p><b>P-Parity Flag</b> This flag is set to 1 if the lower byte of the result contains even number of 1's otherwise it is reset.</p> <p><b>AC-Auxiliary Carry Flag</b> This is set if a carry is generated out of the lower nibble, (i.e. From D<sub>3</sub> to D<sub>4</sub> bit)to the higher nibble</p> <p><b>Z-Zero Flag</b> This flag is set if the result is zero after performing ALU operations. Otherwise it is reset.</p> <p><b>S-Sign Flag</b> This flag is set if the MSB of the result is equal to 1 after performing ALU operation , otherwise it is reset.</p> <p><b>O-Overflow Flag</b> 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.</p> <p><b>Control Flags</b></p> <p><b>T-Trap Flag</b> If this flag is set ,the processor enters the single step execution mode.</p> <p><b>I-Interrupt Flag</b> 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.</p>		<p>Correct diagram 2M</p> <p>Any 4 flag explanation :1/2 M each</p>



		<b>D-Direction Flag</b> It selects either increment or decrement mode for DI &/or SI register during string instructions.	
	<b>c</b>	<b>Explain assembly language program development steps.</b>	<b>4M</b>
	<b>Ans</b>	<p><b>1. Defining the problem:</b> The first step in writing program is to think very carefully about the problem that the program must solve.</p> <p><b>2. Algorithm:</b> The formula or sequence of operations to be performed by the program can be specified as a step in general English is called algorithm.</p> <p><b>3. Flowchart:</b> The flowchart is a graphically representation of the program operation or task.</p> <p><b>4. Initialization checklist:</b> Initialization task is to make the checklist of entire variables, constants, all the registers, flags and programmable ports</p> <p><b>5. Choosing instructions:</b> Choose those instructions that make program smaller in size and more importantly efficient in execution.</p> <p><b>6. Converting algorithms to assembly language program:</b> Every step in the algorithm is converted into program statement using correct and efficient instructions or group of instructions.</p>	Correct steps 4M
	<b>d</b>	<b>Explain logical instructions of 8086.(Any Four)</b>	<b>4M</b>
	<b>Ans</b>	<p><b>Logical instructions.</b></p> <p><b>1) AND- Logical AND</b></p> <p style="padding-left: 40px;"><b>Syntax : AND destination, source</b></p> <p style="padding-left: 40px;"><b>Operation</b></p> <p style="padding-left: 40px;"><b>Destination ← destination AND source</b></p> <p style="padding-left: 40px;"><b>Flags Affected : CF=0,OF=0,PF,SF,ZF</b></p> <p style="padding-left: 40px;">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.</p> <p style="padding-left: 40px;"><b>Example: AND AX, BX</b></p> <ul style="list-style-type: none"> <li>• AND AL,BL</li> <li>• AL 1111 1100</li> <li>• BL 0000 0011</li> <li>• -----</li> <li>• AL←0000 0000 (AND AL,BL)</li> </ul> <p><b>2) OR – Logical OR</b></p> <p style="padding-left: 40px;"><b>Syntax :OR destination, source</b></p>	Any 4 instruction correct explanation 1M each



	<p>Operation</p> <p>Destination ← OR source</p> <p><b>Flags Affected :CF=0,OF=0,PF,SF,ZF</b></p> <p>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.</p> <p>Example :</p> <ul style="list-style-type: none"><li>• OR AL,BL</li><li>• AL 1111 1100</li><li>• BL 0000 0011</li><li>• -----</li><li>• AL←1111 1111</li></ul> <p><b>3) NOT – Logical Invert</b></p> <p><b>Syntax : NOT destination</b></p> <p>Operation: Destination← NOT destination</p> <p><b>Flags Affected :None</b></p> <p>The NOT instruction inverts each bit of the byte or words at the specified destination.</p> <p><b>Example</b></p> <p>NOT BL</p> <p><b>BL = 0000 0011</b></p> <p><b>NOT BL gives 1111 1100</b></p> <p><b>4) XOR – Logical Exclusive OR</b></p> <p><b>Syntax : XOR destination, source</b></p> <p>Operation : Destination ← Destination XOR source</p> <p><b>Flags Affected :CF=0,OF=0,PF,SF,ZF</b></p> <p>This instruction exclusive, OR's each bit in a source byte or word with the same number bit in a destination byte or word.</p>	
--	--	--



		<p><b>Example(optional)</b></p> <p><b>XOR AL,BL</b></p> <ul style="list-style-type: none"> <li>• AL 1111 1100</li> <li>• BL 0000 0011</li> <li>-----</li> <li>• AL←1111 1111 (XOR AL,BL)</li> </ul> <p><b>5)TEST</b></p> <p><b>Syntax : TEST Destination, Source</b> 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.</p> <p><b>Operation performed:</b></p> <p>Flags ← set for result of (destination AND source)</p> <p><b>Example: (Any 1)</b> TEST AL, BL ; AND byte in BL with byte in AL, no result, Update PF, SF, ZF.</p> <p>e.g <b>MOV AL, 00000101</b></p> <p><b>TEST AL, 1 ; ZF = 0.</b></p> <p><b>TEST AL, 10b ; ZF = 1</b></p>	
<b>3.</b>		<b>Attempt any Four of the following:</b>	
	<b>a</b>	<b>Draw functional block diagram of 8086 microprocessor.</b>	<b>4M</b>
	<b>Ans</b>		Block diagram 4M

	<p style="text-align: center;">8086 internal architecture</p>	
<p><b>b</b></p>	<p><b>Write an ALP to add two 16-bit numbers.</b></p>	<p><b>4M</b></p>
<p><b>Ans</b></p>	<pre> DATA SEGMENT NUMBER1 DW 6753H NUMBER2 DW 5856H SUM DW 0 DATA ENDS CODE SEGMENT ASSUME CS: CODE, DS: DATA START: MOV AX, DATA         </pre>	<p>Data segment initialization 1M, Code segment 3M</p>

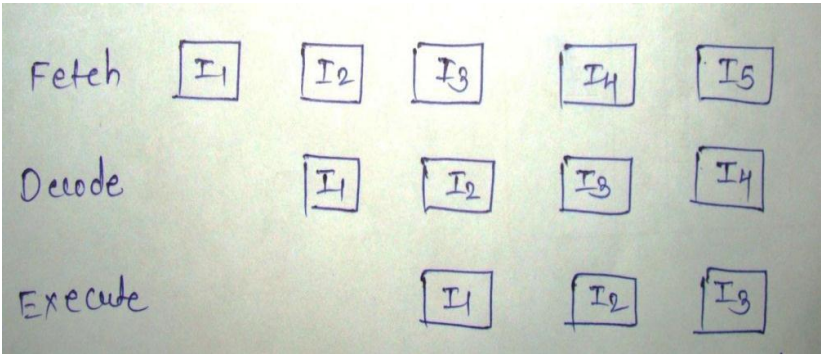




		MOV DS, AX MOV AX, NUMBER1 MOV BX, NUMBER2 ADD AX, BX MOV SUM, AX MOV AH, 4CH INT 21H CODE ENDS END START	
	<b>c</b>	<b>Write an ALP to find length of string.</b>	<b>4M</b>
	<b>Ans</b>	Data Segment STRG DB 'GOOD MORNINGS\$' LEN DB ? DATA ENDS CODE SEGMENT START: ASSUME CS: CODE, DS : DATA MOV DX, DATA 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	
	<b>d</b>	<b>Write an assembly language program to solve <math>p = x^2 + y^2</math> using Macro.(x and y are 8 bit numbers.</b>	<b>4M</b>
	<b>Ans</b>	<pre>.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</pre>	program - 4 M

		MOV p,al  MOV ah,4Ch  Int 21H  END	
<b>4.</b>		<b>Attempt any Three of the following:</b>	
	<b>a</b>	<b>What is pipelining? How it improves the processing speed.</b>	
	<b>Ans</b>	<ul style="list-style-type: none"> <li>• In 8086, pipelining is the technique of overlapping instruction fetch and execution mechanism.</li> <li>• 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.</li> <li>• While executing one instruction other instruction can be fetched. Thus it avoids the waiting time for execution unit to receive other instruction.</li> <li>• 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.</li> <li>• When the EU is ready for its next instruction, it simply reads the instruction from the queue in the BIU.</li> <li>• 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.</li> <li>• This improves overall speed of the processor</li> </ul>  <p>The diagram illustrates the instruction pipelining process. It is organized into three rows representing the stages: Fetch, Decode, and Execute. Each instruction (I1 to I5) progresses through these stages over time. In the Fetch stage, I1 to I5 are present. In the Decode stage, I1 to I4 are present. In the Execute stage, I1 to I3 are present. This shows that while one instruction is being executed, the next is being decoded, and the next is being fetched, allowing for parallel processing and faster overall execution.</p>	Explanation 3 M, Diagram 1 M
	<b>b</b>	<b>Write an ALP to count no.of 0's in 16 bit number.</b>	<b>4M</b>
	<b>Ans</b>	DATA SEGMENT N DB 1237H Z DB 0	Program 4 M



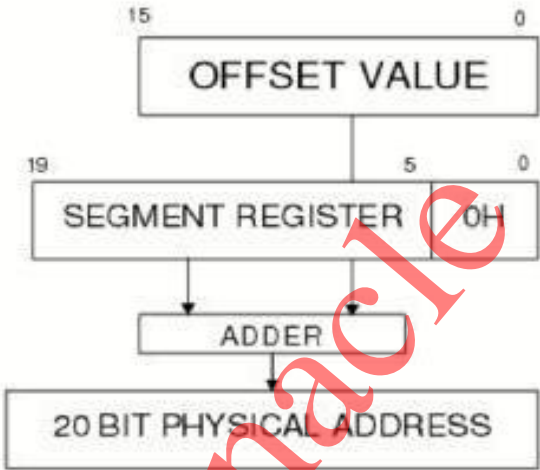
		DATA ENDS CODE SEGMENT ASSUME DS:DATA, CS:CODE START: MOV DX,DATA MOV DS,DX MOV AX, N MOV CL,08 NEXT: ROL AX,01 JC ONE INC Z ONE: LOOP NEXT HLT CODE ENDS END START	
	<b>c</b>	<b>Write an ALP to find largest number in array of elements 10H, 24H, 02H, 05H, 17H.</b>	<b>4M</b>
	<b>Ans</b>	DATA SEGMENT ARRAY 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	<b>Program - 4 M</b>
	<b>d</b>	<b>Write an ALP for addition of series of 8-bit number using procedure.</b>	<b>4M</b>
	<b>Ans</b>	DATA SEGMENT NUM1 DB 10H,20H,30H,40H,50H RESULT DB 0H CARRY DB 0H	<b>Program - 4 M</b>



	<p><b>DATA ENDS</b> <b>CODE SEGMENT</b> 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</p> <p><b>SUM PROC;</b> 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</p>	
<b>e</b>	<b>Describe re-entrant and recursive procedure with schematic diagram.</b>	<b>4M</b>
<b>Ans</b>	<p>In some situation it may happen that Procedure 1 is called from main program Procedure 2 is called from procedure 1 and procedure 1 is again called from procedure 2. In this situation program execution flow reenters in the procedure 1. These types of procedures are called re-entrant procedures. The RET instruction at the end of procedure 1 returns to procedure 2. The RET instruction at the end of procedure 2 will return the execution to procedure 1. Procedure 1 will again be executed from where it had stopped at the time of calling procedure 2 and the RET instruction at the end of this will return the program execution to main program.</p> <p>The flow of program execution for re-entrant procedure is as shown in FIG.</p>	<b>Re-entrant 2 M, recursive 2 M</b>

	<p><b>Sketch :</b></p> <p><b>Recursive Procedure</b></p> <p>A recursive procedure is a procedure which calls itself. Recursive procedures are used to work with complex data structures called trees. If the procedure 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.</p> <p><b>Fig. Flow diagram and pseudo-code for recursive procedure</b></p>	
5.	<b>Attempt any Two of the following:</b>	<b>12 M</b>
a	<b>Define logical and effective address. Describe physical address generation process in 8086. If DS=345AH and SI=13DCH. Calculate physical address.</b>	<b>6M</b>
Ans	<p><b>A logical address</b> 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.</p> <p><b>Effective Address or Offset Address:</b> 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.</p>	<p>Define each Term :1M.</p> <p>Physical Address Generation. Description : 2 M &amp; Calculation 2 M</p>



	<p><b>Generation of 20 bit physical address in 8086:-</b></p> <ol style="list-style-type: none"> <li>1. Segment registers carry 16 bit data, which is also known as base address.</li> <li>2. BIU appends four 0 bits to LSB of the base address. This address becomes 20-bit address.</li> <li>3. Any base/pointer or index register carries 16 bit offset.</li> <li>4. Offset address is added into 20-bit base address which finally forms 20 bit physical address of memory location</li> </ol>  <p>DS=345AH and SI=13DCH</p> $\begin{aligned} \text{Physical address} &= \text{DS} * 10\text{H} + \text{SI} \\ &= 345\text{AH} * 10\text{H} + 13\text{DCH} \\ &= 345\text{A0} + 13\text{DC} \\ &= 3597\text{CH} \end{aligned}$	
b	<p><b>Explain the use of assembler directives. 1) DW 2) EQU 3) ASSUME 4) OFFSET 5) SEGMENT 6) EVEN</b></p>	2M
Ans	<p><b>DW (DEFINE WORD)</b> 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.</p> <p><b>EQU (EQUATE)</b> 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.</p>	Each Directive Use : 1M each



		<p><b>Example</b> Data SEGMENT Num1 EQU 50H Num2 EQU 66H Data ENDS Numeric value 50H and 66H are assigned to Num1 and Num2.</p> <p><b>ASSUME</b> 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.</p> <p><b>OFFSET</b> 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.</p> <p><b>Example</b> 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.</p> <p><b>SEGMENT</b> 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</p> <p><b>EVEN (ALIGN ON EVEN MEMORY ADDRESS)</b> 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.</p>	
	<b>c</b>	<b>Describe any four string instructions of 8086 assembly language.</b>	<b>2M</b>
	<b>Ans</b>	<p><b>1] REP:</b> 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.</p>	each correct instruction 1½ M each





	<p><b>Two more prefix.</b></p> <p>REPE/REPZ: Repeat if Equal /Repeat if Zero.</p> <p>It will cause string instructions to be repeated as long as the compared bytes or words Are equal and CX≠0.</p> <p>REPNE/REPZ: Repeat if not equal/Repeat if not zero.</p> <p>It repeats the strings instructions as long as compared bytes or words are not equal</p> <p>And CX≠0.</p> <p><b>Example: REP MOVSB</b></p> <p><b>2] MOVS/ MOVSB/ MOVSW - Move String byte or word.</b></p> <p>Syntax:</p> <p>MOVS destination, source</p> <p>MOVSB destination, source</p> <p>MOVSW destination, source</p> <p>Operation: ES:[DI]&lt;----- DS:[SI]</p> <p>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.</p> <p><b>Example</b></p> <p>LEA SI, Source</p> <p>LEA DI, destination</p> <p>CLD</p> <p>MOV CX, 04H</p> <p>REP MOVSB</p> <p><b>3] CMPS /CMPSB/CMPSW: Compare string byte or Words.</b></p> <p>Syntax:</p> <p>CMPS destination, source</p>	
--	--	--



	<p>CMPSB destination, source</p> <p>CMPSW destination, source</p> <p>Operation: Flags affected &lt; ----- DS:[SI]- ES:[DI]</p> <p>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.</p> <p><b>Example</b></p> <p>LEA SI, Source</p> <p>LEA DI, destination</p> <p>CLD</p> <p>MOV CX, 100</p> <p>REPE CMPSB</p> <p><b>4] SCAS/SCASB/SCASW: Scan a string byte or word.</b></p> <p>Syntax:</p> <p>SCAS/SCASB/SCASW</p> <p>Operation: Flags affected &lt; ----- AL/AX-ES: [DI]</p> <p>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.</p> <p>When the match is found in the string execution stops and ZF=1 otherwise ZF=0.</p> <p><b>Example</b></p> <p>LEA DI, destination</p> <p>MOV AI, 0DH</p> <p>MOV CX, 80H</p> <p>CLD</p> <p>REPNE SCASB</p>	
--	--	--



		<p><b>5] LODS/LODSB/LODSW:</b></p> <p>Load String byte into AL or Load String word into AX.</p> <p>Syntax:</p> <p>LODS/LODSB/LODSW</p> <p>Operation: AL/AX &lt; ----- DS: [SI]</p> <p>IT copies a byte or word from string pointed by SI in data segment into AL or AX.CX</p> <p>may contain the counter and DF may be either 0 or 1</p> <p><b>Example</b></p> <p>LEA SI, destination</p> <p>CLD</p> <p>LODSB</p> <p><b>6] STOS/STOSB/STOSW (Store Byte or Word in AL/AX)</b></p> <p>Syntax STOS/STOSB/STOSW</p> <p>Operation: ES:[DI] &lt; ----- AL/AX</p> <p>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</p>	
<b>6.</b>		<b>Attempt any Two of the following:</b>	<b>12M</b>
	<b>a</b>	<b>Describe any 6 addressing modes of 8086 with one example each.</b>	<b>6M</b>
	<b>Ans</b>	<p><b>1. Immediate addressing mode:</b></p> <p>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.</p> <p><b>Example:</b></p> <p>MOV AX,67D3H</p> <p><b>2. Register addressing mode</b></p> <p>An instruction in which an operand (data) is specified in general purpose registers, then the addressing mode is known as register addressing mode.</p>	Any 6 mode with example 1 M each



	<p><b>Example:</b></p> <p>MOV AX,CX</p> <p><b>3. Direct addressing mode</b></p> <p>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.</p> <p><b>Example:</b></p> <p>MOV CL,[2000H]</p> <p><b>4. Register Indirect addressing mode</b></p> <p>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.</p> <p><b>Example:</b></p> <p>MOV AX, [BX]</p> <p><b>5. Indexed addressing mode</b></p> <p>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.</p> <p>DS is the default segment for SI and DI.</p> <p>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.</p> <p><b>Example:</b></p> <p>MOV AX,[SI]</p> <p><b>6. Based Indexed addressing mode:</b></p> <p>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</p> <p><b>Example:</b></p> <p>MOV AX, [BX][SI]</p> <p><b>7. Register relative addressing mode:</b> An instruction in which the address of the operand is obtained by adding the displacement (8-bit or 16 bit) with</p>	
--	--	--



	<p>the contents of base registers or index registers (BX, BP, SI, DI). The default segment register is DS or ES.</p> <p><b>Example:</b></p> <p>MOV AX, 50H[BX]</p> <p><b>8. Relative Based Indexed addressing mode</b></p> <p>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.</p> <p><b>Example:</b></p> <p>MOV AX, 50H [BX][SI]</p>	
	<p><b>b</b> Select assembly language for each of the following</p> <ul style="list-style-type: none"><li>i) rotate register BL right 4 times</li><li>ii) multiply AL by 04H</li><li>iii) Signed division of AX by BL</li><li>iv) Move 2000h in BX register</li><li>v) increment the counter of AX by 1</li><li>vi) compare AX with BX</li></ul>	<b>6M</b>
<b>Ans</b>	<p>i) MOV CL, 04H RCL AX, CL1</p> <p>Or</p> <p>MOV CL, 04H ROL AX, CL</p> <p>Or</p> <p>MOV CL, 04H RCR AX, CL1</p>	Each correct instruction 1M



	<p>Or</p> <p>MOV CL, 04H</p> <p>ROR AX, CL</p> <p>ii) MOV BL,04h</p> <p>MUL BL</p> <p>iii) IDIV BL</p> <p>iv) MOV BX,2000h</p> <p>v) INC AX</p> <p>vi) CMP AX,BX</p>	
	<p><b>c</b> Write an ALP to reverse a string. Also draw flowchart for same.</p>	
<p><b>Ans</b></p>	<p><b>Program:</b></p> <p>DATA SEGMENT</p> <p>STRB DB 'GOOD MORNINGS'</p> <p>REV DB 0FH DUP(?)</p> <p>DATA ENDS</p> <p>CODE SEGMENT</p> <p>START:ASSUME CS:CODE,DS:DATA</p> <p>MOV DX,DATA</p> <p>MOV DS,DX</p> <p>LEA SI,STRB</p> <p>MOV CL,0FH</p> <p>LEA DI,REV</p> <p>ADD DI,0FH</p> <p>UP:MOV AL,[SI]</p>	<p>Program 4 M flowchart 2 M</p>



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

Flowchart:

