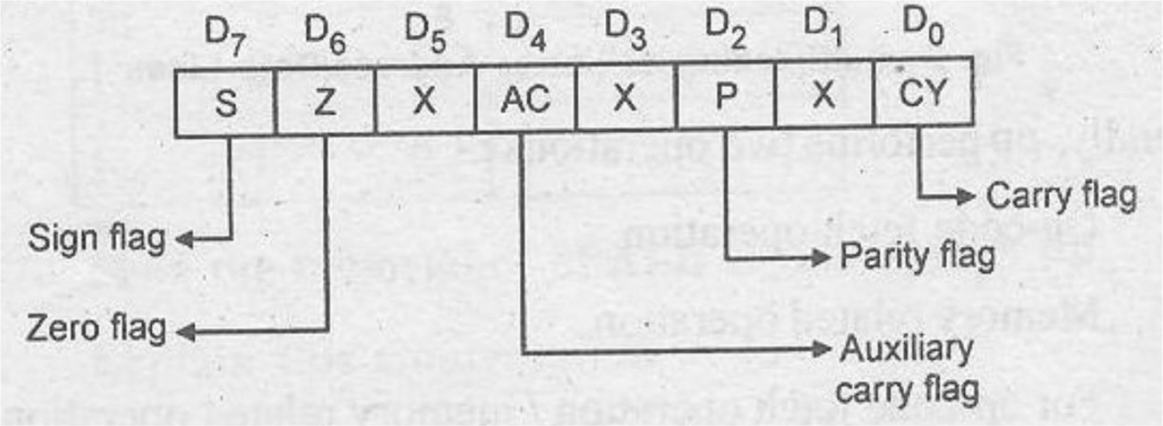
1

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in themodel answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may tryto assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given moreImportance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in thefigure. The figures drawn by candidate and model answer may vary. The examiner may give credit for anyequivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constantvalues 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.	Answers	Marking Scheme
1	**(A)**	**Attempt any SIX of the following:**	**12- Total Marks**
	(a)	**List any four salient features of 8085 microprocessor.**	**2M**
	Ans:	**(Any four)** **Features of 8085:** 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. Provides 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	**½ Mark** **each**

		9. Requires 2 phase, 50% duty cycle TTL clock 10. Provide 2 serial I/O lines, so peripheral can be interfaced with 8085 µp	
	(b)	**State the functions of following pins of 8086 microprocessor:** **(i) ALE** **(ii) M/$\overline{IO}$**	**2M**
	Ans:	**(i) ALE :-**This active high, output signal is used to indicate availability of valid address on address/data lines and is connected to latch enable input of latches (8282 or 74LS373) **(ii) M/$\overline{IO}$:-** This signal is used to differentiate between I/O & memory operations. When it is high, it indicates memory operation and when low, it indicates I/O operation.	**1 Mark for each**
	(c)	**State two examples of each, immediate and based indexed addressing modes.**	**2M**
	Ans:	**1.Immediate addressing mode :** MOV AX,67D3H MOV CL,34 H MOV BX,56D3H MOV BL,76 H **2. Based Indexed addressing mode :** MOV AX, [BX][SI] ADD AL,[BX][DI] SUB BL,[BX][SI] MOV BX,[BX][DI]	**(Any two example of each ½ M each example)**

	(d)	**Define the following terms:** **(i) Algorithm** **(ii) Flow chart**	**2M**
	Ans:	**(i)Algorithm:** The formula or sequence of operations to be performed by the program, specified as steps in general English, is called algorithm. **(ii)Flowchart:** The flowchart is a graphically representation of the program operation or task.	**1 M Each Definition**
	(e)	**Draw the flag register format of 8085 microprocessor.**	**2M**
	Ans:	 D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0 S Z X AC X P X CY Carry flag Sign flag Parity flag Zero flag Auxiliary carry flag **Format of flag register of 8085 µp**	**Format 2 Marks**
	(f)	**Describe the functions of General purpose registers of 8086 microprocessor.**	**2M**
	Ans:	**(i) General Purpose Registers of 8086** 1. AX (Accumulator) – Used to store the result for arithmetic / logical operations All I/O data transfer using IN & OUT instructions use “A” register(AH / AL or AX). 2. BX – Base – used to hold the offset address or data in indirect addressing mode. 3. CX – acts as a counter for repeating or looping instructions.	**(Any 4 General Purpose Register : 1/2 M**

4. DX –Used with AX to hold 32 bit values during multiplication and division.

Used to hold address of I/O port in indirect addressing mode.

5. BP – Base Pointer BP can hold offset address of any location in the stack segment. It is used to access random locations of stack.

6. SP –Stack Pointer – Contains the offset of the top of the stack.

SP is used with SS register to calculate 20-bit physical address.

Used during instructions like PUSH,POP,CALL,RET etc.

7. SI – Source Index – Used in string movement instructions. Holds offset address of source data in Data segment during string operations. Used to hold offset address of data segment.

8.DI – Destination Index – acts as the destination for string movement instructions

Used to hold offset address of Extra segment.

each)

(g) **Write any two differences between NEAR and FAR procedure.** **2M**

Ans:

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	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.
3	It is also called intra-segment procedure.	It is also called inter-segment procedure.
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)

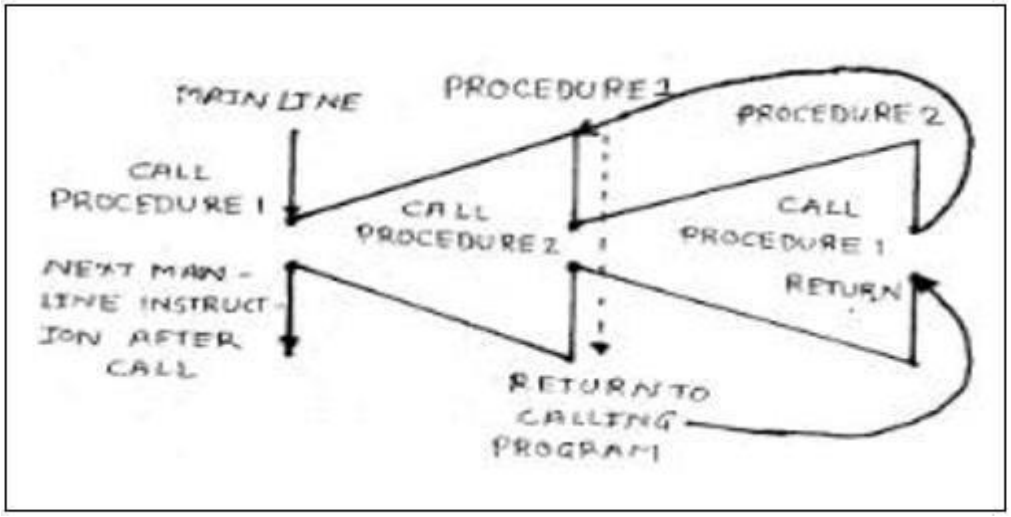
(Any 2 points , 1M each)

5	Less stack locations are required	More stack locations are required
6	Example :- Call Delay	Example :- Call FAR PTR Delay

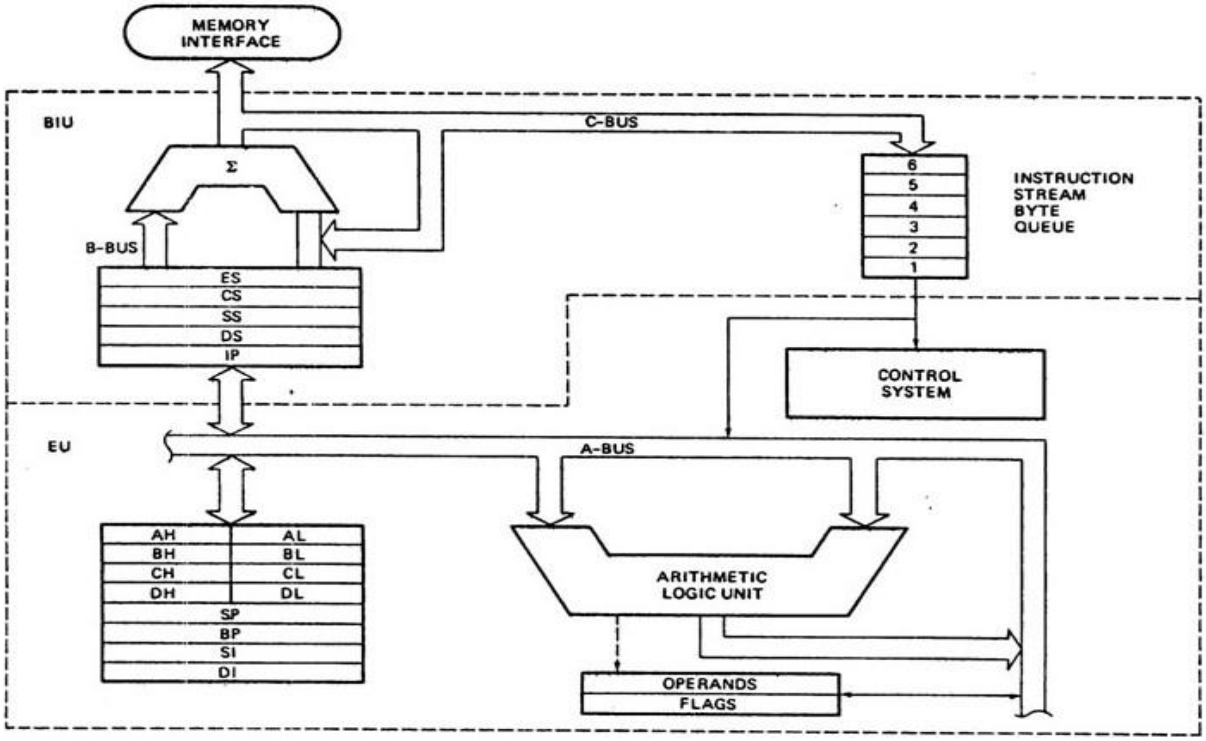
(h)	**Write assembly language instructions of 8086 microprocessor for-** **(i) Rotate the contents of BX register by 4** **(ii) Transfer 1234H to DS register**	
Ans:	**(i) Rotate the contents of BX register by 4** **(Left Rotation)** MOV CL,04H ROL BX, CL (OR) **(Right Rotation)** MOV CL,04H ROR BX, CL **(ii) Transfer 1234H to DS register** MOV DS,1234H	**(1M Each)**
(B)	**Attempt any TWO of the following :**	**08- Total Marks**
(a)	**State the functions of following program development tools:** **(i) Editor** **(ii) Assembler**	**4M**
Ans:	**(i)Editor :-** 1. An Editor is a program which helps to construct assembly language program in right format so that the assembler will translate it correctly to machine language. 2. So, we can type our program using editor. 3. This form of program is called as source program.	**(2M for Each)**

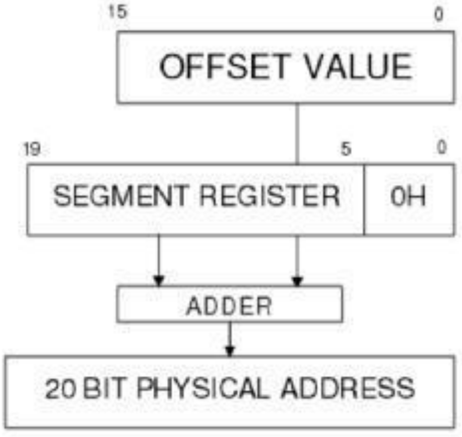
		4. The **DOS** based editor such as EDIT, WordStar and Norton Editor etc can be used to type the program. **(ii) Assembler:-** 1. Assembler is a program that translates assembly language program to the correct binary code. 2. It also generates the file called as object file with extension .obj. 3. It also displays syntax errors in the program, if any. 4. It can be also be used to produce list(.lst) and .crf files.	
	(b)	**Explain the following assembler directives:** **(i) DW** **(ii) EQU** **(iii) SEGMENT** **(iv) END**	**4M**
	Ans:	**(i)DW (Define Word)** 1. This is used to define a word (16-bit) type variable. 2. The range of values : 0 – 65535 for unsigned numbers -32768 to 32767 for signed numbers 3. This can be used to define a single word or multiple words **Syntax:** Name_Of_Variable DW Initialisation_Value(,s) **Example :** NUM DW '78' **(ii) EQU :Equate to** The EQU directive is used to declare the micro symbols to which some constant value is assigned. Microassembler will replace every occurrence of the symbol in a program by its value. **Syntax:**Symbol_name EQU expression **Example :** NUM EQU 50 **(iii)SEGMENT:** Used to indicate the beginning of logical segment. Preceding the	**(1M for Each)**

		SEGMENT directive is the name you want to give the segment **Syntax:** Segment_Name SEGMENT [Word/Public] **Example :** CODE SEGMENT WORD **(iv)END: End of the program** The directive END is used to inform assembler the end of the program. END directive is placed after the last statement of a program to tell the assembler that this is the end of the program module. The assembler will ignore any statement after an END directive. **Syntax:** END[Start_Address] The optional start_address indicates the location in the code segment where execution is to begin. The system loader uses this address to initialize CS register	
	(c)	**Explain re-entrant procedures with suitable example.**	**4M**
	Ans:	**Any other example diagram can also be considered.** In some situation it may happen that Procedure 1 is called from main program Procedure 2 is called from procedure1 and procedure1 is again called from procedure2. In this situation program execution flow reenters in the procedure 1. This type of procedures is called re-entrant procedures. A procedure is said to be re-entrant, if it can be interrupted, used and re-entered Without losing or writing over anything.	**Diagram 2M,Explanation 2M**

Q. No.	Sub Q. N.	Answers	Marking Scheme
2		**Attempt any FOUR of the following:**	**16- Total Marks**
	(a)	**State the function of following pins of 8085 microprocessor:** **(i) READY** **(ii) HOLD** **(iii) SID** (iv) $\overline{RD}$	**4M**
	Ans:	**(i) READY:** This input is used to insert wait state into the timing cycle of the 8086. If the ready pin is at logic 1, it has no effect on the operation of the microprocessor. If it is logic 0, the 8086 enters the waits state and remains the idle. This pin is used to interface the operating peripherals with the 8086. **(ii) HOLD:** Hold is an active high input signal used by the other master controller to request microprocessor for gaining the control of address, data and control buses. When microprocessor receives HOLD request signal, then microprocessor completes current	**(1M**

		machine cycle i.e. operation and release bus control for other master in the system. **(iii) SID:** SID is an active high input serial port pin. It is used to accept one bit data under the software control. When RIM instruction is executed, the SID pin data is loaded at D_7 position of accumulator. **(iv)** $\overline{RD}$: This is an active low output control signal used to read data from memory or I/O Device generated by the microprocessor.	
	(b)	**Draw a neat labeled architecture of 8086 microprocessor.**	**4M**
	Ans:	 MEMORY INTERFACE BIU C-BUS Σ INSTRUCTION STREAM BYTE QUEUE 6 5 4 3 2 1 B-BUS ES CS SS DS IP CONTROL SYSTEM EU A-BUS AH AL BH BL CH CL DH DL SP BP SI DI ARITHMETIC LOGIC UNIT OPERANDS FLAGS	**(Correct Diagram -4M)**
	(c)	**Describe Physical Address generation in 8086. If CS = 2135H and IP = 3478H. Calculate Physical Address.**	**4M**

	Ans:	**Formation of a physical address:** - Segment registers carry 16 bit data, which is also known as base address. BIU attaches 0 as LSB of the base address. So now this address becomes 20-bit address. Any base/pointer or index register carry 16 bit offset. Offset address is added into 20-bit base address which finally forms 20 bit physical address of memory location.  15 0 OFFSET VALUE 19 5 0 SEGMENT REGISTER 0H ADDER 20 BIT PHYSICAL ADDRESS **Physical address formation** CS= 2135H, IP=3478H. CS : 21350H ………0 added by BIU(or Hardwired 0) + IP : 3478H ---------------------------- 247C8H This is the Physical Address	**Description: 2M** **Calculation: 2M**
	(d)	**Explain the function of stack pointer and program counter of 8085 microprocessor.**	**4M**
	Ans:	**Stack Pointer:** It contains the offset of the top of the stack. SP is used with SS register to calculate 20-bit physical address of the stacktop. It is used during instructions like PUSH, POP, CALL, RET etc. A stack (also called a pushdown stack) operates in a last-in/first-out	**(2M for each correct**

		sense. When a new data item is entered or "pushed" onto the top of a stack, the stack pointer increments to the next physical memory address, and the new item is copied to that address. When a data item is "pulled" or "popped" from the top of a stack, the item is copied from the address of the stack pointer, and the stack pointer decrements to the next available item at the top of the stack. **Program Counter**: A program counter is a register in a computer processor that contains the address (location) of the instruction being executed at the current time. As each instruction gets fetched, the program counter increases its stored value by 1. After each instruction is fetched, the program counter points to the next instruction in the sequence. When the computer restarts or is reset, the program counter normally reverts to 0.	**Explanation)**
	(e)	**Explain any four string instructions with suitable example.**	**4M**
	Ans:	**(Any four)** **1] REP:** REP is a prefix which is written before one of the string instructions. It will cause during length counter CX to be decremented and the string instruction to be repeated until CX becomes 0. Two more prefix. **REPE/REPZ**: Repeat if Equal /Repeat if Zero. It will cause string instructions to be repeated as long as the compared bytes or words Are equal and CX≠0. **REPNE/REPNZ:** Repeat if not equal/Repeat if not zero. It repeats the strings instructions as long as compared bytes or words are not equal And CX≠0. Example: REP MOVSB **2] MOVS/ MOVSB/ MOVSW - Move String byte or word.**	**(Any four)** **1 M for Each**

Syntax:

MOVS destination, source

MOVSB destination, source

MOVSW destination, source

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

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

Example

LEA SI, Source

LEA DI, destination

CLD

MOV CX, 04H

REP MOVSB

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

Syntax:

CMPS destination, source

CMPSB destination, source

CMPSW destination, source

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

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

Example

LEA SI, Source

LEA DI, destination

CLD

MOV CX, 100

REPE CMPSB

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

Syntax:

SCAS/SCASB/SCASW

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

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

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

Example

LEA DI, destination

MOV Al, 0DH

MOV CX, 80H

CLD

REPNE SCASB

5] LODS/LODSB/LODSW: Load String byte into AL or Load String word into AX.

Syntax:

LODS/LODSB/LODSW

Operation: AL/AX < ----- DS: [SI]

IT copies a byte or word from string pointed by SI in data segment into AL or AX.CX may contain the counter and DF may be either 0 or 1

Example

LEA SI, destination

CLD

LODSB

6] STOS/STOSB/STOSW (Store Byte or Word in AL/AX)

Syntax STOS**/**STOSB/STOSW

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

It copies a byte or word from AL or AX to a memory location pointed by DI in extra segment CX may contain the counter and DF may either set or reset.

(f) **Compare 8085 and 8086 microprocessor with respect to** **4M**

- **(i) Number of data lines**
- **(ii) Number of address lines**
- **(iii) Registers**
- **(iv) Pipelining**

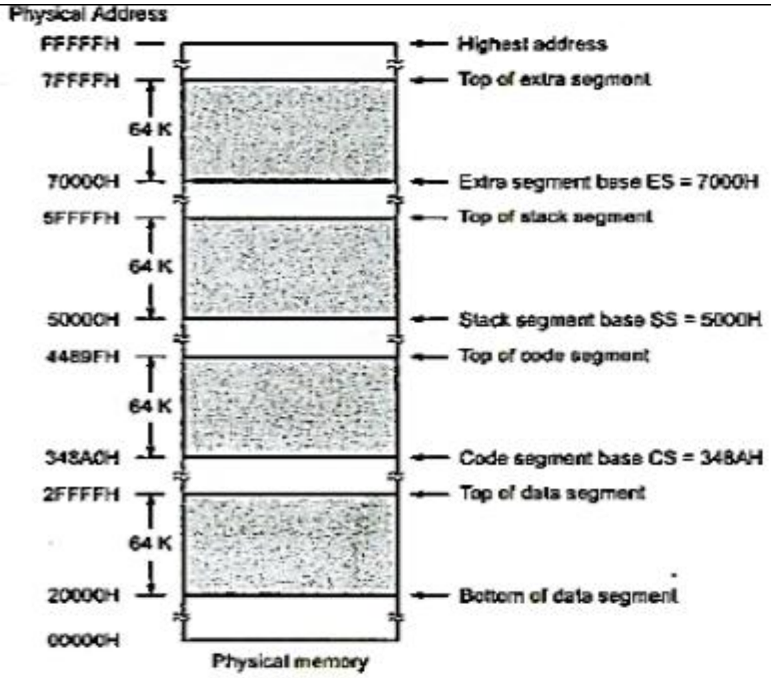
Ans: **(1M for each)**

Parameter	**8085**	**8086**
Number of data lines	8 bits	16 bits
Number of address lines	16 bits	20 bits
Registers	PC & SP , General purpose Registers, Flag register	Segment registers, Pointers and Index registers General purpose Registers, Flag/PSW register
Pipelining	Not Possible	Possible

Q. No.	Sub Q. N.	Answers	Marking Scheme

3		**Attempt any FOUR of the following :**	**16- Total Marks**
	(a)	**Explain DAA instruction with suitable example.**	**4M**
	Ans:	**DAA – (Decimal Adjust AL after BCD Addition)** **Sy**ntax- DAA Explanation: This instruction is used to make sure the result of adding two packed BCD numbers is adjusted to be a correct BCD number. The result of the addition must be in AL for DAA instruction to work correctly. If the lower nibble in AL after addition is > 9 or Auxiliary Carry Flag is set, then add 6 to lower nibble of AL. If the upper nibble in AL is > 9H or Carry Flag is set, and then add 6 to upper nibble of AL. **Example: - (Any Same Type of Example)** if AL=99 BCD and BL=99 BCD Then ADD AL, BL 1001 1001 = AL= 99 BCD + 1001 1001 = BL = 99 BCD -- 0011 0010 = AL =32 H and CF=1, AF=1 After the execution of DAA instruction, the result is CF = 1 0011 0010 =AL =32 H , AH =1 + 0110 0110 ------------------------- 1 001 1000 =AL =98 in BCD	**(Explain ation :2M Example : 2M)**
	(b)	**Explain the concept of memory segmentation in 8086.**	**4M**

<table>
<tr>
<td></td>
<td>Ans:</td>
<td>



Memory Segmentation of 8086

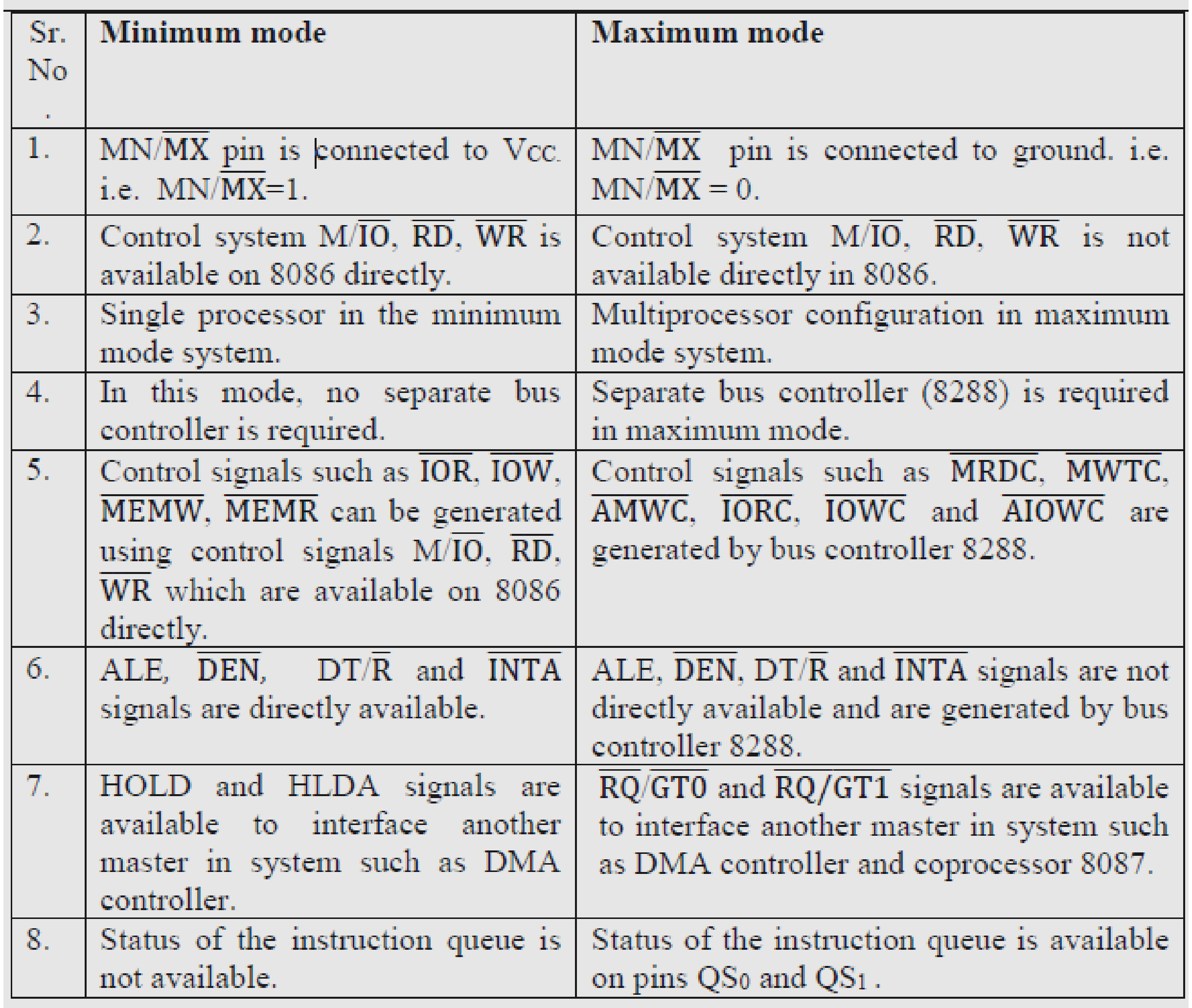
Memory Segmentation: The memory in an 8086 microprocessor is organized as asegmented memory. The physical memory is divided into 4 segments namely,-Data segment, Code Segment, Stack Segment and Extra Segment.

Data segment is used to hold data, Code segment for the executable program, Extrasegment also holds data specifically in strings and stack segment is used to storestack data.
Each segment is 64Kbytes & addressed by one segment register.

The 16 bit segment register holds the starting address of the segment The offsetaddress to this segment address is specified as a 16-bit displacement (offset) between0000 to FFFFH.
Since the memory size of 8086 is 1Mbytes, total 16 segments are possible with eachhaving 64Kbytes.

</td>
<td>2M:Diagram And 2 M: description</td>
</tr>
<tr>
<td></td>
<td>(c)</td>
<td><b>Differentiate between minimum mode and maximum mode of 8086 microprocessor.(4 points)</b></td>
<td>4M</td>
</tr>
</table>

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 by bus controller 8288.
7.	HOLD and HLDA signals are available to interface another master in system such as DMA controller.	$\overline{RQ}/\overline{GT0}$ 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 .

1 M each point

(d) **Explain four rotate instructions with their syntax, operation and example.** **4M**

Ans:

1.ROL – Rotate bits of byte or word left, MSB to LSB and to CF

Syntax: ROL destination, count

Eg:

ROL BL, 2 ; Rotate all bits in BL left by 1 bit ,copy MSB to LSB and to CF

IF BL = 11110000

After Execution 11000011, CF= 1

2. ROR – Rotate bits of byte or word right, LSB to MSB and to CF

1M: each instruction

		Syntax: ROR destination, count **Eg**: **ROR BL, 2** ; Rotate all bits in BL right by 1 bit ,copy LSB to MSB and to CF IF BL = 11110000 After Execution 00111100, CF= 0 **3.RCL** – Rotate bits of byte or word left, MSB to CF and CF to LSB. **Syntax**: RCL destination, count **Eg**: **RCL BL, 2** ; Rotate all bits in BL left by 1 bit ,copy MSB to CF and CF to LSB IF BL = 11110000, CF=0 After Execution 11000001 , CF= 1 **4**. **RCR** – Rotate bits of byte or word right, LSB to CF and CF to MSB. **Syntax**: RCR destination, count **Eg**: **RCR BL, 1** ; Rotate all bits in BL right by 1 bit ,copy LSB to CF and CF to MSB. IF BL = 11110000, CF= 0 After Execution 00111100 , CF= 0	
	(e)	**Write an assembly language program to find largest number from array of 10 numbers.**	**4M**
	Ans:	**[Note: Any other logically correct program can be considered]** DATA SEGMENT ARRAY DB 15H,45H,08H,78H,56H,02H,04H,12H,23H,09H LARGEST DB 00H DATA ENDS CODE SEGMENT	**(Correct Program : 4M)**

		START:ASSUME CS:CODE,DS:DATA MOV DX,DATA MOV DS,DX MOV CX,09H 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 ; AL=78h MOV AX,4C00H INT 21H CODE ENDS END START	
	(f)	**Describe the concept of pipelining in 8086 microprocessor.**	**4M**
	Ans:	Description: 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 where as long as 6 one byte instructions can be stored well in advance and then one by one instruction goes for decoding and executions. So, while executing first instruction in a queue, processor decodes second	**(Diagram:1M** **Explaination: 3 M)**

instruction and fetches 3rd instruction from the memory In this way, 8086 perform fetch, decode and execute operation in parallel i.e. in single clock cycle as shown in above fig (b)

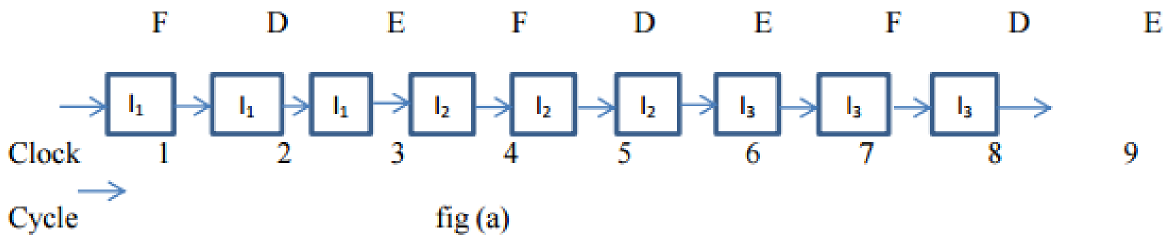


fig (a)

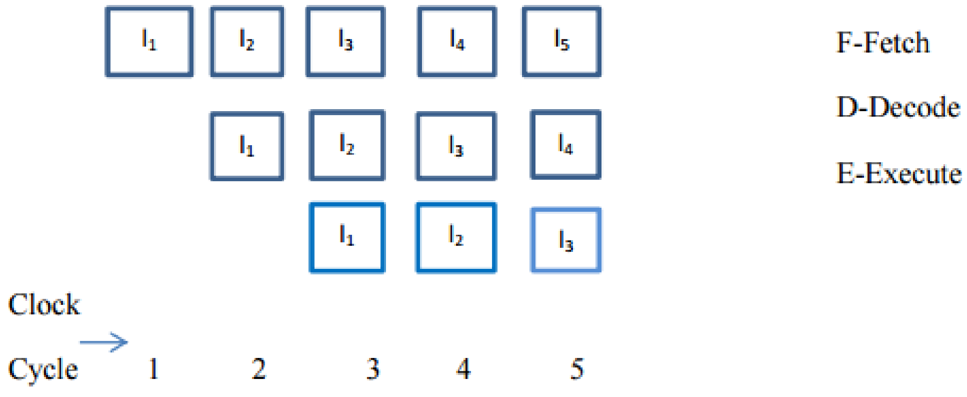


fig (b)

Q. No.	Sub Q. N.	Answers	Marking Scheme
4		**Attempt any FOUR of the following :**	**16- Total Marks**
	(a)	**Explain the following instructions with suitable examples:** **(i) ADC** **(ii) XCHG** **(iii) MUL** **(iv) AND**	**4M**
	Ans:	**(i) ADC Destination, Source**	**(½ M:**

		1) This instruction is used to add the contents of source to the destination and carry flag. 2) The result is stored in the destination. 3) The source operand can be a immediate, a register or a memory location addressed by any of the 24 addressing modes. 4) The destination can be a register or a memory location, but not an immediate data. 5) Both operands cannot be immediate data or memory location. 6) The source and the destination must be of the same data type i.e., ADD instruction adds a byte to byte or a word to word. It adds the two operands with CF. It effects AF, CF, OF, PF, SF, ZF flags. E.g.: ADC AL, 74H ADC DX, AX ADC AX, [BX] **(ii) XCHG Destination, Source** This instruction exchanges Source with Destination. .It cannot exchange two memory locations directly. The source and destination can be any of the general purpose register or memory location, but not two locations simultaneously. No segment registers can be used. E.g.: XCHG DX, AX XCHG BL, CH XCHG AL,[9800] **(iii) MUL (Unsigned multiplication)** Syntax :-- MUL source 1. This instruction multiplies an **unsigned byte** from **source** with an unsigned byte in **AL** register **or Unsigned word** from **source** with an unsigned word in **AX** register.	**explanation , ½M: example for each instruction)**

		2. The source can be a register or memory location but cannot be an immediate data. **Operation Performed :--** a. If source is byte then AX ← AL * unsigned 8 bit source b. If source is word then DX, AX ← AX * unsigned 16 bit source **Examples:--** 1. MUL BL ; Multiply AL by BL & the result in AX 2. MUL CX ; Multiply AX by CX & the result in DX,AX 3. MUL Byte PTR [SI] ; AX ← AL * [SI] **(iv)AND (Logical AND)** This instruction logically ANDs each bit of the source byte or word with the corresponding bit in the destination and stores result in the destination. **Syntax:** AND destination, source **Examples:** AND BH, CL ; AND byte in CL with Byte in BH, result in BH. AND BX,00FFH ; AND word in BX with immediate data 00ffH AND [5000H], DX; AND word in DX with a word in memory with offset 5000 in DS.	
	(b)	**Identify addressing modes of the following instructions:** **(i) ADD CX, DX** **(ii) MOV BX,1378H** **(iii) MOV CX, [BP][SI]** **(iv) MOV [4321H], CL**	**4M**
	Ans:	(i) ADD CX, DX : Register Addressing Mode (ii) MOV BX,1378H : Immediate Addressing Mode (iii) MOV CX, [BP][SI] : Based Indexed addressing mode **(iv)** MOV [4321H], CL :Direct Addressing Mode	**(1M: each instruction)**
	(c)	**Write an assembly language program to perform addition of two 16-bit numbers.**	**4M**

	Ans:	**[Note: Any other program logic can be considered]** DATA SEGMENT N1 DW 2804H N2 DW 4213H DATA ENDS CODE SEGMENT ASSUME CS: CODE, DS:DATA START: MOV AX, DATA MOV DS, AX MOV AX, N1 MOV BX, N2 ADD AL,BL MOV CL,AL MOV AL,AH ADD AL,BH MOV CH,AL MOV AH,4CH INT 21H CODE ENDS END START	**(correct program : 4M)**
	(d)	**Write an assembly language program to sort an array of 10 numbers in ascending order.**	**4M**
	Ans:	**[Note: Any other program logic can be considered]**	**(correct program**

		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 CODE ENDS END START	**: 4M)**
	(e)	**Write an assembly language program to multiply two 16-bit unsigned numbers.**	**4M**
	Ans:	**[Note: Any other program logic can be considered]**	**(correct**

		DATA SEGMENT N1 DW 2401H N2 DW 1324H C DD? DATA ENDS CODE SEGMENT ASSUME CS: CODE, DS:DATA START: MOV AX,DATA MOV DS,AX MOV AX,N1 MOV BX,N2 MUL BX MOV WORD PTR C,AX MOV WORD PTR C+2,DX INT 21H CODE ENDS END START	**program : 4M)**
	(f)	**Explain MACRO with suitable example. List four advantages of it.**	**4M**
	Ans:	**Macro** • Small sequence of the codes of the same pattern are repeated frequently at different places which perform the same operation on the different data of same data type,	**(Explaina tion with any correct**

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_name MACRO[arg1,arg2,…..argN)

….. Endm

Example:

```
MyMacro MACRO p1, p2, p3 ; macro definition with arguments
 MOV AX, p1
 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
```

example: 2 Marks,

any 4 advantages: ½ Mark each)

END START

Advantages of Macro:

1. Simplify and reduce the amount of repetitive coding.

2. Reduces errors caused by repetitive coding.

3. Makes program more readable.

4. Execution time is less as compare to procedure as no extra instructions are required.

(OR Any Same Type of Example can be considered)

Q. No.	Sub Q. N.	Answers	Marking Scheme
5.		**Attempt any FOUR of the following:**	**16- Total Marks**
	a)	**Write an assembly language program to find length of a string.**	**4M**
	Ans:		**Correct Program :4Marks** **(Any other logic also considered)** **(Assume Suitable**

```
        DATA SEGMENT
            STRB DB 'GOOD MORNING$'
            LEN DB ?
        DATA ENDS

        CODE SEGMENT
        START:ASSUME CS:CODE,DS:DATA
            MOV DX,DATA
            MOV DS,DX
            LEA SI,STRB
            MOV CL,00H
            MOV AL,'$'
       NEXT: CMP AL,[SI]
            JZ EXIT
```

```
                ADD CL,01H
                INC SI
                JMP NEXT
         EXIT: MOV LEN,CL    ;Len =0CH for Taken String.
                MOV AH,4CH
                INT 21H
                CODE ENDS
     END START
```

Data)

b) **Write an assembly language program to multiply two 8-bit unsigned numbers.** **4M**

Ans:

```
DATA SEGMENT
    NUM1 DB 05H
    NUM2 DB 02H
    RESULT DW ?
DATA ENDS
CODE SEGMENT
    ASSUME  CS:CODE,DS:DATA
  START:MOV DX,DATA
    MOV DS,DX
    MOV AL,NUM1
    MOV AH,NUM2
    MUL NUM2            ;MUL AH   ALSO ALLOWED
    MOV RESULT,AX
  MOV AX,4C00H
    INT 21H
CODE ENDS
END START
```

Correct Program :4Marks

(Any other logic also considered)

(Assume Suitable Data)

c) **Write an assembly language program to add two 8-bit BCD numbers.** **4M**

Ans:

```
.MODEL SMALL
```

Correct

```
 .DATA
 NUM1 DB 04H
 NUM2 DB 06H
 BCD_SUM DB ?
 .CODE
 MOV AX,@DATA
 MOV DS, AX
     MOV AL, NUM1
     MOV BL, NUM2
     ADD AL,BL
     DAA
     MOV BCD_SUM, AL
 MOV AH,4CH
 INT 21H
END
```

(OR)*[Note: Program with carry can also be considered]*

```
DATA SEGMENT
        OP1 EQU 92H
        OP2 EQU 52H
        RESULT DB 02 DUP(00)
DATA ENDS
ASSUME CS: CODE , DS:DATA
CODE SEGMENT
START: MOV AX,DATA
        MOV DS,AX
        MOV BL,OP1
        XOR AL,AL
        MOV AL,OP2
        ADD AL,BL
```

Program :4Marks

(Any other logic also considered)

```
                DAA
                MOV RESULT ,AL
                JNC MSBO
                INC [RESULT+1]
        MSBO: MOV AH,4CH
                INT 21H
         CODE ENDS
         END START
```

(d)	**Describe any four arithmetic instructions with example.**	**4M**

Ans:

(Any 4 instructions : ½ Mark description/operation, ½ Mark one example of each)

Arithmetic instructions

1) **ADD** – (Addition)

ADD Destination(register/memory), Source(register/memory/immediate data)

Substitute the destination byte or word with the sum of the source and destination.

Ex:

ADD AL, 74H ;Add immediate number 74H to content of AL. Result in AL

2) ADC(Add with carry) :

ADC Destination(register/memory), Source(register/memory/immediate data)

Substitute the destination byte or word with the sum of the source and destination and carry flag.

The ADC also adds the status of the carry flag to the result.

Ex.

ADC CL, BL ;Add content of BL plus carry status to content of CL

SUB(Subtraction):

SUB – Destination(register/memory), Source(register/memory/immediate data)

These instructions subtract the number in some *source* from the number in some *destination* and put the result in the destination.

SUB CX, BX ; CX – BX; Result in CX

3) **SBB (Subtract with borrow)**

SBB Destination(register/memory), Source(register/memory/immediate data)

Destination=destination-source-carry

SBB CH, AL ;Subtract content of AL and content of CF from content of CH. Result in CH

4) **MUL (Unsigned multiplication)**

MUL Source(memory/register)

When a byte is multiplied by the content of AL, the result (product) is put in AX. When a word is multiplied by the content of AX, the result is put in DX : AX registers.

Operation:

when operand is a byte , AX = AL * operand

when operand is a word , (DX:AX) = AX * operand

Example:

MOV AL, 200 ; AL = 0C8h

MOV BL, 4

MUL BL ; AX = 0320h (800)

5) **IMUL(Signed Multiplicaion)**

IMUL Source(memory/register)

This instruction multiplies a signed byte from source with a signed byte in AL or a signed word from some source with a signed word in AX. The source can be a register or a memory location.

when operand is a byte , AX = AL * operand

when operand is a word ,(DX:AX) = AX * operand

IMUL BH :Multiply signed byte in AL with signed byte in BH; result in AX.

IMUL AX :Multiply AX times AX; result in DX and AX

Example:

MOV AL, -2

MOV BL, -4

IMUL BL ; AX = 8

6) **DIV(unsigned division)**

DIV Source(memory/register)

DIV BL ;Divide word in AX by byte in BL; Quotient in AL, remainder in AH

DIV CX ;Divide double word in DX and AX by word in CX;Quotient in AX, and remainder in DX.

when operand is a byte : AL = AX / operand

AH = remainder (modulus)

when operand is a word :

AX = (DX:AX) / operand

DX = remainder (modulus)

Example:

MOV AX, 203 ; AX = 00CBh

MOV BL, 4

DIV BL ; AL = 50 (32h), AH = 3

7) IDIV (signed division)

IDIV Source(memory/register)

This instruction is used to divide a signed word by a signed byte, or to divide a signed double word by a signed word .

IDIV BL ; Signed word in AX/signed byte in BL

IDIV BP ;Signed double word in DX and AX/signed word in BP

Signed divide.

Operation:

when operand is a byte : AL = AX / operand

<table>
<tr><td></td><td></td><td>AH = remainder (modulus)
when operand is a word : AX = (DX:AX) / operand
DX = remainder (modulus)
Example:
MOV AX, -203 ; AX = 0FF35h
MOV BL, 4
IDIV BL ; AL = -50 (0CEh), AH = -3 (0FDh)

8) INC(increment)
INC Reg
This instruction increments the contents of register specified in the instruction by 1. And the contents are stored in the register itself.
Syntax :**INC Reg**
Example:
INC AX
9) DEC(decrement)
DEC Reg
This instruction decrements the contents of register specified in the instruction by 1. And the contents are stored in the register itself.
Syntax:**DECReg**
Example:
DEC AX</td><td></td></tr>
<tr><td></td><td>**(e)**</td><td>**Differentiate between procedure and Macro.(any 4 points)**</td><td>**4M**</td></tr>
<tr><td></td><td>**Ans:**</td><td><table><tr><th>Sr. No.</th><th>MACRO</th><th>PROCEDURE</th></tr></table></td><td>**(Any 4 points – 1M Each)**</td></tr>
</table>

1	Macro is a small sequence of code of the same pattern, repeated frequently at different places, which perform the same operation on different data of the same data type.	Procedure is a series of instructions is to be executed several times in a program, and called whenever required.
2	The MACRO code is inserted into the program, wherever MACRO is called, by the assembler.	Program control is transferred to the procedure, when CALL instruction is executed at run time.
3	Memory required is more, as the code is inserted at each MACRO call	Memory required is less, as the program control is transferred to procedure.
4	Stack is not required at the MACRO call.	Stack is required at Procedure CALL.
5	No overhead time required.	Extra overhead time is required for linkage between the calling program and called procedure.
6	Parameter passed as the part of statement which calls macro.	Parameters passed in registers, memory locations or stack.
7	RET is not used	RET is required at the end of the procedure
8	Macro is called using: <Macro_name> [argument list]	Procedure is called using: CALL <Procedure_name>
9	Directives used: MACRO, ENDM, LOCAL	Directives used: PROC, ENDP, FAR, NEAR
10	Example: Procedure Name PROC ------------------------------------ Procedure Statements	Example: Macro_name MACRO ------ ------ instructions

	------------------------------------ Procedure Name ENDP.	------ ENDM

(f) **Write an assembly language program for sum of series of 05 numbers using procedure.** **4M**

Ans:

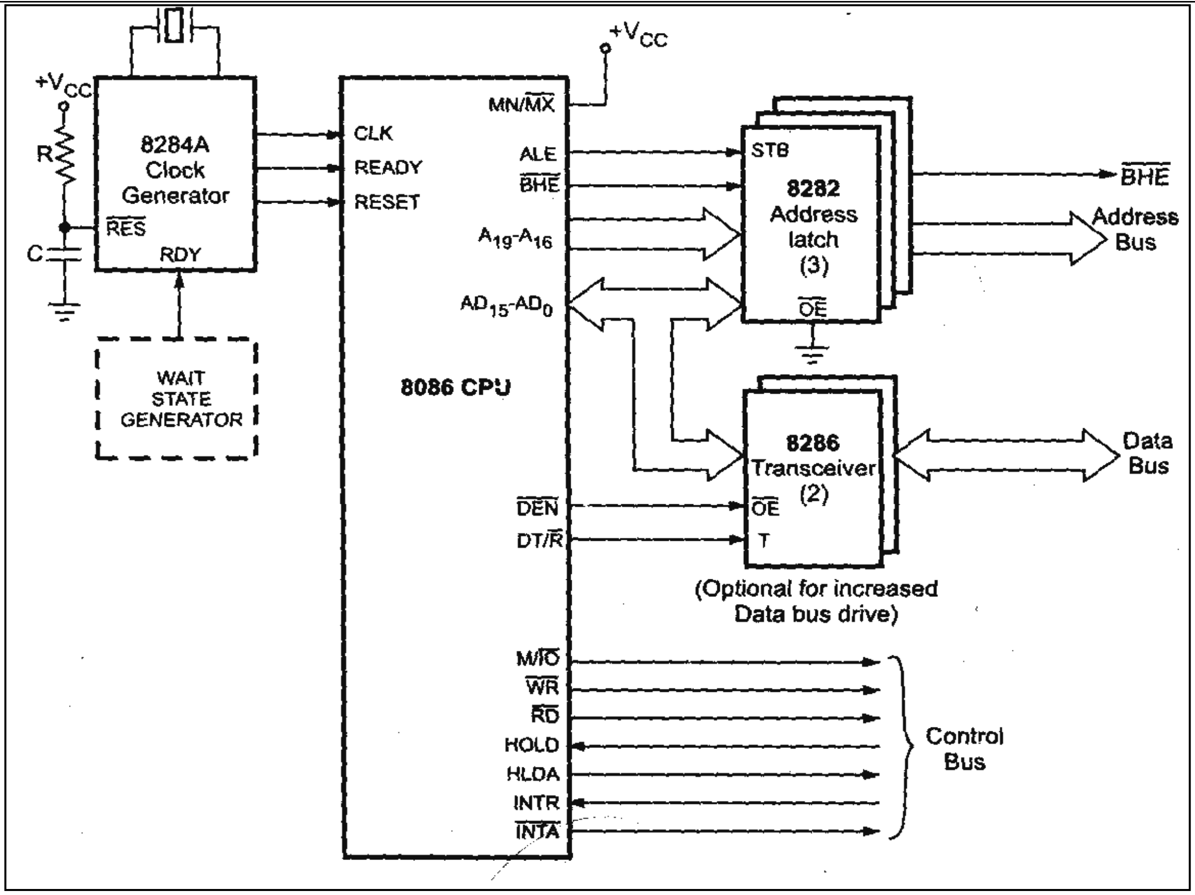
```
DATA SEGMENT
        NUM1 DB 10H,02H,30H,04H,05H
        RESULT DB 1 DUP(0)
        CARRY DB 0H
DATA ENDS
CODE SEGMENT
START:ASSUME CS:CODE,DS:DATA
        MOV DX,DATA
        MOV DS,DX
        MOV CL,05H
            CALL SERIES_ADD ;Procedure Call
       MOV AX,4C00H
       INT 21H

SERIES_ADD  PROC
        MOV SI, OFFSET NUM1
  UP:MOV AL,[SI]
        ADD RESULT,AL
        JNC NEXT
        INC CARRY
  NEXT:INC SI
       LOOP UP
       RET
 SERIES_ADD ENDP
CODE ENDS
END START
```

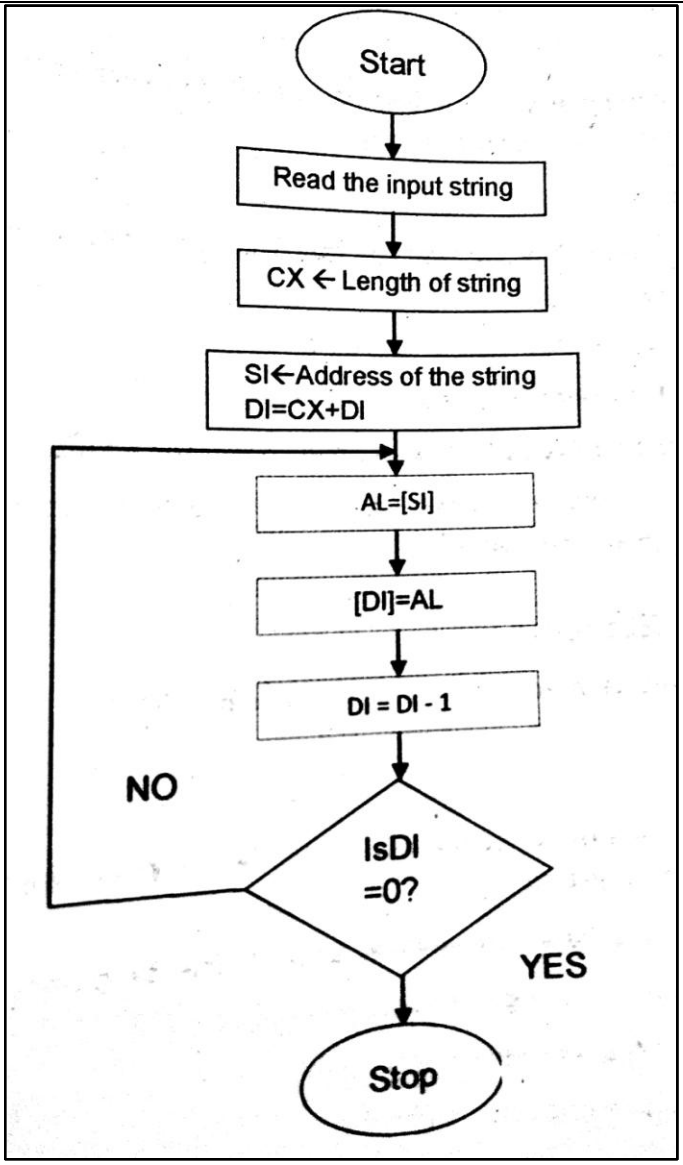
Correct Program :4Marks

(Any other logic also considered)

(Assume Suitable Data)

Q. No.	Sub Q. N.	Answers	Marking Scheme
6.		**Attempt any TWO of the following :**	**16- Total Marks**
	(a)	**Describe minimum mode operation of 8086 microprocessor with neat diagram.**	**8M**
	Ans:	 +VCC R C 8284A Clock Generator RES RDY WAIT STATE GENERATOR +VCC MN/MX CLK READY RESET 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 BHE Address Bus 8286 Transceiver (2) OE T Data Bus (Optional for increased Data bus drive) Control Bus • When MN/$\overline{MX}$ pin is in logic 1, the 8086 microprocessor operates in minimum mode system. • In this mode, the microprocessor chip itself gives out all the control signals. • This is a single processor mode. • The remaining components in the system are latches, transceivers, clock generator, memory or I/O devices.	**(Explanation: 4 Marks Diagram :4 Marks)**

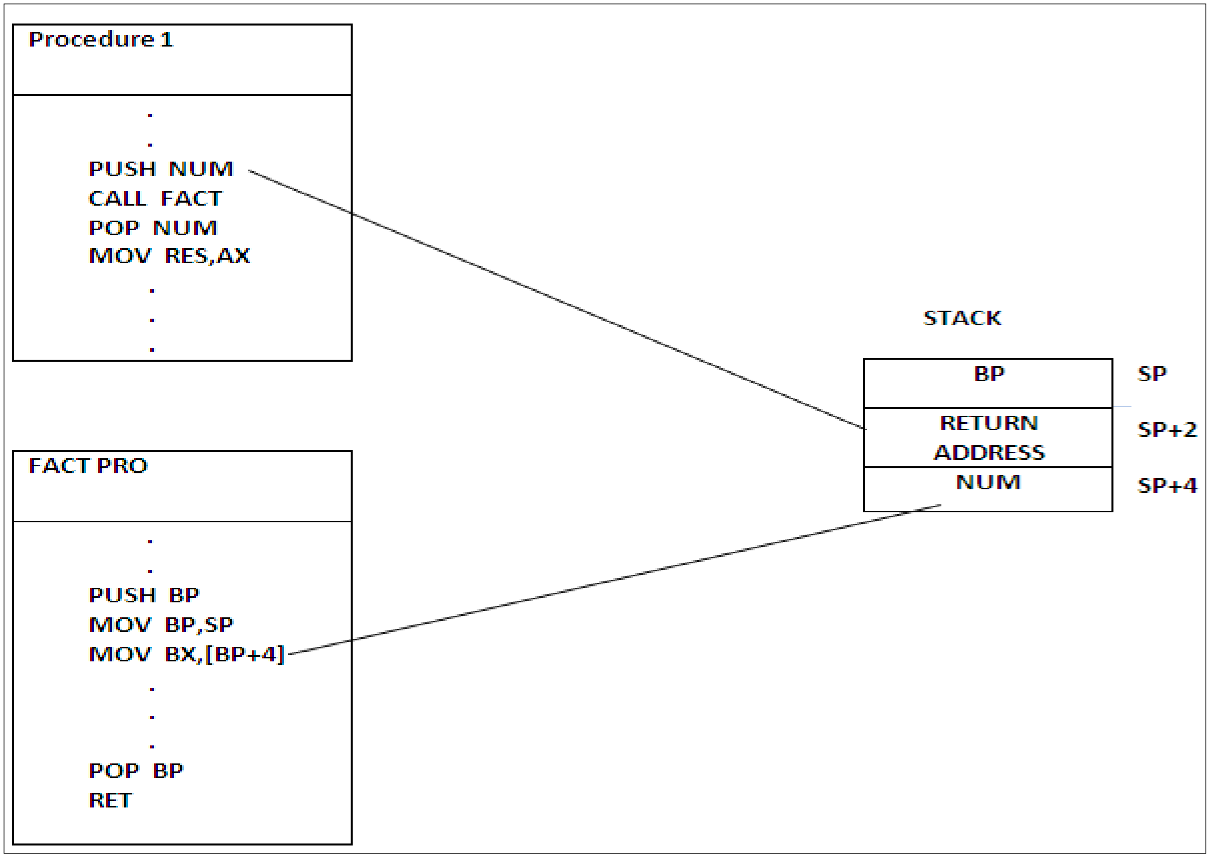
		• This system has three address latches(8282) and two octal data buffers(8286) for the complete 20-bit address and 16 bit data Separation. • The latches are used for separating the valid address from the multiplexed address/data signals and the controlled by the ALE signal generated by 8086. • Transceivers are the bi-directional buffers. They are required to separate the valid data from the time multiplexed address/data signal. This is controlled by two signals, DEN & DT/$\bar{R}$. • DT/$\bar{R}$ indicates that the direction of data, ie. from or to the microprocessor. • $\overline{DEN}$ signal indicates the valid data is available on the data bus. • This system contains memory for the monitor and users program storage. It also contains I/O devices to communicate with the processor. • The clock generator in the system is used to generate the clock and to synchronize some external signals with the system clock.	
	(b)	**Write an assembly language program to find reverse order of a given string. Also, write algorithm and draw flowchart.**	**8M**
	Ans:	**Algorithm:** 1. Initialize the Data segment and Code Segment registers. 2. SI=start of string to be reversed. 3. Assign CX=string length. 4. DI=reverse string address pointer. 5. Assign DI=string length. 6. Read the first character pointed by SI. 7. Store at the last character position pointed by DI. 8. Decrement DI to point to the next character, if DI !=0, go to step 6. 9. Stop. **Flowchart:**	**(Algorithm : 2M,** **Flowchart :2M,** **Correct Program : 4M)** **(Any other logic can be considered)**



```
DATA SEGMENT
      STRB DB 'GOOD MORNING$'
      REV DB 0FH DUP(?)
DATA ENDS
CODE SEGMENT
START:ASSUME CS:CODE,DS:DATA
```

```
        MOV DX,DATA
        MOV DS,DX
        LEA SI,STRB
        MOV CL,0FH
        LEA DI,REV
        ADD DI,0FH
  UP:MOV AL,[SI]
        MOV [DI],AL
        INC SI
        DEC DI
        LOOP UP
        MOV AH,4CH
        INT 21H
CODE ENDS
END START
```

	(c)	**Explain with suitable example how parameters are passed on the stack. Also, list out different parameter passing ways in procedure.**	**8M**
	Ans:	**PARAMETER PASSING ON THE STACK:** To pass a large number of parameters to the called procedure, the parameters can be placed on the stack for the calling procedure. Here, it is useful to use the stack base pointer i.e BP register to make a frame boundary for easy access to the parameters. The stack can also be used to pass parameters back from the called procedure to the calling procedure. The procedure during its execution pops back the appropriate parameters as and when required. In the example given below, The variable NUM is used in the called procedure using BP register, which points to the corresponding location in the stack.	**(Description : 3 Marks , Example : 3 Marks,** **List :Any 2 1Mark each)** **(Any other Example may be considered)**



Procedure 1

```
        .
        .
PUSH NUM
CALL  FACT
POP  NUM
MOV  RES,AX
        .
        .
        .
```

FACT PRO

```
        .
        .
PUSH  BP
MOV  BP,SP
MOV  BX,[BP+4]
        .
        .
        .
POP  BP
RET
```

Different Parameter Passing Ways in Procedures:

1. Passing Parameters through the Registers
2. Passing Parameters in an Argument List