SUMMER – 19 EXAMINATION

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

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

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

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

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

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

		MACRO STATEMENTS ---------- ENDM **Procedure:** A procedure is group of instructions that usually performs one task. It is a reusable section of a software program which is stored in memory once but can be used as often as necessary. A procedure can be of two types. 1) Near Procedure 2) Far Procedure Procedure can be defined as Procedure_name PROC ---- ------ Procedure_name ENDP For Example Addition PROC near ------ Addition ENDP	
	e	**Write ALP for addition of two 8bit numbers. Assume suitable data.**	**2 M**

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

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

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

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

instruction bytes per fetch.

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

Ans

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

Each Point: 1 M (any 4 Points)

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

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

Explanation for each for any two assembler

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

STC:
It sets the carry flag to 1.

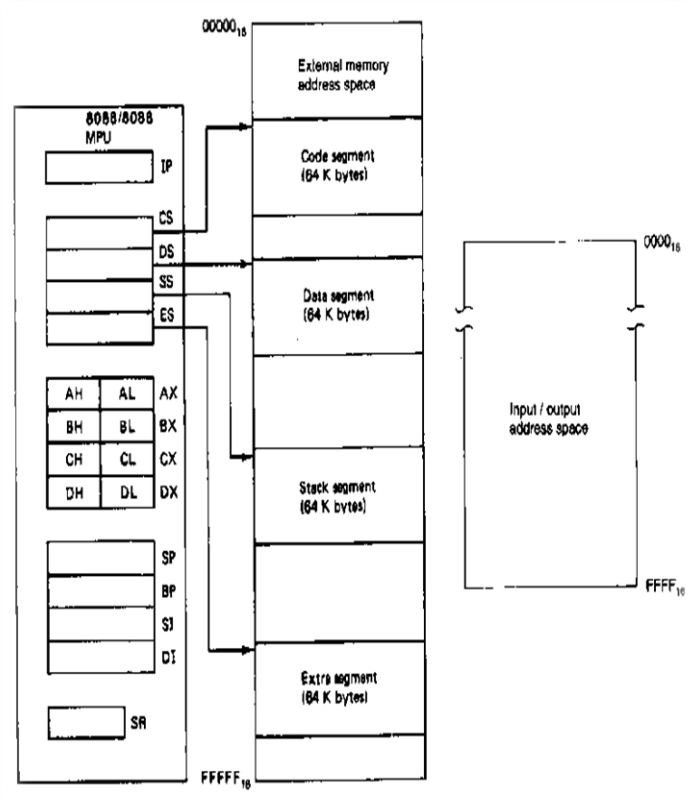
CLC:
It clears the carry flag to 0.

CMC:
It complements the carry flag.

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

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

		memory address.	
3		**Attempt any THREE :**	**12 M**
	a	**Explain memory segmentation in 8086 and list its advantages.(any two)**	**4 M**
	Ans	Memory Segmentation: • In 8086 available memory space is 1MByte. • This memory is divided into different logical segments and each segment has its own base address and size of 64 KB. • It can be addressed by one of the segment registers. • There are four segments. **SEGMENT** | **SEGMENT REGISTER** | **OFFSET REGISTER** Code Segment | CSR | **Instruction Pointer (IP)** Data Segment | DSR | **Source Index (SI)** Extra Segment | ESR | **Destination Index (DI)** Stack Segment | SSR | **Stack Pointer (SP) / Base Pointer (BP)**	Explanation 2M Any two Advantages 2M



Advantages of Segmentation:

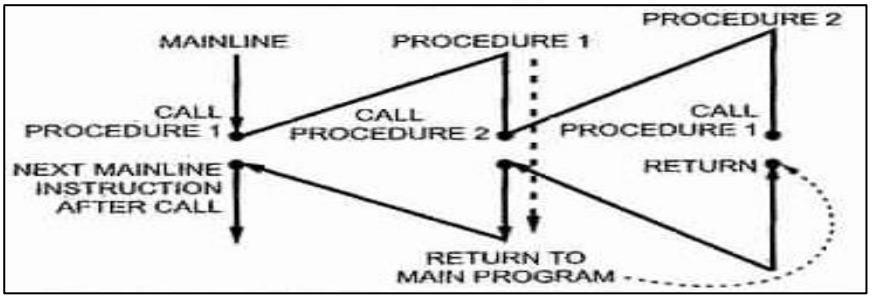
- The size of address bus of 8086 is 20 and is able to address 1 Mbytes ( ) of physical memory.
- The compete 1 Mbytes memory can be divided into 16 segments, each of 64 Kbytes size.
- It allows memory addressing capability to be 1 MB.
- It gives separate space for Data, Code, Stack and Additional Data segment as Extra segment size.
- The addresses of the segment may be assigned as 0000*H* to *F*000*H* respectively.
- The offset values are from 00000H to FFFFFH
- Segmentation is used to increase the execution speed of computer system so that processor can able to fetch and execute the data from memory easily and fast.

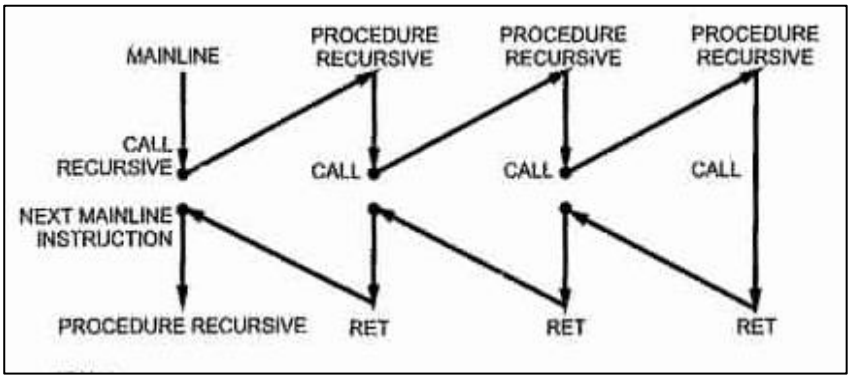
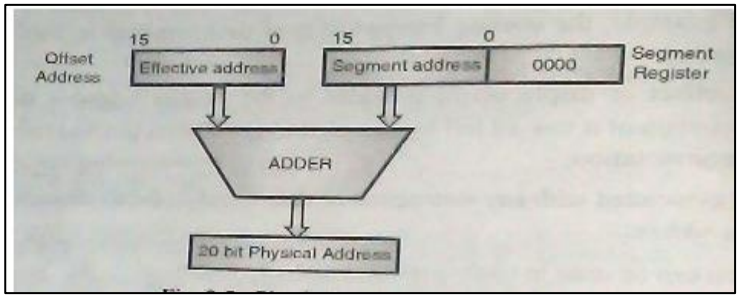
	b	**Write an ALP to count the number of positive and negative numbers in array.**	**4 M**
	Ans	;Count Positive No. And Negative No.S In Given ;Array Of 16 Bit No. **;Assume array of 6 no.s**	Correct program: 4 M

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

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

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

		JNC DOWN INC REULT_MSB DOWN:INC SI LOOP UP CODE ENDS DATA SEGMENT DATABLOCK DB 45H,02H,88H,29H,05H,45H,78H, 95H,62H,30H RESULT_LSB DB 0 RESULT_MSB DB 0 DATA ENDS END	
	d	**With neat sketches demonstrate the use of re-entrant and recursive procedure.**	**4 M**
	Ans	**Reentrant Procedure:** A reentrant procedure is one in which a single copy of the program code can be shared by multiple users during the same period of time. Re-entrance has two key aspects: The program code cannot modify itself and the local data for each user must be stored separately.  **Recursive procedures**: An active **procedure** that is invoked from within itself or from within another	Reentrant: 2 M and recursive procedure explanation With both diagram :2M

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

A9822

OR

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

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

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

	Ans	;Count ODD and EVEN No.S In Given ;Array Of 16 Bit No.
		;Assume array of 10 no.s
		
		CODE SEGMENT
		ASSUME CS:CODE,DS:DATA
		START: MOV AX,DATA
		MOV DS,AX
		MOV DX,0000H
		MOV CX,COUNT
		MOV SI, OFFSET ARRAY1
		NEXT: MOV AX,[SI]
		ROR AX,01H
		JC ODD_1
		INC DL
		JMP COUNT_IT
		ODD_1 : INC DH
		COUNT_IT: INC SI
		INC SI
		LOOP NEXT
		MOV ODD_COUNT,DH
		MOV EVENCNT,DL
		MOV AH,4CH
		INT 21H
		CODE ENDS
		
		DATA SEGMENT
		ARRAY1 DW F423H, 6523H, B658H, 7612H, 9875H,
		2300H, 1559H, 1000H, 4357H, 2981H
		COUNT DW 0AH
		ODD_COUNT DB ?
		EVENCNT DB ?
			Correct program: 4 M
			For basic logic may give 1-2 M

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

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

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

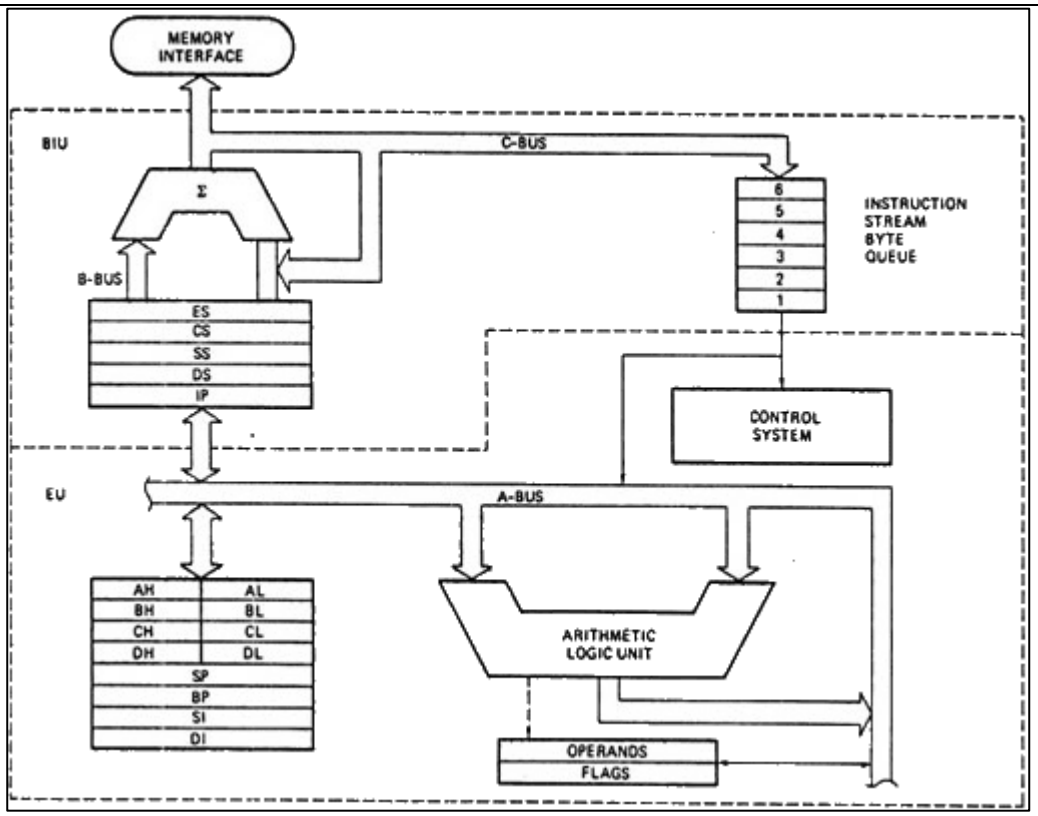
		START:MOV AX,DATA
		MOV DS,AX
		MOV AL,NUM1
		MOV BL,NUM2
		CALL SUM
		MOV CL,AL
		MOV AL, NUM3
		MOV BL,NUM4
		CALL SUM
		MUL CL
		MOV RESULT,AX
		MOV AH,4CH
		INT 21H
		CODE ENDS
		END	

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

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

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

Ans



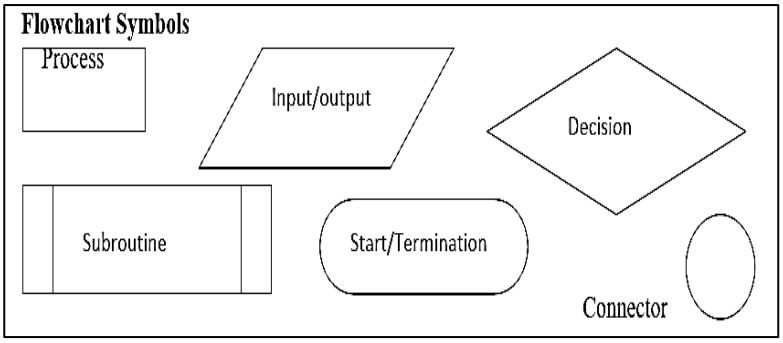
Register Organization of 8086

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

Diagram : 3M

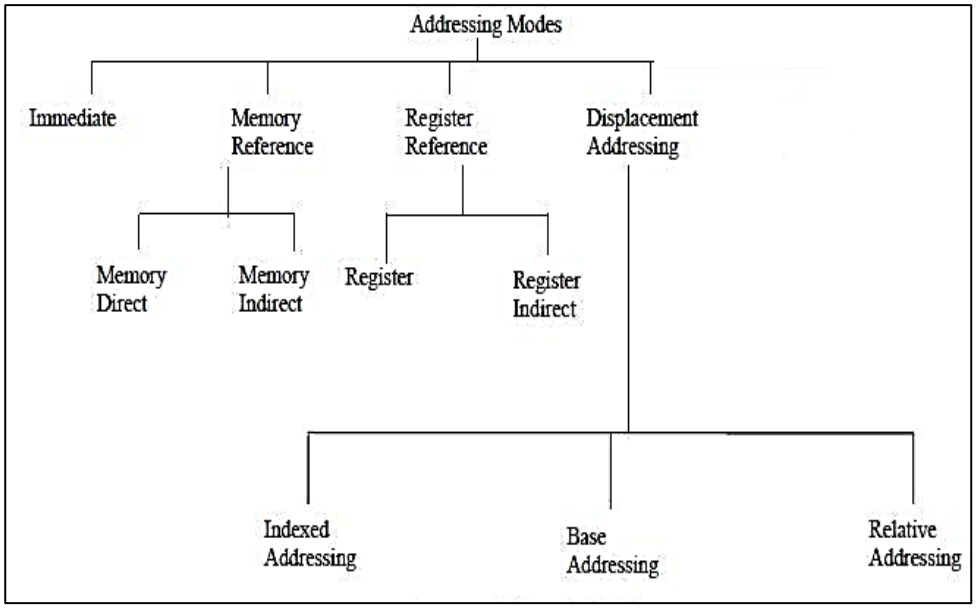
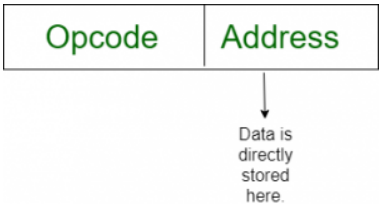
List of Register :1M,

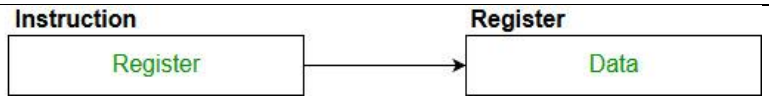
Any 4 registers explanation : ½ M each

		11. **SI** – Source Index – Used in string movement instructions. The source string is pointed to by the SI register. 12. **DI** – Destination Index – acts as the destination for string movement instructions 13. **IP** – Instruction Pointer – contains the offset of the next instruction to be executed. 14. **Flag Register** – individual bit positions within register show status of CPU or results of arithmetic operations.	
	b	**Demonstrate in detail the program development steps in assembly language programming.**	**6 M**
	Ans	**<u>Program Development steps</u>** **1. Defining the problem** The first step in writing program is to think very carefully about the problem that you want the program to solve. **2. Algorithm** The formula or sequence of operations or task need to perform by your program can be specified as a step in general English is called algorithm. **3. Flowchart** The flowchart is a graphically representation of the program operation or task.  **4. Initialization checklist** Initialization task is to make the checklist of entire variables, constants, all the registers, flags and programmable ports. **5. Choosing instructions** We should choose those instructions that make program smaller in size and more importantly efficient in execution. **6. Converting algorithms to assembly language program** Every step in the algorithm is converted into program statement using correct and efficient instructions or group of instructions.	Each step : 1M (Flowchart symbols are optional)

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

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

	Ans	**Different addressing modes of 8086 :**  **1. Immediate**: In this addressing mode, immediate data is a part of instruction, and appears in the form of successive byte or bytes. ex. MOV AX, 0050H  **2. Direct**: In the direct addressing mode, a 16 bit address (offset) is directly specified in the instruction as a part of it. ex. MOV AX ,[1 0 0 0 H]  **3. Register:** In register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers except IP may be used in this mode. ex. 1)MOV AX,BX	Any 6 addressing modes correct description: 1M each



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

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

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

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

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

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

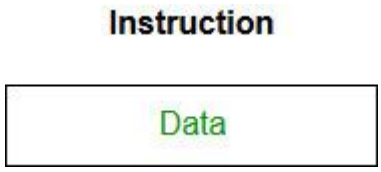
7. Based Indexed: In this addressing mode the effective address of the data is formed by adding the content of a base register (any one of BX or BP) to the content of an index register (any one of SI or DI). The default segment register may be ES or DS.

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

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

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

9 .Implied addressing mode:

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

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

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

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