MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION
(Autonomous)
(ISO/IEC - 27001 - 2005 Certified)

Winter-15 EXAMINATION
Model Answer

Subject Code: 17431 **Subject Name: Microprocessor & Programming**

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 judgment 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.

1. Attempt any FIVE of the following: Marks 20

a) List any eight features of 8085 processor.

(Any eight - ½ Marks each)

Ans :

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. Provide 74 instructions with five addressing modes.
5. 8085 is 8 bit microprocessor.
6. Provides 5 level hardware interrupts and 8 software interrupts.
7. It can generate 8 bit I/O address so 2^8=256 input and 256 output ports can be accessed.
8. Requires a single +5 volt supply
9. Requires 2 phase, 50% duty cycle TTL clock
10. Provide 2 serial I/O lines, so peripheral can be interfaced with 8085 µp

b) State the functions of the following pin of 8086 microprocessor

(i) ALE
(ii) DT/*R*
(iii) HOLD
(iv) M/*IO*
(Correct one function: 1 Mark each)
Ans:

(i) **ALE:** This output signal is used to indicate availability of valid address on address/data lines and is connected to latch enable pin of latches(8282 or 74LS373) .This signal is active high and never tri-state.

(ii) **DT/*R*:** This output line is used to decide the direction of data flow through the transceivers(bidirectional buffer). When the processor sends the data, this signal is high and when the processor is receiving data, this signal is low.

(iii) **HOLD:** This line is used to indicate to 8086 that another master is requesting the bus access.

(iv) **(M/*IO*** :This signal is used to distinguish between Memory access and I/O access. The M/IO is output early in the T1 state and identifies the current bus cycle as a memory(M/IO =1) or I/O (M/IO =0).

c) Name the different types of jump instructions used in 8086 assembly language program. (any eight)

(Any eight : ½ Mark each; Any Other can be considered)
Ans:

1. JMP Label : unconditional jump to address using 16 bit displacement or CS:IP
2. *JZ* Label : Transfer control to 'Label' if $ZF = 1$
3. *JNZ* Label : Transfer control to 'Label' if $ZF = 0$
4. *JS* Label : Transfer control to 'Label' if $SF = 1$
5. *JNS* Label : Transfer control to 'Label' if $SF = 0$
6. *JO* Label : Transfer control to 'Label' if $OF = 1$
7. *JNO* Label : Transfer control to 'Label' if $OF = 0$
8. *JP* Label : Transfer control to 'Label' if $PF = 1$
9. *JNP* Label : Transfer control to 'Label' if $PF = 0$
10. *JPO* Label : Transfer control to 'Label' if parity odd
11. *JPE* Label : Transfer control to 'Label' if parity even

12. *JB* Label : Transfer control to 'Label' if $CF = 1$
13. *JNB* Label : Transfer control to 'Label' if $CF = 0$
14. *JBE* Label : Transfer control to 'Label' if $CF = 1$ or $ZF = 1$
15. *JNBE* Label : Transfer control to 'Label' if $CF = 0$ or $ZF = 0$
16. *JL* Label : Transfer control to 'Label' if $SF = 1$ or $OF = 1$
17. *JNL* Label : Transfer control to 'Label' if $SF = 0$ or $OF = 0$
18. *JLE* Label : Transfer control to 'Label' if $ZF = 1$ or neither *SF* nor *OF* is 1
19. *JNLE* Label : Transfer control to 'Label' if $ZF = 0$ or at least any one of *SF* and *OF* is 1

d) State the steps involved in program development.
(Correct steps : 4 Marks)
Ans:

1. **Defining the problem:** The first step in writing program is to think very carefully about the problem that the program must solve.
2. **Algorithm:** The formula or sequence of operations to be performed by the 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:** 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.

e) Write the program structure for writing program in assembly language with suitable comment.
(Data segment: 2 Marks, Code segment : 2Marks Any one structure can be considered)
Ans: Structure 1: Using SEGMENT , ASSUME and ENDS directive

```
MY_DATA SEGMENT
        ---
        Program Data Declaration
        [data segment of program]
        ----
MY_DATA ENDS
MY_CODE SEGMENT
ASSUME CS: MY_CODE ,DS:MY_DATA ;Assign names to logical segments.
    START:MOV AX,MY_DATA  ; initialization of data segment
    MOV DS,DX
      ---
```

```
        [Program codes][code segment]
        ----
MY_CODE ENDS
END START
```

(OR)

Structure 2: Using .Data and .Code

```
.MODEL SMALL
.STACK 100
.DATA
        ---
        Program Data Declaration
        [data segment of program]
        ----
.CODE
    MOV AX,@DATA  ; initialization of data segement
    MOV DS,AX
        ---
        [Program codes][code segment]
        ----
 END
```

f) Define MACRO. List any four advantages of it.
(Correct Definition: 2 Marks, any 4 advantages: ½ Mark each)

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, such repeated code can be written separately called as Macro.
- Macro is also called as open subroutine.

(OR)

Macro definition or (Macro directive):

Syntax:

```
Macro_name MACRO[arg1,arg2,…..argN)
…..
 ENDM
```

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.

g) What do you mean by re-entrant procedures? Write any assembly language program with re-entrant procedure.

(Re-entrant procedure: 2 Marks, Example: 2 Marks)
(Program not expected)

Ans :

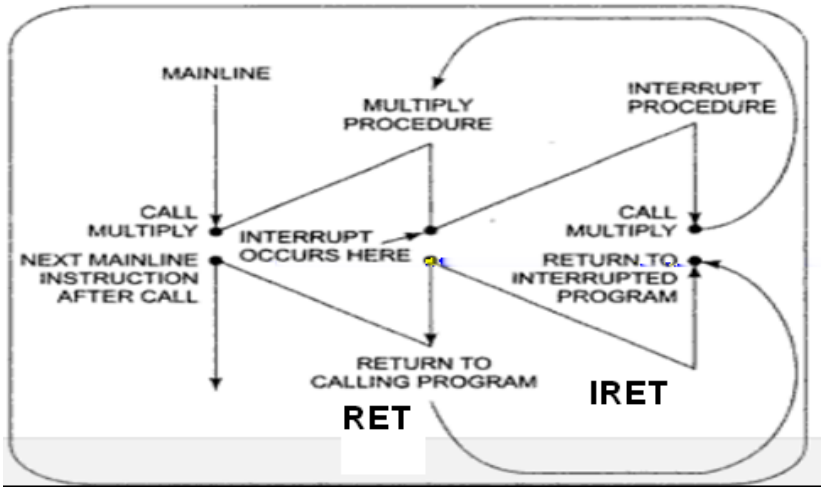
Re-entrant Procedures:

A procedure is said to be re-entrant, if it can be interrupted, used and re-entered without losing or writing over anything.

To be a re-entrant,

- Procedure must first push all the flags and registers used in the procedure.
- It should also use only registers or stack to pass parameters.

In this example



1) The main program calls the multiply procedure.
2) In between execution of multiply procedure , when an interrupt occurs, 8086 branches to interrupt service routine
3) ISR routine then calls the multiply procedure when it needs it. The RET instruction at the end of multiply procedure returns execution to the ISR.
4) After execution of ISR, an IRET instruction returns program execution to the multiply procedure at the instruction next to the point where the interrupt occurred.
5) Once again multiply is executed with data values of the main program.
6) A RET at the end of multiply returns execution to the main program.

2. Attempt any FOUR of the following: Marks 16

a) State the function of stack pointer and program counter in 8085 microprocessor.

(Any 2 functions of each: 2 Marks)

Ans:
Stack pointer:

1. It is a 16 bit register which is used to store the address of topmost filled memory location of stack memory.
2. SP always points current top of stack.
3. If data is stored in stack memory, the content of stack pointer is auto-decremented by two and if data is picked out from stack memory, the content of SP is auto-incremented by two.

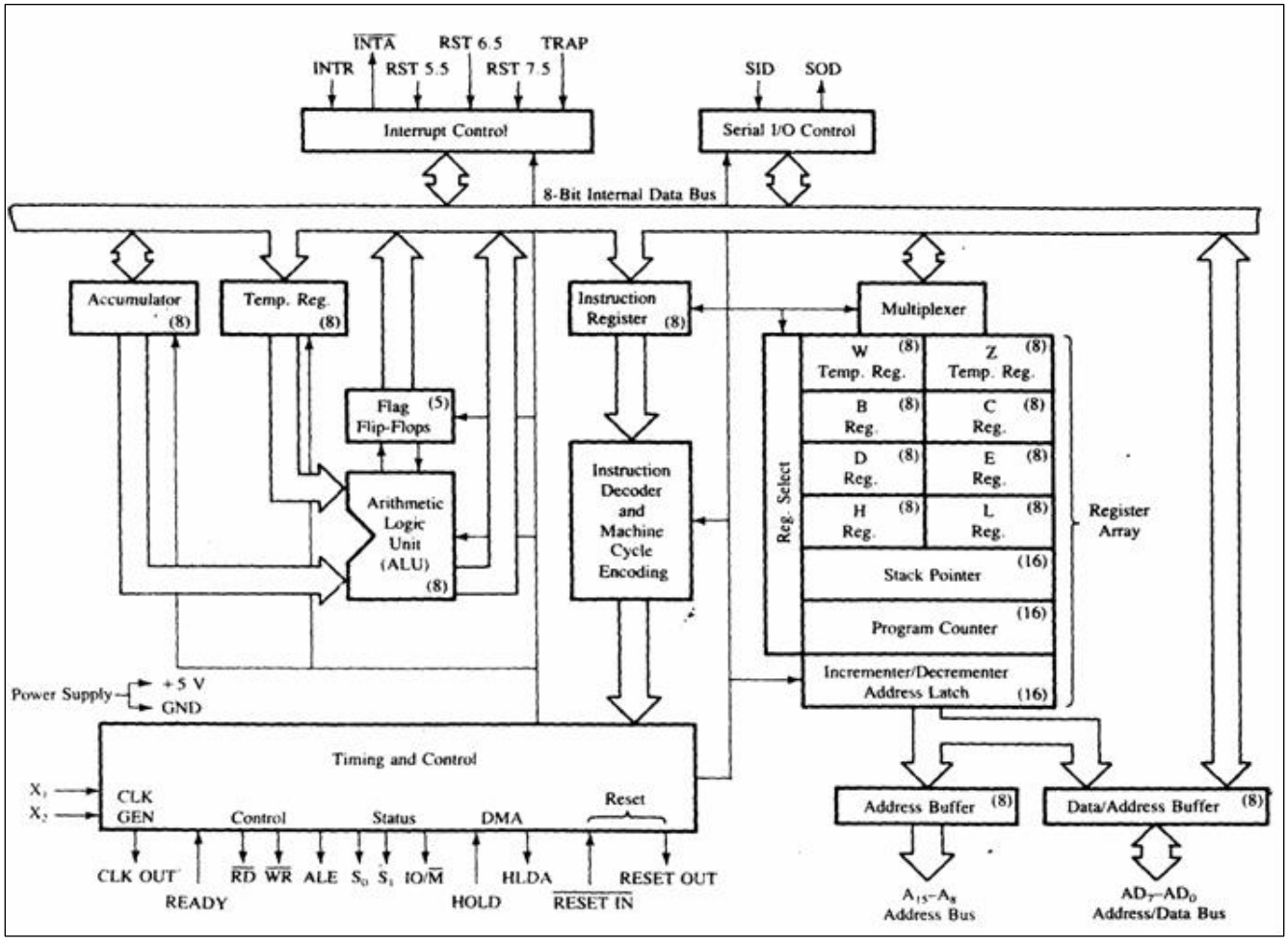
Program counter:

1. It maintains sequential execution of program written in memory.
2. The PC stores the address of the next instruction which is going to execute.
3. Since program counter stores the address of memory and in 8085 the address of memory is 16 bit. Hence program counter is 16 bit register.

b) Draw a neat labeled Architectural diagram of 8085 microprocessor.

(Correct Diagram: 4 Marks)

Ans:



c) Differentiate between minimum mode and maximum mode of 8086 microprocessor (Eight points)

(Any eight points: ½ Mark each)

Ans:

Sr. No.	Minimum mode	Maximum mode
1.	MN/MX pin is connected to $V_{CC.}$ i.e. MN/MX= 1.	MN/MX pin is connected to ground. i.e. MN/MX = 0.
2.	Control system M/IO, RD, WR is available on 8086 directly.	Control system M/IO, RD, 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 IOR, IOW, $MEMW$, $MEMR$ can be generated using control signals M/IO, RD, WR which are available on 8086 directly.	Control signals such as $MRDC$, $MWTC$, $AMWC$, $IORC$, $IOWC$ and $AIOWC$ are generated by bus controller 8288.
6.	ALE, DEN, DT/R and $INTA$ signals are directly available.	ALE, DEN, DT/R and $INTA$ 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.	$RQ/GT0$ and $RQ/GT1$ 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 .

d) Describe the generation of physical address in 8086. If CS=2000 H, and IP = 1122 H, calculate the physical address generation.

(Description - 2 Marks, Example - 2 Marks)

Ans:

Generation of a physical address in 8086 :- Segment registers carry 16 bit data, which is also known as base address. BIU attaches four 0 bits to 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.

Example:- Given : CS = 2000H, IP = 1122H

CS : 20000H ……...0 added by BIU(or Hardwired 0)

\+ IP : 1122H

21122H

e) State the functions of –

(i) Editor

(ii) Assembler

(Any 2 Functions of each - 2 Marks)

Ans:

(i) Editor

1. It is a program which helps to construct assembly language program with a file extension .asm, in right format so that the assembler will translate it to machine language.
2. It enables one to create, edit, save, copy and make modification in source file.

(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

f) What are assembler directives? Explain any two assemble directives.

(Assembler Directives - 1 Mark, Explanation of any two - 1½ Mark each; Any other directives can also considered)

Ans: Assembler Directives: Assembly language program supports a number of reserved words i.e keywords that enables the assembler to control the way in which a program assembles and lists. Assembler directives are the statements that give direction to the assembler and also called as pseudo-instructions that are not translated into machine code.

1) DB :Define byte(8 bits)

It is used to declare a byte type variable of 8 bit. It also can be used to declare an array of bytes.
The range of values that can be stored in a byte is 0 to 255 for unsigned numbers and -128 to +127 for unsigned numbers.
e.g
NUM DB ? ; Allocate one memory location
ARRAY DB 12,25,26,55,65 ; Allocate five memory locations

2) DW: -Define Word (16-bits)
It is used to tell the assembler to define a variable of type word in memory or to reserve storage locations of type word (16) in memory.
e.g. BLOCK DW 1234H, 3456H, 5678H : Declare array of 3 words.

3) DD: -Define Double word (32-bits)
It is used to declare a variable of type doubleword or to reserve memory locations which can be accessed as type doubleword(32-bits.)
e.g. NUMBER DD 1,2,3,4,9 ; allocated 20 memory locations.

4) DQ : Define QuadWord(4 words)
This directive is used to tell the assembler to declare a variable 4 words in length or to reserve 4 words of storage
NUM DQ ? ;Allocate 8 memory locations

5)DT :Define Ten byte . This directive is used to tell the assembler to define a variable which is 10 bytes in length or to reserve 10 bytes of storage in memory.
NUM DT ? ; Allocate Ten memory locations;

6)EQU :Equate to
The EQU directive is used to declare the micro symbols to which some constant value is assigned. Micro assembler will replace every occurrence of the symbol in a program by its value.

Syntax: Symbol_name EQU expression
e.g CORRECTION_FACTOR EQU 100

7) ORG : Originate
The directive ORG assigns the location counter with value specified in the directive. It helps in placing the machine code in the specified location while translating instructions into machine codes by the assembler. $ is used to indicate current value of location counter
Syntax: ORG [$+] Numeric_value
e.g ORG 2000H ; set location counter to 2000H
ORG $+ 100 ; increment value of location counter by 100 from its current.

8) ALIGN : Alignment of memory addresses
This directive is used to force the assembler to align the next data item or instruction according to given value.
Syntax: ALIGN Numeric_value
e.g ALIGN 4 ;advances location counter to the next address that is evenly divisible by 4

9) EVEN: - The directive even is used to tell assembler to increment the location counter to the next even memory address.If the location counter is already pointing to even address it should not be increment.
Example: - DATA SEGMENT
Array DB 9 DUP (?)
EVEN
Block DW 100H DUP (0)
DATA ENDS

10) LABEL:
The LABEL directive is used to give a name to the current value in the location counter .IT enables you to redefine the attributes of a data variable or instruction label.
Syntax: variable_name LABEL type_specifier
e.g STACK_TOP LABEL WORD
TEMP LABEL BYTE

11) DUP: Duplicate memory location
This directive can be used to generate multiple bytes or words with known as well as un-initialized values.

e.g TABLE DW 100 DUP(0) ; Create array of 100 words all contains data 0

12) ASSUME: - Assume directive is used to tell Assembler the name of the logical segment it should use for the specified segment.
When program is loaded the processor segment register should point to the respective logical segments.
Example: - Assume CS: MSBTE_CODE, DS: MSBTE_DATA

13) SEGMENT: Used to indicate the beginning of logical segment. Preceding the SEGMENT directive is the name of the given segment
Syntax:
Segment_Name SEGMENT [Word/Public]
e.g My_data SEGMENT

My_data ENDS

14) ENDS : End of segment . The directive ENDS is used with the name of a segment to indicate the end of that logical segment (which contains instructions or data).
CODE SEGMENT ;Start of logical segment
------ ;Instruction statements
CODE ENDS ;End of segment named CODE.

15) END: End of Program
This directive is used to inform assembler the end of the program. The assembler will ignore any statements after an END directive.

16) Length: The directive length determines the number of elements in some named data item, such as a string or an array
Syntax: LENGTH Variable_Name
MOV CX,Length Array ; Find no of elements in Array and load length to CX.

17) Size: This directive returns the number of bytes allocated to the data item instead of number of elements in it
Syntax: SIZE Variable_name

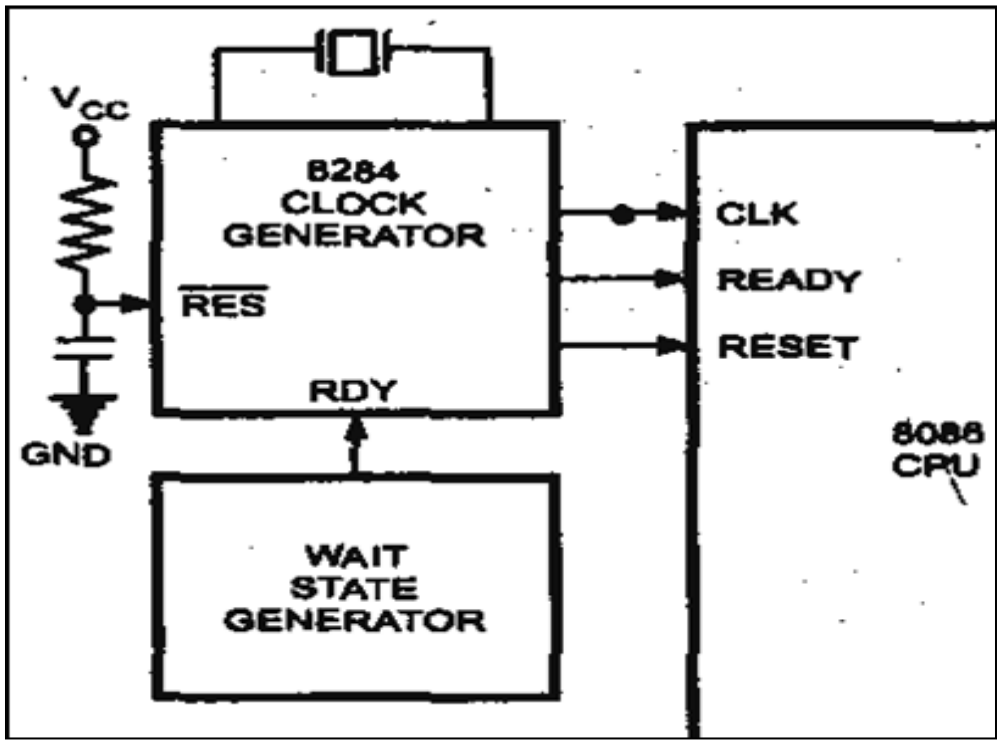
Example : MOV AX, SIZE Total ; Find no of bytes in Total and load length to CX.

18) OFFSET: This directive tells the assembler to determine the offset or displacement of a named data item or procedure from the start of segment which contains it. Used to load the offset of variable into a register so that variable can be accessed with one of the indexed addressing mode.
Syntax : OFFSET Variable_name
Ex OFFSET BX,PRICES ;determine the offset of variable PRICES from seg start and load displacement in BX

3. Attempt any FOUR of the following : Marks 16

a) Draw and explain the interfacing of 8284 clock generator with 8086 microprocessor.
(Diagram - 2 Marks, explanation -2 Marks)

Ans:



1. Generate system clock: The Clock logic results in three different frequencies required for the system. These outputs are CLOCK, OSC and PCLK.
2. Generate READY signal: The Ready logic has a READY output which is connected to the processor. When this is low, wait states are added in the bus cycle.
3. Generate RESET signal: The Reset logic generates a RESET input for the microprocessor. When this signal is High, processor performs reset sequence.

b) What is memory segmentation? How it is done in 8086 microprocessor?

(memory segmentation – 2 Marks, explanation - 2 Marks)

Ans:

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

Description:

- Data segment is used to hold data, Code segment for the executable program, Extra segment also holds data specifically in strings and stack segment is used to store stack data.
- Each segment is 64Kbytes & addressed by one segment register.
- The 16 bit segment register holds the starting address of the segment
- The offset address to this segment address is specified as a 16-bit displacement (offset) between 0000 to FFFFH.
- Since the memory size of 8086 is 1Mbytes, total 16 segments are possible with each having 64Kbytes.

c) Identify the addressing modes in following instructions

(i) MUL AL, BL
(ii) MOV AX, 2100 H
(iii) MOV AL, DS : [SI]
(iv) MOV AX, BX.

(Each addressing mode - 1 Mark each)

Ans:

i) MUL AL,BL:- Register addressing mode (Incorrect MUL instruction; corrected as **MUL BL**)
ii) MOV AX ,BX:- Register addressing mode
iii) MOV AX,2100H:-Immediate addressing mode

iv) MOV AL,DS:[SI] :-Indexed addressing mode

d) State and explain any four addressing modes of 8086 microprocessor with example.
(Each Addressing mode -1 Mark)

Ans:

Addressing modes of 8086:

1. Immediate addressing mode:

An instruction in which 8-bit or 16-bit operand(data) is specified in the instruction, then the addressing mode of such instruction is known as Immediate addressing mode.

Example:

MOV AX,67D3H

2. Register addressing mode

An instruction in which an operand(data) is specified in general purpose registers, then the addressing mode is known as register addressing mode.

Example:

MOV AX,CX

3. Direct addressing mode

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.

Example:

MOV CL,[2000H]

4. Register Indirect addressing mode

An instruction in which address of an operand is specified in pointer register or in index register or in BX, then the addressing mode is known as register indirect addressing mode.

Example:

MOV AX, [BX]

5. Indexed addressing mode

An instruction in which the offset address of an operand is stored in index registers (SI or DI) then the addressing mode of such instruction is known as indexed addressing mode.

DS is the default segment for SI and DI.

For string instructions DS and ES are the default segments for SI and DI resp.

this is a special case of register indirect addressing mode.

Example:

MOV AX,[SI]

6. Based Indexed addressing mode

an instruction in which the address of an operand is obtained by adding the contents 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

Example:

MOV AX, [BX][SI]

7. Register relative addressing mode

An instruction in which the address of the operand is obtained by adding the displacement (8-bit or 16 bit) with the contents of base registers or index registers (BX, BP, SI, DI). the default segment register is DS or ES

Example:

MOV AX, 50H[BX]

8.Relative Based Indexed addressing mode

An instruction in which the address of the operand is obtained by adding the displacement (8 bit or 16 bit) with the base registers(BX or BP) and index registers (SI or DI) to the default segment.

Example:

MOV AX, 50H [BX][SI]

e) Write an ALP for 8086 to find the largest number in an array.
[Assume array size of 10]
(Correct program - 4 Marks)

Ans:

```
        DATA SEGMENT
    ARRAY DB
5H,45H,08H,56H,78H,75H,10H,11H,20H,24H
LARGEST DB 00H
 DATA ENDS
  CODE SEGMENT
  ASSUME CS: CODE, DS: DATA
 START:  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 SMALLEST,AL
MOV AX, 4C00H
INT 21H
     CODE ENDS
     END START
```

f) Write an ALP for 8086 to perform BCD addition of two number [Assume suitable data]

(Correct program - 4 Marks)

(Program with any other logic also be considered)

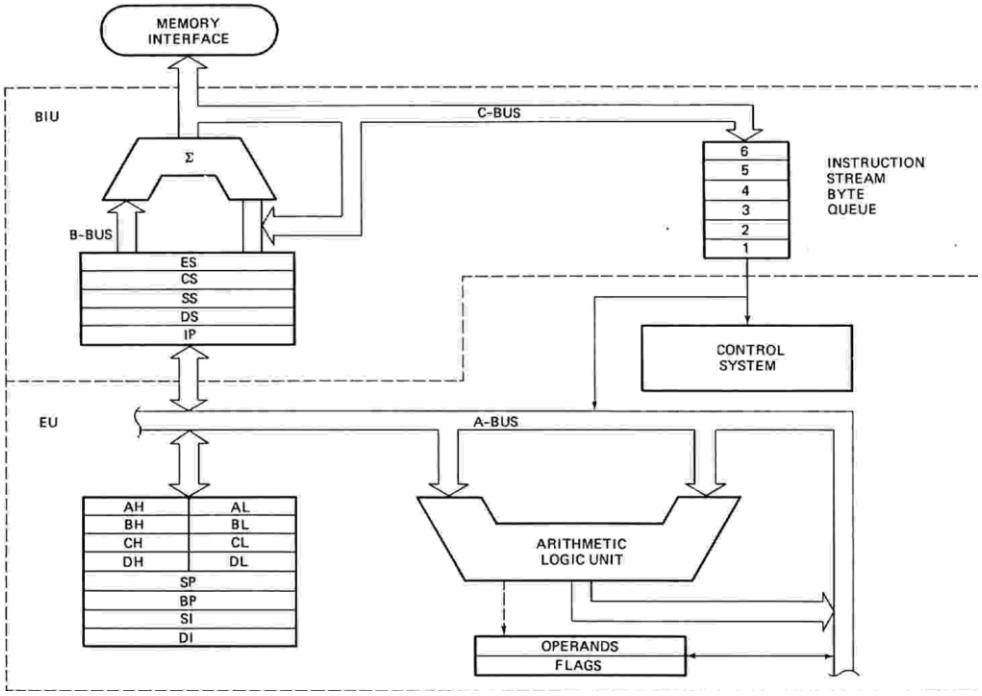
Ans:

```
.MODEL SMALL
.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
```

4. Attempt any FOUR of the following : Marks 16

a) Draw the architecture of 8086 microprocessor and state the function of BIU.
(Architecture - 3 Marks, any one function -1 Mark.)

Ans:



Functions:

- Generates physical address
- Sends address to memory or I/O
- Fetch instruction from memory
- Read data from memory or I/O
- Write data to memory or I/O
- Supports instruction queuing
- Provides address relocation facility

b) List all the 16 bit registers of 8086 and write their function.

(List – 2 Marks, Function - 2 Marks)

Ans.

16 bit registers: AX,BX,CX,DX,CS,SS,DS,ES,BP,SP,SI,DI,IP,FLAG REGISTER

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.
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. **Flags Register** – individual bit positions within register show status of CPU or results of arithmetic operations.

c) Explain Near CALL and Far CALL procedure.

(NEAR CALL - 2 Marks, FAR CALL - 2 Marks)

Ans:

NEAR CALL

- A Near call is a call to a procedure which is in the same code segment.
- In Near call the contents of SP is decremented by '2' and the content of offset address IP is stored
- The contents of CS is not stored

Example: - Call Delay

Operation performed :

SP = SP – 2

Save IP on stack

IP = Address of procedure

FAR CALL:

- A Far call is a call to a procedure which is in each difference different code segment.
- In Far call the contents of SP is decremented by '2' and value of CS is loaded .Then SP is again decremented by 2 and IP is loaded.
- The contents of CS is also stored along with offset

Example :- Call FAR PTR Delay
Operation performed :
SP = SP – 2
Save CS on stack
CS = new segment base address of the called procedure
SP =SP-2
Save IP on the stack and
IP = New offset Address of the called procedure

d) Explain the following instructions of 8086

(i) DAA

(ii) XLAT

(Each instruction explanation - 2 Marks)

Ans:

(i) DAA (Decimal Adjust Accumulator):

Syntax :-- DAA

- This instruction is used to convert the result of the addition of two packed BCD numbers to a valid BCD number.
- The result has to be only in AL.
- After addition if the lower nibble is greater than 9 or AF =1, it will add 06H to the lower nibble in AL.
- After this addition, if the upper nibble is greater than 9 or if CF = 1, DAA instruction adds 60H to AL.
- DAA instruction affects AF,CF,PF and ZF. OF is undefined.

(ii) XLAT

- XLAT : translate
  - Can be used for look up table
  - Default source & destination operand is AL
  - Default base address of look up table is in BX
  - Physical address in look up table = 10H * DS + AL + BX
  - Example:

```
MOV AL, NUM; read the number
MOV BX, OFFSET_TABLE; store the base address of look up table
XLAT
```

e) Write an ALP for 8086 to count the number of odd numbers in array.
[Assume array size of 20 numbers]
(Correct Program - 4 Marks) (Any other shift or rotate instructions may be considered)
Ans:

```
DATA SEGMENT
 NUM DB 10H,12H,11H, 36H,45H,74H,62H,45H,91H,05H,42H,71H,38H,10H,12H,11H,
36H,45H,74H,62H
 COUNT DB ?
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START: MOV AX,DATA
   MOV DS,AX
   MOV CX,14H
   MOV SI, OFFSET NUM
 NEXT:  MOV AL, [SI]
   ROR AL,01
   JNC DOWN
    INC COUNT
   DOWN:   INC SI
    LOOP NEXT
MOV AX, 4C00H
INT 21H
CODE   ENDS
       END START
```

f) Write an ALP to count the number of '1' in a number stored in accumulator.
(Correct program - 4 Marks) **(Any other shift or rotate instructions may be considered)**
(8 bit or 16 bit can be considered)
Ans:

```
DATA SEGMENT
NUM DB 02H
DATA ENDS
```

```
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START:
MOV DX,DATA
MOV DS,DX
MOV CX,08H
MOV BL,00H
MOV AL,NUM
UP:ROR AL,1
JC DN
INC BL
DN:LOOP UP
MOV AX, 4C00H
INT 21H
CODE ENDS
END START
```

Or (16 Bit)

```
DATA SEGMENT
NUM DW 1102H
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START:
MOV DX,DATA
MOV DS,DX
MOV CX,10H
MOV BL,00H
MOV AX,NUM
UP:ROR AX,1
JC DN
INC BL
DN:LOOP UP
MOV AX, 4C00H
INT 21H
CODE ENDS
END START
```

5. Attempt any TWO of the following: 16

a) (i) Draw flag register structure of 8086 and describe operation of each flag.
(Diagram - 2 Marks, description – 2 Marks)

Ans:

Flag Register

8086 has a 16-bit flag register as shown in figure.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	0	D	I	T	S	Z	X	AC	X	P	X	CY

O - Overflow
D- Direction flag
I - Interrupt flag
T-Trap flag
S-Sign flag
Z-Zero flag
AC-Auxiliary carry flag
P-Parity flag
CY-Carry flag
X- Not used

The description of each flag bit is as follows:

S-Sign Flag This flag is set when the result of any computation is negative. For signed computations the sign flag equals the MSB of the result.

Z-Zero Flag This flag is set if the result of the computation or comparison performed by the previous Instruction/ Instructions is zero

P-Parity Flag This flag is set to 1 if the lower byte of the result contains even number of 1s

C-Carry Flag This flag is set when there is carry out of MSB in case in addition or a borrow in case of subtraction

T-Trap Flag If this flag is set ,the processor enters the single step execution mode. In other words a Trap interrupt is generated after execution of each instruction .

I-Interrupt Flag If this flag is set , the maskable interrupts are recognized by the CPU , otherwise they are ignored.

D-Direction Flag This is used by string manipulation instructions to indicate the direction of string operation.

AC-Auxiliary Carry Flag This is set if there is a carry from the lowest nibble, i.e. bit three , during addition or borrow for the lowest nibble , i.e. bit three, during subtraction

O-Overflow Flag This flag is set if an overflow occurs, i.e. if the result of a signed operation is large enough to be accommodated in destination register.

(ii) What is pipelining? State its need and how it is done in 8086
(***Definition -1 Mark, Description -3 Marks)***

Ans:

Definition: Process of fetching the next instruction while the current instruction is executing is called pipelining which will reduce the execution time.

Description: The technique used to enable an instruction to complete with each clock cycle. Normally, on a non – pipelined processor, nine clock cycles are required for fetch, decode and execute cycles for the three instructions as shown in Fig(a). This takes longer time when compared to pipelined processor. In this ,the fetch, decode and execute operations are performed in parallel, so only five clock cycles are required to execute the same three instructions as shown Fig(b). 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 instruction and fetches 3^{ed} 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)

Needs of Pipelining:

- Pipelining enables many instructions to be execute at the same time.
- It allows execution to be done in fewer cycles.
- Speed up the execution speed of the processor
- More efficient use of processor

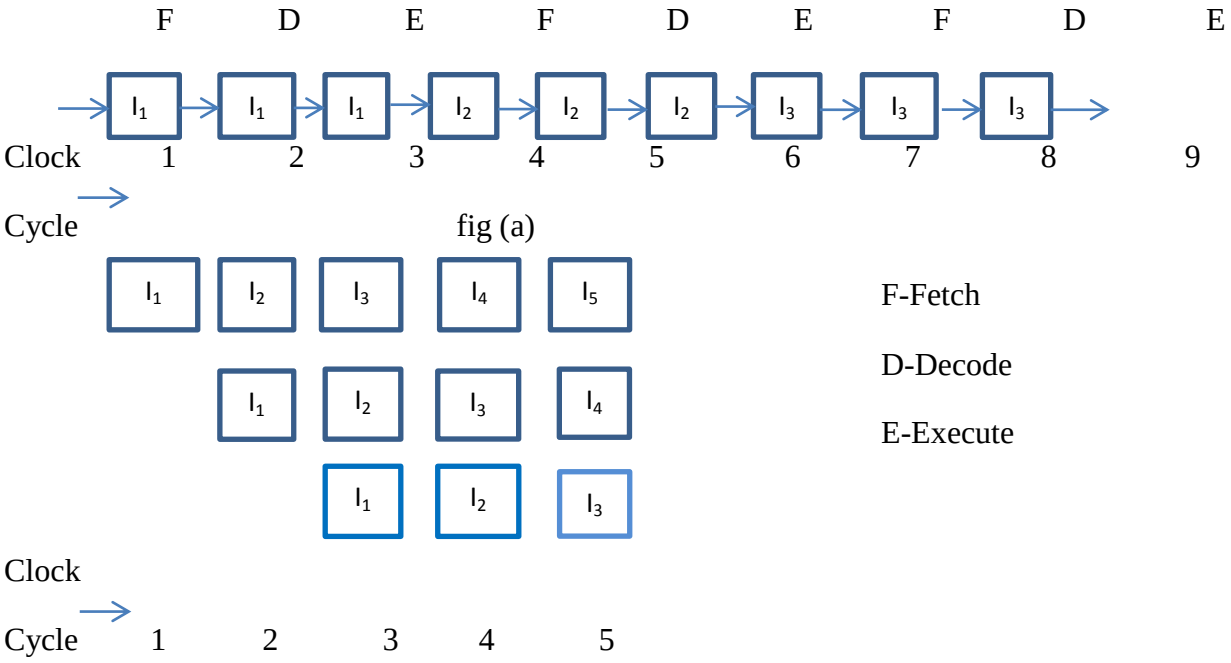


fig (b)

b) Write an ALP to sort a array of 10 numbers in Ascending order.
(Correct program - 4 Marks)
(Eeither 8 bit or 16 bit data may be considered)

Ans:

```
DATA SEGMENT
        ARRAY DB 06H,09H,22H,02H,07H,10H,11H,12H,13H,14H
```

```
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START:  MOV AX,DATA
      MOV DS,AX
      MOV BL,0AH
         STEP1:  MOV SI, OFFSET ARRAY
          MOV CL,09H
         STEP2:  MOV AL,[SI]
          CMP AL,[SI+1]
         JC DOWN
          XCHG AL, [SI+1]
          XCHG AL,[SI]
DOWN:   ADD SI, 01
LOOP STEP2
         DEC BL
 JNZ STEP1
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

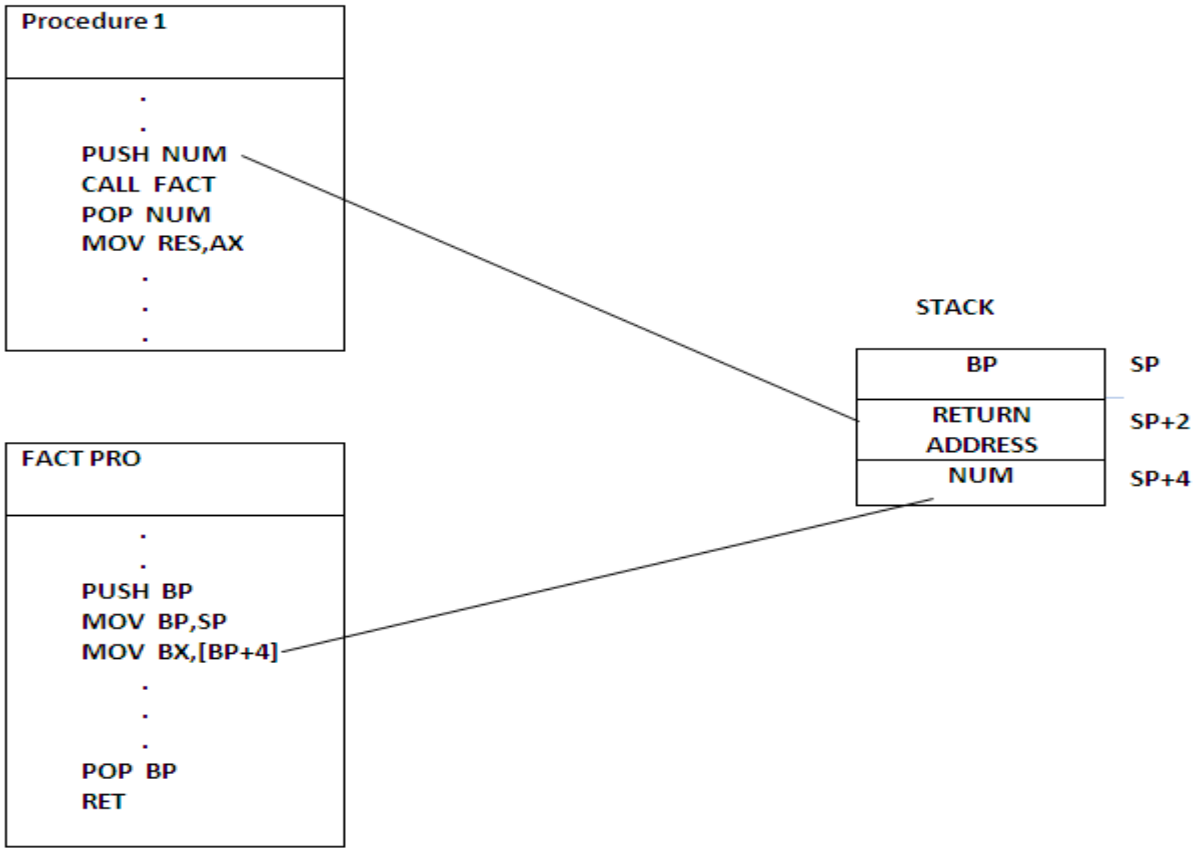
c) Describe with suitable example how parameter is passed on the stack in 8086 assembly language procedure.

(Description - 2 Marks, Example - 2 Marks) (Any other Example may be considered)

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.



6. Attempt any FOUR of the following: **Marks 16**

a) List the string related instructions of 8086 microprocessor and explain any two instructions.
(List - 2 Marks, explanation of any two instructions – 2 Marks)

Ans:

String related instructions of 8086 microprocessor are:

1. MOVS
2. MOVSB
3. MOVSW
4. LODS
5. LODSB

6. LODSW
7. STOS
8. STOSB
9. STOSW
10. CMPS
11. CMPSB
12. CMPSW
13. SCAS
14. SCASB
15. SCASW

1] MOVS/ MOVSB/ MOVSW - Move String byte or word.
Syntax
MOVS destination, source
MOVSB
MOVSW
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 (DF) 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
CMPSW
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. CX 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.

b) Write the appropriate 8086 instructions to perform the following operation.

(i) Multiply AL register contents by 4 using shift instructions.
(ii) Move 2000 H into CS register.

((i) Correct instructions – 2 Marks)
((ii) Examiner can give marks if question is attempted.)

Ans:

I)

```
MOV CL,02H
SHL AL,CL
```

II)

The contents if CS register cannot be modified directly , Hence no instructions are used However examiner can give marks if question is attempted.

c) Write an ALP for 8086 to multiply two 16 bit numbers.

(correct program - 4 Marks)

Ans :

```
DATA SEGMENT
NUM1  DW  0002H
NUM2  DW  0003H
L_RES  DW  ?
H_RES  DW  ?
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START: MOV    AX,DATA  ;INITIALIZE DATA SEGMENTS
MOV     DS,AX
MOV     AX,NUM1     ; MOVE NUM1 IN AX
MUL     NUM2          ; MULTIPLY NUM2 WITH AX CONTENT
MOV     L_RES,AX ; STORE LOWER SIDE 16-BIT RESULT FROM AX TO L_RES
MOV     H_RES,DX; STORE HIGHER SIDE 16-BIT RESULT FROM DX TO L_RES
MOV    AH, 4CH     ;TERMINATE  THE PROGRAM
INT     21H
CODE ENDS
END STAR
```

d) Write an ALP to transfer a block of 50 numbers from 20000 H to 30000H.

(Correct program -4 Marks)

Ans.

```
CODE SEGMENT
ASSUME CS : CODE
START: MOV AX,2000H            ; INITIALIZE DATA SEGMENTS
MOV DS,AX
MOV AX,3000H          ; INITIALIZE EXTRA SEGMENTS
MOV ES,AX
MOV SI, 0000H      ;INITIALIZE MEMORY POINTER
MOV DI, 0000H
MOV CX, 0032H     ; INITIALIZE  COUNTER 50
UP: MOV AL,[SI]    ; TRANSFER DATA FROM SOURCE TO DESTINSTION
    MOV ES:[DI],AL
   INC SI             ;INCREMENT MEMORY PONTER
INC   DI
LOOP UP             ;CHECK COUNTER IF NOT ZERO THEN LOOP UP
MOV AH,4CH       ;TERMINETE THE PROGRAM
INT  21H
CODE ENDS
END START
```

e) Write an ALP to multiply two 8 bit numbers using NEAR procedure

(Correct program – 4 Marks)

Ans:

```
.MODEL SMALL
.DATA
NUM1  DB  04H
NUM2  DB  03H
RESULT   DW  ?

.CODE
```

```
MOV    AX,@DATA   ; INITIALIZE DATA SEGMENTS
MOV     DS, AX
CALL  MUL_NUM       ;  CALLING OF  MUL_NUM PROCEDURE
MOV AH,4CH        ; TERMINETE THE PROGRAM
INT  21H

MUL_NUM   PROC   ; STRAT OF MUL_NUM PROCEDURE
MOV   AL, NUM1     ;MOVE  NUM1 TO AL
MUL   NUM2          ;MULTIPLY  AL WITH NUM2
MOV RESULT, AX   ;MOVE  OUTPUT FROM AX TO RESULT
RET                  ;RETURN TO CALLING PROGRAM
ENDP                 ; END OF PROCEDURE

 ENDS                ;END OF SEGMENT
END                  ;END OF PROGRAM
```

f) What are the functions of CALL and RET instructions? Write the syntax of CALL and RET instructions.

(Function of each - 1 Mark, Syntax - 1 Mark each)

Ans :

CALL instruction: It is used to transfer program control to the sub-program or subroutine. The CALL can be NEAR, where the procedure is in the same segment whereas in FAR CALL, procedure is in a different segment.

RET instruction: it is used to transfer program execution control from a procedure to the next instruction immediate after the CALL instruction in the calling program.

Syntax for CALL:

CALL procedure_name

Syntax for RET:

RET