SUMMER – 13 EXAMINATION

Subject Code: **12109** **Model Answer**

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.

1. Attempt any <u>FIVE</u> of the following: (20)

a. Draw the flag register format and explain in brief various flags related to 8085.

Ans a.

[Diagram – 2 Marks , Explanation – 2 Marks]

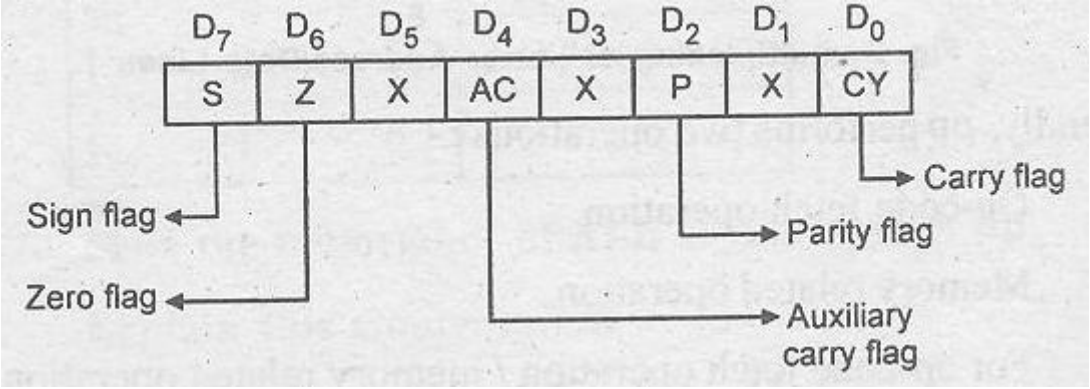

Fig: Format of flag register of 8085 µp

Flag register is a 8 bit register. In 8085 µp, outputs of five bits are called Flags. Each flag will indicate a particular status of the result. Hence flag register is also called Status register.
After arithmetic and logical operation, depending upon the status of the result, the flag indicates the particular status as described below:-

i) Carry flag (CY):
When µp performs addition of 8 bit or 16 bit numbers and if the carry is generated from the MSB, then the carry flag is set (CY=1), otherwise it resets the carry flag (CY=0). Similarly, when µp performs the subtraction of two 8 bit nos and if addition borrow is required, then carry flag is set, otherwise it is reset.

ii) Auxiliary carry flag (AC)/ Half carry/ Nibble carry:
When µp performs addition of 8 bit number and if the carry is generated from D_3^{th} bit, then auxiliary carry flag is set, otherwise it is reset.

iii) Parity flag (P):
The number of 1's in the 8 bit result is called as Parity.
When µp performs addition or logical operations on 8 bit number and if number of 1's bit in 8 bit result is even number, then it is called as Even parity and parity flag is set (P=1).
Similarly, if number of 1's in a 8 bit result odd number, then it is called as Odd parity and parity flag is reset(P=0).

iv) Zero parity (Z):
When µp performs arithmetic and logical operation of two 8 bit or 16 bit numbers and if the result obtained is zero, then flag is set (Z=1), otherwise it is reset (Z=0).

v) Sign flag (S):
When µp 8085 performs arithmethic and logical operations on unsigned numbers, then sign flag is of no use. But when it performs operation on signed numbers, then the MSB indicates the sign of the numbers.
For positive number, MSB bit is 0 & for negative number it is '1'.
Hence, when µp performs arithmethic and logical operations on signed numbers and if the MSB of the result is 1, then sign flag is set. i.e. for negative number sign flag is set (S=1), otherwise it is reset (S=0).

b. Describe the function of SID and SOD pins of 8085 microprocessor.

Ans b.

SID: - (2 Marks)

Serial Input Data – SID pin is used to receive data serially from external devices to the accumulator.

At a time only one bit is received.

LSB of 8-bit data is received first.

SOD: - (2 Marks)

Serial Output Data – SOD pin is used to transmit data serially from accumulator to the external devices connected to the pin.

At a time only one bit is transmitted.

LSB of data is transmitted first.

c. List eight features of 8086. (**1/2 Marks for Each)**

Ans c.

Features of 8086 µp are:-

i) It is a 16- bit µp
ii) It has 20- bit address lines. Hence it can access 2^{20}= 1 M bytes of memory location.
iii) It requires single +5V power supply.
iv) It has multiplexed address and data bus, which reduces the number of pins needed.
v) It is designed to operate in two modes (Minimum mode and Maximum mode.)'
vi) It supports multiprogramming.
vii) To speed up instruction execution, the 8086 prefetches up to six instructions ahead from queue register.
viii) It can generate 16- bit I/O addresses, hence it can access (2^{16} = 65535=) 64k I/O ports.
ix) It provides 14, 16- bit registers.

d. Explain the following instructions of 8086.

i) XLAT
ii) XCHG

Ans d.

i. XLAT (Translate) : (2 Marks)

It is a data transfer instruction. It is used for finding the codes in case of code conversion problem using look up table technique.

The XLAT instruction replaces a byte in the AL register with a byte from a look up table in memory.

ii. XCHG (Exchange): (2 Marks)

XCHG destination, source. It is data transfer instruction. It exchanges the contents of the specified source and destination operands, which may be register or one of them may be memory location.

e. What are assembler directives? Explain the following directives

i) ASSUME

ii) EVEN

Ans e.

i. ASSUME: (2 Marks)

It tells the assembler the name of a logical segment, which is to be used for a specified segment.

Its general format is

ASSUME segment Reg: Segment Name

e.g. DATA_ SEG SEGMENT: Start of DATA _SEG

ASSUME DS : DATA_SEG

It tells the assembler that for any program instruction which refers to the data segment, the logical segment named DATA-SEG is to be used.

ii. EVEN: (2 Marks)

Align as even memory address.

It tells the assembler to increment the location counter to the next even memory address. If it is already pointer then it should not be incremented.

Its general format is.

EVEN.

f. Define MACRO with example.

Ans f. **(4 Marks)**

In an assembly language program, if number of instructions is repeated again and again in main program at different places which perfoms the same operation, then such repeated instructions can be written separately called as MACRO.

A macro can be defined anywhere in a program using directives MACRO and ENDM.

e.g. macro name

```
DISPLAY MACRO
        -------------------------
        MACRO Statements
        -------------------------
        ENDM
```

The label prior to MACRO is the macro name which should be used in the actual program. The ENDM directive marks the end of the instructions sequence assigned with macro name.

A macro can be called by quoting its name along with any values to be passed to the macro.

g.What is procedure? What are the advantages of using procedures in our program?

Ans g

. **(2 Marks+ 2 Marks)**

The repeated group of instructions in a large program can be written separately from the main program. This subprogram is called as Procedure in an assembly language programming i.e. Procedure is a set of statements that can be processed independently from the main program.

For defining procedure. PROC & ENDP assembler directives are used. The PROC indicates the beginning of the procedure and ENDP directive indicates the end of procedure to the assembler. The procedure must be defined in the codes segment only.

The general format for procedure is

```
Procedure name PROC
        ----------------------------------------------
        Procedure Statements
        ----------------------------------------------
Procedure Name ENDP.
```

2. Attempt any FOUR of the following: (16)

a. What is demultiplexing of the address and data bus? Which signal is used to demultiplex the address and data bus?

Ans a. **(4 Marks)**

The data and address buses are multiplexed in order to save pin count on the chip. In the first clock cycle of a read or write cycle, the address is emitted on the address/data bus. The ALE signal is used to strobe the address, after which the address/data bus becomes the data bus. External logic is expected to strobe the address at the trailing edge of ALE.

ALE is generated directly by the 8085, and by the 8086/8088 in minimum mode. In maximum mode in the 8086/8088, ALE is generated by the 8288 Bus Controller.

- 8088 **address/data bus** lines are multiplexed

- and contain the rightmost 8 bits of the memory address or I/O port number whenever ALE is active (logic 1)
- or data whenever ALE is inactive (logic 0)
  - These pins are at their high-impedance state during a hold acknowledged.
  - 8088 **address bus** provides the upper-half memory address bits that are present throughout a bus cycle.
  - These address connections go to their high-impedance state during a hold acknowledge.

Pin Connections AD_{15} - AD_8

- 8086 **address/data bus** lines compose upper multiplexed address/data bus on the 8086.
- These lines contain address bits A_{15}–A_8 whenever ALE is a logic 1, and data bus connections D_{15}–D_8 when ALE is logic 0.
- These pins enter a high-impedance state when a hold acknowledges occurs.

Pin Connections A_{19}/S_6 - A_{16}/S_3

- **Address/status bus** bits are multiplexed to provide address signals A_{19}–A_{16} and status bits S_6–S_3.

- High- impedance state during hold acknowledges.
- Status bit S_6 is always logic 0.
- Bit S_5 indicates the condition of the IF flag bit.
  - S_4 and S_3 show which segment is accessed during the current bus cycle.
- These status bits can address four separate 1M byte memory banks by decoding as A_{21} and A_{20}.

b. Compare Minimum mode and Maximum mode of 8086 (any four points)

Ans b. **(1 Mark for each)**

Minimum Mode	Maximum Mode
1. $\overline{MN/MX}$ = 1 for minimum mode	1. $\overline{MN/MX}$ = 0 for maximum mode.
2. Single µp in the minimum mode system.	2. Multiprocessor configuration.
3. Bus controller IC 8288 not present.	3. Bus controller IC 8288 is used to generate control signal.
4. Advanced I/O write signal is not there	4. Advanced I/O write command is also there.
5. No status S_0 S_1 S_2 pins	5 Status pins S_0 S_1 S_2 are there.

c. List addressing modes of 8086 with examples.

Ans c.

The way in which the operand is addressed in the instructions is called as addressing mode.

Following is the types of addressing modes of 8086µp.(**1/2 Mark for each)**

i) Immediate addressing mode:

In immediate addressing mode, the 8-bit or 16-bit data required for executing the instruction is present along with the instruction itself.

e.g. a) MOV AL, 47H

After execution of above instruction the 8bit immediate data (47H) is loaded in AL register.

b) MOV CX, 1234H

After execution of above instruction the 16-bit immediate data (1234H) is loaded in CX register.

ii) Direct Addressing mode:

In direct addressing mode, the 8 bit or 16bit data required for executing the instruction is present in the memory location and the 16-bit effective address of the memory location is directly given with in the instruction.

e.g. i) MOV AL, [3000H]

ii) MOV AX, [3000H]

In above examples, the effective address is directly given with the instructions. Hence µp will take the base address from DS (data segment) register.

In particular name of the segment register is specified instruction then µp will take the base address from the segment register.

E.g. MOV BL, ES: [4000H]

iii) Register Addressing Mode:

In register addressing mode, the 8 bit or 16 bit data required for executing the instruction is present in register pair and this register pair is specified in the instruction.

E.g. MOV AL, BL

MOV CX, BX

iv) Register Indirect addressing mode:

In Register Indirect addressing mode, the 8 bit or 16 bit required for executing the instruction is present in the memory location 7 the 16 bit effective address of the memory location is present in the BX/SI/DI register. And the name of these register is specified in the instruction.

E.g. MOV AX, [BX]

MOV DL, CS: [BX]

v) Implicit addressing Mode:

In implicit addressing mode, there is no operand along with the instruction.

E.g. STC, CLD

vi) Register relative addressing mode:

In Register relative addressing mode, the 8 bit or 16 bit data required for executing the instruction is present in the memory location. The registers BX/BP/SI/DI and 8 bits or 16 -bits displacement is given along with the instruction. The effective address is calculated by adding the content of BX, SI, and DI with the relative displacement.

E.g. MOV AX, 50[BX]

MOV CL, 1456[BX]

vii) Base Indexed addressing mode:

In base indexed addressing mode, the 8 bit or 16 bit data required for executing the instruction is present in the memory location.
The base register BX or BP and index register SI or DI is given along with instruction.
The effective address is calculated by adding content of base register and index register with default segment DS and ES.

e.g.: MOV AX, [BP] [SI]
MOV AX, [BX + 04]
ADD AL, [BX] [DI]

viii) Relative base indexed addressing modes:

In Relative base indexed addressing modes, the 8 bit or 16 bit data required for executing the instruction is present in the memory location.
The base register BX or BP and index register SI or DI is given along with instruction.
The effective address is calculated by adding content of base register and index register and the relative displacement.

e.g.: MOV AX, 98H [BX] [SI]
MOV AX, 07H [BP] [DI]
MOV AX, [BX+SI+04]

d. Write an ALP to multiply two 16 bit binary numbers.

Ans d.**[Algorithm -1 +3 Marks]**

Algorithm:

1. Initialise data segment.
2. Load first number
3. Multiply 1st number with 2nd number.
4. Store result.
5. Stop.

ALP:

- Model small
- data

```
num1      dw      0FFFH
num2      dw      0FFFH
res_lsb   dw      0
res_lsb   dw      0
```

- code

```
MOV AX, @data              ; Initialise data segment
MOV DS, AX
MOV AX num1                ; AX ⟵ num1
MUL num2                   ; Multiply num1 by num 2
MOV res_lsb, AX            res_lsb⟵ AX
MOV res_lsb, DX            res_msb ⟵ D
ENDS
END
```

e. What are the difference between NEAR and FAR call (any four points**)**

Ans e.(**1 Marks for Each)**

The CALL instruction is used to transfer execution to a sub program. There are two basic types of CALL:-

NEAR CALL	**FAR CALL**
It is a call to a procedure which is in the same code segment as the CALL instruction.	It is a call to a procedure which is in a different segment from that which contains the CALL segment.
It is a call to a procedure which is in the same segment from that which contains the CALL segment.	It is a call to a procedure which is in a different segment from that which contains the CALL segment.
In NEAR CALL, the stack pointer is decremented by 2 only. The IP is saved on stack and address of procedure is saved into IP.	In FAR CALL, the stack pointer is decremented by 4. Procedure. Then IP is saved on stack and new offset address of procedure is saved onto IP.
CS is not saved on stack.	First CS is saved on stack and CS is loaded with new segment base address of the

f. Why is 8086 memory set up as odd and even memory banks?

Ans f. **(4 Marks)**

The 8086 µp has 20 bit address bus. So it can directly address up to 2^{20} bytes = 1 MB using 20 address lines.

A memory location stores 1 bytes of information and a word (2 bytes) is stored in two consecutive memory locations.

Hence to make it possible to read or write a word in one machine cycle, the memory for 8086 µp is interfaced as two banks of 512 kilo bytes each. One memory bank contains all the bytes which have even address. (00000H, 00002H etc.). The data lines of this bank are connected to the lower address lines $D_0 - D_7$.

The other memory bank contains all the bytes which have odd address (00001H, 00003H etc.). The data lines of this bank are connected to the higher address lines $D_8 - D_{15}$.

3. Attempt any FOUR of the following: (16)

a. Draw the register organization of 8086 and mention the function of each register.

Ans a. **(2 Marks+ 2 Marks)**

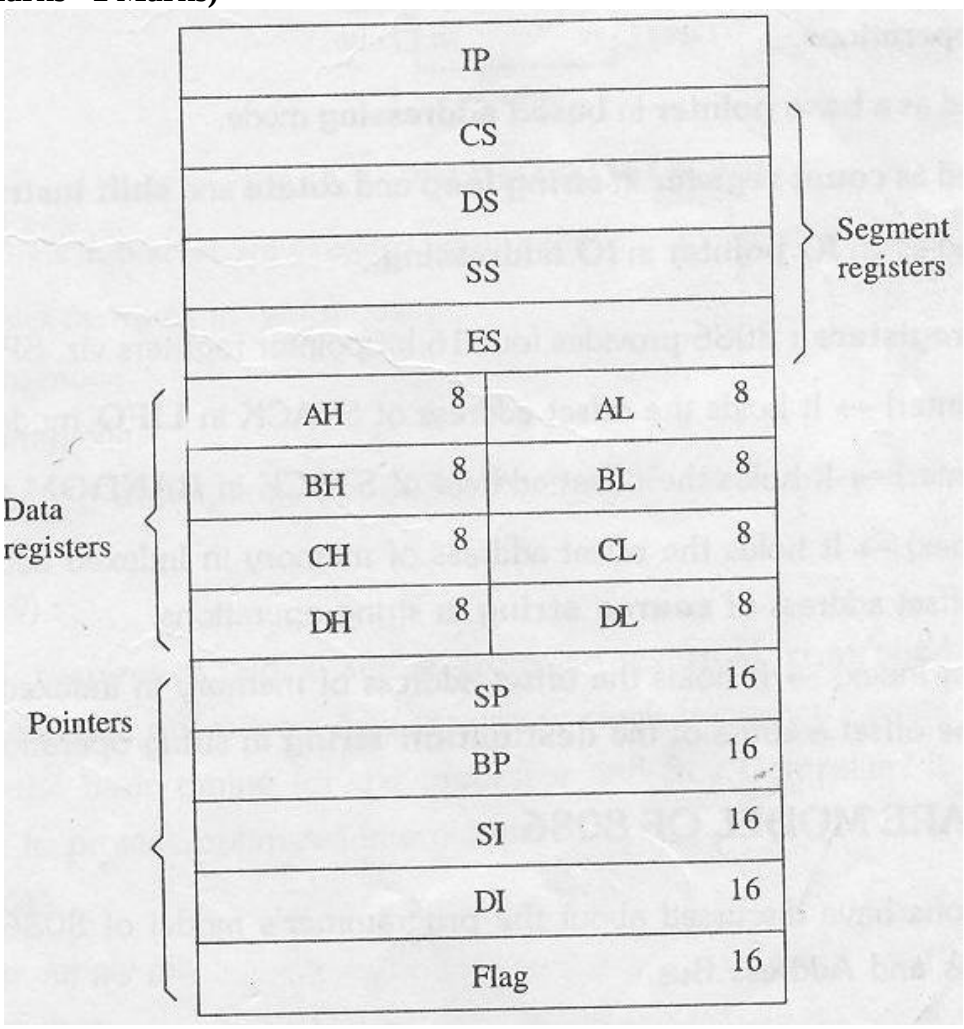

Fig: 8086 register organization

The functions of all the above are as follows:

(1) Instruction pointer: It is never used to hold the data and offset address of data and stack memory.

(2) Segment registers:

CS – It holds the segment address of code or program memory. It is used in conjunction with IP (Instruction pointer) to fetch the next instruction.

DS -- It holds the segment address of data memory by default.

SS – It holds the segment address of stack memory. It is used in conjunction with SP to access stack memory in LIFO mode. It is used in conjunction with BP to access stack memory in random mode.

ES – It holds the segment address of destination string in string operations.

(3) Data registers:

AX, BX, CX and DX registers are used to hold the data operands and results. Each register is divided into two bit registers for 8 bit operations. All data register function as general purpose register. But each register has special function highlighted below

AX -- It is used in IO operations, string operations, multiply and divide operations and rotate and shift operations.

BX – It is used as a base pointer in based addressing mode.

CX – It is used as count register in string loop and rotate and shift instructions.

DX—It is used as an IO pointer in IO addressing.

(4) Pointer registers: 8086 provides four 16 bit pointer registers viz. SP, BP, Si and DI.

SP(Stack Pointer) – It holds the offset address of STACK in LIFO mode.

BP(Base Pointer) —It holds the offset address of STACK in RANDOM mode.

SI (Source Index) —It holds the offset address of memory in Indexed addressing mode. But it also holds the offset address of source string in string operations.
DI(Destination Index) —It holds the offset address of memory in indexed addressing mode but it also holds the offset address of the destination string operations.

(5) Flag Register – A flag is a flip flop which indicates some condition produced by the execution of an instruction or controls certain operations of EU.

b. Distinguished the following instructions (any 2 points)
i) AND and TEST**(2 Marks)**
ii) AAA and DAA**(2 Marks)**

Ans b.
(i)

Sr No.	AND	TEST
1.	The 8/16 bit contents of destination are ANDed with those of source and the result is stored in the destination.	The test instruction ANDs the source and the destination operands but does not store the result nor does it change the operands.
2.	All flags except Auxiliary Flag are affected.	It only affects the flags.

(ii)

Sr No	AAA	DAA
1.	AAA stands for ASCII adjust after additions.	DAA stands for Decimal adjust accumulator.
2.	After the addition of ASCII equivalent of two decimal stored in AL register and carry if any sets the carry flag.	After the addition of two packed BCD numbers, the DAA instruction is used to adjust the hexadecimal result to its BCD equivalent.

c. Write an ALP to transfer a block of 10 data bytes using string instructions.

And c. **(4 Marks)**

BLOCK TRANSFER

```
DATA SEGMENT
        STRNO1   DB    10 DUP(?)          ; Element of string stored here
DATA ENDS
EXTRA SEGMENT
STRNO2   DB   10  DUP(Φ)                  ; Storage for the transferred 10 byte string
EXTRA ENDS
        CODE SEGEMENT
        ASSUME CS: CODE, DS: DATA, ES: EXTRA    ; Initialization of
        MOV  DX, DATA                     ; Code and Data
        MOV  DS,  DX                      ; Segment registers
        MOV DX, EXTRA                     ; initialization of Extra
        MOV ES, DX                        ; Segment register
        LEA SI, STRNO1                    ; Point SI to source string
        LEA DI, STRNO2                    ; Point DI to destination string
        MOV CX, 000AH                     ; Initialize counter register
        CLD                               ; Clear DF to autoincrement Si, DI
REP     MOVSB                             ; Transfer data, Repeat till CX= 0
        MOV AH, 4CH                       ; After CX= 0, terminate the program with
        INT 21H                           ; a return code
CODE ENDS
        END
```

d. Write a procedure to find a factorial of a number.

Ans d. **(4Marks)**

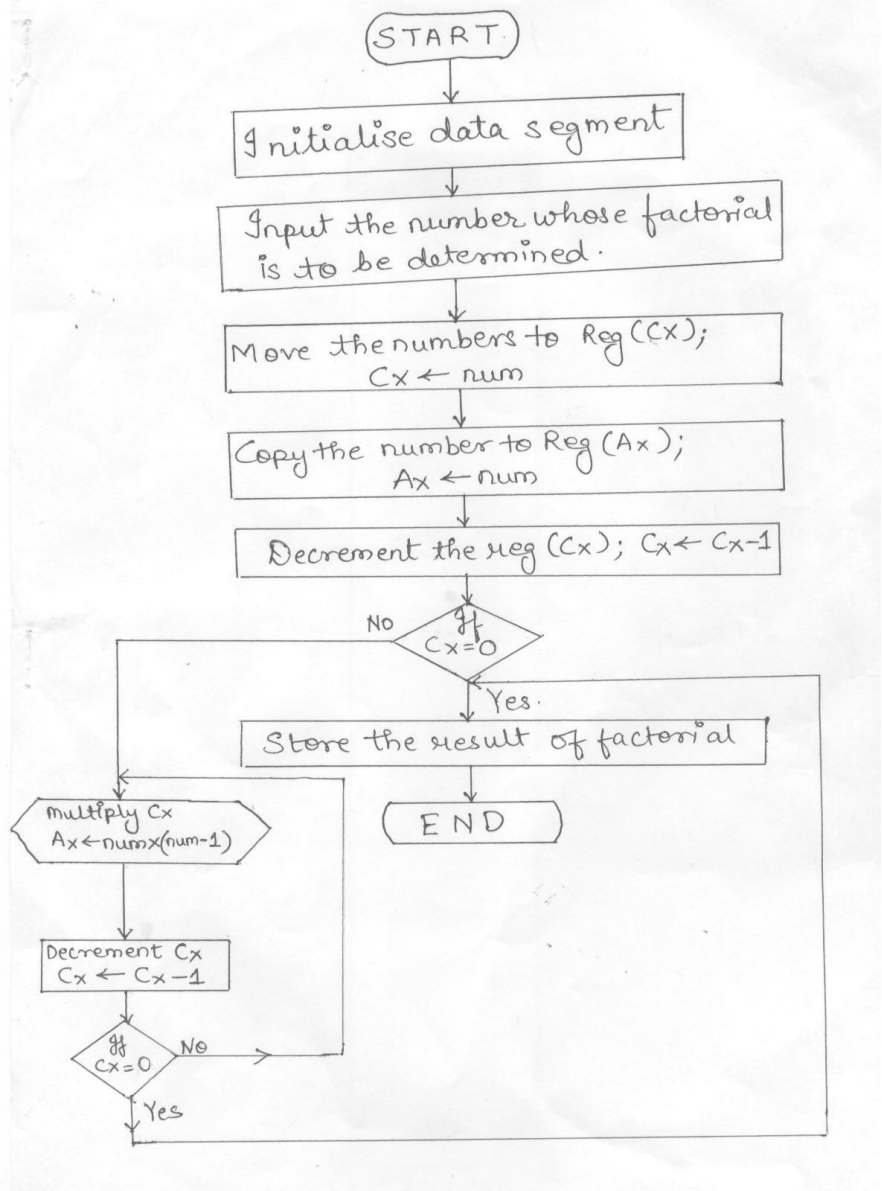

Fig: Flowchart

Procedure:

1. Start
2. Initialise data segment
3. Input the number whose factorial is to be determined.
4. Move the numbers to Register (CX); CX ⟵ Num.
5. Copy the number to Register (AX) ; AX ⟵ Num.
6. Decrement the register (CX); CX ⟵ CX -1-
7. If CX is equal to 0 then Store the result of factorial and end the program.
8. If CX is not equal to 0 then;
9. Multiply CX; AX ⟵ num X (num -1)
10. Decrement the register (CX); CX ⟵ CX -1
11. If CX = 0; then
12. Store the result of factorial and end the program

e. Compare memory Mapped I/o and I/o mapped I/o (any 4 points)

Ans e. **(1 Mark for each)**

Sr no.	I/O mapped I/O	Memory mapped I/O
1.	I/O devices are treated as I/O devices, not memory devices.	I/O devices are treated as memory devices.
2.	Devices addresses are 8 or 16 bits.	Device address is 20 bits.
3.	Separate control signals like IOR and IOW are used to access I/O devices	Control signals like $\overline{\text{MEMR}}$ and $\overline{\text{MEMR}}$ are used to access I/O devices as well as memory devices.
4.	Special I/O instructions like IN and OUT are required to access I/O devices.	All memory instructions are used to access I/O devices.
5.	Data transfer is possible between accumulator and I/O ports only	Data transfer is possible between register and I/O ports.
6.	Address decoder is simple	Address decoder is complicated and expensive.
7.	The 8086 microprocessor can access 1 M memory locations and 64 K I/O ports (additional).	The 8086 microprocessor can access either 1M IO ports or memory locations. The total no. of ports and memory locations should not be greater than 1M.
8.	I/O devices and memory devices are distinguished by control signals.	I/O devices and memory devices are distinguished by address.

4. Attempt any FOUR of the following: (16)

a. What is Queue? How does Queue speeds up the processing of the 8086 microprocessor?

Ans a. **(4 Marks)**

- To speed up program execution, the BIU fetches as many as six instruction bytes ahead of time from memory.
- The prefetched instruction bytes are held for the EU in a first- in – first out group of registers called a queue.
- The BIU can be fetching instructions bytes while the EU is decoding an instruction or executing an instruction which does not require use of the buses.
- When the EU is ready for its next instruction, it simply reads the instruction from the queue in the BIU. This is much faster than sending out an address to the system memory and waiting for memory to send back the next instruction byte or bytes.
- The process is analogous to the way a bricklayer's assistant fetches bricks ahead of time and keeps a queue of bricks lined up so that the bricklayer can just reach out and grab a brick when necessary.
- Except in the cases of JUMP and CALL instructions where the queue must be dumped and then reloaded starting from the address, this prefetch- and- queue scheme greatly speeds up processing.
- Fetching the next instruction while current instruction executes is called pipelining.

b. Explain the assembler directives DD, DUP,DW and ENDS.

And b.

DD- DEFINE DOUBLEWORD (1 Mark)

- The DD directive is used to declare a variable of type doubleword or to reserve memory locations which can be accessed as type doubleword.
- The statement ARRAY_POINTER DD 25629261H, for example, will define a doubleword named ARRAY_POINTER and initialize the doubleword with the specified

value when the program is loaded into memory to be run. The low word, 9261H, will be put in memory at a lower address than the high word.

- A declaration of this type is often used with the LES or LDS instruction. The instruction LES DI, ARRAY _POINTER, for example, will copy the low word of this doubleword,9261H, into the DI register and the high word of the doubleword, 2562H, into the extra segment register.

DUP- DUPLICATE (1 Mark)

Just as its name implies, DUP duplicates text.

Straight Example:

Text DB 10 DUP ('W') ; initializes 20 bytes to W

The number after **DB** defines how many bytes to repeat for, and then the **'W'** defines what to repeat.

DW- DEFINE WORD. (1 Mark)

- The DW directive is used to tell the assembler to define a variable of type word or to reserve storage locations of type word in memory.
- The statement MULTIPLIER DW 437AH, for example, declares a variable of type word named MULTIPLIER.
- The statement also tells the assembler that the variable MULTIPLKIER should be initialized with the value 437 AH when the program is loaded into memory to run.
- Here are a few more examples of DW statements.

THREE_LITTLE_WORDS DW 1234H, 3456H, 5678H
; Declare array of 3 words and initialize with specified values.

STORAGE DW 100 DUP (0)
; Reserve an array of 100 words of memory and initialize all 100 words with 0000. Array is named STORAGE.

STORAGE DW 100 DUP (?)
; Reserve 100 words of storage in memory and give it the name STORAGE, but leave the words uninitialized.

ENDS- END SEGMENT (1 Mark)

- This directive is used with the name of a segment to indicate the end of that logical segment to indicate the end of that logical segment. ENDS is used with the SEGMENT directive to "bracket" a logical segment containing instructions or data. Here's an example.

```
CODE SEGMENT        ; Start of logical segment
                    ; containing code
                    ; Instruction statements
CODE ENDS           ; End of segment named
                    ; code
```

c. Write an ALP for 8086 to divide two 16 bit unsigned number.

Ans c. **(4 Marks)**

Algorithm:

1. Initialise data segment.
2. Load first number.
3. Divide first number with second number.
4. Store quotient and reminder.
5. Stop.

Assembly language program:

- Model small - Select programming model1
- data

```
dividend    dw    1234H
divisor     dw    0012H
quotient    dw    0
remainder   dw    0
```

- code

```
MOV AX, @ data      - Initialise data segment
MOV DS, AX          - DS ⟵ AX
MOV AX, dividend    - Load dividend in AX
CWD                 - Covert word to double
                      Word with sign bit
                      DX = 0000H
                      AX = 1234H
DIV divisor         - Divide double word by word
MOV quotient, AX    - Store quotient
MOV remainder, DX   - Store remainder
ENDS                - Ends a segment
END                 - End of program file
```

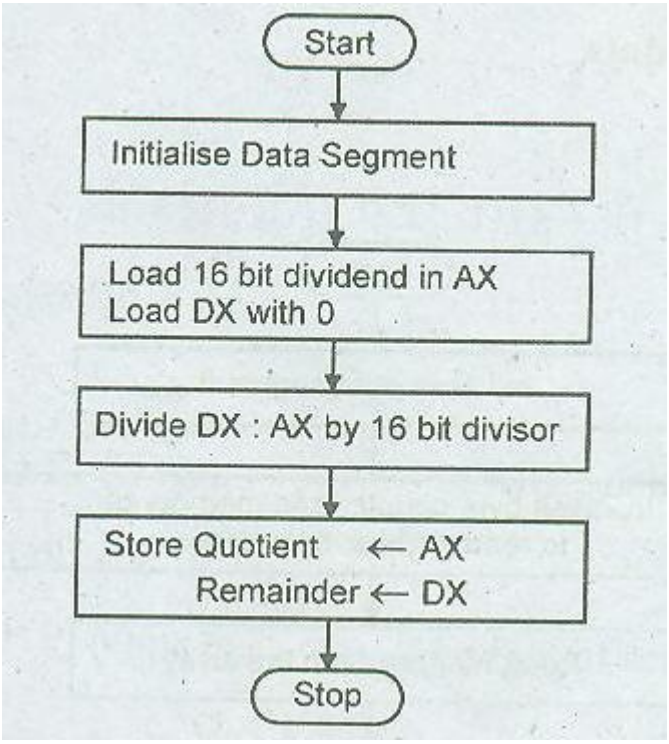

Fig: Flowchart

d. Explain with suitable example how a parameter is passed in register in 8086 assembly language procedure.

Ans d. **(4 Marks)**

- The BCD number is passed to the procedure in the AL register and the hex equivalent is passed back to the calling program in the AL register. We start the procedure by pushing the flag register and the other registers we use in the procedure.
- Notice that we don't need o push and pop the AX register because we are using it to pass a value to the procedure and expecting the procedure to pass a different value back to the calling program in it.
- Hopefully the function of the rest of the instructions in the procedure is reasonably clear from the comments with them. WE first make a copy of the BCD in AL so we have two copies to work on. We then mask the upper nibble of one and save it in BL. Since multiplying this nibble by one would not change its value, we are done with it for now.
- We mask the lower nibble of the other copy of the BCD and rotate this nibble into the lower nibble position of the byte so we can multiply it correctly.
- When we multiply this nibble by the digit weight of 0AH, the result is left in the AX register. However, since the result can never be greater than 8bits, we can disregard the contents of AH.
- Finally, we add the lower nibble we saved in BL to the result in AL to get the hex total. The desired result is left in AL. Before returning to the main program we pop the registers we pushed at the start of the procedure.

e. Compare Procedure and Macro.

Ans e. **(1 Mark for each)**

Sr no.	**Procedure**	**Macro**
1.	Machine Codes for the group of instructions in the procedure have to be put in the memory only once.	The assembler puts the group of instructions defined as macro, each time the macro name appears in a program.
2.	The program takes up less memory compared to equivalent memory required for the program using procedures.	The program takes up more memory compared to equivalent memory required for the program using procedures.
3.	Each time a procedure is called in a program, the current IP contents are saved on the stack and the instruction pointer is loaded with the starting address of the procedure. Returning back from the procedure involves popping back the address and returning to the original program. Hence execution time increases.	In case of Macros, the generated machine codes are right in line with the rest of the program, the processor does not have to go off to a procedure and return.
4.	Example: Procedure Name PROC ---------- Procedure Statements ---------- Procedure Name ENDP.	Example: Macro_name MACRO ------ ------ } instructions ------ ENDM

f. Draw a neat diagram to show interfacing of two 8k X 8 RAM chips using memory mapping as an addresses device.

Ans f. **(4 Marks)**

Here size of RAM =8K

Therefore, 13 address lines are required to address memory location within both RAM [2^{13} = 8192 =8K]

Hence A_1 to A_{13} address lines can be connected to A_0 to A_{12} address pins of both RAMs & A_0 & remaining lines A_{14} –A_{19} can be used to generate chip select signals for both RAM.

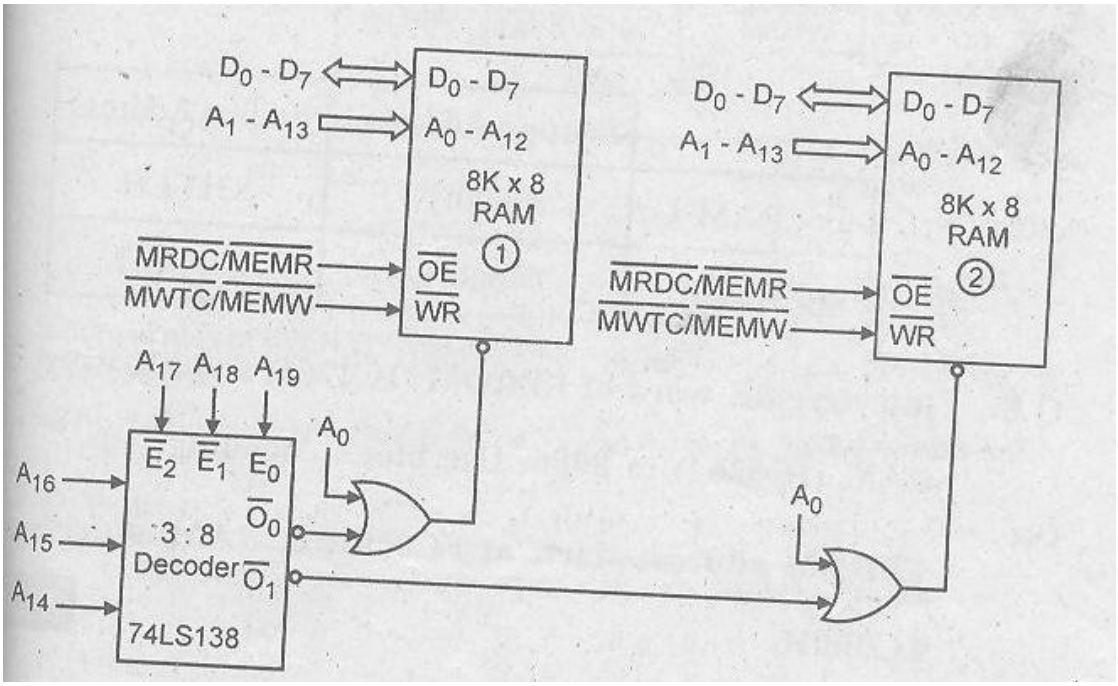

Fig: Interfacing of two 8K RAM as an EVEN addresses device.

In minimum mode, $\overline{MEMR}$ & $\overline{MEMW}$ are connected to $\overline{OE}$ & $\overline{WR}$ pins of RAMs & in maximum mode $\overline{MRDC}$ & $\overline{MWTC}$ pins are connected.

The address maps of RAM 1 & RAM 2.

A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	Address
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80000H
1	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	83FFEH
						* RAM 1														
1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	84000H
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	87FFEH
						* RAM 2														

The $\overline{O_0}$ & $\overline{O_1}$ o/p pin of decoder are used to generate chip select signal along with A_0 for both RAM. Hence A_{14} A_{15} A_{16} = 0 0 0 to select $\overline{O_0}$ o/p for RAM 1 & A_{14} A_{15} A_{16} = 1 0 0 to select $\overline{O_1}$ o/p for RAM 2.

	Starting Address	**Ending Address**
For RAM 1	80000H	83FFEH
RAM2	84000H	87FFEH

5) **Attempt any <u>TWO</u> of the following:** **(16)**

a. What is memory segmentation? How memory segmentation is achieved in 8086µp? State any four advantages of memory segmentation.

Ans a. **(4 Marks)**

In memory segmentation scheme, the complete physical memory is divided into a number of logical segments.

In 8086µp has twenty address pins. Hence each memory location connected with the µp has 20-bit address. The 20- bit address of a memory is called a Physical address. Therefore accessing any memory location, the µp has to transfer 20 bit physical address of that memory location on the address lines.

The memory connected with µp is divided into four segments and each segment is 64k bytes and addressed by one of the segment registers.

The 20-bit starting memory location physical address of a memory segment is called as Base address. The 4LSBs of the base address should always be zero. Rest of the 16MSBs of base address of memory segment should be transferred to the corresponding segments register i.e. to

Code memory segment
Data memory segment
Extra memory segment
Stack memory segment

Advantages of segmentation: (4 Marks)

i) It allows the memory of 1 MB even though the address associated with individual instruction is only 16-bit.
ii) Provides facility for using separate memory areas for program, data & stack.
iii) If allows more than 64k bytes of code, data, stack &extra code memory by using more than one code data, stack and extra code memory.
iv) Permits a program or its data to be put in different areas of memory, each time the program is executed.
i.e. the program can be relocated which is very useful in multiprogramming.

b. With examples, describe the string instruction in 8086 assembly language.

Ans b. **(Any 4 instruction -2M each [1M –Syntax, 1M Description])**

String is a collection of data stored in consecutive memory locations. The various string instructions in 8086 are:

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.

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.

c. Describe the function of following ALP tools
   i) Editor **(2 Marks)**
   ii) Assembler **(2 Marks)**
   iii) Linker **(2 Marks)**
   iv) Debugger **(2 Marks)**

Ans c.

Editor:

It is a program which is used to construct assembly language program in required format so that the assembler easily convert it into machine language.

Assembler:

It is a program which is used to convert a program written in assembly language into machine language. Hence the Assembler is required while using Assembly language.

Linker:

It is a program which is used to combine more than one separately assembled module into one executable program.

Debugger:

Debugger is a program which is used to write, execute and debug assembly language programs and to examine the contents of register and memory.

It can also be used to run a program, one instruction at a time.

E.g. DEBUG command in DOS.

6. Attempt any <u>TWO</u> of the following: (16)

a. Describe how 20 bit physical address is generated in 8086 microprocessor. Calculate the physical address generated by
   i) 4370 : 561E **(4 marks)**
   ii) 7A32: 0028 **(4 marks)**

Ans a.

i))

```
    4  3  7  0  0
+      5  6  1  E
-------------------
    4  8  D  1  E   -------- Physical address.
```

ii))

```
    7  A  3  2  0
+      0  0  2  8
-------------------
    7  A  3  4  8   -------- Physical address.
```

b. Name the different types of jump instructions used in 8086 assembly language programs. Give any two difference between intersegment and intrasegment types of jump. Describe each with example.

Ans b.

There are two types of branching/ jump instruction.

i) Unconditional jump. (2 marks)
ii) Conditional jump. (2 marks)

i] **Unconditional Jump:**

This instruction will always cause the 8086 to fetch its next instruction from the location specified in the instruction rather than from next location after jump instruction.

There are two types of jump.

a. Near/ intrasegment jump.
b. Far/ intersegment jump.

ii] **Conditional Jump:**

This instruction transfers the control to the specified location after condition is satisfied. If condition is not satisfied the next instruction is executed (normal flow).

i) Different types of conditional instructions are: (2 marks)

Sr no	Instruction code	Description	Condition for Jump
1	JA/JNBE	Jump if above /jump if not below nor equal	CF=0 ZF=0
2	JAE/JNB	Jump if above or equal/jump if not below	CF=0 ZF=1
3	JB/JNAE	Jump if below/jump if not above nor equal	CF=1 ZF=0
4	JBE/JNA	Jump if below or equal/jump if not above	CF=1 ZF=1
5	JC	Jump if carry	CF=1
6	JE/JZ	Jump if equal/jump if zero	ZF=1
7	JG/JNLE	Jump if greater/jump if not less than not equal	ZF=0 CF=0
8	JGE/JNL	Jump if greater than or equal/jump if not less than	SF=0
9	JL/JNGE	Jump if less than / jump if not greater than nor equal	SF≠0
10	JL/JNGE	Jump if less than or equal/jump if not greater than	ZF=1 SF≠0
11	JNC	Jump if no carry	CF=0
12	JNE/JNZ	Jump if not equal/Jump if not zero	ZF=0
13	JNO	Jump if no overflow	OF=0
14	JNP/JPO	Jump if not parity/ jump if parity odd	PF=0
15	JNS	Jump if not sign	SF=0
16	JO	Jump if overflow	OF=1
17	JP/JPE	Jump if parity/Jump if parity even	PF=1
18	JS	Jump if sign	SF=1

Difference between intrasegment &intersegment jump: (2 marks)

- An intrasegment jump/ Near jump is a jump where destination location is in the same code segment.
  e.g. JMP NEAR

- An intersegment jump/ far jump is a jump where destination location is from different segment.
  e.g. JMP WORLD PTR [BX]

c. Write an ALP to arrange 16 number (16- bit) in ascending order. Draw flow chart and write result.

Ans c. **(6 Marks+ 2 Marks)**

Assume length of numbers is 'N'

Algorithm:

1. Initialise data segment
2. Initialise iteration counter (N)
3. Initialise memory pointer to read number from array.
4. Initialise comparison counter (N-1)
5. Read numbers from the array.
6. Compare two numbers.
7. If number ≤ next number then go to step 9.
8. Interchange numbers.
9. Increment memory pointer to read next number from array.
10. Decrement comparison counter and go to step 5 until becomes zero.
11. Decrement iteration counter and go to step 3 until becomes zero.
12. Stop.

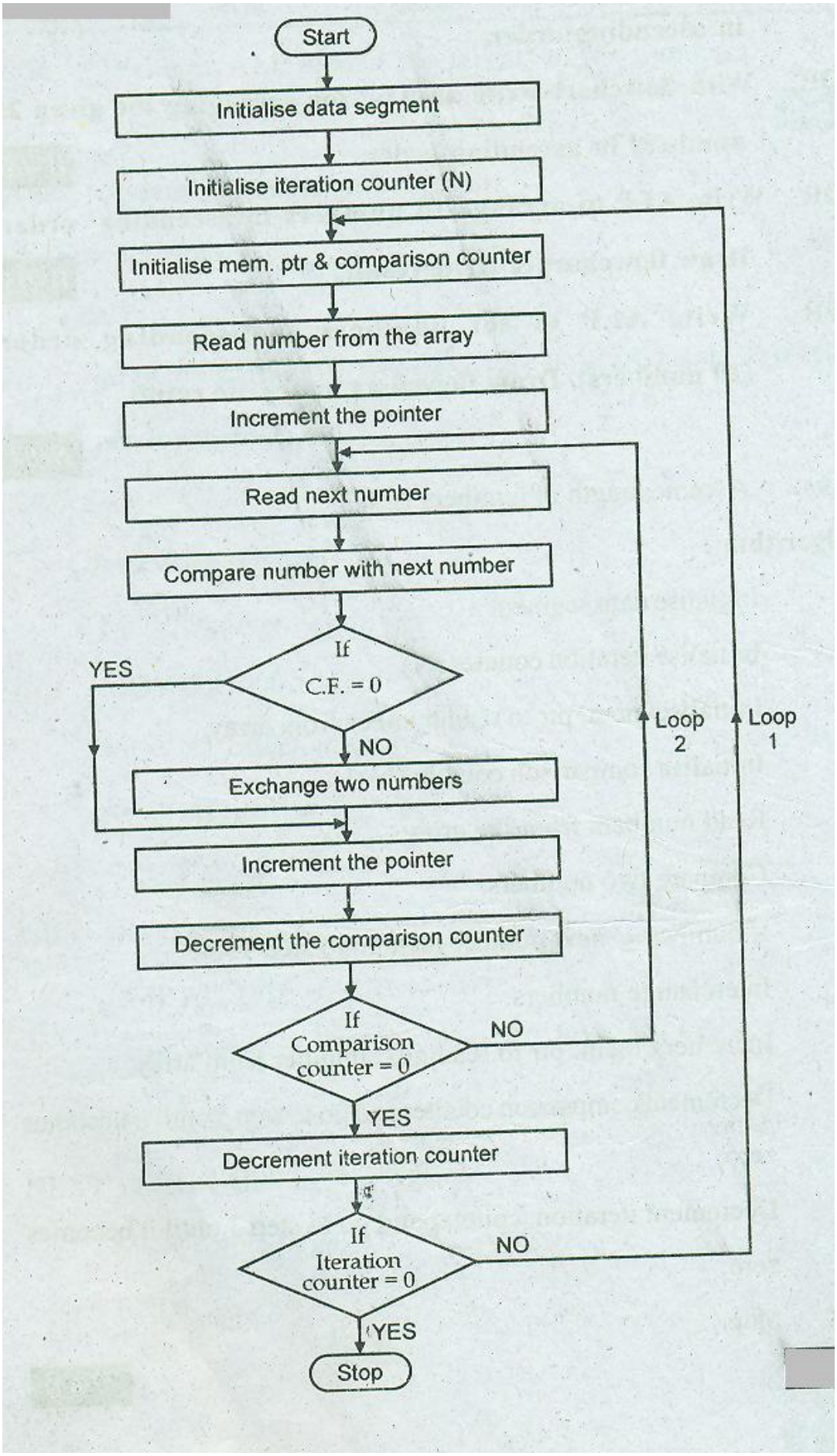

Fig: Flowchart

Assume N = 05

Numbers are 8 bit

Assembly language Program (ALP):

- Model small
- data
  array dw 10H, 11H, 13H, 12H, 09H
- code
  MOVE AX, @data
  MOVE DS, AX
  MOV BX, 0005H

```
Loop1: MOV SI, offset array
       MOV CX, 0004H
Loop2: MOV AL, [SI]
       CMP AL, [SI+1]
       JC: DN
       XCHG AL, [SI+ 1]
       XCHG AL, [SI]
 DN  : INC SI
       LOOP: Loop2          ; if comparison counter ≠ 0, Go to Loop2
       DEC BX
       JNZ: Loop1           ; if comparison counter ≠ 0, Go to Loop 1
       ENDS
       END
```