S.Y. Diploma : Sem. IV
[CO/CM/CW]

Microprocessor

Time: 3 Hrs.] Prelim Question Paper Solution **[Marks : 70**

Q.1 Attempt any FIVE of the following : [10]

Q.1(a) State the function of ALE and Ready pin of 8086. [2]

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

READY: It is available at pin 32. It is an acknowledgement signal from I/O devices that data is transferred. It is an active high signal. When it is high, it indicates that the device is ready to transfer data. When it is low, it indicates wait state.

Q.1(b) What is stack? State its significance. [2]

Ans.: The stack is a block of memory that may be used for temporarily storing the contents of the registers inside the CPU. It is a top-down data structure whose elements are accessed using the stack pointer (SP) which gets decremented by two as we store a data word into the stack and gets incremented by two as we retrieve a data word from the stack back to the CPU register.

Q.1(c) What is the use of REP in string related instruction? [2]

Ans.: Rep is a prefix written before the instruction. It stands for repeat the current string instruction CX number of times.

Q.1(d) State the function of following pins of 8085 microprocessor : [2]

(i) INTR (ii) $\overline{\text{INTA}}$

Ans.: **(i) INTR:**

It is level triggered, non-vectored interrupt. When INTR occurs the microprocessor generates interrupt acknowledgement signal $\overline{\text{INTA}}$

(ii) $\overline{\text{INTA}}$:

It is an active low acknowledgement signal for INTR.
This signal is used to get OPCODE & hence ISR address from external hardware.

Q.1(e) List any four features of 8086 microprocessor. [2]

Ans.:

1. It is a 16 bit μp.
2. 8086 has a 20 bit address bus can access up to 2^{20} memory locations (1MB).
3. It can support up to 64K I/O ports.
4. It provides 16-bit registers. AX, BX, CX, DX, CS, SS, DS, ES, BP, SP, SI, DI, IP & FLAG REGISTER.
5. It has multiplexed address and data bus AD_0-AD_{15} and A_{16} – A_{19}.
6. 8086 is designed to operate in two modes, Minimum and Maximum.
7. It can prefetches up to 6 instruction bytes from memory and queues them in order to speed up instruction execution.
8. Interrupts: 8086 has 256 vectored interrupts.
9. Provides separate instructions for string manipulation.
10. Operating clock frequencies 5MHz, 8MHz, 10MHz.

Q.1(f) Give the steps in physical address generation in 8086 microprocessor. [2]

Ans.: **Generation of 20 bit physical address in 8086:**

1. Segment registers carry 16 bit data, which is also known as base address.
2. BIU appends four 0 bits to LSB of the base address. This address becomes 20-bit address.
3. Any base/pointer or index register carries 16 bit offset.
4. Offset address is added into 20-bit base address which finally forms 20 bit physical address of memory location.

Q.1(g) State the function of following pins of 8085 microprocessor. [2]

(i) Ready (ii) Trap

Ans.: (i) Ready:

- It is an active high signal used to synchronize μp with slower peripherals.
- μp samples Ready input in the beginning of every Machine cycle.
- If found low, μp executes WAIT CYCLE, after which it resamples READY pin till it finds Ready pin HIGH. Therefore, μp remains in the WAIT STATE until the READY pin becomes high again.

(ii) Trap:

- It is an edge as well as level triggered, highest priority, non-maskable vectored interrupt.
- This interrupt transfers the microprocessor's control to location 0024H

Q.1(h) What is pipelining? How it is implemented in 8086 microprocessor. [2]

Ans.: Definition:

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

Description:

- In 8086, pipelining is the technique of overlapping instruction fetch and execution mechanism.
- To speed up program execution, the BIU fetches as many as six instruction bytes ahead of time from
- memory. The size of instruction prefetch queue in 8086 is 6 bytes.
- While executing one instruction other instruction can be fetched. Thus it avoids the waiting time for execution unit to receive other instruction.
- BIU stores the fetched instructions in a 6 byte FIFO 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.
- This improves overall speed of the processor.

Q.1(i) State any two differences between NEAR and FAR procedure. [2]

Ans.:

Sr. No.	Near Procedure	Far Procedure
(i)	A near procedure refers to a procedure which is in the same code segment from that of the call instruction.	A far procedure refers to a procedure which is in the different code segment from that of the call instruction.
(ii)	It is also called intra-segment procedure.	It is also called inter-segment procedure call.
(iii)	A near procedure call replaces the old IP with new IP.	A far procedure call replaces the old CS:IP pairs with new CS:IP pairs.
(iv)	The value of old IP is pushed on to the stack. SP = SP-2; Save IP on stack (address of procedure)	The value of the old CS:IP pairs are pushed on to the stack SP = SP-2; Save CS on stack SP = SP-2; Save IP (new offset address of called procedure)
(v)	Less stack locations are required	More stack locations are required
(vi)	Example: Call Delay	Example: Call FAR PTR Delay

Q.2 Attempt any THREE of the following : [12]

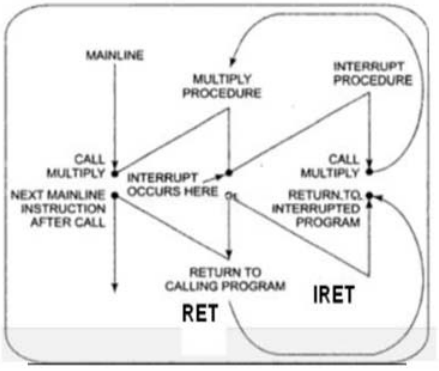
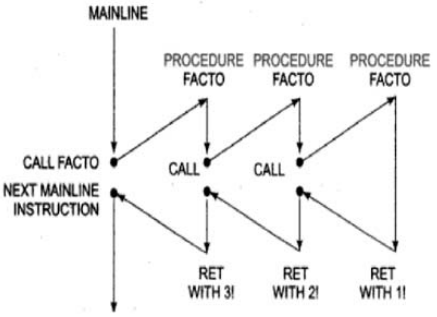
Q.2(a) Give the difference between Inter segment and Intra segment CALL. [4]

Ans.:

Sr. No.	Inter segment CALL	Intra segment CALL
(i)	A near call is a call to procedure which is in same code segment	A far call is a call to procedure which is in different segment
(ii)	The contents of CS is not stored	The contents of CS is also stored along with offset.
(iii)	In near call contents of SP is decremented by 2 and contents of offset address IP is stored	In Far call contents of SP are decremented by 2 and value of CS is loaded. Then SP is again decremented by 2 and IP is loaded.
(iv)	Example : CALL Delay	Example : CALL FAR PTR Delay

Q.2(b) Differentiate between Re-entrant & Recursive procedure. [4]

Ans.:

Sr. No.	Re-entrant procedure	Recursive procedure
(i)	A procedure is said to be re-entrant, if it can be interrupted, used and re-entered without losing or writing over anything.	A recursive procedure is a procedure which calls itself.
(ii)	In Re-entrant Procedure must first push all the flags and registers used in the procedure.	In recursive procedure the program sets aside a few locations in stack for the storage of the parameters which are passed each time the computation is done and the value is returned.
(iii)	To be a re-entrant, It should also use only registers or stack to pass parameters.	In recursive procedure Each value returned is then obtained by popping back from the stack at every RET instruction when executed at the end of the procedure.
(iv)	Example 	Example 

Q.2(c) Explain the following assembler directives. [4]

(i) ORG (ii) EQU (iii) DD (iv) ASSUME

Ans.: (i) 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

Example: ORG 2000H ; set location counter to 2000H
ORG $+100 ; increment value of location counter by 100 from its current.

(ii) 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

Example: CORRECTION_FACTOR EQU 100

(iii) DD: Define Double word (32-bits)

It is used to declare a variable of type double word or to reserve memory locations which can be accessed as type double word(32-bits)

Example: NUMBER DD 12345678H ; Reserve 2 words in memory.

(iv) ASSUME: Assume directive is used to tell Assembler the name of the logical segment it should use for the specified segment.

Example: Assume CS: MAP_CODE, DS: MAP_DATA

Q.2(d) What is MACRO? Explain MACRO with suitable example. [4]

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 definition or (Macro directive):

Syntax:

```
MACRO_NAME MACRO[ARG1,ARG2,
…..ARGN) …..
ENDM
```

Example: (NOTE: Any Same Type of Example can be considered)

```
MYMACRO MACRO P1, P2, P3 ; Macro definition with arguments
MOV AX, P1
MOV BX, P2
MOV CX, P3
ENDM                    ; Indicates end of macro.
DATA SEGMENT
DATA ENDS
CODE SEGMENT
START: ASSUME CS:CODE, DS:DATA
MOV AX, DATA
MOV DS, AX
MYMACRO 1, 2, 3         ; macro call
MYMACRO 4, 5, DX
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

Q.3 Attempt any THREE of the following : [12]

Q.3(a) Describe Memory segmentation in 8086 and list its advantages. [4]

Ans.: Memory Segmentation

The memory in 8086 based system is organized as segmented memory.8086 can access 1M byte memory which is divided into number of logical segments. Each segment is 64KB in size and addressed by one of the segment register. The 4 segment register in BIU hold the 16-

bit starting address of 4 segments. CS holds program instruction code. Stack segment stores interrupt & subroutine address. Data segment stores data for program. Extra segment is used for string data.

Advantages of segmentation

(i) With the use of segmentation the instruction and data is never overlapped.
(ii) The major advantage of segmentation is Dynamic relocatability of program which means that a program can easily be transferred from one code memory segment to another code memory segment without changing the effective address.
(iii) Segmentation can be used in multi-user time shared system.
(iv) Segmentation allows two processes to share data.
(v) Segmentation allows you to extend the addressability of a processor i.e., address up to 1MB although the actual addresses to be handled are of 16 bit size.
(vi) Programs and data can be stored separately from each other in segmentation.

Q.3(b) Write an ALP to perform 32 bit by 16-bit division of unsigned numbers. [4]

Ans.:

```
DATA SEGMENT
NUMBER1 DD 00002222H
NUMBER2 DB 1111H
Quotient DW  DUP(0)
Remainder DW  DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START:MOV DX , DATA
MOV DS ,DX
MOV AX ,NUMBER1
MOV BX ,NUMBER2
DIV BX ; Ans AX :Quotient ,DX :Remainder
MOV Quotient, AX
MOV Remainder, DX
MOV AH , 4CH
INT 21H
CODE ENDS
END START
```

Q.3(c) List and explain any four process control instruction with their function. [4]

Ans.: List of process control instruction:

1. CLC – Clear carry flag
2. CMC – Complement carry flag
3. STC – Set carry flag
4. CLD – Clear direction flag
5. STD – Set direction flag
6. CLI – Clear interrupt flag
7. STI – Set Interrupt flag
8. WAIT – waif for test input pin to go low
9. HLT – Halt the processor
10. NOP – No operation
11. ESC – Escape to external device like NDP (numeric co-processor)
12. LOCK – Bus lock instruction prefix.

Explanation:

1. CLC – This instruction Clear Carry Flag. CF ← 0
2. CMC – This instruction Complement Carry Flag. CF ←~CF
3. STC – This instruction Set Carry Flag. CF ← 1
4. CLD – This instruction Clear Direction Flag. DF ← 0

5. STD – This instruction Set Direction Flag. DF ← 1
6. CLI – This instruction Clear Interrupt Flag. IF ← 0
7. STI – This instruction Set Interrupt Flag. IF ← 1
8. HLT
   - This instruction causes processor to enter the halt state.
   - CPU stop fetching and executing instructions.
9. NOP
   - Used to add wait state of three clock cycles and during these clock cycles CPU does not perform any operation.
   - This instruction is Used to add delay loop in program
10. WAIT
    - It causes processor to enter into an idle state or a wait state and continue to remain in that the processor receives state until one of the following signal.
      - Signal on processor TEST pin
      - Valid interrupt on INTR
      - Valid interrupt on NMI
    - Used to synchronize other external hardware such as math co-processor.
11. LOCK
    - Prevent other processor to take the control of shared resources.
    - Lock the bus attached to lock pin of device while a multicycle instruction completes.
    - The lock prefix this allows a microprocessor to make sure that another processor does not take control of system bus while it is in the middle of a critical instruction.
12. ESC:
    - This instruction is used to pass instructions to a coprocessor, such as the 8087 Math coprocessor, which shares the address and data bus with 8086. Instructions for the coprocessor are represented by a 6-bit code embedded in the ESC instruction.

Q.3(d) State all the control signal generated by S_0, S_1, S_2 with their function by 8086 microprocessor. [4]

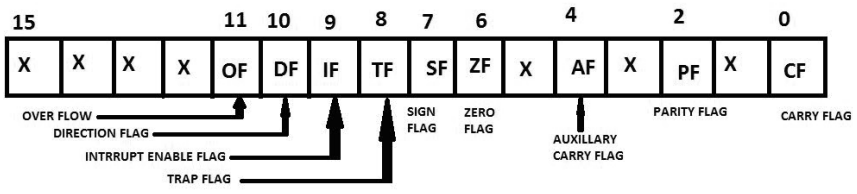
Ans.:

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	**Processor state**
0	0	0	Interrupt Acknowledge
0	0	1	Read I/O Port
0	1	0	Write I/O Port
0	1	1	Halt
1	0	0	Code Access
1	0	1	Read Memory
1	1	0	Write Memory
1	1	1	Passive

Q.4 Attempt any THREE of the following : [12]

Q.4(a) Draw and explain the flag register of 8086. [4]

Ans.:



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.

Q.4(b) Write an ALP to find the smallest number in the Array. [4]

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]
JC NEXT
MOV AL,[SI]
NEXT: DEC CX
JNZ UP
MOV SMALLEST,AL
MOV AX, 4C00H
INT 21H
CODE ENDS
END START
```

Q.4(c) Explain the following instruction of 8086 : [4]

(i) XLAT (ii) XCHG

Ans.: (i) XLAT:

- XLAT replaces a byte in AL register with a byte from 256 byte lookup table beginning at [BX].
- AL is used as offset into this table.
- Flags are not affected
- operation: AL ← [BX + AL]

Example:

```
data segment
```

```
    Table db '0123456789ABCDEF'
    CODE DB 11
data ends
Code segment
    _ _ _
    MOV BX, offset Table
    MOV al, CODE
    XLAT; AL will output code 0BH
Code ends
```

(ii) XCHG Destination, Source

- This instruction exchanges the contents of a register with the contents of another register or memory location.
- **Operation performed:**
  Destination ↔ Source
  None of flag affected
  Example:
  XCHG BL, CL; Exchange the byte in BL with byte in CL.

Q.4(d) Write an ALP to count number of zero's in BL register. [4]

Ans.:

```
DATA SEGMENT
    NUM DB 0F3H          ;BINARY{1111 0011}
    ZEROS DB 0
DATA ENDS
CODE SEGMENT
START: ASSUME CS: CODE, DS:DATA
    MOV AX, DATA
    MOV DS, AX

    MOV CX, 8      ;rotation counter
    MOV BL, NUM
UP:
    ROR BL, 1      ; RCR, ROL, RCL can be used
    JC DN          ; IF CARRY llop
    INC ZEROS      ; else increment 0's count ; ANSWER 02
DN: LOOP UP        ; decrement rotation counter
EXIT: MOV AH, 4CH
    INT 21H
CODE ENDS
END START
```

Q.4(e) Write an assembly language program to find largest number from array of 10 numbers. [4]

Ans.:

```
DATA SEGMENT
    ARRAY DB 15H, 45H, 08H, 78H, 56H, 02H, 04H, 12H, 23H, 09H

    LARGEST DB 00H
DATA ENDS
CODE SEGMENT
START:ASSUME CS:CODE,DS:DATA
    MOV DX, DATA
    MOV DS, DX

    MOV CX, 09H
    MOV SI, OFFSET ARRAY
    MOV AL, [SI]
```

```
UP:INC SI
CMP AL, [SI]
JNC NEXT                ;CHANGE
MOV AL, [SI]
NEXT:DEC CX
JNZ UP
MOV LARGEST, AL ; AL=78h
MOV AX, 4C00H
INT 21H
CODE ENDS
END START
```

Q.5 Attempt any TWO of the following : [12]

Q.5(a) Describe how an assembly language program is developed and debugged using system tools such as editors, assemblers, linkers and debuggers. [6]

Ans.: **(i) Defining the problem:** The first step in writing program is to think very carefully about the problem that the program must solve.

(ii) 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.

(iii) Flowchart: The flowchart is a graphically representation of the program operation or task.

(iv) Initialization checklist: Initialization task is to make the checklist of entire variables, constants, all the registers, flags and programmable ports

(v) Choosing instructions: Choose those instructions that make program smaller in size and more importantly efficient in execution.

(vi) 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.

Tools:

(i) Editor

- 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.
- It enables one to create, edit, save, copy and make modification in source file.

(ii) Assembler

- Assembler is a program that translates assembly language program to the correct binary code.
- It also generates the file called as object file with extension .obj.
- It also displays syntax errors in the program, if any.
- It can be also be used to produce list(.lst) and .crf files

(iii) Linker

- It is a programming tool used to convert Object code into executable program.
- It combines, if requested, more than one separated assembled modules into one executable module such as two or more assembly programs or an assembly language with C program.
- It generates .EXE module.

(iv) Debugger

- Debugger is a program that allows the execution of program in single step mode under the control of the user.
- The errors in program can be located and corrected using a debugger.

Q.5(b) Write an ALP to compute, whether the number in BL register is even or odd. [6]

Ans.:

```
DATA SEGMENT
    NUM DB 9H
    ODD DB 0
    EVEN_NO DB 0
DATA ENDS
CODE SEGMENT
START: ASSUME CS: CODE, DS:DATA
        MOV AX, DATA
        MOV DS, AX
    MOV BL, NUM
    ROR BL, 1       ; or RCR
    JNC DN          ; check ENEN or ODD
    ROL BL, 1       ; restore number
    MOV ODD, BL  ; odd
    JMP EXIT
DN: ROL BL, 1
    MOV EVEN_NO, BL     ; even no

EXIT: MOV AH, 4CH
        INT 21H
CODE ENDS
END START
```

Q.5(c) Explain the directives used for defining MACRO. Give an example. [6]

Ans.:

1. **Macro definition or (Macro directive):**
   The directive MACRO informs the assembler the beginning of MACRO.
   It consist of name of macro followed by keyword MACRO and MACRO arguments if any.

   Syntax:

   ```
   Macro_name      MACRO [arg1, arg2, ……argN)
       ……
   Endm
   ```

2. **ENDM Directive: END OF MACRO**
   The directive ENDM informs the assembler the end of macro.

   Syntax: ENDM

3. **LOCAL**
   - Macros are expanded directly in code, therefore if there are labels inside the macro definition you may get "Duplicate declaration" error when macro is used for twice or more. To avoid such problem, use LOCAL directive followed by names of variables.

   Syntax: LOCAL <lable>

   Example with MACRO, ENDM, LOCAL Directive

   ```
   MyMacro2      MACRO
   LOCAL label 1, label 2
   CMP AX, 2
   JE label 1
   CMP AX, 3
   JE label 2
   label 1:
                INC AX
   ```

```
        label 2:
                ADD AX, 2
ENDM
data segment
data ends
code segment
start:  assume cs: code, ds: data
        mov ax, data
        mov ds, ax
                mov ax, 02h
                MyMacro2
                MyMacro2
              mov ah, 4ch
        int 21h
code ends
end start
```

Q.6 Attempt any TWO of the following : [12]

Q.6(a) With examples, describe any four String instructions in 8086 assembly language. [6]

Ans.: (i) 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
```

(ii) 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
```

(iii) 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
```

(iv) 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
```

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

Syntax STOS**/STOSB/STOSW**

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

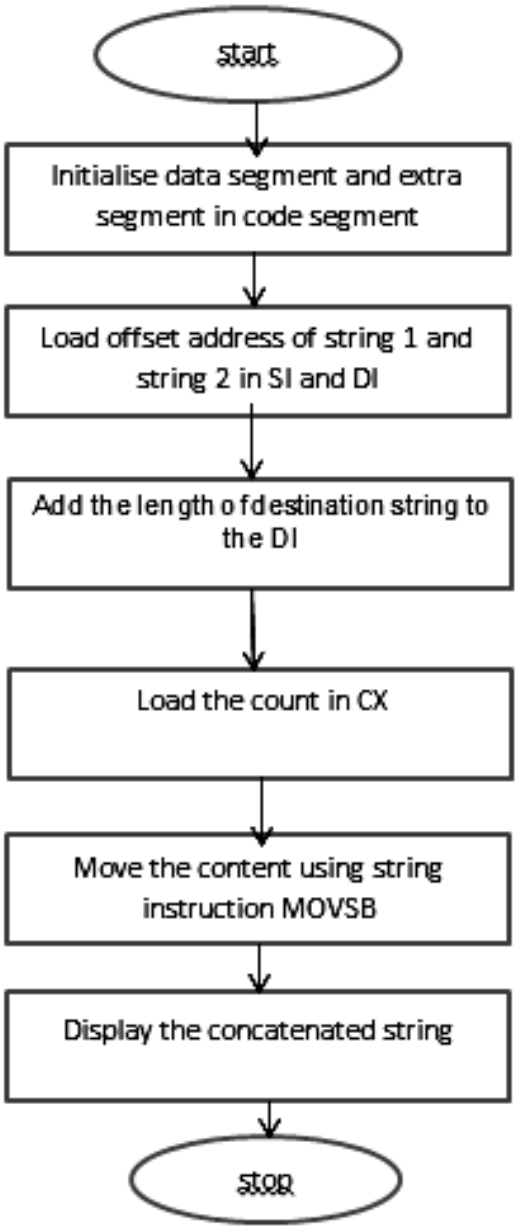
It copies a byte or word from AL or AX to a memory location pointed by DI in extra segment CX may contain the counter and DF may either set or reset.

Q.6(b) Write an ALP for concatenation of two strings. Draw flowchart and assume suitable data. **[6]**

Ans.:

```
Data    segment

string1 db  "HELLO", 0
string2 db  "HII ", 0

Data ends

Code segment

ASSUME  CS: code, DS: data

start:
    cld
    mov    ax,  data
    mov    DS, ax
    mov    ES,  ax
    mov    SI,  offset string1
    mov    DI,  offset string2
    add    DI, 3 ; Adding the length of destination string to the DI

    mov    cx,  5
    rep movsb ; This should concat two strings

    mov    SI, offset string2
    up:  lodsb          ; Printing loop
        mov dl, al
        mov ah, 2h
        int 21h
        jmp up
        mov ah, 4ch
```

```
    int 21h
        code ends
end start
```



Q.6(c) Write an ALP to transfer 10 bytes of data from one memory location to another. [6]
Also draw the flow chart for the same.

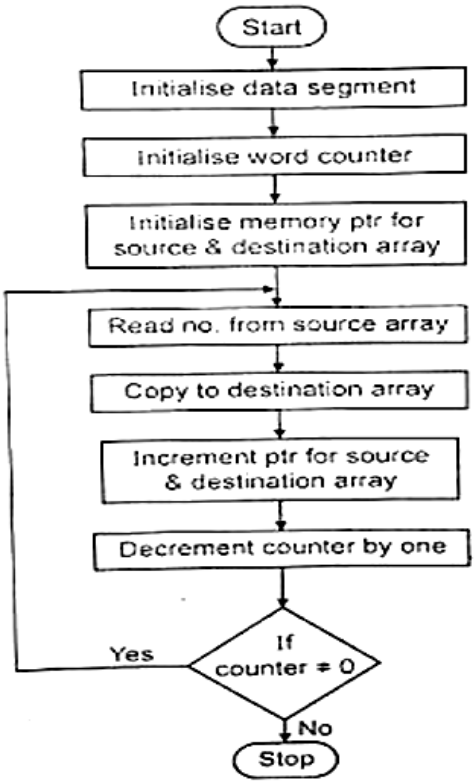
Ans.:

```
DATA segment
block1 db 10 dup(10h)
block2 db 10 dup(0)
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS: DATA, ES: EXTRA
START: MOV DX, DATA        ; initialize data seg
    MOV DS, DX
MOV DX, EXTRA
    MOV ES, DX

    LEA SI, BLOCK1
    LEA DI, BLOCK2
    MOV CX, 000AH
    CLD
```

```
REP MOVSB
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

(OR)

```
DATA SEGMENT
block1 db 10 dup(10h)
block2 db 10 dup(0)
DATA ENDS
CODE SEGMENT
ASSUME CS: CODE, DS, DATA
START: MOV DX, DATA        ; initialize data seg
    MOV DS, DX
    MOV EX, DX

    LEA SI, BLOCK 1
    LEA DI, BLOCK 2
    MOV CX, 000AH
    CLD
BACK: MOV AL, [SI]         ; REP MOVSB
    MOV [DI], AL
    INC SI
    INC DI
    DEC CX
    JNZ BACK
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```



❑ ❑ ❑ ❑ ❑