2 Marks Questions

1. State the function of BHE and A0 pins of 8086.

Ans - BHE: BHE stands for Bus High Enable. It is available at pin 34 and used to indicate the transfer of data using data bus D8-D15. This signal is low during the first clock cycle, thereafter it is active.

A0: A0 is analogous to BHE for the lower byte of the data bus, pinsD0-D7. A0 bit is Low during T1 state when a byte is to be transferred on the lower portion of the bus in memory or I/O operations.

BHE	A0	Word / Byte access
0	0	Whole word from even address
0	1	Upper byte from / to odd address
1	0	Lower byte from / to even address
1	1	None

2. How single stepping or tracing is implemented in 8086?

Ans - By setting the Trap Flag (TF) the 8086 goes to single-step mode. In this mode, after the implementation of every instruction s 8086 generates an internal interrupt and by writing some interrupt service routine we can show the content of desired registers and memory locations. So it is useful for debugging the program.

OR

If the trap flag is set, the 8086 will automatically do a type-1 interrupt after each instruction executes. When the 8086 does a type-1 interrupt, it pushes the flag register on the stack.

OR

The instructions to set the trap flag are:

```
PUSHF                      ; Push flags on stack
MOV BP,SP                  ; Copy SP to BP for use as index
OR WORD PTR[BP+0],0100H    ; Set TF flag
POPF                       ; Restore flag Register
```

3. State the role Debugger in assembly language programming.

Ans - **Debugger:** Debugger is the program that allows the extension of program in single step mode under the control of the user.

The process of locating & correcting errors using a debugger is known as Debugger.

Some examples of debugger are DOS debug command Borland turbo debugger TD, Microsoft debugger known as code view cv, etc…

4. Define Macro & Procedure.

Ans - **Macro:** A MACRO is group of small instructions that usually performs one task. It is a reusable section of a software program. A macro can be defined anywhere in a program using directive MACRO &ENDM.

General Form :

MACRO-name MACRO [ARGUMENT 1,……….ARGUMENT N]

MACRO CODIN GOES HERE

ENDM

E.G DISPLAY MACRO 12,13

MACRO STATEMENTS

ENDM

Procedure: A procedure is group of instructions that usually performs one task. It is a reusable section of a software program which is stored in memory once but can be used as often as necessary. A procedure can be of two types. 1) Near Procedure 2) Far Procedure

```
 Procedure can be defined as

  Procedure_name PROC

 - - -

 - - - -
Procedure_name
ENDP
```

```
 For  Example

  Addition  PROC near

 - - - -

Addition ENDP
```

5. Write ALP for addition of two 8bit numbers. Assume suitable data

Ans –

```
.Model small

      .Data

       NUM DB 12H

       .Code

       START:

       MOV AX, @DATA

       MOV DS,AX

       MOV AL, NUM

       MOV AH,13H

       ADD AL,AH

       MOV AH, 4CH

       INT  21H ENDS

       END
```

6. List any four instructions from the bit manipulation instructions of 8086.

Ans - Bit Manipulation Instructions

These instructions are used to perform operations where data bits are involved, i.e. operations like logical, shift, etc.

Following is the list of instructions under this group –

Instructions to perform logical operation

- **NOT** – Used to invert each bit of a byte or word.
- **AND** – Used for adding each bit in a byte/word with the correspondingbit in another byte/word.
- **OR** – Used to multiply each bit in a byte/word with the correspondingbit in another byte/word.

XOR – Used to perform Exclusive-OR operation over each bit in abyte/word with the corresponding bit in another byte/word.

7. State the use of REP in string related instructions.

Ans –

- This is an instruction prefix which can be used in string instructions.
- It causes the instruction to be repeated CX number of times.
- After each execution, the SI and DI registers are incremented/decremented based on the DF (Direction Flag) in the flag register and CX is decremented i.e. DF = 1; SI, DI decrements.

  E.g. MOV CX, 0023H

  CLD

  REP MOVSB

  The above section of a program will cause the following string operation

  ES: [DI] ← DS: [SI]

  SI ← SI + I

  DI ← DI + I CX ← CX –

  1

  to be executed 23H times (as CX = 23H) in auto incrementing mode (asDF is cleared).

REPZ/REPE (Repeat while zero/Repeat while equal)

- It is a conditional repeat instruction prefix.
- It behaves the same as a REP instruction provided the Zero Flag is set(i.e. ZF = 1).
- It is used with CMPS instruction.

REPNZ/REPNE (Repeat while not zero/Repeat while not equal)

- It is a conditional repeat instruction prefix.
- It behaves the same as a REP instruction provided the Zero Flag is reset(i.e. ZF = 0).

It is used with SCAS instruction.

8. State the function of READY and INTR pin of 8086

Ans –

Ready:

It is used as acknowledgement from slower I/O device or memory. It is Active high signal, when high; it indicates that the peripheral device is ready to transfer data.

INTR:

This is a level triggered interrupt request input, checked during last clock cycle of each instruction to determine the availability of request. If any interrupt request is occurred, the processor enters the interrupt acknowledge cycle.

9. What is role of XCHG instruction in assembly language program? Give example

Ans –

Role of XCHG:

This instruction exchanges the contents of a register with the contents of another register or memory location.

Example: XCHG AX, BX ; Exchange the word in AX with word in BX.

10. List assembly language programming tools.

Ans –

1. Editors

2. Assembler

3. Linker

4. Debugger.

11. Define Macro. Give syntax.

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.

Syntax:

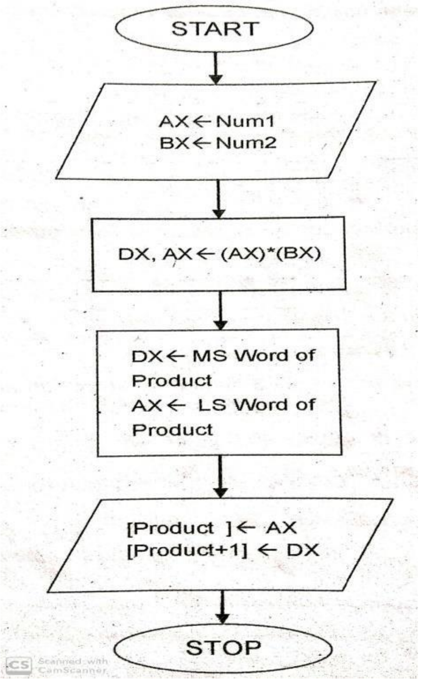
Macro_name MACRO[arg1,arg2,…..argN)

…..

End

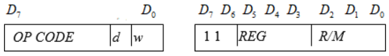
12. Draw flowchart for multiplication of two 16 bit numbers.

Ans –



13. Draw machine language instruction format for Register-to-Registertransfer.

Ans –



14. State the use of STC and CMC instruction of 8086.

Ans –

STC – This instruction is used to Set Carry Flag. CF←1

CMC – This instruction is used to Complement Carry Flag.

CF← ~ CF

15. State the functions of the following pins of 8086 Microprocessor :

i) ALE

ii) M/IO

Ans –

ALE - It stands for address enable latch and is available at pin 25. A positive pulse is generated each time the processor begins any operation. This signal indicates the availability of a valid address on the address/data lines.

M/IO - This signal is used to distinguish between memory and I/O operations. When it is high, it indicates I/O operation and when it is low indicating the memory operation. It is available at pin 28.

16. State the function of STC and CMC Instruction of 8086.

Ans –

STC – This instruction is used to Set Carry Flag. CF ←1

CMC – This instruction is used to Complement Carry Flag. CF ←~ CF

17. List the program development steps for assembly languageprogramming.

Ans –

Program Development steps:

1. Defining the problem

2. Algorithm

3. Flowchart

4. Initialization checklist

5. Choosing instructions

6. Converting algorithms to assembly language program

18. Define MACRO with its syntax.
Ans –

Macro: A MACRO is group of small instructions that usually performs one task. It is a reusable section of a software program. A macro can be defined anywhere in a program using directive MACRO &ENDM.

Syntax: MACRO-name MACRO [ARGUMENT 1,… ARGUMENT N]

```
          ----
ENDM
```

19. Write an ALP to Add two 16-bit numbers.

Ans –

```
data segmenta
dw 0202h b dw
0408h c dw ?
data ends

code segment
assume cs:code,ds:data
start:
mov ax,data
mov ds,ax mov
ax,a mov bx,b
add ax,bx mov
c,ax int  03h
code ends
end start
```

20. State two examples of each, Immediate and based indexedAddressing modes.

Ans –

Immediate Addressing mode:

1. MOV AX, 2000H
2. MOV CL, 0AH
3. ADD AL, 45H
4. AND AX, 0000H

Based indexed Addressing mode:

1. ADD CX, [AX+SI]
2. MOV AX, [AX+DI]
3. MOV AL, [SI+BP+2000]

21. State the use of OF and AF flags in 8086.
Ans –

Auxiliary Carry Flag (AF):

This flag is used in BCD (Binary-coded Decimal) operations.

This flag is set to 1 if there is a CARRY from the lower nibble or BORROW for the lower nibble in binary representation; else it is set to zero.

Overflow Flag (OF):

This flag will be set (1) if the result of a signed operation is too large to fit in the numberof bits available to represent it, otherwise reset (0).

4 Marks Questions

1. Explain the concept of pipelining in 8086. State the advantages ofpipelining (any two).

Ans –

Pipelining:

1. The process of fetching the next instruction when the present instructionis being executed is called as pipelining.
2. Pipelining has become possible due to the use of queue.
3. BIU (Bus Interfacing Unit) fills in the queue until the entire queue isfull.
4. BIU restarts filling in the queue when at least two locations of queue arevacant.

Advantages of pipelining:

- The execution unit always reads the next instruction byte from thequeue in BIU. This is faster than sending out an address to the memory and waiting for the next instruction byte to come.
- More efficient use of processor.
- Quicker time of execution of large number of instruction.

In short pipelining eliminates the waiting time of EU and speeds up the processing. -The 8086 BIU will not initiate a fetch unless and until thereare two empty bytes in its queue. 8086 BIU normally obtains two instruction bytes per fetch.

2. Compare Procedure and Macros. (4 points).

Ans –

Procedure	**Macro**
Procedures are used for large group of instructions to be repeated.	Procedures are used for small group of instructions to be repeated.
Object code is generated onlyonce in memory.	Object code is generated everytime the macro is called.
CALL & RET instructions are used to call procedure and returnfrom procedure.	Macro can be called just bywriting its name.
Length of the object file is less.	Object file becomes lengthy.
Directives PROC & ENDP areused for defining procedure.	MACRO and ENDM are usedfor defining MACRO.
Directives More time is requiredfor its execution.	Less time is required for it's execution.
Procedure can be defined as	Macro can be defined as MACRO-name MACRO

Procedure_name PROC ---- - - - - Procedure_name ENDP	[ARGUMENT,………. ARGUMENT N] - - - - ------- ENDM
For Example Addition PROC near - - - - Addition ENDP	For Example Display MACRO msg - - - - ENDM

3. Explain any two assembler directives of 8086.

Ans –

1. DB – The DB directive is used to declare a BYTE -2-BYTE variable – A BYTE is made up of 8 bits.
Declaration examples:
Byte1 DB 10h

Byte2 DB 255; 0FFh, the max. possible for a BYTE

CRLF DB 0Dh, 0Ah, 24h ;Carriage Return, terminator BYTE

2. DW – The DW directive is used to declare a WORD type variable – A WORD occupies 16 bits or (2 BYTE).
Declaration examples:
Word DW 1234h

Word2 DW 65535; 0FFFFh, (the max. possible for a WORD)

3. DD – The DD directive is used to declare a DWORD – A DWORD double word is made up of 32 bits =2 Word's or 4 BYTE.
Declaration examples:
Dword1 DW 12345678h

Dword2 DW 4294967295 ;0FFFFFFFFh.

4. EQU -
The EQU directive is used to give name to some value or symbol. Each time the assembler finds the given names in the program, it will replace the name with the value or a symbol. The value can be in the range 0 through 65535 and it can be another Equate declared anywhere above or below.

The following operators can also be used to declare an Equate:
THIS BYTE

THIS WORD

THIS DWORD

A variable – declared with a DB, DW, or DD directive – has an address and has space reserved at that address for it in the .COM file. But an Equate does not have an address or space reserved for it in the .COM file.

Example:
A – Byte EQU THIS BYTE

DB 10

A_ word EQU THIS WORD
DW 1000

A_ dword EQU THIS DWORD

DD 4294967295

Buffer Size EQU 1024

Buffer DB 1024 DUP (0)

Buffed_ ptr EQU $; actually points to the next byte after the; 1024th byte in buffer.

5. SEGMENT:
It is used to indicate the start of a logical segment. It is the name given to the segment. Example: the code segment is used to indicate to the assembler the start of logical segment.

6. PROC: (PROCEDURE)
It is used to identify the start of a procedure. It follows a name we give the procedure.

After the procedure the term NEAR and FAR is used to specify the procedure

Example: SMART-DIVIDE PROC FAR identifies the start of procedure named SMART-DIVIDE and tells the assembler that the procedure is far.

4. Write classification of instruction set of 8086. Explain any one type out of them.

Ans –

classification of instruction set of 8086

- Data Transfer Instructions
- Arithmetic Instructions
- Bit Manipulation Instructions
- String Instructions
- Program Execution Transfer Instructions (Branch & Loop Instructions)
- Processor Control Instructions
- Iteration Control Instructions
- Interrupt Instructions

1) **Arithmetic Instructions:**

These instructions are used to perform arithmetic operations like addition, subtraction, multiplication, division, etc.

ADD:

The add instruction adds the contents of the source operand to the destination operand.

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

ADC: Add with Carry

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

SUB: Subtract

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

SBB: Subtract with Borrow

The subtract with borrow instruction subtracts the source operand and theborrow flag (CF) which may reflect the result of the previous calculations, from thedestination operand

Eg. SBB AX, 0100HSBB AX,

BX
SBB AX, [5000H] SBB
[5000H], 0100H

INC: Increment

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

DEC: Decrement

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

NEG: Negate

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

CMP: Compare

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

MUL: Unsigned Multiplication Byte or Word

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

IMUL: Signed Multiplication

This instruction multiplies a signed byte in source operand by a signed byte inAL or a signed word in source operand by a signed word in AX.Eg. IMUL
BH
IMUL CXIMUL [SI]

CBW: Convert Signed Byte to Word

This instruction copies the sign of a byte in AL to all the bits in AH. AH is thensaid to be sign extension of AL.

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

CWD: Convert Signed Word to Double Word

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

DIV: Unsigned division

This instruction is used to divide an unsigned word by a byte or to divide an unsigned double word by a word.
Eg.

```
DIV CL ; Word in AX / byte in CL
        ; Quotient in AL, remainder in AH
DIV CX ; Double word in DX and AX / word
        ; in CX, and Quotient in AX,
       ; remainder in DX
```

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

STC:

It sets the carry flag to 1.

CLC:

It clears the carry flag to 0.

CMC:

It complements the carry flag.

STD:

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

CLD:

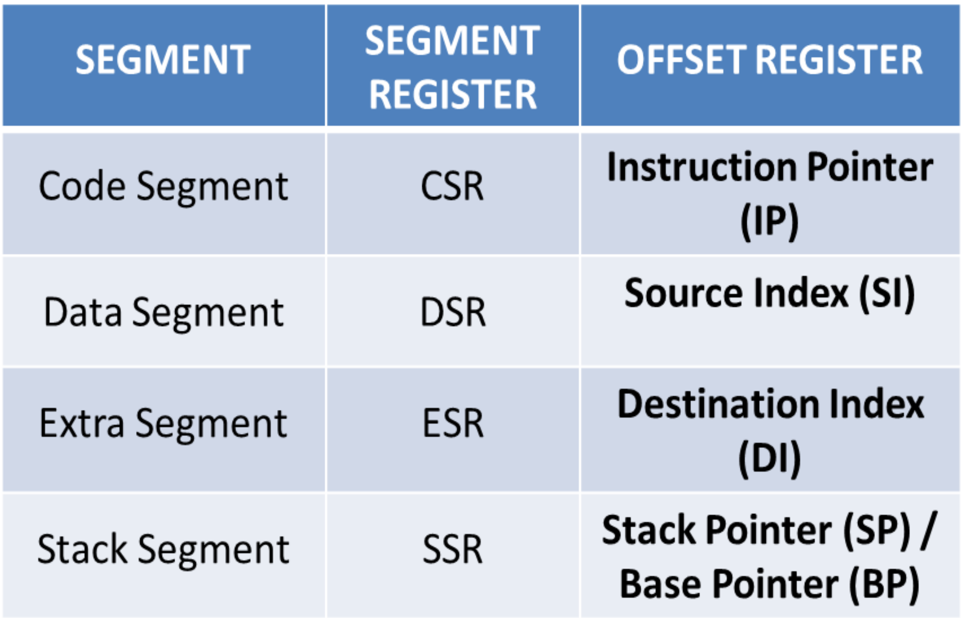
It clears the direction flag to 0.
If it is reset, the string bytes are accessed from lower memory address to higher memory address.

5. Explain memory segmentation in 8086 and list its advantages.(any two)

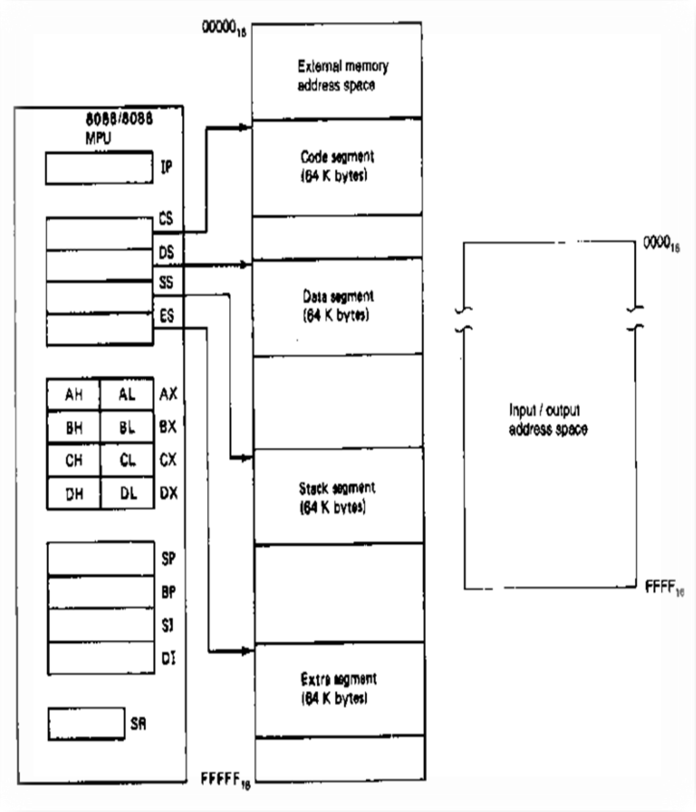
Ans –

Memory Segmentation:

- In 8086 available memory space is 1MByte.
- This memory is divided into different logical segments and eachsegment has its own base address and size of 64 KB.
- It can be addressed by one of the segment registers.
- There are four segments.



SEGMENT	SEGMENT REGISTER	OFFSET REGISTER
Code Segment	CSR	**Instruction Pointer (IP)**
Data Segment	DSR	**Source Index (SI)**
Extra Segment	ESR	**Destination Index (DI)**
Stack Segment	SSR	**Stack Pointer (SP) / Base Pointer (BP)**


00000₁₆
External memory address space
8088/8088 MPU
IP
CS
DS
SS
ES
AH
AL
AX
BH
BL
BX
CH
CL
CX
DH
DL
DX
SP
BP
SI
DI
SR
Code segment (64 K bytes)
Data segment (64 K bytes)
Stack segment (64 K bytes)
Extra segment (64 K bytes)
FFFFF₁₆
0000₁₆
Input / output address space
FFFF₁₆

Advantages of Segmentation:

- The size of address bus of 8086 is 20 and is able to address 1 Mbytes( ) of physical memory.
- The compete 1 Mbytes memory can be divided into 16 segments,each of 64 Kbytes size.
- It allows memory addressing capability to be 1 MB.
- It gives separate space for Data, Code, Stack and Additional Datasegment as Extra segment size.
- The addresses of the segment may be assigned as 0000*H* to F000*H* respectively.
- The offset values are from 00000H to FFFFFH

Segmentation is used to increase the execution speed of computer system so that processor can able to fetch and execute the data from memory easily and fast.

6. Write an ALP to count the number of positive and negative numbers in array.

Ans –

;Count Positive No. And Negative No.S In Given ;Array Of 16 Bit No.
;Assume array of 6 no.s

```
 CODE SEGMENT
ASSUME CS:CODE,DS:DATASTART:
   MOV AX,DATA
                   MOV DS,AX MOV
```

```
                    DX,0000H MOV CX,COUNT
                    MOV SI, OFFSET ARRAYNEXT:
                MOV AX,[SI]
                    ROR AX,01H JC
                    NEGATIVEINC DL
                    JMP COUNT_IT
 NEGATIVE: INC DH COUNT_IT: INC SI
                   INC SI
                    LOOP NEXT
                    MOV     NEG_COUNT,DL    MOV
                    POS_COUNT,DHMOV AH,4CH
                    INT 21H
CODE ENDS

 DATA SEGMENT
 ARRAY  DW  F423H,6523H,B658H,7612H,  2300H,1559H  COUNT
 DW 06H
 POS_COUNT         DB         ?
 NEG_COUNT    DB    ?  DATA
 ENDS
END START
```

7. Write an ALP to find the sum of series. Assume series of 10 numbers.

Ans –

```
; Assume TEN , 8 bit HEX numbers
CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START: MOV AX,DATA

            MOV DS,AX

            LEA SI,DATABLOCK

            MOV CL,0AH

      UP:MOV AL,[SI]

            ADD RESULT_LSB,[SI]
            JNC DOWN

            INC REULT_MSB

  DOWN:INC SI

             LOOP UP

  CODE ENDS
```

```
DATA SEGMENT

DATABLOCK DB 45H,02H,88H,29H,05H,45H,78H,

                     95H,62H,30H

RESULT_LSB DB 0

RESULT_MSB DB 0

DATA ENDS


          END
```

8. With neat sketches demonstrate the use of re-entrant and recursiveprocedure.

Ans –

Reentrant Procedure:

A reentrant procedure is one in which a single copy of the program code can be shared by multiple users during the same period of time. Re-entrance has two key aspects: The program code cannot modify itself and the local data for each user must be stored separately.

Recursive procedures:

An active **procedure** that is invoked from within itself or from within another active **procedure** is a **recursive procedure**. Such an invocation is called **recursion**. A **procedure** that is invoked **recursively** must have the**RECURSIVE** attribute specified in the **PROCEDURE** statement.

9. Describe mechanism for generation of physical address in 8086 with suitable example.

Ans –

Fig.: Mechanism used to calculate physical address in 8086

As all registers in 8086 are of 16 bit and the physical address will be in 20 bits.For this reason the above mechanism is helpful.

Logical Address is specified as segment: offset

Physical address is obtained by shifting the segment address 4 bits to the leftand adding the offset address.

Thus the physical address of the logical address A4FB:4872 is:

A4FB0

+ 4872

A9822

OR

i.e. Calculate physical Address for the given

CS= 3525H, IP= 2450H.

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

10. Write ALP to count ODD and EVEN numbers in an array.

Ans –

```
;Count  ODD and EVEN No.S In Given ;Array Of 16 Bit No.
;Assume array of 10 no.s

CODE SEGMENT
ASSUME CS:CODE,DS:DATA
   START:  MOV AX,DATA
                   MOV DS,AX
```

```
                MOV DX,0000H
                MOV CX,COUNT
                MOV SI, OFFSET ARRAY1
NEXT:           MOV AX,[SI]
                ROR AX,01H
                JC ODD_1
                INC DL
                 JMP COUNT_IT
ODD_1   :        INC DH
COUNT_IT:        INC SI
                 INC SI
                LOOP NEXT
                MOV ODD_COUNT,DH
                MOV EVENCNT,DL
                MOV AH,4CH
                INT 21H
CODE ENDS

DATA SEGMENT
ARRAY1 DW  F423H, 6523H, B658H, 7612H, 9875H,
                 2300H, 1559H, 1000H, 4357H,  2981H
COUNT DW 0AH
ODD_COUNT DB ?
EVENCNT DB ?
DATA ENDS
END START
```

11. Write ALP to perform block transfer operation of 10 numbers.

Ans –;Assume block of TEN 16 bit no.s

;**Data Block Transfer** Using String InstructionCODE

```
    SEGMENT
    ASSUME CS:CODE,DS:DATA,ES:EXTRAMOV
    AX,DATA
    MOV DS,AX MOV
    AX,EXTRAMOV ES,AX
    MOV CX,000AH LEA
    SI,BLOCK1
    LEA DI,ES:BLOCK2CLD
    REPNZ  MOVSWMOV
    AX,4C00H INT 21H
    CODE ENDS DATA
SEGMENT
     BLOCK1 DW 1001H,4003H,6005H,2307H,4569H, 6123H,1865H,
                     2345H,4000H,8888H
 DATA ENDS EXTRA
 SEGMENT
   BLOCK2 DW ?EXTRA
 ENDS
END
```

12. Write ALP using procedure to solve equation such as Z= (A+B)*(C+D)

Ans –

```
; Procedure For Addition
SUM PROC NEAR
ADD AL,BL
RET
SUM ENDP

DATA SEGMENT
NUM1 DB 10H
NUM2 DB 20H
NUM3 DB 30H
NUM4 DB 40H
RESULT DB?
DATA ENDS

CODE SEGMENT
ASSUME CS: CODE,DS:DATA
START:MOV AX,DATA
            MOV DS,AX
            MOV AL,NUM1
            MOV BL,NUM2
            CALL SUM
            MOV CL,AL
            MOV AL, NUM3
            MOV BL,NUM4
            CALL SUM
            MUL CL
            MOV RESULT,AX
MOV AH,4CH
INT 21H CODE ENDSEND
```

13. Write ALP using macro to perform multiplication of two 8 Bit Unsigned numbers.

Ans –

```
; Macro For Multiplication

PRODUCT MACRO FIRST,SECOND
MOV AL,FIRST
MOV BL,SECOND
MUL BL
PRODUCT ENDM
```

```
DATA SEGMENT
NO1 DB 05H
NO2 DB 04H
MULTIPLE DW ?
DATA ENDS

CODE SEGMENT
ASSUME CS: CODE,DS:DATA
START:MOV AX,DATA
            MOV DS,AX
            PRODUCT NO1,NO2
            MOV MULTIPLE, AX
MOV AH,4CH
INT 21H
CODE ENDS
END
```

14. Give the difference between intersegment and intrasegment CALL.

Ans –

Sr.no	**Intersegment Call**	**Intrasegment Call**
1.	It is also called Farprocedure call.	It is also called Nearprocedure call.
2.	A far procedure refers to aprocedure which is in the different code segment from that of the call instruction.	A near procedure refers to a procedure which is in the same code segment from that of the call instruction.
3.	This procedure call replaces the old CS:IP pairswith new CS:IP pairs	This procedure call replacesthe old IP with new IP.
4.	The value of the old CS:IPpairs are pushed on to the stack SP=SP-2 ;Save CS onstack SP=SP-2 ;Save IP (newoffset address of calledprocedure)	The value of old IP ispushed on to the stack. SP=SP-2 ;Save IP on stack(address of procedure)
5.	More stack locations arerequired	Less stack locations arerequired
6.	Example :- Call FAR PTRDelay	Example :- Call Delay

15. Draw flag register of 8086 and explain any four flags.

Ans –

Flag Register of 8086

Conditional /Status Flags

C-Carry Flag : It is set when carry/borrow is generated out of MSB of result. (i.e D_7 bit for 8-bit operation, D_{15} bit for a 16 bit operation).

P-Parity Flag This flag is set to 1 if the lower byte of the result contains evennumber of 1's otherwise it is reset.

AC-Auxiliary Carry Flag This is set if a carry is generated out of the lowernibble, (i.e. From D3 to D4 bit)to the higher nibble

Z-Zero Flag This flag is set if the result is zero after performing ALU operations. Otherwise it is reset.

S-Sign Flag This flag is set if the MSB of the result is equal to 1 after performing ALU operation , otherwise it is reset.

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 destinationregister.

Control Flags

T-Trap Flag If this flag is set ,the processor enters the single step execution mode.

I-Interrupt Flag it is used to mask(disable) or unmask(enable)the INTR interrupt. When this flag is set,8086 recognizes interrupt INTR. When itis reset INTR is masked.

D-Direction Flag It selects either increment or decrement mode for DI &/orSI register during string instructions.

16. Explain assembly language program development steps.

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 theprogram 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.

Converting algorithms to assembly language program: Every step in the algorithm is converted into program statement using correct and efficientinstructions or group of instructions.

17. Explain logical instructions of 8086.(Any Four)

Ans –

Logical instructions.

1) AND- Logical AND

Syntax : AND destination, sourceOperation

Destination ←destination AND sourceFlags Affected

:CF=0,OF=0,PF,SF,ZF

This instruction AND's each bit in a source byte or word with the same number bit in a destination byte or word. The result is put in destination.

Example: AND AX, BX

- **AND AL,BL**
    - **AL 1111 1100**
    - **BL 0000 0011**

- **AL←0000 0000 (AND AL,BL)**

2) OR – Logical OR

Syntax **:OR destination, source**

Operation

Destination OR source

Flags Affected :CF=0,OF=0,PF,SF,ZF

This instruction OR's each bit in a source byte or word with the corresponding bit in a destination byte or word. The result is put in aspecified destination.

Example :

- OR AL,BL
- AL 1111 1100
- BL 0000 0011

- - - - - - - - - - - - - -

- AL←1111 1111

3) NOT – Logical Invert

Syntax : NOT destination

Operation: Destination NOT destination

Flags Affected :None

The NOT instruction inverts each bit of the byte or words at thespecified destination.

Example

NOT BL

BL = 0000 0011

NOT BL gives 1111 1100

4) XOR – Logical Exclusive OR

Syntax : **XOR destination, source**

Operation : **Destination Destination XOR sourceFlags Affected**

:CF=0,OF=0,PF,SF,ZF

This instruction exclusive, OR's each bit in a source byte or word with the same number bit in a destination byte or word.

Example(optional)

XOR AL,BL

- AL 1111 1100
- BL 0000 0011

- **AL←1111 1111 (XOR AL,BL)**

5)TEST

Syntax : TEST Destination, Source
This instruction AND's the contents of a source byte or word with the contents of specified destination byte or word and flags are updated, , flags are updated as result ,but neither operands are changed.
Operation performed:

Flags ← set for result of (destination AND source)

Example: (Any 1)
TEST AL, BL ; AND byte in BL with byte in AL, no result, Update PF,SF, ZF.

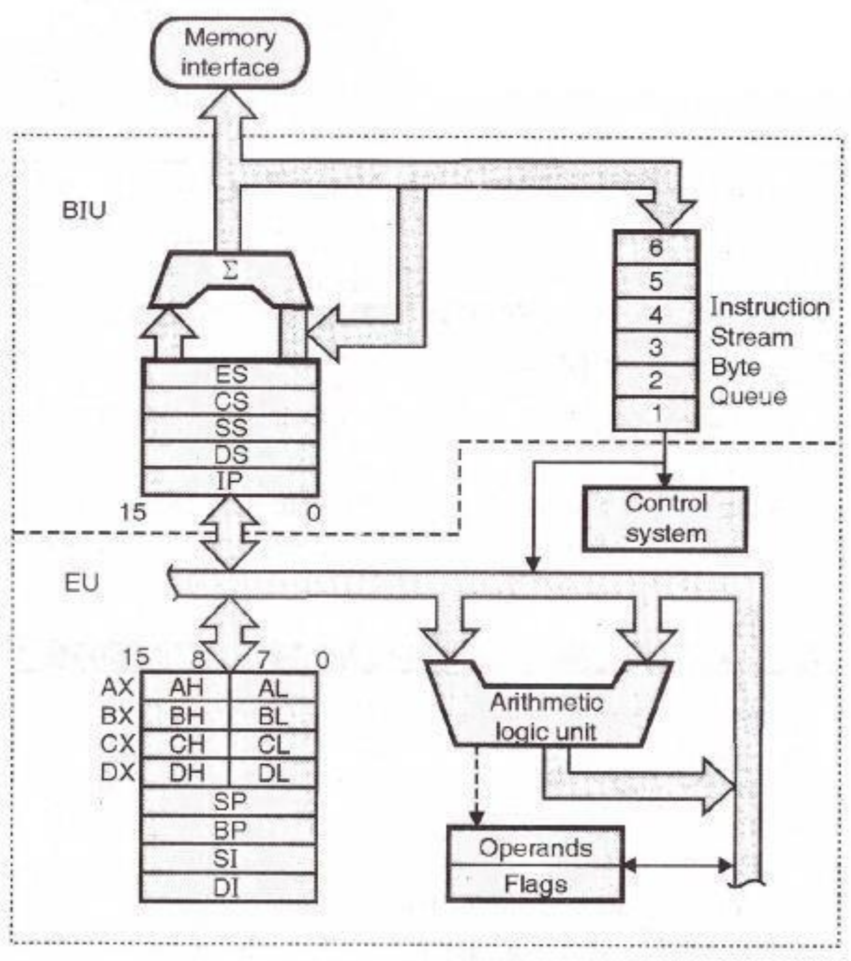
e.g **MOV AL, 00000101**

TEST AL, 1 ; ZF = 0.

TEST AL, 10b ; ZF = 1

18. Draw functional block diagram of 8086 microprocessor.

Ans –



8086 internal architecture

19. Write an ALP to add two 16-bit numbers.

Ans –

```
DATA SEGMENT

NUMBER1 DW 6753H

NUMBER2 DW 5856H

SUM DW 0

 DATA ENDS

CODE SEGMENT

ASSUME CS: CODE, DS: DATA

START: MOV AX, DATA
 MOV DS, AX

MOV AX, NUMBER1

MOV BX, NUMBER2

ADD AX, BX

MOV SUM, AX

MOV  AH,  4CH

INT 21H

CODE ENDS
END START
```

20. Write an ALP to find length of string.

Ans –

```
Data Segment

STRG DB 'GOOD MORNING$'

LEN DB ?

DATA ENDS

CODE SEGMENT

 START:

ASSUME CS: CODE, DS : DATA

 MOV DX, DATA

 MOV DS,DX

LEA SI, STRG
```

```
MOV CL,00H

MOV AL,'$'

 NEXT: CMP AL,[SI]

JZ EXIT

ADD CL,01H

 INC SI
 JMP

 NEXT EXIT: MOV LEN,CL

MOV AH,4CH

INT 21H
 CODE ENDS
```

21. Write an assembly language program to solve p= x^2+y^2 using Macro.(xand y are 8 bit numbers.

Ans –

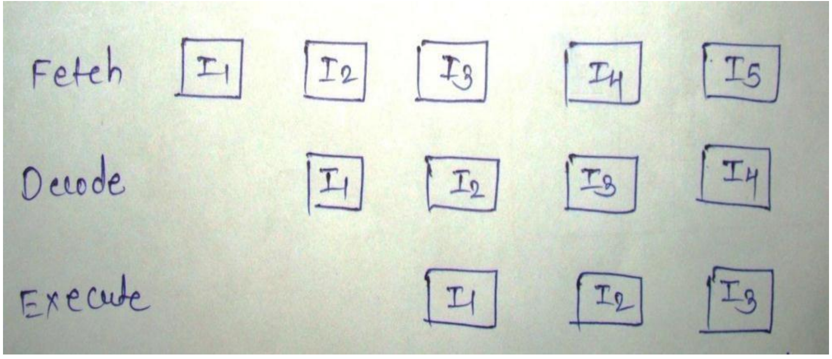
```
.MODEL SMALL

PROG MACRO a,b

MOV al,a

MUL al

MOV bl,al

MOV al,b

MUL al

ADD al,bl

ENDM

.DATA

x DB 02H

y DB 03H

p DB DUP()

.CODE
```

```
START:

MOV ax,data

MOV ds,ax

PROG x, y

MOV p,al

MOV ah,4Ch

Int 21H

END
```

22. What is pipelining? How it improves the processing speed.

Ans –

- In 8086, pipelining is the technique of overlapping instruction fetchand execution mechanism.
- To speed up program execution, the BIU fetches as many as sixinstruction bytes ahead of time from memory. The size of instruction prefetching 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 otherinstruction.
- BIU stores the fetched instructions in a 6 level deep FIFO . The BIU can be fetching instructions bytes while the EU is decoding an instruction or executing an instruction which does not require use ofthe buses.
- When the EU is ready for its next instruction, it simply reads theinstruction 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 instructionbyte or bytes.
- This improves overall speed of the processor



23. Write an ALP to count no.of 0’s in 16 bit number.

Ans –

```
 DATA SEGMENT
 N DB 1237H
Z DB 0
 DATA ENDS
 CODE SEGMENT
 ASSUME DS:DATA, CS:CODE
 START:
 MOV DX,DATA
 MOV DS,DX
 MOV AX, N
 MOV CL,08
 NEXT: ROL AX,01
 JC ONE
 INC Z
 ONE: LOOP NEXT
 HLT
CODE ENDS
END START
```

24. Write an ALP to find largest number in array of elements 10H, 24H,02H, 05H, 17H.

Ans –DATA SEGMENT

```
 ARRAY DB 10H,24H,02H,05H,17H
 LARGEST DB 00H
 DATA ENDS
 CODE SEGMENT
 START:
 ASSUME CS:CODE,DS:DATA
 MOV DX,DATA
  MOV DS,DX
  MOV CX,04H
  MOV SI ,OFFSET
 ARRAY MOV AL,[SI]
 UP: INC SI
  CMP AL,[SI]
  JNC NEXT
  MOV AL,[SI]
  NEXT: DEC CX
  JNZ UP
  MOV LARGEST,AL
  MOV  AX,4C00H
  INT 21H
CODE ENDS
END START
```

25. Write an ALP for addition of series of 8-bit number using procedure.

Ans –

```
DATA SEGMENT
NUM1 DB 10H,20H,30H,40H,50H
RESULT DB 0H
  CARRY DB 0H
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START: MOV DX,DATA
MOV DS, DX
MOV CL,05H
MOV SI, OFFSET NUM1
UP: CALL SUM
INC SI
LOOP UP
MOV AH,4CH
INT 21H

SUM PROC; Procedure to add two 8 bit numbers
MOV AL,[SI]
ADD RESULT, AL
JNC NEXT
INC CARRY
NEXT: RET
SUM ENDP
CODE ENDS
END START
```

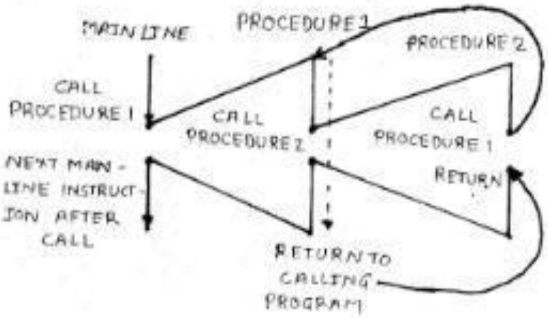
26. Describe re-entrant and recursive procedure with schematic diagram.

Ans –

In some situation it may happen that Procedure 1is called from main programProcrdure2 is called from procedure1And procrdure1 is again called from procdure2. In this situation program execution flow reenters in the procedure1. These types of procedures are called re enterant procedures. TheRET instruction at the end of procrdure1 returns to procedure2. The RET instruction at the end of procedure2 will return the execution to procedure1.Procedure1 will again executed from where it had stopped at thetime of calling procrdure2 and the RET instruction at the end of this will return the program execution to main program.

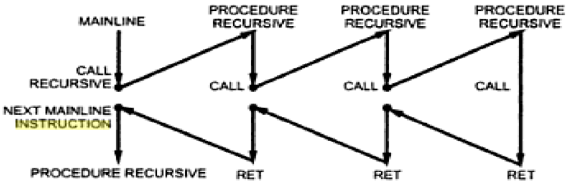
The flow of program execution for re-entrant procedure is as shown in FIG.

Sketch :



Recursive Procedure

A recursive procedure is a procedure which calls itself. Recursive procedures are used to work with complex data structures called trees. If the procedures is called with N (recursion depth) = 3. Then the n is decremented by one after each procedure CALL and the procedure is called until n = 0. Fig. shows the flow diagram and pseudo-code for recursive procedure.

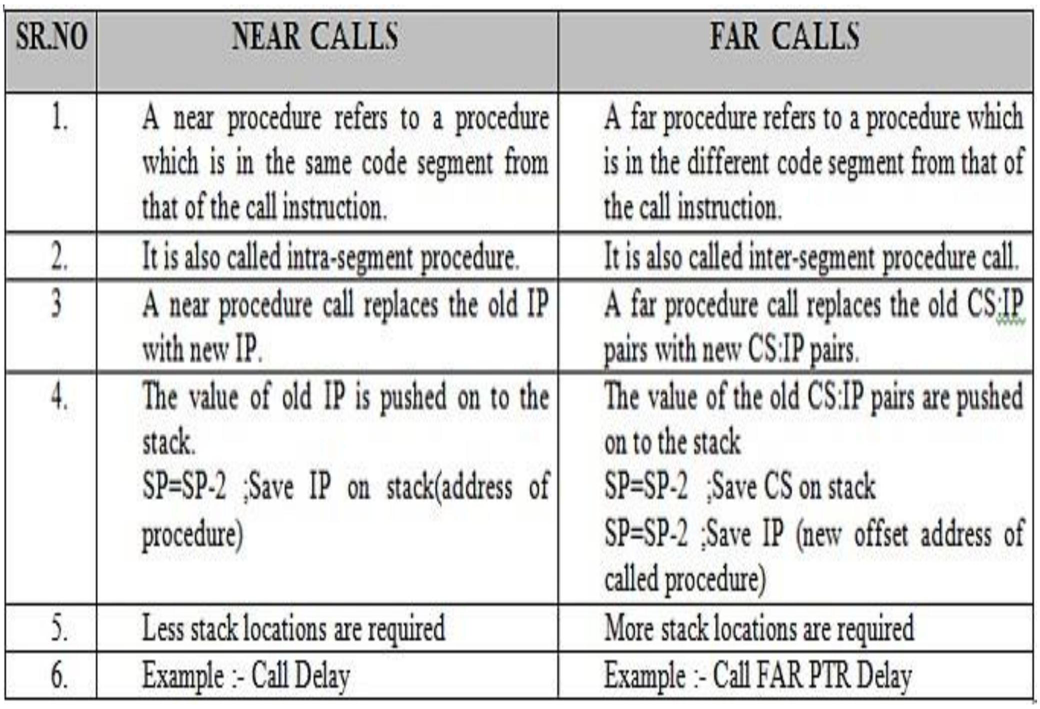


```
PROCEDURE RECURSIVE
IF N≠0
        DECREMENT N
      CALL RECURSIVE
ELSE
        RETURN
```

Fig. Flow diagram and pseudo-code for recursive procedure

27. Differentiate between NEAR and FAR CALLS.

Ans –



SR.NO	NEAR CALLS	FAR CALLS
1.	A near procedure refers to a procedure which is in the same code segment from that of the call instruction.	A far procedure refers to a procedure which is in the different code segment from that of the call instruction.
2.	It is also called intra-segment procedure.	It is also called inter-segment procedure call.
3	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.
4.	The value of old IP is pushed on to the stack. SP=SP-2 ;Save IP on stack(address of procedure)	The value of the old CS:IP pairs are pushed on to the stack SP=SP-2 ;Save CS on stack SP=SP-2 ;Save IP (new offset address of called procedure)
5.	Less stack locations are required	More stack locations are required
6.	Example :- Call Delay	Example :- Call FAR PTR Delay

28. Explain the concept of memory segmentation in 8086.

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.

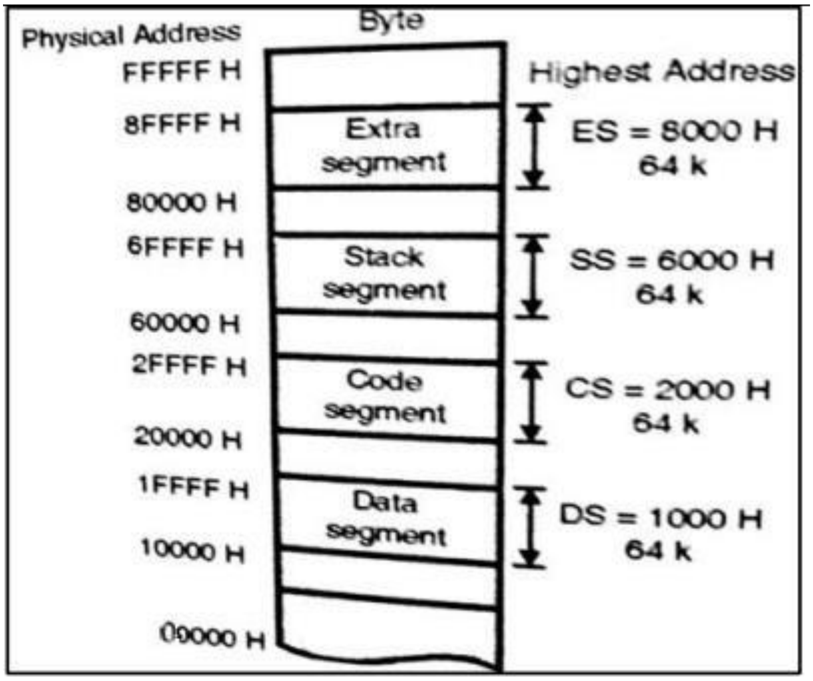
- Each segment is 64Kbytes & addressed by one segment register. i.e. CS, DS, ES or SS

- 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. Hence maximum size of any segment is 216=64K locations.

- Since the memory size of 8086 is 1Mbytes, total 16 segments are possible with each having 64Kbytes.

- The offset address values are from 0000H to FFFFH, so the physical address range from 00000H to FFFFFH.



29. State the Assembler Directives used in 8086 and describe the function of any two.

Ans –

Assembler directives:

1) DW
2) EQU

3) ASSUME
4) OFFSET
5) SEGMENT
6) EVEN

Function of any two:

1) DW (DEFINE WORD):
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, andinitialized with the value 437AH when the program is loaded into memory to be run.

2) EQU (EQUATE):
EQU is used to give a name to some value or symbol. Each time the assembler finds the given name in the program, it replaces the name with the value or symbol you equated with that name.

Example:
Data SEGMENT
Num1 EQU 50H
Num2 EQU 66H
Data ENDS
Numeric value 50H and 66H are assigned to Num1 and Num2.

30. Identify the Addressing Modes for the following instructions:

I. MOV CL, 34H
II. MOV BX, [4100H]
III. MOV DS, AX
IV. MOV AX, [SI+BX+04]

Ans –

I. MOV CL, 34H: Immediate addressing mode.
II. MOV BX, [4100H]: Direct addressing mode.
III. MOV DS, AX: Resister addressing mode.
IV. MOV AX, [SI+BX+04]: Relative Base Index addressing mode.

31. Explain the concept of pipelining in 8086 microprocessor with diagram.

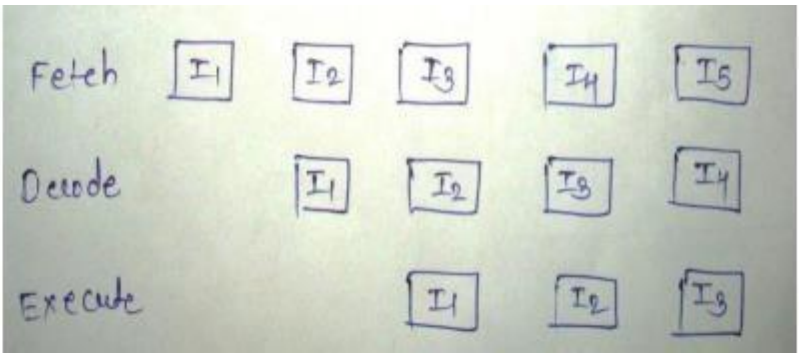
Ans –

- In 8086, pipelining is the technique of overlapping instruction fetch and execution mechanism.

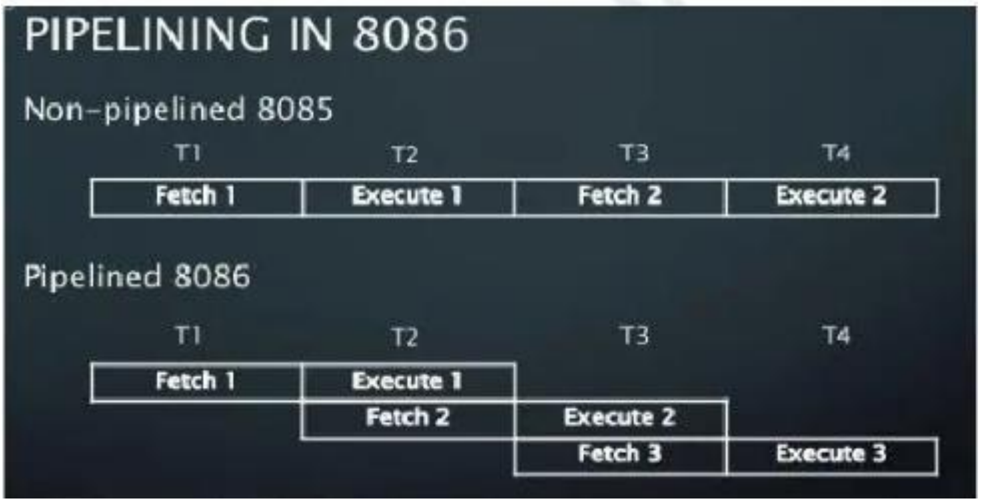
- To speed up program execution, the BIU fetches as many as six instruction bytes ahead of time from memory. The size of instruction prefetching queue in 8086 is6 bytes.

- 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 level deep FIFO. 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 fromthe queue in the BIU
- This is much faster than sending out an address to the system memory and waitingfor memory to send back the next instruction byte or bytes.
- This improves overall speed of the processor.



OR



PIPELINING IN 8086

Non-pipelined 8085

T1	T2	T3	T4
Fetch 1	Execute 1	Fetch 2	Execute 2

Pipelined 8086

T1	T2	T3	T4
Fetch 1	Execute 1		
	Fetch 2	Execute 2	
		Fetch 3	Execute 3

32. Write an alp to perform block transfer operation of 10 numbers
Ans –

WITHOUT STRING INSTRUCTION

```
 .MODEL SMALL

.DATA

 ARR1 DB 00H,01H,02H,03H,04H,05H,06,07H.08H.09H
```

```
ARR2  DB 10 DUP(00H)

ENDS

.CODE

START:

MOV AX, @DATA

MOV DS,AX

MOV SI, OFFSET ARR1

MOV DI, OFFSET ARR2

MOV CX ,0000A

BACK: MOV AL,[SI]

MOV [DI],AL

 INC SI

INC DI
LOOP BACK

MOV AH,4CH

INT  21H

ENDS

END START
```

OR

WITH STRING INSTRUCTION

```
.MODEL SMALL

.DATA

ARR1 DB 00H, 01H,02H,03H,04H,05H,06,07H.08H.09H

ARR2  DB 10 DUP(00H)

ENDS

.CODE

 START:MOV AX,@DATA

 MOV DS,AX

 MOV SI,OFFSET ARR1
MOV DI, OFFSET ARR2

MOV CX,0000A

REP MOVSB

MOV AH,4CH

INT 21H

ENDS

END START
```

33. Write an ALP to subtract two BCD number's.

Ans –

```
.MODEL SMALL

.DATA
NUM1 DB 86H
NUM2 DB 57H
 ENDS

.CODE

START:

MOV AX@,DATA

MOV  DS,AX

MOV AL,NUM1

SUB AL,NUM2

DAS

MOV BL,AL //    STORE FINAL RESULT IN BL REGISTER

MOV AH,4CH

 INT 21H

 ENDS

END START
```

34. Compare procedure and macros (4 points).

Ans –

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

35. Differentiate between minimum mode and maximum of 8086 microprocessor.

Ans –

Sr.No.	**Minimum Mode**	**Maximum Mode**
1	MN/MX’ pin is connected to Vcc. 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	HOLD and HLDA signals are available to interface another master in system such as DMA controller.	RQ / GTQ and RQ / GT 1 signals are available to interface another master in system such as DMA Controller and coprocessor 8087
7	This circuit is simpler	This circuit is complex

36. Write an ALP for sum of series of 05 number’s.

Ans –

```
.MODEL SMALL

.DATA

NUM1 DB 10H,20H,30H,40H,50H

RESULT DB 00H

CARRY DB 00H

 ENDS

.CODE
```

```
 START: MOV AX,@DATA
 MOV DS, AX
 MOV CL,05H
 MOV SI, OFFSET NUM1
 UP:MOV AL,[SI]
 ADD RESULT, AL
 JNC NEXT
 INC CARRY
 NEXT: INC SI
 LOOP UP
 MOV AH,4CH
 INT 21H
 ENDS
END START
```

37. Write an ALP to find largest number from array of 10 number’s.

Ans –

```
 .MODEL SMALL
.DATA
 ARRAY DB 02H,04H,06H,01H,05H,09H,0AH,0CH.00H,07H
  ENDS
  .CODE
 START: MOV AX,@DATA
 MOV DS,AX
 MOV CL,09H
 LEA SI,ARRAY
 MOV  AL,[SI]
 UP : INC SI
 CMP AL,[SI]
 JNC  NEXT
 MOV AL[SI]
 NEXT : DEC CL
 JNZ UP
  MOV AH,4CH
 INT  21H
  ENDS
END START
```

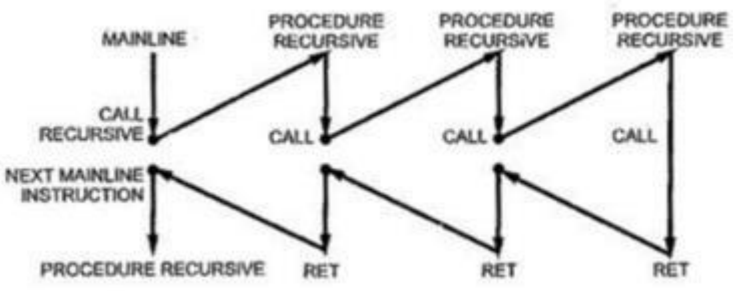
37. Describe re-entrant and Recursive procedure with diagram.

Ans -

A recursive procedure is procedure which calls itself. This results in the procedure call tobe generated from within the procedures again and again.

The recursive procedures keep on executing until the termination condition is reached.

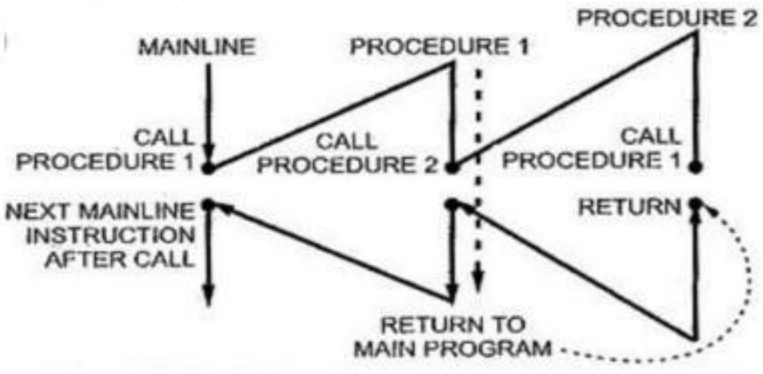
The recursive procedures are very effective to use and to implement but they take a large amount of stack space and the linking of the procedure within the procedure takes more time as well as puts extra load on the processor.



2) Re-entrant procedures:

In some situation it may happen that Procedure 1 is called from main program, Procrdure2 is called from procedure1And procedure1 is again called from procdure2. In this situation program execution flow re-enters in the procedure1. These types of procedures are called re-entrant procedures.

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



38. **Explain MACRO with suitable example. List four advantages of it.**

Ans –

- Macro is a small sequence of code of the same pattern, repeated frequently at different places, which perform the same operation on different data of the same data type
- The MACRO code is inserted into the program, wherever MACRO is called, bythe assembler
- Memory required is more, as the code is inserted at each MACRO call

Syntax: Macro_name MACRO [arg1,arg2,.... argN)

.....

endM

Example:

```
.MODEL SMALL

PROG MACRO A,B

MOV AL,A

MUL AL MOV

BL,ALMOV

AL,B MUL AL

ADD AL,BL

ENDM

.DATA

 X DB 02H Y

 DB  03H P DB

 DUP()

ENDS

 .CODE

START:

MOV AX,DATA

MOV DS,AX

PROG X, Y MOV

P,AL MOV

AH,4CH INT 21H
```

END START

ENDS

Advantages of Macro:

1) Program written with macro is more readable.

2) Macro can be called just writing by its name along with parameters, hence no extra code is required like CALL & RET.

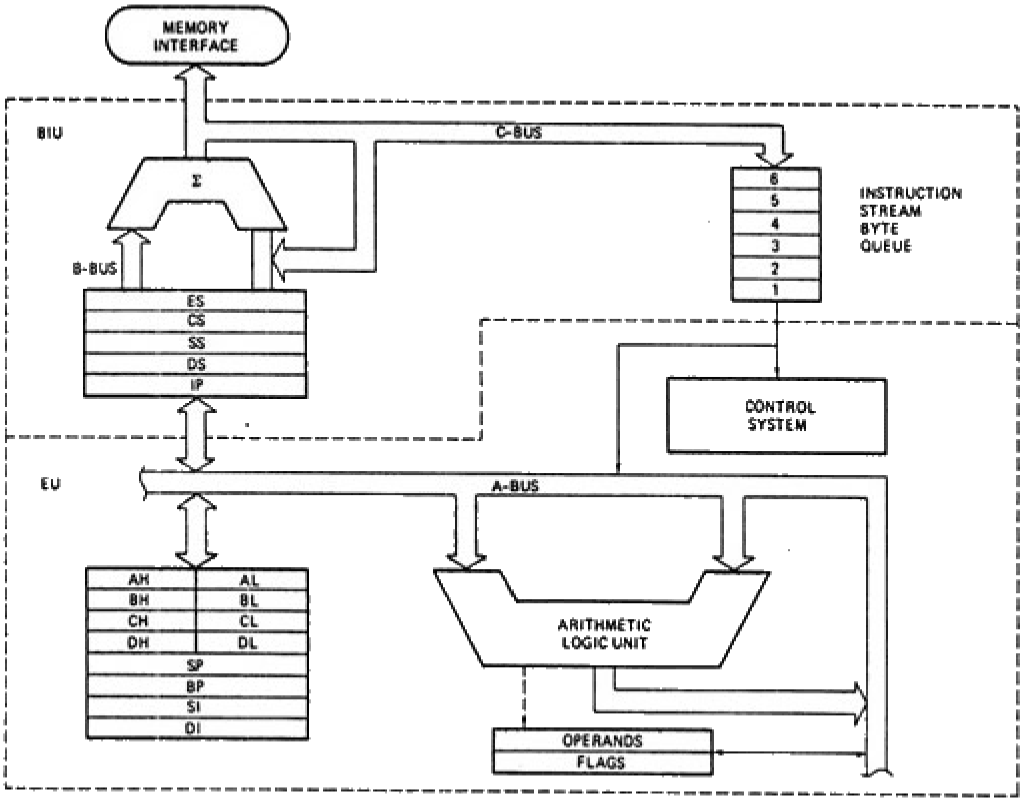
3) Execution time is less because of no linking and returning to main program.

4) Finding errors during debugging is easier.

6 Marks Questions

1. **Draw architectural block diagram of 8086 and describe its registerorganization.**

Ans –



Register Organization of 8086

1. **AX** (Accumulator) – Used to store the result for arithmetic / logicaloperations
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 handlesdivide operations)

5. **CS** – Code Segment – holds base address for all executable instructionsin 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 inextra 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 sourcestring is pointed to by the SI register.

12. **DI** – Destination Index – acts as the destination for string movementinstructions

13. **IP** – Instruction Pointer – contains the offset of the next instruction to beexecuted.

14. **Flag Register** – individual bit positions within register show status ofCPU or results of arithmetic operations.

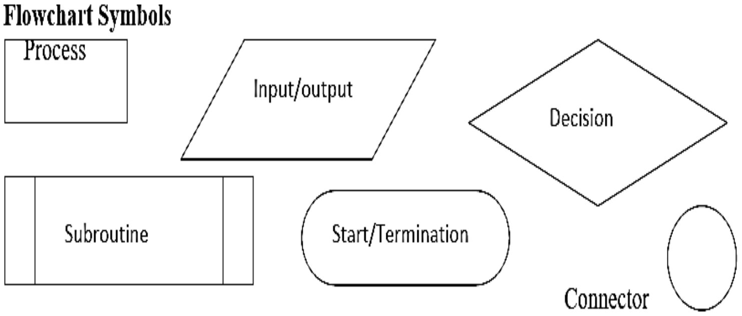
2. **Demonstrate in detail the program development steps in assemblylanguage programming.**

Ans –

<u>Program Development steps</u>

1. **Defining the problem**
   The first step in writing program is to think very carefully about the problem that you want the program to solve.
2. **Algorithm**
   The formula or sequence of operations or task need to perform by your program can be specified as a step in general English is called algorithm.

3. **Flowchart**
   The flowchart is a graphically representation of the program operation or task.



4. **Initialization checklist**
   Initialization task is to make the checklist of entire variables, constants, all the registers, flags and programmable ports.

5. **Choosing instructions**
   We should choose those instructions that make program smaller in size and more importantly efficient in execution.
6. **Converting algorithms to assembly language program**
   Every step in the algorithm is converted into program statement using correct and efficient instructions or group of instructions.

3. **Illustrate the use of any three branching instructions.**

Ans –

BRANCH INSTRUCTIONS

Branch instruction transfers the flow of execution of the program to a new address specified in the instruction directly or indirectly. When this type of instruction is executed, the CS and IP registers get loaded with new values of CS and IP corresponding to the location to be transferred.**Unconditional Branch Instructions :**

1. **CALL : Unconditional Call**
   The CALL instruction is used to transfer execution to a subprogram or procedure by storing return address on stack There are two types of calls- NEAR (Inter-segment) and FAR(Intra-segment call). Near call refers to aprocedure call which is in the same code segment as the call instruction and farcall refers to a procedure call which is in different code segment from that of the call instruction.

   Syntax: CALL procedure_name

2. **RET: Return from the Procedure.**

At the end of the procedure, the RET instruction must be executed. When it is executed, the previously stored content of IP and CS along with Flags are retrieved into the CS, IP and Flag registers from the stack and execution of the main program continues further.

Syntax :RET

3. **JMP: Unconditional Jump**

This instruction unconditionally transfers the control of execution to the specified address using an 8-bit or 16-bit displacement. No Flags are affected by this instruction.

Syntax : JMP Label

4. **IRET: Return from ISR**

When it is executed, the values of IP, CS and Flags are retrieved from the stack to continue the execution of the main program.

Syntax: IRET

Conditional Branch Instructions

When this instruction is executed, execution control is transferred to the address specified relatively in the instruction

1. **JZ/JE Label**

   Transfer execution control to address ‘Label’, if ZF=1.

2. **JNZ/JNE Label**

   Transfer execution control to address ‘Label’, if ZF=0

3. **JS Label**

   Transfer execution control to address ‘Label’, if SF=1.

4. **JNS Label**

Transfer execution control to address ‘Label’, if SF=0.

5. **JO Label**

Transfer execution control to address ‘Label’, if OF=1.

6. **JNO Label**

Transfer execution control to address ‘Label’, if OF=0.

7. **JNP Label**

Transfer execution control to address ‘Label’, if PF=0.

8. **JP Label**

Transfer execution control to address ‘Label’, if PF=1.

9. **JB Label**

Transfer execution control to address ‘Label’, if CF=1.

10. **JNB Label**

Transfer execution control to address ‘Label’, if CF=0.

11. **JCXZ Label**

Transfer execution control to address ‘Label’, if CX=0

Conditional LOOP Instructions.

12. LOOP Label :

Decrease CX, jump to label if CX not zero.

13.LOOPE label

Decrease CX, jump to label if CX not zero and Equal (ZF = 1).

14.LOOPZ label

Decrease CX, jump to label if CX not zero and ZF= 1.

15.LOOPNE label

Decrease CX, jump to label if CX not zero and Not Equal (ZF = 0).

16. LOOPNZ label

Decrease CX, jump to label if CX not zero and ZF=0

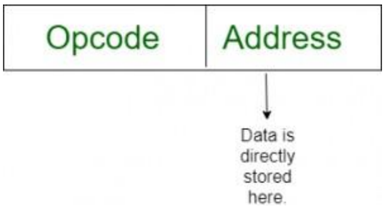
4. **Describe any six addressing modes of 8086 with suitable diagram.**

Ans –

<u>Different addressing modes of 8086 :</u>

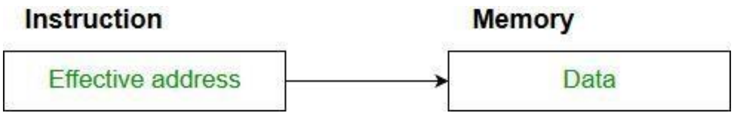
1. Immediate: In this addressing mode, immediate data is a part of instruction, and appears in the form of successive byte or bytes.

ex. MOV AX, 0050H



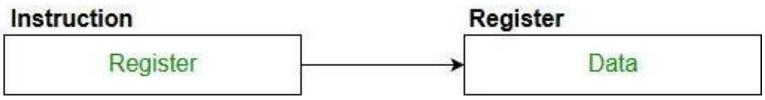
2. Direct: In the direct addressing mode, a 16 bit address (offset) is directly specified in the instruction as a part of it.

ex. MOV AX ,[1 0 0 0 H]



3. Register: In register addressing mode, the data is stored in a register and it is referred using the particular register. All the registers except IP may be used in this mode.

ex. 1)MOV AX,BX



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

e.g. MOV AX, □BX □

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

e.g. MOV AX, □SI □

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

e.g. MOV AX, 50H□BX □

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

e.g MOV AX, □BX □ □SI □

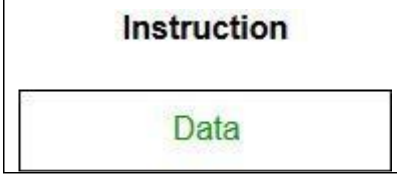
8. Relative Based Indexed: The effective address is formed by adding an 8-bit or 16-bit displacement with the sum of contents of any one of the base register (BX or BP) and any one of the index registers in a default segment.

e.g. MOV AX, 50H□BX □□SI □

9 .Implied addressing mode:

No address is required because the address is implied in the instruction itself.

e.g NOP,STC,CLI,CLD,STD



5. **Select an appropriate instruction for each of the following & write :i)Rotate the content of DX to write 2 times without carry ii)Multiply content of AX by 06H**

iii) Load 4000H in SP register

iv) Copy the contents of BX register to CSv)Signed division of BL and AL

vi) Rotate AX register to right through carry 3 times.

Ans –

i)

MOV CL,02HROR

DX,CL

(OR) ROR

DX,03H

ii)

MOV BX,06hMUL BX

iii)

MOV SP,4000H

iv)

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

v)

IDIV BL

vi)

MOV CL,03HRCR

AX,CL **(OR)**

RCR AX,03H

6. **Write an ALP to arrange numbers in array in descending order.**

Ans –

```
DATA SEGMENT
        ARRAY DB 15H,05H,08H,78H,56H
DATA ENDS
CODE SEGMENT
START:ASSUME CS:CODE,DS:DATA
        MOV DX,DATA
        MOV DS,DX
        MOV BL,05H

    STEP1: MOV SI,OFFSET ARRAY
        MOV CL,04H
    STEP: MOV AL,[SI]
        CMP AL,[SI+1]
        JNC DOWN

        XCHG AL,[SI+1]
        XCHG AL,[SI]

   DOWN:ADD SI,1
        LOOP STEP
        DEC BL
        JNZ STEP1
        MOV AH,4CH
        INT 21H
CODE ENDS
END START
```

7. Define logical and effective address. Describe physical address generation process in 8086. If DS=345AH and SI=13DCH. Calculatephysical address.

Ans –

A logical address is the address at which an item (memory cell, storage element) appears to reside from the perspective of an executing application program. A logical address may be different from the physical address due to the operation of an address translator or mapping function.

Effective Address or Offset Address: The offset for a memory operand is called the operand's effective address or EA. It is an unassigned 16 bit number that expresses the operand's distance in bytes from the beginning of the segment in which it resides. In 8086 we have base registers and index registers.

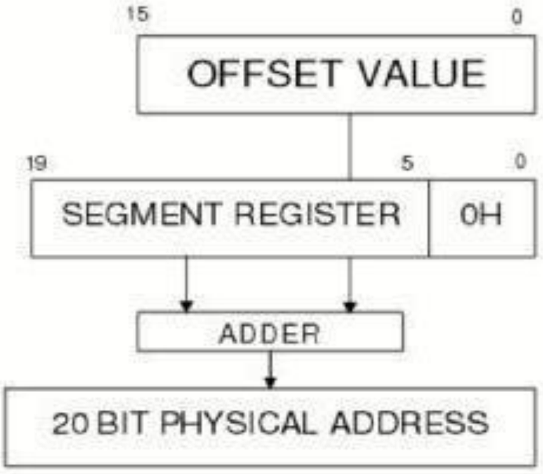
Generation of 20 bit physical address in 8086:-

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



DS=345AH and SI=13DCH
Physical adress = DS*10H + SI
= 345AH * 10H + 13DCH
= 345A0+13DC
= 3597CH

8.	**Explain the use of assembler directives. 1) DW 2) EQU 3) ASSUME 4) OFFSET 5) SEGMENT 6) EVEN**
Ans	**DW (DEFINE WORD)** 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, and initialized with the value 437AH when the program is loaded into memory to be run. **EQU (EQUATE)** EQU is used to give a name to some value or symbol. Each time the assembler finds the given name in the program, it replaces the name with the value or symbol you equated with that name.

Example

Data SEGMENT
Num1 EQU 50H
Num2 EQU 66H
Data ENDS

Numeric value 50H and 66H are assigned to Num1 and Num2.

ASSUME

ASSUME tells the assembler what names have been chosen for Code, Data Extra and Stack segments. Informs the assembler that the register CS is to be initialized with the address allotted by the loader to the label CODE and DS is similarly initialized with the address of label DATA.

OFFSET

OFFSET is an operator, which tells the assembler to determine the offset or displacement of a named data item (variable), a procedure from the start of the segment, which contains it.

Example

MOV BX;
OFFSET PRICES;

It will determine the offset of the variable PRICES from the start of the segment in which PRICES is defined and will load this value into BX.

SEGMENT

The SEGMENT directive is used to indicate the start of a logical segment. Preceding the SEGMENT directive is the name you want to give the segment.

For example, the statement CODE SEGMENT indicates to the assembler the start of a logical segment called CODE. The SEGMENT and ENDS directive are used to “bracket” a logical segment containing code of data

EVEN (ALIGN ON EVEN MEMORY ADDRESS)

As an assembler assembles a section of data declaration or instruction statements, it uses a location counter to keep track of how many bytes it is from the start of a segment at any time. The EVEN directive tells the assembler to increment the location counter to the next even address, if it is not already at an even address. A NOP instruction will be inserted in the location incremented over.

9. **Describe any four string instructions of 8086 assembly language.**

Ans **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**

	10.	**Describe any 6 addressing modes of 8086 with one example each.**
	Ans	**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 content 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:

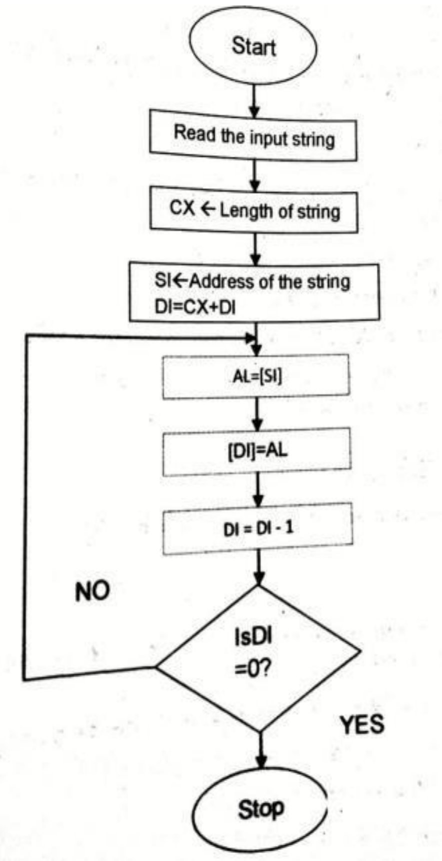
MOV AX, 50H [BX][SI]

	11.	**Select assembly language for each of the following** **i) rotate register BL right 4 times** **ii) multiply AL by 04H** **iii) Signed division of AX by BL** **iv) Move 2000h in BX register** **v) increment the counter of AX by 1** **vi) compare AX with BX**
	Ans	i) MOV CL, 04H RCL AX, CL1 Or MOV CL, 04H ROL AX, CL Or MOV CL, 04H RCR AX, CL1
		Or **MOV CL, 04H ROR AX, CL** **ii) MOV BL,04h** **MUL BL** **iii) IDIV BL** **iv) MOV BX,2000h** **v) INC AX** **vi) CMP AX,BX**

	12.	**Write an ALP to reverse a string. Also draw flowchart for same.**
	Ans	**Program:** DATA SEGMENT STRB DB 'GOOD MORNING$' REV DB 0FH DUP(?) DATA ENDS CODE SEGMENT START:ASSUME CS:CODE,DS:DATA MOV DX,DATA MOV DS,DX LEA SI,STRB MOV CL,0FH LEA DI,REV ADD DI,0FH UP:MOV AL,[SI]

```
 MOV [DI],AL
 INC SI
 DEC DI
 LOOP UP
 MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Flowchart:



13.	**Define logical and effective address. Describe Physical address generation in 8086. If CS = 2135 H and IP = 3478H, calculate Physical Address.**

Ans	**<u>A logical address</u>**: A logical address is the address at which an item (memory cell, storage element) appears to reside from the perspective of an executing application program. A logical address may be different from the physical address due to the operation of an address translator or mapping function. **<u>Effective Address or Offset Address</u>**: The offset for a memory operand is called the operand's effective address or EA. It is an unassigned 16-bit number that expresses the operand's distance in bytes from the beginning of the segment in which it resides. In 8086 we have base registers and index registers. <u>Procedure for Generation of 20-bit physical address in 8086: -</u> 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 bits offset. 4. Offset address is added into 20-bit base address which finally forms 20-bit physical address of memory location CS=2135H and IP=3475H Physical address = CS*10H + IP = 2135H * 10H + 3475H = 21350 + 3475 = 247C5H
14.	**Explain the following assembler directives:** **(i) DB (ii) DW (iii) EQU (iv) DUP (v) SEGMENT (vi) END**
Ans	**(i)** **<u>DB</u>** (Define Byte) – The DB directive is used to declare a BYTE -2-BYTE variable – A BYTE is made up of 8 bits. Declaration Examples:
	Byte1 DB 10h **Byte2 DB 255; 0FFh, the max. possible for a BYTE** **CRLF DB 0Dh, 0Ah, 24h; Carriage Return, terminator BYTE** **(ii) DW (Define Word): 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.** **Example, declares a variable of type word named MULTIPLIER, and initialized with the value 437AH when the program is loaded into memory to be run.**

(iii) EQU (EQUATE): EQU is used to give a name to some value or symbol. Each time the assembler finds the given name in the program, it replaces the name with the value or symbol you equated with that name.

Example -

Data SEGMENT Num1 EQU 50H Num2 EQU 66H

Data ENDS

Numeric value 50H and 66H are assigned to Num1 and Num2.

(iv) DUP: - It can be used to initialize several locations to zero.

e. g. SUM DW 4 DUP(0)

- Reserves four words starting at the offset sum in DS and initializes them to Zero.
- Also used to reserve several locations that need not be initialized. In this case (?) is used with DUP directives.

E. g. PRICE DB 100 DUP(?)

- Reserves 100 bytes of uninitialized data space to an offset PRICE.

(v) SEGMENT: - The SEGMENT directive is used to indicate the start of a logical segment. Preceding the SEGMENT directive is the name you want to give the segment. For example, the statement CODE SEGMENT indicates to the assembler the start of a logical segment called CODE. The SEGMENT and ENDS directive are used to “bracket” a logical segment containing code of data.

(vi) END: - An END directive ends the entire program and appears as the last statement. –

ENDS directive ends a segment and ENDP directive ends a procedure. END PROC-Name

15. Explain with suitable example the Instruction given below :

(i) DAA (ii) AAM

Ans

(i) DAA – Decimal Adjust after BCD Addition: When two BCD numbers are added, the DAA is used after ADD or ADC instruction to get correct answer in BCD.

Syntax- DAA (DAA is Decimal Adjust after BCD Addition)

Explanation: This instruction is used to make sure the result of adding two packed BCD numbers is adjusted to be a correct BCD number. The result of the addition must be in AL for DAA instruction to work correctly. If the lower nibble in AL after addition is > 9 or Auxiliary Carry Flag is set, then add 6 to lower nibble of AL. If the upper nibble in AL is > 9H or Carry Flag is set, and then add 6 to upper nibble of AL.

Example: - (Any Same Type of Example)

AL=99 BCD and BL=99 BCD

Then ADD AL, BL

1001 1001 = AL= 99 BCD +

1001 1001 = BL = 99 BCD

0011 0010 = AL =32 H

and CF=1, AF=1 After the execution of DAA instruction, the result is CF = 1 0011 0010 =AL =32 H AH =1 + 0110 0110 ------------------------ 1001 1000 =AL =98 inBCD

(ii) AAM - Adjust result of BCD Multiplication: This instruction is used after the multiplication of two unpacked BCD.

The AAM mnemonic stands for ASCII adjust for Multiplication or BCD Adjust after Multiply. This instruction is used in the process of multiplying two ASCII digits. The process begins with masking the upper 4 bits of each digit, leaving an unpacked BCD in each byte. These unpacked BCD digits are then multiplied and the AAM instruction is subsequently used to adjust the product to two unpacked BCD digits in AX.

AAM works only after the multiplication of two unpacked BCD bytes, and it works only on an operand in AL.

Example

Multiply 9 and 5

```
MOV AL, 00000101

MOV BH, 00001001
MUL BH                   ;Result stored in AX

                         ;AX = 00000000 00101101 = 2DH = 45 in decimals

AAM                      ;AX = 00000100 00000101 = 0405H = 45 in unpacked BCD

; If ASCII values are required an OR operation with 3030H can follow this step.
```

16. Write an appropriate 8086 instruction to perform following operations.

(i) Rotate the content of BX register towards right by 4 bits.
(ii) Rotate the content of AX towards left by 2bits.
(iii) Add 100H to the content of AX register.
(iv) Transfer 1234H to DX register.
(v) Multiply AL by 08 H.
(vi) Signed division of BL and AL

Ans-

1. Rotate the content of BX register towards right by 4 bits –

```
 MOV CL, 04H
 ROR BX, CL
```

2. Rotate the content of AX towards left by 2bits –

```
MOV CL, 02H
ROL AX, CL
```

3. Add 100H to the content of AX register –

```
ADD AX,0100H.
```

4. Transfer 1234H to DX register –

```
MOV DX,1234H
```

5. Multiply AL by 08H –

```
 MOV BL,08h
  MUL BL
```

Signed division of BL and ALIDIV BL

17. **Explain Addressing modes of 8086 with suitable example.**

Ans

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 content 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]

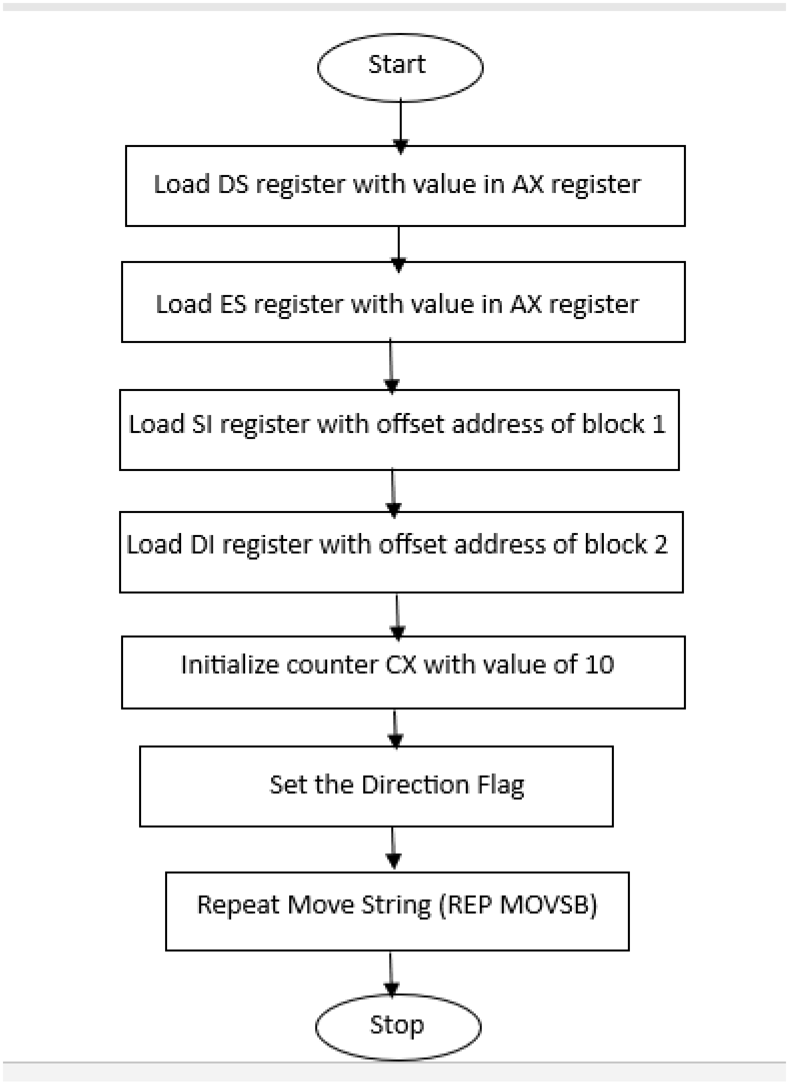
8. Relative Based Indexed addressing mode: An instruction in which the address of the operand is obtained by adding the displacement (8 bit or 16 bit) with the base registers (BX or BP) and index registers (SI or DI) to the default segment.
Example: MOV AX,50H [BX][SI]

18. **Write an ALP to transfer 10 bytes of data from one memory location to another, also draw the flow chart of the same.**

Ans

Data Block Transfer Using String Instruction

```
.MODEL SMALL
.DATA
BLOCK1 DB 01H,02H,03H,04H,05H,06H,07H,08H,09H,0AH
BLOCK2 DB 10(?)
ENDS

.CODE
MOV AX, @DATA
MOV DS, AX
MOV ES, AX

LEA SI, BLOCK1
LEA DI, BLOCK2

MOV CX, 000AH ; Initialize counter for 10 data elements

CLD
REP MOVSB

MOV AH, 4CH
INT 21H
ENDS
END
```


Start
Load DS register with value in AX register
Load ES register with value in AX register
Load SI register with offset address of block 1
Load DI register with offset address of block 2
Initialize counter CX with value of 10
Set the Direction Flag
Repeat Move String (REP MOVSB)
Stop

OR

Data Block Transfer Without String Instruction

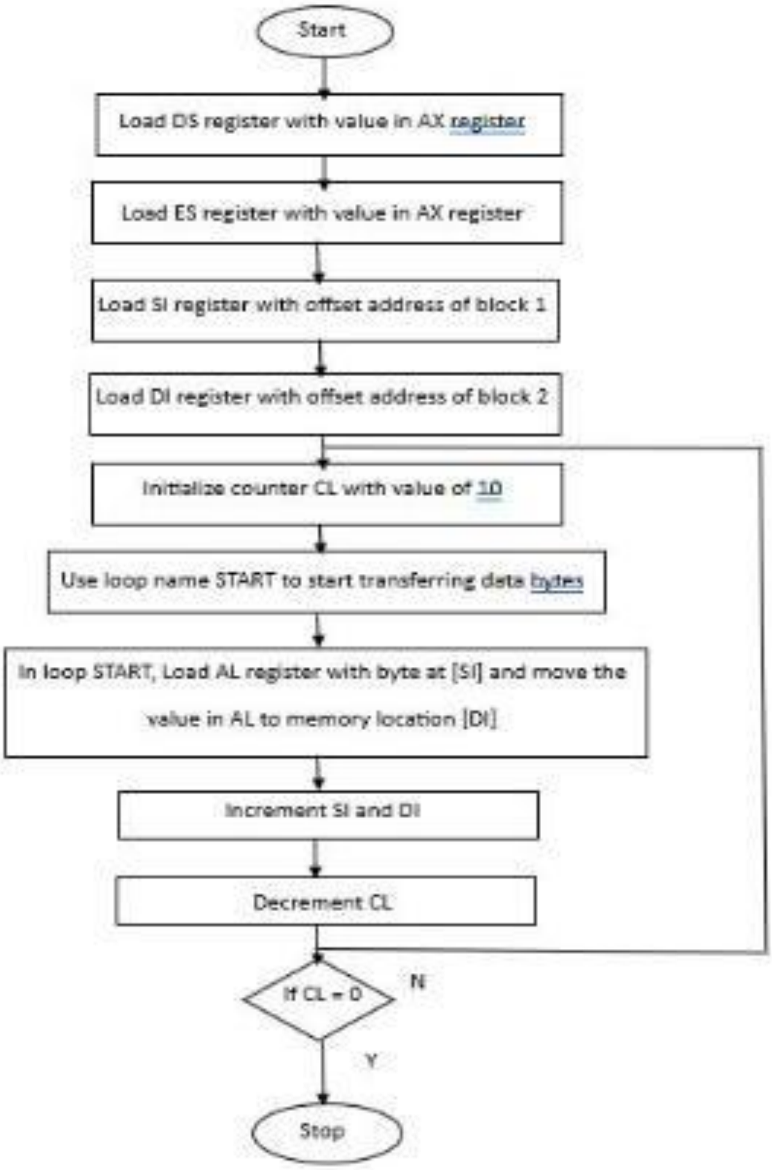
. Model small

. Data

ORG 2000H

Arr1 db 00h,01h,02h,03h,04h,05h,06h,07h,08h,09h

```
Count Equ 10 Dup

Org 3000H

Arr2 db 10 Dup(00h)

Ends


.code

Start: Mov ax,@data

Mov ds,ax

Mov SI,2000H

Mov DI,3000H

Mov cx, count

Back: Mov al, [SI]
Mov [DI], al

Inc SI

Inc DI

Dec cx

Jnc Back

Mov ah, 4ch
Int 21h

Ends
End
```


Start
Load DS register with value in AX register
Load ES register with value in AX register
Load SI register with offset address of block 1
Load DI register with offset address of block 2
Initialize counter CL with value of 10
Use loop name START to start transferring data bytes
In loop START, Load AL register with byte at [SI] and move the value in AL to memory location [DI]
Increment SI and DI
Decrement CL
If CL = 0
N
Y
Stop