MIC Practical

1.Identification and Explain various blocks in 8086 microprocessor architecture

ANS:=1. Bus Interface Unit (BIU)

The **BIU** handles all data and address transfers between the microprocessor and memory or I/O devices. It is responsible for fetching instructions, reading/writing data, and managing the address bus.

- **Components of BIU:**
  - **Instruction Queue (6 bytes):**
    - Prefetches instructions from memory to speed up execution.
    - Works like a pipeline to improve efficiency (fetch while execute).
  - **Segment Registers (CS, DS, SS, ES):**
    - Used to access different segments of memory.
    - Each is 16-bit, and when combined with an offset, gives a 20-bit physical address.
      - **CS**: Code Segment – holds the base address of the code.
      - **DS**: Data Segment – holds data variables.
      - **SS**: Stack Segment – used during stack operations.
      - **ES**: Extra Segment – used for additional data storage.
  - **Instruction Pointer (IP):**
    - Holds the offset of the next instruction within the code segment.

2. Execution Unit (EU)

The **EU** is responsible for decoding and executing instructions. It takes instructions from the BIU's instruction queue and processes them.

- **Components of EU:**
  - **ALU (Arithmetic Logic Unit):**
    - Performs arithmetic (add, subtract, etc.) and logic (AND, OR, NOT, etc.) operations.
  - **General Purpose Registers:**
    - **AX** (Accumulator), **BX**, **CX**, **DX** – each 16-bit, can be used as 8-bit registers too (AH, AL).
  - **Pointer and Index Registers:**
    - **SP (Stack Pointer)**, **BP (Base Pointer)**, **SI (Source Index)**, **DI (Destination Index)** – used in addressing and stack operations.
  - **Flag Register:**
    - Contains status flags (Zero, Carry, Sign, Overflow, etc.) that reflect results of operations and control execution.
  - **Instruction Decoder:**
    - Decodes the instruction fetched into signals the control unit and ALU can act upon.
  - **Control Unit:**
    - Sends control signals to various parts of the processor to carry out instructions

.

2.Use assembly language programming (ALP) tools and directives.

ANS:= Tools Used in ALP (for 8086):

1. Assembler (e.g., MASM, TASM):
    - Converts assembly code into machine code (object file).
2. Linker:
    - Combines object files into a single executable.
3. Debugger (e.g., DEBUG in DOS):
    - Used to test and debug assembly programs.
4. Editor:
    - Used to write the assembly code (Notepad, DOSBox editor, etc.).
5. Emulator/Simulator:
    - Simulates 8086 processor environment for testing code.

Common Directives in 8086 Assembly:

Directive
.MODEL
.DATA
.CODE
.STACK
DB
DW
END
PROC/ENDP
ASSUME

3. ALP to perform addition and subtraction of two given numbers.

ANS:= .MODEL SMALL

```
.STACK 100H

.DATA
   NUM1 DB 0AH
   NUM2 DB 05H
   SUM  DB ?
   DIFF DB ?
.CODE
MAIN PROC
   MOV AX, @DATA     ; Initialize data segment
   MOV DS, AX
```

```
    ; ---- Addition ----
    MOV AL, NUM1
    ADD AL, NUM2
    MOV SUM, AL
    ; ---- Subtraction ----
    MOV AL, NUM1
    SUB AL, NUM2
    MOV DIFF, AL

    MOV AH, 4CH
    INT 21H
MAIN ENDP
END MAIN
```

Algo:

Algorithm for Addition and Subtraction Program

1. **Start**
2. **Initialize data segment**
3. **Load first number (NUM1) into register AL**
4. **Add second number (NUM2) to AL**
5. **Store the result of addition in variable SUM**
6. **Load first number (NUM1) again into AL**
7. **Subtract second number (NUM2) from AL**
8. **Store the result of subtraction in variable DIFF**
9. **Terminate the program**

4. ALP for multiplication of two signed numbers

ANS:=

```
DATA SEGMENT
NUM1 DB -05H
```

```
NUM2 DB -04D RESULT DW ?
DATA ENDS
CODE SEGMENT
ASSUME DS: DATA, CS: CODE
START: MOV AX,DATA
        MOV DS, AX
       MOV AH, 00H
       MOV AL, NUM1
       MOV BL, NUM2
       IMUL BL
        MOV RESULT, AX
MOV AH, 09H
   INT 21H CODE
        ENDS END
        START
```

Algorithm:

1. **Start the program.**
2. **Initialize data segment:**
    a. NUM1 = -05H (hexadecimal, i.e., -5 in decimal)
    b. NUM2 = -04D (decimal, i.e., -4)
    c. RESULT is a 16-bit variable to store the output.
3. **Load the data segment address into AX and then into DS.**
4. **Clear AH (make AH = 00H) to prepare for signed multiplication.**
5. **Load NUM1 into AL.**
6. **Load NUM2 into BL.**
7. **Multiply AL and BL using signed multiplication (IMUL BL):**
    a. Since AL = -5 and BL = -4,
    b. Result = (-5) × (-4) = 20,
    c. Result is stored in AX (since AL is the accumulator).
8. **Store the result from AX into RESULT.**
9. **End the program using INT 21H with function 09H (though this interrupt is used for printing strings, and isn't necessary here).**
10. **Terminate the program.**

5: ALP for multiplication of two unsigned numbers.

ANS:= DATA SEGMENT

```
NUM1 DB 05H
NUM2 DB 04H RESULT DW ?
DATA ENDS
CODE SEGMENT
ASSUME DS: DATA, CS: CODE
START: MOV AX, DATA MOV DS,AX
```

```
        MOV AL,NUM1
        MOV BL,NUM2
        MUL BL
        MOV RESULT,AX
MOV AH, 09H
    INT 21H CODE
        ENDS END
        START
```

Algorithm for the Assembly Program:

1. **Start the program.**
2. **Initialize the Data Segment:**
    a. Load the address of the data segment into the AX register.
    b. Move the contents of AX into the DS register to access data.
3. **Load the numbers:**
    a. Move the value of NUM1 into register AL.
    b. Move the value of NUM2 into register BL.
4. **Multiply the numbers:**
    a. Multiply AL by BL (result goes into AX).
5. **Store the result:**
    a. Move the contents of AX into RESULT.
6. **Terminate the program:**
    a. Load 09H into AH to prepare for DOS interrupt.
    b. Call INT 21H (Note: This is typically used for displaying strings; here it doesn't affect the result storage).
7. **End the program.**

6: ALP to perform division of two signed numbers.

ANS:-

```
DATA SEGMENT

        NUM DW -0123H

        NUM1 DB -12H

                Q_RES DB ? R_RES DB ?

DATA ENDS

CODE SEGMENT

ASSUME DS: DATA,CS: CODE

START:MOV AX,DATA MOV DS,AX

        MOV AX,NUM

        MOV BL,NUM1
```

```
        IDIV BL
        MOV Q_RES, AL MOV R_RES,AH
        MOV AH, 09H
INT 21H
CODE ENDS
END START
```

Algorithm for the Program

1. **Start the program.**
2. **Initialize data segment:**
    a. Load the address of DATA segment into AX.
    b. Move AX into DS to initialize the data segment.
3. **Load the values:**
    a. Load the 16-bit signed number `NUM` into AX.
    b. Load the 8-bit signed number `NUM1` into BL.
4. **Signed Division:**
    a. Divide AX by BL using `IDIV BL`.
        i. Quotient is stored in AL.
        ii. Remainder is stored in AH.
5. **Store the result:**
    a. Store the quotient (AL) into `Q_RES`.
    b. Store the remainder (AH) into `R_RES`.
6. **Terminate (or pause) the program:**
    a. Set AH to 09H (DOS interrupt for printing string – but no string is defined, so this does nothing visible).
    b. Call interrupt 21H.
7. **End the program.**

7: ALP to perform division of two signed numbers.

ANS:-

```
DATA SEGMENT
        NUM DW 0000H
        NUM1 DW 0123H
        NUM2 DW 0012H RES DW ?
        RES1 DW ? DATA ENDS
CODE SEGMENT
ASSUME DS: DATA,CS: CODE
STAR:
        MOV AX,DATA MOV DS, AX
```

```
        MOV DX, NUM
        MOV AX, NUM1
        MOV BX, NUM2
        DIV BX
        MOV RES, AX ;quetient
        MOV RES1, DX ;remainder
MOV AH, 09H
   INT 21H
 CODE ENDS END START
```

Simple Algorithm:

1. **Start**
2. Define and initialize the data segment with variables:
    a. NUM1 = 0123H
    b. NUM2 = 0012H
    c. RES and RES1 are reserved to store the result.
3. Load the data segment into register DS.
4. Load NUM1 into register AX (dividend).
5. Load NUM2 into register BX (divisor).
6. Divide AX by BX.
    a. Quotient goes to AX
    b. Remainder goes to DX
7. Store the quotient (AX) in RES.
8. Store the remainder (DX) in RES1.
9. Call DOS interrupt to terminate the program or perform a dummy service (in this case INT 21H with AH=09H which expects a string, so it's not meaningful here).
10. **End**

8: ALP to perform addition and subtraction of two BCD numbers.

ANS:- data segment

num1 db 25h

num2 db 17h

sum db ?

diff db ?

```
data ends
code segment
Assume cs: Code, ds: data
mov ax, data
 mov ds, ax
 mov al, num1
 add al, num2
 daa mov sum, al
mov al, num1
sub al, num2
daa mov diff, al
mov ah, 4ch
 int 21h
Code ends
end start
```

Algorithm for the Given Assembly Program:

1. **Start the program.**
2. **Initialize the Data Segment:**
    a. Load the address of the `data` segment into AX.
    b. Move the value in AX into the DS register to initialize the data segment.
3. **Addition Operation:**

a. Load the value of num1 into register AL.
b. Add the value of num2 to AL.
c. Apply DAA (Decimal Adjust after Addition) to correct the result if treating numbers as BCD (Binary-Coded Decimal).
d. Store the result in the variable sum.

4. Subtraction Operation:

a. Load the value of num1 again into register AL.
b. Subtract the value of num2 from AL.
c. Apply DAA to adjust the result for BCD if needed.
d. Store the result in the variable diff.

5. Terminate the Program:

a. Move 4Ch into AH (DOS terminate function).
b. Call interrupt 21h to terminate the program.

9: ALP to find sum of series

ANS:

```
.model small
.stack 100h
.data
   N dw 10
   SUM dw 0
.code
main:
   mov ax, @data
   mov ds, ax

   mov cx, N
   mov ax, 0
   mov bx, 1

sum_loop:
```

```
    add ax, bx
    inc bx
    loop sum_loop
    mov SUM, ax

    ; Exit program
    mov ah, 4ch
    int 21h
end main
```

Algorithm:

1. **Start**
2. **Initialize**:
    a. `N = 10` (we want to sum numbers from 1 to 10)
    b. `SUM = 0`
3. **Set** a counter `BX = 1`
4. **Set** accumulator `AX = 0` to store the running sum
5. **Repeat** the following steps N times:
    a. Add BX to AX → `AX = AX + BX`
    b. Increment BX by 1 → `BX = BX + 1`
6. **Store** the final result from AX into `SUM`
7. **End the program**

10:

ALP to find smallest number from array of numbers

ANS:-

```
DATA SEGMENT
    ARRAY DW 12H,31H,02H,45H,65H
    SMALL DW 0 DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
 START:MOV AX,DATA MOV DS,AX
       MOV CX,05H
```

```
        MOV SI,OFFSET ARRAY
        MOV AX,[SI]
        DEC CX
    UP: INC SI
        INC SI
        CMP AX,[SI]
        JC NEXT
        MOV AX,[SI]
    NEXT:LOOP UP
        MOV SMALL,AX
MOV AH, 09H
    INT 21H
        CODE ENDS
        END START
```

Algorithm:

1. Load first array element into AX.
2. Set counter to 4 (remaining elements).
3. Loop through the rest of the array:
    a. Compare next element with AX.
    b. If smaller, update AX.
4. After loop, store AX into SMALL.

11: ALP to find largest number from array of numbers

ANS:-

```
DATA SEGMENT
    ARRAY DB 12H,31H,02H,45H,65H
    LARGE DB 0 DATA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA

 START:MOV AX,DATA
        MOV DS,AX
```

```
        MOV CX,05H
        MOV SI,OFFSET ARRAY
        MOV AL,[SI]
        DEC CX
    UP: INC SI
        CMP AL,[SI]
        JNC NEXT
        MOV AL,[SI]
    NEXT:LOOP UP
        MOV LARGE,AL
MOV AH, 09H
    INT 21H
        CODE ENDS
        END START
```

Simple Algorithm:

1. **Initialize Variables:**
    a. Start with the first element of the array as the smallest value.
2. **Check All Elements:**
    a. Go through each element of the array one by one.
    b. If any element is smaller than the current smallest value, update the smallest value to this element.
3. **Store the Result:**
    a. Once all elements have been checked, store the smallest value in LARGE.
4. **Display the Result:**
    a. Show the smallest value stored in LARGE.
5. **End Program.**

12: ALP to arrange numbers in an array in ascending order.

ANS:- DATA SEGMENT
ARRAY DB 12H,11H,21H,9H,19H
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA

START:MOV AX,DATA MOV DS,AX

```
        MOV BX,05H
    UP1:MOV SI,OFFSET ARRAY
        MOV CX,04H
     UP:MOV AL,[SI] CMP
        AL,[SI+1]
        JC DN
        XCHG
        AL,[SI+1]
        XCHG AL,[SI]
DN:INC SI
        LOOP UP
        DEC BX
        JNZ UP1
MOV AH, 09H
    INT 21H
        CODE ENDS
        END START
```

Algorithm for Sorting Array Using Bubble Sort:

1. **Initialize** an array of numbers:
   a. ARRAY = [12, 11, 21, 9, 19]
2. **Repeat** the sorting process for n-1 passes, where n is the number of elements in the array (5 in this case).
3. **For each pass** (from 1 to n-1):
   a. Compare each adjacent pair of numbers in the array.
   b. If the current number is greater than the next, **swap** them.
4. **End** when all passes are completed.
5. **Output** the sorted array.

13: ALP to find the length of string.

ANS:- .model small
.stack 100h
.data
    str db 'Hello, World!$'
    len db 0

.code
main:
    mov ax, @data
    mov ds, ax

    lea si, str
    mov cx, 0

find_len:
    mov al, [si]
    cmp al, '$'
    je done_len
    inc si
    inc cx
    jmp find_len
done_len:
    mov len, cl

    ; Exit program
    mov ah, 4ch
    int 21h
end main

Algorithm:

1. **Start the Program.**
2. **Initialize Registers:**
    a. Set up the `ds` register to point to the data segment.
    b. Set the `si` register to the address of the string `str`.
    c. Set the `cx` register to 0 (counter for the string length).
3. **Find the Length of the String:**
    a. Repeat the following steps until the end of the string is found:
        i. Read the current character from `str` using `si`.
        ii. If the character is not `$`, increment the `si` register (move to the next character) and increment `cx`.
        iii. If the character is `$`, stop the loop.
4. **Store the Length:**
    a. The value in `cx` (the string length) is stored in the variable `len`.
5. **Exit the Program.**

14: ALP to arrange numbers in an array in descending order.

ANS:

```
DATA SEGMENT
ARRAY DB 12H,11H,21H,9H,19H
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE,DS:DATA

START:MOV AX,DATA
      MOV DS,AX
      MOV BX,05H
```

```
   UP1:MOV SI,OFFSET ARRAY MOV CX,04H
    UP:MOV AL,[SI] CMP AL,[SI+1]
       JNC DN
       XCHG
       AL,[SI+1]
       XCHG AL,[SI]
DN:INC SI
       LOOP UP
       DEC BX
       JNZ UP1
MOV AH, 09H
   INT 21H
       CODE ENDS
       END START
```

Simple Algorithm to Sort Elements in Descending Order

1. **Start** with an array of elements: ARRAY = [12H, 11H, 21H, 9H, 19H].
2. **Compare** each pair of adjacent elements in the array:
    a. If the current element is **smaller** than the next element, **swap** them.
3. **Repeat** the process for every element in the array until no more swaps are needed.
4. **Stop** when the array is sorted in descending order.

15: ALP to perform concatenation of two strings.

ANS:-

```
DATA SEGMENT
STR1 DB 'ABCD$'
STR2 DB 'PQRS$'
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START: MOV DX,DATA
       MOV DS, DX LEA SI,STR1
```

```
    LEA
    DI,STR2
    MOV AL,'$'
UP: CMP AL,[SI]
    JZ NEXT
    INC SI
    JMP UP
NEXT: CMP AL,[DI] JZ EXIT
    MOV BL,[DI]
    MOV [SI], BL
    INC SI
    INC DI
    JMP
    NEXT

EXIT: MOV [DI],AL
  MOV AH,4CH INT 21H
    CODE ENDS
    END START
```

Simple Algorithm for String Concatenation

1. **Start** with two strings STR1 and STR2, both ending with a $ symbol.
2. **Find the end of STR1**:
    a. Start at the first character of STR1.
    b. Move through STR1 until you reach the $ symbol (this marks the end of STR1).
3. **Append STR2 to STR1**:
    a. After finding the $ in STR1, start copying characters from STR2 to STR1 (right after the $ symbol).
    b. Keep copying each character of STR2 until you reach its $ symbol.
4. **Terminate the concatenated string**:
    a. After copying all characters from STR2, add a $ symbol at the end to mark the end of the concatenated string.
5. **End the program**.

16: ALP for string operations such as string reverse

ANS:- DATA SEGMENT

```
 STR1 DB 'ABCDEF$'
STR2 DB 10 DUP('$')
DATA ENDS

 CODE SEGMENT
 ASSUME CS:CODE, DS:DATA
 START: MOV DX,DATA MOV DS, DX
        LEA SI,STR1
        MOV CL,06H LEA DI,STR2
        ADD DI,05H MOV AL,'$'
      UP: MOV AL,[SI]
          MOV [DI],AL
          DEC DI
          INC SI
          LOOP UP
         MOV AH,4CH INT 21H
           CODE ENDS
           END START
```

Simple Algorithm for Reversing a String:

1. **Define the Strings:**
    a. Have two strings:
        i. STR1 with the value "ABCDEF".
        ii. STR2 as an empty string to store the reversed result.
2. **Initialize Pointers:**
    a. Set a pointer to the start of STR1.
    b. Set another pointer to the end of STR2.
3. **Loop Through the Characters:**
    a. For each character in STR1:
        i. Take the character from STR1.
        ii. Place it in STR2 at the current end position.
        iii. Move the pointer in STR1 forward and the pointer in STR2 backward.
4. **Stop When All Characters are Reversed:**

a. Repeat the process until all characters from STR1 are moved into STR2 in reverse order.

5. **End the Program.**

17: ALP for string operations such as string copy

ANS: .model small

```
.stack 100h
.data
    src db 'Hello, 8086!$',
    dest db 20 dup('$')

.code
main:
    mov ax, @data
    mov ds, ax
    mov es, ax

    lea si, src
    lea di, dest

copy_loop:
    lodsb
    stosb
    cmp al, '$'
```

Simple Algorithm for String Copy:

1. **Initialize the pointers:**
    a. Set the source string pointer (`SI`) to the start of the source string.
    b. Set the destination string pointer (`DI`) to the start of the destination string.
2. **Copy each character:**

a. Copy one character at a time from the source string to the destination string.
b. Check if the copied character is the end of the string (`$`), which marks the end of the string.

3. **Stop when end of string is reached:**
   a. If the character is `$`, stop copying.

Simple Pseudocode:

1. Set `SI` to the source string.
2. Set `DI` to the destination string.
3. Repeat until the character at `SI` is `$`:
   a. Copy the character at `SI` to `DI`.
4. End.

18: ALP to check a given number is odd or even.

ANS:

```
DATA SEGMENT
NO1 DB 13H
MS1 DB 10,13, "NUMBER IS ODD$"
MS2 DB 10,13, "NUMBER IS EVEN$" DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA
START: MOV DX,DATA
        MOV DS, DX
        MOV
        AL,NO1
        MOV  AH,00H  MOV
        BH,01H       MOV
        BL,02H
        DIV BL
        CMP AH,BH
        JE NEXT
        LEA DX,MS2
        MOV AH,09H
        INT 21H
NEXT:LEA DX,MS1
        MOV AH,09H
        INT 21H
         MOV AH,4CH
          INT 21H
              CODE ENDS
```

```
END START
```

Simple Algorithm:

1. **Start**
2. **Store the Number**: Store the number you want to check (in this case, it's NO1).
3. **Divide the Number by 2**:
   a. Divide the number by 2.
   b. Get the remainder.
4. **Check the Remainder**:
   a. If the remainder is **0**, the number is **even**.
   b. If the remainder is **not 0**, the number is **odd**.
5. **Display the Result**:
   a. If the number is even, display "NUMBER IS EVEN".
   b. If the number is odd, display "NUMBER IS ODD".
6. **End**.

19: ALP to count number of' ‘0' and '1's in a given number.

Ans:

```
DATA SEGMENT
NO1 DB 13H ZERO DB ?
ONE DB ?

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE, DS:DATA

START: MOV DX,DATA

    MOV DS, DX

  MOV AL,NO1

  MOV BH,00H

  MOV BL,00H

  MOV CL,08H

  UP:RCL AL,01H

  JC NEXT

    INC BL

      LOOP UP

  NEXT: INC BH

      LOOP UP
```

```
EXIT:MOV ZERO, BL
     MOV ONE, BH

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

Simple Algorithm:

1. **Start the Program**: Initialize the data segment and load the number to be checked into a register (AL).
2. **Initialize Counters**:
    a. Set two counters: BL for counting '0's and BH for counting '1's.
3. **Check Each Bit**:
    a. Repeat for each of the 8 bits:
        i. Look at the least significant bit (LSB) of the number.
        ii. If the LSB is 1, increment the BH counter (count of '1's).
        iii. If the LSB is 0, increment the BL counter (count of '0's).
        iv. Shift the number right by one bit (to move the next bit into position).
4. **Store the Results**:
    a. After checking all 8 bits, store the counts of '0's in BL and the counts of '1's in BH.
5. **Exit the Program**: End the program.

20: ALP to perform arithmetic operations on given numbers using procedure.

Ans:

```
DATA SEGMENT
   NUM1 DB 5
  NUM2 DB 3
   RESULT DB ?
   OPERATION DB 'A'
DATA ENDS

CODE SEGMENT
ASSUME CS:CODE, DS:DATA
```

```
START:
    MOV DX, DATA
    MOV DS, DX
      MOV AL, OPERATION
    CALL PROCESS_OPERATION


    MOV AH, 4Ch
    INT 21H

PROCESS_OPERATION:
    CMP AL, 'A'
    JE ADDITION

    CMP AL, 'S'
    JE SUBTRACTION

    CMP AL, 'M'
    JE MULTIPLICATION

    CMP AL, 'D'
    JE DIVISION
    RET

ADDITION:
    MOV AL, NUM1
    ADD AL, NUM2
    MOV RESULT, AL
   RET

SUBTRACTION:
    MOV AL, NUM1
    SUB AL, NUM2
    MOV RESULT, AL
    RET

MULTIPLICATION:
    MOV AL, NUM1
    MOV BL, NUM2
```

```
    MUL BL
    MOV RESULT, AL
    RET

DIVISION:
    MOV AL, NUM1
    MOV BL, NUM2
    DIV BL
    MOV RESULT, AL
    RET

CODE ENDS
END START
```

Algorithm: Arithmetic Operations Using Procedures

1. **Start the program**.
2. **Initialize the Data Segment**:
    a. Define two numbers: `NUM1` and `NUM2`.
    b. Define a variable `RESULT` to store the final result.
    c. Define a variable `OPERATION` to specify the type of arithmetic operation ('A', 'S', 'M', or 'D').
3. **Load the Data Segment** into the DS register.
4. **Load the Operation Type** (`OPERATION`) into a register (e.g., `AL`).
5. **Call the Main Procedure** `PROCESS_OPERATION`:
    a. Check the operation type:
        i. If operation = `'A'` → call the `ADDITION` procedure.
        ii. If operation = `'S'` → call the `SUBTRACTION` procedure.
        iii. If operation = `'M'` → call the `MULTIPLICATION` procedure.
        iv. If operation = `'D'` → call the `DIVISION` procedure.
6. **Each Procedure Does the Following**:
    a. **ADDITION**: `RESULT = NUM1 + NUM2`
    b. **SUBTRACTION**: `RESULT = NUM1 - NUM2`
    c. **MULTIPLICATION**: `RESULT = NUM1 * NUM2`
    d. **DIVISION**: `RESULT = NUM1 / NUM2` (stores the **quotient**)
7. **Store the Final Result** in the `RESULT` variable.
8. **Exit the Program** using DOS interrupt `INT 21H`.