Week #1 Lab # 1

Introduction to Assembly Language Programming. Assembler, Linker, Directives, Data types and etc.

SYNTAX OF 8086ASSEMBLY LANGUAGE

- The language is not case sensitive.
- There may be only one statement per line. A statement may start in any column.
- A statement is either an instruction, which the assembler translates into machine code, or an assembler directive (pseudo-op), which instructs the assembler to perform some specific task.

- **Syntax of a statement:**

  {name} mnemonic {operand(s)} {; comment}

  (a) The curly brackets indicate those items that are not present or are optional in some statements.
  (b) The name field is used for instruction labels, procedure names, segment names, macro names, names of variables, and names of constants.
  (c) MASM 6.1 accepts identifier names up to 247 characters long. All characters are significant, whereas under MASM 5.1, names are significant to 31 characters only. Names may consist of letters, digits, and the following 6 special characters: **? . @ _ $ %** .If a period is used; it must be the first character. Names may not begin with a digit.
  (d) Instruction mnemonics, directive mnemonics, register names, operator names and other words are reserved.

- **Syntax of an instruction:**

  {label:} mnemonic {operand { , operand} } {; comment}

  The curly brackets indicate those items that are not present or are optional in some instructions. Thus an instruction may have zero, one, or two operands.

- **Operators:**

  The 8086/8088 Assembly language has a number of operators. An operator acts on an operand or operands to produce a value at assembly time. Examples are: + , - , *, / , DUP, and OFFSET

- **Comments:**

A semicolon starts a comment. A comment may follow a statement or it may be on a separate line. Multiple-line comments can be written by using the COMMENT directive. The syntax is:

COMMENT **delimiter** {comment}

comment

. . .

delimiter { comment }

where delimiter is any non-blank character not appearing in comment. The curly brackets indicate an item that is optional.

e.g.,

```
COMMENT *
    This program finds
    the maximum element in a byte array
*
```

❖ **Numbers:**

(a) A binary number is suffixed by b or B.

e.g., 11010111B

(b) A decimal number is suffixed by an optional d or D.

e.g., 42d -22D 3578

(c) A hexadecimal number must begin with a decimal digit and it is suffixed by h or H

e.g., 20H 0bF2Ah

Characters:

A character is enclosed in a pair of single quotes or in a pair of double quotes.

e.g., 'x' "B"

❖ **Strings:**

A string is enclosed in a pair of single quotes or in a pair of double quotes.

e.g., ‘ENTER YOUR NAME: ’

“THE MAXIMUM VALUE IS ”

‘Omar shouted, “help !” ’

“say, ‘hello’ ”

“Omar’s books”

For a string delimited by single quotes, a pair of consecutive single quotes stands for a single quote.

e.g., ‘Omar’ ’s books’

❖ **Data definition**

Each variable has a data type and is assigned a memory address by the program. The data-defining directives are:

Directive	Description of Initializers
BYTE, **DB** (byte)	Allocates unsigned numbers from 0 to 255.
SBYTE (signed byte)	Allocates signed numbers from –128 to +127.
WORD, DW (word = 2 bytes)	Allocates unsigned numbers from 0 to 65,535 (64K).
SWORD (signed word)	Allocates signed numbers from –32,768 to +32,767.
DWORD, DD (doubleword = 4 bytes),	Allocates unsigned numbers from 0 to 4,294,967,295 (4 megabytes).
SDWORD (signed doubleword)	Allocates signed numbers from –2,147,483,648 to +2,147,483,647.

e.g., ALPHA DB 4

VAR1 DB ?

ARRAY1 DB 40H, 35H, 60H, 30H

VAR2 DW 3AB4h

ARRAY2 DW 500, 456, 700, 400, 600

PROMPT DB 'ENTER YOUR NAME $'

POINTER1 DD 6BA7000AH

A **?** in place of an initializer indicates you do not require the assembler to initialize the variable. The assembler allocates the space but does not write in it. Use **?** for buffer areas or variables your program will initialize at run time.

```
integer     BYTE   16
negint      SBYTE  -16
expression  WORD   4*3
signedexp   SWORD  4*3
empty       QWORD  ?        ; Allocate uninitialized long int
            BYTE   1,2,3,4,5,6 ; Initialize six unnamed bytes
long        DWORD  4294967295

longnum     SDWORD -2147433648
```

The DUP operator can be used to generate multiple bytes or words with known as well as un-initialized values.

e.g., table dw 100 DUP(0)

stars db 50 dup('*')

ARRAY3 DB 30 DUP(?)

```
ARRAY4    DB   10   DUP(50), 45, 22, 20  DUP(60)

STRINGS   DB    20H  DUP('Dhahran')
```

Note: If a variable name is missing in a data definition statement, memory is allocated; but no name is associated with that memory. For example:

```
DB  50  DUP(?)
```

allocates 50 un-initialized bytes; but no name is associated with those 50 bytes.

In MASM 6.1 and obove, a comma at the end of a data definition line (except in the comment field) implies that the line continues. For example, the following code is legal in MASM 6.1:

```
longstring    BYTE    "This string ",
              "continues over two lines."
bitmasks      BYTE    80h, 40h, 20h, 10h,
              08h, 04h, 02h, 01h
```

- **<u>Named constants</u>:**

  The EQU (equate) directive, whose syntax is:

  ```
  name   EQU   constant_expression
  ```

  assigns a name to a constant expression. Example:

  ```
  MAX   EQU   32767

  MIN    EQU   MAX  -  10

  LF      EQU   0AH

  PROMPT   EQU   'TYPE YOUR NAME: $'
  ```

  Note: (i) No memory is allocated for EQU names

  (ii) A name defined by EQU may not be redefined later in a program.

- The LABEL directive, whose syntax is:

```
name   LABEL   type
```

where type (for MASM Version 5.1 and lower versions) is BYTE, WORD, DWORD, QWORD, TBYTE, NEAR, or FAR provides a way to define or redefine the type associated with a variable or a label.

Example1:

```
ARRAY1   LABEL    WORD

ARRAY2    DB   100   DUP(0)
```

Here ARRAY1 defines a 50-word array, and ARRAY2 defines a 100-byte array. The same memory locations are assigned to both arrays. Thus the array can be accessed as either the byte array ARRAY1 or the word array ARRAY2.

Example2:

```
VAR3  LABEL  DWORD

WORD1   LABEL   WORD

BYTE1  DB  ?

BYTE2  DB  ?

WORD2   LABEL    WORD

BYTE3  DB  50H

BYTE4  DB  66H
```

in this example, each of the words, and each of the bytes of the double word variable VAR3 can be accessed individually.

Week #2 Lab # 2

Program Segment; Segment directives; Program model Size; Assembling and executing program

❖ SEGMENT DEFINITION

An 8086/8088 assembly language program file must have the extension **.asm**

There are two types of 8086/8088 assembly language programs: exe-format and com-format.

An exe-format program generates executable files with extension **.exe**. A com-format program generates executable files with extension **.com** .

An exe-format program must contain a code segment and a stack segment. It may contain a data segment or an extra segment.

A com-format program contains only the code segment (the stack segment is explicit).

A programmer chooses an appropriate size for the stack segment, depending on the size of his program. Values in the range 100H to 400H are sufficient for most small programs.

Note: In a program, the data, code, and stack segments may appear in any order. However, to avoid forward references it is better to put the data segment before the code segment.

❖ SIMPLIFIED SEGMENT DIRECTIVES

MASM version 5.0 and above, and TASM provide a simplified set of directives for declaring segments called simplified segment directives. To use these directives, you must initialize a memory model, using the **.**MODEL directive, before declaring any segment. The format of the **.**MODEL directive is:

```
.MODEL  memory-model
```

The memory-model may be TINY, SMALL, MEDIUM, COMPACT, LARGE, HUGE or FLAT :

memory-model	description
TINY	One segment. Thus code and data together may not be greater than 64K
SMALL	One code-segment. One data-segment. Thus neither code nor data may be greater than 64K
MEDIUM	More than one code-segment. One data-segment. Thus code may be greater than 64K
COMPACT	One code-segment. More than one data-segment. Thus data may be greater than 64K
LARGE	More than one code-segment. More than one data-segment. No array larger than 64K. Thus both code and data may be greater than 64K
HUGE	More than one code-segment. More than one data-segment. Arrays may be larger than 64K. Thus both code and data may be greater than 64K
FLAT	One segment up to 4GB. All data and code (including system resources) are in a single 32-bit segment.

All of the program models except TINY result in the creation of exe-format programs. The TINY model creates com-format programs.

Memory Model	**Operating System**	Data and Code Combined
Tiny	MS-DOS	Yes
Small	MS-DOS, Windows	No
Medium	MS-DOS, Windows	No
Compact	MS-DOS, Windows	No

Large	MS-DOS, Windows	No
Huge	MS-DOS, Windows	No
Flat	Windows NT	Yes

The simplified segment directives are: **.**CODE , **.**DATA , **.**STACK .

The **.**CODE directive may be followed by the name of the code segment.

The **.**STACK directive may be followed by the size of the stack segment, by default the size is 1K i.e., 1,024 bytes.

The definition of a segment extends from a simplified segment directive up to another simplified segment directive or up to the END directive if the defined segment is the last one.

ASSEMBLING AND EXECUTING THE PROGRAM

Writing an ALP

Assembly level programs generally abbreviated as ALP are written in text editor EDIT.

Type *EDIT* in front of the command prompt to open an untitled text file.

EDIT<file name>

After typing the program save the file with appropriate file name with an extension *.ASM*

Ex:

Add.ASM

Assembling an ALP

To assumble an ALP we needed executable file called MASM.EXE. Only if this file is in current working directory we can assemble the program. The command is

MASM<filename.ASM>

If the program is free from all syntactical errors, this command will give the **OBJEC**T file.In

case of errors it list out the number of errors, warnings and kind of error.
Note:No object file is created until all errors are rectified.

Linking

After successful assembling of the program we have to link it to get **Executable file.**
The command is
LINK<File name.OBJ>
This command results in *<Filename.exe>* which can be executed in front of the command prompt.

Executing the Program

Open the program in debugger by the command(note only exe files can be open)by the command.
CV <Filename.exe>
This will open the program in debugger screen where in you can view the assemble code with the CS and IP values at the left most side and the machine code. Register content ,memory content also be viewed using **VIEW** option of the debugger.
Execute option in the menu in the menu can be used to execute the program either in single steps(F7) or burst execution(F5).

i)Byte and word data transfer in different addressing modes

```
DATA SEGMENT
DATA1 DB 23H
DATA2 DW 1234H
DATA3 DB 0H
DATA4 DW 0H
DATA5 DW 2345H, 6789H
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX, DATA; Initialize DS to point to start of the memory
MOV DS, AX; set aside for storing of data
MOV AL, 25H; copy 25H into 8 bit AL registers
MOV AX, 2345H; copy 2345H into 16 bit AX register
MOV BX, AX; copy the content of AX into BX register (16 bit)
MOV CL, AL; copy the content of AL into CL register
MOV AL, DATA1; copies the byte contents of data segment memory
                ; Location DATA1 into 8 bit AL
MOV AX, DATA2; copies the word contents of data segment memory
                 ; location DATA2 into 16 bit AX
MOV DATA3, AL; copies the AL content into the byte contents of data
                ;segment memory location DATA3
MOV DATA4, AX; copies the AX content into the word contents of
                ;data segment memory location DATA4
MOV BX, OFFSET DATA5; The 16 bit offset address of DS memory location
; DATA5 is copied into BX
MOV AX,[BX] ; copies the word content of data segment
              ; memory location addressed by BX into
              ;AX(register indirect addressing)
MOV DI, 02H; address element
MOV AX,[BX+DI} ; copies the word content of data segment
```

```
                    ;memory location addressed by BX+DI into
                    ; AX(base plus indirect addressing)
MOV AX,[BX+0002H] ; copies the word content of data segment(16 bit)
MOV AL, [DI+2]      ; register relative addressing
MOV AX,[BX+DI+0002H] ;copies the word content of data segment
                    ;  memory location addressed by BX+DI+000 into AX(16 bit)
MOV AH, 4CH; Exit to DOS with function call 4CH
INT 21H
CODE ENDS; Assembler stop reading
END START
```

ii) Block Interchange

```
DATA SEGMENT
X DB 01H,02H,03H,04H,05H
Y DB 11H,12H,13H,14H,15H
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS, AX
MOV CX, 05H; Load the counter
LEA SI, X; SI pointed to the source location x
LEA DI, Y; DI pointed to the destination location y
UP: MOV BL,[SI] ; Move the SI content to BL register
MOV AL,[DI] ; Move the DI content to AL register
MOV [SI], AL ; Move AL register content to content of SI
MOV [DI], BL ; Move BL register content to content of DI
INC SI ; Update SI and DI
INC DI
DEC CX; Decrement the counter till it becomes zero
JNZ UP
MOV AH, 4CH       INT 21H      CODE ENDS
```

Week #4 Lab # 04
Programs involving Logical Operation; AND, OR, NOT XOR, TEST.

Write following codes and perform indicated operations.

Program 1:

```
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
MOV BX, 3256H
MOV CX, 1554H
AND CX, BX
HLT
CODE ENDS
END
```

Observe content of CX register. What operation happened here?

Program 2:

```
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
MOV BX, 3256H
MOV CX, 1554H
XOR CX, BX
HLT
CODE ENDS
END
```

Observe content of CX register. What operation happened here?

Program 3:

```
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
MOV AX, 1027H
MOV BX, 5A27H
MOV CX, 54A5H
OR AX, BX
XOR AX, CX
NOT AX
TEST CX, BX
AND CX, AX
HLT
CODE ENDS
END
```

Perform this operation in single step mode and write the values of registers for every step. Obtain binary values for upper hexadecimal values and perform bit by bit operation for every step. Compare your hand calculation with obtained result.

Part 2:

Write following codes and perform indicated operations.

Program 1:

```
CODE SEGMENT
ASSUME CS: CODE, DS: CODE
MOV AX, 7A24H
```

```
MOV BX, 15A3H
SUB AX, BX
JMP L3T2
EEE316: DIV BX
JMP Last
L3T2: MOV CX, 45B1H
AND AX, CX
TEST AX, BX
JMP EEE316
Last: HLT
CODE ENDS
END
```

Perform this operation in single step mode and write the values of registers for every step.

Program 1:

```
CODE SEGMENT
ASSUME CS:CODE, DS:CODE
MOV AX, 7A24H
MOV BX, 95A3H
ADD AX, BX
JC L3T2
EEE316: OR AX, 23H
JNZ Last
```

```
L3T2: MOV CX, 0FC7H
SUB AX,CX
JZ EEE316
Last: HLT
CODE ENDS
END
```

Week #5 Lab # 05

Programs involving Arithmetic Operations like Addition and subtraction 8bit addition & 8 bit subtraction; 16 bit addition & 16 bit subtraction and 32 bit addition & 32 bit subtraction.

i) 16 Bit Addition

```
DATA SEGMENT
NUM DW 1234H, 0F234H
SUM  DW 2 DUP(0)
DATA ENDS
```

```
CODE SEGMENT
ASSUME CS: CODE, DS: DATA
START: MOV AX, DATA
MOV DS, AX
MOV AX, NUM; First number loaded into AX
MOV BX, 0H; For carry BX register is cleared
ADD AX, NUM+2; Second number added with AX
JNC DOWN; Check for carry
INC BX; If carry generated increment the BX
DOWN: MOV SUM, AX; Storing the sum value
MOV SUM+2,BX ; Storing the carry value
MOV AH, 4CH
INT 21H
CODE ENDS
END START
```

INPUT : 1234H, F234H

OUTPUT : 10468H

ii) 32 Bit addition

```
DATA SEGMENT
NUM1 DW 0FFFFH,0FFFFH
NUM2 DW 1111H,1111H
SUM DW 4 DUP(0)
dATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
MOV AX,NUM1      ;Move LSB of NUM1 to AX
ADD AX,NUM2      ;Add LSB of NUM2 to AX
```

```
MOV SUM,AX          ;Store the LSB in SUM
MOV AX,NUM1+2   ; Move MSB of NUM1 to AX
ADC AX,NUM2+2    ; Add MSB of NUM2 to AX
JNC EXIT              ; Check for carry
MOV SUM+4,01H   ; Store the carry in SUM+4
EXIT: MOV SUM+2,AX ; Store the MSB in SUM+2
        MOV AH,4CH
        INT 21H
        CODE ENDS
        END START
        INPUT: 0FFFFFFFFH, 011111111H
        OUTPUT: 0111111110H
```

iii) 32 Bit addition using DD directive

```
DATA SEGMENT
NUM1 DW 12345678H
NUM2 DW 12345678H
SUM DW 3 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START:MOV AX,DATA
MOV DS,AX
LEA SI,NUM1 ; SI pointed to the of LSB of NUM1
LEA DI,NUM2 ; DI pointed to the of LSB of NUM2
```

```
MOV AX,[SI] ; Move the content of SI to AX
ADD AX,[DI]
MOV CX,[SI+2]
; Add DI content to AX
; Move the SI to point MSB of NUM1 and move that
;content to CX
ADC CX,[DI+2] ; Move the DI to point MSB of NUM2 and add
;with carry to CX
JNC DOWN ; Check for carry
MOV SUM+4,01H ; Store the carry in SUM+4
DOWN:MOV SUM,AX ; Store the LSB in SUM
MOV SUM+2,CX ; Store the MSB in SUM+2
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

INOUT: 12345678H,12345678H

OUTPUT:2468ACF0H

v) 16 Bit Subtraction

```
DATA SEGMENT
NUM DW 4567H,2345H
DIF DW 1 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME
CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
CLC
; Clearing Carry
```

```
LEA SI,NUM ; SI pointed to the NUM
MOV AX,[SI] ; Move NUM1 to AX
SBB AX,[SI+2] ; Move the SI to Num2 and subtract with AX(Takes
;care for both smaller as well as larger
;Number subtraction)
MOV DIF,AX ;Store the result
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

INPUT: 4567H,2345H

OUTPUT:2222

v) 32 Bit Subtraction

```
DATA SEGMENT
NUM1 DW 2345H,6762H
NUM2 DW 1111H,1111H
DIF DW 2 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
LEA SI,NUM1      ; SI pointed to the LSB of NUM1
LEA DI,NUM2      ; DI pointed to the LSB of NUM2
```

```
MOV AX,[SI]        ; Move the content of SI to AX
MOV BX,[DI]        ; Move the content of DI to BX
SUB AX,BX          ; Subtract from BX to AX
MOV DIF,AX         ; Store the LSB result in DIF
INC SI        ;Update SI to point the MSB of NUM1(if
INC SI        ;ADD SI,02 instruction its affect carry flag)
INC DI      ;Update DI to point the MSB of NUM2
INC DI
MOV AX,[SI]     ; Move the content of SI to AX
MOV BX,[DI]      ; Move the content of DI to BX
SBB AX,BX        ;  Subtract with borrow from BX to AX
MOV DIF+2,AX   ; Store the MSB result in DIF+2
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

INPUT: 23456762,-11111111 INPUT:11111111,-23451234

OUTPUT:12345651 OUTPUT:EDCBFEDD

Week # Lab # 06

Multiplication and division of unsigned and signed hexadecimal numbers: 16 bit multiplication for unsigned number; 16 bit multiplication for signed number; 8 bit division for unsigned number; 8 bit division for signed number; 16 bit division for unsigned number; 16 bit division for signed number.

16 Bit multiplication for signed numbers

```
DATA SEGMENT
NUM DW 1234H,1234H
PROD DW 2 DUP(0)
DATA ENDS
```

```
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
LEA SI,NUM ; SI pointed to the Multiplicand
MOV AX,[SI] ; Multiplicand has to be in AX register
MOV BX,[SI+2] ; SI+2 pointed to the Multiplier and move it to BX
MUL BX ;Perform the multiplication
MOV PROD,AX ;32 bit product stored in DX-AX registers
MOV PROD+2,DX
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

INPUT: Multiplicand- 1234H,

Multiplier - 1234H

OUTPUT: DX-01 4B

AX-54 90

16 Bit multiplication for signed numbers

```
DATA SEGMENT
NUM DW -2,1
PROD DW 2 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
LEA SI,NUM ; SI pointed to the Multiplicand
MOV AX,[SI] ; Multiplicand has to be in AX register
MOV BX,[SI+2] ; SI+2 pointed to the Multiplier and move it to BX
```

```
IMUL BX ; Perform the sign multiplication using sign
        ;Multiplication operator (IMUL)
MOV PROD,AX ; 32 bit product stored in DX-AX registers
MOV PROD+2,DX
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

INPUT: Multiplicand- -2,

Multiplier - 1

OUTPUT: DX – FF FF

AX – FF FE ; Result is in two complement form.

8 Bit Division for Unsigned numbers

```
DATA SEGMENT
NUM1 DB 72H,
NUM2 DB 02H
QUO DB 1 DUP(0)
REM DB 1 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
```

```
MOV AL,NUM1 ;Move the Dividend to AL
MOV AH,0H        ; Zero extended for 16 bit/8 bit division
DIV NUM2         ; Perform the Division operation
MOV QUO,AL ; Store the quotient to AL
MOV REM,AH ;Store the reminder to AH
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

INPUT: Dividend - 72H,
Divisor - 02 H,
0UTPUT: AL - 39H (quotient);
AX - 00H (reminder);
INPUT: Dividend - 55H,
Divisor - 04 H,
0UTPUT: AL - 15H (quotient);
AX - 01H (reminder);

8 Bit Division for Signed numbers

```
DATA SEGMENT
NUM1 DB -10
NUM2 DB 02
QUO DB 1 DUP(0)
REM DB 1 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
```

```
MOV AL,NUM1 ;Move the Dividend to AL
CBW
IDIV NUM2 ; Perform the Sign Division operation using IDIV operator
MOV QUO,AL ; Store the quotient to AL
MOV REM,AH ;Store the reminder to AH
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

INPUT: Dividend - -10

Divisor - 02

0UTPUT: AL – FBH (quotient) ; Result is in two complement form

INPUT: Dividend - -10

Divisor - 03

0UTPUT: AL – FDH (quotient);

AX – FF H (reminder) ; Result is in two complement form

16 Bit Division for Unsigned numbers

```
DATA SEGMENT
NUM1 DW 4567H,2345H
NUM2 DW 4111H
QUO DW 2 DUP(0)
REM DW 1 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
```

```
MOV DS,AX
MOV AX,NUM1 ;Move the lower bit of Dividend to AX
MOV DX,NUM1+2   ; Move the higher bit of Dividend to DX
DIV NUM2        ; Perform the Division operation
MOV QUO,AX      ; Store the quotient to AX
MOV REM,DX      ; Store the reminder to DX
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

INPUT: Dividend - 23454567,
Divisor - 4111,
0UTPUT: AX – 8AC5H (quotient);
DX – 0952H (reminder);

16 Bit Division for Signed numbers

```
DATA SEGMENT
NUM1 DW 4567H,2345H
NUM2 DW 4111H
QUO DW 2 DUP(0)
REM DW 1 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
```

```
MOV AX,NUM1 ; Move the lower bit of Dividend to AX
MOV DX,NUM1+2   ; Move the higher bit of Dividend to DX
CWD
IDIV NUM2       ; Perform the sign Division operation using IDIV
MOV QUO,AX    ; Store the quotient to AX
MOV REM,DX     ; Store the reminder to DX
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

INPUT: Dividend - -44444444,

Divisor - 2222,

0UTPUT: AX – FE (quotient);

DX – FF (reminder) ; Result is in two complement form.

Week #7 Lab # 07
Arithmetic programs to find squire, cube and factorial.

Program to find square and cube of a number

```
DATA SEGMENT
X DW 04H
SQUARE DW ?
CUBE DW ?
DATA ENDS
CODE SEGMENT
```

```
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA    ;Load the Data to AX.
MOV DS,AX          ;Move the Data AX to DS.
MOV AX,X           ;Move the X number Data to AX.
MOV BX,X           ;Move the X number Data to BX.
MUL BX             ;Perform the multiplication by BX.
MOV SQUARE,AX   ;Store value in SQUARE.
MUL BX             ;Perform the multiplication by BX.
MOV CUBE,AX        ;Store value in CUBE.
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Input: x ---------- 4h

Output: Square ---------10h

Program to find factorial of a given number

```
DATA SEGMENT

X DW 06H

FACT DW ?

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA

START: MOV AX,DATA
```

```
MOV DS,AX

MOV AX,01H     ;Set the value of AX as 01H.

MOV CX,X       ;Move the i/p number to CX.

UP: MUL CX     ;Perform the Loop multiplication operation.

LOOP UP

MOV FACT,AX    ;Store the FACT value.

MOV AH,4CH

INT 21H

CODE ENDS

END START
```

Input: 06

Output: 2D0H

Week #8 Lab # 08
Arithmetic programs to find LCM and GCD.

Program to find LCM of a given number

```
DATA SEGMENT
NUM DW 05,04
LCM DW 2 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA     ;Load the Data to AX.
MOV DS,AX              ;Move the Data AX to DS.
MOV DX,0H              ;Initialize the DX.
MOV AX,NUM             ;Move the first number to AX.
MOV BX,NUM+2           ;Move the second number to BX.
UP: PUSH AX            ;Store the quotient/first number in AX.
```

```
PUSH DX                    ;Store the remainder value in DX.
DIV BX                     ;Divide the first number by second number.
DX,0                       ;Compare the remainder.
JE EXIT                    ;If remainder is zero, go to EXIT label.
                           ;If remainder is non-zero,
POP DX                     ;  Retrieve the remainder.
POP AX                     ;Retrieve the quotient.
ADD AX,NUM                 ;Add first number with AX.
JNC DOWN                   ;If no carry jump to DOWN label.
INC DX                     ;Increment DX.
DOWN: JMP UP               ;Jump to Up label.
EXIT: POP LCM+2            ;If remainder is zero, store the value at LCM+2.
POP LCM
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Input: 0A, 04
Output: 02

Program to find GCD of two numbers

```
DATA SEGMENT
NUM1 DW 000AH
NUM2 DW 0004H
GCD DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA  ;Load the Data to AX.
MOV DS,AX           ;Move the Data AX to DS.
MOV AX,NUM1         ;Move the first number to AX.
MOV BX,NUM2         ;Move the second number to BX.
UP: CMP AX,BX       ;Compare the two numbers.
```

```
JE EXIT                 ;If first number is below than second,
                        ;If equal, go to EXIT label.
JB EXCG                 ;go to EXCG label.
UP1:MOV DX,0H           ;Initialize the DX.
DIV BX                  ;Divide the first number by second number.
CMP DX,0                ;Compare remainder is zero or not.
JE EXIT                 ;If zero, jump to EXIT label.
MOV AX,DX               ;If non-zero, move remainder to AX.
JMP UP                  ;Jump to UP label.
EXCG:XCHG AX,BX         ;Exchange the remainder and quotient.
JMP UP1                 ;Jump to UP1.
EXIT:MOV GCD,BX         ;Store the result in GCD.
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Input: 0A,04

Output: 02

Week #9 Lab # 09

Programs involving bit manipulation instructions: If given data is positive or negative and If given data is odd or even.

If given data is positive or negative

```
DATA SEGMENT
NUM DB 12H
MES1 DB 10,13,'DATA IS POSITIVE $'
MES2 DB 10,13,'DATA IS NEGATIVE $'
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
```

```
MOV DS,AX
MOV AL,NUM    ;Move the Number to AL.
ROL AL,1      ;Perform the rotate left side for 1 bit position.
JC NEGA       ;Check for the negative number.
MOV DX,OFFSET MES1    ;Declare it positive.
JMP EXIT      ;Exit program.
NEGA: MOV DX,OFFSET MES2;  Declare it negative.
EXIT: MOV AH,09H
INT 21H
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Output: Data is positive

Positive Numbers: 00-7F

Negative numbers: 80-FF

If given data is odd or even

```
DATA SEGMENT
X DW 27H
MSG1 DB 19,13,'NUMBER IS EVEN$'
MSG2 DB 10,13,'NUMBER IS ODD$'
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
MOV AX,X
TEST AX, 01H   ;Test for Even/Odd number.

JNZ EXIT       ;If it is Even go to Exit label.
```

```
                    ;(alternate logic)
                    ;MOV BL,2
                    ;DIV BL
                    ;CMP AH,0H
                    ;JNZ EXIT
LEA DX,MSG1       ;Declare it is Even number.
MOV AH,09H
INT 21H
JMP LAST
EXIT: LEA DX,MSG2  ;Declare it is Odd number.
MOV AH,09H
INT 21H
LAST: MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Output: Number is ODD

Week #10 Lab # 10

Code Conversion: ASCII Adjustment instructions, Binary to BCD code conversion and BCD to Binary code conversion.

ASCII adjustment instructions

```
CODE SEGMENT
ASSUME CS:CODE
START: MOV AX,31H  ;Load ASCII 1
ADD AX,39H          ;Load ASCII 9
AAA                 ;ASCII Adjust, AX=0100 UNPACKED BCD
ADD AX,3030H        ;Answer in ASCII
MOV BL,9            ;Load divisor
MOV AX,0702H        ;Load dividend, AAD instruction requires
                    ;Ax register to contain a two digit unpacked
                    ;BCD number before executing
```

```
AAD                 ;AAD appears before division
DIV BL              ;Contents of adjusted AX register is devised
                    ;by an unpacked BCD number to generate
                    ;a single digit result in AL with any remainder in AH
MOV AL, 5           ;Load multiplicand
MOV CL ,5           ;Load multiplier
MUL CL              ;AX=0019H
AAM                 ;AX=0205(Unpacked BCD)
ADD AX,3030H        ;AX=3235H
MOV AX,38H          ;Load ASCII 8
SUB AX,31H          ;Load ASCII 1
AAS                 ;AX=0007
AX,3030H            ;AX=3037H
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Binary to BCD code conversion

```
DATA SEGMENT
BIN DW 01A9H
BCD DB 2 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA  ;Load the Data to AX.
MOV DS,AX           ;Move the Data AX to DS.
 MOV AX,BIN         ;Move the Binary Data to AX.
MOV CL,64H          ;100 in decimal
DIV CL              ;Perform the division by 100.
MOV BCD+1,AL        ;Store the quotient in BCD+1.
MOV AL,AH           ;Move the Reminder value to AL.
```

```
MOV AH,00H          ;Initialize the AH.
MOV CL,0AH          ;10 in decimal.
DIV CL              ;Perform the division by 10.
MOV CL,04           ;Perform the Right side rotation 4 times.
ROR AL,CL           ;Adding the Reminder in LSB.
ADD AL,AH
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Input: binary--------01A9

Output: bcd----------425

-

BCD to Binary code conversion

```
DATA SEGMENT
BCD DW 27H
BIN DW ?
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA    ;Load the Data to AX.
MOV DS,AX           ;Move the Data AX to DS.
MOV AX,BCD          ;Move the BCD Data to AX.
AND AX,07H          ;Perform the AND operation between 07H and input BCD
MOV BX,AX           ;Move data AX to BX
MOV AX,BCD          ;Move the BCD Data to AX.
AND AX,0F0H         ;Perform the AND with 0F0H for shifting operation.
```

```
MOV CX,0AH      ;10 in decimal.
MUL CX          ;Perform the multiplication by 10.
ADD AX,BX       ;Perform the addition operation to get the LSB.
MOV BIN,AX      ;Move the result to binary.
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Input: BCD----------27

Output:----------1B

Week #11 Lab # 11
Programs involving Branch/ looping instruction/ programs on array: Addition of N numbers; Program to subtract N numbers.

i)ADDITION OF n NUMBERS

```
DATA SEGMENT    ;start of data segment
ARR DW 0010H,0020H,0030H,0040H,0050H
LEN EQU ($-ARR)/2
SUM  DW  ?
DATA ENDS       ;end of data segment
CODE SEGMENT    ;start of code segment
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX       ;initialize data segment
```

```
LEA SI,ARR          ;SI points to the LSB of data ARR
CLC                 ;clear carry
XOR AX,AX           ;clear AX register
MOV CX,LEN          ;load CX with the number of data words in ARR
UP: ADC AX,[SI]     ;add the number pointed by SI to A register
INC SI              ;point to the next data word
INC SI
DEC CX              ;decrement Cx
JNZ UP              ;and check if all numbers are added if no then add
MOV SUM,AX          ;store the addition result in user defined memory location sum
MOV AH,4CH          ;terminate the process
INT 21H
CODE ENDS           ;end of code segment
END START
```

OUTPUT: 00F0

ii)PROGRAM TO SUBTRACT n NUMBERS

```
DATA SEGMENT                ;start of data segment
ARR DW 50H,10H,20H,10H,05H
LEN EQU ($-ARR)/2
DIF DW ?
DATA ENDS                   ;end of data segment
CODE SEGMENT                ;start of code segment
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX                   ;initialize data segment
LEA SI,ARR                  ;SI points to the LSB of data ARR
CLC                         ;clear carry flag
MOV CX,LEN-1                ;load CX register with the number of data words in ARR
MOV AX,[SI]                 ;make a copy of the first number pointed by SI in AX
```

```
UP:
 SUB AX,[SI+2];subtract the next number from the contents of AX and store the result in AX
INC SI              ;point to the next number
INC SI
DEC CX              ;decrement CX
JNZ UP              ;and check if all subtraction of all numbers is complete if no then subtract
MOV DIF,AX          ;store the difference in user defined memory location DIFF
MOV AH,4CH          ;terminate the process
INT 21H
CODE ENDS           ;end of code segment
END START
```

OUTPUT: 0005

Week #12 Lab # 12

Program to find largest and Smallest number; Program to find Largest number among series; Program to find Smallest number among series.

PROGRAM TO FIND LARGEST NUMBER AMONG THE SERIES

```
DATA SEGMENT                ;start of data segment
X DW 0010H,52H,30H,40H,50H
LAR DW ?
DATA ENDS           ;end of data segment
CODE SEGMENT        ;start of code segment
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
```

```
MOV DS,AX          ;initialize data segment
MOV CX,05H         ;load CX register with number of data words in X
LEA SI,X           ;initialize SI to point to the first number
MOV AX,[SI]        ;make a copy of the number pointed by SI in AX
DEC CX             ;set count value in CX for comparison
UP:
 CMP AX,[SI+2];compare two adjacent numbers(one is in AX and the other is pointed by SI+2)
JA CTINUE;if contents of AX is greater than the next number in array retain the contents of AX
MOV AX,[SI+2]      ;if not make a copy of the larger number in AX
CTINUE:ADD SI,2    ;point to the next number
DEC CX             ;decrement CX to check if all numbers are compared
JNZ UP             ;if no continue to compare
MOV LAR,AX;if yes make a copy of AX(largest number) in user defined memory location LAR
MOV AH,4CH         ;terminate the process
INT 21H
CODE ENDS          ;end of code segment
END START
```

SMALLEST NUMBER AMONG THE SERIES

```
DATA SEGMENT
X DW 0060H,0020H,0030H,0040H,0050H
MES DB 10,13,'SMALLEST NUMBER AMONG THE SERIES IS $'
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
MOV CX,05H         ;load CX with number of Data words in array X
LEA SI,X           ;SI points to the first number in array X
MOV AX,[SI]        ;make a copy of the first number in AX
DEC CX             ;intialize CX with count value for comparison
UP: CMP AX,[SI+2]  ;compare the contents of AX with next number in array pointed by SI+2
JB CONTINUE        ;if AX is smaller than the next number retain the contents of AX
MOV AX,[SI+2]      ;else make a copy of the smaller number in AX
CONTINUE:ADD SI,2  ;SI points to the next number
DEC CX             ;decrement the count value
```

```
JNZ UP          ;check if all the numbers are compared if no continue comparison
AAM             ;if yes convert the smallest binary number to unpacked BCD
ADD AX,3030H    ;convert the unpacked BCD to unpacked ASCII equivalent
MOV BX,AX       ;make a copy of the unpacked ASCII in BX
MOV AH,09H      ;display the message stored at user defined memory
                 location MES using DOS interrupts
LEA DX,MES      ;display the smallest number in array X using DOS interrupts
INT 21H
MOV DL,BH       ;display the smallest number in array X using DOS interrupts
MOV AH,02H
INT 21H
MOV DL,BL
INT 21H
MOV AH,4CH              ;terminate the process
INT 21H
CODE ENDS
END START
```

OUTPUT: SMALLEST NUMBER AMONG THE SERIES IS 0020

Week #13 Lab # 15

Program to short the numbers in ascending / descending order; Program for string transfer and Program to search for a character in a string.

PROGRAM TO SORT THE NUMBERS IN ASCENDING/DESCENDING ORDER

```
DATA SEGMENT
x DW 42H,34H,26H,17H,09H
LEN EQU 05
ASCD DB 10 DUP(0)
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
MOV DS,AX
```

```
MOV BX,LEN-1      ;load BX(counter1) with countvalue(number of datawords in array - 1)

MOV CX,BX         ;make a copy of the count value in CX(counter2)

UP1: MOV BX,CX    ;load the updated CX in BX
LEA SI,X          ;SI points to the first number in the array
UP: MOV AX,[SI]   ;make a copy of the number pointed by SI in AX
MOV DX,[SI+2]     ;make a copy of the next number in DX

CMP AX,DX         ;compare both the numbers
JB DOWN/JA DOWN       ;if AX < DX/AX > DX retain them as it is

MOV [SI],DX           ;if not sort the numbers in ascending order
MOV [SI+2],AX

DOWN: INC SI      ;point to the next number

INC SI

DEC BX      ;decrement the counter1
JNZ UP      ;compare till the larger number is sorted at the end of the array
DEC CX      ;decrement counter2
JNZ UP1     ;compare till the numbers are sorted in ascending order
MOV AH,4CH  ;terminate the process
INT 21H

CODE ENDS

END START           OUTPUT: 09 17 26 34 42
```

PROGRAM FOR STRING TRANSFER

```
DATA SEGMENT

STR1 DB 'HOW ARE YOU'

LEN EQU $-STR1

STR2 DB 20 DUP(0)

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE,DS:DATA,ES:DATA

START: MOV AX,DATA

MOV DS,AX         ;initialize data segment

MOV ES,AX         ;initialize extra segment for string operations

LEA SI,STR1       ;SI points to starting address of string at ;STR1

LEA DI,STR2       ;DI points to starting address of where the string has to be transferred
```

```
MOV CX,LEN          ;load CX with length of the string
CLD                 ;clear the direction flag for auto increment SI and DI
REP MOVSB           ;the source string is moved to destination address till
                    CX=0(after every     move CX isdecremented)
MOV AH,4CH          ;terminate the process
INT 21H
CODE ENDS
END START
```

PROGRAM TO SEARCH FOR A CHARACTER IN A STRING

```
DATA SEGMENT
MSG DB 'HELLO'
CNT EQU $-MSG
SRC EQU 'E'
MSG1 DB 10,13,'CHARACTER FOUND$'
MSG2 DB 10,13,'CHARACTER NOT FOUND$'
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA,ES:DATA
START: MOV AX,DATA
MOV DS,AX
MOV ES,AX
LEA SI,MSG  ;SI points to the starting address of the string
MOV AL,SRC ;the character to be searched in the string is stored in AL
```

```
MOV CL,CNT ;CX is loaded with count value equal to number of characters in the string
MOV CH,00H
CLD             ;clear the direction flag for auto increment SI and DI
UP: SCASB ; check if the character in AL is the same as that pointed by index register
JZ DOWN      ;if it is same jump to label DOWN
LOOP UP      ;if not decrement CX and continue checking till CX is zero
LEA DX,MSG2
MOV AH,09H          ;display the message at MSG2 that isCHARACTER NOT FOUND
INT 21H
JMP EXIT            ;jump to label EXIT
DOWN: LEA DX,MSG1
MOV AH,09H          ;if the character is found display the message CHARACTER FOUND
INT 21H
EXIT: MOV AH, 4CH
INT 21H
CODE ENDS     END START     OUTPUT: CHARACTER FOUND
```

Week #14 Lab # 14

Program to use DOS interrupt INT 21H function call for reading a character from keyboard, display a character and string on console

```
DATA SEGMENT

INKEY DB ?

BUF DB 20 DUP(0)

MES DB 10,13, 'DAYANANDA SAGAR COLLEGE OF ENGINEERING $'

DATA ENDS

CODE SEGMENT

ASSUME CS:CODE , DS:DATA
```

```
START:   MOV AX,DATA
MOV DS,AX
MOV AH,01H    ;DOS function to read a character from keyboard ;with
                    echo. [AL = 8bit character]
INT 21H
MOV INKEY,AL  ;Returns ASCII value of the pressed key.
MOV BUF,10        ;Load how many characters to enter.
MOV AH,0AH      ;Dos function to read string of characters from keyboard.
LEA DX,BUF
INT 21H
MOV AH,06H      ;Dos function to display a character.
MOV DL,'A'        ;Load the character to be displayed.
INT 21H
 MOV AH,09H     ;Dos function to read string of characters from keyboard.
LEA DX,MES       ;DX = offset address of the message
INT 21H
MOV AH,4CH
INT 21H
CODE ENDS
END START
```

Week #15 Lab # 15
INTERFACING EXPERIMENTS

(i) SEVEN SEGMENT DISPLAY INTERFACE

```
DATA SEGMENT

PORTA EQU 120H

PORTB EQU 121H

PORTC EQU 122H

CWRD EQU 123H

TABLE DB  8CH, 0C7H, 86H, 89H

DATA ENDS
```

```
CODE SEGMENT
  ASSUME CS:CODE, DS:DATA
  START:  MOV AX,DATA       ;intialise data segment
MOV DS,AX
MOV AL,80H                  ;initialise 8255 portb and portc as o/p
MOV DX,CWRD
OUT DX,AL
MOV BH,04                ; BH = no of digitsto be displayed
LEA SI,TABLE             ; SI = starting address of lookup table
 NEXTDIGIT:MOV CL,08     ; CL = no of segments = 08
MOV AL,[SI]
 NEXTBIT:  ROL AL,01
MOV CH,AL             ;save al
MOV DX,PORTB      ;one bit is sent out on portb
OUT DX,AL
MOV AL,01
MOV DX,PORTC     ;one clock pulse sent on pc0
OUT DX,AL
DEC AL
MOV DX,PORTC
OUT DX,AL
MOV AL,CH             ; get the sevensegment code back in al
DEC CL                ;send all 8 bits,thus one digit is displayed
```

```
JNZ NEXTBIT
DEC BH
INC SI
JNZ NEXTDIGIT
;display all the four digits
MOV AH,4CH          ;exit to dos
INT 21H
        CODE ENDS
        END START
```

(ii) STEPPER MOTOR INTERFACE

```
DATA SEGMENT
PORTA EQU 120H
PORTB EQU 121H
PORTC EQU 122H
CWRD EQU 123H
DATA ENDS
CODE SEGMENT
ASSUME CS:CODE,DS:DATA
```

START:

UP:

DELAY:

UP2:

UP1:

CODE ENDS

MOV AX,DATA

MOV DS,AX

MOV AL,80H ;initialise 8255 ,porta as o/p port

MOV DX,CWRD

OUT DX,AL

MOV DX,PORTA

MOV AL,88H ;load initial bit pattern

OUT DX,AL ;output on porta

CALL DELAY

ROL AL,01H ;rotate left to get exitation sequence of 11,22,44,88

OUT DX,AL

JMP UP

MOV CX,0FFFFH ;delay can be adjusted to get different speeds

MOV BX,0FFH

DEC BX

JNZ UP1

DEC CX

JNZ UP2

RET

MOV AH,4CH

INT 21H

END START