

# **LAB MANUAL**

**Course : CS1446 [ Assembly Language  
Programming Lab]**

**Class : B.Sc Computer Science**

**Semester : 4**

**Prepared By : Reshma Ann Mathews**



**COLLEGE OF APPLIED SCIENCE**

# **PERISSERY**

## **TABLE OF CONTENTS**

1. Familiarisation of 8086 Microprocessor
2. 8bit addition
3. 16 bit addition
4. 8bit subtraction
5. 16 bit subtraction
6. 8bit unsigned multiplication
7. 8bit signed multiplication
8. 16bit unsigned multiplication
9. 16bit signed multiplication
10. BCD addition
11. BCD subtraction
12. 16/8 division
13. 8 bit addition of a series
14. Smallest number from the list
15. Largest number from the list

## 1. FAMILIARISATION OF 8086 MICROPROCESSOR

8086 Microprocessor is an enhanced version of 8085 Microprocessor that was designed by Intel in 1976. It is a 16-bit Microprocessor having 20 address lines and 16 data lines that provides up to 1MB storage. It consists of powerful instruction set, which provides operations like multiplication and division easily.

It supports two modes of operation, i.e. Maximum mode and Minimum mode. Maximum mode is suitable for system having multiple processors and Minimum mode is suitable for system having a single processor.

### Microprocessor - 8086 Instruction Sets

The 8086 microprocessor supports 8 types of instructions –

- 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

### Data Transfer Instructions

These instructions are used to transfer the data from the source operand to the destination operand. Following are the list of instructions under this group –

#### **Instruction to transfer a word**

- **MOV** – Used to copy the byte or word from the provided source to the provided destination.
- **PPUSH** – Used to put a word at the top of the stack.
- **POP** – Used to get a word from the top of the stack to the provided location.

- **PUSHA** – Used to put all the registers into the stack.
- **POPA** – Used to get words from the stack to all registers.
- **XCHG** – Used to exchange the data from two locations.
- **XLAT** – Used to translate a byte in AL using a table in the memory.

### Instructions for input and output port transfer

- **IN** – Used to read a byte or word from the provided port to the accumulator.
- **OUT** – Used to send out a byte or word from the accumulator to the provided port.

### Instructions to transfer the address

- **LEA** – Used to load the address of operand into the provided register.
- **LDS** – Used to load DS register and other provided register from the memory
- **LES** – Used to load ES register and other provided register from the memory.

### Instructions to transfer flag registers

- **LAHF** – Used to load AH with the low byte of the flag register.
- **SAHF** – Used to store AH register to low byte of the flag register.
- **PUSHF** – Used to copy the flag register at the top of the stack.
- **POPF** – Used to copy a word at the top of the stack to the flag register.

## Arithmetic Instructions

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

Following is the list of instructions under this group –

### Instructions to perform addition

- **ADD** – Used to add the provided byte to byte/word to word.
- **ADC** – Used to add with carry.
- **INC** – Used to increment the provided byte/word by 1.
- **AAA** – Used to adjust ASCII after addition.
- **DAA** – Used to adjust the decimal after the addition/subtraction operation.

### Instructions to perform subtraction

- **SUB** – Used to subtract the byte from byte/word from word.
- **SBB** – Used to perform subtraction with borrow.
- **DEC** – Used to decrement the provided byte/word by 1.
- **NPG** – Used to negate each bit of the provided byte/word and add 1/2's complement.
- **CMP** – Used to compare 2 provided byte/word.
- **AAS** – Used to adjust ASCII codes after subtraction.
- **DAS** – Used to adjust decimal after subtraction.

### Instruction to perform multiplication

- **MUL** – Used to multiply unsigned byte by byte/word by word.
- **IMUL** – Used to multiply signed byte by byte/word by word.
- **AAM** – Used to adjust ASCII codes after multiplication.

### Instructions to perform division

- **DIV** – Used to divide the unsigned word by byte or unsigned double word by word.
- **IDIV** – Used to divide the signed word by byte or signed double word by word.
- **AAD** – Used to adjust ASCII codes after division.
- **CBW** – Used to fill the upper byte of the word with the copies of sign bit of the lower byte.
- **CWD** – Used to fill the upper word of the double word with the sign bit of the lower word.

### 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 corresponding bit in another byte/word.

- **OR** – Used to multiply each bit in a byte/word with the corresponding bit in another byte/word.
- **XOR** – Used to perform Exclusive-OR operation over each bit in a byte/word with the corresponding bit in another byte/word.
- **TEST** – Used to add operands to update flags, without affecting operands.

### Instructions to perform shift operations

- **SHL/SAL** – Used to shift bits of a byte/word towards left and put zero(S) in LSBs.
- **SHR** – Used to shift bits of a byte/word towards the right and put zero(S) in MSBs.
- **SAR** – Used to shift bits of a byte/word towards the right and copy the old MSB into the new MSB.

### Instructions to perform rotate operations

- **ROL** – Used to rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF].
- **ROR** – Used to rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF].
- **RCR** – Used to rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB.
- **RCL** – Used to rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB.

### String Instructions

String is a group of bytes/words and their memory is always allocated in a sequential order.

Following is the list of instructions under this group –

- **REP** – Used to repeat the given instruction till  $CX \neq 0$ .
- **REPE/REPZ** – Used to repeat the given instruction until  $CX = 0$  or zero flag  $ZF = 1$ .
- **REPNE/REPNZ** – Used to repeat the given instruction until  $CX = 0$  or zero flag  $ZF = 1$ .
- **MOVS/MOVSMB/MOVSX** – Used to move the byte/word from one string to another.

- **COMS/COMP SB/COMP SW** – Used to compare two string bytes/words.
- **INS/INSB/INSW** – Used as an input string/byte/word from the I/O port to the provided memory location.
- **OUTS/OUTSB/OUTSW** – Used as an output string/byte/word from the provided memory location to the I/O port.
- **SCAS/SCASB/SCASW** – Used to scan a string and compare its byte with a byte in AL or string word with a word in AX.
- **LODS/LODSB/LODSW** – Used to store the string byte into AL or string word into AX.

### Program Execution Transfer Instructions (Branch and Loop Instructions)

These instructions are used to transfer/branch the instructions during an execution. It includes the following instructions –

Instructions to transfer the instruction during an execution without any condition –

- **CALL** – Used to call a procedure and save their return address to the stack.
- **RET** – Used to return from the procedure to the main program.
- **JMP** – Used to jump to the provided address to proceed to the next instruction.

Instructions to transfer the instruction during an execution with some conditions –

- **JA/JNBE** – Used to jump if above/not below/equal instruction satisfies.
- **JAE/JNB** – Used to jump if above/not below instruction satisfies.
- **JBE/JNA** – Used to jump if below/equal/ not above instruction satisfies.
- **JC** – Used to jump if carry flag CF = 1
- **JE/JZ** – Used to jump if equal/zero flag ZF = 1
- **JG/JNLE** – Used to jump if greater/not less than/equal instruction satisfies.
- **JGE/JNL** – Used to jump if greater than/equal/not less than instruction satisfies.
- **JL/JNGE** – Used to jump if less than/not greater than/equal instruction satisfies.

- **JLE/JNG** – Used to jump if less than/equal/if not greater than instruction satisfies.
- **JNC** – Used to jump if no carry flag ( $CF = 0$ )
- **JNE/JNZ** – Used to jump if not equal/zero flag  $ZF = 0$
- **JNO** – Used to jump if no overflow flag  $OF = 0$
- **JNP/JPO** – Used to jump if not parity/parity odd  $PF = 0$
- **JNS** – Used to jump if not sign  $SF = 0$
- **JO** – Used to jump if overflow flag  $OF = 1$
- **JP/JPE** – Used to jump if parity/parity even  $PF = 1$
- **JS** – Used to jump if sign flag  $SF = 1$

### **Processor Control Instructions**

These instructions are used to control the processor action by setting/resetting the flag values.

Following are the instructions under this group –

- **STC** – Used to set carry flag  $CF$  to 1
- **CLC** – Used to clear/reset carry flag  $CF$  to 0
- **CMC** – Used to put complement at the state of carry flag  $CF$ .
- **STD** – Used to set the direction flag  $DF$  to 1
- **CLD** – Used to clear/reset the direction flag  $DF$  to 0
- **STI** – Used to set the interrupt enable flag to 1, i.e., enable INTR input.
- **CLI** – Used to clear the interrupt enable flag to 0, i.e., disable INTR input.

### **Iteration Control Instructions**

These instructions are used to execute the given instructions for number of times. Following is the list of instructions under this group –

- **LOOP** – Used to loop a group of instructions until the condition satisfies, i.e.,  $CX = 0$
- **LOOPE/LOOPZ** – Used to loop a group of instructions till it satisfies  $ZF = 1$  &  $CX = 0$
- **LOOPNE/LOOPNZ** – Used to loop a group of instructions till it satisfies  $ZF = 0$  &  $CX = 0$

- **JCXZ** – Used to jump to the provided address if CX = 0

### **Interrupt Instructions**

These instructions are used to call the interrupt during program execution.

- **INT** – Used to interrupt the program during execution and calling service specified.
- **INTO** – Used to interrupt the program during execution if OF = 1
- **IRET** – Used to return from interrupt service to the main program

## **PART A**

### **Experiment No : 1**

#### **AIM**

Write a tasm program to perform 8 bit addition

#### **PROGRAM**

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

```
DATA SEGMENT
```

```
A DB 02H
```

```
B DB 03H
```

```
R DW 0080H
```

```
ENDS
```

```
CODE SEGMENT
```

```
START:
```

```
    MOV AX , DATA
```

```
    MOV DS, AX
```

```
    MOV AL,A
```

```
    ADD AL,B
```

```
    MOV DI,R
```

```
    MOV [DI] ,AX
```

```
MOV AH ,4CH
```

```
INT 21H
```

ENDS

END START

**OUTPUT**

0080:0005

**Experiment No: 2**

**AIM**

Write a tasm program to perform 16 bit addition

**PROGRAM**

ASSUME CS:CODE, DS:DATA

DATA SEGMENT

A DW 0040H

B DW 0020H

R DW 0080H

ENDS

CODE SEGMENT

START:

    MOV AX,DATA

    MOV DS,AX

    MOV AX,A

    ADD AX,B

    MOV DI,R

    MOV [DI],AX

MOV AH,4CH

INT 21H

ENDS

END START

**OUTPUT**

0080H:0060

## **Experiment No: 3**

### **AIM**

Write a tasm program to perform 8 bit subtraction

### **PROGRAM**

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

```
DATA SEGMENT
```

```
A DB 05H
```

```
B DB 02H
```

```
R DW 0050H
```

```
ENDS
```

```
CODE SEGMENT
```

```
START:
```

```
    MOV AX,DATA
```

```
    MOV DS,AX
```

```
    MOV AL,A
```

```
    SUB AL,B
```

```
    MOV DI,R
```

```
    MOV [DI],AX
```

```
MOV AH,4CH
```

```
INT 21H
```

```
ENDS
```

```
END START
```

### **OUTPUT**

```
0050H:03H
```

## **Experiment No: 4**

### **AIM**

Write a tasm program to perform 16 bit subtraction

### **PROGRAM**

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

```
DATA SEGMENT
```

```
A DW 0040H
```

```
B DW 0020H
```

```
R DW 0005H
```

```
ENDS
```

```
CODE SEGMENT
```

```
START:
```

```
    MOV AX,DATA
```

```
    MOV DS,AX
```

```
    MOV AX,A
```

```
    ADD AX,B
```

```
    MOV DI,R
```

```
    MOV [DI],AX
```

```
MOV AH,4CH
```

```
INT 21H
```

```
ENDS
```

```
END START
```

### **OUTPUT**

```
0005H:0020
```

## **Experiment No: 5**

### **AIM**

Write a tasm program to perform 8 bit signed multiplication

## **PROGRAM**

ASSUME CS:CODE, DS:DATA

DATA SEGMENT

A DW 04H

B DW 02H

R DW 0060H

ENDS

CODE SEGMENT

START:

    MOV AX,DATA

    MOV DS,AX

    MOV AL,A

    MUL B

    MOV DI,R

    MOV [DI],AX

MOV AH,4CH

INT 21H

ENDS

END START

## **OUTPUT**

0005H:08

## **Experiment No: 6**

### **AIM**

Write a tasm program to perform 8 bit unsigned multiplication

### **PROGRAM**

ASSUME CS:CODE, DS:DATA

DATA SEGMENT

A DW 02H

B DW 03H

R DW 0070H

ENDS

CODE SEGMENT

START:

MOV AX,DATA

MOV DS,AX

MOV AL,A

MUL B

MOV DI,R

MOV [DI],AX

MOV AH,4CH

INT 21H

ENDS

END START

### **OUTPUT**

0005H:05

## **Experiment No: 7**

### **AIM**

Write a tasm program to perform 16 bit unsigned multiplication

### **PROGRAM**

ASSUME CS:CODE, DS:DATA

DATA SEGMENT

A DW 0040H

B DW 0020H

R DW 0050H

ENDS

CODE SEGMENT

START:

MOV AX,DATA

MOV DS,AX

MOV AX,A

```
    IMUL B
    MOV DI,R
    MOV [DI],AX

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

### **OUTPUT**

0050H:0080

## **Experiment No: 8**

### **AIM**

Write a tasm program to perform 16 bit signed multiplication

### **PROGRAM**

```
ASSUME CS:CODE, DS:DATA

DATA SEGMENT
A DW 0002H
B DW 0004H
R DW 0080H
ENDS

CODE SEGMENT
START:
    MOV AX,DATA
    MOV DS,AX
    MOV AX,A
    IMUL B
    MOV DI,R
    MOV [DI],AX

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

END START

### **OUTPUT**

0050H:0008

## **Experiment No: 9**

### **AIM**

Write a tasm program to perform BCD addition

### **PROGRAM**

ASSUME CS:CODE, DS:DATA

DATA SEGMENT

A DW 02H

B DW 04H

R DW 0080H

ENDS

CODE SEGMENT

START:

MOV AX,DATA

MOV DS,AX

MOV AL,A

MOV BL,B

ADD AX,BX

DAA

MOV DI,R

MOV [DI],AX

MOV AH,4CH

INT 21H

ENDS

END START

### **OUTPUT**

0050H:06

## **Experiment No: 10**

### **AIM**

Write a tasm program to perform BCD subtraction

### **PROGRAM**

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

```
DATA SEGMENT
```

```
A DW 03H
```

```
B DW 02H
```

```
R DW 0080H
```

```
ENDS
```

```
CODE SEGMENT
```

```
START:
```

```
    MOV AX,DATA
```

```
    MOV DS,AX
```

```
    MOV AL,A
```

```
    MOV BL,B
```

```
    SUB AX,BX
```

```
    DAS
```

```
    MOV DI,R
```

```
    MOV [DI],AX
```

```
MOV AH,4CH
```

```
INT 21H
```

```
ENDS
```

```
END START
```

### **OUTPUT**

```
0080H:01
```

## **Experiment No: 11**

### **AIM**

Write a tasm program to perform 16/8 bit division

### **PROGRAM**

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

```
DATA SEGMENT
```

```
A DW 0008H
```

```
B DW 02H
```

```
R DW 0060H
```

```
ENDS
```

```
CODE SEGMENT
```

```
START:
```

```
    MOV AX,DATA
```

```
    MOV DS,AX
```

```
    MOV AX,A
```

```
    DIV B
```

```
    MOV DI,R
```

```
    MOV [DI],AX
```

```
MOV AH,4CH
```

```
INT 21H
```

```
ENDS
```

```
END START
```

### **OUTPUT**

```
0060H:04
```

## **Experiment No: 12**

### **AIM**

Write a tasm program to perform 8 bit addition of series

### **PROGRAM**

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

```
DATA SEGMENT
```

```
LIST DB 01H,02H,03H,04H
```

```
DES DW 0055H
```

```
ENDS
```

```
CODE SEGMENT
```

```
START:
```

```
    MOV AX,DATA
```

```
    MOV DS,AX
```

```
    MOV CX,0004H
```

```
    XOR AX,AX
```

```
    MOV SI,OFFSET LIST
```

```
AA:
```

```
    MOV BX,[SI]
```

```
    ADD AX,BX
```

```
    INC SI
```

```
LOOP AA
```

```
MOV DI,R
```

```
MOV [DI],AX
```

```
MOV AH,4CH
```

```
INT 21H
```

```
ENDS
```

```
END START
```

### **OUTPUT**

```
0055H:0A
```

## **Experiment No: 13**

### **AIM**

Write a tasm program to find smallest number from the list

### **PROGRAM**

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

```
DATA SEGMENT
```

```
LIST DB 22H, 31H, 13H, 55H, 10H
```

```
COUNT DW 04H
```

```
RES DW 0400H
```

```
ENDS
```

```
CODE SEGMENT
```

```
START:
```

```
    MOV AX,DATA
```

```
    MOV DS,AX
```

```
    MOV CX,COUNT
```

```
    MOV SI,OFFSET LIST
```

```
    MOV BL, [SI]
```

```
AGAIN:
```

```
    INC SI
```

```
    CMP BL, [SI]
```

```
    JNC NEXT
```

```
    MOV BL, [SI]
```

```
NEXT:
```

```
    LOOP AGAIN
```

```
    MOV DI, RES
```

```
    MOV [DI], BL
```

```
MOV AH, 4CH
```

```
INT 21H
```

```
ENDS
```

END START

## OUTPUT

0400H:10H

## Experiment No: 14

### AIM

Write a tasm program to find largest number from the list

### PROGRAM

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

```
DATA SEGMENT
```

```
LIST DB 22H, 31H, 13H, 55H, 10H
```

```
COUNT DW 04H
```

```
R DW 0400H
```

```
ENDS
```

```
CODE SEGMENT
```

```
START:
```

```
    MOV AX,DATA
```

```
    MOV DS,AX
```

```
    MOV CX,COUNT
```

```
    MOV SI,OFFSET LIST
```

```
    MOV BL, [SI]
```

```
AGAIN:
```

```
    INC SI
```

```
    CMP BL, [SI]
```

```
    JNC NEXT
```

```
    MOV BL, [SI]
```

```
NEXT:
```

```
    LOOP AGAIN
```

```
    MOV DI, R
```

```
    MOV [DI], BL
```

MOV AH, 4CH

INT 21H

ENDS

END START

**OUTPUT**

0400H:55H