**EXPERIMENT 8**

**AIM:** Implement 8086 based Assembly programs.

**THEORY:**

Intel 8086 is built on a single semiconductor chip and packaged in a 40-pin IC package. The type of package is DIP (Dual Inline Package). Intel 8086 uses 20 address lines and 16 data- lines. It can directly address up to 220 = 1 Mbyte of memory. 8086 is designed to operate in two modes, i.e., Minimum and Maximum mode.

Table

Description automatically generated

It consists of a powerful instruction set, which provides operation like division and multiplication very quickly.

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

**CODE:**

**1) Program to add two word length numbers**

OPR1: DW 0x6969 ; declare first number

OPR2: DW 0x0420 ; declare second number

RESULT: DW 0 ; declare place to store result

; actual entry point of the program

start:

MOV AX, word OPR1 ; move first number to AX

MOV BX, word OPR2 ; move second number to BX

CLC ; clear the carry flag

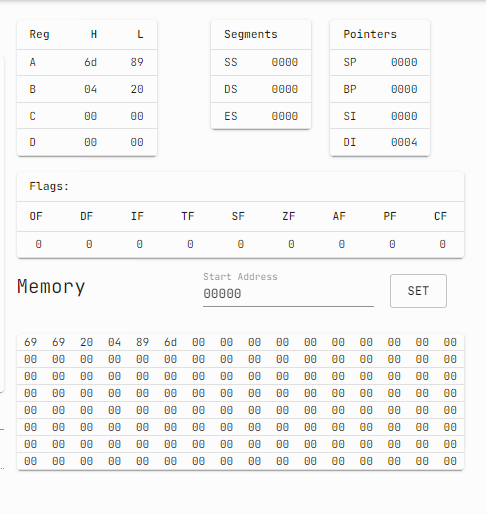
ADD AX, BX ; add BX to AX

MOV DI, OFFSET RESULT ; move offset of result to DI

MOV word [DI], AX ; store result

print reg ; print result

**OUTPUT:**



**2) A Program to move data from one segment to another**

SET 0 ; set address for segment 1

src:DB 0x3 ; store data

DB 0x5

DB 0x7

SET 0x1 ; set addresss for segment 2

dest:DB [0,3] ; store data

; actual entry point of the program

start:

print mem 0:8 ; print initial state of segment 1

print mem 0x10:8 ; print initial state of segment 2

MOV AX, 0 ; move address of seg1

MOV DS,AX ; to ds

MOV AX , 0x1 ; move address of seg2

MOV ES,AX ; to es

MOV SI, OFFSET src ; move offset of source data

MOV SI, OFFSET dest ; move offset of destination data

MOV CX, 0x3 ; move number of data items

print reg ; print state of registers

\_loop:

mov AH, byte DS[SI] ; move one byte from source to ah

mov byte ES[DI],AH ; move ah to destination

inc SI

inc DI

dec CX ; decrement count

jnz \_loop ; if count is not zero jump back

print mem 0:8 ; print final state of segment 1

print mem 0x10:8 ; print final state of segment 2

**OUTPUT:**

Table

Description automatically generated

**3) Program to calculate factorial using looping**

NUM: DW 0x6 ; calculate factorial of 6

RESULT: DW 0 ; place to store the reult

; actual entry point of the program

start:

MOV CX,word NUM ; move number into cx

MOV AX, 0x1 ; initialize accumulator with 1

NOTZEROLOOP: ; label to jump back to

MUL CX ; multiple by the number

DEC CX ; decrement the number

JNZ NOTZEROLOOP ; if not zero jump back

MOV word RESULT,AX ; store the result in memory

print reg ; print registers

**OUTPUT:**

Table

Description automatically generated

**4) Program to show use of interrupts**

hello: DB "Hello World" ; store string

; actual entry point of the program, must be present

start:

MOV AH, 0x13 ; move BIOS interrupt number in AH

MOV CX, 11 ; move length of string in cx

MOV BX, 0 ; mov 0 to bx, so we can move it to es

MOV ES, BX ; move segment start of string to es, 0

MOV BP, OFFSET hello ; move start offset of string in bp

MOV DL, 0 ; start writing from col 0

int 0x10 ; BIOS interrupt

**OUTPUT:**

Shape, rectangle

Description automatically generated with medium confidence

Table

Description automatically generated

**5) Program to show use of interrupts**

hello: DB "Hello World" ; store string

; actual entry point of the program

start:

MOV AH, 0x13 ; move BIOS interrupt number in AH

MOV CX, 12 ; move length of string in cx

MOV BX, 0 ; mov 0 to bx, so we can move it to es

MOV ES, BX ; move segment start of string to es, 0

MOV BP, OFFSET hello ; move start offset of string in bp

MOV DL, 0 ; start writing from col 0

int 0x10 ; BIOS interrupt

**OUTPUT:**

Graphical user interface, table

Description automatically generated

**6) Program to calculate LCM and GCD of two numbers**

no1: dw 0x6 ; number 1

no2: dw 0x5 ; number 2

gcd: dw 0 ; place to store gcd

lcm: dw 0 ; place to store lcm

; actual entry point of the program

start:

mov ax, word no1 ; move number 1 in accumulatore

mov bx, word no2 ; move number 2 in register BX

loop0: mov dx, 0x0 ; place to loop back

; cannot use 'loop' as label, as loop is an opcode which will give error when used with jumps

div bx ; divide accumulator by bx

mov ax, bx

mov bx, dx

cmp bx, 0x0 ; check if bx is 0

jnz loop0 ; if not loop back

mov word gcd, ax ; store gcd

mov cx, ax ; move ax in cx

mov ax, word no1 ; move number 1 in accumulatore

mov bx, word no2 ; move number 2 in register BX

mul bx ; multiply accumulator by BX

div cx ; divide accumulator by CX

mov word lcm, ax ; store lcm

print mem :16 ; print memory

**OUTPUT:**

Graphical user interface, table

Description automatically generated

**CONCLUSION:** In this experiment, I implemented 8086 microprocessor’s assembly language based programs. The codes were run on an online 8086 emulator.The programs were to add two word length numbers, to calculate LCM and GCD of two numbers, to transfer the data, to calculate the factorial using loop in 8086 assembly instruction set and programs to implement interrupts.