(An Autonomous Institute of Government of Maharashtra)
GECA
Government College of Engineering, Aurangabad
शासकीय अभियांत्रिकी महाविद्यालय, औरंगाबाद
In pursuit of Technical Excellence
महाराष्ट्र शासन

Department of Electronics and Telecommunication

ET3012: Lab Computer Architecture And Organization

Name : Ram Shingane

Enrollment No : BE19F04F059

Class : TY (ENTC)

Batch : T4

EXPERIMENT NO. 01

AIM: Write and execute following program for 8085 processor

a. Addition of two 8-bit numbers stored in memory, also store result in memory.
b. To count no of 1's and 0's in a given byte.
c. To obtain division of two 8-bit numbers.
d. To move a block of data. (10 numbers from one memory location to another)
e. To arrange 10 random numbers in ascending as well as in descending order.
f. To obtain Fibonacci series.
g. To obtain factorial of a number.

THEORY: 8085 is pronounced as 'Eighty-Eighty-Five' microprocessor. It is an 8bit microprocessor designed by Intel in 1977 using NMOS technology. It has following configurations:-

- 8-bit data bus
- 16-bit address bus, which can address up to 64kb
- A 16-bit programmer counter
- A 16-bit stack pointer
- six 8-bit registers arranged in pairs:-BC,DF,HL
- Requires +5V supply to operate at 3.2MHz single phase clock
- It is used in washing machines, micro-wave oven, mobile phones, etc.

FUNCTIONAL UNITS OF 8085 MICROPROCESSER:-

Accumulator: It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operation. It is connected to internal data bus ALU.

Arithmetic & Logic Unit: As the name suggests, it performs arithmetic & logical operations like Addition, Subtraction, AND, OR on 8-bit data.

General Purpose Register: There are 6 general purpose registers in 8085 processors, i.e. B,C,D,F,H & L. Each register can hold 8-bit data. There registers can work in pair to hold 16-bit

data.

Program Counter: It is the 16-bit register used to store the memory address location of next instruction to be executed. Microprocessor increments the program wherever an instruction is being executed ,so that program counter points to memory address of next instruction that is going to be executed.

Stack Pointer: It is also 16-bit register works like stack which is always incremented /decremented by 2 during push & pop operations.

Temporary register: It is an 8-bit register which holds the temporary data of arithmetic & logical operation.

Address Bus & Data Bus: Data bus carries the data to be stored .It is bidirectional, whereas address bus carries the location to where it should be stored & it is unidirectional .It is used to transfer the data & address I/O devices.

Addressing Modes in 8085: These are the instructions used to transfer the data from one register to another register ,from the memory to the register, and from the memory to the register and from register to memory without any alteration in the content .Addressing modes in 8085 are classified into 5 groups:-

1) **Immediate Addressing Mode:-** In this mode ,the 8/16-bit data is specified in the instructions itself as one of its operands .e.g.: MVI K,20F ; 20F is copied into register K.
2) **Direct Addressing Mode:-** In this mode, the data is directly copied from the given address to the register

.e.g.: LDB 5000 K; means the data at address 5000K is copied to register B.

3) **Indirect Addressing Mode:-** In this mode, the data is transferred from one register to another by using the address pointed by the register to another .e.g.: MOV K,B ; means data is transferred from the memory address pointed by the register to register K.
4) **Implied Addressing Mode:-** This mode doesn't require any operand ;the data is specified by the e.g. :CMP

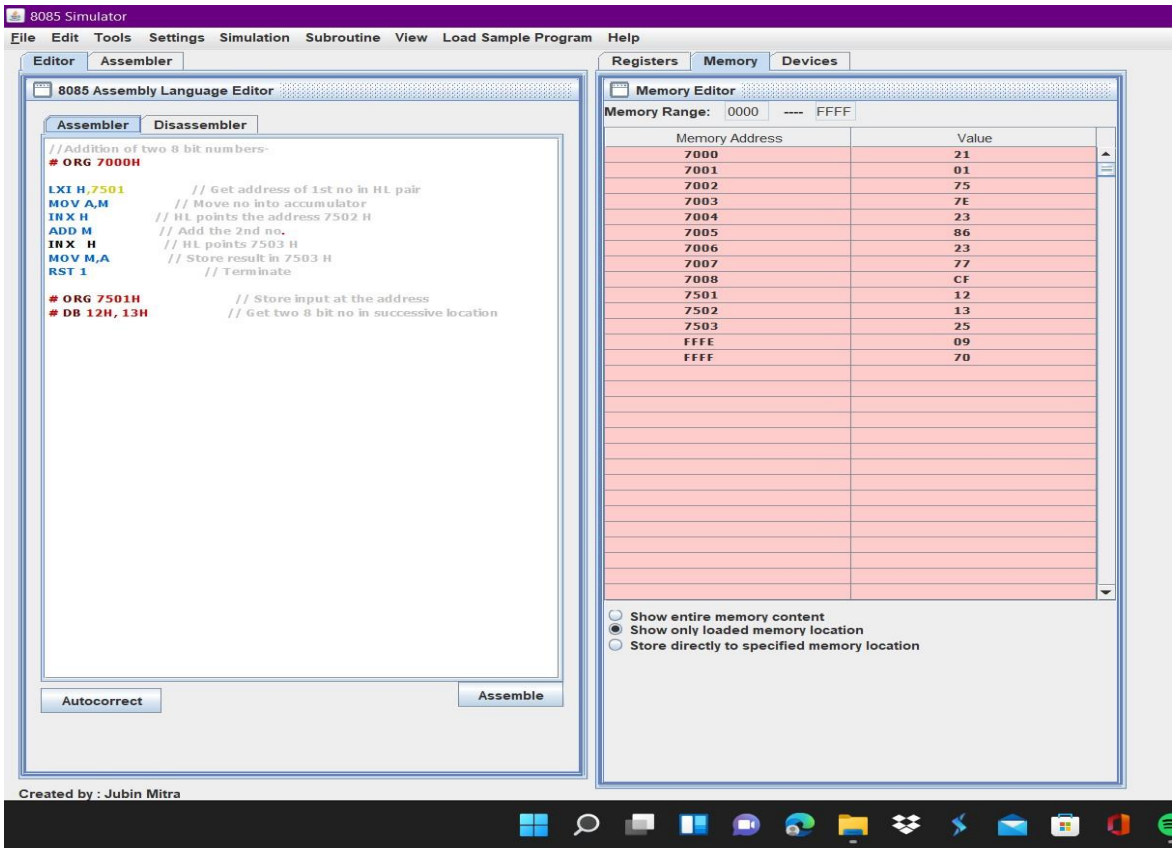
a. Addition on two 8 bit numbers-

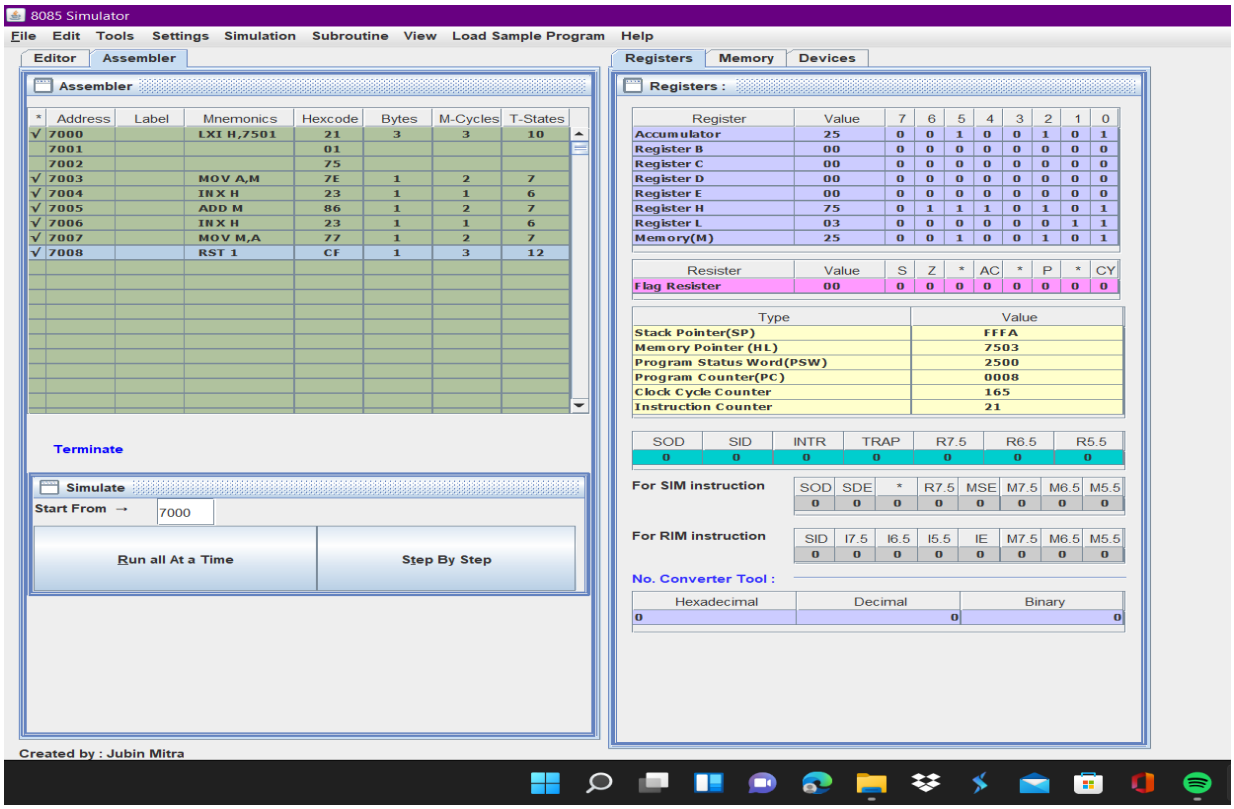
Input :

CODE:

```
//Addition of two 8 bit numbers-
# ORG 7000H
LXI H,7501              // Get address of 1st no in HL pair
MOV A,M                 // Move no into accumulator
INX H                   // HL points the address 7502 H
ADD M                   // Add the 2nd no.
INX   H                 // HL points 7503 H
MOV M,A                 // Store result in 7503 H
RST 1                   // Terminate
# ORG 7501H             // Store input at the address
# DB 12H, 13H           // Get two 8 bit no in successive location
```

INPUT & OUTPUT:





B. To count no of 1's and 0's in a given byte.

CODE:

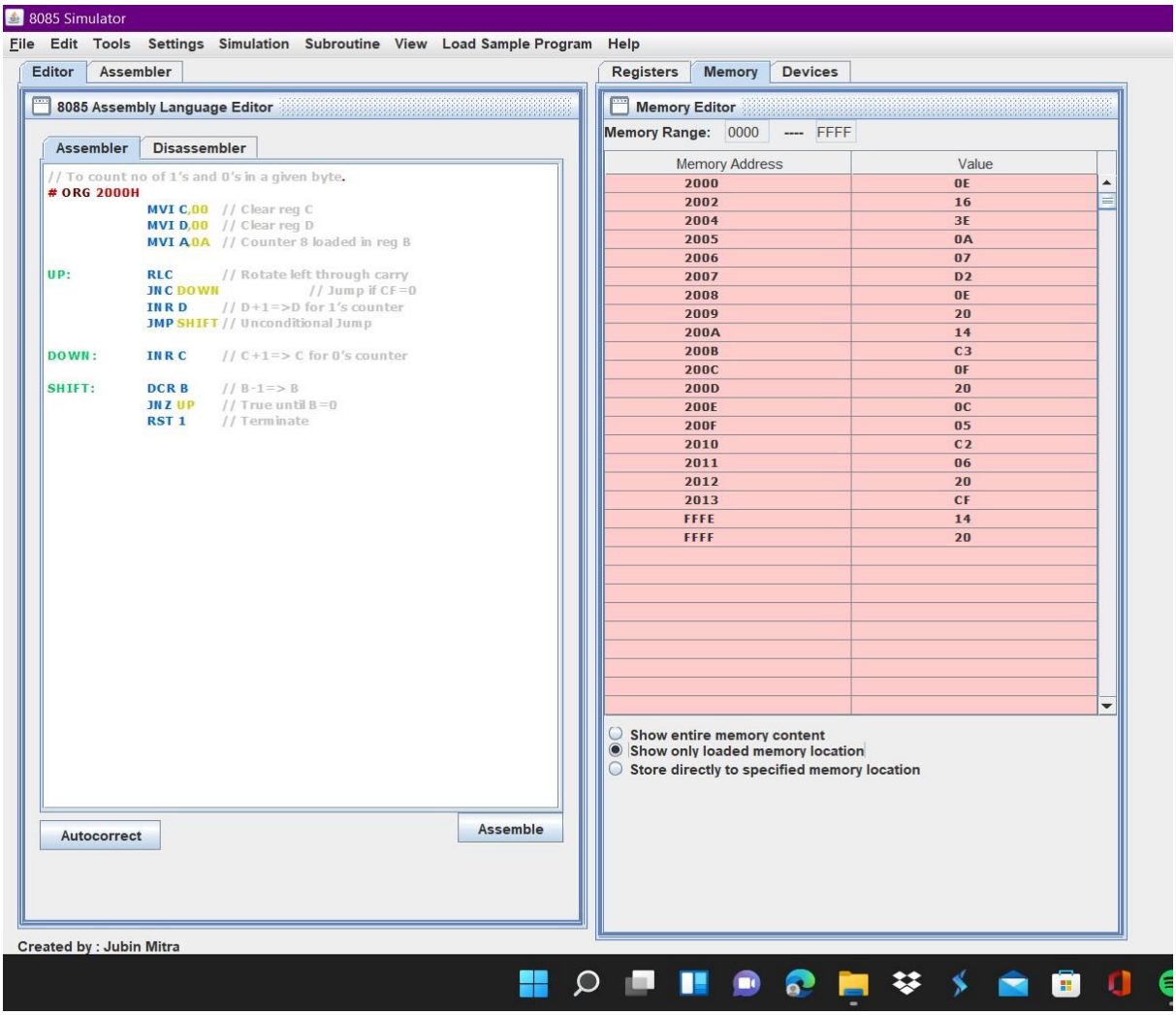
```
# ORG 2000H
MVI C,00        // Clear reg.C
MVI D,00        // Clear reg.D
MVI A,0A        // Take number into Accumulator
MVI B,08        // Counter 8 loaded in reg.B
up: RLC         // Rotate left through carry
JNC down        // Jump if CF=0
INR D           // D+1=>D for 1's counter
JMP shift       // Unconditional Jump
down: INR C     // C+1=> C for 0's counter
```

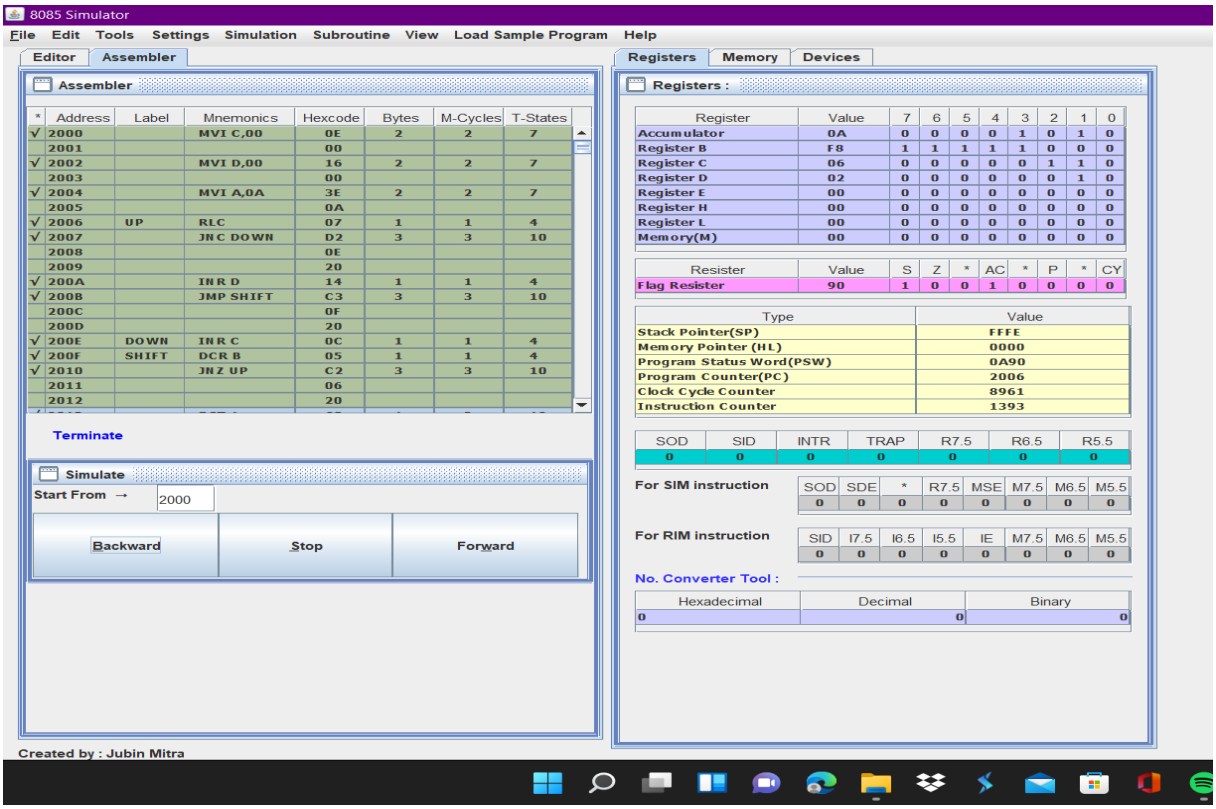
shift: DCR B // B-1=> B

JNZ up // True until B=0

RST 1 // Terminate

INPUT & OUTPUT:





C. Division of two 8 bit numbers

(without remainder)

CODE:

```
# ORG 7000H
 LDA      7501     // [7501]=>A (Divisor)
MOV   B,A     // Take divisor in reg,B
LDA   7502      // Take dividend in reg,A
MVI   C,00      // Quotient=00
CMP   B     // Compare A to B
JC   down     // Jump if carry
up: SUB   B     // Dividend-divisor=>A
INR   C     // C=C+1
CMP   B     // Is dividend < divisor
```

JNC up // If not,go back

down:

MOV A,C // C=>A

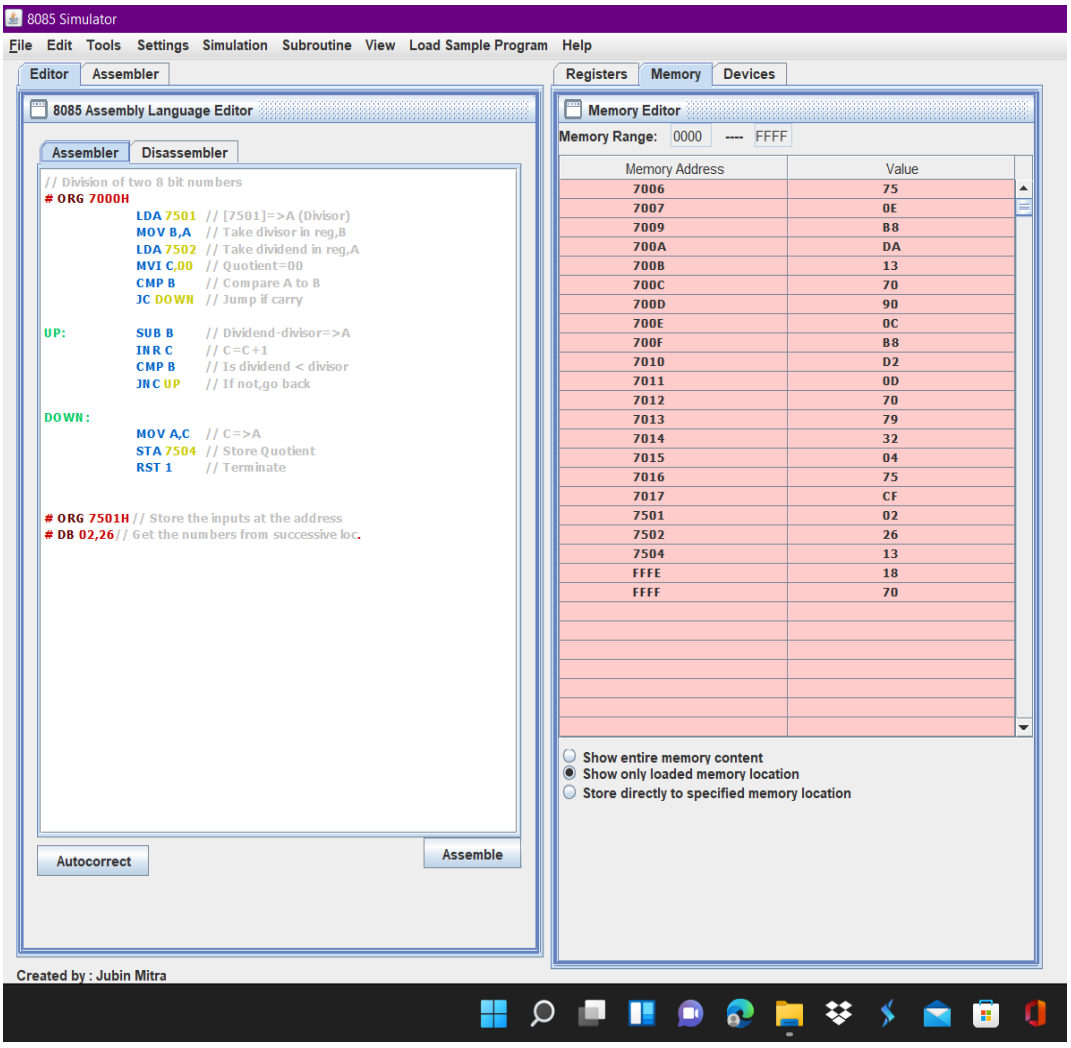
STA 7504 // Store Quotient

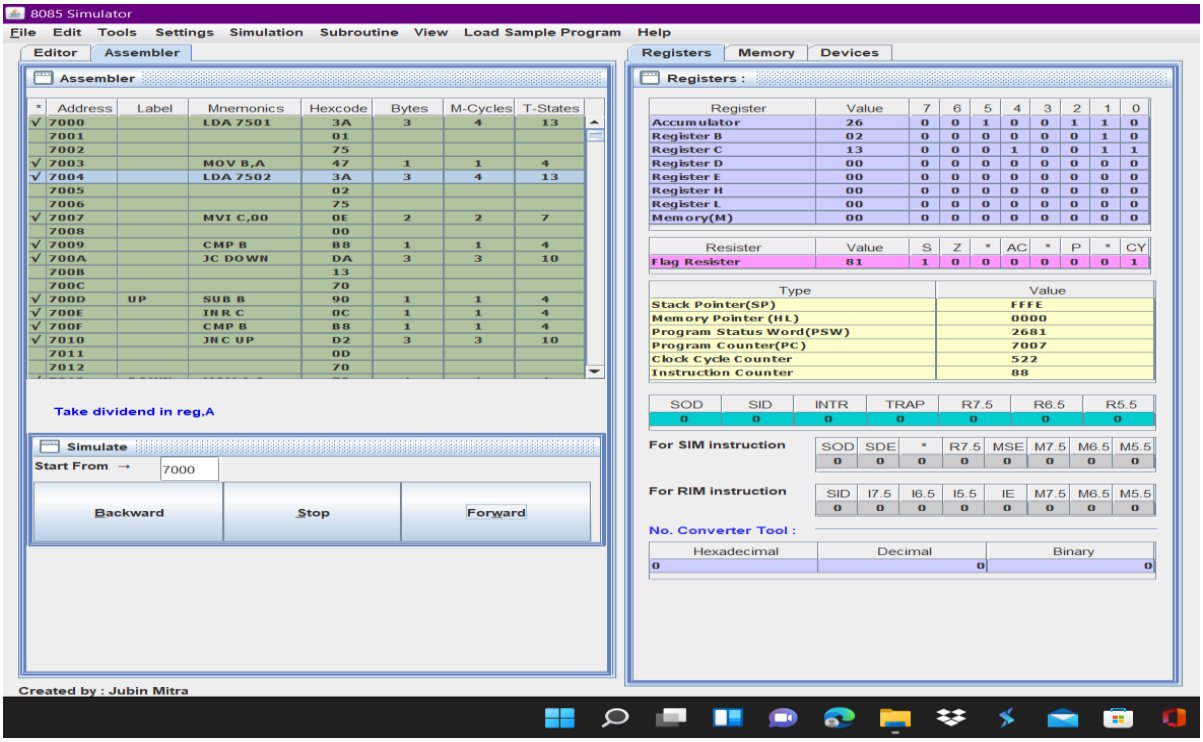
RST 1 // Terminate

ORG 7501H // Store the inputs at the address

DB 02,26 // Get the numbers from successive loc.

INPUT & OUTPUT:





(with remainder)

CODE:

```
# ORG 7000H
LDA     7501    // [7501]=>A (Divisor)
MOV   B,A    // Take divisor in reg,B
LDA   7502     // Take dividend in reg,A
MVI   C,00     // Quotient=00
CMP   B    // Compare A to B
JC  down    // Jump if carry
up: SUB   B    // Dividend-divisor=>A
INR   C    // C=C+1
CMP   B    // Is dividend < divisor
JNC   up    // If not,go back
```

down: STA 7503 // Store Remainder

MOV A,C // C=>A

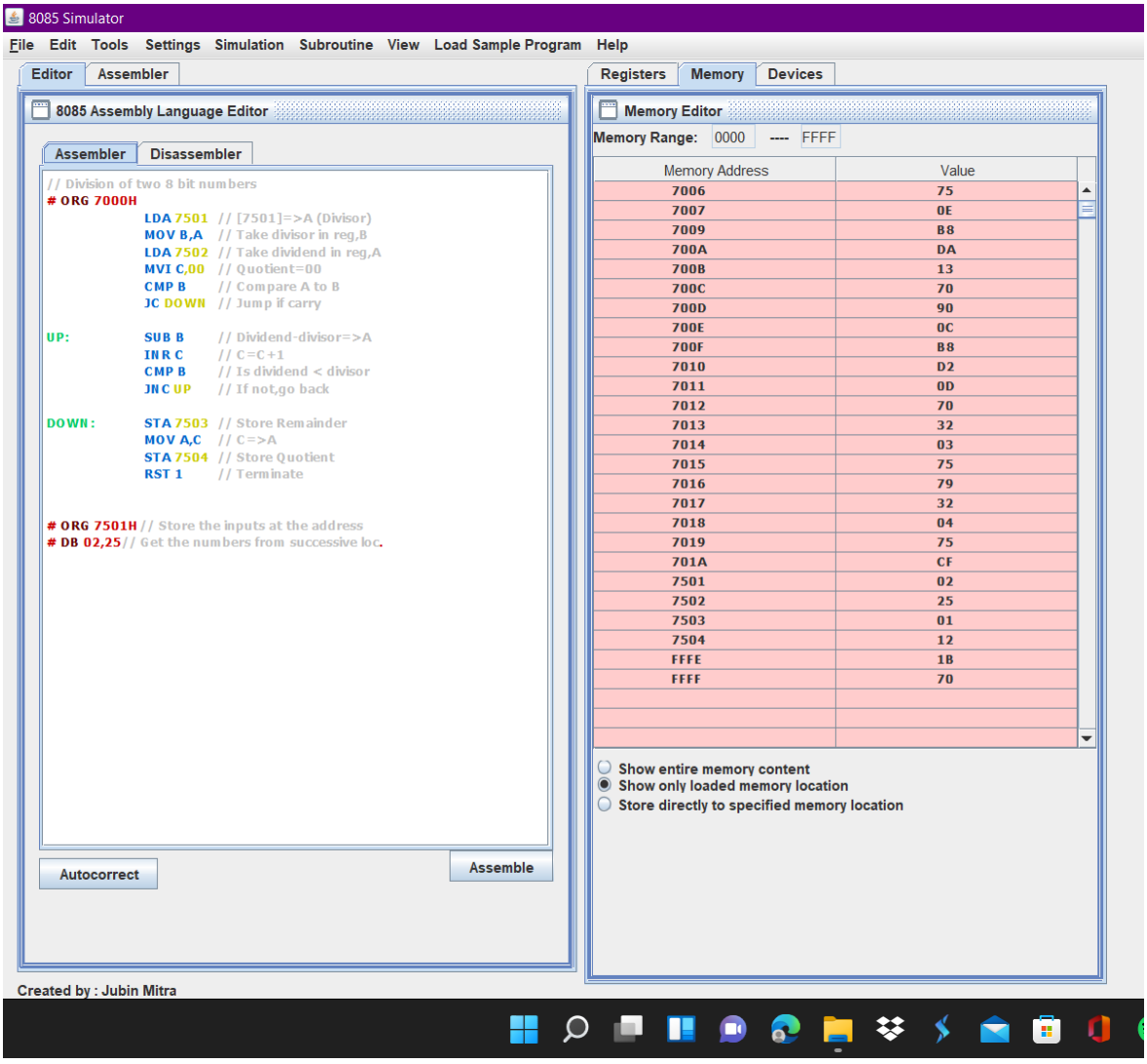
STA 7504 // Store Quotient

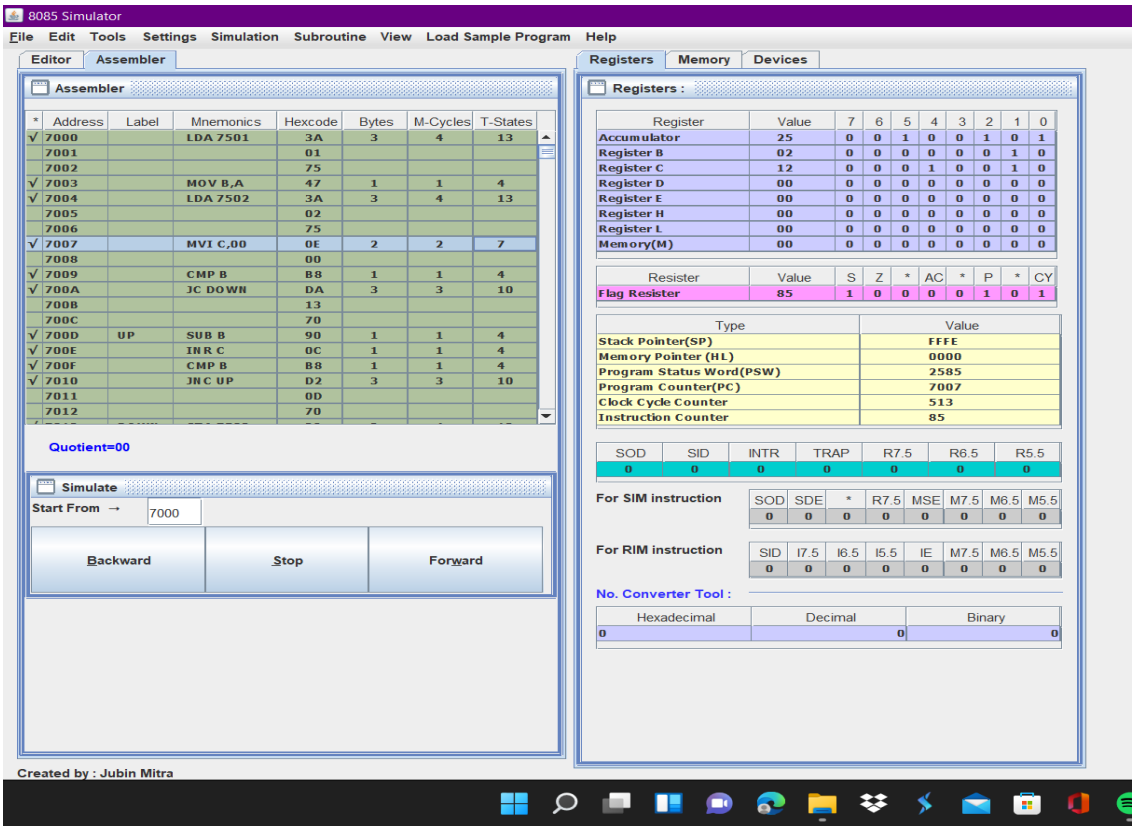
RST 1 // Terminate

ORG 7501H // Store the inputs at the address

DB 02,25 // Get the numbers from successive loc.

INPUT & OUTPUT:





D. Move a block of data (10 numbers from one memory location to another)

CODE:

```
// Move a block of data
# ORG 2000H
# BEGIN 2000H
        LXI  H,3090
        LXI D,30B0
        MOV C,M
L1:     INX H
        MOV A,M
        STAX D
```

INX D

DCR C

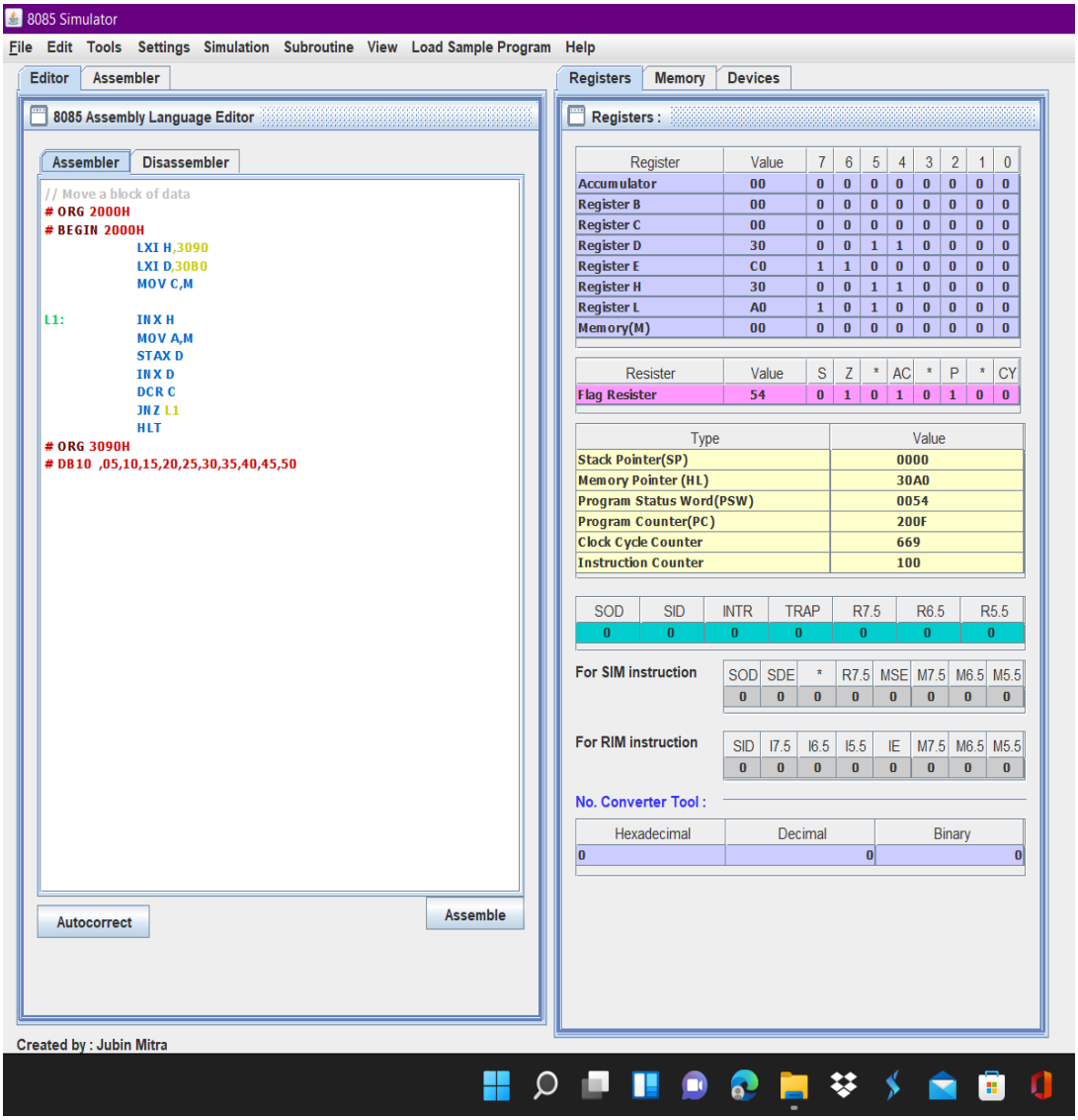
JNZ L1

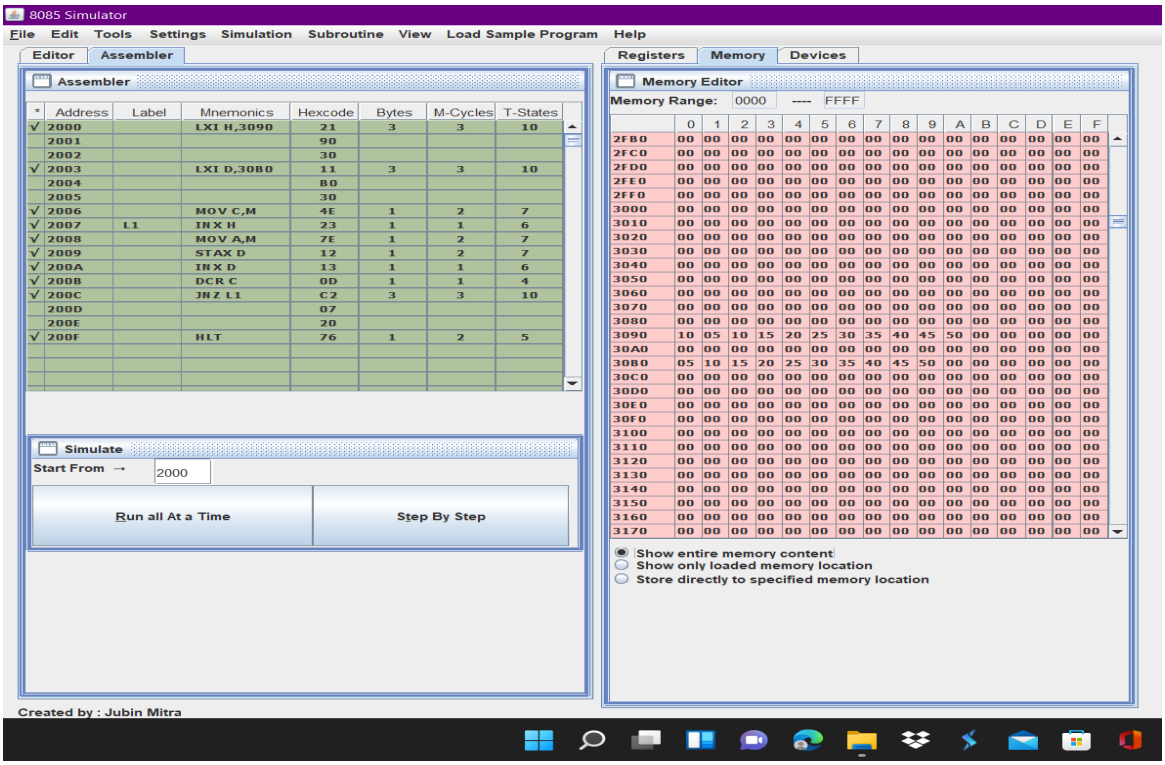
HLT

ORG 3090H

DB10 ,05,10,15,20,25,30,35,40,45,50

INPUT & OUTPUT:





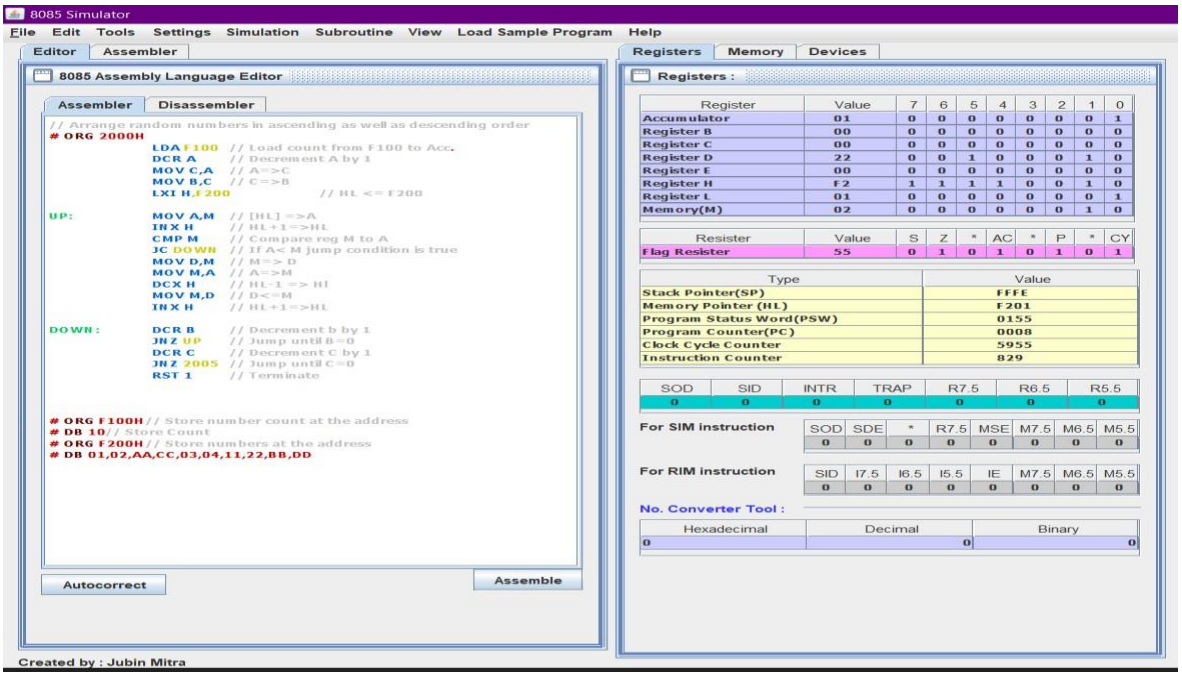
Arrange random numbers in ascending as well as descending order.

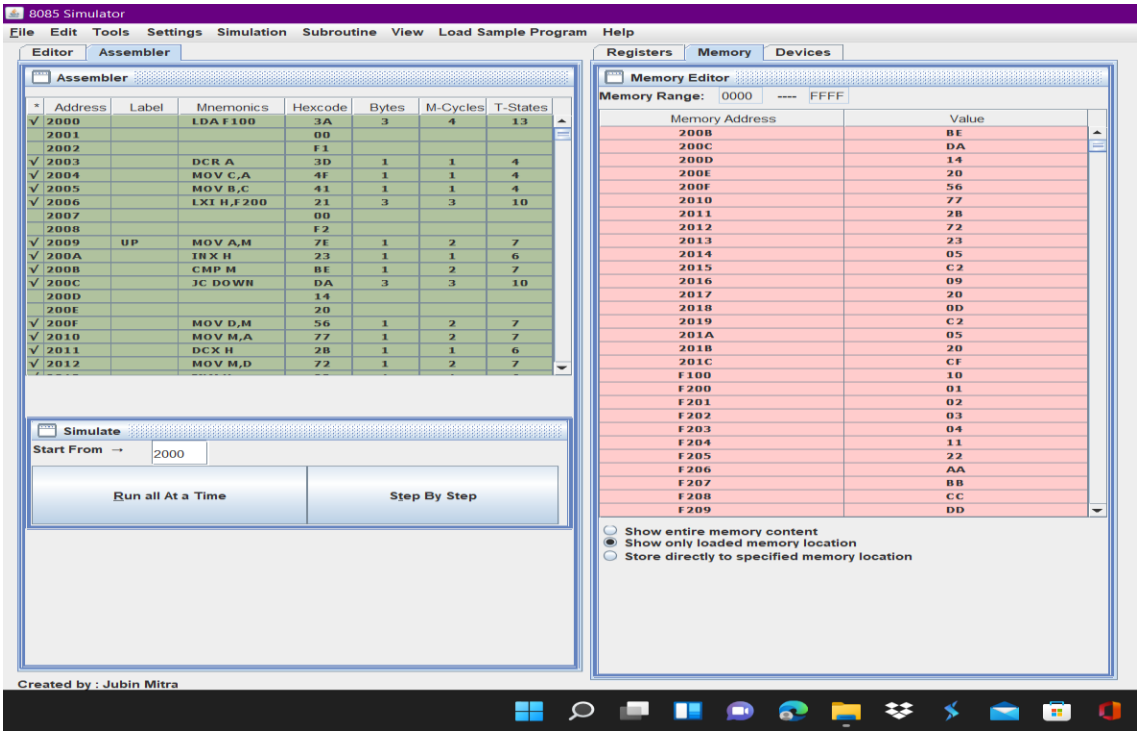
CODE:

```
// Arrange random numbers in ascending
# ORG 2000H
        LDA F100    // Load count from F100 to Acc.
        DCR A       // Decrement A by 1
        MOV C,A     // A=>C
        MOV B,C     // C=>B
        LXI H,F200 // HL <= F200
UP:     MOV A,M // [HL] =>A
        INX H       // HL+1=>HL
        CMP M       // Compare reg M to A
        JC DOWN     // If A< M jump condition is true
```

```
        MOV D,M // M=> D
        MOV M,A   // A=>M
        DCX H       // HL-1 => Hl
        MOV M,D // D<=M
        INX H       // HL+1=>HL
DOWN:  DCR B       // Decrement b by 1
        JNZ UP      // Jump until B=0
        DCR C       // Decrement C by 1
        JNZ 2005    // Jump until C=0
        RST 1       // Terminate
# ORG F100H// Store number count at the address
# DB 10// Store Count
# ORG F200H// Store numbers at the address
# DB 01,02,AA,CC,03,04,11,22,BB,DD
```

INPUT & OUTPUT:



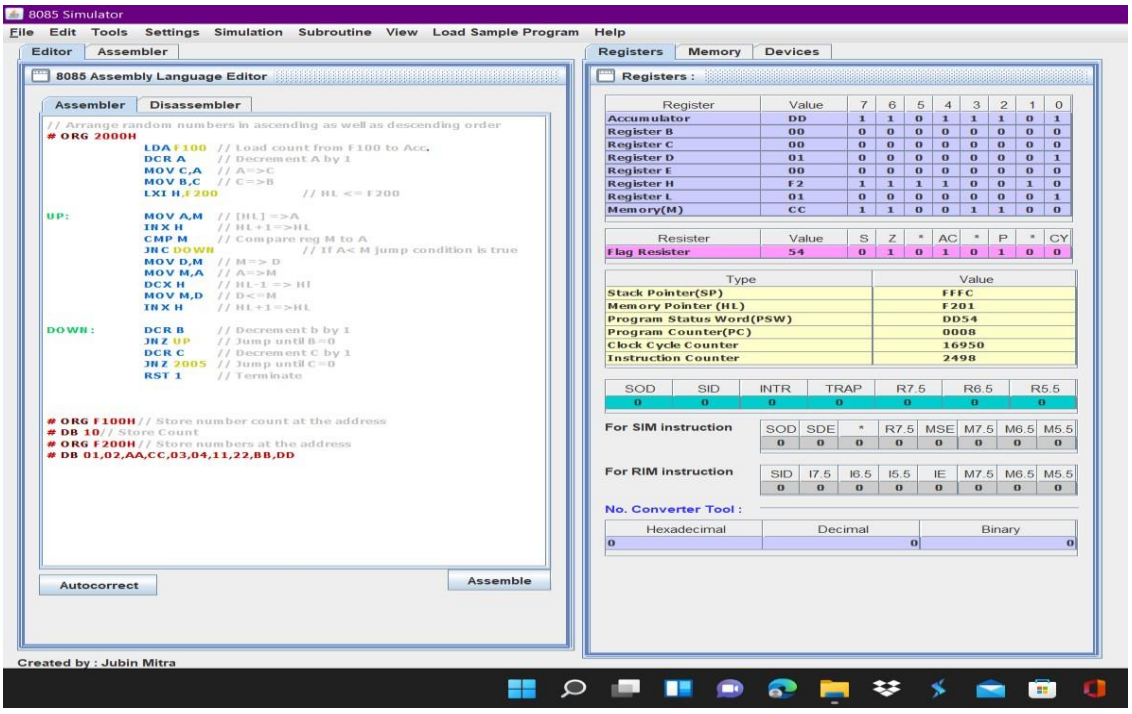


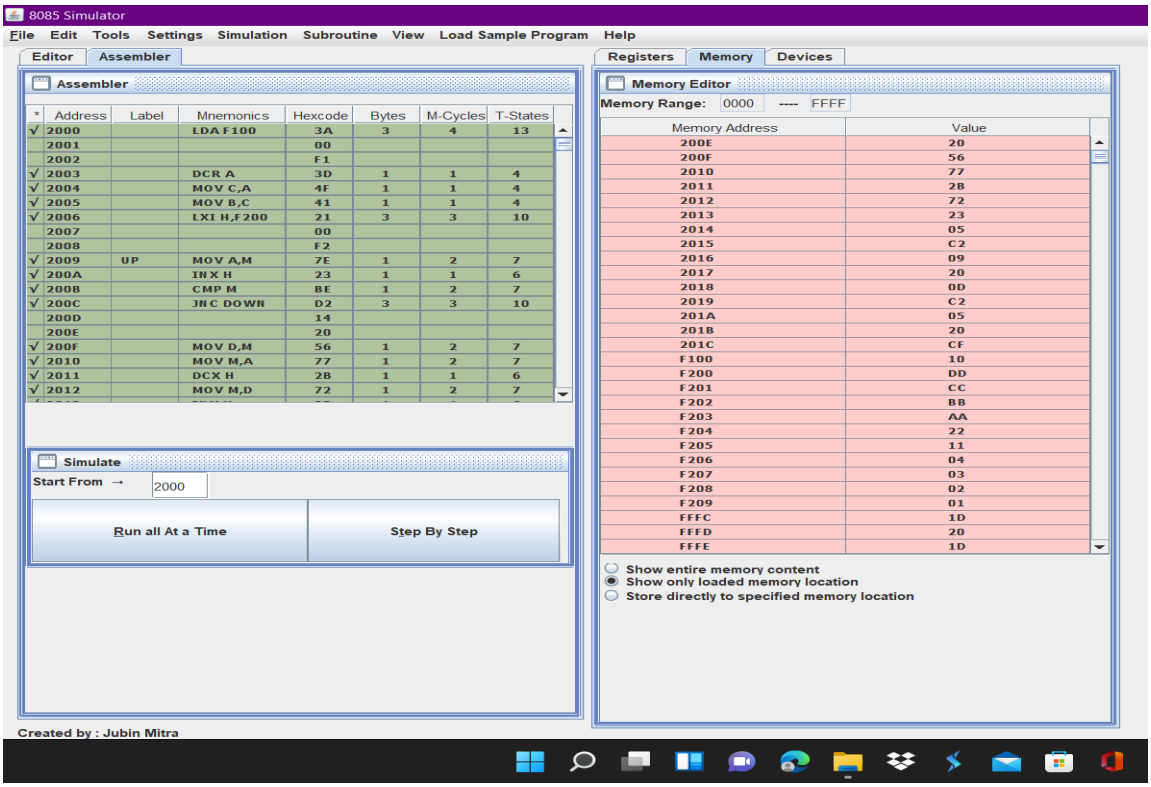
// Arrange random numbers in descending order

```
    # ORG 2000H
            LDA F100    // Load count from F100 to Acc.
            DCR A       // Decrement A by 1
            MOV C,A     // A=>C
            MOV B,C     // C=>B
            LXI H,F200 // HL <= F200
    UP:     MOV A,M // [HL] =>A
            INX H       // HL+1=>HL
            CMP M       // Compare reg M to A
            JNC DOWN// If A< M jump condition is true
            MOV D,M // M=> D
            MOV M,A  // A=>M
```

```
        DCX H       // HL-1 => Hl
        MOV M,D // D<=M
        INX H       // HL+1=>HL
DOWN: DCR B         // Decrement b by 1
        JNZ UP      // Jump until B=0
        DCR C       // Decrement C by 1
        JNZ 2005    // Jump until C=0
        RST 1       // Terminate
# ORG F100H// Store number count at the address
# DB 10// Store Count
# ORG F200H// Store numbers at the address
# DB 01,02,AA,CC,03,04,11,22,BB,DD
```

INPUT & OUTPUT:





F. Obtain Fibonacci Series.

CODE:

```
//Fibonacci Series.
        MVI A,00H
        STA C04FH
        MVI A,01H
        STA C050H
        MVI D,08
        LXI H,C04FH
BACK:  MOV A,M
        INX H
        ADD M
        INX H
```

MOV M,A

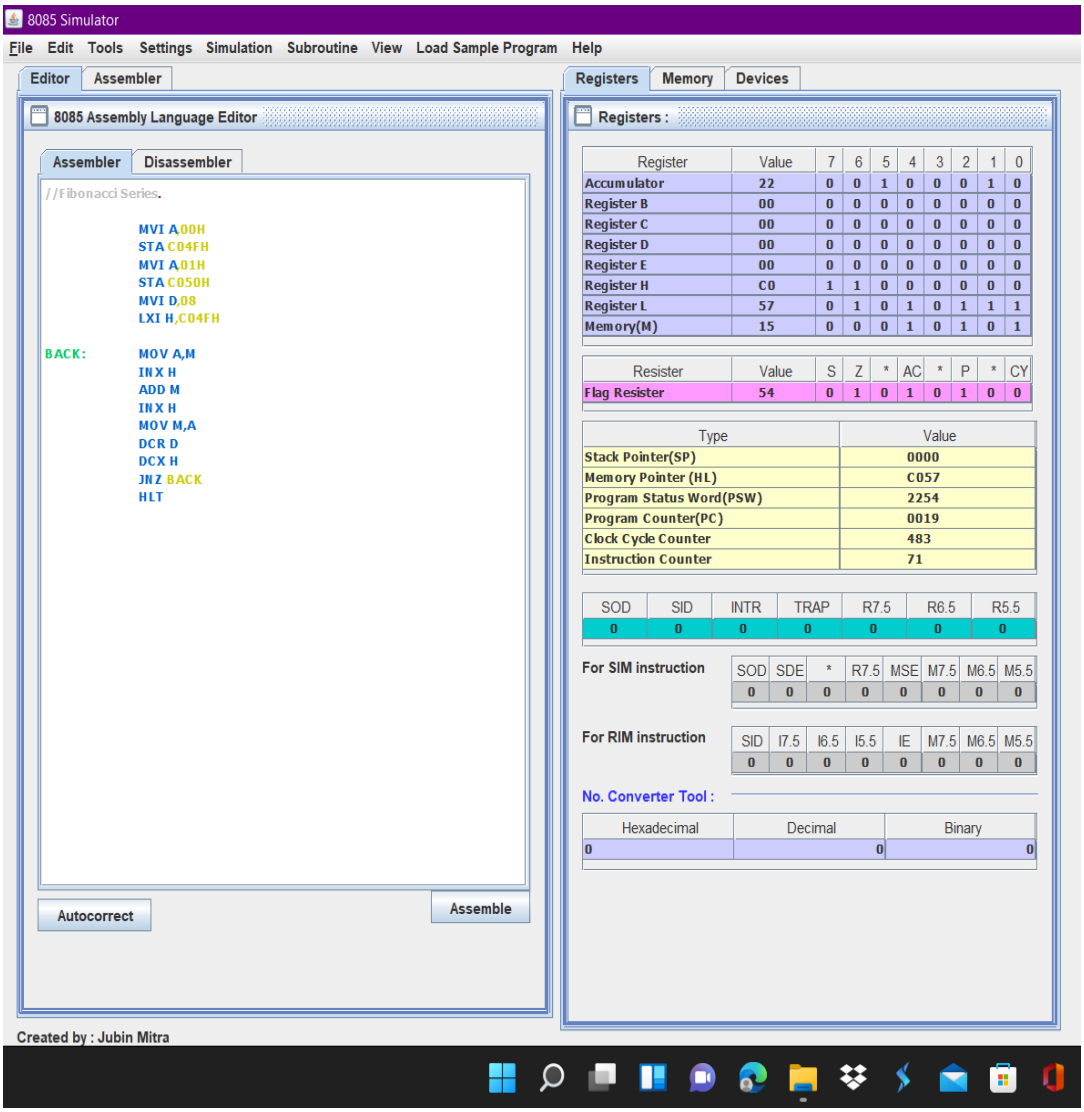
DCR D

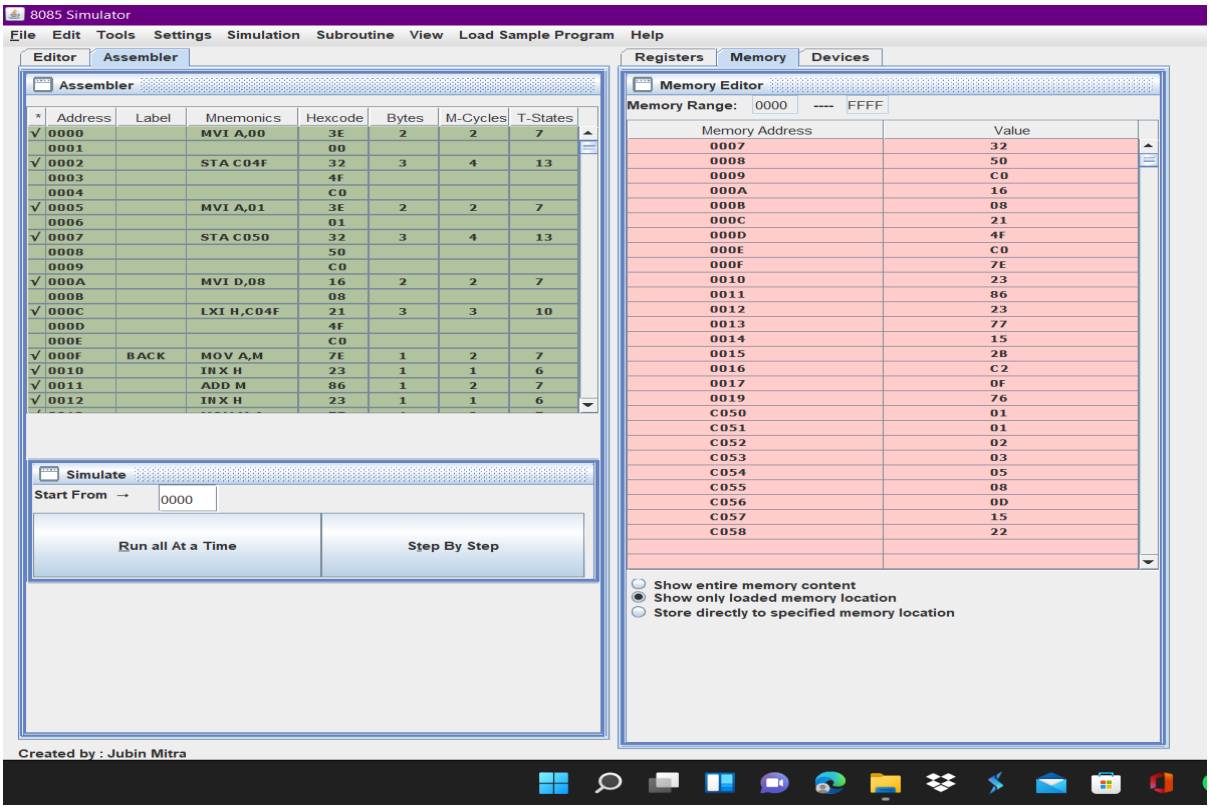
DCX H

JNZ BACK

HLT

INPUT & OUTPUT:





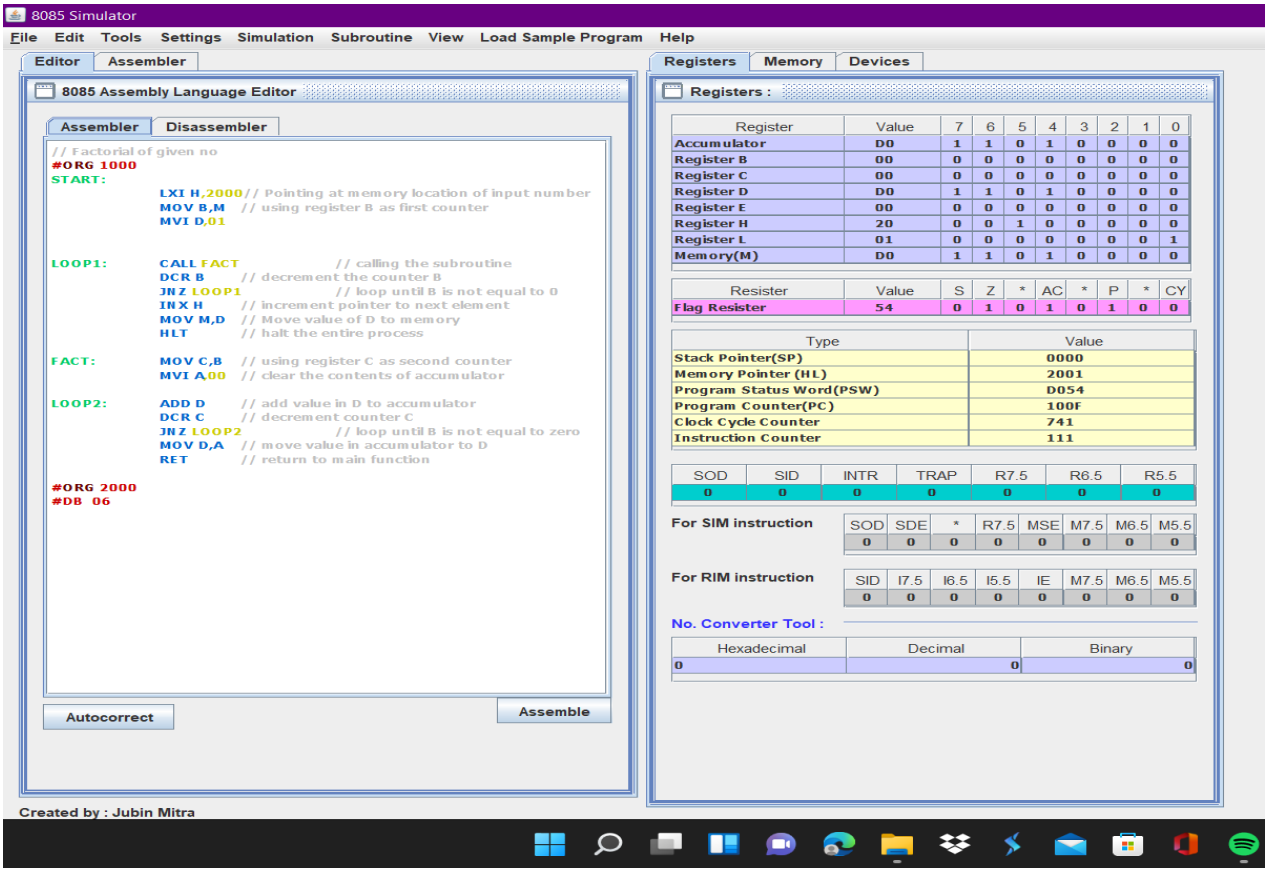
G. Obtain factorial of a number.

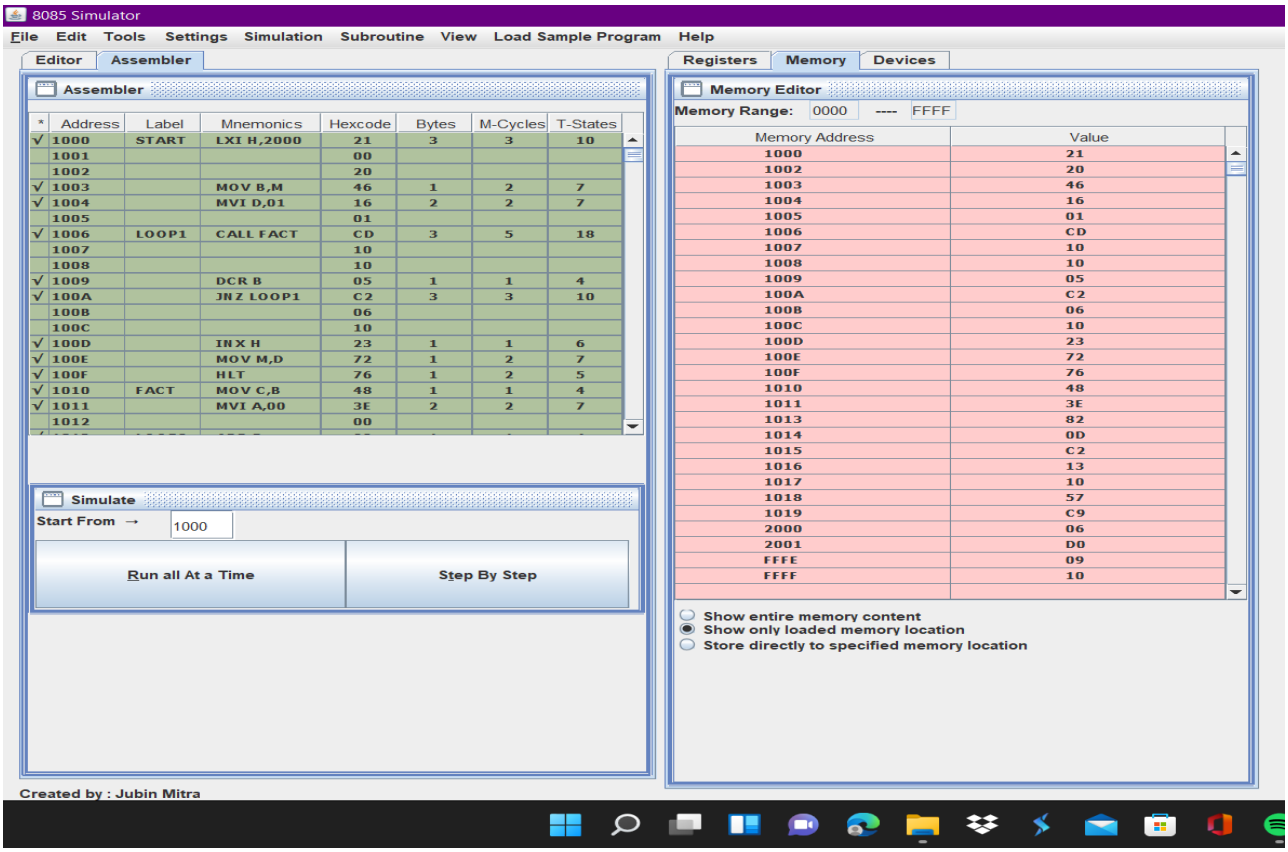
CODE:

```
// Factorial of given no

#ORG 1000

START:

        LXI H,2000// Pointing at memory location of input number

        MOV B,M   // using register B as first counter

        MVI D,01

LOOP1: CALL FACT          // calling the subroutine

        DCR B         // decrement the counter B

        JNZ LOOP1 // loop until B is not equal to 0

        INX H         // increment pointer to next element
```

```
        MOV M,D // Move value of D to memory
        HLT // halt the entire process
FACT:   MOV C,B    // using register C as second counter
        MVI A,00   // clear the contents of accumulator
LOOP2:          ADD D       // add value in D to accumulator
        DCR C       // decrement counter C
        JNZ LOOP2 // loop until B is not equal to zero
        MOV D,A // move value in accumulator to D
        RET // return to main function
#ORG 2000
#DB  06
```

INPUT & OUTPUT:





RESULTS:

TOPIC	INPUT	OUTPUT
Addition of two 8-bit no.	**7501=12H , 7502=13H**	**7503=25H**
Count no of 1's & 0's in give byte	**0AH**	**0=06, 1=02**
Division of two 8-bit no (without remainder) (with remainder)	**7501=02H, 7502=26H** **7501=02 , 7502=25H**	**7504= 13H** **7503=01H, 7504=12H**

Move block of Data	**3090 onward=05,10,15,20,25,30,35,40,45,50**	**30B0 onward= 05,10,15,20,25,30,35,40,45,50**
Arrange Random no Ascending & Descending	**F100=10 F200=01,02,AA,CC,03,04,11,22,BB,DD**	**F200Onward= 01,02,03,04,11,22,AA,BB,CC,DD F200Onward= DD,CC,BB,AA,22,11,04,03,02,01**
Fibonacci series	**C050H=01H**	**C050H Onward= 01,01,02,03,05.08,0D**
Factorial of given no	**2000=06H**	**2001=D0**

Conclusion:

Hence, we have performed assembly language for 8085 Processor by using virtual lab simulator of 8bit microprocessor 8085.

EXPERIMENT NO. 2

Aim: Write and execute the program to blink LED using 8051 microcontrollers, generate four random patterns on LED.

Theory: 8051 microcontrollers is designed by Intel in 1981. It is an 8-bit microcontroller. It is built with 40 pins DIP (dual inline package), 4kb of ROM storage and 128 bytes of RAM storage, 2 16-bit timers. It consists of are four parallel 8-bit ports, which are programmable as well as addressable as per the requirement. An on-chip crystal oscillator is integrated in the microcontroller having crystal frequency of 12 MHz

The microcontroller is used in-

- Television
- Washing Machines
- Telephones
- Other Electronic gadget

LED interfacing with 8051 microcontroller: A microcontroller, being an integrated circuit with a processor, contains support devices like program memory, data memory, I/O ports and serial communication interface integrated together. Since all the required support devices are available with the microcontroller, external interfacing of devices is not required. One of the most popular microcontrollers is Intel 8051. It belongs to the MCS-51 family of Intel microcontrollers. Many companies later adopted the MCS-51 core to develop their own microcontrollers and all these devices could be operated with the MCS-51 instruction sets. The basic difference between these devices is in the size of the memory or in the presence of an ADC or DAC.

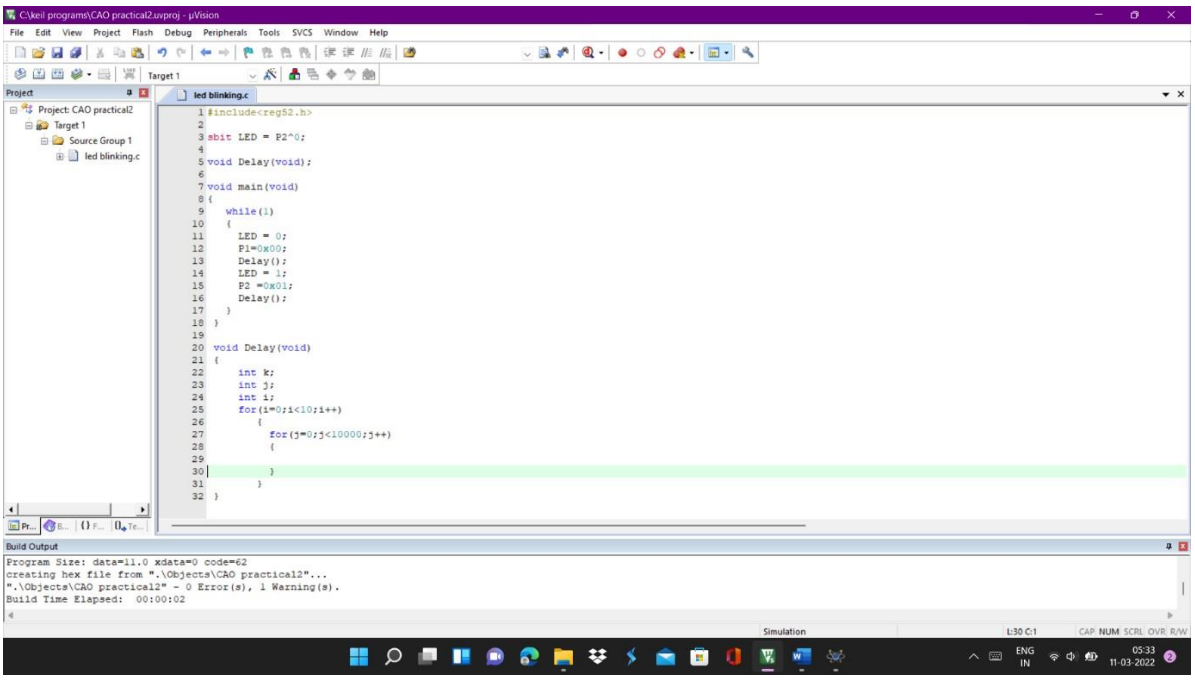
The main principle of this circuit is to interface LEDs to the 8051 family micro controller. Commonly, used LEDs will have voltage drop of 1.7v and current of 10mA to glow at full intensity. This is applied through the output pin of the micro controller.

Interface the Led with 8051 microcontroller in PROTEUS software and insert the screen shot of the same. Write the C program in KEIL and include the screen shot of program as well as output.

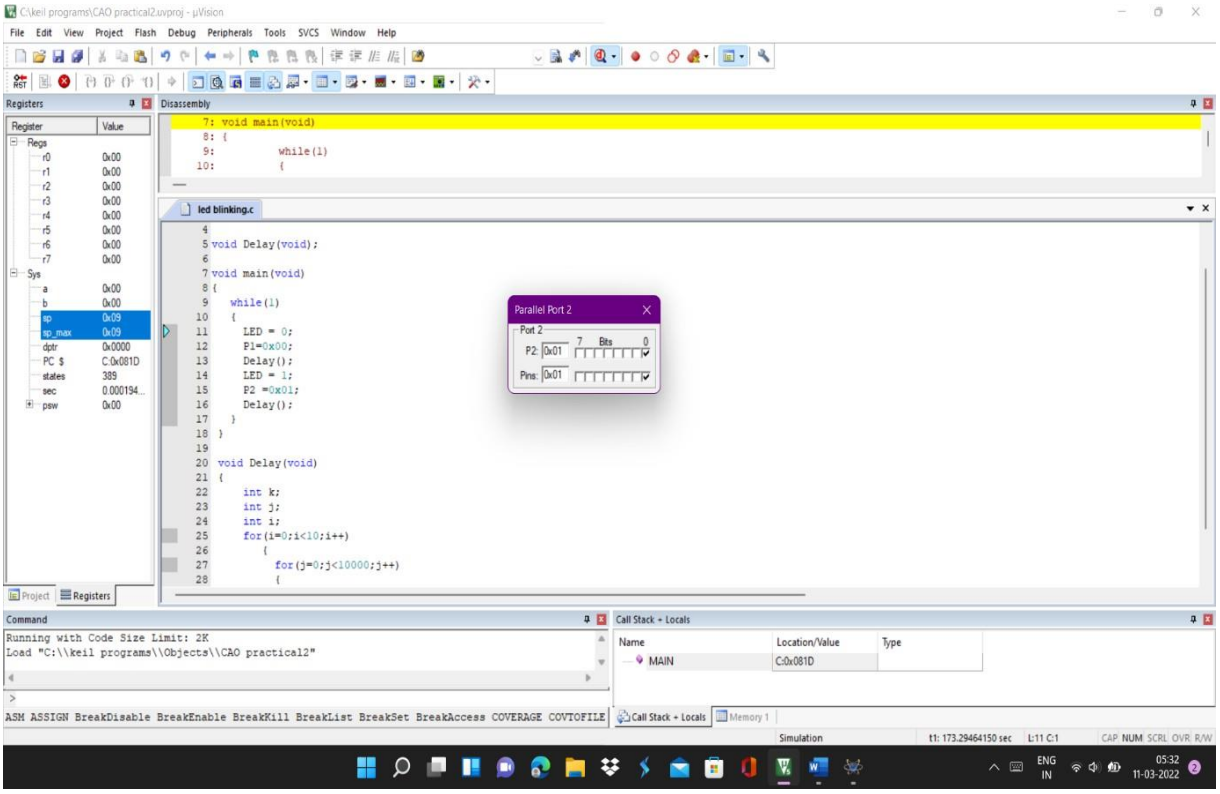
- **Blinking of single LED:**

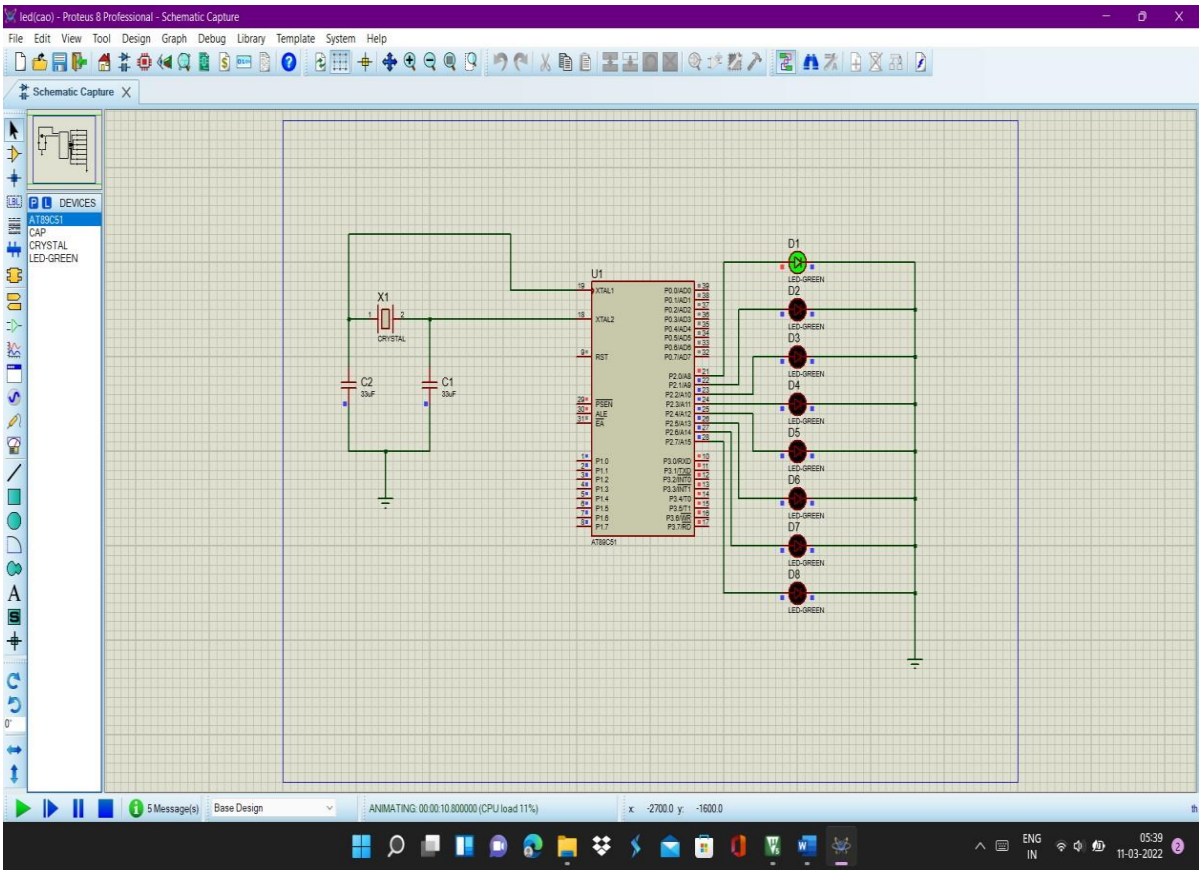
Code:

```
#include<reg52.h>
sbit LED = P2^0;
void Delay(void);
void main(void)
{
        while(1)
        {
                LED = 0;
   P1=0x00;
   Delay();
                LED = 1;
                P2 =0x01;
                Delay();
        }
 }

void Delay(void)
 {
          int k;
          int  j;
          int i;
          for(i=0;i<10;i++)
            {
              for(j=0;j<10000;j++)
              {

              }
            }
 }
```



Output:

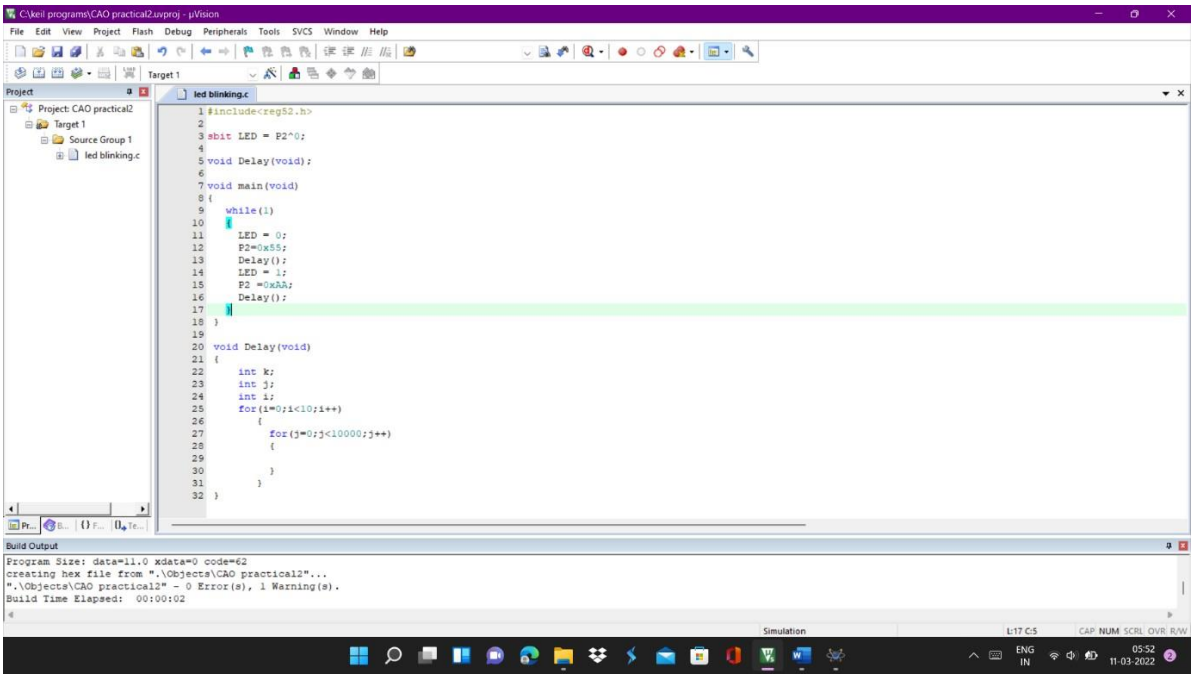




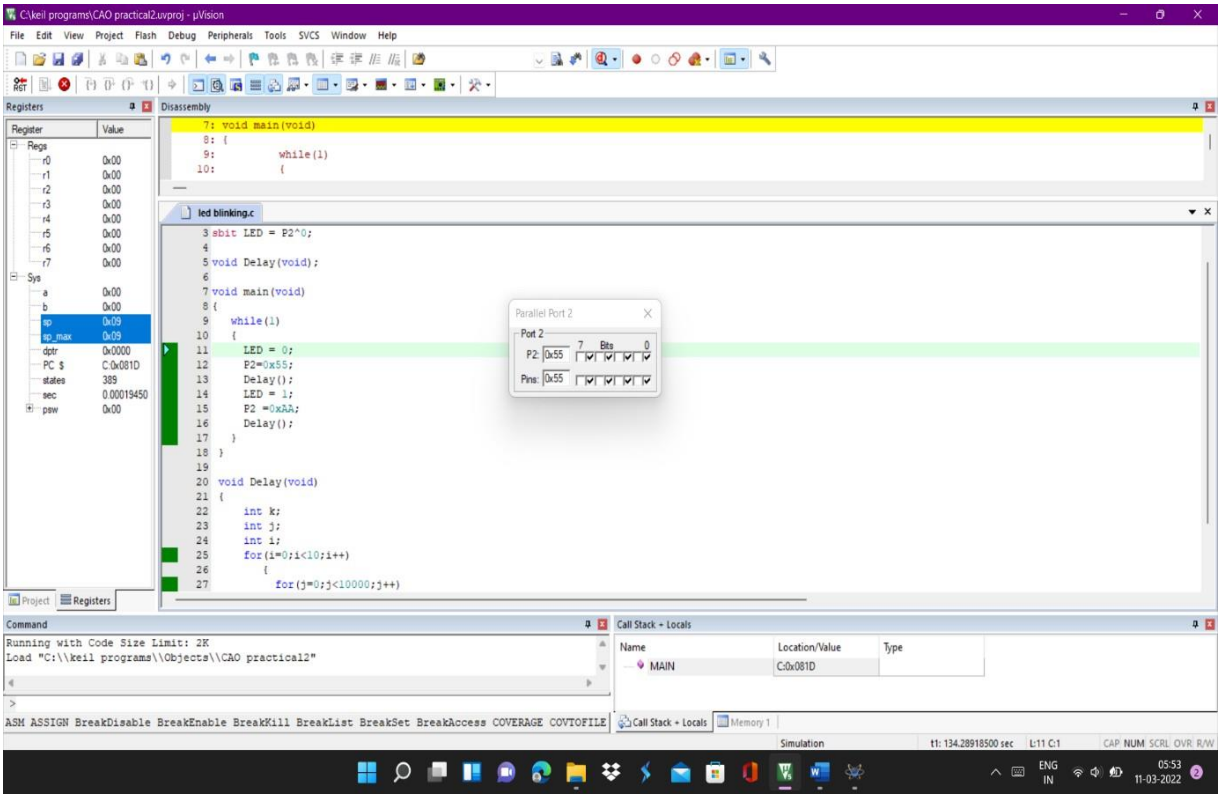
- **Blinking of LED alternatively**

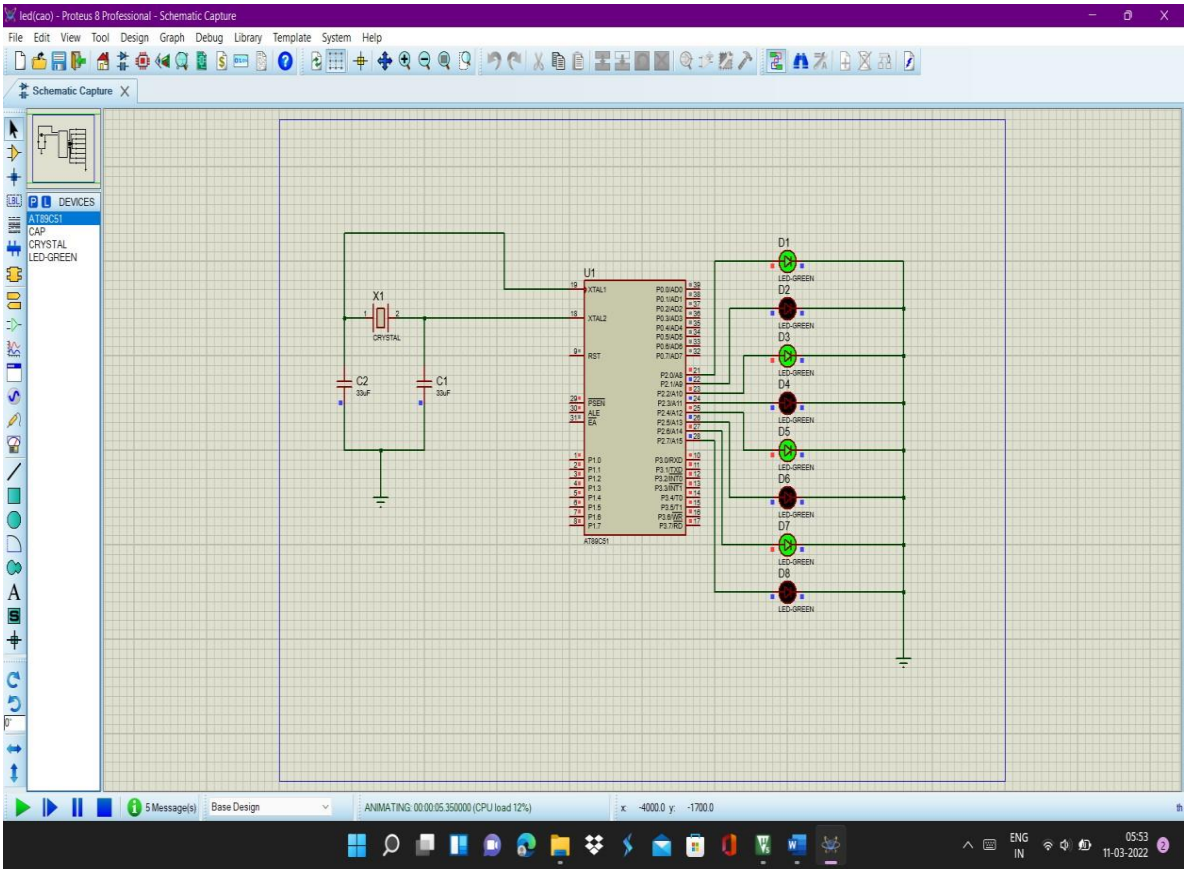
Code:

```
#include<reg52.h>
sbit LED = P2^0;
void Delay(void);
void main(void)
{
	while(1)
	{
		LED = 0;
		P1=0x55;
		Delay();
		LED = 1;
		P2 =0xAA;
		Delay();
	}
 }
void Delay(void)
 {
	int k;
	int  j;
	int i;
	for(i=0;i<10;i++)
	   {
		for(j=0;j<10000;j++)
		{

		}
	   }
 }
```



Output:

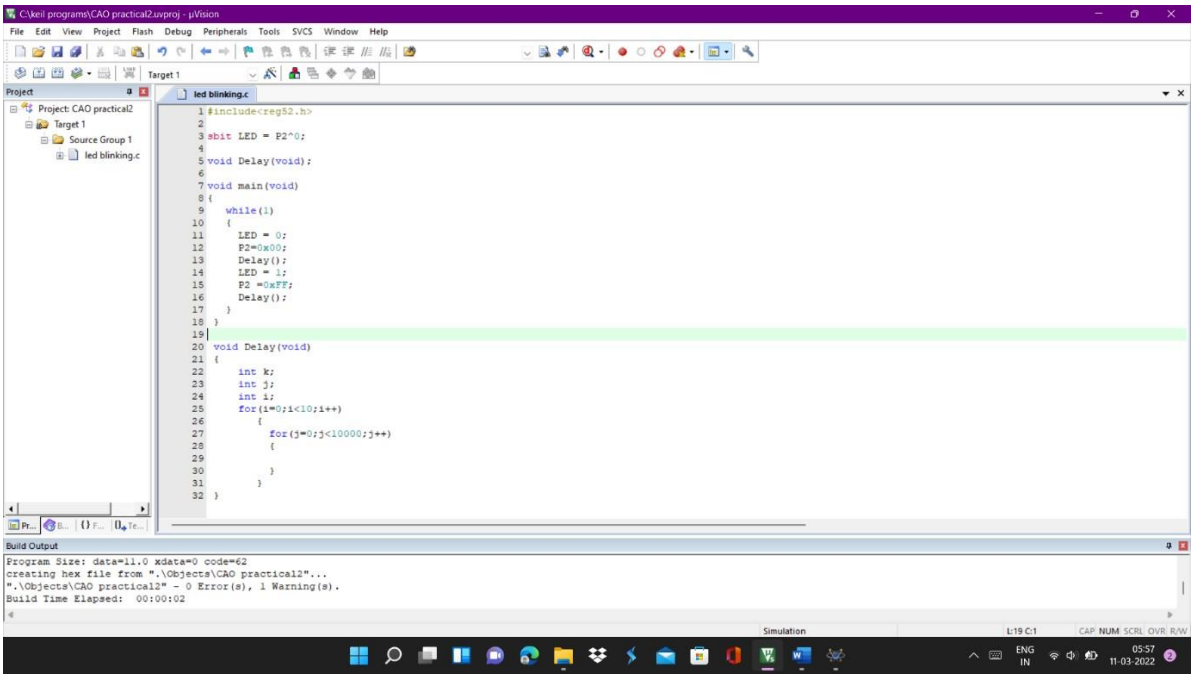




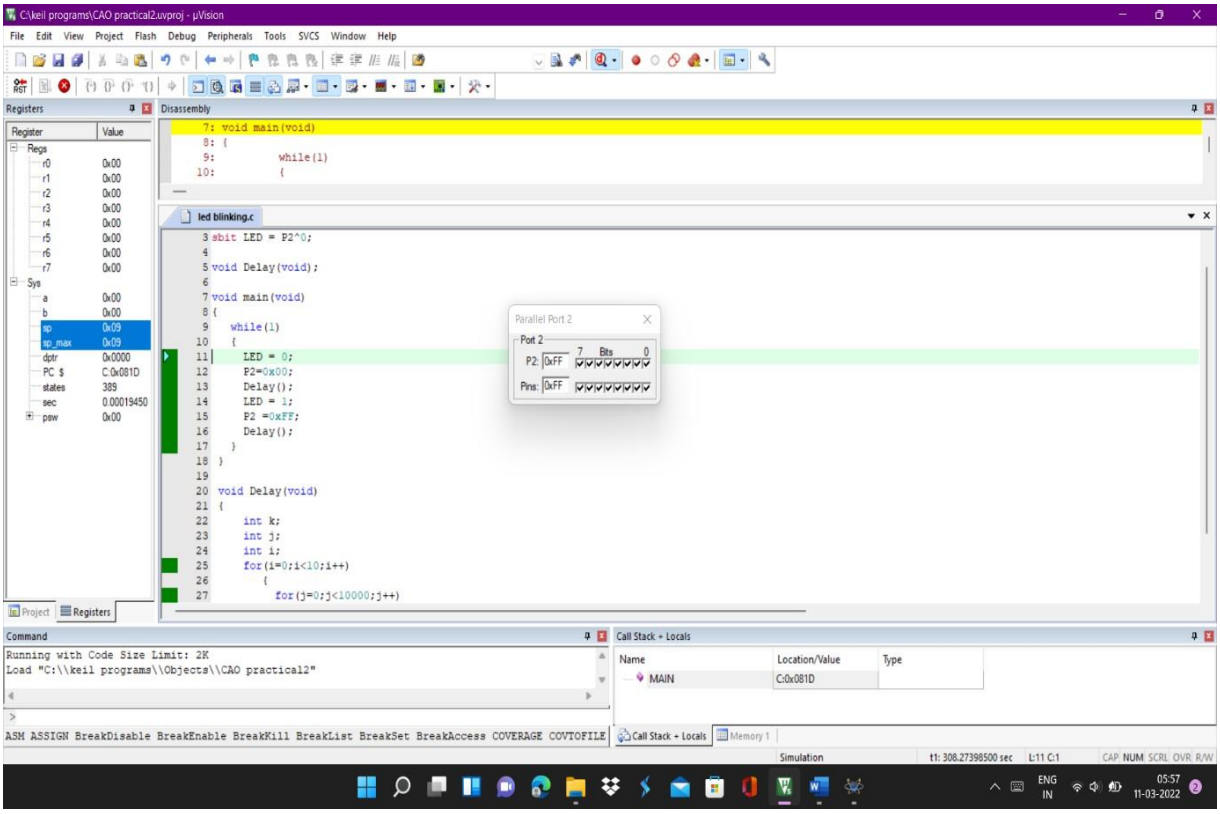
- **LED Flashing**

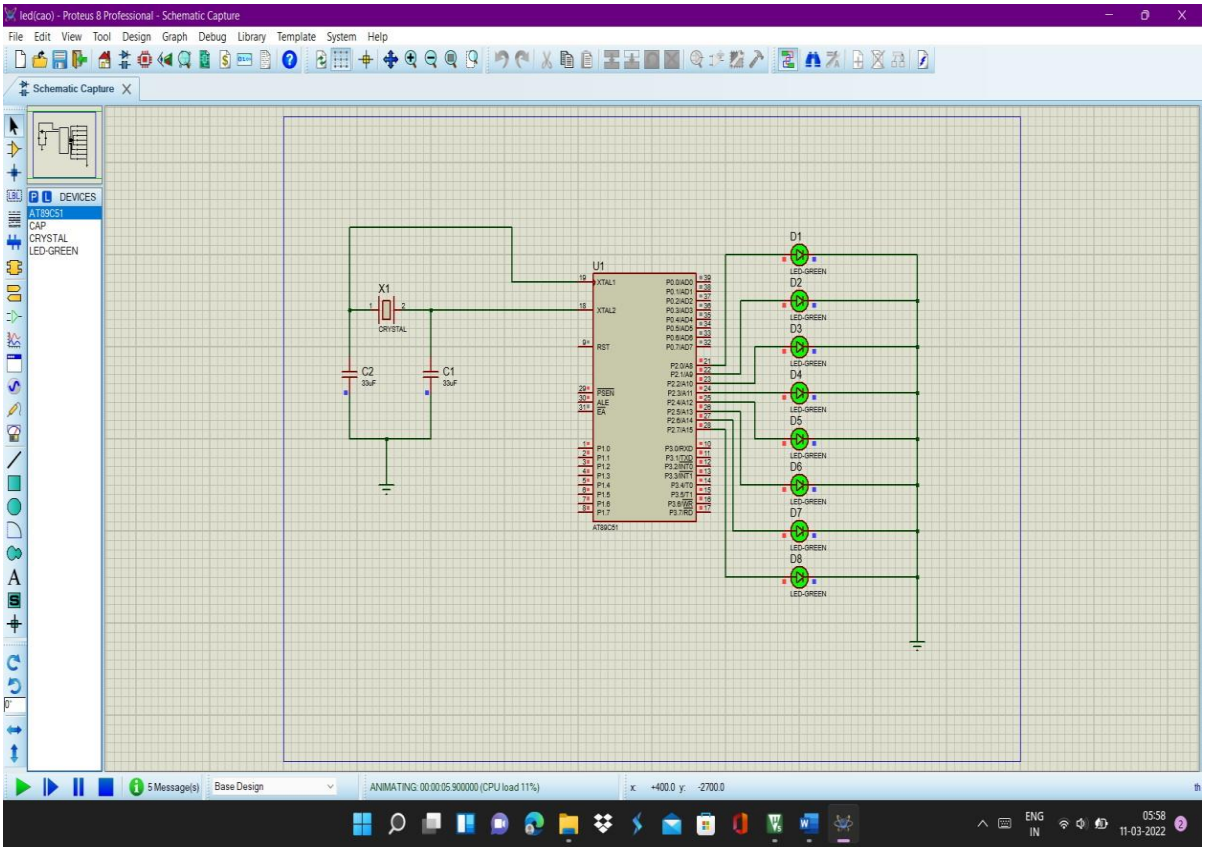
Code:

```
#include<reg52.h>
sbit LED = P2^0;
void Delay(void);
void main(void)
{
	while(1)
	{
		LED = 0;
		P2=0x00;
		Delay();
		LED = 1;
		P2 =0xFF;
		Delay();
	}
 }
void Delay(void)
 {
	int k;
	int  j;
	int i;
	for(i=0;i<10;i++)
	   {
		for(j=0;j<10000;j++)
		{

		}
	   }
 }
```



Output:

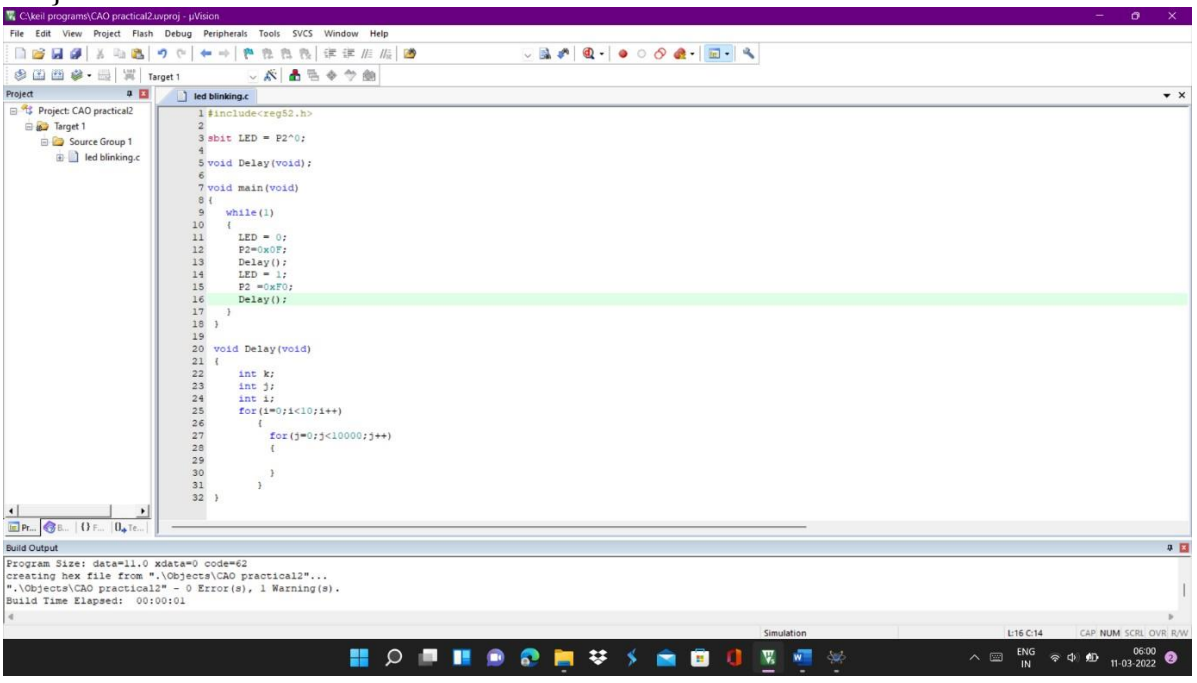




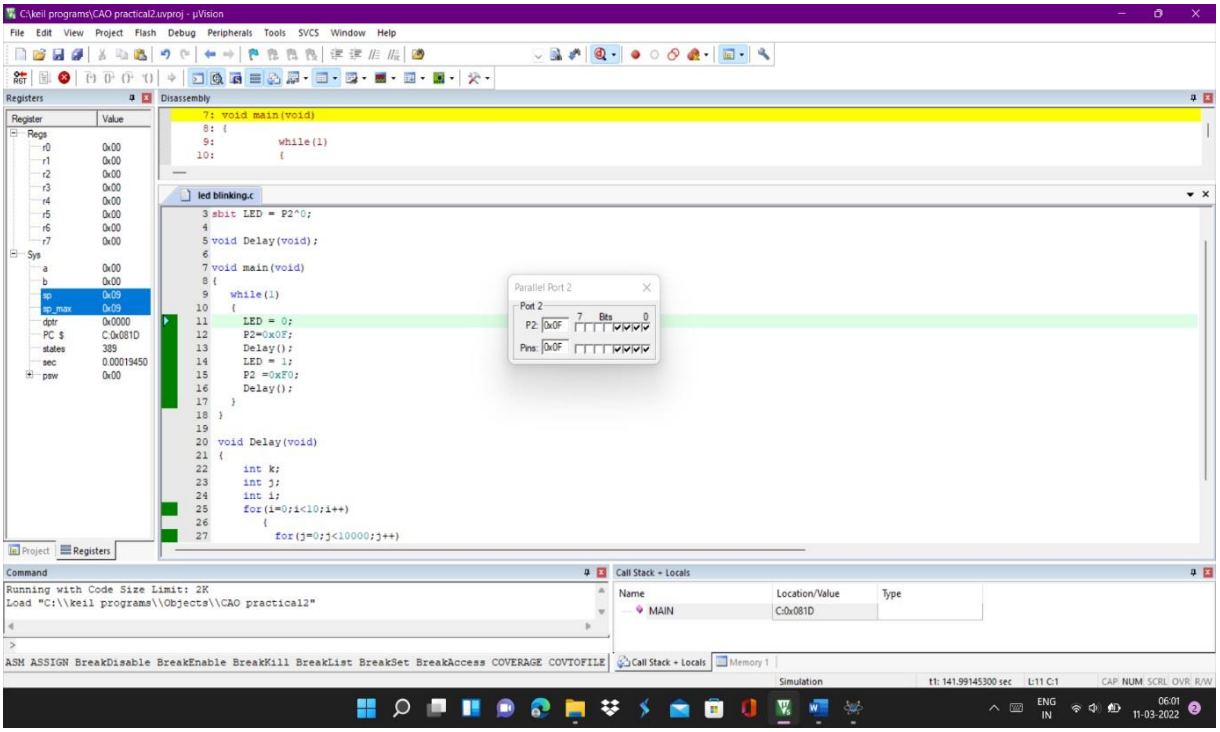
- **4 LEDs on 4 LEDs off**

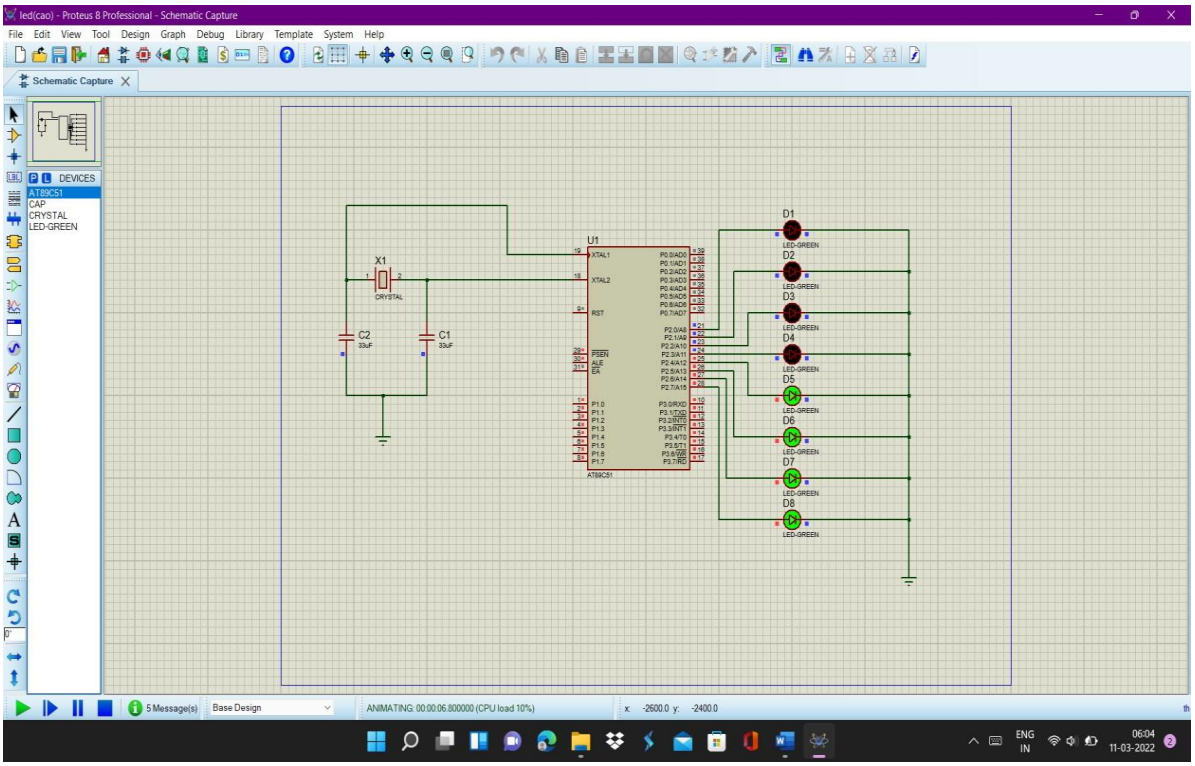
Code:

```
#include<reg52.h>
sbit LED = P2^0;
void Delay(void);
void main(void)
{
	while(1)
	{
		LED = 0;
		P2=0x0F;
		Delay();
		LED = 1;
		P2 =0xF0;
		Delay();
	}
 }
void Delay(void)
 {
	int k;
	int  j;
	int i;
	for(i=0;i<10;i++)
	   {
		for(j=0;j<10000;j++)
		{

		}
	   }
 }
```



Output:





Conclusion: Here, we have performed the blinking of LEDs in using 8051 Microcontroller.

EXPERIMENT NO. 3

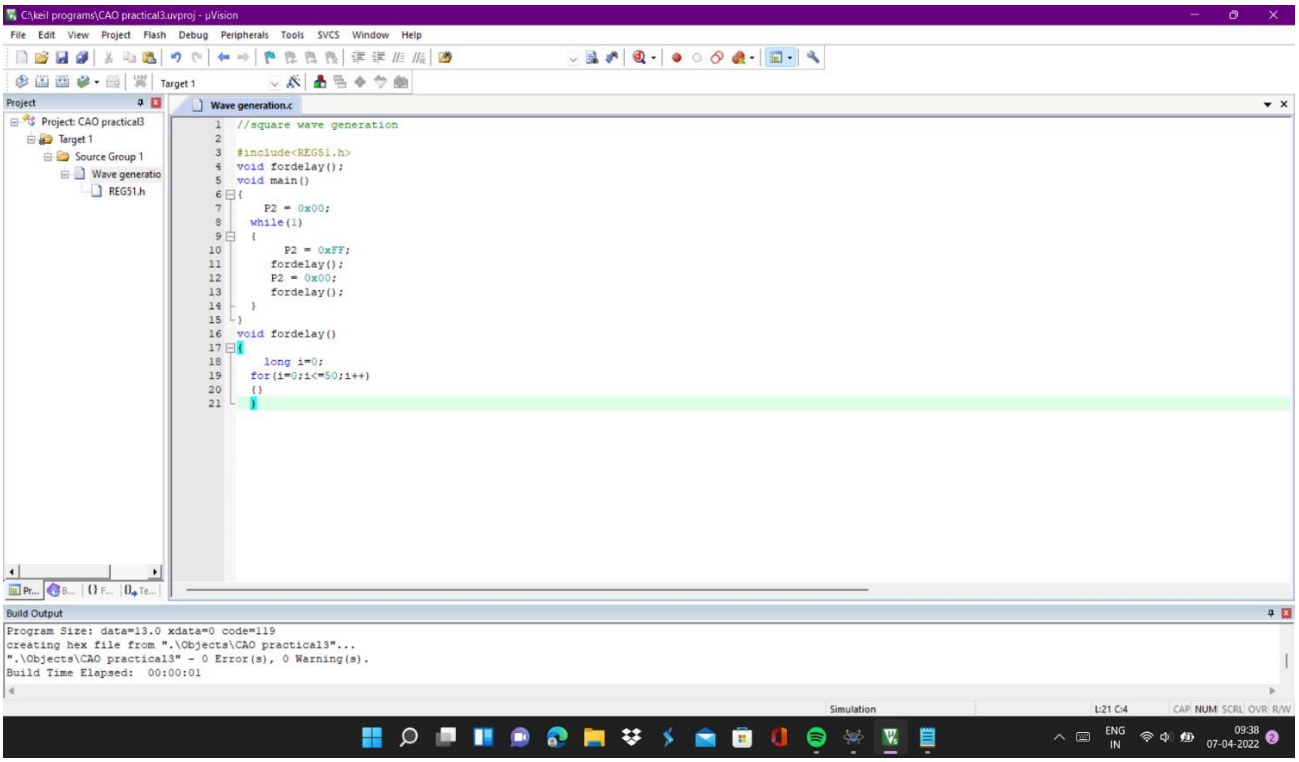
Aim: Write and Execute a program for generation of Square, Rectangular, Sine, Cosine, Sawtooth and triangular waveform for 8051.

Theory: To generate the various types of waveforms like sine, cosine, sawtooth, triangular and square we have to interface Digital to Analog converter (DAC) with 8051 microcontrollers.

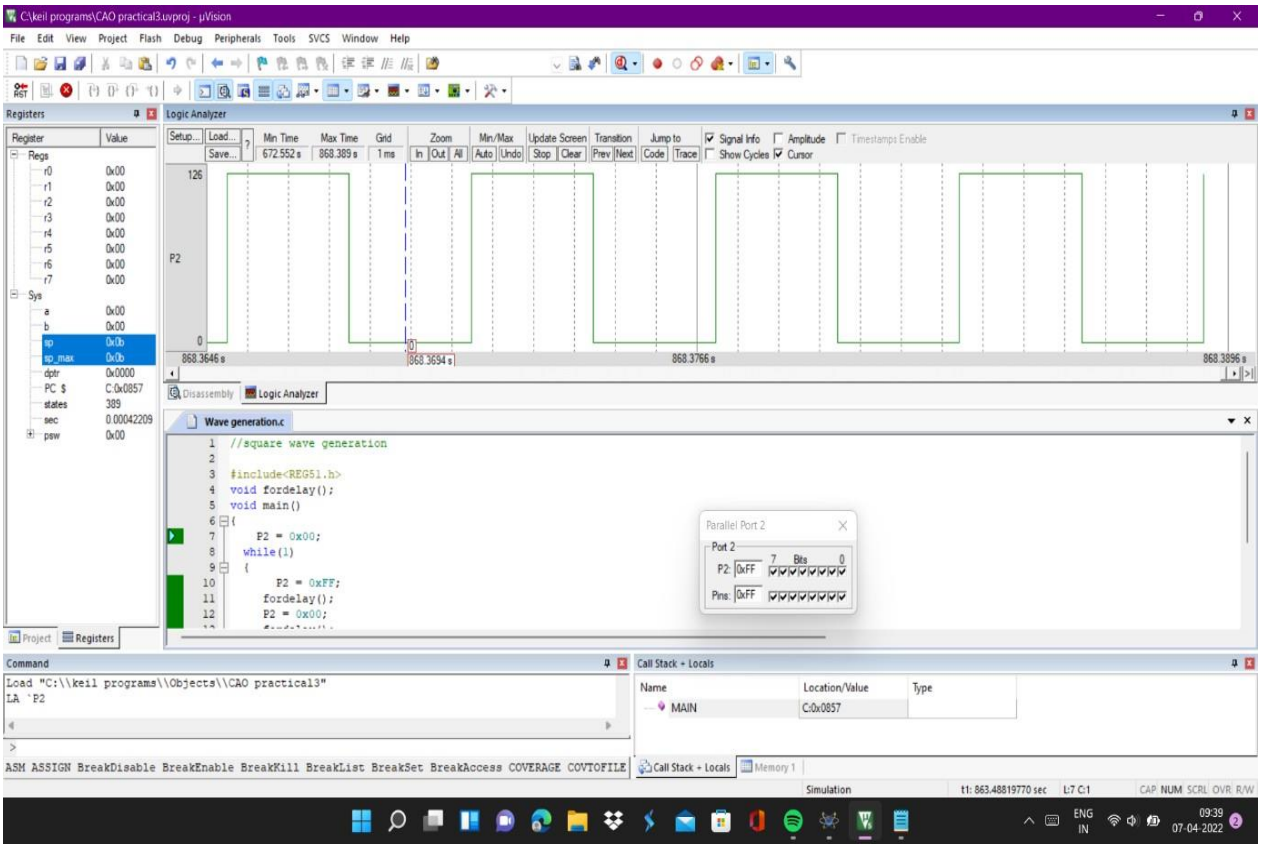
1. Square wave

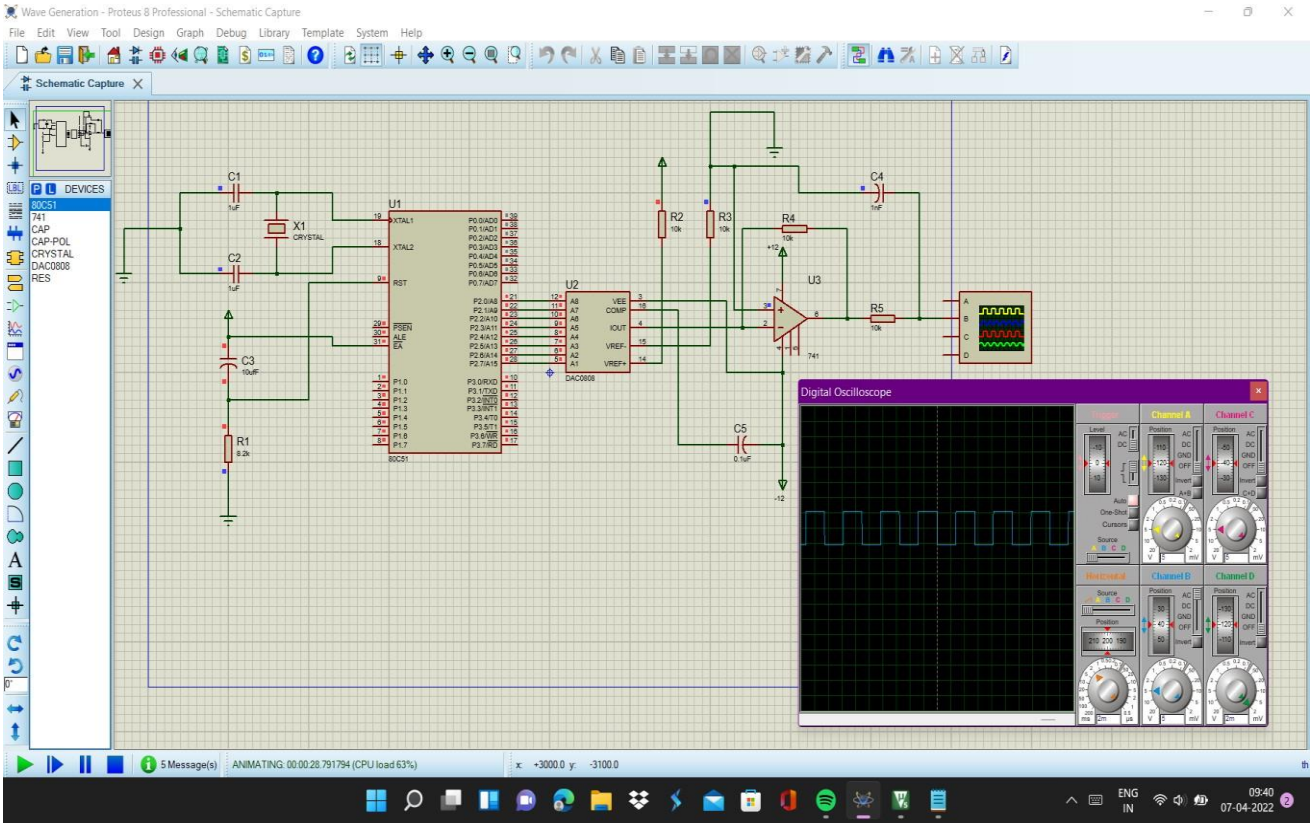
Code:

```
//square wave generation
#include<REG51.h>
void fordelay();
void main()
{
P2 = 0x00;
while(1)
{
P2 = 0xFF;
fordelay();
P2 = 0x00;
fordelay();
}
}
void fordelay()
{
long i=0;
for(i=0;i<=5000;i++)
{}
}
```


C:\keil programs\CAO practical3.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Project: CAO practical3
Target 1
Source Group 1
Wave generatio
REG51.h
Wave generation.c
1 //square wave generation
2
3 #include<REG51.h>
4 void fordelay();
5 void main()
6 {
7 P2 = 0x00;
8 while(1)
9 {
10 P2 = 0xFF;
11 fordelay();
12 P2 = 0x00;
13 fordelay();
14 }
15 }
16 void fordelay()
17 {
18 long i=0;
19 for(i=0;i<=50;i++)
20 {}
21 }
Build Output
Program Size: data=13.0 xdata=0 code=119
creating hex file from ".\Objects\CAO practical3"...
".\Objects\CAO practical3" - 0 Error(s), 0 Warning(s).
Build Time Elapsed: 00:00:01
Simulation
L:21 C:4
CAP NUM SCRL OVR R/W
ENG
IN
09:38
07-04-2022

Output :

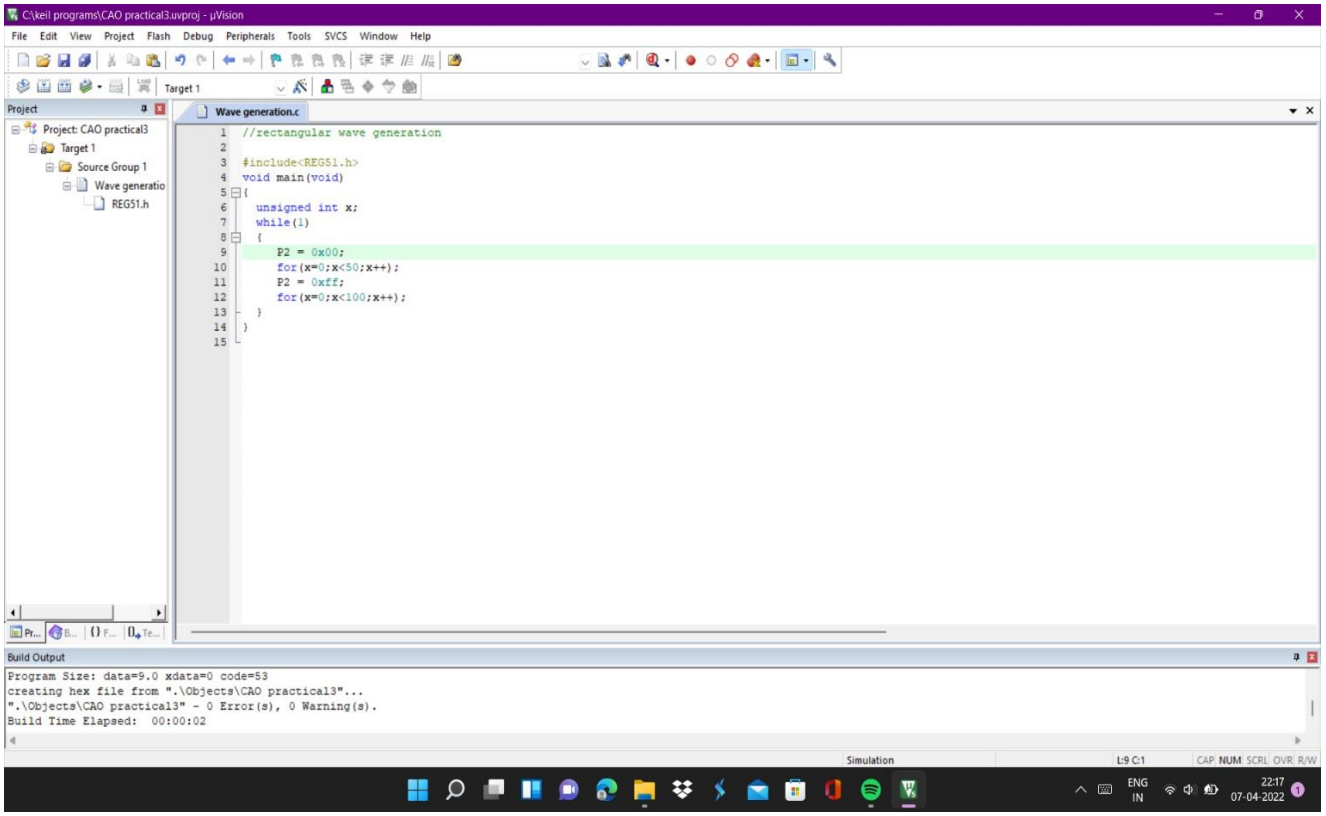

C:\keil programs\CAO practical3.uvproj - µVision
Logic Analyzer
Wave generation.c
//square wave generation
#include<REG51.h>
void fordelay();
void main()
P2 = 0x00;
while(1)
P2 = 0xFF;
fordelay();
P2 = 0x00;
Parallel Port 2
Command
Load "C:\\keil programs\\Objects\\CAO practical3"
LA `P2
Call Stack + Locals
MAIN


Wave Generation - Proteus 8 Professional - Schematic Capture
Schematic Capture
DEVICES
80C51
741
CAP
CAP-POL
CRYSTAL
DAC0808
RES
U1
U2
U3
X1
C1
C2
C3
C4
C5
R1
R2
R3
R4
R5
Digital Oscilloscope

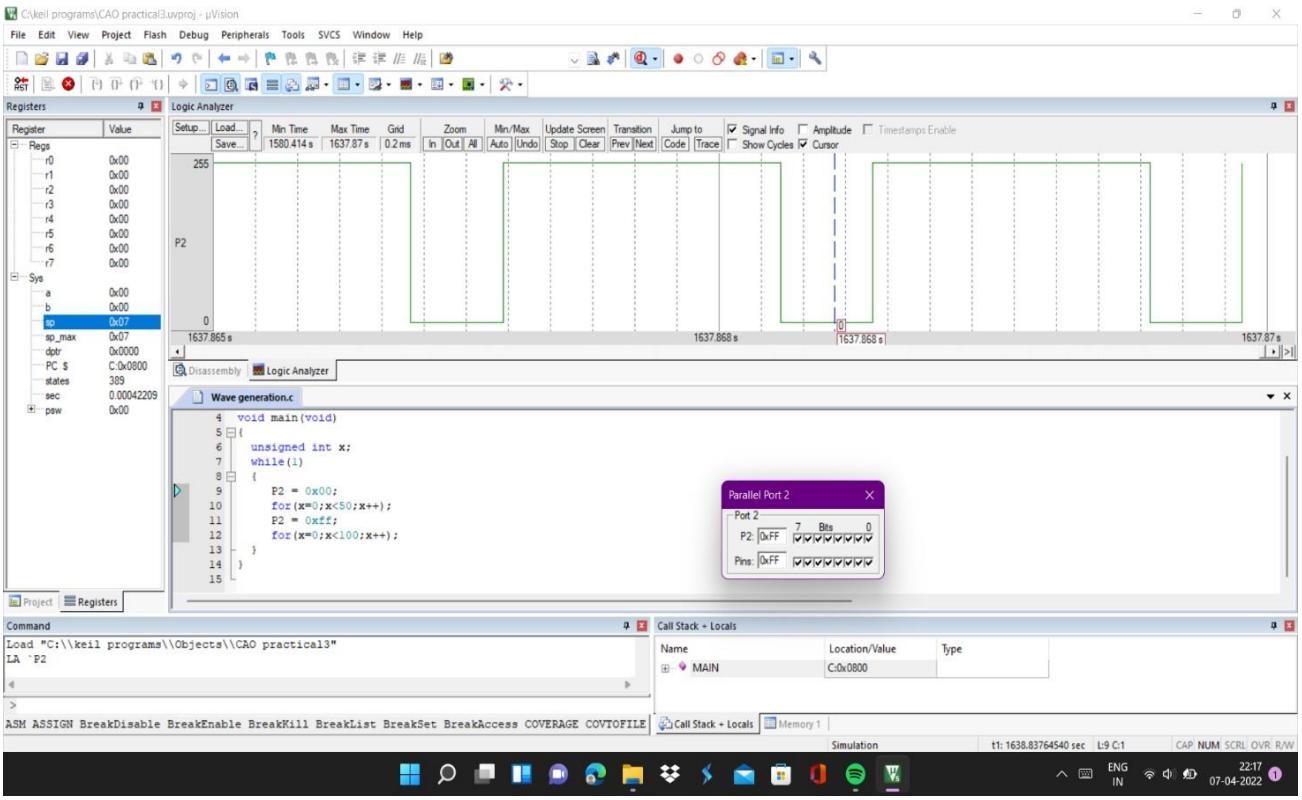
2.Rectangular wave:

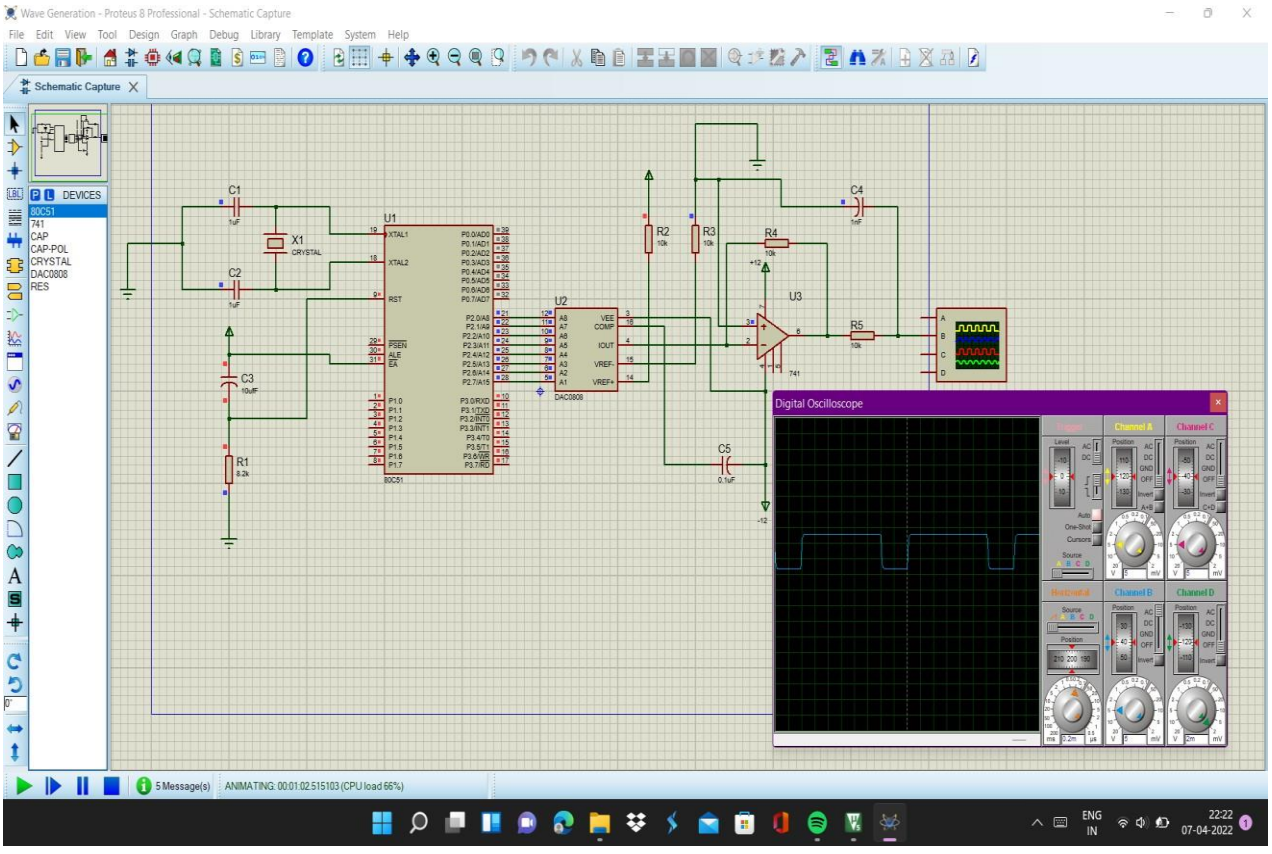
Code:

```
//rectangular wave generation
#include<REG51.h>
void main(void)
{
unsigned int x;
while(1)
{
P2 = 0x00;
for(x=0;x<50;x++);
P2 = 0xFF;

for(x=0;x<100;x++);
}
}
```



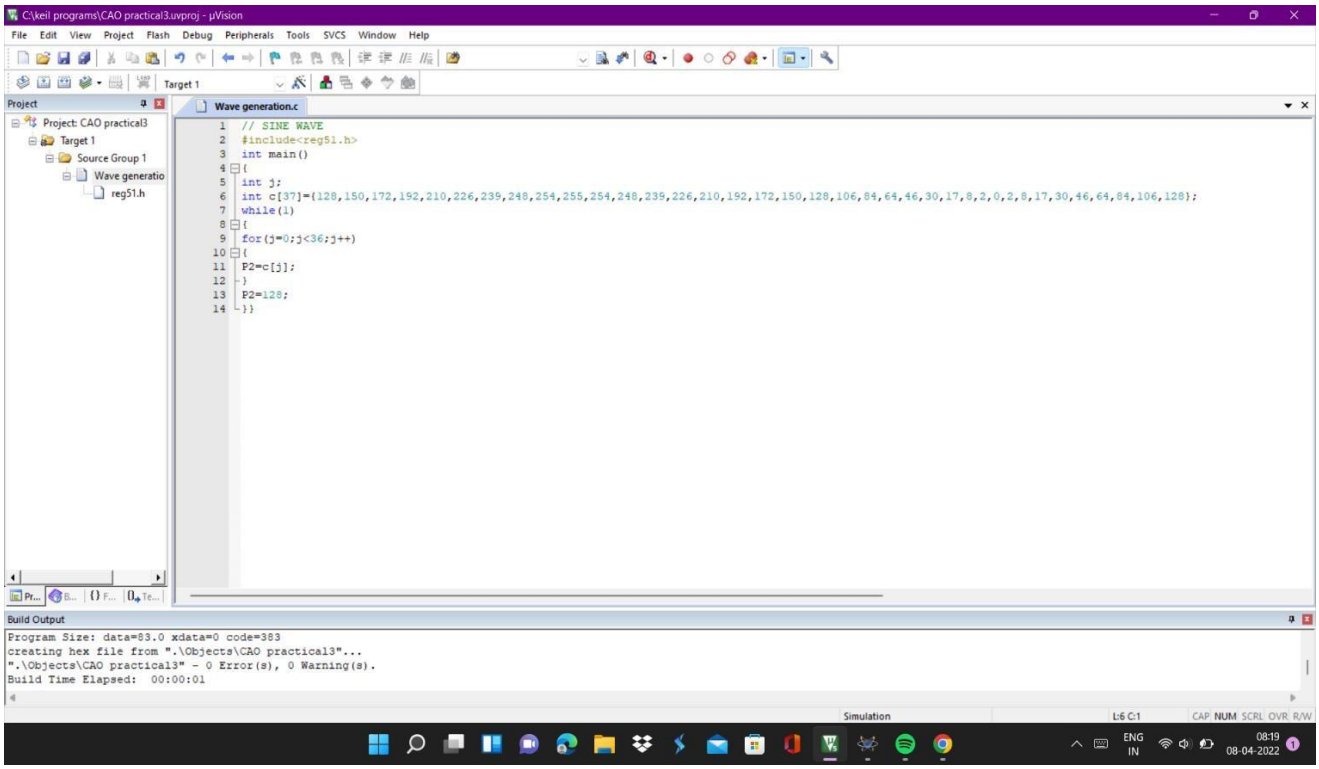
Output:


C:\keil programs\CAO practical3.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Logic Analyzer
Wave generation.c
void main(void)
{
unsigned int x;
while(1)
{
P2 = 0x00;
for(x=0;x<50;x++);
P2 = 0xff;
for(x=0;x<100;x++);
}
}
Parallel Port 2
Command
Load "C:\\keil programs\\Objects\\CAO practical3"
LA `P2
Call Stack + Locals
Name
Location/Value
Type
MAIN
C:0x0800
ASM ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE COVTOFILE

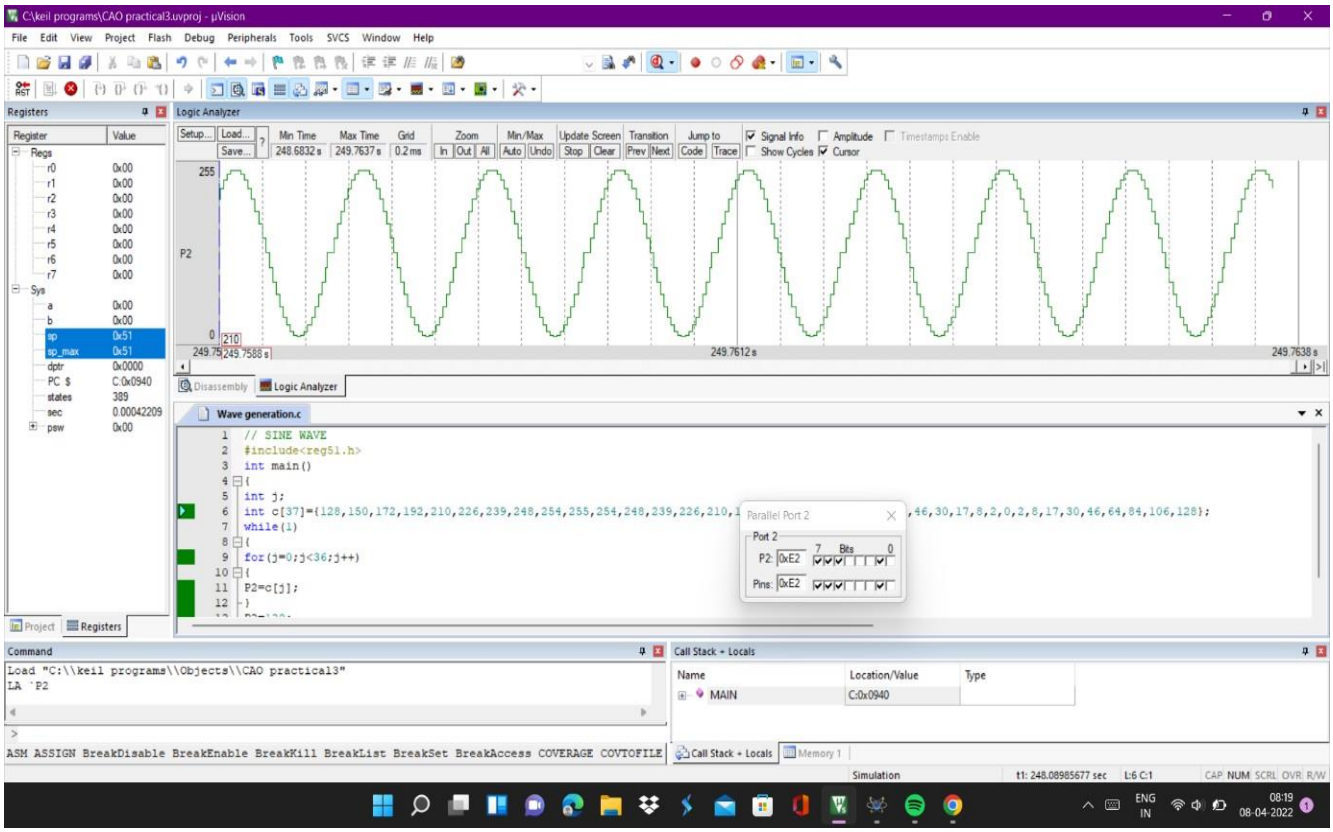

Wave Generation - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
DEVICES
80C51
741
CAP
CAP-POL
CRYSTAL
DAC0808
RES
U1
U2
U3
X1
C1
C2
C3
C4
C5
R1
R2
R3
R4
R5
Digital Oscilloscope
5 Message(s)
ANIMATING: 00:01:02.515103 (CPU load 66%)

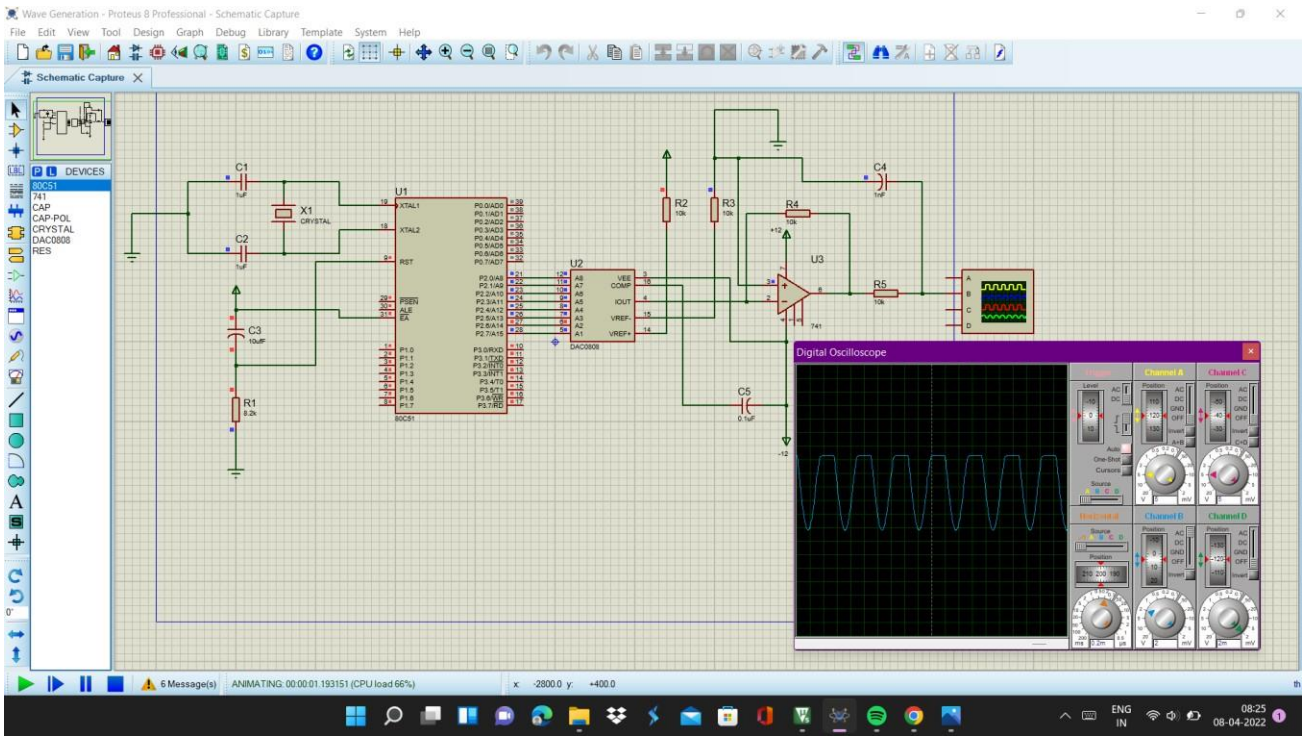
3.Sine wave

```
//sine wave generation
#include<reg51.h>
int main()
{
int j;
int
c[37]={128,150,172,192,210,226,239,248,254,255,254,248,239,226,210,192,172,150
,128,106,84,64,46,30,17,8,2,0,2,8,17,30,46,64,84,106,128};
while(1)
{
for(j=0;j<36;j++)
{
P2=c[j];
}
P2=128;
}}
```



Output:

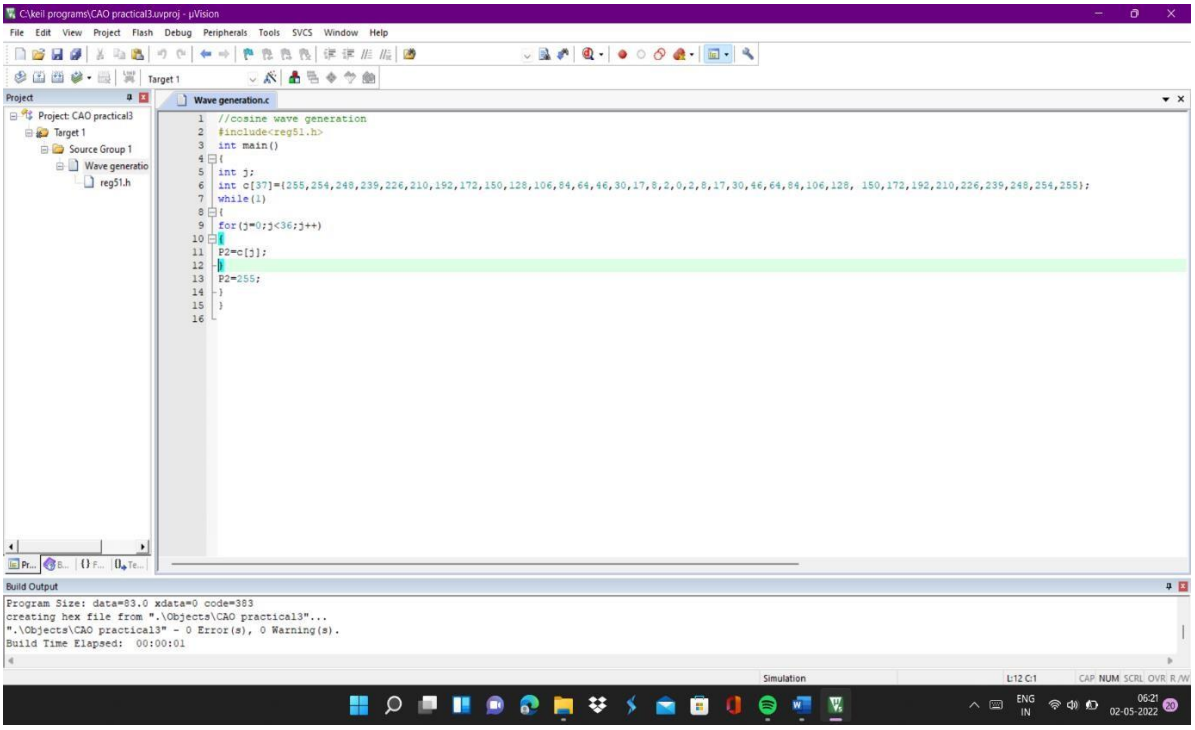

C:\keil programs\CAO practical3.uvproj - µVision
Logic Analyzer
Wave generation.c
// SINE WAVE
#include<reg51.h>
int main()
Parallel Port 2
Command
Load "C:\\keil programs\\Objects\\CAO practical3"
LA `P2
Call Stack + Locals
Name
Location/Value
Type
MAIN
C:0x0940


Wave Generation - Proteus 8 Professional - Schematic Capture
Schematic Capture
DEVICES
80C51
741
CAP
CAP-POL
CRYSTAL
DAC0808
RES
Digital Oscilloscope

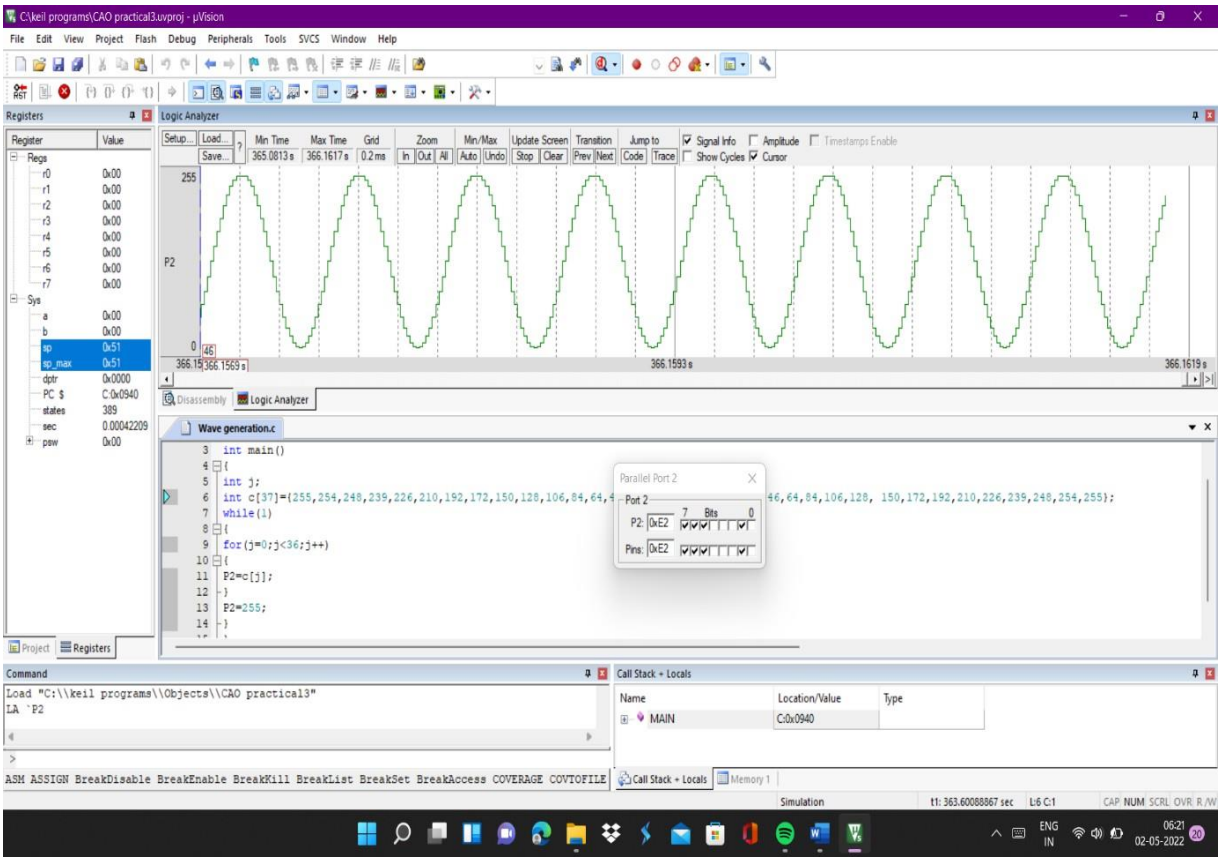
4.Cosine wave :

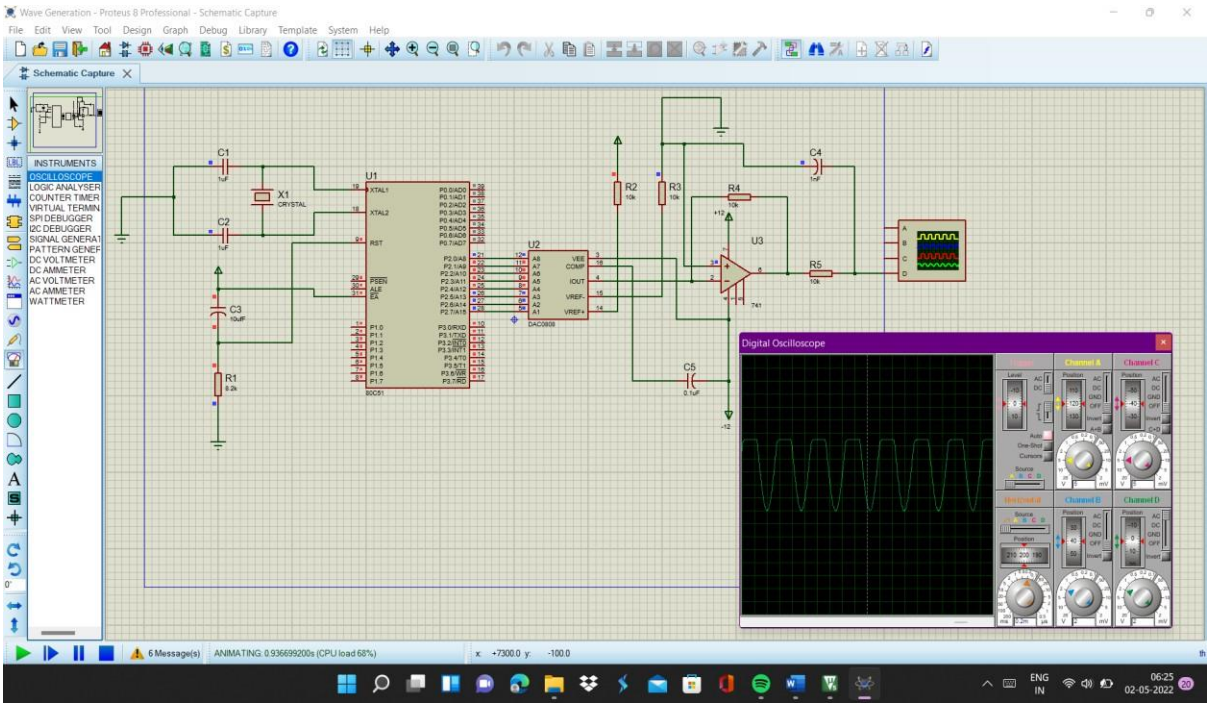
Code:

```
//cosine wave generation
#include<reg51.h>
int main()
{
int j;
int
c[37]={255,254,248,239,226,210,192,172,150,128,106,84,64,46,30,17,8,2,0,2,8,17,3
0,46,64,84,106,128, 150,172,192,210,226,239,248,254,255};
while(1)
{
for(j=0;j<36;j++)
{
P2=c[j];
}
P2=255;
}}
```



Output:

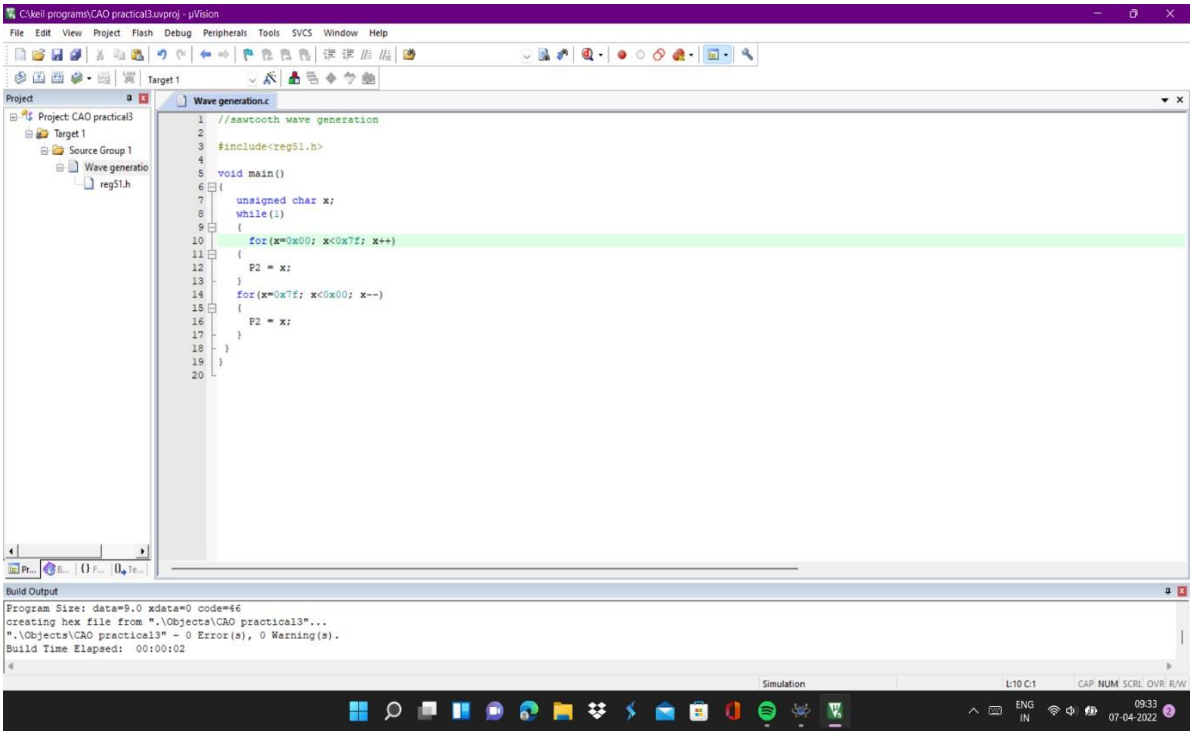

C:\keil programs\CAO practical3.uvproj - µVision
Logic Analyzer
Wave generation.c
int main()
{
int j;
int c[37]={255,254,248,239,226,210,192,172,150,128,106,84,64,
while(1)
for(j=0;j<36;j++)
P2=c[j];
P2=255;
Parallel Port 2
Command
Load "C:\\keil programs\\Objects\\CAO practical3"
LA `P2
Call Stack + Locals
Name
Location/Value
Type
MAIN
C:0x0940


Wave Generation - Proteus 8 Professional - Schematic Capture
Schematic Capture
INSTRUMENTS
OSCILLOSCOPE
LOGIC ANALYSER
COUNTER TIMER
VIRTUAL TERMIN
SPI DEBUGGER
I2C DEBUGGER
SIGNAL GENERAT
PATTERN GENER
DC VOLTMETER
DC AMMETER
AC VOLTMETER
AC AMMETER
WATTMETER
Digital Oscilloscope
6 Message(s)
ANIMATING: 0.936699200s (CPU load 68%)

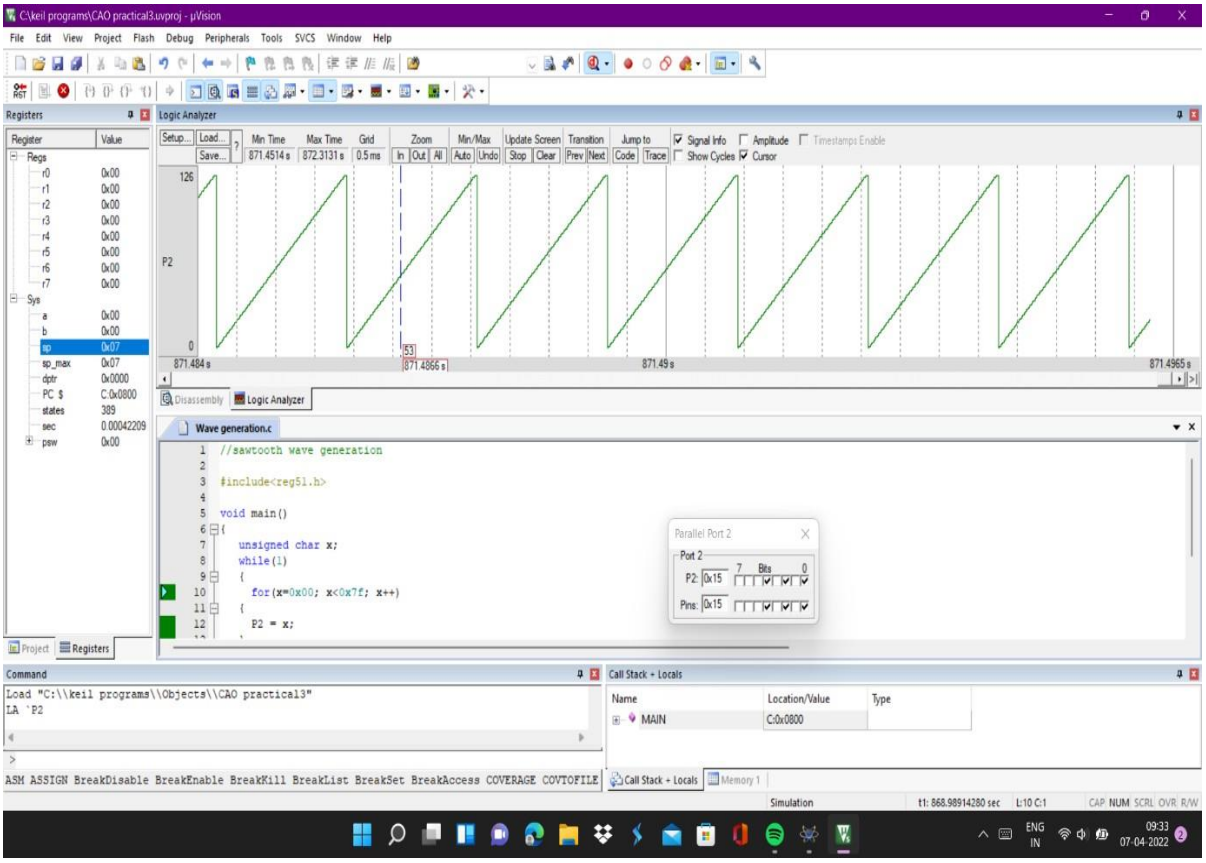
5.Sawtooth wave:

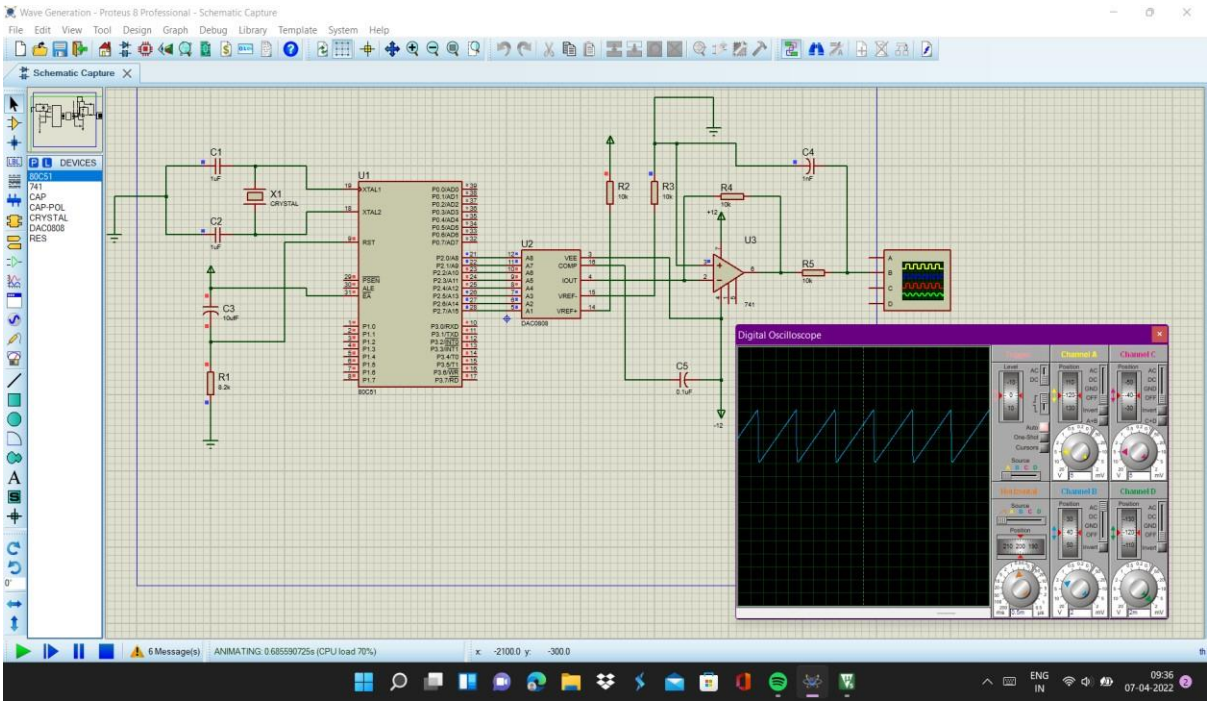
Code:

```
//sawtooth wave generation
#include<reg51.h>
void main ()
{
  unsigned char x;
  while(1)
      {
     for(x=0x00; x<0x7f; x++)
       {
          P2 = x;
        }
     for(x=0x7f; x<0x00; x--)
        {
          P2 = x;
        }
      }
      }
```



Output:

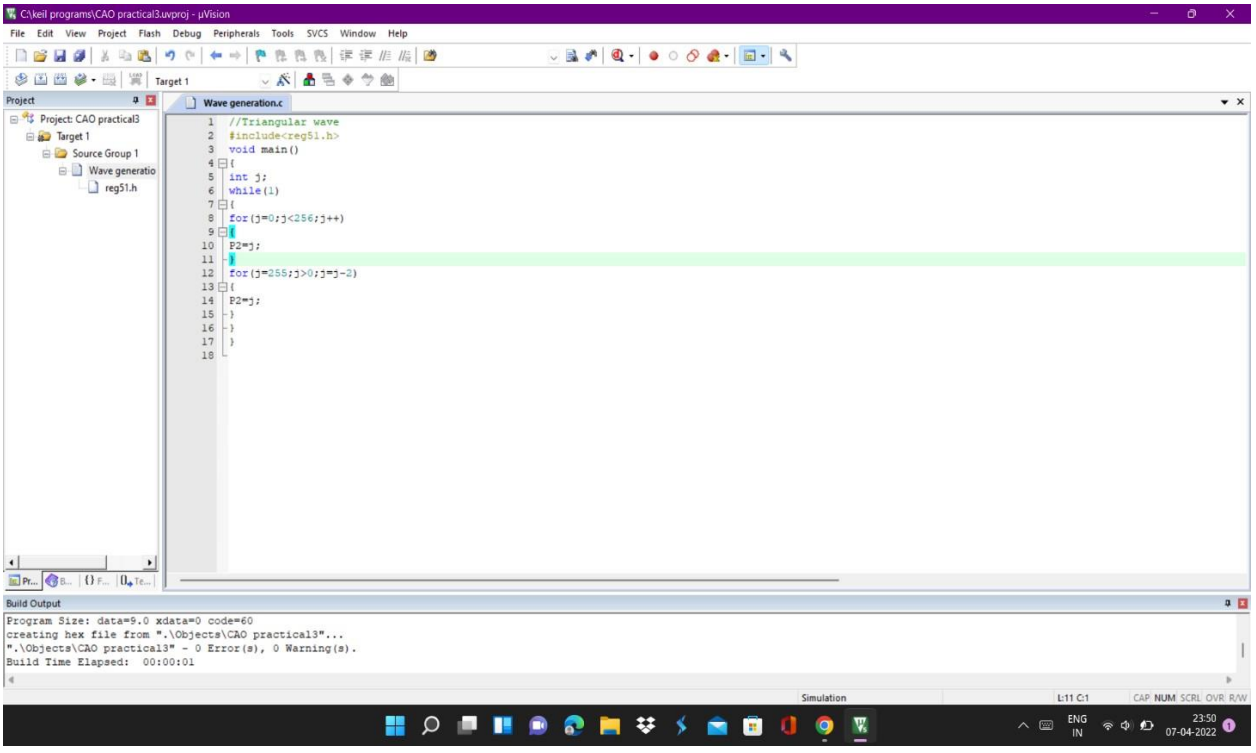

C:\keil programs\CAO practical3.uvproj - µVision
Logic Analyzer
Wave generation.c
//sawtooth wave generation
#include<reg51.h>
void main()
unsigned char x;
while(1)
for(x=0x00; x<0x7f; x++)
P2 = x;
Parallel Port 2
Command
Load "C:\\keil programs\\Objects\\CAO practical3"
LA `P2
Call Stack + Locals


Wave Generation - Proteus 8 Professional - Schematic Capture
Schematic Capture
DEVICES
Digital Oscilloscope

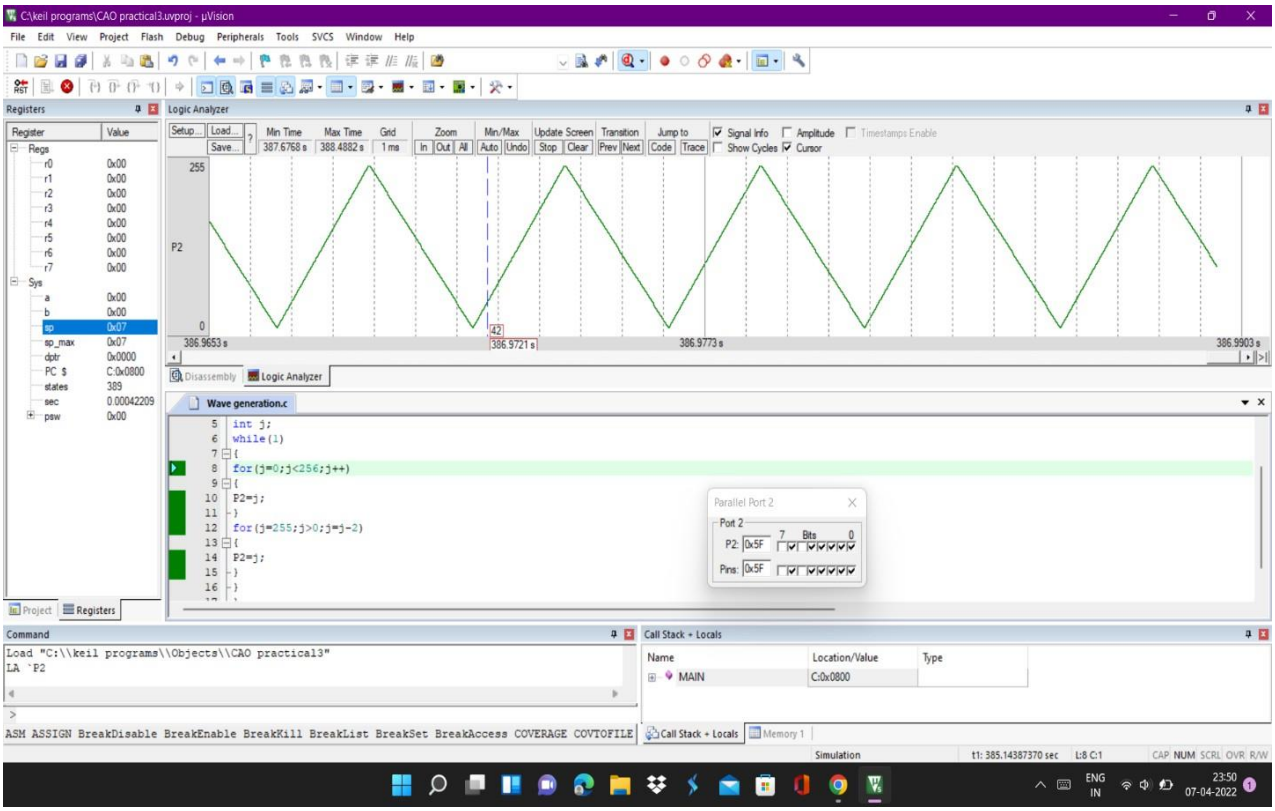
6.Triangular wave

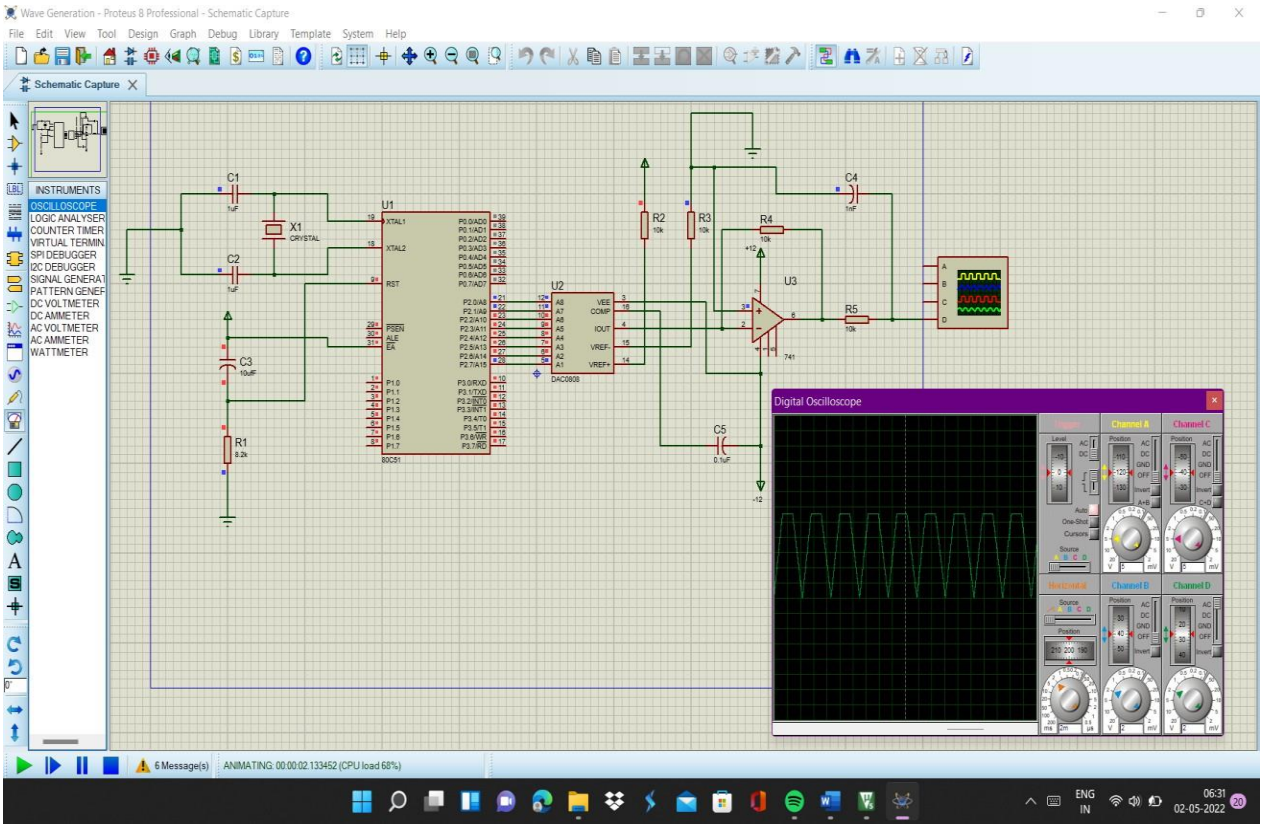
Code:

```
//Triangular wave
#include<reg51.h>
void main()
{
int j;
while(1)
{
for(j=0;j<256;j++)
{
P2=j;
}
for(j=255;j>0;j=j-2)
{
P2=j;
}
}
}
```



Output:


C:\keil programs\CAO practical3.uvproj - µVision
Logic Analyzer
Wave generation.c
Parallel Port 2
Command
Call Stack + Locals


Wave Generation - Proteus 8 Professional - Schematic Capture
Digital Oscilloscope

Conclusion: Hence, we performed the generation of Square, Rectangular, Sine wave, Cosine wave, Sawtooth wave and Triangular wave using 8051.

EXPERIMENT NO. 4

Aim: To interface LCD using 8051 microcontroller and displaying characters (AT89C51).

Theory: It is very important to keep a track of the working of almost all the automated and semi-automated devices, be it a washing machine, an autonomous robot or anything else. This is achieved by displaying their status on a small display module. LCD (Liquid Crystal Display) screen is such a display module and a 16x2 LCD module is very commonly used. These modules are replacing seven segments and other multi segment LEDs for these purposes.

The reasons being: LCDs are economical, easily programmable, have no limitation of displaying special & even custom characters (unlike in seven segments), animations and so on. LCD can be easily interfaced with a microcontroller to display a message or status of a device. This topic explains the basics of a 16x2 LCD and how it can be interfaced with AT89C51to display a character. A 16x2 LCD means it can display 16 characters per line and there are 2 such lines. In this LCD each character is displayed in 5x7 pixel matrix. This LCD has two registers.

1. **Command/Instruction Register**- stores the command instructions given to the LCD. A command is an instruction given to LCD to do a predefined task like initializing, clearing the screen, setting the cursor position, controlling display etc.

2. **Data Register**- stores the data to be displayed on the LCD. The data is the ASCII value of the character to be displayed on the LCD

The AT89C51 is a low-power, high-performance CMOS 8-bit microcomputer with 4K bytes of Flash programmable and erasable read only memory (PEROM). The device is manufactured using Atmel's high-density non-volatile memory technology and is compatible with the industry-standard MCS-51 instruction set and pin-out. The on-chip Flash allows the program memory to be reprogrammed in-system or by a conventional non-volatile memory programmer. By combining a versatile 8-bit CPU with Flash on a

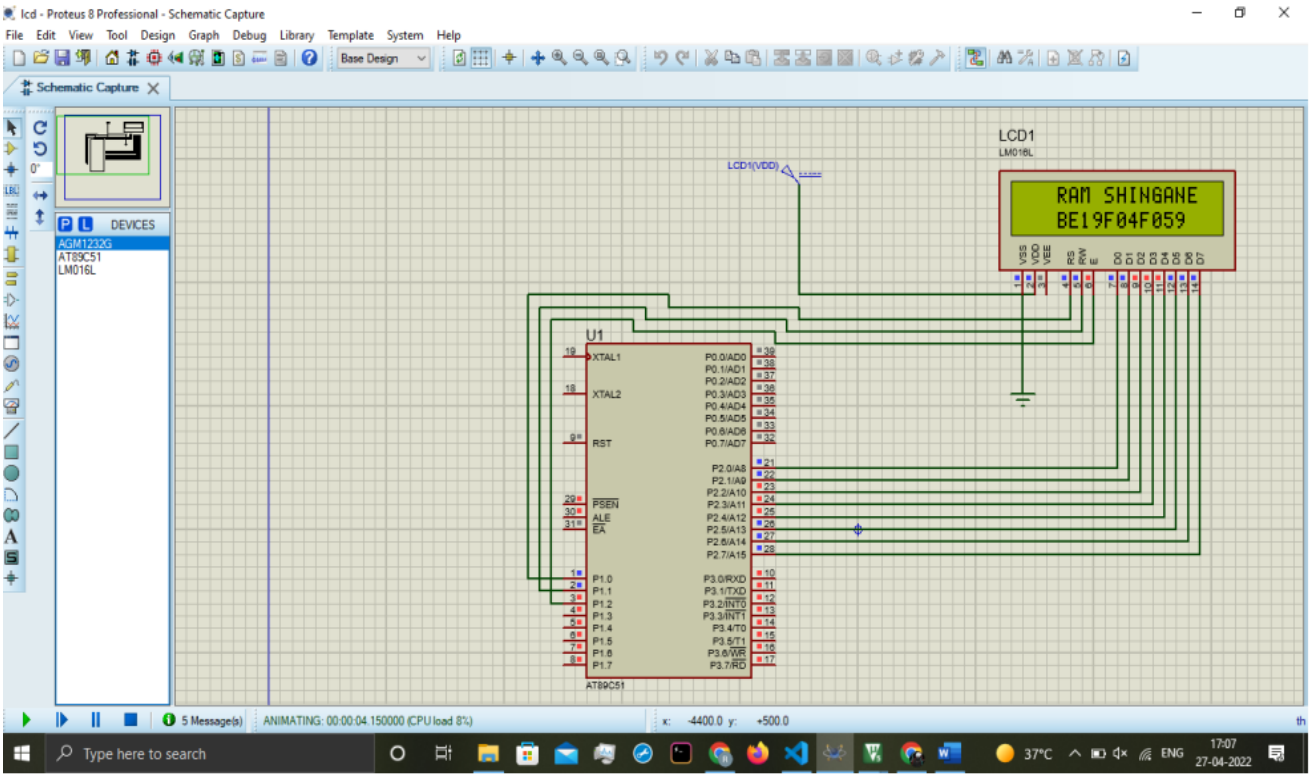
monolithic chip, the Atmel AT89C51 is a powerful microcomputer which provides a highly-flexible and cost-effective solution to many embedded control applications.

Code:

```
#include <reg51.h>
sbit rs=P1^0;
sbit rw=P1^1;
sbit e= P1^2;
void delay(unsigned int);
void cmd(unsigned char);
void dat(unsigned char);

void main(void)
{
        unsigned char ch[]="RAM
        SHINGANE"; unsigned char
        ch1[]="BE19F04F059";unsigned int
        i,j,k;

        cmd(0x38);
        cmd(0x01);
        cmd(0x0c);
        cmd(0x83);
        cmd(0x06);


        for(i=0;ch[i]!='\0';i++)
           dat(ch[i]);

                cmd(0xc3);
        for(j=0;ch1[j]!='\0';j++)
        {
                dat(ch1[j]);
         }
```

```
        while(1){
        for(k=0;k<16;k++)
        {
        cmd(0x1c);
        }
        }
}
void delay(unsigned int t)
{
        unsigned int i,j;
        e=1;
        for(i=0;i<t;i++)
        for(j=0;j<1275;j++);
        e=0;
}

void cmd(unsigned char ch)
{

        rs=0;
        rw=0;
        P2=ch;
        delay(20);
}
void dat(unsigned char ch)
{

        rs=1;
        rw=0;
        P2=ch;
        delay(20);
}
```

Output:



Conclusion:
Thus we have performed LCD interfacing and displayed the message name and roll number successfully.

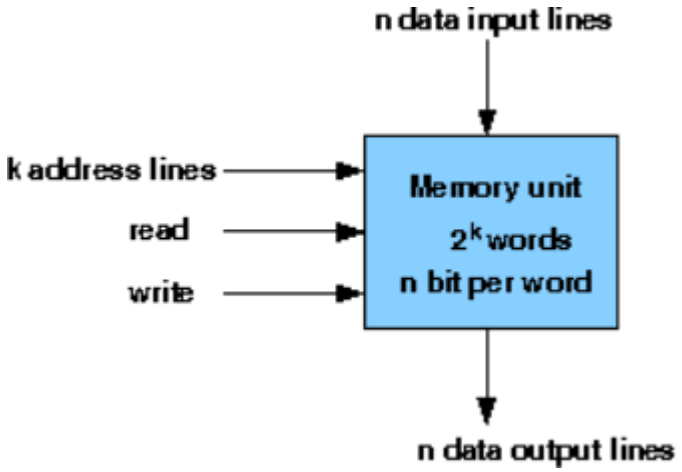
EXPERIMENT NO. 5

AIM: To design memory units and understand how it operates during read and write operation.

Theory:

Design of Memory:

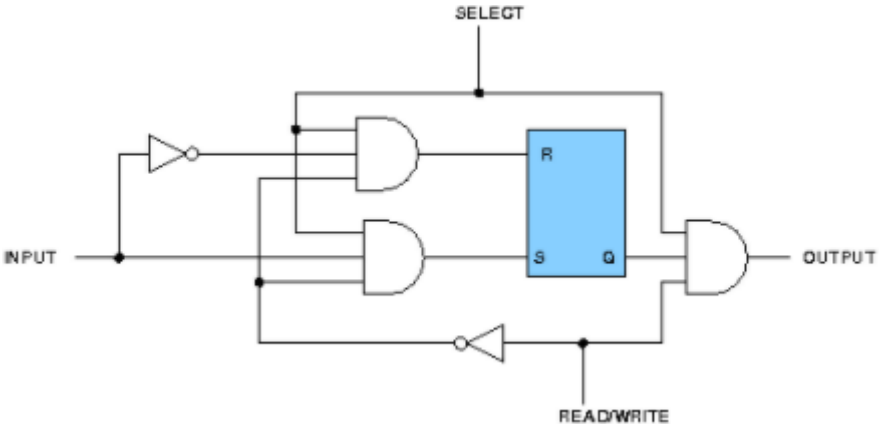
A memory unit is a collection of storage cells together with associated circuits needed to transform information in and out of the device. Memory cells which can be accessed for information transfer to or from any desired random location is called random access memory (RAM). The block diagram of a memory unit-



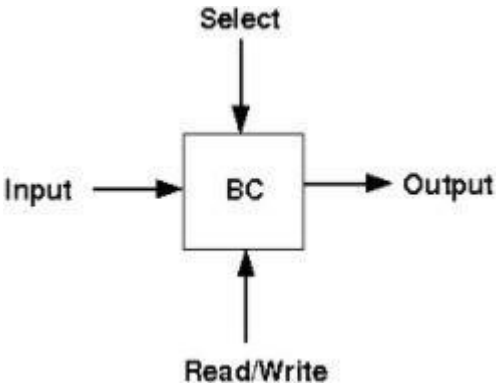
Internal Construction: The internal construction of a random-access memory of m words with n bits per word consists of m*n binary storage cells and associated decoding circuits for selecting individual words. The binary cell is the basic building block of a memory unit.

RAM Design:

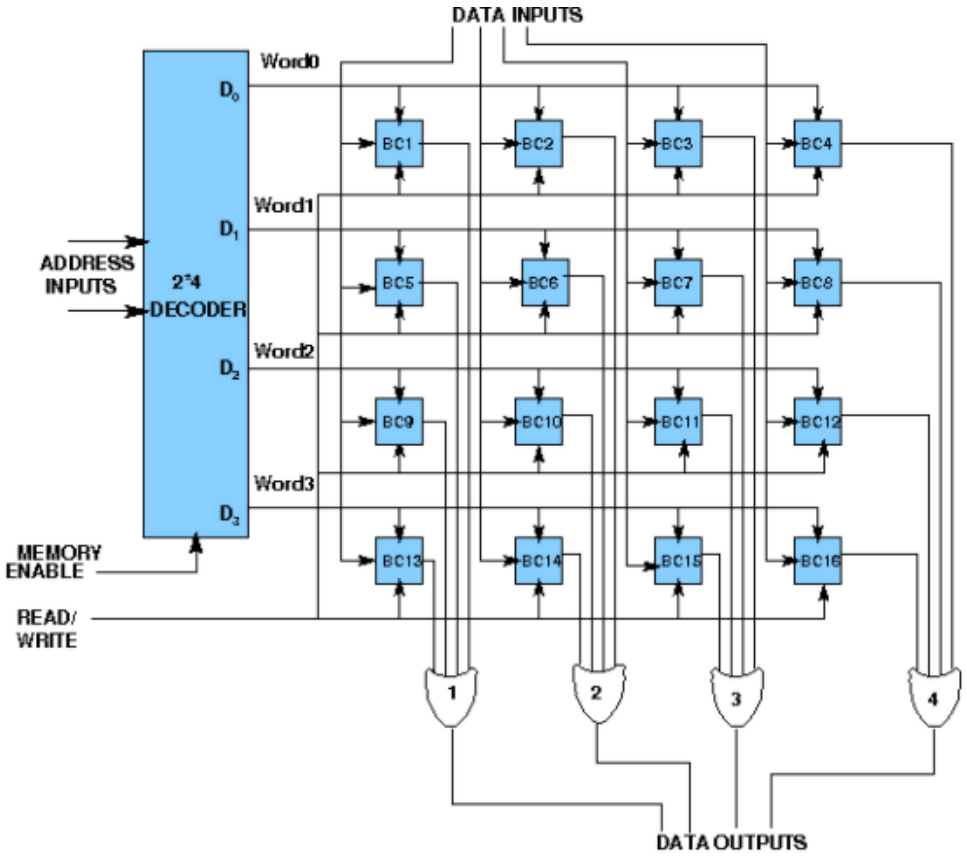
Design of a RAM cell: The binary cell has three inputs and one output. The select input enables the cell for reading or writing and the read/write input determines the cell operation when it is selected. A 1 in the read/write input provides the read operation by forming a path from the flip-flop to the output terminal. A 0 in the read/write input provides the write operation by forming a path from the input terminal to the flip-flop. the logic diagram is-



Design of a 4X4 RAM: The logical construction of a small RAM 4X3 is shown below. It consists of 4 words of 3 bits each and has a total of 12 binary cells. Each block labelled BC represents the binary cell with its 3 inputs and 1 output. The block diagram of a binary cell-



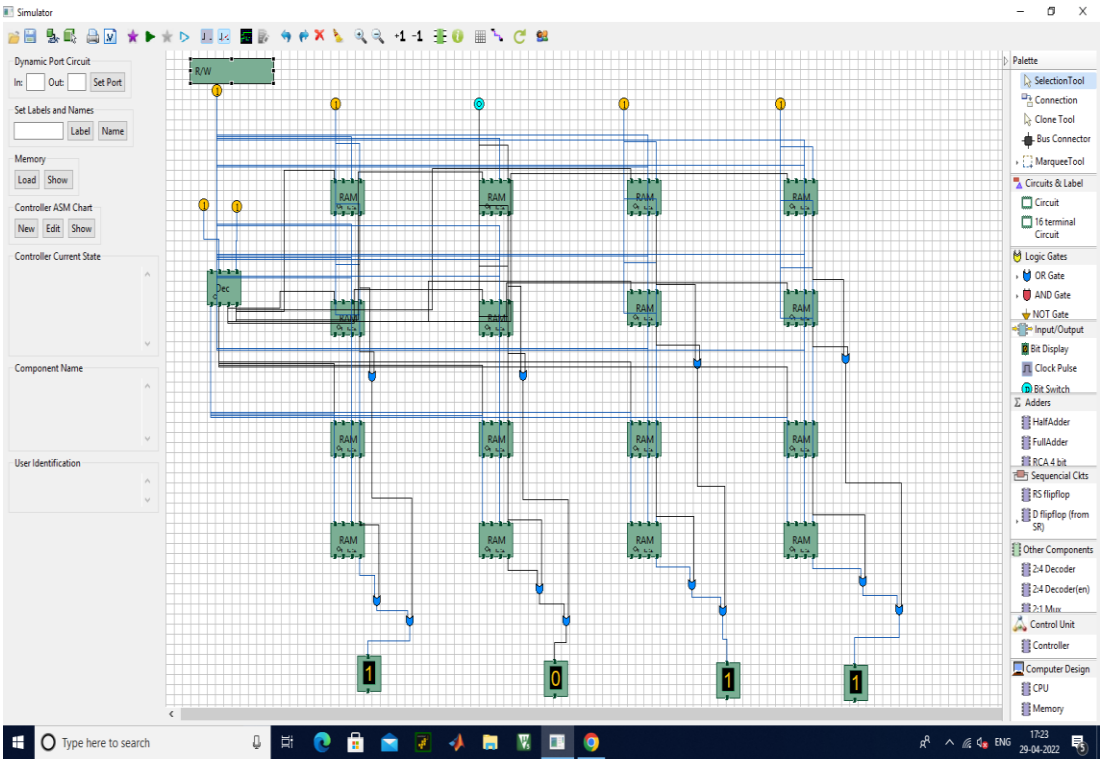
A memory with 4 words needs two address lines. The two address inputs go through a 2*4 decoder to select one of the four words. The decoder is enabled with the memory enable input. When the memory enable is 0, all outputs of the decoder are 0 and none of the memorywords are selected. With the memory enable at 1, one of the four words is selected, dictated by the value in the two address lines. Once a word has been selected, the read/write input determines the operation. the logic diagram is-



Design Issues:

A basic RAM cell has been provided here as a component which can be used to design larger memory units. An IC memory consisting of 4 words each having 3 bits has been also provided

Output:



Conclusion:

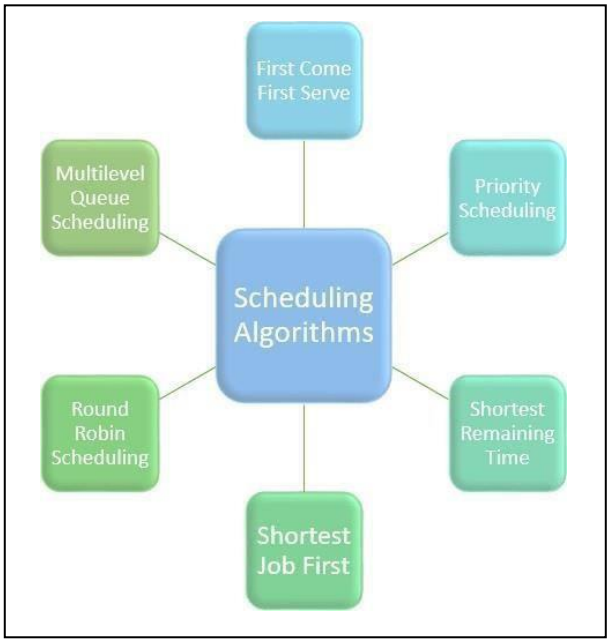
Memory Unit is Designed and its read -write operation is observed. Decoder decides the inputs to be selected using select inputs. Inputs can be changed while write operation. Decoder reads inputs from 2
nd line of RAM cells and when its select inputs are 01.

Experiment No. 6

AIM: Write a C program for CPU scheduling algorithm

SOFTWARE USED: Online C/C++ Editor

THEORY: A process scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to mention are as follows:



These algorithms are either non preemptive or preemptive. Non-preemptive algorithms are designed so that once it cannot be preempted until it completes its allotted time, whereas a scheduler may preempt a low priority running process time when a high priority process enters into a ready state.

Input Table :

Round Robin Scheduling		
Mode- Pre-emptive, Quantum time- 1 unit		
Process Id	Arrival Time	Burst Time
P1	0	5
P2	1	6
P3	2	3
P4	3	1
P5	4	5
P6	6	4

Gantt Chart:

Ready Queue:

p1	p2	p1	p3	p4	p1	p2	p1	p5	p6	p1	p2

p3	p2	p3	p2	p6	p5	p2	p5	p6	p6	p5	p5

Running Queue:

p1	p2	p1	p3	p4	p1	p2	p1	p5	p6	p1	p2

0 1 2 3 4 5 6 7 8 9 10 11 12

p3	p2	p3	p2	p6	p5	p2	p5	p6	p6	p5	p5

12 13 14 15 16 17 18 19 20 21 22 23 24

Output Table:

Shortest Job First Scheduling
Mode- Non-pre-emptive

Process Id	Arrival Time	Burst Time	Completion Time	Turn Around Time	Waiting Time	Response Time
P1	0	5	11	11	6	0
P2	1	6	19	18	12	0
P3	2	3	15	13	10	1
P4	3	1	5	2	1	1
P5	4	5	24	20	15	4
p6	6	4	22	16	12	3

Average TAT :- 13.33

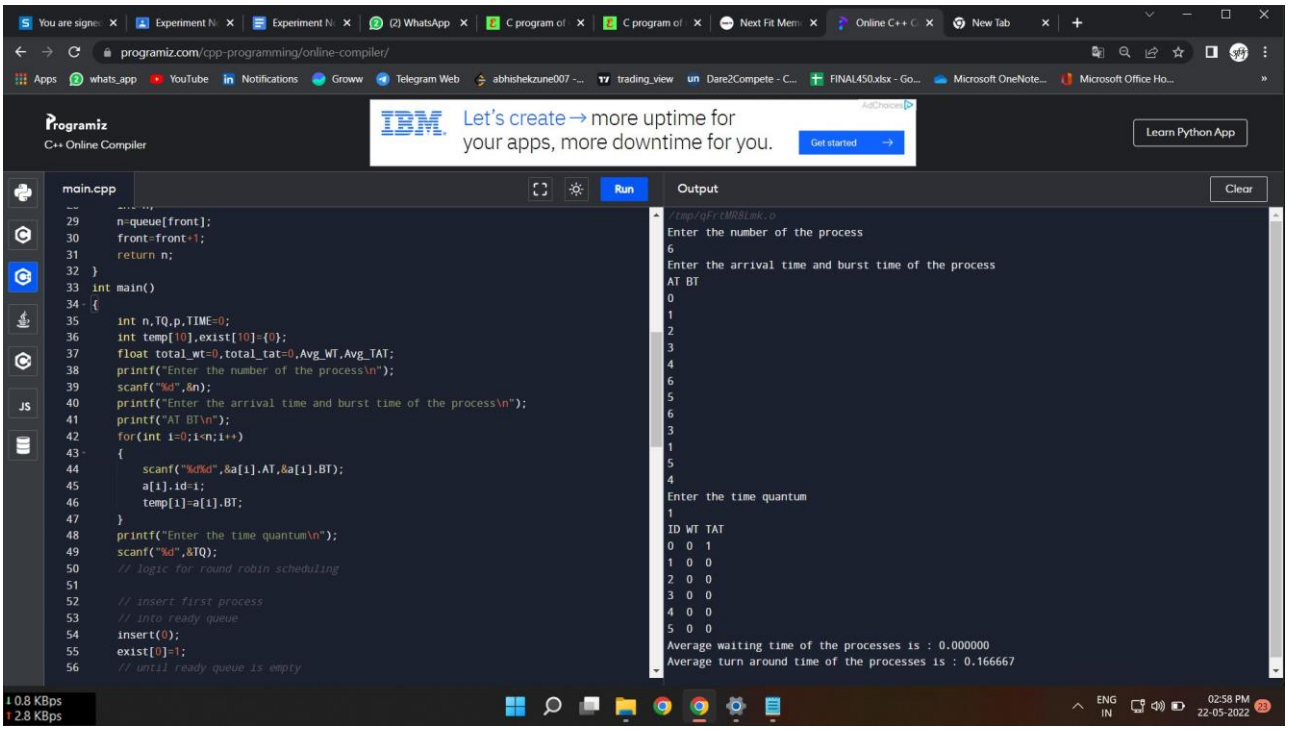
Average WT :- 9.33

Code :

```
#include<stdio.h> struct
process
{ int id,AT,BT,WT,TAT;
};


struct process a[10];// declaration of the ready queue
int queue[100]; int front=-1; int rear=-1;


// function for insert the element // into
queue void insert(int n)
{ if(front==-1)
    front=0;
    rear=rear+1;
    queue[rear]=n;
}
// function for delete the //
element from queue int
_delete()
{ int n; n=queue[front];
front=front+1; return n;
} int main()
{ int n,TQ,p,TIME=0; int temp[10],exist[10]={0}; float
    total_wt=0,total_tat=0,Avg_WT,Avg_TAT; printf("Enter the number of the
    process\n"); scanf("%d",&n); printf("Enter the arrival time and burst time of the
    process\n"); printf("AT BT\n"); for(int i=0;i<n;i++)
    { scanf("%d%d",&a[i].AT,&a[i].BT);
        a[i].id=i; temp[i]=a[i].BT;
    } printf("Enter the time quantum\n");
    scanf("%d",&TQ);
    // logic for round robin scheduling
```

```
// insert first process
// into ready queue insert(0);
exist[0]=1; // until ready queue is
empty while(front<=rear)
{ p=_delete();
    if(a[p].BT>=TQ)
    { a[p].BT=a[p].BT-TQ;
        TIME=TIME+TQ;

    }
    else
    {

        TIME=TIME+a[p].BT; a[p].BT=0;
    }




    //if process is not exist

    // in the ready queue even a single

    // time then insert it if it arrive // at time
    'TIME' for(int i=0;i<n;i++)
    { if(exist[i]==0 && a[i].AT<=TIME)
        { insert(i);
            exist[i]=1;
        }
    }

    // if process is completed if(a[p].BT==0)
    {

        a[p].TAT=TIME-a[p].AT;
        a[p].WT=a[p].TAT-temp[p];
        total_tat=total_tat+a[p].TAT;
        total_wt=total_wt+a[p].WT;
    }
    else
    { insert(p);
    }

}


Avg_TAT=total_tat/n;

Avg_WT=total_wt/n;


// printing of the answer
printf("ID WT TAT\n"); for(int
i=0;i<n;i++)
{ printf("%d %d %d\n",a[i].id,a[i].WT,a[i].TAT);
} printf("Average waiting time of the processes is : %f\n",Avg_WT); printf("Average turn around
time of the processes is : %f\n",Avg_TAT); return 0;
```

Output:



Conclusion:

Hence, we performed CPU Scheduling algorithm practical by Mode- Pre-emptive Round Robin Scheduling using c programming**.**

Experiment No. 7

Aim: Write a C program for the memory management algorithm.

Theory: Memory Management is the process of controlling and coordinating computer memory, assigning portions known as blocks to various running programs to optimize the overall performance of the system.

It is the most important function of an operating system that manages primary memory. It helps processes to move back and forward between the main memory andexecution disk. It helps OS to keep track of every memory location, irrespective of whether it is allocated to some process or it remains free.

In **Partition Allocation**, when there is more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed. A partition allocation method is considered better if it avoids internal fragmentation.

When it is time to load a process into the main memory and if there is more than one free block of memory of sufficient size then the OS decides which free block to allocate.

There are different Placement Algorithm:

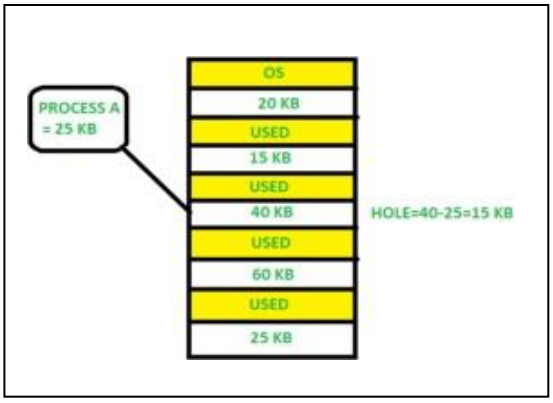
A. First Fit

B. Best Fit

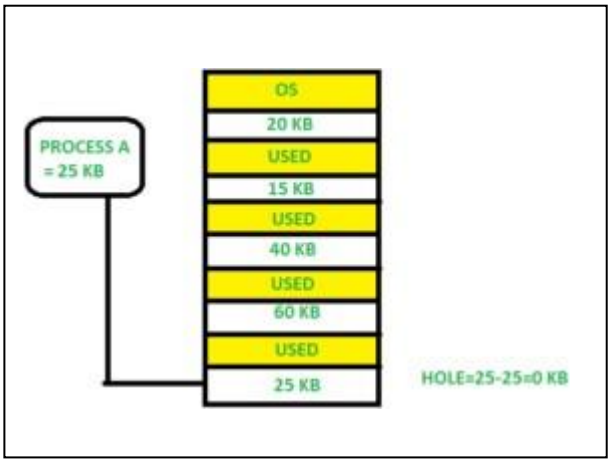
C. Worst Fit

D. Next Fit

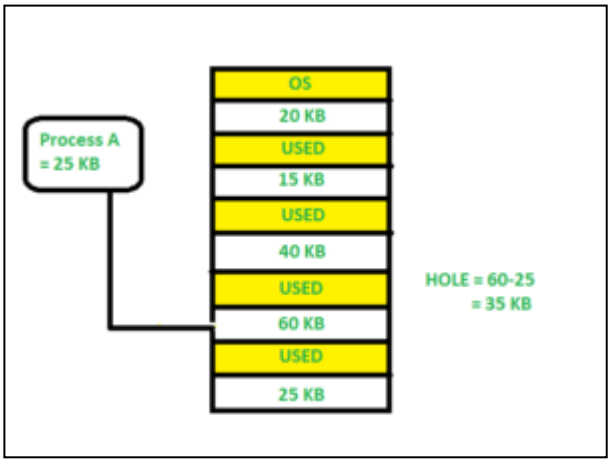
1. First Fit: In the first fit, the partition is allocated which is the first sufficient block from the top of Main Memory. It scans memory from the beginning and chooses the first available block that is large enough. Thus, it allocates the first hole that is large enough.



2. Best Fit Allocate the process to the partition which is the first smallest sufficient partition among the free available partition. It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.



3. Worst Fit Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory. It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it toprocess.



4. Next Fit: Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

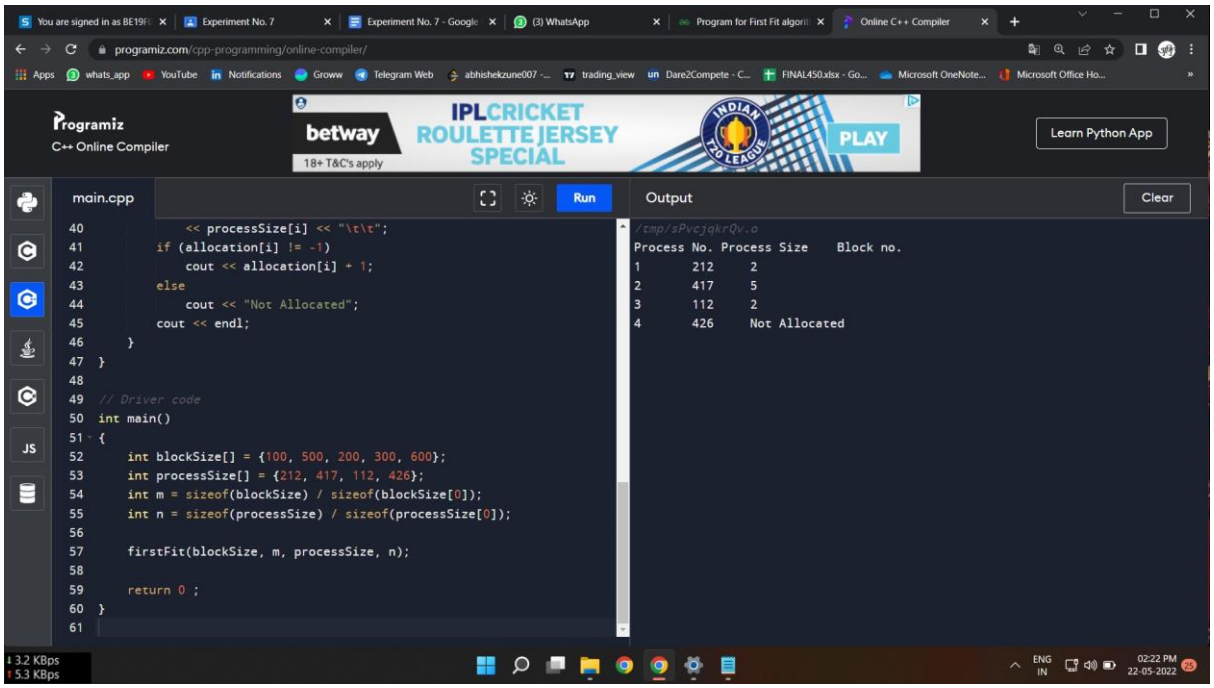
CODE :

A. First Fit

```
// C++ implementation of First - Fit algorithm
#include<bits/stdc++.h> using namespace
std;

// Function to allocate memory to // blocks as
per First fit algorithm void firstFit(int
blockSize[], int m,
                    int processSize[], int n)
{
        // Stores block id of the
        // block allocated to a process
        int allocation[n];

        // Initially no block is assigned to any process memset(allocation, -1,
        sizeof(allocation));

        // pick each process and find suitable blocks // according
        to its size ad assign to it for (int i = 0; i < n; i++)
        { for (int j = 0; j < m; j++)
                { if (blockSize[j] >= processSize[i])
                        {
                                // allocate block j to p[i] process allocation[i] = j;
```

```
                    // Reduce available memory in this block. blockSize[j] -=
                    processSize[i];

                    break;
                }
            } }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n";
    for (int i = 0; i < n; i++)
    { cout << " " << i+1 << "\t\t"
                << processSize[i] << "\t\t"; if
        (allocation[i] != -1) cout << allocation[i] + 1;
        else cout << "Not Allocated";
        cout << endl;
    } }

// Driver code int main()
{ int blockSize[] = {100, 500, 200, 300, 600}; int processSize[] = {212,
    417, 112, 426}; int m = sizeof(blockSize) / sizeof(blockSize[0]);
    int n = sizeof(processSize) / sizeof(processSize[0]);

    firstFit(blockSize, m, processSize, n);

    return 0 ;
}
```

OUTPUT :


programiz.com/cpp-programming/online-compiler/
Programiz
C++ Online Compiler
betway
IPLCRICKET
ROULETTE JERSEY
SPECIAL
18+ T&C's apply
PLAY
Learn Python App
main.cpp
Run
Output
Clear
<< processSize[i] << "\t\t";
if (allocation[i] != -1)
cout << allocation[i] + 1;
else
cout << "Not Allocated";
cout << endl;
}
}
// Driver code
int main()
{
int blockSize[] = {100, 500, 200, 300, 600};
int processSize[] = {212, 417, 112, 426};
int m = sizeof(blockSize) / sizeof(blockSize[0]);
int n = sizeof(processSize) / sizeof(processSize[0]);
firstFit(blockSize, m, processSize, n);
return 0 ;
}
/tmp/sPvcjqkrQv.o
Process No. Process Size Block no.
1 212 2
2 417 5
3 112 2
4 426 Not Allocated
3.2 KBps
5.3 KBps
ENG
IN
02:22 PM
22-05-2022

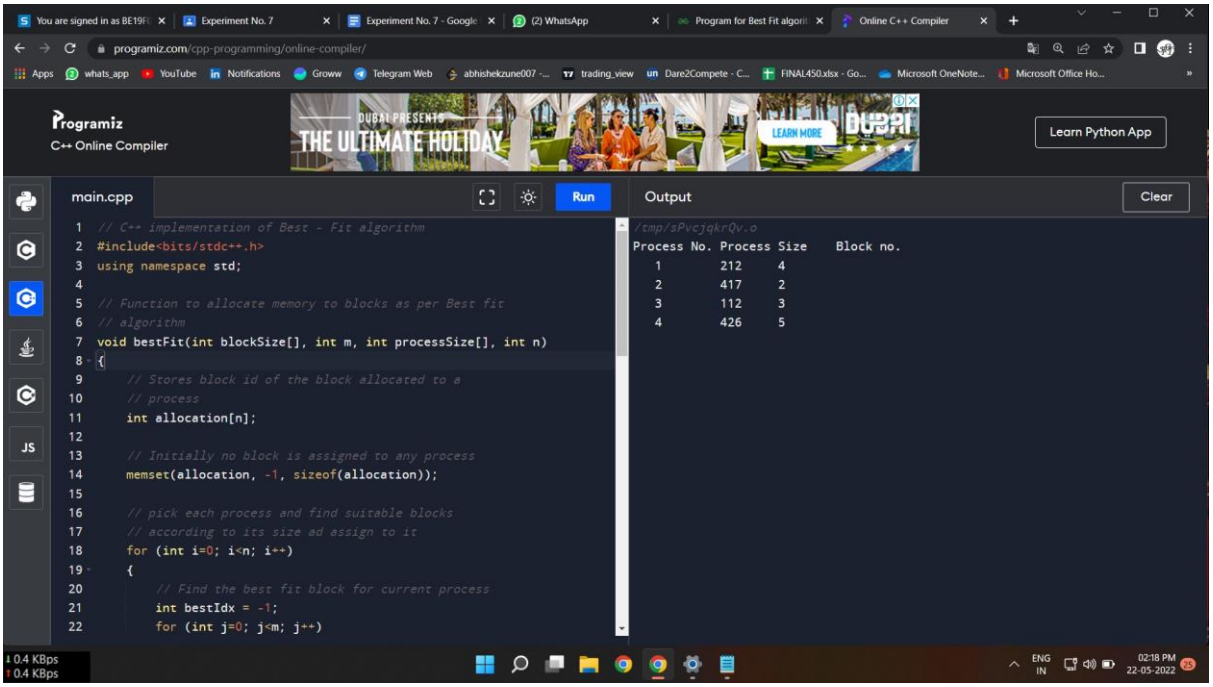
B. Best Fit

```
                // C++ implementation of Best - Fit algorithm
#include<bits/stdc++.h> using namespace
std;

// Function to allocate memory to blocks as per Best fit
// algorithm
void bestFit(int blockSize[], int m, int processSize[], int n)
{
    // Stores block id of the block allocated to a
    // process int allocation[n];

    // Initially no block is assigned to any process memset(allocation, -1,
    sizeof(allocation));

    // pick each process and find suitable blocks // according
    to its size ad assign to it for (int i=0; i<n; i++)
    {
        // Find the best fit block for current process int bestIdx =
        -1; for (int j=0; j<m; j++)
        { if (blockSize[j] >= processSize[i])
            { if (bestIdx == -1) bestIdx = j;

                else if (blockSize[bestIdx] > blockSize[j])
                    bestIdx = j;
            } }

        // If we could find a block for current process if (bestIdx != -
        1)
```

```
        {
            // allocate block j to p[i] process allocation[i] = bestIdx;

            // Reduce available memory in this block.
            blockSize[bestIdx] -= processSize[i];
        } }

    cout << "\nProcess No.\tProcess Size\tBlock no.\n"; for (int i = 0; i <
    n; i++)
    { cout << "   " << i+1 << "\t\t" << processSize[i] << "\t\t"; if (allocation[i]
        != -1)
            cout << allocation[i] + 1;
        else
            cout << "Not Allocated";
        cout << endl;
    }
}

// Driver code int main()
{ int blockSize[] = {100, 500, 200, 300, 600};

    int processSize[] =  {212, 417, 112, 426};

    int m = sizeof(blockSize)/sizeof(blockSize[0]); int n =
    sizeof(processSize)/sizeof(processSize[0]);

    bestFit(blockSize, m, processSize, n);

    return 0 ;
```

}

OUTPUT :

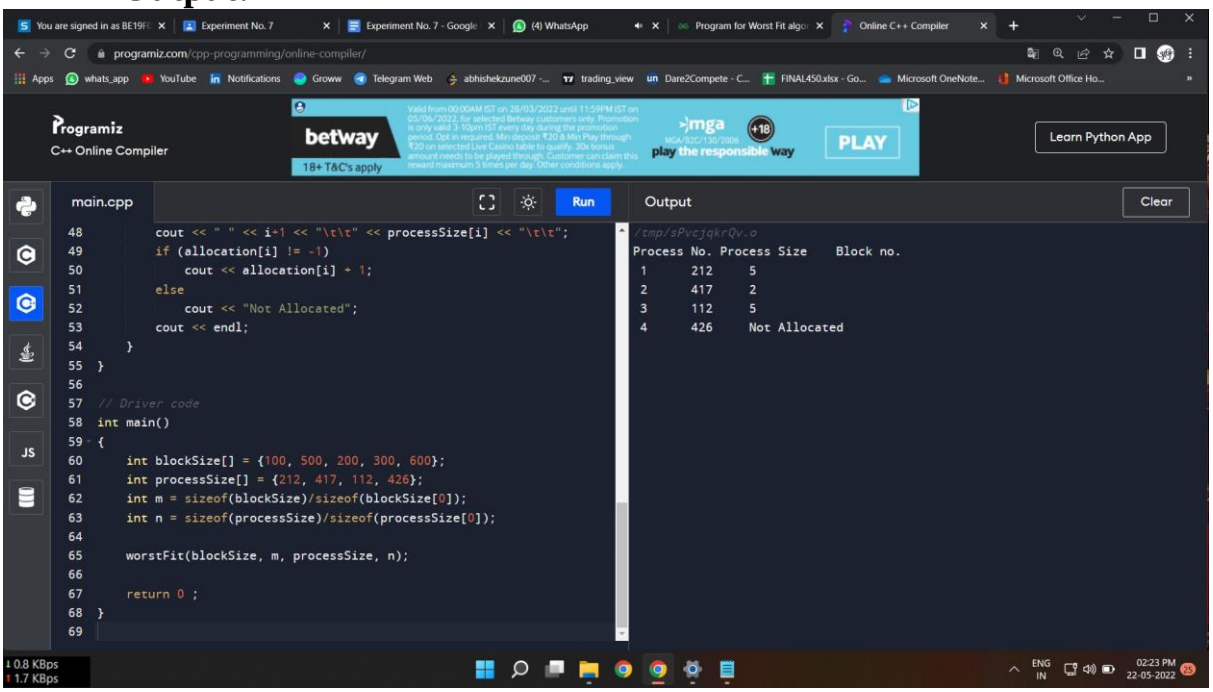

Programiz
C++ Online Compiler
Learn Python App
main.cpp
Run
Output
Clear
// C++ implementation of Best - Fit algorithm
#include<bits/stdc++.h>
using namespace std;
// Function to allocate memory to blocks as per Best fit
// algorithm
void bestFit(int blockSize[], int m, int processSize[], int n)
{
// Stores block id of the block allocated to a
// process
int allocation[n];
// Initially no block is assigned to any process
memset(allocation, -1, sizeof(allocation));
// pick each process and find suitable blocks
// according to its size ad assign to it
for (int i=0; i<n; i++)
{
// Find the best fit block for current process
int bestIdx = -1;
for (int j=0; j<m; j++)
/tmp/sPvcjqkrQv.o
Process No. Process Size Block no.
1 212 4
2 417 2
3 112 3
4 426 5

C. Worst Fit

```
// C++ implementation of worst - Fit algorithm
#include<bits/stdc++.h> using namespace
std;

// Function to allocate memory to blocks as per worst fit
// algorithm
void worstFit(int blockSize[], int m, int processSize[],
                                                    int n)
{
    // Stores block id of the block allocated to a
    // process int allocation[n];

    // Initially no block is assigned to any process memset(allocation, -1,
    sizeof(allocation));

    // pick each process and find suitable blocks // according
    to its size ad assign to it for (int i=0; i<n; i++)
    {
        // Find the best fit block for current process int wstIdx = -
        1; for (int j=0; j<m; j++)
        { if (blockSize[j] >= processSize[i])
            { if (wstIdx == -1) wstIdx = j;
                else if (blockSize[wstIdx] < blockSize[j]) wstIdx = j;
            } }

        // If we could find a block for current process if (wstIdx != -
        1)
        {
```

```
                // allocate block j to p[i] process allocation[i] = wstIdx;

                // Reduce available memory in this block.
                blockSize[wstIdx] -= processSize[i];
            } }

        cout << "\nProcess No.\tProcess Size\tBlock no.\n"; for (int i = 0; i <
        n; i++)
        { cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t"; if (allocation[i] !=
            -1) cout << allocation[i] + 1;
            else cout << "Not Allocated";
            cout << endl;
        } }

// Driver code int main()
{ int blockSize[] = {100, 500, 200, 300, 600}; int processSize[] = {212,
        417, 112, 426}; int m = sizeof(blockSize)/sizeof(blockSize[0]);
        int n = sizeof(processSize)/sizeof(processSize[0]);

        worstFit(blockSize, m, processSize, n);

        return 0 ;
}
```

Output:


programiz.com/cpp-programming/online-compiler/
Programiz
C++ Online Compiler
betway
18+ T&C's apply
play the responsible way
PLAY
Learn Python App
main.cpp
Run
Output
Clear
cout << " " << i+1 << "\t\t" << processSize[i] << "\t\t";
if (allocation[i] != -1)
cout << allocation[i] + 1;
else
cout << "Not Allocated";
cout << endl;
}
}
// Driver code
int main()
{
int blockSize[] = {100, 500, 200, 300, 600};
int processSize[] = {212, 417, 112, 426};
int m = sizeof(blockSize)/sizeof(blockSize[0]);
int n = sizeof(processSize)/sizeof(processSize[0]);
worstFit(blockSize, m, processSize, n);
return 0 ;
}
/tmp/sPvcjqkrQv.o
Process No. Process Size Block no.
1 212 5
2 417 2
3 112 5
4 426 Not Allocated
0.8 KBps
1.7 KBps
ENG
IN
02:23 PM
22-05-2022

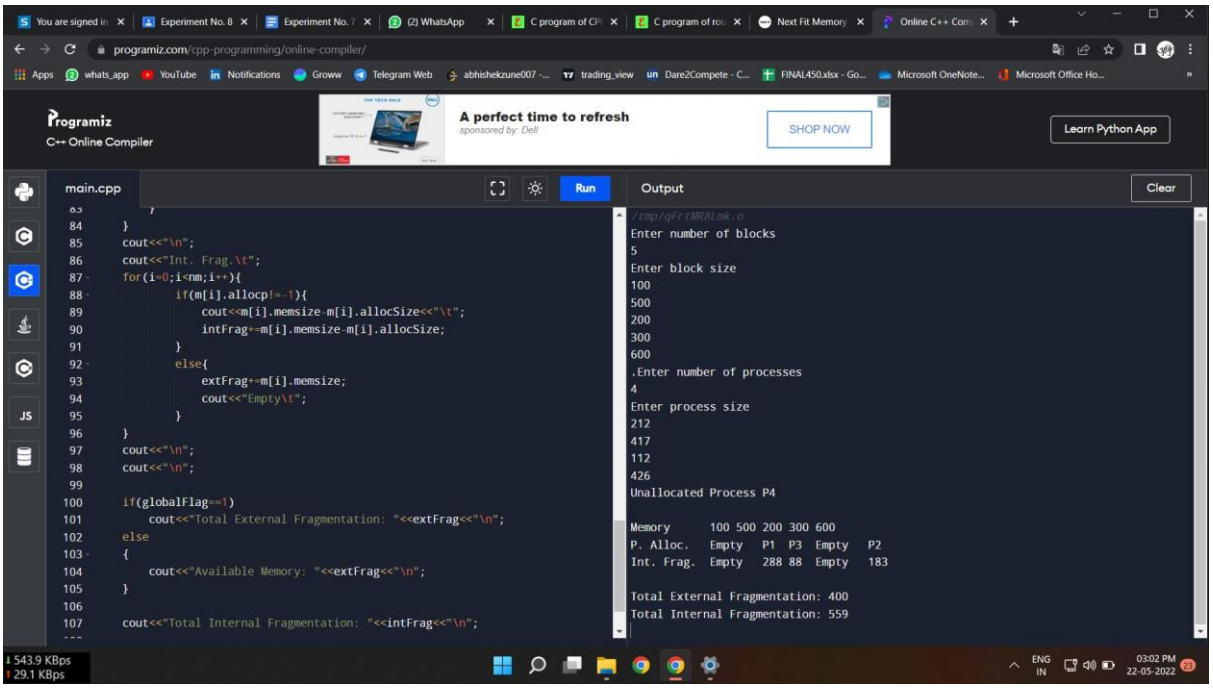
D. Next Fit

```
#include<iostream>
#include<algorithm> using namespace
std;

struct node{ int memsize; int
allocp=-1; int pos; int
allocSize; }m[200];


bool posSort(node a,node b){
   return a.pos < b.pos;
}

bool memSort(node a,node b){
   return a.memsize < b.memsize;
}

int main(){
   int nm,np,choice, i, j, p[200]; cout<<"Enter
   number of blocks\n"; cin>>nm; cout<<"Enter
   block size\n"; for(i=0;i<nm;i++){
   cin>>m[i].memsize; m[i].pos=i; }

   cout<<"Enter number of processes\n"; cin>>np;

   cout<<"Enter process size\n"; for(i=0;i<np;i++){
      cin>>p[i]; } cout<<"\n\n"; int
   globalFlag=0; int pos = -1;
```

```
for(i=0;i<np;i++){
   int flag=0; for(j=pos+1;j<nm;j++){
      if(j==nm){
         j=0;

      } if(j==pos)
         break;

                        if(p[i]<=m[j].memsize && m[j].allocp==-1){
         m[j].allocp=i; m[j].allocSize=p[i]; flag=1; pos =
         j; if(j==nm-1){ j=0; pos = -1; } break;
      } } if(flag==0){ cout<<"Unallocated Process P"<<i+1<<"\n";
   globalFlag=1;
      }
   } sort(m,m+nm,posSort);
cout<<"\n"; int intFrag=0,extFrag=0;
cout<<"Memory\t\t";
for(i=0;i<nm;i++){
   cout<<m[i].memsize<<"\t";
} cout<<"\n"; cout<<"P.
Alloc.\t"; for(i=0;i<nm;i++){
   if(m[i].allocp!=-1){
      cout<<"P"<<m[i].allocp+1<<"\t";
   }
   else{
      cout<<"Empty\t";
   }
} cout<<"\n"; cout<<"Int. Frag.\t";
for(i=0;i<nm;i++){ if(m[i].allocp!=-
```

```
   1){
            cout<<m[i].memsize-m[i].allocSize<<"\t"; intFrag+=m[i].memsize-
            m[i].allocSize;
         } else{
            extFrag+=m[i].memsize; cout<<"Empty\t";
         }
   } cout<<"\n"; cout<<"\n";

   if(globalFlag==1)
                        cout<<"Total External Fragmentation: "<<extFrag<<"\n";
   else
   { cout<<"Available Memory: "<<extFrag<<"\n";
   }

   cout<<"Total Internal Fragmentation: "<<intFrag<<"\n"; return 0;
}
```

OUTPUT :


programiz.com/cpp-programming/online-compiler/
Programiz
C++ Online Compiler
A perfect time to refresh
sponsored by: Dell
SHOP NOW
Learn Python App
main.cpp
Run
Output
Clear
cout<<"\n";
cout<<"Int. Frag.\t";
for(i=0;i<nm;i++){
if(m[i].allocp!=-1){
cout<<m[i].memsize-m[i].allocSize<<"\t";
intFrag+=m[i].memsize-m[i].allocSize;
}
else{
extFrag+=m[i].memsize;
cout<<"Empty\t";
}
}
cout<<"\n";
cout<<"\n";
if(globalFlag==1)
cout<<"Total External Fragmentation: "<<extFrag<<"\n";
else
{
cout<<"Available Memory: "<<extFrag<<"\n";
}
cout<<"Total Internal Fragmentation: "<<intFrag<<"\n";
/tmp/qFrtMR8Lmk.o
Enter number of blocks
5
Enter block size
100
500
200
300
600
.Enter number of processes
4
Enter process size
212
417
112
426
Unallocated Process P4
Memory 100 500 200 300 600
P. Alloc. Empty P1 P3 Empty P2
Int. Frag. Empty 288 88 Empty 183
Total External Fragmentation: 400
Total Internal Fragmentation: 559
543.9 KBps
29.1 KBps
ENG
IN
03:02 PM
22-05-2022

Conclusion:

Here, we performed memory management algorithm like First fit, Next fit, Worst fit, Best fit practical using C programming language.

Experiment No. 8

Aim: Installing Ubuntu using virtual box, and study virtualization.

Theory:

Introduction to VirtualBox-

VirtualBox allows us to run an entire operating system inside another operating system.

Advantages of virtual installation

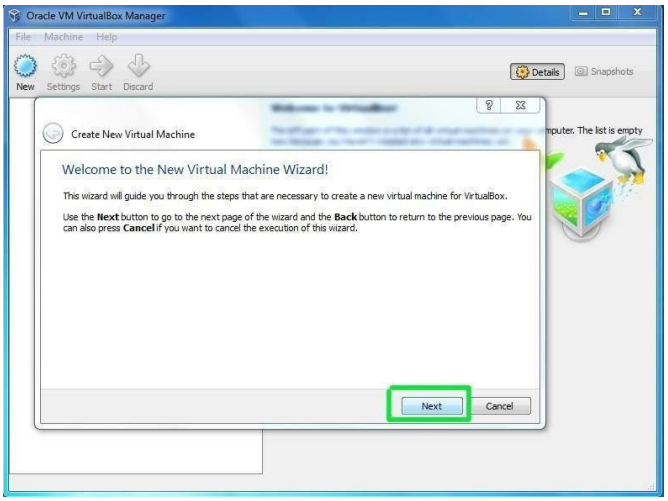
- The size of the installation does not have to be predetermined. It can be a dynamically resized virtual hard drive.
- No reboot needed in order to switch between Ubuntu and Windows.
- The virtual machine will use Windows internet connection.
- The virtual machine will set up its own video configuration.
- For troubleshooting purposes, take screenshots of any part of Ubuntu (including the boot menu or the login screen).
- It's low commitment. We can uninstall VirtualBox and terminate Ubuntu.

Disadvantages of virtual installation

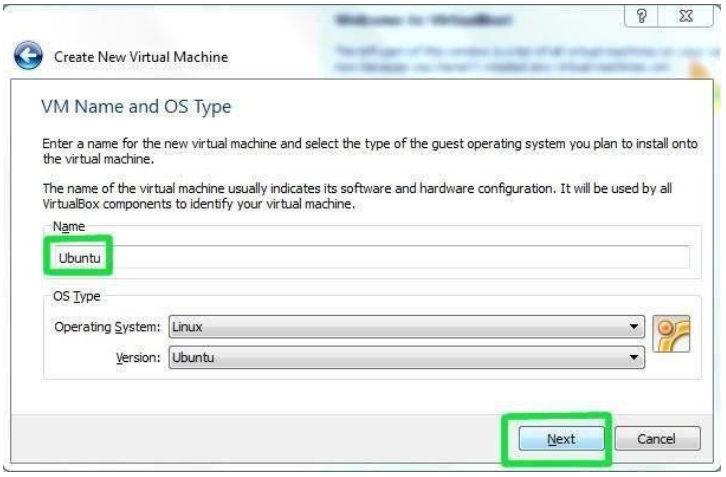
- In order to get any kind of decent performance, at least 512 MB of RAM is needed, because we are running an entire operating system (Ubuntu) inside another entire operating system (Windows).
- Even though the low commitment factor can seem like an advantage at first, if we later decide we want to switch to Ubuntu and ditch Windows completely, we cannot simply delete Windows partition.
- Every time we want to use Ubuntu, we have to wait for two boot times (the time it takes to boot Windows, and then the time it takes to boot Ubuntu within Windows).

Installation steps-

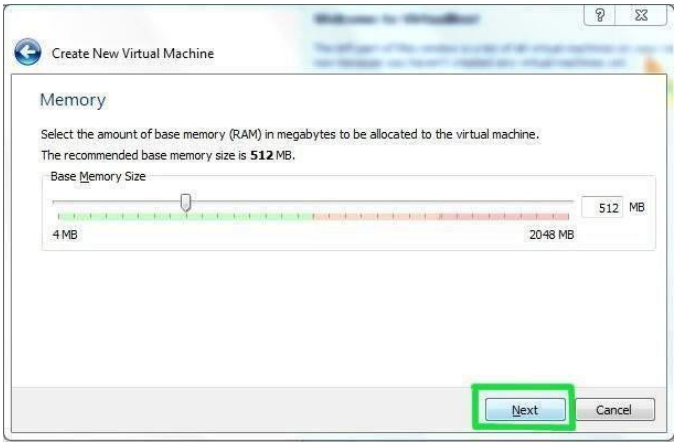
1. Download VirtualBox using the link - https://www.virtualbox.org/wiki/Downloads
2. Follow these instructions to get an Ubuntu disk image (.iso file).



3. After you launch VirtualBox from the Windows Start menu, click on New to create a new virtual machine. When the New Virtual Machine Wizard appears, click Next.

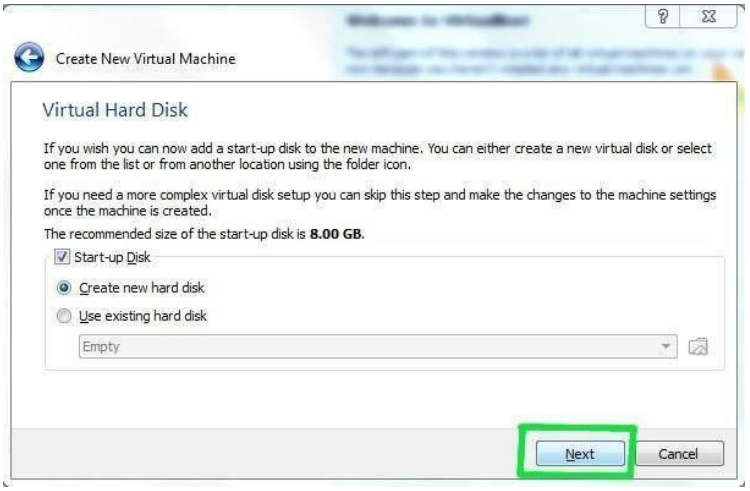


4. Assign title to the machine also specify that the operating system is Linux.

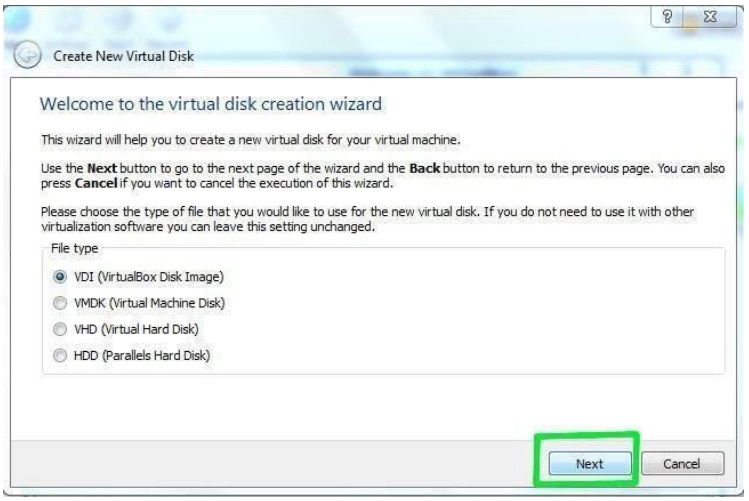


5. VirtualBox will try to guess memory (or RAM) to allocate for the virtual machine. If we have 1 GB or less of RAM, proceed as it is with the recommendation. If we have over 1 GB,about a quarter RAM or less should be fine. For example, if we have 2 GB of RAM,

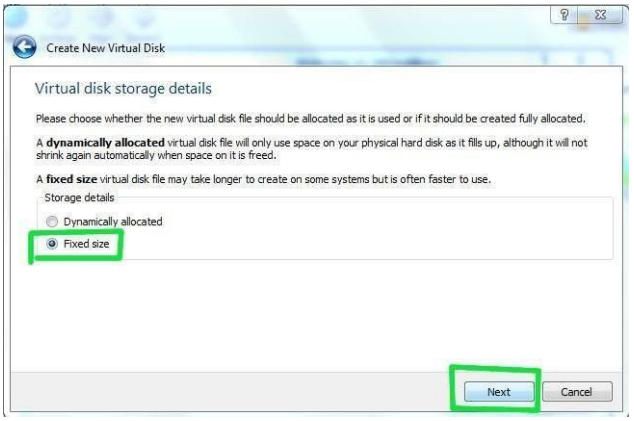
512 MB is fine to allocate. If we have 4 GB of RAM, 1 GB is fine to allocate. If we have no idea what RAM is or how much of it we have, just go with the default. Click Next.



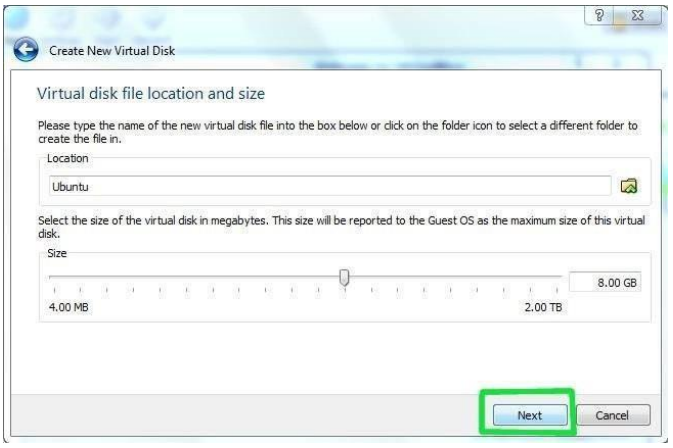
6. *Create new hard disk* and then click Next.



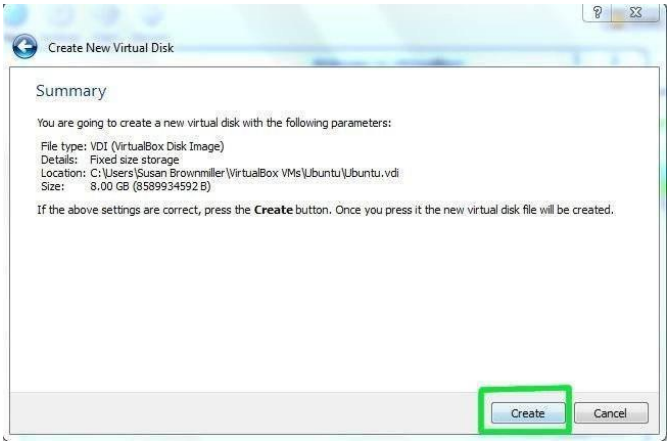
7. Click Next again.

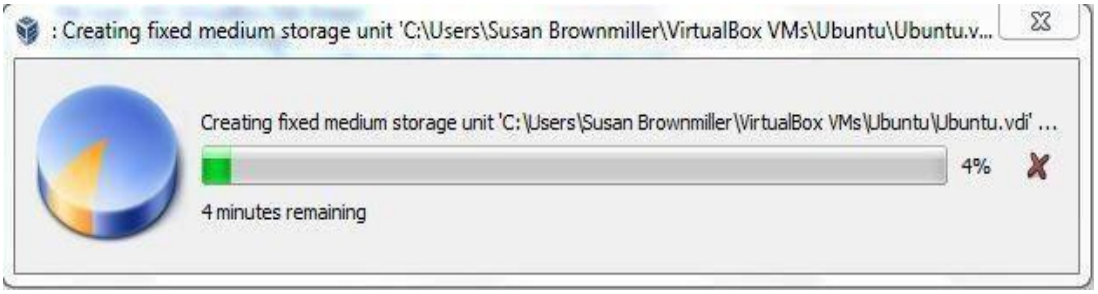


8. A dynamically expanding virtual hard drive is best, because it take up only what actually is used. But when installing new software in a virtualized Ubuntu, in which the virtual hard drive fills up instead of expanding. So it is recommended to Fixed-size storage.

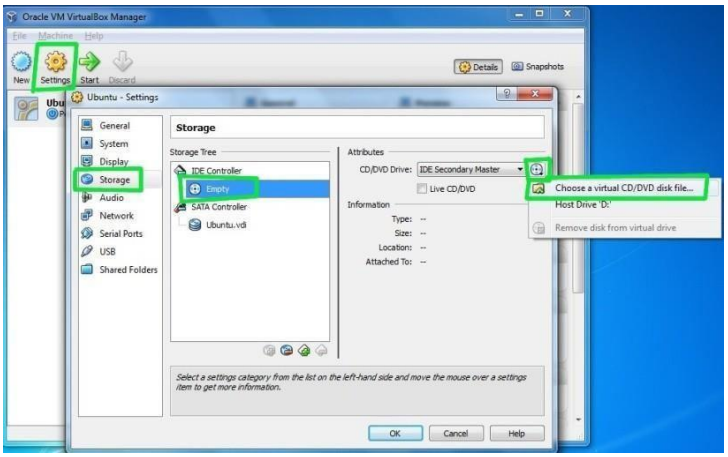


9. Ubuntu's default installation is less than 3 GB

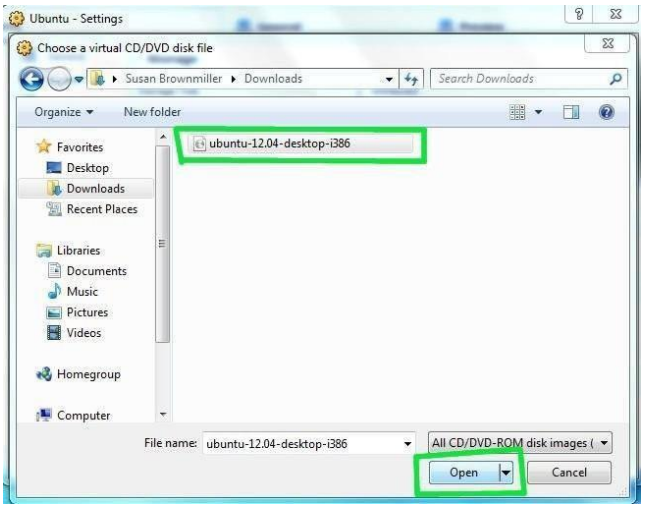




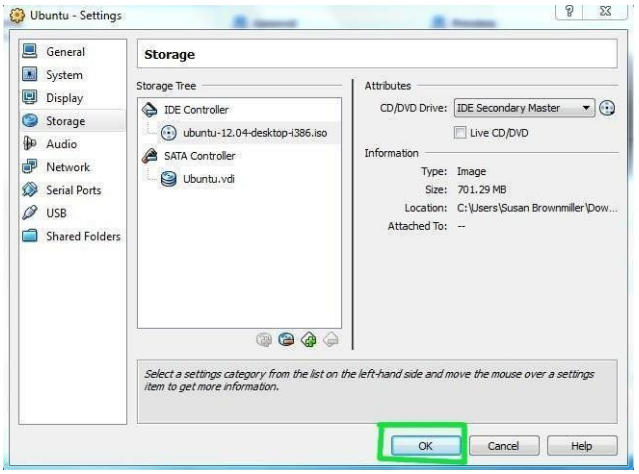
10. Click Create and wait for the virtual hard drive to be created. This is actually a very large file that lives inside of Windows installation.



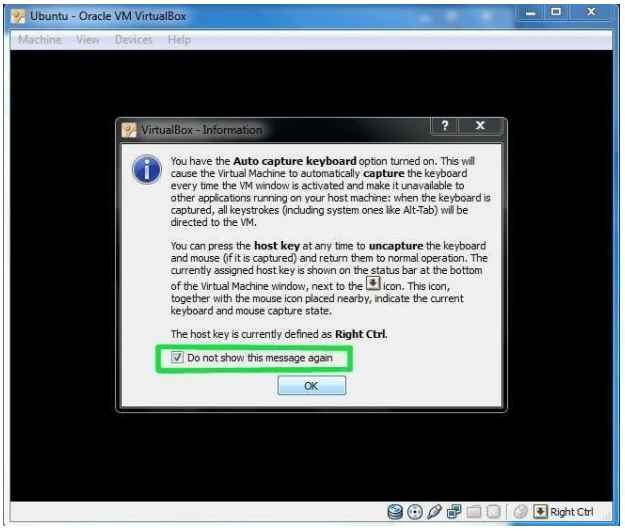
11. Now make the (currently blank) virtual hard drive useful is to add the downloaded **Ubuntu disk image (the .iso) boot on virtual machine. Click on Settings and Storage. Then, under *CD/DVD Device*, next to *Empty*,Click the folder icon.**



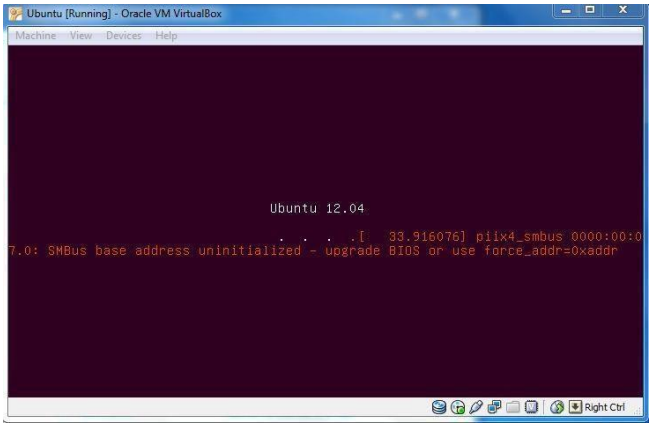
12. Select the Ubuntu .iso that was downloaded earlier.



13. After selection, click OK. Then double-click virtual machine to start it up.

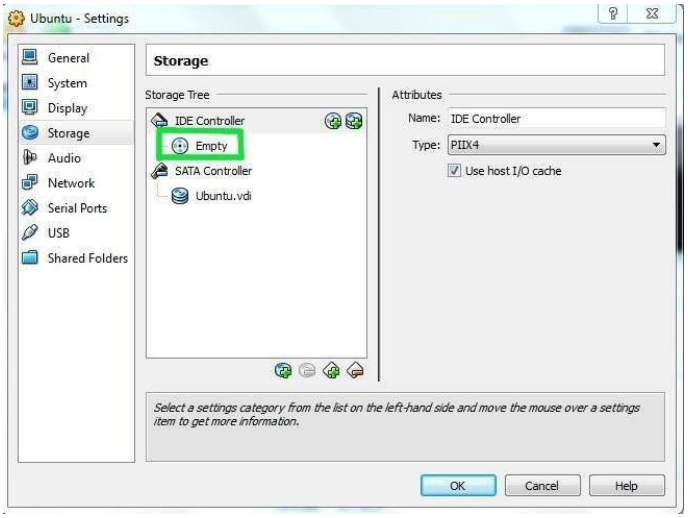


14. A bunch of random warnings/instructions about how to operate the guest operating system within VirtualBox.



15. Wait for Ubuntu to boot up.

16. Once it is started up, just follow the regular installation procedure



17. In order to use virtualized installation (instead of continually booting the live CD), double-check that the *CD/DVD Device* entry is Empty again

Conclusion: Thus, we have studied virtualization and undergone the steps of installing Ubuntu using virtual box