****

**Microprocessors and Embedded systems Lab File.**

**Registration No. R142218019**

**SAP-ID 500069161**

**Submitted By Bishal Changder.**

**Submitted to:**

**Mr. Devendra Rawat**

**List of Index:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **S.No.** | **Objective of the Experiments** | **Date** | **Remarks** | **Page no.** |
| **1.** | 1A - Write a program using 8085 & verify for addition of Two 8-Bit numbers and Addition of Two 16-Bit Numbers 1B - Write a program using 8085 & verify for Subtraction of Two 8-Bit Numbers & Subtraction of Two 16-Bit Numbers.  1C - Write a program using 8085 & verify for Division of Two 8-Bit Numbers by Repeated Subtraction Method Division of Two 8-Bit Numbers by Repeated Subtraction  Method. | 19 August 2020 |  | 8-18 |
| **2.** | Writing Program for 1’s complement of eight-bit number, 2’s complement of eight-bit number, 1’s complement of sixteen-bit number, 2’s complement of  sixteen-bit number. | 02 September  2020 |  | 19-26 |
| **3.** | Writing Program for mask off least significant 4 bits of an eight-bit number & mask off Most significant 4 bits of an eight-bit number. | 09 September  2020 |  | 27-28 |
| **4.** | Write a program to shift an 8-bit number left by 1 bit. Write a program to shift an 8-bit number left by 2 bits Write a program to shift a 16-bit number left by 1 bit  Write a program to shift a 16-bit number left by 2 bits. | 16 September  2020 |  | 29-39 |
| **5.** | To find square from lookup table. To find larger of two numbers.  To find smaller of two numbers.  To find larger number in a data array. To find smaller number in a data array.  To find the sum of series of 8-bit numbers. | 30 September  2020 |  | 40-62 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | To find the sum of series of 8-bit numbers but sum is a 16-bit number.  To arrange the data in ascending order. To arrange the data in descending order. |  |  |
| **6.** | Interfacing of 8051 Microcontroller with various display  devices. | 11 November  2020 | 63-68 |
| **7.** | Display number 2 ,9 and char A using SETB and CLR  instruction. | 18 November  2020 | 69-71 |
| **8.** | Interfacing of 8051 Microcontroller with DC motor. | 25 November  2020 | 72-77 |

**Short Questions and Answers**

Q1: What is Microprocessor?

A microprocessor is a computer processor that incorporates the functions of a central processing unit on a single (or more) integrated circuit (IC) of MOSFET construction. The microprocessor is a multipurpose, clock driven, register based, digital integrated circuit that accepts binary data as input, processes it according to instructions stored in its memory and provides results (also in binary form) as output.

Q2: What is the technology used in Microprocessor?

A microprocessor is an electronic component that is used by a computer to do its work. It is a central processing unit on a single integrated circuit chip containing millions of very small components including transistors, resistors, and diodes that work together.

Q3:  What is the function of Flag , Register, Program counter and Stack pointer?

A flag is a value that acts as a signal for a function or process. The value of the flag is used to determine the next step of a program. The Stack Pointer register will hold the address of the top location of the stack. And the program counter is a register always it will hold the address of the memory location from where the next instruction for execution will have to be fetched.

Q4: What are the meant by Low level and High level language?

A low-level language is a type of programming language that contains basic instructions recognized by a computer. ... Two common types of low-level programming languages are assembly language and machine language. Software programs and scripts are written in high-level languages, like C#, Swift, and PHP.

Q5: What is machine level programming?

Machine language, or machine code, is a low-level language comprised of binary digits (ones and zeros). High-level languages, such as Swift and C++ must be compiled into machine language before the code is run on a computer.

Q6: What jobs ALU of 8085 can perform?

The ALU takes two 8-bit inputs, which I'll call A and X, and performs one of five basic operations: ADD, OR, XOR, AND, and SHIFT-RIGHT. As well, if the input X is inverted, the ALU can perform subtraction and complement operations.

Q7: How many hardware interrupts 8085 supports. ?

There are 5 Hardware Interrupts in 8085 microprocessor.

Q8: How many I/P ports can 8085 access?

256 ports can be accessed.

Q9: Describe the accumulator register of 8085?

It is one of the most important 8 bit register of 8085

It is responsible for coordinating input and output to and from the microprocessor through it.The primary purpose of this register is to store temporary data and for the placement of final values of arithmetic and logical operations. This accumulator register is mainly used for arithmetic, logical, store and rotate operations.

Q10: What is the function of RESET IN and RESET OUT function

RESET IN − This signal is used to reset the microprocessor by setting the program counter to zero. RESET OUT − This signal is used to reset all the connected devices when the microprocessor is reset.

Q11.What is an Instructions?

The 8085A instructions can be classified into the following five functional categories. i) Data transfer (copy) operations, ii) Arithmetic operations , iii) Logical operations iv) Branching operations and v) Machine-control operations.

Q12. What is meant by Instruction Set?

The function of a microprocessor system is implemented by a sequence of data transfers between memory, processor and I/O devices and data transformations that occur in the registers within the microprocessor, manipulating a register under program control , addressing it and using it for data transfer and transformation requires a set of binary codes which comprise the INSTRUCTION SET OF A MICRO-PROCESSOR

Q13. In how many categories the instruction of 8085 be classified?

The instructions of 8085 can be classified into 5 categories.

Q14. What is meant by ‘addressing mode’ different mode? Mention the different addressing

mode?

Since data and instructions may reside anywhere in internal registers, external register or memory, locating them requires a particular addressing. The instructions in these five functional groups can be categorized according to their method of addressing the hardware registers and memory. This method is called the ADDRESSING MODE and six modes are available with 8085A which are explained in detail here, i) Implied addressing ii) Register addressing iii) Immediate addressing iv) Direct addressing v) Register indirect addressing vi) Combined addressing.

Q15. Mention the Interrupts pin of 8085?

There are 8 software interrupts in 8085, i.e. RST0, RST1, RST2, RST3, RST4, RST5, RST6, and RST7. Hardware interrupt − There are 5 interruptpins in 8085 used as hardware interrupts, i.e. TRAP, RST7. 5, RST6. 5, RST5.

Q16. Explain the maskable and Non-maskable Interrupt?

Maskable interrupt is a hardware Interrupt that can be disabled or ignored by the instructions of CPU.

A non-maskable interrupt is a hardware interrupt that cannot be disabled or ignored by the instructions of CPU.

Q17. What is meant by priority of interrupts?

The interrupt priority level is a part of the current system interrupt state, which indicates the interrupt requests that will currently be accepted.

Q18. What are the different types of data transfer operations possible?

The various types of data transfer operations possible are:  
  
Data transfer is possible between two registers.  
It is also possible between a memory location and a register.

Also it can occur between an input/output device and an accumulator.  
In reality data is never transferred it can only be copied from one location to another.

9. Mention the different type of operations with asthmatic, logical, branch & machine control

These type of instructions control machine functions such as Halt, Interrupt, or do nothing. This type of instructions alters the different type of operations executed in the processor.

Following are the type of Machine control instructions:

1. NOP (No operation)

2. HLT (Halt)

3. DI (Disable interrupts)

4. EI (Enable interrupts)

5. SIM (Set interrupt mask)

6. RIM (Reset interrupt mask)

Q20. What are the different instruction word sizes in 8085?

In 8085, the length is measured in terms of “byte” rather then “word” because 8085microprocessor has 8-bit data bus. Three types of instruction are: 1-byte instruction, 2-byte instruction, and 3-byte instruction.

**EXPERIMENT 1**

Date: 19-August,2020.

Registration Number: R142218019.

SAP-ID: 500069161

Aim:

1. To explain the architecture of the 8085 Micro-Processor.
2. To find the sum of 2, 8 Bit Numbers.
3. To find the sum 2 ,16 bit number.
4. To find the subtraction of 2, 8 bit numbers.
5. To find the subtraction of 2,16 bit numbers
6. To find the multiplication of 8 bit numbers
7. To find the division of 8 bit numbers

Apparatus Required:

Microprocessor Kit 8085, Key Board, Op-Code Sheet.

Theory:

The first data is brought to Accumulator A and the second one in any one of the other registers, say B. The addition is done using ADD. The result is then stored at 4252. The ADD instruction affects flags depending on result.

Procedure:

**Addition of 2, 8 bits number**

1.Get address of 1st no. in HL pair

2. Move no. into accumulator

3. HL points the address 7502 H

4. Add the 2nd no.

5. Terminate

6. Store input at the address

7. Get two 8 bit no. in successive location

8. Output: A- 12H, 7503- 12H

**Addition of 2, 16 bits number**

1. Get 1st no. in HL pair from memory 7601 H
2. Exchange cont. of DE HL
3. Get 2st no. in HL pair from location 7603 H
4. Clear reg. C.
5. Get HL+DE & store result in HL
6. If no carry move to loop/if carry then move to next step
7. Increment register .C
8. Move carry from reg. C to reg. A
9. Store carry at 7502 H
10. Store carry at 7502 H
11. Store result in 7500 H.
12. Store input at the address
13. Get two 16 bit no. in successive location

Program:

**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 7501 H

RST 1 // Terminate

// Store input at the address

// Get two 8 bit no. in successive location

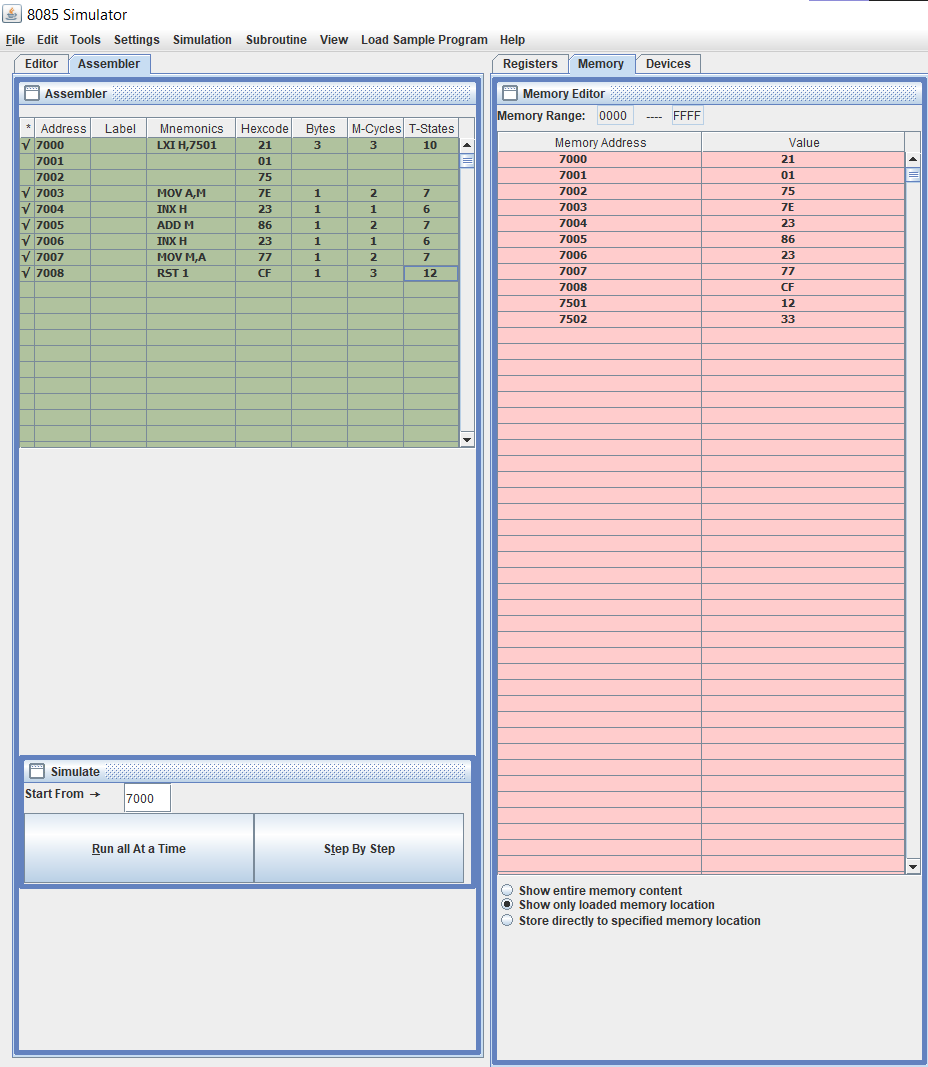
// Output: A- 12H, 7503- 12H

# ORG 7501H

# DB 12H, 3H

# INPUT: 7501- 9H, 7502- 3H

**RESULTS**

****

**Addition of Two 16-Bit Numbers**

# ORG 7000H

LHLD 7601 //Get 1st no. in HL pair from memory 7601 H

XCHG //Exchange cont. of DE HL

LHLD 7603 //Get 2st no. in HL pair from location 7603 H

MVI C,00 //Clear reg. C.

DAD D //Get HL+DE & store result in HL

JNC down //If no carry move to loop/if carry then move to next step.

INR C //Increment reg.C

MOV A,C //Move carry from reg. C to reg. A

STA 7502 //Store carry at 7502 H

down: SHLD 7500 //Store result in 7500 H.

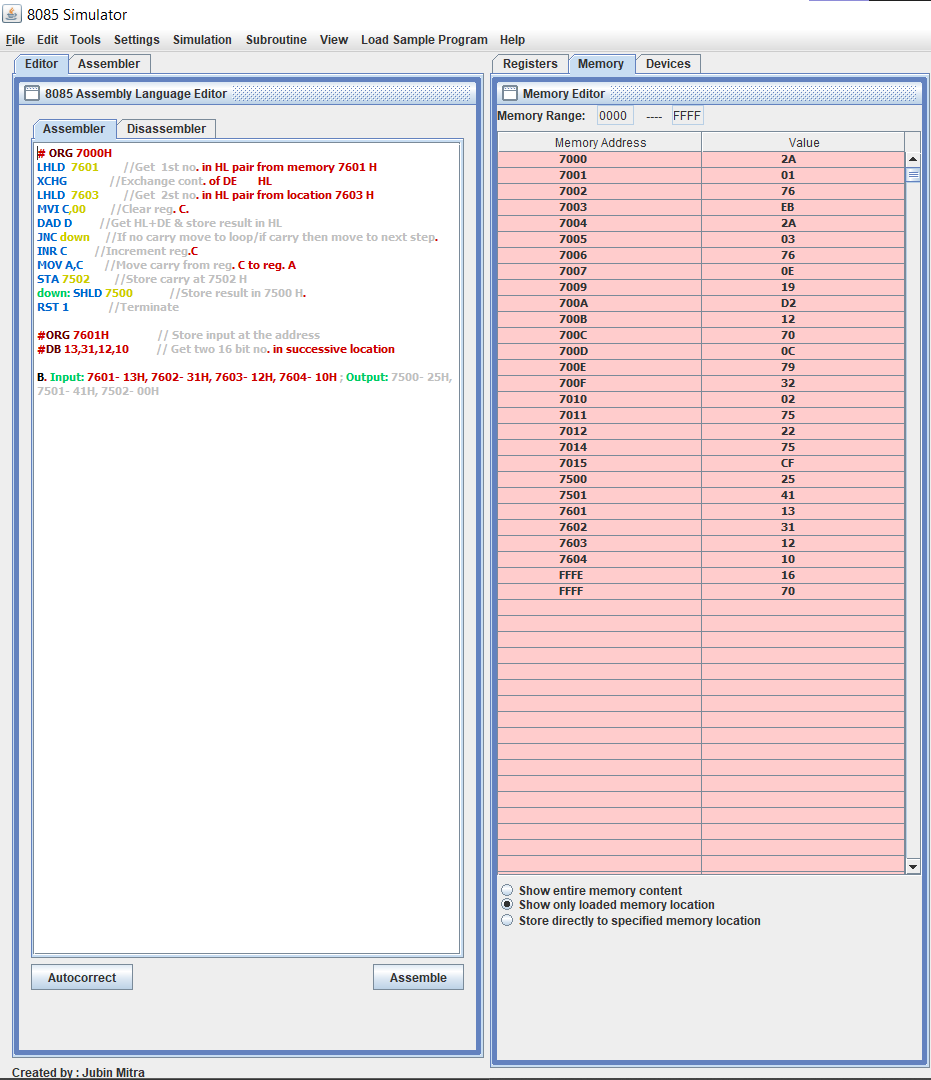
RST 1 //Terminate

#ORG 7601H // Store input at the address

#DB 13,31,12,10 // Get two 16 bit no. in successive location

B. Input: 7601- 13H, 7602- 31H, 7603- 12H, 7604- 10H ; Output: 7500- 25H, 7501- 41H, 7502- 00H

**RESULTS:**



**Subtraction of 2,8bit numbers**

**Substraction of Two 8-Bit Numbers**

# ORG 7000H

LXI H, 7501 // Get address of ist no. in HL pair

MOV A, M // Move no. into accumulator

INX H // HL points 7502 H.

SBB M // Substract 2nd no. from Ist no.

INX H //HL points 7503 H.

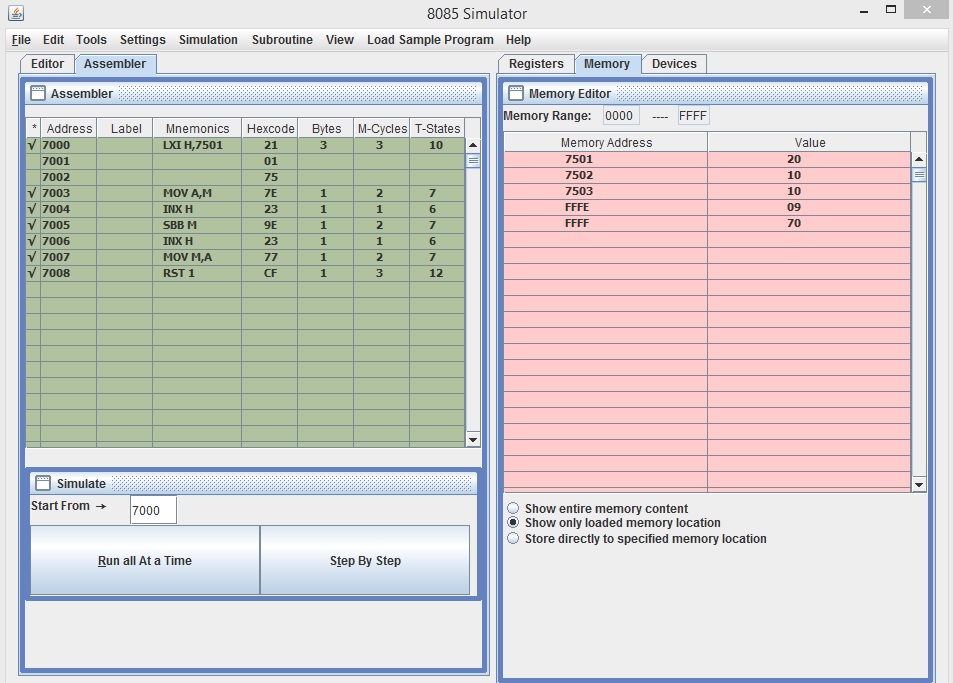
MOV M, A // Move contents of acc. to memory

RST 1 // Terminate

#ORG 7501H // Store no. at address

#DB 20,10 // Get the two 8 bit no. at successive location

A. Input: 7501- 20H, 7502- 10H ; Output: 7503- 10H

**RESULT** 

**Subtraction of two 16bit numbers**

**Subtraction of 16 bit two numbers**

#ORG address number.

MOV CX, 0000 // [CX] <- 0000

MOV AX, [3000] //[AX] <- [3000]

MOV BX, [3002] //[BX] <- [3002]

SUB AX, BX //[AX] <- [AX] – [BX]

JNC 2010 Jump if no borrow

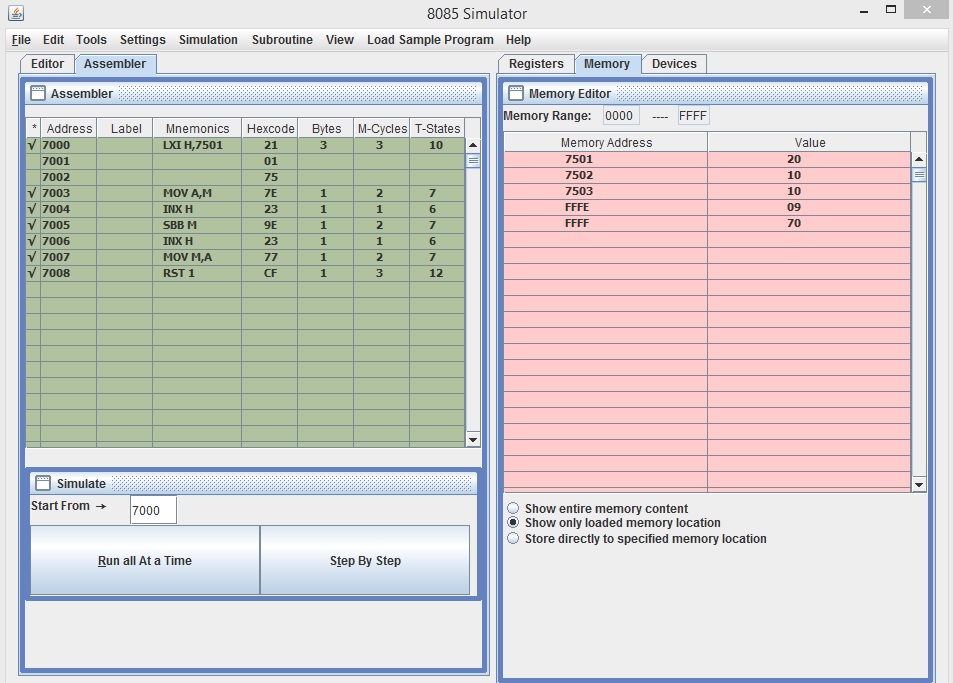
INC CX //[CX] <- [CX] + 1

MOV [3004], AX //[3004] <- [AX]

MOV [3006], CX //[3006] <- [CX]

HLT Stop

**RESULT**

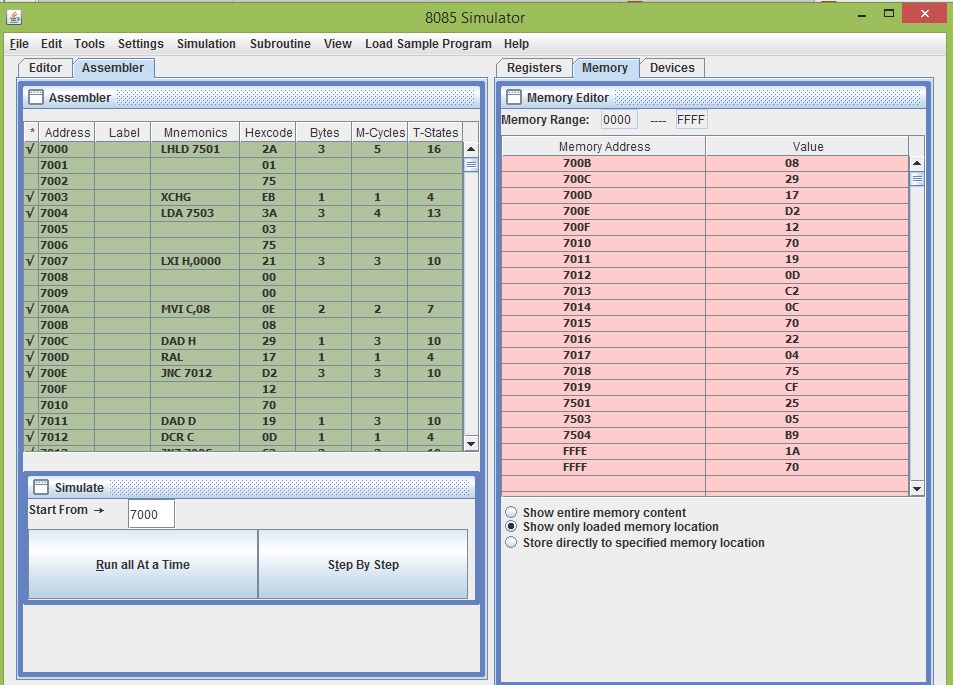


**Multiplication of Two 8-Bit Numbers by Bit Rotation Method.**

**PROCEDURE**

1. Get Multiplicand in H-L pair.
2. Exchange HL pair with DE pair.
3. Get 2nd no. in acc.
4. Initial product in HL=00 .
5. Count=08 in reg .C .
6. Shift partial product left by 1 bit .
7. Rotate multi. by 1 bit. Is multiplier = 1? .
8. No, go to ahead .
9. Product=Product + Multiplicand .
10. Decrement Count .
11. Jump until C=0 .
12. Store result .
13. Terminate .
14. Store inputs at the address.
15. Get the numbers from successive locations.

**RESULT:**

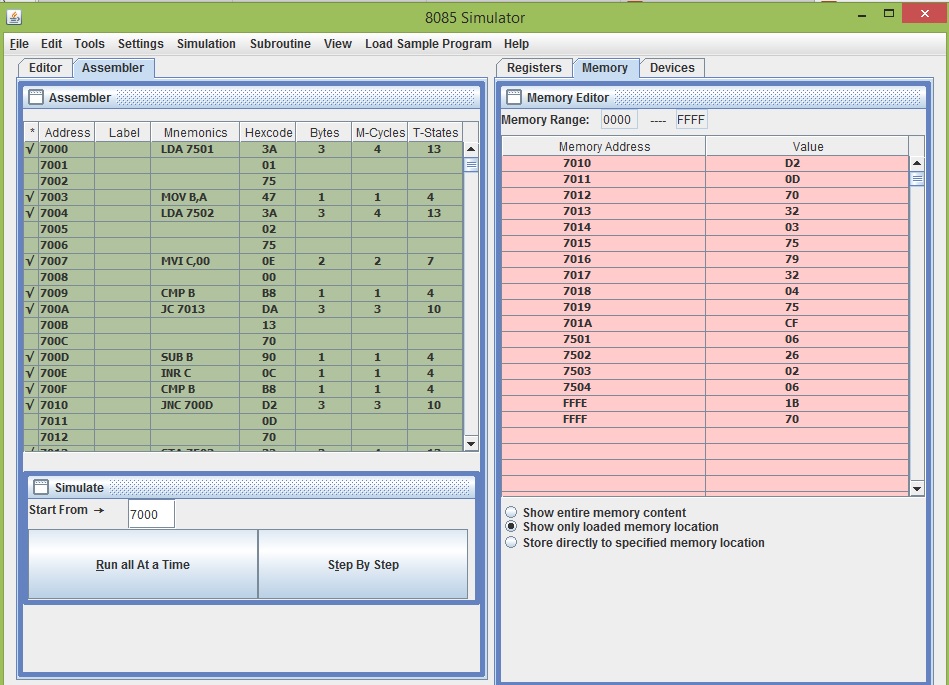


**Division of Two 8-Bit Numbers by Repeated Subtraction Method.**

**PROCEDURE**

1. [7501]=>A (Divisor)
2. Take divisor in reg,B
3. Take dividend in reg,A
4. Quotient=00
5. Compare A to B
6. Jump if carry
7. Dividend-divisor=>A
8. C=C+1
9. Is dividend < divisor
10. If not,go back
11. Store Remainder
12. C=>A
13. Store Quotient
14. Terminate
15. Store the inputs at the address
16. Get the numbers from successive loc.

**RESULT:**



**Experiment no.2:**

**Aim**

Program for:

a. 1’s complement of 8 bit number.

b. 2’s complement of 8 bit number

c.  1’s complement of 16  bit number

d.  2’s complement of 16 bit number

**Apparatus Required :**

Microprocessor Kit 8085 , Key Board, Op-Code Sheet.

**Theory :**

**1.Program for 1’s  complement of 8-bit  number**

In this experiment to obtain one’s compliment of a number its 0 bits are replaced by 1 and 1 by 0.

The number is placed in the memory location 4250.

The result is stored the memory location 4251

**2.Program for 2’s complement of 8-bit  number**

In this experiment to obtain Two’s compliment of a number is obtain by adding 1 to the 1’s compliment of the number.

The number is placed in the memory location 4250.

The result is to be stored in the memory location  4251.

**3.Program for 1’s  complement of 16-bit  number**

In this experiment, The 8 LSBs of the number are in the memory location 4250. The address 4250 is placed in H-L pair. The 8 LSBs of the number are transferred from 4250 to the accumulator. The instruction CMA takes one’s complement of 8 LSBs. The 8 LSBs of the result are stored in the memory location 4252. The address 8 MSBs of the number is 4251, and it is placed in H-L pair .The 8 MSBs of the number are transferred from 4251 to the accumulator. The instruction CMA takes one’s compliment of 8 MSBs. The 8 MSBs of the result are stored in memory location 4253.

**4.Program for 2’s  complement of 16-bit  number**

In this experiment, The 8 LSBs of the number are in memory location 4251. They are taken first and 1’s compliment is obtained. To obtain 2’s complement 1 is added to 1’s complement. 8 LSBs of 2’s complement are stored in 4253. The carry resulting from the addition of 1 to 1’s complement is stored in register. After this 8 MSBs of the number are taken and 1’s complement is obtained. They carry is added to it. The 8 MSBs of the result are stored in 4254. In case of no carry the program jumps from JNC GO to INX H and the content of register B is not incremented. It remains zero. The addition of zero to 1’s complement of 8 MSBs does not affect result.

**Procedure:**

**EXPERIMENTAL PROCEDURE**: **STEP WISE – 1's complement of 8 bit number**

**i)** Key in the Opcodes from the address  specified.

**ii)**Enter data at 4250 specified in the example.

**iii)**Execute the program and check for the result at 4251.

**iv)**Change data at 4250 and execute each time and check for result.

**EXPERIMENTAL PROCEDURE: STEP WISE – 2's complement of 8 bit number**

**i)**Key in the Opcodes from the address  specified.

**ii)**Enter data at 4250 specified in the example.

**iii)**Execute the program and check for the result at 4251.

**iv)**Change data at 4250 and execute each time and check for result.

**EXPERIMENTAL PROCEDURE: STEP WISE – 1's complement of 16 bit number**

        i) Key in the Op-codes from the address specified.

        ii) Enter data at 4251 and 4252 specified in the example.

        iii) Execute the program and check for the result at 4253 and 4254.

        iv)Change data at 4251 and 4252 and execute each time and check for result.

**EXPERIMENTAL PROCEDURE: STEP WISE – 2's complement of 16 bit number**

i) Key in the Op-codes from the address specified.

ii)  Enter data at 4251 and 4252 specified in the example.

iii) Execute the program and check for the result at 4253 and 4254.

iv)  Change data at 4251 and 4252 and execute each time and check for result.

**Practical:**

**EXPERIMENTAL PROCEDURE**: **STEP WISE – 1's complement of 8 bit number**

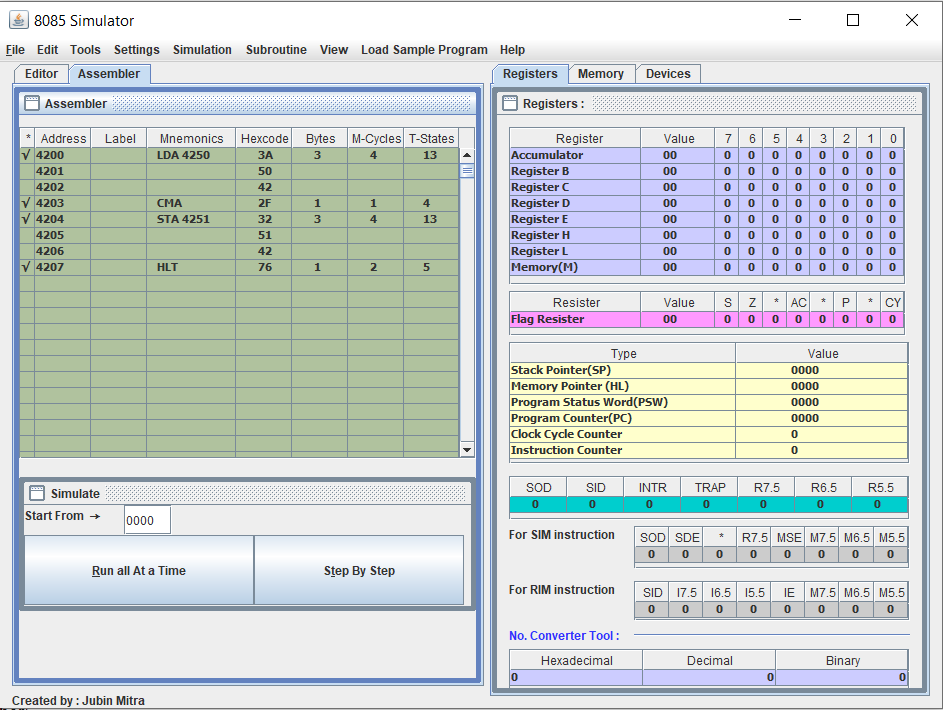
Step wise procedure:

1. Get data in accumulator
2. Take its complement.
3. Store result in 4251
4. Stop

Code:

#ORG 4200

1. LDA 4250
2. CMA
3. STA
4. HLT

Output:

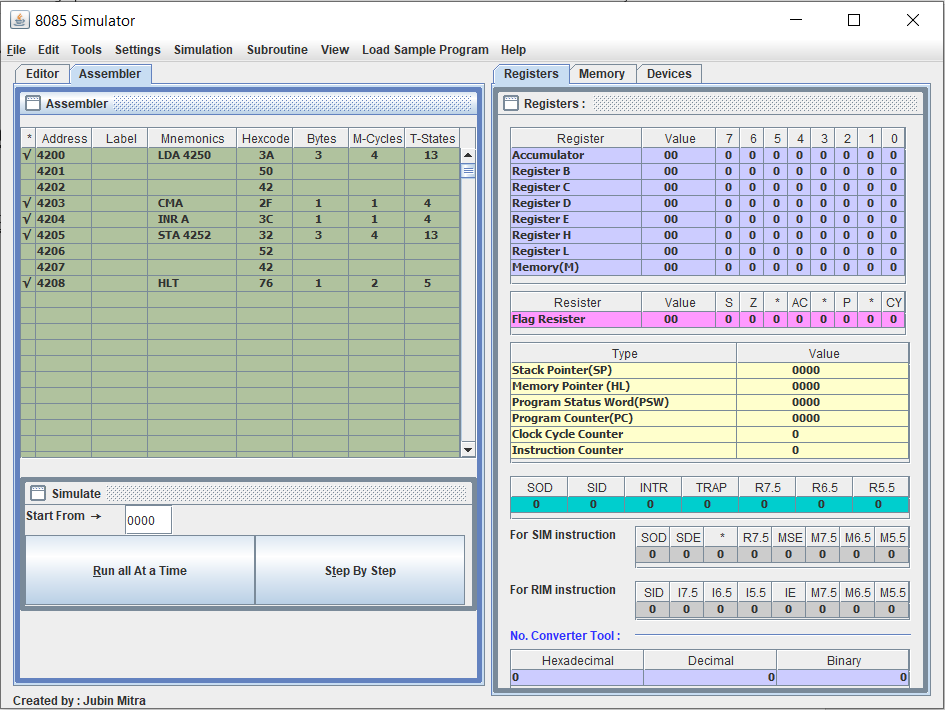
**EXPERIMENTAL PROCEDURE: STEP WISE – 2's complement of 8 bit number**

Step wise procedure:

1. Get data in accumulator
2. Take its 1’s complement.
3. Take 2’s complement.
4. Store result in 4251.
5. Stop

Code:

1. LDA 4250
2. CMA
3. INR A
4. STA 4252
5. HLT

Output:

**EXPERIMENTAL PROCEDURE: STEP WISE – 1's complement of 16 bit number**

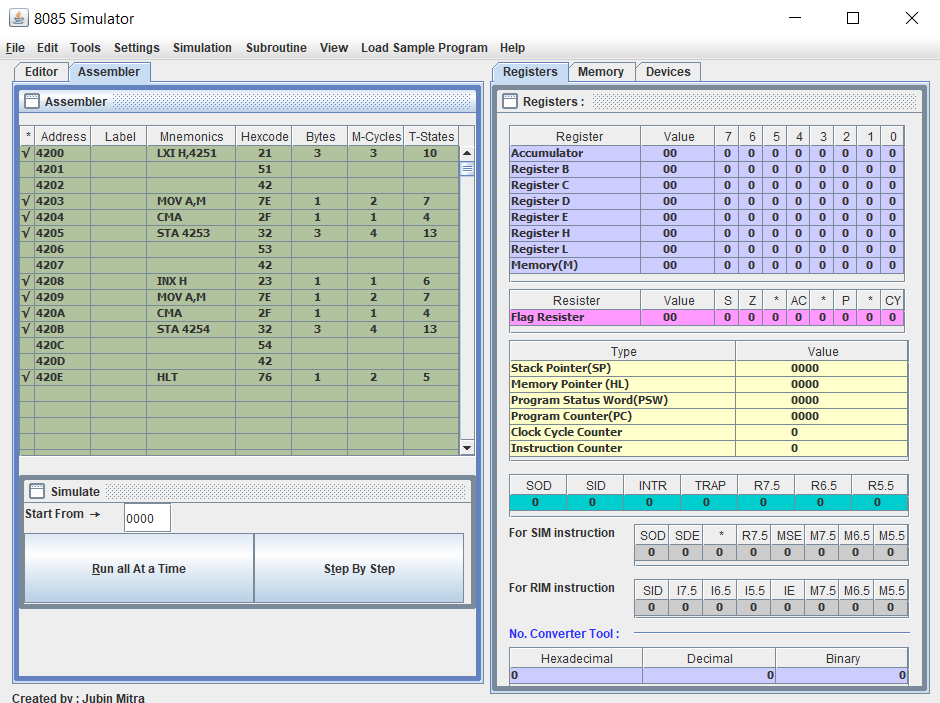
Step wise procedure:

1. Address OF LSBs of the number.
2. 8 LSBs of the number in accumulator.
3. Complement of 8 LSBs of the number.
4. Store 8 LSBs of result.
5. Address of 8 MSBs of the number.
6. 8 MSBs of the number in accumulator.
7. Complement of 8 MSBs of the number.
8. Store 8 MSBs of the result.
9. Stop

Code:

1. LXI H, 4251
2. MOV A,M
3. CMA
4. STA 4253
5. INX H
6. MOV A, M
7. CMA
8. STA 4254
9. HLT

Output:



**EXPERIMENTAL PROCEDURE: STEP WISE – 2's complement of 16 bit number**

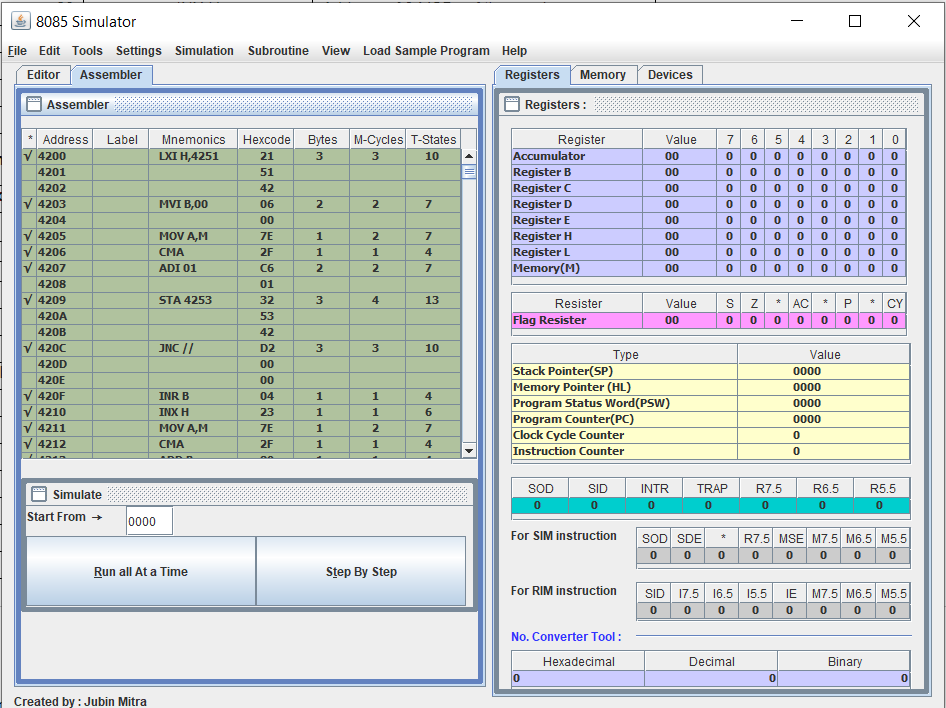
Step wise procedure:

1. Address OF LSBs of the number.
2. Use registers B to store carry.
3. 8 LSB in accumulator.
4. 1’s Complement of 8 LSBs of the number.
5. 2’s complement of 8 LSBs of the number
6. Store 8 LSBs of result.
7. Store carry.
8. Address of 8 MSBs of the number.
9. 8 MSBs in accumulator.
10. 1’s Complement of 8 LSBs of the number.
11. Add carry
12. Store 8 MSBs of the result.
13. Stop.

Code:

1. LXI H,4251
2. MVI B,00
3. MOV A,M
4. CMA
5. ADI 01
6. STA 4253
7. JNC ; GO
8. //Label
9. INR B
10. GO; INX H
11. MOV A,M
12. CMA
13. ADD B
14. STA 4254
15. HLT
16. Stop

Output:



**Experiment no.3:**

**Aim:**

1. Mask off Least significant 4 bits of an eight bit number

2. Mask off Most significant 4 bits of an eight bit number

**Apparatus Required:**

Microprocessor Kit 8085 , Key Board, Op-Code Sheet.

**Theory:**

Mask off Least significant 4 bits of an eight-bit number:

Get data in accumulator using LDA operation

Mask off the most significant 4 bits using ANI F0

Store result in 4252 using STA 4252

Stop using HLT

Mask off Most significant 4 bits of an eight-bit number:

Get data in accumulator using LDA.

Mask off the most significant 4 bits using ANI 0F

Store result in 4252 using STA 4252

Stop using HLT.

Precautions:

1. All steps should be followed carefully.

2. Make sure all power sources are disconnected

3. Make sure you are properly grounded.

4. Don’t touch the live wire.

Procedure:

Mask off Least significant 4 bits of an eight-bit number:

1. Key in the Opcodes from the address specified.

2. Enter data at 4250 specified in the example.

3. Execute the program and check for the result at 4251.

4. Change data at 4250 and execute each time and check for result.

Mask off Most significant 4 bits of an eight-bit number:

1. Key in the Opcodes from the address specified.

2. Enter data at 4250 specified in the example.

3. Execute the program and check for the result at 4251.

4. Change data at 4250 and execute each time and check for result.

Practical:

Mask off Least significant 4 bits of an eight-bit number:

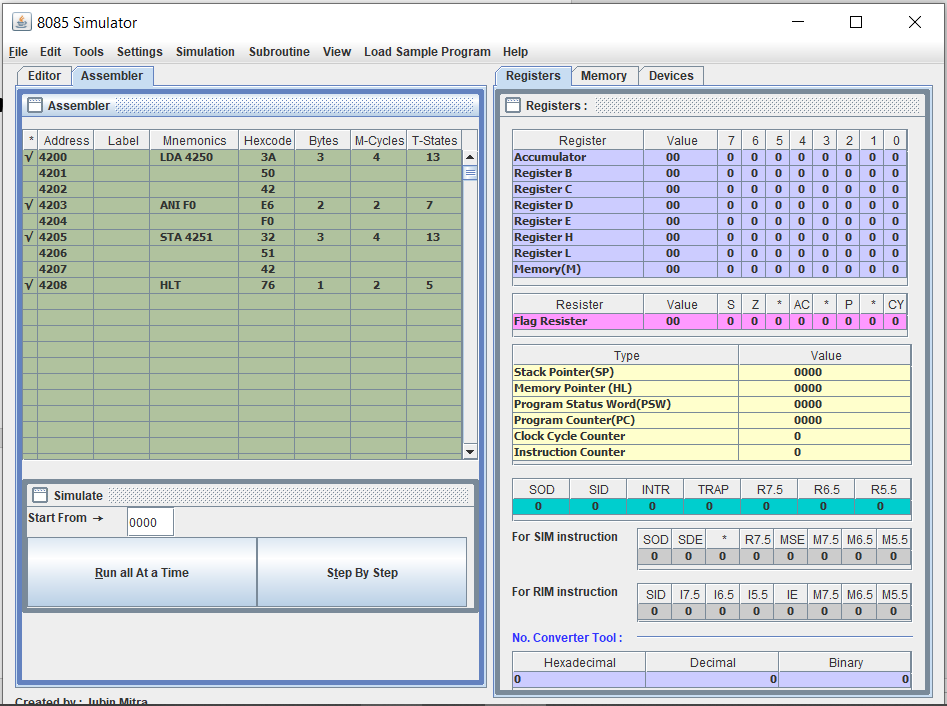
Step wise Procedure:

1. Get data in accumulator
2. Mask off the most significant 4 bits
3. Store result in 4252
4. Stop

Code:

1. #ORG 4200
2. LDA 4250
3. ANI F0
4. STA 4251
5. HLT

Output:



Mask off Most significant 4 bits of an eight-bit number:

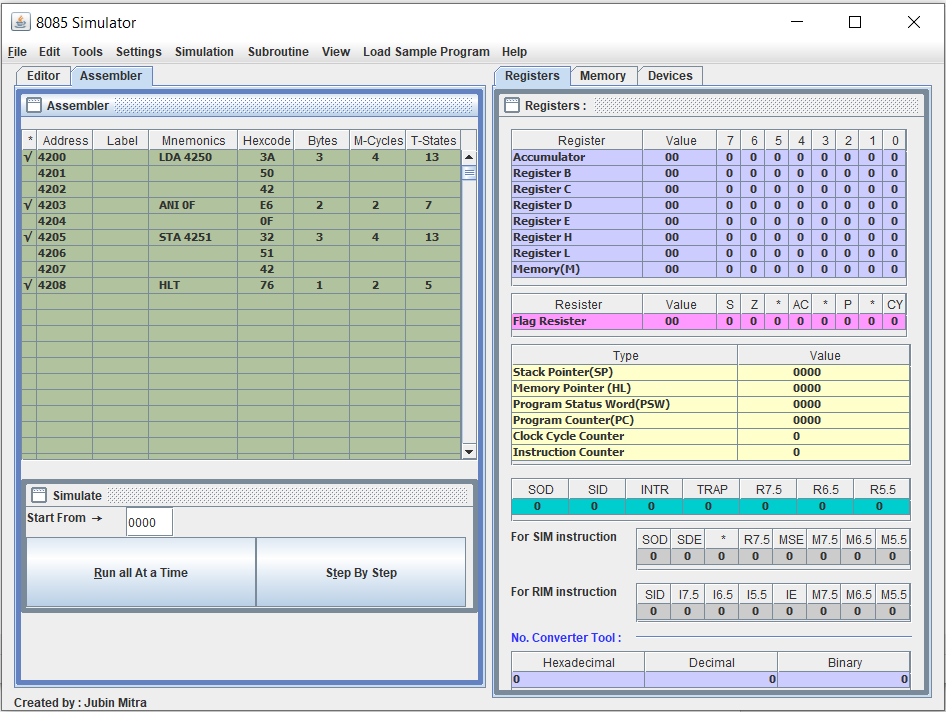
Step wise Procedure:

1. Get data in accumulator
2. Mask off the most significant 4 bits
3. Store result in 4252
4. Stop

Code:

1. #ORG 4200
2. LDA 4250
3. ANI 0F
4. STA 4251
5. HLT

Output:



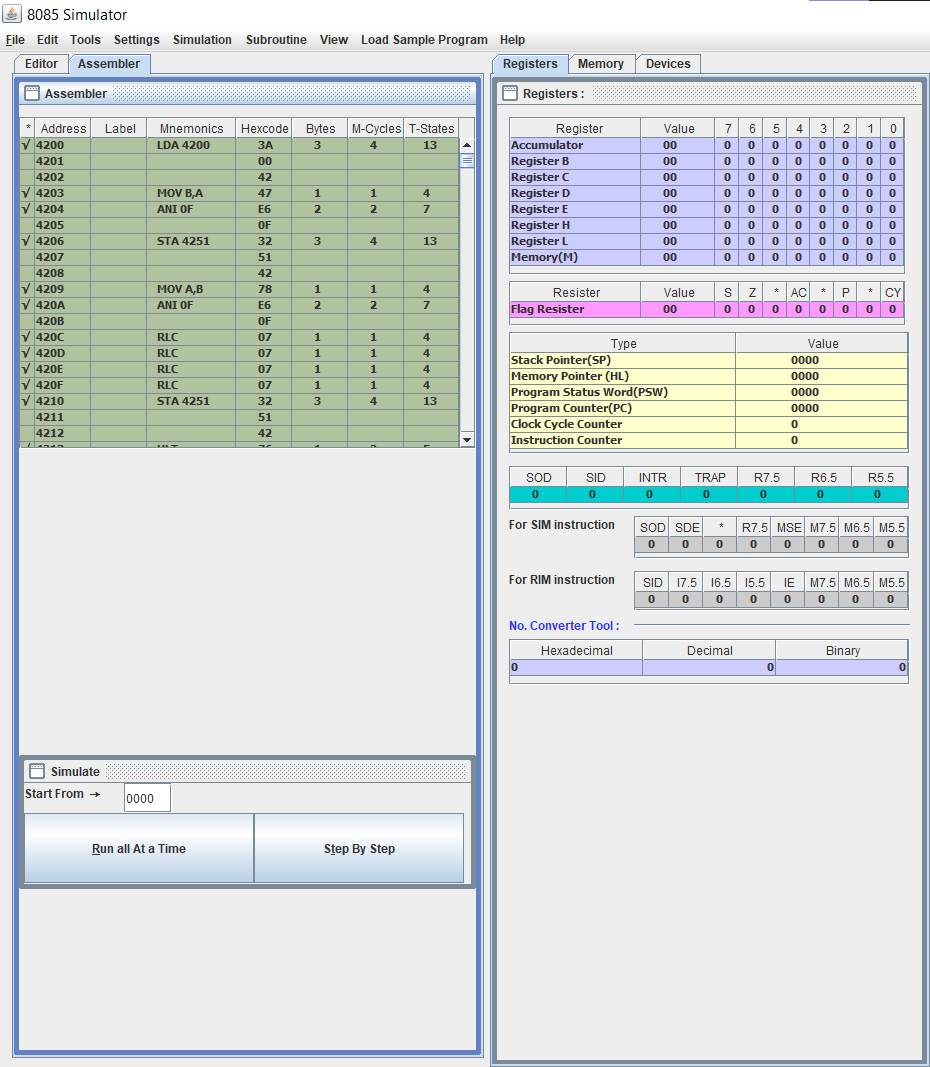
Alternate Code:

1. #ORG 4200
2. LDA 4200
3. MOV B, A
4. ANI 0F
5. STA 4251
6. MOV A, B
7. ANI 0F
8. RLC
9. RLC
10. RLC
11. RLC
12. STA 4251
13. HLT

Stepwise Procedure:

1. A <- M[4200]
2. B <- A
3. //Perform masking of least significant bit half nibble.
4. A <- A (AND) 0F
5. M[4251] <- A
6. A <- B
7. ///Perform masking of upper Significant Bits upper Nibble
8. A <- A (AND) 0F
9. Rotation with Steps.
10. rotate content of A left by 1 bit without carry
11. rotate content of A left by 1 bit without carry
12. rotate content of A left by 1 bit without carry
13. rotate content of A left by 1 bit without carry
14. M[4251] <- A
15. END

Output:



Experiment 4

Aim:

Shifting left an 8-bit number by 1 bit Shifting left an 8-bit number by 2 bit Shifting left a 16-bit number by 1 bit Shifting left a 16-bit number by 2 bit

Apparatus Required: Microprocessor Kit 8085, Key Board, Op-Code Sheet.

Theory:

Shifting left an 8-bit number by 1 bit

In this experiment to obtain, the instruction LDA 4251 transfers the number from memory location 4251 to the accumulator. ADD A adds the contents of the accumulator to itself. The results are twice the number and thus the number is shifted left by one bit. This program does not take carry account after ADD instruction. If the number to be handled is likely to produce carry the program may be modified to store it.

Shifting left an 8-bit number by 2 bit

In this experiment to obtain, the instruction LDA 4251 transfers the number from memory location 4251 to the accumulator. We have to use two ADD A it adds the contents of the accumulator to itself twice. The results are twice the number and thus the number is shifted left by two-bit. This program does not take carry account after ADD instruction. If the number to be handled is likely to produce carry the program may be modified to store it.

Shifting left a 16-bit number by 1 bit

In this experiment to obtain, the instruction LHLD 4251 transfers the number from memory location 4251 and 4252 to the accumulator. DAD H adds the contents of the accumulator to itself twice. The results are twice the number and thus the number is shifted left by one bit. This program does not take carry account after DAD instruction. If the number to be handled is likely to produce carry the program may be modified to store it.

Shifting left a 16-bit number by 2 bit

In this experiment to obtain, the instruction LHLD 4251 transfers the number from memory location 4251 and 4252 to the accumulator. We have to use two DAD H it adds the contents of the accumulator to itself twice. The results are twice the number and thus the number is shifted left by two-bit. This program does not take carry account after DAD instruction. If the number to be handled is likely to produce carry the program may be modified to store it.

Procedure:

Shifting left an 8-bit number by 1 bit

1. Key in the Opcodes from the address specified.
2. Enter data at 4250 specified in the example.
3. Execute the program and check for the result at 4251.
4. Change data at 4250 and execute each time and check for the result.

Shifting left an 8-bit number by 2 bit

1. Key in the Opcodes from the address specified.
2. Enter data at 4250 specified in the example.
3. Execute the program and check for the result at 4251.
4. Change data at 4250 and execute each time and check for the result.

Shifting left a 16-bit number by 1 bit

1. Key in the Opcodes from the address specified.
2. Enter data at 4250 specified in the example.
3. Execute the program and check for the result at 4252.
4. Change data at 4250 and execute each time and check for the result.

Shifting left a 16-bit number by 2 bit

1. Key in the Opcodes from the address specified.
2. Enter data at 4250 specified in the example.
3. Execute the program and check for the result at 4252.
4. Change data at 4250 and execute each time and check for the result.

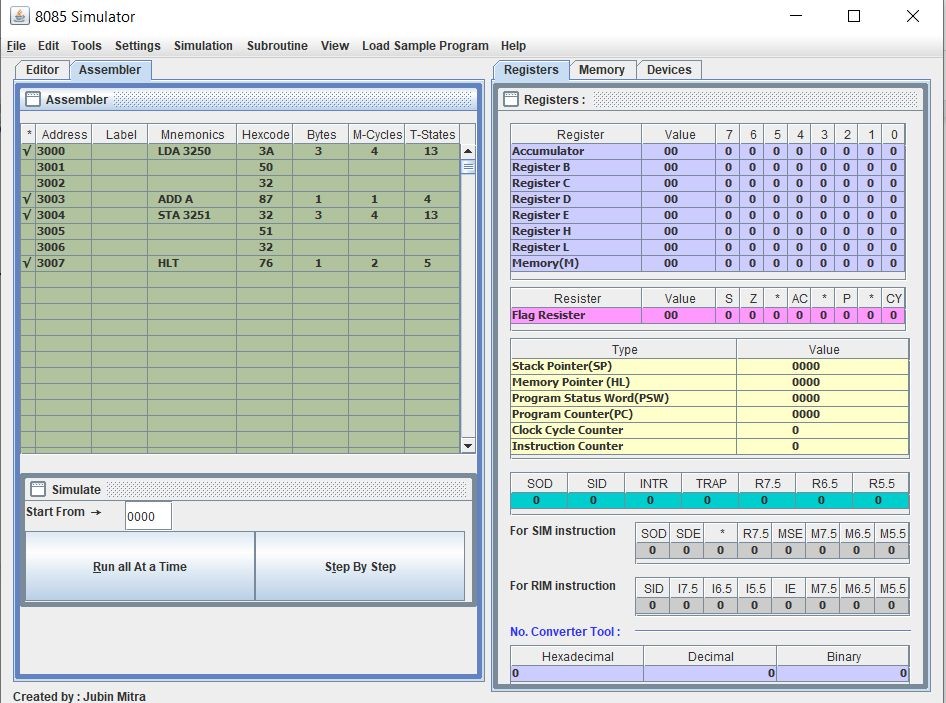
Program:

Shifting left an 8-bit number by 1 bit

# ORG 3000H LDA 3250 ADD A

STA 3251 HLT

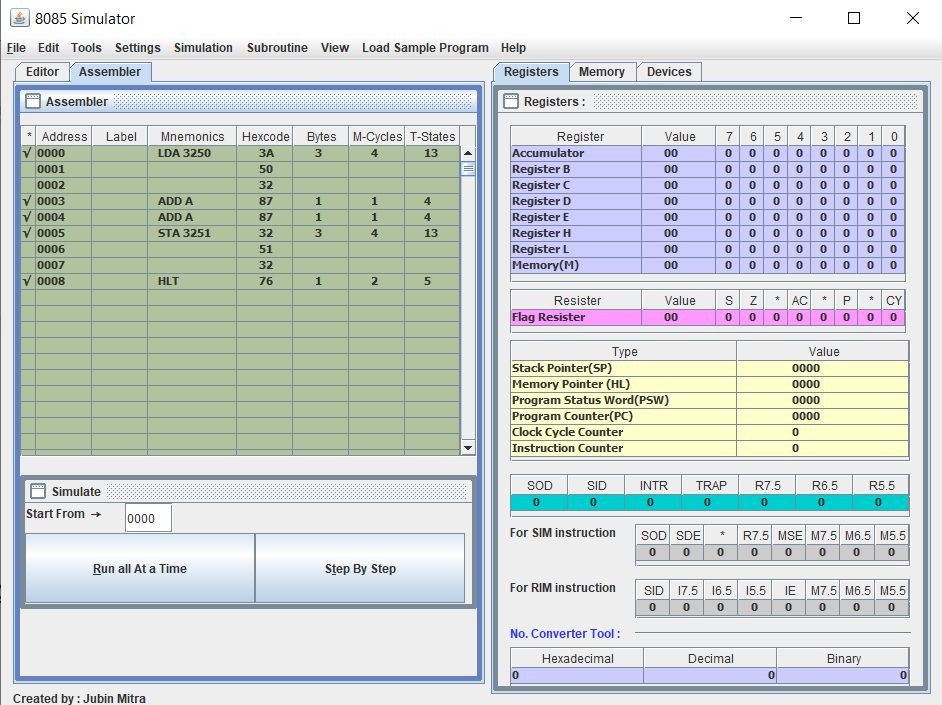
# ORG 3250 # DB 25H

 Output 1:

Shifting left an 8-bit number by 2 bit

LDA 3250 ADD A ADD A STA 3251 HLT

# ORG 3250 # DB 25H

 Output 2:

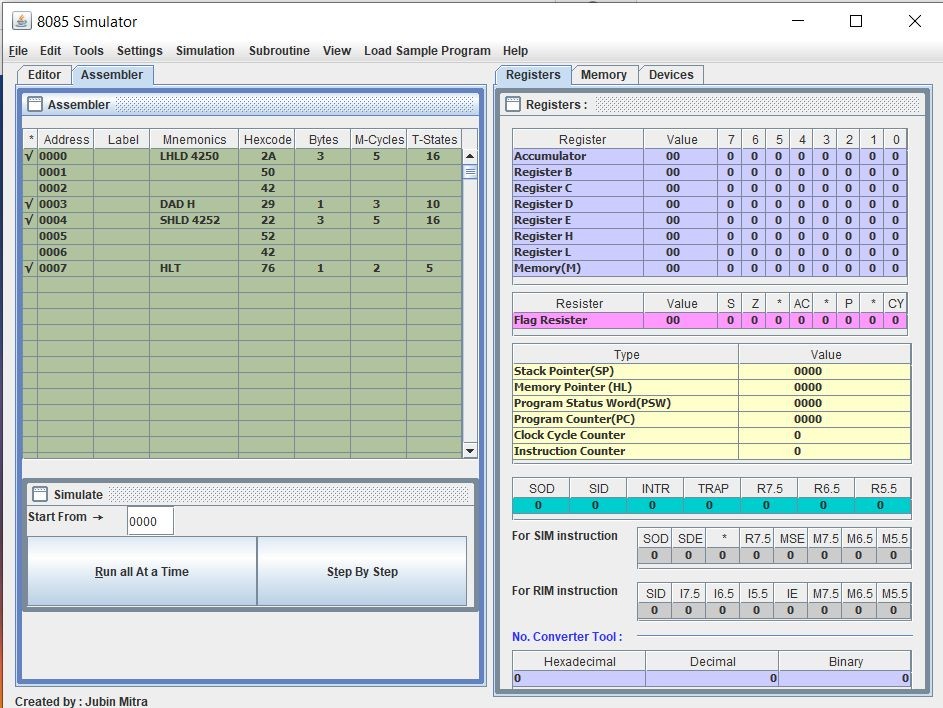
Shifting left a 16 bit number by 1 bit

LHLD 4250 DAD H SHLD 4252 HLT

# ORG 4250

# DB 05H,20H

Output 3:



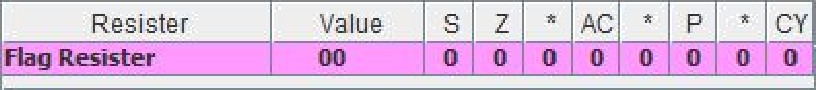
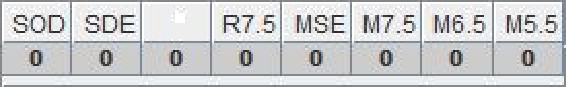
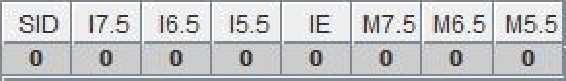
Shifting left a 16-bit number by 2 bit

LHLD 4250 DAD H DAD H SHLD 4252 HLT

# ORG 4250

# DB 05,20

Output 4:



8085 Simulator

Fñe Edit Tools Setflags Serb+ilañon Mbrouâne View Load Sample Program Bdlp

Editor Asxemt er Itegzsters 34emory 0e ñces

Registers: ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

” Addres s Label Mnemonics Hexcode Bytes M- de s T-States

Register

Value

7 6 5 4 3 2 1 0

9egpste E.

Resistor

Value

S Z ” AC ” P ” CY

Type

Vaue

Simulate::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

0000

SOD

SID

INTR

TRAP

R7.5

R6 S

RS.5

For S0d instmcfion q :DE

R7.5 MSE MW.S M6 S MS S

Run aB At a Jane

Na. Converter To6-I

Hexadecimal

Decimal

Bin ary

**EXPERIMENT: 5**

**Aim:**

1. To find square from lookup table.
2. To find larger of two numbers.
3. To find smaller of two numbers.
4. To find larger number in a data array.
5. To find smaller number in a data array.
6. To find the sum of series of 8-bit numbers.
7. To find the sum of series of 8-bit numbers but sum is a 16-bit number.

**Apparatus Required:** Microprocessor Kit 8085, Key Board, Op-CodeSheet.

**Theory:**

We are putting away the square outcome into the memory. At the point when the number is more noteworthy than F, we are putting away FFH into memory to demonstrate an ERROR. We are taking the number from area 8000H, and the query table is put away at the 9000H area. We can locate the square of the numbers at various spaces.

The arrangement is to instate max as first component, at that point cross the given exhibit from second component till end. For each navigated

Component, contrast it and max, on the off chance that it is more noteworthy than max, at that point update max.

In this program the information are put away at area 8001H onwards. The 8000H is containing the size of the square. Subsequent to executing this program, it will restore the most modest number and store it at area 9000H.

Rationale is basic, we are taking the primary number at register B to begin the activity. In every emphasis we are getting the number from memory and putting away it into register A. At that point in the event that B > An, at that point we basically update the estimation of B with An, in any case go for the following cycle. Consequently, we can locate the most modest number in a square of bytes.

Utilizing area 8000H to hold the length of the square. The principle block is put away from address 8010H. We are putting away the outcome at area 9000H and 9001H. The 9000H holding the lower byte, and 9001H is holding the upper byte.

Over and over we are taking the number from the memory, at that point including it with collector and increment the register E content when convey banner is set. At first E is cleared.**Precautions:**

1. All steps should be followed carefully.
2. Make sure all power sources are disconnected
3. Make sure you are properly grounded.
4. Don’t touch the live wire.

**Procedure:**

**To find square from lookup table:**

1. Assign 20 to register H, 50 to register L and 00 to accumulator A.
2. Load the content of memory location which is specified by M in register B.
3. Add content of M in accumulator A and decrement value of B by 01.
4. Check if B holds 00, if true then store the value of A at memory location 3050 otherwise go to step 3.

**To find larger of two numbers:**

1. Load value in the accumulator.
2. Then, copy the value to any of the register.
3. Load next value in the accumulator.
4. Compare both values.
5. Check carry flag, if reset then jump to the required address to store the value.
6. Copy the result in the accumulator.7.Store the result at the required address.

**To find smaller of two numbers:**

1. Load the content from memory location.
2. Move content of Accumulator into Register B.
3. Load the content from Memory location.
4. Compare the content of Register B.
5. If carry flag is equal to 1 go to step 7.
6. Move content of Register B into Accumulator.
7. Store the content into Memory.
8. End of program.

**To find larger number in a data array:**

1. We are taking first element of array in A.
2. Comparing A with other elements of array, if A is smaller then store that element in A otherwise compare with next element.
3. The value of A is the answer.

**To find smaller number in a data array:**

1. Load the address of the first element of the array in HL pair.
2. Move the count to B - reg.
3. Increment the pointer.
4. Get the first data in A - reg.
5. Decrement the count.
6. Increment the pointer.
7. Compare the content of memory addressed by HL pair with that of A - reg.
8. If carry = 1, go to step 10 or if Carry = 0 go to step 9.
9. Move the content of memory addressed by HL to A - reg.
10. Decrement the count.
11. Check for Zero of the count. If ZF = 0, go to step 6, or if ZF = 1 go to next step.
12. Store the smallest data in memory.
13. Terminate the program.

**To find the sum of series of 8-bit numbers:**

1. This program finds the sum of numbers in an array.
2. In order to find the sum of numbers, first, the counter must be initialized with the size of an array and accumulator must be initialized to zero.
3. Then, the first number is moved to register B and added with the accumulator.
4. After addition, the counter is decremented and checked whether it has reached zero. If it has, the loop terminates otherwise, the next number is moved to register B and added.
5. Let us assume that the memory location 3000H stores the counter. The next memory locations store the array.
6. Initially, H-L pair is loaded with the address of the counter and is moved to register C.
7. Accumulator is initialized with 00H.
8. Then, H-L pair is incremented to point to the first number in the array and it is moved to register B.
9. Register B is added to the accumulator and the result is stored in the accumulator.
10. Then, the counter is decremented and checked whether it has become zero.
11. If it hasn’t become zero, it means there are numbers left in the array. In this case, the control jumps back to increment the H-L pair and moves the next number to register B.
12. This process continues until counter becomes zero, i.e. all the numbers in the array are added.
13. At last, H-L pair is incremented and the result is moved from accumulator to memory.

**To find the sum of series of 8-bit numbers but sum is a 16-bit number:**

1. This program finds the sum of numbers in an array.
2. In order to find the sum of numbers, first, the counter must be initialized with the size of an array and accumulator must be initialized to zero.
3. Then, the first number is moved to register B and added with the accumulator.
4. After addition, the counter is decremented and checked whether it has reached zero. If it has, the loop terminates otherwise, the next number is moved to register B and added.
5. Let us assume that the memory location 3000H stores the counter. The next memory locations store the array.
6. Initially, H-L pair is loaded with the address of the counter and is moved to register C.
7. Accumulator is initialized with 00H.
8. Then, H-L pair is incremented to point to the first number in the array and it is moved to register B.
9. Register B is added to the accumulator and the result is stored in the accumulator.
10. Then, the counter is decremented and checked whether it has become zero.
11. If it hasn’t become zero, it means there are numbers left in the array. In this case, the control jumps back to increment the H-L pair and moves the next number to register B.
12. This process continues until counter becomes zero, i.e. all the numbers in the array are added.
13. At last, H-L pair is incremented and the result is moved from accumulator to memory.

**Program Code and Object code:**

**Program to find square from lookup table:**

* ORG 2000H
* LDA 2500

HLT

STA 2501

* ORG 2500
* DB 07
* ORG 2600
* DB 00, 01, 02, 09, 16, 25, 36, 49, 64, 81

OR

LDA 8050

ADI 60

MOV L, A

MVI H,80

MOV A, M

STA 8051

HLT

* ORG 8060
* DB 00H,01H,04H,09H,16H,25H,36H,49H,64H,81H
* ORG 8050
* DB 09

**Program to find larger of two numbers:**

# ORG 2000H

LXI H,2501

MOV A, M

INX H

CMP M

JNC AHEAD

MOV A, M

AHEAD: STA 2503

HLT

* ORG 2501
* DB 98H,87H

**Program to find smaller of two numbers:**

# ORG 2000H

LXI H,2501

MOV A, M

INX H

CMP M

JC AHEAD

MOV A, M

AHEAD:

STA 2503

HLT

* ORG 2501
* DB B9H,8AH

**Program to find larger number in a data array:**

# ORG 2000H

LXI H,2500

MOV C, M

INX H

MOV A, M

DCR C

LOOP: CMP M

JNC AHEAD

MOV A, M

AHEAD:

JNZ LOOP

STA 2609

* ORG 2500
* DB 03,98,75,99

**Program to find smaller number in a data array:**

# ORG 2000H

LXI H,2500

MOV B, M

INX H

MOV A, M

DCR B

LOOP:

INX H

CMP M

JC AHEAD

MOV A, M

AHEAD: DCR B

JNZ LOOP

STA 2609

HLT

* ORG 2500
* DB 03,98,75,99

**To find the sum of series of 8-bit numbers:**

|  |  |
| --- | --- |
| # ORG 2000 |  |
| LXI | H,2500 |
| MOV | C, M |
| MVI | A,00 |
| LOOP: | INX H |
| ADD | M |
| DCR | C |

JNZ LOOP

STA 2600

HLT

* ORG 2500
* DB 04,16,20,39,12

**To find the sum of series of 8-bit numbers but sum is a 16-bit number:**

* ORG 2000 LXI H, 2500

MOV C, M MVI A, 00

MOV B, A Loop: INX H ADD M

JNC AHEAD INR B AHEAD: DCR C JNZ Loop STA, 2600

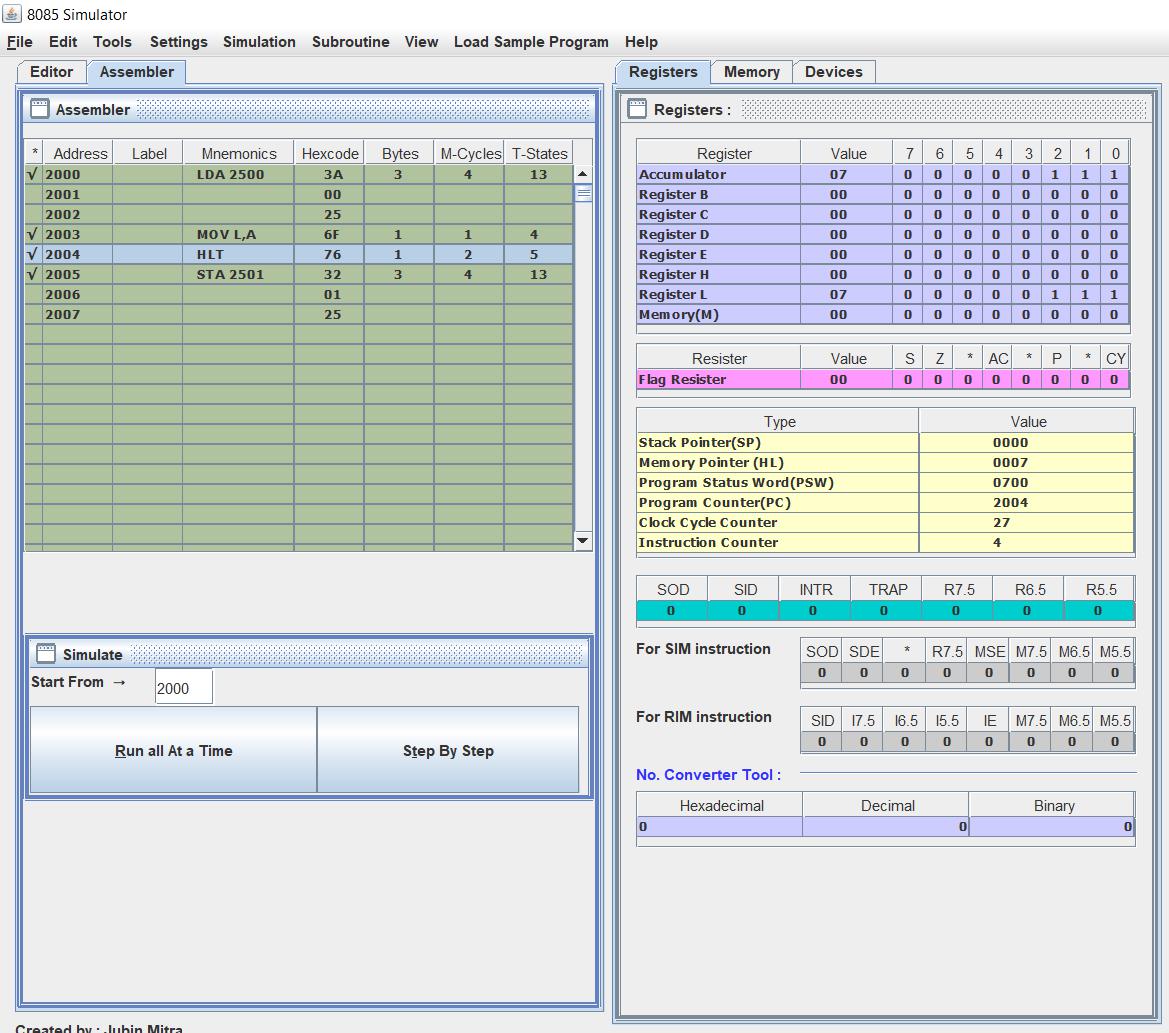
MOV A, B STA 2601 HLT

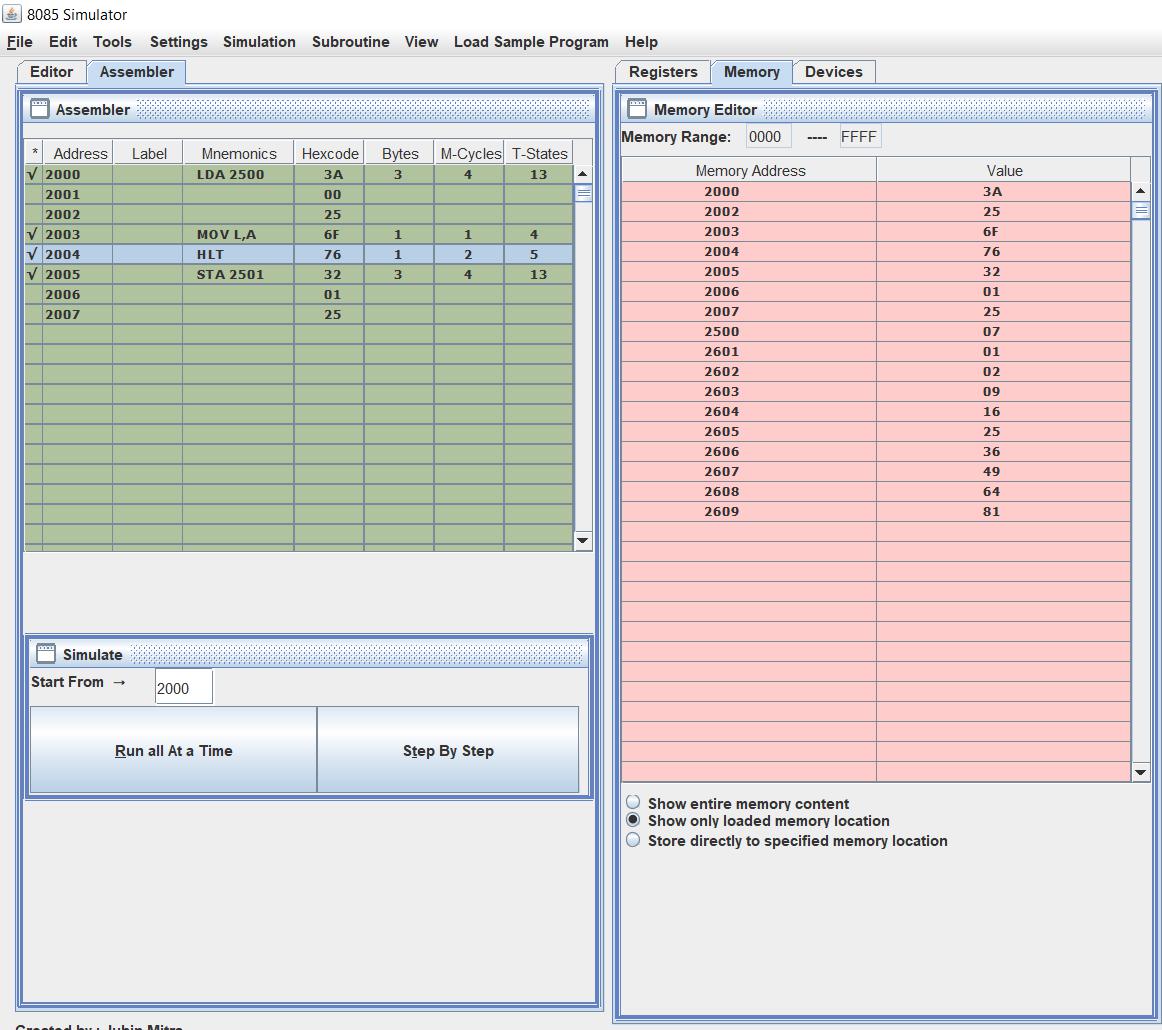
* ORG 2500
* DB 04,16,20,39,12

**Results:**

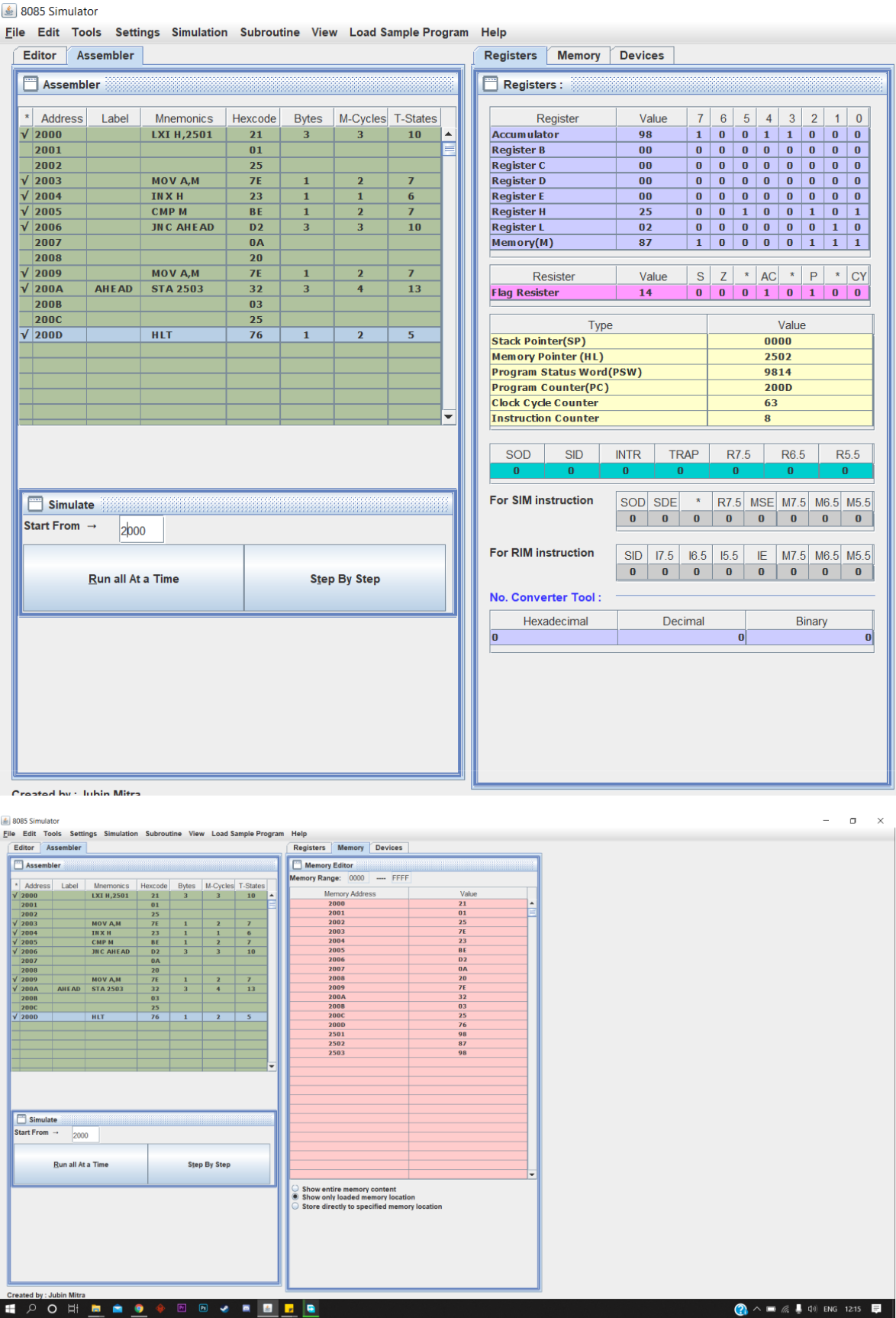
**To find square from lookup table.**

Output:

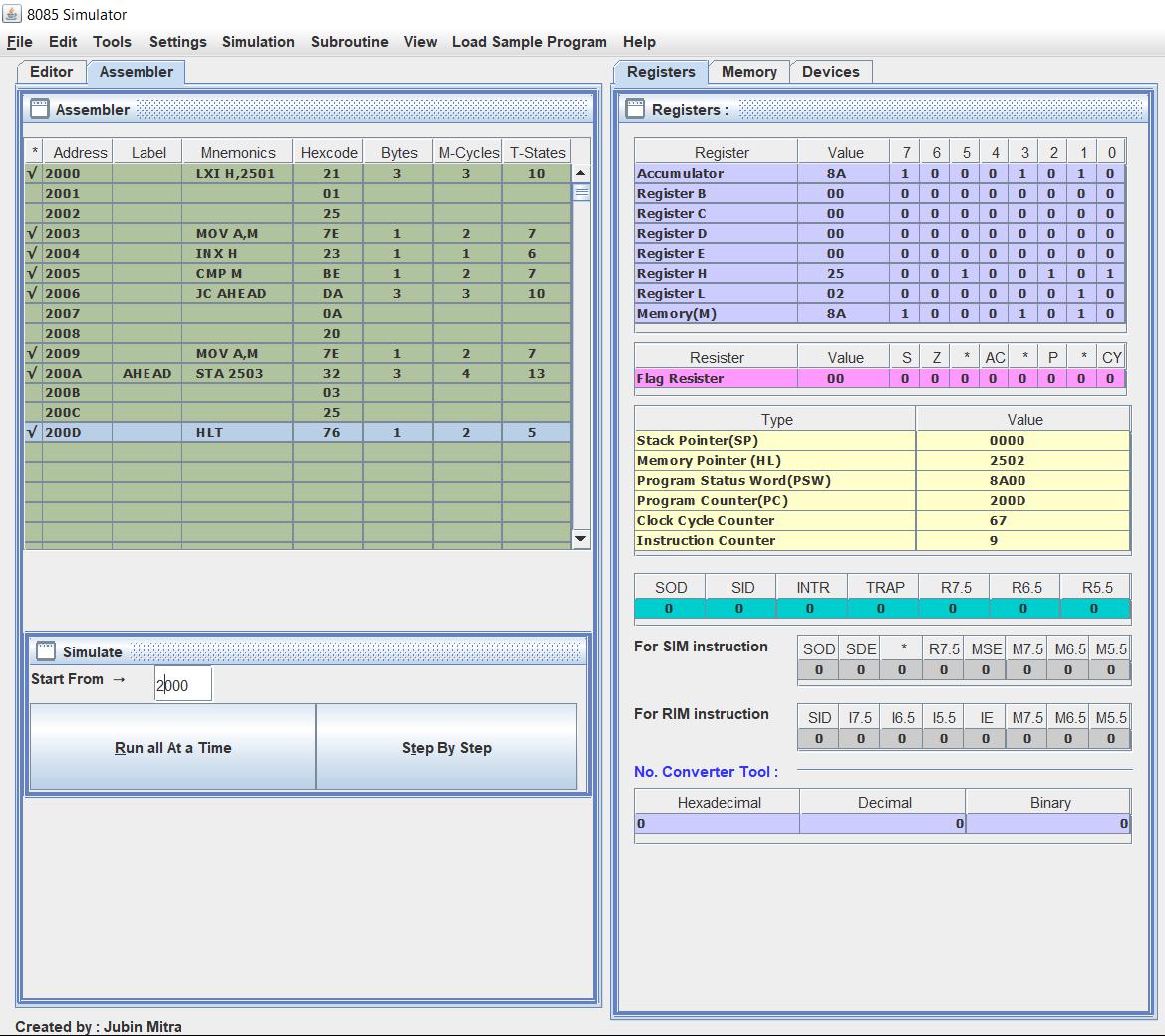


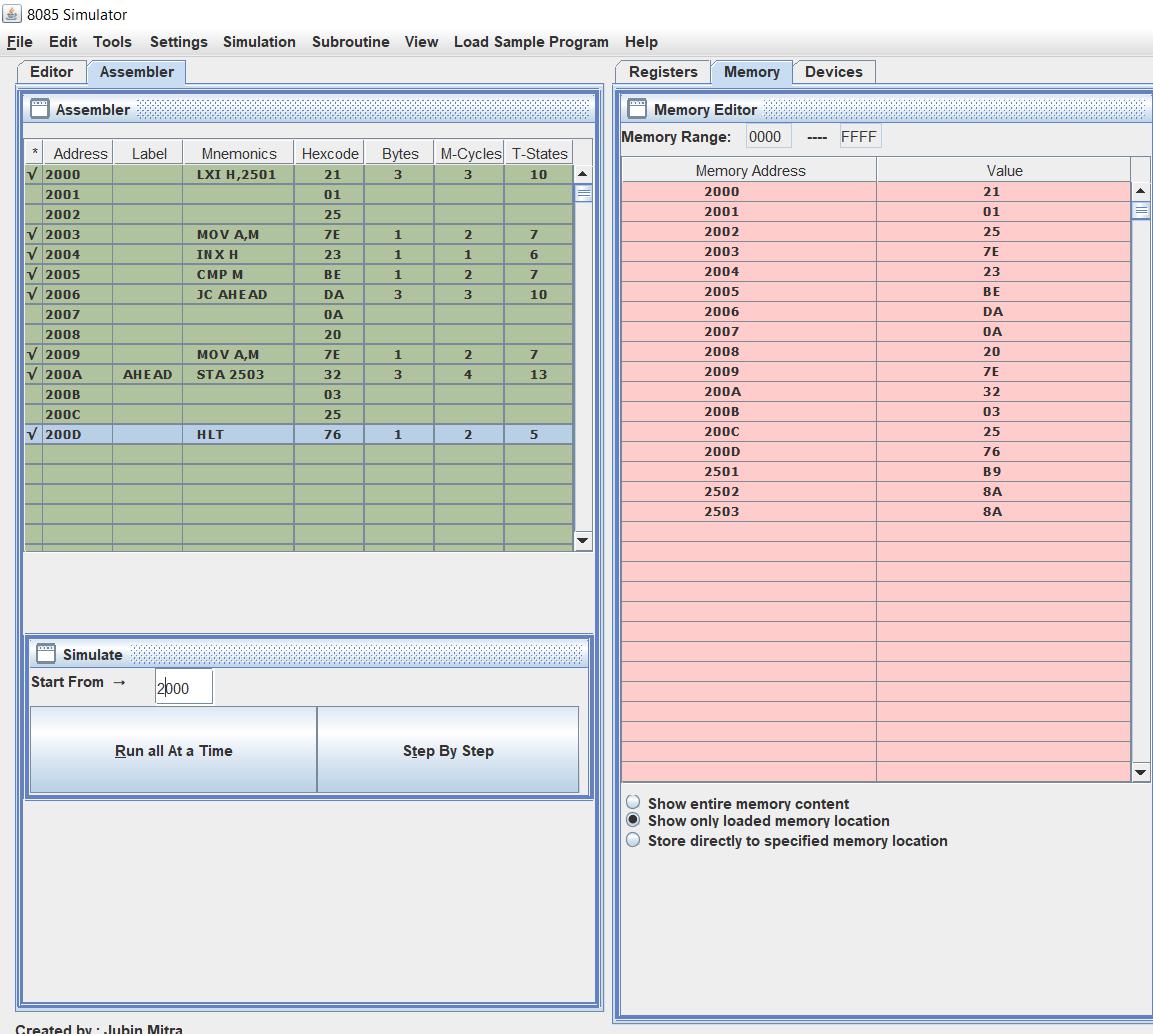


**To find larger of two numbers.**

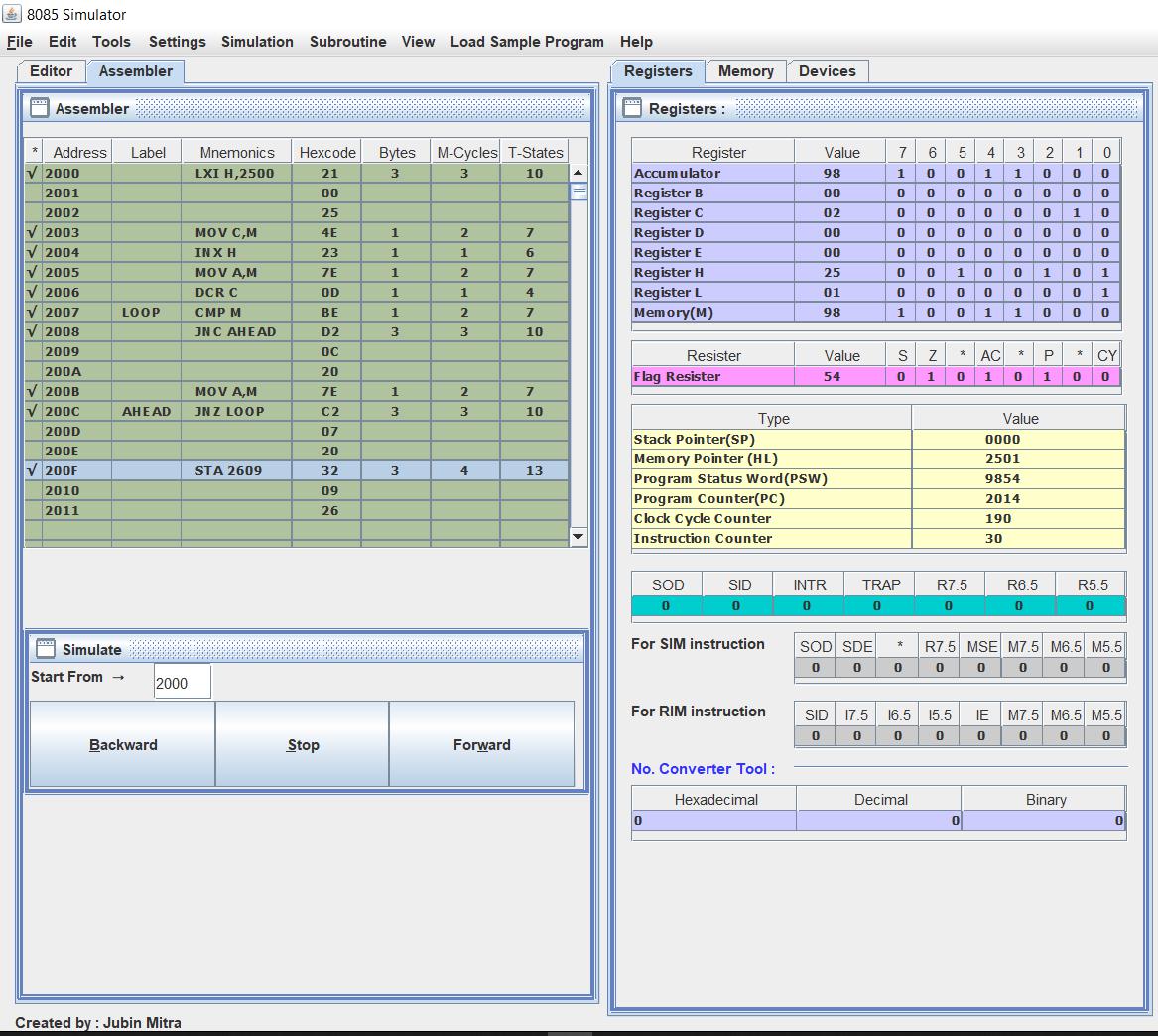


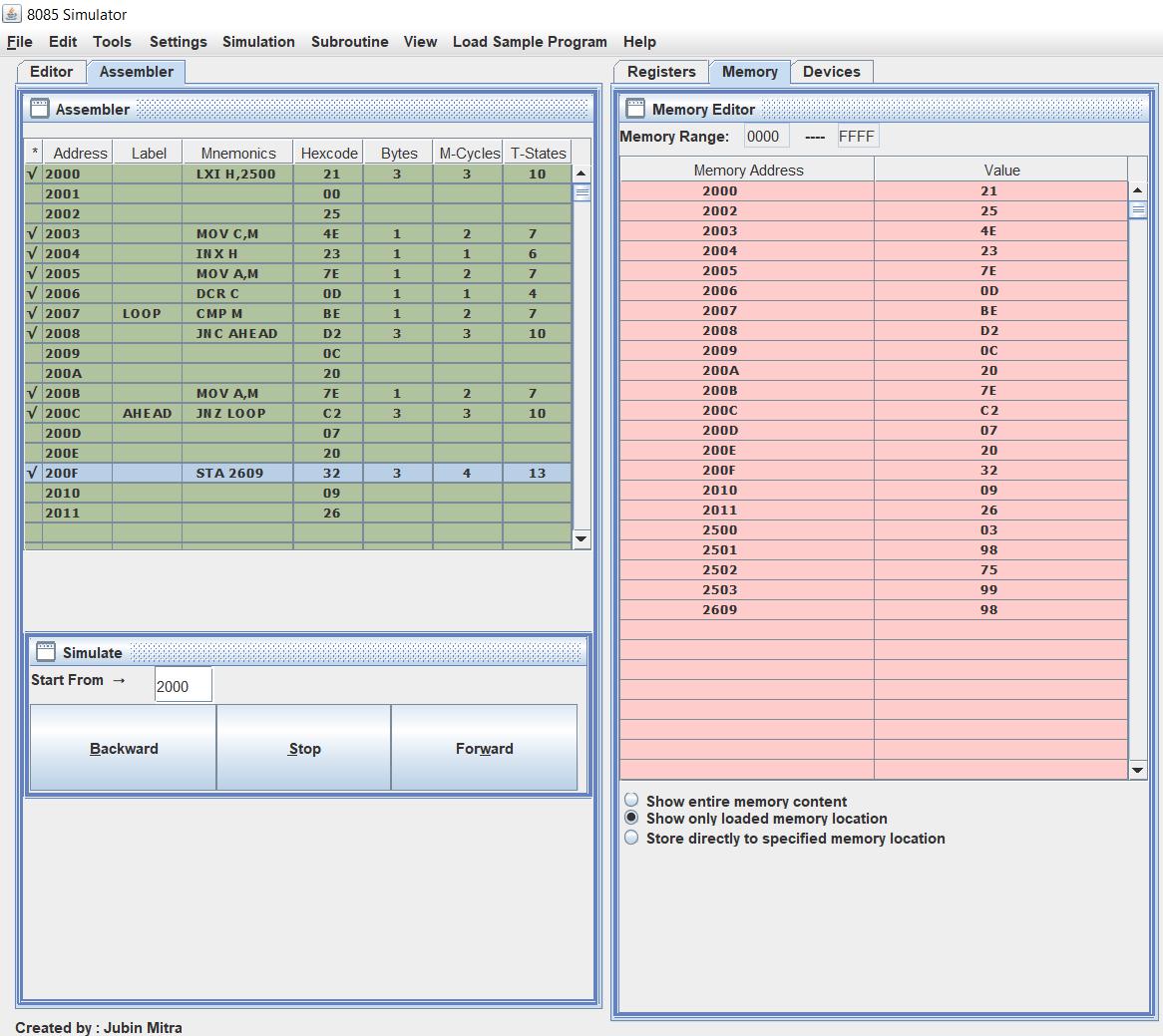
**To find smaller of two numbers.**



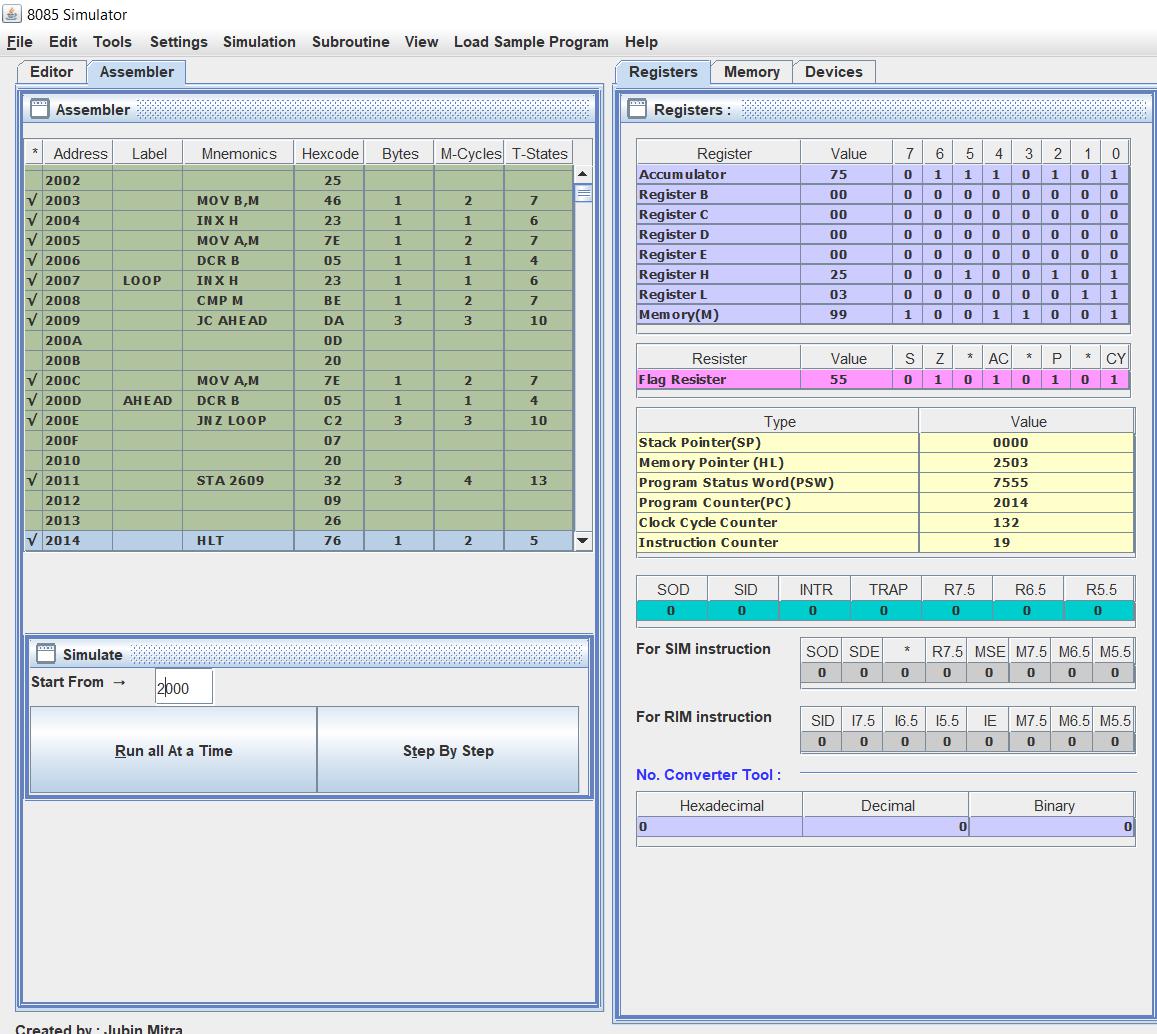


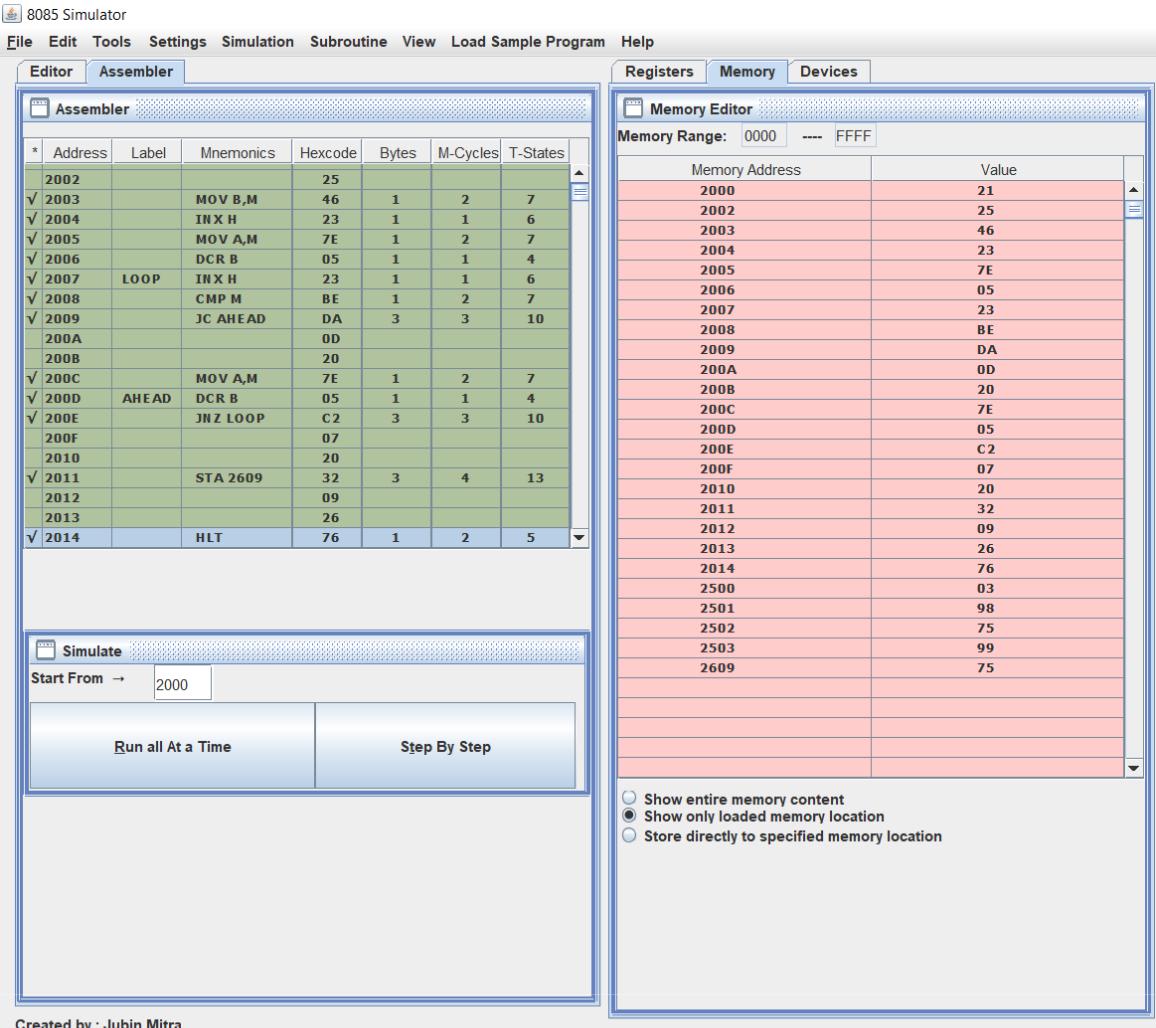
**To find larger number in a data array.**



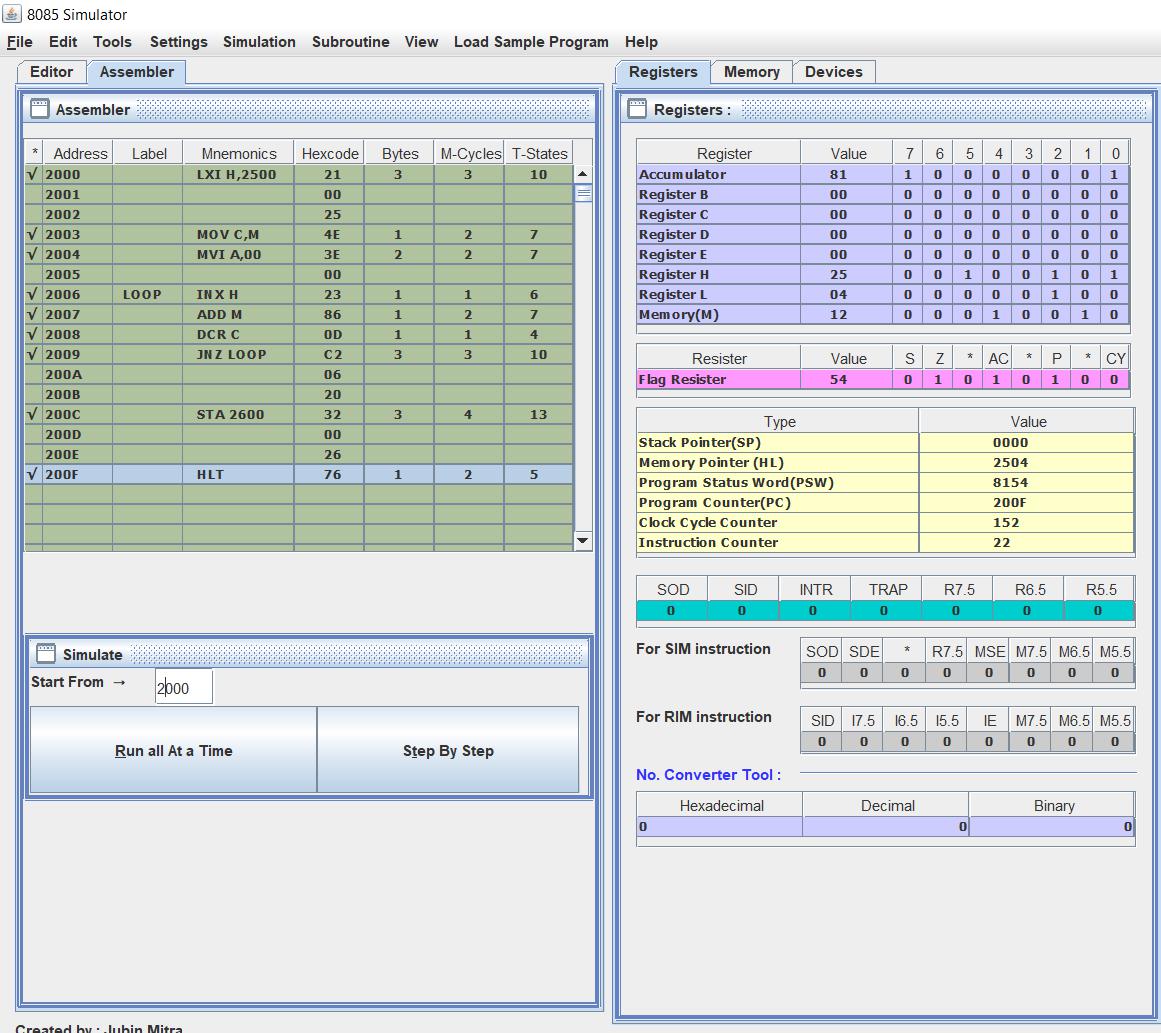


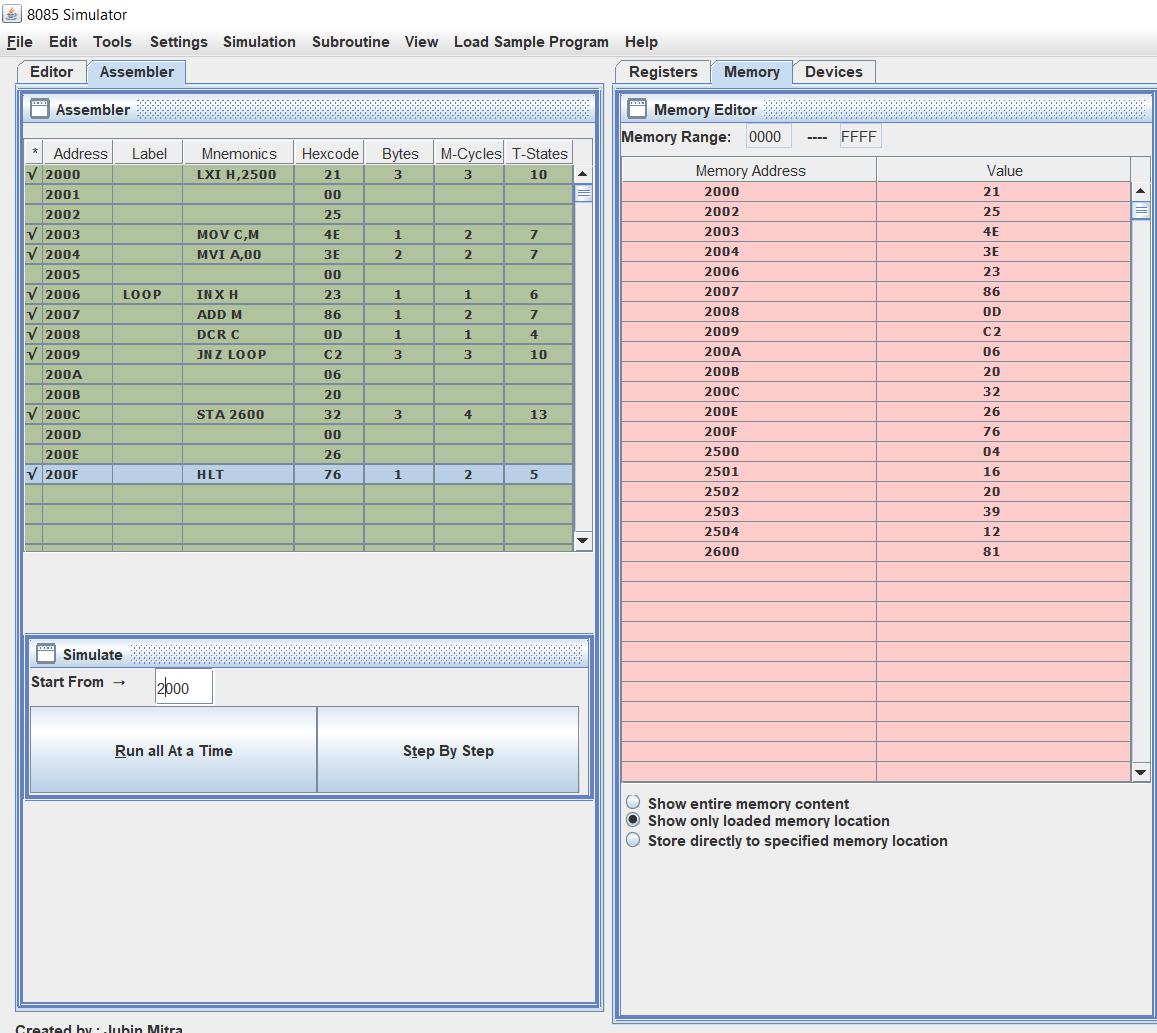
**To find smaller number in a data array.**



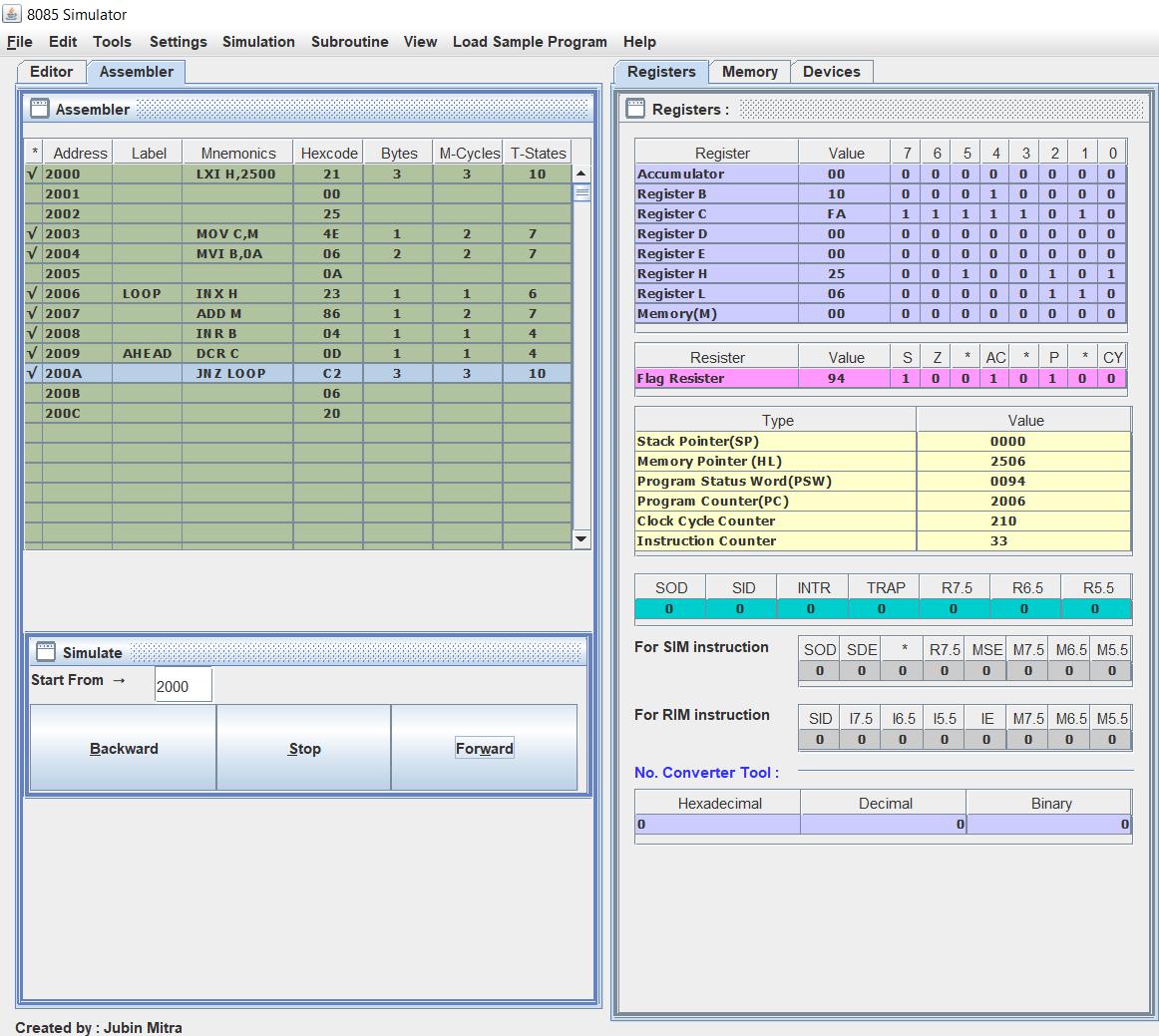


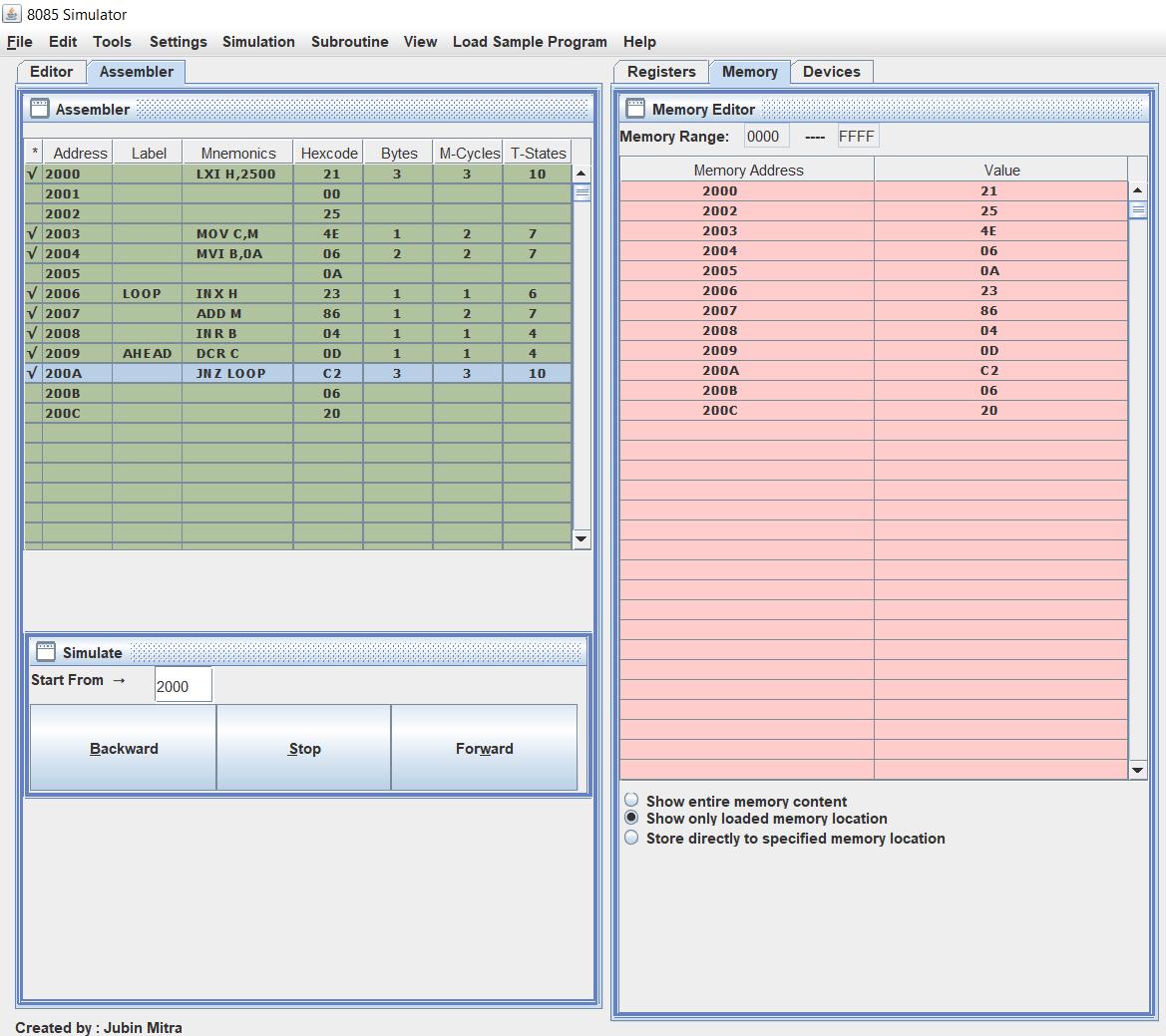
**To find the sum of series of 8-bit numbers.**





**To find the sum of series of 8-bit numbers but sum is a 16-bit number.**





**Sources of Error:**

1. Select op-codes carefully whenever you doing programming.
2. Make space whenever you doing program on kit.
3. You should properly enter the program and exit through proper command.

**-------------------------------------------X----------------------------------------------**

**EXPERIMENT: 6**

Aim:

Interfacing of 8051 Micro-controller with a various LED’s.

Apparatus Required:

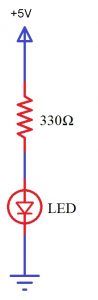
* AT89C51 (8051 Microcontroller)
* 8 LEDs
* 8 Resistors – 1KΩ
* Crystal oscillator – 11.0592MHz
* 2 Capacitors – 33pF
* 2 Resistors – 10KΩ
* 1 Capacitor – 10μF
* 1 Push Button
* 8051 Programmer
* 5V Power Supply.

Theory:

1. A Led working:

Light Emitting Diodes are the semi conductor light sources. Commonly used LEDs will have a cut-off voltage of 1.7V and current of 10mA. When an LED is applied with its required voltage and current it glows with full intensity. The Light Emitting Diode is similar to the normal PN diode but it emits energy in the form of light. The colour of light depends on the band gap of the semiconductor. The following figure shows “how an LED glows?” Thus, LED is connected to the AT89C51 microcontroller with the help of a current limiting resistor. The value of this resistor is calculated using the following formula.

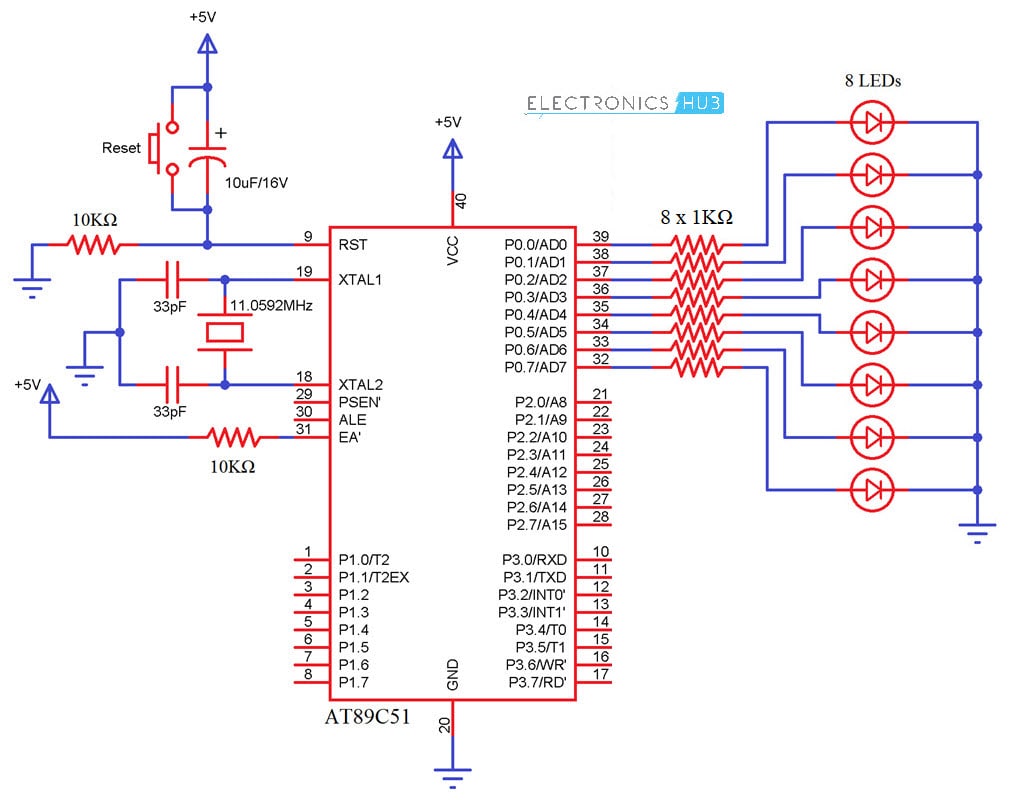
R= (V-1.7)/10mA, where V is the input voltage.

Generally, microcontrollers output a maximum voltage of 5V. Thus, the value of resistor calculated for this is 330 Ohms. This resistor can be connected to either the cathode or the anode of the LED. 

### 2. Principle behind Interfacing LED with 8051

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

**Circuit Diagram**

[](https://www.electronicshub.org/wp-content/uploads/2018/04/Interfacing-LED-with-8051-Microcontroller-Circuit-Diagram.jpg)

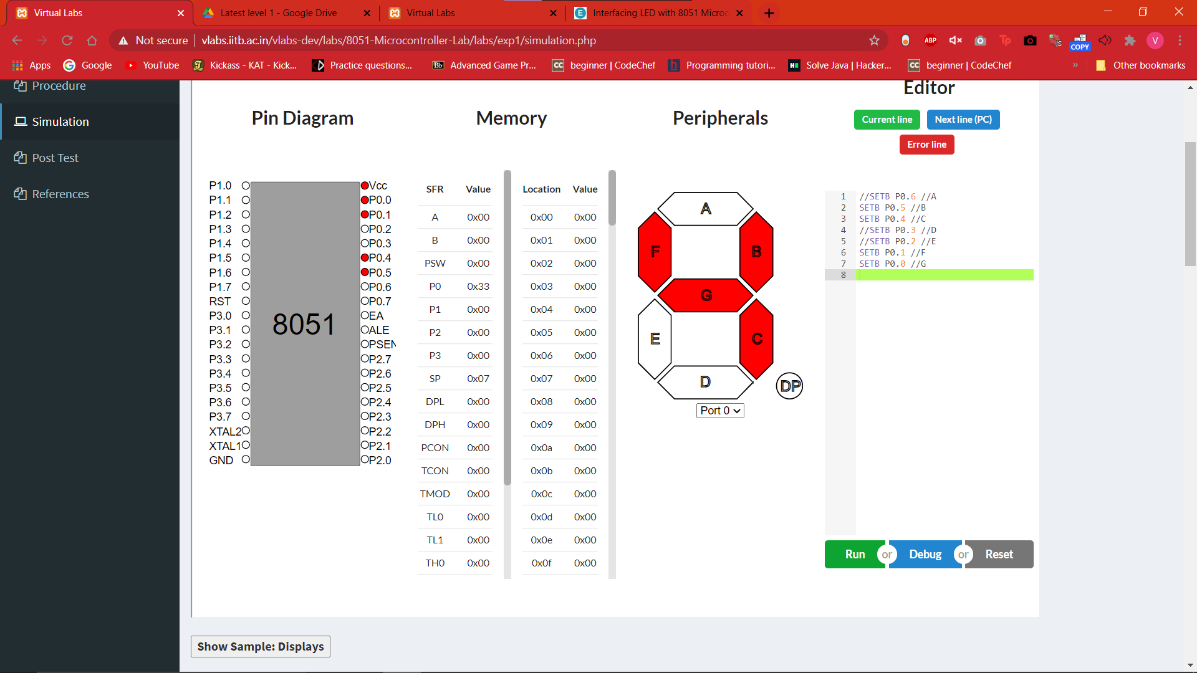
Procedure:

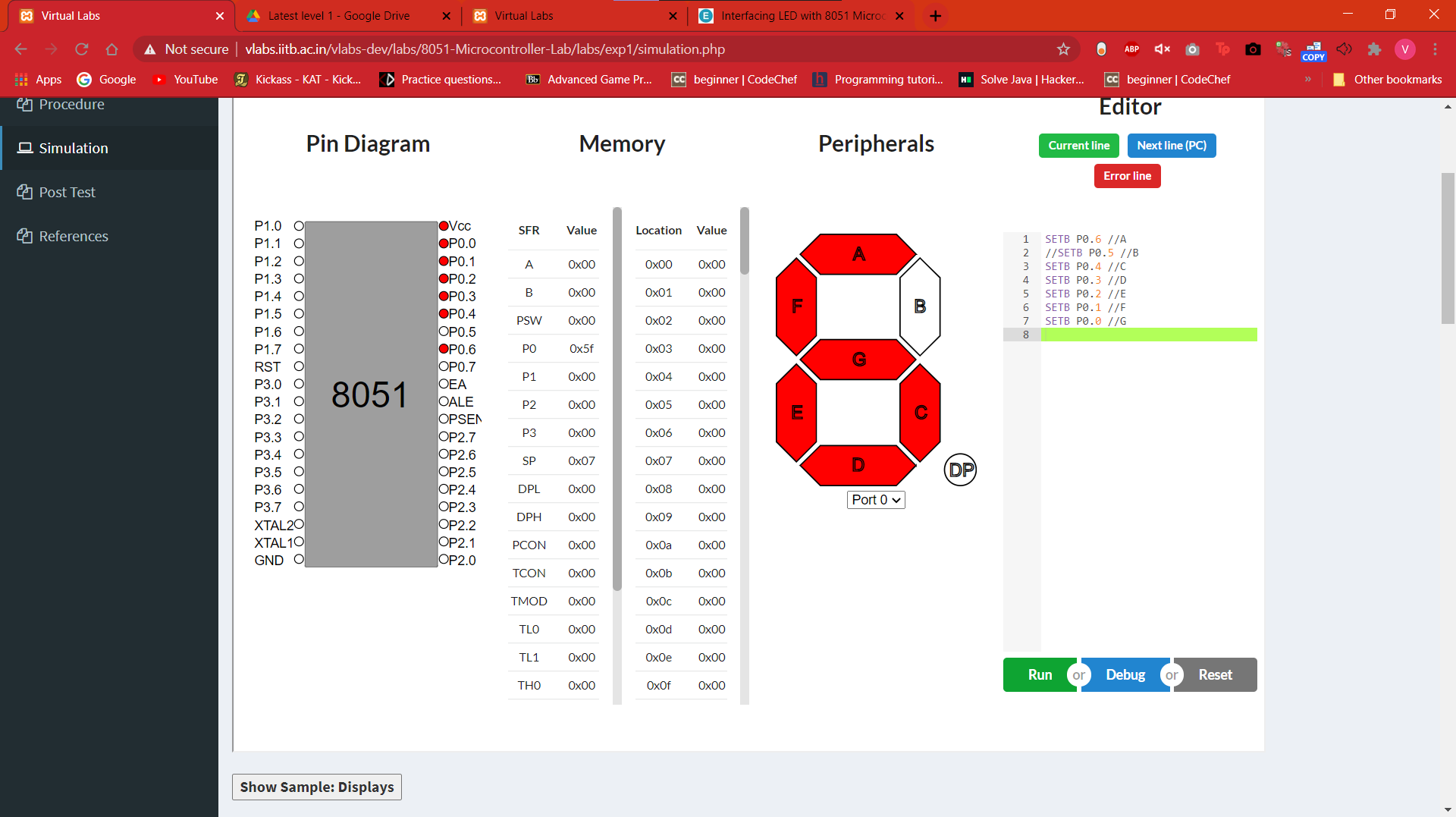
* Go to the simulator section to perform and experiment.
* Read all the instructions.
* Copy the below one of the codes to enlighten a LED in the program section and generate an output.
* Use debug for a line by line verification and use Run to see an Output.
* Use the below written code or make a new yourself.

|  |  |  |
| --- | --- | --- |
| Serial number: | Digit | Code |
| 1 | 0 | SETB P0.6  SETB P0.5  SETB P0.3  SETB P0.4  SETB P0.2  SETB P0.1 |
| 2 | 1 | //SETB P0.6 //A  SETB P0.5 //B  SETB P0.4 //C  //SETB P0.3 //D  //SETB P0.2 //E  //SETB P0.1 //F  //SETB P0.0 //G |
| 3 | 2 | ///For Zero  SETB P0.6 //A  SETB P0.5 //B  //SETB P0.4 //C  SETB P0.3 //D  SETB P0.2 //E  //SETB P0.1 //F  SETB P0.0 //G |
| 4 | 3 | SETB P0.6 //A  SETB P0.5 //B  SETB P0.4 //C  SETB P0.3 //D  //SETB P0.2 //E  //SETB P0.1 //F  SETB P0.0 //G |
| 5 | 4 | //SETB P0.6 //A  SETB P0.5 //B  SETB P0.4 //C  //SETB P0.3 //D  //SETB P0.2 //E  SETB P0.1 //F  SETB P0.0 //G |
| 6 | 5 | SETB P0.6 //A  //SETB P0.5 //B  SETB P0.4 //C  SETB P0.3 //D  //SETB P0.2 //E  SETB P0.1 //F  SETB P0.0 //G |
| 7 | 6 | SETB P0.6 //A  //SETB P0.5 //B  SETB P0.4 //C  SETB P0.3 //D  SETB P0.2 //E  SETB P0.1 //F  SETB P0.0 //G |
| 8 | 7 | SETB P0.6 //A  SETB P0.5 //B  SETB P0.4 //C  //SETB P0.3 //D  //SETB P0.2 //E  //SETB P0.1 //F  //SETB P0.0 //G |
| 9 | 8 | SETB P0.6 //A  SETB P0.5 //B  SETB P0.4 //C  SETB P0.3 //D  SETB P0.2 //E  SETB P0.1 //F  SETB P0.0 //G |
| 10 | 9 | SETB P0.6 //A  SETB P0.5 //B  SETB P0.4 //C  SETB P0.3 //D  //SETB P0.2 //E  SETB P0.1 //F  SETB P0.0 //G |

Common Code:  
MOV P0,#7Eh     //to display 0  
MOV P0,#30h     //to display 1  
MOV P0,#6Dh     //to display 2  
MOV P0,#79h     //to display 3  
MOV P0,#33h     //to display 4  
MOV P0,#5Bh     //to display 5  
MOV P0,#5Fh     //to display 6  
MOV P0,#70h     //to display 7  
MOV P0,#7Fh     //to display 8  
MOV P0,#7Bh     //to display 9  
MOV P0,#77h     //to display A  
MOV P0,#4Eh     //to display C  
MOV P0,#4Fh     //to display E  
MOV P0,#47h     //to display F

Output:





**Sources of Error:**

1. Select op-codes carefully whenever you doing programming.
2. Make space whenever you doing program on kit.
3. You should properly enter the program and exit through proper command.

**EXPERIMENT: 7**

Aim:

Display the integers 2,9 and Character A using SETB and CLR instructions by interfacing a Micro-Controller 8051.

Apparatus Required:

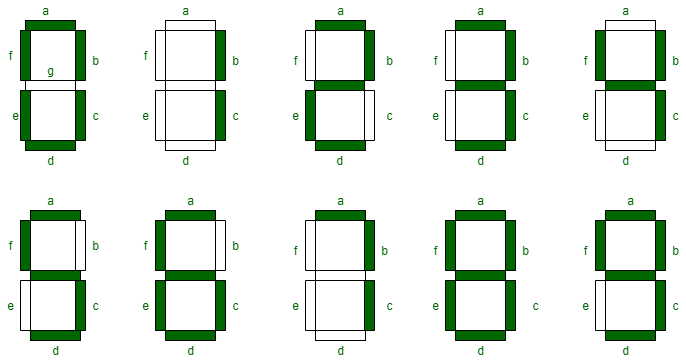
* Virtual lab simulator.

Theory:

Light Emitting Diode (LED) is the most widely used semiconductor which emits either visible light or invisible infrared light when forward biased. Remote controls generate invisible light. A Light emitting diode (LED) is an optical electrical energy into light energy when voltage is applied.

**Seven Segment Displays:**  
Seven segment displays are the output display device that provide a way to display information in the form of image or text or decimal numbers which is an alternative to the more complex dot matrix displays. It is widely used in digital clocks, basic calculators, electronic meters, and other electronic devices that display numerical information. It consists of seven segments of light emitting diodes (LEDs) which is assembled like numerical 8.

Diagram:

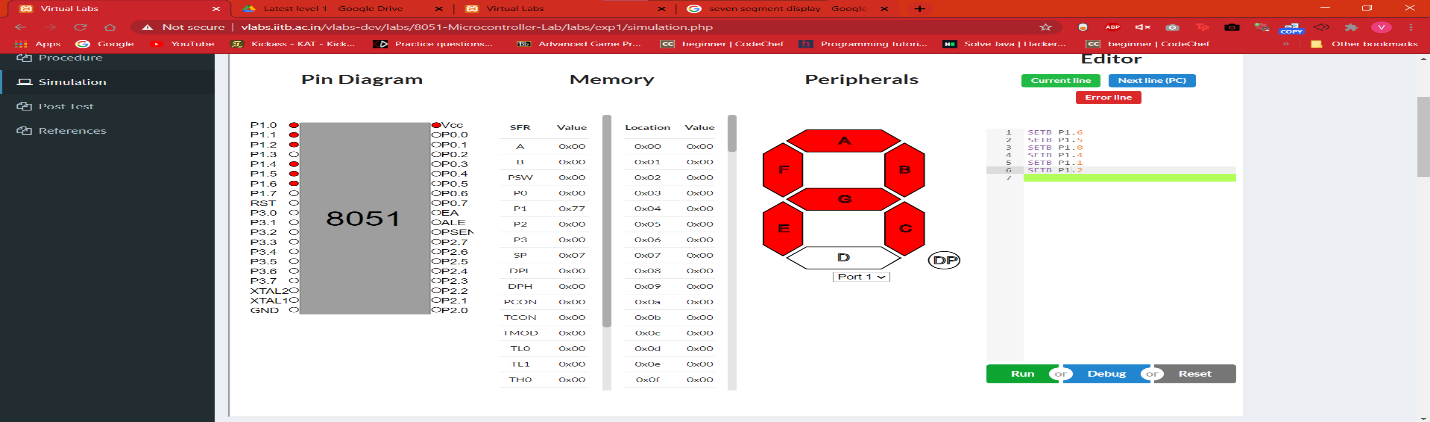
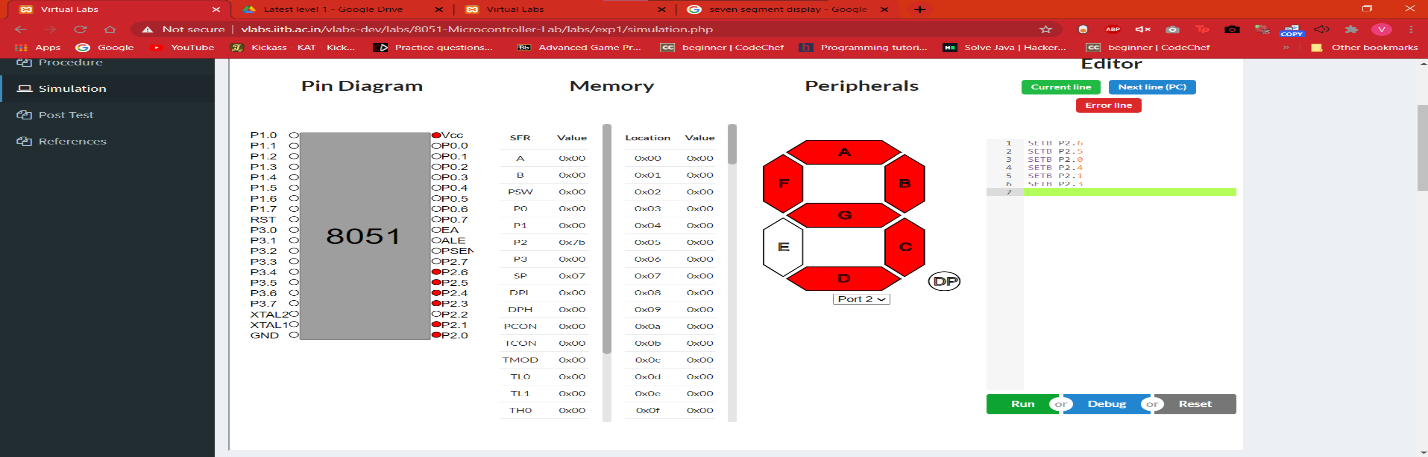
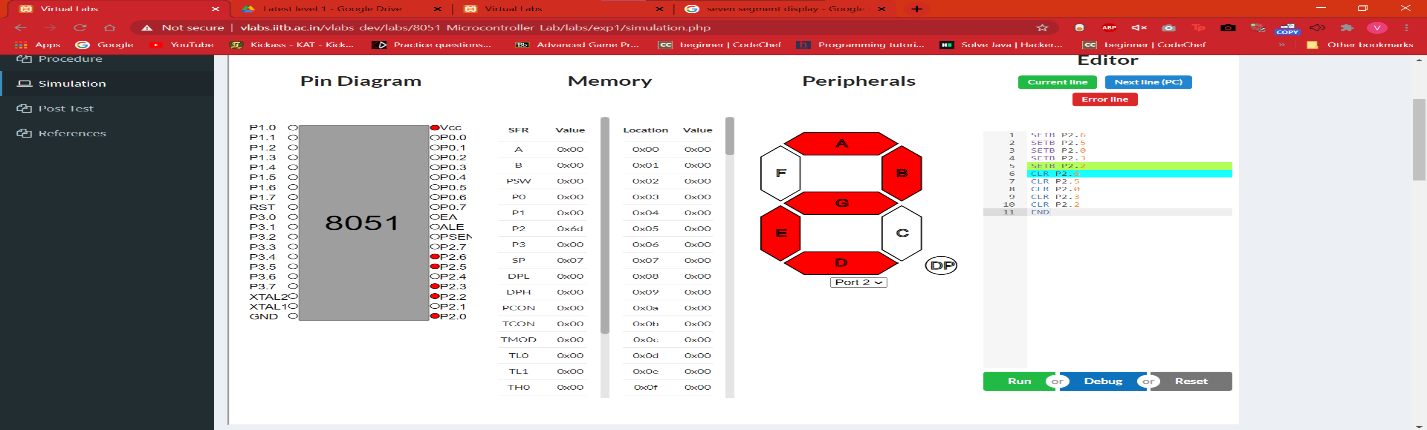


Procedure:

* Go to the simulator
* Go to the text editor.
* Paste the codes below in the text editor.

|  |  |  |
| --- | --- | --- |
| Serial no. | Digit | Code |
| 1. | 2 | **SETB P2.6**  **SETB P2.5**  **SETB P2.0**  **SETB P2.3**  **SETB P2.2**  **CLR P2.6**  **CLR P2.5**  **CLR P2.0**  **CLR P2.3**  **CLR P2.2** |
| 2. | 9 | **SETB P2.6**  **SETB P2.5**  **SETB P2.0**  **SETB P2.4**  **SETB P2.1**  **SETB P2.3** |
| 3. | A | **SETB P1.6**  **SETB P1.5**  **SETB P1.0**  **SETB P1.4**  **SETB P1.1**  **SETB P1.2** |

Output

****

**Sources of Error:**

1. Select op-codes carefully whenever you doing programming.
2. Make space whenever you doing program on kit.
3. You should properly enter the program and exit through proper command.

**EXPERIMENT: 8**

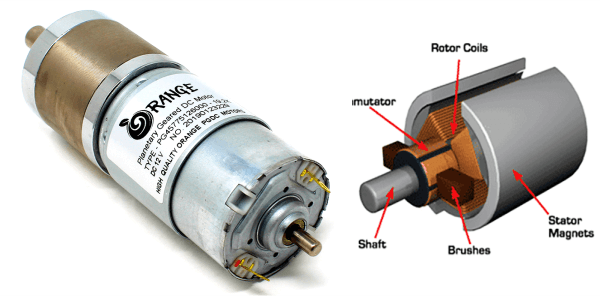
Aim:

To interface a DC motor with a microcontroller using virtual lab simulator.

Apparatus Required:

Virtual Simulator.

Theory:



The[DC motor](https://robu.in/product-category/motors-drivers-actuators/dc-motor/) is the motor which converts the direct current into the mechanical work. It works on the principle of Lorentz Law, which states that “the current carrying conductor placed in a magnetic and electric field experience a force”. And that force is the [Lorentz force](https://en.wikipedia.org/wiki/Lorentz_force).

## Types of DC Motor

There are 4 major [types of DC motor](https://robu.in/types-of-dc-motor/) and they are,

* Series DC Motor
* Permanent Magnet DC Motor
* Shunt/Parallel DC Motor
* Compound DC Motors

MUST READ ON DC MOTORS:

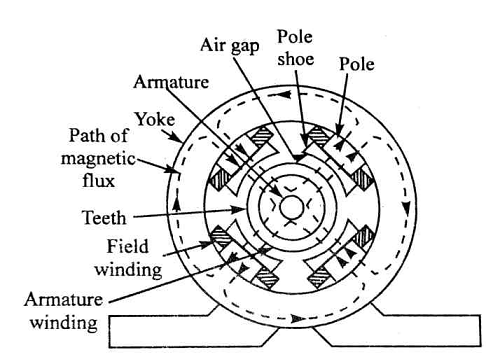
* [Different Types of DC Motor](https://robu.in/types-of-dc-motor/)
* [Difference Between AC and DC Motor](https://robu.in/difference-between-ac-and-dc-motor/)
* [Speed Control of DC Motor](https://robu.in/speed-control-of-dc-motor)
* [Applications of Dc motor](https://robu.in/dc-motor-applications/)

## Construction of DC Motor

Before understanding the working of DC motor first, we have to know about their construction. There are two main parts of the[DC motor](https://robu.in/product-category/motors-drivers-actuators/dc-motor/).

* Armature
* Stator

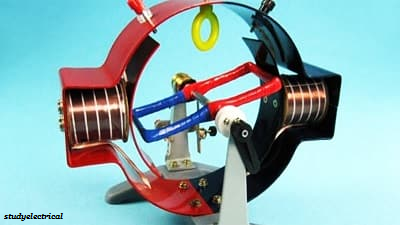
The rotating part is the armature and the Stator is their stationary part. The armature coil is connected to the DC supply.

[](https://aws.robu.in/wp-content/uploads/2020/04/Construction-of-Dc-motot.png)

The [armature](https://en.wikipedia.org/wiki/Armature_(electrical)) coil consists the commutators and brushes. The commutator converts the AC induces in the armature into DC and brushes transfer the current from rotating part of the motor to the stationary external load. The armature places between the north and south pole of the permanent or electromagnet.

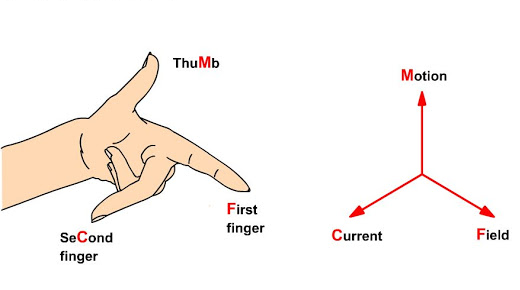
## Working Principle of DC Motor

A DC motor is an electrical machine which converts electrical energy into mechanical energy. The basic working principle of the DC motor is that whenever a current carrying conductor places in the magnetic field, it experiences a mechanical force.

[](https://aws.robu.in/wp-content/uploads/2020/04/Working-of-DC-motot.png)

Fleming’s left-hand rule and its magnitude decide the direction of this force.

### Fleming’s Left Hand Rule:

If we stretch the first finger, second finger and thumb of our left hand to be perpendicular to each other, and first finger represents the direction of the magnetic field, the second finger represents the direction of the current, then the thumb represents the direction of the force experienced by the current carrying conductor.[](https://aws.robu.in/wp-content/uploads/2020/04/Flemings-left-hand-rule.png)

F = BIL Newtons

Where,

B = magnetic flux density,

I = current and

L = length of the conductor within the magnetic field.

When armature winding is connected to a DC supply, an electric current sets up in the winding.  Permanent magnets or field winding (electromagnetism) provides the magnetic field. In this case, current carrying armature conductors experience a force due to the magnetic field, according to the principle stated above.

The [Commutator](https://www.google.com/search?q=commutator&rlz=1C1AVFA_enIN801IN801&oq=commutator&aqs=chrome..69i57j0l7.2814j0j7&sourceid=chrome&ie=UTF-8) is made segmented to achieve unidirectional torque. Otherwise, the direction of force would have reversed every time when the direction of movement of the conductor is reversed in the magnetic field. This is how a DC motor works!

Procedure:

* Go to the simulator
* Use the below written code to rotate the motor anti-clockwise.

**SETB P1.0**

**CLR P1.0**

**CLR P1.1**

**CLR P1.2**

**CLR P1.3**

**SETB P2.7**

**SETB P1.0**

**CLR P1.1**

**CLR P1.2**

**SETB P1.3**

**CLR P1.0**

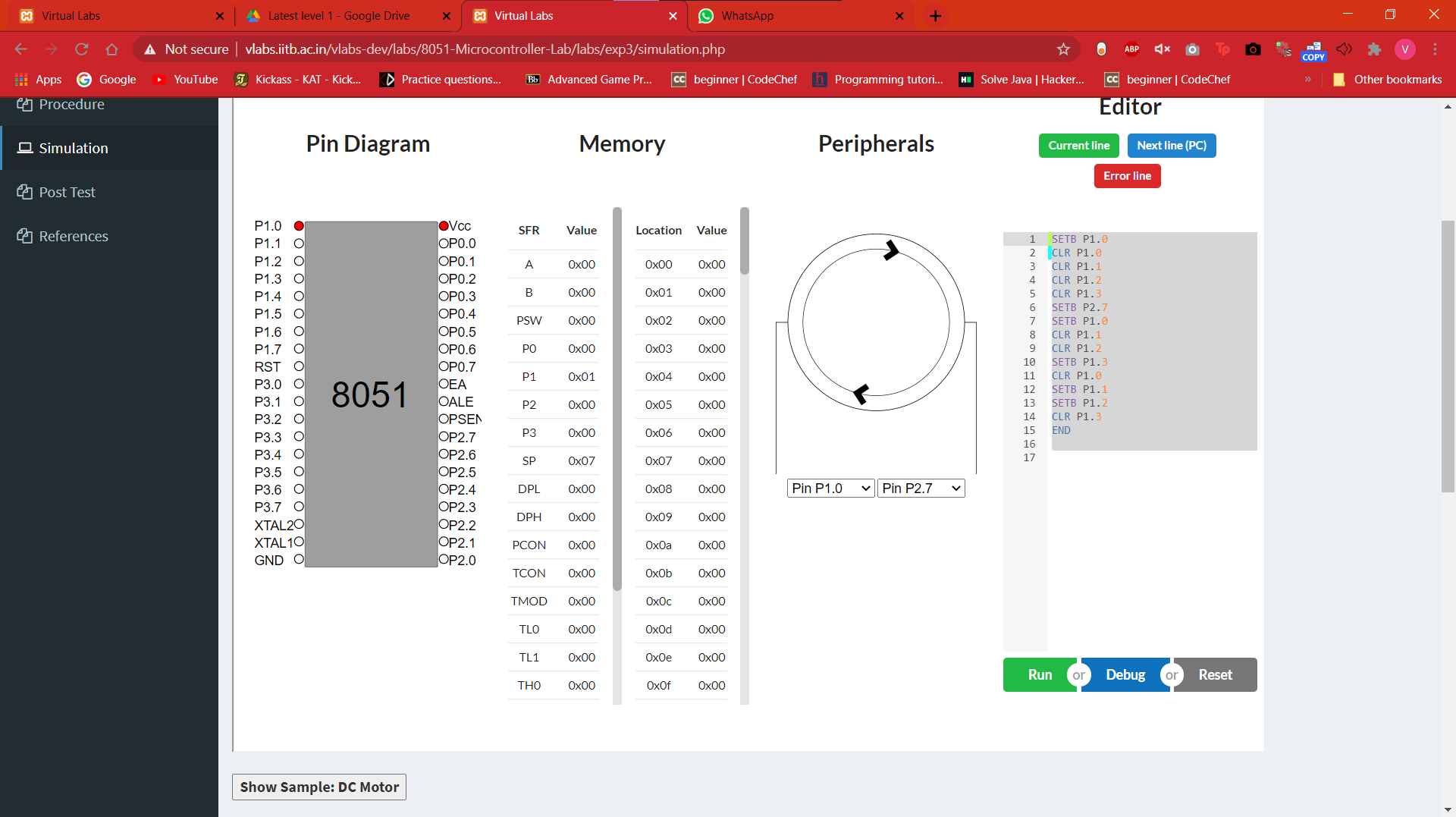
**SETB P1.1**

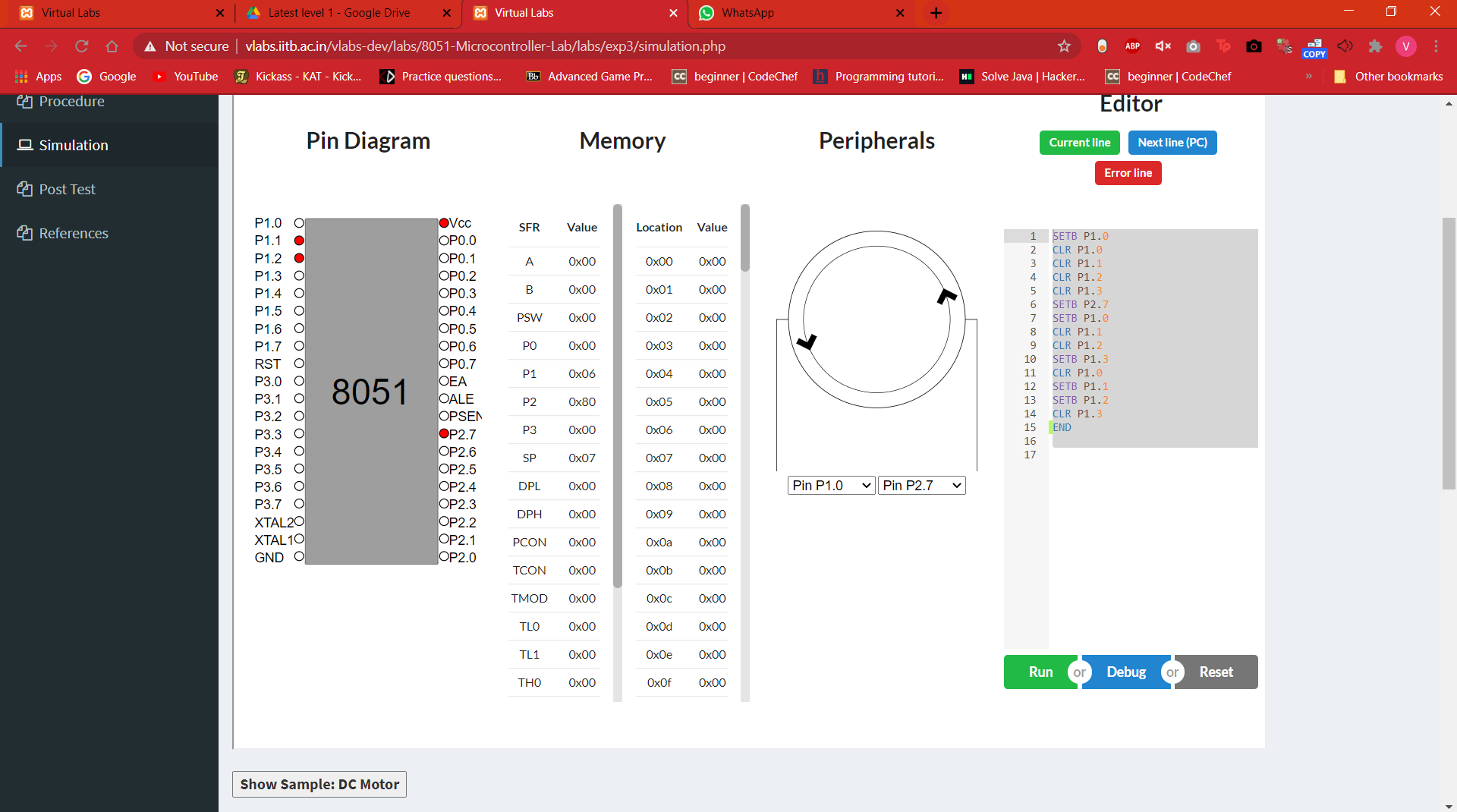
**SETB P1.2**

**CLR P1.3**

**END**

Output





**Sources of Error:**

* Select op-codes carefully whenever you doing programming.
* Make space whenever you doing program on kit.
* You should properly enter the program and exit through proper command.