Computer Organization and Architecture

(ETCS-204)

**Faculty name**: Ms. Neetu Garg

**Student name:** ARIHANT JAIN

**Roll No.:** 08914802719

**Group:** 4C5



Maharaja Agrasen Institute of Technology, PSP area,

Sector – 22, Rohini, New Delhi – 110085

INDEX OF THE CONTENTS

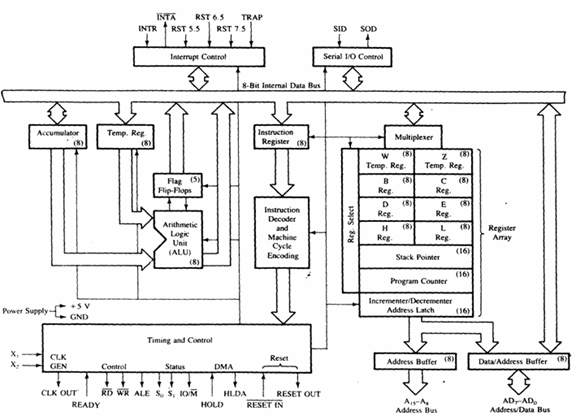
|  |  |  |  |
| --- | --- | --- | --- |
| S.No. | DATE | EXPERIMENTS | REMARKS |
| **1.** | 22/03/2021 | To draw and explain   1. Block diagram and pin diagram of 8085. 2. Instruction set of 8085. |  |
| **2.** | 05/04/2021 | Write the working of GNUsim8085. |  |
| **3.** | 12/04/2021 | Write a program to perform:   1. Addition of two 8-bit numbers without carry. 2. Addition of two 8-bit numbers with carry. |  |
| **4.** | 19/04/2021 | Write a program to perform:   1. Subtraction of two 8-bit numbers without borrows. 2. Subtraction of two 8-bit numbers with borrows. |  |
| **5.** | 26/04/2021 | Write a program to find 1‘s complement of an 8-bit number. |  |
| **6.** | 03/05/2021 | Write a program to find 2‘s complement of an 8-bit number. |  |
| **7.** | 10/05/2021 | Write an assembly language code to store numbers in reverse order in memory location. |  |
| **8.** | 17/05/2021 | Write an assembly language in GNUsim8085 to add two 16-bit numbers without using lxi instruction |  |
| **9.** | 24/05/2021 | Write a program to perform Multiplication of two 8-bit numbers. |  |
| **10.** | 31/05/2021 | Write a program to perform Division of two 8-bit numbers. |  |

Experiment-1

**Aim:** To draw and explain

1. Block diagram and pin diagram of 8085.
2. Instruction set of 8085*.*

**Theory:**



**List of registers used in 8085 to perform various operations:**

**Accumulator**:-It is a 8-bit register which is used to perform arithmetical and logical operation. It stores the output of any operation. It also works as registers for i/o accesses.

It can be one of the operand in the instruction.  
  
**Temporary Register:**-It is a 8-bit register which is used to hold the data on which the accumulator is computing operation. It is also called as operand register because it provides operands to ALU.  
  
**Registers**:-These are general purposes registers. Microprocessor consists 6 general purpose registers of 8-bit each named as B,C,D,E,H and L. Generally theses registers are not used for storing the data permanently. It carries the 8-bits data. These are used only during the execution of the instructions.  
These registers can also be used to carry the 16 bits data by making the pair of 2 registers. The valid register pairs available are BC,DE,HL. We cannot use other pairs except BC,DE and HL. These registers are programmed by user.

**Flag Registers:-**It consists of 5 flip flop which changes its status according to the result stored in an accumulator. It is also known as status registers. It is connected to the ALU.  
There are five flip-flops in the flag register are as follows:

**The bit position of the flip flop in flag register is:**

  
  
All of the three flip flop set and reset according to the stored result in the accumulator.  
1.**Sign**- If D7 of the result is 1 then sign flag is set otherwise reset. As we know that a number on the D7 always desides the sign of the number.  
🡪 if D7 is 1: the number is negative.  
🡪 if D7 is 0: the number is positive.

2.**Zeros(Z)**-If the result stored in an accumulator is zero then this flip flop is set as 1 otherwise it is reset and also if the result of any arithmetic or logical operation is zero its set as 1( The result of this operation can be in any registers).

3.**Auxiliary carry(AC)**-If any carry goes from D3 to D4 in the output then it is set otherwise it is reset.

4.**Parity(P)**-If the no of 1's is even in the output stored in the accumulator then it is set otherwise it is reset for the odd.

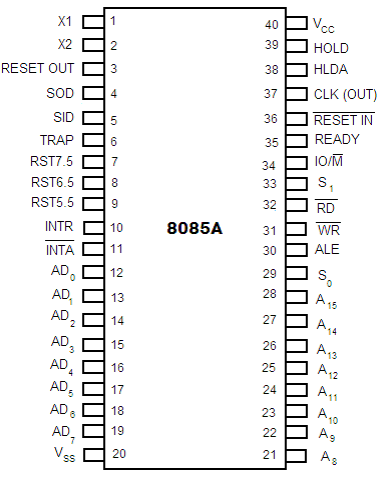
5.**Carry(C)**-If the result stored in an accumulator generates a carry in its final output then it is set otherwise it is reset.

6.**Instruction registers(IR)**:-It is a 8-bit register. When an instruction is fetched from memory then it is stored in this register.

**Description of other components of 8085:**

**Instruction Decoder: -** Instruction decoder identifies the instructions. It takes the information from instruction register and decodes the instruction to be performed.  
  
**Program Counter:**-It is a 16 bit register used as memory pointer. It stores the memory address of the next instruction to be executed. So we can say that this register is used to sequencing the program. Generally the memory has 16 bit addresses so that it has 16 bit memory.  
The program counter is set to 0000H.  
  
**Stack Pointer:**-It is also a 16 bit register used as memory pointer. It points to the memory location called stack. Generally stack is a reserved portion of memory where information can be stores or taken back together.  
  
**Timing and Control Unit:**-It provides timing and control signal to the microprocessor to perform the various operation. It has three control signals. It controls all external and internal circuits. It operates with reference to clock signal. It synchronizes all the data transfers.  
There are three control signal:  
1.ALE-Airthmetic Latch Enable, It provides control signal to synchronize the components of microprocessor.  
2.RD- This is active low used for reading operation.  
3.WR-This is active low used for writing operation.  
  
There are three status signal used in microprocessor S0, S1 and IO/M. It changes its status according the provided input to these pins.  
  
**Serial Input Output Control**-There are two pins in this unit. This unit is used for serial data communication.  
  
**Interrupt Unit**-There are 6 interrupt pins in this unit. Generally an external hardware is connected to these pins. These pins provide interrupt signal sent by external hardware to microprocessor and microprocessor sends acknowledgement for receiving the interrupt signal. Generally INTA is used for acknowledgement.

**PIN DIAGRAM OF 8085**



**Description:**

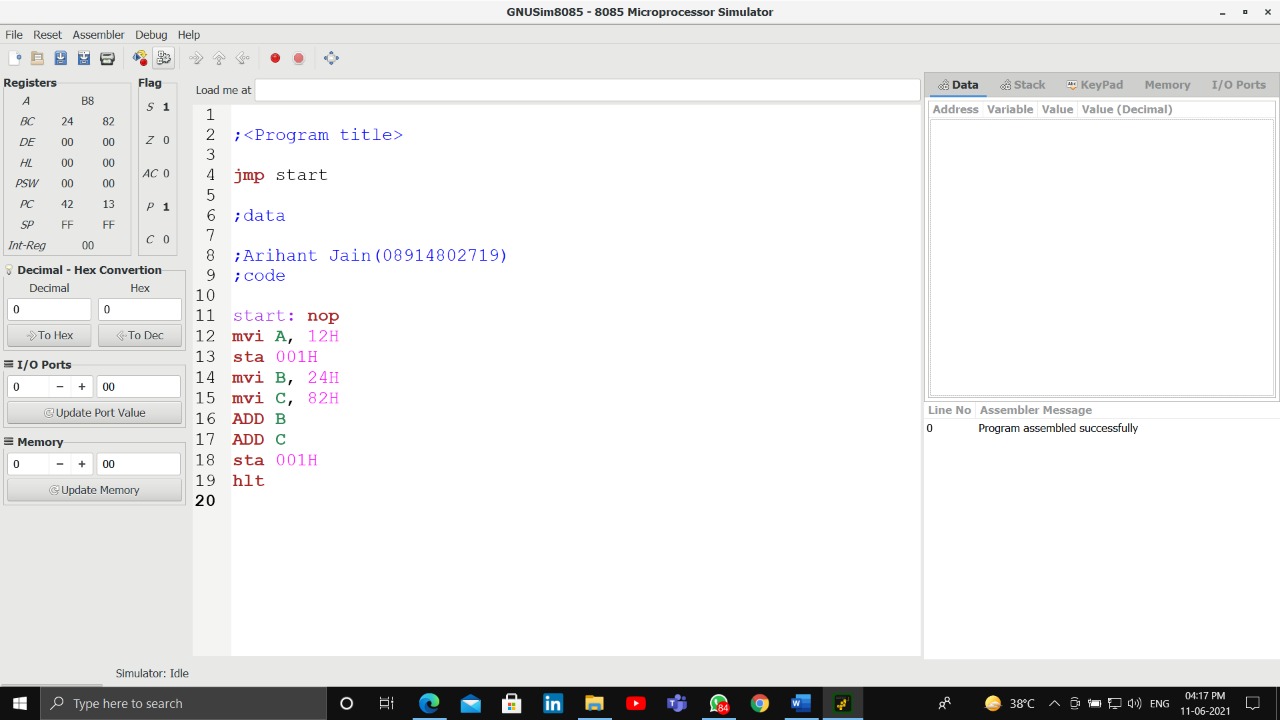
* Higher Order Address pins- A15– A8
* The address bus has 8 signal lines A8 – A15 which are unidirectional.
* Lower Order Address/ Data Pins- AD7-AD0
* These are time multiplexed pins and are de-multiplexed using the pin ALE
* So, the bits AD0 – AD7 are bi-directional and serve as A0 – A7 and D0 – D7 at the same time.
* During the execution of the instruction, these lines carry the address bits during the early part, then during the late parts of the execution, they carry the 8 data bits.
* In order to separate the address from the data, we can use a latch to save the value before the function of the bits changes.
* Control Pins – RD, WR
* These are active low Read & Write pins
* Status Pins – ALE, IO/M (active low), S1, S0
* ALE (Address Latch Enable)-Used to de-multiplex AD7-AD0
* IO/M – Used to select I/O or Memory operation
* S1,S0– Denote the status of data on data bus
* Interrupt Pins – TRAP, RST7.5, RST 6.5, RST 5.5, INTR, INTA
* These are hardware interrupts used to initiate an interrupt service routine stored at predefined locations of the system memory.
* Serial I/O pins – SID (Serial Input Data), SOD (Serial Output Data)
  + These pins are used to interface 8085 with a serial device.
  + Clock Pins- X1, X2, CLK(OUT)
  + X1, X2– These are clock input pins. A crystal is connected between these pins such that fcrystal= 2f8085where fcrystal= crystal frequency & f8085= operating frequency of 8085
  + CLK(OUT) – This is an auxiliary clock output source
  + Reset Pins – Reset In **(active low)**, Reset Out
  + **Reset In** is used to reset 8085 whereas **Reset Out** can be used to reset other devices in the system
  + DMA (Direct Memory Access) pins – HOLD, HLDA
  + These pins are used when data transfer is to be performed directly between an external device and the main memory of the system.
  + Power Supply Pins – +VCC, VSS

**Procedure:**

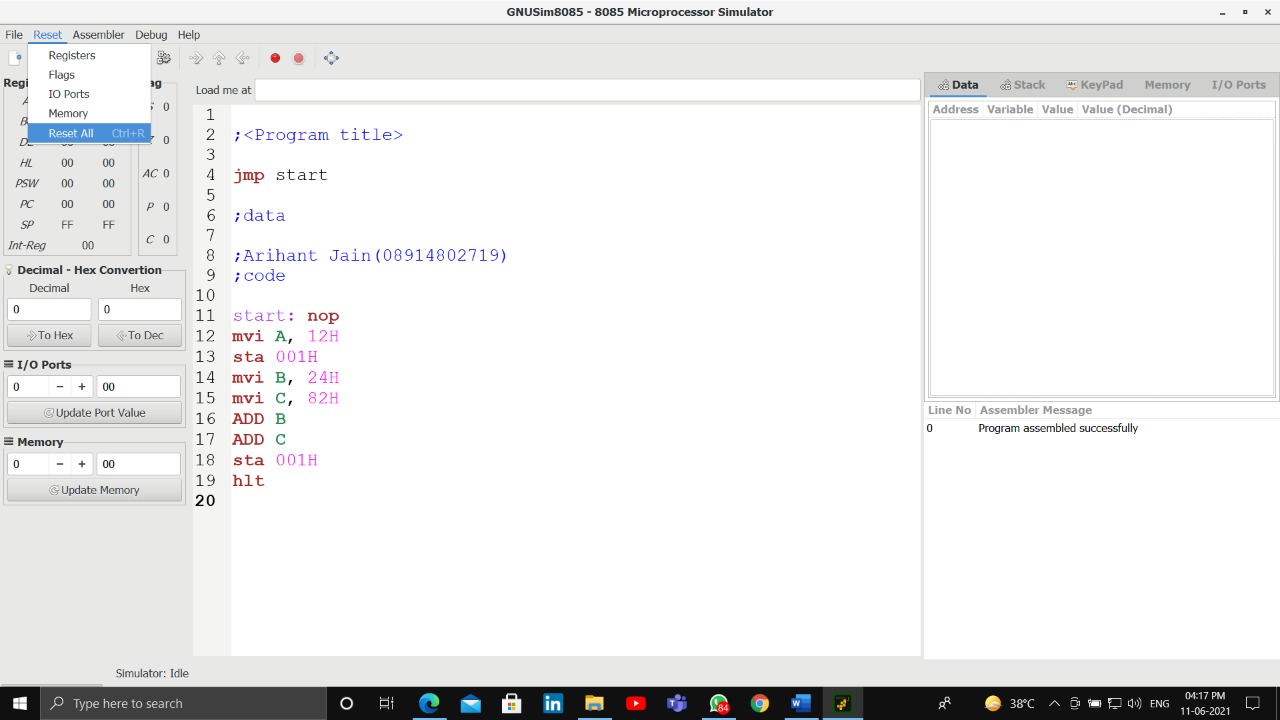
*Starting with GNU sim 8085*

**Step1:** Open GNU Sim 8085 above window will open. Now click on close button highlighted in the above screen shot.

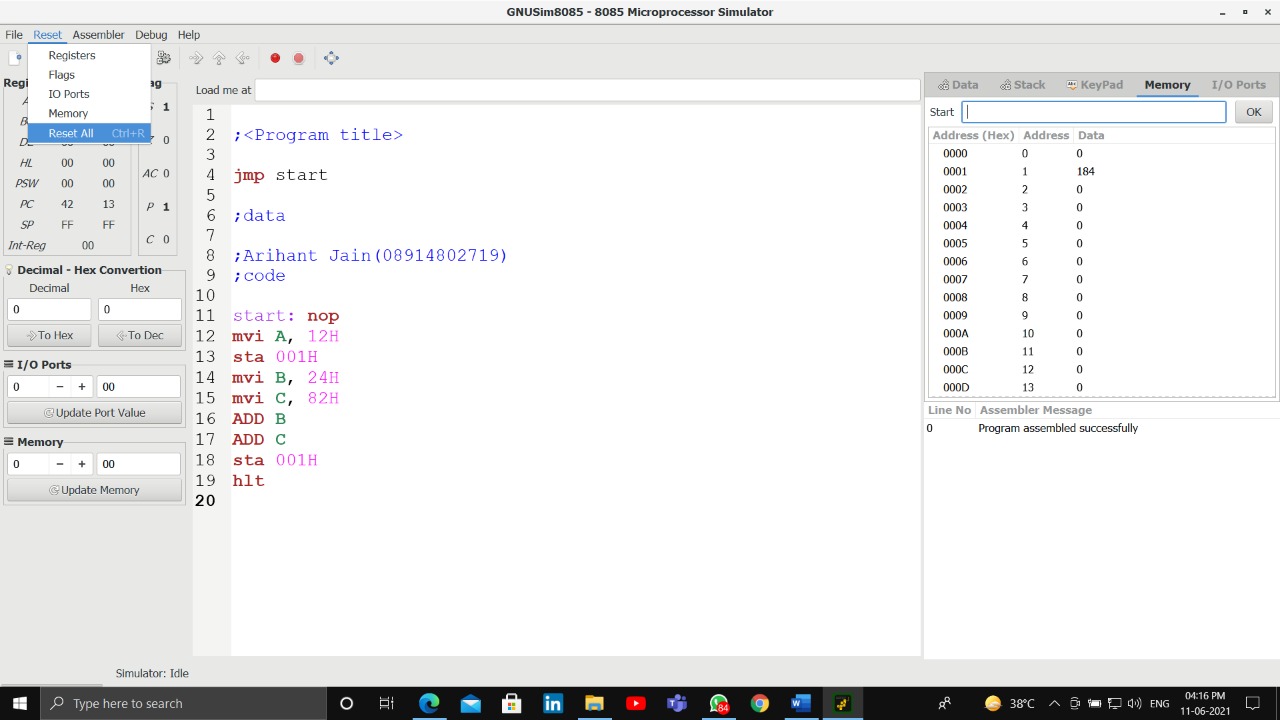
**Step2:** Start writing the code after start: nop in load me at 10 that is at load me at 11.



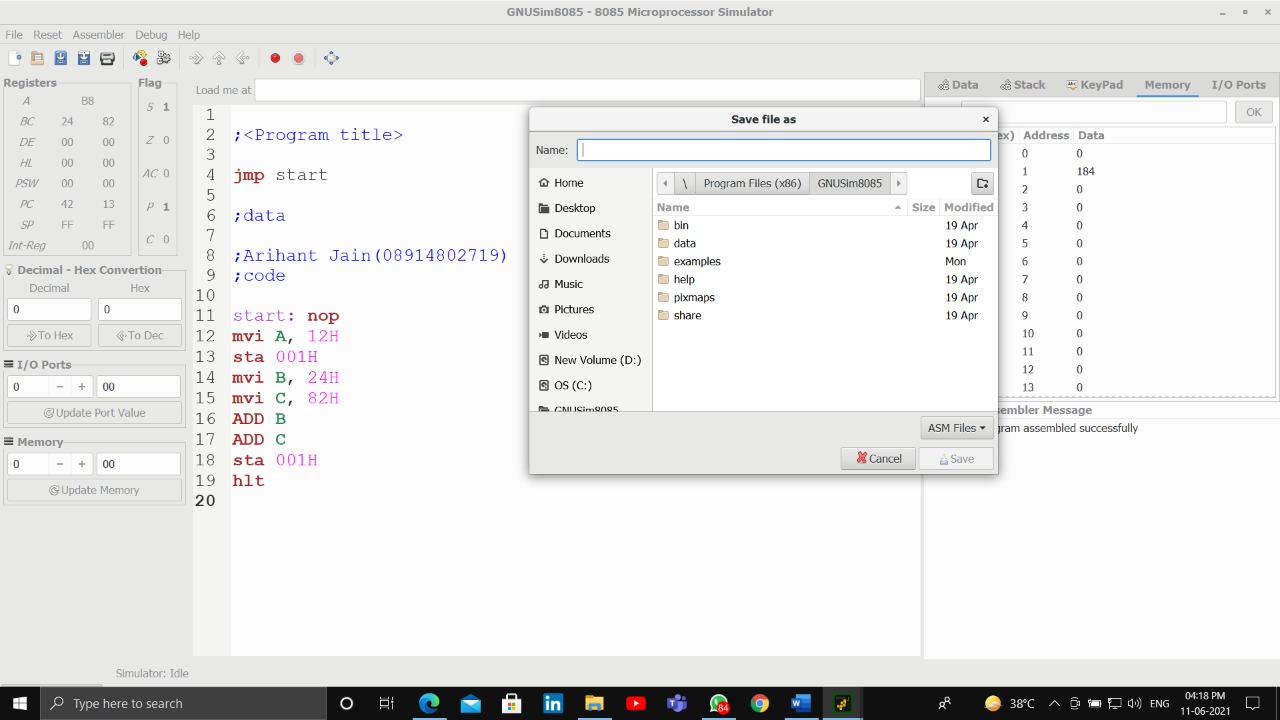
**Step 3**: Click on reset and reset all the registers by clicking on reset all.

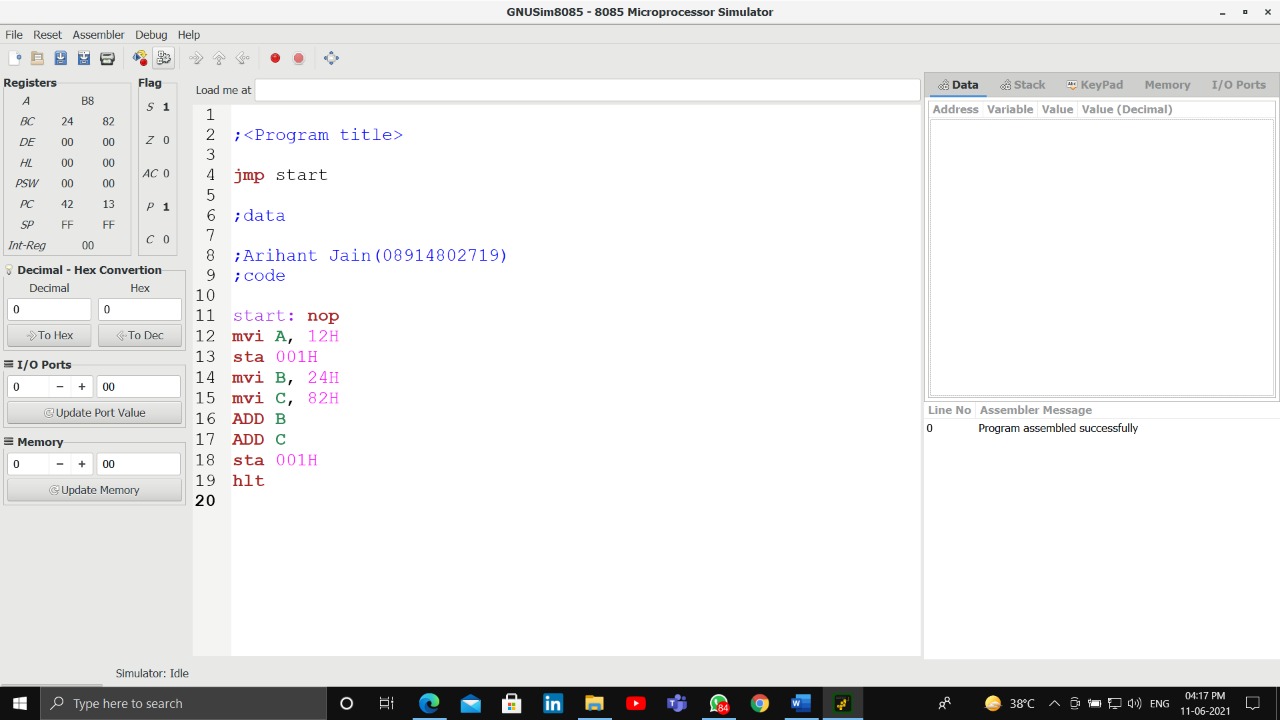


**Step 4**: Click on the highlighted button to execute the code



**Step 5**: After you execute the code mention the name your program by writing the name in the name section as mentioned in the screen shot in picture 5 and the drive where you want to save it. After that click on save.



**Step 6**: After this you will see the result of the instructions in the respective registers as seen in the above picture 6.

**INSTRUCTION SET**

**Data Transfer Group:**

Instructions which are used to transfer the data from a register to another register from memory to register or register to memory come under this group.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction Set | Explanation | States | Flags | Addre-ssing | Machine Cycles | Example |
| MOV r1, r2 [r1] ← [r2] | Move the content of the one register to another | 4 | none | Register | 1 | MOV A, B |
| MOV r, M [r]←[[H-L]] | Move the content of memory to register | 7 | none | Register Indirect | 2 | MOV B, M |
| MOV M, r [[H-L]]←[r] | Move the content of register to memory | 7 | none | Register Indirect | 2 | MOV M, C |
| MVI r, data [r] ←data | Move immediate data to register | 7 | None | Immediate Register | 3 | MVI M, 08 |
| LXI rp, data 16 [rp] ←data 16 bits, [rh] ←8 MSBs, [rl] ←8 LSBs of data | Load Register pair immediate | 10 | None | Immediate | 3 | LXI H, 2500H |
| LDA addr [A] ←[addr] | Load Accumulator direct | 13 | None | Direct | 4 | LDA 2400 H |
| STA Addr [addr] ←[A] | Store accumulator direct | 13 | None | Direct | 4 | STA 2000H |
| LHLD addr [L] ←[addr], [H] ← [addr + 1 ] | Load H-L pair direct | 16 | None | Direct | 5 | LHLD 2500H |
| SHLD addr [addr] ←[L], [addr +1] ← [H] | Store H-L pair direct | 16 | None | Direct | 5 | SHLD 2500 H |
| LDAX rp [A] ←[[rp]] | Load accumulator indirect | 7 | None | Register Indirect | 2 | LDAX B |
| STAX rp [[rp]] ←[A] | Store accumulator indirect | 7 | None | Register Indirect | 2 | STAX D |
| XCHG [H-L] ↔[D-E] | Change the contents of H-L with D-E pair | 4 | None | Register | 1 |  |

**Arithmetic Group:**

The instructions of this group perform arithmetic operations such as addition, subtraction, increment or decrement of the content of a register or a memory.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Instruction Set | Explanation | States | Flags | Addre-ssing | Machine Cycles | Example |
| ADD r [A] ←[A]+[r] | Add register to accumulator | 4 | All | Register | 1 | ADD K |
| ADD M [A] ← [A] + [[H-L]] | Add memory to accumulator | 7 | All | Register indirect | 2 | ADD K |
| ACC r [A] ← [A] + [r] + [CS] | Add register with carry to accumulator | 4 | All | Register | 1 | ACC K |
| ADC M [A] ← [A] + [[H-L]] [CS] | Add memory with carry to accumulator | 7 | All | Register indirect | 2 | ADC K |
| ADI data [A] ← [A] + data | Add immediate data to accumulator | 7 | All | Immediate | 2 | ADI 55K |
| ACI data [A] ← [A] + data + [CS] | Add with carry immediate data to accumulator | 7 | All | Immediate | 2 | ACI 55K |
| DAD rp [H-L] ←[H-L] + [rp] | Add register paid to H-L pair | 10 | CS | Register | 3 | DAD K |
| SUB r [A] ←[A]-[r] | Subtract register from accumulator | 4 | All | Register | 1 | SUB K |
| SUB M [A] ← [A] - [[H-L]] | Subtract memory from accumulator | 7 | ALL | Register indirect | 2 | SUB K |
| SBB r [A] ←[A]-[H-L]] - [CS] | Subtract memory from accumulator with borrow | 7 | All | Register indirect | 2 | SBB K |
| SUI data [A] ←[A]-data | Subtract immediate data from accumulator | 7 | All | Immediate | 2 | SUI 55K |
| SBI data [A] ←[A]-data-[CS] | Subtract immediate data from accumulator with borrow | 7 | All | Immediate | 2 | XCHG |
| INR r [r] ←[r]+1 | Increment register content | 4 | All except carry flag | Register | 1 | INR K |
| INR M [[H-L]] ←[[H-L]]+1 | Increment memory content | 10 | All except carry flag | Register indirect | 3 | INR K |
| DCR r [r] ←[r] -1 | Decrement register content | 4 | All except carry flag | Register | 1 | DCR K |
| DCR M [[H-L]] ← [[H-L]]-1 | Decrement memory content | 10 | All except carry flag | Register indirect | 3 | DCR K |
| INX rp [rp] ←[rp]+1 | Increment memory content | 6 | None | Register | 1 | INX K |
| DCX rp [rp] ←[rp]-1 | Decrement register pair | 6 | None | Register | 1 | DCX K |
| DAA | Decimal adjust accumulator | 4 |  |  | 1 | DAA |

**Logical Group:**

The instructions in this group perform logical operation such as AND, OR, compare, rotate, etc.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Instruction Set | Explanation | States | Flags | Addressing | Machine Cycles |
| ANA r [A] ←[A]∧[r] | AND register with accumulator | 4 | All | Register | 1 |
| ANA M [A] ←[A]∧[[H-]] | AND memory with accumulator | 4 | All | Register indirect | 2 |
| ANI data [A] ← [A] ∧ [data] | AND immediate data with accumulator | 7 | All | Immediate | 2 |
| ORA r [A] ←[A]∨[r] | OR-register with accumulator | 4 | All | Register | 1 |
| ORA M [A] ←[A]∨[[H-L]] | OR-memory with accumulator | 7 | All | Register indirect | 2 |
| ORI data [A] ← [A] ∨ [data] | OR -immediate data with accumulator | 7 | All | Immediate | 2 |
| XRA r [A] ← [A]∀[r] | XOR register with accumulator | 4 | All | Register | 1 |
| XRA M [A] ← [A] ∀ [[H-L]] | XOR memory with accumulator | 7 | All | Register indirect | 2 |
| XRI data [A] ←[A] ∀ [data] | XOR immediate data with accumulator | 7 | All | Immediate | 2 |
| CMA [A] ←[A] | Complement the accumulator | 4 | None | Implicit | 1 |
| CMC [CS] ←[CS] | Complement the carry status | 4 | CS |  | 1 |
| STC [CS] ← 1 | Set carry status | 4 | CS |  | 1 |
| CMP r [A]-[r] | Compare register with accumulator | 4 | All | Register | 1 |
| CMP M [A] - [[H-L]] | Compare memory with accumulator | 7 | All | Register indirect | 2 |
| CPI data [A] – data | Compare immediate data with accumulator | 7 | All | Immediate | 2 |
| RLC [An+1] ←[An], [A0] ←[A7], [CS] ←[A7] | Rotate accumulator left | 4 | Cs | Implicit | 1 |
| RRC [A7] ←[A0], [CS] ←[A0], [An] ←[An+1] | Rotate accumulator right |  | CS | Implicit | 1 |
| RAL [An+1] ←[An], [CS] ←[A7], [A0] ←[CS] | Rotate accumulator left through carry |  | CS | Implicit | 1 |
| RAR [An] ←[An+1], [CS] ←[A0], [A7] ←[CS] | Rotate accumulator right through carry |  | CS | Implicit | 1 |

**Viva - Questions:**

**1.What is the use of accumulator in the 8085 microprocessor?**

Ans: 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.

**2. How can we store data in 8085 microprocessor?**

Ans: Suppose, the instruction is ADD A and B. This simply tells the ALU to add the data present in the B register with the data present in the accumulator i.e., A register. But, in 8085 the decoded instruction is simply **ADD B**. So, automatically, the ALU adds the value present in the accumulator with the data in register B.

**3.Explain the use of flag registers in the 8085 architecture.**

Ans: The Flag register is a Special Purpose Register. Depending upon the value of result after

any arithmetic and logical operation, the flag bits become set (1) or reset (0). In the 8085 microprocessor, the flag register consists of 8 bits and only 5 of them are useful.

Experiment-2

**Aim:** Write the working of GNUsim8085.

**INTRODUCTION TO GNU Simulator 8085:**

8085 simulator is software on which instructions are executed by writing the programs in assembly language.

1. GNUSim8085 is a 8085 microprocessor simulator with following features.

2. A simple editor component with syntax highlighting.

3. A keypad to input assembly language instructions with appropriate arguments.

4. Easy view of register contents.

5. Easy view of flag contents.

6. Hexadecimal to Decimal converter.

7. View of stack, memory and I/O contents.

8. Support for breakpoints for programming debugging.

9. Stepwise program execution.

10. One click conversion of assembly program to opcode listing.

11. Printing support (known not to work well on Windows).

12. UI translated in various languages.

Format of the instruction is as follows:-

|  |  |  |  |
| --- | --- | --- | --- |
| Label | Operation | Operands | Comments |
| Its optional | Necessary | Necessary | Its optional |

A basic assembly program consists of 4 parts.

**1. Labels**

**2. Operations:-** these operations can be specified as

Machine operations (mnemonics):- used to define operations in the form of opcode as mentioned in the instruction set of microprocessor 8085.

Pseudo operations (like preprocessor in C):- these are assembly directives.

**3. Operands**

**4. Comments**

In addition, you have constants in an assembly program. Unless otherwise specified, a constant which is always numeric is in decimal form. If appended with a character h it is assumed to be in hexadecimal form. If a hex constant starts with an alpha-char don't forget to include the number 0 in the beginning, since that will help the assembler to differentiate between a label and a constant.

**Labels:** When given to any particular instruction/data in a program, takes the address of that instruction or data as its value. But it has a different meaning when given to the EQU directive. Then it takes the operand of EQU as its value. Labels must always be placed in the first column and must be followed by an instruction (no empty line). Labels must be followed by a : (colon), to differentiate it from other tokens.

**Operations:** As mentioned above the operations can be specified in two ways that are mnemonics and pseudo operation.

Pseudo operations can be defined by using following directives:-

There are only 3 directives currently available in our assembly language.

1. DB - define byte ( 8 bits )

2. DS - define size (no. of bytes)

3. EQU - like minimalistic #define in C

DB is used to define space for an array of values specified by comma separated list. And the label (if given to the beginning of DB) is assigned the address of the first data item.

DS is used to define the specified number of bytes to be assigned and initialize them to zero. To access each byte you can use the + or -operator along with a label.

EQU behaves similar to #defined in C. But it is simple. It can be used to give names only to numeric constants. Nesting of EQU is not allowed. You can use EQU only in operands for pseudo ops and mnemonics.

**Operands:** Operands are specified according to the user. The register set specified in the architecture of 8085 (A, B, C, D, H and L) are used to access and store data. These registers are specified as operands. In case of accessing data or storing data in the memory ‘m’ is specified as an operand and the address of this memory location is taken from the HL pair (data in HL pair).

**Step1:** Open GNU Sim 8085 above window will open.

**Step2:** Start writing the code after start: nop in load me at 10 that is to load me at 11.

**Step 3:** Click on reset and reset all the registers by clicking on reset all.

**Step 4:** Click on the highlighted button to execute the code

**Step 5:** After you execute the code, mention the name of your program by writing the name in the name section as mentioned in the screenshot in picture 5 and the drive where you want to save it. After that click on save.

**Step 6:** After this you will see the result of the instructions in the respective registers as seen in the above picture 6.

**VIVA QUESTIONS:**

**1. What are the important CPU registers in the 8085 microprocessor? Explain.**

a. Accumulator (A): It is an 8-bit register which involves in all accumulator related instructions.

b. Program counter (PC): It is a 16-bit register which carries the address of the next instruction to be executed.

c. General purpose registers (A, B, C, D, E, H, L): These registers can he used individually or as BC, DE and HL register pairs.

d. Stack pointer (SP): A 16-bit register always points the top of the stack.

e. Flag register (F): Contains five flags which indicate the processor status.

Example: STA 4100; Address 4100 is specified in the instruction

**2. Which are the addressing modes of 8085 ? Explain with examples.**

a. Direct addressing: In this mode of addressing the address of the operand is specified in the instruction.

Example: STA 4100; Address 4100 is specified in the instruction

b. Register addressing: Operands will be in the register.

Example: MOV A, B ; Name of the register will be specified in the instruction.

c. Register indirect addressing: Address of the operand will be placed in the register.

Example: MOV A, M ; address of the memory location is kept in HL pair.

d. Immediate addressing: Operand is specified in the instruction.

Example: MVI A, 44 ; datum 44 is specified in the instruction.

**3. How many data lines and address lines are in an 8085 microprocessor?**

8 data lines and 16 address lines.

Experiment-3

**Aim:** Write a program to perform:

1. Addition of two 8-bit numbers without carry.

2. Addition of two 8-bit numbers with carry.

**Theory: *Add without Carry using mvi***

• First Load “Zero” value into accumulator and the memory location where the final answer is to be stored.

• Load a couple of Numbers in registers

• Add the values of registers into the Accumulator one at a time using “ADD”

• Using “sta” load the value in the accumulator into a memory location

• The memory location has the sum of two numbers

**#Code**

*;<Adding two numbers without carry using mvi>*

*jmp start*

*;data*

*;Arihant jain(08914802719)*

*;code*

*start: nop*

*mvi A, 22H*

*sta 001H*

*mvi B, 21H*

*mvi C, 86H*

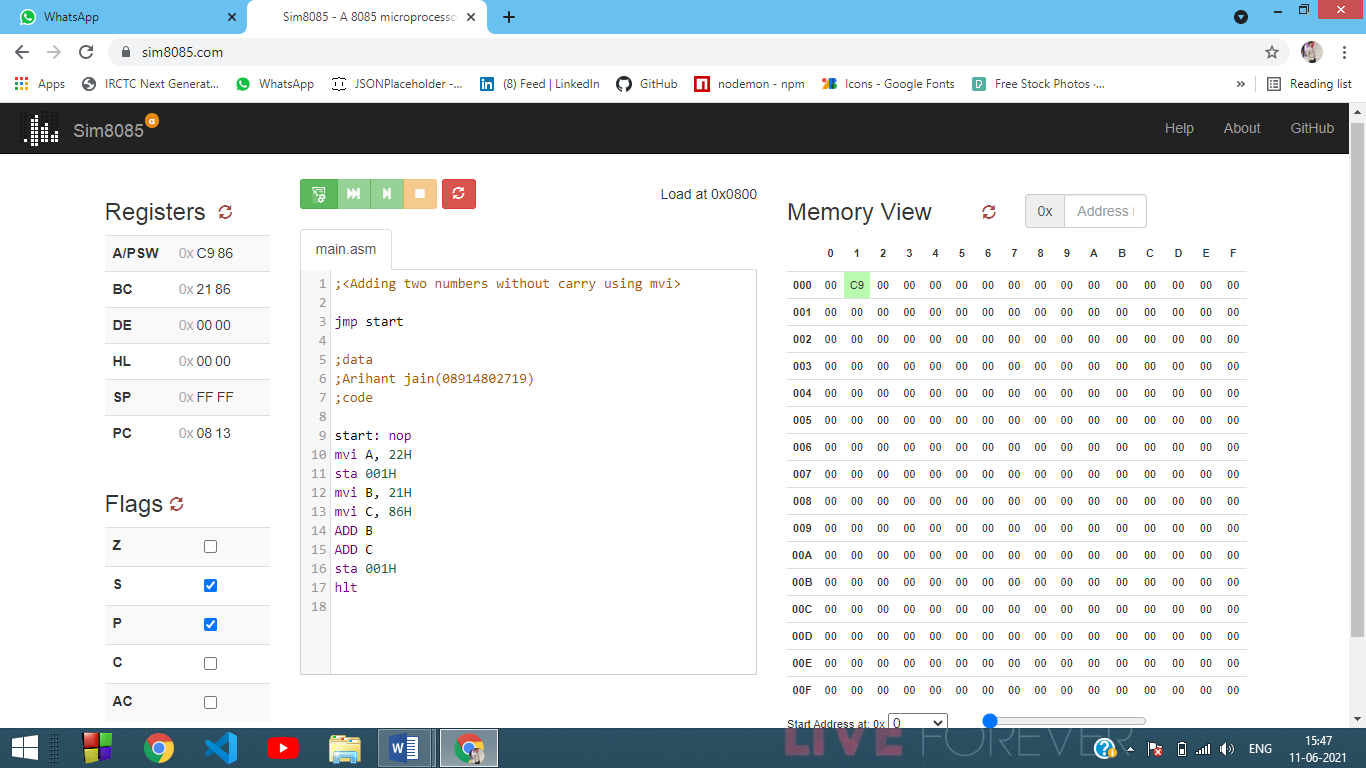
*ADD B*

*ADD C*

*sta 001H*

*hlt*

*OUTPUT*

**

**Theory: *Add without Carry using lxi***

• First make sure that accumulator and the memory location where answer is to be stored are set to Zero

• Load values from memory locations using lxi and store them into registers (earlier half)

• One at a time add the values of registers in accumulator

• Store the now value of accumulator into a desired memory location

**#Code**

*;< Adding two numbers without carry lxi>*

*jmp start*

*;data*

*;Arihant jain(08914802719)*

*;code*

*start: nop*

*mvi a,73h*

*lxi b,65h*

*lxi d,39h*

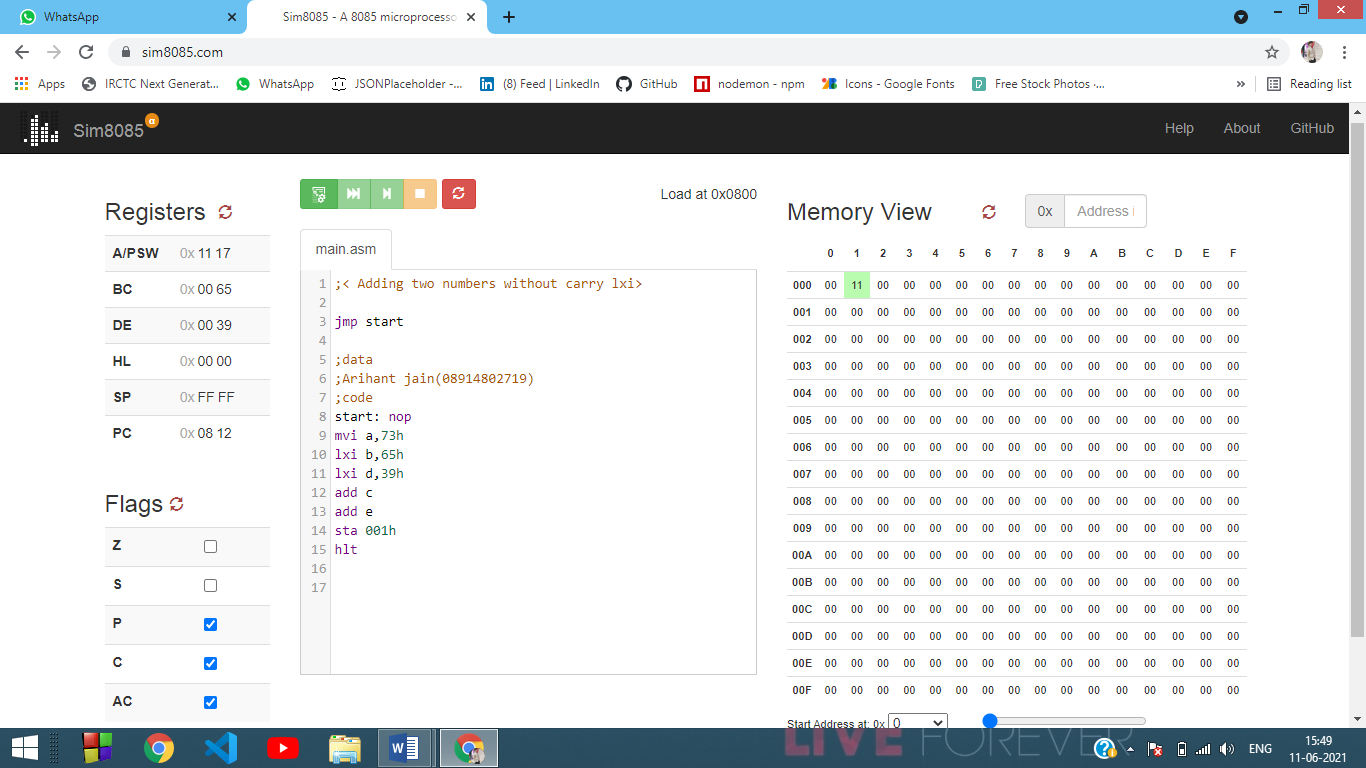
*add c*

*add e*

*sta 001h*

*hlt*

**Output**

****

**Theory: *Add with Carry***

• First Load “Zero” value into the memory location where the final answer is to be stored.

• Load a couple of Numbers in registers/Accumulator.

• Add the values of registers into the Accumulator one at a time using “ADD”.

• If in-case there is any Carry the C Flag will take note of it, and then the value has to be increase, else the increase command is to be skipped.

• Using “sta” load the value in the accumulator into a memory location

• The memory location has the sum of two numbers

**#Code**

;<Addition with carry>

jmp start

;data

;ARIHANT(08914802719)

;code

start: nop

mvi a, 69h

mvi b, 55h

add b

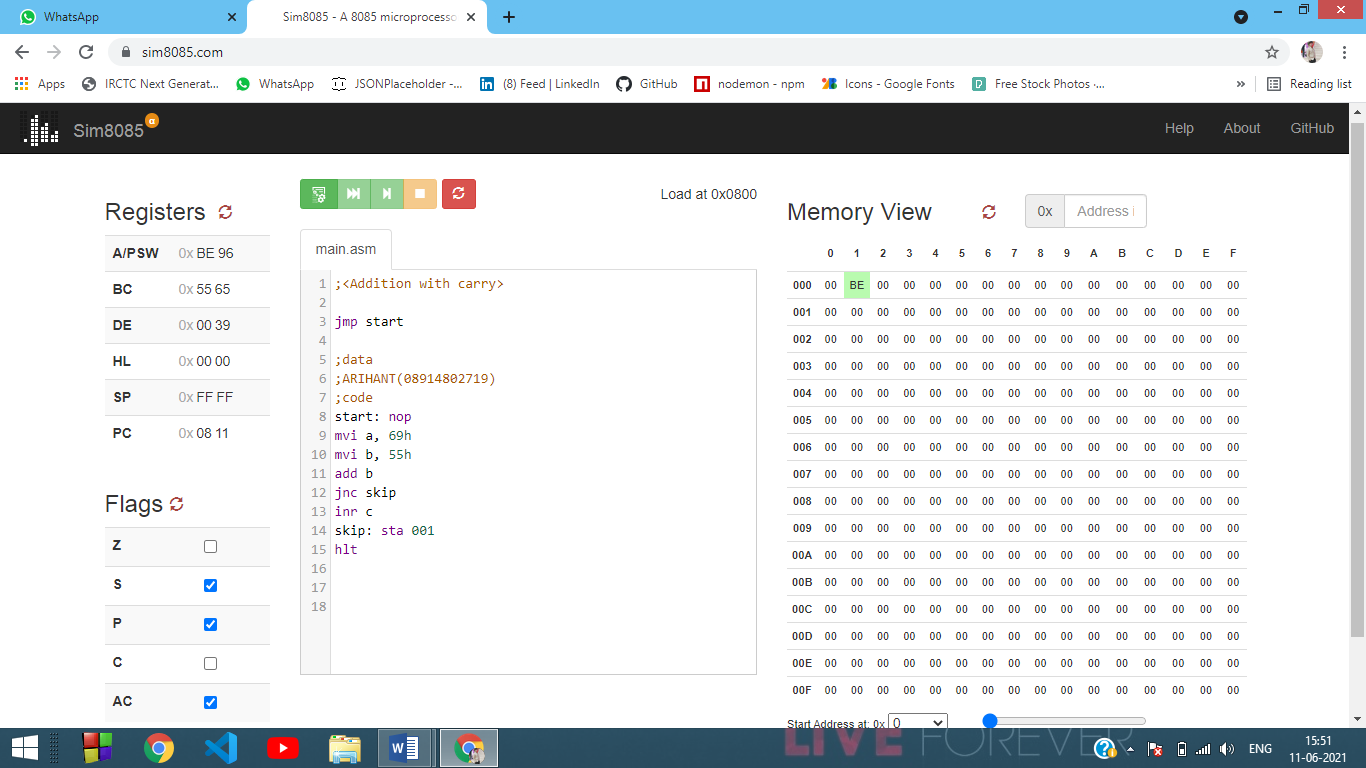
jnc skip

inr c

skip: sta 001

hlt

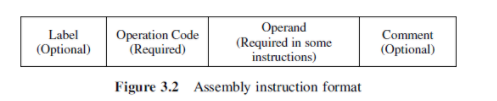
Output



**Viva Questions**

**1. What is the syntax for writing instructions in assembly code using mnemonics?**

Ans:



**2. Which language makes use of mnemonics instead of numeric op-codes & symbolic names for data locations instead of numeric addresses?**

Ans: Assembly language.

Experiment-4

**Aim:** Write a program to perform:

1. Subtraction of two 8-bit numbers without borrows.
2. Subtraction of two 8-bit numbers with borrows*.*

**Theory: Subtraction without Carry**

* Load some value into the registers/accumulator
* First add the larger value into the Accumulator using “ADD”
* Subtract the other Value from the Accumulator using “SUB”
* Using “STA” you can load the answer into a desired memory location (which was set to zero before hand)

**#Code**

;<Subtract without Carry>

jmp start

;data

;ARIHANT JAIN(08914802719)

;code

start: nop

mvi a,30h

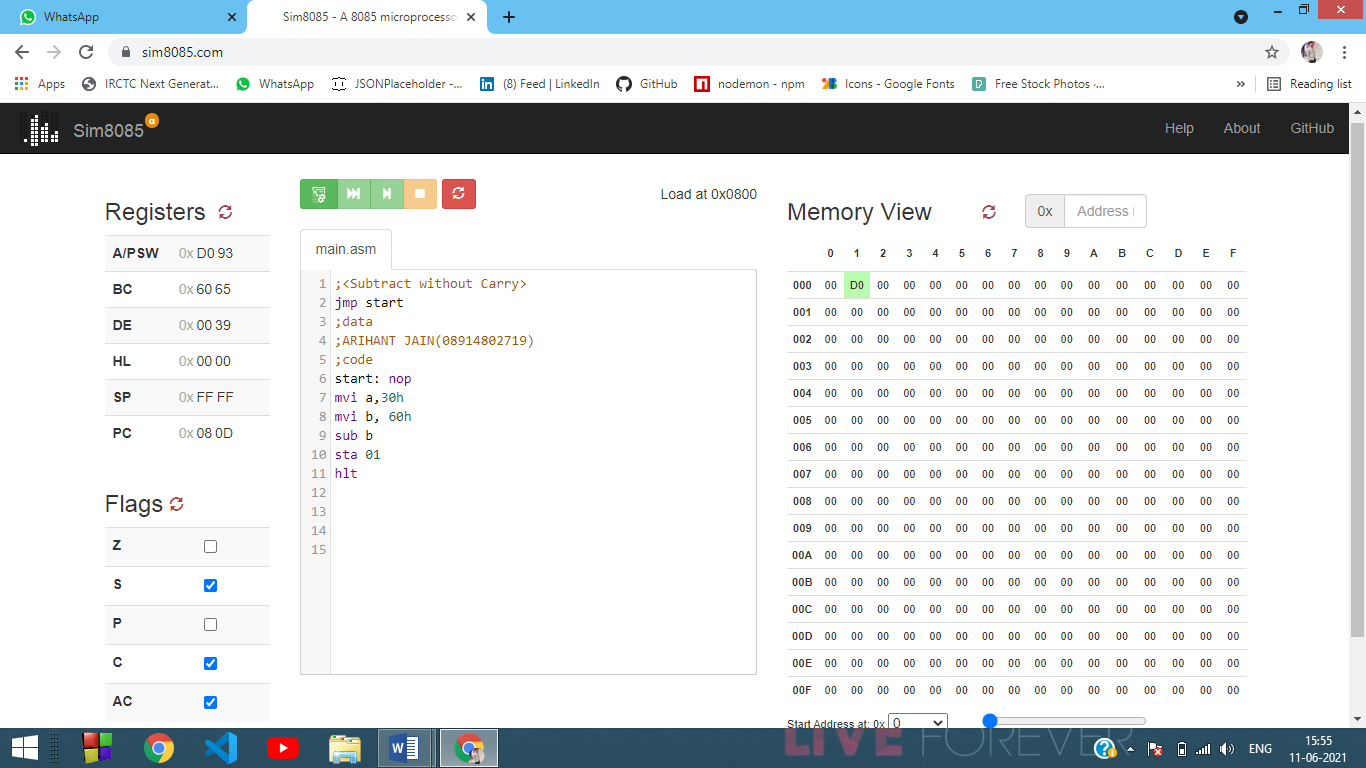
mvi b, 60h

sub b

sta 01

hlt

Output



**Theory*:* Subtraction with Carry**

* Load some value into the accumulator/registers.
* Add one value into the accumulator if it’s empty.
* Subtract the other value from the accumulator.
* If there is a carry it’s caught by C-Flag.
* Take complement of the value in accumulator
* Increase the value of accumulator and C register by one
* Store the answer in a memory location.
* **#Code**

;<Subtract with Carry>

jmp start

;data

;ARIHANT JAIN(08914802719)

;code

start: nop

mvi a, 56h

mvi b, 83h

sub b

jnc skip

cma

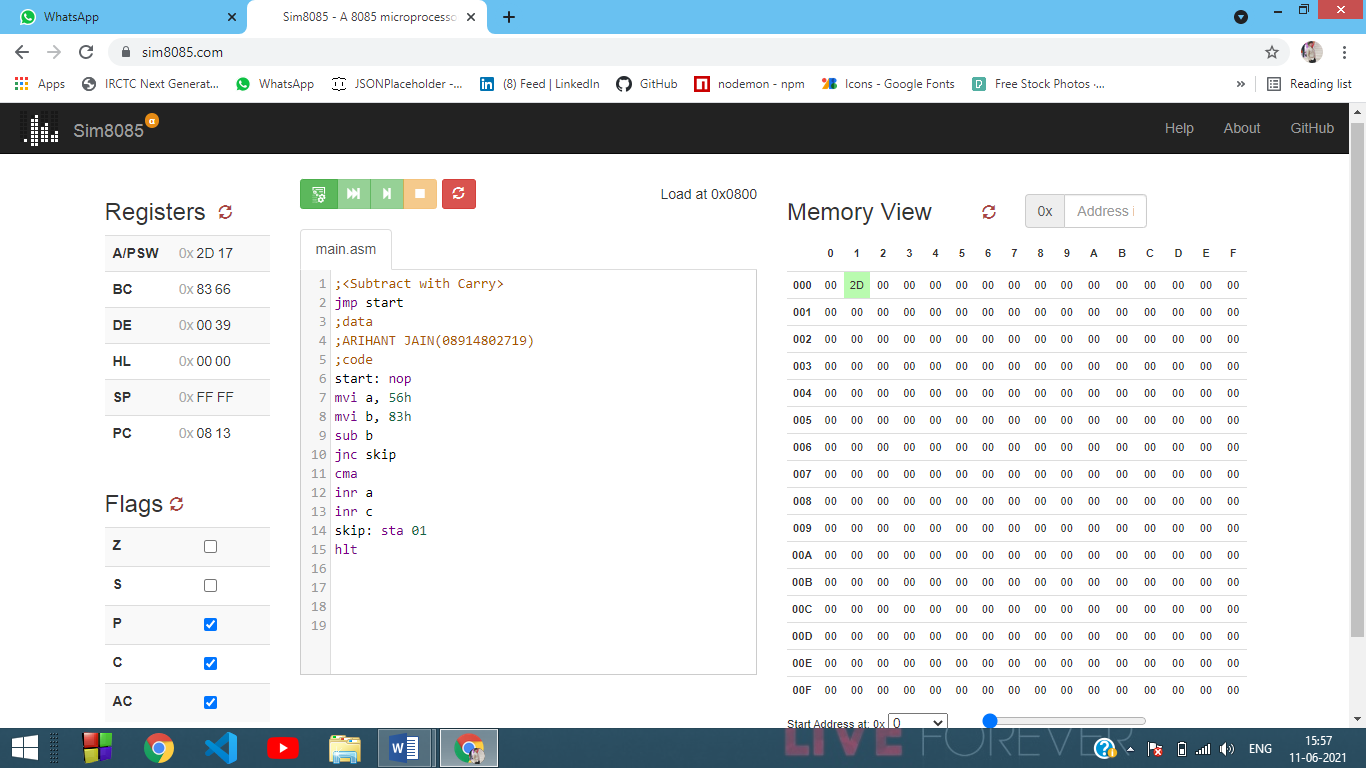
inr a

inr c

skip: sta 01

hlt

Output



**VIVA QUESTIONS**

**1.MVI m, 30h. The address of the memory location will be taken from which registers?**

Ans: In the 8085 Instruction set, this instruction MVI M, d8 is used to load a memory location pointed by HL pair with an 8-bit value directly. This instruction uses immediate addressing for specifying the data. It occupies 2-Bytes in memory.

**2.Which instructions for an 8085 microprocessor can be used if we need to store 45H into a register?**

Ans: MVI R,DATA

MVI R(NAME OF REGISTER),DATA(45H)

Experiment-5

**Aim**: Write a program to find 1‘s complement of an 8-bit number.

**Theory:**

There is an inbuilt function for 1’s complement, it’s **“cma”.**

* Load some value into the accumulator.
* Use the command “**cma”**
* *Store the Answer into a memory location*

**Other Method:**

* First load **0FFH**(Maximum value) in accumulator.
* Subtract the value who’s one’s complement is to be evaluated.
* The value in Accumulator is 1’s complement of that value.
* **#Code**

*;<1's Complement>*

*jmp start*

*;data*

*;ARIHANT JAIN(08914802719)*

*;code*

*start: nop*

*mvi a , 20h*

*mvi b , 0fh*

*mvi a , 0ffh*

*sub b*

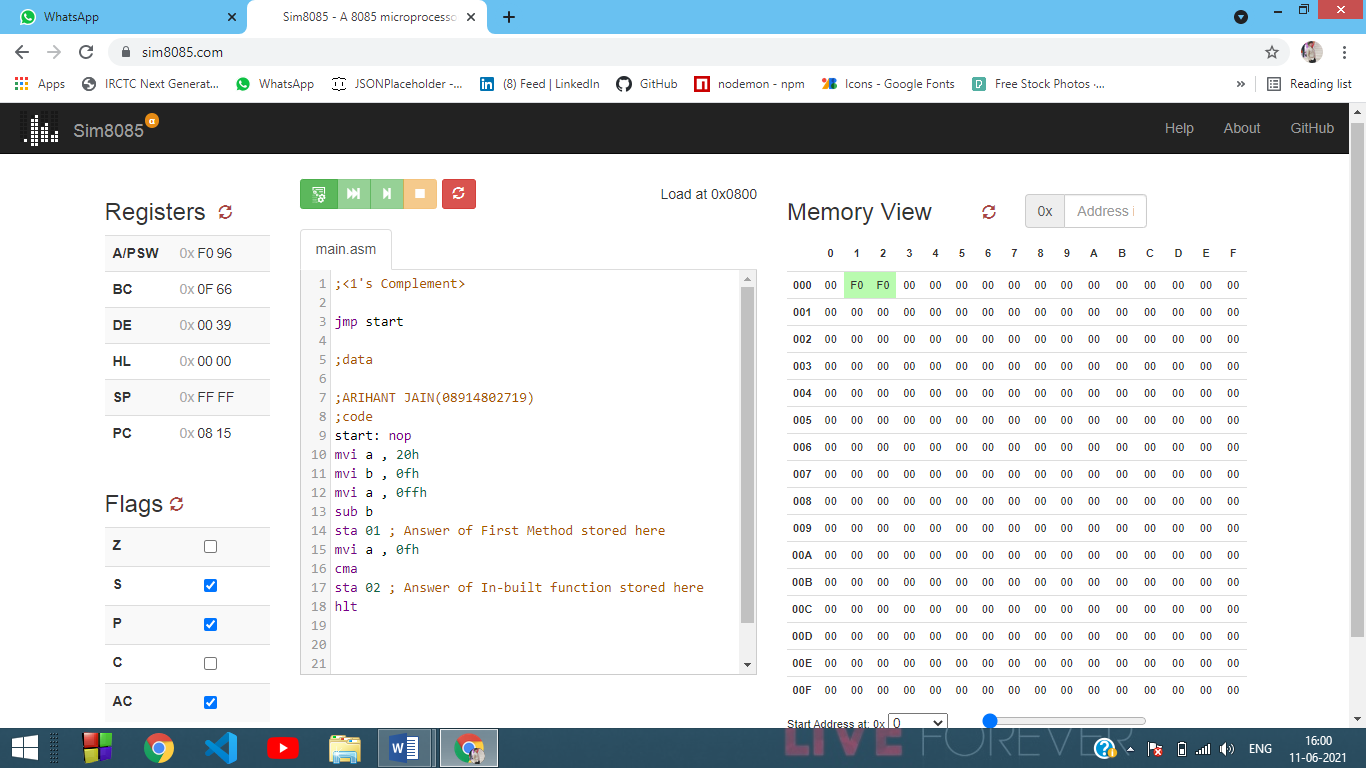
*sta 01 ; Answer of First Method stored here*

*mvi a , 0fh*

*cma*

*sta 02 ; Answer of In-built function stored here*

*hlt*

**

**VIVA QUESTIONS**

**1. What will be the result of the following code when written in GNU sim 8085 ?**

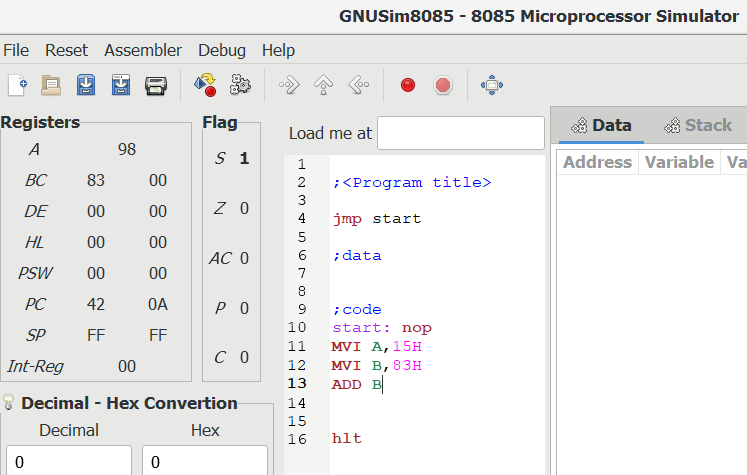
MVI A, 15h

MVI B, 83h

ADD B

HLT

Ans:



**2.Which instruction is used to get the BCD addition result in gnu sim?**

Ans:

1. **MVI** is used to move data immediately into any of registers (2 Byte)
2. **LHLD** is used to load register pair direct using 16-bit address (3 Byte instruction)
3. **MOV** is used to transfer the data from memory to accumulator (1 Byte)
4. **ADD** is used to add accumulator with any of register (1 Byte instruction)
5. **STA** is used to store data from accumulator into memory address (3 Byte instruction)
6. **DAA** is used to check if sum > 9 or AC = 1 add 06 (1 Byte instruction)
7. **JNC** is used jump if no carry to given memory location (3 Byte instruction)
8. **INR** is used to increase given register by 1 (1 Byte instruction)
9. **HLT** is used to halt the program

**3. The Carry flag is undefined after performing the \_\_\_\_\_\_\_\_\_\_operation.**

Ans: AAD

Experiment-6

**Aim**:Write a program to find 2‘s complement of an 8-bit number.

**Theory:** To find 2’s complement first calculate 1’s complement of the number then increase the value by 1 to obtain 2’s compliment.

* Load some value into the accumulator.
* Find the complement of that value using “**cma”.**
* Using **inr** increase the value of accumulator by one.
* Store the Answer into a memory location.
* **#Code**

;<2's Complement>

jmp start

;data

; ARIHANT JAIN(08914802719)

;code

start: nop

mvi a , 0fh

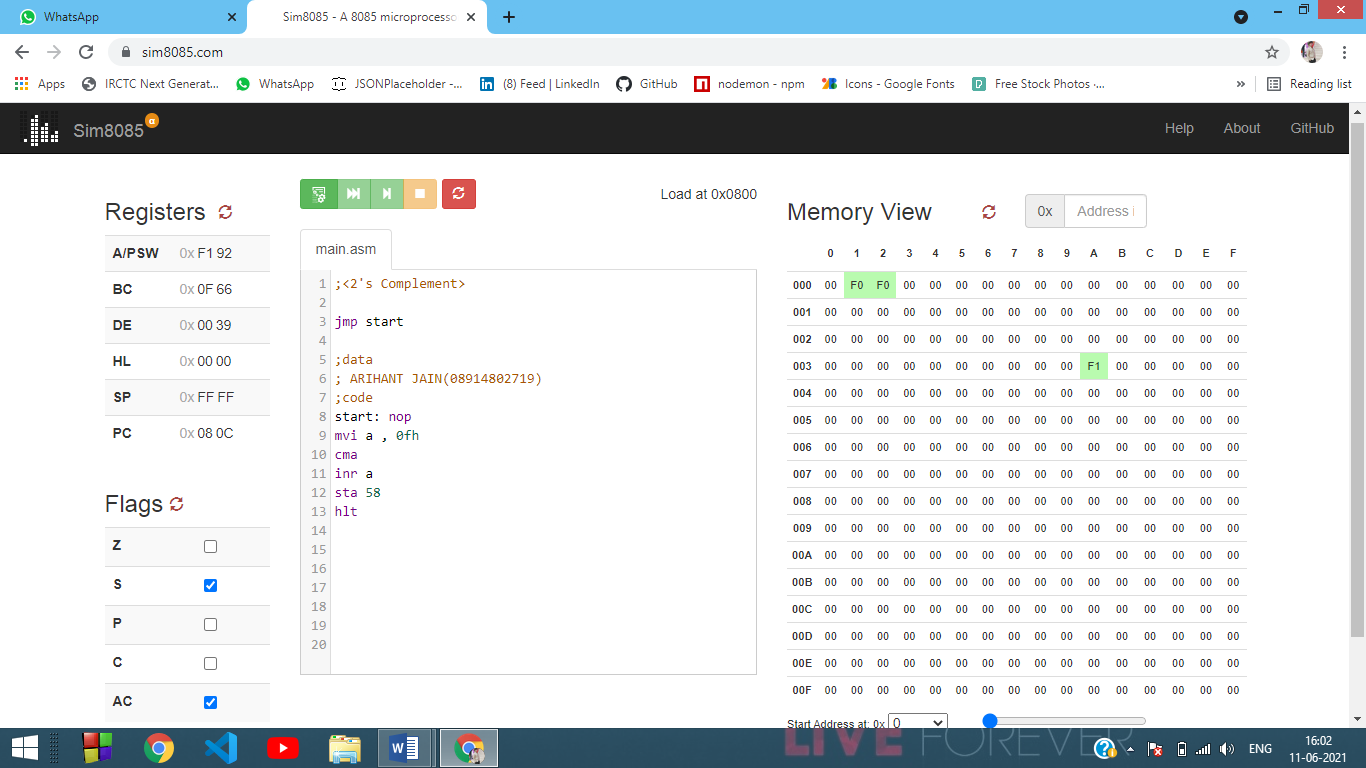
cma

inr a

sta 58

hlt

Output

****

**VIVA QUESTIONS**

**1. How does the data get stored when we use LXI instruction?**

Ans: In the 8085 Instruction set there are four instructions, which belong to the type LXI rp, d16. These instructions are used to load the 16-bit address into the register pair. We can use this instruction to load data from memory location using the memory address, which is stored in the register pair rp. For an example, if the instruction is LXI H, FE50. It means that the FE50 is loaded into the HL register pair.

**2.How are negative numbers represented as data?**

Ans: Whenever a number with minus sign is encountered, the number (ignoring minus sign) is converted to its binary equivalent. Then the two’s complement of the number is calculated. That two’s complement is kept at place allocated in memory and the sign bit will be set to 1 because the binary being kept is of a negative number. Whenever it comes on accessing that value firstly the sign bit will be checked if the sign bit is 1 then the binary will be two’s complement and converted to equivalent decimal number and will be represented with a minus sign.

EXPERIMENT-7

**AIM:** Write an assembly language code to store numbers in reverse order in memory location.

**CODE:**

*;<program title>*

*jmp start*

*;data*

*;ARIHANT JAIN(08914802719)*

*;code*

*start: nop*

*LDA 0000H ;Load value of memory location in accumulator*

*RLC ; Rotate content of accumulator left by 1 bit*

*RLC ;Rotate content of accumulator left by 1 bit*

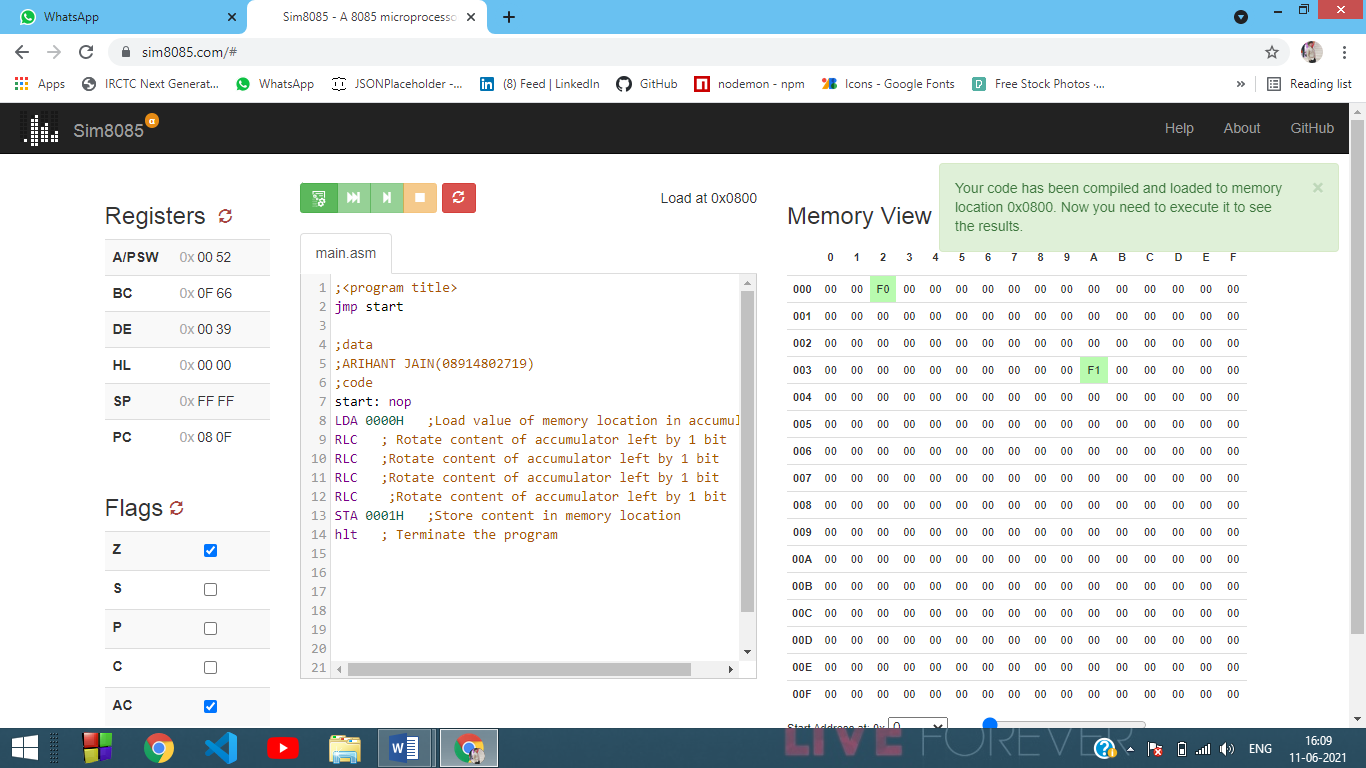
*RLC ;Rotate content of accumulator left by 1 bit*

*RLC ;Rotate content of accumulator left by 1 bit*

*STA 0001H ;Store content in memory location*

*hlt ; Terminate the program*

**OUTPUT:**

****

**VIVA QUESTIONS**

**Q1) List the instructions from the instruction set of 8085 which work on immediate addressing mode?**

A1) In this mode, the 8/16-bit data is specified in the instruction itself as one of its operands. For example: MVI K, 20F: means 20F is copied into register K.

**Q2) What is the significance of stack pointers?**

A2) The Stack Pointer (SP) register is used to indicate the location of the last item put onto the stack. When you PUT something ONTO the stack (PUSH onto the stack), the SP is decremented before the item is placed on the stack.

**Q3) The data entered in the registers is in which number system. Is the data stored in the memory of the same number system as in registers?**

A3) In Binary form.

EXPERIMENT-8

**AIM:** Write an assembly language in GNUsim8085 to add two 16 bit numbers without using lxi instruction.

**CODE:**

;<program tiltle >

jmp start

;data

;ARIHANT JAIN(08914802719)

;code

start: nop

LDA 0000H ; Load value of memory location in accumulator

MOV B, A ; B <- A

LDA 0003H

ADD B ; A <- A + B

STA 0006H

LDA 0001H

MOV B, A ; B <- A

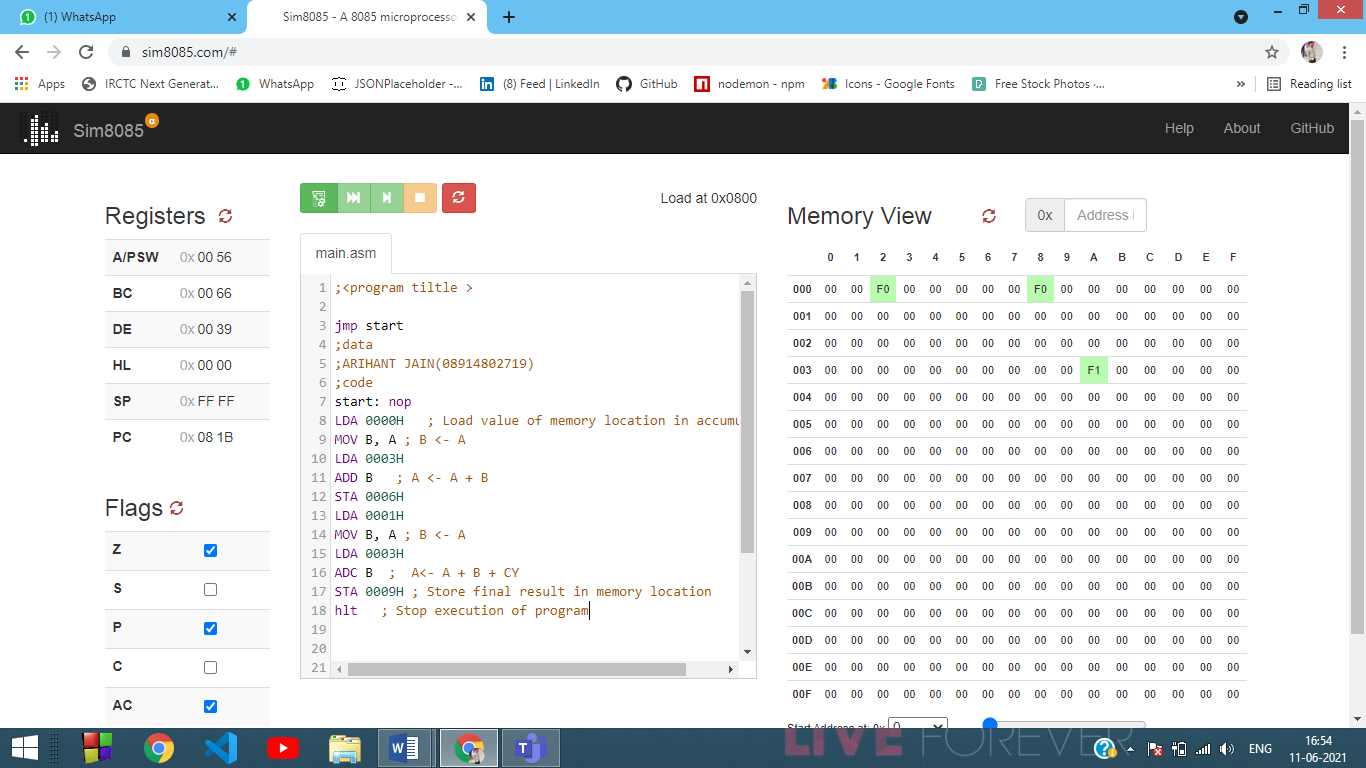
LDA 0003H

ADC B ; A<- A + B + CY

STA 0009H ; Store final result in memory location

*hlt ; Stop execution of program*

**OUTPUT:**

****

**VIVA QUESTIONS**

1. **What is the use of a READY pin of 8085?**

Microprocessor waits until the signal on this line is high to access data from a peripheral device. This is used to delay the microprocessor Read or Write cycles until a slow peripheral device is ready to access the data.

1. **List the instructions from the instruction set of 8085 which work on immediate addressing mode?**

In this mode, the 8/16-bit data is specified in the instruction itself as one of its operands. For example: MVI K, 20F: means 20F is copied into register K.

Experiment-9

**Aim:** Write a program to perform Multiplication of two 8 bit numbers.

**Theory:**

* Make sure that the accumulator and the desired memory location to store the values and final answer are initially set to Zero.
* Load two numbers to be multiplied into the registers.
* Make a loop where you keep on adding one of the two values into the accumulator, while you gradually decrease the other value.
* As long as the other value is not zero repeat the loop.
* When the loop ends store the value into the desired memory location.
* **#Code**

*;<Multiplication of 2 8-bit numbers>*

*jmp start*

*;data*

*;ARIHANT JAIN(08914802719)*

*;code*

*start: nop*

*mvi a, 01h*

*mvi b, 59h*

*mvi c, 60h*

*loop: add b*

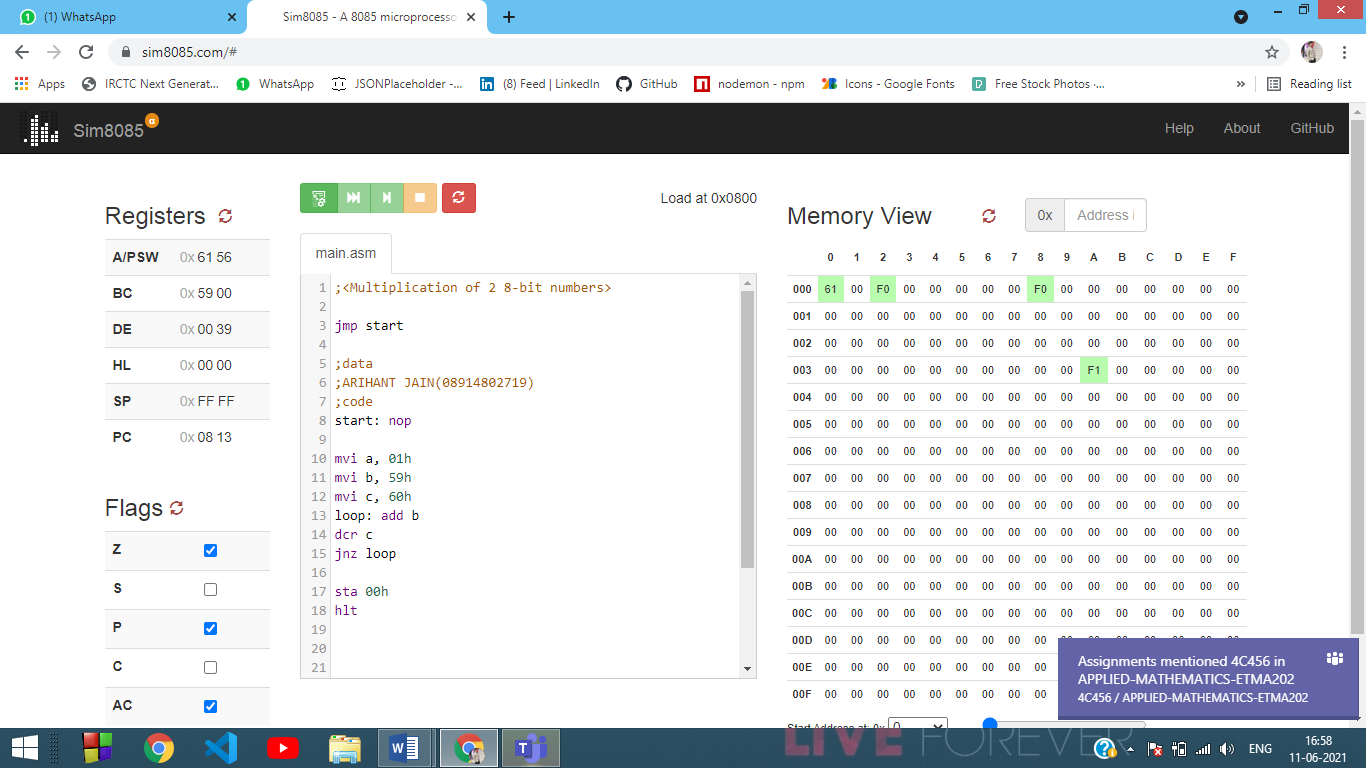
*dcr c*

*jnz loop*

*sta 00h*

*hlt*

Output



Experiment-10

**Aim:** Write a program to perform Division of two 8-bit numbers.

**Theory:**

* Load the first value to accumulator.
* Start a loop where you subtract the second value from the accumulator and keep a tally of the iterations in an initially empty register.
* Keep on comparing the second value with the value in accumulator, if the value in accumulator is higher then that of second value there will be no carry flag, and thus, keep on repeating the loop.  
  If the values in second register is equal to that of in accumulator (at that time, the zero flag will be true, but not carry flag). Only when **Value in accumulator is less than second value** carry flag will be true, then terminate the loop.
* The then value in accumulator is the remainder in division.
* The then value in the initially Zero register where the tally was kept contains the Quotient of division.
* Store them separately in memory locations.
* **#Code**

*jmp start*

*;data*

*;Arihant Jain(08914802719)*

*;code*

*start: nop*

*mvi a ,00h*

*mvi b ,0fh*

*mvi c ,06h*

*mvi d ,00h*

*add b*

*loop: sub c*

*inr d*

*cmp c*

*jnc loop*

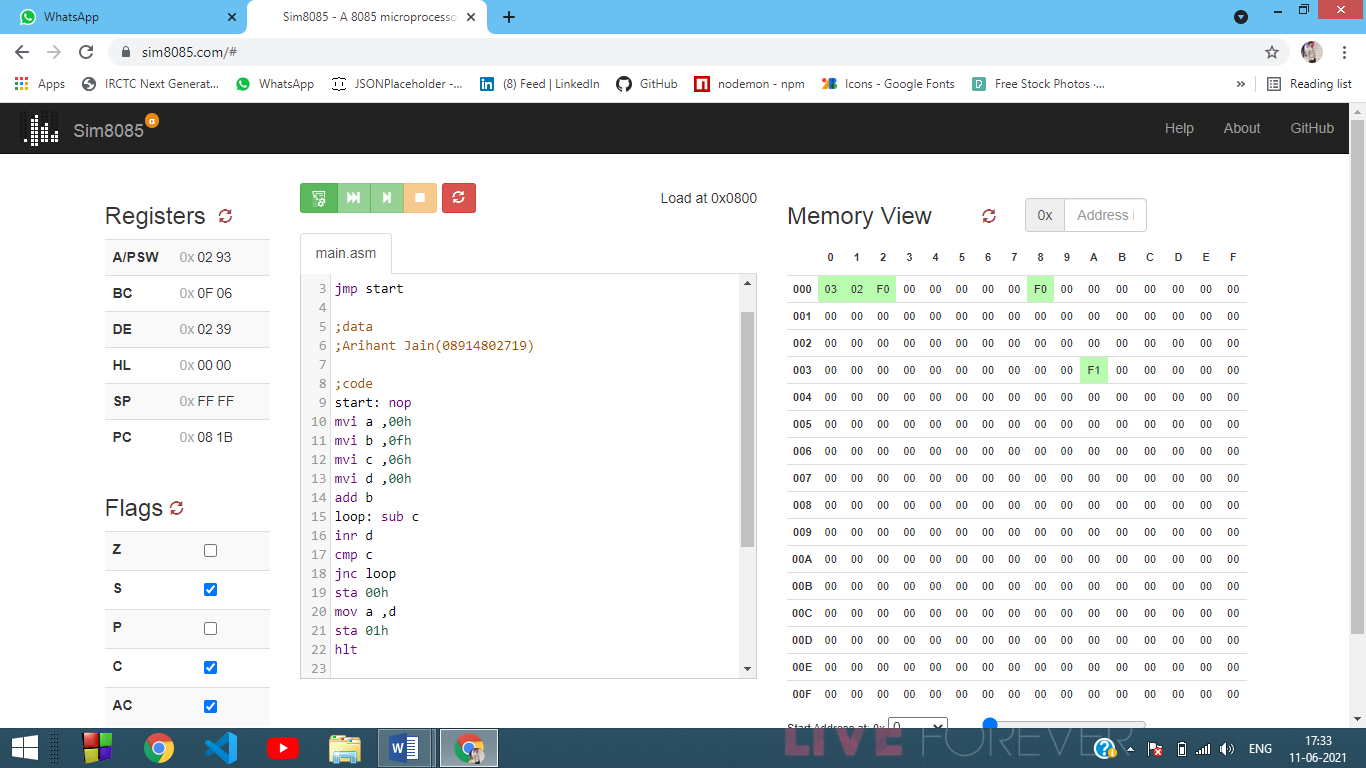
*sta 00h*

*mov a ,d*

*sta 01h*

*hlt*

Output



**VIVA QUESTIONS**

1. **What is the use of a READY pin of 8085?**

Microprocessor waits until the signal on this line is high to access data from a peripheral device. This is used to delay the microprocessor Read or Write cycles until a slow peripheral device is ready to access the data.

1. **List the instructions from the instruction set of 8085 which work on immediate addressing mode?**

In this mode, the 8/16-bit data is specified in the instruction itself as one of its operands. For example: MVI K, 20F: means 20F is copied into register K.