Practical – 4

Aim: Introduction to GNU Simulator 8085.

GNU Simulator 8085

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

GNUSim8085 is a 8085 microprocessor simulator with following features.

- A simple editor component with syntax highlighting.
- A keypad to input assembly language instructions with appropriate arguments.
- Easy view of register contents.
- Easy view of flag contents.
- Hexadecimal <--> Decimal converter.
- View of stack, memory and I/O contents.
- Support for breakpoints for programming debugging.
- Stepwise program execution.
- One click conversion of assembly program to opcode listing.
- Printing support (known not to work well on Windows).
- UI translated in various languages.

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 mention in the instruction set of microprocessor 8085.

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

3. Operands

4. Comments

Registers of 8085 microprocessor

→ A **microprocessor** is a multipurpose, programmable, clock-driven, register-based electronic device that reads binary instructions from a storage device called memory, accepts binary data as input and processes data according to those instructions and provide results as output.

→ A 8085 microprocessor, is a second generation 8-bit microprocessor and is the base for studying and using all the microprocessor available in the market.

Registers in 8085:

(a) General Purpose Registers –

→ The 8085 has six general-purpose registers to store 8-bit data; these are identified as- B, C, D, E, H, and L. These can be combined as register pairs – BC, DE, and HL, to perform some 16-bit operation. These registers are used to store or copy temporary data, by using instructions, during the execution of the program.

(b) Specific Purpose Registers –

→ **Accumulator:**

- The accumulator is an 8-bit register (can store 8-bit data) that is the part of the arithmetic and logical unit (ALU).
- After performing arithmetical or logical operations, the result is stored in accumulator. Accumulator is also defined as register A.

→ **Flag registers:**

B_7	B_6	B_5	B_4	B_3	B_2	B_1	B_0
S	Z	—	AC	—	P	—	CY

fig(a)-Bit position of various flags in flag registers of 8085

- The flag register is a special purpose register and it is completely different from other registers in microprocessor.
- It consists of 8 bits and only 5 of them are useful.
- The other three are left vacant and are used in the future Intel versions.
- These 5 flags are set or reset (when value of flag is 1, then it is said to be set and when value is 0, then it is said to be reset) after an operation according to data condition of the result in the accumulator and other registers.

The 5 flag registers are:

1. **Sign Flag:** It occupies the seventh bit of the flag register, which is also known as the most significant bit. It helps the programmer to know whether the number stored in the accumulator is positive or negative. If the sign flag is set, it means that number stored in the accumulator is negative, and if reset, then the number is positive.
2. **Zero Flag:**: It occupies the sixth bit of the flag register. It is set, when the operation performed in the ALU results in zero(all 8 bits are zero), otherwise it is reset. It helps in determining if two numbers are equal or not.
3. **Auxillary Carry Flag:** It occupies the fourth bit of the flag register. In an arithmetic operation, when a carry flag is generated by the third bit and passed on to the fourth bit,

then Auxillary Carry flag is set. If not flag is reset. This flag is used internally for BCD(Binary-Coded decimal Number) operations.
Note – This is the only flag register in 8085 which is not accessible by user.

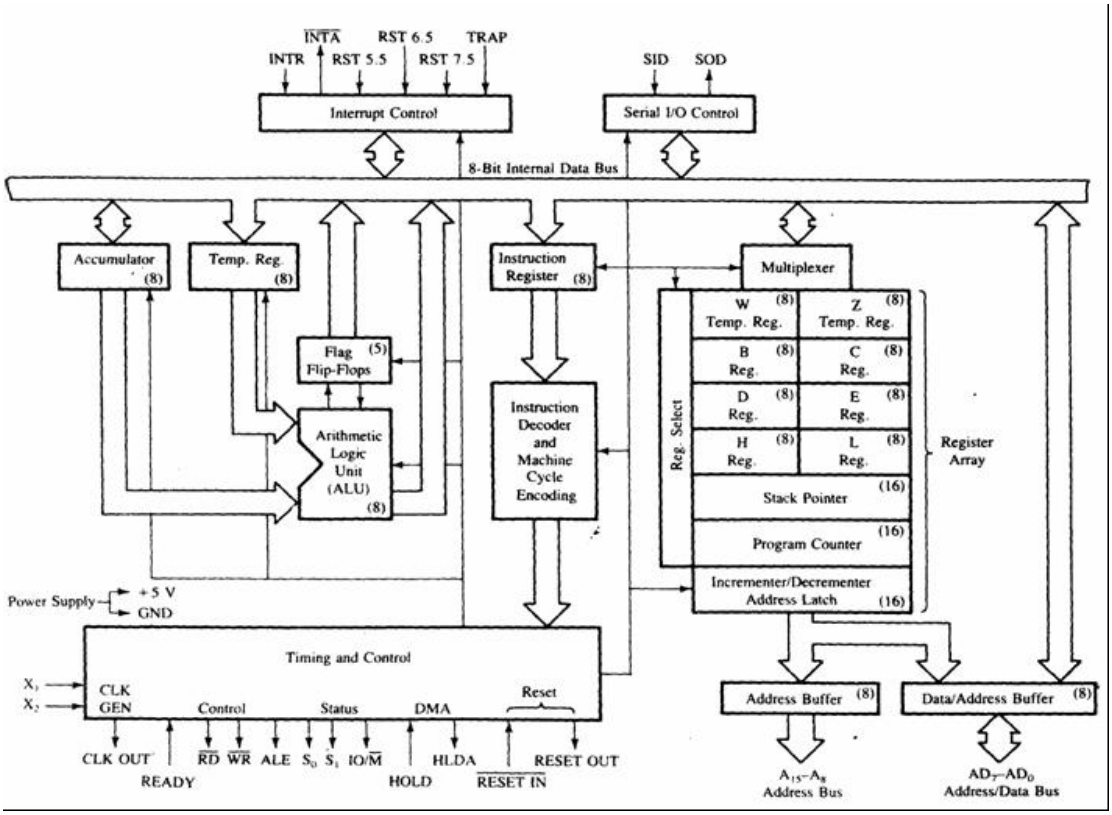
4. **Parity FlagL:** It occupies the second bit of the flag register. This flag tests for number of 1's in the accumulator. If the accumulator holds even number of 1's, then this flag is set and it is said to even parity. On the other hand if the number of 1's is odd, then it is reset and it is said to be odd parity.
5. **Carry Flag:** It occupies the zeroth bit of the flag register. If the arithmetic operation results in a carry(if result is more than 8 bit), then Carry Flag is set; otherwise it is reset.

(c) Memory Registers –

→ There are two 16-bit registers used to hold memory addresses. The size of these registers is 16 bits because the memory addresses are 16 bits. They are :-

- **Program Counter:** This register is used to sequence the execution of the instructions. The function of the program counter is to point to the memory address from which the next byte is to be fetched. When a byte (machine code) is being fetched, the program counter is incremented by one to point to the next memory location.
- **Stack Pointer:** It is used as a memory pointer. It points to a memory location in read/write memory, called the stack. It is always incremented/decremented by 2 during push and pop operation.

ARCHITECTURE OF MICROPROCESSOR 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 have 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.

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 different meaning when given to 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

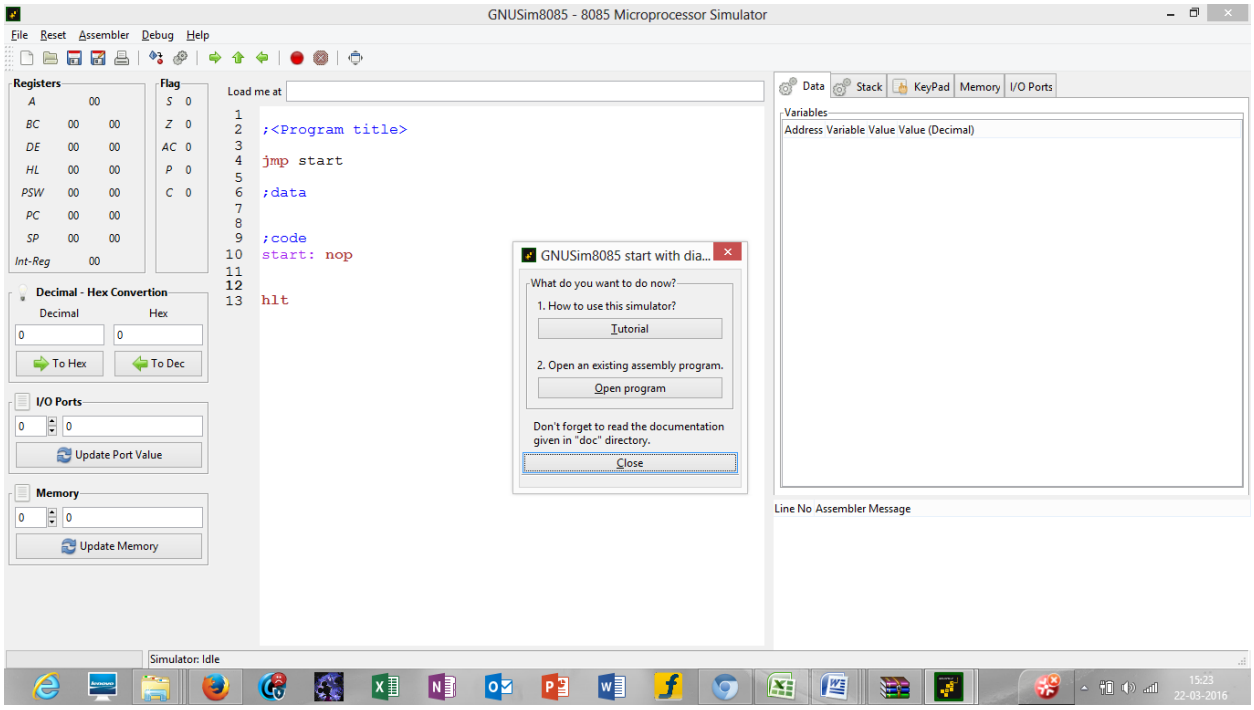
- DB is used to define space for an array of values specified by comma separated list. And the label (if given to the beginning of DB) is assigned the address of the first data item.
- DS is used to define the specified number of bytes to be assigned and initialize them to zero. To access each byte you can use the + or -operator along with label.

- EQU behaves similar to #define 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 operand. 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).

Getting Started with GNU Simulator 8085:



Pin Description

The following describes the function of each pin:

→ A6 - A1s (Output 3 State)

Address Bus; The most significant 8 bits of the memory address or the 8 bits of the I/0 address, 3 stated during Hold and Halt modes.

→ AD0 - 7 (Input/output 3state)

Multiplexed Address/Data Bus; Lower 8 bits of the memory address (or I/0 addresses) appear on the bus during the first clock cycle of a machine state. It then becomes the data bus during the second and third clock cycles. 3 stated during Hold and Halt modes.

→ ALE (Output)

Address Latch Enable: It occurs during the first clock cycle of a machine state and enables the address to get latched into the on chip latch of peripherals. The falling edge of ALE is set to guarantee setup and hold times for the address information.
ALE can also be used to strobe the status information. ALE is never 3stated.

→ SO, S1 (Output)

Data Bus Status. Encoded status of the bus cycle:
S1 S0
O HALT
0 1 WRITE
1 0 READ
1 1 FETCH
S1 can be used as an advanced R/W status.

→ RD (Output 3state)

READ; indicates the selected memory or 1/0 device is to be read and that the Data Bus is available for the data transfer.

→ WR (Output 3state)

WRITE; indicates the data on the Data Bus is to be written into the selected memory or 1/0 location. Data is set up at the trailing edge of WR. 3stated during Hold and Halt modes.

→ READY (Input)

If Ready is high during a read or write cycle, it indicates that the memory or peripheral is ready to send or receive data. If Ready is low, the CPU will wait for Ready to go high before completing the read or write cycle.

→ HOLD (Input)

HOLD; indicates that another Master is requesting the use of the Address and Data Buses.
The CPU, upon receiving the Hold request. will relinquish the use of buses as soon as the completion of the current machine cycle. Internal processing can continue.

The processor can regain the buses only after the Hold is removed. When the Hold is acknowledged, the Address, Data, RD, WR, and IO/M lines are 3stated.

→ HLDA (Output)

HOLD ACKNOWLEDGE; indicates that the CPU has received the Hold request and that it will relinquish the buses in the next clock cycle. HLDA goes low after the Hold request is removed. The CPU takes the buses one half clock cycle after HLDA goes low.

→ INTR (Input)

INTERRUPT REQUEST; is used as a general purpose interrupt. It is sampled only during the next to the last clock cycle of the instruction. If it is active, the Program Counter (PC) will be inhibited from incrementing and an INTA will be issued. During this cycle a **RESTART or CALL** instruction can be inserted to jump to the interrupt service routine. The INTR is enabled and disabled by software. It is disabled by Reset and immediately after an interrupt is accepted.

→ INTA (Output)

INTERRUPT ACKNOWLEDGE; is used instead of (and has the same timing as) RD during the Instruction cycle after an INTR is accepted. It can be used to activate the 8259 Interrupt chip or some other interrupt port.
RST 5.5
RST 6.5 - (Inputs)
RST 7.5
RESTART INTERRUPTS; These three inputs have the same timing as I NTR except they cause an internal RESTART to be automatically inserted.
RST 7.5 ~~ Highest Priority
RST 6.5
RST 5.5 o Lowest Priority
The priority of these interrupts is ordered as shown above. These interrupts have a higher priority than the INTR.

→ TRAP (Input)

Trap interrupt is a nonmaskable restart interrupt. It is recognized at the same time as INTR. It is unaffected by any mask or Interrupt Enable. It has the highest priority of any interrupt.

→ RESET IN (Input)

Reset sets the Program Counter to zero and resets the Interrupt Enable and HLDA flipflops. None of the other flags or registers (except the instruction register) are affected The CPU is held in the reset condition as long as Reset is applied.

→ RESET OUT (Output)

Indicates CPlJ is being reset. Can be used as a system RESET. The signal is synchronized to the processor clock.

→ **X1, X2 (Input)**
Crystal or R/C network connections to set the internal clock generator X1 can also be an external clock input instead of a crystal. The input frequency is divided by 2 to give the internal operating frequency.

→ **CLK (Output)**
Clock Output for use as a system clock when a crystal or R/ C network is used as an input to the CPU. The period of CLK is twice the X1, X2 input period.

→ **IO/M (Output)**
IO/M indicates whether the Read/Write is to memory or l/O Tristated during Hold and Halt modes.

→ **SID (Input)**
Serial input data line The data on this line is loaded into accumulator bit 7 whenever a RIM instruction is executed.

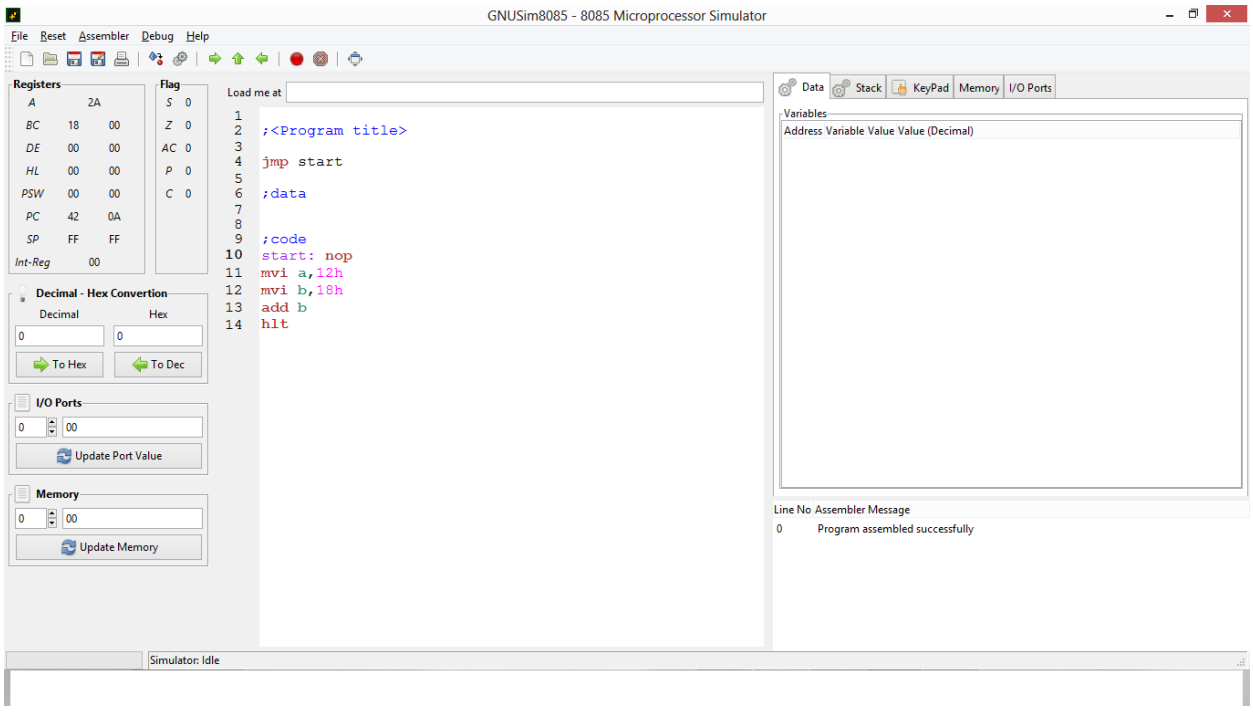
→ **SOD (output)**
Serial output data line. The output SOD is set or reset as specified by the SIM instruction.

→ **Vcc**
+5 volt supply.

→ **Vss**
Ground Reference.

Program: ADDITION OF TWO NUMBERS

Source Code:



Source Code:

```
lda var1
mov b,a
lda var2
add b
sta var3
hlt
var1: db 04h
var2: db 09h
var3: db 00h
```

Output:

Instruction and Data Formats

The various techniques to specify data for instructions are:

1. 8-bit or 16-bit data may be directly given in the instruction itself.
2. The address of the memory location, I/O port or I/O device, where data resides, may be given in the instruction itself.
3. In some instructions, only one register is specified. The content of the specified register is one of the operands.
4. Some instructions specify two registers. The contents of the registers are the required data.
5. In some instructions, data is implied. The most instructions of this type operate on the content of the accumulator.

Due to different ways of specifying data for instructions, the machine codes of all instructions are not of the same length. It may 1-byte, 2-byte or 3-byte instruction.

Addressing Modes

Each instruction requires some data on which it has to operate. There are different techniques to specify data for instructions. These techniques are called **addressing modes**. Intel 8085 uses the following addressing modes:

- **Direct Addressing**

In this addressing mode, the address of the operand (data) is given in the instruction itself.

Example

STA 2400H: It stores the content of the accumulator in the memory location 2400H.

32, 00, 24: The above instruction in the code form.

In this instruction, 2400H is the memory address where data is to be stored. It is given in the instruction itself. The 2nd and 3rd bytes of the instruction specify the address of the memory location. Here, it is understood that the source of the data is accumulator.

- **Register Addressing**

In register addressing mode, the operand is in one of the general purpose registers. The opcode specifies the address of the register(s) in addition to the operation to be performed.

Example:

MOV A, B: Move the content of B register to register A.

78: The instruction in the code form.

In the above example, MOV A, B is 78H. Besides the operation to be performed the opcode also specifies source and destination registers.

The opcode 78H can be written in binary form as 01111000. The first two bits, i.e. 0 1 are for MOV operation, the next three bits 1 1 1 are the binary code for register A, and the last three bits 000 are the binary code for register B.

- **Register Indirect Addressing**

In Register Indirect mode of addressing, the address of the operand is specified by a register pair.

Example

- **LXI H, 2500 H** - Load H-L pair with 2500H.
- **MOV A, M** - Move the content of the memory location, whose address is in H-L pair (i.e. 2500 H) to the accumulator.
- **HLT** - Halt.

In the above program the instruction MOV A, M is an example of register indirect addressing. For this instruction, the operand is in the memory. The address of the memory is not directly given in the instruction. The address of the memory resides in H-L pair and this has already been specified by an earlier instruction in the program, i.e. LXI H, 2500 H.

- **Immediate Addressing**

In this addressing mode, the operand is specified within the instruction itself.

Example

LXI H, 2500 is an example of immediate addressing. 2500 is 16-bit data which is given in the instruction itself. It is to be loaded into H-L pair.

- **Implicit Addressing**

There are certain instructions which operate on the content of the accumulator. Such instructions do not require the address of the operand.

Example

CMA, RAL, RAR, etc.

Status Flags

There is a set of five flip-flops which indicate status (condition) arising after the execution of arithmetic and logic instructions. These are:

- Carry Flag (CS)
- Parity Flag (P)
- Auxiliary Carry Flags (AC)
- Zero Flags (Z)
- Sign Flags (S)

Symbols and Abbreviations

The symbol and abbreviations which have been used while explaining Intel 8085 instructions are as follows:

Symbol/Abbreviations	Meaning
Addr	16-bit address of the memory location.
Data	8-bit data
data 16	16-bit data
r, r1, r2	One of the registers A, B, C, D, E, H or L
A, B, C, D, H, L	8-bit register
A	Accumulator
H-L	Register pair H-L
B-C	Register pair B-C
D-E	Register pair D-E
PSW	Program Status Word
M	Memory whose address is in H-L pair

H	Appearing at the end of the group of digits specifies hexadecimal, e.g. 2500H
Rp	One of the register pairs.
Rh	The high order register of a register pair
Rl	The low order register of a register pair
PC	16 bit program counter, PCH is high order 8 bits and PCL low order 8 bits of register PC.
CS	Carry Status
[]	The contents of the register identified within bracket
[[]]	The content of the memory location whose address is in the register pair identified within brackets
^	AND operation
v	OR operation
⊕ or ∀	Exclusive OR
←	Move data in the direction of arrow
⇔	Exchange contents