EXPERIMENT No. 1

1. **Title –**

   To understand the programming model of 8085 microprocessor by making programs for data transfer between registers and/or memory.

   Following is a list of subprograms –

   1) **[Concept illustration program]** - Write a program in ALP 8085 (on GNU 8085) to initialize the register B with 05H, then move the data to register C. Finally observe the result after executing the program.
   2) **[Concept illustration program]** - Write a program in ALP 8085 (on GNU 8085) to load a byte from address 3000H into register B, increment it by one and store the result back to location 3001H using HL pair as pointer register.
   3) **[Concept application program]** - Write a program in ALP 8085 (on GNU 8085) to load a byte from address 3000H into register B, increment it by one and store the result back to location 3001H using LDAX/STAX commands.

2. **Theoretical Background –**

   - **Accumulator (A)** - An 8-bit register used for arithmetic, logic, and data transfer operations.
   - **General-purpose registers** - B, C, D, E, H, and L, which can hold 8-bit data or be paired (BC, DE, HL) to handle 16-bit addresses or operations.
   - **Temporary registers** - Used internally and inaccessible to the programmer.
   - **Program Counter (PC)** - Holds the address of the next instruction to be executed.
   - **Stack Pointer (SP)** - Points to the top of the stack.

3. **Experimental Setup –**

   The current experiment is performed on GNU 8085 software emulator program.

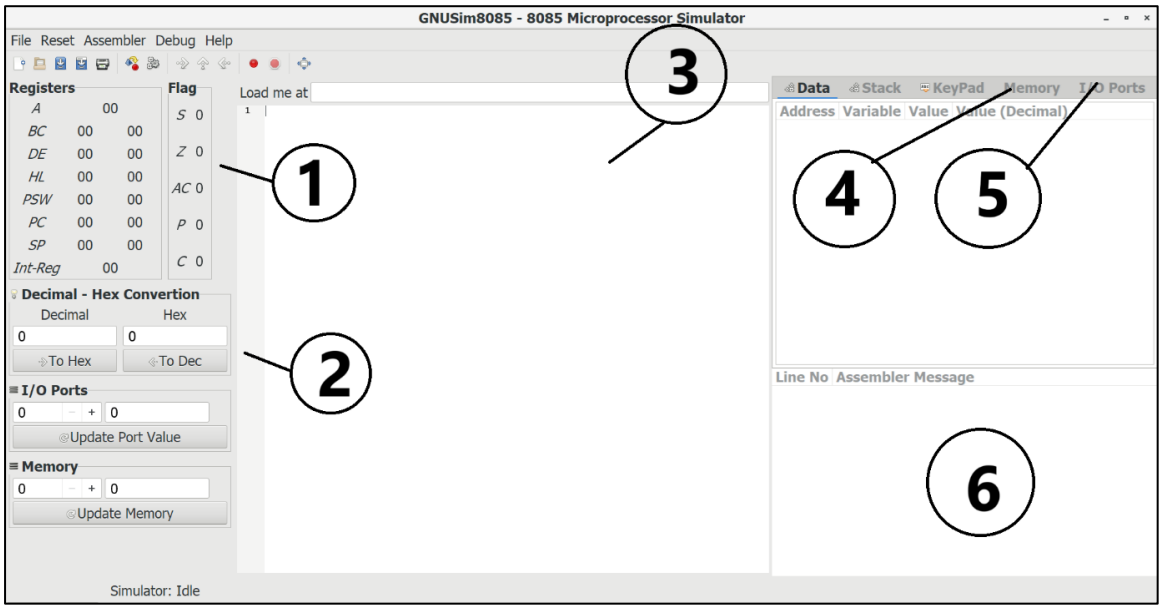
4. **Precautions-**

   Following precautions are to be taken while using GNU 8085 software-

   1) While using numbers that begin with a character (for example A1H, they must be written as 0A1H to ensure that the software understands them as hexadecimal number A1. For numbers thar begin with a digit, no such appending is required.
   2) PSW register does not display the data (it's a bug) so if it is required to see the data of PSW at any stage, we may push the data of PSW onto the stack and see it.
   3) PC register by default begins from 4200 H in this software but it does not display the first value correctly. So, it must be kept in mind.

5. Procedure

The interface/menu of GNU 8085 simulator is explained below -



Region 1 – Registers in 8085 – This region displays all the registers of the 8085 microprocessors accessible to the programmer, such as A, B, C, D, E, H, L, SP (Stack Pointer), and PC (Program Counter). Flags such as S (Sign), Z (Zero), AC (Auxiliary Carry), P (Parity), and C (Carry) are also shown here to indicate the state of the processor.

Region 2 – DEC to HEX and vice versa conversion menu – This region provides a convenient tool to convert numbers between Decimal and Hexadecimal formats. Users can input values in either format and use the provided buttons to perform the conversion.

Region 3 – Code Entry Area - This is the primary area where the programmer can write assembly language code for the 8085 microprocessors. The code editor allows users to input and edit instructions line by line.

Region 4 – Memory View - This region shows the memory map of the 8085 microprocessors. It displays the memory addresses, variables stored at those addresses, and their corresponding values in both hexadecimal and decimal formats. This is useful for monitoring and updating memory content during program execution or debugging.

Region 5 – Stack - This region represents the stack memory in the 8085 microprocessors. It displays the current stack pointer (SP) value and allows users to monitor and manipulate data stored in the stack during program execution. The stack is used for temporary storage, especially during function calls or interrupts.

Region 6 – Assembler Messages - This area displays the assembler messages, including line numbers and error descriptions, if any. It helps debug the code by showing errors, warnings, or success messages related to the written assembly cod

6. **Code**

Code for concept illustration program 1		
Write a program in ALP 8085 (on GNU 8085) to initialize the register B with 05H, then move the data to register C. Finally observe the result after executing the program.		
S. No.	**Code**	**Explanation**
1	MVI B,05H	These line stores the value 05H in B register
2	MOV C,B	It copies the data of B register to C register
3	HLT	Stops the execution of the program.

Code for concept illustration program 2		
Write a program in ALP 8085 (on GNU 8085) to load a byte from address 3000H into register B, increment it by one and store the result back to location 3001H using HL pair as pointer register.		
S. No.	**Code**	**Explanation**
1	LXI H,3000H	Load the HL register pair with the memory address 3000H.
2	MOV B,M	Move the value at memory location 3000H (pointed by HL) into register B.
3	INR B	Increment the value in register B by 1.
4	INX H	Increment the HL pair, so HL now points to the next memory location (3001H).
5	MOV M,B	Store the incremented value from register B into the memory location pointed to by HL (3001H).
6	HLT	Stops the execution of the program.

Code for concept illustration program 3		
Write a program in ALP 8085 (on GNU 8085) to load a byte from address 3000H into register B, increment it by one and store the result back to location 3001H using LDAX/STAX commands.		
S. No.	**Code**	**Explanation**
1	LXI D,3000H	Load the DE register pair with the memory address 3000H.
2	LDAX D	Load the value from memory location 3000H (address in DE) into accumulator A.
3	MOV B, A	Move the value in the accumulator (A) to register B.
4	INR B	Increment the value in register B by 1.
7	MOV A, B	Move the incremented value from register B back to accumulator A.
8	INX D	Increment the DE register pair. DE now points to the next memory location (3001H).
9	STAX D	Store the value in accumulator A into the memory location pointed to by DE (3001H).
10	HLT	Stops the execution of the program.

7. **Observations / Calculations / Results –**
   Necessary observations were made and program correctly executed.

8. **Conclusion –**
   The experiment demonstrated data transfer between registers and memory in the 8085 microprocessor, providing a clear understanding of its programming model and basic operations.