Programming Model (8085)

Introduction to 8085 Programming Model

The 8085 programming model serves as the foundation for developing assembly language programs for the 8085 microprocessor. This model defines the architecture and organization of key elements that contribute to the execution of instructions and data manipulation.

- Utilizing the 8085 programming model is essential for crafting efficient programs as the programming model provides a structured framework for organizing and manipulating data, allowing programmers to implement logical and arithmetic operations effectively.
- The programming model of 8085 provides crucial information necessary for writing assembly language programs.
- This model comprises six registers, including one accumulator, and one flag register.
- It features two 16-bit registers for addressing: the stack pointer and the program counter.

Accumulator A (8)	Flag Register F (8)
B (8)	C (8)
D (8)	E (8)
H (8)	L (8)
SP (16)	
PC (16)	

Programming Model 8085

- **General-Purpose Registers:** The 8085 microprocessor includes six 8-bit general-purpose registers – B, C, D, E, H, and L. These registers can be paired up to form three 16-bit register pairs: (B, C), (D, E), and (H, L). This allows the storage of data larger than 8 bits by combining the values of two registers.
- **Special-Purpose Registers:**
    - **A (Accumulator):** The accumulator is a crucial register where all arithmetic and logical operations take place. It serves as the default location for storing one of the operands during these operations, and the result is also stored in the accumulator.
    - **SP (Stack Pointer):** The stack pointer is a 16-bit register that holds the address of the top of the stack. After each push or pop operation, the stack pointer's

holds the address of the next instruction to be fetched from memory. As each instruction is fetched, the program counter's value is automatically incremented by 1, ensuring sequential execution of instructions.

Flag Register

The flag register is a vital component in the 8085 programming model, providing information about the status and conditions of the processor.

- The flag register consists of five flip-flops at positions D0, D2, D4, D6, and D7. Each flip-flop can either be set (1) or reset (0), indicating specific conditions or states.

S	Z		AC		P		CY
D7	D6	D5	D4	D3	D2	D1	D0

- **S (Sign Flag):** Indicates the sign of the result in the accumulator.
    - If 1, the answer in the accumulator is negative.
    - If 0, the number in the accumulator is positive.
- **Z (Zero Flag):** Identifies whether the answer in the accumulator is zero or non-zero.
    - If 1, the answer in the accumulator is 0.
    - If 0, the answer in the accumulator is non-zero.
- **AC (Auxiliary Carry Flag):** Reflects the presence of an auxiliary carry in intermediate positions during arithmetic operations. It works in conjunction with the Carry Flag (CY), which indicates the final carry.
    - If 1, there is an auxiliary carry.
    - If 0, there is no intermediate carry.
- **P (Parity Flag):** Checks for parity, where odd parity has an odd number of 1s, and even parity has an even number of 1s.
    - If 1, the binary representation has even parity (e.g., 1010).
    - If 0, the binary representation has odd parity (e.g., 11001).
- **CY (Carry Flag):** Indicates the final carry resulting from arithmetic operations.
    - If 1, there is a final carry.
    - If 0, there is no final carry.

Numerical 1:
Assume A (Accumulator) = 88H & B = 99H

```
1 as final carry      1 as auxiliary carry
      1                      1
          1 0 0 0     1 0 0 0      (99H)
          1 0 0 1     1 0 0 1      (88H)
         ____________________
          0 0 1 0     0 0 0 1
         |___________________|
          Data in Accumulator = 21H
```

S = 0 as answer is not negative
Z = 0 as answer is not zero
AC = 1 as there was a auxiliary carry
P = 1 as answer is even parity
Cy = 1 as there was a final carry

Numerical 2:
A = 13H and B = 24H
determine the status of all flags when following 8085 instruction is executed: SUB B (A ← A - B).

```
    0 0 0 1     0 0 1 1       (13H)
    0 0 1 0     0 1 0 0       (24H)

    2's complement of 0010 0100 (24H) = 1101 1100

    0 0 0 1     0 0 1 1      (13H)
  + 1 1 0 1     1 1 0 0      (2's comp of 24H)
   ____________________
    1 1 1 0     1 1 1 1      (-11H)
```

For human understanding we can convert this to HEX
but before that we have to find its 2's compliment.
so 2's compliment of 1110 1111 = 11H (in negative but)

Flag register will have flag value according to the content stored in accumulator
S = 1 as answer is negative
Z = 0 as answer is not zero
AC = 0 as there was no auxiliary carry

Memory Mapped I/O and Peripheral Mapped I/O in 8085 Microprocessor

In the world of microprocessor architecture, efficient communication between the processor and external devices is essential for system functionality. Two key concepts that facilitate this communication in the 8085 microprocessor are memory mapped I/O and peripheral mapped I/O. These techniques play a crucial role in interfacing the microprocessor with memory devices, input/output (I/O) devices, and other peripherals. Memory mapped I/O involves treating I/O devices as if they were memory locations, memory address space, requiring dedicated I/O instructions for communication. Understanding these concepts is fundamental for designing efficient systems and programming interfaces in 8085-based applications.

Memory Mapped I/O

- Definition: Memory mapped I/O is a technique where the microprocessor treats I/O devices as if they were memory locations. This means that I/O devices are assigned specific memory addresses, and accessing these addresses allows the microprocessor to communicate with the devices.
- Usage: In the 8085 microprocessor, memory mapped I/O is used to access I/O devices by mapping them to specific memory addresses. This simplifies programming as I/O operations are performed using standard memory read and write instructions.
- Addressing: Memory mapped I/O uses memory addresses to access I/O devices, making them appear as part of the memory address space.
- Example: Accessing an input device (e.g., keyboard) or an output device (e.g., display) through memory addresses mapped to specific I/O ports.

Example:

Reading from an input device mapped to a memory address:

```
MOV A, M  ; Move data from memory (input device address) to accumulator
```

Writing to an output device mapped to a memory address:

```
MOV M, A  ; Move data from accumulator to memory (output device address)
```

Peripheral Mapped I/O:

- Definition: Peripheral mapped I/O involves assigning specific addresses to I/O ports separate from the memory address space. Each I/O port is assigned a unique address, allowing the microprocessor to communicate with external devices connected to these ports.
- Usage: In the 8085 microprocessor, peripheral mapped I/O is used to access external devices through dedicated I/O ports. Instructions and signals specific to peripheral devices are utilized for data transfer, control, and communication, enabling seamless interaction between the microprocessor and external hardware components. This method allows the microprocessor to manage input and output operations efficiently, supporting various applications in embedded systems, industrial automation, and computer peripherals interfacing.
- Addressing: Peripheral mapped I/O uses separate port addresses for each I/O device, distinct from memory addresses.

Example:

Reading from an input device connected to an I/O port:

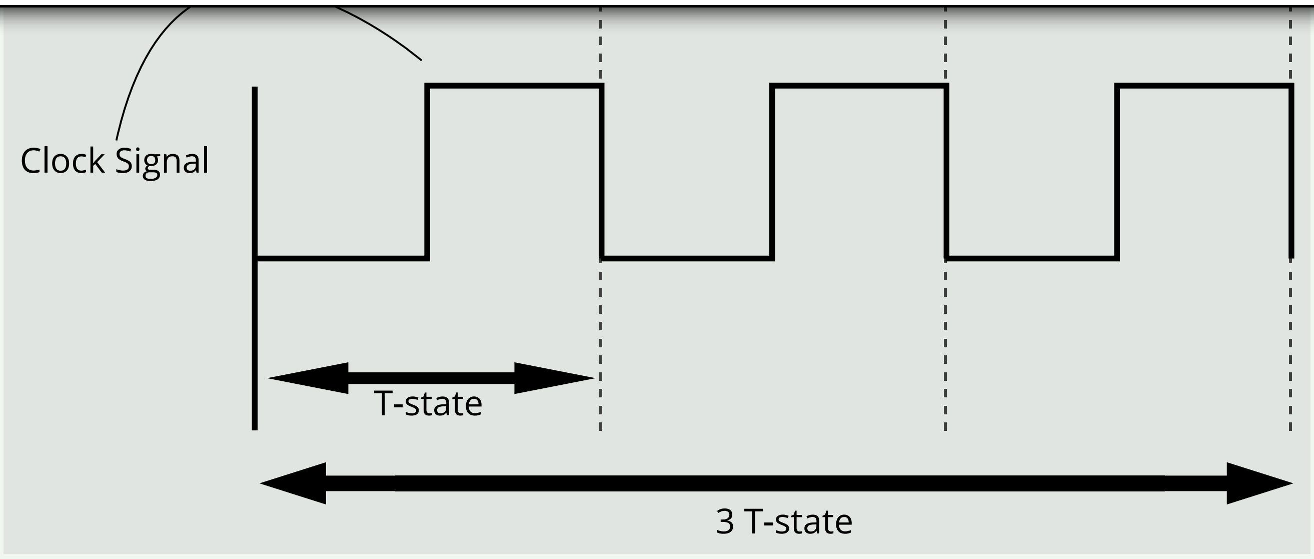
```
IN 30H  ; Input data from port address 30H into accumulator
```

Writing to an output device connected to an I/O port:

```
OUT 40H  ; Output data from accumulator to port address 40H
```

Machine Cycle

- The machine cycle refers to the time required to complete one operation of accessing memory or an I/O port, acknowledging an external request, or performing a specific task within the microprocessor.
- Various operations are encompassed within the machine cycle, including:
  - **OP Code Fetch:** Fetching the operation code (OP code) from memory to execute an instruction.
  - **Memory Read (MEMR):** Reading data from a memory location.
  - **Memory Write (MEMW):** Writing data to a memory location.
  - **I/O Read:** Reading data from an input/output (I/O) port.
  - **I/O Write:** Writing data to an I/O port.
  - **Interrupt Acknowledge:** Acknowledging an interrupt request from external devices.
  - **Control Signals:** Generating and managing control signals such as clock pulses, enable signals, and timing signals.
- The machine cycle is crucial in determining the overall performance and efficiency of a microprocessor-based system. It sets the pace for executing instructions, processing data, and handling input/output operations.
- A machine cycle is composed of multiple T-states, where a T-state represents a single clock cycle or the time duration of a clock signal.
- Understanding the machine cycle and its components is essential for designing efficient microprocessor systems, optimizing instruction execution, and ensuring accurate timing in data processing and communication.



Note:

- All machine cycles such as $\overline{\text{MEMR}}$, $\overline{\text{MEMW}}$, I/O Read, and I/O Write typically require 3 T-states to complete their operations. However, the OP code fetch cycle is an exception, taking 4 T-states to complete.
- The OP code fetch cycle includes an additional T-state for decoding the OP code, extending its duration to 4 T-states.
- For example, let's consider the instruction cycle, which encompasses the complete process of fetching an instruction, decoding its OP code, and executing it. This cycle's duration is determined by the number of machine cycles required for the specific instruction.
- During the 4 T-state OP code fetch cycle, the first three T-states are typically used for fetching the instruction's address from memory, and the fourth T-state is dedicated to decoding the OP code retrieved in the previous T-states.
- On the other hand, in a 3 T-state machine cycle such as I/O Read or I/O Write, each T-state is utilized for specific tasks like initiating the I/O operation, transferring data, and completing the operation within the cycle's duration.

Determine the machine cycles and total T-states required for execution of the following instructions:

- MOV A, B: only op code fetch (4T) (1m cycle) as there is no memory access or additional processing involved. This instruction moves data from register B to register A.
- MOV A, M: Op fetch (4T) + MR (3T) (2m cycle) where MR stands for Memory Read. This instruction involves fetching the op code, then reading data from the memory location addressed by the HL pair and moving it to register A.
- MVI B, 32H: Op fetch (4T) + Data Write (3T) (2m cycle) as it involves fetching the op code and writing immediate data (32H) to register B.

- LDA 300H: Op fetch (4T) + MR (3T) (2m cycle) where MR stands for Memory Read. This instruction involves fetching the op code and loading accumulator A with data from memory address 300H.
- STA 3050H: Op fetch (4T) + MW (3T) (2m cycle) where MW stands for Memory Write. This instruction involves fetching the op code and storing accumulator A data into memory address 3050H.
- IN 00h: Op fetch (4T) + Input (3T) (2m cycle) as it involves fetching the op code and inputting data from input port 00h.
- OUT FFh: Op fetch (4T) + Output (3T) (2m cycle) as it involves fetching the op code and outputting data to output port FFh.
- LDAX B: Op fetch (4T) + MR (3T) (2m cycle) where MR stands for Memory Read. This instruction involves fetching the op code and loading accumulator A with data from the address pointed to by register pair BC.
- STA X D: Op fetch (4T) + MW (3T) (2m cycle) where MW stands for Memory Write. This instruction involves fetching the op code and storing accumulator A data into the address pointed to by register pair XD.