



Unit – 2

Software Architecture and Programming

Instruction Set & Execution of 8085

The Intel 8085 microprocessor is an 8-bit microprocessor with a well-defined instruction set architecture (ISA) that defines the set of instructions it can execute. The 8085-instruction set comprises a variety of instructions for performing operations like data movement, arithmetic and logic operations, branching, and control operations. Each instruction is represented by an opcode, and the operands may include registers, memory addresses, or immediate values. The execution of these instructions involves fetching the instruction, decoding, executing the operation, and updating the processor's state accordingly.

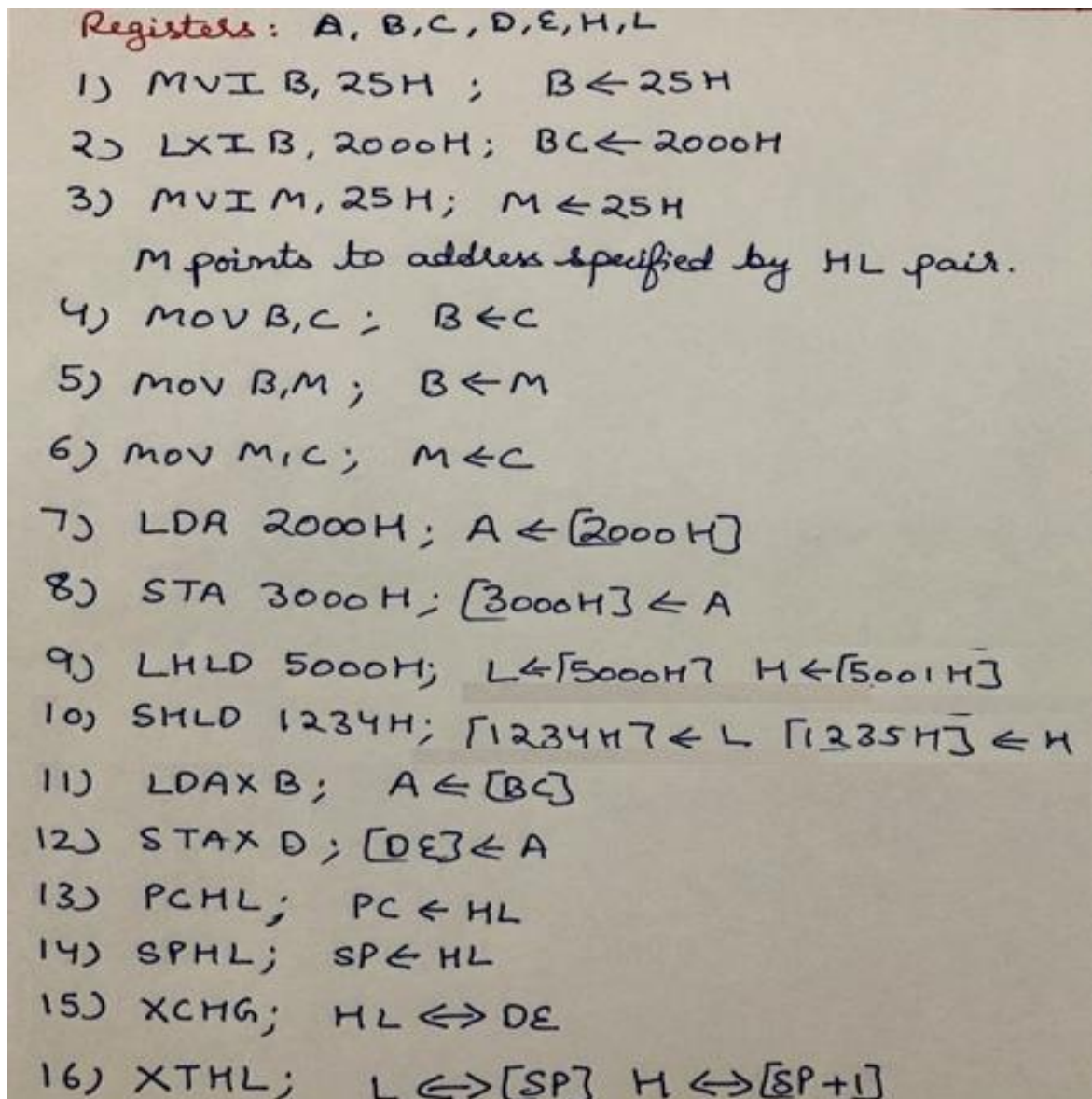
Execution of 8085 Instructions:

1. **Fetch:** The 8085 fetches the next instruction from memory. The program counter (PC) is used to keep track of the memory location containing the next instruction. The PC is incremented after each fetch.
2. **Decode:** The fetched instruction is decoded to determine the operation to be performed and the operands involved. This is done by examining the opcode and addressing modes.
3. **Execute:** The instruction is executed according to its operation. This may involve various actions, such as data movement, arithmetic or logic operations, branching, or control operations.
4. **Update:** After execution, the processor updates its registers, flags, and memory as necessary. For example, the accumulator may hold the result of an arithmetic operation, and flags (e.g., carry, zero) may be updated based on the result.
5. **Repeat:** The cycle of fetch, decode, execute, and update continues with the next instruction in memory until a halt (HLT) instruction or branch instruction causes a change in program flow.



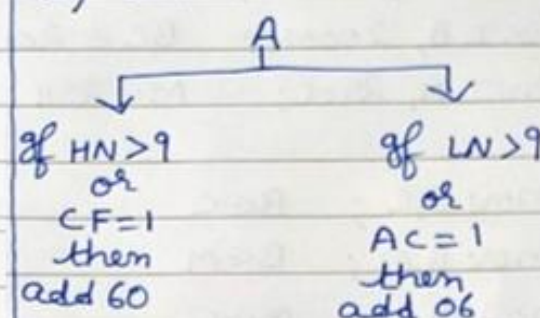
1. Data Transfer Instructions

Data transfer instructions in the 8085 microprocessor involves moving data between various registers, memory locations, and input/output ports. These instructions are essential for manipulating data within the processor and interfacing with external devices. Immediate data refers to constant values or literals that are directly specified within the instruction. **These instructions do not affect the Flag register.**



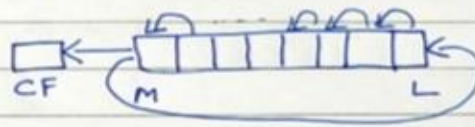
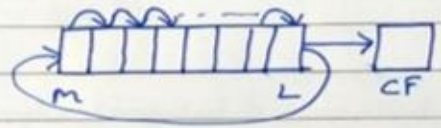
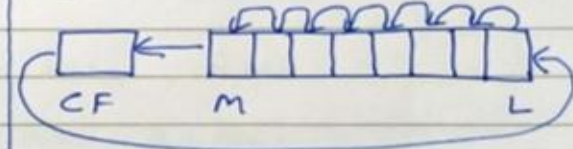
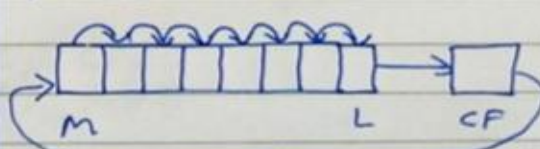
2. Arithmetic instructions

Arithmetic instructions in the 8085 microprocessor are operations that perform arithmetic operations on data stored in registers and memory locations. These instructions are fundamental for executing mathematical computations within the processor. After executing arithmetic instructions, the 8085 microprocessor updates flags in the flag register to indicate the result of the operation. These flags are crucial for decision-making in subsequent program execution.

<ol style="list-style-type: none"> ① ADD B; $A \leftarrow A+B$ ② ADD M; $A \leftarrow A+M$ ③ ADI 25H; $A \leftarrow A+25H$ ④ SUB B; $A \leftarrow A-B$ ⑤ SUB M; $A \leftarrow A-M$ ⑥ SUI 25H; $A \leftarrow A-25H$ 7) ADC B; $A \leftarrow A+B+CF$ 8) ADC M; $A \leftarrow A+M+CF$ 9) ACI 25H; $A \leftarrow A+25H+CF$ 10) SBB B; $A \leftarrow A-B-CF$ 11) SBB M; $A \leftarrow A-M-CF$ 12) SBI 25H; $A \leftarrow A-25H-CF$ 13) INR B; $B \leftarrow B+1$ 14) INX B; $BC \leftarrow BC+1$ 15) INRM; $M \leftarrow M+1$ 16) DCR B; $B \leftarrow B-1$ 17) DCX B; $BC \leftarrow BC-1$ 18) DCRM; $M \leftarrow M-1$ 19) DAD D; $HL \leftarrow HL+DE$ 	<p>20) DAA</p> <div style="text-align: center;">  </div> <table style="margin-top: 20px;"> <tr> <td style="text-align: right;">20H</td> <td style="text-align: right;">24H</td> <td style="text-align: right;">25H</td> </tr> <tr> <td style="text-align: right;">+30H</td> <td style="text-align: right;">+34H</td> <td style="text-align: right;">+35H</td> </tr> <tr> <td style="text-align: right; border-top: 1px solid black;">50H</td> <td style="text-align: right; border-top: 1px solid black;">58H</td> <td style="text-align: right; border-top: 1px solid black;">5AH</td> </tr> </table> <table style="margin-top: 20px;"> <tr> <td style="text-align: right;">26H</td> <td style="text-align: right;">28H</td> </tr> <tr> <td style="text-align: right;">+26H</td> <td style="text-align: right;">+28H</td> </tr> </table> <table style="margin-top: 20px;"> <tr> <td style="text-align: right;">40H</td> <td style="text-align: right;">50H</td> <td style="text-align: right;">80H</td> <td style="text-align: right;">99H</td> </tr> <tr> <td style="text-align: right;">+50H</td> <td style="text-align: right;">+50H</td> <td style="text-align: right;">+80H</td> <td style="text-align: right;">+99H</td> </tr> </table>	20H	24H	25H	+30H	+34H	+35H	50H	58H	5AH	26H	28H	+26H	+28H	40H	50H	80H	99H	+50H	+50H	+80H	+99H
20H	24H	25H																				
+30H	+34H	+35H																				
50H	58H	5AH																				
26H	28H																					
+26H	+28H																					
40H	50H	80H	99H																			
+50H	+50H	+80H	+99H																			

3. Logical Instructions

Logical instructions in the 8085 microprocessor are operations that perform logical operations on data stored in registers and memory locations. These instructions are fundamental for executing logical operations within the processor. After executing arithmetic instructions, the 8085 microprocessor updates flags in the flag register to indicate the result of the operation.

AND (clear)	Rotates
1) $ANA\ B; A \leftarrow A \wedge B$ 2) $ANA\ M; A \leftarrow A \wedge M$ 3) $ANI\ 25H; A \leftarrow A \wedge 25H$	1) RLC 
OR (set) 1) $ORA\ B;$ 2) $ORA\ M;$ 3) $ORI\ 25H$	2) RRC 
XOR (complement) 1) $XRA\ B;$ 2) $XRA\ M;$ 3) $XRI\ 25H$	3) RAL 
Compare 1) $CMP\ B\ (A-B)$ $A > B$ $A = B$ $A < B$ 2) $CMP\ M$ 3) $CPI\ 25H$ * $STC\ CF \leftarrow 1$ * $CMC\ CF \leftarrow \overline{CF}$ * $CMA\ A \leftarrow \overline{A}$	4) RAR 



4. Branch Instructions

Opcode			Operand	Meaning	Explanation
JMP			16-bit address	Jump unconditionally	The program sequence is transferred to the memory address given in the operand.
Opcode	Description	Flag Status	16-bit address	Jump conditionally	The program sequence is transferred to the memory address given in the operand based on the specified flag of the PSW.
JC	Jump on Carry	CY = 1			
JNC	Jump on no Carry	CY = 0			
JP	Jump on positive	S = 0			
JM	Jump on minus	S = 1			
JZ	Jump on zero	Z = 1			
JNZ	Jump on no zero	Z = 0			
JPE	Jump on parity even	P = 1			
JPO	Jump on parity odd	P = 0			

CALL Instructions

Opcode	Description	Flag Status	16-bit address	Unconditional subroutine call	The program sequence is transferred to the memory address given in the operand. Before transferring, the address of the next instruction after CALL is pushed onto the stack.
CC	Call on Carry	CY = 1			
CNC	Call on no Carry	CY = 0			
CP	Call on positive	S = 0			
CM	Call on minus	S = 1			
CZ	Call on zero	Z = 1			
CNZ	Call on no zero	Z = 0			
CPE	Call on parity even	P = 1			
CPO	Call on parity odd	P = 0			



Return Instructions

RET			None	Return from subroutine unconditionally	The program sequence is transferred from the subroutine to the calling program.
Opcode	Description	Flag Status	None	Return from subroutine conditionally	The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW and the program execution begins at the new address.
RC	Return on Carry	CY = 1			
RNC	Return on no Carry	CY = 0			
RP	Return on positive	S = 0			
RM	Return on minus	S = 1			
RZ	Return on zero	Z = 1			
RNZ	Return on no zero	Z = 0			
RPE	Return on parity even	P = 1			
RPO	Return on parity odd	P = 0			



Addressing Modes of 8085

Addressing Modes are the different ways by which the μP specifies the operands in an instruction. 8085 supports the following Addressing Modes:

1. Immediate Addressing Mode: Data is specified in the **Instruction** itself. Eg:

MVI A, 35H ; Move immediately the value 35 into the Accumulator.
; i.e., $A \leftarrow 35H$
LXI B, 4000H ; Move immediately the value 4000 into the register pair BC.
; i.e., $BC \leftarrow 4000H$

Advantage: Programmer can easily **identify** the **operands**.

Disadvantage: Always more than one byte hence requires **more space**.
 μP requires **2 or 3 machine cycles** to fetch the instruction hence **slow**.

2. Register Addressing Mode: Data is specified in the **Instruction** itself. Eg:

MOV B, C ; Move the Contents of C-Register into B-Register.
; i.e. $B \leftarrow C$
INR B ; Increments the contents of B Register.
; i.e. $B \leftarrow B+1$

Advantage: The μP requires **only one machine cycle** to fetch the instruction.

Disadvantage: Operands **cannot** be easily **identified**.

3. Direct Addressing Mode: Address of the operand is specified in the **Instruction** itself.

LDA 2000H ; Loads the Accumulator with the Contents of Location 2000H.
; i.e. $A \leftarrow [2000]$
STA 2000H ; Stores the Contents of the Accumulator at the Location 2000H.
; i.e. $[2000] \leftarrow A$

Advantage: The programmer **can identify** the address of the operand.



4. Indirect Addressing Mode: Address of the operand is specified in **Registers**. Hence, the instruction indirectly points to the operands. Even the Memory Pointer "M" can be used as it is pointed by the HL register pair.

STAX B ; Stores the contents of the Accumulator at the location pointed by the contents of BC pair; i.e. $[[BC]] \leftarrow A$.
; So, if contents of BC pair 4000 i.e. $[BC] = 4000$ then $[4000] \leftarrow A$

INR M ; Increments the contents of the location pointed by HL pair
; (i.e., M) i.e. $[[HL]] \leftarrow [[HL]] + 1$

Advantage: Address of the operand is **not fixed** and hence can be used in a **loop**.
Size of the instruction is **small** as compared to direct addressing mode.

Disadvantage: **Requires initialization** of the register pair hence requires at least one more instruction.

5. Implied Addressing Mode: **Operand** is **implied** in the instruction. This instruction work only on that implied operand, and not on any other operand. Eg:

STC ; Sets the Carry Flag in the Flag register.
; $Cy \leftarrow 1$.

CMC ; Complements the Carry Flag in the Flag register.

Advantage: Instructions are generally **only one byte**.

Disadvantage: Programmer **cannot** easily **identify** the value of the operand.



Machine Cycles & Timing Diagram

Instruction Cycle:

This is the time required by the μP to fetch and execute one complete instruction. The instruction cycle is in two parts:

1. Fetch Cycle
2. Execute Cycle

Fetch Cycle:

This is the time required by the μP to fetch all bytes of an instruction. The length of the fetch cycle is thus determined by the number of bytes in an instruction.

Execution Cycle:

This is the time required by the μP to execute a fetched instruction.

T-State:

T-State is one clock cycle of the μP . Therefore, $T = \text{Clock Period} = 1/\text{Clock Frequency}$.

Machine Cycle:

It is the time required by the μP doing one operation and accessing one byte from the external module (Memory or I/O).

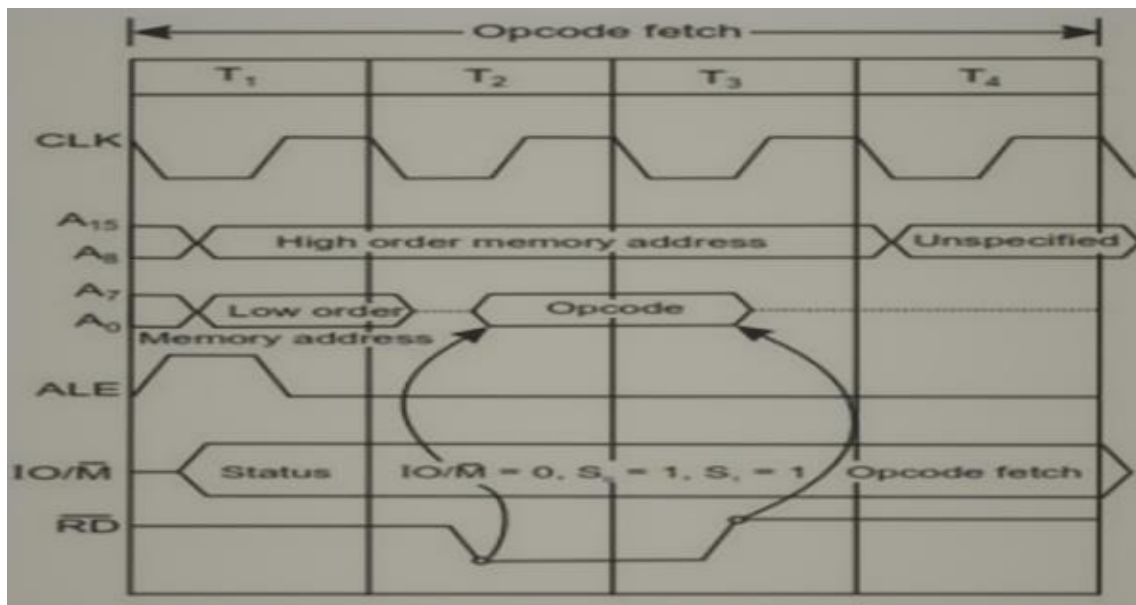
Machine Cycles of 8085

The Machine cycles of 8085 are given below:

Name	IO/M	RD	WR	S1	S0	INTA	T-States
Opcode Fetch	0	0	1	1	1	1	4/6
Mem Read	0	0	1	1	0	1	3
Mem Write	0	1	0	0	1	1	3
IO Read	1	0	1	1	0	1	3
IO Write	1	1	0	0	1	1	3
Int. Acknowledge	1	1	1	1	1	0	3 or 6
Bus Idle	0	1	1	0	0	1	3

Timing diagram of Opcode Fetch

This cycle is used to fetch the Opcode from the memory. This is the First Machine Cycle of every instruction. It is a compulsory Machine Cycle. It is generally of 4 T-States but some instructions require a 6 T-State Opcode Fetch.



During T1

- A15-A8 contains the higher byte of the address (PCH).
- As ALE is high AD7-AD0 contains the lower byte of the address (PCL).
- Since it is an Opcode fetch cycle, S1 and S0 goes high.
- Since it is a memory operation, IO/M goes low.

During T2

- As ALE goes low address is removed from AD7-AD0.
- As RD goes low, data appears on AD7-AD0.
- The μP examines the state of the READY pin. If it is low then the μP enters wait-state by executing wait cycles and remains in the wait-state until READY goes high.

During T3

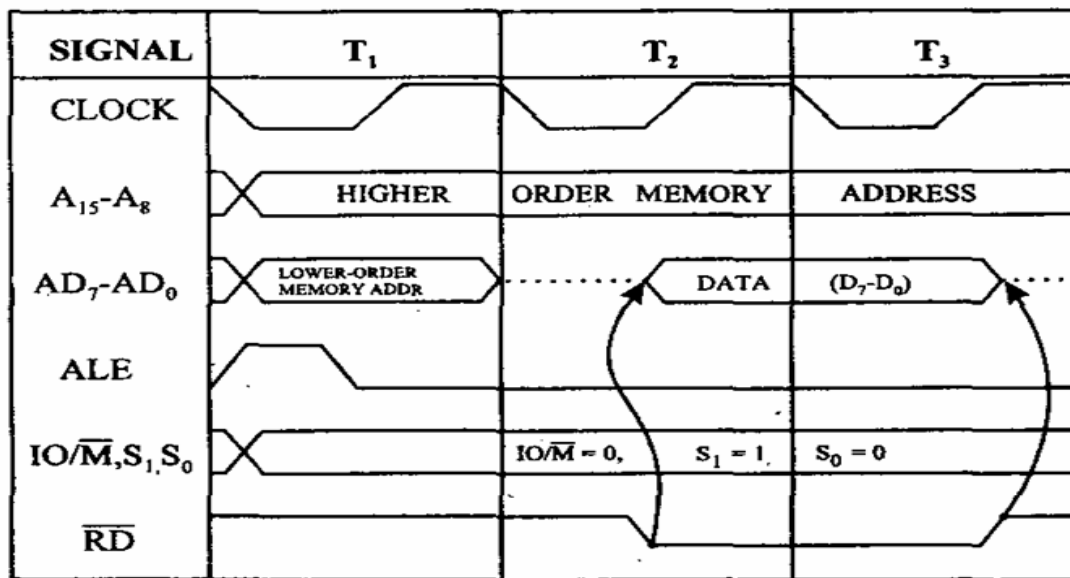
Data remains on AD7-AD0 till RD is low.

During T4

T4 state is used by the μP to decode the Opcode.

Timing diagram of Memory Read

This cycle is used to fetch **one byte from the memory**. This cycle can be used to fetch the operand bytes of an instruction or any data from the memory. It is not a compulsory Machine Cycle. It requires **3 T-States**.



During T₁

- A₁₅-A₈ contains the higher byte of the address (PCH).
- As ALE is **high** AD₇-AD₀ contains the lower byte of the address (PCL).
- Since it is a Memory Read cycle, S₁ goes high.
- Since it is a memory operation, IO/ \overline{M} goes low.

During T₂

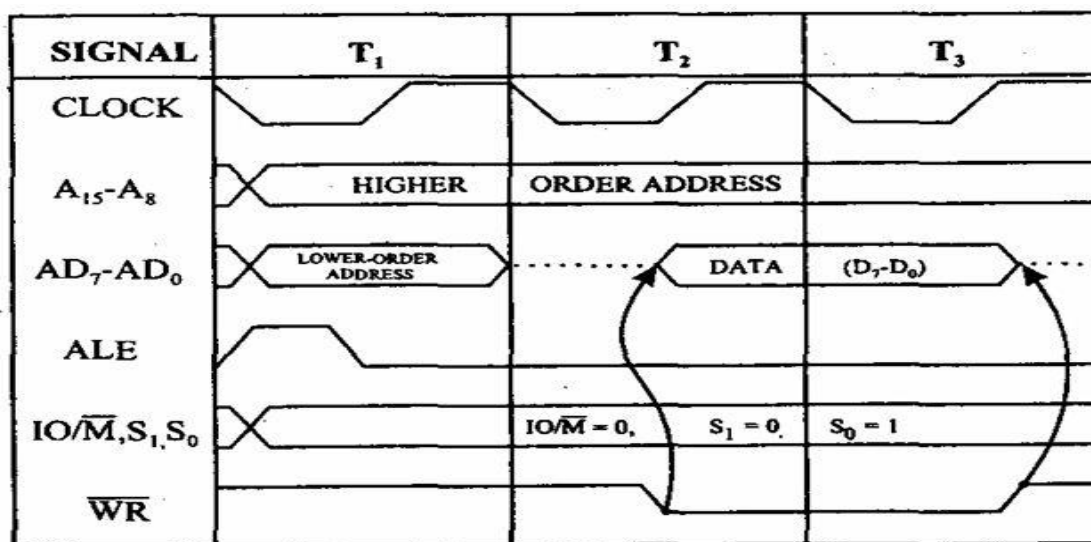
- As ALE goes low, address is removed from AD₇-AD₀.
- As \overline{RD} goes low, data appears on AD₇-AD₀.

During T₃

Data remains on AD₇-AD₀ till \overline{RD} is low.

Timing diagram of Memory Write

This cycle is used to send (write) **one byte into the memory**. It is not a compulsory Machine Cycle. It requires **3 T-States**.



During T₁

- A₁₅-A₈ contains the higher byte of the address (**PCH**).
- As ALE is **high** AD₇-AD₀ contains the lower byte of the address (**PCL**).
- Since it is a Memory Write cycle, **S₀** goes high.
- Since it is a memory operation, IO/ \overline{M} goes low.

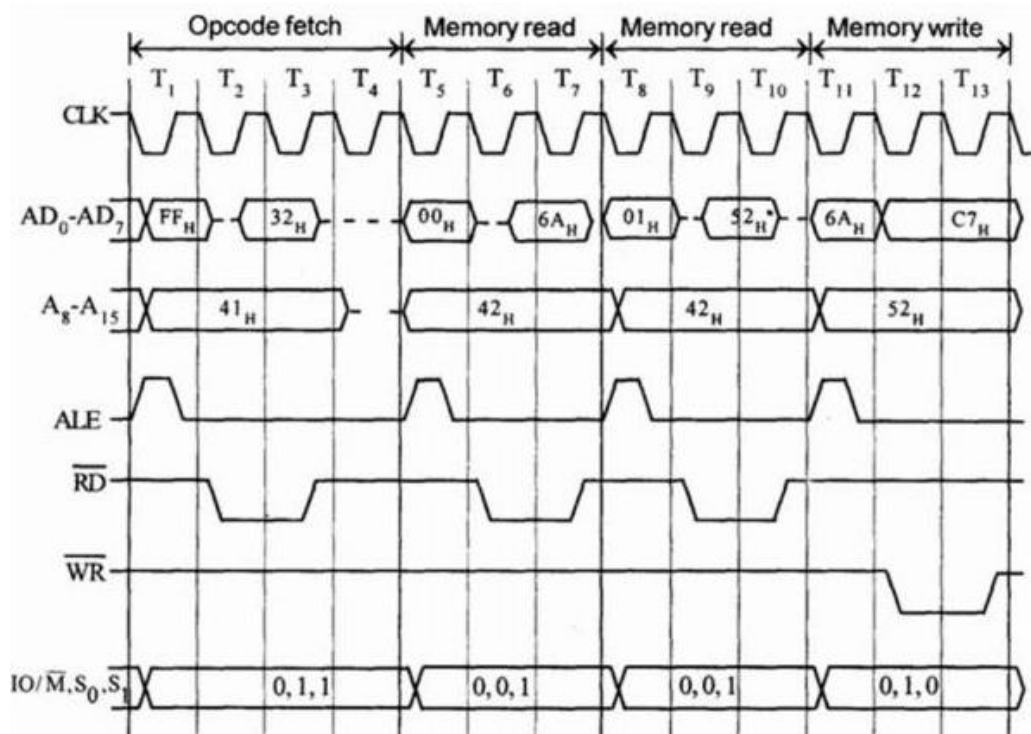
During T₂

- As ALE goes low, address is removed from AD₇-AD₀.
- As \overline{WR} goes low, data appears on AD₇-AD₀.

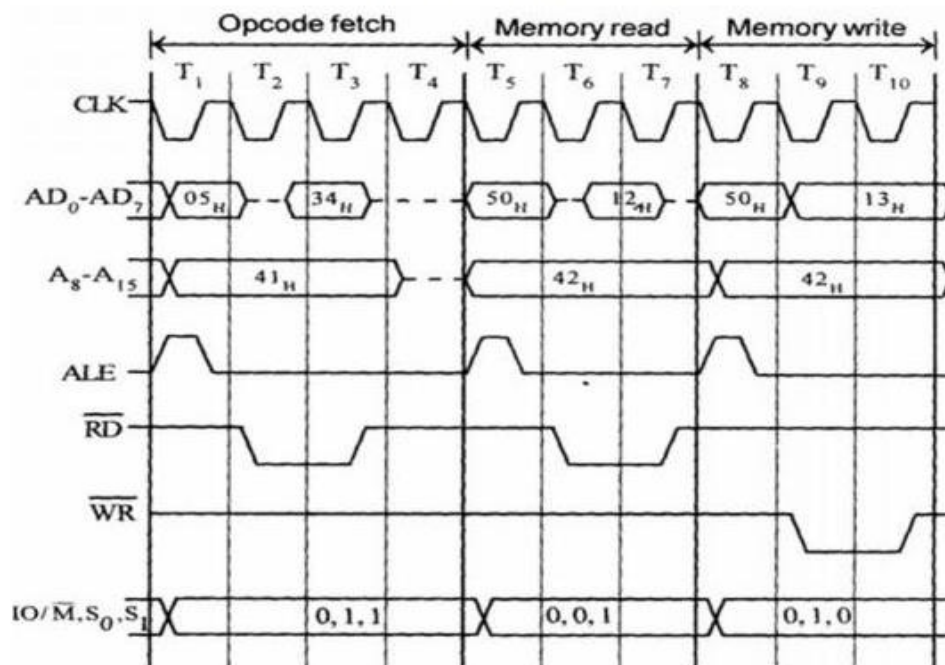
During T₃

Data remains on AD₇-AD₀ till \overline{WR} is low.

Timing diagram for STA 526AH



Timing Diagram for INR M





Debugging a program

Debugging a program in the 8085-microprocessor assembly language involves identifying and correcting errors or issues in your code to ensure it runs correctly. Here's a step-by-step guide to debugging a program in the 8085:

1. **Check for Syntax Errors:** Start by reviewing assembly language code for any syntax errors, such as misspelled mnemonics, incorrect registers, or missing operands. Use an assembler or assembly language IDE to identify and correct these errors.
2. **Simulate Execution:** Use 8085 simulator for the simulation of the execution of the program that allows observing how the program executes step by step.
3. **Breakpoints:** Set breakpoints at specific locations in the code where you suspect issues. When the program reaches a breakpoint, it will pause, allowing inspecting the registers, memory, and other information.
4. **Examine Register Contents:** Use the simulator to view the contents of the CPU registers (e.g., accumulator, program counter, and stack pointer) at various points in the program. Verify that the register values match your expectations based on the program's logic.
5. **Memory Inspection:** Examine the contents of memory locations where the program reads from or writes to. Ensure that the data in memory is as expected. Pay close attention to memory addresses used for storing variables, constants, or program instructions.
6. **Step Through the Program:** Execute program one instruction at a time (single-stepping) to identify where issues may arise. Pay attention to the execution flow, branching instructions, and any conditional jumps.
7. **I/O and Peripherals:** If the program involves input and output operations, make sure to check the status of I/O ports and devices. Verify that the input data is being read correctly, and output data is written as expected.
8. **Compare with Documentation:** Refer to the 8085 microprocessor's documentation, including the instruction set, addressing modes, and timing diagrams, to ensure that correct instructions and addressing modes are used.
9. **Fix the Issue:** Once you identify the problem, correct the code accordingly. This may involve adjusting instructions, correcting data, or changing the program flow.
10. **Re-test:** After making changes, re-run the program in the simulator or emulator to verify that the issue has been resolved.