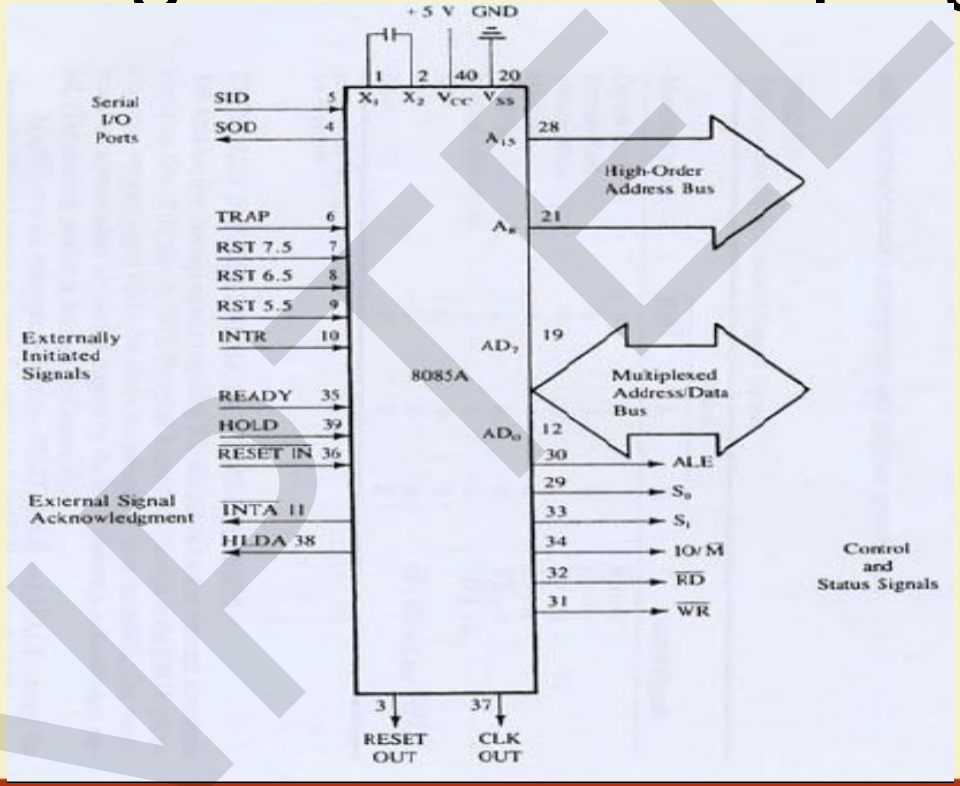
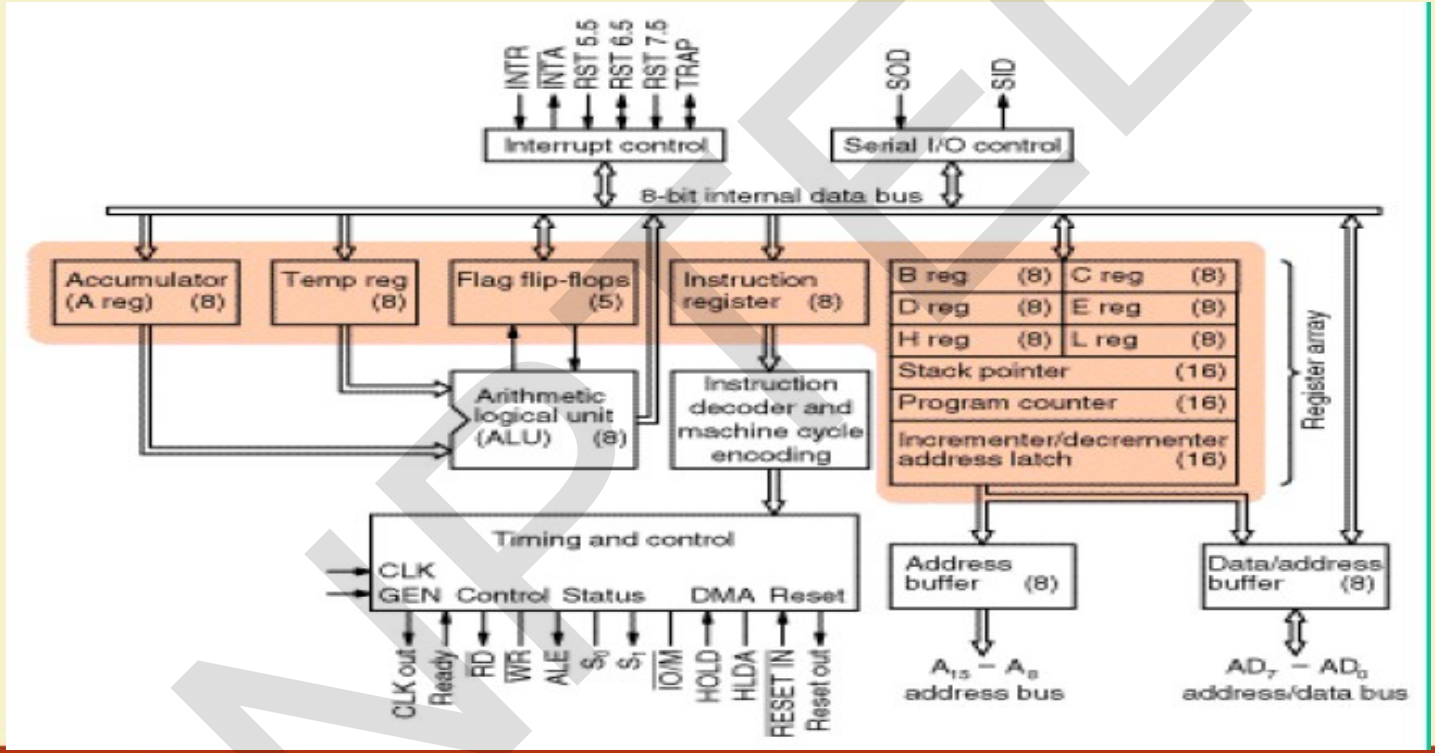
Pin Diagram with Grouping



System Bus

- System Bus – wires connecting memory & I/O to microprocessor
  - Address Bus
    - Unidirectional
    - Identifying peripheral or memory location
  - Data Bus
    - Bidirectional
    - Transferring data
  - Control Bus
    - Synchronization signals
    - Timing signals
    - Control signal

Architecture of Intel 8085 Microprocessor



Intel 8085 Microprocessor

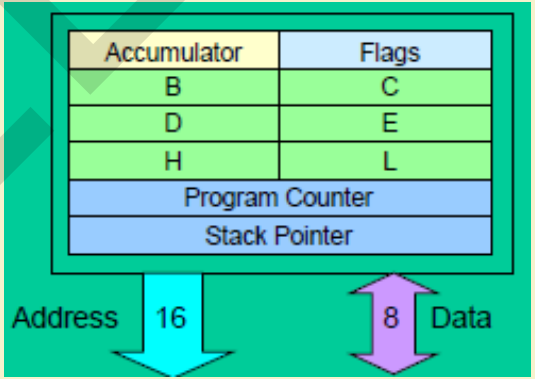
- Microprocessor consists of:
  - **Control unit**: control microprocessor operations.
  - **ALU**: performs data processing function.
  - **Registers**: provide storage internal to CPU.
  - **Interrupts**
  - **Internal data bus**

The ALU

- In addition to the arithmetic & logic circuits, the ALU includes the accumulator, which is part of every arithmetic & logic operation.
- Also, the ALU includes a temporary register used for holding data temporarily during the execution of the operation. This temporary register is not accessible by the programmer.

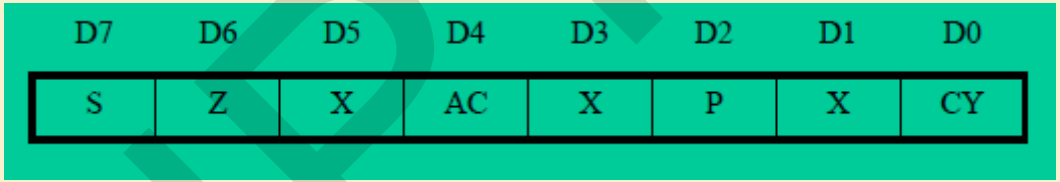
Registers

- General Purpose Registers
  - B, C, D, E, H & L (8 bit registers)
  - Can be used singly
  - Or can be used as 16 bit register pairs
    - BC, DE, HL
  - H & L can be used as a data pointer (holds memory address)
- Special Purpose Registers
  - Accumulator (8 bit register)
    - Store 8 bit data
    - Store the result of an operation



Flag Register

- 8 bit register – shows the status of the microprocessor before/after an operation
- S (sign flag), Z (zero flag), AC (auxillary carry flag), P (parity flag) & CY (carry flag)



Sign Flag

- Used for indicating the sign of the data in the accumulator
- The sign flag is set if negative (1 – negative)
- The sign flag is reset if positive (0 –positive)

Zero Flag

- Is set if result obtained after an operation is 0
- Is set following an increment or decrement operation of that register

Carry Flag

- Is set if there is a carry or borrow from arithmetic operation

Auxillary Carry and Parity

- Auxillary Carry Flag is set if there is a carry out of bit 3
- Parity Flag Is set if parity is even and is cleared if parity is odd

The Internal Architecture

- We have already discussed the general purpose registers, the Accumulator, and the flags.
- The Program Counter (PC)
  - This is a register that is used to control the sequencing of the execution of instructions.
  - This register always holds the address of the next instruction.
  - Since it holds an address, it must be 16 bits wide.

The Internal Architecture

- The Stack pointer
  - The stack pointer is also a 16-bit register that is used to point into memory.
  - The memory this register points to is a special area called the stack.
  - The stack is an area of memory used to hold data that will be retreived soon.
  - The stack is usually accessed in a Last In First Out (LIFO) fashion.

Non Programmable Registers

- Instruction Register & Decoder
  - Instruction is stored in IR after fetched by processor
  - Decoder decodes instruction in IR
- Internal Clock generator
  - 3.125 MHz internally
  - 6.25 MHz externally

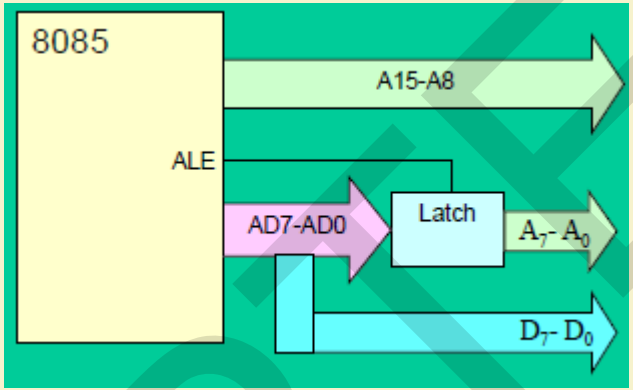
The Address and Data Busses

- The address bus has 8 signal lines A8 – A15 which are unidirectional.
- The other 8 address bits are multiplexed (time shared) with the 8 data bits.
  - 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.

Demultiplexing AD7-AD0

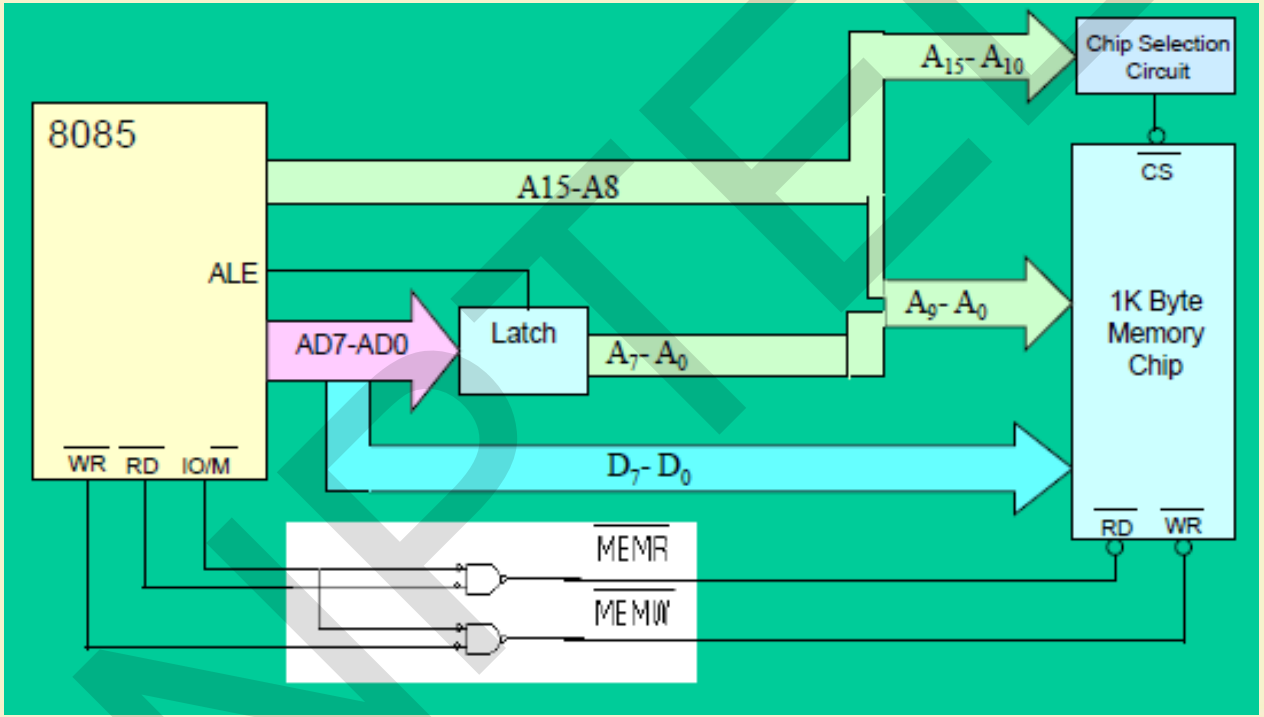
- The high order bits of the address remain on the bus for three clock periods. However, the low order bits remain for only one clock period and they would be lost if they are not saved externally.
- To make sure we have the entire address for the full three clock cycles, we will use an external latch to save the value of AD7– AD0 when it is carrying the address bits. We use the ALE signal to enable this latch.

Demultiplexing AD7-AD0



- Given that ALE operates as a pulse during T1, we will be able to latch the address. Then when ALE goes low, the address is saved and the AD7– AD0 lines can be used for their purpose as the bi-directional data lines.

The Overall Picture



The 8085 Instructions

- Since the 8085 is an 8-bit device it can have up to 2^8 instructions.
  - However, the 8085 only uses 246 combinations that represent a total of 74 instructions.
  - Most of the instructions have more than one format.
- These instructions can be grouped into five different groups:
  - Data Transfer Operations
  - Arithmetic Operations
  - Logic Operations
  - Branch Operations
  - Machine Control Operations

Instruction and Data Formats

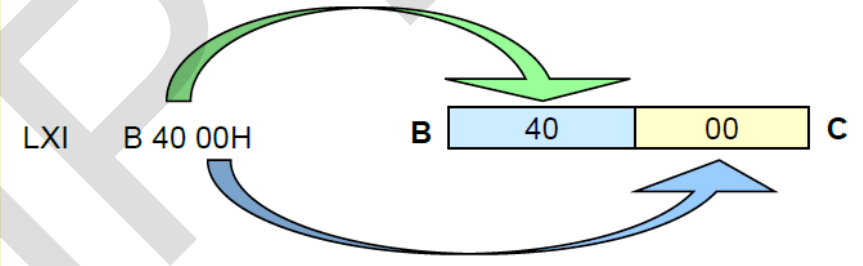
- Each instruction has two parts.
  - The first part is the task or operation to be performed.
    - This part is called the "opcode" (operation code).
  - The second part is the data to be operated on
    - Called the "operand".

Data Transfer Operations

- These operations simply **COPY** the data from the source to the destination.
- MOV, MVI, LDA, and STA
- They transfer:
  - Data between registers.
  - Data Byte to a register or memory location.
  - Data between a memory location and a register.
  - Data between an I\O Device and the accumulator.
- The data in the source is not changed.

The LXI instruction

- The 8085 provides an instruction to place the 16-bit data into the register pair in one step.
  - **LXI Rp, <16-bit address>** (Load eXtended Immediate)
  - The instruction **LXI B 4000H** will place the 16-bit number 4000 into the register pair B, C.
  - The upper two digits are placed in the 1st register of the pair and the lower two digits in the 2nd



The Memory "Register"

- Most of the instructions of the 8085 can use a memory location in place of a register.
  - The memory location will become the "memory" register M.
    - **MOV M B**
    - copy the data from register B into a memory location.
  - Which memory location?
- The memory location is identified by the contents of the HL register pair.
  - The 16-bit contents of the HL register pair are treated as a 16-bit address and used to identify the memory location.

Using the Other Register Pairs

- There is also an instruction for moving data from memory to the accumulator without disturbing the contents of the H and L register.
  - **LDAX Rp** (LoaD Accumulator eXtended)
    - Copy the 8-bit contents of the memory location identified by the Rp register pair into the Accumulator.
    - This instruction only uses the **BC** or **DE** pair.
    - It does not accept the **HL** pair.

Indirect Addressing Mode

- Using data in memory directly (without loading first into a Microprocessor's register) is called Indirect Addressing.

- Indirect addressing uses the data in a register pair as a 16-bit address to identify the memory location being accessed.
  - The HL register pair is always used in conjunction with the memory register "M".
  - The BC and DE register pairs can be used to load data into the Accumultor using indirect addressing.

Arithmetic Operations

- Addition (ADD, ADI):
  - Any 8-bit number.
  - The contents of a register.
  - The contents of a memory location.
  - Can be added to the contents of the accumulator and the result is stored in the accumulator.
- Subtraction (SUB, SUI):
  - Any 8-bit number
  - The contents of a register
  - The contents of a memory location
  - Can be subtracted **from** the contents of the accumulator. The result is stored in the accumulator.

Arithmetic Operations Related to Memory

- These instructions perform an arithmetic operation using the contents of a memory location while they are still in memory.
  - ADD M
    - Add the contents of M to the Accumulator
  - SUB M
    - Sub the contents of M from the Accumulator
  - INR M / DCR M
    - Increment/decrement the contents of the memory location in place.
  - All of these use the contents of the HL register pair to identify the memory location being used.

Arithmetic Operations

– Increment (INR) and Decrement (DCR):
  - The 8-bit contents of any memory location or any register can be directly incremented or decremented by 1.
  - No need to disturb the contents of the accumulator.

Manipulating Addresses

- Now that we have a 16-bit address in a register pair, how do we manipulate it?
  - It is possible to manipulate a 16-bit address stored in a register pair as one entity using some special instructions.

- **INX Rp**
- **DCX Rp**

The register pair is incremented or decremented as one entity. No need to worry about a carry from the lower 8-bits to the upper. It is taken care of automatically.

Logic Operations

- These instructions perform logic operations on the contents of the accumulator.
  - ANA, ANI, ORA, ORI, XRA and XRI
  - Source: Accumulator and
    - An 8-bit number
    - The contents of a register
    - The contents of a memory
  - Destination: Accumulator

ANA	R/M	AND Accumulator With Reg/Mem
ANI	#	AND Accumulator With an 8-bit number
ORA	R/M	OR Accumulator With Reg/Mem
ORI	#	OR Accumulator With an 8-bit number
XRA	R/M	XOR Accumulator With Reg/Mem
XRI	#	XOR Accumulator With an 8-bit number

Logic Operations

– Complement:

- 1’s complement of the contents of the accumulator.
  - CMA No operand

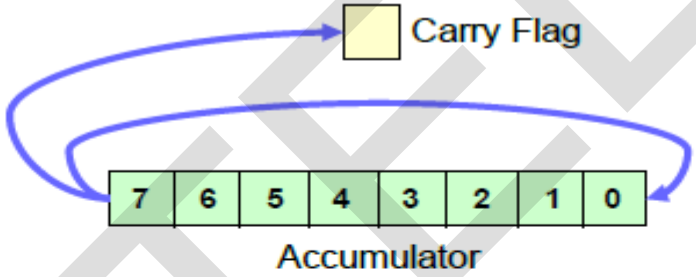
Additional Logic Operations

- Rotate
  - Rotate the contents of the accumulator one position to the left or right.

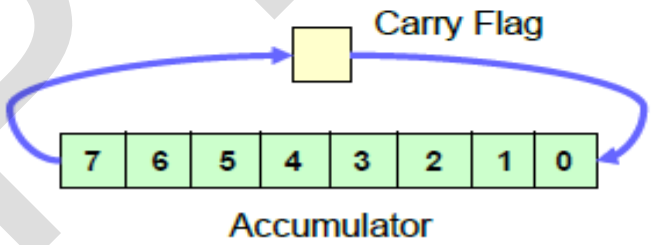
–	RLC	Rotate the accumulator left. Bit 7 goes to bit 0 AND the Carry flag.
–	RAL	Rotate the accumulator left through the carry. Bit 7 goes to the carry and carry goes to bit 0.
–	RRC	Rotate the accumulator right. Bit 0 goes to bit 7 AND the Carry flag.
–	RAR	Rotate the accumulator right through the carry. Bit 0 goes to the carry and carry goes to bit 7.

RLC vs. RLA

- RLC



- RAL



Logical Operations

- Compare
  - Compare the contents of a register or memory location with the contents of the accumulator.
    - CMP R/M — Compare the contents of the register or memory location to the contents of the accumulator.
    - CPI # — Compare the 8-bit number to the contents of the accumulator.
  - The compare instruction sets the flags (Z, Cy, and S).
  - The compare is done using an internal subtraction that does not change the contents of the accumulator.

    A – (R / M / #)

Branch Operations

- Two types:
  - Unconditional branch.
    - Go to a new location no matter what.
  - Conditional branch.
    - Go to a new location if the condition is true.

Unconditional Branch

- JMP Address
  - Jump to the address specified (Go to).
- CALL Address
  - Jump to the address specified but treat it as a subroutine.
- RET
  - Return from a subroutine.
- The addresses supplied to all branch operations must be 16-bits.

Conditional Branch

- Go to new location if a specified condition is met.
  - JZ Address (Jump on Zero)
    - Go to address specified if the **Zero flag is set**.
  - JNZ Address (Jump on NOT Zero)
    - Go to address specified if the **Zero flag is not set**.
  - JC Address (Jump on Carry)
    - Go to the address specified if the **Carry flag is set**.
  - JNC Address (Jump on No Carry)
    - Go to the address specified if the **Carry flag is not set**.
  - JP Address (Jump on Plus)
    - Go to the address specified if the **Sign flag is not set**
  - JM Address (Jump on Minus)
    - Go to the address specified if the **Sign flag is set**.

Machine Control

- HLT
  - Stop executing the program.
- NOP
  - No operation
  - Exactly as it says, do nothing.
  - Usually used for delay or to replace instructions during debugging.

Operand Types

- There are different ways for specifying the operand:
  - There may not be an operand (implied operand)
    - CMA
  - The operand may be an 8-bit number (immediate data)
    - ADI 4FH
  - The operand may be an internal register (register)
    - SUB B
  - The operand may be a 16-bit address (memory address)
    - LDA 4000H

Instruction Size

- Depending on the operand type, the instruction may have different sizes. It will occupy a different number of memory bytes.
  - Typically, all instructions occupy one byte only.
  - The exception is any instruction that contains immediate data or a memory address.
    - Instructions that include immediate data use two bytes.
      - One for the opcode and the other for the 8-bit data.
    - Instructions that include a memory address occupy three bytes.
      - One for the opcode, and the other two for the 16-bit address.

Instruction with Immediate Data

- Operation: Load an 8-bit number into the accumulator.
  - MVI A, 32
    - Operation: MVI A
    - Operand: The number 32
    - Binary Code:

0011 1110	3E	1st byte.
0011 0010	32	2nd byte.

Instruction with a Memory Address

- Operation: go to address 2085.
  - Instruction: JMP 2085
    - Opcode: JMP
    - Operand: 2085
    - Binary code:

1100 0011	C3	1st byte.
1000 0101	85	2nd byte
0010 0000	20	3rd byte

Addressing Modes

- The microprocessor has different ways of specifying the data for the instruction. These are called "addressing modes".
- The 8085 has four addressing modes:

– Implied	CMA
– Immediate	MVI B, 45
– Direct	LDA 4000
– Indirect	LDAX B

Data Formats

- In an 8-bit microprocessor, data can be represented in one of four formats:
  - ASCII
  - BCD
  - Signed Integer
  - Unsigned Integer.
- It is important to recognize that the microprocessor deals with 0's and 1's.
  - It deals with values as strings of bits.
  - It is the job of the user to add a meaning to these strings.

Data Formats

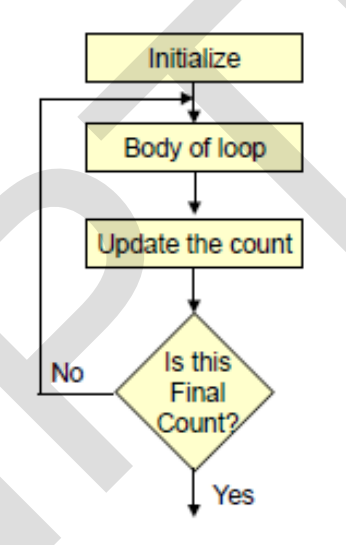
- Assume the accumulator contains the following value: 0100 0001. There are four ways of reading this value:
  - It is an unsigned integer expressed in binary, the equivalent decimal number would be 65.
  - It is a number expressed in BCD (Binary Coded Decimal) format. That would make it, 41.
  - It is an ASCII representation of a letter. That would make it the letter A.
  - It is a string of 0's and 1's where the 0^{th} and the 6^{th} bits are set to 1 while all other bits are set to 0.

Counters

- A loop counter is set up by loading a register with a certain value
- Then using the DCR (to decrement) and INR (to increment) the contents of the register are updated.
- A loop is set up with a conditional jump instruction that loops back or not depending on whether the count has reached the termination count.

Counters

- The operation of a loop counter can be described using the following flowchart.



Sample ALP for implementing a loop Using DCR instruction

```
        MVI C, 15H
LOOP    DCR C
        JNZ LOOP
```

Register Pair as a Loop Counter

- Using a single register, one can repeat a loop for a maximum count of 255 times.
- It is possible to increase this count by using a register pair for the loop counter instead of the single register.
  - A minor problem arises in how to test for the final count since DCX and INX do not modify the flags.
  - However, if the loop is looking for when the count becomes zero, we can use a small trick by ORing the two registers in the pair and then checking the zero flag.

Register Pair as a Loop Counter

- The following is an example of a loop set up with a register pair as the loop counter.

```
•       LXI B, 1000H
LOOP  DCX B
      MOV A, C
      ORA B
      JNZ LOOP
```

Delays

- Knowing how many T-States an instruction requires, we can calculate the time using the following formula:
    - Delay = No. of T-States / Frequency
- For example a “MVI” instruction uses 7 T-States. Therefore, if the Microprocessor is running at 2 MHz, the instruction would require 3.5 μSeconds to complete.

Delay loops

- We can use a loop to produce a certain amount of time delay in a program.

- The following is an example of a delay loop:

```
        MVI C, FFH          7 T-States
LOOP    DCR C               4 T-States
        JNZ LOOP           10 T-States
```

- The first instruction initializes the loop counter and is executed only once requiring only 7 T-States.
- The following two instructions form a loop that requires 14 T-States to execute and is repeated 255 times until C becomes 0.

Delay Loops (Contd.)

- We need to keep in mind though that in the last iteration of the loop, the JNZ instruction will fail and require only 7 T-States rather than the 10.
- Therefore, we must deduct 3 T-States from the total delay to get an accurate delay calculation.
- To calculate the delay, we use the following formula:

$$T_{delay} = T_O + T_L$$

  - T_{delay} = total delay
  - T_O = delay outside the loop
  - T_L = delay of the loop
- T_O is the sum of all delays outside the loop.

Delay Loops (Contd.)

- Using these formulas, we can calculate the time delay for the previous example:
- T_O = 7 T-States
  - Delay of the MVI instruction
- T_L = (14 X 255) - 3 = 3567 T-States
  - 14 T-States for the 2 instructions repeated 255 times (FF_{16} = 255_{10}) reduced by the 3 T-States for the final JNZ.

Register Pair as a Loop Counter

- The following is an example of a delay loop set up with a register pair as the loop counter.

```
        LXI B, 1000H      10 T-States
LOOP    DCX B              6 T-States
        MOV A, C           4 T-States
        ORA B              4 T-States
        JNZ LOOP          10 T-States
```

Register Pair as a Loop Counter

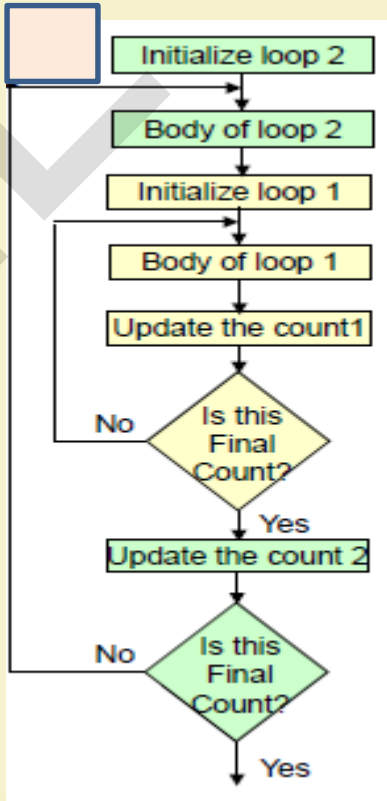
- Using the same formula from before, we can calculate:

- T_O = 10 T-States
  - The delay for the LXI instruction

- T_L = (24 X 4096) - 3 = 98301 T- States
  - 24 T-States for the 4 instructions in the loop repeated 4096 times ($1000_{16} = 4096_{10}$) reduced by the 3 T- States for the JNZ in the last iteration.

Nested Loops

- Nested loops can be easily setup in Assembly language by using two registers for the two loop counters and updating the right register in the right loop.
  - –In the figure, the body of loop2 can be before or after loop1.



Nested Loops for Delay

- Instead (or in conjunction with) Register Pairs, a nested loop structure can be used to increase the total delay produced.

	MVI B, 10H	7 T-States
LOOP2	MVI C, FFH	7 T-States
LOOP1	DCR C	4 T-States
	JNZ LOOP1	10 T-States
	DCR B	4 T-States
	JNZ LOOP2	10 T-States

Delay Calculation of Nested Loops

- The calculation remains the same except that it the formula must be applied recursively to each loop.
  - Start with the inner loop, then plug that delay in the calculation of the outer loop
- Delay of inner loop
  - T_{O1} = 7 T-States
    - MVI C, FFH instruction
  - T_{L1} = (255 X 14) - 3 = 3567 T-States
    - 14 T-States for the DCR C and JNZ instructions repeated 255 times (FF= 255) minus 3 for the final JNZ

Delay Calculation of Nested Loops

- Delay of outer loop
  - T_{O2} = 7 T-States
    - MVI B, 10H instruction
  - T_{L1} = (16 X (14 + 3574)) - 3 = 57405 T-States
    - 14 T-States for the DCR B and JNZ instructions and 3574
    - T-States for loop1 repeated 16 times ($10_{16} = 16_{10}$) minus 3 for the final JNZ.
  - T_{Delay} = 7 + 57405 = 57412 T-States
- Total Delay, T_{Delay} = 57412 X 0.5 μSec = 28.706 mSec

Increasing the delay

- The delay can be further increased by using register pairs for each of the loop counters in the nested loops setup.
- It can also be increased by adding dummy instructions (like NOP) in the body of the loop.

Timing Diagram

- Representation of Various Control signals generated during Execution of an Instruction.
- Following Buses and Control Signals must be shown in a Timing Diagram:
  - Higher Order Address Bus.
  - Lower Address/Data bus
  - ALE
  - RD
  - WR
  - IO/M

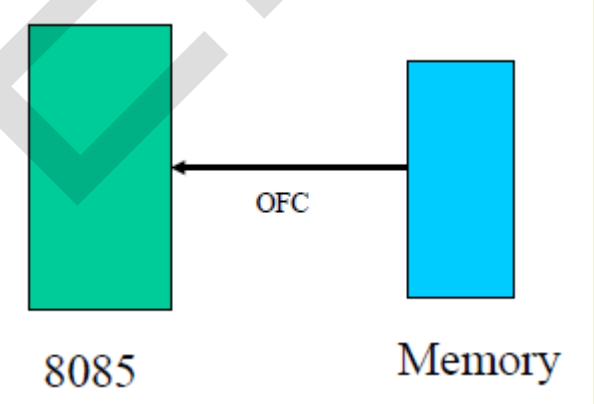
Timing Diagram

Instruction:

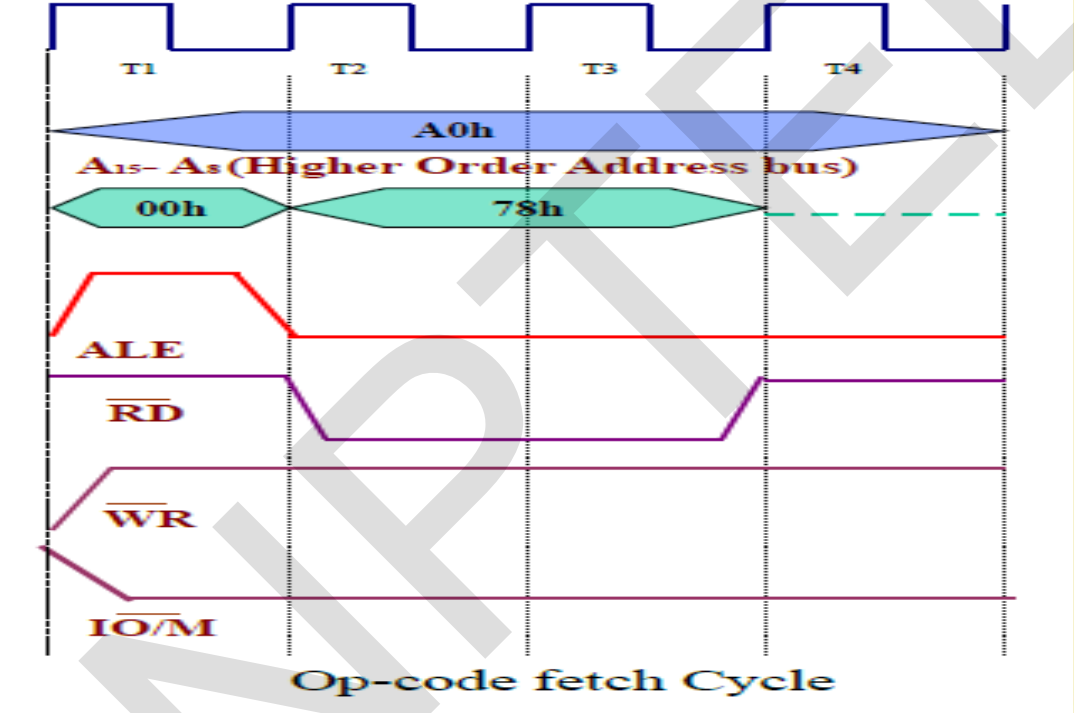
A000h MOV A,B

Corresponding Coding:

A000h 78



Timing Diagram



Op-code fetch Cycle

Timing Diagram

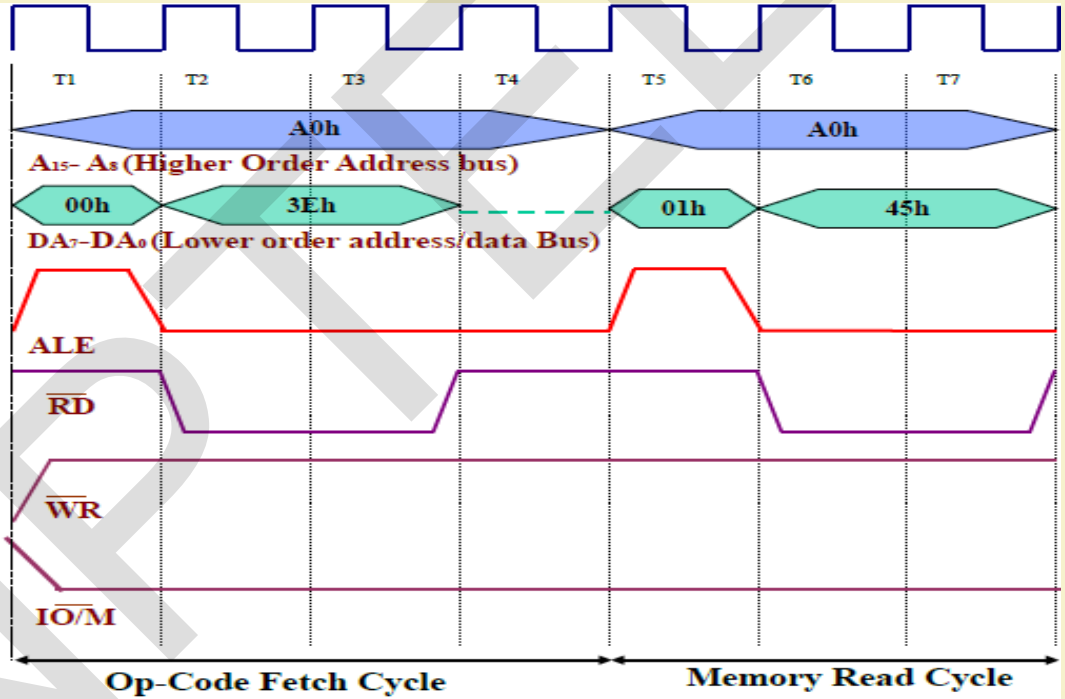
Instruction:

A000h MVI A,45h

Corresponding Coding:

A000h 3E

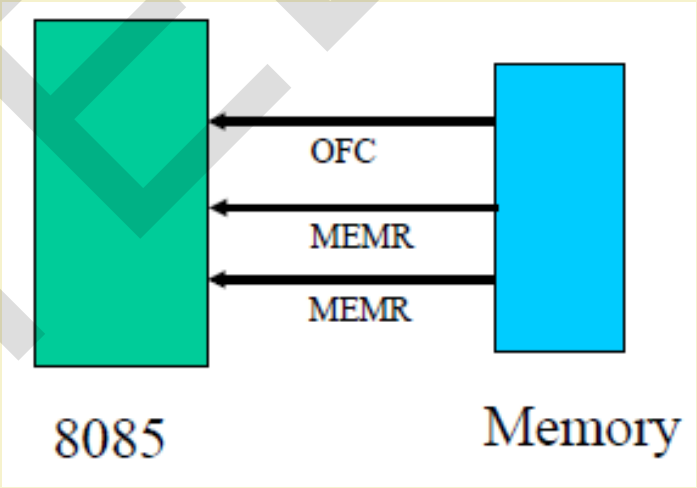
A001h 45



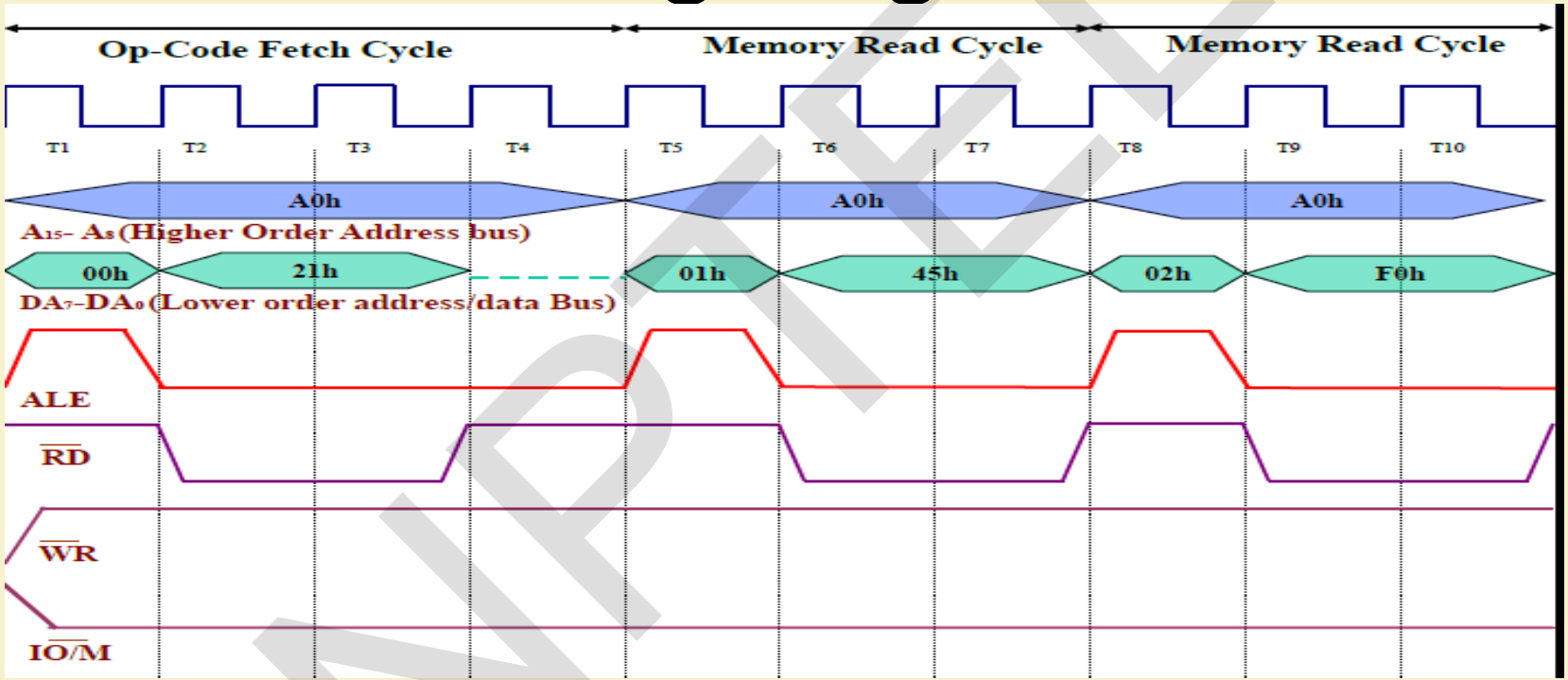
Timing Diagram

- Instruction:
- A000h LXI B,FO45h
- Corresponding Coding:

A000h	21
A001h	45
A002h	F0



Timing Diagram



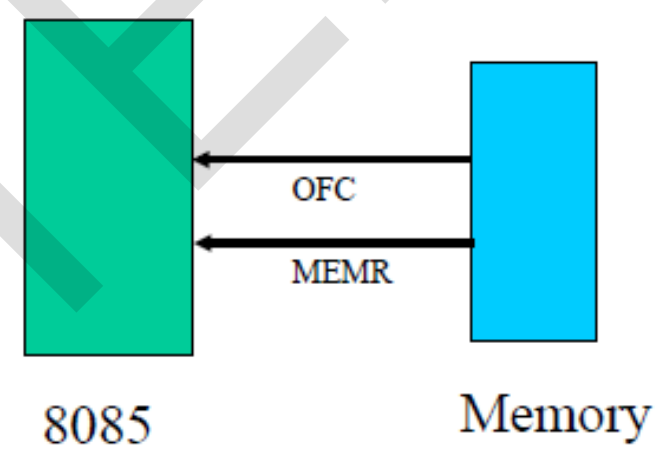
Timing Diagram

Instruction:

A000h MOV A,M

Corresponding Coding:

A000h 7E



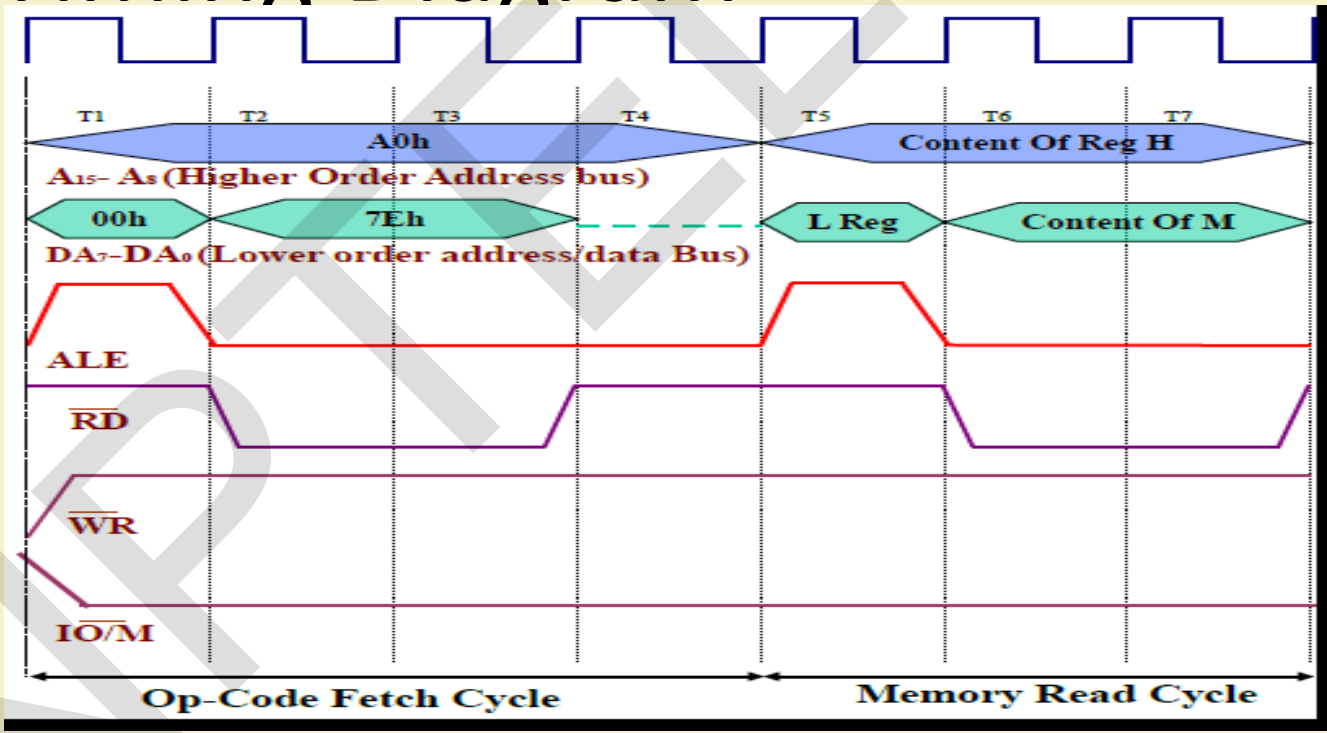
Timing Diagram

Instruction:

A000h MOV A,M

Corresponding Coding:

A000h 7E



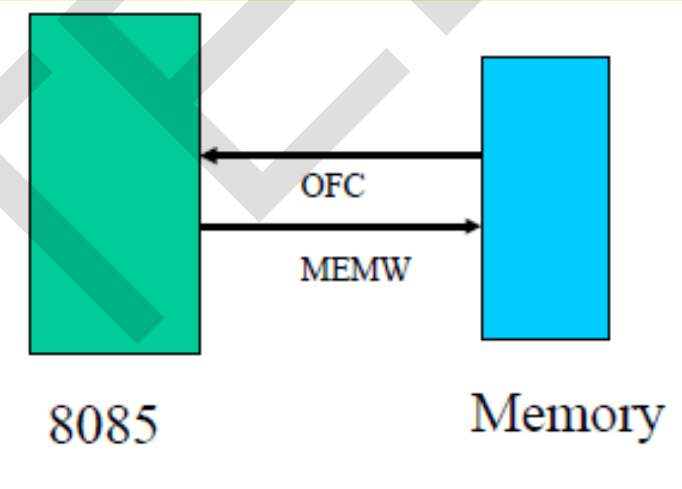
Timing Diagram

Instruction:

A000h MOV M,A

Corresponding Coding:

A000h 77



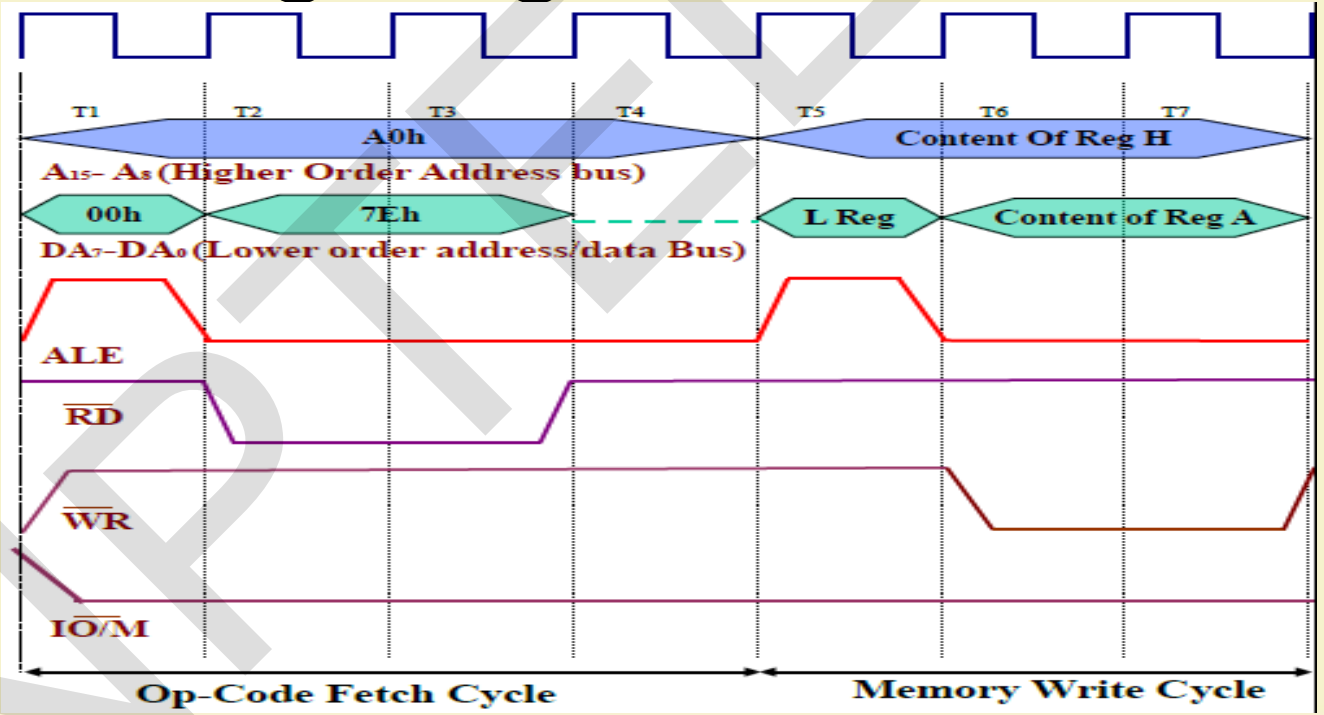
Timing Diagram

Instruction:

A000h MOV M,A

Corresponding Coding:

A000h 77



The Stack

- The stack is an area of memory identified by the programmer for temporary storage of information.
- The stack is a LIFO structure.
  - Last In First Out.
- The stack normally grows backwards into memory.
  - In other words, the programmer defines the bottom of the stack and the stack grows up into reducing address range.

The Stack

- Given that the stack grows backwards into memory, it is customary to place the bottom of the stack at the end of memory to keep it as far away from user programs as possible.
- In the 8085, the stack is defined by setting the SP (Stack Pointer) register.
- LXI SP, FFFFH
- This sets the Stack Pointer to location FFFFH (end of memory for the 8085).

Saving Information on the Stack

- Information is saved on the stack by PUSHing it on.
  - It is retrieved from the stack by POPing it off.
- The 8085 provides two instructions: PUSH and POP for storing information on the stack and retrieving it back.
  - Both PUSH and POP work with register pairs ONLY.

The PUSH Instruction

- PUSH B
  - Decrement SP
  - Copy the contents of register B to the memory location pointed to by SP
  - Decrement SP
  - Copy the contents of register C to the memory location pointed to by SP