

UNIT 2

PROGRAMMING WITH INTEL 8085 MICROPROCESSOR

10/7/2020

1

CONTENTS

- **MICROPROCESSOR ARCHITECTURE AND OPERATIONS, MEMORY, I/O DEVICES, MEMORY AND I/O OPERATIONS,**
- **8085 MICROPROCESSOR ARCHITECTURE**
- **ADDRESS, DATA AND CONTROL BUSES**
- **8085 PIN FUNCTIONS**
- **DE-MULTIPLEXING OF BUSES**
- **GENERATION OF CONTROL SIGNALS**
- **INSTRUCTION FORMAT AND DATA FORMAT**
- **ADDRESSING MODES**
- **INTEL 8085 INSTRUCTION SET**
- **VARIOUS PROGRAMS IN 8085**

OBJECTIVES

- **To explain the various functions of the registers in the 8085 programming model.**
- **Define the term flag and explain how the flags are affected.**
- **Explain the term operation code (opcode) and the operand, and illustrate these terms by writing instructions.**
- **Classify the instructions in terms of their word size and specify the number of memory registers required to store the instructions in memory.**
- **Define and explain the term addressing mode.**

INTERNAL ARCHITECTURE OF AN 8-BIT 8085 MICROPROCESSOR AND ITS REGISTERS

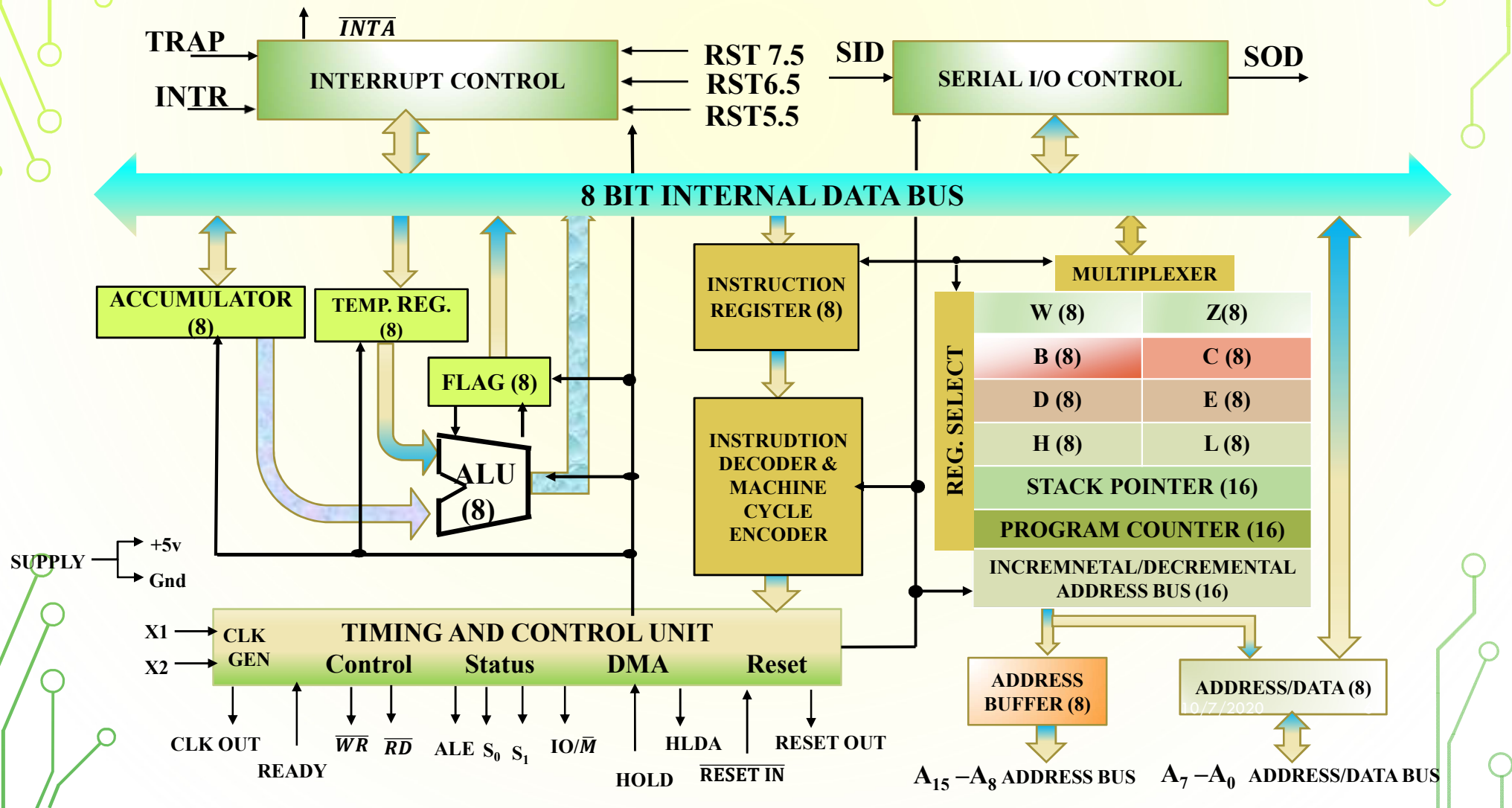
- The Intel 8085A is a complete 8-bit parallel central processing unit.
- The main components are:-
 - Array of registers
 - The arithmetic and logic unit
 - The encoder/decoder
 - The timing and control unit
- All linked by internal data bus.

FEATURES OF 8085

- **The main features of 8085 are:**
 - It is an 8 bit processor.
 - It is a single chip N-MOS device with 40 pins.
 - It has multiplexed address and data bus.(AD0-AD7).
 - It works on 5 Volt dc power supply.
 - The maximum clock frequency is 3 MHz while minimum frequency is 500kHz.
 - It provides 74 instructions with 5 different addressing modes.
 - It provides 16 address lines so it can access $2^{16}=64K$ bytes of memory.
 - It generates 8 bit I/O address so it can access $2^8=256$ input ports.


10/7/2020

INTERNAL ARCHITECTURE OF 8085 MICROPROCESSOR



A decorative graphic consisting of green lines and circles, resembling a circuit board or a stylized tree, located on the left side of the slide.

Microprocessor:

- **Clock driven semiconductor device consisting of electronic logic circuits manufactured either by using LSI or VLSI technique.**
 - **Capable of performing various computing functions and making decisions to change the sequence of program execution.**
 - **Can be divided into three segments for the sack of clarity; arithmetic unit(ALU), register array and control unit.**
- 
- A decorative graphic consisting of green lines and circles, resembling a circuit board or a stylized tree, located on the right side of the slide.

1. ARITHMETIC & LOGIC UNIT (ALU)

- **Performs the arithmetic/logical computing functions.**
- **Includes the accumulator, registers, the arithmetic and logic circuits and five flags and two temporary registers.**
- **Temporary registers are used to hold data during an arithmetic/logic operations.**
- **Result is stored in accumulator; the flags are set/reset according to the result of the operation.**

2. ACCUMULATOR (Register A)

- **8-bit register that is part of ALU and accessible to users.**
- **Used to store 8-bit data and to perform arithmetic/logic operations.**
- **8085 is called Accumulator based Microprocessor as of the two operands for all operations and result is also stored in it.**
- **When data is read from the input port, it is first placed in Accumulator and when data is sent to output port, it must be first placed in Accumulator.**

3. Temporary Registers (W& Z)

- **8-bit registers and not accessible to programmers.**
- **Data is placed in it for short period of time during execution.**

4. INSTRUCTION REGISTERS (IR)

- **8-bit register not accessible to programmers.**
- **Receives the operation code of instruction from the internal data bus and passes it to instruction decoder.**
- **Decoder decodes the instruction so that what operation is to be performed by the Microprocessor.**

5. REGISTER ARRAYS (B,C,D,E, H and L)

- **General purpose registers.**
- **Each 8-bit registers accessible to programmers.**
- **Data are stored on it during program execution.**
- **Can be used individually as 8-bit registers and as 16-bit registers in pair forming BC,DE, & HL.**
- **Data can be directly added or transformed to one another.**
- **Their content can be incremented or decrement and combined logically with the content of accumulator.**

6. STACK POINTER (SP)

- **16-bit register used as memory pointer.**
- **Points to the memory location in R/W memory, called the stack.**
- **Also called LIFO queue.**
- **The beginning of the stack is defined by loading the 16-bit address in the stack pointer.**

7. PROGRAM COUNTER (PC)

- **16-bit register that holds address of the next instruction to be executed.**
- **As microprocessor begins to execute a program , the memory location of first instruction is placed in PC.**
- **PC maintains the sequence of execution of instructions.**
- **Automatically incremented by one to point the next memory location when a byte is being fetched; i.e. it keeps the record of program by counting the memory address, hence name PC.**

8. FLAGS

- 5 flip-flops in 8085, each holding the status of different states separately known as flag register.

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

- Each flip-flop is called flags.
- 8085 can set or reset each flags depending on the type of operation.
- The flags are:
 - Sign (S)
 - Zero(Z)
 - Auxiliary Carry (AC)
 - Parity (P)
 - Carry (CY)

FLAGS Cond...

- **The state of flags indicate the result of arithmetic/ logic operation, which in turns used for decision making processes.**
- **Carry (CY):**
 - **Stores the carry or borrow from one byte to another.**
 - **It is set (CY=1), when the last arithmetic operation generates carry or borrow, otherwise reset (CY=0).**
- **Zero (Z):**
 - **Z=1, if the result of last operation of ALU is zero, otherwise, Z=0.**
 - **Often used in loop control and in searching for particular data value.**

FLAGS Cond...

- **Sign (S):**
 - After the execution of an arithmetic/logic operation, if bit D7 (MSB) of the result (usually accumulator) is 1, the sign flag is set.
 - Used with signed numbers.
 - In a given byte, if D7 is 1, it is viewed as *negative*; if it is 0, it is viewed as *positive*.
 - This flag is irrelevant to the operation of unsigned numbers.
- **Parity (P):**
 - After arithmetic/logic operation, if the result has even number of 1's (even parity), the flag is set, otherwise reset.

9. TIMING & CONTROL UNIT

- Synchronizes all the operations with the clock.
- Generates the control signals necessary for communication between the microprocessor and peripherals.
- Control signals are similar to sync pulse in an *oscilloscope*.
- RD(bar) and WR(bar) signals are sync pulses indicating the presence of data on the data bus.

10. INTERRUPT CONTROL

- **Mainly 5 types of interrupt:**

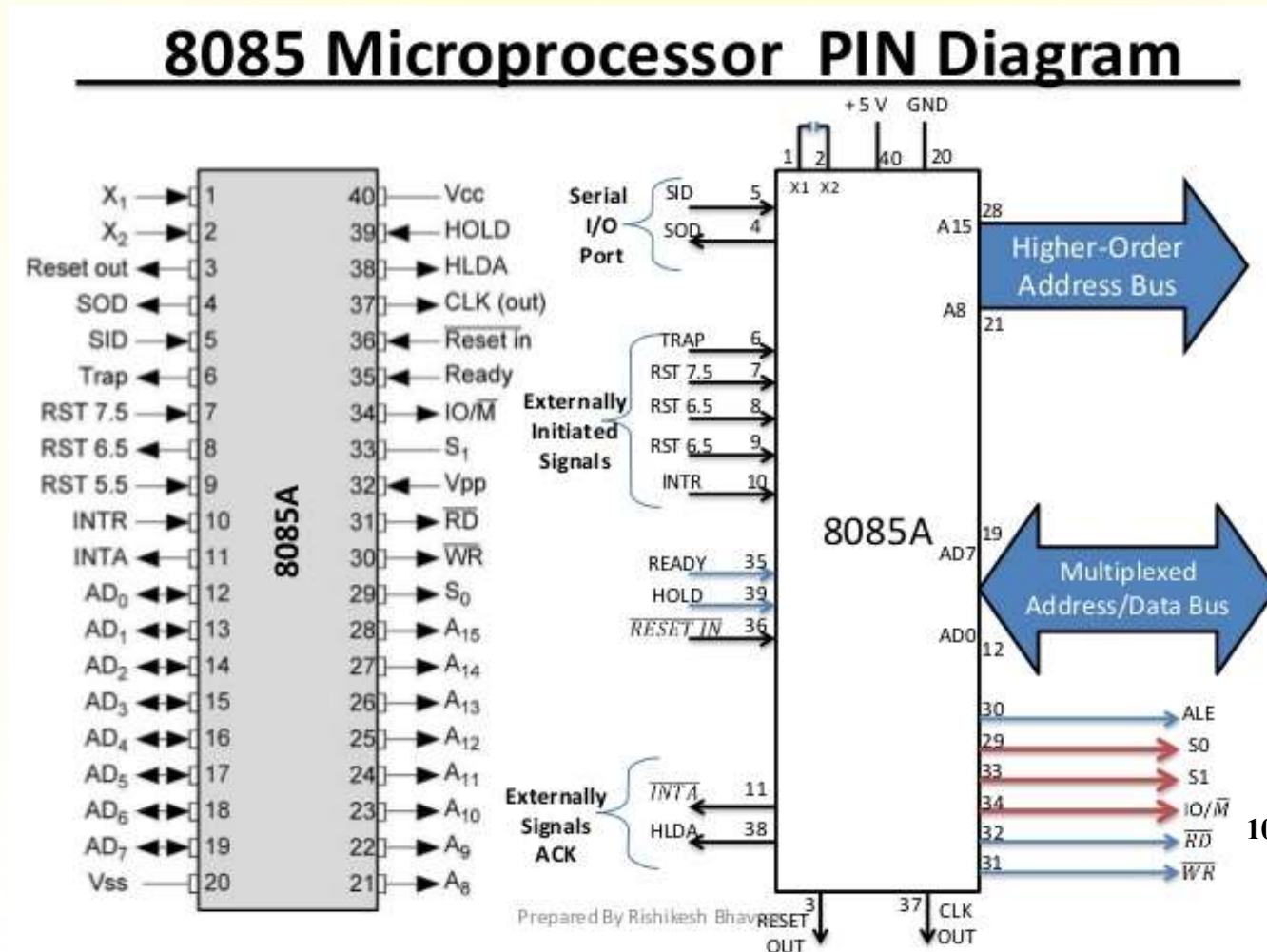
- **INTR**
- **TRAP**
- **RST5.5**
- **RST6.5**
- **RST7.5**



11. SERIAL I/O

- **Two serial I/O control signals:**
 - **Serial In Data (SID) and**
 - **Serial Out Data (SOD)**
- **Used to implement the serial data communication.**

8085 PIN DESCRIPTION



10/7/2020



8085 is 40 pin IC, DIP package. The total pin can be categorized to six groups:

I. Address Bus

II. Multiplexed Address/Data Bus

III. Control and status signal

IV. Power supply and clock signal

V. Interrupt and externally initiated signals

VI. Serial I/O ports

1. ADDRESS BUS

- **16 signal lines that are used as the address bus; however, these lines are split into two segments A_{15} - A_8 and AD_7 - AD_0 .**
- **A_{15} - A_8 are unidirectional and carries higher order address and the lower order AD_7 - AD_0 are multiplexed and bidirectional.**

2. MULTIPLEXED ADDRESS/DATA BUS

- **8-bit data bus.**
- **Multiplexed bidirectional AD_7 - AD_0 .**
- **These multiplexed lines are de-multiplexed to work as address bus and data bus separately using Address Latch Enable (ALE).**
- **If $ALE=1$, AD_7 - AD_0 acts as address bus, otherwise it acts as data bus. By default, it acts as data bus.**

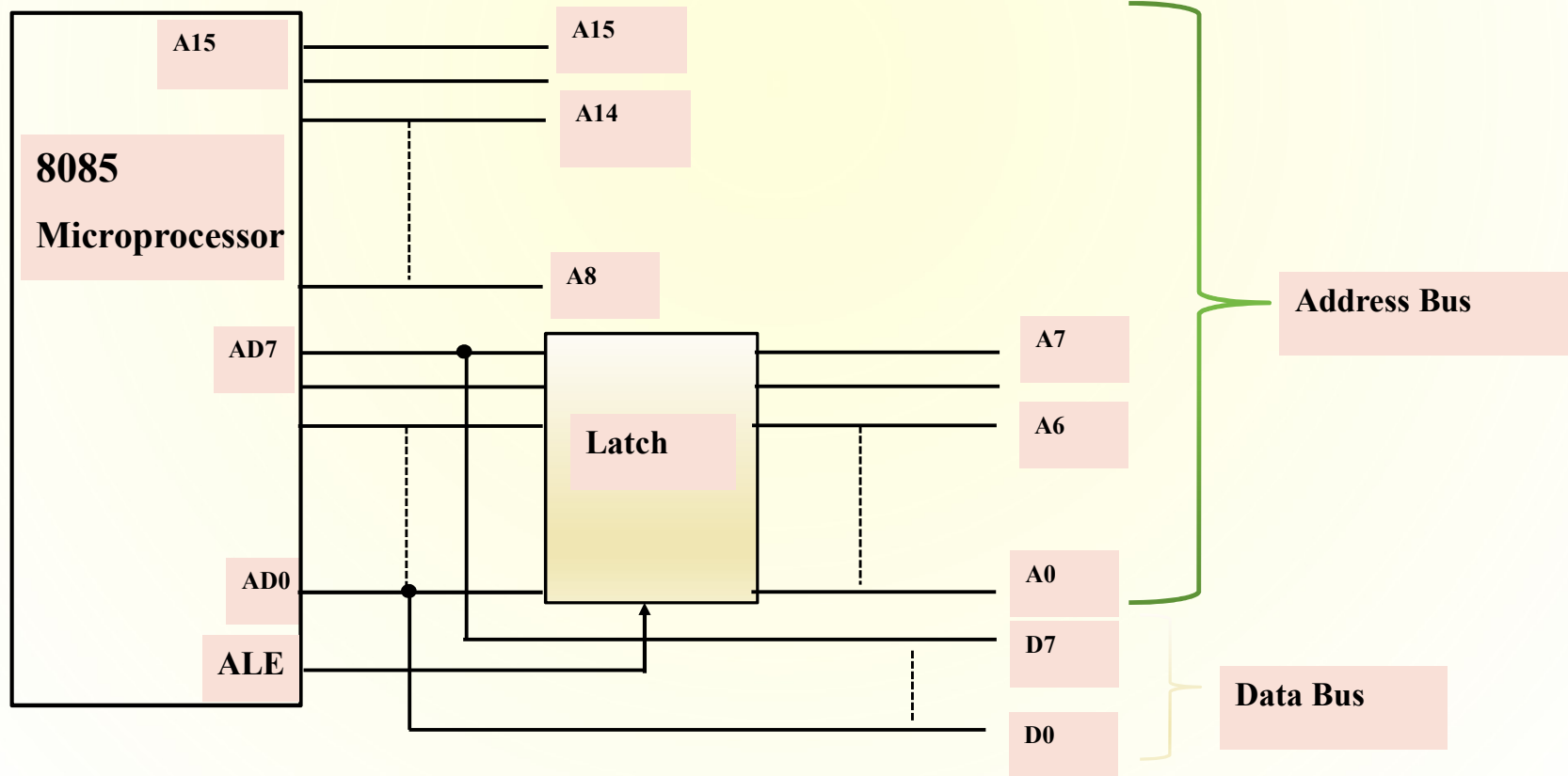
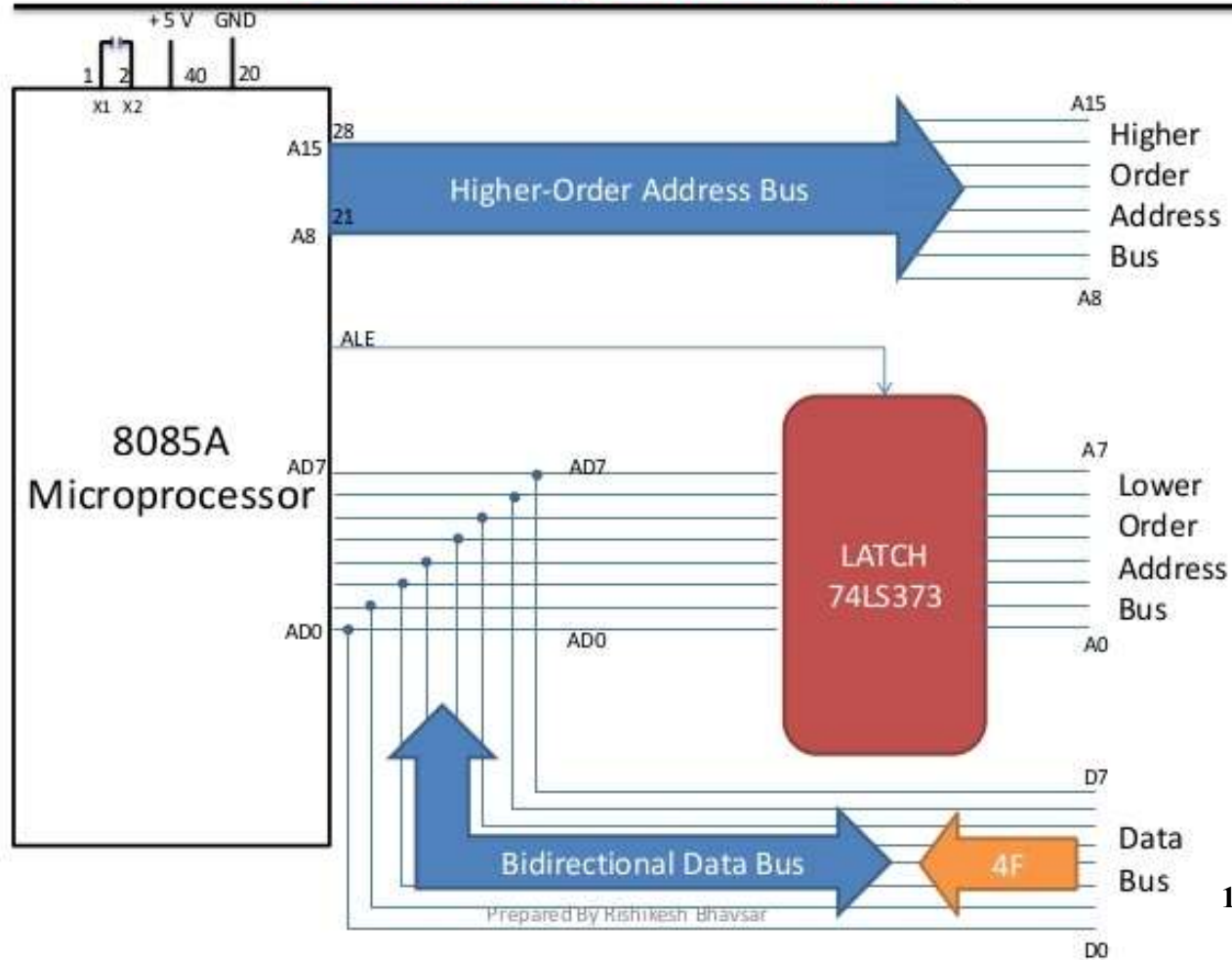


Fig: time multiplexed address and data bus

De-multiplexing the Bus AD_7 to AD_0



10/7/2020

26

3. CONTROL AND STATUS SIGNAL

- This group of signals includes two control signals (\overline{RD} and \overline{WR}).
- Three status signals (IO/\overline{M} , S_1 , S_0) to identify the nature of the operation, one special signal (ALE) to indicate the beginning of operation.

IO/\overline{M}'	S_1	S_0	Operation
Z	0	0	HALT
0	0	1	Memory Write
0	1	0	Memory Read
1	0	1	I/O Write
1	1	0	I/O Read
0	1	1	Opcode Fetch

4. POWER SUPPLY AND CLOCK SIGNAL

- **VCC: +5V power supply.**
- **VSS: Ground reference.**
- **X₁, X₂: A crystal (or RC, LC network) is connected at these two pins. The frequency is initially divided by 2; there for to operate a system at 3 MHz, the crystal should have frequency of 6 MHz**
- **CLK-clock output: This signal can be as a system clock for other devices.**

5. INTERRUPT AND EXTERNALLY INITIATED SIGNALS

- The 8085 has 5 interrupt signals that can be used to interrupt a program execution.

I. INTR (input)

II. $\overline{\text{INTA}}$ (output)

III. RST 7.5, 6.5, 5.5 (inputs)

IV. TRAP (input)

V. HOLD (input)

VI. HLDA (output)

VII. READY (Input)

VIII. $\overline{\text{RESETIN}}$

IX. RESET OUT

6. SERIAL I/O PORTS

- **Two signals to implement the serial transmission: SID (Serial Input Data) and SOD (Serial Output Data).**
- **In serial transmission, data bits are sent over a single line, one bit at a time, such as the transmission over telephone lines.**

Instruction Format:

On the basis of size they occupy, the instruction formats are:

- **One byte instruction:**

- ✓ It includes the op-code and operand in the same byte (all register transfer). E.g. MOV A, B

- **Two byte instruction:**

- ✓ The first byte specifies op-code and second byte specifies the operand. E.g. MVI A, 32H

- **Three byte instruction:**

- ✓ The first byte specifies op-code and following two byte specifies 16-bit operand. E.g. LXI B, 2032H

ADDRESSING MODES

- **EACH INSTRUCTION PERFORMS AN OPERATION ON THE SPECIFIED DATA.**
- **AN OPERAND MUST BE SPECIFIED FOR AN INSTRUCTION TO BE EXECUTED.**
- **THE OPERAND MAY BE IN THE GENERAL PURPOSE REGISTERS, ACCUMULATOR OR A MEMORY LOCATION.**
- **THE METHOD IN WHICH THE OPERAND IS SPECIFIED IN AN INSTRUCTION IS CALLED ADDRESSING MODE.**
- **TO PERFORM ANY CORRESPONDING INSTRUCTIONS TO THE MICROPROCESSOR, IN EACH INSTRUCTION, PROGRAMMER HAS TO SPECIFY 3 THINGS:**
 - ✓ **Operation to be performed.**
 - ✓ **Address of source of data.**
 - ✓ **Address of destination of result.**

TYPES

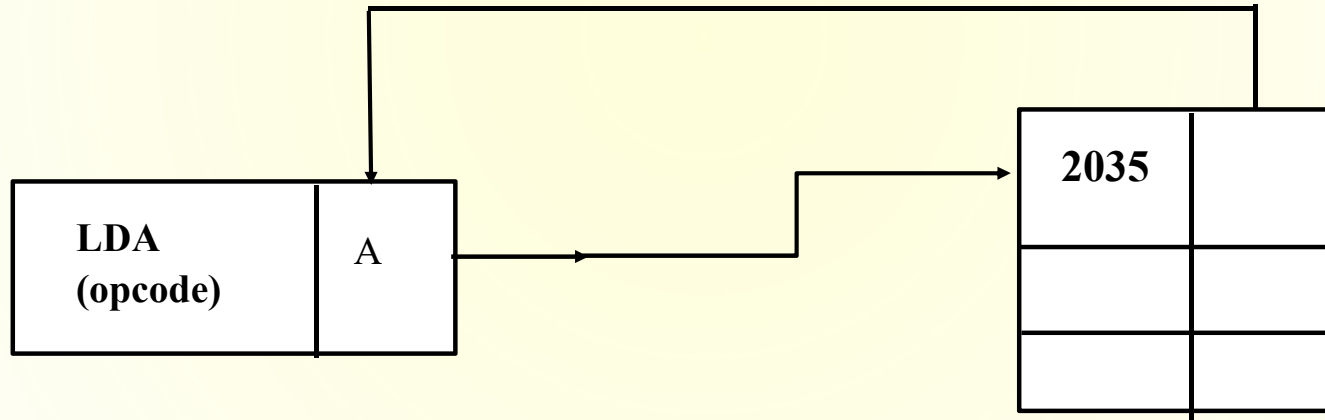
- **Intel 8085 uses the following addressing modes:**

- 1. Direct Addressing Mode*
- 2. Register Addressing Mode*
- 3. Register Indirect Addressing Mode*
- 4. Immediate Addressing Mode*
- 5. Implicit Addressing Mode*

DIRECT ADDRESSING:

- The effective address of the operand is specified directly in the instruction, 'or' Address of the operand is given in the instruction itself.
- Instructions using this mode may contain 2 or 3 bytes, with first byte as the Op-code followed by 1 or 2 bytes of address of data.
- Loading and storing in memory use 2 bytes of address while IN and OUT have one byte address.
- For example:
 - **LDA 2035H** ($A \leftarrow M [2035H]$, i.e. Load the memory content to the accumulator)
 - **STA 2500H** ($M [2500H] \leftarrow A$)
 - **IN 07H** ($A \leftarrow \text{port address } 07H$)

E.g: LDA 2035H



REGISTER ADDRESSING:

- register is the source of an operand for an instruction.
- similar to direct addressing.
- For example:

• **MOV A, B** **(A←B)** **//MOV is the operation.**

//B is the source of data.

//A is the destination.

• **ADD B** **(A←A+B)**

REGISTER INDIRECT ADDRESSING:

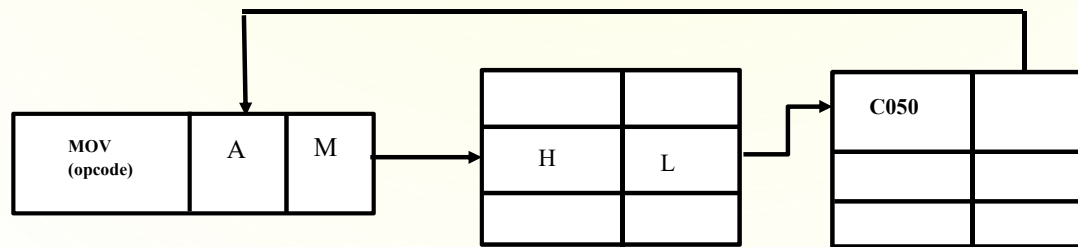
- The address of the operand is specified by register pair.

- **LDAX B** (if B=23 and C=50 then $A \leftarrow M[2350H]$)
- **STAX D** (if D=30 and E=10 then $M[3010H] \leftarrow A$)
- **MOV A, M** (M=HL; if H=68 and L=32, then $A \leftarrow M[6832H]$)

//Mov is the operation.

//M is the memory location specified by H-L register pair.

//A is the destination.



IMMEDIATE ADDRESSING:

- **simplest way of addressing.**
- **operate on immediate hexadecimal number.**
- **operand is present in instruction in this mode.**
- **used to define and use constants of set initial values of variables.**
- **operand may be 8 bit data or 16 bit data.**
- **For example:**
 - **MVI B, 05H (B←05H) //MVI is the operation.**
//05 H is the immediate data (source).
//A is the destination.
 - **LXI B, 7A21H (B←7A and C←21)**
 - **ADI 72H (A←A+72H)**

IMPLIED/IMPLICIT OR INHERENT ADDRESSING:

- **The instructions of this mode do not have operands.**
- **For example:**

EI (Enable Interrupt),

STC (set the carry flag),

NOP (No operation),

CMA (complement A)

//CMA is the operation.

//A is the source.

//A is the destination.



Instruction Types:

According to their functions, 8085 instructions can be classified as:

- 1) Data transfer**
- 2) Arithmetic**
- 3) Logical**
- 4) Branching**
- 5) Miscellaneous**

DATA TRANSFER INSTRUCTION

i. MOV

Syntax: MOV Rd, Rs
MOV M, Rs
MOV Rd, M

E.g. MOV A, B

ii. MVI

Syntax: MVI Rd/M, 8-bit data

iii. LXI

Syntax: LXI Reg. pair, 16-bit data

E.g. LXI B, 3434H

iv. LDA

Syntax: LDA 16-bit address

E.g. LDA 8085 $A \leftarrow M[8085]$

v. STA

Syntax: **STA 16-bit address**

e.g. STA 8085

$M[8085] \leftarrow A$

vi. LDAX

Syntax: **LDAX B/D register pair**

e.g. LDAX B

$A \leftarrow M[8050]$ (if B=80, C=50)

vii. STAX

Syntax: **STAX B/D register pair**

e.g. STAX B

$M[8050] \leftarrow A$ (if B=80, C=50)

viii. LHLD

Syntax: LHLD 16-bit address e.g. LHLD 8085
 $L \leftarrow M[8085]$ $H \leftarrow M[8086]$

ix. SHLD

Syntax: SHLD 16-bit address
e.g. SHLD 8085
 $M[8085] \leftarrow L$ $M[8086] \leftarrow H$

x. XCHG

Syntax: XCHG ; interchanges between D and H; and E and L
 $H \leftarrow D$ and $L \leftarrow E$
 $D \leftarrow H$ and $E \leftarrow L$

xi. IN

Syntax: IN 8-bit port address
e.g. IN 15; data received from port address 15 is transferred into Accumulator.

xii. OUT

Syntax: OUT 8-bit port address
e.g. OUT 15 ; content of Accumulator is transferred to port address 15.

2. Arithmetic Instructions

- | | | |
|-------|----------------|--|
| i) | ADD | |
| | Syntax: | ADD R/M
e.g. ADD B ; $A \leftarrow A+B$ |
| ii) | ADC | |
| | Syntax: | ADC R/M
e.g. ADC B ; $A \leftarrow A+B+CY$ |
| iii) | ADI | |
| | Syntax: | ADI 8-bit data
e.g. ADI 45 ; $A \leftarrow A+45$ |
| iv) | ACI | |
| | Syntax: | ACI 8-bit data
e.g. ACI 45 ; $A \leftarrow A+45+CY$ |
| v) | SUB | |
| | Syntax: | SUB R/M
e.g. SUB B ; $A \leftarrow A-B$ |
| vi) | SBB | |
| | Syntax: | SBB R/M
e.g. SBB B ; $A \leftarrow A-B-CY$ |
| vii) | SUI | |
| | Syntax: | SUI 8-bit data
e.g. SUI 45 ; $A \leftarrow A-45$ |
| viii) | SBI | |

Syntax: SBI 8-bit data
e.g. SBI 45; $A \leftarrow A - 45 - CY$

ix) **DAD**

Syntax: DAD register pair; contents of register pair is added to HL and the sum is saved in HL pair.
e.g. DAD B ; $HL \leftarrow HL + BC$

x) **INR**

Syntax: INR R/M
e.g. INR B ; $B \leftarrow B - 1$

xi) **DCR**

Syntax: DCR R/M
e.g. DCR C ; $C \leftarrow C - 1$

xii) **INX**

Syntax: INX register pair
e.g. INX H ; $HL \leftarrow HL + 1$

xiii) **DCX**

Syntax: DCX register pair
e.g. DCX B ; $BC \leftarrow BC - 1$

3) Logical instructions

i) **ANA**

Syntax: ANA reg./M

e.g. ANA B ; $A \leftarrow A \wedge B$

ii) **ANI**

Syntax: ANI 8-bit data

e.g. ANI 23H ; $A \leftarrow A \wedge 23H$

iii) **ORA**

Syntax: ORA Reg./M

e.g. ORA B ; $A \leftarrow A \vee B$

iv) **ORI**

Syntax: ORI 8-bit data

e.g. ORI 23H ; $A \leftarrow A \vee 23H$

v) **XRA**

Syntax: XRA Reg./M

e.g. XRA B ; $A \leftarrow A \oplus B$

vi)

XRI

Syntax:

XRI 8-bit data

e.g. XRI 23H ; $A \leftarrow A \oplus 23H$

vii)

CMA

e.g. CMA ; complements the content of accumulator

viii)

CMP

Syntax: CMP Reg./M

The contents of the operand (register/memory) are compared with the content of accumulator and both contents are preserved and the comparison is shown by setting the flags.

If $A > \text{Reg./M}$, CY=0, Z=0

If $A = \text{Reg./M}$, CY=0, Z=1

If $A < \text{Reg./M}$, CY=1, Z=0

ix)

CPI

Syntax: CPI 8-bit data

If $A > \text{data}$, CY=0, Z=0

If $A = \text{data}$, CY=0, Z=1

If $A < \text{data}$, CY=1, Z=0

x)

RLC

Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the carry flag (CY). Other flags are not affected.

Syntax: RLC

Before RLC: 10100111, CY=0

After RLC: 01001111; CY=1

xi)

RRC

Syntax:

RRC

Before RLC: 10100111, CY=0

After RLC: 11010011; CY=1

xii)

RAL

Each binary bit of the accumulator is rotated left by one position through the carry flag (CY). Other flags are not affected.

Syntax:

RAL

Before RAL: 10100111, CY=0

After RAL: 01001110; CY=1

xiii)

RAR

Each binary bit of the accumulator is rotated right by one position through the carry flag. Other flags are not affected.

Syntax:

RAR

Before RAR: 10100111, CY=0

After RAR: 01010011; CY=1

xiv)

DAA (Decimal Adjust Accumulator)

The contents of accumulator are changed from binary value to two binary-coded decimal (BCD) digits. This is the only instruction that uses the AC flag to perform binary to BCD conversion. All flags are affected.

4) Branching instructions

a) Jump instructions:

i) Unconditional jump:

The program sequence is transferred to the memory location specified by the 16-bit address without checking any condition.

Syntax: JMP 16-bit
 address e.g. JMP
 8085

ii) Conditional jump:

Op-code	Description	Flag status
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

JPE	Jump on even parity	P=1
JPO	Jump on odd parity	P=0

JZ	Jump on zero	Z=1
JNZ	Jump on non-zero	Z=0

b) Call instructions / Return instructions:

i) Unconditional call/return:

Syntax: CALL 16-bit address
 :
 :
 : RET

ii) Conditional Call/return:

Op-code	Description	Flag status
CC/RC	Call/Return on carry	CY=1
CNC/RNC	Call/Return on no carry	CY=0
CP/RP	Call/Return on positive	S=0
CM/RM	Call/Return on minus	S=1
CPE/RPE	Call/Return on even parity	P=1
CPO/RPO	Call/Return on odd parity	P=0
CZ/RZ	Call/Return on zero	Z=1
CNZ/RNZ	Call/Return on non-zero	Z=0

c) Restart instructions

The RST instructions are equivalent to 1-byte call instructions to one of the eight memory locations on page 0. The instructions are generally used in conjunction with interrupts and inserted using external hardware. However, these can be used as software instructions in a program to transfer program execution to one of the eight locations.

Op-code	Restart Address (H)
RST 0	0000
RST 1	0008
RST 2	0010
RST 3	0018
RST 4	0020
RST 5	0028
RST 6	0030
RST 7	0038

5. MISCELLANEOUS GROUP

- **Stack:** set of memory space defined by programmer for R/W purpose in a main memory.
- **PUSH Rp/PSW-** copie the content of specified reg. pair/PSW on the stack.
- **POP Rp/PSW-** copies top two data of memory of stack into specified reg. pair or **program status word**
- **XTHL-**exchange top of stack with HL
- **SPHL-**move HL to SP
- **PCHL-**move HL to PC
- **DI-**disable interrupt
- **EI-**enable interrupt
- **SIM-**set interrupt mask
- **RIM-** read interrupt mask
- **NOP-** no operation
- **HLT-**halt

TIME DELAY & COUNTERS

- **Counter:**

- Designed simply by loading an appropriate number into one of the registers.
- Use of the INR or DCR instructions.
- A loop is established to update a count, and each count is checked to determine whether it has reached the final number, if not the loop is repeated.

- **Time Delay:**

- The loop causes the delay.
- Depending upon the clock period of the system, the time delay occurred during looping.
- The instructions within the loop use their own T-states so they need certain time to execute resulting delay.

- Suppose we have an 8085 microprocessor with 2MHz clock frequency.
- Let us use the instruction MVI which takes 7 T-states.
- Clock frequency of system (f) = 2 MHz
- Clock period (T) = $1/f = 1/(2 \times 10^6) = 0.5\mu\text{s}$
- Time to execute MVI = 7 T-states * $0.5\mu\text{s} = 3.5\mu\text{s}$

• E.g.

MVI C, FFH	7
LOOP: DCR C	4
JNZ LOOP	10/7

- Here register C is loaded with count FFH (255_{10}) by using MVI which takes 7 T-states.
- Next 2 instructions DCR and JNZ form a loop with a total of 14 (4+10) T-states.
- The loop is repeated 255 times until C=0.
- The time delay in loop (TC) with 2 MHz frequency is

$$Tl = (T * \text{loop T- states} * \text{count})$$

- Where, Tl = time delay in loop

T = system clock period

Count = decimal value for counter

- $Tl = 0.5 * 10^{-6} * 14 * 255$
 $= 1785 \mu s$

- But JNZ takes only 7 T-states when exited from loop i. e. last count = 0.50
adjusted loop delay

$$Tl_{adj} = Tl - (3 \text{ T-states}) = 1785\mu s - 1.5\mu s = 1783.5\mu s$$

- Total delay loop of program,

$$\begin{aligned} TD &= \text{Time to execute outside code} + Tl_{adj} \text{ inside Loop} \\ &= 7 * 0.5\mu s + 1783.5\mu s \\ &= 1787\mu s \approx 1.8 \text{ ms} \end{aligned}$$

- To increase the time delay beyond 1.8 ms for 2MHZ microprocessor, we need to use counter for register pair or loop within a loop.

The slide features decorative circuit-like lines in a light green color. These lines are positioned along the left and right margins, extending from the top and bottom edges. They consist of vertical and horizontal segments connected by small circles, resembling a printed circuit board (PCB) layout.

NUMBER CONVERSION

- **BCD to Binary**
- **Binary to BCD**
- **Binary to ASCII**
- **ASCII to Binary**
- **BCD to 7-Segment LED code**

BCD TO BINARY

- Steps required for converting the 2-digit BCD numbers into its binary equivalent are:
- Separate an 8- bit packed BCD number into two 4 bit unpacked BCD digits: BCD_1 and BCD_2 .
- Convert each digit into its binary value according to its position.
- Add both binary numbers to obtain the binary equivalent of the BCD number.

EXAMPLE: CONVERT 43_{BCD} INTO ITS BINARY EQUIVALENT.

- $43_{10} = 01000011_{\text{BCD}}$
- **STEP1:** $0100\ 0011 \rightarrow 0000\ 0011$ Unpacked BCD_1
- $\rightarrow 0000\ 0100$ Unpacked BCD_2
- **STEP2:** Multiply BCD_2 by 10 (4×10)
- **STEP3:** Add BCD_1 to the answer in STEP2
- The multiplication of BCD_2 by 10 can be performed by various methods. One method is multiplication with repeated addition: add 10, 4 times.

BINARY TO BCD

- $1111\ 1111_2$ (FFH) = 255_{10} = $0010\ 0101\ 0101_{\text{BCD}}$

- **Step 1:** If < 100 go to step 2

Else subtract 100 repetitively.

Quotient is BCD_1 (Divide by 100)

- **Step 2:** If < 10 go to step 3

Else subtract 10 repetitively.

Quotient is BCD_2 (Divide by 10)

- **Step 3:** Remainder is BCD_3

BINARY TO ASCII

- 7-bit code with 128 combinations and each combination from 01H to 7FH is organized to a letter, decimal number, symbol or machine command.
- For eg. 30H to 39H represents 0 to 9, 41H to 5AH represents A to Z, 21H to 2FH represents various symbols and 61H to 7AH represents a to z.

General Letters / Numbers	ASCII (Hex)	ASCII (Decimal)
0 – 9	30 – 39	48 – 57
A – Z	41 – 5A	65 – 90
a – z	61 – 7A	97 – 122

ASCII TO BINARY

- **Step 1:** Subtract 30H
- **Step 2:** If $< 0AH$, then binary as it is
Else subtract 07H
- **For example:** If ASCII is 41H, then $41H - 30H = 11H$; $11H - 07H = 0AH$

BCD TO 7 – SEGMENT LED CODE CONVERSION:

- Table lookup technique (TLT) is used.
- In TLT, the codes of digits to be displayed are stored sequentially in memory.
- Common cathode:

BCD Number		7-Segment Code
0		3FH
1		06H
2		5BH
3		4FH
4		66H
5		6DH
6		7DH
7		07H
8		7FH
9		6FH
Invalid		00H