Intel 8085 Microprocessor

Interfacing

Microprocessor

- A microprocessor can perform some operation on a data and give the output.
- But to perform the operation we need an input to enter the data and an output to display the results of the operation.
- So we are using a keyboard and monitor as Input and output along with the processor.
- Microprocessors engineering involves a lot of other concepts and we also interface memory elements like ROM, EPROM to access the memory.

Contents

- Salient features of 8085
- Pin diagram of 8085
- 8085 Operations
- Peripheral I/O instruction for Intel 8085 Microprocessor and its timing diagram
- Interfacing with LED and seven segment display
- 8080A interrupts
- RST instructions

Salient Features of 8085

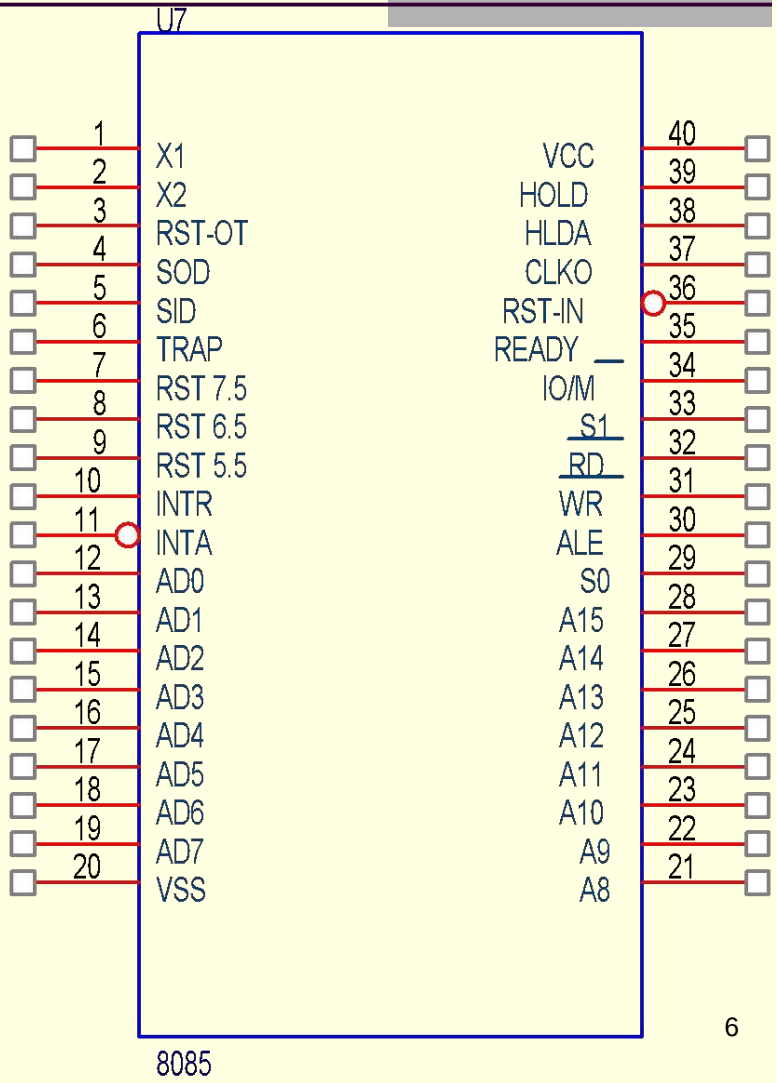
- It is a 8 bit microprocessor.
- It is manufactured with N-MOS technology.
- It has 16-bit address bus and hence can address up to 2^{16} = 65536 bytes (64KB) memory locations through A0-A15.
- The first 8 lines of address bus and 8 lines of data bus are multiplexed AD0 – AD7.
- Data bus is a group of 8 lines D0 – D7.

Salient Features of 8085 (Cont.)

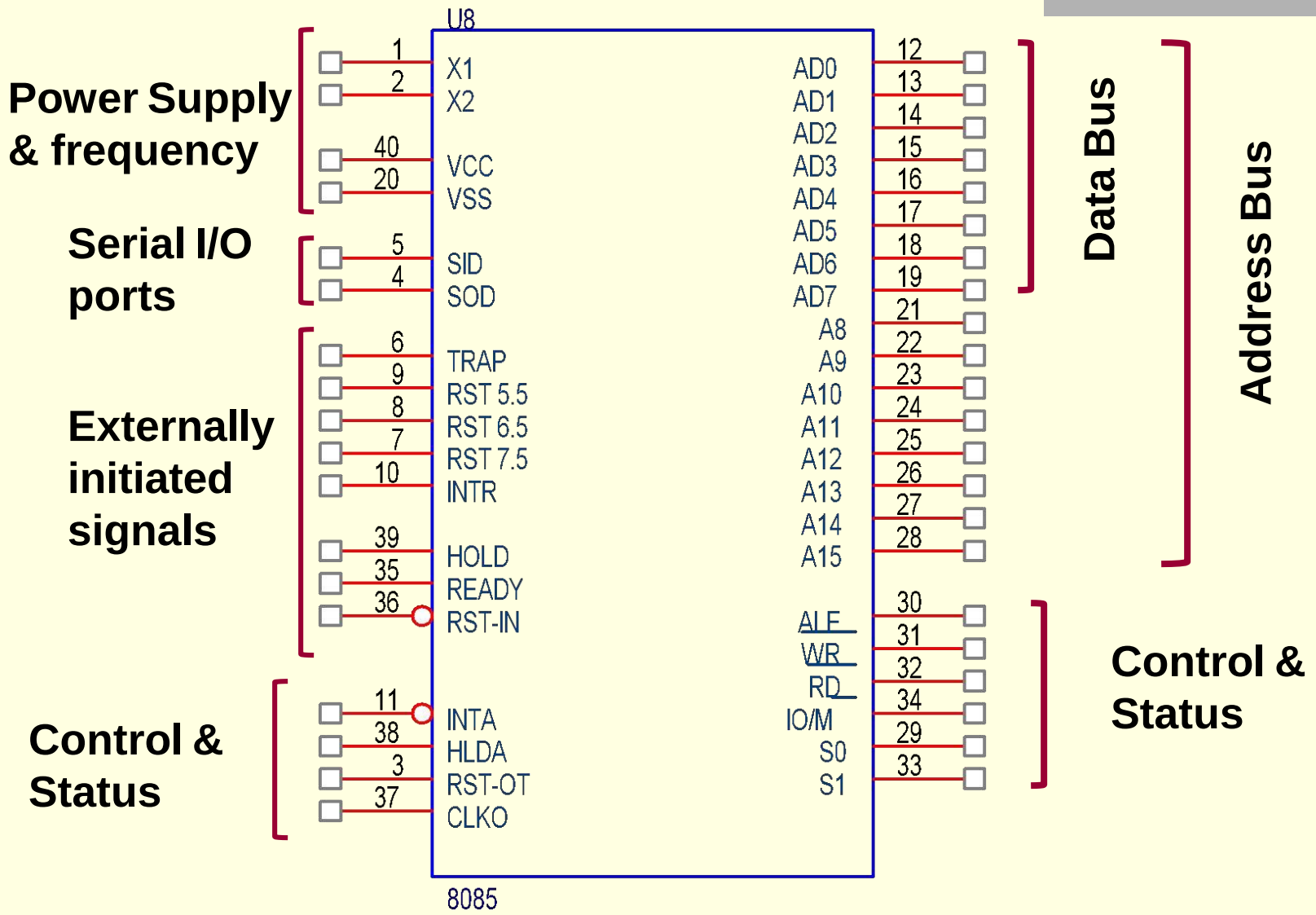
- It supports external interrupt request.
- A 16 bit program counter (PC)
  A 16 bit stack pointer (SP)
- Six 8-bit general purpose register arranged in pairs: BC, DE, HL.
- It requires a signal +5V power supply and operates at 3.2 MHZ single phase clock.
- It is enclosed with 40 pins DIP (Dual in line package).

Pinout Diagram of 8085

- A 40-pin IC
- Six groups of signals
  - Address Bus
  - Data Bus
  - Control and Status pins
  - Power Supply & frequency signals
  - Externally initiated Signals
  - Serial I/O ports



Logic Pinout of 8085



8085 Operations

- Microprocessor Initiated Operations
- Internal Operations
- Peripheral/Externally Initiated Operations

Microprocessor Initiated Operations

- Memory Read
- Memory Write
- I/O Read
- I/O Write

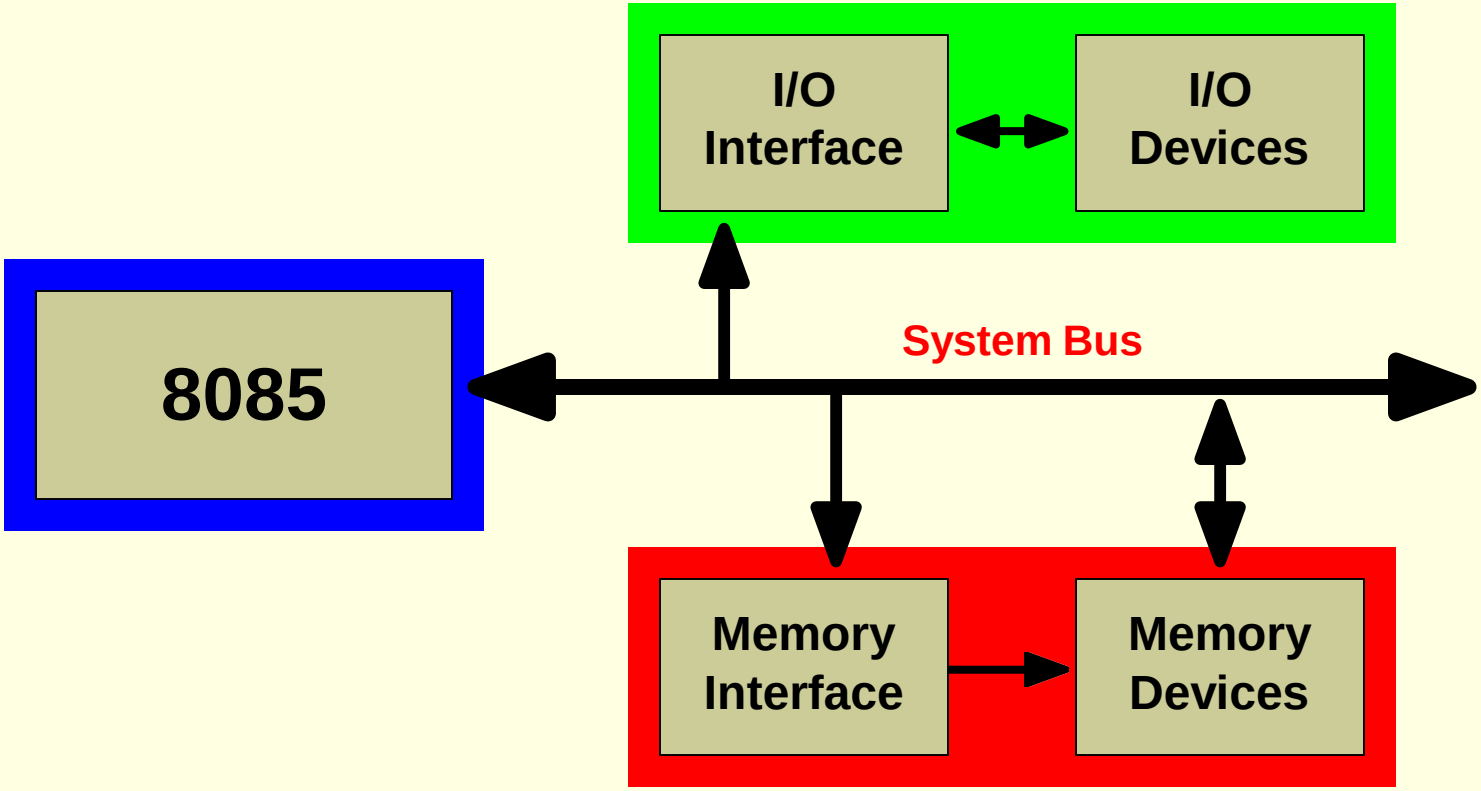
Internal Operations

- Store 8-bit data
- Perform Arithmetic and Logic Operations
- Test for conditions
- Sequence the execution of instructions
- Store/Retrieve data from stack during execution

Peripheral/Externally Initiated Operations

- Reset
- Interrupt
- Ready
- Hold

Interfacing I/O devices with 8085



Techniques for I/O Interfacing

- Memory-mapped I/O
- Peripheral-mapped I/O

Memory-mapped I/O

- 8085 uses its 16-bit address bus to identify a memory location
- Memory address space: 0000H to FFFFH
- 8085 needs to identify I/O devices also
- I/O devices can be interfaced using addresses from memory space
- 8085 treats such an I/O device as a memory location
- This is called Memory-mapped I/O

Peripheral-mapped I/O

- 8085 has a separate 8-bit addressing scheme for I/O devices
- I/O address space: 00H to FFH
- This is called Peripheral-mapped I/O or I/O-mapped I/O

8085 Communication with I/O devices

- Involves the following three steps
  1. Identify the I/O device (with address)
  2. Generate Timing & Control signals
  3. Data transfer takes place
- 8085 communicates with a I/O device only if there is a Program Instruction to do so

1. Identify the I/O device (with address)

1. Memory-mapped I/O (16-bit address)
2. Peripheral-mapped I/O (8-bit address)

2.Generate Timing & Control Signals

- Memory-mapped I/O
  - Reading Input: IO/$\overline{M}$ = 0, $\overline{RD}$ = 0
  - Write to Output: IO/$\overline{M}$ = 0, $\overline{WR}$ = 0
- Peripheral-mapped I/O
  - Reading Input: IO/$\overline{M}$ = 1, $\overline{RD}$ = 0
  - Write to Output: IO/$\overline{M}$ = 1, $\overline{WR}$ = 0

3. Data transfer takes place

Peripheral I/O Instructions

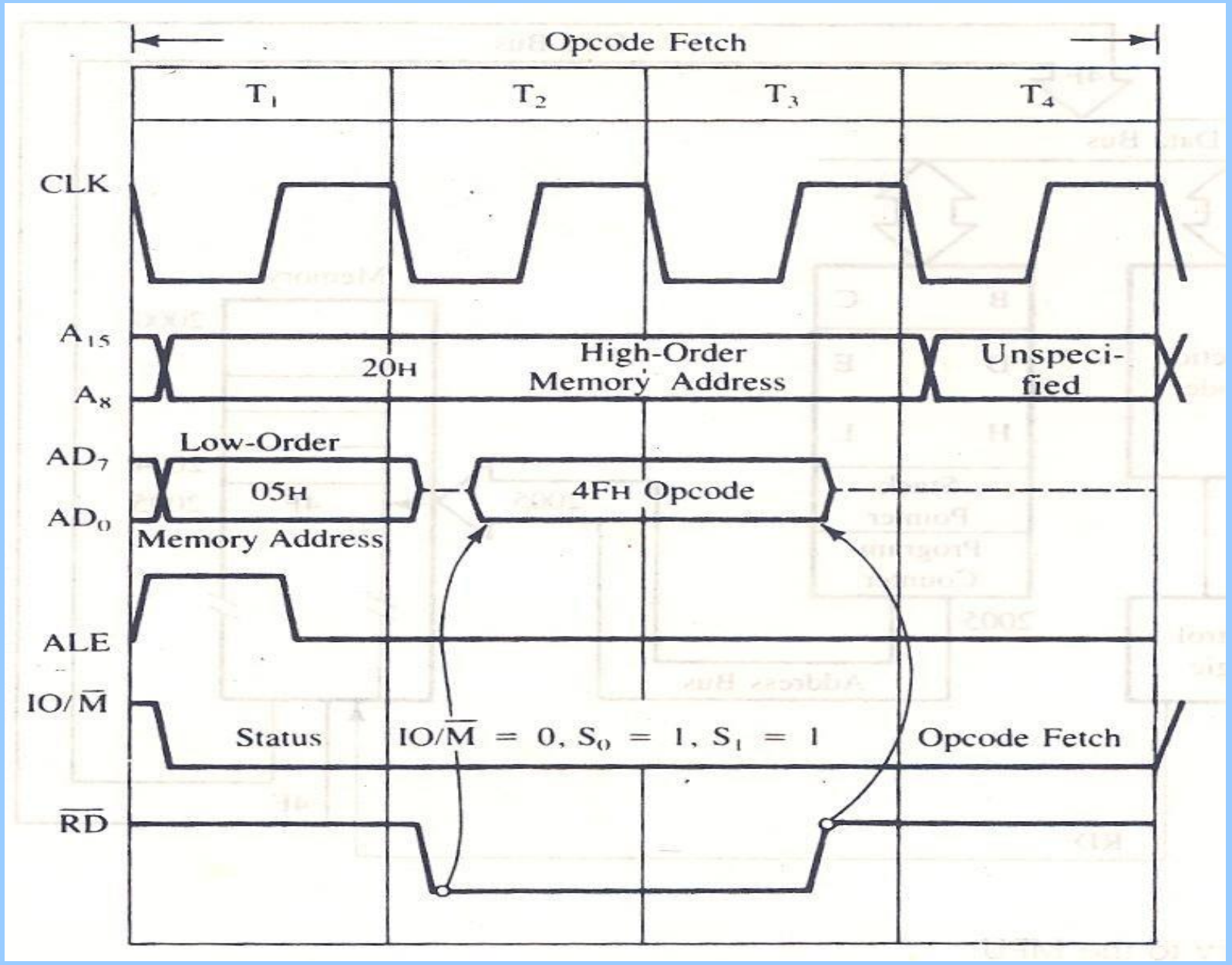
- **IN Instruction**
  - Inputs data from input device into the accumulator
  - It is a 2-byte instruction
  - Format: IN 8-bit port address
  - Example: IN 01H

- **OUT Instruction**
  - Outputs the contents of accumulator to an output device
  - It is a 2-byte instruction
  - Format: OUT 8-bit port address
  - Example: OUT 02H

Memory-mapped I/O Instructions

- I/O devices are identified by 16-bit addresses
- 8085 communicates with an I/O device as if it were one of the memory locations
- Memory related instructions are used
- For e.g. LDA, STA
- LDA 8000H
  - Loads A with data read from input device with 16-bit address 8000H
- STA 8001H
  - Stores (Outputs) contents of A to output device with 16-bit address 8001H

Timing Diagram



Interfacing with LED and seven segment display

- An **LED or Light Emitting Diode**, is a solid state optical pn-junction diode which emits light energy in the form of photons.
- the main advantage of light emitting diodes is that because of their small die size, several of them can be connected together within one small and compact package producing what is generally called a **7-segment Display**.

Cont…

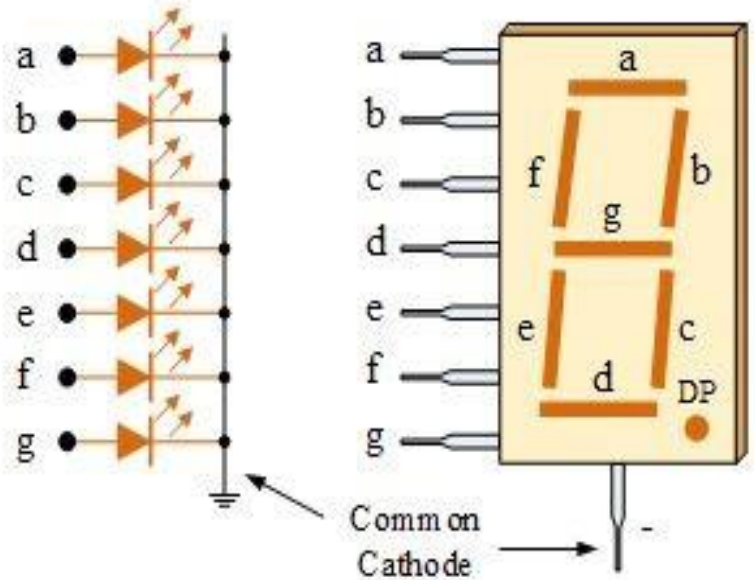
- The *7-segment display* consists of seven LEDs (hence its name) arranged in a rectangular fashion as shown. Each of the seven LEDs is called a segment because when illuminated the segment forms part of a numerical digit (both Decimal and Hex) to be displayed.

7-segment Display

- There are two types of LED 7-segment display called: **Common Cathode** (CC) and **Common Anode** (CA).

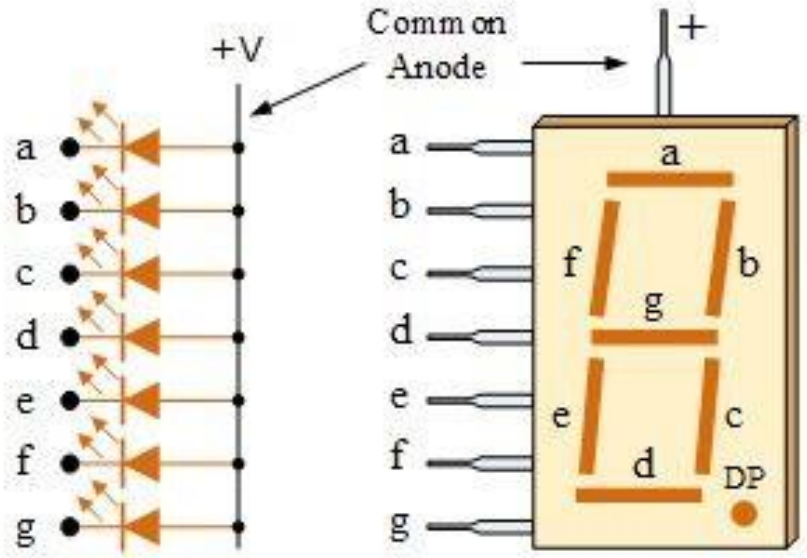
1. **The Common Cathode (CC) –** In the common cathode display, all the cathode connections of the LED segments are joined together to logic “0” or ground. The individual segments are illuminated by application of a “HIGH”, or logic “1” signal via a current limiting resistor to forward bias the individual Anode terminals (a-g).

Common Cathode 7-segment Display

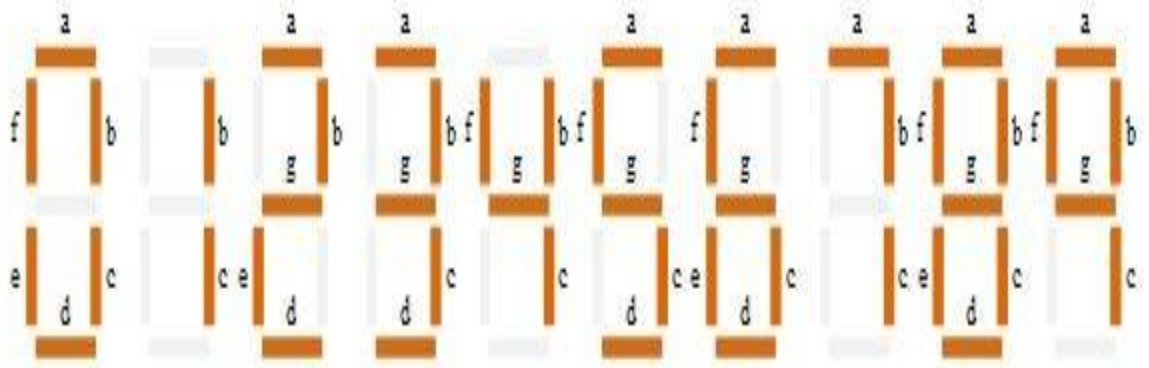


- 2. **The Common Anode (CA) –** In the common anode display, all the anode connections of the LED segments are joined together to logic “1”. The individual segments are illuminated by applying a ground, logic “0” or “LOW” signal via a suitable current limiting resistor to the Cathode of the particular segment (a-g).

Common Anode 7-segment Display



7-Segment Display Segments for all Numbers.



7-segment Display Truth Table

Decimal Digit	Individual Segments Illuminated						
	a	b	c	d	e	f	g
0	×	×	×	×	×	×	
1		×	×				
2	×	×		×	×		×
3	×	×	×	×			×
4		×	×			×	×
5	×		×	×		×	×
6	×		×	×	×	×	×
7	×	×	×				
8	×	×	×	×	×	×	×
9	×	×	×			×	×

Interrupts

- **Interrupt** is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor.
- The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.
- **Interrupts** can be internal or external.

8080A interrupts

There are five interrupt signals in the 8085 microprocessor:

1.TRAP: The TRAP interrupt is a non-maskable interrupt that is generated by an external device, such as a power failure or a hardware malfunction. The TRAP interrupt has the highest priority and cannot be disabled. (**A 'Maskable Interrupt' in computer science refers to an interrupt mechanism that can be enabled or disabled by the programmer. It is contrasted with a 'Non-maskable Interrupt' which has a higher priority and cannot be disabled, typically used for critical events like power loss.)**

2.RST 7.5: The RST 7.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the second highest priority.

3.RST 6.5: The RST 6.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the third highest priority.

4.RST 5.5: The RST 5.5 interrupt is a maskable interrupt that is generated by a software instruction. It has the fourth highest priority.

5.INTR: The INTR interrupt is a maskable interrupt that is generated by an external device, such as a keyboard or a mouse. It has the lowest priority and can be disabled.

Priority of Interrupts – When microprocessor receives multiple interrupt requests simultaneously, it will execute the interrupt service request (ISR) according to the priority of the interrupts.

8080A interrupts

- The processor has 5 interrupts. They are presented below in the order of their priority (from lowest to highest):

1. **INTR** is maskable 8080A compatible interrupt.
   - ❖ When the interrupt occurs the processor fetches from the bus one instruction, usually one of these instructions: One of the 8 RST instructions (RST^0 - RST^7).
   - ❖ The processor saves current program counter into stack and branches to memory location N * 8 (where N3-bit number from 0 to 7 supplied with the RST instruction).

8080A interrupts (Cont.)

2. **RST5.5** is a maskable interrupt.
   - ❖ When this interrupt is received the processor saves the contents of the PC register into stack and branches to 2CH (hexadecimal) address.

3. **RST6.5** is a maskable interrupt.
   - ❖ When this interrupt is received the processor saves the contents of the PC register into stack and branches to 34H (hexadecimal) address.

8080A interrupts (Cont.)

4. **RST7.5** is a maskable interrupt.
   - When this interrupt is received the processor saves the contents of the PC register into stack and branches to 3CH (hexadecimal) address.
5. **TRAP** is a non-maskable interrupt.
   - saves the contents of the PC register into stack and branches to 24H (hexadecimal) address.
     - All maskable interrupts RST 5.5, RST6.5 and RST7.5 interrupts can be enabled or disabled individually using SIM instruction.

RST Instructions

- In 8085 Instruction set, RSTn is actually standing for “Restart n”. And in this case, n has a value from 0 to 7 only.
- Thus the eight possible RST instructions are there, e.g. RST 0, RST 1, …, RST 7.
- They are 1-Byte call instructions. (A one-byte instruction includes a opcode and a operand in the same byte. Operand(s) are internal registers and are in the instruction in form of codes. If there is no numeral present in the instruction then that instruction will be of one-byte, for example, MOV C, A, RAL, and ADD B, etc.)
- Functionally RST n instruction is similar with:

  RST n = CALL n*8

RST Instructions (Cont.)

- For example, let us consider RST 4 is functionally equivalent to CALL 4*8, i.e. CALL 32 = CALL 0020H.
- The advantage of RST 2 is that it is only 1 Byte, whereas CALL 0010H is 3-Byte long.
- Thus RST instructions are useful for branching to frequently used subroutines.