Chapter 12
8085 Interrupts

Interrupts

- An interrupt is considered to be an emergency signal.
  - The Microprocessor should respond to it as soon as possible.

- When the Microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to an Interrupt Service Routine (ISR) to respond to the incoming interrupt.
  - Each interrupt will most probably have its own ISR.

Responding to Interrupts

- Responding to an interrupt may be immediate or delayed depending on whether the interrupt is maskable or non-maskable and whether interrupts are being masked or not.

- There are two ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored.
  - The vector is already known to the Microprocessor
  - The device will have to supply the vector to the Microprocessor

Interrupts

- Interrupt is a process where an external device can get the attention of the microprocessor.
  - The process starts from the I/O device
  - The process is asynchronous.
- Classification of Interrupts
  - Interrupts can be classified into two types:
    - Maskable Interrupts (Can be delayed or Rejected)
    - Non-Maskable Interrupts (Can not be delayed or Rejected)
- Interrupts can also be classified into:
    - Vectored (The address of the service routine is already known to the Microprocessor)
    - Non-vectored (the address of the service routine needs to be supplied externally by the device)

8085 Interrupt Process

The 8085 interrupt process can be described in terms of those eight steps.

Step 1: The interrupt process should be enabled by writing the instruction **EI** in the main program. The instruction EI sets the Interrupt Enable flip-flop. The instruction DI resets the flip-flop and disables the interrupt process.

Instruction EI (Enable Interrupt)

- This is a 1-byte instruction.
- The instruction sets the Interrupt Enable flip-flop and enables the interrupt process.
- System reset or an interrupt disables the interrupt process.

Instruction DI (Disable Interrupt)

- This is a 1-byte instruction.
- The instruction resets the Interrupt Enable flip-flop and disables the interrupt.
- It should be included in a program segment where an interrupt from an outside source cannot be tolerated.

Step 2: When the microprocessor is executing a program, it checks the INTR line during the execution of each instruction.

Step 3: If the line INTR is high and the interrupt is enabled, the microprocessor completes the current instruction, disables the Interrupt Enable flip-flop and sends a signal called INTA—Interrupt Acknowledge (active low). The processor cannot accept any interrupt requests until the interrupt flip-flop is enabled again.

8085 Interrupt Process

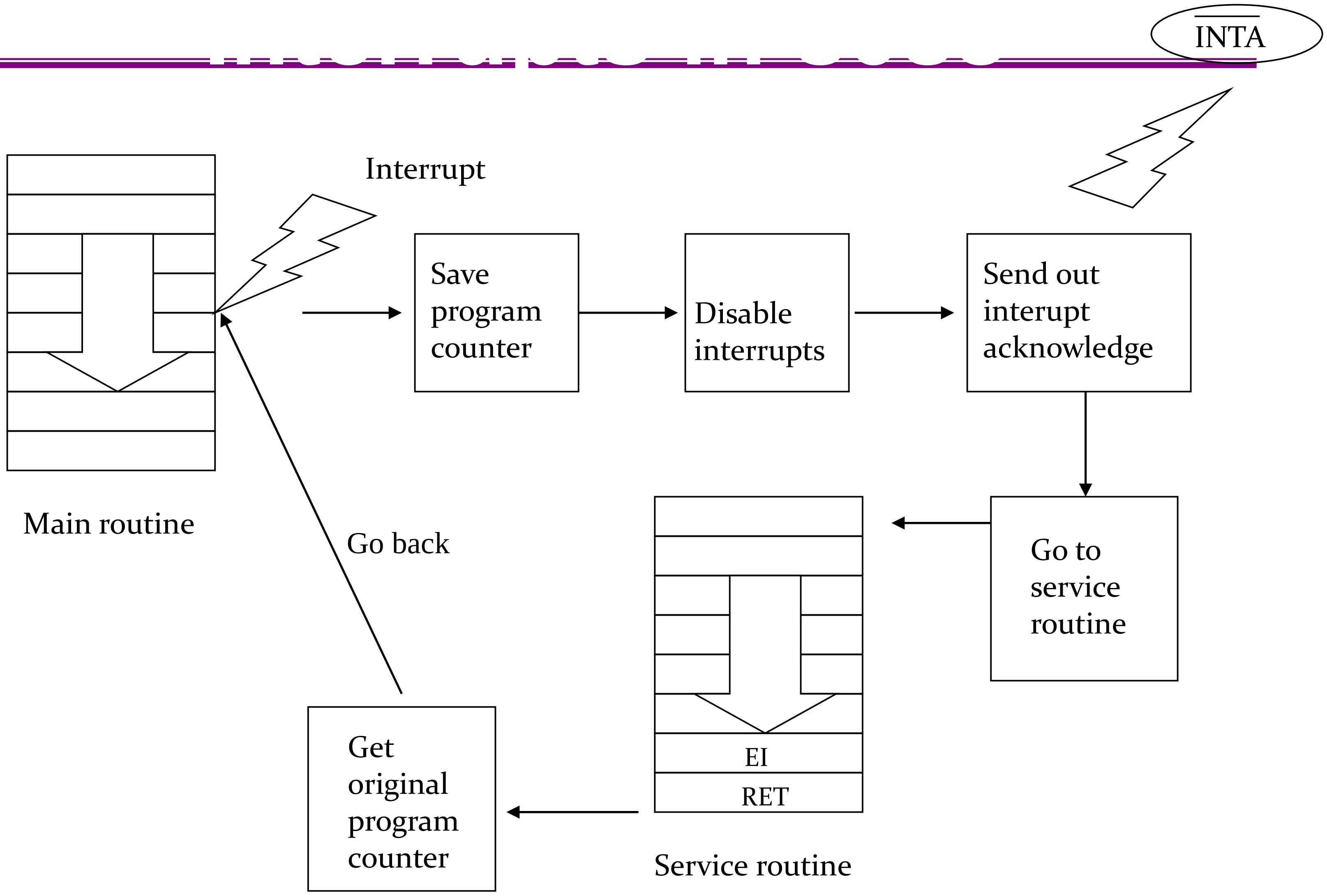
Step 4: The signal INTA is used to insert a restart (RST) instruction (or a Call instruction) through external hardware. The RST instruction is a 1-byte call instruction that transfers the program control to a specific memory location on page 00H and restarts the execution at that memory location after executing Step 5.

Step 5: When the microprocessor receives an RST instruction (or a Call instruction), it saves the memory address of the next instruction on the stack. This is similar to inserting a bookmark. The program is transferred to the CALL location.

Step 6: Assuming that the task to be performed is written as a subroutine at the specified location, the processor performs the task. This subroutine is known as a service routine.

Step 7: The service routine should include the instruction EI to enable the interrupt again. This is similar to putting the receiver back on the hook.

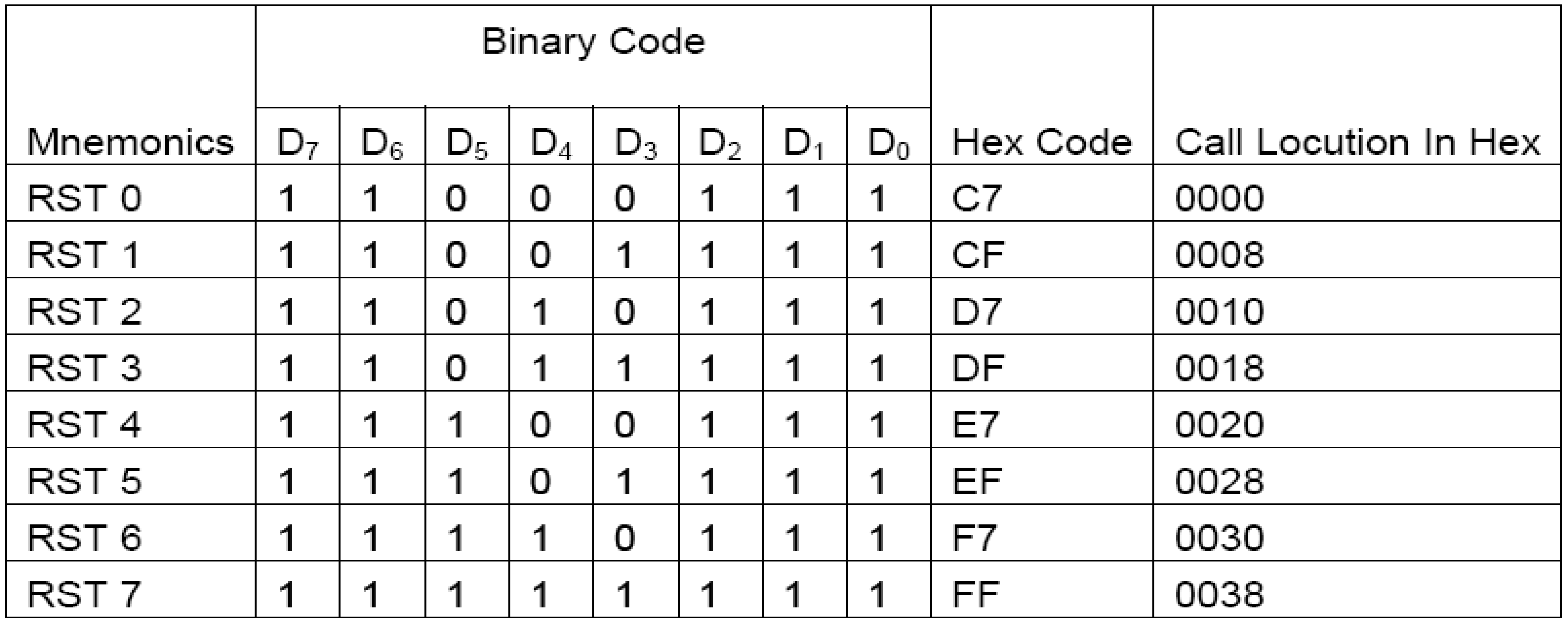
Step 8: At the end of the subroutine, the RET instruction retrieves the memory address where the program was interrupted and continues the execution.


$\overline{\text{INTA}}$
Interrupt
Save program counter
Disable interrupts
Send out interupt acknowledge
Main routine
Go back
Go to service routine
EI
RET
Get original program counter
Service routine

RST (RESTART) Instructions

- The 8085 recognizes 8 RST (RESTART) instructions: RST0 - RST7.
  - Special Restart Instruction used with interrupts.
  - It can be used as software instruction in a program to transfer program execution to one of the eight

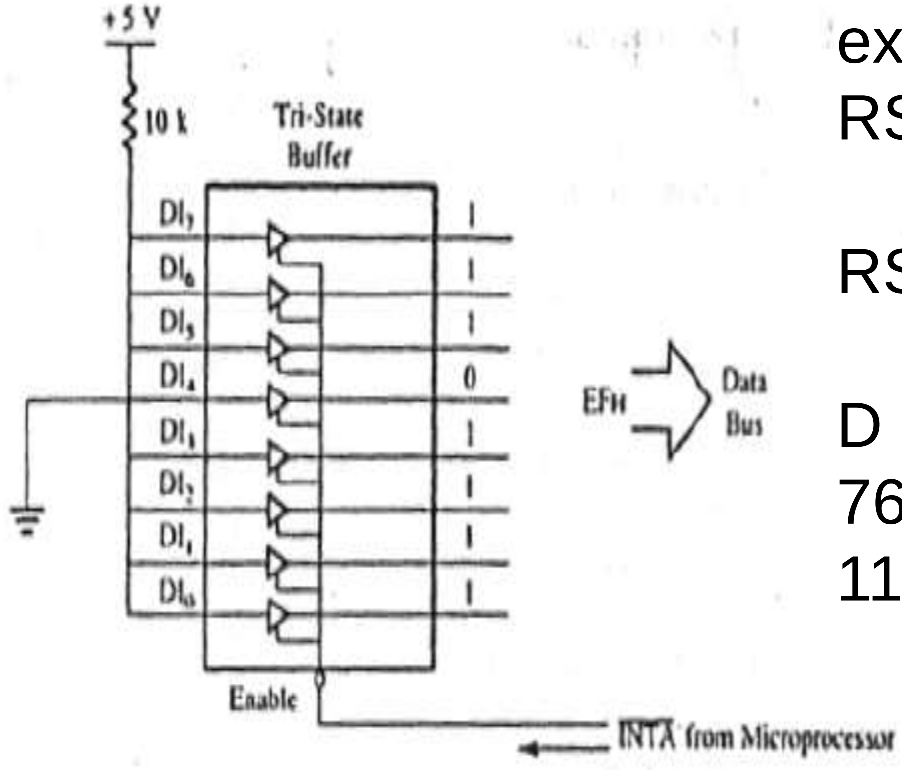
Table 3 *Restart Instructions*



	Binary Code								
Mnemonics	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Hex Code	Call Locution In Hex
RST 0	1	1	0	0	0	1	1	1	C7	0000
RST 1	1	1	0	0	1	1	1	1	CF	0008
RST 2	1	1	0	1	0	1	1	1	D7	0010
RST 3	1	1	0	1	1	1	1	1	DF	0018
RST 4	1	1	1	0	0	1	1	1	E7	0020
RST 5	1	1	1	0	1	1	1	1	EF	0028
RST 6	1	1	1	1	0	1	1	1	F7	0030
RST 7	1	1	1	1	1	1	1	1	FF	0038

Hardware Generation of RST Opcode

To implement Step 4 in the interrupt process, insert one of these instructions in the microprocessor by using external hardware and the signal $\overline{\text{INTA}}$ (Interrupt Acknowledge),



The following is an example of generating RST 5:

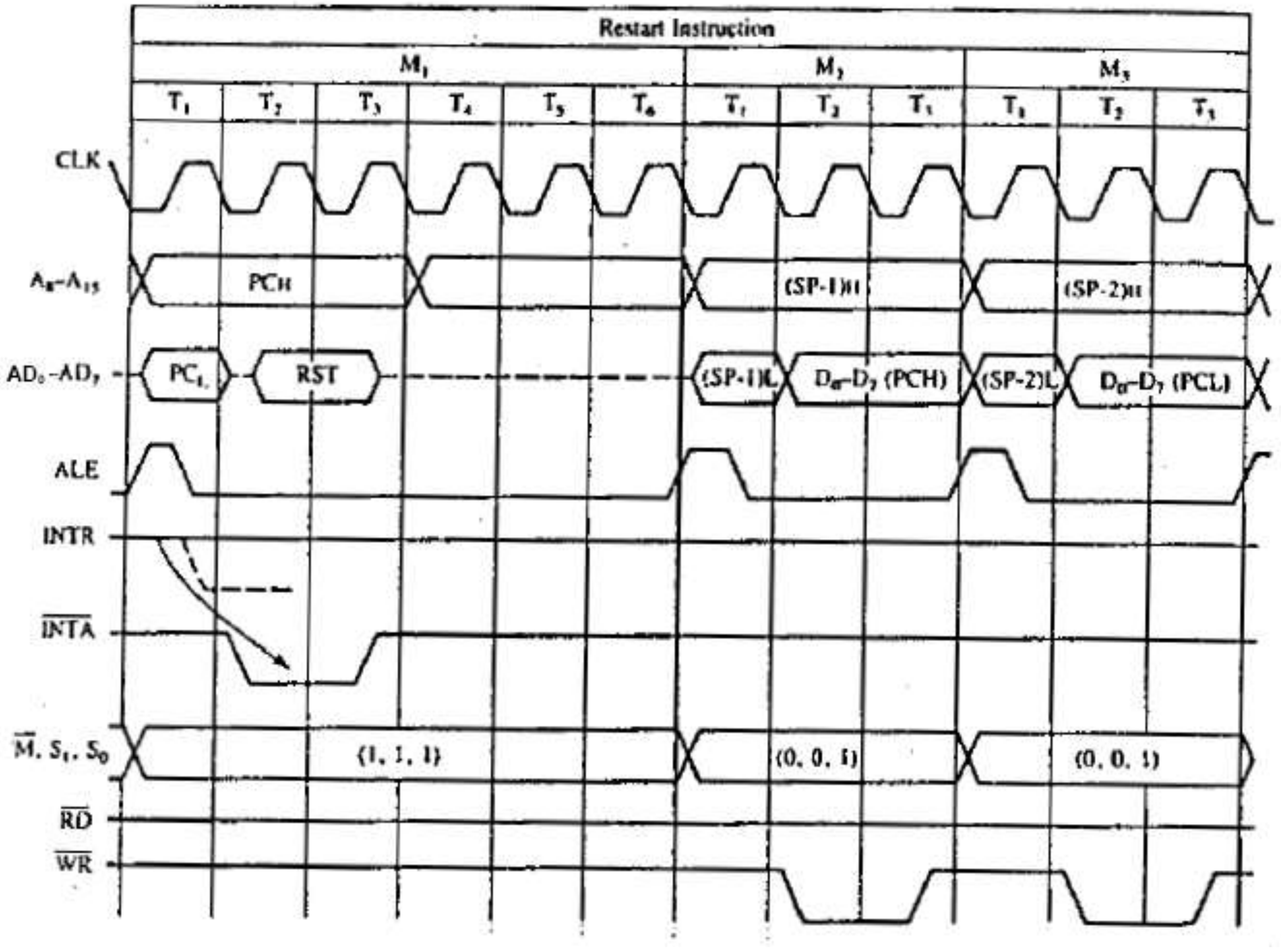
RST 5's opcode is EF =

D D
76543210
11101111

Restart Sequence

- The restart sequence is made up of three machine cycles
  - In the 1st machine cycle:
    - The microprocessor sends the INTA signal.
    - While INTA is active the microprocessor reads the data lines expecting to receive, from the interrupting device, the opcode for the specific RST instruction.
  - In the 2nd and 3rd machine cycles:
    - the 16-bit address of the next instruction is saved on the stack.
    - Then the microprocessor jumps to the address associated with the specified RST instruction.

RST Timing Sequence



Example-1

Illustration: An Implementation of the 8085 Interrupt PROBLEM STATEMENT

1. Write a main program to count continuously in binary with a one-second delay between each count.
2. Write a service routine at XX70H to flash FFH five times when the program is interrupted, with some appropriate delay between each flash.

MAIN PROGRAM

Memory ADDRESS	**Label**	**Mnemonics**	**Comments**
XX00		LXI SP, XX99H	;initialize stack pointer
03		EI	;Enable interrupt process
04		MVI A, 00H	;Initialize counter
06	NXTCNT:	OUT PORT 1	;display count
08		MV1 C, 01 H	;Parameter for 1-second delay
0A		CALL DELAY	;Wait one second
0D		INR A	;Next count
0E		JMP NXTCNT	; Continue

Delay Routine: Use delay subroutine

Service Routine

Memory ADDRESS	**Label**	**Mnemonics**	**Comments**
XX70	SERV:	PUSH B	;Save contents
71		PUSH PSW	
72		MVI B, 0AH	;Load register B for five flashes and five blanks
74		MVI A, 00 H	;Loads 00 to blank display
76	FLASH:	OUT PORT 1	
78		MVI C, 01H	; Parameter for 1 second delay
7A		CALL DELAY:	
7D		CMA	; Complement display count
7E		DCR B	;Reduce count
7F		JNZ FLASH	
82		POP PSW	
83		POP B	
84		EI	;Enable Interrupts process
85		RET	; service is complete, go back ;to main program

Description of the Interrupt Process

1. The main program initializes the stack pointer at XX99H and enables the interrupts. The program will count continuously from 00H to FFH, with a delay of one second between each count.
2. To interrupt the processor, push the switch. The INTR line goes high.
3. Assuming the switch is pushed when the processor is executing the instruction OUT at memory location XX06H, the following sequence of events occurs.
   a) The microprocessor completes the execution of the instruction OUT.
   b) It senses that the line INTR is high, and that the interrupt is enabled.
   c) The microprocessor disables the interrupt, stops execution, and sends out a control signal $\overline{\text{INTA}}$ (Interrupt Acknowledge).
   d) The $\overline{\text{INTA}}$ (active low) enables the tri-state buffer, and the instruction EFH is placed on the data bus.
   e) The microprocessor saves the address XX08H of the next instruction (MVI C, 01H) on the stack at locations XX98H and XX97H. and the program is transferred to memory location 0028H. The locations 0028-29-2AH should have the following Jump instruction to transfer the program to the service routine.

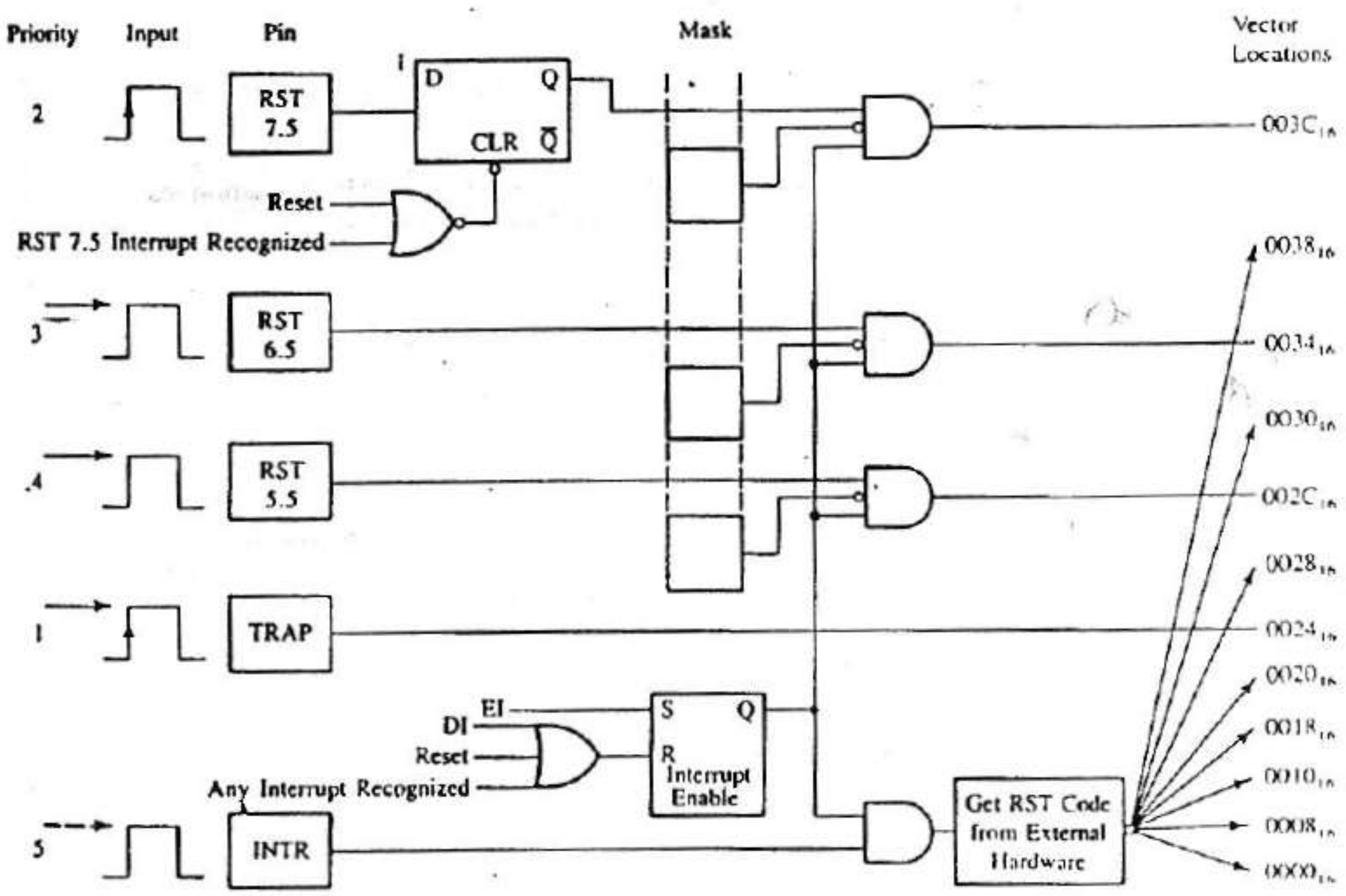
JMP XX70H

4. The program jumps to the service routine at XX70H.
5. The service routine saves the registers that are being used in the subroutine and loads the count ten in register B to output five flashes and also five blanks.
6. The service routine enables the interrupt before returning to the main program.
7. When the service routine executes the RET instruction, the microprocessor retrieves the memory address XX08H from the top of the stack and continues the binary counting.

The 8085 Interrupts

- The 8085 has 5 interrupt inputs.
  - The INTR input.
    - The INTR input is the only non-vectored interrupt.
    - INTR is maskable using the EI/DI instruction pair.
  - RST 5.5, RST 6.5, RST 7.5 are all automatically vectored
    - Transferred to specific location on memory page 00H without any external hardware .They do not require the INTA signal or an input port ; the necessary hardware is already implemented inside the 8085
    - RST 5.5, RST 6.5, and RST 7.5 are all maskable.
  - TRAP is the only non-maskable interrupt in the 8085
    - TRAP is also automatically vectored

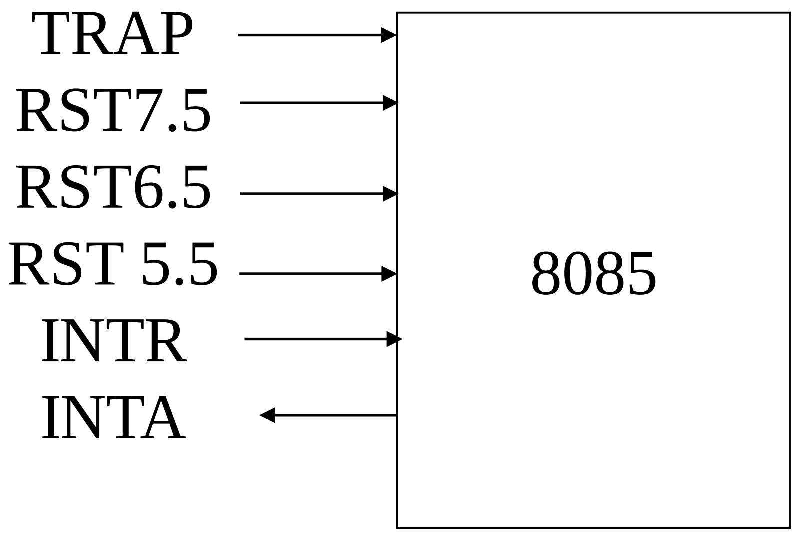
The TRAP has the highest priority, followed by RST 7.5, 6.5, 5.5. and INTR, in that order; however, the TRAP has a lower priority than the Hold signal used for DMA



The 8085 Interrupts

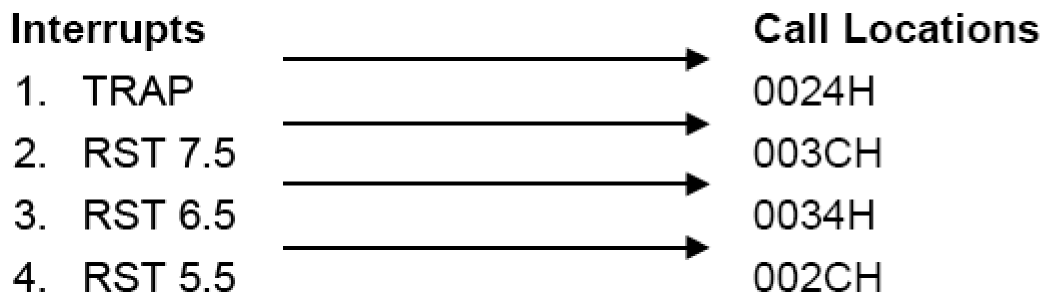
Interrupt name	Maskable	Vectored
INTR	Yes	No
RST 5.5	Yes	Yes
RST 6.5	Yes	Yes
RST 7.5	Yes	Yes
TRAP	No	Yes

8085 Interrupts



The 8085 Maskable/Vectored Interrupts

- The 8085 has 4 Masked/Vectored interrupt inputs.
  - TRAP, RST 5.5, RST 6.5, RST 7.5
    - They are automatically vectored according to the following table:



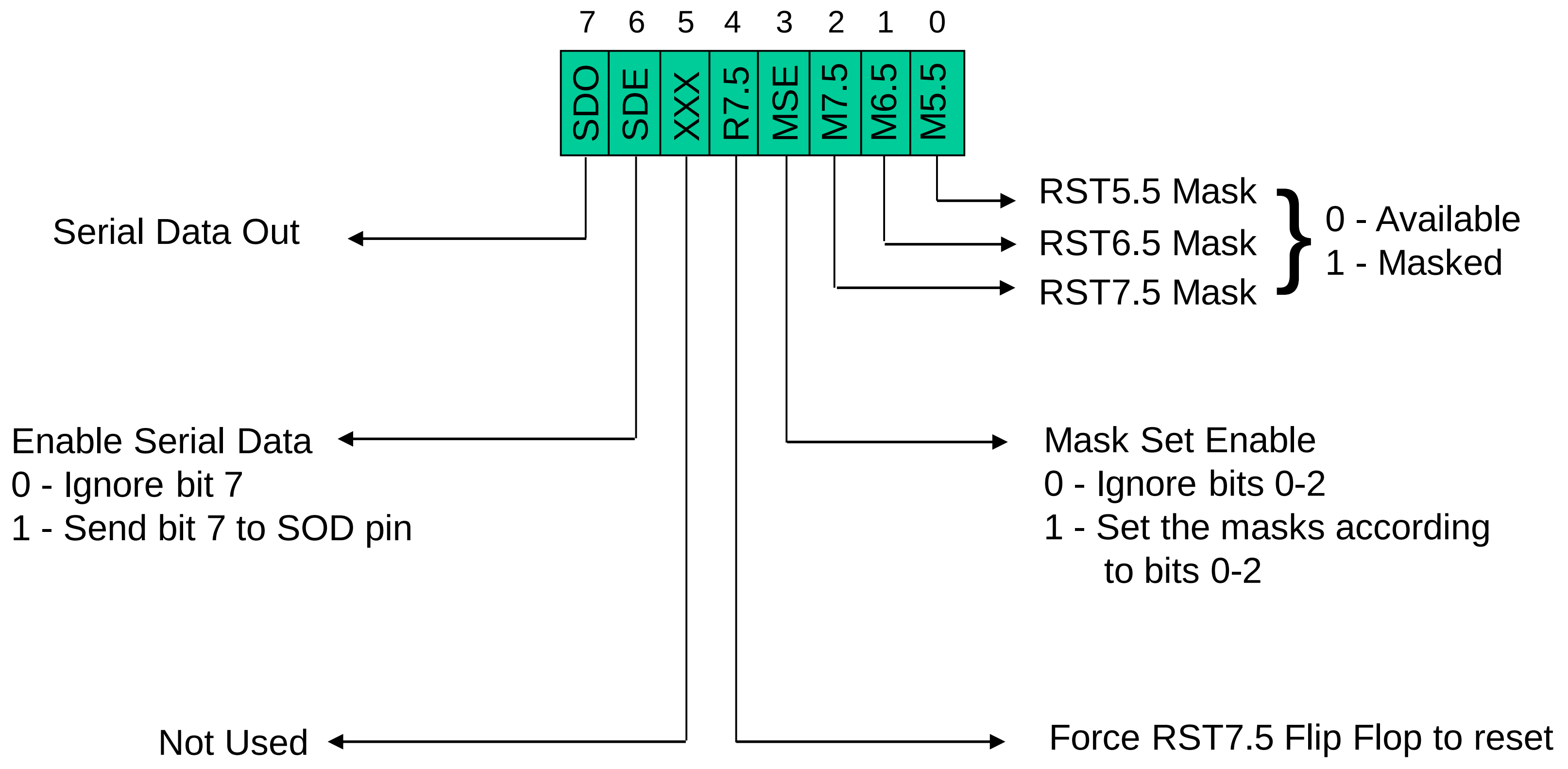
Interrupts	Call Locations
1. TRAP	0024H
2. RST 7.5	003CH
3. RST 6.5	0034H
4. RST 5.5	002CH

- Finding the address of these vectored interrupts are very easy . Just multiply 8 with the RST value i.e for RST 7.5 the subroutine (ISR) address=8*7.5=60=(3c)H.
- For TRAP ,its RST value is 4.5,then the subroutine address is 8*4.5=36=(24)H.
- Memory page for all interrupts are (00).

Manipulating the Masks

- The Interrupt Enable flip flop is manipulated using the EI/DI instructions.
- The individual masks for RST 5.5, RST 6.5 and RST 7.5 are manipulated using the SIM instruction.
  - This instruction takes the bit pattern in the Accumulator and applies it to the interrupt mask enabling and disabling the specific interrupts.

How SIM Interprets the Accumulator



SIM and the Interrupt Mask

- Bit 0 is the mask for RST 5.5, bit 1 is the mask for RST 6.5 and bit 2 is the mask for RST 7.5.
    - If the mask bit is 0, the interrupt is available.
    - If the mask bit is 1, the interrupt is masked.
- Bit 3 (Mask Set Enable - MSE) is an enable for setting the mask.
    - If it is set to 0 the mask is ignored and the old settings remain.
    - If it is set to 1, the new setting are applied.
- Bit 4 of the accumulator in the SIM instruction allows explicitly resetting the RST 7.5 memory even if the microprocessor did not respond to it.
- Bit 5 is not used by the SIM instruction
- Bit 6 & Bit 7 is used for extra functionality such as serial data transmission.

Using the SIM Instruction to Modify the Interrupt Masks

- Example: Set the interrupt masks so that RST5.5 is enabled, RST6.5 is masked, and RST7.5 is enabled.
  - First, determine the contents of the accumulator

- Enable 5.5	bit 0 = 0
- Disable 6.5	bit 1 = 1
- Enable 7.5	bit 2 = 0
- Allow setting the masks	bit 3 = 1
- Don’t reset the flip flop	bit 4 = 0
- Bit 5 is not used	bit 5 = 0
- Don’t use serial data	bit 6 = 0
- Serial data is ignored	bit 7 = 0

SDO	SDE	XXX	R7.5	MSE	M7.5	M6.5	M5.5
0	0	0	0	1	0	1	0

Contents of accumulator are: 0AH

```
EI              ; Enable interrupts including INTR
MVI A, 0A       ; Prepare the mask to enable RST 7.5, and 5.5, disable 6.5
SIM             ; Apply the settings RST masks
```

Example: Reset the 7.5 interrupt

<u>Instructions</u>

```
MVI A, 18H        ; Set D4 = 1
SIM               ; Reset 7.5 interrupt flip-flop
```

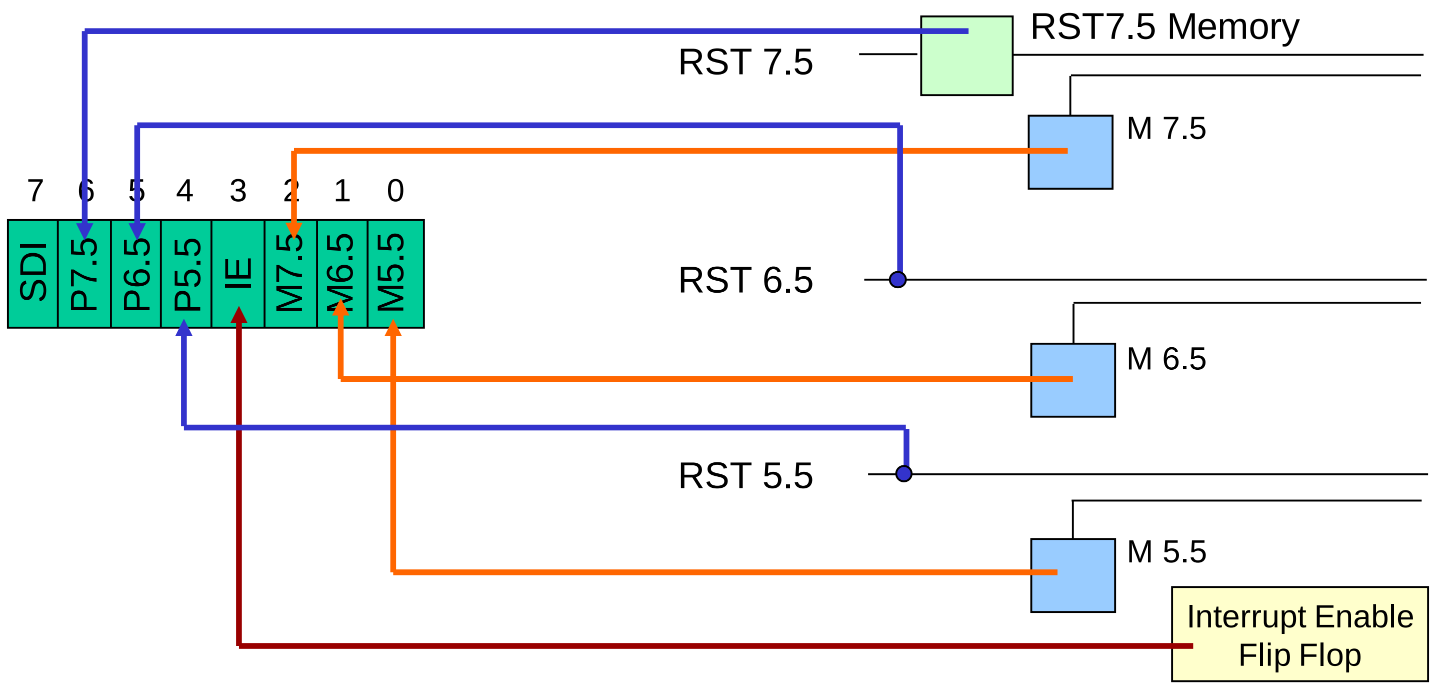
Triggering Levels

- RST 7.5 is positive edge sensitive.
  - When a positive edge appears on the RST7.5 line, a logic 1 is stored in the flip-flop as a "pending" interrupt.
  - Since the value has been stored in the flip flop, the line does not have to be high when the microprocessor checks for the interrupt to be recognized.
  - The line must go to zero and back to one before a new interrupt is recognized.

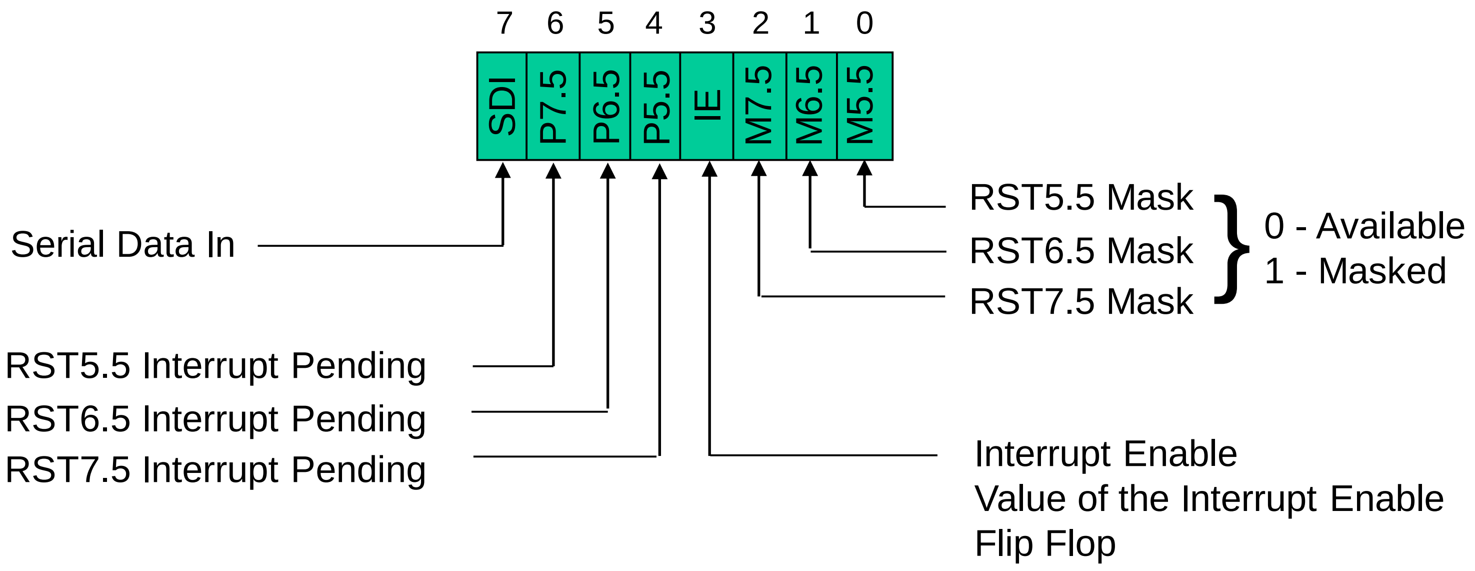
- RST 6.5 and RST 5.5 are level sensitive.
  - The interrupting signal must remain present until the microprocessor checks for interrupts.

Determining the Current Mask Settings

- RIM instruction: Read Interrupt Mask
  - Load the accumulator with an 8-bit pattern showing the status of each interrupt pin and mask.



How RIM sets the Accumulator’s different bits



The RIM Instruction and the Masks

- Bits 0-2 show the current setting of the mask for each of RST 7.5, RST 6.5 and RST 5.5
  - They return the contents of the three mask flip flops.
  - They can be used by a program to read the mask settings in order to modify only the right mask.

- Bit 3 shows whether the maskable interrupt process is enabled or not.
  - It returns the contents of the Interrupt Enable Flip Flop.
  - It can be used by a program to determine whether or not interrupts are enabled.

The RIM Instruction and the Masks

- Bits 4-6 show whether or not there are pending interrupts on RST 7.5, RST 6.5, and RST 5.5
  - Bits 4 and 5 return the current value of the RST5.5 and RST6.5 pins.
  - Bit 6 returns the current value of the RST7.5 memory flip flop.
- Bit 7 is used for Serial Data Input.
  - The RIM instruction reads the value of the SID pin on the microprocessor and returns it in this bit.

Pending Interrupts

- Since the 8085 has five interrupt lines, interrupts may occur during an ISR and remain pending.
  - Using the RIM instruction, it is possible to can read the status of the interrupt lines and find if there are any pending interrupts.

Example

Assuming the microprocessor is completing an RST 7.5 interrupt request, check to see if RST 6.5 is pending. If it is pending, enable RST 6.5 without affecting any other interrupts; otherwise, return to the main program.

Solution:

Instructions

```
          RIM         ;Read interrupt mask
          MOV B,A     ;Save mask information
          ANI 20H     ;Check whether RST 6.5 is pending
          JNZ NEXT
          EI
          RET         ;RST 6.5 is not pending, return to main program
NEXT:     MOV A,B     ;Get bit pattern; RST 6.5 is pending
          ANI 0DH     ;Enables RST 6.5 by setting D1 = 0
          ORI 08H     ;Enable SIM by setting D3 = 1
          SIM
          JMP SERV    ;Jump to service routine for RST 6.5
```

The instruction RIM checks for a pending interrupt. Instruction ANI 20H masks all the bits except D_5 to check pending RST 6.5. If $D_5 = 0$, the program control is transferred to the main program. $D_5 = 1$ indicates that RST 6.5 is pending. Instruction ANI 0DH sets $D_1 = 0$ (RST 6.5 bit for SIM), instruction ORI sets $D_3 = 1$ (this is necessary for SIM to be effective), and instruction SIM enables RST 6.5 without affecting any other interrupts. The JMP instruction transfers the program to the service routine (SERV) written for RST 6.5.

TRAP

- TRAP is the only non-maskable interrupt.
  - It does not need to be enabled because it cannot be disabled.
- It has the highest priority amongst interrupts.
- It is edge and level sensitive.
  - It needs to be high and stay high to be recognized.
  - Once it is recognized, it won't be recognized again until it goes low, then high again.
- TRAP is usually used for power failure and emergency shutoff.

The 8085 Interrupts

Interrupt Name	Maskable	Masking Method	Vectored	Memory	Triggering Method
INTR	Yes	DI / EI	No	No	Level Sensitive
RST 5.5 / RST 6.5	Yes	DI / EI SIM	Yes	No	Level Sensitive
RST 7.5	Yes	DI / EI SIM	Yes	Yes	Edge Sensitive
TRAP	No	None	Yes	No	Level & Edge Sensitive