UNIT 4

Chapter 3

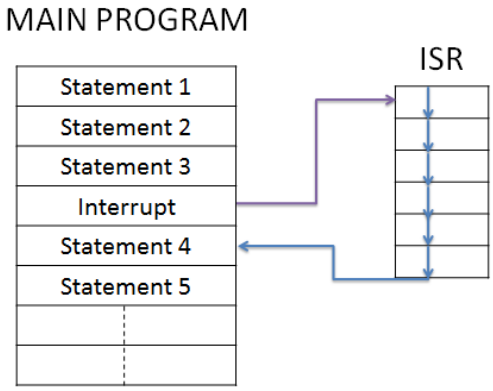
INTERRUPTS

3.1 INTRODUCTION

An interrupt is a subroutine (sequence of instructions) called & initiated by external device (hardware interrupt) or microprocessor itself (software interrupt).

It is a signal, which suspends the normal sequence of microprocessor & then µp gives service to that device which has given this signal. After completing the service the µp again returns to the main program.

Microprocessor is connected to different peripheral devices and to communicate with them. An interrupt is an input signal which transfers control to specific routine called Interrupt Service Routine (ISR). After executing ISR the control is again transferred to the main program.



Interrupt requests can be classified into two categories: **Maskable interrupts and Nonmaskable interrupts**. There is only one non-maskable hardware interrupt – TRAP and four maskable hardware interrupts – RST 7.5, RST 6.5, RST 5.5 and INTR.

Among the four Maskable interrupts INTR is non-vectored, which requires external hardware to supply a Call location to start the execution. The other three are vectored to specific locations.

The microprocessor can ignore or delay a maskable interrupt requests if it is performing some critical task. Whereas it has to respond to a non-maskable request immediately.

3.2 THE 8085 INTERRUPT

The 8085 Interrupt process is controlled by the Interrupt Enable flip-flop in the processor. This flip-flop can be set or reset using software instructions. If this flip-flop is enabled and the input interrupt signal INTR goes high then microprocessor is interrupted. INTR is maskable interrupt and can be disabled.

3.2.1 Steps in Interrupt Process

The 8085 interrupt process can be described in terms of following 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. While the instruction DI resets the flip-flop and disables the interrupt process. (EI and DI instructions are explained later)

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

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

Step 4: The signal $\overline{\text{INTA}}$ is used to insert a restart (RST) 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 Step 5.

Step 5: When microprocessor receives an RST instruction (or 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 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 Interrupt Service Routine (ISR).

Step 7: The ISR should include the instruction EI to enable the interrupt flip-flop again.

Step 8: At the end of the ISR, the RET instruction should be written. This retrieves the memory address where the program was interrupted (return to main program from where left) and continues the execution.

3.2.2 Instructions to Enable and Disable the Interrupt Process

Instruction	Meaning	Bytes	Addressing mode	Explanation
EI	Enable Interrupt	1	Implied	• This instruction sets the Interrupt Enable flip-flop and enables the interrupt process. • System reset or interrupt disables the interrupt process.
DI	Disable Interrupt	1	Implied	• This instruction resets the Interrupt Enable flip-flop and disables the interrupt. • It should be included in a program where an interrupt from outside source cannot be tolerated.

3.2.3 RST (Restart) Instructions

- The 8085 instruction set includes eight RST instructions (RST0 to RST7).
- These are 1-byte Call instructions that transfers the program execution to a specific location on page 00H.
- Whenever any RST instruction is executed, the address in Program Counter (address of instruction next to RST) is stored on stack before the program execution is transferred to the RST call location.
- When the processor encounters a RET (return) instruction, the program returns to the address that was stored on stack.
- In case of hardware interrupt, RST instruction is used to restart the program execution.

Mnemonics	Hex Code	Call location in Hex
RST0	C7	0000
RST1	CF	0008
RST2	D7	0010
RST3	DF	0018
RST4	E7	0020
RST5	EF	0028
RST6	F7	0030
RST7	FF	0038

We can insert RST instruction using signal $\overline{\text{INTA}}$

Example:

The instruction RST5 is built using resistors and tri-state buffer. In response to INTR high signal, the 8085 sends $\overline{\text{INTA}}$ low signal, which is used to enable the buffer and the RST5 (EF H) instruction is placed on data bus. This input data is taken in during fetch machine cycle, hence will be treated as instruction RST5. As soon as RST5 is detected the program execution is transferred to location 0028 H.

Difference when **RST instruction is written in program** and **when inserted during $\overline{\text{INTA}}$ low**

When RST instruction is written in memory as part of a program	**When RST instruction is inserted in microprocessor through input using $\overline{\text{INTA}}$ signal**
$\overline{\text{RD}}$ Signal is sent out	$\overline{\text{INTA}}$ Signal is sent out
$IO/\overline{M} = 0$, $S_1 = 1$, $S_0 = 1$ (fetch instruction from memory)	$IO/\overline{M} = 1$, $S_1 = 1$, $S_0 = 1$ (fetch instruction from Input)

3.2.4 Illustration: An Implementation of the 8085 Interrupt

a) Problem Statement:

1. Write a main program to count continuously in binary with a 1 second delay.
2. Write a service routine at 7070H to flash FFH five times when the program is interrupted, with some appropriate delay between each flash

Main Program:

Memory Address	Label	Mnemonics	Comments
7000		LXI SP, 7050H	Initialize Stack Pointer
7003		EI	Enable interrupt process
7004		MVI A, 00H	Initialize Accumulator at 0 (to count from 0 onwards)
7006	LOOP	OUT PORT1	Display count at output
7008		CALL DELAY	Wait 1 second
700B		INR A	Increment accumulator by 1 for next count
700C		JMP LOOP	Continue

Delay Subroutine:

Memory Address	**Label**	**Mnemonics**	**Explanation**
7100	**DELAY**	MVI B, countM	Load multiplier count in reg B
7102	LOOP2	MVI C, countD	Load delay count in reg B
7104	LOOP1	DCR C	Decrement delay count by 1
7105		JNZ LOOP1	Jump to LOOP1 until delay count becomes 0
7108		DCR B	Decrement multiplier count by 1
7109		JNZ LOOP2	Jump to LOOP1 until multiplier count becomes 0
710C		RET	Return to Main Program

Interrupt Service Routine (ISR):

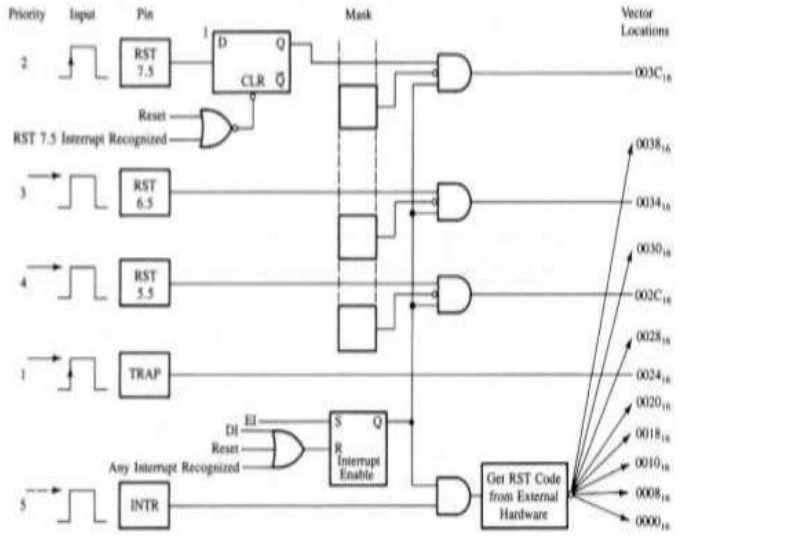
Memory Address	Label	Mnemonics	Comments
7070	**SERV:**	PUSH B	Save contents of BC pair (delay) on Stack
7071		PUSH PSW	Save contents of accumulator on Stack
7072		MVI D, 05H	Load reg D for count of 5
7074	FLASH	XRA A	Make accumulator empty for blank display
7075		OUT PORT1	Display accumulator at output
7077		CALL DELAY	Wait 1 second
707A		CMA	Complement accumulator to make it FFH
707B		OUT PORT1	Display accumulator at output
707D		CALL DELAY	Wait 1 second
7080		DRC D	Decrement count
7081		JMP FLASH	Continue
7084		POP PSW	Retrieve content of Accumulator from stack
7085		POP B	Retrieve content of BC reg pair from stack
7086		EI	Enable Interrupt process
7087		RET	Return from where left incomplete

3.3 8085 VECTORED INTERRUPT

Microprocessor 8085 has four vectored hardware interrupts. They are given below in decreasing order of priority.

Interrupts	**Call / Vector location**
TRAP	0024H
RST 7.5	003CH
RST 6.6	0034H
RST 5.5	002CH

In hardware interrupts TRAP has the highest priority, followed by RST 7.5, RST 6.5, RST 5.5 and INTR. In these interrupts only INTR is non-vectored interrupt.



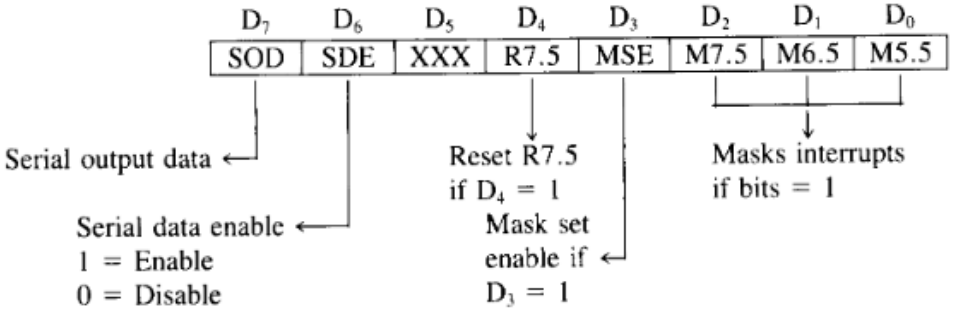
3.3.1 TRAP

- Non-maskable interrupt
- Analogous to smoke detector where signal cannot be ignored
- Has highest priority among hardware interrupts.
- It need not be enabled and cannot be disabled
- It is level and edge sensitive, meaning that the signal should go high and remain high to be acknowledged. Also it cannot be acknowledged again until it goes low and makes transition from low to high.
- When this interrupt is received, the program control is transferred to location 0024H.
- TRAP is generally used for critical events such as power failure and emergency shut-off.

3.3.2 RST 7.5, RST 6.5, RST 5.5

- Maskable interrupts.
- Among these interrupts RST 7.5 has highest priority, RST 6.5 has intermediate priority and RST 5.5 has the lowest priority.
- Are needed to be enabled under program control with two instructions EI and SIM.
- The below diagram shows the interpretation of the Accumulator Bit pattern for SIM instruction

SIM: Set Interrupt Mask



- 1-byte instruction and can be used for different functions.
- The function of SIM instruction is to set mask for RST 7.5, 6.5, 5.5 interrupts. SIM instruction reads content of accumulator and enables or disables the interrupt according to the content of the accumulator.
    - Bit D_3 is a control bit and should be at logic 1 for bits D_0, D_1 and D_2 to be effective.
    - If $D_0 = 1$ then RST 5.5 is disabled, if $D_1 = 1$ then RST 6.5 is disabled and if $D_2 = 1$ then RST 7.5 is disabled.
- The second function is to reset RST 7.5 flip-flop. If bit $D_4 = 1$ then RST 7.5 flip flop is reset.
- The third function is to implement serial I/O. Bits D_7 and D_6 of accumulator are used for serial I/O and do not affect interrupts. If bit $D_6 = 1$ it enables serial I/O and bit D_7 is transmitted as output bit.

RST 7.5

- Positive edge sensitive
- Can be triggered with short pulse
- The request is stored internally by D flip-flop until the microprocessor responds to the request or until it is cleared by Reset or by bit D_4 in SIM instruction.

RST 6.5 and 5.5

- Level sensitive
- Triggering level should be on until microprocessor completes the execution of current instruction.
- If the microprocessor is unable to respond to these interrupt requests immediately, they should be stored (kept pending) by external hardware.

Illustration 1: To enable all the RST interrupts in 8085

```
EI              ; Enable interrupt process
MVI  A, 08H     ; Load bit pattern to enable RST 7.5, 6.5, 5.5 (refer SIM instruction diag.)
SIM             ; Enable RST 7.5, 6.5, 5.5
```

Bit D_3 = 1 (MSE = 1) in accumulator makes the instruction SIM functional.
Bits D_2, D_1 and D_0 enable interrupts 7.5, 6.5 and 5.5

Illustration 2: To reset RST 7.5 interrupt

```
EI              ; Enable interrupt process
MVI  A, 18H     ; Load bit pattern to enable RST 7.5, 6.5, 5.5 and also reset RST7.5
SIM             ; Enable RST 7.5, 6.5, 5.5 and Reset RST 7.5
```

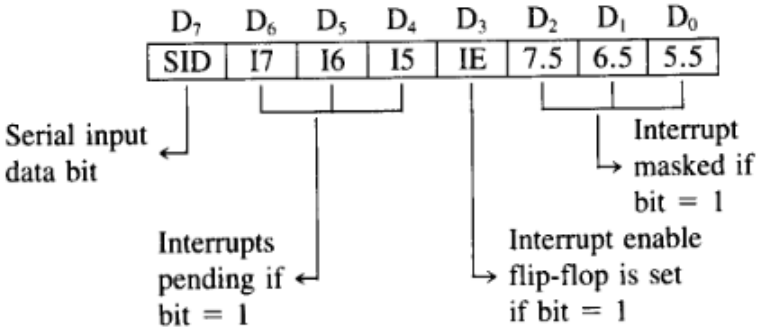
Bit D_3 = 1 (MSE = 1) in accumulator makes the instruction SIM functional.
Bits D_2, D_1 and D_0 enable interrupts 7.5, 6.5 and 5.5
Bit D_4 = 1 resets the flip-flop RST 7.5

Illustration 3: To enable only RST 7.5 interrupt

```
EI              ; Enable interrupt process
MVI  A, 0BH     ; Load bit pattern to enable RST 7.5 and mask RST 6.5, 5.5
SIM             ; Enable RST 7.5, Mask 6.5, 5.5
```

Bit D_3 = 1 (MSE = 1) in accumulator makes the instruction SIM functional.
Bit D_2 = 0 enables RST7.5, D_1 = 1 masks RST6.5 and D_0 =1 masks RST5.5

RIM: Read Interrupt Mask



- 1-byte instruction and can be used for following functions.
- The function of RIM instruction is to read interrupt masks. RIM instruction loads the accumulator with 8-bit data indicating the current status of the interrupt masks.
    - Bit $D_3 = 1$ indicates interrupt process is enabled.
    - Bit $D_2 = 1$ indicates RST7.5 is masked, Bit $D_1 = 1$ indicates RST6.5 is masked and Bit $D_0 = 1$ indicates RST5.5 is masked.
- To identify pending interrupts.
    - Bit $D_6 = 1$ indicates RST7.5 is pending, Bit $D_5 = 1$ indicates RST6.5 is pending and Bit $D_4 = 1$ indicates RST5.5 is pending.
- To receive data from SID pin through bit D_7.

Problem: Assuming the microprocessor is completing an RST7.5 interrupt request, check

3.4 RESTART AS SOFTWARE INSTRUCTIONS,

External hardware is necessary to insert an RST instruction (RST0 to RST7) when interrupt is given to INTR. However these instructions can also be given as software interrupts by writing them in program.

- RST instructions are commonly used to set up break-points as a debugging technique.
- A breakpoint is a RST instruction in a program where the execution of the program stops temporarily and program control is transferred to vector location of that RST instruction.
- The program should be transferred from the RST vector location to the breakpoint service routine to allow the user to examine register or memory contents when specific keys are pressed.
- These instructions are like 1-byte Call to a specific fixed location called Restart vector location.
- For example: If RST6 instruction is written in a program, the program execution is transferred to location 0030H.
- The vector locations and hex code of RST instructions are as follows:

Mnemonics	Hex Code	Call location in Hex
RST0	C7	0000
RST1	CF	0008
RST2	D7	0010
RST3	DF	0018
RST4	E7	0020
RST5	EF	0028
RST6	F7	0030
RST7	FF	0038

3.4.1 Illustrative Program: Implementation of Breakpoint Technique

Problem Statement:
Implement a breakpoint facility at RST5 for user. When the user writes RST5 in the program, the program should –

1. Be interrupted at the instruction RST5
2. Display the accumulator content and flags when Hex key ‘A’ is pressed
3. Exit the breakpoint routine and continue execution when Zero key is pressed.

(Assume that when a keyboard subroutine KBRD is called, it returns the binary code of the pressed key in accumulator)

Solution

The breakpoint routine should display accumulator content and the flags when user writes RSt5 instruction in program. The technique used to display register contents after executing an instruction in user’s program is as follows:

1. Store the content of registers on Stack.
2. Assign a key from keyboard for accumulator display. In this case hex key ‘A’ is assigned.
3. Wait for the key to be pressed and retrieve the contents from stack by manipulating stack pointer when key is pressed.
4. Assign a key to return to user’s program. In this case hex key ‘0’ (zero) is assigned.

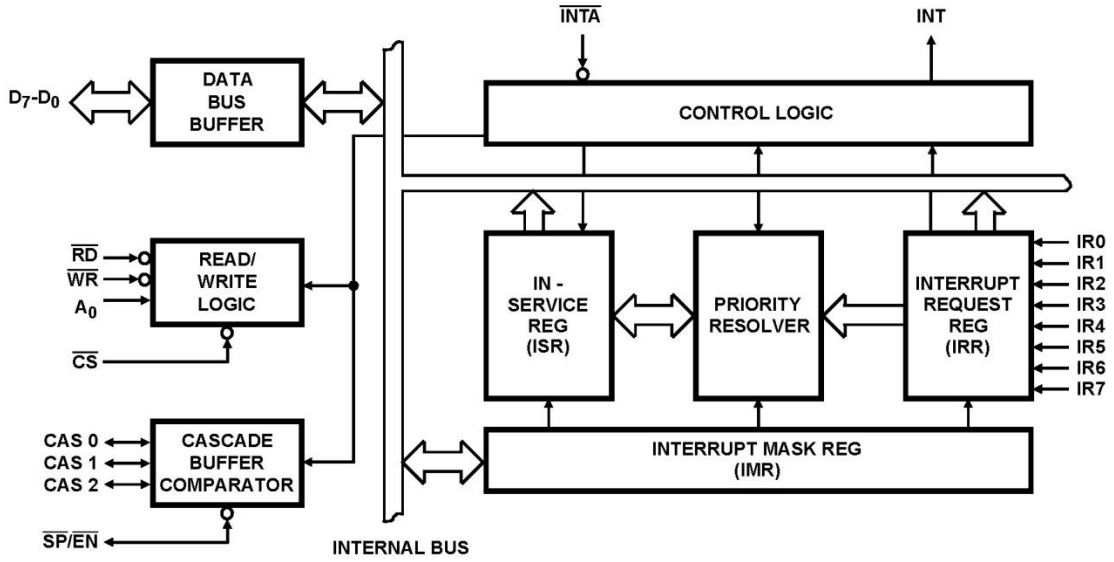
3.5 ADDITIONAL I/O CONCEPTS AND PROCESSES

The 8085 interrupt described earlier, is limited because of its single interrupt pin and hardware requirements to determine interrupt priorities. To overcome these limitations, a programmable interrupt controller such as 8259A is used to implement and the capability of the 8085 interrupt. Another I/O process, Direct Memory Access (DMA) is commonly used for high speed data transfer. For this DMA controller 8257 is used.

3.5.1 Programmable Interrupt Controller 8259A

- Programmable interrupt managing device
- Designed for use with interrupt signals (INTR/INT)
- It manages eight interrupt requests.
- It can vector an interrupt request anywhere in the memory map through program control without additional hardware for Restart instruction. However, all eight requests are spaced at the interval of four or eight locations.
- With additional 8259A devices the priority scheme can be expanded to 64 levels.

The 8259A Block Diagram:



1. One or more interrupt request lines go high requesting the service.
2. The 8259A resolves the priorities and sends an INT signal to MPU.
3. The MPU acknowledges the interrupt by sending $\overline{\text{INTA}}$.
4. After $\overline{\text{INTA}}$ has been received, the 8-bit opcode for the CALL instruction is placed on data bus along with 16-bit address of the interrupt vector location.
5. The program sequence of MPU is transferred to the memory location specified by the CALL instruction.

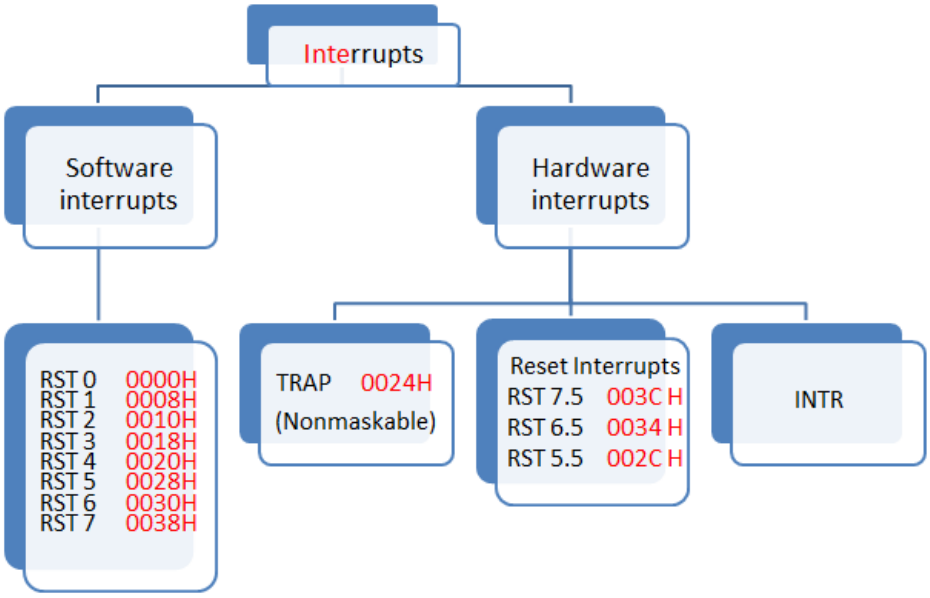
3.5.2 Direct Memory Access (DMA)

The Direct Memory Access (DMA) is a process of communication or data transfer controlled by DMA controller. Microprocessor controlled data transfer is too slow. DMA is generally used to speed up the process. The 8085 microprocessor has two pins available for this type of I/O communication: HOLD and HLDA.

HOLD is active high input signal from DMA to microprocessor, while HLDA is active high output signal from microprocessor to DMA.

- Whenever there is need of data transfer by DMA, DMA sends HOLD signal to microprocessor requesting the control of buses.
- As soon as microprocessor receives HOLD signal it finishes the current operation and releases the data bus and address bus in next machine cycle. It sends HLDA (hold acknowledge) signal to DMA.
- When Data transfer in complete then DMA sends signal to microprocessor by making HOLD line low.
- Then microprocessor regains the control of the buses and makes HLDA line low.

SUMMARY:



Hardware Interrupts (in order of highest to lowest priority):

Interrupts	Type	Requirements	Trigger	Vector
TRAP	Non-maskable	• Independent of EI and DI	Level and edge sensitive	0024H
RST 7.5	Maskable	• Controlled by EI and DI • Unmasked by SIM	Edge sensitive	003CH
RST 6.5	Maskable	• Controlled by EI and DI • Unmasked by SIM	Level sensitive	0034H
RST 5.5	Maskable	• Controlled by EI and DI • Unmasked by SIM	Level sensitive	002CH
INTR	Maskable	• Controlled by EI and DI	Level sensitive	Non-vectored

Difference between Hardware and Software interrupts:

Hardware Interrupts	Software Interrupts
Used to handle asynchronous events	Cannot be used to handle asynchronous events
Requested by external device	Requested by microprocessor itself through program.
After execution of these interrupts **program counter is not incremented**	After execution of these interrupts **program counter is incremented**
All hardware interrupts **are maskable except TRAP**	**All** software interrupts **are nonmaskable**
Lower priority than software interrupts	**Higher priority** than hardware interrupts
Improves throughput of the system	**Does not improve throughput** of the system
Affect interrupt control logic	**Does not affect** interrupt control logic
Microprocessor executes interrupt acknowledge cycle to acknowledge this interrupt	**Microprocessor does not execute any interrupt acknowledge cycle,** & only normal machine cycle is executed
Eg. TRAP, RST 7.5, RST 6.5, RST 5.5, INTR,	Eg. RST1, RST2, RST3, RST4, RST5, RST6, RST7,

Difference between Maskable and Non-maskable interrupts

Maskable Interrupts	Nonmaskable Interrupts
Can be masked or made pending	Cannot be masked or made pending
Can be disabled	Cannot be disabled
Lower priority than nonmaskable interrupts	**Higher priority** than maskable interrupts
May be **vectored or non-vectored**	**All are vectored**
Response time is **high (slow)**	Response time is **low (fast)**
Used to interface with peripheral devices	Used for emergency purpose. For eg. Power failure, smoke detector etc.
Eg. RST 7.5, RST 6.5, RST 5.5, INTR	Eg. RST1, RST2, RST3, RST4, RST5, RST6, RST7, TRAP