Notes for B.Tech.(Electronics) 5th sem on interrupts
EC-504 (unit-4)

Rachana Pathak
IEJU,Gwalior-474011

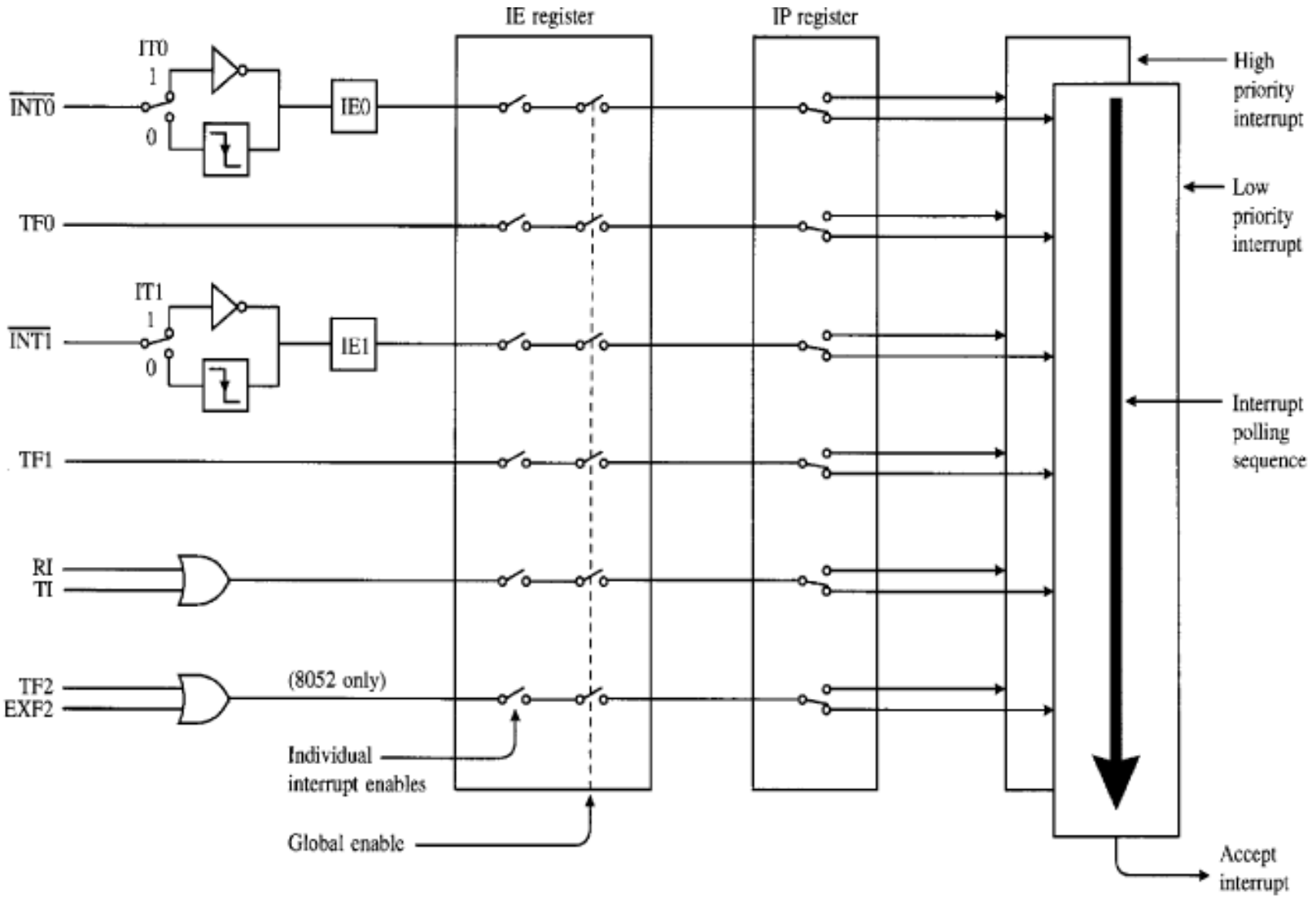
Interrupt / Interrupt sources

- It is a sub-routine calls that given by the microcontroller when some other program with high priority is requested for acquiring the system buses than interrupt occur in current running program.

- Interrupts provide a method to postpone or delay the current process, performs a sub-routine task and then restart the standard program again.

- 8051 has 5 sources of interrupts
  - External Interrupt 0 ($\overline{\text{INT0}}$)
  - External Interrupt 1 ($\overline{\text{INT1}}$)
  - Timer 0 overflow (TF0)
  - Timer 1 overflow (TF1)
  - Serial Port events (TI / RI)

Interrupt Structure



Interrupt Handling/ Interrupt Vector

- To distinguish between various interrupts and executing different code depending on what interrupt was triggered 8051may be jumping to a fixed address when a given interrupt occurs.

Interrupt Handling/ Interrupt Vector

Interrupt	Flag	Interrupt handler address
External 0	IE0	0003H
Timer0	TF0	000BH
External1	IE1	0013H
Timer1	TF1	001BH
Serial	RI/TI	0023H

- If Timer 0 overflows (i.e., the TF0 bit is set), the main program will be temporarily suspended and control will jump to 000BH if we have code at address 0003H that handles the situation of Timer 0 overflowing.

Setting Up Interrupts

- By default at power up, all interrupts are disabled. Even if, for example, the TF0 bit is set, the 8051 will not execute the interrupt. Your program must specifically tell the 8051 that it wishes to enable interrupts and specifically which interrupts it wishes to enable.
- Your program may enable and disable interrupts by modifying the IE register.
- **IE: Interrupt Enable Register**

EA	----	----	ES	ET1	EX1	ET0	EX0

- EA : Global interrupt enable.
- ES : Serial interface.
- **ET1** : Timer 1.
- EX1 : External interrupt 1.
- **ET0** : Timer 0.
- EX0 : External interrupt 0.

- 0 = Disabled.
- 1 = Enabled.

Setting Up Interrupts

- Each of the 8051’sinterrupts has its own bit in the IE register. You enable a given interrupt by setting the corresponding bit. For example, if you wish to enable Timer 1 Interrupt, you would execute either:

```
MOV IE,#08h || SETB ET1
```

- Both of the above instructions set bit 3 of IE, thus enabling Timer 1 Interrupt. Once Timer 1 Interrupt is enabled, whenever the TF1 bit is set, the 8051 will automatically put "on hold” the main program and execute the Timer 1 Interrupt Handler at address 001Bh. However, before Timer 1 Interrupt (or any other interrupt) is truly enabled, you must also set bit 7 of IE.

- Bit 7, the Global Interrupt Enable/Disable, enables or disables all interrupts simultaneously. That is to say, if bit 7 is cleared then no interrupts will occur, even if all the other bits of IE are set. Setting bit 7 will enable all the interrupts that have been selected by setting other bits in IE.

Continue…

- This is useful in program execution if you have time-critical code that needs to execute. In this case, you may need the code to execute from start to finish without any interrupt getting in the way. To accomplish this you can simply clear bit 7 of IE (CLR EA) and then set it after your time critical code is done.

- To enable the Timer 1 Interrupt execute the following two instructions:

```
SETB ET1

SETB EA
```

- Thereafter, the Timer 1 Interrupt Handler at 01Bh will automatically be called whenever the TF1 bit is set (upon Timer 1 overflow).

Polling Sequence

- The 8051 automatically evaluates whether an interrupt should occur after every instruction.
- When checking for interrupt conditions, it checks them in the following order:

  1) External 0 Interrupt

  2) Timer 0 Interrupt

  3) External 1 Interrupt

  4) Timer 1 Interrupt

  5) Serial Interrupt

Interrupt Priorities

- The 8051 offers two levels of interrupt priority: high and low. By using interrupt priorities you may assign higher priority to certain interrupt conditions. For example, you may have enabled Timer 1 Interrupt which is automatically called every time Timer 1 overflows.

- Additionally, you may have enabled the Serial Interrupt which is called every time a character is received via the serial port. However, you may consider that receiving a character is much more important than the timer interrupt. In this case, if Timer 1 Interrupt is already executing you may wish that the serial interrupt itself interrupts the Timer 1 Interrupt. When the serial interrupt is complete, control passes back to Timer 1 Interrupt and finally back to the main program. You may accomplish this by assigning a high priority to the Serial Interrupt and a low priority to the Timer 1 Interrupt.

- Interrupt priorities are controlled by the IP register . The IP register has the following format:

IP: Interrupt priority register

---	---	---	**PS**	**PT1**	**PX1**	**PT0**	**PX0**

- PS : Serial interface.
- **PT1** : Timer 1.
- PX1 : External interrupt 1.
- **PT0** : Timer 0.
- PX0 : External interrupt 0.

- 0 = Low priority.
- 1 = High priority.

interrupt priorities

- **When considering interrupt priorities, the following rules apply:**

- Nothing can interrupt a high-priority interrupt--not even another high priority interrupt.
- A high-priority interrupt may interrupt a low priority interrupt.
- A low-priority interrupt may only occur if no other interrupt is already executing.
- If two interrupts occur at the same time, the interrupt with higher priority will execute first. If both interrupts are of the same priority the interrupt which is serviced first by polling sequence will be executed first.

When an Interrupt Occurs?

- When an interrupt is triggered, the following actions are taken automatically by the microcontroller:
- The current Program Counter is saved on the stack, low-byte first.
- Interrupts of the same and lower priority are blocked.
- In the case of Timer and External interrupts, the corresponding interrupt flag is set.
- Program execution transfers to the corresponding interrupt handler vector address.
- The Interrupt Handler Routine executes. Take special note of the third step: If the interrupt being handled is a Timer or External interrupt, the microcontroller automatically clears the interrupt flag before passing control to your interrupt handler routine.

When an Interrupt Ends?

- An interrupt ends when your program executes the RETI instruction. When the RETI instruction is executed the following actions are taken by the microcontroller:

- Two bytes are popped off the stack into the Program Counter to restore normal program execution.

- Interrupt status is restored to its pre-interrupt status.