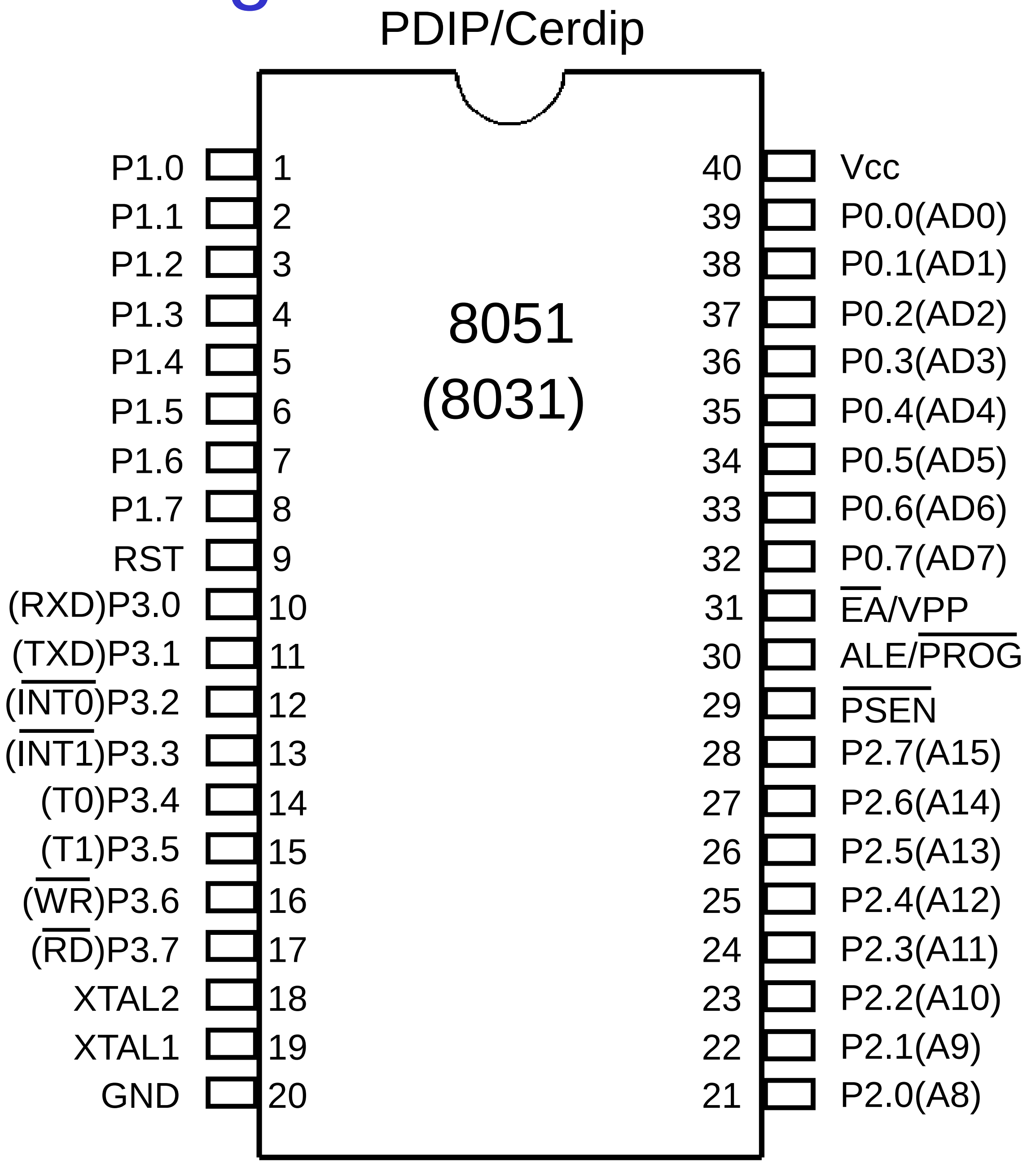
Module 7

8051 Interrupt Interface

8051 Pin Diagram



external hardware interrupt

Inside Architecture of 8051

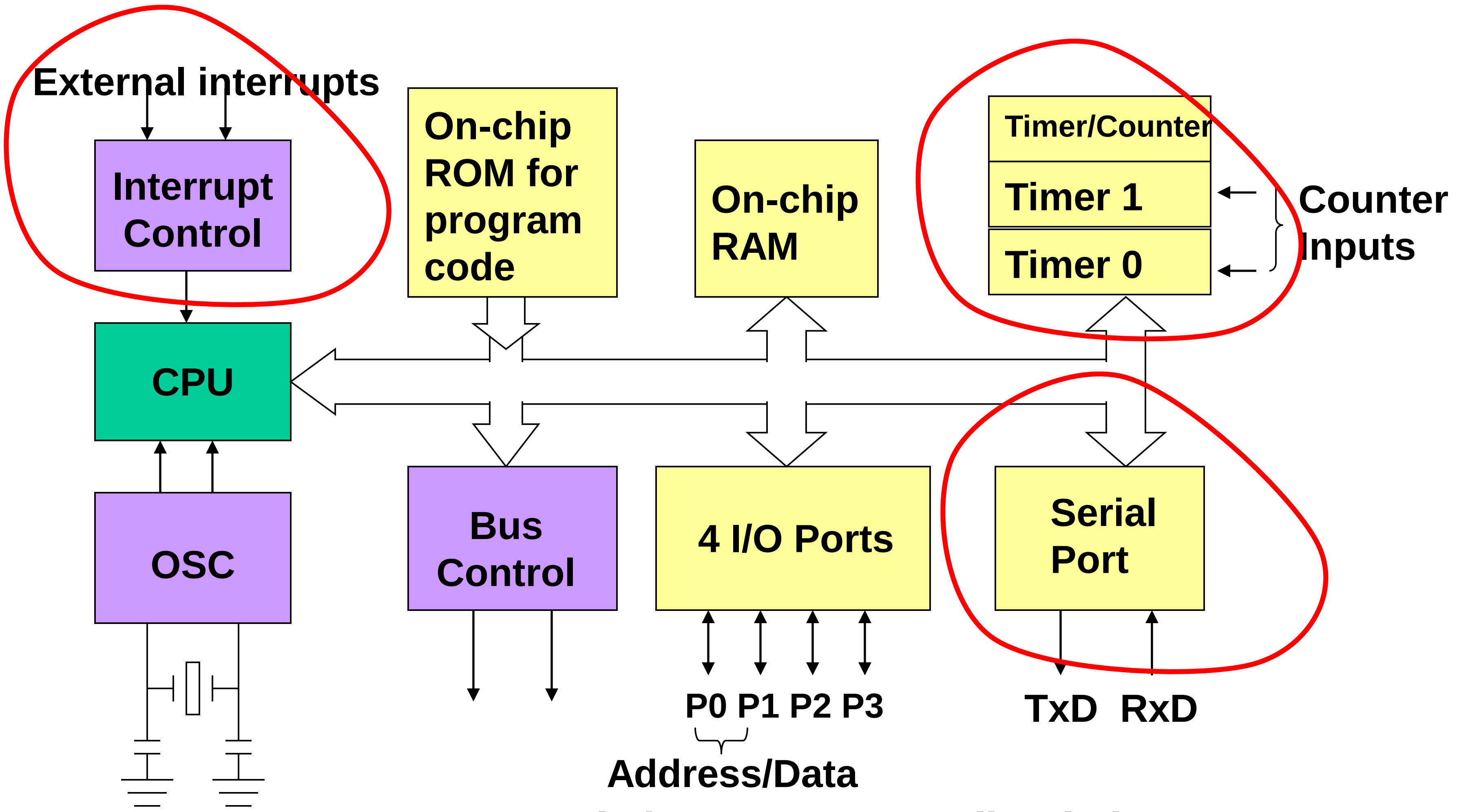


Figure 1-2. Inside the 8051 Microcontroller Block Diagram

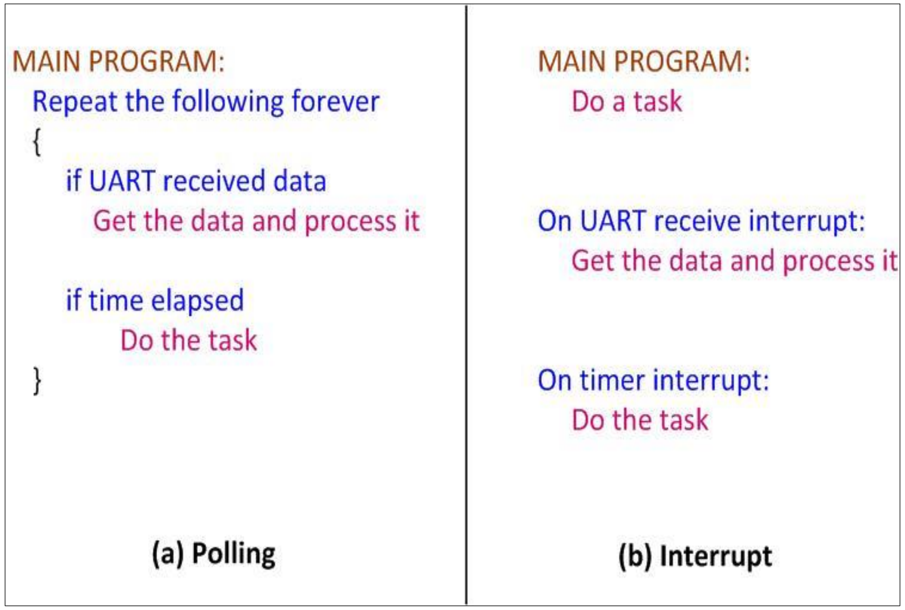
I/O Services

- A single microcontroller can serve several devices.
- Two ways:
  - Interrupt method
    - An interrupt is an external or internal event that interrupts the microcontroller to inform that a device needs its service.
  - Polling method

Polling vs Interrupt Driven

- The microcontroller continuously monitors the status of a given device.
- When the condition is met, it serves the device.
- After that, it moves on to monitor the next device until every one is serviced.
- The microcontroller checks all devices in a round-robin fashion.

- Whenever any device needs its service, the device notifies the microcontroller by sending it an interrupt signal.
- Upon receiving an interrupt signal, the microcontroller interrupts (suspends execution) whatever it is doing and serves the device.
- The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler.



```
MAIN PROGRAM:
  Repeat the following forever
  {
     if UART received data
        Get the data and process it

     if time elapsed
           Do the task
  }
```

(a) Polling

```
MAIN PROGRAM:
   Do a task


On UART receive interrupt:
   Get the data and process it


On timer interrupt:
   Do the task
```

(b) Interrupt

Interrupt Service Routine

- For every interrupt, there is a fixed location in memory that holds the address of its ISR.
- The group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table.

Interrupt	**ROM Location**	**Pin**
Reset	0000	9
External hardware interrupt 0 (INT0)	0003	P3.2 (12)
Timer 0 interrupt (TF0)	000B	
External hardware interrupt 1 (INT1)	0013	P3.3 (13)
Timer 1 interrupt (TF1)	001B	
Serial COM interrupt (RI or TI)	0023	

Steps in Executing an 8051 Interrupt

- Upon activation of an interrupt, the microcontroller goes through the following steps:
  1. It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack.
  2. It jumps to a fixed location in memory called the interrupt vector table.
  3. The microcontroller gets the address of the ISR from the interrupt vector table and jumps to it.
  4. The microcontroller starts to execute the interrupt service routine until it reaches the last instruction of the subroutine which is RETI (return from interrupt).
  5. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted.
  6. First, it gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC.
  7. Then it starts to execute from that address.

Six Interrupts in the 8051

- Reset
- Two interrupts for the timers
  - TF0, TF1
- Two interrupts for external hardware interrupts
  - INT0, INT1
- Serial communication
  - TI or RI
- Refer to Table

Six Interrupts in the 8051

- ROM address 0-30H contains the interrupt vector table
- There is a limited number of bytes for code for each interrupt.
  - 3 bytes for reset
  - 8 bytes for timers and external hardware interrupts
  - If the service routine is too long to fit the ISR, an LJMP instruction is placed in the vector table to point to the address of the ISR.
- Programmers must enable these interrupts before using them.

Redirecting the 8051 From the Interrupt Vector Table At Power-Up

```
;---- the first instruction is executed as the 8051 powers up
                ORG   0       ;ROM reset location
                LJMP MAIN  ;by-pass interrupt vector table
                               ; ISR entry point

;---- the wake-up program
                ORG   30H
MAIN:
                ....
                END
```

Larger Interrupt Service Routines

```
;---- the first instruction is executed as the 8051 powers up
            ORG    000              ;ROM reset location
            LJMP MAIN               ;by-pass interrupt vector table
            ORG  000Bh              ;Timer 0 entry point
           LJMP T0ISR
           ORG   0030H             ;above interrupt vectors
MAIN:
            ....
            END


T0ISR:


          RETI
```

IE (Interrupt Enable)

- In the 8051, the IE (interrupt enable) register denotes the usage of these interrupts.
  - Upon reset, all interrupts are disabled (masked).
  - The interrupts must be enabled by software.
  - IE is bit-addressable.
  - If we want to enable a special interrupt, set EA=1 first.
  - Enable each interrupt by setting its corresponding bit in IE.

D7							D0
EA	--	ET2	ES	ET1	EX1	ET0	EX0

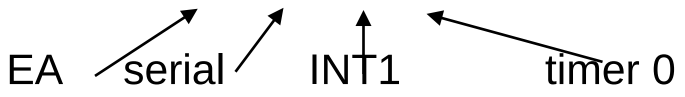
Example

Show the instructions to

(a) enable the serial interrupt, timer 0 interrupt, and external hardware interrupt 1 (EX1)

Solution:

(a) `MOV IE,#10010110B`



Another way to perform the "**MOV IE,#10010110B**" instruction is by using single-bit instructions as shown below.

```
SETB IE.7     ;EA=1, Global enable
SETB IE.4     ;enable serial interrupt
SETB IE.1     ;enable hardware interrupt 1
SETB IE.2     ;enable Timer 0
```

Programming Timer Interrupts

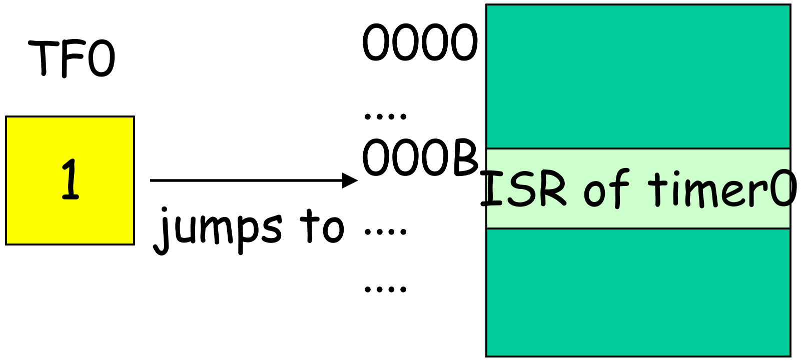
Roll-over Timer Flag and Interrupt

- In polling TF, we have to wait until the TF is raised.

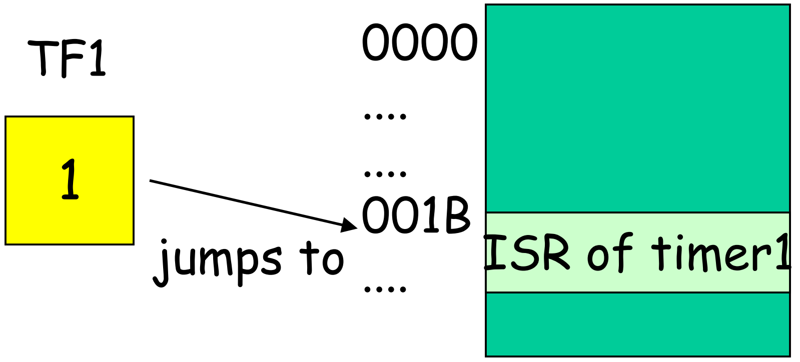
```
    JNB   TF, target
```

- Using interrupts, whenever TF is raised, the microcontroller is interrupted and jumps to the interrupt vector table to execute the ISR.
- When a timer interrupt is generated, the flag that generated it is cleared by the on-chip hardware when the service routine is vectored to.

Timer 0 Interrupt Vector: 000BH



Timer 1 Interrupt Vector: 001BH



Example – Two square waves using interrupts

- Write a program using interrupts to create 7 KHz and 500 Hz square waves on P1.7 and P1.6. Assume 12 Mhz clock.
- P1.7: half a period=71μs
  - Use timer 0 auto-reload mode and initialize with -71
- P1.6: half a period=1ms
  - Use timer 1 16-bit mode and initialize with -1000
  - Make sure to stop the timer,reinitialize, and rerun.

Solution

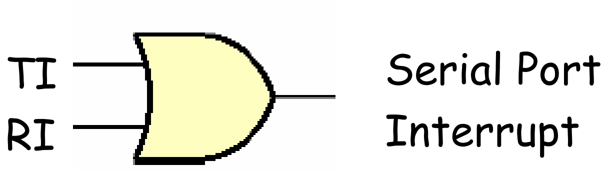
```
     ORG 0
     LJMP MAIN
     ORG 000Bh
     LJMP T0ISR
     ORG 001Bh
     LJMP T1ISR
     ORG 0030h
MAIN:MOV TMOD,#12h
     MOV TH0,#-71
     SETB TR0
     SETB TF1   ;force
; timer 1 interrupt
     MOV IE,#8Ah
     SJMP $
```

```
TOISR:CPL P1.7
      RETI


T1ISR:CLR TR1
     MOV TH1,#0FCh
     MOV TL1,#018h
     SETB TR1
     CPL P1.6
     RETI

     END
```

Programming the Serial Communication Interrupt

RI and TI Flags and Interrupts

- In the 8051 these is only one interrupt set aside for serial communication.
  - This interrupt is used to both send and receive data.



- If the interrupt bit IE.4 is enabled, when RI or TI is raised, the 8051 jumps to memory 0023H to execute the ISR.
- The last instruction before RETI is the clearing of the RI or TI flags.
  - Because the 8051 does not know who generated it.

Example

Write a program using interrupts to do the following:

(a) Receive data serially and send it to P0. (Interrupt-driven)

(b) Have P1 port read and transmitted serially (Interrupt-driven) and a copy given to P2. (main task)

(c) Make timer 0 generate a square wave of 5 kHz frequency on P3.6.

Assume that XTAL = 11.0592MHz. Set the baud rate at 4800 bits/s.

Solution:

Two interrupts must be set:

TF0 (address 000BH) for square wave: toggle P0.1 (c)

RI and TI (address 0023H) for receive data $\rightarrow$ P0 (a)

(notice: Timer 1 is used for serial communication. But it is not necessary to enable TF1)

An indefinite loop: P1 $\rightarrow$ P2 (b)

Example

```
ORG   0
LJMP MAIN
ORG   000BH                  ;ISR for Timer 0
CPL   P3.6                   ;toggle P0.1
RETI
ORG   23H
LJMP SERIAL                  ;jump to serial ISR
```

Example

```
; ------  main program, initialization ------
        ORG   30H
MAIN:MOV P1,#0FFH       ;make P1 as an input port
        MOV TMOD,#22H      ;Timer 0 &1,mode 2, auto-reload
        MOV SCON,#50H      ;8-bit,1 stop bit, REN enabled
        MOV TH1,#0FAH      ;4800 baud rate for (a)
        MOV TH0,#-92       ;5KHz square wave for (c)
        MOV IE,#10010010B  ;enable serial, timer 0 interrupt
        SETB TR1           ;start Timer 1
        SETB TR0           ;start Timer 0
;------ stay in loop indefinitely for (b) ------
            MOV   A,P1
            MOV SBUF,A
     BACK:  MOV A,P1
            MOV   P2,A
            SJMP BACK
```

Example

```
;------  Serial communication ISR ------
          ORG 100H
SERIAL:JB TI,TRANS  ;jump if TI is high
           PUSH 0E0h
          MOV A,SBUF    ;for (a)
          MOV P0,A
          CLR RI
           POP 0E0h
          RETI
TRANS:  MOV SBUF,A
          CLR TI
          RETI
          END
```

Programming External Hardware Interrupts

External Hardware Interrupts

- The 8051 has two external hardware interrupts:
  - EX0: INT0, Pin 12 (P3.2)
  - EX1: INT1, Pin 13 (P3.3)
- These two pins are also used for timers.
  - INT is also used to control the timer in the gate mode (GATE=1). (Not related to interrupts)
- They are enabled and disabled using the IE register.
  - EX0 by IE.0
  - EX1 by IE.2

External Hardware Interrupts

- Upon activation of these pins, the 8051 gets interrupted and jumps to the vector table to perform the ISR.
- There are two activation levels for the external hardware interrupts:
  - Low level triggered
  - Falling edge triggered
- This is chosen by IT0/IT1 in TCON.
  - On Reset, IT0 and IT1 are both low, making external interrupts low level-triggered.

(MSB) (LSB)

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
Timer 1		Timer0		for Interrupt			

Interrupt In-service Flags IE0 and IE1

- Fallinge-edge triggered:
  - The IE0 and IE1 of TCON register goes high whenever a falling edge is detected.
    - The IE0 and IE1 function as interrupt-in-service flags.
  - Jump to the interrupt vector table clears IE0 (IE1) flag.
  - During the time that the ISR is being executed, if another external interrupt arrives, it is latched by raising the IE0/IE1 flag, but new interrupt is not serviced until the end of current ISR.
- Low-level triggered:
  - For level triggered interrupts, it is the inverse of the signal value at the corresponding INT pin. It is 0 if INT is high, 1 if INT is low.

When are TF0, TF1, IE0, IE1 Cleared?

- 8051 clears TFx and IEx flags internally upon jumping to the interrupt vector table.
- RETI does not restore/clear the flags; it allows the microcontroller to accept interrupts with equal or lower priorities after the current one is finished.
- Without RETI, the microcontroller assumes that the ISR is being executed; hence, equal or lower priority interrupts cannot be served.
- There is an internal interrupt-in-progress flip flop per priority which should be cleared with RETI (and not with RET)

What does RETI do?

- 8051 hardware does not allow the same interrupt to be processed until its ISR is not finished with the RETI instruction.
- Therefore, new interrupt event will be detected but its ISR will NOT be called as long as ISR of another interrupt with the same or higher priority is being executed.
- The *edge-triggered* external interrupt event, if occurs, will be "latched" into IEx bit and so after RETI, the new ISR will be called. The *level-triggered* external interrupt event will NOT be "latched" and so after RETI, the new ISR will be called only if the low level still presents on input pin.

Interrupt Priority Register (Bit Addressable)

--	--	PT2	PS	PT1	PX1	PT0	PX0

--	IP.7	Reserved
--	IP.6	Reserved
PT2	IP.5	Timer 2 interrupt priority bit (8052 only).
PS	IP.4	Serial port interrupt priority bit.
PT1	IP.3	Timer 1 interrupt priority bit.
PX1	IP.2	External interrupt 1 priority bit.
PT0	IP.1	Timer 0 interrupt priority bit.
PX0	IP.0	External interrupt 0 priority bit.

User software should never write 1s to unimplemented bits, since they may be used in future products.

Interrupt Priority upon Reset

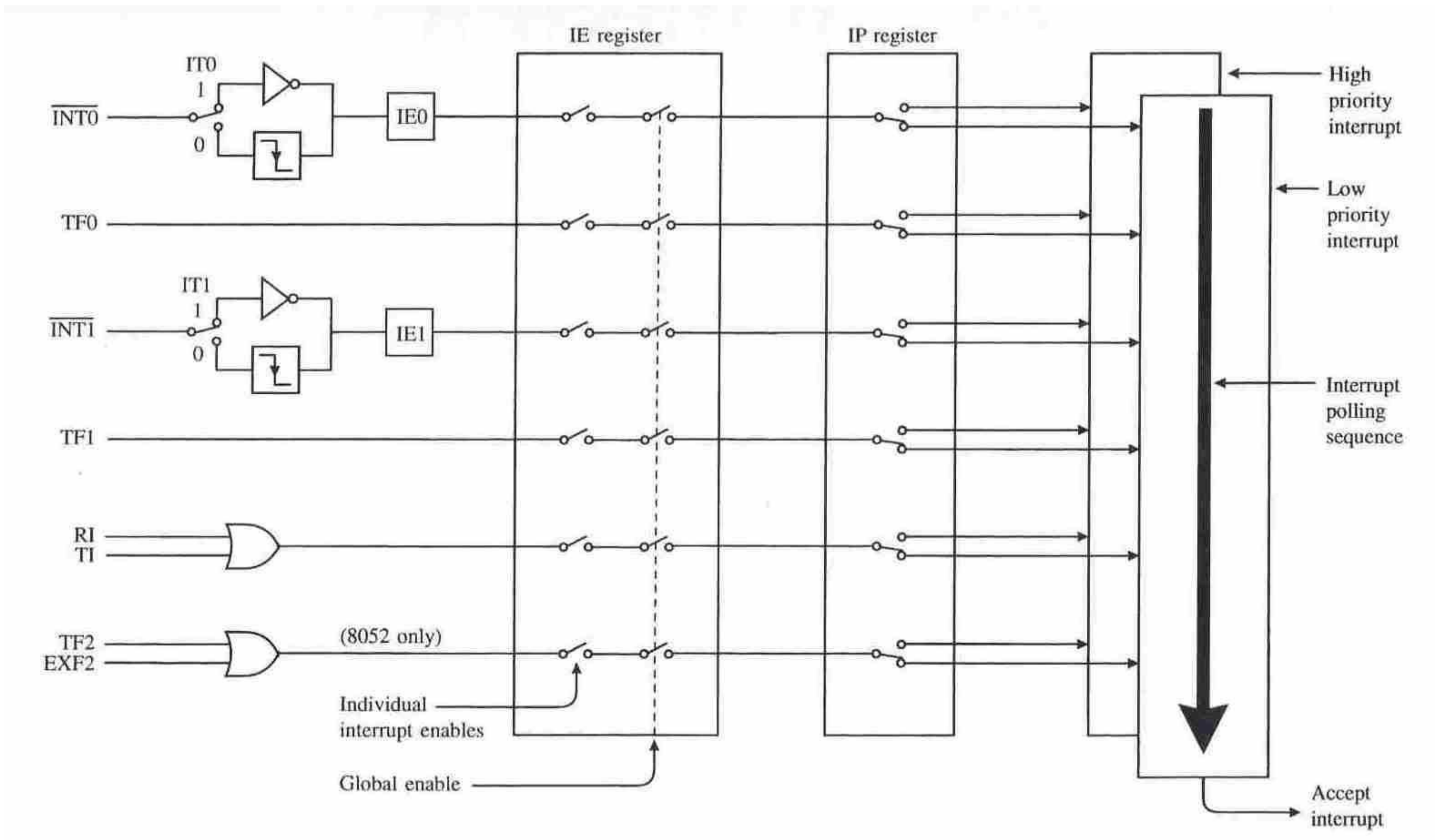
- Upon reset, all interrupts have low priority (IP bits are 0).
- The interrupt flags are polled (checked) at each MC.
- For the interrupts with same priority, the order of polling sequence is given below:
  - If multiple interrupt requests arrive simultaneously, they are served according to this order.
  - This polling sequence can be thought of another priority structure for the interrupts with same priority (The IP bits having the same value).

Polling Sequence:

External Interrupt 0	**(INT0)**	**high priority**
Timer Interrupt 0	**(TF0)**	**(In terms of polling order)**
External Interrupt 1	**(INT1)**	
Timer Interrupt 1	**(TF1)**	
Serial Communication	**(RI+TI)**	
Timer 2 (8052 only)	**(TF2)**	**low priority**

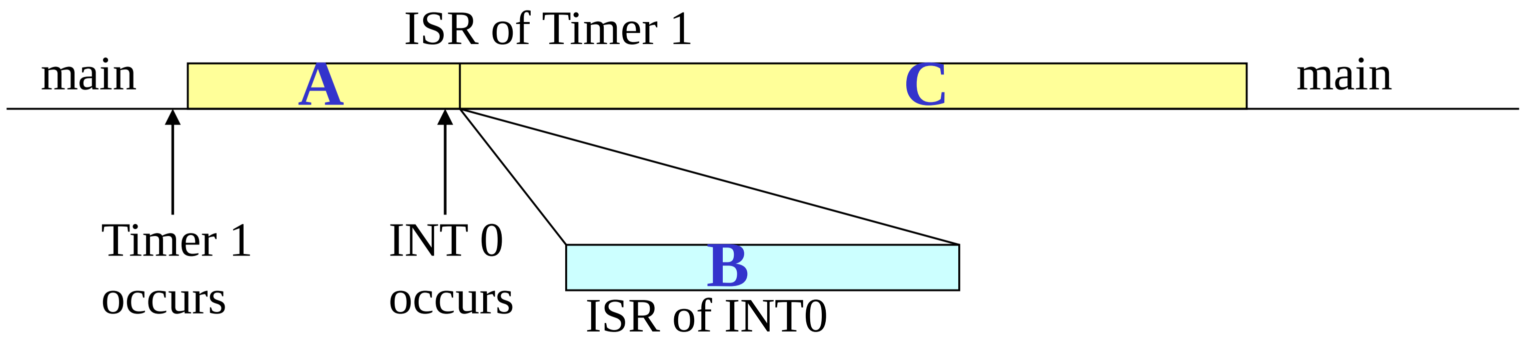
- **Remember that, independent of this sequence, an interrupt with higher priority (the IP bit set to 1) is polled before all the low priority interrupts.**

Priority Diagram



Interrupt inside an Interrupt

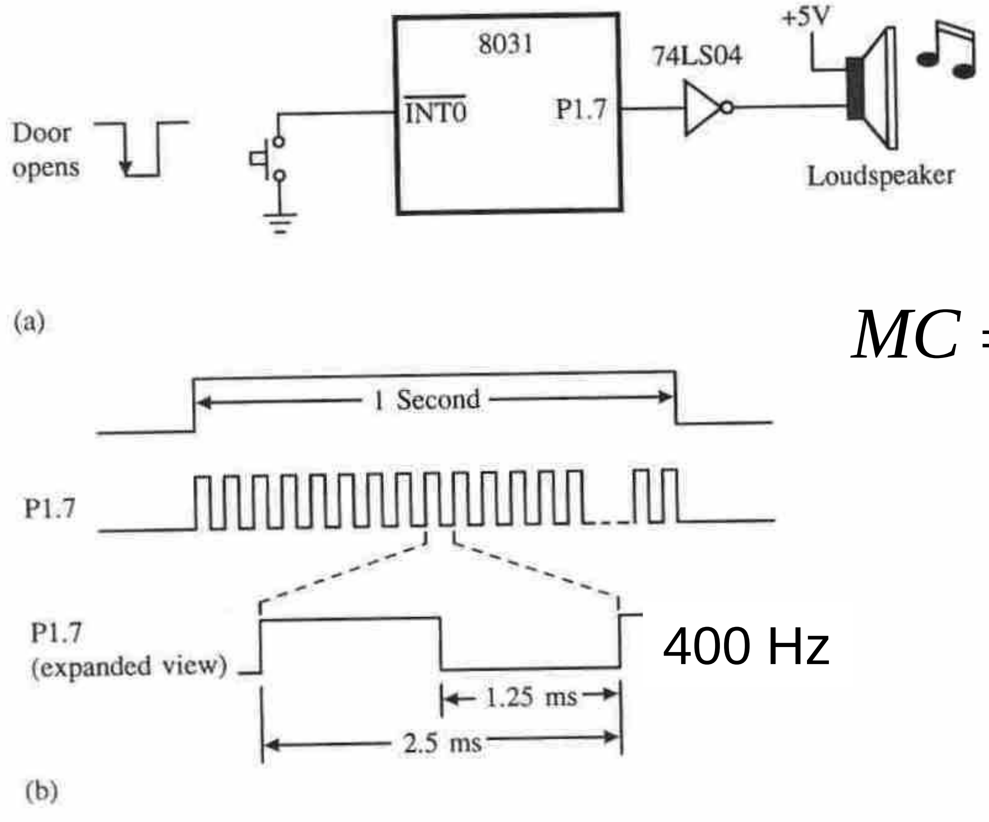
- What happens if the 8051 is executing an ISR belonging to an interrupt and another interrupt is activated?
  - A high-priority interrupt can interrupt a low-priority interrupt.
  - This is an interrupt inside an interrupt.
  - Assume PT1=0 and PX0=1 (Int 0 has high priority):



Triggering the Interrupt by Software

- Sometimes we need to test ISR.
- We can cause an interrupt with an instruction which raises the interrupt flag.
  - For example, if the IE bit for timer 1 is set, "**`SETB TF1`**" will interrupt the 8051 and force it jump to the interrupt vector table.

Example



$MC = 1\mu s$

400 Hz

- Intrusion Warning System
  - Using interrupts that sounds a 400 Hz tone for 1 second (loudspeaker connected to P1.7) whenever a door sensor connected to INT0 makes a high to low transition

Solution

```
MAIN:     SETB IT0          ; negative edge
                            ;triggered
          MOV TMOD, #11h ; 16 bit timer mode
          MOV IE,#81h       ; enable only
                            ;external int
          SJMP $

EXOISR:   MOV R7,#21;20x50000us = 1 sec.
          SETB TF0
          SETB TF1
          SETB ET0
          SETB ET1
          RETI
```

Solution - more

```
T0ISR:          CLR TR0
                DJNZ R7, SKIP
                CLR ET0
                CLR ET1
                LJMP EXIT
SKIP:           MOV TH0,#HIGH(-50000)
                MOV TL0,#LOW(-50000)
                SETB TR0
EXIT:           RETI
T1ISR:          CLR TR1
                MOV TH1,#HIGH(-1250)
                MOV TL1,#LOW(-1250)
                CPL P1.7
                SETB TR1
                RETI
                END
```