8051 TIMER & Interrupt
PROGRAMMING IN ASSEMBLY AND C
• Basic registers of the timer
  - Timer 0 and Timer 1 are 16 bits wide
  - each 16-bit timer is accessed as two separate registers of low byte and high byte.
PROGRAMMING 8051 TIMERS

• Timer 0 registers
  - low byte register is called TL0 (Timer 0 low byte) and the high byte register is referred to as TH0 (Timer 0 high byte)
  - can be accessed like any other register, such as A, B, R0, R1, R2, etc.
  - "MOV TL0, #4 FH" moves the value 4FH into TL0
  - "MOV R5, TH0" saves TH0 (high byte of Timer 0) in R5
PROGRAMMING 8051 TIMERS

Timer 0 Registers
PROGRAMMING 8051 TIMERS

• Timer 1 registers
  - also 16 bits
  - split into two bytes TL1 (Timer 1 low byte) and TH1 (Timer 1 high byte)
  - accessible in the same way as the registers of Timer 0.
SECTION 9.1: PROGRAMMING 8051 TIMERS

Timer 1 Registers
PROGRAMMING 8051 TIMERS

• **TMOD (timer mode) register**
  - Timers 0 and 1 use TMOD register to set operation modes *(only learn Mode 1 and 2)*
  - 8-bit register
  - Lower 4 bits are for Timer 0
  - Upper 4 bits are for Timer 1
  - Lower 2 bits are used to set the timer mode
    - *(only learn Mode 1 and 2)*
  - Upper 2 bits to specify the operation
    - *(only learn timer operation)*
**PROGRAMMING 8051 TIMERS**

<table>
<thead>
<tr>
<th>(MSB)</th>
<th>GATE</th>
<th>C/T</th>
<th>M1</th>
<th>M0</th>
<th>GATE</th>
<th>C/T</th>
<th>M1</th>
<th>M0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Timer 1</td>
<td></td>
<td></td>
<td></td>
<td>Timer 0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**GATE**  Gating control when set. The timer/counter is enabled only while the INTx pin is high and the TRx control pin is set. When cleared, the timer is enabled whenever the TRx control bit is set.

**C/T**  Timer or counter selected cleared for timer operation (input from internal system clock). Set for counter operation (input from Tx input pin).

**M1**  Mode bit 1

**M0**  Mode bit 0

<table>
<thead>
<tr>
<th>M1</th>
<th>M0</th>
<th>Mode</th>
<th>Operating Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>13-bit timer mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8-bit timer/counter THx with TLx as 5-bit prescaler</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>16-bit timer mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16-bit timer/counters THx and TLx are cascaded; there is no prescaler</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>8-bit auto reload</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>8-bit auto reload timer/counter; THx holds a value that is to be reloaded into TLx each time it overflows.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>Split timer mode</td>
</tr>
</tbody>
</table>

**TMOD Register**
PROGRAMMING 8051 TIMERS

• Clock source for timer
  - timer needs a clock pulse to tick
  - if C/T = 0, the crystal frequency attached to the 8051 is the source of the clock for the timer
  - frequency for the timer is always 1/12th the frequency of the crystal attached to the 8051
  - XTAL = 11.0592 MHz allows the 8051 system to communicate with the PC with no errors
  - In our case, the timer frequency is 1MHz since our crystal frequency is 12MHz
PROGRAMMING 8051 TIMERS

• Mode 1 programming
  - 16-bit timer, values of 0000 to FFFFH
  - TH and TL are loaded with a 16-bit initial value
  - timer started by "SETB TR0" for Timer 0 and "SETB TR1" for Timer 1
  - timer count ups until it reaches its limit of FFFFH
  - rolls over from FFFFH to 0000H
  - sets TF (timer flag)
  - when this timer flag is raised, can stop the timer with "CLR TR0" or "CLR TR1"
  - after the timer reaches its limit and rolls over, the registers TH and TL must be reloaded with the original value and TF must be reset to 0
Timer 1 with External Input (Mode 1)

Timer 1 external input pin 3.5

C/T = 1

TF1 goes high when FFFF \rightarrow 0

Overflow flag

Timer 1 with External Input (Mode 1)
PROGRAMMING 8051 TIMERS

• Steps to program in mode 1
  - Set timer mode 1 or 2
  - Set TL0 and TH0 (for mode 1 16 bit mode)
  - Set TH0 only (for mode 2 8 bit auto reload mode)
  - Run the timer
  - Monitor the timer flag bit
In the following program, we are creating a square wave of 50% duty cycle (with equal portions high and low) on the P1.5 bit.

Timer 0 is used to generate the time delay.

```
01 MOV TMOD,#01 ;Timer 0, mode 1(16-bit mode)
02 HERE: MOV TL0,#0F2H ;TL0 = F2H, the Low byte
03 MOV TH0,#0FFH ;TH0 = FFH, the High byte
04 CPL P1.5 ;toggle P1.5
05 ACALL DELAY
06 SJMP HERE ;load TH, TL again
07
08 DELAY: ;delay using Timer 0
09 SETB TR0 ;start Timer 0
10 AGAIN: JNB TF0,AGAIN ;monitor Timer 0 flag until ;it rolls over
11 CLR TR0 ;stop Timer 0
12 CLR TF0 ;clear Timer 0 flag
13 RET
14
15 END
```
PROGRAMMING 8051 TIMERS

• Finding values to be loaded into the timer
  - XTAL = 11.0592 MHz (12MHz)
  - divide the desired time delay by 1.085μs (1μs) to get n
  - \(65536 - n = N\)
  - convert \(N\) to hex yyxx
  - set TL = xx and TH = yy
Assuming XTAL = 11.0592 MHz, write a program to generate a square wave of 50 Hz frequency on pin P2.3.

- \( T = \frac{1}{50} \text{ Hz} = 20 \text{ ms} \)
- 1/2 of it for the high and low portions of the pulse = 10 ms
- 10 ms / 1.085 us = 9216
- 65536 - 9216 = 56320 in decimal = DC00H
- TL = 00 and TH = DCH
- The calculation for 12MHz crystal uses the same steps
Assuming \( XTAL = 11.0592 \, \text{MHz} \), write a program to generate a square wave of 50 Hz frequency on pin P2.3.

```
01 MOV TMOD,#10H ;Timer 1 mode 1 (16-bit)
02 AGAIN: MOV TL1,#00 ;TL1 = 00, Low byte
03 MOV TH1,#0DCH ;TH1 = ODCH, High byte
04 05 SETB TR1 ;start Timer 1
06 BACK: JNB TF1,BACK ;stay until timer rolls over
07 CLR TR1 ;stop Timer 1
08 CPL P2.3 ;compliment P2.3 to get hi, lo
09 CLR TF1 ;clear Timer 1 flag
10 SJMP AGAIN ;reload timer since
11 ;mode 1 is not auto reload
12 13 END
```
PROGRAMMING 8051 TIMERS

• Generating a large time delay
  - size of the time delay depends
    • crystal frequency
    • timer's 16-bit register in mode 1
  - largest time delay is achieved by making both TH and TL zero
Examine the following program and find the time delay in seconds. Exclude the time delay due to the instructions in the loop.

```
01  MOV TMOD,#10H //Timer 1, mode 1(16-bit)
02  MOV R3,#200 //counter for multiple delay
03  AGAIN: MOV TL1,#08H //TL1 = 08, Low byte
04  MOV TH1,#01H //TH1 = 01, High byte
05  SETB TR1 //start Timer 1
06  BACK: JNB TF1,BACK //stay until timer rolls over
07  CLR TR1 //stop Timer 1
08  CLR TF1 //clear Timer 1 flag
09  DJNZ R3,AGAIN //if R3 not zero then
10  END //reload timer
```
Examine the following program and find the time delay in seconds. Exclude the time delay due to the instructions in the loop.

```
01 MOV TMOD,#10H ;Timer 1, mode 1 (16-bit)
02 MOV R3,#200 ;counter for multiple delay
04 AGAIN: MOV TL1,#08H ;TL1 = 08, Low byte
05 MOV TH1,#01H ;TH1 = 01, High byte
06 SETB TR1 ;start Timer 1
07 BACK: JNB TF1,BACK ;stay until timer rolls over
08 CLR TR1 ;stop Timer 1
09 CLR TF1 ;clear Timer 1 flag
10 DJNZ R3,AGAIN ;if R3 not zero then
11 END ;reload timer
14 ;TH- TL = 0108H = 264 in decimal
15 ;65536 - 264 = 65272
16 ;65272 * 1.085us = 70.820ms
17 ;200 * 70.820ms = 14.164024s
```
• Mode 0
  - works like mode 1
  - 13-bit timer instead of 16-bit
  - 13-bit counter hold values 0000 to 1FFFH
  - when the timer reaches its maximum of 1FFFH, it rolls over to 0000, and TF is set
• Mode 2 programming
  - 8-bit timer, allows values of 00 to FFH
  - TH is loaded with the 8-bit value
  - a copy is given to TL
  - timer is started by,"SETB TR0" or "SETB TR1"
  - starts to count up by incrementing the TL register
  - counts up until it reaches its limit of FFH
  - when it rolls over from FFH to 00, it sets high TF
  - TL is reloaded automatically with the value in TH
  - To repeat, clear TF
  - mode 2 is an auto-reload mode
PROGRAMMING 8051 TIMERS

- Steps to program in mode 2
  1. load TMOD, select mode 2
  2. load the TH
  3. start timer
  4. monitor the timer flag (TF) with "JNB"
  5. get out of the loop when TF=1
  6. clear TF
  7. go back to Step 4 since mode 2 is auto-reload
Assemblers and negative values

- can let the assembler calculate the value for TH and TL which makes the job easier
- "MOV TH1, # -100", the assembler will calculate the -100 = 9CH
- "MOV TH1,#high(-10000) "
- "MOV TL1,#low(-10000) "

PROGRAMMING 8051 TIMERS
COUNTER PROGRAMMING

- **C/T bit in TMOD register**
  - C/T bit in the TMOD register decides the source of the clock for the timer
  - C/T = 0, timer gets pulses from crystal
  - C/T = 1, the timer used as counter and gets pulses from outside the 8051
  - C/T = 1, the counter counts up as pulses are fed from pins 14 and 15
  - pins are called T0 (Timer 0 input) and T1 (Timer 1 input)
  - these two pins belong to port 3
  - Timer 0, when C/T = 1, pin P3.4 provides the clock pulse and the counter counts up for each clock pulse coming from that pin
  - Timer 1, when C/T = 1 each clock pulse coming in from pin P3.5 makes the counter count up
### COUNTER PROGRAMMING

<table>
<thead>
<tr>
<th>Pin</th>
<th>Port Pin</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>P3.4</td>
<td>T0</td>
<td>Timer/Counter 0 external input</td>
</tr>
<tr>
<td>15</td>
<td>P3.5</td>
<td>T1</td>
<td>Timer/Counter 1 external input</td>
</tr>
</tbody>
</table>

Port 3 Pins Used For Timers 0 and 1
Timer 0 with External Input (Mode 1)

- **TF0** goes high when $FFFF \rightarrow 0$
- External input pin 3.4
- C/T = 1
- TR0
COUNTER PROGRAMMING

Timer 1 with External Input (Mode 2)

C/\bar{T} = 1 \quad \text{TF1 goes high when FF \rightarrow 0}

Timer 1 external input pin 3.5

overflow flag

reload

TR1

TL1

TH1

TF1
SECTION 9.2: COUNTER PROGRAMMING

Timer 1
external
input
pin 3.5

TR1

TL1

TH1

TF1

overflow flag

reload

\[ C/\overline{T} = 1 \quad \text{TF1 goes high when FF} \rightarrow 0 \]
## COUNTER PROGRAMMING

### Port 3 Pins Used For Timers 0 and 1

<table>
<thead>
<tr>
<th>Port 3 Pins</th>
<th>TR0</th>
<th>TR1</th>
<th>TF0</th>
<th>TF1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>For Timer 0</strong></td>
<td>SETB</td>
<td>SETB</td>
<td>CLR</td>
<td>CLR</td>
</tr>
<tr>
<td></td>
<td>TR0</td>
<td>TR0</td>
<td>TR0</td>
<td>TR0</td>
</tr>
<tr>
<td></td>
<td>TCON.4</td>
<td>TCON.4</td>
<td>TCON.4</td>
<td>TCON.4</td>
</tr>
<tr>
<td><strong>For Timer 1</strong></td>
<td>SETB</td>
<td>SETB</td>
<td>CLR</td>
<td>CLR</td>
</tr>
<tr>
<td></td>
<td>TR1</td>
<td>TR1</td>
<td>TR1</td>
<td>TR1</td>
</tr>
<tr>
<td></td>
<td>TCON.6</td>
<td>TCON.6</td>
<td>TCON.6</td>
<td>TCON.6</td>
</tr>
<tr>
<td><strong>TCON</strong>: Timer/Counter Control Register</td>
<td>SETB</td>
<td>SETB</td>
<td>CLR</td>
<td>CLR</td>
</tr>
<tr>
<td></td>
<td>TF1</td>
<td>TF1</td>
<td>TF1</td>
<td>TF1</td>
</tr>
</tbody>
</table>

**Notes:**
- TR0 and TR1 are used for timer control.
- TF0 and TF1 are used for flag control.
- TCON register controls the timer operation.
COUNTER PROGRAMMING

• **TCON register**
  
  - TR0 and TR1 flags turn on or off the timers
  - bits are part of a register called TCON (timer control)
  - upper four bits are used to store the TF and TR bits of both Timer 0 and Timer 1
  - lower four bits are set aside for controlling the interrupt bits
  - "SETB TR1" and "CLR TR1"
  - "SETB TCON. 6" and "CLR TCON. 6"
### COUNTER PROGRAMMING

#### Equivalent Instructions for the Timer Control Register (TCON)

<table>
<thead>
<tr>
<th>For Timer 0</th>
<th>For Timer 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>SETB TR0 = SETB TCON.4</td>
<td>SETB TR1 = SETB TCON.6</td>
</tr>
<tr>
<td>CLR TR0 = CLR TCON.4</td>
<td>CLR TR1 = CLR TCON.6</td>
</tr>
<tr>
<td>SETB TF0 = SETB TCON.5</td>
<td>SETB TF1 = SETB TCON.7</td>
</tr>
<tr>
<td>CLR TF0 = CLR TCON.5</td>
<td>CLR TF1 = CLR TCON.7</td>
</tr>
</tbody>
</table>

TCON: Timer/Counter Control Register

<table>
<thead>
<tr>
<th>TF1</th>
<th>TR1</th>
<th>TF0</th>
<th>TR0</th>
<th>IE1</th>
<th>IT1</th>
<th>IE0</th>
<th>IT0</th>
</tr>
</thead>
</table>
COUNTER PROGRAMMING

• The case of GATE = 1 in TMOD
  - GATE = 0, the timer is started with instructions "SETB TR0" and "SETB TR1"
  - GATE = 1, the start and stop of the timers are done externally through pins P3.2 and P3.3
  - allows us to start or stop the timer externally at any time via a simple switch
COUNTER PROGRAMMING

Timer/Counter 1
Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL1 count on P2. (for information only)

```
01 MOV TMOD,#01100000B ;counter 1, mode 2, C/T=1
02              ;external pulses
03 MOV TH1,#0 ;clear TH1
04 SETB P3.5 ;make T1 input
05 AGAIN: SETB TR1 ;start the counter
06 BACK: MOV A, TL1 ;get copy of count TL1
07 MOV P2,A ;display it on port 2
08 JNB TF1,BACK ;keep doing it if TF=0
09 CLR TR1 ;stop the counter 1
10 CLR TF1 ;make TF=0
11 SJMP AGAIN ;keep doing it
12
13 END
14```
Interrupts vs. Polling

• An interrupt is an external or internal event that interrupts the microcontroller
  - To inform it that a device needs its service

• A single microcontroller can serve several devices by two ways
  - Interrupts
    • 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 whatever it is doing and serves the device
Interrupts vs. Polling (cont.)

- The program which is associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler

- Polling
  - The microcontroller continuously monitors the status of a given device
    - ex. JNB TF, target
  - When the conditions met, it performs the service
  - After that, it moves on to monitor the next device until every one is serviced
    - Polling can monitor the status of several devices and serve each of them as certain conditions are met
  - The polling method is not efficient, since it wastes much of the microcontroller’s time
Interrupts vs. Polling (cont.)

• The advantage of interrupts is:
  - The microcontroller can serve many devices (not all at the same time)
  • Each device can get the attention of the microcontroller based on the assigned priority
  • For the polling method, it is not possible to assign priority since it checks all devices in a round-robin fashion
  - The microcontroller can also ignore (mask) a device request for service
    • This is not possible for the polling method
Interrupt Service Routine

• For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler
  – When an interrupt is invoked, the microcontroller runs the interrupt service routine
  – 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 interrupt vector table
Steps in Executing an Interrupt

- Upon activation of an interrupt, the microcontroller goes through:
  - It finishes the instruction it is executing and saves the address of the next instruction (PC) on the stack
  - It also saves the current status of all the registers internally (not on the stack)
  - It jumps to a fixed location in memory, called the interrupt vector table, that holds the address of the ISR.
Steps in Executing an Interrupt (cont.)

- It gets the address of the ISR from the interrupt vector table and jumps to ISR
  - It starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine which is RETI (return from interrupt)

- Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted
  - It gets the program counter (PC) address from the stack by popping the top two bytes of the stack into the PC
  - It starts to execute from that address
Six Interrupts in 8051

- Six interrupts are allocated as follows:
  - Reset – power-up reset
  - Two interrupts are set aside for the timers:
    - One for timer 0 and one for timer 1
  - Two interrupts are set aside for hardware external interrupts
    - P3.2 and P3.3 are for the external hardware interrupts INT0 (or EX1), and INT1 (or EX2)
  - Serial communication has a single interrupt that belongs to both receive and transfer
Enabling and Disabling an Interrupt

• Upon reset, all interrupts are disabled (masked)
  – None will be responded to by the microcontroller if they are activated
    • The interrupts must be enabled by software in order for the microcontroller to respond to them
  – There is a register called IE (interrupt enable) that is responsible for enabling (unmasking) and disabling (masking) the interrupts
<table>
<thead>
<tr>
<th>Field</th>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA</td>
<td>IE.7</td>
<td>Disables all interrupts. If EA = 0, no interrupt is acknowledged. If EA = 1, each interrupt source is individually enabled or disabled by setting or clearing its enable bit.</td>
</tr>
<tr>
<td>--</td>
<td>IE.6</td>
<td>Not implemented, reserved for future use.*</td>
</tr>
<tr>
<td>ET2</td>
<td>IE.5</td>
<td>Enables or disables Timer 2 overflow or capture interrupt (8052 only).</td>
</tr>
<tr>
<td>ES</td>
<td>IE.4</td>
<td>Enables or disables the serial port interrupt.</td>
</tr>
<tr>
<td>ET1</td>
<td>IE.3</td>
<td>Enables or disables Timer 1 overflow interrupt.</td>
</tr>
<tr>
<td>EX1</td>
<td>IE.2</td>
<td>Enables or disables external interrupt 1.</td>
</tr>
<tr>
<td>ET0</td>
<td>IE.1</td>
<td>Enables or disables Timer 0 overflow interrupt.</td>
</tr>
<tr>
<td>EX0</td>
<td>IE.0</td>
<td>Enables or disables external interrupt 0.</td>
</tr>
</tbody>
</table>

*User software should not write 1s to reserved bits. These bits may be used in future flash microcontrollers to invoke new features.

Figure 11-2. IE (Interrupt Enable) Register
To enable an interrupt, we take the following steps:

- Bit D7 of the IE register (EA) must be set to high to allow the rest of the register to take effect.
- The value of EA
  - If EA = 1, interrupts are enabled and will be responded to if their corresponding bits in IE are high.
  - If EA = 0, no interrupt will be responded to, even if the associated bit in the IE register is high.
Timer Interrupts

• The timer flag (TF) is raised when the timer rolls over
  - In polling TF, we have to wait until the TF is raised
    • The microcontroller is tied down while waiting for TF to be raised, and can not do anything else
  - Using interrupts to avoid tying down the controller
    • If the timer interrupt in the IE register is enabled, whenever the timer rolls over, TF is raised
The microcontroller is interrupted in whatever it is doing, and jumps to the interrupt vector table to service the ISR. In this way, the microcontroller can do other until it is notified that the timer has rolled over.
External Hardware Interrupts

• The 8051 has two external hardware interrupts
  – Pin 12 (P3.2) and pin 13 (P3.3) of the 8051
    • Designated as INT0 and INT1
    • Used as external hardware interrupts
  – The interrupt vector table locations 0003H and 0013H are set aside for INT0 and INT1
  – There are two activation levels for the external hardware interrupts
    • Level trigged
Level-Triggered Interrupt

- INT0 and INT1 pins are normally high
  - If a low-level signal is applied to them, it triggers the interrupt
- The microcontroller stops whatever it is doing and jumps to the interrupt vector table to service that interrupt
- The low-level signal at the INT pin must be removed before the execution of the last instruction of the ISR, RETI
  - Otherwise, another interrupt will be generated
- This is called a level-triggered or level-activated interrupt and is the default mode upon reset
Sampling Low Level-Triggered Interrupt

- P3.2 and P3.3 are used for normal I/O
  - Unless the INT0 and INT1 bits in the IE register are enabled
    - After the hardware interrupts are enabled, the controller keeps sampling the INTn pin for a low-level signal once each machine cycle
    - The pin must be held in a low state until the start of the execution of ISR
      - If the INTn pin is brought back to a logic high before the start of the execution of ISR, there will be no interrupt
    - If INTn pin is left at a logic low after the RETI instruction of the ISR, another interrupt will be activated after one instruction is executed
Sampling Low Level-Triggered Interrupt (cont.)

• To ensure the activation of the hardware interrupt at the INTn pin,
  - The duration of the low-level signal is around 4 machine cycles, but no more
    • This is due to the fact that the level-triggered interrupt is not latched
    • Thus the pin must be held in a low state until the start of the ISR execution

![Diagram showing 1 MC (1.085us) repeated 4 times to equal 4 machine cycles (4.34us)]

note: On reset, IT0 (TCON.0) and IT1 (TCON.2) are both low, making external interrupt level-triggered
Edge-Triggered Interrupt

- To make INT0 and INT1 edge-triggered interrupts, we must program the bits of the TCON register
  - The TCON register holds the IT0 and IT1 flag bits that determine level- or edge-triggered mode of the hardware interrupt
    - IT0 and IT1 are bits D0 and D2 of TCON
      - They are also referred to as TCON.0 and TCON.2 since the TCON register is bit-addressable
<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>D7</td>
<td>TF1</td>
<td>TR1</td>
<td>TF0</td>
<td>TR0</td>
<td>IE1</td>
<td>IT1</td>
<td>IE0</td>
<td>IT0</td>
</tr>
<tr>
<td>TF1</td>
<td>TCON.7</td>
<td>Timer 1 overflow flag. Set by hardware when timer/counter 1 overflows. Cleared by hardware as the processor vectors to the interrupt service routine.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TR1</td>
<td>TCON.6</td>
<td>Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TF0</td>
<td>TCON.5</td>
<td>Timer 0 overflow flag. Set by hardware when timer/counter 0 overflows. Cleared by hardware as the processor vectors to the service routine.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TR0</td>
<td>TCON.4</td>
<td>Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 on/off.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bit</td>
<td>Register</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>----------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE1</td>
<td>TCON.3</td>
<td>External interrupt 1 edge flag. Set by CPU when the external interrupt edge (H-to-L transition) is detected. Cleared by CPU when the interrupt is processed. <em>Note:</em> This flag does not latch low-level triggered interrupts.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IT1</td>
<td>TCON.2</td>
<td>Interrupt 1 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IE0</td>
<td>TCON.1</td>
<td>External interrupt 0 edge flag. Set by CPU when external interrupt (H-to-L transition) edge is detected. Cleared by CPU when interrupt is processed. <em>Note:</em> This flag does not latch low-level triggered interrupts.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IT0</td>
<td>TCON.0</td>
<td>Interrupt 0 type control bit. Set/cleared by software to specify falling edge/low-level triggered external interrupt.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Figure 11-6. TCON (Timer/Counter) Register (Bit-addressable)*
Sampling Edge-Triggered Interrupt

• The external source must be held high for at least one machine cycle, and then held low for at least one machine cycle
  - The falling edge of pins INT0 and INT1 are latched by the 8051 and are held by the TCON.1 and TCON.3 bits of TCON register
    • Function as interrupt-in-service flags
    • It indicates that the interrupt is being responded to until this service is finished
Sampling Edge-Triggered Interrupt (cont.)

- When the ISRs are finished, TCON.1 and TCON.3 are cleared
  - The interrupt is finished and the 8051 is ready to respond to another interrupt on that pin
    - During the time that the interrupt service routine is being executed, the INTn pin is ignored, no matter how many times it makes a high-to-low transition
    - RETI clears the corresponding bit in TCON register (TCON.1 or TCON.3)
  - There is no need for instruction CLR TCON.1 before RETI in the ISR associated with INT0
Serial Communication Interrupt

• TI (transfer interrupt) is raised when the stop bit is transferred
  - Indicating that the SBUF register is ready to transfer the next byte

• RI (received interrupt) is raised when the stop bit is received
  - Indicating that the received byte needs to be picked up before it is lost (overrun) by new incoming serial data
RI and TI Flags and Interrupts

- In the 8051 there is only one interrupt set aside for serial communication
  - Used to both send and receive data
  - If the interrupt bit in the IE register (IE.4) is enabled, when RI or TI is raised the 8051 gets interrupted and jumps to memory location 0023H to execute the ISR.
  - In that ISR we must examine the TI and RI flags.

Serial interrupt is invoked by TI or RI flags.
Use of Serial COM in 8051

• The serial interrupt is used mainly for receiving data and is never used for sending data serially
  - This is like getting a telephone call in which we need a ring to be notified
  - If we need to make a phone call there are other ways to remind ourselves and there is no need for ringing
  - However in receiving the phone call, we must respond immediately no matter what we are doing in order to miss the call
# Interrupt Flag Bits

- The TCON register holds four of the interrupt flags in the 8051.
- The SCON register has the RI and TI flags.

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Flag</th>
<th>SFR Register Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>External 0</td>
<td>IE0</td>
<td>TCON.1</td>
</tr>
<tr>
<td>External 1</td>
<td>IE1</td>
<td>TCON.3</td>
</tr>
<tr>
<td>Timer 0</td>
<td>TF0</td>
<td>TCON.5</td>
</tr>
<tr>
<td>Timer 1</td>
<td>TF1</td>
<td>TCON.7</td>
</tr>
<tr>
<td>Serial port</td>
<td>T1</td>
<td>SCON.1</td>
</tr>
<tr>
<td>Timer 2</td>
<td>TF2</td>
<td>T2CON.7 (AT89C52)</td>
</tr>
<tr>
<td>Timer 2</td>
<td>EXF2</td>
<td>T2CON.6 (AT89C52)</td>
</tr>
</tbody>
</table>
Interrupt Priority

• When the 8051 is powered up, the priorities are assigned
  – In reality, the priority scheme is nothing but an internal polling sequence in which the 8051 polls the interrupts in the sequence listed

<table>
<thead>
<tr>
<th>Highest to Lowest Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Interrupt 0</td>
</tr>
<tr>
<td>(INT0)</td>
</tr>
<tr>
<td>Timer Interrupt 0</td>
</tr>
<tr>
<td>(TF0)</td>
</tr>
<tr>
<td>External Interrupt 1</td>
</tr>
<tr>
<td>(INT1)</td>
</tr>
<tr>
<td>Timer Interrupt 1</td>
</tr>
<tr>
<td>(TF1)</td>
</tr>
<tr>
<td>Serial Communication</td>
</tr>
<tr>
<td>(RI + TI)</td>
</tr>
<tr>
<td>Timer 2 (8052 only)</td>
</tr>
<tr>
<td>TF2</td>
</tr>
</tbody>
</table>
## Interrupt Priority Register (Bit-addressable)

<table>
<thead>
<tr>
<th>D7</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>PT2</td>
<td>PS</td>
</tr>
<tr>
<td>PT1</td>
<td>PX1</td>
</tr>
<tr>
<td>PT0</td>
<td>PX0</td>
</tr>
</tbody>
</table>

| -- | 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 |

Priority bit = 1 assigns high priority
Priority bit = 0 assigns low priority
To test an ISR by way of simulation can be done with simple instructions to set the interrupts high

- Thereby cause the 8051 to jump to the interrupt vector table
- ex. If the IE bit for timer 1 is set, an instruction such as SETB TF1 will interrupt the 8051 in whatever it is doing and will force it to jump to the interrupt vector table

- We do not need to wait for timer 1 to roll over to have an interrupt