CHAPTER 11

INTERRUPTS

PROGRAMMING
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 by polling devices that do not need service
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
Table 11-1: Interrupt Vector Table for the 8051

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>ROM Location (Hex)</th>
<th>Pin</th>
<th>Flag Clearing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>0000</td>
<td>9</td>
<td>Auto</td>
</tr>
<tr>
<td>External hardware interrupt 0 (INT0)</td>
<td>0003</td>
<td>P3.2 (12)</td>
<td>Auto</td>
</tr>
<tr>
<td>Timer 0 interrupt (TF0)</td>
<td>000B</td>
<td></td>
<td>Auto</td>
</tr>
<tr>
<td>External hardware interrupt 1 (INT1)</td>
<td>0013</td>
<td>P3.3 (13)</td>
<td>Auto</td>
</tr>
<tr>
<td>Timer 1 interrupt (TF1)</td>
<td>001B</td>
<td></td>
<td>Auto</td>
</tr>
<tr>
<td>Serial COM interrupt (RI and TI)</td>
<td>0023</td>
<td></td>
<td>Programmer clears it.</td>
</tr>
</tbody>
</table>

```
ORG 0       ; wake-up ROM reset location
LJMP MAIN   ; by-pass int. vector table

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

Only three bytes of ROM space assigned to the reset pin. We put the LJMP as the first instruction and redirect the processor away from the interrupt vector table.
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>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>EA</td>
<td>--</td>
<td>ET2</td>
<td>ES</td>
<td>ET1</td>
<td>EX1</td>
<td>ET0</td>
<td>EX0</td>
</tr>
</tbody>
</table>

**EA**  
IE.7  
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.

**--**  
IE.6  
Not implemented, reserved for future use.*

**ET2**  
IE.5  
Enables or disables Timer 2 overflow or capture interrupt (8052 only).

**ES**  
IE.4  
Enables or disables the serial port interrupt.

**ET1**  
IE.3  
Enables or disables Timer 1 overflow interrupt.

**EX1**  
IE.2  
Enables or disables external interrupt 1.

**ET0**  
IE.1  
Enables or disables Timer 0 overflow interrupt.

**EX0**  
IE.0  
Enables or disables external interrupt 0.

*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**
Enabling and Disabling an Interrupt (cont.)

- 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 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
Example 11-1

Show the instructions to (a) enable the serial interrupt, Timer 0 interrupt, and external hardware interrupt 1 (EX1), and (b) disable (mask) the Timer 0 interrupt, then (c) show how to disable all the interrupts with a single instruction.

Solution:

(a) MOV IE,#10010110B ;enable serial, Timer 0, EX1
Since IE is a bit-addressable register, we can use the following instructions to access individual bits of the register.

(b) CLR IE.1 ;mask(disable) Timer 0 interrupt only
(c) CLR IE.7 ;disable all interrupts

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 Timer 0 interrupt
SETB IE.2 ;enable EX1
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
Timer Interrupts (cont.)

- 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.

![Diagram showing timer interrupts]

TF0: Timer 0 Interrupt Vector
- Jump to 000BH

TF1: Timer 1 Interrupt Vector
- Jump to 001BH
Example 11-2

Write a program that continuously gets 8-bit data from P0 and sends it to P1 while simultaneously creating a square wave of 200 $\mu$s period on pin P2.1. Use Timer 0 to create the square wave. Assume that XTAL = 11.0592 MHz.

Solution:

We will use Timer 0 in mode 2 (auto-reload). $TH0 = 100/1.085 \, \mu s = 92$.

;--Upon wake-up go to main, avoid using memory space ;allocated to Interrupt Vector Table
ORG 0000H
LJMP MAIN ;bypass interrupt vector table

;--ISR for Timer 0 to generate square wave
ORG 000BH ;Timer 0 interrupt vector table
CPL P2.1 ;toggle P2.1 pin
RETI ;return from ISR

;--The main program for initialization
ORG 0030H ;after vector table space
MAIN:
MOV TMOD,#02H ;Timer 0, mode 2 (auto-reload)
MOV P0,#0FFH ;make P0 an input port
MOV TH0,#-92 ;TH0=A4H for -92
MOV IE,#82H ;IE=10000010 (bin) enable Timer 0
SETB TR0 ;Start Timer 0

BACK:
MOV A,P0 ;get data from P0
MOV P1,A ;issue it to P1
SJMP BACK ;keep doing it
;loop unless interrupted by TF0
END
Example 11-3
Rewrite Example 11-2 to create a square wave that has a high portion of 1085 us and a low portion of 15 us. Assume XTAL=11.0592MHz. Use timer 1.

Solution:
Since 1085 us is $1000 \times 1.085$ we need to use mode 1 of timer 1.

;--upon wake-up go to main, avoid using
;memory allocated to Interrupt Vector Table
ORG 0000H
LJMP MAIN ;by-pass int. vector table
;--ISR for timer 1 to generate square wave
ORG 001BH ;Timer 1 int. vector table
LJMP ISR_T1 ;jump to ISR
...
;--The main program for initialization
ORG 0030H ;after vector table space
MAIN: MOV TMOD,#10H ;Timer 1, mode 1
       MOV P0,#OFFH ;make P0 an input port
       MOV TL1,#018H ;TL1=18 low byte of -1000
       MOV TH1,#0FCH ;TH1=FC high byte of -1000
       MOV IE,#88H ;10001000 enable Timer 1 int
       SETB TR1 ;Start Timer 1
BACK: MOV A,P0 ;get data
       MOV P1,A ;issue
       SJMP BACK ;keep doing

;Timer 1 ISR. Must be reloaded, not auto-reload
ISR_T1: CLR TR1 ;stop Timer 1
       MOV R2,#4 ; 2MC
       CLR P2.1 ;P2.1=0, start of low portion
HERE: DJNZ R2,HERE ;4x2 machine cycle 8MC
       MOV TL1,#18H ;load T1 low byte value 2MC
       MOV TH1,#0FCH ;load T1 high byte value 2MC
       SETB TR1 ;starts timer1 1MC
       SETB P2.1 ;P2.1=1, back to high 1MC
RETI ;return to main
END

Low portion of the pulse is created by 14 MC
14 x 1.085 us = 15.19 us
Example 11-4

Write a program to generate a square wave of 50 Hz frequency on pin P1.2. This is similar to Example 9-12 except that it uses an interrupt for Timer 0. Assume that XTAL = 11.0592 MHz.

Solution:

```
ORG 0
LJMP MAIN
ORG 000BH ;ISR for Timer 0
CPL P1.2 ;complement P1.2
MOV TL0,#00 ;reload timer values
MOV TH0,#0DCH
RETI ;return from interrupt
ORG 30H ;starting location for prog.

;-----main program for initialization
MAIN: MOV TMOD,#00000001B ;Timer 0, Mode 1
       MOV TL0,#00
       MOV TH0,#0DCH
       MOV IE,#82H ;enable Timer 0 interrupt
       SETB TR0 ;start timer
HERE: SJMP HERE ;stay here until interrupted
END
```

8051

![Diagram showing a 50 Hz square wave on P1.2]
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 triggered
    - Edge triggered
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
Example 11-5

Assume that the INT1 pin is connected to a switch that is normally high. Whenever it goes low, it should turn on an LED. The LED is connected to P1.3 and is normally off. When it is turned on it should stay on for a fraction of a second. As long as the switch is pressed low, the LED should stay on.

Solution:

```assembly
ORG  0000H
LJMP MAIN ;bypass interrupt vector table

;--ISR for hardware interrupt INT1 to turn on the LED
ORG  0013H ;INT1 ISR
SETB P1.3 ;turn on LED
MOV  R3,#255 ;load counter
BACK: DJNZ R3,BACK ;keep LED on for a while
CLR  P1.3 ;turn off the LED
RETI ;return from ISR

;--MAIN program for initialization
ORG  30H
MAIN: MOV  IE,#10000100B ;enable external INT1
HERE: SJMP HERE ;stay here until interrupted
END
```
Pressing the switch will turn the LED on. If it is kept activated, the LED stays on.
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

\[
\begin{align*}
1 \text{ MC} & \quad 4 \text{ machine cycles} & \quad 4 \times 1.085\text{us} \\
1.085\text{us} & \quad \text{To INT0 or INT1 pins} \\
\end{align*}
\]

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.
<table>
<thead>
<tr>
<th>D7</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF1</td>
<td>TR1</td>
</tr>
</tbody>
</table>

**TF1** TCON.7  
Timer 1 overflow flag. Set by hardware when timer/counter 1 overflows. Cleared by hardware as the processor vectors to the interrupt service routine.

**TR1** TCON.6  
Timer 1 run control bit. Set/cleared by software to turn timer/counter 1 on/off.

**TF0** TCON.5  
Timer 0 overflow flag. Set by hardware when timer/counter 0 overflows. Cleared by hardware as the processor vectors to the service routine.

**TR0** TCON.4  
Timer 0 run control bit. Set/cleared by software to turn timer/counter 0 on/off.
<table>
<thead>
<tr>
<th></th>
<th>TCON</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
</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>
</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>
</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>
</tr>
</tbody>
</table>

Figure 11-6. TCON (Timer/Counter) Register (Bit-addressable)
Example 11-6

Assuming that pin 3.3 (INT1) is connected to a pulse generator, write a program in which the falling edge of the pulse will send a high to P1.3, which is connected to an LED (or buzzer). In other words, the LED is turned on and off at the same rate as the pulses are applied to the INT1 pin. This is an edge-triggered version of Example 11-5.

Solution:

```assembly
ORG 0000H
LJMP MAIN

;--ISR for hardware interrupt INT1 to turn on the LED
ORG 0013H ;INT1 ISR
SETB P1.3 ; turn on the LED
MOV R3,#255
BACK: DJNZ R3,BACK ; keep the LED on for a while
CLR P1.3 ; turn off the LED
RETI ; return from ISR

;--MAIN program for initialization
ORG 30H
MAIN: SETB TCON.2 ; make INT1 edge-trigger interrupt
MOV IE,#10000100B ; enable External INT1
HERE: SJMP HERE ; stay here until interrupted
END
```

When the falling edge of the signal is applied to pin INT1, the LED will be turned on momentarily.

The on-state duration depends on the time delay inside the ISR for INT1.
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 serviced now
    - On this INTn pin, and no new interrupt will be responded to until this service is finished

Minimum pulse duration to detect edge-triggered interrupts XTAL=11.0592MHz

\[
\text{1 MC} \quad \text{1085us} \quad \text{1 MC} \quad \text{1085us}
\]
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
Figure 11-4. Activation of INTO and INT1
Example 11-7
What is the difference between the RET and RETI instructions? Explain why we cannot use RET instead of RETI as the last instruction of an ISR.

Solution:
Both perform the same actions of popping off the top two bytes of the stack into the program counter, and marking the 8051 return to where it left off.

However, RETI also performs an additional task of clearing the interrupt-in-service flag, indicating that the servicing of the interrupt is over and the 8051 now can accept a new interrupt on that pin. If you use RET instead of RETI as the last instruction of the interrupt service routine, you simply block any new interrupt on that pin after the first interrupt, since the pin status would indicate that the interrupt is still being serviced. In the cases of TF0, TF1, TCON.1, and TCON.3, they are cleared due to the execution of RETI.
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 to see which one caused the interrupt and respond accordingly

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 or we will miss the call
Example 11-8

Write a program in which the 8051 reads data from P1 and writes it to P2 continuously while giving a copy of it to the serial COM port to be transferred serially. Assume that XTAL = 11.0592 MHz. Set the baud rate at 9600.

Solution:

ORG 0
LJMP MAIN
ORG 23H
LJMP SERIAL ;jump to serial interrupt ISR
ORG 30H
MAIN:
MOV P1,#0FFH ;make P1 an input port
MOV TMOD,#20H ;timer 1, mode 2 (auto-reload)
MOV TH1,#0FDH ;9600 baud rate
MOV SCON,#50H ;8-bit, 1 stop, REN enabled
MOV IE,#10010000B ;enable serial interrupt
SETB TR1 ;start timer 1
BACK:
MOV A,P1 ;read data from port 1
MOV SBUF,A ;give a copy to SBUF
MOV P2.A ;send it to P2
HERE:
JNB TI, HERE
CLR TI
SJMP BACK ;stay in loop indefinitely
In the above program notice the role of TI and RI. The moment a byte is written into SBUF it is framed and transferred serially. As a result, when the last bit (stop bit) is transferred the TI is raised, which causes the serial interrupt to be invoked since the corresponding bit in the IE register is high. In the serial ISR, we check for both TI and RI since both could have invoked the interrupt. In other words, there is only one interrupt for both transmit and receive.
Example 11-9

Write a program in which the 8051 gets data from P1 and sends it to P2 continuously while incoming data from the serial port is sent to P0. Assume that XTAL = 11.0592 MHz. Set the baud rate at 9600.

Solution:

```
ORG  0
LJMP MAIN
ORG  23H
LJMP SERIAL ;jump to serial ISR
ORG  30H
MAIN:  MOV  P1,#0FFH ;make P1 an input port
MOV   TMOD,#20H ;timer 1, mode 2 (auto-reload)
MOV   TH1,#0FDH ;9600 baud rate
MOV   SCON,#50H ;8-bit, 1 stop, REN enabled
MOV   IE,#1001000B ;enable serial interrupt
SETE  TR1 ;start Timer 1
BACK:  MOV   A,P1 ;read data from port 1
MOV   P2,A ;send it to P2
SJMP BACK ;stay in loop indefinitely

;-------------------------------------SERIAL PORT ISR
ORG  100H
SERIAL: JB   TI,TRANS ;jump if TI is high
MOV   A,SBUF ;otherwise due to receive
MOV   P0,A ;send incoming data to P0
CLRR  RI ;clear RI since CPU doesn’t
TRANS: RETI ;return from ISR
          ;clear TI since CPU doesn’t
          ;return from ISR
END
```
Example 11-10

Write a program using interrupts to do the following:
(a) Receive data serially and send it to P0,
(b) Have port P1 read and transmitted serially, and a copy given to P2,
(c) Make Timer 0 generate a square wave of 5 kHz frequency on P0.1.
Assume that XTAL = 11.0592 MHz. Set the baud rate at 4800.

Solution:

```
ORG 0
LJMP MAIN
ORG 000BH ;ISR for Timer 0
CPL P0.1 ;toggle P0.1
RETI ;return from ISR
ORG 23H
LJMP SERIAL ;jump to serial int. ISR
ORG 30H
MAIN:
MOV P1,#0FFH ;make P1 an input port
MOV TMOD,#22H ;timer 0&1,mode 2, auto-reload
MOV TH1,#0F6H ;4800 baud rate
MOV SCON,#50H ;8-bit, 1 stop, REN enabled
MOV TH0,#-92 ;for 5 KHz wave
MOV IE,#10010010B ;enable serial, timer 0 int.
SETB TR1 ;start timer 1
SETB TR0 ;start timer 0
```
BACK:    MOV   A,P1 ;read data from port 1
         MOV   SBUF,A ;give a copy to SBUF
         MOV   P2,A ;write it to P2
HERE:    JNB   TI, HERE ;stay in loop indefinitely
         CLR   TI
         SJMP  BACK

;---------------------------------- SERIAL PORT ISR

ORG    100H

SERIAL: JB    TI,TRANS ;jump if TI is high
         MOV   A,SBUF ;otherwise due to received
         MOV   P0,A ;send serial data to P0
         CLR   RI ;clear RI since CPU does not
TRANS:  RETI ;return from ISR
         CLR    TI ;clear TI since CPU does not
         END ;return from ISR
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 and responds accordingly

<table>
<thead>
<tr>
<th>Highest to Lowest Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Interrupt 0 (INT0)</td>
</tr>
<tr>
<td>Timer Interrupt 0 (TF0)</td>
</tr>
<tr>
<td>External Interrupt 1 (INT1)</td>
</tr>
<tr>
<td>Timer Interrupt 1 (TF1)</td>
</tr>
<tr>
<td>Serial Communication (RI + TI)</td>
</tr>
<tr>
<td>Timer 2 (8052 only) (TF2)</td>
</tr>
</tbody>
</table>
**Example 11-11**
Discuss what happens if interrupts INT0, TF0, and INT1 are activated at the same time. Assume priority levels were set by the power-up reset and the external hardware interrupts are edge-triggered.

**Solution:**
If these three interrupts are activated at the same time, they are latched and kept internally. Then the 8051 checks all five interrupts according to the sequence listed in Table 11-3. If any is activated, it services it in sequence. Therefore, when the above three interrupts are activated, IE0 (external interrupt 0) is serviced first, then timer 0 (TF0), and finally IE1 (external interrupt 1).
Altering Interrupt Priority

- We can alter the sequence of interrupt priority by programming a register called IP (interrupt priority)
  - To give a higher priority to any of the interrupts, we make the corresponding bit in the IP register high
  - When two or more interrupt bits in the IP register are set to high
    - While these interrupts have a higher priority than others, they are serviced according to the sequence of Table 11-13
## Interrupt Priority Register (Bit-addressable)

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

- **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
Example 11-12
(a) Program the IP register to assign the highest priority to INT1 (external interrupt 1), then
(b) discuss what happens if INT0, INT1, and TF0 are activated at the same time. Assume the interrupts are both edge-triggered.

Solution:
(a) MOV IP, #00000100B ; IP.2 = 1 assign INT1 higher priority. The instruction SETB IP.2 also will do the same thing as the above line since IP is bit-addressable.
(b) The instruction in Step (a) assigned a higher priority to INT1 than the others; therefore, when INT0, INT1, and TF0 interrupts are activated at the same time, the 8051 services INT1 first, then it services INT0, then TF0. This is due to the fact that INT1 has a higher priority than the other two because of the instruction in Step (a). The instruction in Step (a) makes both the INT0 and TF0 bits in the IP register 0. As a result, the sequence in Table 11-3 is followed which gives a higher priority to INT0 over TF0.
Example 11-13
Assume that after reset, the interrupt priority is set the instruction
MOV IP, #00001100B. Discuss the sequence in which the
interrupts are serviced.

Solution:
The instruction “MOV IP #00001100B” (B is for binary) and timer 1
(TF1) to a higher priority level compared with the reset of the
interrupts. However, since they are polled according to Table,
they will have the following priority.

<table>
<thead>
<tr>
<th>Highest Priority</th>
<th>External Interrupt 1 (INT1)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Timer Interrupt 1 (TF1)</td>
</tr>
<tr>
<td></td>
<td>External Interrupt 0 (INT0)</td>
</tr>
<tr>
<td></td>
<td>Timer Interrupt 0 (TF0)</td>
</tr>
<tr>
<td>Lowest Priority</td>
<td>Serial Communication (RI+TI)</td>
</tr>
</tbody>
</table>
Interrupt inside an Interrupt

- In the 8051 a low-priority interrupt can be interrupted by a higher-priority interrupt but not by another low priority interrupt
  - Although all the interrupts are latched and kept internally, no low-priority interrupt can get the immediate attention of the CPU until the 8051 has finished servicing the high-priority interrupts
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 go roll over to have an interrupt