Chapter 6: Interrupts
Interrupts

• An interrupt is the occurrence of an event that causes a temporary suspension of a program while the condition is serviced by another program.
  – Allows a system to respond asynchronously to an event and deal with the event while another program is executing.

• An interrupt driven system gives the illusion of doing many things simultaneously.
  – Of course, the CPU cannot execute more than one instruction at a time.
    • It can temporarily suspend execution of one program, execute another, then return to the first program.
  – In a way, interrupts are like subroutines. Except that one does not know when the interrupt code will be executed.
Interrupts

• Examples we covered in previous chapters did not use interrupts but made use of “wait loops” to test timer overflow flag or serial port transmit and receive flags.
• In this approach the CPU’s time is wasted waiting for flags to be set.
• We can use interrupts to avoid using wait loops.

• Interrupts
  – If and when a device is ready and needs attention, it informs the CPU
  – CPU drops whatever it was doing and serves the device and then returns back to its original task
  – CPU is always “free”, when not serving any interrupts
• Example: Write a program using Timer 0 to create a 10 kHz square wave on P1.0
  
  MOV TMOD,#02H
  MOV TH0,#-50
  SETB TR0

  LOOP: JNB TF0, LOOP
  CLR TF0
  CPL P1.0
  SJMP LOOP

• This program creates a square wave on P1.0 with a high-time of 50 us and low time of 50 us.
Interrupt Service Routines

- CPUs have fixed number of interrupts
  - Every interrupt has to be associated with a piece of code called “Interrupt Service Routine”, or ISR.
  - If interrupt-x is received by CPU, the ISR-x is executed
- ISRs are basically “subroutines”, but they end with the RETI, instruction instead of RET
- CPU architecture defines a specific “code address” for each ISR, which is stored in the “Interrupt vector Table (IVT)”
- When an interrupt occurs, the CPU fetches its ISR code address from the IVT and executes it.
## Interrupt vector table

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Vector address</th>
</tr>
</thead>
<tbody>
<tr>
<td>System reset</td>
<td>0000H</td>
</tr>
<tr>
<td>External 0</td>
<td>0003H</td>
</tr>
<tr>
<td>Timer 0</td>
<td>000BH</td>
</tr>
<tr>
<td>External 1</td>
<td>0013H</td>
</tr>
<tr>
<td>Timer 1</td>
<td>001BH</td>
</tr>
<tr>
<td>Serial port</td>
<td>0023H</td>
</tr>
</tbody>
</table>
Interrupt Execution

1. CPU finishes the instruction it is currently executing and stores the PC on the stack
2. CPU saves the current status of all interrupts internally
3. Interrupts are blocked at the level of the interrupt
4. Fetches the ISR address for the interrupt from IVT and jumps to that address
5. Executes the ISR until it reaches the RETI instruction
6. Upon RETI, the CPU pops back the old PC from the stack and continues with whatever it was doing before the interrupt occurred
8051 Interrupts

- There are 5 interrupts in the 8051.
  - Two external interrupts (INT0 and INT1), two timer interrupts (TF0 and TF1) and one serial port interrupt.

- Interrupts can be individually enabled or disabled. This is done in the IE (Interrupt Enable) register (A8H).
  - IE is bit addressable.

- All interrupts correspond to bits in registers.
  - Therefore, it is possible to cause an interrupt by setting the appropriate bit in the appropriate register.
    - The end result is exactly as if the hardware interrupt occurred.
The IE Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| IE.7 | EA | Enable/Disable all interrupts  
If 0 all interrupts are disabled.  
If 1, interrupts are enabled based on their individual bits |
| IE.6 | - | Reserved |
| IE.5 | ET2 | Enable/Disable Timer 2 interrupt (8052) |
| IE.4 | ES | Enable/Disable Serial Input Interrupt |
| IE.3 | ET1 | Enable/Disable Timer 1 Interrupt (TF1) |
| IE.2 | EX1 | Enable/Disable External Interrupt 1 (INT1) |
| IE.1 | ET0 | Enable/Disable Timer 0 Interrupt (TF0) |
| IE.0 | EX0 | Enable/Disable External Interrupt 0 (INT0) |

- Putting a 1 in a bit enables its interrupt.
- Putting a 0 masks that interrupt.
Interrupt Priority

• The 8051 implements 2 types of interrupt priority.

• User Defined Priority.
  – Using the IP register, the user can group interrupts into two levels – high and low.
    • An interrupt is assigned a “high” priority level by setting its bit in the IP register to 1. If the bit is set to 0, the interrupt gets a “low” priority.

• Automatic Priority.
  – Within each priority level, a strict order is observed.
    • Interrupts are polled in this sequence: INT0, TF0, INT1, TF1, Serial port.
The IP Register

- Putting a 1 in a bit assigns its interrupt to the high priority level.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP.7</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>IP.6</td>
<td>-</td>
<td>Reserved</td>
</tr>
<tr>
<td>IP.5</td>
<td>ET2</td>
<td>Timer 2 interrupt priority (8052)</td>
</tr>
<tr>
<td>IP.4</td>
<td>ES</td>
<td>Serial Port Interrupt priority</td>
</tr>
<tr>
<td>IP.3</td>
<td>ET1</td>
<td>Timer 1 Interrupt priority (TF1)</td>
</tr>
<tr>
<td>IP.2</td>
<td>EX1</td>
<td>External Interrupt 1 priority (INT1)</td>
</tr>
<tr>
<td>IP.1</td>
<td>ET0</td>
<td>Timer 0 Interrupt priority (TF0)</td>
</tr>
<tr>
<td>IP.0</td>
<td>EX0</td>
<td>External Interrupt 0 priority (INT0)</td>
</tr>
</tbody>
</table>

- Putting a 1 in a bit assigns its interrupt to the high priority level.
## Interrupts

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>FLAG bit generating the interrupt</th>
<th>SFR and Bit position</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External 0</td>
<td>IE0</td>
<td>TCON.1</td>
<td>Set by hardware when external interrupt detected. Cleared by hardware only if edge activated</td>
</tr>
<tr>
<td>External 1</td>
<td>IE1</td>
<td>TCON.3</td>
<td>Set by hardware when external interrupt detected. Cleared by hardware only if edge activated</td>
</tr>
<tr>
<td>Timer 1</td>
<td>TF1</td>
<td>TCON.7</td>
<td>Set by hardware on timer overflow. Cleared by hardware when processor vectors to interrupt routine</td>
</tr>
<tr>
<td>Timer 0</td>
<td>TF0</td>
<td>TCON.5</td>
<td>Set by hardware on timer overflow. Cleared by hardware when processor vectors to interrupt routine</td>
</tr>
<tr>
<td>Serial port</td>
<td>TI or RI</td>
<td>SCON.1 or SCON.0</td>
<td>Set by hardware on transmission or reception of data. Cleared by software</td>
</tr>
</tbody>
</table>
TCON Register

- Bit addressable.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCON.7</td>
<td>TF1</td>
<td>Timer 1 overflow flag</td>
</tr>
<tr>
<td>TCON.6</td>
<td>TR1</td>
<td>Timer 1 run-control bit. Used to turn the timer on/off</td>
</tr>
<tr>
<td>TCON.5</td>
<td>TF0</td>
<td>Timer 0 overflow flag</td>
</tr>
<tr>
<td>TCON.4</td>
<td>TR0</td>
<td>Timer 0 run-control bit.</td>
</tr>
<tr>
<td>TCON.3</td>
<td>IE1</td>
<td>External Interrupt 1 flag</td>
</tr>
<tr>
<td>TCON.2</td>
<td>IT1</td>
<td>External Interrupt 1 type flag</td>
</tr>
<tr>
<td>TCON.1</td>
<td>IE0</td>
<td>External Interrupt 0 flag</td>
</tr>
<tr>
<td>TCON.0</td>
<td>IT0</td>
<td>External Interrupt 0 type flag</td>
</tr>
</tbody>
</table>
Pending Interrupts

• If an interrupt occurs while it is disabled, or while another interrupt is active (not lower priority), it becomes pending.
  – As soon as the interrupt is enabled, it will cause a call.
  – It is also possible to cancel it by software by clearing the appropriate bit in the register.
Interrupt service routines

- Interrupt service routines must begin near the bottom of code memory
- There are only eight bytes between each interrupt entry point
- This 8 bytes might be enough for short ISR
- If the ISR is long we can use a jump instruction to jump to another area of code memory where ISR can stretch out

<table>
<thead>
<tr>
<th>Interrupt</th>
<th>Vector address</th>
</tr>
</thead>
<tbody>
<tr>
<td>System reset</td>
<td>0000H</td>
</tr>
<tr>
<td>External 0</td>
<td>0003H</td>
</tr>
<tr>
<td>Timer 0</td>
<td>000BH</td>
</tr>
<tr>
<td>External 1</td>
<td>0013H</td>
</tr>
<tr>
<td>Timer 1</td>
<td>001BH</td>
</tr>
<tr>
<td>Serial port</td>
<td>0023H</td>
</tr>
</tbody>
</table>
Example

- Write a program using Timer 0 and interrupts to create a 10 kHz square wave on P1.0

```
ORG 0
0000  LJMP  MAIN
ORG 000BH
000B  T0ISR:  CPL     P1.0
      RETI
ORG 0030H
0030  MAIN:  MOV     TMOD,#02H
0033  MOV     TH0,#-50
0036  SETB    TR0
0038  MOV     IE,#82H
003B  SJMP    $
```

- TF0 is not cleared by software because it is automatically cleared by hardware when CPU services the interrupt
Example

- Write a program using interrupts to simultaneously create 7 kHz and 500 Hz square waves on P1.7 and P1.6.
  - Timer 0 operating in mode 2 is used to generate the 7 kHz signal and timer 1 is used for 500 Hz signal operating in mode 1.

```
ORG 0
0000 LJMP MAIN
ORG 000BH
000B LJMP T0ISR
ORG 001BH
001B LJMP T1ISR
ORG 0030H
0030 MAIN: MOV TMOD, #12H
0033 MOV TH0,-710036
SETB TR0
0038 SETB TF1
003A MOV IE,#8AH
003D SJMP $
```
Example

003F  T0ISR:  CPL  P1.7
0041  RETI
0042  T1ISR:  CLR  TR1
0044  MOV  TH1,#HIGH(-1000)
0047  MOV  TL1,#LOW(-1000)
004A  SETB  TR1
004C  CPL  P1.6
004E  RETI
END
Serial port interrupts

- Serial port interrupts occur when either TI or RI is set.
- TI is set when transmission of previous character written to SBUF has finished.
- RI is set when a character has been completely received and is waiting in SBUF to be read.
- The flag that causes a serial port interrupt (TI or RI) is not cleared by hardware when CPU vectors to the interrupt.
- Software has to clear the interrupt flag (unlike timer interrupt which is cleared by hardware).
Serial port interrupts

- Write a program using interrupts to continually transmit the ASCII code set (excluding control codes) to a terminal attached to the 8051’s serial port at 1200 bit/s
  - ASCII characters to be transmitted are 7 bits and are from 20H to 7EH.

```
ORG 0
0000 LJMP MAIN
ORG 0023H
0023 LJMP SPISR
ORG 0030H
0030 MAIN: MOV TMOD, #20H
0033 MOV TH1, #-26
0036 SETB TR1
0038 MOV SCON, #42H
003B MOV A, #20H
003D MOV IE, #90H
0040 SJMP $
```
Serial port interrupts

0042  SPISR:  CJNE  A,#7FH,SKIP
0045  MOV      A,#20H
0047  SKIP:  MOV  SBUF,A
0049  INC      A
004A  CLR      TI
004C  RETI    END
External Interrupts

- External interrupts occur as a result of low-level or negative edge on INT0 or INT1 (P3.2 and P3.3)
- Flags that actually generate interrupts are bits IE0 and IE1 in TCON
- Choice of low-level activated interrupt versus negative-edge-activated interrupts is programmable through IT0 and IT1 bits in TCON
- If IT1=0 external interrupt 1 is triggered by a detected low at INT1 pin
- If IT1=1, external interrupt 1 is edge triggered
External Interrupts

• If the external interrupt is edge-activated, external source must hold request pin high for at least one machine cycle and then hold it low for at least one more cycle

• IE0 and IE1 are automatically cleared when CPU vectors to interrupts in edge-activated interrupts

• If external interrupt is level-activated, external source must hold the request active until the requested interrupt is generated

• External source then must deactivate the request before interrupt service routine is completed

• Usually an action taken in ISR causes the requesting source to return the interrupting signal to inactive state
Furnace controller

• Using interrupts, design an 8051 furnace controller that keeps a building at between 19 and 21 degrees.
  – Assume that the furnace ON/OFF solenoid is connected to P1.7 and P1.7=1 turns the furnace ON and P1.7=0 turns it off.
  – Temperature sensors are connected to INT0 and INT1 and provide HOT and COLD signals:
    \[ \text{HOT}=0 \text{ if } T>21 \]
    \[ \text{COLD}=0 \text{ if } T<19 \]
### Furnace controller

<table>
<thead>
<tr>
<th>Line</th>
<th>Assembly Code</th>
<th>Description</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>ORG 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0000</td>
<td>LJMP MAIN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0003</td>
<td>EX0ISR: CLR</td>
<td>P1.7</td>
<td></td>
</tr>
<tr>
<td>0005</td>
<td>RETI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0013</td>
<td>EX1ISR: SETB</td>
<td>P1.7</td>
<td></td>
</tr>
<tr>
<td>0015</td>
<td>RETI</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ORG 30H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0030</td>
<td>MAIN: MOV IE,#85H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0033</td>
<td>SETB ITO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0035</td>
<td>SETB IT1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0037</td>
<td>SETB P1.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0039</td>
<td>JB P3.2,SKIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td>003C</td>
<td>CLR P1.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>003E</td>
<td>SKIP: SJMP $</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

END
Intrusion Warning

• Design an intrusion warning system using interrupts that sounds a 400 Hz tone for 1 second (using a loudspeaker connected to P1.7) whenever a door connected to INT0 makes a high to low transition

• We use three interrupts: external 0 (door sensor), Timer 1 (400 Hz tone) and Timer 0 (1 second timeout)