# Chapter 5 Interrupt Handling on the MCS-51


视频讲解

## 5.1 Interrupt concepts

Microcontrollers are normally found in situations where the flow of a program will be subject to external events. These will come from hardware either outside the microcontroller or within the chip itself. Therefore an important feature of these devices is their ability to respond to signals known as interrupts which are received by the microcontroller.

The CPU usually has several lines connected to it which can receive interrupts in the form of voltage changes. When an interrupt is received, the following actions are carried out by the microcontroller:

(1) The current instruction in the main program is allowed to complete execution.

(2) The address value in the program counter (PC) is pushed to the stack.

(3) Control jumps to the start of a subprogram, known as an ISR(interrupt service routine).

(4) The ISR code is executed.

(5) When the instruction RETI (return from interrupt) is encountered in the ISR, the return address is popped from the stack into the PC.

(6) Control is returned to the original location in the main program.

An ISR (sometimes called interrupt handler), is similar in form to a subroutine. However, the great difference between the two is that the subroutine is called by an instruction within the program, while the ISR is activated by a hardware voltage change in the CPU.

A typical sequence of events is shown in Figure 5.1. The main program receives an interrupt during instruction number 5. After completing the current instruction (number 5), the address of the next instruction (number 6) is pushed onto the stack, and is called the return address. All the instructions in the ISR are then executed, after which the return address is popped off the stack, and operation is transferred back to instruction 6 in the main program. Thus the sequence of the main program has not been changed, but a time delay has occurred to allow for the execution of the ISR.

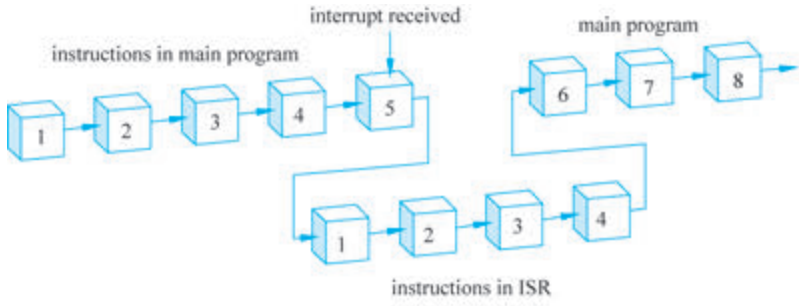


Figure 5.1 Sequence of instructions when an interrupt is received

The use of interrupts in a program gives rise to the concept of foreground and background tasks. From the functional point of view, it will seem as though the microcontroller is performing several tasks at once. However, as Figure 5.1 shows, only one is actively using the CPU at any time, and the control moves from one to the other. The Main Program is usually called the foreground task, while the various ISRs which run at different times are seen as running in the background.

## 5.2 Types of interrupt on the MCS-51


### 5.2.1 Available interrupts

There are basically six interrupts on the 8051 chip, as shown in Figure 5.2.

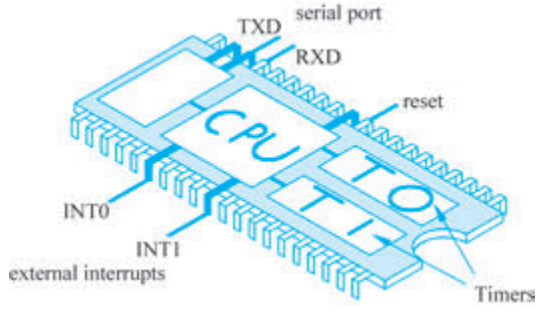


Figure 5.2 The six interrupt connections to the CPU

### 5.2.2 Reset

Strictly speaking, the reset is not an interrupt, since it does not cause the program counter to be pushed to the stack, and does not have a return instruction. However, it does cause the program control to jump to a specific address, and is therefore similar to an interrupt. A reset occurs when the RST pin on the microcontroller is brought from low (0V) to high (+5V), held high for at least two machine cycles, and then brought low again. It will then always cause operation to commence at program address 0000h. Using the circuit given in Figure 2.13 of Chapter 2, a reset will occur automatically whenever the

power to the chip is turned on, or when the reset switch is closed and opened.

### 5.2.3 External interrupts

There are two pins, known as INT0 and INT1, which are available to receive interrupts from external sources. The program-designer can choose the precise nature of the event which will cause an interrupt. This choice is made by setting values of bits in the TCON register as follows:

(1) When the IT0 or IT1 (interrupt type) bit in TCON is cleared (0) then an interrupt will be generated whenever the pin voltage (INT0 or INT1) is low. This is known as voltage level triggering.

(2) When the IT0 or IT1 bit is set (1) then an interrupt will be generated whenever the pin voltage falls from high to low. This is known as edge triggering.

### 5.2.4 Timer interrupts

The two timers/counters can also be enabled to generate interrupts when the count value reaches the maximum, which may be FFh, 1FFFh or FFFFh depending on the mode of timer operation.

### 5.2.5 Serial port interrupts

A single interrupt is used to indicate events concerning data being either received or transmitted through the serial port. The use of this interrupt frees the microcontroller from the need to constantly poll the serial port, thereby making more efficient use of the processor. On the receiver side, an interrupt is generated when new data is received by the serial port. When transmitting an item of data, an interrupt will signal when the data has actually left the microcontroller. To find which of these two events is being signaled, the interrupt flags TI (transmit interrupt) and RI (receive interrupt) may be checked.


视频讲解

## 5.3 Enabling and disabling interrupts

The process of causing the CPU to respond to an interrupt is known as enabling. It is possible to run a program with no interrupts enabled, or to enable individual interrupts during the course of the program. If interrupts are required to be active, they must be enabled by means of bits in the IE (interrupt enable) register. Figure 5.3 shows the 8051 interrupt control circuit.

All the switches in the circuit are controlled by software through the setting or clearing of bits in the appropriate SFRs. The IE register contains single bits for all the interrupts, except for Reset. Before any interrupt can be enabled, it is necessary to set the EA (Enable All) bit, after which the individual interrupts can be enabled by their relevant bits. It should be noted that EA does not actually enable all interrupts, but allows them to

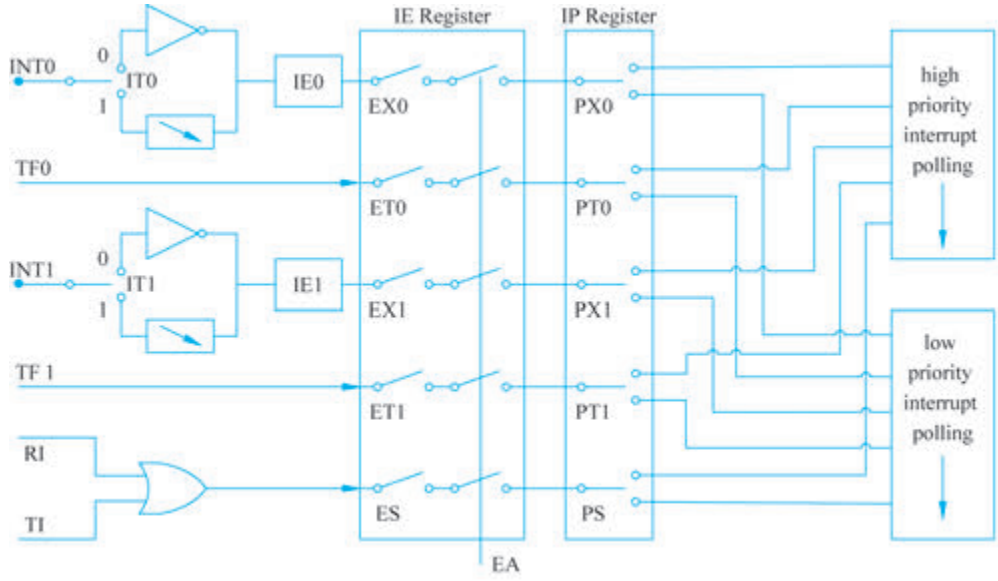


Figure 5.3 8051 interrupt control circuit

be enabled separately. Another way of viewing this is to say that if the EA bit is cleared, then all the interrupts will be disabled. The meanings of IE register bits are shown in Table 5.1.

Table 5.1 The meanings of each bit in IE register

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| EA | — | — | ES | ET1 | EX1 | ET0 | EX0 |

Bit 7-EA=all interrupts disabled if cleared

Bit 6-not implemented

Bit 5-not implemented

Bit 4-ES=serial interrupt

Bit 3-ET1=timer 1 interrupt

Bit 2-EX1=external interrupt 1

Bit 1-ET0=timer 0 interrupt

Bit 0-EX0=external interrupt 0

For example, the following instructions will enable the Timer 1 and Serial Port interrupts:

```
SETB   EA
SETB   ET1
SETB   ES
```

## 5.4 Interrupt flags

When an interrupt is received, the event will always be signaled by setting an appropriate interrupt flag. Each type of interrupt has its own flag, which is actually a bit

in the TCON or SCON registers. Even when the particular interrupt is not enabled, the flag will still be set to indicate that an interrupt signal has been received. The names of the flags, all of which can be used in assembly language programs, are given in Table 5.2.

Table 5.2 List of interrupt flags on the 8051

| Interrupt name | Name of flag | SFR location of flag | How flag is cleared |
|---|---|---|---|
| external Interrupt 0 | IE0 (interrupt edge) | TCON | if IT0 = 1, cleared by ISR<br>if IT0 = 0, cleared by program |
| external Interrupt 1 | IE1 (interrupt edge) | TCON | if IT1 = 1, cleared by ISR<br>if IT1 = 0, cleared by program |
| Timer Interrupt 0 | TF0 | TCON | cleared at start of ISR |
| Timer Interrupt 1 | TF1 | TCON | cleared at start of ISR |
| serial port interrupt | RI (receive)<br>TI (transmit) | SCON | cleared by program |

Table 5.2 shows how an interrupt flag is cleared after an interrupt has been received. In some cases it is cleared automatically when the ISR begins. In other cases, the program-designer must ensure that the program contains an instruction to clear the flag, either in the ISR or the main program. It will be seen from the circuit of Figure 5.3 that the TF0, TF1, RI and TI flags provide the bit inputs to the interrupt control circuit.


视频讲解

## 5.5 Handling simultaneous interrupts

### 5.5.1 Interrupt priorities

If two interrupts are received during the execution of the same instruction, the microcontroller must decide which one should be serviced first. This is achieved by ranking the interrupts in a priority order as shown in Table 5.3.

Table 5.3 Priority ranking of interrupt sources

| Interrupt sources | Priority rank |
|---|---|
| External Interrupt 0 | Highest priority |
| Timer 0 | ↓ |
| External Interrupt 1 | ↓ |
| Timer 1 | ↓ |
| Serial Port | Lowest priority |

Note: The priority order of Timer 0, External Interrupt 1 and Timer 1 is from high to low.

Thus, if two interrupts occur during the same instruction, the one with the higher priority will be serviced first, followed by the other. Figure 5.4 shows an example where instruction 3 of the main program is interrupted by both External Interrupt 0 and Timer 0. The two interrupt ISRs will be run in the order shown, according to their priorities.

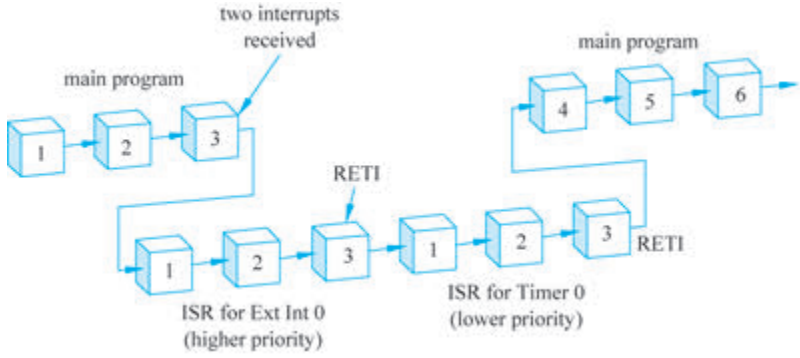


Figure 5.4 Sequence of instructions when two interrupts are received together

## 5.5.2 Interruption of an ISR by a lower priority interrupt

Sometimes one interrupt is received and the ISR starts to run, but during the execution of the ISR another interrupt is received. The action taken by the microcontroller depends on the relative priority of the second interrupt. In the example shown in Figure 5.5, the main program is interrupted by External Interrupt 0 at instruction 3. The ISR starts to run, and at instruction 2 an interrupt from Timer 0 is received. Since the second interrupt has lower priority than the first, the ISR for External Interrupt 0 will be completed, and then the ISR for Timer 0 will run.

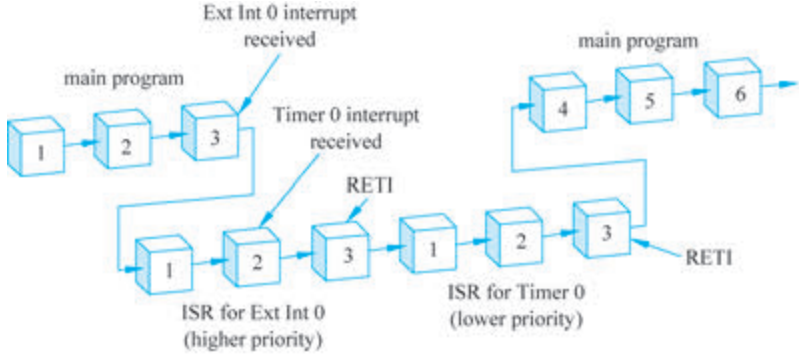


Figure 5.5 Interruption of a higher priority interrupt by a lower one

## 5.5.3 Interruption of an ISR by a higher priority interrupt

If the second interrupt has a higher priority than the one running the ISR, then the ISR will be suspended while the other ISR is run. At completion of this, the first ISR will continue to run. Figure 5.6 shows the case where Timer 0 interrupts the main program, and then the timer ISR is interrupted by External Interrupt 0.

## 5.5.4 Changing the interrupt priorities

The 8051 system allows the program-designer to assign individual interrupts to one of

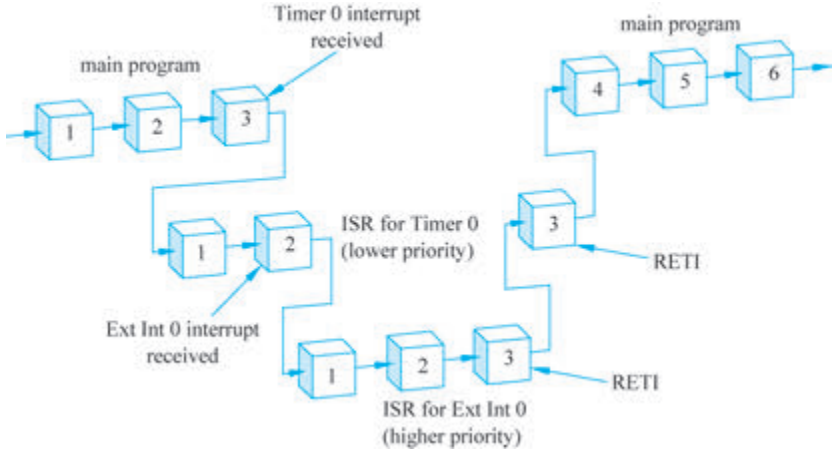


Figure 5.6 Interruption of a lower priority interrupt by a higher one

two priority levels, so that important interrupt sources receive preferential treatment. This is achieved by setting the appropriate bits in the IP (interrupt priority) register. Setting a bit gives a high level of priority, while clearing it gives a low level. The meanings of IP register bits is shown in Table 5.4.

Table 5.4 The meanings of each bit in IP register

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| — | — | — | PS | PT1 | PX1 | PT0 | PX0 |

Bit 7-Bit5-Not implemented

Bit 4-PS=serial Interrupt

Bit 3-PT1=Timer 1 Interrupt

Bit 2-PX1=external Interrupt 1

Bit 1-PT0=Timer 0 Interrupt

Bit 0-PX0=external Interrupt 0

For example, if one wanted to ensure that External Interrupt 1 was always serviced in preference to all the other interrupts, the following instructions could be used:

```
SETB   PX1
CLR    PS
CLR    PT1
CLR    PT0
CLR    PX0
```


视频讲解

## 5.6 Locations of ISRs in memory

Unlike some other microprocessors and microcontrollers, the 8051 does not have interrupt vectors (indirect addresses pointing to ISRs). Instead each interrupt causes the program to jump directly to an address at which an ISR is located. If the address space for the ISR is too small, a jump may then be made to a free area of the program memory to

continue the ISR execution. In the case of the Reset interrupt there are only three bytes allocated, since the intention is that a jump instruction be programmed there to transfer execution to the appropriate starting address. The locations of the ISRs are given in Table 5.5 and the relevant memory map in Figure 5.7.

**Table 5.5 Locations of ISRs in memory**

| Interrupt name | ISR starting address |
|---|---|
| reset | 0000h |
| external Interrupt 0 | 0003h |
| Timer 0 | 000Bh |
| external Interrupt 1 | 0013h |
| Timer 1 | 001Bh |
| serial port | 0023h |

When interrupts are in use, it is necessary for the main program to jump over the ISRs, although it should be noted that in cases where the ISR has very few instructions it is possible to write them in the original ISR space without the need to jump to another location.

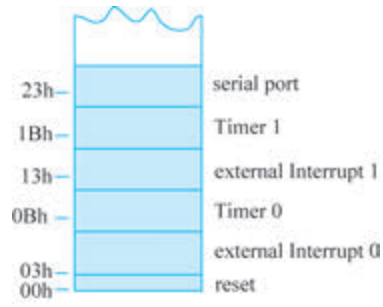


**Figure 5.7 Memory map of ISRs**

One way to jump over the ISRs is by putting the main program code at a fixed address above the ISR memory space, as illustrated in this program section:

```
        ORG 0000h
        SJMP 0050h          ; Jump over ISRs
        ORG 0003h
        AJMP EXTINT0        ; Jump to ISR for External Interrupt 0
        ORG 000Bh
        AJMP TIMER0         ; Jump to ISR for Timer 0
        ORG 0013h
        AJMP EXTINT1        ; Jump to ISR for External Interrupt 1
        ORG 001Bh
        AJMP TIMER1         ; Jump to ISR for Timer 1
        ORG 0023h
        AJMP SERIAL         ; Jump to ISR for Serial Port
        ORG 0050h           ; Start of Main program
MAIN:   --------------
        --------------
```

This arrangement will waste memory between the end of the serial port ISR and the start of the main program. It is more efficient to use labels to allow the assembler to fit the machine code together without spaces. The following shows the same program section with the location of the main program being set by the label MAIN:

```
        ORG 0000h
        SJMP MAIN           ; Jump over ISRs
```

```
            ORG 0003h
            AJMP EXTINT0        ; Jump to ISR for External Interrupt 0
            ORG 000Bh
            AJMP TIMER0         ; Jump to ISR for Timer 0
            ORG 0013h
            AJMP EXTINT1        ; Jump to ISR for External Interrupt 1
            ORG 001Bh
            AJMP TIMER1         ; Jump to ISR for Timer 1
            ORG 0023h
            AJMP SERIAL         ; Jump to ISR for Serial Port
MAIN:       --------------      ; Start of Main program
            --------------
```

These two methods of jumping-over the ISRs are illustrated in Figure 5.8.

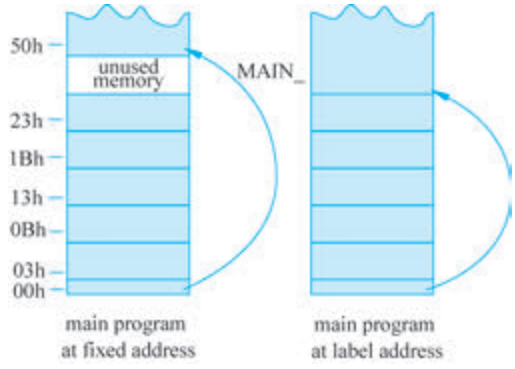


Figure 5.8 Two methods of jumping-over the ISR memory


视频讲解

## 5.7 Handling multiple-source interrupts

Although the two external interrupts provided on the 8051 chip are adequate for small projects, there is sometimes a need for a program to be capable of receiving interrupts from a larger number of sources. This multi-interrupt requirement can be met by the addition of inverter gates, and by using some of the I/O port pins for polling the interrupts. The circuit of Figure 5.9 shows a situation where External Interrupt 0 is used to service five interrupt lines, each of which will go high to signal an interrupt.

For this circuit, the following program uses a jump from a main ISR to five virtual-ISRs, which are actually subroutines containing the code for the five operations:

```
            ORG   0000h
            AJMP  MAIN
            ORG   0003h           ; ISR for INT0
            AJMP  ISR_INT0
MAIN:       ------------          ; Main program activity
            ------------
            SJMP  $
ISR_INT0:   JB    P1.1, ISR1      ; Poll the interrupt lines
            JB    P1.2, ISR2
```

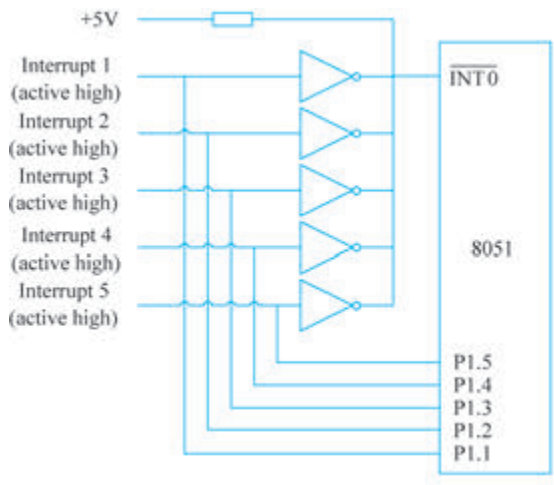


Figure 5.9 Connection of five interrupt sources to the 8051

```
          JB    P1.3, ISR3
          JB    P1.4, ISR4
          JB    P1.5, ISR5
ISR1:     ------------      ; ISR for Interrupt 1
          ------------
          RETI
ISR2:     ------------
          ------------
          RETI
ISR3:     ------------
          ------------
          RETI
ISR4:     ------------
          ------------
          RETI
ISR5:     ------------
          ------------
          RETI
```

## 5.8 Use of a latch for level interrupts

When INT0 or INT1 is configured for voltage-level triggering, it is necessary to turn off the interrupt source before the end of the ISR. Otherwise the interrupt will be repeatedly serviced as long as the INT0/INT1 pin remains low. To overcome this problem, the circuit of Figure 5.10 can be used in which the 74LS74 D-type latch is used to remove the interrupt on pin INT0 by a software instruction in the ISR.

The following program shows the way in which the setting of P1.0 causes the output at *Q* to go high, thereby removing the interrupt. If the interrupt source remains high or low, there will be no further interrupts until there is a low-to-high transition on the CLK pin.

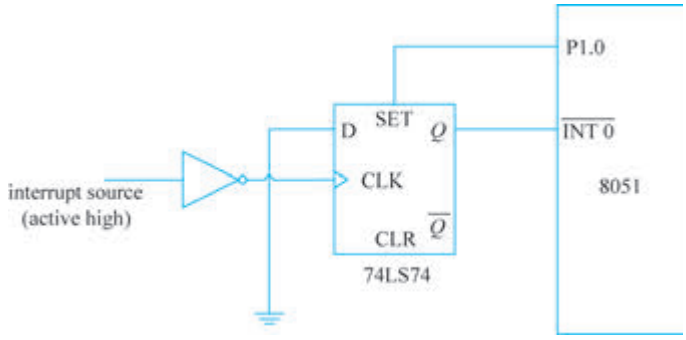


Figure 5.10 Connection of latch for level-actuated interrupt

```
            ORG   0000h
            AJMP  MAIN
            ORG   0003h
            AJMP  ISR_INT0
MAIN:       SETB  EA
            SETB  EX0
            CLR   IT0           ; Enable INT0 for level triggering
LOOP:       ---------------
            ----------------; Main program actions
            SJMP  LOOP
ISR_INT0:
            SETB  P1.0          ; Set D-type latch
            NOP
            CLR   P1.0          ; Clear D-type latch

            ---------------
            ---------------  ; ISR actions
            RETI
```

## 5.9 Examples of programs with interrupt operations

### 5.9.1 Control of traffic lights

**Task:**

Write a program to control the traffic lights shown in Figure 5.11. Each sensor is a photocell (Light Dependent Resistor) connected so that, when a vehicle is present, the input pin to the 8051 will go high, and the external interrupt will go low. To simplify the problem consider only two sets of traffic lights and sensors, one in each direction. The microcontroller should run a main program which is interrupted each time a car blocks a sensor. The lights should then change to allow the vehicle to proceed.

The connections of the light sensors to the interrupt pin are shown in the circuit of Figure 5.12.

**Possible solution:**

Use the following pin assignments for the connections:

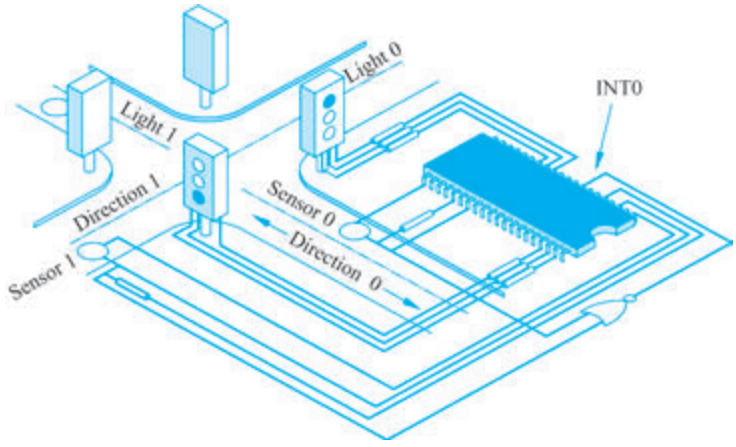


Figure 5.11 Microcontroller-operated traffic lights

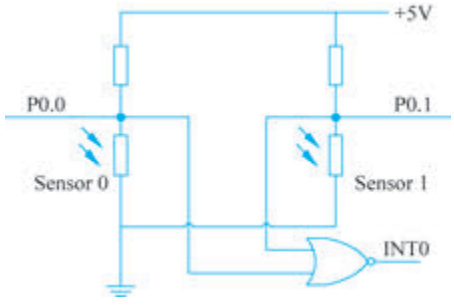


Figure 5.12 Connections between the sensors and the 8051 interrupt pin

| | | | |
|---|---|---|---|
| Sensors: | Sensor 0=P0.0 | Sensor 1=P0.1 | |
| Light 0: | Red 0=P1.0 | Yellow 0=P1.1 | Green 0=P1.2 |
| Light 1: | Red 1=P1.3 | Yellow 1=P1.4 | Green 1=P1.5 |

The main program simply initializes the registers, and then runs some activity in a loop, as shown in the flowchart of Figure 5.13. The Interrupt Service Routine (ISR) then responds to changes from the sensors. The flowchart of Figure 5.14 shows the ISR. The two subroutines which change the direction of the traffic lights are given in Figure 5.15.

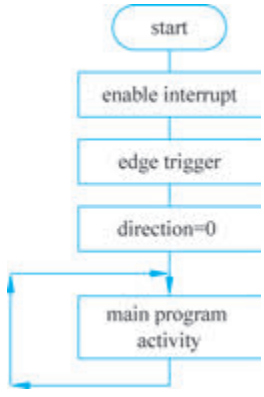


Figure 5.13 Flowchart for traffic light main program

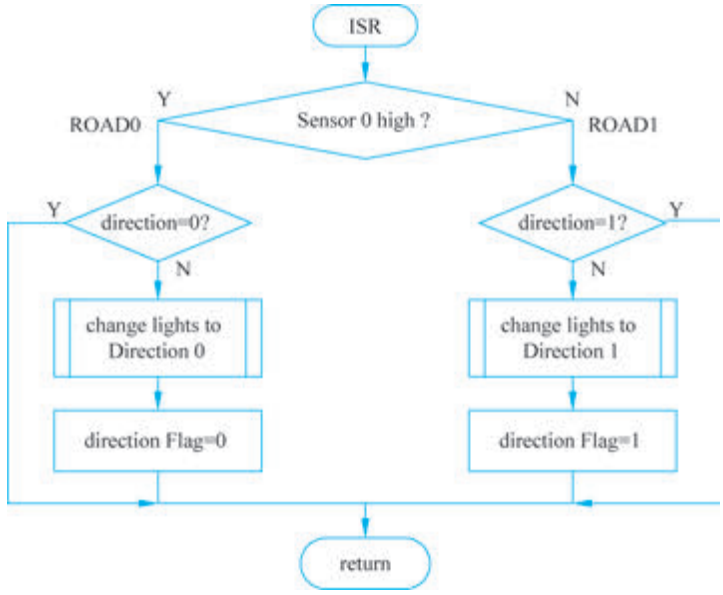


Figure 5.14 Flowchart of the interrupt service routine for traffic light control

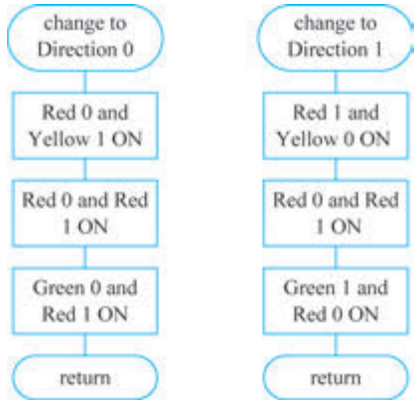


Figure 5.15 Flowcharts of the subroutines to change traffic light directions

The following assembly language program can then be developed. (Some of the obvious EQU statements have been omitted to save space.)

```
DIRFLAG     EQU   20.0h          ; Direction Flag to indicate Green direction
            ORG   0000h
            SJMP  START
            ORG   0003h
            AJMP  ISRINT0
START:      SETB  EA
            SETB  EX0            ; Enable interrupt INT0
            SETB  IT0            ; Set for edge triggering
            CLR   DIRFLAG        ; Set Direction Flag to be 0
MAIN:       NOP
            NOP                  ; Main program activity
            SJMP  MAIN
```

```
; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
ISRINT0:            ; Interrupt Service Routine
                CLR   EA                ; Disable interrupts during ISR
                JNB   SENSOR0, ROAD1
ROAD0:          JNB   DIRFLAG, RETURN
                ACALL CHANGE_TO_0
                CLR   DIRFLAG
                SJMP  RETURN
ROAD1:          JB    DIRFLAG, RETURN
                ACALL CHANGE_TO_1
                SETB  DIRFLAG
RETURN:         SETB  EA                ; Enable interrupts again
                RETI
; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ +
   ; Subroutines for changing direction
CHANGE_TO_0:  ; Operations to change direction of lights from 1 to 0
                ACALL LIGHTS_OFF
                CLR   RED0              ; Turn on Red 0
                CLR   YELLOW1           ; Turn on Yellow 1
                ACALL DELAY
                ACALL LIGHTS_OFF
                CLR   RED0              ; Turn on Red 0
                CLR   RED1              ; Turn on Red 1
                ACALL DELAY
                ACALL LIGHTS_OFF
                CLR   GREEN0            ; Turn on Green 0
                CLR   RED1              ; Turn on Red 1
                ACALL DELAY
                RET
CHANGE_TO_1:        ;Operations to change direction of lights from 0 to 1
                ACALL LIGHTS_OFF
                CLR   RED1              ; Turn on Red 1
                CLR   YELLOW0           ; Turn on Yellow 0
                ACALL DELAY
                ACALL LIGHTS_OFF
                CLR   RED1              ; Turn on Red 1
                CLR   RED0              ; Turn on Red 0
                ACALL DELAY
                ACALL LIGHTS_OFF
                CLR   GREEN1            ; Turn on Green 1
                CLR   RED0              ; Turn on Red 0
                ACALL DELAY
                RET
LIGHTS_OFF:   ; To turn off all the lights
                SETB  RED0
                SETB  YELLOW0
                SETB  GREEN0
                SETB  RED1
                SETB  YELLOW1
                SETB  GREEN1
                RET
```

The subroutine DELAY would use loops to provide the required time delay. In practice there would be different delays between the various stages of the program.

## 5.9.2 Remote sensor with switch interrupt

**Task:**

An 8051 receives temperature values from a remote sensor, as shown in Figure 5.16. Each time the A-D converter sends a new value to Port 0, it will also send a short pulse on the P1.0 pin. Write a program to push the received values of temperature onto the stack. In addition, a switch is connected to the Interrupt 0 line. The program should sense when P3.2 pin goes from +5V to 0V, and then calculate the average value of the last 10 data values on the stack. This average should be stored in location 7Fh. (It should be assumed that values in the calculation never exceed 255.)

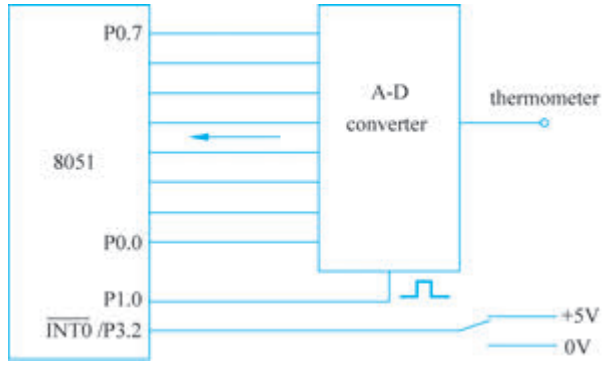


Figure 5.16 Remote temperature circuit with interrupt facility

**Possible solution:**

The main program will wait for pulses on the P1.0 line, and then push the data values to the stack. When External Interrupt 0 is received, the ISR will carry out the calculation of the average of the topmost 10 items on the stack. A program of the following form can then be written:

```
            TEMP    EQU 30h         ; Use a location for temp storage
            ORG     0000h
            SJMP    MAIN
            ORG     0003H
            AJMP    ISR_INT0
MAIN:       SETB    IT0             ; Edge triggering of INT0
            SETB    EA
            SETB    EX0
            SETB    P1.0            ; P1.0 as an input
NEXT_TEMP:  JNB     P1.0, $         ; Wait for pulse high
            JB      P1.0, $         ; Wait for pulse low
            PUSH    P0
            SJMP    NEXT_TEMP
ISR_INT0:   MOV     R0, #0Ah        ; Number of items to be averaged
            CLR     A
```

```
NEXT_NUM:  POP     TEMP
           ADD     A, TEMP          ; Add the top 10 values
           DJNZ    R0, NEXT_NUM
           MOV     B, #0Ah
           DIV     AB               ; Calculate average
           MOV     7Fh, A           ; Store result
           RETI
```

### 5.9.3 Switching between flashing LEDs by means of an interrupt


视频讲解

**Task:**

With reference to the circuit shown in Figure 5.17, write a program to flash one of the LEDs at a frequency of 10Hz. Every time the switch is closed, an interrupt should flash the other LED instead. If, subsequently, the switch remains closed the same LED must continue to flash.

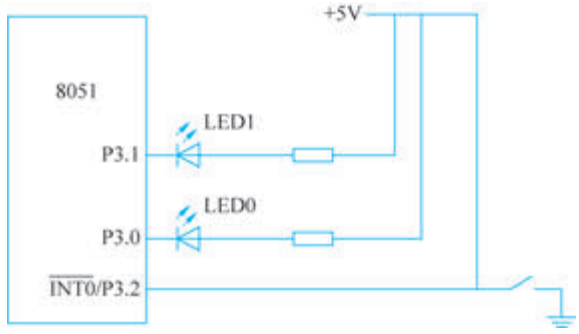


Figure 5.17 Flashing LED circuit

**Possible solution:**

The following program uses a bit called FLAG to indicate which of the LEDs is currently flashing. The main program controls the flashing, while the ISR toggles the value of the flag whenever an edge-triggered interrupt is received.

```
           LED0    BIT  P3.0
           LED1    BIT  P3.1
           FLAG    BIT  00h
           ORG     0000h
           SJMP    MAIN
           ORG     0003h
           AJMP    ISR_INT0
MAIN:      CLR     FLAG             ; Start with LED0 on
           SETB    IT0              ; Edge triggering
           SETB    EA
           SETB    EX0              ; Interrupt enable
CHECK:     JB      FLAG, LIGHT
           CPL     LED0
           SETB    LED1
           ACALL   DELAY            ; 50ms delay subroutine
           SJMP    CHECK
```

```
LIGHT:    CPL     LED1
          SETB    LED0
          ACALL   DELAY           ; 50ms delay subroutine
          SJMP    CHECK
ISR_INT0: CPL     FLAG
          RETI
DELAY:    MOV     R0, #0FFH
LOOP:     MOV     R1, #0C3H
          DJNZ    R1, $
          DJNZ    R0, LOOP
          RET
          END
```

It should be noted that the above example is a case where the ISR does not really need to call a subroutine. Since the ISR contains only two instructions, these can be put directly into the memory space at address 0003h. The program can then be simplified, as follows:

```
          LED0    BIT  P3.0
          LED1    BIT  P3.1
          FLAG    BIT  00h
          ORG     0000h
          AJMP    MAIN
          ORG     0003h         ; ISR located at address 0003h
          CPL     FLAG
          RETI
MAIN:     CLR     FLAG          ; Start with LED0
          SETB    IT0           ; Edge triggering
          SETB    EA
          SETB    EX0           ; Interrupt enable
CHECK:    JB      FLAG, GREEN
LED0:     CPL     LED0
          ACALL   DELAY         ; 50ms delay subroutine
          SJMP    CHECK
GREEN:    CPL     LED1
          ACALL   DELAY         ; 50ms delay subroutine
          SJMP    CHECK
```

## 5.10 Exercises on the chapter

**Question 1.**

List the various types of interrupt on the 8051 microcontroller.

**Question 2.**

If interrupts are received from the $\overline{\text{INT0}}$ pin and Timer 1 during the execution of the same instruction, what action will the microcontroller take?

**Question 3.**

A microcontroller receives an interrupt from the serial port, and starts running the ISR. During the execution of this ISR, another interrupt is received from Timer 0. What

action will be taken?

**Question 4.**

If the IE register contains the number 08h, what action will be taken if interrupts are received at the same time from External Interrupt 0 and Timer 1.

**Question 5.**

A microcontroller receives an interrupt from Timer 1, and starts running the ISR. During the execution of this ISR, another interrupt is received from the serial port. What action will be taken?

**Question 6.**

The IP register contains 04h, and IE contains 9Fh. An interrupt is received on pin INT0, and the ISR starts to run. While this ISR is running, an interrupt is received on INT1. What action will be taken?

**Question 7.**

The IP register contains 10h, and the IE register 82h. An interrupt is received from Timer 0, and the ISR starts to run. During the execution of this ISR, an interrupt is received from the serial port. What action will be taken?

**Question 8.**

If the IT1 bit is set when the input to pin $\overline{\text{INT1}}$ is low, what effect will this have?

**Question 9.**

With reference to the circuit diagram of Figure 5.17, write a program in which the lights do not flash. The Main Program should execute a random loop, while the ISR should change the lights in the form of a 2-bit binary number. LED1 is the MSB and LED0 is the LSB. Each time an edge-triggered interrupt is received the binary number should be incremented.

**Question 10.**

With reference to the traffic light diagrams of Figure 5.11 and Figure 5.12, write a program in which the lights are normally green in Direction 0. When a vehicle is present on Sensor 1, the lights should change to allow the vehicle to proceed in Direction 1, and then after 20 seconds return to Direction 0. Assume a subroutine DELAY20 is available for the delay.

**Question 11.**

As Figure 5.18 shows, a circuit is designed to control an LED using a pushbutton, which is connected so as to pull the $\overline{\text{INT0}}$ line to ground. The cathode of the LED is connected to pin P1.0, while the anode is connected through a pull-up resistor to +5V. One push of the button turns on the LED, and a following push turns it off. This function is repeated indefinitely. In the following program, fill in the blanks.

```
LED     ____    P1.0        ; LED on P1.0
FLAG    ____    20h.4       ; Flag to indicate light on
ORG     0000h
```

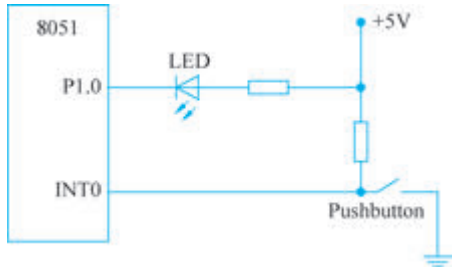


Figure 5.18 On/off LED circuit

```
        AJMP    MAIN
        ORG     _______             ; ISR for External Interrupt 0
        AJMP    ISR_INT0
MAIN:   SETB    EA
        SETB    ____                ; Set edge triggering
        SETB    EX0                 ; Enable the External Interrupt 0
        SETB    FLAG                ; Initially light flagged as on
        SJMP    $
ISR_INT0: ____  FLAG                ; Complement the button flag
        MOV     C, FLAG
        ____    LIGHT               ; If  Flag = 0, then turn on the LED
        ____    LED                 ; If  Flag = 1, then turn off the LED
        AJMP    EXIT
LIGHT:  CLR     LED                 ; Turn on the LED
EXIT:   ____                        ; Return from interrupt
```

**Question 12.**

An 8051 microcontroller, with an oscillator of 11.0592MHz, runs the following program. Exactly 9$\mu$s after starting the program, an interrupt is received on the INT1 pin. What will be the data value in accumulator A when the program finishes?

```
        ORG   0000h
        SJMP  MAIN
        ORG   0013h
        AJMP  ISR_INT1
MAIN:   SETB  IT1
        SETB  EA
        SETB  EX1
        MOV   A, #54h
        INC   VA
        CLR   C
        SUBB  A, #40h
        INC   A
FINISH: SJMP  $
ISR_INT1: CPL A
        RETI
```

## 5.11 English-Chinese vocabulary from the chapter

| English | Chinese | Pin Yin |
|---|---|---|
| D-type latch | D 型锁存器 | D xing2 suo3 cun2 qi4 |
| Edge triggering | 边沿触发 | bian1 yan2 chu4 fa1 |
| Enabling and disabling | 允许和屏蔽 | yun3 xu3 he2 ping2 bi4 |
| Hardware | 硬件 | ying4 jian4 |
| Interrupt vector | 中断矢量；中断向量 | zhong1 duan4 shi3 liang4；zhong1 duan4 xiang4 liang4 |
| IP (interrupt priority) | 中断优先级 | zhong1 duan4 you1 xian1 ji2 |
| ISR(interrupt service routine) | 中断服务程序 | zhong1 duan4 fu2 wu4 cheng2 xu4 |
| Poll | 轮询 | lun2 xun2 |
| Return address | 返回地址；断点地址 | fan3 hui2 di4 zhi3；duan4 dian3 di4 zhi3 |
| Voltage level triggering | 电平触发 | dian4 ping2 chu4 fa1 |
| Vector | 向量；矢量 | xiang4 liang4；shi3 liang4 |