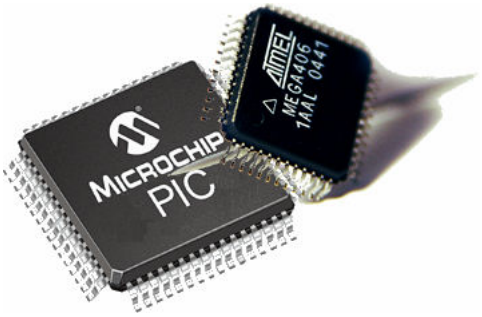
Interrupts



Interrupts

An interrupt causes a temporary diversion of program execution in a similar sense to a program subroutine call, but an interrupt is triggered by some event, external to the currently operating program. We say the interrupt event occurs *asynchronously* to the currently operating program as it is not necessary to know in advance when the interrupt event is going to occur.

4.1 8051 INTERRUPTS

There are five interrupt sources for the 8051. Since the main RESET input can also be considered as an interrupt, six interrupts can be listed as follows:

Interrupt	Flag	Vector address
System RESET	RST	0000h
External interrupt 0	**IE0**	**0003h**
Timer/counter 0	TF0	000Bh
External interrupt 1	**IE1**	**0013h**
Timer/counter 1	TF1	001Bh
Serial port	RI or TI	0023h

We will concentrate on the external interrupts for now, and later we will examine the other interrupt sources. Here's a brief look at some of the register bits which will be used to set up the interrupts in the example programs.

The Interrupt Enable, IE, register is an SFR register at location A8h in Internal RAM. The EA bit will enable all interrupts (when set to 1) and the individual interrupts must also be enabled.

Interrupt Enable register

EA *msb*			**ES**	**ET1**	**EX1**	**ET0**	**EX0** *lsb*

For example, if we want to enable the two external interrupts we would use the instruction:

```
MOV  IE, #10000101B
```

Each of the two external interrupt sources can be defined to trigger on the external signal, either on a negative going edge or on a logic low level state. The negative edge trigger is usually preferred as the interrupt flag is automatically cleared by hardware, in this mode. Two bits in the TCON register are used to define the trigger operation. The TCON register is another SFR register and is located at location 88h in Internal RAM. The other bits in the TCON register will be described later in the context of the hardware Timer/Counters.

TCON register

msb					**IT1**		**IT0** *lsb*

To define negative edge triggering for the two external interrupts use instructions as follows:

```
SETB IT0        ; negative edge trigger for interrupt 0
SETB IT1        ; negative edge trigger for interrupt 1
```

Figure 4.1 shows the flow of operation when a system is interrupted. In the example it is assumed that some program, say the main program, is executing when the external interrupt INT0 occurs. The 8051 hardware will automatically complete the current machine level (assembler level) instruction and save the Program Counter to the stack. The IE register is also saved to the stack. The IE0 flag is disabled (cleared) so that another INT0 interrupt will be inhibited while the current interrupt is being serviced. The Program Counter is now loaded with the vector location 0003h. This vector address is a predefined address for interrupt INT0 so that program execution will always trap to this address when an INT0 interrupt occurs. Other interrupt sources have uniquely defined vector addresses for this purpose. The set of these vector addresses is referred to as the interrupt vector table.

Program execution is now transferred to address location 0003h. In the example a LJMP instruction is programmed at this address to cause the program to jump to a predefined start address location for the relevant ISR (Interrupt Service Routine) routine. The ISR routine is a user written routine, which defines what action is to occur following the interrupt event. It is good practice to save (PUSH) to the stack any registers used during the ISR routine and to restore (POP) these registers at the end of the ISR routine, thus preserving the registers' contents, just like a register is preserved within a subroutine program. The last instruction in the ISR routine is a RETI (RETurn from Interrupt) instruction and this instruction causes the 8051 to restore the IE register values, enable the INT0 flag, and restore the Program Counter contents from the stack.

Since the Program Counter now contains the address of the next instruction which was to be executed before the INT0 interrupt occurred, the main program continues as if it had never being interrupted. Thus only the temporal behaviour of the interrupted program has been affected by the interrupt; the logic of the program has not been otherwise affected.

4.2 EXAMPLE INTERRUPT DRIVEN PROGRAM

Figure 4.2 shows and oven control system where a heating oven, as part of a manufacturing process, is to be controlled within the temperature range, between 190^{o}C and 200 ^{o}C . An 8051 microcomputer based system is used to control the temperature. The oven has two built-in temperature sensors. The low threshold sensor outputs a logic 0 if the temperature is below 190 ^{o}C, otherwise it outputs a logic high level (say 5 volts). The high threshold sensor outputs a logic low level if the temperature exceeds 200 ^{o}C, otherwise it outputs a logic high level. The temperature sensors are connected to the 8051's interrupt inputs, INT0 and INT1, as shown in the diagram. Both of these interrupt

inputs are set to trigger at negative voltage transitions. The microcomputer outputs a logic 1 on the P1.0 output pin to turn on the heater element and it outputs a logic 0 to turn off the heating element. Assume the necessary hardware driver circuitry, to switch power to the oven, is included in the oven.

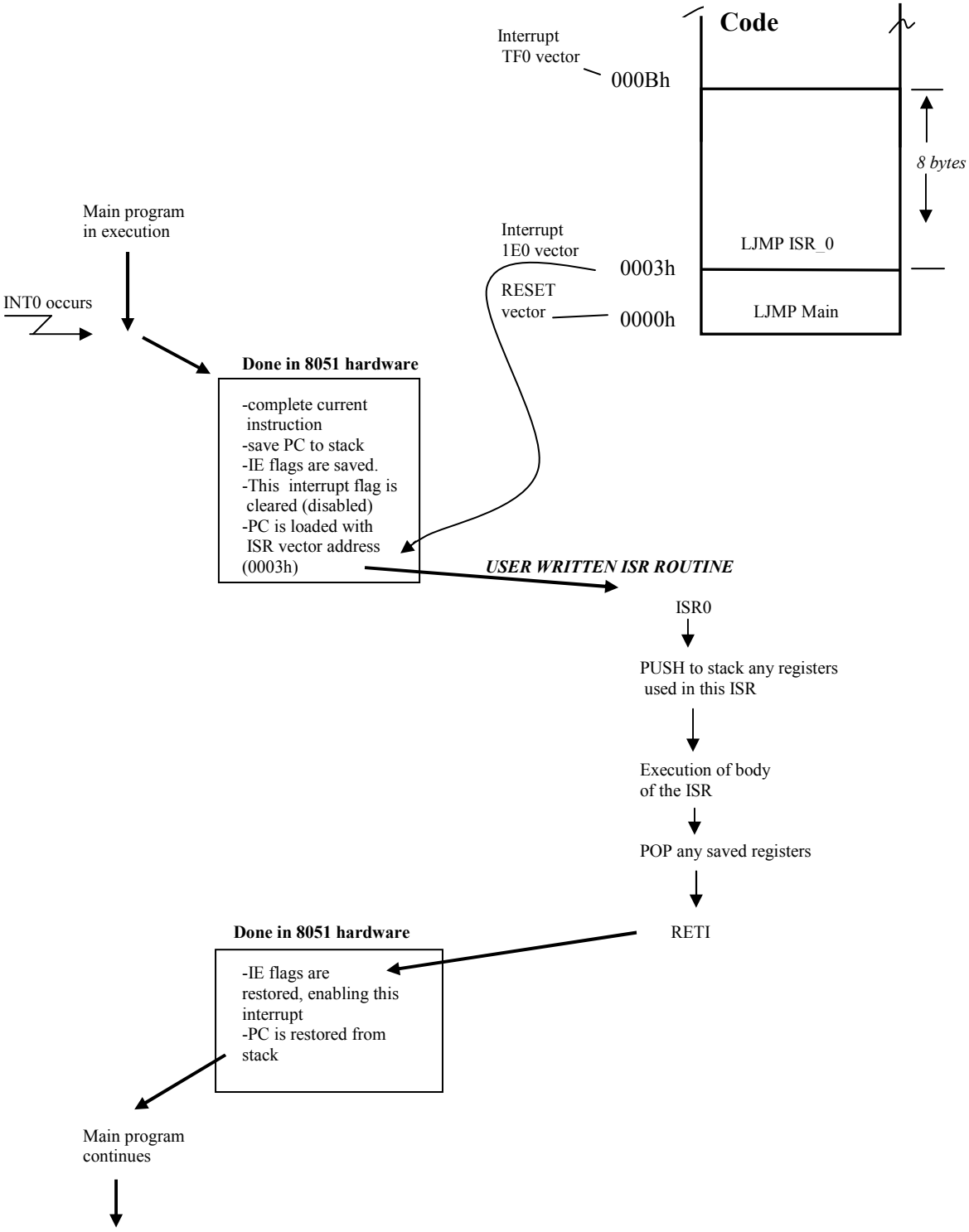


Figure 4.1 Interrupt operation example

The microcomputer's program is written so that an interrupt from the low threshold sensor will cause the heating element to turn on and interrupt from the high threshold sensor will cause the heating element to turn off. Figure 4.3 shows a timing diagram for the oven's operation.

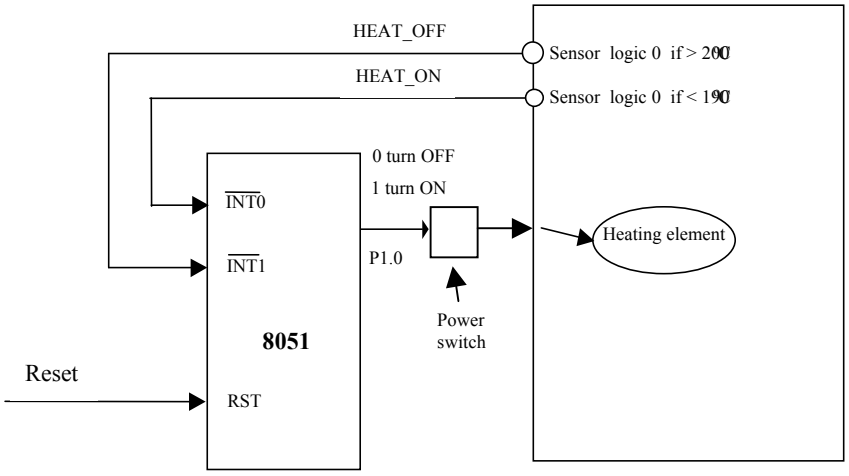


Figure 4.2 Temperature controlled heating oven

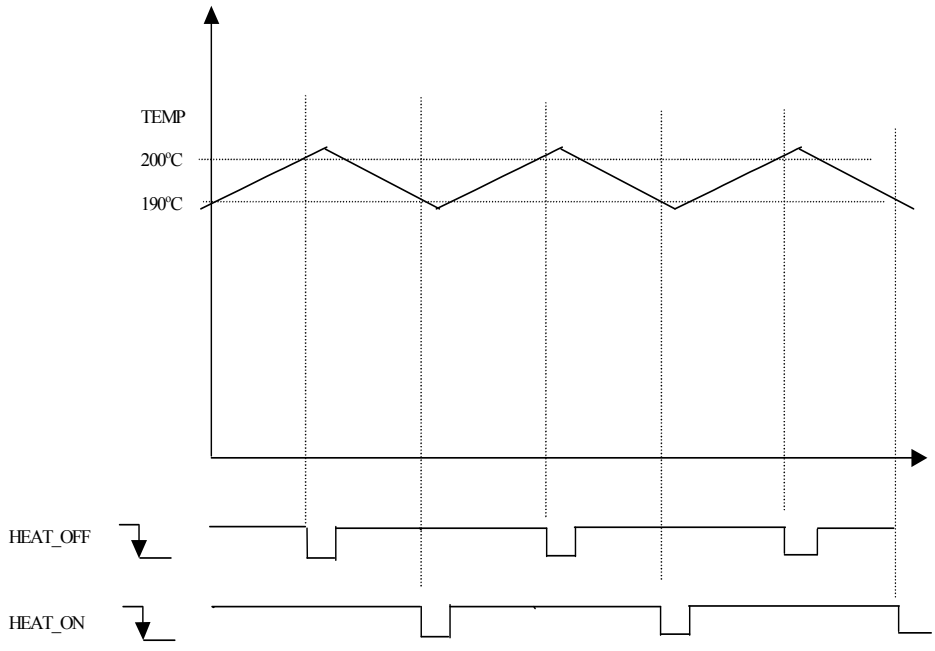


Figure 4.3 Timing diagram for the oven control

The assembler language source program to control the oven is shown in listing 4.1. Since the ISR routines (**I**nterrupt **S**ervice **R**outines) are very short they could have been positioned within the 8 bytes of memory available at the respective vector locations. However, the ISR routines are located higher up in memory to show the memory positioning structure which would be used for larger ISR routines. Three vector locations are defined at the beginning of the program. The RESET vector, at address 0000h, contains a jump instruction to the MAIN program. Location 0003h is the vector location for external interrupt 0, and this contains a jump instruction to the relevant ISR routine, ISR0. External interrupt 1uses the vector location 0013h which contains a jump instruction to the ISR routine, ISR1. In this oven control program example the main program just loops around doing nothing. When an interrupt occurs, the required action is carried out by the relevant ISR routine. However, in a more sophisticated program the main program could be doing something very useful and would be interrupted only when the oven temperature needs to be adjusted, on or off. Thus the main program does not have to waste time polling the sensor inputs.

The resulting allocation of space in code memory for the OVEN.A51 program is shown in figure 4.4

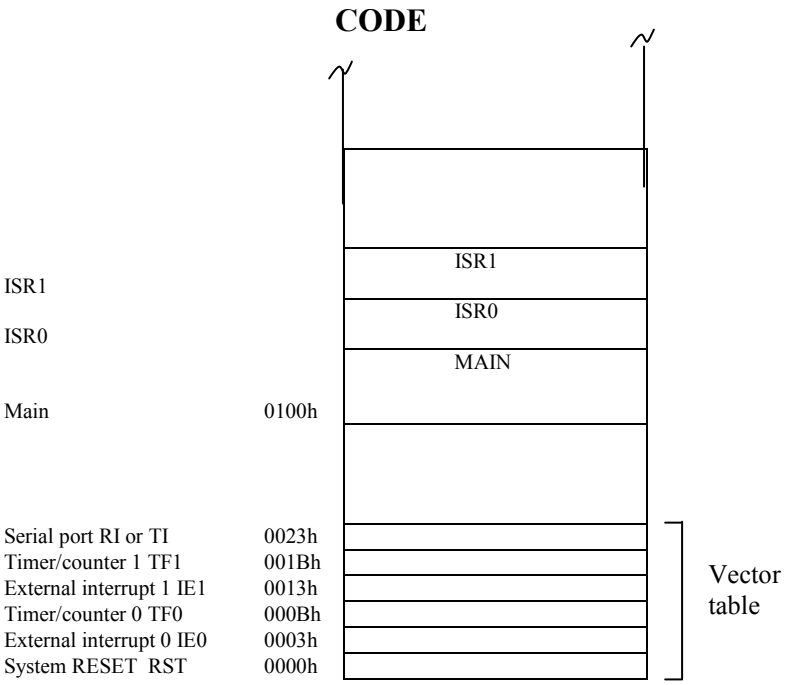


Figure 4.4 Code positioning in code memory space

```
;================================================
; OVEN.A51
; Simple interrupt driven program to control oven temperature. If 200C
; sensor goes low INT1 interrupts causing ISR1 to turn off heater. If
; 190C sensor goes low INT0 interrupts causing ISR0 to turn on heater.
; Port 1, bit0, i.e. P1.0 connects to the heater.
;
;;===============================================================

        ORG 0000h           ; entry address for 8051 RESET
        LJMP MAIN           ; MAIN starts beyond interrupt vector space

        ORG 0003h           ; vector address for interrupt 0
        LJMP ISR0           ; jump to start of ISR0

        ORG 0013h           ; vector address for interrupt 1
        LJMP ISR1           ; jump to start of ISR1

;================================================================
; MAIN enables the interrupts and defines negative trigger operation.
; Heater is turned on and program just loops letting the ISRs do the work.
;================================================================
        ORG 0100h           ; defines where MAIN starts..
MAIN:
        MOV IE, #10000101B  ; enable external interrupts IE0, IE1
        SETB IT0            ; negative edge trigger for interrupt 0
        SETB IT1            ; negative edge trigger for interrupt 1

; Initialise heater ON
        SETB P1.0           ; heater is ON

LOOP:
        LJMP LOOP           ; loop around doing nothing!

;================================================================
; ISR0 simply turns ON the heater
;================================================================
ISR0:
        SETB P1.0           ; turn ON heater
        RETI                ; return from interrupt


;================================================================
; ISR1 simply turns OFF the heater
;================================================================
ISR1:
        CLR P1.0            ; turn OFF heater
        RETI                ; return from interrupt


END                         ; end of program
```

Listing 4.1 Program for interrupt driven oven control

4.3 OTHER SOURCES OF INTERRUPTS

Figure 4.5 shows the set of 8051 interrupt sources. If we follow the external interrupt INT0, for example, we see that this external interrupt connects to the processor at the P3.2 pin. Note Port 3 can be used as a standard input/output port as shown earlier – but various Port 3 pins have alternative functionality. When INT0 is activated (negative edge usually), internally within the 8051 the EX0 request is raised. This flags an interrupt request but the relevant interrupt bit within the IE register must be set, along with the EA bit if this interrupt request is to raise an interrupt flag. The interrupt flag IE0 is then raised and causes the program counter (PC) to vector to vector location 0003h, as discussed earlier. Note, the Timer/Counter interrupt flags can be software polled even if the ETx bits are not enabled. Interrupts can also be software generated by setting the interrupt flags in software. The interrupt flags are accessible as flags on the TCON and SCON registers as follows:

TCON register

TF1 *msb*		**TF0**		**IE1**	IT1	**IE0**	IT0 *lsb*

SCON register

msb						**TI**	**RI** *lsb*

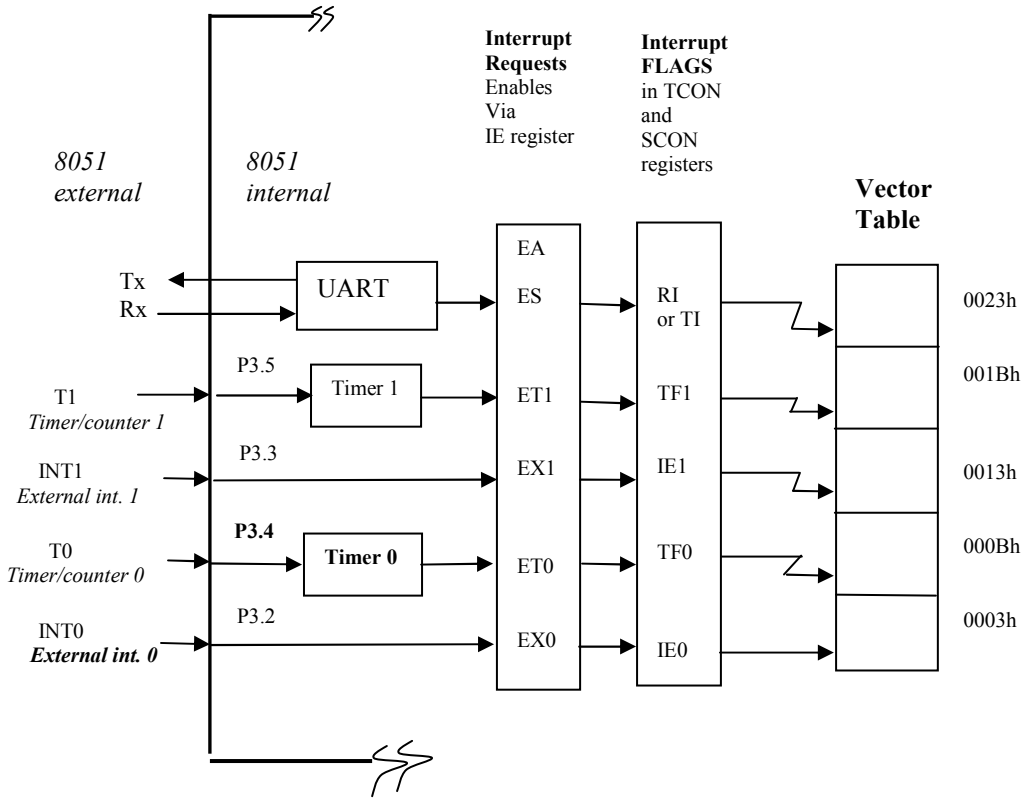


Figure 4.5 Interrupt sources

4.4 INTERRUPT PRIORITY LEVEL STRUCTURE

An individual interrupt source can be assigned one of two priority levels. The Interrupt Priority, IP, register is an SFR register used to program the priority level for each interrupt source. A logic 1 specifies the *high* priority level while a logic 0 specifies the *low* priority level.

IP register

x *msb*	**x**	**PT2**	**PS**	**PT1**	**PX1**	**PT1**	**PX0** *lsb*

IP.7	x	reserved
IP.6	x	reserved
IP.5	PT2	Timer/counter-2 interrupt priority (8052 only, not 8051)
IP.4	PS	Serial port interrupt priority
IP.3	PT1	Timer/Counter-1 interrupt priority
IP.2	PX1	External interrupt-1 priority
IP.1	PT0	Timer/Counter-0 interrupt priority
IP.0	PX0	External interrupt-0 priority

An ISR routine for a high priority interrupt cannot be interrupted. An ISR routine for a low priority interrupt can be interrupted by a high priority interrupt, but not by a low priority interrupt.

If two interrupt requests, at different priority levels, arrive at the same time then the high priority interrupt is serviced first. If two, or more, interrupt requests at the same priority level arrive at the same time then the interrupt to be serviced is selected based on the order shown below. Note, this order is used only to resolve simultaneous requests. Once an interrupt service begins it cannot be interrupted by another interrupt at the same priority level.

Interrupt source	**Priority within a given level**
IE0	*highest*
TF0	
IE1	
TF1	
RI, TI	
TF2 (8052, not 8051)	*lowest*