EE2016 - Microprocessor Theory + Lab

Experiment-6: AVR ATMega8 Interrupt Handling

1 Introduction

In this experiment, we will extend the "blinking LED" of experiment 5 in a few ways. First, we will write a program in C to show how control is transferred to an interrupt service routine (from the main program) when a switch is pressed. We will then "break up" the task of interrupt handling via an assembly language program.

2 How does an interrupt differ from a subroutine call ?

You have already studied how to transfer (control) to a subroutine for performing some task in your "Introduction to programming" course (in the first year). A subroutine, by the way, is a common name for a sequence of program instructions that perform a specific task and a subroutine is written as a separate 'unit'. A subroutine can be "called" multiple times within a program. The processor transfers program execution from the main program to the subroutine when it is called. Once the subroutine execution is completed, there is a transfer of control (back) to the next instruction after the call. Several pieces of information need to be stored before a program (control) switches to a subroutine including the program counter contents as well as some other registers. In this microprocessors course too, we will have transfer of control taking place but commonly via hardware "events". In particular, the press of a key connected to the processor leads to control transfer and this is a special instance of what are called *interrupts* that come from different devices which seek the attention of the processor. The processor provides the requested service by executing an *Interrupt Service Routine (ISR)*. Prior to this, the processor saves the "state" (namely, the contents of the program counter and various registers). When the interrupt service routine is completed, the state of the processor is restored.

3 C program to illustrate the operation of an interrupt

A program in C to transfer control from a white LED (turned on) to a red LED (blinking) upon a key press is given below. In the task (described later), you need to connect two LEDs on a breadboard that also has the Atmega8 microcontroller and resistors.

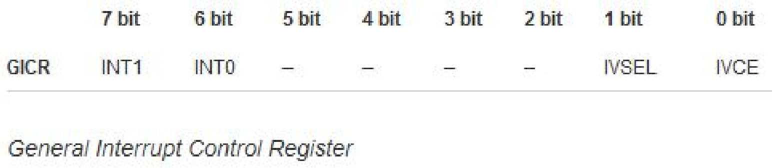
```
#define F_CPU 8000000UL
#include <avr/io.h>
#include<util/delay.h>
#include <avr/interrupt.h>


int main(void)
{
  DDRB=0x03;
  DDRD=0x00;
  GICR=0x40;
  SREG=0x80;
  while(1)
   {
    PORTB=0x01;
   }
}


ISR(INT0_vect)
{

 cli();
 PORTB=0x02;
_delay_ms(100);
 PORTB=0x00;
_delay_ms(100);
 sei();
}
```

The line $\#define\ F\_CPU$ $8000000UL$ sets the microcontroller clock speed (UL denotes unsigned long; we have 8 MHz here). Setting DDRB to 0x03 allows output to a white LED as well as a red LED. Note that both of these are connected to Port B. DDRD is set to 0 since the switch is an input device. The General Interrupt Control Register (GICR) is an 8-bit register with the structure shown in Figure 1. Our interest is in INT0 hence we set "6 bit " (7th from right end) in the figure to 1. The remaining bits are set to 0 (note that IVSEL and IVCE correspond to interrupt vector select and interrupt vector change enable and these are set to 0 to allow interrupts). Setting status register SREG to 0x80 corresponds to *global interrupt enable* as shown in Figure 2.



	7 bit	6 bit	5 bit	4 bit	3 bit	2 bit	1 bit	0 bit
GICR	INT1	INT0	–	–	–	–	IVSEL	IVCE

General Interrupt Control Register

Figure 1: Structure of GICR

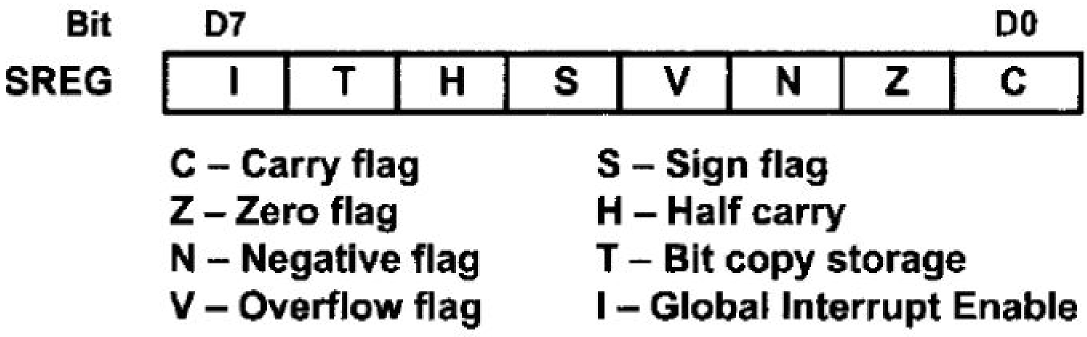


Figure 2: Structure of status register

Within the ISR, we note that CLI clears the global interrupt flag (I) in the status register (SREG) and thereby disables the interrupts. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with CLI. However, while coming out of the ISR, we have an SEI which sets the global interrupt flag (I) to 1.

Note: Unlike function (or subroutine) calls, note that there is no explicit call to the interrupt inside the main function. However, the transfer of control takes place whenever there is a key press.

4 Assembly Language Programming for Interrupt Handling

The program written in C hides some "low-level" details from the user. For instance, setting up of the stack pointer is not explicitly observed in the C program. Note that an interrupt is not treated exactly like an input we want to give (hence PINx register will not be used in the program). However, PORTx and DDRx are of interest as in the previous experiment and appropriate DDRx setting for 'allowing' an interrupt would be important.

5 Tasks for this experiment

Task 1: Wire up the circuit on the breadboard and connect the press button switch, one LED that gives white light and another that gives red light (both via appropriate resistors). Also, type in the C program given and show to the TAs the response (on the breadboard) to key press.

Task 2: Complete the assembly language program below to cause a (red) LED to blink upon receiving an interrupt via a button press. Run the program and demonstrate the (red) LED blinking.

```
.org 0
rjmp reset ; on reset, program starts here


.org 0x002 ;  Interrupt vector address for INT1.
rjmp int1_ISR       ;

reset:

   ldi R16,0x70 ; setup the stack pointer to point to address 0x0070
   out spl,....
   ldi ...,0
   out sph,...

   ldi R16,....     ; make PB1 output
   out .... ,....

   ldi R16,...    ; make PORTD input
   out ...,....

   ldi R16,0x08     ; use pull-up resistor for PD3
   out ...,....


   in R16,....
   ori R16,0x80      ; enable INT1 interrupt
   out .....,R16

   ldi R16,... ;  Turn off LED
   out ...,....

   .... ;  enable interrupts

 indefiniteloop: .............. indefiniteloop

 int1_ISR: ; INT1 interrupt handler or ISR
        .... ; clear interrupts
        in R16,.... ; save status register SREG
        push ....

ldi R16, .... ; blink led 10 times
        mov R0,R16

back5:  ldi R16,02 ; Turn on LED
    out ...,.....

delay1:  LDI R16,0xFF ; delay
back2:  LDI R17,0xFF
back1:   DEC R7
         BRNE back1
         DEC R16
         BRNE back2

 ldi R16,.... ; Turn off LED
 out ....,.....

delay2:  LDI R16,0xFF ;  delay
back3:  LDI R17,0xFF
back4:   DEC R17
         BRNE back4

         DEC R16
         BRNE back3

        DEC ....
BRNE ...... ; ; check if LED has blinked 10 times

pop ..... ; retrieve status register
out ....,......

        ..... ; go back to main program
```

Task 3: Modify the "main loop" of the assembly code of Task 2 to include a white LED (just as we did in the C program). Run the program and demonstrate the white LED glowing.

Task 4: Submit a report on the experiment on Moodle (within a week of this experiment). One report per group (with the names of the group members) is sufficient. The report should contain details of the solution (including the code) and your observations and experience (in programming, debugging etc.). Please note that reports that closely match those of other groups will be penalized.