CSC211
Laboratory B: External Interrupts

Background Information

External Interrupts on the AVR:
Three of the AVR pins can be used by external devices to signal the AVR, requesting an interrupt. Not surprisingly, this is called generating an “external interrupt.” Each of these three interrupt signals has its own name (INT0, INT1, INT2), and each is associated with its own position in the interrupt vector.

There are two registers in the “I/O register” memory space of the AVR that are used to control external interrupts.

- The General Interrupt Flag Register (GICR) contains bits that enable/disable the external interrupts. For example, to enable INT0 we need to set bit 6 in GICR to 1.
- The MCU Control Register (MCUCR) contains bits that configure the external interrupts. Using these, we can specify whether an interrupt should be generated on the rising or falling edge (i.e., when the voltage on the associated pin rises from 0 to 1, or falls from 1 to 0). For example, to specify that INT0 should be triggered on the rising edge, we set bit0 and bit1 in MCUCR to 1.

Interrupt Vector:
The AVR model we are using (Atmega8515L) has 17 possible interrupts. INT0, INT1, and INT2 correspond to positions 2, 3, and 14 of the interrupt vector.

Recall from class that the interrupt vector is written starting at address $000$ of program memory, and that the program will begin execution at address $000$. Therefore, the first instruction should be an rjmp to the starting point of your code. To handle INT0, the second instruction should be an rjmp to an interrupt service routine written to respond to INT0. And so forth. Therefore, to set up your interrupt vector, you will want to use code similar to the following:

```
cseg ; write to program memory
.org 0x000 ; starting at address 0
rjmp start ; jump to program start
rjmp ext_int0 ; jump to ISR for INT0
```

Laboratory Exercises

1. Obtain a copy of the program `/home/davisjan/share/csc211/slowmarch.asm`.
   The goal of this program is to display an recognizable light pattern more complex than a simple on/off, so that you can tell where the main program is resumed after an interrupt.

2. Look at page 2 of the “ATmega 8515L Summary” datasheet that is posted on the course website. Figure 1 shows the pinout of the AVR 8515L chip we are using. Locate pin 12 and note that it is labeled “(INT0) PD2”. This means that the same pin is used as bit2 of Port D, and as the trigger for an INT0 external interrupt. Therefore, we will be able to trigger the interrupt by pressing switch 2 on the STK500 board.

Next look at page 54 of the “ATmega 8515L Datasheet” (not the summary), which is also posted on the course website. Here Table 22 lists all of the available interrupts and their positions within the interrupt vector.

Finally, take a quick look at pages 77-79 of the same Datasheet. These give information about the meaning of each bit in the GICR and MCUCR registers described above. For example, Table 41 at the top of page 78 shows that if both bit0 and bit1 of MCUCR are set to 1, INT0 will be triggered on the rising edge. Similarly, at the top of page 79 you will find information about bit6 of GICR which enables the INT0 interrupt.
3. Modify your program from part 1 above such that it can respond to the external interrupt INT0. This will involve:

- setting up the interrupt vector (as described above),
- setting up the stack,
- enabling and configuring INT0, with code similar to the following:
  
  ldi r16, 0b01000000  
  out GICR, r16 ; enable INTO
  ldi r16, 0b00000011  
  out MCUCR, r16 ; configure INTO (respond to rising edge)
  sei ; enable global interrupts

- writing an interrupt service routine. Your interrupt service routine should cause all of the LEDs to blink on and off 3 or 4 times. Be sure to save and restore the previous contents of any registers that you use, as well as the contents of the status register (SREG), so the ISR can be called safely from anywhere in the program.

To test your program, press and release switch 2 (marked SW2) on the STK500 development board. Releasing the switch should generate an interrupt, causing the LEDs to flash 3 or 4 times. Afterward, your original light pattern should be resumed.

4. Add a second interrupt (INT1) to your program. You will need to enable and configure INT1, add an entry to the interrupt vector, and write a new interrupt service routine. This interrupt should again cause a new, recognizable pattern to be briefly displayed before returning to the original pattern.