Lab 7: Interrupts

Submit timers_interrupt.asm at the end of your lab.

I. What is an interrupt?

An interrupt is a signal that can interrupt and alter the flow of current program execution. It is a method that allows programs to respond immediately to external events. Interrupts can be triggered by external or internal signals; they can be caused by software or hardware. When a program is interrupted, the routine that is executed in response to the interrupt is called an **interrupt service routine (ISR)**, or **interrupt handler**. The process of interrupting the current program, executing the interrupt handler, and returning, is called **servicing the interrupt**. Interrupts can be dedicated or shared. If interrupts are shared among multiple devices (I/O), then the ISR further has to check which device or signal caused that interrupt. Usually the interrupts are numbered and when an interrupt occurs, the processor jumps to a pre-defined location assigned to that interrupt. Hence an interrupt vector table has to be setup to jump and branch to appropriate ISR to handle interrupts. The ATMega 2560 processor we use in the lab has following interrupt vector table:

Table 13-1. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016(3)	PCINT2	Pin Change Interrupt Request 2
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	$0030	SPI, STC	SPI Serial Transfer Complete
26	$0032	USART0 RX	USART0 Rx Complete
27	$0034	USART0 UDRE	USART0 Data Register Empty
28	$0036	USART0 TX	USART0 Tx Complete
29	$0038	ANALOG COMP	Analog Comparator

The AVR processor has many built-in timers (two 8-bit and four 16-bit timer counters). These can be setup in many different modes. We are only going to use Time 1 in this lab. This timer will be set to normal overflow mode. That is the 16-bit timer counter is initialized to a value (say 0) and a hardware clock signal is used to increment the timer counter. When the timer counter overflows, it generates an interrupt 21 (see the above table) causing the processor to jump to a pre-defined location 0x0028. Therefore, first, let's learn the structure of interrupt in AVR assembly language:

```
1   .org 0x0000
2           jmp setup
3
4   .org 0x0028
5           jmp timer1_isr
6
7   setup:
8           nop
9           call timer_init
10  lp:     jmp lp
11
12  timer_init:
13          nop
14          ; enable timer interrupts
15          ldi r16, 0x01
16          sts TIMSK1, r16
17          sei
18          ret
19
20  timer1_isr:
21          push r16
22          push r17
23          push r18
24          lds r16, SREG
25          push r16
26
27          nop
28
29          ; timer interrupt flag is automatically
30          ; cleared when this ISR is executed
31          ; per page 168 ATmega datasheet
32
33          pop r16
34          sts SREG, r16
35          pop r18
36          pop r17
37          pop r16
38          reti
```

Interrupt Vector Table (IVT) for each interrupt

Bit 0 of **TIMSK1** is the Timer/Counter Overflow Interrupt Enable. Set it to 1 to signal the counter changes from 0xFFFF to 0x0. "sei" set the I flag in SREG to 1. Combine both, the Timer/Counter Overflow Interrupt is enabled. Then the corresponding Interrupt Vector is executed.

Interrupt service routine

At power on (RESET), the program control jumps to location 0x0000. This is where you need to initialize the timer for the desired functionality. In the above program, the control jumps to setup which initializes the timer and sets in the desired mode by calling "**`timer_init`**" and then waits in a tight loop. After the initialization, timer counter automatically starts counting with the specified clock rate and causes an interrupt when the counter overflows. This interrupt (21) takes the CPU control to location 0x0028 that further jumps to the ISR (**`timer1_isr`**).

The AVR processor provides many timers and these timers can be set to operate in different modes so that a right timer can be used in an application. The timer of the AVR can be specified to monitor several events. Status flags in the TIMSK (Timer Interrput Mask Register) register show if an event has occurred. Some of the modes the timers can operate are: a) Timer Overflow mode; b) Compare Match and c) Input capture. **For the purpose of this lab, we are going to setup the timer to operate only in the Timer Overflow mode** (also known as normal mode). A timer overflow means that the counter has counted up to its maximum value and is reset to zero in the next timer clock cycle. Below is the summary of registers associated with Timer/Counter 1:

- TIMSK1 is the interrupt enable register. In order to cause the interrupt by the timer, a flag has to be set in the register as well as enable the global interrupt in SREG (using SEI instruction).
- The 16-bit timer counter value is loaded at: TCNT1H:TCNT1L. The counter will start counting up from the initial value loaded here and count up to 0xFFFF. In the next clock pulse, the 16-bit counter will overflow and reset to 0x0000 and causes an interrupt.
- The timer mode (normal counting mode) is set by writing a "zero" to TCCR1A control register.
- To count up, the timer can use an internal clock signal or an external signal. In this lab, the timer is setup to use an internal clock. Note that the CPU is operating at 16Mhz. We will use this clock but reduce the counting speed by setting up a pre-scaler of 256. That is the clock signal is divided by 256 to reduce the counting speed. This is done using the TCCR1B control register.

II. Timer/Counter 1 in AVR ATmega 2560

In this lab, the 16-bit Timer/Counter 1 is used. The full names of the registers are:

.equ TCCR1A = 0x80 Timer/Counter 1 Control Register A
.equ TCCR1B = 0x81 Timer/Counter 1 Control Register B
.equ TCCR1C = 0x82 Timer/Counter 1 Control Register C **(Not used in this lab)**
.equ TCNT1H = 0x85 high byte of the Timer/Counter 1
.equ TCNT1L = 0x84 low byte of the Timer/Counter 1
.equ TIFR1 = 0x36 Timer/Counter 1 Interrupt Flag Register **(Not used in this lab)**
.equ TIMSK1 = 0x6F Timer/Counter 1 Interrupt Mask Register
.equ SREG = 0x5F Status Register

TCCR1A - Timer/Counter 1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1A is set to 0, which means in normal mode, disconnect Pin OC1 from Timer/Counter 1. The other modes are: "COM1A1:COM1A0": Compare Output Mode for Channel A, "WGM11:WGM10": Waveform Generation Mode. Only when one of the OC1A/B/C is connected to the pin, the function of the COM1x1:0 bits is dependent of the WGM13:0 bits setting. **It doesn't apply to our example since the values are all 0's.**

TCCR1B - Timer/Counter 1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 2:0 – CSn2:0: Clock Select**

The three clock select bits select the clock source to be used by the Timer/Counter, see Figure 17-10 and Figure 17-11 on page 152.

Table 17-6. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

If external pin modes are used for the Timer/Countern, transitions on the Tn pin will clock the counter even if the pin is configured as an output. This feature allows software control of the counting.

The least significant three bits of TCCR1B are used to slow down the interval in our example. Instead of counting 1 per clock cycle, we count 1 every 256 clock cycles in the example.

TCCR1C - Timer/Counter 1 Control Register C

Bit	7	6	5	4	3	2	1	0	
(0x82)	FOC1A	FOC1B	FOC1C	–	–	–	–	–	TCCR1C
Read/Write	W	W	W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

Not explicitly initialized in this example.

TIMSK1 – Timer/Counter 1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

It is set to 0x01. "**Bit 0 – TOIEn: Timer/Counter, Overflow Interrupt Enable**
When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter Overflow interrupt is enabled. The corresponding Interrupt Vector is executed by program control jumping to address 0x0028 and further jumping to timer1_isr.

TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	OCF1C	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Not explicitly initialized in this example.

The TCNT1 is a 16-bit register and can be accessed by the AVR CPU via the 8-bit data bus. The 16-bit register must be byte accessed using two read or write operations. Each 16-bit timer has a single 8-bit register for temporary storing of the high byte of the 16-bit access. Accessing the low byte triggers the 16-bit read or write operation. When the low byte of a 16-bit register is written by the CPU, the high byte stored in the Temporary Register, and the low byte written are both copied into the 16-bit register in the same clock cycle. When the low byte of a 16-bit register is read by the CPU, the high byte of the 16-bit register is copied into the Temporary Register in the same clock cycle as the low byte is read.

We are going to write blink.asm program in today's lab, but using a timer interrupt to manipulate the intervals between on and off of the LED lights.

III. Exercises: download timers_interrupt.asm, Finish implementing the timer1_isr interrupt. This exercise flashes the lower two LEDs alternately.

Optional: download blinkOne.asm from lab3, write it as a subroutine in timers_interrupt.asm. Let the top LED light on and off using subroutines, and the bottom two LED lights on and off using interrupts.
Submit timers_interrupt.asm at the end of your lab.

For more information, refer to section 14 (page 101) and section 17 (page 133) of the datasheet of ATMEGA 2560 at http://www.atmel.com/images/doc2549.pdf.