**// Purpose: To demonstrate a MID-level implementation of a Timer1 Normal Mode Overflow Interrupt  
// Result: LED flashes at 60Hz on PB5 (digital pin 13 on the Arduino)  
// Reference:** [**https://www.easycalculation.com/engineering/electrical/avr-timer-calculator.php**](https://www.easycalculation.com/engineering/electrical/avr-timer-calculator.php) **// Author: C. D'Arcy  
// Date: 2017 01 07  
// Status: Working**#define hi 0xFD // preload for TCNT1H (0xFDDD yields 120ovf/s=60Hz)#define lo 0xDD // preload for TCNT1L (0xFD70=50Hz, 0xFCCC=40Hz, 0xFAAA=24Hz)// prescale constants//uint8\_t clkDiv1 = 1 << CS10;//uint8\_t clkDiv8 = 1 << CS11;//uint8\_t clkDiv64 = 1 << CS11 | 1 << CS10;uint8\_t clkDiv256 = 1 << CS12;//uint8\_t clkDiv1024 = 1 << CS12 | 1 << CS10;void setup() {  
 // declare digital pin 13 as output  
 DDRB |= (1 << PB5);  
 // Normal Mode  
 TCCR1A = 0;  
 // set up timer with prescaler  
 TCCR1B = clkDiv256;  
 // initialize counter(+ has higher precedence than <<)  
 TCNT1 = (hi<<8)+lo;  
 // Enable Timer1 interrupt ability  
 TIMSK1 = 1 << TOIE1;  
 // Enable global interrupt ability... sei();}  
void loop(){} // stand down and let ISR respond to interrupts  
// Timer 1 Overflow Interrupt Service Routine  
ISR(TIMER1\_OVF\_vect) {  
 TCNT1 = (hi<<8)+lo; // initialize counter ([+ has higher precedence than <<](http://winavr.scienceprog.com/short-introduction-to-c/c-language-operators-and-expressions.html))  
 PORTB ^= (1 << PB5); // toggle pin 13}

**// Purpose: To demonstrate a LOW-level implementation of a Timer1 Normal Mode Overflow Interrupt  
// Result: LED flashes at 60Hz on PB5 (digital pin 13 on the Arduino)   
// Reference:** [**https://www.easycalculation.com/engineering/electrical/avr-timer-calculator.php**](https://www.easycalculation.com/engineering/electrical/avr-timer-calculator.php) **// Author: C. D'Arcy  
// Date: 2017 01 06  
// Status: Working**

#include "avr/io.h"  
#define hi 0xFD ; preload for TCNT1H (0xFDDD yields 120ovf/s=60Hz)  
#define lo 0xDD ; preload for TCNT1L (green LED)  
  
.global setup  
setup:  
 sbi \_SFR\_IO\_ADDR(DDRB), DDB5 ; pinMode(13,OUTPUT)  
 clr r16 ; prepare for Normal Mode  
 sts TCCR1A, r16 ; set Normal Mode, now configure the prescaler...  
; ldi r16, 0b00000001 ; T1:2^24/2^16 (no prescale) > 2^8 ovf/s > 128Hz  
; ldi r16, 0b00000010 ; T1:2^24/2^3/2^16 (prescale) > 2^5 ovf/s > 16Hz  
; ldi r16, 0b00000011 ; T1:2^24/2^6/2^16 (prescale) > 2^2 ovf/s > 2Hz  
 ldi r16, 0b00000100 ; T1:2^24/2^8/2^16 (prescale) > 1 ovf/s > 0.5Hz  
; ldi r16, 0b00000101 ; T1:2^24/2^10/2^16 (prescale) > 0.25 ovf/s > 0.125Hz  
 sts TCCR1B, r16 ; Timer1 clock = system clock / prescale  
 ldi r16,hi ; load TCNT1 (Timer1's 2-byte counter)  
 sts TCNT1H,r16 ; T1:2^16-(2^24/2^8/120)=64989=0xFDDD->120ovf/s=60Hz  
 ldi r16,lo ; LED flashing at 24Hz does not seem to produce PoV  
 sts TCNT1L,r16 ; even at 60Hz there appears to be some noticeable flicker  
 ldi r16,1<<TOIE1 ; prepare to enable Timer/Counter1 Overflow Interrupt  
 sts TIMSK1,r16 ; enable Timer1 Overflow interrupts  
 sei ; enable global interrupts  
 ret

.global loop  
loop:  
 rjmp loop ; this avoids returning to the C driver

.global TIMER1\_OVF\_vect  
TIMER1\_OVF\_vect: ; destination for TCNT1 overflow interrupt   
; push r16 ; preserve r16 if used externally to this ISR  
; push r17 ; preserve r17 if used externally to this ISR  
; in r16, \_SFR\_IO\_ADDR(SREG) ; preserve the Status Register if necessary  
; push r16 ;  
 ldi r16,hi ; reload TCNT1 (Timer1's 2-byte counter)  
 sts TCNT1H,r16 ; T1:2^16-(2^24/2^8/120)=64989=0xFDDD->120ovf/s=60Hz  
 ldi r16,lo ; LED flashing at 24Hz does not seem to produce PoV  
 sts TCNT1L,r16 ; even at 60Hz there appears to be some noticeable flicker  
 in r16,\_SFR\_IO\_ADDR(PORTB) ; read the port  
 ldi r17,1<<PB5 ; prepare the bit(s) to toggle  
 eor r16,r17 ; toggle the bit(s), leaving the others unaffected  
 out \_SFR\_IO\_ADDR(PORTB),r16 ; update the port  
; pop r16 ; if necessary, restore the data stacked on input to this ISR  
; out \_SFR\_IO\_ADDR(SREG),r16 ; restore Status Register  
; pop r17 ; restore r17 if used externally to this ISR  
; pop r16 ; restore r16 if used externally to this ISR  
 reti ; return from interrupt