Goals:
The Goal of this lab is to generate code that will blink an LED using the internal timer of a microcontroller.

Equipment Usage
For this lab the following equipment will be used:

- LED
- Atmega328P
- AVR Studio
- AVR RISP mkII

Background:

- **Timers in AVR:** AVR microcontrollers generally have access to 2 types of internal timers: an 8 bit timer (timer0) and a 16 bit timer (timer1). Most manipulation of these timers can be placed a set of specific bits. The **Waveform Generation Mode (WGM)** bits dictate the mode of operation for the counter. The mode of operation specifies when the overflow flag is set, what type of waveform is generated and what the peak value of the waveform is. The table below is list of the different modes in the Atmega328P microcontroller.

<table>
<thead>
<tr>
<th>Mode</th>
<th>WGM2</th>
<th>WGM1</th>
<th>WGM0</th>
<th>Timer/Counter Mode of Operation</th>
<th>Top</th>
<th>Update OCRx at</th>
<th>TOV Flag Set on</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Normal</td>
<td>0xFF</td>
<td>Immediate</td>
<td>Max</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>PWM, Phase Correct</td>
<td>0xFF</td>
<td>Top</td>
<td>Bottom</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>CTC</td>
<td>OCR0A</td>
<td>Immediate</td>
<td>Max</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>Fast PWM</td>
<td>0xFF</td>
<td>Top</td>
<td>Max</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>PWM, Phase Correct</td>
<td>OCR0A</td>
<td>Top</td>
<td>Bottom</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Reserved</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Fast PWM</td>
<td>OCR0A</td>
<td>Top</td>
<td>Top</td>
</tr>
</tbody>
</table>
To activate the clock, the clock selector bits must be set. By default the bits will all be zero, disabling the use of the timer/counter. Setting these bits to the appropriate values will activate and configure the timer parameters. Below a table showing the basic values for clock selection:

**Clock Select Bit Description (timer0)**

<table>
<thead>
<tr>
<th>CS02</th>
<th>CS01</th>
<th>CS00</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>No clock source</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(Timer/Counter Stopped)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>clkI/O/(No prescaling)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>clkI/O/8 (From prescaler)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>clkI/O/64 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>clkI/O/256 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>clkI/O/1024 (From prescaler)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>External clock source on T0 pin. Clock on falling edge.</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>External clock source on T0 pin. Clock on rising edge.</td>
</tr>
</tbody>
</table>

**NOTE:** each timer has their own specific registers that apply to it. While the values may be similar the names of the register may be different (i.e. CS12 is used when referencing timer1 instead of CS02.) Please refer to the datasheet for confirmation of the proper register names and descriptions.

**Formulas:**

\[ \text{TCNT} = \left[ \frac{\text{clock\_speed}}{\text{prescaler\_value}} \right] \times \text{desired\_time\_in\_seconds} - 1 \]

**Calculating Timer Counter (TCNT)**

Calculating the timer counter register value will give you the number of cycles needed to reach the end of its cycle (i.e. overflow.) For example, if \(2^n\) is 256 and \(\text{TCNT} = 156\) then the counter will cycle 100 times before overflow occurs.

**Calculating Timer Control Register TCCR**

Calculating TCCR values are dependent on manipulating the WGM bits and the **Clock Select (CS)** bits. By selecting which bits you wish to be high and low you will find the appropriate value to place in your TCCR.
For example, to start the clock with a prescaler of 64 bit CS02 is set to 0 and bits CS01 and CS00 are set to 1. Setting the wave generation mode to normal we set WGM bits to 0. Now look at the register below. Based on the location of the desired bits we would achieve our goal by placing a value of 0x03** into the register (i.e. TCCR0B = 0x03).

**0x03(hexadecimal) = 00000011 (binary) = 3 (decimal)

Prelab:
Design 1: Modify the given pseudo code to produce a square wave of period 2 seconds using Atmega328P (Assume 1 MHZ clock speed). The LED needs to be connected to the PIN PB.5 that toggles every second. You must also select a suitable internal clock.***
hint: some of the OUT commands must be replaced by STS (refer to page 7 of datasheet)

```
.include "m328Pdef.inc"

.MACRO INITSTACK
  LDI R16, HIGH(RAMEND)
  OUT SPH, R16
  LDI R16, LOW(RAMEND)
  OUT SPL, R16
.ENDMACRO

INITSTACK

LDI R16, 32
SBI DDRC, 5 ;PC5 as output
LDI R17, 0   ;needed to toggle led
OUT PORTC, R17
LDI R20, 5   ;to set prescaler
STS TCCR1B, R20 ;Prescaler: 1024

begin:
RCALL delay   ;calling timer to wait for 1 sec
EOR R17, R16  ;XOR to toggle led
OUT PORTC, R17
RJMP begin    ;repeating i.e, while(1)
```

```
delay:
  LDS R29, TCNT1H  ;loading upper bit of counter to R29
  LDS R28, TCNT1L  ;loading lower bit of counter to R28
  CPI R28,0xc6     ;comparing if lower is 0xC6
  BRSH body
  RJMP delay

body:
  CPI R29,0x2D
  BRSH done
  RJMP delay

done:
  LDI R20,0x00
  STS TCNT1H,R20  ;resetting the counter to 0 for next round
  LDI R20,0x00
  STS TCNT1L,R20  ;resetting the counter to 0 for next round
  RET

Show how you calculated the values for TCNT1, TCCR1A, TCCR1B
***REMEMBER: the above code is meant for M32 not Atmega328P. You will have to make
modifications to the code to make it work properly. A useful guide would be to run this code in AVR
studio using the M32 device to better understand how the code operates.

C code:

#include <avr/io.h>

int main(void)
{
  int R16 = 32;
  int R17 = 0;
  DDRC =0xFF;
  TCCR1B |= (1 << CS12) | (1 << CS10);
  while(1)
  {
    if (TCNT1 >= 11718) //0x2dc6
    {
      R17 = R17 ^ R16;
      PORTC = R17;
      TCNT1 = 0;
    }
  }
}
**Lab Experiments:**

**Experiment 1:** Redo the prelab except this use the default frequency of you microcontroller (Assume 8MHz for 328P). Program this code into your microcontroller and demonstrate it to the TA.

**Experiment 2:** Modify your code to do the following:

a) Change the period from 2 seconds to 6 seconds for Timer1
b) Change the code so that when PB5 is toggled off, PB4 is toggled on.

**Post-Lab Deliverables:**

1) Submit your working code along with your calculations
   a. Timer1 code w/6 second period on PB4 and PB5
2) Answer the following Questions
   a. Recalculate the TCNT value for a 6 second period using a 4 MHZ clock for timer1
   b. Given the following code, calculate how long it will take the timer to overflow(assume 8MHZ clock)

**DELAY:**
- LDI R20, 0xA4
- STS TCNT1H, R20
- LDI R20, 0x73
- STS TCNT1L, R20
- LDI R20, 0x00
- STS TCC1C, R20
- LDI R20, 0x05
- STS TCCR1B, R20

   c. Read the datasheet and explain the difference between Normal and CTC modes for the WGM.