1- Consider the SHORTDELAY subroutine that is outlined below. Write AVR assembly so that it utilizes the 16-bit Timer/Counter1 to delay for 201 milliseconds (0.201 seconds). Assume that the system clock frequency is 16 MHz. Note the following requirements:

(a) Timer/Counter1 must be initialized to operate in the Normal mode.

(b) The SHORTDELAY subroutine loads the proper value into TCNT1 and waits until TOV1 is set. Once TOV is set, it is cleared and the SHORTDELAY subroutine returns.

(c) The code must utilize a polling strategy to check TOV1 (do not use interrupt vectors in this homework problem).

$$Delay_{Normal} = \frac{(Max+1-value)*prescale}{clk_{I/O}}$$

$$value = 65535 - \frac{201ms}{prescale*62.5ns} = 65535 - \frac{201ms}{64*62.5ns} = 15285 = 0x3BB5$$

```
.include "m128def.inc"

.def mpr = r16

...

.ORG $0000

        RJMP Initialize

.ORG $0046                          ; End of interrupt vectors

Initialize:
        …
        ; initialize stack
        LDI         mpr, high(RAMEND)
        OUT         SPH, mpr
        LDI         mpr, low(RAMEND)
        OUT         SPL, mpr

        ; initialize TCNT1
        SBI         DDRB, PB7           ; set bit 7 of Port B for output
        LDI         mpr, 0b00000000     ; activate Normal mode, OC1A, OC1B, and OC1C
                                                disconnected
        OUT         TCCR1A, mpr
        LDI         mpr, 0b00000001     ; set prescale to 64
        OUT         TCCR1B, mpr
        LDI         mpr, 0b00000000
        OUT         TCCR1C, mpr
        ...
```

SHORTDELAY:

```
    …
        ; load value into TCNT1
        LDI     mpr, 0x3B           ; load 0x3B to mpr
        OUT     TCNT1H, mpr         ; write to high byte first
        LDI     mpr, 0xB5           ; load 0xB5 to mpr
        OUT     TCNT1L, mpr         ; write to low byte second
CHECK:
        IN      mpr, TIFR           ; read in TIFR
        ANDI    mpr, 0b00000100     ; check if TOV1 is set
        BREQ    CHECK               ; loop if TOV1 is not set
        LDI     mpr 0b00000100      ; otherwise, reset TOV1
        OUT     TIFR, mpr           ; note - write 1 to reset
    ...
    RET
```

2- Imagine that you want to program your lab board to handle the I/O configuration illustrated in the image below. Review the starter code that is provided below and fill in the missing lines (based on the instructions) to accomplish the given tasks. mpr is the only general purpose register that you are allowed to use in the code.

(a) Fill in lines 1-2 so that the pin directions are properly configured to control the engine enable and engine direction for both left and right wheels. Any unused pins must be configured as inputs

```
(1)     LDI     mpr, (1<<0) | (1<<1) | (1<<2) | (1<<3); DDRA = 00001111 for output
(2)     OUT     DDRA, mpr                               ; set DDR register for Port A
```

(b) Fill in lines 3-4 so that the pin directions are properly configured to detect left and right bumper movements. Once again, any unused pins must be configured as inputs.

```
(3)     LDI     mpr, (0<<4) | (0<<5)                    ; DDRE = 00000000 for input
(4)     OUT     DDRE, mpr                               ; set DDRE register for Port E
```

(c) What are the addresses needed in lines (i) and (ii) to properly control the execution of interrupt service routines for left and right bumpers? Fill in lines 5-6 to enable the pull-up resisters for these whiskers.

```
(i)     $000A           ; INT4 is located at address $000A

(ii)    $000C           ; INT5 is located at  address $000C
```

```
(5)   LDI     mpr, (1<<4) | (1<<5)
(6)   OUT     PORTE, mpr              ; activate pull-up resistors
```

(d) Fill in lines 7-8 with the necessary code to set External Input Sense Control to detect bumper hits (i.e., interrupts) on a falling edge.

```
(7)   LDI     mpr, (1<<ISC01) | (0<<ISC00) | (1<<ISC11) | (0<<ISC10)
(8)   STS     EICRA, mpr              ; EICRA = 00001010 => detect on falling edge
```

(e) Fill in lines 9-10 to unmask the interrupts for whisker movements.

```
(9)   LDI     mpr, (1<<INT4) | (1<<INT5)   ; turn on interrupts for L/R bumpers
(10)  OUT     EIMSK, mpr                   ; EIMSK = 00110000
```

3- Consider the AVR code segment shown below (with some missing information) that configures Timer/Counter0 for Fast PWM operation, and modifies the Fast PWM duty cycle whenever a specific button on Port D is pressed.

01101010

(a) Fill in lines (1-2) with the instructions necessary to configure Timer/Counter0 for Fast PWM mode, non-inverting output, and a prescale value of 8.

```
(1)   LDI     mpr, 0b01101010         ; activate Fast PWM with toggle
(2)   OUT     TCCR0, mpr              ; (non-inverting) and set prescale to 8
```

(b) Based on the prescale value used in part (a), what is the frequency of the PWM signal (fPWM) being generated by Timer/Counter0? Assume the system clock frequency is 16 MHz.

$$f_{PWM} = \frac{clk_{I/O}}{prescale*256} = \frac{16MHz}{8*256} = 0.0078125\ MHz$$

(c) Fill in lines (3-4) to provide the compare value for Timer/Counter0 so that the initial duty cycle is 51% (use the closest available settings).

```
(4)   LDI     mpr, 131                ; set compare value
(5)   OUT     OCR0, mpr               ; (131/256)*0.51 = 131
```

(d) What would be the value necessary for the variable step to increase the duty cycle by 12.5% each time the DUTY_STEP subroutine is executed? Ignore the case when/if the compare value overflows.

$\frac{x}{256} = 0.125 \Rightarrow x = 256 * 0.125 = 32$