1. (3 pts each) Complete the following table by writing in values (or updating values) for variables changed by the expression in the “Operation” column. Assume all variables have datatype uint8_t. Perform each operation independent from the others; that is do not carry the results of one operation into the next row of the table. If a value is updated, then strikethrough the value and write-in a new value. If a cell does not change value, leave the cell unchanged.

<table>
<thead>
<tr>
<th>Operation</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>z = x + y;</td>
<td>100</td>
<td>20</td>
<td>120</td>
</tr>
<tr>
<td>z += x;</td>
<td>100</td>
<td>20</td>
<td>105</td>
</tr>
<tr>
<td>z = x++;</td>
<td>101</td>
<td>20</td>
<td>100</td>
</tr>
<tr>
<td>z = 3*x + y;</td>
<td>100</td>
<td>20</td>
<td>320 = 64</td>
</tr>
</tbody>
</table>

Values and answers in binary (space put between every four bits for readability)

<table>
<thead>
<tr>
<th>Operation</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>z = x &amp; y;</td>
<td>0b1010 0101</td>
<td>0b1100 1100</td>
<td>0b1000 0100</td>
</tr>
<tr>
<td>z = x ^ 0x0F;</td>
<td>0b1010 0101</td>
<td>0b1100 1100</td>
<td>0b1010 1010</td>
</tr>
<tr>
<td>z = x &lt;&lt; 1;</td>
<td>0b1010 0101</td>
<td>0b1100 1100</td>
<td>0b1010 1010</td>
</tr>
<tr>
<td>z = x &amp;&amp; y;</td>
<td>0b1010 0101</td>
<td>0b0101 1010</td>
<td>0b0000 0001</td>
</tr>
</tbody>
</table>

2. (10 pts) Execute the following program to determine the value of **sum** when the for-loop exits.

```c
uint8_t i;
uint8_t sum = 0;
for(i=5; i<12; i+=2) {
    sum += i;
}
```

```
<table>
<thead>
<tr>
<th>i</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>32</td>
</tr>
</tbody>
</table>
```

3. (15 pts) Write a if/then structure which assigns **z** a value based on the value of **x**, a uint8_t type. A filled/closed circle means to include that value of **x** and an open circle means to not include that value of **x**.

```c
if (x <= 8) z=2;
else if (x <= 12) z=1;
else z=0;
```

4. (10 pts) Determine the values in the array after the following snippet executes.

```c
uint8_t i;
uint8_t array[10] = {0,1,2,3,4,5,6,7,8,9};
for (i=0; i<9; i++) {
    array[i+1] = array[i]
} // end for
```

```
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
5. (10 pts) Consider the following code over a 1 second period during which there was a single 30 ms logic low pulse on RA2. Consider the number of times each line of code is executed.

```
for (;;) {
    1. while(PORTAbits.RA2 == 1);
    2. start = TMR0;
    3. while(PORTAbits.RA2 == 0);
    4. end = TMR0;
    5. duration = end - start;
} // end infinite loop
```

Below “Most executed” put the line number of the line of code that is executed most. Continue ranking the lines of code from most to least executed. If lines are tied, the order is unimportant.

<table>
<thead>
<tr>
<th>Most executed</th>
<th>Least executed</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>

Use the following C-code snippet to answer questions

```
1. TOCONbits.PSA = 0;
2. TOCONbits.T0PS = <question #7>;
3. TOCONbits.TMROI = 1;
4. TMR0_WriteTimer(0x10000 – delayValue);  // Note format!!!!
5. while(INTCONbits.TMR0IF == 0);
```

6. (10 pts) What prescaler value (in binary) for line #2 will create the most accurate 10 ms delay on line 5 when run on the PIC as normally configured? State your answer in binary.

```
A 1:4 prescaler is the smallest prescaler with a maximum period over 10 ms. It has a maximum period of 16.4ms. The binary code for this prescaler is 0b001;
```

7. (10 pts) What delay, in milliseconds, is generated in line 5 with TOCONbits.T0PS = 0b011; and delayValue=20,000? State your answer using dimensional analysis.

```
1 second    1000 ms   16 clk
------------ * -------- * ------- * 20,000 counts = 20 ms
16*10^6 clks 1 second 1 count
```

8. (10 pts) Timer 1 is configured with a 1:8 prescaler, starts at 50,000. What is the count value after 10ms? State your answer using 1 or more dimensional analysis calculations. Show all conversions.

```
16*10^6 clks 1 second 1 count
------------ * -------- * ------- * 10 ms = 20,000
1 second 1000 ms 8 clk

Final count: 20,000 - (2^16 - 50,000) = 4,464
```
9. (10 pts.) RC1 and RC2 are both configured in PWM mode to control the brightness of the TOP and BOTTOM LEDs shown in the schematic. Both CCP subsystems are using the same TMR so the period of both PWM waveforms are the same and the positive (rising) edges of both PWM waveforms occur at the same time. Let the duty cycle of RC1 be $A$ and the duty cycle of RC2 be $B$. Duty cycles are measured in TMR counts.

a. What is the inequality relationship between $A$ and $B$ to illuminate the TOP LED?
b. With this inequality, how long (over one PWM period) will the TOP LED be illuminated?

Duty cycle of $A >$ Duty cycle of $B$
The BOTTOM LED is illuminated for $A-B$ TMR counts per PWM period

10. (30 pts) TMR0 ISR has been configured with a 1:8 prescaler to interrupt once every 10 millisecond. Write an ISR to detects 3 press/release of BUTTON_PIN, and then set the global threePressRelease to true after the 3rd release. The button is nominally logic 0 and goes to logic 1 when pressed. Use BUTTON_PIN_GetValue() in your code.

```c
// define globals here
uint8_t threePressRelease = false;

void myTMR0ISR(void) {
    static uint8_t prevButton = 0;
    static uint8_t count = 0;

    if ((BUTTON_PIN_GetValue() == 0) & (prevButton == 1)) count += 1;
    prevButton = BUTTON_PIN_GetValue();

    if (count == 3) threePressRelease = true;

    TMR0_WriteTimer(0x10000 - 20000); // Set TMR0 value for next interrupt3.
    T0CONbits.TMR0ON = 1; // Clear TMR0 flag
}
```

// Show dimensional for TMR0 counts here
//
//    16*10^6 clks   1 second   1 count
//    ------------ * -------- * ------- * 10 ms =
//    1 second       1000 ms     64 clk
//```
11.0 TIMERS MODULE

The Timer module comprises the following features:
- Software selectable operation as a timer or counter in both 8-bit or 16-bit modes.
- Readable and writeable registers.
- Dedicated bit, software programmable prescaler.
- Selectable clock source (external or internal).
- Overflow interrupt.

11.1 Register Definitions: Timer Control

**REGISTER 11-1: TSCON: TIMER CONTROL REGISTER**

- Bit 7: TRM: Timer/Counter Enable bit.
  - 0: Timer/Counter off.
  - 1: Timer/Counter on.
- Bit 0: TRC: Timer/Counter Run Control bit.
  - 0: Stop Timer/Counter.
  - 1: Run Timer/Counter.

**REGISTER 12-0: CCPCON: ENHANCED CCP CONTROL REGISTER**

- Bit 7: CCP3M1: Enhanced PWM Output Configuration bits.
  - 0: Complete PWM mode.
  - 1: Enhanced PWM mode.
- Bit 6: CCP3M0: Enhanced PWM Output Configuration bits.
  - 0: Single output, PWM disabled, PWM assigned as port pin.
  - 1: Single output, PWM enabled, PWM assigned as port pin.
- Bit 5: CCP2M1: Enhanced PWM Output Configuration bits.
  - 0: Single output, PWM disabled, PWM assigned as port pin.
  - 1: Single output, PWM enabled, PWM assigned as port pin.
- Bit 4: CCP2M0: Enhanced PWM Output Configuration bits.
  - 0: Single output, PWM disabled, PWM assigned as port pin.
  - 1: Single output, PWM enabled, PWM assigned as port pin.
- Bit 3: PWL: PWM Logic Level selection bit.
  - 0: PWM logic level.
  - 1: PWM logic level.
- Bit 2: PWL: PWM Logic Level selection bit.
  - 0: PWM logic level.
  - 1: PWM logic level.
- Bit 1: PWL: PWM Logic Level selection bit.
  - 0: PWM logic level.
  - 1: PWM logic level.
- Bit 0: PWL: PWM Logic Level selection bit.
  - 0: PWM logic level.
  - 1: PWM logic level.

Note: See Table 11-1 to determine full-bridge and half-bridge ECCPs for the device being used.