1. (5 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 an empty cell does not change value, leave the cell empty.

<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>120</td>
<td>80</td>
<td>40</td>
</tr>
<tr>
<td>(z += x;)</td>
<td>100</td>
<td></td>
<td>200 - 300 - 256 = 44</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Values and answers in decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>(z = x - y;)</td>
</tr>
<tr>
<td>(z += x;)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Values and answers in binary (space put between every four bits for readability)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(z = x \ll 2;)</td>
</tr>
<tr>
<td>(z = \neg x;)</td>
</tr>
</tbody>
</table>

2. (5 pts) Write a single line of C-code that toggles every bit except bit 3 of an 8-bit variable \(x\). Store the result in an 8-bit variable \(z\). Index bits starting at 0.

```c
z = x ^ 0b1111 0111;
```

3. (5 pts) Write a single line of C-code that sets bit 5 and clears bit 3 of an 8-bit variable \(x\) and leaves all the other bits the same. Store the result in an 8-bit variable \(z\). Index bits starting at 0.

```c
z = (x | 0b0010 0000) & 0b1111 0111;
```

4. (10 pts each) Given the following C-code, find the value of \(a\) after executing the code with the value of \(x, y, z\) given in each row of the table.

```c
if ((x < 30) && (y >= 20) && (z > 40)) {
  a = 0;
} else if (((x <= 25) && ((y != 30) || (z == 20))) {
  a = 1;
} else if ((x > 15) && (y != 15)) {
  a = 2;
} else {
  a = 3;
} // end if
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>10</td>
<td>50</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>30</td>
<td>30</td>
<td>2</td>
</tr>
</tbody>
</table>

5. (10 pts) Write a single line of C-code that waits (does nothing) while an active-low push button on RA3 is not pressed. Use the correct register/field syntax.

```c
while(PORTAbits.RA3 == 1);
```
6. (10 pts) Write the function prototype for `minimum`, a function that takes as input a 1-D array of `uint16_t` called `array` and the length of the array `length`. The array has less than 100 elements. The function returns the smallest element in the array. Use the correct sized data types.

```c
Uint16_t minimum(uint16_t array[], uint8_t length);
```

7. (10 pts) Write a valid line of C-code that illuminates the LED at right using register and bit-field naming convention for our PIC.

```c
LATCbits.LATC2 = 0;
```

8. (10 pts) Calculate the resistance value, R, that will provide the LED with 10mA when \( V_f \) for the LED = 2.3v. Show your work for full credit.

\[
V = IR \quad (3.3V - 2.3V) = 0.01 \times R \quad R = 100 \text{ ohms}
\]

Use the following C-code snippet to answer questions 9-11.

1. T0CONbits.PSA = 0;
2. T0CONbits.T0PS = <question #9>;
3. T0CONbits.TMR0ON = 1;
4. TMR0 = 0xFFFF - delayValue;
5. while(INTCONbits.TMR0IF == 0);

9. (10 pts) What prescaler value (in binary) for line #2 will create the most accurate 25 ms delay on line 5 when run on the PIC as normally configured? State your answer in binary.

A 1:8 prescaler is the smallest prescaler with a maximum period over 25 ms. It has a maximum period of 32.8 ms. The binary code is 0b010;

10. (10 pts) With a 1:64 prescaler what `delayValue` generates a 100 ms delay in line 5? State your answer using dimensional analysis.

<table>
<thead>
<tr>
<th>1 second</th>
<th>1000 ms</th>
<th>64 clk</th>
</tr>
</thead>
<tbody>
<tr>
<td>16*10^6 clks</td>
<td>1 second</td>
<td>1 count</td>
</tr>
</tbody>
</table>

\[
1 \text{ second} \times 1000 \text{ ms} \times 64 \text{ clk} = 100 \times 64 = 6400 \times 16^6 \\
(16^6) \times 1 \text{ count} = 1 \text{ second} = 1 \text{ count}
\]

11. (10 pts) What delay, in milliseconds, is generated in line 5 with `T0CONbits.T0PS = 0b101;` and `delayValue=20,000`? State your answer using dimensional analysis.

<table>
<thead>
<tr>
<th>1 second</th>
<th>1000 ms</th>
<th>64 clk</th>
</tr>
</thead>
<tbody>
<tr>
<td>16*10^6 clks</td>
<td>1 second</td>
<td>1 count</td>
</tr>
</tbody>
</table>

\[
1 \text{ second} \times 1000 \text{ ms} \times 64 \text{ clk} = 20,000 \times 64 = 1280000 \times 16^6 \\
(16^6) \times 20,000 \text{ counts} = 80 \text{ ms}
\]
12. You are building a device that measures the speed of individual alpha particles using a detector loop attached to RC2. The detector loop outputs logic-1 while the alpha particle is inside the detector loop. The detector loop is 5 cm wide. Thus, the duration of the logic-1 pulse on RC2 is equal to the amount of time that it takes the alpha particle to travel 5 cm. Alpha particles move fast, so the timer prescaler is initialized to 1:1.

a) (10 pts) Use dimensional analysis to convert the pulse duration (measured in timer counts) on RC2 into the velocity of the alpha particle in meters per second. You can assume that the pulse duration is stored in a variable called x.

\[
\begin{align*}
16 \times 10^6 \text{ clks} & \quad 1 \text{ count} \quad 1 \text{ flight} \quad 5 \text{ cm} \quad 1 \text{ m} \\
-------------------- & \quad * \quad * \quad * \quad * \quad * \quad * \quad * \quad * \quad = 800,000/x \quad \text{m/s} \\
1 \text{ second} & \quad 1 \text{ clk} \quad x \text{ counts} \quad 1 \text{ flight} \quad 100 \text{ cm}
\end{align*}
\]

b) (40 pts) Write a C-code to measure the pulse width (in timer counts) using the capture subsystem inside the infinite loop and store the pulse width in the variable x. Assume that the timer and compare subsystems are initialized for you and associated with the CCP module. Declare any additional variables at the top of your program.

```c
uint16_t x;
uint16_t start, end;

for (;;) {
    CCP1CONbits.CCP1M = 0b0101;
    PIR1bits.CCP1IF = 0;
    while(PIR1bits.CCP1IF == 0);
    start = CCP1;

    CCP1CONbits.CCP1M = 0b0100;
    PIR1bits.CCP1IF = 0;
    while(PIR1bits.CCP1IF == 0);
    end = CCP1;

    x = end - start;
}
```

} // end infinite loop