1. (1 pt 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.

<table>
<thead>
<tr>
<th>Operation</th>
<th>x</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>z += x;</td>
<td>10</td>
<td>30</td>
<td>60 70</td>
</tr>
<tr>
<td>z = 3*(x+y);</td>
<td>10</td>
<td>30</td>
<td>60 120</td>
</tr>
<tr>
<td>z = x*y;</td>
<td>10</td>
<td>30</td>
<td>60 300-256 = 44</td>
</tr>
</tbody>
</table>

2. (6 pts) Complete the following code snippet that counts the number of times that an active-low push button on RA3 is pressed. Use the correct register/field syntax. RA3 has already been configured as an input.

```c
uint8_t numButtonPresses = 0;
for(;;) {
    while(PORTAbits.RA3 == 1); // Put some line(s) of code here
    while(PORTAbits.RA3 == 0);
    numButtonPresses = numButtonPresses + 1;
}
```

3. (1 pt each) Given the following C-code, find the value of a for each value of x, y, z given in the following table.

```c
if ((x>20) && (y<20) && (z=='0')) {
    a = 0;
} else if ((x>20) || (y<20) || (z=='5')) {
    a = 1;
} else if ((x>20) || (y<5)) && (z!='5') ) {
    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>10</td>
<td>30</td>
<td>‘1’</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>‘1’</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>10</td>
<td>‘0’</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>30</td>
<td>‘5’</td>
<td>1</td>
</tr>
</tbody>
</table>
4. (5 pts) Complete the following code-snippet that moves all the values of array to an index one less. For example, move the value in array[6] into array[5]. Move 0 into the value at array index 9.

```c
uint8_t i=0;
uint8_t array[10];

for (i=0; i<9; i++) {
    array[i] = array[i+1];
} // end for
array[9] = 0;
```

5. (4 pts) Write a function prototype that takes as input a 1-D array of uint16_t elements and the length of the array (less than 100) and returns the smallest element in the array. Make up your own argument names.

```c
uint16_t minimum(uint16 array[], uint8_t length);
```

Use the following C-code snippet to answer questions 6-8.

1. TOCONbits.T0PS = <question #6>;
2. TMR0 = 0x10000 - delayValue;
3. INTCONbits.TMR0IF = 0;
4. while(INTCONbits.TMR0IF == 0);

6. (4 pts) What prescaler (in binary) value will create the most accurate 100 ms delay (on line 4) when run on the PIC as normally configured?

A table lookup in lecture 10 reveals that 1:32 prescaler is the smallest prescaler

```
TOCONbits.T0PS = 0b100;
```

7. (6 pts) With a 1:16 prescaler (on line 1) what delayValue generates a 25 ms delay (on line 4)? Formulate your answer using dimensional analysis.

```
16*10^6 clks 1 second 1 count
------------  *  ---------  *  ---------------  *  25 ms = 25,000
1 second 10^3 ms 16 clk
```

8. (6 pts) What delay, in milliseconds, is generated (on line 4) with a 1:32 prescaler and delayValue=20,000? Formulate your answer using dimensional analysis.

```
16*10^6 clks 1 second 1 count
------------  *  ---------  *  -----------  *  20,000 counts = 40 ms
16*10^6 clks 1 second 1 count
```

9. (6 pts) Timer 1, configured with a 1:8 prescaler, has been counting up for 2.7 seconds. How many (integer) times has Timer 1 rolled over (when started at 0)? Formulate your using dimensional analysis.

```
16*10^6 clks 1 count rollover
------------  *  ---------  *  1 count  *  2.7 seconds = 82.4
1 second  8 clk  2^16 counts
82 times
```
10. 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) (4 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. Call the pulse duration “x” in your dimensional analysis.

\[
\begin{array}{cccccc}
16\times10^6 \text{ clks} & 1 \text{ count} & 1 \text{ flight} & 5 \text{ cm} & 1 \text{ m} \\
\hline
1 \text{ second} & 1 \text{ clk} & x \text{ counts} & 1 \text{ flight} & 100 \text{ cm} \\
\end{array}
\]

\[
\frac{16\times10^6 \text{ clks}}{1 \text{ second}} \times \frac{1 \text{ count}}{1 \text{ clk}} \times \frac{1 \text{ flight}}{5 \text{ cm}} \times \frac{1 \text{ m}}{100 \text{ cm}} = 800,000/x \text{ m/s}
\]

b) (12 pts) Write a C-code snippet to measure the pulse width on RC2 (in timer counts) using the capture subsystem inside an infinite loop. Store the pulse width in pulseDuration at the bottom of the loop. The timer and capture subsystems are initialized. Declare any additional variables at the top of your program.

```c
uint16_t pulseDuration;
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
11.0 TIMERS MODULE

The Timer0 module incorporates the following features:
- Software selectable operation as a timer or counter in both 8-bit or 16-bit mode.
- Readable and writable registers.
- Dedicated interrupt flag, software-programmable prescaler.
- Selectable clock source (internal or external).
- Edge select for external clock.
- Interrupt-on-overflow.

11.1 Register Definitions: Timer0 Control

REGISTER 11-1: TCCR0: TIM0 CONTROL REGISTER

Legend:
\( R = \) Readable bit
\( W = \) Writable bit
\( U = \) Unimplemented bit, read as '0'
\( = \) Value at POR
\( x = \) Bit is reserved

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>T0MD0</td>
</tr>
<tr>
<td>6</td>
<td>T0MD1</td>
</tr>
<tr>
<td>5</td>
<td>T0MD2</td>
</tr>
<tr>
<td>4</td>
<td>T0MD3</td>
</tr>
<tr>
<td>3</td>
<td>T0CS0</td>
</tr>
<tr>
<td>2</td>
<td>T0CS1</td>
</tr>
<tr>
<td>1</td>
<td>T0CS2</td>
</tr>
<tr>
<td>0</td>
<td>T0CS3</td>
</tr>
<tr>
<td>7</td>
<td>TOCS0</td>
</tr>
<tr>
<td>6</td>
<td>TOCS1</td>
</tr>
<tr>
<td>5</td>
<td>TOCS2</td>
</tr>
<tr>
<td>4</td>
<td>TOCS3</td>
</tr>
</tbody>
</table>

Note: User software should ensure the appropriate interrupt flag bits are set or cleared prior to enabling an interrupt and after servicing that interrupt.

---

REGISTER 14-2: CCP0CON: ENHANCED CCP CONTROL REGISTER

Legend:
\( R = \) Readable bit
\( W = \) Writable bit
\( U = \) Unimplemented bit, read as '0'
\( = \) Value at POR
\( x = \) Bit is reserved

<table>
<thead>
<tr>
<th>Bit</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>PM0</td>
</tr>
<tr>
<td>6</td>
<td>PM0</td>
</tr>
<tr>
<td>5</td>
<td>PM0</td>
</tr>
<tr>
<td>4</td>
<td>PM0</td>
</tr>
<tr>
<td>3</td>
<td>PM0</td>
</tr>
<tr>
<td>2</td>
<td>PM0</td>
</tr>
<tr>
<td>1</td>
<td>PM0</td>
</tr>
<tr>
<td>0</td>
<td>PM0</td>
</tr>
</tbody>
</table>

Note: See Table 14-1 to determine full-bridge and half-bridge EOC0s for the device being used.