You may use a non-programming calculator only. You may use only the provided reference materials.

Recall that the ‘d’ bit in a machine word indicating a destination is ‘0’ if the destination is W, is ‘1’ if the destination is the file register. For the ‘a’ bit, use the assumptions we have used in class (‘a’ bit is ‘0’ if address in ACCESS RAM, ‘a’ bit is ‘1’ if not in ACCESS RAM). Addresses 0x000 – 0x07F and 0xF80-0xFFF, which lie in access RAM, are automatically assigned an a = 0 by the assembler, while all other addresses lie in banked memory and are therefore assigned a = 1 by the assembler.

All instructions which require a ‘d’ bit MUST end with a “, w” or “, f”. Unlike the assembler, no default destination will be assumed and this portion of your solution will be marked as incorrect.

Unless stated otherwise, all multi-byte data values are stored in little-endian ordering.

Please note the relative value of each problem in the table below.

Answers should be clearly indicated. Placing them in a BOX would be ideal.

Be as neat and well organized as possible. This is in your grade’s best interest.

If you need additional space to work, do so on the backside of the page. Make sure it is clear where your work continues.

Absolutely NO cheating is allowed. If you are caught in the attempt of, the act of, or the past action of academic dishonesty, you will receive the maximum punishment allowed by University policy.

No panicking allowed!

<table>
<thead>
<tr>
<th>Page</th>
<th>Maximum</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Part I: (84 pts)

a. (4 pts) Write C code that configures PORTB for use with an LED on RB7 and RB5 and a pushbutton switch on RB3 and RB1. The internal weak pullup must be enabled. Do not assume any default bit values.

\[
\text{RBPU} = 0; \\
\text{TRISB} = 0x0A; \quad // \text{since 1 = Input, 0 = Output,} \\
\text{// the pattern is 0x0x 1x1x.}
\]

b. (6 pts) How many bytes can be sent in 5 seconds assuming a baud rate of 9600, and an asynchronous data format of 1 start bit, 8 data bits, and 1 stop bit assuming the bytes are sent as fast as possible?

\[
5 \text{ seconds} \times 9600 \text{ bits/second} / (1 + 8 + 1) \text{ bits/byte} = 4800 \text{ bytes.}
\]

c. (6 pts) Write C code that implements the \texttt{char getch(void)} function (read one character from the serial port). No interrupts are enabled.

\[
\text{char getch (void)}
\]
\[
\text{while (!RCIF)};
\]  
\[
\text{return RCREG;}
\]

d. (8 pts) Explain what happens in the code below by specifying what appears on the console after the PIC is powered on, and justify your answer by explaining the sequence of events. You must clearly specify if output continually appears on the console or if at some point it stops.

main() {
    char c;
    serial_init(95,1);
    if (!POR) {
        POR = 1;
        SWDTEN = 1;
        printf(“Hello!”);pcrlf();
    } else
        SWDTEN = 0;
    if(SWDTEN) {
        asm(“sleep”);
    }
    while (1) {
        printf(“Looping”);pcrlf();
    } //end main()
}

Answer:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a.</td>
<td>PIC is powered on.</td>
</tr>
<tr>
<td>b.</td>
<td>POR is bit is ‘0’, so ‘Hello’ printed and the watchdog timer is enabled.</td>
</tr>
<tr>
<td>c.</td>
<td>The WDT enabled, so the PIC goes to sleep.</td>
</tr>
<tr>
<td>d.</td>
<td>The PIC wakes up when the WDT expires and continues executing code.</td>
</tr>
<tr>
<td>e.</td>
<td>The loop is entered and ‘Looping’ is printed many times to the console. At some point, the WDT expires, and the PIC is reset.</td>
</tr>
<tr>
<td>f.</td>
<td>The POR bit is not set this time, so the SWDTEN bit is set to zero; ‘Looping’ is printed until the PIC is power cycled.</td>
</tr>
</tbody>
</table>
e. (8 points) Implement the `intIncCopy()` function given below. Assume FSR0 already contains the pointer value for `int *dest` on function entry but that the pointer value for `int *src` is passed in the CBLOCK. In the subroutine, use FSR1 to implement the pointer operations for `int *src`. Note that dest and src are ints.

```c
void intIncCopy(unsigned int* dest, unsigned int* src, unsigned char length)
{
    while (length)
    {
        *dest = *src + 1;
        dest++;
        src++;
        length--;
    }
}
```

```asm
; Parameter block for the intIntCopy function
CBLOCK 0x040
    length, src:2;  Space for parameters
ENDC

CBLOCK 0x000  ;space for main
    array1:10, array2:10
ENDC

```

f. (8 points) Implement the main() code below in PIC assembly. Pass the value for “`int *src`” directly in FSR0; pass the value for “`int *src`” and “char length” using the CBLOCK space for “intIncCopy”.

```asm
void intIncCopy(unsigned int* dest, unsigned int* src, unsigned char length)
{
    // some code
}

main()
{
    int array1[5], array2[5];

    // Code to initialize array2
    intIncCopy(array1, array2, 5);
}
```
For the following instructions, assume the memory contents BEFORE THE EXECUTION OF EACH CODE SNIPPET is given in the box below.

<table>
<thead>
<tr>
<th>Location</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x059</td>
<td>0xA8</td>
</tr>
<tr>
<td>0x05A</td>
<td>0x30</td>
</tr>
<tr>
<td>0x05B</td>
<td>0xF2</td>
</tr>
<tr>
<td>0x05C</td>
<td>0x6E</td>
</tr>
<tr>
<td>W register</td>
<td>0x02</td>
</tr>
</tbody>
</table>

**g.** (6 pts) Give the final contents of any changed REGISTERS or MEM locations.

```
lfsr FSR0, array1
movlw low array2
movwf src
movlw high array2
movwf src+1
movlw 5
movwf length
call intIncCopy
```

**h.** (6 pts) Give the final contents of any changed REGISTERS or MEM locations.

```
movlw 0x59
movwf FSR0L
clrf FSR0H
movff 0x05C, INDF0
```

**i.** (6 pts) Give the final contents of any changed REGISTERS or MEM locations.

```
lfsr FSR0, 0x05A
movff POSTDEC0, 0x059
```

**j.** (6 pts) Give the final contents of any changed REGISTERS or MEM locations.

```
lfsr FSR0, 0x05A
decf PLUSW0, f
```

«User_ID»
h. (20 points) After the execution of ALL of the C code below, fill in the memory location values. Assume little-endian order for multi-byte values.

```c
signed char a[2];
signed int b;
signed long c;
signed int *ptrb;

a[0] = 3;
a[1] = -7;
ptrb = &a[0];
b = *ptrb + 2;
ptrb++;  // Computes *ptr >> 1 as int, then converts to a long

c = *ptrb >> 1;
```

Location | Contents (**MUST** be given in hex)
---|---
0x1A0 | 0x03 ; a[0] = 3
0x1A1 | 0xF9 ; a[1] = -7 = 0xF9
0x1A2 | 0x05 ; b = *ptrb + 2. ptrb points to a[0 – 1].
0x1A3 | 0xF9 0xF9 + 1 = 0xF905. 0x05 is low byte, 0xF9 is high byte.
0x1A4 | 0x82 ; c = *ptrb >> 1. ptrb now points to b, so
0x1A5 | 0xFC 1111 1001 0000 0101 >> 1 = 1 1111 1001 0000 010 = 0xFFFF FC82
0x1A6 | 0xFF As a long, this is 0xFFFF FC82
0x1A7 | 0xFF
0x1A8 | 0xA2 ; ptrb points to b, at 0x1A2
0x1A9 | 0x01
Part II: (16 points) Answer the 4 out of the next 5 questions. Each question is worth 4 points. Cross out the question you do not want graded.

a. Write PIC18 assembly code for the C statement below assuming that FSR0 already contains the address of “ptr”.

```assembly
long *ptr;
ptr++;
```

```assembly
movf POSTINC0, w
movf POSTINC0, w
movf POSTINC0, w
movf POSTINC0, w ; increment 4x since point to long
```

b. Draw the waveform for the value 0x37 using 1 start bit, 1 stop bit, and 8 data bits as transmitted by the PIC from its TX pin using asynchronous serial IO. Label each bit and indicate which bit is the start bit and which bit is the stop bit.

```
0x37 = 0011 0111. Sent over serial, this is reversed to 1110 1100. As sent:
```

```
STAR  T (0) 1 1 1 0 1 1 0 0 STOP (1)
```

```c
SPBRG (x in the equation in the datasheet) = Fosc/(16*baud) – 1 when BRGH = 1.
Computing, SPBRG = 10 MHz/(16 * 9600) – 1 = 64.1, so SPBRG = 64.
When BRGH = 0, SPBRG = Fosc/(64*baud) – 1 = 10 MHz/(64 * 9600) – 1 = 15.3, so SPBRG = 15.
```

c. Give the value of the SPBRG register required to communicate at 9600 baud with Fosc = 10 MHz. State the value of the BRGH bit used to achieve this baud rate.

```
Other correct answers: EN = 0 and IN = 0 or 1 produces OUT = hi-Z for the tri-state buffer; IN = 1 produces OUT = hi-Z for the open-drain buffer.
```

d. What inputs cause a tri-state buffer and an open-drain buffer to produce the same output (i.e. drive to Vdd, drive to ground, high-impedence)? Label the inputs and outputs in the diagram below.

```
Tri-state buffer
```

```
Open-drain buffer
```

```
Same output
```

e. Draw a pushbutton connected to a PIC18 port that will provide a logic ‘1’ when the switch is pressed, and a logic ‘0’ when the switch is released. Your connection must be electrically viable, i.e, you cannot produce a short between VDD and GND or any other undesirable condition.