• You may NOT use a calculator. 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>12</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
Part I: (60 pts)

a. (6 pts) Write a PIC18 assembly code fragment to implement the following:

```assembly
signed int   i, j, k;

j = k - i;
```

b. (6 pts) Write a PIC18 assembly code fragment to implement the following:

```assembly
signed int   j, k;
signed char  i;

while (j>=k) {
    i--;
}
```
c. (6 pts) Write a PIC18 assembly code fragment to implement the following:

```assembly
unsigned long j, k;

k = k & j;
```

d. (6 pts) Write a PIC18 assembly code fragment to implement the following:

```assembly
unsigned int j, k;
unsigned char i;

if (j==k)
    i = i>>1;
```

e. (6 pts) What is the value of j (in HEX) after the execution of the following code?

```assembly
signed char i, j;

i = 0x88;
j = i>>3;
```
f. (8 pts) Write a PIC18 assembly code fragment to implement the following. The code of the if body has been left intentionally blank; I am only interested in the comparison test. For the if/else body code, just use a couple of dummy instructions or a comment so I can see the start/begin of the if/else body.

```assembly
unsigned int i, j, k;

if ( (i < j) && k )
{ // if stuff
} else {
    // else stuff
}
```

g. (8 pts) Write a PIC18 assembly code fragment to implement the following:

```assembly
signed int i, j;

j = i >> 2;
```

h. (8 pts) Write a PIC18 assembly code fragment to implement the following. The code of the loopbody has been left intentionally blank; I am only interested in the comparison test. For the loop body code, just use a couple of dummy instructions or a comment so I can see the start/begin of the loop body.

```assembly
signed int a;
signed char b, c;

do {
    // stuff
} while (!a || (b == c) )
```
i. (6 pts) Starting at instruction “Start:”, fill in the table with the order in which instructions are executed. Give the label and instruction as shown. The first instruction is already filled in for you.

```
Start: call SubA
Start1: rcall SubB
Start2: goto SubC
Start3: bra Start3
SubA: goto SubB
SubA2: nop
SubA3: return
SubA4: return
SubB: nop
SubB1: return
SubB2: return
SubC: nop
SubC1: goto SubB
```

<table>
<thead>
<tr>
<th>#</th>
<th>Instruction label</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Start</td>
<td>call SubA</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Part II. (40 pts) Answer the following questions.

1. (4 pts) When would you HAVE to use a call instruction instead of an rcall instruction?

2. (4 pts) The value 0xDF is a two’s complement, 8-bit number. What is the decimal value?

3. (4 pts) Give the value of –2 as a 16-bit two’s complement number. Give your answer in hex.

4. (5 pts) Give the result of the operation 0x73 – 0xD0, and the V, N, C, Z flag settings.

   Result = ______
   V = ___
   N = ___
   C = ___
   Z = ___

5. (4 pts) Give the result of the operation 0x40 + 0xA3, and the V, N, C, Z flag settings.

   Result = ______
   V = ___
   N = ___
   C = ___
   Z = ___

6. (5 pts) What value is pushed on the stack by the rcall instruction below?

<table>
<thead>
<tr>
<th>Location</th>
<th>Contents</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0150</td>
<td>0xDFD7</td>
<td>rcall 0x0100</td>
</tr>
<tr>
<td>0x0152</td>
<td>0x2A7F</td>
<td>incf 0x07F,f</td>
</tr>
</tbody>
</table>
7. (5 pts) In the code below, what is the value of \( i \) (in \textit{HEX}) when the loop is exited?

```c
unsigned char i;

i = 0xD3;
while (i<0) {
    i = i<<1;
}
```

8. (4 pts) In the code below, \( j \) is a \texttt{LONG} variable which starts at memory location 0xEA. Give the contents of locations 0xEA,0xEB,0xEC,0xED if the bytes are stored in \textit{little-endian} order:

```c
signed long j;

j = 0xDEADBEEF;
```

9. (5 pts) Write the machine code for all five (5) assembly instructions. Assume some VALID data memory address for variables "a" and "i". Write down your assumptions. Give your answer in \textit{HEX}.

```
loop_top:
    movlw 5
    addwf a, f
    decf i, f
    bz loop_top
    retlw 1
```