• 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;
```

```assembly
movf i, w  ; move i’s LSB to W
subwf k, w  ; w=LSB of k-i
movwf j  ; j=k-i   LSB
movf i+1, w  ; move i’s MSB to W
subwfb k+1, w  ; w=k-i   MSB
movwf j+1   ; j=k-i   MSByte
```

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--;
}
```

```assembly
loop_top:
    movf k, w  ; NOTE that j and k
    subwf j, w  ; are INTs (16-bit)
    movf k+1, w
    subwfb j+1, w  ; set STATUS on (j-k)
    bov v_1  ; if V then goto v_1
    bn loop_end  ; exit if V=0,N=1
    bra loop_body  ; else goto loop_body
v_1:
    bnn loop_end  ; exit if V=1,N=0
loop_body:
    decf i, f  ; else goto loop_body
    bra loop_top
loop_end:
    ...; rest of code...
```
c. (6 pts) Write a PIC18 assembly code fragment to implement the following:

\[ \text{unsigned long } j, k; \]

\[ k = k \& j; \]

\[
\begin{align*}
\text{movf} & \quad j, w; \\
\text{andwf} & \quad k, f; \quad ; k = k \& j, \text{LSByte} \\
\text{movf} & \quad j+1, w; \\
\text{andwf} & \quad k+1, f; \quad ; k = k \& j, \text{byte #2} \\
\text{movf} & \quad j+2, w; \\
\text{andwf} & \quad k+2, f; \quad ; k = k \& j, \text{byte #3} \\
\text{movf} & \quad j+3, w; \\
\text{andwf} & \quad k+3, f; \quad ; k = k \& j, \text{MSByte}
\end{align*}
\]

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

\[ \text{unsigned int } j, k; \]
\[ \text{unsigned char } i; \]

\[
\begin{align*}
\text{if } (j==k) \\
\quad & i = i >> 1; \\
\text{movf} & \quad k, w; \\
\text{subwf} & \quad j, w; \quad ; \text{you could “andwf” as well} \\
\text{bnz} & \quad \text{if_end}; \quad ; \text{exit if LSBytes unequal} \\
\text{movf} & \quad k+1, w; \\
\text{subwf} & \quad j+1, w; \\
\text{bnz} & \quad \text{if_end}; \quad ; \text{exit if MSBytes unequal} \\
\text{bcf STATUS, C} \\
\text{rrcf i, f}; \quad ; i = i >> 1; \\
\text{if_end}: \\
\text{rest of code ...}
\end{align*}
\]

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

\[ \text{signed char } i, j; \]

\[
\begin{align*}
i & = 0x88; \\
j & = i >> 3;
\end{align*}
\]

This is a signed char, so it must maintain sign bit thru the right shifts.

\[
\begin{align*}
0x88 >> 3 & = 1000 1000 >> 3 \\
& = 1111 0001 \\
& = 0xF1 \quad (-120/8 = -15)
\end{align*}
\]
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 loop body 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.

<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>SubA</td>
<td>goto SubB</td>
</tr>
<tr>
<td>3</td>
<td>SubB</td>
<td>nop</td>
</tr>
<tr>
<td>4</td>
<td>SubB1</td>
<td>return</td>
</tr>
<tr>
<td>5</td>
<td>Start1</td>
<td>rcall SubB</td>
</tr>
<tr>
<td>6</td>
<td>SubB</td>
<td>nop</td>
</tr>
<tr>
<td>7</td>
<td>SubB1</td>
<td>return</td>
</tr>
<tr>
<td>8</td>
<td>Start2</td>
<td>goto subC</td>
</tr>
<tr>
<td>9</td>
<td>SubC</td>
<td>nop</td>
</tr>
<tr>
<td>10</td>
<td>SubC1</td>
<td>goto subB</td>
</tr>
<tr>
<td>11</td>
<td>SubB</td>
<td>nop</td>
</tr>
<tr>
<td>12</td>
<td>SubB1</td>
<td>return</td>
</tr>
<tr>
<td>13</td>
<td>RESET (due to stack underflow)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>NOTE: Do NOT write code like this!!!!!!</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?

   If target address of the rcall is further away than –1024 or +1023 instruction words, have to use a CALL.

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

   Sign of decimal number is negative. Magnitude is 0x00 – 0xDF = 0x21 = 33
   Final answer = -33.

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

   -2 = 0xFE as an 8-bit number, as a 16 bit number, sign extend and get 0xFFF

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

   \[
   \text{Result} = \frac{0x73 - 0xD0}{0xA3}.
   \]
   \[
   V = 1 \quad \text{because } +N - (-N) \text{ is same as } (+N) + (+N), \text{ should give positive number}.
   \]
   \[
   \text{But result is negative, so overflow.}
   \]

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

   \[
   \text{Result} = \frac{0x40 + 0xA3}{0xE3}.
   \]
   \[
   V = 0 \quad \text{as } +N + (-N) \text{ cannot produce overflow}.
   \]

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>

   Address of next instruction, 0x0152 is pushed on stack.
7. (5 pts) In the code below, what is the value of \(i\) (in HEX) when the loop is exited?

```c
unsigned char i;

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

DOH!! “\(i\)” should be defined as a “signed char”

If you read the prof’s mind:
“\(i\)” is a signed char. As decimal number, \(i\) equals –45.

As \(i\) is left shifted, eventually (after the third shift) the MSb of \(i\) will be ZERO. The contents of \(i\) will be 0x4C (or 76 in decimal). At this point \(i\) is no longer less than 0, so loop is exited with value \(i = 0x4C\).

If you worked the problem as stated:
“\(i\)” is an unsigned char and is either positive or zero. Since 0xD3 is nonzero, then “\(i\)” is positive, and the while body will never execute. The while loop is exited with value \(i = 0xD3\).

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

```c
signed long j;
j = 0xDEADBEEF;
```

<table>
<thead>
<tr>
<th>Location</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0EA</td>
<td>0xEF</td>
</tr>
<tr>
<td>0x0EB</td>
<td>0xBE</td>
</tr>
<tr>
<td>0x0EC</td>
<td>0xAD</td>
</tr>
<tr>
<td>0x0ED</td>
<td>0xDE</td>
</tr>
</tbody>
</table>

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 HEX.

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

movlw 5 is assembled to 0x0E05
addwf a,f is assembled to 0x26?? (access location for a) or 0x27?? (banked location for a)
decf i,f is assembled to 0x06?? (access location for i) or 0x07?? (banked location for i)

The “bz” instruction is encoded as 1110 0000 nnnn nnnn where the destination is PC + 2 + 2n. At PC + 2 is the retlw instruction. From there, we need to “branch” \(n = -4\) instructions. In binary, \(-4 = 1111 1100 = 0xFC\). The overall instruction is therefore 0xE0FC.

retlw 1 is assembled to 0x0C01