You may NOT use a calculator. You may use only the provided reference materials. If a binary result is required, give the value in HEX. Assume all variables are in the first 128 locations of bank 0 (access bank) unless stated otherwise. *For any signed right shifts, assume that the sign bit is preserved.*

a. (4 points) Write a PIC18 assembly language code fragment to implement the following.

```
unsigned long i, k;   // this is a LONG, be careful
k = k - i;
```

```
 movf i,w
 subwf k,f
 movf i+1,w
 subwfb k+1,f
 movf i+2,w
 subwfb k+2,f
 movf i+3,w
 subwfb k+3,f
```

b. (10 points) 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 so I can see the start/begin of the loop body.

```
unsigned int j,k;
while (j == 0 && k != 0) {
  ...operation 1...
  ...operation 2...
}
```

```
loop_top
  movf j,w
  iorwf j+1,w
  bnz loop_end ;if test is false, can skip loop since ‘&&’ condition
  movf k,w
  iorwf k+1,w
  bz loop_end // loop body
  // loop body
  bra loop_top ;don’t forget to loop back to top
loop_end
```
c. (10 points) Fill out the following table:

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Test</th>
<th>TRUE case</th>
<th>FALSE case</th>
</tr>
</thead>
<tbody>
<tr>
<td>p &gt; k</td>
<td>K - P</td>
<td>$V = <em>0</em> \text{ and } N = <em>1</em>\text{ OR}$</td>
<td>$V = <em>0</em> \text{ and } N = <em>0</em>\text{ OR}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$V = <em>1</em> \text{ and } N = <em>0</em> $</td>
<td>$V = <em>1</em> \text{ and } N = <em>1</em> $</td>
</tr>
</tbody>
</table>

signed int p, k;

d. (8 points) 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 so I can see the start/begin of the loop body.

```
signed char i, k;
while (i >= k) {
    ...operation 1...
    ...operation 2...
}
```

```
loop_top:
    movf k, w
    subwf i, w
    bov L1
    bnn loop_body ; if true, loop body
bra loop_exit ; exit
L1:
    bnn loop_exit ; exit
loop_body:
    ...code for operation 1...
    ...code for operation 2...
bra loop_top
loop_exit:
    ...rest of code...
```
e. (8 points) 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{} body code, just use a couple of dummy instructions so I can see the start/begin of the if{} body.

```assembly
movf p,w
subwf k,w
bnz if_body
movf p+1,w
subwf k+1,w
bz if_else
if_body
//if body instructions
bra if_end ; don’t forget this!
if_else
//else body instructions
if_end
```

```c
signed int k, p;
if (k != p)
{
    ...operation 1...
    ...operation 2...
} else {
    ...operation 3...
    ...operation 4...
}
```

f. (10 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 filled in).

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Start</td>
<td>Start1: call subB</td>
</tr>
<tr>
<td>2:subB</td>
<td>call subA</td>
</tr>
<tr>
<td>3:subA</td>
<td>nop</td>
</tr>
<tr>
<td>4:subA1</td>
<td>goto subB1</td>
</tr>
<tr>
<td>5:subB1</td>
<td>nop</td>
</tr>
<tr>
<td>6:subB2</td>
<td>return</td>
</tr>
<tr>
<td>7:subB1</td>
<td>nop</td>
</tr>
<tr>
<td>8:subB2</td>
<td>return</td>
</tr>
<tr>
<td>9:Start1</td>
<td>nop</td>
</tr>
<tr>
<td>10:Start2</td>
<td>nop</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Start:</th>
<th>call subB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start1:</td>
<td>nop</td>
</tr>
<tr>
<td>Start2:</td>
<td>nop</td>
</tr>
</tbody>
</table>

| subA:      | nop               |
| subA1:     | goto subB1       |
| subA2:     | return            |

| subB:      | call SubA         |
| subB1:     | nop               |
| subB2:     | return            |
g. (5 pts) Write a PIC18 assembly language code fragment to implement the following.

```assembly
g. (5 pts) Write a PIC18 assembly language code fragment to implement the following.

```unsigned int i, k;
```i = k << 1;
```  
bcf STATUS,C  
rlcf k,w  
movwf i  
rlcf k+1,w  
movwf i+1  

```

h. (4 pts) Write a PIC18 assembly language code fragment to implement the following.

```assembly
h. (4 pts) Write a PIC18 assembly language code fragment to implement the following.

```unsigned int i, k;
```i = k & i;
```  
```movf k,w  
andwf i,f  
movf k+1,w  
andwf i+1,f  

```

i. (5 pts) Write a PIC18 assembly language code fragment to implement the following.

```assembly
i. (5 pts) Write a PIC18 assembly language code fragment to implement the following.

```signed int i, k; //this is SIGNED!!!
```i = k >> 1;
```  
bcf STATUS,C  
btfsc k+1,7 ;test MSbit of MSbyte  
bsf STATUS,C  
rrcf k+1,w  
movwf i+1  
rrcf k,w  
movwf i  

```
j. (12 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 int a[2];
signed int *ptra

a[0] = 5;
a[1] = -7;
ptra = &a[1];
*ptra = *ptra + 3;
ptra--;  
```

<table>
<thead>
<tr>
<th>Location</th>
<th>Contents (MUST be given in hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0020</td>
<td>0x05</td>
</tr>
<tr>
<td>0x0021</td>
<td>0x00</td>
</tr>
<tr>
<td>0x0022</td>
<td>0xFC</td>
</tr>
<tr>
<td>0x0023</td>
<td>0xFF</td>
</tr>
<tr>
<td>0x0024</td>
<td>0x20</td>
</tr>
<tr>
<td>0x0025</td>
<td>0x00</td>
</tr>
<tr>
<td>0x0026</td>
<td></td>
</tr>
<tr>
<td>0x0027</td>
<td></td>
</tr>
<tr>
<td>0x0028</td>
<td></td>
</tr>
<tr>
<td>0x0029</td>
<td></td>
</tr>
<tr>
<td>0x002A</td>
<td></td>
</tr>
<tr>
<td>0x002B</td>
<td></td>
</tr>
<tr>
<td>0x002C</td>
<td></td>
</tr>
<tr>
<td>0x002D</td>
<td></td>
</tr>
<tr>
<td>0x002E</td>
<td></td>
</tr>
</tbody>
</table>

- **a[0]** = 0x00 05
- **a[1]** = 0xFF FC
- **ptra** = &a[1];  
  ; points at a[1]
- **ptra** = *ptra + 3;  ; add 3 to a[1]
  
  
  
  
  +4 = 0x04;  -4 = 0x00 – 0x04 = 0xFC
  
  
  
  
  Sign extend to 16 bits

  
  
  
  
  -4 = 0xFF FC

  
  
  
  
  ptra = &a[1];
  
  
  
  
  so
  
  
  
  
  ptra = 0x0022
  
  
  
  
  ptra--;  ;decrement ptra
  
  
  
  
  this is :
  
  
  
  
  ptra = ptra – 1 * sizeof(INT)
  
  
  
  
  ptra = ptra – 1 * 2
  
  
  
  
  ptra = ptra – 2
  
  
  
  
  ptra = 0x0022 – 2 = 0x0020
k. (16 points) After the execution of ALL of the C code below, fill in the memory location values. Assume for each of the following problems, give the FINAL contents of changed registers or memory locations. Give me the actual ADDRESSES for a changed memory location (e.g. Location 0x0100 = 0x??). Assume these memory/register contents at the BEGINNING of EACH problem.

W register = 0x01

Memory:
0x02F0  0x03
0x02F1  0x01
0x02F2  0xF0
0x02F3  0x02
0x02F4  0xAC

1. (4 points)
   movff 0x2F2, FSR1L
   movff 0x2F3, FSR1H
   movff 0x2F4, POSTINC1

   Location 0x2F0 = 0xAC

   FSR1 = 0x2F1

2. (4 points)
   lfsr FSR1, 0x2F2
   movff POSTDEC1, 0x2F0

   Location 0x2F0 = 0xF0

   FSR1 = 0x2F1

3. (4 points)
   lfsr FSR1, 0x2F3
   movff PREINC1, 0x2F1

   Location 0x2F1 = 0xAC

   FSR1 = 0x2F4

4. (4 points)
   lfsr FSR1, 0x2F1
   movff PLUSW1, 0x2F4

   Location 0x2F4 = 0xF0

   FSR1 = 0x2F1
l. (4 pts) What return address is pushed on the stack for the following code?

```
0x0120    call  0x208
```

return address = 0x0120 + number of bytes in ‘Call 0x208’ machine word
= 0x0120 + 4
= 0x0124

m. (4 pts) Write some assembly code that will FORCE a return address stack overflow.

```
here
    call here       ;;infinite loop of call to itself
```