ECE 3724 Test #2 – Fall 2006 – Jones/Reese
Net ID: ____________________________ (no names, please)

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 must be preserved by the assembly code that you write.

Part I: (82 points)

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. (7 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...
}
```

```
op_top
    movf j,w
    iorf j+1,f
    bz   loop_body
    movf k,w
    iorf k+1,w
    bz   loop_end
loop_body
    ...operation 1..
    ...operation 2..
    bra  loop_top
loop_end
```
c. (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.

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

```assembly
loop_top  .operation 1
            .operation 2
            movf i,w
            subwf k,w ;k-i
            bov  V_1
            bnn  loop_top  ;true V=0,N=0
            bra  loop_exit

V_1            bn  loop_top  ;true V=1,N=1
loop_exit ....other instr...
```

d. (8 points) Implement the `strswap()` function given below. Assume FSR0 already contains the pointer value for `char *Va` on function entry, and FSR1 contains the pointer value for `char *Vb` but that the pointer value for `char *Vc` is passed in the CBLOCK. In the subroutine, use FSR2 to implement the pointer operations for `char *Vc`.

```c
void vecAdd(unsigned char* Va, unsigned char* Vb, unsigned char* Vc, unsigned char length)
{
    while (length)
    {
        *Vc = *Va + *Vb;
        Va++;
        Vb++;
        Vc++;
        length--;
    }
}
```

```assembly
vecAdd         movff Vc, FSR2L
               movff Vc+1, FSR2H
loop_top       movf length,w
               bz  exit
               . movf INDF0,w  ;could have used POSTINC0 here for more efficiency
               addwf INDF1,w  ;could have used POSTINC1 here for more efficiency
               movf INDF2
               movf POSTINC0,w ;not needed if POSTINC0 used previously
               movf POSTINC1,w ;not needed if POSTINC1 used previously
               movf POSTINC2,w ;not needed if POSTINC2 used previously
               decf length,f
               bra  loop_top
exit           return
```
e. (8 points) Implement the main() code below in PIC assembly. Pass the value for “char *Va” directly in FSR0, for “char *Vb” directly in FSR1. Pass the value for “char *Vc” and “char length” using the CBLOCK space for “vecAdd”.

```c
void vecAdd(unsigned char* Va, 
             unsigned char* Vb, unsigned char* Vc, unsigned char length)
{
    // some code
}
char vec1[]={120,3,10,23, 24};
char vec2[]={4,89, 12,39,210};
char vec3[5];
main()
{
    vecAdd(&vec1[0], &vec2[0], &vec3[0], 5);
}
```

```assembler
lfsr FSR0, vec1
lfsr FSR1, vec2
movlw low vec3
movwf Vc
movlw high vec3
movwf Vc+1
movlw 5
movwf length
call vecAdd
```

f. (7 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.

```assembler
movf p, w
subwf k, w
bnz if_body
movf p+1, w
subwf k+1, w
bz if_end
if_body...
if_end...
```
g. (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). WARNING: This code is written in a ‘strange’ way to determine if you really understand what the call/return statements do. Fill in the blanks until you either run out of spaces or execute the instruction at location “Start3”.

<table>
<thead>
<tr>
<th>![Table Image]</th>
<th>![Table Image]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start1:</td>
<td>call subA1</td>
</tr>
<tr>
<td>Start2:</td>
<td>nop</td>
</tr>
<tr>
<td>Start3:</td>
<td>nop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>subA1:</th>
<th>movlw 2 ; w = 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>subA2:</td>
<td>call subB1</td>
</tr>
<tr>
<td>subA3:</td>
<td>return</td>
</tr>
<tr>
<td>subA4:</td>
<td>nop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>subB1:</th>
<th>decf WREG, w ; w = w - 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>subB2:</td>
<td>bnz subA2</td>
</tr>
<tr>
<td>subB3:</td>
<td>return</td>
</tr>
<tr>
<td>subB4:</td>
<td>nop</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Label</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: Start1</td>
<td>call subA1</td>
</tr>
<tr>
<td>2: subA1</td>
<td>movlw 2</td>
</tr>
<tr>
<td>3: subA2</td>
<td>call subB1</td>
</tr>
<tr>
<td>4: subB1</td>
<td>decf wreg,w</td>
</tr>
<tr>
<td>5: subB2</td>
<td>bnz subA2</td>
</tr>
<tr>
<td>6: subA2</td>
<td>call subB1</td>
</tr>
<tr>
<td>7: subB1</td>
<td>decf wreg,w</td>
</tr>
<tr>
<td>8: subB2</td>
<td>bnz subA2</td>
</tr>
<tr>
<td>9: subB3</td>
<td>return</td>
</tr>
<tr>
<td>10: subA3</td>
<td>return</td>
</tr>
<tr>
<td>11: subA3</td>
<td>return</td>
</tr>
<tr>
<td>12: Start2</td>
<td>nop</td>
</tr>
</tbody>
</table>
(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 int a[2];
signed int *ptra;
signed long b;
signed char c;
signed long *ptrb;

a[0] = 5;
a[1] = -6;
ptra = &a[1];
*ptra = *ptra + 1;
ptra--;
ptrb = &b;
b = a[1] << 1;
c = *ptra;  // type conv. in C causes ptra to be treated as char * here
ptrb++;
```

<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>0xFB</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>0xF6</td>
</tr>
<tr>
<td>0x0027</td>
<td>0xFF</td>
</tr>
<tr>
<td>0x0028</td>
<td>0xFF</td>
</tr>
<tr>
<td>0x0029</td>
<td>0xFF</td>
</tr>
<tr>
<td>0x002A</td>
<td>0x05</td>
</tr>
<tr>
<td>0x002B</td>
<td>0x2A</td>
</tr>
<tr>
<td>0x002C</td>
<td>0x00</td>
</tr>
</tbody>
</table>

- a[0] = 5, LSB a[0] = 0x05
- MSB a[0] = 0x00
- a[1] = -5 (-6 + 1); LSB a[1] = -5 = 0xFB
- MSB a[1] = 0xFF
- ptra final = 0x0020; LSB 0x20
- MSB = 0x00
- b = -10= 0xFFFFFFFF6; LSB b = 0xF6
- b 2nd byte = 0xFF
- b 3rd byte = 0xFF
- b 4th byte = 0xFF
- c = 5 = 0x05
- ptrb final = 0x0026 +4 = 0x002A; LSB = 0x2A
- ptrb MSB = 0x00
i. (12 pts) 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

i. (4 points)
    movff 0x2F2, FSR1L
    FSR1 = __0x02F1________

    movff 0x2F3, FSR1H
    Location_0x02F0___ = __0xAC____

    movff POSTINC1, 0x2F4

j. (4 points)
    lfsr FSR1, 0x02F2
    FSR1 = __0x2F1________

    movff POSTDEC1, 0x02F0

    Location_0x2F0_____ = _______0xF0

k. (4 points)
    lfsr FSR1, 0x02F3
    FSR1 = __0x2F4________

    movff PREINC1, 0x2F1

    Location_0x2F1_____ = __0xAC____
Part II: (16 points) Answer 4 of the next 6 questions. Cross out the 2 questions you do not want graded. Each question is worth 4 points.

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

```assembly
int *ptr;
movf POSTINC0,w
ptr++;       ;;increment twice since point to int
```

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

```assembly
int *ptr;
movlw 0
incf POSTINC0,f       ;increment LSByte
addwfc POSTINC0,f      ;increment MSByte
*ptr = *ptr+1;
```

c. (1) Write an addition of two 2’s complement 8-bit numbers that will produce the following flag conditions: V = 1, N = 0, C = 1, Z = 0. (2) What range of numbers can be represented in 6 bits using 2’s complement encoding? Give the value range, do not give an equation.

```plaintext
(1) 0x80 + 0xFF = 0x7F       V=1, N=0, C=1, Z = 0
(2) in 6 bits, from -32 to +31
```
d. Give the machine code for the ‘bnz’ instruction in the following code fragment:

```assembly	nonz there  
call subA  
there:   incf j,f
```

```
branch_target = PC+2+2*N
N = (branch_target - (PC+2))/2
   = ( PC+6 - PC -2)/2 = 4/2 = 2

offset is 2. Encoding for 'bnz' with an offset of '2' is 0xE102
```

e. Write assembly code for the following:

```c
signed int k;
k = k >> 1;
```

```
bcf    STATUS,C
btfsc  k+1,7
bsf    STATUS,C
rrcf   k+1, f
rrcf   k, f
```

f. Write assembly code for the following:

```c
signed int k;
k = k << 1;
```

```
bcf    STATUS,C
rlcf   k,f
rlcf   k+1,f
```