

Circle one:      JONES section      /      REESE section

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;
```

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...
};
```

- 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.

```
signed char i, k;

do {
    ...operation 1...
    ...operation 2...
}while (k >= i)
```

- 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*.

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

<pre>; Parameter block for the vecAdd function CBLOCK 0x040     length, Vc:2; Space for parameters ENDC</pre>
---

- 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”.

```

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);

}

```

<pre> Parameter block for the vecAdd function CBLOCK 0x040     length, Vc:2;  Space for parameters ENDC  CBLOCK 0x000    ;space for main vec1:5, vec2:5, vec3:5 ENDC </pre>
---

- 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.

```

signed int p, k;

if (p != k)
{
    ...operation 1...
    ...operation 2...
}

```

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”.

```

Start1:    call subA1
Start2:    nop
Start3:    nop

subA1:    movlw 2      ;w = 2
subA2:    call subB1
subA3:    return
subA4:    nop

subB1:    decf WREG, w;   ;w=w-1
subB2:    bnz  subA2
subB3:    return
subB4:    nop

```

Label	Instruction
1: Start1	call subA1
2:	
3:	
4:	
5:	
6:	
7:	
8:	
9:	
10:	
11:	
12:	

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.

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

```
CBLOCK 0x020
  a:4, ptrb:2,b:4,c:1,ptrb:2
ENDC
```

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

Location Contents (**MUST** be given in hex)

0x0020	_____
0x0021	_____
0x0022	_____
0x0023	_____
0x0024	_____
0x0025	_____
0x0026	_____
0x0027	_____
0x0028	_____
0x0029	_____
0x002A	_____
0x002B	_____
0x002C	_____

- 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
movff 0x2F3, FSR1H
movff 0x2F4, POSTINC1
```

FSR1 = \_\_\_\_\_

Location \_\_\_\_\_ = \_\_\_\_\_

- j. (4 points)

```
lfsr FSR1, 0x02F2
movff POSTDEC1, 0x02F0
```

FSR1 = \_\_\_\_\_

Location \_\_\_\_\_ = \_\_\_\_\_

- k. (4 points)

```
lfsr FSR1, 0x02F3
movff PREINC1, 0x2F1
```

FSR1 = \_\_\_\_\_

Location \_\_\_\_\_ = \_\_\_\_\_

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".

```
int *ptr;  
  
ptr++;
```

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

```
int *ptr;  
  
*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.

- d. Give the machine code for the 'bnz' instruction in the following code fragment:

```
        bnz there
        call subA
there:   incf j,f
```

- e. Write assembly code for the following:

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

- f. Write assembly code for the following:

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