

ECE 3724 Spring 2006 Test #2 – Jones / Reese (Circle one)

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 is preserved.*

Part I: (82 points)

- a. (4 points) Write a PIC18 assembly language code fragment to implement the following. Caution – *i* is a *signed int* !!!

```
signed int i;  
  
i = i >> 1;
```

- b. (8 points) Write a PIC18 assembly code fragment to implement the following. The code of the while{} body has been left intentionally blank; I am only interested in the comparison test. For the while{} body code, just use a couple of dummy instructions so I can see the start/begin of the while{} body.

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



- e. (8 points) Implement the main() code below in PIC assembly. Pass the value for “char \*ptrb” directly in FSR0. Pass the value for “int \*ptrb” and “char c” using the CBLOCK space for “a\_sub”.

```

a_sub(char* ptrb, int *ptrb, char c)
{
    // some code
}

main()
{
    char i, n;
    int k;

    // Some code that initializes
    // n, i, k
    // Don't include this in your assembly; assume it's done elsewhere

    a_sub(&n, &k, i);
}

```

<pre> CBLOCK 0x20          ; param. block     ptrb:2, c        ; for a_sub ENDC  CBLOCK 0x30          ; param. Block for main()     n:1, i:1, k:2 ENDC </pre>
---

- f. (6 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 i, j;

if (i != j)
{
    ...operation 1...
    ...operation 2...
}

```

- g. (4 points) Write a PIC18 assembly language code fragment to implement the following.  
CAREFUL: the variables are LONG data type!!!!!!!!!!

```
signed long a, b;  
b = b ^ a; // xor operation
```



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 = 0x03

Memory:

0x01AD	0x6F
0x01AE	0x2A
0x01AF	0x59
0x01B0	0x0B
0x01B1	0x96

i. (4 points)

```
lfsr FSR1, 0x01AE
movff PLUSW1, 0x01AE
```

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

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

j. (4 points)

```
lfsr FSR1, 0x01AF
movff 0x01AD, PREINC1
```

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

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

k. (4 points)

```
lfsr FSR1, 0x01B0
movff POSTDEC1, 0x01B1
```

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

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

l. (4 points)

```
lfsr FSR1, 0x01AF
movff INDF1, 0x01B0
```

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

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

Part II: (18 points) Answer 6 of the next 8 questions. Cross out the 2 question you do not want graded. Each question is worth 3 points.

a. If a new PIC was designed with 8KB of file registers instead of the 4KB present in the 18F242, how many bits wide should the FSR0 register be? Why?

b. What will happen when the following code is executed on the PIC?  
`here: rcall here`

c. When doing addition with signed numbers, when does an overflow occur? What bit is used in unsigned numbers which prevents overflow problems?

d. How is a `bra` different from a `goto`? When MUST you use a `goto` instead of a `bra`?

e. Why are three registers necessary to determine the address in program memory stored at the top of the stack (TOSU, TOSH, TOSL), when just two registers compose FSR0 (FSR0L, FSR0H)?

f. In what cases can signed numbers be correctly compared by examining only the C (carry) flag? When does this fail? Give an example of each case.

g. Why do some instructions, such as `goto`, `call`, and `lfsr`, require 4 bytes of program memory when most others require only two bytes?

h. The C `long` data type occupies 4 bytes. If a new C data type called `extralong` was defined which occupies 5 bytes, what address does it point to after executing the following code?

```
extralong *ptr1;  
ptr1 = 0; // Make pointer point to address 0x000  
ptr1 = ptr1 + 5; // What address does it now point it?
```