Circle one: JONES section / REESE section

ECE 3724 Test #1 – Fall 2005 – Jones/Reese -- there 5 pages (3 pages front/back)

Student ID: ________________________ (no names please)

Part I: (20 pts)

a. What is the maximum size of PIC18 data memory in bytes given that a 12-bit address is used to address it? (give your answer in Kbytes, ie, 1 Kbytes, 2 Kbytes, 4 Kbytes, 8 Kbytes, etc).

$2^{12} = 2^2 * 2^{10} = 4 * 1024 = 4$ Kbytes

b. What data memory locations comprise the ACCESS bank in the PIC18 architecture? (circle one)

1. 0x000 – 0x0FF
2. 0x000 – 0x07F and 0xF00 - 0xF7F
3. 0x080 – 0x0FF and 0xF80 – 0xFFF
4. 0x000 – 0x07F and 0xF80 – 0xFFF
5. 0xF00 – 0xFFF

The access bank is first 128 locations of Bank 0, and the last 128 locations of Bank 15.

c. What is the distinguishing feature between a Finite State Machine approach to implementing a digital system and a stored program machine approach?

The Stored program machine has MEMORY; its behavior can be altered by changing the instructions in memory. The operation of the finite state machine is hardwired; you have to change the wiring to change the behavior.

d. How many instruction cycles does it take to execute the following instructions? How many clock cycles? With a 20 MHz clock, how long does it take to execute the following instructions? (give the answer in **nanoseconds**). For reference, 1 MHz has a period = 1 us = 1000 ns.

20 MHZ clock has period of 1 MHz period/20 = 1000 ns /20 = 50 ns
1 Instruction cycle = 4 clock cycles.

	Instruction Cycles	Clock Cycles	Time
incf 0x045,f	**1**	**1*4 = 4**	**4*50 = 200 ns**
movff 0x1F0, 0x2A0	**2**	**2*4 = 8**	**8 *50 = 400 ns**
goto 0x01030	**2**	**2*4 = 8**	**8 * 50 = 400 ns**
Totals	**5**	**20**	**1000 ns**

e. The Number Sequencing Computer in Lab #2 had a 16 x 6 memory for a maximum program size of 16 instructions. You then modified it to have a maximum program size of 32 instructions. What would the memory size be (stated as K x N), if the maximum program size is increased to 128 instructions?

16 x 6 memory – 16 locations, 6 bits for instruction (two bits for opcode, four bits to address 16 locations).
128 x ? memory - 128 locations, now need seven bits to address memory.
So instruction size grows byTHREE BITS. New memory size is 128 x 9 bits.

Location	Contents
0x048	0x01
0x049	0xFB
0x04A	0x90
0x04B	0xFF

Assume the W register has the value 0x5E in it, and that initial values of C, Z are both ‘0’.

Part II. (35 pts) Assume the above memory contents, W register value, initial C,Z values at the START of each instruction.

a. incf 0x04B, f

Circle one: W dest. Reg. file dest.

New value (hex) 0x00_ C_flag :_1_ , Z flag:_1_

```
[0x4B]   =  0xFF; increment 0xFF by 1, write
back to location 0x4B
   0xFF + 1 = 0x00.  Result is zero, Zero flag=1.
 Carry out of MSb, so Carry = 1.
```

b. subwf 0x04A, w

Circle one: W dest. Reg. file dest.

New value (hex) _0x32__ C_flag :__1__ , Z flag:_0_

```
[0x4A]         =  0x90
   W          = - 0x5E
new W         =   0x32
C = 1 because of NO borrow out of MSB
```

c. xorwf 0x04B, f

Circle one: W dest. Reg. file dest.

New value (hex) _0xA1_ C_flag :_0___ , Z flag:_0_

```
[0x04B] = 0xFF =  1111 1111     XOR
operation
   W    = 0x5E =  0101 1110
new [0x4B]   =    1010 0001 = 0xA1
```

d. bsf 0x48,7

Circle one: W dest. Reg. file dest.

New value (hex) 0x81__ C_flag :__0__ , Z flag:_0_

```
                  7654 3210
[0x48]   = 0x01=  0000 0001   (set bit 7)
new value [0x48]= 1000 0001 = 0x81
```

e. addlw 0x48

Circle one: W dest. Reg. file dest.

New value (hex) 0xA6__ C_flag :_0___ , Z flag:_0_

add literal 0x48 to W reg

W ← (W) + 0x48 **W = 0x5E**

+ literal 0x48

0xA6

(45 pts) PART III. Convert the following C code fragments to PIC18 assembly.

UNLESS otherwise stated in a particular problem, assume all variables are in locations 0x000 to 0x07F.

If you use a temporary memory location, use temp and assume it is in bank 0. When writing code, you **must use** symbolic names for variable names, register names, and bit names for (i.e, use: bsf STATUS, C instead of bsf 0xFD8, 0x0). You do not have to show the CBLOCK declaration for variables.

Hint: A common mistake in these problems is to write code that modifies variables to the right of the '=' sign (i.e, for 'a = b – c;' the code you write somehow modifies *b*, or *c*, as well as *a*). This in incorrect; make sure that your code only modifies variables to the left of the '=' sign.

Also, recall that 'k++' is the same as 'k=k+1;', 'j- - ' is the same as 'j = j – 1', that " i = = j" is true if *i* is equal to *j*, that "i != j" is true if *i* is not equal to *j*, "<<" is a left shift, ">>" is a right shift, '|' is bitwise logical OR, '&' is a bitwise logic AND, '^' is a bitwise logical XOR.

unsigned char i,j,k,p,q,r,s,t;

a. (7 pts)
k = (j >> 2) + i;

```
; one solution
movf  j,w          ; w = j
bcf   STATUS,C      ;C_flag=0
rrcf  WREG,w        ; w = w >> 1
bcf   STATUS,C      ; C flag = 0
rrcf  WREG,w        ; w = w >> 1
addwf i, w          ; w = i + w
movwf  k            ; k = w
```

b. (9 pts)

```
if ( (i != 0) && (j == 5) {
    //if-body – just write a placeholder here
} else {
   //else-body – just write a placeholder here
}
```

```
;;; AND condition, can execute else body if one test is false
     movf  i,f         ; i = i, test i
     bz    else_body    ;if zero, test false, execute else body
     movf  j,f       ; j = j, test j
     sublw 0x5       ; does 0x05 - j
     bnz   else_body    ; test false, do else_body
if_body
     ...some code      ;only reach here if both tests are true
     ...some code
     bra end_if       ;DO NOT FORGET to skip else_body!!
else_body
     ...some code
     ...some code

end_if
        ..rest of code
```

c. (6 pts) Write the following in assembly language

i = (k ^ j) | 0x80;

```
; one solution
movf  j,w        ;w = j
xorwf k,w        ;w = k ^ j
iorlw 0x80       ;w = w | 0x80
movwf i          ;i = w
```

d. (10 pts)

```
while ( (i > 0x20) || (j == k) )
    //loop-body – just write a placeholder here
};
```

For i > 0x20, do "0x20 – i ". The instruction *sublw* is good for this, does "literal – W"

	Operation	True case	False Case
i > 0x20	0x20 – i	Borrow, C = 0	No borrow, C = 1

Use the TRUE case because of logical OR (||) – if one of the tests is TRUE, then can execute the loop body.

```
     ; one solution
loop_top
     movf   i,w          ;w = i
     sublw  0x20         ;w = 0x20 - i   (do 0x20 – i)
     bnc    loop_body    ; if C=0, borrow, i > 0x20, perform loop
     movf   j,w
     subwf  k,w          ; k – j
     bnz    loop_exit    ; skip loop is this is false as both tests are false
     ...loop body...
     ...loop body...
     bra loop_top        ; DO NOT FORGET to loop back to top!
loop_exit
     ....rest of code....
```

e. (6 pts) Assume that ***r*** is in bank 1, ***s*** is in bank 2, and ***t*** is in bank 3. Implement the following code in assembly language:

```
t =   (r - s) << 1;
```

```
; one solution
movlb  2        ; BSR = 2
movf   s,w      ; w = s
movlb  1        ; BSR = 1, bank 1
subwf  r,w      ; w = r -s
bcf    STATUS,C
rlcf   WREG,W   ; w = w << 1
movlb  3        ; BSR = 3, bank 3
movwf  t        ; t = w
```

f. (7 pts) Write a PIC18 instruction sequence that does

```
do {
  //loop body, place holder
} while ( k < q);
```

For q > k, do “k - q “.

	Operation	True case	False Case
q > k	k-q	Borrow, C = 0	No borrow, C = 1

Use the TRUE case because if condition is true, branch back to loop top.

```
     ; one solution
loop_top
     ...loop body...
     ...loop body...
     movf   q,w
     subwf  k,w         ; w = k-q
     bnc    loop_top    ; back to top if q > k
loop_exit
     ....rest of code....
```