

ECE 3724/CS 3124 Test #2 – Spring 2005- Reese

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.

Part I: (70 pts)

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

```
signed int i, k;
```

```
i = k >> 1;
```

```
bcf STATUS,C      ;shift in '0'
movf k+1,f        ;test sign bit on int value
bnn skip          ;skip if positive
bsf STATUS,C      ;negative, shift in '1'
skip
rrcf k+1,w        ;shift MSByte
movwf i+1         ;save new MSByte
rrcf k,w          ;shift LSByte
movwf i           ;save new LSByte
```

- b. (8 pts) 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*.

```
int i, k;
```

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

```
movf i,w          ;
subwf k,w         ;test if LSBytes are equal
bnz if_body       ;if not equal, know i!=k so do if_body
movf i+1,w
subwf k+1,w
bz end_if         ;skip if both LSBytes and MSBytes are equal
if_body
    ...operation 1...
    ...operation 2...
end_if
```

- c. (8 pts) Write a PIC18 assembly code fragment to implement the following:

```
signed char j, k;
```

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

For $k > j$, do $j - k$.
If $k > j$ is true, then $j - k$ will be a negative number ($N=1, V=0$). If overflow occurs, then number will be positive ($N=0, V = 1$).

```
loop_top:
    ...code for operation 1...
    ...code for operation 2....
    movf    _k_,w
    subwf   _j_,w    ;do j-k
    bov     L1
    bn      loop_top ;true loop top
    bra     loop_exit ;exit

L1
    bnn     loop_top ;true loop top
loop_exit
    ....rest of code....
```

- d. (8 pts) Implement the *doadd* subroutine in PIC18 assembly language. Assume the parameters have been initialized by the calling function. Do NOT forget that this is a subroutine!!!!!!

```
// doadd function
doadd (unsigned int *ptrA, unsigned int *ptrB){

    *ptrA = *ptrA + *ptrB;

}
```

```
;parameter space for doadd subroutine
CBLOCK 0x020
ptrA:2, ptrB:2, ; ptrA, ptrB contains pointers to
integers
ENDC
```

```
;solution A, goes with
;solution A, problem E
;uses ptrA, ptrB parameters
movff ptrA, FSR0L
movff ptrA+1, FSR0H
movff ptrB, FSR1L
movff ptrB+1, FSR1H
movf POSTINC1,w
addwf POSTINC0,f ;add LSByte
movf POSTINC1,w
addwfc POSTINC0,f ;add MSByte
return
```

```
;solution B, goes with
;solution B, problem E
;ignores ptrA, ptrB parameters
;assumes calling routine inits
;FSR0 and FSR1
movf POSTINC1,w
addwf POSTINC0,f ;add LSByte
movf POSTINC1,w
addwfc POSTINC0,f ;add MSByte
return
```

- e. (8 pts) Implement the following in PIC18 assembly, which is a call to the subroutine *doadd* of the previous problem. The assembly code should work regardless of where the parameter block for main is located. The '&p' and '&q' passes the addresses of variables *p* and *q* to the *doadd* subroutine (these are the **ptr*, **ptrb* parameters).

```
main() {
  int p,q;
  //call function
  doadd( &p, &q);
}
```

```
;allocation for main
CBLOCK 0x????
p:2, q:2;
ENDC
;parameter space for doadd subroutine
CBLOCK 0x020
ptr:2, ptrb:2, ; ptr, ptrb contains pointers to integers
ENDC
```

```
;solution A, goes with
;solution A, problem d
;uses ptr, ptrb parameters
movlw low p
movwf ptr
movlw high p
movwf ptr+1
movlw low q
movwf ptrb
movlw high q
movwf ptrb+1
call doadd
```

```
;solution B, goes with
;solution B, problem d
;uses FSR0, FSR1 to pass
;parameters, ignores ptr, ptrb
lfsr FSR0,p ;FSR0 contains p address
lfsr FSR1,q ;FSR1 contains q address
call doadd
```

Solution B is an optimization that removes the need for the *ptr*, *ptrb* variables. I mentioned in class that I would allow this.

- f. (8 pts) 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*.

```
int i, k;

if (i || k) {
  ..operation 1...
  ..operation 2....
}
```

```
movf i,w ;
iorwf i+1,w
bnz if_body ; if i is nonzero do if_body
movf k,w
iorwf k+1,w
bz end_if ; if both i and k are zero, skip
if_body
...operation 1...
...operation 2...
end_if
```

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

```
signed int s, p, q;
```

```
s = p - q;
```

```
movf    q,w          ;  
subwf   p,w          ; w = p - q, LSByte  
movwf   s            ; save result  
movf    q+1,w        ;  
subwfb  p+1,w        ; w = p - q, MSByte, subtract with borrow  
movwf   s+1          ; save result
```

Assume the following memory contents at the START of EACH of these code fragments for problems g to h.

W register = 0x02

<pre> CBLOCK 0x015A s:1, p:1, q:1, ; char s,p,q; r:2, ; unsigned int r; t:4 ; unsigned long t ENDC Assume the following initializations: s = 0x39; p = 0x5A; q = 0xA5; r = 0x3044; (this will be stored in little ENDIAN order!!) t = 0xA5DC39FF; (this will be stored in little ENDIAN order!!) </pre>		<table border="1"> <tr><td>0x015A</td><td>s (0x39)</td></tr> <tr><td>0x015B</td><td>p (0x5A)</td></tr> <tr><td>0x015C</td><td>q (0xA5)</td></tr> <tr><td>0x015D</td><td>r (0x44) LSBByte</td></tr> <tr><td>0x15E</td><td>r (0x30) MSByte</td></tr> <tr><td>0x15F</td><td>t (0xFF) LSBByte</td></tr> <tr><td>0x160</td><td>t (0x39)</td></tr> <tr><td>0x161</td><td>t (0xDC)</td></tr> <tr><td>0x162</td><td>t (0xA5) MSByte</td></tr> </table>	0x015A	s (0x39)	0x015B	p (0x5A)	0x015C	q (0xA5)	0x015D	r (0x44) LSBByte	0x15E	r (0x30) MSByte	0x15F	t (0xFF) LSBByte	0x160	t (0x39)	0x161	t (0xDC)	0x162	t (0xA5) MSByte
0x015A	s (0x39)																			
0x015B	p (0x5A)																			
0x015C	q (0xA5)																			
0x015D	r (0x44) LSBByte																			
0x15E	r (0x30) MSByte																			
0x15F	t (0xFF) LSBByte																			
0x160	t (0x39)																			
0x161	t (0xDC)																			
0x162	t (0xA5) MSByte																			

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 0x15B = 0x??)

h. (5 pts)

```

lfsr      FSR1, s
movff    PLUSW1, p

```

FSR1 = __0x15A (plusW1 does not change FSR1)
 Location __0x15B__ = __0xA5__
 FSR1 initialized to **s** address of 0x15A
 The movff moves 0x15A+2 contents to location p, 0x15B.
 So contents of 0x15C copied to location 0x15B. FSR1 unchanged at 0x15A.

i. (5 pts)

```

movlw    low q
movwf    FSR1L
movlw    high q
movwf    FSR1H
movff    POSTDEC1, s

```

FSR1 = __0x15B__
 Location __0x15A__ = __0xA5__
 FSR1 initialized to **q** address, 0x15C.
 movff moves 0x15C to s address (0x15A).
 So 0x15A changed to 0xA5.
 FSR0 is then decremented to 0x15B.

j. (5 pts)

```

movff    r+1, r

```

Location __0x015D__ = __0x30__
 Copy contents of location 0x15E to location 0x15D.

k. (5 pts)

```

lfsr      FSR1, t
movff    POSTINC1, s

```

FSR1 = __0x160__
 Location __0x015A__ = __0xFF__
 FSR1 initialized to **t** address, 0x15F.
 movff moves 0x15F to s address (0x15A).
 So 0x15A changed to 0xFF.
 FSR0 is then incremented to 0x160.

Part II: (30 pts) Answer 10 out of the next 12 questions. Cross out the 2 questions that you do not want graded. Each question is worth 3 pts.

1. What return address is pushed on the stack for the instruction CALL 0x0300 if the location of the call instruction is 0x0154?

$$\begin{aligned} \text{new PC} &= \text{Old PC} + 4 \text{ (because CALL is 2 instruction words or 4 bytes)} \\ &= 0x0154 + 4 = 0x0158 \end{aligned}$$

2. The value 0xED is a two's complement, 8-bit number. What is the decimal value?

Because MSdigit is > 7 , number is negative. Compute magnitude as $0x00 - 0xED = 0x13$. Magnitude in decimal is $0x13 = 19$. Final answer: -19.

3. Give the value of -6 as a 16-bit two's complement number.

$+6 = 0x06$. In 8 bits, $0 - 6 = -6$, so $0x00 - 0x06 = 0xFA$.
In 16-bits, sign extend by adding 'F' digits. Final answer: 0xFFFFA

4. Give the V, N flag settings after the operation $0x80 + 0x7F$.

$0x80 + 0x7F = 0xFF$. Result is negative as MSbit is '1'.
Negative + Positive cannot produce overflow. Final answer: V = 0, N = 1

5. Give the V, N flag settings after the operation $0x7F + 0x10$.

$0x7F + 0x10 = 0x8F$. Result is negative as MSbit is '1'.
Positive + Positive produces negative, so overflow. Final answer: V = 1, N = 1

6. In the code below, what is the value of *i* when the loop is exited? Give the value in HEX!!!

```
signed char i;

i = 0x01;
while (i > 0) {
    i = i << 1;
}
```

Each time through loop, *i* is shifted left.
 Progression of *i* values is 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80.
 At this point *i* is a negative number, so the signed comparison of while(*i*>0) is false, loop is exited.
 Final *i* value is 0x80.

7. For the C code and CBLOCK show below, what is the value of *ptr* after the statement '*ptr++*'? Careful, *ptr* is pointer to type *int*

```
int *ptr;
char a[4];
int b[4];

ptr = b;
ptr++;
```

```
CBLOCK 0x200
ptr:2, a:4, b:8
ENDC
```

ptr address is 0x200.
a address is 0x200+2 = 0x202.
b address is 0x202+4 = 0x206

The assignment *ptr = b* gives *ptr* a value of 0x206. The *ptr++* operation computes 0x206+2 = 0x208 because *ptr* is a pointer to type INT, which is 2 bytes in size. Final answer: 0x0208

8. Write the CBLOCK that allocates space for the C variables below in a similar manner as done for problem 7.

```
long *ptr;
char a[4];
long b[4];

ptr = b;
ptr++;
```

```
CBLOCK 0x200

ptr: _2_, a: _4_, b: _16_

ENDC
```

The size of *ptr* DOES NOT CHANGE even though *ptr* is now points to a LONG instead of an INT. This is because *ptr* contains an address, which does not change in size. The number of bytes needed for *b* is now 4*4 = 16, as each long is 4 bytes in size.

9. Write a simple PIC18 code fragment that will force return address stack underflow.

```
; assumes there is no previous 'call' active
goto SUBA
.....
SUBA
    RETURN
```

Execute a RETURN without a corresponding CALL.

10. Give the machine code for the 'bov 0x208' instruction below given the locations shown:

location		
0x0200	bov	0x208
0x0202	???	
0x0204	???	
0x0206	???	
0x0208	incf	0x002,f

$$\begin{aligned} \text{OFFSET} &= (\text{branch target} - (\text{PC} + 2))/2 \\ &= (0x0208 - (0x200+2))/2 = (0x208-0x202)/2 = 6/2 = 3 = 0x03 \end{aligned}$$

Encoding from datasheet: 1110 0100 nnnnn nnnnn (nnnn is the 8bit displacement)

Final answer: 0x E403 .

11. Write a PIC18 assembly code fragment to implement the following.

```
signed long k,j;
```

```
k = k & j;
```

```
; a LONG is 4 bytes!!!!
movf j,w
andwf k,f      ;LSByte
movf j+1,w
andwf k+1,w
movf j+2,w
andwf k+2,w
movf j+3,w    ;MSByte
andwf k+3,w
```

12. When does a *call* instruction have to be used instead of an *rcall* instruction?

When the distance to the CALL target exceeds the 11 bit displacement, which allows +1023 instruction words forward or -1024 instruction words backwards.