

ECE 3724/CS 3124 Test #2 Solutions – Summer 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: (82 pts)

- a. (6 pts) Write a PIC18 assembly code fragment to implement the following.

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

```
i = k << 1;
```

```
;left shift, signed/unsigned makes  
;no difference  
bcf    STATUS,C    ;shift in '0'  
rlcf   k,w         ;LSByte  
movwf  i  
rlcf   k+1,w       ;MSByte  
rlcf   i+1
```

- 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  j,w  
bnz    end_if  
movf   i+1,w  
subwf  j+1,w  
bnz    end_if  
if_body  
    ... operation 1...  
    ... operation 1...  
end_if  
    ... rest of code...
```

- c. (6 pts) Write a PIC18 assembly code fragment to implement the following:

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

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

for $k \geq j$, do $k - j$

if $k \geq j$ is true, then $k - j$ results in

N=0 (positive result), V=0 (no overflow)

OR

N=1 (negative result), V=1 (overflow)

```
loop_top:
    movf    j,w
    subwf   k,w    ; k - j
    bov     L1
    bnn     loop_body ;if true, loop body
    bra     loop_exit ;exit
L1
    bnn     loop_exit ;if false, exit
loop_body:
    ...code for operation 1...
    ...code for operation 2....
loop_exit
    ....rest of code....
```

- d. (8 pts) Implement the *doshift* subroutine in PIC18 assembly language. Assume the value of *ptr* has been passed in the FSR0 register by the calling subroutine. Do not forget that this is a subroutine!!!!!!

```
// shift function
doshift (unsigned int *ptr){
```

```
    *ptr = (*ptr) >> 1;
```

```
}
```

```
;remember that for right shift, must do MSByte first
; on entry, FSR0 is pointing to LSByte, so must increment
; it before accessing *ptr
movf    POSTINC0,f    ;FSR0++, FSR points at MSByte
bcf     STATUS,C      ; shift in '0', unsigned

rrcf    POSTDEC0,f    ;shift MSByte, FSR0-- to point at LSByte
rrcf    INDF0,f       ;shfit LSByte
return
```

- e. (9 pts) Implement the *main()* code below in PIC assembly. You MUST pass the parameters for the *a_sub()* function using the CBLOCK locations for the function *a_sub()*. You CANNOT just use FSR0 for passing the *ptr* value to *a_sub()*.

```
a_sub (char c, long *ptr){
// some code ..... //

}
```

```
CBLOCK 0x040 // parm. block for a_sub
c: 1, ptr: 2
ENDC
```

```
main() {
char p;
long k;
// some code that initializes
// p, k ....., don't worry about this
//now, call a_sub() function
```

```
CBLOCK 0x060 // parm. block for main
// define p, k space here, fill in blanks
p: _1_, k: __4__

ENDC
```

```
a_sub( p, &k);
}
```

```
movff p,c ; init parameter 'c'
movlw low k ; init low byte of ptr
movwf ptr ; with low byte of &k
movlw high k ; init high byte of ptr
movwf ptr+1 ; with high byte of &k
call a_sub
```

- 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 == 0) && (k != 0) ) {
..operation 1...
..operation 2....
}
```

```
movf i,w;
iorwf i+1,w ;test 'i'
bnz end_if ;skip if i!=0
movf k,w
iorwf k+1,w ;test 'k'
bz end_if ;skip if k==0
if_body
... operation 1...
... operation 1...
end_if
... rest of code...
```

g. (6 pts) Write a PIC18 assembly code fragment to implement the following:

```
long p, q;
```

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

```
; long data types are 4 bytes
; don't forget the subtract with borrow after
; the first byte!

movf    q,w;
subwfb  p,f    ; p - q LSByte (byte 0)

movf    q+1,w;
subwfb  p+1,f  ; p - q byte 1

movf    q+2,w;
subwfb  p+2,f  ; p - q byte 2

movf    q+3,w;
subwfb  p+3,f  ; p - q MSByte (byte 3)
```

h. (15 pts)

After the execution of ALL of the C code below, fill in the memory location values.
Assume little-endian order for multi-byte values.

```

CBLOCK 0x0150
r:2, t:4, s:1, ptra:2, ptrb:2
ENDC
C code:
unsigned int r;
signed long t;           // this is SIGNED!!!!!!
signed char s;          // this is SIGNED!!!!!!
signed char *ptrs;
unsigned int *ptrb;
r = 256;                 // specified in decimal!!   (3 pts)
t = -2;                 // specified in decimal!!   (3 pts)
s = -49;                // specified in decimal!!!!  (3 pts)
ptrs = &s;              (3 pts)
ptrb = &r;              (3 pts)
ptrb++;                 (3 pts)

```

Location	Contents (MUST BE GIVEN IN HEX!!!!)	
0x0150	___ 0x00 ___	} location 'r', an <i>int</i> is two bytes 256 = 0x0100, store in little endian order
0x0151	___ 0x01 ___	
0x0152	___ 0xFE ___	} location 't', a <i>long</i> is four bytes -2 = 0 - (+2) = 0x00 - 0x02 = 0xFE Sign extend 0xFE to 32-bits, or -2 = 0xFF FF FF FE Store in little endian order
0x0153	___ 0xFF ___	
0x0154	___ 0xFF ___	
0x0155	___ 0xFF ___	
0x0156	___ 0xCF ___	} location 's', a <i>char</i> is one byte -49 = 0 - (+49) = 0x00 - 0x31 = 0xCF
0x0157	___ 0x56 ___	} location 'ptrs', a pointer to data memory is two bytes contains the address of 's', or 0x0156, store in little endian order
0x0158	___ 0x01 ___	
0x0159	___ 0x52 ___	
0x015A	___ 0x01 ___	} location 'ptrb', a pointer to data memory is two bytes initially contains the address of 'r', or 0x0150, but then ptrb++ increments by sizeof(INT), which means the pointer is incremented by 2 or 0x0150+2 = 0x0152 Store in little endian order.

i. (16 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!!!

Memory:

0x0100	0x45
0x0101	0xFF
0x0102	0xBA
0x0103	0x3C
0x0104	0x64

W register = 0x03

j. (4 pts)

FSR1 loaded with 0x0101, but PREINC1 increments FSR1 before using FSR1, so movff becomes movff 0x0102,0x100

```
lfsr    FSR1, 0x0101
movff   PREINC1, 0x0100
```

FSR1 = 0x0102

Location _0x0100_ = _0xBA_

k. (4 pts)

FSR1 loaded with 0x0100, PLUSW1 causes movff to do: movff 0x0100+3,0x100 or movff 0x0103, 0x100
FSR1 is NOT modified by PLUSW1

```
lfsr    FSR1, 0x0100
movff   PLUSW1, 0x0100
```

FSR1 = __0x0100__

Location __0x0100_ = __0x3C

l. (4 pts)

FSR1 loaded with 0x0100, POSTDEC1 causes movff to do: movff 0x0103,0x100
after the move, POSTDEC1 does FSR1-- so FSR1 final value is 0x0102

```
lfsr    FSR1, 0x0103
movff   POSTDEC1, 0x100
```

FSR1 = __0x0102__

Location _0x100_ = _0x3C_

m. (4 pts) (careful on this one!!!!)

FSR1 loaded with 0x0100,
the movff moves the contents of FSR1H (high byte of FSR1) to location 0x100, so the value 0x01 is placed in location 0x100

```
lfsr    FSR1, 0x0103
movff   FSR1H, 0x100
```

FSR1 = __0x0103__

Location _0x0100_ = __0x01

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

1. Fill in memory location below, and either a CALL or RCALL instruction (use mnemonic, not machine code) such that a value of 0x0104 is pushed as the return address on the stack

Mem location

instruction

0x100_____

CALL some_location (a call is 4 bytes)

OR

0x102

RCALL some_location (a rcall is 2 bytes)

2. Write an 8-bit addition such that afterwards, both the V and the N flags are set.

0x7F + 0x01 = 0x80 afterwards, V=1,N=1

Had to be a positive+positive = negative to produce overflow and N=1

3. Given an N-bit number, what number range can I represent using 2's complement encoding?

$-2^{(N-1)}$ to $+2^{(N-1)} - 1$, for N=8 range is -128 to +127

MSByte of 'bnn' is E7, low byte is the 8-bit branch offset

4. In the code below, what is the value of *i* when the loop is exited? Give the value in either hex or decimal.

```
signed char i;

i = 0x80;
while (i <= -32) {
    i = i >> 1;
}
```

Our convention has been that a signed right shift maintains the sign bit. 'i' is initially -128. Each right shift is a divide by 2, so 'i' becomes -64, -32, -16 at which point the loop exits.
Final value: $i = -16 = 0xF0$

5. Give the machine code for the 'bnn 0x200' instruction below given the locations shown:

location	
0x0200	decf 0x02,f
0x0202	???
0x0204	???
0x0206	???
0x0208	bnn 0x200

MSByte of 'bnn' is E7, low byte is the 8-bit branch offset, a two's complement value

branch offset = $(\text{target location} - (\text{branch location} + 2)) / 2$
 $= (0x0200 - (0x208+2))/2 = -10/2 = -5 = (0x00 - 0x05) = 0xFB$
Final instruction: 0xE7FB

6. On the PIC18, can I nest subroutine calls as deep as I want? (i.e. subroutine A calls subroutine B calls Subroutine C calls Subroutine Detc). If NO, then why? Be detailed.

The return address stack only room for 31 addresses; on the 32nd nested call without a return the stack will overflow. So no, you cannot nest subroutine calls as deep as you want. There is always some limit the number of return addresses that can be stored in a stack since memory is always finite.

7. **Why** is the width of the FSR0, FSR1, FSR2 registers different from the width of registers like the WREG and general purpose registers in the data memory?

The FSRn register are used as pointers to data memory, which means that they contain the address of a data memory value. The width of this address depends on the number of locations in memory; the data memory on the PIC18 has a maximum of 4096 locations, or 2^{12} , requiring them to be 12-bits wide.

8. Why can't subroutine calls just be implemented with GOTO statements? What is the special feature of CALL/RETURN instructions that is absolutely essential to implementing subroutines?

A GOTO does not save a return address; for a subroutine call to function correctly, the subroutine must know where to return to after the subroutine is finished. This is the return address, which is saved by the call instruction on the return address stack. The return address is the address of the instruction following the call instruction.