

ECE 3724/CS 3124 Test #2 Solutions – Fall 2004- 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 bank 0.

Part I: (70 pts)

- a. (10 pts) Write a PIC18 assembly code fragment to implement the following (NOTE: these are **LONG** variables, which are 4 bytes!!!!)

long i, k;

i = i + k;

```

movf    k,w
addwf   i,f
movf    k+1,w
addwfc  i+1,f
movf    k+2,w
addwfc  i+2,f
movf    k+3,w
addwfc  i+3,f
    
```

Longs are 4 bytes, so need 4 adds. Last three adds are 'addwfc' (add with carry)

- b. (10 pts) Write a PIC18 assembly code fragment to implement the following:

signed int j, k;

```

do{
    j = j << 1;
}while(k >= j)
    
```

Must do k-j. If k >= j, then k-j is a positive number (N=0, V=0). If overflow, then must test (V=1, N=1)

```

loop_top:
    bcf        STATUS,C
    rlcf       j,f
    rlcf       j+1,f
    movf       j,w
    subwf      k,w
    movf       j+1,w
    subwfb     k+1,w
    bov        L1
    bnn        loop_top ;true loop top
    bra        loop_exit ;exit

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

- c. (5 pts) Write a PIC18 assembly code fragment to implement the following:
signed int k,j;

k = k >> 1 ;

This is an INT, so must shift both bytes. Must preserve the sign bit; the sign bit is in the MSByte (k+1).

```

bcf        STATUS,C
movf       k+1,w        ; test sign bit
bnn        positive    ; if positive, keep C=0
bsf        STATUS,C    ; to keep sign=1, set C
positive
rrfc       k+1,f        ;shift MSByte, C into sign bit
rrfc       k,f          ;shift LSByte
    
```

- d. (10 pts) Implement the swap subroutine in PIC18 assembly language. Assume the parameters have been initialized by the calling function. Do NOT forget that this is a subroutine!!!!!!

```
// do swap
swap (unsigned char *s1, unsigned char *s2){
    unsigned char c;

    c = *s1;    // save *s1 value
    *s1 = *s2; // copy value
    *s2 = c;    // s2 gets old s1 value
}
```

```
;parameter space for swap subroutine
CBLOCK 0x020
s1:2, s2:2, c ; s1,s2 contains pointers to strings
ENDC
```

```
movff s1,FSR0L
movff s1+1,FSR0H ;FSR0 = s1
movff s2,FSR1L
movff s2+1,FSR1H ;FSR1 = s2
movff INDF0,c ;c = *s1
movff INDF1,INDF0 ;*s1 = *s2;
movff c, INDF1 ;*s2 = c
return
```

- e. (10 pts) Implement the following in PIC18 assembly. Assume the parameters have been initialized by the calling function. CAREFUL: *ptr* is a pointer to an INT value, and *val* is an INT value as well! The return value of '0' or '1' must be returned in the W register.

```
// do equality test
char test(unsigned int *ptr, unsigned int val){

    if (*ptr == val) return(0);
    else return(1);

}
```

```
;parameter space for test subroutine
CBLOCK 0x020
ptr:2, val:2
ENDC
```

```
movff ptr,FSR0L
movff ptr+1,FSR0H ;FSR0 = ptr
movf POSTINC0,w ;w= *ptr; ptr++ , need to increment FSR0 to next byte
subwf val,w
bnz return_1
movf INDF0,w ;
subwf val+1,w ;do MSByte
bnz return_1
retlw 0
return_1
retlw 1
```

It is invalid to do:
movf INDF0+1,w

You need to increment FSR0 by using either POSTINC0 or PREINC0.

- f. (10 pts) Implement the following in PIC18 assembly, which is a call to the subroutine 'test' of the previous problem. The assembly code should work regardless of where the parameter block for main is located. The '&i' passes the address of variable *i* to the test subroutine (this is the *ptr parameter). The value of *j* is passed to parameter 'val'. The return value of the subroutine *test* returns in W and should be written to 'k'.

```
main() {
int i,j;
char k;
k = test( &i, j);
}
```

```
;allocation for main
CBLOCK 0x????
i:2,j:2:, k
ENDC

;parameter space for test
CBLOCK 0x020
ptr:2, val:2
ENDC
```

```
movlw low i ;low byte of address of i (&i)
movwf ptr ;ptr is low byte of &i
movlw high i
movwf ptr+1 ;ptr+1 is high byte of &i
movff j, val ; value of j lowbyte to val lowbyte
movff j+1,val+1 ; value of j high byte to val high byte
call test
movwf k
```

An alternate solution

```
lfsr FSR0,i ; FSR0 = address of i
movff FSR0L,ptr
movff FSR0H,ptr+1
movff j, val ; value of j lowbyte to val lowbyte
movff j+1,val+1
call test
movwf k
```

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

Location	Contents:
0x060	0xF2
0x061	0x50
0x062	0xA5
0x063	0x0B

W register = 0x02

g. (5 pts) Give the final contents of any changed REGISTERS or MEM locations.

```
lfsr    FSR0, 0x061
movff   PREINC0, 0x60
```

Final value of FSR0 is 0x062 (incremented from 0x60 to 0x61 by PREINC0). Contents of location 0x062 copied to location 0x060, so 0x060 contents changed to 0xA5

h. (5 pts) Give the final contents of any changed REGISTERS or MEM locations.

```
movlw   0x060
movwf   FSR0L
clrf    FSR0H
movff   0x063, INDF0
```

Final value of FSR0 is 0x060.
Contents of location 0x063 copied to location 0x060, so 0x060 contents changed to 0x0B

i. (5 pts) Give the final contents of any changed REGISTERS or MEM locations.

```
lfsr    FSR0, 0x061
movff   POSTDEC0, 0x60
```

Final value of FSR0 is 0x060 (decremented from 0x61 to 0x60 by POSTDEC0).
Contents of location 0x061 copied to location 0x060, so 0x060 contents changed to 0x50

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. How does a call instruction differ from a goto instruction?

A call instruction pushes the address of next instruction on the return address stack, then jumps to its target location. A goto simply jumps to its target location.

2. The value 0xE2 is a two's complement, 8-bit number. What is the decimal value?

This is a negative number (MSb = 1), so $0x00 - 0xE2 = 0x1E = 30$ is magnitude.
Final answer: -30

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

$-3 = 0x00 - 0x03 = 0xFD$, sign extended to 16 bits is 0xFFFD

4. Give the V, N flag settings after the operation $0x60 + 0x40$.

$0x60 + 0x40 = 0xA0$. MSb = 1, so N=1
Two positive numbers added give a negative, so V = 1

5. Give the V, N flag settings after the operation $0x80 - 0x1$.

$0x80 - 0x01 = 0x7F$. MSb = 0, so N=0
In decimal $-128 - 1$ should equal -129 , but this is too large for 8 bits, so V = 1.

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

```
signed char i;

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

Each time through the loop, *i* is divided by 2 by right shift
So *i* is -128, -64, then -32.

i = -32 (0xE0) on loop exit.

7. Define precisely what causes the stack *overflow* condition on the PIC18.

When a return address is pushed on a full stack. The stack has 31 positions for storing return addresses.

8. In the code below, *j* is a LONG variable which starts at memory location 0x020. Give the contents of locations 0x020,0x021,0x022,0x023 if the bytes are stored in LITTLE ENDIAN order:

```
long j;
j = 0xA433FA02;
```

Location	Contents
0x020	0x02
0x021	0xFA
0x022	0x33
0x023	0xA4

9. In the C code fragment below, assume FSR0 is used to implement the 'ptr' variable. Write a PIC18 code fragment that implements the 'ptr++' operation. CAREFUL.....*ptr is a pointer to type 'int', which is two bytes long.

```
int *ptr;

ptr++;
```

FSR0 must be incremented twice

movf POSTINC0, w
movf POSTINC0, w

10. Give the machine code for the 'bnn 0x0200' instruction below given the locations shown:

location		
0x0200	incf	0x020,f
0x0202	???	
0x0204	???	
0x0206	bnn	0x0200

Displacement = (target - (PC+2))/2 = 0x0200 - 0x0206 = 0xFFF8/2 = 0xFFFC (lower 8 bits)
 Machine code = 0x E 7 ?? where ?? is displacement, so answer = 0x E7FC

11. Write a PIC18 assembly code fragment to implement the following:
 signed int k,j;

k = k | j;

```

movf    j,w
iorwf   k,f      ;LSByte
movf    j+1,w
iorwf   k+1,f    ;MSByte
    
```

12. Of the signed comparisons below, circle the one that is TRUE!!!

unsigned char r, s;
 signed char i, k;
 signed int p, q;

r = 0x8F; s = 0x40;
 i = 0x8F; k = 0x40;
 p = 0xFF8F; q = 0x0040;

a. i less than (<) k

i is negative, k is positive; TRUE

b. r less than (<) s

unsigned char, r is greater than s; FALSE

c. p greater than (>) q

p is negative, q is positive; FALSE