ECE 3724/CS 3124 Test #2 Solutions – Summer 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.

Part I: (70 pts)

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

```
int k, j, i;

j = k - i;
```

```
movf  i,w        ;
subwf k,w
movwf j      ; j=k-i, LSByte
movf  i+1,w        ;
subwfb k+1,w
movwf j+1      ; j=k-i, MSByte
```

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

```
signed int j, k;
char i;

while (j >= k){
     i--;
  }
```

Do j-k, check for V=0, N=0 or V=1, N=1 as true condition

```
loop_top
    movf  k,w       ;
    subwf j,w
    movf  k+1,w       ;
    subwfb j+1,w     ; j - k
    bov    v_1
    bn    loop_end  ;exit if V=0,N=1
    bra   loop_body
v_1
    bnn   loop_end  ;exit if V=1,N=0
loop_body
    decf  i,f
    bra   loop_top
loop_end
     ...;rest of code...
```

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

int k,j;

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

```
movf  j,w       ;
andwf k,f       ;k = k & j, LSByte
movf  j+1,w       ;
andwf k+1,f     ;k = k & j, MSByte
```

d. ( 10 pts) Implement the 'FIND_CHAR' subroutine in PIC18 assembly language. The subroutine return value should be returned in the W register.

```
/* find first alpha char */
find_char (s)
unsigned char *s;
{
  unsigned char c;
  do {
    c= *s;
   if (c == 0) return(0);
   if (c > 0x40) return(c);
    s++;
  } while(1);
}

char *mystr = "  123  Hello";
main ()
{
   unsigned char a_char;

   a_char = find_char (mystr);

}
```

```
;memory space for find_char
CBLOCK 0x20
s:2, c     ; s contains pointer to character string, c is local variable
ENDC
```

```
    movff   s,FSR0L
    movff   s+1,FSR0H  ; FSR0 = s
loop
    movf  INDF0,w    ;
    movwf c          ;c = *s
    bz    exit       ; exit if zero
    sublw 0x40       ;0x40 – w
    bnc    exit    ;exit on c=0, borrow, so c > 0x40
    movf POSTINC0,w   ;s++
    bra   loop
exit
    movf  c,w        ; return c in W reg
    return
```

Common error: "c > 0x040" is an unsigned comparison, you cannot test N, V flags!!! Must check C flag!

```
; alternate solution
    movff   s,FSR0L
    movff   s+1,FSR0H  ; FSR0 = s
loop
    movf  POSTINC0,w ;get char, s++
    movwf c          ;c = *s
    bz    exit       ; exit if zero
    movlw 0x40       ;w = 0x40
    cpfsgt c         ; c > w?, skip bra if true
    bra   loop
exit
    movf  c,w
    return
```

e. (10 pts) Implement the *main()* code for the previous problem.

```
main
    movlw  low mystr
    movwf  s;
    movlw  high mystr
    movwf  s+1;
    call   find_char
    movwf  a_char
     ...rest of code...
```

f. (10 pts) Write a PIC18 assembly code fragment to implement the following:
signed int j, k;
unsigned char i;

```
if (j == k) {
     i = i >> 1;
        }
```

```
    movf  k,w       ;
    subwf j,w
    bnz   if_end     ;exit if LSBytes unequal
    movf  k+1,w
    subwf j+1,w
    bnz   if_end      ;exit if MSBytes unequal
    bcf   STATUS,C
    rrcf  i,f         ; i = i >> 1;
if_end
     ...rest of code ...
```

g. (5 pts) What is the value of j in HEX after execution of the following code?

```
signed char i, j;

 i = 0x80;
j = i >> 1;
```

```
   This is a signed char, so must maintain
sign bit on right shift.

  0x80 = 1000 0000 >> 1
       = 1100 0000
       = 0xC0        (-128/2 = -64)
```

h. ( 5 pts) Assume the following memory contents. Give the value of FSR0 and any changed memory locations after the code segment below is executed.

| Location | Contents: |
|---|---|
| 0x059 | 0xA8 |
| 0x05A | 0x30 |
| 0x05B | 0xF2 |
| 0x05C | 0x6E |

```
lfsr FSR0, 0x05B
decf PREINC0, f
```

```
   FSR0 = 0x5B  by 'LFSR' instruction

The 'PREINC0" says to increment FSR0 first
before using it, so new FSR0 = 0x5C

The instruction then becomes:
  decf 0x05C, f

so decrement contents of memory location
0x05C, or new value location 0x05C is 0x6D.

Final values: FSR0 = 0x5C
Location 0x5C =  0x6D
```

Part II: (30 pts) Answer 6 out of the next 9 questions. Cross out the 3 questions that you do not want graded.

1. When would you HAVE to use a **`call`** instruction instead of an **`rcall`** instruction?

   If target address of the rcall is further away than –1024 or +1023 instruction words, have to a use a CALL.

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

   Sign of decimal number is negative. Magnitude is 0x00 – 0xDF = 0x21 = 33
   Final answer = -33.

3. Give the value of –2 as a 16-bit two's complement number.

   -2 = 0xFE as an 8-bit number, as a 16bit number, sign extend and get 0xFFFE

4. Give the result of the operation 0x73 – 0xD0, and the V, N, C, Z flag settings.

   0x73 – 0xD0 = 0xA3. V=1, N=1, C = 0 (borrow), Z = 0

   V = 1 because +N – (-N) is same as (+N) + (+N), should give positive number. But result is negative, so overflow.

5. Give the result of the operation 0x40 + 0xA3, and the V, N, C, Z flag settings

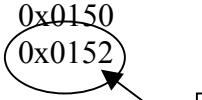
   0x40 + 0xA3 = 0xE3, V=0, N = 1, C = 0, Z = 0

   V =0 as +N + (-N) cannot produce overflow.

6.

7. What value is pushed on the stack by the rcall instruction below?

| Location: | Contents | Instruction |
|---|---|---|
| 0x0150 | DFD7 | rcall 0x0100 |
| 0x0152 | 2A7F | incf 0x07F,f |



Address of next instruction, 0x0152 is pushed on stack.

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

```
signed char i;

 i = 0x80;
while (i < 0) {
  i++;
 }
```

"i" is a signed char, as decimal number it is –128.
As i is incremented, it will eventually reach 0xFF (-1), then the next increment, will rollover to 0x00. At this point i is no longer less than 0, so loop is exited with value i = 0.

9. The TBLPTR register is used in table read operations. What is the size of this register and what is it used for?

It is used to hold the address of the program memory location being written or read; the size of the TBLPTR is 21 bits.

10. Define precisely what causes the stack ***underflow*** condition on the PIC18.

If STKPTR is 0, and a subroutine return is executed (this tries to pop a return address from an empty stack). This can happen you execute one more return statement than call statements.