You may NOT use a calculator. Assume the following memory/register contents at the beginning of each instruction:

Location        Contents:
0x035               0xA6
0x036               0x4B
0x037               0xFC
0x038               0x1D

a. (3 pts) In the code below, give the FINAL values of FSR0, FSR1 and the final value of any changed memory locations after the instruction sequence is executed.

```
lfsr            FSR0, 0x037
lfsr            FSR1, 0x035
movff        POSTDEC1, PREINC0
```

- FSR0 = 0x037 (load literal into FSR0)
- FSR1 = 0x035 (load literal into FSR1)
- POSTDEC1 means to access the contents of the memory location pointed to by FSR1, then decrement FSR1.
- PREINC0 means to increment FSR0 first, then access the contents of the memory location pointed to by FSR0.

So the ‘movff’ instruction does the following move:

```
movff      [0x035], 0x035     ; so location [0x035] → 0x035, or 0xA6 → 0x038.
```

The final results are:
- mem location 0x035 changed to 0xFC. Final value of FSR0 = 0x038. Final value of FSR1 = 0x034

b. (3 pts) In the code below, give the FINAL values of FSR0 and the final value of any changed memory locations after the instruction sequence is executed.

```
lfsr                FSR0, 0x35
movlw      3
movff        PLUSW0, INDF0
```

- POSTDEC1 means to access the contents of the memory location pointed to by FSR1, then decrement FSR1.
- PREINC0 means to increment FSR0 first, then access the contents of the memory location pointed to by FSR0.

So the ‘movff’ instruction does the following move:

```
movff      [0x035+W], 0x035     ; so location [0x035 + 3] → 0x035, or 0xA6 → 0x038.
```

Final result:
- mem location 0x035 changed to 0x1D. Final value of FSR0 = 0x038. Final value of FSR1 = 0x034

PLUSW0 means to access the contents of the memory location pointed to by [FSR0 + W].
INDF0 means to access the contents of the mem. location pointed to by FSR0. So the ‘movff’ instruction does the following move:

```
movff      [0x035+W], 0x035     ; so location [0x035 + 3] → 0x035, or 0x1D → 0x035
```

The PLUSW0 mode does NOT change the value of FSR0.

The final results are:
- mem location 0x035 changed to 0x1D. Final value of FSR0 = 0x035. Final value of W is 3.
c. (4 pts) Write the C function below in PIC18 assembly language. Assume that the value of iptr is PASSED into the function within the FSR0 register. What this code does is DECREMENT the value of the int pointed to by the pointer iptr.

```c
mysub (int *iptr) {
    (*iptr) --;
}
```

A solution:

```
movlw    0                           ; WREG = 0 so can do 16-bit decrement of [FSR0] - 1
decf     POSTINC0,f           ; decrement the LSByte of the INT pointed to by FSR0
    ; then have POSTINC0 increment FSR0 to point to MSByte of the INT
subwfb   INDF0,f                 ; subtract 0 - borrow from MSByte to complete 16-bit decrement
return
```

A common error:

```
decf       INDF0,f          ; decrement the value pointed to by FSR0
return
```

This would be correct if the subroutine was declared as below, where *iptr is a pointer to a CHAR type (8-bit value) instead of an INT type.

```c
mysub (char *iptr) {  // 'char' would make the above code correct
    (*iptr) --;
}
```

However, because in the original code iptr is a pointer to an INT type, you need to do a 16-bit decrement.

What if a CBLOCK had been used to pass the *iptr value like:

```
CBLOCK   0x????
    iptr: 2
ENDC
```

In this case, you would have had to write a bit more code in order to initialize FSR0 from the iptr contents:

```
movff     iptr, FSR0L
movff     iptr+1, FSR0H           ; FSR0 now contains the pointer passed in the iptr variable.
movlw     0                        ; WREG = 0 so can do 16-bit decrement of [FSR0] - 1
decf      POSTINC0,f             ; decrement the LSByte of the INT pointed to by FSR0
    ; then have POSTINC0 increment FSR0 to point to MSByte of the INT
subwfb    INDF0,f                 ; subtract 0 - borrow from MSByte to complete 16-bit decrement
return
```