Chapter 6. Problems

"All programming problems should include design pseudo code either as a separate design document on embedded comments in the code."

1S. Write an assembly code equivalent for the following C code segment:

```
int i;              //int is 2 bytes
i=0;
while (I < 35){
      i = i+2;
}
```

Solution

```
i0  equ  0x80                 // least significant byte of integer i
i1  equ  0x81                 // most significant byte of integer i

          CLRF     i0
          CLRF     i1
          MOVLW    35

w_loop:   CPFSLT   i0
          BRA      done
          BRA      cont

cont:     INCF     i0
          INCF     i0
          BRA      w_loop

done      ; continue with the rest of the program
```

1U. Write an assembly code equivalent for the following C code segment:

```
int i;      // int is 2 bytes
i=200;
while (i < 126){
      i = i-4;
}
```

Solution

2S. Write an assembly code equivalent for the following C code segment:

```
int i;              //int is 2 bytes
for (i=5; i≤29; i++)
      i = i+3;
```

Solution

```
i0  equ  0x80                 // least significant byte of integer i
i1  equ  0x81                 // most significant byte of integer i

MOVLW    5
MOVWF    i0
```

```
        CLRF    i1

        MOVLW   29

f_loop: CPFSGT  i0
        BRA     cont
        BRA     done

cont:   INCF    i0
        INCF    i0
        INCF    i0
        INCF    i0
        BRA     f_loop

done    ; continue with the rest of the program
```

2U. Write an assembly code equivalent for the following C code segment:

```
int i;              //int is 2 bytes
for (i=15; i≤129; i=i+2)
    i = i+6;
```

Solution

3S. Rewrite the following C code in assembly code and show the location of each variable in memory:

```
float fnum;
fnum = 200.625;
```

Solution

```
fnum    equ  0x80       ; 4 bytes

        ; convert 200.625 to single precision float → 11001000.101 → 1.1001000101x2^7
        ; 0100 0011 0100 1000 1010 0000 0000 0000
        ; Use indirect addressing to move the 4 bytes into to fnum
        LFSR    FSR0,0x80       ; Se the base
        MOVLW   0x00
        MOVWF   POSTINC0
        MOVLW   0xA0
        MOVWF   POSTINC0
        MOVLW   0x48
        MOVWF   POSTINC0
        MOVLW   0x43
        MOVWF   POSTINC0
```

3U. Rewrite the following C code in assembly code and show the location of each variable in memory:

```
float fnum;
fnum = 1280.625;
```

Solution

4S. Rewrite the following C code in assembly code and show the location of each variable in memory:

```
int  id;
char  grade;
float f_a[3];
struct  student{
        unsigned char name[40];
        int grade;
}  class [35];

class[3].grade = 100;
```

Solution

```
id      equ  0x80     ; 2 bytes
grade   equ  0x82     ; 1 byte
f_a     equ  0x83     ; 4 bytes each, 3 elements
class   equ  0x8F     ; 40 + 2 bytes each, 35 elements

        MOVLW    .100        // dot before the number indicated decimal
        MOVWF    0x135
        CLRF     0x136
```

4U. Rewrite the following C code in assembly code and show the location of each variable in memory:

```
int  id;
float f_a[5];
char  grade;
struct  student{
        unsigned char name[20];
        int grade;
}  class [15];

class[4].grade = 20;
```

Solution

5S. Write an assembly code segment using indirect addressing to implements the following C code segment:

```
char name1 [32], name2[32];
char equ_flag;
if (strcmp (name1, name2)
        equ_flag = 0;
else
        equ_flag = 1;
```

Solution

```
name1      equ  0x80     ; 32 bytes
name2      equ  0xA0     ; 32 bytes
equ_flag   equ  0xC0     ; 1 byte
i          equ  0xC1     ; 1byte

           LFSR     FSR0, name1    ; set base
           LFSR     FSR1, name2
           CLRF     i
```

```
                CLRF        equ_flg

cmp:            MOVF        POSTINC0, 0
                CPFSEQ      POSTINC1
                BRA         done            ; not equal
                INCF        i
                MOVLW       .31
                CPFSGT      i
                BRA         cmp             ; continue comparing
                INCF        equ_flg         ; strings are equal
done:
```

5U. Write an assembly code segment using indirect addressing to implements the following C code segment:

```
char apt1 [24], apt2[24];
char next;
if (strcmp (apt1, apt2))
        next = 0;
else
        next = 1;
```

<u>Solution</u>

6S. Write an assembly code segment using indirect addressing to implements the following C code segment:

```
char source[48], dest[48];
char I;

for (i=0; i<48 ; i++){
        dest[i] = source[i];
}
```

<u>Solution</u>

```
source          equ  0x80       ; 48 bytes
dest            equ  0xB0       ; 48 bytes
i               equ  0xE0       ; 1 byte

                LFSR        FSR0, source    ; set base
                LFSR        FSR1, dest
                CLRF        i

xfer:           MOVFF       POSTINC0, POSTINC1
                INCF        i
                MOVLW       .47
                CPFSGT      i
                BRA         xfer
```

6U. Write an assembly code segment using indirect addressing to implements the following C code segment:

```
char source[43], exch[43], temp;
int i;
```

```
i=0;
while (i<43){
        temp = exch[i];
        exch[i] = source[42 - i];
        source[42-i] = temp;
        i++;
}
```

Solution

7S. Using MPLAB and the following sample Bubble Sort C code, write equivalent PICmicro assembly code using indirect addressing.

```
void Bsort( int data[20] )
{
    int temp, i, swapped;
    swapped = 1;
    while (swapped == 1){
        swapped = 0;
        for (i=0;  i<20 ; i++){
            if (data[i] > data[i+1]){
                temp = data[i];
                data[i]= data[i+1];
                data[i+1]=temp;
                swapped = 1;
            }  // if()
        } // for()
    } // while()
} //Bsort()
```

Solution

```
            LFSR        0, 0x90             ; prep with first address
            LFSR        1, 0x91             ; prep with the address after the first
Loop2:      MOVF        INDF0, W            ; copy value at FSR0 to WREG
            CPFSGT      INDF1               ; compare WREG with FSR1
            BRA         NGT1                ; if !( (FSR0) > (FSR1) )
             ;swap the value of FSR0 is already cached in WREG
            MOVFF       INDF1, INDF0  ; move FSR1 to FSR0
            MOVWF       INDF1               ;move prev value of FSR0 into FSR1
            MOVF        POSTINC0            ; temp increment addr of FSR0
             ; time to reset the FSR1 address...
;            ; FSR0 only moves when FSR1 reaches the end and is never reset backwards
            MOVFF       FSR0L, FSR1L        ; copy address of FSR0 to FSR1
            MOVFF       FSR0H, FSR1H
            MOVF        POSTDEC0            ; decrement FSR0 back to it's proper value
            BRA Loop2
NGT1:       MOVF        POSTINC1            ; increment FSR1
            ; this is a specialized piece, for a general case we have to load in the end value
             ; and compare the low and high bytes
            MOVLW       0x95                ; move the end address into WREG
            CPFSEQ      FSR1L               ;compare with the ONLY the low byte
            BRA         Loop2               ;  not equal..continue
            ;is equal.. increment FSR0 and reset FSR1 to FSR0+1
            MOVF        POSTINC0            ; increment FSR0
            MOVF        POSTINC0            ; increment FSR0 (temp)
```

```
MOVFF     FSR0L, FSR1L     ; copy low-byte
MOVFF     FSR0H, FSR1H     ; copy high-byte
MOVF      POSTDEC0         ; decrement back to proper address
MOVLW     0x94             ; end address – 1
CPFSEQ    FSR0L            ;compare
BRA       Loop2            ; not equal .. continue
; all done and the 'array' is sorted largest to smallest
```

7U. Using MPLAB and the following sample Binary Search C code, write equivalent PICmicro assembly code using indirect addressing.

```
/*  Search for key in array A and return the location of key:
 *  A[] is a sorted array of elements
 *  imin and imax are limits of search range.
 */
int binary_search(int A[], int key, int imin, int imax)
{
  // test if array is empty
  if (imax < imin)
    // set is empty, so return value showing not found
    return (-1);   // search failed to find the key
  else
    {
      // calculate midpoint to cut set in half
      int imid = midpoint(imin, imax);

      // three-way comparison
      if (A[imid] > key)
        // key is in lower subset
        return binary_search(A, key, imin, imid-1);
      else if (A[imid] < key)
        // key is in upper subset
        return binary_search(A, key, imid+1, imax);
      else
        // key has been found
        return imid;
    }
}
```

<u>Solution</u>

8U. Write a C program that includes a function call, if statement, indirect addressing and a loop. Upon compilation, analyze the .lst file. The analysis should clearly relate the C statements and variables to their equivalent and location in the assembly code. Also, determine the final value assigned to every variable after the program execution.

<u>Solution</u>