# EE2361 Summer 2015, HW1 Due 06/26/2015 before midnight.

*NOTE: Please submit your homework electronically in a file (either pdf, word, open office) through moodle. I expect all code to have been run and tested in the MPLABX IDE environment before submission.*

1- Translate the following piece of code to assembly. List test cases that you used to test the correctness of your program.

outputVal is a 16-bit variable (presumably at address 0x800)  
 a, b, c are also 16-bit variables

outputVal = -1 ; the content of memory location 0x800 is -1  
 if (a+b < c ) AND (a >= 100) then  
 outputVal = 1;   
 else outputVal = 0;

|  |
| --- |
| .equ \_\_P24FJ64GA002,1 ;required "boiler-plate" (BP)  .include "p24Fxxxx.inc" ;BP  #include "xc.inc" ;BP  .global \_main    .section \*,address(0x800),data,near ;no padding, store as initialized data in near memory  .global outputVar  .global \_a  .global \_b ;using \_ prefix because a/b operands reserved for acca/accb registers for ADD  .global \_c  .align 2  outputVar: .word 0xFFFF ;variable "outputVar" set to -1, address 0x0800  \_a: .word 101  \_b: .word 3  \_c: .word 120  .text  \_main:  mov \_a, WREG  add \_b, WREG ;add b to w0 (a)  cp \_c ;compare c to a+b  bra GT, pass ;if c GT than a, goto pass  goto fail ;else goto fail  pass:  mov #100, w0  cp \_a ;compare a to 100  bra LT, fail ;if a <100, goto fail  mov #1, w0  mov w0, outputVar ;both conditions true, writes 1 to outputVar  goto end    fail:  mov #0, w0  mov w0, outputVar ;both conditions false, writes 0 to outputVar  end:  .end |
| First test: a=101,b=3,c=120. Expected result (1) was achieved  Second: a=99, b=3, c=120. Expected result (0) was achieved  Third: a=101, b=3, c=50. Expected result (0) was achieved |

2- Similar to the above problem, but this time the two conditions are OR instead of AND:

outputVal = -1 ; the content of memory location 0x800 is -1  
 if (a+b < c ) **OR** (a >= 100) then  
 outputVal = 1;   
 else outputVal = 0;

|  |
| --- |
| .equ \_\_P24FJ64GA002,1 ;required "boiler-plate" (BP)  .include "p24Fxxxx.inc" ;BP  #include "xc.inc" ;BP  .global \_main    .section \*,address(0x800),data,near ;no padding, store as initialized data in near memory  .global outputVar  .global \_a  .global \_b ;using \_ prefix because a/b operands reserved for acca/accb registers for ADD  .global \_c  .align 2  outputVar: .word 0xFFFF ;variable "outputVar" set to -1, address 0x0800  \_a: .word 101  \_b: .word 3  \_c: .word 120  .text  \_main:  mov \_a, WREG  add \_b, WREG ;add b to w0 (a)  cp \_c ;compare c to a+b  bra GT, pass ;if c GT than a, goto pass  mov #100, w0 ;else perform other condition  cp \_a ;compare a to 100  bra LT, fail ;if a <100, goto fail    pass:  mov #1, w0  mov w0, outputVar ;both conditions true, writes 1 to outputVar  goto end    fail:  mov #0, w0  mov w0, outputVar ;both conditions false, writes 0 to outputVar    end:  .end |
| First test: a=101,b=3,c=120. Expected result (1) was achieved  Second: a=101, b=3, c=0. Expected result (1) was achieved  Third: a=99, b=3, c=0. Expected result (0) was achieved |

3- Write a program to perform a 48-bit addition. The first 48-bit number is at address 0x800-0x805 (least significant byte first), and the second 48-bit number is at 0x806-0x80B. The result should be stored at w10-w12.

|  |
| --- |
| .equ \_\_P24FJ64GA002,1 ;required "boiler-plate" (BP)  .include "p24Fxxxx.inc" ;BP  #include "xc.inc" ;BP  .global \_main    .section \*,address(0x800),data,near ;no padding, store as initialized data in near memory  .global var1  .global var2  .align 2  var1: .word 0xFFFF,0xFFFF,0x00FF ;0x0800-0x805  var2: .word 0x0001,0x0000,0x0000 ;0x806-0x080B  .text  \_main:  mov 0x0800, wreg ;lsw of var1  add 0x0806, wreg ;add lsws of var1/var2  mov w0, w10 ;move lsw sum to w10    mov 0x0802, wreg ;does same for middle words  addc 0x0808, wreg  mov w0, w11    mov 0x0804,wreg ;does same for msw  addc 0x080A, wreg  mov w0, w12  .end |
| The first variable was set to 0x00FFFFFFFFFF and the second 0x000000000001, with the addition expected to produce a carry between each adjacent word. The test result was found to be 0x010000000000 as expected, showing carries are handled. |

4- Consider the following assembly program.

1. Write program memory contents next to each line (do manual conversion using the prog ref manual, but OK to use MPLAB to verify).
2. Show calculations for the destination address in the bra and goto instruction.
   1. bra: (pc+2) +2\*slit16= (0x0206+0x2)+ 2\*slit16=0x202 → slit16 = -3 = 0xFFFD
   2. goto: address (nn...nn) = 0x0200 = 100000000 (last 0 ignored)
3. Show what fields in the instruction opcode correspond to W5 and ++ and [] in the instruction addc w3, #22, [w5++]
   1. W5 is in yellow (last nibble of opcode), the green is the addressing mode [indirect with post increment)

0x0200 L1: mov #123, w2 ; 0010 0000 0000 0111 1011 0010

0x0202 L2: dec w2 ; 1110 1001 0000 0001 0000 0010

0x0204 addc w3, #22, [w5++]; 0100 1001 1001 1010 1111 0110

0x0206 bra nz, L2 ; 0011 1010 1111 1111 1111 1101

0x0208 goto L1 ; 0000 0100 0000 0000 0000 0100

; 0000 0000 0000 0000 0000 0000

5- An array of 10 unsigned 16-bit numbers is stored at address 0x810. Write a program that uses indirect addressing to find the smallest number in the list. Store the value of the smallest number at address 0x800.

|  |
| --- |
| .equ \_\_P24FJ64GA002,1 ;required "boiler-plate" (BP)  .include "p24Fxxxx.inc" ;BP  #include "xc.inc" ;BP  .global \_main    .section \*,address(0x0810),data,near ;no padding, store as initialized data in near memory  .global array  .align 2  array: .word 0x0123,0x0473,0x0638,0x0101,0x0011,0x0992,0x0377,0x0093,0x0993,0x0999  .text  \_main:  mov #0x0810, w1 ;adress checking  mov 0x0810, w2 ;current lowest  mov #0x0822, w3 ;last address    loop1:  cp w1, w3  bra GE, end  cp w2, [++w1]  bra GT, loop2  goto loop1    loop2:  mov [w1], w2  goto loop1  end:  mov w2, 0x0800  .end |
| The array listed was used first, and the lowest value (0x0011) was correctly stored at 0x0800. The last item in the array was then changed to 0x0001, and 0x0000 was added onto the array (at position 11). The program correctly determined 0x0001 (the 10th element) to be the lowest. |

6- A list of ten 16-bit numbers is stored at address 0x800. Write a program to calculate the sum of the absolute difference between consecutive numbers and store it as a 16-bit number in w5. For example, for a list of four numbers 5, 9, 2, 4, the sum is |5-9| + |9-2| + |2-4| = 4+7+2 = 13. Use indirect addressing.

|  |
| --- |
| .equ \_\_P24FJ64GA002,1 ;required "boiler-plate" (BP)  .include "p24Fxxxx.inc" ;BP  #include "xc.inc" ;BP  .global \_main    .section \*,address(0x0800),data,near ;no padding, store as initialized data in near memory  .global array  .align 2  numlist: .word 0x3,0x2,0x3,0x4,0x5,0x6,0x7,0x8,0x9,0xA ;list of 10 words  .text  \_main:  mov #0x0800 w1 ;start address of data  mov #0x0812 w2 ;end address of data  clr w3      main\_loop:  mov [w1], w0  sub w0,[++w1],w0 ;subtract # from adress w1+1 from # at adress in w1  btsc 0x0042, #3 ;bit test negative bit of SR  neg w0,w0 ;if negative, convert to positive value  add w0,w3,w3 ;accumalte the differences  cp w1,w2 ;test if reached end of list  bra GE end ;if so end  goto main\_loop ;otherwise continue    end:  .end |
| The list of numbers in the program was used. This list results in positive and negative differences. The resulting sum of abs. value of differences was correctly found to be 9, |

7- Write a program (in Assembly) to convert a string to uppercase. For example if the string is at address 0x800 and contains the characters in “ApPle%4”, it should be changed to “APPLE%4” (i.e., overwrite the contents of address 0x800).

|  |
| --- |
| Paste your code here, including variable definitions: |
| How did you test your program? Be specific, and list all cases you tested for and what variable values you used for each case. Include a screen shot of mplabx showing that your program works (one test case is enough). |