Electrical and Computer Engineering Department
University of Maryland
College Park, MD 20742-3285

Glenn L. Martin Institute of Technology ♦ A. James Clark School of Engineering

ENEE 350 Homework Problem Set 8
Programming Project 3
(Due: Class 21, Mon., Jul. 7, 2008)

Dr. Charles B. Silio, Jr.
Telephone 301-405-3668
Fax 301-314-9281
silio@umd.edu

The MAC-1 Instruction Repertoire has been extended with the addition of four new instructions and this new Instruction Set Architecture is called the MAC-2. The MAC-2's Instruction Repertoire is attached, as is the corresponding microprogram that fetches, decodes and executes these MAC-2 instructions. This microprogram's 32-bit binary words reside in the file **mpc1**, and the corresponding MAL source code resides in the file **mpc1.pascal**. These two ascii text files reside in the course webpage, URL: "**http://www.ece.umd.edu/class/enee350-1.Sum2008/Homework/**".

A program to test the new MAC-2 instructions also resides there called **tst**, and the new opcodes table used by the assembler (*assem*) is in the file **tst.opcodes_new** also on this webpage. Use an internet browser and go to the course webpage and save these files. You should first copy these four files into the working directory that you will use to accomplish this programming assignment. That way *assem* and *sim* can look for them there.

For each subprogram (subroutine) which uses the new MAC-2 opcodes that you wish to assemble using *assem*, (e.g., in files **prg3sub1** and **prg3sub2**) first copy **tst.opcodes_new** into **prg3sub1.opcodes_new** and into **prg3sub2.opcodes_new**, respectively. Recall that *assem* looks for the new opcodes table under the same file name as the program being assembled but with the ".opcodes_new" file name extension appended to the name.

1. Write & test a procedure (i.e., a function subprogram) **inv(x)** to compute the bit-wise logical complement (i.e., the 1's complement) of the argument (passed on the stack by reference) and returned by value in the ac register. (Recall DRC(x) = RC(x) - 1, and x is a 16-bit word.)

2. Making use of the new MAC-2 instructions and the inv(x) function, write and test a procedure (i.e., a function subprogram) **xor(x,y)** that computes the bit-wise logical exclusive-OR of the n-tuples x and y. Again, the arguments are passed by reference, with address y pushed on the stack first followed by address x pushed on the stack followed by a call to function xor, which returns the value computed in the ac register (return by value). Recall $x \oplus y = x' \cdot y + x \cdot y' = [(x' \cdot y)' \cdot (x \cdot y')']'$. (Assume that ac and f are volatile registers; i.e., only the calling program needs to save and restore them if they contain values to be preserved, so the called routines can use them as scratch registers. Also, n=16, so that x and y are 16-bit words.)

3. Making use of the new MAC-2 instructions and the xor(x,y) function, write and test a procedure (i.e., function subprogram) **hd(x,y)** that computes the Hamming Distance between the two binary n-tuples x and y. Again, the arguments are passed by reference, with address y pushed on the stack first followed by address x pushed on the stack followed by a call to function hd, which returns the value computed in the ac register (return by value). (Recall that Hamming Distance can be computed by counting the number of one's in the n-tuple z, where the difference n-tuple z = xor(x,y); again assume that n=16 for 16-bit words x and y.)

4. Test your functions with the following main program (called "prg3main") and its data. This program is located on the next page and the source code can be copied from the course webpage for your convenience. Turn in printouts of the separate assemblies, linkage, and the snapshots of memory before and after execution with simulator *sim*. Assuming that the absolute program after linking is **prg3main.abs**, then you must call *sim* with the following statement: "**sim prg3main.abs mpc1 mpc1.pascal**". Highlight and explain the final contents of the locations containing the test results. Be sure to comment cogently your code.

see next page for prg3main

```
          /prg3main
           EXTRN   inv
           EXTRN   xor
           EXTRN   hd
ans1       RES      1
ans2       RES      1
ans3       RES      1
ans4       RES      1
ans5       RES      1
ans6       RES      1
ans7       RES      1
x1         0x6A34
x2         0x3CD1
begin      loco    4020
           swap
           loco    x2
           push
           call    inv
           stod    x2
           loco    x1
           push
           call    inv
           stod    x1
           call    xor
           stod    ans1
           call    hd
           stod    ans2
           loco    x3
           stol    1
           call    xor
           stod    ans3
           call    hd
           stod    ans4
           loco    x2
           stol    0
           call    xor
           stod    ans5
           call    hd
           stod    ans6
           loco    x2
           stol    1
           call    xor
           stod    ans7
           insp    2
           halt
x3         0x7E0B
           END     begin
```