ENEE 350 Homework Set No. 9

and

Programming Assignment 4
(Due: Class 21, Thurs., Apr. 14, 2005)

1. Read Appendix B of text by A. Tanenbaum, *Structured Computer Organization*, 4th ed., Prentice-Hall, 1999, and work the following problems from Appendix B:
   a. Problem B-1.
   b. Problem B-2.
   c. Problem B-3.

2. What sign-magnitude decimal values are represented by the following IEEE 754 single precision floating-point words whose contents are shown using hexadecimal shorthand? (Hint: use C-compiler and formatted output to save yourself from doing considerable work.)
   a. B9EEDF8A
   b. 7F800000
   c. 40490FDB
   d. FFD3F03


4. The designers of a particular computer have decided that the computer must be capable of representing single-precision (single-word) floating-point numbers in the range ±(10^{-17} to 10^{17}) with a precision of one part in 10^9. Determine the minimal binary word length which must be chosen for this machine, and indicate the floating-point format you would choose for doing this in order to facilitate the sorting of floating-point numbers. (Assume that 2^{10} = 10^3 to facilitate decimal to binary conversions.)

5. Recall from your reading of Silio’s notes on floating-point representations that the UNIVAC 1100 series computers have 36-bit words and perform 1’s complement arithmetic. Suppose UNIVAC 1100 registers A1 and A2 contain the following bit patterns in octal shorthand.

   \[(A1) = 57205377777\quad \quad (A2) = 206556400000\]

   Viewing the contents of A1 and A2 as single-precision floating-point numbers:
   a. What sign-magnitude decimal number is contained in A1?
   b. What sign-magnitude decimal number is contained in A2?

6. In a DEC PDP-11 the contents of two consecutive memory words are (in binary):

   \[
   \begin{array}{c|c}
   10111111111010000 & 00000000000000000
   \end{array}
   \]

   Recall that the single-precision floating-point format for this machine is of the form: 1+8+23(24) bits with a binary normalized mantissa 0.1xxx as in

   \[
   \begin{array}{c|c|c}
   S_M & \text{BIASED} & \text{BINARY NORMALIZED} \\
   \text{EXPONENT} & \text{MANTISSA} \\
   \hline
   1 & 8 & 23
   \end{array}
   \]

   If this 32-bit pattern is interpreted as a single-precision floating-point number, what sign-magnitude decimal number does it represent?

   \(-\text{continued}-\)
7. The IBM 360/370 series computers use a sign-magnitude hexademically normalized, biased-exponent, 32-bit representation for single precision floating-point numbers in the following format:

<table>
<thead>
<tr>
<th>S_M</th>
<th>BIASED EXPONENT</th>
<th>HEXADEMICALLY NORMALIZED MANTISSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>24</td>
</tr>
</tbody>
</table>

Write the 8-digit hexadecimal representation of the bit pattern in the 32-bits known to contain the single-precision representation of the following floating-point number shown here in both its decimal and octal forms:

\[-(\frac{27}{13})_{10} = -(33.1166116611661166...)_8\]

8. Consider the following biased exponent (bias = 2^n), sign-magnitude floating point format for representing binary normalized numbers in single-precision words in a machine with 2’s complement fixed-point arithmetic; the mantissa (significand) is a binary normalized fraction, and there are no hidden bits:

<table>
<thead>
<tr>
<th>S_M</th>
<th>BIASED EXPONENT</th>
<th>BINARY NORMALIZED MANTISSA</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

Suppose we are given the following two operands represented in this format:

\( X = 1000010 1010001 \quad Y = 0001011 1100110 \)

Show the bit pattern in the single-precision word \( S \) that results from the floating add of the contents in \( X \) and \( Y \), assuming that the result is truncated to a 7-bit precision fraction.

9. Programming Assignment 4: Consider the following biased exponent (bias = 2^n), sign-magnitude floating point format for representing binary normalized numbers in 16-bit single-precision words in a machine with 2’s complement fixed-point arithmetic; the mantissa (significand) is a binary normalized mixed number with hidden bit similar to IEEE754:

<table>
<thead>
<tr>
<th>S_M</th>
<th>BIASED EXPONENT</th>
<th>BINARY NORMALIZED SIGNIFICAND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>7</td>
<td>8</td>
</tr>
</tbody>
</table>

For example, the following two operands represented in this format:

\( A = 10000010 10100011 \quad B = 00001011 11001100 \)

where \( A = 0x82A3 = -1.10100111\times2^{-62} \) and \( B = 0x5CC = +1.10011100\times2^{-59} \).

a. Making use of the MAC-2 instruction repertoire and the \( \text{inv}(x) \) function you wrote and tested in programming assignment 3, write and test a procedure (i.e., a function subprogram) \( \text{or}(x,y) \) that computes the bit-wise logical OR of the n-tuples \( x \) and \( y \). 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 \( \text{or} \), which returns the value computed in the ac register (return by value).

b. Making use of the MAC-2 instruction repertoire, write a (void function) procedure \( \text{ashr}(x) \) that performs a 1-bit position arithmetic (algebraic) right shift of the contents of memory location \( x \) and leaves the result in memory location \( x \), where the address \( x \) is passed by reference on the stack.

c. Again, making use of the MAC-2 instruction repertoire and whatever other functions (such as the OR function and procedure \( \text{ashr}(x) \) from parts a) and b) write and test a procedure (i.e., a function subprogram) \( \text{fadd}(x,y) \) that performs a floating add of single-precision floating point numbers in memory locations \( x \) and \( y \) and returns the single-precision floating-point format result in the ac register, where all single-precision floating point numbers are represented in the format specified above in Problem 9. 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 \( \text{fadd} \), which returns the value computed in the ac register (return by value).

c. Test your \( \text{fadd} \) function using the following main program (\text{prg4main}):
Repair the following main program, if necessary, to accomplish the desired results as stated in the comments.

/prgmain
EXTRN inv
EXTRN or
EXTRN fadd

x1 0x7B5C
x2 0x7A33
x3 0x0b98
x4 0x02A3
ans1 RES 1
ans2 RES 1
ans3 RES 1
ans4 RES 1
ans5 RES 1
ans6 RES 1

start loco 4020
swap
locos x1
push
call inv
stod x1 /create data x1=0x82A3
stod ans1
locos ans1
push
call ashr /make sure ashr is working
insp 1

locos x2
push
call inv
stod x2 /create data x2=0x85CC
call or
stod ans2 /make sure OR is working
call fadd
stod ans3 /ans3=fadd(x1,x2)
locos x3
stol 0
call ashr /ashr shifts x3 right arithmetically
call fadd
stod ans4 /ans4=fadd(x1,x3)
locos x4
stol 1
call fadd
stod ans5 /ans5=fadd(x3,x4)
locos x2
stol 0
call fadd
stod ans6 /ans6=fadd(x2,x4)
insp 2
halt
END start