Assignment 2

Parts 1 and 2 are due in the dropbox on mycourses Friday, 04/06/12 @10pm.

Parts 3 and 4 are due in the dropbox on mycourses Tuesday, 04/10 @10pm.

1) (10 points) In the scoreboard example shown in class, switch the MUL and DIV instructions, including their operands and destination. Starting in the same state as in the example in class, and accounting for the necessary changes in this initial state due to the changes in the code, show:

    a. The scoreboard in this initial state,
    b. The scoreboard when MUL is ready to proceed to "Write Result",
    c. The scoreboard when DIV is ready to proceed to "Write Result".

Use the scoreboard template posted on mycourses, and the latencies used in the class example.

2) Using TomasuloTemplate.doc (20 points) and TomasuloSpeculationTemplate.doc (20 points), simulate the following code:

```
        L.D     F6, 34(R2)
        L.D     F2, 45(R3)
        MUL.D   F0, F2, F4
        BNZ     F0, Label
        SUB.D   F8, F6, F2
        DIV.D   F10, F0, F6
Label:  ADD.D   F6, F8, F 2
```

Assume the following total execution latencies:

- ADD/SUB Integers 2
- ADD/SUB Float 4
- MUL Integers 8
- MUL Floats 10
- DIVIDE Integers 36
- DIVIDE Floats 40

Note: You can use faster than light computing during cycles in which nothing changes.

3) (10 points) Read the two papers posted on mycourses, and submit a ~500 words or less (~<1page) summary for each one. The third paper is FYI, i.e. optional. Address the following:
    a. What is the paper talking about
    b. What did you find out new that you didn't know before
    c. What you agree and/or disagree
    d. What has history proven right or wrong since the paper was written

4) (40 points) Project 2:
    a. For the processor and instruction set created in project 1, create an assembler program using your preferred high level programming language.
    b. Recommendations:

i. The assembly code file will contain one instruction per line, ending with a <;>, after which comments can be added, which will be ignored by the assembler.
ii. You can choose the code to be case sensitive or not.
iii. Labels start in the first column only. If there is a space, then there is no label.
iv. Mnemonics cannot start in the first column.
v. Mnemonics can be identified by reading in one character at a time, or reading a character string until the next space character is encountered. There is a space between the last character of the mnemonic and the first character of the next alphanumeric symbol.
vi. The alphanumeric symbols following the mnemonic are separated by a comma, optionally with a space following it.
vii. Parsing can be implemented by reading an entire line up to the <;> into a string variable, and then converting it to machine code by analyzing using *if-then-else* or *switch* statements into machine code.
viii. If an unknown symbol or syntax is encountered, the assembler will exit immediately printing to the terminal the line number it has found an error on.
ix. Once you have implemented the basic functionality of the assembler, you can add directives to specify the locations of the code, and/or initialized data structures, such as arrays.

c. To test your processor and assembler, convert the following code to assembly:

```
for (i=16; i>0; i=i–1)
        x[i] = x[i] + s; where s is a scalar value of your choice
```

This is similar to the code used in the loop unrolling example in class, except that your array elements are integer bytes.

d. Now, assemble and run it. Collect the following runtime information:
   i. Total number of instructions executed
   ii. Total number of pipeline cycles executed – may not necessary equal your clock period
   iii. Total number of pipeline stalls
   iv. Total number of use of each instruction type
   v. Total number of use of each addressing mode
e. Unroll the original loop iteration four times. Assemble, run, and collect the same information as in part d.
f. Submit:
   i. A “user manual” for your assembler.
   ii. The code for the assembler.
   iii. The executable code and the necessary files to test it, and reproduce your experimental work in parts d and e.
   iv. Using excel or your favorite spreadsheet program, capture the data collected in parts d and e in a single bar graph, comparing each metric side by side for the two runs.