ENEE 446: Digital Computer Design

Fall 2018 **Handout #12**

Problem Set # 2 **Due: October 10**

ILP

Problem 1

H&P 3.5

Problem 2

H&P 3.6

Problem 3

H&P 3.7

Problem 4

In this problem, we will look at how a common vector loop runs on two processors, one that implements scoreboarding and another that implements Tomasulo's algorithm. The loop is the so-called SAXPY loop and is the central operation in Gaussian elimination. The loop implements the vector operation Y = a * X + Y for a vector of length 100. The code is shown below:

```
foo:  L.D      F2, 0(R1)      ; load X[i]
      MUL.D    F4, F2, F0     ; multiply a*X[i]
      L.D      F6, 0(R2)      ; load Y[i]
      ADD.D    F6, F4, F6     ; add a*X[i] + Y[i]
      S.D      F6, 0(R2)      ; store Y[i]
      DADDUI   R1, R1, #8     ; increment X index
      DADDUI   R2, R2, #8     ; increment Y index
      DSGTUI   R3, R1, #800   ; test if done
      BEQZ     R3, foo        ; loop if not done
```

You will be asked to specify the state of the processor after some number of instructions from the SAXPY loop have executed. When doing this, use the structure on page 1 of handout #10 for scoreboarding, and the structure on page 2 of handout #10 for Tomasulo's algorithm. Before drawing the processor state, you must first draw the pipeline diagram. In order to do this, you will need some rules about how to draw the pipeline diagrams. These rules are given at the end of this problem.

a. Using the SAXPY code, show the state of the scoreboard when the DSGTUI instruction reaches write result. Assume that there is one integer functional unit. Assume the FP unit configuration of H&P Figure C.54 with the FP latencies of H&P Figure 3.2 (see below for modifications to these latencies). The branch should not be included in the scoreboard.

b. Use the SAXPY code and a fully pipelined FPU with the latencies of H&P Figure 3.2 (see below for modifications to these latencies). Assume Tomasulo's algorithm for the hardware with one integer unit. Show the state of the reservation stations and register file ready bits when the DSGTUI writes its result on the CDB. Do not include the branch.

c. Using the SAXPY code, assume a scoreboard with the FP functional units described in H&P Figure C.54, plus one integer functional unit (also used for load-store). Assume the latencies shown below in the table (see below for modifications to these latencies). Show the state of the scoreboard when the branch issues for the second time. Assume the branch was correctly predicted taken and took 1 cycle. How many clock cycles does each loop iteration take? You may ignore any register port/bus conflicts.

Instruction producing result	Instruction using result	Latency in clock cycles
FP multiply	FP ALU op	6
FP add	FP ALU op	4
FP multiply	FP store	5
FP add	FP store	3
Integer operation (including load)	Any	0

d. Use the SAXPY code. Assume Tomasulo's algorithm for the hardware using one fully pipelined FP unit and one integer unit. Assume the latencies shown in the above table from part c. (see below for modifications to these latencies). Show the state of the reservation stations and register file ready bits when the branch is executed for the second time. Assume the branch was correctly predicted as taken and took 1 cycle. How many clock cycles does each loop iteration take?

As mentioned earlier, you must draw the pipeline diagrams before specifying the state of the processor. Here are some rules that will help you draw the pipeline diagrams. For all the rules, assume all units are fully pipelined. Also, assume there is no forwarding in scoreboarding, but there is forwarding in Tomasulo.

1. For scoreboarding, issue and register read happen on two separate cycles. For instance, the DADDUI instruction would look like the following:

```
F D I R E W
```

And the store double instruction would look like:

```
F D I R M W
```

Also, in scoreboarding, WAR hazards are detected and stalled during the writeback stage, as discussed in class.

2. For Tomasulo, register read happens in the same cycle as issue. For instance, the DADDUI instruction would look like the following:

```
F D I E W
```

And the store double instruction would look like:

```
F D I M W
```

3. H&P Figure 3.2 gives the *latencies* of various instructions. The latency is the number of intervening cycles required between two dependent instructions:

```
MUL.D f4, f2, f0
-
-
-
ADD.D f6, f4, f6
```

means that you need the following pipeline diagram for scoreboarding:

```
MUL.D      F D I R E E E W

ADD.D        F D I - - - R E E E W
```

because there is no forwarding in scoreboarding. For Tomasulo, you would have:

```
MUL.D      F D I E E E E W

ADD.D        F D I - - - E E E E W
```

4. In H&P Figure 3.2, the book claims FPop → Store has latency 2 cycles. Change this to latency 3 cycles. This is an inconsistency that cannot be rectified if you assume that FPop → FPop is 3 cycles:

```
MUL.D      F D I R E E E W

S.D          F D I - - - R M W
```

5. For scoreboarding, a scoreboard slot does not free up until the instruction in the slot enters writeback.

```
DADDUI        F D I R E W

DADDUI          F D - - I R E W
```

6. For Tomasulo, create enough reservation stations such that you never have any structural hazards.

7. For parts a. and b., you are to use the latencies in H&P Figure 3.2. However, in this figure, the LOAD $\rightarrow$ FPop latency is specified as 1 cycle. This is correct for scoreboarding, but for Tomasulo, the LOAD $\rightarrow$ FPop latency should be 0 cycles because there is forwarding (in scoreboarding, there is no forwarding, thus the stall):

```
L.D          F D I M W

MUL.D          F D I E E E E W
```

8. For parts c. and d., you are to use the latencies given in the table in part c. However, in this table, the LOAD $\rightarrow$ FPop latency is specified as 0 cycles. This is correct for Tomasulo, but for scoreboarding, the LOAD $\rightarrow$ FPop latency should be 1 cycle because there is no forwarding (in Tomasulo, there is forwarding, thus there is no stall):

```
L.D          F D I R M W

MUL.D          F D I - R E E E E E E W
```