Homework Assignment 5
CIS501 Fall 2005

Due: Wednesday, December 14*th*, 2005 at 7:00 PM.

Instructions: This assignment serves as both the questions and the worksheet. You should answer all of the questions on this assignment/worksheet, except for the graph on Question 3, which should be drawn via computer on a separate sheet of paper.

Question 1 (10 points):

In the previous assignment we applied loop unrolling to increase the amount of ILP available in a statically scheduled machine. In this assignment, we will explore the execution of code on a dynamically scheduled (or out-of-order) processor implementation.

Consider the following code:

```
start:
#1 ldf [r1] -> f1
#2 mulf f1 * f2 -> f3
#3 stf f3 -> [r2]
#4 ld [r1+4] -> r1
#5 ld [r2+4] -> r2
#6 bz r1, done
#7 bnz r2, start
```

Complete the pipeline diagram for two iterations of this loop for an out-of-order pipelines. The machine has 1-cycle load-use penalty loads, 3-cycle floating point multiplications (mulf), 2-cycle floating point adds (addf), and all other operations take one cycle. The pipeline is two-way superscalar (i.e. it can fetch, dispatch, issue, and retire two instructions per cycle), has full bypassing, has fully pipelined functional units (for multicycle operations). Assume that the processor can writeback any number of instructions per cycle, and that all branches are predicted correctly. Also, the processor is capable of executing two loads and retiring one store in a cycle. Stores require one cycle for address generation (`X`), and a second cycle to write their value into the store queue (`M`). Stores need their address input for the `X` stage, but do not need their data input until the `M` stage.

(a). Complete the pipeline diagram assuming the processor implements **conservative** load scheduling.

(b). Complete the pipeline diagram assuming the processor implements **optimistic** load scheduling. Assume that no memory ordering violations occur.

	Question 1, part (a)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
#1	ldf [r1] → f1	F	D	I	X	M	W	R														
#2	mulf f1 * f2 → f3	F	D			I	X	X	X	W	R											
#3	stf f3 → [r2]		F	D				I	X	M	W	R										
#4	ld [r1+4] → [r1]		F	D					I	X	M	W	R									
#5	ld [r2+4] → [r2]			F	D				I	X	M	W	R									
#6	bz r1, done			F	D						I	X	W	R								
#7	bnz r2, start				F	D					I	X	W	R								
#1	ldf [r1] → f1																					
#2	mulf f1 * f2 → f3																					
#3	stf f3 → [r2]																					
#4	ld [r1+4] → [r1]																					
#5	ld [r2+4] → [r2]																					
#6	bz r1, done																					
#7	bnz r2, start																					

	Question 1, part (b)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
#1	ldf [r1] → f1	F	D	I	X	M	W	R														
#2	mulf f1 * f2 → f3	F	D			I	X	X	X	W	R											
#3	stf f3 → [r2]		F	D				I	X	M	W	R										
#4	ld [r1+4] → [r1]		F	D	I	X	M	W				R										
#5	ld [r2+4] → [r2]			F	D	I	X	M	W				R									
#6	bz r1, done			F	D		I	X	W				R									
#7	bnz r2, start				F	D		I	X	W				R								
#1	ldf [r1] → f1																					
#2	mulf f1 * f2 → f3																					
#3	stf f3 → [r2]																					
#4	ld [r1+4] → [r1]																					
#5	ld [r2+4] → [r2]																					
#6	bz r1, done																					
#7	bnz r2, start																					

Question 2 (10 points):

A two processor system uses an MSI cache-coherence protocol. The processors use 16-bit addresses, and each processor has an 16-byte direct mapped cache with two 8-byte blocks. In the space provided below, the first column shows which processor is taking an action (i.e., processor 0 or processor 1), and the second column indicates what operation that processor takes (i.e. **Ld** or **St**. The third column is the address of the operation. For each operation, fill in the next four columns with the state of each processor's cache after the action is concluded—the state should be written as the MSI state, followed by the tag (except for I, which needs no tag). For example, `M:123` means "Tag 123 in the Modified state". All addresses are given as hexadecimal numbers. Finally, in the last column, categorize the result as one of the following:

- Hit
- Upgrade miss
- Compulsory miss
- Conflict miss
- Capacity miss
- Coherence miss

CPU#	Op	Addr	CPU 0's cache		CPU 1's cache		Result
			Line 0	Line 1	Line 0	Line 1	
Initial State:			S:000	M:001	S:000	M:111	–
0	St	`0004`					
0	St	`0F0F`					
1	Ld	`0000`					
1	St	`111F`					
0	Ld	`0018`					
0	St	`0F08`					
0	Ld	`0004`					
1	Ld	`111F`					
0	St	`0004`					
1	Ld	`0000`					

How might the results change if the block size was halved (to 4-byte blocks)?

Question 3 (5 points): Pretend you're a designer looking to create the a next-generation processor. The current-generation processor is 4-way superscalar with a 32-entry instruction window. Given the choice, which change would improve performance more: (a) increasing the instruction window size from 32 to 128 or (b) increasing the width of the processor from a 4-issue to a 8-issue pipeline?

To answer this question, download the timing simulator (sim-R10K) from `~cis501/SimpleScalar/hwk5/` and build it with `make`. There is a script provided (`run.sh`) that will prompt for window size, pipeline width, and benchmark name and then run the simulator with the corresponding parameters. Run twolf and vpr.route for both 4-wide and 8-wide pipelines at window sizes of 32, 64, and 128. Fill in the following table with the IPCs of each run (each entry is `width x window`, i.e., `4 x 64` is for the 4 wide machine with a window size of 64).

	twolf	**vpr.route**
4 x 32		
4 x 64		
4 x 128		
8 x 32		
8 x 64		
8 x 128		

Next, draw a line graph with IPC as the y-axis and window size as the x-axis. Finally, answer the question from above in the space below: