

Homework 2

Prof. Loh

CS6290 - Spring 2007

Handed Out: 05 Feb 2006 (Mon)

Due: 16 Sep 2006 (Fri - by email)

For problems 1-3, unless otherwise specified, make the following assumptions:

- One instruction can issue (allocate) per cycle
- Latencies: ADD (1), LOAD(4), MUL(8), DIV(20)
- There is one adder, one multiply/divide unit, one load port
- Functional units are *not* pipelined
- Write Result and Exec occur in the same cycle (i.e., an instruction writes to the CDB in cycle n , and its dependent instruction can start execution in cycle n as well)
- There is a single CDB
- Reservation stations: ADD(3), LD(2), MD(2); call these AD1-AD3, LD1-LD2, MD1-MD2
- An instruction stalls at issue if an appropriate reservation station (RS) is not available. It may issue in the cycle *after* the previous instruction writesback (i.e., if ADD X is stalled because all add RS's are occupied, and then one of the ADD's writes back on cycle n , then the ADD X can issue on cycle $n+1$).
- If more than one reservation station is available (e.g., ADD1 and ADD3), issue the instruction to the RS with the lower number (e.g., ADD1).

3. Superscalar Out-of-Order

Repeat the previous exercise using the same assumptions except with the following differences:

- There is a unified set of six reservation stations (RS1-RS6).
- Don't worry about renaming for this problem (it would still happen, but you don't need to show the details).
- Instructions can issue *two* per cycle.
- Instructions can commit *two* per cycle.
- Still only *one* CDB.
- Separate MUL and DIV units. (One of each can execute in parallel.)

Instruction		Issue	Execute	Writeback	Commit	Comments
DIV	R1 = R2/R3	1	2-21	22	23	
LOAD	R4 = 0[R5]	1				
MUL	R2 = R1 * R6					
ADD	R3 = R5 + R6					
DIV	R6 = R7/R3					
ADD	R1 = R4+R3					
LOAD	R4 = 4[R5]					
ADD	R1 = R1 + R2					
ADD	R1 = R1 + R5					
LOAD	R3 = 0[R5]					

4. Branch Prediction

Textbook problem 2.13, except only implement the branch direction predictor, *not* the BTB. For this problem, however, you will implement a predictor different than what is specified in the book. You may use whatever language you wish (C, C++, Java, Perl, etc.). In particular:

- (a) What is the branch prediction rates for a static always-taken predictor, and a static backwards-taken/forwards-not-taken (BTFNT) predictor?
- (b) Do parts (b,c,e,g) assuming a four-entry predictor, where each entry is a 2-bit saturating-counter predictor. Assume each entry is initialized to the value of 1 (weakly not-taken). Select an entry by using a simple modulo hash (e.g., for branch X, use counter $X \bmod 4$).
- (c) Do parts (b,c,e,g) assuming a global-history predictor with a history length of 4. That is, take the four most recent branch outcomes (globally), and use that as an index into a table of 16 entries (each a two-bit saturating counter initialized to a value of 1).