ECE/CS 250 Recitation #13- Pipelining

Prof. Lee

(material from Prof. Bletsch)

Objective: In this recitation, you will learn about real chips and improve your understanding of how ware runs on pipelines.

There are three tasks in this recitation. The UTAs will help to manage the time.

1. Pipelining, Dependences, and Hazards

You must understand the different types of dependences in programs. The most important is RAW (read-after-write), and you must be able to identify RAW dependences through registers and through memory. You must also be able to understand so-called "name dependences" (or "false dependences"): WAR (write-after-read) and WAW (write-after-write). Do not worry about RAR (read-after-read), since these "dependences" are never problematic.

(a) Construct a 2-line MIPS code snippet that has a RAW dependence through a register.
(b) Construct a 3-line MIPS code snippet in which the 2nd and 3rd instructions have RAW dependences on the register written by the 1st instruction.
(c) Construct a 2-line MIPS code snippet that has a RAW dependence through memory.
(d) Construct a 2-line MIPS code snippet that *may or may not* have a RAW dependence through memory. (Think about how you might not be able to tell if there's a RAW dependence!)
(e) Construct a 2-line MIPS code snippet that has a WAR dependence through a register.
(f) Construct a 2-line MIPS code snippet that has a WAW dependence through a register.
(g) Construct a 3 or 4 line MIPS code snippet that has at least one RAW, WAR, and WAW dependence.

Some dependences lead to hazards and some don't. Whether a dependence leads to a hazard is a function of the pipeline.

(h) Assume the 5-stage pipeline from class (and the textbook), and assume full bypassing wherever possible. Construct a 4-line MIPS code snippet with two RAW dependences through registers. One of the RAW dependences should be a hazard and the other should not.

2. Stalls and Bypasses

Assume the 5-stage pipeline we've used in class (F, D, X, M, W). Assume the pipeline forwards/bypasses operands whenever possible and stalls only when needed to satisfy a RAW dependence that can't be forwarded/bypassed. Assume all loads and stores hit in the 1-cycle data cache. Assume we're running the code from exercise 1 above.

Complete a pipeline diagram for the first 7 cycles, and assume that the add is in the Fetch (F) stage in cycle 1, as shown below.

	1	**2**	**3**	**4**	**5**	**6**	**7**
add $1, $2, $3	F						
xor $2, $1, $2							
lw $1, 4($2)							
sub $2, $5, $1							
sw $2, 4($5)							

3. Pipelining in the Real World

Time permitting, break up into groups of 3 or 4 people each, and in each group, pick one of these real-world processors. UTAs will help to make sure that every group has a different processor.

1. Intel's Core i7, code name: Broadwell
2. Intel's Atom, code name: Cedarville
3. AMD's Bobcat, code name: Llano
4. ARM's Cortex-M7
5. IBM's Power7
6. Qualcomm's Scorpion CPU
7. Alpha 21364 (also called EV7)
8. Atmel ATMEGA328 microcontroller[1]

Now learn as much as you can about your chosen processor:

(a) What is the ISA?
(b) How many registers does it have?
(c) How many stages are in its pipeline?
(d) What are those stages? Don't be worried if many stage names don't make sense yet, e.g., "register renaming", "wakeup", etc.
(e) What is the clock frequency? (There is likely to be a range of frequencies.)
(f) What are the sizes and associativities of the caches (L1I, L1D, L2, L3)? What are the cache block sizes?
(g) What are the sizes and associativities of the TLBs?

[1] This is a microcontroller rather than a CPU, meaning it has lots of general purpose IO pins for interfacing with custom circuits. You'll find that it's different from the CPUs in this exercise in a few other ways, too.