Computer Systems Organization

CSE 521/560M
Lecture 5
Prof. Patrick Crowley

Plan for Today

- Course Tip
- VHDL preparation
- Questions
- Pipelining discussion 2
Course Tip

- Question 1: *When you read, are you getting it?*
- Question 2: *How do you answer Question 1?*

Answer: **Write a program!**

Pipelining Refresher

- How does it help?
  - How much can it help?
- How does a program’s structure affect pipelining and its implementation?
- Is pipelining an easy or hard concept?
Terminology

• When an instruction moves
  – From ID to EX it is said to issue
  – From MEM to WB it is said to commit

Exceptions

• A.k.a: interrupts, faults (precise meanings are machine-specific)
• All handle ‘exceptional’ cases via a dynamic control flow transfer
• Examples:
  – External I/O requests
  – Operating system calls
  – User-requested breakpoints
  – Memory access violations
  – Page faults
  – Overflows
• Question: What’s the problem?
Handling Exceptions

- **Precise handling must:**
  - Accommodate the request (e.g., branch to some handler code)
  - ‘re-start’ the machine, as it was
    - Do not lose any data
    - Do not prematurely change state

- **Roughly:**
  1. Insert ‘trap’ instruction (a branch to ‘trusted code’, usually privileged) on next IF
  2. Don’t change state until trap takes effect
  3. Trap target saves state (PC, regs if needed) and restarts when completed

Exceptions (cont’d)
Pipeline Exceptions

• Can happen in most stages
  – IF: page fault, bad alignment, memory protection violation
  – ID: Illegal opcode
  – EX: Arithmetic exception (e.g., overflow)
  – MEM: same as IF
  – WB: None
• Must be treated in instruction order
  – Instruction $I$ starts at time $t$
  – Exception in MEM stage at time $t+3$ (treat it first)
  – Instruction $I+1$ starts at time $t+1$
  – Exc. in IF stage at time $t+1$ (occurs earlier but treated 2nd)

Treating Exceptions in Order

• Status vector of possible exceptions carried on with the instruction
• Once an exception is posted, no writing (no change of state; easy in integer pipeline – just prevent store in memory)
• When an instruction leaves MEM, check for exception
Difficulties

- Due to instruction set ("long" instructions)
  - String instructions (which use general regs to hold state)
  - Instructions that change state before that last stage (need to be able to "back up")
- Delayed branches
  - Need to keep 2 PC's
- Condition codes
  - Must remember when they were changed last
- Multicycle stages (stay tuned)

Multicycle Operations

- So far: EX stage lasts 1 cycle
- Not so for floating point operations!
- Assume four functional units
  - (0) Integer (loads, stores, integer ALU ops, branches)
  - (3) FP and integer multiplier
  - (6) FP adder
  - (25) FP and integer divider
Adding floating-point units

- When a unit is pipelined, an operation can be initiated every cycle; if not, pipeline must wait for latency
- Result of instruction $I$ can be forwarded to instruction $I + \text{latency}$

Unpipelined Functional Units

© 2003 Elsevier Science (USA). All rights reserved.
Hazards in Example Pipeline

- **Structural**: Yes
  - Divide unit is not pipelined. Any divides fewer than 25 cycles apart will stall the pipe
- **RAW**: Yes
  - Essentially handled as in integer pipe but with higher frequency of stalls
- **WAW**: Yes
- **Out of order completion**: Yes
Conflict Using the WB Stage

• E.g., `Mult` issued at time $t$ and `Add` issued at time $t+3$
• Solution: reserve the WB stage at ID stage (as in Cray-1)
  – Keep track of WB stage usage in a shift register
  – Reserve the right time slot. If busy, stall for a cycle and repeat
  – Shift every clock cycle

WAW Hazards

• Instruction $I$ writes F-P register $Fx$ at time $t$,
  Instruction $I+k$ writes F-P register $Fx$ at time $t-m$.
• But no instruction $I+1, I+2, \ldots, I+k$ uses $Fx$
  (otherwise there would be a stall)
• Only requirement is that $I+k$’s result be stored
• Solutions:
  – Squash $I$: difficult to track in the pipe
  – At ID stage check that result register is not a result register in all subsequent stages of other units. If it is, stall appropriate number of times.
Out-of-order Completion

• Instruction $I$ finishes at time $t$
  Instruction $I+k$ finishes at time $t-m$
  No hazard, etc.
• What happens if instruction $I$ causes an exception at a time in $[t+1,t-m]$ and Instruction $I+k$ writes in one of its source operands?

Exception Handling (Multicycle FP)

• Do nothing (e.g., imprecise exceptions; hard to implement virtual memory)
• Have a precise and imprecise mode (which restricts concurrency of F-P operations)
• Buffer results until previous (in-order) instructions have completed; can be costly when large differences in latencies (but same technique used for OOO execution)
• Restrict concurrency of F-P operations and on an exception “simulate in software” the instructions in between the faulting and finished one
• Flag early those operations that might result in an exception and stall accordingly
Assignment

- Test your understanding with a program!
- Readings
  - For Wednesday
    - Commentary: A Comparison of Two Pipeline Organizations
    - Re-read Appendix A, section A.8
  - For Monday
    - H&P 3.1-3.4
    - Ash: Skim Chapters 2-4 (syntax), Read Chapter 5