Computer Systems
Architecture I

CSE 560M
Lecture 7
Prof. Patrick Crowley
Plan for Today

• Announcement
  – HW1 due Wednesday
  – Commentary due next Monday

• Questions

• Pipelining III discussion
The Story with ILP So Far

• Pipelining (a good idea)
  – Pipeline CPI = Ideal CPI + CPI from stalls

• 5-stage RISC integer pipeline
  – Must watch for hazards and exceptions, but not too bad

• Multicycle (e.g., floating-point) operations
  – Hazard and exception detection harder, more stalls

• Problem: Since instructions execute in fetch order, one
  stalled instruction stalls all subsequent ones
Dynamic Scheduling

• Idea: re-order instructions in HW to reduce stalls \textit{without altering the program outcome}

• Implies possibility of
  – Out of order execution
  – Out of order completion
  – Imprecise exceptions (deal with this later)
Example

DIV.D R1,R2,R3 ; long latency
ADD.D R2,R1,R5 ; stall, RAW on R1
SUB.D R6,R7,R8 ; must this wait?
Checking for Issue

• MIPS pipeline only issues from ID once hazards are cleared

• Idea: Split ID stage into:
  – *Issue*: decode instructions; check for structural hazards (stall if nec.). Instructions flow in-order.
  – *Read operands*: wait until no data hazards then read operands. Instructions can flow out of order.

• After ID stage, instruction enters EX as before

• Note that WAR hazards may now occur
  
  \[
  \text{DIV.D} \quad R1, R2, R3 \quad ; \text{long latency} \\
  \text{ADD.D} \quad R2, R1, R4 \quad ; \text{stall, RAW on R1} \\
  \text{SUB.D} \quad R4, R7, R8 \quad ; \text{WAR hazard on R4}
  \]
Implementations of Dynamic Scheduling

• Goal: allow instructions to execute out of order when there are no structural or data hazards

• To compute correct results, need to keep track of:
  – Execution pipelines (recall that we now have several)
  – Register usage for read and write
  – Operation completion

• Two major techniques
  – Scoreboard (Seymour Cray for CDC 6600 in 1964)
  – Tomasulo’s algorithm (in IBM 360/91 in 1967)
CDC 6600 Scoreboard Structure

• Scoreboard responsible for:
  – Instruction issue
  – Instruction execution
  – Hazard detection

• Goal: complete one instruction every cycle
Scoreboarding Steps

- **4+1 steps**
  - replace ID, EX, MEM and WB in original MIPS pipeline

1. **Issue**
   - The execution unit must be free
   - There should be no WAW hazard
   - If either of these are false, the instruction stalls. No further issue is allowed (although fetching may continue if there is an instruction fetch buffer)
Scoreboarding Steps

2. Read Operands
   - When the instruction is issued, the execution unit is reserved (i.e., becomes busy)
   - Operands are read into the execution unit when they are ready (i.e., there is no RAW hazard)

3. Execution
   - One or more cycles depending on functional unit latency
   - When execution completes, the unit notifies the scoreboard that it’s ready to write the result
Scoreboarding Steps

4. Write result
   – Before writing, check for WAR hazards. If one exists, the unit is stalled until all WAR hazards are cleared

5. Delay
   – Because forwarding is not implemented, there should be one unit of delay between writing and reading the same register
   – Similarly, it takes one unit of time between the release of a unit and its next possible occupancy
What the Scoreboard Contains

• Status of each functional unit
  – Free or busy
  – Operation to be performed
  – Names of the result $F_i$ and source $F_j,F_k$ registers
  – Flags $R_j,R_k$ indicating whether the source registers are ready
  – Names $Q_j,Q_k$ of the units (if any) producing values for $F_j,F_k$

• Status of result registers
  – For each $F_i$ the name of unit (if any) that will produce its contents

• The instruction status
  – Has it been issued, is it in execution, is it ready to write?
Scoreboarding Example

Time \(i + 12\): MUL.D enters Execution Complete

Find:
Status tables when MUL.D is ready to move to Write Result state

Latencies:
Add/Sub: 2
Mult: 10
Div: 40
Simplifications and Optimizations

• We have assumed that there could be concurrent updates to the register file
  – Can be ameliorated by grouping execution units together and preventing concurrent writes in the same group

• Opportunities for optimization
  – Forwarding
  – Buffer first-available operand in functional unit rather than waiting for both (reduce WAR hazards)
Load Mult Sub Div

R2 F0 Integer
F2 Mult1
F3 F4 Mult2
F6 Add Yes
F8 Divide No
F10
Assignment

• HW1 due Wednesday

• Readings
  – For Wednesday
    • H&P: 2.1, 2.4-2.5
    • V&L: Chapter 5
  – For Monday
    • Commentary: *A Comparison of Two Pipeline Organizations*