18-447 Lecture 7: Pipelined Implementation

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

◆ Your goal today
  - getting started on pipelined implementations
◆ Notices
  - Lab 1, Part B, due this week
  - HW1, past due
  - Handout #5: Lab 2
  - Handout #6: HW 2
  - Handout #7: HW 1 solutions
◆ Readings
  - P&H Ch 4
1. “place one dirty load of clothes in the **w**asher”
2. “when the washer is finished, place the wet load in the **d**ryer”
3. “when the dryer is finished, you **f**old the dried load”
4. “when folding is finished, ask your **447** lab partner to put the clothes away”

   - steps to do a load are sequentially dependent
   - no dependence between different loads
   - different steps do not share resources

Doing laundry more quickly: in theory

- 4-loads of laundry in parallel
- no additional resources
  (all resources always busy!)
- throughput increased by 4
- latency per load is the same
Doing laundry more quickly: in practice

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Doing laundry more quickly: in practice

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Throughput restored (2 loads per hour) using 2 dryers
Ideal HW Pipelining

- Combinational logic $T$ psec
  - $BW = \sim(1/T)$

- $T/2$ ps
  - $BW = \sim(2/T)$

- $T/3$ ps
  - $BW = \sim(3/T)$

Notice: evenly divisible; no feedback wires

Performance Model

- Nonpipelined version with delay $T$
  - $BW = 1/(T+S)$ where $S = $ latch delay

- $T$ ps

- $k$-stage pipelined version
  - $BW_{k\text{-stage}} = 1 / (T/k + S)$
  - $BW_{\text{max}} = 1 / (1 \text{ gate delay} + S)$

- $T/k$ ps
  - $\cdots$
Cost Model

- **Nonpipelined version with combinational cost** $G$
  \[ \text{Cost} = G + L \text{ where } L = \text{latch cost} \]

- **k-stage pipelined version**
  \[ \text{Cost}_{k\text{-stage}} = G + Lk \]

Cost/Performance Trade-off  

[Peter M. Kogge, 1981]

Cost/Performance:
\[ \frac{C}{P} = \frac{[Lk + G]}{[1/(T/k + S)]} = (Lk + G) \frac{T}{k} + S \]
\[ = LT + GS + LSk + GT/k \]

Optimal Cost/Performance: find min. $C/P$ w.r.t. choice of $k$

\[ \frac{d}{dk} \left( \frac{Lk + G}{1 + \frac{T}{k} + S} \right) = 0 + 0 + LS - \frac{GT}{k^2} \]
\[ LS - \frac{GT}{2} = 0 \]
\[ k_{opt} = \sqrt{\frac{GT}{LS}} \]
Pipeline Idealism

Motivation: Increase throughput with little increase in hardware

- Repetition of identical tasks
  The same task is repeated on a large number of different inputs
- Repetition of independent tasks
  No ordering dependencies between repeated tasks
- Uniformly partitionable suboperations
  Can be evenly divided into uniform-latency suboperations (that do not share resources)

Good examples: automobile assembly line, doing laundry, but instruction pipeline???

The Reality of Pipelining

Instruction Execution . . . .
RISC Instruction Processing

- 5 generic steps
  - instruction fetch
  - instruction decode and operand fetch
  - ALU/execute
  - memory access
  - write-back

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Is this the correct partitioning? Why not 4 or 6 stages? Why not different boundaries

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Pipelining

Program execution order (in instructions)

lw $1, 100($0)
lw $2, 200($0)
lw $3, 300($0)

Time

Instruction fetch Reg ALU Data access Reg

800ps

Instruction fetch Reg ALU Data access Reg

200ps

Instruction fetch Reg ALU Data access Reg

200ps

Instruction fetch Reg ALU Data access Reg

200ps

Instruction fetch Reg ALU Data access Reg

200ps

Instruction fetch Reg ALU Data access Reg

200ps

Instruction fetch Reg ALU Data access Reg

200ps

Instruction fetch Reg ALU Data access Reg

200ps

... 800ps

5-stage speedup is 4, not 5 as predicated by the ideal model

Based on original figure from [P&H CODE, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Pipeline Registers

No resource is used by more than 1 stage!

Based on original figure from [P&H CODE, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Pipelined Operation

All instruction classes must follow the same path and timing through the pipeline stages. Any performance impact?
Illustrating Pipeline Operation: Resource View

<table>
<thead>
<tr>
<th></th>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>t₆</th>
<th>t₇</th>
<th>t₈</th>
<th>t₉</th>
<th>t₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>l₀</td>
<td>l₁</td>
<td>l₂</td>
<td>l₃</td>
<td>l₄</td>
<td>l₅</td>
<td>l₆</td>
<td>l₇</td>
<td>l₈</td>
<td>l₉</td>
<td>l₁₀</td>
</tr>
<tr>
<td>ID</td>
<td>l₀</td>
<td>l₁</td>
<td>l₂</td>
<td>l₃</td>
<td>l₄</td>
<td>l₅</td>
<td>l₆</td>
<td>l₇</td>
<td>l₈</td>
<td>l₉</td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td>l₀</td>
<td>l₁</td>
<td>l₂</td>
<td>l₃</td>
<td>l₄</td>
<td>l₅</td>
<td>l₆</td>
<td>l₇</td>
<td>l₈</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td>l₀</td>
<td>l₁</td>
<td>l₂</td>
<td>l₃</td>
<td>l₄</td>
<td>l₅</td>
<td>l₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td>l₀</td>
<td>l₁</td>
<td>l₂</td>
<td>l₃</td>
<td>l₄</td>
<td>l₅</td>
<td>l₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Illustrating Pipeline Operation: Operation View

Inst₀  | IF | ID | EX | MEM | WB |
Inst₁  | IF | ID | EX | MEM | WB |
Inst₂  | IF | ID | EX | MEM | WB |
Inst₃  | IF | ID | EX | MEM | WB |
Inst₄  | IF | ID | EX | MEM |     |
Example: Read-after-Write Hazard

Example: Pipeline Stalls

<table>
<thead>
<tr>
<th></th>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>t₆</th>
<th>t₇</th>
<th>t₈</th>
<th>t₉</th>
<th>t₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>l₀</td>
<td>l₁</td>
<td>l₂</td>
<td>l₃</td>
<td>l₄</td>
<td>l₄</td>
<td>l₄</td>
<td>l₅</td>
<td>l₆</td>
<td>l₇</td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>l₀</td>
<td>l₁</td>
<td>l₂</td>
<td>l₃</td>
<td>l₃</td>
<td>l₃</td>
<td>l₄</td>
<td>l₅</td>
<td>l₆</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td>l₀</td>
<td>l₁</td>
<td>l₂</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>l₃</td>
<td>l₄</td>
<td>l₅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td>l₀</td>
<td>l₁</td>
<td>l₂</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>l₃</td>
<td>l₄</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td>l₀</td>
<td>l₁</td>
<td>l₂</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>l₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

l₂ = addi ra, r-, - and l₃ = addi r-, ra, -
Sequential Control: Special Case

- For a given instruction
  - same control settings as single-cycle, but
  - control signals required at different cycles, depending on stage

\[ \text{decode once using the same logic as single-cycle and buffer control signals until consumed} \]

\[ \Rightarrow \text{or carry relevant “instruction word/field” down the pipeline and decode locally within each stage (still same logic)} \]

Which one is better?
Pipelined Control

Instruction Pipeline Reality

- Identical tasks ... NOT!
  - unifying instruction types
    - coalescing instruction types into one “multi-function” pipe
    - external fragmentation (some idle stages)
- Uniform suboperations ... NOT!
  - balance pipeline stages
    - group or sub-divide steps into stages to minimize variance
    - internal fragmentation (some too-fast stages)
- Independent tasks ... NOT! (next lecture)
  - resolve data and resource hazards
    - duplicate contended resources
    - inter-instruction dependency detection and resolution

MIPS ISA features are engineered for 5-stage pipelining