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
Doing laundry more quickly: in theory

1. “place one dirty load of clothes in **welder**”
2. “when washer is finished, place wet clothes in **dryer**”
3. “when dryer is finished, **you** fold dried clothes”
4. “when folding is finished, ask **friend** to put 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 for a 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

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

- Combinational logic: $T$ psec
  - Rate $\approx \frac{1}{T}$

- $T/2$ psec
  - Rate $\approx \frac{2}{T}$

- $T/3$ psec
  - Rate $\approx \frac{3}{T}$

Notice: evenly divisible; no feedback wires
Performance Model

• Nonpipelined version with delay $T$

$$\text{Rate} = \frac{1}{(T+S)} \quad \text{where} \quad S = \text{latch delay}$$


• $k$-stage pipelined version

$$\text{Rate}_{k\text{-stage}} = \frac{1}{(T/k + S)}$$
$$\text{Rate}_{\text{max}} = \frac{1}{(1 \text{ gate delay} + S)}$$

per-task latency became longer: $T+kS$
Cost Model

• Nonpipelined version with combinational cost $G$

Cost = $G + L$ where $L = \text{latch cost}$

• $k$-stage pipelined version

Cost$_{k\text{-stage}} = G + Lk$
Reality of Instruction Pipelining . . . .

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
18-447-S20-L07-S10, James C. Hoe, CMU/ECE/CALCM, ©2020
Pipeline Idealism

Motivation: Increase throughput without adding hardware cost

• Repetition of identical tasks
  same task repeated for many different inputs

• Repetition of independent tasks
  no ordering dependencies between repeated tasks

• Uniformly partitionable suboperations
  arbitrary number and placement of boundaries

Good examples: automobile assembly line, doing laundry, but instruction execution???
RISC Instruction Processing

• 5 generic steps
  – instruction fetch
  – instruction decode and operand fetch
  – ALU/execute
  – memory access
  – write-back
Coalescing and “External Fragmentation”

<table>
<thead>
<tr>
<th>steps</th>
<th>IF</th>
<th>ID</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-type</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>I-type</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>LW</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>SW</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Bxx/JALR</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>JAL</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

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

Is this the correct partitioning?
Why not 4 or 6 stages? Why not different boundaries
Internal and External Fragmentation

- 5-stage speedup is only 4
- Not all resources 100% utilized

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

No resource is used by more than 1 stage!
Pipelined Operation

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

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Illustrating Pipeline Operation: Resource View

<p>| | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>$t_0$</td>
<td>$t_1$</td>
<td>$t_2$</td>
<td>$t_3$</td>
<td>$t_4$</td>
<td>$t_5$</td>
<td>$t_6$</td>
<td>$t_7$</td>
<td>$t_8$</td>
<td>$t_9$</td>
</tr>
<tr>
<td></td>
<td>$I_0$</td>
<td>$I_1$</td>
<td>$I_2$</td>
<td>$I_3$</td>
<td>$I_4$</td>
<td>$I_5$</td>
<td>$I_6$</td>
<td>$I_7$</td>
<td>$I_8$</td>
<td>$I_9$</td>
</tr>
<tr>
<td>ID</td>
<td>$I_0$</td>
<td>$I_1$</td>
<td>$I_2$</td>
<td>$I_3$</td>
<td>$I_4$</td>
<td>$I_5$</td>
<td>$I_6$</td>
<td>$I_7$</td>
<td>$I_8$</td>
<td>$I_9$</td>
</tr>
<tr>
<td>EX</td>
<td>$I_0$</td>
<td>$I_1$</td>
<td>$I_2$</td>
<td>$I_3$</td>
<td>$I_4$</td>
<td>$I_5$</td>
<td>$I_6$</td>
<td>$I_7$</td>
<td>$I_8$</td>
<td>$I_{10}$</td>
</tr>
<tr>
<td>MEM</td>
<td>$I_0$</td>
<td>$I_1$</td>
<td>$I_2$</td>
<td>$I_3$</td>
<td>$I_4$</td>
<td>$I_5$</td>
<td>$I_6$</td>
<td>$I_7$</td>
<td>$I_{10}$</td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td>$I_0$</td>
<td>$I_1$</td>
<td>$I_2$</td>
<td>$I_3$</td>
<td>$I_4$</td>
<td>$I_5$</td>
<td>$I_6$</td>
<td>$I_{10}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Illustrating Pipeline Operation: Operation View

Inst_0

Inst_1

Inst_2

Inst_3

Inst_4

t_0    t_1    t_2    t_3    t_4    t_5

IF    ID    EX    MEM    WB
IF    ID    EX    MEM    WB
IF    ID    EX    MEM    WB
IF    ID    EX    MEM    WB
IF    ID    EX    MEM    WB

18-447-S20-L07-S20, James C. Hoe, CMU/ECE/CALCM, ©2020
Example: Read-after-Write Hazard

```
addi   ra r- r- - 
addi   r- ra - 
addi   r- ra - 
addi   r- ra - 
addi   r- ra - 
addi   r- ra - 
```

```
t_0  | t_1  | t_2  | t_3  | t_4  | t_5  
IF   | ID   | EX   | MEM  | WB   |
IF   | ID   | EX   | MEM  | WB   |
IF   | ID   | EX   | MEM  |
IF   | ID   | EX   |
IF   | ID   |
IF   |
```
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><strong>IF</strong></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><strong>ID</strong></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><strong>EX</strong></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><strong>MEM</strong></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><strong>WB</strong></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-, - ; 
l₃ = addi r-, ra, - ;
Control Points

Identical set of control points as the single-cycle datapath!!
Sequential Control: Special Case

- For a given instruction
  - same control settings as single-cycle, but
  - control signals required at different cycles, depending on stage
  - decode once using the same logic as single-cycle and buffer control signals until consumed
Pipelined Control

This is all there is to it (without hazards)!!
Instruction Pipeline Reality

• Not identical tasks
  – coalescing instruction types into one “multi-function” pipe
  – external fragmentation (some idle stages)

• Not uniform suboperations
  – group or sub-divide steps into stages to minimize variance
  – internal fragmentation (some too-fast stages)

• Not independent tasks
  – dependency detection and resolution
  – next lecture(s)

Even more messy if not RISC
Data Dependence

Data dependence
\[ r_3 \leftarrow r_1 \text{ op } r_2 \]
\[ \ldots \]
\[ r_5 \leftarrow r_3 \text{ op } r_4 \]

Read-after-Write (RAW)

Anti-dependence
\[ r_3 \leftarrow r_1 \text{ op } r_2 \]
\[ \ldots \]
\[ r_1 \leftarrow r_4 \text{ op } r_5 \]

Write-after-Read (WAR)

Output-dependence
\[ r_3 \leftarrow r_1 \text{ op } r_2 \]
\[ \ldots \]
\[ r_3 \leftarrow r_6 \text{ op } r_7 \]

Write-after-Write (WAW)

Don’t forget memory instructions
Control Dependence

- C-Code

{ code A }
if X==Y then
  { code B }
else
  { code C }
{ code D }

Does B or C come after A?