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 Friday midnight
  – HW1, past due
  – Handout #5: HW 2
  – Handout #6: HW 1 solutions

• Readings
  – P&H Ch 4
Doing laundry more quickly: in theory

1. “place one dirty load of clothes in **washer**”
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

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Doing laundry more quickly: in practice

The slowest step decides throughput
Doing laundry more quickly: in practice

Throughput restored (2 loads per hour) using 2 dryers

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

- Combinational logic: $T$ psec
  - Throughput: $\sim (1/T)$

- Pipelining: $T/2$ psec
  - Throughput: $\sim (2/T)$
  - Speedup: $2$

- Pipelining: $T/3$ psec
  - Throughput: $\sim (3/T)$
  - Speedup: $3$

Notice: evenly divisible; no feedback wires
Performance Model

- Nonpipelined version with delay $T$
  \[ \text{throughput} = \frac{1}{T+S} \text{ where } S = \text{latch delay} \]

- $k$-stage pipelined version
  \[ \text{throughput}_{k\text{-stage}} = \frac{1}{T/k + S} \]
  \[ \text{throughput}_{\text{max}} = \frac{1}{1 \text{ gate delay} + S} \]

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

• Nonpipelined version with combinational cost $G$

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

• $k$-stage pipelined version

$\text{Cost}_{k\text{-stage}} = G + L_k$

18-447-S21-L07-S9, James C. Hoe, CMU/ECE/CALCM, ©2021
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???
Reality of Instruction Pipelining . . . .

never mind this complication today

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

18-447-S21-L07-S11, James C. Hoe, CMU/ECE/CALCM, ©2021
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

200ps  
IF: Instruction fetch

100ps  
ID: Instruction decode/ 
register file read

200ps  
EX: Execute/ 
address calculation

200ps  
MEM: Memory access

100ps  
WB: Write back

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.]
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!

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]  
18-447-S21-L07-S16, James C. Hoe, CMU/ECE/CALCM, ©2021
Pipelined Operation

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

What if LW dest is $2$?
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>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
<td>I₆</td>
<td>I₇</td>
<td>I₈</td>
<td>I₉</td>
<td>I₁₀</td>
</tr>
<tr>
<td>ID</td>
<td>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
<td>I₆</td>
<td>I₇</td>
<td>I₈</td>
<td>I₉</td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
<td>I₆</td>
<td>I₇</td>
<td>I₈</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
<td>I₆</td>
<td>I₇</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
<td>I₆</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

18-447-S21-L07-S19, James C. Hoe, CMU/ECE/CALCM, ©2021
Illustrating Pipeline Operation: Operation View
Example: Read-after-Write Hazard

1. addi x1, x0, 0
2. addi x2, x1, 0
3. addi x3, x1, 0
4. addi x4, x1, 0
5. addi x5, x1, 0
6. addi x6, x1, 0
**Example: Pipeline Stalls**

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
<th>$t_6$</th>
<th>$t_7$</th>
<th>$t_8$</th>
<th>$t_9$</th>
<th>$t_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF</strong></td>
<td>$l_0$</td>
<td>$l_1$</td>
<td>$l_2$</td>
<td>$l_3$</td>
<td>$l_4$</td>
<td>$l_4$</td>
<td>$l_4$</td>
<td>$l_5$</td>
<td>$l_6$</td>
<td>$l_7$</td>
<td></td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>$l_0$</td>
<td>$l_1$</td>
<td>$l_2$</td>
<td>$l_3$</td>
<td>$l_3$</td>
<td>$l_3$</td>
<td>$l_3$</td>
<td>$l_4$</td>
<td>$l_5$</td>
<td>$l_6$</td>
<td></td>
</tr>
<tr>
<td><strong>EX</strong></td>
<td>$l_0$</td>
<td>$l_1$</td>
<td>$l_2$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$l_3$</td>
<td>$l_4$</td>
<td>$l_5$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MEM</strong></td>
<td>$l_0$</td>
<td>$l_1$</td>
<td>$l_2$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$l_3$</td>
<td>$l_4$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WB</strong></td>
<td>$l_0$</td>
<td>$l_1$</td>
<td>$l_2$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$\emptyset$</td>
<td>$l_3$</td>
<td>$l_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$l_2 = \text{addi } x1, x0, 0$;  
$l_3 = \text{addi } x2, x1, 0$;
Control Points

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

Identical set of control points as the single-cycle datapath!!

18-447-S21-L07-S23, James C. Hoe, CMU/ECE/CALCM, ©2021
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

\[
\begin{align*}
x_3 & \leftarrow x_1 \text{ op } x_2 \\
\ldots & \\
x_5 & \leftarrow x_3 \text{ op } x_4
\end{align*}
\]

Read-after-Write (RAW)

Anti-dependence

\[
\begin{align*}
x_3 & \leftarrow x_1 \text{ op } x_2 \\
\ldots & \\
x_1 & \leftarrow x_4 \text{ op } x_5
\end{align*}
\]

Write-after-Read (WAR)

Output-dependence

\[
\begin{align*}
x_3 & \leftarrow x_1 \text{ op } x_2 \\
\ldots & \\
x_3 & \leftarrow x_6 \text{ op } x_7
\end{align*}
\]

Write-after-Write (WAW)

Don’t forget memory instructions
Control Dependence

• C-Code

\[
\{ \text{code A} \} \\
\text{if } X==Y \text{ then} \\
\{ \text{code B} \} \\
\text{else} \\
\{ \text{code C} \} \\
\{ \text{code D} \}
\]

Does B or C come after A?