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
  – HW 2, due Mon 2/19

  Look for Handout #7: Lab 2 on Friday

• Readings
  – P&H Ch 4
1. “place one load of dirty clothes in washer”
2. “when washer is finished, place washed clothes in dryer”
3. “when dryer is finished, you fold dried clothes”
4. “when folding is finished, friend put away folded clothes”

- 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.]
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???
(Ideal) HW Pipelining

- Combinational logic $T \ \text{psec}$
  - Throughput $\sim \frac{1}{T}$

- $T/2 \ \text{psec}$
  - Throughput $\sim \frac{2}{T}$
  - Speedup $\sim 2$

- $T/3 \ \text{psec}$
  - Throughput $\sim \frac{3}{T}$
  - Speedup $\sim 3$

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 $G$

Cost $= G + L$ where $L$ = latch cost

$G$ gates

• $k$-stage pipelined version

Cost$_{k$-stage} $= G + Lk$

$G/k$ $\rightarrow \cdots \rightarrow G/K$
Cost/Performance Trade-off

[Peter M. Kogge, 1981]

Cost/Performance:

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

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

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

\[ LS - \frac{GT}{k^2} = 0 \]

\[ k_{opt} = \sqrt{\frac{GT}{LS}} \]
Reality of Instruction Pipelining . . . .

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

18-447-S24-L07-S12, James C. Hoe, CMU/ECE/CALCM, ©2024
RISC Instruction Processing

- 5 generic steps
  - instruction fetch
  - instruction decode and operand fetch
  - ALU/execute
  - memory access (not required by non-mem instructions)
  - write-back

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

18-447-S24-L07-S14, James C. Hoe, CMU/ECE/CALCM, ©2024

18-447-S24-L07-S14, James C. Hoe, CMU/ECE/CALCM, ©2024
First-Cut at 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.]
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

What if LW dest is $2$?
Optimize Latency of ALU Insts?

<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.]
# 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>
Illustrating Pipeline Operation: Operation View

\[\begin{array}{ccccccc}
& t_0 & t_1 & t_2 & t_3 & t_4 & t_5 \\
\text{Inst}_0 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} & \\
\text{Inst}_1 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} & \\
\text{Inst}_2 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} & \\
\text{Inst}_3 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} & \\
\text{Inst}_4 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} &
\end{array}\]

18-447-S24-L07-S22, James C. Hoe, CMU/ECE/CALCM, ©2024
Example: Read-after-Write Hazard

\[
\begin{array}{c|c|c|c|c|c|c}
\text{addi} & x1, x0, 0 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} \\
\text{addi} & x2, x1, 0 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} \\
\text{addi} & x3, x1, 0 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} \\
\text{addi} & x4, x1, 0 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} \\
\text{addi} & x5, x1, 0 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} \\
\text{addi} & x6, x1, 0 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} \\
\end{array}
\]
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>$I_0$</td>
<td>$I_1$</td>
<td>$I_2$</td>
<td>$I_3$</td>
<td>$I_4$</td>
<td>$I_4$</td>
<td>$I_4$</td>
<td>$I_5$</td>
<td>$I_6$</td>
<td>$I_7$</td>
<td></td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>$I_0$</td>
<td>$I_1$</td>
<td>$I_2$</td>
<td>$I_3$</td>
<td>$I_3$</td>
<td>$I_3$</td>
<td>$I_4$</td>
<td>$I_5$</td>
<td>$I_6$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EX</strong></td>
<td>$I_0$</td>
<td>$I_1$</td>
<td>$I_2$</td>
<td></td>
<td></td>
<td></td>
<td>$I_3$</td>
<td>$I_4$</td>
<td>$I_5$</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MEM</strong></td>
<td>$I_0$</td>
<td>$I_1$</td>
<td>$I_2$</td>
<td></td>
<td></td>
<td></td>
<td>$I_3$</td>
<td>$I_4$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WB</strong></td>
<td>$I_0$</td>
<td>$I_1$</td>
<td>$I_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$I_3$</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$l_2 = \text{addi } x1, x0, 0$; \hspace{0.5cm} 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!!
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
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

\[ x_3 \leftarrow x_1 \text{ op } x_2 \]
\[ \ldots \]
\[ x_5 \leftarrow x_3 \text{ op } x_4 \]

Read-after-Write (RAW)

Anti-dependence

\[ x_3 \leftarrow x_1 \text{ op } x_2 \]
\[ \ldots \]
\[ x_1 \leftarrow x_4 \text{ op } x_5 \]

Write-after-Read (WAR)

Output-dependence

\[ x_3 \leftarrow x_1 \text{ op } x_2 \]
\[ \ldots \]
\[ x_3 \leftarrow x_6 \text{ op } x_7 \]

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 }
```

Control Flow Graph

```
True
  code A
    if X==Y
    code B
    code C
False
    code D
```

Assembly Code (linearized)

```
  code A
    if X==Y
      goto
    code C
      goto
      code D
```

Does B or C happen after A?