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 2/21

  Look for Handout #7: Lab 2 on Friday

• Readings
  – P&H Ch 4
Doing laundry: by the book (P&H 4.6)

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.]

18-447-S22-L07-S4, James C. Hoe, CMU/ECE/CALCM, ©2022
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
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**
  - 1 cycle
  - Throughput: $\sim (1/T)$

- **Ideal HW Pipelining**
  - 3 cycles
  - Throughput: $\sim (3/T)$
  - Speedup: 3

Notice: evenly divisible; no feedback wires
Performance Model

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

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

- $k$-stage pipelined version

  \[ \text{Cost}_{k\text{-stage}} = G + Lk \]
Cost/Performance Trade-off

[Peter M. Kogge, 1981]

Cost/Performance:

\[
C/P = \frac{[Lk + G]}{[1/(T/k + S)]} = (Lk + G) \frac{T}{k + S}
\]

\[
= LT + GS + LSk + GT/k
\]

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

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

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

\[
k_{opt} = \sqrt{\frac{GT}{LS}}
\]
never mind this complication today
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

Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
• 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-S22-L07-S17, James C. Hoe, CMU/ECE/CALCM, ©2022
Pipelined Operation
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><strong>IF</strong></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><strong>ID</strong></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><strong>EX</strong></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><strong>MEM</strong></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><strong>WB</strong></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

Illustration of pipeline operations at different time intervals.

- **Inst₀**:
  - $t₀$: IF, ID
  - $t₁$: ID, EX
  - $t₂$: MEM, WB

- **Inst₁**:
  - $t₀$: IF, ID
  - $t₁$: ID, EX
  - $t₂$: MEM, WB

- **Inst₂**:
  - $t₀$: IF, ID
  - $t₁$: ID, EX
  - $t₂$: MEM, WB

- **Inst₃**:
  - $t₀$: IF, ID
  - $t₁$: ID, EX
  - $t₂$: MEM, WB

- **Inst₄**:
  - $t₀$: IF, ID
  - $t₁$: ID, EX
  - $t₂$: MEM, WB

The pipeline operations are shown at different time intervals (t₀ to t₅) with each stage: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), and Write Back (WB).

Diagram shows the progression of instructions through the pipeline stages over time.
Example: Read-after-Write Hazard

```
addi   x1, x0, 0
addi   x2, x1, 0
addi   x3, x1, 0
addi   x4, x1, 0
addi   x5, x1, 0
addi   x6, x1, 0
```

Time:
- \(t_0\), \(t_1\), \(t_2\), \(t_3\), \(t_4\), \(t_5\)
### Example: Pipeline Stalls

<table>
<thead>
<tr>
<th>IF</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>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₄</td>
<td>I₄</td>
<td>I₄</td>
<td>I₅</td>
<td>I₆</td>
<td>I₇</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₃</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
<td>I₆</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>I₃</td>
<td>I₄</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>I₃</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

I₂ = addi x₁, x₀, 0;
I₃ = addi x₂, x₁, 0;
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
Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

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 }

Does B or C come after A?