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
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
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

\[ T \text{ psec} \]

\[ \text{throughput} = \sim \left( \frac{1}{T} \right) \]

\[ T/2 \text{ psec} \]

\[ \text{throughput} = \sim \left( \frac{2}{T} \right) \]

\[ \text{speedup} = 2 \]

\[ T/3 \text{ psec} \]

\[ \text{throughput} = \sim \left( \frac{3}{T} \right) \]

\[ \text{speedup} = 3 \]

Notice: evenly divisible; no feedback wires
**Performance Model**

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

  ![Diagram of nonpipelined version]

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

  ![Diagram of pipelined version]

  **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:

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

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-S22-L07-S12, James C. Hoe, CMU/ECE/CALCM, ©2022
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>
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

Program execution order (in instructions)

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

Time

- Instruction fetch
- Reg
- ALU
- Data access
- Reg

200 ps clock period

Program execution order (in instructions)

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

Time

- Instruction fetch
- Reg
- ALU
- Data access
- Reg

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>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>

*IF* stands for Instruction Fetch, *ID* for Instruction Decode, *EX* for Execution, *MEM* for Memory Access, and *WB* for Write Back.
Illustrating Pipeline Operation: Operation View

```
<table>
<thead>
<tr>
<th>t0</th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
<th>t4</th>
<th>t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td></td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>WB</td>
</tr>
<tr>
<td>IF</td>
<td>ID</td>
<td>EX</td>
<td>MEM</td>
<td>WB</td>
<td>WB</td>
</tr>
</tbody>
</table>
```

Inst0, Inst1, Inst2, Inst3, Inst4
Example: Read-after-Write Hazard

\[
\begin{array}{ccccccc}
 t_0 & t_1 & t_2 & t_3 & t_4 & t_5 \\
\text{addi} & x1, x0, 0 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} \\
\text{addi} & x2, x1, 0 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} & \text{WB} \\
\text{addi} & x3, x1, 0 & \text{IF} & \text{ID} & \text{EX} & \text{MEM} \\
\text{addi} & x4, x1, 0 & \text{IF} & \text{ID} & \text{EX} \\
\text{addi} & x5, x1, 0 & \text{IF} & \text{ID} \\
\text{addi} & x6, x1, 0 & \text{IF} \\
\end{array}
\]
### 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>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>
</tr>
<tr>
<td><strong>ID</strong></td>
<td></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>
</tr>
<tr>
<td><strong>EX</strong></td>
<td></td>
<td></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>
</tr>
<tr>
<td><strong>MEM</strong></td>
<td></td>
<td></td>
<td></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>
</tr>
<tr>
<td><strong>WB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>Ø</td>
<td>Ø</td>
<td>Ø</td>
<td>I₃</td>
</tr>
</tbody>
</table>

\[I₂=\text{addi}\ x₁,\ x₀,\ 0\; ;\; \; I₃=\text{addi}\ x₂,\ x₁,\ 0\; ;\]
Control Points

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

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

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

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

Does B or C come after A?