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 tomorrow
  – HW1, past due
  – Handout #5: Lab 2
  – Handout #6: HW 2
  – Handout #7: HW 1 solutions
  – **Office Hours: M 11~12 and F 1:30~2:30**

• 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 per load is the same
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.]

18-447-S18-L07-S6, James C. Hoe, CMU/ECE/CALCM, ©2018
(Ideal) HW Pipelining

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

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

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

Notice: evenly divisible; no feedback wires
Performance Model

• Nonpipelined version with delay $T$

  Rate = $1/(T+S)$ where $S$ = latch delay

  $T$ psec

• k-stage pipelined version

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

  Rate$_{\text{max}} = 1 / (1\text{ gate delay} + S )$

  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
  
  $\text{Cost}_{k\text{-stage}} = G + Lk$
Pipeline Idealism

Motivation: Increase throughput without adding hardware cost

- Repetition of identical tasks
  
  \textit{same task repeated for many different inputs}

- Repetition of independent tasks
  
  \textit{no ordering dependencies between repeated tasks}

- Uniformly partitionable suboperations
  
  \textit{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

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.]

18-447-S18-L07-S15, James C. Hoe, CMU/ECE/CALCM, ©2018
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-S18-L07-S16, James C. Hoe, CMU/ECE/CALCM, ©2018
Pipelined Operation
Pipelined Operation
## 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>I₁₀</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>I₁₀</td>
<td>I₁₀</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>I₁₀</td>
<td>I₁₀</td>
<td>I₁₀</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>I₁₀</td>
<td>I₁₀</td>
<td>I₁₀</td>
<td>I₁₀</td>
</tr>
</tbody>
</table>
Illustrating Pipeline Operation: Operation View

Inst_0

Inst_1

Inst_2

Inst_3

Inst_4
Example: Read-after-Write Hazard

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

Activity Schedule:

- **t_0**: IF
- **t_1**: ID
- **t_2**: EX
- **t_3**: MEM
- **t_4**: WB
- **t_5**: IF

Note: The schedule diagrams illustrate the dependencies and hazards in a pipeline for executing the given instructions.
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>IF</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>ID</td>
<td>I₀</td>
<td>I₁</td>
<td>I₂</td>
<td>I₃</td>
<td>I₃</td>
<td>I₃</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</td>
<td>I₆</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>Ø</td>
<td>I₃</td>
<td>I₄</td>
<td>I₅</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>Ø</td>
<td>I₃</td>
<td>I₄</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>I₃</td>
</tr>
</tbody>
</table>

I₂ = addi ra, r-, -;  I₃ = addi r-, ra, -;
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)!!

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