Advanced Computer Architecture
ELEC3219 (2017/18)
Pipeline, Superscalar and OoO

Dr. Terrence Mak
tmak@ecs.soton.ac.uk
Pipelining
What Is Pipelining

• Laundry Example
  – Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
    • Washer takes 30 minutes
    • Dryer takes 40 minutes
    • “Folder” takes 20 minutes
Sequential laundry takes 6 hours for 4 loads
If they learned pipelining, how long would laundry take?
What Is Pipelining

Start work ASAP

• Pipelined laundry takes 3.5 hours for 4 loads versus 6 hours without using pipeline
Pipelining Lessons

- Multiple tasks operating simultaneously
- Doesn’t help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
Pipelining in circuits

Without Pipeline

With Pipeline

Clock period is maximum propagation delay from flip-flop output (or circuit input) to flip-flop input
Pipelining in circuits: Bad Example

Without Pipeline

With Pipeline (ILL-formed)
Well-formed pipeline

• Same number of flip-flops (latches) along any path from any input to any output
  – Ensures that every computation unit sees inputs in phase

With Pipeline (Well-formed)
$k$-pipelines

- $k$ flip-flops on each input-output path
- Always have flip-flops on each output

![Diagram of k-pipelines]

$K = 1$  $K = 2$
Pipelining a Digital System

• Key idea: break big computation up into pieces

• Separate each piece with a **pipeline register**
Superscalar and Superpipeline
Superscalar

Yes. This improvement is called a **superscalar pipeline**. What we have seen so far are called **scalar pipelines**.

- Overcome the limit of scalar pipeline using **multiple issue**
  - Two instructions per stage at once, or three, or four, or eight...

Today, typically “4-wide” (Intel Core i7, AMD Opteron)
- Some more (Power5 is 5-issue; Itanium is 6-issue)
- Some less (dual-issue is common for simple cores, e.g., ARM A-8)

http://en.wikipedia.org/wiki/Superscalar
Scalar (like Pipeline) vs Superscalar

**Scalar**
- lw 0(r1) → r2
- lw 4(r1) → r3
- lw 8(r1) → r4
- add r14, r15 → r6
- add r12, r13 → r7
- add r17, r16 → r8
- lw 0(r18) → r9

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**2-way Superscalar**
- lw 0(r1) → r2
- lw 4(r1) → r3
- lw 8(r1) → r4
- add r14, r15 → r6
- add r12, r13 → r7
- add r17, r16 → r8
- lw 0(r18) → r9

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>F</td>
<td>D</td>
<td>X</td>
<td>M</td>
<td>W</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

14
Scalar (like Pipeline) vs Superpipeline

Scalar (Pipeline)
Vaguely defined as deep pipelining, i.e., lots of stages

Superpipeline

Vaguely defined as $M$ minor cycle

major cycle = $M$ minor cycle

minor cycle
Superpipeline example

- Intel Atom
- 6 colours!
- Each operation is divided
- Each stage executes a micro-op (not an instruction)
- More simple stages faster clock, but greater likelihood of stalls.
Superpipeline - Saving More Power

- 1/4 Power consumption comparing to Intel
- Timing issue for superpipeline
- Although embedded, still quite big (Intel)
Superscalar and superpipelined machines of equal degree have roughly the same performance, i.e. if \( n = m \) then both have about the same IPC.
Comments about Pipelining and Superscalar

• The good news
  – Multiple instructions are being processed at same time
  – This works because stages are isolated by registers
  – Best case speedup of $N$

• The bad news
  – Interrupt response can be poor with longer pipelines
  – Instructions interfere with each other, i.e. they have dependency
    • Example: different instructions may need the same piece of hardware (e.g., memory) in same clock cycle
    • Example: instruction may require a result produced by an earlier instruction that is not yet complete
In-order and Out-of-Order
Out-of-Order (OoO) execution

• So far, we have assumed in-order execution
  – Issue instructions in the order they occur
• Out-of-order in a nutshell
  – HW examines a sliding window of consecutive instructions, i.e., the “instruction window”
  – Ready instructions get picked up from window
  – Executed *out of program order*
  – Instruction results are committed to the machine state (memory+reg. file) in original program order
  – User is unaware (except that the program runs faster)
In-order vs out-of-order execution

(1) \( r_1 \leftarrow r_4 / r_7 \); assume divide takes 20 cycles
(2) \( r_8 \leftarrow r_1 + r_2 \)
(3) \( r_5 \leftarrow r_5 + 1 \)
(4) \( r_6 \leftarrow r_6 - r_3 \)
(5) \( r_4 \leftarrow r_5 + r_6 \)
(6) \( r_7 \leftarrow r_8 \times r_4 \)

Following the Mathematical order

Following the quickest order

Data Flow Graph

In-order execution

Out-of-order execution

Time
Out-of-order execution

When an instruction does not depend on subsequent instructions, there will be no stalling on the pipeline of subsequent instructions.
Out-of-Order (O3) execution

- Assume a 3 stage execution in a pipeline that can issue two instructions, execute three instructions and write back two results every cycle.
Question: In-order and Out-of-Order

• Assume:
  – I1 requires 2 cycles to execute
  – I3 and I4 are in conflict for a functional unit
  – I5 depends on the value produced by I4
  – I5 and I6 are in conflict for a functional unit

IPC = ?
In-order issue and in-order completion
In-order issue and out-of-order completion
Out-of-order issue and Out-of-order completion
• END