CSCI-564 Advanced Computer Architecture

Lecture 8: Handling Exceptions and Interrupts / Superscalar

Bo Wu
Colorado School of Mines
Branch Delay Slots
(expose control hazard to software)

• Change the ISA semantics so that the instruction that follows a jump or branch is always executed
  — gives compiler the flexibility to put in a useful instruction where normally a pipeline bubble would have resulted.

\begin{center}
\begin{tabular}{c|c|l}
  I_1 & 096 & ADD  \\
  I_2 & 100 & BEQZ r1 +200  \\
  I_3 & 104 & ADD  \\
  I_4 & 304 & ADD  \\
\end{tabular}
\end{center}

Delay slot instruction executed regardless of branch outcome

• Other techniques include more advanced branch prediction, which can dramatically reduce the branch penalty... to come later
In the Compiler

Control flow graph

Assembly

t0 = 0
t1 = 0

t2 = t1 < 32

popcount:
ori $v0, $zero, 0
ori $t1, $zero, 0

t2 == 0

t2 != 0

notone:
beq $zero, $zero, top
addi $t1, $t1, 1

end:
addi $t1, $t1, 1
jr $ra

return t0
Branch Pipeline Diagrams
(branch delay slot)

\[ \text{time} \]
\[ t0 \quad t1 \quad t2 \quad t3 \quad t4 \quad t5 \quad t6 \quad t7 \quad \ldots \]

(I_1) 096: ADD
\[ \begin{align*}
\text{IF}_1 & \quad \text{ID}_1 & \quad \text{EX}_1 & \quad \text{MA}_1 & \quad \text{WB}_1 \\
\end{align*} \]

(I_2) 100: BEQZ +200
\[ \begin{align*}
\text{IF}_2 & \quad \text{ID}_2 & \quad \text{EX}_2 & \quad \text{MA}_2 & \quad \text{WB}_2 \\
\end{align*} \]

(I_3) 104: ADD
\[ \begin{align*}
\text{IF}_3 & \quad \text{ID}_3 & \quad \text{EX}_3 & \quad \text{MA}_3 & \quad \text{WB}_3 \\
\end{align*} \]

(I_4) 304: ADD
\[ \begin{align*}
\text{IF}_4 & \quad \text{ID}_4 & \quad \text{EX}_4 & \quad \text{MA}_4 & \quad \text{WB}_4 \\
\end{align*} \]

Resource Usage

\[ \text{time} \]
\[ t0 \quad t1 \quad t2 \quad t3 \quad t4 \quad t5 \quad t6 \quad t7 \quad \ldots \]

IF
\[ I_1 \quad I_2 \quad I_3 \quad I_4 \]

ID
\[ I_1 \quad I_2 \quad I_3 \quad I_4 \]

EX
\[ I_1 \quad I_2 \quad I_3 \quad I_4 \]

MA
\[ I_1 \quad I_2 \quad I_3 \quad I_4 \]

WB
\[ I_1 \quad I_2 \quad I_3 \quad I_4 \]
Scheduling in Action

• Suppose every branch brings two stalls, how do you schedule the following instructions?

```
SUB t2, t3, t5
ADD t4, t3, t5
bneqz t4, 400
ADD t1, t3, t8
ADD t3, t5, t9
beqz t4, 500
```
Suppose every branch brings two stalls, how do you schedule the following instructions?

Scheduling in Action

SUB t2, t3, t5
ADD t4, t3, t5
bneqz t4, 400
ADD t1, t3, t8
ADD t3, t5, t9
beqz t4, 500
ADD t4, t3, t5
bneqz t4, 400
SUB t2, t3, t5
beqz t4, 500
ADD t1, t3, t8
ADD t3, t5, t9
Scheduling in Action

• Suppose every branch brings two stalls, how do you schedule the following instructions?

SUB t2, t3, t5
ADD t4, t3, t5
bneqz t4, 400
ADD t1, t3, t8
ADD t3, t5, t9
beqz t4, 500

ADD t4, t3, t5
bneqz t4, 400
SUB t2, t3, t5
beqz t4, 500
ADD t1, t3, t8
ADD t3, t5, t9

stall
Exceptions

• Causes
  • Arithmetic overflow
    • In an 8-bit machine, what if you do 255+1?
  • Undefined instruction
  • System call

• When to handle
  • when detected

• Who should handle
  • Process
Interrupts

• Causes
  • external events
    • arrival of network package, hard disk, …

• When to handle
  • when convenient except for high priority ones

• Who should handle
  • System
Precise exceptions/interrupts

- The architectural state should be consistent when the exception/interrupt is ready to be handled
  - All previous instructions should be completely retired.
  - No later instruction should be retired.

Retire = commit = finish execution and update arch. state
Multi-cycle execute

FMUL R4 ← R1, R2
ADD  R3 ← R1, R2
FMUL R2 ← R5, R6
ADD  R4 ← R5, R6

• instructions may take multiple cycles in ALU
• What’s wrong here?
  • what if floating point ALU incurs an exception?
Ensuring precise exceptions in pipelining

- Idea 1: make each operation take the same amount of time

- Downside
  - worse-case latency determines all instructions’ latency
  - chance is high for structural hazards
Ensuring precise exceptions in pipelining

- Idea 2: Reorder buffer (ROB)
  - Complete instructions out-of-order, but reorder them before making results visible to architectural state
  - When instruction is decoded it reserves an entry in the ROB
  - When instruction completes, it writes result into ROB entry
  - When instruction oldest in ROB and it has completed without exceptions, its result moved to reg. file
Reorder buffer instruction flow

- Results first written to ROB, then to register file at commit time
Where is the register value?

- The register value can be in the register file, the ROB or bypassing path
In-order pipeline with reorder buffer

- **Decode (D):** Access regfile/ROB, allocate entry in ROB, and dispatch instruction
- **Execute (E):** Instructions can complete out-of-order
- **Completion (R):** Write result to reorder buffer
- **Retirement/Commit (W):** Check for exceptions; if none, write result to architectural register file or memory; else, flush pipeline and start from exception handler
- **In-order dispatch/execution, out-of-order completion, in-order retirement**
In-order pipeline with reorder buffer

- **Decode (D):** Access regfile/ROB, allocate entry in ROB, and dispatch instruction
- **Execute (E):** Instructions can complete out-of-order
- **Completion (R):** Write result to reorder buffer
- **Retirement/Commit (W):** Check for exceptions; if none, write result to architectural register file or memory; else, flush pipeline and start from exception handler
- **In-order dispatch/execution, out-of-order completion, in-order retirement**

![Diagram of in-order pipeline with reorder buffer](image-url)
Ensuring precise exceptions in pipelining

- Idea 3: History buffer (HB)
  - When instruction is decoded, it reserves an HB entry
  - When the instruction completes, it stores the old value of its destination in the HB
  - When instruction is oldest and no exceptions interrupts, the HB entry discarded
  - When instruction is oldest and an exception needs to be handled, old values in the HB are written back into the architectural state from tail to head
Superscalar: the essential concept that differentiates undergrad and graduate students
Introduction to Superscalar Processor

• Processors studied so far are fundamentally limited to CPI $\geq 1$
Introduction to Superscalar Processor

• Processors studied so far are fundamentally limited to CPI $\geq 1$
• Superscalar processors enable CPI $< 1$ (IPC $> 1$) by executing multiple instructions in parallel
• Can have both in-order and out-of-order superscalar processors. We will start with in-order.
Baseline 2-Way In-Order Superscalar Processor

PC - Instr. Cache - RF Read - ALU A - Branch Cond. - ALU B - Data Cache - RF Write

Pipe A: Integer Ops., Branches
Pipe B: Integer Ops., Memory
Baseline 2-Way In-Order Superscalar Processor

Fetch 2 Instructions at same time

Pipe A: Integer Ops., Branches
Pipe B: Integer Ops., Memory
Baseline 2-Way In-Order Superscalar Processor
Baseline 2-Way In-Order Superscalar Processor

- PC
  - Instr. Cache
  - addr
  - rdata
- RF Read
- Decode A
- Decode B
- ALU A
- ALU B
- Data Cache
- RF Write
- Pipe A: Integer Ops., Branches
- Pipe B: Integer Ops., Memory

Duplicate Control

Branch Cond.
Issue Logic Pipeline Diagrams

<table>
<thead>
<tr>
<th>Opcode</th>
<th>F</th>
<th>D</th>
<th>A0</th>
<th>A1</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpA</td>
<td>F</td>
<td>D</td>
<td>A0</td>
<td>A1</td>
<td>W</td>
</tr>
<tr>
<td>OpB</td>
<td>F</td>
<td>D</td>
<td>B0</td>
<td>B1</td>
<td>W</td>
</tr>
<tr>
<td>OpC</td>
<td>F</td>
<td>D</td>
<td>A0</td>
<td>A1</td>
<td>W</td>
</tr>
<tr>
<td>OpD</td>
<td>F</td>
<td>D</td>
<td>B0</td>
<td>B1</td>
<td>W</td>
</tr>
<tr>
<td>OpE</td>
<td>F</td>
<td>D</td>
<td>A0</td>
<td>A1</td>
<td>W</td>
</tr>
<tr>
<td>OpF</td>
<td>F</td>
<td>D</td>
<td>B0</td>
<td>B1</td>
<td>W</td>
</tr>
</tbody>
</table>

CPI = 0.5 (IPC = 2)

Double Issue Pipeline
Can have two instructions in same stage at same time
Dual Issue Data Hazards

No Bypassing:

ADDIU R1,R1,1  F  D  A0  A1  W
ADDIU R3,R4,1  F  D  B0  B1  W
ADDIU R5,R6,1  F  D  A0  A1  W
ADDIU R7,R5,1  F  D  D  D  D  D  A0  A1  W
Dual Issue Data Hazards

No Bypassing:
ADDIU R1,R1,1   F  D  A0  A1  W
ADDIU R3,R4,1   F  D  B0  B1  W
ADDIU R5,R6,1   F  D  A0  A1  W
ADDIU R7,R5,1   F  D  D  D  D  D  A0  A1  W

Full Bypassing:
ADDIU R1,R1,1   F  D  A0  A1  W
ADDIU R3,R4,1   F  D  B0  B1  W
ADDIU R5,R6,1   F  D  A0  A1  W
ADDIU R7,R5,1   F  D  D  A0  A1  W
Bypassing in Superscalar Pipelines
Bypassing in Superscalar Pipelines
Bypassing in Superscalar Pipelines
Bypassing in Superscalar Pipelines
Breaking Decode and Issue Stage

- Bypass Network can become very complex
- Can motivate breaking Decode and Issue Stage

D = Decode, Possibly resolve structural Hazards
I = Register file read, Bypassing, Issue/Steer Instructions to proper unit

\begin{align*}
\text{OpA} & : F \quad D \quad I \quad A0 \quad A1 \quad W \\
\text{OpB} & : F \quad D \quad I \quad B0 \quad B1 \quad W \\
\text{OpC} & : F \quad D \quad I \quad A0 \quad A1 \quad W \\
\text{OpD} & : F \quad D \quad I \quad B0 \quad B1 \quad W
\end{align*}
Dual Issue Data Hazards

Order Matters:
ADDIU R1,R1,1  F  D  A0 A1 W
ADDIU R3,R4,1  F  D  B0 B1 W
ADDIU R7,R5,1  F  D  A0 A1 W
ADDIU R5,R6,1  F  D  B0 B1 W

WAR Hazard Possible?
Fetch Logic and Alignment

<table>
<thead>
<tr>
<th>Cyc</th>
<th>Addr</th>
<th>Instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x000</td>
<td>OpA</td>
</tr>
<tr>
<td>0</td>
<td>0x004</td>
<td>OpB</td>
</tr>
<tr>
<td>1</td>
<td>0x008</td>
<td>OpC</td>
</tr>
<tr>
<td>1</td>
<td>0x00C</td>
<td>J 0x100</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0x100</td>
<td>OpD</td>
</tr>
<tr>
<td>2</td>
<td>0x104</td>
<td>J 0x204</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0x204</td>
<td>OpE</td>
</tr>
<tr>
<td>3</td>
<td>0x208</td>
<td>J 0x30C</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0x30C</td>
<td>OpF</td>
</tr>
<tr>
<td>4</td>
<td>0x310</td>
<td>OpG</td>
</tr>
<tr>
<td>5</td>
<td>0x314</td>
<td>OpH</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>0x000</th>
<th>0x100</th>
<th>0x200</th>
<th>0x300</th>
<th>0x310</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cyc</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Addr</td>
<td>0</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>Instr</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

Fetching across cache Lines is very hard. May need extra ports
## Fetch Logic and Alignment

<table>
<thead>
<tr>
<th>Cyc</th>
<th>Addr</th>
<th>Instr</th>
<th>理想，无对齐约束条件</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x000</td>
<td>OpA</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0x004</td>
<td>OpB</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0x008</td>
<td>OpC</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0x00C</td>
<td>J 0x100</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0x100</td>
<td>OpD</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0x104</td>
<td>J 0x204</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0x204</td>
<td>OpE</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0x208</td>
<td>J 0x30C</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0x30C</td>
<td>OpF</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0x310</td>
<td>OpG</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0x314</td>
<td>OpH</td>
<td></td>
</tr>
</tbody>
</table>
# With Alignment Constraints

<table>
<thead>
<tr>
<th>Cyc</th>
<th>Addr</th>
<th>Instr</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x000</td>
<td>OpA</td>
</tr>
<tr>
<td>1</td>
<td>0x004</td>
<td>OpB</td>
</tr>
<tr>
<td>2</td>
<td>0x008</td>
<td>OpC</td>
</tr>
<tr>
<td>2</td>
<td>0x00C</td>
<td>J 0x100</td>
</tr>
<tr>
<td>3</td>
<td>0x100</td>
<td>OpD</td>
</tr>
<tr>
<td>3</td>
<td>0x104</td>
<td>J 0x204</td>
</tr>
<tr>
<td>4</td>
<td>0x200</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>0x204</td>
<td>OpE</td>
</tr>
<tr>
<td>5</td>
<td>0x208</td>
<td>J 0x30C</td>
</tr>
<tr>
<td>5</td>
<td>0x20C</td>
<td>?</td>
</tr>
<tr>
<td>6</td>
<td>0x308</td>
<td>?</td>
</tr>
<tr>
<td>6</td>
<td>0x30C</td>
<td>OpF</td>
</tr>
<tr>
<td>7</td>
<td>0x310</td>
<td>OpG</td>
</tr>
<tr>
<td>7</td>
<td>0x314</td>
<td>OpH</td>
</tr>
</tbody>
</table>
### Superscalars Multiply Branch Cost

<table>
<thead>
<tr>
<th>Instruction</th>
<th>F</th>
<th>D</th>
<th>I</th>
<th>A0</th>
<th>A1</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>BEQZ</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpA</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>B0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpB</td>
<td></td>
<td>F</td>
<td>D</td>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpC</td>
<td></td>
<td>F</td>
<td>D</td>
<td>I</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpD</td>
<td>F</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpE</td>
<td>F</td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpF</td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpG</td>
<td></td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpH</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>A0</td>
<td>A1</td>
<td>W</td>
</tr>
<tr>
<td>OpI</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>B0</td>
<td>B1</td>
<td>W</td>
</tr>
</tbody>
</table>