EE 660: Computer Architecture
Out-of-Order Processors

Yao Zheng
Department of Electrical Engineering
University of Hawai‘i at Mānoa

Based on the slides of Prof. David Wentzlaff
Agenda

• I4 Processors
• I2O2 Processors
• I2OI Processors
• IO3 Processors
• IO2I Processors
Agenda

• I4 Processors
• I2O2 Processors
• I2O1 Processors
• IO3 Processors
• IO2I Processors
# Out-Of-Order (OOO) Introduction

<table>
<thead>
<tr>
<th>Name</th>
<th>Frontend</th>
<th>Issue</th>
<th>Writeback</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>I4</td>
<td>IO</td>
<td>IO</td>
<td>IO</td>
<td>IO</td>
</tr>
<tr>
<td>I2O2</td>
<td>IO</td>
<td>IO</td>
<td>OOO</td>
<td>OOO</td>
</tr>
<tr>
<td>I2O1</td>
<td>IO</td>
<td>IO</td>
<td>OOO</td>
<td>IO</td>
</tr>
<tr>
<td>I03</td>
<td>IO</td>
<td>OOO</td>
<td>OOO</td>
<td>OOO</td>
</tr>
<tr>
<td>I02I</td>
<td>IO</td>
<td>OOO</td>
<td>OOO</td>
<td>IO</td>
</tr>
</tbody>
</table>
Two independent sequences of instructions enable flexibility in terms of how instructions are scheduled in total order.

We can schedule statically in software or dynamically in hardware.
I4: In-Order Front-End, Issue, Writeback, Commit
I4: In-Order Front-End, Issue, Writeback, Commit
I4: In-Order Front-End, Issue, Writeback, Commit (4-stage MUL)

To avoid increasing CPI, needs full bypassing which can be expensive. To help cycle time, add Issue stage where register file read and instruction “issued” to Functional Unit.
I4: In-Order Front-End, Issue, Writeback, Commit (4-stage MUL)
### Basic Scoreboard

**Data Avail.**

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>F</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **P**: Pending, Write to Destination in flight
- **F**: Which functional unit is writing register
- **Data Avail.**: Where is the write data in the functional unit pipeline

- A One in Data Avail. In column ‘I’ means that result data is in stage ‘I’ of functional unit F
- Can use F and Data Avail. fields to determine when to bypass and where to bypass from
- A one in column zero means that cycle functional unit is in the Writeback stage
- Bits in Data Avail. field shift right every cycle.
# Basic Scoreboard

Data Avail.

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>F</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **P**: Pending, Write to Destination in flight
- **F**: Which functional unit is writing register
- **Data Avail.**: Where is the write data in the functional unit pipeline

- A One in Data Avail. In column ‘I’ means that result data is in stage ‘I’ of functional unit F
- Can use F and Data Avail. fields to determine when to bypass and where to bypass from
- A one in column zero means that cycle functional unit is in the Writeback stage
- Bits in Data Avail. field shift right every cycle.
0  MUL  R1, R2, R3  F  D  I  Y0  Y1  Y2  Y3  W
1  ADDIU  R11, R10, 1  F  D  I  X0  X1  X2  X3  W
2  MUL  R5, R1, R4  F  D  I  I  I  Y0  Y1  Y2  Y3  W
3  MUL  R7, R5, R6  F  D  D  D  I  I  I  I  Y0  Y1  Y2  Y3  W
4  ADDIU  R12, R11, 1  F  F  F  D  D  D  D  D  I  X0  X1  X2  X3  W
5  ADDIU  R13, R12, 1  F  F  F  F  D  I  X0  X1  X2  X3  W
6  ADDIU  R14, R12, 2  F  D  I  X0  X1  X2  X3  W

Cyc  D  I  4  3  2  1  0  Dest  Regs
1  0
2  1  0
3  2  1  1
4  1  1  1
5  1  1
6  3  2  1  1
7  1  1  1
8  1  1
9  1
10  4  3  1
11  5  4  1  1
12  6  5  1  1
13  6  1  1  1
14  1  1  1  1
15  1  1  1  1
16  1  1  1
17  1  1
18  1

RED Indicates if we look at F Field, we can bypass on this cycle
Agenda

• l4 Processors
• I2O2 Processors
• I2O1 Processors
• IO3 Processors
• IO2I Processors
I2O2: In-order Frontend/Issue, Out-of-order Writeback/Commit
I2O2 Scoreboard

• Similar to I4, but we can now use it to track structural hazards on Writeback port
• Set bit in Data Avail. according to length of pipeline
• Architecture conservatively stalls to avoid WAW hazards by stalling in Decode therefore current scoreboard sufficient. More complicated scoreboard needed for processing WAW Hazards
0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W
1 ADDIU R11, R10, 1 F D I X0 W
2 MUL R5, R1, R4 F D I I I I Y0 Y1 Y2 Y3 W
3 MUL R7, R5, R6 F D D D I I I I Y0 Y1 Y2 Y3 W
4 ADDIU R12, R11, 1 F F F D D D D D I X0 W
5 ADDIU R13, R12, 1 F F F F D I X0 W
6 ADDIU R14, R12, 2

Cyc D I
1 0
2 1 0
3 2 1
4 1 1
5 1 1
6 3 2
7 1 1
8 1
9 1
10 4 3
11 5 4
12 6 5
13 1 1
14 1 1
15 1 1
16 1

红色标记指示，如果我们查看F字段，我们可以在这一周期内跳过。

写入具有两周期延迟。结构性风险。
Early Commit Point?

0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 /
1 ADDIU R11, R10, 1 F D I X0 W /
2 MUL R5, R1, R4 F D I I I I /
3 MUL R7, R5, R6 F D D D D /
4 ADDIU R12, R11, 1 F F F /
5 ADDIU R13, R12, 1 /
6 ADDIU R14, R12, 2

• Limits certain types of exceptions.
Agenda

• I4 Processors
• I2O2 Processors
• I2O1 Processors
• IO3 Processors
• IO2I Processors
I2O1: In-order Frontend/Issue, Out-of-order Writeback, In-order Commit

PRF=Physical Register File (Future File), ROB=Reorder Buffer, FSB=Finished Store Buffer (1 entry)
## Reorder Buffer (ROB)

<table>
<thead>
<tr>
<th>State</th>
<th>S</th>
<th>ST</th>
<th>V</th>
<th>Preg</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**State:** {Free, Pending, Finished}

**S:** Speculative

**ST:** Store bit

**V:** Physical Register File Specifier Valid

**Preg:** Physical Register File Specifier
### Reorder Buffer (ROB)

<table>
<thead>
<tr>
<th>State</th>
<th>S</th>
<th>ST</th>
<th>V</th>
<th>Preg</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**State:** \{Free, Pending, Finished\}

**S:** Speculative

**ST:** Store bit

**V:** Physical Register File Specifier Valid

**Preg:** Physical Register File Specifier

- **Head of ROB**
- **Tail of ROB**
- **Speculative because branch is in flight**
- **Instruction wrote ROB out of order**
- **Next instruction allocates here in D**
- **Commit stage is waiting for Head of ROB to be finished**
Finished Store Buffer (FSB)

- Only need one entry if we only support one memory instruction inflight at a time.
- Single Entry FSB makes allocation trivial.
- If support more than one memory instruction, we need to worry about Load/Store address aliasing.
0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 ADDIU R11,R10,1 F D I X0 W r C
2 MUL R5, R1, R4 F D I I I I Y0 Y1 Y2 Y3 W C
3 MUL R7, R5, R6 F D D D I I I I Y0 Y1 Y2 Y3 W C
4 ADDIU R12,R11,1 F F F D D D D I X0 W r C
5 ADDIU R13,R12,1 F F F F D I X0 W r C
6 ADDIU R14,R12,2 F D I I X0 W r C

Cyc D I ROB 0 1 2 3

0
1 0
2 1 0
3 2 1
4
5
6 3 2
7
8
9
10 4 3
11 5 4
12 6 5
13
14 6
15
16
17
18
19

Empty = free entry in ROB

State of ROB at beginning of cycle

Pending entry in ROB

Circle=Finished (Cycle after W)

Last cycle before entry is freed from ROB (Cycle in C stage)

Entry becomes free and is freed on next cycle
What if First Instruction Causes an Exception?

0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W /
1 ADDIU R11,R10,1 F D I X0 W r -- /
2 MUL R5, R1, R4 F D I I I Y0 /
3 MUL R7, R5, R6 F D D D I /
4 ADDIU R12,R11,1 F F F D /

F D I...
What About Branches?

Option 2

0 BEQZ R1, target F D I X0 W C
1 ADDIU R11, R10, 1 F D I X0 /
2 ADDIU R5, R1, R4 F D I /
3 ADDIU R7, R5, R6 F D /
T ADDIU R12, R11, 1 F D I . . .

Option 1

0 BEQZ R1, target F D I X0 W C
1 ADDIU R11, R10, 1 F D I - Squash instructions earlier. Has more complexity. ROB needs many ports.
2 ADDIU R5, R1, R4 F D -
3 ADDIU R7, R5, R6 F -
T ADDIU R12, R11, 1 F D I . . .

Option 3

0 BEQZ R1, target F D I X0 W C
1 ADDIU R11, R10, 1 F D I X0 W / Wait for speculative instructions to reach the Commit stage and squash in Commit stage
2 ADDIU R5, R1, R4 F D I X0 W /
3 ADDIU R7, R5, R6 F D I X0 W /
T ADDIU R12, R11, 1 F D I X0 W C
What About Branches?

• Three possible designs with decreasing complexity based on when to squash speculative instructions and de-allocate ROB entry:
  1. As soon as branch resolves
  2. When branch commits
  3. When speculative instructions reach commit

• Base design only allows one branch at a time. Second branch stalls in decode. Can add more bits to track multiple in-flight branches.
Avoiding Stalling Commit on Store Miss

<table>
<thead>
<tr>
<th>0</th>
<th>OpA</th>
<th>F</th>
<th>D</th>
<th>I</th>
<th>X0</th>
<th>W</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SW</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>S0</td>
<td>W</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>OpB</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>X0</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>3</td>
<td>OpC</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>4</td>
<td>OpD</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>I</td>
<td>I</td>
<td>I</td>
</tr>
</tbody>
</table>

With Retire Stage

<table>
<thead>
<tr>
<th>0</th>
<th>OpA</th>
<th>F</th>
<th>D</th>
<th>I</th>
<th>X0</th>
<th>W</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>SW</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>S0</td>
<td>W</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>OpB</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>X0</td>
<td>W</td>
<td>C</td>
</tr>
<tr>
<td>3</td>
<td>OpC</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>X</td>
<td>W</td>
<td>C</td>
</tr>
<tr>
<td>4</td>
<td>OpD</td>
<td>F</td>
<td>D</td>
<td>I</td>
<td>X</td>
<td>W</td>
<td>C</td>
</tr>
</tbody>
</table>

CSB=Committed Store Buffer
Agenda

• I4 Processors
• I2O2 Processors
• I2O1 Processors
• IO3 Processors
• IO2I Processors
IO3: In-order Frontend, Out-of-order Issue/Writeback/Commit
## Issue Queue (IQ)

<table>
<thead>
<tr>
<th>Op</th>
<th>Imm</th>
<th>S</th>
<th>V</th>
<th>Dest</th>
<th>V</th>
<th>P</th>
<th>Src0</th>
<th>V</th>
<th>P</th>
<th>Src1</th>
</tr>
</thead>
</table>

- **Op**: Opcode
- **Imm.**: Immediate
- **S**: Speculative Bit
- **V**: Valid (Instruction has corresponding Src/Dest)
- **P**: Pending (Waiting on operands to be produced)

Instruction Ready = (!V_{src0} | | !P_{src0}) && (!V_{src1} | | !P_{src1}) && no structural hazards

- For high performance, factor in bypassing
Centralized vs. Distributed Issue Queue

Centralized

Distributed
## Advanced Scoreboard

### Data Avail.

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R31</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- P: Pending, Write to Destination in flight
- **Data Avail.**: Where is the write data in the pipeline and which functional unit

- Data Avail. now contains functional unit identifier
- A non-empty value in column zero means that cycle functional unit is in the Writeback stage
- Bits in Data Avail. field shift right every cycle.
0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W
1 ADDIU R11, R10, 1 F D I X0 W
2 MUL R5, R1, R4 F D i I Y0 Y1 Y2 Y3 W
3 MUL R7, R5, R6 F D i I Y0 Y1 Y2 Y3 W
4 ADDIU R12, R11, 1 F D i I X0 W
5 ADDIU R13, R12, 1 F D i I X0 W
6 ADDIU R14, R12, 2 F D i I X0 W

Dest/Src0/Src1, Circle denotes value present in ARF
Value bypassed so no circle, present bit
Value set present by Instruction 1 in cycle 5, W Stage
Assume All Instruction in Issue Queue

0 MUL  R1, R2, R3 F D i                I  Y0 Y1 Y2 Y3 W
1 ADDIU R11,R10,1  F D i                I  X0 W
2 MUL  R5, R1, R4  F D i                      I  Y0 Y1 Y2 Y3 W
3 MUL  R7, R5, R6  F D i                      I  Y0 Y1 Y2 Y3 W
4 ADDIU R12,R11,1  F D i              I  X0 W
5 ADDIU R13,R12,1  F D i              I  X0 W
6 ADDIU R14,R12,2  F D i              I  X0 W

• Better performance than previous?
Agenda

• I4 Processors
• I2O2 Processors
• I2O1 Processors
• IO3 Processors
• IO2I Processors
IO2I: In-order Frontend, Out-of-order Issue/Writeback, In-order Commit
0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 ADDIU R11, R10, 1 F D I X0 W r C
2 MUL R5, R1, R4 F D i I Y0 Y1 Y2 Y3 W C
3 MUL R7, R5, R6 F D i I Y0 Y1 Y2 Y3 W C
4 ADDIU R12, R11, 1 F D i I X0 W r C
5 ADDIU R13, R12, 1 F D i I X0 W r C
6 ADDIU R14, R12, 2 F D i I X0 W r C

0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 ADDIU R11, R10, 1 F D I X0 W r C
2 MUL R5, R1, R4 F D i I Y0 Y1 Y2 Y3 W C
3 MUL R7, R5, R6 F D i I Y0 Y1 Y2 Y3 W C
4 ADDIU R12, R11, 1 F D i I X0 W r C
5 ADDIU R13, R12, 1 F D i I X0 W r C
6 ADDIU R14, R12, 2 F D i I X0 W r C
Out-of-order 2-Wide Superscalar with 1 ALU

0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19

0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 ADDIU R11,R10,1 F D I X0 W r C
2 MUL R5, R1, R4 F D i I Y0 Y1 Y2 Y3 W C
3 MUL R7, R5, R6 F D i I Y0 Y1 Y2 Y3 W C
4 ADDIU R12,R11,1 F D I X0 W r C
5 ADDIU R13,R12,1 F D i I X0 W r C
6 ADDIU R14,R12,2 F D i I X0 W r C
Acknowledgements

• These slides contain material developed and copyright by:
  – Arvind (MIT)
  – Krste Asanovic (MIT/UCB)
  – Joel Emer (Intel/MIT)
  – James Hoe (CMU)
  – John Kubiatowicz (UCB)
  – David Patterson (UCB)
  – Christopher Batten (Cornell)

• MIT material derived from course 6.823
• UCB material derived from course CS252 & CS152
• Cornell material derived from course ECE 4750