Computer Architecture
ELE 475 / COS 475
Slide Deck 6: Superscalar 3

David Wentzlaff
Department of Electrical Engineering
Princeton University
Agenda

• Speculation and Branches
• Register Renaming
• Memory Disambiguation
Agenda

• Speculation and Branches
• Register Renaming
• Memory Disambiguation
Speculation and Branches: I4

0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W
1 ADDIU R4, R5, 1 F D I X0 X1 X2 X3 W
2 MUL R6, R1, R4 F D I I I Y0 Y1 Y2 Y3 W
3 BEQZ R6, Target F D D D I I I I X0 X1 X2 X3 W
4 ADDIU R8, R9 ,1 F F F D D D D I -- -- -- -- --
5 ADDIU R10,R11,1 F F F F D -- -- -- -- -- --
6 ADDIU R12,R13,1 F -- -- -- -- -- -- --
T

• No Speculative Instructions Commit State
Speculation and Branches: I2O2

0  MUL  R1, R2, R3  F D I  Y0 Y1 Y2 Y3 W
1  ADDIU R4, R5, 1  F D I  X0 W
2  MUL  R6, R1, R4  F D I  I  I  Y0 Y1 Y2 Y3 W
3  BEQZ  R6, Target  F D D D I  I  I  I  X0 W
4  ADDIU R8, R9, 1   F F F  D  D  D  D  I  --  --
5  ADDIU R10, R11, 1  F F F  F  D  --  --  --
6  ADDIU R12, R13, 1  F  --  --  --  --

• No Speculative Instructions Commit State
Speculation and Branches: I201

0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 ADDIU R4, R5, 1 F D I X0 W r C
2 MUL R6, R1, R4 F D I I I Y0 Y1 Y2 Y3 W C
3 BEQZ R6, Target F D D D I I I I I I X0 W C
4 ADDIU R8, R9,1 F F F D D D D I -- -- --
5 ADDIU R10, R11,1 F F F F D -- -- -- --
6 ADDIU R12, R13,1 F -- -- -- -- --
T F D I . . .

• Must Squash Instructions in Pipeline after Branch to prevent PRF Write.
• Can remove from ROB immediately or wait until Commit
Speculation and Branches: IO3

- No Control speculation for IO3
- Could Stall on Branch
Speculation and Branches: IO2I

0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 ADDIU R4, R5, 1 F D I X0 W r C
2 MUL R6, R1, R4 F D i I Y0 Y1 Y2 Y3 W C
3 BEQZ R6, Target F D i I X0 W C
4 ADDIU R8, R9 ,1 F D i I X0 W r --
5 ADDIU R10,R11,1 F D i I X0 W --
6 ADDIU R12,R13,1 F D i --
7 ???
8 ???
9 ???
10???
11??
T

Need to clean up Speculative state In PRF. Needs Selective Rollback

Diagram:

```
F --D
Q
I
SB
X0
L0
S0
Y0
L1
Y1
Y2
Y3
PRF
ARE
```
Speculation and Branches: IO2I

0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 ADDIU R4, R5, 1 F D I X0 W r C
2 MUL R6, R1, R4 F D i I Y0 Y1 Y2 Y3 W C
3 BEQZ R6, Target F D i I X0 W C
4 ADDIU R8, R9 ,1 F D i I X0 W r /
5 ADDIU R10,R11,1 F D i I X0 W r /
6 ADDIU R12,R13,1 F D i I X0 /
7 ???
8 ???
9 ???
10??
11??
12??
13???
T ???

• Copy ARF to PRF on Mispredict

Speculative Instructions
Wrote to PRF
Not ARF
Agenda

• Speculation and Branches
• Register Renaming
• Memory Disambiguation
WAW and WAR “Name” Dependencies

• WAW and WAR are not “True” data dependencies
• RAW is “True” data dependency because reader needs result of writer
• “Name” dependencies exist because we have limited number of “Names” (register specifiers or memory addresses)
WAW and WAR “Name” Dependencies

• WAW and WAR are not “True” data dependencies
• RAW is “True” data dependency because reader needs result of writer
• “Name” dependencies exist because we have limited number of “Names” (register specifiers or memory addresses)

Breaking all “Name” Dependencies (Causes problems)

0        MUL      R1, R2, R3 F D I   Y0 Y1 Y2 Y3 W C
1        MUL      R4, R1, R5 F D i   I   Y0 Y1 Y2 Y3 W C
2        ADDIU    R6, R4, 1    F D i    I   X0 W C
3        ADDIU    R4, R7, 1    F D i    I   X0 W r    C
WAW and WAR “Name” Dependencies

- WAW and WAR are not “True” data dependencies
- RAW is “True” data dependency because reader needs result of writer
- “Name” dependencies exist because we have limited number of “Names” (register specifiers or memory addresses)

**Breaking all “Name” Dependencies (Causes problems)**

0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 MUL R4, R1, R5 F D i I Y0 Y1 Y2 Y3 W C
2 ADDIU R6, R4, 1 F D i I X0 W C
3 ADDIU R4, R7, 1 F D i I X0 W r C
WAW and WAR “Name” Dependencies

- WAW and WAR are not “True” data dependencies
- RAW is “True” data dependency because reader needs result of writer
- “Name” dependencies exist because we have limited number of “Names” (register specifiers or memory addresses)

Breaking all “Name” Dependencies (Causes problems)

```
0  MUL  R1, R2, R3  F  D  I  Y0  Y1  Y2  Y3  W  C
1  MUL  R4, R1, R5  F  D  i  I  Y0  Y1  Y2  Y3  W  C
2  ADDIU  R6, R4, 1  F  D  i  I  X0  W  r  C
3  ADDIU  R4, R7, 1  F  D  i  I  X0  W  r  C
```
Adding More Registers

Breaking all “Name” Dependencies
0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 MUL R4, R1, R5 F D i I Y0 Y1 Y2 Y3 W C
2 ADDIU R6, R4, 1 F D i I X0 W C
3 ADDIU R4, R7, 1 F D i I X0 W r C

IO2I Microarchitecture Conservatively Stalls
0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 MUL R4, R1, R5 F D i I Y0 Y1 Y2 Y3 W C
2 ADDIU R6, R4, 1 F D i I X0 W C
3 ADDIU R4, R7, 1 F D D D D D D D D D D I X0 W C

Manual Register Renaming. What if we could use more registers? Second R4 Write to R8?
0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 MUL R4, R1, R5 F D i I Y0 Y1 Y2 Y3 W C
2 ADDIU R6, R4, 1 F D i I X0 W C
3 ADDIU R8, R7, 1 F D i I X0 W r C
Register Renaming

• Adding more "Names" (registers/memory) removes dependence, but architecture namespace is limited.
  – Registers: Larger namespace requires more bits in instruction encoding. 32 registers = 5 bits, 128 registers = 7 bits.

• **Register Renaming**: Change naming of registers in hardware to eliminate WAW and WAR hazards
Register Renaming Overview

• 2 Schemes
  – Pointers in the Issue Queue/ReOrder Buffer
  – Values in the Issue Queue/ReOrder Buffer

• IO2I Uses pointers in IQ and ROB therefore start with that design.
IO2I: Register Renaming with Pointers in IQ and ROB

- All data structures same as in IO2I Except:
  - Add two fields to ROB
  - Add Rename Table (RT) and Free List (FL) of registers
- Increase size of PRF to provide more register “Names”
IO2I: Register Renaming with Pointers in IQ and ROB

Table:

<table>
<thead>
<tr>
<th>ARF</th>
<th>SB</th>
<th>PRF</th>
<th>ROB</th>
<th>FSB</th>
<th>IQ</th>
<th>RT</th>
<th>FL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R/W</td>
<td>R</td>
<td>R/W</td>
<td>R/W</td>
<td>W</td>
<td>R/W</td>
<td>R/W</td>
</tr>
</tbody>
</table>

Diagram:

- FL
- RT
- SB
- IQ
- X0
- L0
- L1
- S0
- Y0
- Y1
- Y2
- Y3
- PRF
- ARF
- ROB
- FSB
- C
## Modified Reorder Buffer (ROB)

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

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

**S**: Speculative

**ST**: Store bit

**V**: Destination is valid

**Preg**: Physical Register File Specifier

**Areg**: Architectural Register File Specifier

**Ppreg**: Previous Physical Register
### Rename Table (RT)

<table>
<thead>
<tr>
<th></th>
<th>P</th>
<th>Preg</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**P**: Pending, Write to Destination in flight  
**Preg**: Physical Register Architectural Register maps to.
Free List (FL)

<table>
<thead>
<tr>
<th></th>
<th>Free</th>
</tr>
</thead>
<tbody>
<tr>
<td>p1</td>
<td></td>
</tr>
<tr>
<td>p2</td>
<td></td>
</tr>
<tr>
<td>p3</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>pN</td>
<td></td>
</tr>
</tbody>
</table>

**Free**: Register is free for renaming

If Free == 0, physical register is in use and cannot be used for renaming
0 MUL R1, R2, R3 F D I Y0 Y1 Y2 Y3 W C
1 MUL R4, R1, R5 F D I Y0 Y1 Y2 Y3 W C
2 ADDIU R6, R4, 1 F D I X0 W C
3 ADDIU R4, R7, 1 F D I X0 W r C
Freeing Physical Registers

\[ \text{ADDU } R1,R2,R3 \quad \text{<-Assume Arch. Reg } R1 \text{ maps to Phys. Reg } p0 \]
\[ \text{ADDU } R4,R1,R5 \]
\[ \text{ADDU } R1,R6,R7 \quad \text{<-Next write of Arch Reg } R1, \text{ Mapped to Phys. Reg } p1 \]
\[ \text{ADDU } R8,R9,R10 \]

\[
\begin{array}{cccccc}
0 & \text{ADDU } R1,R2,R3 & \text{I} & \text{X} & \text{W} & \text{C} \\
1 & \text{ADDU } R4,R1,R5 & & & & \text{I} & \text{X} & \text{W} & \text{C} \\
2 & \text{ADDU } R1,R6,R7 & & \text{I} & \text{X} & \text{W} & \text{r} & \text{C} \\
3 & \text{ADDU } R8,R9,R10 & & \text{F} & \text{D} & \text{I} & \text{X} & \text{W} & \text{r} & \text{C} \\
\end{array}
\]

Write \( p0 \)  Free \( p0 \)  Alloc \( p0 \)  Write \( p0 \)  Read Wrong value in \( p0 \)

\[
\begin{array}{cccccc}
0 & \text{ADDU } R1,R2,R3 & \text{I} & \text{X} & \text{W} & \text{C} \\
1 & \text{ADDU } R4,R1,R5 & & & & \text{I} & \text{X} & \text{W} & \text{C} \\
2 & \text{ADDU } R1,R6,R7 & & \text{I} & \text{X} & \text{W} & \text{r} & \text{C} \\
3 & \text{ADDU } R8,R9,R10 & & \text{F} & \text{D} & \text{I} & \text{X} & \text{W} & \text{r} & \text{C} \\
\end{array}
\]

Write \( p0 \)  Alloc \( p2 \)  Write \( p2 \)  Dealloc \( p0 \)

- If Arch. Reg \( Ri \) mapped to Phys. Reg \( pj \), we can free \( pj \) when the next instruction that writes \( Ri \) commits
Unified Physical/Architectural Register File

• Combine PRF and ARF into one register file
• Replace ARF with Architectural Rename Table
• Instead of copying **Values**, Commit stage copies Preg pointer into appropriate entry of Architectural Rename Table
• Unified Physical/Architectural Register file can be smaller than separate
IO2I: Register Renaming with Values in IQ and ROB

- All data structures same as previous Except:
  - Modified ROB (Values instead of Register Specifier)
  - Modified RT
  - Modified IQ
  - No FL
  - No PRF, values merged into ROB
## Modified Reorder Buffer (ROB)

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

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

**S**: Speculative

**ST**: Store bit

**V**: Destination is valid

**Value**: Actual Register Value

**Areg**: Architectural Register File Specifier
## Modified 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>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></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>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

If Pending, Source Field contains index into ROB. Like a Preg identifier.

On Commit, Source Field contains value.
Modified Rename Table (RT)

<table>
<thead>
<tr>
<th>V</th>
<th>P</th>
<th>Preg</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R31</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

V: Valid Bit
P: Pending, Write to Destination in flight
Preg: Index into ROB

V:
If V == 0:
  Value in ARF is up to date
If V == 1:
  Value is in-flight or in ROB

P:
If P == 0:
  Value is in ROB
if P == 1:
  Value is in flight
0  MUL  R1, R2, R3  F  D  I  Y0  Y1 Y2 Y3  W  C
1  MUL  R4, R1, R5  F  D  i  I  Y0  Y1 Y2 Y3  W  C
2  ADDIU  R6, R4, 1  F  D  i  I  X0  W  C
3  ADDIU  R4, R7, 1  F  D  i  I  X0  W  r  C
Agenda

• Speculation and Branches
• Register Renaming
• Memory Disambiguation
Memory Disambiguation

\[ \text{st } R1, \ 0(R2) \]
\[ \text{ld } R3, \ 0(R4) \]

When can we execute the load?
In-Order Memory Queue

• Execute all loads and stores in program order

=> Load and store cannot leave IQ for execution until all previous loads and stores have completed execution

• Can still execute loads and stores speculatively, and out-of-order with respect to other (non-memory) instructions

• Need a structure to handle memory ordering...
IO2I: With In-Order LD/ST IQ
Conservative OOO Load Execution

\[
\text{st R1, 0(R2)} \\
\text{ld R3, 0(R4)}
\]

- Split execution of store instruction into two phases: address calculation and data write
- Can execute load before store, if addresses known and \( r4 \neq r2 \)
- Each load address compared with addresses of all previous uncommitted stores \( \text{(can use partial conservative check i.e., bottom 12 bits of address)} \)
- Don’t execute load if any previous store address not known

\( \text{(MIPS R10K, 16 entry address queue)} \)
Address Speculation

\[ \text{st } R1, \ 0(R2) \]
\[ \text{ld } R3, \ 0(R4) \]

• Guess that \( r4 \neq r2 \)

• Execute load before store address known

• Need to hold all completed but uncommitted load/store addresses in program order

• If subsequently find \( r4==r2 \), squash load and \textit{all} following instructions

=> Large penalty for inaccurate address speculation
IO2I: With OOO Load and Stores
Memory Dependence Prediction

(Alpha 21264)

\[
\text{st } r1, (r2) \\
\text{ld } r3, (r4)
\]

• Guess that r4 != r2 and execute load before store

• If later find r4==r2, squash load and all following instructions, but mark load instruction as \textit{store-wait}

• Subsequent executions of the same load instruction will wait for all previous stores to complete

• Periodically clear \textit{store-wait} bits
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
Speculative Loads / Stores

Just like register updates, stores should not modify the memory until after the instruction is committed.

- A speculative store buffer is a structure introduced to hold speculative store data.
Speculative Store Buffer

- On store execute:
  - mark entry valid and speculative, and save data and tag of instruction.
- On store commit:
  - clear speculative bit and eventually move data to cache
- On store abort:
  - clear valid bit
Speculative Store Buffer

- If data in both store buffer and cache, which should we use?
  Speculative store buffer
- If same address in store buffer twice, which should we use?
  Youngest store older than load