Superscalar* Club Meeting #2

*we really mean: superscalar speculative out-of-order

James C. Hoe
Department of ECE
Carnegie Mellon University
Today’s Goal:
One step closer to understanding R10K
References Used

  • Yeager, MIPS R10K Superscalar Microprocessor, 1996.
Agree on Terminology btw Us (not a universal agreement)
Program State Views

Figure 5-1. Illustration of In-Order, Lookahead, and Architectural States from [Johnson, 1990]
Execution Stages

ICache

Fetch

Decode

Rename

Dispatch

Reorder Buffer

Issue Queue

L/S Queue

DCache

Registers

Execution

Write-back

Complete

Retire

Not shown rename table, or register alias table, or register map table

or active list

or reservation station

operand fetch from [Gonzalez, et al., 2010]
Generic Mental Model

- **fetch**
- **decode**
- **rename tbl**

**RS**
- dispatch
- schedule
- issue

**regfile** (inorder & lookahead)

- **ALU1**
- **ALU2**
- **LD/ST**

**ROB**
- complete
- writeback

**release dependents**

[Internal Use Draft] Superscalar Club Meeting #2, Slide 7, James C. Hoe, CMU/ECE/CALCM, ©2022 [Do not redistribute.]
Review Tomasulo+ROB
H&P
WAW Example:

\[
\begin{align*}
  i: & \quad R4 \leftarrow R0 \times R8 \\
  j: & \quad R2 \leftarrow R0 + R4 \\
  k: & \quad R4 \leftarrow R0 + R8 \\
  l: & \quad R8 \leftarrow R4 \times R8
\end{align*}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
& RS & dst & Qj & Vj & Qk & Vk \\
\hline
1 & & & & & &  \\
2 & & & & & &  \\
3 & & & & & &  \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
& RS & dst & Qj & Vj & Qk & Vk \\
\hline
4 & & & & & &  \\
5 & & & & & &  \\
\hline
\end{array}
\]

Add

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
& RS & dst & Qj & Vj & Qk & Vk \\
\hline
1 & & & & & &  \\
2 & & & & & &  \\
3 & & & & & &  \\
\hline
\end{array}
\]

Mult

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
& RS & dst & Qj & Vj & Qk & Vk \\
\hline
4 & & & & & &  \\
5 & & & & & &  \\
\hline
\end{array}
\]

Mult

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
& RS & dst & Qj & Vj & Qk & Vk \\
\hline
1 & & & & & &  \\
2 & & & & & &  \\
3 & & & & & &  \\
\hline
\end{array}
\]

Add

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
& RS & dst & Qj & Vj & Qk & Vk \\
\hline
4 & & & & & &  \\
5 & & & & & &  \\
\hline
\end{array}
\]

Mult

\[
\begin{array}{c|c|c|c|c|c|c}
\hline
& RS & dst & Qj & Vj & Qk & Vk \\
\hline
1 & & & & & &  \\
2 & & & & & &  \\
3 & & & & & &  \\
\hline
\end{array}
\]

Add
assume 1-cyc add, 2-cyc mult
issue up to 1 add & 1 mult per cycle
complete up to 1 add & 1 mult per cycle

\begin{align*}
\text{cyc1:} \\
\text{RS} & \quad \text{dst} & Q_j & V_j & Q_k & V_k \\
1 & ii & 0 & 6.0 & i & - \\
2 & & & & & \\
3 & & & & & \\
\text{RS} & \quad \text{dst} & Q_j & V_j & Q_k & V_k \\
4 & i & 0 & 6.0 & 0 & 7.8 \\
5 & & & & & \\
\text{Mult} & \quad \text{issue 4} \\
\text{cyc2:} \\
\text{Add} \\
\text{RS} & \quad \text{dst} & Q_j & V_j & Q_k & V_k \\
1 & ii & 0 & 6.0 & i & - \\
2 & iii & 0 & 6.0 & 0 & 7.8 \\
3 & & & & & \\
\text{RS} & \quad \text{dst} & Q_j & V_j & Q_k & V_k \\
4 & i & 0 & 6.0 & 0 & 7.8 \\
5 & iv & iii & - & 0 & 7.8 \\
\text{Mult} \\
\text{cyc3:} \\
\text{Add} & \quad \text{issue 2} \\
\text{RS} & \quad \text{dst} & Q_j & V_j & Q_k & V_k \\
1 & ii & 0 & 6.0 & i & - \\
2 & iii & 0 & 6.0 & 0 & 7.8 \\
3 & & & & & \\
\text{RS} & \quad \text{dst} & Q_j & V_j & Q_k & V_k \\
4 & i & 0 & 6.0 & 0 & 7.8 \\
5 & iv & iii & - & 0 & 7.8 \\
\text{Mult} & \quad \text{cmpl} <i, 46.8> \\
\end{align*}
**assume 1-cyc add, 2-cyc mult**

**issue up to 1 add & 1 mult per cycle**

**complete up to 1 add & 1 mult per cycle**

---

**cyc3:**

<table>
<thead>
<tr>
<th>RS dst Qj Vj Qk Vk</th>
<th>RS dst Qj Vj Qk Vk</th>
<th>RF bsy rob value</th>
<th>ROB dst rdy value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i: R4 ← R0 x R8</td>
<td>ii: R2 ← R0 + R4</td>
<td>i: R4 0 -</td>
<td>i: R4 1 46.8</td>
</tr>
<tr>
<td>j: R2 ← R0 + R4</td>
<td>iii: R4 ← R0 + R8</td>
<td>ii: R2 0 -</td>
<td>ii: R2 0 -</td>
</tr>
<tr>
<td>k: R4 ← R0 + R8</td>
<td>iv: R8 ← R4 x R8</td>
<td>iii: R4 0 -</td>
<td>iii: R4 1 13.8</td>
</tr>
</tbody>
</table>

---

**cyc4:**

<table>
<thead>
<tr>
<th>RS dst Qj Vj Qk Vk</th>
<th>RS dst Qj Vj Qk Vk</th>
<th>RF bsy rob value</th>
<th>ROB dst rdy value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i: R4 ← R0 x R8</td>
<td>ii: R2 ← R0 + R4</td>
<td>i: R4 0 -</td>
<td>i: R4 1 46.8</td>
</tr>
<tr>
<td>j: R2 ← R0 + R4</td>
<td>iii: R4 ← R0 + R8</td>
<td>ii: R2 0 -</td>
<td>ii: R2 0 -</td>
</tr>
<tr>
<td>k: R4 ← R0 + R8</td>
<td>iv: R8 ← R4 x R8</td>
<td>iii: R4 0 -</td>
<td>iii: R4 1 13.8</td>
</tr>
</tbody>
</table>

---

**cyc5:**

<table>
<thead>
<tr>
<th>RS dst Qj Vj Qk Vk</th>
<th>RS dst Qj Vj Qk Vk</th>
<th>RF bsy rob value</th>
<th>ROB dst rdy value</th>
</tr>
</thead>
<tbody>
<tr>
<td>i: R4 ← R0 x R8</td>
<td>ii: R2 ← R0 + R4</td>
<td>i: R4 0 -</td>
<td>i: R4 1 46.8</td>
</tr>
<tr>
<td>j: R2 ← R0 + R4</td>
<td>iii: R4 ← R0 + R8</td>
<td>ii: R2 0 -</td>
<td>ii: R2 0 -</td>
</tr>
<tr>
<td>k: R4 ← R0 + R8</td>
<td>iv: R8 ← R4 x R8</td>
<td>iii: R4 0 -</td>
<td>iii: R4 1 13.8</td>
</tr>
</tbody>
</table>

---

Add cmpl <i>, 46.8>

Add cmpl <iii>, 13.8>

Mult cmpl <i>, 46.8>

Mult issue 5

Add cmpl <ii>, 52.8>
Metaflow DRIS
Metaflow Lightning SPARC Processor

- Superscalar fetch, issue, and execution
- Micro-dataflow instruction scheduling
- Register renaming + memory renaming
- Speculative execution with rapid rewinding
- Precise Exceptions

circa 1991

Claimed “Factor of 2-3 performance advantage from architecture”
Metaflow Datapath

- branch pred.
- i-cache
- issue
- DRIS (Renaming + Reservation Stations + Reorder Buff.)
- Retire
- Register File
- Scheduler

- lookahead state
- inorder state
ROB Rename Registers

from [Gonzalez, et al., 2010]

Register Map Table

Log. Reg.

ROB RF  ROB pointer

Architectural Register File

Value

comitted
inorder state

Reorder Buffer

Value

physically one
SRAM array

look-ahead
state

to “form” architectural state for decode

Recall
DRIS is ROB-like

- Circular Queue Structure
- Instructions held in original program order
  - new entry allocated at tail of queue when instruction issues
  - completed entry committed inorder from head of queue to update regfile or memory
Color Bit

- Head and tail pointers count around N-entry DRIS using $1 + \log_2(N)$ bit
  - bottom $\log_2(N)$ index bits work the way you think
  - top bit color bit alternate each round through
- Given indices $i$ and $j$, $i$ older than $j$ when
  
  $\text{if (color}(i)==\text{color}(j))$ then $\text{index}(i) < \text{index}(j)$
  
  else $\text{index}(i) \geq \text{index}(j)$
DRIS is also everything else

- **ROB**: inorder record of in-flight instructions
- **Rename Regfile**: Data holds lookahead state of RD
- **Rename Table (CAM-based)**: given operand rs, search RD youngest to oldest for re-definition
- **Reservation Station**: if !(Lock1 || Lock2)
- **Address Queue**: ... 

<table>
<thead>
<tr>
<th>Source 1</th>
<th>Source 2</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock1</td>
<td>Lock2</td>
<td>latest</td>
</tr>
<tr>
<td>RN1</td>
<td>RN2</td>
<td>RD</td>
</tr>
<tr>
<td>ID1</td>
<td>ID2</td>
<td>Data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dispatched</td>
</tr>
<tr>
<td>Fxn Unit</td>
</tr>
<tr>
<td>Executed</td>
</tr>
<tr>
<td>PC</td>
</tr>
</tbody>
</table>

Know everything, look everywhere philosophy . . .
“Issue”: (Decode+Dispatch+Rename)

• A new ID (aka tag) is allocated to each instruction when issued into DRIS
  – ID is index to next free DRIS circular queue entry
• Search DRIS (young to old) to see if $rs_{1/2}$ matches $RD$ of older entry $i$.
  – if found, set $ID_{1/2}$ to $i$; set $Lock_{1/2}$ if $!Executed[i]$
  – if not found, set $ID_{1/2}$ to (?) ; set $RN_{1/2}=rs_{1/2}$

<table>
<thead>
<tr>
<th>Source 1</th>
<th>Source 2</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock1</td>
<td>Lock2</td>
<td>latest</td>
</tr>
<tr>
<td>RN1</td>
<td>RN2</td>
<td>RD</td>
</tr>
<tr>
<td>ID1</td>
<td>ID2</td>
<td>Data</td>
</tr>
</tbody>
</table>
Rename: add rd, rs, rt

Let’s be a little more precise. You give it a try first.
**Rename: add rd, rs, rt**

Assume *new* is ID of current instruction

<table>
<thead>
<tr>
<th>RN1[\textit{new}]</th>
<th>rs</th>
<th>RN2[\textit{new}]</th>
<th>rt</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{Locked1[\textit{new}]}</td>
<td>false</td>
<td>\textit{Locked2[\textit{new}]}</td>
<td>false</td>
</tr>
<tr>
<td>ID1[\textit{new}]</td>
<td>“not _valid”</td>
<td>ID2[\textit{new}]</td>
<td>“not _valid”</td>
</tr>
</tbody>
</table>

forall valid \textit{id} // over all active DRIS entries

```plaintext
if (((RD[\textit{id}] == rs) && Latest[\textit{id}] )

ID1[\textit{new}] = \textit{id} ;
Locked1[\textit{new}] = !Executed[\textit{id}] ;
```

forall valid \textit{id}

```plaintext
if (((RD[\textit{id}] == rt) && Latest[\textit{id}] )

ID2[\textit{new}] = \textit{id} ;
Locked2[\textit{new}] = !Executed[\textit{id}] ;
```

<table>
<thead>
<tr>
<th>Source 1</th>
<th>Source 2</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock1</td>
<td>RN1</td>
<td>ID1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lock2</td>
<td>RN2</td>
<td>ID2</td>
</tr>
<tr>
<td>latest</td>
<td>RD</td>
<td>Data</td>
</tr>
</tbody>
</table>
Associative Lookup for Just 1 Rename

Need round-robin priority encoder to resolve multiple hits if not for Latest
Superscalar Rename

- Replicate the previous circuit, 2 per instruction
- All new entries read and update DRIS together
  - replicated circuits all rename relative to same “architectural” view (i.e., from same tail ptr)
  - what happens for 3-wide rename:
    
    | Instruction 1 | Instruction 2 | Instruction 3 |
    |---------------|---------------|---------------|
    | add r1, r2, r3 | add r1, r1, r3 |
    | add r4, r5, r6 | add r1, r1, r4 |
    | add r7, r8, r9 | add r1, r1, r5 |
  
- Must do on-the-fly RAW dependence check and re-direct, $O(N^2)$ complexity
Rest is Easy: add rd, rs, rt

RD[new] = rd ;
forall valid id

if (RD[id] == rd)
    Latest[id] = false ;
Latest[new] = true ;
Dispatched[new] = false ;
Executed[new] = false ;
FxnU[new] = Integer ALU ;

Source 1
Lock1 | RN1  | ID1 | Source 2
Lock2 | RN2  | ID2 | Destination
latest | RD  | Data

Dispatched  Fxn Unit  Executed  PC
Micro-Dataflow Scheduling

- The scheduler dispatches according to
  - availability of pending instructions’ operands
  - availability of the functional units
  - chronological order of the instructions

Is oldest-first the “best” strategy?

- Find instructions such that

\[
\text{valid}[id] \&\& \text{Locked1}[id] \&\& \text{Locked2}[id] \&\& \text{Dispatched}[id] \&\& \text{Executed}[id] \&\& \text{notBusy}(\text{fxnUnit}[id])
\]

Think about the circuits & multiply for superscalar

yet one more CAM
**Issue:** add rd, rs, rt

- A issued instruction is sent to the functional unit with its operands and its *id*
- Operands come from:
  - DRIS: Data[ID1/2[id]] when ID1/2[id] is younger than head ptr
    - *in lookahead state*
  - Regfile: Regfile[RN1/2[id]] when ID1/2[id] is older than head ptr (*id* may have retired since)
    - *in inorder state*

<table>
<thead>
<tr>
<th>Source 1</th>
<th>Source 2</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock1</td>
<td>RN1</td>
<td>ID1</td>
</tr>
</tbody>
</table>
Writeback/Complete

- A Fxn unit returns both the **result** and the associated **id**
  
  \[
  \text{Data}[id] = \text{result} ; \\
  \text{Executed}[id] = true ; \\
  \]

- Unlock RAW dependent instructions
  - forall **valid** \(i\)
    - if (ID1[i] == id) Locked1[i] = false;
  - forall **valid** \(i\)
    - if (ID2[i] == id) Locked2[i] = false;

<table>
<thead>
<tr>
<th>Source 1</th>
<th>Source 2</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock1</td>
<td>Lock2</td>
<td>latest</td>
</tr>
<tr>
<td>RN1</td>
<td>RN2</td>
<td>RD</td>
</tr>
<tr>
<td>ID1</td>
<td>ID2</td>
<td>Data</td>
</tr>
</tbody>
</table>

[Internal Use Draft] Superscalar Club Meeting #2, Slide 27, James C. Hoe, CMU/ECE/CALCM, ©2022 [Do not redistribute.]
Retire

- Instructions retire from DRIS inorder from oldest
  - must be **Dispatched**, wait if not **Executed**
  - not on wrongpath
  - no older exceptions, itself could be
- Commit lookahead state to inorder state
  \[\text{Regfile}[\text{RD}[\text{retiree}]] = \text{Data}[\text{retiree}]\]
- Similarly, SW can only write memory when retiring
Logical vs. Physical

Reservation Stations

Source 1
- Lock1
- RN1
- ID1

(CAM) (RAM) (CAM)

issue

forward

Source 2
- Lock2
- RN2
- ID2

(CAM) (RAM) (CAM)

Destination

latest
RD
Data

(CAM*)

“inverse”

map
table

Status

Dispatched
Fxnn Unit
Executed
PC

Reorder Buffer (RAMs)

Rename Registers (RAM)
The Cost of Implementing DRIS

- To support \( N\)-way rename per cycle
  - \( N \times 3 \) associative lookup and read
  - \( N \times 2 \) indexed read, \( N \times 2 \) indexed write
- To support \( N\)-way issue per cycle
  - 1 prioritized associative lookup of \( N \) entries
  - \( N \) indexed write
  - \( N \times 2 \) indexed read in DRIS
  - \( N \times 2 \) indexed read in Regfile
- To support \( N\)-way complete \textit{per cycle}
  - \( N \) indexed write to DRIS
  - \( N \times 2 \) associative lookup and write in DRIS
- To support \( N\)-way commit per cycle
  - \( N \) indexed lookup in DRIS
  - \( N \) indexed write to DRIS
  - \( N \) indexed write to Regfile
Why CAM and not MAP Table?

from [Gonzalez, et al., 2010]

What happens on exception and branch rewind?
Precise Exceptions

- On exception, stop fetching
- Wait until exception oldest in DRIS, set tail ptr to head ptr
- Done!

*Does this work for branch rewind?*
Rewind cannot wait for the head/oldest

- Set tail ptr to after mispredicted branch to restart . . .
- What about Latest?

Branch Stack
(more next wk)

oldest
mispredict
youngest

this
can’t
be
SRAM
To Sum Up

- Superscalar, speculative, out-of-order made “easy” by centralizing bookkeeping (if you could know everything at the same time)
- Circuits involved (gratuitous use of large CAM especially) inefficient (size and critical path) for what need to be accomplished
- On branch rewind
  - identifying younger instructions occupying out-of-order structures by index-comparison too costly
  - how to recovery the “latest” column?
- What is IPC when ILP=1? (paper doesn't say but it is handled the right way)
Issue and Forwarding Timing

from [Gonzalez, et al., 2010]
Scheduling Consumer of Loads

What if LW misses?

from [Gonzalez, et al., 2010]
Forwarding Paths Required

from [Gonzalez, et al., 2010]
Forwarding Path Complexity

from [Gonzalez, et al., 2010]
Read R10K very, very carefully for next week