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

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

[Internal Use Draft] Superscalar Club Meeting #2, Slide 6, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]
Generic Mental Model

- **Fetch**
- **Decode**
- **Rename tbl**
- **Dispatch**
- **Schedule**
- **Issue**

**ROB**
- **Release dependents**
- **Complete**
- **Writeback**

**RS**
- **Regfile** (inorder & lookahead)

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

[Diagram showing the flow of data from Register Map Table to Architectural Register File, with annotations: from [Gonzalez, et al., 2010], committed inorder state, look-ahead state.]

Recall

to “form” architectural state for decode
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

```
0
1
2
3
.  
.  
.  
.  

oldest

youngest

N-2
N-1
```
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
  
  \[
  \text{if} \ (\text{color}(i)=\text{color}(j)) \ \text{then} \ \text{index}(i) < \text{index}(j) \\
  \text{else} \ \text{index}(i) \geq \text{index}(j)
  \]
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>RN1</td>
<td>ID1</td>
</tr>
<tr>
<td>latest</td>
<td>RD</td>
<td>Data</td>
</tr>
<tr>
<td>Dispatched</td>
<td>Fxn Unit</td>
<td>Executed</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 rs1/2 matches RD of older entry $i$.
  - if found, set ID1/2 to $i$; set Lock1/2 if !Executed[$i$]
  - if not found, set ID1/2 to (?) ; set RN1/2=rs1/2

<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.*

<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 16, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]
Rename: add rd, rs, rt

Assume new is ID of current instruction

\[
\begin{align*}
\text{RN1}[	ext{new}] &= \text{rs} ; \\
\text{RN2}[	ext{new}] &= \text{rt} ; \\
\text{Locked1}[	ext{new}] &= \text{false} ; \\
\text{Locked2}[	ext{new}] &= \text{false} ; \\
\text{ID1}[	ext{new}] &= \text{"not_valid"} ; \\
\text{ID2}[	ext{new}] &= \text{"not_valid"} ;
\end{align*}
\]

forall valid id // over all active DRIS entries

if ((\text{RD}[id] == \text{rs}) \&\& \text{Latest}[id])

\[
\begin{align*}
\text{ID1}[	ext{new}] &= \text{id} ; \\
\text{Locked1}[	ext{new}] &= \text{!Executed}[id] ;
\end{align*}
\]

forall valid id

if ((\text{RD}[id] == \text{rt}) \&\& \text{Latest}[id])

\[
\begin{align*}
\text{ID2}[	ext{new}] &= \text{id} ; \\
\text{Locked2}[	ext{new}] &= \text{!Executed}[id] ;
\end{align*}
\]
Associative Lookup for Just 1 Rename

Return My Tag

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:
    
    add r1, r2, r3
    add r4, r5, r6
    add r7, r8, r9

    add r1, r1, r3
    add r1, r1, r4
    add r1, r1, r5

• Must do on-the-fly RAW dependence check and re-direct, $O(N^2)$ complexity
Rest is Easy: \textit{add rd, rs, rt}

\begin{align*}
\text{RD}[\text{new}] &= \text{rd} ; \\
\text{forall valid id} &\rightarrow \text{one more CAM}
\end{align*}

\begin{align*}
\text{if} (\text{RD}[\text{id}] &= \text{rd}) \\
\text{Latest}[\text{id}] &= \text{false} ; \\
\text{Latest}[\text{new}] &= \text{true} ; \\
\text{Dispatched}[\text{new}] &= \text{false} ; \\
\text{Executed}[\text{new}] &= \text{false} ; \\
\text{FxnU}[\text{new}] &= \text{Integer ALU} ;
\end{align*}
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


  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
  - Regfile: Regfile[RN1/2[id]] when ID1/2[id] is older than head ptr (*id* may have retired since)

<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>

*in lookahead state*

*in inorder state*
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 \( i \)
    \[ \text{if (ID1}[i]=id) \text{Locked1}[i]=false; \]
  – forall valid \( i \)
    \[ \text{if (ID2}[i]=id) \text{Locked2}[i]=false; \]

<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 23, James C. Hoe, CMU/ECE/CALCM, ©2021 [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

Destination

latest  RD  Data
(CAM*)
“inverse” map table

Status

Dispatched  Fxn Unit  Executed  PC

Rename Registers
(RAM)

Reorder Buffer (RAMs)
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 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
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)

<table>
<thead>
<tr>
<th>Latest</th>
<th>Latest</th>
<th>Latest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>oldest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mispredict</th>
</tr>
</thead>
<tbody>
<tr>
<td>this</td>
</tr>
<tr>
<td>can’t</td>
</tr>
<tr>
<td>be</td>
</tr>
<tr>
<td>SRAM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>youngest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

[Internal Use Draft] Superscalar Club Meeting #2, Slide 28, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]
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 CAM especially) inefficient (size and critical path) for what it needs to accomplish
- What is IPC when ILP=1? (Paper doesn't say but it is handled the right way)
Issue and Forwarding Timing

1. WakeUp signal received when value becomes available

Producer:
- Wake-Up
- Select
- Drive
- Execution
- WriteBack

Consumer:
- Wake-Up
- Select
- Drive
- Execution

2. WakeUp signal received 3 cycles before becomes available

Producer:
- Wake-Up
- Select
- Drive
- Execution
- WriteBack

Consumer:
- Wake-Up
- Select
- Drive
- Execution

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

1. Conservative WakeUp signal generated after Hit/Miss Computation
   - Producer: Wake-Up, Select, Drive, Address computation, Hit/Miss computation, L1 cache Access
   - Consumer: Wake-Up, Select, Drive, Execution

2. Speculative WakeUp of load consumers
   - Producer: Wake-Up, Select, Drive, Address computation, Hit/Miss computation, L1 cache Access
   - Consumer: Wake-Up, Select, Drive, Execution

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]
Why CAM and not MAP Table?

What happens on exception and branch rewind?

[Internal Use Draft] Superscalar Club Meeting #2, Slide 34, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]
Read R10K very, very carefully for next week