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]](image-url)
Execution Stages

- Execution Stages
  - dispatch
  - from [Gonzalez, et al., 2010]
  - Not shown rename table, aka register alias table, aka register map table

- Execution Stages
  - ICache
  - L/S Queue
  - DCache
  - Registers
  - Execution
  - Write-back
  - Commit

- Execution Stages
  - Fetch
  - Decode
  - Rename
  - Issue
  - Issue Queue
  - Reorder Buffer
  - operand fetch
  - complete
  - retire

- Execution Stages
  - aka active list
  - register alias table
  - register map table

- Execution Stages
  - dispatch
  - aka reservation station
Generic Mental Model

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

- **ROB**
  
  **RS**
    
    **regfile** (inorder & lookahead)
      
      - **ALU1**
      - **ALU2**
      - **LD/ST**

- **issue**
- **schedule**
- **release dependents**

- **complete**
- **writeback**
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

to “form” architectural state for decode

Architectural Register File

Value

committed
inorder state

Reorder Buffer

Value

physically one SRAM array

look-ahead state

[Internal Use Draft] Superscalar Club Meeting #2, Slide 11, James C. Hoe, CMU/ECE/CALCM, ©2023 [Do not redistribute.]
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
- Instruction’s position in ROB is its unique tag
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
  
  if ($\text{color}(i) == \text{color}(j)$) 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>

Status

- Dispatched
- Fxn Unit
- Executed
- PC

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 \( RN_{1/2} = rs_{1/2} \)
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>
Rename: \textbf{add rd, rs, rt}

Assume \textit{new} is ID of current instruction

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

\begin{align*}
\text{forall valid id} & \quad \text{// over all active DRIS entries} \\
\text{if ((RD[id] == rs) && Latest[id])} & \quad \text{ID1[^{new}]} = \text{id}; \\
& \quad \text{Locked1[^{new}]} = \lnot \text{Executed[id]}; \\
\text{forall valid id} & \quad \text{if ((RD[id] == rt) && Latest[id])} \\
& \quad \text{ID2[^{new}]} = \text{id}; \\
& \quad \text{Locked2[^{new}]} = \lnot \text{Executed[id]}; \\
\end{align*}
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:
    - `add r1, r2, r3` vs `add r1, r1, r3`
    - `add r4, r5, r6` vs `add r1, r1, r4`
    - `add r7, r8, r9` vs `add r1, r1, r5`
- Must do on-the-fly RAW dependence check and re-direct, $O(N^2)$ complexity
Rest is Easy: \textbf{add rd, rs, rt}

\begin{verbatim}
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 ;
\end{verbatim}

Source 1
\begin{tabular}{|c|c|c|}
  \hline
  Lock1 & RN1 & ID1 \\
  \hline
\end{tabular}
Source 2
\begin{tabular}{|c|c|c|}
  \hline
  Lock2 & RN2 & ID2 \\
  \hline
\end{tabular}
Destination
\begin{tabular}{|c|c|c|}
  \hline
  latest & RD & Data \\
  \hline
  Dispatched & Fxn Unit & Executed & PC \\
  \hline
\end{tabular}
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? What is?

- Find instructions such that
  
  \[
  \text{valid}[\text{id}] \&\& \neg \text{Locked1}[\text{id}] \&\& \neg \text{Locked2}[\text{id}] \&\&
  \neg \text{Dispatched}[\text{id}] \&\& \neg \text{Executed}[\text{id}] \&\& \neg \text{notBusy}(\text{fxnUnit}[\text{id}])
  \]

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 (producer was already retired at decode or has since retired)
    *in inorder state*
Writeback/Complete

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

• Unlock RAW dependent instructions
  
  – forall valid \texttt{i}
    
    if (\text{ID1}[i] == \texttt{id}) \text{Locked1}[i] = \text{false} ;

  – forall valid \texttt{i}
    
    if (\text{ID2}[i] == \texttt{id}) \text{Locked2}[i] = \text{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, ©2023 [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

<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 24, James C. Hoe, CMU/ECE/CALCM, ©2023 [Do not redistribute.]
Logical vs. Physical

Reservation Stations

Source 1
- Lock1
- RN1
- ID1

Source 2
- Lock2
- RN2
- ID2

Destination
- latest
- RD
- Data

(CAM)*(CAM*)

issue
forward

additional complications

“inverse” map table

Status
- Dispatched
- Fxn 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 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]

Register Map Table

Log. Reg. -> ROB/RF ROB pointer

to “form” architectural state for decode

Architectural Register File

Value

Committed inorder state

Reorder Buffer

Value

Look-ahead state

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)

- This
- Can’t
- Be
- Vanilla
- CAM

oldest
mispredict
youngest

[Internal Use Draft] Superscalar Club Meeting #2, Slide 29, James C. Hoe, CMU/ECE/CALCM, ©2023 [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 too elaborate (gratuitous use of large CAM especially) and 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