Superscalar* Club Meeting #3

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

James C. Hoe
Department of ECE
Carnegie Mellon University
References Used

- Yeager, MIPS R10K Superscalar Microprocessor, 1996.
Today’s Goal:
Really get under the R10K paper
This should mean something to you

```
This should mean something to you

fetch

decode

rename tbl

RS

issue

disable

regfile (inorder & lookahead)

ALU1

ALU2

LD/ST

ROB

release dependents

complete

writeback

[Internal Use Draft] Superscalar Club Meeting #3, Slide 4, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]```
Last Time: Metaflow DRIS
Metaflow Datapath

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

lookahead state
inorder state
Not Unreasonable if . . .

• Separate RS and address queue from DRIS/ROB
  – RS sized to expose ILP
    Can’t be large: CAM-intensive, critical timing loop
  – ROB sized to cover long latencies (cache miss)
    Modern ROB size much larger than RS size

• Use a map-table for rename, keeping in mind
  – cheaper but not exactly cheap
  – still need to see how to rewind a map-table (see branch rewind stack today)

End up looking like Pentium-Pro
Onto MIPS R10K
MIPS R10000 circa 1996

- 4-way superscalar
- 5 execution pipelines
  - 2 integer, FP add, FP mult, ld/st
- Micro-dataflow instruction scheduling
  - 16 int +16 FP instruction window
- Register renaming + memory renaming
  - 64 int registers for inorder and lookahead
- Speculative OOO
  - 32 instructions in-flight; 4 unresolved branches
- Precise Exception
Superscalar, Speculative, Out-of-order

[Fig 2, Yeager 1996, IEEE Micro]
Pipeline Stages

6 independent pipelines

Execution unit pipelines (5)

Dynamic issue

Floating-point latency=2

Load/store latency=2

Integer latency=1

Stage 1 | Stage 2 | Stage 3 | Stage 4 | Stage 5 | Stage 6 | Stage 7
---|---|---|---|---|---|---
Issue | RF→ | Alignment | Add | Pack | →RF
Issue | RF→ | Multiply | Sum product | Pack | →RF
Issue | RF→ | ACalc | TLB, D-cache
Queues

Load | →RF (Integer or FP)

Write results in register file

Busy?

Fetch | Dec | Map | Write

I-cache | Branch address

Read operands from register file

Instruction fetch and decode pipeline fills queues
4 instructions in parallel
Up to 4 branch instructions are predicted
Fetching continues speculatively until prediction verified

[Fig 2, Yeager 1996, IEEE Micro]

[Internal Use Draft] Superscalar Club Meeting #3, Slide 11, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]
Let’s Talk About

• Register renaming

• Instruction Scheduling

• Speculative Execution Rewind
On-the-fly HW Register Renaming

- Maintain mapping from ISA reg. names to physical registers
- When decoding an instruction that updates ‘rₓ’:
  - allocate unused physical register tᵧ to hold inst result
  - set new mapping from ‘rₓ’ to tᵧ
  - younger instructions using ‘rₓ’ as input finds tᵧ
- De-allocate a physical register for reuse when it is never needed again?

---

**ISA name**
e.g. **r12**

**rename table**

**rename**

**physical registers**

(t0 ... t63)

---

^^^^^^when is this exactly?
Rename: add rd, rs, rt

Assume *new* is ID of current instruction

\[
\begin{align*}
RN1[\text{new}] &= rs; \quad RN2[\text{new}] = rt; \\
Locked1[\text{new}] &= false; \quad Locked2[\text{new}] = false; \\
ID1[\text{new}] &= \text{not\_valid}; \quad ID2[\text{new}] = \text{not\_valid}; \\
\text{forall valid id} & \quad // \text{over all active DRIS entries} \\
\text{if } ((RD[\text{id}] == rs) \land (Latest[\text{id}]) ) \\
ID1[\text{new}] &= id; \\
Locked1[\text{new}] &= \neg \text{Executed}[\text{id}]; \\
\text{forall valid id} & \\
\text{if } ((RD[\text{id}] == rt) \land (Latest[\text{id}]) ) \\
ID2[\text{new}] &= id; \\
Locked2[\text{new}] &= \neg \text{Executed}[\text{id}];
\end{align*}
\]
Elements of Register Renaming

• A pool of extra registers
  – Use as temporary, single-assignment registers in lookahead state *(eliminates WAW and WAR)*
  – logically separate from inorder committed state

• Allocation and mapping mechanism
  – given a source architectural reg name, where is its current definition *(value, location, ready?)*
  – given a dest architectural reg name, where to find an available new rename register
  – when to reclaim a rename register?
  – how to recover after misprediction or exception
ROB Rename Registers

from [Gonzalez, et al., 2010]

Need to copy from lookahead to inorder on commit

[to “form” architectural state for decode]
Physical Register File Rename

No need to copy from lookahead to inorder on commit

from [Gonzalez, et al., 2010]
For Example Intel P3 vs P4

Figure 5: Pentium® III vs. Pentium® 4 processor register allocation
[The Microarchitecture of the Pentium 4 Processor, Intel Technology Journal, 2001]
PReg Life Cycle, R10K

- at any moment, each *preg* index (*ptag*) must be in exactly one entry of *map-table*, valid *free*, or valid *last*
- # *preg* (and freelist size) can be decoupled from ROB
- Steps 1 and 2 need to be reversible
Easier Than You Think

• # physical register (preg) =
  # arch reg (=32)
+ # ROB entries (=32 in R10K)

• At any moment
  – 32 preg hold committed
    inorder state
  – rest associated 1 per ROB
    entry---either in-use
    (lookahead dest) or not in-
    use (freelist)

• Freelist management can ride ROB’s coattail

[Internal Use Draft] Superscalar Club Meeting #3, Slide 20, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]
Management Algorithm

- At rename/dispatch
  - rename `rd` to corresponding ROB/freelist `preg`
  - save in ROB `rd`’s previous `preg` mapping (*read from map-table before updating*)
  - if no dest or `rd=r0`, save unused new `preg` as `last`
- At commit
  - current dest `preg` ⇒ inorder
  - write `last` mapping into freelist (*deallocated*)
- On rewind? On exception?

[Internal Use Draft] Superscalar Club Meeting #3, Slide 21, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]
Register Map-table

- To rename **rs**, look up **ptag** and **busy**
- R10K map-table needs 4-way x (3 read + 1 write port)!!
  *Also need redirect across same-cycle renamed instructions*
  
- On rewind, map-table restored by **Branch Rewind Stack**
- On exception, map-table restored sequentially from **last**
Let’s Talk About

• Register renaming

• **Instruction Scheduling**

• Speculative Execution Rewind
Dataflow Execution Ordering

- Maintain a buffer of many pending instructions, a.k.a. reservation stations (RSs)
  - wait for functional unit to be free
  - wait for register RAW hazards to resolve (i.e., required input operands to be produced)
- Issue instructions for execution in dataflow order
  - select instructions in RS whose operands are available
  - give preference to older instructions (heuristical)
- A completing instruction frees pending, RAW-dependent instructions to execute

Sounds like good plan but exactly how?
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

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

  Think about the circuits & multiply for superscalar
R10K Integer Queue

• Like Tomasulo’s Reservation Stations but without operand value
  – operands represented by renamed \texttt{ptag} and \texttt{busy} status
  – an instruction issues when operands ready (either in regfile or can be forwarded in time)

• Keep in mind, \texttt{busy} is cleared when dependent-on instruction \texttt{selected for issue} not when it \texttt{completes}
Basic Integer Timing (Best Case)

(station 2) Request
(stations 3) Issue
(stations 4) Operands
(stations 4) Execute

- I₁’s new data dep. info latched
- I₁ requests issue
- I₁ granted issue
- I₁ fetches operands (RF or forwarding)
- I₁ announces data dep. resolution (to I₂)
- I₁’s operand latched
- I₁ fetches operands (RF or forwarding)
- I₁ announces data dep. resolution (to I₂)
- I₁’s result written to RF
- I₁’s result forwarded to I₂
- dataflow resolution
Integer Queue Entry

How many CAM ports do you count?

Priority scheduler 1-per-cyc

Request grant

Fetch operand (RF or forward)

Issue/op, stage 3

Execute, stage 4
Scheduling Loop Critical Path

How many CAM ports do you count?

priority scheduler 1-per-cyc

request grant

issue/op, stage 3
execute, stage 4

fetch operand (RF or forward)
Forwarding

- ALU₁ result
- ALU₂ result
- Load result
- Integer reg file
- Integer queue
- ptag

Stage 3
Stage 4

[Internal Use Draft] Superscalar Club Meeting #3, Slide 30, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]
Load Data Path
Load Timing if Hit

Request

Issue

Operands

Addr Calc

D-cache/TLB

Execute

$L_0$ requests issue

$L_0$ granted issue if tag and data array free next cycle

$L_0$’s result forwarded to $I_1$

$L_0$’s result presented to cache & TLB

hit data

dataflow resolution 1 cyc later

[Internal Use Draft] Superscalar Club Meeting #3, Slide 32, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]
Load Timing if Miss

- Request: \( L_0 \)
- Issue: \( L_0 \)
- Operands: \( L_0 \)
- Addr Calc: \( L_0 \)
- D-cache/TLB

\( L_0 \)'s address presented to cache & TLB

\( L_0 \) requests issue
\( L_0 \) granted issue if tag and data array free next cycle

dependence resolution 1 cyc later

miss signal

\( L_0 \) will rerun later

L0 will cancel and lose 1 issue cycle since too late to find an alternate
Address Queue

from IntQ

from IntQ

to IntQs

stage 3: issue

stage 4: addr calculation

stage 5: cache lookup

stage 6: Writeback

[Internal Use Draft] Superscalar Club Meeting #3, Slide 34, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]
“Tentative” Release and Cancellation

- IntQ Dest
- IntQ Dest
- Load Dest (tentative)

OpA

set reset

Rdy

set reset
tentative

cancel

issue
decision

set 1-cyc cancellation window if released by load

cache miss

[Internal Use Draft] Superscalar Club Meeting #3, Slide 35, James C. Hoe, CMU/ECE/CALCM, ©2021 [Do not redistribute.]
Let’s Talk About

• Register renaming

• Instruction Scheduling

• Speculative Execution Rewind
Rewinding Tomasulo with ROB

- Inorder RF state never needs undo’ing ✓
- Lookahead RF state tied to ROB ✓
- Restoring architectural state view (i.e. map-table)
  - at decode, record in ROB, the logical dest and the overwritten previous mapping
  - on rewind, walk-back ROB one entry at a time to restore register map-table

What happens if previous mapping is to an already retired ROB entry? How do we know that?

from [Gonzalez, et al., 2010]
Rewind cannot wait for the head/oldest

- Set tail ptr to after mispredicted branch to restart . . .
- What about \textbf{Latest}?

Branch Stack
(more next wk)

\begin{itemize}
  \item \textit{oldest}
  \item \textit{mispredict}
  \item \textit{youngest}
\end{itemize}

```
from 0002
```
**Control Flow Speculation**

- **Leading Speculation**
  - follow through multiple branch predictions
  - track speculative instructions as lookahead
  - preserve μarch state at branch dispatch for rewind
Mis-speculation Recovery

- When a branch is evaluated, if prediction confirmed, nothing more to do (except to deallocate no-longer-needed recovery state)
- Else use recovery state (deallocate after use)
  - clear wrongpath instructions and their effects
  - restart down “correct” path
Branch Rewind Stack (BRS)

- Not a stack; not a monolithic structure
  - allocate a slot when a branch dispatch
  - deallocate when branch resolve (right or wrong)
  - deallocate when branch (on wrongpath) killed

- A BRS slot snapshots at branch dispatch
  - tail (but not head) pointer of ROB
  - head (but not tail) pointer of freelist (if decoupled)
  - complete map-table (*Map-table cannot be vanilla multiported SRAM*)

- R10K superscalar dispatch stops after the first branch in a cycle; continues rest next cycle
Rewinding Out-of-Order Entities

- A bitmask indicate currently allocated BRS slots
  - each set-bit corresponds to an unresolved branch
  - a speculative, out-of-order entity picks up bitmask value at time of its creation—*need to be removed if any of the indicated branches mispredicts*

- Examples of speculative out-of-order entities
  - instructions in RS or anywhere else not ROB
  - a BRS slot

- A resolved branch broadcasts its BRS position
  - ignored by older entities, bit not set in their mask
  - caught by younger entities, bit is set in their mask
Reset and Abort

- On misprediction, any out-of-order entity with branch mask intersecting with the abort mask is eliminated.
- On correct prediction, the corresponding bit (reset mask) is cleared in **ALL** branch masks in system—
  corresponding BRS slot free for reallocation to another branch.
What about Exception

- Different from branch misprediction rewind
  - could occur at any instruction (not just branches)
  - doesn’t happen very frequently
  - only handled as the oldest instruction in ROB
- Easy to clear younger instructions once exception is oldest in ROB:
  - rewind ROB tail pointer (and freelist head pointer)
  - zap all out-of-order structures and state
- No backup map-table at all exception points
  - R10K reads back sequentially from last of ROB
  - Intel P4 maintains a retirement map-table
Before Next Time

• Memory Dataflow
  – Section 6.4 of Gonzalez, et al.
  – “Address Queue”, R10K, p34.
• https://github.com/jhoecmu/ooo-beta
  – download and build
  – run in a debugger to step through a few instructions’ worth of operations
  – drill into a specific structure, e.g., map table