CS425
Computer Systems Architecture

Fall 2019

Re-Order Buffer:
Precise Exceptions and Speculation
Scoreboard Architecture (CDC 6600)
Tomasulo Organization
<table>
<thead>
<tr>
<th>Tomasulo</th>
<th>Scoreboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipelined Functional Units</td>
<td>Multiple Functional Units</td>
</tr>
<tr>
<td>(6 load, 3 store, 3 +, 2 ( \times ) / ( \div ))</td>
<td>(1 load/store, 1 +, 2 ( \times ), 1 ( \div ))</td>
</tr>
<tr>
<td>window size: ≤ 14 instructions</td>
<td>≤ 5 instructions</td>
</tr>
<tr>
<td>No issue on structural hazard</td>
<td>same</td>
</tr>
<tr>
<td>WAR: renaming avoids them</td>
<td>stall completion</td>
</tr>
<tr>
<td>WAW: renaming avoids them</td>
<td>stall issue</td>
</tr>
<tr>
<td>Broadcast results from FU</td>
<td>Write/read registers</td>
</tr>
<tr>
<td>Control: reservation stations</td>
<td>Central scoreboard</td>
</tr>
</tbody>
</table>
Exception Behavior with ROB

CPI = CPI\textsubscript{IDEAL} + Stalls\textsubscript{STRUC} + Stalls\textsubscript{RAW} + Stalls\textsubscript{WAR} + Stalls\textsubscript{WAW} + Stalls\textsubscript{CONTROL}

• Have to maintain:
  – Data Flow
  – Exception Behavior

<table>
<thead>
<tr>
<th>Dynamic instruction scheduling (HW)</th>
<th>Static instruction scheduling (SW/compiler)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scoreboard (reduce RAW stalls)</td>
<td>Loop Unrolling</td>
</tr>
<tr>
<td>Register Renaming (reduce WAR &amp; WAW stalls)</td>
<td>SW pipelining</td>
</tr>
</tbody>
</table>
  • Tomasulo                          |                                            |
  • Reorder buffer                    |                                            |
| Branch Prediction (reduce control stalls) | Trace Scheduling                            |
| Multiple Issue (CPI < 1)            |                                            |
| Multithreading (CPI < 1)            |                                            |
Device Interrupt

Network Interrupt

add r1,r2,r3
subi r4,r1,#4
slli r4,r4,#2

Hiccup(!)

lw r2,0(r4)
lw r3,4(r4)
add r2,r2,r3
sw 8(r4),r2

Raise priority
Disable All Ints
Save registers

lw r1,20(r0)
lw r2,0(r1)
addi r3,r0,#5
sw 0(r1),r3

Restore registers
Clear current Int
Re-enable All Ints
Restore priority
RTE

Or Could be interrupted by disk

Note that priority must be raised to avoid recursive interrupts!
Types of Interrupts/Exceptions

- I/O device request
- Invoking an operating system service from a user program
- Breakpoint (programmer-requested interrupt)
- Integer arithmetic overflow
- FP arithmetic anomaly
- Page fault (not in main memory)
- Misaligned memory accesses (if alignment is required)
- Memory protection violation
- Using an undefined or unimplemented instruction
- Hardware malfunctions
- Power failure
Precise Interrupts/Exceptions

- An interrupt or exception is precise if there is an instruction (or interrupt point) for which:
  - All instructions before this instruction have fully completed
  - None of the instructions after this instruction (including the interrupting instruction) has modified the machine state

- This means that we can restart the execution from the interrupt point and still “get the correct results”
  - In the example: the Interrupt point is the `lw` instruction
Imprecise Interrupt/Exception

• An exception is imprecise if the processor state when an exception is raised does not look exactly as if the instructions were executed sequentially in strict program order.

• Occurrence in two possibilities:
  – The pipeline may have already completed instructions that are later in program order.
  – The pipeline may have not yet completed some instructions that are earlier in program order.
Precise interrupt point requires multiple PCs when there are delayed branches

```
addi r4, r3, #4  
sub  r1, r2, r3  
```

Interrupt point described as <PC, PC+4>

```
bne r1, there  
and r2, r3, r5  
```

Interrupt point described as:

```
addi r4, r3, #4  
sub  r1, r2, r3  
```

<PC+4, there> (branch was taken)

or

<PC+4, PC+8> (branch was not taken)
Why do we need precise interrupts?

• Several interrupts/exceptions need to be restartable
  – i.e. TLB faults. Fix translation and then restart the faulting load/store
  – IEEE gradual underflow, illegal operation,

  
  \[ f(x) = \frac{\sin(x)}{x} \]

  \[ x \to 0 \quad f(0) = \frac{0}{0} \Rightarrow NaN + \text{illegal operation} \]

  Want to take exception, replace NaN with 1, then restart.

• Restartability does not require precisionness. However, precisionness makes restarts much simpler

• Simplifies the Operating System (OS)
  – Less state needs to be saved away if unloading process.
  – Quick to restart (for fast interrupts)
Precise Exceptions in 5-stage DLX

• Exceptions may occur in different stages of the processor pipeline (i.e. out of order):
  – Arithmetic exceptions occur in execution stage
  – TLB faults can occur in instruction fetch or memory stage

• How do we guarantee precise exceptions? Mark the instructions with an “exception status” and wait until the WB stage to take the exception
  – Interrupts are marked as NOPs (like bubbles) that are placed into pipeline instead of an instruction.
  – Assume that interrupt condition persists in case NOP flushed
  – Clever instruction fetch might start fetching instructions from interrupt vector, but this is complicated and needs to switch to supervisor mode, saving of one or more PCs, etc
Another look at the exception problem

- Use the pipeline!
  - Each instruction has an exception status field.
  - Keep the PCs for every instruction in the pipeline.
  - Check the exception status when the instruction reaches the WB stage

- When an instruction reaches the WB stage and has an exception:
  - Save PC ⇒ EPC, Interrupt vector addr ⇒ PC
  - Convert all fetched instructions to NOPs

- It works because of in-order completion/WB
# Scoreboard Example: Cycle 62

## Instruction status:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Oper</th>
<th>Comp</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+</td>
<td>R2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+</td>
<td>R3</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>6</td>
<td>9</td>
<td>19</td>
</tr>
<tr>
<td>SUBD</td>
<td>F8</td>
<td>F6</td>
<td>F2</td>
<td>7</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td>8</td>
<td>21</td>
<td>61</td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td>13</td>
<td>14</td>
<td>16</td>
</tr>
</tbody>
</table>

### Functional unit status:

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Fi</th>
<th>Fj</th>
<th>Fk</th>
<th>Qj</th>
<th>Qk</th>
<th>Rj</th>
<th>Rk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Integer</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Divide</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Register result status:

<table>
<thead>
<tr>
<th>Clock</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
<th>F10</th>
<th>F12</th>
<th>...</th>
<th>F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>62</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Tomasulo Example: Cycle 57

**Instruction status:**

<table>
<thead>
<tr>
<th>Instruction</th>
<th>j</th>
<th>k</th>
<th>Issue</th>
<th>Comp</th>
<th>Result</th>
<th>Busy</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>F6</td>
<td>34+</td>
<td>R2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>Load1</td>
</tr>
<tr>
<td>LD</td>
<td>F2</td>
<td>45+</td>
<td>R3</td>
<td>2</td>
<td>4</td>
<td>5</td>
<td>Load2</td>
</tr>
<tr>
<td>MULTD</td>
<td>F0</td>
<td>F2</td>
<td>F4</td>
<td>3</td>
<td>15</td>
<td>16</td>
<td>Load3</td>
</tr>
<tr>
<td>SUBD</td>
<td>F8</td>
<td>F6</td>
<td>F2</td>
<td>4</td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>DIVD</td>
<td>F10</td>
<td>F0</td>
<td>F6</td>
<td>5</td>
<td>56</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>ADDD</td>
<td>F6</td>
<td>F8</td>
<td>F2</td>
<td>6</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

**Reservation Stations:**

<table>
<thead>
<tr>
<th>Time</th>
<th>Name</th>
<th>Busy</th>
<th>Op</th>
<th>Vj</th>
<th>Vk</th>
<th>Qi</th>
<th>Qk</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Add1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add2</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Add3</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult1</td>
<td>No</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mult2</td>
<td>Yes</td>
<td>DIVD</td>
<td>M*F4</td>
<td>M(A1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Register result status:**

<table>
<thead>
<tr>
<th>Clock</th>
<th>F0</th>
<th>F2</th>
<th>F4</th>
<th>F6</th>
<th>F8</th>
<th>F10</th>
<th>F12</th>
<th>...</th>
<th>F30</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**FU Result**

| Reg File | M*F4 | M(A2) | (M-M+M) | (M-M) |

---

**In-order issue**

**Out-of-order execute**

**Out-of-order commit!**
Issue: “Fetch” unit

- Instructions from a potentially mispredicted branch path have been already executed.
- Instruction fetch decoupled from execution

Return to the right path when the outcome of the branch is known
Branch must execute fast for loop overlap!

• In the loop-unrolling example, we assume that the branches are executed from a “fast” integer unit to achieve overlap!

```
Loop:   LD       F0   0   R1
       MULTD   F4   F0   F2
       SD       F4   0   R1
       SUBI   R1   R1   #8
       BNEZ   R1   Loop
```

• What happens if the branch depends on the outcome of MULTD?
  – We lose all benefits
  – We have to predict the outcome of the branch
  – If we predict “taken” the prediction would be correct most of the time.
Prediction: Branches, Dependencies, Data

• Branch Prediction is necessary for good performance
• We studied branches in the previous lecture. Modern architectures now predict many things: data dependencies, actual data, and results of groups of instructions

• Why does prediction work?
  – Underlying algorithm has regularities.
  – Data that is being operated on has regularities.
  – Instruction sequence has redundancies that are artifacts of way that humans/compilers think about problems.
Problem: Out-of-Order Completion

• Scoreboard and Tomasulo operate as follows:
  – In-order issue, out-of-order execution, out-of-order completion

• We need a way to synchronize the completion stage of instructions with the program order (i.e. with issue-order)
  – Easiest way is with in-order completion (i.e. re-order buffer)
  – Other Techniques (Smith paper): Future File, History Buffer
Precise Interrupts and Speculation:

• During the Issue stage of instructions we operate as if we are predicting that all previous instructions do not generate exceptions
  – Branch prediction, data prediction
  – If we speculate and are wrong, need to back up and restart execution to the point at which we predicted incorrectly
  – This is exactly same as precise exceptions!

• Common technique for precise interrupts/exceptions and speculation: **in-order completion or commit**
  – All modern out-of-order processors typically use a form of re-order buffer (ROB)
HW support for precise interrupts/exceptions

• Idea behind Reorder Buffer (ROB): keep the instructions in a FIFO, with the exact order that they are issued.
  – Each ROB entry contains PC, dest reg/mem, result, exception status
• When an instruction completes execution then the results are placed in the allocated entry in the ROB.
  – Supplies operands to other instruction between execution complete & commit ⇒ more registers like RS
  – Tag results with ROB buffer number instead of reservation station
• The instructions change the machine state at the commit stage not on the WB ⇒ in order commit ⇒ values at head of ROB are placed in registers
• This technique allows us to cancel/squash speculatively executed instructions during mispredicted branches or exceptions
HW Support for Reorder Buffer (ROB)?

- How do we find the last “version” of each register?
- Multi-ported ROB like the register file
- Integrate store buffer into ROB since we have in order commit. Stores use Result field for ROB tag until data ready on CDB.
- Can we also integrate the reservation stations?
Tomasulo with ROB: Basic Block Diagram
Four Stages of Tomasulo with ROB

1. **Issue**: Get Instruction from Op Queue
   - If there are free reservation stations and reorder buffer slot, issue instr & send operands & reorder buffer no. for destination (sometimes called “dispatch”)

2. **Execution**: Execute the Instruction in the Execution Unit (EX)
   - When the values of the 2 source regs are ready then execute the instruction; otherwise, watch CDB for result; when both in reservation station, execute; checks RAW (“issue”)

3. **Write result**: End of Execution (WB)
   - Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

4. **Commit**: Update the dst reg with the value from the reorder buffer
   - When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch or exception flushes reorder buffer. (also called “graduation” or “retirement”)
Tomasulo With Reorder buffer

FP Op Queue

Reorder Buffer

FP adders

FP multipliers

Reservation Stations

Registers

Dest shows #ROB

Dest

Commit ptr

ROB7
ROB6
ROB5
ROB4
ROB3
ROB2
ROB1

To Memory

from Memory

ROB1

1

10+R2

Dest

Ready

Instr. Type

Value

Dest.

L.D F0,10(R2)

N

F0

Oldest

Newest

L.D F0,10(R2)

Dest.

10+R2

CS495, Vassilis Papaefstathiou
Reorder buffer (after 2 cycles)

Reorder Buffer

<table>
<thead>
<tr>
<th>Dest Value</th>
<th>Instr Type</th>
<th>Ready</th>
</tr>
</thead>
<tbody>
<tr>
<td>F2</td>
<td>DIV.D F2,F10,F6</td>
<td>N</td>
</tr>
<tr>
<td>F10</td>
<td>ADD.D F10,F4,F0</td>
<td>N</td>
</tr>
<tr>
<td>F0</td>
<td>L.D F0,10(R2)</td>
<td>N</td>
</tr>
</tbody>
</table>

Registers

FP adders

FP multipliers

Reservation Stations

Dest | FP adders | FP multipliers | Reservation Stations |
--- | --- | --- | --- |
2   | ADDD R(F4),ROB1 | | |
3   | DIVD ROB2,R(F6) | | |
1   | 10+R2 | | |

“2” means ROB2

FP Op Queue

To Memory

From Memory

Commit ptr

Newest

Oldest

CS485: Vassilis Papaefstathiou
Reorder buffer (after 3 cycles)

<table>
<thead>
<tr>
<th>Dest Value</th>
<th>Instr. Type</th>
<th>Ready</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0</td>
<td>ADD.D F0,F4,F6</td>
<td>N</td>
</tr>
<tr>
<td>F4</td>
<td>L.D F4,0(R3)</td>
<td>N</td>
</tr>
<tr>
<td>--</td>
<td>BNE F2,&lt;…&gt;</td>
<td>N</td>
</tr>
<tr>
<td>F2</td>
<td>DIV.D F2,F10,F6</td>
<td>N</td>
</tr>
<tr>
<td>F10</td>
<td>ADD.D F10,F4,F0</td>
<td>N</td>
</tr>
<tr>
<td>F0</td>
<td>L.D F0,10(R2)</td>
<td>N</td>
</tr>
</tbody>
</table>

Reorder Buffer

FP Op Queue

Registers

FP adders

Reservation Stations

FP multipliers

Commit ptr

To Memory

from Memory

10+R2
0+R3
Reorder buffer (after 1 cycle)

Reorder Buffer

FP Op Queue

Dest Value

Instr. Type

Ready

ROB5

S.D 0(R3),F4

N

ROB5

ADD.D F0,F4,F6

N

ROB5

L.D F4,0(R3)

N

ROB5

BNE F2,<…>

N

ROB5

DIV.D F2,F10,F6

N

ROB5

ADD.D F10,F4,F0

N

ROB5

L.D F0,10(R2)

N

Dest

ADD.D R(F4),ROB1

ADD.D ROB5, R(F6)

Dest

DIV.D ROB2,R(F6)

Dest

10+R2

0+R3

FP adders

Reservation Stations

FP multipliers

To Memory

from Memory

FP Op Queue

Registers

Oldest

Newest

Commit ptr

CS485 - Vassilis Papaefstathiou
Tomasulo With Reorder buffer

Reorder Buffer

FP Op Queue

Registers

FP adders

FP multipliers

Reservation Stations

Dest | Instr. Type | Ready | Destination Value
--- | --- | --- | ---
R3 | M[A1] | Y | S.D 0(R3),F4
F0 | <val2> | Ex | ADD.D F0,F4,F6
F4 | M[A1] | Y | L.D F4,0(R3)
--- | -- | N | BNE F2,<…>
F2 | | N | DIV.D F2,F10,F6
F10 | | N | ADD.D F10,F4,F0
F0 | | N | L.D F0,10(R2)

To Memory

from Memory

Commit ptr

Newest

Oldest

CS485 - Vassilis Papaefstathiou
Tomasulo With Reorder buffer

**FP Op Queue**

**Reorder Buffer**

**Registers**

**FP adders**

**FP multipliers**

**Reservation Stations**

<table>
<thead>
<tr>
<th>Dest</th>
<th>Value</th>
<th>Instr. Type</th>
<th>Ready</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3</td>
<td>M[A1]</td>
<td>S.D 0(R3),F4</td>
<td>Y</td>
</tr>
<tr>
<td>F0</td>
<td>&lt;val2&gt;</td>
<td>ADD.D F0,F4,F6</td>
<td>Ex</td>
</tr>
<tr>
<td>F4</td>
<td>M[A1]</td>
<td>L.D F4,0(R3)</td>
<td>Y</td>
</tr>
<tr>
<td>--</td>
<td></td>
<td>BNE F2,&lt;…&gt;</td>
<td>N</td>
</tr>
<tr>
<td>F2</td>
<td></td>
<td>DIV.D F2,F10,F6</td>
<td>N</td>
</tr>
<tr>
<td>F10</td>
<td></td>
<td>ADD.D F10,F4,F0</td>
<td>N</td>
</tr>
<tr>
<td>F0</td>
<td>M[A2]</td>
<td>L.D F0,10(R2)</td>
<td>Y</td>
</tr>
</tbody>
</table>

**To Memory**

**From Memory**

**Commit ptr**
Tomasulo With Reorder buffer

FP Op Queue

Reorder Buffer

Commit ptr

Registers

FP adders

Reservation Stations

FP multipliers

dest | instr.type | value | ready |
--- | --- | --- | --- |
R3 | M[A1] | S.D 0(R3),F4 | Y |
F0 | <val2> | ADD.D F0,F4,F6 | Ex |
F4 | M[A1] | L.D F4,0(R3) | Y |
F2 | -- | BNE F2,<...> | N |
F10 | -- | DIV.D F2,F10,F6 | N |
F0 | -- | ADD.D F10,F4,F0 | N |

CS485 - Vassilis Papaefstathiou 31
Tomasulo With Reorder buffer

Reorder Buffer

Dest. Value | Instr. Type | Ready
---|---|---
R3 | M[A1] | Y
F0 | <val2> | Ex

Registers

Dest

FP adders

FP multipliers

Reservation Stations

FP Op Queue

Commit ptr

To Memory

from Memory

Oldest

Newest

ROB7
ROB6
ROB5
ROB4
ROB3
ROB2
ROB1

Dest

ROB7
ROB6
ROB5
ROB4
ROB3
ROB2
ROB1

CS485 - Vassilis Papaefstathiou 32
Reorder buffer: Precise Exceptions

### Reorder Buffer

<table>
<thead>
<tr>
<th>Dest. Value</th>
<th>Instr. Type</th>
<th>Ready</th>
<th>ROB7</th>
<th>ROB6</th>
<th>ROB5</th>
<th>ROB4</th>
<th>ROB3</th>
<th>ROB2</th>
<th>ROB1</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>M[10]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F0</td>
<td>&lt;val2&gt;</td>
<td>Ex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F4</td>
<td>M[10]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--</td>
<td>BNE F2,&lt;&gt;</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F2</td>
<td>DIV.D F2,F10,F6</td>
<td>Ex</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F10</td>
<td>ADD.D F10,F4,F0</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F0</td>
<td>L.D F0,10(R2)</td>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Registers

- FP Op Queue
- FP adders
- FP multipliers
- Reservation Stations

### Many Exceptions?

- FP adders
- FP multipliers
- Reservation Stations

### To Memory

- Memory
- Oldest

### From Memory

- Oldest
Reorder buffer: Branch Misprediction

<table>
<thead>
<tr>
<th>Dest. Value</th>
<th>Instr. Type</th>
<th>Ready</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3 M[A1]</td>
<td>S.D 0(R3),F4</td>
<td>Y</td>
</tr>
<tr>
<td>F0</td>
<td>ADD.D F0,F4,F6</td>
<td>Ex</td>
</tr>
<tr>
<td>F4 M[A1]</td>
<td>L.D F4,0(R3)</td>
<td>Y</td>
</tr>
<tr>
<td>--</td>
<td>BNE F2,&lt;...&gt;</td>
<td>Y</td>
</tr>
<tr>
<td>F2</td>
<td>DIV.D F2,F10,F6</td>
<td>N</td>
</tr>
<tr>
<td>F10</td>
<td>ADD.D F10,F4,F0</td>
<td>N</td>
</tr>
</tbody>
</table>

Branch misprediction?

FP Op Queue
Reorder Buffer
Registers

FP adds
FP multipliers
Reservation Stations

To Memory
from Memory

FP adders
FP multipliers
Reservation Stations

ROB7, ROB6, ROB5, ROB4, ROB3, ROB2, ROB1

Newest
Oldest

CS485: Vassilis Papaefstathiou

34
Reorder buffer: Branch Misprediction

Reorder Buffer

FP Op Queue

Dest. Value | Instr. Type | Ready
--- | --- | ---
| | | 
-- | BNE F2, <...> | Y
F2 | DIV.D F2, F10, F6 | N
F10 | ADD.D F10, F4, F0 | N

Registers

M[A21]

Branch misprediction
Flush ROB=> Exception gone

FP adders

FP multipliers

Reservation Stations

Commit ptr

Oldest

Newest

FP adders

FP multipliers

Reservation Stations

Commit ptr

Reorder Buffer

Dest

ADD.D R(F4), M[A21]

Dest

DIVD ROB2, R(F6)

Dest

CS415 - Vassilis Papaefstathiou 35
Tomasulo With Reorder buffer

---

**FP Op Queue**

**Reorder Buffer**

---

**Registers**

---

FP adders

FP multipliers

Reservation Stations

---

What about memory hazards???
Memory Disambiguation: WAW/WAR Hazards

• Like Hazards in Register File, we must avoid hazards through memory:
  – WAW and WAR hazards through memory are eliminated with speculation because the actual updating of memory occurs in order, when a store is at the head of the ROB, and hence, no earlier loads or stores can still be pending.
Memory Disambiguation: RAW Hazards

- **Challenge:** Given a load that follows a store in program order, are these two related?
  - What if there is a RAW hazard between the store and the load?  
    
    Eg:  
    
    ```c
    st  0(R2), R5
    ld  R6, 0(R3)
    ```

- Can we proceed and issue the load to the memory system?
  - Store address could be delayed for a long time by some calculation that leads to R2 (e.g. divide).
  - We might want to issue/begin execution of both operations in same cycle.
  - **Solution1:** Answer is that we are not allowed to start load until we know that address `0(R2) \neq 0(R3)`
  - **Solution2:** We might guess at whether or not they are dependent (called “dependence speculation”) and use reorder buffer to fixup if we are wrong.
HW support for Memory Disambiguation

• **Store buffer** keeps all pending stores to memory, in program order
  - Keep track of address (when becomes available) and value (when becomes available)
  - FIFO ordering: will retire stores from this buffer in program order

• When issuing a load, record the head of the store buffer (which stores precede)

• When we have the address of the load, check the buffer:
  - If *any* store prior to load is waiting for its address, stall load
  - If load address matches earlier store address (*associative lookup*), then we have a *memory-induced RAW hazard*:
    - store value available ⇒ return value
    - store value not available ⇒ return ROB number of source
  - Otherwise, send out request to memory

• Stores commit in order, there are no WAW/WAR hazards in memory.
## Memory Disambiguation

### Reorder Buffer

- **FP Op Queue**
- **Registers**
- **Reservation Stations**
- **FP adders**
- **FP multipliers**

### Table: Reorder Buffer

<table>
<thead>
<tr>
<th>ROB</th>
<th>Instruction</th>
<th>Done?</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROB3</td>
<td>L.D F4, 10(R3)</td>
<td>N</td>
</tr>
<tr>
<td>ROB2</td>
<td>S.D 10(R3), F5</td>
<td>N</td>
</tr>
<tr>
<td>ROB1</td>
<td>S.D 0(R3), F4</td>
<td>Y</td>
</tr>
<tr>
<td>ROB4</td>
<td>L.D F5,32(R2)</td>
<td>N</td>
</tr>
<tr>
<td>ROB5</td>
<td>S.D 10(R3), F5</td>
<td>N</td>
</tr>
</tbody>
</table>

### Register Access

- **Dest**
- **FP adders**
- **FP multipliers**

### Memory Access

- **To Memory**
- **From Memory**

### Oldest to Newest

- **ROB7**
- **ROB6**
- **ROB5**
- **ROB4**
- **ROB3**
- **ROB2**
- **ROB1**

### Address in ROB

- **F4**
- **R3**
- **ROB2**
- **ROB2**

### Memory Address

- **32+R2**

---

CS425: Vassilis Papaefstathiou
Register Renaming

• What happens with branches?
• Tomasulo can handle renaming across branches
Explicit register renaming

- Hardware equivalent of static, single-assignment (SSA) compiler form
- Physical register file bigger than ISA register file (e.g. 32 Phys regs και 16 ISA regs)
- Upon issue, every instruction that write a register allocates a new physical register from the freelist
Explicit register renaming

- Note that physical register P0 is “dead” (or not “live”) past the point of this load.
  - When we commit the load, we free up
Explicit register renaming

Current Map Table

Freelist

Done?

Newest

Oldest

Issue ADD F10,F4, F0

F0 F2 F4 F6 F8 F10 F12 F14 F16 F18 F20 F22 F24 F26 F28 F30

P32 P2 P4 P6 P8 P34 P12 P14 P16 P18 P20 P22 P24 P26 P28 P30

F10 P10 ADDD P34,P4,P32 N
F0 P0 LD P32,10(R2) N
Explicit register renaming

Current Map Table

Freelist

Checkpoint at BNE instruction

Done?

Newest

Oldest

Checkpoint at BNE instruction
## Explicit register renaming

### Current Map Table

<table>
<thead>
<tr>
<th>Register</th>
<th>Operation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>ST 0(R3),P40</td>
<td>Y</td>
</tr>
<tr>
<td>P1</td>
<td>ADDD P40,P38,P6</td>
<td>Y</td>
</tr>
<tr>
<td>P2</td>
<td>LD P38,0(R3)</td>
<td>Y</td>
</tr>
<tr>
<td>P3</td>
<td>BNE P36,&lt;…&gt;</td>
<td>N</td>
</tr>
<tr>
<td>P4</td>
<td>DIVD P36,P34,P6</td>
<td>N</td>
</tr>
<tr>
<td>P5</td>
<td>ADDD P34,P4,P32</td>
<td>Y</td>
</tr>
<tr>
<td>P6</td>
<td>LD P32,10(R2)</td>
<td>Y</td>
</tr>
</tbody>
</table>

### Freelists

- **Oldest Freelists:** P42, P44, P48, P50
- **Current Freelists:** P0, P10

### Checkpoint

Checkpoint at BNE instruction

### New Freelists

- **Next Freelists:** P32, P36, P4, P6, P8, P34, P12, P14, P16, P18, P20, P22, P24, P26, P28, P30
- **New Freelists:** P38, P40, P44, P48

### Processors

- **Processors:** P0 to P30
- **Available Processors:** P60, P62

---

CS425 - Vassilis Papafstathiou
Explicit register renaming

Current Map Table

Freelist

Checkpoint at BNE instruction

Speculation error fixed by restoring map table and freelist