Exceptions - Basics

Exception = unprogrammed control transfer
- system takes action to handle the exception
  » must record the address of the "offending" instruction
  » record any other information necessary to return afterwards
- returns control to user
- must save & restore user state

normal control flow:
sequential, jumps, branches, calls, returns

user program

System Exception Handler

return from exception
Two Types of Exceptions

♦ Interrupts
  • caused by external events:
    » Network, Keyboard, Disk I/O, Timer
  • asynchronous to program execution
    » Most interrupts can be disabled for brief periods of time
  • may be handled between instructions
  • simply suspend and resume user program

♦ Traps
  • caused by internal events
    » exceptional conditions (overflow)
    » errors (parity)
    » page faults (non-resident page)
  • synchronous to program execution
  • condition must be remedied by the handler
  • instruction may be retried and program continued or program may be aborted

Exceptions in MIPS pipeline

<table>
<thead>
<tr>
<th>Stage</th>
<th>Possible exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>Page fault on instruction fetch; misaligned memory</td>
</tr>
<tr>
<td></td>
<td>access; memory-protection violation</td>
</tr>
<tr>
<td>ID</td>
<td>Undefined or illegal opcode</td>
</tr>
<tr>
<td>EX</td>
<td>Arithmetic exception</td>
</tr>
<tr>
<td>MEM</td>
<td>Page fault on data fetch; misaligned memory access;</td>
</tr>
<tr>
<td></td>
<td>memory-protection violation; memory error</td>
</tr>
</tbody>
</table>

♦ How do we stop the pipeline? How do we restart it?
♦ Do we interrupt immediately or wait?
♦ 5 instructions, executing in 5 different pipeline stages!
  • Who caused the interrupt?
Multiple exceptions

Time (clock cycles)

<table>
<thead>
<tr>
<th>Cycle 1</th>
<th>Cycle 2</th>
<th>Cycle 3</th>
<th>Cycle 4</th>
<th>Cycle 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="Diagram1.png" alt="Diagram" /></td>
<td><img src="Diagram2.png" alt="Diagram" /></td>
<td><img src="Diagram3.png" alt="Diagram" /></td>
<td><img src="Diagram4.png" alt="Diagram" /></td>
<td><img src="Diagram5.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

- **Data page fault**
- **Arithmetic exception**
- **Instruction page fault**

Precise Interrupts/Exceptions

- Exceptions should be **Precise** or clean, i.e., the outcome should be exactly the same as in a non-pipelined machine.
- Precise ⇒ state of the machine is preserved as if program executed up to the offending instruction:
  - All previous instructions completed
  - Offending instruction and all following instructions act as if they have not even started
  - Same code will work on different processor implementations
  - Difficult in the presence of pipelining, out-of-order execution, ...
- Imprecise ⇒ system software has to figure out what is where and put it all back together
- Modern techniques for out-of-order execution and branch prediction help implement precise interrupts
Relationship between precise interrupts and speculation

♦ Speculation: guess and check
♦ Important for handling branches:
  • Need to “take our best shot” at predicting branch direction
♦ If we speculate and are wrong, need to back up and restart execution to point at which we predicted incorrectly:
  • This is exactly the same as precise exceptions!
♦ Technique for both precise interrupts/exceptions and speculation: *in-order completion or commit*

HW support for precise interrupts

♦ Need HW buffer for results of uncommitted instructions: reorder buffer (ROB)
  • An instruction commits when it completes its execution and all its predecessors have already committed
  • Once instruction commits, result is put into register
  • Therefore, easy to undo speculated instructions on mispredicted branches or exceptions
  • Supplies operands between execution complete & commit
Reorder Buffer

J. Smith & A. Pleszkun, IEEE TC, May 1988 (available on the class web page)
4 Steps of Speculative Tomasulo Algorithm

1. **Issue**— get instruction from FP Op Queue
   - If reservation station, reorder buffer slot, and result shift register slot free, issue instr & send operands & reorder buffer no. for destination. (this stage also called “dispatch”)

2. **Execution**— operate on operands (EX)
   - When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; this takes care of RAW. (sometimes called “issue”)

3. **Write result**— finish execution (WB)
   - Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available

4. **Commit**— update register with result from reorder buffer
   - When instr. at head of reorder buffer & result valid (present), update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch flushes entries in reorder buffer. (this stage sometimes called “graduation”)

**Code Example**

1. LD F0, 10(R2)  
   Finishes 2\text{nd}
2. ADDD F10, F4, F0
3. DIVD F2, F10, F6
4. BNE F2, <...>
5. LD F4, 0(R3)  
   Finishes 1\text{st}
6. ADDD F0, F4, F6
7. ADDD F0, F4, F6
Tomasulo With Reorder buffer - step 1

FP Op Queue

Reorder Buffer

Reserved Stations

FP Adders

FP Multipliers

Registers

FP Values

Instruction

Done?

ROB1

ROB2

ROB3

ROB4

ROB5

ROB6

ROB7

ROB8

ROB9

ROB10

Dest

Newest

Oldest

To Memory

From Memory

Head

Tail

RAT

Newest

Oldest

FP adders

FP multipliers

Destination

Done

ROB1

ROB2

ROB3

ROB4

ROB5

ROB6

ROB7

ROB8

ROB9

ROB10

Tomasulo With Reorder buffer - 2

FP Op Queue

Reorder Buffer

Reserved Stations

FP Adders

FP Multipliers

Registers

FP Values

Instruction

Done?

ROB1

ROB2

ROB3

ROB4

ROB5

ROB6

ROB7

ROB8

ROB9

ROB10

Dest

Newest

Oldest

To Memory

From Memory

Head

Tail

RAT

Newest

Oldest

FP adders

FP multipliers

Destination

Done

ROB1

ROB2

ROB3

ROB4

ROB5

ROB6

ROB7

ROB8

ROB9

ROB10

Page 8
Tomasulo With Reorder buffer - 8

FP Op Queue

Reorder Buffer

RAT Head

Registers

Dest Value Instruction Done?

\begin{tabular}{|c|c|c|c|}
\hline
F0 & <val3> & ADDD F0,F4,F6 & Y \hline
F0 & <val2> & ADDD F0,F4,F6 & Y \hline
F4 M[LD2] & LD F4,0(R3) & Y \hline
-- & BNE F2,<…> & N \hline
F2 & DIVD F2,F10,F6 & N \hline
F10 & ADDD F10,F4,F0 & N \hline
F0 & LD F0,10(R2) & N \hline
\end{tabular}

\begin{align*}
\text{FP adders} & \rightarrow \text{To Memory} \\
\text{Reservation Stations} & \rightarrow \text{From Memory} \\
\text{FP multipliers} & \rightarrow \text{From Memory} \\
\end{align*}

Tomasulo With Reorder buffer - 9 (1\textsuperscript{st} Load)

FP Op Queue

Reorder Buffer

RAT Head

Registers

Dest Value Instruction Done?

\begin{tabular}{|c|c|c|c|}
\hline
F0 & <val3> & ADDD F0,F4,F6 & Y \hline
F0 & <val2> & ADDD F0,F4,F6 & Y \hline
F4 M[LD2] & LD F4,0(R3) & Y \hline
-- & BNE F2,<…> & N \hline
F2 & DIVD F2,F10,F6 & N \hline
F10 & ADDD F10,F4,F0 & N \hline
F0 M[LD1] & LD F0,10(R2) & Y \hline
\end{tabular}

\begin{align*}
\text{FP adders} & \rightarrow \text{To Memory} \\
\text{Reservation Stations} & \rightarrow \text{From Memory} \\
\text{FP multipliers} & \rightarrow \text{From Memory} \\
\end{align*}
FP Op Queue → Reorder Buffer → FP adders

Reorder Buffer:
- F0, F1, F2, F3, F4, F5, F6, F7
- RAT: ROB1, ROB2, ROB3, ROB4, ROB5, ROB6, ROB7

FP adders → FP multipliers

FP multipliers → Reservation Stations

Reservation Stations:
- F0, F1, F2, F3, F4, F5, F6
- ROB1, ROB2, ROB3, ROB4, ROB5, ROB6, ROB7

Registers:
- F0, F1, F2, F3, F4, F5, F6
- Dest Value: a, b, c, d, e

Tomasulo With Reorder buffer

Instructions:
- ADDD F0, F4, F6
- ADDD F0, F4, F6
- LD F4, 0(R3)
- BNE F2, ...
- DIVD F2, F10, F6
- ADDD F10, F4, F0
- LD F0, 10(R2)

Done?
- F0, F1, F2, F3, F4, F5, F6
- ROB1, ROB2, ROB3, ROB4, ROB5, ROB6, ROB7