Predict Not Taken

Branch I+1 I+2 I+3
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB

Branch I+1 I+2 I+3
(bubble) (bubble) (bubble) (bubble)
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB

Branch Target T+1
branch taken
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB

Delayed Branch

Branch I+1 (delay slot) I+2 I+3
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB

Branch I+1 (delay slot) I+2 I+3
branch taken
branch not taken
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB

Branch Target T+1
IF ID EX MEM WB
IF ID EX MEM WB
IF ID EX MEM WB

Filling the delay slot (e.g., in the compiler)

lw R7, 10000(R7)
add R5, R6, R7
beqz R5, label:
sub R8, R1, R3
add R4, R8, R9
and R2, R4, R8

label: lw R2, 1024(R8)

Problems filling delay slot

1. need to predict direction of branch to be most effective
2. limited by correctness restrictions
   correctness restriction can be removed by a canceling branch
   branch likely or branch not likely
   e.g.,
   beqz likely
   delay slot instruction
   fall-through instruction
   squashed/nullified/canceled if branch not taken
Branch Likely

| Branch likely | IF | ID | EX | MEM | WB |
| I+1 (delay slot) | IF | ID | EX | MEM | WB |
| I+2 | IF | ID | EX | MEM | WB |
| I+3 | IF | ID | EX | MEM | WB |

Branch likely

| Branch likely | IF | ID | EX | MEM | WB |
| I+1 (delay slot) | IF | ID | EX | MEM | WB |
| Branch Target | IF | ID | EX | MEM | WB |
| T+1 | IF | ID | EX | MEM | WB |

Branch Performance

CPI = BCPI + pipeline stalls from branches per instruction
= 1.0 + branch frequency * branch penalty
assume 20% branches, 67% taken:

branch scheme taken penalty not taken penalty CPI
stall predict taken predict not taken delayed branch

Delay Slot Utilization

- 18% of delay slots left empty
- 11% of delay slots (1) use canceling branches and (2) end up getting canceled

Delay Slots, the scorecard

- Pros
- Cons
Static Branch Prediction

- Static branch prediction takes place at compile time, dynamic branch prediction during program execution.
- Static branch prediction is done by software, dynamic branch prediction is done in hardware.
- Static branch prediction enables more effective code scheduling around hazards.

Misprediction rate:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>0%</th>
<th>5%</th>
<th>10%</th>
<th>15%</th>
<th>20%</th>
<th>25%</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>eqntott</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>espresso</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gcc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>li</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lodduc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ear</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>hydro2d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mdljdp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>su2cor</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

DLX Integer Pipeline Performance

- Only stalls for load hazards and branch hazards, both of which can be reduced (but not eliminated) by software.

Percentage of all instructions that stall:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>0%</th>
<th>2%</th>
<th>4%</th>
<th>6%</th>
<th>8%</th>
<th>10%</th>
<th>12%</th>
<th>14%</th>
</tr>
</thead>
<tbody>
<tr>
<td>compress</td>
<td>4%</td>
<td>7%</td>
<td>14%</td>
<td>4%</td>
<td>5%</td>
<td>6%</td>
<td>5%</td>
<td>7%</td>
</tr>
<tr>
<td>eqntott</td>
<td>5%</td>
<td>3%</td>
<td>9%</td>
<td>14%</td>
<td>4%</td>
<td>5%</td>
<td>6%</td>
<td>5%</td>
</tr>
<tr>
<td>espresso</td>
<td>9%</td>
<td>1%</td>
<td>6%</td>
<td>9%</td>
<td>5%</td>
<td>6%</td>
<td>5%</td>
<td>7%</td>
</tr>
<tr>
<td>gcc</td>
<td>12%</td>
<td>4%</td>
<td>6%</td>
<td>7%</td>
<td>5%</td>
<td>6%</td>
<td>5%</td>
<td>7%</td>
</tr>
<tr>
<td>li</td>
<td>9%</td>
<td>1%</td>
<td>6%</td>
<td>9%</td>
<td>5%</td>
<td>6%</td>
<td>5%</td>
<td>7%</td>
</tr>
<tr>
<td>lodduc</td>
<td>5%</td>
<td>3%</td>
<td>9%</td>
<td>14%</td>
<td>4%</td>
<td>5%</td>
<td>6%</td>
<td>5%</td>
</tr>
<tr>
<td>ear</td>
<td>11%</td>
<td>5%</td>
<td>6%</td>
<td>7%</td>
<td>5%</td>
<td>6%</td>
<td>5%</td>
<td>7%</td>
</tr>
<tr>
<td>hydro2d</td>
<td>9%</td>
<td>1%</td>
<td>6%</td>
<td>9%</td>
<td>5%</td>
<td>6%</td>
<td>5%</td>
<td>7%</td>
</tr>
<tr>
<td>mdljdp</td>
<td>5%</td>
<td>3%</td>
<td>9%</td>
<td>14%</td>
<td>4%</td>
<td>5%</td>
<td>6%</td>
<td>5%</td>
</tr>
<tr>
<td>su2cor</td>
<td>12%</td>
<td>4%</td>
<td>6%</td>
<td>7%</td>
<td>5%</td>
<td>6%</td>
<td>5%</td>
<td>7%</td>
</tr>
</tbody>
</table>

But now, the real world interrupts...

- Pipelining is not as easy as we have made it seem so far...
  - Interrupts and exceptions
  - Long-latency instructions

Exceptions and Interrupts

- Transfer of control flow (to an exception handler) without an explicit branch or jump.
- Are often unpredictable.
- Examples:
  - I/O device request
  - OS system call
  - Arithmetic overflow/underflow
  - FP error
  - Page fault
  - Memory-protection violation
  - Hardware error
  - Undefined instruction
Classes of Exceptions

- synchronous vs. asynchronous
- user-initiated vs. coerced
- user maskable vs. nonmaskable
- within instruction vs. between instructions
- resume vs. terminate

when the pipeline can be stopped just before the faulting instruction, and can be restarted from there (if necessary), the pipeline supports precise exceptions. Otherwise, it supports imprecise exceptions.

Basic Exception Methodology

- turn off writes for faulting instruction and following
- force a trap into the pipeline at the next IF
- save the PC of the faulting instruction (not quite enough for delayed branches)

Exceptions Can Occur In Several Places in the pipeline

- IF -- page fault on memory access, misaligned memory access, memory-protection violation
- ID -- illegal opcode
- EX -- arithmetic exception
- MEM -- page fault, misaligned access, memory-protection violation
- WB -- none

(and, of course, asynchronous can happen anytime)

Simplifying Exceptions in the ISA

- Each instruction changes machine state only once
  - autoincrement
  - string operations
  - condition codes
- Each instruction changes machine state at the end of the pipeline (when you know it will not cause an exception)
Handling Multicycle Operations

- Unrealistic to expect that all operations take the same amount of time to execute
- FP, some memory operations will take longer
- This violates some of the assumptions of our simple pipeline

Multiple Execution Pipelines

<table>
<thead>
<tr>
<th>FU</th>
<th>Latency</th>
<th>Initiation interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Memory</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>FP add</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>FP multiply</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>FP divide</td>
<td>24</td>
<td>24</td>
</tr>
</tbody>
</table>

New problems

- What structural hazards are now possible?
  - divide unit
  - WB stage
- out-of-order completion
- WAW hazards are possible
- WAR hazards still not possible
- Many more RAW hazards possible

structural hazards and WAW hazards

- structural hazards
  - divide unit
  - WB stage
- WAW hazards
  - ADDD
  - ... IF IF A1 A2 A3 A4 MEM WB
  - ... IF IF ID EX ID ID ID EX MEM EX WB
  - LD
  - ADDD F8, ... IF IF A1 A2 A3 A4 MEM WB
  - LD F8, ... IF IF ID EX MEM WB
Hazard Detection in the ID stage

- An instruction can only issue (proceed past the ID stage) when:
  - there are no structural hazards (divide unit is free, WB port will be free when needed)
  - no RAW data hazards
  - no WAW hazards with instructions in long pipes

Out-of-order completion and precise exceptions

- Two solutions (with slight variations)
  - imprecise interrupts (possibly with software fixes to make it look precise)
  - buffer results, forcing in-order WB’s

MIPS R4000 Pipeline

- scalar, superpipelined
  - IF—first half of fetching of instruction; PC selection happens here as well as initiation of instruction cache access.
  - IS—second half of access to instruction cache.
  - RF—instruction decode and register fetch, hazard checking and also instruction cache hit detection.
  - EX—execution, which includes effective address calculation, ALU operation, and branch target computation and condition evaluation.
  - DF—data fetch, first half of access to data cache.
  - DS—second half of access to data cache.
  - TC—tag check, determine whether the data cache access hit.
  - WB—write back for loads and register-register operations.

R4000 Data Load Hazard

- Is there an integer arithmetic data hazard?
R4000 Data Load Hazard

- lw R1,...   IF IS RF EX DF DS TC WB
- add R3, R1, R2   IF IS RF (stall) (stall) EX DF

- 2-cycle load delay

R4000 Branch Hazard

- predict not taken, branch delay slot
- not taken -> no penalty (unless branch likely or no delay slot instruction)
- taken -> 2 stall cycles if delay slot instruction used

MIPS R4000 Floating Point

- FP Adder, FP Multiplier, FP Divider
- Last step of FP Multiplier/Divider uses FP Adder HW
- 8 kinds of stages in FP units:

<table>
<thead>
<tr>
<th>Stage</th>
<th>Functional unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>FP adder</td>
<td>Mantissa ADD stage</td>
</tr>
<tr>
<td>D</td>
<td>FP divider</td>
<td>Divide pipeline stage</td>
</tr>
<tr>
<td>E</td>
<td>FP multiplier</td>
<td>Exception test stage</td>
</tr>
<tr>
<td>M</td>
<td>FP multiplier</td>
<td>First stage of multiplier</td>
</tr>
<tr>
<td>N</td>
<td>FP multiplier</td>
<td>Second stage of multiplier</td>
</tr>
<tr>
<td>R</td>
<td>FP adder</td>
<td>Rounding stage</td>
</tr>
<tr>
<td>S</td>
<td>FP adder</td>
<td>Operand shift stage</td>
</tr>
<tr>
<td>U</td>
<td></td>
<td>Unpack FP numbers</td>
</tr>
</tbody>
</table>

MIPS FP Pipe Stages

<table>
<thead>
<tr>
<th>FP Instr</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add, Subtract</td>
<td>U</td>
<td>S+A</td>
<td>A+R</td>
<td>R+S</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multiply</td>
<td>U</td>
<td>E+M</td>
<td>M</td>
<td>M</td>
<td>M</td>
<td>N</td>
<td>N+A</td>
<td>R</td>
</tr>
<tr>
<td>Divide</td>
<td>U</td>
<td>A</td>
<td>R</td>
<td>D²⁸</td>
<td>...</td>
<td>D+R</td>
<td>D+R</td>
<td>D+A</td>
</tr>
<tr>
<td>Square root</td>
<td>U</td>
<td>E</td>
<td>(A+R)¹⁰⁸</td>
<td>...</td>
<td>A</td>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Negate</td>
<td>U</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Absolute value</td>
<td>U</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP compare</td>
<td>U</td>
<td>A</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stages:</th>
<th>A</th>
<th>M</th>
<th>N</th>
<th>R</th>
<th>S</th>
<th>U</th>
</tr>
</thead>
<tbody>
<tr>
<td>First stage of multiplier</td>
<td>A</td>
<td>Mantissa ADD stage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Second stage of multiplier</td>
<td>D</td>
<td>Divide pipeline stage</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rounding stage</td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operand shift stage</td>
<td>S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unpack FP numbers</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
R4000 Performance

Key Points

- Data Hazards can be significantly reduced by forwarding
- Branch hazards can be reduced by early computation of condition and target, branch delay slots, branch prediction
- Data hazard and branch hazard reduction require complex compiler support
- Exceptions are hard, precise exceptions are really hard
- variable-length instructions introduce structural hazards, WAW hazards, more RAW hazards