18-447 Lecture 9:
Control Hazard and Resolution

James C. Hoe
Department of ECE
Carnegie Mellon University
Housekeeping

• Your goal today
  – “simple” control flow resolution in in-order pipelines
  – there is more fun to come on this

• Notices
  – Lab 2, status check next week, due wk of 2/25
  – HW 2, due 2/20, **before class**
  – Midterm 2/25 in class; covers Lectures 1~9

• Readings
  – P&H Ch 4
Control Dependence

- C-Code

{ code A }
if X==Y then
{ code B }
else
{ code C }
{ code D }

At ISA-level, control dependence == “data dependence on PC”
### Applying Hazard Analysis on PC

<table>
<thead>
<tr>
<th></th>
<th>R/I-Type</th>
<th>LW</th>
<th>SW</th>
<th>Bxx</th>
<th>Jal</th>
<th>Jalr</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF</strong></td>
<td>use</td>
<td>use</td>
<td>use</td>
<td>use</td>
<td>use</td>
<td>use</td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>produce</td>
<td>produce</td>
<td>produce</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EX</strong></td>
<td></td>
<td>produce</td>
<td>produce</td>
<td>produce</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MEM</strong></td>
<td></td>
<td></td>
<td></td>
<td>produce</td>
<td>produce</td>
<td>produce</td>
</tr>
<tr>
<td><strong>WB</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- All instructions read and write PC
- PC dependence distance is exactly 1
- PC hazard distance in 5-stage is at least 1
  - ⇒ Yes, there is RAW hazard
  - ⇒ forwarding is no help; but stall always works
Resolve Control Hazard by Stalling

Keep in mind, this is still if decoding to non-control-flow
Only 1 way to beat “true” dependence
Resolve Control Hazard by Guessing

What is your best guess?
What is known at this point?

PC + 4
Control Speculation for Dummies

• Guess nextPC = PC+4 to keep fetching every cycle
  Is this a good guess?

• ~20% of the instruction mix is control flow
  – ~50% of “forward” control flow taken (if-then-else)
  – ~90% of “backward” control flow taken (end-of-loop)
    Over all, typically ~70% taken and ~30% not taken
    [Lee and Smith, 1984]

• Expect “nextPC = PC+4” ~86% of the time, but what about the remaining 14%?
  What do you do when wrong?
  What do you lose when wrong?
Control Speculation: PC+4

Inst\(_h\) is a taken branch

- branch target (Inst\(_k\)) is fetched
- flush instructions fetched since Inst\(_h\) (“wrong-path”)

When Inst\(_h\) branch resolves

control flow “restitched”
Pipeline Flush on Misprediction

Inst_h is a taken branch; Inst_i and Inst_l fetched but not executed
### Pipeline Flush on Misprediction

<table>
<thead>
<tr>
<th></th>
<th>$t_0$</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
<th>$t_6$</th>
<th>$t_7$</th>
<th>$t_8$</th>
<th>$t_9$</th>
<th>$t_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IF</strong></td>
<td>h</td>
<td>i</td>
<td>j</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td>h</td>
<td>i</td>
<td>bub</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EX</strong></td>
<td>h</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MEM</strong></td>
<td>h</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WB</strong></td>
<td>h</td>
<td>bub</td>
<td>bub</td>
<td>bub</td>
<td>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Branch resolved
Performance Impact

• Correct guess ⇒ no penalty  most of the time!!
• Incorrect guess ⇒ 2 bubbles
• Assume
  – no data hazard stalls
  – 20% control flow instructions
  – 70% of control flow instructions are taken
  IPC = \(\frac{1}{1 + (0.20 \times 0.7) \times 2}\) =
    \(= \frac{1}{1 + 0.14 \times 2} = \frac{1}{1.28} = 0.78\)

misprediction rate  
misprediction penalty

How to reduce the two penalty terms?
Reducing Mispredict Penalty

[Diagram showing the flow of instructions through the pipeline stages: IF, ID, EX, MEM, WB, with control signals and data paths.]
MIPS R2000 ISA Control Flow Design

• Simple address calculation based on IR only
  – branch PC-offset: 16-bit full-addition
  + 14-bit half-addition
  – jump PC-offset: concatenation only
• Simple branch condition based on RF
  – one register relative (>, <, =) to 0
  – equality between 2 registers
    No addition/subtraction necessary!

Explicit ISA design choices to make possible branch resolution in ID of a 5-stage pipeline
Branch Resolved in ID

IPC = 1 / [ 1 + (0.2*0.7) * 1 ] = 0.88

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Branch Delay Slots

• Throwing PC+4 away cost 1 bubble; letting PC+4 finish won’t hurt performance . . . . .

• R2000 jump/branch has 1 inst. architectural latency
  – PC+4 after jump/branch always executed
  no need for pipeline flush logic
  – if delay slot always do useful work, effective IPC=1
  – ~80% of “delay slots” can be filled by compilers

\[
IPC = \frac{1}{1 + (0.2 \times 0.2) \times 1} = 0.96
\]
MIPS R2000 Interlock Free Pipeline

<table>
<thead>
<tr>
<th></th>
<th>R/I-Type</th>
<th>LW</th>
<th>SW</th>
<th>Bxx</th>
<th>Jal</th>
<th>Jalr</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td>use</td>
<td>use</td>
<td>use</td>
<td>use</td>
<td>use</td>
<td>use</td>
</tr>
<tr>
<td>ID</td>
<td>produce</td>
<td>produce</td>
<td>produce</td>
<td>produce</td>
<td>produce</td>
<td>produce</td>
</tr>
<tr>
<td>EX</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Simple branch $\Rightarrow$ PC hazard distance is always 1
- Delayed branch $\Rightarrow$ PC dependence distance is always 2
  
  (ALU instructions really says nextnextPC = nextPC+4)

**MIPS** = Microproc. without Interlocked Pipeline Stages
## Wait just a second . . . .

<table>
<thead>
<tr>
<th>R/I-Type</th>
<th>LW</th>
<th>SW</th>
<th>Bxx</th>
<th>J</th>
<th>Jr</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td></td>
<td>use</td>
<td>use</td>
<td>use</td>
<td>use</td>
</tr>
<tr>
<td>EX</td>
<td>use</td>
<td>use</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td></td>
<td>produce</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Last lecture, all instruction **used** RF values in EX
  - no RAW hazard on everything but LW if forwarding
  - no RAW hazard if MIPS “delayed” LW
- But delayed branch “trick” needs RF values in ID . . .
Forwarding Paths (v1)

to be latched by PC

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Forwarding Paths (v2)

Combinationally to inst. mem.

[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]
Making a Better Guess
(for when it is not MIPS or 5-stage)

• For non-control-flow instructions
  – can’t do better than guessing $\text{nextPC}=\text{PC}+4$
  – still tricky since must guess before knowing it is control-flow or non-control-flow

• For control-flow instructions
  – why not always guess taken since 70% correct
  – need to know taken target to be helpful

• Guess $\text{nextPC}$ from current PC alone, and fast!

• Fortunately
  – instruction at same PC doesn’t change
  – PC-offset target doesn’t change
  – okay to be wrong some of the time
In case you needed motivation

![Basic Pentium III Processor Misprediction Pipeline](chart1)

![Basic Pentium 4 Processor Misprediction Pipeline](chart2)

[The Microarchitecture of the Pentium 4 Processor, Intel Technology Journal, 2001]
Branch Target Buffer (magic version)

- **BTB**
  - a giant table indexed by PC
  - returns the “guess” for nextPC
- When seeing a PC first time, after decoding, record in BTB . . .
  - PC + 4 if ALU/LD/ST
  - PC+offset if Branch or Jump
  - ?? if JR
- Effectively guessing branches are always taken (and where to)

\[
IPC = \frac{1}{1 + (0.20 \times 0.3) \times 2}
\]

\[
= 0.89
\]
Locality Principle to the Rescue

• Temporal Locality: If you just did something, very likely you will do the same again soon
  – since you are here today, there is a good chance you will be here again and again regularly
  – inverse is also true

• Spatial Locality: If you just did something, very likely you will do something similar/related
  – you are probably sitting near the same people

• Programs even predictable than people
  \(\Rightarrow\) **BTB does not need to track every PC value, just a small footprint of active ones!**
Locality says just do this

- “Hash” PC into a $2^N$ entry table
- What happens when two branches hash to the same entry?
Tagged BTB

Only store branch instructions (save 80% storage)
Update tag and BTB for new branch after collision
Final 5-stage RISC Datapath & Control