18-447 Lecture 9: Control Hazard and Resolution

James C. Hoe
Department of ECE
Carnegie Mellon University
2-Cycle Synchronous Mem Read/Write

CLK

ADDR

XXX addr0 addr1 addr2

RDATA

XXX data0 data1 data2 XXX new

WDATA

XXX new XXX

RE

this read sees

WE

this write
2024 Lab 2 with 7 stages

200ps
IF: Instruction fetch

100ps
ID: Instruction decode/register file read

200ps
EX: Execute/address calculation

200ps
MEM: Memory access

100ps
WB: Write back

ignore for today

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

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

• Notices
  – HW 2, due Mon 2/19
  – Lab 2, status check wk6, due wk7 (Handout #7)

  Look (with your brain!!!) at the counter values

• 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
  \[\implies\] Yes, there is RAW hazard
  \[\implies\] Can’t eliminate by forwarding; so must stall
Resolve Control Hazard by Stalling

Note: this is if decoding to non-control-flow; BR resolves in EX
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

When $\text{inst}_h$ branch resolves
- branch target ($\text{Inst}_k$) is fetched
- flush instructions fetched since $\text{inst}_h$ (“wrong-path”)
Pipeline Flush on Misprediction

Inst_h is a taken branch; Inst_i and Inst_j 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><strong>h</strong></td>
<td><strong>i</strong></td>
<td><strong>j</strong></td>
<td><strong>k</strong></td>
<td><strong>l</strong></td>
<td><strong>m</strong></td>
<td><strong>n</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ID</strong></td>
<td><strong>h</strong></td>
<td><strong>i</strong></td>
<td><strong>bub</strong></td>
<td><strong>k</strong></td>
<td><strong>l</strong></td>
<td><strong>m</strong></td>
<td><strong>n</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EX</strong></td>
<td><strong>h</strong></td>
<td><strong>bub</strong></td>
<td><strong>bub</strong></td>
<td><strong>k</strong></td>
<td><strong>l</strong></td>
<td><strong>m</strong></td>
<td><strong>n</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MEM</strong></td>
<td><strong>h</strong></td>
<td><strong>bub</strong></td>
<td><strong>bub</strong></td>
<td><strong>bub</strong></td>
<td><strong>k</strong></td>
<td><strong>l</strong></td>
<td><strong>m</strong></td>
<td><strong>n</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WB</strong></td>
<td><strong>h</strong></td>
<td><strong>bub</strong></td>
<td><strong>bub</strong></td>
<td><strong>bub</strong></td>
<td><strong>k</strong></td>
<td><strong>l</strong></td>
<td><strong>m</strong></td>
<td><strong>n</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Branch resolved
Performance Impact

• Correct guess \( \Rightarrow \) no penalty \ most of the time!!
• Incorrect guess \( \Rightarrow \) 2 bubbles
• Assume
  – no data hazard stalls
  – 20% control flow instructions
  – 70% of control flow instructions are taken

\[
\text{IPC} = \frac{1}{1 + (0.20 \times 0.7) \times 2} = \\
= \frac{1}{1 + 0.14 \times 2} = \frac{1}{1.28} = 0.78
\]

How to reduce the two penalty terms?
Reducing Mispredict Penalty

Why not resolve in ID so penalty=1?

P&H figure resolves in MEM, penalty=3

Why not resolve in ID so penalty=1?
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 = \frac{1}{1 + (0.2 \times 0.7) \times 1} = 0.88

what about this?
Forwarding (v1): extend critical path

where to fetch next clock period (mux before PC reg)
Forwarding (v2): retiming hack

where to fetch this clock period (mux after PC reg)

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

• 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

\[
\text{IPC} = \frac{1}{[1 + (0.2 \times 0.2) \times 1]} = 0.96
\]
Also MIPS Load “Delay Slot”

\[ \begin{align*}
I_1 : & \text{ LW } \ x1 \ --- \\
I_2 : & \text{ addi } \ r2, \ x1, 0 \\
I_3 : & \text{ addi } \ r3, \ x1, 0
\end{align*} \]

- R2000 defined LW with arch. latency of 1 inst
  - invalid for \( I_2 \) (in LW’s delay slot) to ask for LW’s result
  - any dependence on LW at least distance 2

- Delay slot vs dynamic stalling
  - fill with an independent instruction (no difference)
  - if not, fill with a NOP (no difference)

- **MIPS**=Microproc. without Interlocked Pipeline Stages

*Delay slots good idea? non-atomic, \( \mu \)arch specific*
Performance Impact

• Correct guess ⇒ no penalty most of the time!!
• Incorrect guess ⇒ 2 bubbles; 1 if resolve in ID
• 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

Need to do more? how about?

\( 0.87 \) resolve in ID; \( 0.96 \) MIPS delay slot
In case you needed motivation

**Basic Pentium III Processor Misprediction Pipeline**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Fetch</td>
<td>Decode</td>
<td>Decode</td>
<td>Decode</td>
<td>Rename</td>
<td>ROB Rd</td>
<td>Rdy/Sch</td>
<td>Dispatch</td>
<td>Exec</td>
</tr>
</tbody>
</table>

**Basic Pentium 4 Processor Misprediction Pipeline**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
<th>17</th>
<th>18</th>
<th>19</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC</td>
<td>Nxt IP</td>
<td>TC Fetch</td>
<td>Fetch</td>
<td>Drive</td>
<td>Alloc</td>
<td>Rename</td>
<td>Que</td>
<td>Sch</td>
<td>Sch</td>
<td>Disp</td>
<td>Disp</td>
<td>RF</td>
<td>RF</td>
<td>Ex</td>
<td>Flgs</td>
<td>Br Ck</td>
<td>Drive</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

[The Microarchitecture of the Pentium 4 Processor, Intel Technology Journal, 2001]
Can we make better guesses? (for when it is not MIPS or 5-stage)

• For control-flow instructions
  – why not always guess taken since 70% correct
  – need to know taken target to be helpful
• For non-control-flow instructions
  – can’t do better than guessing nextPC=PC+4
  – still tricky since must guess before knowing it is control-flow or non-control-flow
• Guess nextPC from current PC alone, and fast!
• Fortunately
  – instruction at same PC doesn’t change
  – PC-offset target doesn’t changes
  – okay to be wrong some of the time
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 Jump Indirect

- Effectively guessing branches are always taken (and where to)
  \[
  \text{IPC} = \frac{1}{1 + (0.20 \times 0.3) \times 2}
  \]
  \[
  = 0.89
  \]

If not taken
Locality Principle to the Rescue

• **Temporal:** after accessing A, how many other distinct addresses before accessing A again?

• **Spatial:** after accessing A, how many other distinct addresses before accessing B?

• “Typical” programs have strong locality in memory references—instruction and data we put them there ... BB, loops, arrays, structs ...

• **Corollary:** a program with strong temporal and spatial locality access only a compact “working set” at any point in time

⇒ just need BTB big enough for *hot* instructions
Smaller BTB by Hashing

- "Hash" PC into a $2^N$ entry table
- What happens when two "hot" instructions collide? *No problem, as long as infrequent*
Even Smaller BTB after Tagging

Add tag to tell control-flow from non-control flow

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