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
  – HW 2, due Mon 2/21
  – Lab 2, status check wk6, due wk7 (Handout #7)
  – HW 3, due Mon 2/28 (Handout #8)
  – Midterm 1, Wed 3/2 (look for practice midterm 1)

• Readings
  – P&H Ch 4
Format of the Midterm

• Covers lectures (L1~L10), HW, labs, assigned readings (from textbooks and papers)

• Types of questions
  – freebies: remember the materials
  – >> probing: understand the materials <<
  – applied: apply the materials in original interpretation

• **70 minutes, 70 points**
  – point values calibrated to time needed
  – closed-book, one 8½x11-in² hand-written cribsheet
  – no electronics
  – use pencil or black/blue ink only
Control Dependence

- **C-Code**

\[
\{ \text{code A} \} \\
\text{if } X == Y \text{ then} \\
\{ \text{code B} \} \\
\text{else} \\
\{ \text{code C} \} \\
\{ \text{code D} \}
\]

**Control Flow Graph**

- **True**
  - code B
  - code D

- **False**
  - code C
  - code D

**Assembly Code** (linearized)

- code A
- if $X == Y$
- goto code C
- goto code B
- code D
- 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>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></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td>produce</td>
<td>produce</td>
<td>produce</td>
<td>produce</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>

- 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
  ⇒ 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

future
Resolve Control Hazard by Guessing

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

Inst_h
Inst_i
Inst_j
Inst_k
Inst_l

IF

???

PC+4
Control Speculation for Dummies

• Guess \( \text{nextPC} = \text{PC}+4 \) to keep fetching every cycle
  Is this a good guess?

• \( \sim20\% \) of the instruction mix is control flow
  – \( \sim50\% \) of “forward” control flow taken (if-then-else)
  – \( \sim90\% \) of “backward” control flow taken (end-of-loop)

  Over all, typically \( \sim70\% \) taken and \( \sim30\% \) not taken
  [Lee and Smith, 1984]

• Expect “nextPC = PC+4” \( \sim86\% \) 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>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>k</td>
<td>l</td>
<td>m</td>
<td>n</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MEM</strong></td>
<td>h</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>
<td></td>
</tr>
<tr>
<td><strong>WB</strong></td>
<td>h</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>
<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 = 1 / [ 1 + (0.20*0.7) * 2 ] =
    = 1 / [ 1 + 0.14 * 2 ] = 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
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
IPC = $\frac{1}{1 + (0.2 \times 0.7) \times 1} = 0.88$
Forwarding (v1): extend critical path

to nPC mux before the PC reg

Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.
Forwarding (v2): retiming hack

to PC mux before inst. mem.

Registers

ID/EX

EX/MEM

MEM/WB

Mux

Forwarding unit

ForwardB

ForwardA

br cond?

ALU

Data memory

EX/MEM.RegisterRd

MEM/WB.RegisterRd

rs1Ex

rs2Ex

rdEx

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

18-447-S22-L09-S18, James C. Hoe, CMU/ECE/CALCM, ©2022

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

Bxx r- L1

PC+4

L1 if taken else PC+8

• 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”

1. R2000 defined LW with arch. latency of 1 inst
   - invalid for l₂ (in LW’s delay slot) to ask for LW’s result
   - any dependence on LW at least distance 2
2. Delay slot vs dynamic stalling
   - fill with an independent instruction (no difference)
   - if not, fill with a NOP (no difference)
3. MIPS = Microproc. without Interlocked Pipeline Stages

Delay slots good idea? non-atomic, μarch specific
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
- \( \text{IPC} = \frac{1}{1 + (0.20 \times 0.7) \times 2} = \frac{1}{1 + 0.14 \times 2} = \frac{1}{1.28} = 0.78 \)

How to reduce the two penalty terms?

misprediction rate
misprediction penalty

How about?
In case you needed motivation

<table>
<thead>
<tr>
<th>Basic Pentium III Processor Misprediction Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Fetch</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Basic Pentium 4 Processor Misprediction Pipeline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 TC Nxt IP</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 change
  – 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)
  \[
  \text{IPC} = 1 / \left[ 1 + (0.20 \times 0.3) \times 2 \right] \\
  = 0.89
  \]

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

Based on Figure 4.53, P&H CO&D-RV, COPYRIGHT 2020 Elsevier Inc. ALL RIGHTS RESERVED.