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
  – HW2, due Mon, March 8
  – Midterm 1, Wed, March 10 (try rehearsal on Canvas)
  – Lab 2, status check due Fri, March 12
  – Need to balance work between this week and next

• Readings
  – P&H Ch 4
Format of Midterm

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

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

• **55 minutes, 55 points**
  – 11 short-answer, typed-response questions
  – start of class on 3/10, online through Canvas
  – communicate with me privately by Zoom chat
  – openbook, individual effort
What to Expect

• 11 “5-point” short answer questions
  – ordered “easier” to “harder”
  – 1 question at a time and cannot go back
  – only first 45 words of each response graded

• Recommended strategy
  – give each question about 5min—as if taking 11 separate 5-min quizzes

• Be prepared
  – rehearse mock midterm on Canvas
  – have your space and equipment ready
  – have a clock on your desk
Control Dependence

- C-Code

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

Control Flow Graph

True

False

code A
if X==Y
code B
code C
code D

Assembly Code (linearized)

code A
code C
goto
code B
code D

code A
if X==Y

goto
code B
code D

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

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>ID</td>
<td>produce</td>
<td>produce</td>
<td>produce</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX</td>
<td></td>
<td>produce</td>
<td>produce</td>
<td>produce</td>
<td></td>
</tr>
<tr>
<td>MEM</td>
<td></td>
<td></td>
<td>produce</td>
<td>produce</td>
<td>produce</td>
</tr>
<tr>
<td>WB</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

18-447-S21-L09-S6, James C. Hoe, CMU/ECE/CALCM, ©2021
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

Inst_{h} \rightarrow t_0 \rightarrow IF \rightarrow ID \rightarrow ALU \rightarrow MEM \rightarrow WB

Inst_{i} \rightarrow t_1 \rightarrow IF \rightarrow ID \rightarrow ALU \rightarrow MEM \rightarrow WB \rightarrow WB

Inst_{j} \rightarrow t_2 \rightarrow IF \rightarrow ID \rightarrow ALU \rightarrow MEM \rightarrow WB

Inst_{k} \rightarrow t_3 \rightarrow IF \rightarrow ID \rightarrow ALU \rightarrow MEM \rightarrow WB

Inst_{h} \rightarrow t_4 \rightarrow IF \rightarrow ID \rightarrow ALU \rightarrow MEM \rightarrow WB

Inst_{h} \rightarrow t_5 \rightarrow IF \rightarrow ID \rightarrow ALU \rightarrow MEM

future
Resolve Control Hazard by Guessing

What is your best guess?
What is known at this point?
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 inst_h branch resolves
- branch target (Inst_k) is fetched
- flush instructions fetched since 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>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>t₆</th>
<th>t₇</th>
<th>t₈</th>
<th>t₉</th>
<th>t₁₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>IF</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>ID</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>EX</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>MEM</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>WB</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>

The pipeline stages are as follows:
- **IF** (Instruction Fetch)
- **ID** (Instruction Decode)
- **EX** (Execute)
- **MEM** (Memory Access)
- **WB** (Write Back)

The flush occurs when a branch is mispredicted, indicated by the red arrows pointing from **ID** to **EX** and **MEM**. The flush is resolved after the branch prediction is re-evaluated, as shown by the red arrow from **MEM** to **WB**.

Branch resolved
Performance Impact

• Correct guess $\implies$ no penalty  
• Incorrect guess $\implies$ 2 bubbles

Assume

– no data hazard stalls
– 20% control flow instructions
– 70% of control flow instructions are taken

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

How to reduce the two penalty terms?
Reducing Mispredict Penalty

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

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

what about this?
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.
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”

1. LW x1 --- IF ID EX MEM WB
2. addi r2, x1, 0 IF ID EX MEM WB
3. addi r3, x1, 0 IF ID EX MEM WB

- R2000 defined LW with arch. latency of 1 inst
  - invalid for I2 (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, μarch specific
Can we make better guesses? (for when it is not MIPS or 5-stage)

- 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
- For control-flow instructions
  - why not always guess taken since 70% correct
  - need to know taken target to be helpful
- 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
In case you needed motivation

[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)
  
  $$IPC = \frac{1}{1 + (0.20 \times 0.3) \times 2}$$
  
  $= 0.89$

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 branches hash to the same entry?
Even Smaller BTB after Tagging

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

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