## Branch Hazards

Consider executing this sequence of instructions in the pipeline:

<table>
<thead>
<tr>
<th>address</th>
<th>instruction</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>36:</td>
<td>sub $10, $4, $8</td>
<td></td>
</tr>
<tr>
<td>40:</td>
<td>beq $1, $3, 72</td>
<td></td>
</tr>
<tr>
<td>44:</td>
<td>and $12, $2, $5</td>
<td></td>
</tr>
<tr>
<td>48:</td>
<td>or $13, $2, $6</td>
<td></td>
</tr>
<tr>
<td>52:</td>
<td>add $14, $4, $2</td>
<td></td>
</tr>
<tr>
<td>56:</td>
<td>slt $15, $6, $7</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>72:</td>
<td>lw $4, 50($7)</td>
<td></td>
</tr>
</tbody>
</table>

**Issue:**

Should we fetch the instruction at address 44 when `beq` moves into the ID stage?

- It may or may not be executed next, depending on whether $1 == $3
- We don’t know the address of the branch target instruction yet anyway
Branch Hazards

When $beq$ moves into the ID stage, we don’t even know it is a conditional branch.

And… we won’t know if the branch should be taken until $beq$ reaches the end of EX.

Hey! It’s a branch!

<table>
<thead>
<tr>
<th>address</th>
<th>instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>36:</td>
<td>$sub$ $10, 4, 8$</td>
</tr>
<tr>
<td>40:</td>
<td>$beq$ $1, 3, 7$</td>
</tr>
<tr>
<td>44:</td>
<td>$and$ $12, 2, 5$</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>72:</td>
<td>$lw$ $4, 50(7)$</td>
</tr>
</tbody>
</table>
So… we will have already fetched the next (sequential) instruction.

```
36:    sub  $10, $4, $8
40:    beq  $1, $3,  7
44:    and  $12, $2, $5
...  
72:    lw   $4, 50($7)
```
Stalling for Branch Hazards

Idea: insert stalls until we know if the branch will be taken.

We can’t act on that information before `beq` reaches the MEM stage.
Stalling for Branch Hazards

Idea: insert stalls until we know if the branch will be taken.

That’s expensive.

If we don’t take the branch, we needlessly delayed the and instruction for 3 cycles.

<table>
<thead>
<tr>
<th>cycle</th>
<th>action</th>
<th>beq</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>fetch sub</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>fetch beq</td>
<td>IF</td>
</tr>
<tr>
<td>2</td>
<td>fetch and</td>
<td>ID</td>
</tr>
<tr>
<td>3</td>
<td>stall</td>
<td>EX</td>
</tr>
<tr>
<td>4</td>
<td>stall</td>
<td>MEM</td>
</tr>
<tr>
<td>5</td>
<td>fetch and/lw</td>
<td></td>
</tr>
</tbody>
</table>

Diagram:

```
fetch -> stall -> stall -> beq -> sub
```
Rollback for Branch Hazards

Idea: proceed as if the branch will not be taken; turn mis-fetched instructions into nops if wrong.

Time (in clock cycles)

<table>
<thead>
<tr>
<th>CC 1</th>
<th>CC 2</th>
<th>CC 3</th>
<th>CC 4</th>
<th>CC 5</th>
<th>CC 6</th>
<th>CC 7</th>
<th>CC 8</th>
<th>CC 9</th>
</tr>
</thead>
</table>

Program execution order (in instructions)

- 40 beq $1, $3, 28
- 44 and $12, $2, $5
- 48 or $13, $6, $2
- 52 add $14, $2, $2
- 72 lw $4, 50($7)

If branch is taken, flush* these instructions (set control values to 0)

*QTP: will this be too late?
Questions

Could we rearrange the datapath so that the branch decision could be made earlier?

Questions to ponder:

- What about calculating the branch target address?
  Can that be done in the ID stage?

- What about the register comparison?
  Can that be done in the ID stage?
  What about other kinds of conditional branches (e.g., bgez,)?
Ideas: simple hardware suffices to compare two registers moving the branch adder is relatively simple

We can determine if the branch will be taken before `beq` reaches the EX stage.
Making Branch Decisions Earlier

Stall if branch is taken, proceed normally otherwise:

Cost is now one stall if branch is taken, nothing if branch is not taken.
New Control Features

Handling Branches 10

Clock 4

IF:ID

Instruction memory

Mux 76 PC 72

Lw $4, 50($7)

Bubble (nop)

Hazard detection unit

Control

Sign extend

EX

Mux

= 0

ID/EX

M

$1

EX/MEM

Mux

$3

ALU

Data memory

Forwarding unit

MEM/WB

Mux

sub $10, ... before<1>

beq $1, $3, 7

©2005-2015 McQuain
Data Hazards for Branches

If a comparison register in \texttt{beq} is a destination of 2\textsuperscript{nd} or 3\textsuperscript{rd} preceding ALU instruction

Can resolve using forwarding

\begin{align*}
\text{add } &\$1, \$2, \$3 \\
\text{add } &\$4, \$5, \$6 \\
\text{...} \\
\text{beq } &\$1, \$4, \text{target}
\end{align*}

\textbf{QTP:} why is the forwarding-to issue different (now) for \texttt{beq} than for the other instructions?
Data Hazards for Branches

If a comparison register is a destination of preceding ALU instruction or 2\textsuperscript{nd} preceding load instruction

- Need to stall for 1 cycle

\begin{verbatim}
lw    $1, addr
add  $4, $5, $6
beq  stalled
beq  $1, $4, target
\end{verbatim}
Data Hazards for Branches

If a comparison register is a destination of immediately preceding load instruction
  - Need to stall for 2 cycles

\[
\text{lw} \quad \text{$1$, addr} \quad \text{IF} \quad \text{ID} \quad \text{EX} \quad \text{MEM} \quad \text{WB} \\
\text{beq} \quad \text{stalled} \\
\text{beq} \quad \text{$1$, $0$, target} \quad \text{IF} \quad \text{ID} \quad \text{EX} \quad \text{MEM} \quad \text{WB}
\]
Dynamic Branch Prediction

In deeper and superscalar pipelines, the branch penalty is more significant

Use dynamic prediction

- need a branch history table (aka branch prediction buffer)
- indexed by addresses of recent branch instructions
- stores recent outcome(s) for branch (taken/not taken)

To execute a branch:

- check the table, expect consistent behavior with recent past
- start fetching (fall-through instruction or branch target)
- if wrong, flush pipeline (stall) and flip prediction
- update table accordingly
1-Bit Predictor

Shortcoming: inner loop branches are mispredicted twice!

Idea: use one bit to remember if the branch was taken or not the last time.

outer: ...
   ...
inner: ...
   beq ..., ..., inner
   ...
   beq ..., ..., outer

Mispredict as taken on last iteration of inner loop

Then mispredict as not taken on first iteration of inner loop next time around
2-Bit Predictor

Idea: use two bits to remember if the branch was taken or not the last time.

Only change prediction on two successive mispredictions.
Calculating the Branch Target

Even with predictor, still need to calculate the target address
- 1-cycle penalty if branch is taken

Branch target buffer
- Add a cache of target addresses
- Indexed by PC when instruction is fetched
  - If hit (i.e., target address is in cache) and
    instruction is branch predicted taken, can fetch target immediately