Branch Hazards

or

“Which way did he go?”

Control Dependence

• Just as an instruction will be dependent on other instructions to provide its operands ( ), it will also be dependent on other instructions to determine whether it gets executed or not ( or )

• Control dependences are particularly critical with branches.

Branch Hazards

• Branch dependences can result in (when they are too close to be handled correctly in the pipeline).
Dealing With Branch Hazards

- Hardware
  - until you know which direction
  - reduce hazard through of branch direction
  - which direction
    - assume not taken (easiest)
    - more educated guess based on history (requires that you know it is a branch before it is even decoded!)
- Hardware/Software
  - nops, or instructions that get executed either way (delayed branch).

Stalling for Branch Hazards

- Seems wasteful, particularly when the branch isn’t taken.
- Makes all branches cost cycles.

Assume Branch Not Taken

- works pretty well when you’re right
Assume Branch *Not Taken*

- same performance as stalling when you’re wrong

```
beq $4, $0, there
and $12, $2, $5
or ...
add ...
```

there: `sub $12, $4, $2`

```
• same performance as stalling when you’re wrong

- depends on percentage of time you guess right.

- Flushing an instruction means to prevent it from changing any (registers, memory, PC).
  - sounds a lot like a ...
  - But notice that we need to be able to insert those bubbles later in the pipeline

Reducing the Branch Delay

```
• can easily get to 2-cycle stall
```

Stalling for Branch Hazards

```
beq $4, $0, there
and $12, $2, $5
or ...
add ...
sw ...
```

```
Reducing the Branch Delay

• Harder, but possible, to get to 1-cycle stall

Stalling for Branch Hazards

The Pipeline with flushing for taken branches

Eliminating the Branch Stall

• There’s no rule that says we have to see the effect of the branch ... Why not wait an instruction before branching?
• The original SPARC and MIPS processors each used a single branch delay slot to eliminate single-cycle stalls after branches.
• The instruction after a conditional branch is always executed in those machines, regardless of whether the branch is taken or not!
Branch Delay Slot

Branch delay slot instruction (next instruction after a branch) is executed even if the branch is taken.

Filling the branch delay slot

• The branch delay slot is only useful if you can find something to put there.
• If you can’t find anything, you must put a *nop* to insure correctness.

- From before
  - add $s1, $s2, $s3
  - if $s2 = 0 then
    - Delay slot

- From target
  - add $s1, $s2, $s3
  - if $s1 = 0 then
    - Delay slot

- From fall through
  - add $s1, $s2, $s3
  - if $s1 = 0 then
    - Delay slot

Branch Delay Slots

• This works great for this implementation of the architecture, but becomes a permanent part of the ISA.
• What about the MIPS R10000, which has a 5-cycle branch penalty, and executes 4 instructions per cycle???
Branch Prediction

- Always assuming the branch is not taken is a crude form of
- What about loops that are taken 95% of the time?
  - we would like the option of assuming not taken for some branches, and taken for others, depending on ???

Two-bit predictors give better loop prediction

Pipeline performance

```plaintext
loop: lw $15, 1000($2)
    add $16, $15, $12
    lw $18, 1004($2)
    add $19, $18, $12
    beq $19, $0, loop:
```

Control Hazards -- Key Points

• Control (or branch) hazards arise because we must fetch the next instruction before we know if we are branching or where we are branching.
• Control hazards are detected in hardware.
• We can reduce the impact of control hazards through:
  – early detection of branch address and condition
  – branch prediction
  – branch delay slots