Processor (III)
- branch and exception

Hwansoo Han
Branch Hazards

- If branch outcome determined in MEM

Flush these instructions (Set control values to 0)
Reducing Branch Delay

- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator

Example: branch taken

```
36:  sub  $10, $4, $8
40:  beq  $1, $3, 7
44:  and  $12, $2, $5
48:  or   $13, $2, $6
52:  add  $14, $4, $2
56:  slt  $15, $6, $7
...
72:  lw   $4, 50($7)
```

Target address

- $7 \times 4 = 28$
- $44 + 28 = 72$
Example: Branch Taken

- and $12, $2, $5
- beq $1, $3, 7
- sub $10, $4, $8

Clock 3
Example: Branch Taken

lw $4, 50($7)

Bubble (nop)

beq $1, $3, 7

sub $10, ...

before<1>

Clock 4
Data Hazards for Branches

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

```
add $1, $2, $3
add $4, $5, $6
...
beq $1, $4, target
```

- Can resolve using forwarding
Data Hazards for Branches

- If a comparison register is a destination of preceding ALU instruction or 2\textsuperscript{nd} preceding load instruction
  - Need 1 stall cycle

```
lw  $1, addr
add $4, $5, $6
beq stalled
beq $1, $4, target
```
Data Hazards for Branches

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

```
lw   $1, addr
beq  stalled
beq  stalled
beq  $1, $0, target
```
Dynamic Branch Prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction
1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!
  
  ```
  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

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

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

- Branch target buffer
  - Cache of target addresses
  - Indexed by PC when instruction fetched
    - If hit and instruction is branch predicted taken, can fetch target immediately
Exceptions and Interrupts

- “Unexpected” events requiring change in flow of control
  - Different ISAs use the terms differently

- Exception
  - Arises within the CPU
    - e.g., undefined opcode, overflow, syscall, …

- Interrupt
  - From an external I/O controller

- Dealing with them without sacrificing performance is hard
Handling Exceptions

- In MIPS, exceptions managed by a System Control Coprocessor (CP0)

- Save PC of offending (or interrupted) instruction
  - In MIPS: Exception Program Counter (EPC)

- Save indication of the problem
  - In MIPS: Cause register
  - We’ll assume 1-bit
    - 0 for undefined opcode, 1 for overflow

- Jump to handler at 8000 00180
An Alternate Mechanism

- Vectored Interrupts
  - Handler address determined by the cause

- Example:
  - Undefined opcode: C000 0000
  - Overflow: C000 0020
  - ...: C000 0040

- Instructions either
  - Deal with the interrupt, or
  - Jump to real handler
**Handler Actions**

- Read cause, and transfer to relevant handler

- Determine action required

- If restartable
  - Take corrective action
  - use EPC to return to program

- Otherwise
  - Terminate program
  - Report error using EPC, cause, …
Exceptions in a Pipeline

- Another form of control hazard
  - Consider overflow on add in EX stage
    
    add $1, $2, $1
  
  - Prevent $1 from being clobbered
  - Complete previous instructions
  - Flush add and subsequent instructions
  - Set Cause and EPC register values
  - Transfer control to handler

- Similar to mispredicted branch
  - Use much of the same hardware
Pipeline with Exceptions
Exception Properties

- Restartable exceptions
  - Pipeline can flush the instruction
  - Handler executes, then returns to the instruction
    - Refetched and executed from scratch

- PC saved in EPC register
  - Identifies causing instruction
  - Actually PC + 4 is saved
    - Handler must adjust
Exception Example

- Exception on **add** in
  
  40  sub  $11, $2, $4
  44  and  $12, $2, $5
  48  or  $13, $2, $6
  4C  **add**  $1, $2, $1
  50  slt  $15, $6, $7
  54  lw  $16, 50($7)

- Handler
  
  80000180  sw  $25, 1000($0)
  80000184  sw  $26, 1004($0)

...
Exception Example

\[\text{lw } \$16, 50(\$7) \rightarrow \text{slt } \$15, \$6, \$7 \rightarrow \text{add } \$1, \$2, \$1 \rightarrow \text{or } \$13, \ldots \rightarrow \text{and } \$12, \ldots\]
Exception Example

sw $25, 1000($0)

bubble (nop)  bubble  bubble  or $13, ...

Hazard detection unit

Control

ID/EX

multiplexer

ID/Flush

EX/Flush

multiplexer

EX/MEM

multiplexer

MEM/WB

Clock 7

Instruction memory

Registers

ALU

Forwarding unit

Data memory

sign extension

shift left 2

8000186

8000184
Multiple Exceptions

- Pipelining overlaps multiple instructions
  - Could have multiple exceptions at once

- Simple approach – deal with exception from earliest instruction
  - Flush subsequent instructions
  - “Precise” exceptions

- In complex pipelines
  - Multiple instructions issued per cycle
  - Out-of-order completion
  - Maintaining precise exceptions is difficult!
Imprecise Exceptions

- Just stop pipeline and save state
  - Including exception cause(s)

- Let the handler work out
  - Which instruction(s) had exceptions
  - Which to complete or flush
    - May require “manual” completion

- Simplifies hardware, but more complex handler software
- Not feasible for complex multiple-issue out-of-order pipelines