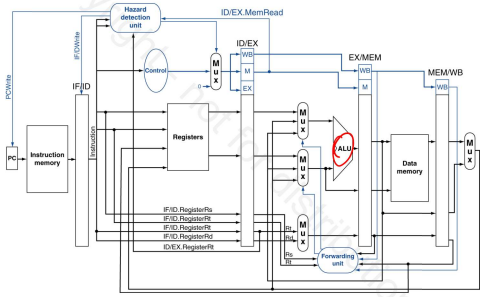
ECE 485/585
Computer Organization and Design

Lecture 11: Branch Hazards and Exceptions
Fall 2022

Won-Jae Yi, Ph.D.

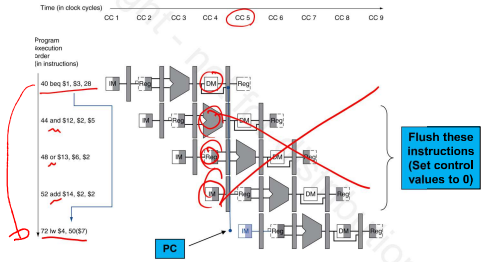
Department of Electrical and Computer Engineering
Illinois Institute of Technology

Datapath with Hazard Detection



Branch Hazards

- If branch outcome determined in MEM

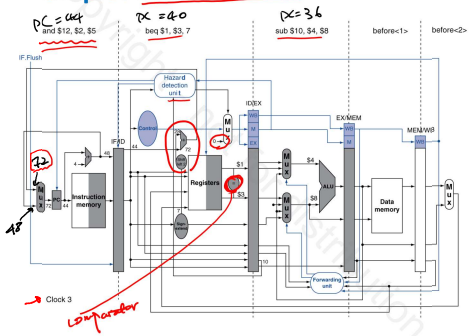


Reducing Branch Delay

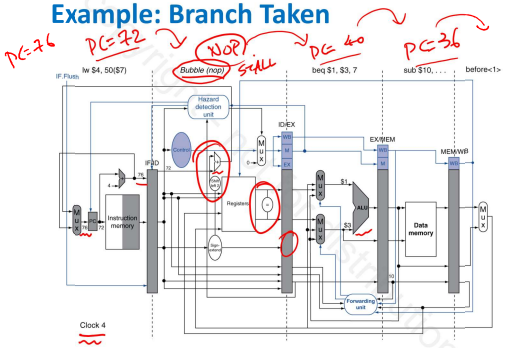
- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- Example: branch taken

```
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)
```


Example: Branch Taken

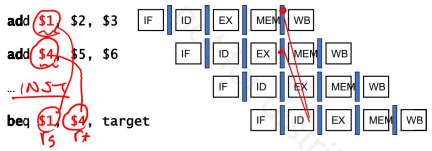


Example: Branch Taken



Data Hazards for Branches

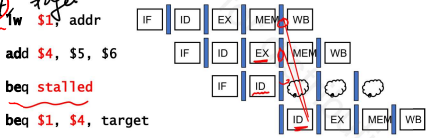
- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction



Can resolve using forwarding

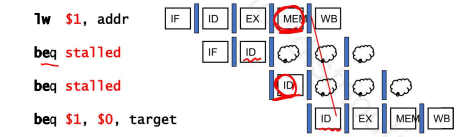
Data Hazards for Branches

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



Data Hazards for Branches

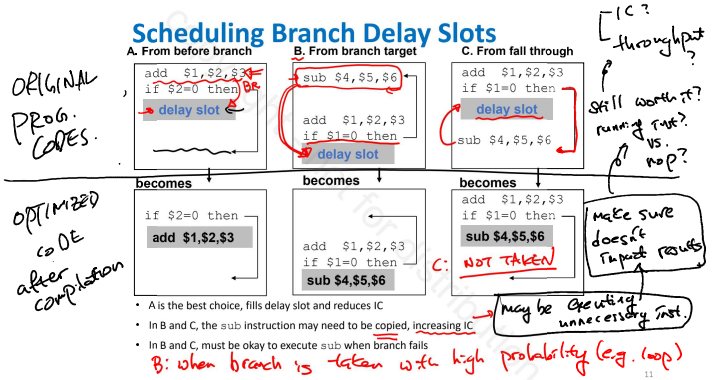
- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles



Delayed Decision

- If the branch hardware has been moved to the ID stage, then we can eliminate all branch stalls with delayed branches which are defined as always executing the next sequential instruction after the branch instruction – the branch takes effect *after* that next instruction
  - MIPS compiler moves an instruction to immediately after the branch that is not affected by the branch (a safe instruction) thereby hiding the branch delay
- With deeper pipelines, the branch delay grows requiring more than one delay slot
  - Delayed branches have lost popularity compared to more expensive but more flexible (dynamic) hardware branch prediction
  - Growth in available transistors has made hardware branch prediction relatively cheaper

Scheduling Branch Delay Slots



- A is the best choice, fills delay slot and reduces IC
- In B and C, the `sub` instruction may need to be copied, increasing IC
- In B and C, must be okay to execute `sub` when branch fails

Dynamic Branch Prediction

- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (a.k.a. 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

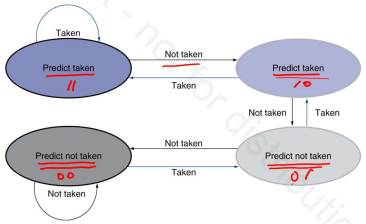
- Inner loop branches mispredicted twice!

```
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 exception handler at 0x80000180

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

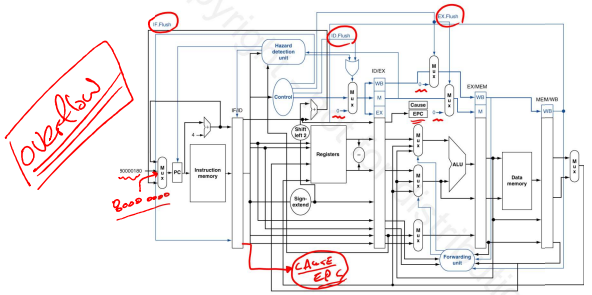
Handler Actions

- 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

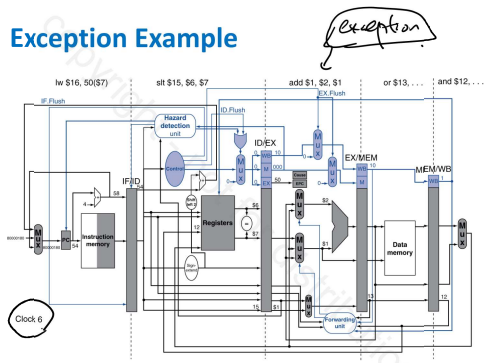
- Exception on `add` in

```
    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)
    …
```

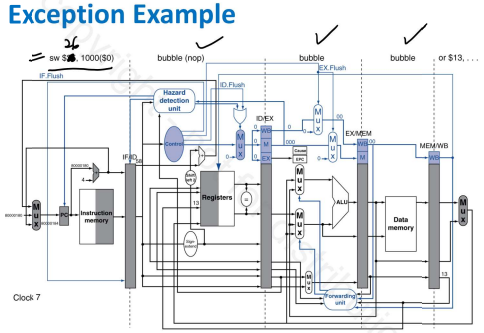
- Handler

```
    0x80000180  sw   $26, 1000($0)
    0x80000184  sw   $27, 1004($0)
    …
```


Exception Example



Exception Example



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