# The Processor: Branch Hazard, Prediction, Exceptions

Dr. Vincent C. Emeakaroha

27-02-2017

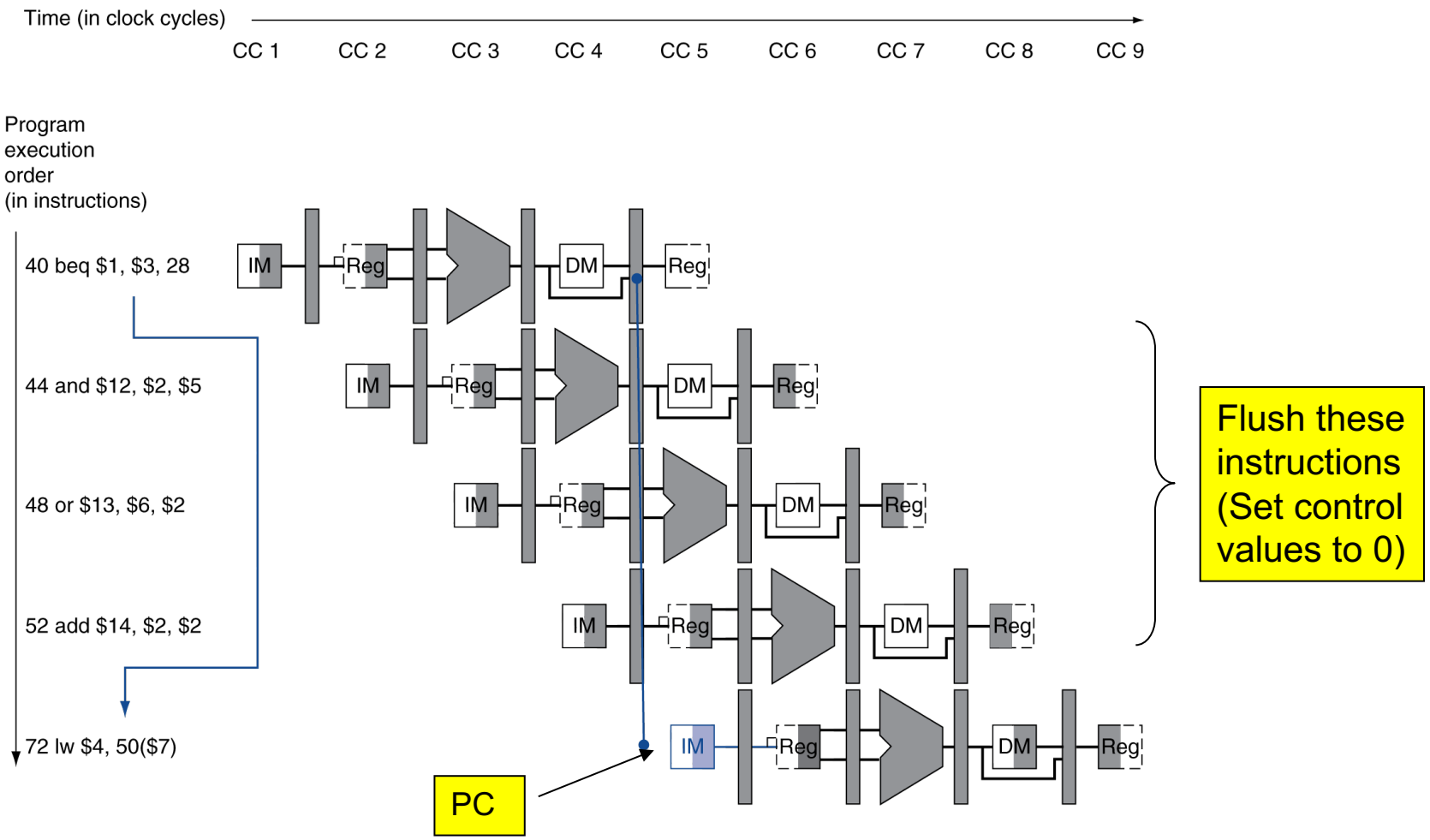
vc.emeakaroha@cs.ucc.ie

# Stalls and Performance

- Stalls reduce performance
  - But are required to get correct results
- Compiler can arrange code to avoid hazards and stalls
  - Requires knowledge of the pipeline structure

# Branch Hazards

- If branch outcome determined in MEM

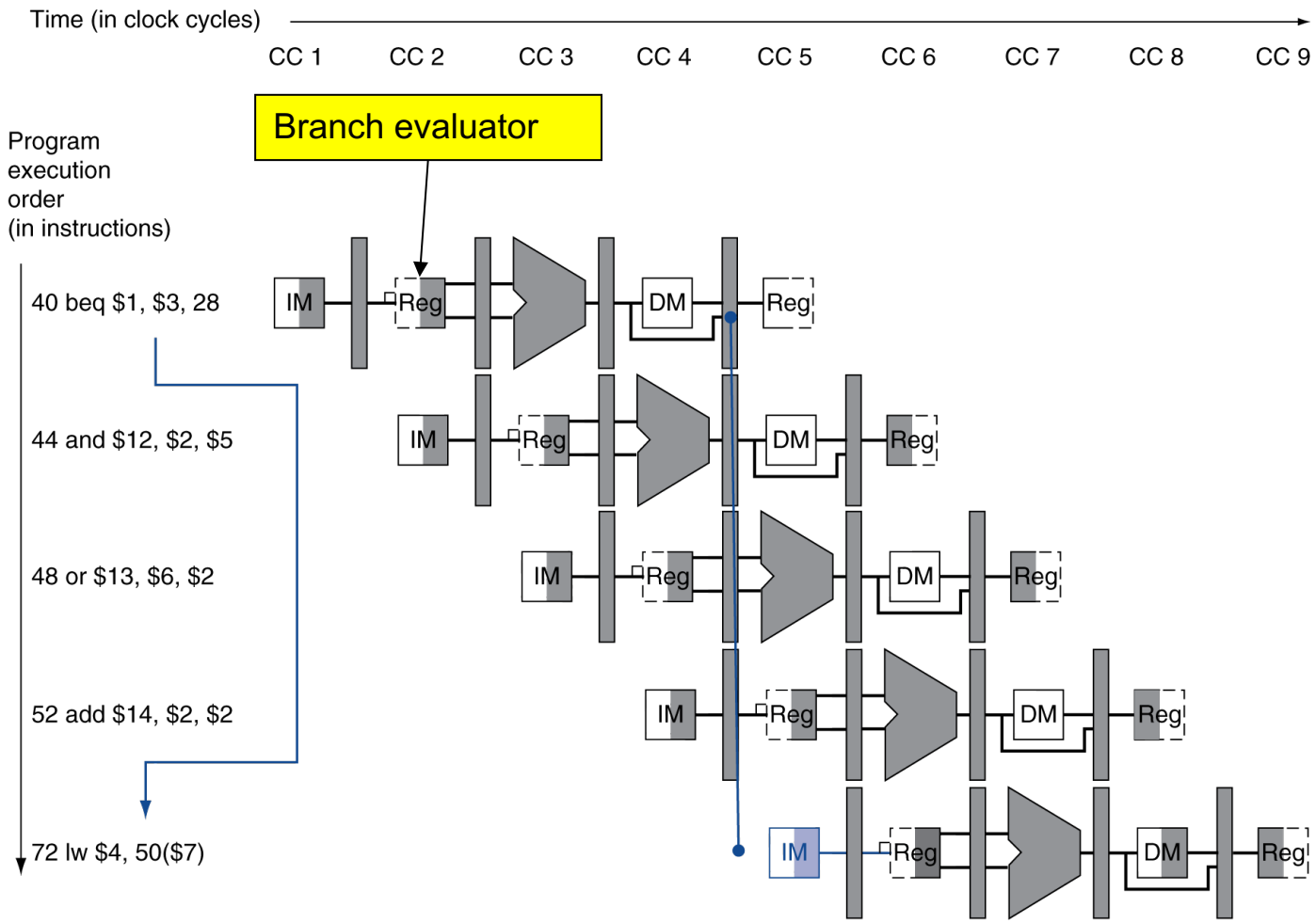


# Reducing Branch Delay

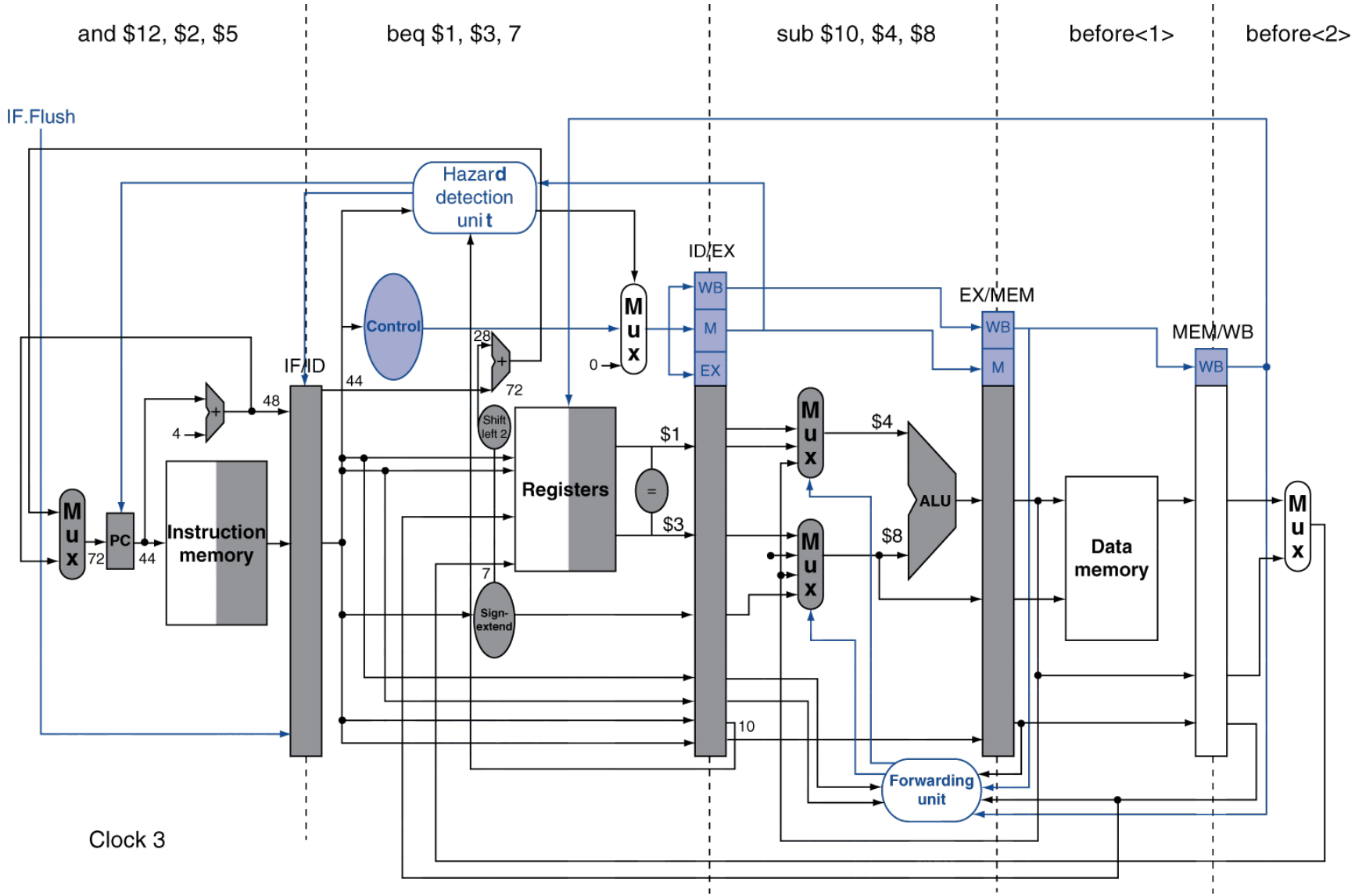
- Move hardware to determine outcome to ID stage
  - Target address adder
  - Register comparator
- Example: branch taken

```
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)
```

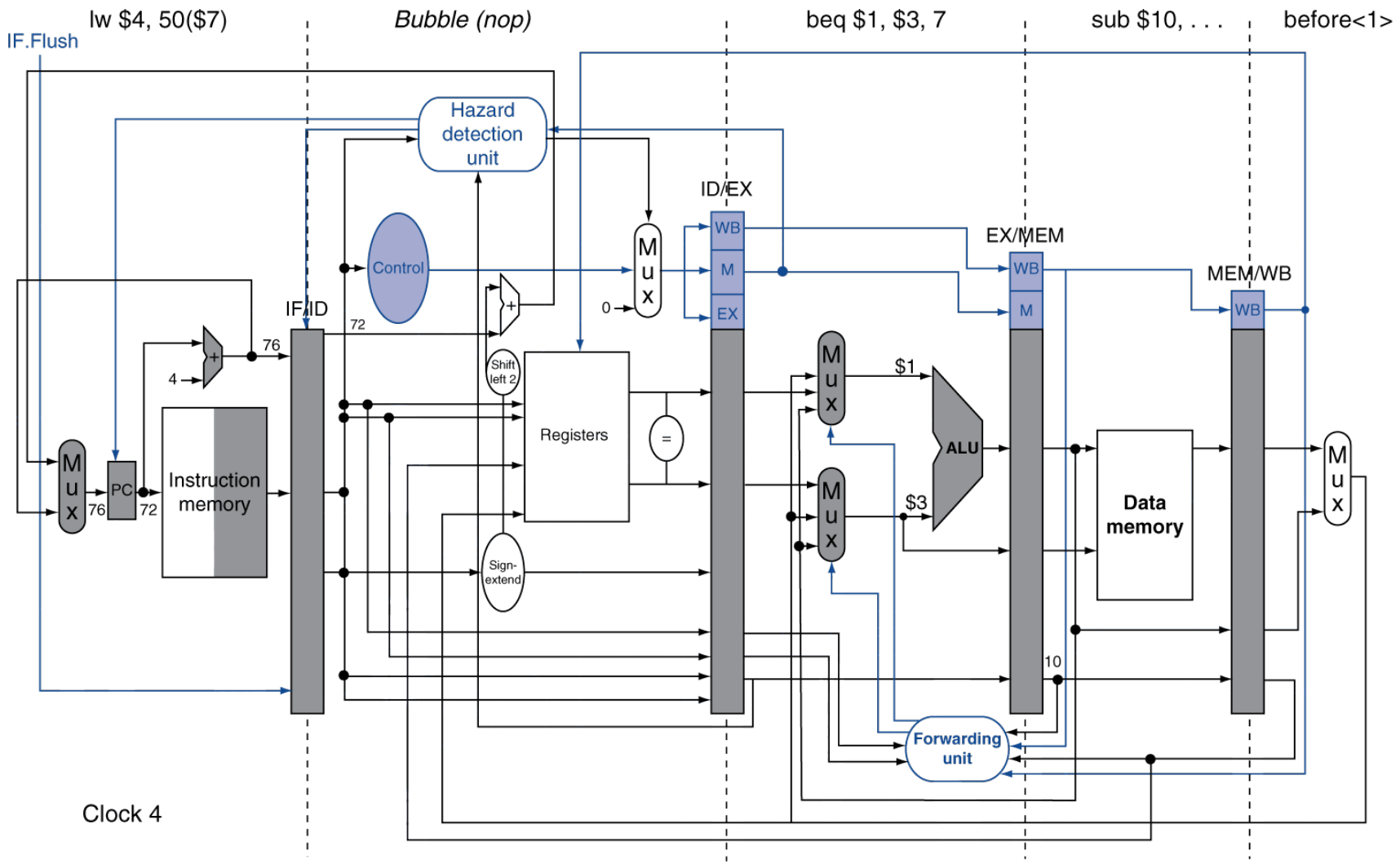
# Branch Delay Reduction



# Branch Taken Implementation



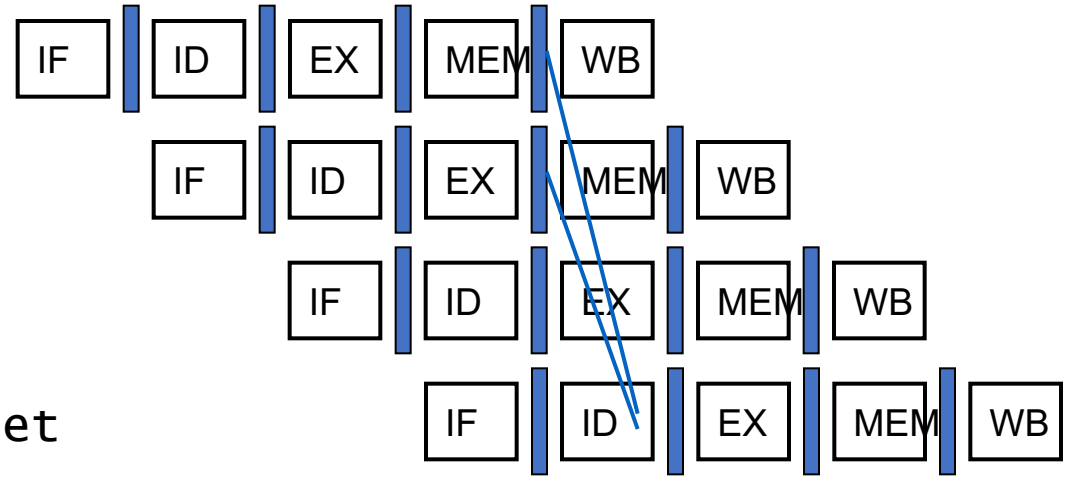
# Branch Taken Implementation



# Data Hazards for Branches

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

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

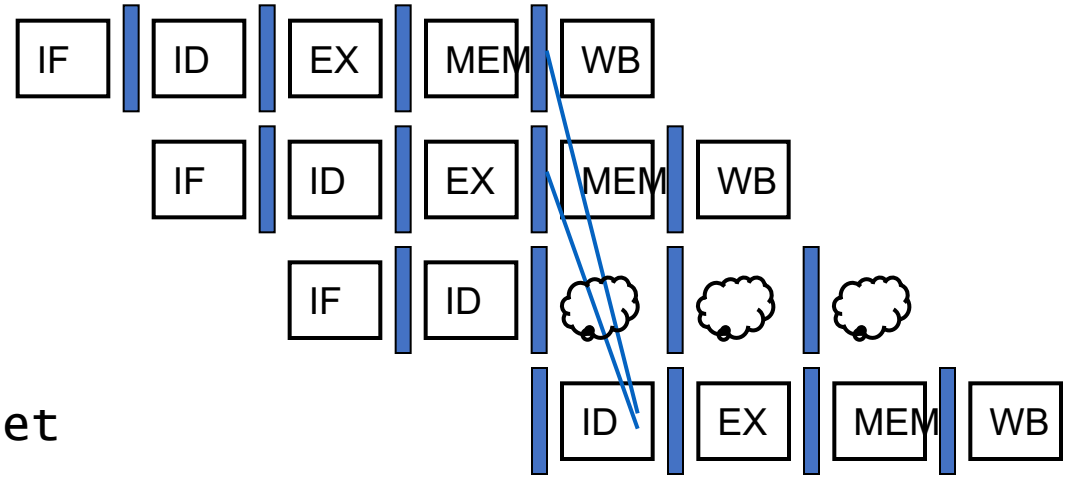



- Can be resolved using forwarding

# Data Hazards for Branches

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

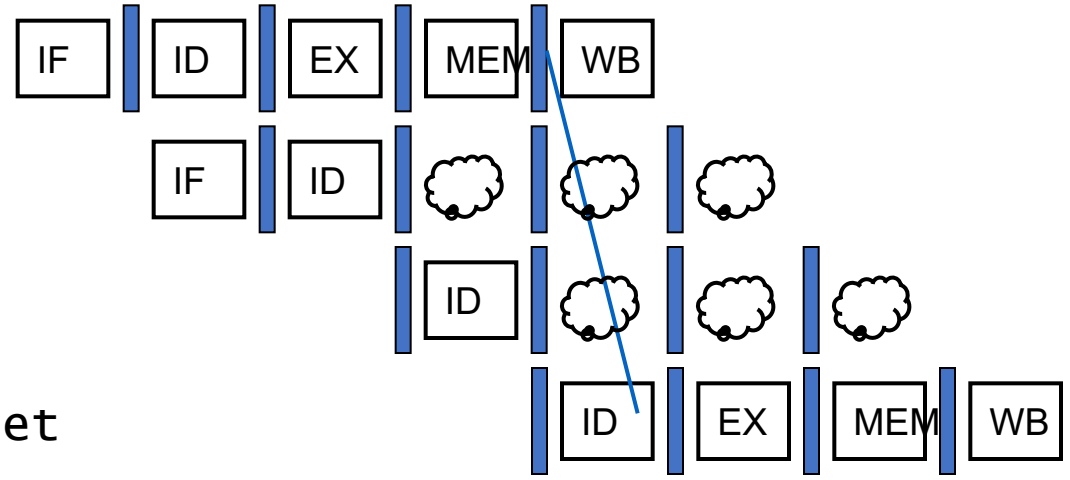
```
lw  $1, addr
add $4, $5, $6
beq stalled
beq $1, $4, target
```



# 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

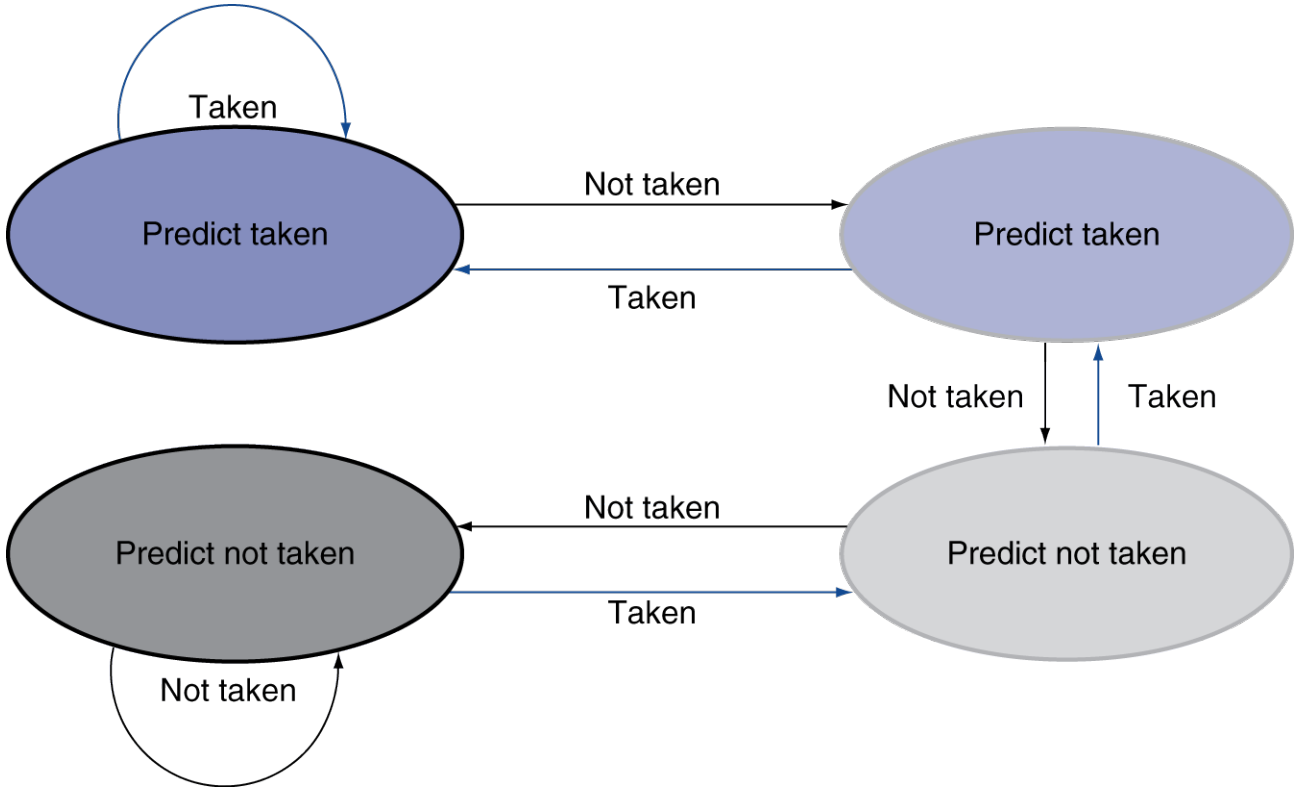
# 1-Bit Predictor: Shortcoming

- Inner loop branches mispredicted twice!


- 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, current implementation generates two types of exceptions
  - Undefined instruction
  - Arithmetic overflow
- Handling steps
  - Save address of offending (or interrupted) instruction in Exception Program Counter (EPC)
  - Save indication of the problem in Cause register
- Transfer execution to operating system
  - Takes appropriate action
  - Uses EPC to determine where to restart program execution

# An Alternate Mechanism

- Vectored Interrupts
  - Handler address determined by the cause
- Example:
  - Undefined instruction: $8000\ 0000_{16}$
  - Overflow: $8000\ 0180_{16}$
  - …: C000 0040
- Operating system decodes exception cause
  - Based on address at which it is initiated

# 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
- Given the instruction sequence

$40_{hex}$ sub $11, $2, $4

$44_{hex}$ and $12, $2, $5

$48_{hex}$ or $13, $2, $6

$4C_{hex}$ add $1, $2, $1

$50_{hex}$ slt $15, $6, $7

$54_{hex}$ lw $16, 50($7)

# Exceptions in a Pipeline

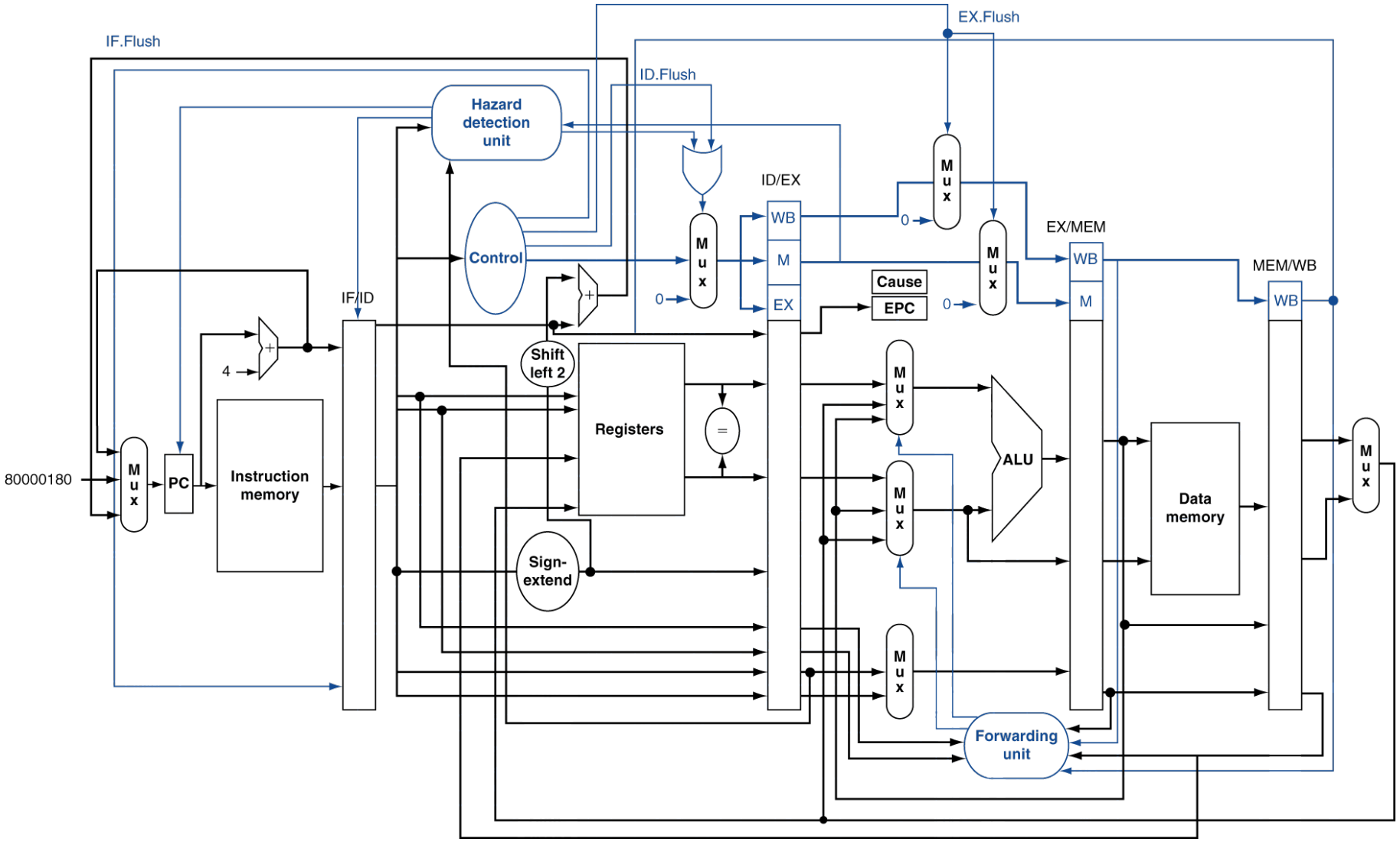
- Consider overflow on add in EX stage

  `add $1, $2, $1`

  - 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

| Handler instruction | | |
|---|---|---|
| $80000180_{hex}$ | slt | $26, 1000 ($0) |
| $80000184_{hex}$ | lw | $27, 1004($) |

# Pipeline with Exceptions



# Exception Properties

- Restartable exceptions
  - Pipeline can flush the instruction
  - Handler executes, then returns to the instruction
    - Refetched and executed from scratch
- Address saved in EPC register
  - Identifies causing instruction
  - Actually PC + 4 is saved
    - Handler must adjust