Course on: “Advanced Computer Architectures”

# Branch Prediction Techniques

Prof. Cristina Silvano
Politecnico di Milano
email: cristina.silvano@polimi.it

## The Problem of Control Hazards

## Conditional Branch Instructions

- A branch is **taken** if the condition is satisfied: the **branch target address** is stored in the Program Counter (PC) instead of the address of the next instruction in the sequential instruction stream (PC + 4).

- Examples of conditional branches for MIPS processor: **beq** (*branch on equal*) and **bne** (*branch on not equal*)
  - `beq $s1, $s2, L1   # go to L1 if ($s1 == $s2)`
  - `bne $s1, $s2, L1   # go to L1 if ($s1 != $s2)`

# Execution of conditional branches for 5-stage MIPS pipeline

**`beq $x,$y,L1`**

| | | | | |
|---|---|---|---|---|
| Instr. Fetch & PC Increm. | Register Read **`$x`** e **`$y`** | ALU Op. **`($x-$y)`** **`& (PC+4+offset)`** | Write of PC | |

- Instruction fetch and PC increment
- Registers read **`($x`** and **`$y)`** from Register File.
- ALU operation to compare registers ($x and $y) to derive **Branch Outcome** (branch taken or branch not taken).
- Computation of **Branch Target Address** **(PC+4+offset):** the value (PC+4) is added to the least significant 16 bit of the instruction after sign extension
- The result of registers comparison from ALU (Branch Outcome) is used to decide the value to be stored in the PC: **(PC+4)** or **(PC+4+offset).**

## Execution of conditional branches for 5-stage MIPS pipeline

| IF<br>Instruction Fetch | ID<br>Instruction Decode | EX<br>Execution | ME<br>Memory Access | WB<br>Write Back |
|---|---|---|---|---|

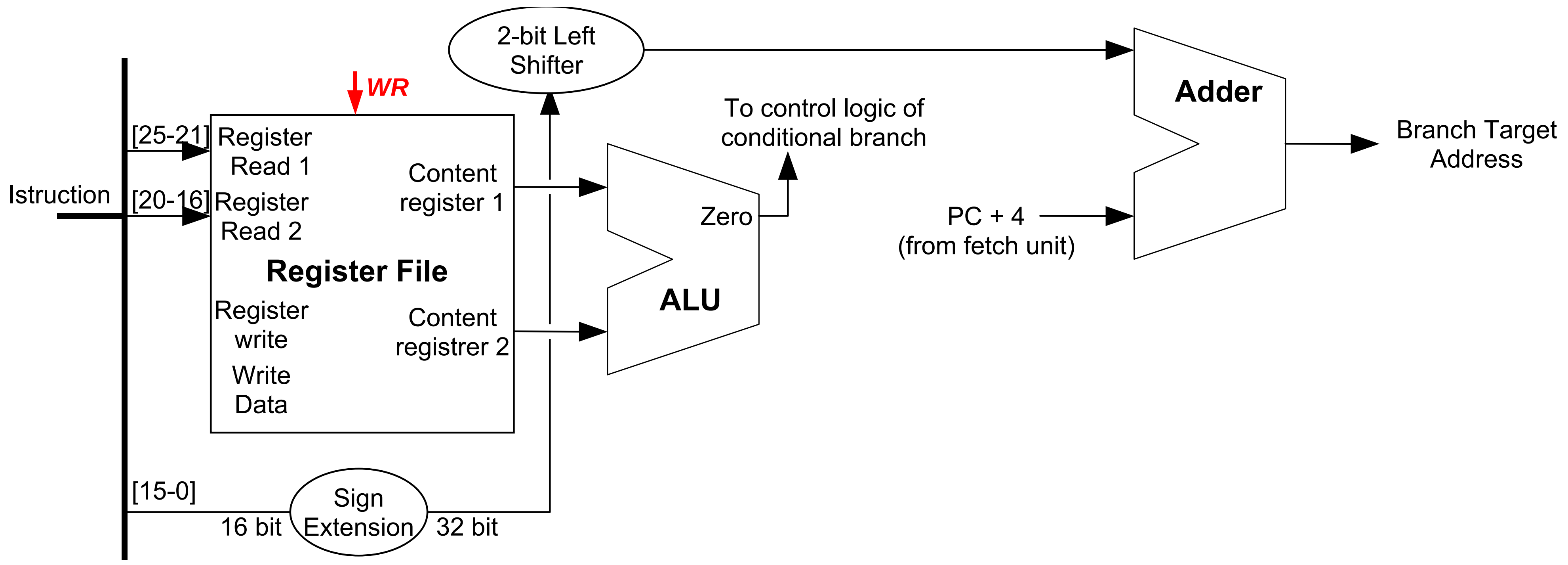
**`beq $x,$y,L1`**

| | | | | |
|---|---|---|---|---|
| Instr. Fetch<br>& PC Increm. | Register Read<br>**`$x`** e **`$y`** | ALU Op. **`($x-$y)`**<br>**`& (PC+4+offset)`** | Write of<br>PC | |

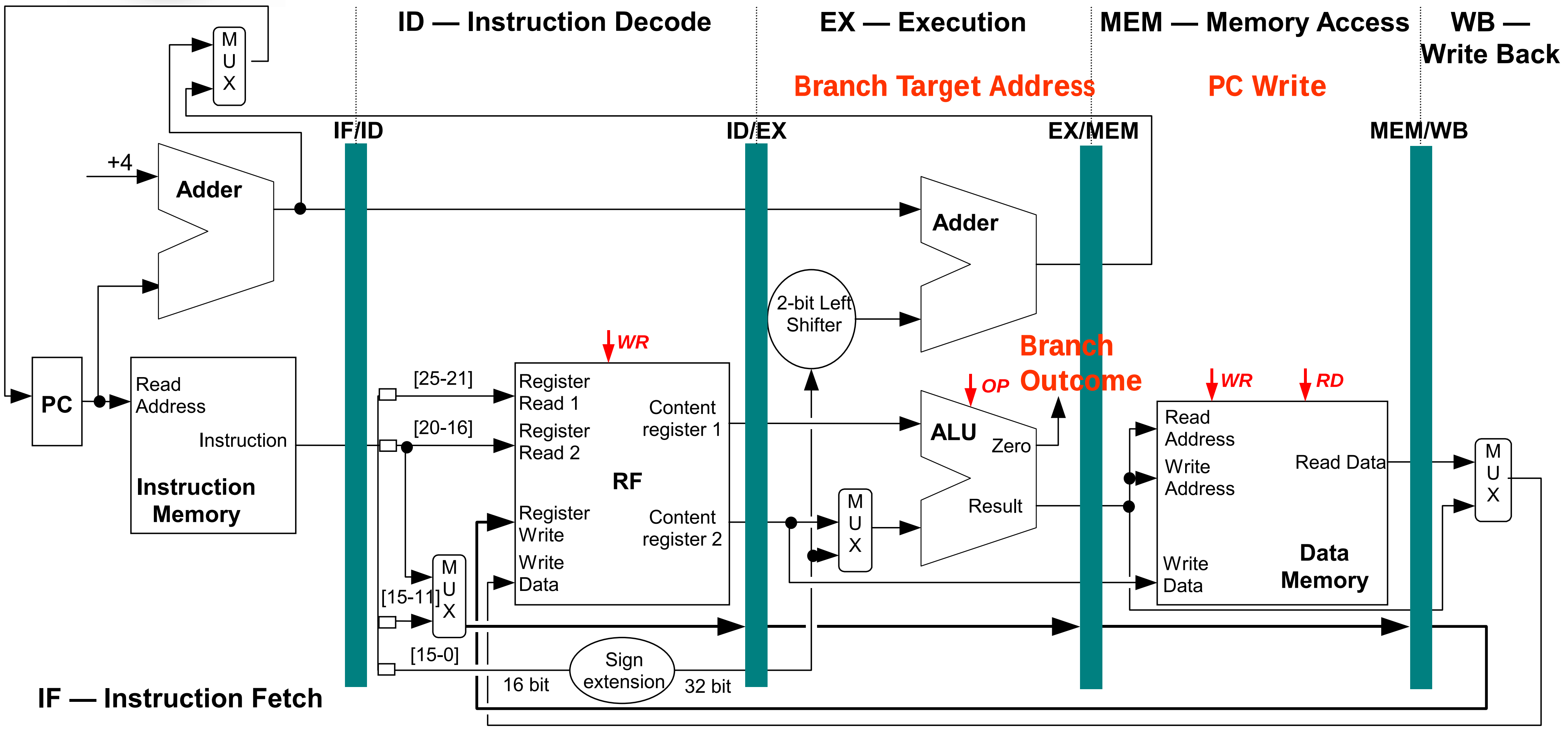
- **Branch Outcome** and **Branch Target Address** are ready at the end of the EX stage (3th stage)
- Conditional branches are solved when **PC** is updated at the end of the ME stage (4th stage)

## Execution of conditional branches for MIPS

- Processor resources to execute conditional branches:



# Implementation of the 5-stage MIPS Pipeline



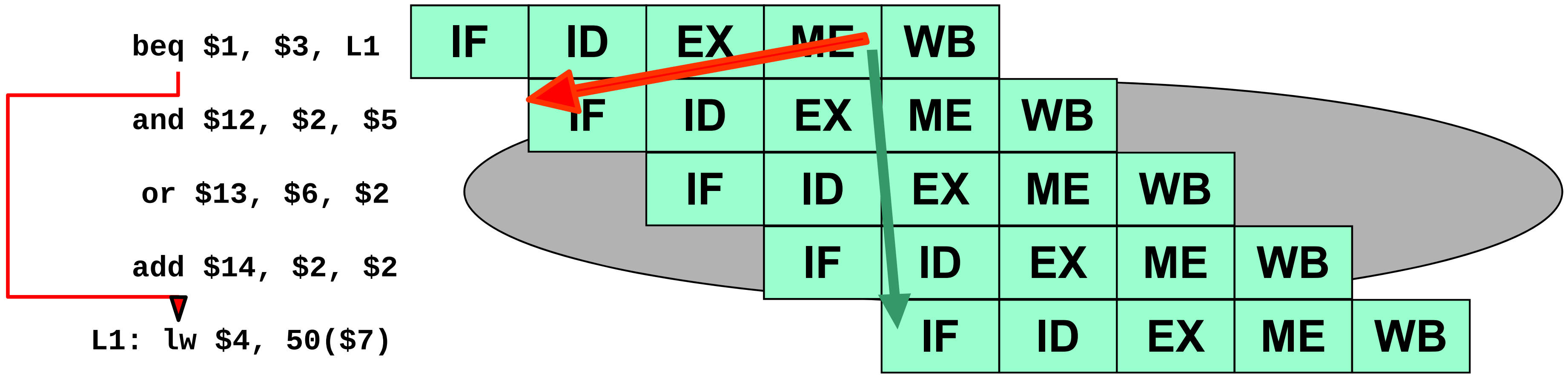
## The Problem of Control Hazards

- **Control hazards:** Attempt to make a decision on the next instruction to fetch before the branch condition is evaluated.
- Control hazards arise from the pipelining of conditional branches and other instructions changing the PC.
- Control hazards reduce the performance from the ideal speedup gained by the pipelining since they can make it necessary to **stall** the pipeline.

## Branch Hazards

- To feed the pipeline we need to fetch a new instruction at each clock cycle, but the branch decision (to change or not change the PC) is taken during the MEM stage.
- This delay to determine the correct instruction to fetch is called **Control Hazard or Conditional Branch Hazard**
- If a branch changes the PC to its target address, it is a **taken branch**
- If a branch falls through, it is **not taken** **or** **untaken.**

# Branch Hazards: Example



- The branch instruction may or may not change the PC in MEM stage, but the next 3 instructions are fetched and their execution is started.
- If the branch is **not taken,** the pipeline execution is fine
- If the branch is **taken,** it is necessary to *flush* the next 3 instructions in the pipeline before they are writing their results, then we need to fetch the **`lw`** instruction at the branch target address **`(L1)`**

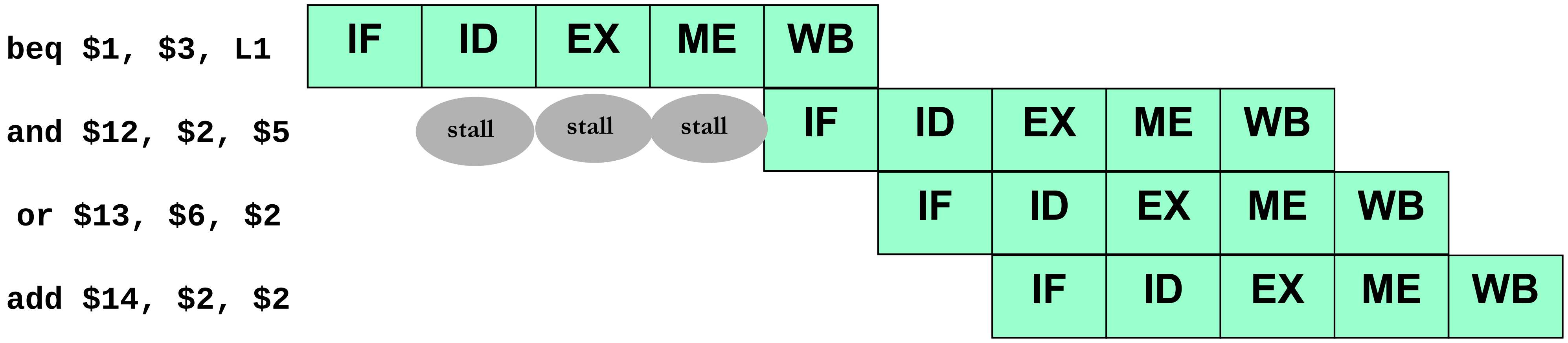
## Branch Hazards: Solutions

- If the branch is *not taken*, introducing three cycles penalty is not justified ⇒ throughput reduction.
- **Solution:** We can assume the *branch not taken*, and **flush** the next 3 instructions in the pipeline only if the branch will be taken. (We cannot assume the *branch taken* because we don't know the branch target address)
- This solution introduces the idea of *branch prediction*
- *But, let's assume to be conservative…*

## Branch Hazards: Conservative assumption

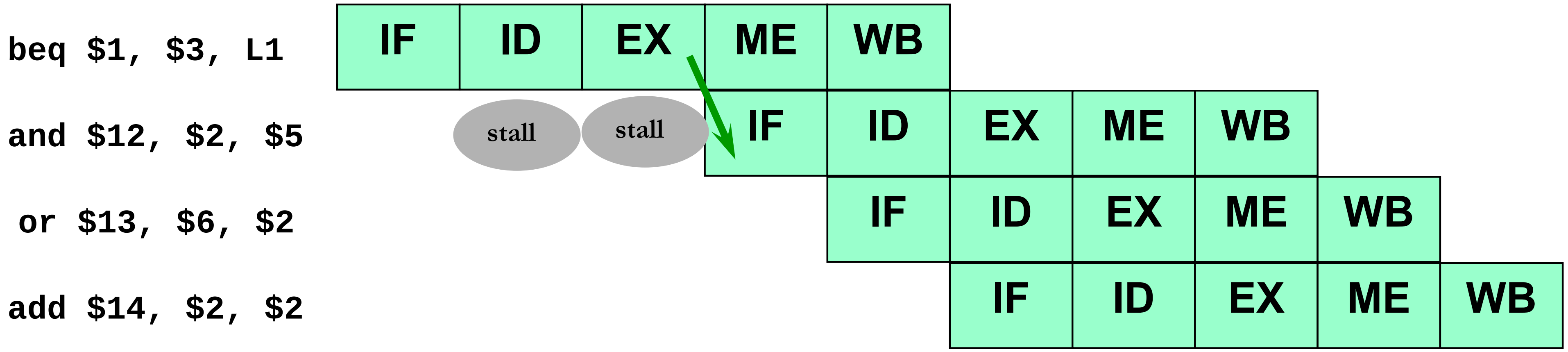
- **Conservative assumption:** To stall the pipeline until the branch decision is taken **(stalling until resolution),** then fetch the correct instruction flow.
  - Without forwarding : We need to stall for **3** clock cycles
  - With forwarding: We need to stall for **2** clock cycles

# Branch Stalls without Forwarding



- **Conservative assumption:** Stalling until resolution at the end of the ME stage.
- Each branch costs **three stalls** to fetch the correct instruction flow: (PC+4) or Branch Target Address

## Branch Stalls with Forwarding



- **Conservative assumption:** Stalling until resolution at the end of the EX stage (when the BO and BTA are known)
- Each branch costs **two stalls** to fetch the correct instruction flow: (PC+4) or Branch Target Address

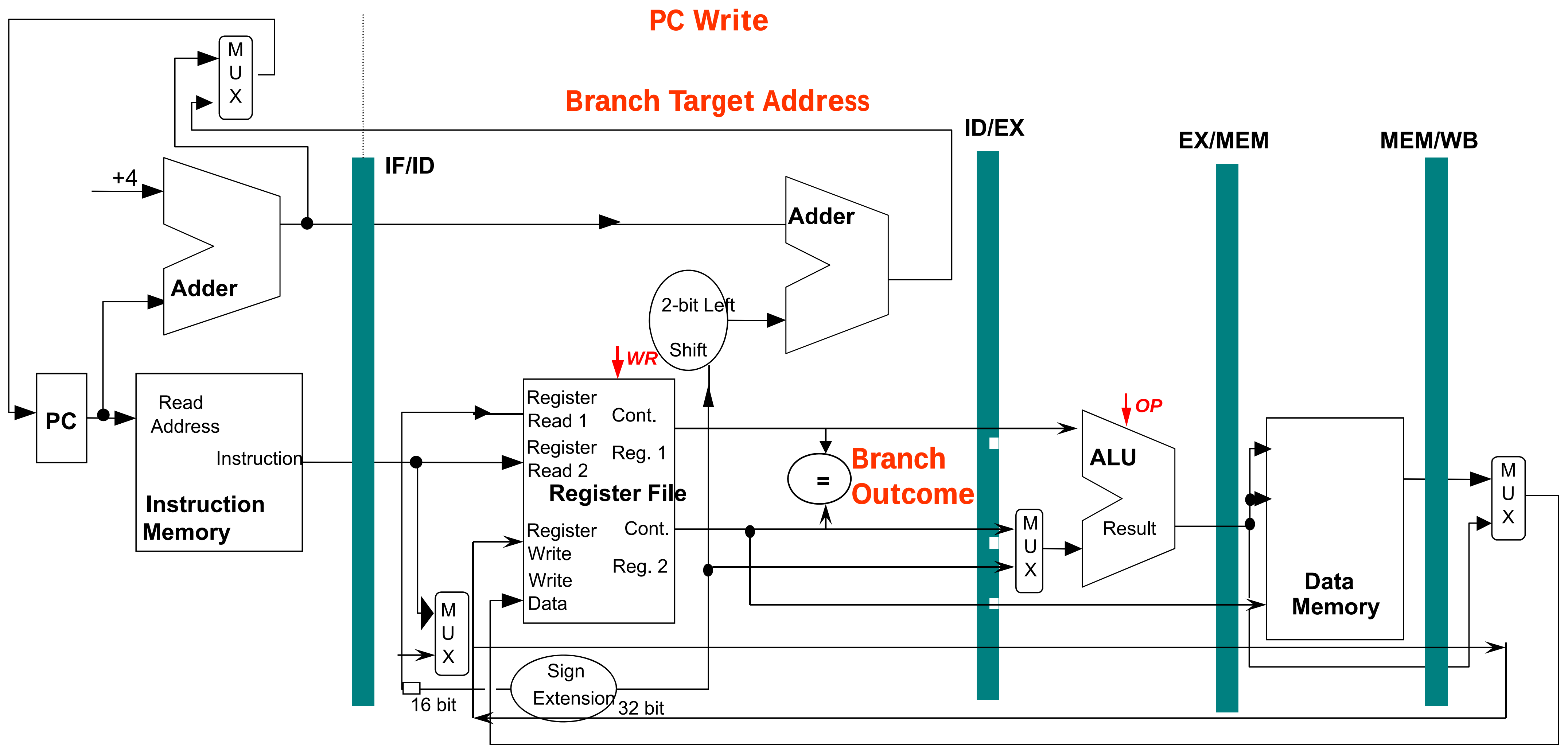
## Early Evaluation of the PC

- To improve performance in case of branch hazards, we need to add more hardware resources to:
  1. Compare registers to derive the **Branch Outcome**
  2. Compute the **Branch Target Address**
  3. Update the PC register

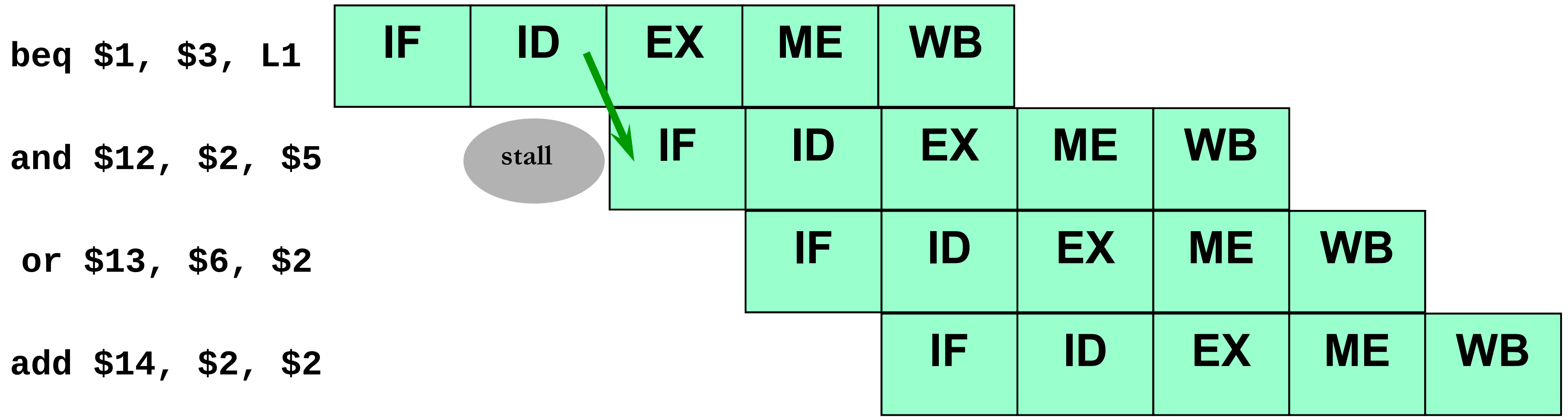
  **as soon as possible in the pipeline.**

- MIPS processor anticipated the comparison of registers, computation of BTA and update of PC *during ID stage.*

# MIPS Processor: Early Evaluation of the PC



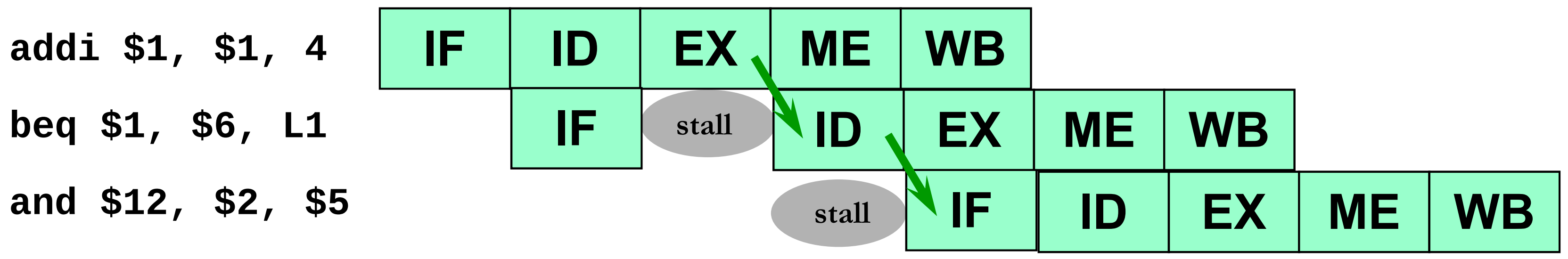
# MIPS Processor: Early Evaluation of the PC



- Conservative assumption: Stalling until resolution at the end of the ID stage (when the BO and BTA are known)
- Each branch costs **one stall** to fetch the correct instruction flow: (PC+4) or Branch Target Address

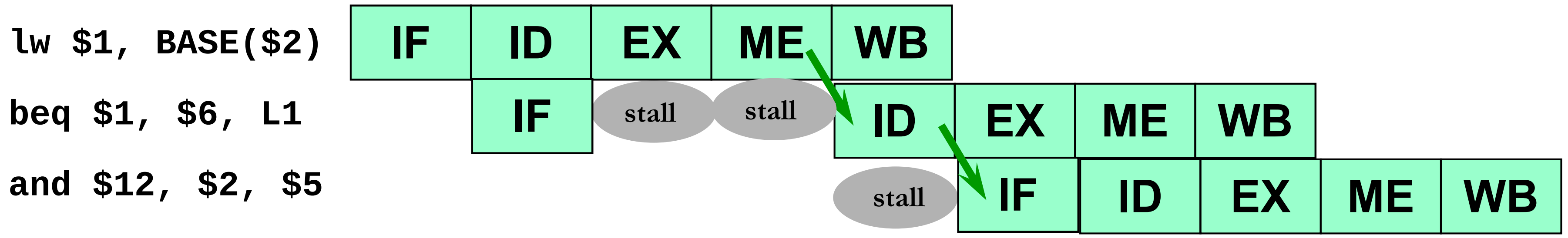
## MIPS Processor: Early Evaluation of the PC

- **Consequence** of early evaluation of the branch decision in ID stage:
  - In case of **add** instruction followed by a **`branch`** testing the result ⇒ we need to introduce **one stall before ID stage of branch** to enable the forwarding (EX-ID) of the result from EX stage of previous instruction. As usual we need one stall **after** the branch for branch resolution.



## MIPS Processor: Early Evaluation of the PC

- **Consequence** of early evaluation of the branch decision in ID stage:
  - In case of **`load`** instruction followed by a **`branch`** testing the result $\Rightarrow$ we need to introduce **two stalls before ID stage of branch** to enable the forwarding (ME-ID) of the result from EX stage of previous instruction. As usual we need one stall **after** the branch for branch resolution.



```
lw $1, BASE($2)
beq $1, $6, L1
and $12, $2, $5
```

## MIPS Processor: Early Evaluation of the PC

- With the branch decision made during ID stage, there is a reduction of the cost associated with each branch **(branch penalty):**
  - We need only **one-clock-cycle stall after** each branch
  - Or a **flush** of only **one** instruction following the branch

- One-cycle-delay for every branch still yields a performance loss of 10% to 30% depending on the branch frequency

- Pipeline Stall Cycles per Instruction due to Branches =
  Branch Frequency x Branch Penalty

- We will examine some branch prediction techniques to deal with this performance loss.