Course on: “Advanced Computer Architectures”

The problem of control hazards

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

Conditional Branch Instructions

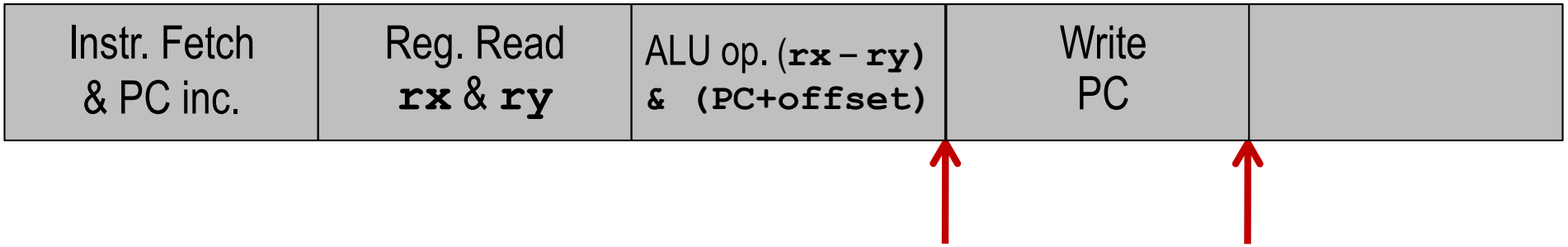
- Examples of conditional branches for RISC-V processor:
  `beq` (*branch on equal*) and **`bne`** (*branch on not equal*)

```
beq rs1, rs2, L1    # go to L1 if (rs1 == rs2)
bne rs1, rs2, L1    # go to L1 if (rs1 != rs2)
```

- The ***Branch Target Address*** is the address where to branch.
- If the branch condition is satisfied => the branch is ***taken*** and the ***Branch Target Address*** is stored in the Program Counter (PC).
- Otherwise => the branch is ***not taken*** **or** ***untaken*** and the instruction stream is executed sequentially with the next instruction address ***(PC + 4).***

Execution of branches for 5-stage RISC-V pipeline

`beq rx,ry,L1`



IF: Instruction Fetch and PC increment

ID: Instruction Decode and Registers Read (**`rx`** and **`ry`**)

EX: Compare registers (**`rx`** and **`ry`**) in the ALU to derive the **Branch Outcome: Taken / Not Taken.**

Computation of **Branch Target Address: (PC+offset)**

ME: The Branch Outcome is used to decide the next PC:

PC <= (PC+4) if Branch Not Taken

PC <= (PC+offset) if Branch Taken

Execution of branches for 5-stage MIPS pipeline

IF Instruction Fetch	ID Instruction Decode	EX Execution	ME Memory Access	WB Write Back

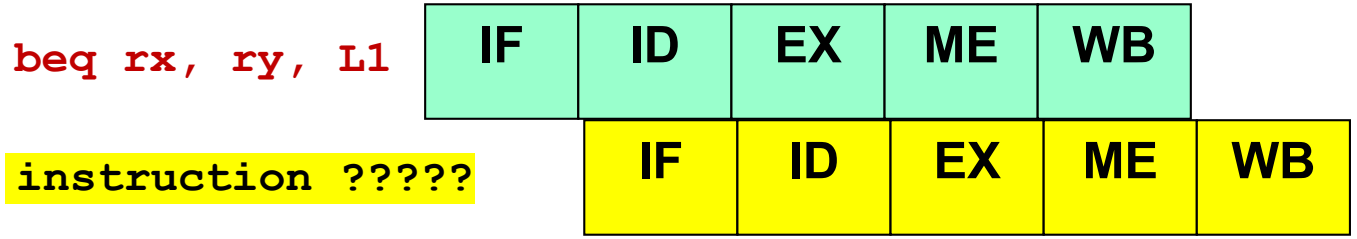
`beq rx,ry,L1`

Instr. Fetch & PC inc.	Reg. Read **`rx`** & **`ry`**	ALU op. (**`rx – ry`**) **`& (PC+offset)`**	Write PC	

- **Branch Outcome (T/NT)** and **BTA** are ready at the end of EX stage.
- Branches are solved when **PC** is updated at the end of ME stage.

Problem of Control Hazards in Pipelining (1)

- 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 not yet ready!

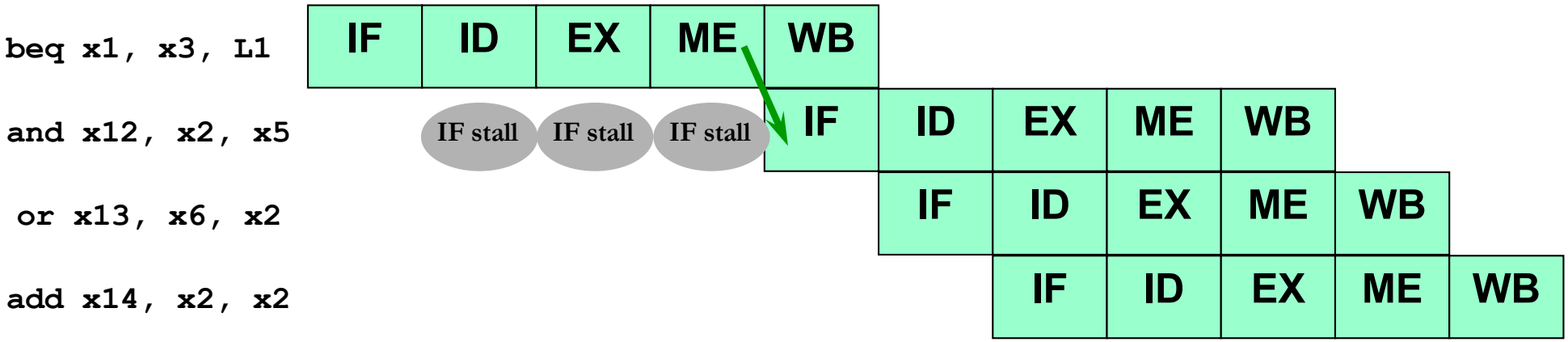


- The problem to choose the correct instruction to be fetched after a branch is called *Control Hazard* **or** *Branch Hazard*

Problem of Control Hazards in Pipelining (2)

- **Control hazards:** Attempt to take a decision on the next instruction to fetch in the pipeline ***before*** the branch condition is evaluated and PC updated.
- **Control hazards** arise from the pipelining of conditional branches and other ***jump*** instructions changing the PC.
- **Control hazards** reduce the performance from the ideal speedup gained by the pipelining because ***it is needed to stall the pipeline until branch resolution.***

Branch Stalls: Conservative Solution



- **Conservative assumption:** ***Stalling until resolution*** at the end of the ME stage of the branch.
- **Performance reduction:** Each branch costs a penalty of ***3 stalls*** to decide and fetch the correct instruction flow in the pipeline.

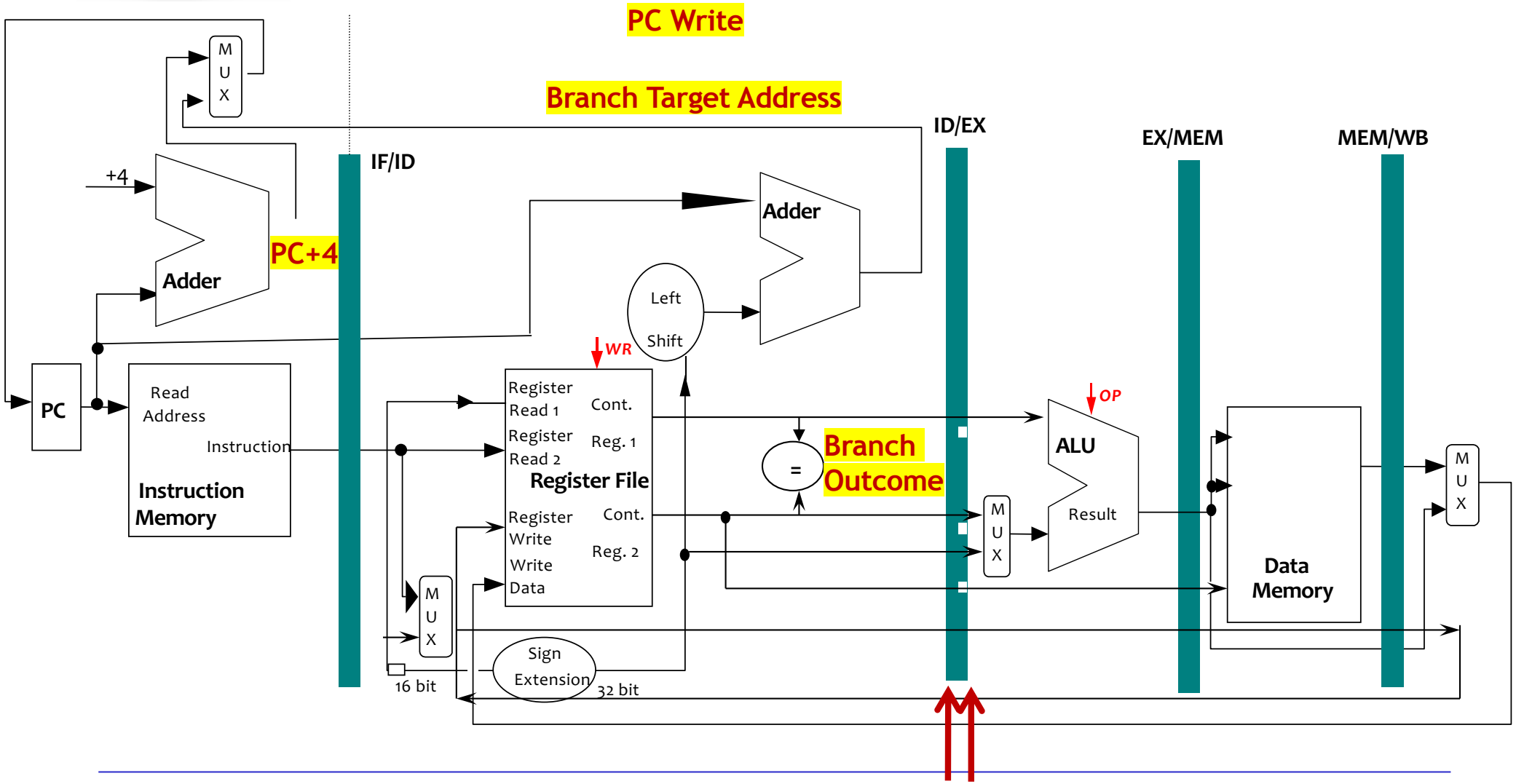
Early Evaluation of branch in ID stage

- **Idea:** 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

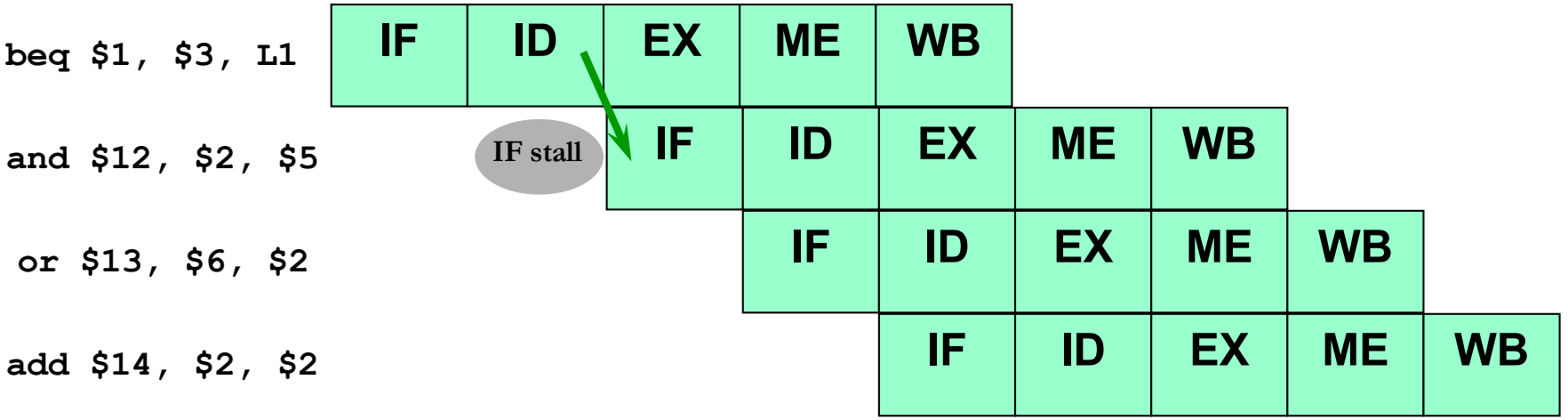
  as soon as possible in the pipeline.

==> *RISC-V optimized pipeline anticipates the 1,2,3 steps **during the ID stage =>** Both BO and BTA are known at the end of ID stage.*

Early Evaluation of branch in ID stage (2)



Early Evaluation of branch in ID stage (3)

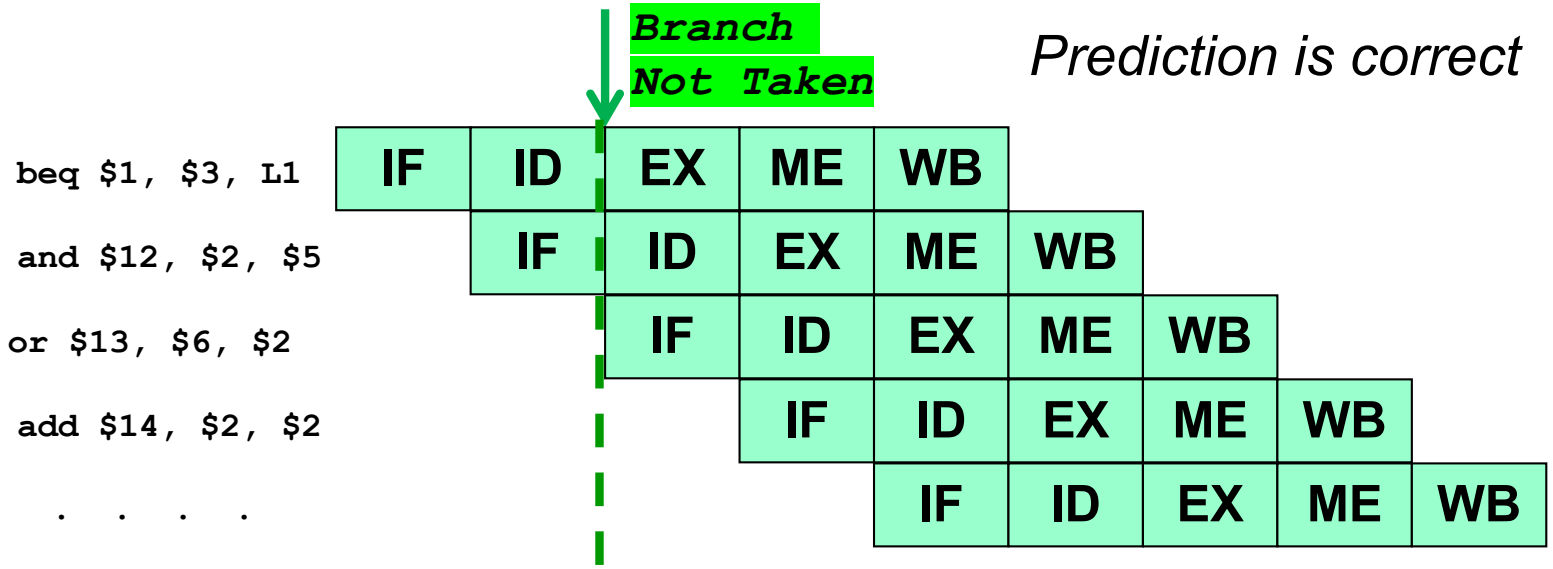


- ***Conservative assumption:*** Stalling until resolution at the end of the ID stage (when the **BO** and **BTA** are known) to decide which instruction to fetch => Now *each branch costs **one stall penalty** to fetch the correct instruction*.

Branch Hazards: Possible Solution

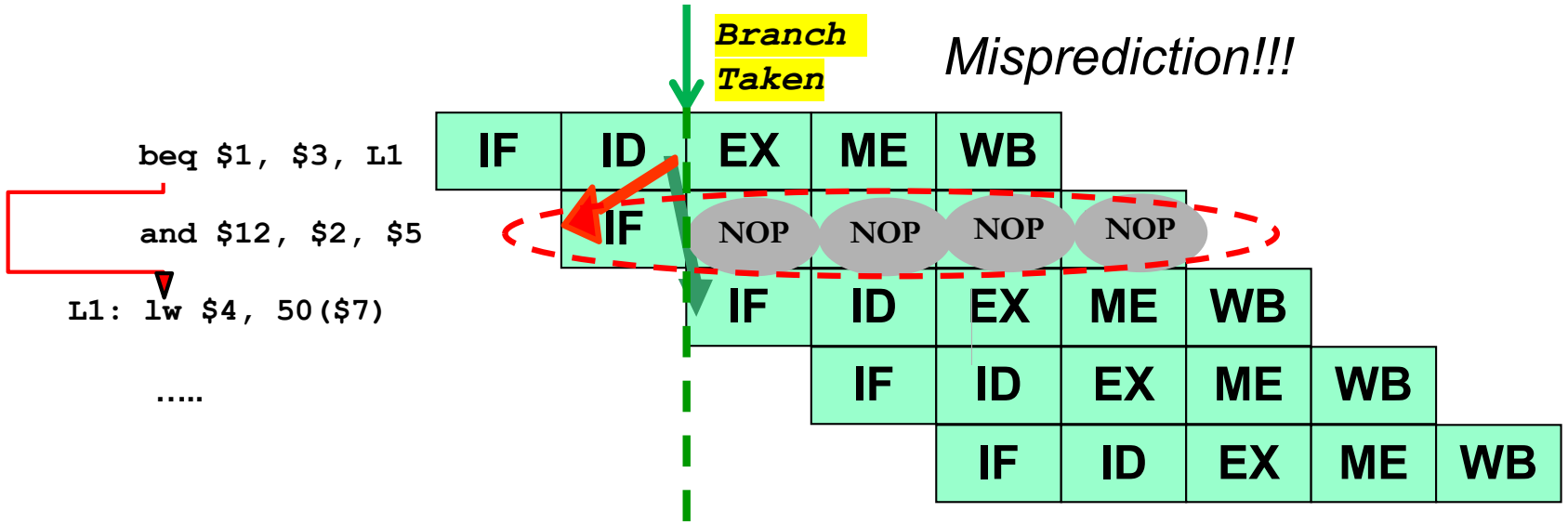
- **Possible solution:** Let's assume the ***branch as not taken*** and continue to execute sequentially.
- ***Only if*** the branch outcome will be ***taken =>*** **flush** the next instruction in the pipeline.
- Be careful: We cannot assume the ***branch as taken*** because we don't know yet the branch target address!
- This solution introduces the idea of ***branch prediction.***

Let's predict the branch as not taken (1)



- Let's assume the branch as ***not taken*** :
  1. If the branch outcome will be ***not taken,*** the pipeline execution of sequential instructions is fine!
  2. ***What happens if the branch will be taken?***

Let's predict the branch as not taken (1)



- If the branch outcome will be ***taken,*** it will be necessary to:
  1. ***Flush*** only ***one*** instructions before writing its results;
  2. Fetch next instruction at the **Branch Target Address (L1)**

=> *In this case, each branch costs **one stall penalty.***

Summary

- With the branch decision is anticipated at ID stage:
  => reduction of the cost associated to each branch **(branch penalty):**
  - We need **only one-clock-cycle IF 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 about 10% to 30% depending on the branch frequency:

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

- Let's introduce in the next chapter some ***branch prediction techniques*** to deal with this performance loss.