High Performance Computing
Lecture 24

Matthew Jacob

Indian Institute of Science

Solving Data Hazards

1. Interlocks & stalling dependent instructions
2. Forwarding or Bypassing
3. Load delay slot
4. Instruction Scheduling
   - Reorder the instructions of the program so that dependent instructions are far enough apart
   - This could be done either
     - by the compiler, before the program runs: Static Instruction Scheduling
     - by the hardware, when the program is running: Dynamic Instruction Scheduling

Static Instruction Scheduling

- Reorder the instructions of the program to eliminate data hazards …
  - or in general to reduce the execution time of the program
- Reordering must be safe

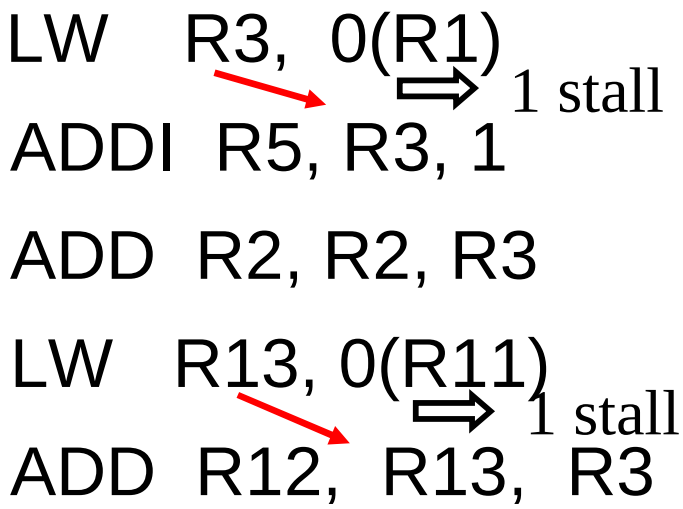
```
ADD   R1,  R2,  R3        /*  R1 = R2 + R3  */
SUB   R2,  R4,  R5        /*  R2 = R4 – R5  */
```

Static Instruction Scheduling

- Reorder the instructions of the program to eliminate data hazards …
  - or in general to reduce the execution time of the program
- Reordering must be safe
  - should not change the meaning of the program
- Two instructions can be exchanged if they are independent of each other

Example: Static Instruction Scheduling

Program fragment:



2 stalls

Scheduling:

LW R3, 0(R1)

ADDI R5, R3, 1

ADD R2, R2, R3

LW R13, 0(R11)

ADD R12, R13, R3

0 stalls

Solving Data Hazards

1. Interlocks & stalling dependent instructions
2. Forwarding or Bypassing
3. Load delay slot
4. Instruction Scheduling
   - Reorder the instructions of the program so that dependent instructions are far enough apart
   - This could be done either
     - by the compiler, before the program runs: Static Instruction Scheduling
     - by the hardware, when the program is running: Dynamic Instruction Scheduling

Kinds of Data Dependence

- True dependence

```
ADD     R1,   R2,   R3
SUB     R4,   R1,   R5
```

- Anti-dependence

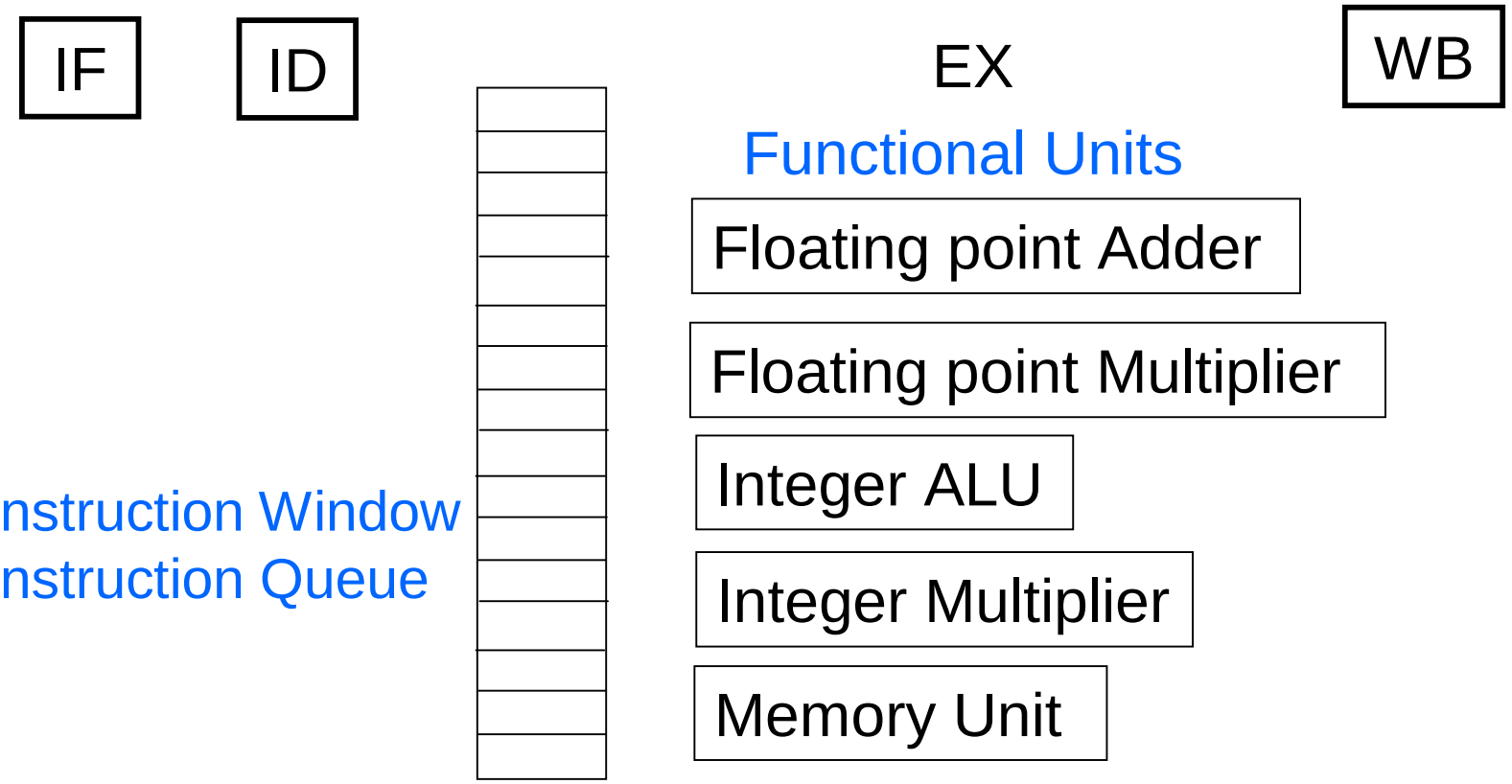
```
ADD     R1,   R2,   R3
SUB     R2,   R4,   R5
```

- Output dependence

```
ADD     R1,   R2,   R3
SUB     R1,   R4,   R5
```

Dynamic Instruction Scheduling

With dynamic instruction scheduling …



Dynamic Instruction Scheduling

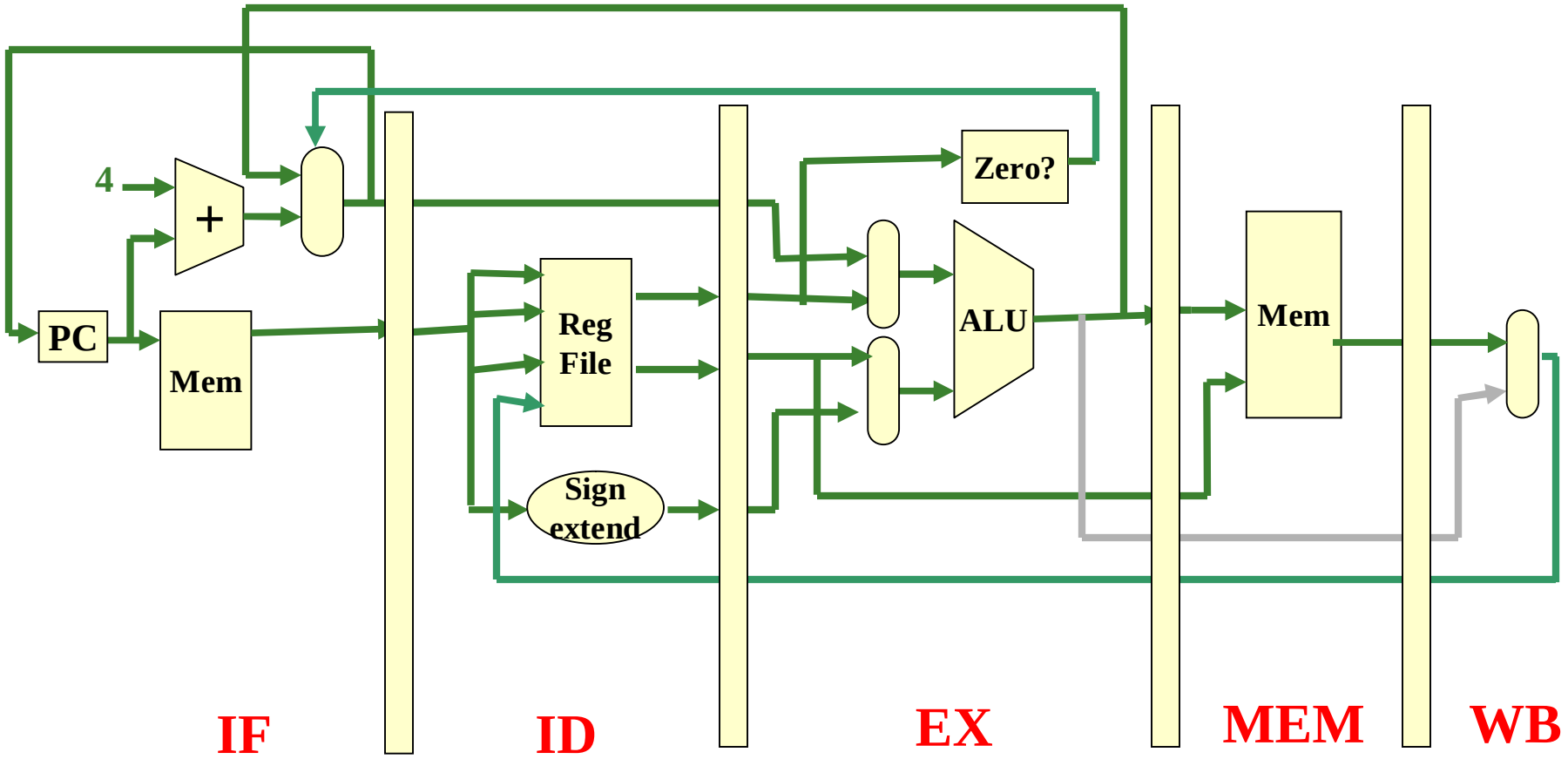
- The hardware dynamically schedules instructions from the Instruction Window for execution on the functional units
- The instructions could execute in an order that is different from that specified by the program
  - with the same result
- Such processors are called “out of order” processors
  - as opposed to “in order” processors

Problem: Pipeline Hazards

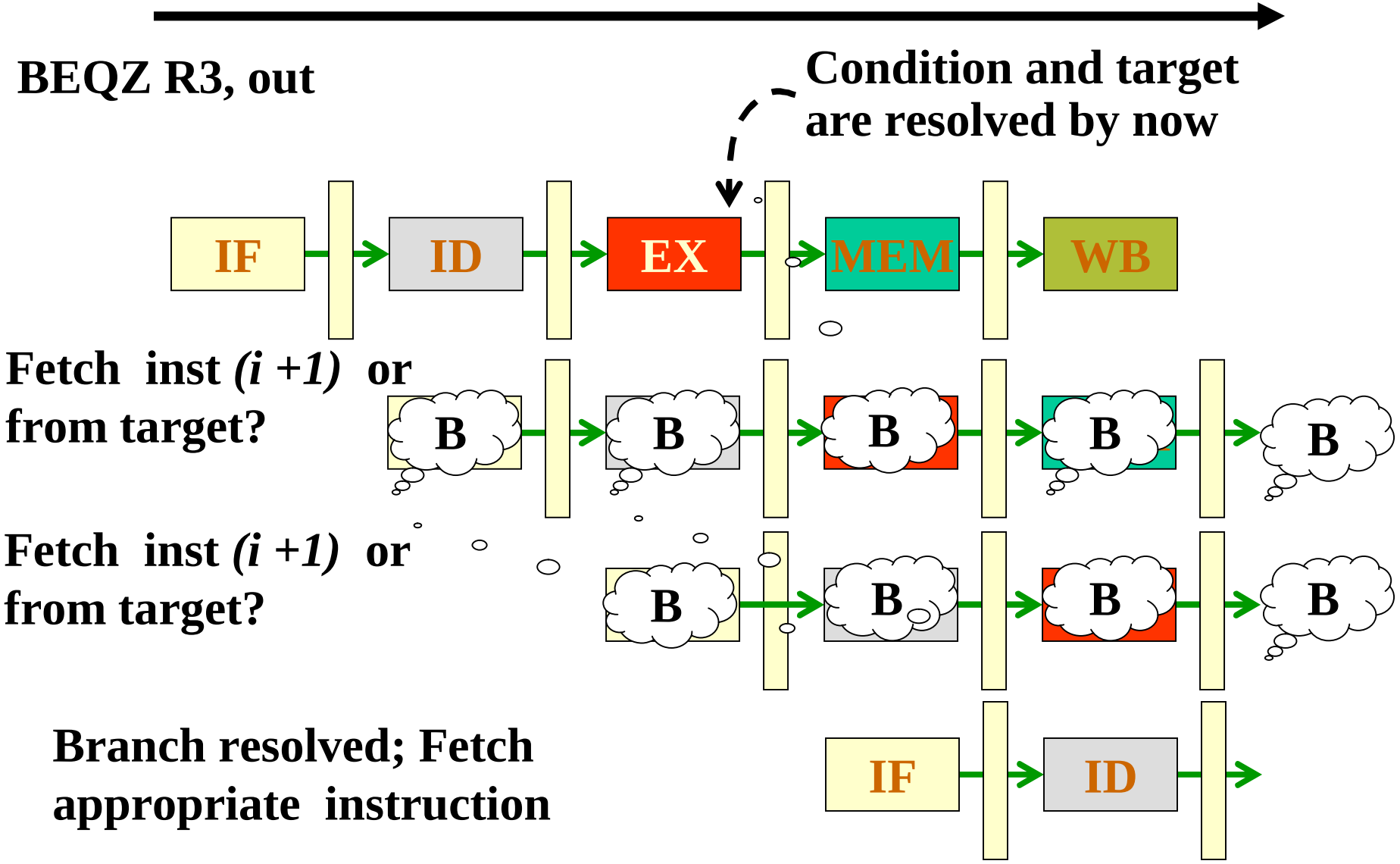
A situation where an instruction cannot proceed through the pipeline as it should

1. Structural hazard: When 2 or more instructions in the pipeline need to use the same resource at the same time
2. Data hazard: When an instruction depends on the data result of a prior instruction that is still in the pipeline
3. Control hazard: A hazard that arises due to control transfer instructions

Recall: Execution of Branch Instruction



Control Hazards



Control Hazards

- Observation: Since the branch is resolved only in the EX stage, there must be 2 stall cycles after every conditional branch instruction

Reducing Impact of Branch Stall

- The execution of a conditional branch instruction involves 2 activities
  1. evaluating the branch condition (determine whether it is to be taken or not-taken)
  2. computing the branch target address
- To reduce branch stall effect we could
  - evaluate the condition earlier (in ID stage)
  - compute the target address earlier (in ID stage)
- The number of stall cycles would then be reduced to 1 cycle

Control Hazard Solutions

1. Static Branch Prediction

   Prediction?

   reasoning about the future

   guessing what is going to happen

   Static

   The behaviour of a branch instruction is predicted once before the program starts executing

Prediction and Correctness

- Prediction: guessing what is going to happen
- What if the guess is incorrect?
  - The pipelined processor hardware must be built to detect the misprediction and take appropriate corrective action

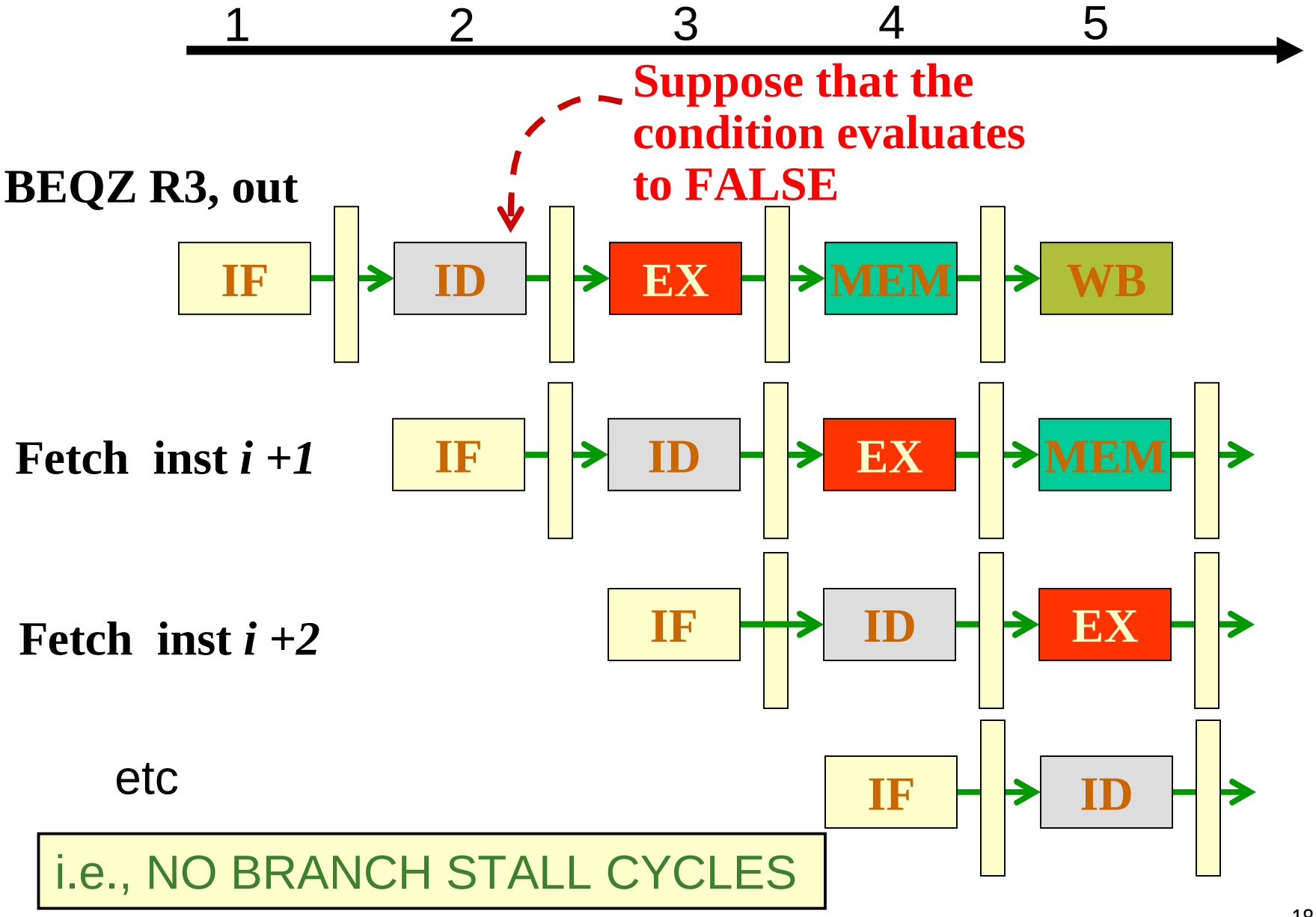
Control Hazard Solutions

1. Static Branch Prediction

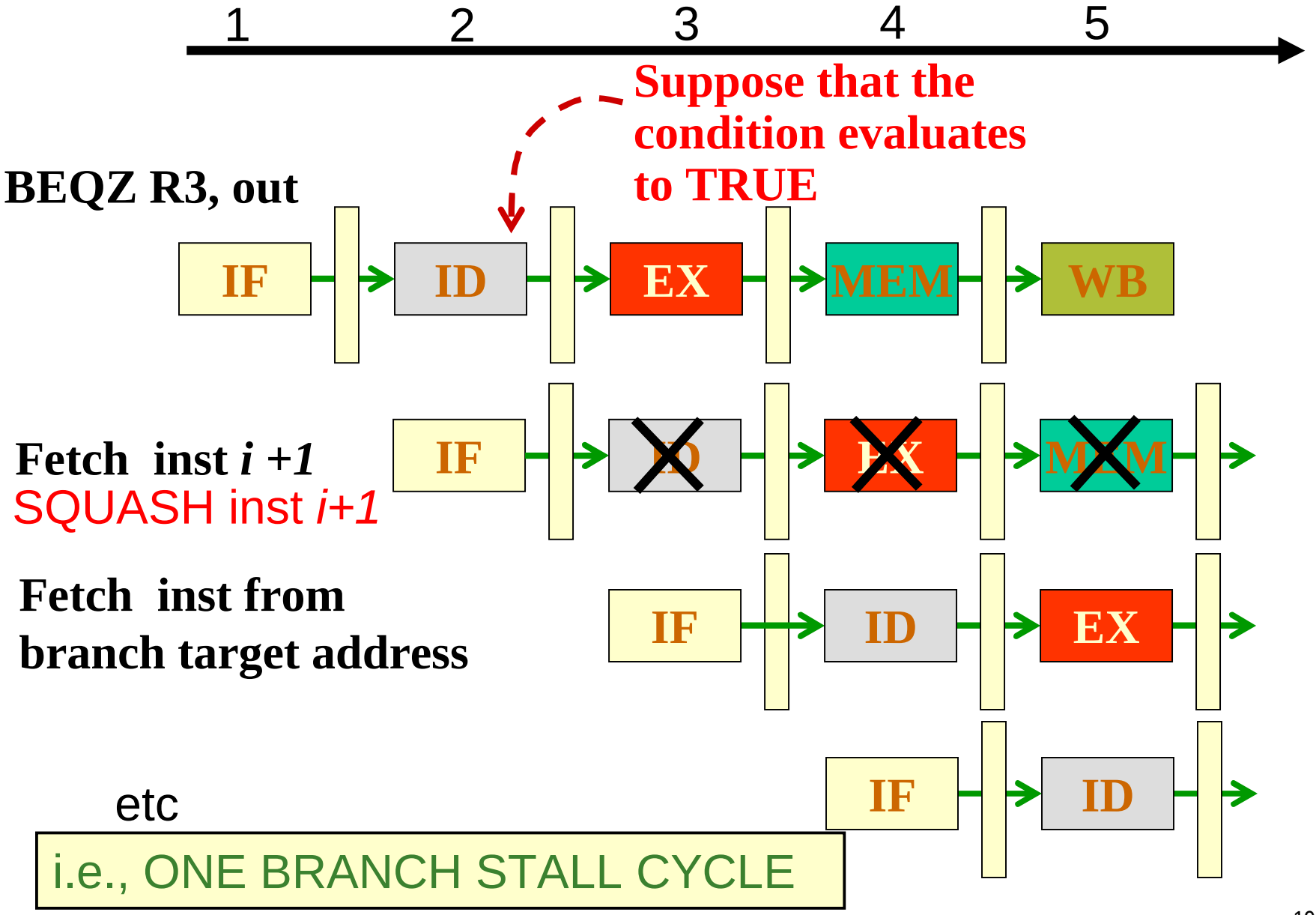
   Example: Static Not-Taken policy

   - The hardware is built to fetch next from PC + 4
   - After ID stage, if it is found that the branch condition is false (i.e., not taken), continue with the fetched instruction (from PC + 4)
     - Else, squash the fetched instruction and re-fetch from the branch target address
       - squash: cancel, annul the processing of that instruction

Static Not-Taken Branch Prediction



Static Not-Taken Branch Prediction



i.e., ONE BRANCH STALL CYCLE

Control Hazard Solutions

1. Static Branch Prediction

   Example: Static Not-Taken policy

   - The hardware is built to fetch next from PC + 4
   - After ID stage, if it is found that the branch condition is false (i.e., not taken), continue with the fetched instruction (from PC + 4) 0 stall cycles
     - Else, squash the fetched instruction and re-fetch from the branch target address 1 stall cycle
   - Thus, average branch penalty < 1 cycle

Control Hazard Solutions

1. Static Branch Prediction
2. Delayed Branching

- Design hardware so that control transfer takes place after a few of the following instructions

```
BEQ   R1, R2, target
ADD   R3, R2, R3
```

Recall: Interesting ISA Notes

- For load instructions: the loaded value might not be available in the destination register for use by the instruction immediately following the load
  - LOAD DELAY SLOT
- For control transfer instructions: the transfer of control takes place only following the instruction immediately after the control transfer instruction
  - BRANCH DELAY SLOT

Control Hazard Solutions

1. Static Branch Prediction
2. Delayed Branching

- Design hardware so that control transfer takes place after a few of the following instructions

```
BEQ   R1, R2, target
ADD   R3, R2, R3
```

- Delay slots: following instructions that are executed whether or not the branch is taken
- Stall cycles are avoided if the delay slots are filled with useful instructions