Other Types of Data Hazards

* Our pipeline is in-order

Definition: In an in-order pipeline (such as ours), a preceding instruction is always ahead of a succeeding instruction in the pipeline. Modern processors however use out-of-order pipelines that break this rule. It is possible for later instructions to execute before earlier instructions.

* We will only have RAW hazards in our pipeline.
* Out-of-order pipelines can have WAR and WAW hazards

WAW Hazards

```
[1]: add r1, r2, r3
[2]: sub r1, r4, r3
```

* Instruction [2] cannot write the value of r1, before instruction [1] writes to it, will lead to a WAW hazard

WAR Hazards

```
[1]: add r1, r2, r3
[2]: add r2, r5, r6
```

* Instruction [2] cannot write the value of r2, before instruction [1] reads it $\longrightarrow$ will lead to a WAR hazard

```
[1] add r1, r2, r3
[2] ld r4, 8[r1]
[3] sub r5, r5, r2
[4] st r4, 12[r2]
[5] sub r6, r4, r3
[6] add r7, r4, r3
```

Q. How many dependencies?

Q. How many dependencies require conflict resolution?

Control Hazards

```
[1]: beq .foo
[2]: mov r1, 4
[3]: add r2, r4, r3
...
...
.foo:
[100]: add r4, r1, r2
```

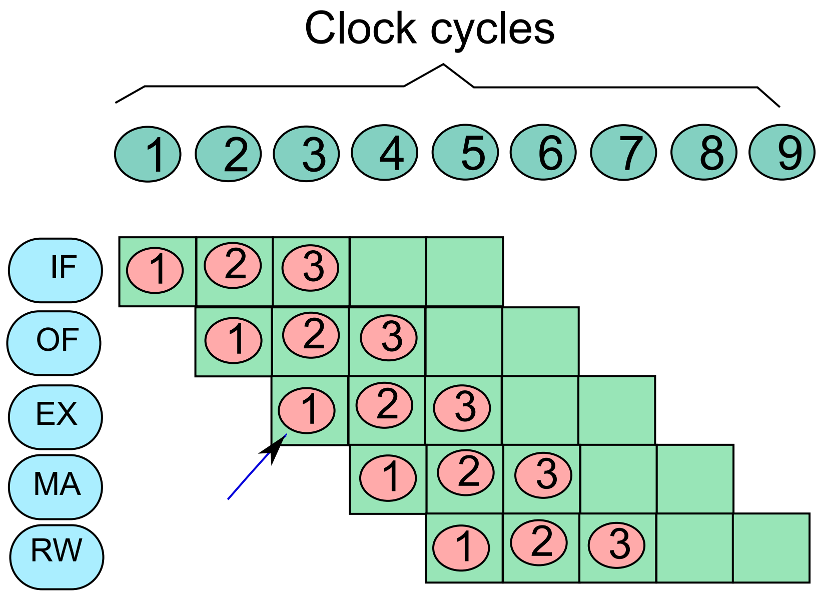
* If the branch is taken, instructions [2] and [3], might get fetched, incorrectly

Control Hazard – Pipeline Diagram

```
[1]: beq .foo

[2]: mov r1, 4

[3]: add r2, r4, r3
```



* The two instructions fetched immediately after a branch instruction might have been fetched incorrectly.

```
[1] mov r1, 0
[2] mov r2, 0
[3] cmp r1, r2
[4] beq .foo
[5] sub r6, r4, r3
[6] add r7, r4, r3
[7] mul r8, r4, r3
…
…
.foo
[100] add r5, r4, r1
```

Control Hazards

* The two instructions fetched immediately after a branch instruction might have been fetched incorrectly.
* These instructions are said to be on the wrong path
* A control hazard represents the possibility of erroneous execution in a pipeline because instructions in the wrong path of a branch can possibly get executed and save their results in memory, or in the register file

Structural Hazards

- A structural hazard may occur when two instructions have a conflict on the same set of resources in a cycle
- Example :
  - Assume that we have an add instruction that can read one operand from memory
  - add r1, r2, 10[r3]

Structural Hazards - II

```
[1]: st r4, 20[r5]
[2]: sub r8, r9, r10
[3]: add r1, r2, 10[r3]
```

* This code will have a structural hazard
  * [3] tries to read 10[r3] (MA unit) in cycle 4
  * [1] tries to write to 20[r5] (MA unit) in cycle 4
* Does not happen in our pipeline

Solutions in Software

* Data hazards
  * Insert nop instructions, reorder code

```
[1]: add r1, r2, r3
[2]: sub r3, r1, r4
```

```
[1]: add r1, r2, r3
[2]: nop
[3]: nop
[4]: nop
[5]: sub r3, r1, r4
```

McGraw Hill Education

Code Reordering

```
add r1, r2, r3
add r4, r1, 3
add r8, r5, r6
add r9, r8, r5
add r10, r11, r12
add r13, r10, 2
```

Code Reordering

```
add r1, r2, r3
add r4, r1, 3
add r8, r5, r6
add r9, r8, r5
add r10, r11, r12
add r13, r10, 2
```

```
add r1, r2, r3
add r8, r5, r6
add r10, r11, r12
nop
add r4, r1, 3
add r9, r8, r5
add r13, r10, 2
```

Control Hazards

* Trivial Solution : Add two nop instructions after every branch
* Better solution :
  * Assume that the two instructions fetched after a branch are valid instructions
  * These instructions are said to be in the delay slots
  * Such a branch is known as a delayed branch

Example with 2 Delay Slots

```
add r1, r2, r3
add r4, r5, r6
b .foo
add r8, r9, r10
```

```
b .foo
add r1, r2, r3
add r4, r5, r6
add r8, r9, r10
```

* The compiler transfers instructions before the branchto the delay slots.

* If it cannot find 2 valid instructions, it inserts nops.