

CSE301 – Computer Organization

Lecture 5 – Pipeline Processor

Converting between single cycle processor to pipeline

The Single-Cycle Processor executes an entire instruction within one clock cycle. The major limitation of this design is that the duration of the clock cycle is dictated by the longest-running instruction, causing simpler instructions to waste time and resulting in low overall throughput.

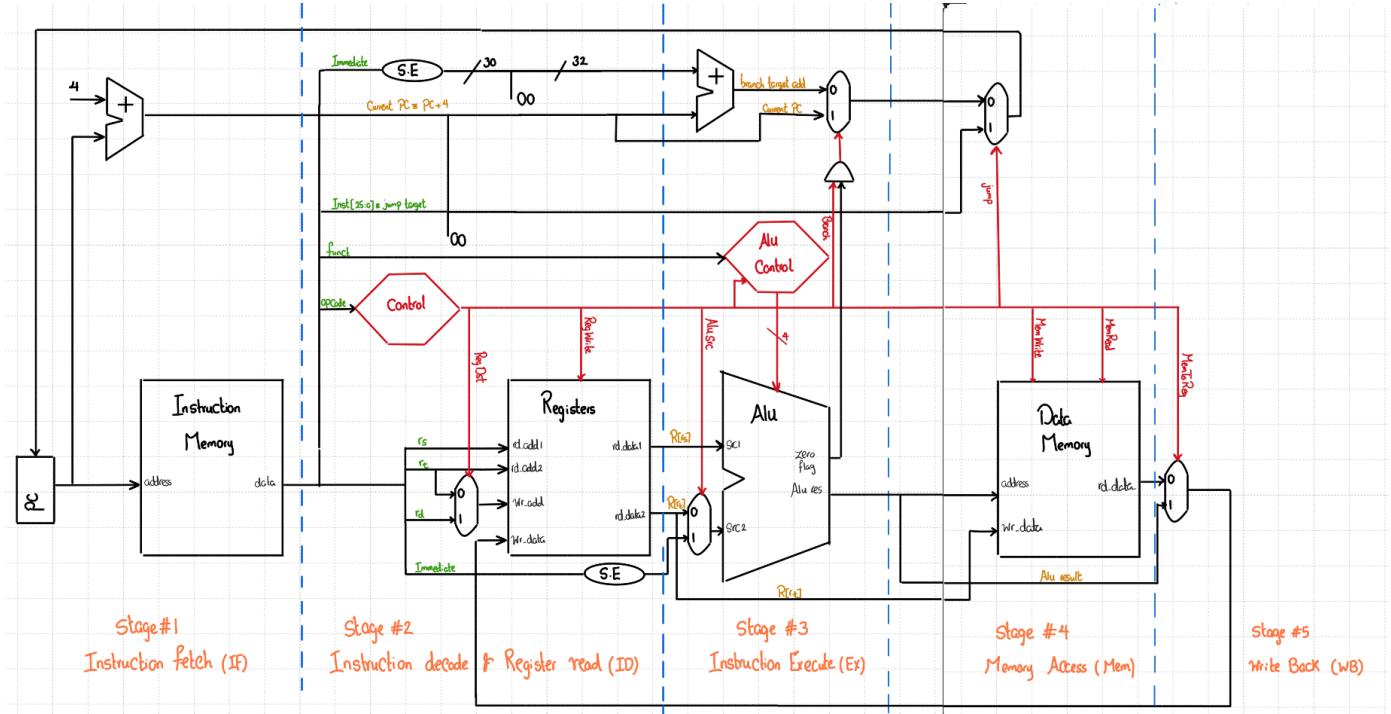


Figure 1: Single Cycle Processor

The Pipelined Processor uses the same basic functional units as the single-cycle design, but achieves massive performance gains by dividing the instruction process into discrete stages (e.g., IF, ID, EX, MEM, WB).

The critical component introduced for this transformation is the **Interstage Register** (also known as a Pipeline Register). These registers are placed between each stage to hold the results of the preceding stage until the next clock cycle.

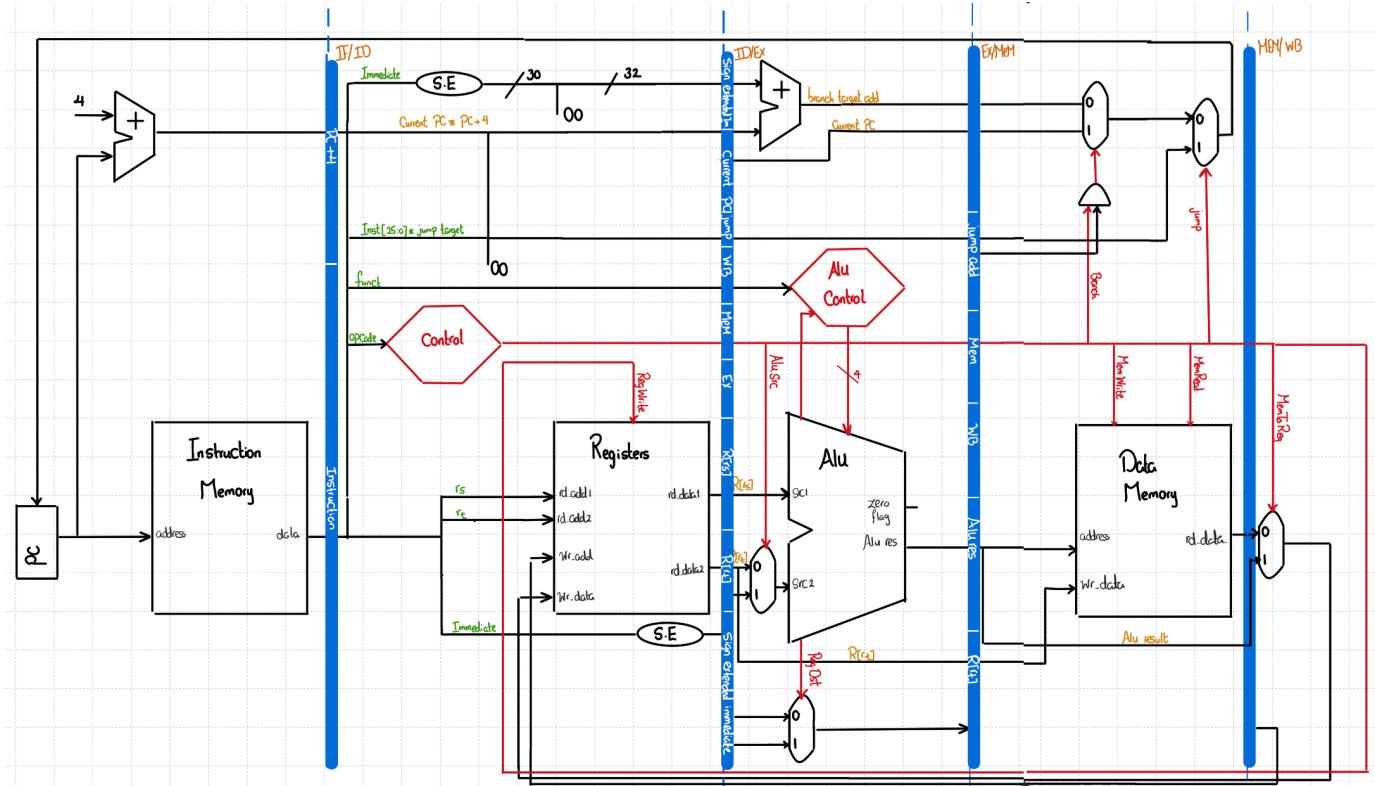


Figure 2: Pipeline Processor

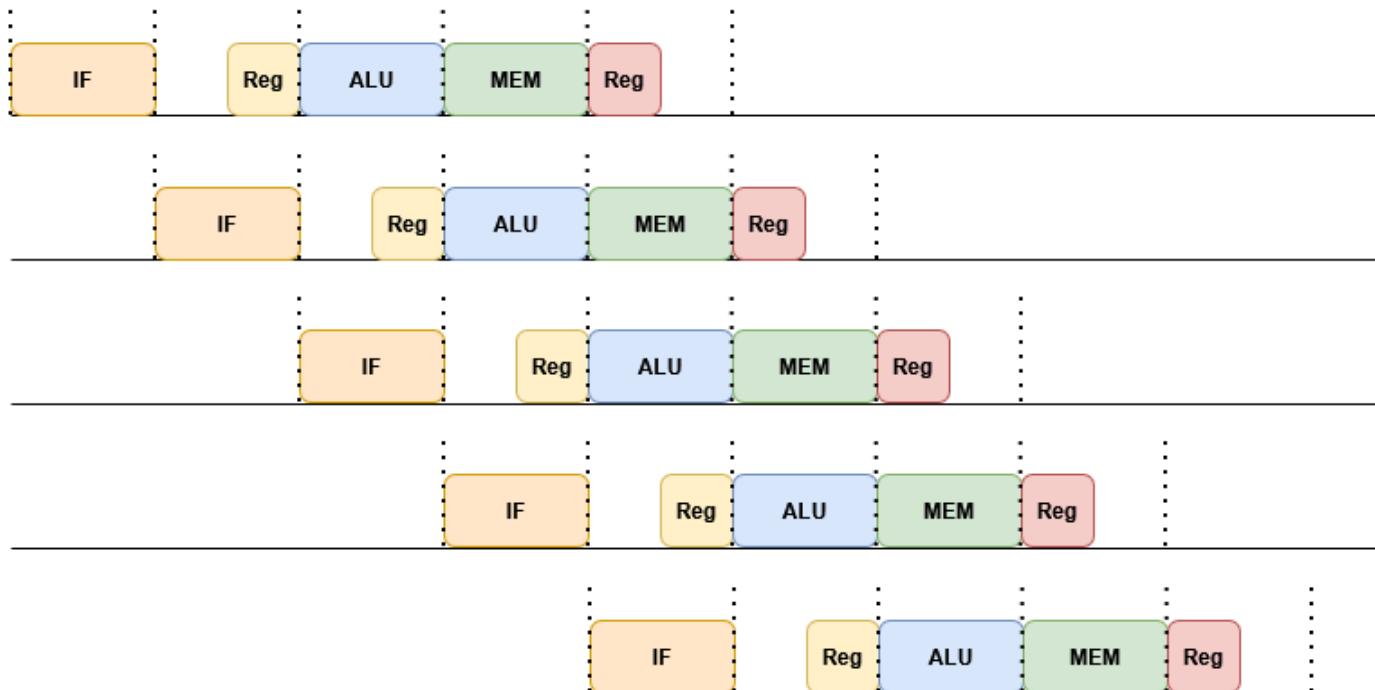


Figure 3: Pipeline Instructions execution

Cycle time, latency and throughput

Non pipelined processor

$$\text{cycle time} = T_{IF} + T_{ID} + T_{EX} + T_{MEM} + T_{WB}$$

latency = cycle time

$$\text{Throughput(IPS)} = 1/\text{CT}$$

Pipelined processor

$$\text{cycle time} = \max(T_{IF}, T_{ID}, T_{EX}, T_{MEM}, T_{WB}) + T_{\text{inter stage reg}}$$

latency = CT x no. of stages

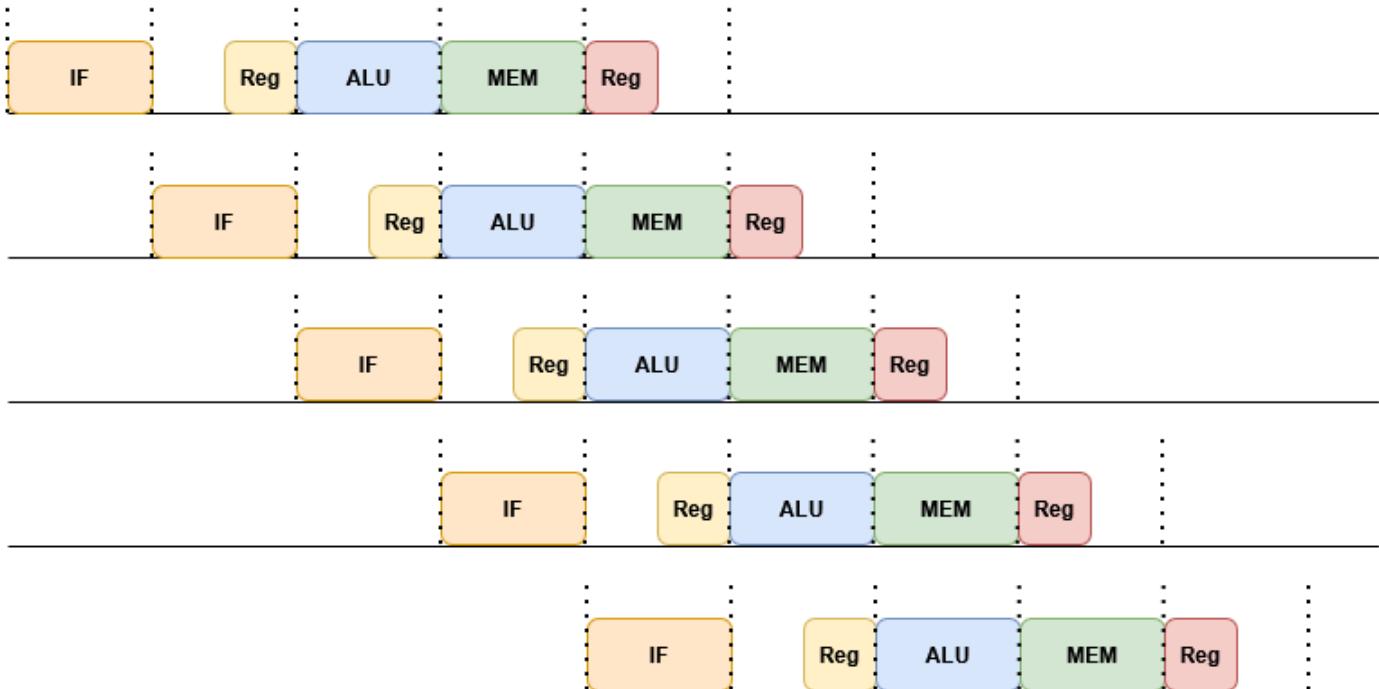
Throughput = IPS

[!NOTE] It is preferred to make stages balanced "time for each stage is almost the same" to make the pipeline efficient.

Pipelined hazards

[1] Structural Hazard resource contention, using the same resource by two different instructions at the same time

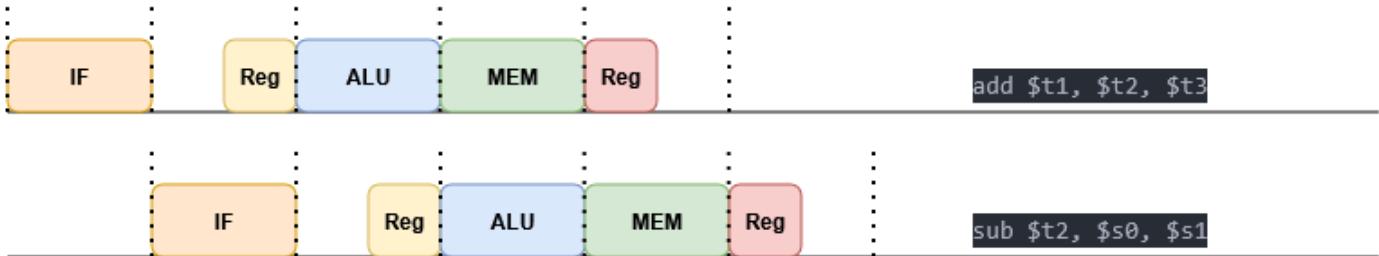
MIPS doesn't have such a problem, How it possible?



[2] Data hazards data dependencies between instructions.

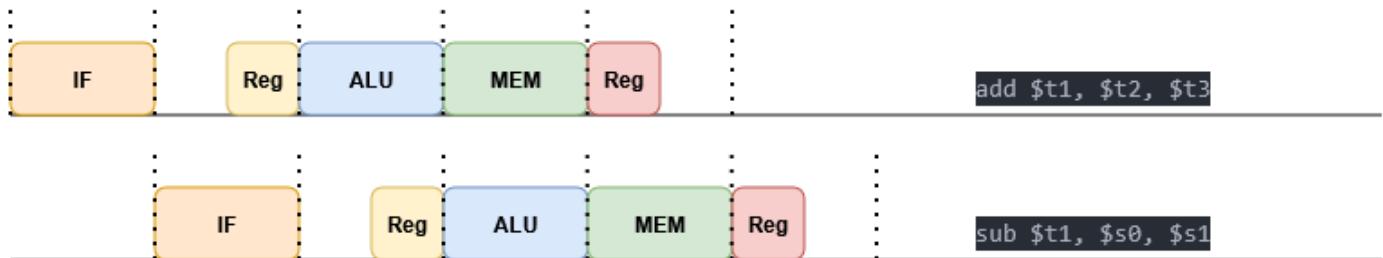
i. WAR: Write After Read

```
add $t1, $t2, $t3  
sub $t2, $s0, $s1
```



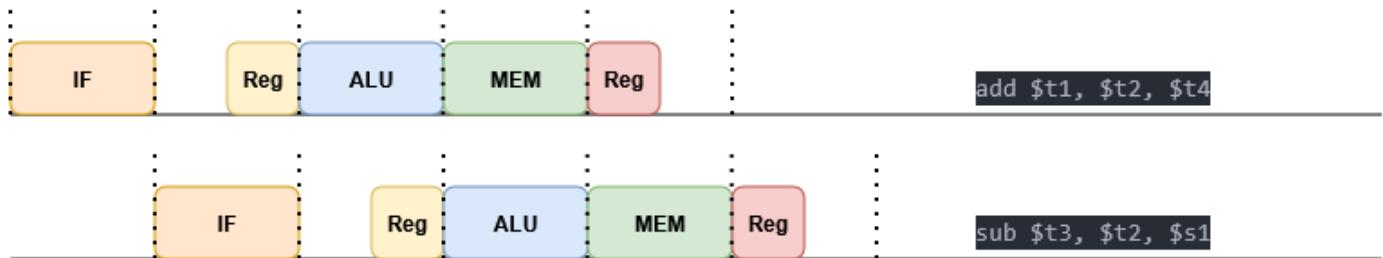
ii. WAW: Write After Write

```
add $t1, $t2, $t3
sub $t1, $s0, $s1
```



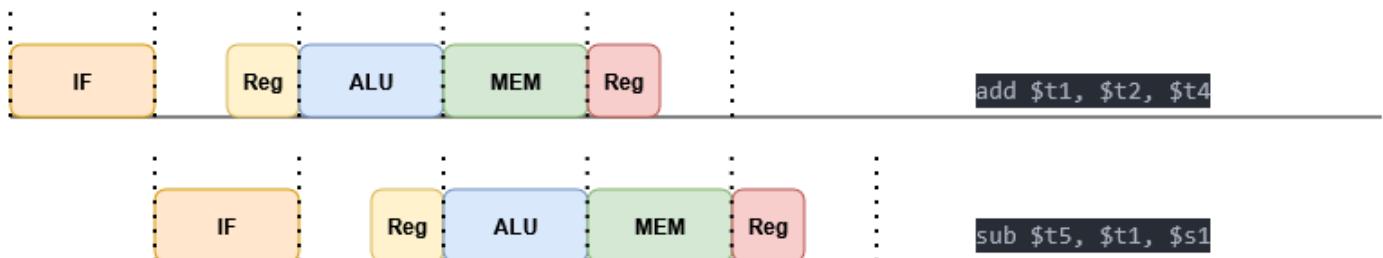
iii. RAR : Read After Read

```
add $t1, $t2, $t4
sub $t3, $t2, $s1
```



iv. RAW : Read After Write

```
add $t1, $t2, $t4
sub $t5, $t1, $s1
```

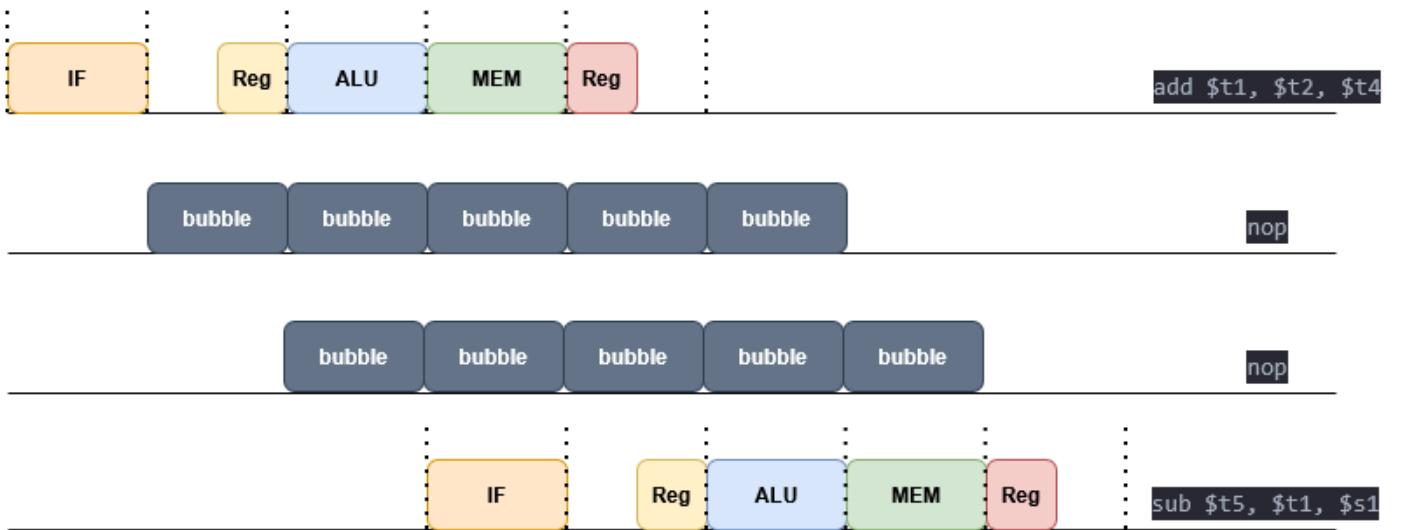


RAW is the only one that has a problem of data dependency in MIPS

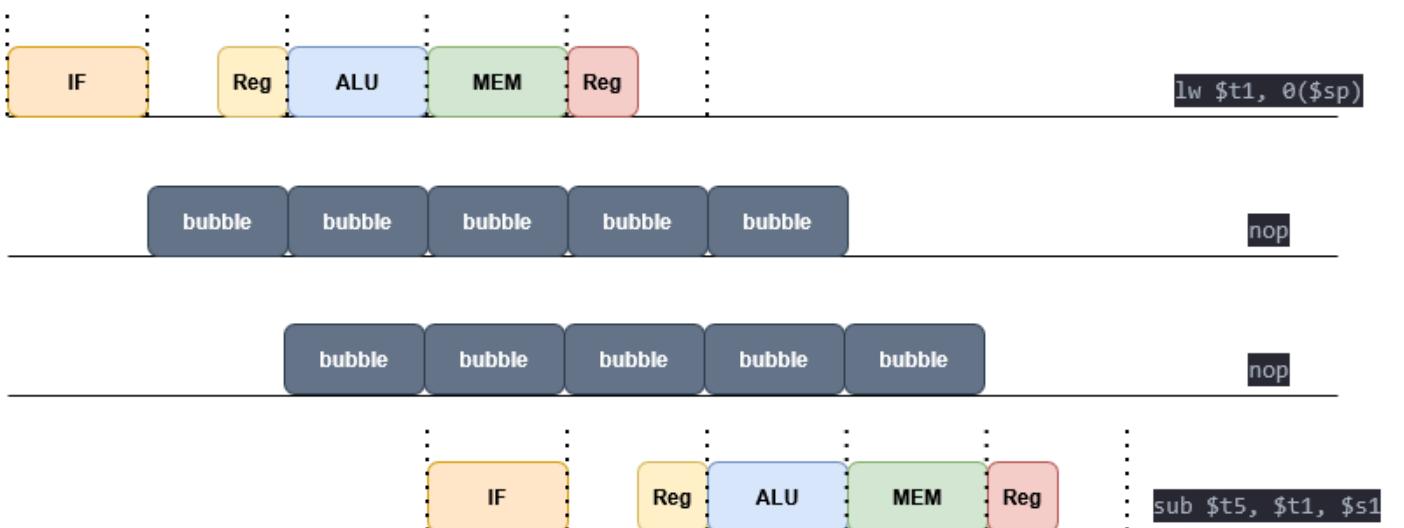
Solutions:

1. Add stall or nop instructions

Arithmetic instructions

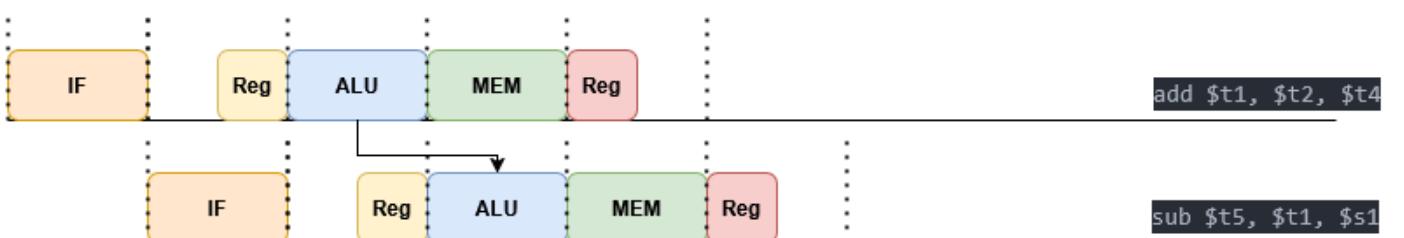


Load instruction

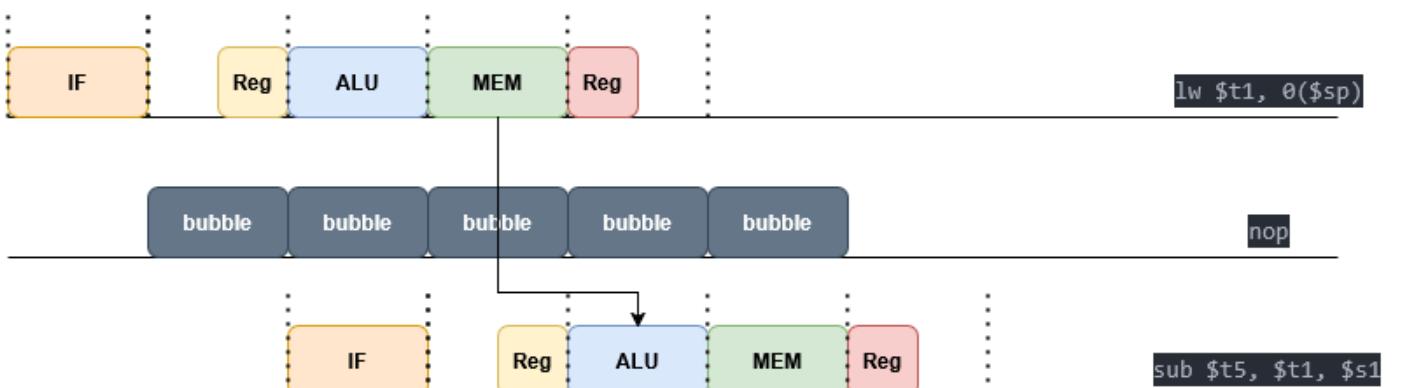


2. Forwarding

Arithmetic instructions



Load instruction



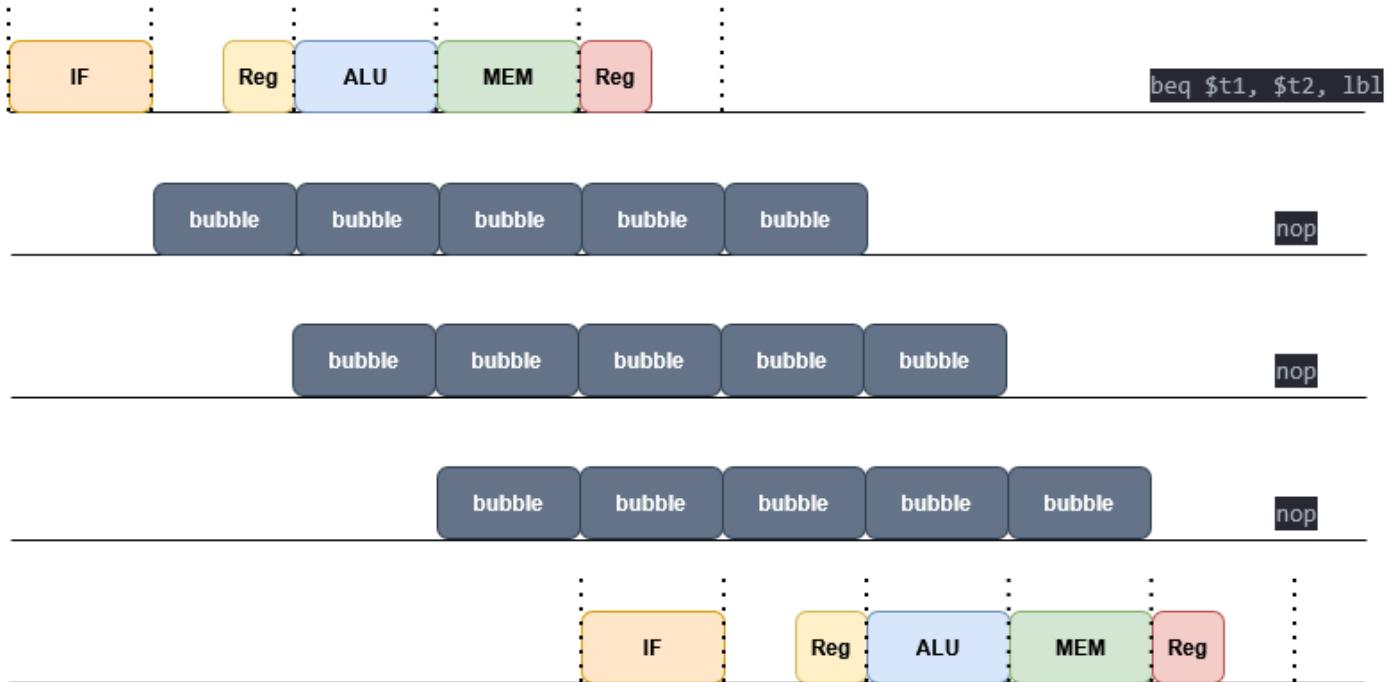
3. Code scheduling: a good compiler can add instructions in place of nop instruction that is have no dependnecies

[3] Control hazards Branching instructions

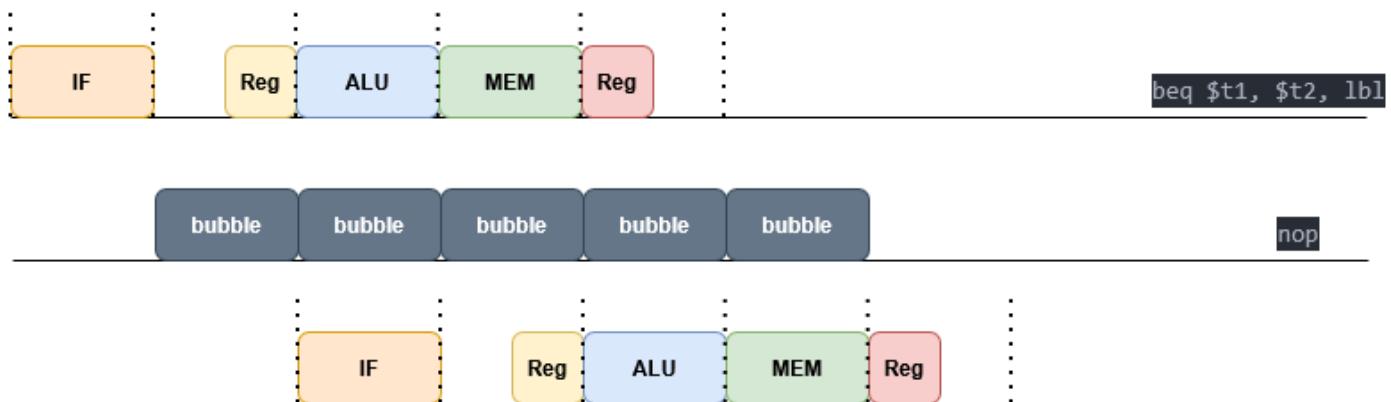
We are not sure that branch is taken or not until stage #4 "After Execute instruction and get the flag

Solutions:

1. Add 3 nops



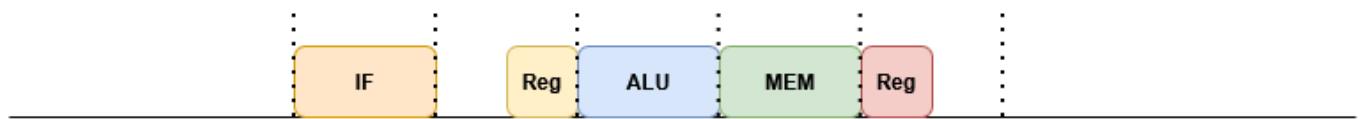
2. Move branch hardware to stage #2 & add one nop



3. Move branch hardware to stage #2 & use speculation "branch prediction"



incorrect prediction



correct prediction

4. Delayed branch "most commonly used method"



independent instruction

