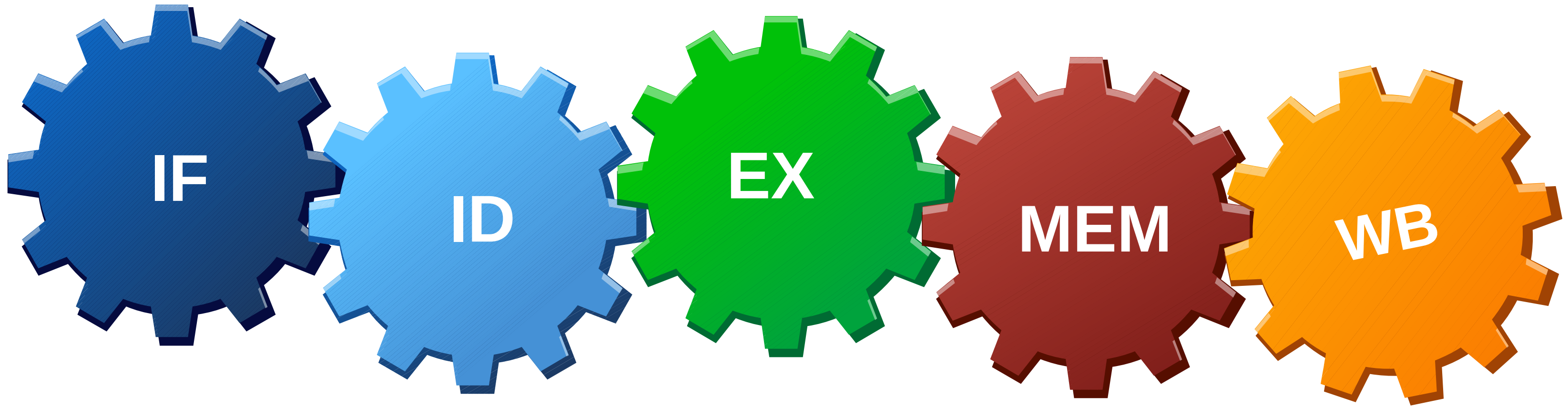
The Processor: Pipelined Architecture

Dr. Ahmed Zahran
WGB 182
a.zahran@cs.ucc.ie

MIPS Pipeline

- Five stages, one step per stage

1. **IF**: Instruction fetch from memory
2. **ID**: Instruction decode & register read
3. **EX**: Execute operation or calculate address
4. **MEM**: Access memory operand [not always]
5. **WB**: Write result back to register [not always]



Pipelining and MIPS ISA Design

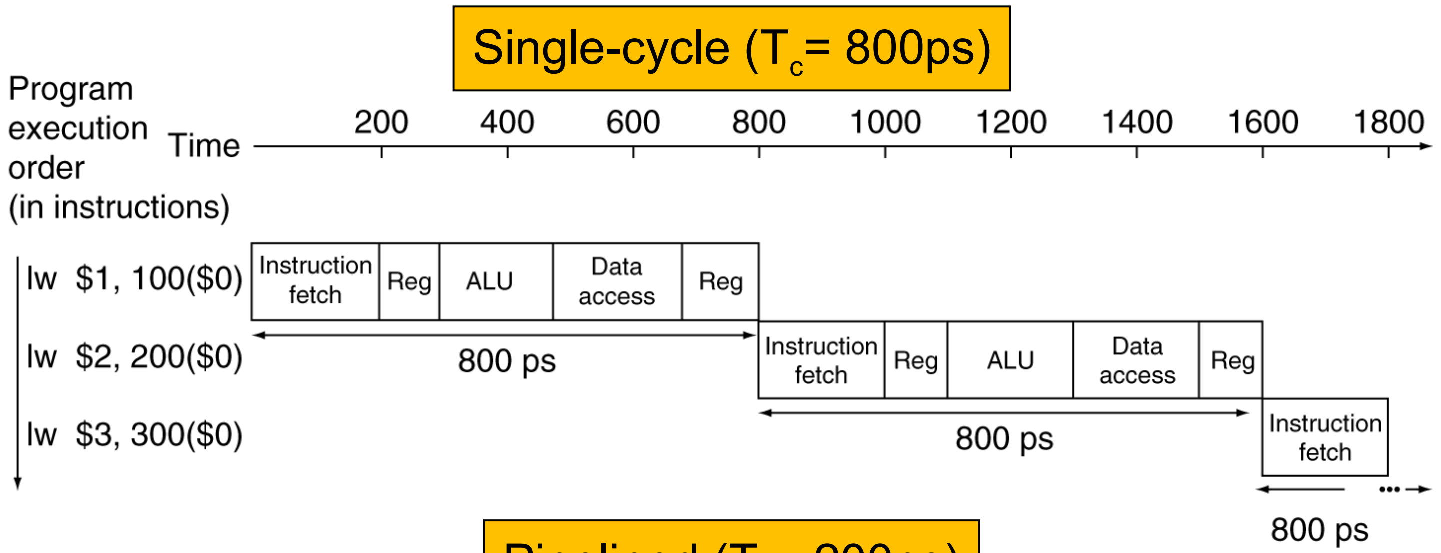
- MIPS ISA designed for pipelining
  - One instruction size [32-bits]
    - Easier to fetch in one cycle
    - c.f. x86: 1- to 17-byte instructions
  - Few and regular instruction formats
    - Can decode and read registers in one step
  - Simple Load/store addressing
    - Can calculate address in 3rd stage, access memory in 4th stage
  - Alignment of memory operands

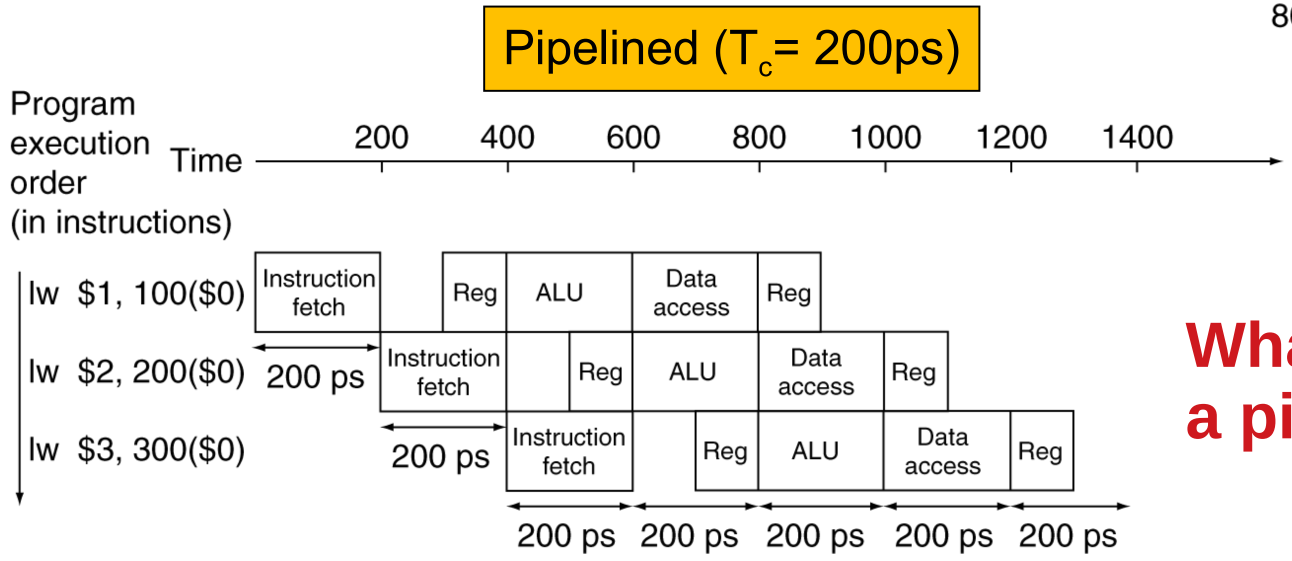
Pipeline Performance

- Assume time for stages is
  - 100ps for register read or write
  - 200ps for other stages
- Table below shows the clock-cycle time for **single-cycle datapath**

Instr	Instr fetch	Register read	ALU op	Memory access	Register write	Total time
lw	200ps	100 ps	200ps	200ps	100 ps	800ps
sw	200ps	100 ps	200ps	200ps		700ps
R-format	200ps	100 ps	200ps		100 ps	600ps
beq	200ps	100 ps	200ps			500ps

Pipeline Performance Comparison





What is the clock cycle of a pipelined processor?

Pipeline Speedup

- If all stages are balanced
  - i.e., all take the same time

$$Speedup = \frac{(Time\, between\, instructions)_{pipelined}}{(Time\, between\, instructions)_{single-cycle}}$$
$$= Number\, of\, Pipeline\, Stages$$

- If not balanced, speedup is less
- Speedup due to increased throughput
  - Latency (time for each instruction) does not decrease

So, Pipelining is Good, right?

What if?

- The processor needs to fetch an instruction from the memory at stage 1 and you need to execute a memory load instruction?
- The Processor executing this instruction sequence

  -
    ```
    add   $s0, $t0, $t1
    sub   $t2, $s0, $t3
    ```
- You are executing a branch instruction and the branch is taken?

Pipeline Hazards

- Situations that prevent starting the next instruction in the next cycle

1. Structure hazard

Two instructions need to access the same resource at the same time, e.g., memory

2. Data hazard

An instruction needs data that from a previous instruction in the pipeline, e.g., incomplete memory load

3. Control hazard

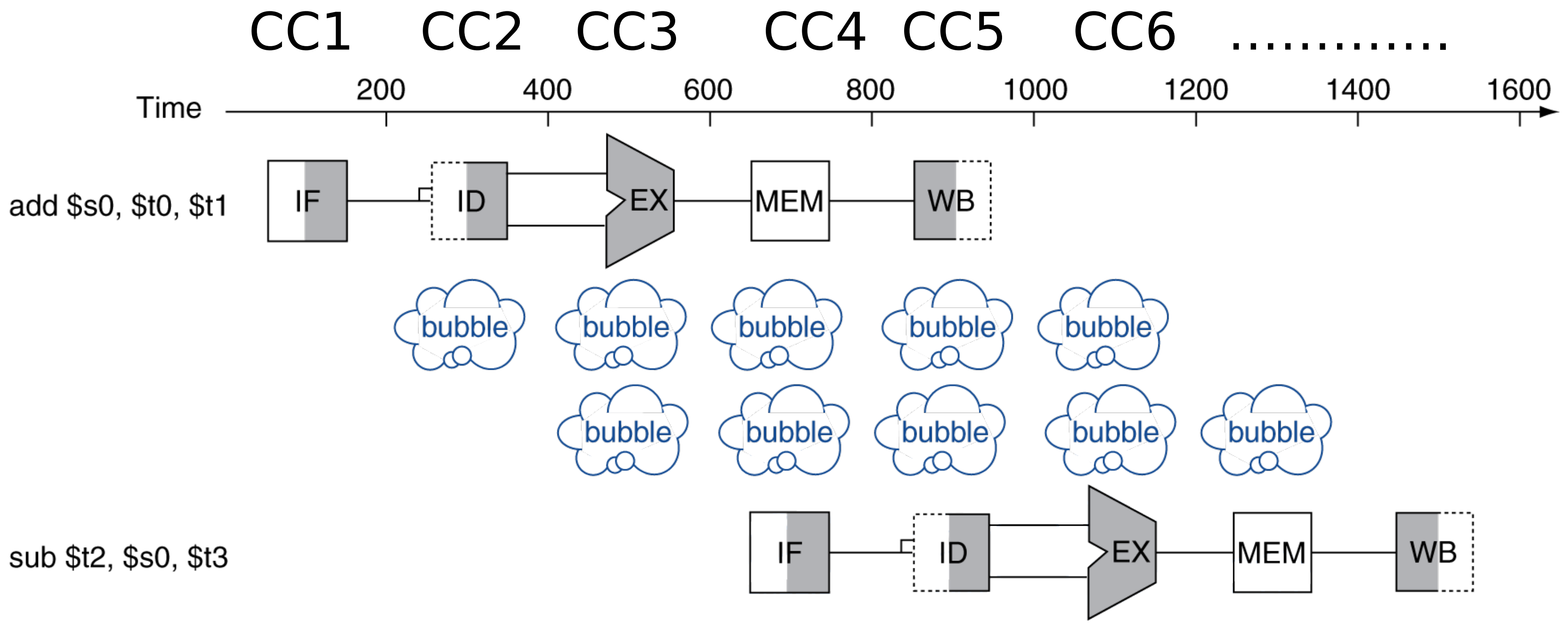
Changing instruction sequence (branch, call, ..) invalidates the pipeline fetch/decode

Structure Hazards

- Conflict for use of a resource
- In MIPS pipeline with a single memory
  - Load/store requires data access
  - Instruction fetch would have to *stall* for that cycle
    - Would cause a pipeline “bubble”
- Solution: separate hardware
  - pipelined datapaths require separate instruction/data memories

Data Hazards

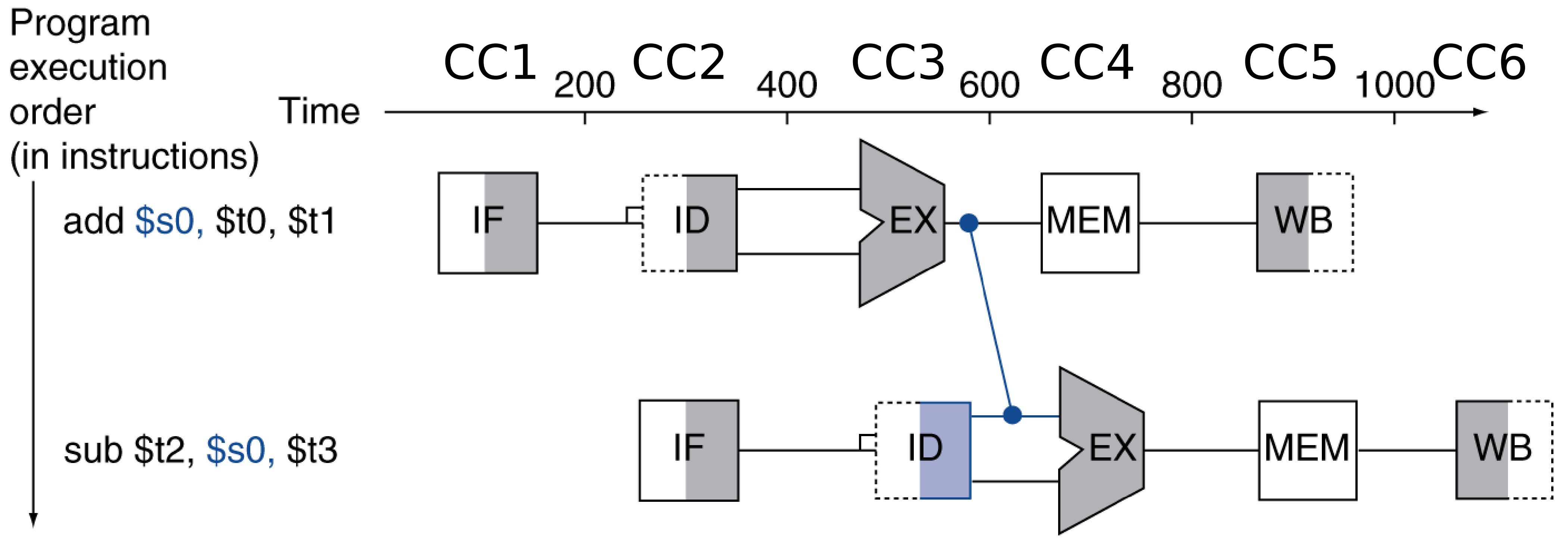
- An instruction depends on completion of data access by a previous instruction
  - add $s0, $t0, $t1
    sub $t2, $s0, $t3



Forwarding (aka Bypassing)

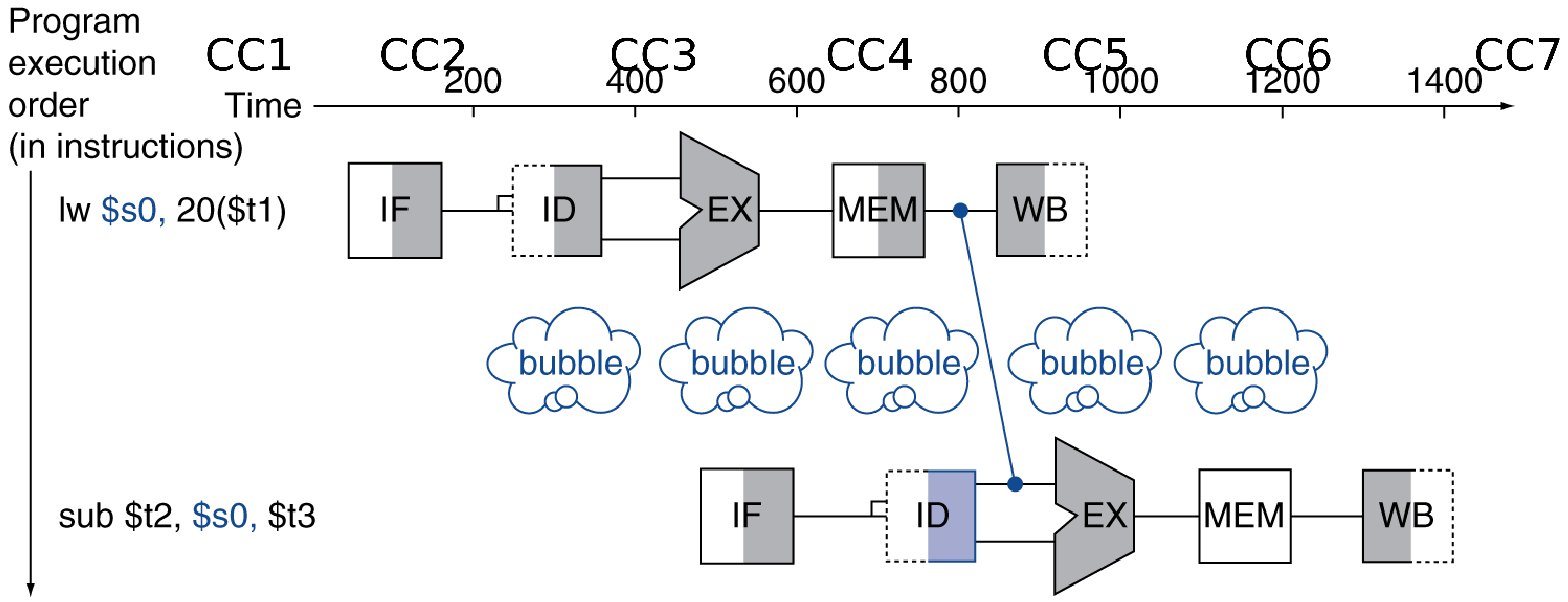
- Use result when it is computed
  - Don’t wait for it to be stored in a register
  - Requires ***extra connections*** in the datapath + **additional control [later]**

Hardware Solution



Load-Use Data Hazard

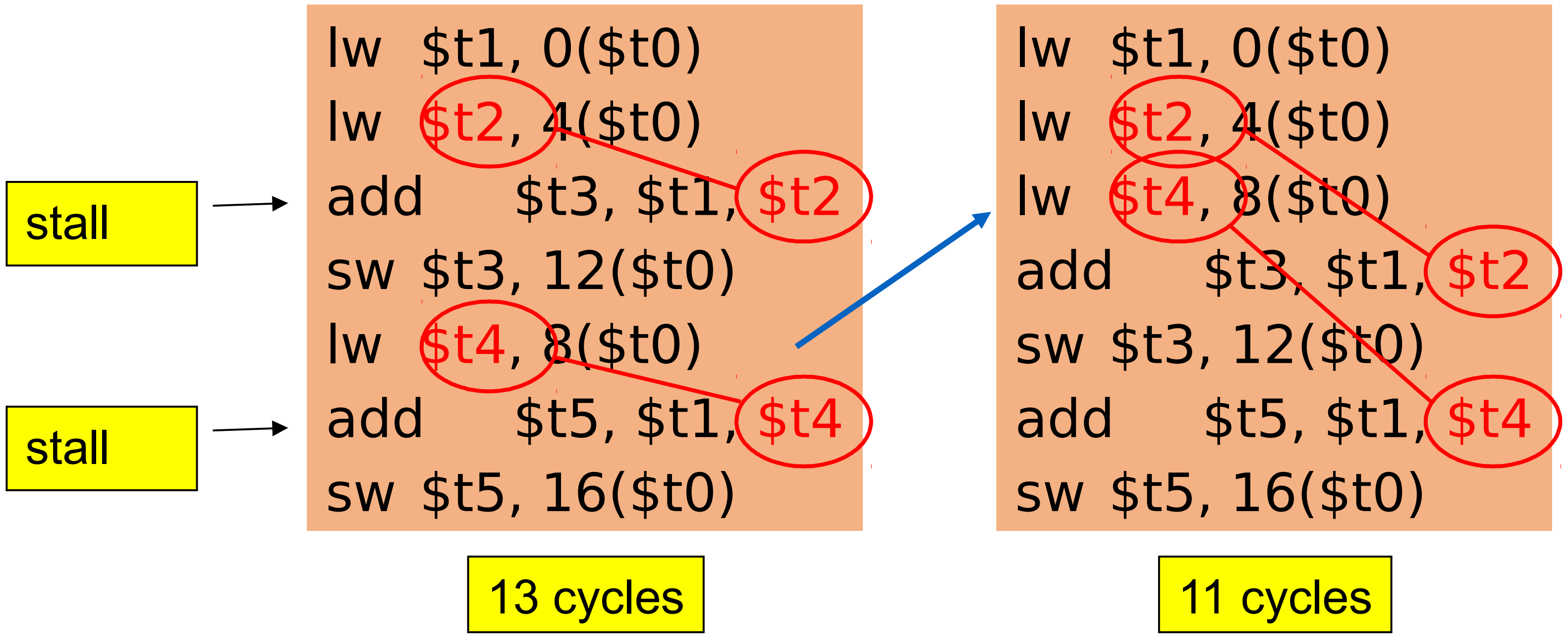
- Can’t always avoid bubble/stalls by forwarding
  - If value not computed when needed
  - Can’t forward backward in time!



Code Scheduling to Avoid Stalls

Software Solution

- Reorder code to avoid use of load result in the next instruction
- C code for A = B + E; C = B + F;



```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add     $t3, $t1, $t2
sw  $t3, 12($t0)
lw   $t4, 8($t0)
add     $t5, $t1, $t4
sw  $t5, 16($t0)
```

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
lw   $t4, 8($t0)
add     $t3, $t1, $t2
sw  $t3, 12($t0)
add     $t5, $t1, $t4
sw  $t5, 16($t0)
```

Discussion

- Explain two techniques to reduce the impact of data hazards in MIPS pipeline.
  - Forwarding (hardware)
  - Instruction scheduling (software)
- Explain what is meant by structure hazard. Provide an example of structure hazard.

Reading

- Section 4.5