An Overview of Pipelining

DR. RAJIB RANJAN MAITI

CSIS, BITS-PILANI, HYDERABAD

Four types of Data Hazards

Read After Write (RAW)

R-Type

```
ADD R1, R2, R3
SUB R4, R1, R6
```

Mem-Access

```
store R1, 0($t0)
load R4, 0($t0)
```

Write After Read (WAR)

```
add r4, r1,
add r1, r3, r5
```

```
add R1, R2, R3
sub R2, R4, R1
or R1, R6, R3
```

Write After Write (WAW)

```
add R1,R2,R3
sub R2,R4,R1
or R1,R6,R3
```

```
add r1, r2, r3
add r4, r1, r5
add r1, r3, r5
```

Read After Read (RAR)

```
add R1, R2, R3
sub R2, R4, R1
or R1, R6, R3
```

For MIPS integer pipeline, all data hazards can be checked during ID phase

Data Hazard

- Antidependence:
  - A dependence between two instructions using a same register. . .
  - . . . 1st instruction reads
  - . . . 2nd instruction writes

Write After Read (WAR)

```
add r1, r2, r3
sub r2, r4, r5
```

Name Dependency

- Output Dependence:
  - A dependence between two instructions where . . .
  - . . . both write at the same location . . .
  - . . . Both write at the same register

Write After Write (WAW)

```
add R1,R2,R3
sub R2,R4,R1
or R1,R6,R3
```

Control Hazards

Caused by delay between the fetching of instructions and decisions about changes in control flow

add \$4, \$5, \$6	IF	ID	EX	MEM	WB					
beq \$1, \$2, ELSE		IF	ID	EX	MEM	WB				
or \$7, \$8, \$9			---	---	IF	ID	EX	MEM	WB	
ELSE: and \$7, \$8, \$9						IF	ID	EX	MEM	WB

Assume enough extra hardware in 2nd stage of the pipeline for :

1. test registers,
2. calculate the branch address, and
3. update the PC

Control Hazards

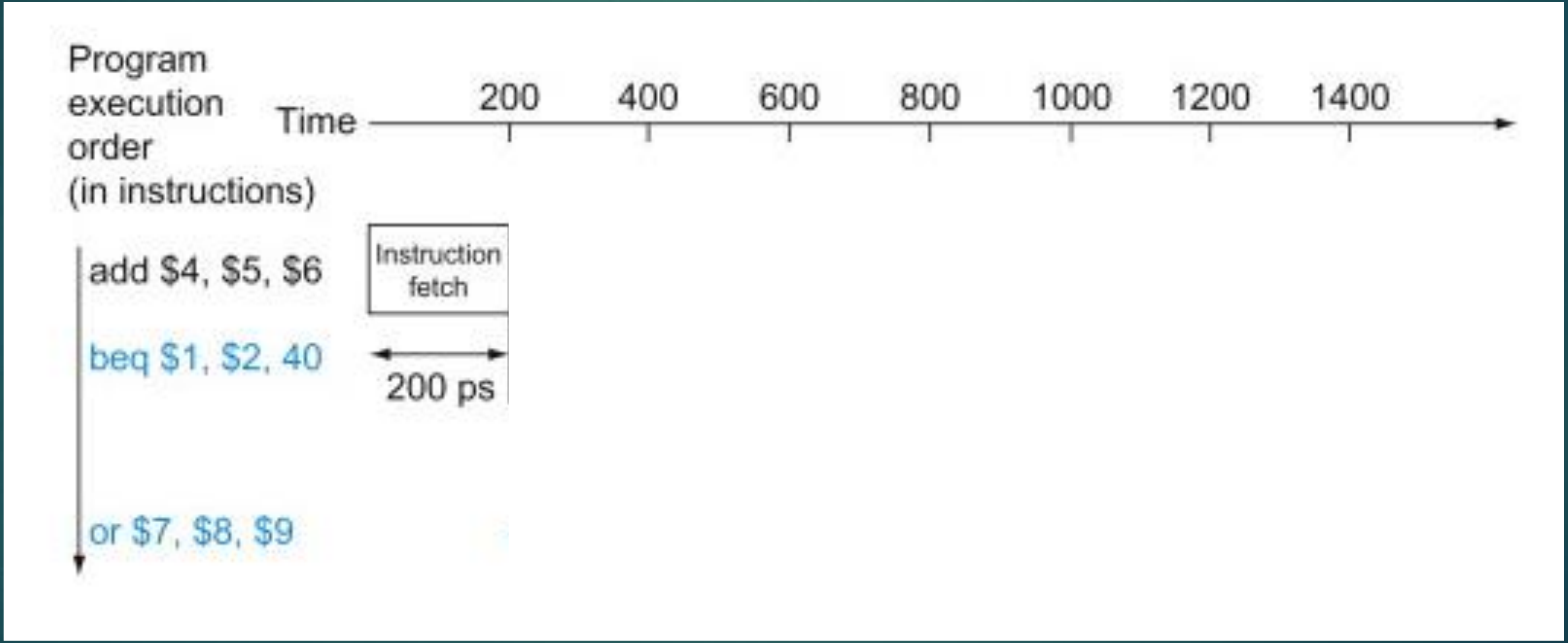
Caused by delay between the fetching of instructions and decisions about changes in control flow

add $4, $5, $6	IF	ID	EX	MEM	WB		
beq $1, $2, ELSE		IF	ID	EX	MEM	WB	
or $7, $8, $9			---	IF	ID	EX	MEM
ELSE: and $7, $8, $9					IF	ID	EX

stall immediately after we fetch a branch, waiting until the pipeline determines the outcome of the branch and knows what instruction address to fetch from

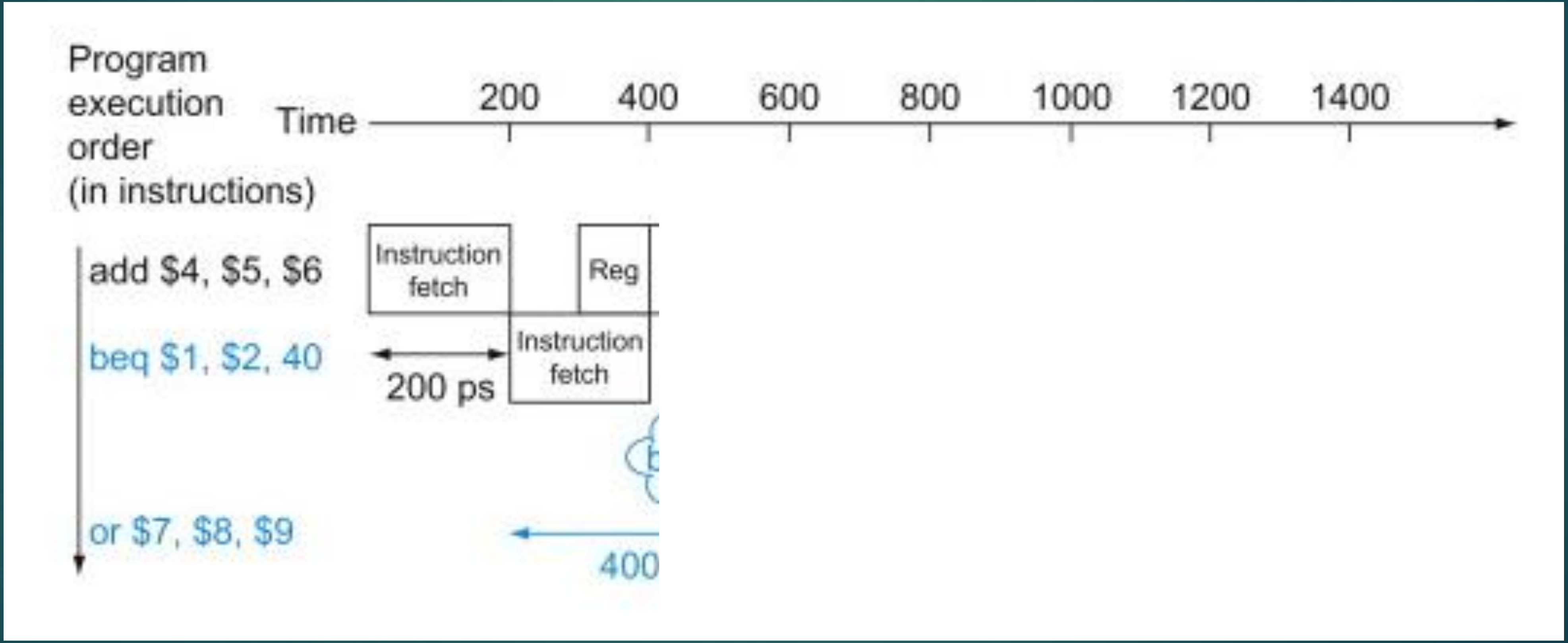
Control Hazards

- One solutions to control hazards
  - *Stall:* certainly works, but it is slow



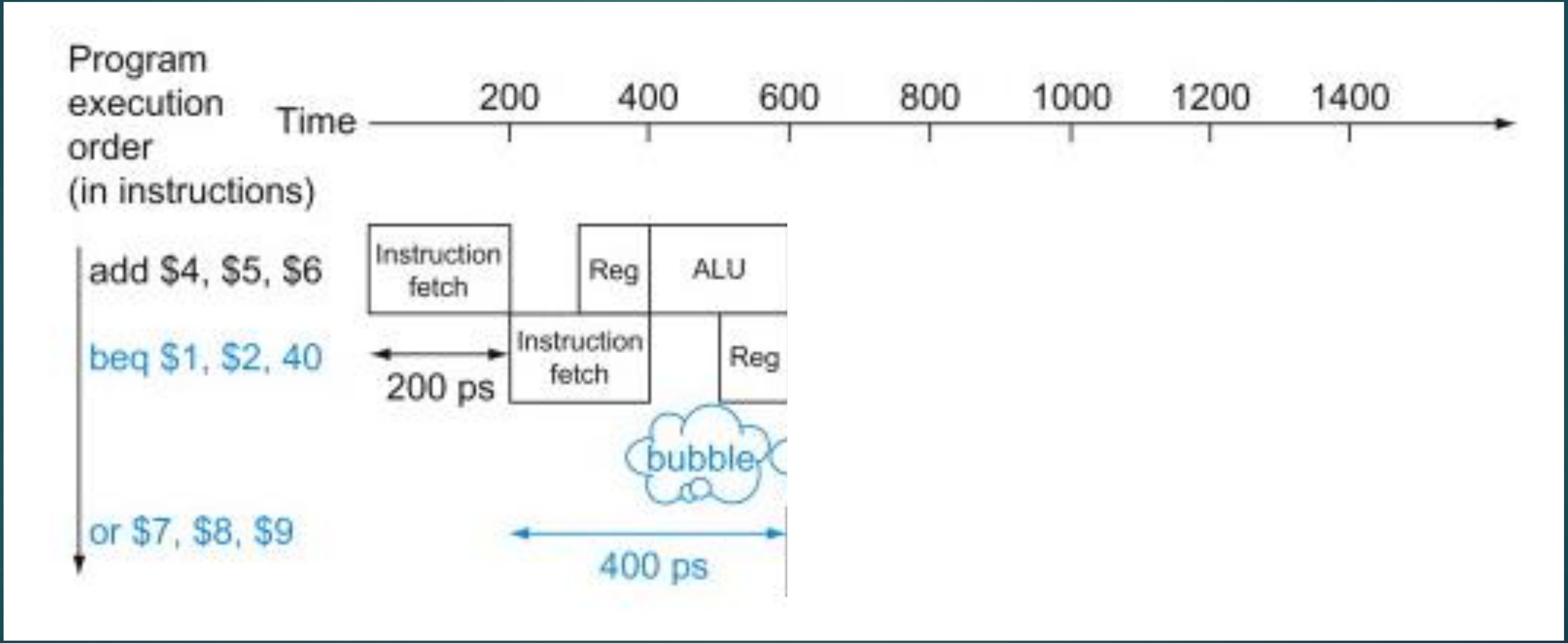
Control Hazards

- One solutions to control hazards
  - *Stall:* certainly works, but it is slow



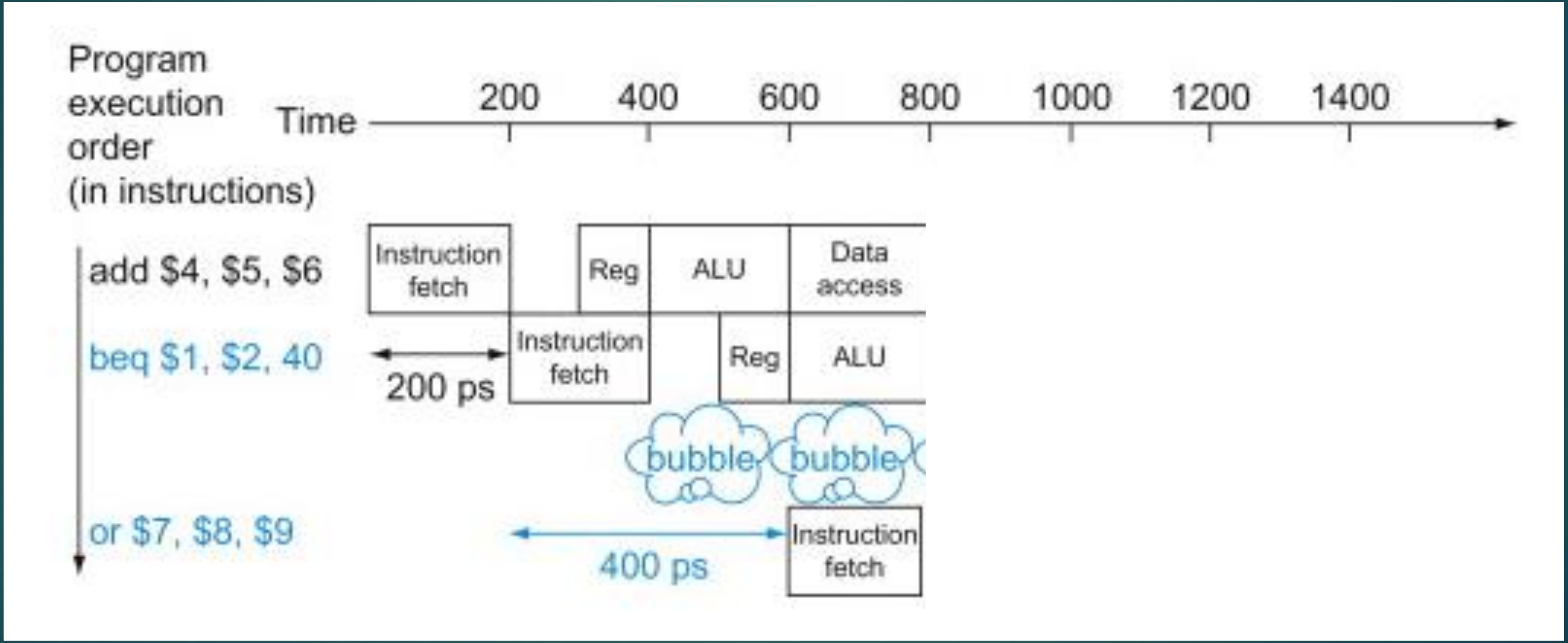
Control Hazards

- One solutions to control hazards
  - *Stall:* certainly works, but it is slow



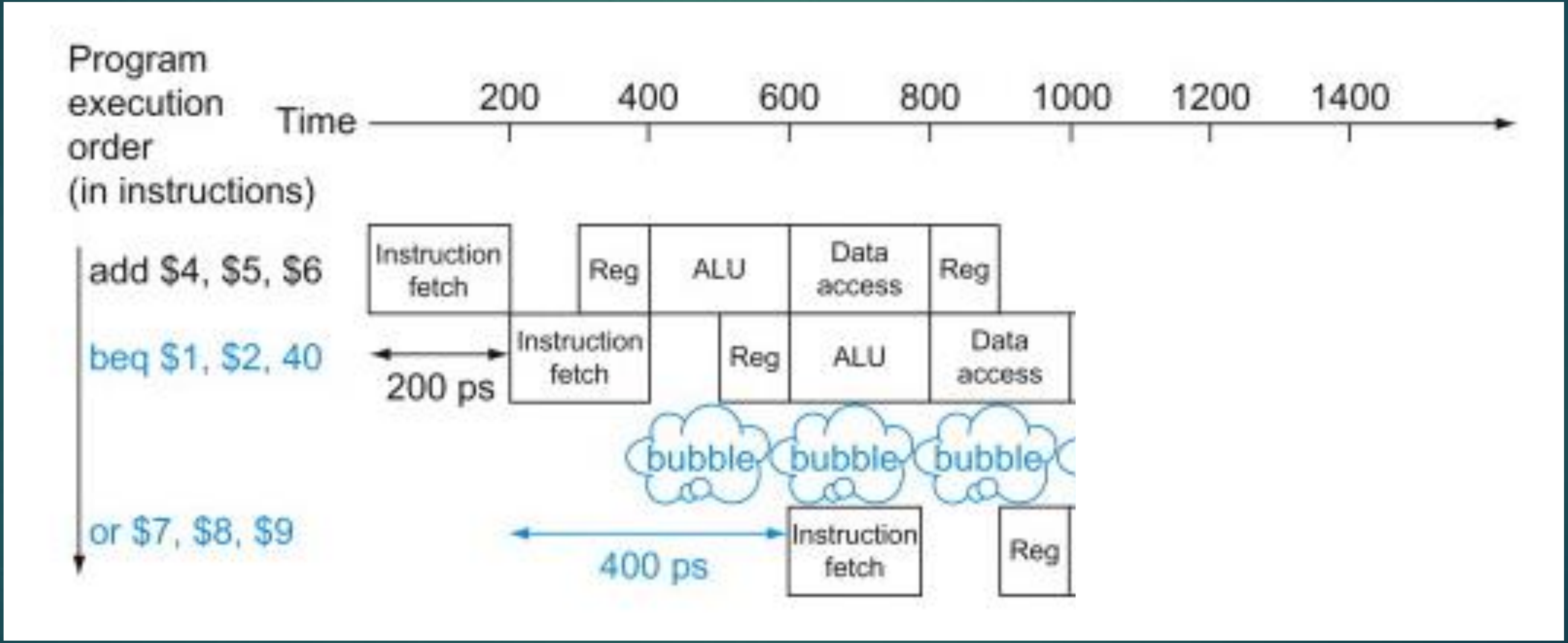
Control Hazards

- One solutions to control hazards
  - *Stall:* certainly works, but it is slow



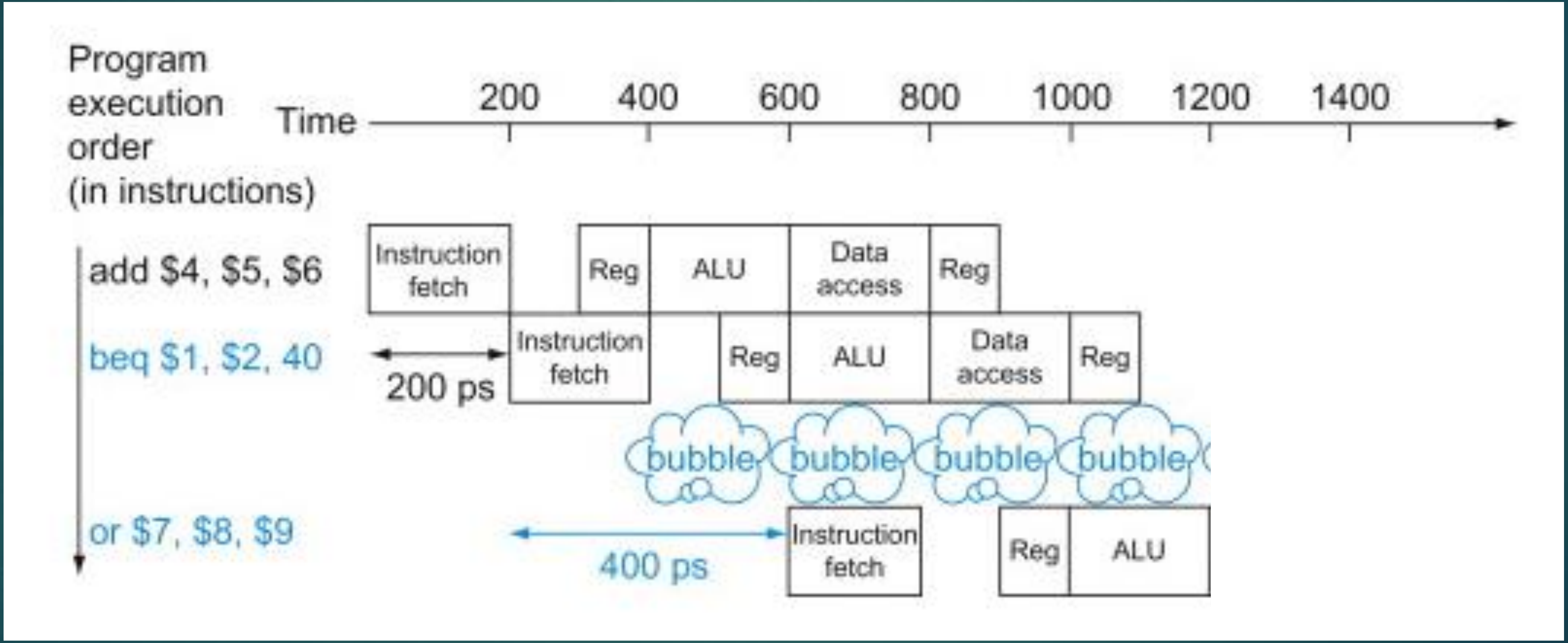
Control Hazards

- One solutions to control hazards
  - *Stall:* certainly works, but it is slow



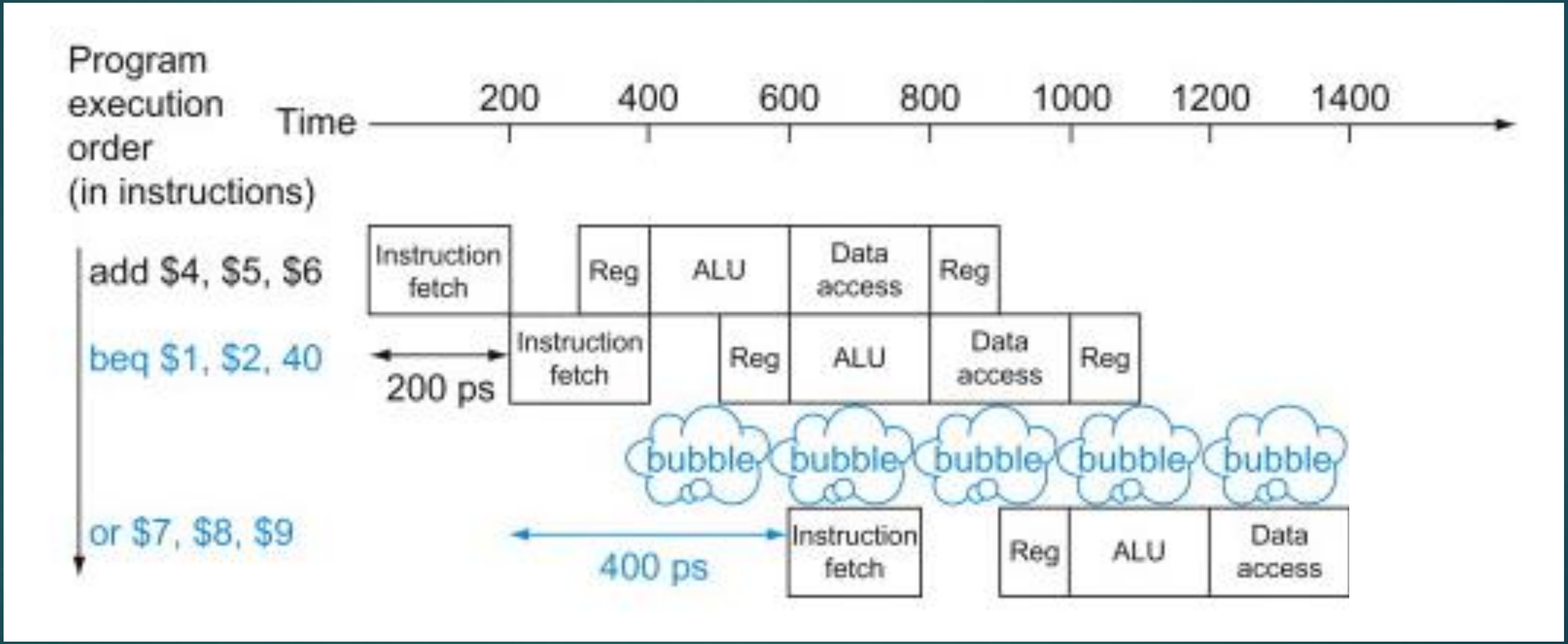
Control Hazards

- One solutions to control hazards
  - *Stall:* certainly works, but it is slow



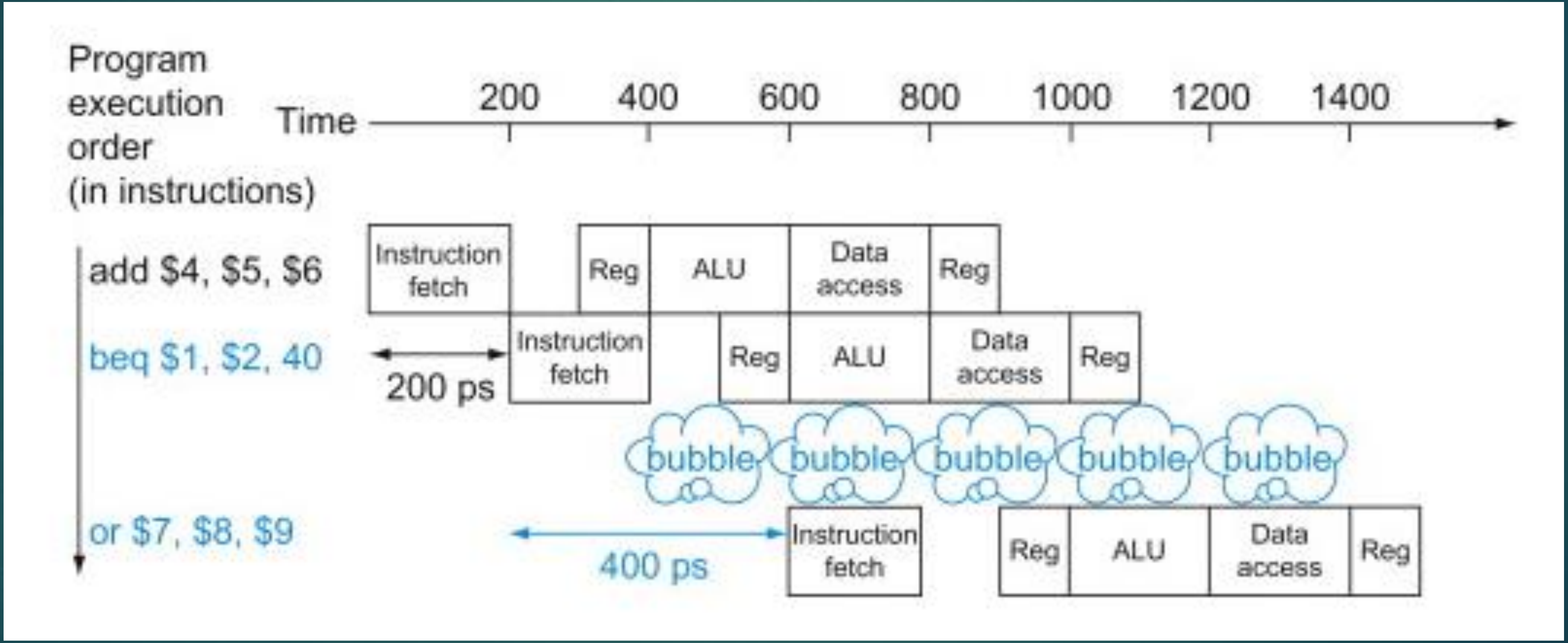
Control Hazards

- One solutions to control hazards
  - *Stall:* certainly works, but it is slow



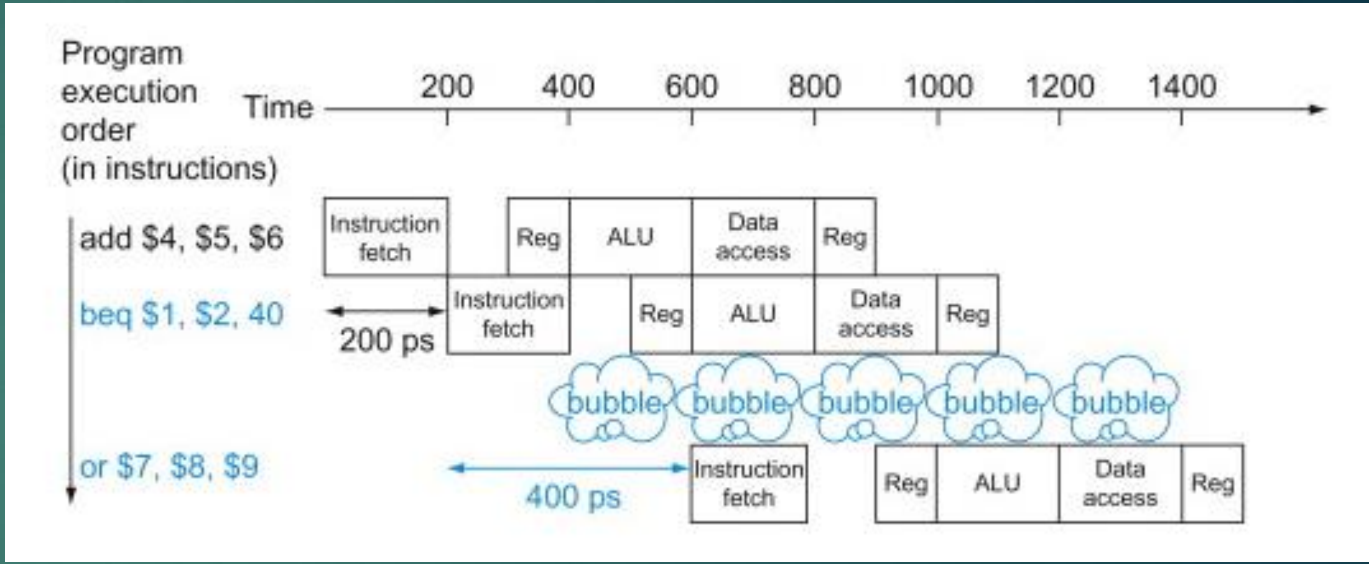
Control Hazards

- One solutions to control hazards
  - *Stall:* certainly works, but it is slow



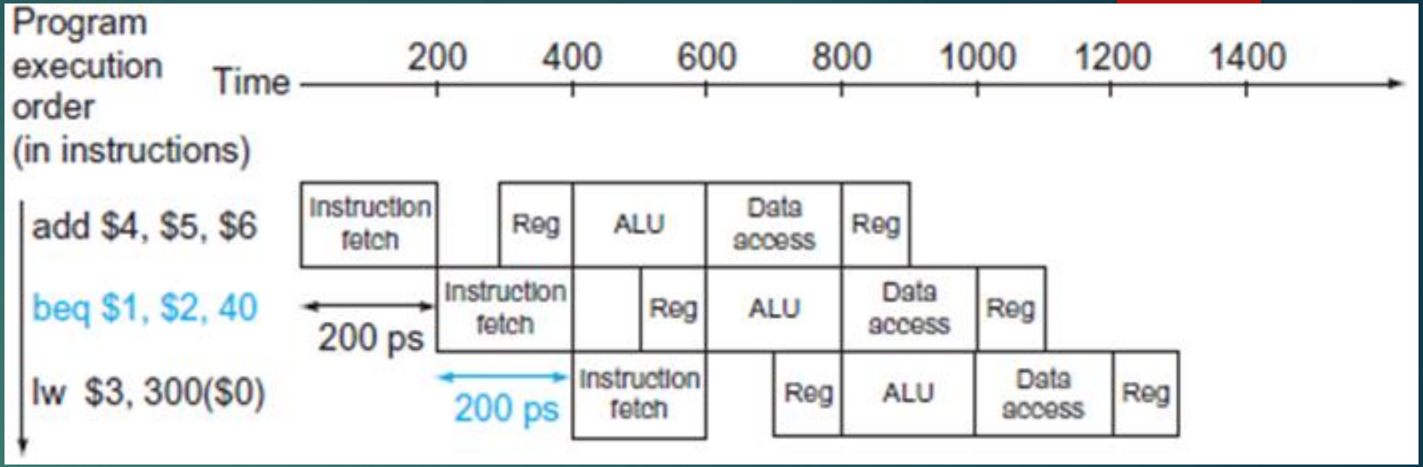
Control Hazards

- One solutions to control hazards
  - *Stall:* certainly works, but it is slow
- **Performance of “Stall on Branch”**
  - Assume all other instructions have a CPI of 1
  - Branches are 17% of the instructions executed in SPECint2006
  - Since branches took one extra clock cycle for the stall,
    - It will take a CPI of 1.17 and hence a slowdown of 1.17 versus the ideal case



Control Hazards

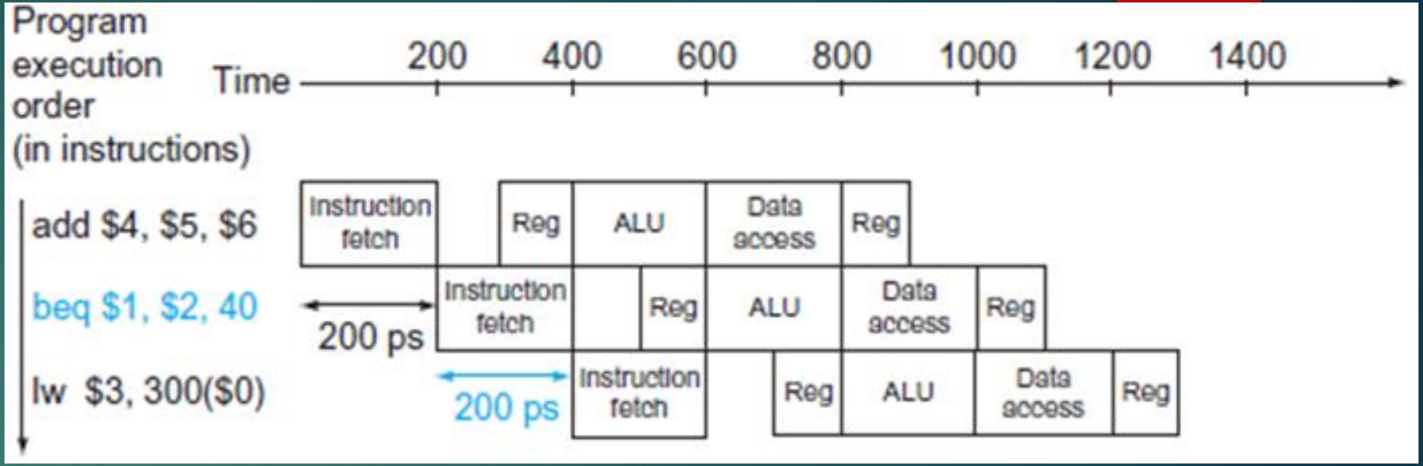
- Strategy: Predict
- When the guess is wrong,
  - the pipeline control must ensure that the instructions following the wrongly guessed branch have no effect
  - And, must restart the pipeline from the proper branch address



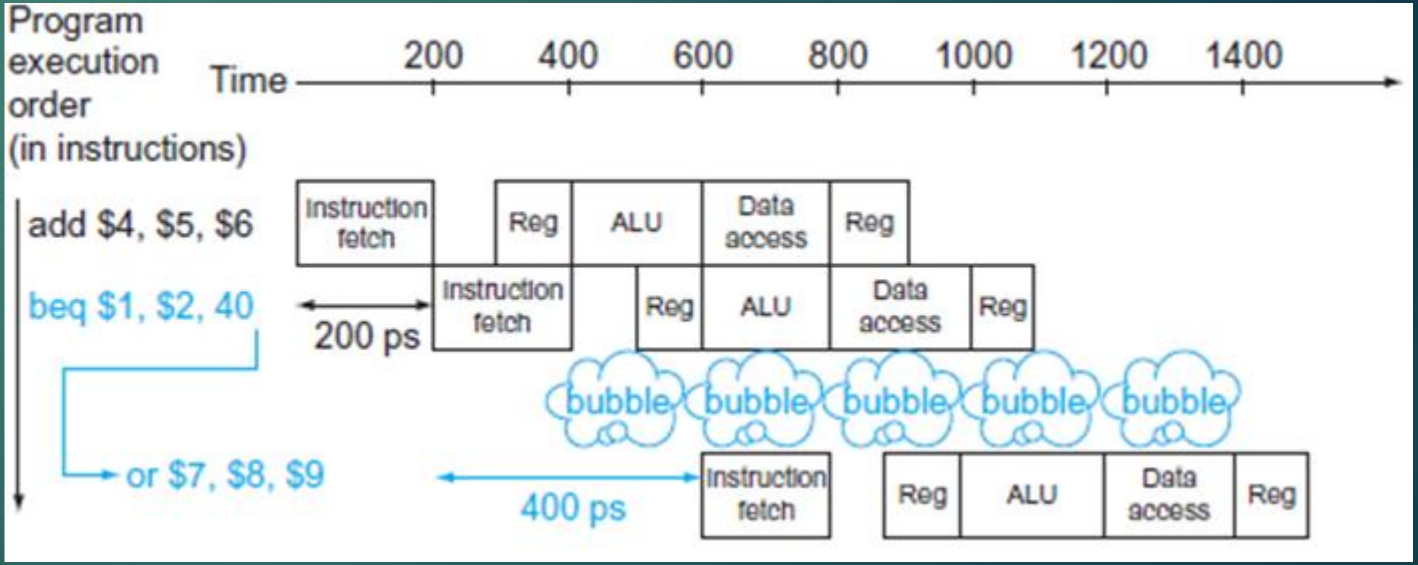
Predicting that branches are not taken as a solution to control hazard.

Control Hazards

- Strategy: Predict
- When the guess is wrong,
  - the pipeline control must ensure that the instructions following the wrongly guessed branch have no effect
  - And, must restart the pipeline from the proper branch address



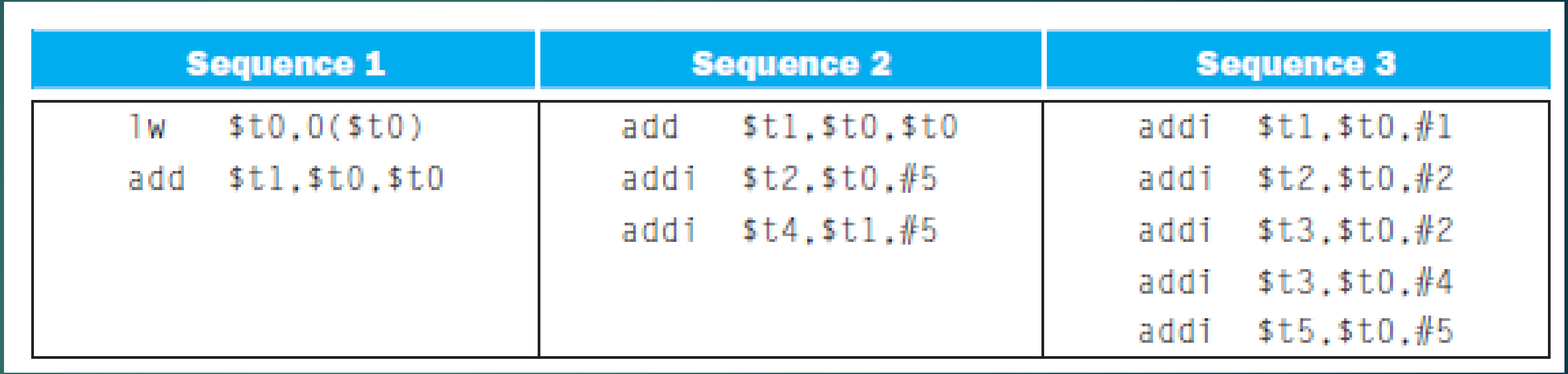
Predicting that branches are not taken as a solution to control hazard.



Prediction was wong, branch be actually taken

Pipeline Overview Summary

- For each code sequence below,
  - state whether it must stall,
  - can avoid stalls using only forwarding, or
  - can execute without stalling or forwarding



Sequence 1	Sequence 2	Sequence 3
`lw $t0,0($t0)`	`add $t1,$t0,$t0`	`addi $t1,$t0,#1`
`add $t1,$t0,$t0`	`addi $t2,$t0,#5`	`addi $t2,$t0,#2`
	`addi $t4,$t1,#5`	`addi $t3,$t0,#2`
		`addi $t3,$t0,#4`
		`addi $t5,$t0,#5`

Pipelining increases the number of simultaneously executing instructions and the rate at which instructions are started and completed. Pipelining does not reduce the time it takes to complete an individual instruction, also called the **latency**.

For example, the five-stage pipeline still takes 5 lock cycles for the instruction to complete. In the terms used in Chapter 1, pipelining improves instruction *throughput* rather than individual instruction *execution time* or *latency*.

Hazard Types:

- Structural Hazard:
  - Needed resource currently busy.
- Data Hazard:
  - Needed value not yet available or overwritten.
- Control Hazard:
  - Needed instruction not yet available or wrong instruction executing.

Thank You