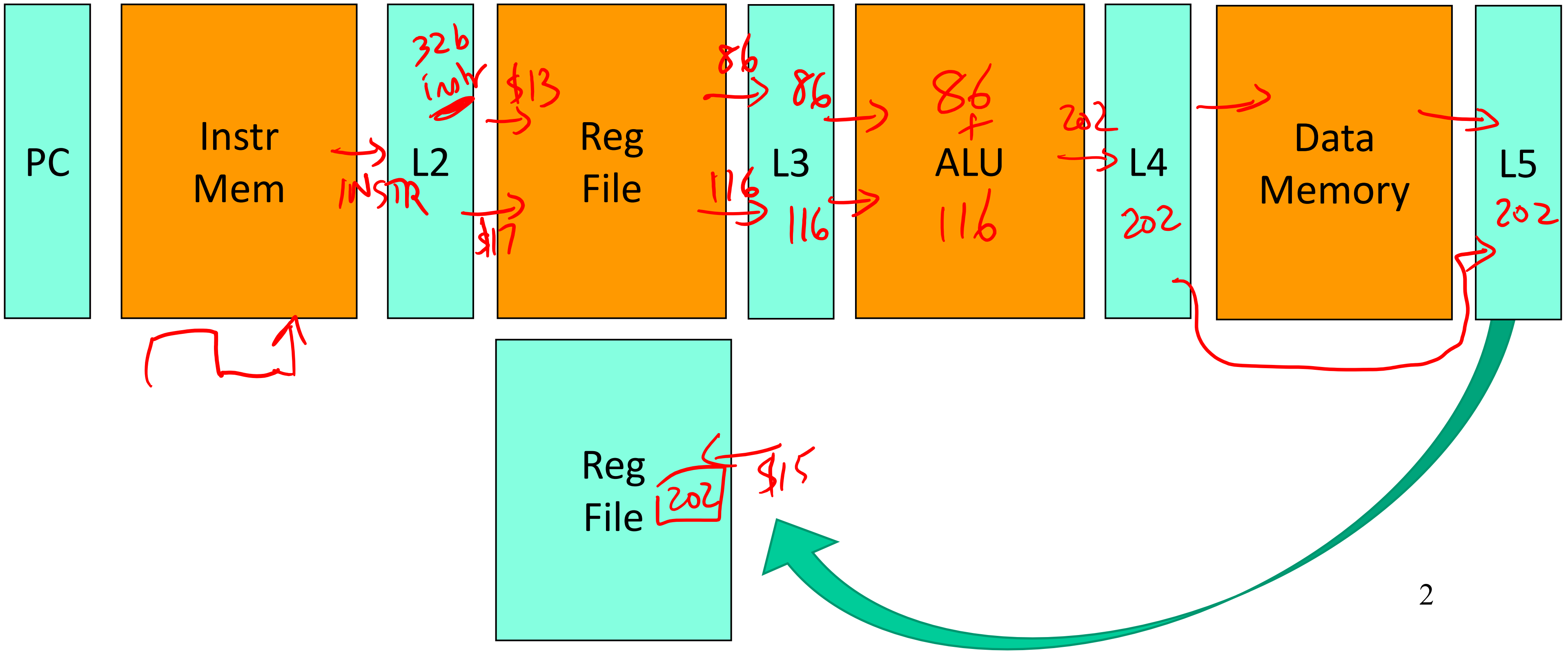
Lecture 17: Basic Pipelining

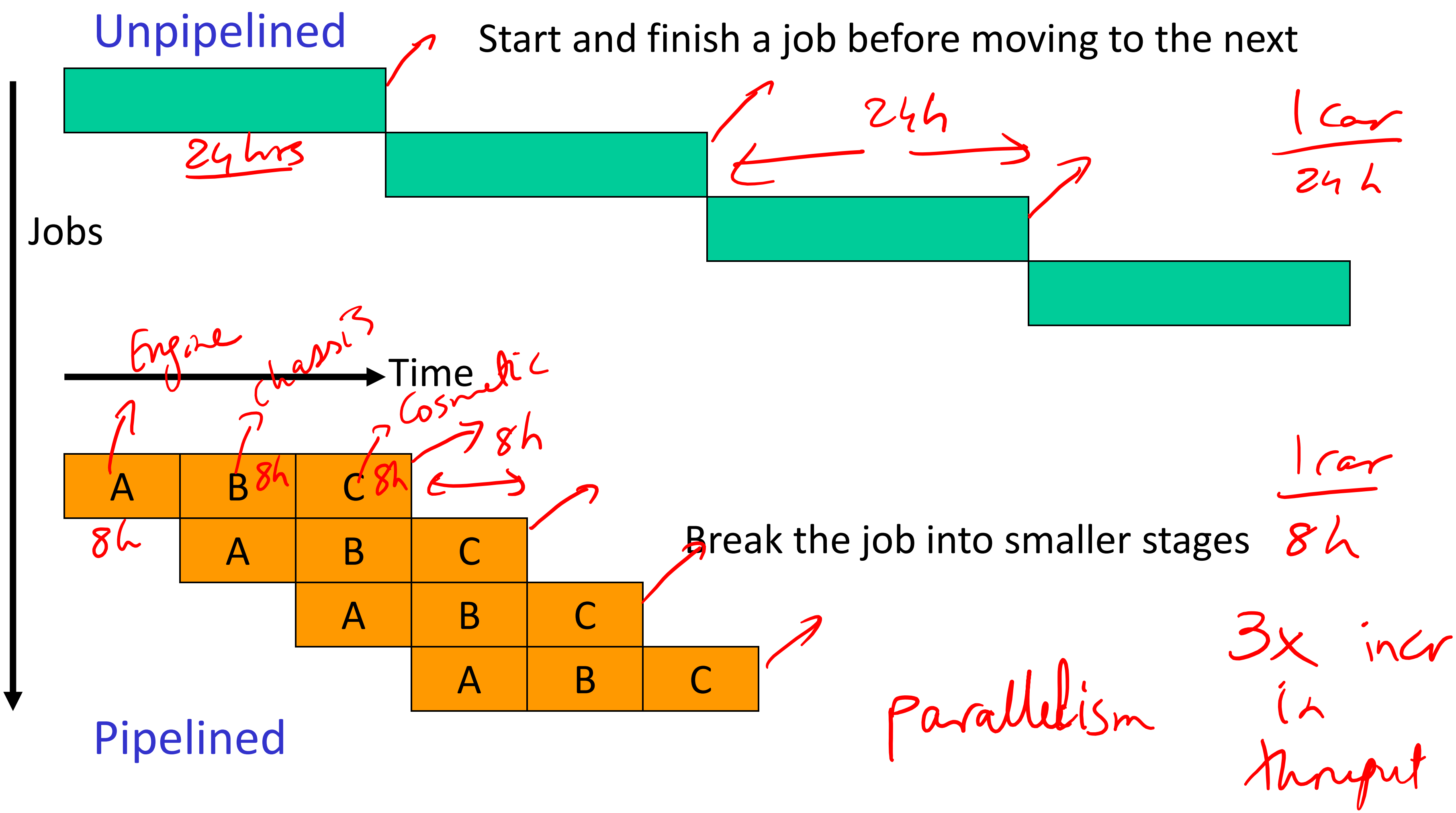
- Today's topics:
  - 5-stage pipeline
  - Hazards

Multi-Stage Circuit

- Instead of executing the entire instruction in a single cycle (a single stage), let's break up the execution into multiple stages, each separated by a latch



The Assembly Line



Performance Improvements?

- Does it take longer to finish each individual job?

  Under ideal conditions, NO
  If latch overhead considered, Yes

- Does it take shorter to finish a series of jobs?

  YES

- What assumptions were made while answering these questions?

  Ideal condition – no latch ovhd, no dependences

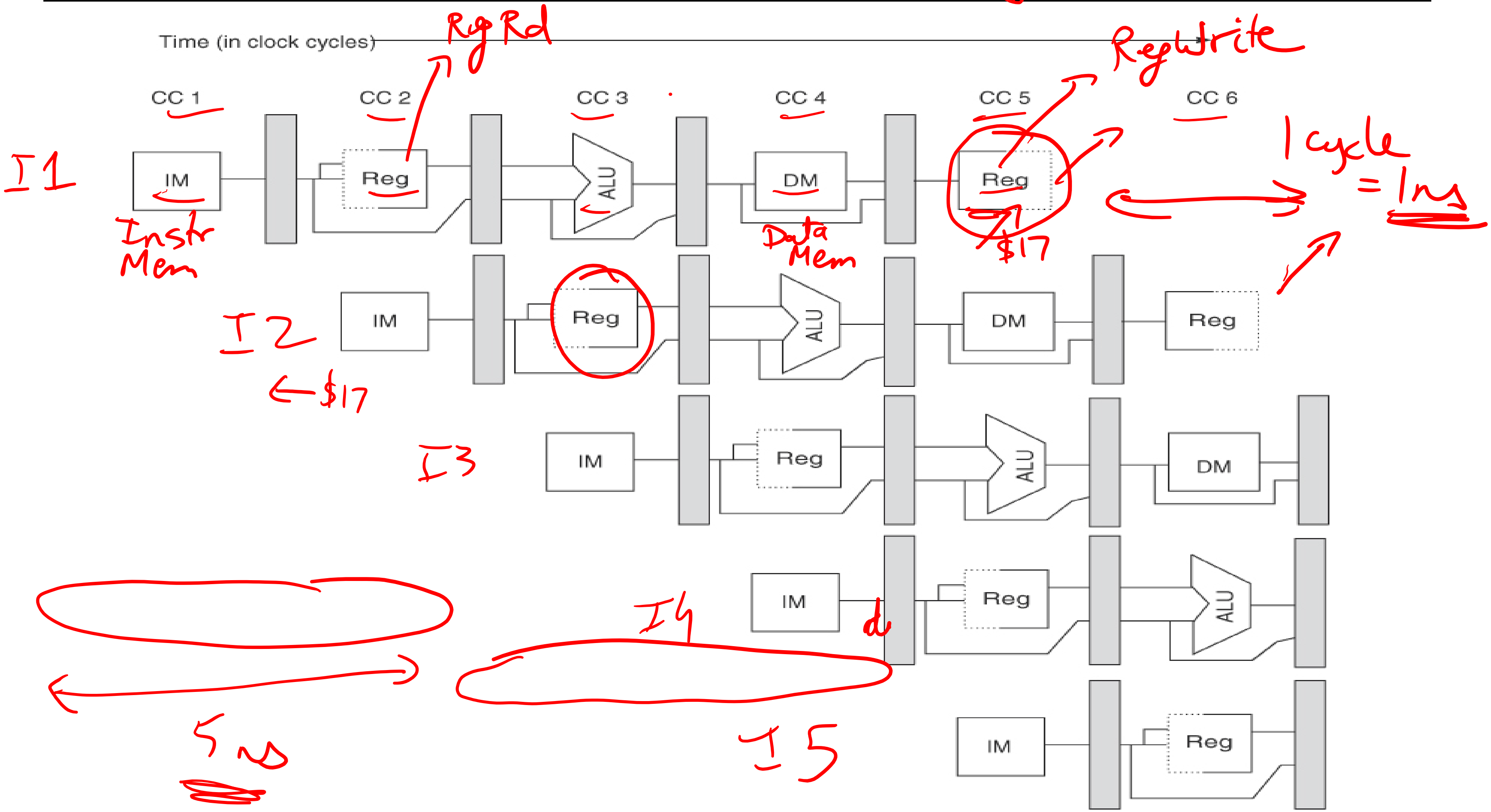
- Is a 10-stage pipeline better than a 5-stage pipeline?

  RAW

  producer I1 $17 ← ...

  consumer I2 ← $17 + ...

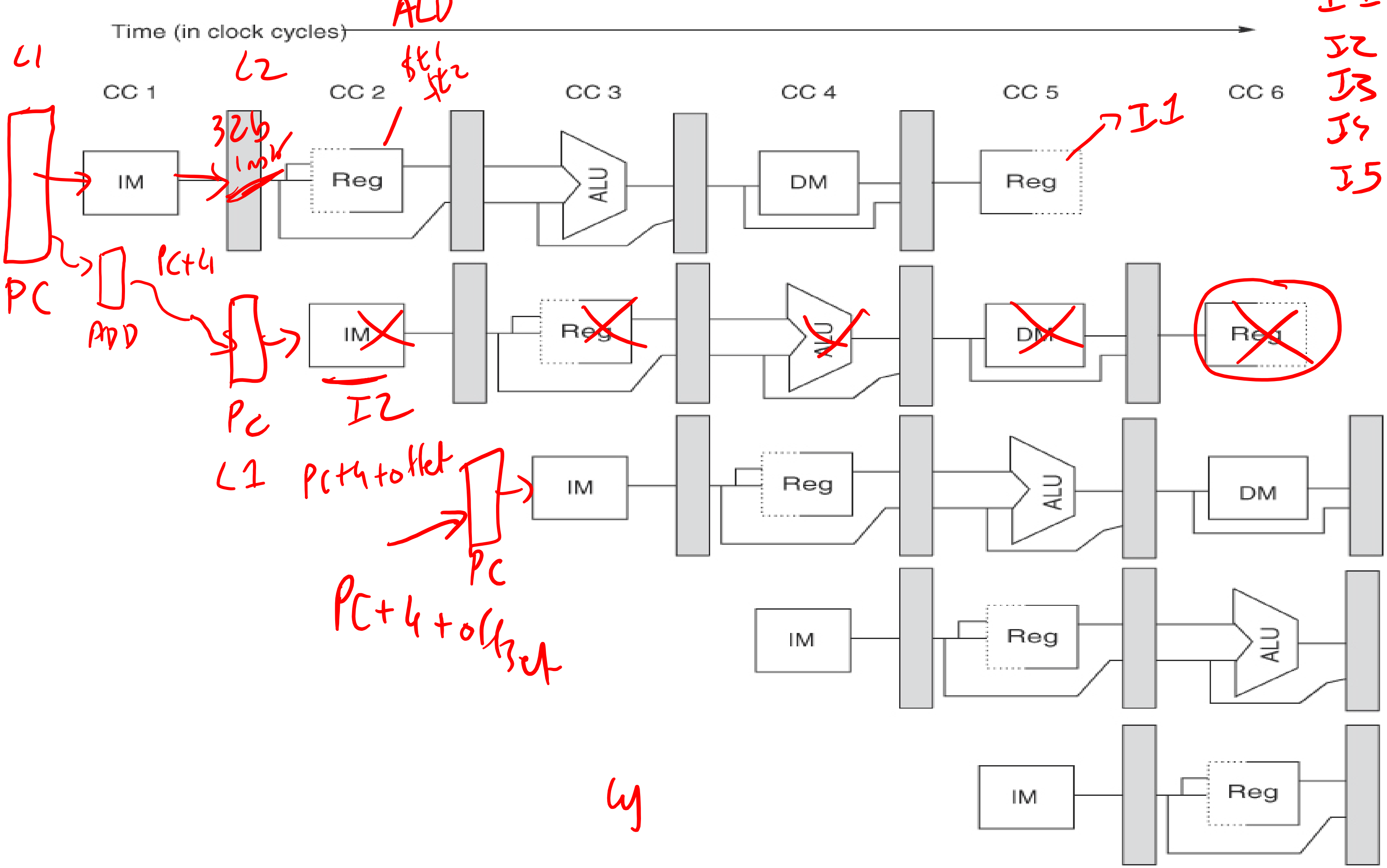
A 5-Stage Pipeline



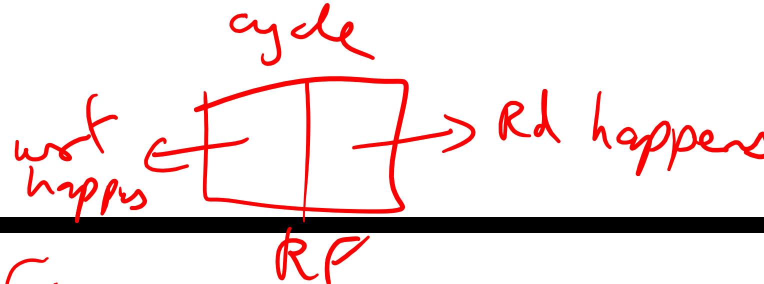
Source: H&P textbook

A 5-Stage Pipeline

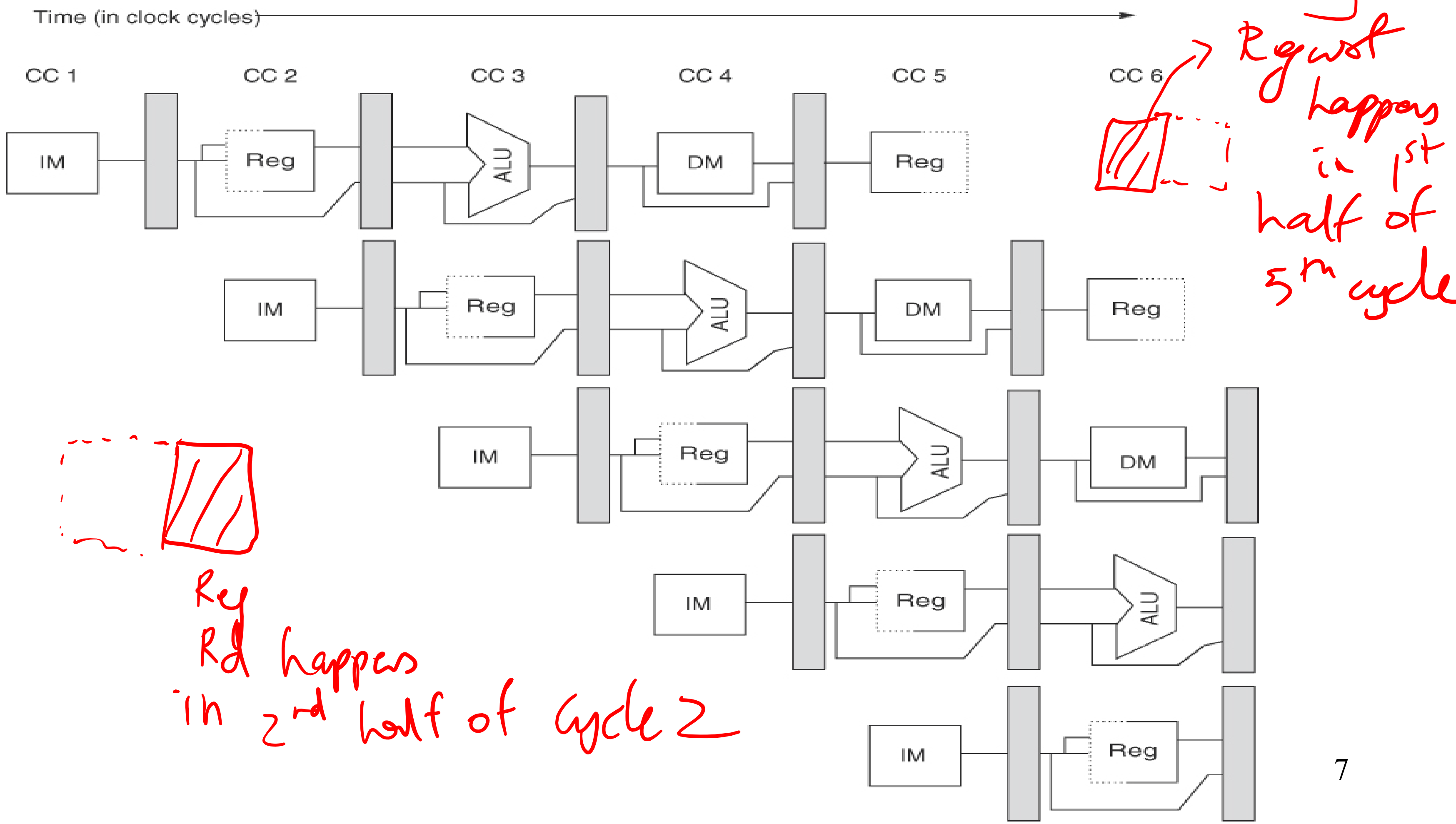
Use the PC to access the I-cache and increment PC by 4



A 5-Stage Pipeline



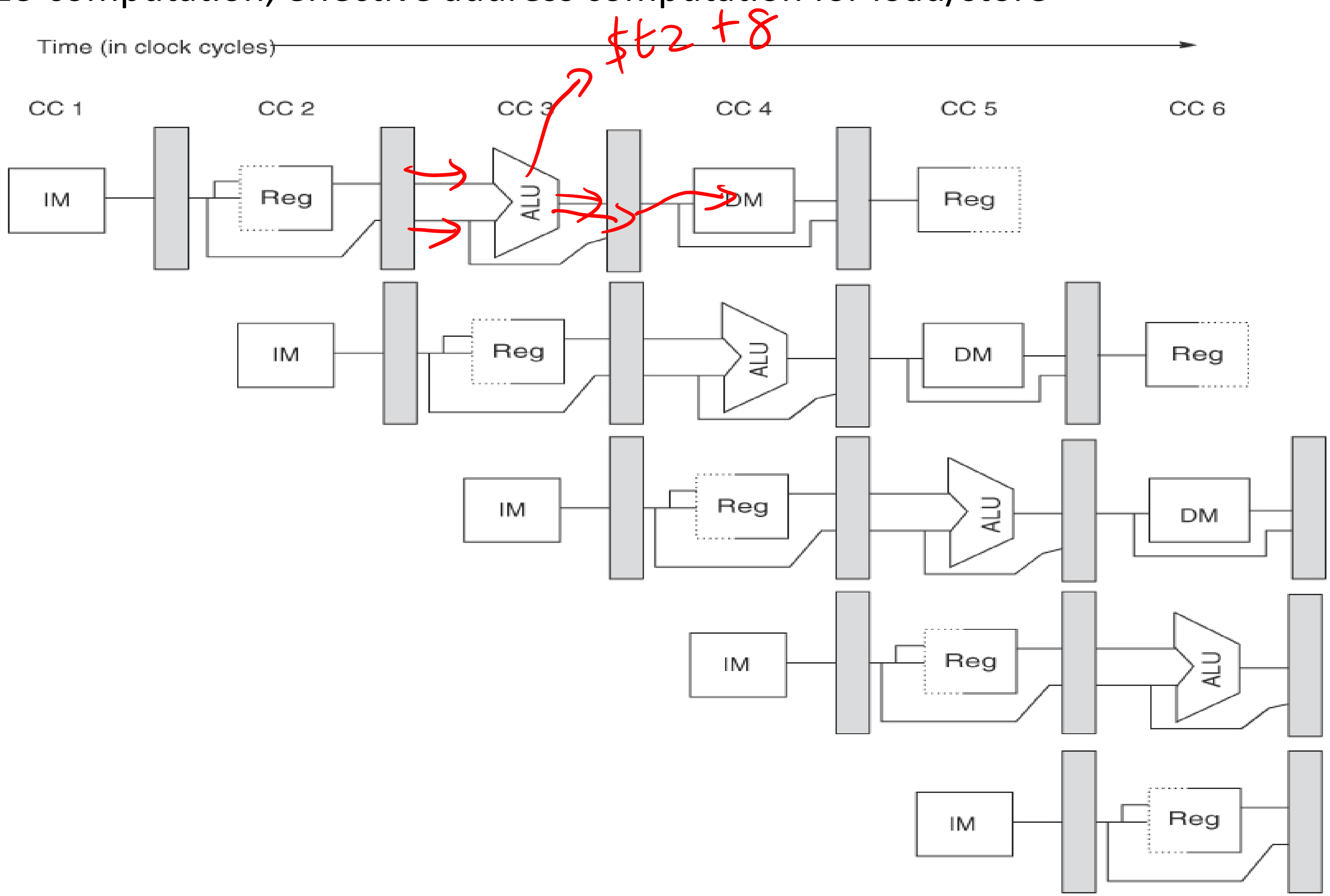
Read registers, compare registers, compute branch target; for now, assume branches take 2 cyc (there is enough work that branches can easily take more)



A 5-Stage Pipeline

lw $t1, 8($t2)

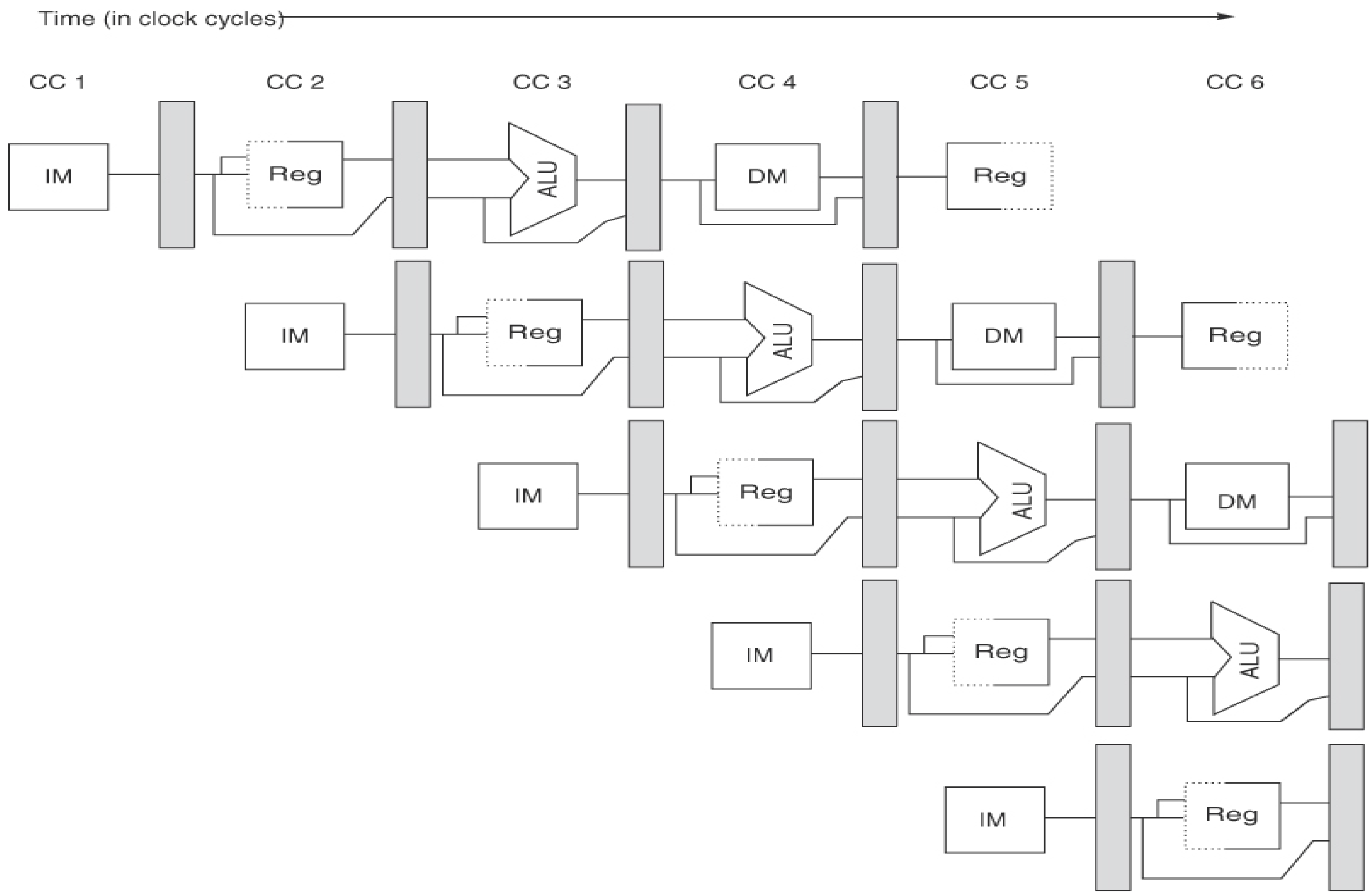
ALU computation, effective address computation for load/store



A 5-Stage Pipeline

lw/sw

Memory access to/from data cache, stores finish in 4 cycles



A 5-Stage Pipeline

Caches

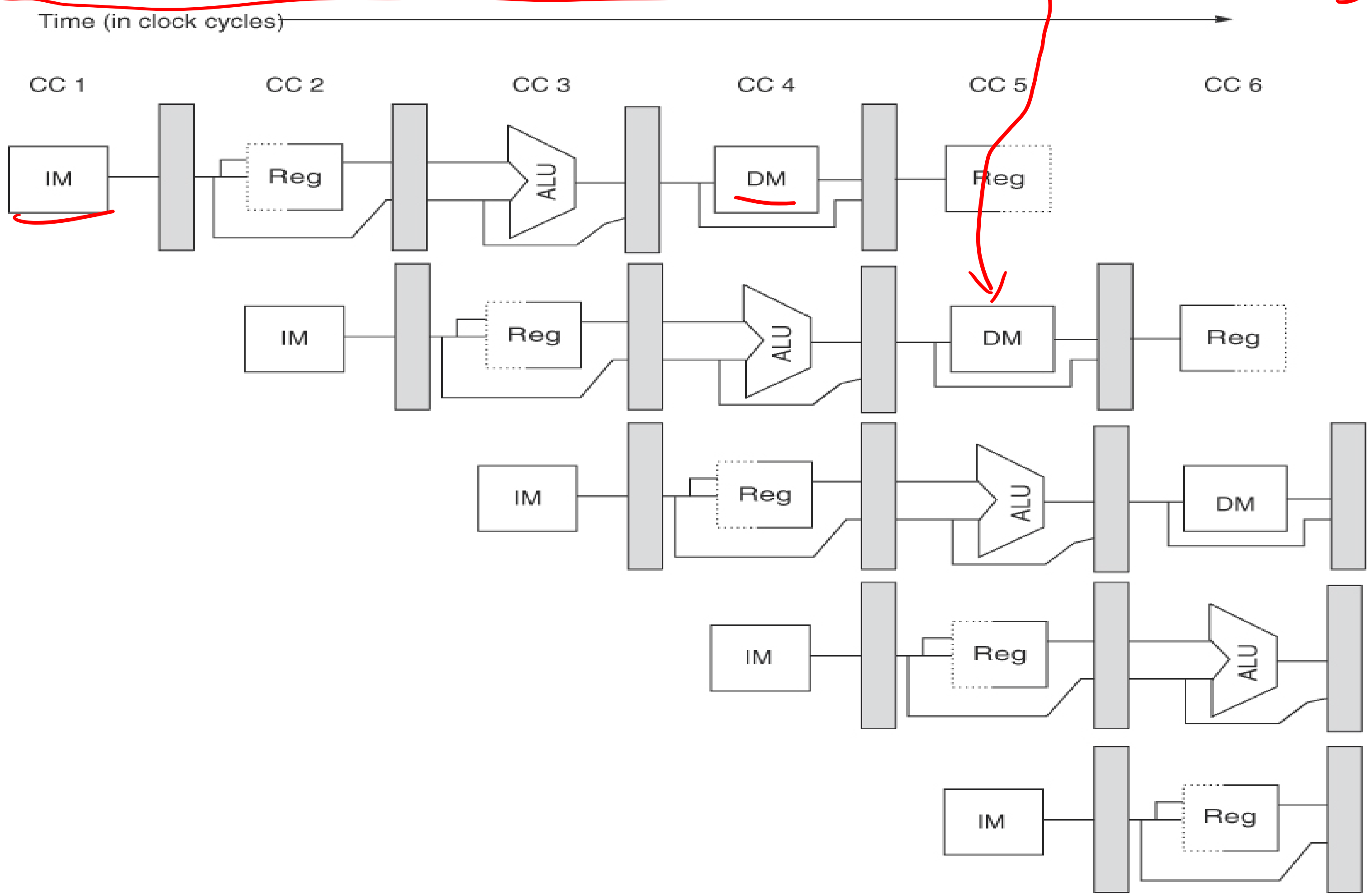
90% hit rate

1 cycle

Mem

300 cyc

Write result of ALU computation or load into register file



Pipeline Summary

	RR	ALU	DM	RW
ADD R1, R2, → R3	Rd R1,R2	R1+R2	--	Wr R3
BEQ R1, R2, 100	Rd R1, R2 Compare, Set PC	--	--	--
LD 8[R3] → R6	Rd R3	R3+8	Get data	Wr R6
ST 8[R3] ← R6	Rd R3,R6	R3+8	Wr data	--

Performance Improvements?

- Does it take longer to finish each individual job?
- Does it take shorter to finish a series of jobs?
- What assumptions were made while answering these questions?
  - No dependences between instructions
  - Easy to partition circuits into uniform pipeline stages
  - No latch overhead
- Is a 10-stage pipeline better than a 5-stage pipeline?

Quantitative Effects

- As a result of pipelining:
  - Time in ns per instruction goes up
  - Each instruction takes more cycles to execute
  - But… average CPI remains roughly the same
  - Clock speed goes up
  - Total execution time goes down, resulting in lower average time per instruction
  - Under ideal conditions, speedup
    = ratio of *elapsed times between successive instruction completions*
    = number of pipeline stages = increase in clock speed

Conflicts/Problems

- I-cache and D-cache are accessed in the same cycle – it helps to implement them separately

- Registers are read and written in the same cycle – easy to deal with if register read/write time equals cycle time/2

- Branch target changes only at the end of the second stage -- what do you do in the meantime?

Hazards

- Structural hazards: different instructions in different stages (or the same stage) conflicting for the same resource
- Data hazards: an instruction cannot continue because it needs a value that has not yet been generated by an earlier instruction
- Control hazard: fetch cannot continue because it does not know the outcome of an earlier branch – special case of a data hazard – separate category because they are treated in different ways