


Lecture 16: Basic Pipelining

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

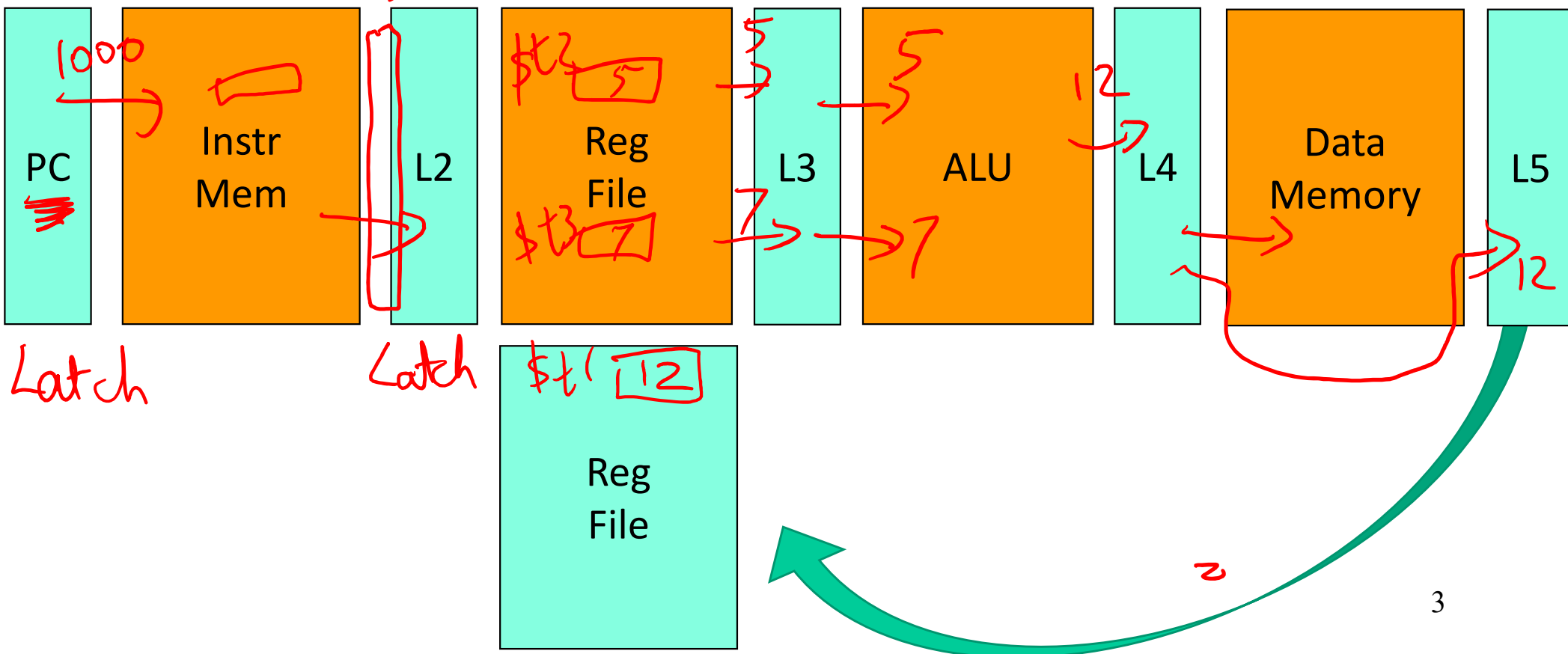
Tue 12th Mar Review Session
Thu 14th Mar Midterm
HW6 due Fri 11:59pm

Midterm Prep

- In-class midterm 2 weeks away
- Prep: homework, notes/slides/examples, videos, sample midterm
- 80% homeworks, 10% brief concept questions, 10% difficult/new
- Time constrained
- ⇒ • MIPS assembly questions  proc saves/restores
find the bugs
write code — 15 lines
- Single sheet of notes (both sides) – green sheet allowed
- Phone/calculator allowed for calculations
- 90 minute test – 10:40 – 12:10 1.27×2^4

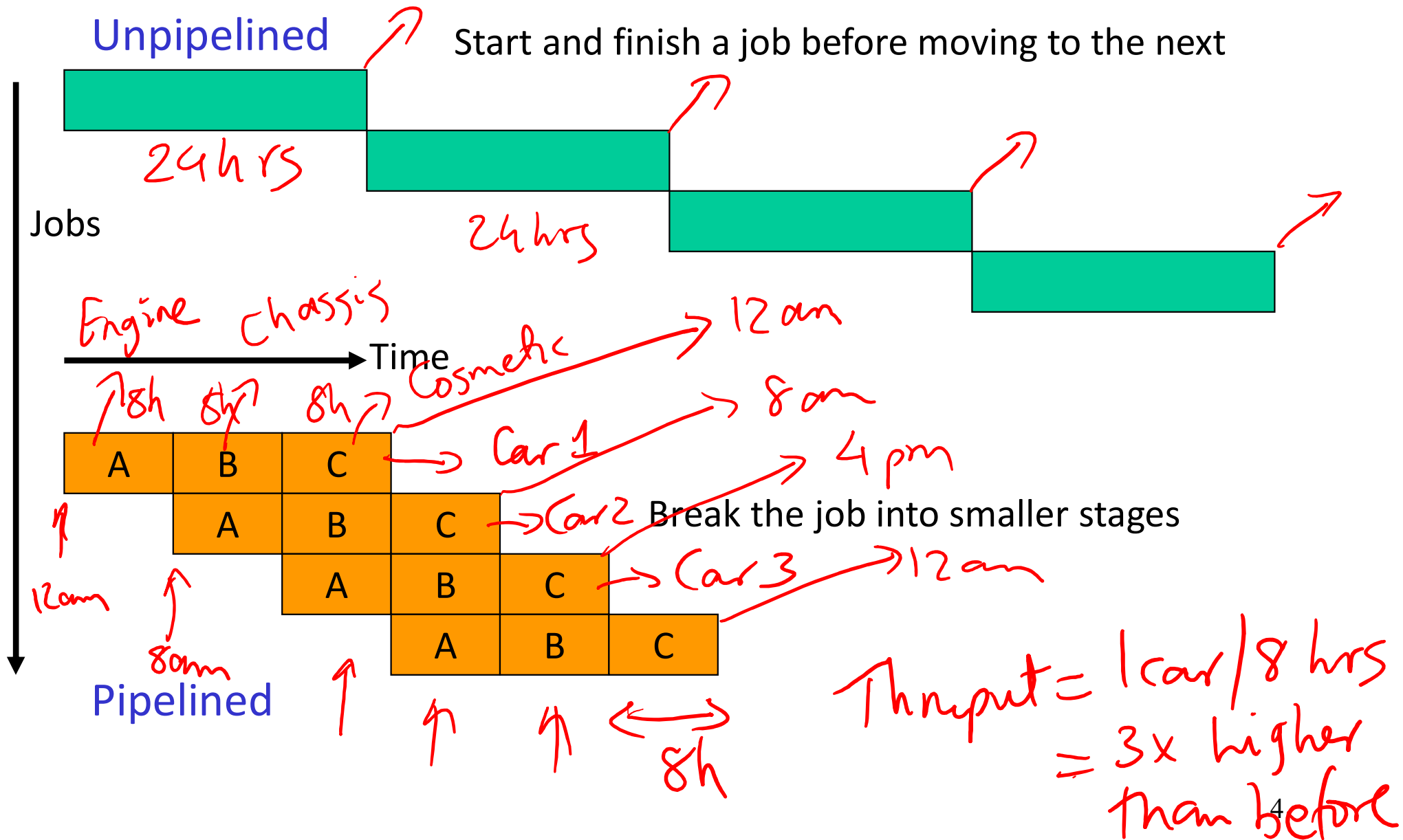
Multi-Stage Circuit

- 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

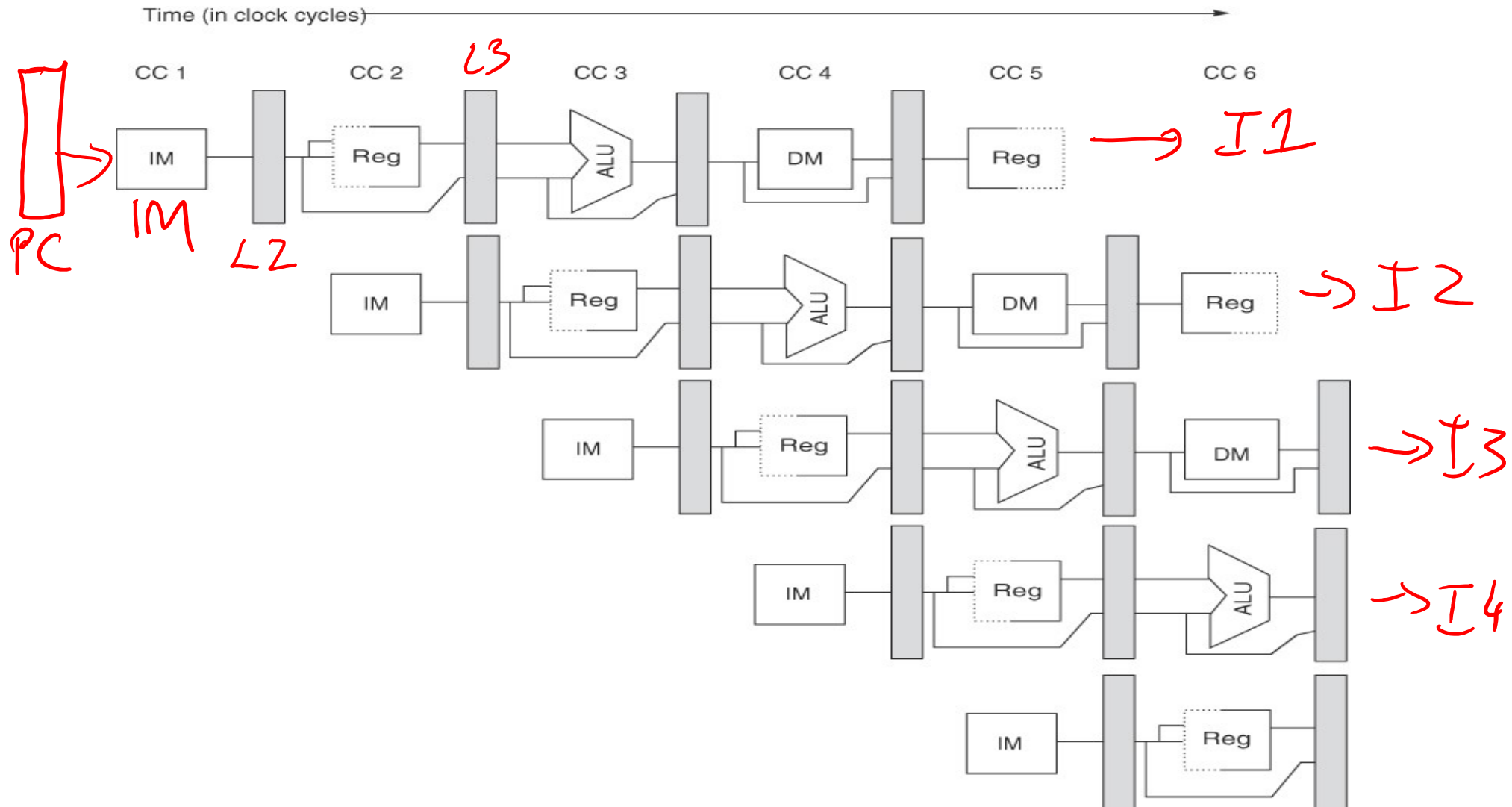
Thruput = 1 car / 24 hrs



Performance Improvements?

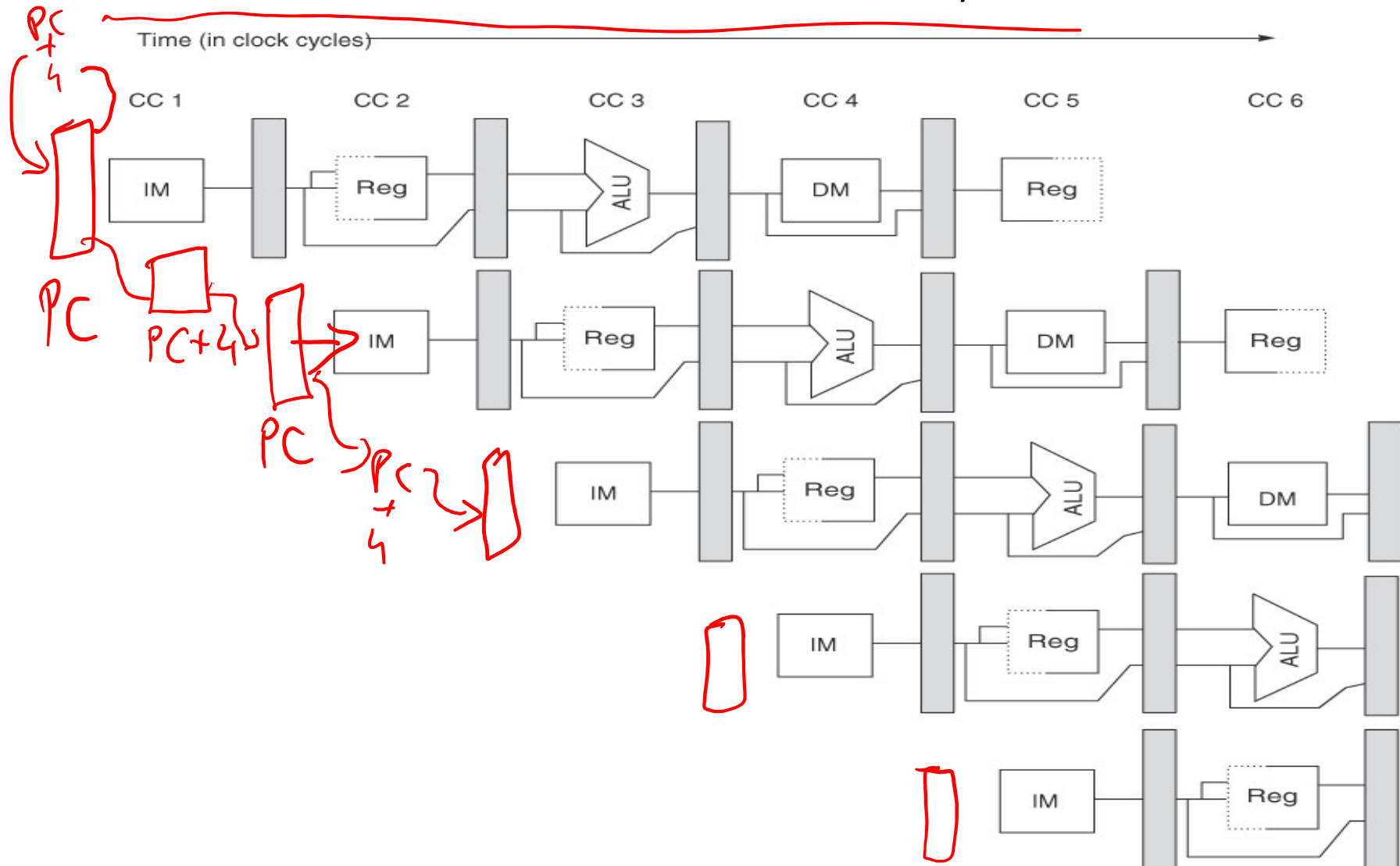
- With pipelining:
- Does it take longer to finish each individual job?
*(ideal)
Equal time
or more time
(overheads)*
 - Does it take shorter to finish a series of jobs?
Yes (beoz of parallelism)
 - What assumptions were made while answering these questions?
 - Is a 10-stage pipeline better than a 5-stage pipeline?

A 5-Stage Pipeline



A 5-Stage Pipeline

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



A 5-Stage Pipeline

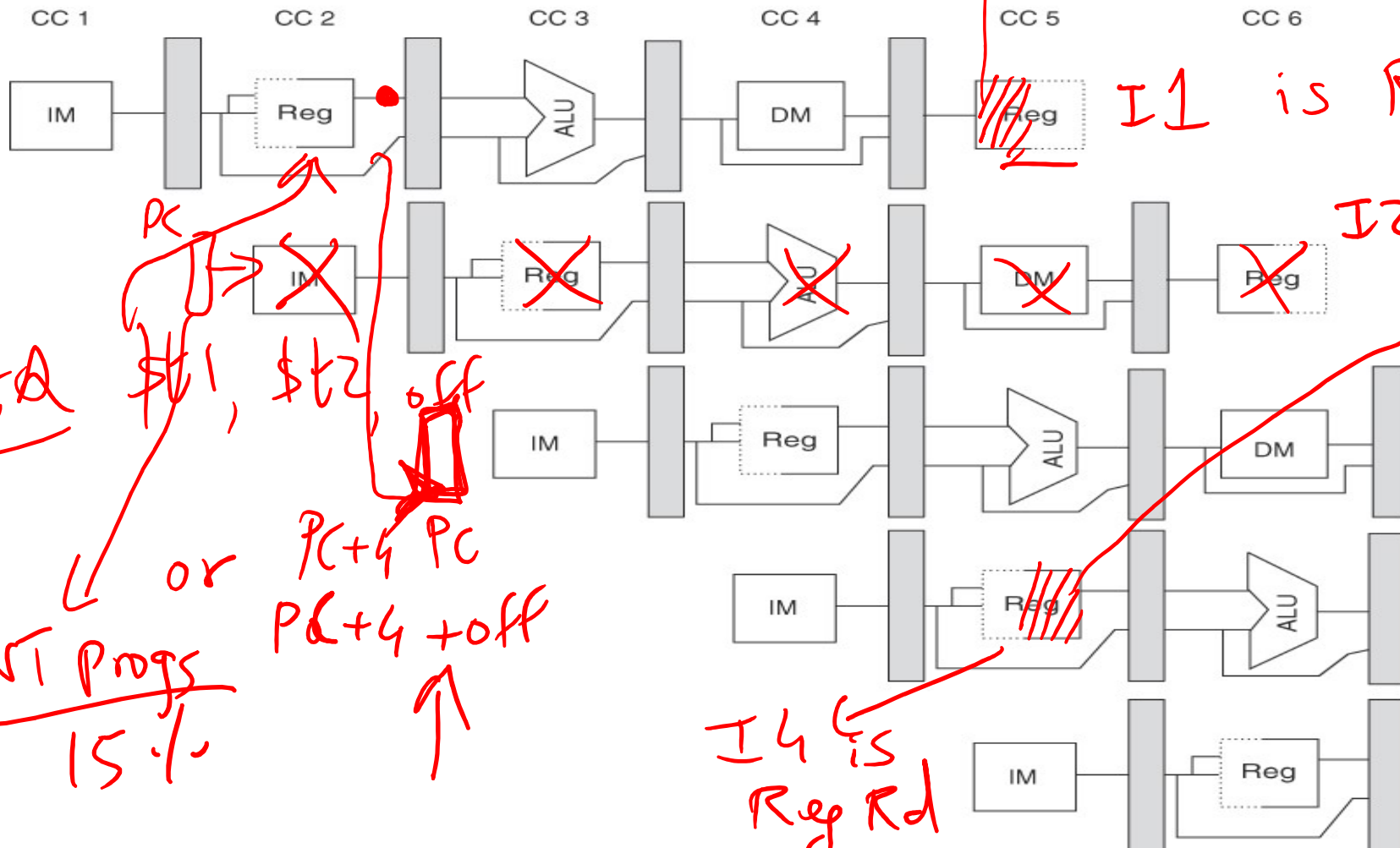
PC=1000 BEQ
PC=1004 ADD

Reg ~~Wr~~ Wr 10%
5%

40%
1%

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

Time (in clock cycles)

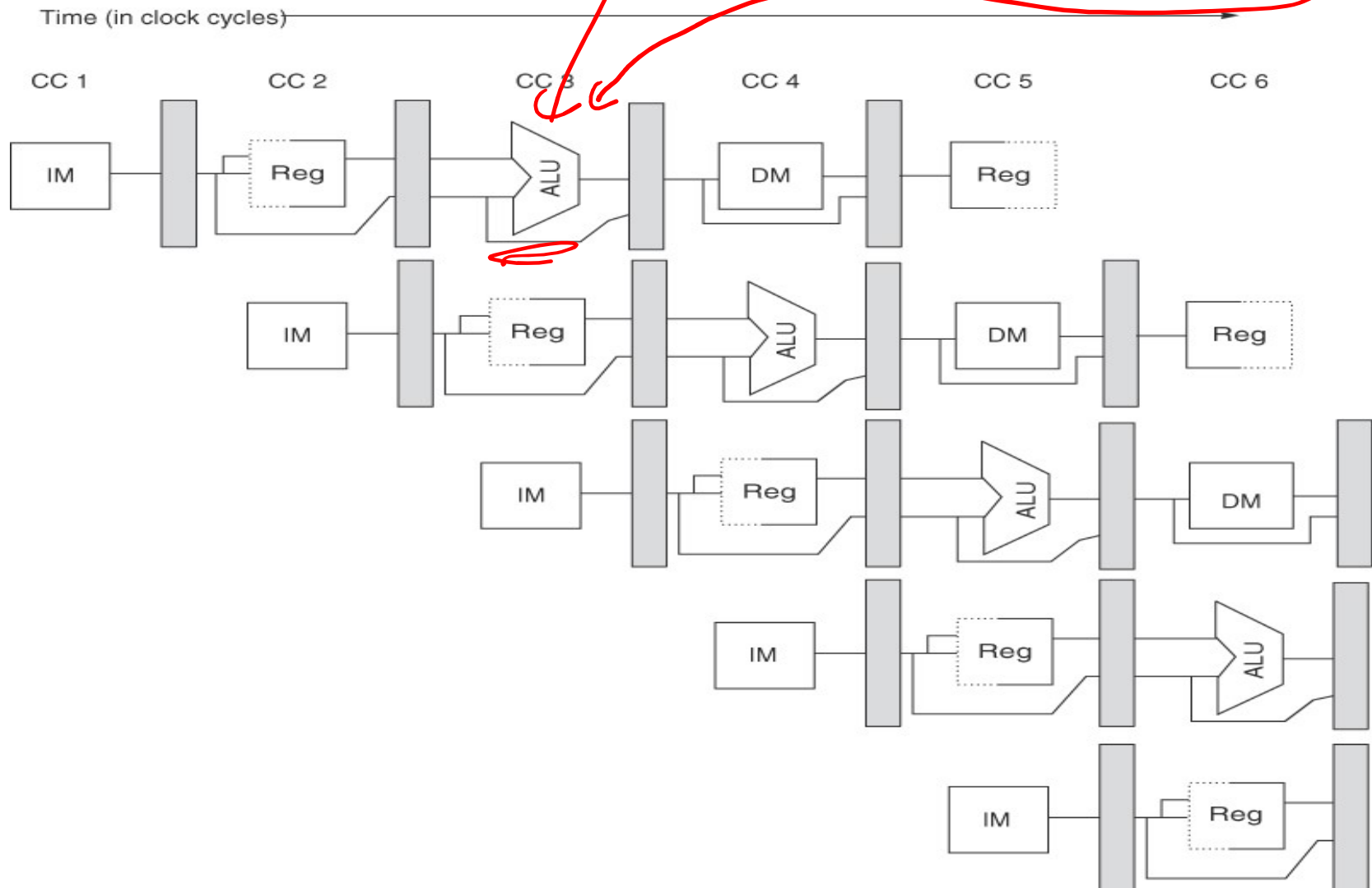


A 5-Stage Pipeline

add \$t1, \$t2, \$t3
lw \$t1, 8(\$t2)

ALU computation, effective address computation for load/store

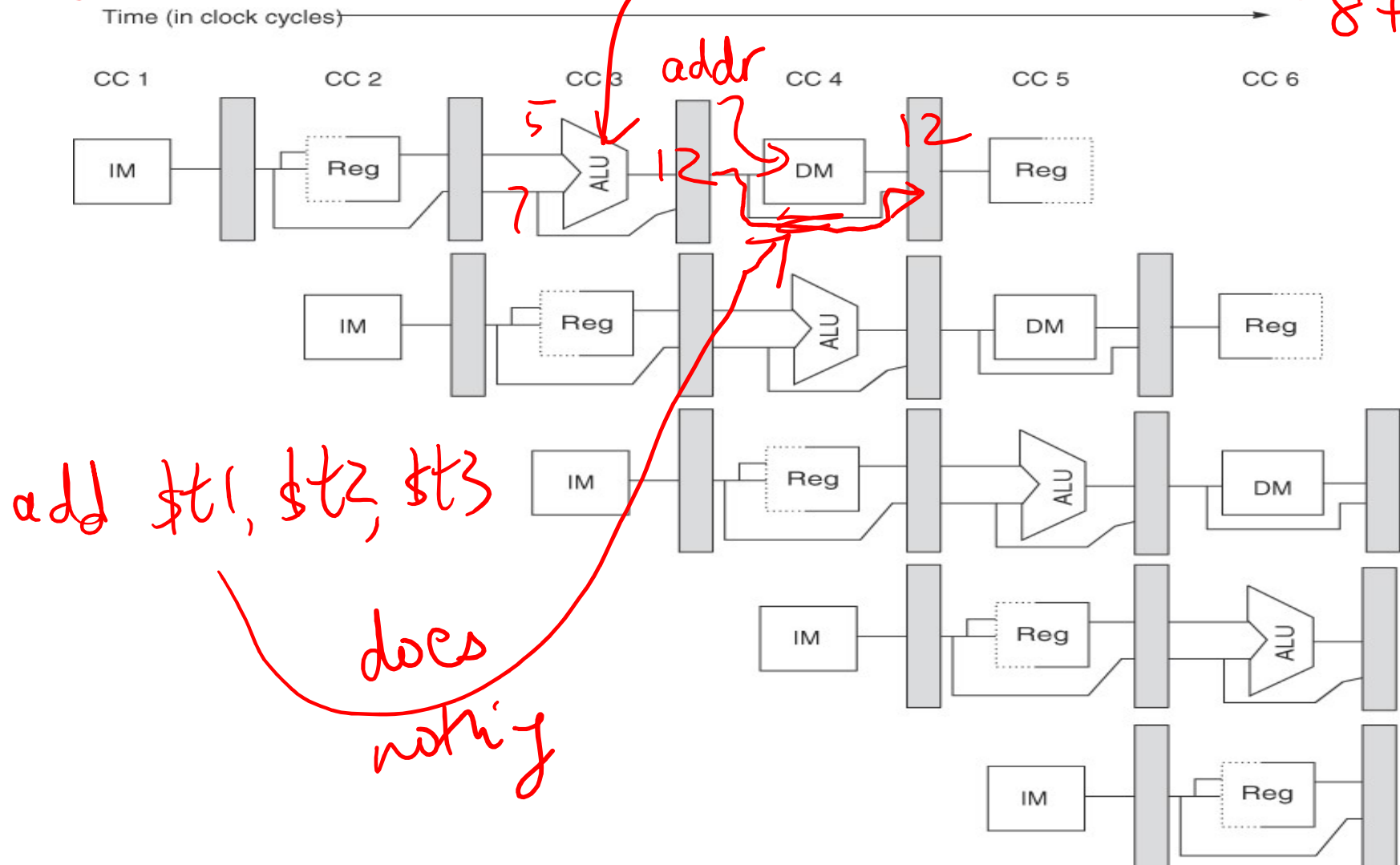
8 + \$t2



A 5-Stage Pipeline

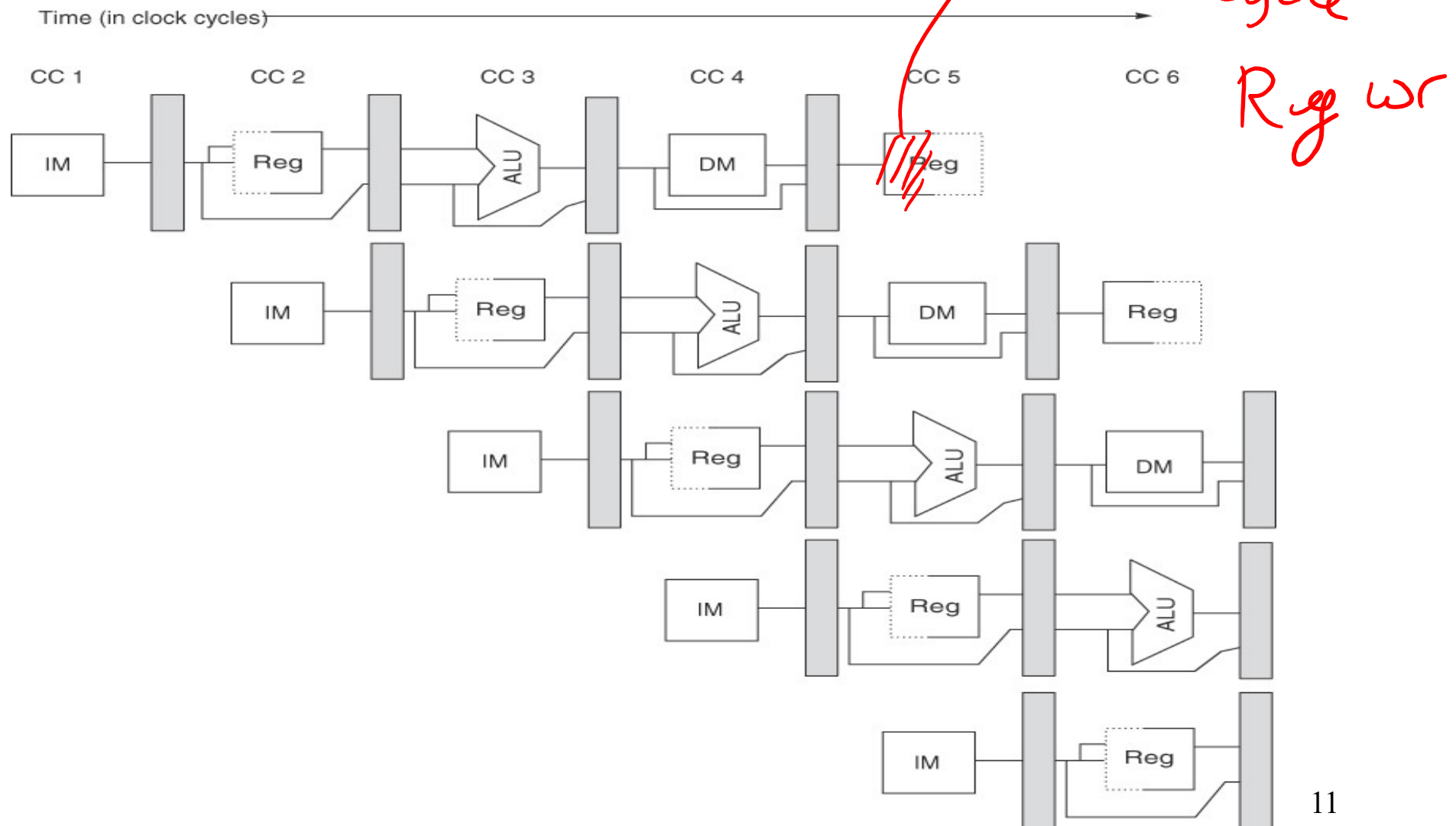
Memory access to/from data cache, stores finish in 4 cycles

$lw \$t1, 8(\$t2)$
 $8 + \$t2$



A 5-Stage Pipeline

Write result of ALU computation or load into register file



Pipeline Summary

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

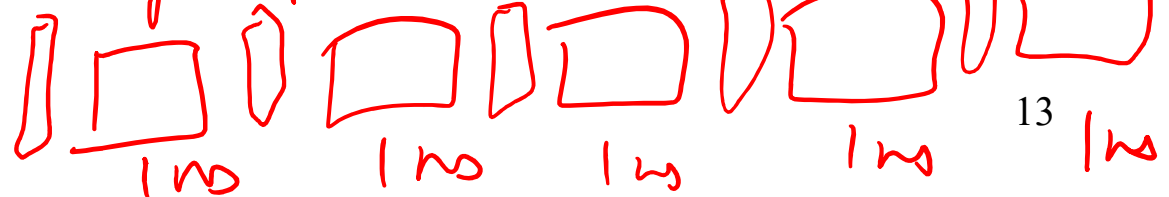
Performance Improvements?

Unpipelined proc/cct
5ns to finish
1 instr

- Does it take longer to finish each individual job? 6ns
Yes (latch overhead)
- Does it take shorter to finish a series of jobs?
1M cycles \Rightarrow complete 1M instrs
1 instr per cycle
1 instr per 1ns
1.2ns
- What assumptions were made while answering these questions?
no hazards
 - 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?

15 stages

5-stage pipeline



ideally 5x
in practice 3.2x

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