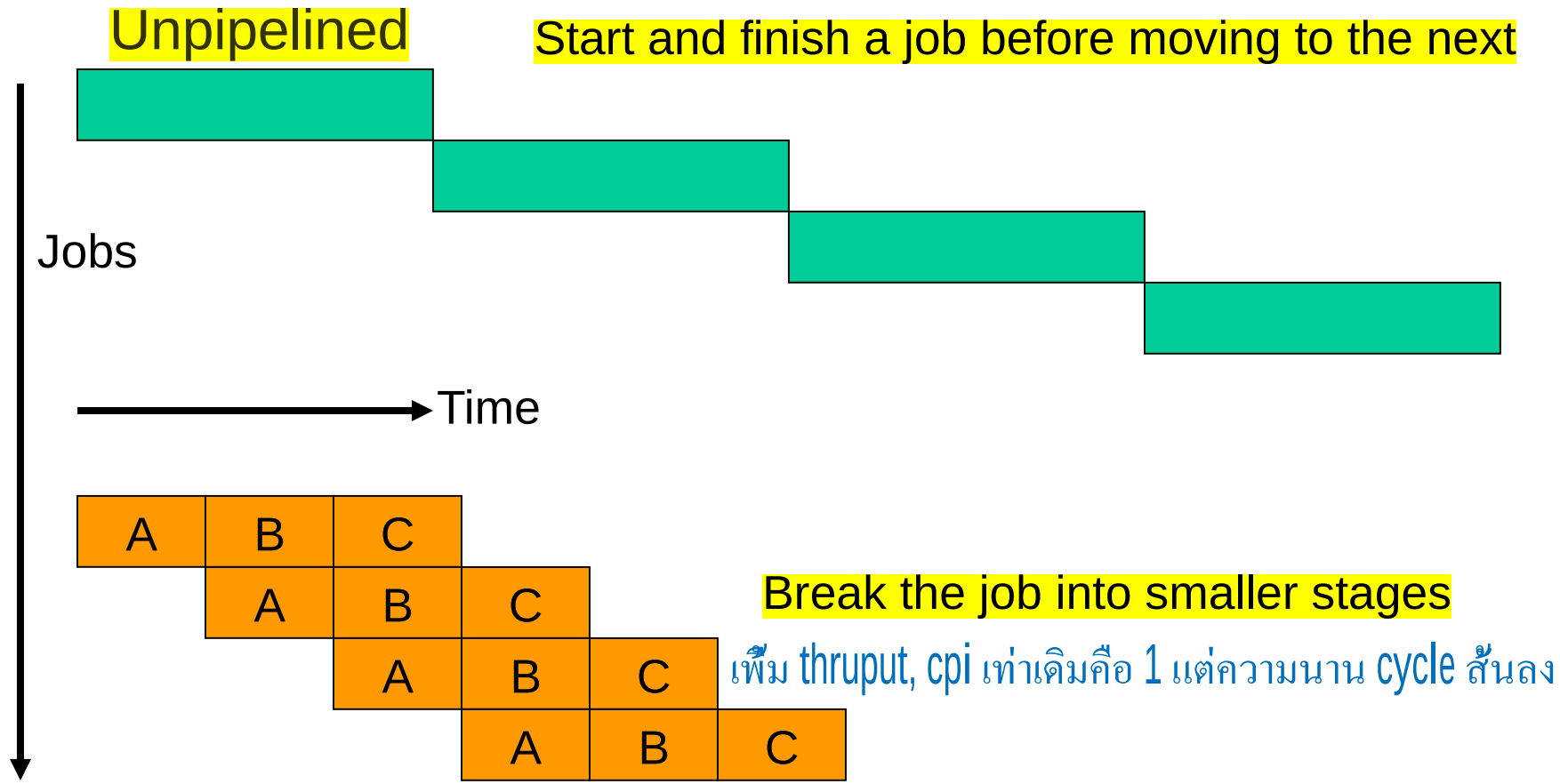
Lecture 17: Basic Pipelining

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

The Assembly Line



Performance Improvements?

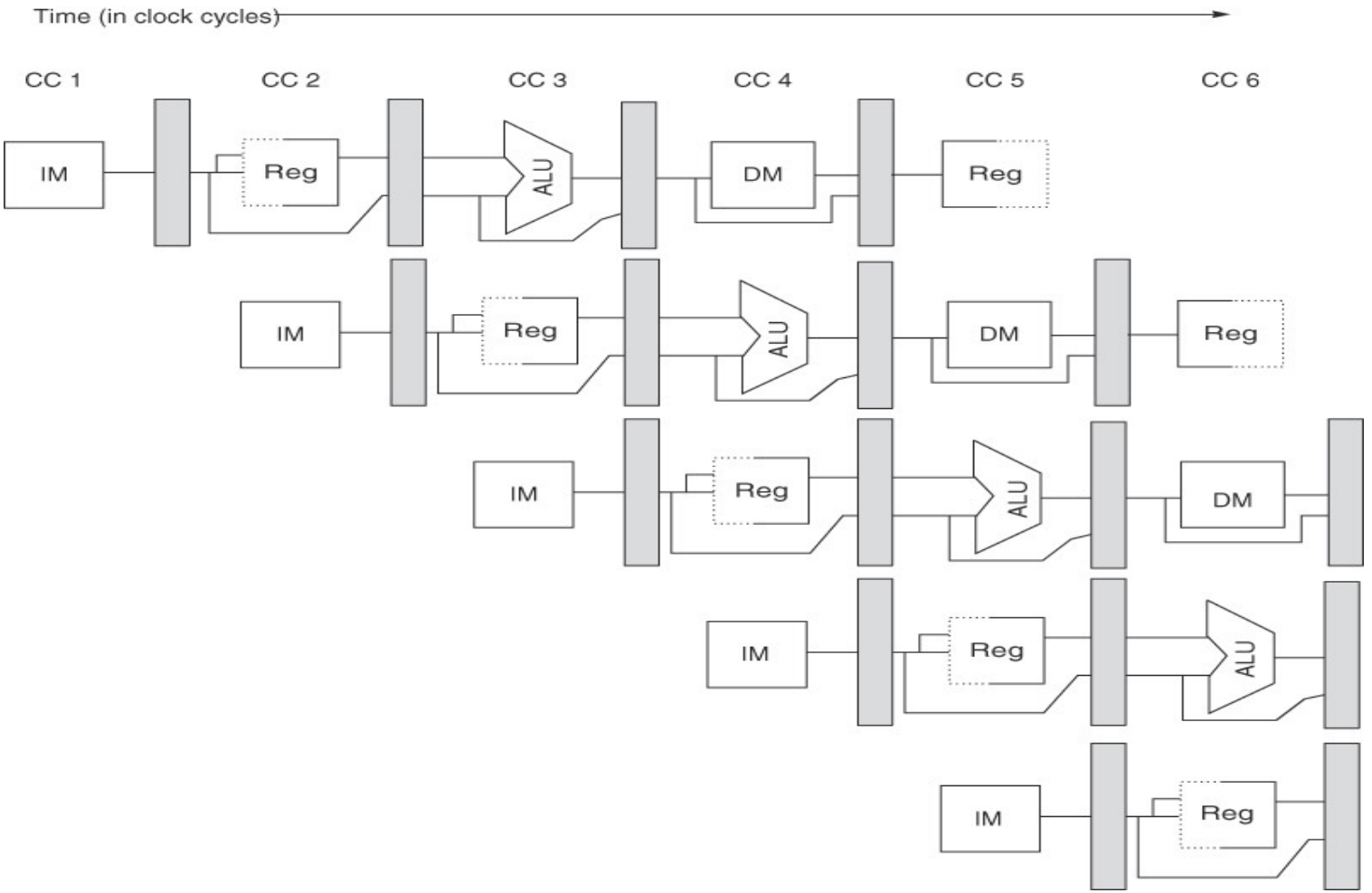
- Does it take longer to finish each individual job? No
- Does it take shorter to finish a series of jobs? Yes
- What assumptions were made while answering these questions? [illegible] overhead (The meaning of overhead is usually an additional expense occurring in addition to normal cost (whatever normal cost may be).) ดีเลน้นแหละ
- Is a 10-stage pipeline better than a 5-stage pipeline?

เพิ่ม stage เยอะๆ latch overhead จะเริ่มมีผลมากขึ้น
แล้วมีโอกาสที่คำสั่งจะเกี่ยวข้องกันในแต่ละขั้นมากขึ้น
สรุปคือไม่ได้ดีกว่าเสมอไป

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

A 5-Stage Pipeline

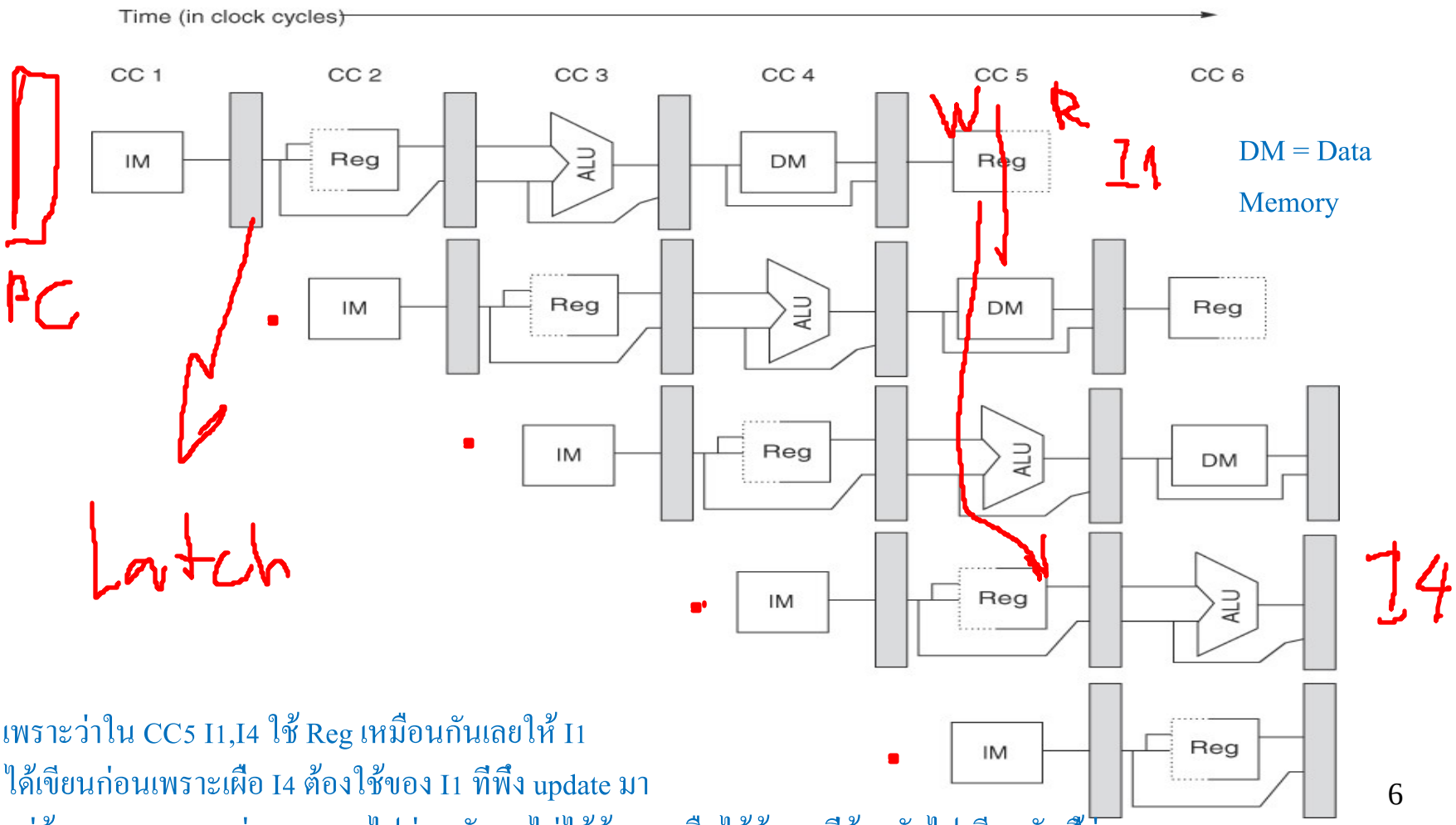


Source: H&P textbook

A 5-Stage Pipeline

REG แบ่งเป็นเขียนครึ่งแรกอ่านครึ่งหลัง
เพื่อตัวอื่นมาทีหลังแล้วอยากอ่านจะได้อ่านได้ข้อมูลล่าสุด

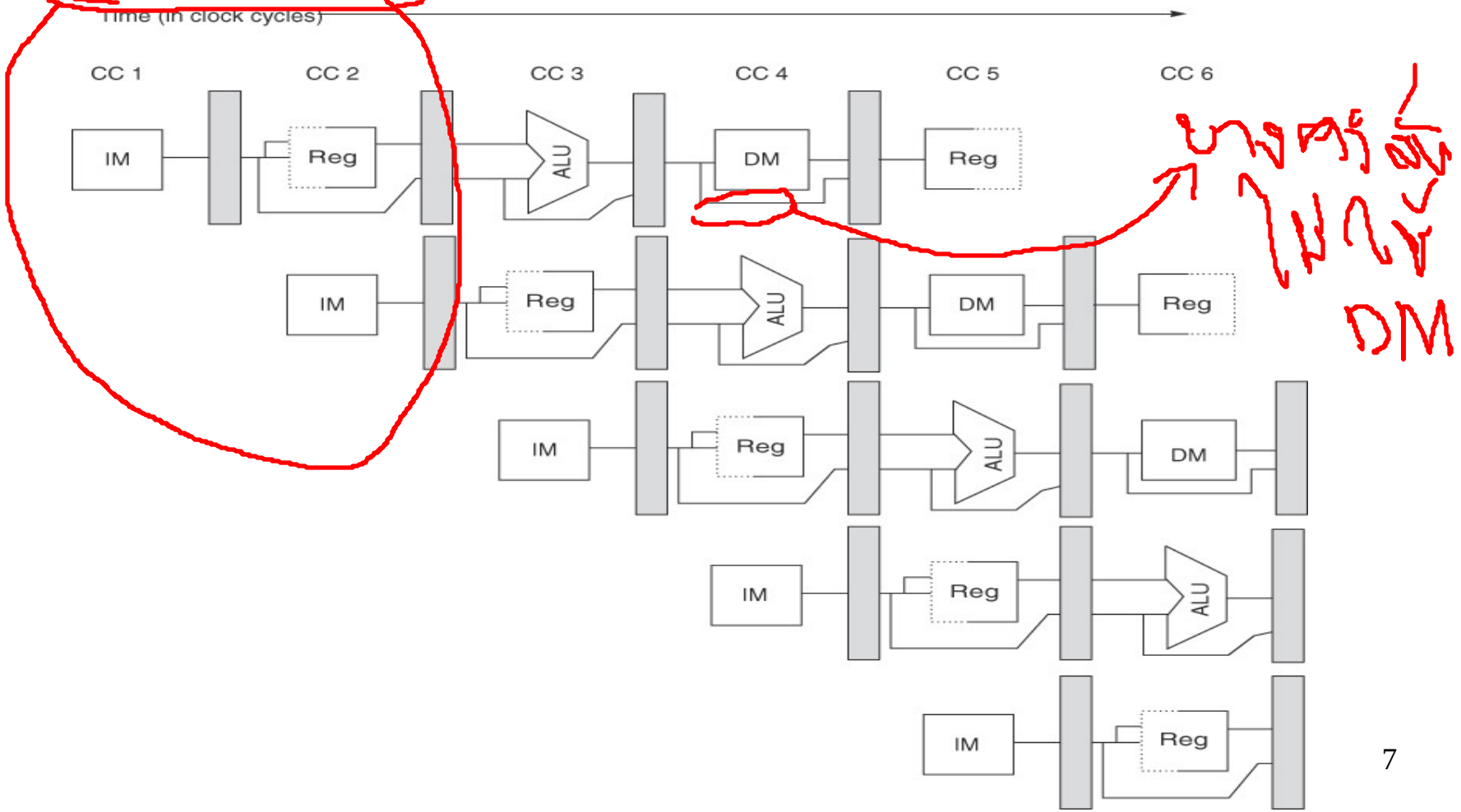
Use the PC to access the I-cache and increment PC by 4



เพราะว่าใน CC5 I1,I4 ใช้ Reg เหมือนกันเลยให้ I1
ได้เขียนก่อนเพราะเผื่อ I4 ต้องใช้ของ I1 ทีพึง update มา
แต่ถ้า I3 พยายามจะอ่านจาก I1 ไปก่อนมันจะไม่ได้ช้อมูลหรือได้ข้อมูลทีล้าสมัยไป เรียนอันนี้ว่า dependency

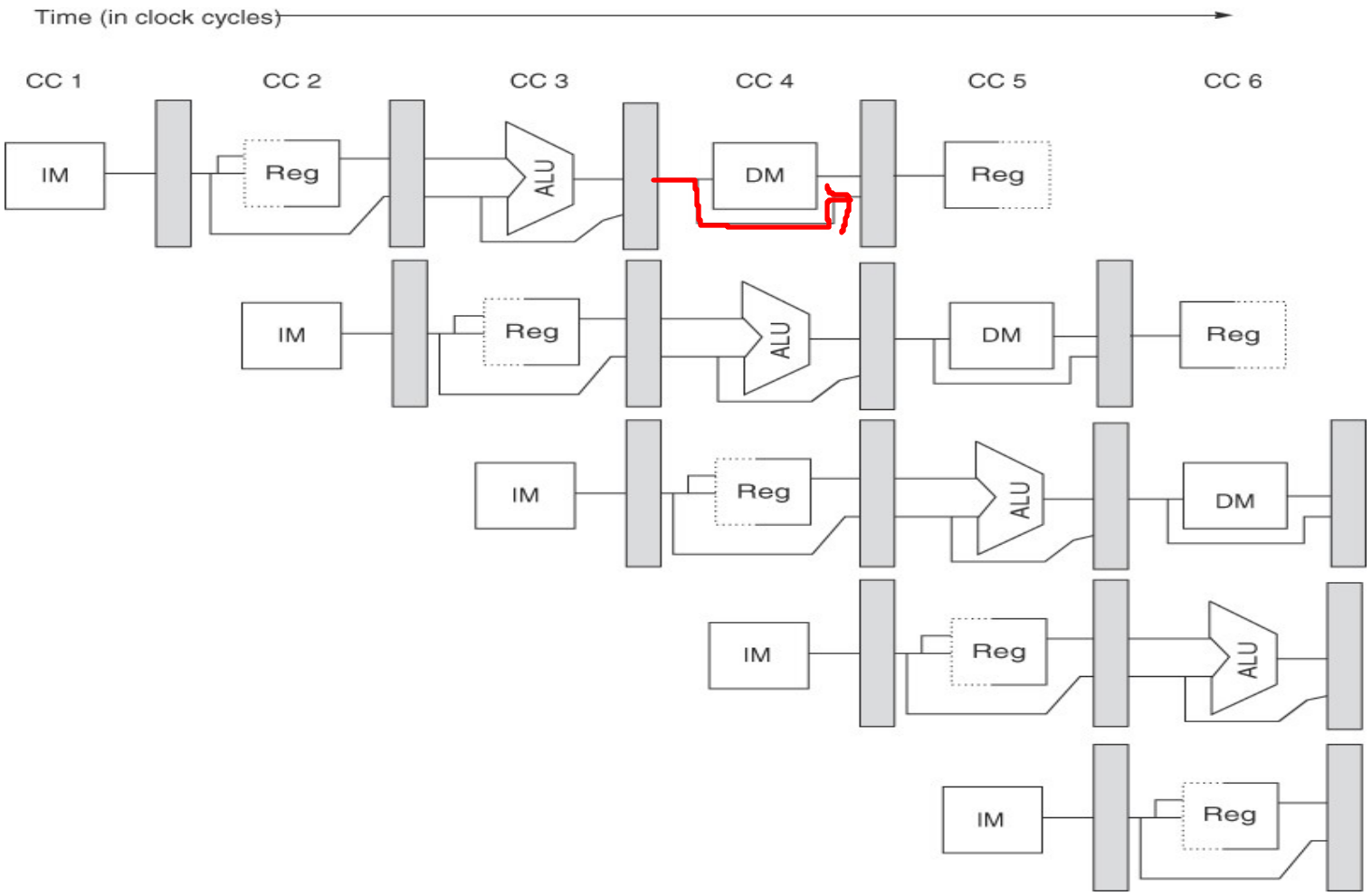
A 5-Stage Pipeline

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



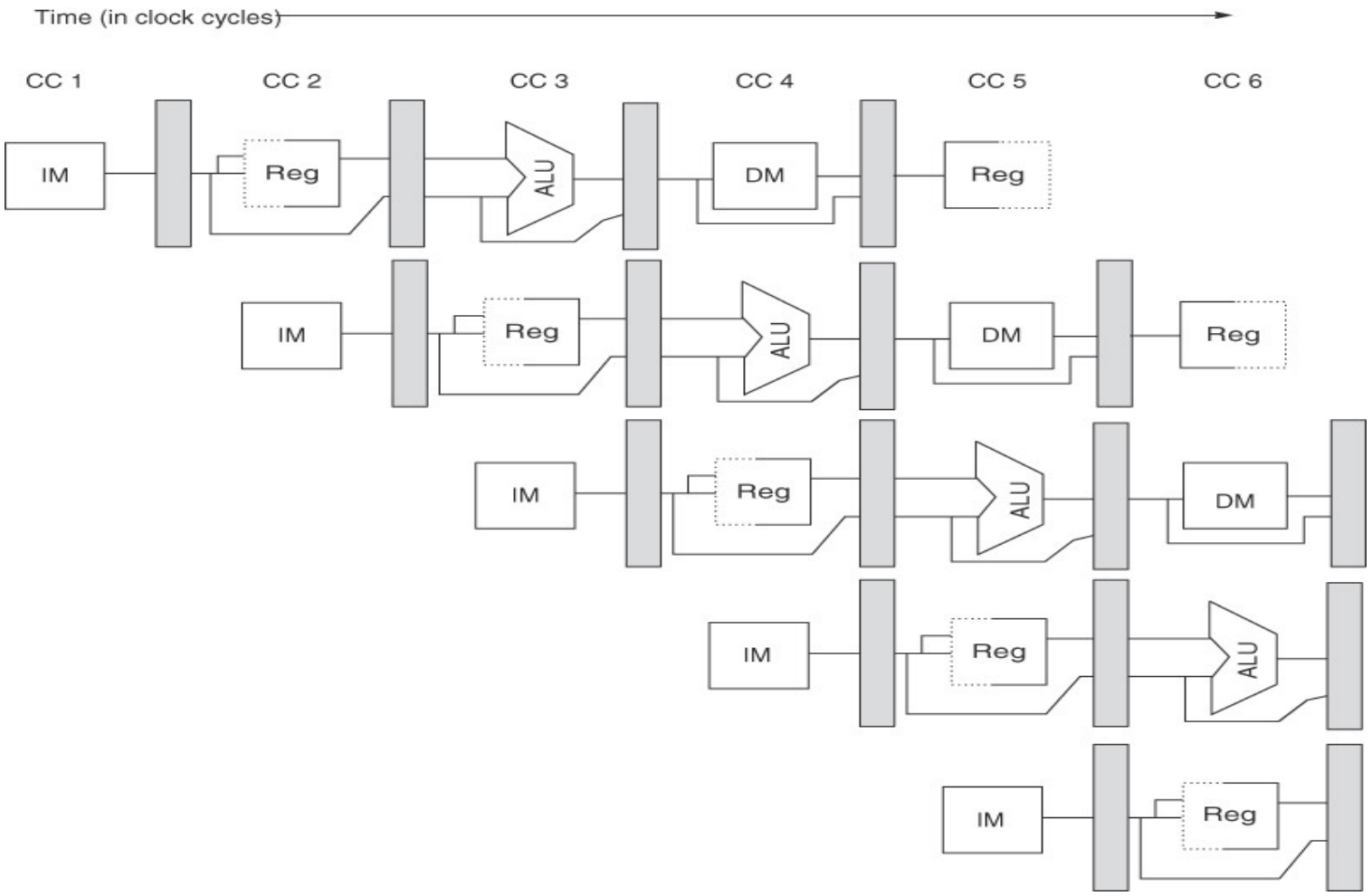
A 5-Stage Pipeline

ALU computation, effective address computation for load/store



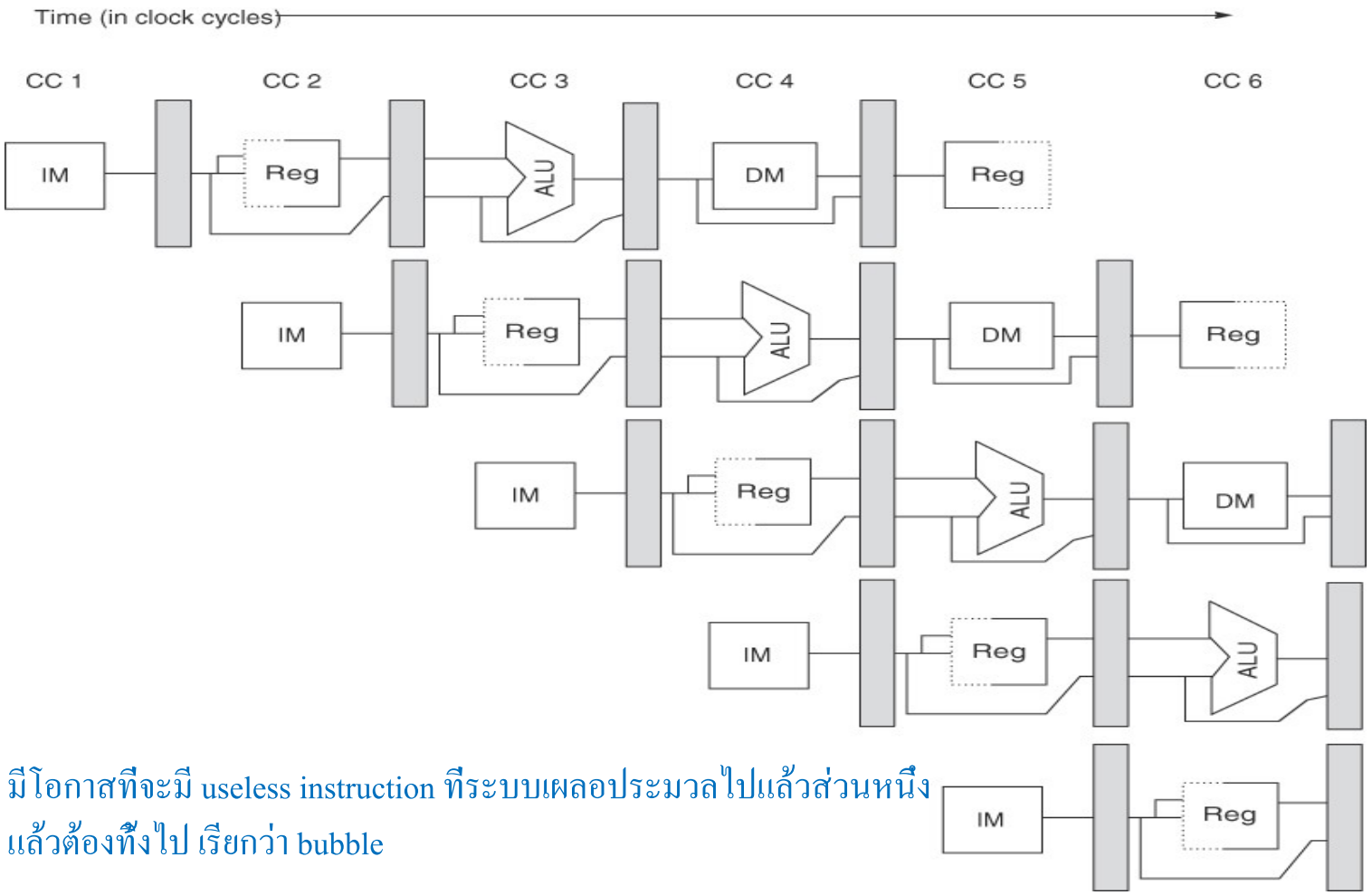
A 5-Stage Pipeline

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



A 5-Stage Pipeline

Write result of ALU computation or load into register file



มีโอกาสทีจะมี useless instruction ทีระบบเผลอประมวลไปแล้วส่วนหนึง แล้วต้องทิ้งไป เรียกว่า bubble

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	--

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