Pipelining The Datapath

Part 1

Adding in Paralellism to the Datapath

- Analogies : Laundry (Wash, Dry, Fold , Put-Away)
  - OR : Car Manufacturer Assembly Line:

Do We really need to wait for One Instruction to Finish Before We Start a New Instruction.

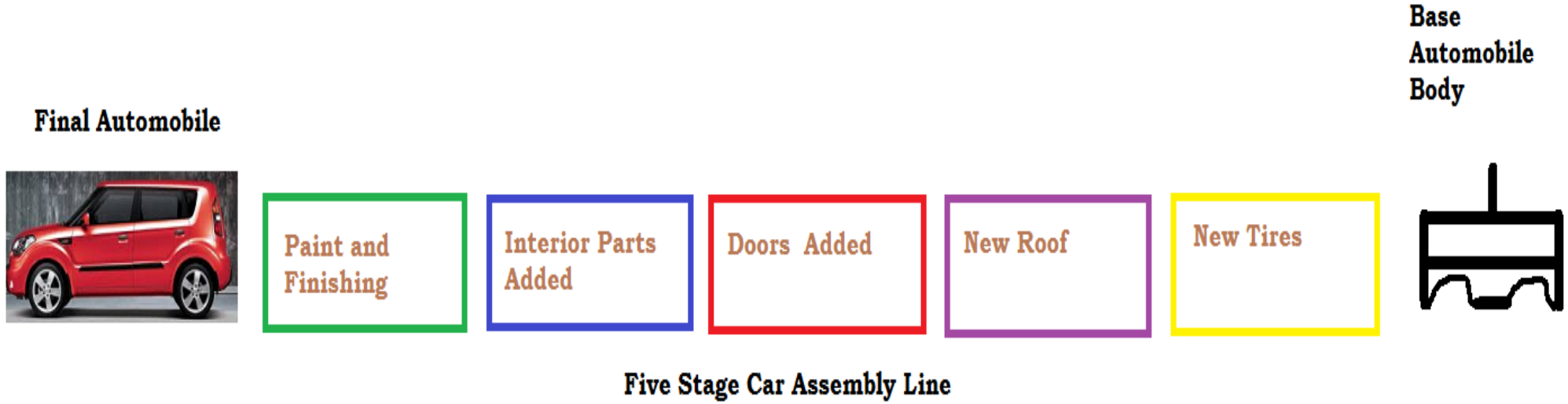
If we are to start a new instruction Right Behind the Existing Instruction: What changes

If Any do we need to add to the Datapath…..

Pipelining

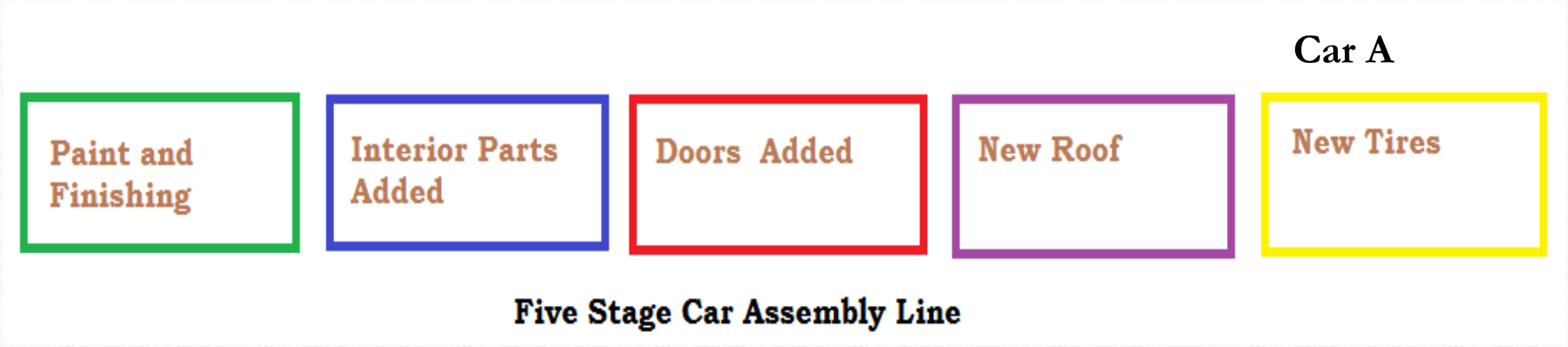
- Readings: Chapter 4, sections 4.5–4.9
- Idea: increase parallelism by overlapping execution of multiple instructions
- Analogy: laundry (wash/dry/fold/put-away)
- Analogy: industrial assembly line

Car Assembly Line:
5 stages



Every Newly Produced Car must pass through the 5 stages

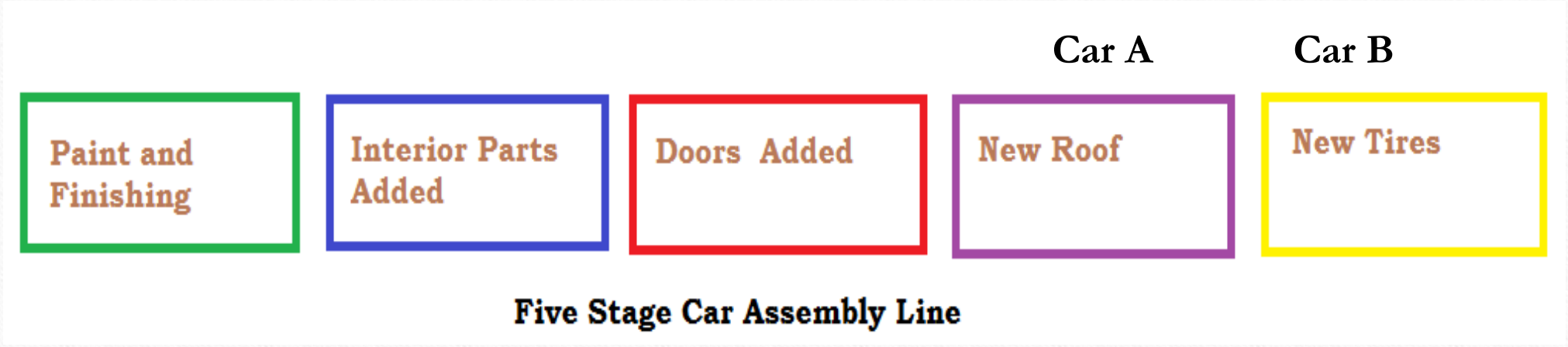
Car Assembly Line: Begin with One Car



Every Newly Produced Car must pass through the 5 stages

Car Assembly Line:

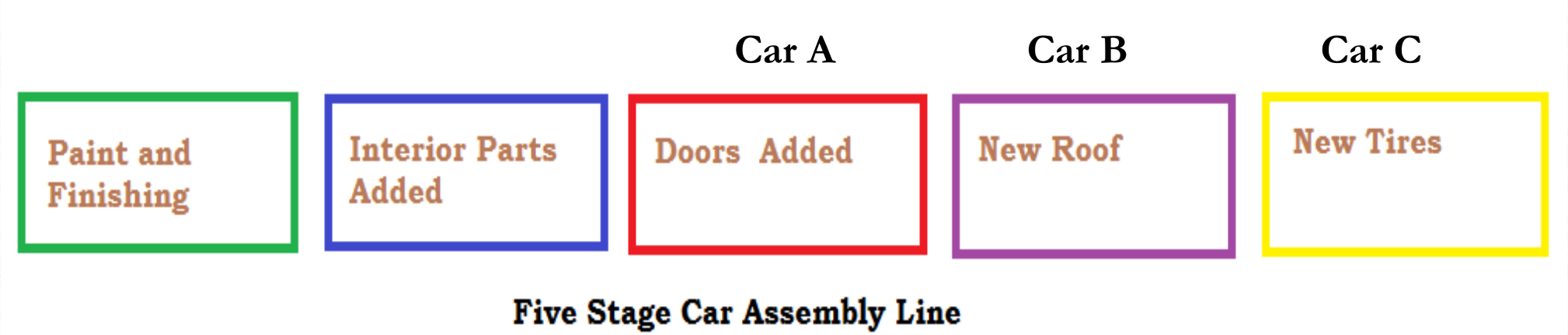
Car A moves on in Assembly Line: New Car Begins assembly right behind it



Every Newly Produced Car must pass through the 5 stages

Car Assembly Line:

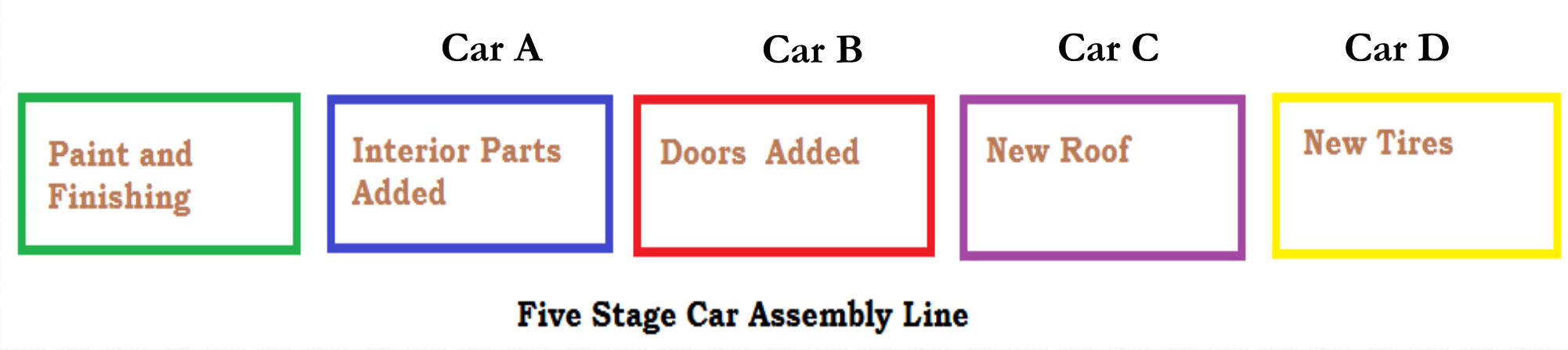
A New Car begins every cycle



Five Stage Car Assembly Line

Every Newly Produced Car must pass through the 5 stages

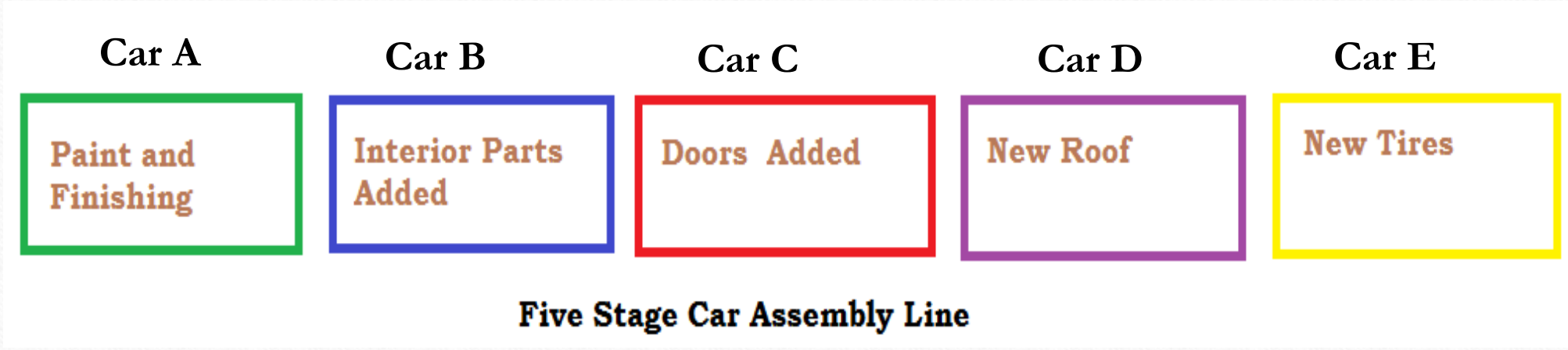
Car Assembly Line:



Five Stage Car Assembly Line

Every Newly Produced Car must pass through the 5 stages

Car Assembly Line:

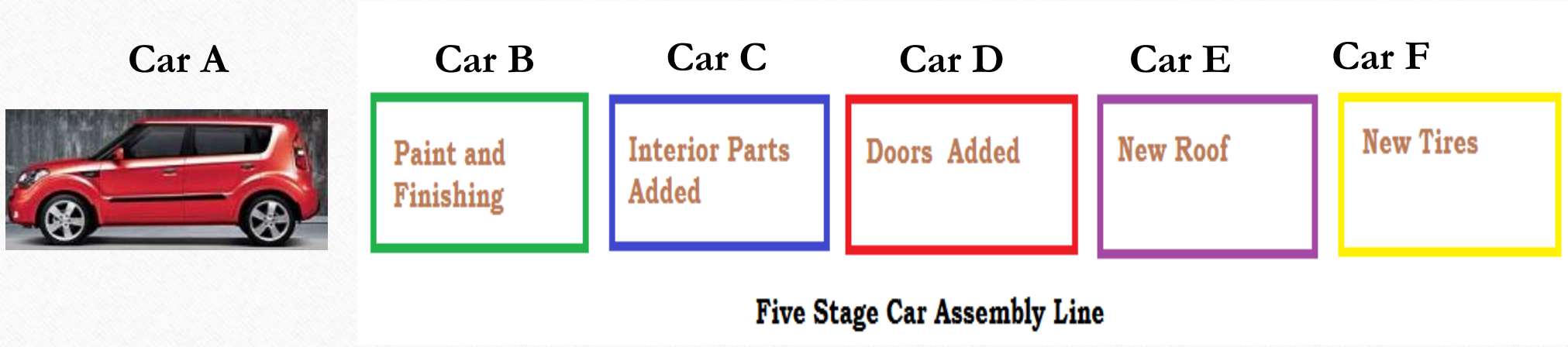


Five Stage Car Assembly Line

Now A NEW Car is complete in Every Cycle

Car Assembly Line:

Now New Cars are generated Every Cycle and A New Car Begins Every Cycle



Now A NEW Car is complete in Every Cycle

Car Assembly Line:

Car A **Car B** **Car C** **Car D** **Car E** **Car F** **Car G**

Paint and Finishing

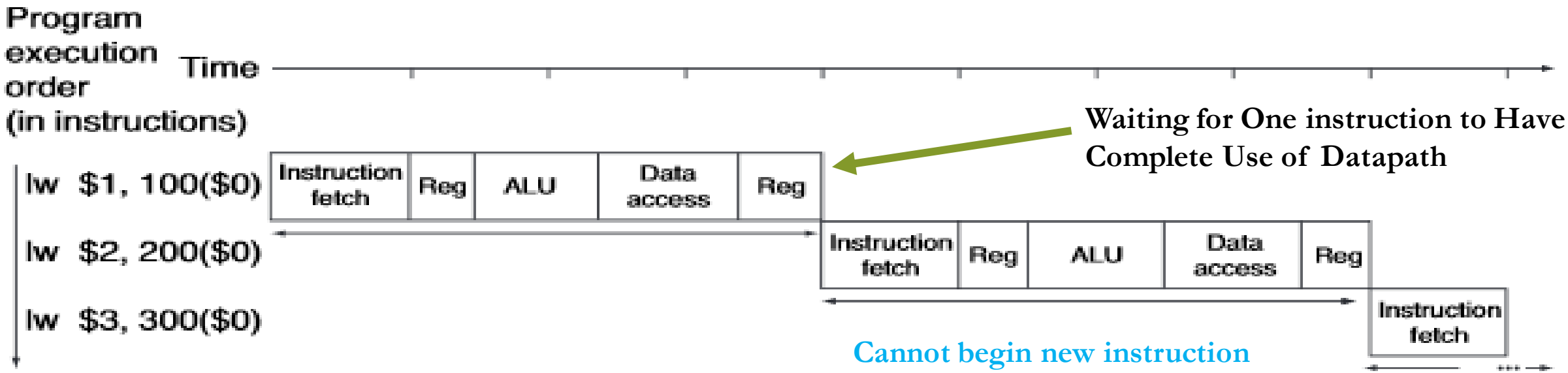
Interior Parts Added

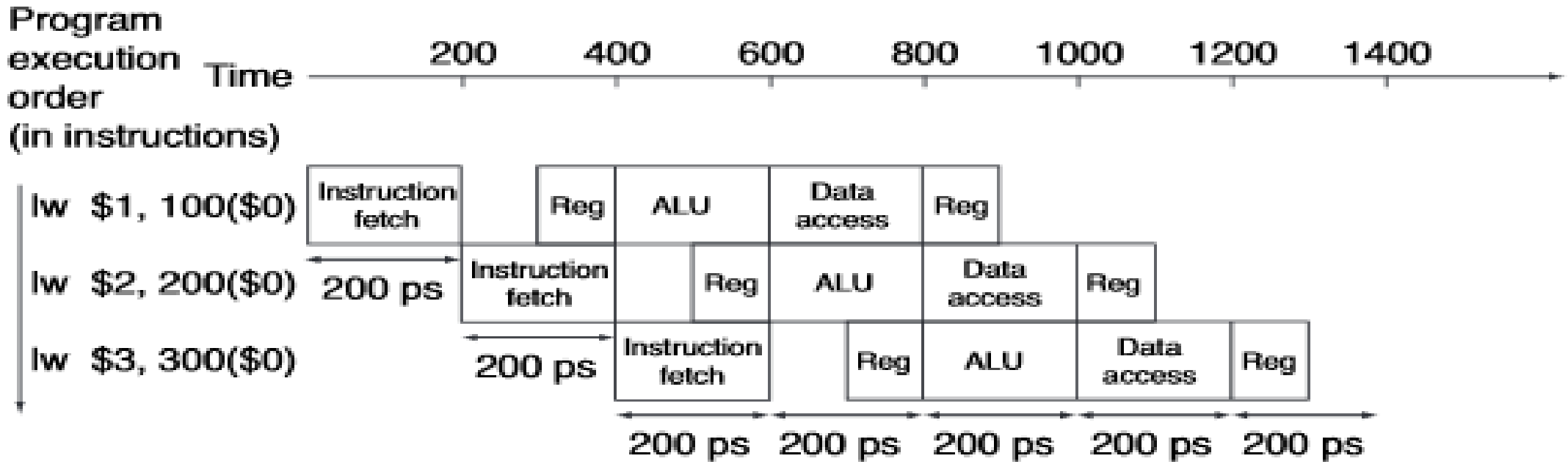
Doors Added

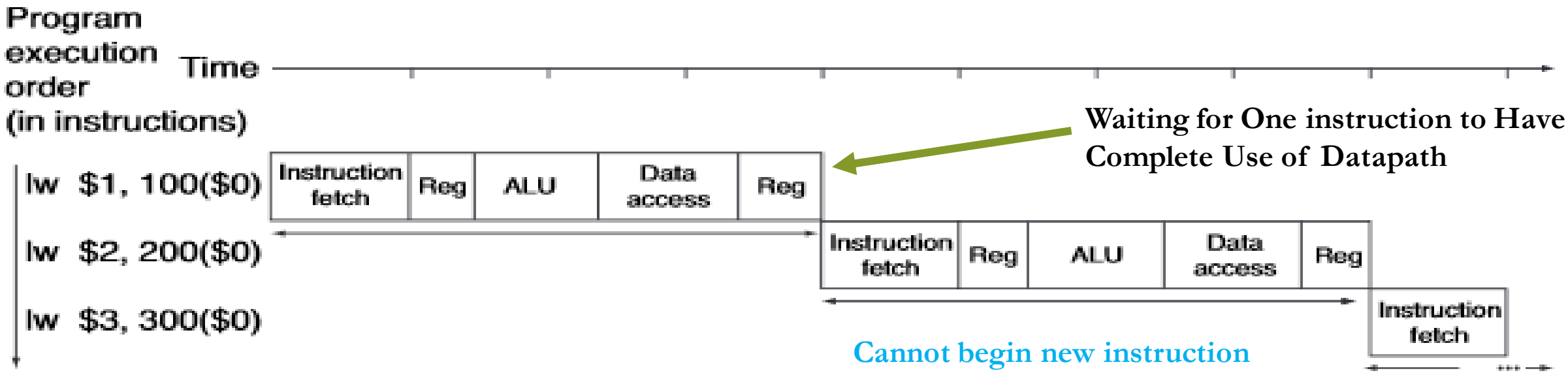
New Roof

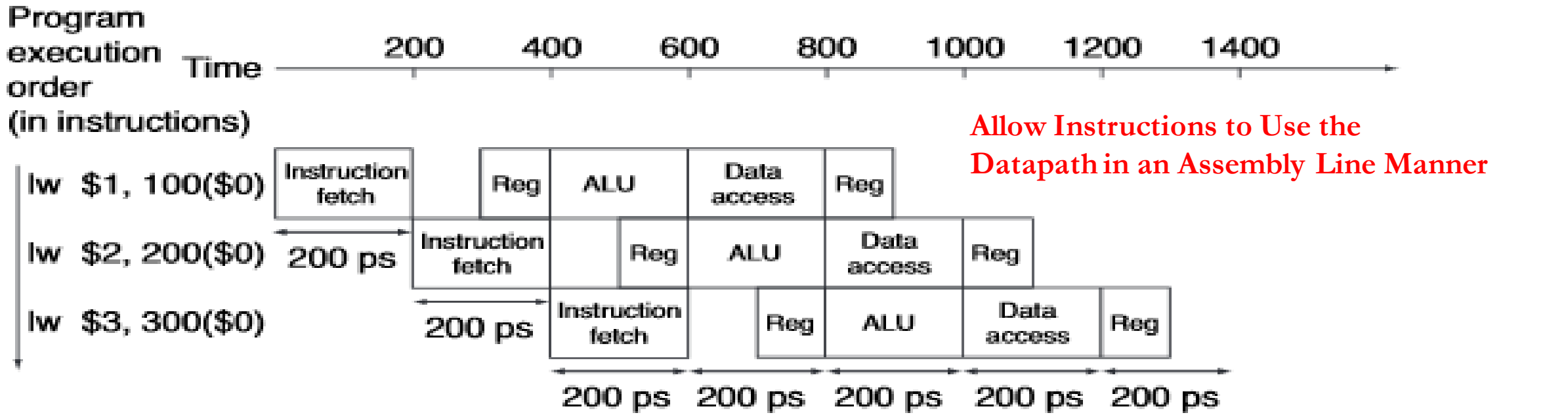
New Tires

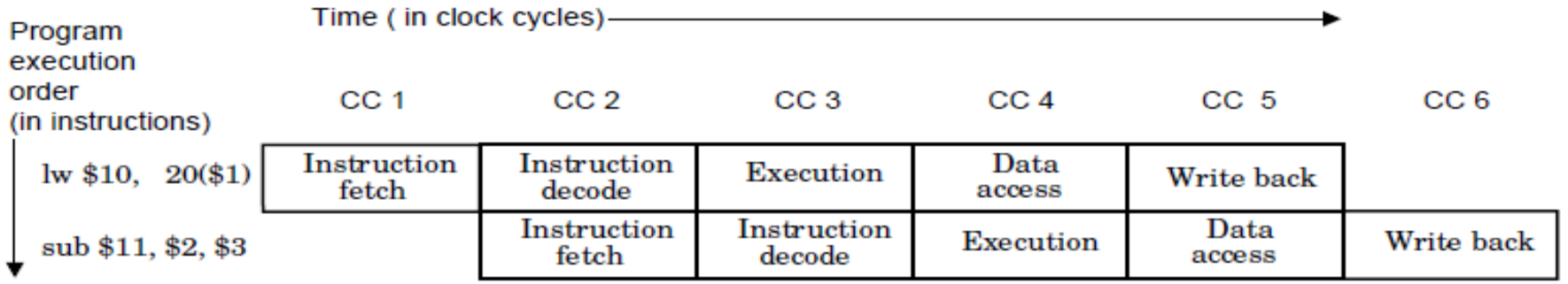
Five Stage Car Assembly Line


Program execution order (in instructions)
Time
lw $1, 100($0)
lw $2, 200($0)
lw $3, 300($0)
Instruction fetch
Reg
ALU
Data access
Reg
Waiting for One instruction to Have Complete Use of Datapath
Cannot begin new instruction
Until First instruction completes


Program execution order (in instructions)
Time
200
400
600
800
1000
1200
1400
lw $1, 100($0)
lw $2, 200($0)
lw $3, 300($0)
Instruction fetch
Reg
ALU
Data access
Reg
200 ps
200 ps
200 ps
200 ps
200 ps
200 ps

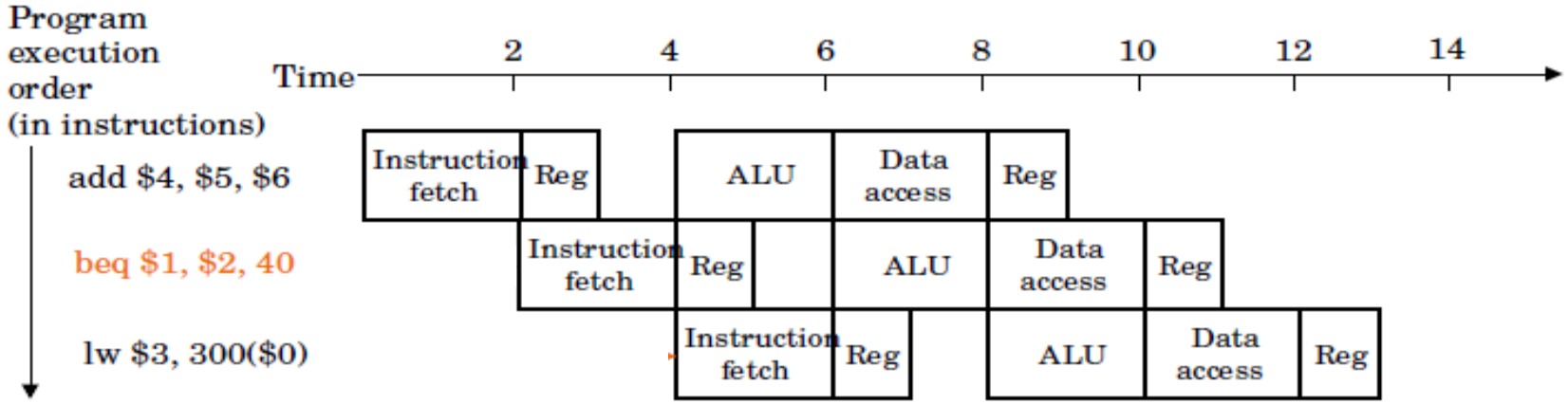

Program execution order (in instructions)
Time
lw $1, 100($0)
lw $2, 200($0)
lw $3, 300($0)
Instruction fetch
Reg
ALU
Data access
Reg
Waiting for One instruction to Have Complete Use of Datapath
Cannot begin new instruction Until First instruction completes


Program execution order (in instructions)
Time
200
400
600
800
1000
1200
1400
lw $1, 100($0)
lw $2, 200($0)
lw $3, 300($0)
Instruction fetch
Reg
ALU
Data access
Reg
200 ps
200 ps
200 ps
200 ps
200 ps
200 ps
Allow Instructions to Use the Datapath in an Assembly Line Manner



Name the Stages

IF ID EX MEM WB



In the Pipeline : **Add** instruction will always be one step ahead of the **Beq** Instruction which will be one step ahead of the **Lw** instruction

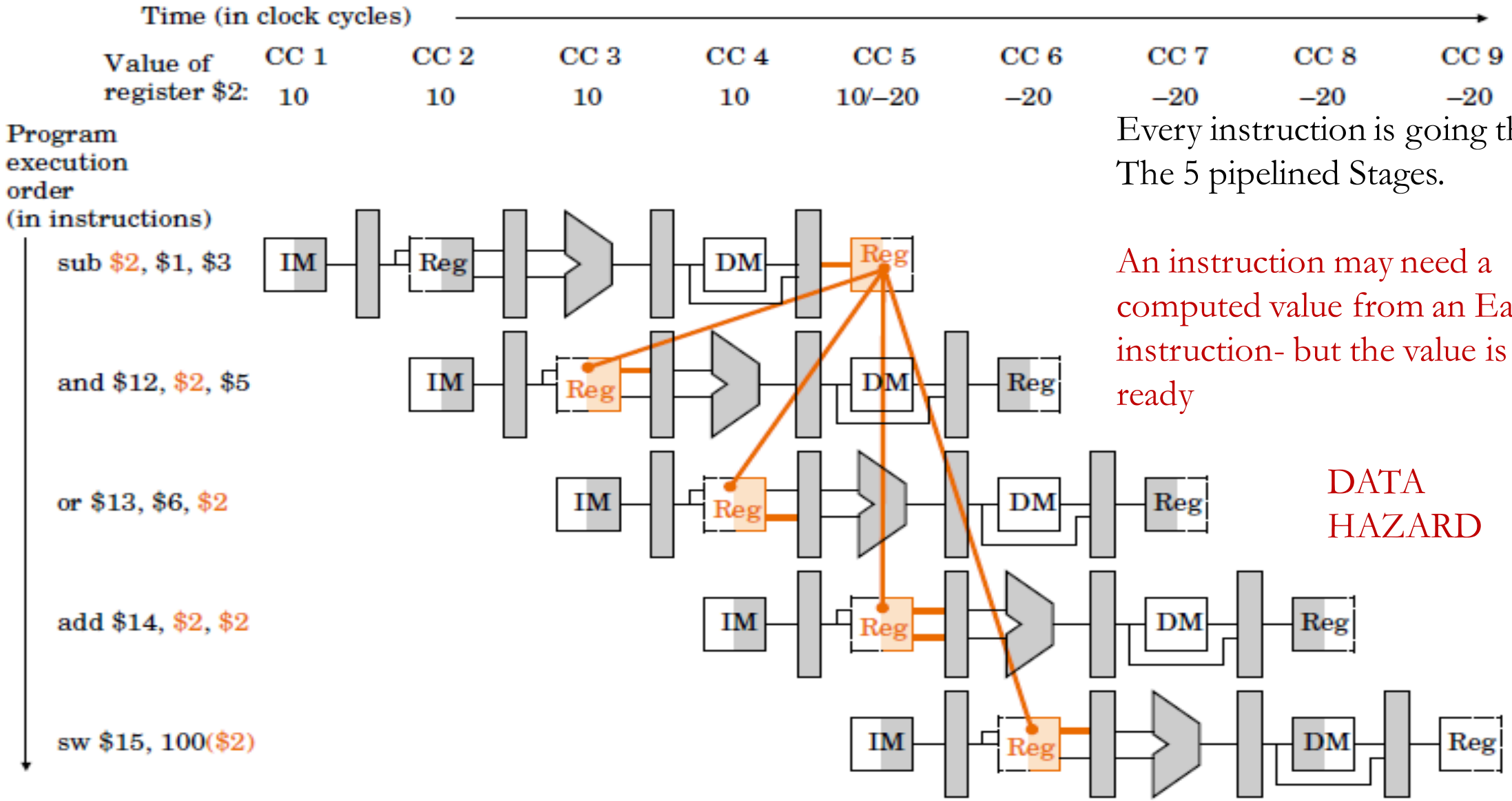
Build Up the Pipelined Datapath

- Will One Memory Unit work?
- Need Temporary Registers to store all the information needed for each instruction.
- ALU Doing All Computations is not going to Work.
- Need to Add back adders into the datapath

Build Up the Pipelined Datapath

- Will One Memory Unit work? **Need to separate Data memory and Instr Memory Again- as it was in Single Cycle**
- Need Temporary Registers to store all the information needed for each instruction.
- ALU Doing All Computations is not going to Work.
- Need to Add back adders into the datapath
- REFER TO COURSE NOTES- Datapath for Pipelining

Data Hazards and Forwarding

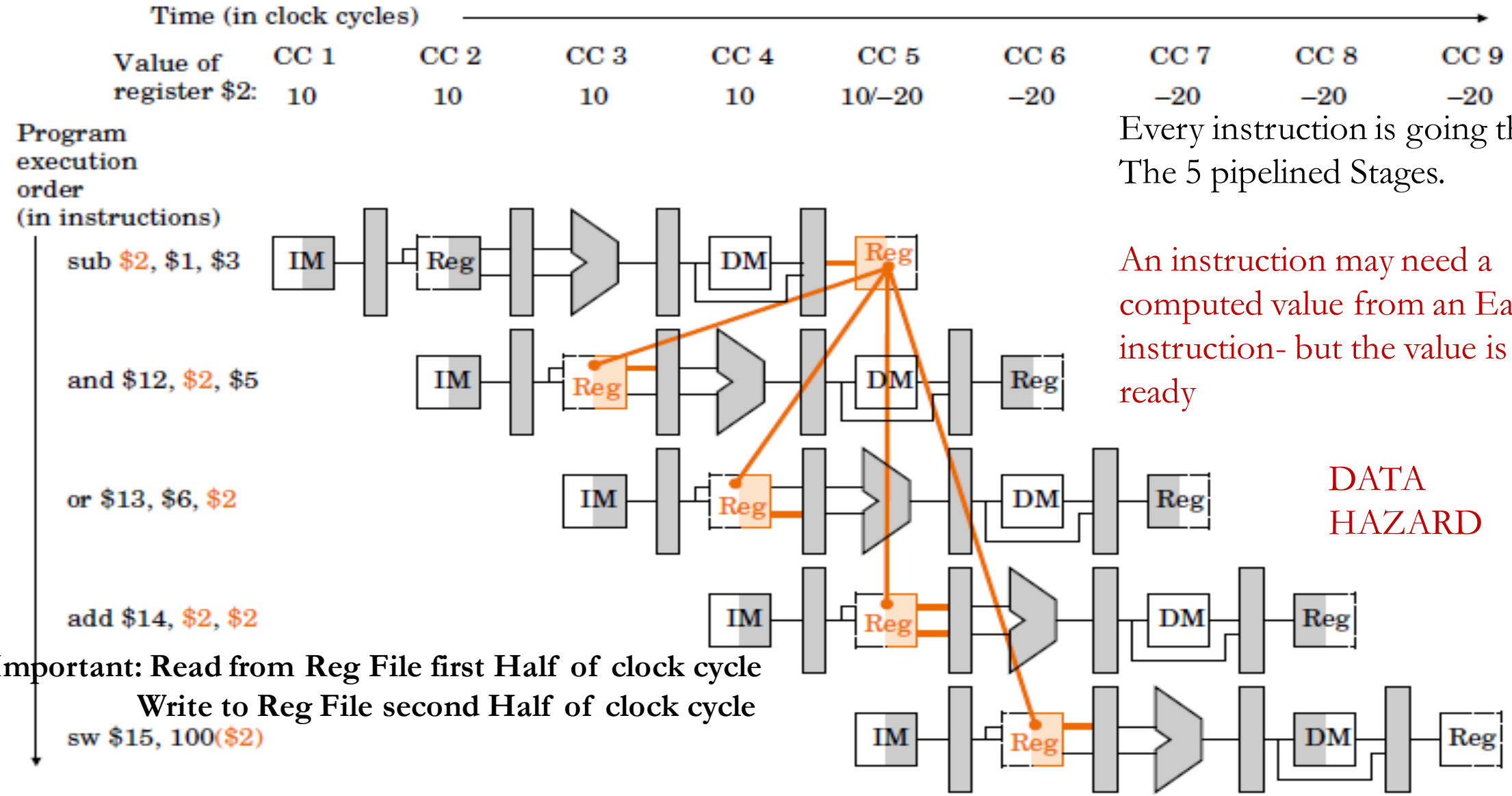


Every instruction is going through The 5 pipelined Stages.

An instruction may need a computed value from an Earlier instruction- but the value is not ready

DATA HAZARD

Data Hazards and Forwarding



Every instruction is going through The 5 pipelined Stages.

An instruction may need a computed value from an Earlier instruction- but the value is not ready

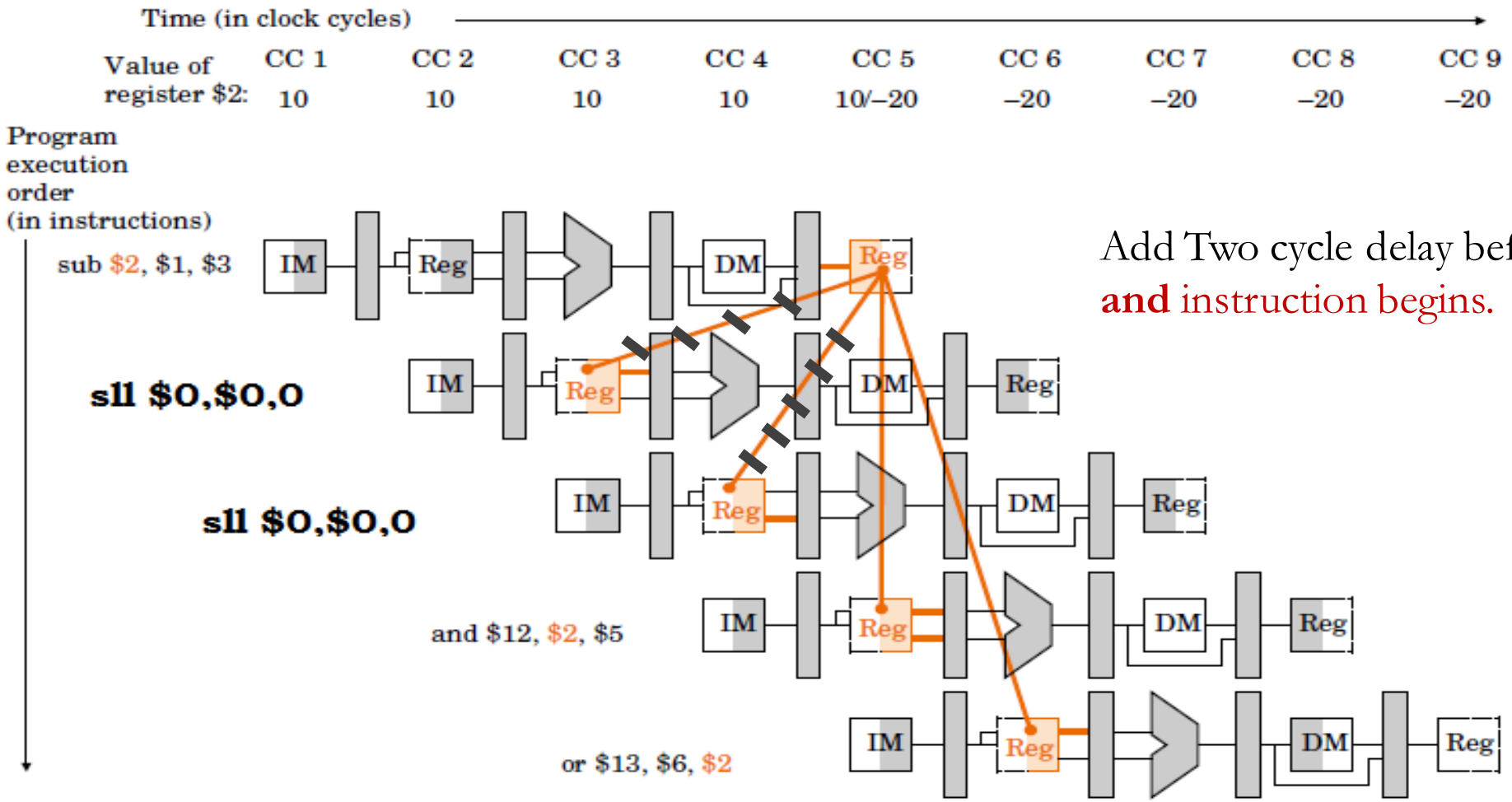
DATA HAZARD

Important: Read from Reg File first Half of clock cycle
Write to Reg File second Half of clock cycle

We will use Data Forwarding (coming soon ☺)

- However, another solution is to add a stall to an instruction that needs a value from a previous instruction
- Stall in the Pipeline
- Use NOPs. Insert an instruction that has no outcome.
- **add $0, $0, $0**
- **sll $0, $0, 0**

Data Hazards and Forwarding



Add Two cycle delay before **and** instruction begins.

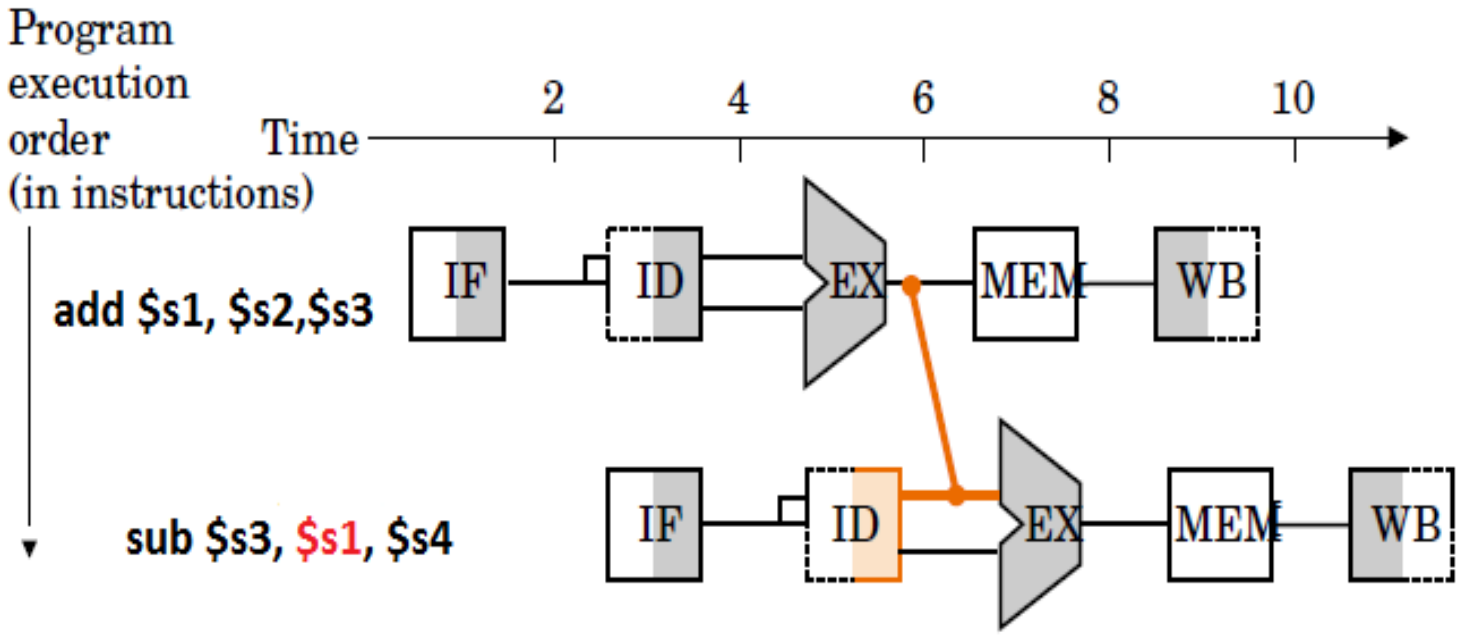
No More Hazard

Pipelining

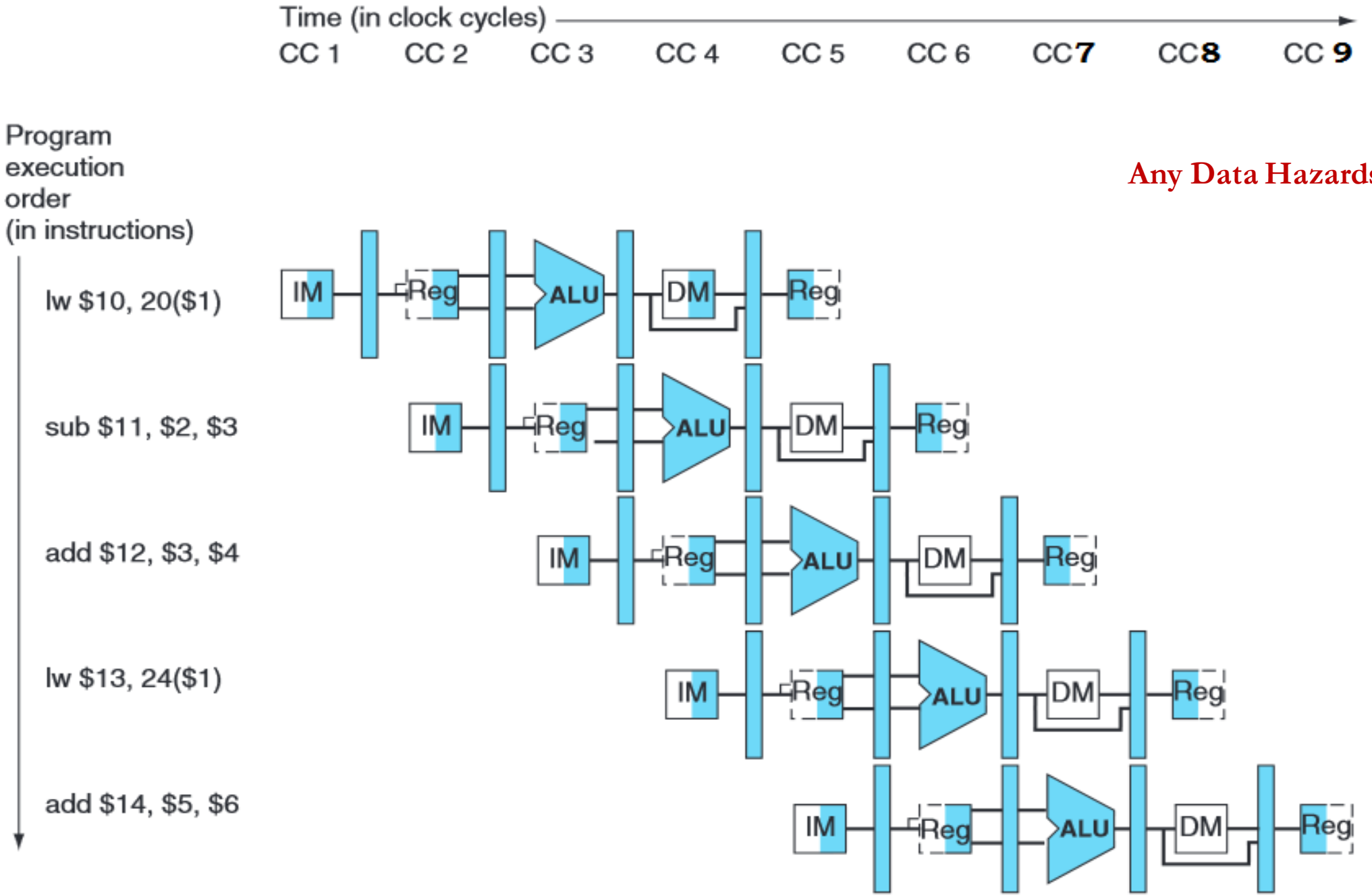
- **The purpose of Pipelining :**

- A)Improve the Thoughput of instructions in the system : ie) more instructions completed in a given time
- B) Improve the individual execution time of One instruction, ie) Add instruction completes in less time
- C) Instructions are ideally going to begin one clock cycle after another
- D) Decreasing the amount of required hardware in the datapath
- E) A and C

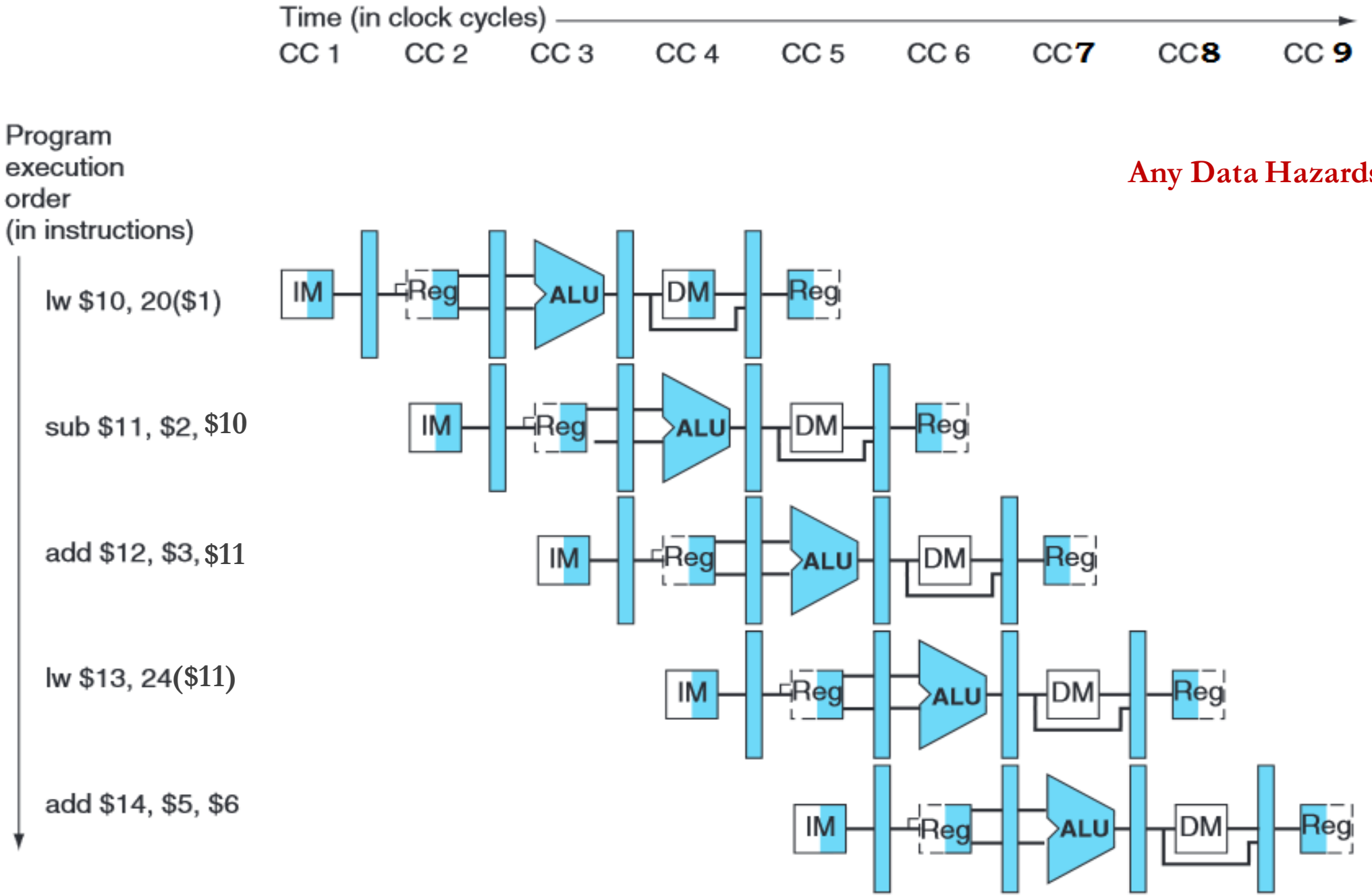
Solving Data Hazards (Overview)



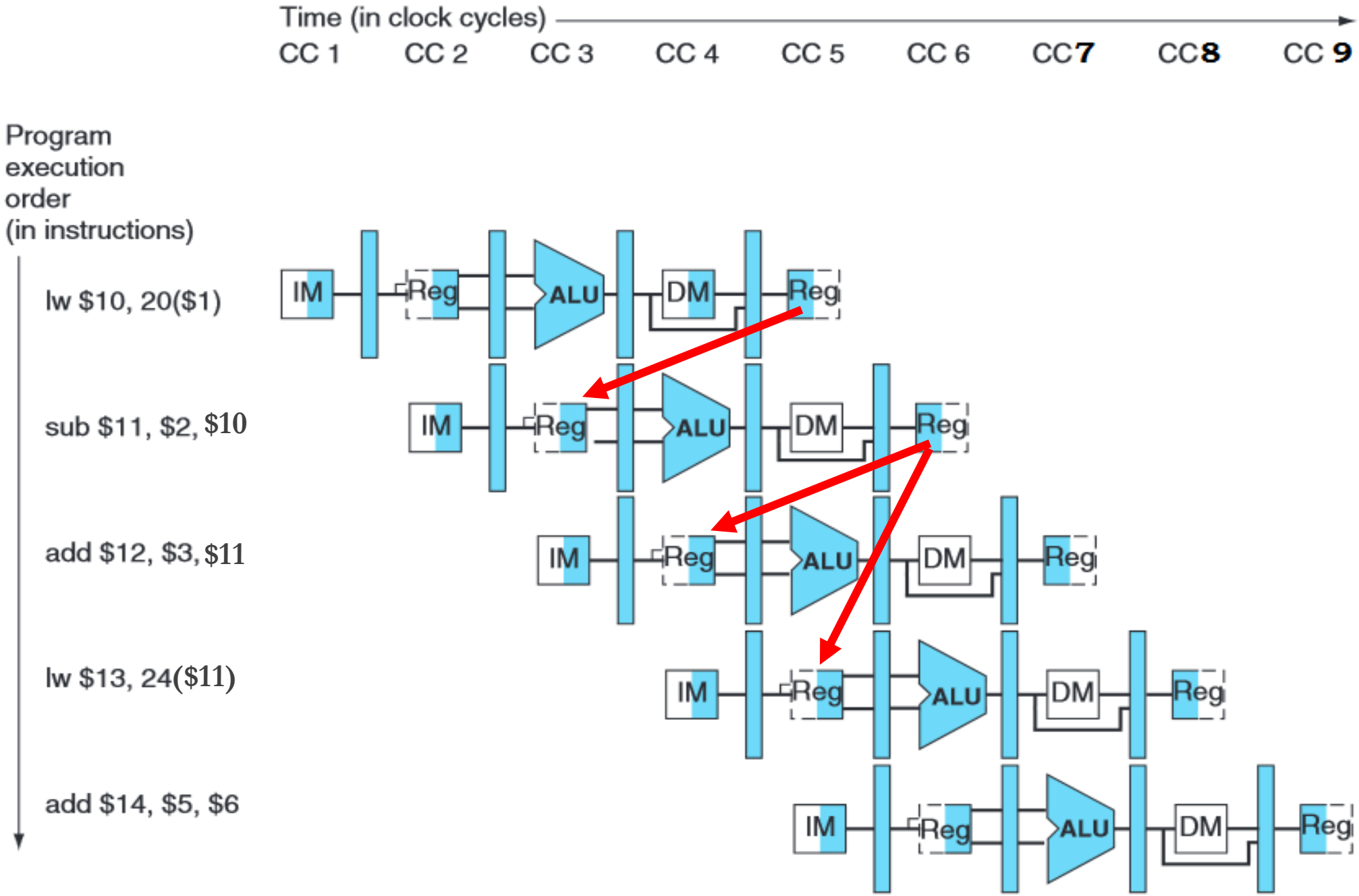
- New value of **$s1** not available from register file in time
- Solution: forwarding – take value as soon as it is ready, before it is written to register file



Any Data Hazards?



Any Data Hazards? NOW


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
lw $10, 20($1)
sub $11, $2, $10
add $12, $3, $11
lw $13, 24($11)
add $14, $5, $6
IM
Reg
ALU
DM
Reg

Data Hazards

- R-format instructions/ any ALU computation can be successfully forwarded

To the instruction immediately following. Thereby avoiding a stall in the pipeline

- LW data coming from memory needs to also be forwarded to subsequent instruction: However one stall cannot be avoided. Load-Use Hazard