Pipelined Architecture

Virendra Singh

Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

CS-683: Advanced Computer Architecture

Lecture 6 (10 Aug 2021)

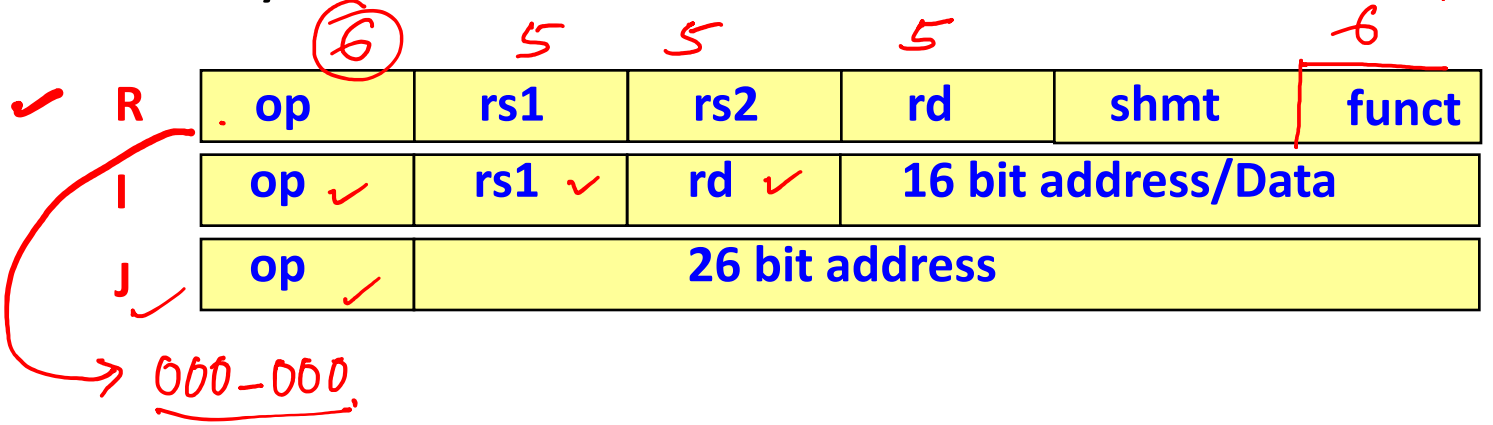
CADSL

RISC Architecture

- Simple instructions
- Fixed Instruction Encoding
- Limited Addressing Mode
- Instruction count increases
- Simple controller
- Load/Store architecture
- ~~Limited addressing modes~~

Overview of MIPS

- Simple instructions, all 32 bits wide
- Very structured, no unnecessary baggage
- Only three instruction formats



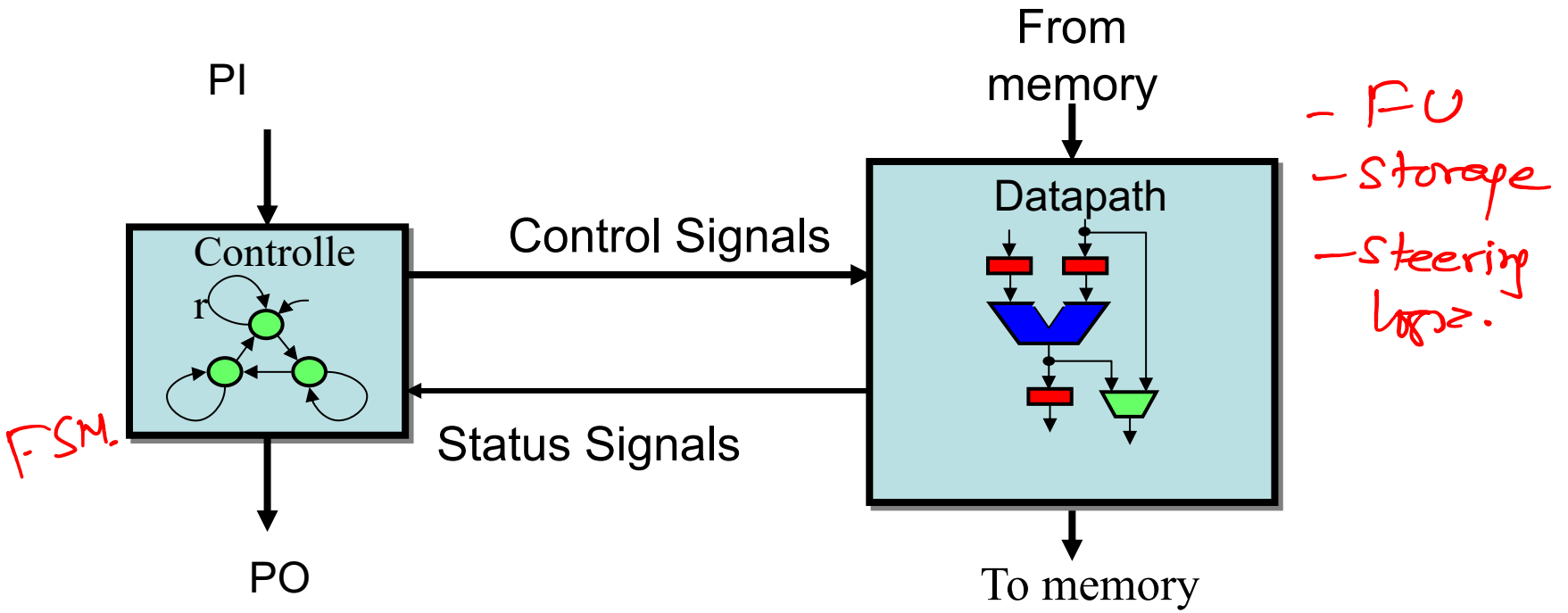
- Rely on compiler to achieve performance

Example processor MIPS subset

MIPS Instruction – Subset

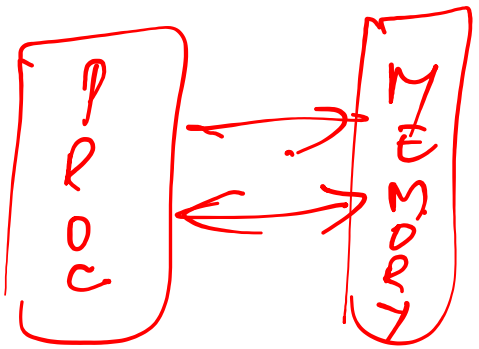
- Arithmetic and Logical Instructions
  - add, sub, or, and, slt
- Memory reference Instructions
  - lw, sw
- Branch
  - beq, j

Processor Architecture



Simple ADD / SUB / MUL

1. fetch an instruction (PC / IP)
2. Understand (Decode)
3. Read operands
4. Execute instruction
5. Update the result



⇒ Can complete in one step

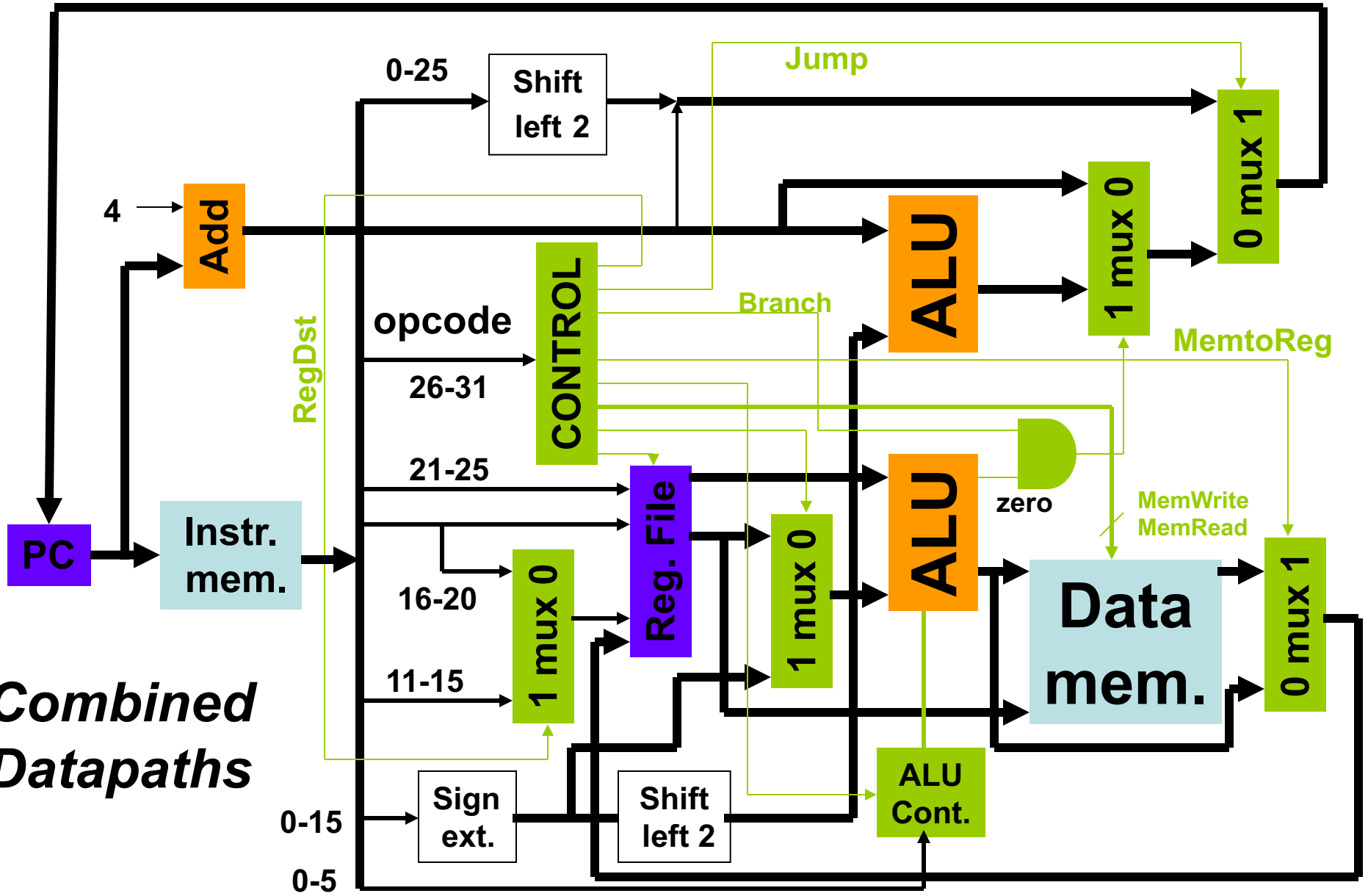
add. r_1, r_2, r_3

$r_1 = r_2 + r_3$

$T = \text{time} = t_f + t_d + t_{or} + t_e + t_u$

Single cycle implementation CPI = 1

$$\frac{\text{Time}}{\text{Prog}} = IC \times \underbrace{CPI \times C}_{1}$$


Jump
0-25
Shift left 2
4
Add
ALU
1 mux 0
0 mux 1
RegDst
opcode
26-31
CONTROL
Branch
MemtoReg
21-25
Reg. File
ALU
zero
MemWrite
MemRead
PC
Instr. mem.
16-20
1 mux 0
1 mux 0
Data mem.
0 mux 1
11-15
Combined Datapaths
Sign ext.
Shift left 2
ALU Cont.
0-15
0-5

Time for Jump (J-Type)

- ALU (R-type) 6ns
- Load word (I-type) 8ns
- Store word (I-type) 7ns
- Branch on equal (I-type) 5ns
- Jump (J-type)
  - Fetch (memory read) 2ns
  - Total 2ns

1. fetch
2. understand (decode)
3. operand read
4. Execute
5. Update the result

⇒ one step → only one sub task.

Multiple steps.

time = max{$t_f, t_d, t_{or}, t_e, t_{ui}$}

CPI > 1

Multi-cycle Implementation

AL

1. fetch 2. decode 3. Op. read. 4. Ex- 5. Updat-] 5

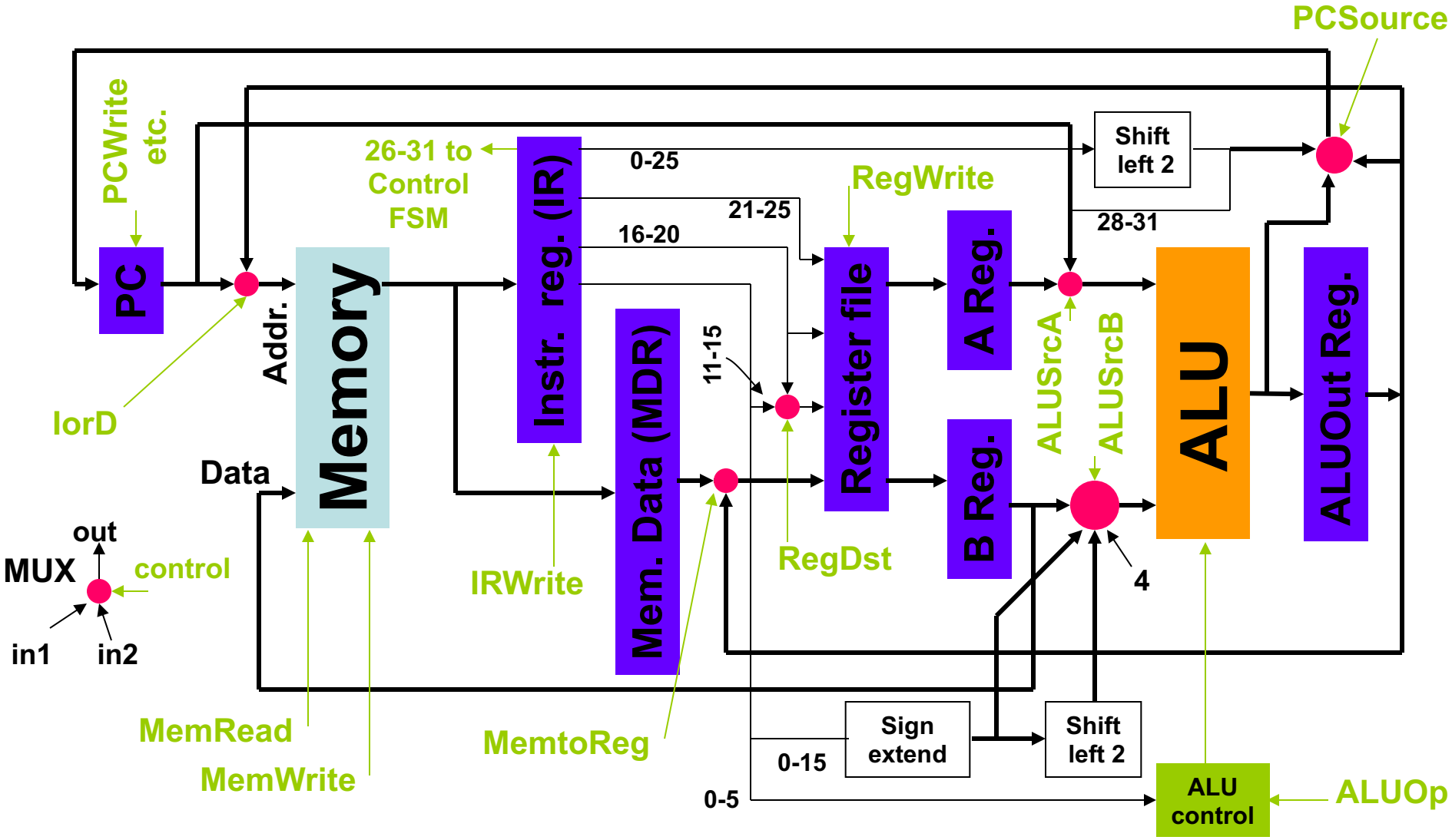
Load

1. fetch, 2. Decode 3. Op. read 4. Add computation, 5. Read memry 6. Updat] 6

Multicycle Instruction Execution

Step	R-type (4 cycles)	Mem. Ref. (4 or 5 cycles)	Branch type (3 cycles)	J-type (3 cycles)
Instruction fetch	IR ← Memory[PC]; PC ← PC+4			
Instr. decode/ Reg. fetch	A ← Reg(IR[21-25]); B ← Reg(IR[16-20]) ALUOut ← PC + (sign extend IR[0-15]) << 2			
Execution, addr. Comp., branch & jump completion	**ALUOut ← A op B**	**ALUOut ← A+sign extend (IR[0-15])**	**If (A= =B) then PC←ALUOut**	**PC←PC[28-31] || (IR[0-25]<<2)**
Mem. Access or R-type completion	**Reg(IR[11-15]) ← ALUOut**	**MDR←M[ALUout] or M[ALUOut]←B**		
Memory read completion		**Reg(IR[16-20]) ← MDR**		

Multicycle Datapath



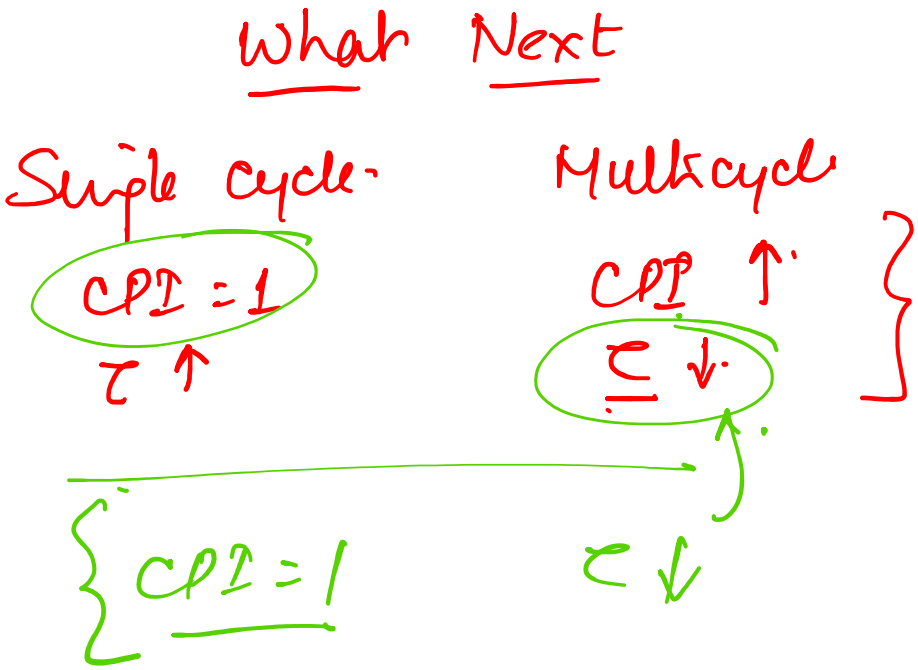
CPI of a Computer

$$CPI = \frac{\sum_k (\text{Instructions of type k}) \times CPI_k}{\sum_k (\text{instructions of type k})}$$

where

CPI_k = Cycles for instruction of type k

Note: CPI is dependent on the instruction mix of the program being run. Standard benchmark programs are used for specifying the performance of CPUs.


What Next
Simple cycle
CPI = 1
τ ↑
Multicycle
CPI ↑
C ↓
CPI = 1
C ↓

Traffic Flow

Single Lane Traffic

ILP: Instruction Level Parallelism

- Single-cycle and multi-cycle datapaths execute one instruction at a time.
- How can we get better performance?
- Answer: Chaining of instructions
- Takes advantage of
  - CPI of single cycle
  - Cycle time of multicycle implementation

Automobile Team Assembly

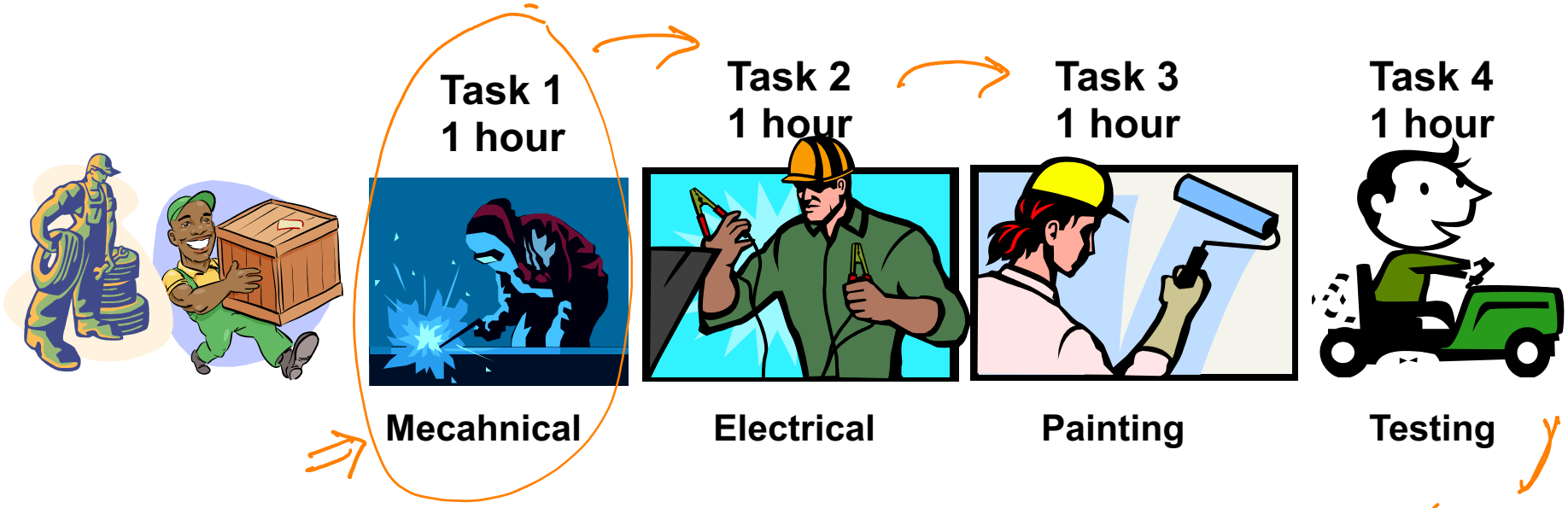
1 car assembled every four hours

6 cars per day

180 cars per month

2,040 cars per year

Automobile Assembly Line



First car assembled in 4 hours (pipeline latency)

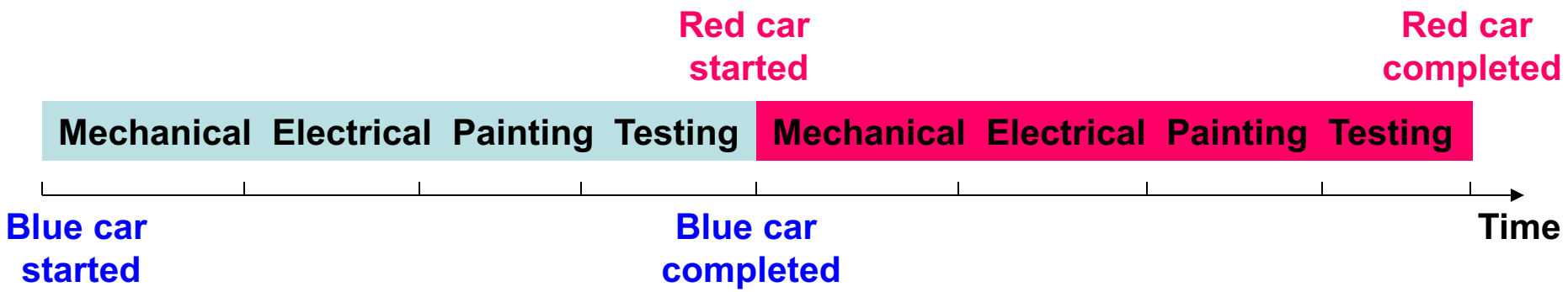
thereafter 1 car per hour

21 cars on first day, thereafter 24 cars per day

717 cars per month

8,637 cars per year

Throughput: Team Assembly

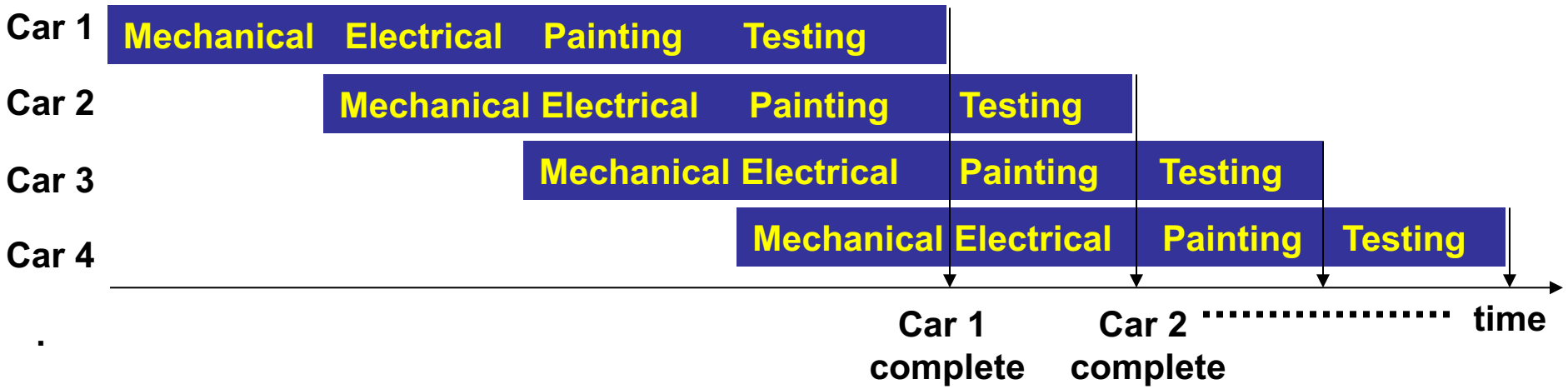


Time of assembling one car = n hours

where n is the number of nearly equal subtasks, each requiring 1 unit of time

Throughput = $1/n$ cars per unit time

Throughput: Assembly Line



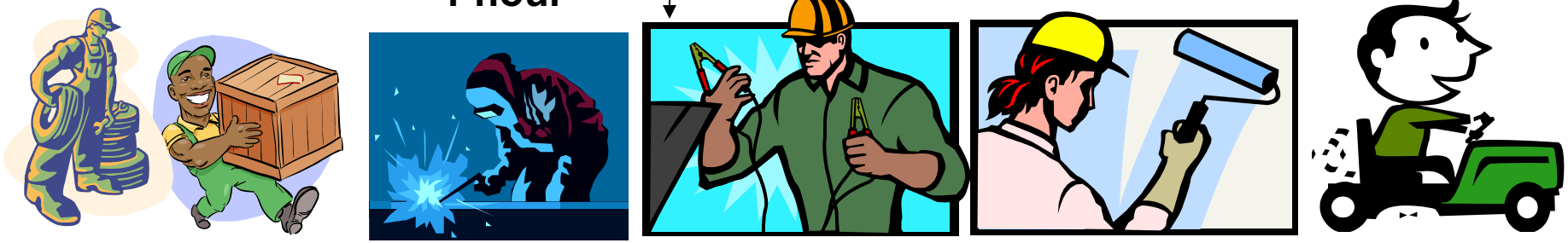
Time to complete first car = n time units (latency)

Cars completed in time T = $T - n + 1$

Throughput = $1 - (n - 1)/T$ car per unit time

$$\frac{\text{Throughput (assembly line)}}{\text{Throughput (team assembly)}} = \frac{1 - (n-1)/T}{1/n} = n - \frac{n(n-1)}{T} \rightarrow n \text{ as } T \rightarrow \infty$$

Some Features of Assembly Line



Stall assembly line to fix the cause of defect

3 cars in the assembly line are suspects, to be removed (flush pipeline)

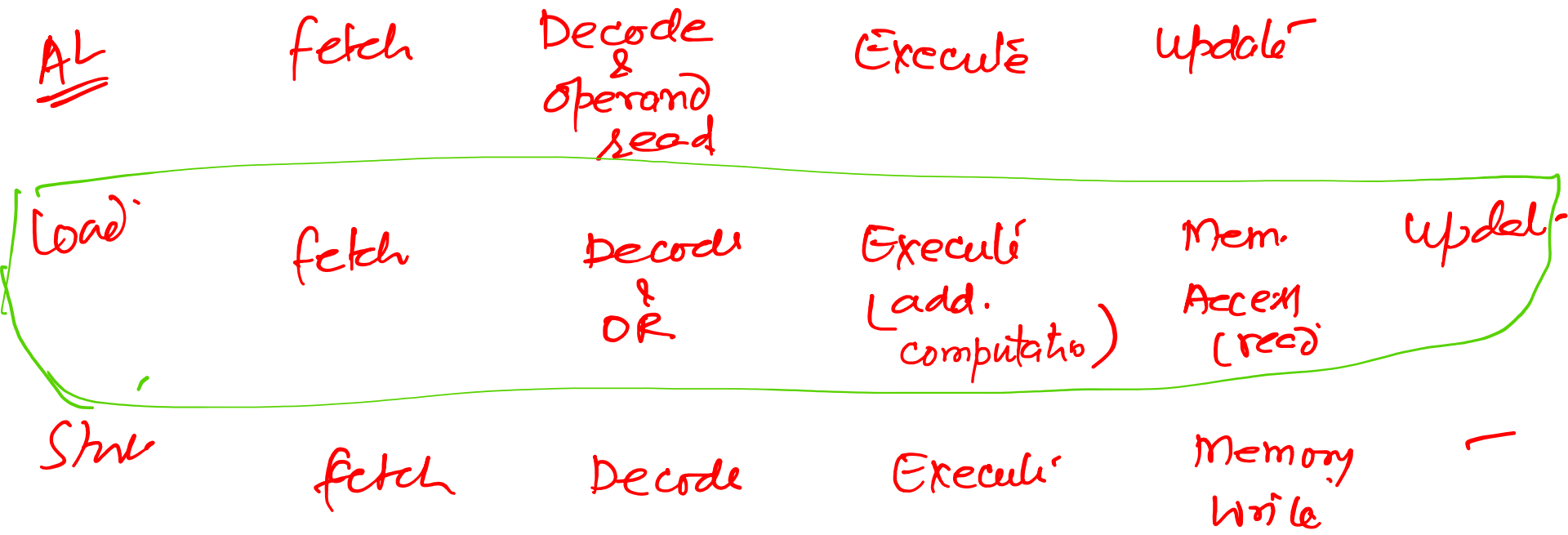
Defect found

Pipelining in a Computer

- Divide datapath into nearly equal tasks, to be performed serially and requiring non-overlapping resources.
- Insert registers at task boundaries in the datapath; registers pass the output data from one task as input data to the next task.
- Synchronize tasks with a clock having a cycle time that just exceeds the time required by the longest task.
- Break each instruction down into a fixed number of tasks so that instructions can be executed in a staggered fashion.

SUPER SET

fetch, understand (Decode), Operand read, Execute, Update rw result

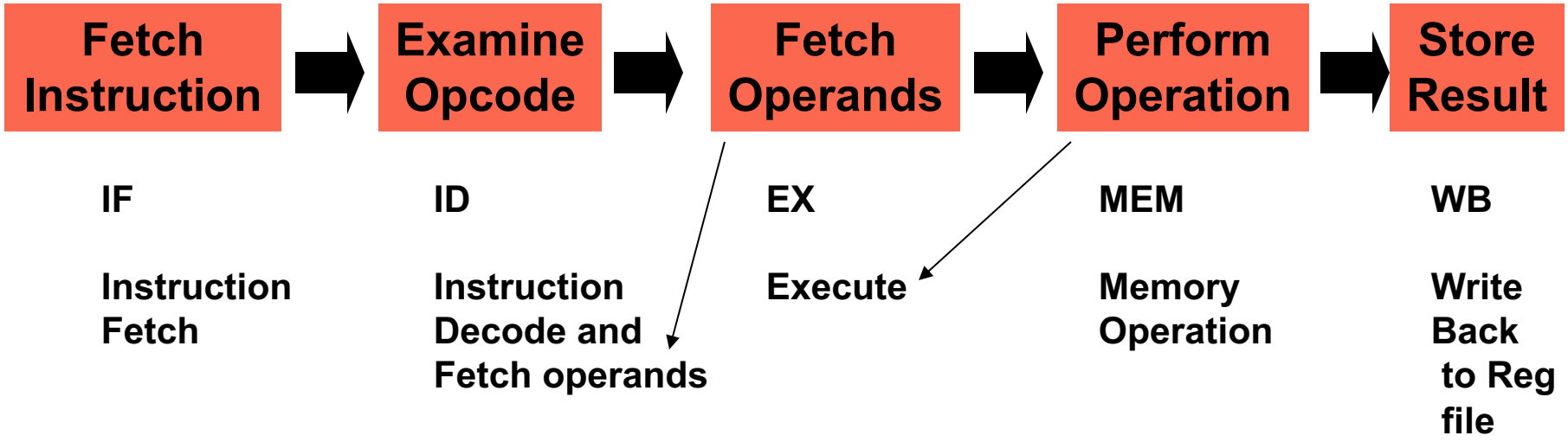

AL
fetch
Decode & Operand read
Execute
Update
Load
fetch
Decode & OR
Execute (add. computation)
Mem. Access (read
Update
Store
fetch
Decode
Execute
Memory Write

Pipelined Datapath

Instruction class	Instr. fetch (IF)	Instr. Decode (also reg. file read) (ID)	Execu-tion (ALU Opera-tion) (EX)	Data access (MEM)	Write Back (Reg. file write) (WB)	Total time
lw	2ns	~~1ns~~ 2ns	2ns	2ns	~~1ns~~ 2ns	10ns
sw	2ns	~~1ns~~ 2ns	2ns	2ns	~~1ns~~ 2ns	10ns
R-format: add, sub, and, or, slt	2ns	~~1ns~~ 2ns	2ns	2ns	~~1ns~~ 2ns	10ns
B-format: beq	2ns	~~1ns~~ 2ns	2ns	2ns	~~1ns~~ 2ns	10ns

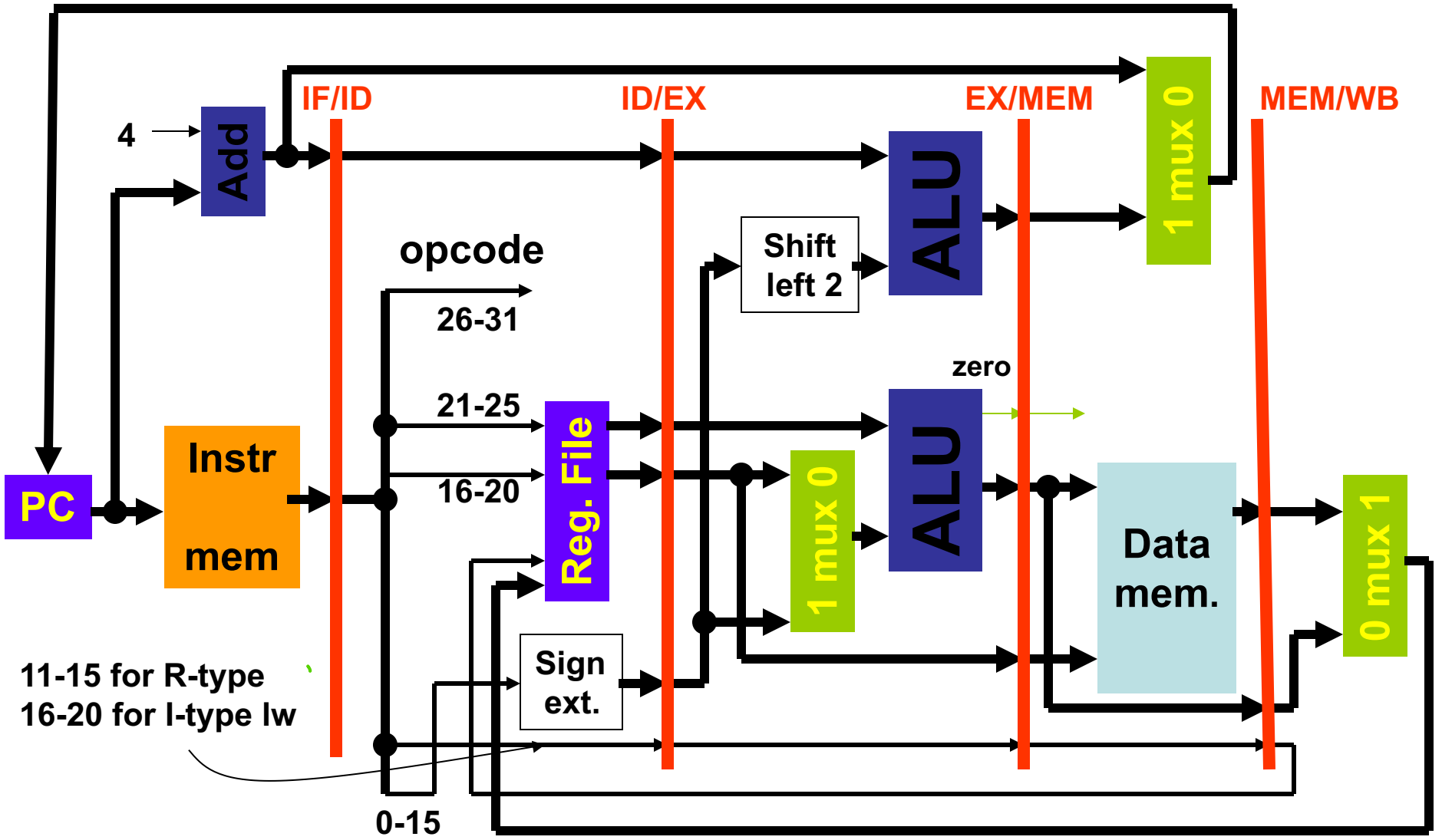
No operation on data; idle time inserted to equalize instruction lengths.

Pipelining of RISC Instructions

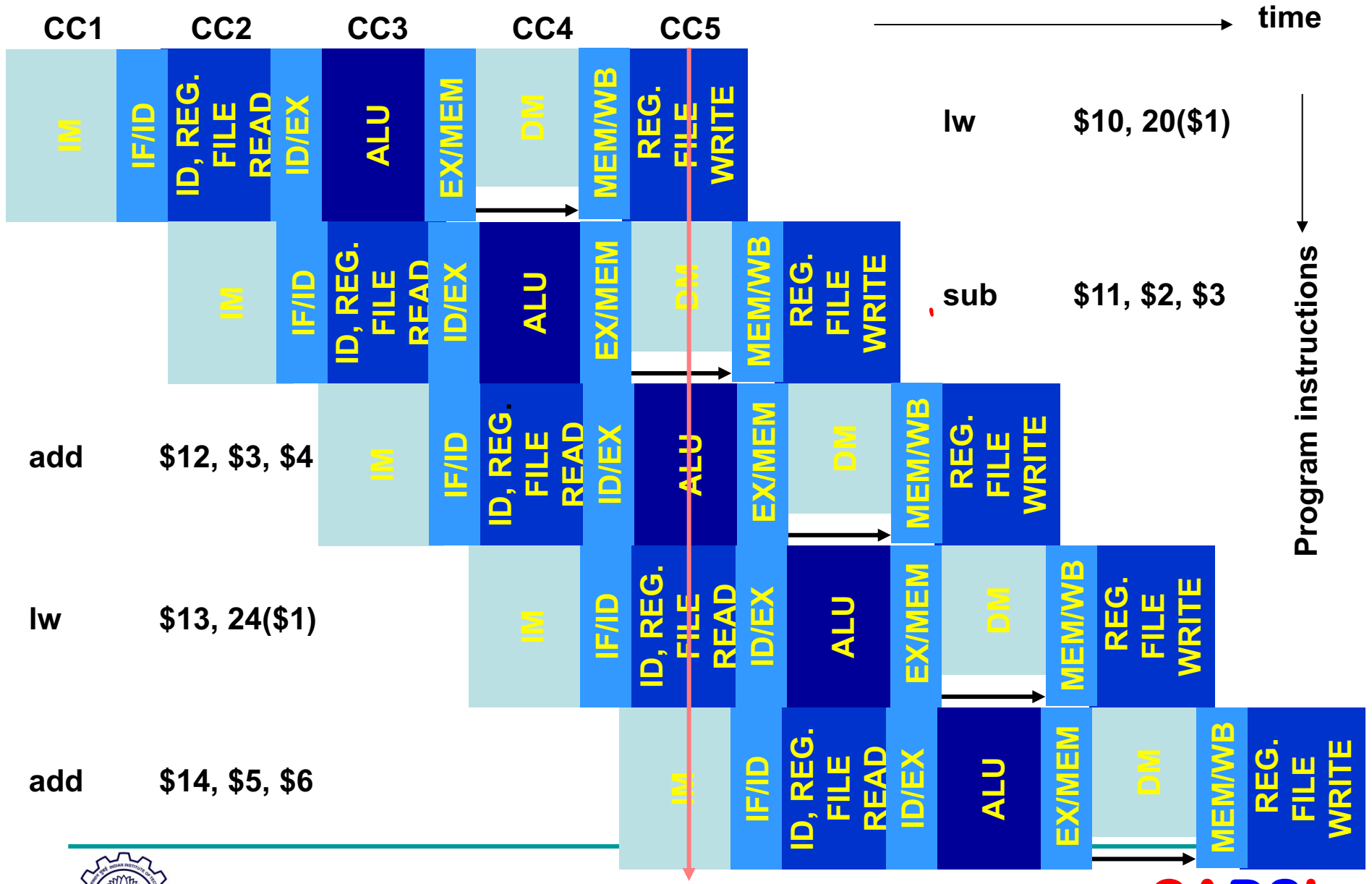


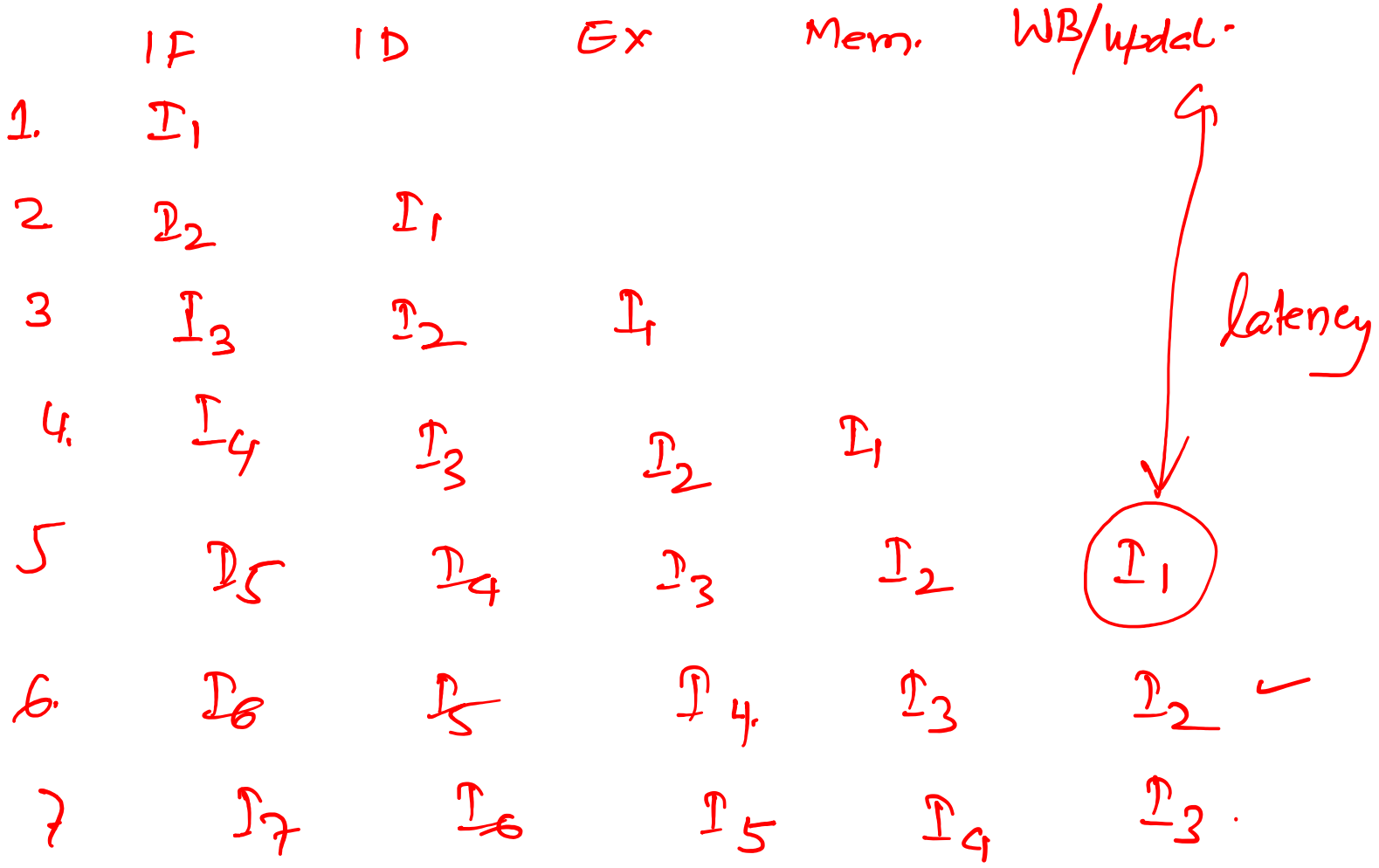
Although an instruction takes five clock cycles, one instruction is completed every cycle.

Pipelined Datapath



Program Execution




IF
ID
Ex
Mem.
WB/update
1. I_1
2 I_2 I_1
3 I_3 I_2 I_1
4. I_4 I_3 I_2 I_1
5 I_5 I_4 I_3 I_2 I_1
6. I_6 I_5 I_4 I_3 I_2
7 I_7 I_6 I_5 I_4 I_3
latency

Assumption

1. All the instructions are independent
2. Follow straight code.
   - No branch. (No change in control flow)

I_1
I_2
I_3
I_4
I_5
I_6
I_7

REACTIVE
- Detect
- Correct

Hazards

CPI = 1
C: ↓

Advantages of Pipeline

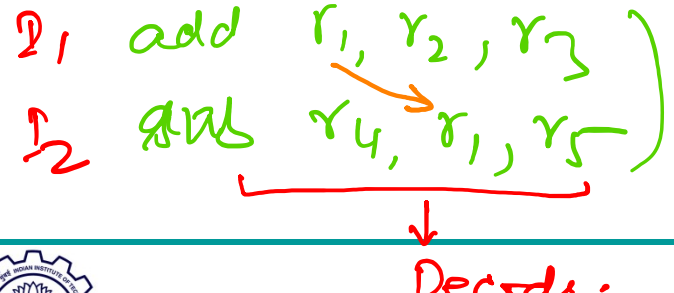
- After the fifth cycle (CC5), one instruction is completed each cycle; CPI ≈ 1, neglecting the initial pipeline latency of 5 cycles.
  - *Pipeline latency is defined as the number of stages in the pipeline, or*
  - *The number of clock cycles after which the first instruction is completed.*
- The clock cycle time is about four times shorter than that of single-cycle datapath and about the same as that of multicycle datapath.
- For multicycle datapath, CPI = 3. ….
- So, pipelined execution is faster, but . . .

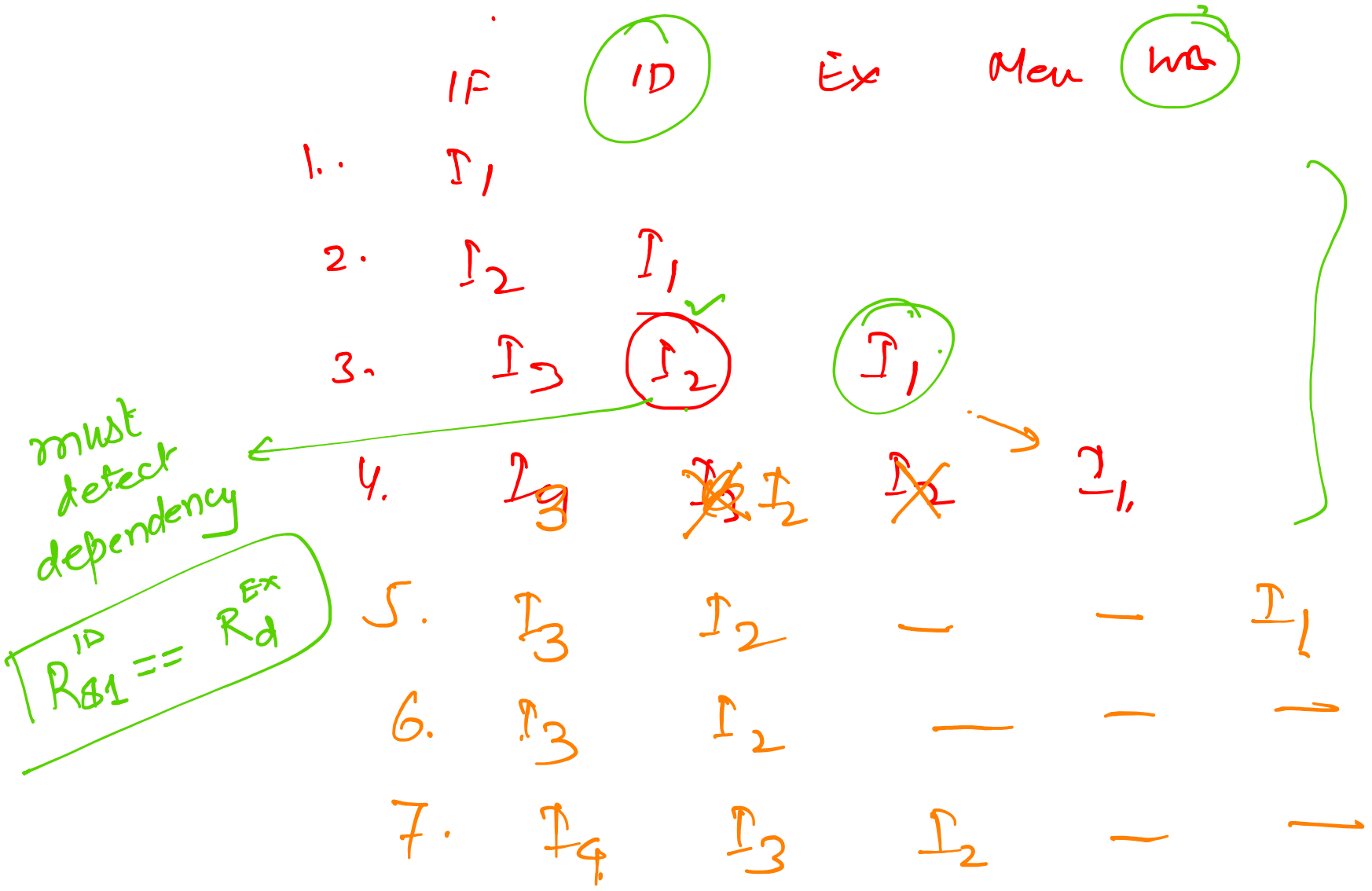
Pipeline Hazards

- Definition: *Hazard in a pipeline is a situation in which the next instruction cannot complete execution one clock cycle after completion of the present instruction.*
- Three types of hazards:
  - Structural hazard (resource conflict)
  - Data hazard
  - Control hazard

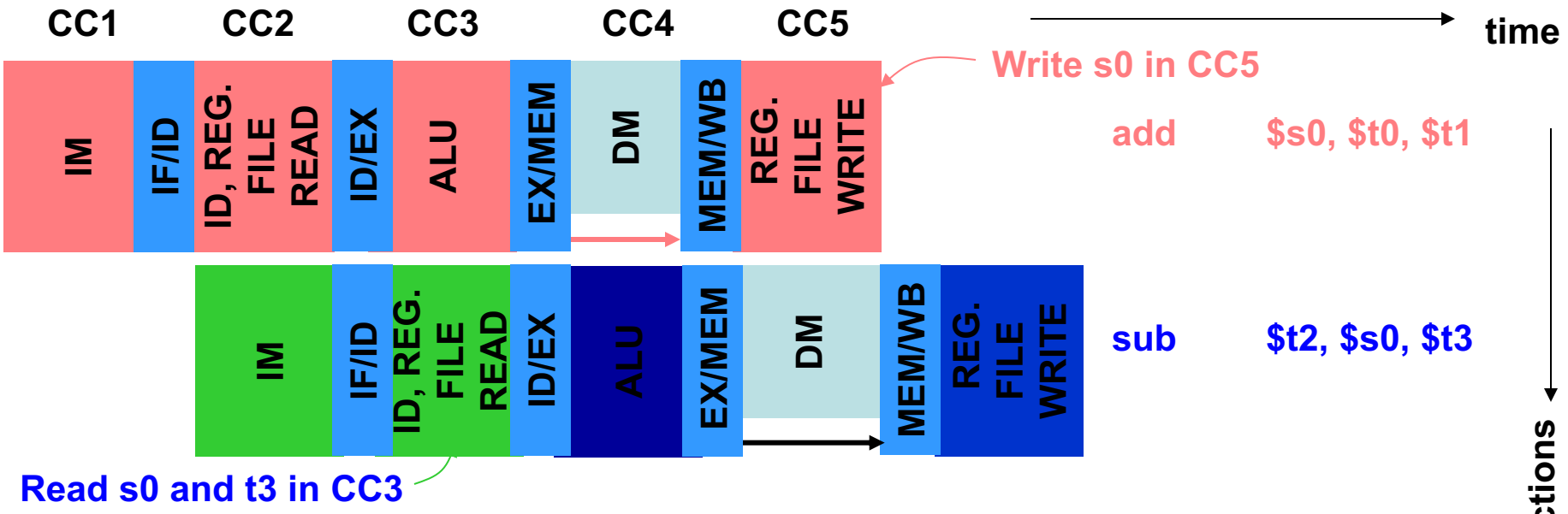
Data Hazard

- Data hazard means that an instruction cannot be completed because the needed data, to be generated by another instruction in the pipeline, is not available.
- Example: consider two instructions:
  - ✧ add $s0, $t0, $t1
  - ✧ sub $t2, $s0, $t3 # needs $s0




IF
ID
Ex
Mem
WB
1. I1
2. I2 I1
3. I3 I2 I1
4. I4 I3 I2 I1
5. I5 I4 I3 I2 I1
6. I6 I5 I4 I3 I2
7. I7 I6 I5 I4 I3
must detect dependency
R_d^EX == R_s1^ID

Example of Data Hazard



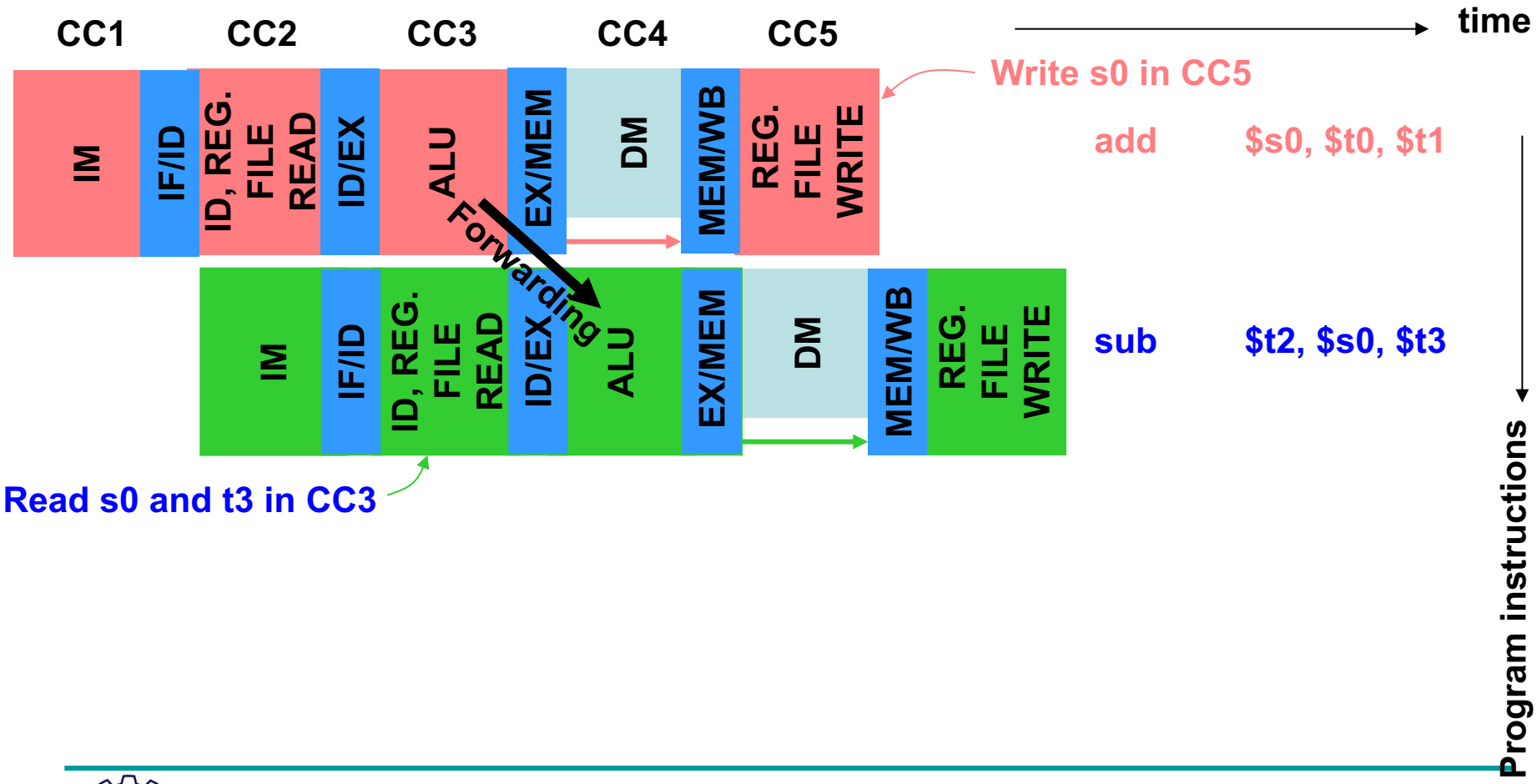
We need to read s0 from reg file in cycle 3
But s0 will not be written in reg file until cycle 5

However, s0 will only be used in cycle 4
And it is available at the end of cycle 3

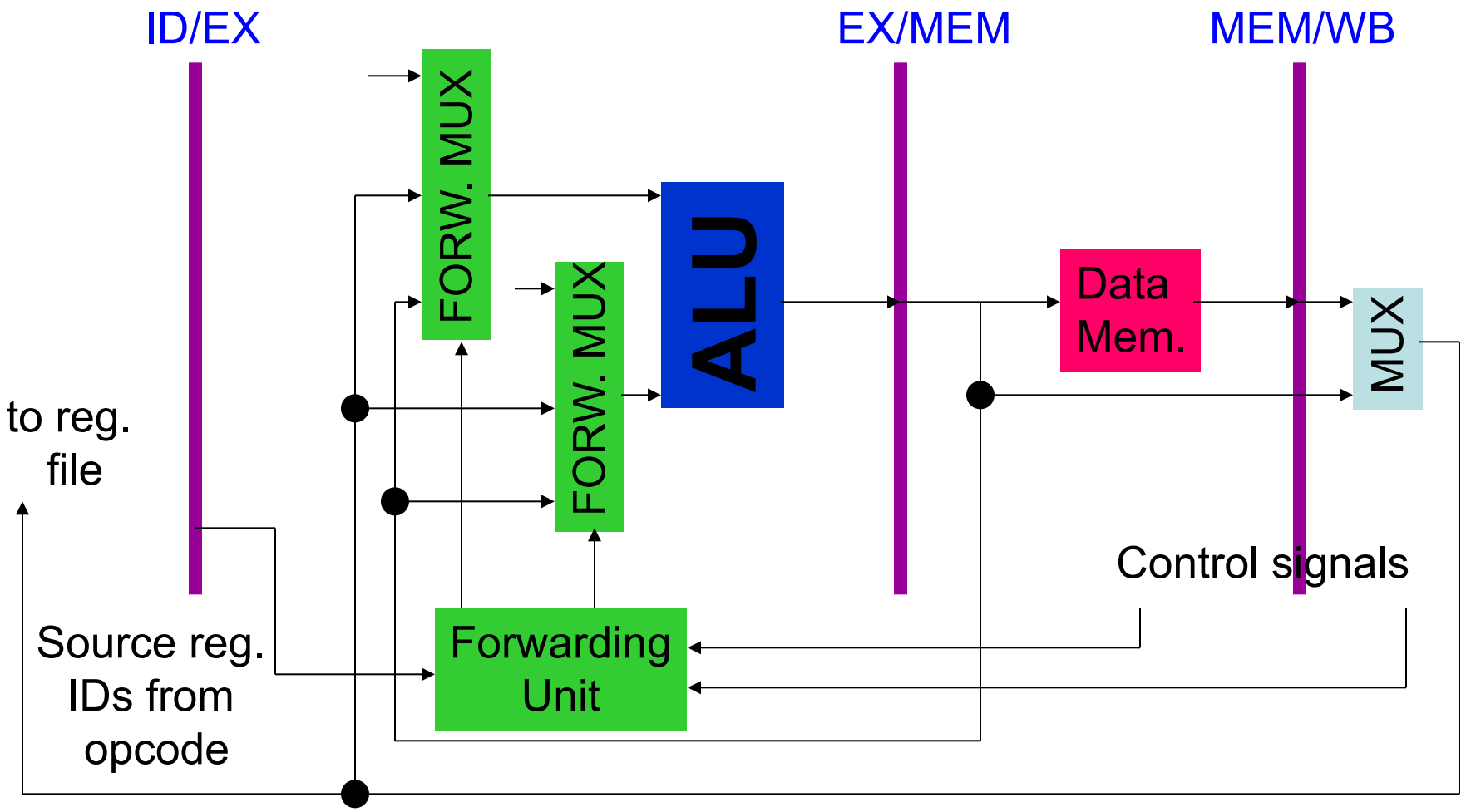
Forwarding or Bypassing

- Output of a resource used by an instruction is forwarded to the input of some resource being used by another instruction.
- Forwarding can eliminate some, but not all, data hazards.

Forwarding for Data Hazard



Forwarding Unit Hardware



Thank You