Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

3 & 5 Stage ARM Processor

Dr. D. C. Kiran
Department of Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

- 3-Stage ARM Processor
- 5-Stage Pipeline Processor
- Introduction to Pipeline Processor

Unit 3: **Memory Design**
Unit 4: **Input/Output Device Design**
Unit 5: **Advanced Architecture**

Microprocessor & Computer Architecture (µpCA)

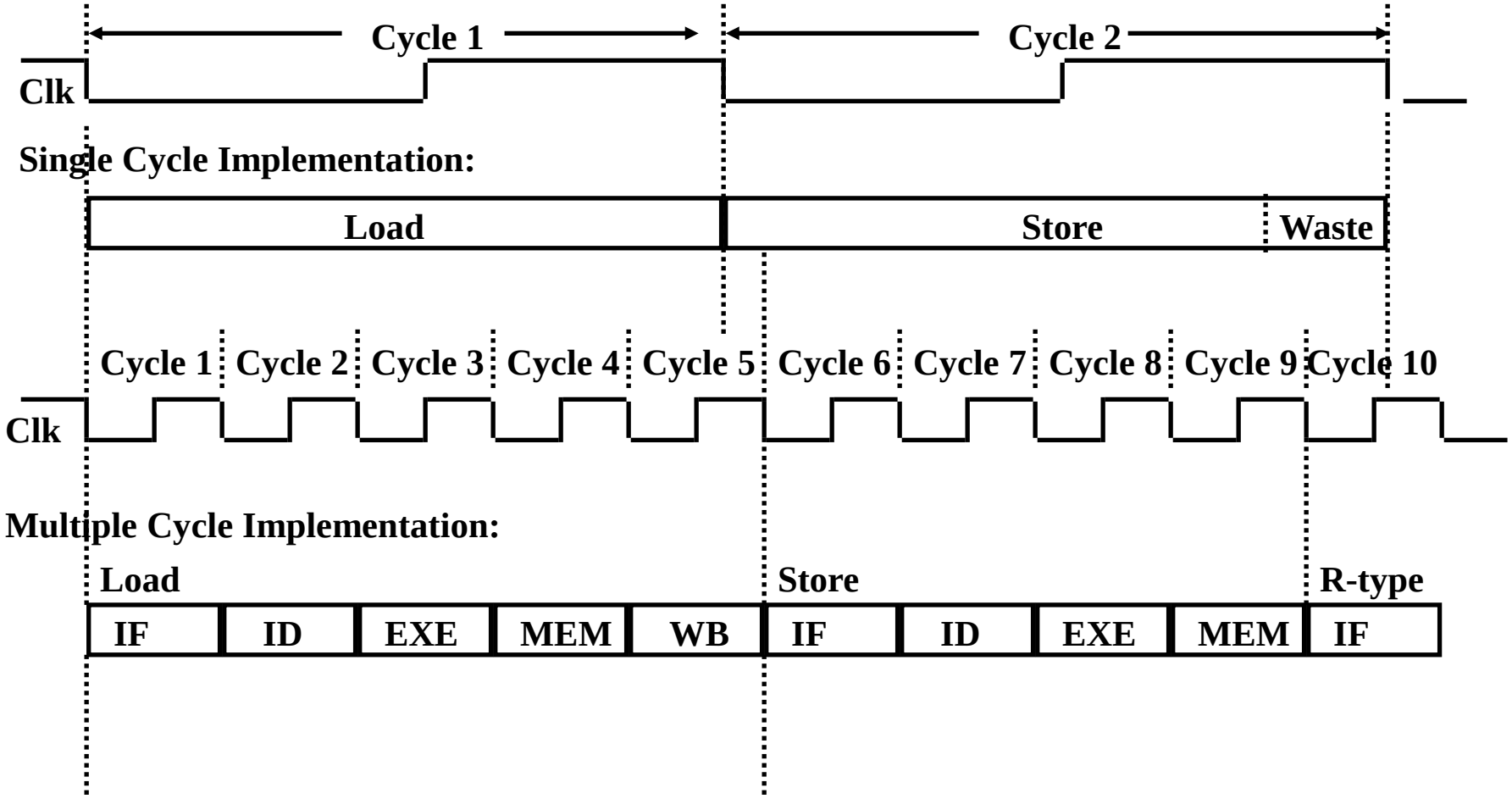
Text 1: "Computer Organization and Design", Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

Reference 1:"Computer Architecture: A Quantitative Approach", Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.

Appendix C **Pipelining: Basic and Intermediate Concepts**

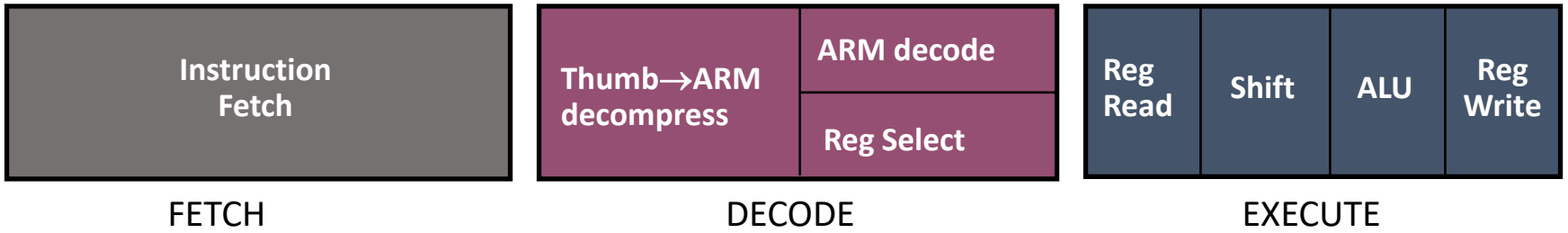
C.1	Introduction	C-2
C.2	The Major Hurdle of Pipelining—Pipeline Hazards	C-11
C.3	How Is Pipelining Implemented?	C-30
C.4	What Makes Pipelining Hard to Implement?	C-43
C.5	Extending the MIPS Pipeline to Handle Multicycle Operations	C-51
C.6	Putting It All Together: The MIPS R4000 Pipeline	C-61
C.7	Crosscutting Issues	C-70
C.8	Fallacies and Pitfalls	C-80
C.9	Concluding Remarks	C-81
C.10	Historical Perspective and References	C-81
	Updated Exercises by Diana Franklin	C-82

Recall → Single Cycle vs Multiple Cycle

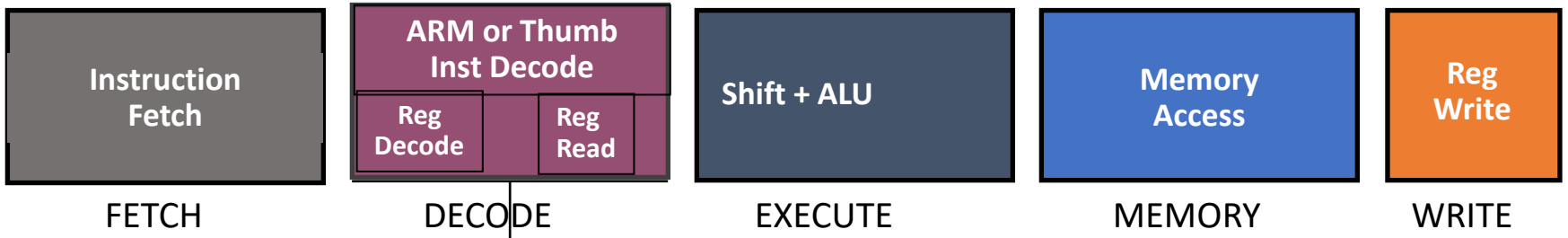


ARM Processors

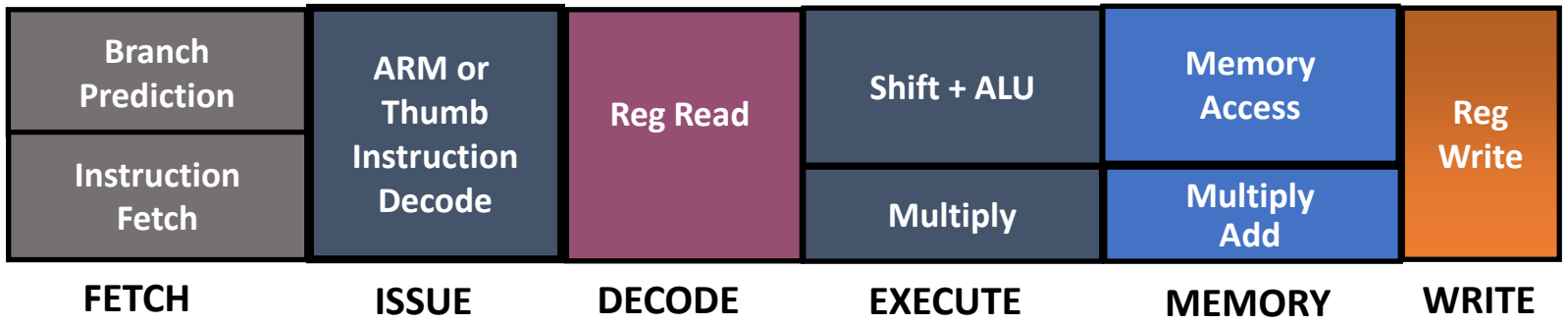
ARM7TDMI



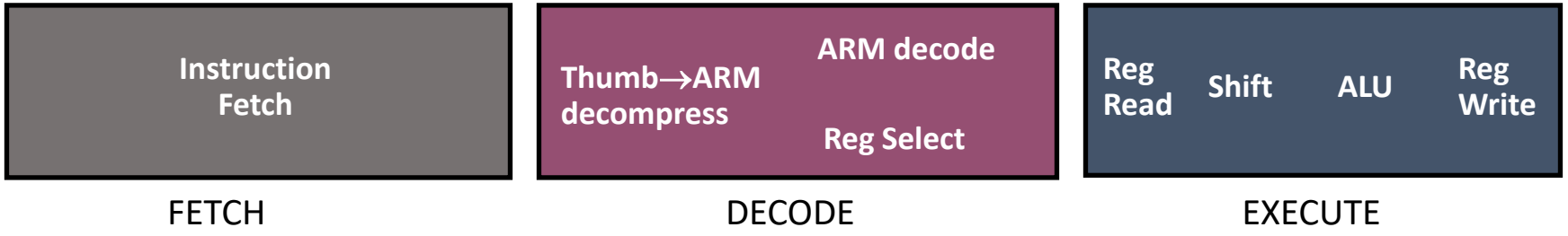
ARM9TDMI



ARM10



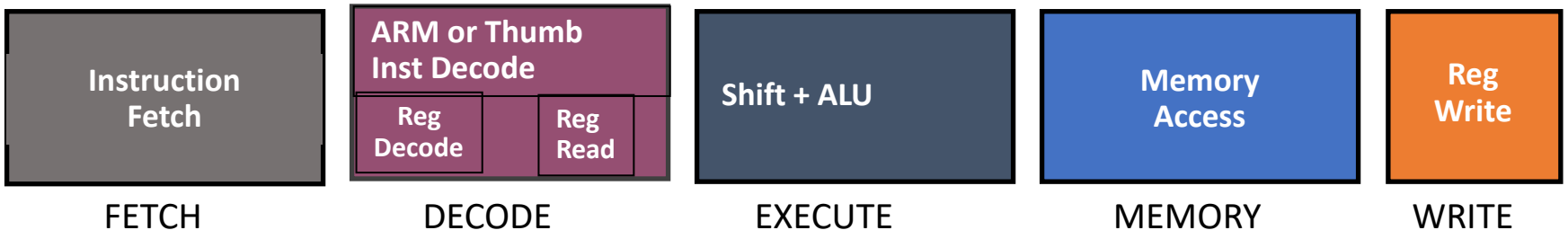
ARM7TDMI – 3 Stage



- **Fetch (IF)**
  - The instruction is fetched from memory and placed in the instruction pipeline
- **Decode (ID)**
  - The instruction is decoded and the datapath control signals prepared for the next cycle.
- **Execute (EX)**
  - The register bank is read, an operand shifted, the ALU result generated and written back into a destination register

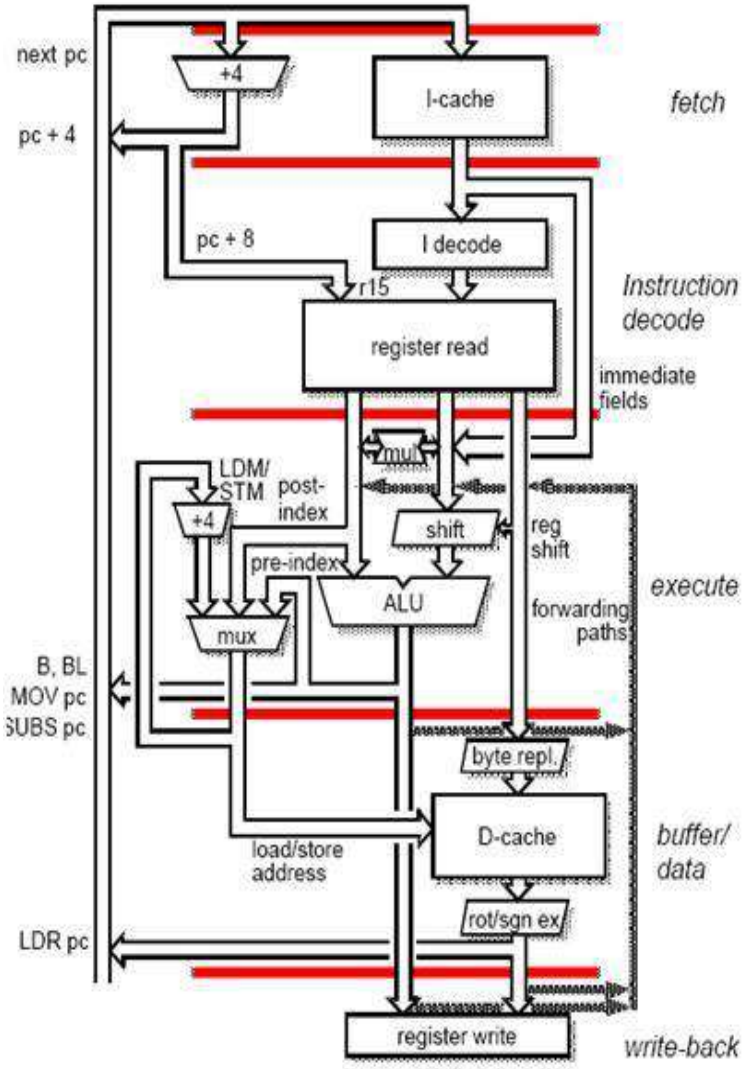
Microprocessor & Computer Architecture (µpCA)

ARM9TDMI- 5 Stage



Microprocessor & Computer Architecture (µpCA)

ARM 5-stage (ARM 9 Architecture)

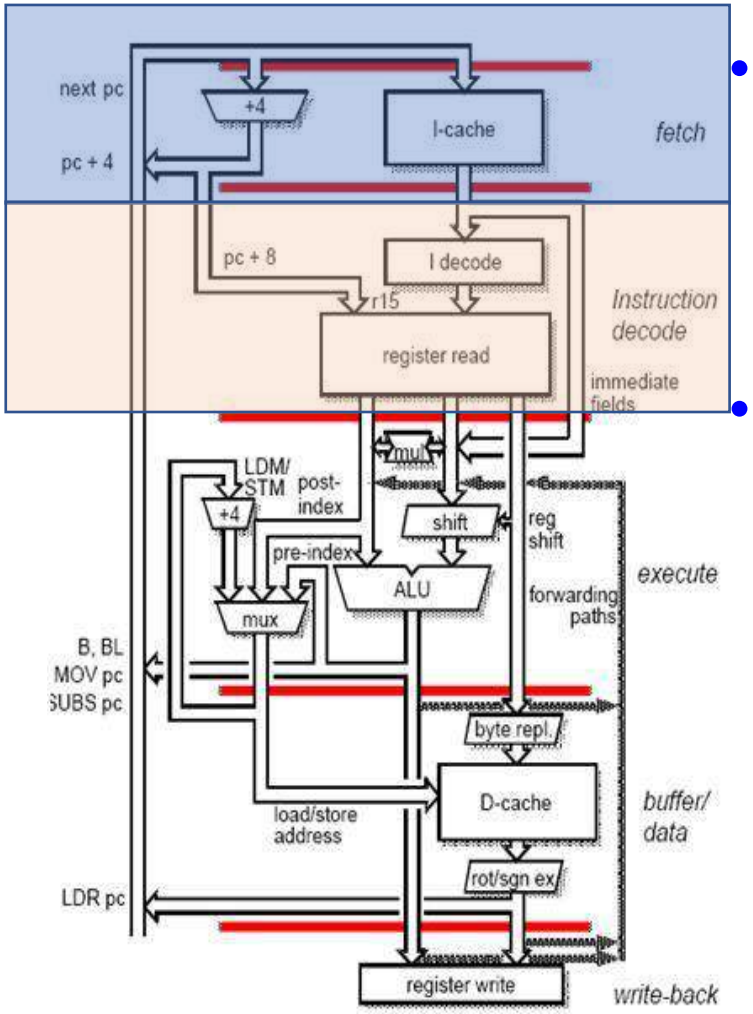


- **Fetch - [IF]**
- **Decode – [ID]**
- **Execute – [EX]**
- **Buffer/Data or Memory Access-[MEM]**
- **Write back – [WB]**

Instruction & Data Memory?
Split

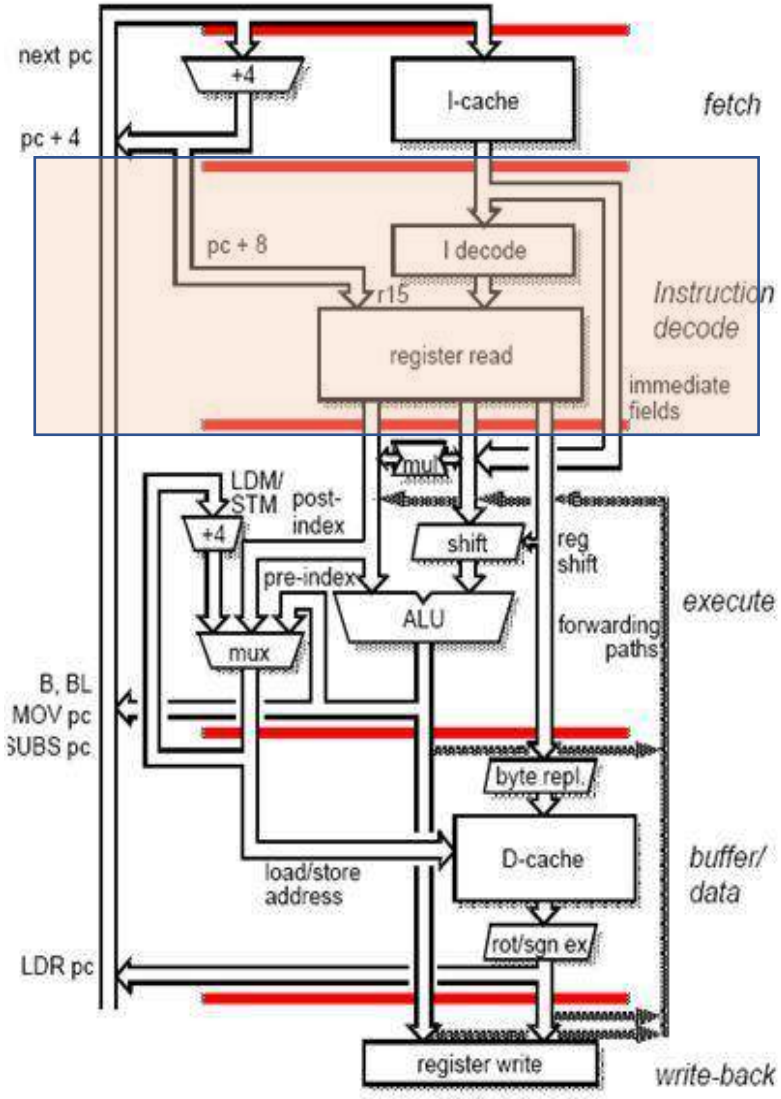
Harvard Architecture

ARM Architecture - 5 Stage Pipelining - Fetch - [IF], Decode – [ID]



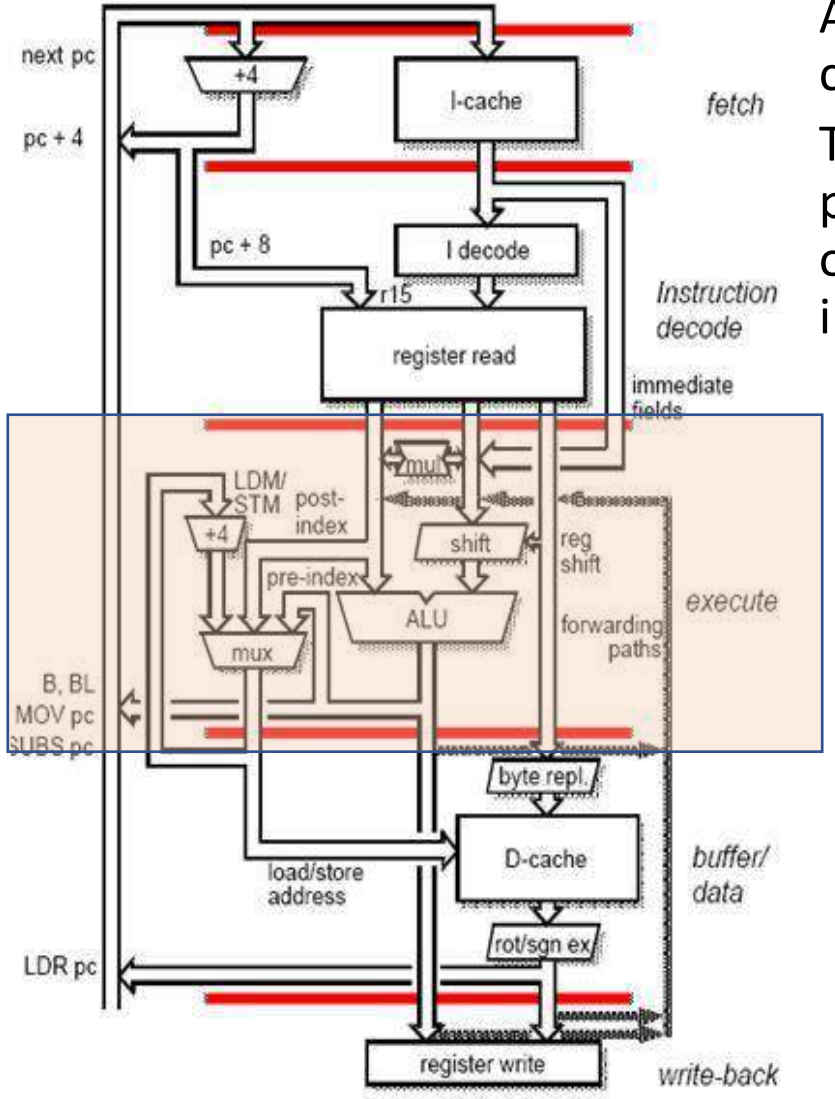
- **Fetch - [IF]**
  - The instruction is **fetched** from memory and placed in the instruction pipeline.
  - Update the PC to the next sequential PC by adding 4 to PC.
- **Decode – [ID]**
  - The instruction is **decoded** and **register operands read** from the register files. There are 3 operand read ports in the register file so most ARM instructions can source all their operands in one cycle.
  - Do the equality test on the registers as they are read, for a possible branch.
  - Sign extend the offset field of the instruction in case it is needed.

ARM Architecture - 5 Stage Pipelining- Decode stage - [ID]



- Compute the possible branch target address by adding the sign-extended offset to the incremented PC.
- Further, the branch can be completed by the end this stage by storing the branch target address into the PC if condition yielded true.
- Decoding is done in parallel with reading registers, as the register specifiers are at fixed location in a RISC architecture.

ARM Architecture - 5 Stage Pipelining Execute – [EX]

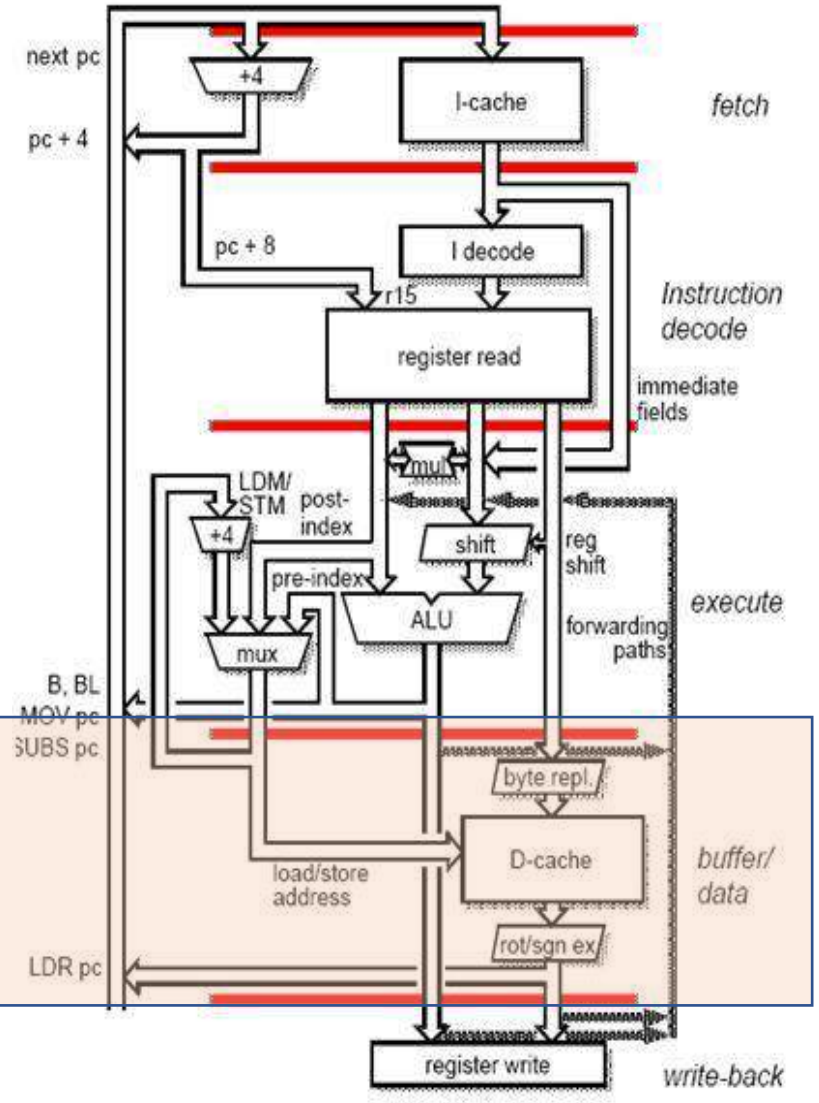


Also called as execute / effective address cycle.

The ALU operates on the operands prepared in the previous cycle, performing of the three functions depending on the instruction type.

- Memory Reference: the ALU adds the base register and the offset to form the effective address.
- Register – Register ALU instruction: The ALU performs the operation specified by the opcode on the values read from the register file.
- Register – Immediate ALU instruction: The ALU performs the operations specified by the opcode on the first value read from the register file and the sign extended immediate.

Buffer/Data or Memory Access-[MEM]



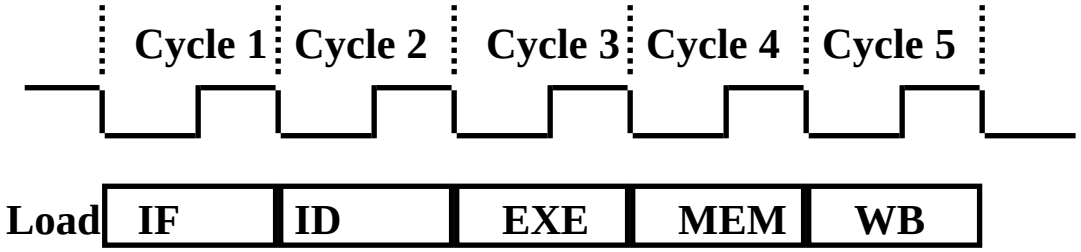
Data memory is accessed if required. Otherwise the ALU result is simply buffered for one cycle.

If the instruction is a LOAD, the memory does a read using effective address computed in the previous cycle.

If the instruction is a STORE, then the memory writes the data from the second register read using the effective address.

Buffer/Data or Memory Access-[MEM]

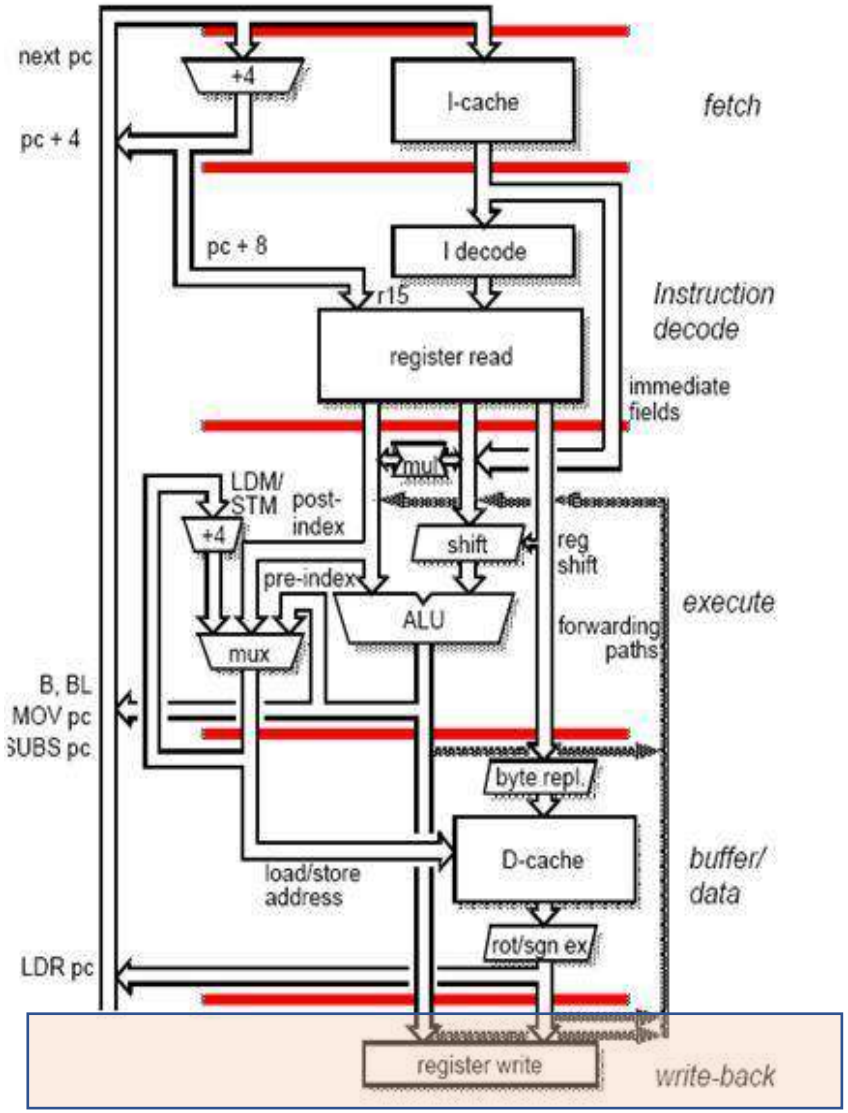
The Five Stages of Load



- IF: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- ID: Registers Fetch and Instruction Decode
- EXE: Calculate the memory address
- MEM: Read the data from the Data Memory
- WB: Write the data back to the register file

Microprocessor & Computer Architecture (µpCA)

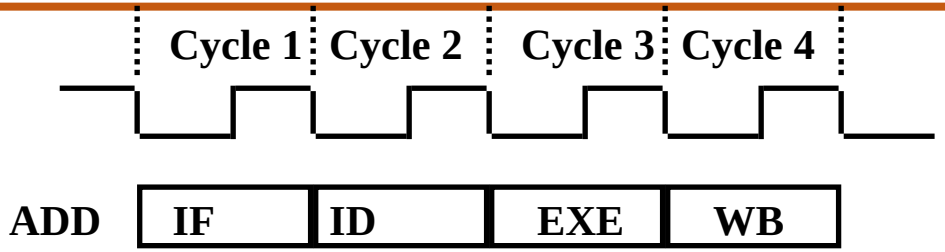
ARM Architecture - 5 Stage Pipelining - Write back – [WB]



The result generated by the instruction is **written back to the register file**.

The data may come either from memory system [for LOAD], or from the ALU [for an ALU instruction].

The Four Stages of Data Processing



- IF: Instruction Fetch
  - Fetch the instruction from the Instruction Memory
- ID: Registers Fetch and Instruction Decode
- EXE:
  - ALU operates on the two register operands
  - Update PC
- WB: Write the ALU output back to the register file

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135

Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Introduction to Pipeline Processor

Dr. D. C. Kiran
Department of Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

Introduction to Pipeline Processor

Unit 3: **Memory Design**

Unit 4: **Input/Output Device Design**

Unit 5: **Advanced Architecture**

Microprocessor & Computer Architecture (µpCA)

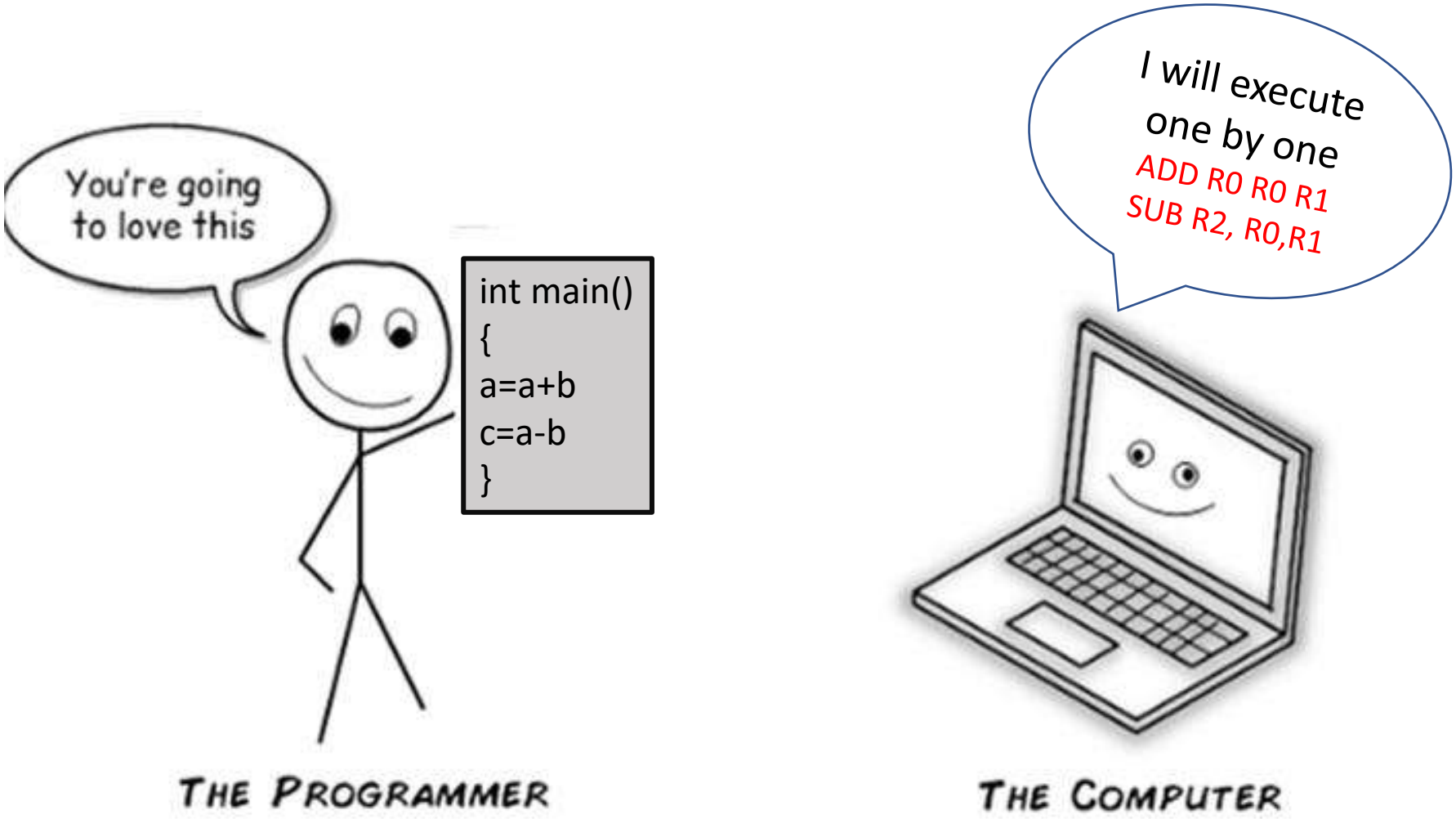
Text 1: "Computer Organization and Design", Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

Reference 1:"Computer Architecture: A Quantitative Approach", Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.

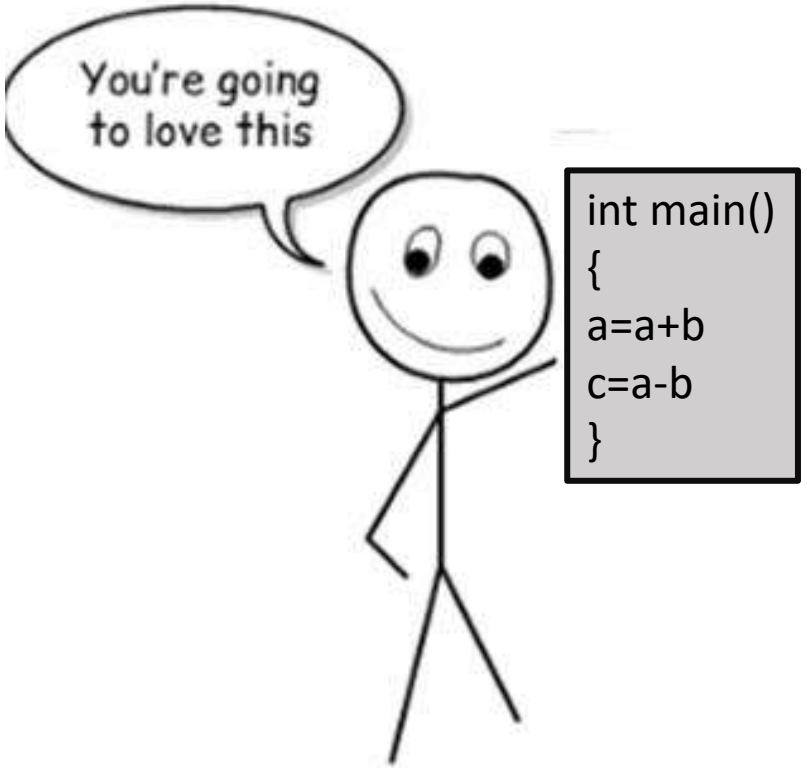
Appendix C **Pipelining: Basic and Intermediate Concepts**

C.1	Introduction	C-2
C.2	The Major Hurdle of Pipelining—Pipeline Hazards	C-11
C.3	How Is Pipelining Implemented?	C-30
C.4	What Makes Pipelining Hard to Implement?	C-43
C.5	Extending the MIPS Pipeline to Handle Multicycle Operations	C-51
C.6	Putting It All Together: The MIPS R4000 Pipeline	C-61
C.7	Crosscutting Issues	C-70
C.8	Fallacies and Pitfalls	C-80
C.9	Concluding Remarks	C-81
C.10	Historical Perspective and References	C-81
	Updated Exercises by Diana Franklin	C-82

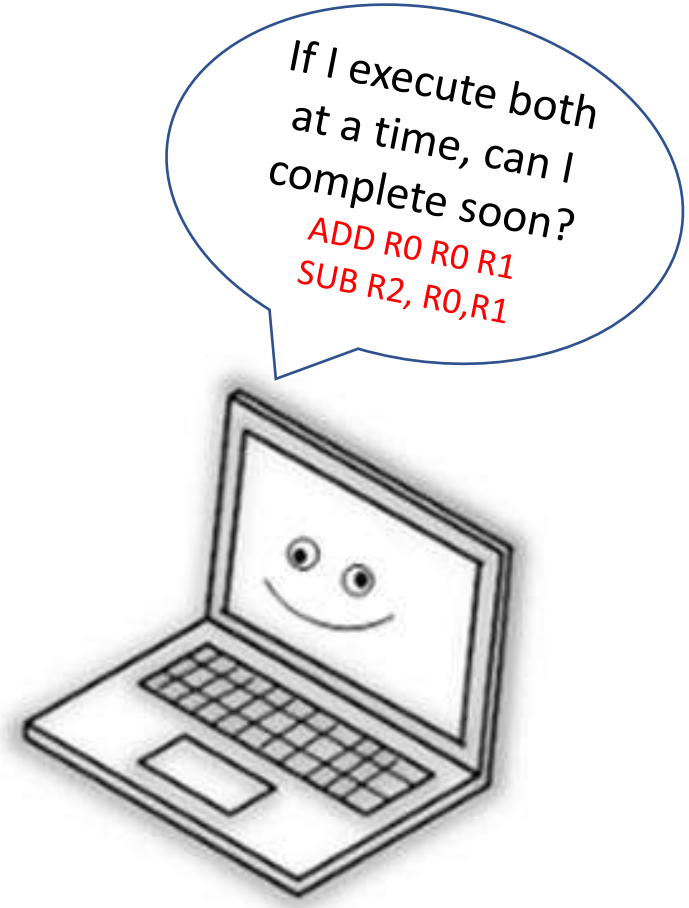
Processor


You're going to love this
int main()
{
a=a+b
c=a-b
}
THE PROGRAMMER
I will execute one by one
ADD R0 R0 R1
SUB R2, R0,R1
THE COMPUTER

Processor



THE PROGRAMMER

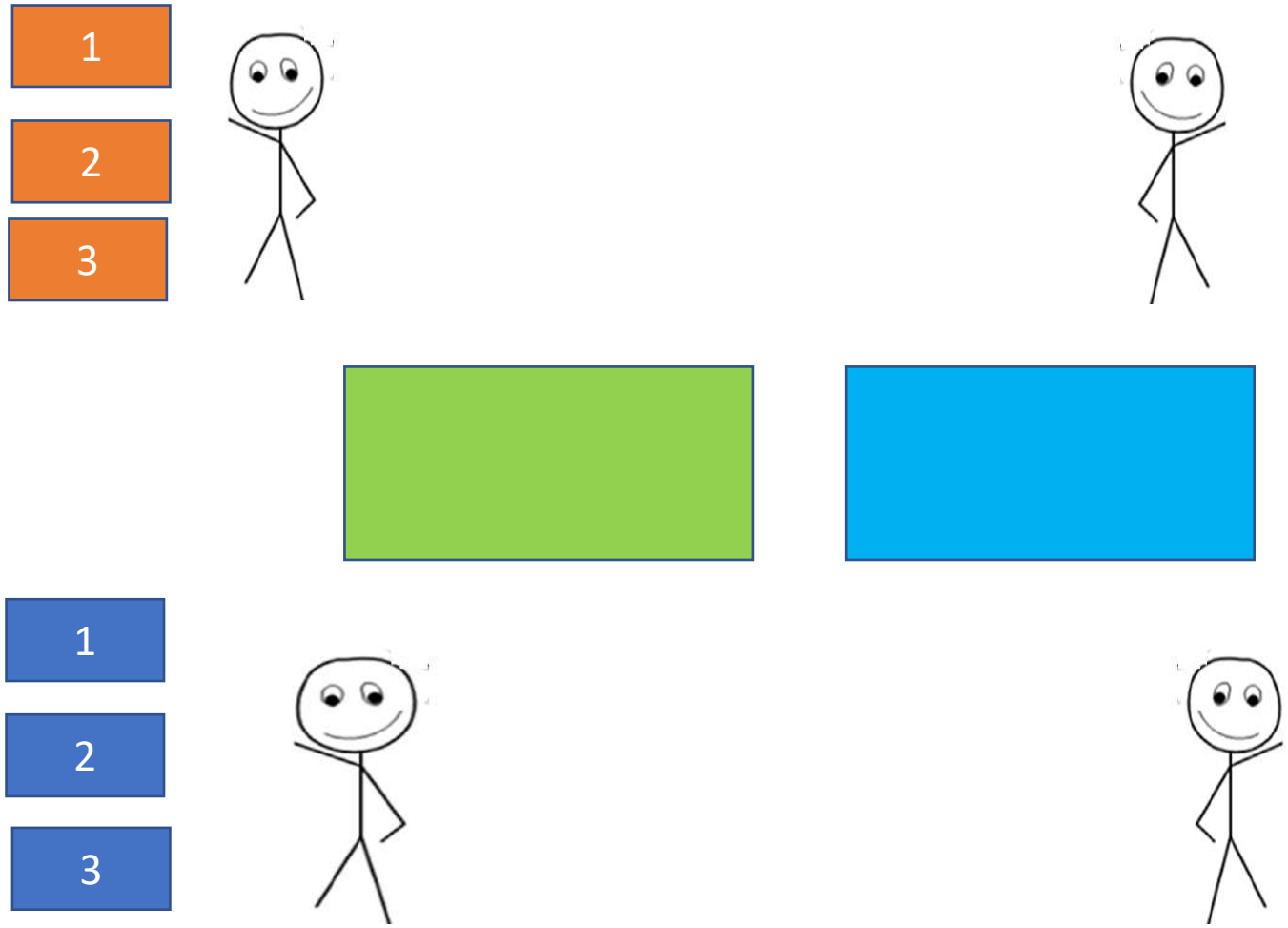


THE COMPUTER

Technique 1

Shifting One Brick take 10 mins

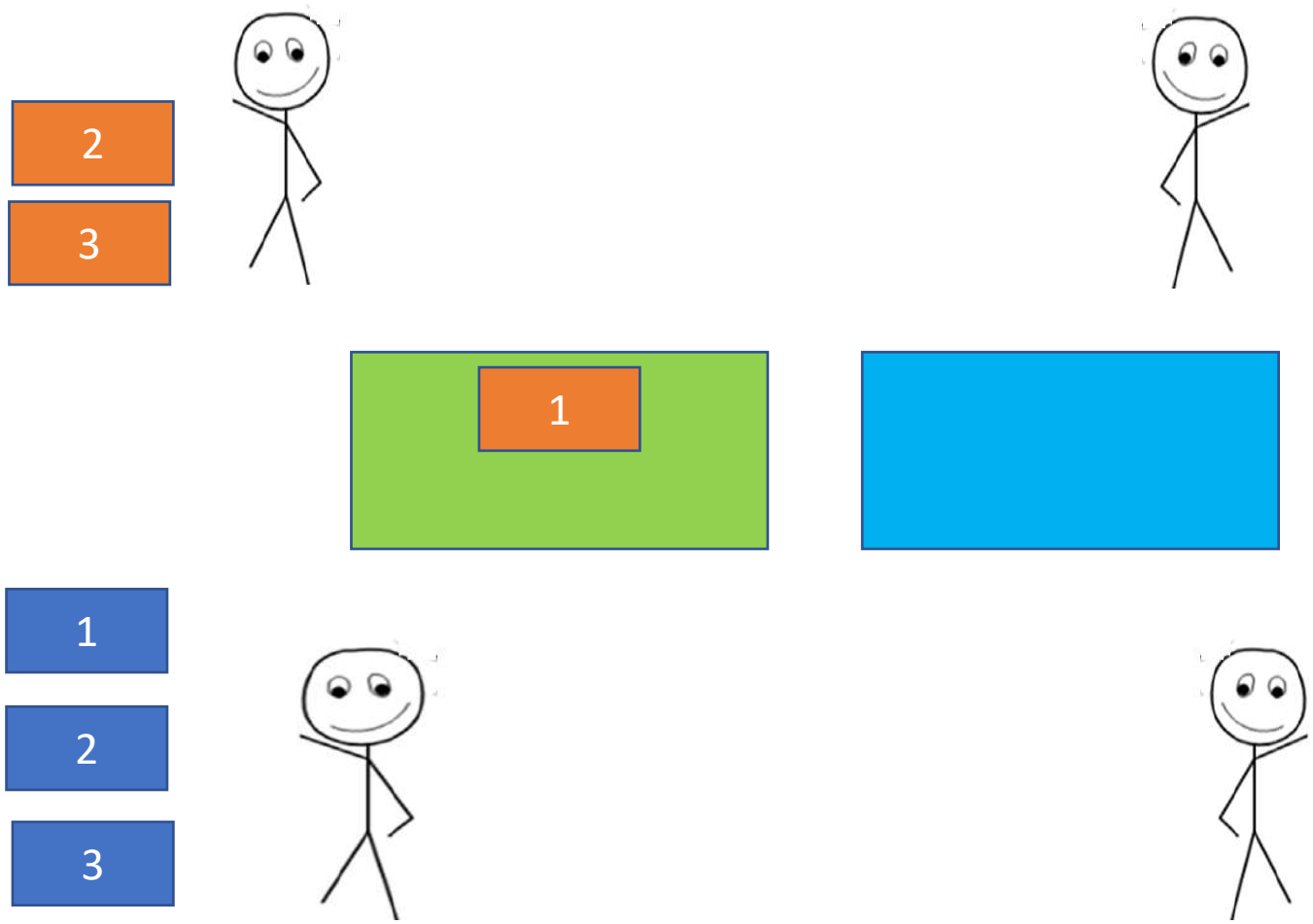
Timer: 0



Technique 1

Shifting One Brick take 10 mins

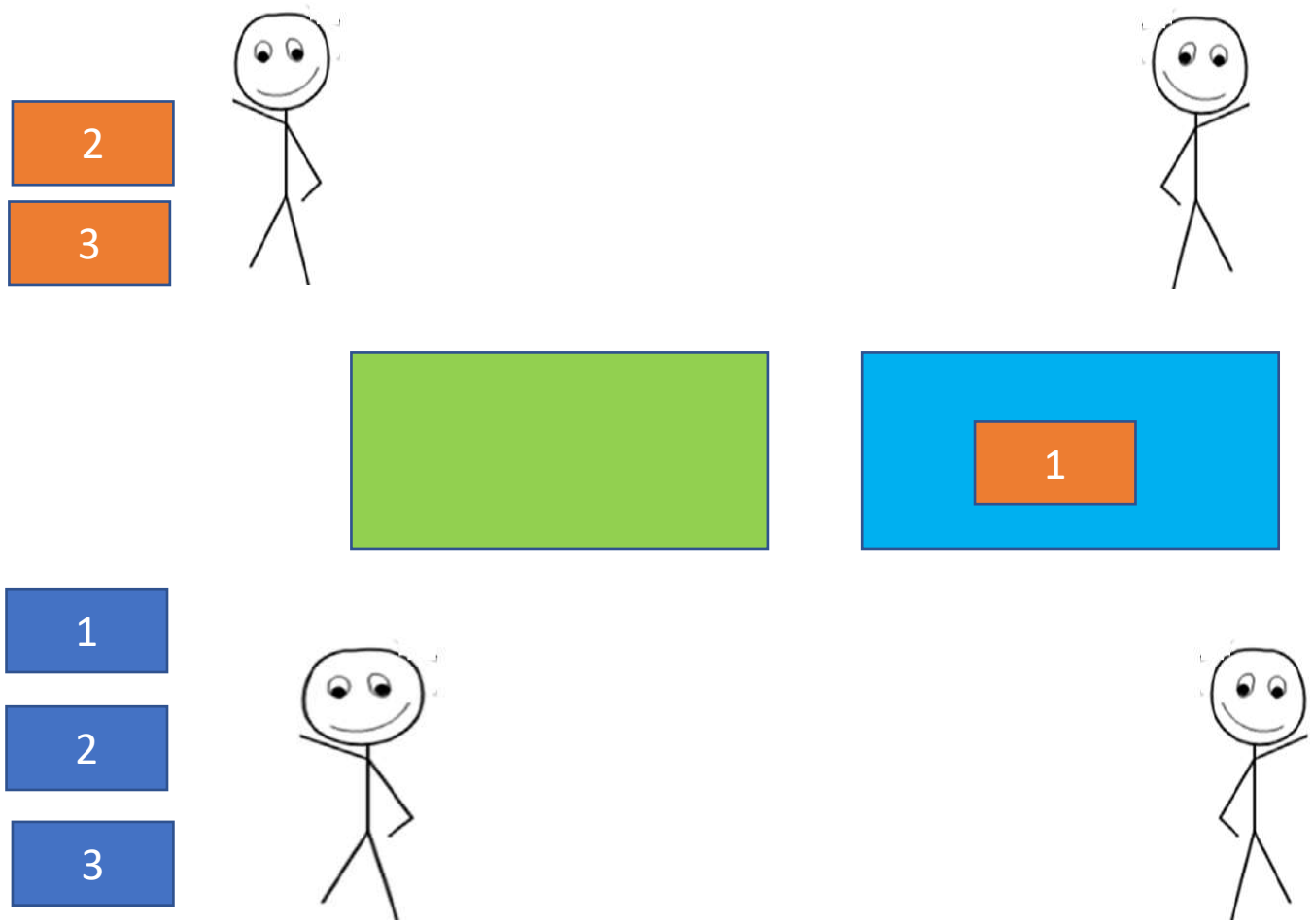
Timer: 5



Technique 1

Shifting One Brick take 10 mins

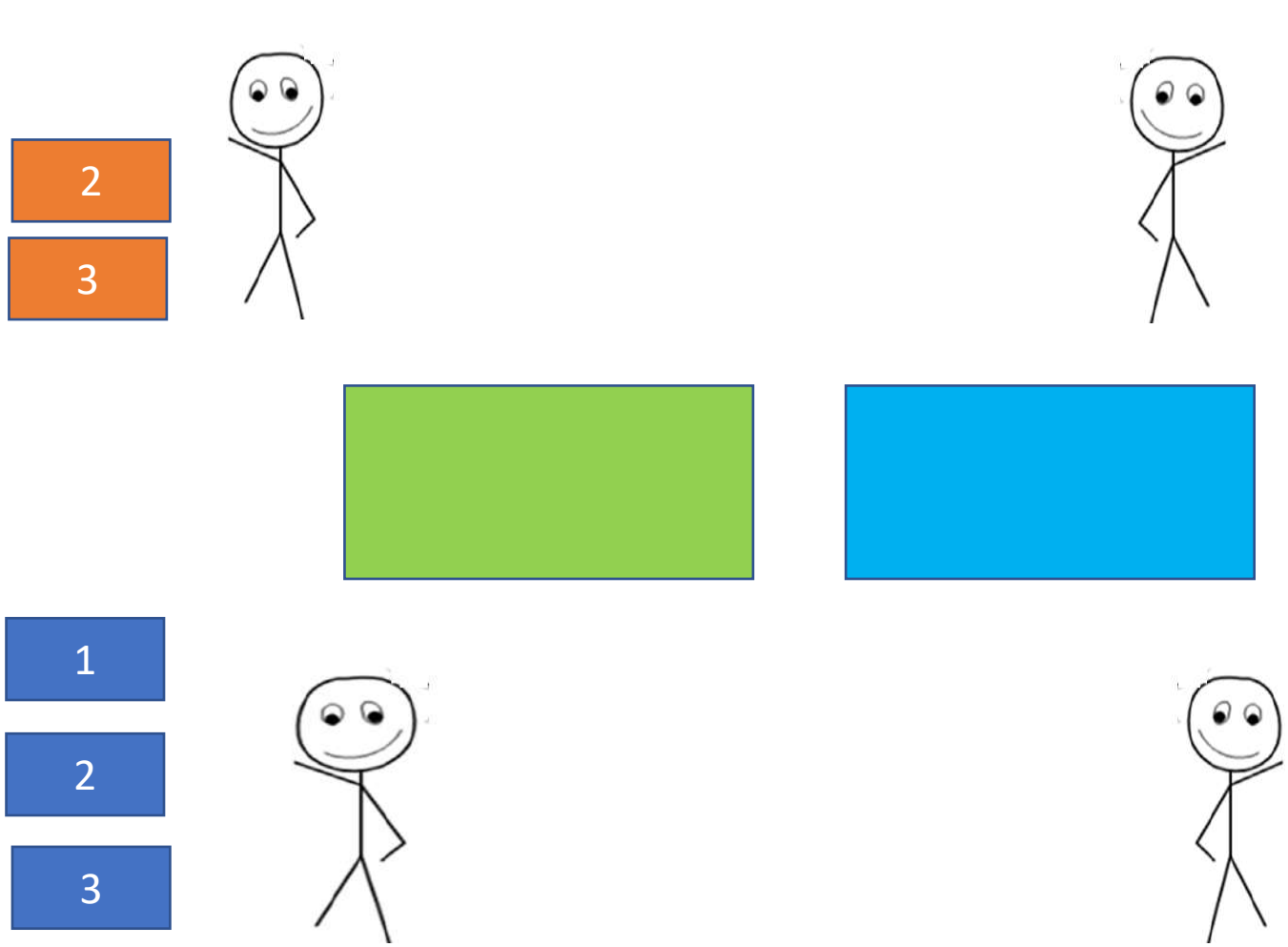
Timer: 10



Technique 1

Shifting One Brick take 10 mins

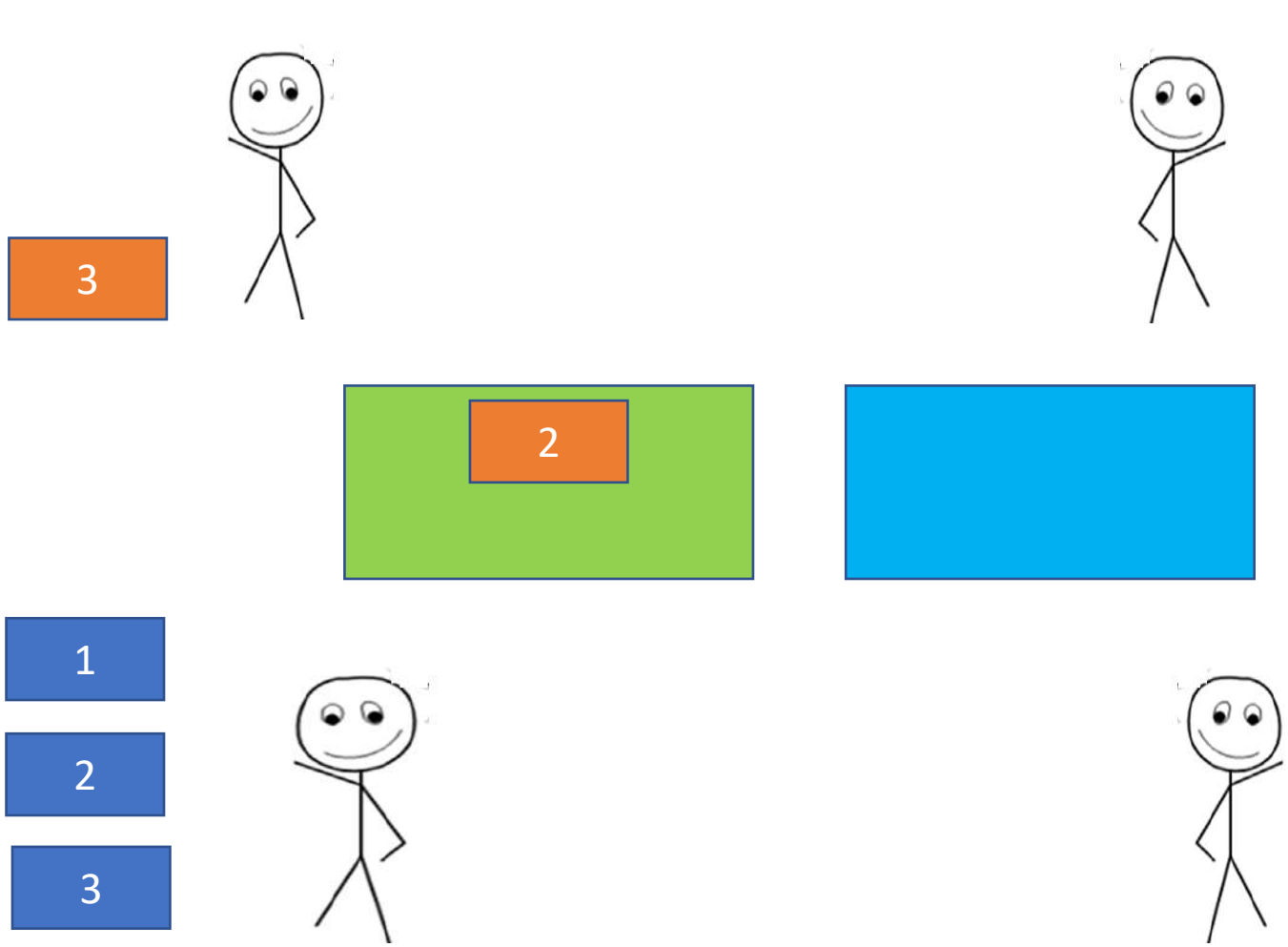
Timer: 10



Technique 1

Shifting One Brick take 10 mins

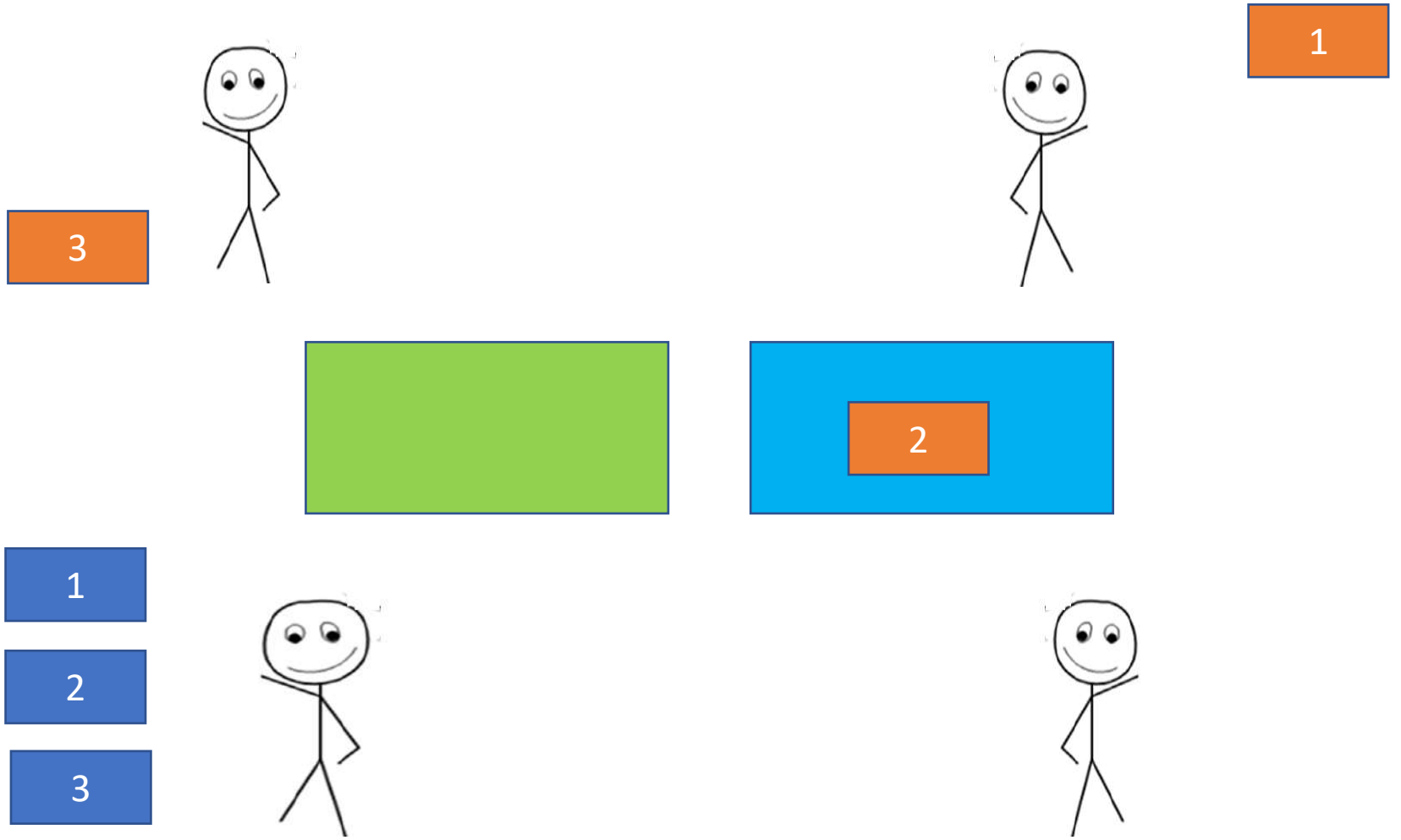
Timer: 15



Technique 1

Shifting One Brick take 10 mins

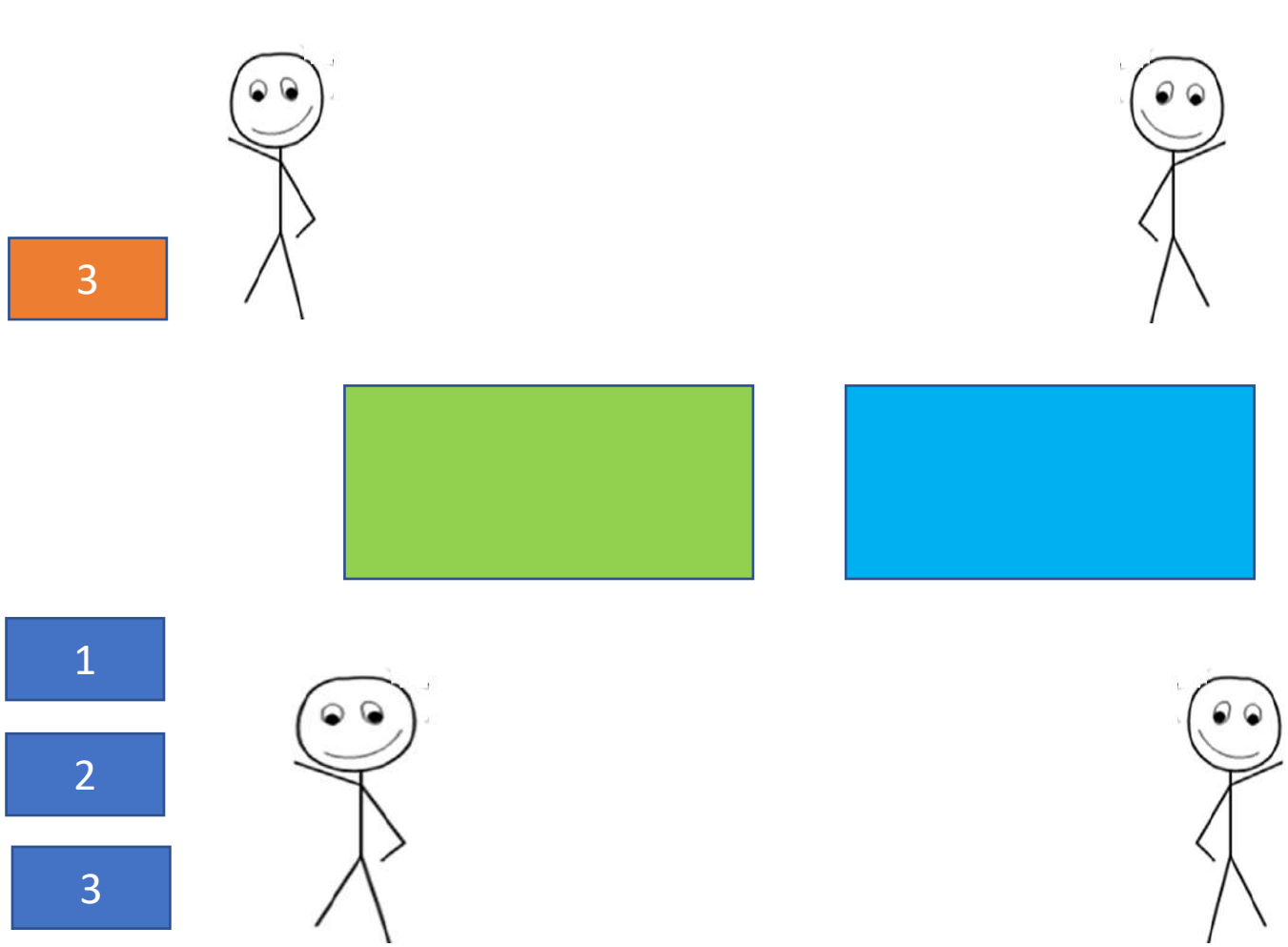
Timer: 20



Technique 1

Shifting One Brick take 10 mins

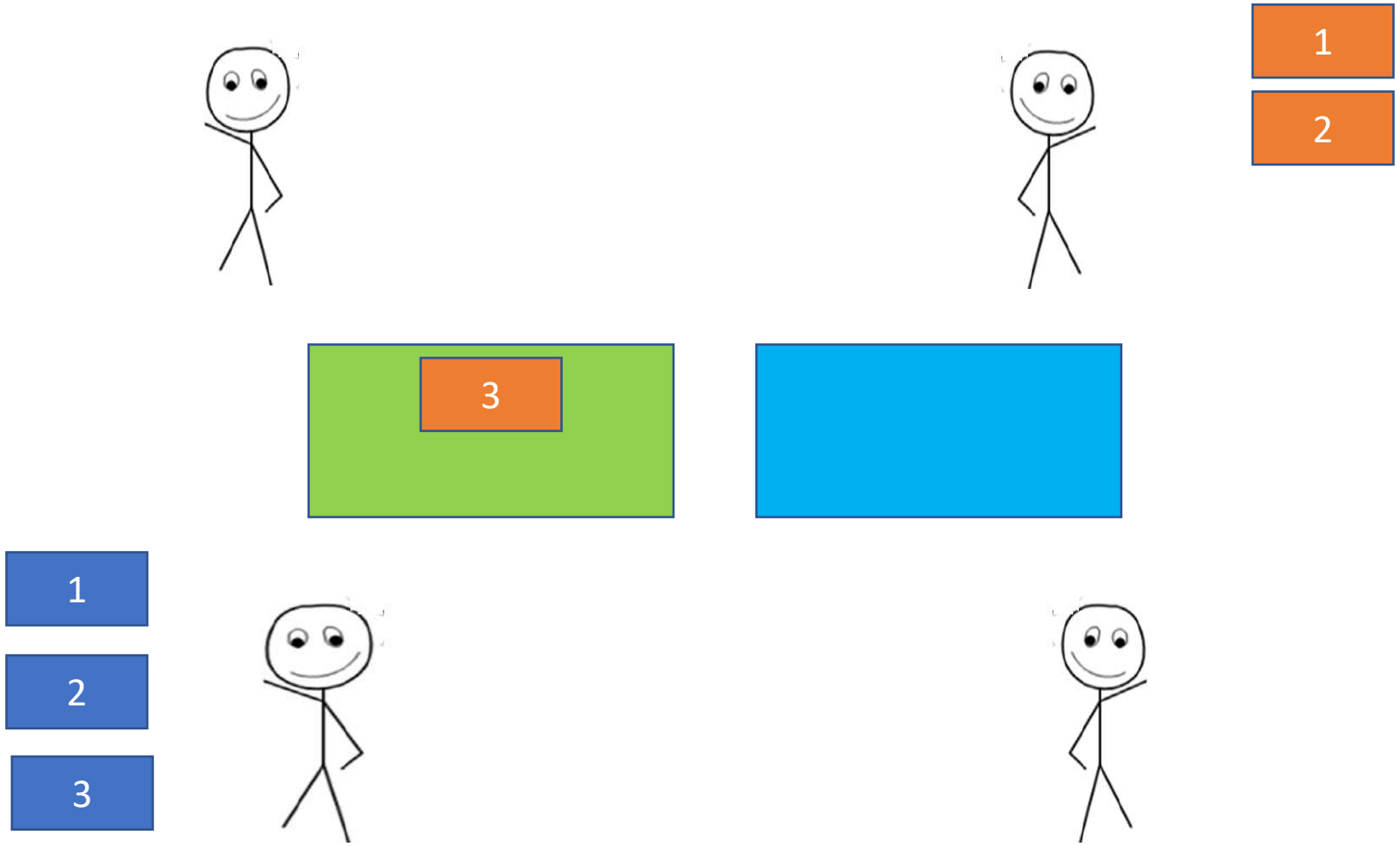
Timer: 20



Technique 1

Shifting One Brick take 10 mins

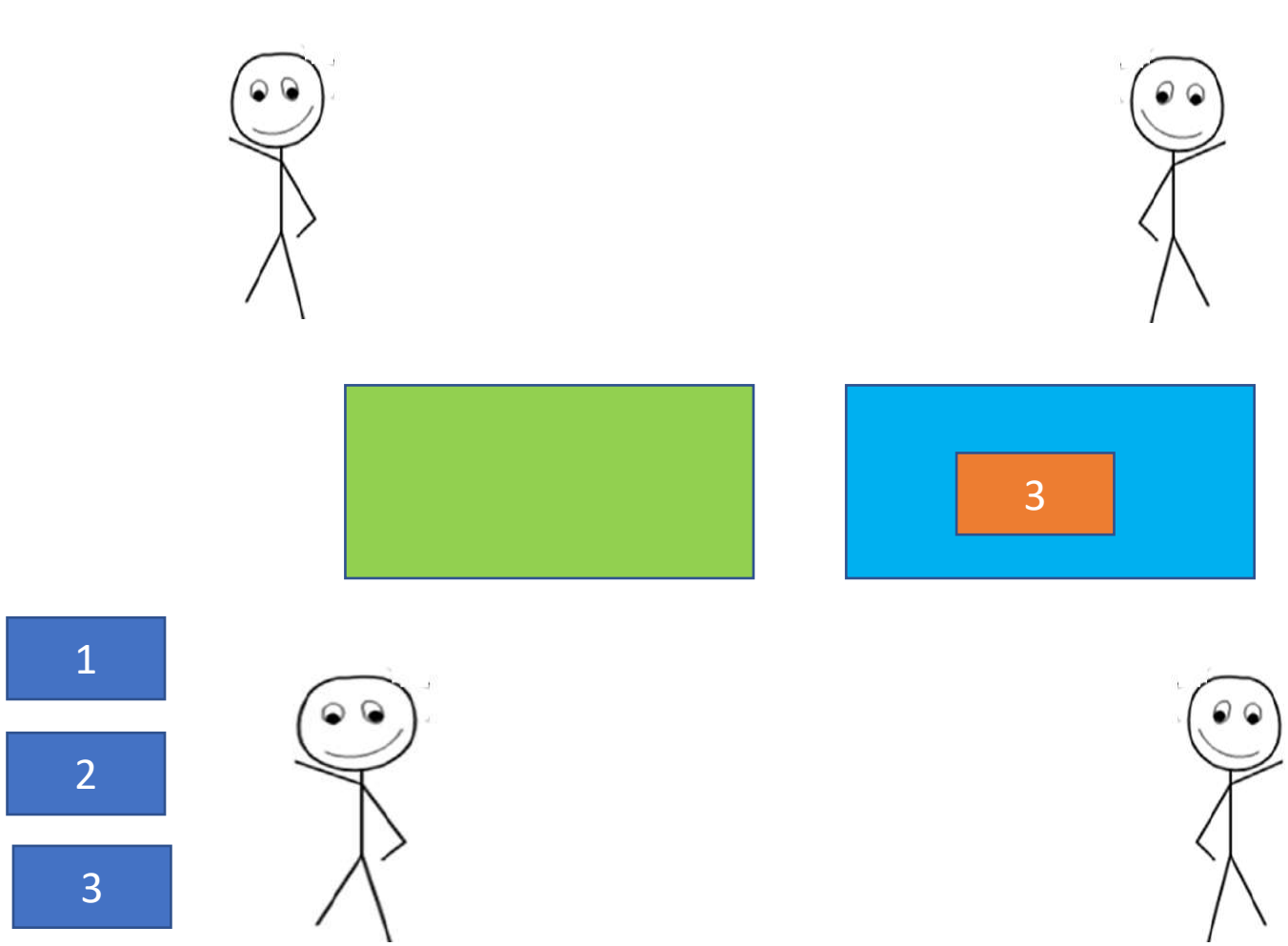
Timer: 25



Technique 1

Shifting One Brick take 10 mins

Timer: 30



Technique 1

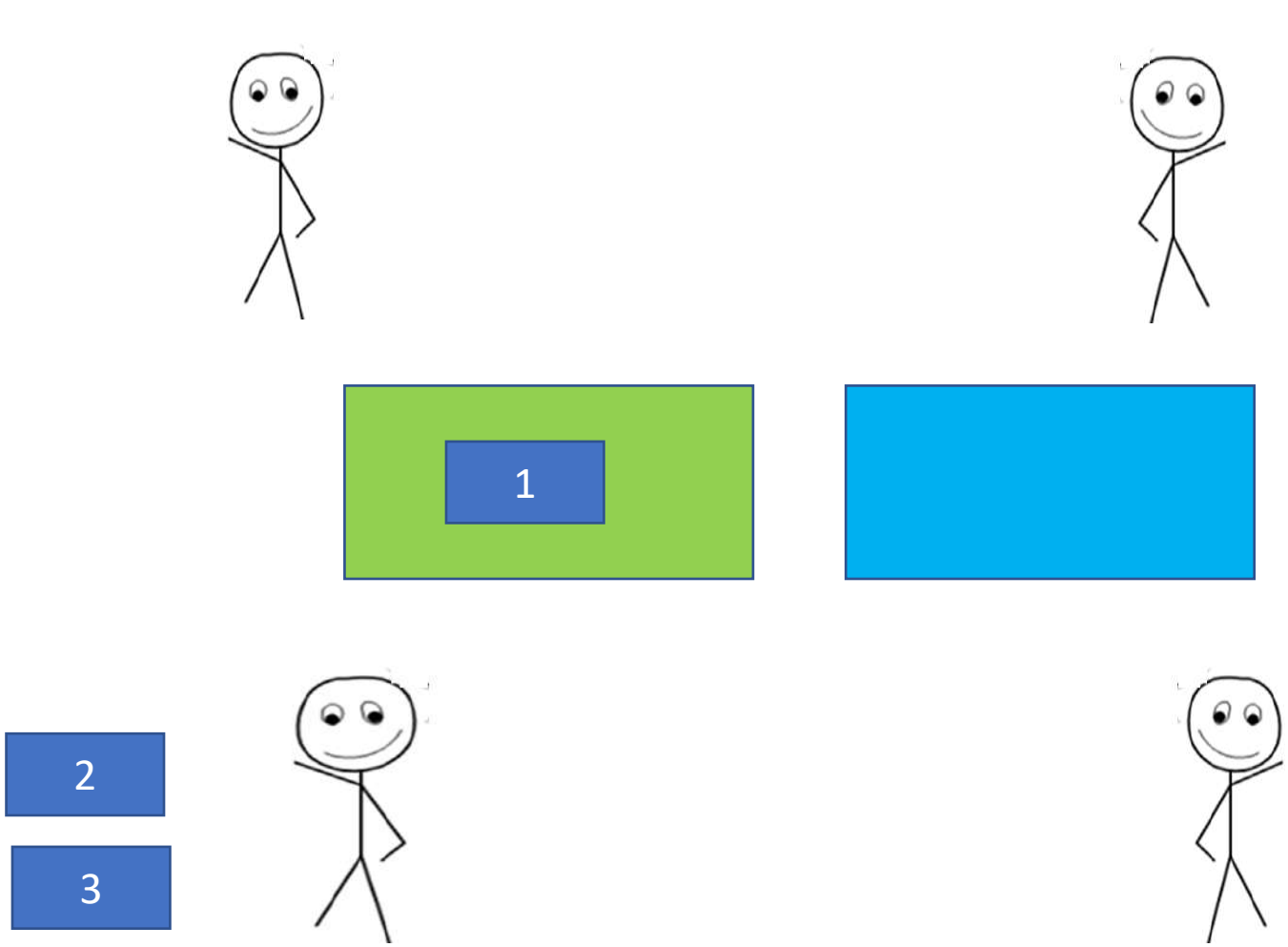
Shifting One Brick take 10 mins

Timer: 30

Technique 1

Shifting One Brick take 10 mins

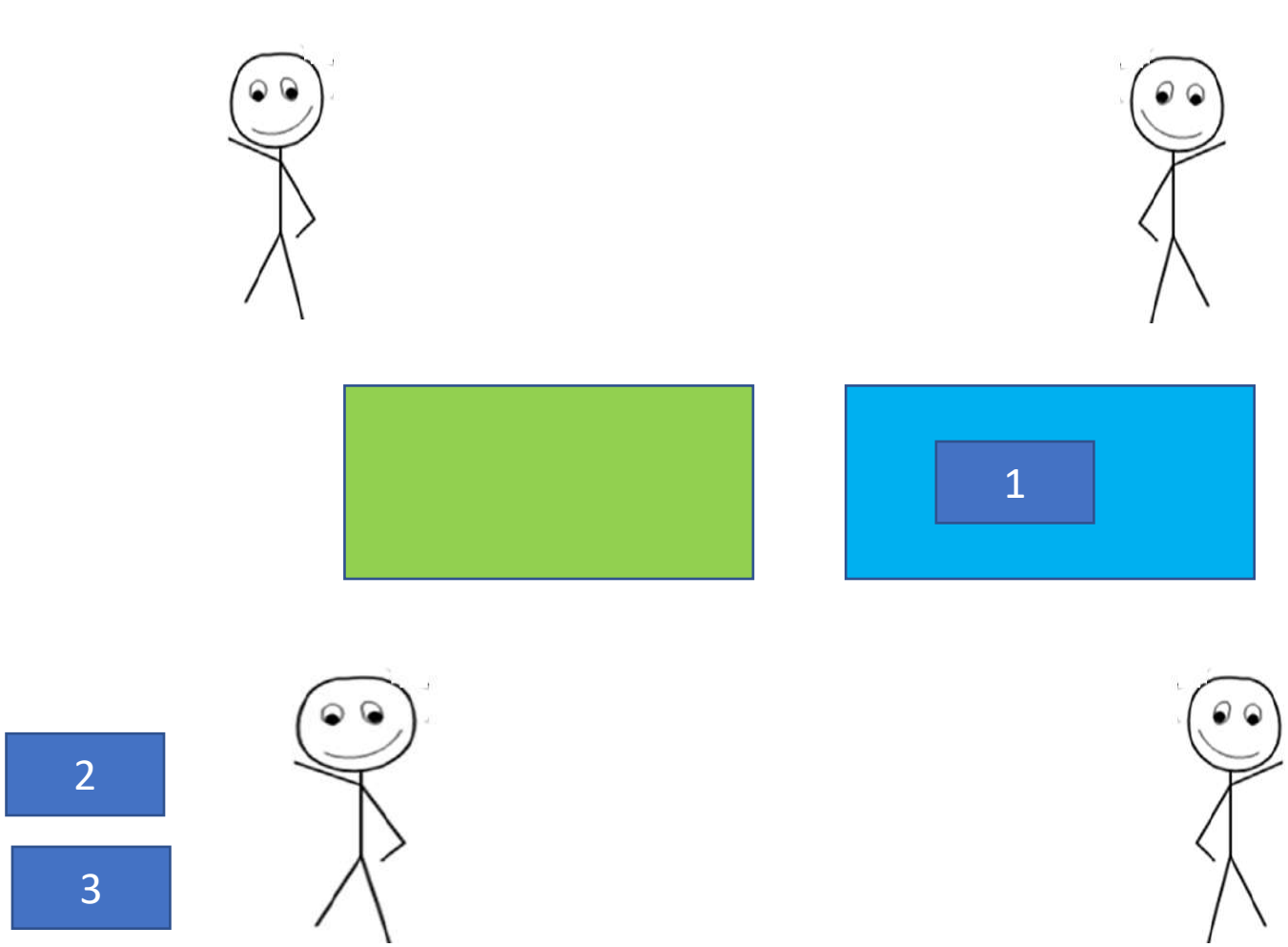
Timer: 35



Technique 1

Shifting One Brick take 10 mins

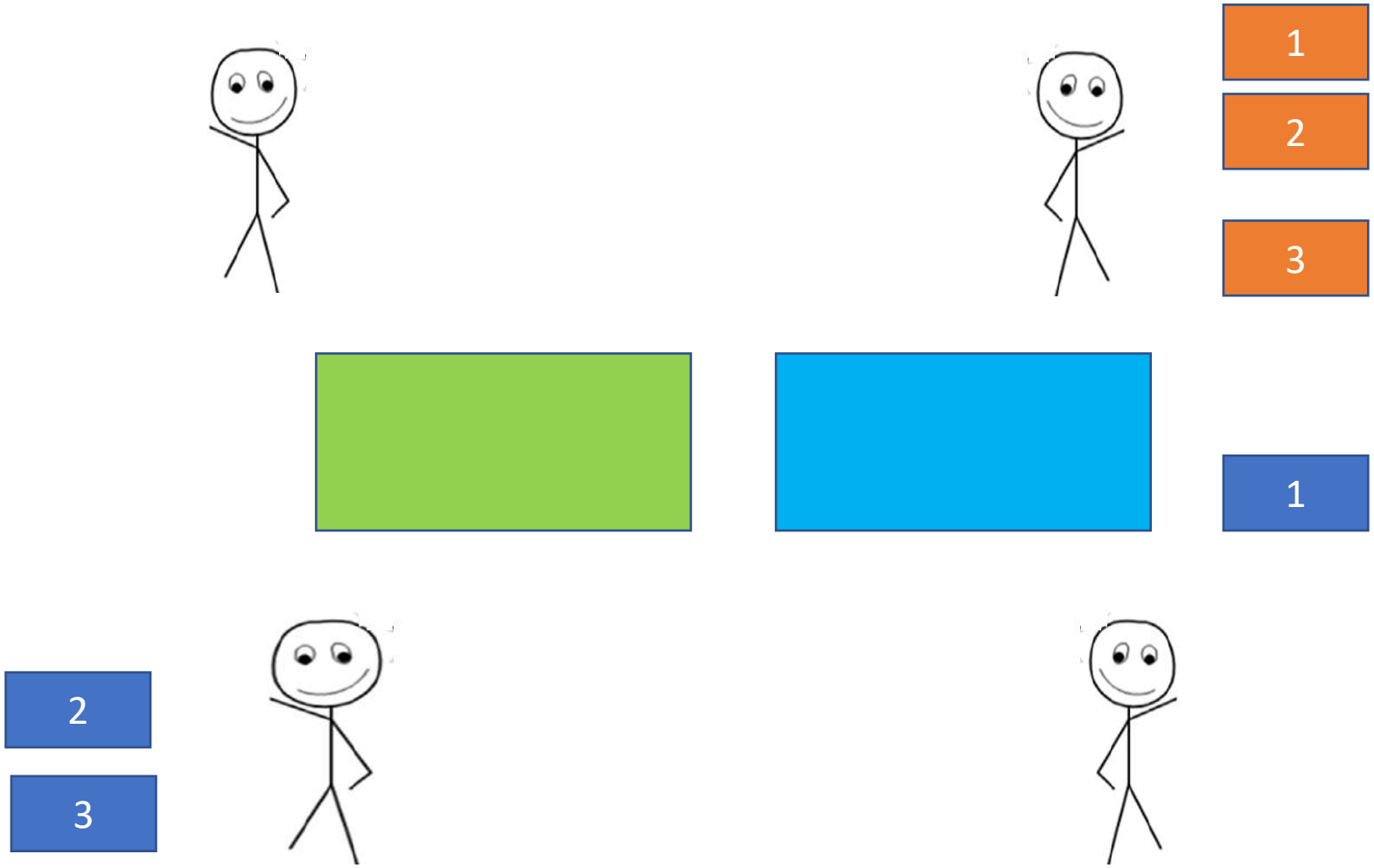
Timer: 40



Technique 1

Shifting One Brick take 10 mins

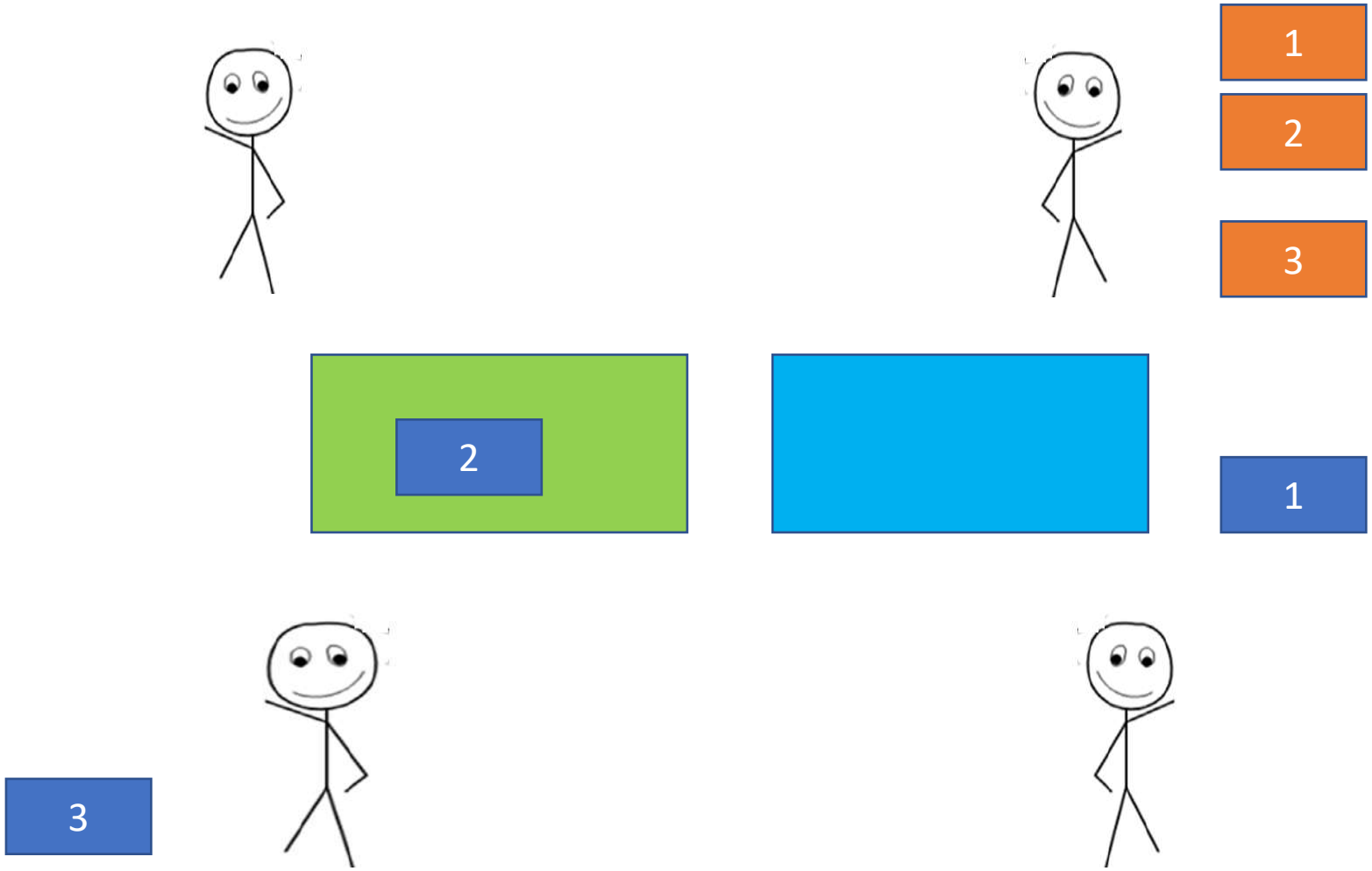
Timer: 40



Technique 1

Shifting One Brick take 10 mins

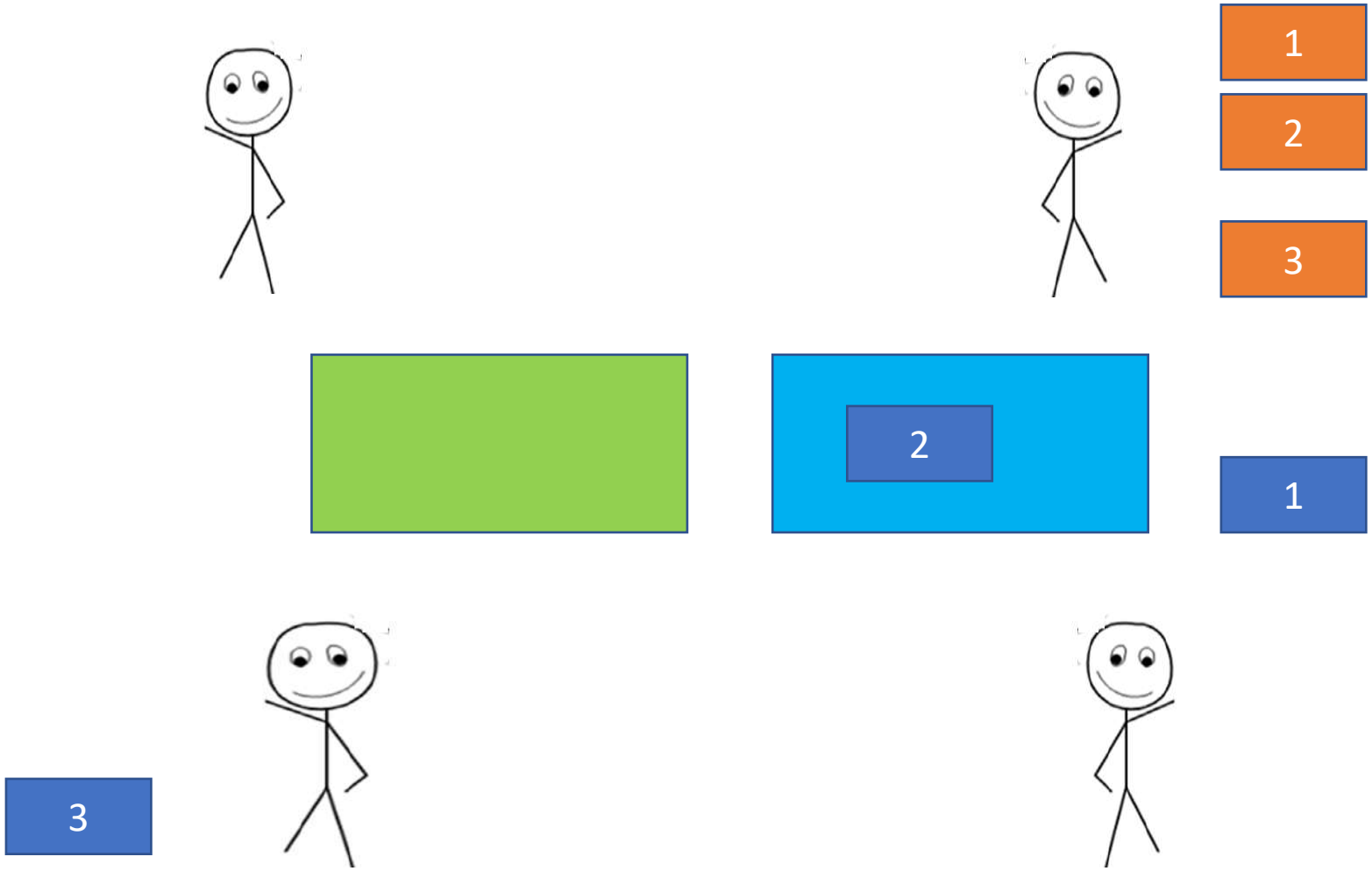
Timer: 45



Technique 1

Shifting One Brick take 10 mins

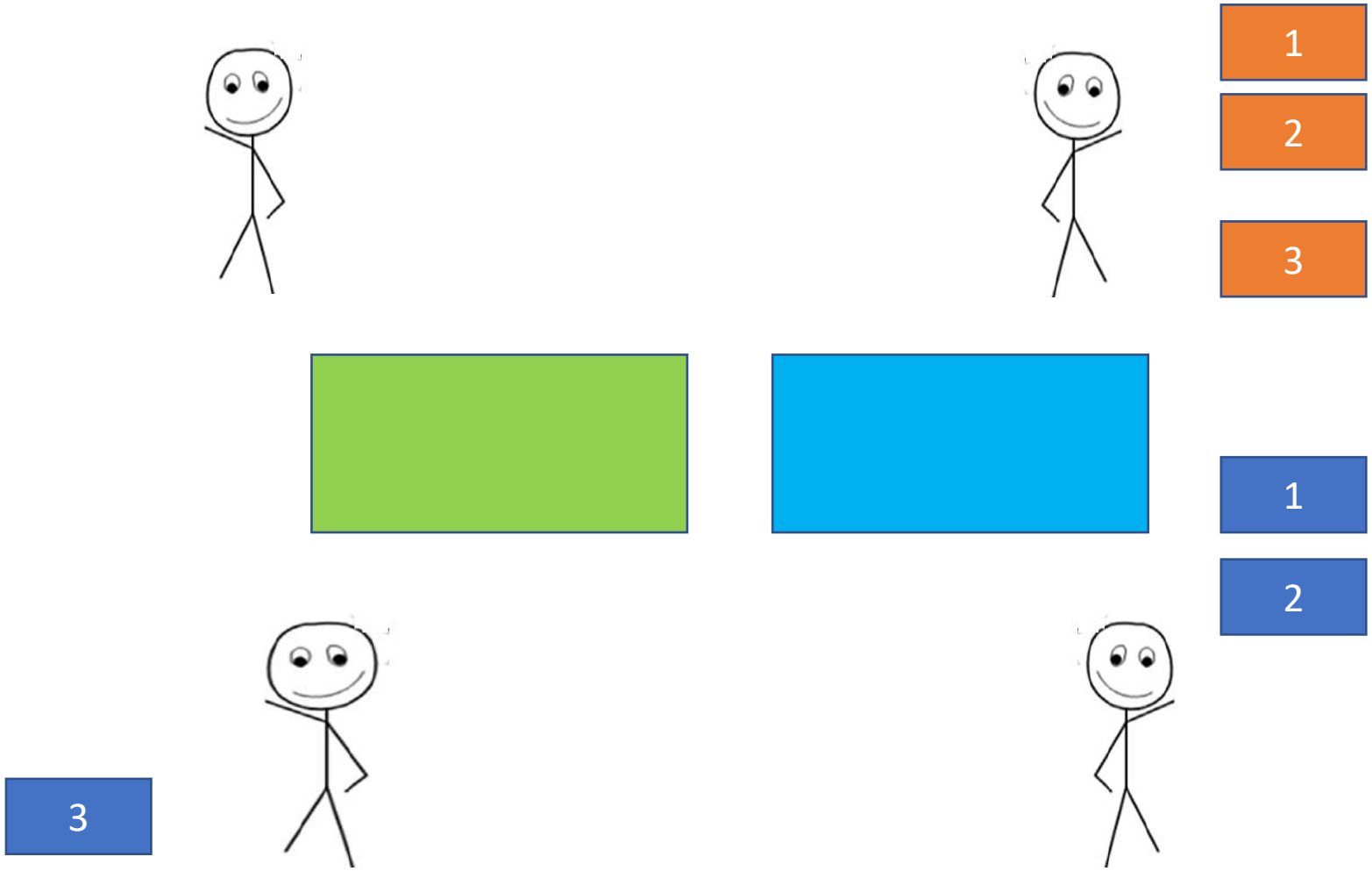
Timer: 50



Technique 1

Shifting One Brick take 10 mins

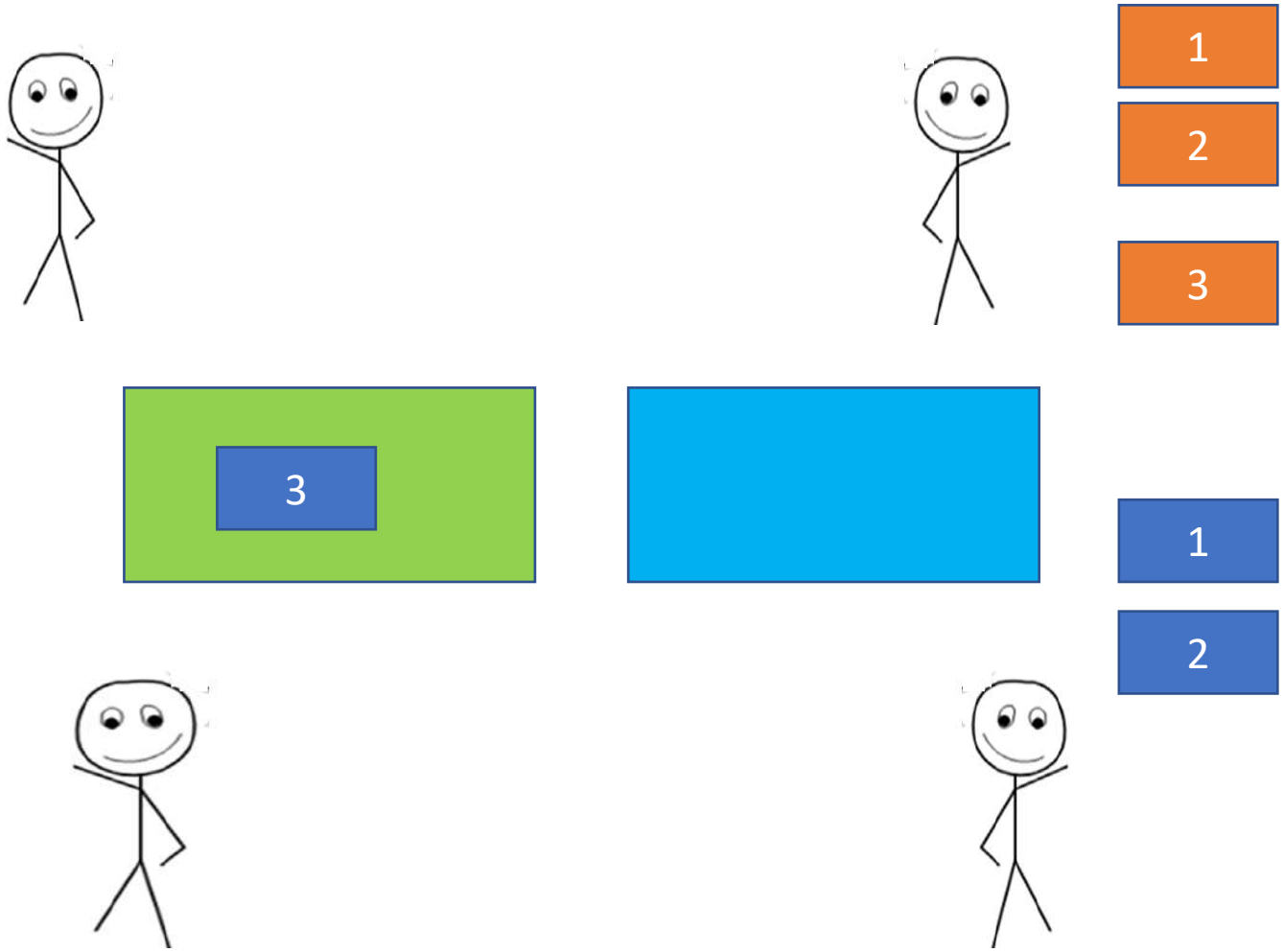
Timer: 50



Technique 1

Shifting One Brick take 10 mins

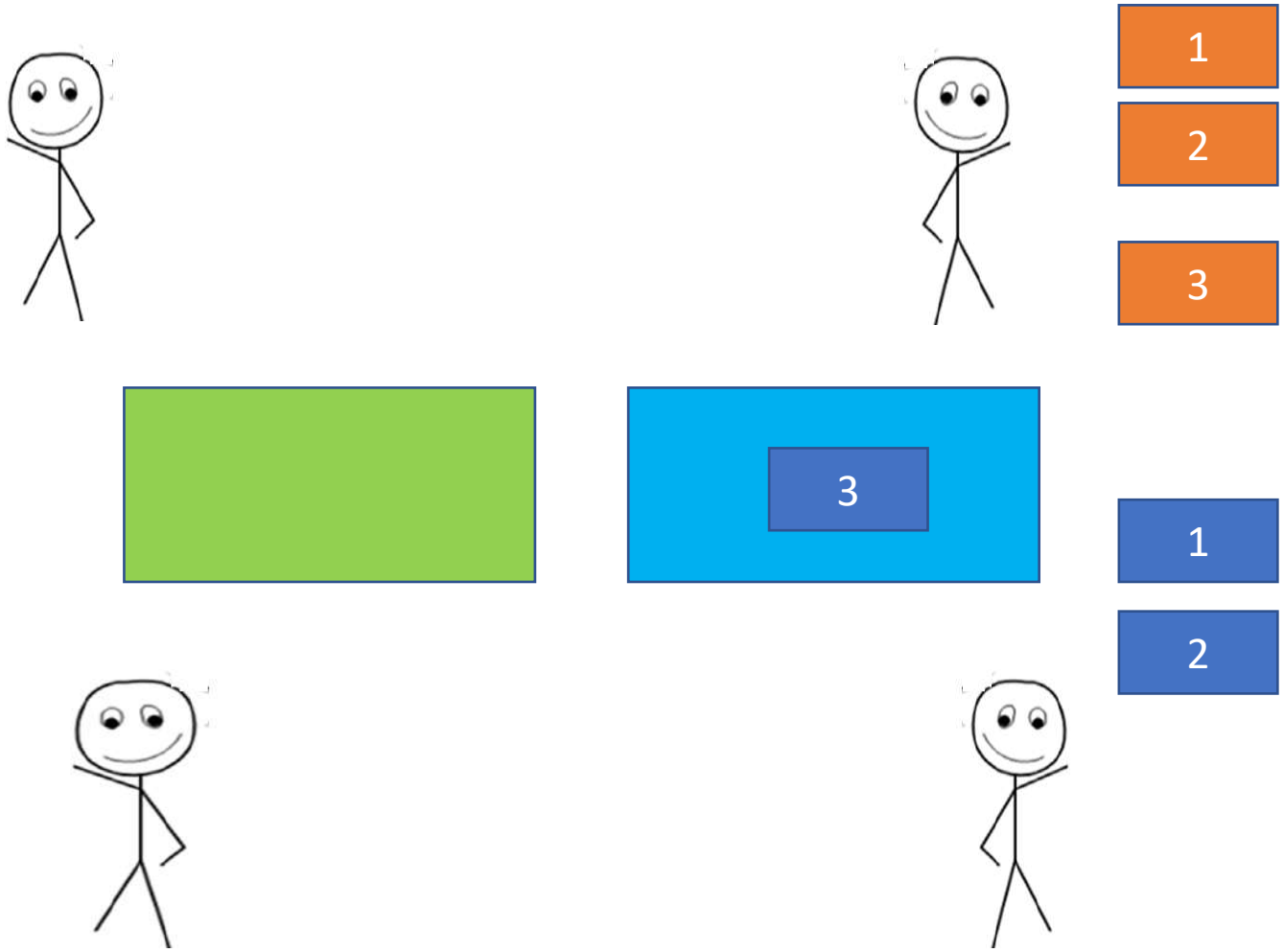
Timer: 55



Technique 1

Shifting One Brick take 10 mins

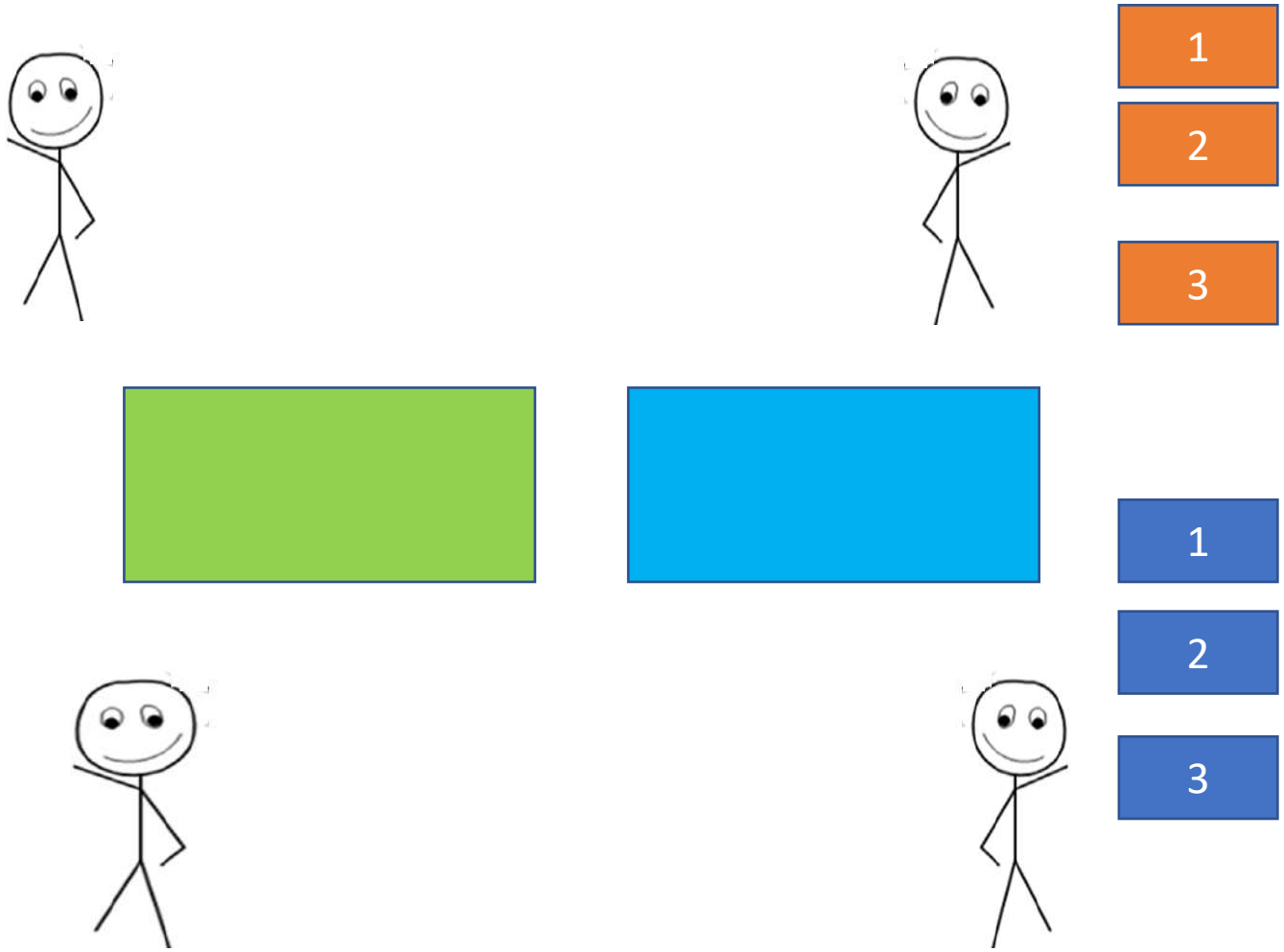
Timer: 60



Technique 1

Shifting One Brick take 10 mins

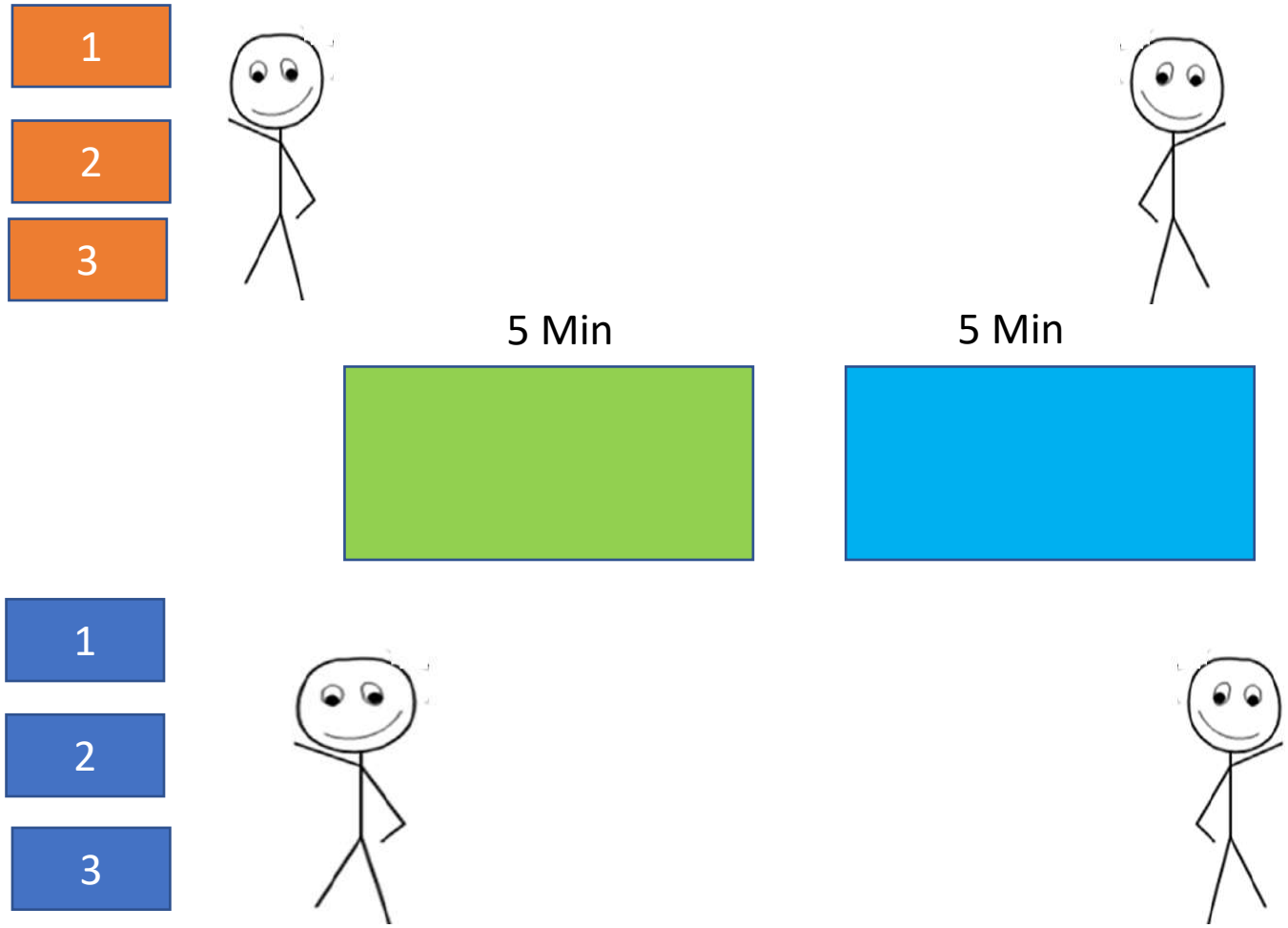
Timer: 60



Technique 2

Shifting One Brick take 10 mins

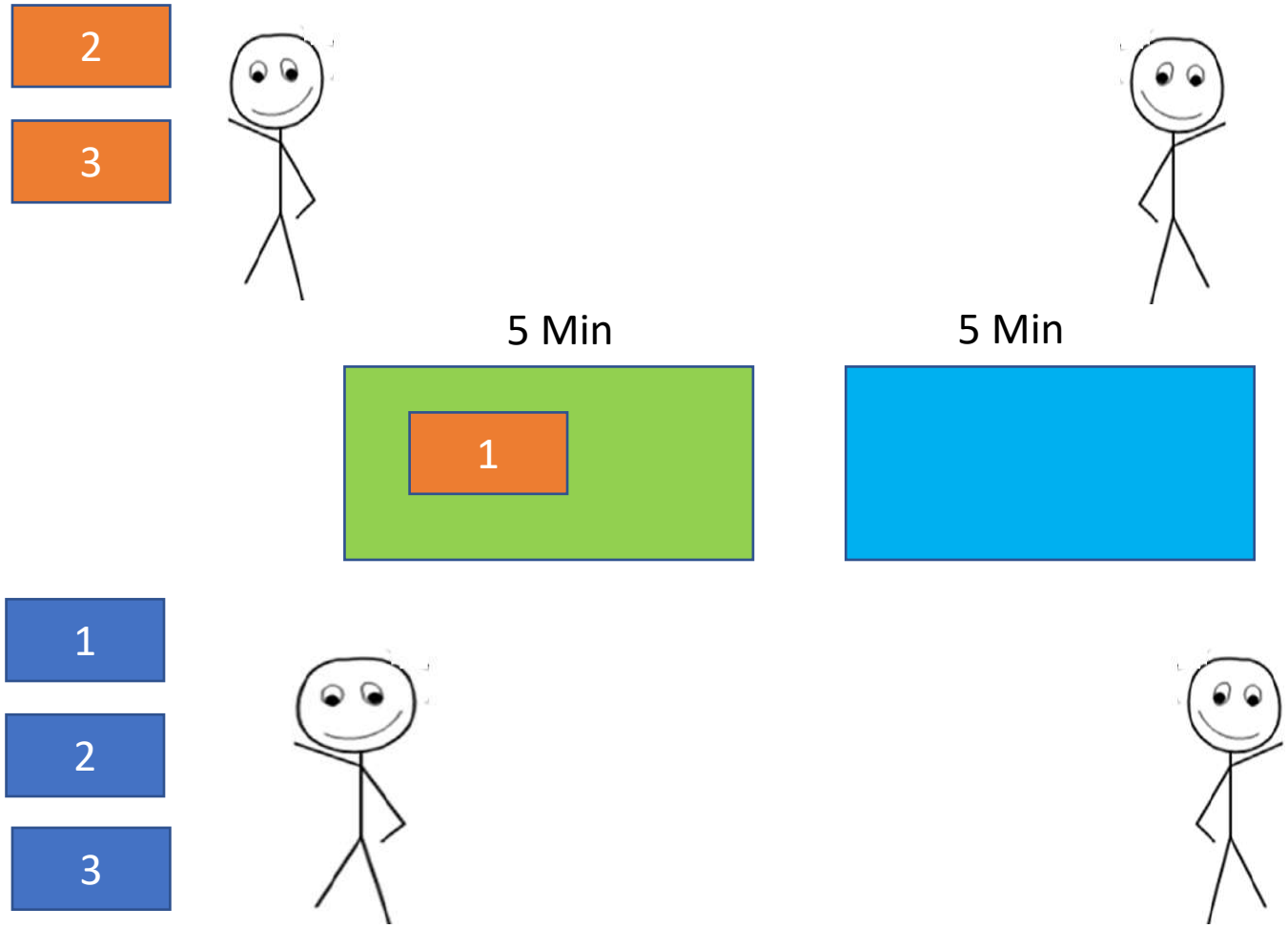
Timer: 0



Technique 2

Shifting One Brick take 10 mins

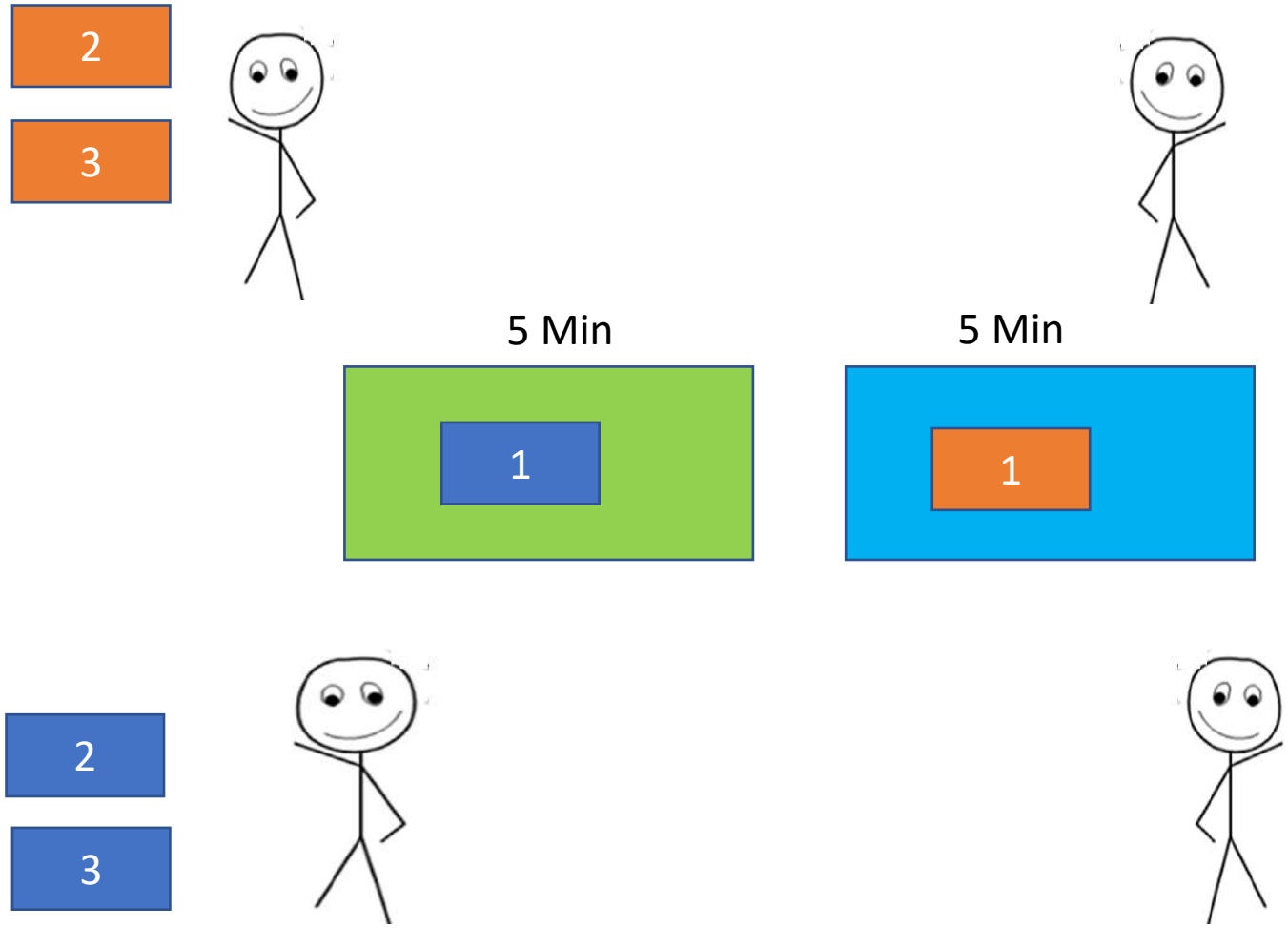
Timer: 5



Technique 2

Shifting One Brick take 10 mins

Timer: 10

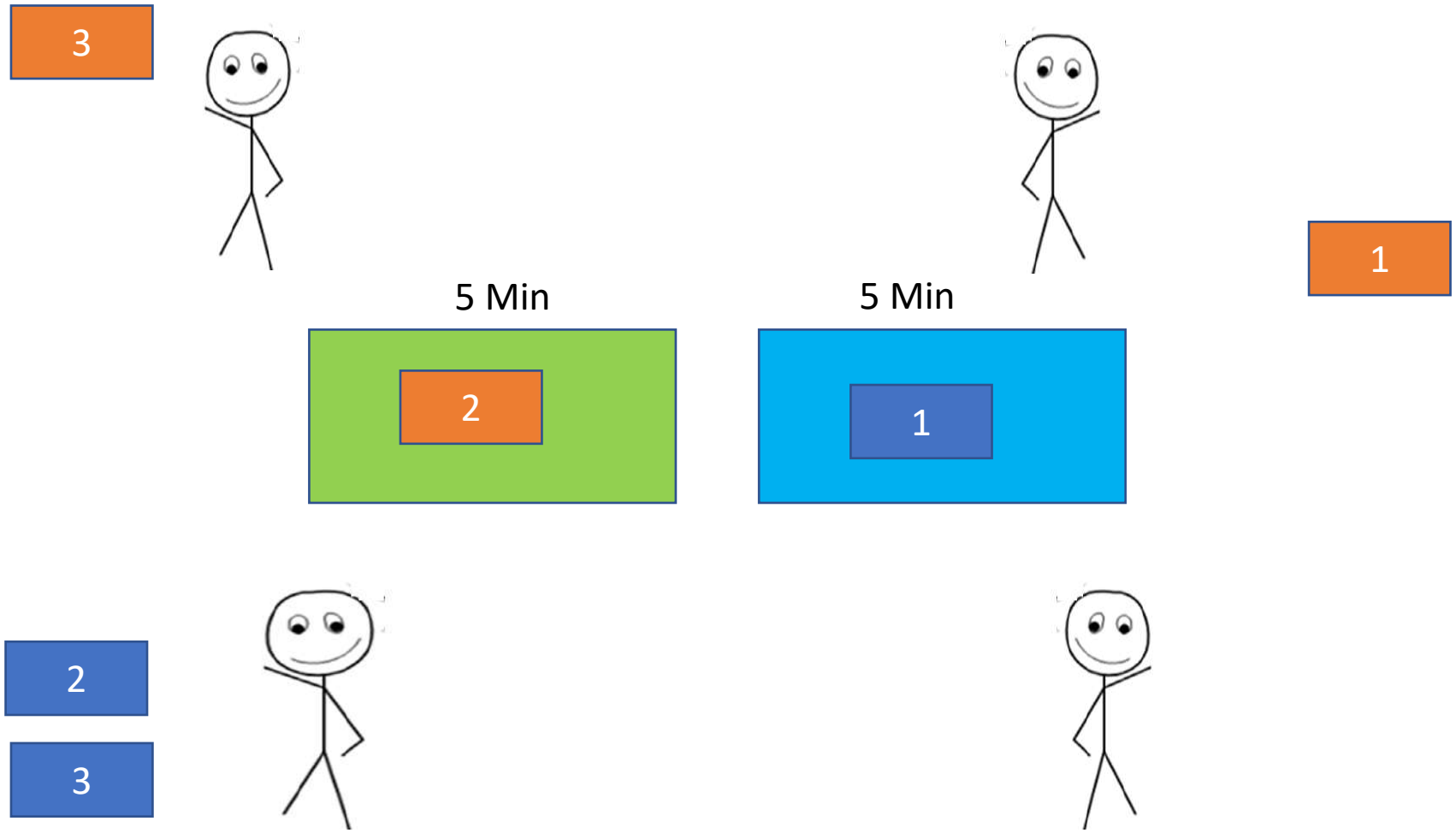


Microprocessor & Computer Architecture (µpCA)

Technique 2

Shifting One Brick take 10 mins

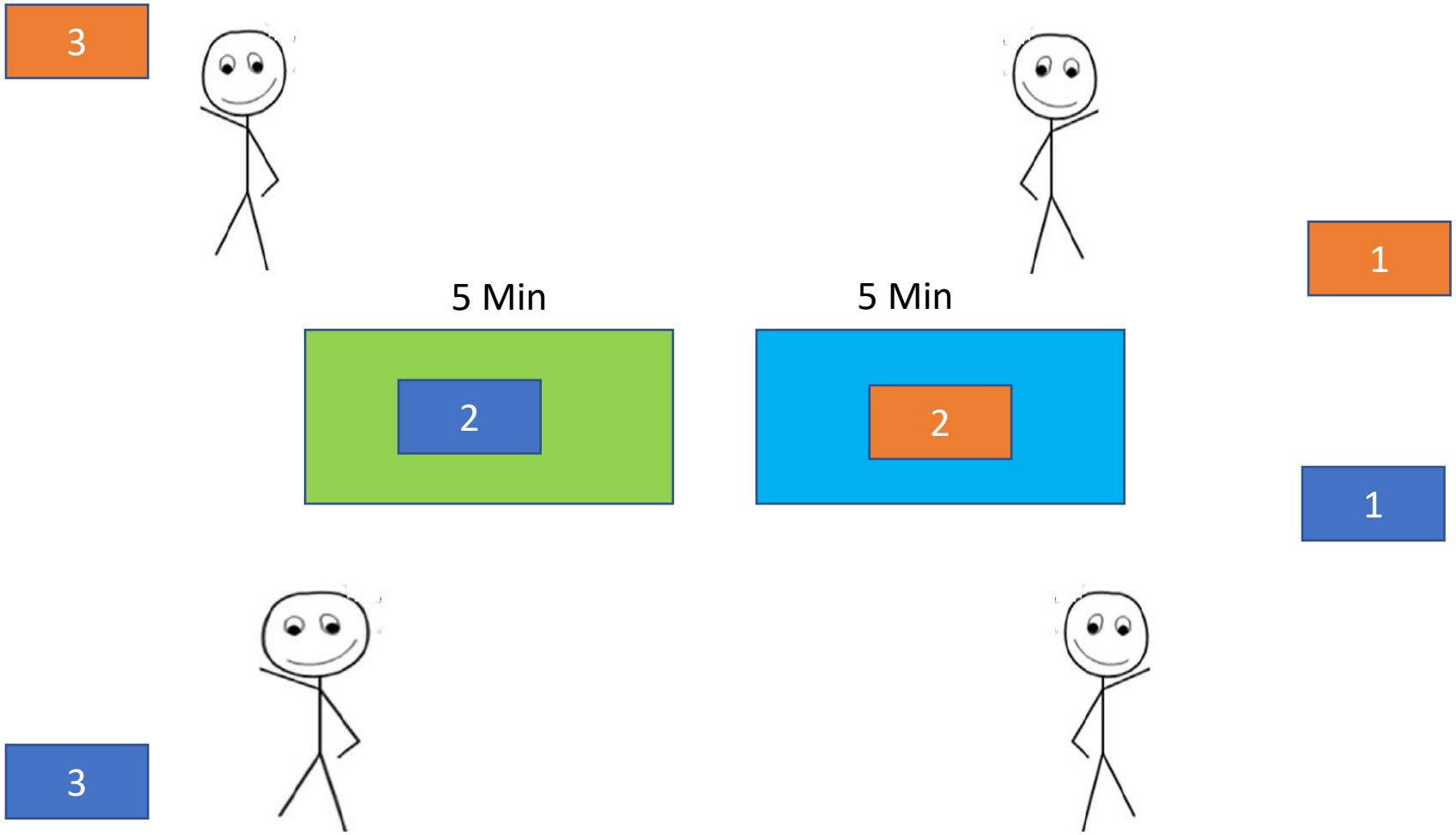
Timer: 10



Technique 2

Shifting One Brick take 10 mins

Timer: 15

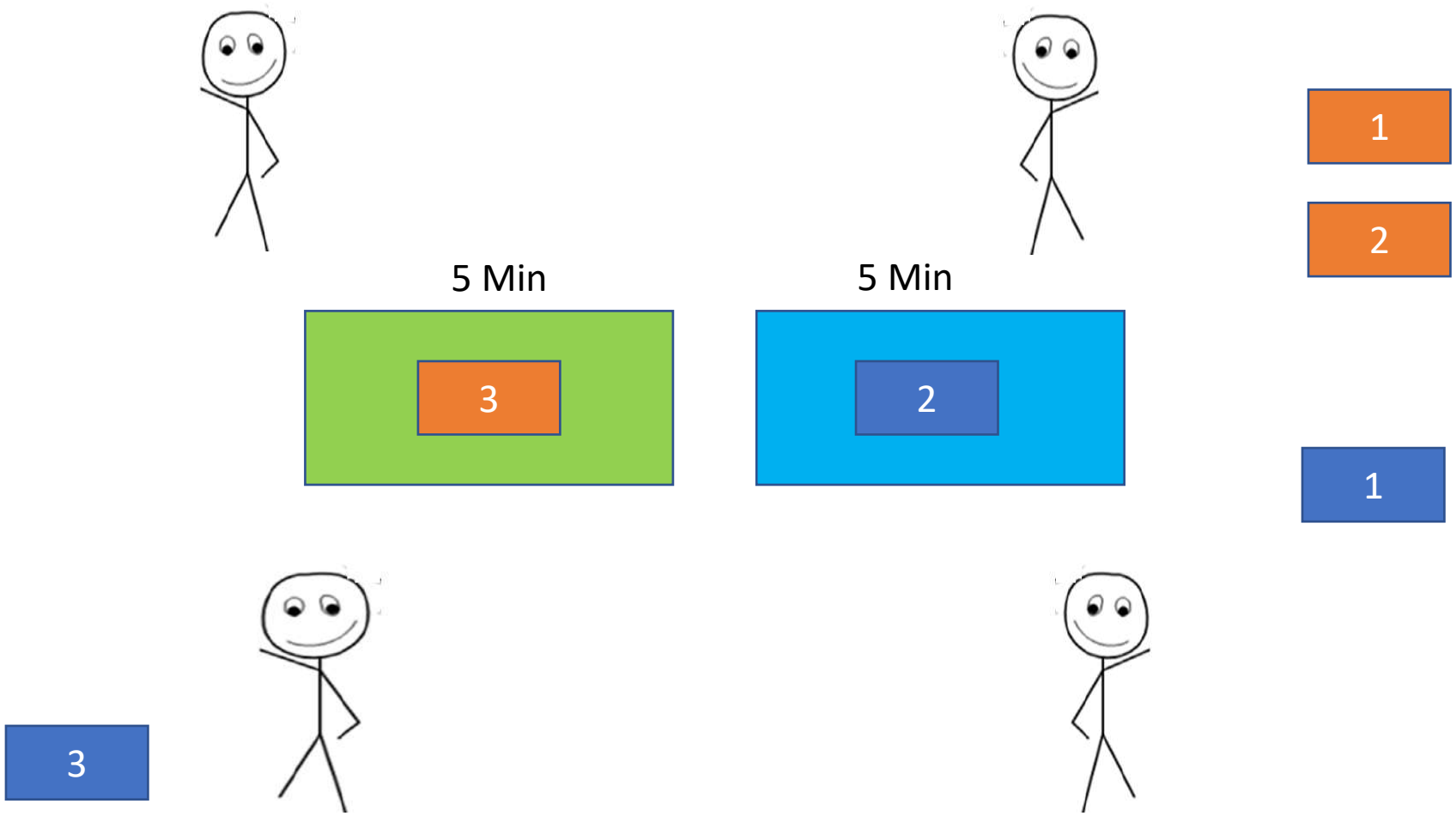


Microprocessor & Computer Architecture (µpCA)

Technique 2

Shifting One Brick take 10 mins

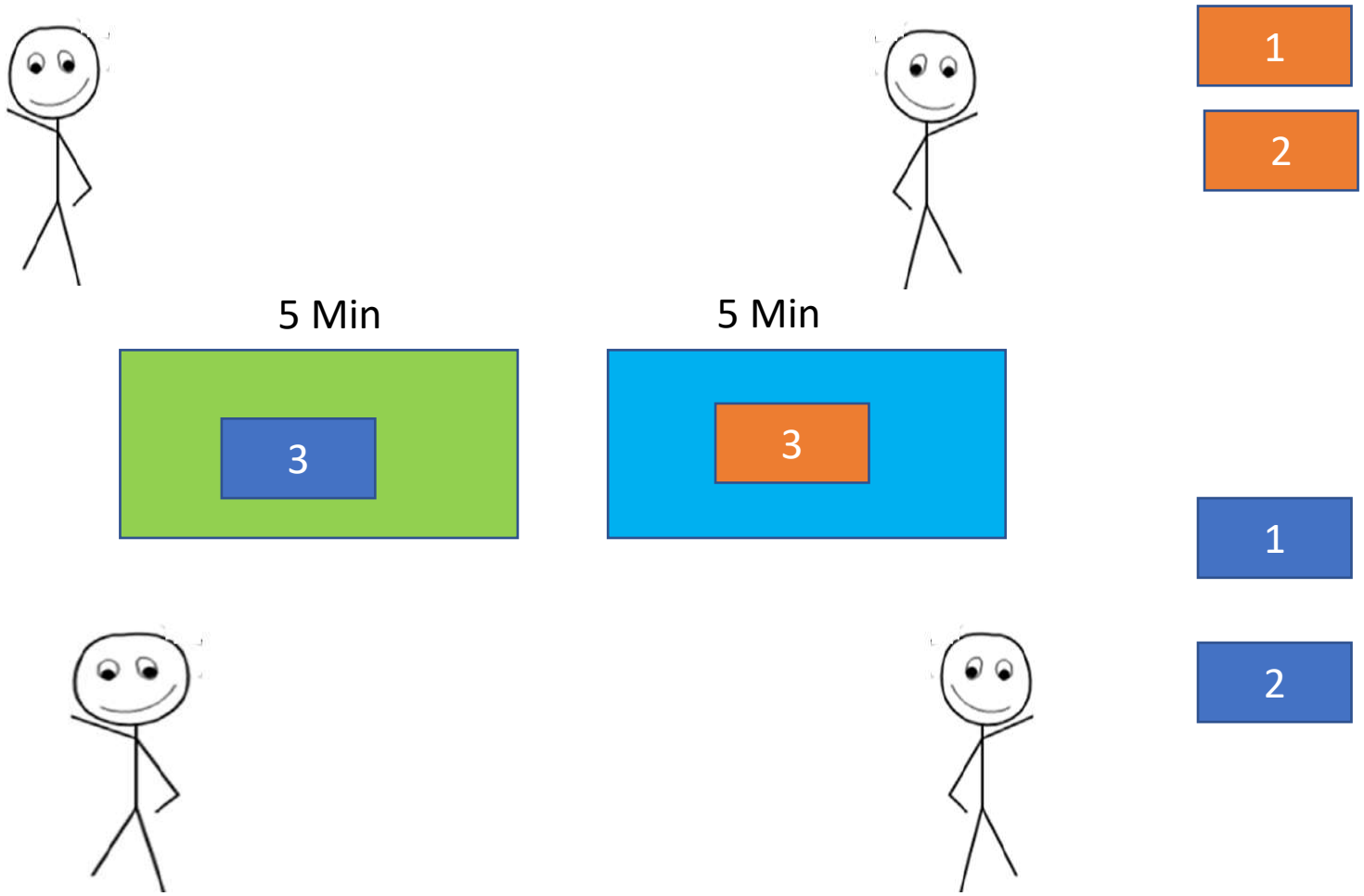
Timer: 20



Technique 2

Shifting One Brick take 10 mins

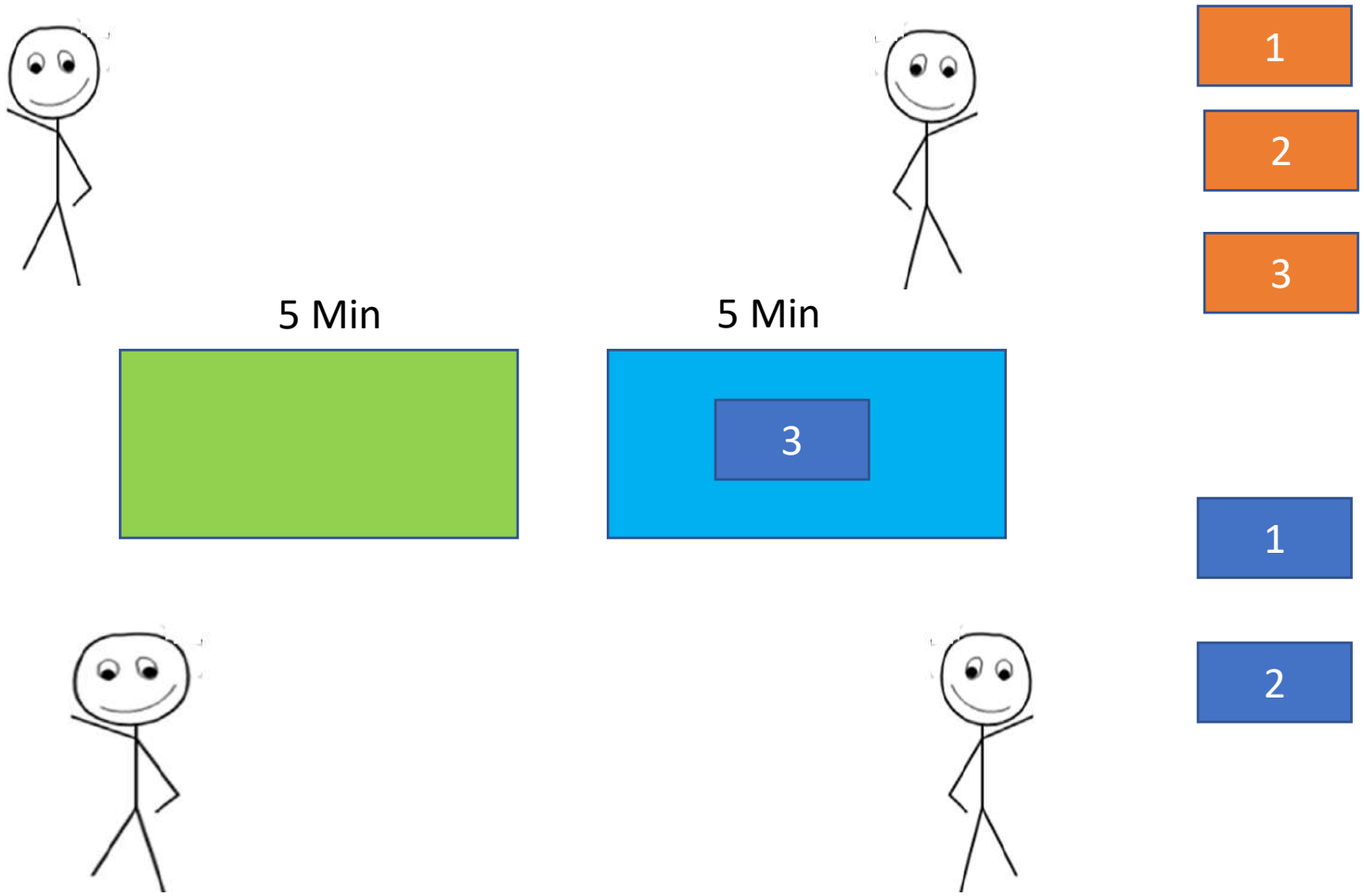
Timer: 25



Technique 2

Shifting One Brick take 10 mins

Timer: 30



Technique 2

Shifting One Brick take 10 mins

Timer: 35

1

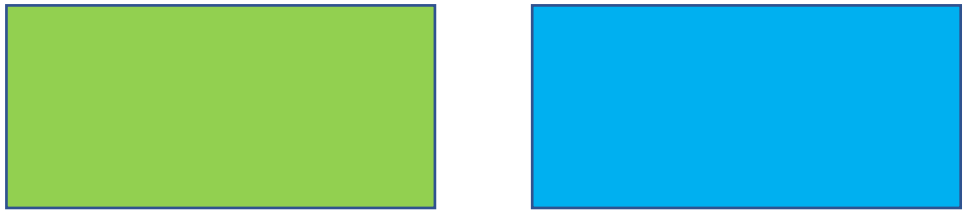
2

3

I took 30 Mins to Complete

5 Min

5 Min



1

I took 30 Mins to Complete

2

3

But we together took 35 Mins to Complete

Technique1 Vs Technique 2

Technique 1: Is called Non-Pipelined Execution

Technique 2: Is called Pipelined Execution

Lesson Learnt

Latency: Time taken to complete the task by each team in both Techniques is 30 Mins each

Throughput: Time taken to complete 2 tasks.

- Technique 1 took 60 Mins
- Technique 2 took 35 Mins.
- Since Resource and Time was shared without overlapping of the task.

Lesson Learnt 2

Time taken to complete the task in **Technique 1** is 6 bricks X 10 Mins= 60 Mins

In **Technique 2**,

1st Brick took 10 Mins

Rest of the bricks were shifted in every 5 Mins

[(1 Brick x 10 Mins)] + [(6-1)*5]= 10+25= 35 Mins

Which Technique Shows Best Performance?

Time taken by Technique 1= 60

Time taken by Technique 2= 35

How better is Technique 2 over Technique 1

$$\frac{\text{Execution time of Technique 1}}{\text{Execution Time of Technique 2}} = \frac{60}{35} = 1.714$$

Technique 2 is ***1.714*** times faster than Technique 1

If Technique 2 is ***n*** times faster than Technique 1

$$n = \frac{\text{Execution Time of Technique 1}}{\text{Execution Time of Technique 2}}$$

Technique1 Vs Technique 2

Also, If Technique 2 is ***n*** times faster than Technique 1

$$n = \frac{\text{Performance of Technique2}}{\text{Performance of Technique1}}$$

$$\text{Performance} = \frac{1}{\text{Execution Time}}$$

Computer X vs Computer Y

If Computer X is ***n*** times faster than Computer Y

$$n = \frac{\text{Execution Time of Computer Y}}{\text{Execution Time of Computer X}}$$

or

If Computer X is ***n*** times faster than Computer Y

$$n = \frac{\text{Performance of Computer X}}{\text{Performance of Computer Y}}$$

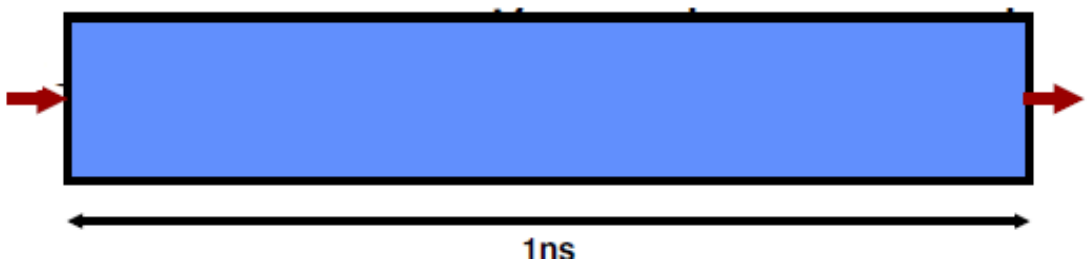
Execution Time

CPU_{Time} = = Instruction Count (IC) X Clock Cycle X CPI

Reducing any of the 3 factors will lead to improve performance or Reduce Execution time is

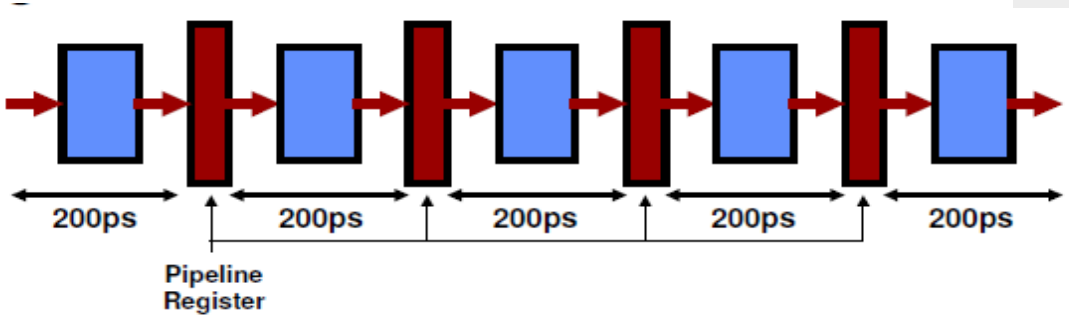
- CPI: Cycles per instruction
- Clock Cycle
- Instruction count

How to Reduce ?



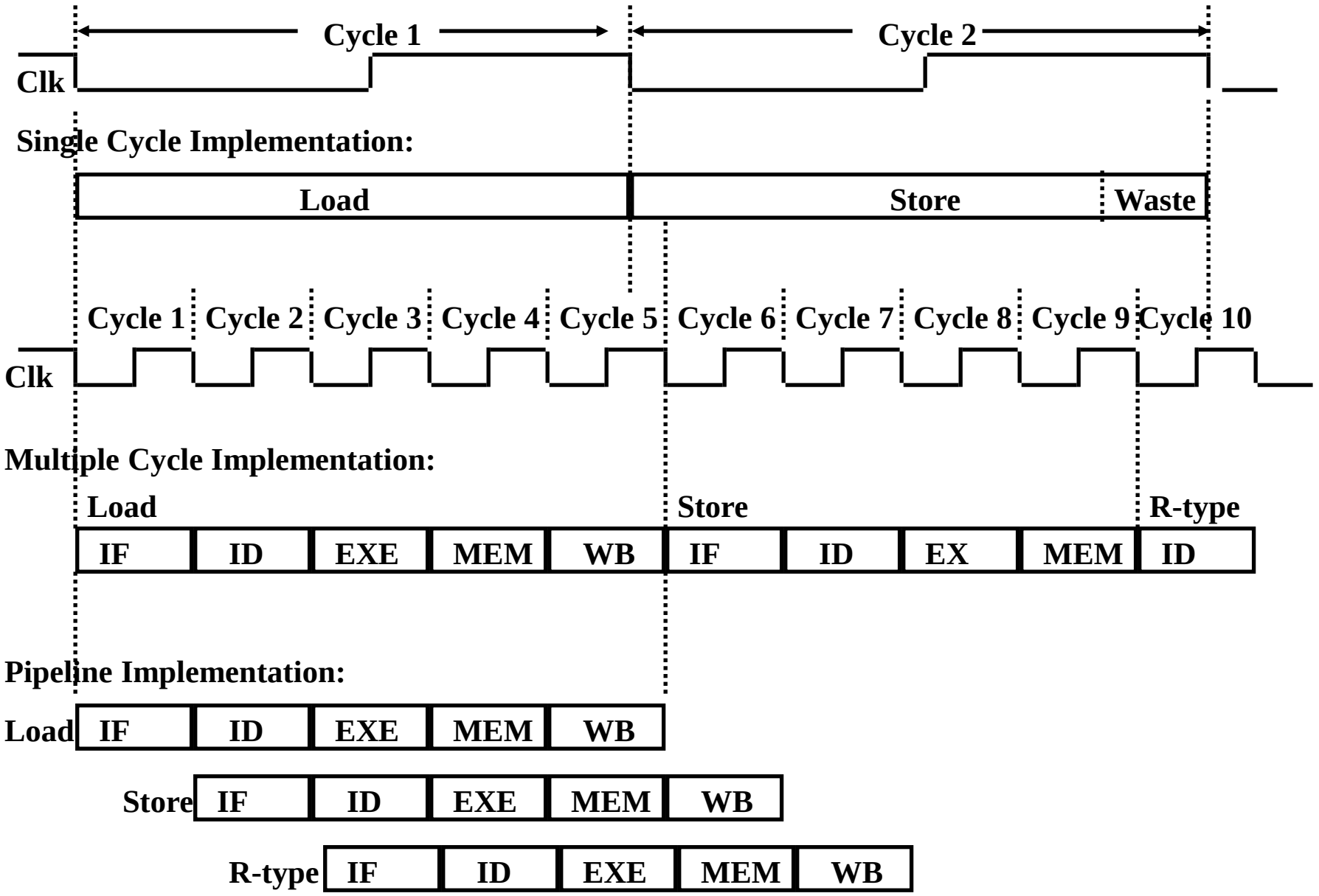
Step 1: Divide instruction execution into multiple stage with small Clock

- **Fetch - [IF]**
- **Decode – [ID]**
- **Execute – [EX]**
- **Buffer/Data or Memory Access-[MEM]**
- **Write back – [WB]**



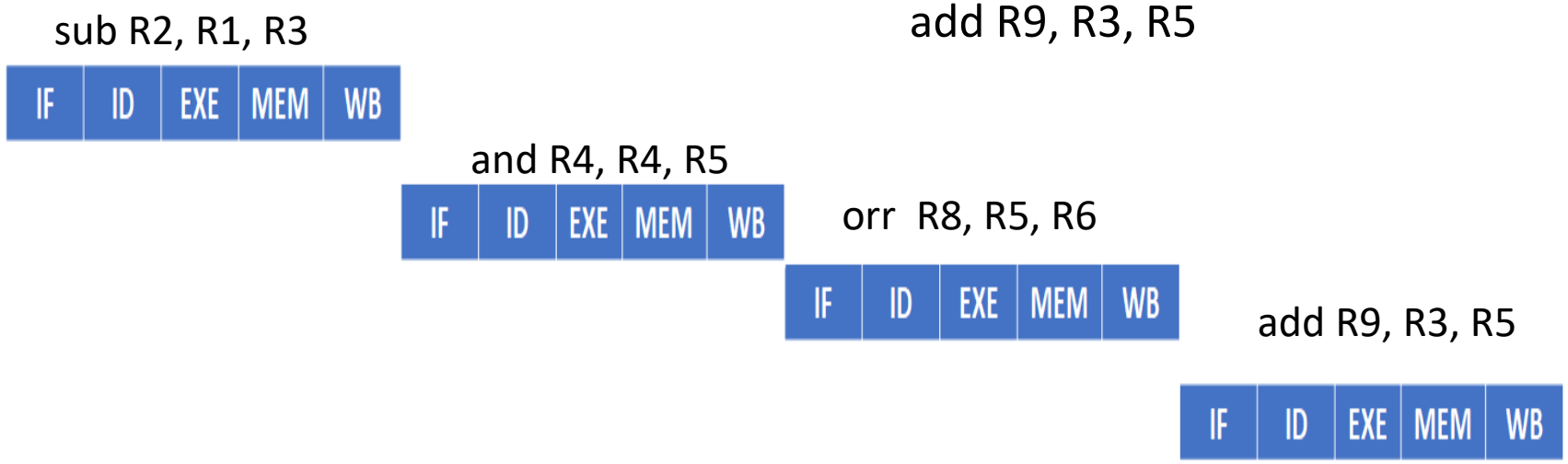
Step 2: Overlap the Execution time of instructions such that, more than one instruction will use different stages in different time slice

Single Cycle, Multiple Cycle, vs. Pipeline



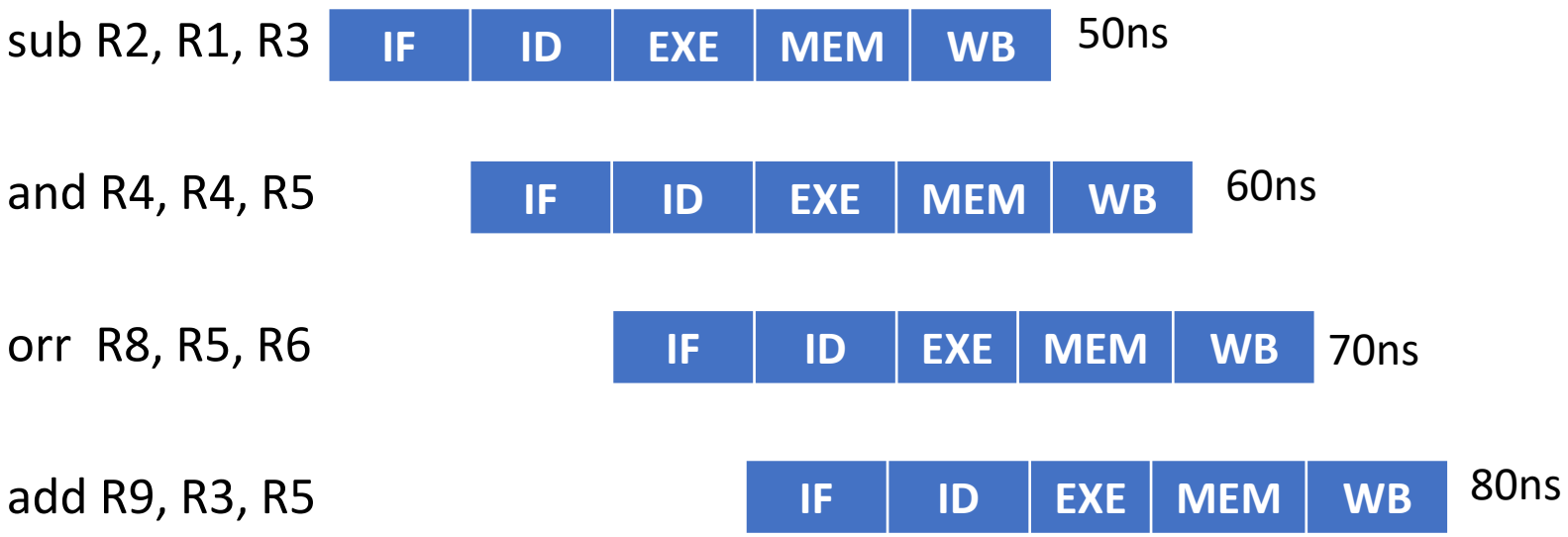
Computer Y Without Overlapping of Time

sub R2, R1, R3
and R4, R4, R5
orr R8, R5, R6
add R9, R3, R5



If Each stage takes 10ns, the latency will be 50 ns and the throughput is 200 ns

Computer X With Overlapping of Time



If Each stage take 10ns, the latency will be 50 ns

If Each stage take 10ns, the throughput is 80 ns instead of 200 ns

Which Computer Shows Best Performance?

Execution Time of Computer Y= 200

Execution Time of Computer X= 80

How better is Computer X over Computer Y

$$\frac{\text{Execution time of Computer Y}}{\text{Execution Time of Computer X}} = \frac{200}{80} = 2.5$$

Computer X is ***2.5*** times faster than Computer Y

Lesson Learnt

Technique 2 is Called Pipelining
or
Computer x has a Pipelined Processor

What May Go Wrong?

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135

Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Introduction to Pipeline Processor

Dr. D. C. Kiran
Department of Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

- ~~3-Stage ARM Processor~~
- ~~5-Stage Pipeline Processor~~
- ~~Introduction to Pipeline Processor~~
- Understanding the Pipeline Execution

Unit 3: **Memory Design**

Unit 4: **Input/Output Device Design**

Unit 5: **Advanced Architecture**

Microprocessor & Computer Architecture (µpCA)

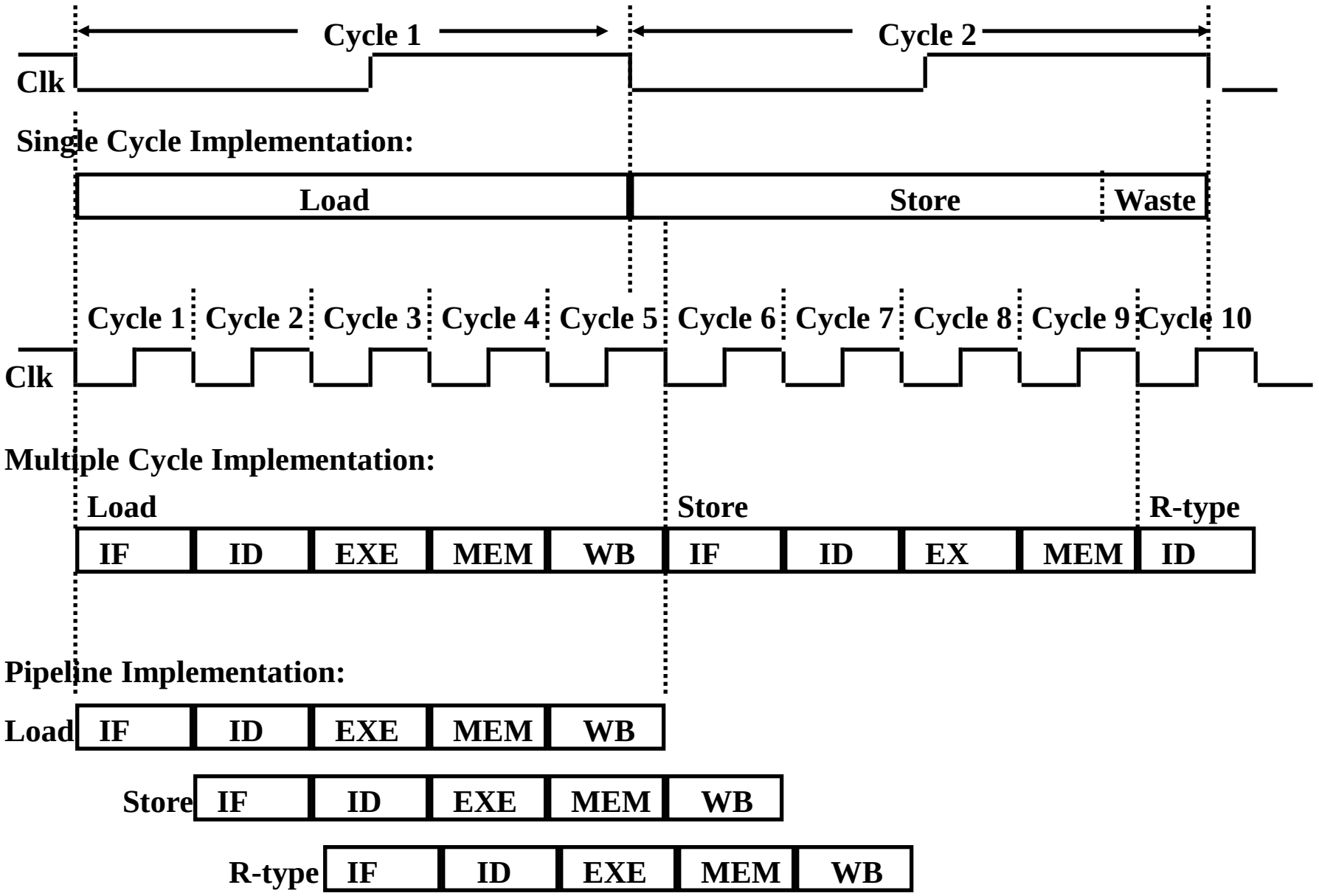
Text 1: "Computer Organization and Design", Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

Reference 1:"Computer Architecture: A Quantitative Approach", Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.

Appendix C **Pipelining: Basic and Intermediate Concepts**

C.1	Introduction	C-2
C.2	The Major Hurdle of Pipelining—Pipeline Hazards	C-11
C.3	How Is Pipelining Implemented?	C-30
C.4	What Makes Pipelining Hard to Implement?	C-43
C.5	Extending the MIPS Pipeline to Handle Multicycle Operations	C-51
C.6	Putting It All Together: The MIPS R4000 Pipeline	C-61
C.7	Crosscutting Issues	C-70
C.8	Fallacies and Pitfalls	C-80
C.9	Concluding Remarks	C-81
C.10	Historical Perspective and References	C-81
	Updated Exercises by Diana Franklin	C-82

Single Cycle, Multiple Cycle, vs. Pipeline



Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

5*1*1

(5-1)*1

- 1ST instruction take 5 CLOCK (Each stage, 1 clock time Tc)
- Rest will take 1 more than the previous instruction
- **Execution time on pipeline processor: =**Kstage * 1 instuction + (n-1) :
  = 5*1+4= 9 clocks
- **Execution on a non-pipeline processor:** =Kstage * n Instructions
  = 5*5= 25 clocks

Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

k*tc*1

(n-1)*tc

Number of stages= k

Clock Cycle = tc

Number of Instructions = n

- Execution time$_{pipeline}$ = k*tc*1 + (n-1)*tc=[k+(n-1)]t_c
- Execution time$_{unpipeline}$ = n* t_p =n*k* t_c

Speedup vs # of Stages in Pipeline Processor

- $Speedup(S) = \dfrac{Execution\ Time\ on\ Unpipelined}{Execution\ Time\ on\ Pipelined}$

$$\frac{n\mathrm{t_p}}{(k+n-1)\mathrm{t}c} \rightarrow \frac{n\mathrm{t_p}}{(k-1+n)\mathrm{t}c}$$

If n is too big or as number of instructions increases n>k-1 will tend to n

$$\text{Speedup (S)} = \frac{n\mathrm{t_p}}{n\mathrm{t}c}$$

$$\text{Speedup (S)} = \frac{\mathrm{t_p}}{\mathrm{t}c}$$

$$\text{Speedup (S)} = \frac{k*\mathrm{t}c}{\mathrm{t}c}$$

$$\text{Speedup (S)} = \mathrm{k}$$

Pipeline Execution

Compute the Execution time on 5 Stage processor (Pipeline vs Non Pipeline Processor):

Number of instructions = 100.

Cycle Time Tc= 60

Solution:

Execution on a non-pipeline processor: = K_{stage}* Tc* n Instructions

= 5*60*100= 30000 clocks

Execution Time on Pipeline Processor: = K_{stage} * 1 instruction *Tc + (n-1)* Tc :

= (5*1*60)+(99*60)= 300+5940=6240 clocks

Clock time per stage= Cycle time= Tc

Speedup vs # of Stages in Pipeline Processor

- $Speedup(S) = \frac{Execution\ Time\ on\ Unpipelined}{Execution\ Time\ on\ Pipelined}$

- $Speedup(S) = \frac{30000}{6240}$

- $Speedup(S) = 4.8 \approx 5$

Theoretical Claim

$$\text{Time taken}_{\text{pipeline}} = \frac{Time\ taken\ on\ unpipelined}{No.of\ pipeline\ stages}$$

$$\text{Time taken}_{\text{pipeline}} = 30000/5 = 6000 \approx 6240$$

Design issue 1

Clock Cycle Time of all the stages cannot be same!

Example

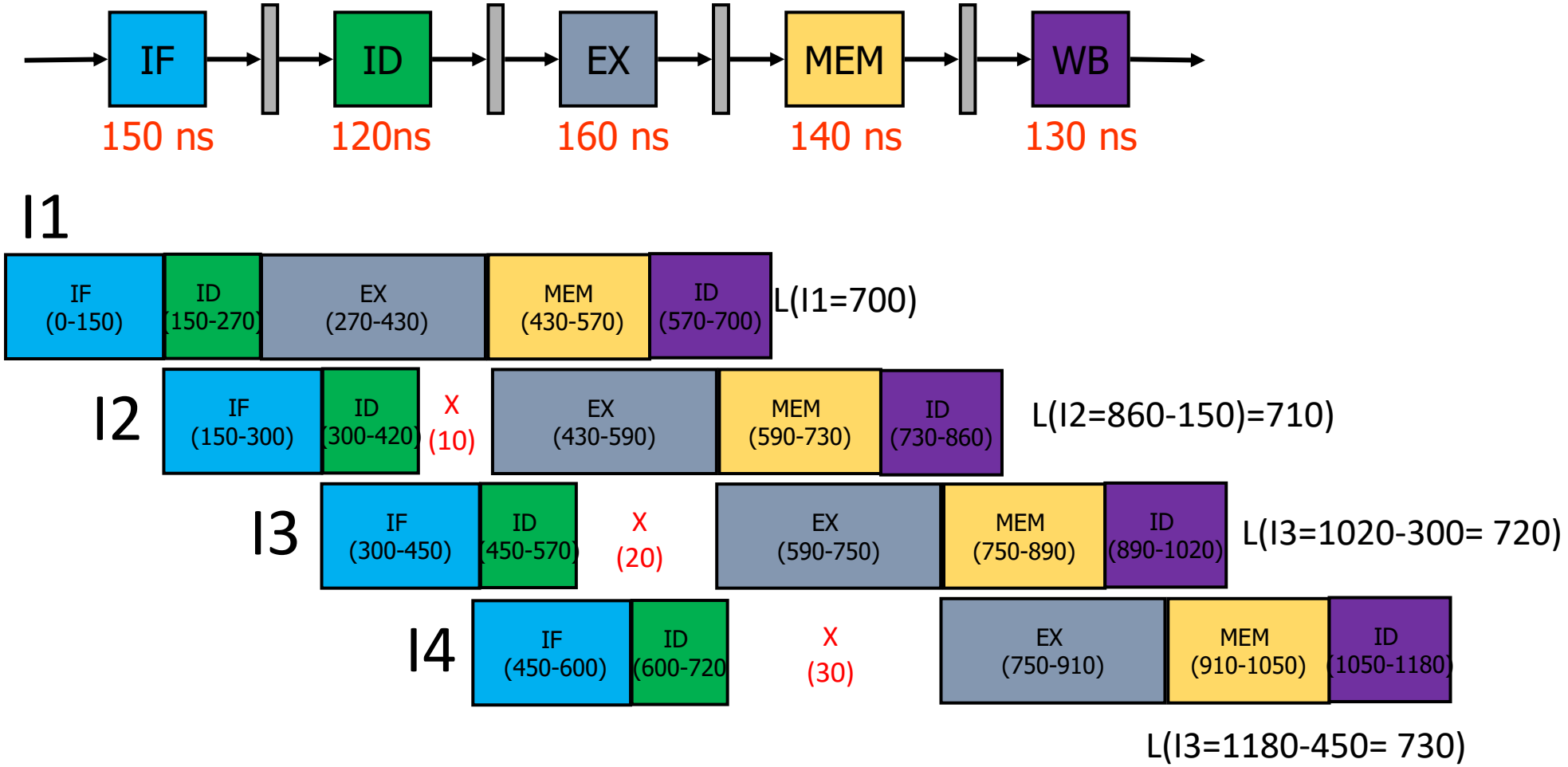
IF	ID	EXE	MEM	WB
300 ps	400 ps	350 ps	550 ps	100 ps
200 ps	150 ps	100 ps	190 ps	140 ps

Leads to imbalance latency

Solution:

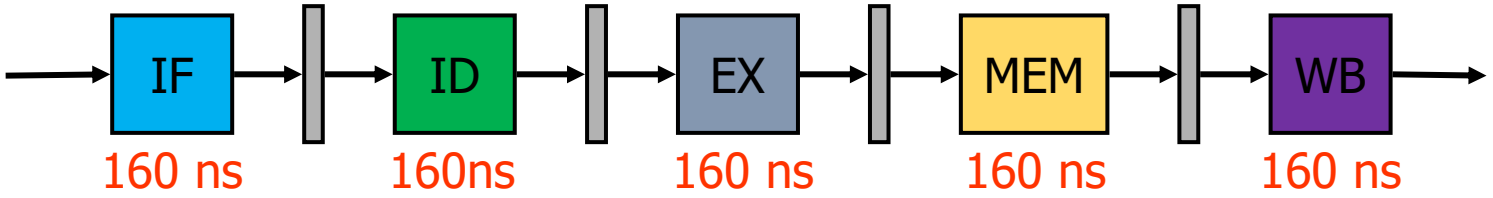
Make Length of each stage equal to Length of Longest stage or slowest Stage

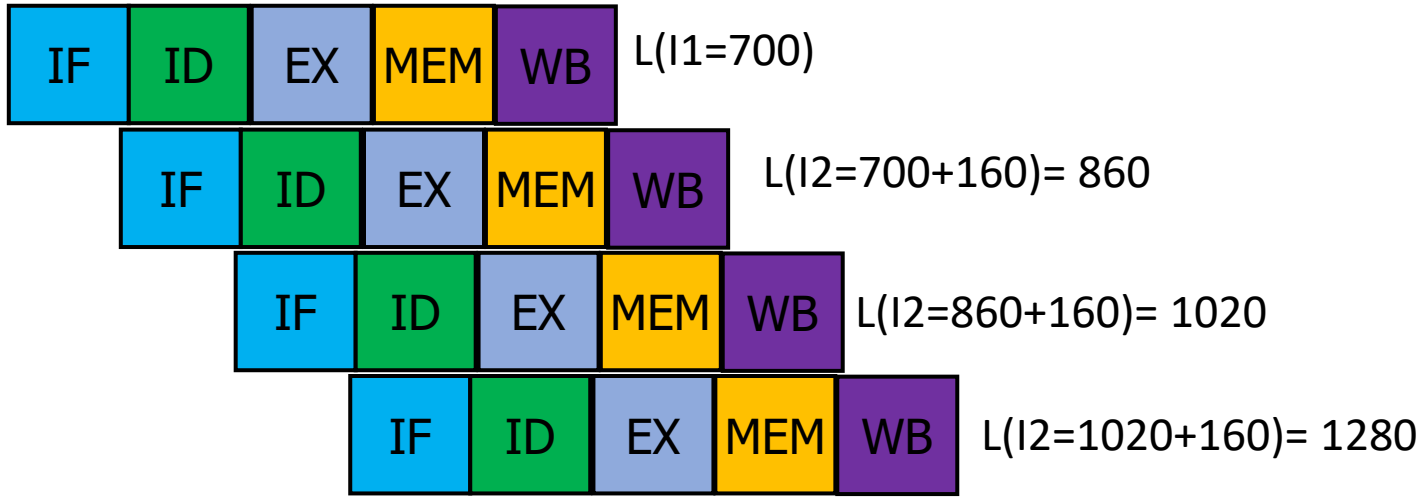
Pipeline Throughput and Latency



Uneven latency and Throughput= 1180

Pipeline Throughput and Latency





Balanced latency and Throughput= 1280 instead of 1180

Evaluating 5-Stage Pipeline Processor

A 5 stage pipeline processor has stage delays as 150, 120, 160, 140 and 130 ns.
What is the time taken to execute 100 instructions.
What is the Speed up of pipeline processor?

Solution:

Execution on a non-pipeline processor: (150+120+160+140+130) * 100= 700*100=70000ns

Execution on a pipeline processor:

Slowest stage: Max(150,120,160,140,130)= 160ns
Clock time Tc of each stage = 160 ns

=(1*Tc*#stages)+((IC-1)*Tc)
=(1*160*5)+(99*160)
= 800+15840
= 16640 ns

Performance of 5-Stage Pipeline Processor

Execution on a non-pipeline processor

(150+120+160+140+130) * 100= 362*100=70000ns

Execution on a pipeline processor:

(1*160*5)+(99*160)= 800+15840= 16640ns

Speedup(S): 70000/16640= 4.2

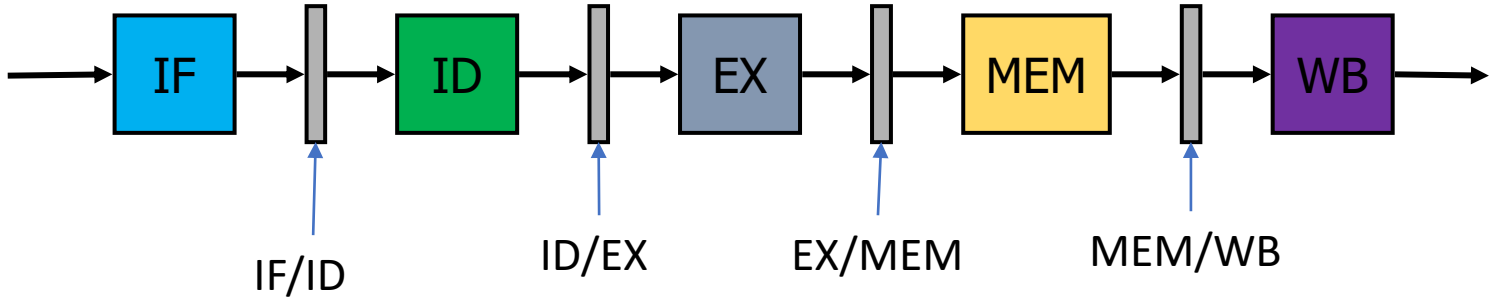
Theoretical Claim

$$\text{Time taken}_{\text{pipeline}} = \frac{Time\ taken\ on\ unpipelined}{No.of\ pipeline\ stages}$$

Performance of 5-Stage Pipeline Processor

$$16640 = \frac{(160*5*100)}{5} = \frac{80000}{5} \approx 16000$$

Design issue 2: Pipeline Register Overhead



- We shall refer to the pipeline registers that are set between two stages with the names of the stages. So, we will have IF/ID, ID/EX, EX/MEM and MEM/WB registers.
- They serve the purpose of transferring outputs produced in a phase to the subsequent phase in the multi-cycle implementation.
- These registers must be large enough to contain all data moving from one phase to the following one

Design issue 2: Pipeline Register Overhead

Pipeline overhead: combination of pipeline register delay and the clock skew.

- Pipeline registers delay: Setup time that triggers a write or when data input changes and propagation delay to the clock.
- Clock skew: Maximum delay between when the clock arrives at any two registers.

Reference

Evaluating 5-Stage Pipeline Processor With Register Overhead

A 5 stage pipeline processor has stage delays as 150, 120, 160, 140 and 130 ns.
The register overhead is 5 ns each.
What is the time taken to execute 100 instructions.
What is the Speed up of pipeline processor?

Solution:

Execution on a non-pipeline processor: (150+120+160+140+130) * 100= 700*100=70000ns

Execution on a pipeline processor:

Slowest stage: Max(150,120,160,140,130)= 160ns
Clock time Tc of each stage = 160 ns
Register overhead of each stage= 5ns
Clock time Tc of each stage = 165 ns

Execution on a pipeline processor: =(1*Tc*#stages)+((IC-1)*Tc)
=(1*165*5)+(99*165)
= 825+16335
= 17160 ns

Performance of 5-Stage Pipeline Processor

Execution on a non-pipeline processor

(150+120+160+140+130) * 100= 362*100=70000ns

Execution on a pipeline processor:

(1*165*5)+(99*165)= 825+15840= 17160ns

Speedup(S): 70000/171600= 4.07

Theoretical Claim

$$\text{Time taken}_{\text{pipeline}} = \frac{Time\ taken\ on\ unpipelined}{No.of\ pipeline\ stages}$$

Performance of 5-Stage Pipeline Processor

$$17160 = \frac{(165*5*100)}{5} = \frac{82500}{5} \approx 16500$$

What Else May Go Wrong?

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135

Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Pipeline Processor- Hazards

Dr. D. C. Kiran
Department of Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

- ~~3-Stage ARM Processor~~
- ~~5-Stage Pipeline Processor~~
- ~~Introduction to Pipeline Processor~~
- ~~Understanding the Pipeline Execution~~
- What May Go Wrong
  - Introduction to Hazards
  - Attacking Hazards
  - Performance with Stalls
  - Structural Hazards

Unit 3: **Memory Design**

Unit 4: **Input/Output Device Design**

Unit 5: **Advanced Architecture**

Microprocessor & Computer Architecture (µpCA)

Text 1: "Computer Organization and Design", Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

Reference 1:"Computer Architecture: A Quantitative Approach", Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.

Appendix C **Pipelining: Basic and Intermediate Concepts**

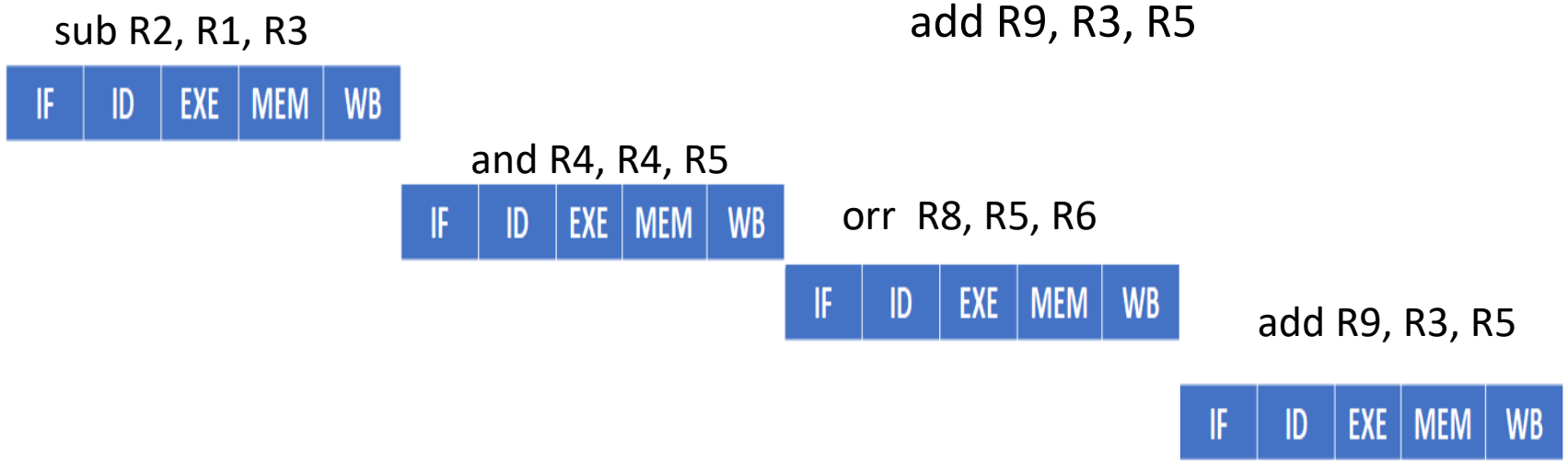
C.1	Introduction	C-2
C.2	The Major Hurdle of Pipelining—Pipeline Hazards	C-11
C.3	How Is Pipelining Implemented?	C-30
C.4	What Makes Pipelining Hard to Implement?	C-43
C.5	Extending the MIPS Pipeline to Handle Multicycle Operations	C-51
C.6	Putting It All Together: The MIPS R4000 Pipeline	C-61
C.7	Crosscutting Issues	C-70
C.8	Fallacies and Pitfalls	C-80
C.9	Concluding Remarks	C-81
C.10	Historical Perspective and References	C-81
	Updated Exercises by Diana Franklin	C-82

Facts About Pipeline Processor

- Pipelining increases the CPU instruction throughput. ***(Pipeline Throughput)*** Ideally, CPI = 1.
- Pipelining does not reduce the execution time of an instruction. ***(Pipeline Latency)***
- Infact, it slightly increases the execution time due to the increased **control overhead** of the pipeline stage register delays.
- All instructions that share a pipeline must have the same stages in the same order.
  - ✓ **Add** does nothing during Mem stage
  - ✓ **sw** does nothing during WB stage
  - ✓ **b{cond}** does nothing in EX, MEM & WB stage

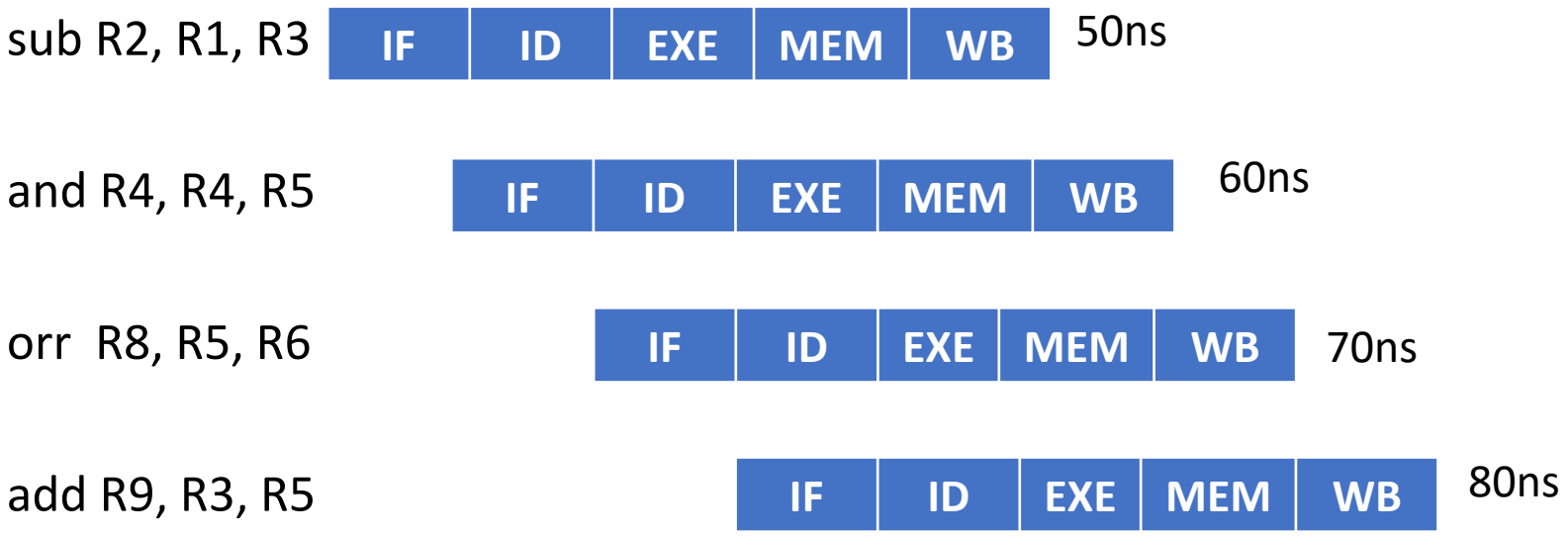
Non-Pipeline Execution

sub R2, R1, R3
and R4, R4, R5
orr R8, R5, R6
add R9, R3, R5



If Each stage takes 10ns, the latency will be 50 ns and the throughput is 200 ns

Pipelining is Good



If Each stage take 10ns, the latency will be 50 ns

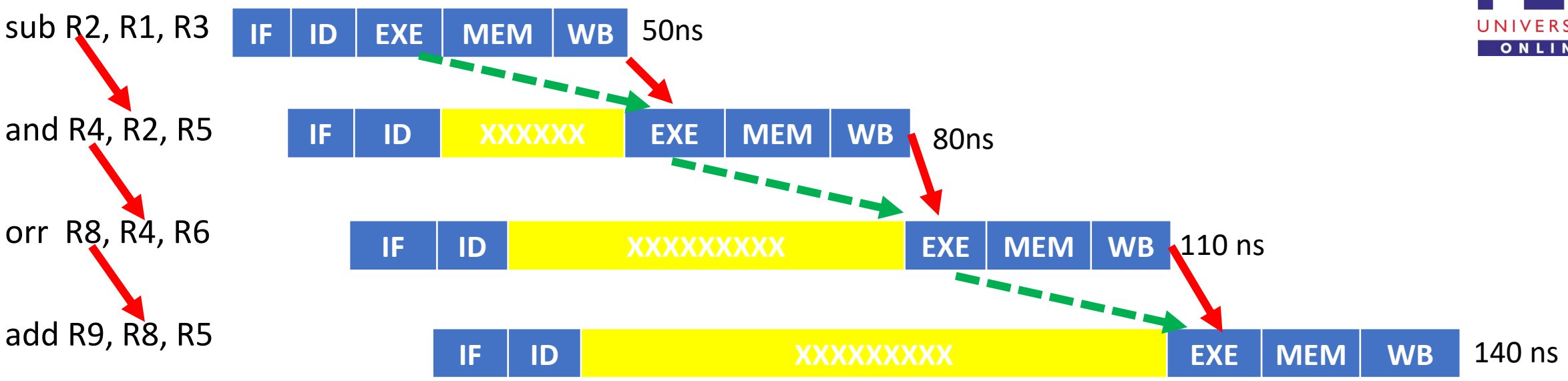
If Each stage take 10ns, the throughput is 80 ns instead of 200 ns

Pipelining, What May Go Wrong?

- Is Good, When every instruction is independent of the other and can execute in the pipeline without any constraint
- But in application programs, most of the statements are dependent on each other
  - C = A + B;
  - E = C * D;
- This causes constraint on the pipeline to execute these dependent instructions in given 5 cycles.
  - Increases Latency

Microprocessor & Computer Architecture (µpCA)

Pipelining, What May Go Wrong?



If Each stage take 10ns, the latency will be 50 ns

If Each stage take 10ns, the throughput is 80 ns instead of 200 ns

With gap between stages, the throughput is 140 ns instead of 80ns

Pipelining, What May Go Wrong?→ Hazards

- Situations that prevent the next instruction in the instruction stream from executing during its designated cycle
- Condition that prevents an instruction in the pipe from executing its next scheduled pipe stage
- Where one instruction cannot **immediately** follow another

- **Hazards reduce the performance from the ideal speedup gained by pipelining**

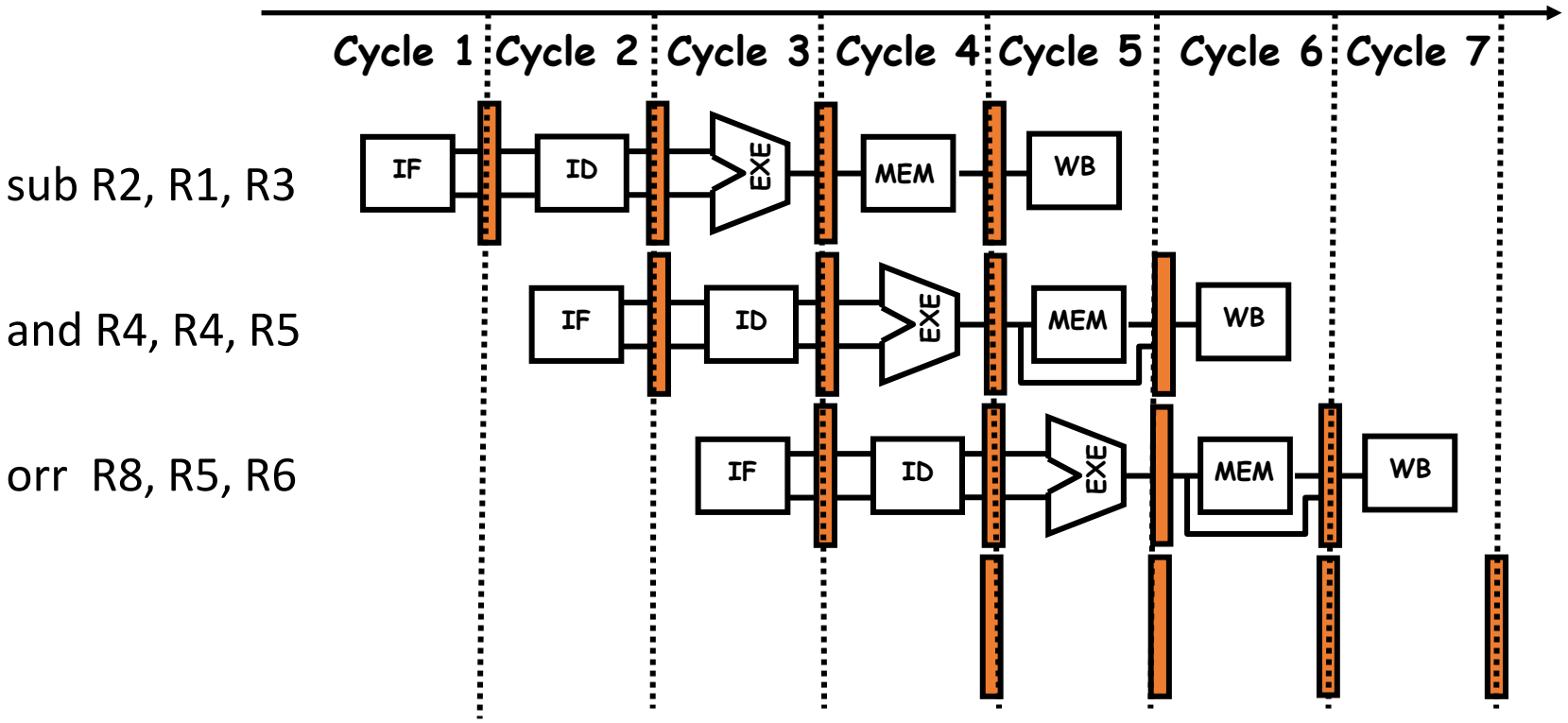
Hazards: Limits to Pipelining

- ***Structural Hazard:***
  - Due to structure of the pipeline
  - Hardware cannot support combination of instructions in the pipeline needing the same resources
- ***Data Hazard:***
  - Due to data dependencies between instructions
  - Instruction depends on the result of prior instruction still in the pipeline
- ***Control Hazard:***
  - Due to control instructions
  - Pipelining of branches & other instructions that change the PC

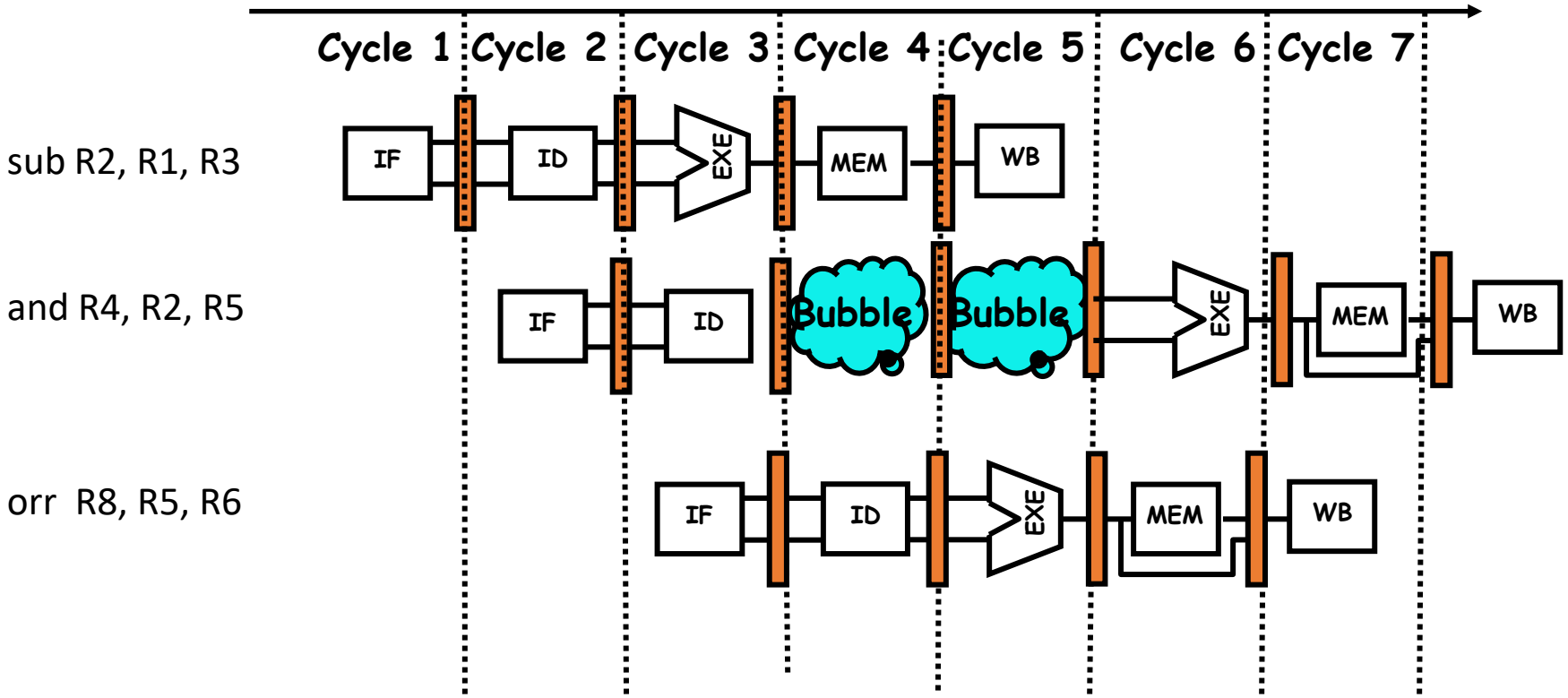
Stalls

- Common solution for any type of hazard is to **stall** the pipeline until the hazard is resolved (Stall Cycles)
- This is achieved by inserting one or more **"bubbles"** in the pipeline
- To do this, hardware or software must **detect** that a hazard has occurred (Hazard Detection)

Stalls


Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
sub R2, R1, R3
and R4, R4, R5
orr R8, R5, R6
IF
ID
EXE
MEM
WB

Stalls


Cycle 1
Cycle 2
Cycle 3
Cycle 4
Cycle 5
Cycle 6
Cycle 7
sub R2, R1, R3
IF
ID
EXE
MEM
WB
and R4, R2, R5
IF
ID
Bubble
Bubble
EXE
MEM
WB
orr R8, R5, R6
IF
ID
EXE
MEM
WB

Stalls

Instruction Number	Clock number									
	1	2	3	4	5	6	7	8	9	10
sub R2, R1, R3	IF	ID	EX	MEM	WB					
and R4, R2, R5		IF	ID	**stall**	**stall**	EX	MEM	WB		
orr R8, R5, R6			IF	ID	EX	MEM	WB			

Disadvantage

- Stall cycle is a waste cycle
  - Opportunity wasted to execute one instruction
- Adds to pipeline delay and hence increases CPI
  - (CPI > 1)
  - Ex: If 30% are load/store, with only one memory; CPI ≥ 1.3
- Not a preferred solution
- Affects performance of pipeline architecture

Performance of Pipelines with Stalls

A stall causes the pipeline performance to degrade the ideal performance.

$$\text{Speedup from pipelining} = \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}}$$

$$= \frac{CPI_{unpipelined} * \text{Clock Cycle Time}_{unpipelined}}{CPI_{pipelined} * \text{Clock Cycle Time}_{pipelined}}$$

$$= \frac{CPI_{unpipelined}}{1 + \text{Pipeline stall cycles per instruction}}$$

Note 1: Ignoring the cycle time overhead of pipelining and assume the stages are all perfectly balanced, then the cycle time of the two machines are equal

Note 2: Ideal CPI of pipeline processor is almost always =1

$CPI_{pipelined}$ = Ideal CPI + Pipeline stall clock cycles per instruction

= 1 + Pipeline stall clock cycles per instruction

Performance of Pipelines with Stalls

If all instructions take the same number of cycles, which must also equal the number of pipeline stages (the depth of the pipeline) then unpipelined CPI is equal to the depth of the pipeline

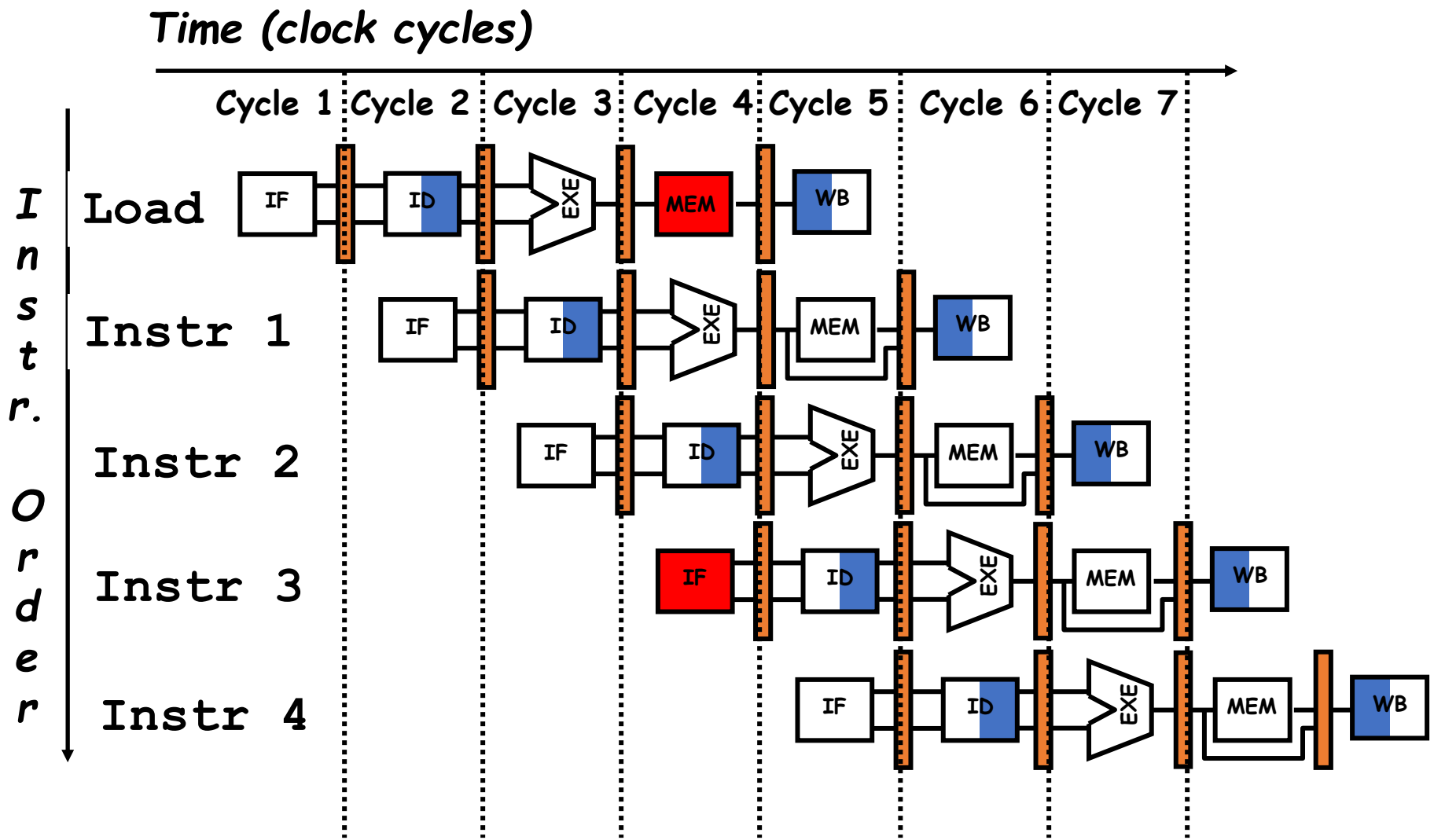
$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall cycles per instruction}}$$

If there are no pipeline stalls, this leads to the intuitive result that pipelining can improve performance by the depth of pipeline.

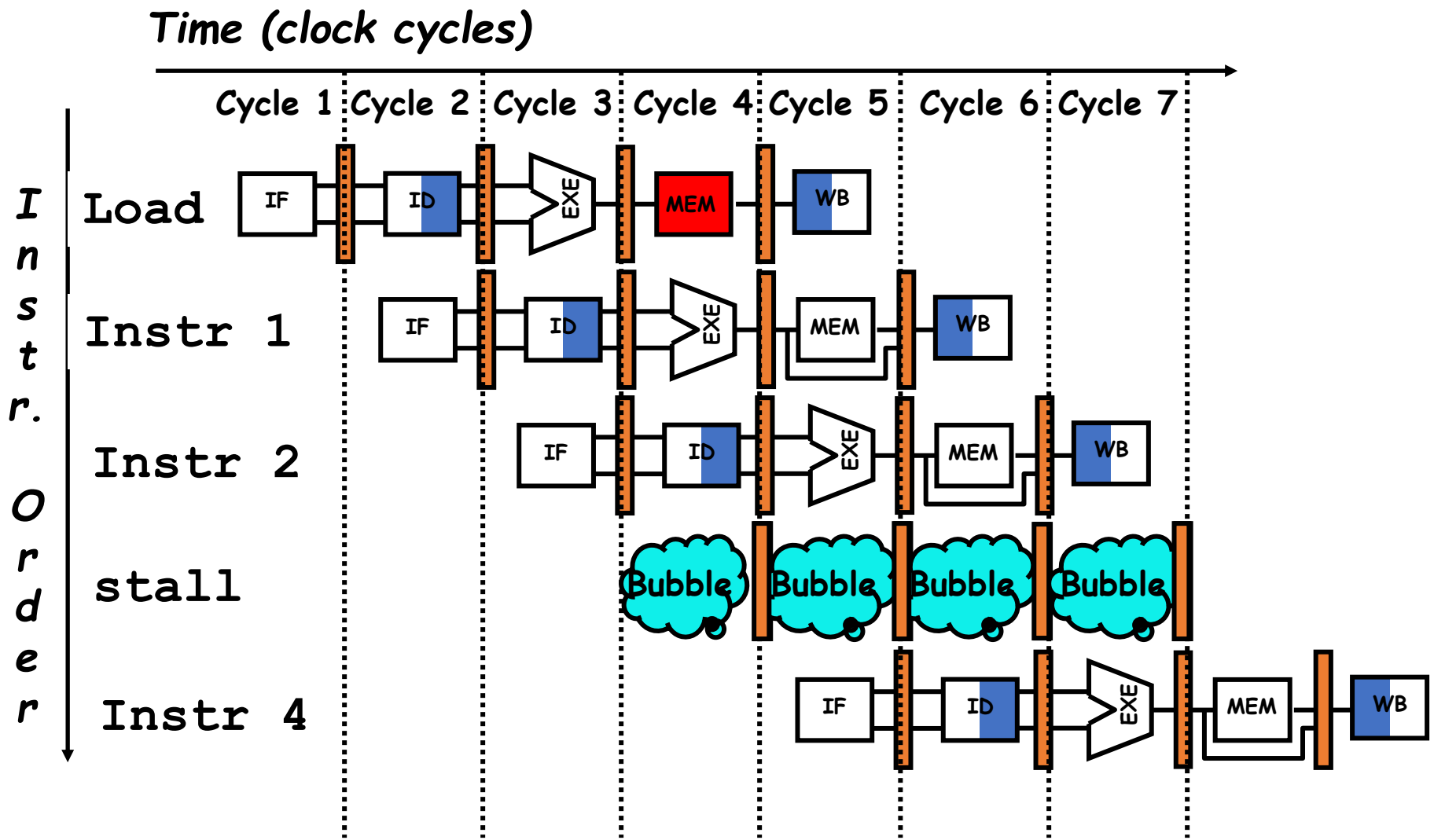
Structural Hazard

- If a resource conflict arises due to a hardware resource being required by more than one instruction in a single cycle, and one or more such instructions cannot be accommodated, then a structural hazard occurs, for example:
  - **when a pipelined machine has a shared single-memory for data and instructions.**
    - **Unified Memory**
  - **when a machine has only one register file write port**
    - **Skip MEM stage for add/sub..**
  - **when a machine has overlapping of read and write**
    - **Overlap ID & WB**

Structural Hazard: IF VS MEM:

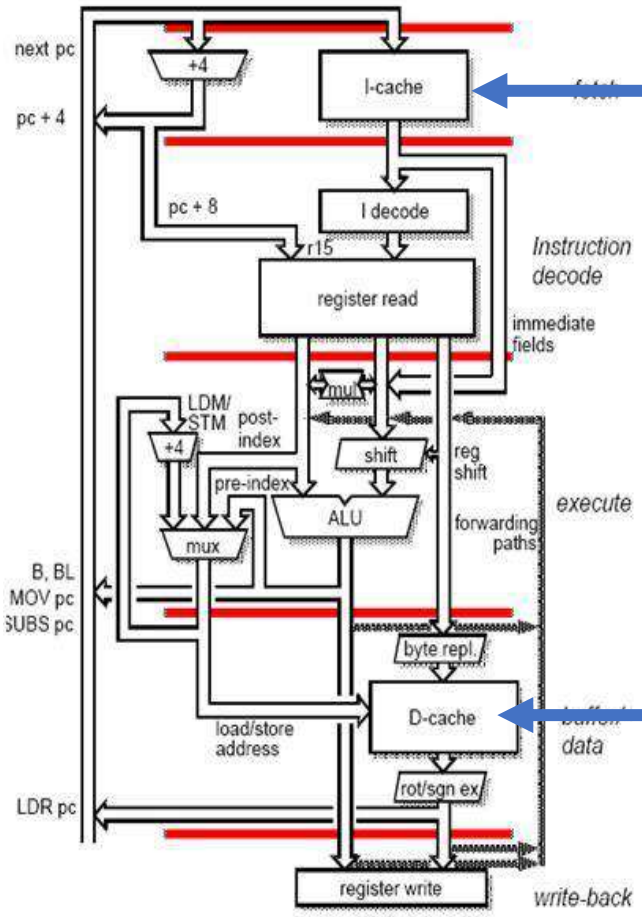


Structural Hazard: IF VS MEM:



Structural Hazard: IF VS MEM: Solution

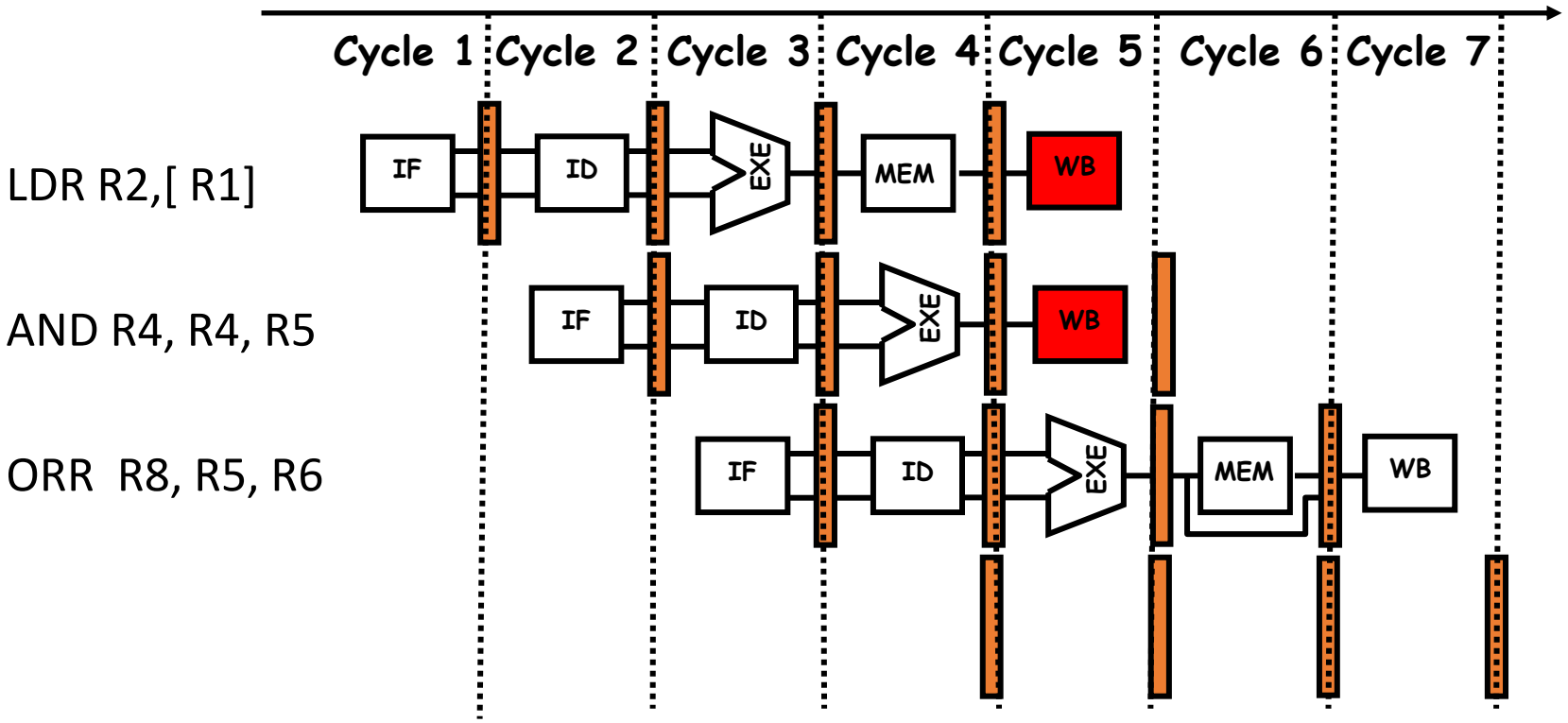
- **Replication of resources**
  - Split Memory (Instruction Memory & Data Memory)



Instruction Memory

Data Memory

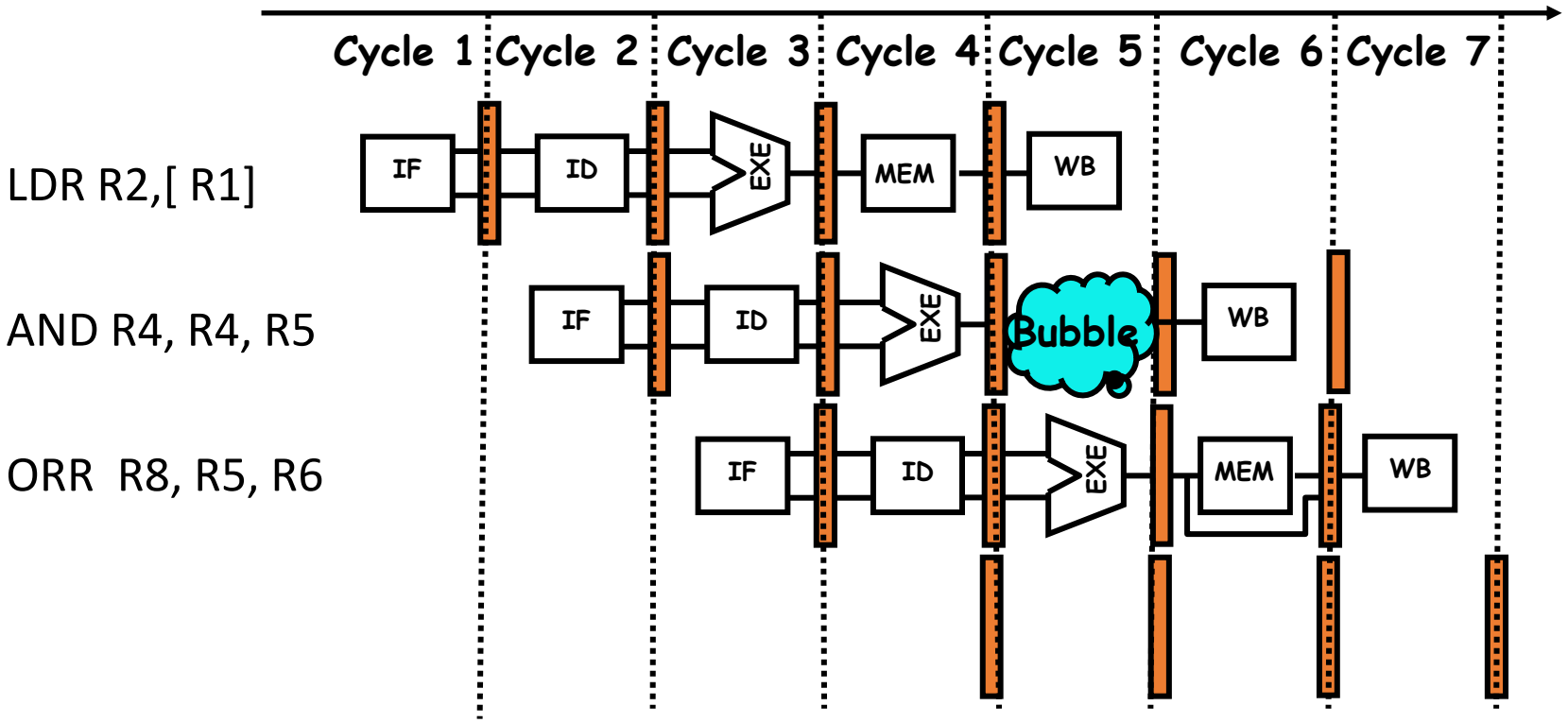
Structural Hazard: WB Vs WB:



2 ways to solve this pipeline hazard.

- **Stall the stage**
- **Let all the instruction follow all stages, AND may take longer than usual**

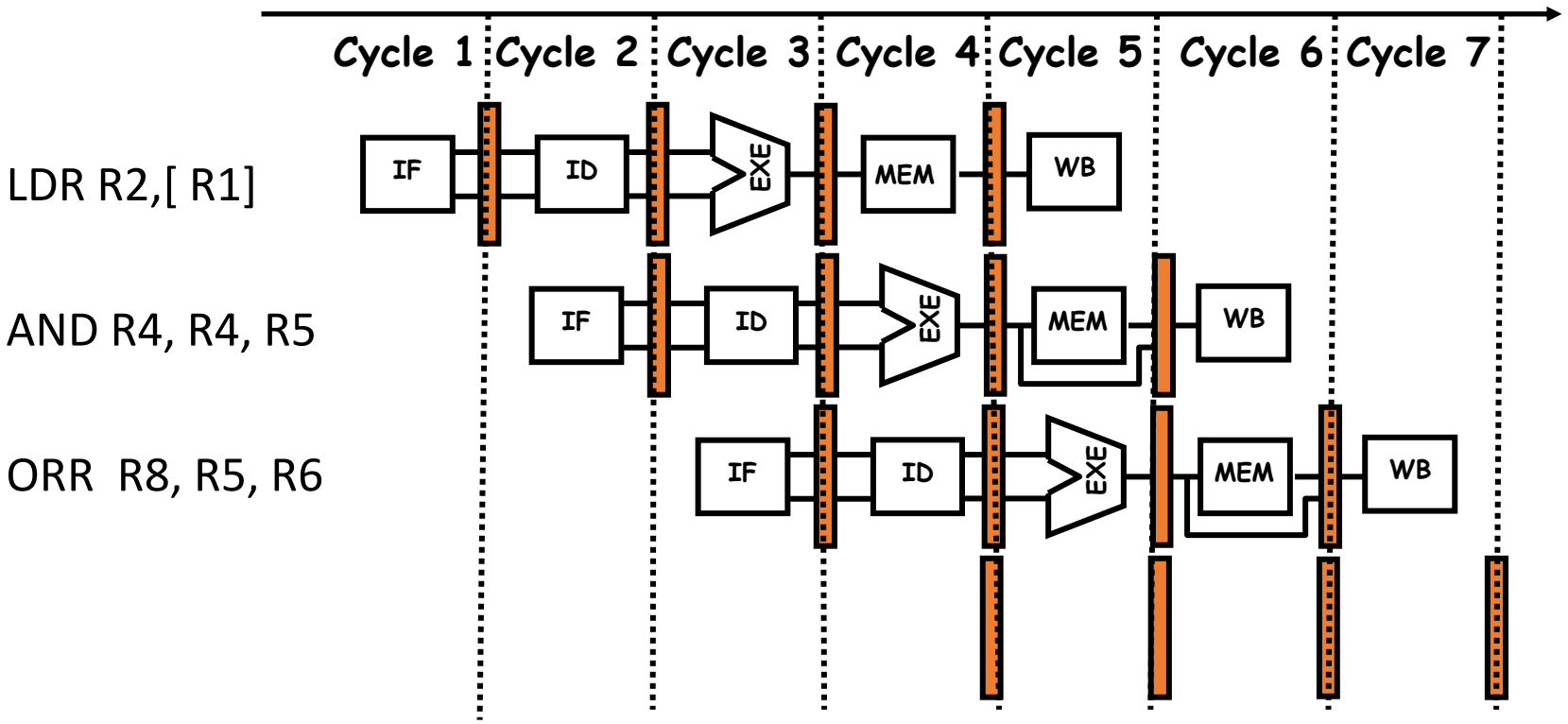
Structural Hazard: WB Vs WB: → Solution 1



2 ways to solve this pipeline hazard.

- **Stall the stage**
- **Let all the instruction follow all stages, AND may take longer than usual**

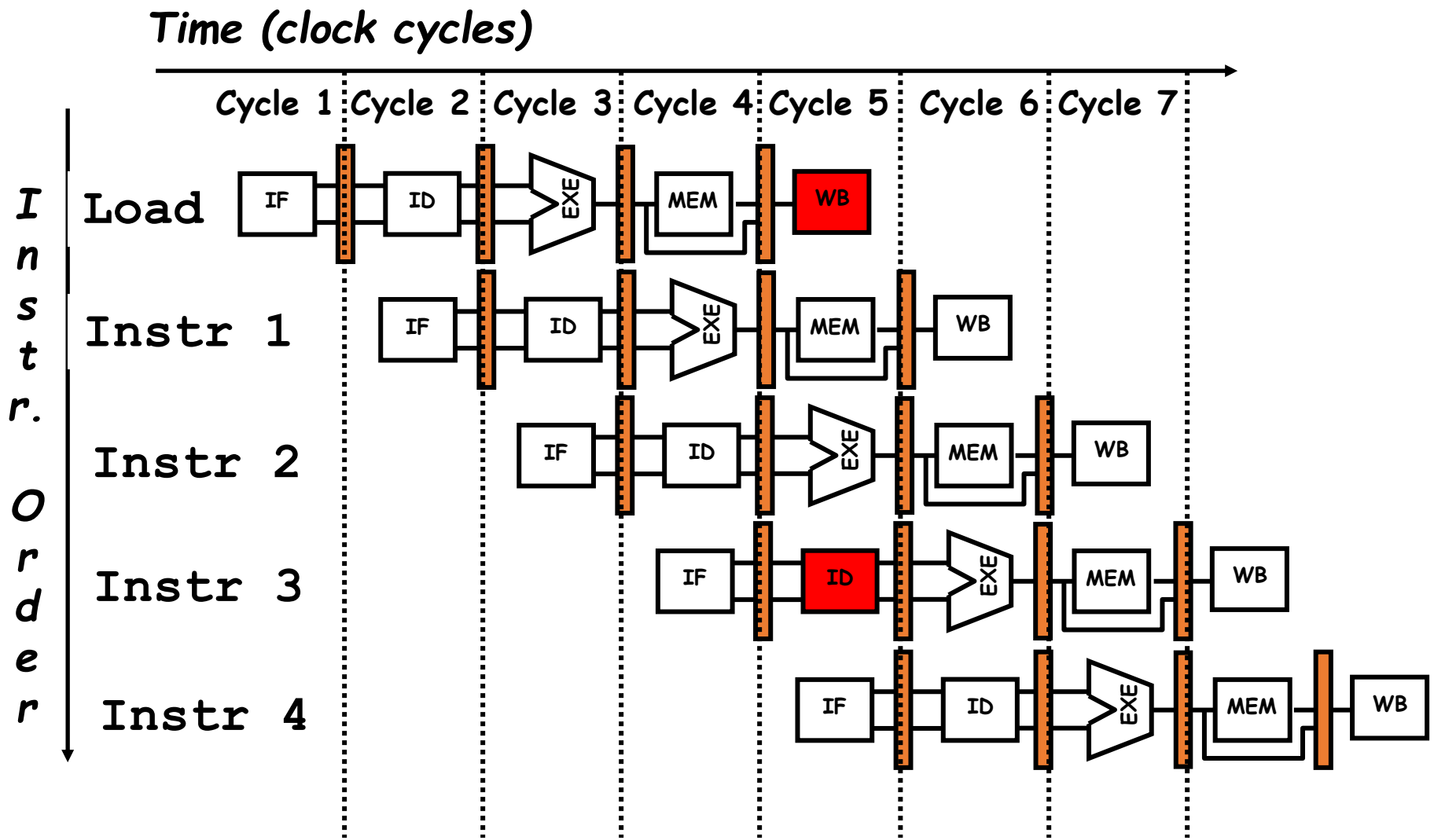
Structural Hazard: WB Vs WB: → Solution 2



2 ways to solve this pipeline hazard.

- **Stall the stage**
- **Let all the instruction follow all stages, AND may take longer than usual**

Structural Hazard: ID VS WB



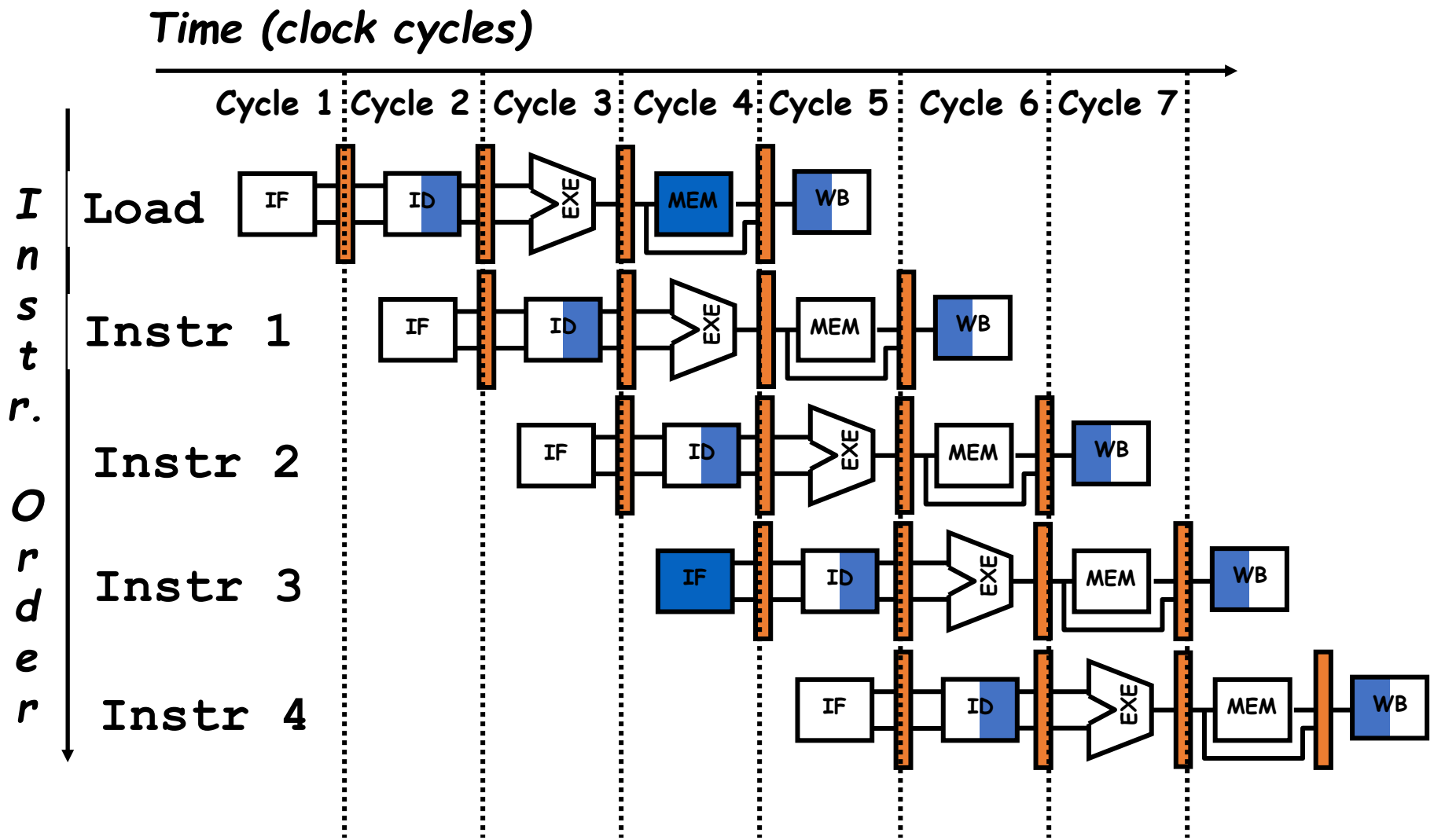
Structural Hazard: ID VS WB

Partitioning/Overlapping

1st half of the cycle *Reg Write*

2nd half of the same cycle *Reg Read*

Partitioning/Overlapping



Performance (with stalls)

- **n >> k; Speedup = k; no. of stages (also known as Depth of Pipeline)**

- *For 5-stage pipeline with stalls:*

- $Speed\ up = \frac{Depth\ of\ Pipeline}{CPI}$ = 5

- $Speed\ up = \frac{Depth\ of\ Pipeline}{1 + Pipeline\ Stall\ cycles\ per\ instruction}$

- If stalls are 0, then Speedup = k = 5
- If say, 30% are Load/Store in a unified memory pipeline, then every load/store overlap with IF will introduce 3 stalls. Thus: CPI = 1+(3x0.3) = 1.9

- $Speed\ up = \frac{5}{1 + (3x0.3)}$ = 2.63

Next Session

Data Hazards

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135

Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Pipeline Processor: Data Hazard 1

Dr. D. C. Kiran
Department of Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

- ~~3-Stage ARM Processor~~
- ~~5-Stage Pipeline Processor~~
- ~~Introduction to Pipeline Processor~~
- ~~What May Go Wrong?~~
- ~~Introduction to Hazards, Stalls,~~
- ~~Structural Hazards~~
- Data Hazard

  RAW, WAR, WAW Hazards

Unit 3: **Memory Design**

Unit 4: **Input/Output Device Design**

Unit 5: **Advanced Architecture**

Microprocessor & Computer Architecture (µpCA)

Text 1: "Computer Organization and Design", Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

Reference 1:"Computer Architecture: A Quantitative Approach", Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.

Appendix C **Pipelining: Basic and Intermediate Concepts**

C.1	Introduction	C-2
C.2	The Major Hurdle of Pipelining—Pipeline Hazards	C-11
C.3	How Is Pipelining Implemented?	C-30
C.4	What Makes Pipelining Hard to Implement?	C-43
C.5	Extending the MIPS Pipeline to Handle Multicycle Operations	C-51
C.6	Putting It All Together: The MIPS R4000 Pipeline	C-61
C.7	Crosscutting Issues	C-70
C.8	Fallacies and Pitfalls	C-80
C.9	Concluding Remarks	C-81
C.10	Historical Perspective and References	C-81
	Updated Exercises by Diana Franklin	C-82

Data Hazard

Consider a Scenario

Suppose initially, register ***i*** holds the number 2i
(ie. R3 = 6; R6=12; R8 = 16…)

What happens when the following sequence is executed in 5-stage pipeline?

add R3, R10, R11 - this should add 20 + 22, putting result 42 into r3

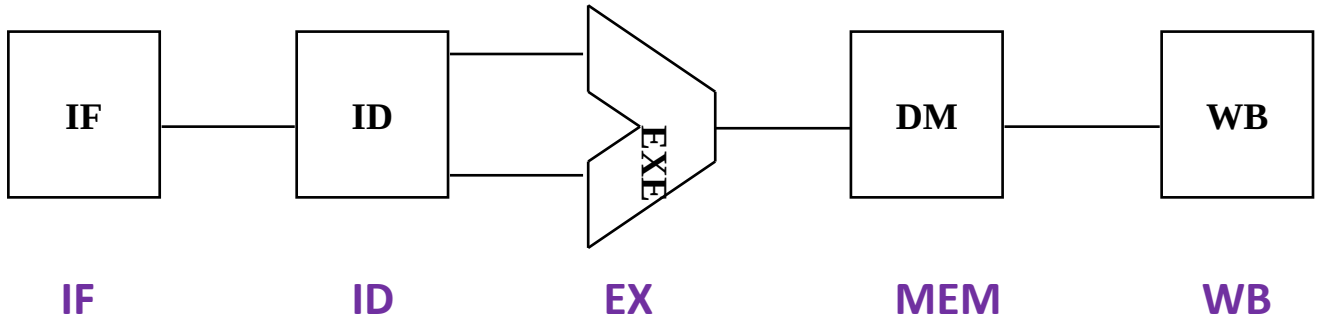
ldr R8, [R3, #50] - this should load r8 from memory location 42+50 = 92

sub R11, R8, R7 - this should subtract 14 from that just-loaded value from Mem[92]

Microprocessor & Computer Architecture (µpCA)

Cycle 1

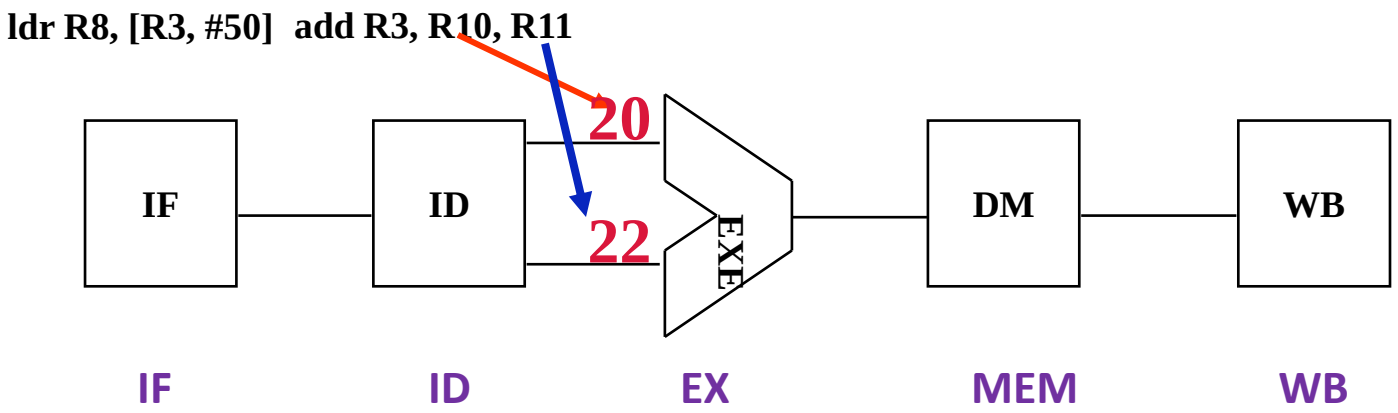
add R3, R10, R11



add	**R3, R10, R11**	- this should add 20 + 22, putting result 42 into r3
ldr	**R8, [R3, #50]**	- this should load r8 from memory location 42+50 = 92
sub	**R11, R8, R7**	- this should subtract 14 from that just-loaded value from Mem[92]

Microprocessor & Computer Architecture (µpCA)

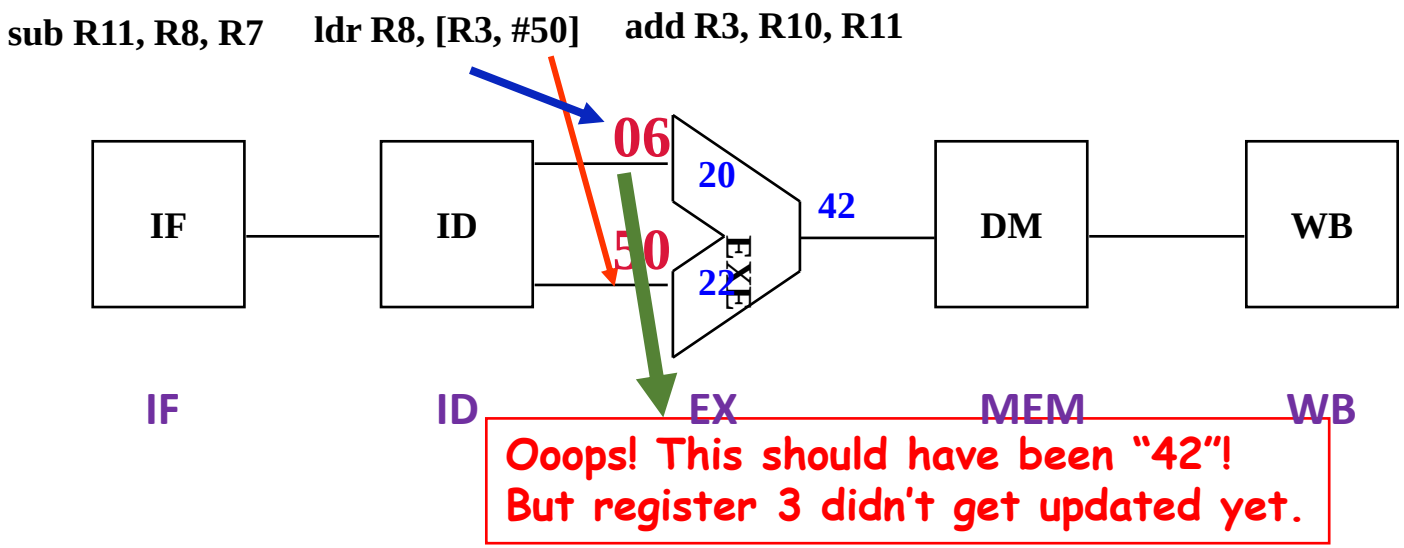
Cycle 2



add	**R3, R10, R11**	- this should add 20 + 22, putting result 42 into r3
ldr	**R8, [R3, #50]**	- this should load r8 from memory location 42+50 = 92
sub	**R11, R8, R7**	- this should subtract 14 from that just-loaded value from Mem[92]

Microprocessor & Computer Architecture (µpCA)

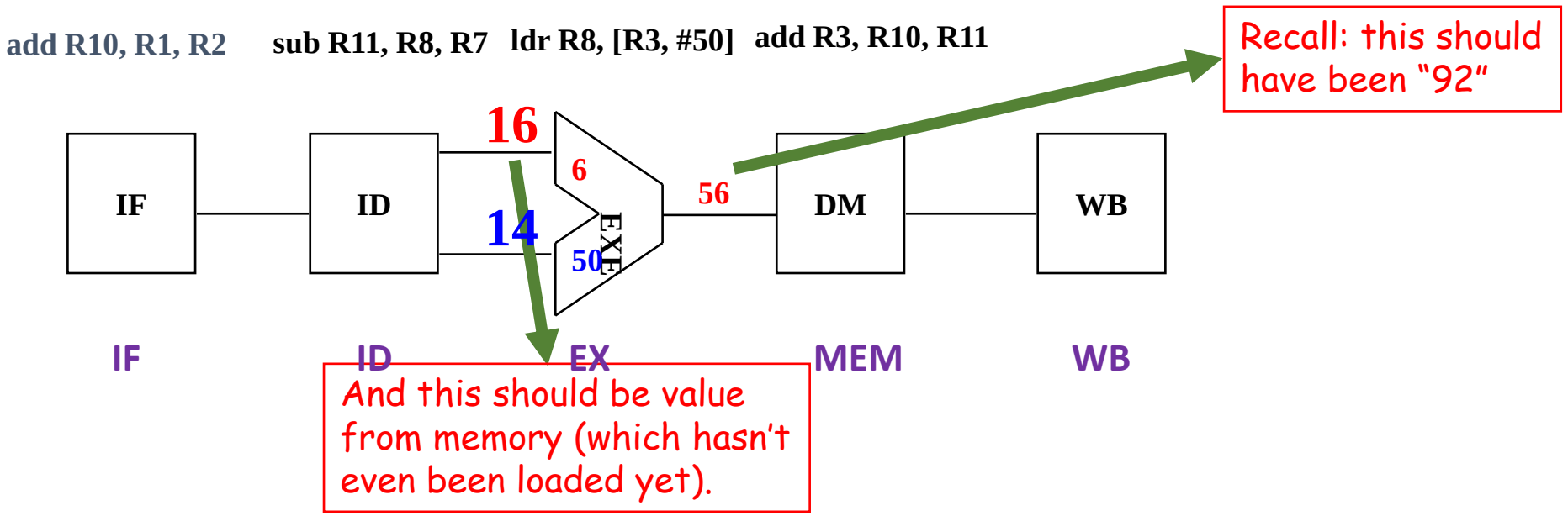
Cycle 3



add	**R3, R10, R11**	- this should add 20 + 22, putting result 42 into r3
ldr	**R8, [R3, #50]**	- this should load r8 from memory location 42+50 = 92
sub	**R11, R8, R7**	- this should subtract 14 from that just-loaded value from Mem[92]

Microprocessor & Computer Architecture (µpCA)

Cycle 4



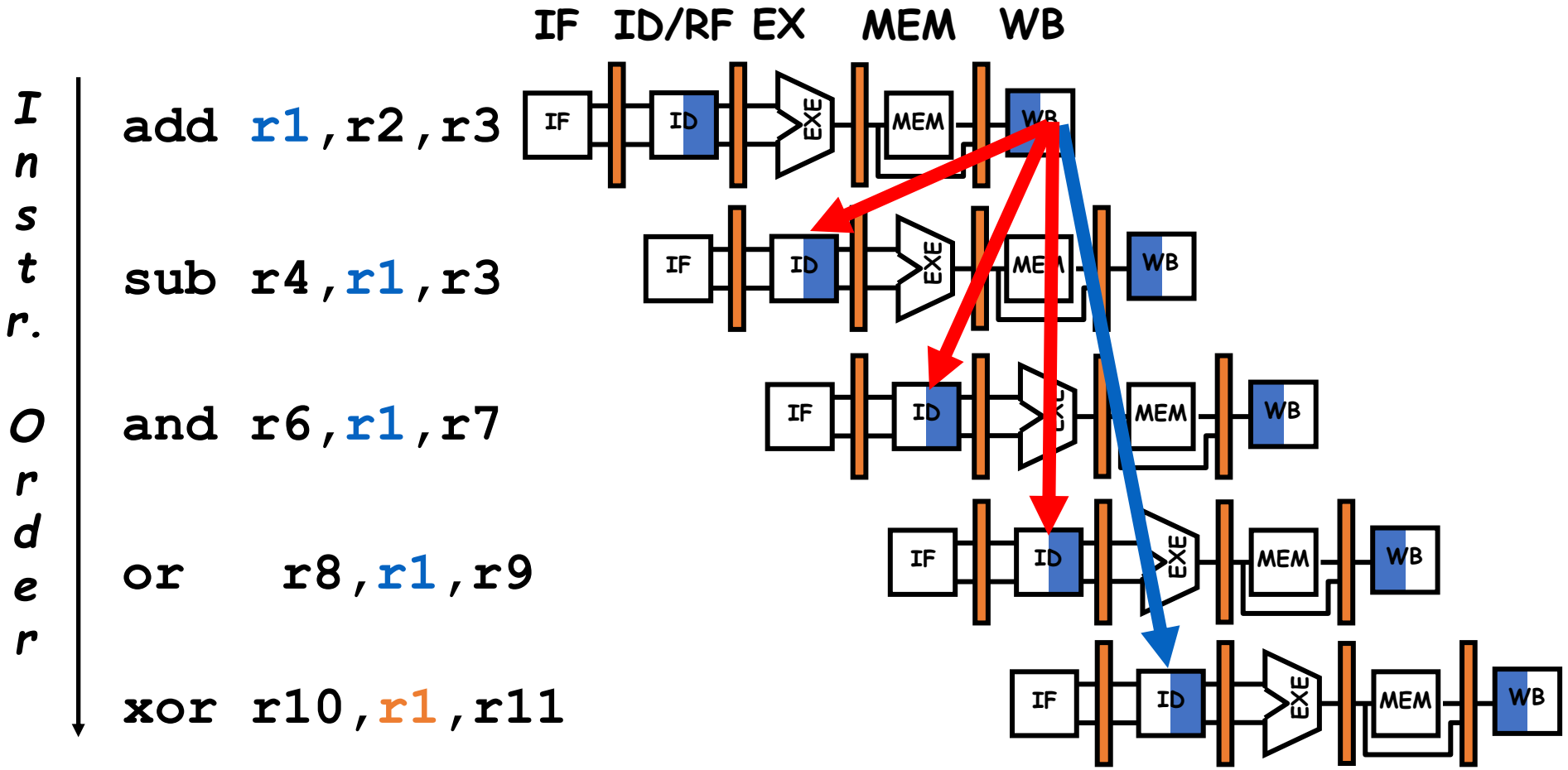
add **R3, R10, R11** - this should add 20 + 22, putting result 42 into r3

ldr **R8, [R3, #50]** - this should load r8 from memory location 42+50 = 92

sub **R11, R8, R7** - this should subtract 14 from that just-loaded value from Mem[92]

- Problem with starting next instruction before first is finished
  - Data dependencies here, that "go backward in time" create data hazards.

Data Dependency



Data Dependencies

- Three types of data dependencies defined in terms of how succeeding instruction depends on preceding instruction
  - **RAW:** Read after Write or Flow dependency
    (True Dependency)
  - **WAR:** Write after Read or anti-dependency
    (Anti Dependency)
  - **WAW:** Write after Write
    (Output Dependency)

- Read After Write (RAW)
  $Instr_J$ tries to read operand before $Instr_I$ writes it

```
I: add r1,r2,r3
J: sub r4,r1,r3
```

- Caused by a "Dependence" (in compiler nomenclature). This hazard results from an actual need for communication.

Data Dependencies

- Example program (a) with two instructions
  - i1: load **r1,** a;
  - i2: add r2, **r1,**r1;
- Program (b) with two instructions
  - i1: mul **r1,** r4, r5;
  - i2: add r2, **r1,** r1;
- Both cases we cannot read in i2 until i1 has completed writing the result
  - In **(a)** this is due to *load-use dependency*
  - In **(b)** this is due to *define-use dependency*

Data Dependencies

- Write After Read (WAR)
  Instr$_J$ writes operand *before* Instr$_I$ reads it

```
I: sub r4,r1,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

- Called an "anti-dependence" by compiler writers. This results from reuse of the name "r1".

- Can't happen if in pipeline:-
  - All instructions take 5 stages, and
  - Reads are always in stage 2, and
  - Writes are always in stage 5

Data Dependencies

- Write After Write (WAW)
  Instr$_J$ writes operand *before* Instr$_I$ writes it.
- Called an "output dependence" by compiler writers
  This also results from the reuse of name "r1".
- Can't happen if in pipeline:
- All instructions take 5 stages, and Writes are always in stage 5

```
I: sub r1,r4,r3
J: add r1,r2,r3
K: mul r6,r1,r7
```

If Executed Out of order by the Compiler, may lead to wrong results

WAR and WAW Dependency

- Example program (a):
  - i1: mul r1, r2, r3;
  - i2: add r2, r4, r5;
- Example program (b):
  - i1: mul r1, r2, r3;
  - i2: add r1, r4, r5;
- both cases we have dependence between i1 and i2
  - in (a) due to r2 must be read before it is written into
  - in (b) due to r1 must be written by i2 after it has been written into by i1

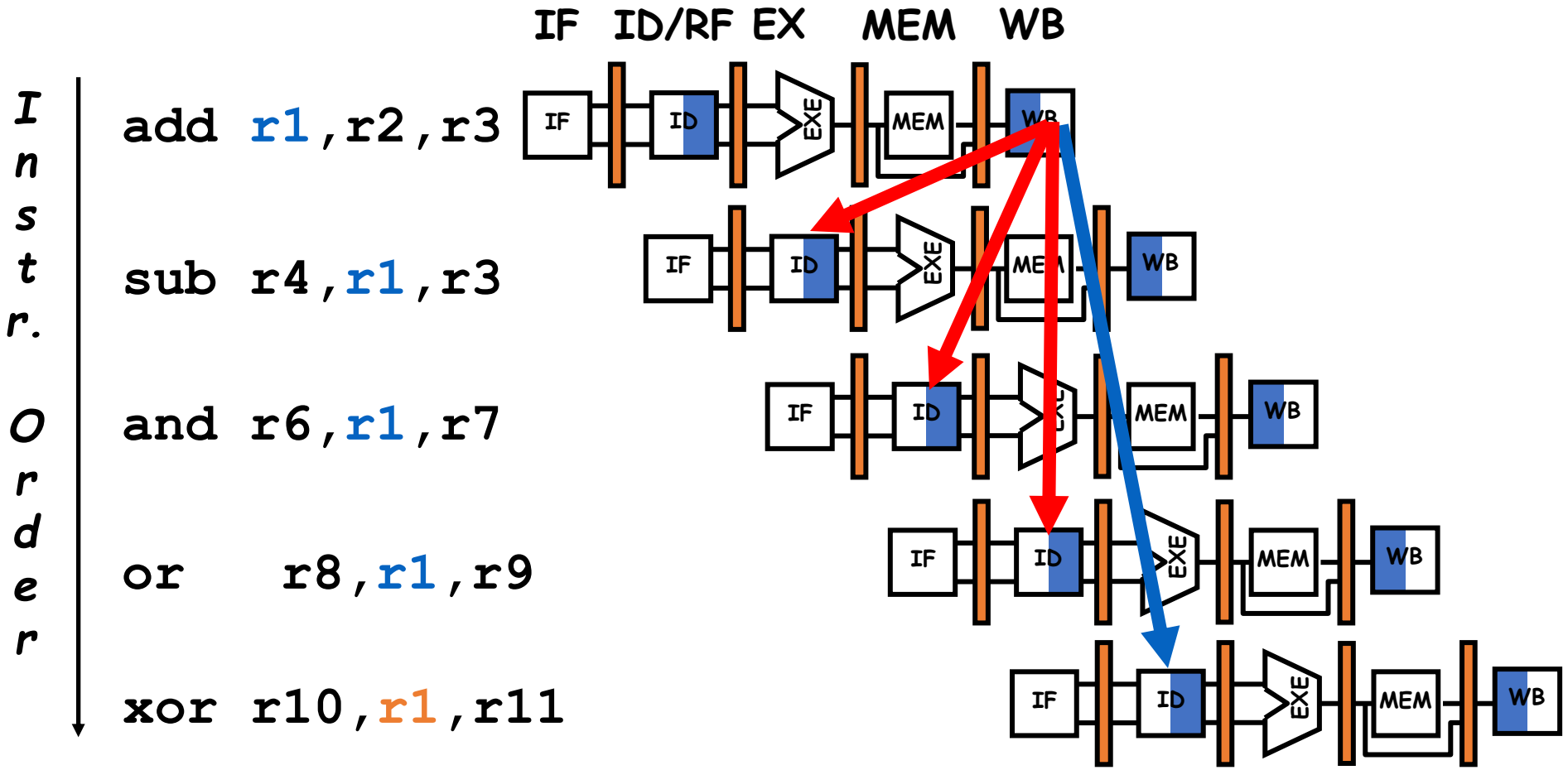
WAR and WAW Dependency

- Problem:
  - i1: mul r1, r2, r3;
  - i2: add r2, r4, r5;
- Is this really a dependence/hazard ?

WAR and WAW Dependency

- Solution: Rename Registers
  - i1: mul r1, r2, r3;
  - i2: add r6, r4, r5;
- Register renaming can solve many of these *false dependencies*
  - note the role that the compiler plays in this
  - specifically, the register allocation process--i.e., the process that assigns registers to variables

Next Session: How to Attack RAW Dependency?



THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135

Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Pipeline Processor: Data Hazard 2

Dr. D. C. Kiran
Department of Computer Science and Engineering

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

- ~~3-Stage ARM Processor~~
- ~~5-Stage Pipeline Processor~~
- ~~Introduction to Pipeline Processor~~
- ~~What May Go Wrong?~~
- ~~Introduction to Hazards, Stalls,~~
- ~~Structural Hazards~~
- ~~Data Hazard~~

  ~~RAW, WAR, WAW Hazards~~
- Attacking Data Hazard

  Software Approach

  Hardware Approach

Microprocessor & Computer Architecture (µpCA)

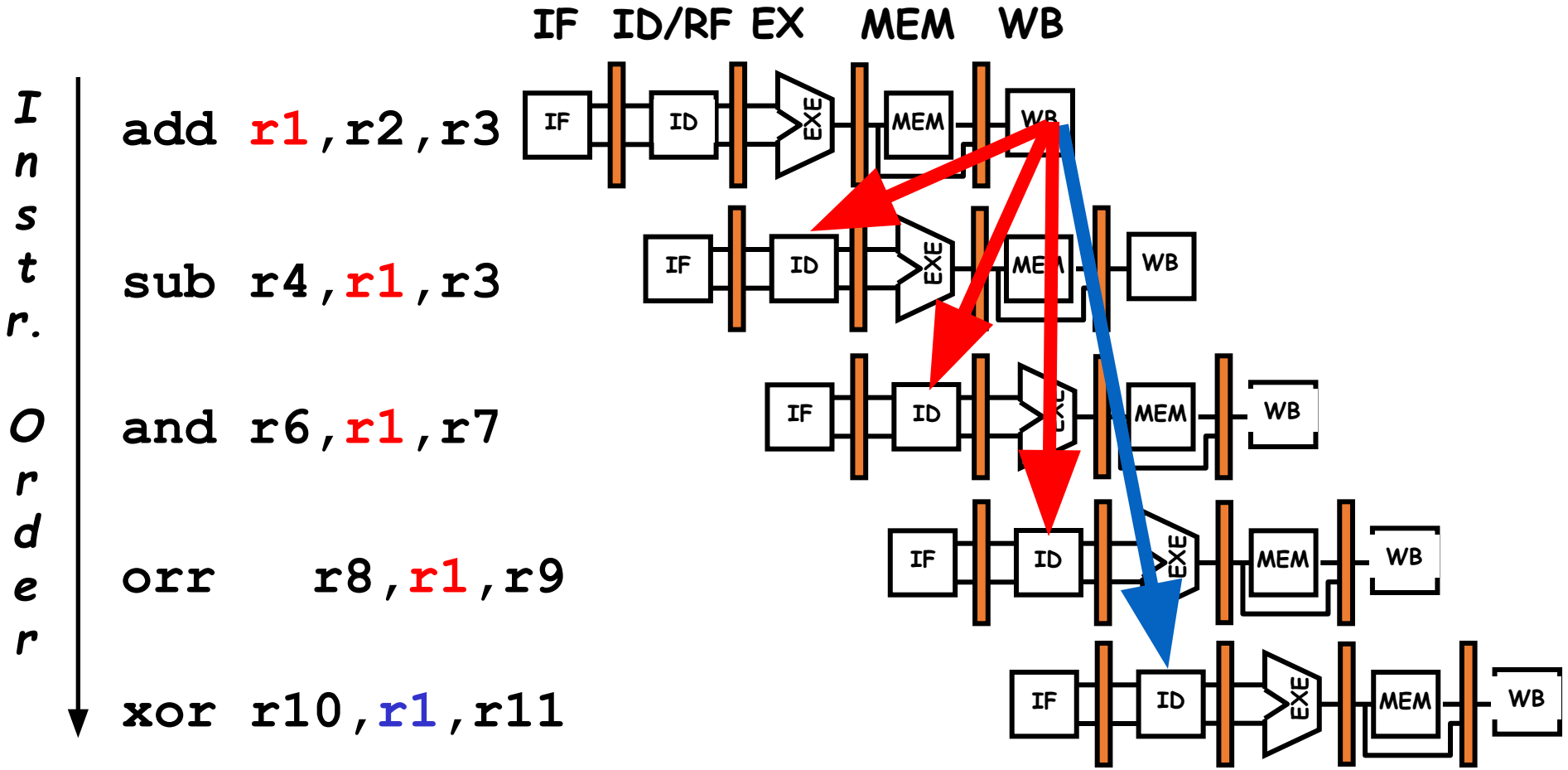
Text 1: "Computer Organization and Design", Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

Reference 1:"Computer Architecture: A Quantitative Approach", Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.

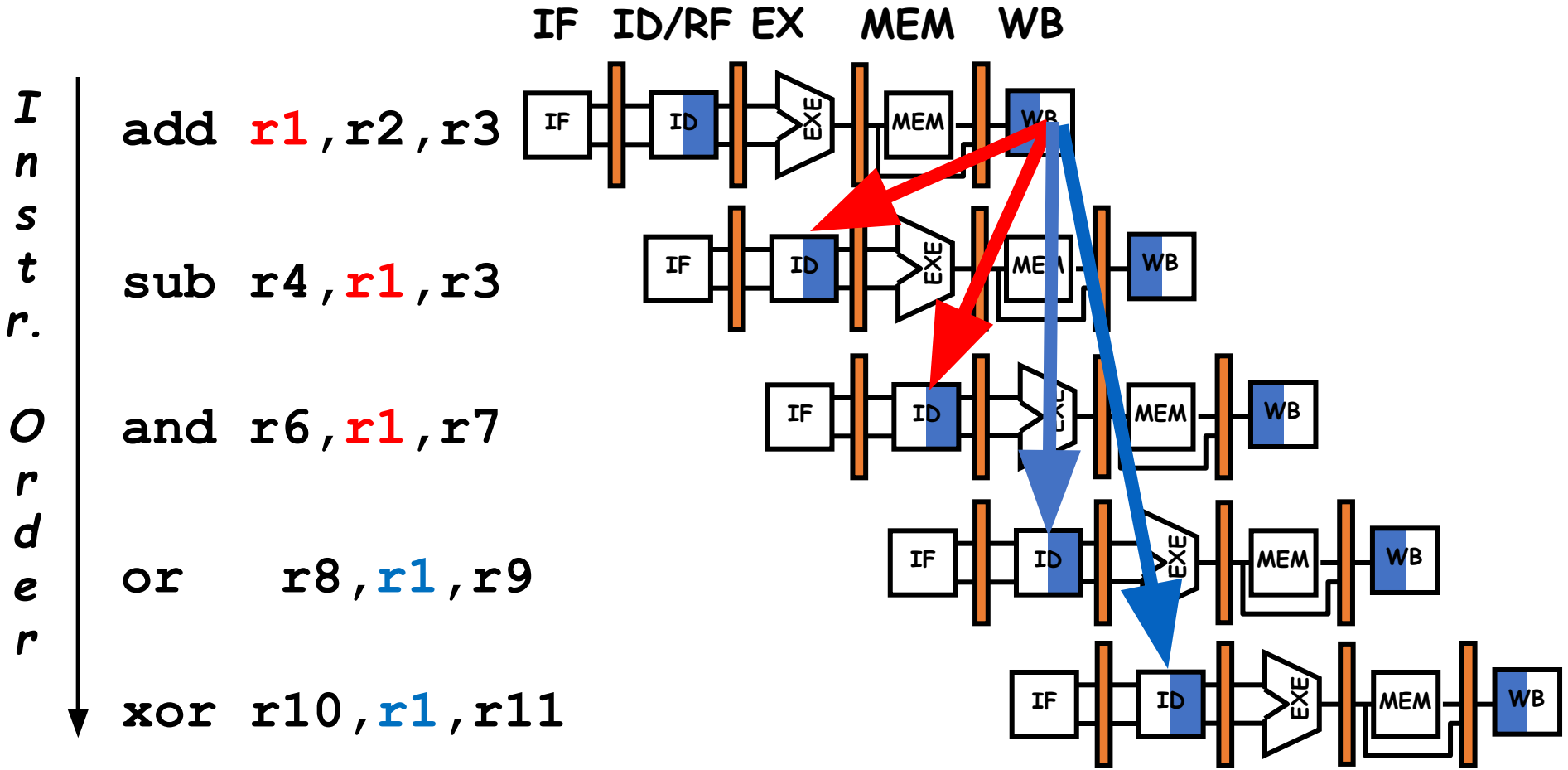
Appendix C **Pipelining: Basic and Intermediate Concepts**

C.1	Introduction	C-2
C.2	The Major Hurdle of Pipelining—Pipeline Hazards	C-11
C.3	How Is Pipelining Implemented?	C-30
C.4	What Makes Pipelining Hard to Implement?	C-43
C.5	Extending the MIPS Pipeline to Handle Multicycle Operations	C-51
C.6	Putting It All Together: The MIPS R4000 Pipeline	C-61
C.7	Crosscutting Issues	C-70
C.8	Fallacies and Pitfalls	C-80
C.9	Concluding Remarks	C-81
C.10	Historical Perspective and References	C-81
	Updated Exercises by Diana Franklin	C-82

How to Attack RAW Dependency?



How to Attack RAW Dependency?



Attacking Data Hazard

- In Software
  - **Solution 1: Re-order instructions**
  - Solution 2: insert independent instructions (or no-ops) Ex: MOV R0, R0
- In Hardware
  - Solution 1: Insert bubbles (i.e. stall the pipeline)
  - Solution 2: Data Forwarding

Software Solution 1 By Compiler

```
ADD R0,R0,R1
ADD R2,R3,R4
ADD R5,R0,R4
SUB R6,R4,R7
XOR R8,R4,R7
```

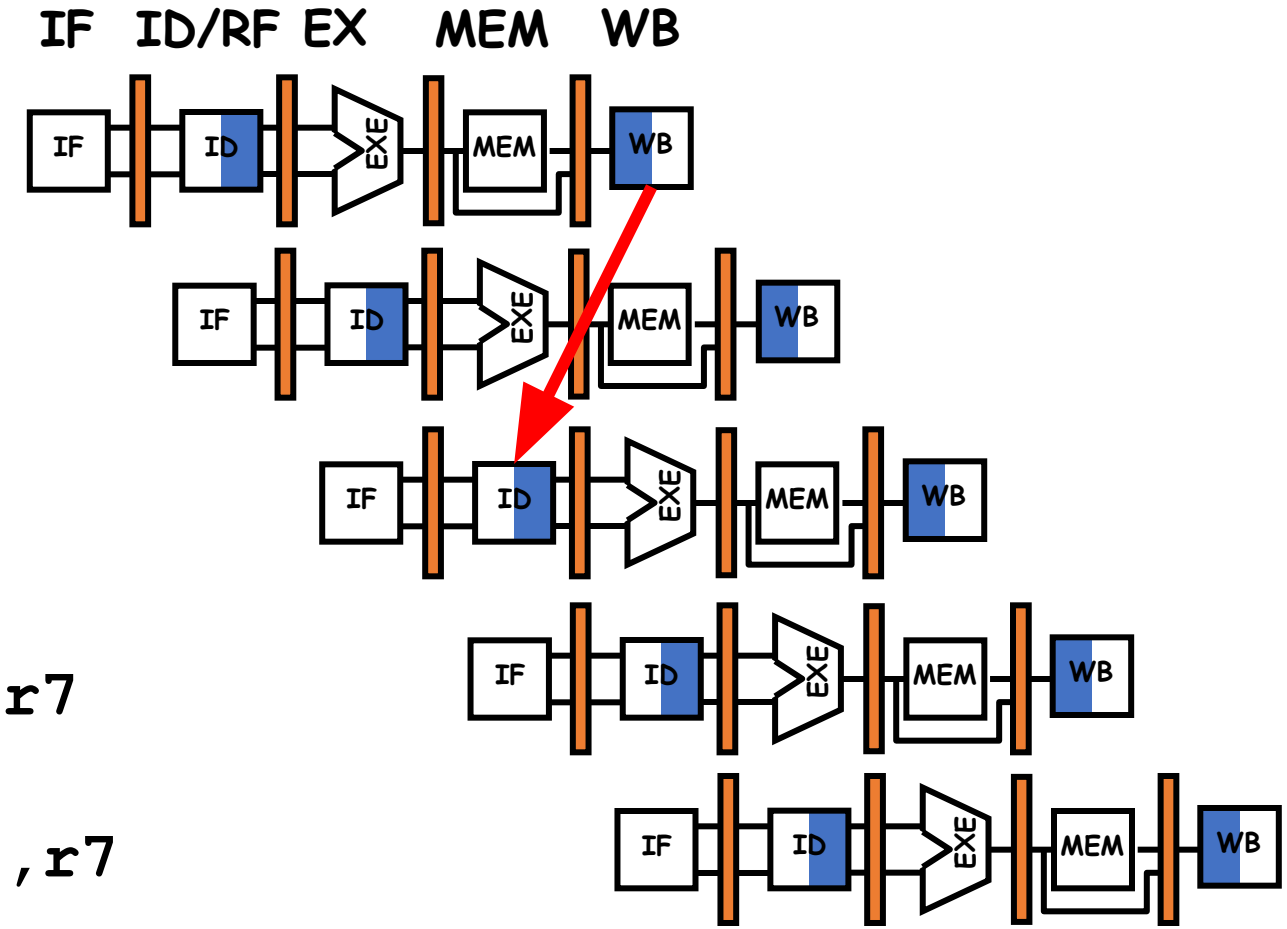
```
ADD R0,R0,R1
ADD R2,R3,R4
SUB R6,R4,R7
XOR R8,R4,R7
ADD R5,R0,R4
```

Out of Order Execution by Compiler

Software Solution 1 By Compiler

Instr. Order

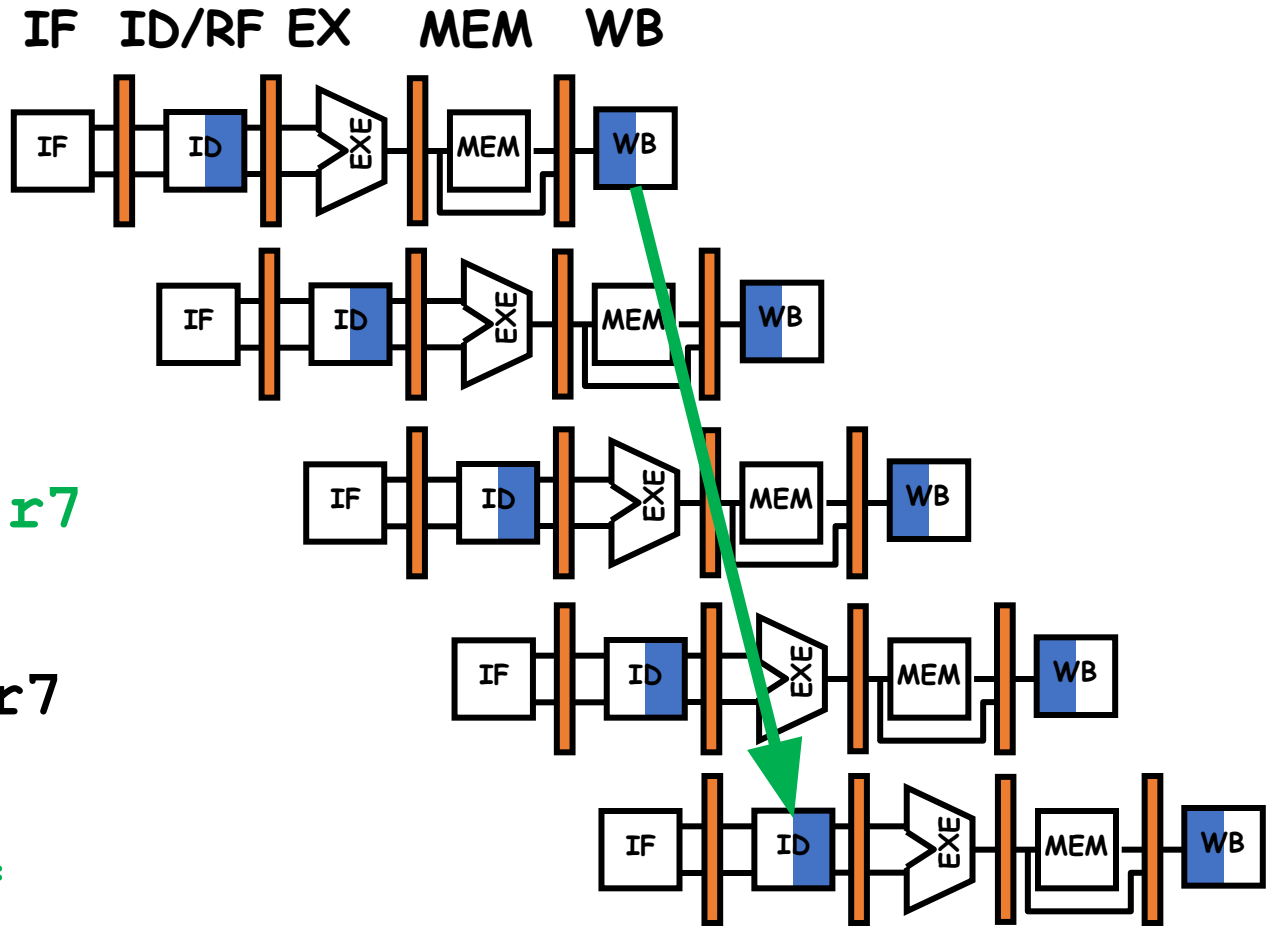
```
add r0,r0,r1
add r2,r3,r4
add r5,r0,r4
sub   r6,r4,r7
xor   r8, r4,r7
```



Software Solution 1 By Compiler

Instr. Order

```
add r0,r0,r1
add r2,r3,r4
xor  r8, r4,r7
sub  r6,r4,r7
add r5,r0,r4
```



Attacking Data Hazard

- In Software
  - Solution 1: Re-order instructions
  - **Solution 2: insert independent instructions (or no-ops) Ex: MOV R0, R0**
- In Hardware
  - Solution 1: Insert bubbles (i.e. stall the pipeline)
  - Solution 2: Data Forwarding

Software Solution 2 By Compiler

Insert NOP or MOV R0, R0

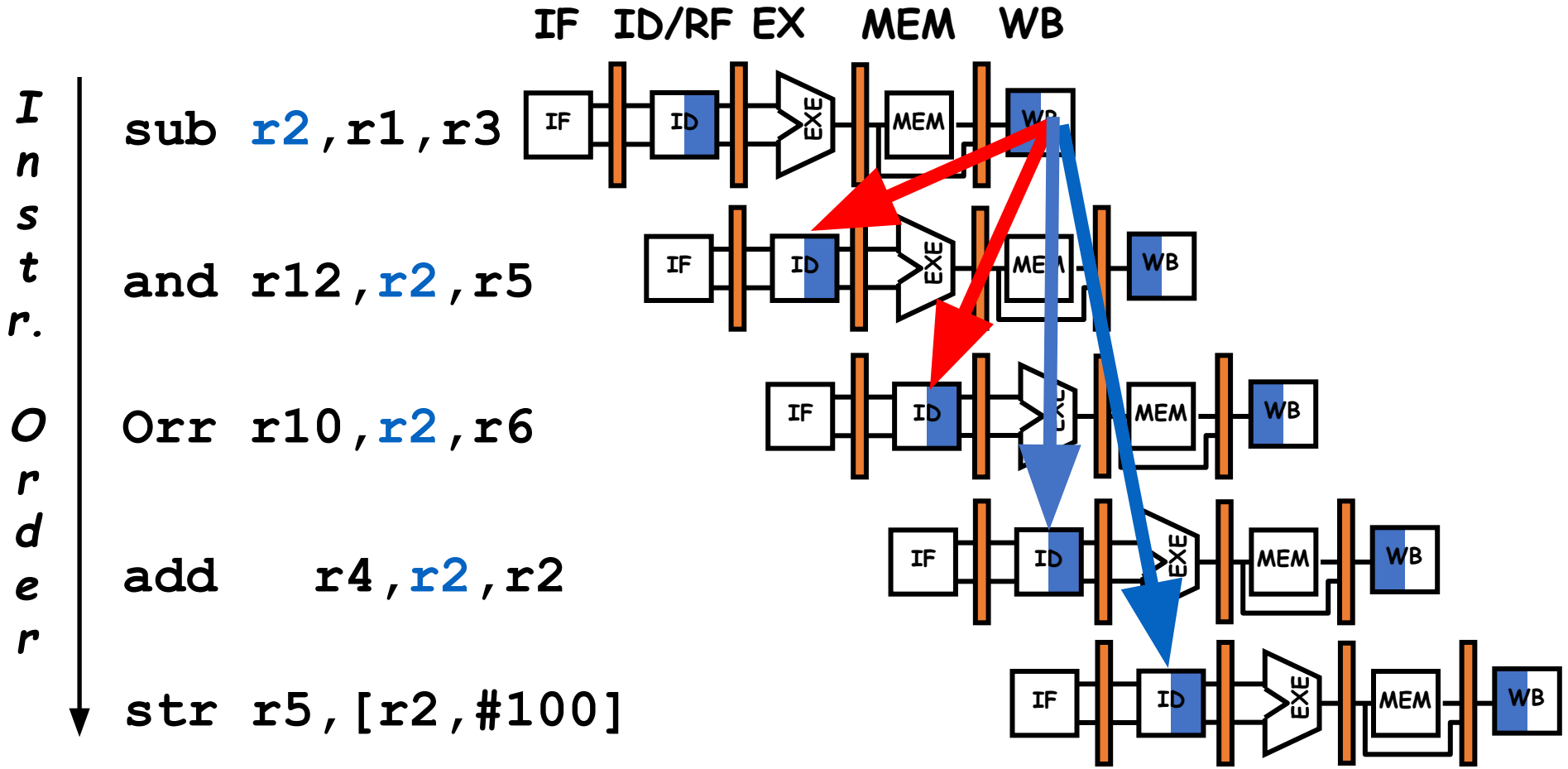
- The compiler can guarantee that no data hazards exist adding NOP or MOV instructions where needed. [Instruction Scheduling]

```
sub   R2, R1, R3
and   R12, R2, R5
orr   R10, R6, R2
add   R4, R2, R2
str   R5, [R2, #100]
```

```
sub   R2, R1, R3
NOP or MOV R0, R0
NOP or MOV R0, R0

and   R12, R2, R5
orr   R10, R6, R2
add   R4, R2, R2
str   R5, [R2, #100]
```

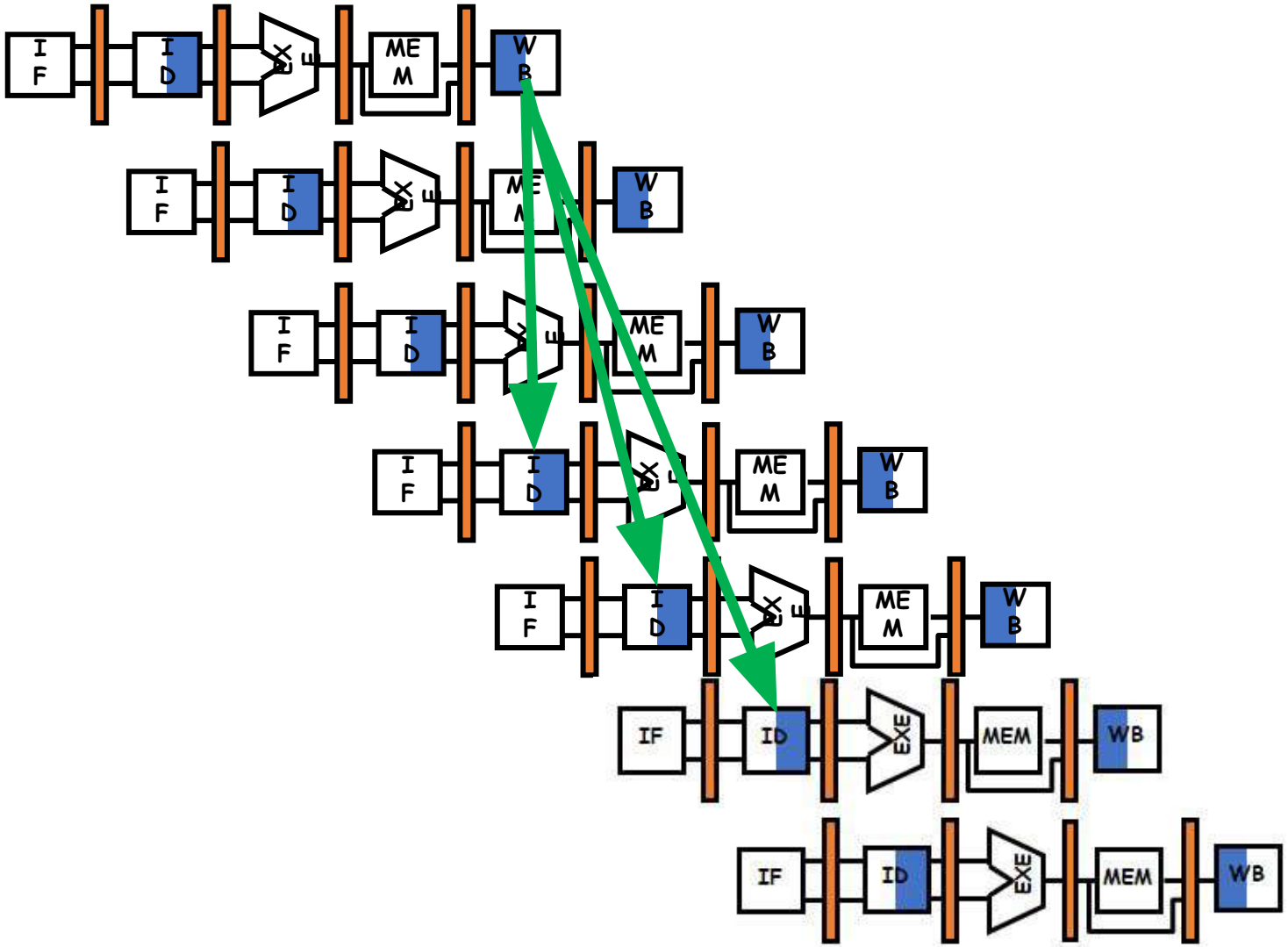
Software Solution 2 By Compiler



Microprocessor & Computer Architecture (µpCA)

Software Solution 2 By Compiler

```
sub r2,r1,r3
mov r0,r0
mov r0,r0
and r12,r2,r5
orr r10,r2,r6
add   r4,r2,r2
str r5,[r2,#100]
```



Where are NOPs needed?

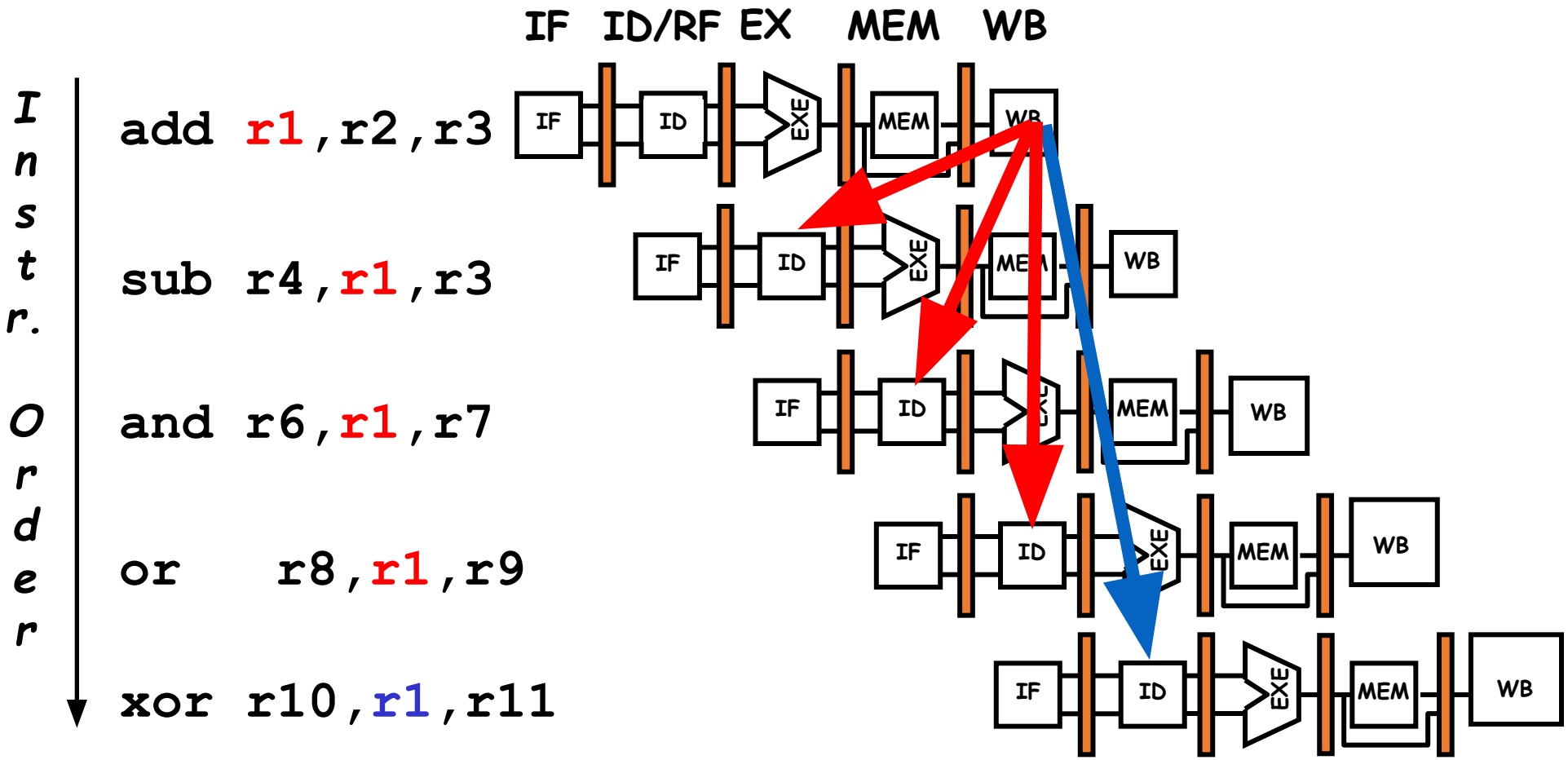
```
sub   R2, R1, R3
and   R4, R2, R5
orr   R8, R2, R6
add   R9, R4, R2
rsb   R1, R6, R7
```

```
sub   R2, R1, R3
NOP
NOP
and   R4, R2, R5
orr   R8, R2, R6
rsb   R1, R6, R7
add   R9, R4, R2
```

Attacking Data Hazard

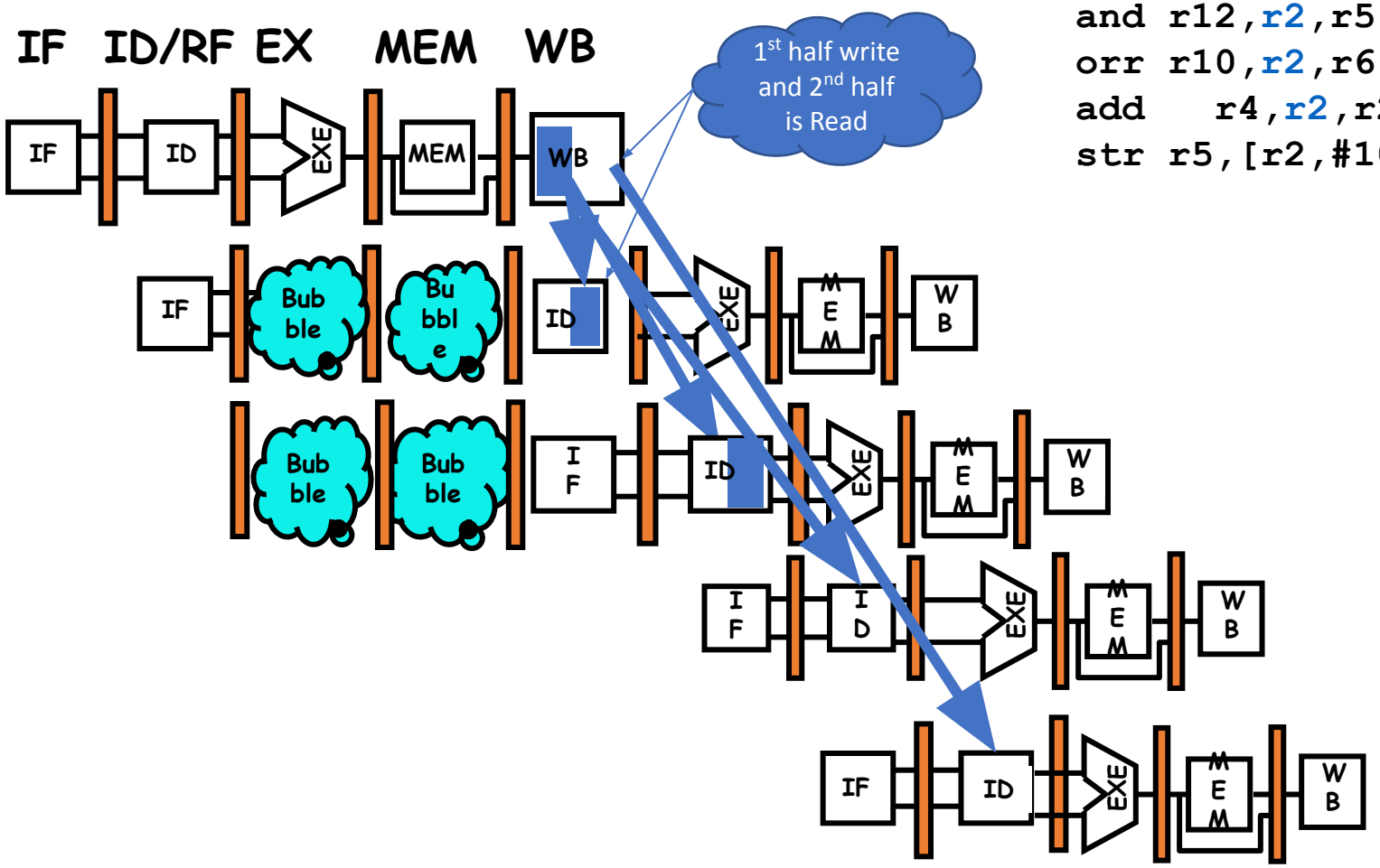
- In Software
  - Solution 1: Re-order instructions
  - Solution 2: Insert independent instructions (or no-ops) Ex: MOV R0, R0
- **In Hardware**
  - **Solution 1: Insert bubbles (i.e. stall the pipeline)**
  - Solution 2: Data Forwarding

Hardware Solution 1



Hardware Solution 1

```
sub r2,r1,r3
and r12,r2,r5
orr r10,r2,r6
add   r4,r2,r2
str r5,[r2,#100]
```



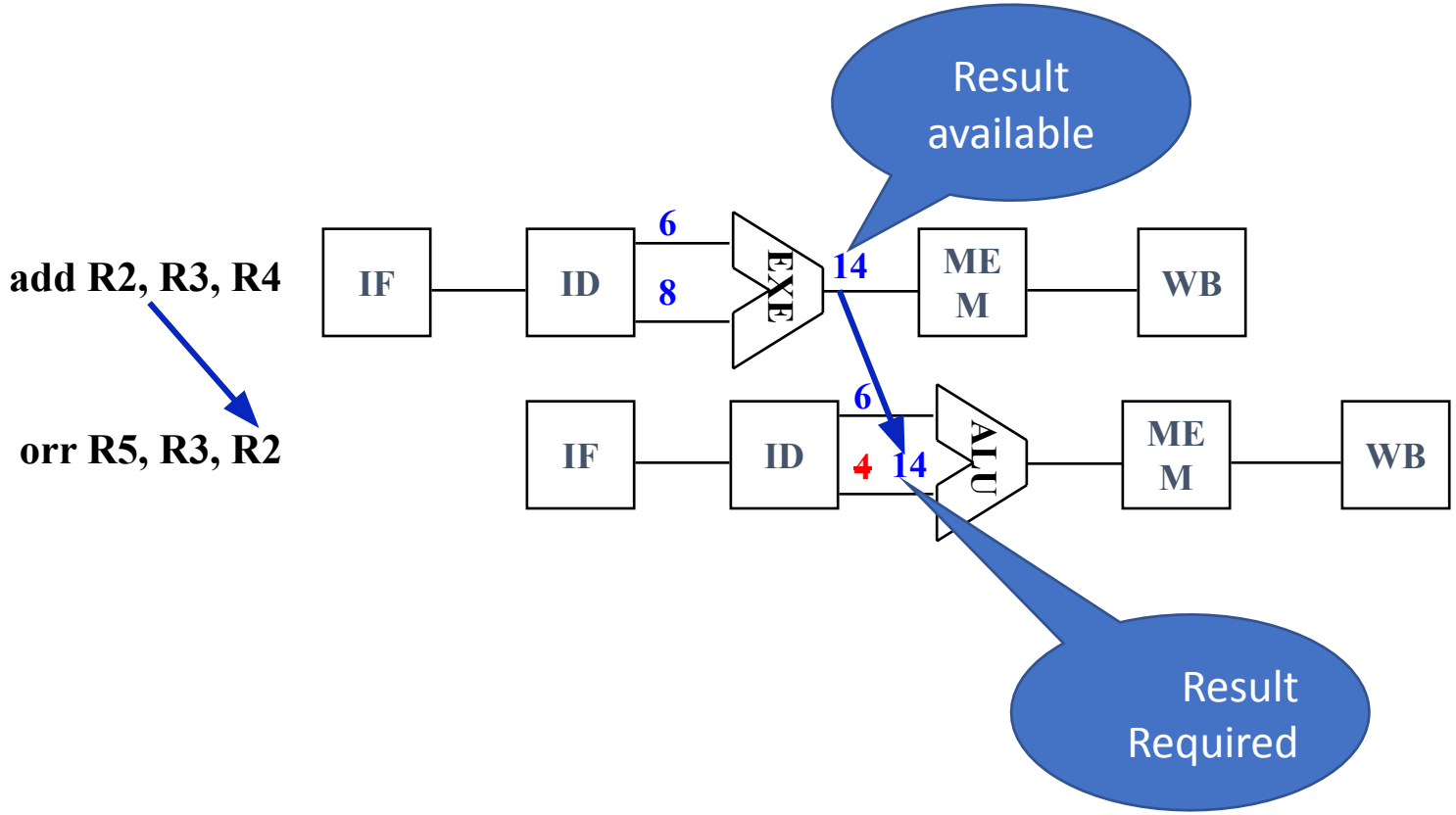
Attacking Data Hazard

- In Software
  - Solution 1: Re-order instructions
  - Solution 2: Insert independent instructions (or no-ops) Ex: MOV R0, R0
- **In Hardware**
  - Solution 1: Insert bubbles (i.e. stall the pipeline)
  - **Solution 2: Data Forwarding**

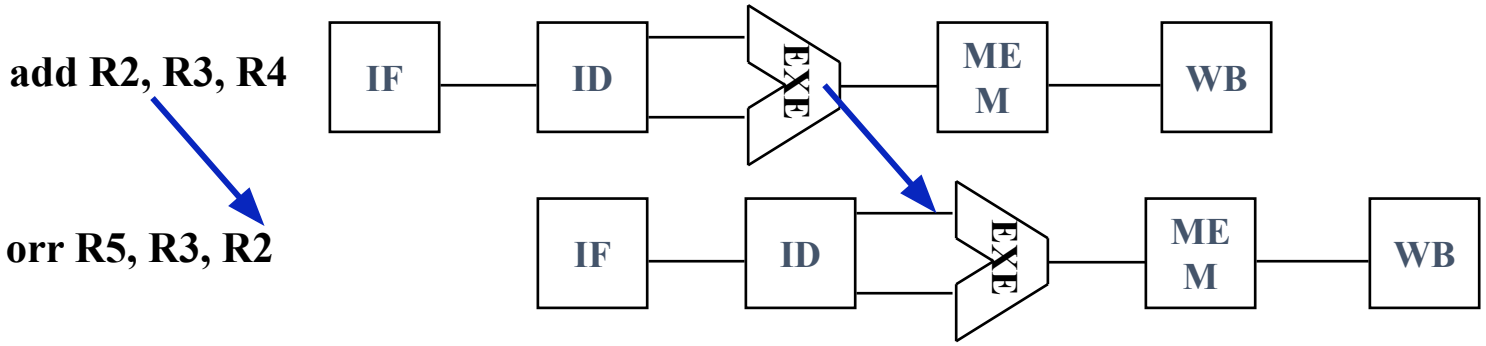
Hardware Solution 2: Data Forwarding

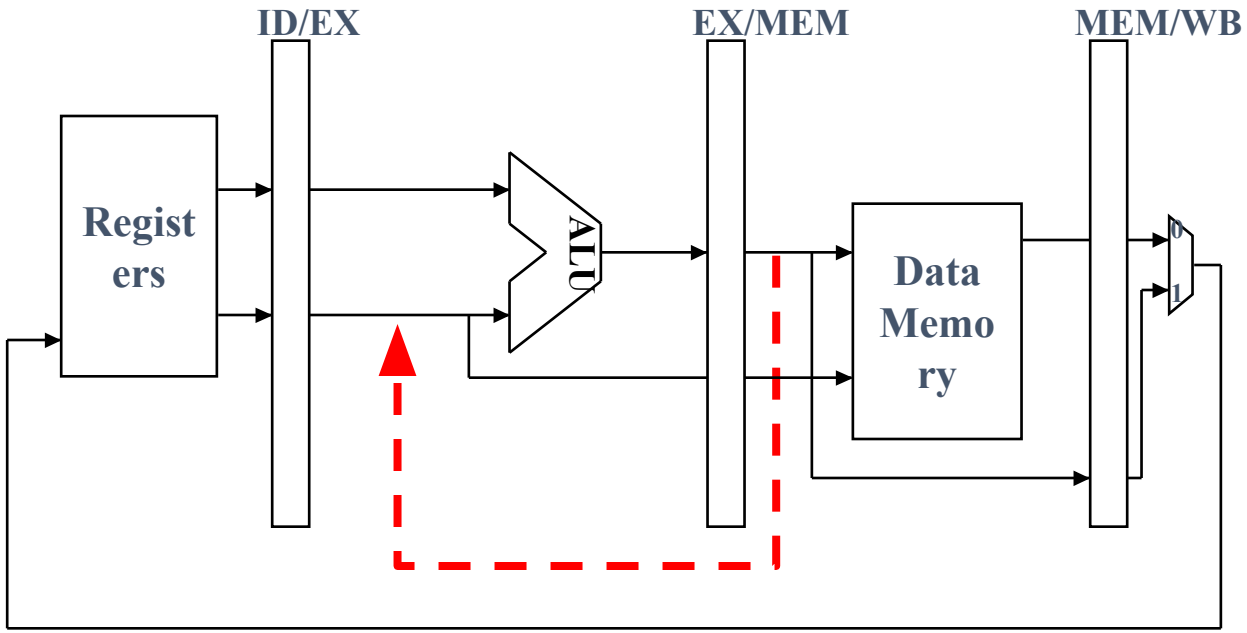
We could avoid stalling if we could get to EX stage the ALU output from "previous instruction" to ALU input for the "current instruction"

Suppose initially, register i holds the number 2i (ie. R3 = 6; R6=12; R8 = 16…)



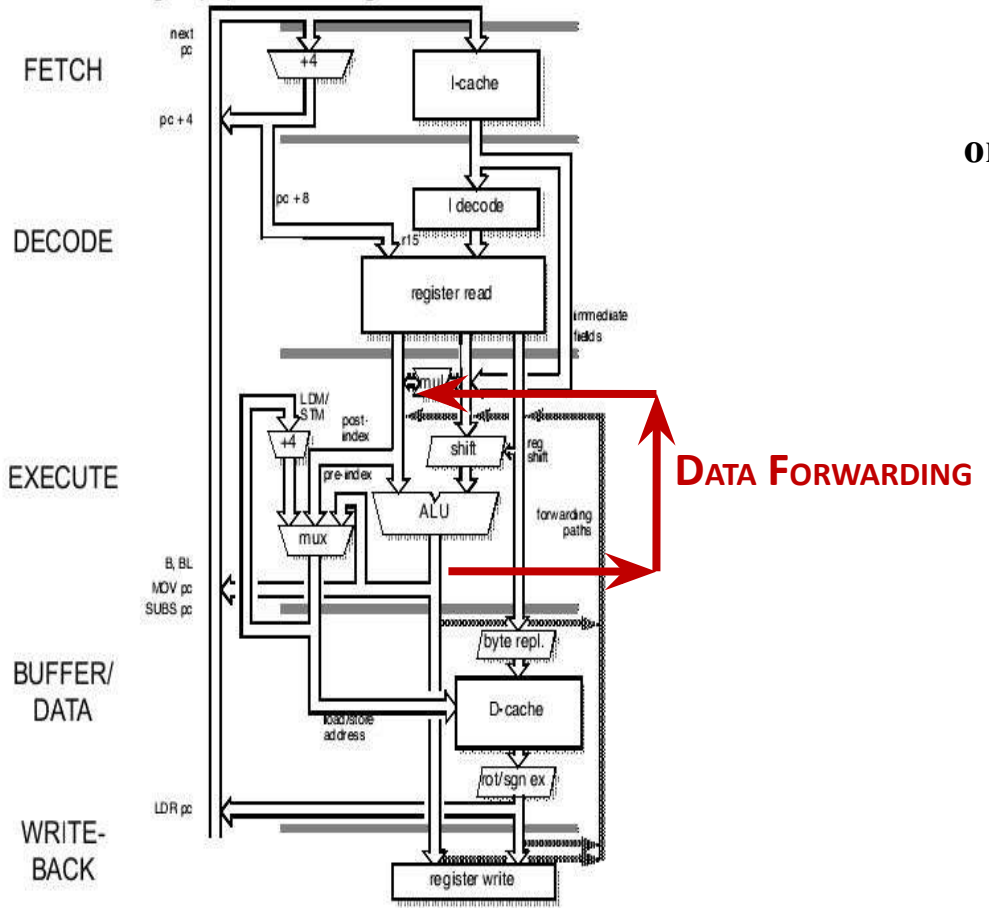
Hardware Solution 2: Data Forwarding





Hardware Solution 2: Data Forwarding

ARM9TDMI 5-stage pipeline organization



DATA FORWARDING

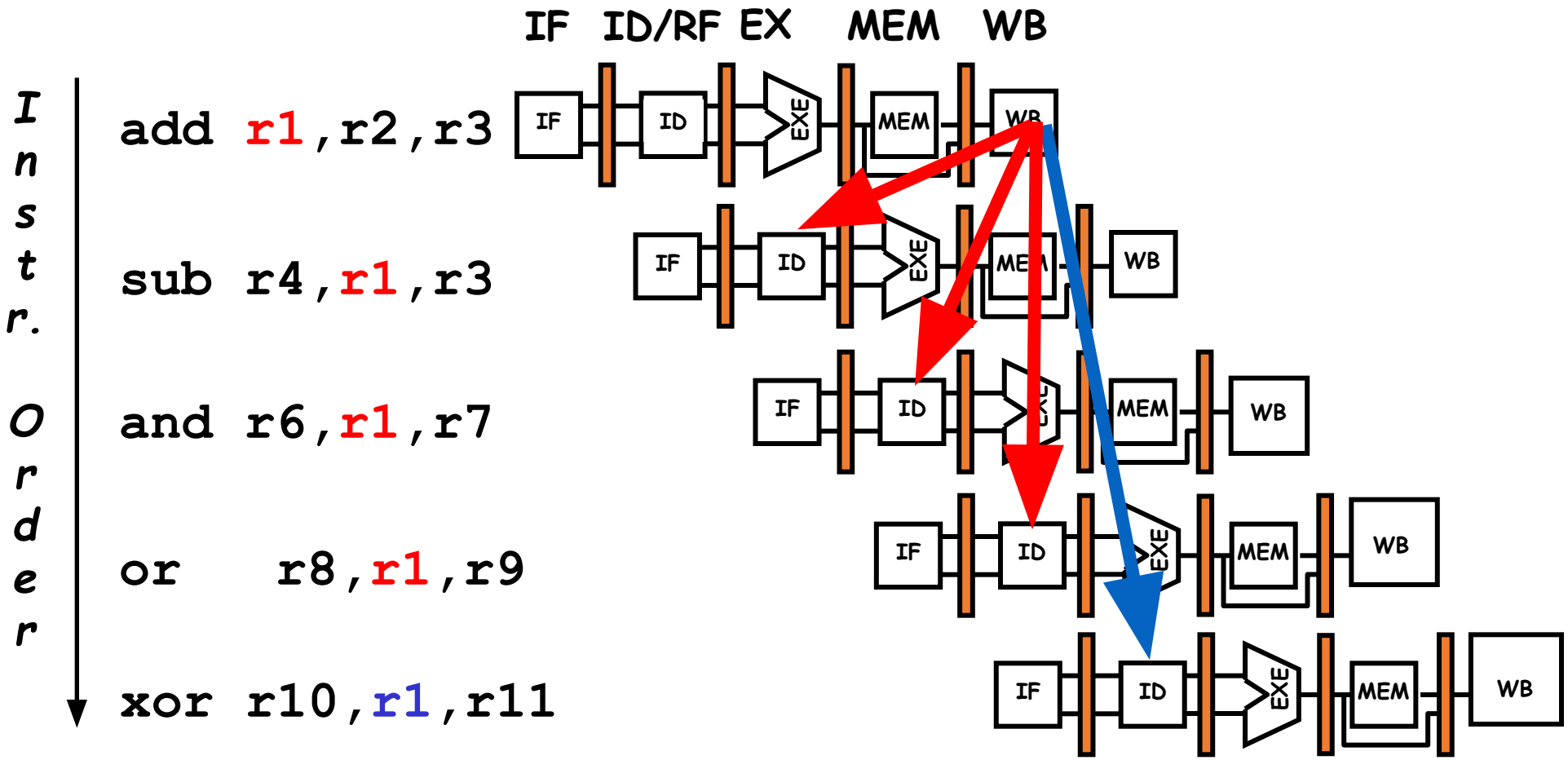
add R2, R3, R4

orr R5, R3, R2

Hardware Solution 2: Data Forwarding

- Also known as:
  - Register –bypassing
  - Short-circuiting
- Forwarding handles hazards at both
  - EX stage
  - MEM stage

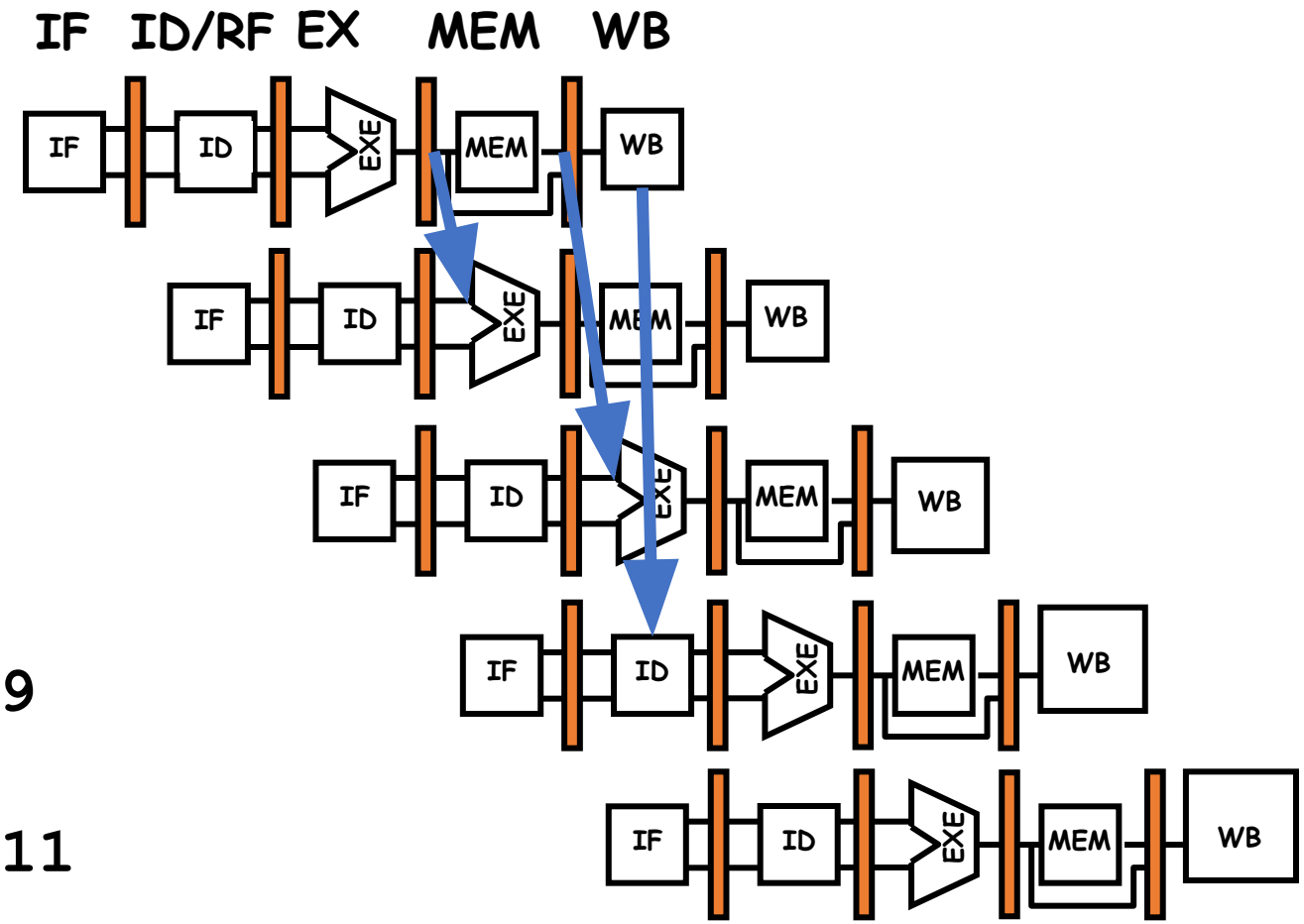
Hardware Solution 2: Data Forwarding



Hardware Solution 2: Data Forwarding

Instr. Order

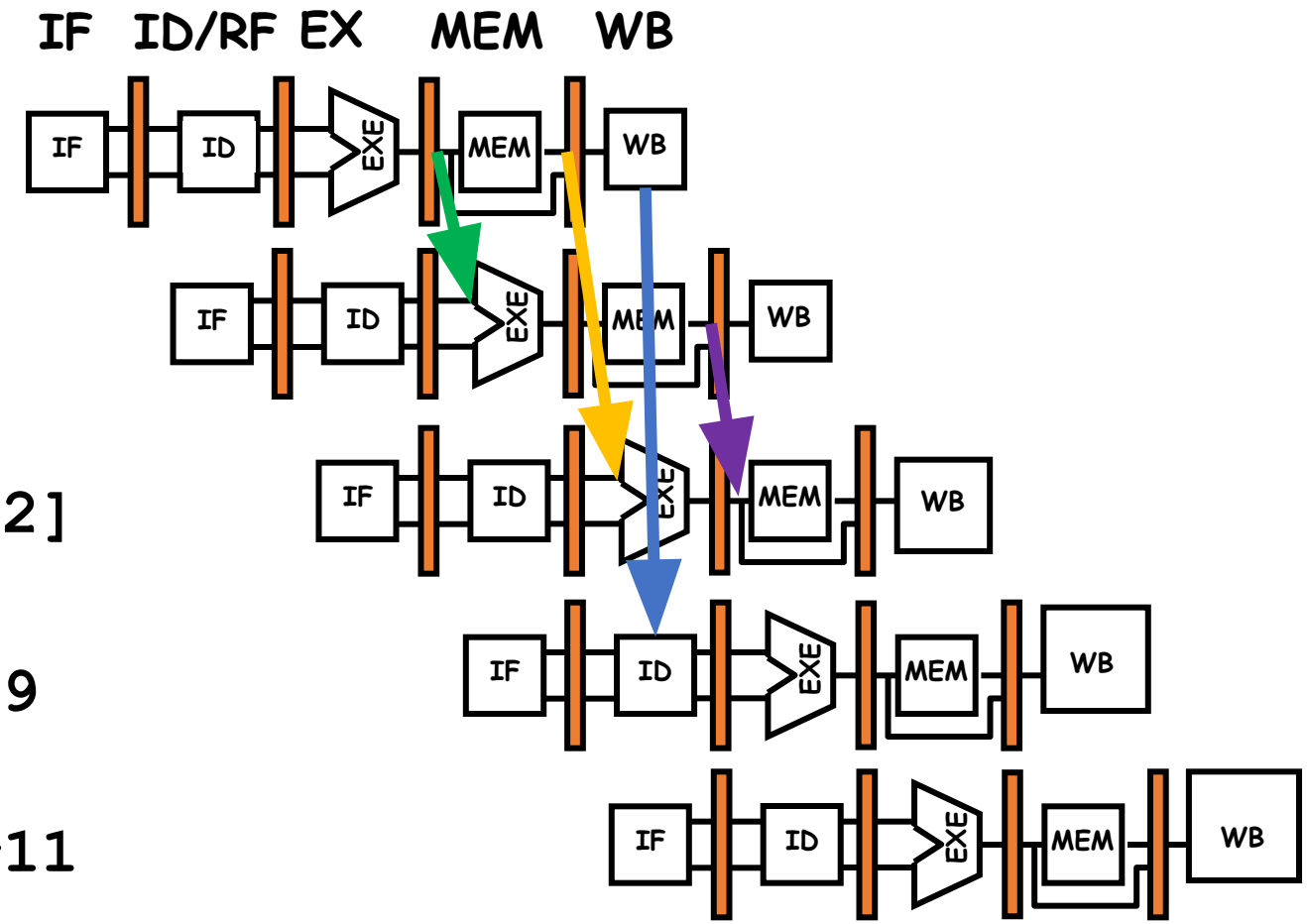
```
add r1,r2,r3
sub r4,r1,r3
and r6,r1,r7
or  r8,r1,r9
xor r10,r1,r11
```



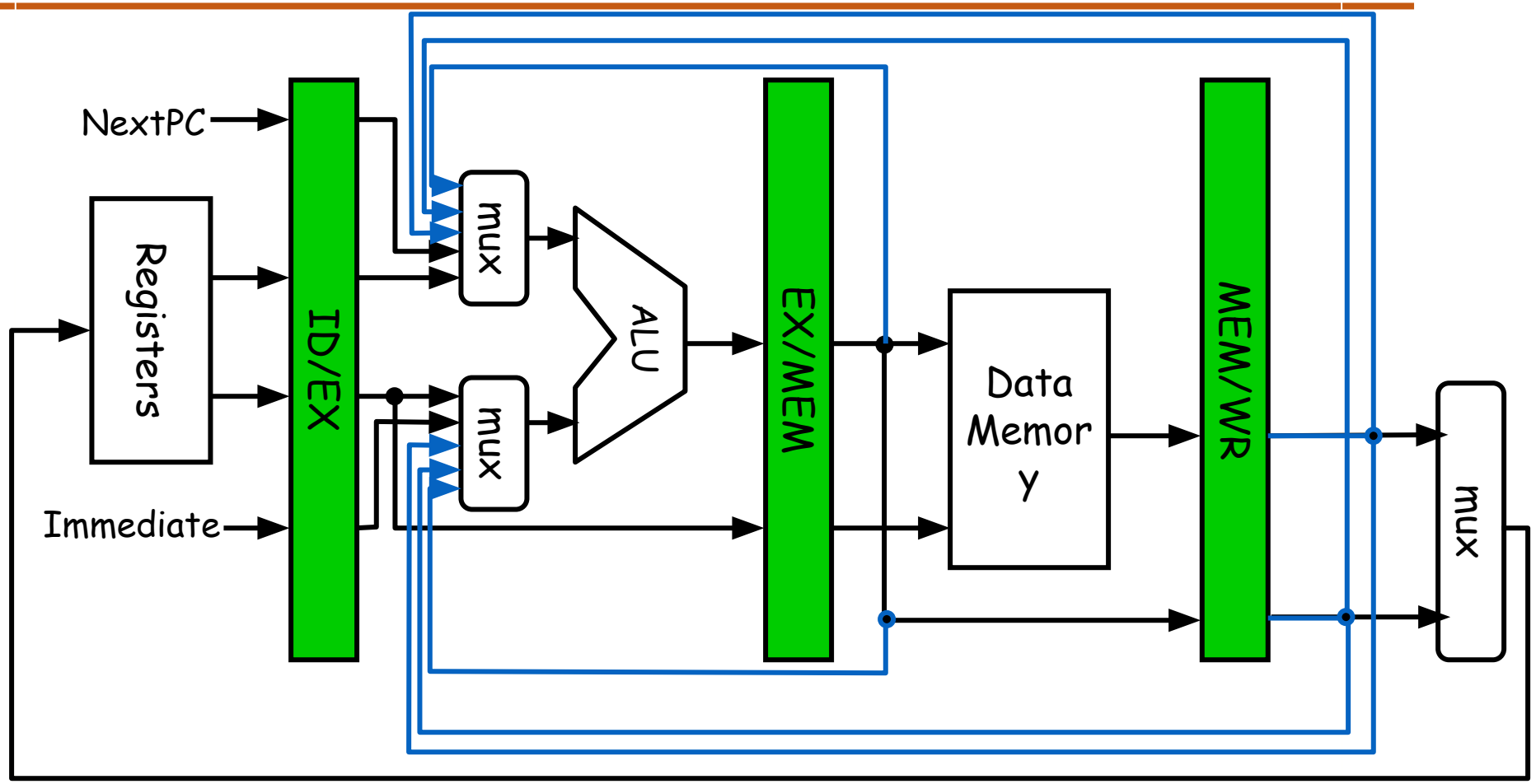
Hardware Solution 2: Data Forwarding

Instr. Order

```
add r1,r2,r3
ldr r4,[r1]
str r4,[r1,#2]
or  r8,r1,r9
xor r10,r1,r11
```



Hardware change for Forwarding



What circuit detects and resolves this hazard?

Does Forwarding eliminate all hazards??

Consider this example:

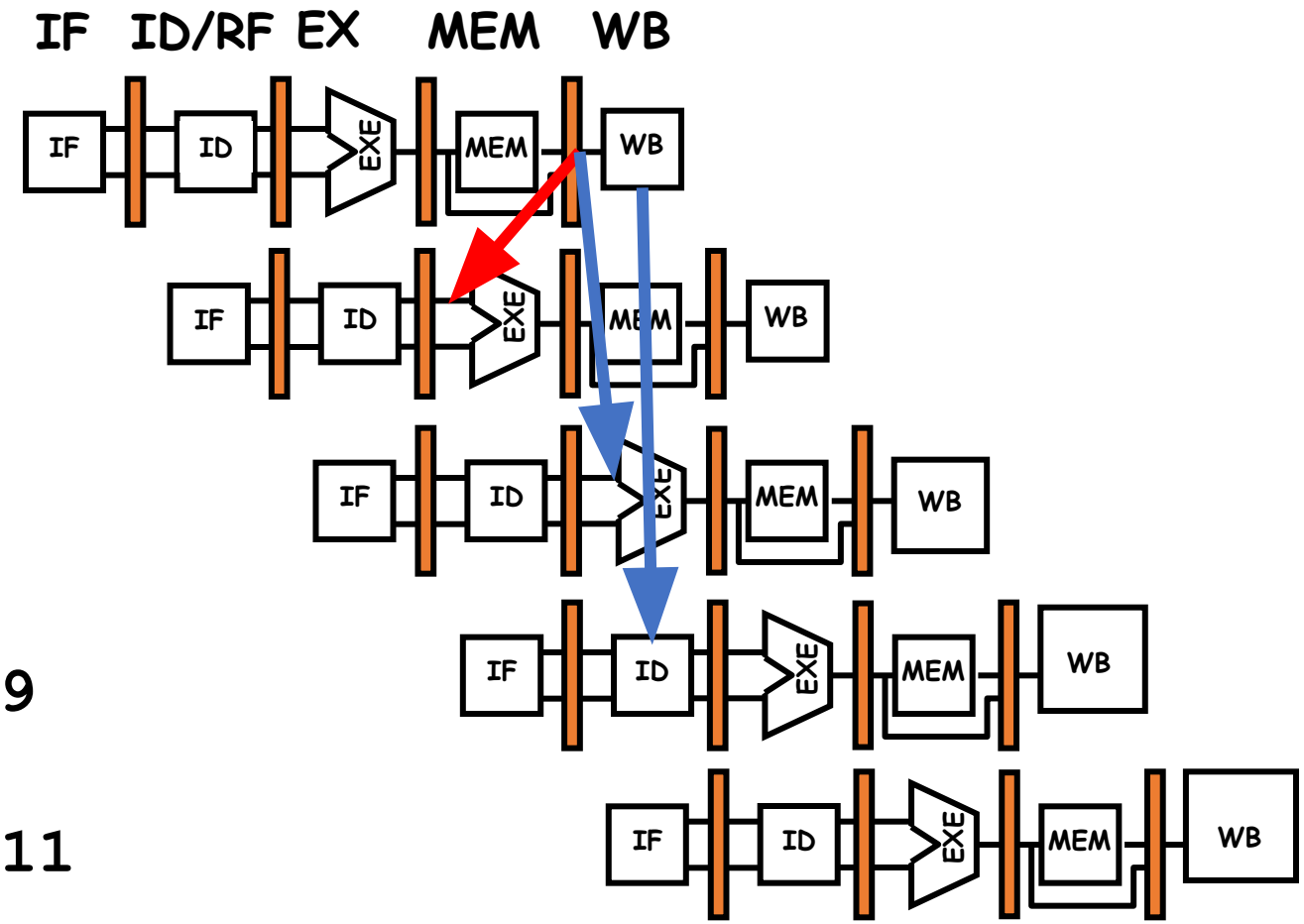
```
LDR     R0, [R1, #60]
ADD     R2, R0, R4
```

NO! You may need to stall after loads

Hardware Solution 2: Data Forwarding

Instr. Order

```
ldr r1,[r2]
sub r4,r1,r5
and r6,r1,r7
or  r8,r1,r9
xor r10,r1,r11
```



Think About It

```
ldr R7, [R2]
ldr R6, [R2, #4]
add R4, R5, R6
str R6, [R2, #4]
```

With forwarding we need to find only one independent instructions to place between them,
swapping the ldr instructions works:

```
ldr R6, [R2, #4]        IF  ID  EXE  MEM  WB
ldr R7, [R2]                IF  ID   EXE  MEM  WB
add R4, R5, R6                  IF   ID   EXE  MEM  WB
str R6, [R2, #4]                     IF   ID   EXE  MEM  WB
```

Without forwarding we need two independent instructions to place between them, so in addition a nop is added.

```
ldr R6, [R2, #4]
ldr R7, [R2]
nop
add R4, R5, R6
str R6, [R2, #4]
```

Why?

Think About It

a = b + c;

d = e – f;

Before:

```
ldr  R2, =a
ldr  R3, =b
add      R1, R2, R3
str  R1, =a
ldr  R2, =e
ldr  R3, =f
sub R1, R2, R3
str  R1, =d
```

Eliminate dependency by renaming registers

```
ldr  R2, =a
ldr  R3, =b
add      R1, R2, R3
str  R1, =a
ldr  R5, =e
ldr  R6, =f
sub R4, R5, R6
str  R4, =d
```

After Reordering & Assuming Data Forwarding

```
ldr  R2, =a
ldr  R3, =b
ldr  R5, =e
add      R1, R2, R3
ldr  R6, =f
str  R1, =a
sub R4, R5, R6
str  R4, =d
```

Draw the timing diagram check, if stalls are required?

Next Session

Control Hazards

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135

Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Pipeline Processor: Control Hazard

Dr. D. C. Kiran
Department of Computer Science and Engineering

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

- ~~3-Stage ARM Processor~~
- ~~5-Stage Pipeline Processor~~
- ~~Introduction to Pipeline Processor~~
- ~~What May Go Wrong?~~
- ~~Introduction to Hazards, Stalls,~~
- ~~Structural Hazards~~
- ~~Data Hazard~~
  - ~~RAW, WAR, WAW Hazards~~
- ~~Attacking Data Hazard~~
  - ~~Software Approach~~
  - ~~Hardware Approach~~
- Control Hazards

Microprocessor & Computer Architecture (µpCA)

Text 1: "Computer Organization and Design", Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

Reference 1:"Computer Architecture: A Quantitative Approach", Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.

Appendix C **Pipelining: Basic and Intermediate Concepts**

C.1	Introduction	C-2
C.2	The Major Hurdle of Pipelining—Pipeline Hazards	C-11
C.3	How Is Pipelining Implemented?	C-30
C.4	What Makes Pipelining Hard to Implement?	C-43
C.5	Extending the MIPS Pipeline to Handle Multicycle Operations	C-51
C.6	Putting It All Together: The MIPS R4000 Pipeline	C-61
C.7	Crosscutting Issues	C-70
C.8	Fallacies and Pitfalls	C-80
C.9	Concluding Remarks	C-81
C.10	Historical Perspective and References	C-81
	Updated Exercises by Diana Franklin	C-82

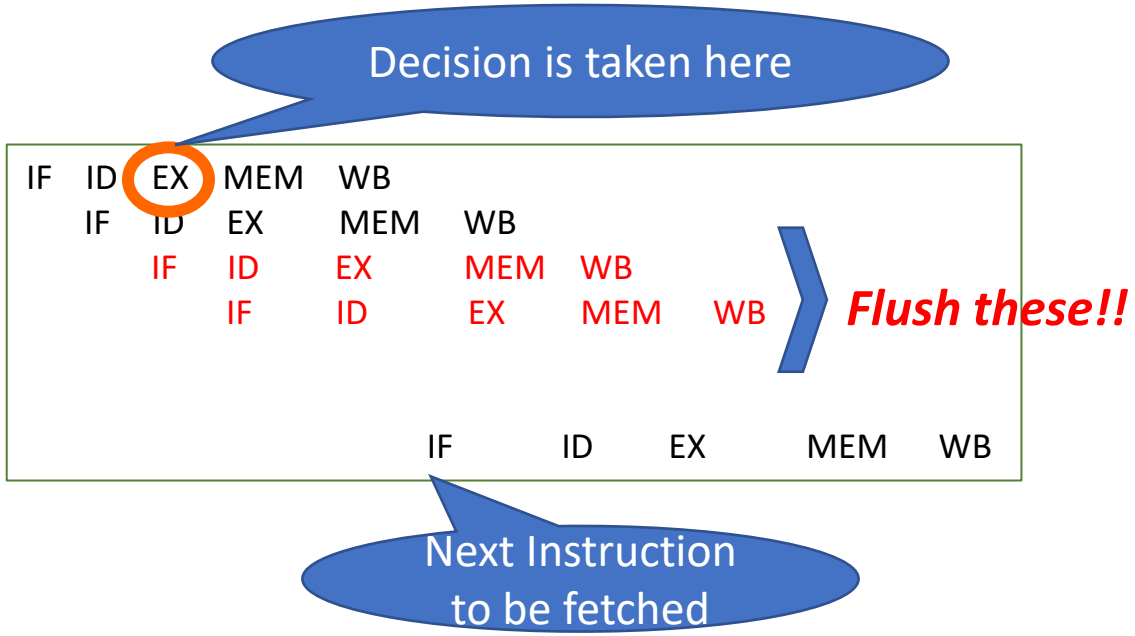
Control Hazards

- When the flow of instruction addresses is not sequential (i.e., PC = PC + 4); incurred by change of flow instructions
  - Conditional branches (**beq, bne**)
  - Unconditional branches (**b, bal**)
  - Exceptions
- Undesirable instructions get into the pipeline even before branch instruction is fetched, if branch is taken (Compare and Change PC in EX Stage)

Ex:

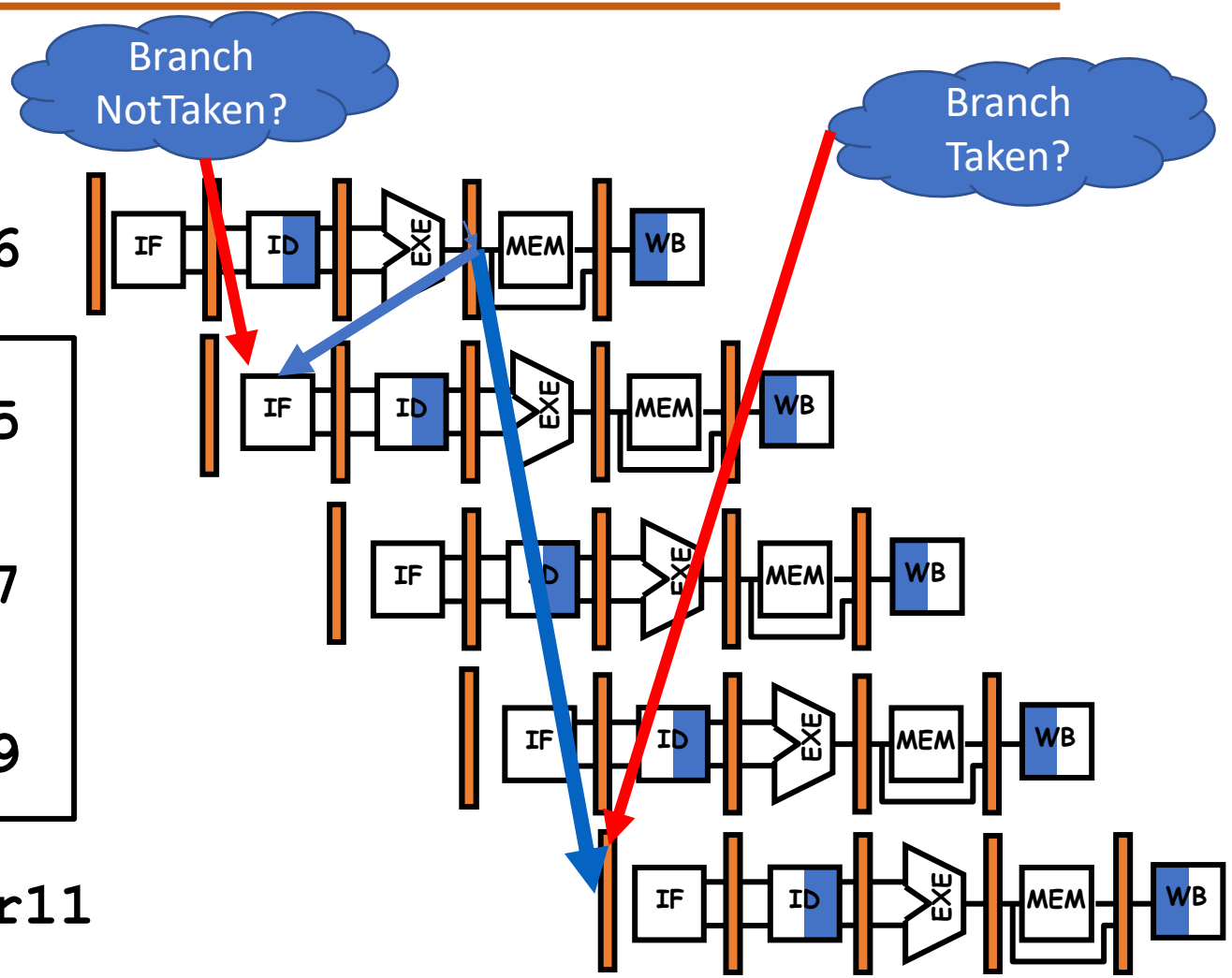
```
add   R1, R2, R3
beq   R1, R2, target
sub   R1, R4, R5
orr   R2, R7, R8

                 ....
target:          add      R1, R4, R5
```



Control Hazard on Branches: Three Stage Stall

```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```



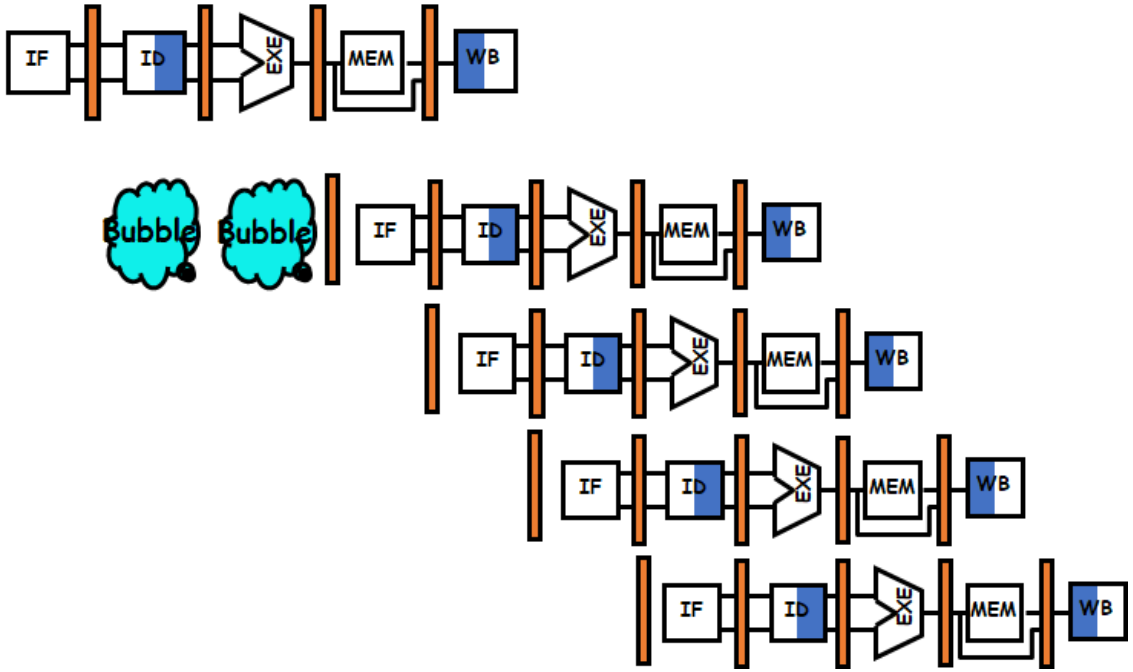
All branches waste 2 cycles irrespective of whether branch is taken or not.

Wasteful..

Better way???

- Reduce stall cycles
- Guess or Predict

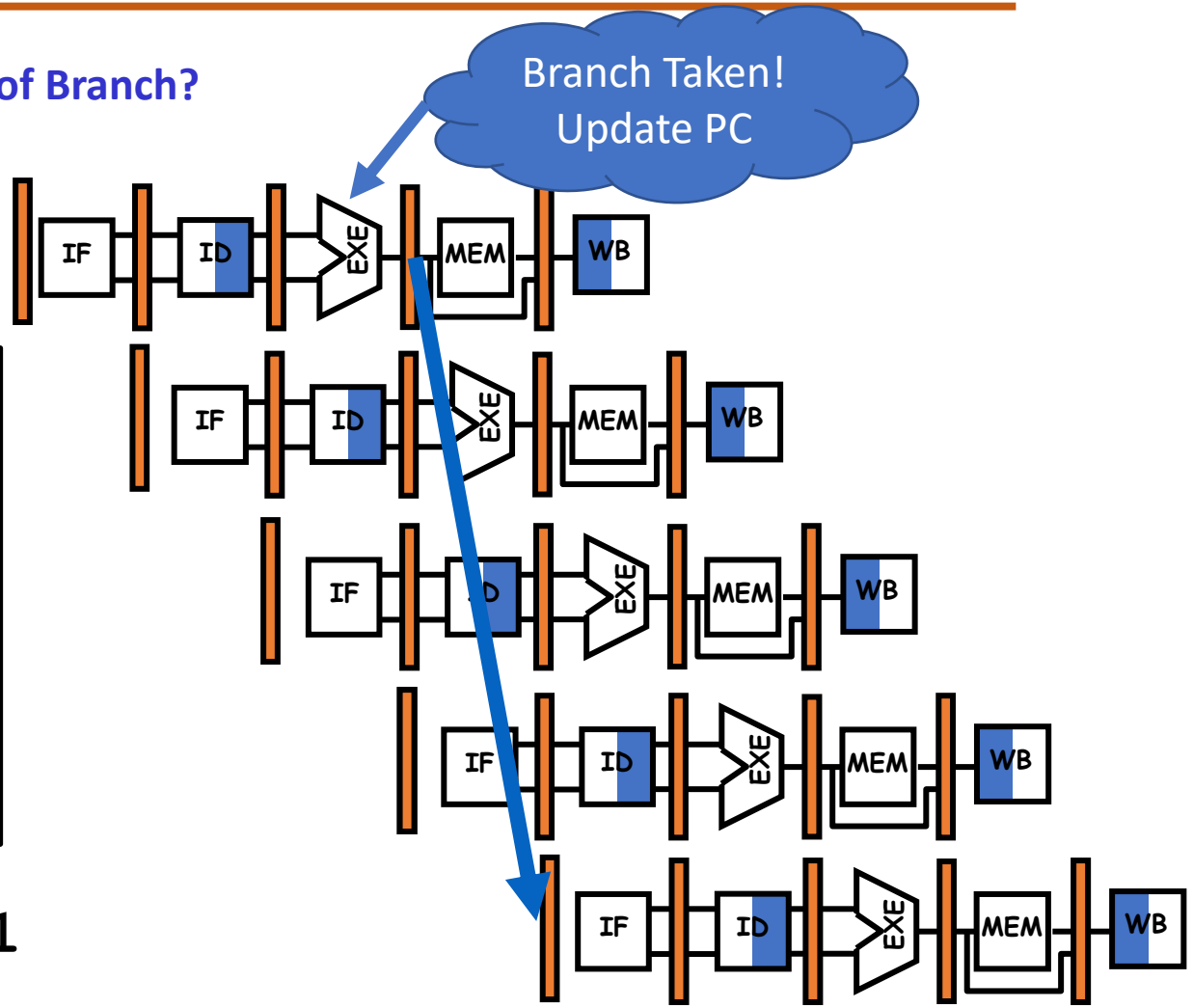
```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```



Control Hazard on Branches

When Do I know the result of Branch?
When Do I update PC?

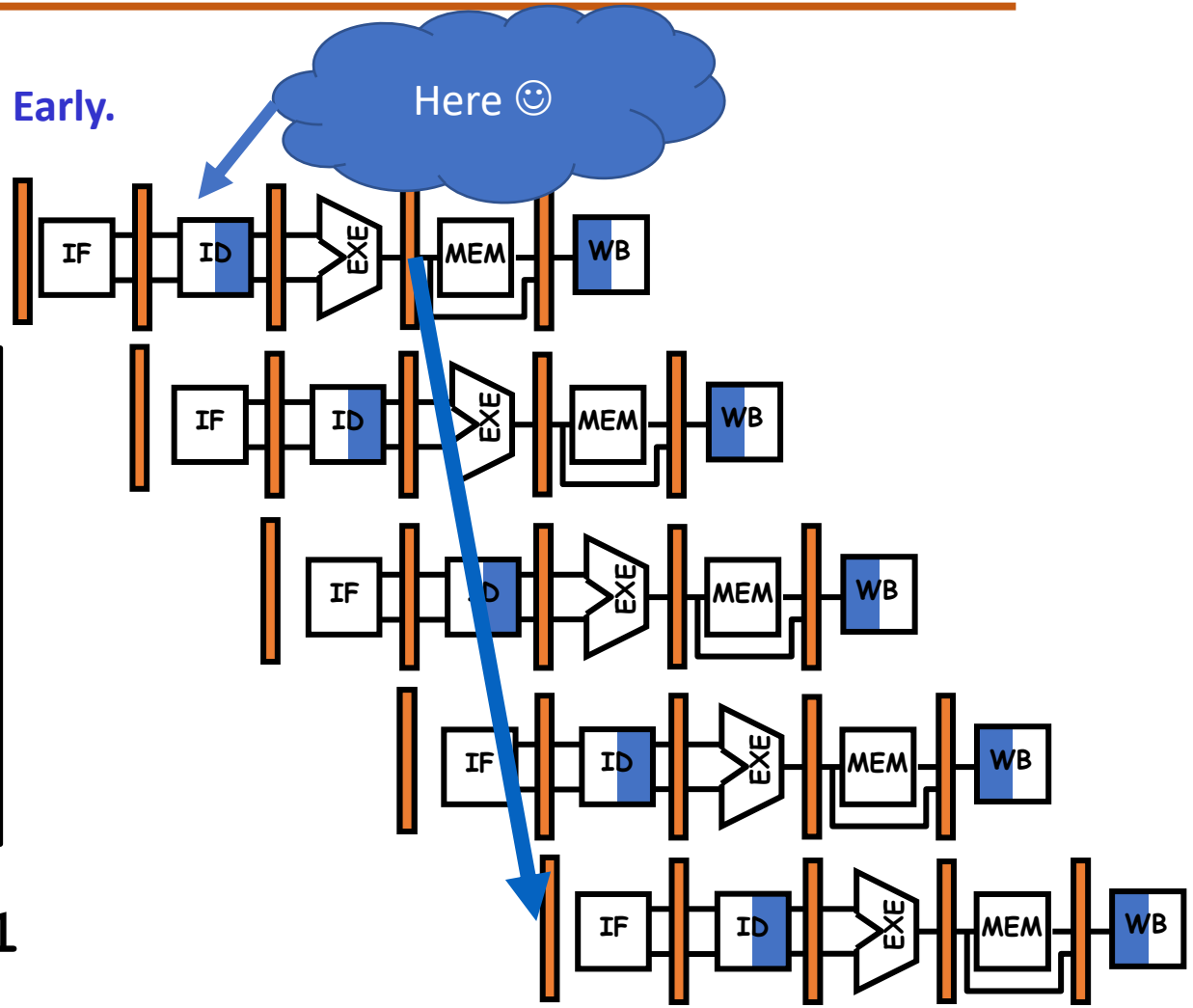
```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```



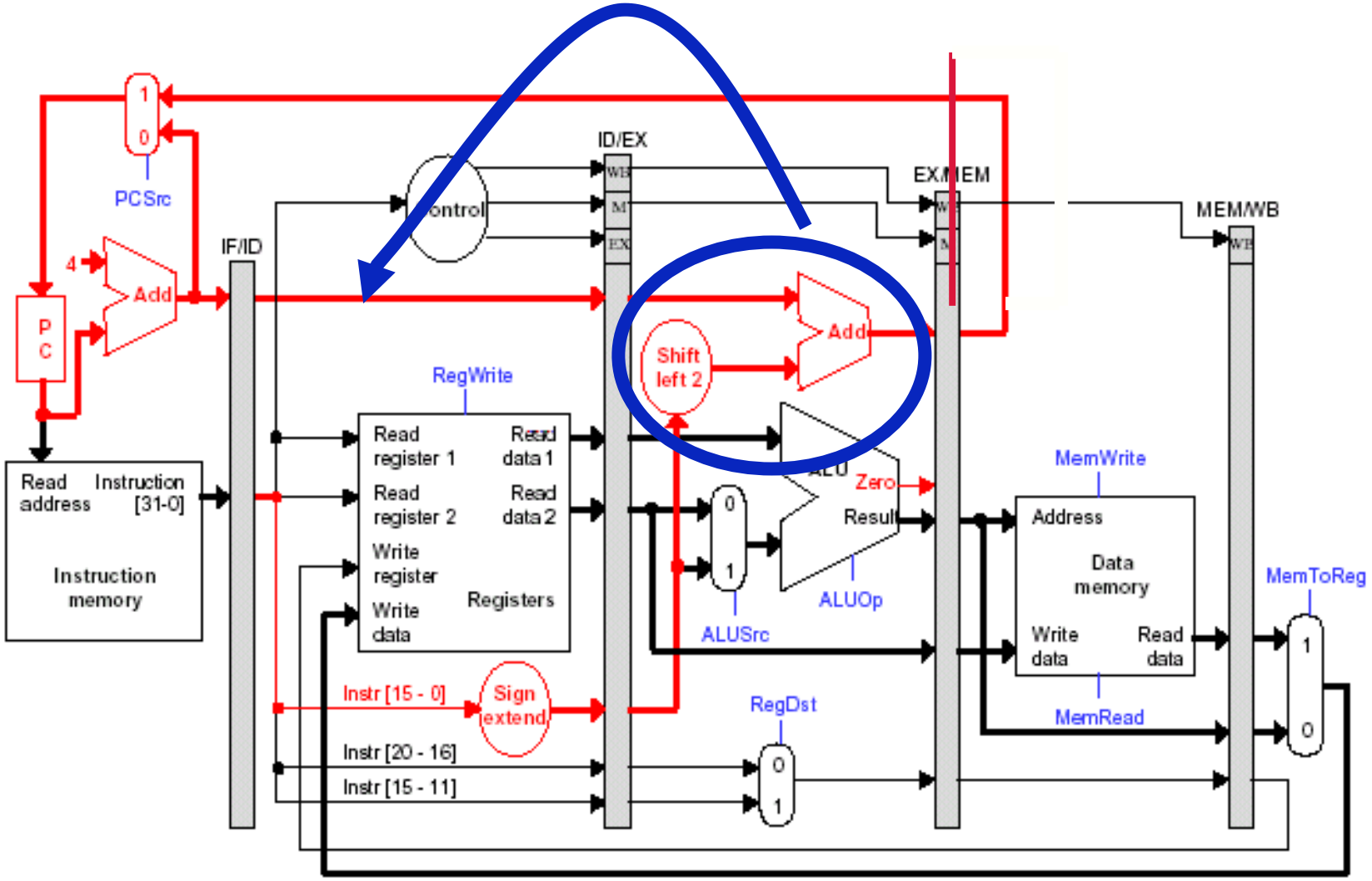
Control Hazard on Branches

Can I Reduce the Stalls by
Computing Result of Branch Early.
Update the PC Early.

```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```

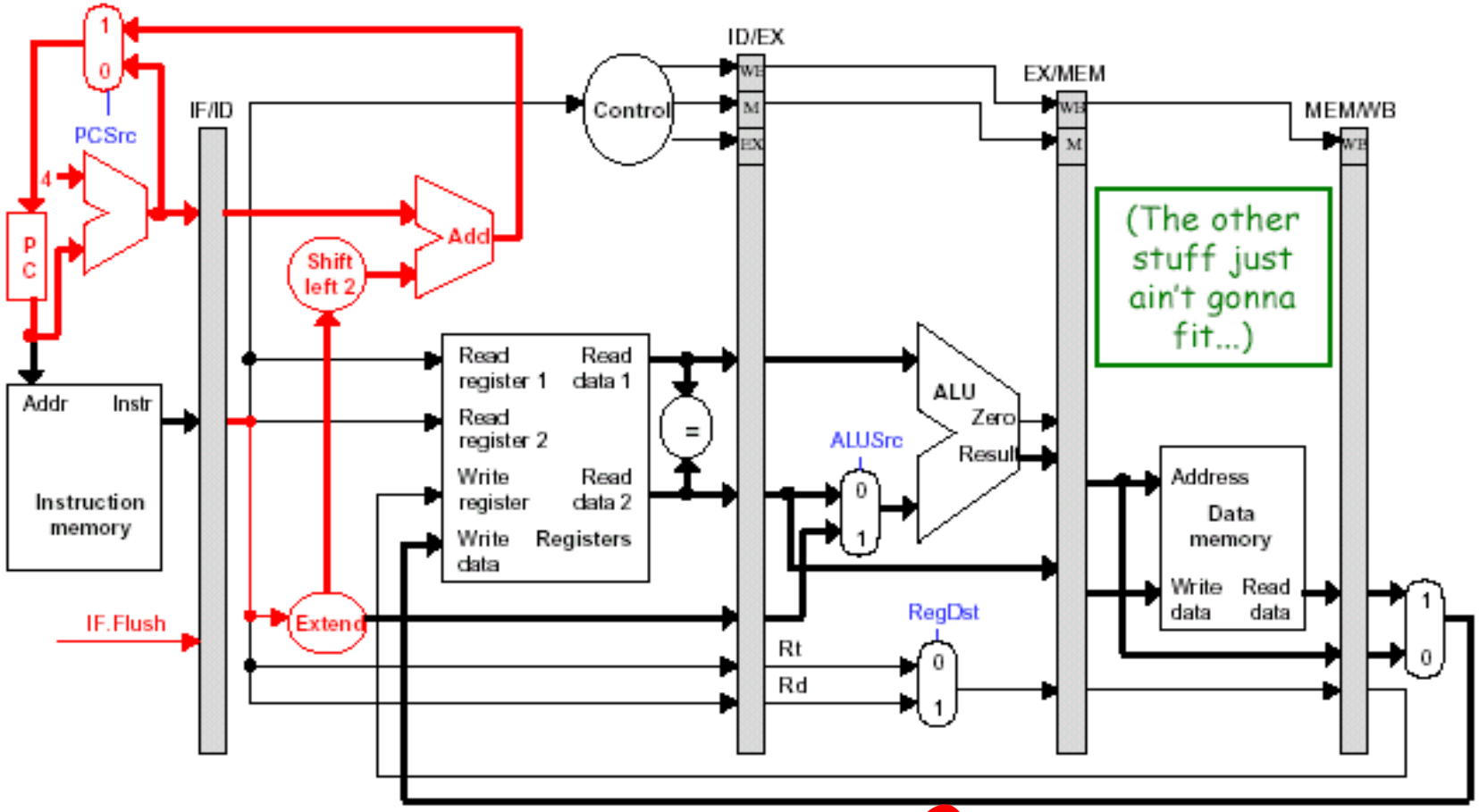


Reduce Stall Cycles?



- By moving updating of PC from EX to ID???
- Branch Delay = 1 cycle

Reduce Stall Cycles



- By moving updating of PC from EX to ID???
- Branch Delay = 1 cycle

- **Can we reduce it further by moving the updating of PC from ID to IF???** NO

Example

- Assume the following instruction mix:

Type	Frequency	
Arith/Logic	40%	
Load	30%	of which 25% are followed immediately by an instruction using the loaded value
Store	10%	
branch	20%	of which 45% are taken

- What is the resulting CPI for the pipelined processor with forwarding and branch address calculation in ID stage?

- CPI = Ideal CPI + Pipeline stall clock cycles per instruction

  = 1 + stalls by loads + stalls by branches

  = 1 + .3x.25x1 + .2 x .45x1

  = 1 + .075 + .09

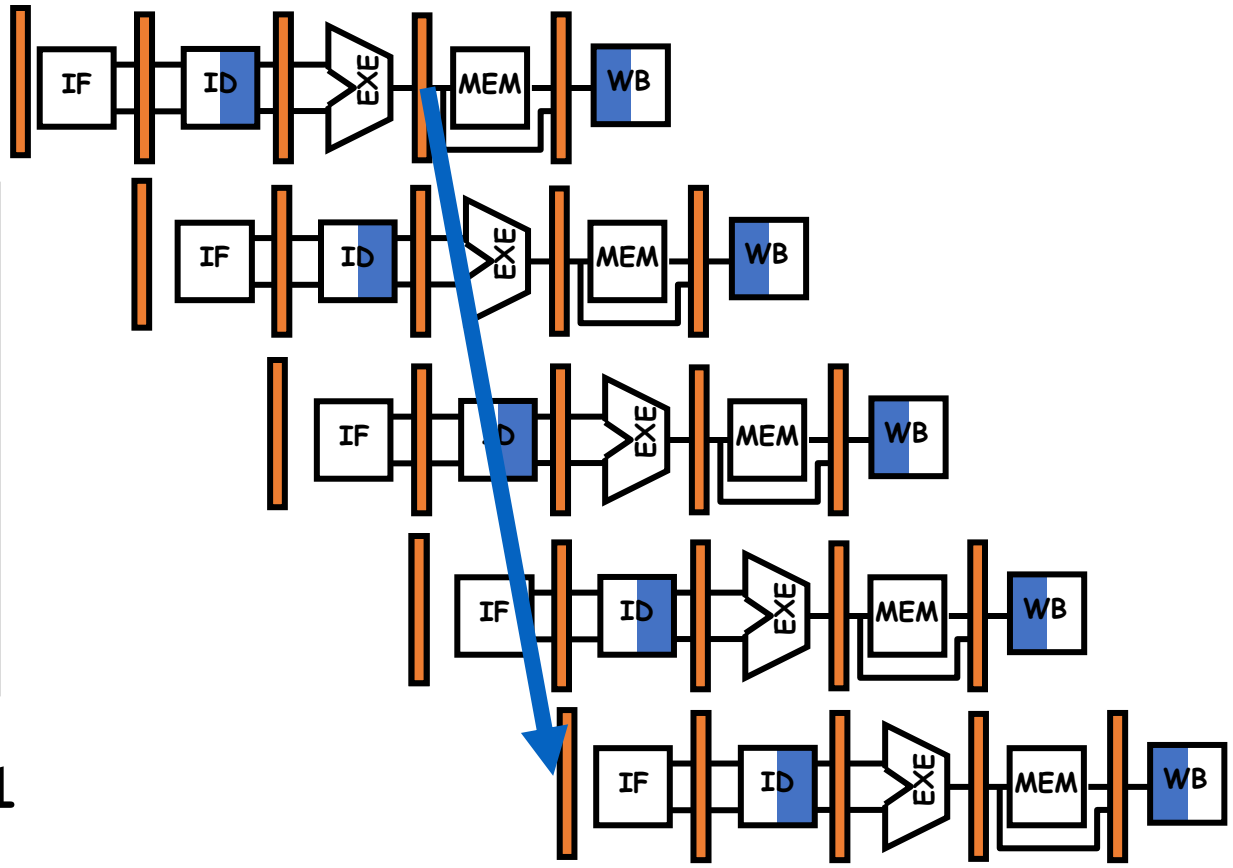
  = 1.165

Delayed Branching

- Similar to *insert no-ops*
- Compiler inserts….??
- Used to eliminate branch stalls
- Instruction after branch is known as delay slot
- Instruction in delay slot is always executed
- Fill the slots with useful instructions

Control Hazard on Branches: Delay Slot

```
10: beq r1,r3,36
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```

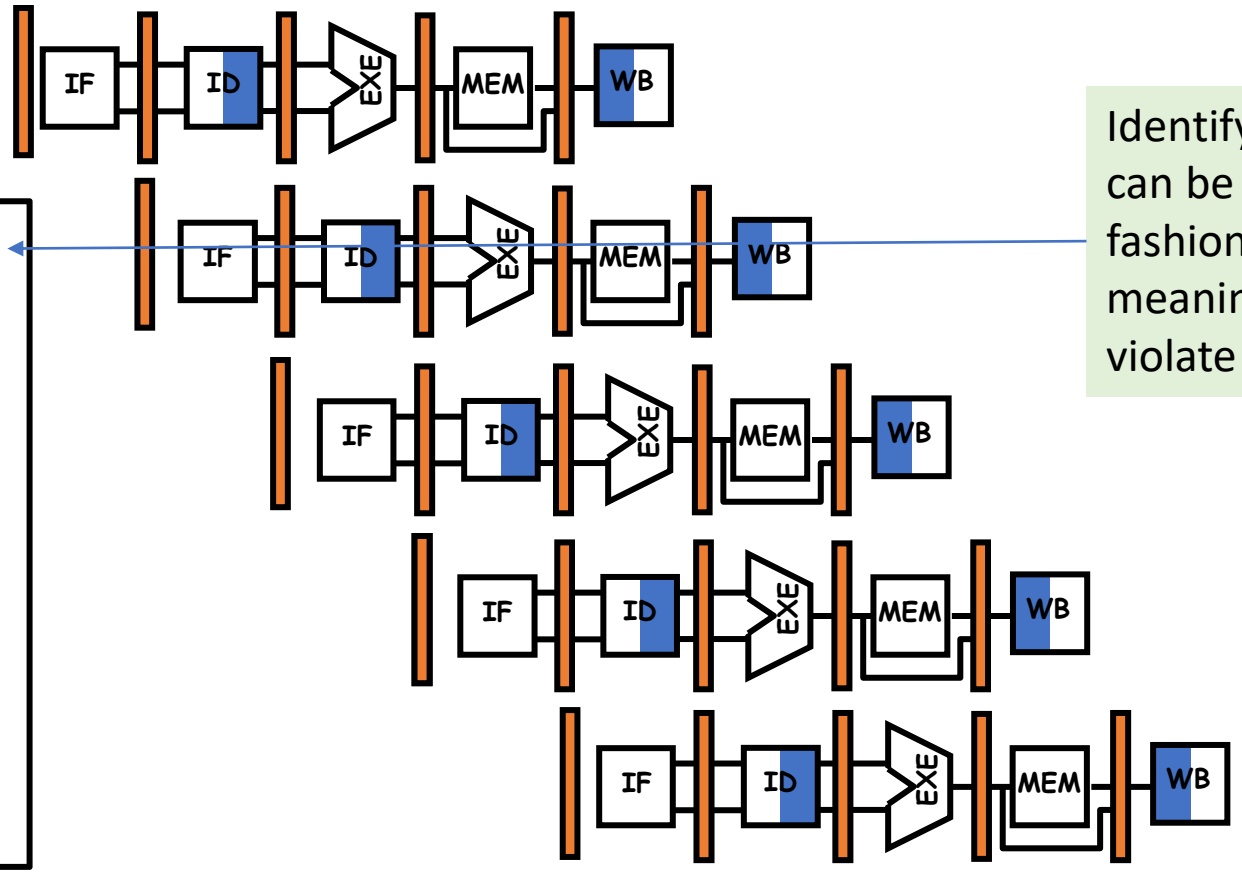



```
48: and r3,r1,r4
```

Identify the Instruction which can be executed out of order fashion i.e will not change the meaning of the program or not violate register write

Control Hazard on Branches: Delay Slot

```
10: beq r1,r3,36
48: sub r3,r1,r4
14: and r2,r3,r5
18: or  r6,r1,r7
22: add r8,r1,r9
36: xor r10,r1,r11
```



Identify the Instruction which can be executed out of order fashion i.e will not change the meaning of the program or not violate register write

A: Filling the delay slot Before Branch

- MUL R3, R4, R5

SUB R2, R1, R0

ADD R1, R2, R2

BEQZ R1, R7, there

ADD R1, R4, R7

there:

Delay Slot

```
# SUB R2, R1, R0
  ADD R1, R2, R2
  BEQZ R1, R7, there
  MUL R3, R4, R5
  ADD R1, R4, R7

there:
```

```
# In case, the compiler is
  not able to find suitable
  instruction, then
# MUL R3, R4, R5
  SUB R2, R1, R0
  ADD R1, R2, R2
  BEQZ R1, R7, there
  NOP
  ADD R1, R4, R7

there:
```

B: Filling the delay slot From Branch Target

```
SUB     R2,  R1, R0
ADD     R1, R2, R2
BEQZ    R1, R7, there
```

Delay Slot

```
ADD     R1, R4,  R7
```

Branch Not Taken Part

```
there: MUL R3, R4, R5
```

Branch Taken Part

```
SUB R2, R1, R0
ADD R1, R2, R2
BEQZ R1, R7,  there
MUL  R3, R4, R5
ADD R1, R4,  R7

there:
```

- Useful when it is predicted, 80 % Branch Taken & 20% Not Taken
- It should ensure no Register write violation in the Not Taken part of the instructions
- If Branch is not taken, then FLUSH i.e Waste of Cycle time

Microprocessor & Computer Architecture (µpCA)

C: Filling the delay slot From Fall through

```
SUB     R2,  R1, R0
ADD     R1, R2, R2
BEQZ    R1, R7, there
MUL R3, R4, R5
ADD     R1, R4,  R7
```

Branch Not Taken Part

```
there:
```

Branch Taken Part

Delay Slot

```
#   SUB R2, R1, R0
    ADD R1, R2, R2
    BEQZ R1, R7,  there
    MUL  R3, R4, R5
    ADD R1, R4,  R7

there:
```

- Useful when it is predicted, 80 % Branch Not Taken & 20% Taken
- It should ensure no Register write violation in the Not Taken part of the instructions
- If Branch is not taken, then FLUSH i.e Waste of Cycle time

Next Session

Branch Prediction

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135

Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Pipeline Processor: Branch Prediction

Dr. D. C. Kiran
Department of Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

- ~~3-Stage ARM Processor~~
- ~~5-Stage Pipeline Processor~~
- ~~Introduction to Pipeline Processor~~
- ~~What May Go Wrong?~~
- ~~Introduction to Hazards, Stalls,~~
- ~~Structural Hazards~~
- ~~Data Hazard~~
  - ~~RAW, WAR, WAW Hazards~~
- ~~Attacking Data Hazard~~
  - ~~Software Approach~~
  - ~~Hardware Approach~~
- ~~Control Hazards~~
- Branch History Table
- Branch Prediction

Microprocessor & Computer Architecture (µpCA)

Text 1: "Computer Organization and Design", Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

Reference 1:"Computer Architecture: A Quantitative Approach", Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.

Appendix C **Pipelining: Basic and Intermediate Concepts**

C.1	Introduction	C-2
C.2	The Major Hurdle of Pipelining—Pipeline Hazards	C-11
C.3	How Is Pipelining Implemented?	C-30
C.4	What Makes Pipelining Hard to Implement?	C-43
C.5	Extending the MIPS Pipeline to Handle Multicycle Operations	C-51
C.6	Putting It All Together: The MIPS R4000 Pipeline	C-61
C.7	Crosscutting Issues	C-70
C.8	Fallacies and Pitfalls	C-80
C.9	Concluding Remarks	C-81
C.10	Historical Perspective and References	C-81
	Updated Exercises by Diana Franklin	C-82

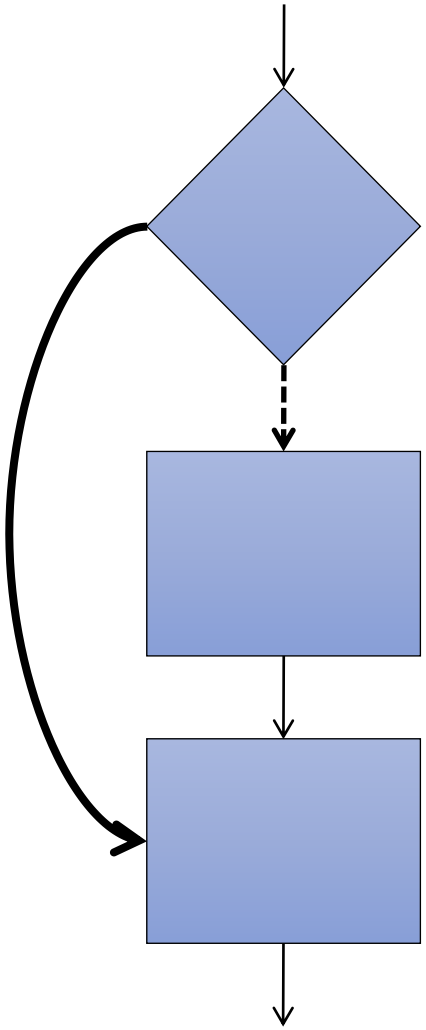
Branch Prediction

- Guessing the branch
  - One of the important problems in “Computer Architecture”
  - Static: prediction by compiler
  - Dynamic: prediction by hardware

- Static Prediction
  - Always Not Taken (easiest)
  - Always Taken
  - Taken / Not taken

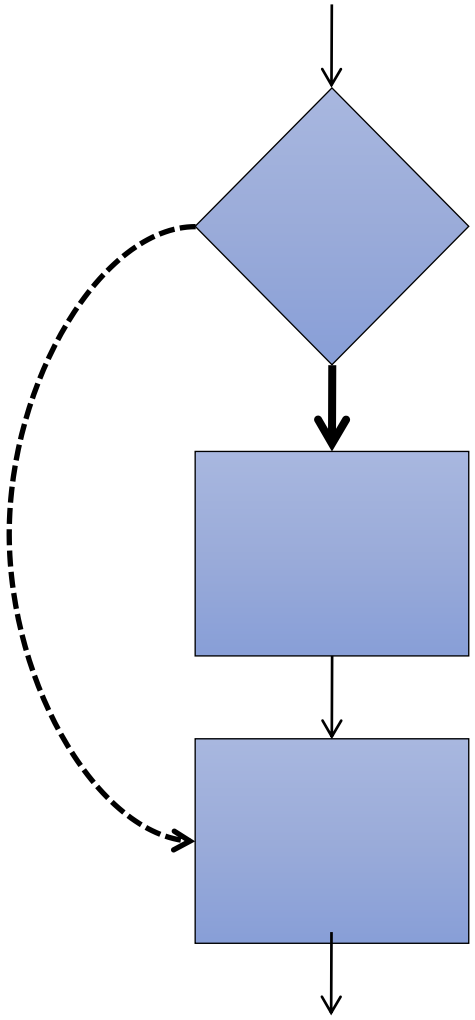
- Dynamic Prediction
  - 1- bit
  - 2-bit
  - others

Static Predictions



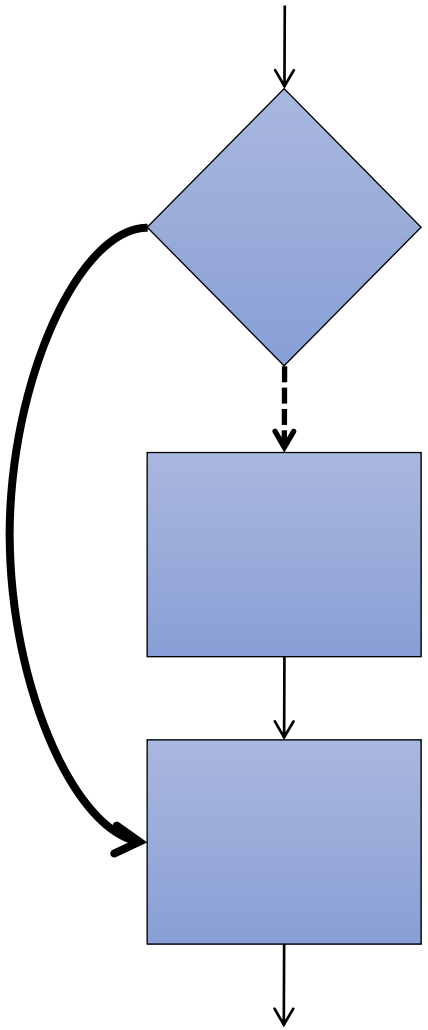
Always Taken

OR



Always Not Taken

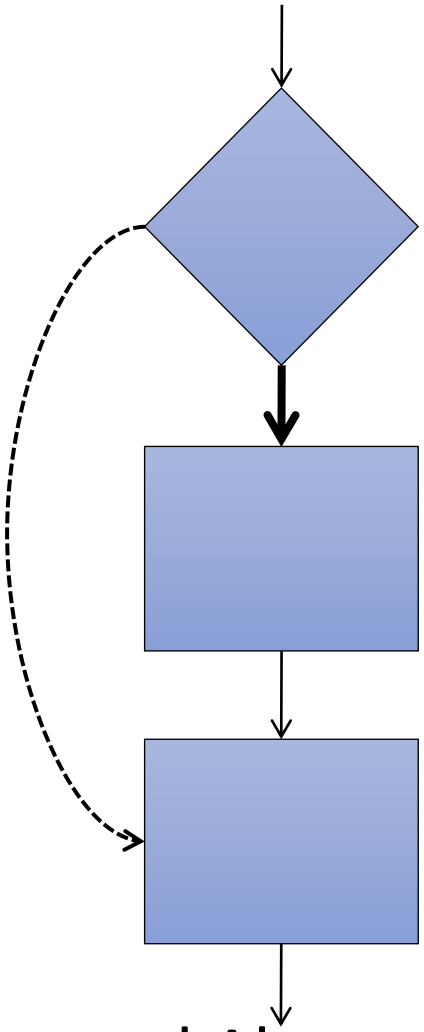
Static Predictions



Always Taken

- Early studies indicated that 2/3 of branches are taken
  - but 30% of those branches were unconditional!
- For conditional branches there appears to be no preferred direction.

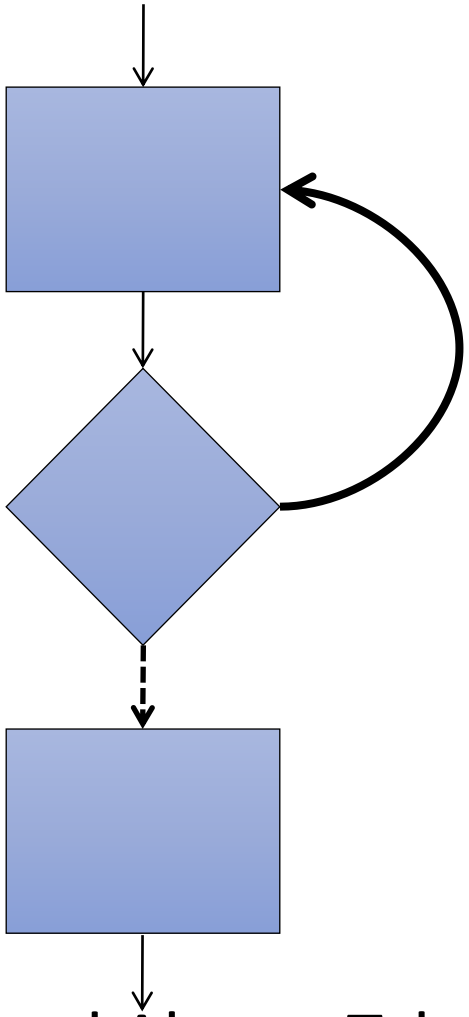
Alternative Static Predictions



Accuracy improvements are barely noticeable.

Static prediction based on profiling is slightly better.

Static branch-not-taken has no implementation cost on pipeline.



Forward Always Not Taken

Backward Always Taken

Microprocessor & Computer Architecture (µpCA)

Static Prediction→ Taken

Consider a program with two branch statement with the following behavior

T T N T TN TT T T N T T T T TT N T T T T TNT

How many Miss Prediction ?

5 miss predictions

T	T	NT	T	NT	T	T	T	NT	T	T	T	T	T	NT	T	T	T	T	NT
√	√	X	√	X	√	√	√	X	√	√	√	√	√	X	√	√	√	√	X

Microprocessor & Computer Architecture (µpCA)

Static Prediction→ Not Taken

Consider a program with two branch statement with the following behavior

T T N T TN TT T T N T T T T TT N T T T T TNT

How many Miss Prediction?

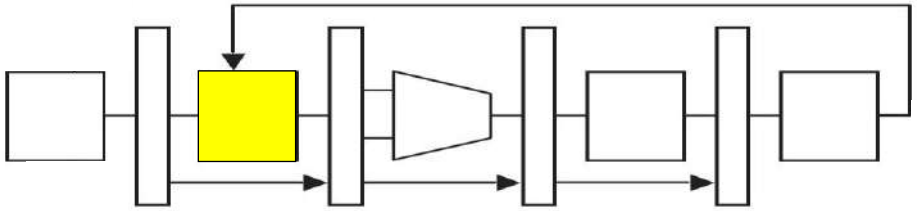
15 miss predictions

T	T	NT	T	NT	T	T	T	NT	T	T	T	T	T	NT	T	T	T	T	NT
X	X	√	X	√	X	X	X	√	X	X	X	X	X	√	X	X	X	X	√

Dynamic Predictors

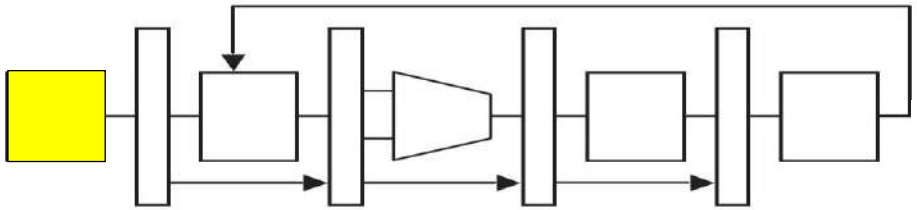
- Prediction of a given branch changes with the execution of the program.
  - **Simple**: a finite-state machine encodes the outcome of a few recent executions of the branch.
  - **Elaborate**: Not only early branch outcomes, but other correlated parts of the programs are considered.

When to Predict?



- Static prediction: at the Instruction Decode stage
  - Know that the instruction is a branch

- Dynamic prediction: at the Instruction Fetch stage
  - How to Know that the instruction is Branch
  - How to Know the target Address?
  - Should check Irrespective of branch instruction or not



How?

One-bit Predictor

- Simplest method:
  - A branch prediction buffer or Branch History Table (BHT) indexed by low address bits of the branch instruction.
  - Each buffer location (or BHT entry) contains one bit indicating whether the branch was recently taken or not.
  - Change bit on misprediction
  - Always mispredicts in first and last loop iterations.

Microprocessor & Computer Architecture (µpCA)

Branch History Table or Branch Prediction Buffer:-> One Bit

Address	Branch Address	Target Address	Prediction
000	432	456	1
001	97	123	0
010	130	143	1
011	67	98	0
100	84	244	1
101	261	532	1
110	518	786	1
111	1031	1134	0

32 bit 84

0	.....	0	0	0	1	0	1	1	0	0

Ex:

000080 : add R1, R2, R3
0000084: beq R1, R2, 244
0000088: sub R1, R4, R5
orr R2, R7, R8

....
0000244: add R1, R4, R5

PC=84	IF	ID	EXE	MEM	WB
PC=88 or 244	IF				

Branch History Table or Branch Prediction Buffer:-> One Bit

Address	Branch Address	Target Address	Prediction
000	432	456	1
001	97	123	0
010	130	143	1
011	67	98	0
100	84	244	1
101	261	532	1
110	518	786	1
111	1031	1134	0

32 bit 84

0	.....	0	0	0	1	1	0	1	0	0

PC=244

IF	ID	EXE	MEM	WB

IF	ID	EXE	MEM	WB

If Branch Taken, update PC

Ex:

000080 : add R1, R2, R3
0000084: beq R1, R2, 244
0000088: sub R1, R4, R5
orr R2, R7, R8

....

0000244: add R1, R4, R5

Branch History Table or Branch Prediction Buffer:-> One Bit

Address	Branch Address	Target Address	Prediction
000	432	456	1
001	97	123	0
010	130	143	1
011	67	98	0
100	84	244	0
101	261	532	1
110	518	786	1
111	1031	1134	0

32 bit 84

0	.....	0	0	0	1	1	0	1	0	0

PC=88

IF	ID	EXE	MEM	WB
	IF	ID	EXE	MEM	WB

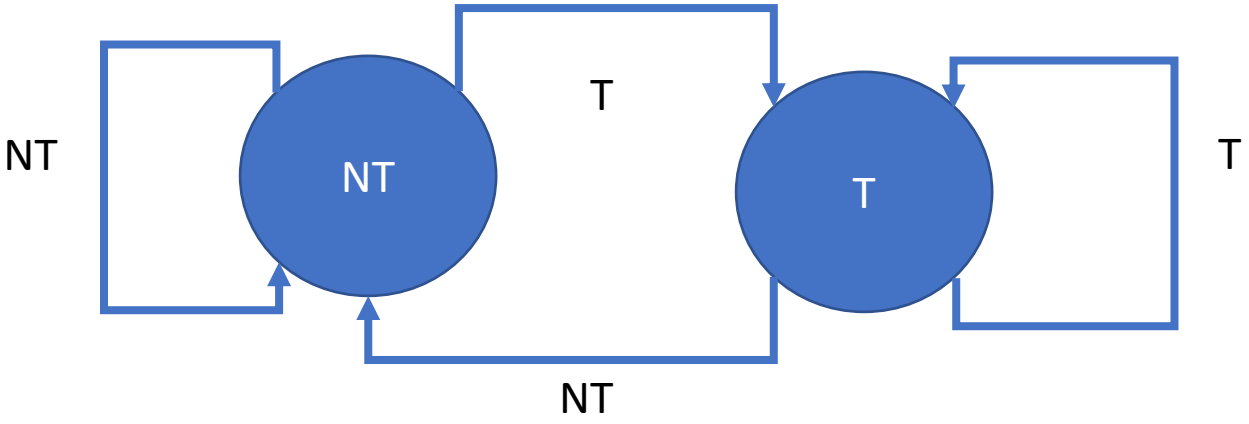
If Branch Not Taken
Continue with
normal execution

Ex:

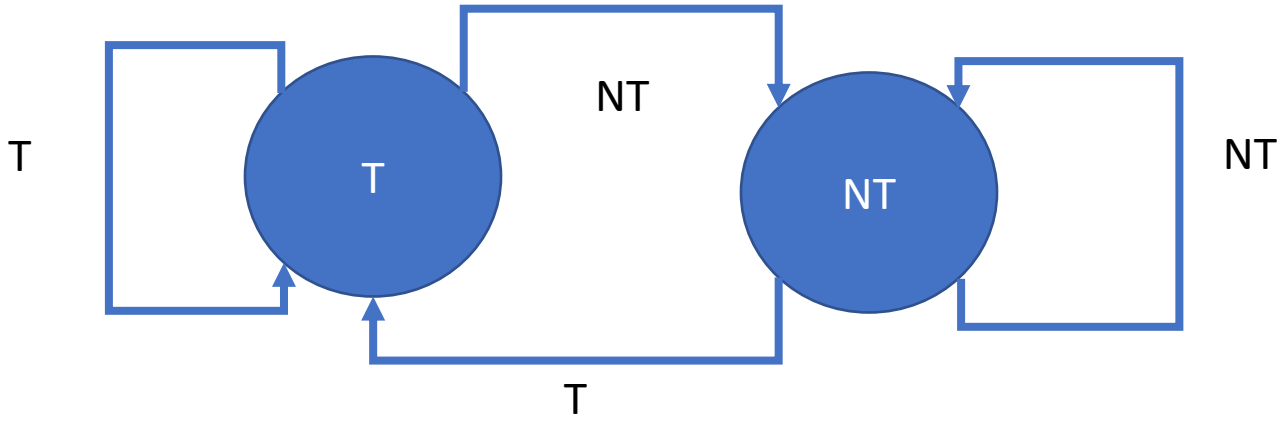
000080 : add R1, R2, R3
0000084: beq R1, R2, 244
0000088: sub R1, R4, R5
orr R2, R7, R8

....
0000244: add R1, R4, R5

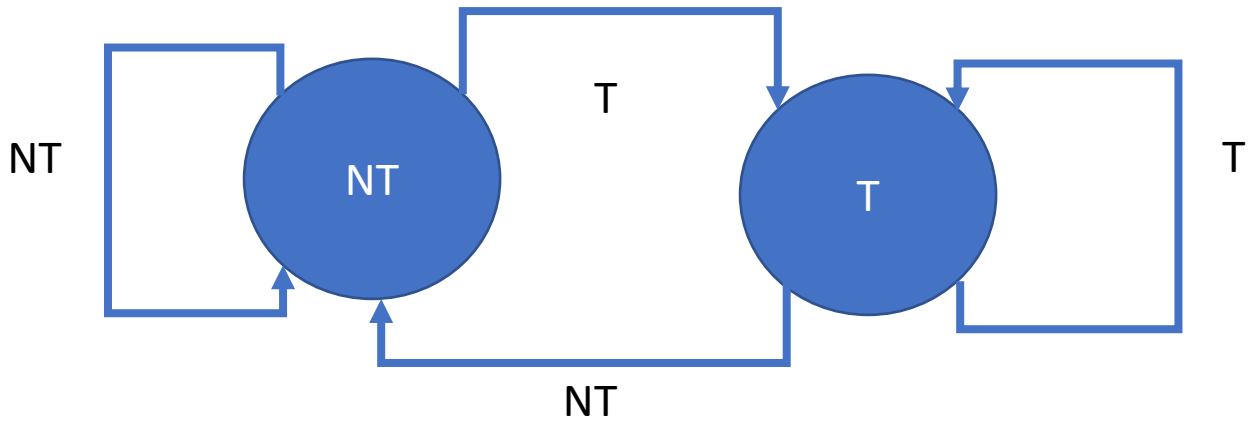
1 Bit Branch Prediction



OR



1 Bit Branch Prediction



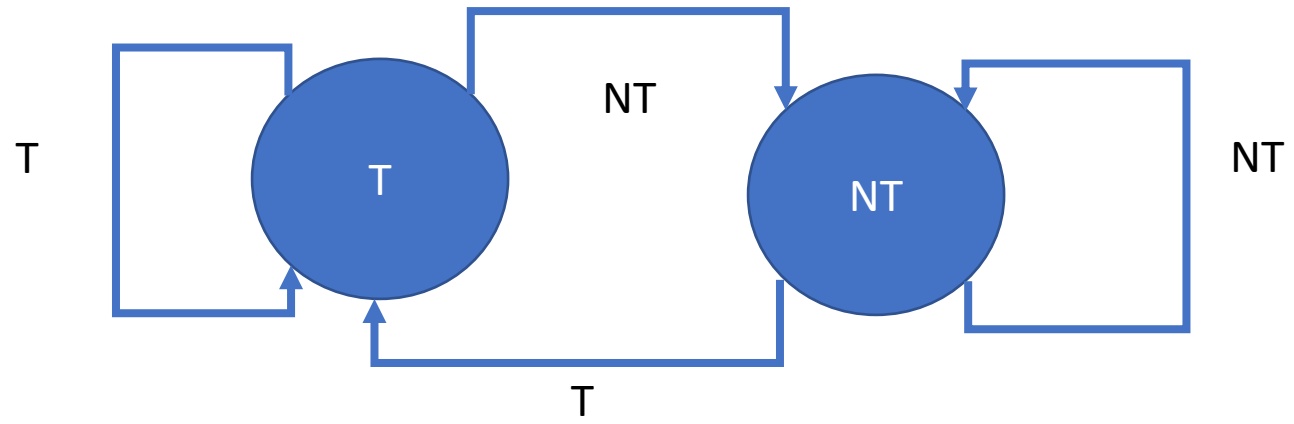
Consider a program with two branch statement with the following behavior

T T N T TN TT T T N T T T T TT N T T T T TNT

How many Miss Prediction if the Initial state is NT? **10 miss predictions**

T	T	NT	T	NT	T	T	T	NT	T	T	T	T	T	NT	T	T	T	T	NT
X	√	X	X	X	X	√	√	X	X	√	√	√	√	X	X	√	√	√	X

1 Bit Branch Prediction



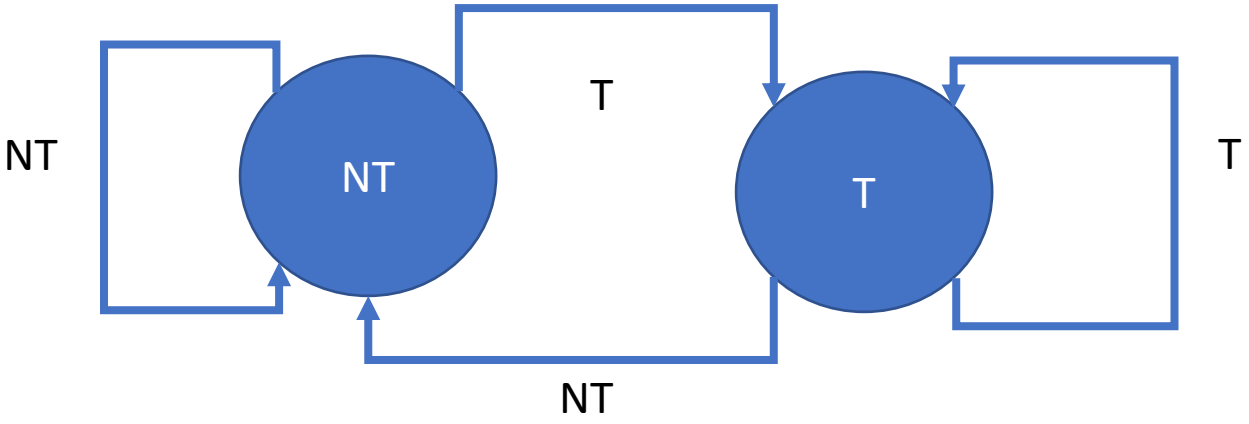
Consider a program with two branch statement with the following behavior

T T N T TN TT T T N T T T T TT N T T T T TNT

How many Miss Prediction if the Initial state is NT? **9 miss predictions**

T	T	NT	T	NT	T	T	T	NT	T	T	T	T	T	NT	T	T	T	T	NT
√	√	X	X	X	X	√	√	X	X	√	√	√	√	X	X	√	√	√	X

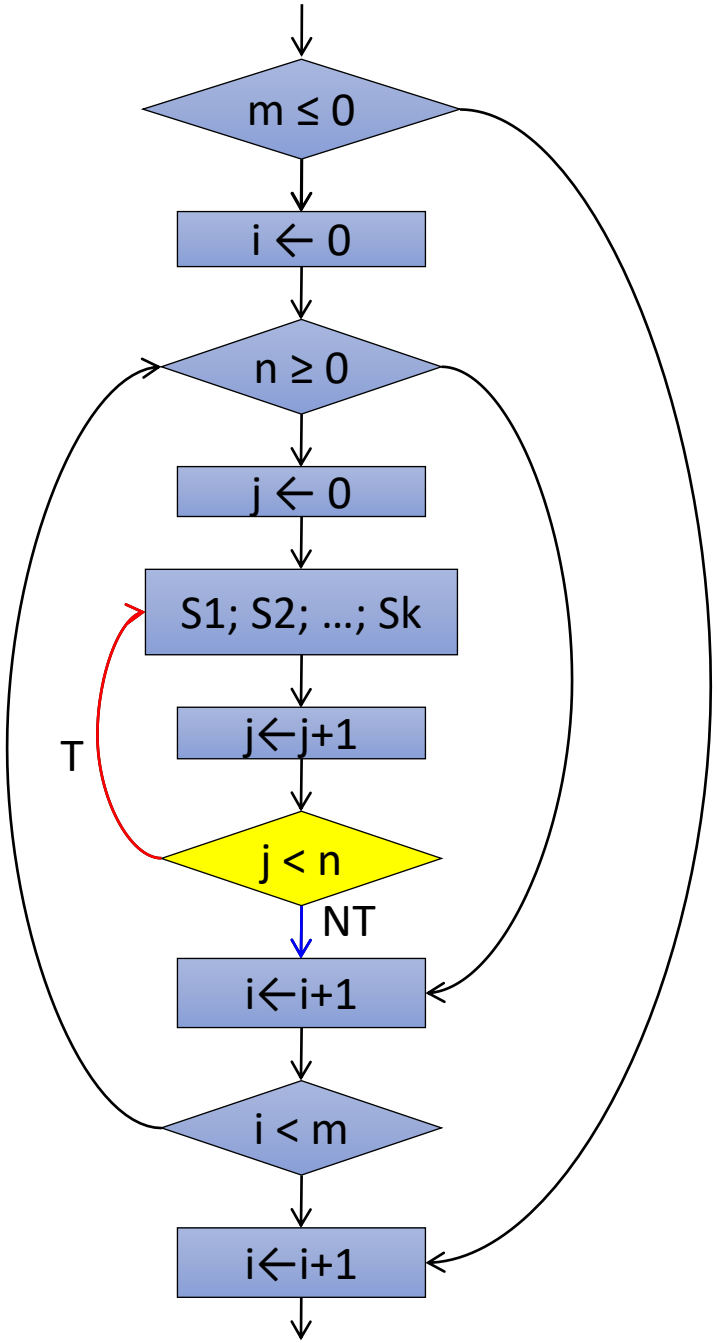
1 Bit Branch Prediction




```
for(j=0; j<n ; j++)
        begin S1; S2; ...; Sk end;
```

	1-bit	
i	**Pred**	**Outc**
0	NT	T
1	T	T
n	T	NT
0	NT	T
1	T	T

2 Misprediction for each loop




```
for(i=0 ; i < m ; i++)
    for(j=0; j<n ; j++)
        begin S1; S2; …; Sk end;
```

		1-bit	
i	**j**	**Pred**	**Outc**
0	0	NT	T
0	1	T	T
0	n	T	NT
1	0	NT	T
1	1	T	T

$2 \times m$ misspredictions

Next Session

2 Bit Branch Prediction

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135

Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Pipeline Processor: Branch Prediction 2

Dr. D. C. Kiran
Department of Computer Science and Engineering

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

- ~~3-Stage ARM Processor~~
- ~~5-Stage Pipeline Processor~~
- ~~Introduction to Pipeline Processor~~
- ~~What May Go Wrong?~~
- ~~Introduction to Hazards, Stalls,~~
- ~~Structural Hazards~~
- ~~Data Hazard~~
  - ~~RAW, WAR, WAW Hazards~~
- ~~Attacking Data Hazard~~
  - ~~Software Approach vs ardware Approach~~
- ~~Control Hazards~~
- ~~Branch History Table~~
- ~~1 bit Branch Prediction~~
- 2 bit Branch Prediction

Microprocessor & Computer Architecture (µpCA)

Text 1: "Computer Organization and Design", Patterson, Hennessey, 5th Edition, Morgan Kaufmann, 2014.

Reference 1:"Computer Architecture: A Quantitative Approach", Hennessey, Patterson, 5th Edition, Morgan Kaufmann, 2011.

Appendix C **Pipelining: Basic and Intermediate Concepts**

C.1	Introduction	C-2
C.2	The Major Hurdle of Pipelining—Pipeline Hazards	C-11
C.3	How Is Pipelining Implemented?	C-30
C.4	What Makes Pipelining Hard to Implement?	C-43
C.5	Extending the MIPS Pipeline to Handle Multicycle Operations	C-51
C.6	Putting It All Together: The MIPS R4000 Pipeline	C-61
C.7	Crosscutting Issues	C-70
C.8	Fallacies and Pitfalls	C-80
C.9	Concluding Remarks	C-81
C.10	Historical Perspective and References	C-81
	Updated Exercises by Diana Franklin	C-82

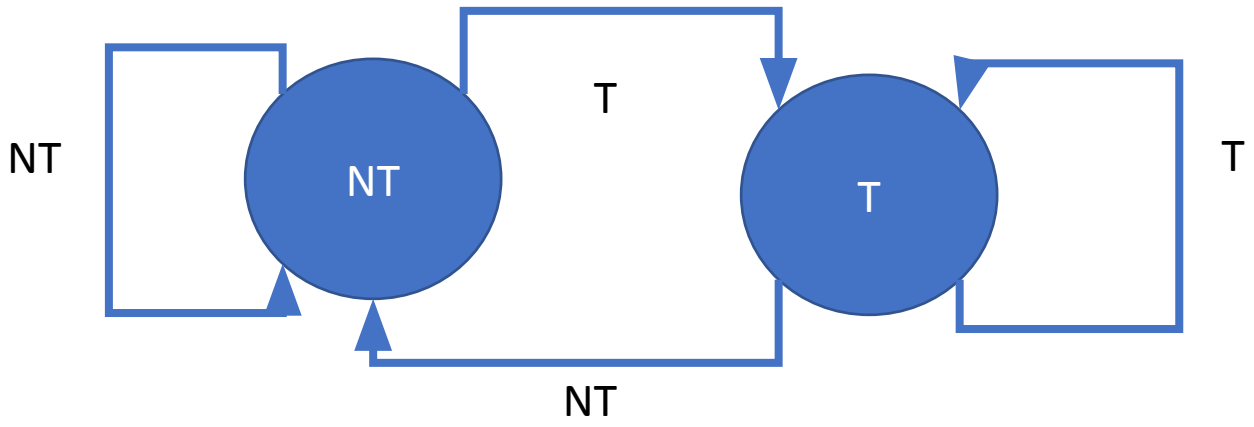
Towards 2-Bit Predictor

What is the problem with 1-Bit predictor

- **Aliasing Problem**
  - branches with same lower order bits will reference the same entry, causing mutual prediction

- **Shortcomings with loops**
  - Always mispredict twice for every loop
    - Mispredict upon exiting a loop, since this is a surprise
    - If we repeat the loop, we'll miss again since we'll predict, branch not taken

1 Bit Branch Prediction

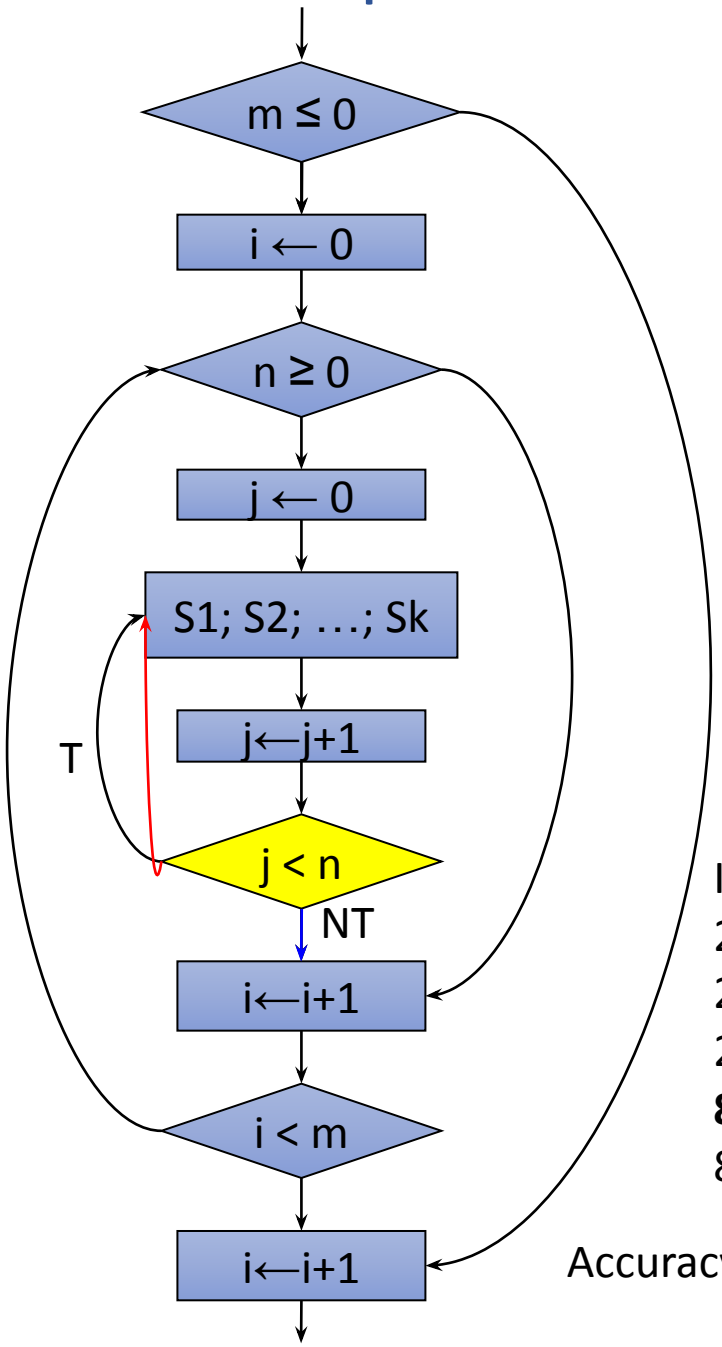



```
for(j=0; j<n ; j++)
        begin S1; S2; …; Sk
end;
```

2 Misprediction for each loop

	1-bit	
i	Pred	Outc
0	NT	T
1	T	T
n	T	NT
0	NT	T
1	T	T

What is the problem with 1-Bit predictor




```
for(i=0 ; i < m ; i++)
    for(j=0; j<n ; j++)
        begin S1; S2; …; Sk end;
```

		1-bit	
i	**j**	**Pred**	**Outc**
0	0	NT	T
0	1	T	T
0	n	T	NT
1	0	NT	T
1	1	T	T

If m= 100 & n=10

2 miss prediction per Iteration

2 × *m* miss predictions for iterations

2x100= 200 miss predictions

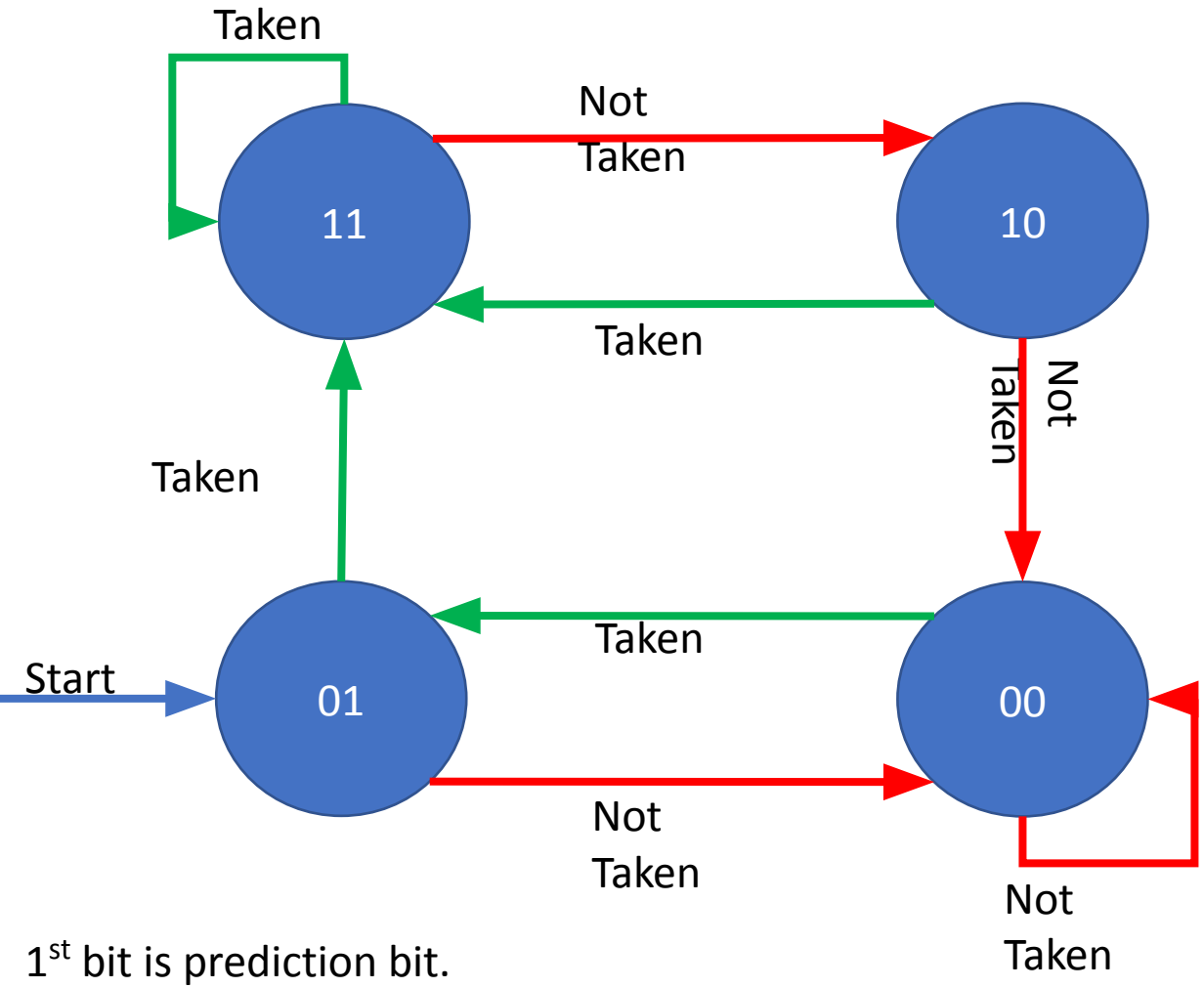
8 x m Correct Predictions (Single Iteration)

8x100= 8000 Correct Predictions for nxm iterations

Accuracy of Correct Predictions = (800/1000)x100= 80%

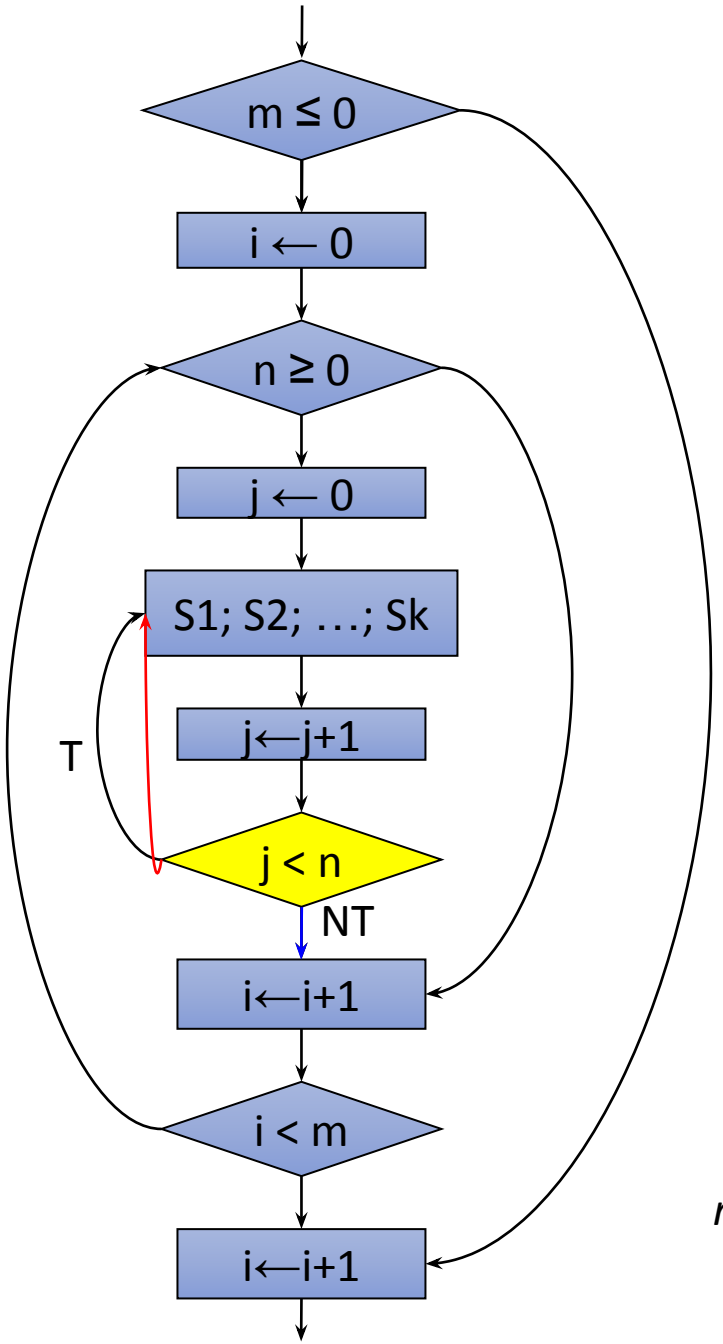
Two -bit Predictor

Think Twice Before Mis predicting



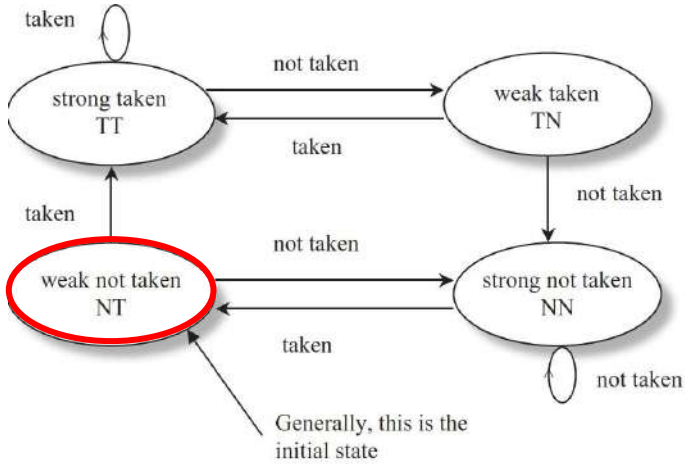
00: Strong Not Taken
01: Weak Not Taken
10: Weak Taken
11: Strong Taken

1st bit is prediction bit.
2nd bit is conviction bit (How much sure that the prediction is correct)

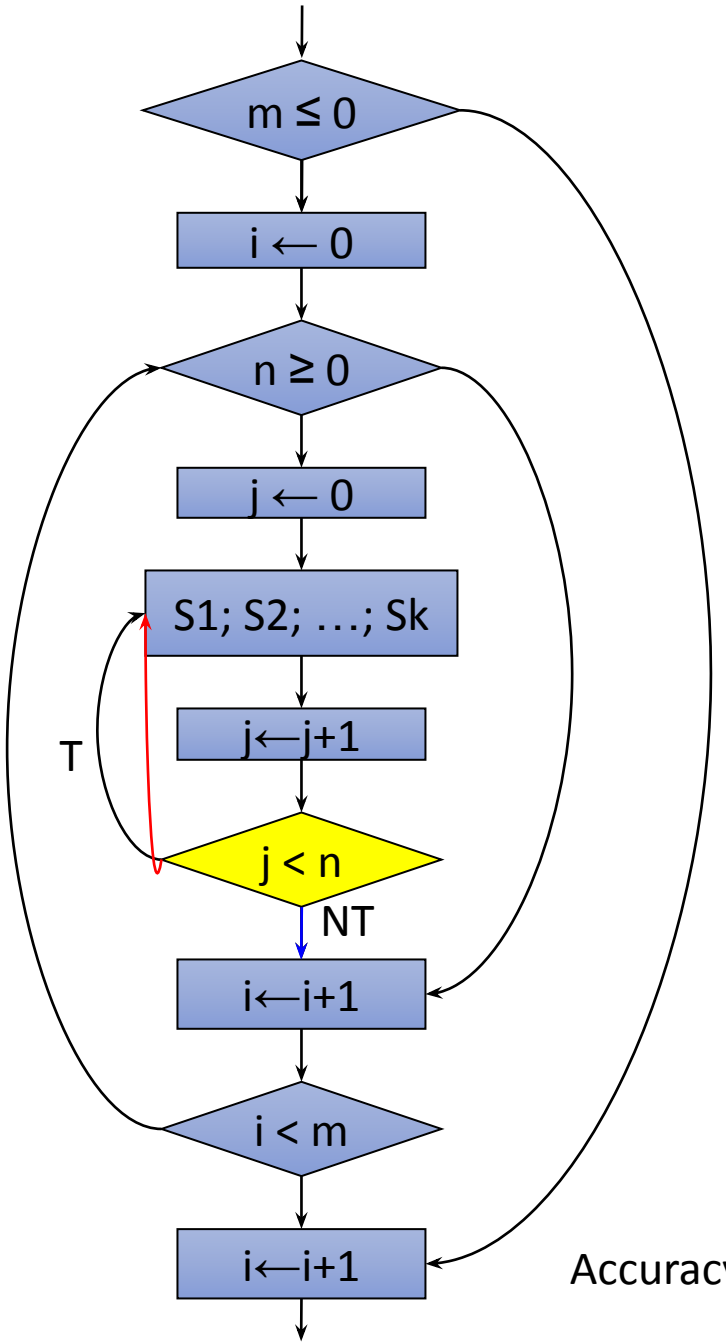



```
for(i=0 ; i < m ; i++)
    for(j=0; j<n ; j++)
        begin S1; S2; …; Sk end;
```

		1-bit		2-bit		
i	**j**	**Pred**	**Outc**	**State**	**Pred**	**Outc**
0	0	NT	T	wNT	NT	T
0	1	T	T	sT	T	T
0	n	T	NT	sT	T	NT
1	0	NT	T	wT	T	T
1	1	T	T	sT	T	T



m + 1 misspredictions




```
for(i=0 ; i < m ; i++)
    for(j=0; j<n ; j++)
        begin S1; S2; …; Sk end;
```

m + 1 misspredictions

		1-bit		2-bit		
i	**j**	**Pred**	**Outc**	**State**	**Pred**	**Outc**
0	0	NT	T	wNT	NT	T
0	1	T	T	sT	T	T
0	n	T	NT	sT	T	NT
1	0	NT	T	wT	T	T
1	1	T	T	sT	T	T

If m= 100 & n=10

1 miss prediction per Iteration

1 + *m* miss predictions for iterations

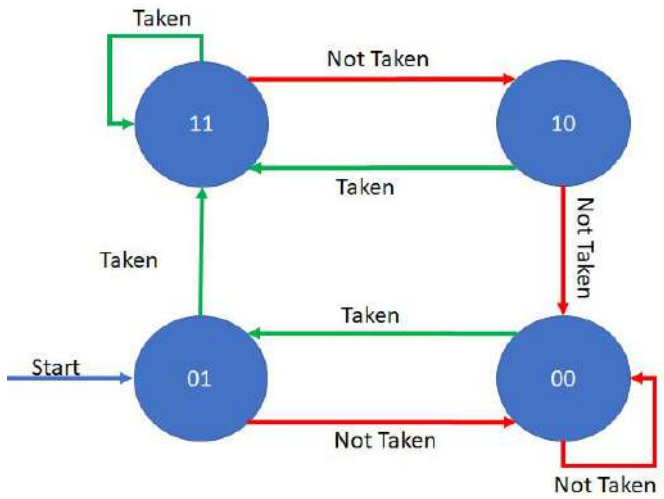
1+100= 101 miss predictions

9 x m Correct Predictions (Single Iteration)

9x100= 900 Correct Predictions for nxm iterations

Accuracy of Correct Predictions = (900/1000)x100≈ 90%

Two -bit Predictor: Initial state: 01



00: Strong Not Taken
01: Weak Not Taken
10: Weak Taken
11: Strong Taken

Consider a program with two branch statement with the following behavior

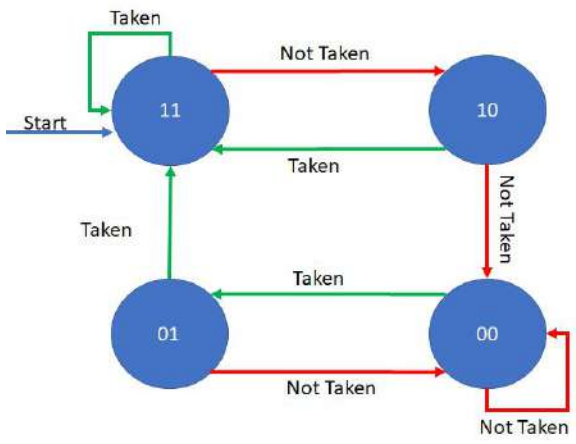
T T NT T NTT T T NT T T T TT NT T T T T NT

How many Miss Prediction?

6 miss predictions

01	11	11	10	11	10	11	11	11	10	11	11	11	11	11	10	11	11	11	11	10
T	T	NT	T	NT	T	T	T	NT	T	T	T	T	T	NT	T	T	T	T	NT	
X	√	X	√	X	√	√	√	X	√	√	√	√	√	X	√	√	√	√	X	

Two -bit Predictor: Initial state: 11



00: Strong Not Taken
01: Weak Not Taken
10: Weak Taken
11: Strong Taken

Consider a program with two branch statement with the following behavior

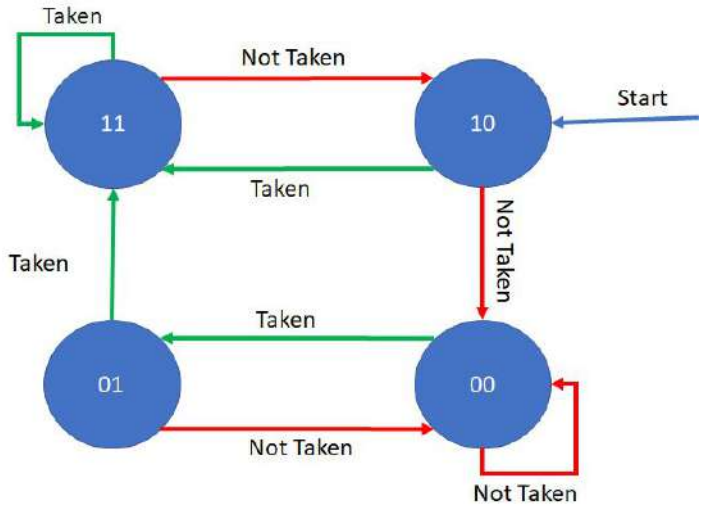
T T NT T NTT T T NT T T T TT NT T T T T NT

5 miss predictions

How many Miss Prediction?

11	11	11	10	11	10	11	11	11	10	11	11	11	11	11	10	11	11	11	11	10
T	T	NT	T	NT	T	T	T	NT	T	T	T	T	T	NT	T	T	T	T	NT	
√	√	X	√	X	√	√	√	X	√	√	√	√	√	X	√	√	√	√	X	

Two -bit Predictor: Initial state: 10



00: Strong Not Taken
01: Weak Not Taken
10: Weak Taken
11: Strong Taken

Consider a program with two branch statement with the following behavior

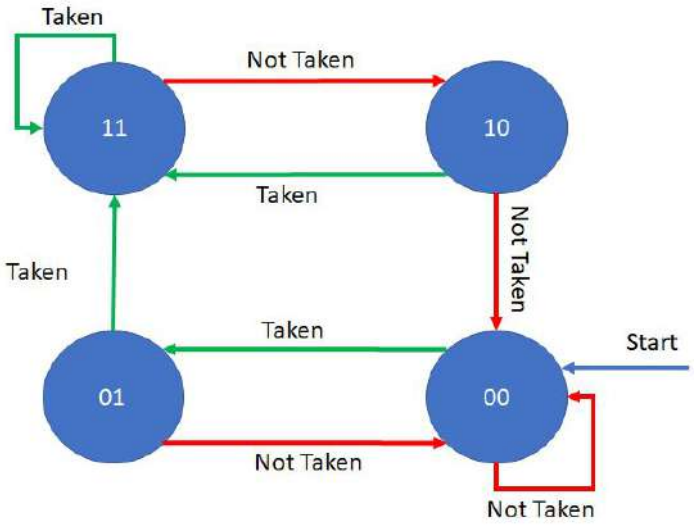
T T NT T NTT T T NT T T T TT NT T T T T NT

5 miss predictions

How many Miss Prediction?

10	11	11	10	11	10	11	11	11	10	11	11	11	11	11	10	11	11	11	11	10
T	T	NT	T	NT	T	T	T	NT	T	T	T	T	T	NT	T	T	T	T	NT	
√	√	X	√	X	√	√	√	X	√	√	√	√	√	X	√	√	√	√	X	

Two -bit Predictor: Initial state 00



00: Strong Not Taken
01: Weak Not Taken
10: Weak Taken
11: Strong Taken

Consider a program with two branch statement with the following behavior

T T NT T NTT T T NT T T T TT NT T T T T NT

7 miss predictions

How many Miss Prediction?

00	01	11	10	11	10	11	11	11	10	11	11	11	11	11	10	11	11	11	11	10
T	T	NT	T	NT	T	T	T	NT	T	T	T	T	T	NT	T	T	T	T	NT	
X	X	X	√	X	√	√	√	X	√	√	√	√	√	X	√	√	√	√	X	

Think About It

Which predictor is best suitable if the following is the likely outcome of branch instruction? Suggest the suitable initial state

T NT T NT T NT T NT T TN

T T T T T T T T T T T T T T T T

NT NT NT NT NT NT NT NT NT

Next Session

Performance Analysis

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135

Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Introduction to Pipeline Processor

Dr. D. C. Kiran
Department of Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

Performance Analysis

Unit 3: **Memory Design**

Unit 4: **Input/Output Device Design**

Unit 5: **Advanced Architecture**

Pipelined Execution

K stage	T1	T2	T3	T4	T5	T6	T7	T8	T9
IF	I1	I2	I3	I4	I5				
ID		I1	I2	I3	I4	I5			
EX			I1	I2	I3	I4	I5		
MB				I1	I2	I3	I4	I5	
WB					I1	I2	I3	I4	I5

k*tc*1

(n-1)*tc

Number of stages= k

Clock Cycle = tc

Number of Instructions = n

- Execution time$_{pipeline}$ = k*tc*1 + (n-1)*tc= $\mathbf{[k+(n-1)]t_c}$

- Execution time$_{unpipeline}$ = $\mathbf{n^* t_p}$ =n*k* t_c

Speedup vs # of Stages in Pipeline Processor

- $Speedup(S) = \dfrac{Execution\ Time\ on\ Unpipelined}{Execution\ Time\ on\ Pipelined}$

$$\frac{n\mathrm{t_p}}{(k+n-1)\mathrm{t}c} \rightarrow \frac{n\mathrm{t_p}}{(k-1+n)\mathrm{t}c}$$

If n is too big or as number of instructions increases n>k-1 will tend to n

$$\text{Speedup (S)} = \frac{n\mathrm{t_p}}{n\mathrm{t}c}$$

$$\text{Speedup (S)} = \frac{\mathrm{t_p}}{\mathrm{t}c}$$

$$\text{Speedup (S)} = \frac{k*\mathrm{t}c}{\mathrm{t}c}$$

$$\text{Speedup (S)} = \mathrm{k}$$

Theoretical Claim

$$\text{Time taken}_{\text{pipeline}} = \frac{Time\ taken\ on\ unpipelined}{No.of\ pipeline\ stages}$$

Pipelining

- 5-stage Pipeline
  - Fetch
  - Decode
  - Execute
  - Mem
  - WB
- n instructions, k-stages, t_c (cycle time)
- Execution $time_{pipeline} = [k+(n-1)]t_c$
- $Speedup(S) = \frac{Execution\ Time\ on\ Unpipelined}{Execution\ Time\ on\ Pipelined}$
- $Efficiency(\mu) = \frac{Speedup}{k}$
  - n << k
  - n == k
  - n >> k

$$\text{Time taken}_{pipeline} = \frac{Time\ taken\ on\ unpipelined}{No.of\ pipeline\ stages}$$

- n instructions, k-stages, t_c (cycle time)
- Execution $time_{pipeline} = [k+(n-1)]t_c$
- Execution $time_{unpipeline} = nkt_c$
- $Speedup(S) = \frac{Execution\ Time\ on\ Unpipelined}{Execution\ Time\ on\ Pipelined} = \frac{nkt}{(k+n-1)t} = \frac{nk}{k+n-1}$
- $Efficiency(\mu) = \frac{Speedup}{k}$
  - n << k ; S = n; µ < 1
  - n == k; S = n/2; µ = 0.5
  - **n >> k; S = k; µ = 1** ✓

$$\text{Time taken}_{pipeline} = \frac{Time\ taken\ on\ unpipelined}{No.of\ pipeline\ stages}$$

Performance of Pipelines with Stalls

A stall causes the pipeline performance to degrade the ideal performance.

$$\text{Speedup from pipelining} = \frac{\text{Average instruction time unpipelined}}{\text{Average instruction time pipelined}}$$

$$= \frac{CPI_{unpipelined} * \text{Clock Cycle Time}_{unpipelined}}{CPI_{pipelined} * \text{Clock Cycle Time}_{pipelined}}$$

$$= \frac{CPI_{unpipelined}}{1 + \text{Pipeline stall cycles per instruction}}$$

Note 1: Ignoring the cycle time overhead of pipelining and assume the stages are all perfectly balanced, then the cycle time of the two machines are equal

Note 2: Ideal CPI of pipeline processor is almost always =1

$CPI_{pipelined}$ = Ideal CPI + Pipeline stall clock cycles per instruction

= 1 + Pipeline stall clock cycles per instruction

Performance of Pipelines with Stalls

If all instructions take the same number of cycles, which must also equal the number of pipeline stages (the depth of the pipeline) then unpipelined CPI is equal to the depth of the pipeline

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall cycles per instruction}}$$

If there are no pipeline stalls, this leads to the intuitive result that pipelining can improve performance by the depth of pipeline.

Exercise 1

Consider an instruction pipeline with four stages with the stage delay 8 nsec respectively. Assume that the delay of an inter-stage register stage of the pipeline is Nill. What is the approximate speedup of the pipeline in the steady state under ideal conditions as compared to the corresponding non-pipelined implementation when 100 instructions are executed?

Solution:

Time taken to execute N instructions in non-pipelined implementation will be KxTcxN = 8x4x100=3200.

Time taken for the pipelined implementation = (K+N-1)*Tc
= (4+100-1)*8
= 824

Speedup = 3200/ 824 = 3.8

Exercise 2

Consider an instruction pipeline with four stages with the stage delays 5 nsec, 6 nsec, 11 nsec, and 8 nsec respectively. The delay of an inter-stage register stage of the pipeline is 2. What is the approximate speedup of the pipeline in the steady state under ideal conditions as compared to the corresponding non-pipelined implementation when 100 instructions are executed?

Solution:

Time taken to execute N instructions in non-pipelined implementation will be (5 + 6 + 11 + 8)N = 30N= 30x100=3000.

Clock period for pipelined implementation = max{5,6,11,8} = 11.

Time taken for the pipelined implementation = (K+N-1)*Tc
= (4+N-1)*11
= (4 + 99)*11 = 1133

Speedup = 3000/ 1133 = 2.6

Exercise 3

Consider an instruction pipeline with four stages with the stage delays 5 nsec, 6 nsec, 11 nsec, and 8 nsec respectively. The delay of an inter-stage register stage of the pipeline is 2. What is the approximate speedup of the pipeline in the steady state under ideal conditions as compared to the corresponding non-pipelined implementation when 100 instructions are executed?

Solution:

Time taken to execute N instructions in non-pipelined implementation will be (5 + 6 + 11 + 8)N = 30N= 30x100=3000.

Clock period for pipelined implementation = max{5,6,11,8} = 11.
Clock period for pipelined implementation with register overhead= 11+2=13

Time taken for the pipelined implementation = (K+N-1)*Tc
= (4+N-1)*13
= (4 + 99)*11 = 1339

Speedup = 3000/ 1 = 2.2

Exercise 4

Consider an instruction pipeline for the MIPS32 processor where data references constitute 42% of the instructions, and the ideal CPI ignoring memory structural hazards is 1.25. How much faster is the ideal machine without the memory structural hazard versus the machine with the hazard?

Solution:

Speedup = (Ideal CPI x Pipeline Depth) / (Ideal CPI + Stall cycles per instr)

So, $Speedup_{\_ideal} = 1.25 \times K / (1.25 + 0) = K$

$Speedup_{\_real} = 1.25 \times K / (1.25 + 0.42 \times 1)$
$= 1.25 \times K / 1.67$

Required answer $= K / (1.25 \times K / 1.67)$
$= 1/(1.25/1.67)$
$= 1.67 / 1.25$
$= 1.34$

Exercise 6 With Branch

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{CPI-Penalty}}$$

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch Frequency} * \text{Branch Penalty}}$$

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \%\text{branch} * [(\%_T * \text{penalty}_T) + (\%_{NT} * \text{penalty}_{NT})]}$$

Performance with Branch Hazards

- CPI-penalty = %branch * [($\%_T$ * $penalty_T$) + ($\%_{NT}$ * $penalty_{NT}$)]
- simple branch statistics
  - 14% “branches”
  - 65% of branches are “taken”

scheme	$penalty_T$	$penalty_{NT}$	CPI penalty
stall	2	2	0.28
fast branch	1	1	0.14
delayed branch	1.5	1.5	0.21
not-taken	2	0	0.18
taken	0	2	0.10

Performance with Branch Hazards

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Branch Frequency} * \text{Branch Penalty}}$$

Simple branch statistics

- 14% “branches”
  - 65% of branches are “taken”

K=5

Scheduling Scheme	Branch Penalty	CPI	Speed-Up
Stalling	3	1.42	3.5
Predict Taken	1	1.14	4.4
Predict Not Taken	1	1.09	4.5
Delayed Branch	0.5	1.07	4.6

Consider the MIPS32 pipeline with ideal CPI of 1. Assume that 30% of all instructions executed are branch, out of which 80% are taken branches. The pipeline speedup for predict taken and delayed branch approaches to reduce branch penalties will be:

Solution:

For predict taken, branch penalty = 1
Speedup = 5 / (1 + 0.30 x 1) = 3.85

For delayed branch, branch penalty = 0.5
Speedup = 5 / (1 + 0.30 x 0.5) = 4.35

https://questions.examside.com/past-years/gate/question/comparing-the-time-t1-taken-for-a-single-instruction-on-a-pi-gate-cse-2000-marks-1-kbnd2d29g7jvmys0.htm

https://questions.examside.com/past-years/gate/question/an-instruction-pipeline-consists-of-4-stages-fetch-f-decode-gate-cse-1999-marks-5-v5wbawwui7szocue.htm

https://questions.examside.com/past-years/gate/question/the-performance-of-a-pipelined-processor-suffers-if-gate-cse-2002-marks-2-qmgeshbxcjnk5kzt.htm

https://gateoverflow.in/103937/branch-prediction

https://www.gatevidyalay.com/pipelining-practice-problems/

https://questions.examside.com/past-years/gate/question/comparing-the-time-t1-taken-for-a-single-instruction-on-a-pi-gate-cse-2000-marks-1-kbnd2d29g7jvmys0.htm

https://questions.examside.com/past-years/gate/question/an-instruction-pipeline-consists-of-4-stages-fetch-f-decode-gate-cse-1999-marks-5-v5wbawwui7szocue.htm

https://questions.examside.com/past-years/gate/question/the-performance-of-a-pipelined-processor-suffers-if-gate-cse-2002-marks-2-qmgeshbxcjnk5kzt.htm

https://gateoverflow.in/103937/branch-prediction

Performance Analysis 2
Summary

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135

Execution Time

CPU_{Time} = = Instruction Count (IC) X Clock Cycle X CPI

Reducing any of the 3 factors will lead to improve performance or Reduce Execution time is

- CPI: Cycles per instruction
- Clock Cycle
- Instruction count

Consider an unpipelined processor. Assume that it has a 1ns clock cycle and that it uses 4 cycles for ALU operations and branches and 5 cycles for memory operations. Assume that the relative frequencies of these operations are 40%, 20%, and 40%, respectively. Suppose that due to clock skew and setup, pipelining the processor adds 0.2 ns of overhead to the clock. Ignoring any latency impact, how much speedup in the instruction execution rate will we gain from a pipeline?

Solution:

Average instruction execution time$_{unpipeline}$ = # Clock cycle × Average CPI

= 1 ns X [(40% + 20%) x 4 + 40% x 5] = 4.4 ns

Average instruction execution time$_{pipeline}$ = Clock cycle time + Pipeline overhead

= 1 + 0.2 = 1.2ns

SPEEDUP from pipeline = $\frac{Average\ instruction\ execution\ time\ umpipelined}{Average\ instruction\ execution\ time\ pipelined} = \frac{4.4\ ns}{1.2\ ns} = 3.7\ times$

Microprocessor & Computer Architecture (µpCA)

UE19CS252

Dr. D. C. Kiran

Department of
Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Unit2 : Pipeline Processor

Dr. D. C. Kiran
Department of Computer Science and Engineering

Microprocessor & Computer Architecture (µpCA)

Syllabus

~~Unit 1: Basic Processor Architecture and Design~~

Unit 2: Pipelined Processor and Design

Performance Analysis

Unit 3: **Memory Design**

Unit 4: **Input/Output Device Design**

Unit 5: **Advanced Architecture**

Microprocessor & Computer Architecture (µpCA)

Exercise: Summarize Pipeline Execution

For all following questions we assume that:

a) Pipeline contains 5 stages: IF, ID, EX, M and W)
b) Each stage requires one clock cycle;

```
// ADD TWO INTEGER ARRAYS
LDR R4, #400
L1: LDR R1, [R4]          ; Load first operand
    LDR R2, [R4,#400]     ; Load second operand
    ADD R3, R1, R2        ; Add operands
    STR R3, [R4]          ; Store result
    SUB R4, R4, #4        ; Calculate address of next element
    BNEZ R4, L1           ; Loop if (R4) != 0
```

Calculate how many clock cycles will take execution of this segment on the regular (nonpipelined) architecture. Show calculations:

Exercise: Summarize Pipeline Execution

Calculate how many clock cycles will take execution of this segment on the regular (nonpipelined) architecture. Show calculations:

```
// ADD TWO INTEGER ARRAYS
      LDR R4, # 400
L1: LDR R1, [R4]
      LDR R2, [R4,#400]
      ADD R3, R1, R2
      STR R3, [R4]
      SUB R4, R4, #4
      BNEZ R4, L1
```

Solution

Number of cycles = [Initial instruction + (Number of instructions in the loop L1) x number of loop cycles] x number of clock cycles=

= [1 + (6) x 400/4] x 5 c.c. = 3005 c.c.

Microprocessor & Computer Architecture (µpCA)

Exercise: Summarize Pipeline Execution

Calculate how many clock cycles will take execution of this segment on the simple pipeline without forwarding or bypassing when result of the branch instruction (new PC content) is available after WB stage. Show timing of one loop cycle in Table

Solution:

Instruction	Clock cycle number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L1: LDR R1, [R4]	IF	ID	Ex	M	W											
LDR R2, [R4,#400]		IF	ID	Ex	M	W										
ADD R3, R1, R2			IF	ID	*	*	Ex	M	W							
STR R3, [R4]				IF	*	*	ID	Ex	*	M	W					
SUB R4, R4, #4							IF	ID	*	Ex	M	W				
BNEZ R4, L1								IF	*	ID	*	*	Ex	M	W	

Microprocessor & Computer Architecture (µpCA)

Exercise: Summarize Pipeline Execution

Comments

Instruction	Clock cycle number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L1: LDR R1, [R4]	IF	ID	Ex	M	W											
LDR R2, [R4,#400]		IF	ID	Ex	M	W										
ADD R3, R1, R2			IF	ID	*	*	Ex	M	W							
STR R3, [R4]				IF	*	*	ID	Ex	*	M	W					
SUB R4, R4, #4							IF	ID	*	Ex	M	W				
BNEZ R4, L1								IF	*	ID	*	*	Ex	M	W	

1. Two stall cycles (c.c. # 5 and 6) are caused by the delay of data in the register R2 for the ADD.
2. Same stall cycles in ID stage for the SW instruction are because ID stage circuits are busy for ADD and becoming available only on 7-th c.c.
3. SUB can start only on 7-th c.c. because IF stage is busy with STR instruction.
4. One c.c. stall in the pipeline happens because the content of R3 (for STR) is not ready. However, “Ex” stage can be executed for STR instruction. This becomes possible because during the “Ex” stage the address in memory is calculated (only for Load or Store instructions).
5. Two stall cycles (c.c. # 11 and 12) in BNEZ are coming from the delay of updating the R4. New content of R4 becomes available only after 12 c.c. Thus, the content of PC is updated on W-stage of BNEZ (after15 c.c.).

Microprocessor & Computer Architecture (µpCA)

Exercise: Summarize Pipeline Execution

Comments

Instruction	Clock cycle number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L1: LDR R1, [R4]	IF	ID	Ex	M	W											
LDR R2, [R4,#400]		IF	ID	Ex	M	W										
ADD R3, R1, R2			IF	ID	*	*	Ex	M	W							
STR R3, [R4]				IF	*	*	ID	Ex	*	M	W					
SUB R4, R4, #4							IF	ID	*	Ex	M	W				
BNEZ R4, L1								IF	*	ID	*	*	Ex	M	W	

Number of cycles in the loop = 15 c.c.

Number of clock cycles for segment execution on pipelined processor =
= 1 c.c. (IF stage of the initial instruction) + (Number of clock cycles in the loop L1) x Number of loop cycles
= 1 + 15 x 400/4 = 1501 c.c.

Speedup = 3005 c.c. / 1501 = 2 times

Microprocessor & Computer Architecture (µpCA)

Exercise: Summarize Pipeline Execution

Calculate how many clock cycles will take execution of this segment on the simple pipeline with normal forwarding and bypassing when result of branch instruction (new PC content) is available after completion of the ID stage. Show timing of one loop cycle in Table.

Instruction	Clock cycle number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L1: LDR R1, [R4]	IF	ID	Ex	M	W											
LDR R2, [R4,#400]		IF	ID	Ex	M	W										
ADD R3, R1, R2			IF	ID	*	Ex	M	W								
STR R3, [R4]				IF	*	ID	Ex	M	W							
SUB R4, R4, #4						IF	ID	Ex	M	W						
BNEZ R4, L1							IF	ID	Ex	M	W					
L1: LDR R1, [R4]								*	IF	ID	Ex	M	W			

Microprocessor & Computer Architecture (µpCA)

Exercise: Summarize Pipeline Execution

Comments:

Instruction	Clock cycle number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L1: LDR R1, [R4]	IF	ID	Ex	M	W											
LDR R2, [R4,#400]		IF	ID	Ex	M	W										
ADD R3, R1, R2			IF	ID	*	Ex	M	W								
STR R3, [R4]				IF	*	ID	Ex	M	W							
SUB R4, R4, #4						IF	ID	Ex	M	W						
BNEZ R4, L1							IF	ID	Ex	M	W					
L1: LDR R1, [R4]								*	IF	ID	Ex	M	W			

1. Data (R2) for the ADD is ready after “M” stage of the LDR R2. During the “WB” stage the requested operand will be written to the R2 and operation register (e.g. Reg. A) of the ALU.
2. ID stage for the SW is delayed because it is busy with ADD.
3. BNEZ can initiate IF stage of the LDR R1, [R4] because new PC-content is ready after 8 c.c.

Number of cycles in the loop = 8 c.c.

Speedup =3005 c.c. / (1 c.c. + 400/4 x 8 c.c.) = 3005 / 801 = 3.75 times

Microprocessor & Computer Architecture (µpCA)

Exercise: Summarize Pipeline Execution

Schedule the segment instructions including branch-delay slot to get minimum processing time assuming that pipeline has normal forwarding and bypassing hardware. It is possible to reorder instructions and change position of loop label (L1) but not name of registers or op-code modification. Show scheduled segment, position of L1 and pipeline timing diagram in Table and calculate number of clock cycles needed to execute this task segment.

Instruction	Clock cycle number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L1: LDR R1, [R4]	IF	ID	Ex	M	W											
LDR R2, [R4,#400]		IF	ID	Ex	M	W										
SUB R4, R4, #4			IF	ID	Ex	M	W									
ADD R3, R1, R2				IF	ID	Ex	M	W								
BNEZ R4, L1					IF	ID	Ex	M	W							
STR R3, [R4, #4]						IF	ID	Ex	M	W						
L1: LDR R1, [R4]							IF	ID	Ex	M	W					

Microprocessor & Computer Architecture (µpCA)

Exercise: Summarize Pipeline Execution

Instruction	Clock cycle number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L1: LDR R1, [R4]	IF	ID	Ex	M	W											
LDR R2, [R4,#400]		IF	ID	Ex	M	W										
ADD R3, R1, R2			IF	ID	*	Ex	M	W								
STR R3, [R4]				IF	*	ID	Ex	M	W							
SUB R4, R4, #4						IF	ID	Ex	M	W						
BNEZ R4, L1							IF	ID	Ex	M	W					
L1: LDR R1, [R4]								*	IF	ID	Ex	M	W			

Vs

Instruction	Clock cycle number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L1: LDR R1, [R4]	IF	ID	Ex	M	W											
LDR R2, [R4,#400]		IF	ID	Ex	M	W										
SUB R4, R4, #4			IF	ID	Ex	M	W									
ADD R3, R1, R2				IF	ID	Ex	M	W								
BNEZ R4, L1					IF	ID	Ex	M	W							
STR R3, [R4, #4]						IF	ID	Ex	M	W						
L1: LDR R1, [R4]							IF	ID	Ex	M	W					

Exercise: Summarize Pipeline Execution

Instruction	Clock cycle number															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
L1: LDR R1, [R4]	IF	ID	Ex	M	W											
LDR R2, [R4,#400]		IF	ID	Ex	M	W										
SUB R4, R4, #4			IF	ID	Ex	M	W									
ADD R3, R1, R2				IF	ID	Ex	M	W								
BNEZ R4, L1					IF	ID	Ex	M	W							
STR R3, [R4, #4]						IF	ID	Ex	M	W						
L1: LDR R1, [R4]							IF	ID	Ex	M	W					

The maximum speedup comparing with non-pipelined processor is = 3005 / (1+ 6 x 100) = 5 times It means that all stages of 5-stage pipeline are always busy (no stalls) during the task segment execution

Next Session

Unit 3
Memory

THANK YOU

Dr. D. C. Kiran
Department of Computer Science and Engineering

dckiran@pes.edu

9829935135