Pipelining

Computer Architecture
ECE 6913

Brandon Reagen

NYU TANDON SCHOOL OF ENGINEERING

What we'll cover today

1) Review: ISAs and single-cycle MIPS machine
2) Pipelining: improving performance by overlapping execution
3) Hazards and optimizations

We have TAs!!

As promised.. For weeks ☺

Yichen Wang: yw4604@nyu.edu

Jiazhen Han: jh6419@nyu.edu

Lab and HW experts. Post questions on Piazza and they'll get back to you.

HW 1 solutions also posted.

Did anything exciting happen this week?

NVIDIA = GPU Giant. Why?

ARM = Mobile Giant.
What does that even mean?

FB: >2B MAUs. >90% ad rev from mobile.
"A53 represents more than 48%"
"A7 represents more than 15%"

2 old ARM cores (e.g., ARM gives A53s to universities) account for >63% of the mobile market*

Paper: Machine Learning at Facebook: Understanding Inference at the Edge

Trends in Computer Architecture (From GaTech~2008)

50s to 60s: Computer Architecture ~ Computer Arithmetic

70s to mid 80s: Instruction Set Design, especially ISA appropriate for compilers

90s: Speculation: Predict this, predict that; memory system; I/O system; Multiprocessors; Networks

2000s: Power efficiency , Communication, On-die Interconnection Network, Multi-this, Multi-that. (We are here)

2015 and beyond: Thousand-core processors, Self adapting systems? Self organizing structures? DNA Systems/Quantum Computing?

Today

2015 and beyond: Thousand-core processors, Self adapting systems? Self organizing structures? DNA Systems/Quantum Computing? Not so much

1) Hardware for ML (e.g., nvidia, TPU, cerebras, etc.)
2) HW-SW co-design: understand how code is running, find bottleneck, fix it (in both hw and sw)
3) SoC design and Accelerator-centric computing!
4) New ISAs? New killer-workloads making us go back in time
5) Some quantum/dna stuff,

Hard job. Need to be expert in multiple fields

On getting active

Just go for it..

Read papers: Top venues: ISCA, MICRO, HPCA, ASPLOS

Come up with ideas and ask NYU profs what they think

Instruction Set Architecture (ISA)

Contract between the programmer (SW) and machine (HW)

- ISA defines instructions and their functionality
  - Visible state of the machine, what the programmer/compiler can see
  - Changes to state as instructions are processed
- Underlying machine implementation faithfully executes instructions
- Programmers/compiler express functions as series of instructions

Specification: Instructions and State

- Exactly how each instruction changes the machine's state
- Instruction and memory representation

What does an ISA NOT do?

- How instructions are implemented
- Instruction performance or energy usage
- These are left to the microarchitecture

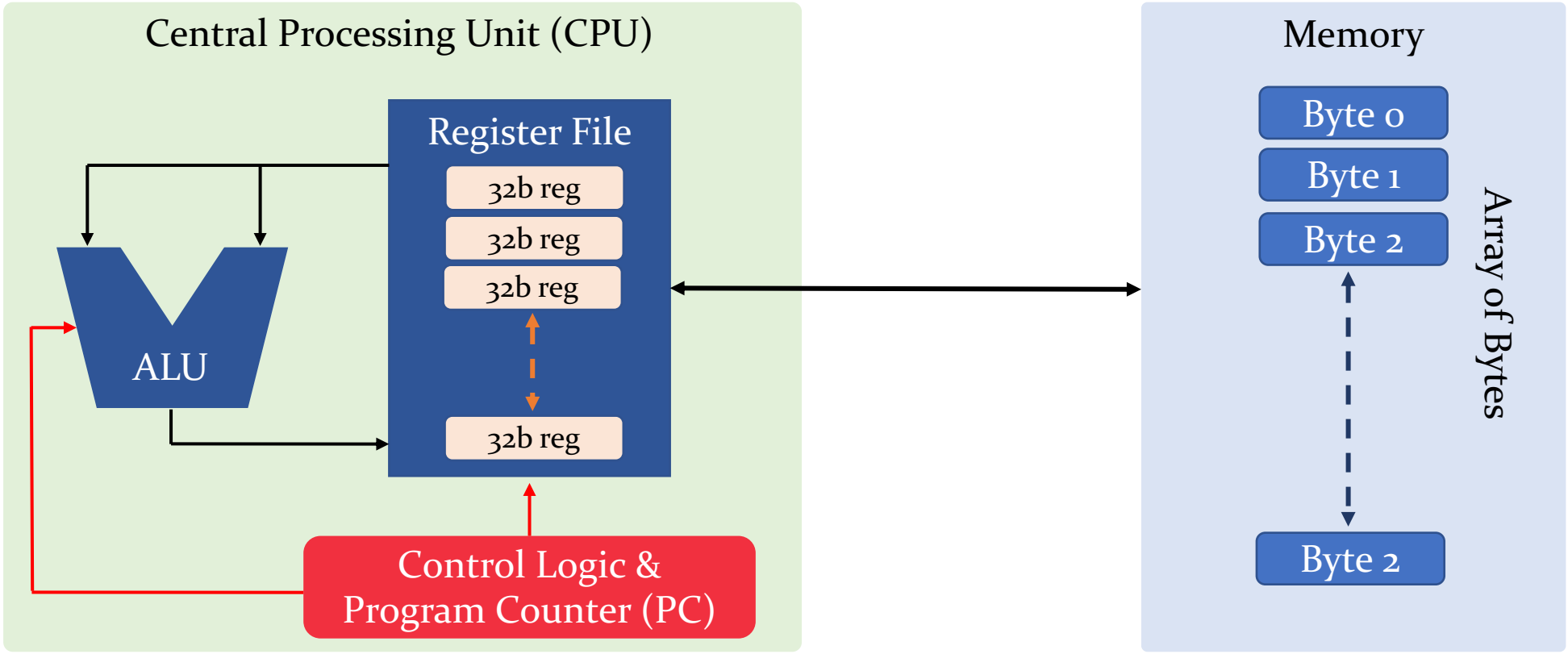
MIPS Summary

32b RISC architecture
- Fixed instruction length
- 32b registers and memory addresses
- Byte addressable memory.. How much?

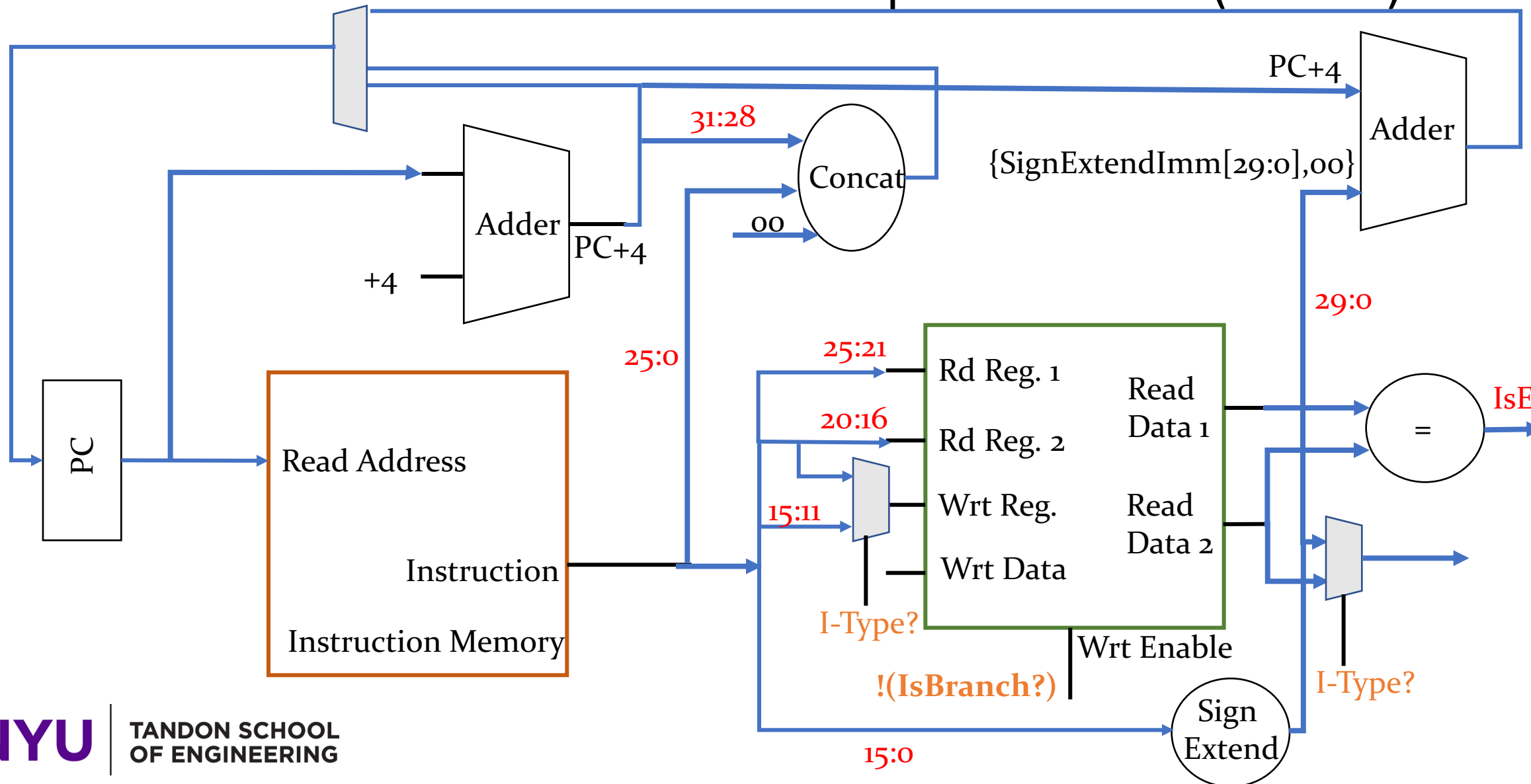
Load-store architecture
- All compute use values from registers
- Only special instructions access memory

32 general purpose registers
- We will assume the simplified calling convention
- Program counter for finding next instruction
- Hi/lo registers for multiplication

Architectural (ISA) view of a processor

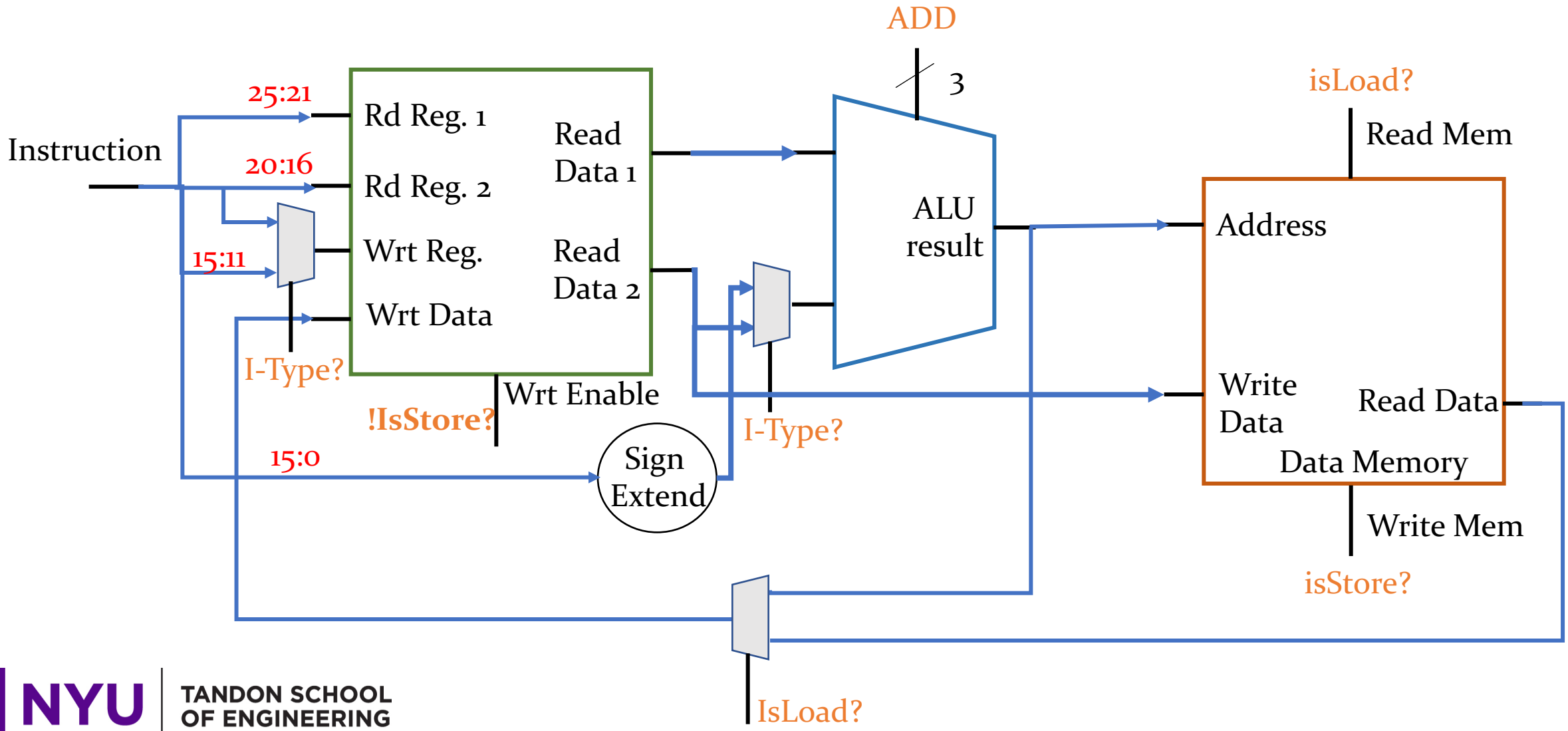


Microarchitectural view of a processor (front)

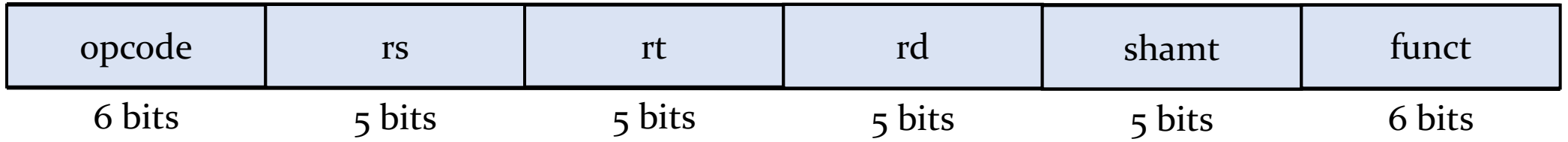


Microarchitectural view of a processor (back)

Instruction Fetch Part Not Shown



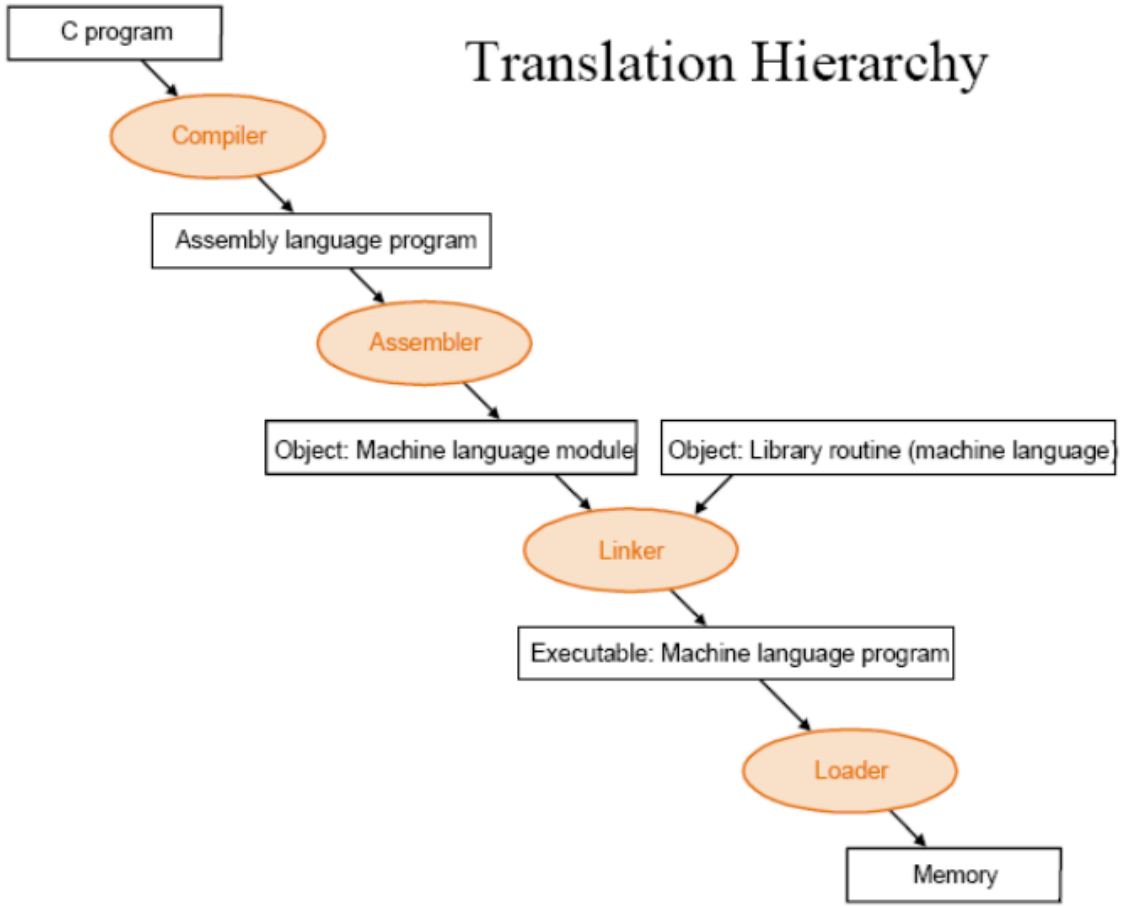
Question from last time: ISA design: Shamt



Shamt says how many bits to shift (left or right)
In MIPS, it's part of the ISA.

Why is this a clever design decision?
How else could you do it?
What are the performance tradeoffs?

Question from last time: Pseudoinstruction set



Move $r1, $r2 is actually or $r1, $r2, $r0
What's $ro?

Why do we have these?

What makes a good ISA?

Simplicity favors regularity
- Instruction size, instruction format, data format
- Makes hardware implementation cleaner

Smaller is faster
- Fewer bit to move, write, and read per instruction
- Register file is faster than memory

Make the common case fast
- Constants tend to be small, immediate field optimized for this

Good design demands compromise
- Special formats for important exception
- E.g., jumping far away (as we saw)

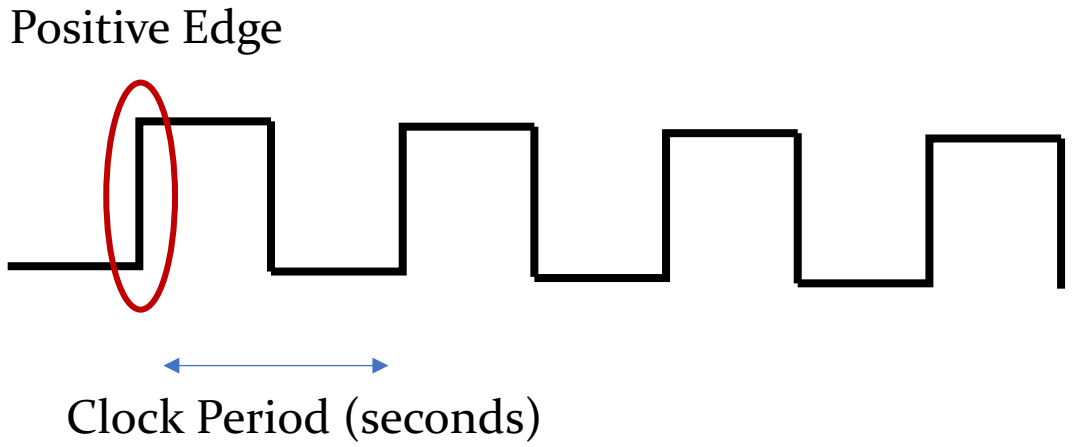
Synchronous Implementation

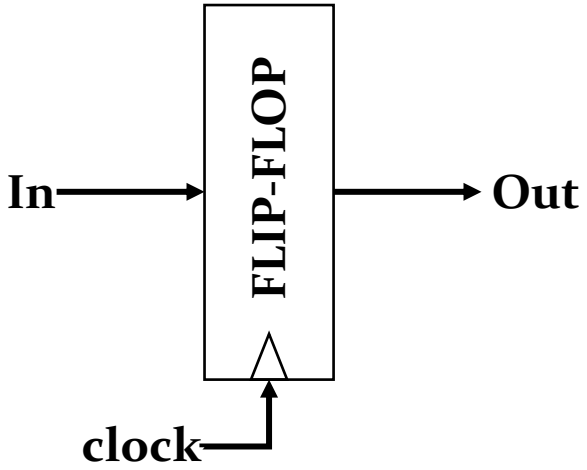
Most digital circuits make use of a periodic “clock”

Synchronous elements enable us to store state

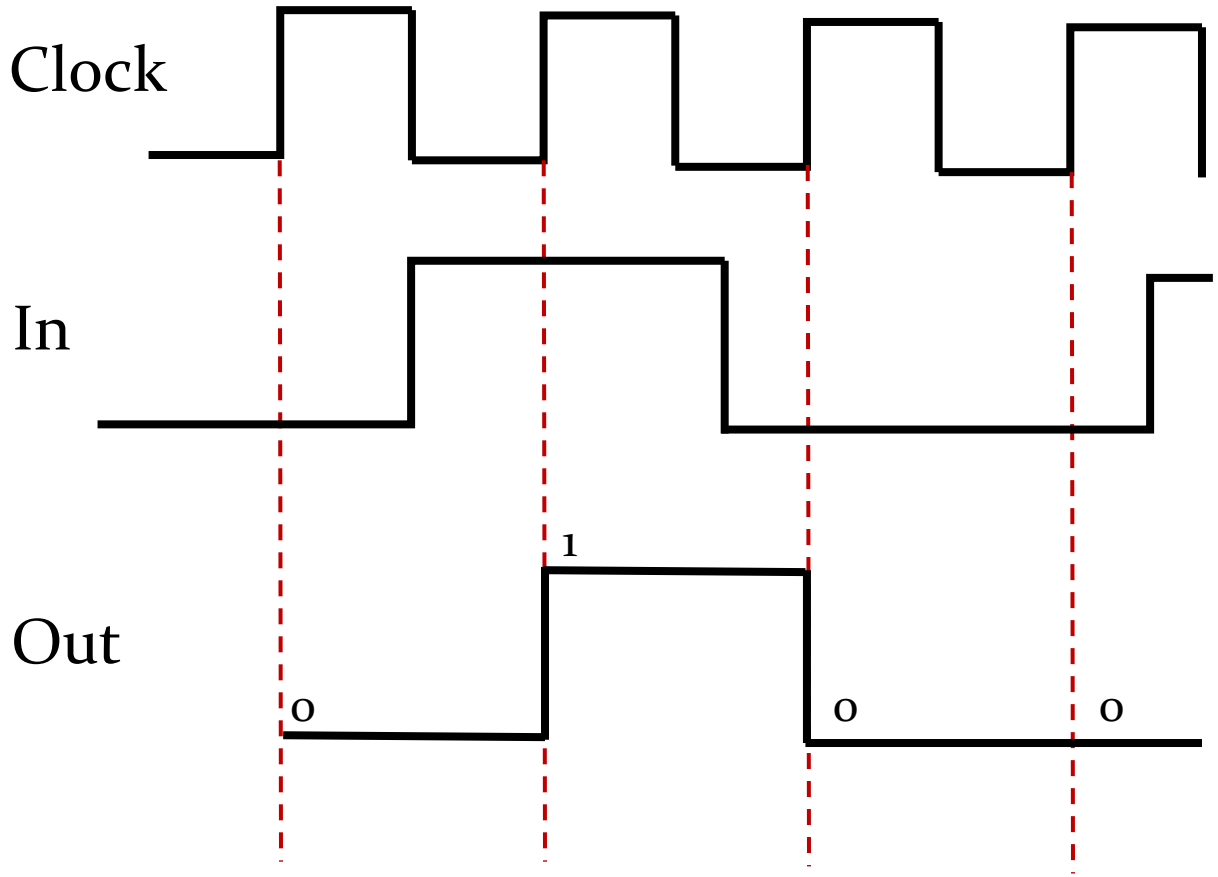
Flip-flop: basic component used in synchronous designs

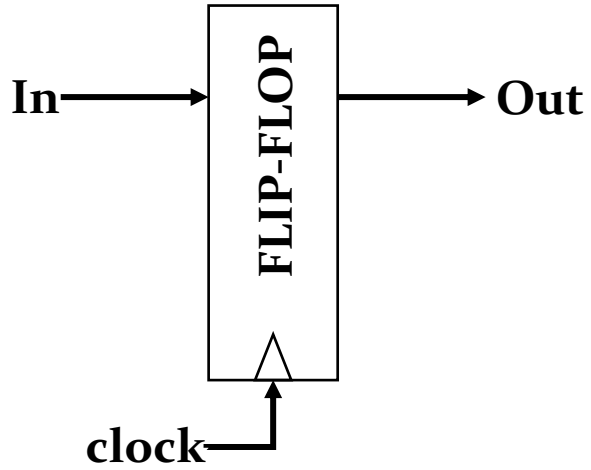
- Looks at In at every positive clock edge and sets Out = In
- Holds the value of Out steady till next positive edge



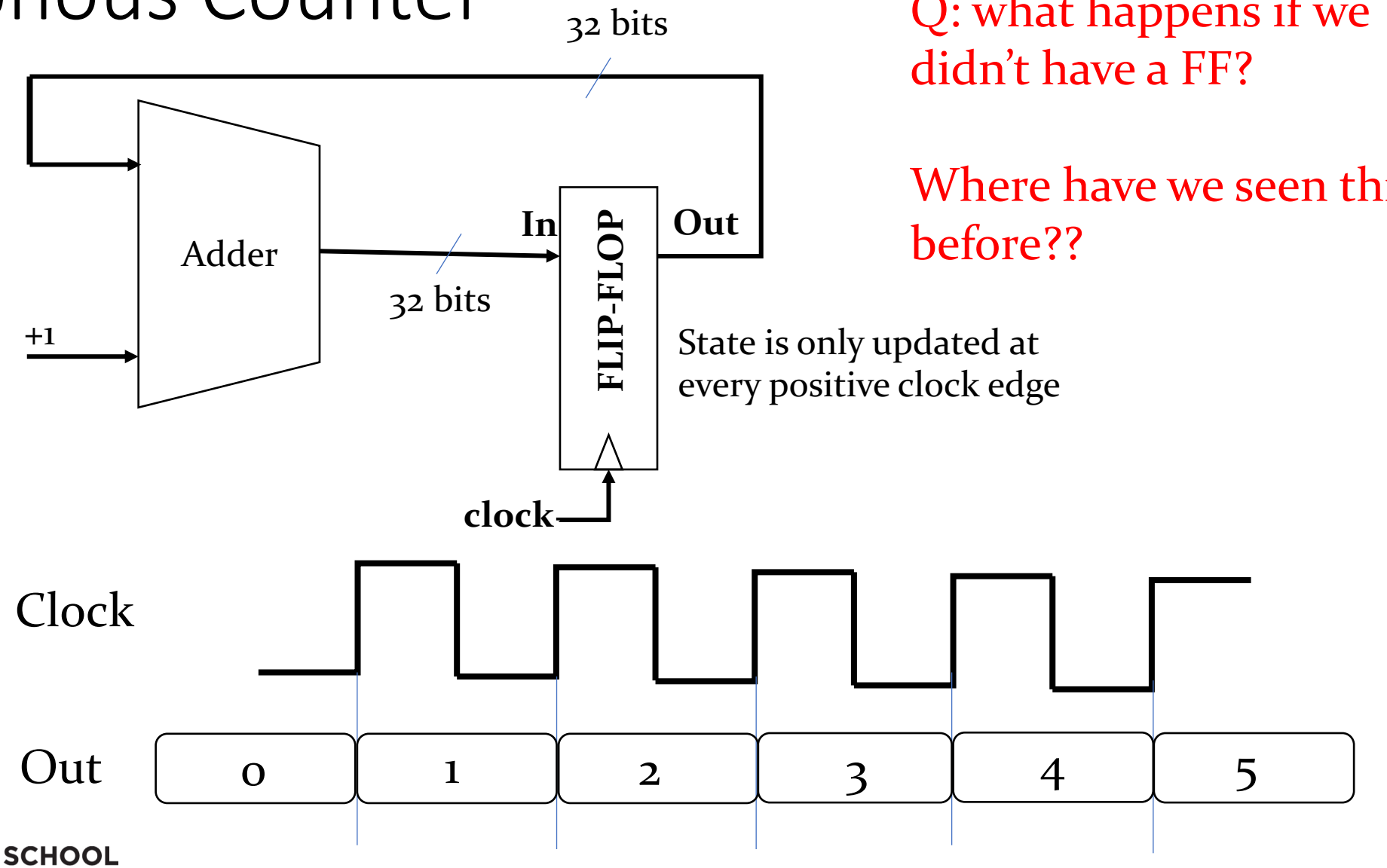


Flip-flop Operation





Synchronous Counter



State is only updated at every positive clock edge

Q: what happens if we didn't have a FF?

Where have we seen this before??

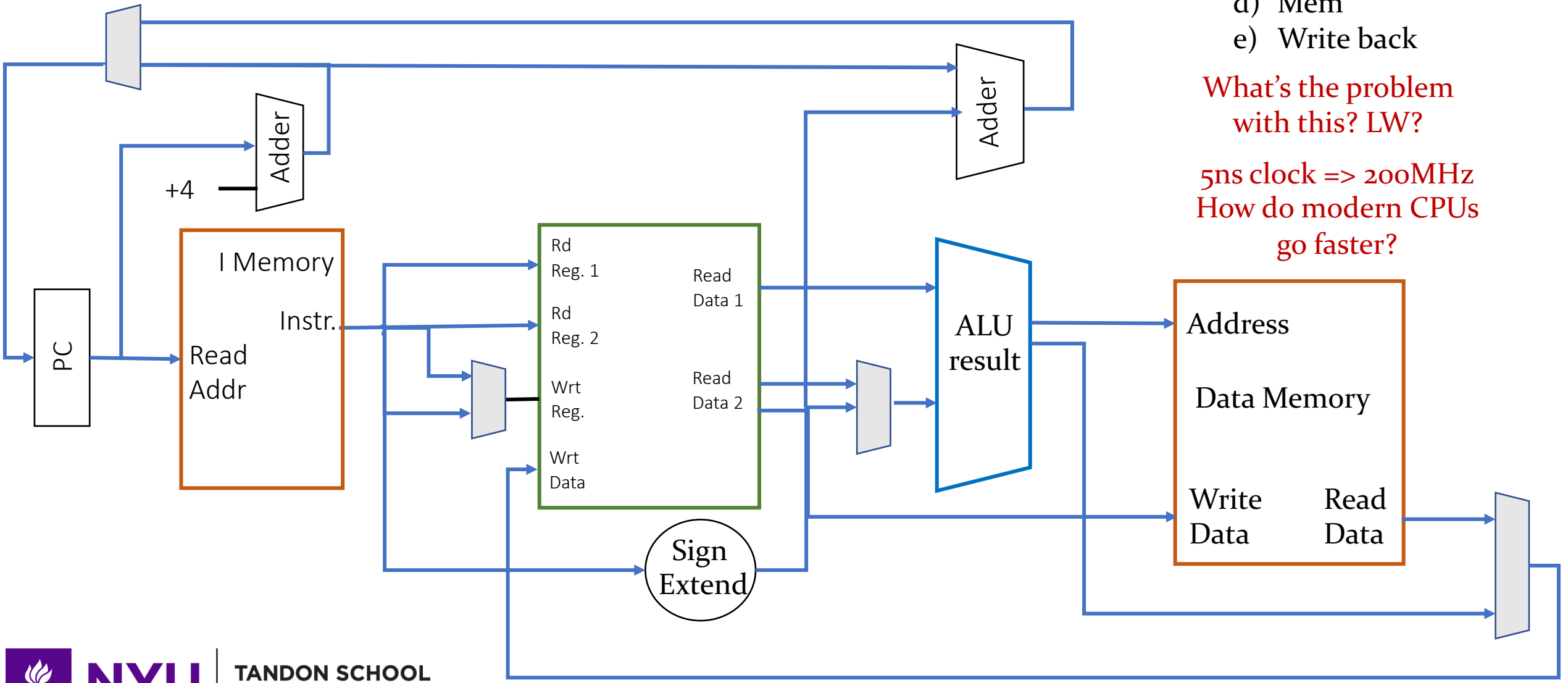
From last time (simplified)

Each takes 1ns..

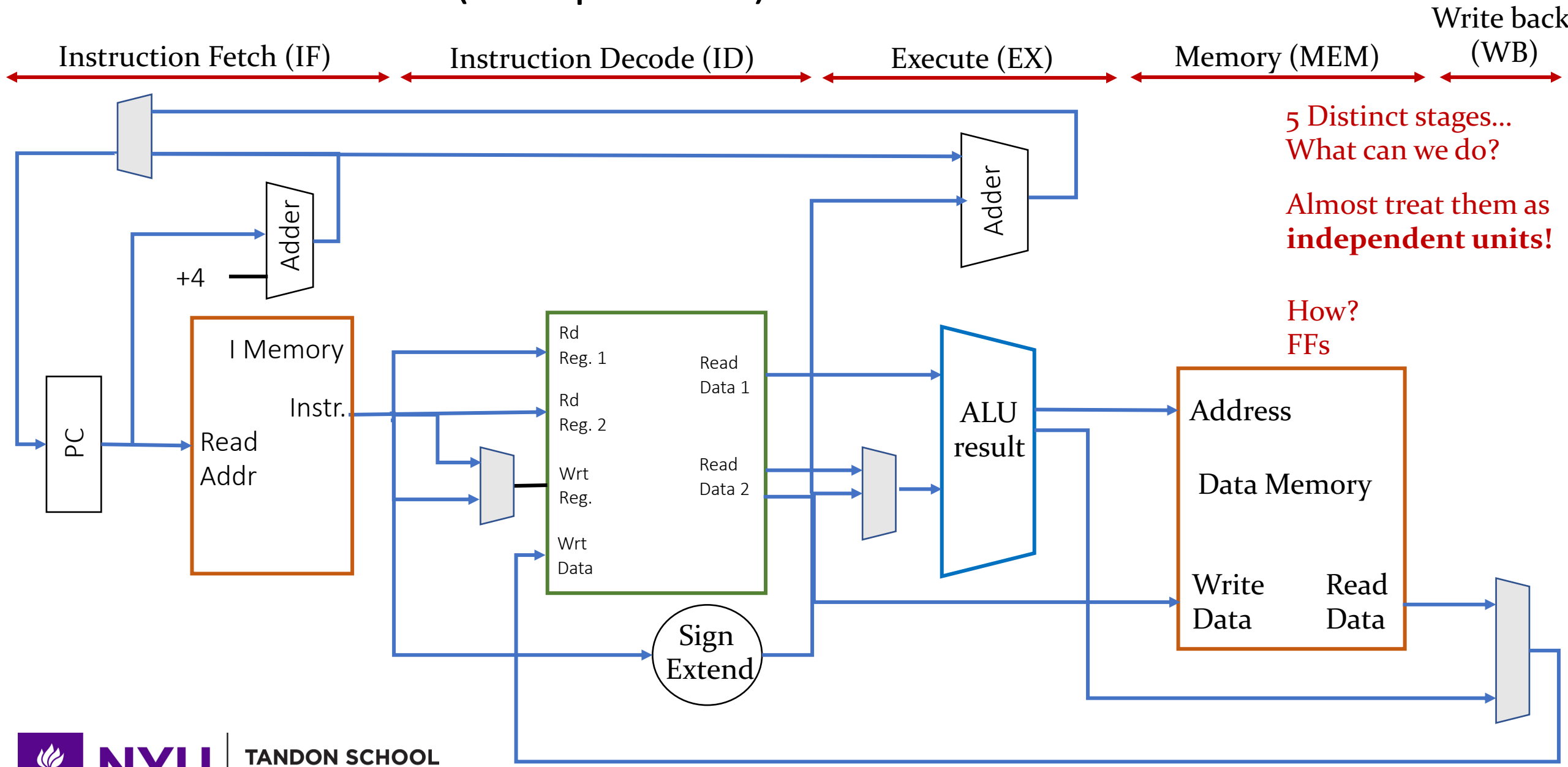
a) Imem
b) RF
c) ALU
d) Mem
e) Write back

What's the problem with this? LW?

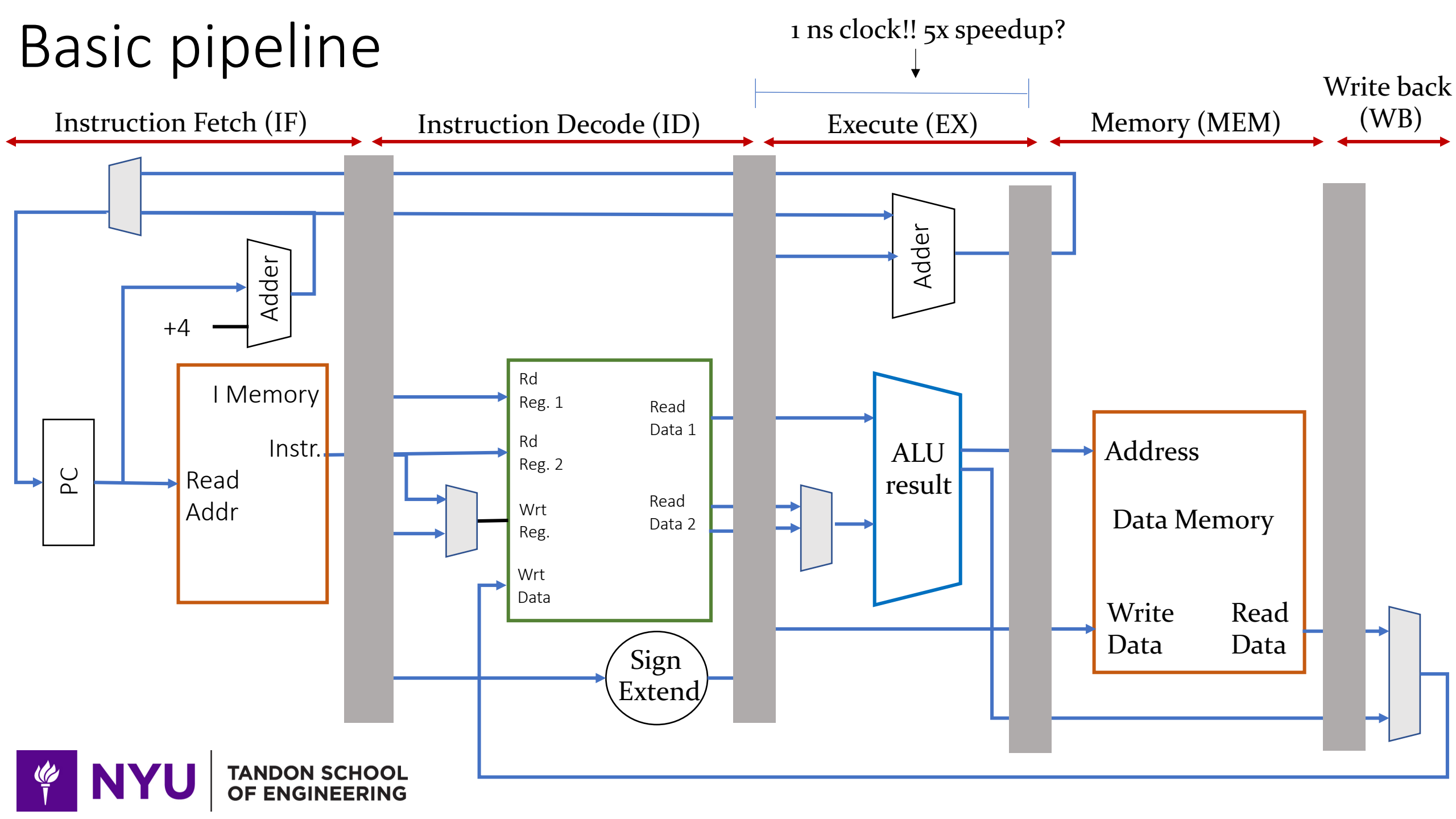
5ns clock => 200MHz
How do modern CPUs go faster?



From last time (simplified)



Basic pipeline



Why does pipelining help performance?

The previous diagram actually hurts perf.. Why?

single cycle machine = 5ns/inst

Pipeline = (1ns logic delay + FF delay) * 5 stages = >5ns/inst

So we improved our clock speed, but things got slower?

What'd we miss?

Improving performance is related to hardware utilization.

Core concept: Laundry example

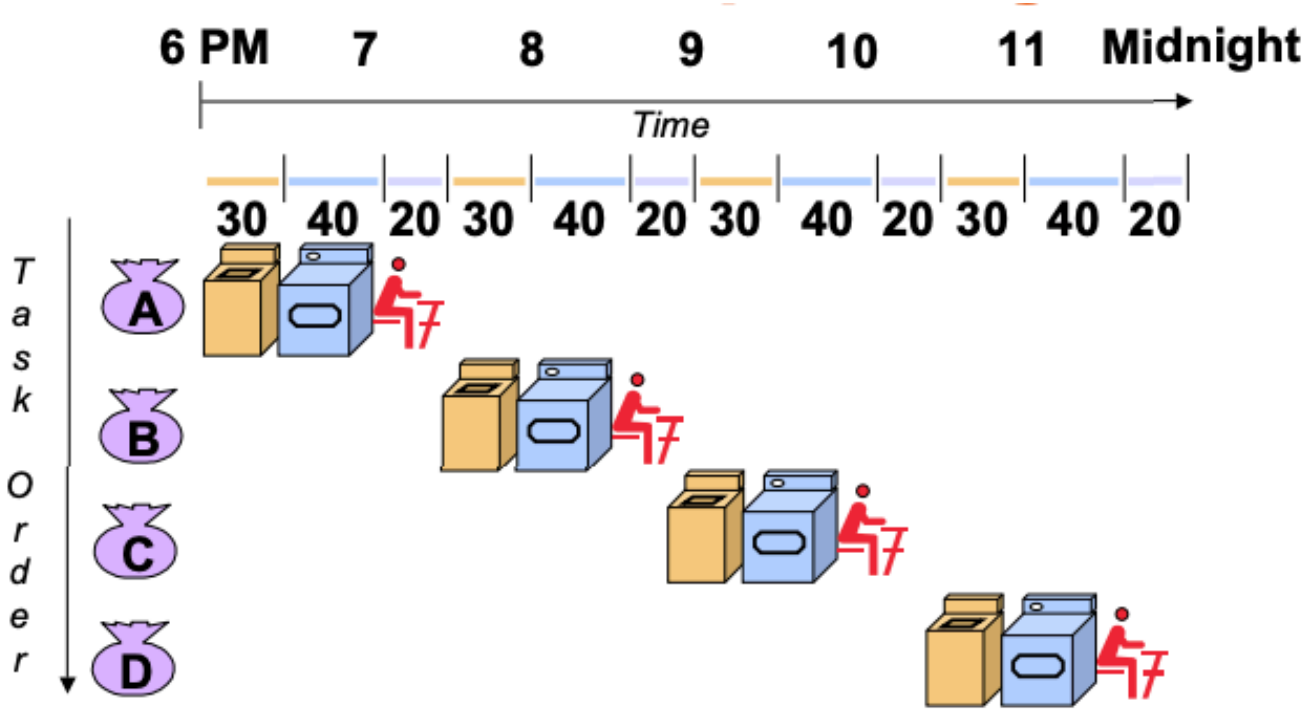
Ann, Brian, Cathy, Dave
need to do laundry

Wash = 30m

Dry = 40m

Fold = 20m

“Naïve” Solution: like single-cycle MIPS



Takes 6 hours.

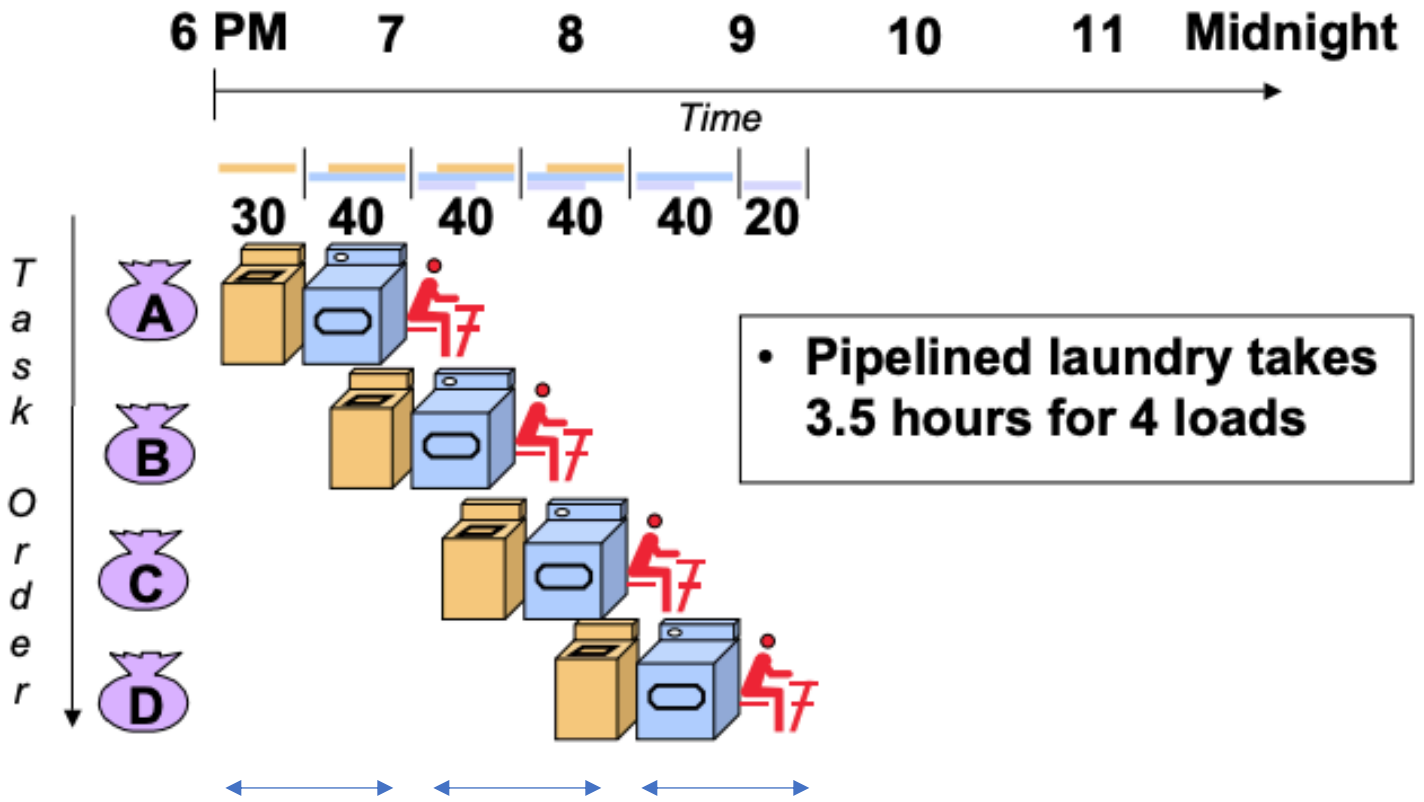
What’s wrong with this?

You’re not utilizing available resources!

Wasting time.

How can we do better?

What happens if we overlap work?

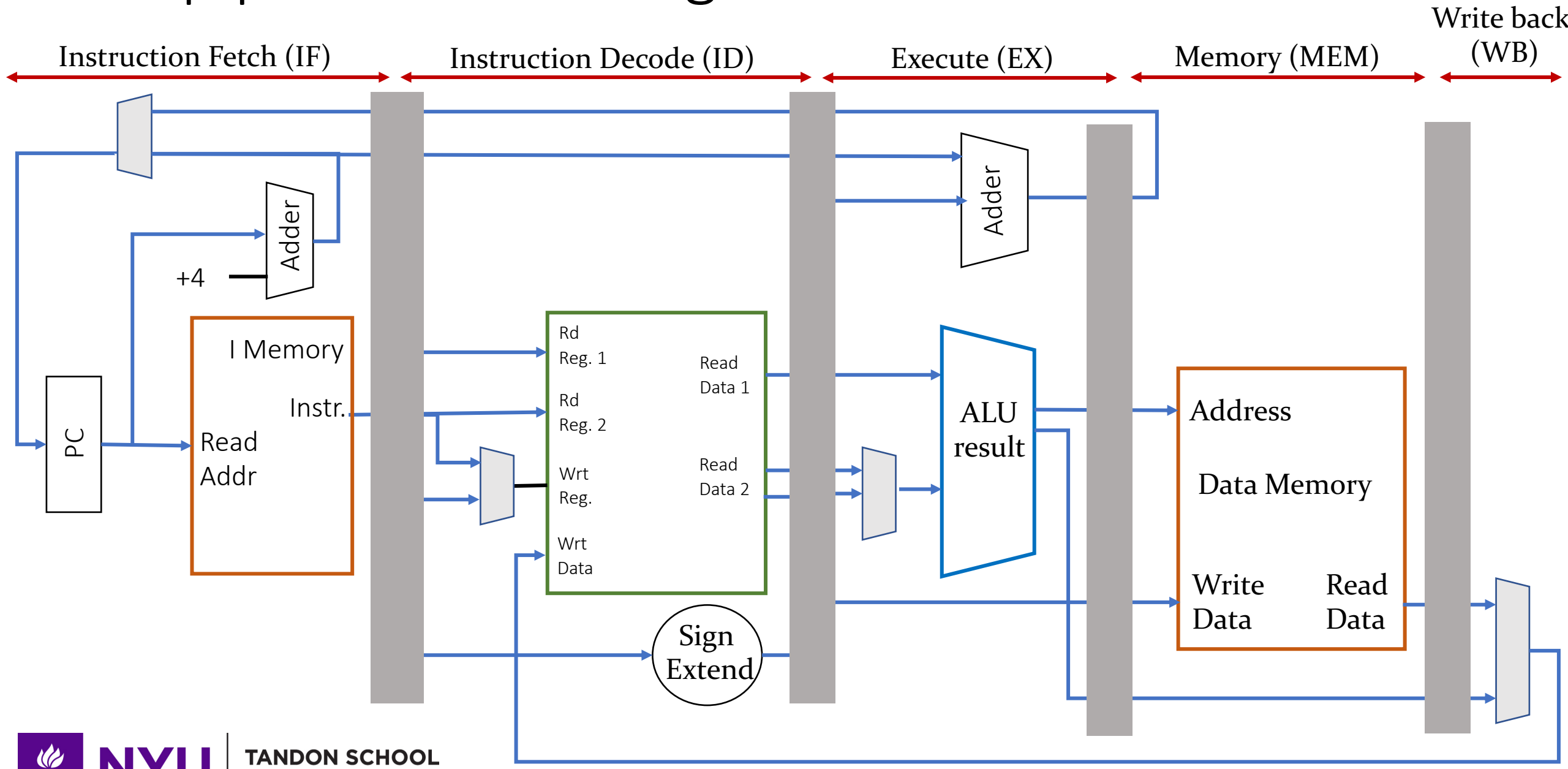


Pipelining takeaways:

- Doesn't improve single task latency, it improves throughput
- Pipeline "rate" governed by slowest pipeline stage
- Speedup ~= number of pipeline stages
- Unbalanced stages reduces speedup
- Time to fill and drain pipe also reduces speedup

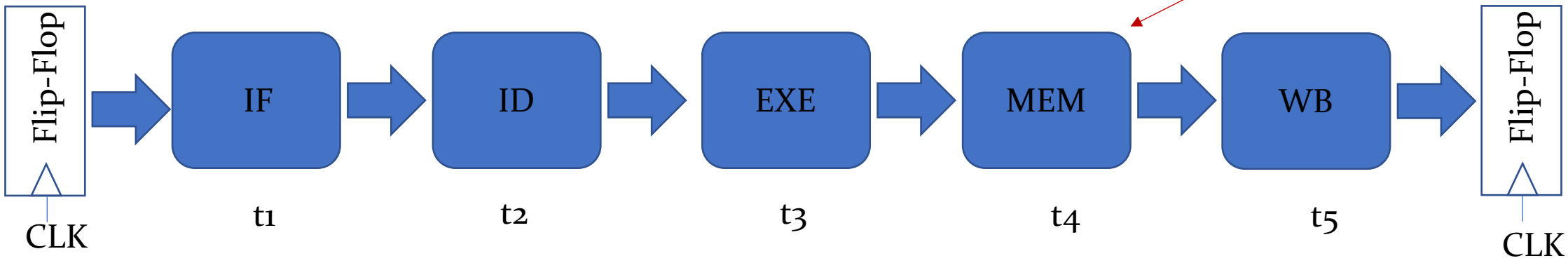
What's my throughput?
What happens when I have more loads to do?
What if I own a laundry mat, is this good or bad?
Power bill?

Basic pipeline: each stage its own machine!

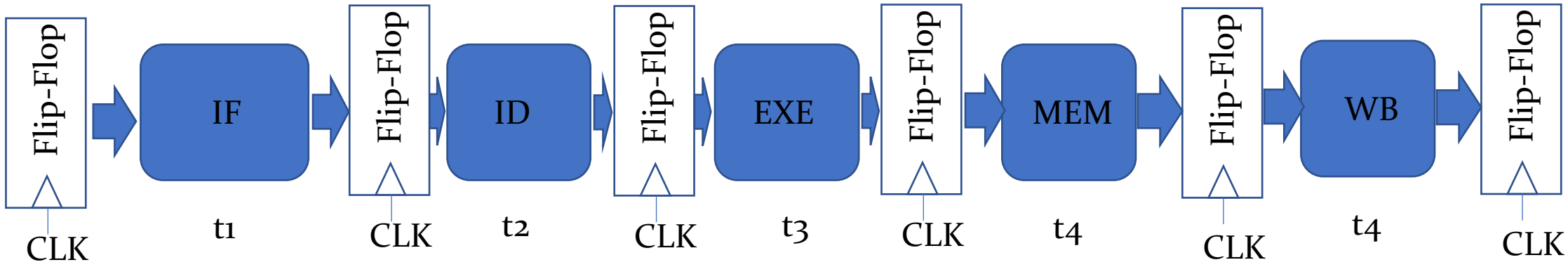


Improving Throughput: Pipelining

Think about HW utilization when accessing memory…



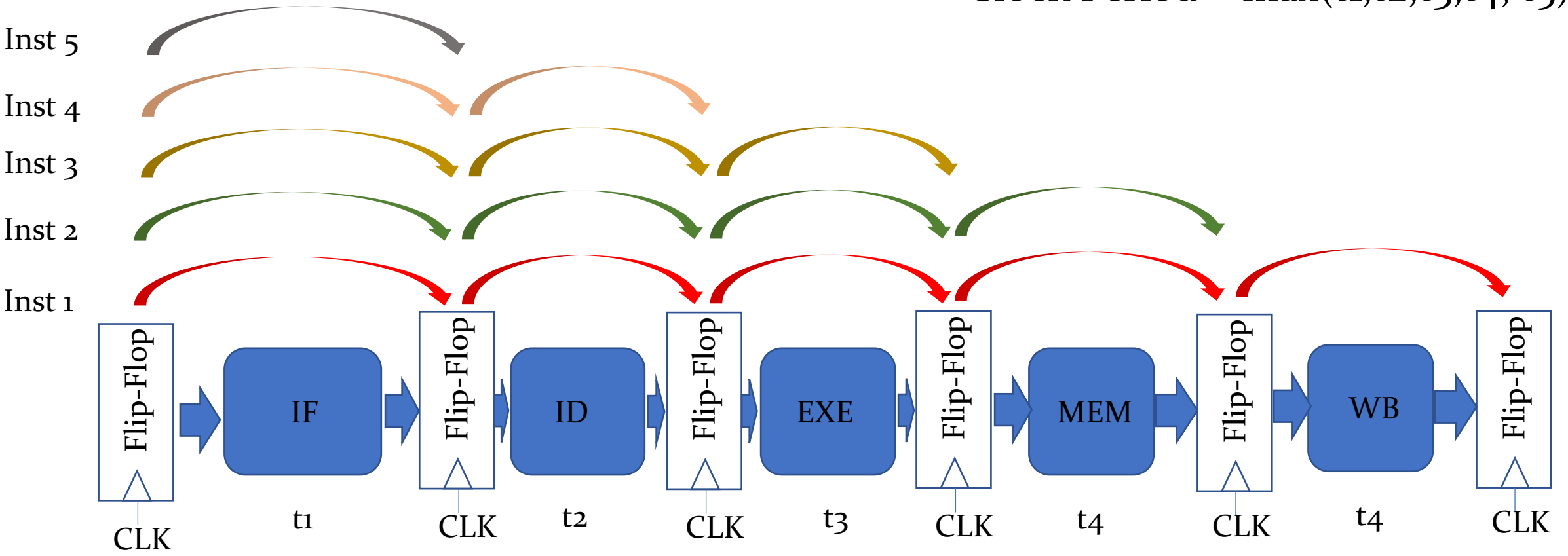
Throughput = 1/(t1+t2+t3+t4+t5)



Throughput = ?

Pipeline Operation

Clock Period = max(t1,t2,t3,t4, t5)



In steady state, each module processes data in parallel

Throughput = 1 / max(t1,t2,t3,t4,t5)

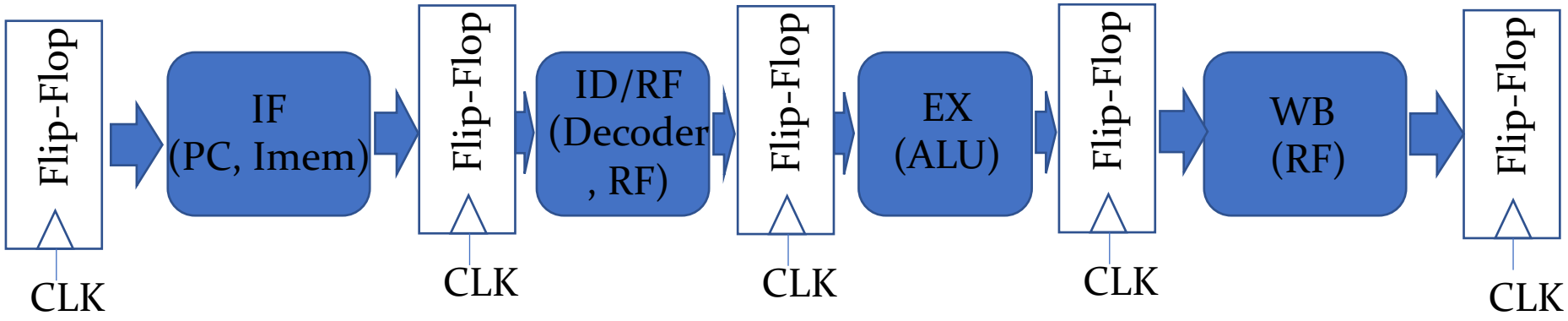
Q: If each stage takes 1ns, what's our speedup?

Did we miss anything?

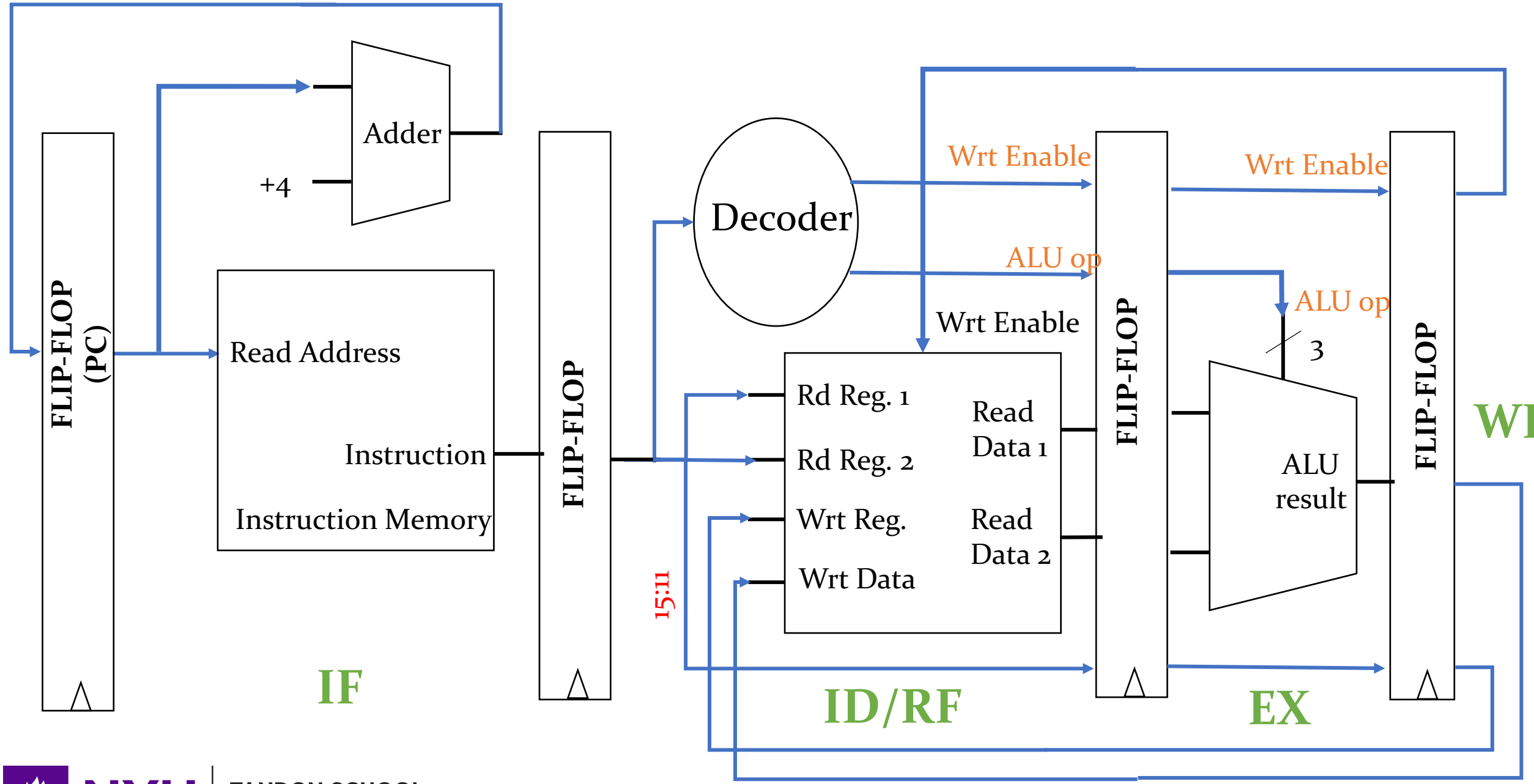
Pipelining the Single-Cycle R-Type MIPS machine

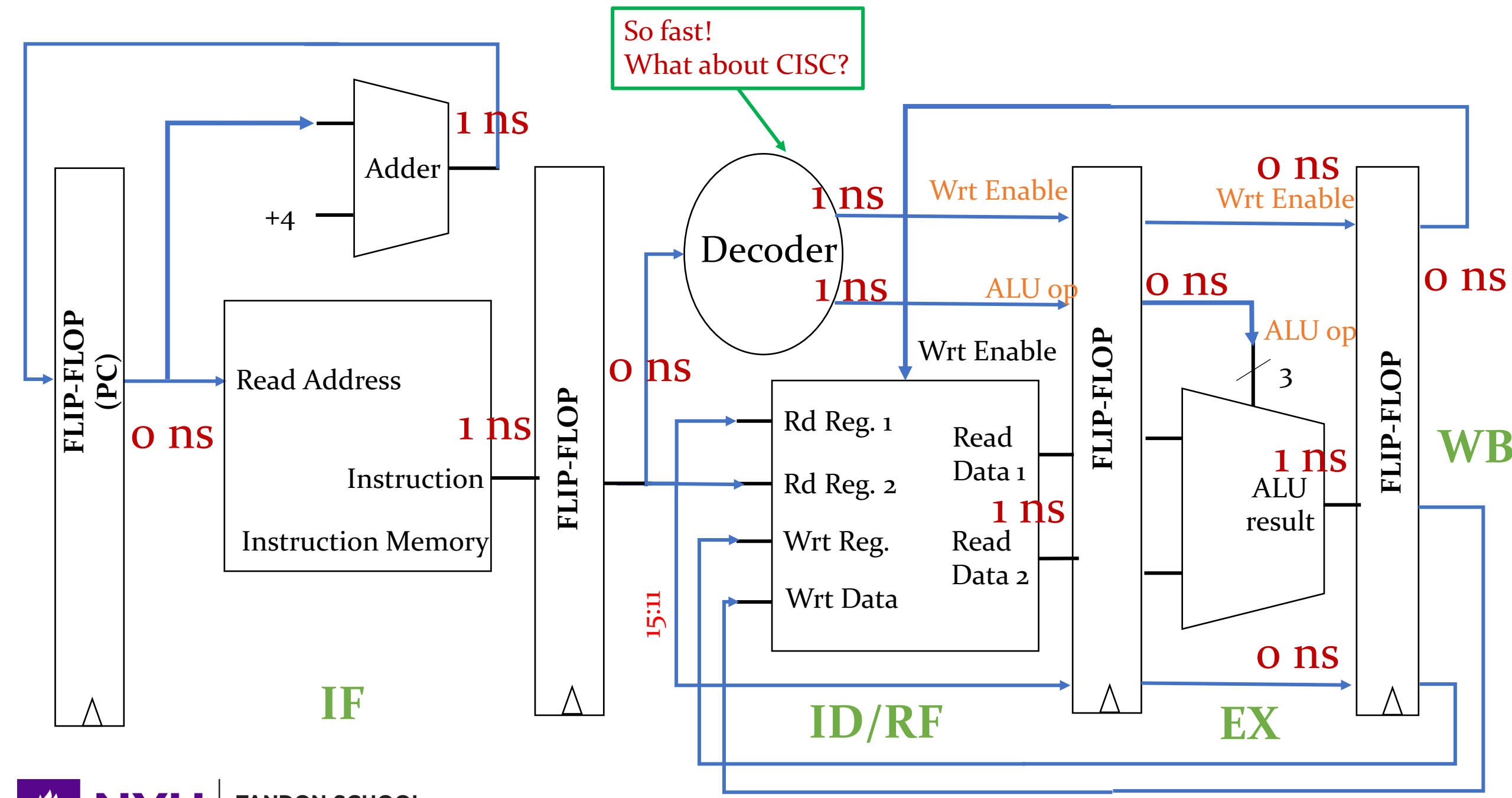
Basic steps in executing an R-type instruction

- Instruction fetch (IF)
- Instruction Decode/Register File Read (ID/RF)
- Execute in ALU (EX)
- Write-back Result to RF (WB)



For now we will ignore ld/st instructions!

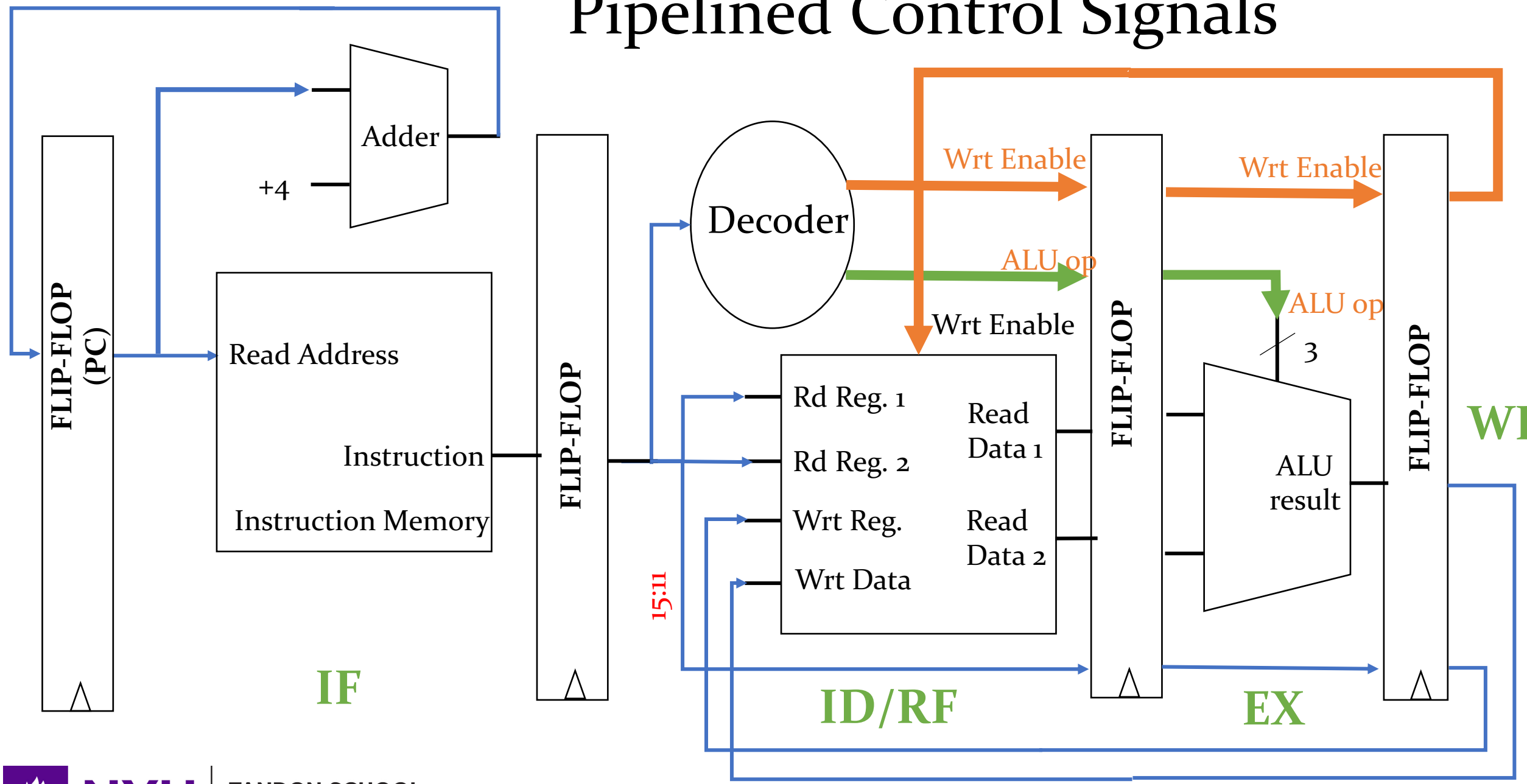

Adder
+4
FLIP-FLOP (PC)
Read Address
Instruction
Instruction Memory
FLIP-FLOP
Decoder
Wrt Enable
ALU op
Wrt Enable
Rd Reg. 1
Rd Reg. 2
Wrt Reg.
Wrt Data
Read Data 1
Read Data 2
15:11
FLIP-FLOP
Wrt Enable
ALU op
3
ALU result
FLIP-FLOP
WB
IF
ID/RF
EX

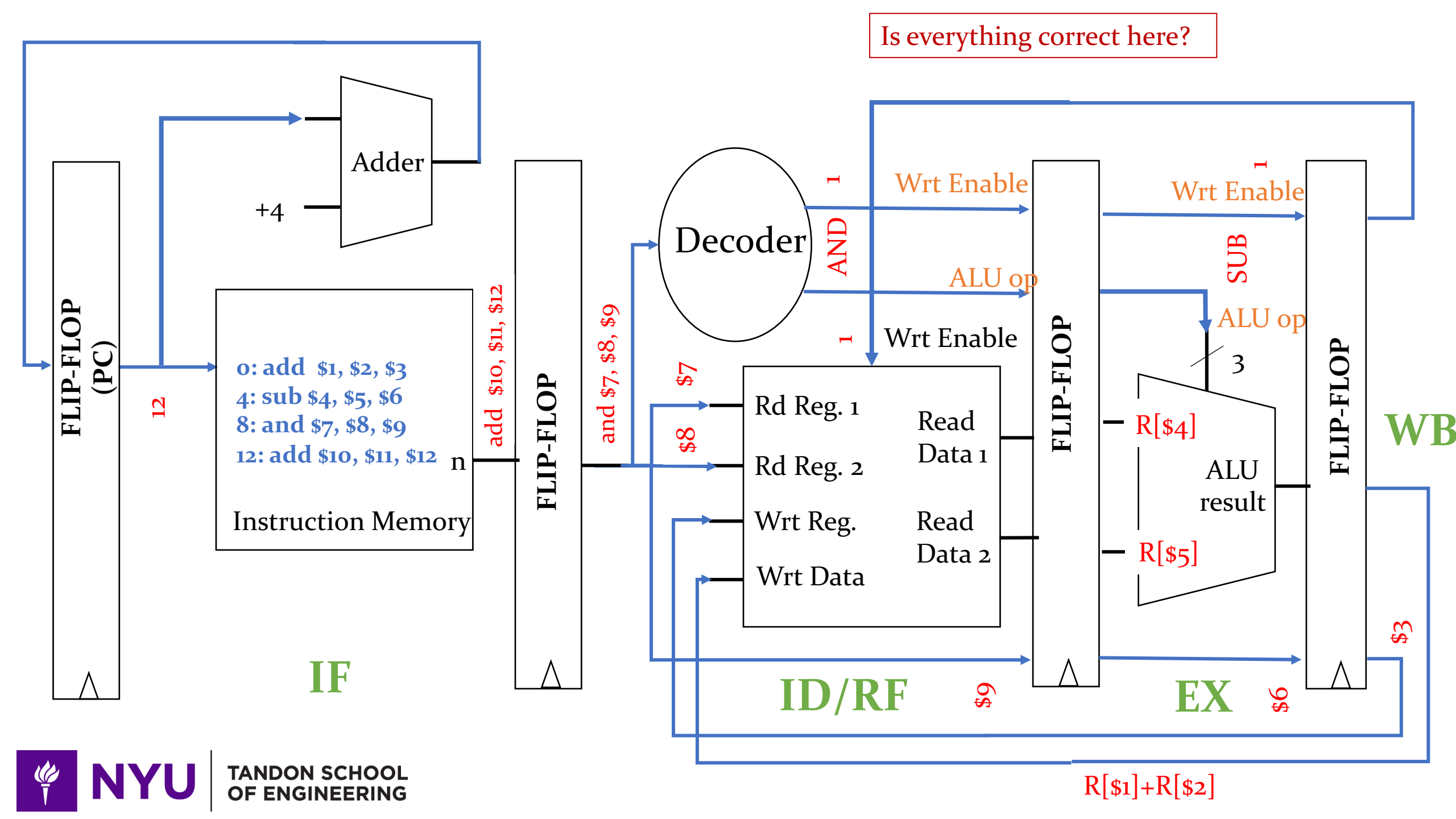


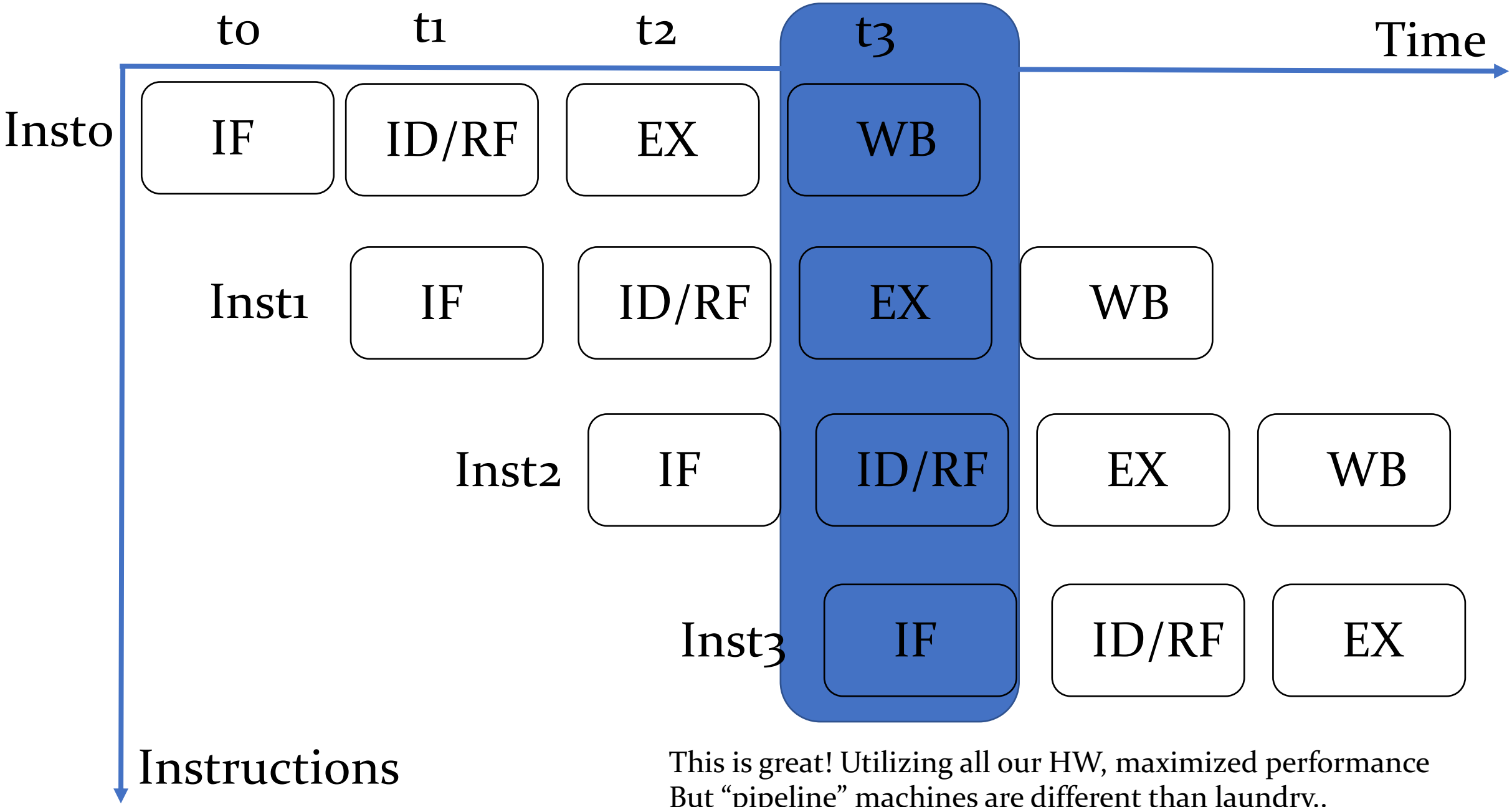
NYU TANDON SCHOOL OF ENGINEERING

Clock Period: 1 ns

Pipelined Control Signals




Is everything correct here?
Adder
+4
FLIP-FLOP (PC)
12
0: add $1, $2, $3
4: sub $4, $5, $6
8: and $7, $8, $9
12: add $10, $11, $12
n
Instruction Memory
add $10, $11, $12
FLIP-FLOP
and $7, $8, $9
Decoder
1
AND
Wrt Enable
ALU op
1
Wrt Enable
$7
$8
Rd Reg. 1
Rd Reg. 2
Wrt Reg.
Wrt Data
Read Data 1
Read Data 2
FLIP-FLOP
SUB
ALU op
3
R[$4]
R[$5]
ALU result
FLIP-FLOP
WB
$3
$9
$6
R[$1]+R[$2]
IF
ID/RF
EX



This is great! Utilizing all our HW, maximized performance
But "pipeline" machines are different than laundry..
Why?

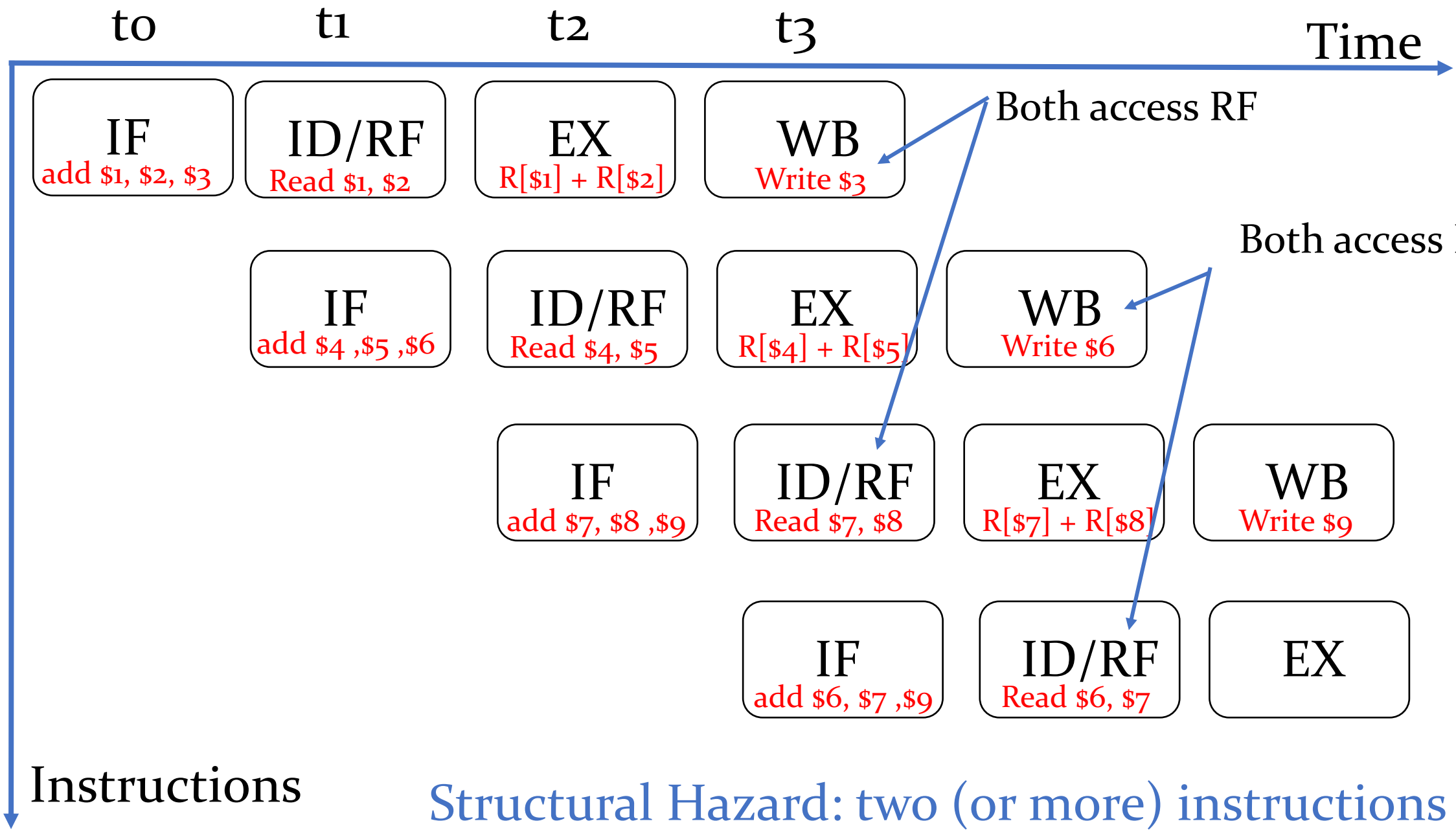
Hazards!

Hazard (formal) := “when the next instruction cannot execute in the following clock cycle”

Hazard (me) := “you need something you don’t have”

There are three of them..
Think about what goes into a program and how they execute

1) Structural : hardware resources
2) Data : need result from prior computation
3) Control : need to know where to go next



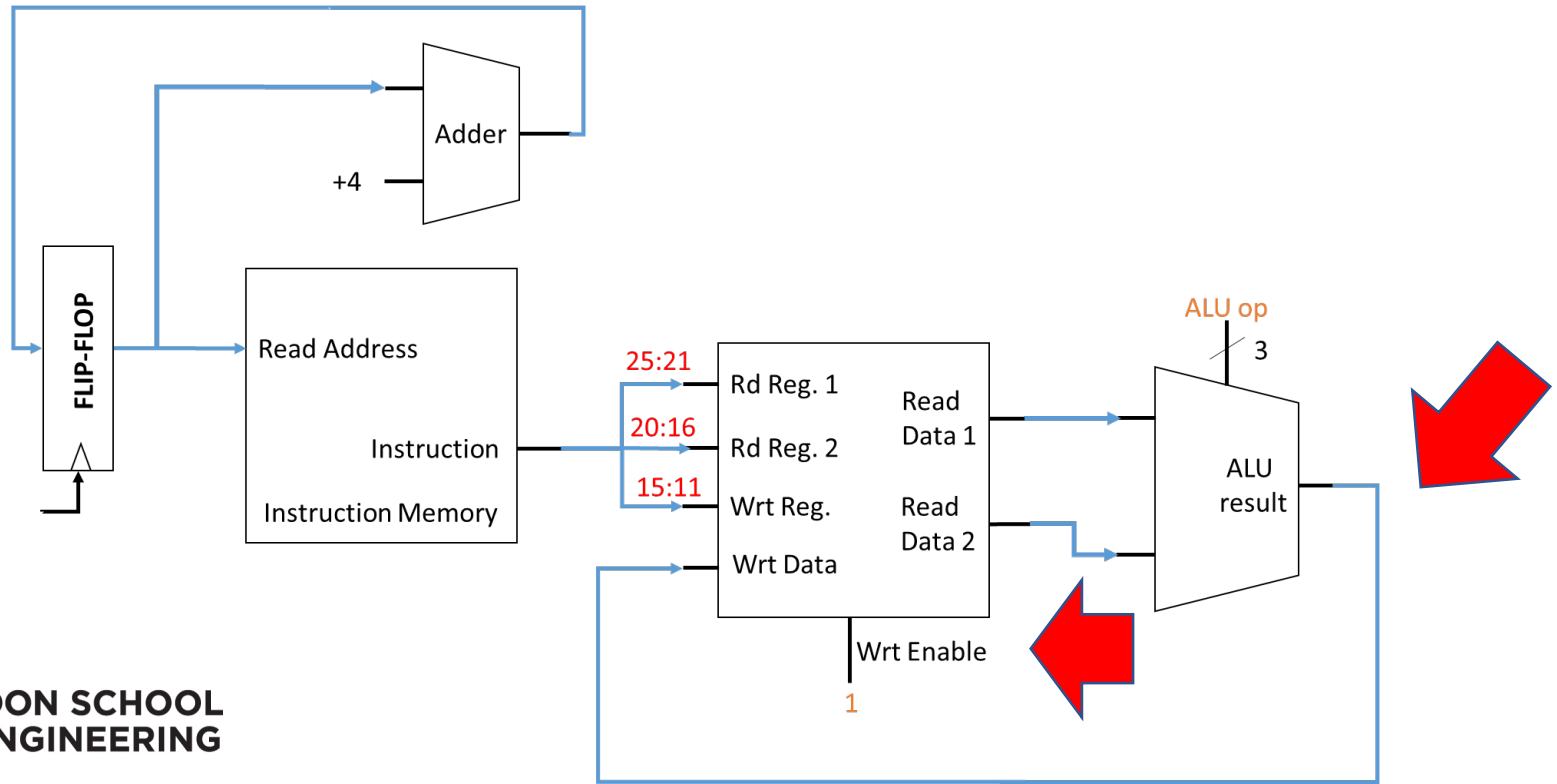
Structural Hazard: two (or more) instructions accessing the same hardware module

Execute/Write-Back Stage Operation

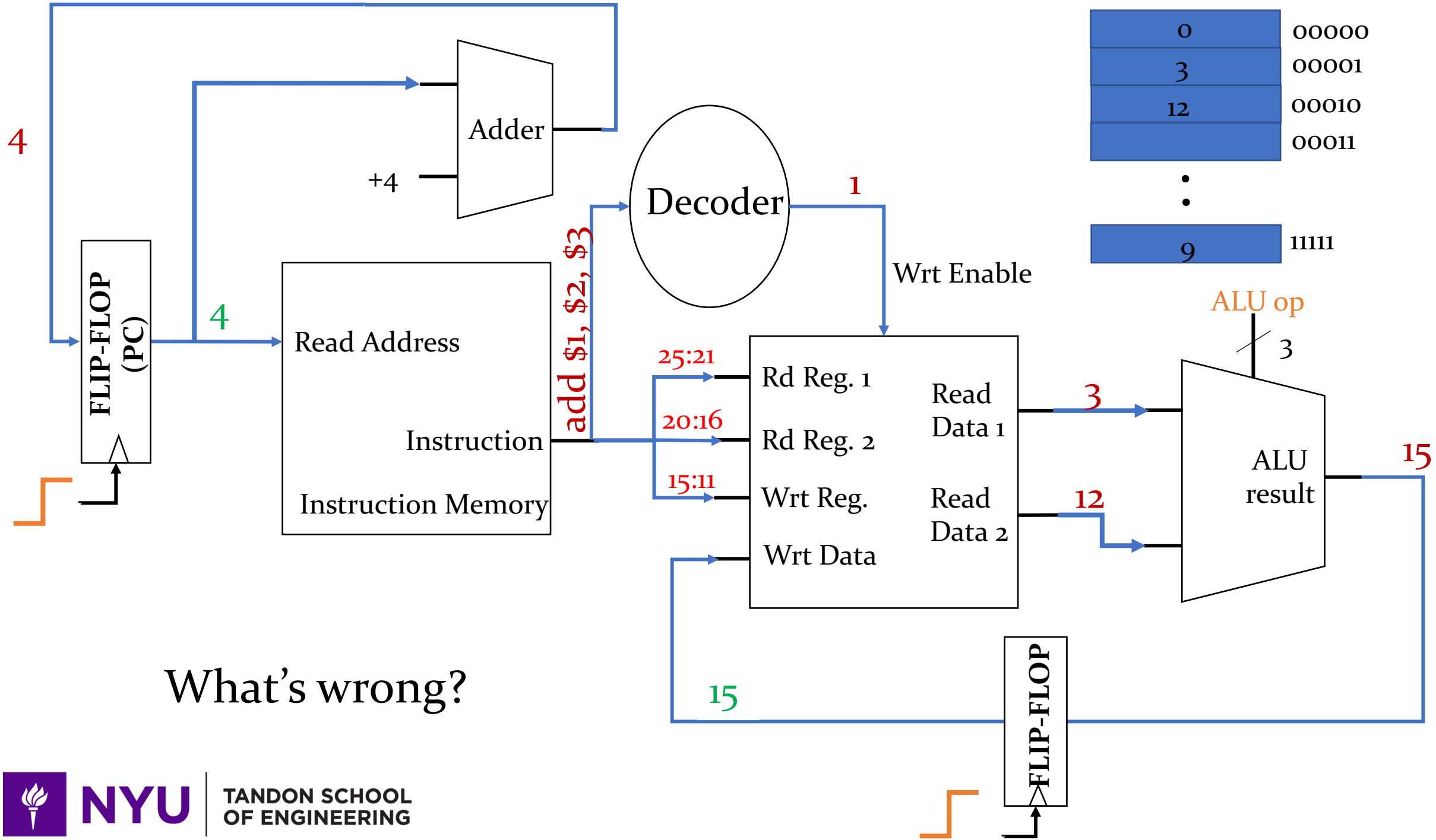
All updates to architectural state should occur in first half of period

- PC (already done!)
- Register file
- Data memory (not relevant for R-type instructions)

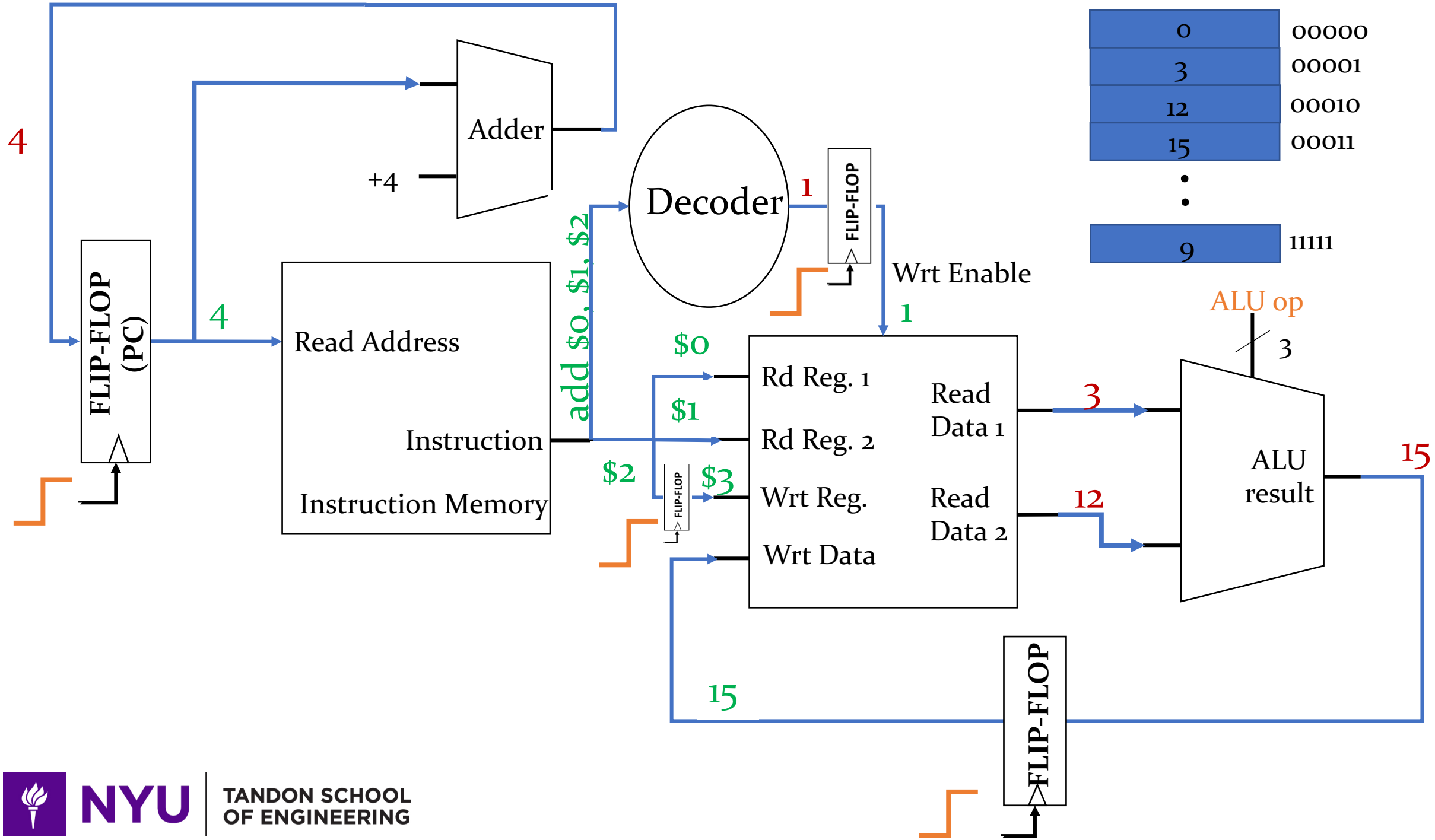
If we achieve this, we can write and read in same cycle!

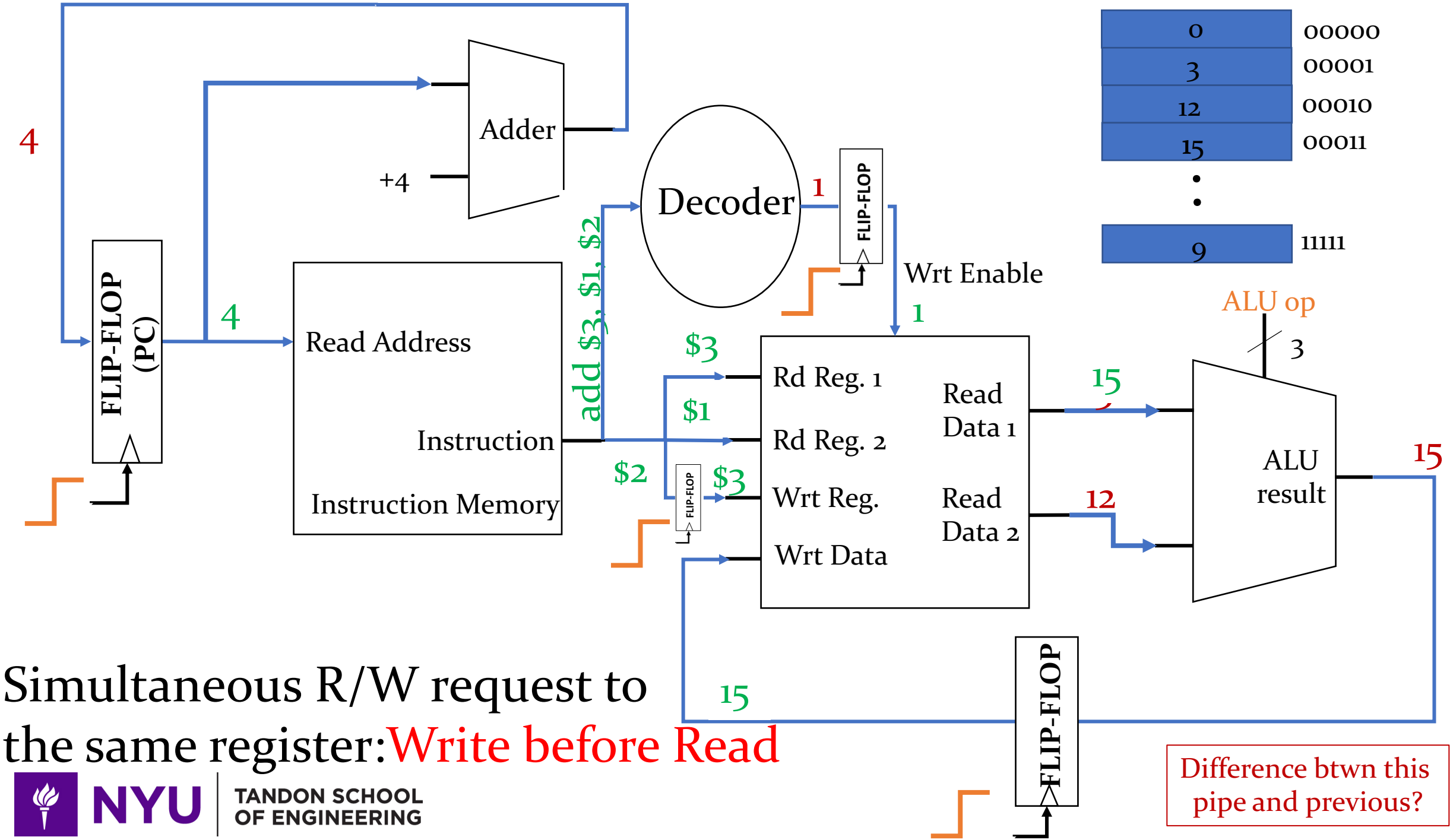


Test understanding of FFs and hazards with example..


4
Adder
+4
FLIP-FLOP
(PC)
4
Read Address
Instruction
Instruction Memory
add $1, $2, $3
Decoder
1
Wrt Enable
25:21
20:16
15:11
Rd Reg. 1
Rd Reg. 2
Wrt Reg.
Wrt Data
Read
Data 1
Read
Data 2
3
12
ALU op
3
ALU
result
15
15
FLIP-FLOP
0
3
12
9
00000
00001
00010
00011
11111

What's wrong?


4
Adder
+4
Decoder
1
FLIP-FLOP
Wrt Enable
1
0
3
12
15
9
00000
00001
00010
00011
11111
ALU op
3
FLIP-FLOP
(PC)
4
Read Address
Instruction
Instruction Memory
add $0, $1, $2
$0
$1
$2
$3
FLIP-FLOP
Rd Reg. 1
Rd Reg. 2
Wrt Reg.
Wrt Data
Read Data 1
Read Data 2
3
12
ALU result
15
15
FLIP-FLOP



Simultaneous R/W request to the same register: Write before Read

Difference btwn this pipe and previous?

NYU TANDON SCHOOL OF ENGINEERING

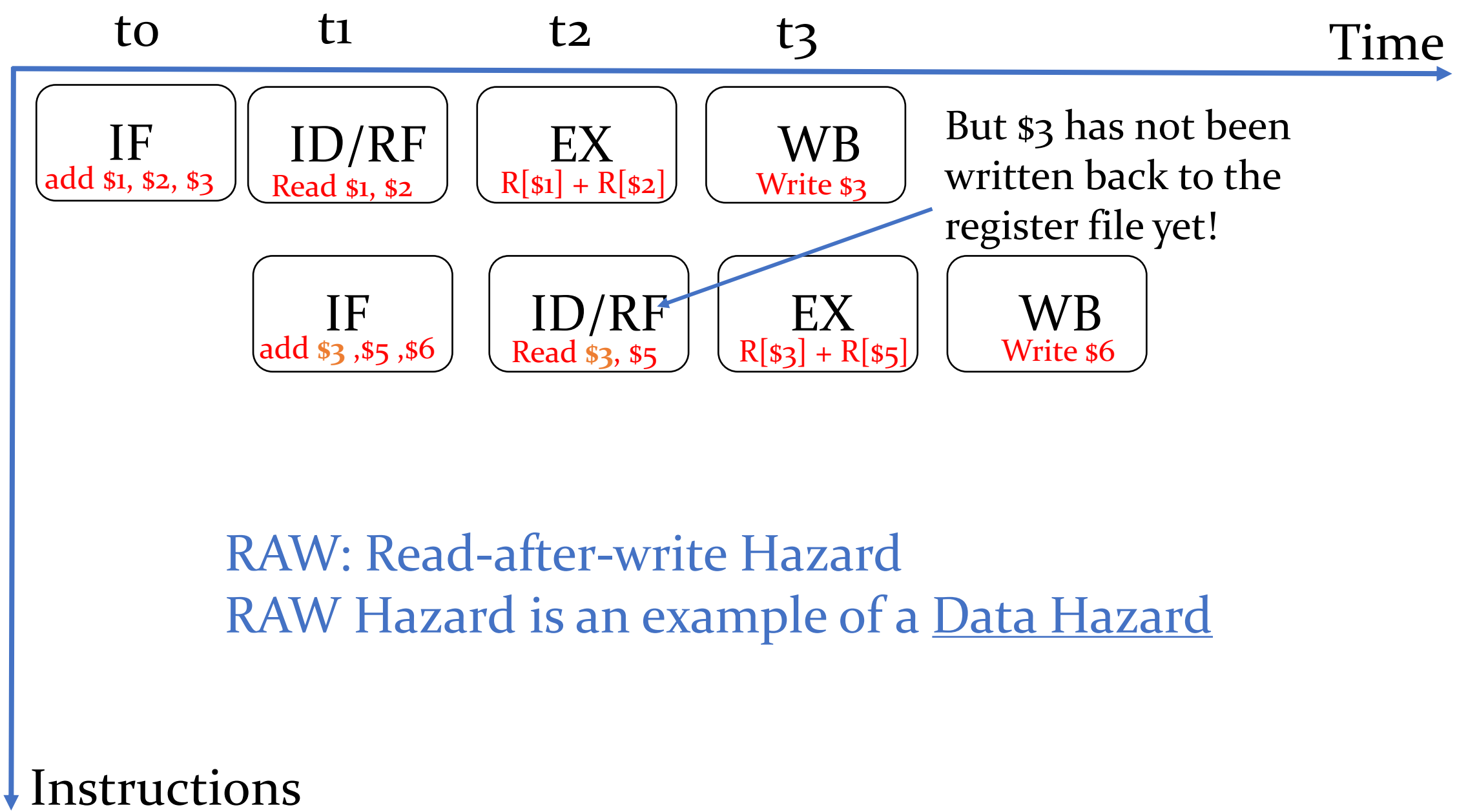
Hazards!

Hazard (formal) := “when the next instruction cannot execute in the following clock cycle”

Hazard (me) := “you need something you don’t have”

There are three of them..
Think about what goes into a program and how they execute

1) Structural : hardware resources : Solve with write before read
2) Data : need result from prior computation
3) Control : need to know where to go next



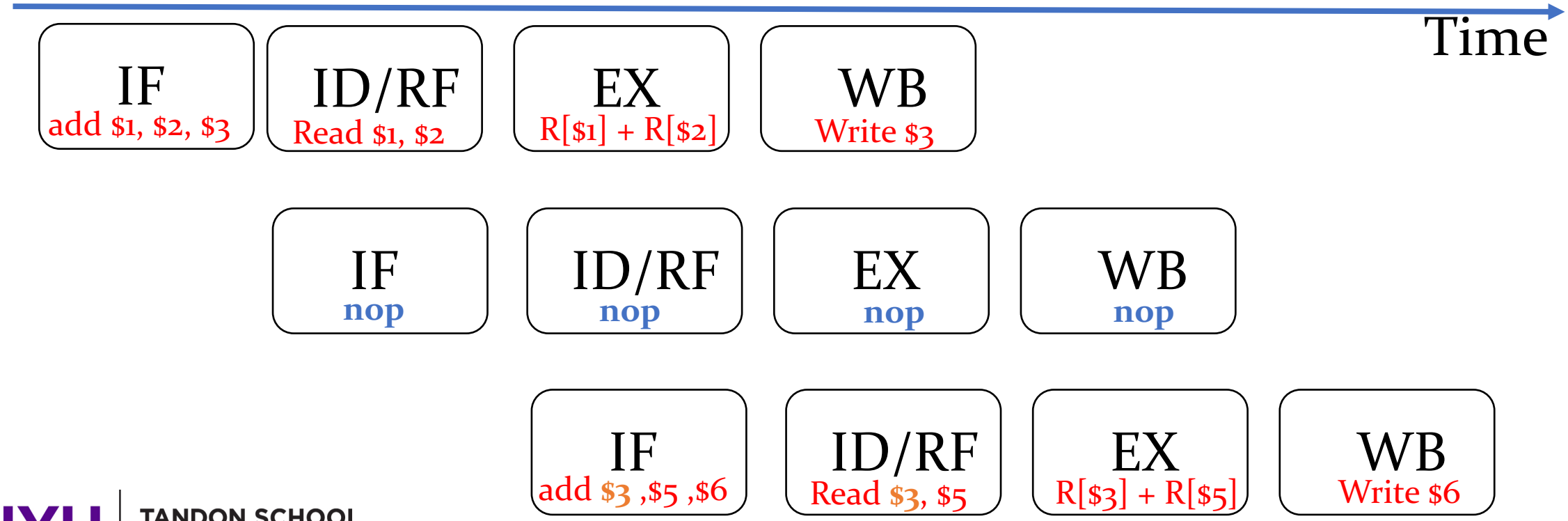
RAW: Read-after-write Hazard

RAW Hazard is an example of a Data Hazard

Handling RAW Hazards

Solution 1

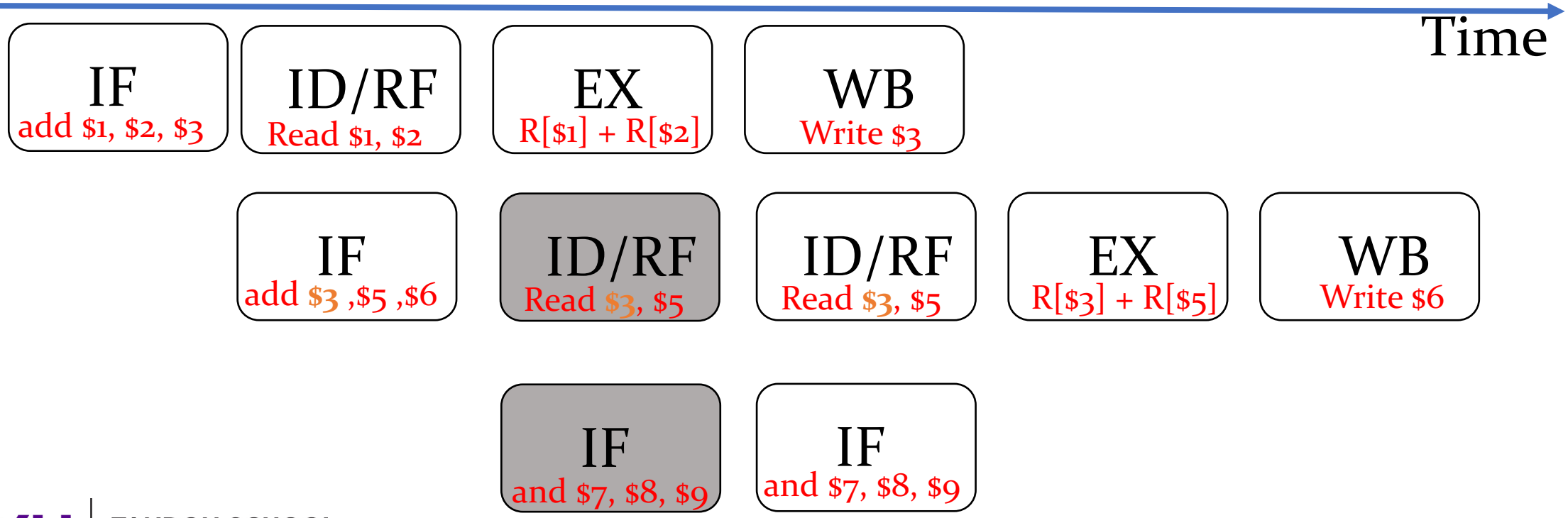
- Detect and avoid in software: compiler inserts nops (example: sll $0, $0, 0)
- Pros? Cons?



Handling RAW Hazards

Solution 2

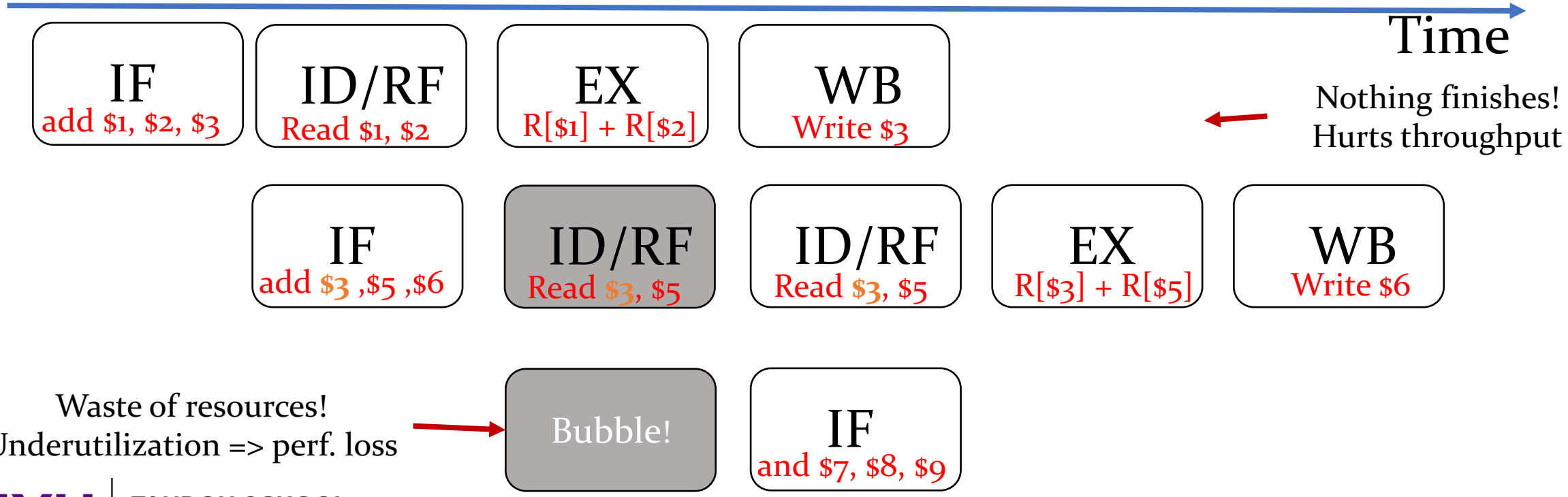
- Detect and avoid in hardware: stall the pipeline
- A stall does not update the pipeline register's values, keeps them the same
- Pros? Cons?



Handling RAW Hazards

Solution 2

- Detect and avoid in hardware: stall the pipeline
- A stall does not update the pipeline register's values, keeps them the same
- Pros? Cons?



Implementing Stalls: Details

Detect Stall in IF/ID stage by checking

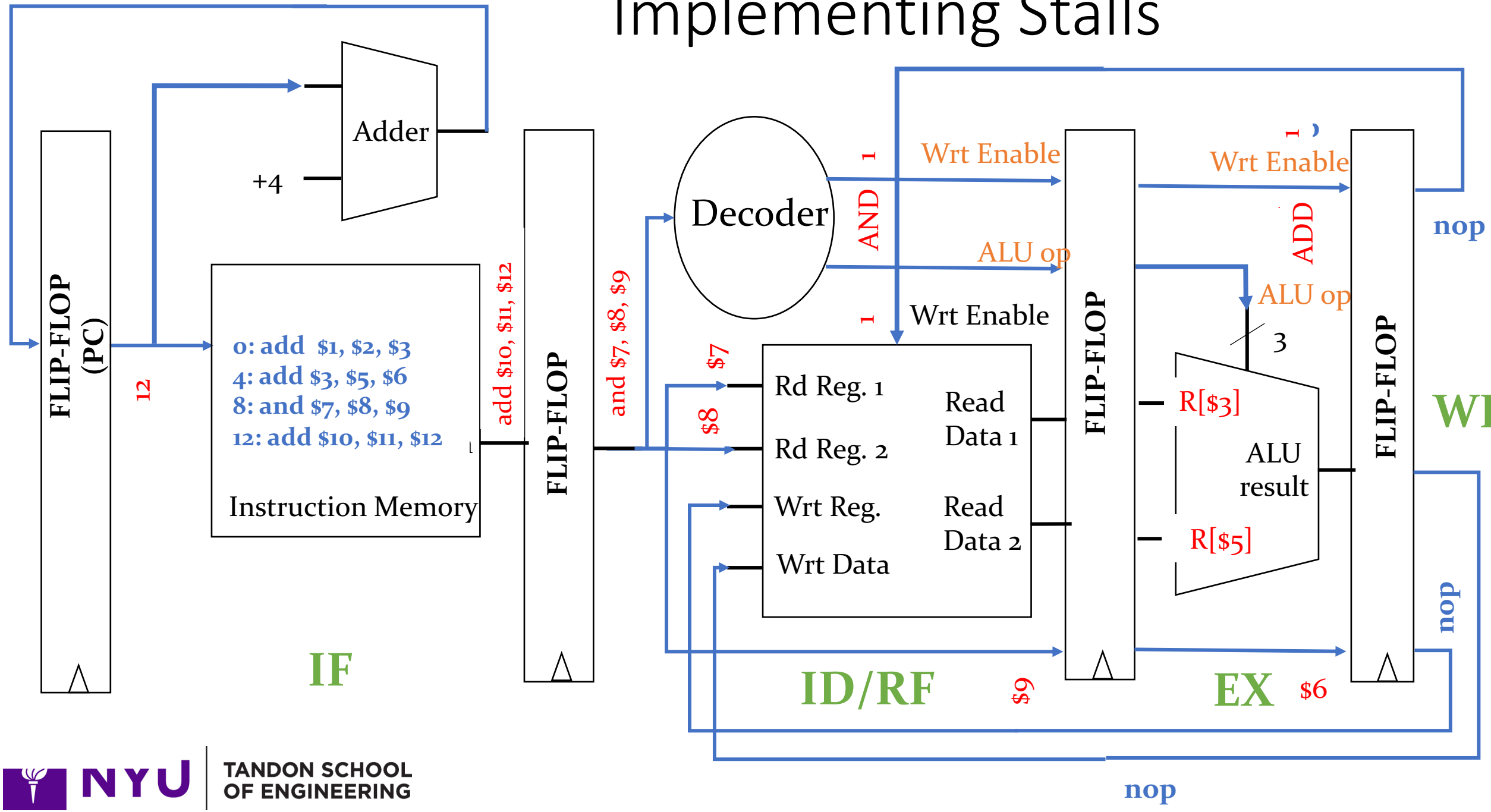
- Destination register of previous inst. == source register of current inst

When Stall is detected

- PC flip-flop is "stalled"
  - Pipeline register outputs the same value as in previous clock cycle
  - In other words: no update, ignore the positive edge of the clock
- ID/RF flip-flop is stalled
- Nop inserted in EX flip-flop
  - Else we'll keep detecting the hazard

Stall clears in next clock cycle

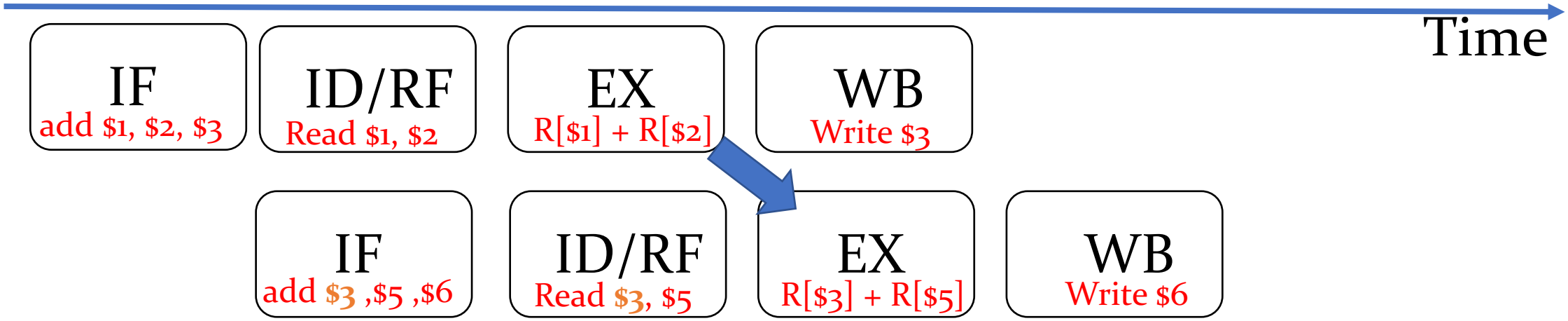
Implementing Stalls



Handling RAW Hazards

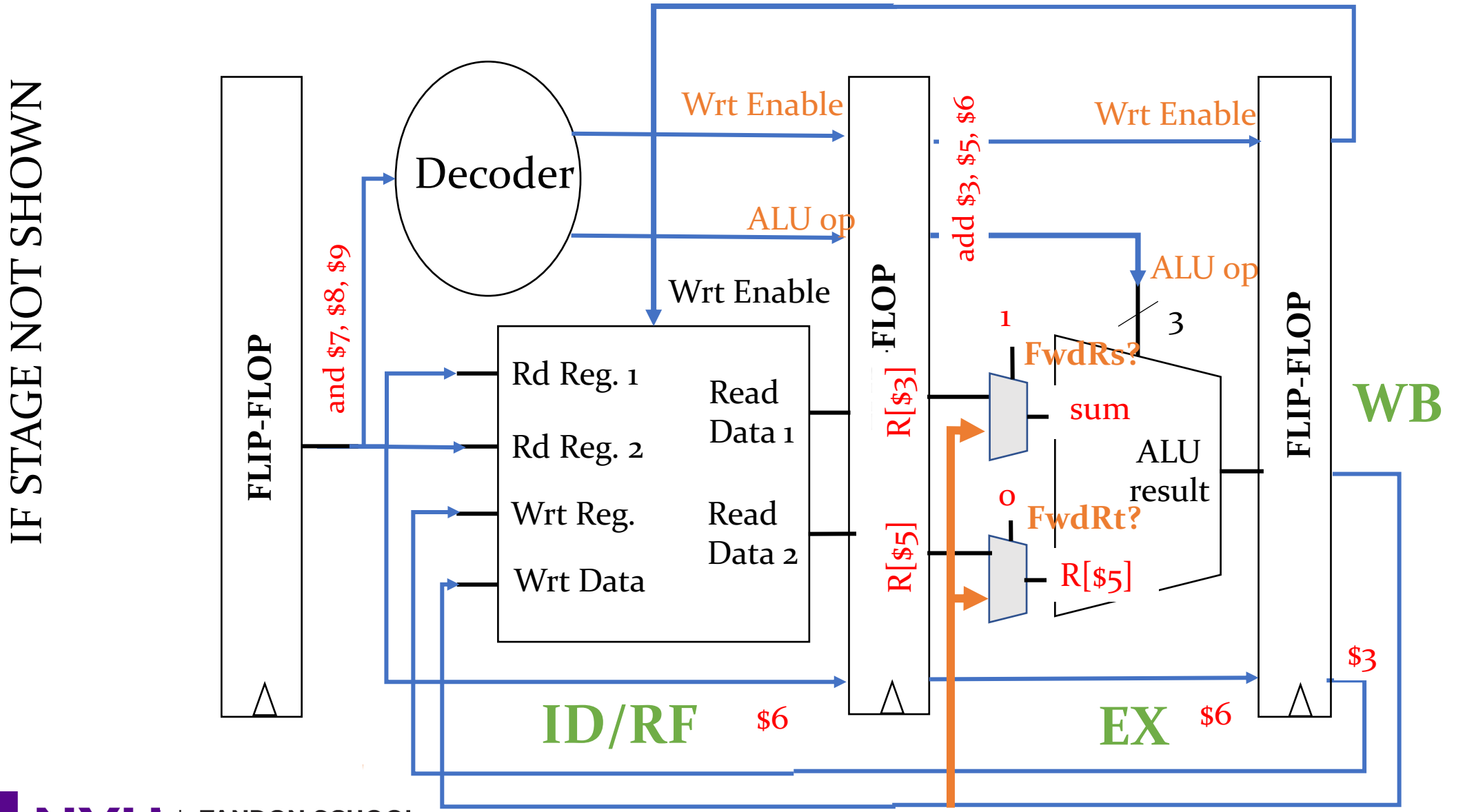
Solution 3

- Forward data from the EX stage
- Pros? Cons?

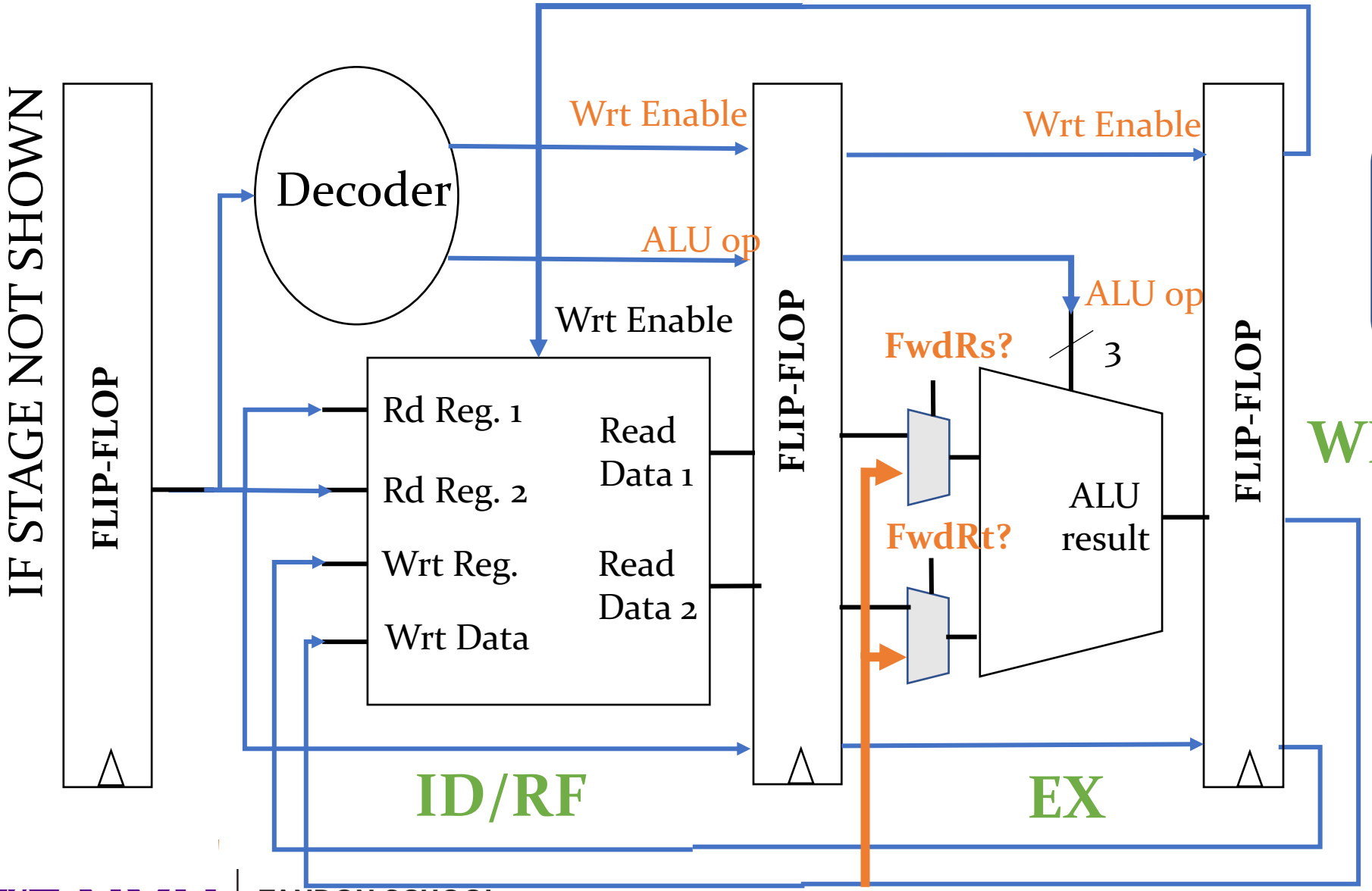


How would we implement this in our pipeline?

Implementing Forwarding



Forwarding Control

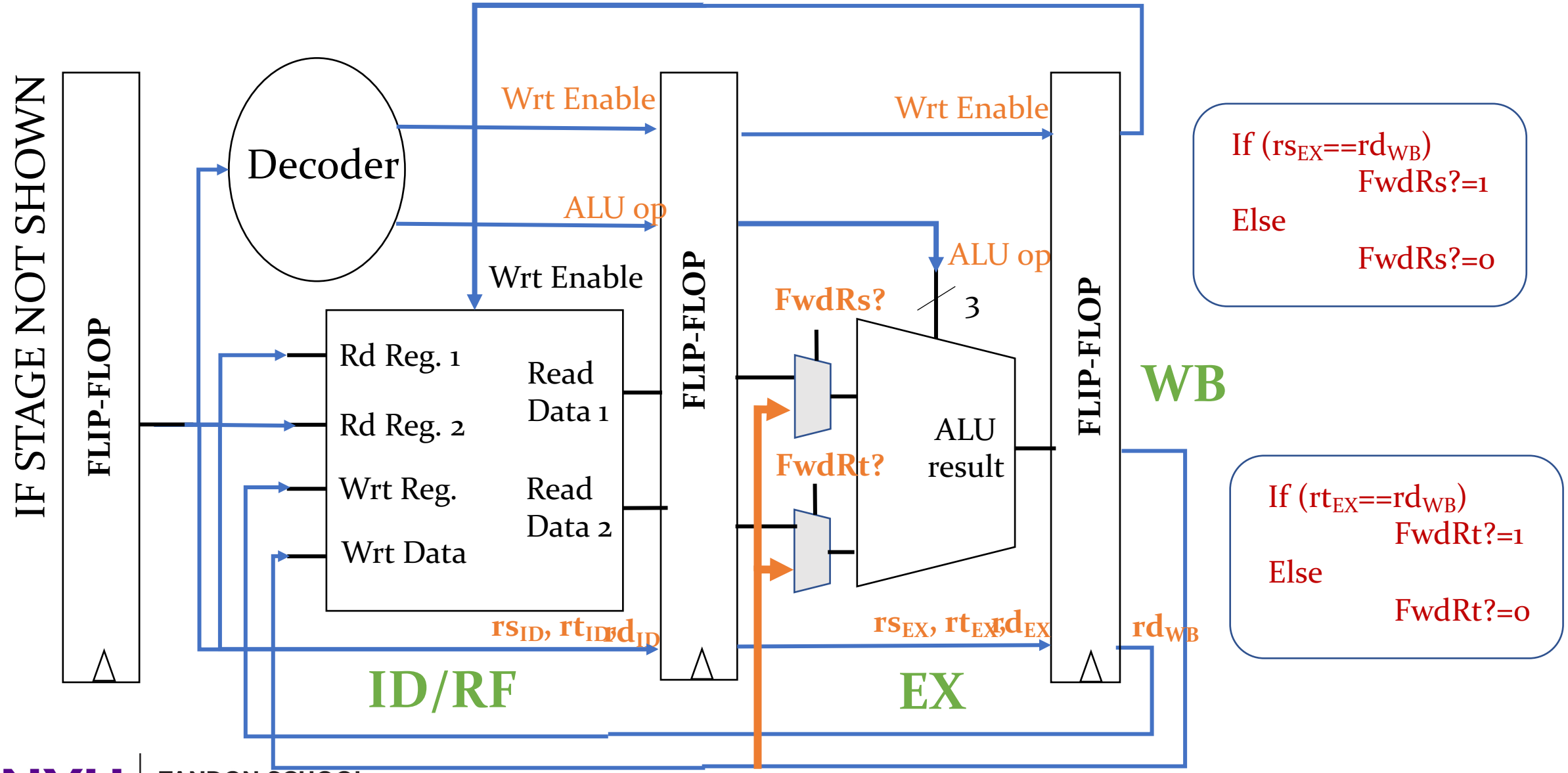


FwdRs? = 1 if destination register (rd) for inst in WB stage is same as source register (rs) for instruction in EX stage

But we don't know rs and rt in EX stage!

FwdRt? = 1 if destination register (rd) for inst in WB stage is same as source register (rt) for instruction in EX stage

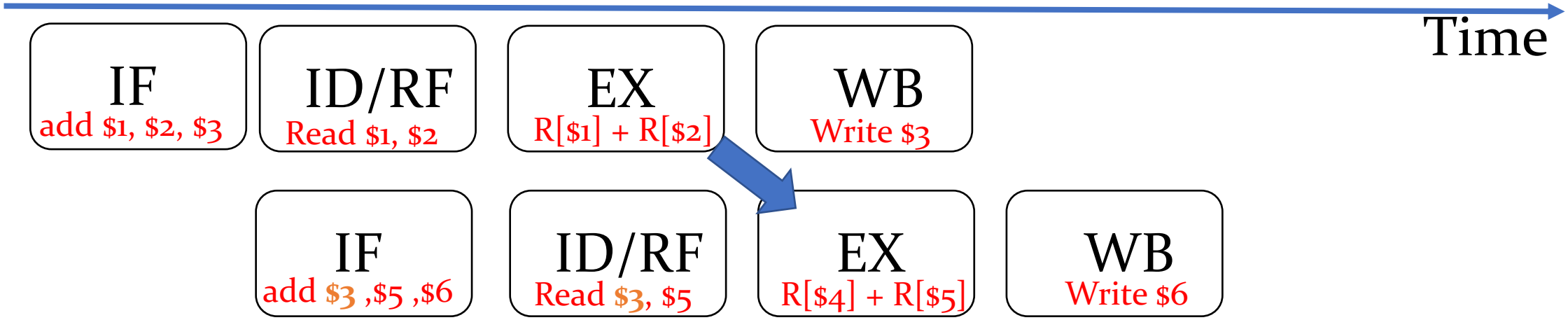
Forwarding Control



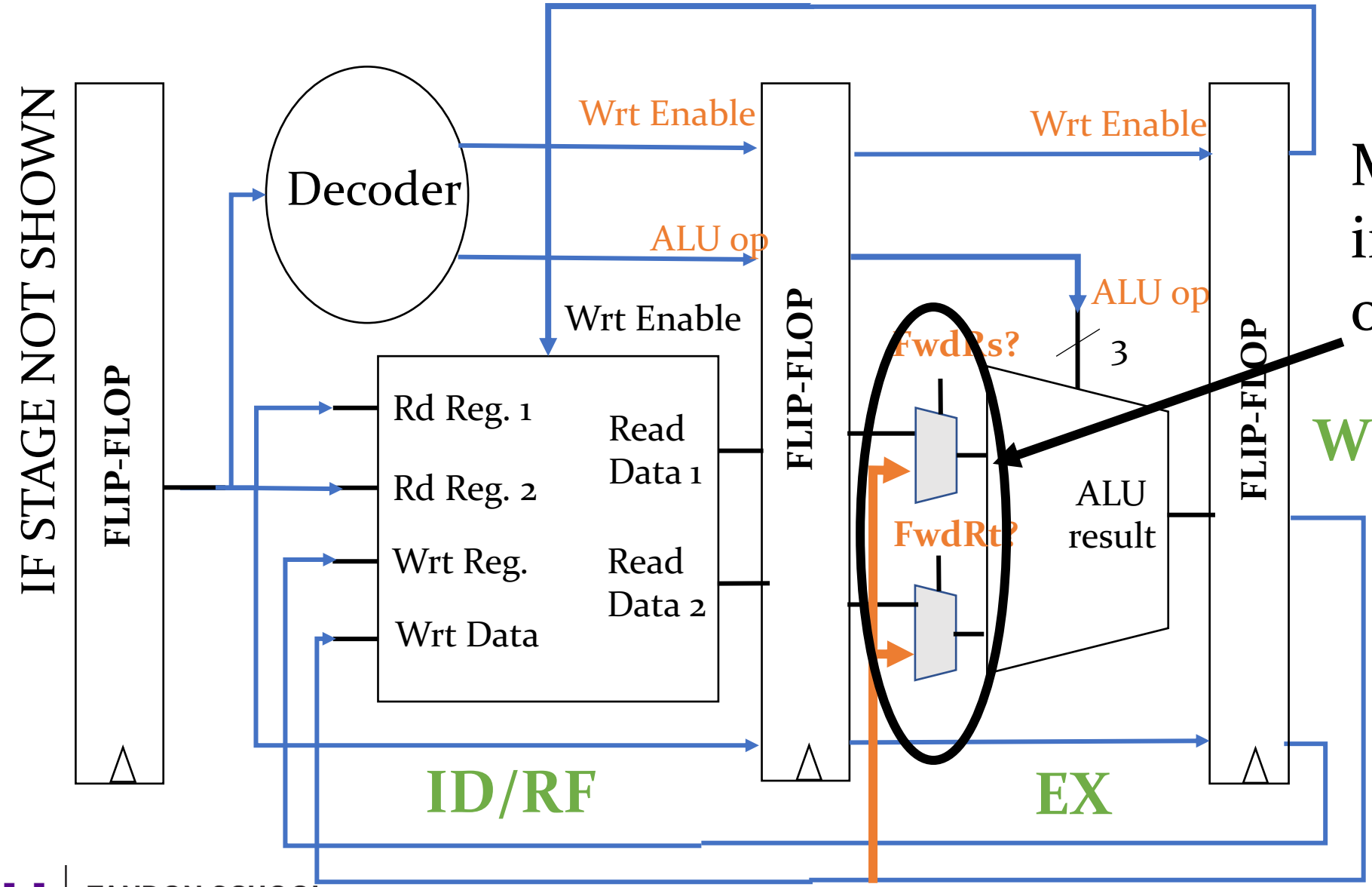
Handling RAW Hazards

Solution 3

- Forward data from the EX stage
- Impact on IPC (Instructions/cycle)?
- Impact on clock period?



Impact on Cycle Time

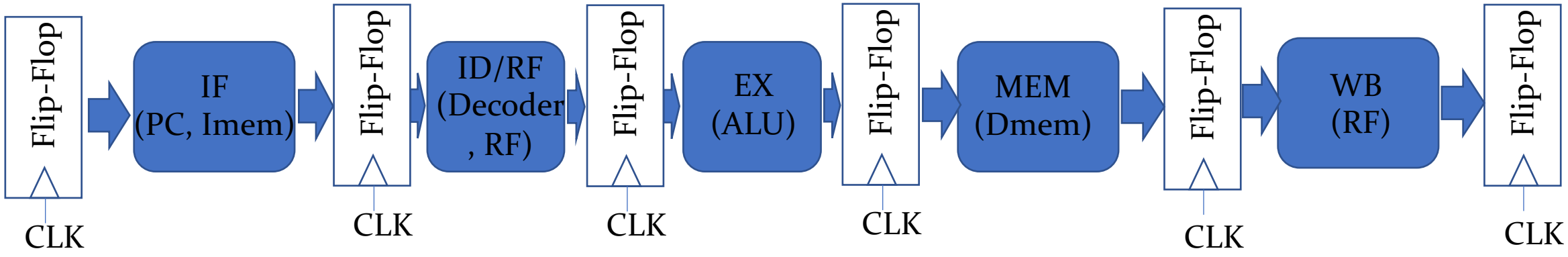


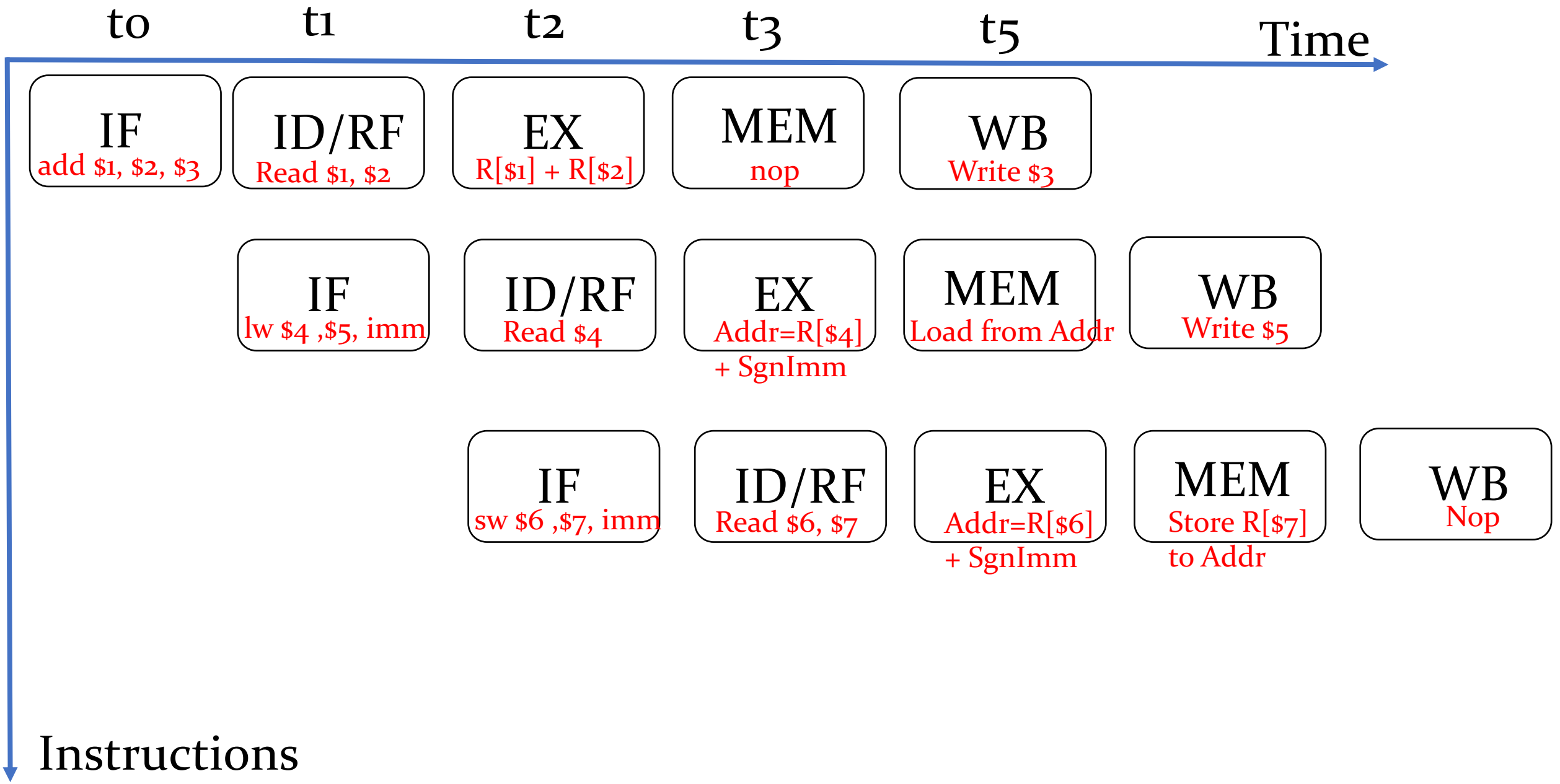
Muxes increase delay of EX stage

5 Stage Pipeline (with support for load/store)

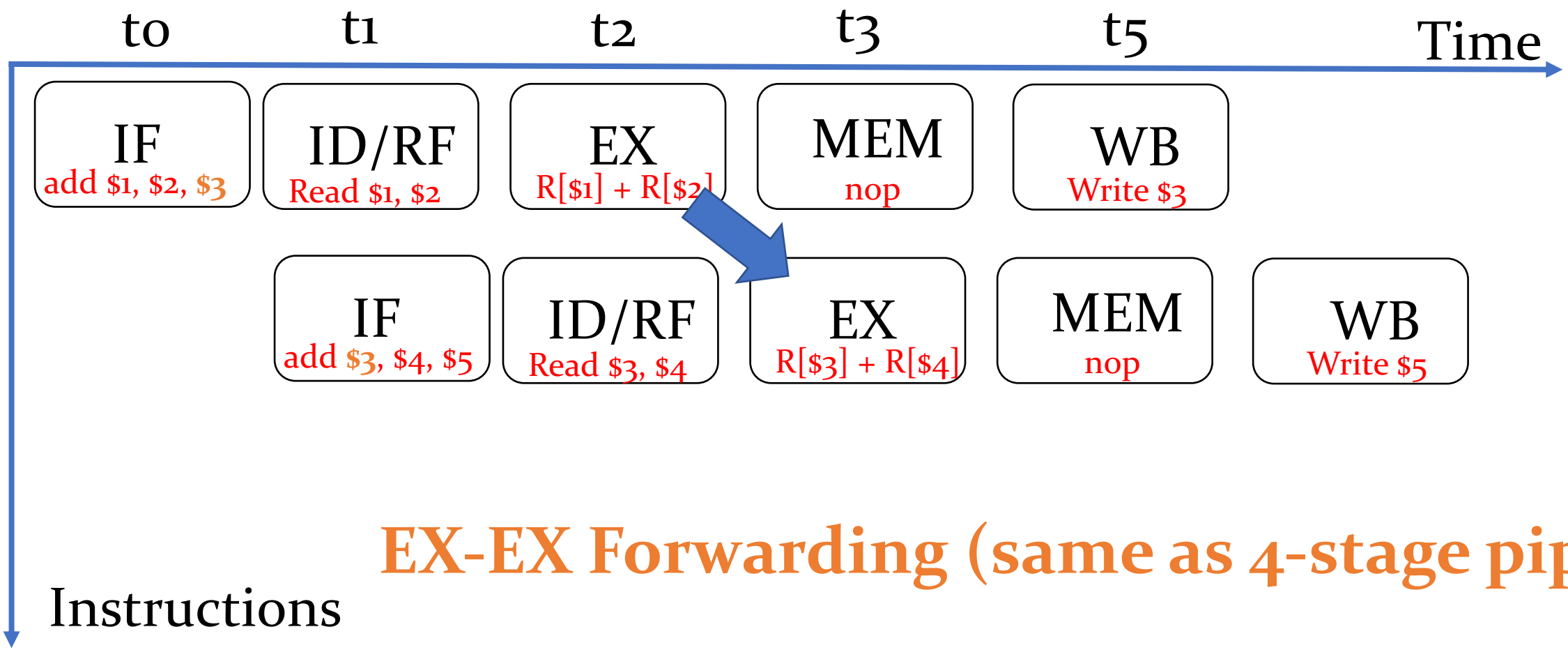
Basic steps in executing an R-type instruction

- Instruction fetch (IF)
- Instruction Decode/Register File Read (ID/RF)
- Execute in ALU (EX)
- Load result from or store result to data memory (MEM)
- Write-back Result to RF (WB)



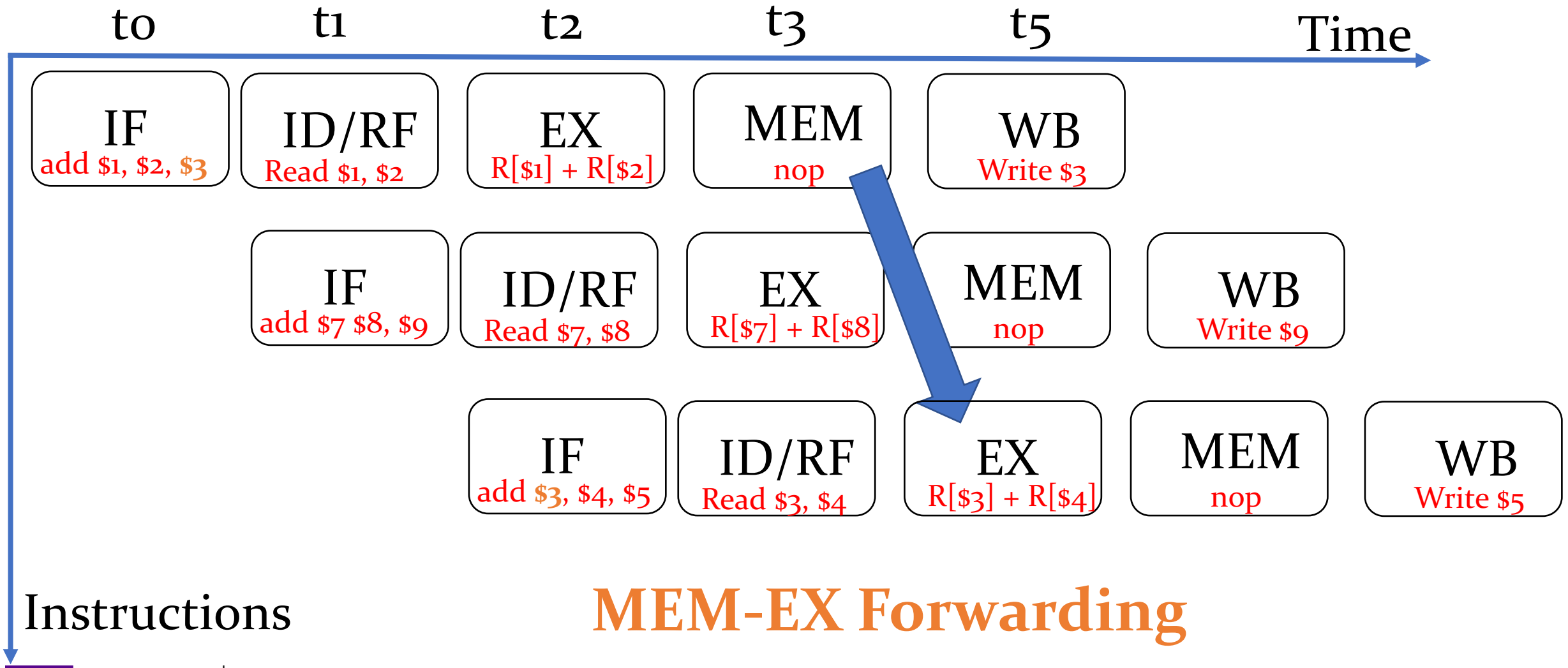

to
t1
t2
t3
t5
Time
IF
add $1, $2, $3
ID/RF
Read $1, $2
EX
R[$1] + R[$2]
MEM
nop
WB
Write $3
IF
lw $4 ,$5, imm
ID/RF
Read $4
EX
Addr=R[$4]
+ SgnImm
MEM
Load from Addr
WB
Write $5
IF
sw $6 ,$7, imm
ID/RF
Read $6, $7
EX
Addr=R[$6]
+ SgnImm
MEM
Store R[$7]
to Addr
WB
Nop
Instructions

5-stage RAW Hazards: Add-Add Dependency



EX-EX Forwarding (same as 4-stage pipeline)

RAW Hazards: Add-Add Dependency

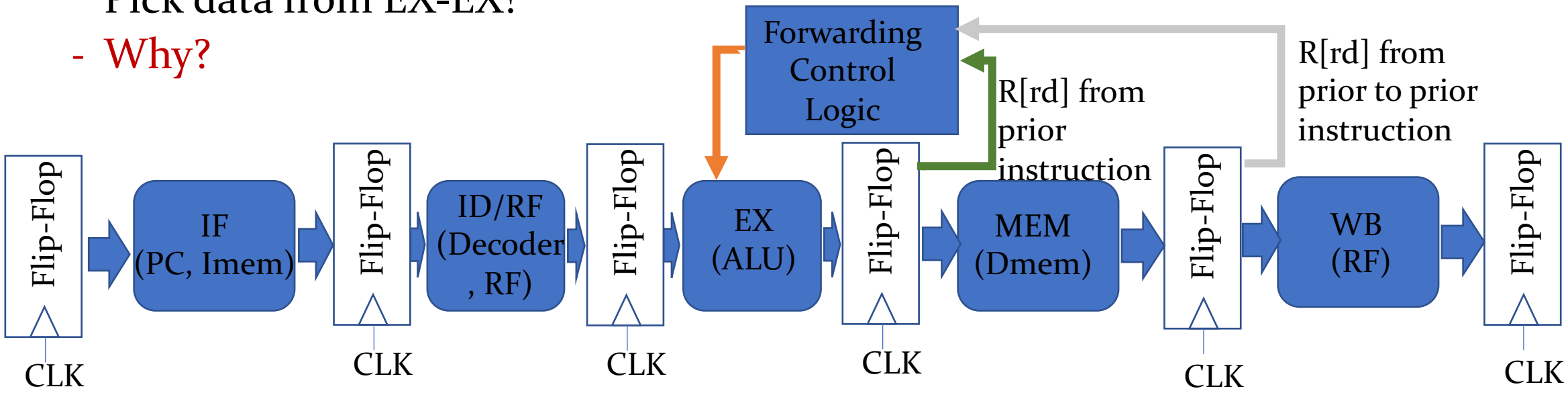


MEM-EX Forwarding

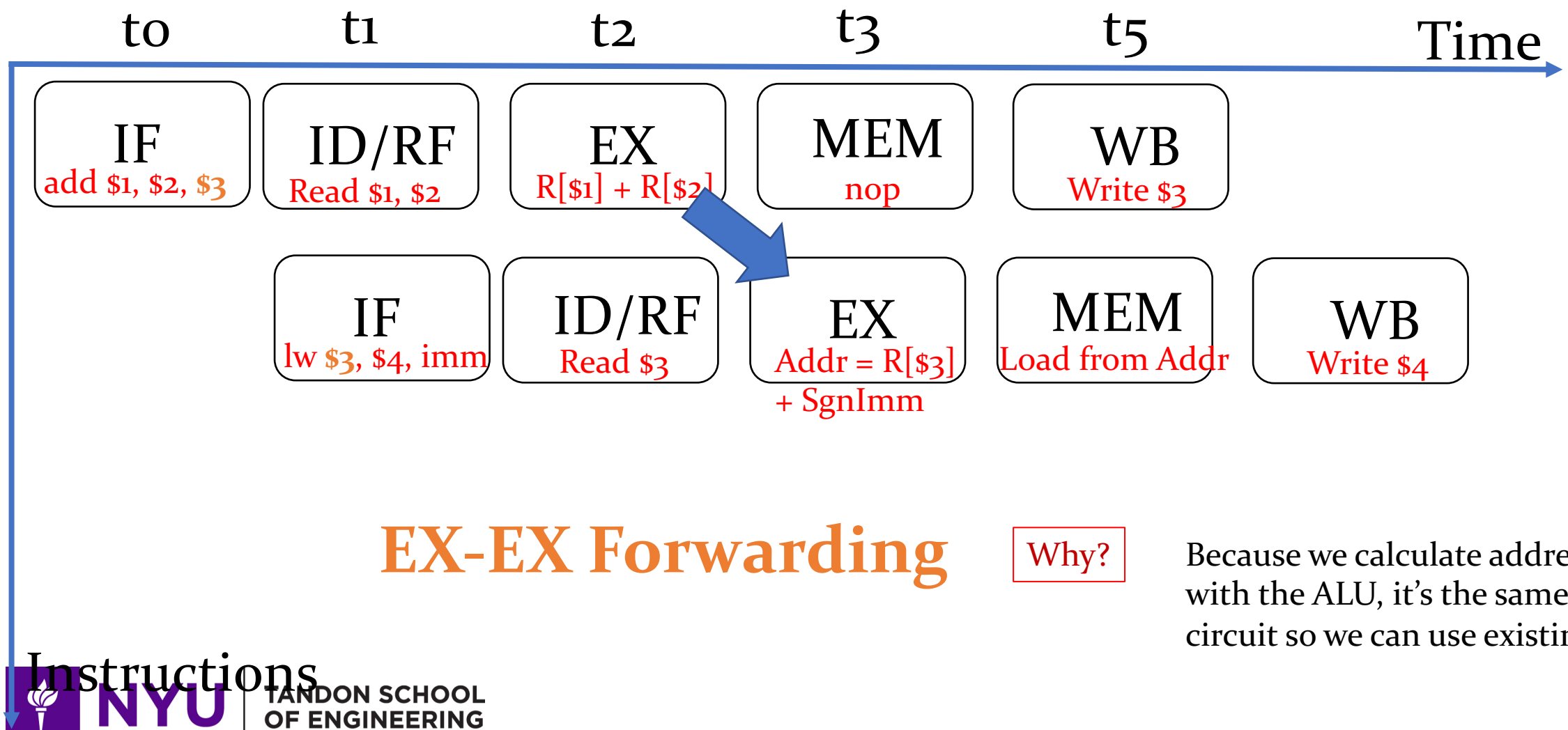
5 Stage Pipeline (with support for load/store)

Two forwarding paths

- EX – green
- MEM - gray
- Same destination register in both forwarding paths? Pick data from EX-EX!
- Why?



RAW Hazards: Add-Load Dependency

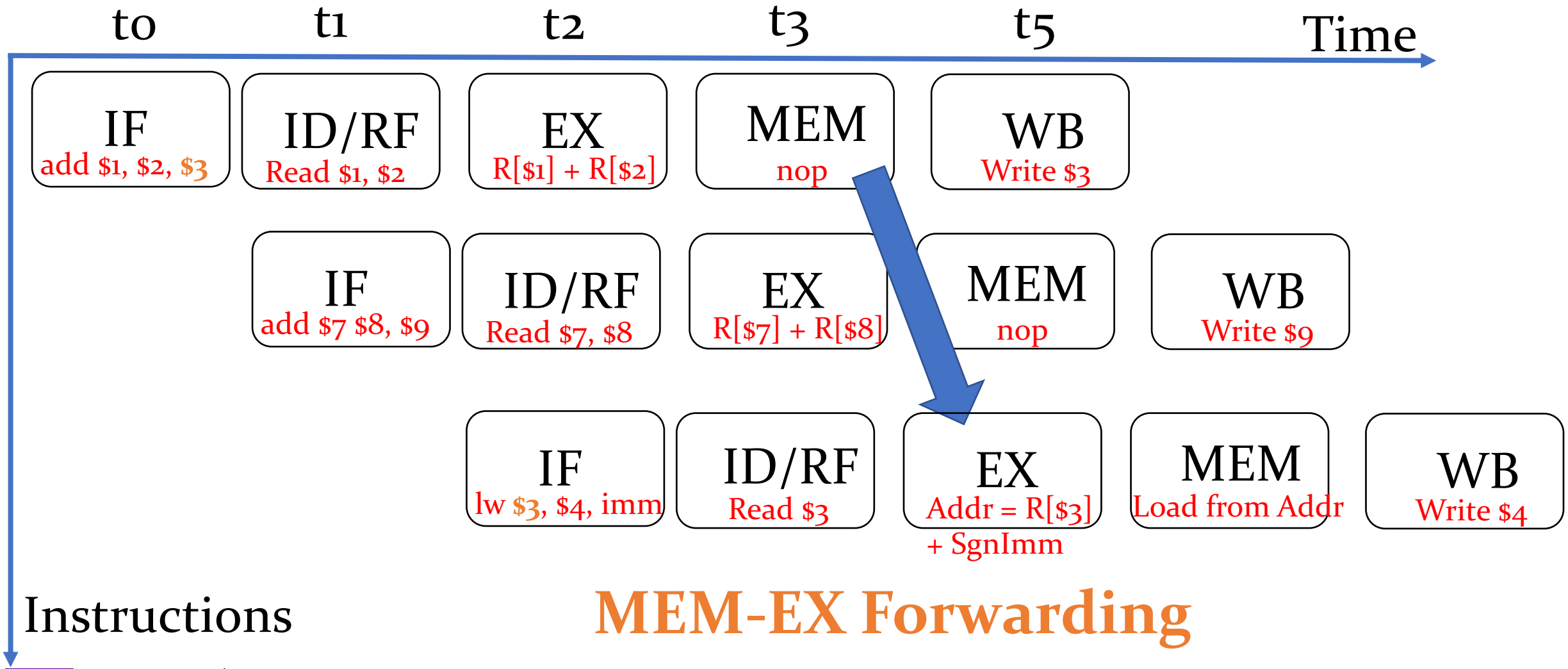


EX-EX Forwarding

Why?

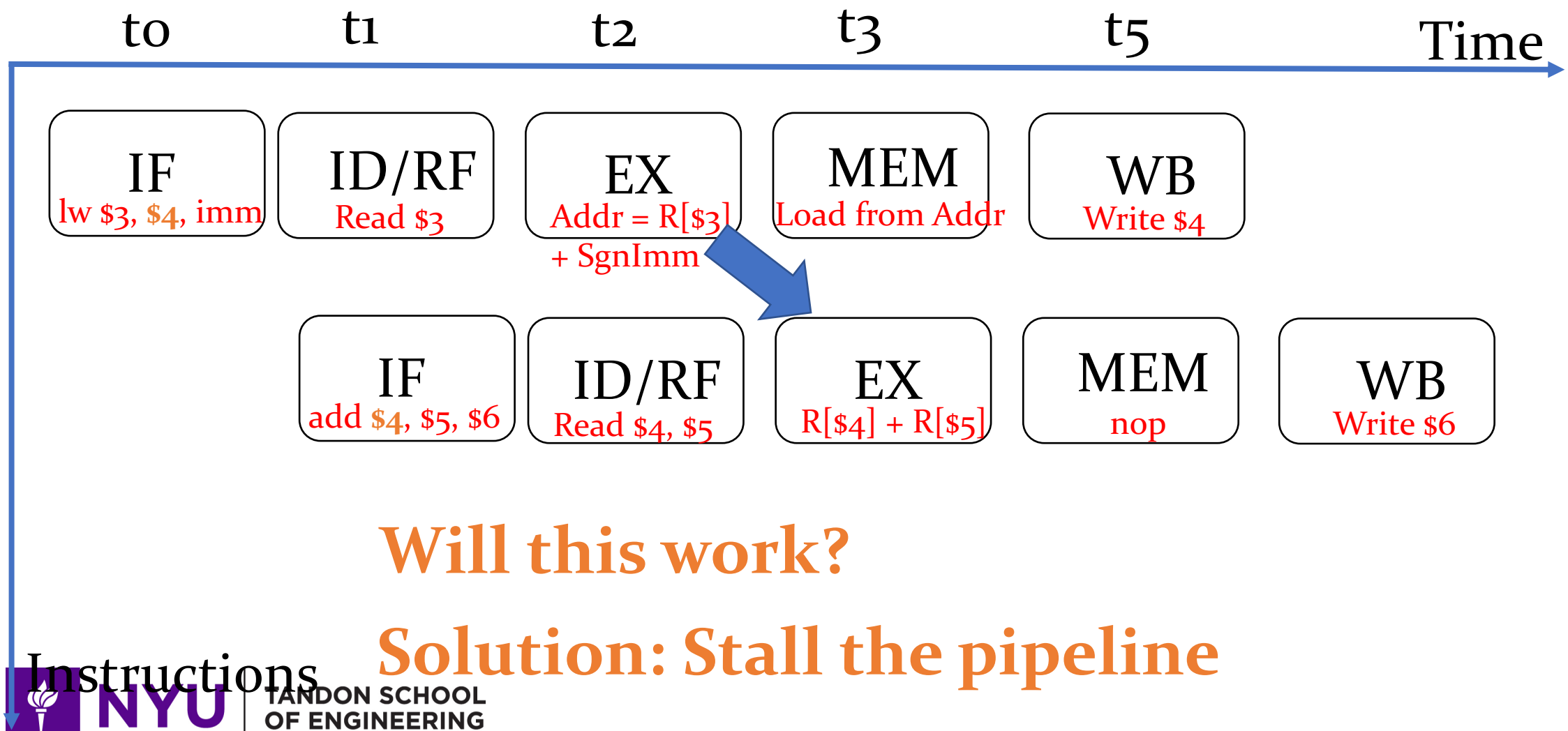
Because we calculate address with the ALU, it's the same circuit so we can use existing path

RAW Hazards: Add-Load Dependency



MEM-EX Forwarding

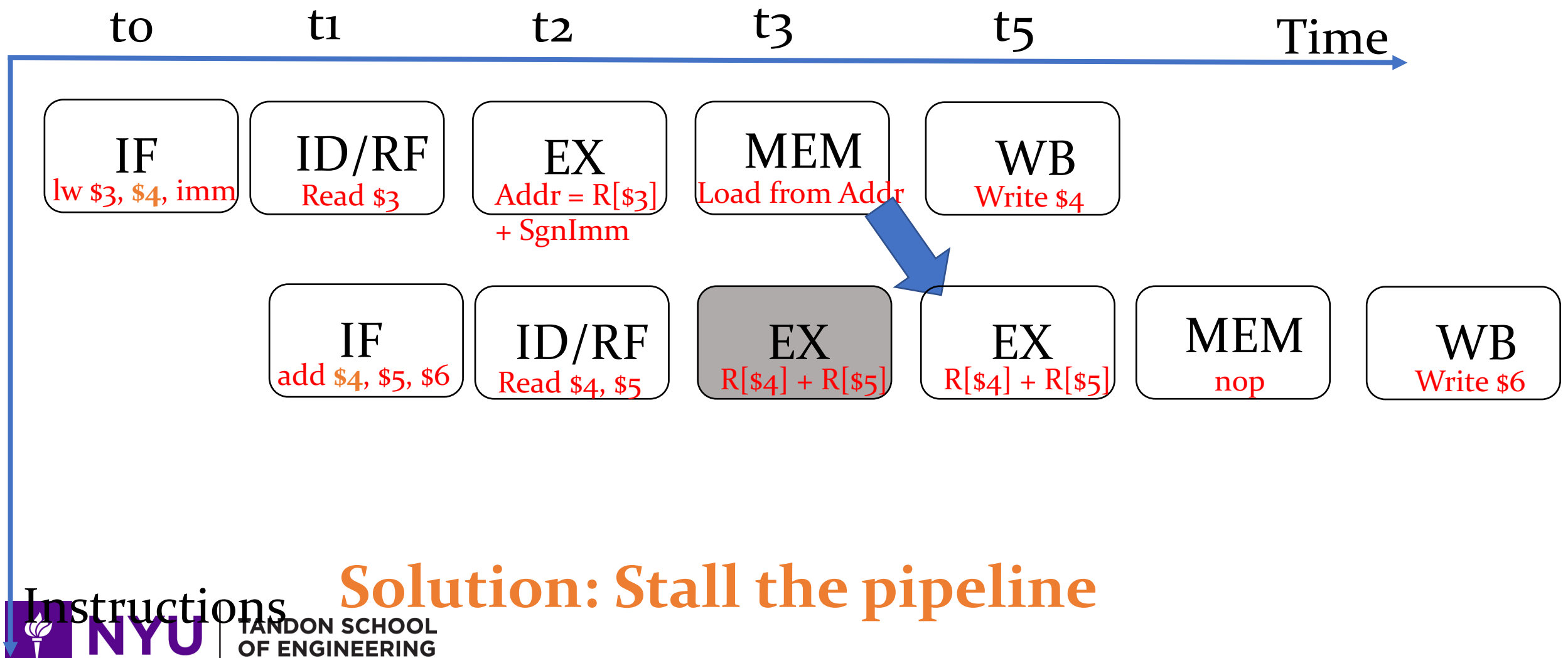
RAW Hazards: Load-Add Dependency



Will this work?

Solution: Stall the pipeline

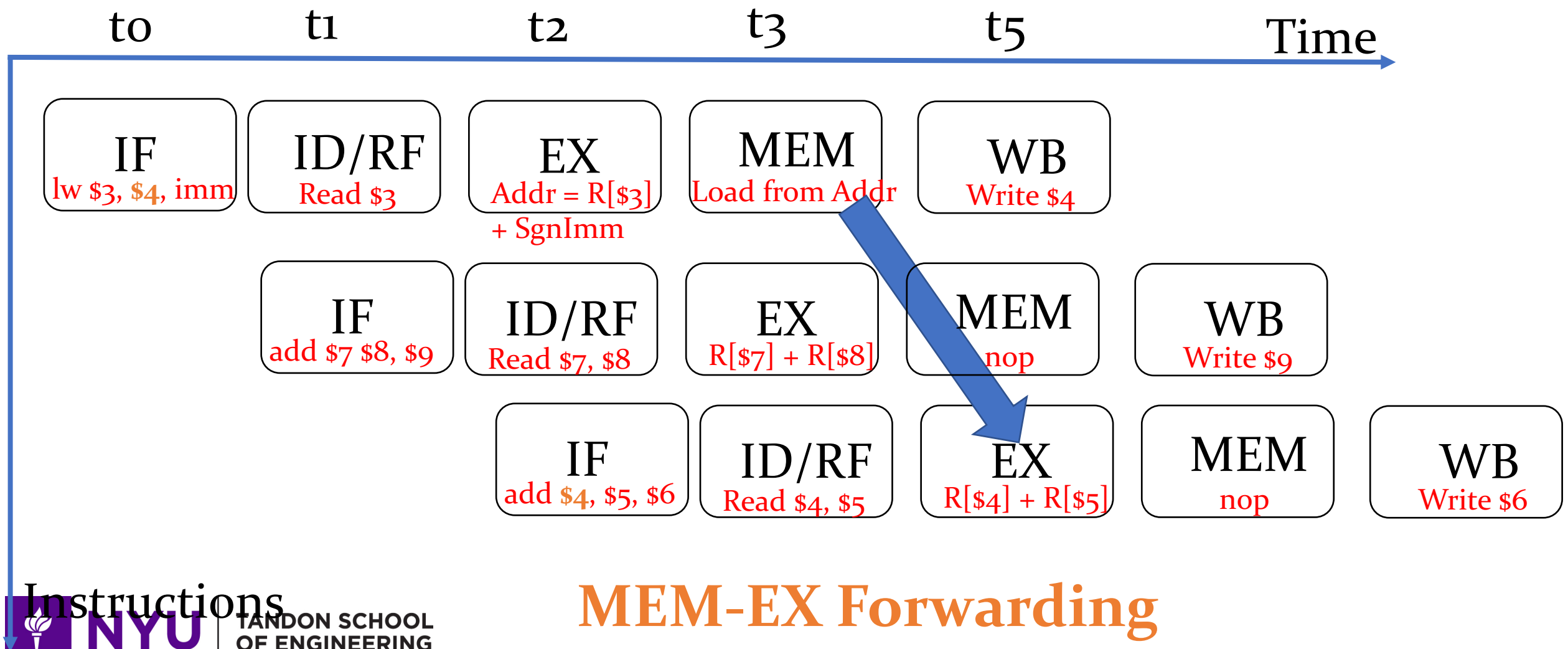
RAW Hazards: Load-Add Dependency



Solution: Stall the pipeline

RAW Hazards: Load-Add Dependency

SW Trick!
(They help us for once..)
Compilers save the day



MEM-EX Forwarding