Topic 10: Pipelining

COS / ELE 375

Computer Architecture and Organization

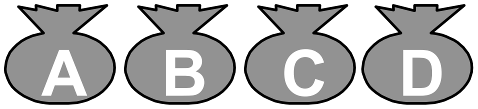
Princeton University
Fall 2015

Prof. David August

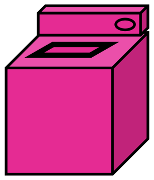
Pipelining is Natural: Assembly Line!

Laundry Example

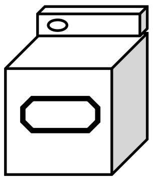
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold



- Washer takes 30 minutes

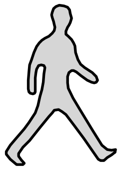


- Dryer takes 30 minutes

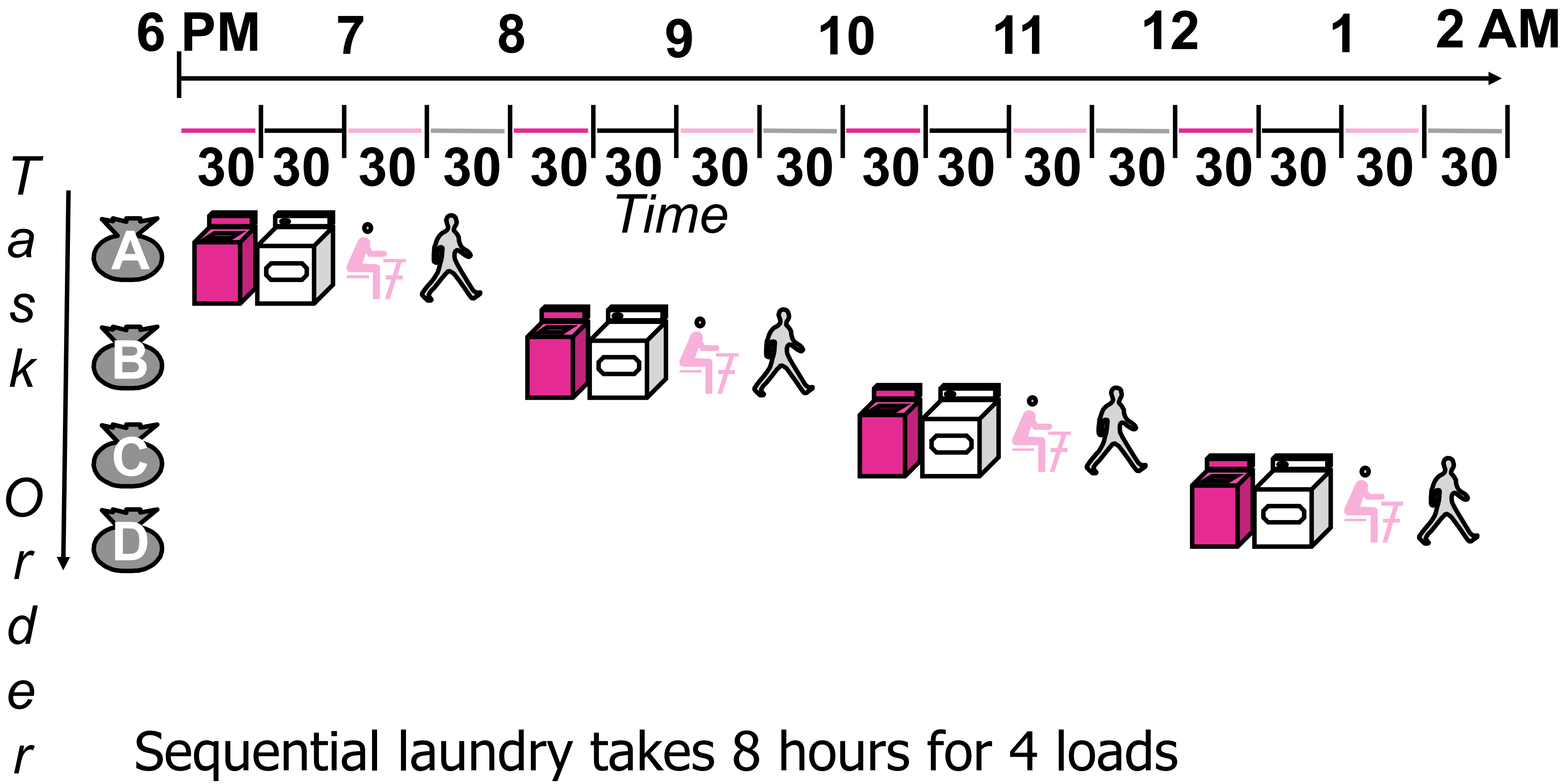


- “Folder” takes 30 minutes

- “Stasher” takes 30 minutes to put clothes into drawers



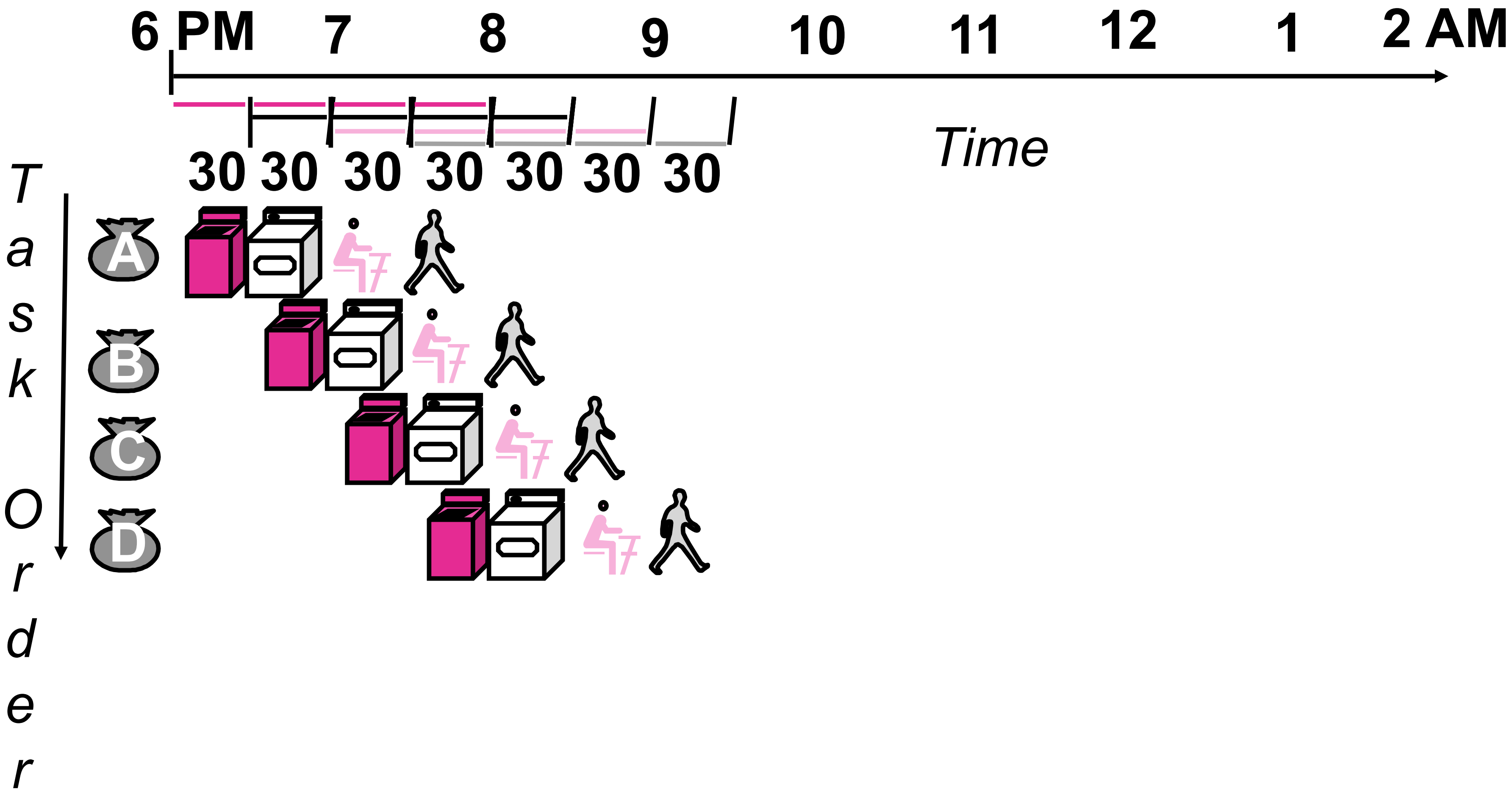
Sequential Laundry



Sequential laundry takes 8 hours for 4 loads

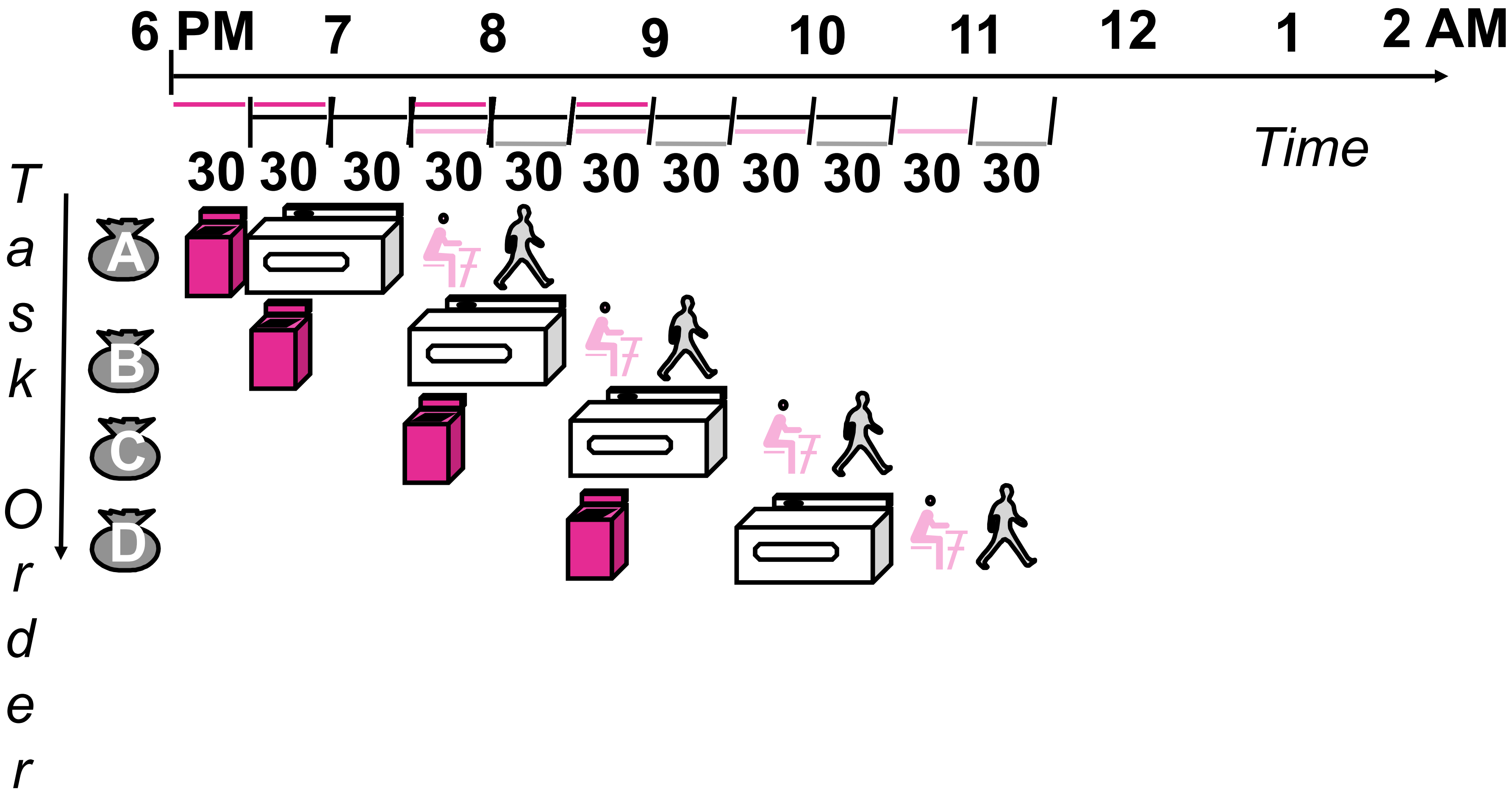
If they learned pipelining, how long would laundry take?

Pipelined Laundry: Start work ASAP



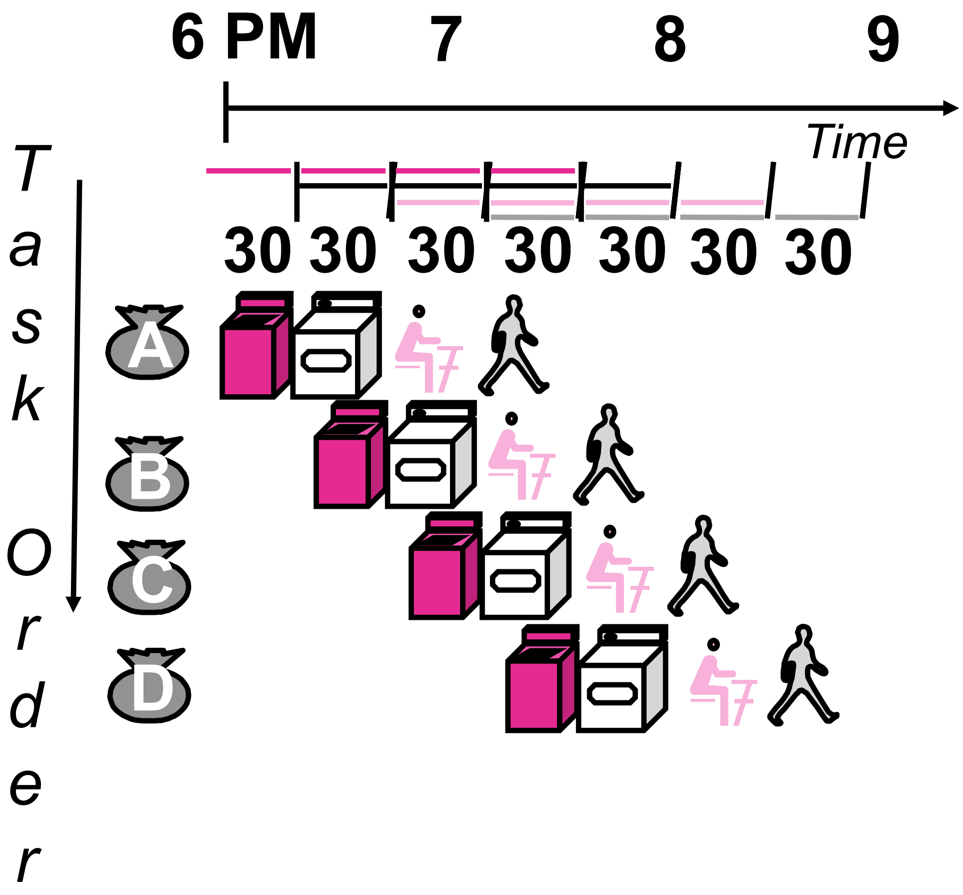
- Pipelined laundry takes 3.5 hours for 4 loads!

Slow Dryers



5.5 Hours. What is going on here?

Pipelining Lessons



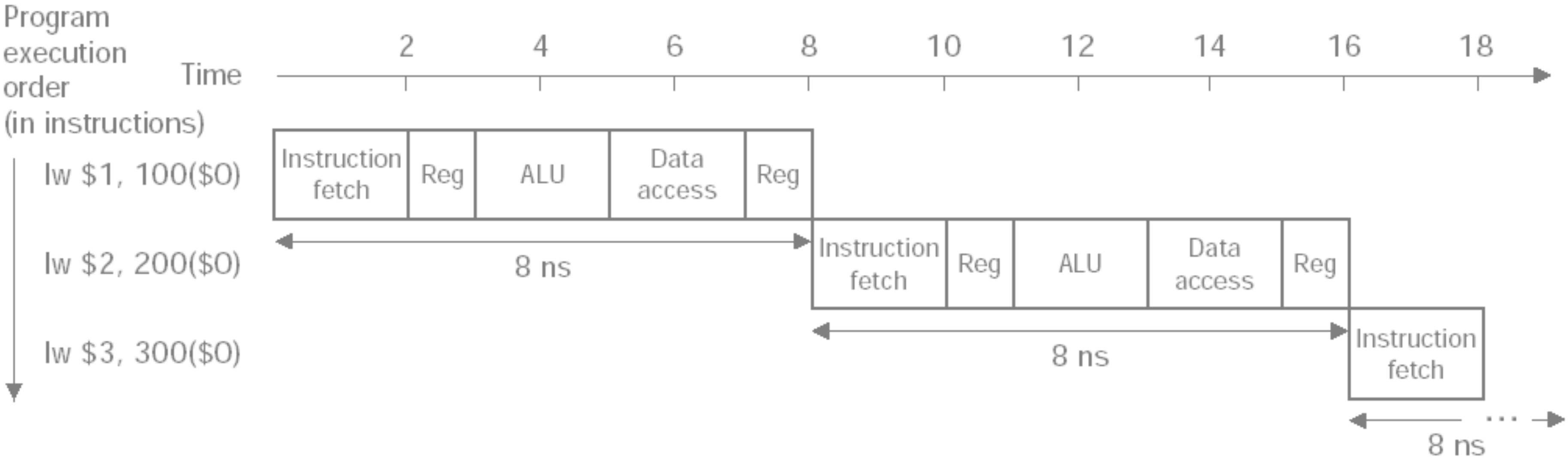
1. Pipelining doesn’t help latency of single task, it helps throughput of entire workload
2. Multiple tasks operate simultaneously using different resources
3. Potential speedup = Number pipe stages
4. Pipeline rate limited by slowest pipeline stage
5. Unbalanced lengths of pipe stages reduces speedup
6. Time to “fill” pipeline and time to “drain” it reduces speedup
7. Stall for Dependences

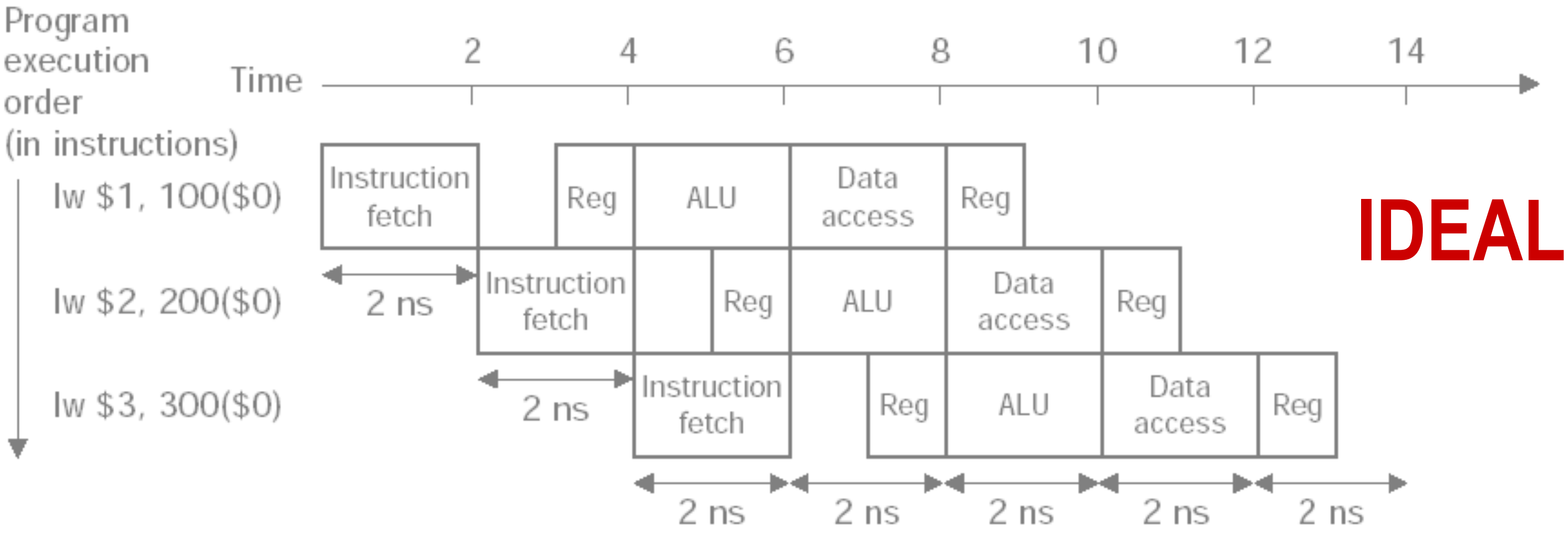
MIPS

Pipe Stages == The Five Execution Steps

1. Instruction Fetch
2. Instruction Decode and Register Fetch
3. Execution, Memory Address Computation, or Branch Completion
4. Memory Access or R-type instruction completion
5. Write-Back Step

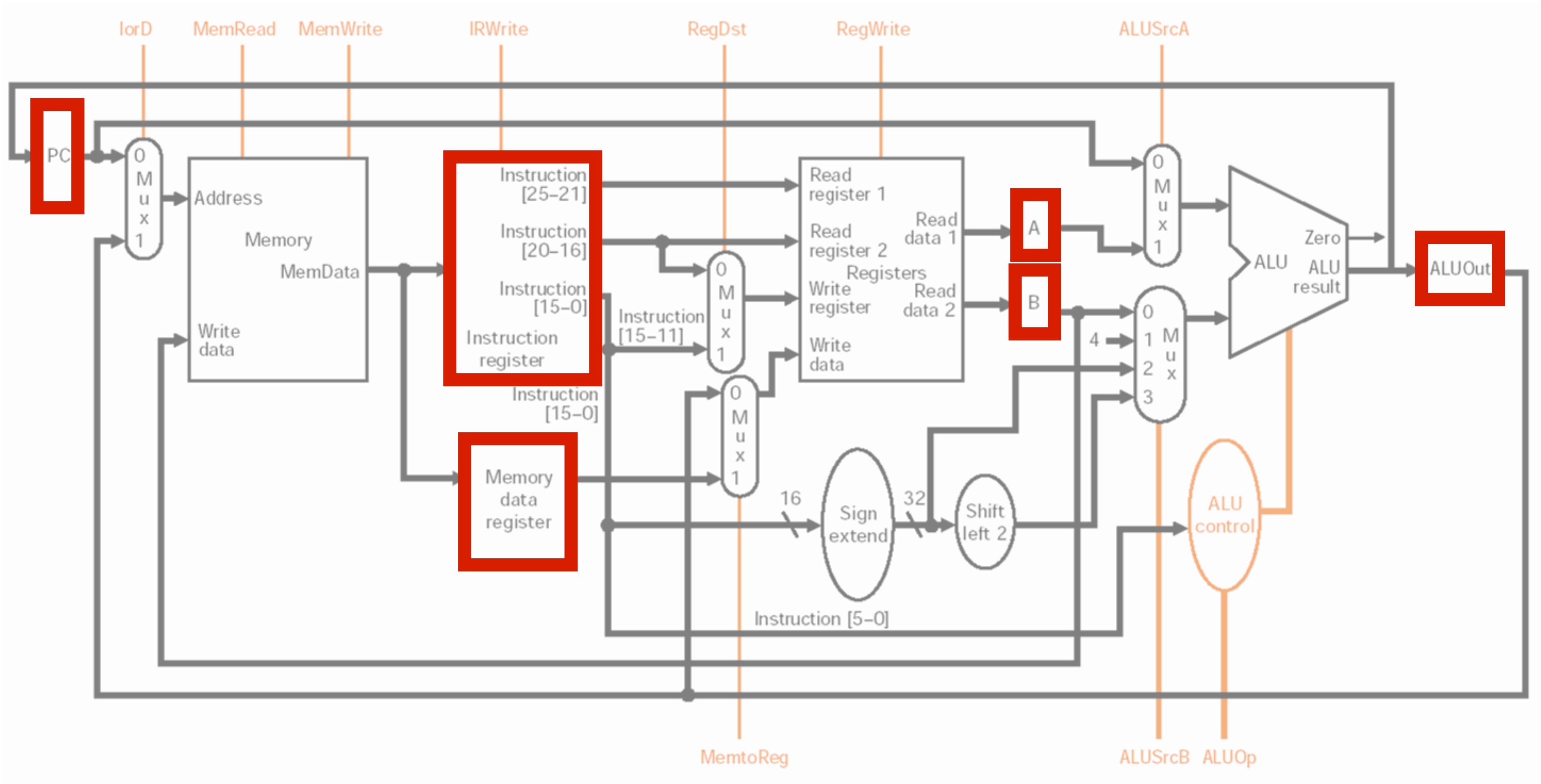
Pipelining in MIPS



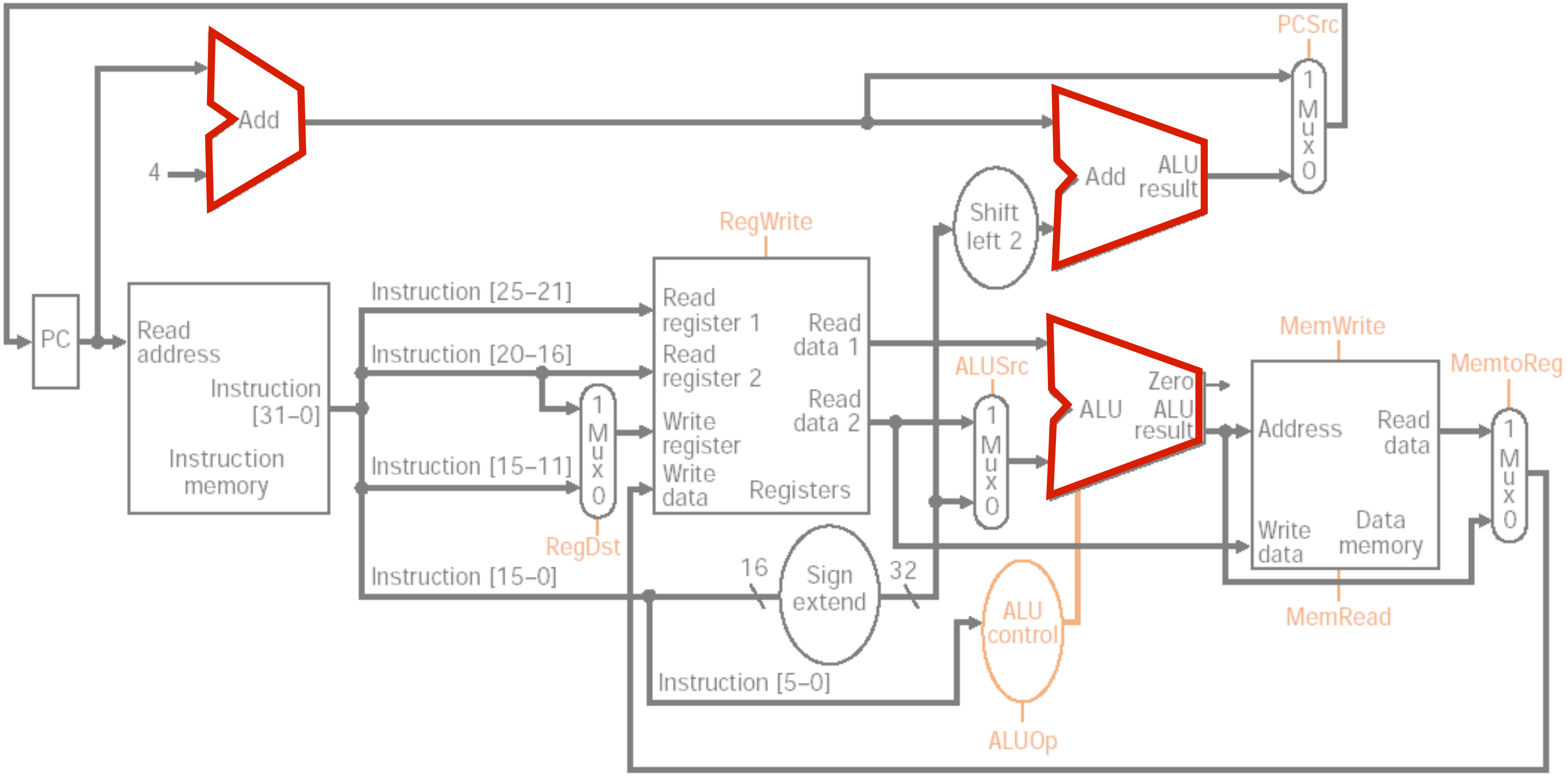


IDEAL?

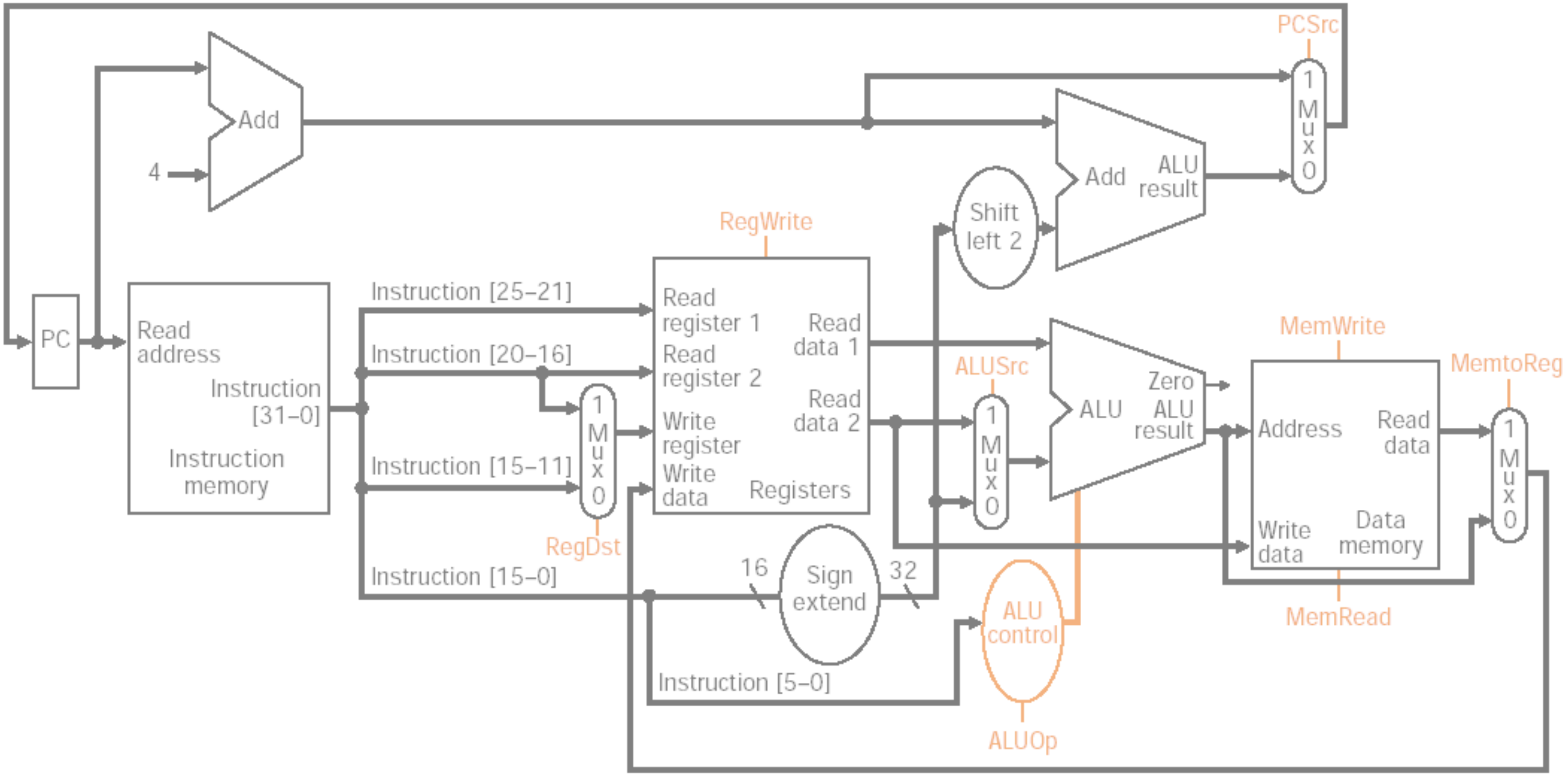
Can We Pipeline the Multicycle Datapath?



Can We Pipeline the Unicycle Datapath?

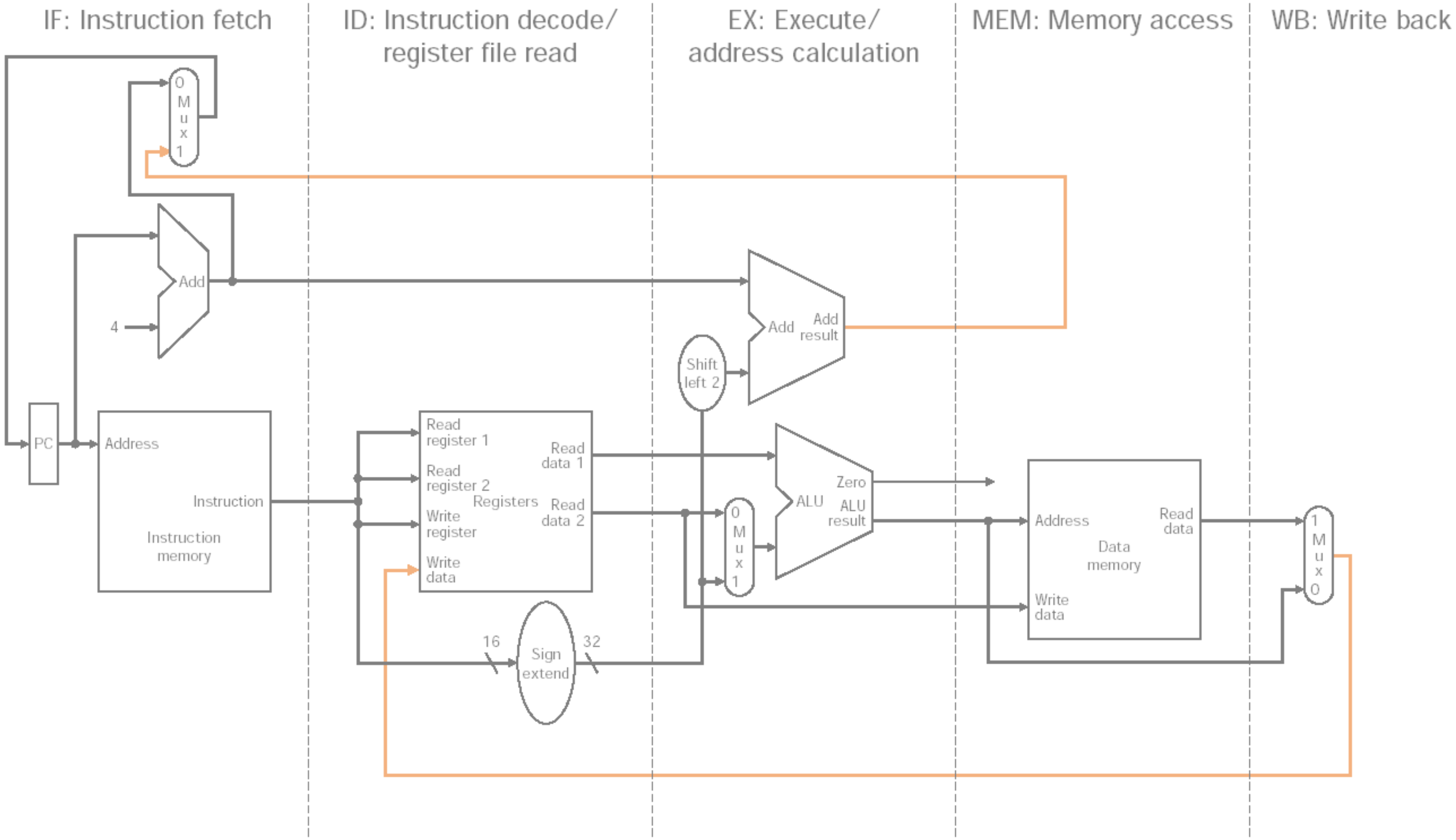


Unicycle



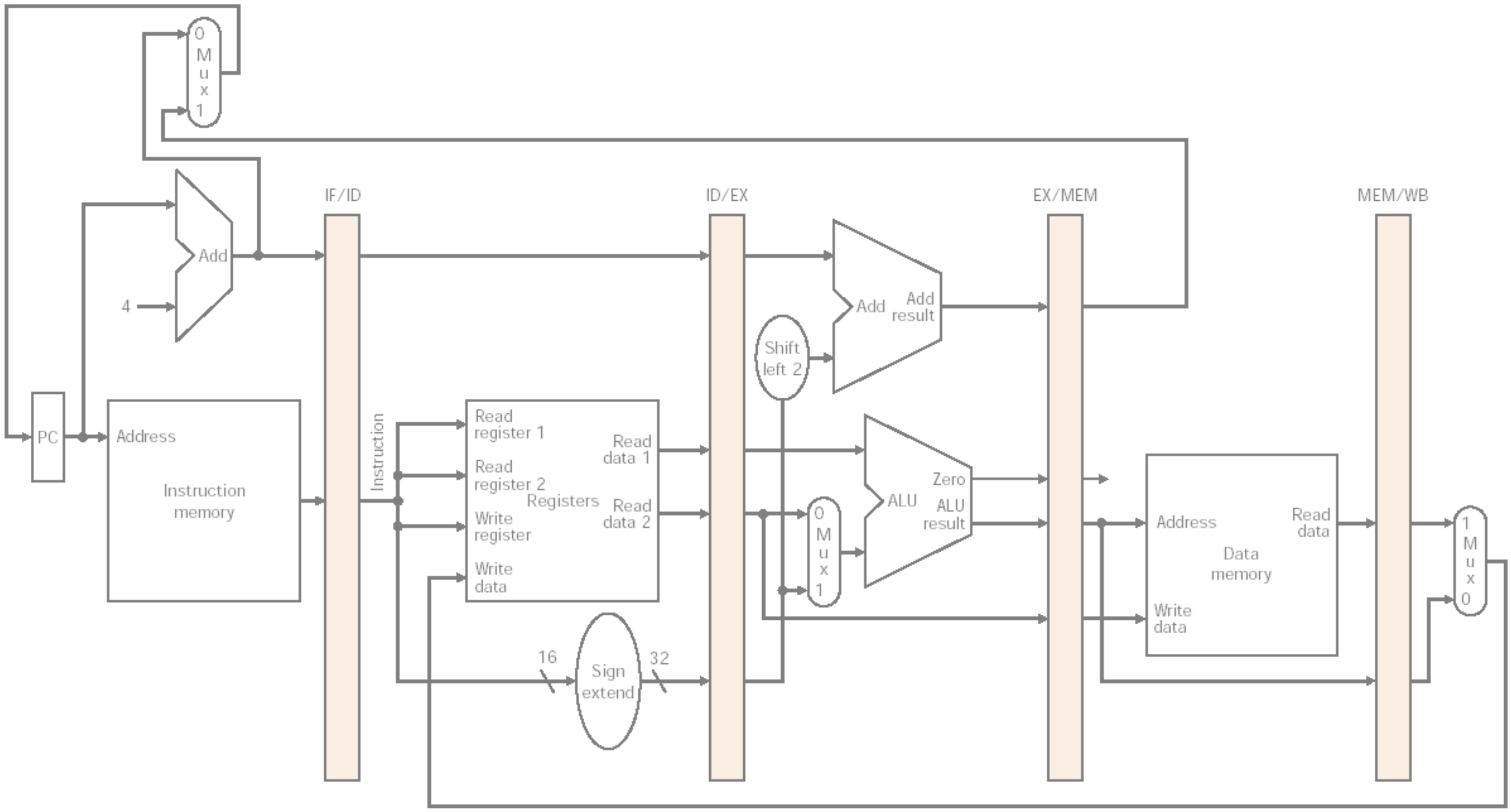
How do we split the datapath into stages?

Basic Idea


IF: Instruction fetch
ID: Instruction decode/ register file read
EX: Execute/ address calculation
MEM: Memory access
WB: Write back
0
M
u
x
1
Add
4
PC
Address
Instruction
Instruction memory
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
16
Sign extend
32
Shift left 2
Add
Add result
0
M
u
x
1
ALU
Zero
ALU result
Address
Data memory
Write data
Read data
1
M
u
x
0

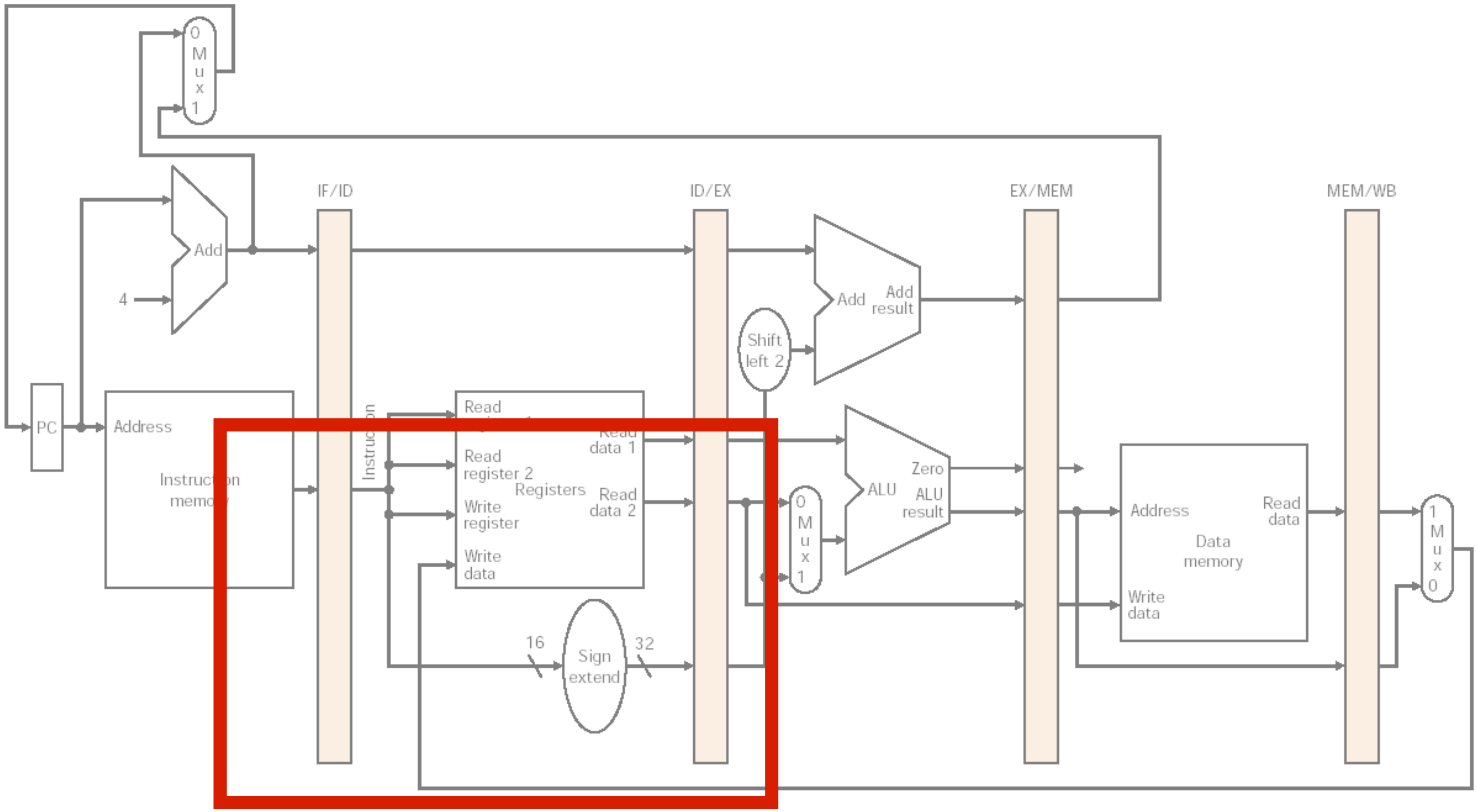
Slicing of Datapath

Rectangles are pipeline registers

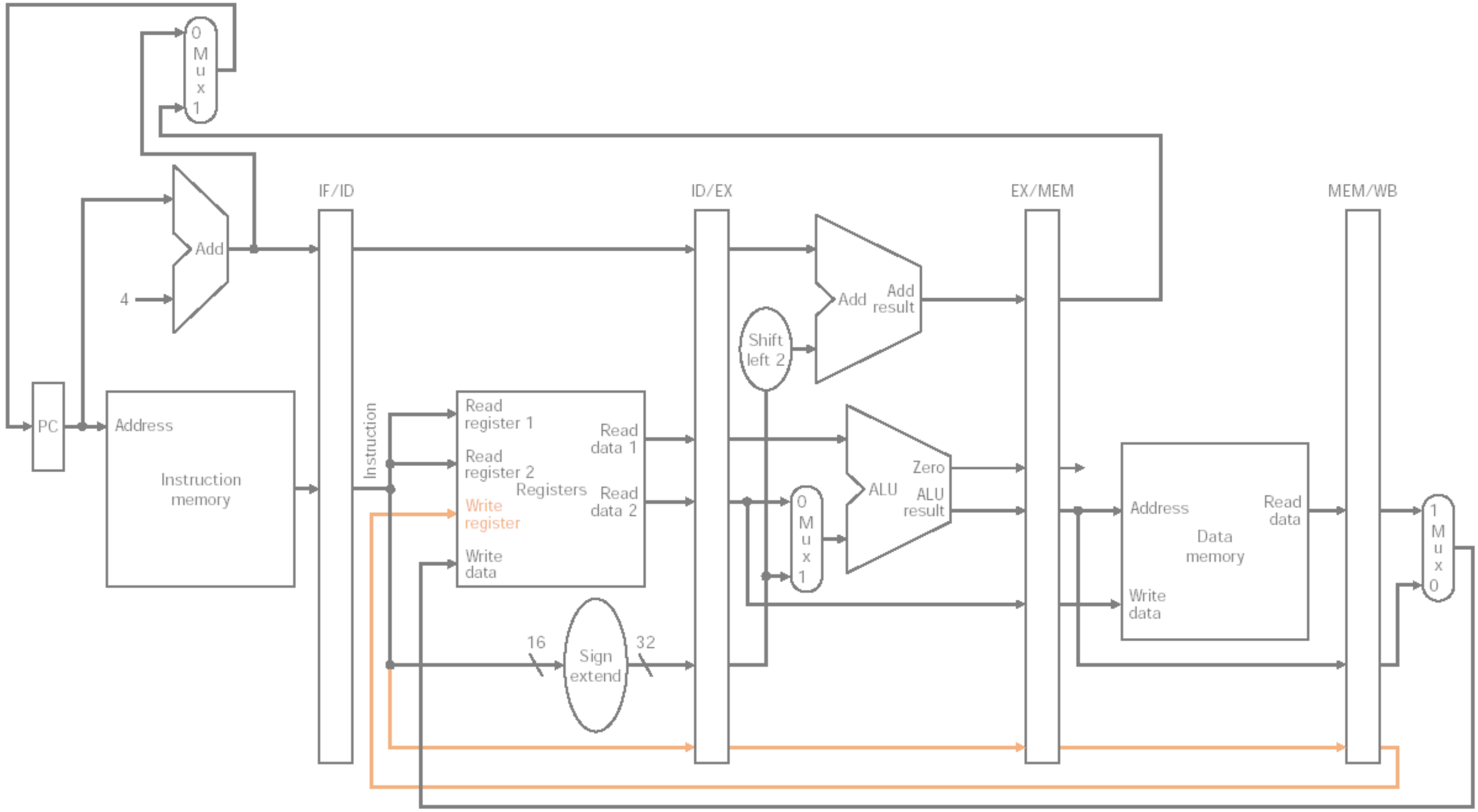


Slicing of Datapath

Anything wrong in this picture?



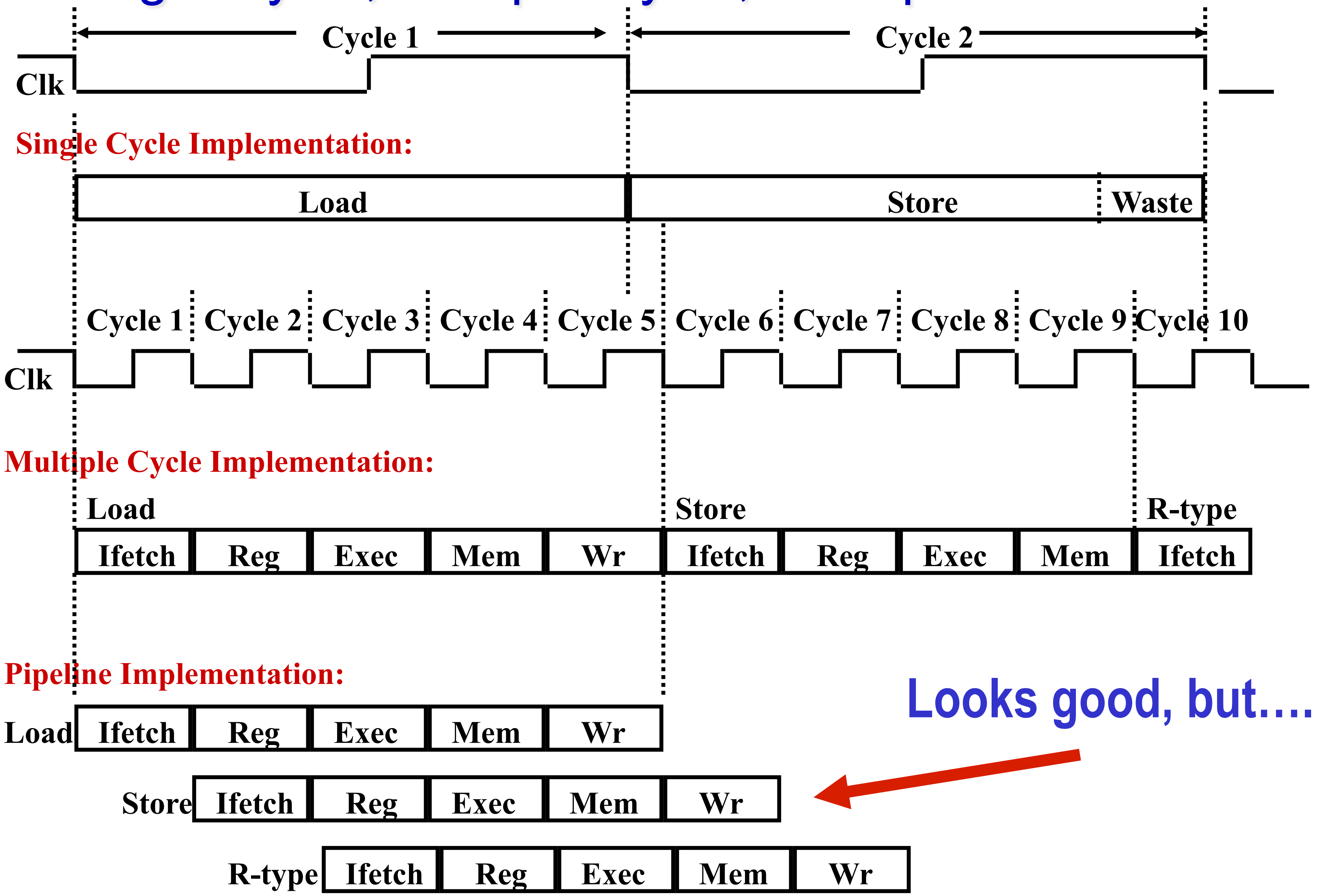
Corrected Datapath



Other(?) Control Signals?

Another View:

Single Cycle, Multiple Cycle, vs. Pipeline



Cycle 1 Cycle 2

Clk

Single Cycle Implementation:

Load Store Waste

Cycle 1 Cycle 2 Cycle 3 Cycle 4 Cycle 5 Cycle 6 Cycle 7 Cycle 8 Cycle 9 Cycle 10

Clk

Multiple Cycle Implementation:

Load Store R-type

Ifetch Reg Exec Mem Wr Ifetch Reg Exec Mem Ifetch

Pipeline Implementation:

Load Ifetch Reg Exec Mem Wr

Store Ifetch Reg Exec Mem Wr

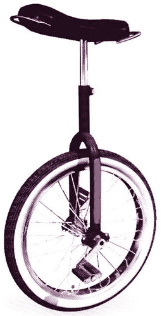
R-type Ifetch Reg Exec Mem Wr

Looks good, but….

Performance?
(Is it worth the pain?)

Unicycle Machine

45 ns/cycle x 1 CPI x 100 inst = 4500 ns



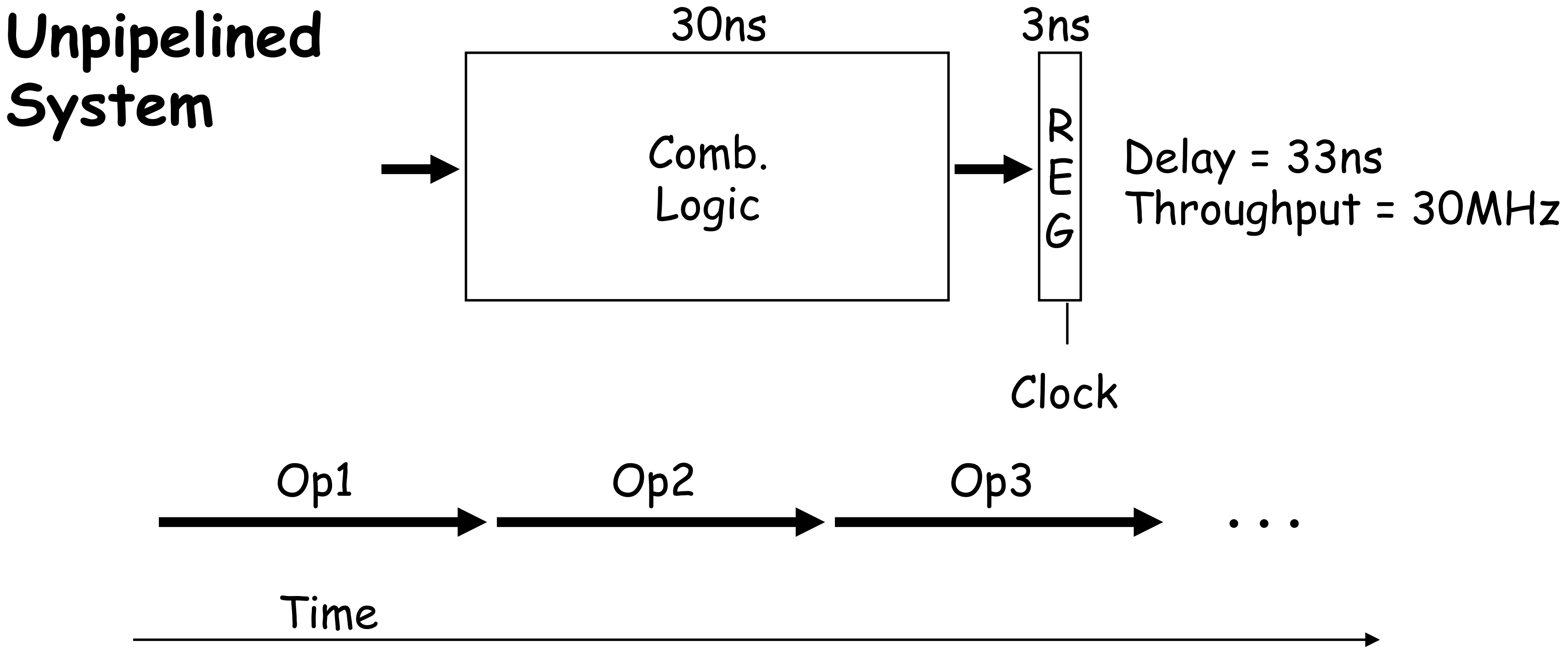
Multicycle Machine

10 ns/cycle x 4.6 CPI (inst mix) x 100 inst = 4600 ns

Ideal pipelined machine with 5 pipeline stages

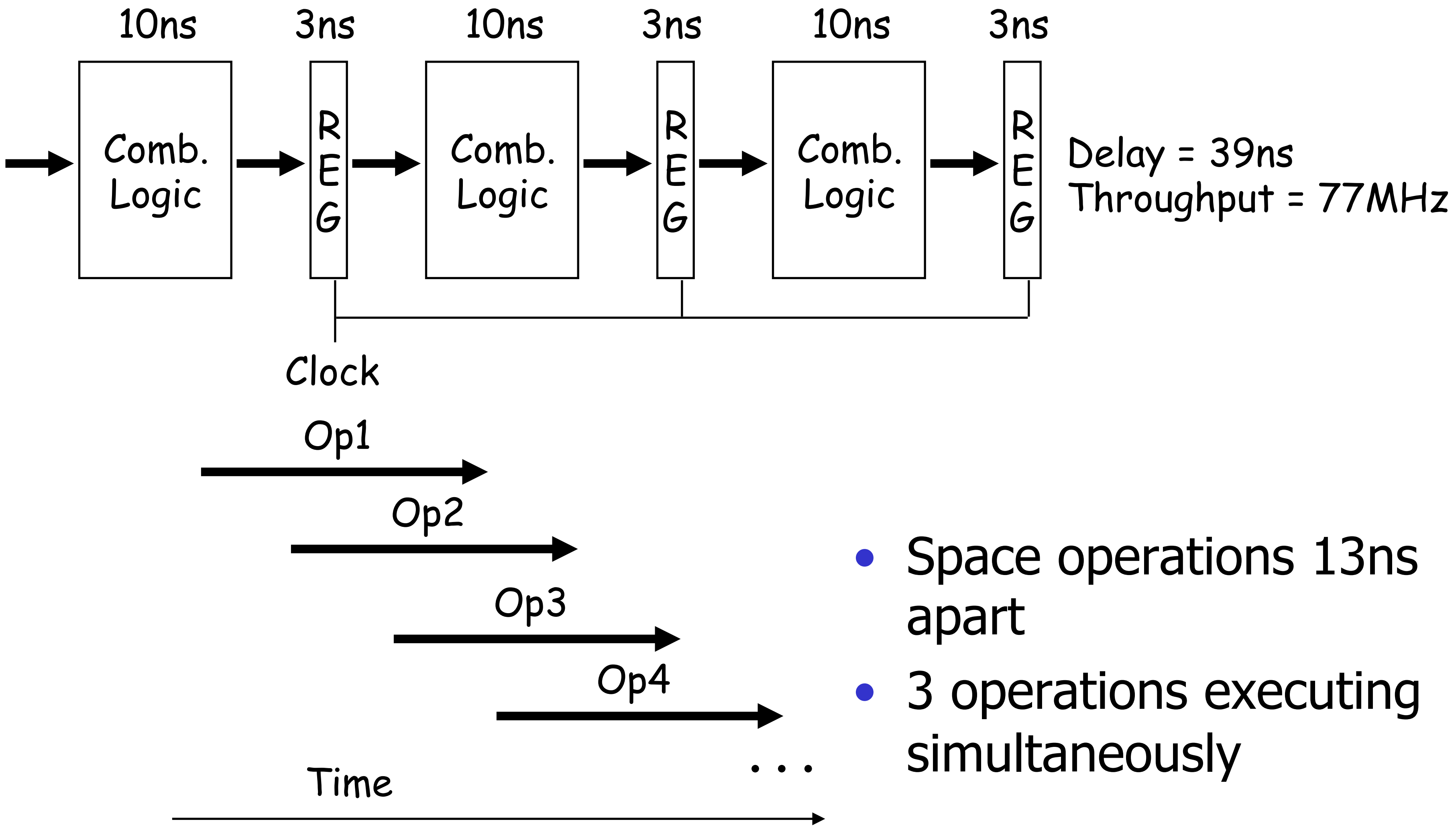
10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns

Unicycle Implementation Detail



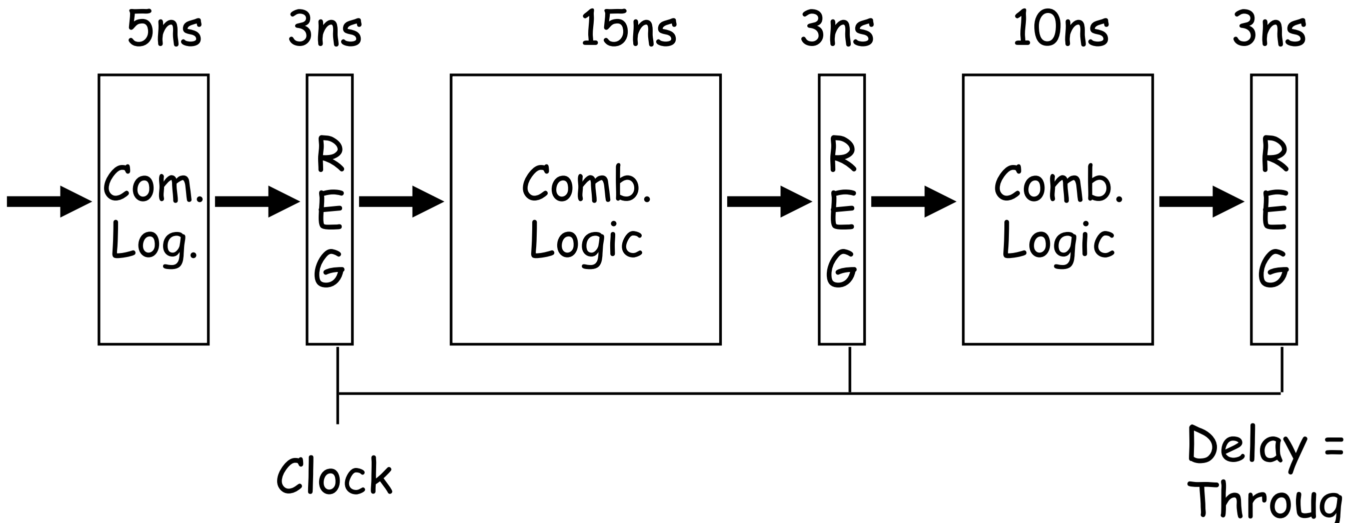
- One operation must complete before next can begin
- Operations spaced 33ns apart

3 Stage Pipeline Implementation Detail



- Space operations 13ns apart
- 3 operations executing simultaneously

Limitation 1: Nonuniform Pipelining

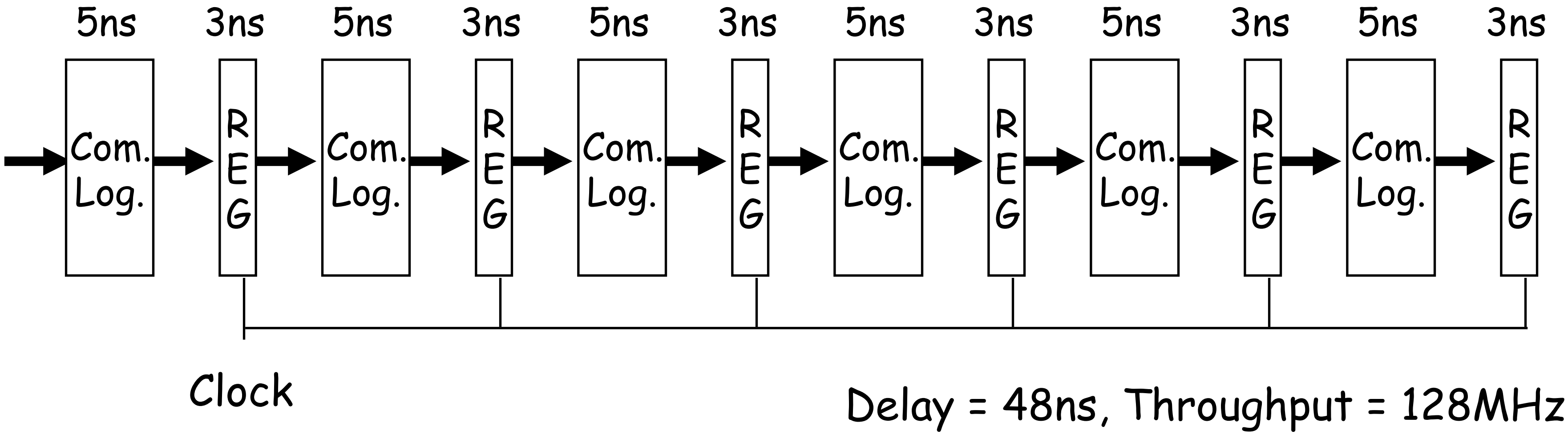


Delay = 18 * 3 = 54 ns
Throughput = 55MHz

- Throughput limited by slowest stage
  Delay determined by clock period * number of stages

- Must attempt to balance stages

Limitation 2: Deep Pipelines



- Diminishing returns as we add more pipeline stages
- Register delays become limiting factor
  - Increased latency
  - Small throughput gains

Unfortunately, there are other complications…

Pipeline Hazards

Next instruction cannot immediately follow previous instruction in the presence of a hazard.

Three types: Structural, Control, Data

Structural Hazards

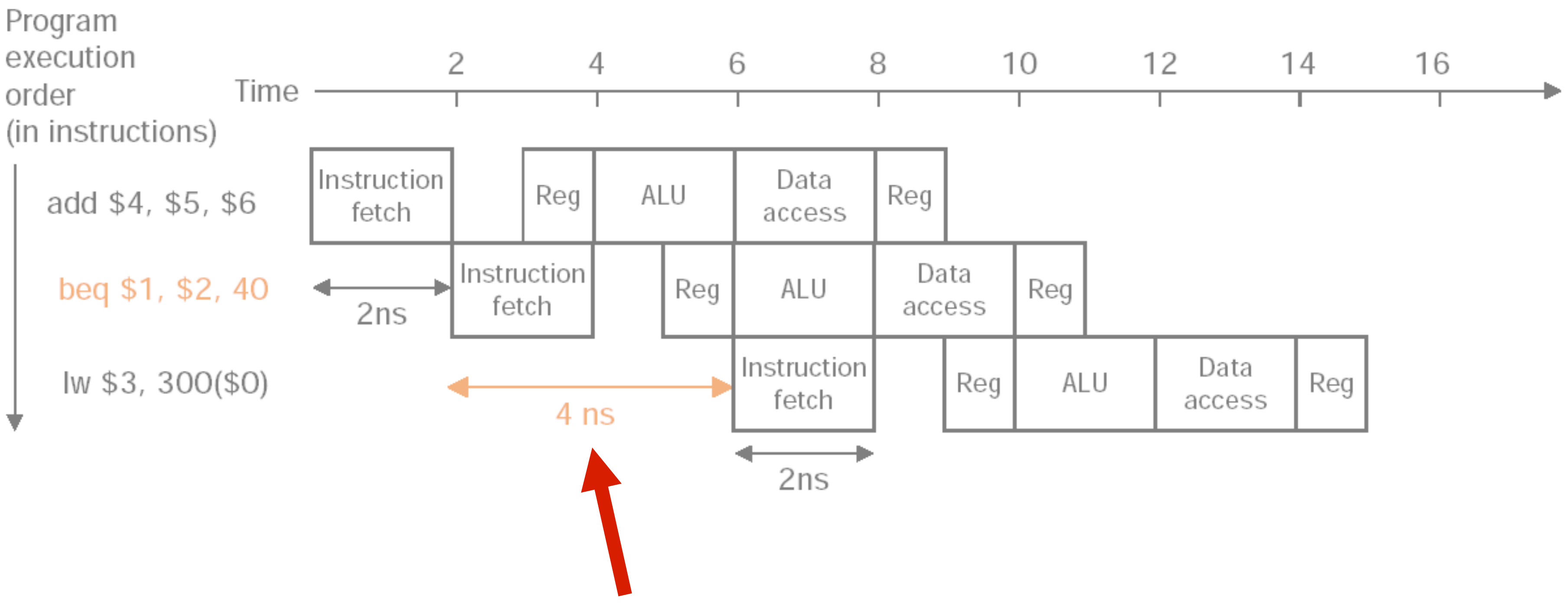
- Resource oversubscription
- Suppose we had only one memory
- In laundry, think of a washer/dryer combo unit

Pipeline Hazards

Control Hazards

- What is the next instruction?
- Branch instructions take time to compute this.

Solution 1: Stall



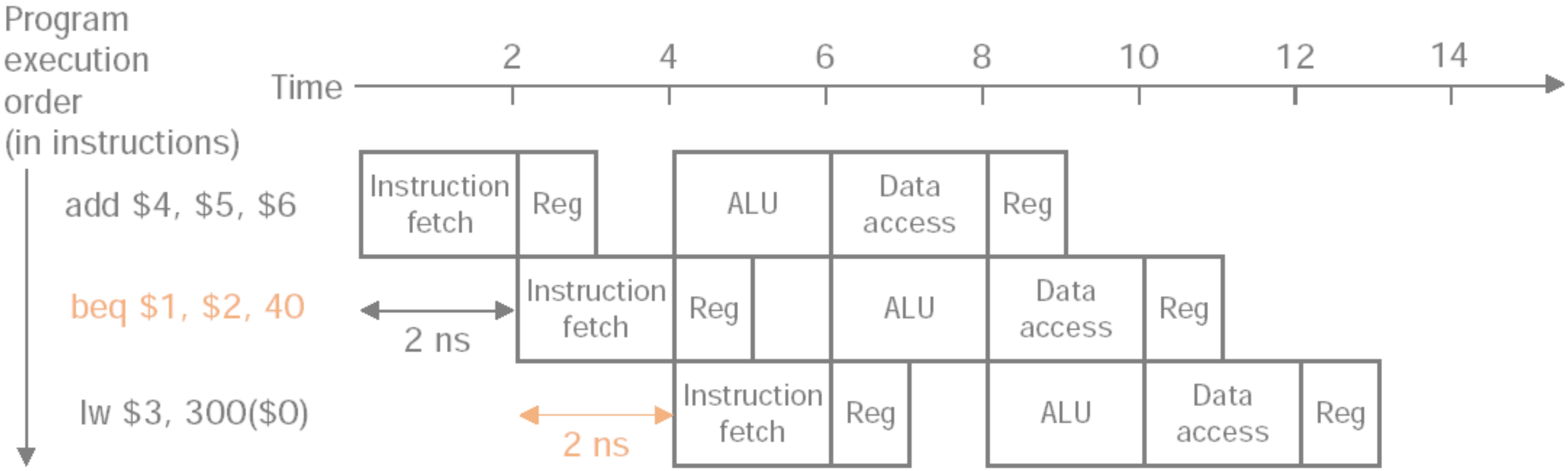
Pipeline Stall (AKA Bubble)

Pipeline Hazards

Control Hazards

- What is the next instruction?
- Branch instructions take time to compute this.

Solution 2: Predict the Branch Target

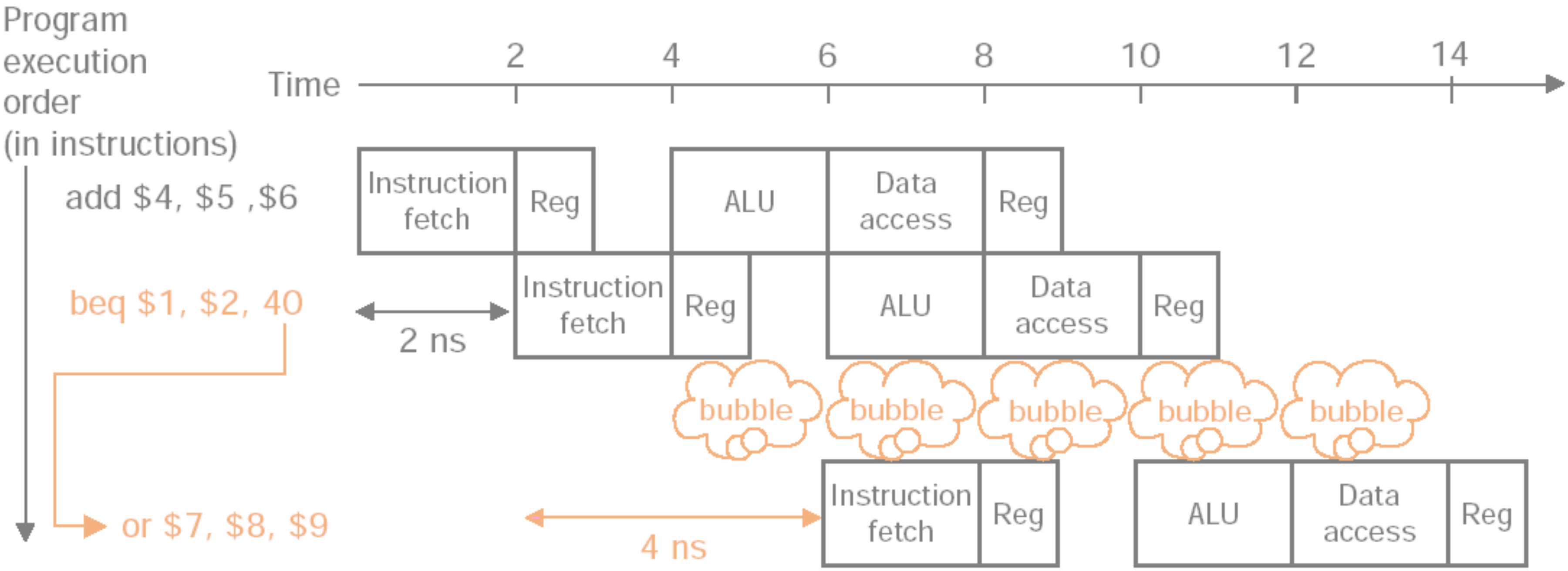


Pipeline Hazards

Control Hazards

- What is the next instruction?
- Branch instructions take time to compute this.

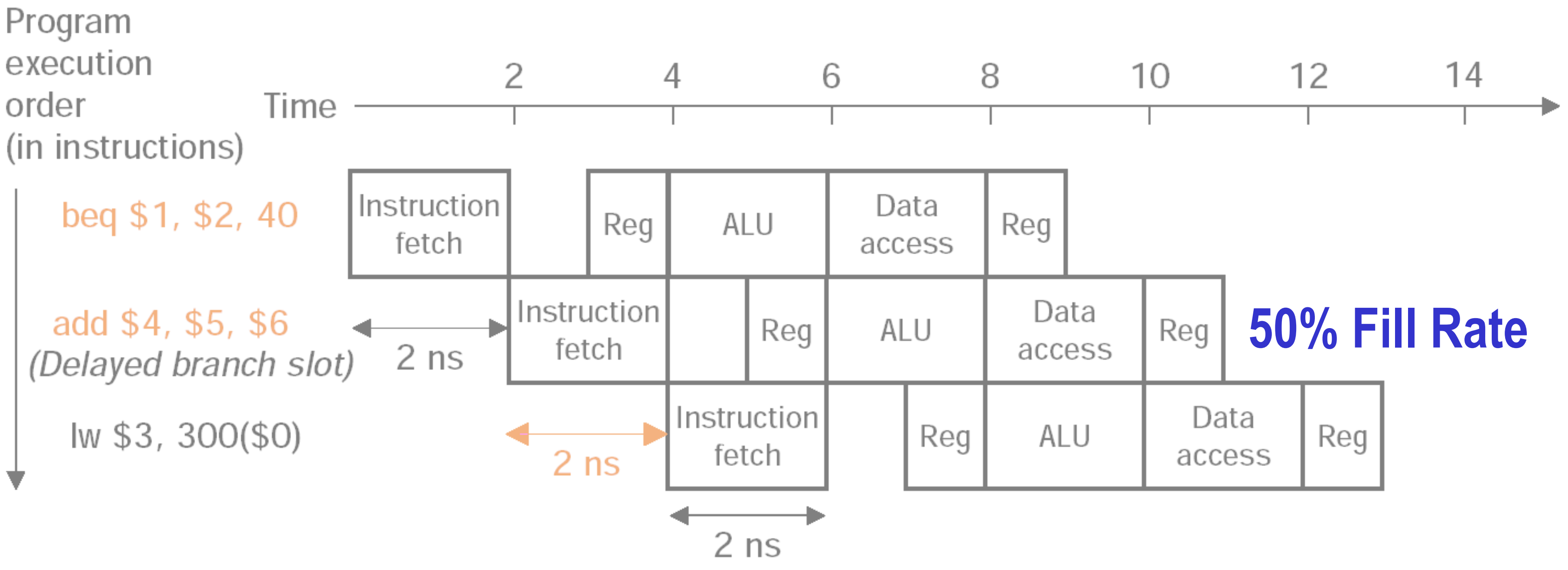
Solution 2: (Mis)Predict the Branch Target



Pipeline Hazards
Control Hazards

- What is the next instruction?
- Branch instructions take time to compute this.

Solution 3: Delayed Decision (Used in MIPS)



50% Fill Rate

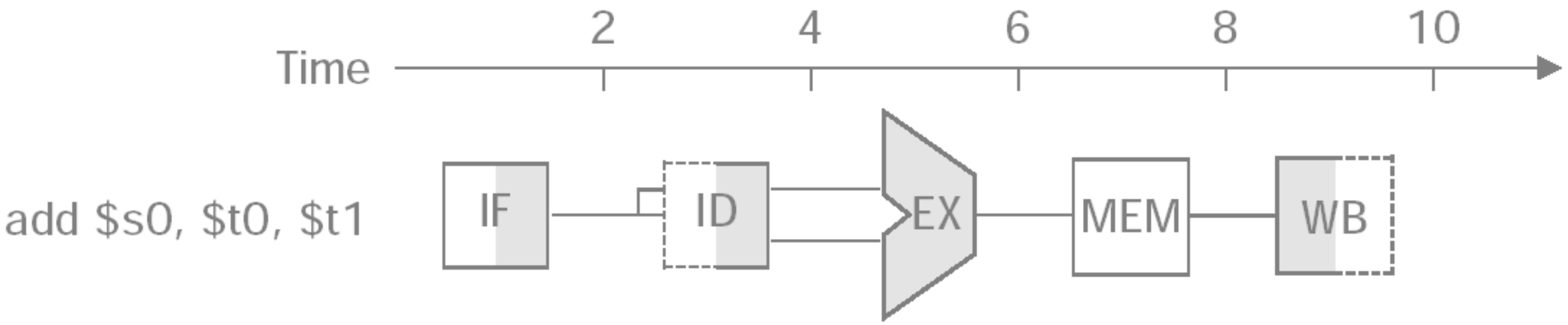
More about Branch Prediction/Delayed Branching Later…

Pipeline Hazards

Data Hazards

Value from prior instruction is needed before write back

Typical Instruction (new representation):



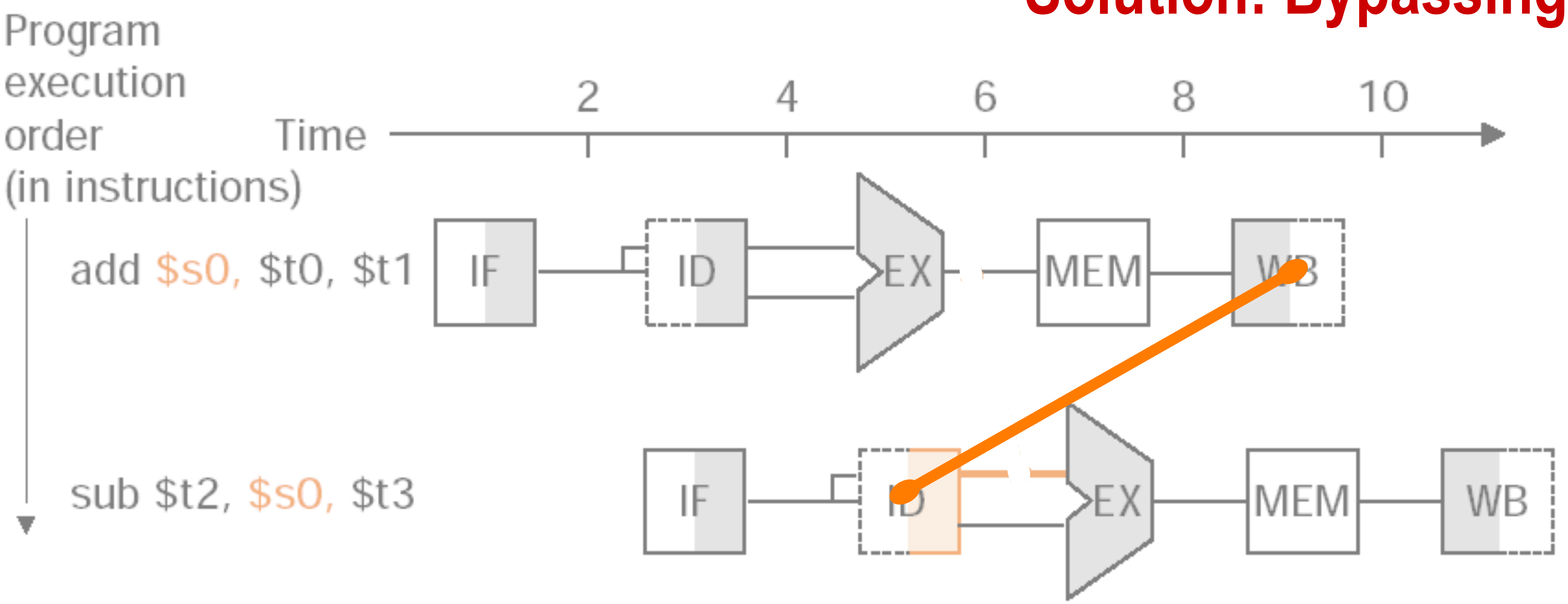
Pipeline Hazards

Data Hazards

Value from prior instruction is needed before write back

Data Hazard:

Solution: Bypassing

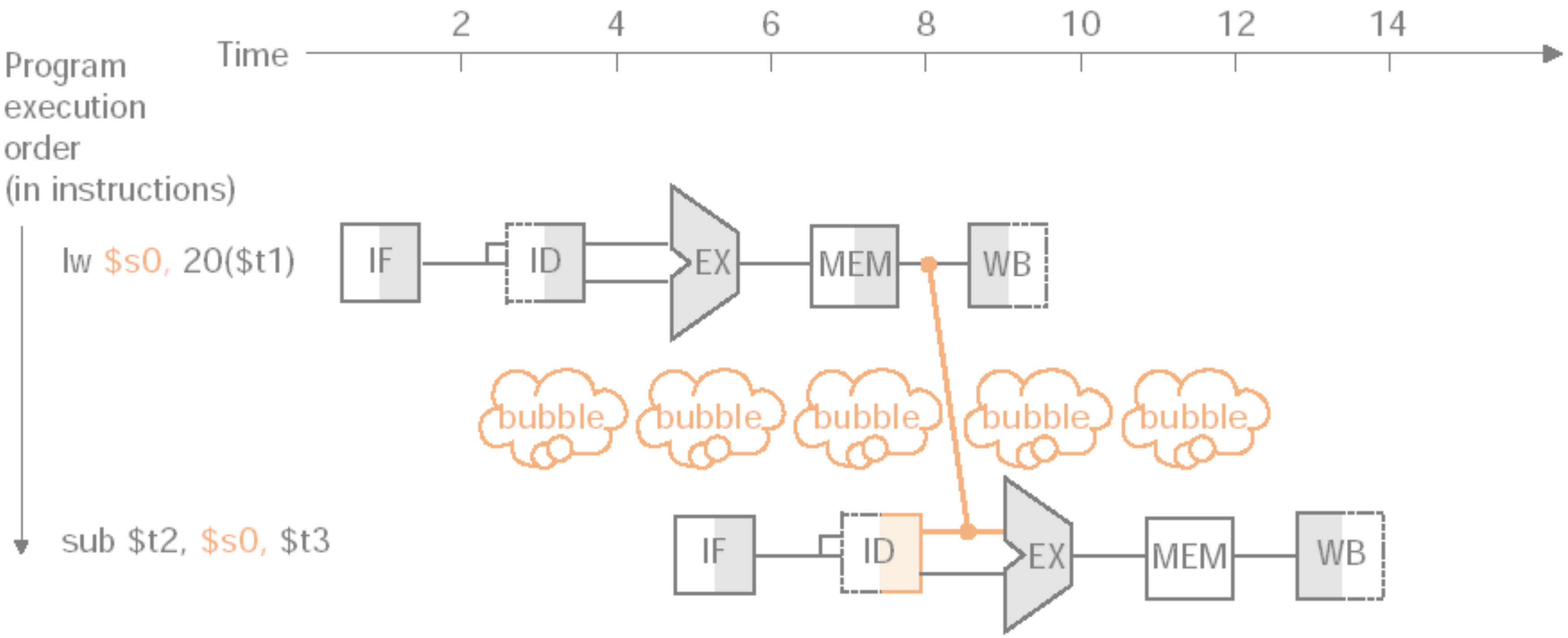


Pipeline Hazards

Data Hazards

Value from prior instruction is needed before write back

Load-Use Data Hazard: **Options: Delayed Load or Bubble**



Summary and Real Stuff

Summary

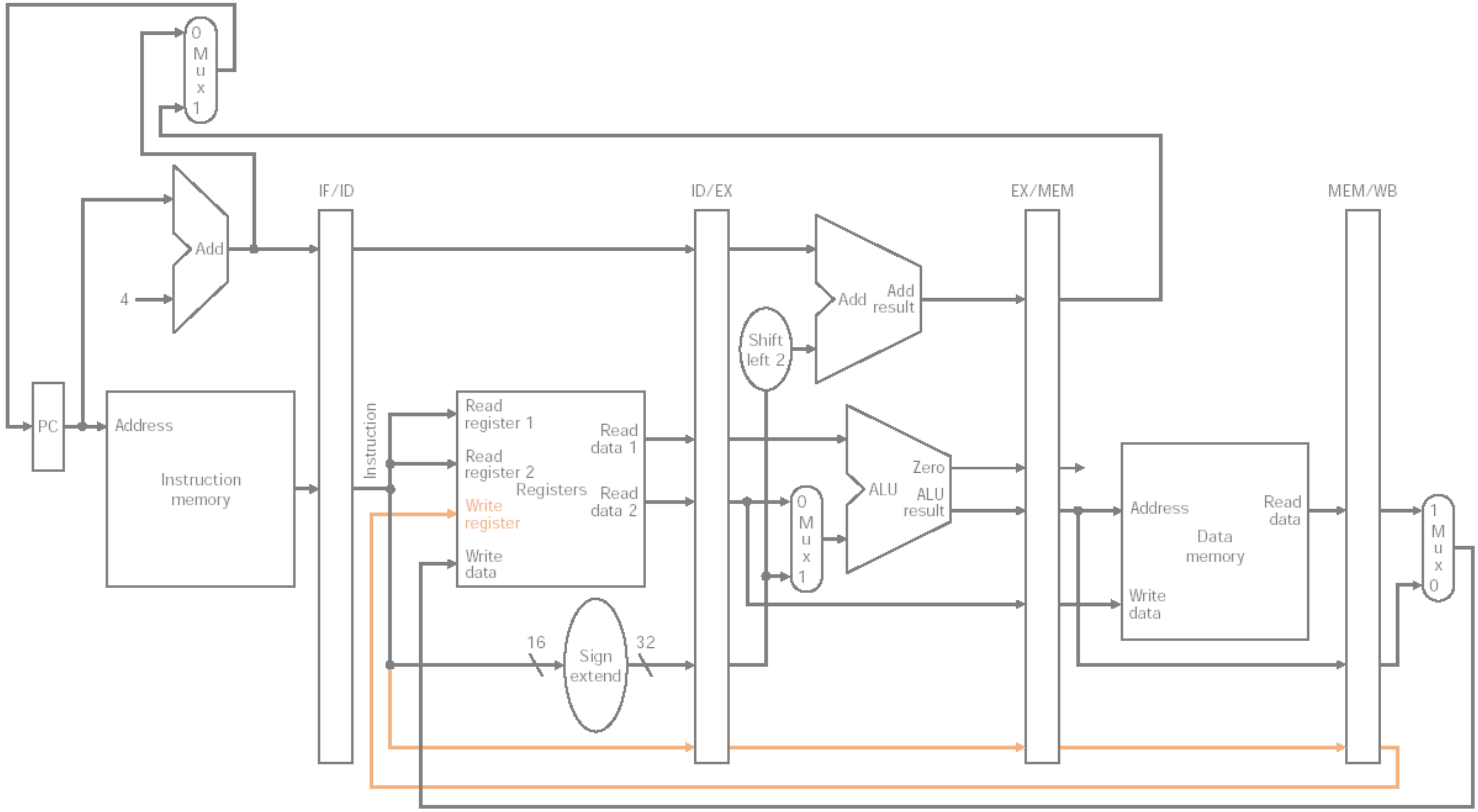
- Pipelining is a fundamental concept in computers/nature
  - Multiple instructions in flight
  - Limited by length of longest stage, Latency vs.Throughput
- Hazards gum up the works

Real Stuff

- MIPS I instruction set architecture made pipeline visible (delayed branch, delayed load)
- More performance from deeper pipelines, parallelism to a point
- Pentium 4 has 22 pipe stages!

3
4
5
6
7
8
9
10

Review: Pipelined Datapath

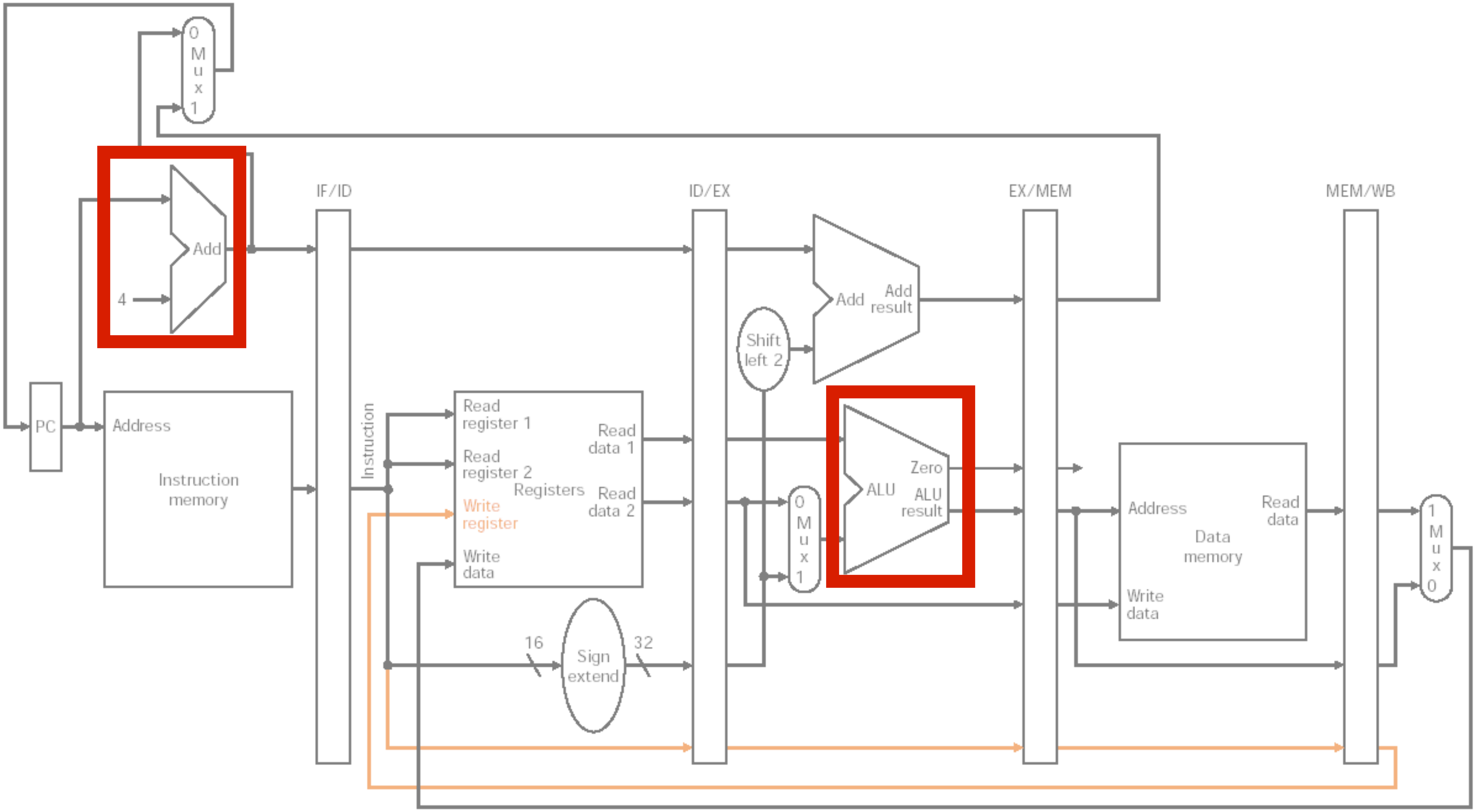


Note that all R-Type Instructions have a NULL stage!

Review: Pipeline Hazards

Structural Hazards

Resource oversubscription:

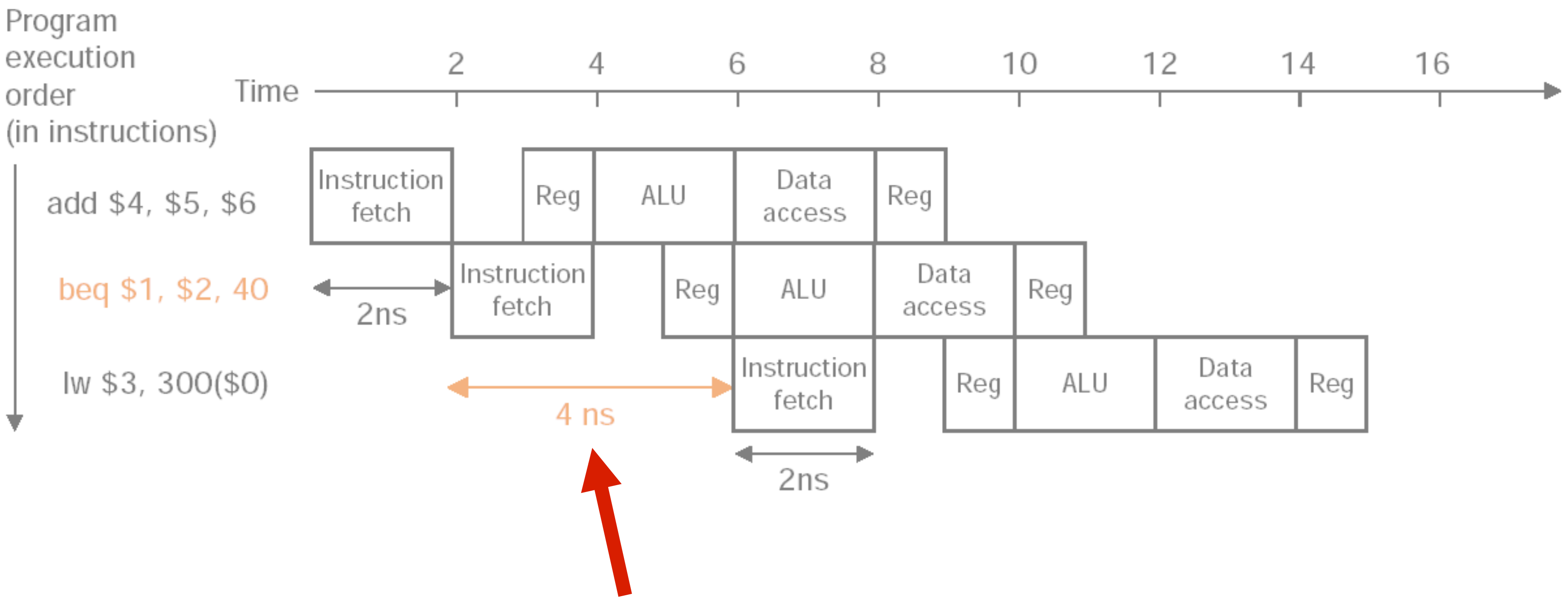


Review: Pipeline Hazards

Control Hazards

- What is the next instruction?
- Branch instructions take time to compute this.

Stall, Predict, or Delay:

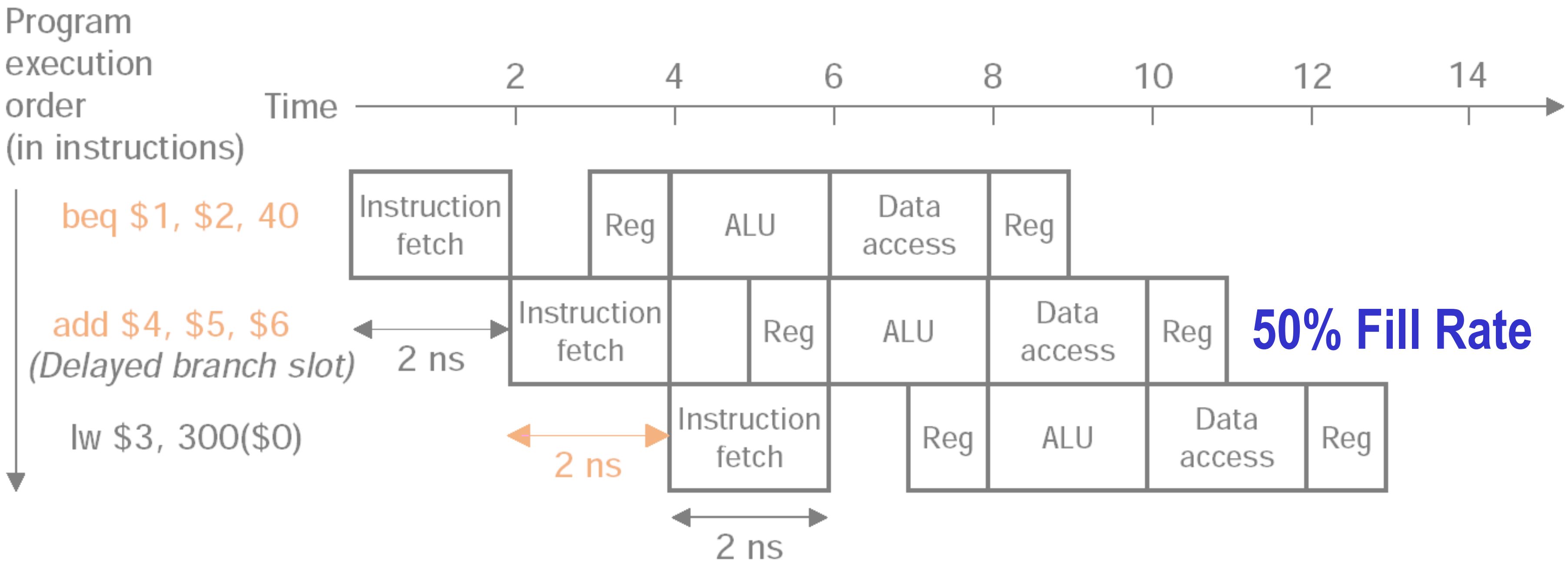


Pipeline Stall - only 1 cycle/stage delay…

Review: Pipeline Hazards
Control Hazards

- What is the next instruction?
- Branch instructions take time to compute this.

Delayed Decision (Used in MIPS):



50% Fill Rate

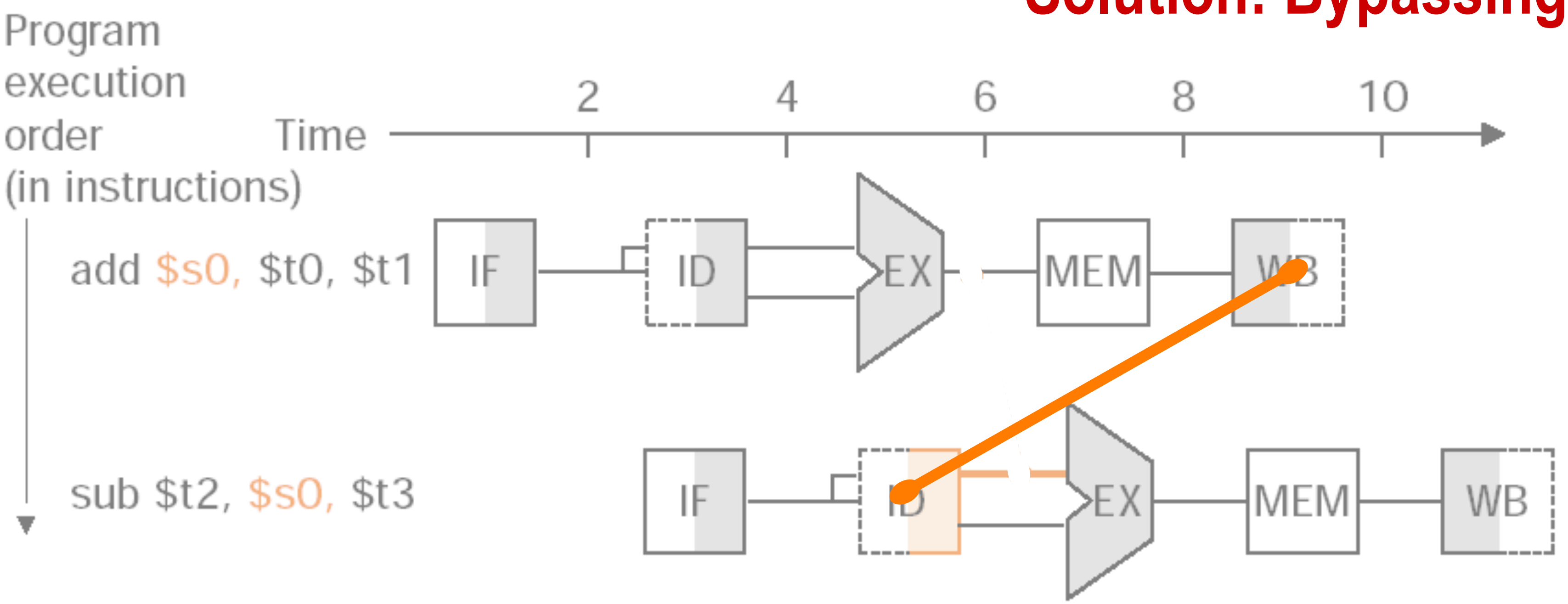
More about Branch Prediction/Delayed Branching Later…

Review: Pipeline Hazards
Data Hazards

Value from prior instruction is needed before write back

Data Hazard:

Solution: Bypassing

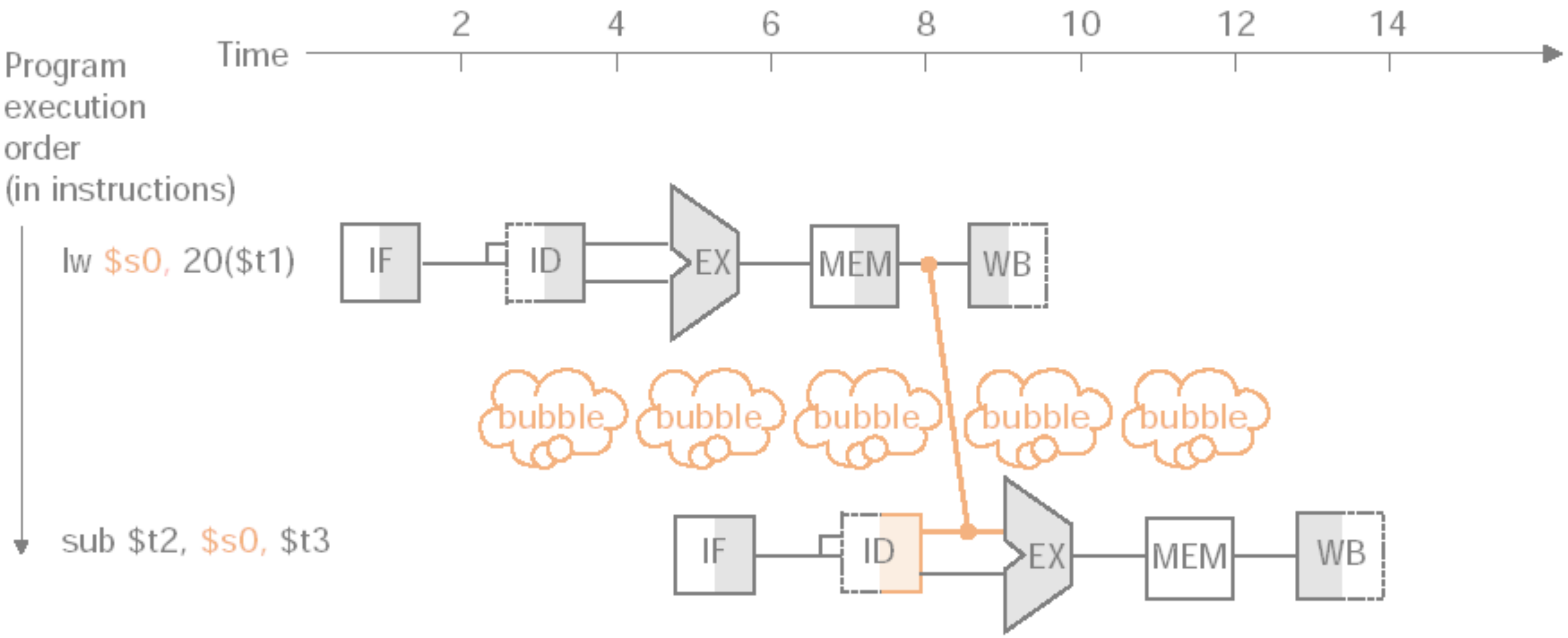


Review: Pipeline Hazards

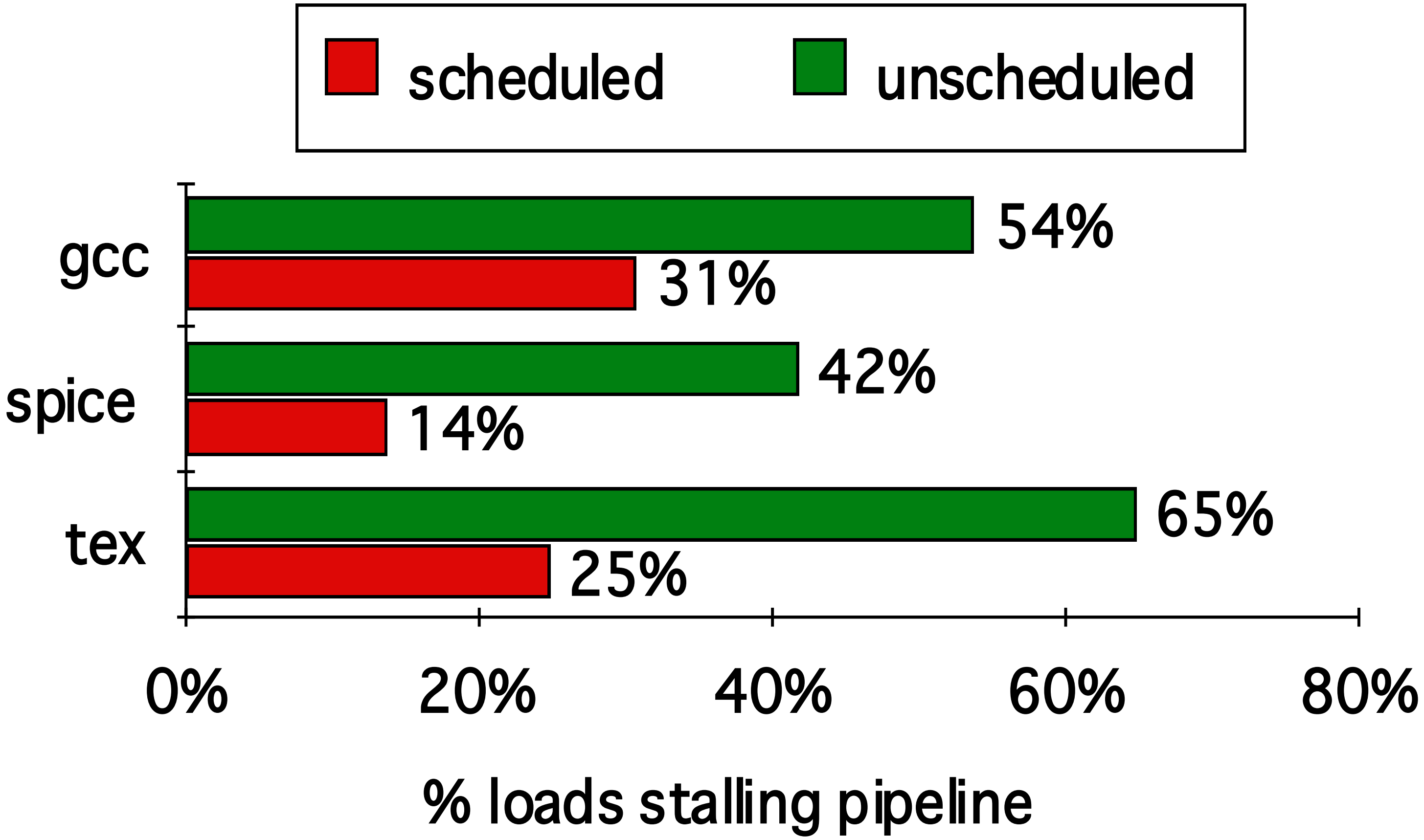
Data Hazards

Value from prior instruction is needed before write back

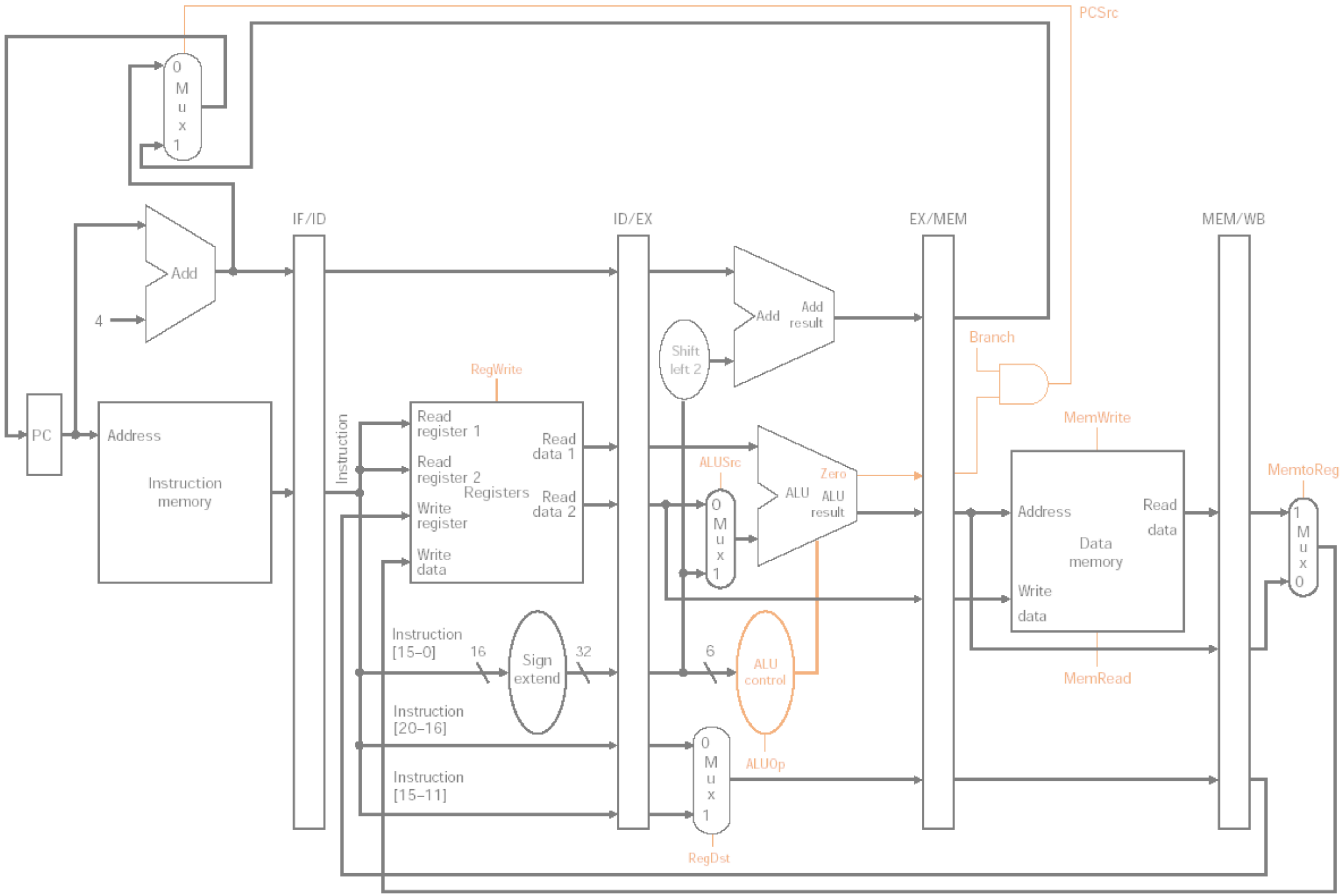
Load-Use Data Hazard: **Options: Delayed Load or Bubble**



Compiler Avoidance of Load Stalls

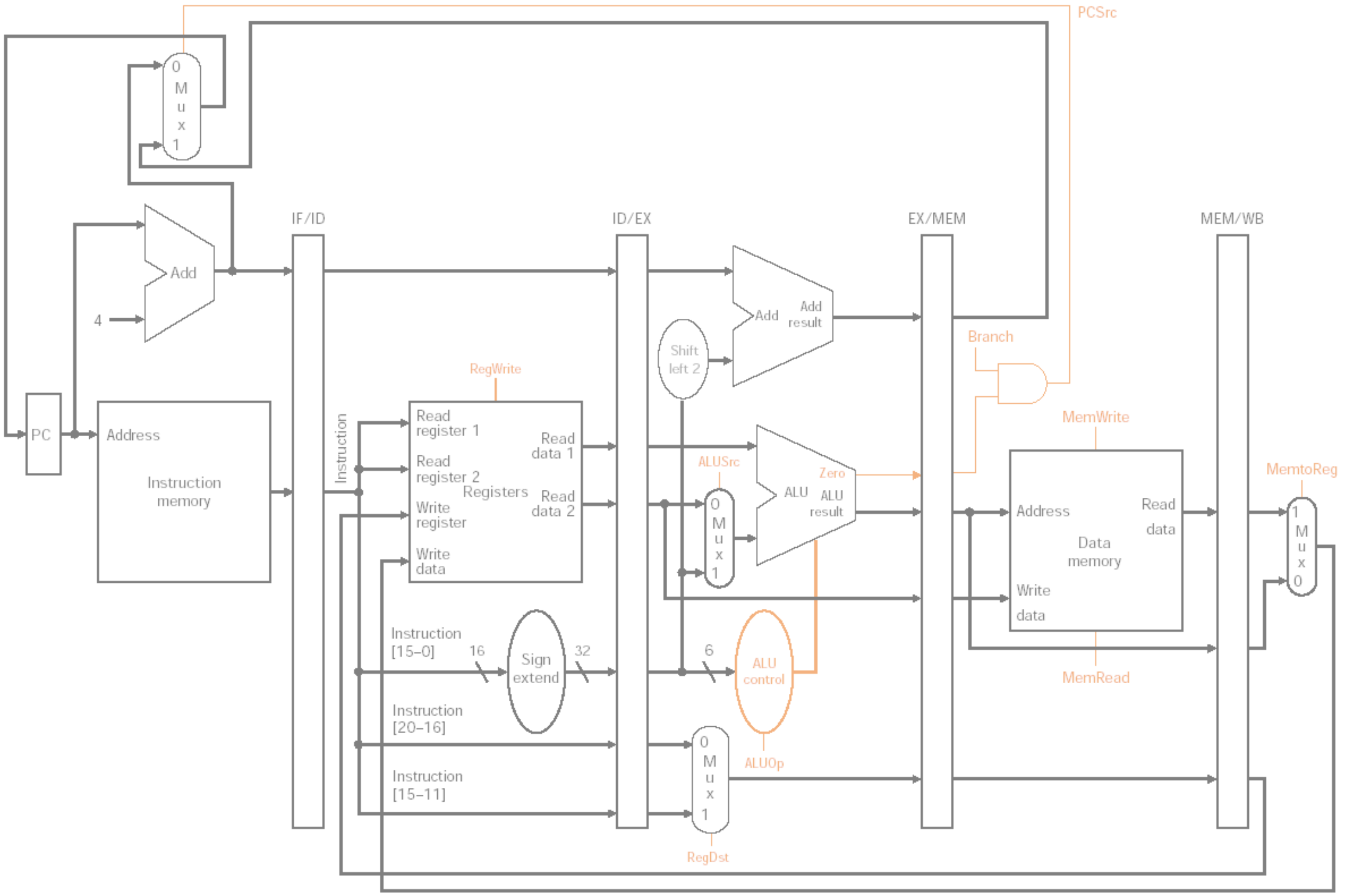


Pipeline Control



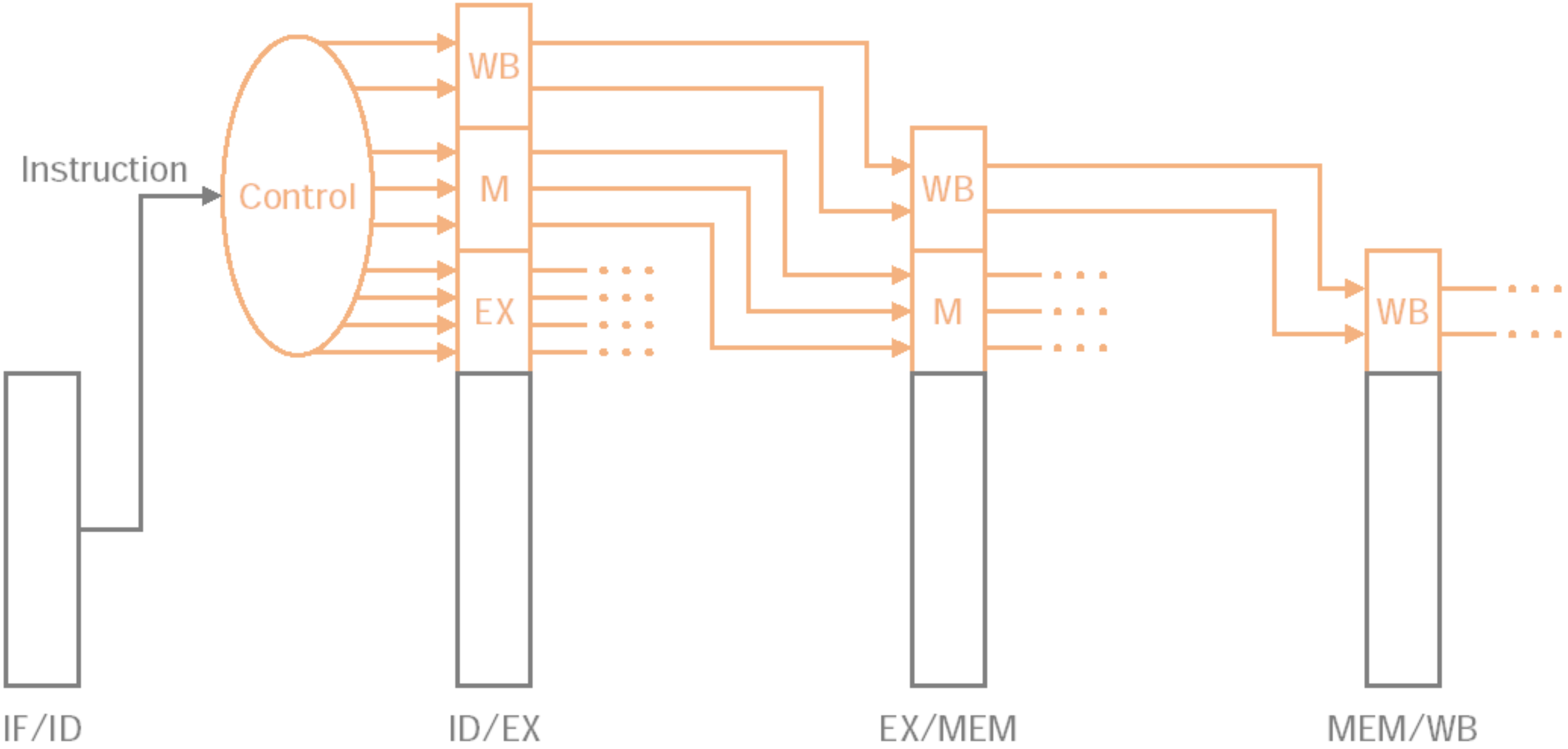
Pipeline Control

- Control is divided into 5 stages
- Signal values same as unicycle case!
- Timing is different…

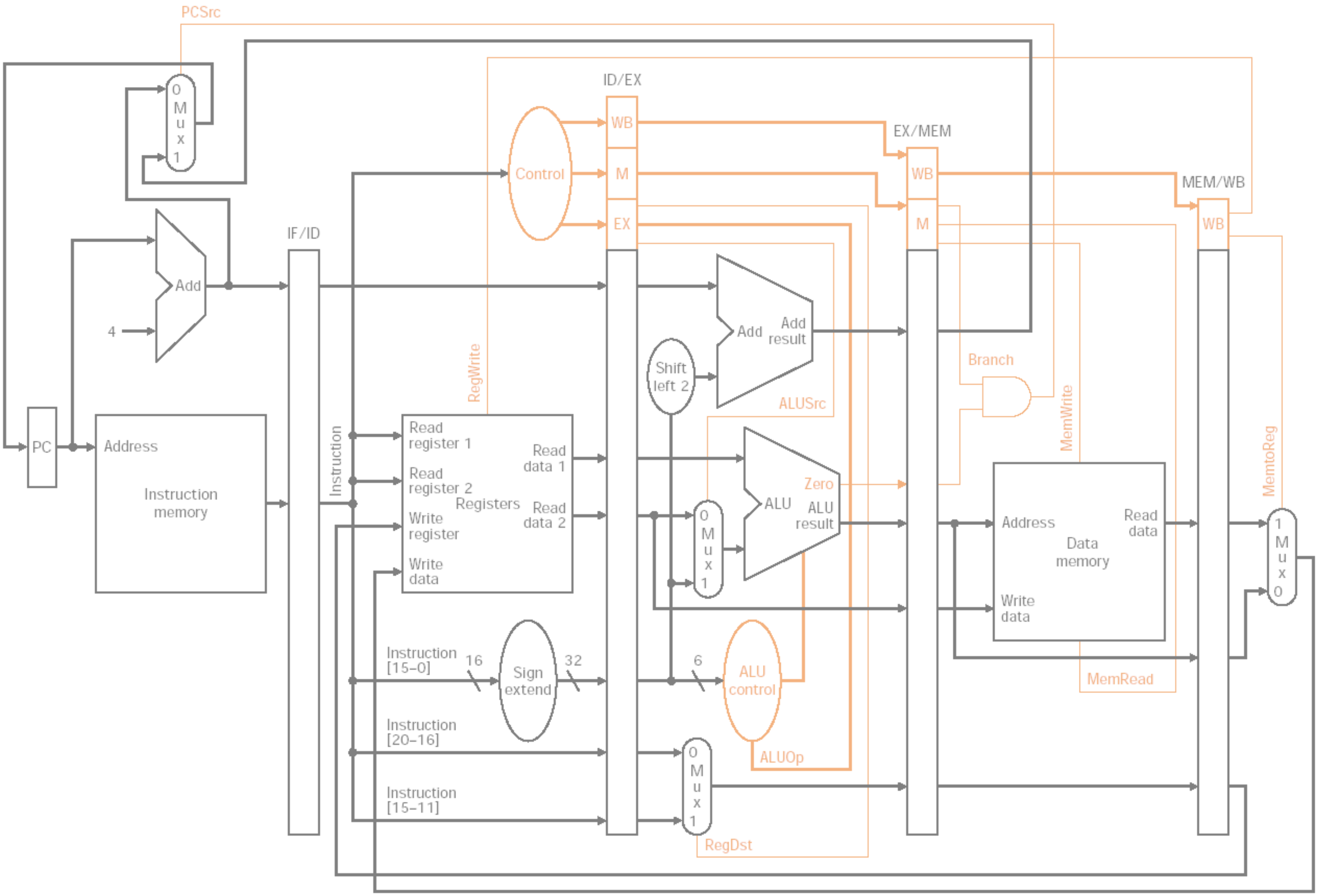


Pipeline Control

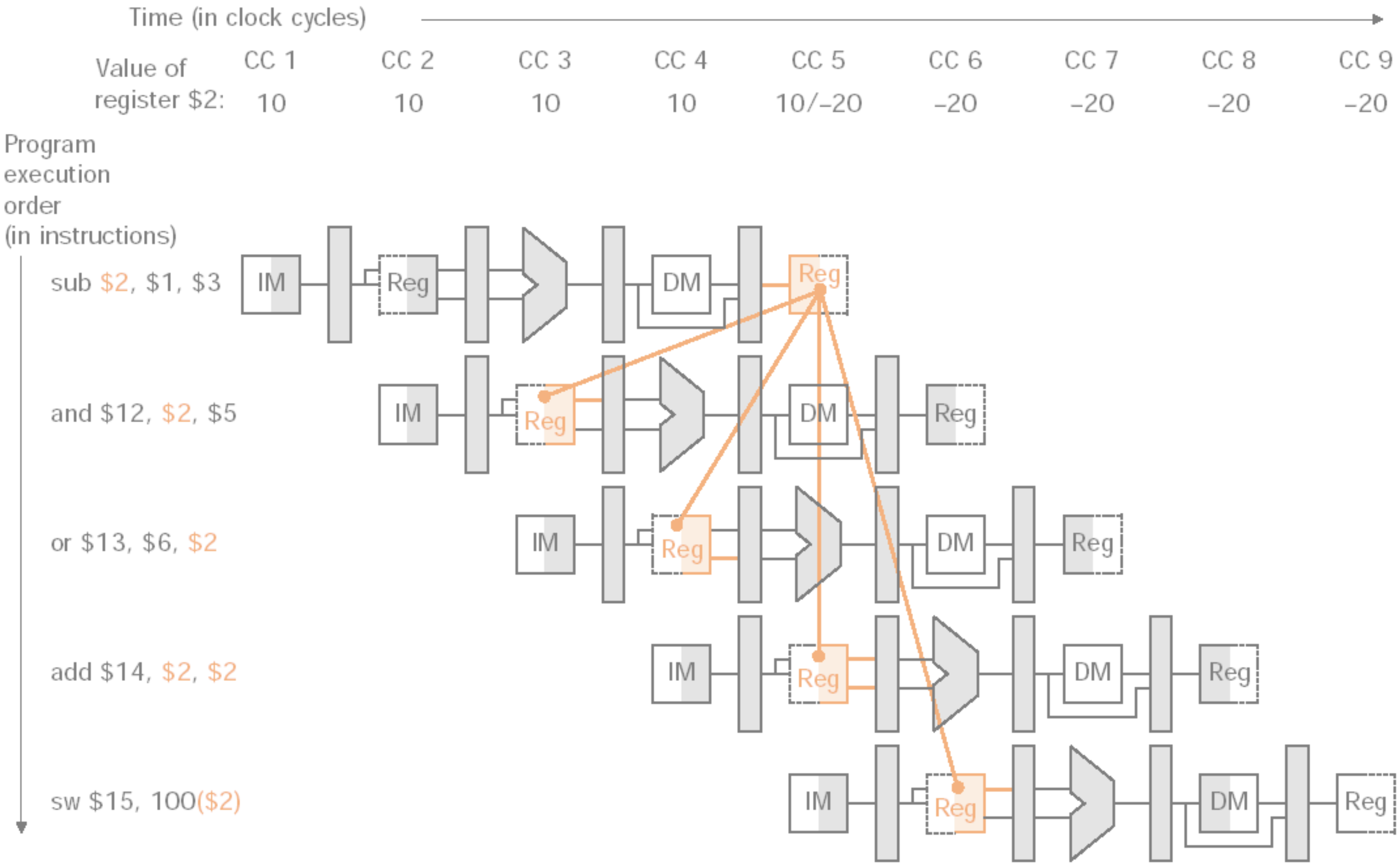
- Signal values same as unicycle case!
- Timing is different…
- Simplest method: Extend pipe registers



Pipeline Control



What About Data Hazards?



Time (in clock cycles)

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2:	10	10	10	10	10/–20	–20	–20	–20	–20

Program execution order (in instructions)

sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)

What About Data Hazards?

Time (in clock cycles) →

	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Value of register $2 :	10	10	10	10	10/–20	–20	–20	–20	–20
Value of EX/MEM :	X	X	X	–20	X	X	X	X	X
Value of MEM/WB :	X	X	X	X	–20	X	X	X	X

Program
execution order
(in instructions)

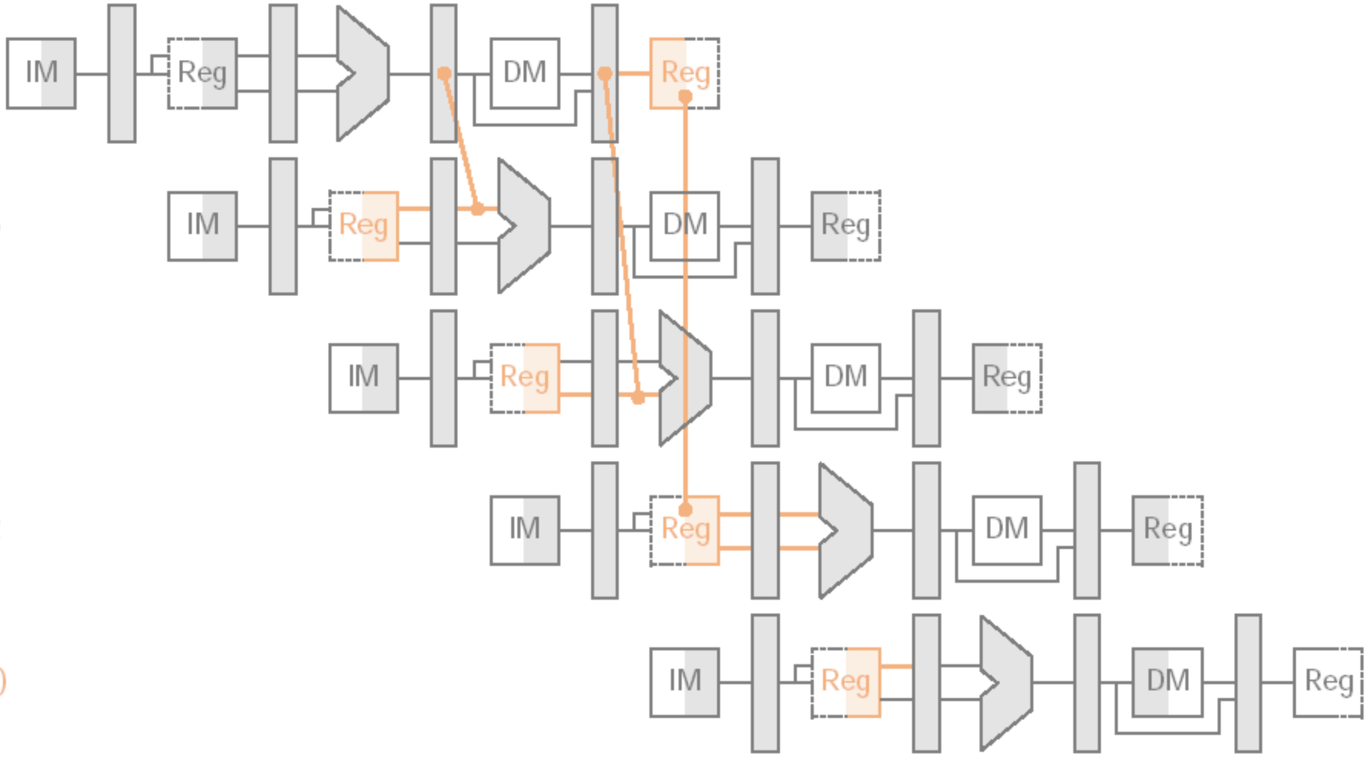
sub $2, $1, $3

and $12, $2, $5

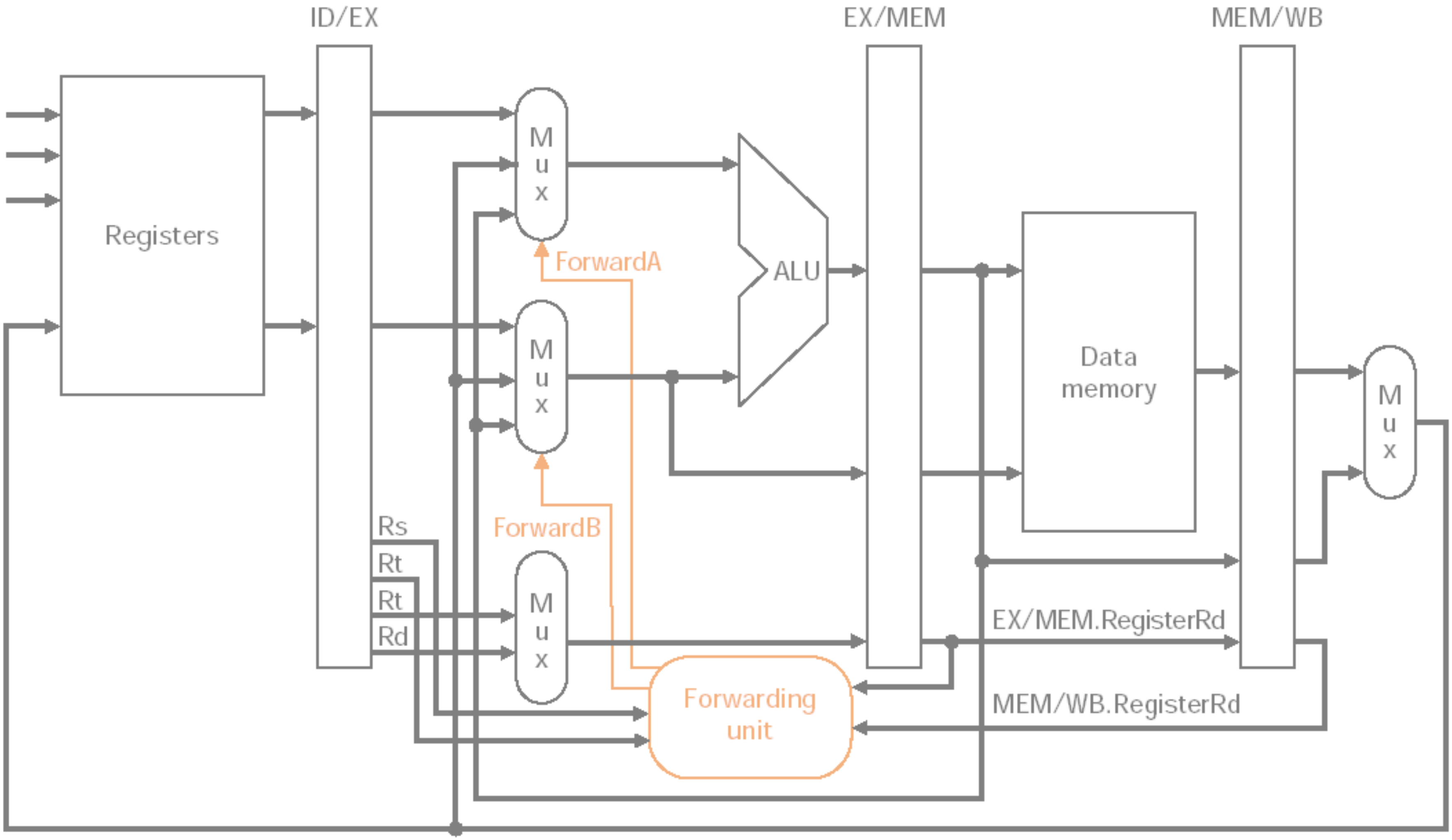
or $13, $6, $2

add $14, $2, $2

sw $15, 100($2)

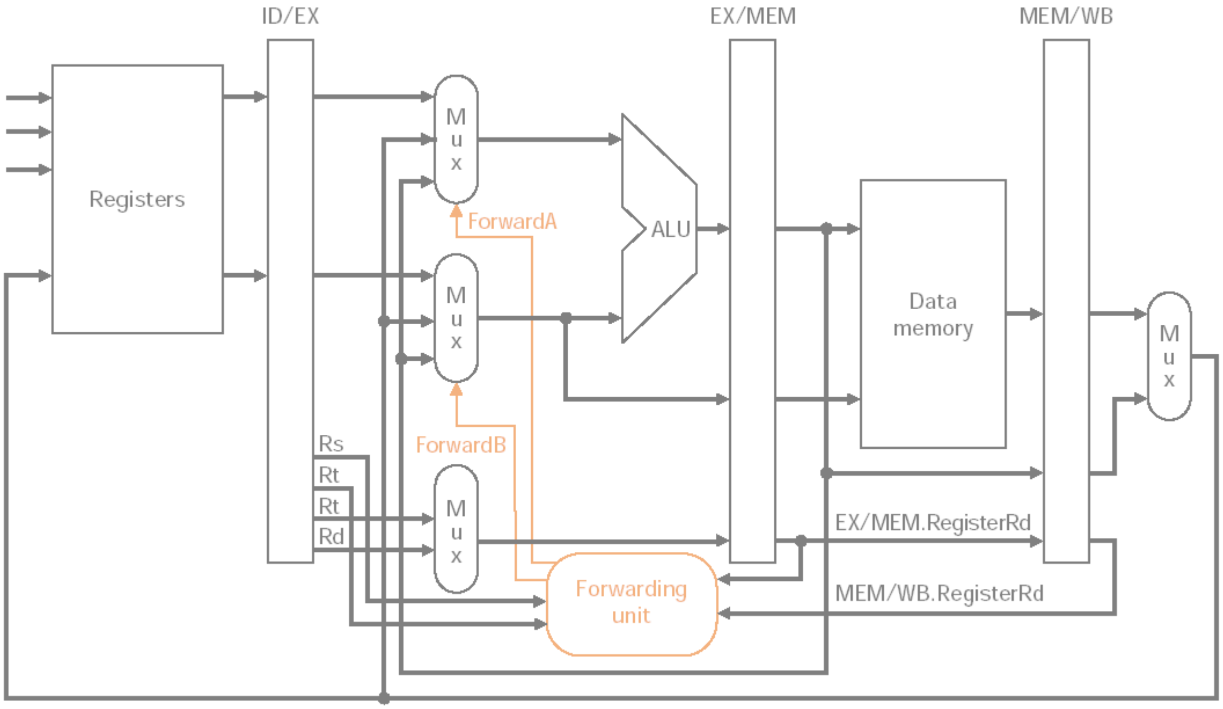


Forwarding Unit



How does the Forwarding Unit know when to forward?

Forwarding Unit



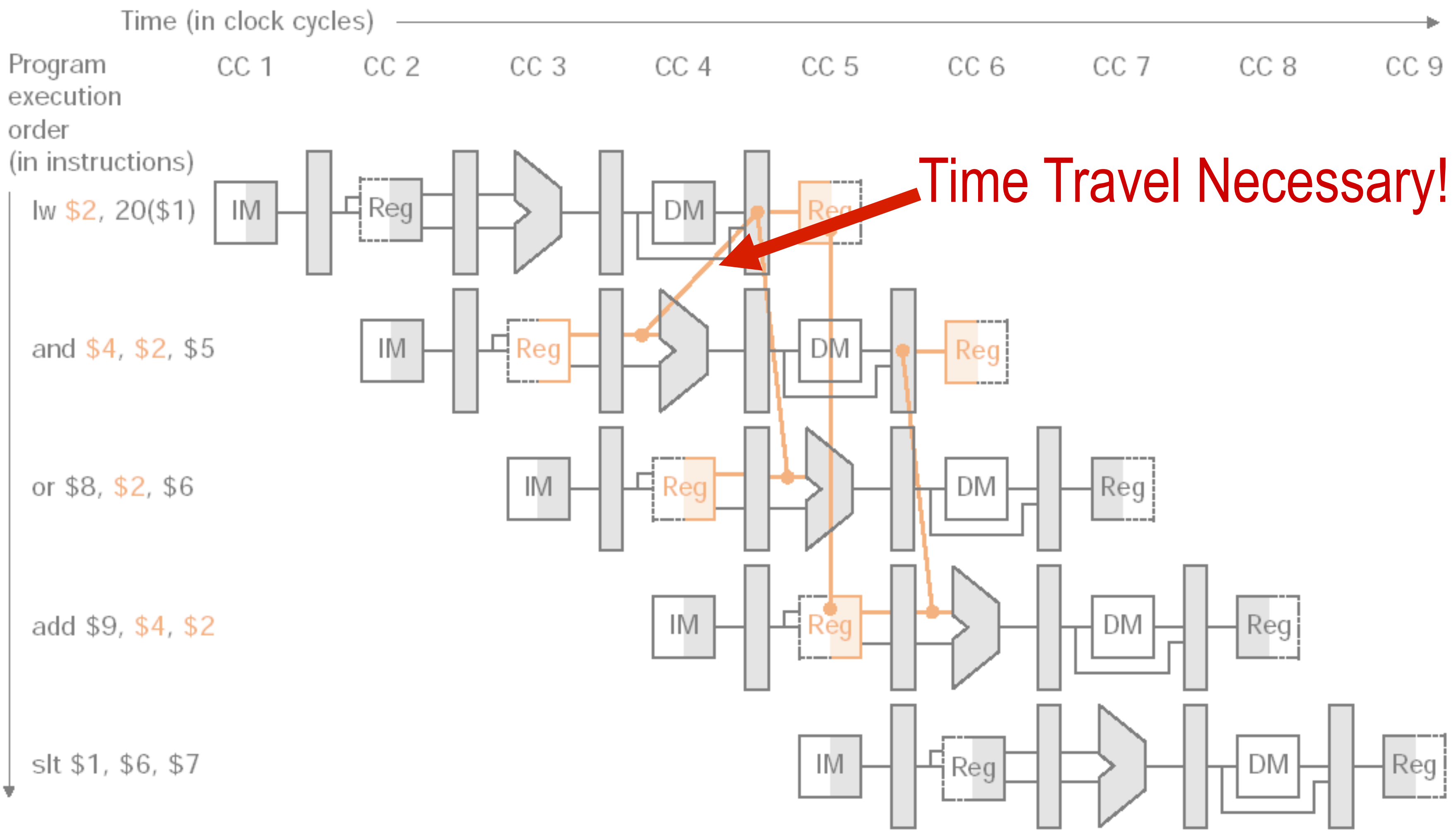
EX Hazard:

EX/MEM.RegWrite AND EX/MEM.RegisterRd != 0 AND EX/MEM.RegisterRd == ID/EX.RegisterReadRs(Rt)

MEM Hazard very similar, but prefer MEM over WB value

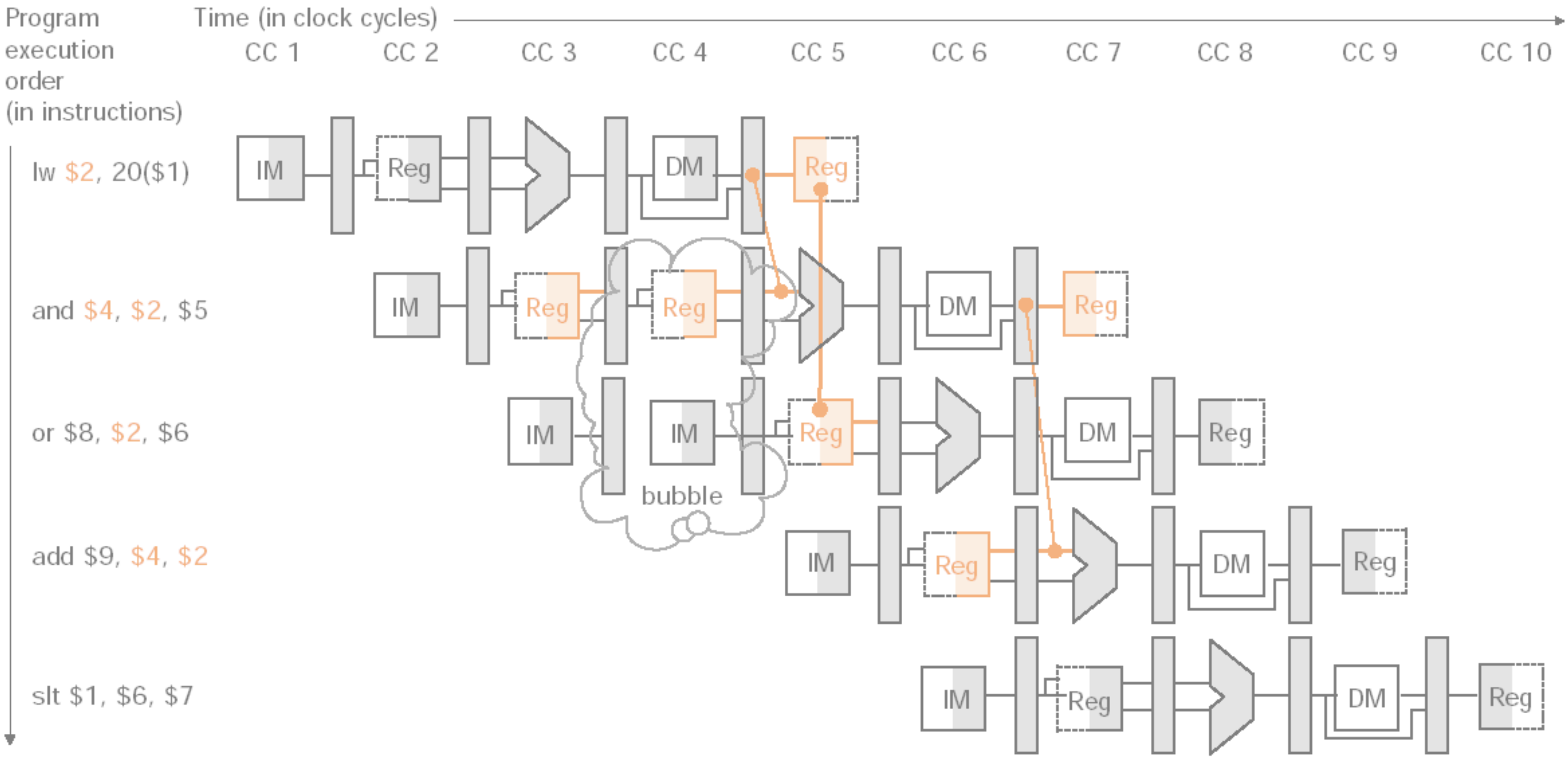
What About Load-Use Stall?

- Forwarding can’t save the day
- Need to introduce stall in hardware or compiler



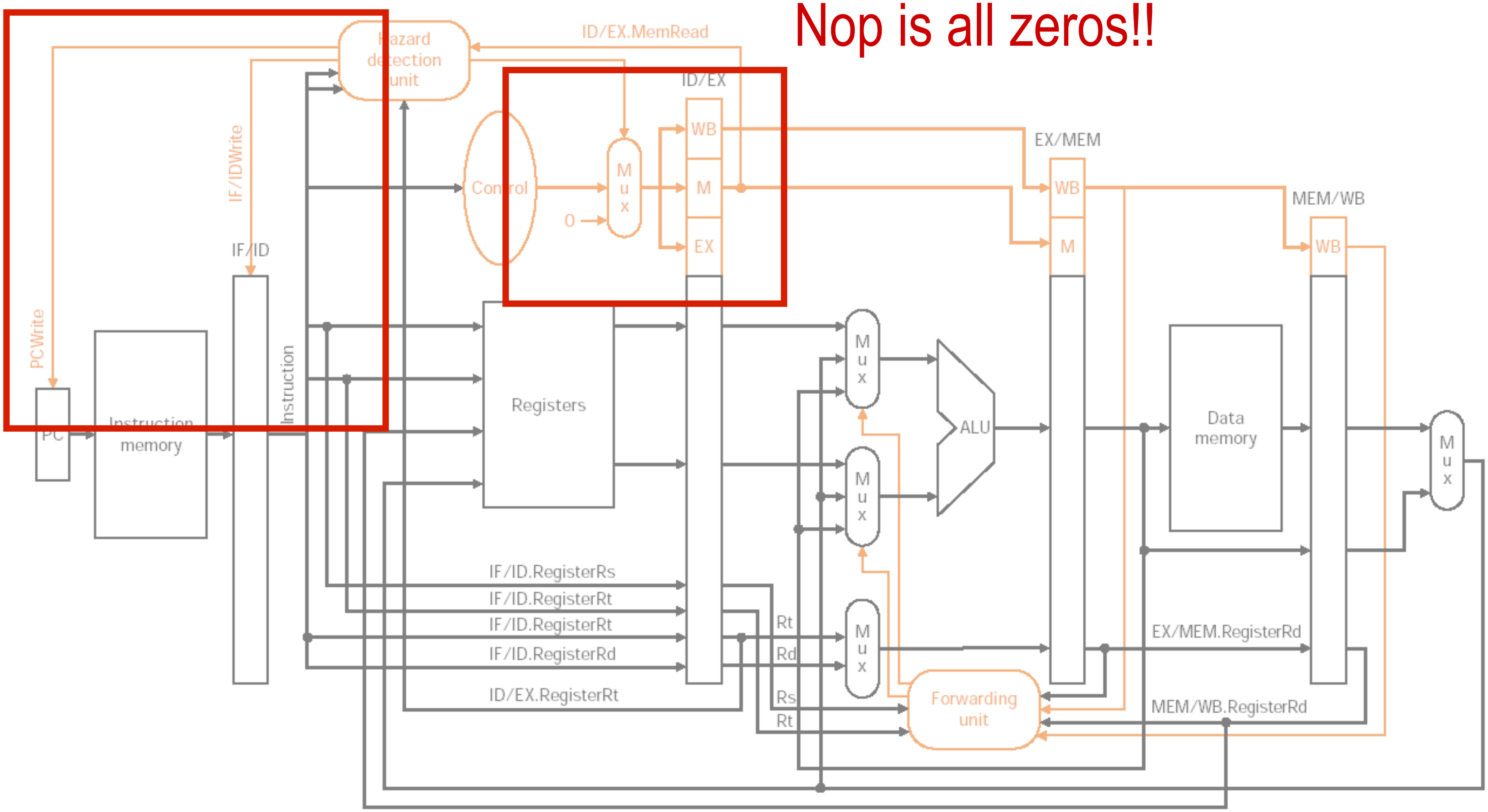
Time Travel Necessary!

What About Load-Use Stall?



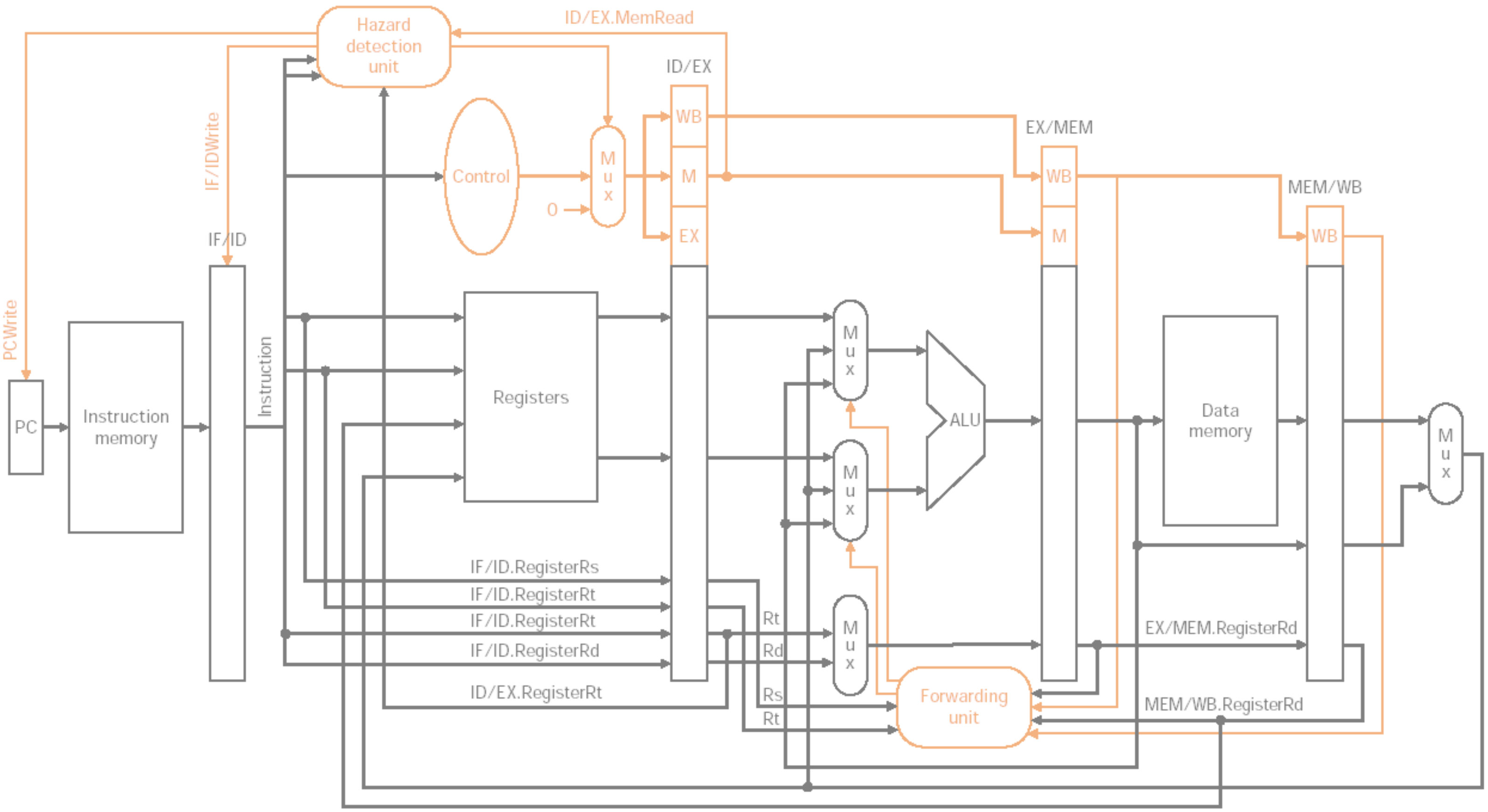
Hazard Detection Unit

Nop is all zeros!!



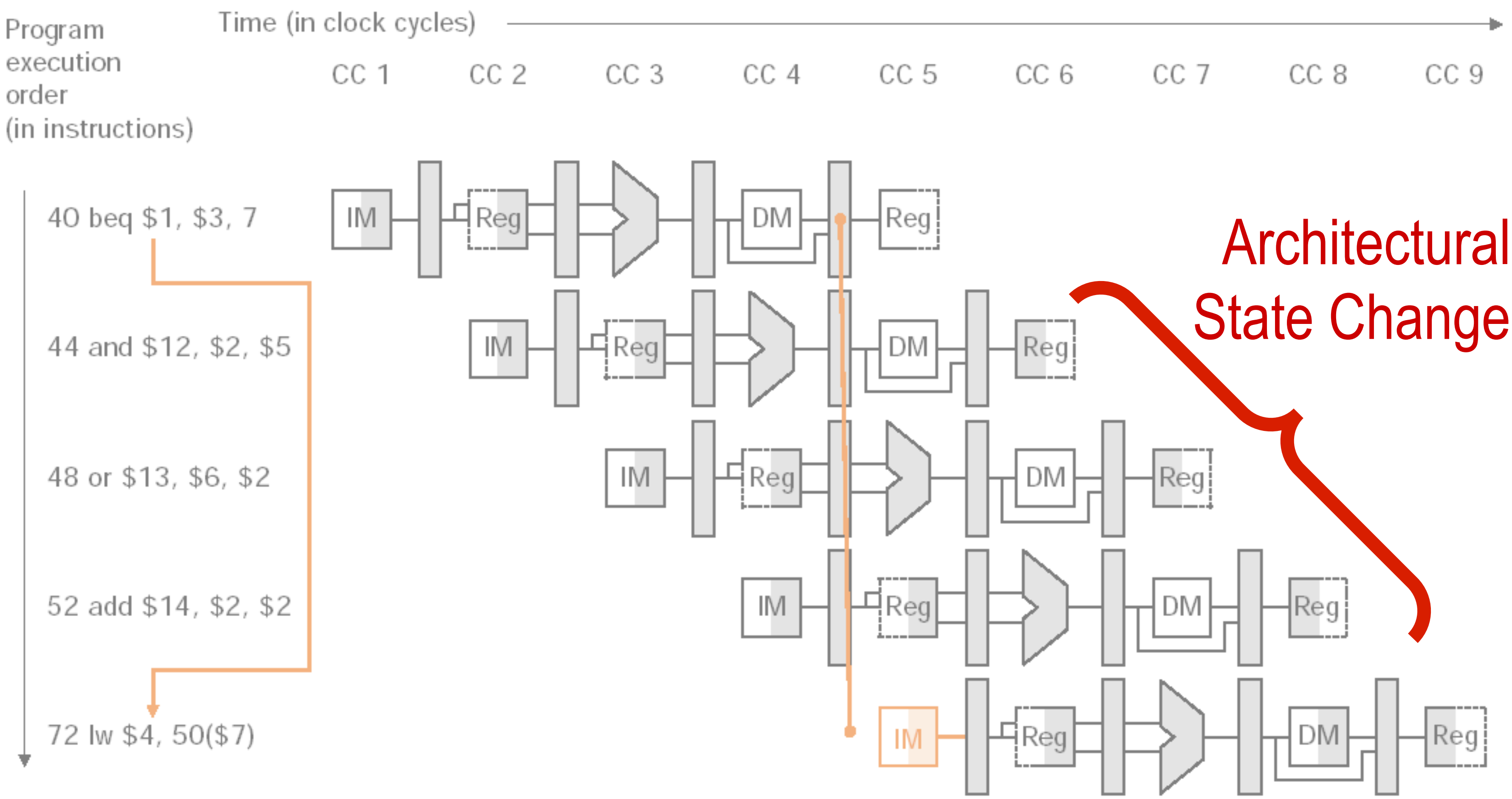
How does the Hazard Detection Unit know when to forward?

Hazard Detection Unit



ID/EX.MemRead AND
(ID/EX.RegisterRt == IF/ID.RegisterRs OR ID/
EX.RegisterRt == IF/ID.RegisterRt)

What About Control Hazards?
(Predict Not-Taken Machine)



Architectural State Change?

We are OK, as long as we squash. **Can we reduce delay?**

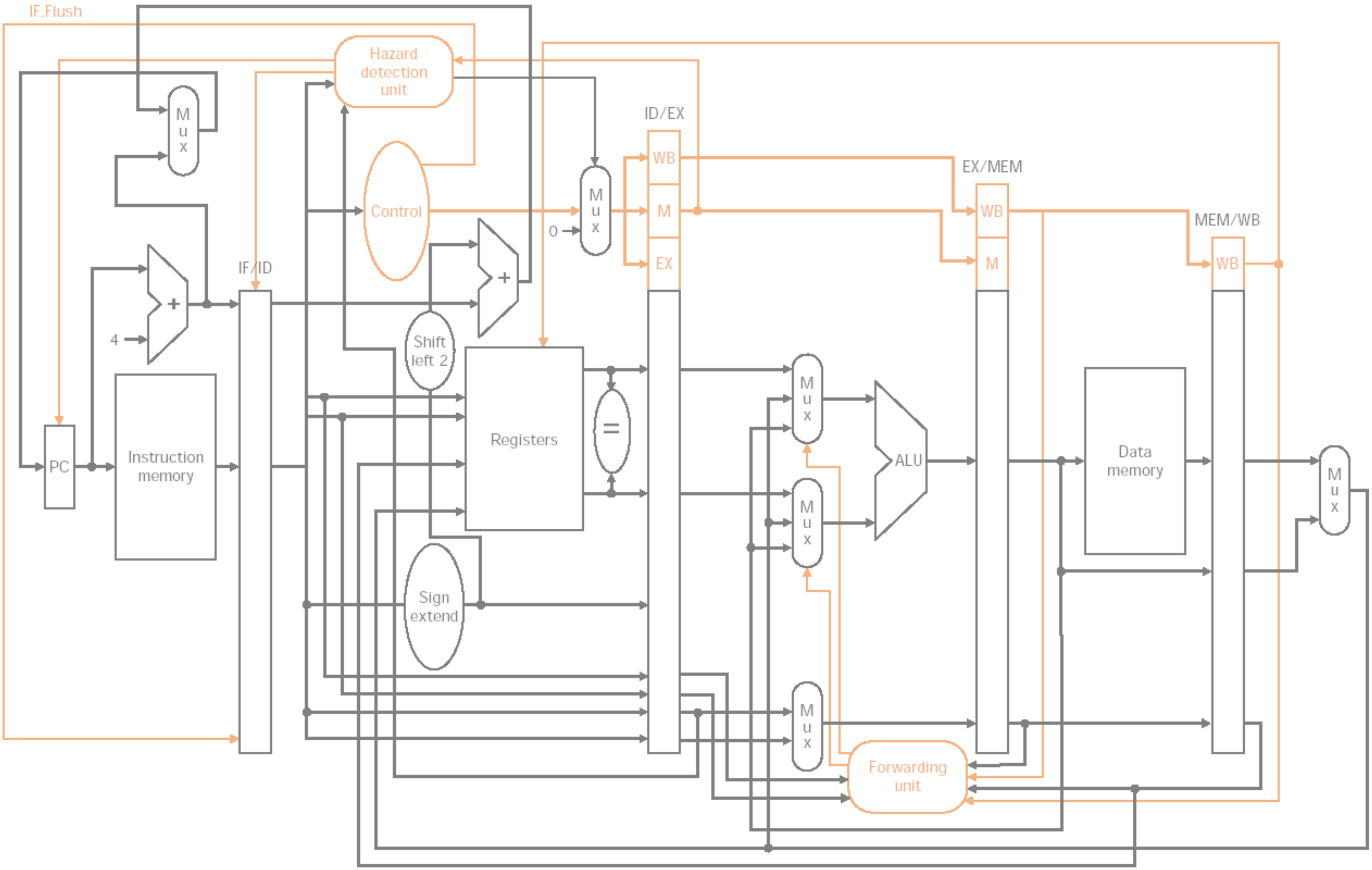
Reduce Branch Delay

1. Move branch address calculation to decode stage (from MEM stage)

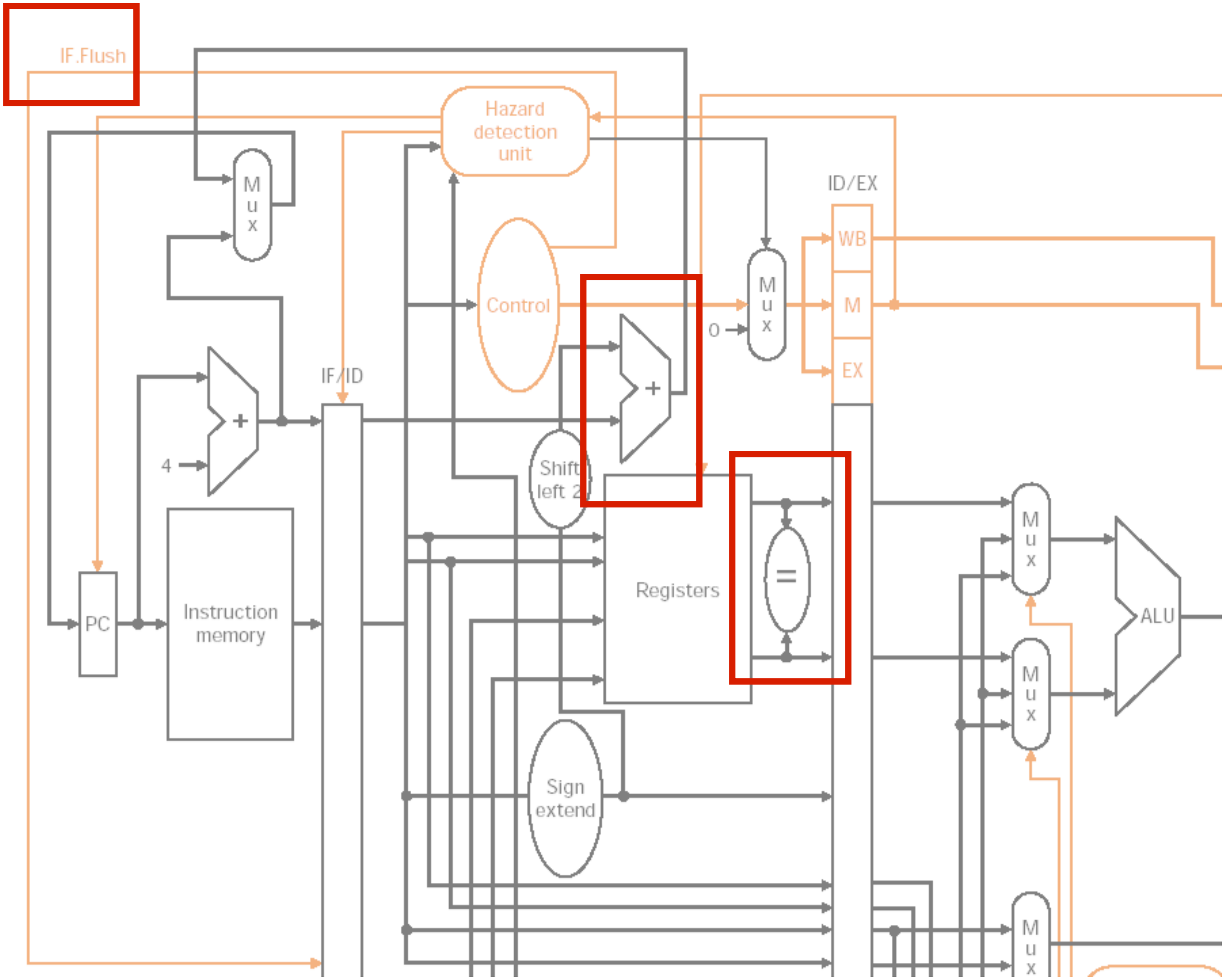
2. Move branch decision up (Harder
   - Bitwise-XOR, test for zero
   - Only need Equality testing
   - Much faster: No carry

Everything is done in decode stage!!

What About Control Hazards?



What About Control Hazards?



NO EXCEPTIONS

Review: Exceptions

- What happens if instruction encoding is not valid?
- What about arithmetic overflow?

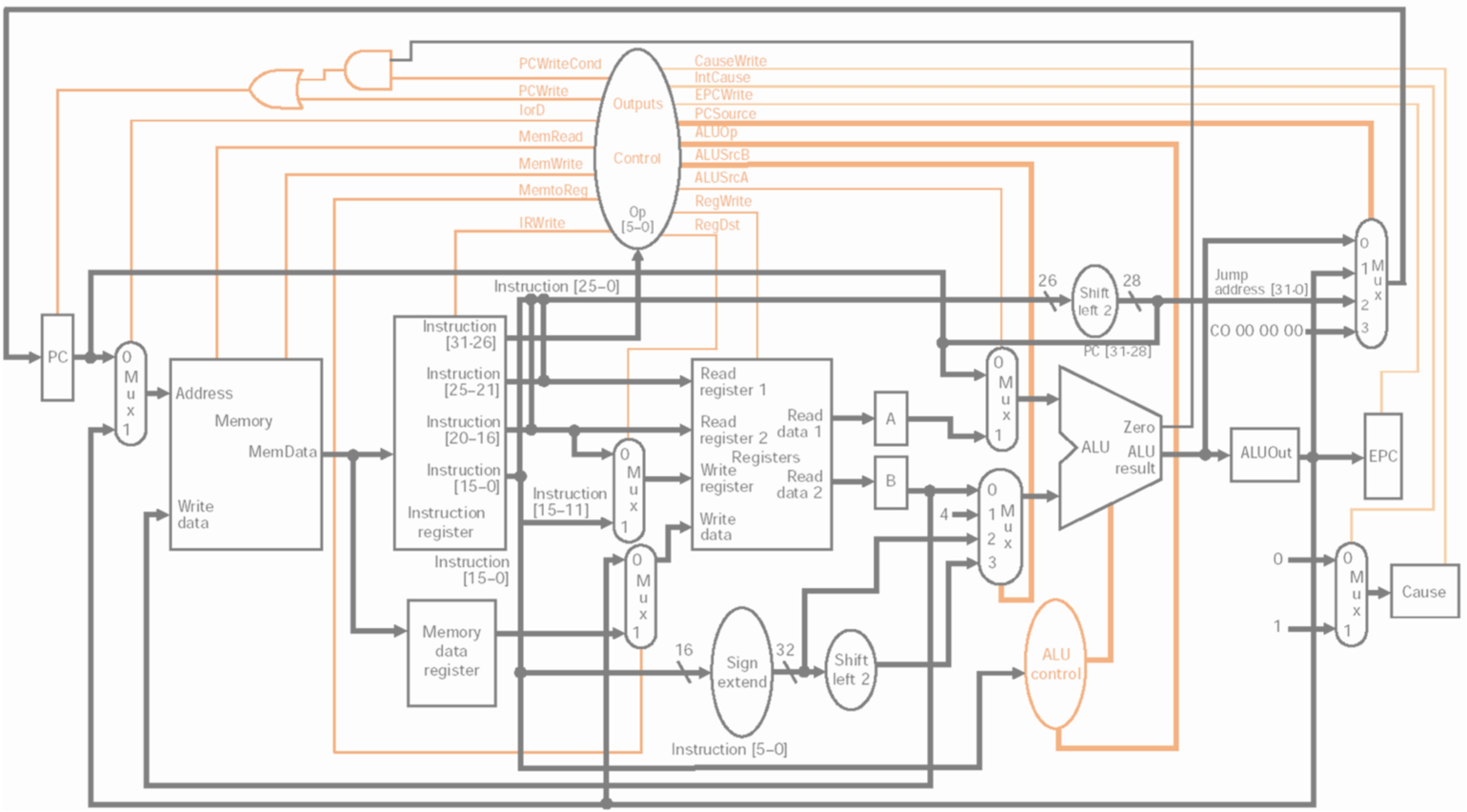
Exception

An event that disrupts program execution.

When an exception occurs:

- Save the current PC in the EPC
- Cause = 0 for Undefined Instruction, 1 for Overflow
- Jump to the OS at $C0000000_{16}$ (not vectored)

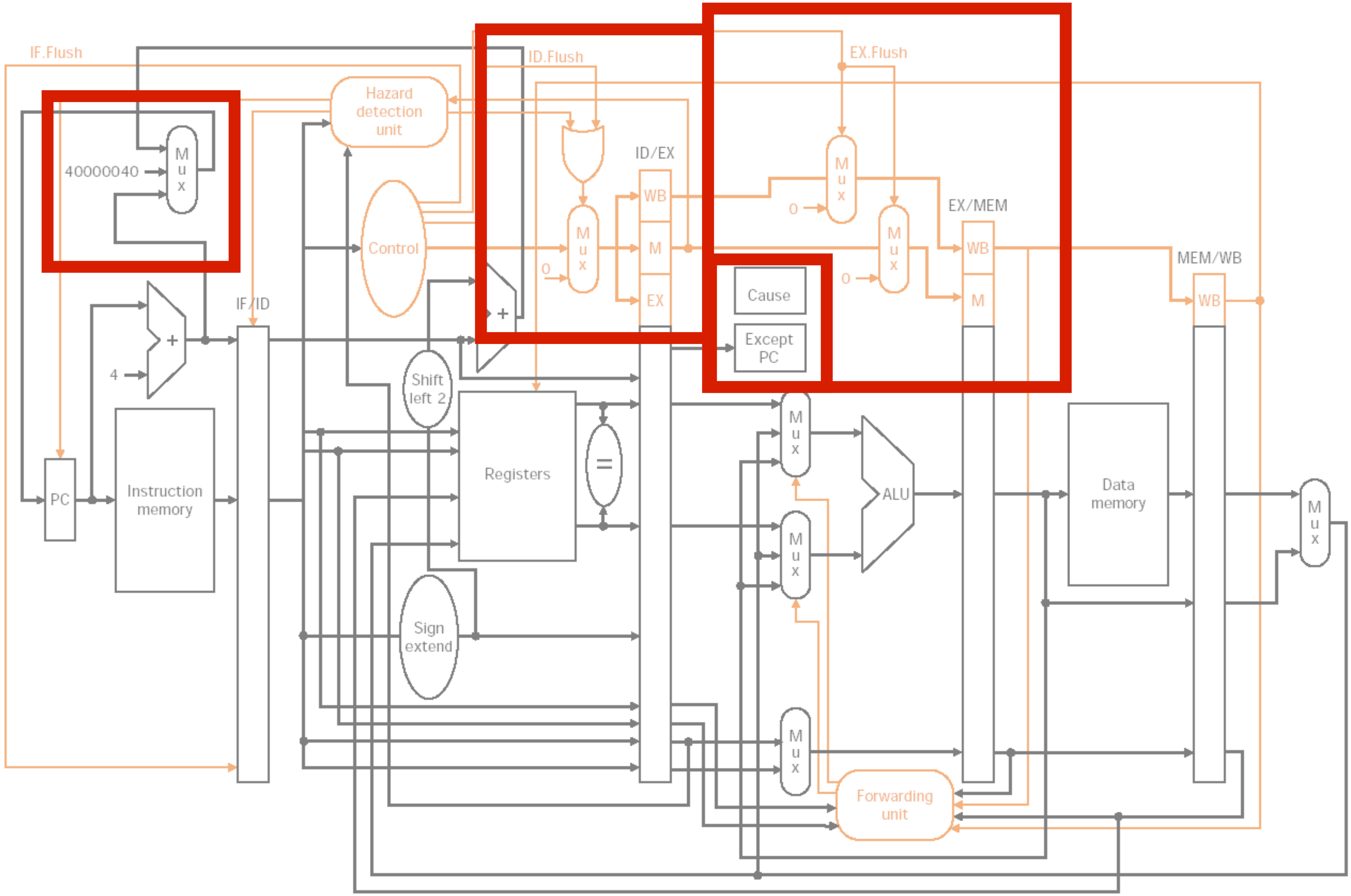
Review: Multicycle Exception Handling



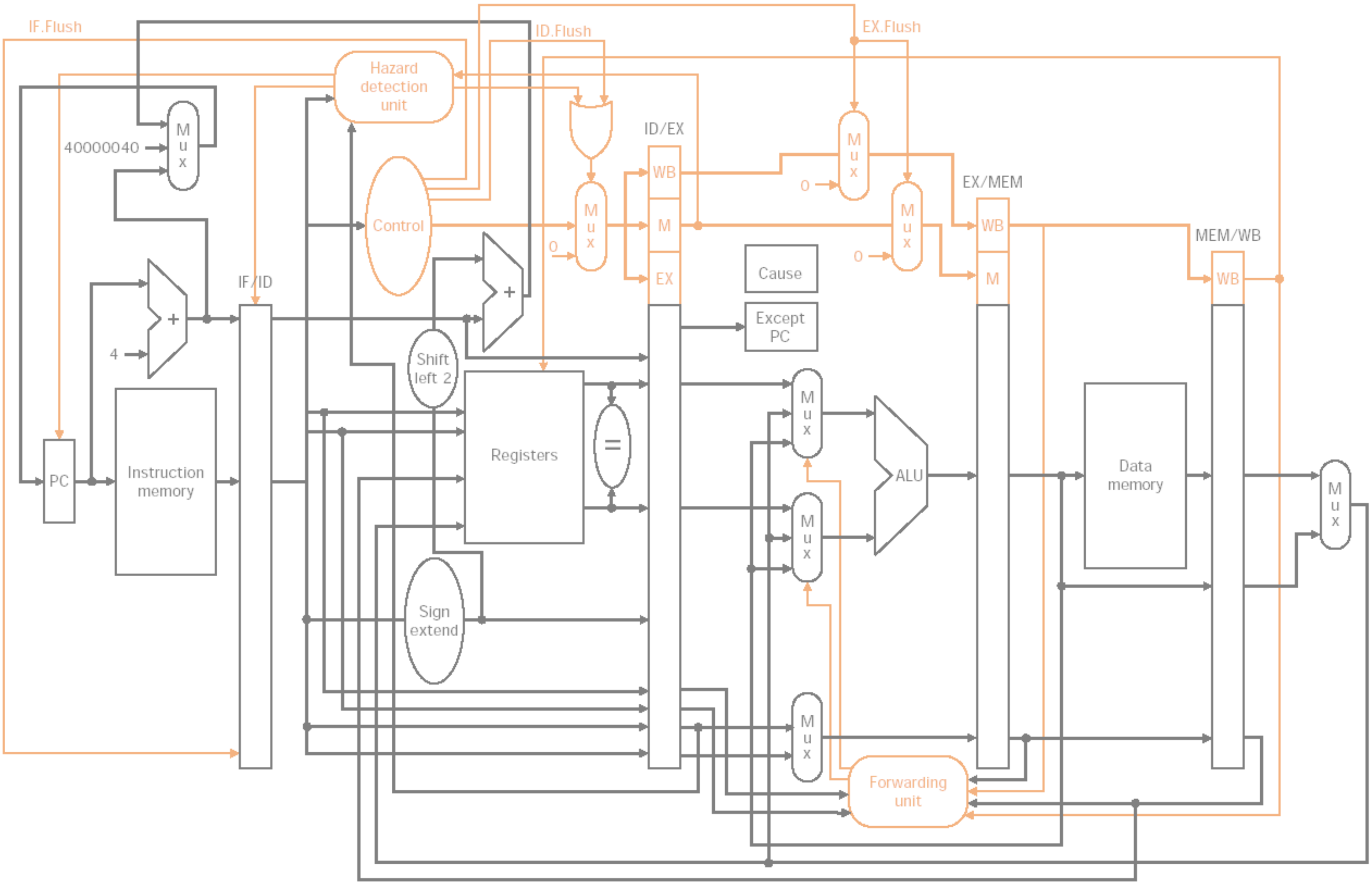
Exceptions in Pipelines

- Exception must appear to programmer/OS as it would in unicycle/multicycle
- Must squash in-flight instructions after excepting inst
- Looks a lot like a branch…

Pipeline Exception Handling



Look at this mess!!!



Precise vs. Imprecise Exceptions

Precise Exceptions

- EPC has value of excepting instruction PC
- Easy for OS to handle
- We have been looking at precise exception machine

Imprecise Exceptions

- Reduce pipeline complexity by putting current PC or other approximation into EPC
- OS figures it out

Summary

- Pipelining is a fundamental concept in computers/nature
  - Multiple instructions in flight
  - Limited by length of longest stage, Latency vs.Throughput
- Hazards gum up the works
- Pipeline Control can be messy!