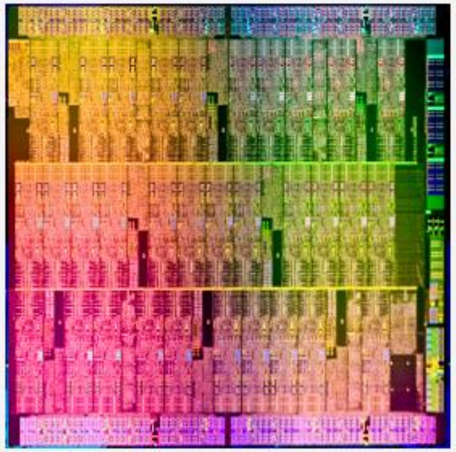
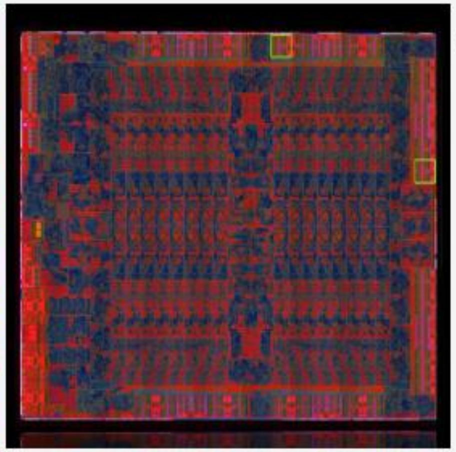
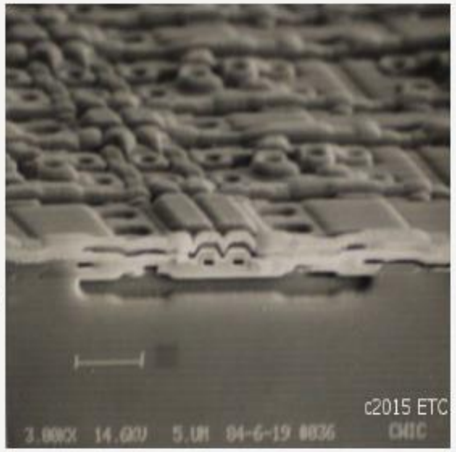
# 18-344: Computer Systems and the Hardware-Software Interface

Fall 2023




## Course Description

**Lecture 5: Pipelines and Hazards**

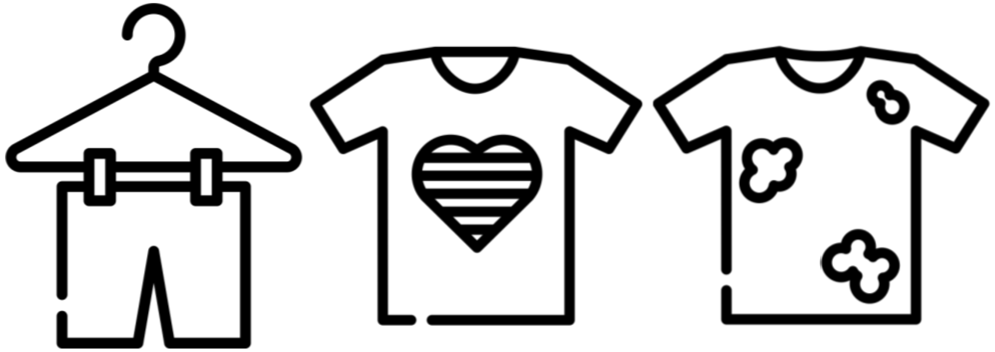
This course covers the design and implementation of computer systems from the perspective of the hardware software interface. The purpose of this course is for students to understand the relationship between the operating system, software, and computer architecture. Students that complete the course will have learned operating system fundamentals, computer architecture fundamentals, compilation to hardware abstractions, and how software actually executes from the perspective of the hardware software/boundary. The course will focus especially on understanding the relationships between software and hardware, and how those relationships influence the design of a computer system's software and hardware. The course will convey these topics through a series of practical, implementation-oriented lab assignments.

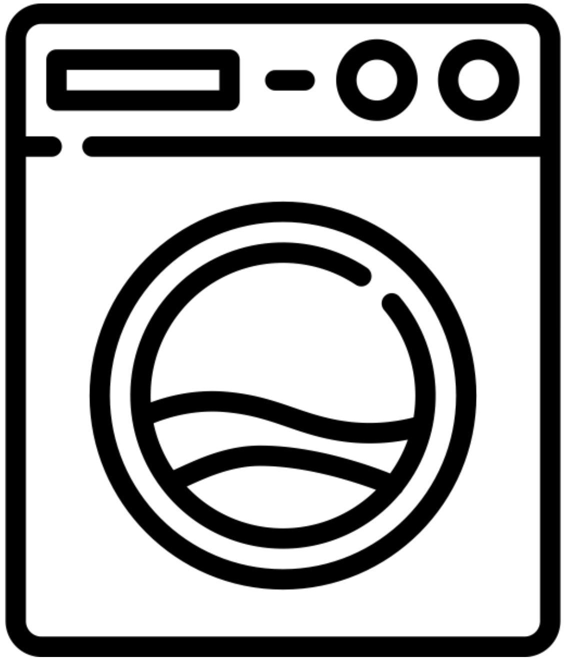
**Credit: Brandon Lucia**

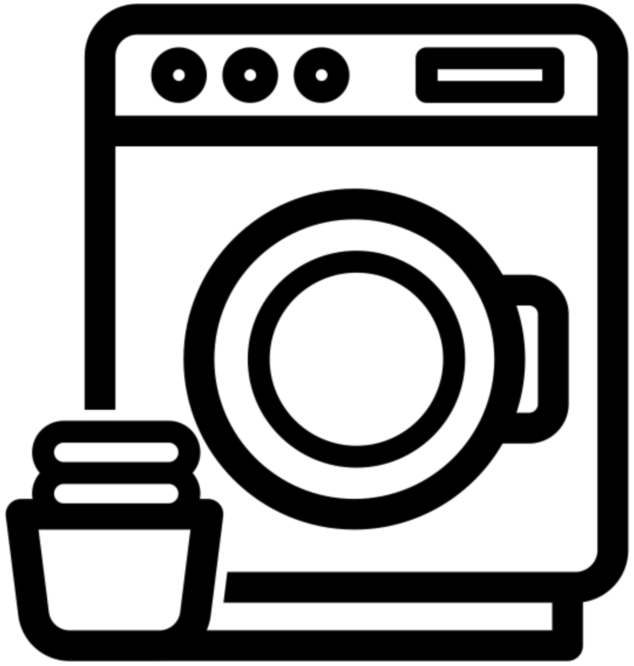
# Pipelined Microarchitectural Implementation

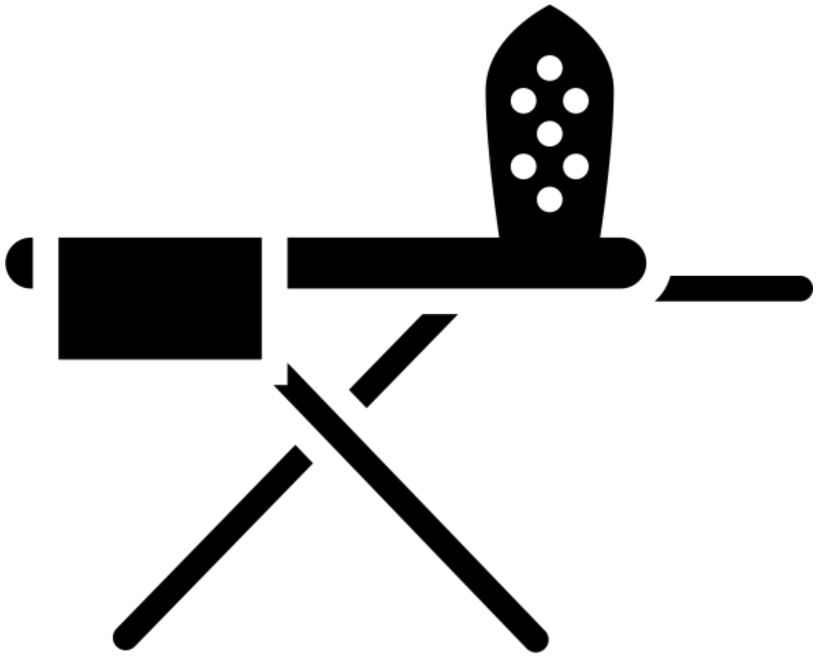
- Pipelining for Instruction-Level Parallelism (ILP)
- Pipelined microarchitecture design sketch
- Control hazards
- Branch prediction for dealing with control hazards

# Pipelining in the abstract: A laundry efficiency problem



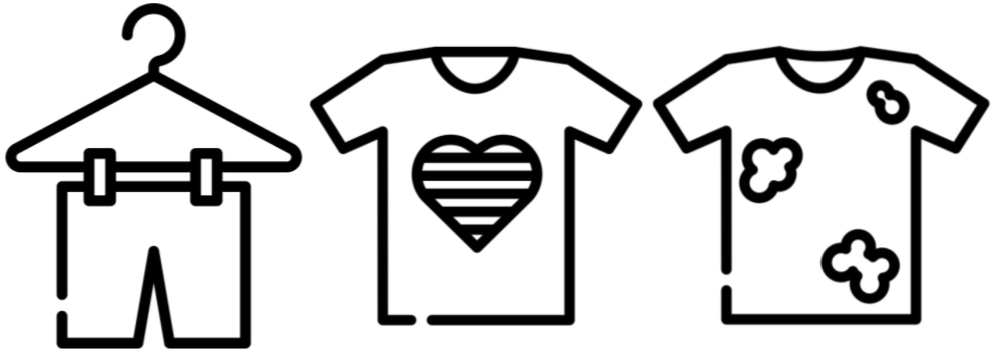




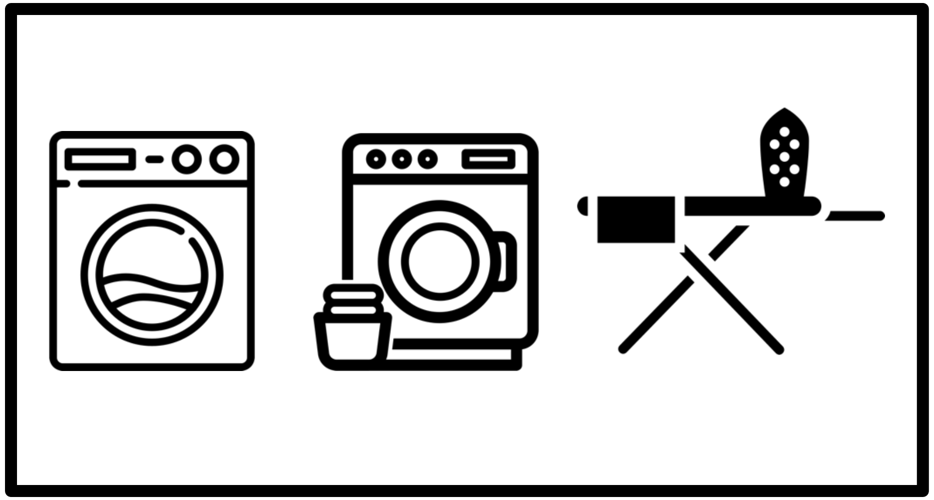


Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

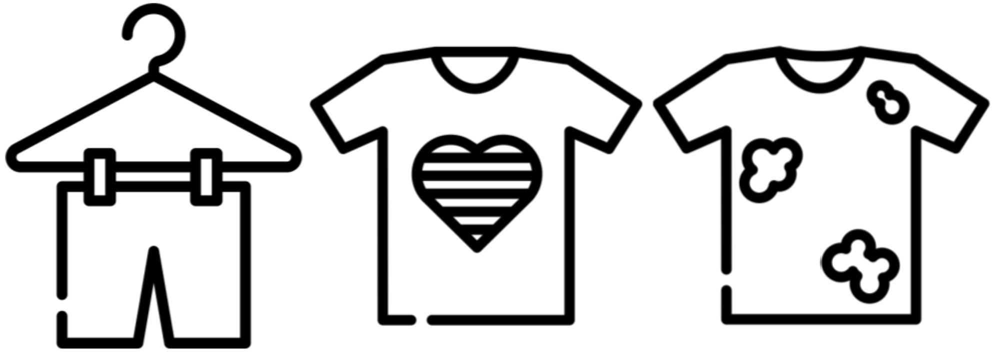


**Private Laundry Room Model: only one person at a time allowed in laundry room**

**Done being washed**

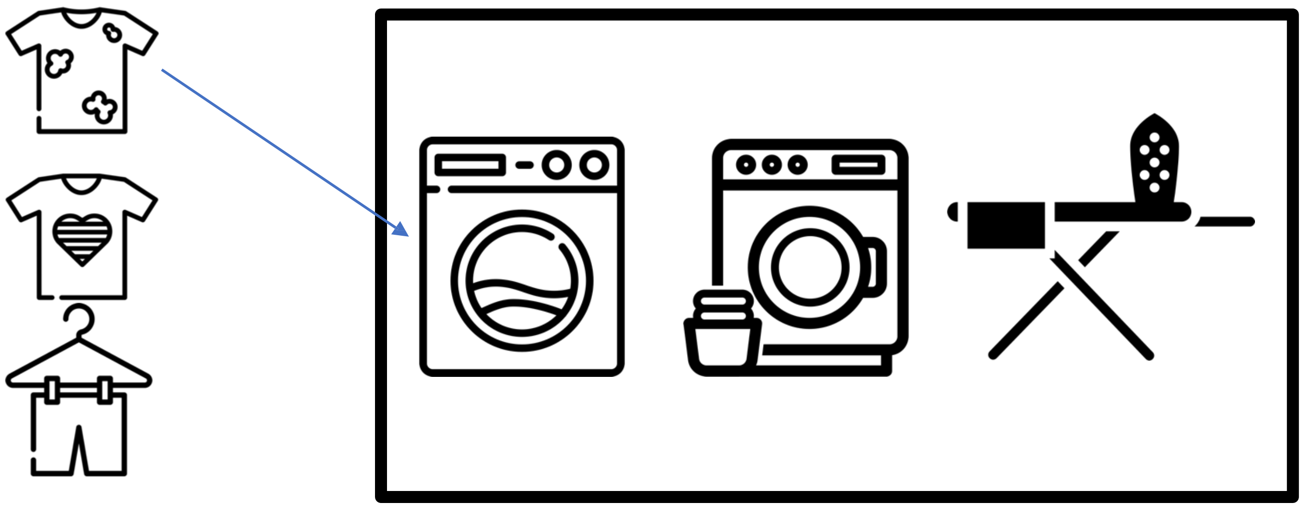
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract:
# A laundry efficiency problem



**To be washed**

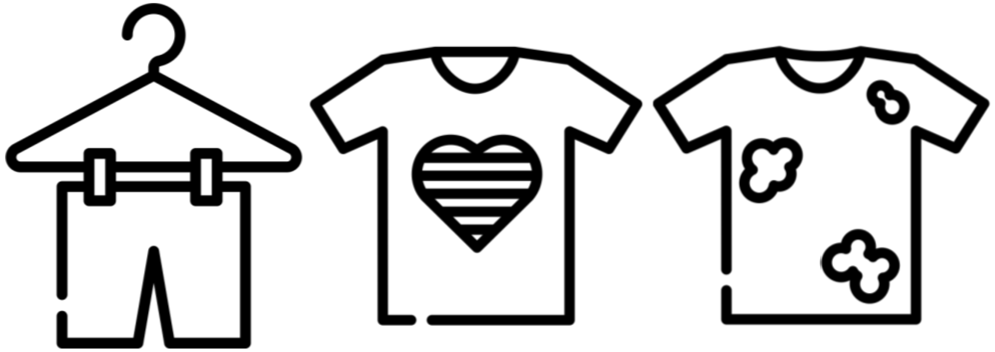
**Done being washed**



**Time = 0**

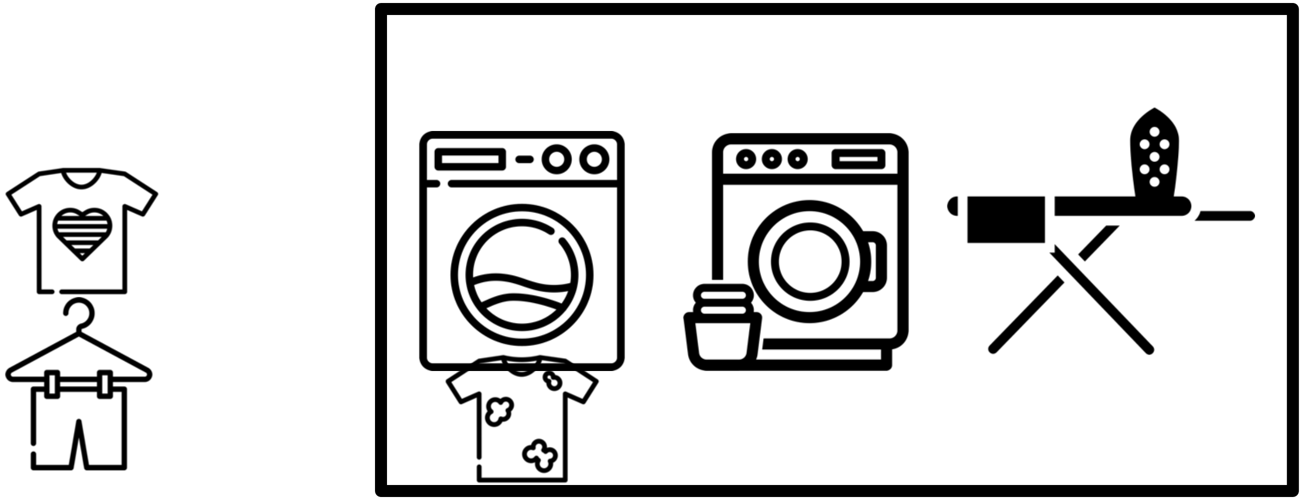
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

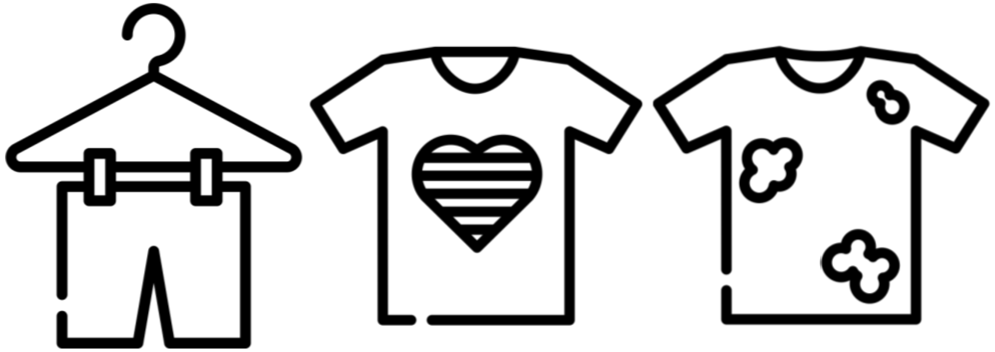
**Done being washed**

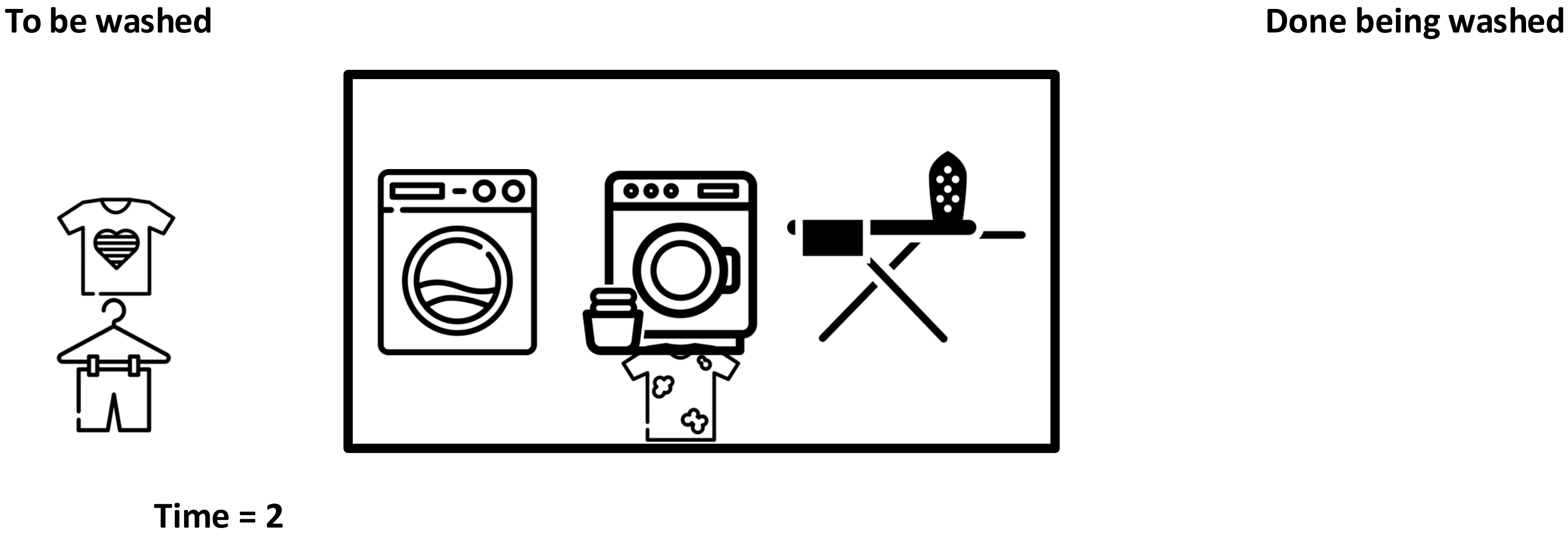


**Time = 1**

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

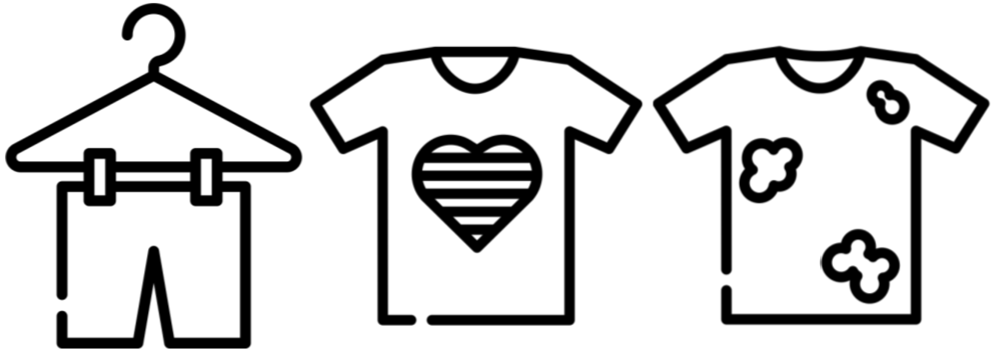
# Pipelining in the abstract: A laundry efficiency problem





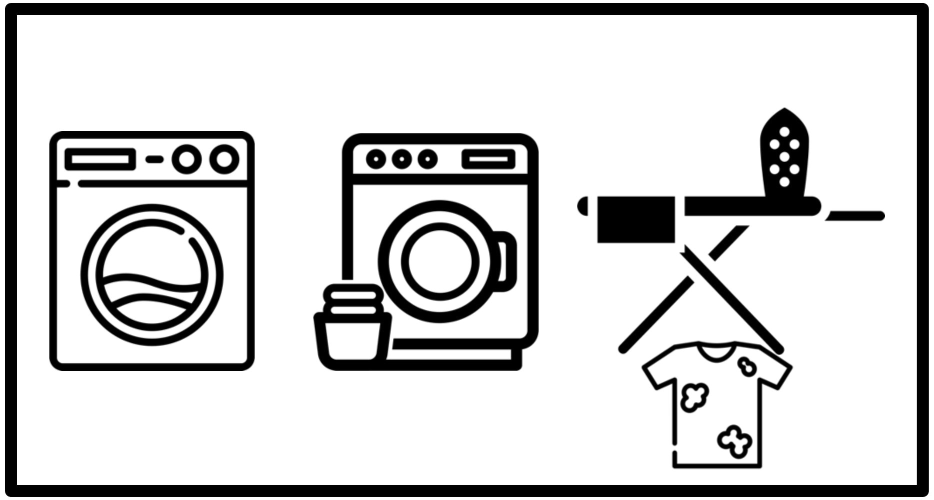

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

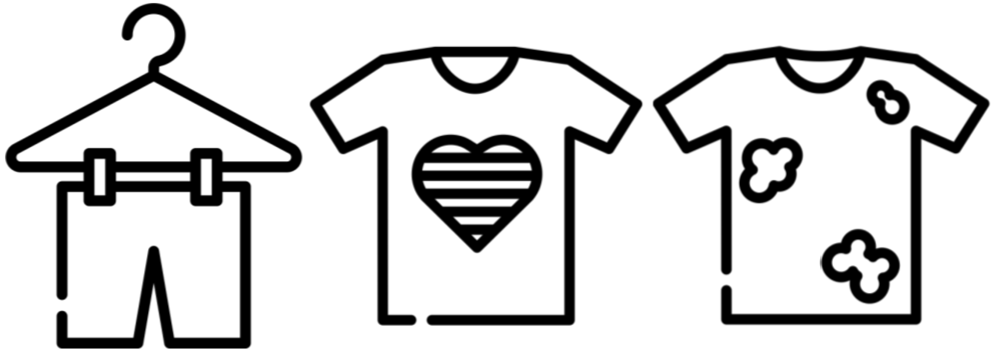
**Done being washed**



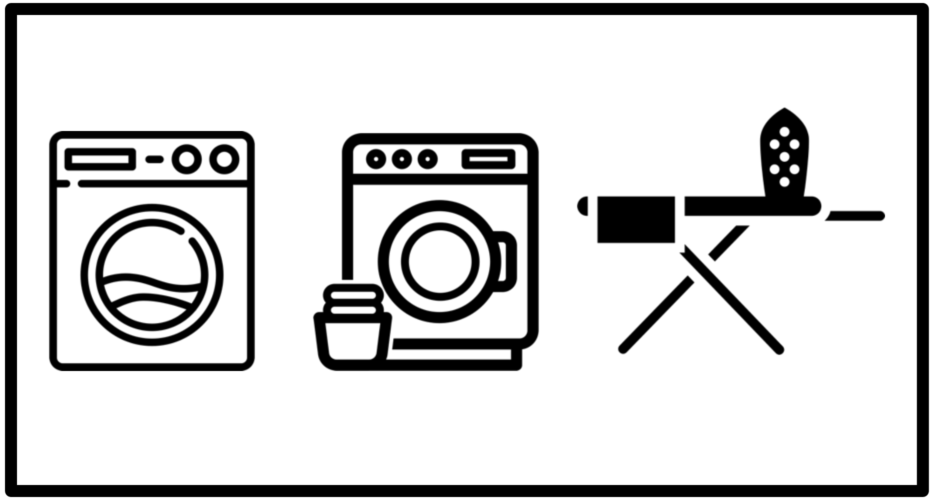
**Time = 3**

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

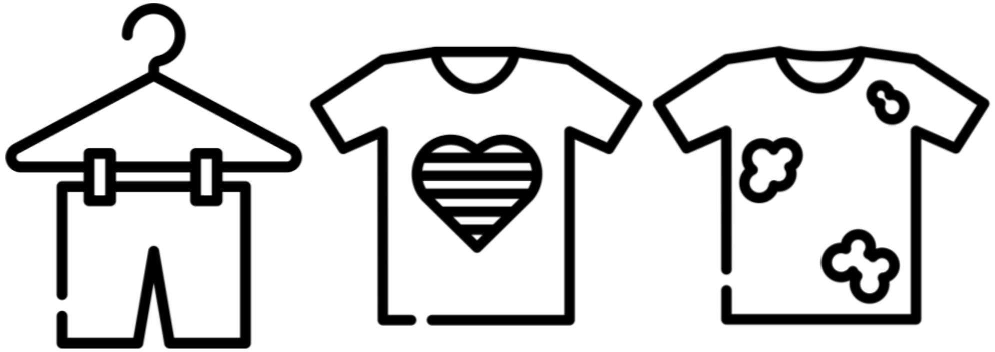


**Done being washed**

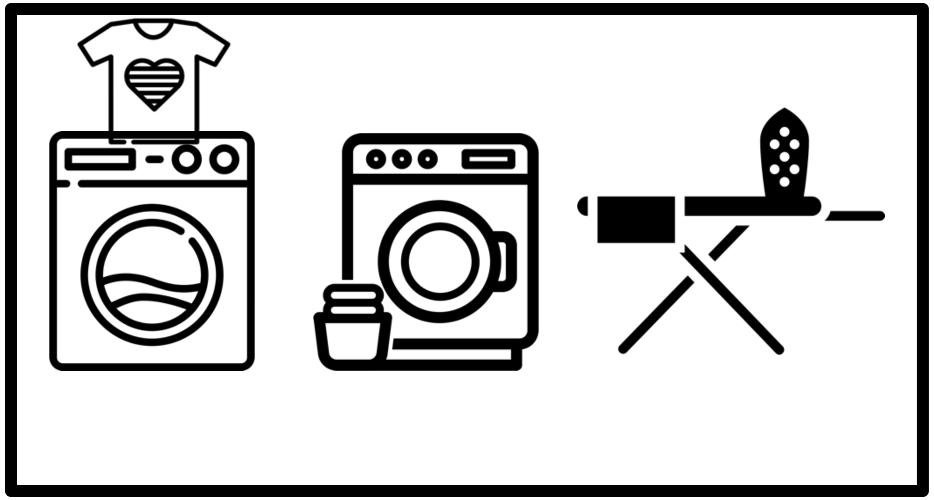
**Time = 4**

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract:
# A laundry efficiency problem



**To be washed**

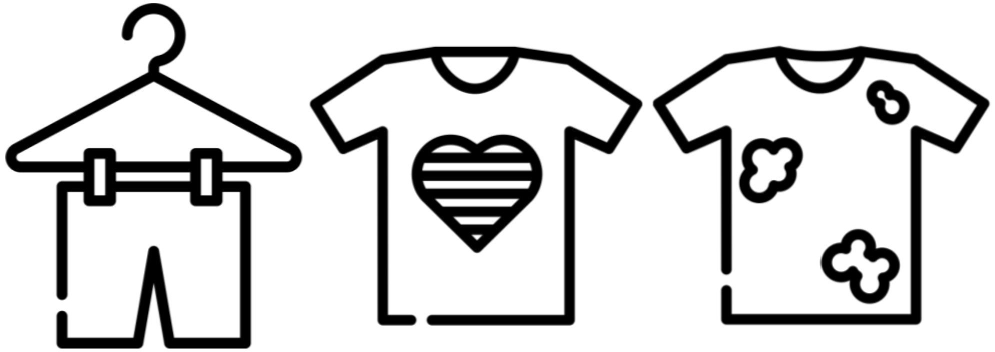


**Done being washed**

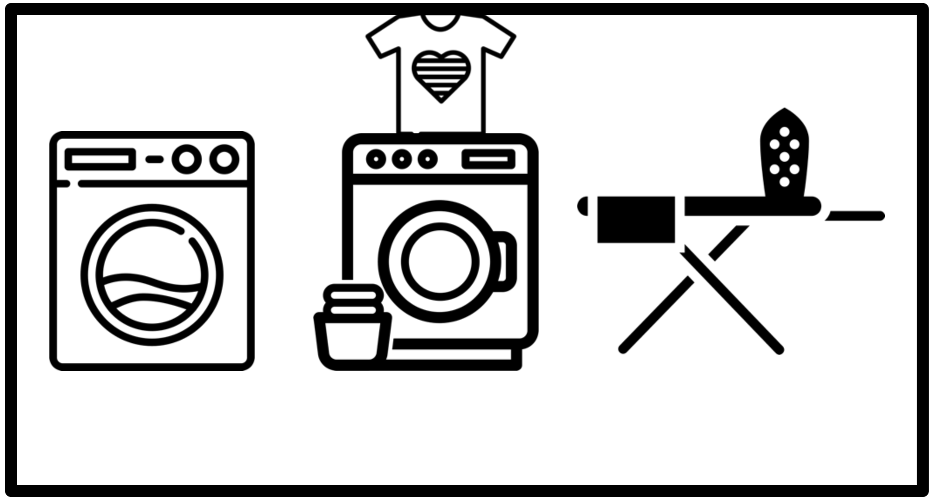
**Time = 5**

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract:
# A laundry efficiency problem



**To be washed**

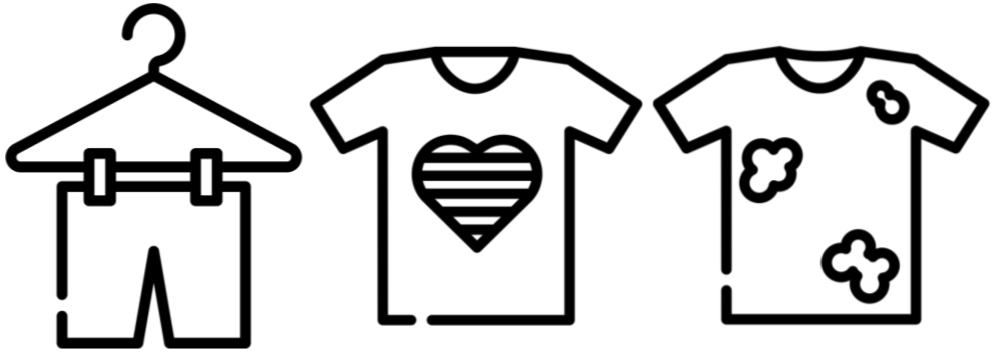


**Done being washed**

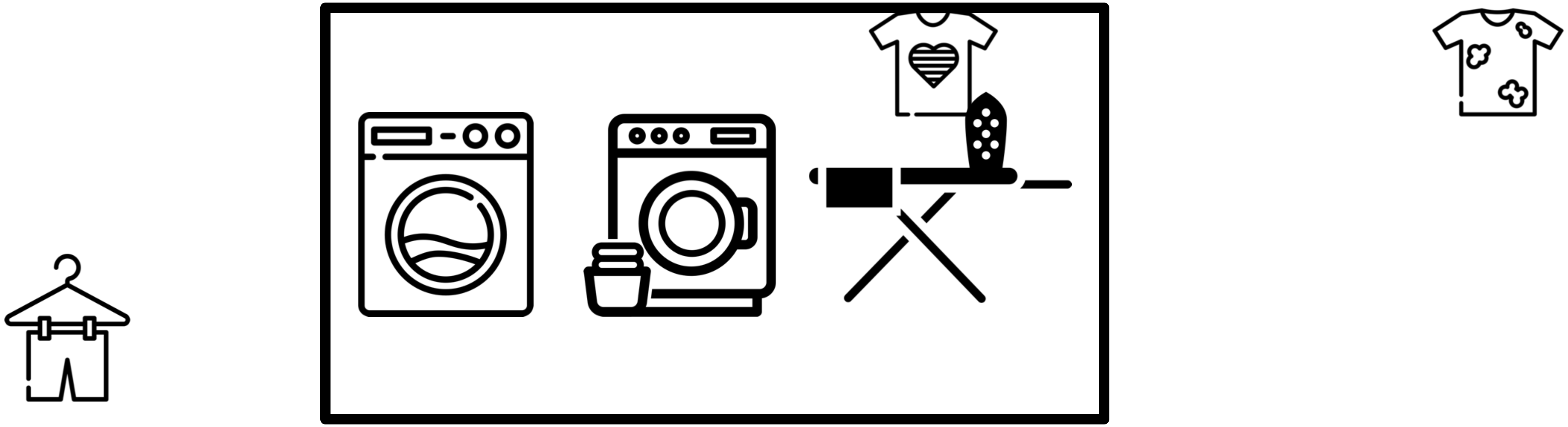
**Time = 6**

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract:
# A laundry efficiency problem



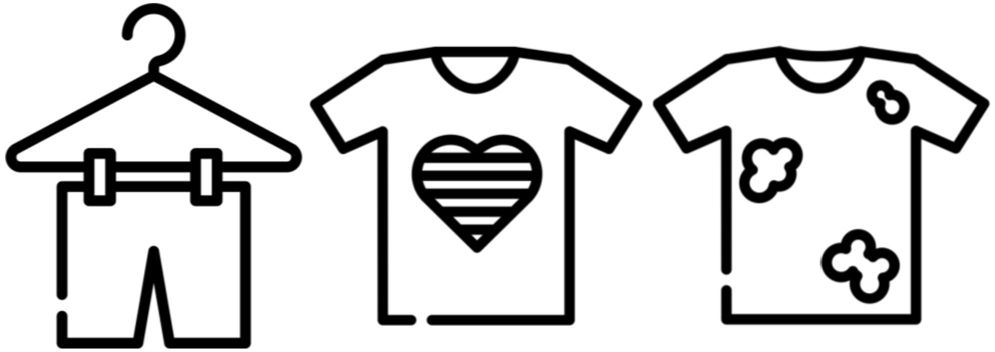
**To be washed** **Done being washed**



**Time = 7**

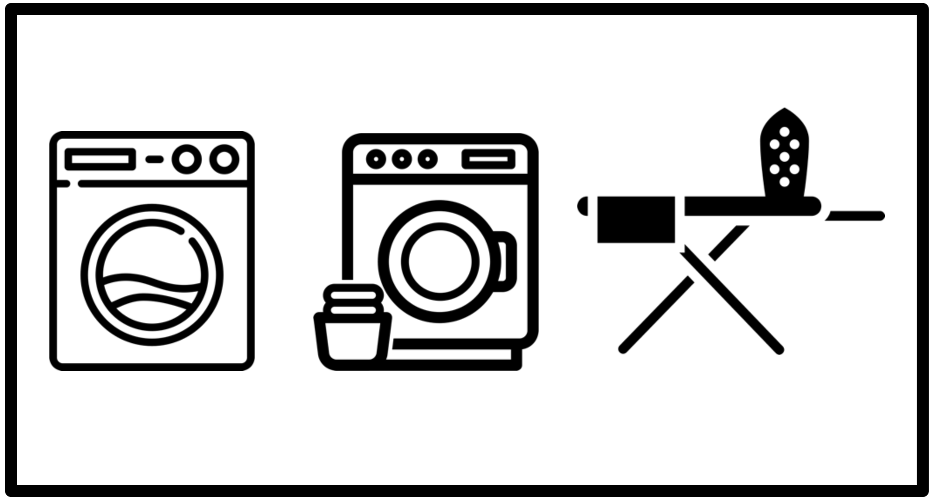
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

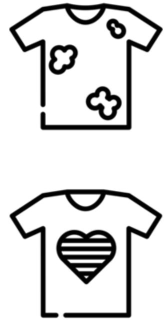
# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

**Done being washed**

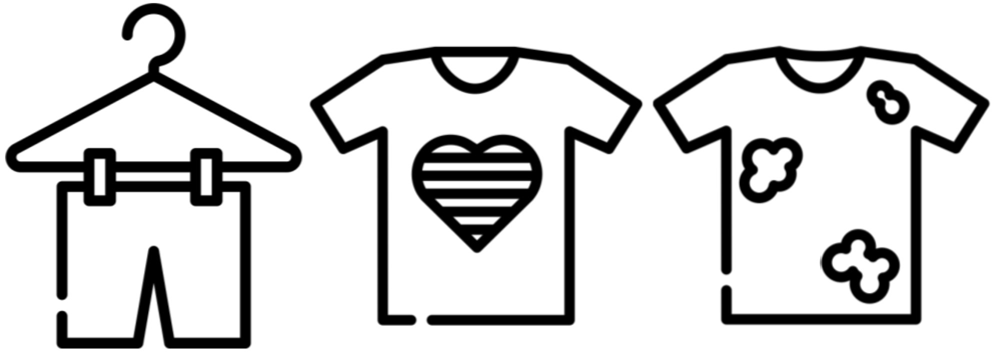




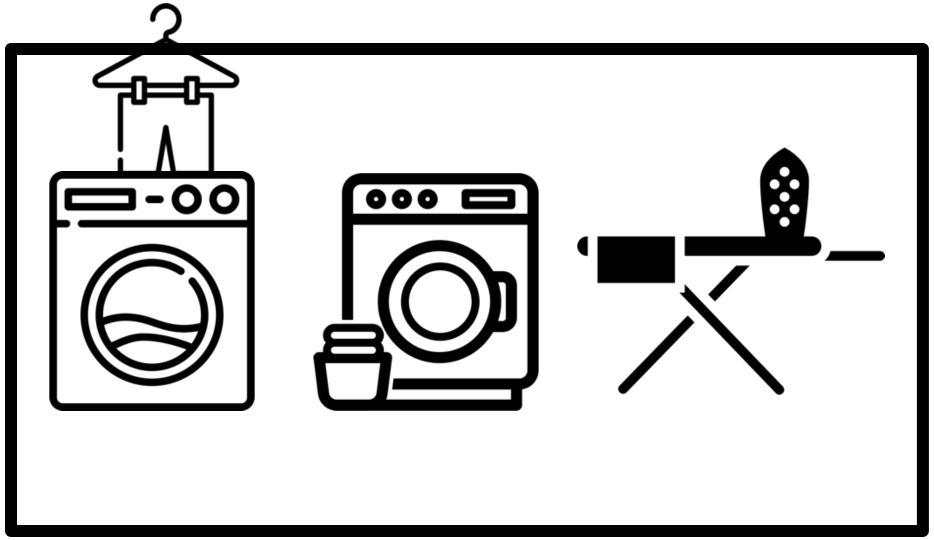
**Time = 8**

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

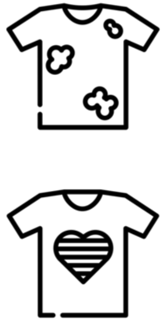
# Pipelining in the abstract: A laundry efficiency problem



**To be washed**



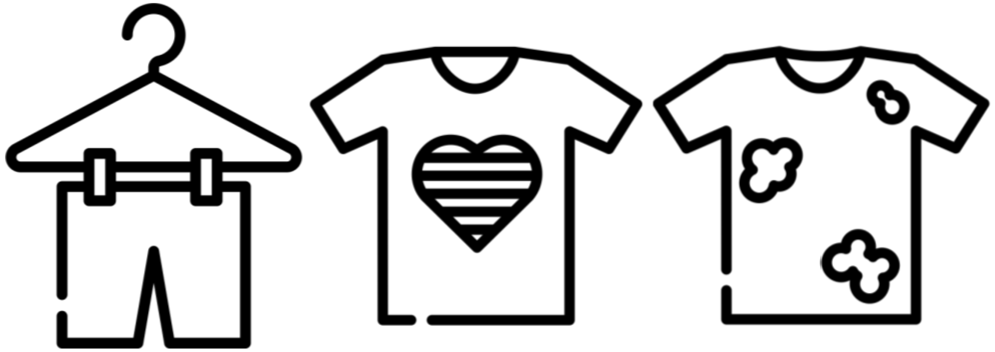
**Done being washed**



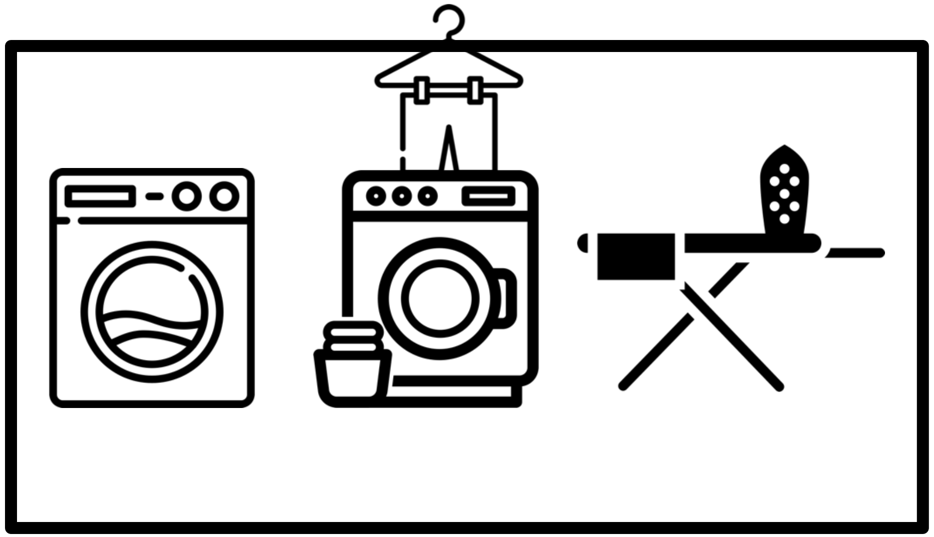
**Time = 9**

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

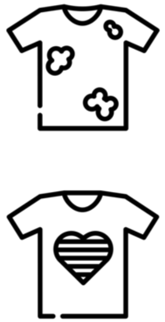
# Pipelining in the abstract: A laundry efficiency problem



**To be washed**



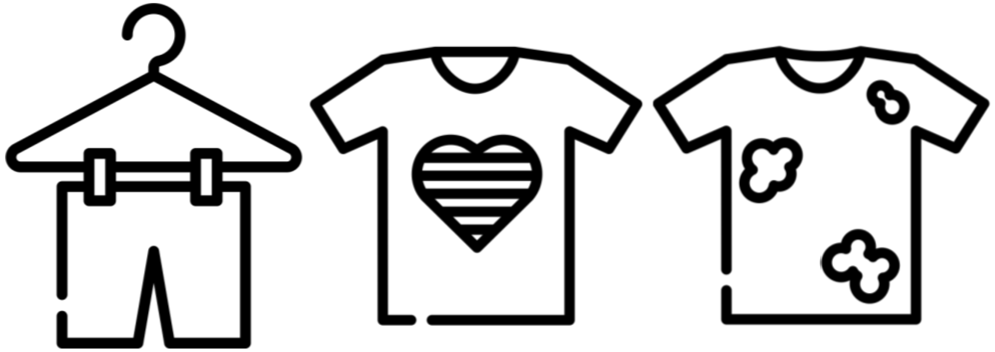
**Done being washed**



**Time = 10**

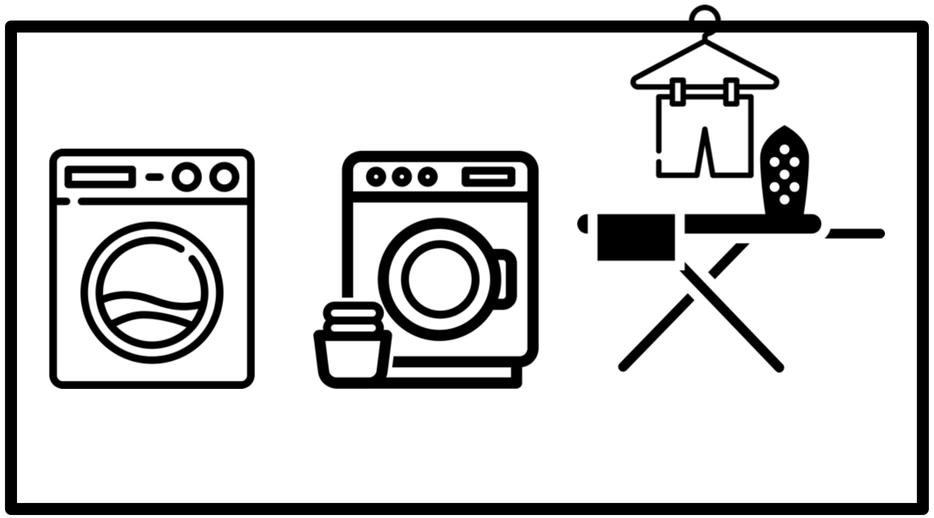
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

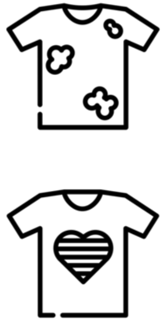
# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

**Done being washed**

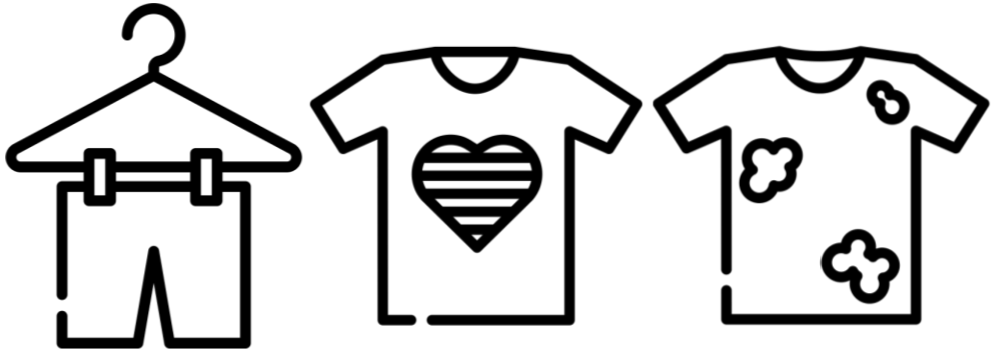




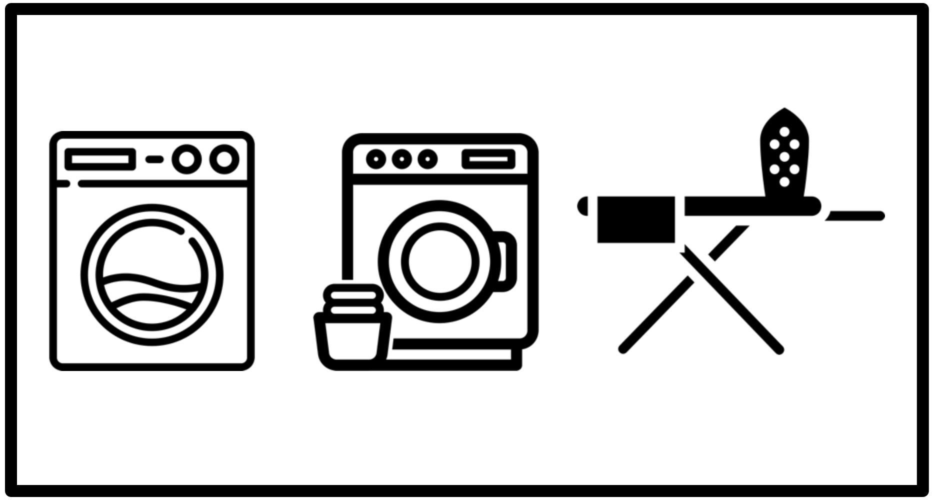
**Time = 11**

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

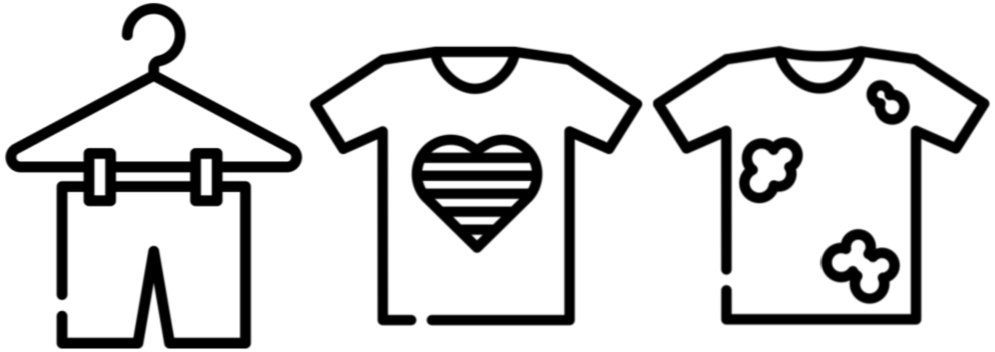


**Done being washed**

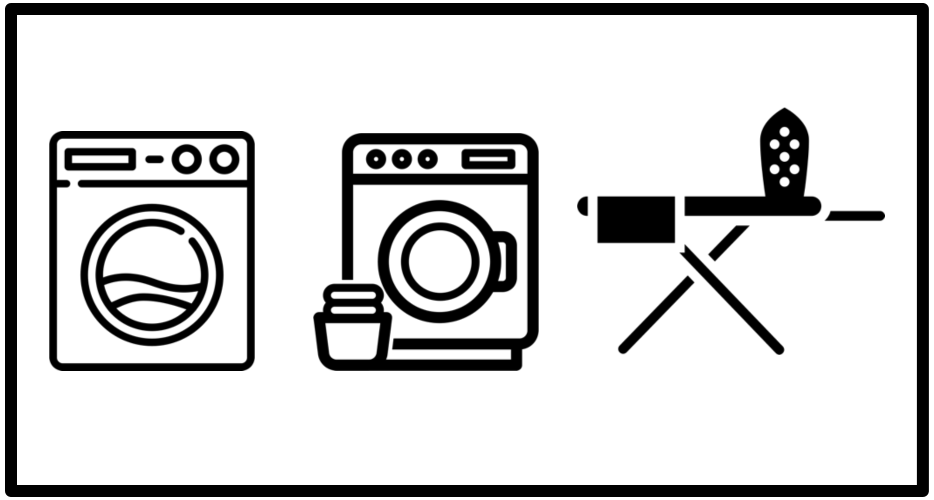
**Time = 12**

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

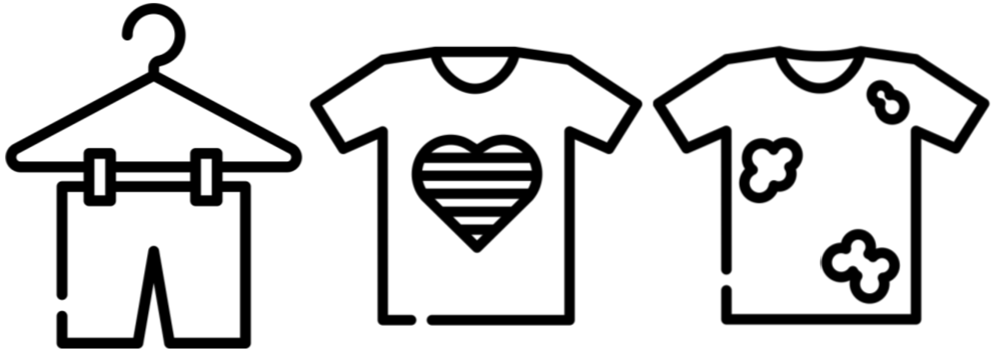


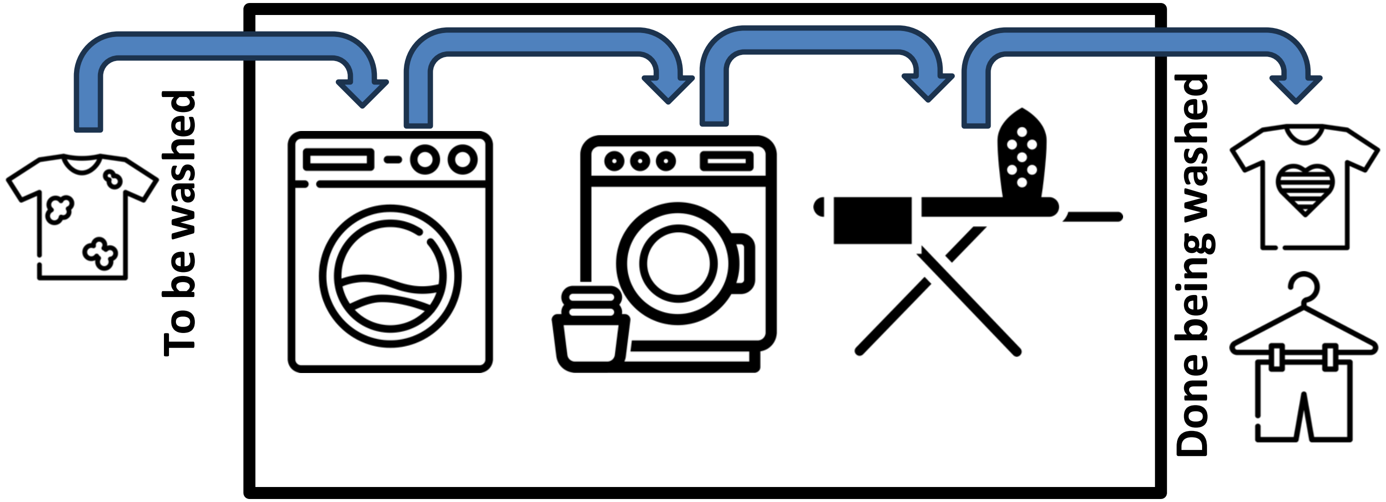
**Done being washed**

**Analysis: With 3 resources ( , , ) and 3 units of work ( , , ) our laundry took 12 units of time**

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem






**Analysis: With 3 resources ( , , ) and 3 units of work ( , , ) our laundry took 12 units of time**

**12 units of time?**

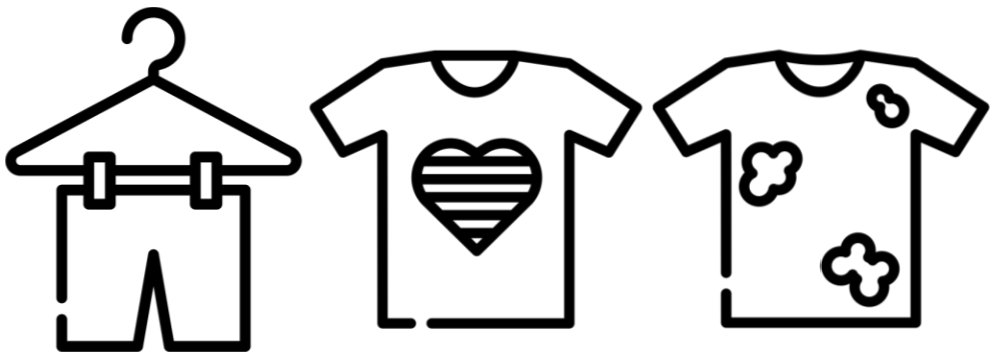
Why 12 units of time vs 9 units of time overall?

Why 4 units of time per load vs 3 units of time?

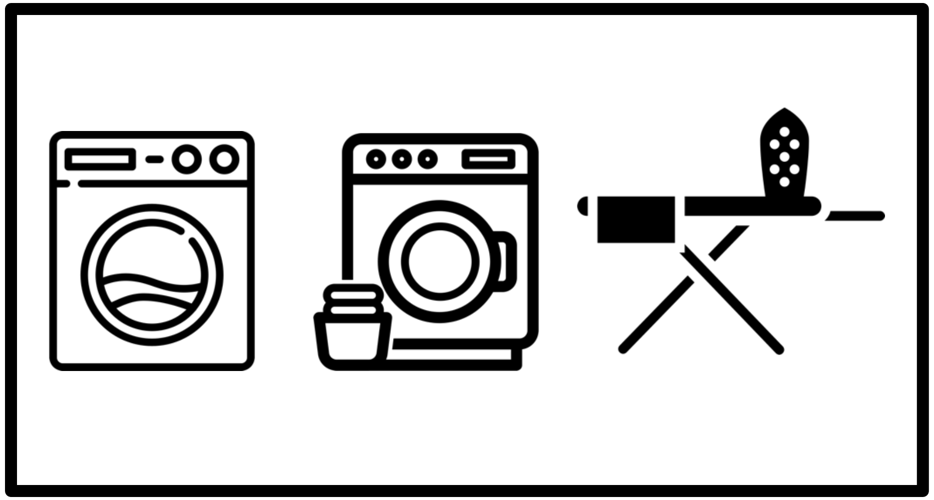
- Processors and their workings are triggered devices.
- It takes 4 triggers for the dirty laundry pile to be washed, dried, folded, and available.

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

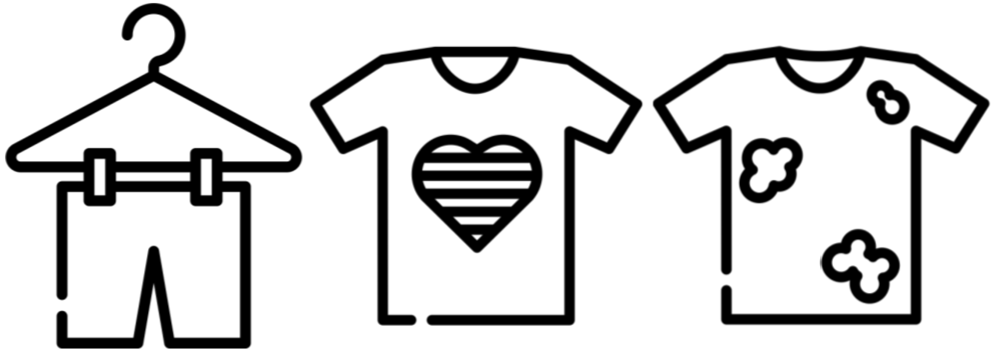


**Done being washed**

**Let’s redesign our laundry room to make it more efficient**

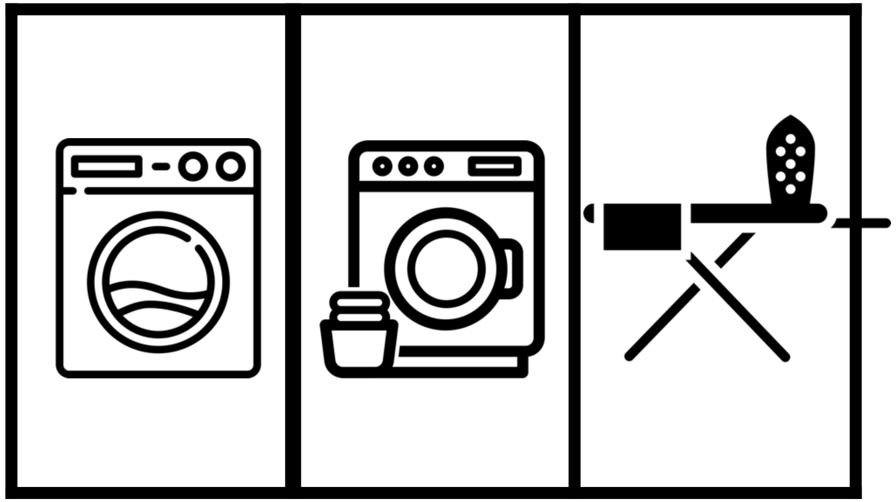
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

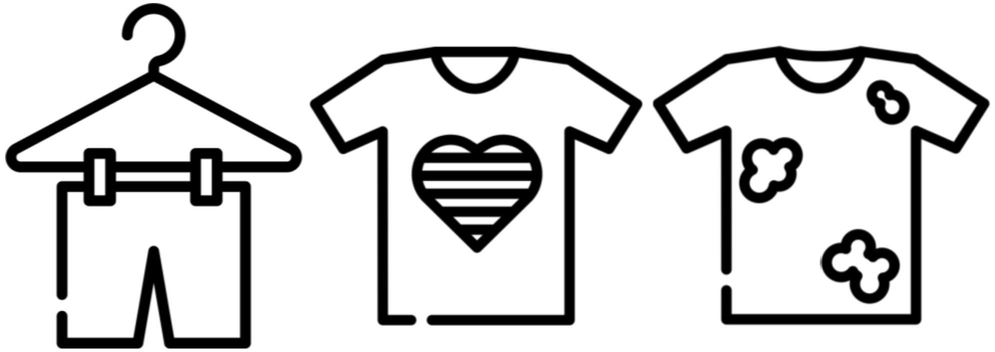
**Done being washed**



**Shared Laundry Room: single laundry task uses single machine at a time, not entire room. Multiple roommates allowed in at once.**

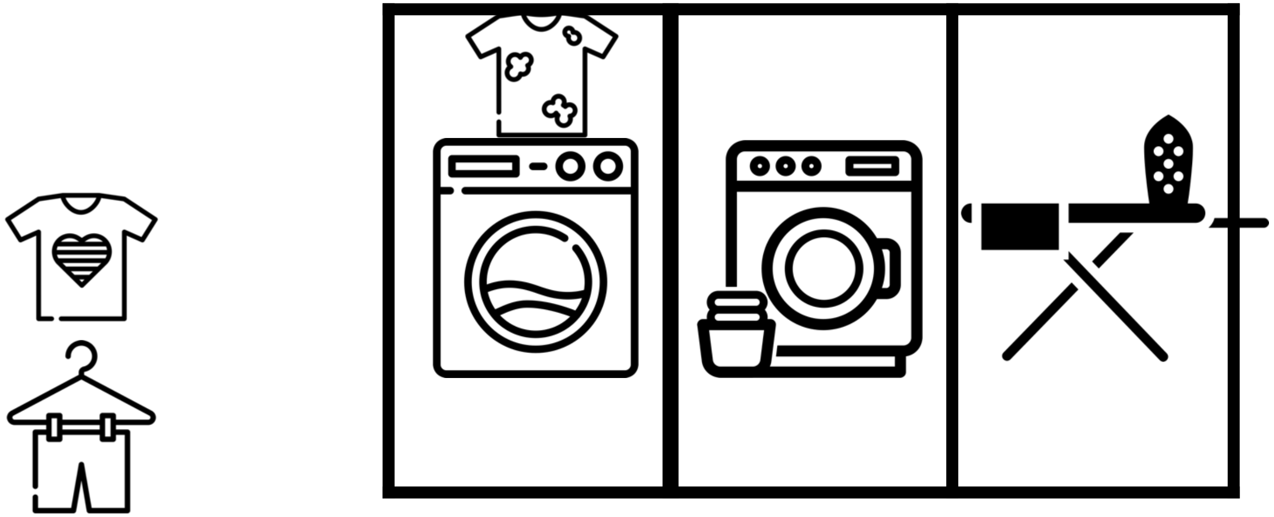
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

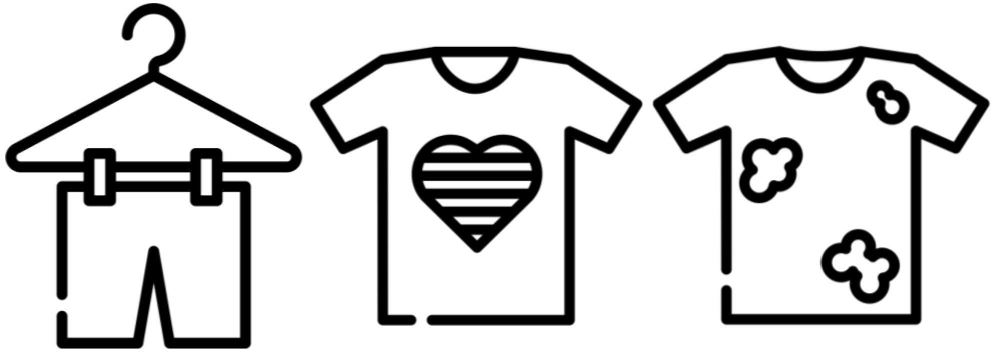
**Done being washed**



**Time = 1**

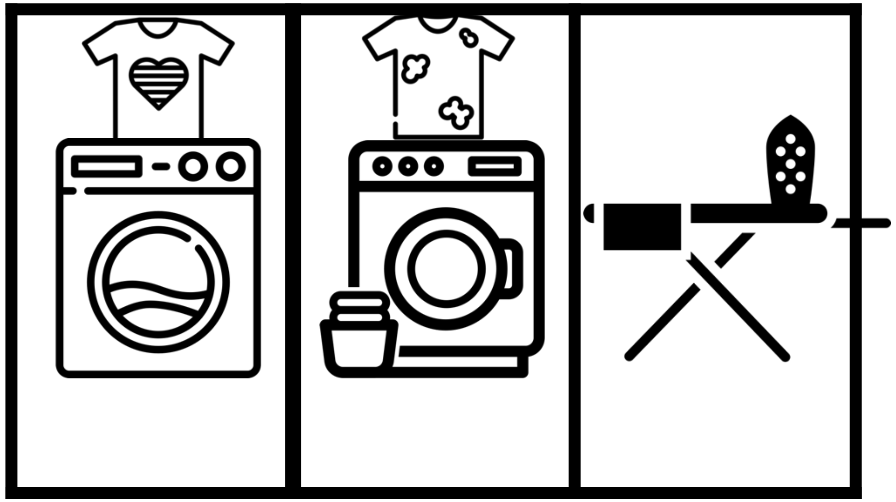
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

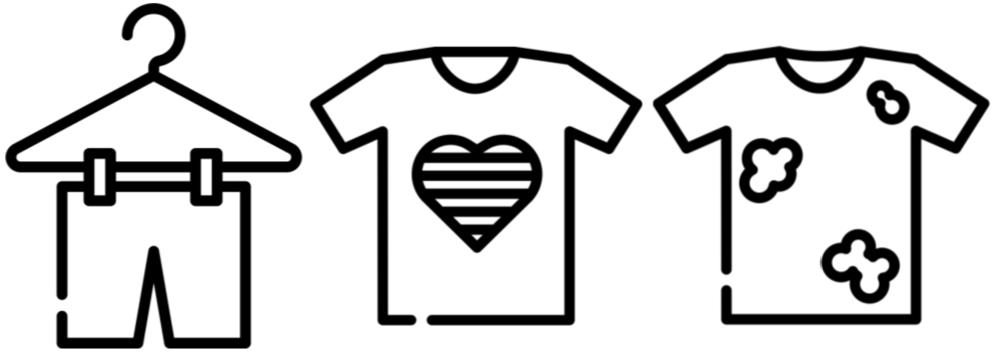
**Done being washed**



**Time = 2**

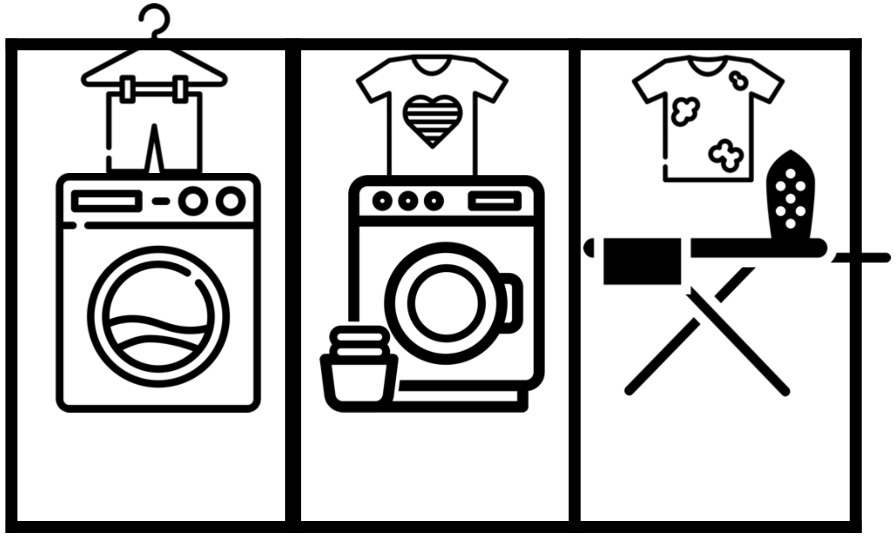
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

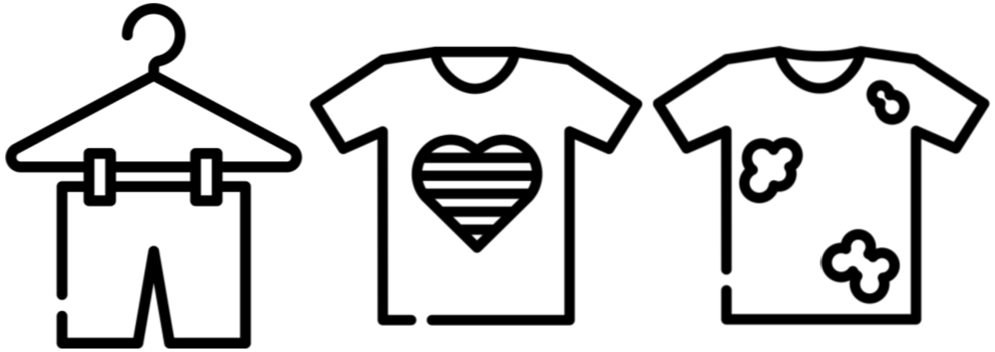
**Done being washed**



**Time = 3**

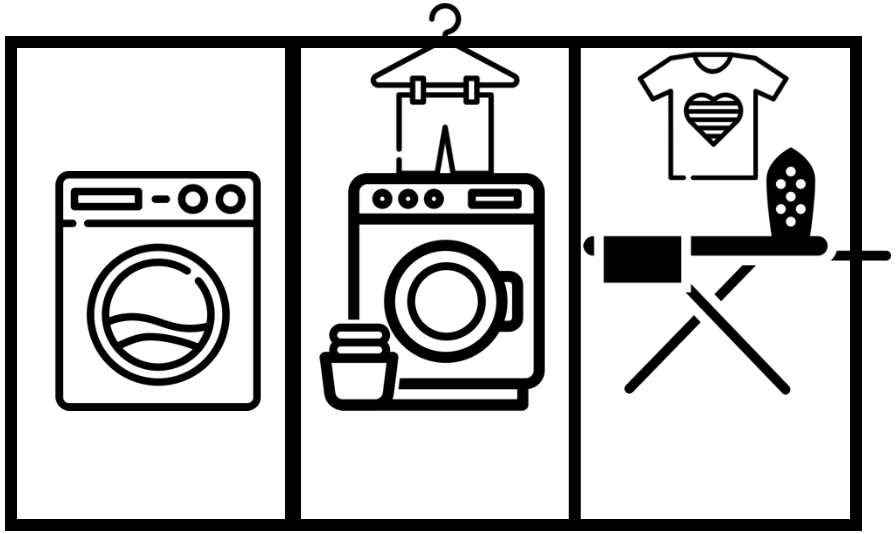
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

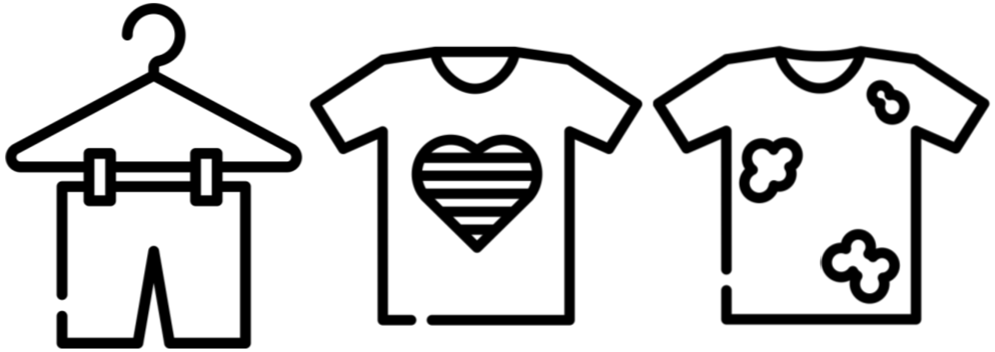
**Done being washed**



**Time = 4**

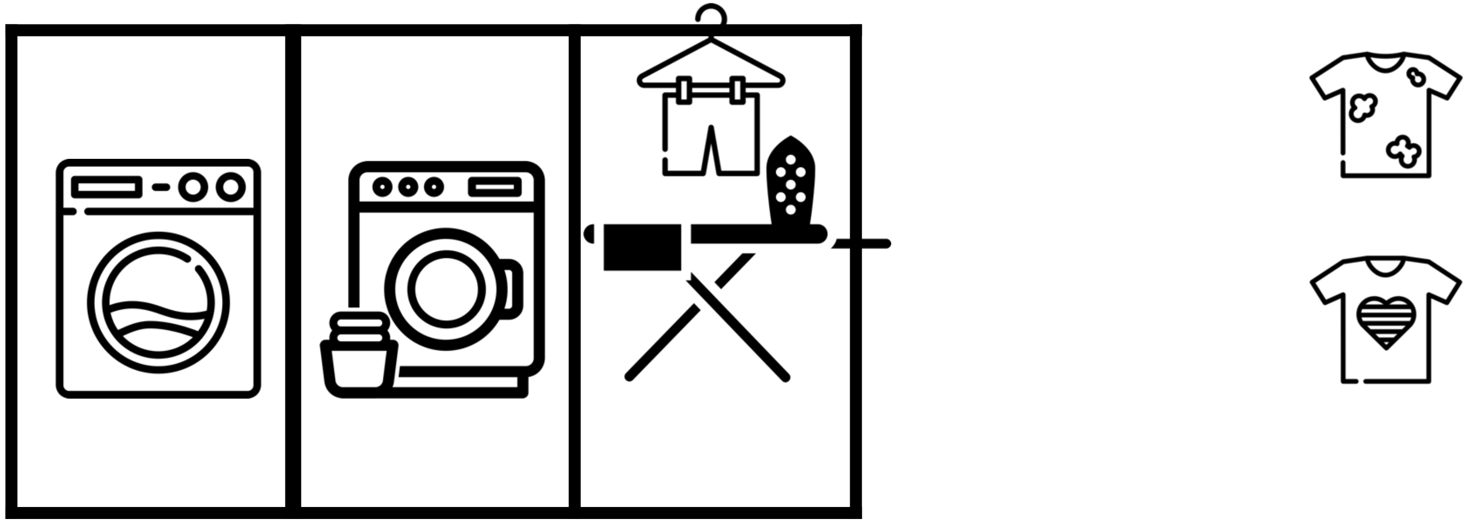
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

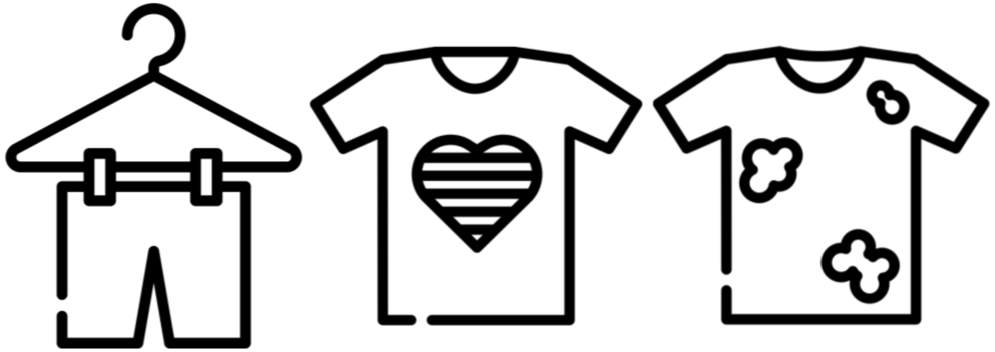
**Done being washed**



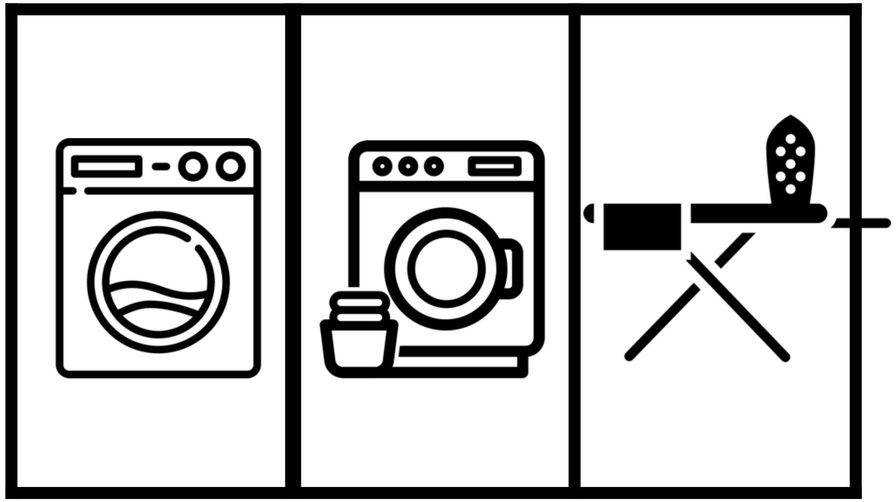
**Time = 5**

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**

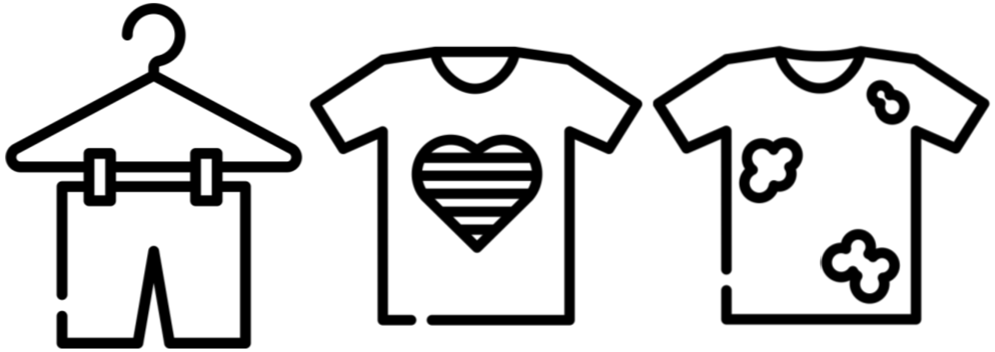


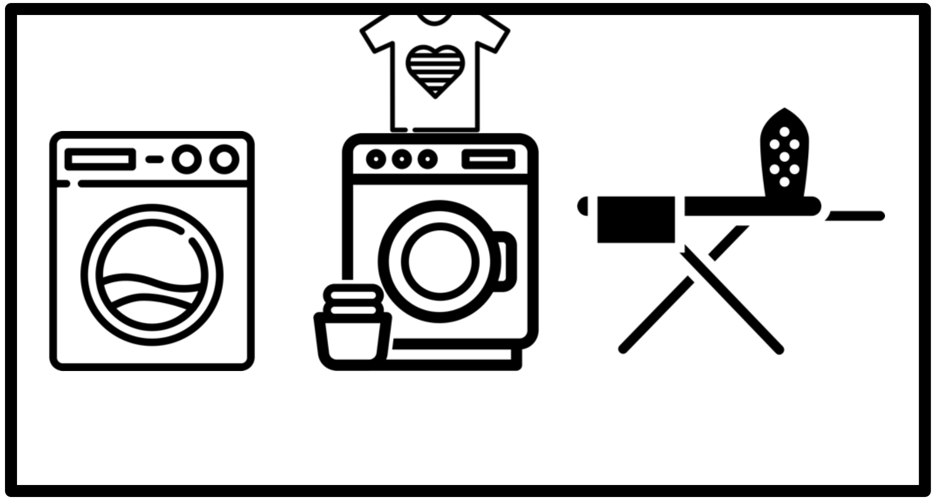
**Done being washed**

**Analysis: With 3 resources ( , , ) and 3 units of work ( , , ) our laundry took 6 units of time**

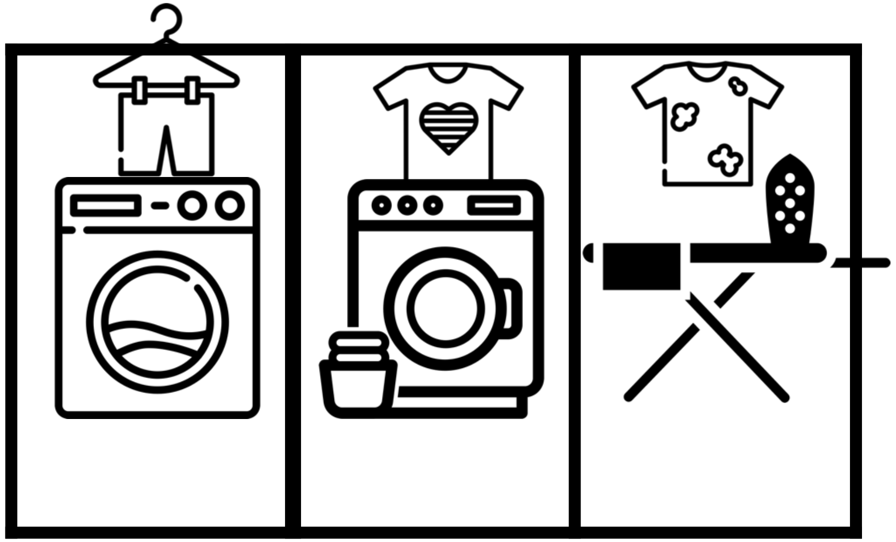
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem





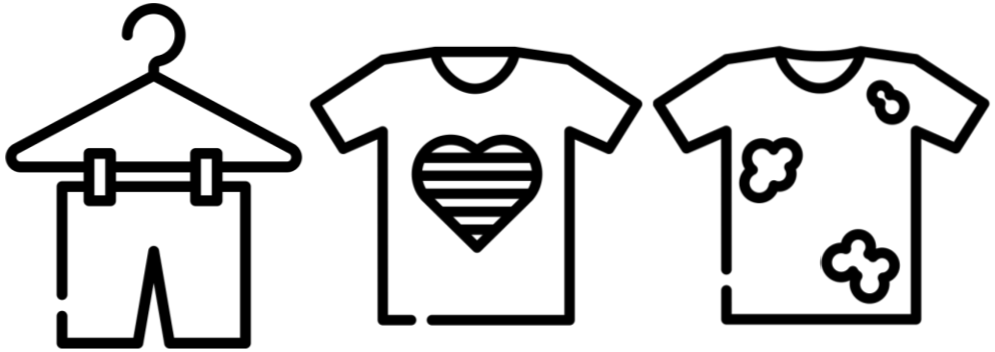
**Vs.**

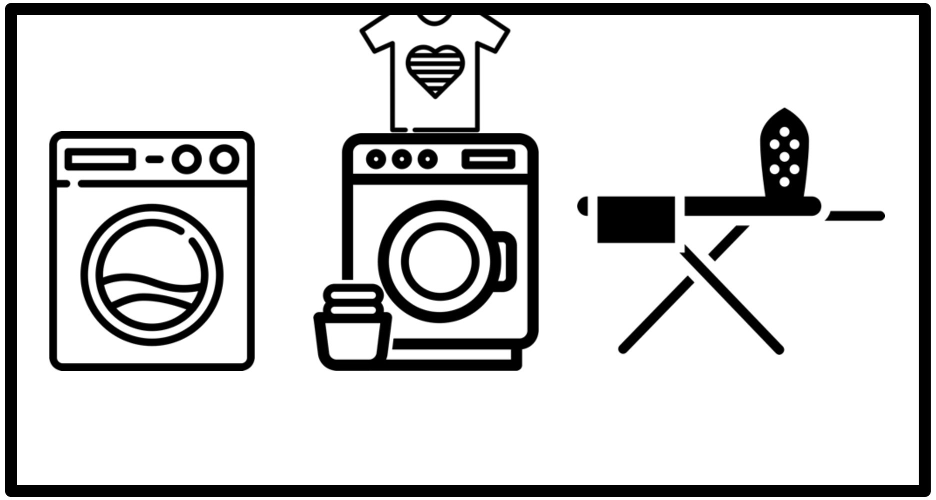


**General observations about private laundry room model vs. shared laundry room model?**

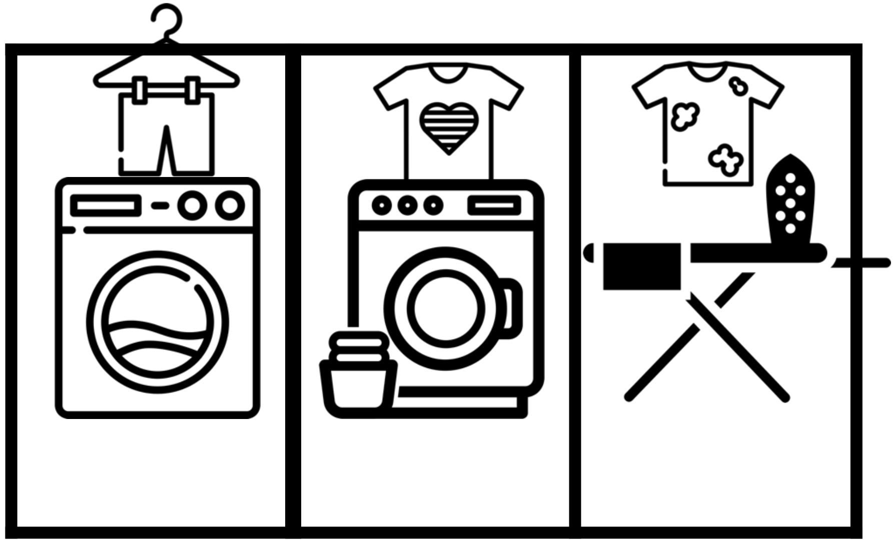
Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract:
# A laundry efficiency problem





**Vs.**

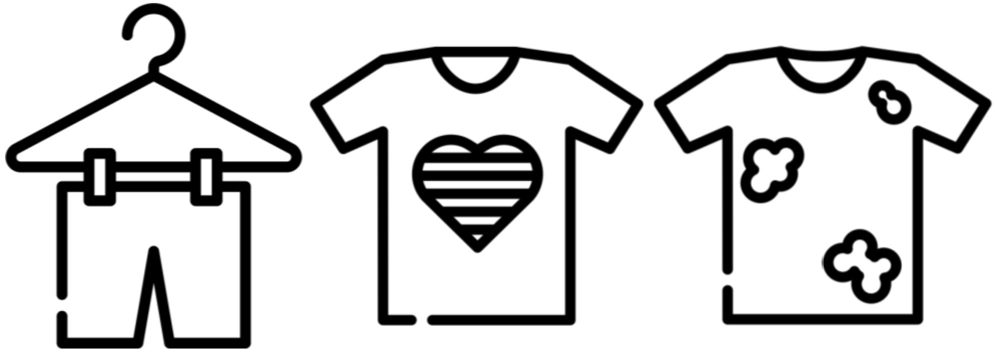


**General observations about private laundry room model vs. shared laundry room model?**

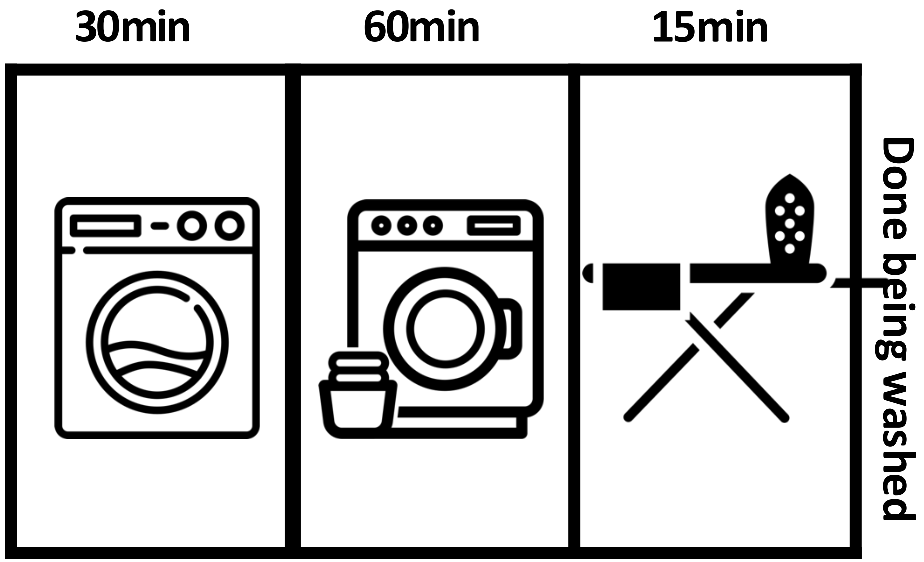
- Using machines *in parallel* in the shared laundry model
- At time step 3 (“steady state”) all machines are active
- Private model: always leaving 2/3 of laundry machines idle, despite laundry yet to wash!

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

# Pipelining in the abstract: A laundry efficiency problem



**To be washed**



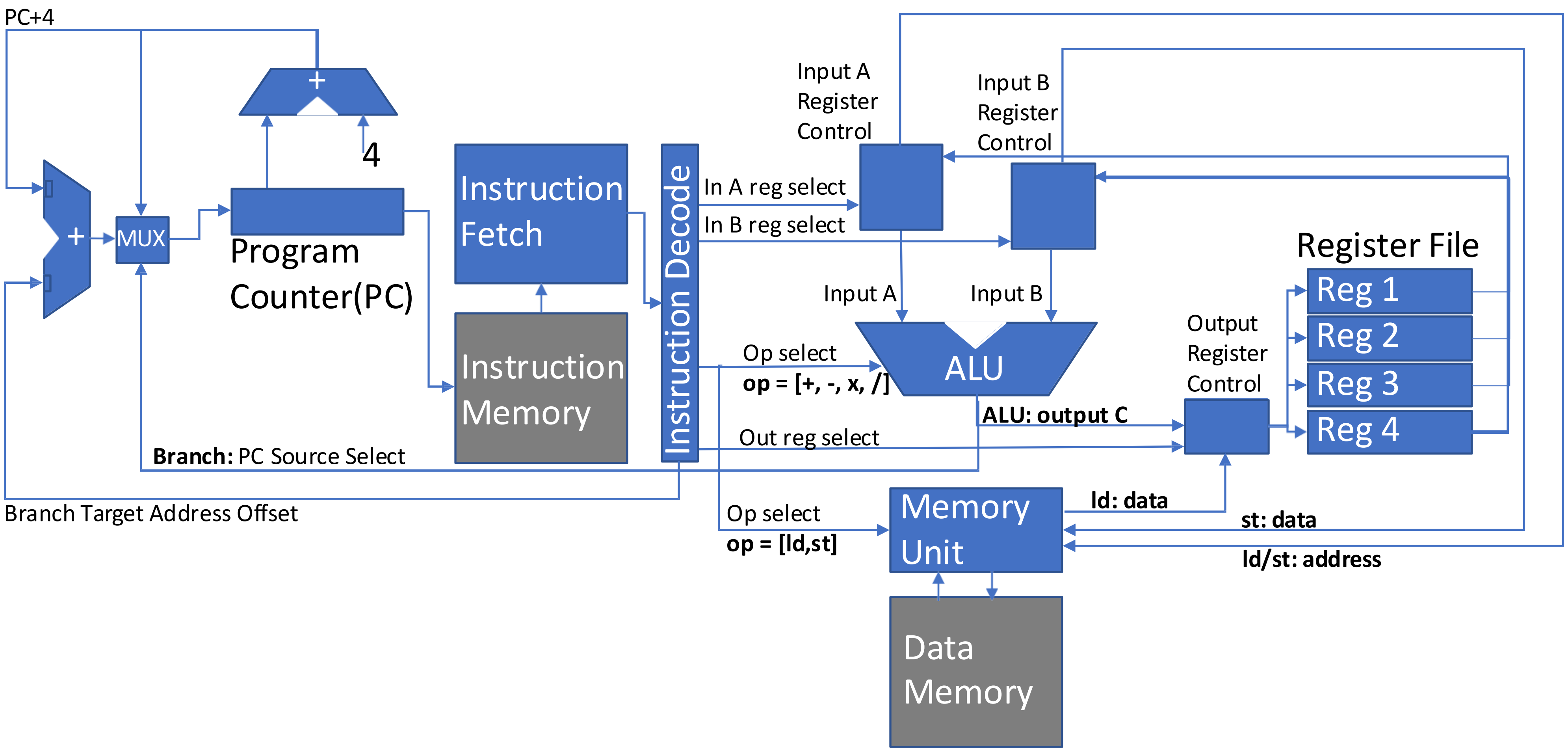

**Shared Laundry Room: single laundry task uses single machine at a time, not entire room. Multiple roommates allowed in at once.**

**Improving Pipeline Performance**

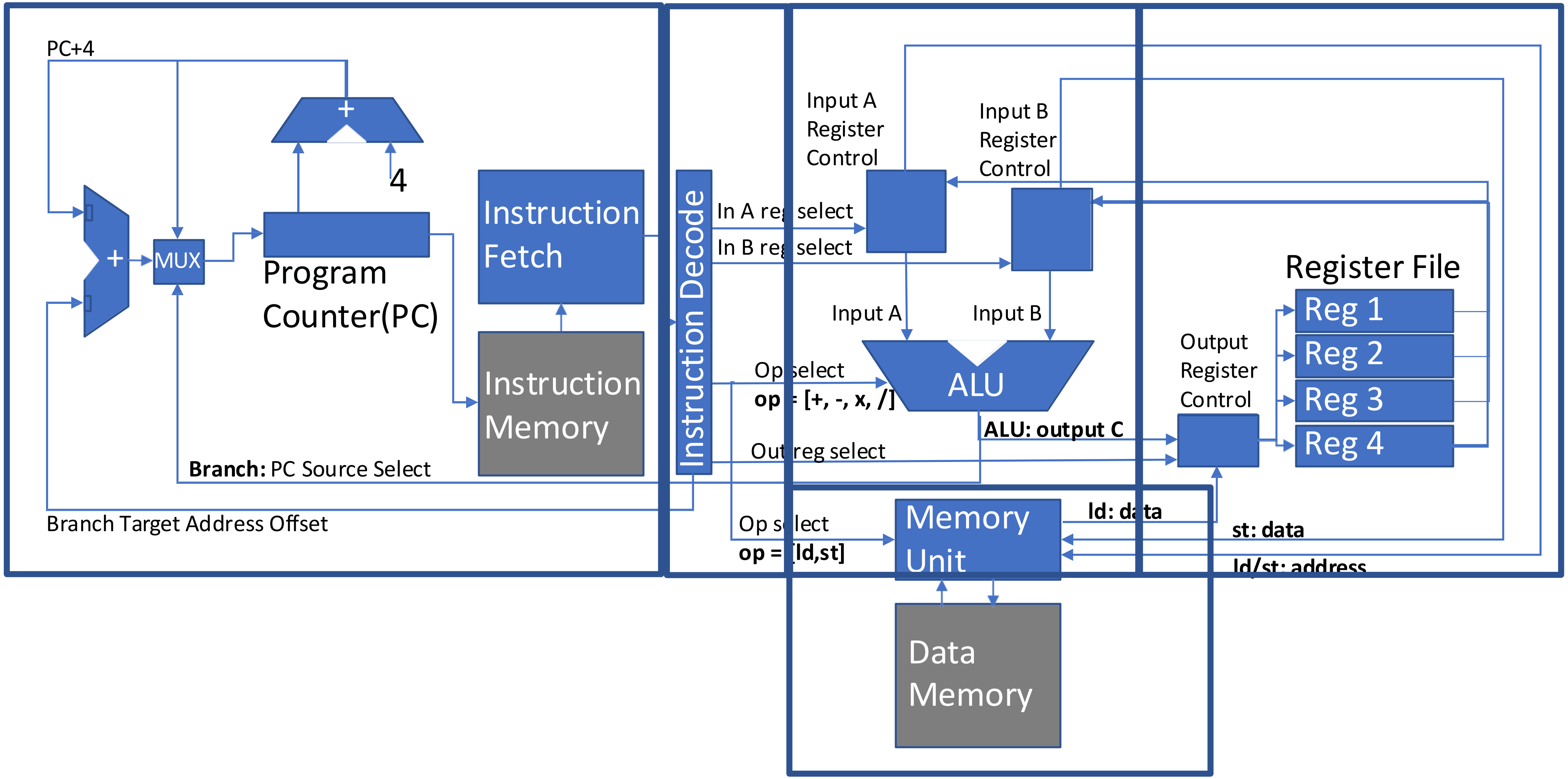
- If you could make washing take only 15 minutes what would be the impact upon throughput?
- What if you could make ironing take only 10 minutes?
- What if you could make drying take 45 minutes? Why is that different?
- *Hint: What (stage) limits the throughput? Why?*

Art attribution: Andrejs Kirma from the Noun ProjectKoson Rattanaphan from the Noun Project, Symbolon from the Noun Project

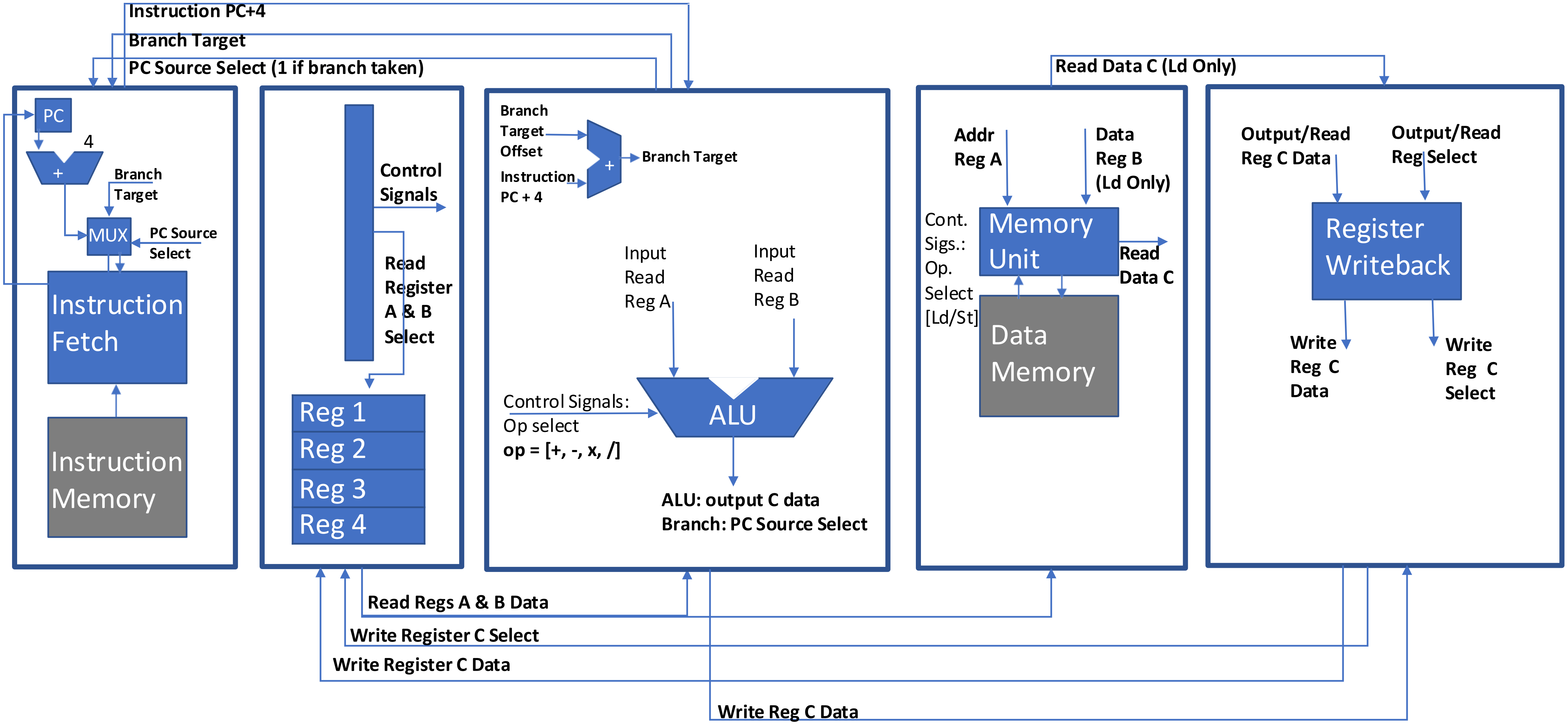
# Let’s do some grouping together of functionality



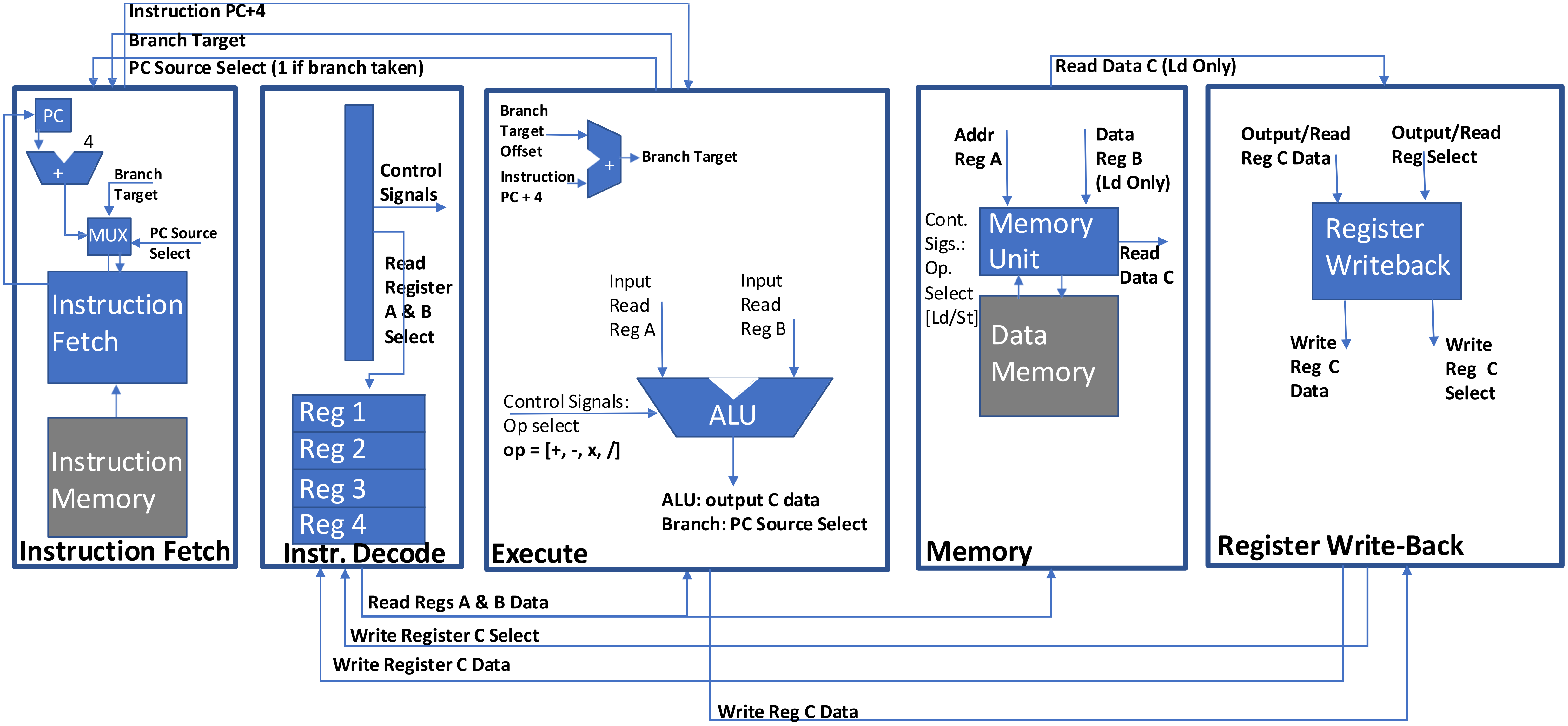
# Let’s do some grouping together of functionality



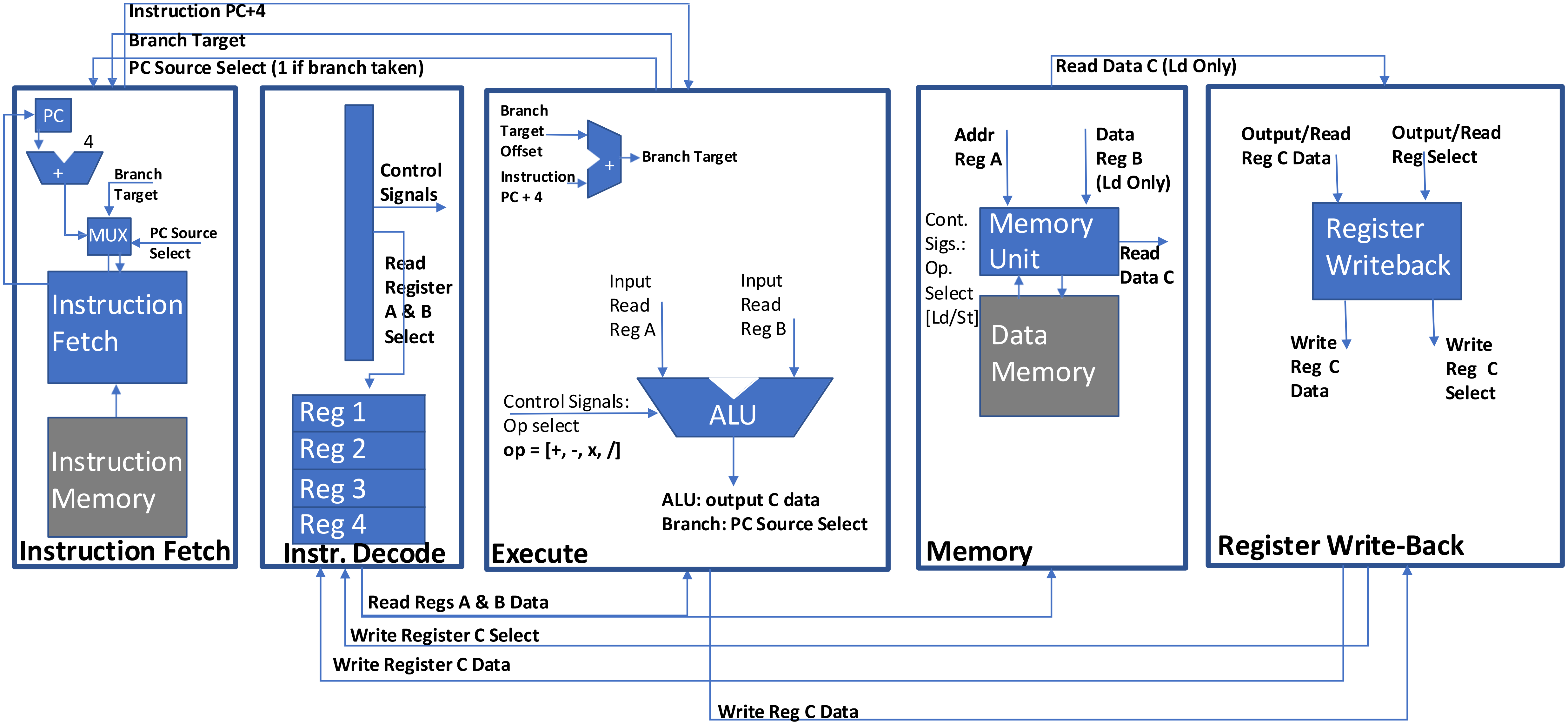
# Let’s do some grouping together of functionality



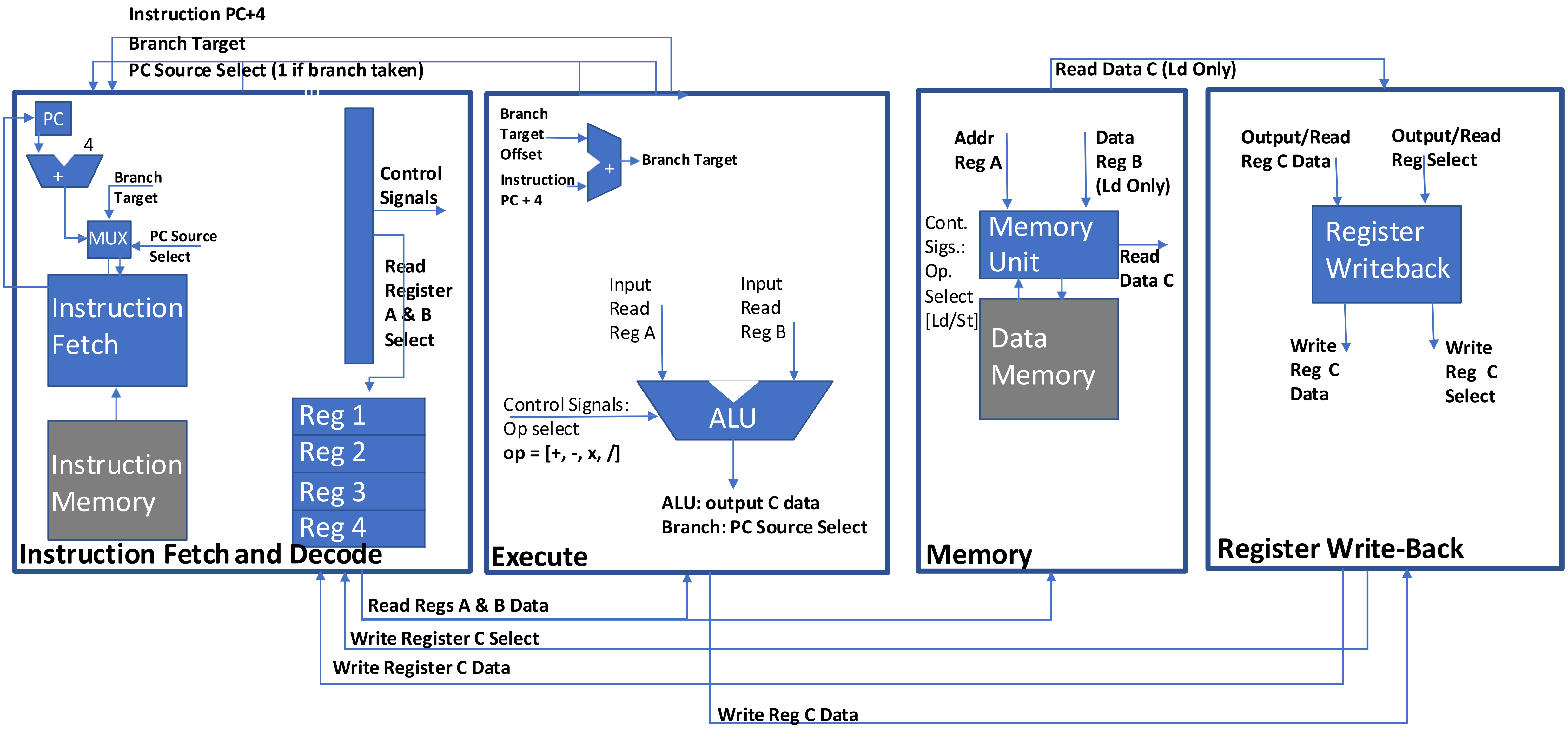
# A Simple 5-Stage Pipelined Processor Datapath



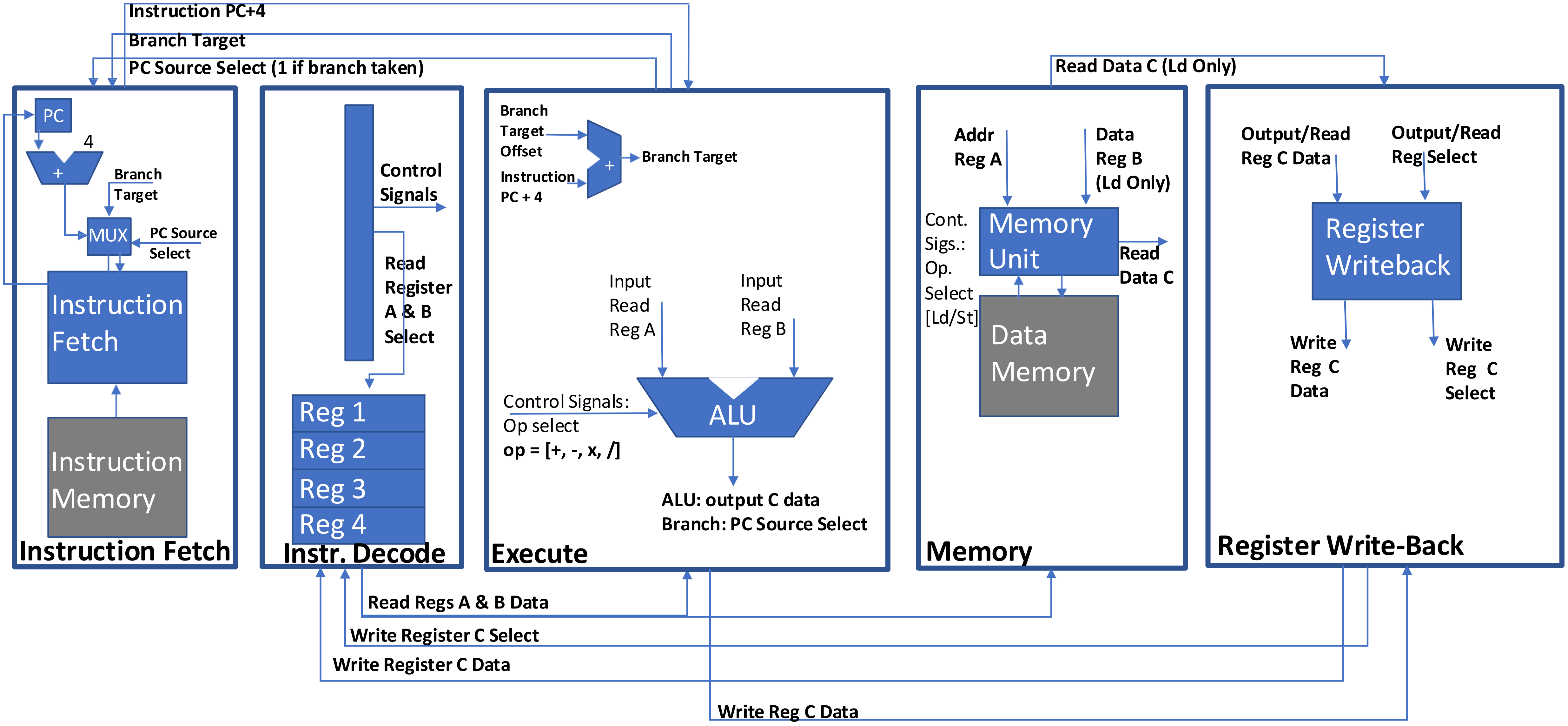
# What about an alternative decomposition?



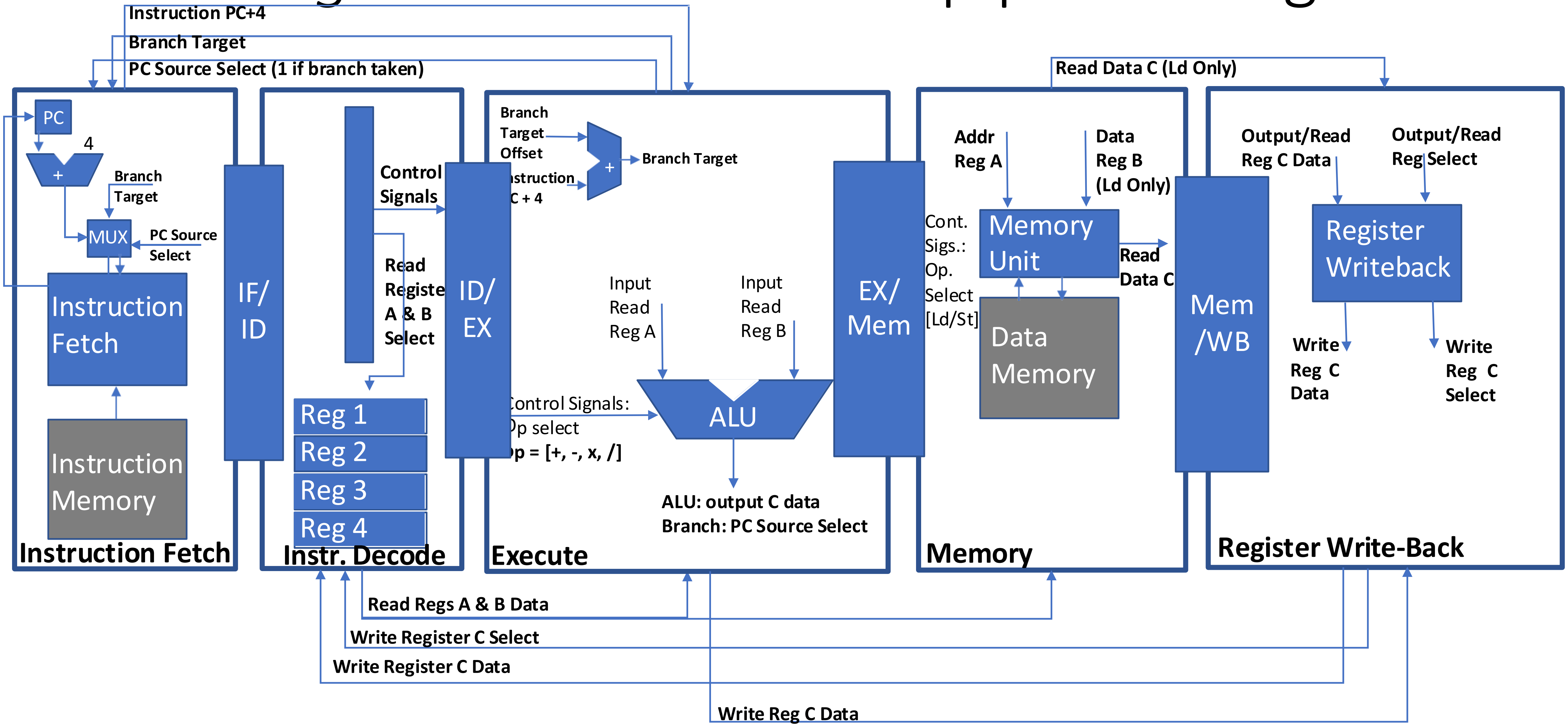
# 4-stage? Pro / con?



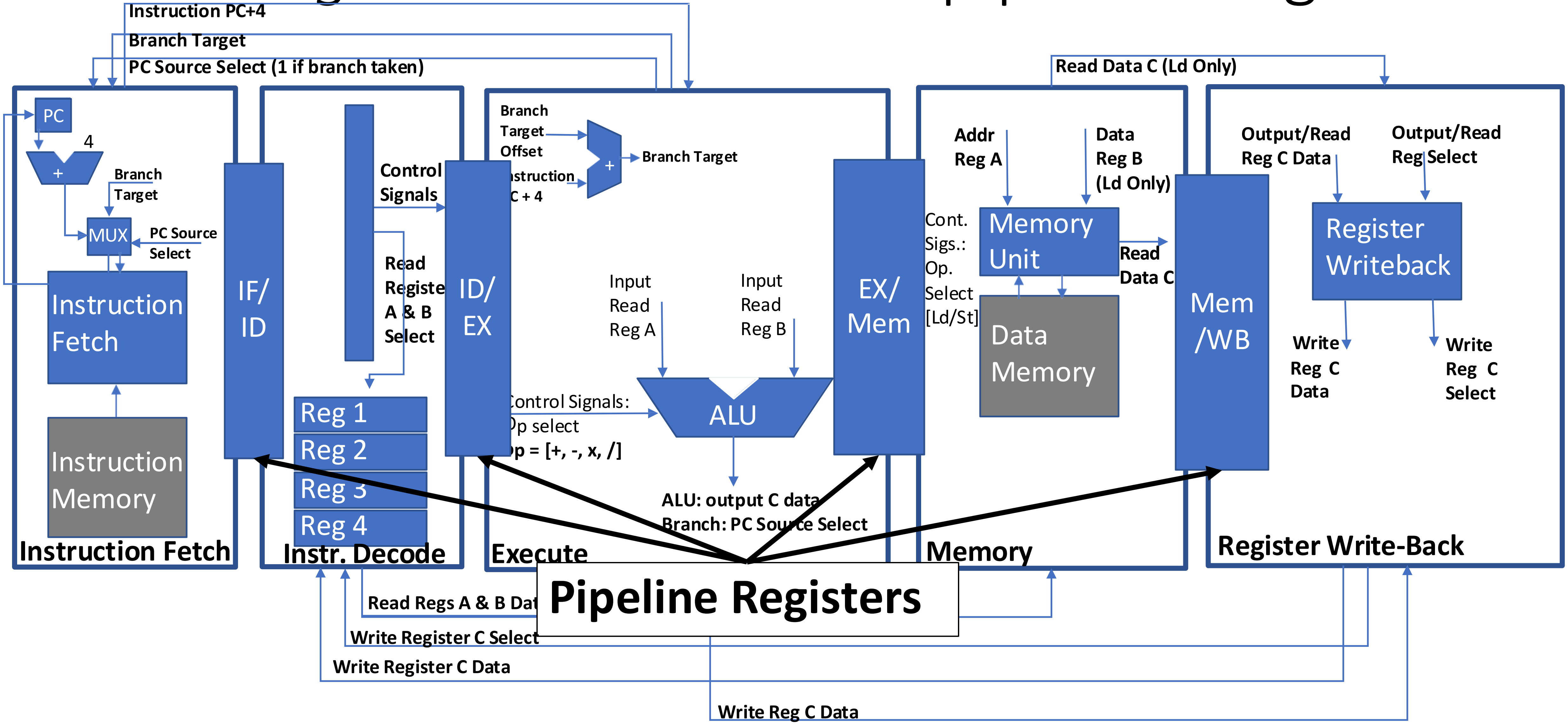
# What does ALU op do in Mem? Memop in EX?



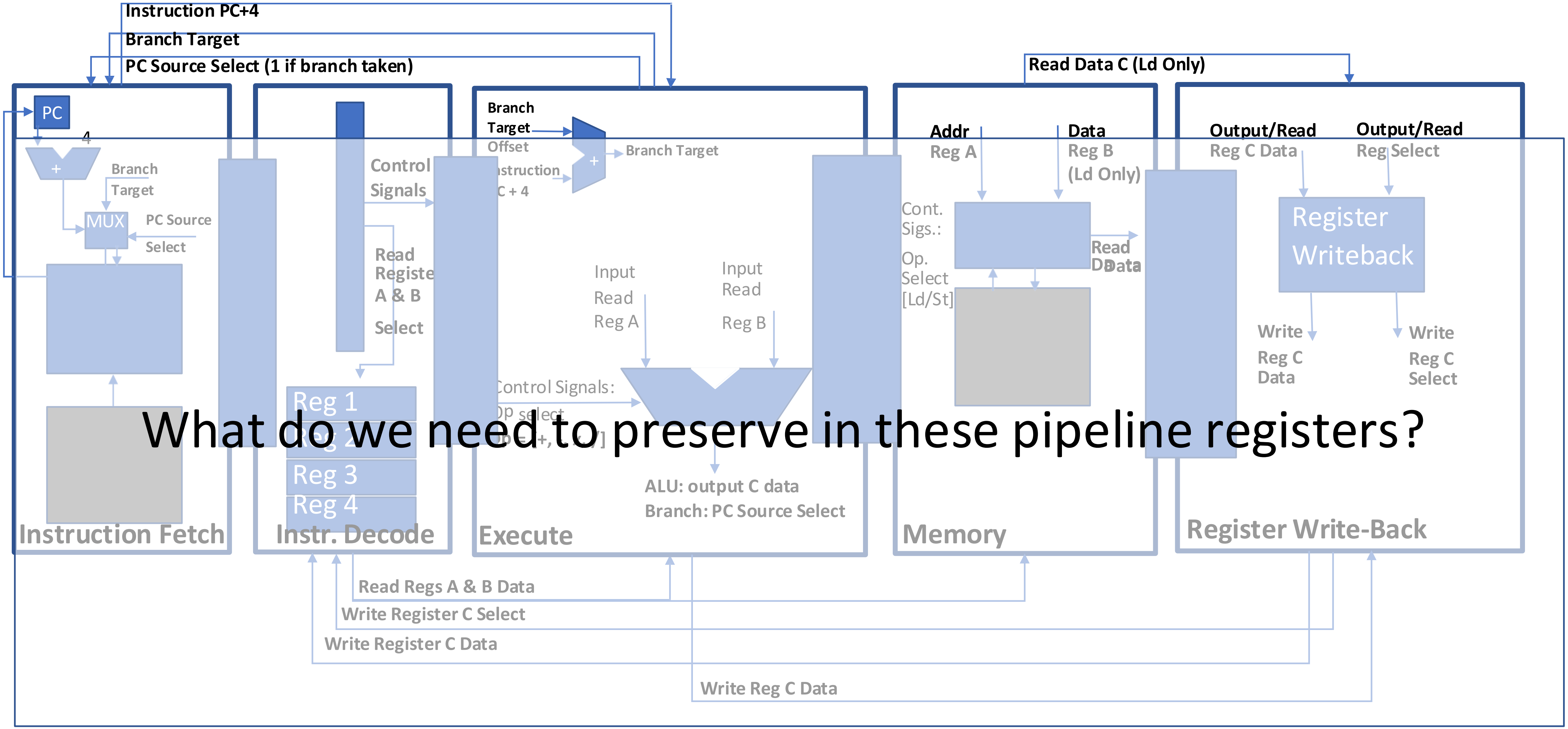
# Cost of pipelining: Need to *register* state between pipeline stages



# Cost of pipelining:
# Need to *register* state between pipeline stages

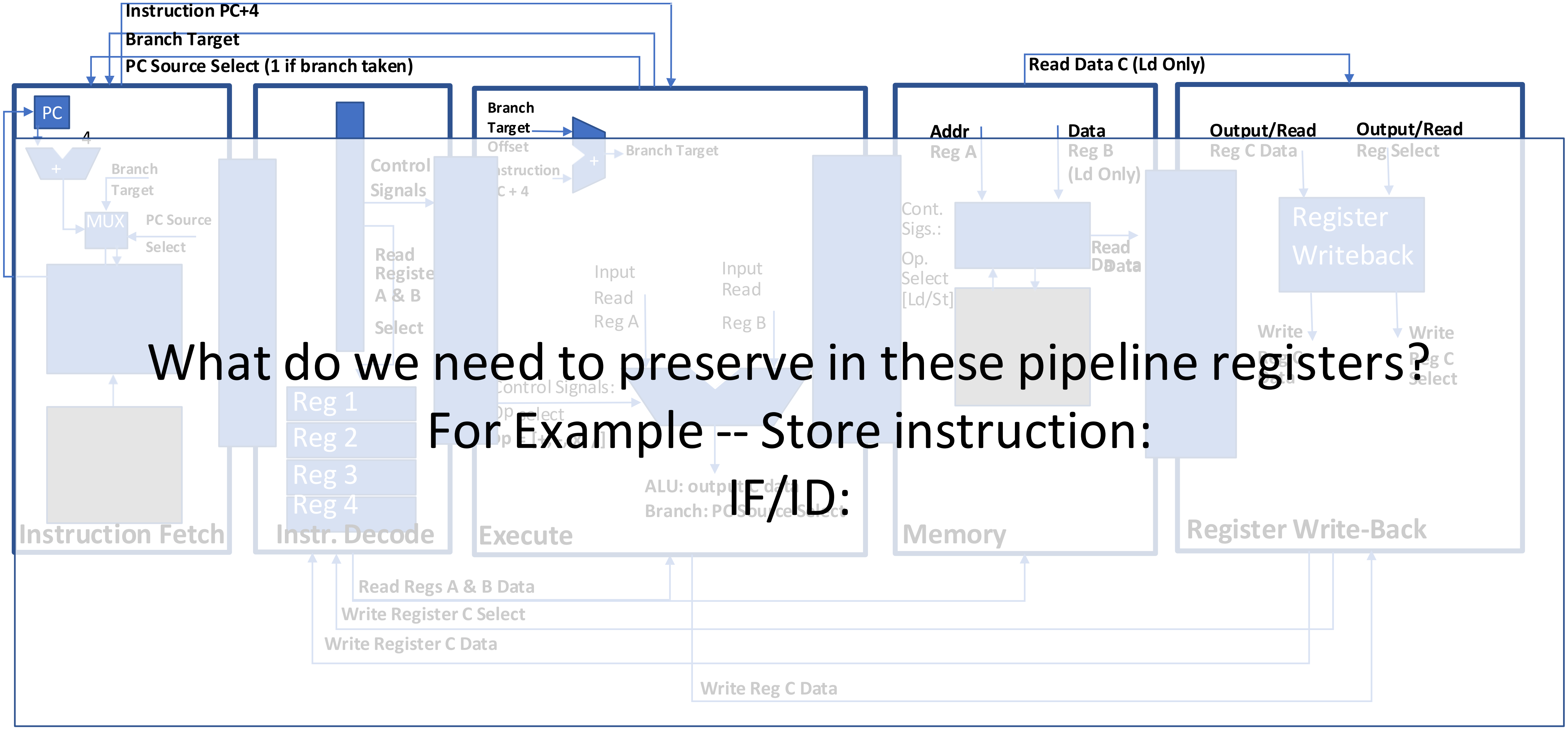


# Need to store state between pipeline stages




What do we need to preserve in these pipeline registers?

# Need to store state between pipeline stages

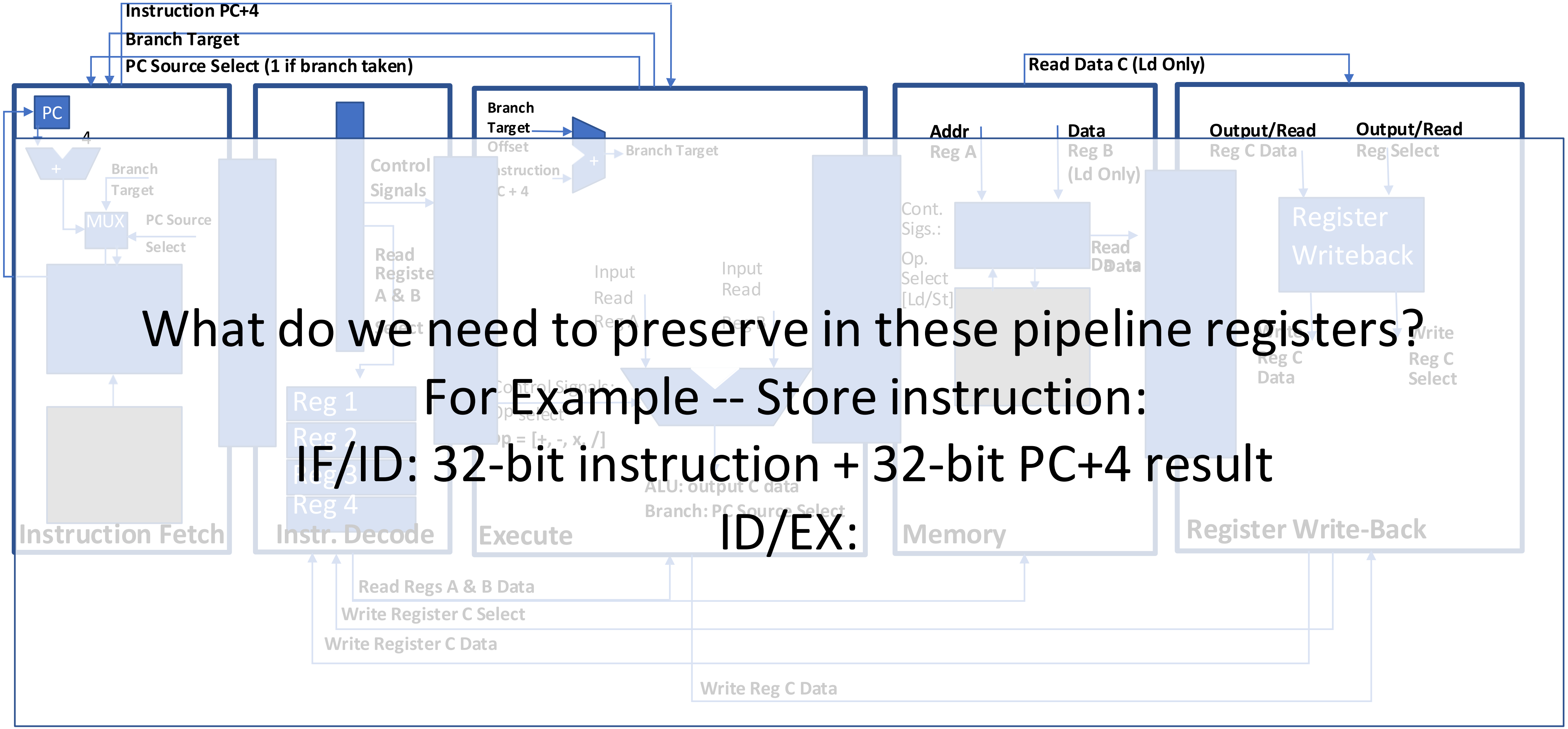



What do we need to preserve in these pipeline registers?

For Example -- Store instruction:

IF/ID:

# Need to store state between pipeline stages



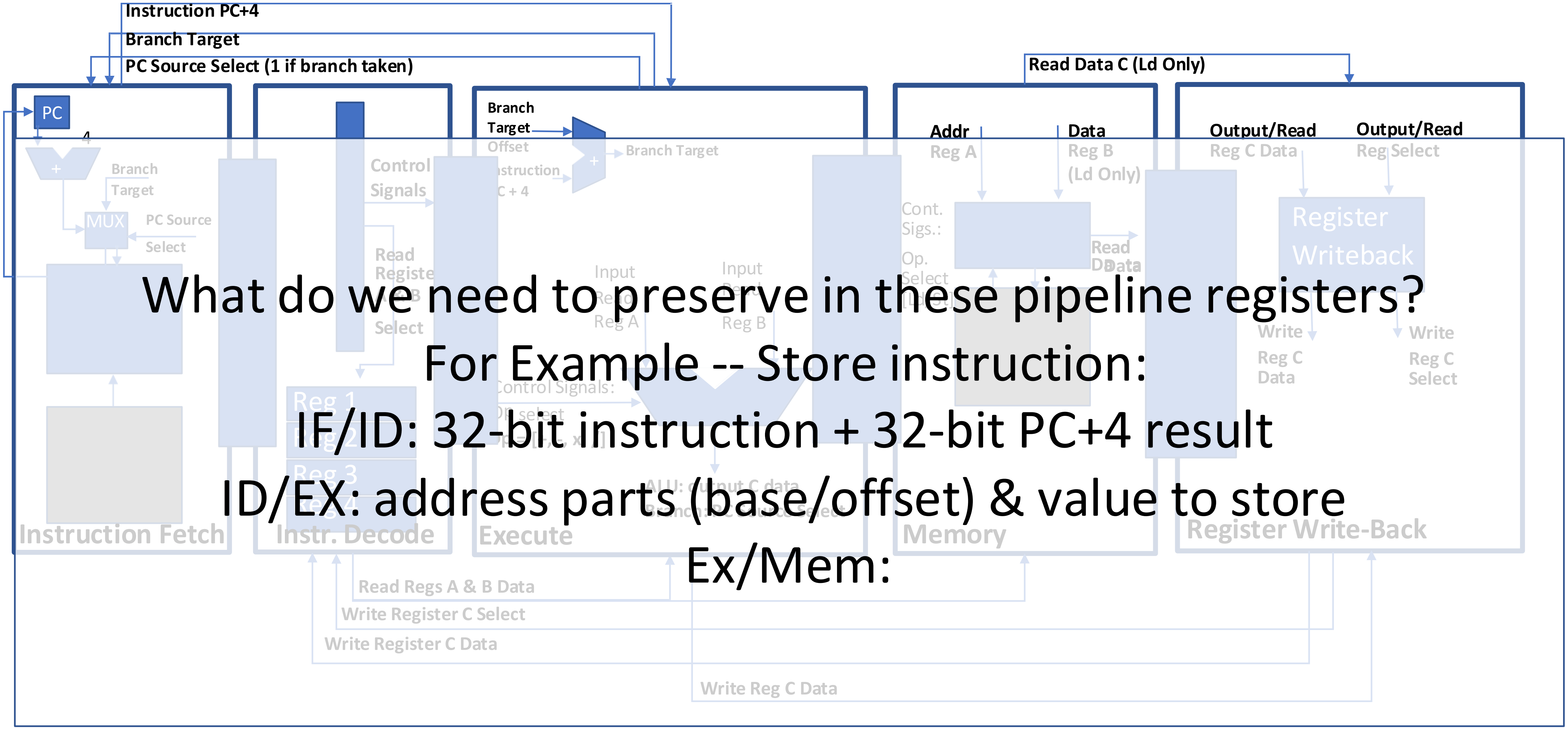

What do we need to preserve in these pipeline registers?

For Example -- Store instruction:

IF/ID: 32-bit instruction + 32-bit PC+4 result

ID/EX:

# Need to store state between pipeline stages




What do we need to preserve in these pipeline registers?

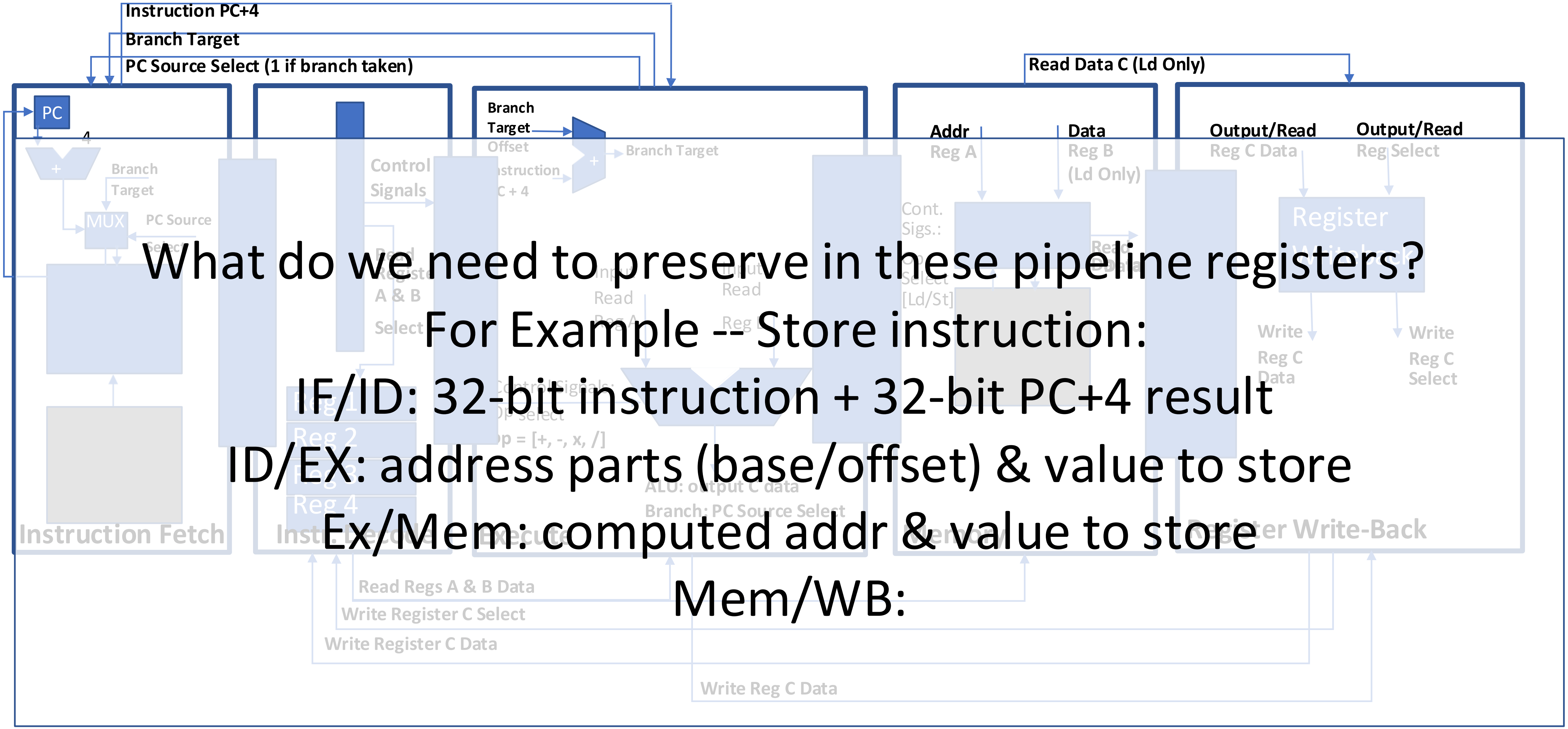
For Example -- Store instruction:

IF/ID: 32-bit instruction + 32-bit PC+4 result

ID/EX: address parts (base/offset) & value to store

Ex/Mem:

# Need to store state between pipeline stages




What do we need to preserve in these pipeline registers?

For Example -- Store instruction:

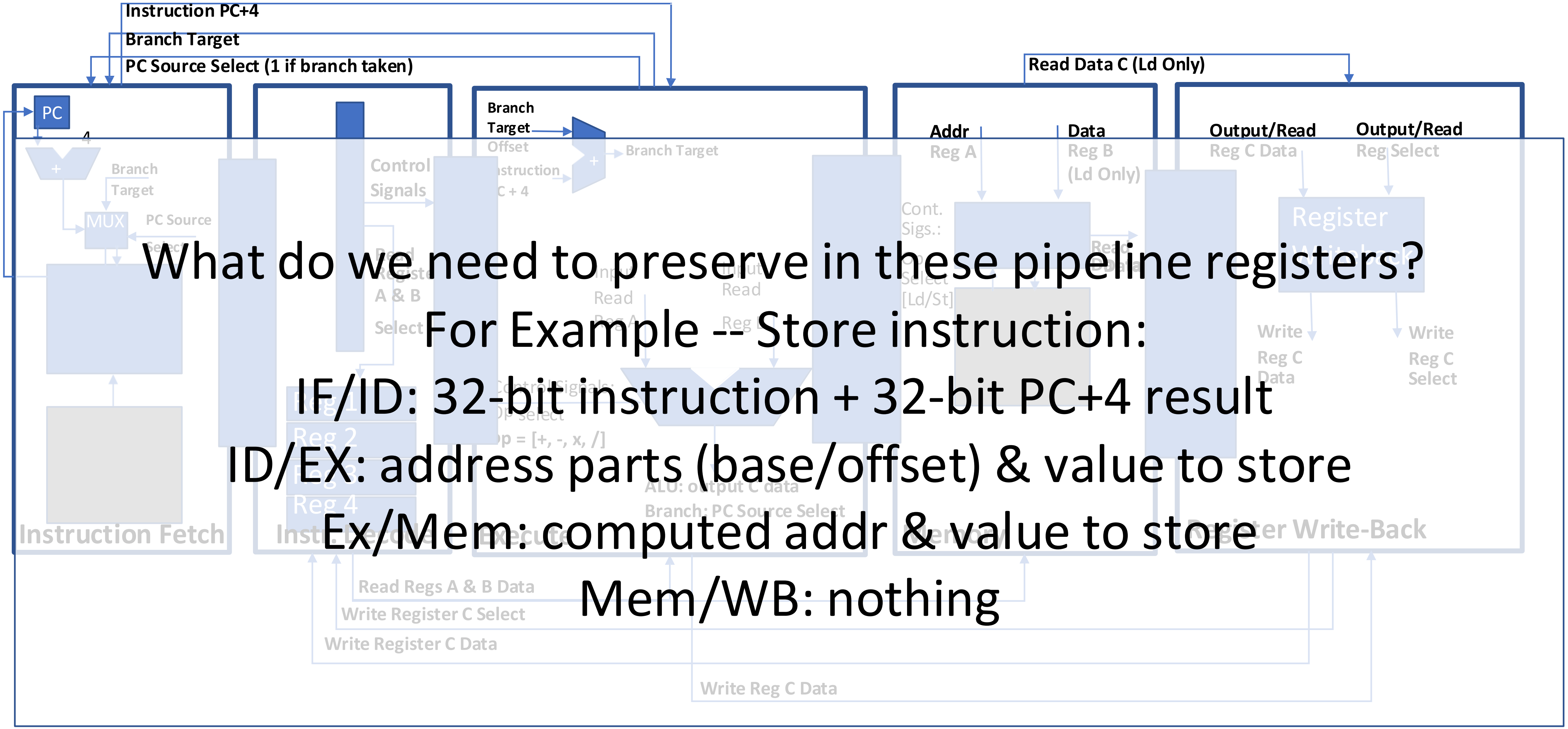
IF/ID: 32-bit instruction + 32-bit PC+4 result

ID/EX: address parts (base/offset) & value to store

Ex/Mem: computed addr & value to store

Mem/WB:

# Need to store state between pipeline stages




What do we need to preserve in these pipeline registers?

For Example -- Store instruction:

IF/ID: 32-bit instruction + 32-bit PC+4 result

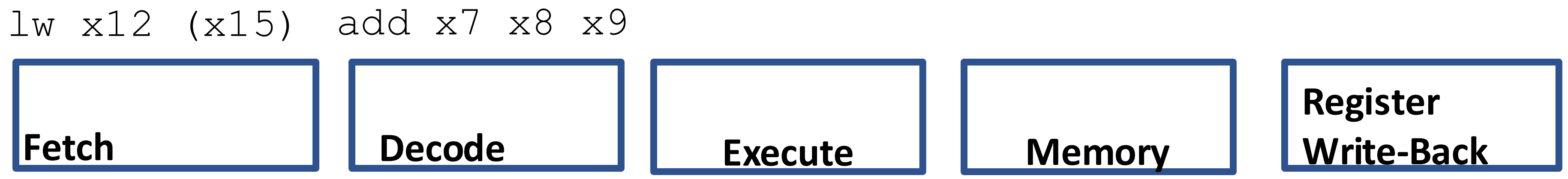
ID/EX: address parts (base/offset) & value to store

Ex/Mem: computed addr & value to store

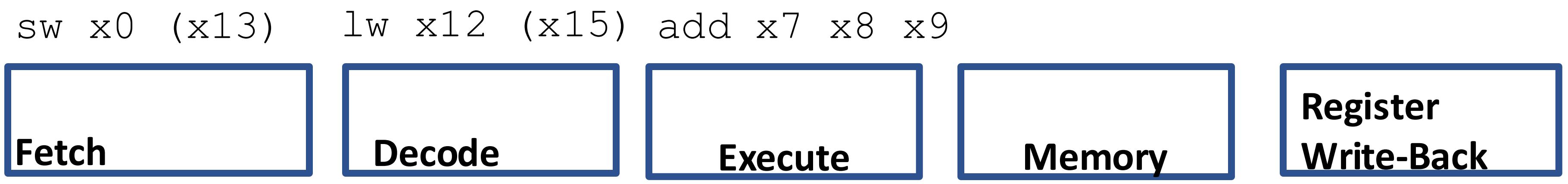
Mem/WB: nothing

# Example: Pipelined Execution

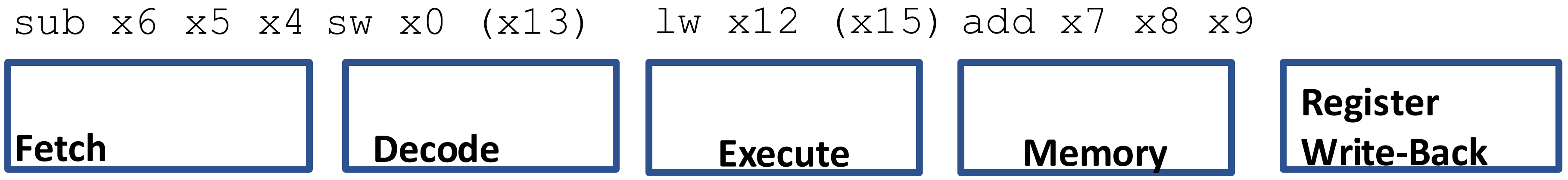
# Example: Pipelined Execution



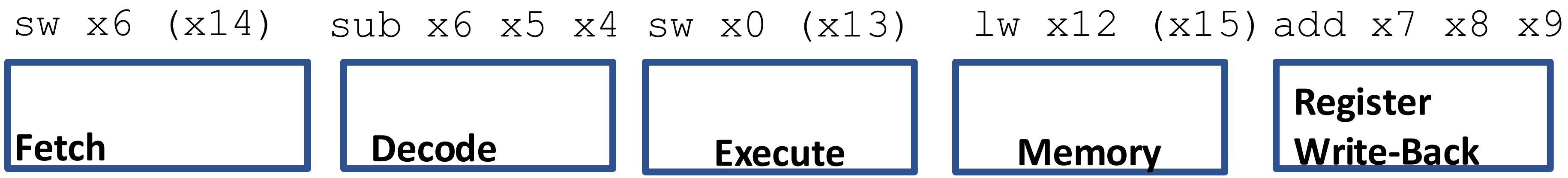
# Example: Pipelined Execution



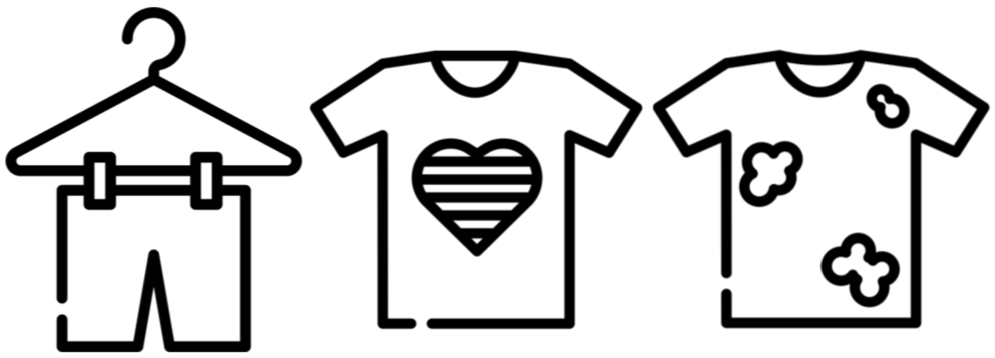
# Example: Pipelined Execution

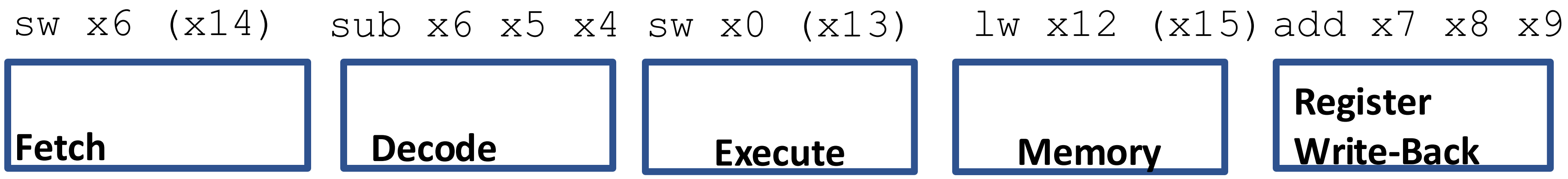


# Example: Pipelined Execution

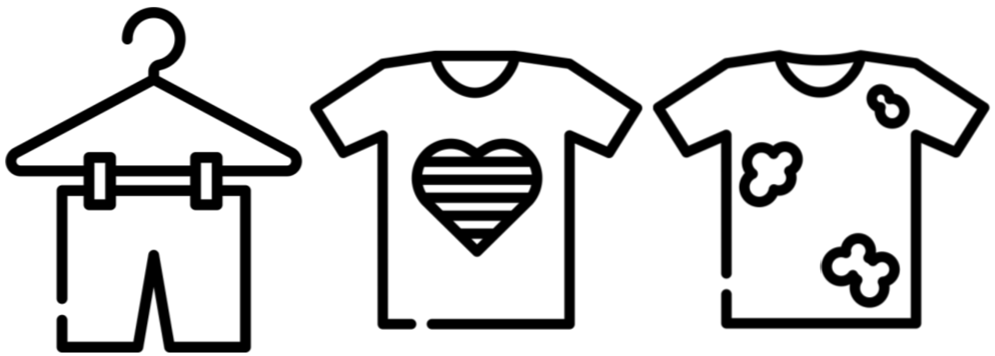


# Example: Pipelined Execution





# Example: Pipelined Execution



| Fetch | Decode | Execute | Memory | Register Write-Back |
|---|---|---|---|---|
| `sw x6 (x14)` | `sub x6 x5 x4` | `sw x0 (x13)` | `lw x12 (x15)` | `add x7 x8 x9` |

Key Idea:Pipelining unlocks

***Instruction Level Parallelism (ILP)***

one of the great ideas in computer architecture

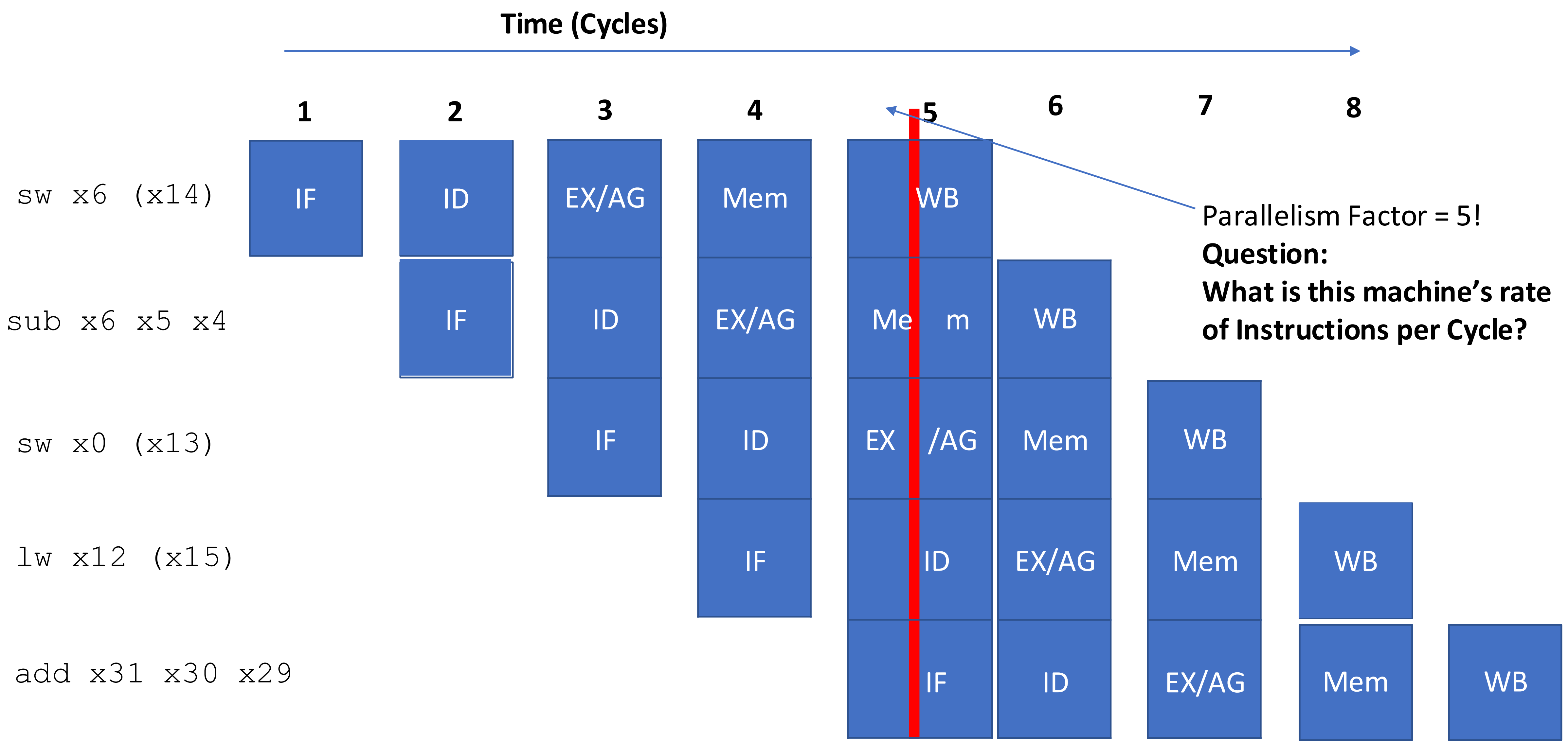
**Practical Implications of adding ILP to the system?**

# Pipeline Diagram Illustrates Parallelism

**Time (Cycles)** →

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| `sw x6 (x14)` | IF | ID | EX/AG | Mem | WB | | | | |
| `sub x6 x5 x4` | | IF | ID | EX/AG | Mem | WB | | | |
| `sw x0 (x13)` | | | IF | ID | EX/AG | Mem | WB | | |
| `lw x12 (x15)` | | | | IF | ID | EX/AG | Mem | WB | |
| `add x31 x30 x29` | | | | | IF | ID | EX/AG | Mem | WB |

# Pipeline Diagram Illustrates Parallelism

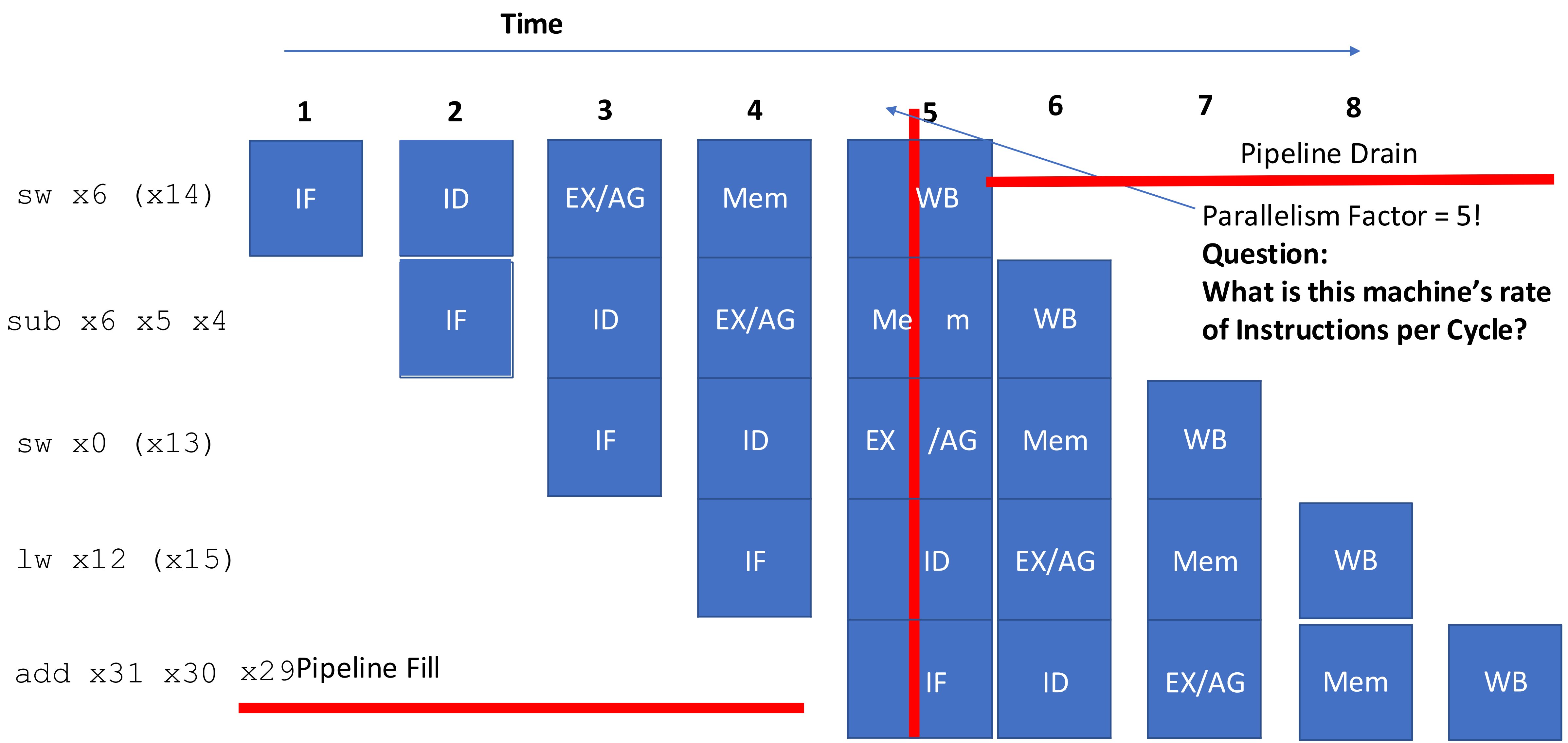



Parallelism Factor = 5!

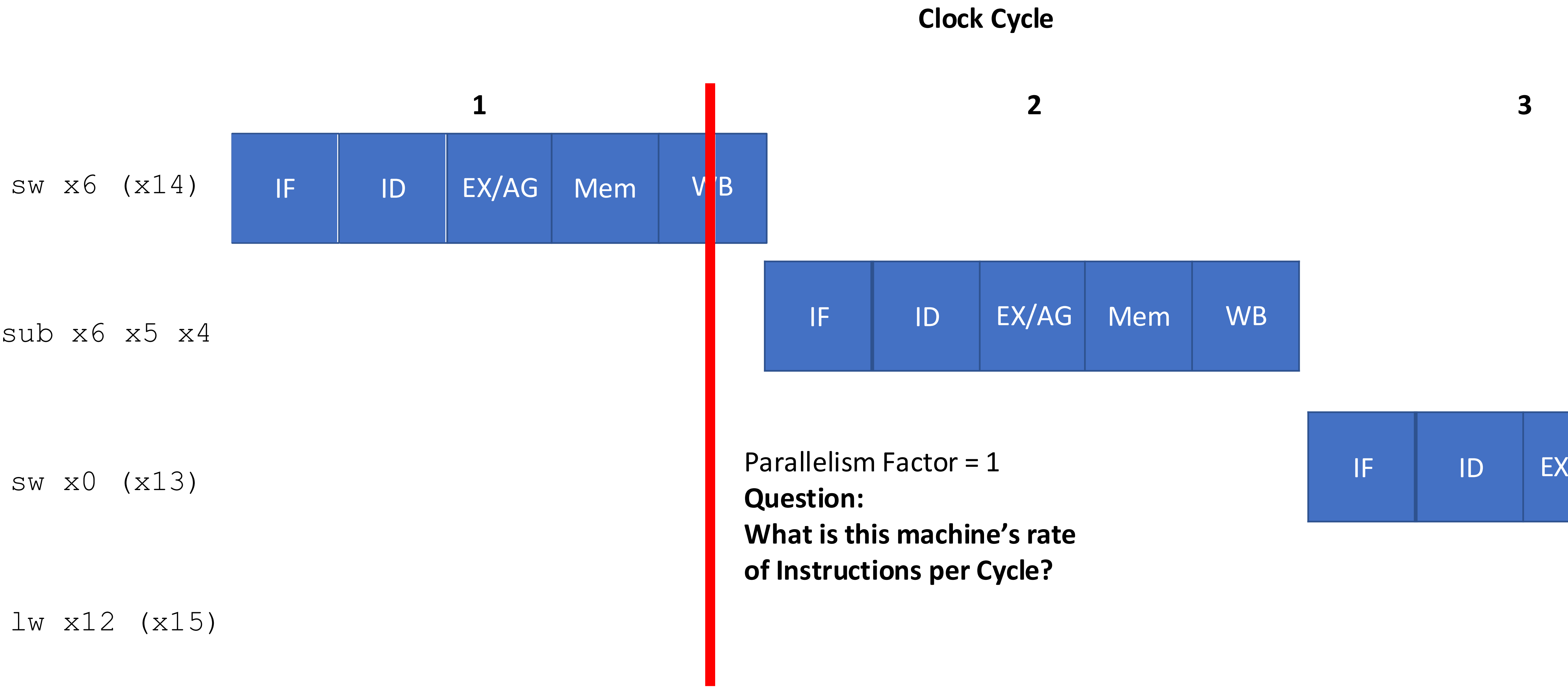
**Question:**

**What is this machine's rate of Instructions per Cycle?**

# Pipeline Diagram Illustrates Parallelism



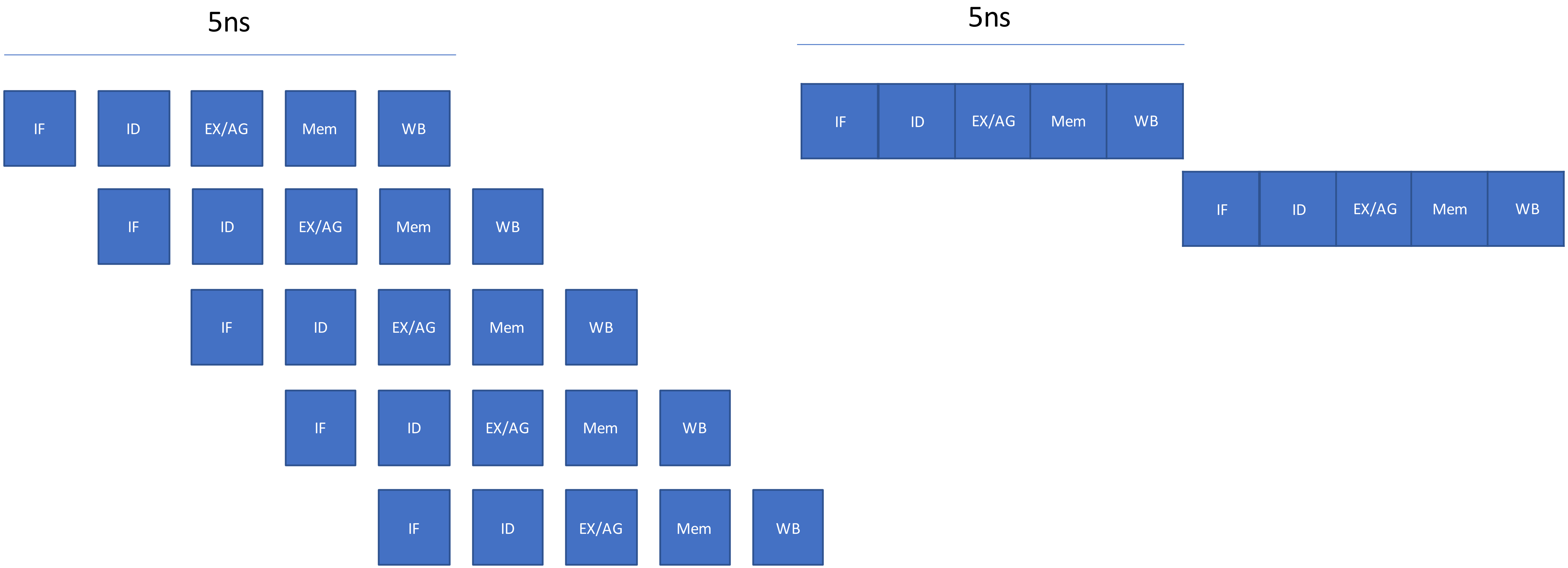
# Pipeline Diagram: Single Cycle Design



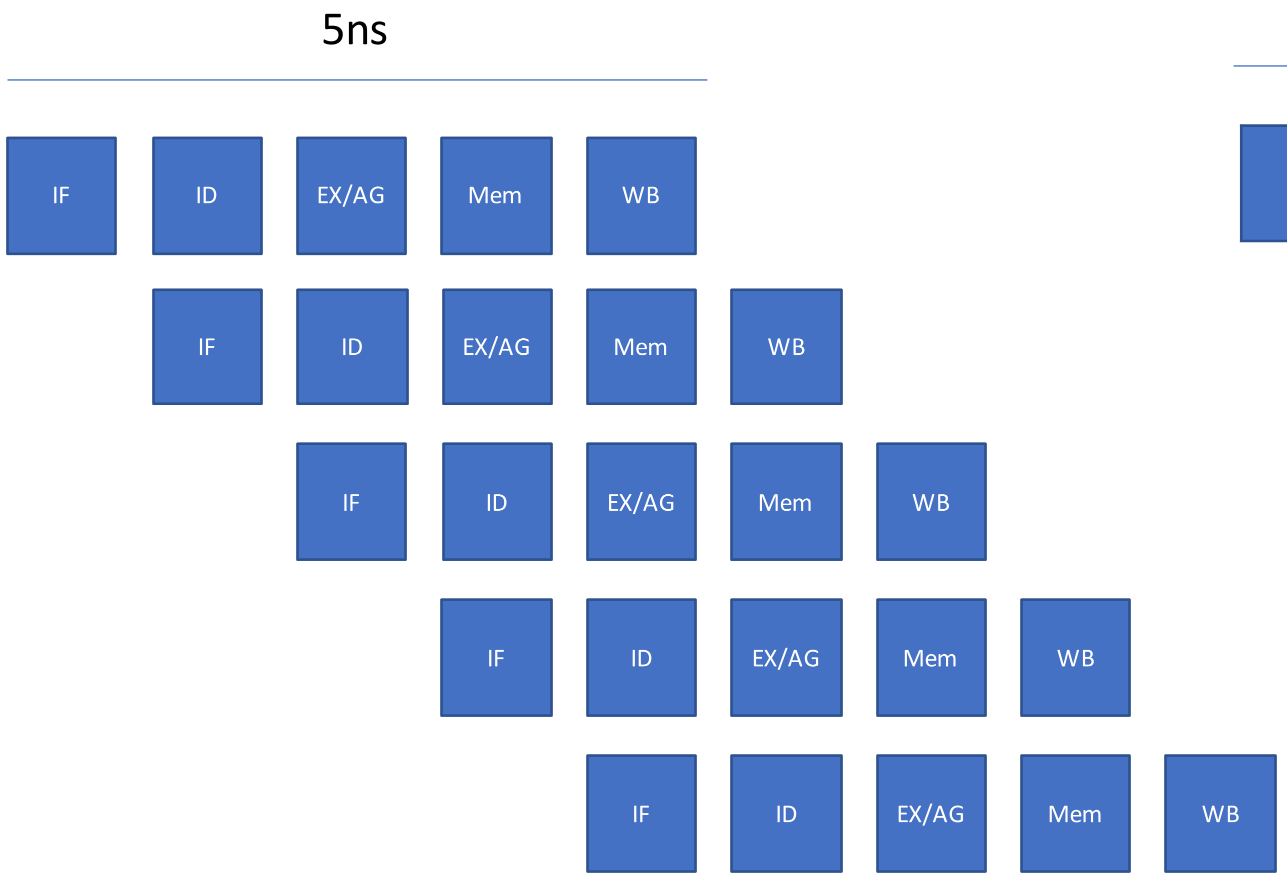

Parallelism Factor = 1

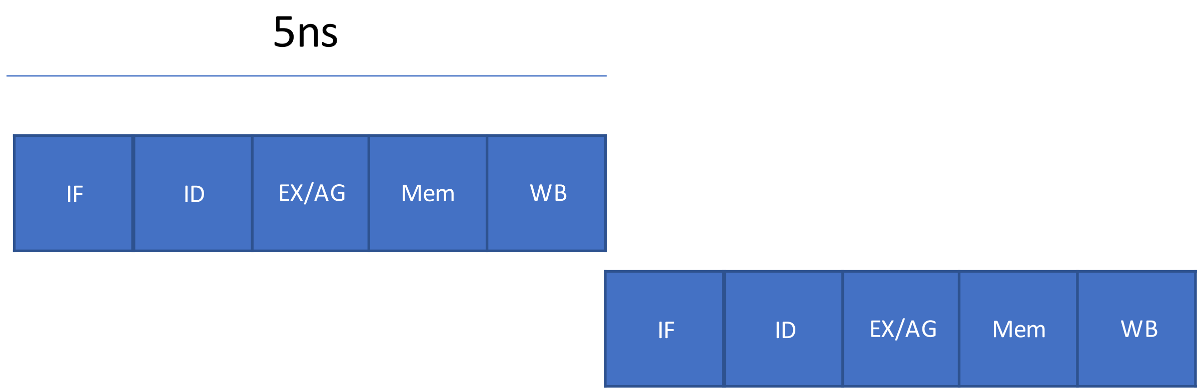
**Question:**

**What is this machine's rate of Instructions per Cycle?**




What gives? IPC is 1 for both and each instruction's ***latency*** is still 5ns.




**Instructions/Cycle**
**Vs**
**Instructions/Second**

**5ns = (up to) 25(*) stages of work**
**vs**
**5ns = 5 stages of work**

*** 15 here due to pipeline filling**

What gives?  IPC is 1 in both cases!

**Key Idea:** Pipelined ***Instruction Throughput*** is higher.

**Shorter clock period + parallelism = 1 completed instruction per ns even though *each* instruction takes 5ns to complete**

# Iron Law of Computer Performance

| instructions / program | x | cycles / instruction | x | seconds / cycle |
| --- | --- | --- | --- | --- |

# Iron Law of Computer Performance

| | | | | |
|---|---|---|---|---|
| instructions / program | X | cycles / instruction | X | seconds / cycle |

**Question: what term does pipelining optimize? how else might we approach optimization in light of this performance expression?**

# Pipelining Code Example

```
p = 0xabc;
x = y - z
m = *p;
t = x + w;
```

```
sub x6 x5 x4
lw  x16 0xabc
add x12 x6 x14
```

**What is interesting about this short program?**

# Pipelining Code Example

```
sub x6 x5 x4
lw  x16 0xabc
add x12 x6 x14
```

**What happens to `x6` as we execute this code?**

# Example: Pipelined Execution

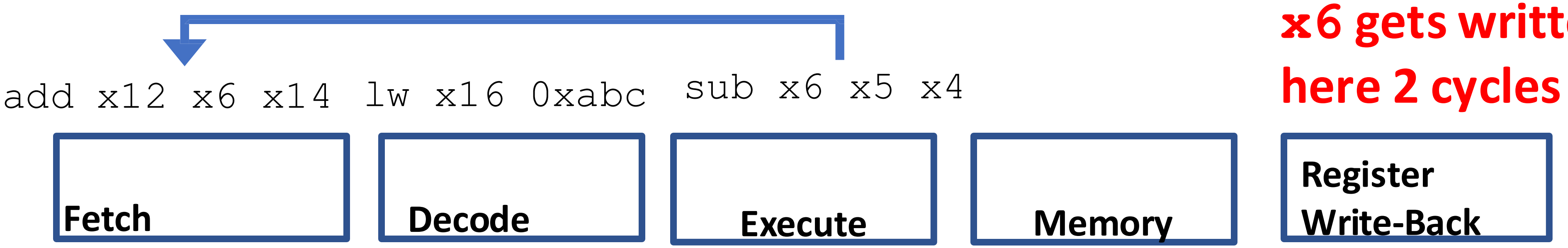
# Example: Pipelined Execution

`lw x16 0xabc sub x6 x5 x4`

# Example: Pipelined Execution

**`add x12 x6 x14`**
**reads the value of `x6` from**
**`sub x6 x5 x4`**

**`x6` gets written back here 2 cycles later!**




**Read-After-Write (RAW) Hazard:**

**Input register does not contain updated data during register read cycle due to yet-to-be-completed register writeback from older instruction**

# Types of Data Hazards

```
sub x6 x5 x4
lw  x16 0xabc
add x12 x6 x14
```

**Read-After-Write (RAW)**

```
sub x8 x16 x4
add x16 x6 x14
lw  x16 0xabc
```

**Write-After-Read (WAR)**

```
lw  x6 0xabc
sub x6 x5 x4
add x12 x6 x14
```

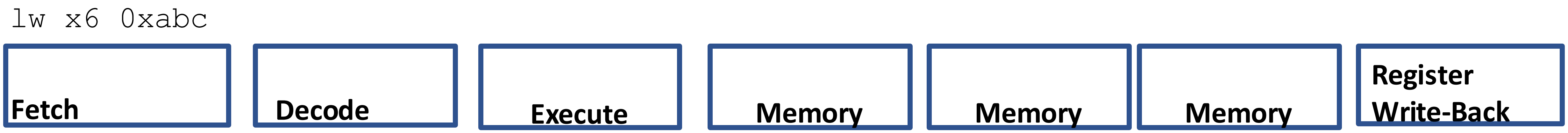
**Write-After-Write (WAW)**

***Only Read-After-Write (RAW) hazards are possible in our simple pipeline***

# Types of Data Hazards

```
lw  x6 0xabc
sub x6 x5 x4
add x12 x6 x14
```

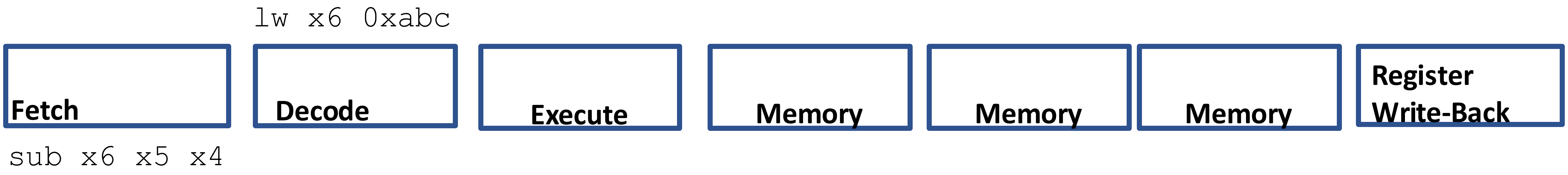
**Write-After-Write (WAW)**



# Types of Data Hazards

```
lw  x6 0xabc
sub x6 x5 x4
add x12 x6 x14
```

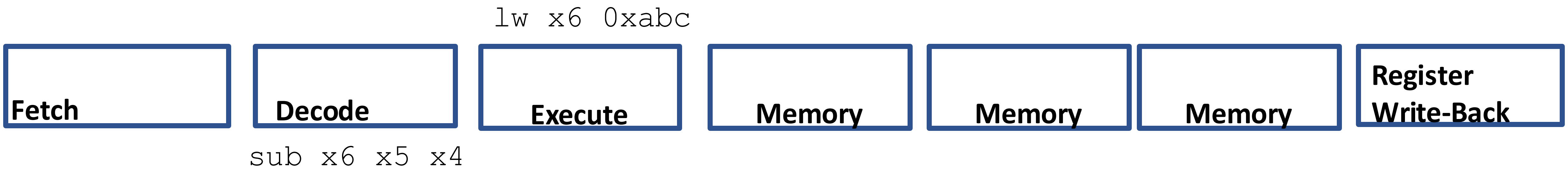
**Write-After-Write (WAW)**



# Types of Data Hazards

```
lw  x6 0xabc
sub x6 x5 x4
add x12 x6 x14
```

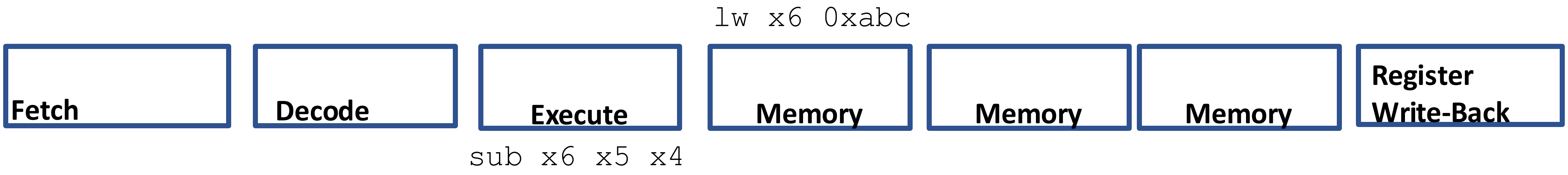
**Write-After-Write (WAW)**



# Types of Data Hazards

```
lw  x6 0xabc
sub x6 x5 x4
add x12 x6 x14
```

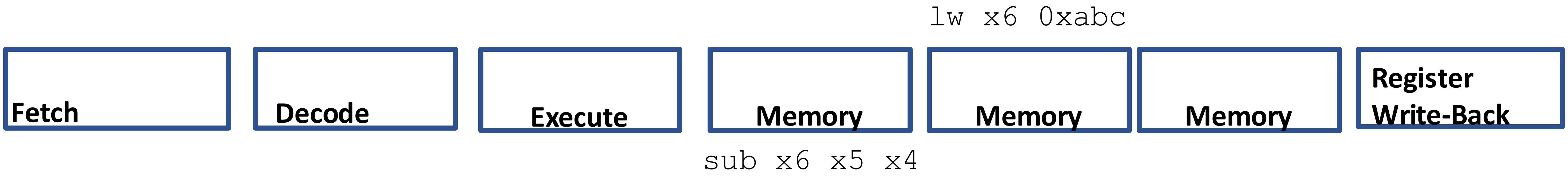
**Write-After-Write (WAW)**



# Types of Data Hazards

```
lw  x6 0xabc
sub x6 x5 x4
add x12 x6 x14
```

**Write-After-Write (WAW)**

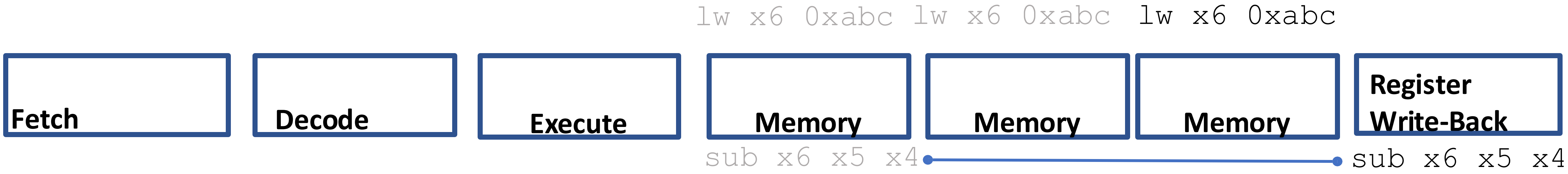


# Types of Data Hazards

```
lw  x6  0xabc
sub x6  x5  x4
add x12 x6  x14
```

**Write-After-Write (WAW)**

**Multi-cycle latency memory op**




**Non-mem-op, single memory cycle**

Earlier `lw` instruction finishes after later `sub` instruction. Both write `x6`. Wrong final value in `x6`.

**Explicitly handled with logic to maintain ordering in processors that allow this behavior (not our datapath)**

# Types of Data Hazards

```
sub x8 x16 x4
add x16 x6 x14
lw  x11 0xabc
```

**Write-After-Read (WAR)**

**Stalled at decode/reg. read**
**(why? wait a few lectures & more in 447)**

`sub x8 x16 x4`

- Fetch
- Decode
- Execute
- Memory
- Register Write-Back

`add x16 x6 x14`

**Completes quickly and writes reg.**

**Later `add` instruction writes `x16` before earlier `sub` instruction reads `x16`. `sub` sees wrong value!**

# What can we do about these data hazards?

```
sub  x6  x5  x4
lw   x16 0xabc
add  x12 x6  x14
```

**Read-After-Write (RAW)**

```
sub  x8  x16 x4
add  x16 x6  x14
lw   x16 0xabc
```

**Write-After-Read (WAR)**

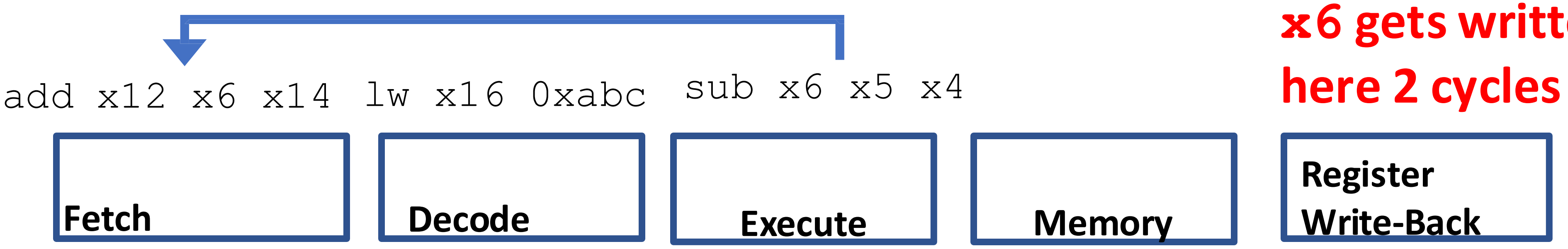
```
lw   x6  0xabc
sub  x6  x5  x4
add  x12 x6  x14
```

**Write-After-Write (WAW)**

***Only Read-After-Write (RAW) hazards are possible in our simple pipeline***

# Example: Pipelined Execution w/ RAW Hazard

**`add x12 x6 x14`**
**reads the value of `x6` from**
**`sub x6 x5 x4`**

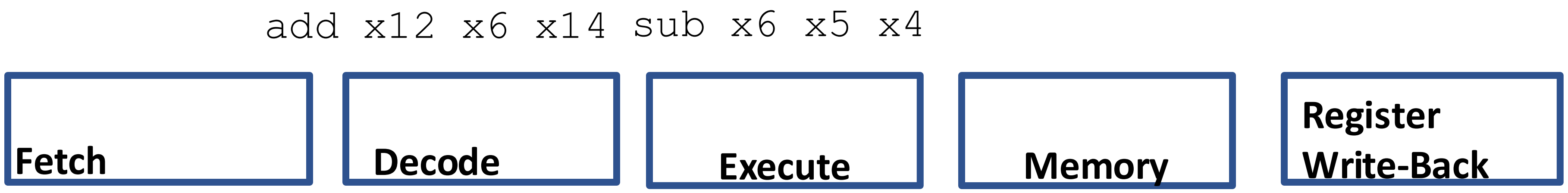



**`x6` gets written back here 2 cycles later!**

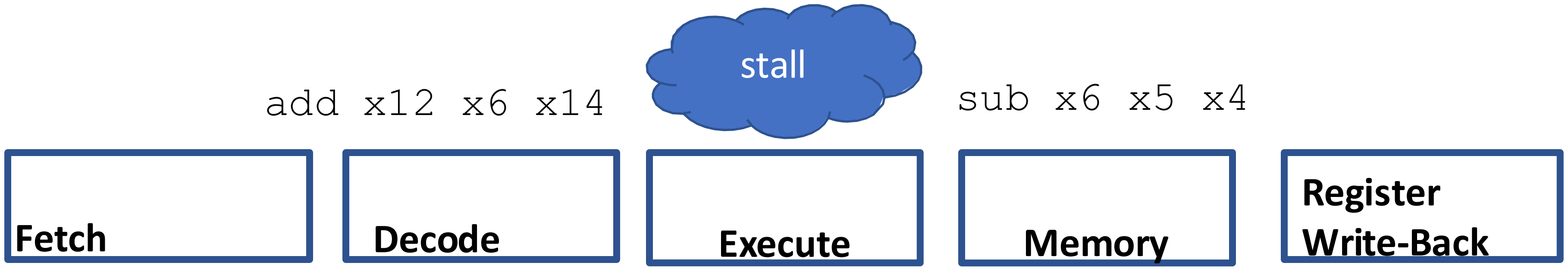
**Read-After-Write (RAW) Hazard:**
**Input register does not contain updated data during register read cycle due to yet-to-be-completed register writeback from older instruction**

# Example: Pipelined Execution w/ RAW Hazard

# Example: Pipelined Execution w/ RAW Hazard

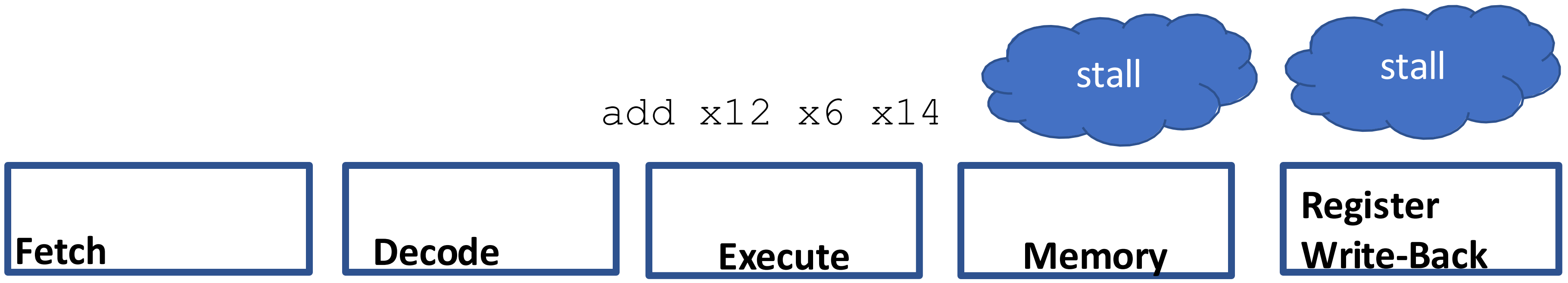


# Example: Pipelined Execution w/ RAW Hazard

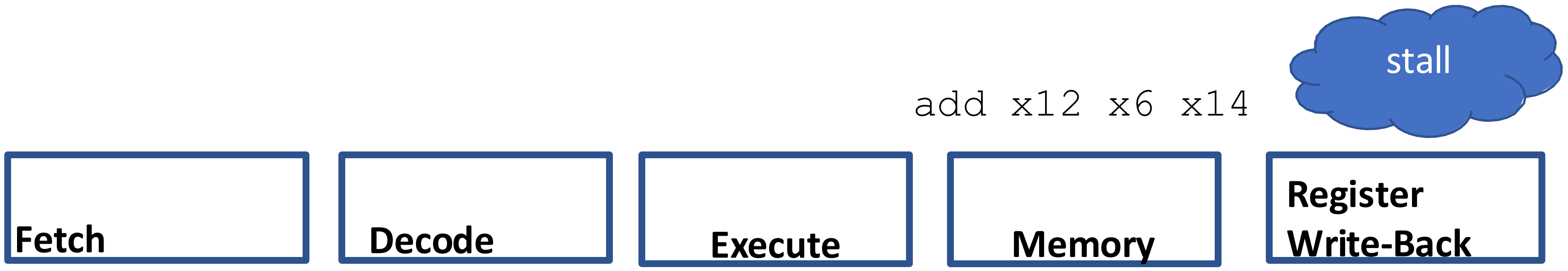


# Example: Pipelined Execution w/ RAW Hazard

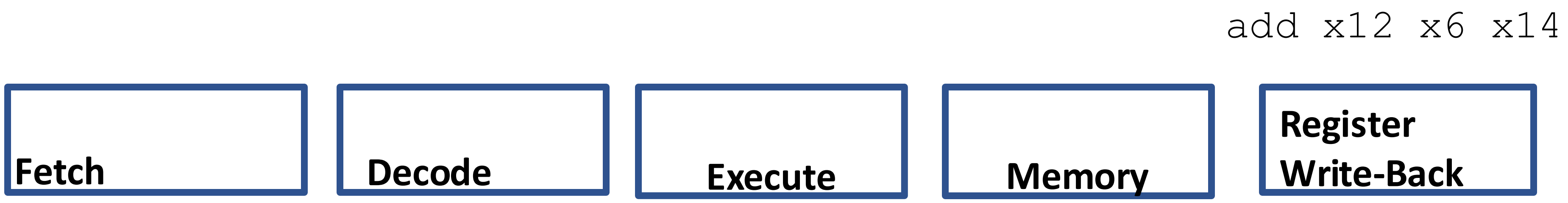
# Example: Pipelined Execution w/ RAW Hazard



# Example: Pipelined Execution w/ RAW Hazard

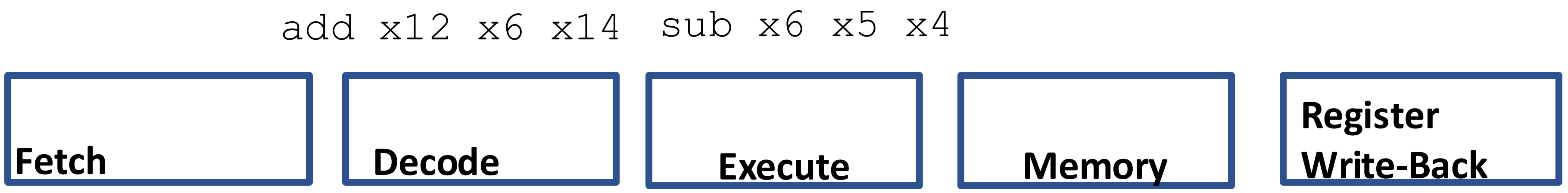


# Example: Pipelined Execution w/ RAW Hazard



# How do we avoid the stall cycles?

# Example: Pipelined Execution w/ RAW Hazard

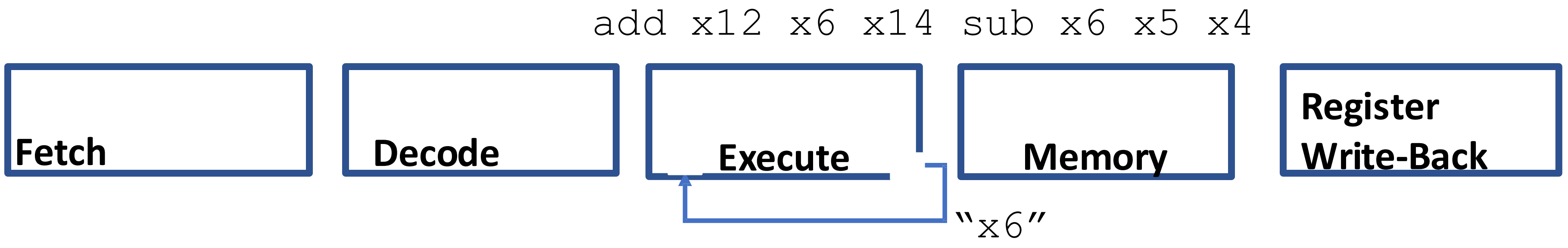



**Value of x6 is available after `sub` Executes**

**We can *forward* the value to the add!**

# Forwarding to avoid a pipeline RAW Hazard

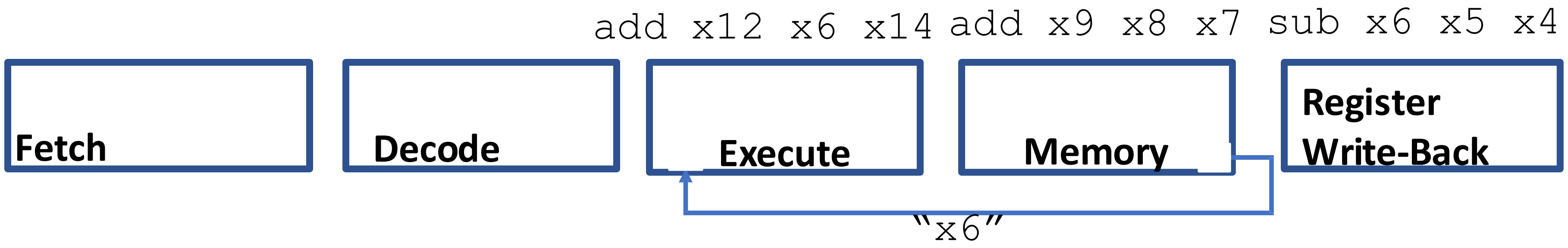
**Value of x6 is available from Execute!**




**We can *forward* the value in the EX/MEM pipeline register from the sub back to Execute to act as the input operand for the add**

# Forwarding to avoid a pipeline RAW Hazard

**Can also forward if there are intervening instructions**

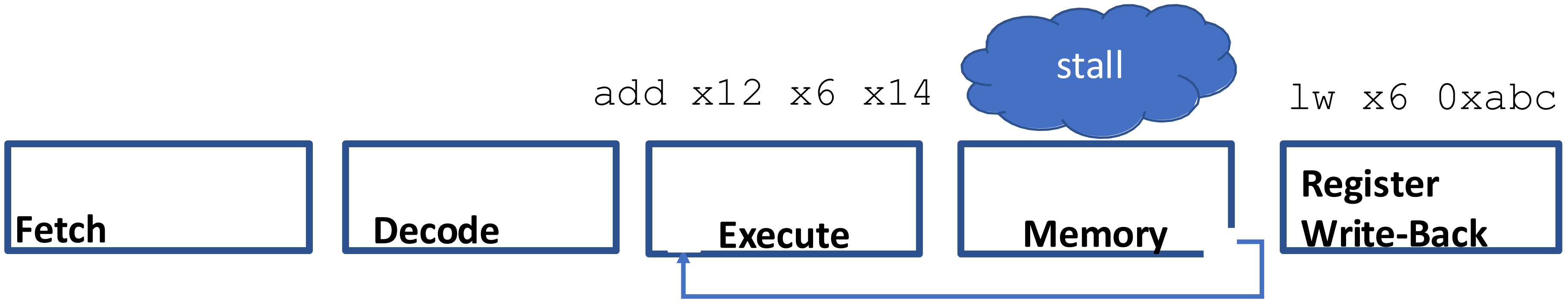



**We can *forward* the value in the MEM/WB pipeline register from the sub back to Execute to act as the input operand for the add (going around the unrelated operation in the memory stage)**

# Pipeline Can Forward Between Different Stages

```
lw x6 0xabc
add x12 x6 x14
```

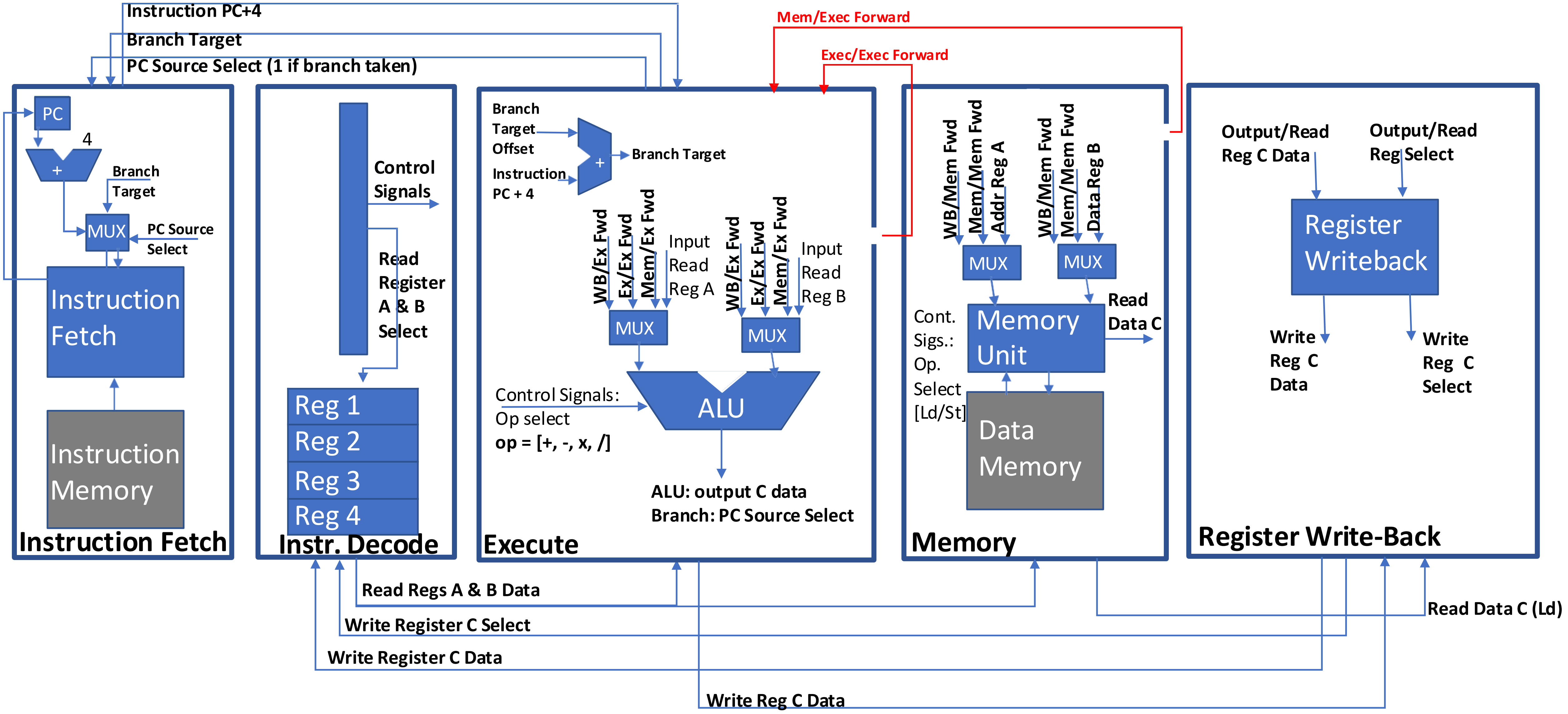
**Value of x6 is available from Memory!**



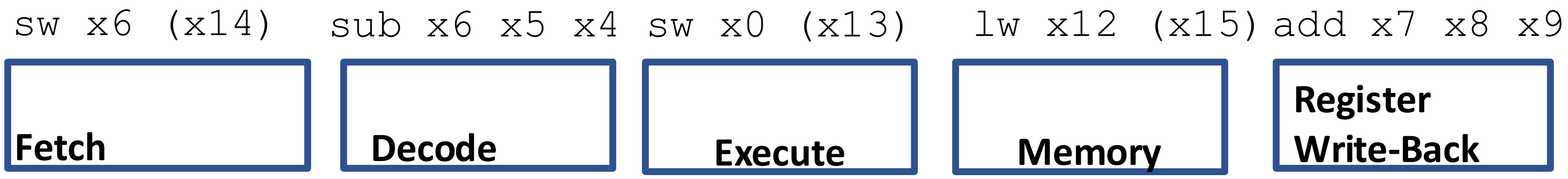

**We can *forward* the value in Memory's pipeline register from the lw back to Execute's input for the add**

**(Still requires stalling…)**

# Adding Forwarding Support

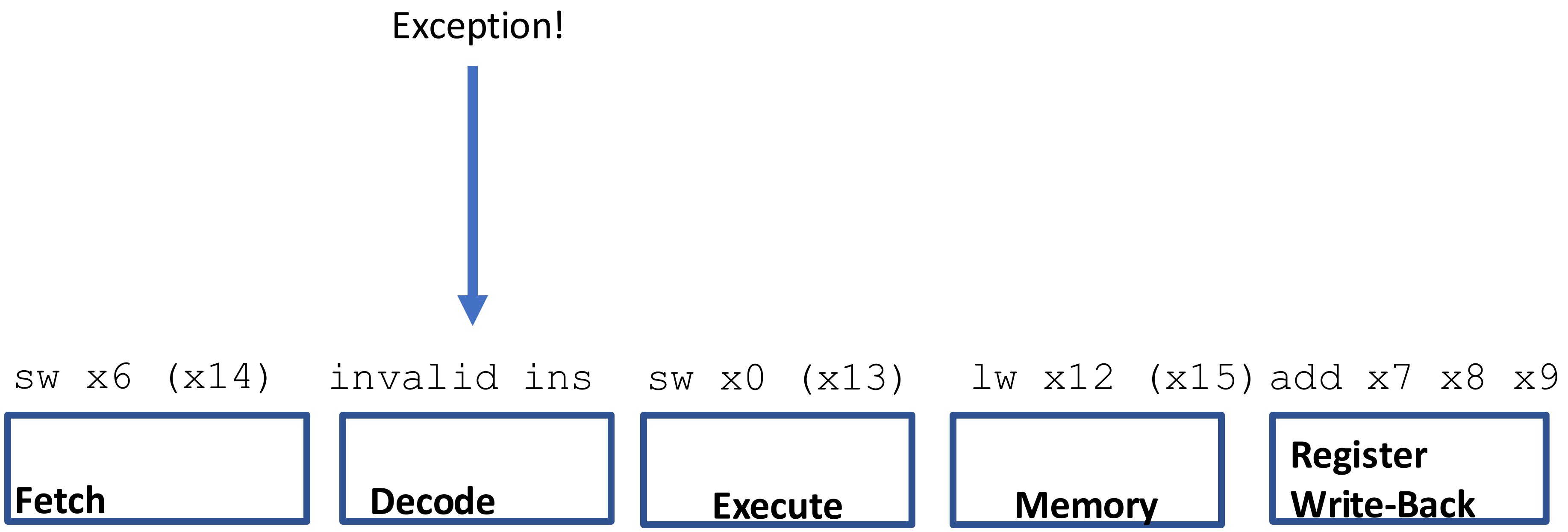


# Question: What is time in a pipelined system?



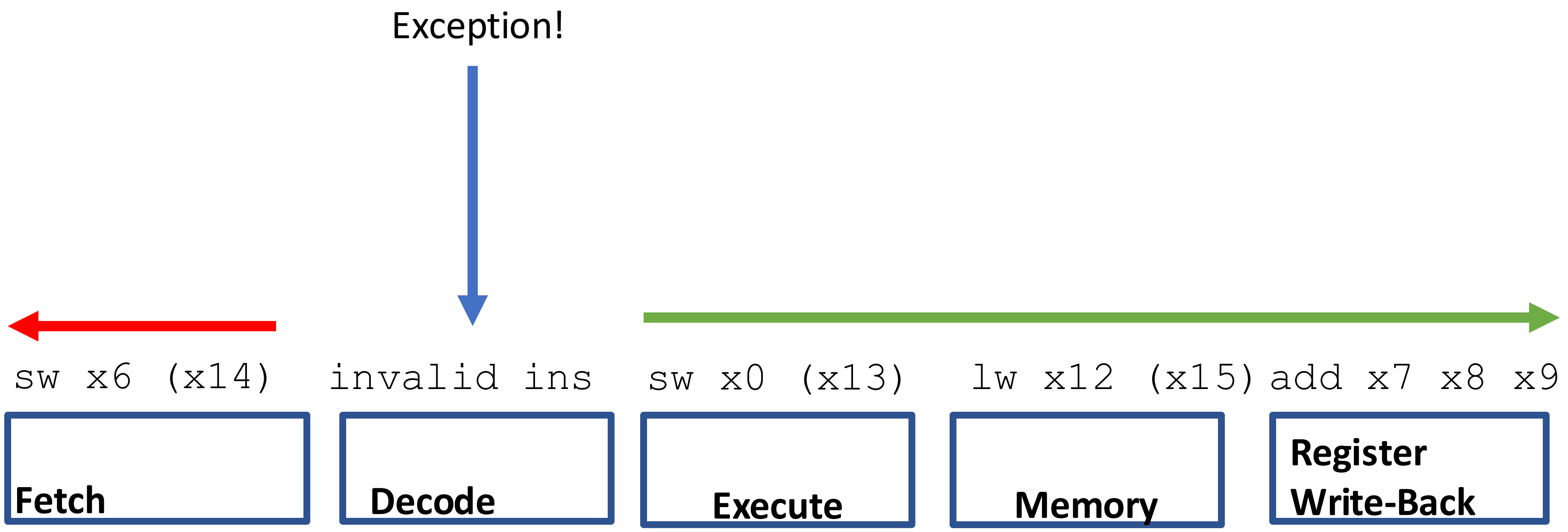

What if one of our instructions were to throw an exception (e.g., illegal instruction in decode or page fault on a memop)?

# Exception Handling



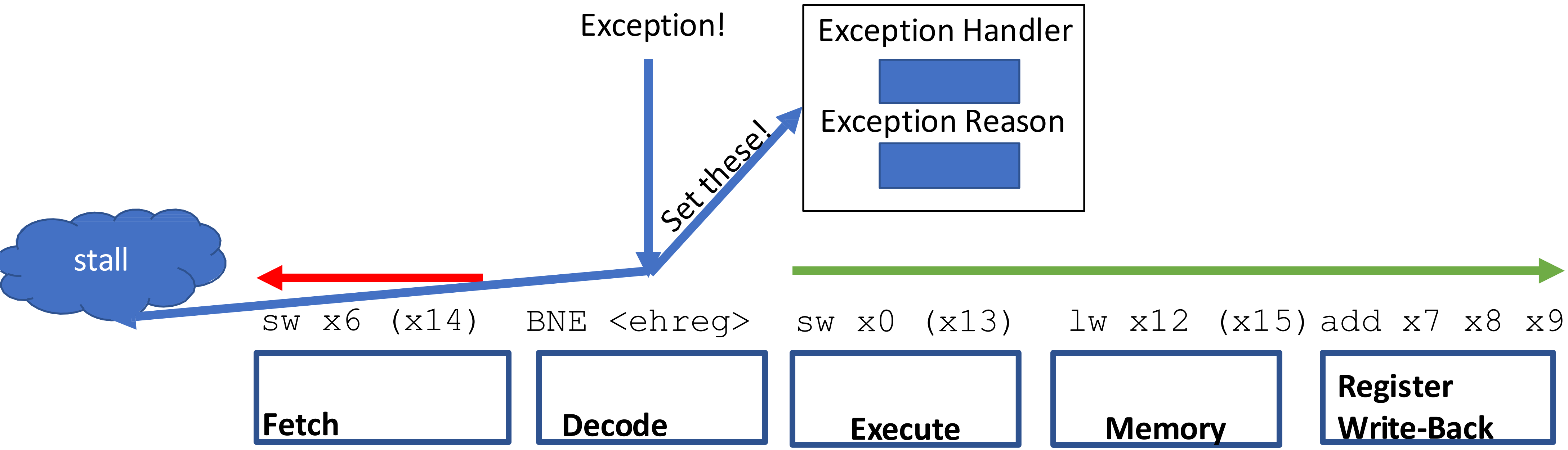

What if one of our instructions were to throw an exception (e.g., illegal instruction in decode or page fault on a memop)?

# Exception Handling




Basic Exception Idea: Nuke everything that started after the current instruction, finish everything that started before the current instruction, jump to exception handler

# Exception Handling




Basic Exception Idea: Nuke everything that started after the current instruction, finish everything that started before the current instruction, jump to exception handler, no new insns

# What did we just learn?

- Basics of pipelining as a first technique for Instruction-level parallelism
- Datapath decomposition to support pipelined execution
- Hazards and their impediment to pipelined execution
- Forwarding in the pipeline to avoid stalling on data hazards

# What to think about next?

- More microarchitectural concepts (next time)
  - Control hazards & branch prediction
- Caches as a microarchitectural optimization (next time)
  - Implementation of cache hierarchies
  - Cache design tradeoffs
- Performance Evaluation (next next time)
  - Design spaces, Pareto Frontiers, and design space exploration