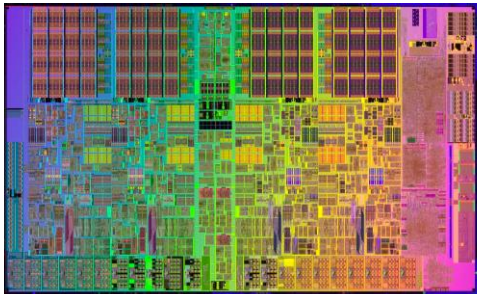
Computer Architecture

CH4 Processor Microarchitecture (III)

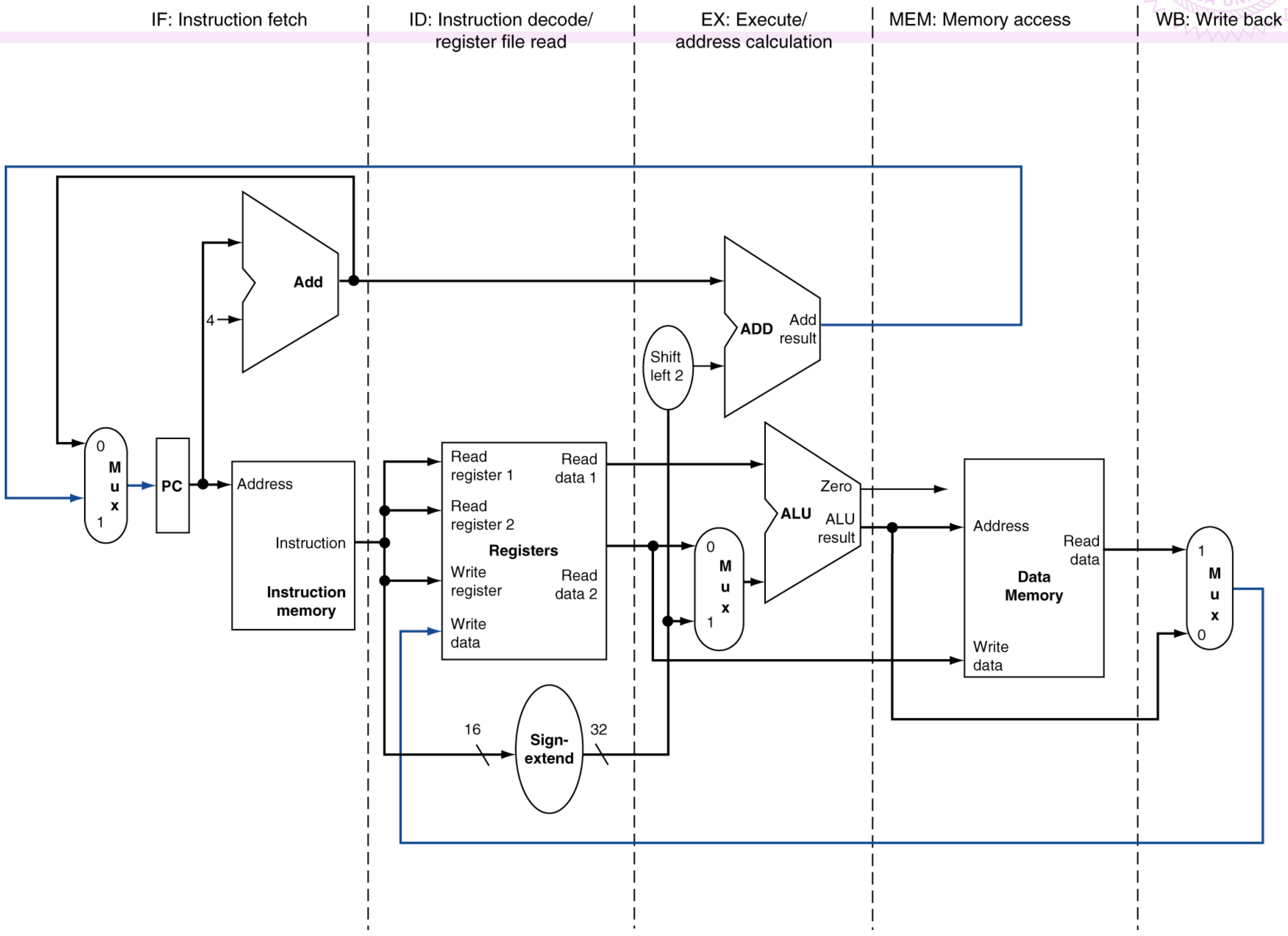
Prof. Ren-Shuo Liu
NTHU EE
Fall 2017



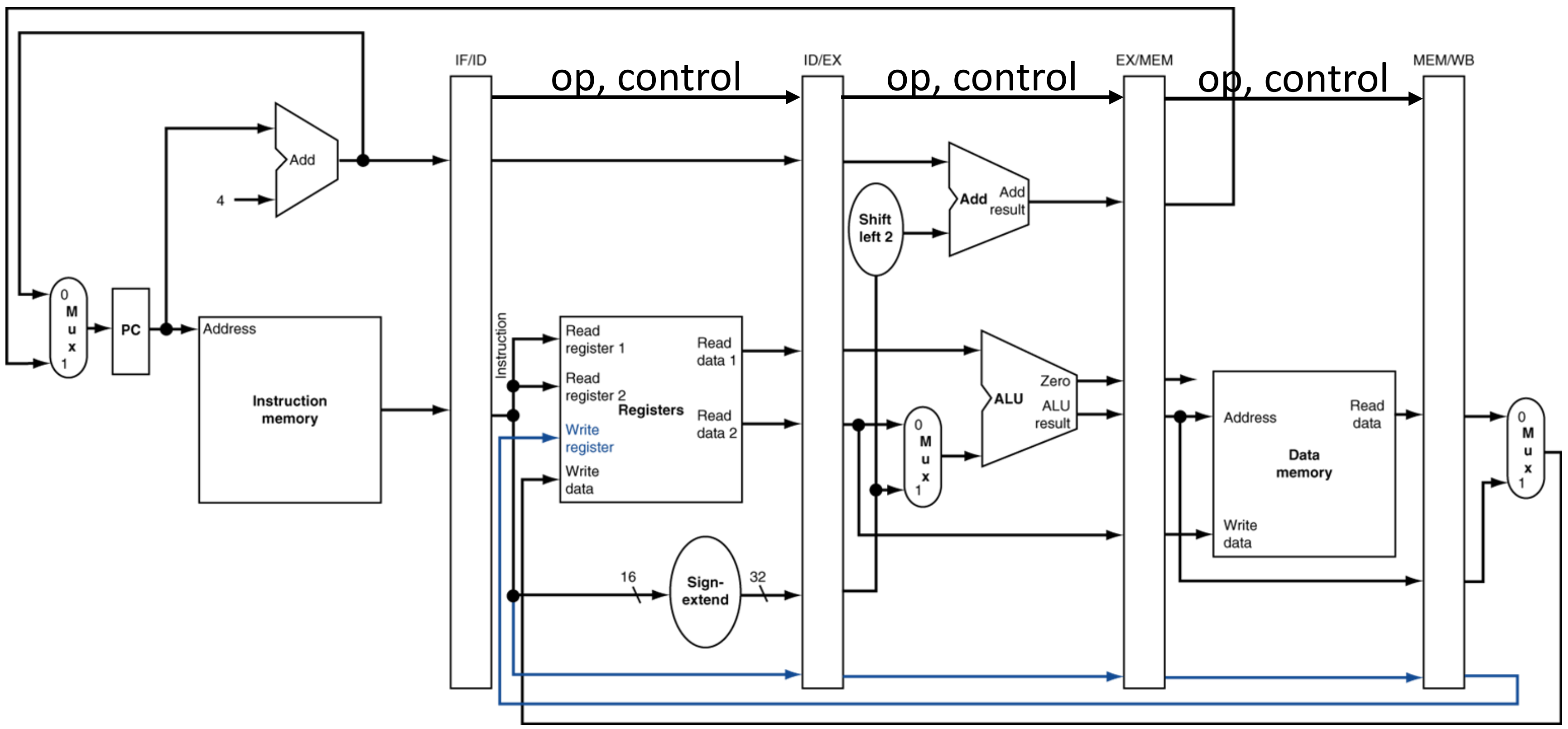
Outline

- Background
- Single-cycle design
- Pipelined design
  - Pipeline concepts and MIPS's pipeline
  - Cost and issues of pipelining
- Detailed pipelined datapath and control
  - Trace the pipeline
  - Dependencies, hazards, and forwarding
  - Stalls and exceptions

Recap Single-Cycle Datapath



Multi-Cycle Datapath

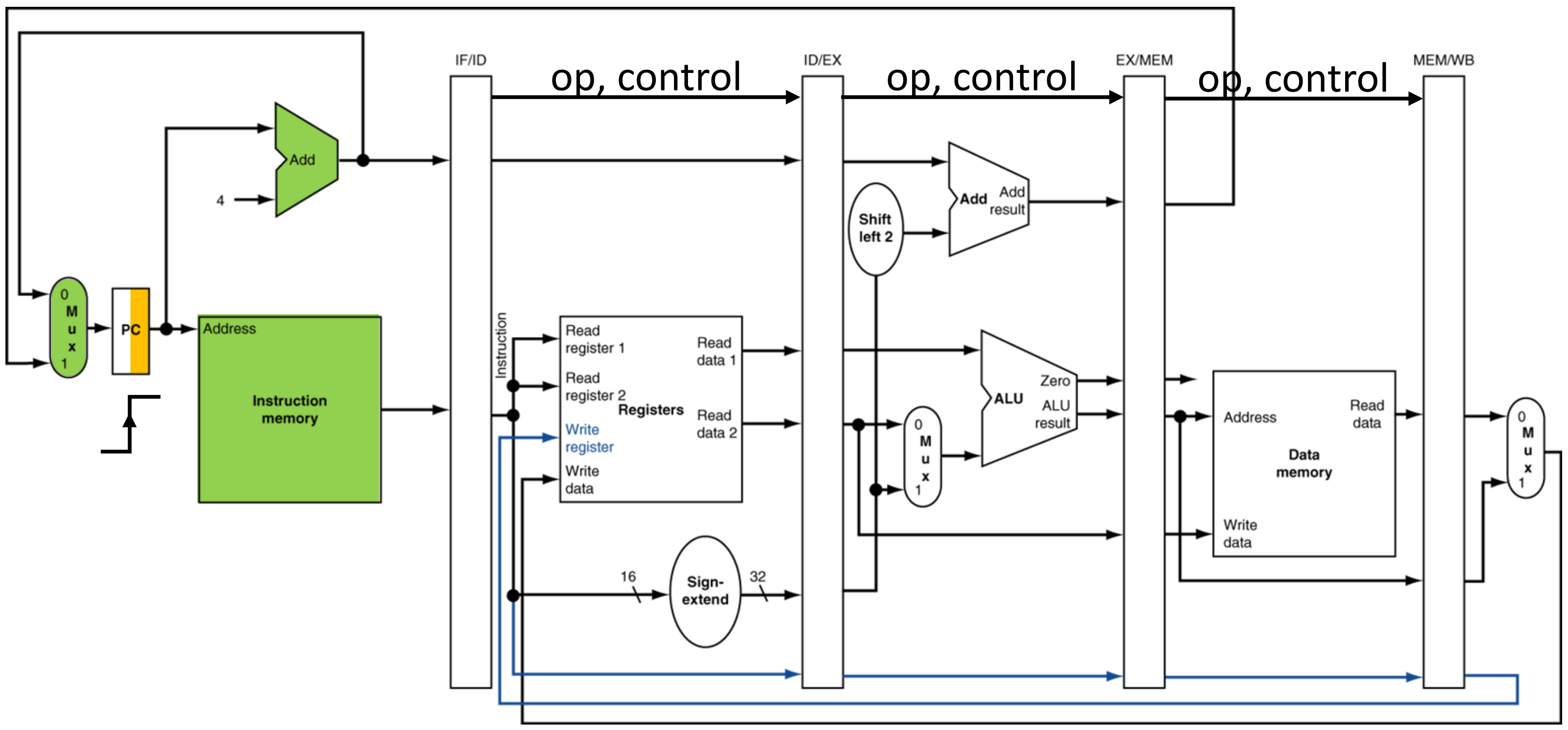


Trace the Pipeline

- Four examples
  - Only one lw instruction
  - Only one sw instruction
  - Only one R-type instruction
  - Five consecutive instructions

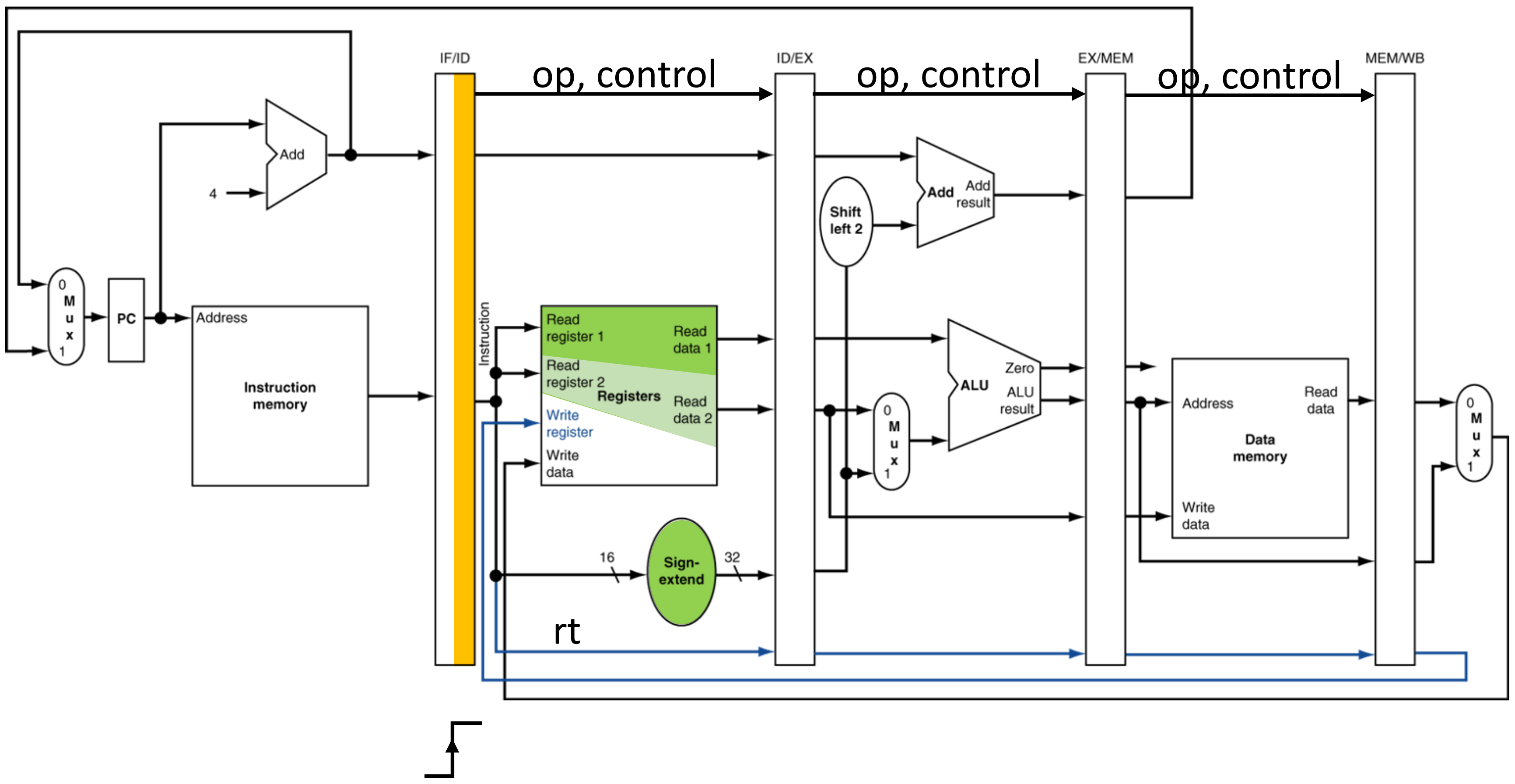
LW's 1st (IF) Stage/Cycle

pc



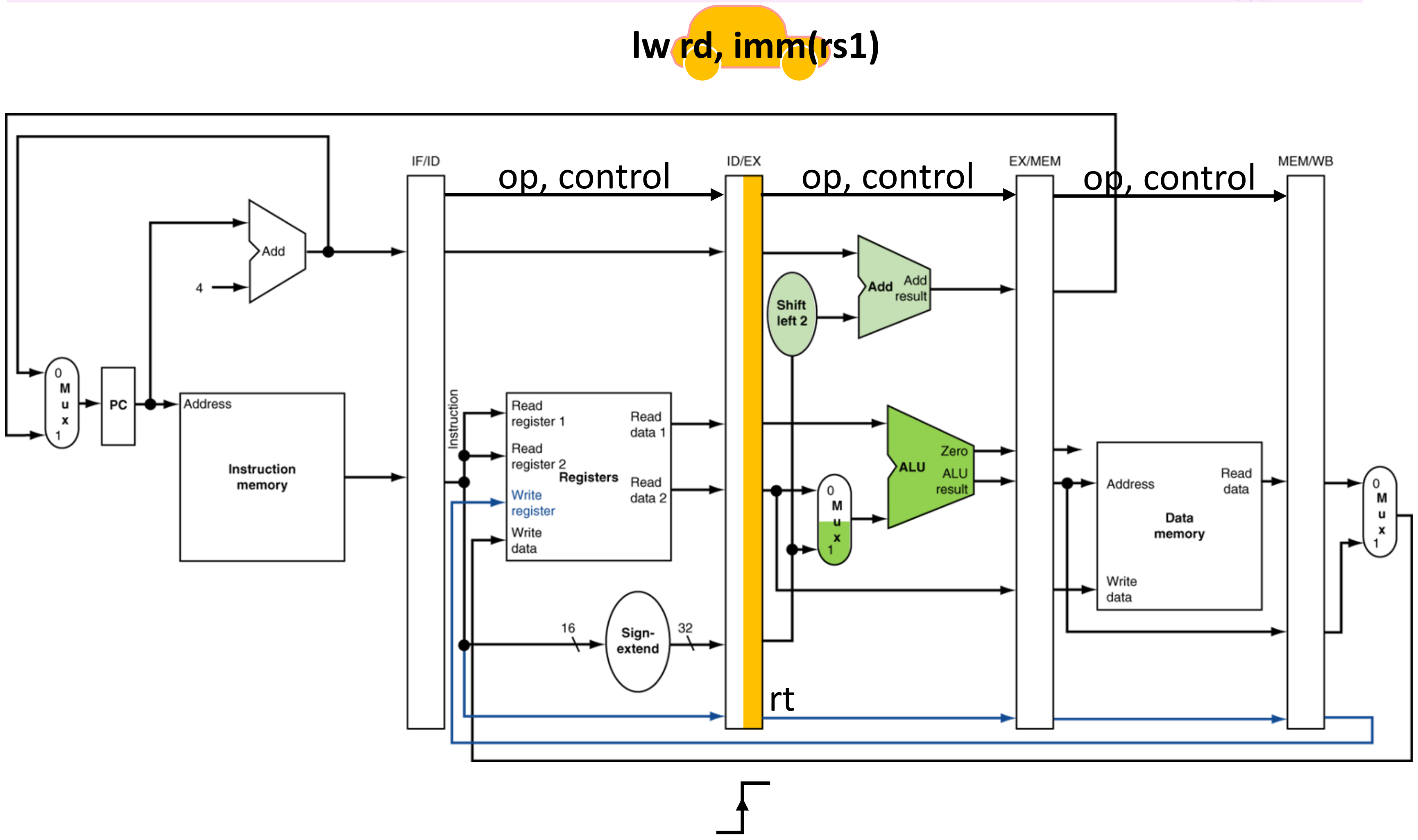
LW's 2nd (ID) Stage/Cycle

lw rd, imm(rs1)



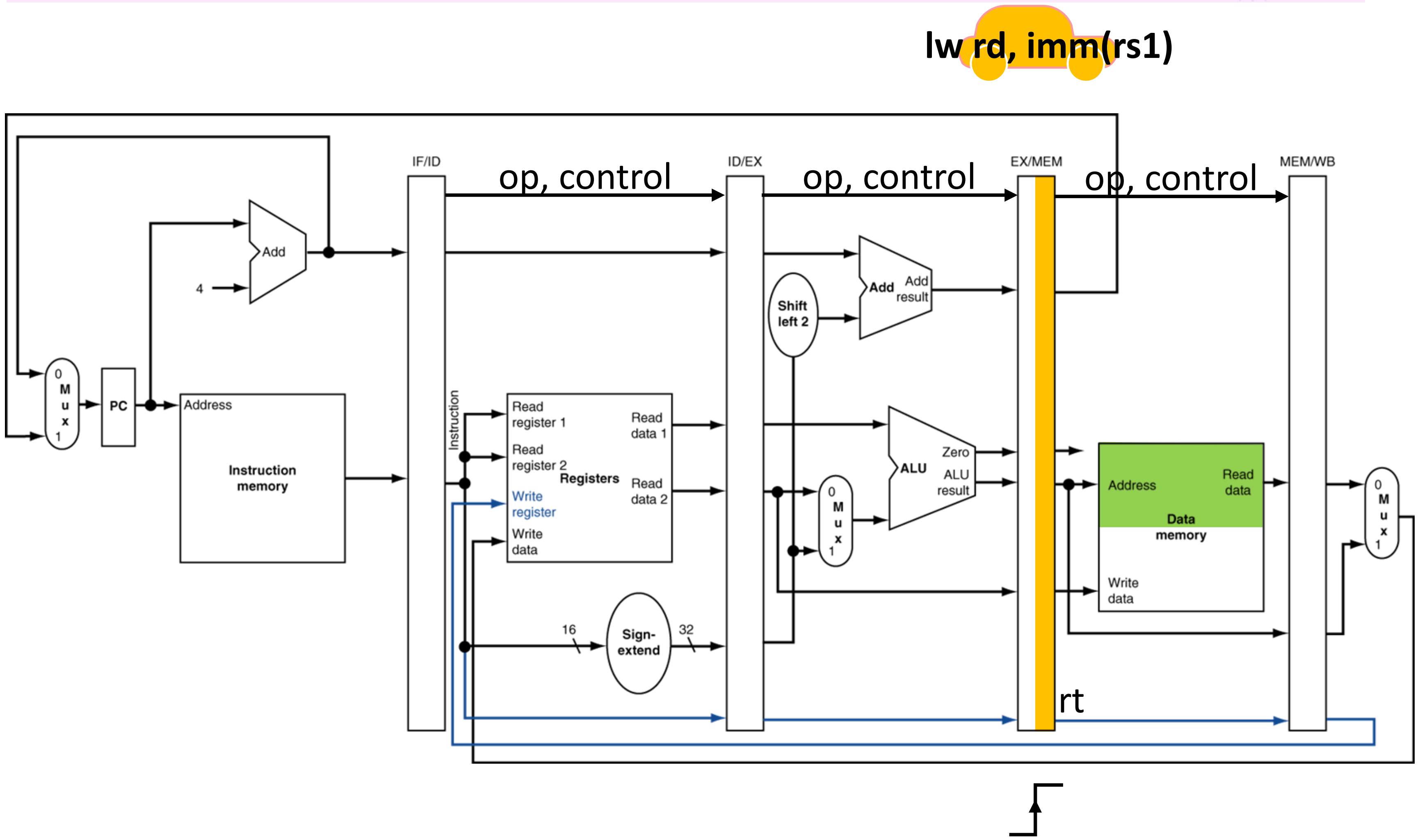
LW's 3rd (EX) Stage/Cycle

lw rd, imm(rs1)



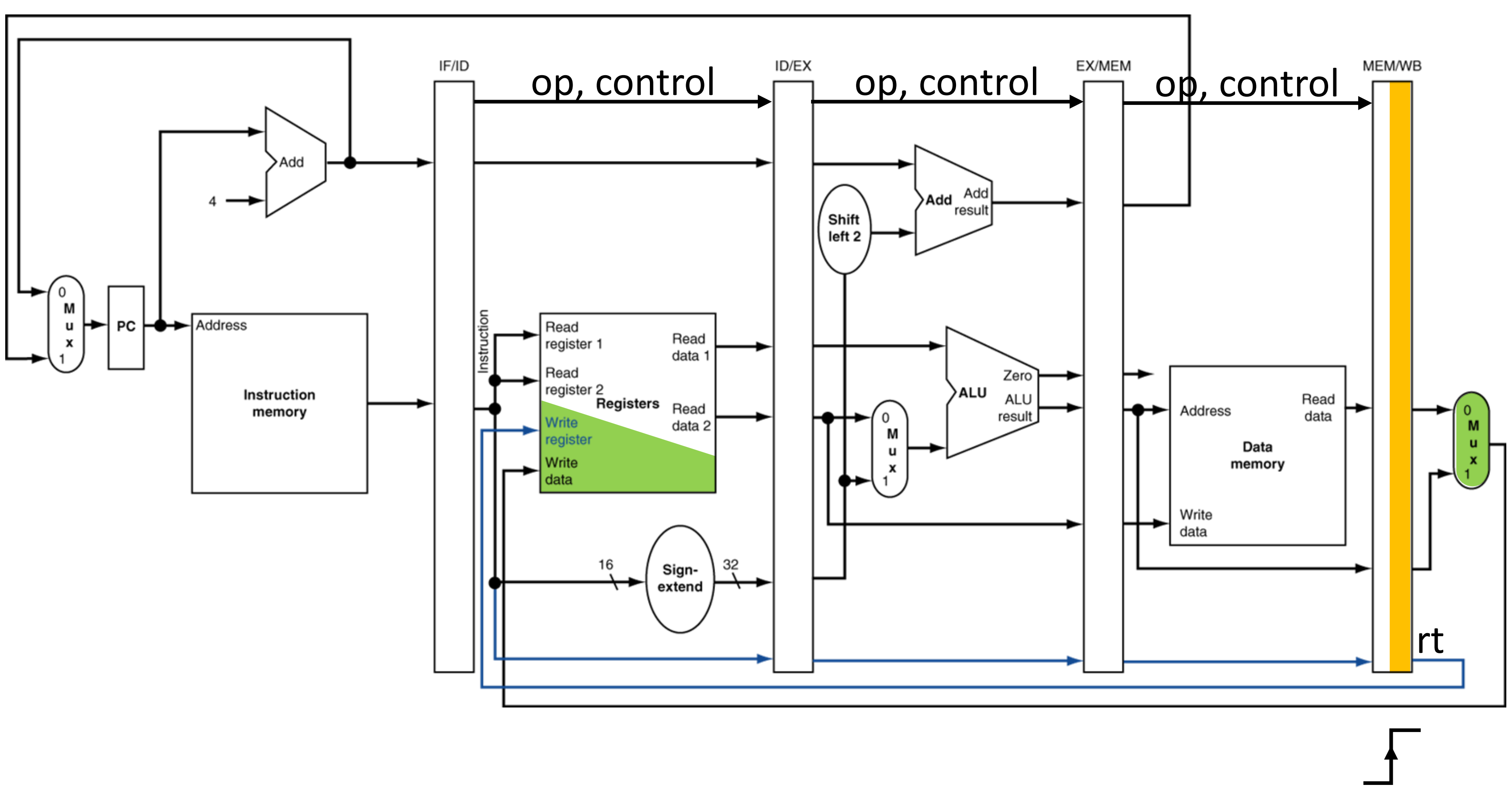
LW's 4th (MEM) Stage/Cycle

lw rd, imm(rs1)

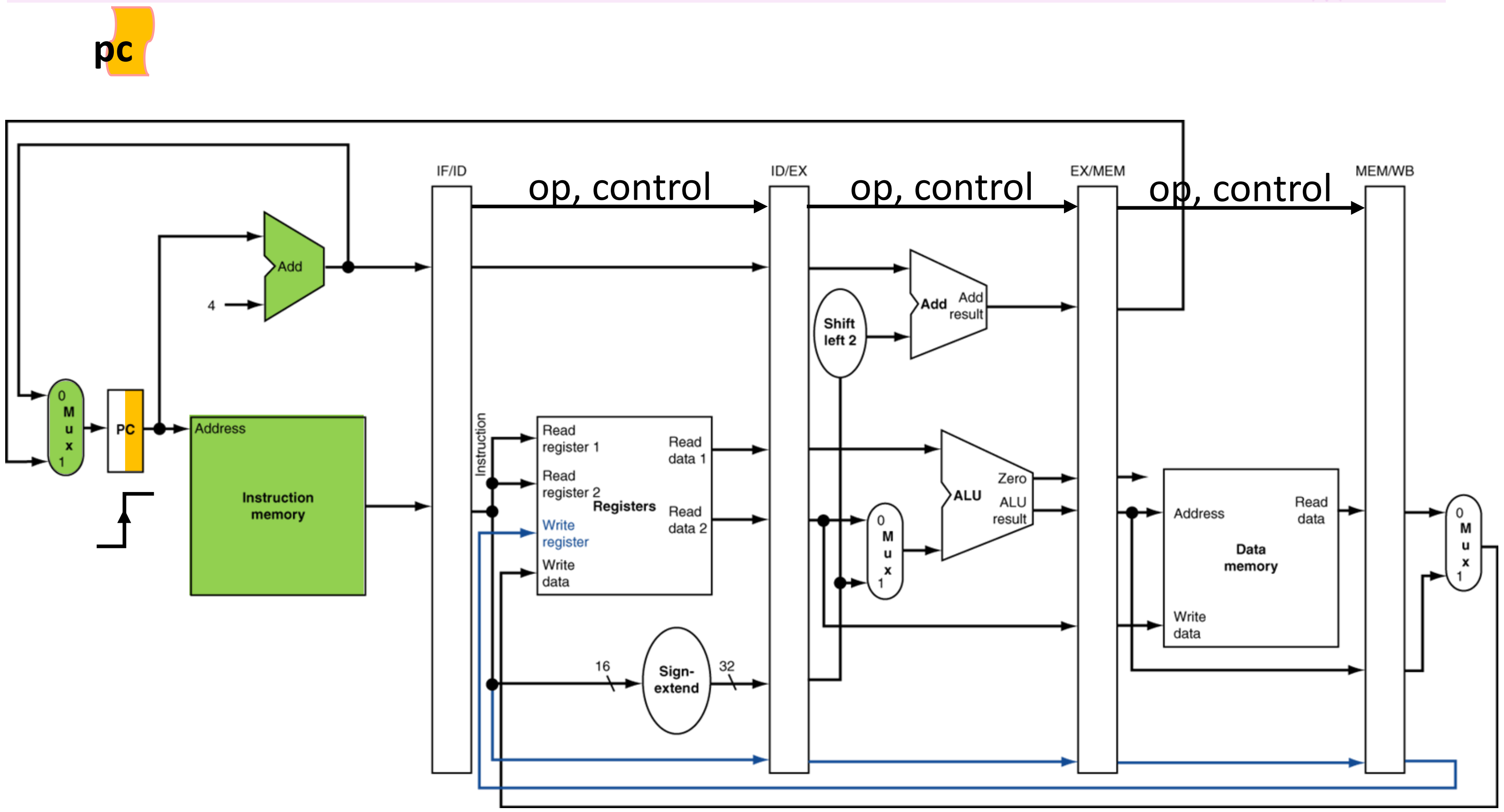


LW's 5th (WB) Stage/Cycle

lw rd, imm(rs1)

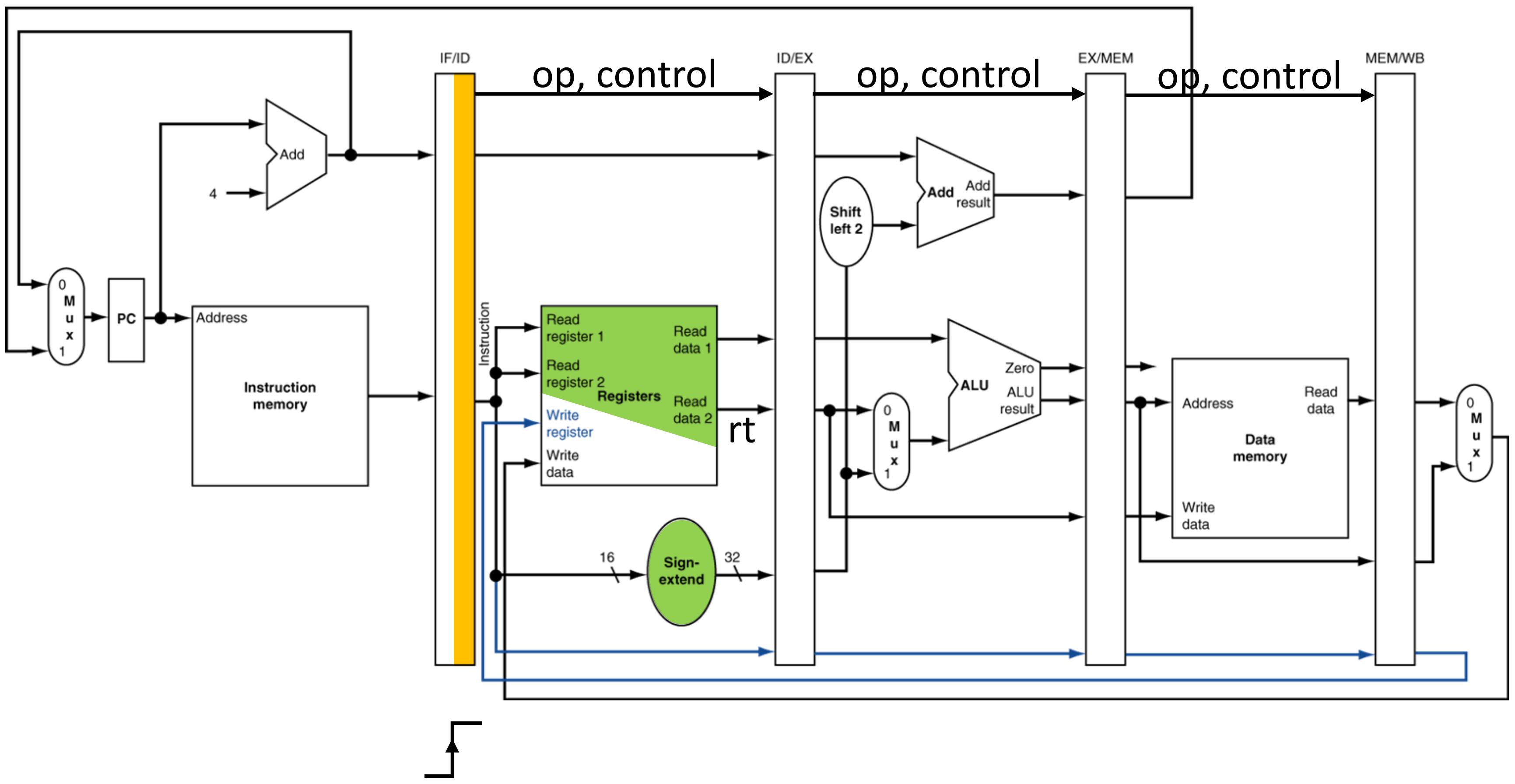


SW's 1st (IF) Stage/Cycle

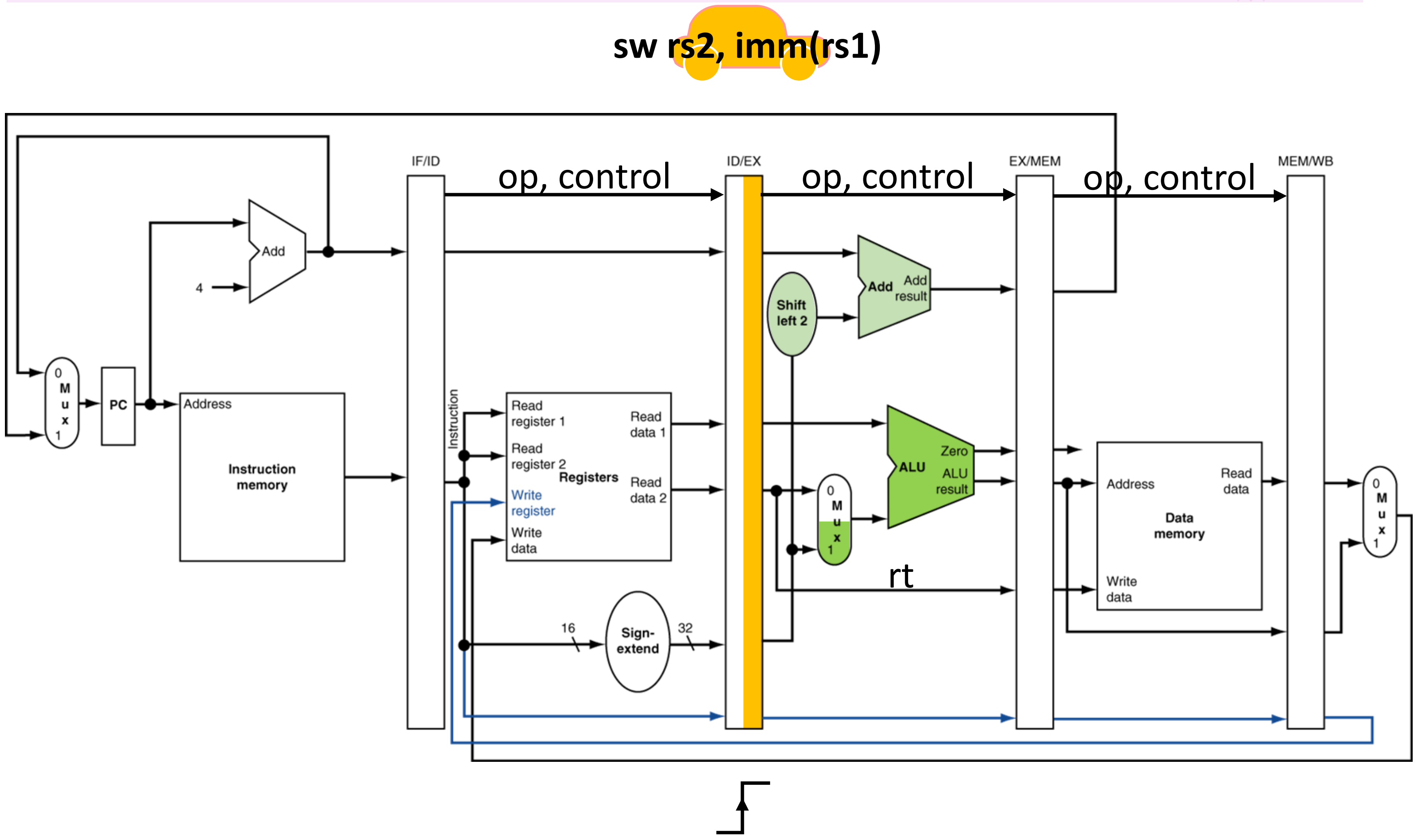


SW's 2nd (ID) Stage/Cycle

sw rs2, imm(rs1)

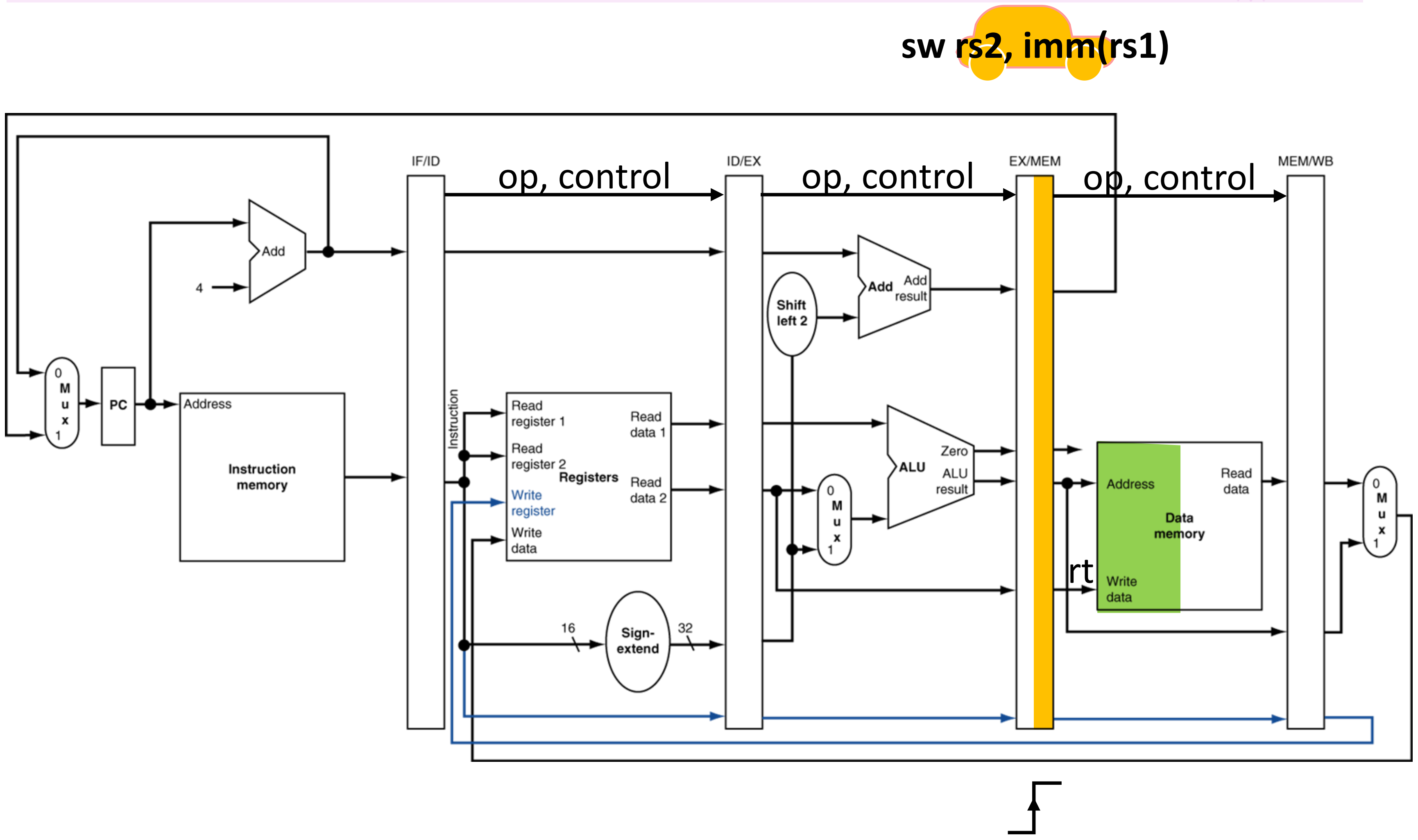


SW's 3rd (EX) Stage/Cycle



SW's 4th (MEM) Stage/Cycle

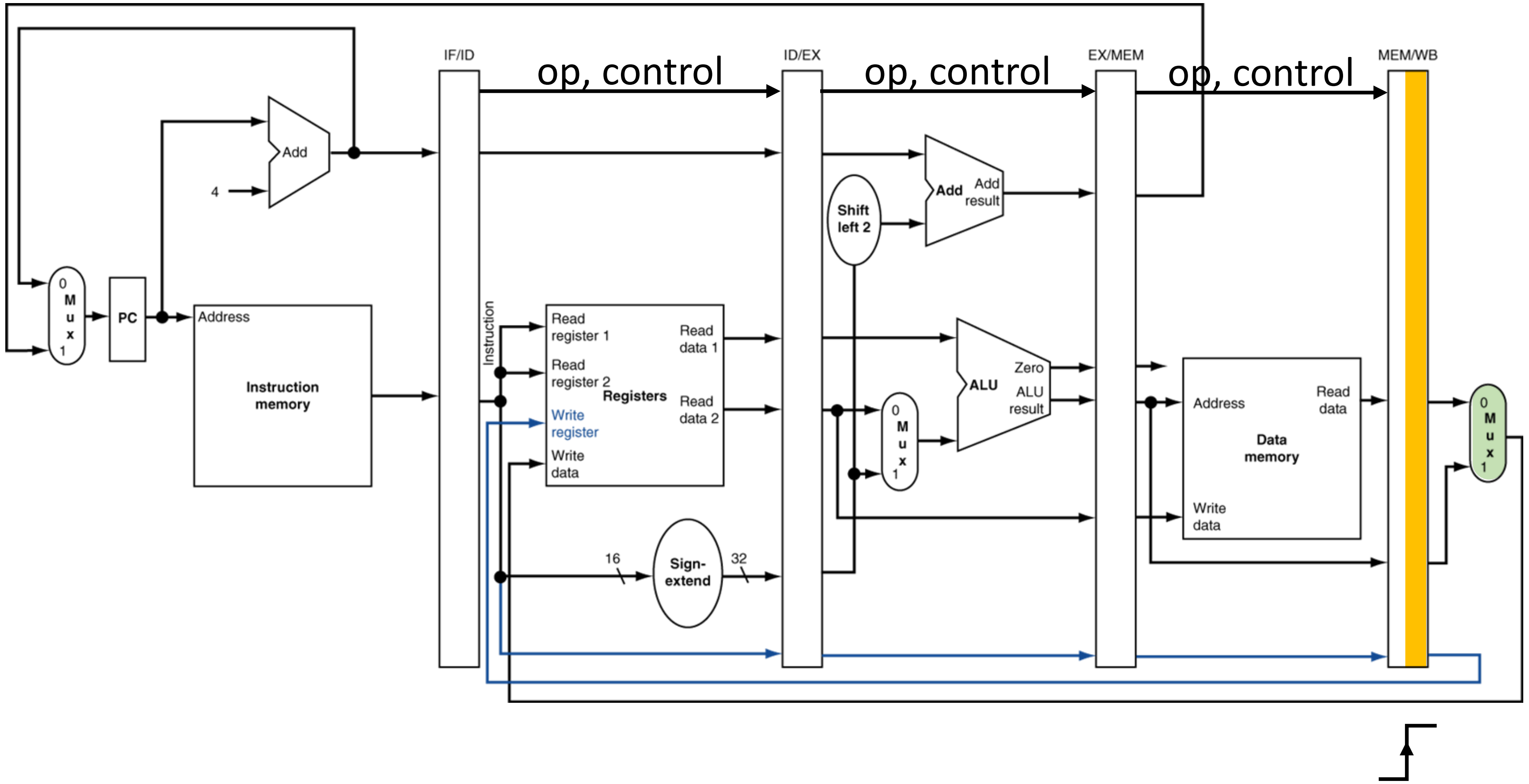
sw rs2, imm(rs1)



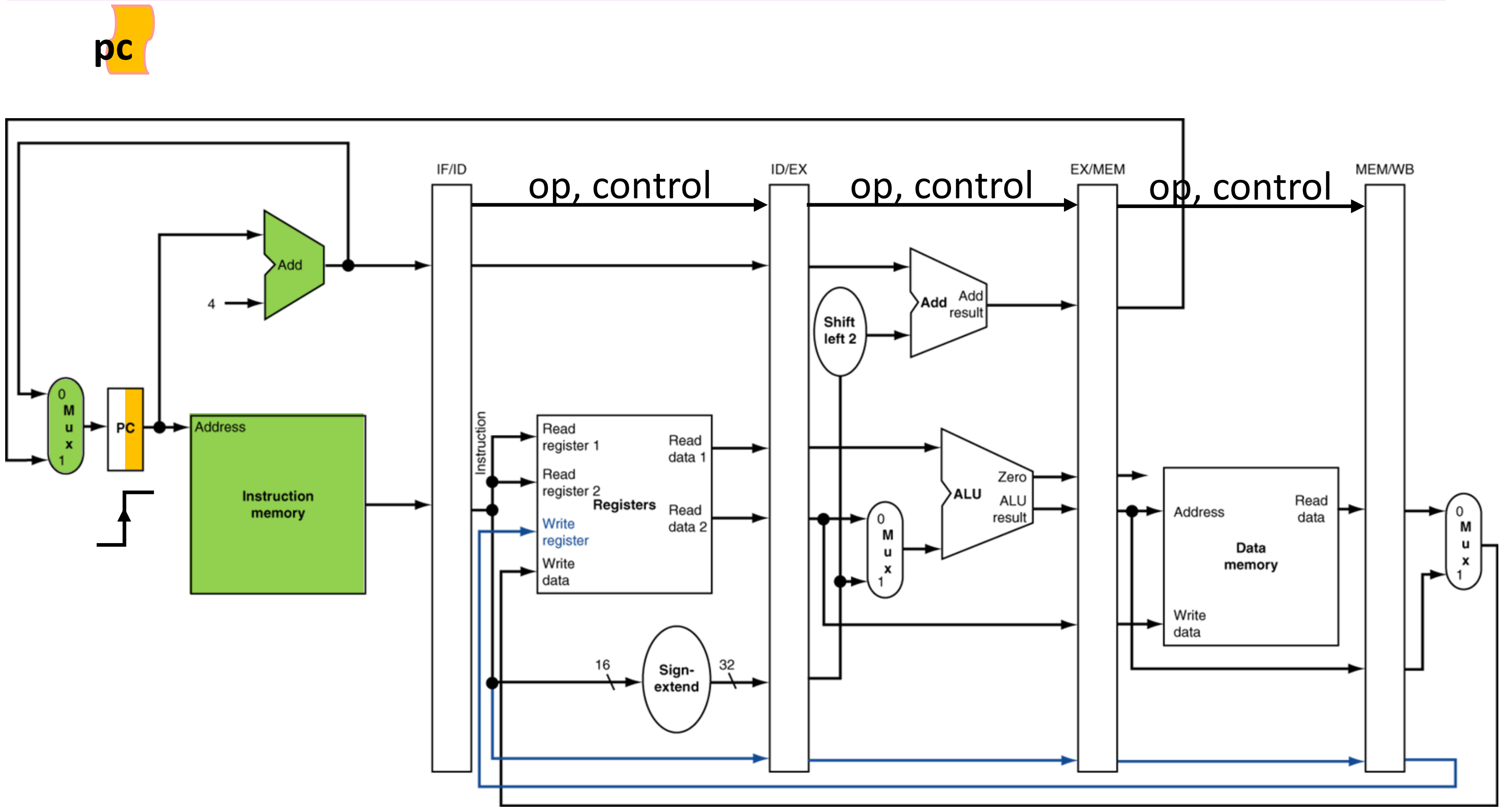
SW's 5th (WB) Stage/Cycle

- Do nothing

sw rs2, imm(rs1)

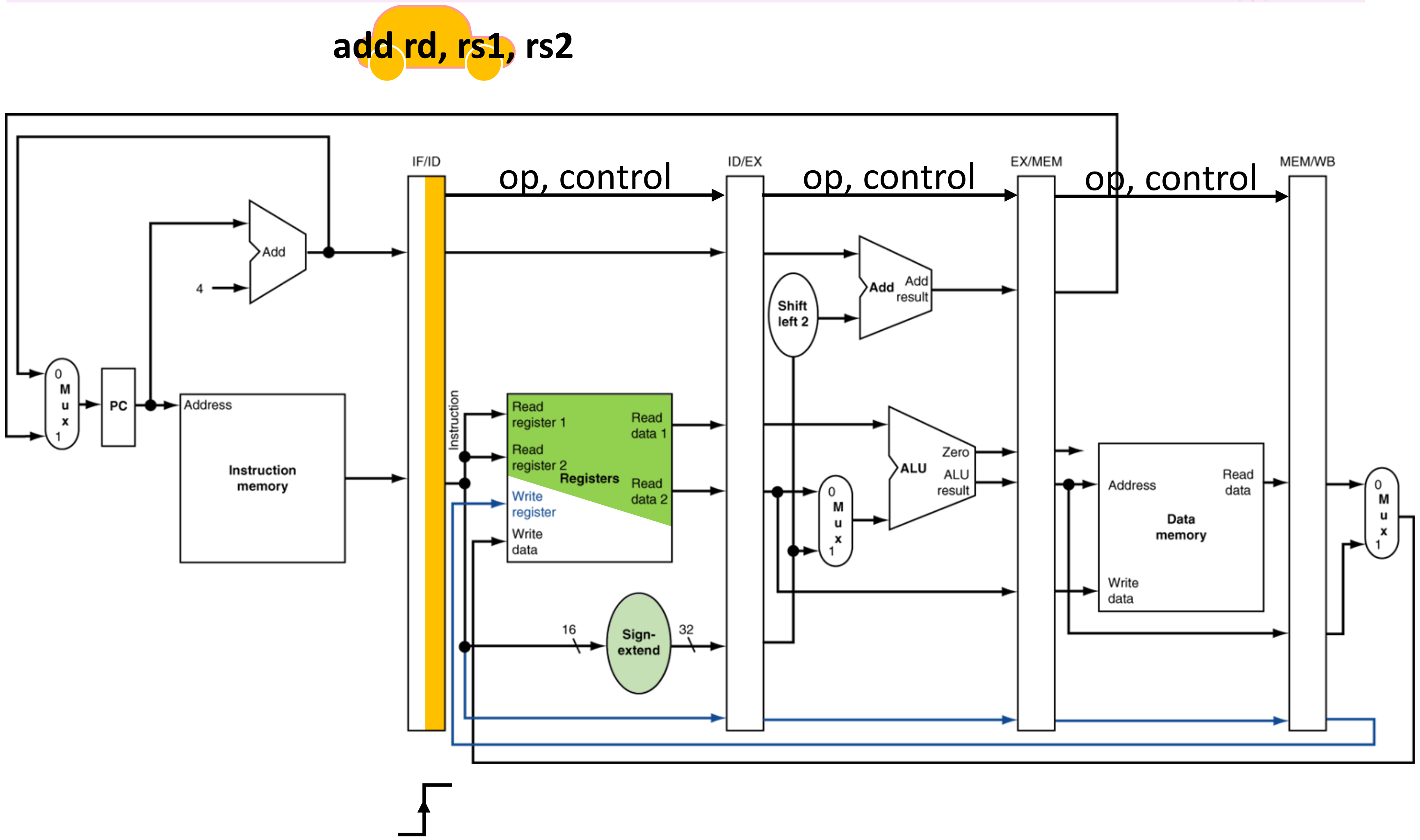


R-Type's 1st (IF) Stage/Cycle

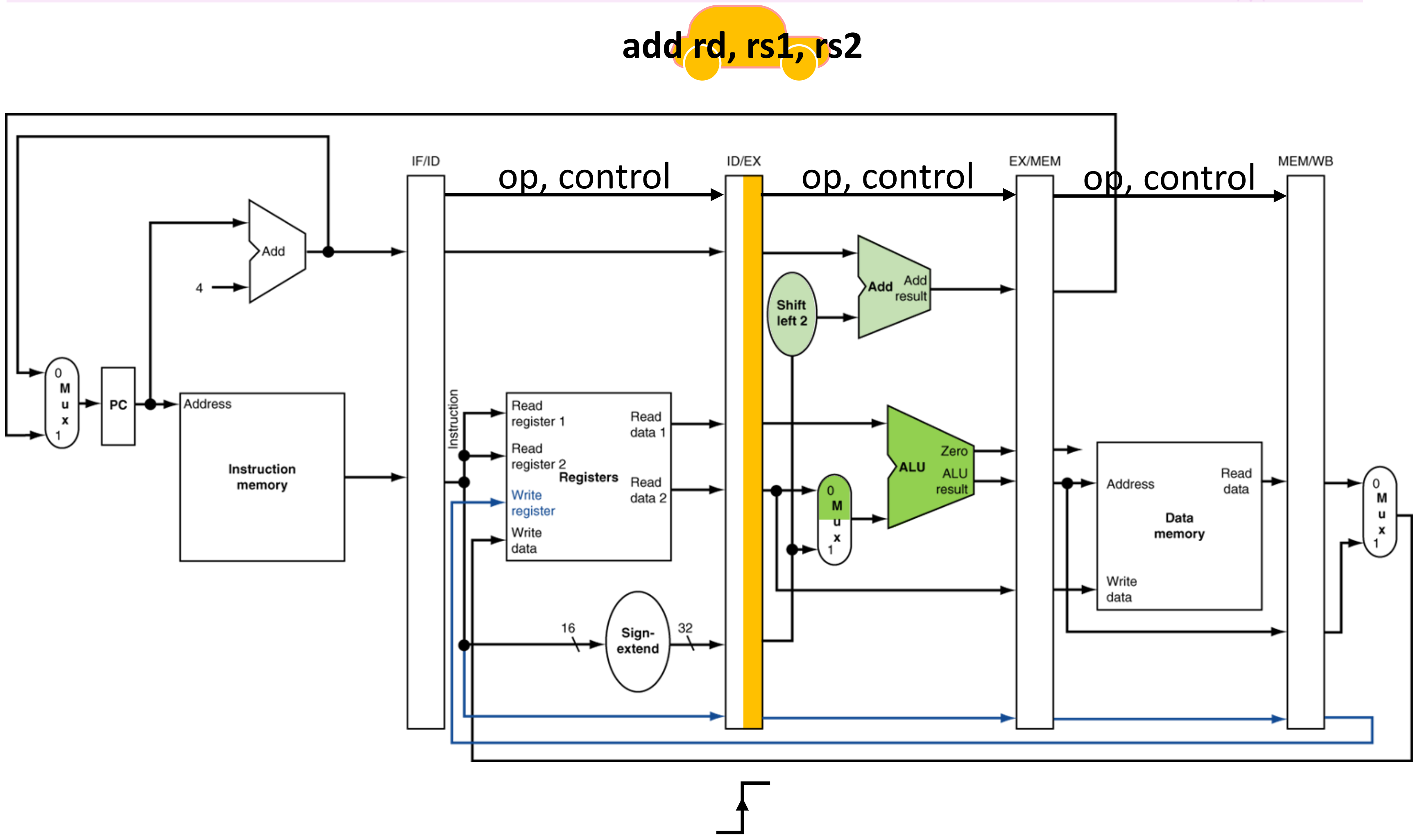


R-Type's 2nd (ID) Stage/Cycle

add rd, rs1, rs2

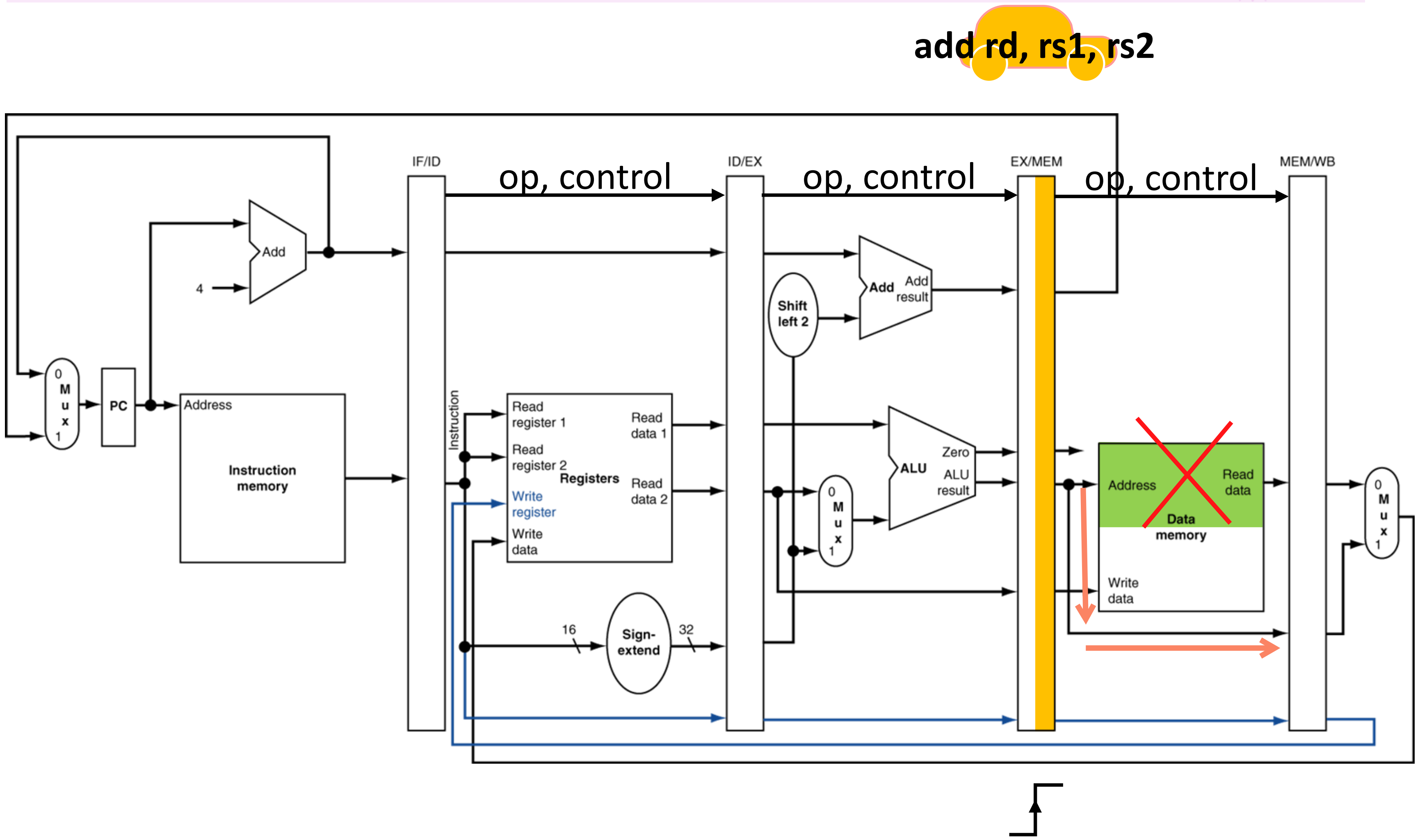


R-Type's 3rd (EX) Stage/Cycle



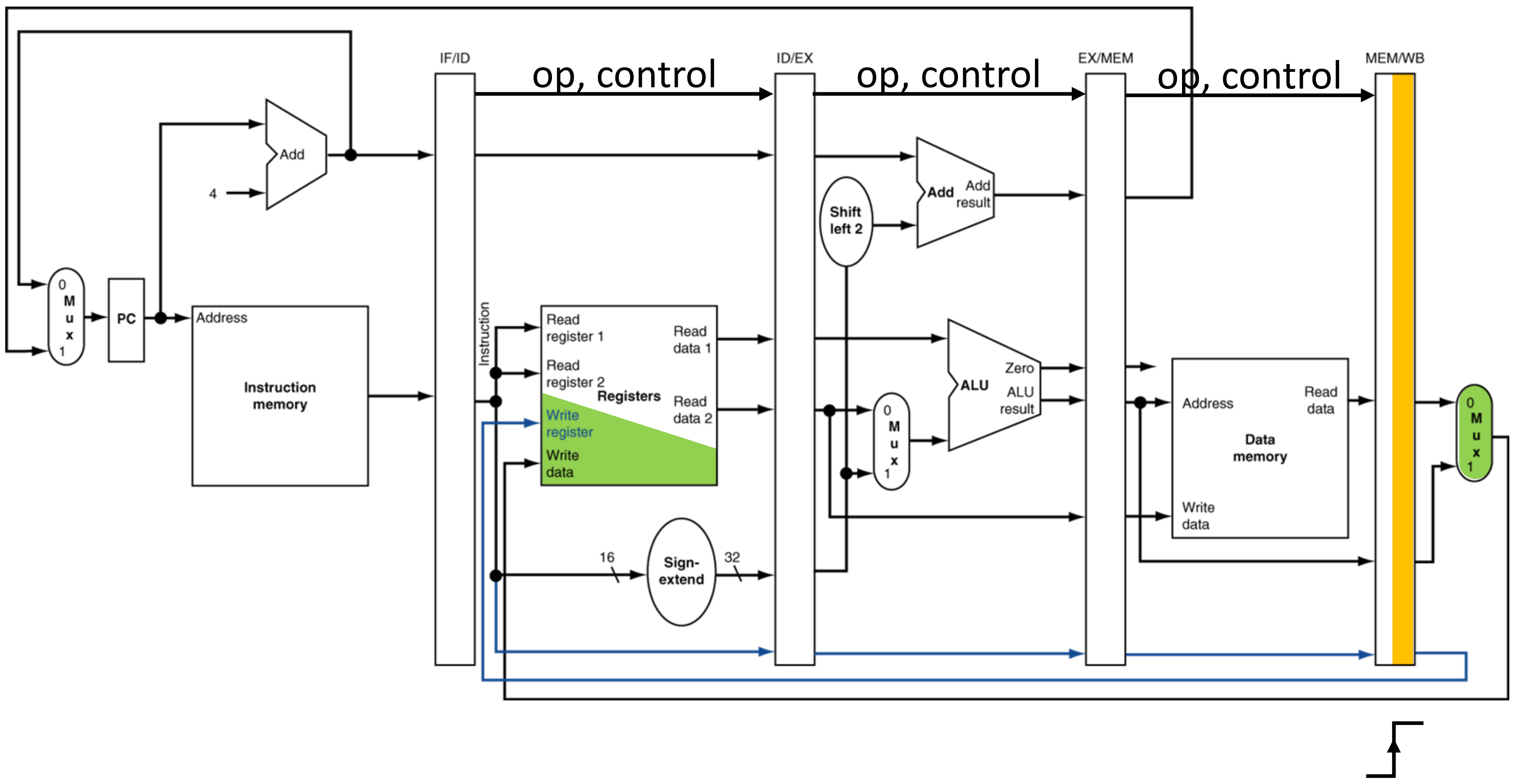
R-Type's 4th (MEM) Stage/Cycle

add rd, rs1, rs2



R-Type's 5th (WB) Stage/Cycle

add rd, rs1, rs2

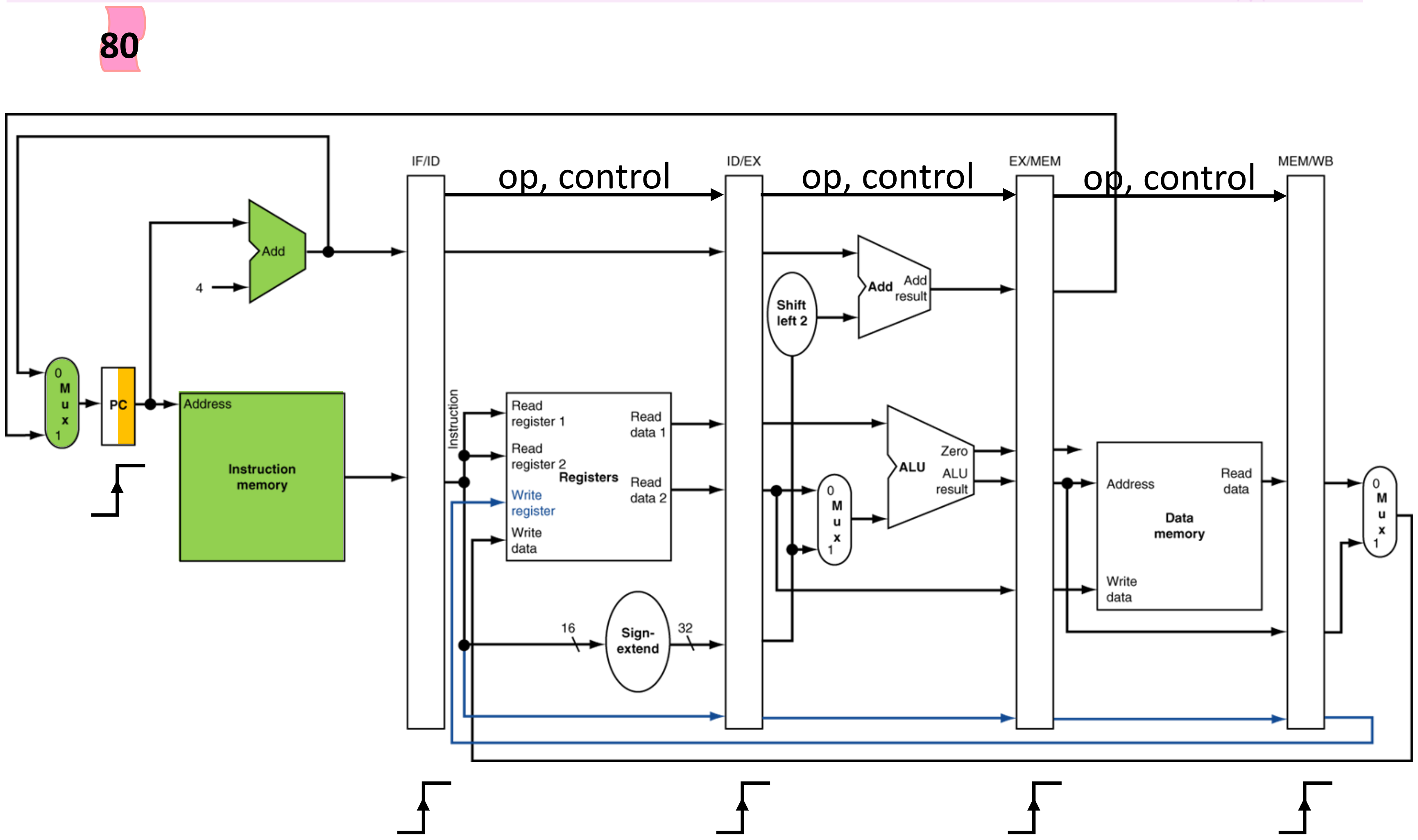


Consecutive Instruction Example

```
                                # int a;
                                # int b;
                                # int c;
  …  …                          # …  …
80:  lw  t1, 0(gp)              #
84:  lw  t2, 4(gp)              #
88:  nop                        #
92:  add t3, t1, t2             # c = a + b;
96:  sw  t3, 8(gp)              #
```

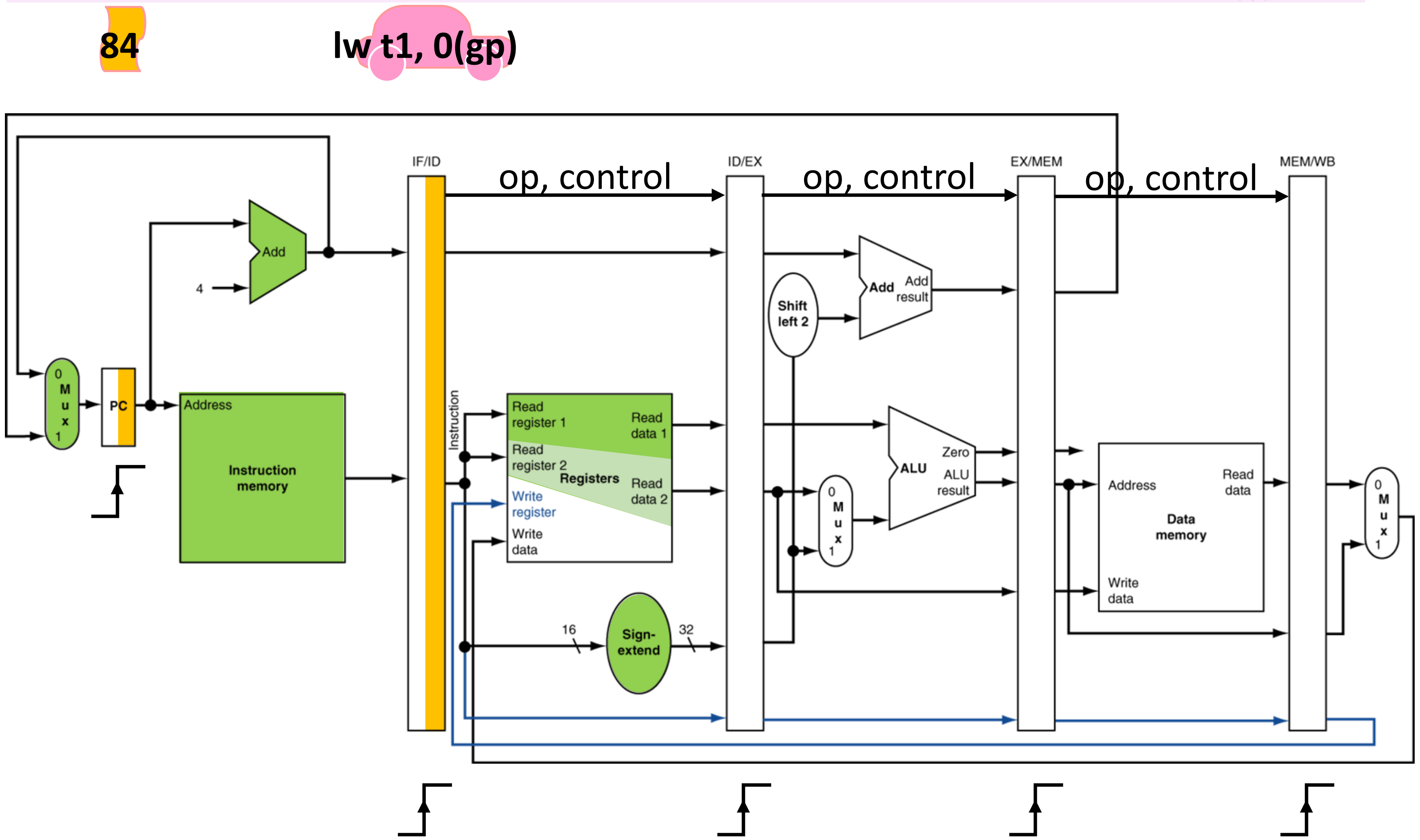
First Cycle

```
80:  lw   t1, 0(gp)
84:  lw   t2, 4(gp)
88:  nop
92:  add  t3, t1, t2
96:  sw   t3, 8(gp)
```



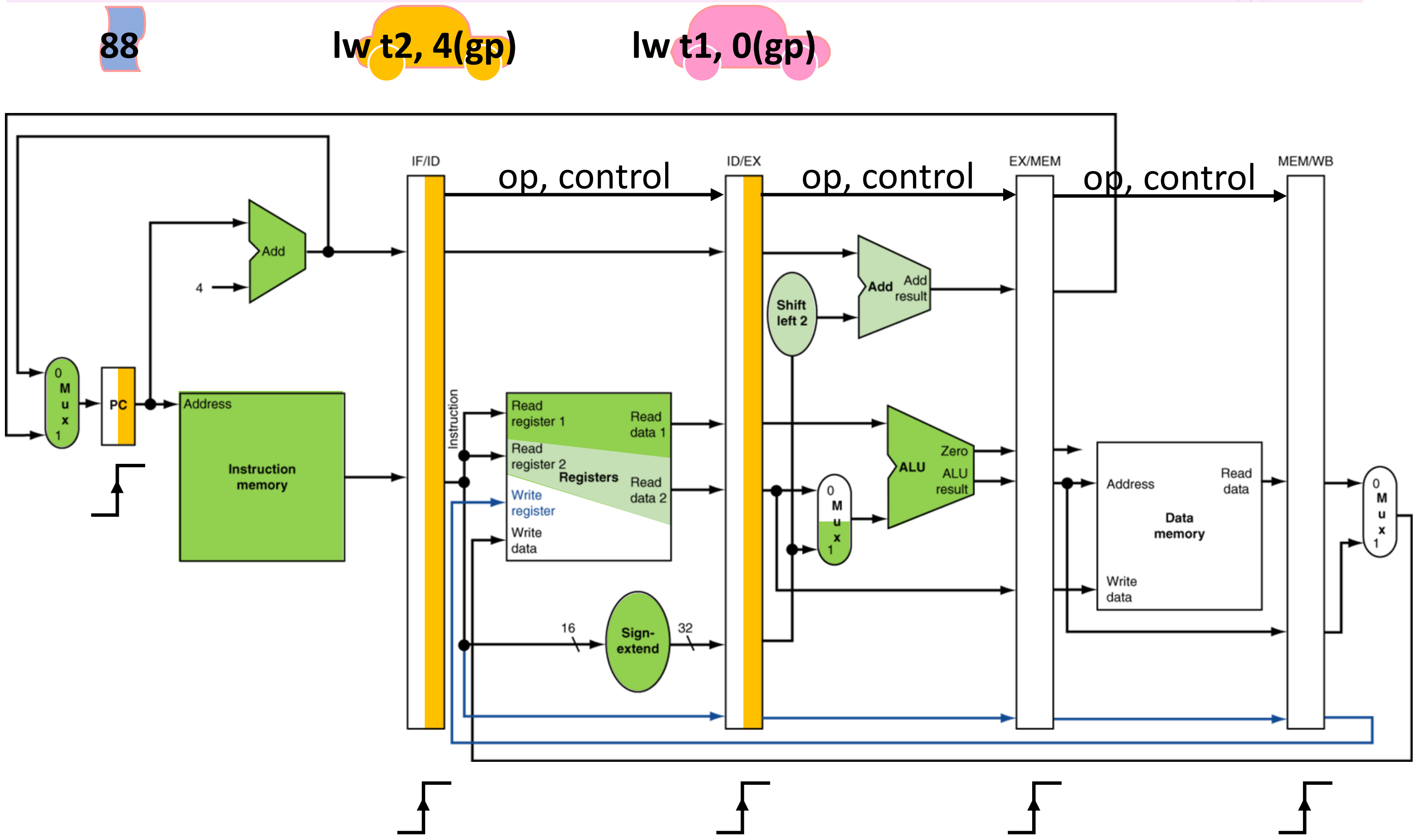
```
80:  lw   t1, 0(gp)
84:  lw   t2, 4(gp)
88:  nop
92:  add  t3, t1, t2
96:  sw   t3, 8(gp)
```

Second Cycle



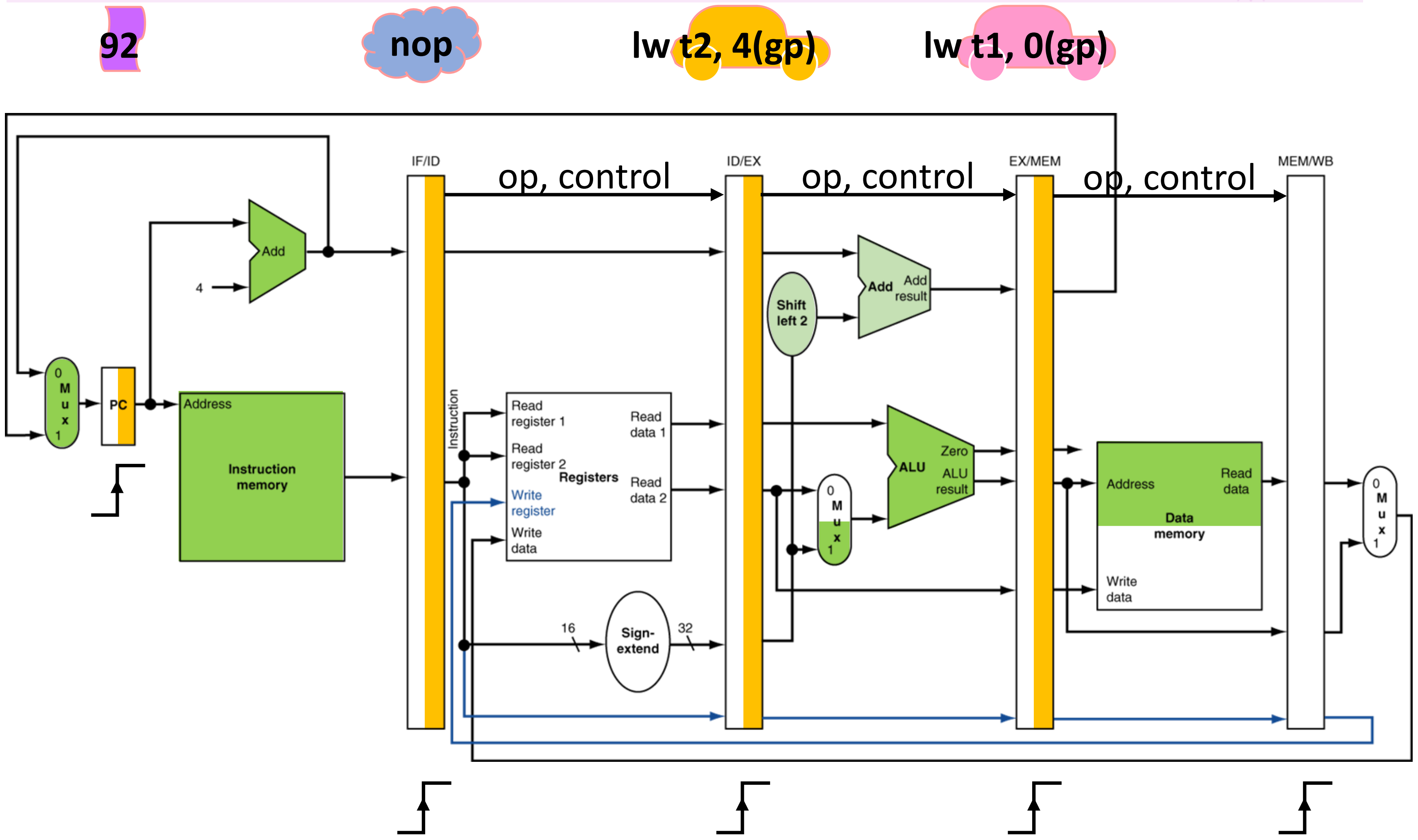
Third Cycle

```
80:  lw  t1, 0(gp)
84:  lw  t2, 4(gp)
88:  nop
92:  add t3, t1, t2
96:  sw  t3, 8(gp)
```



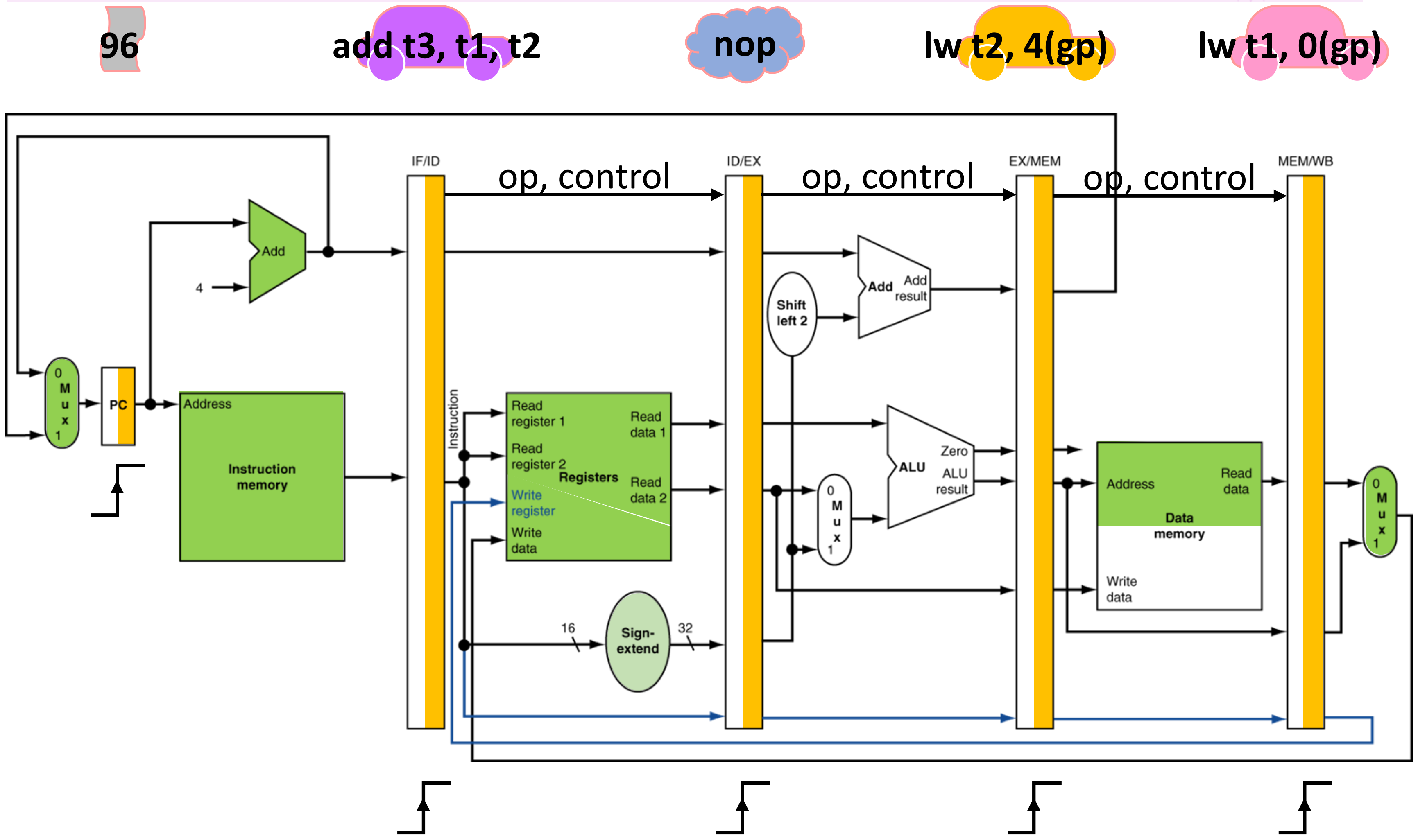
Forth Cycle

```
80:  lw  t1, 0(gp)
84:  lw  t2, 4(gp)
88:  nop
92:  add t3, t1, t2
96:  sw  t3, 8(gp)
```



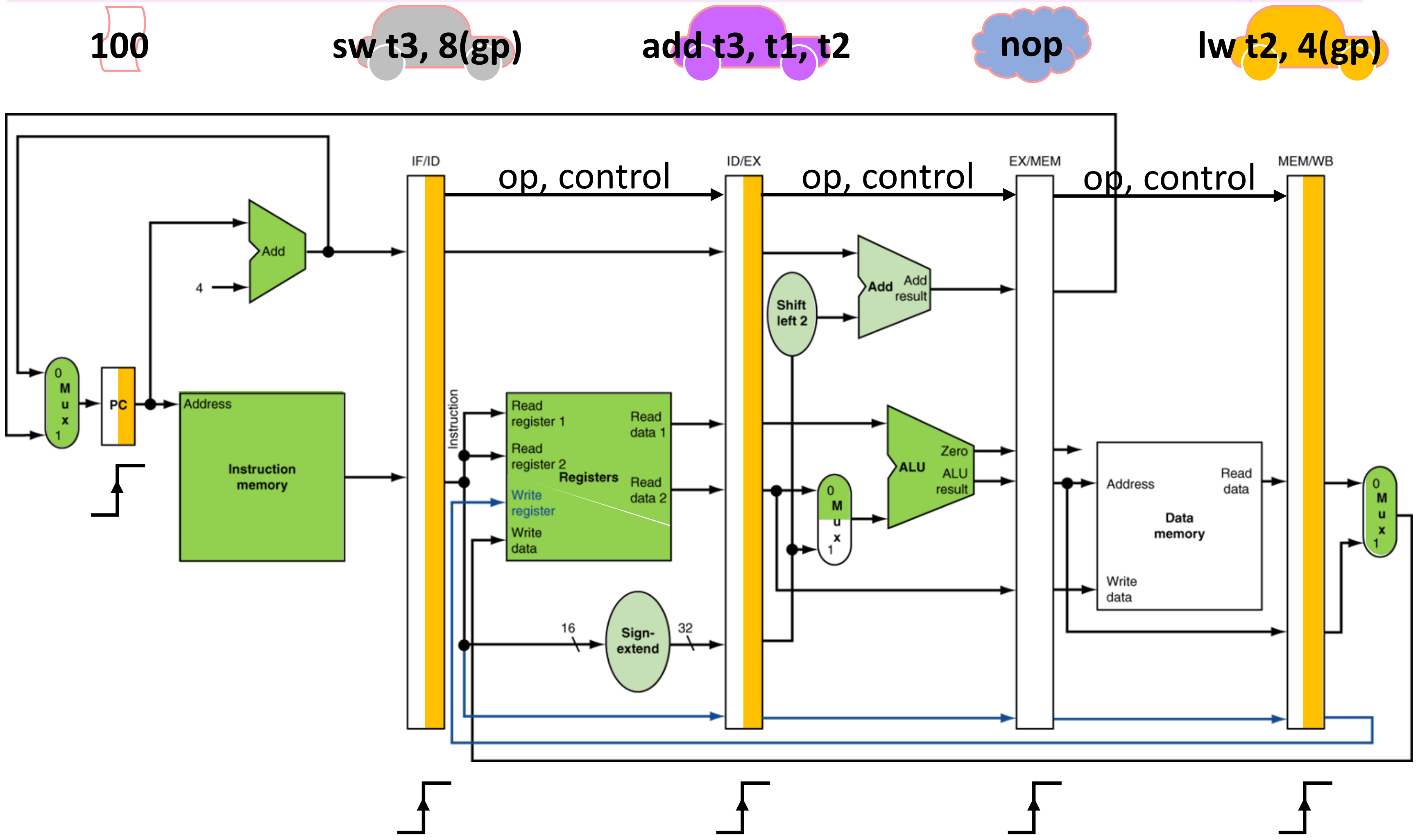
Fifth Cycle

```
80:  lw  t1, 0(gp)
84:  lw  t2, 4(gp)
88:  nop
92:  add t3, t1, t2
96:  sw  t3, 8(gp)
```



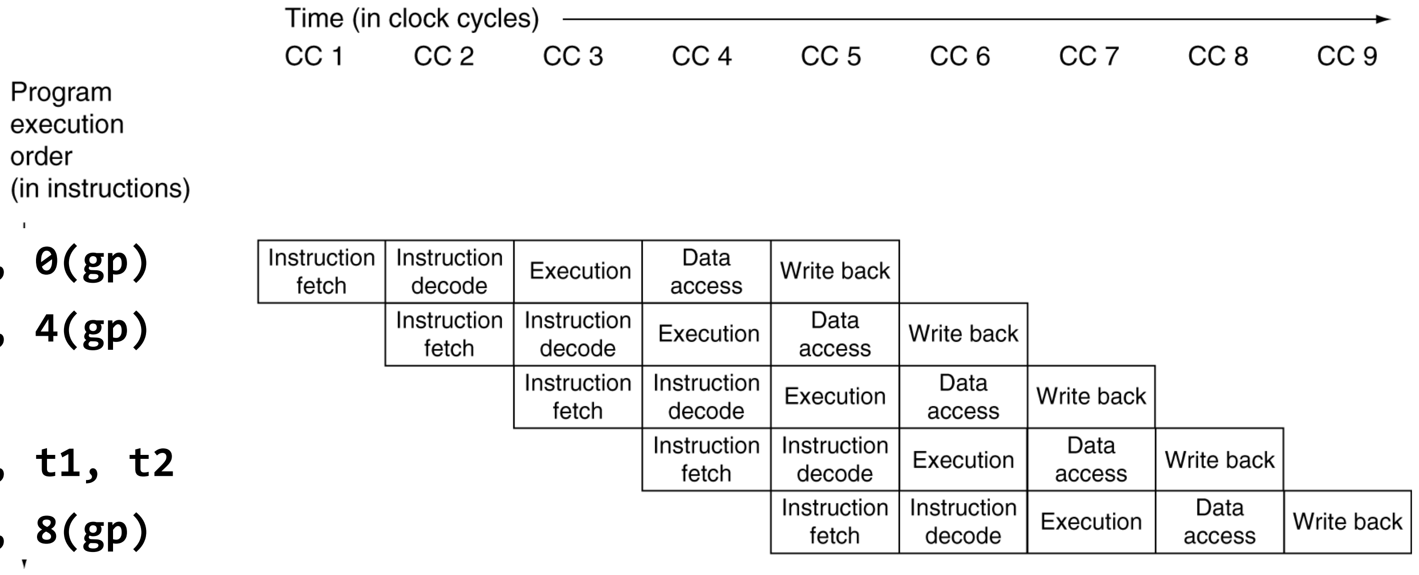
Sixth Cycle

```
80:  lw  t1, 0(gp)
84:  lw  t2, 4(gp)
88:  nop
92:  add t3, t1, t2
96:  sw  t3, 8(gp)
```



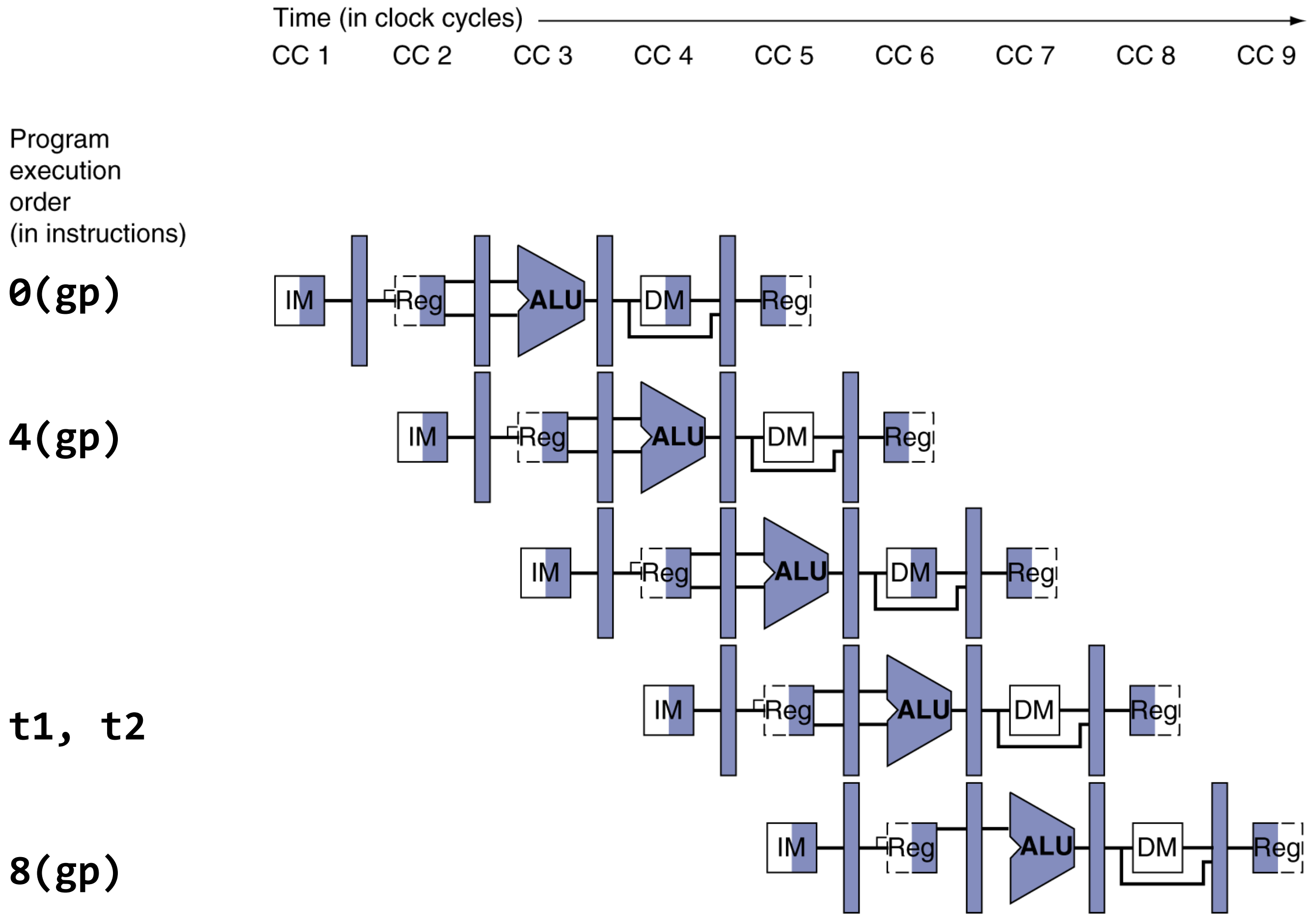
Simplified Pipeline Diagram

```
lw  t1, 0(gp)
lw  t2, 4(gp)
nop
add t3, t1, t2
sw  t3, 8(gp)
```



CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9
Instruction fetch	Instruction decode	Execution	Data access	Write back				
	Instruction fetch	Instruction decode	Execution	Data access	Write back			
		Instruction fetch	Instruction decode	Execution	Data access	Write back		
			Instruction fetch	Instruction decode	Execution	Data access	Write back	
				Instruction fetch	Instruction decode	Execution	Data access	Write back

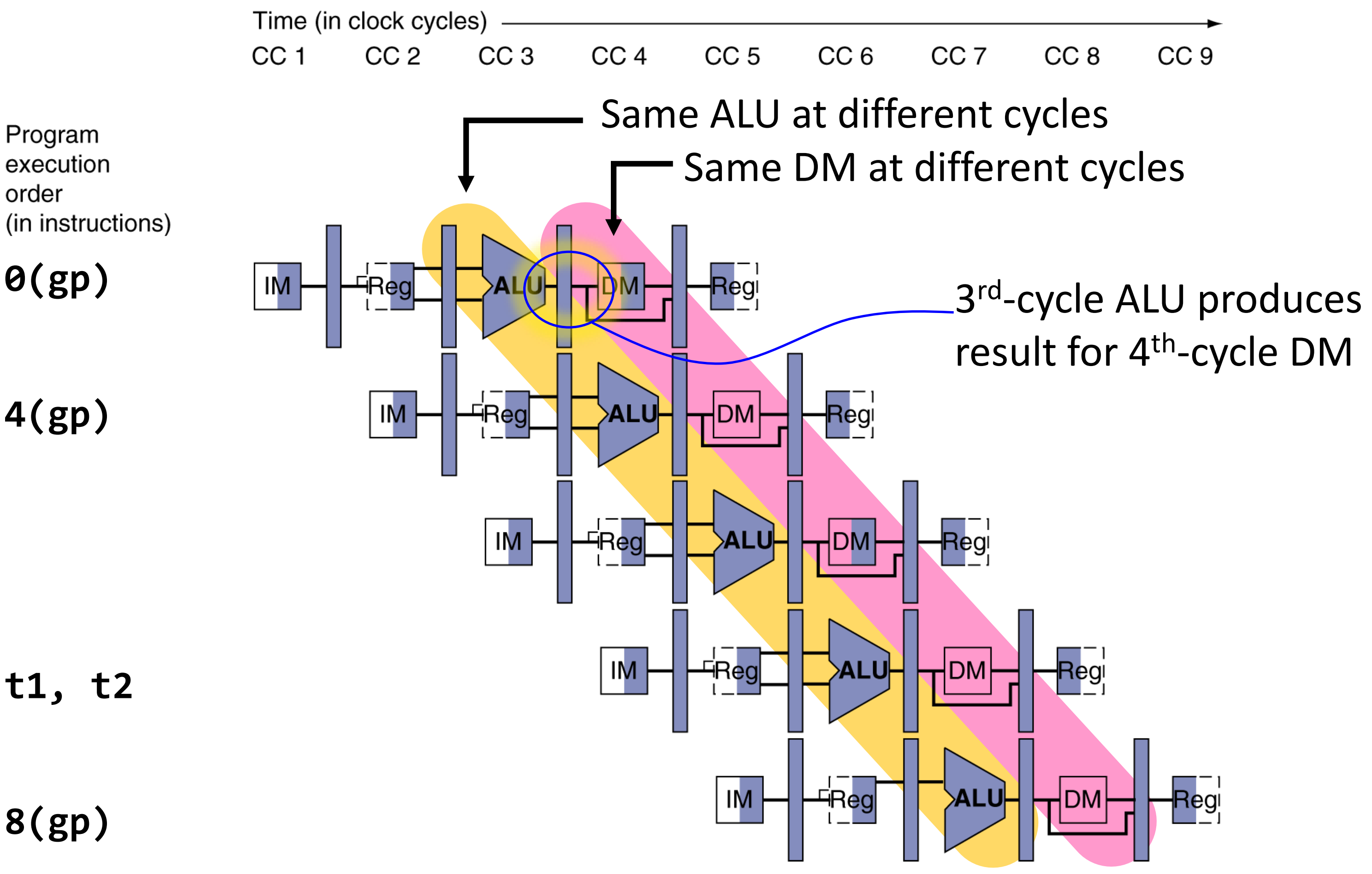
Resource Usage Diagram



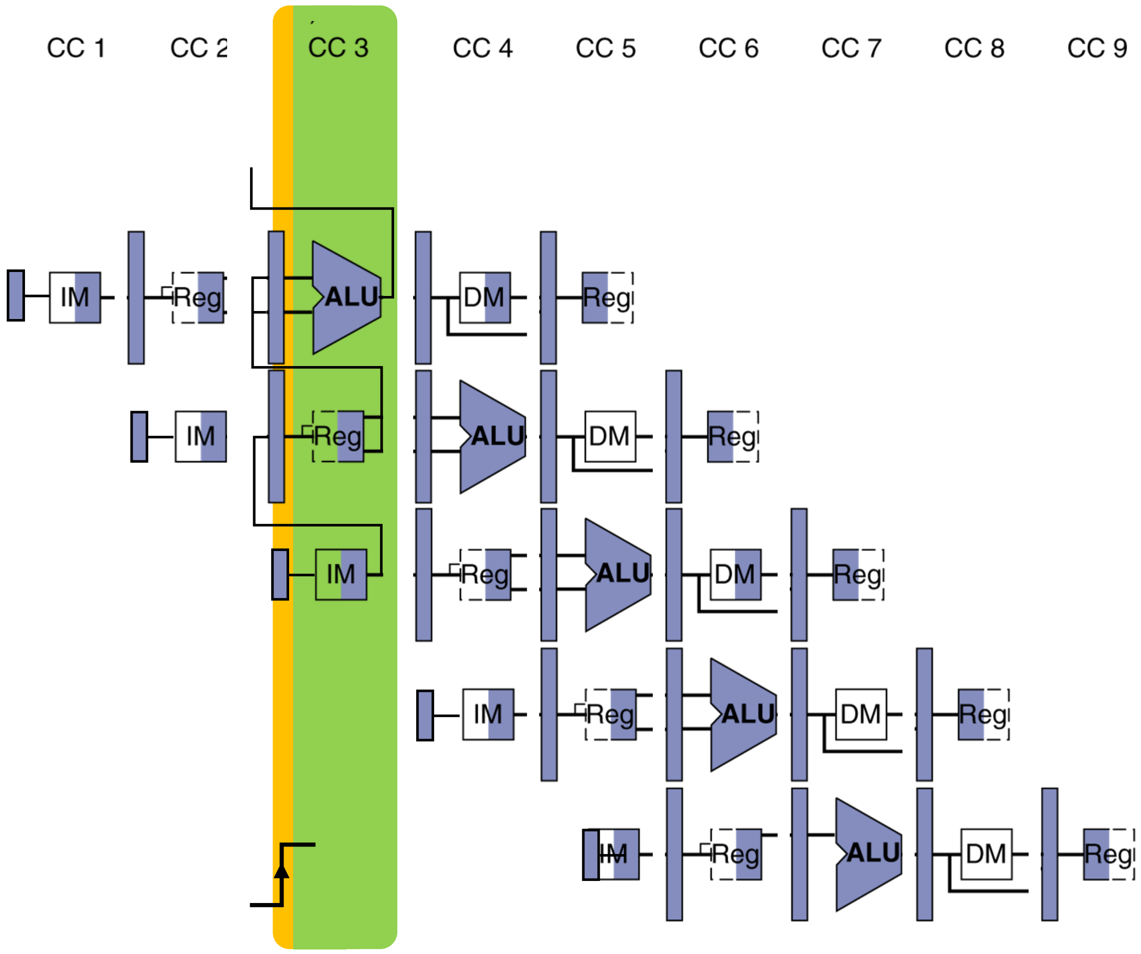
```
lw  t1, 0(gp)

lw  t2, 4(gp)

nop

add t3, t1, t2

sw  t3, 8(gp)
```

Resource Usage Diagram

```
lw   t1, 0(gp)

lw   t2, 4(gp)

nop

add t3, t1, t2

sw   t3, 8(gp)
```


Time (in clock cycles)
CC 1
CC 2
CC 3
CC 4
CC 5
CC 6
CC 7
CC 8
CC 9
Program execution order (in instructions)
Same ALU at different cycles
Same DM at different cycles
3rd-cycle ALU produces result for 4th-cycle DM
IM
Reg
ALU
DM
Reg

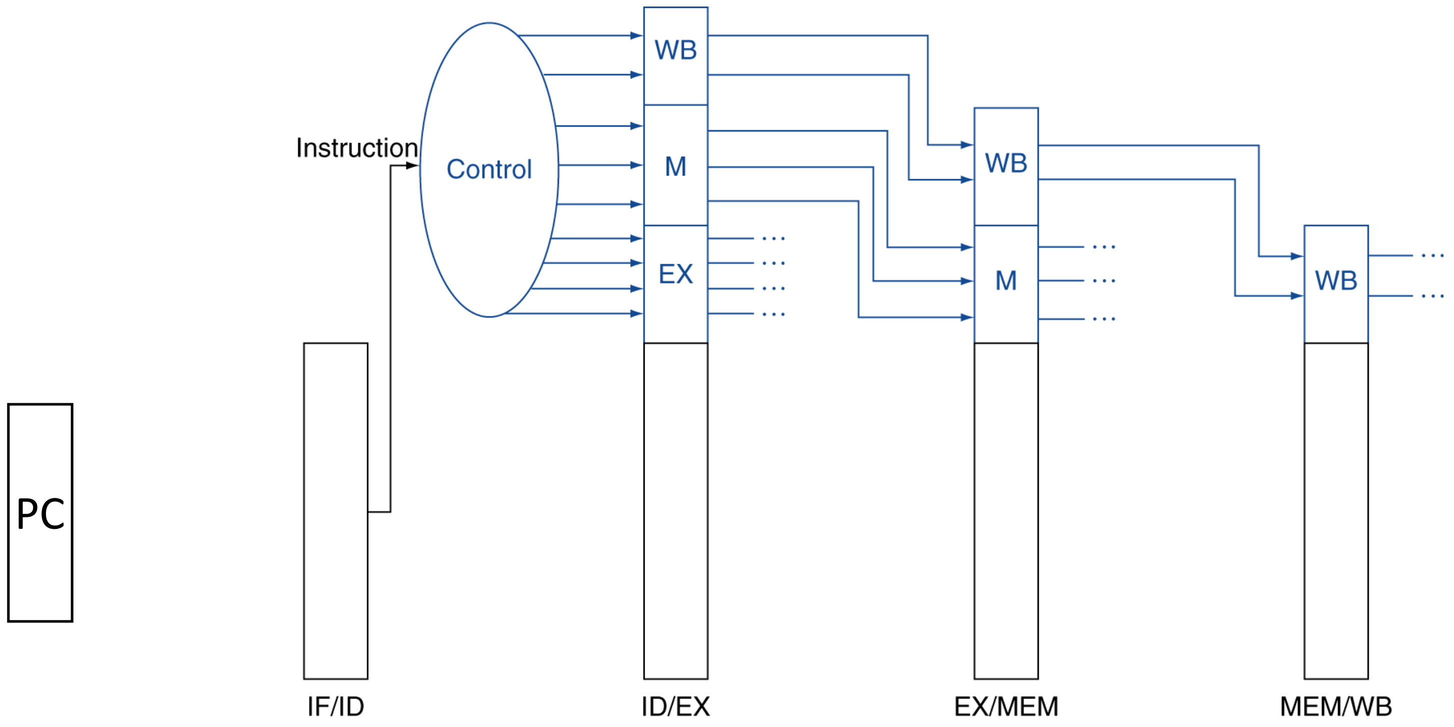
Real Hardware Diagram



```
lw  t1, 0(gp)

lw  t2, 4(gp)

nop

add t3, t1, t2

sw  t3, 8(gp)
```

Pipeline Control

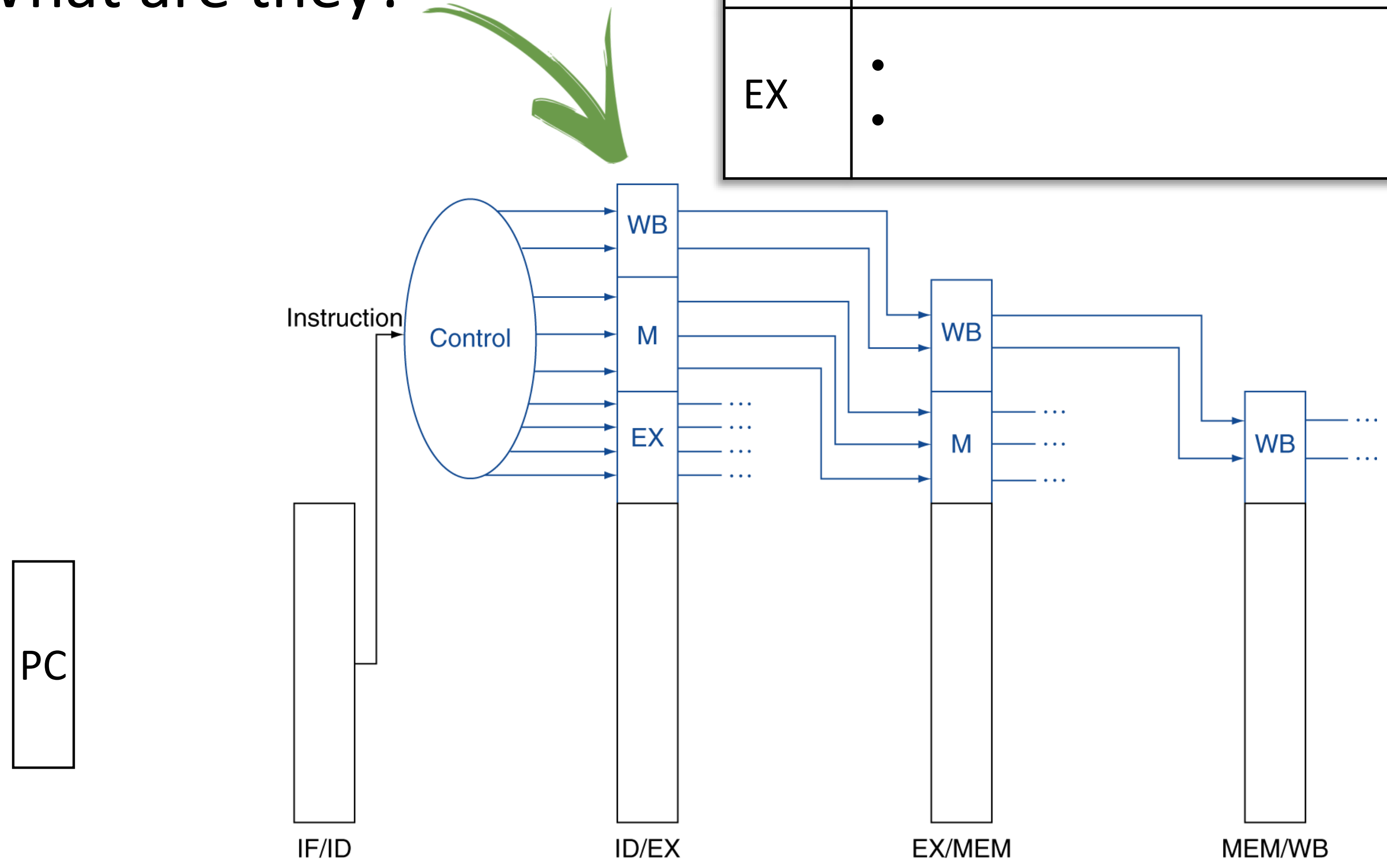
- Control signals are derived from each instruction
  - Just like single-cycle datapath
- Control signals are passed along just like the data



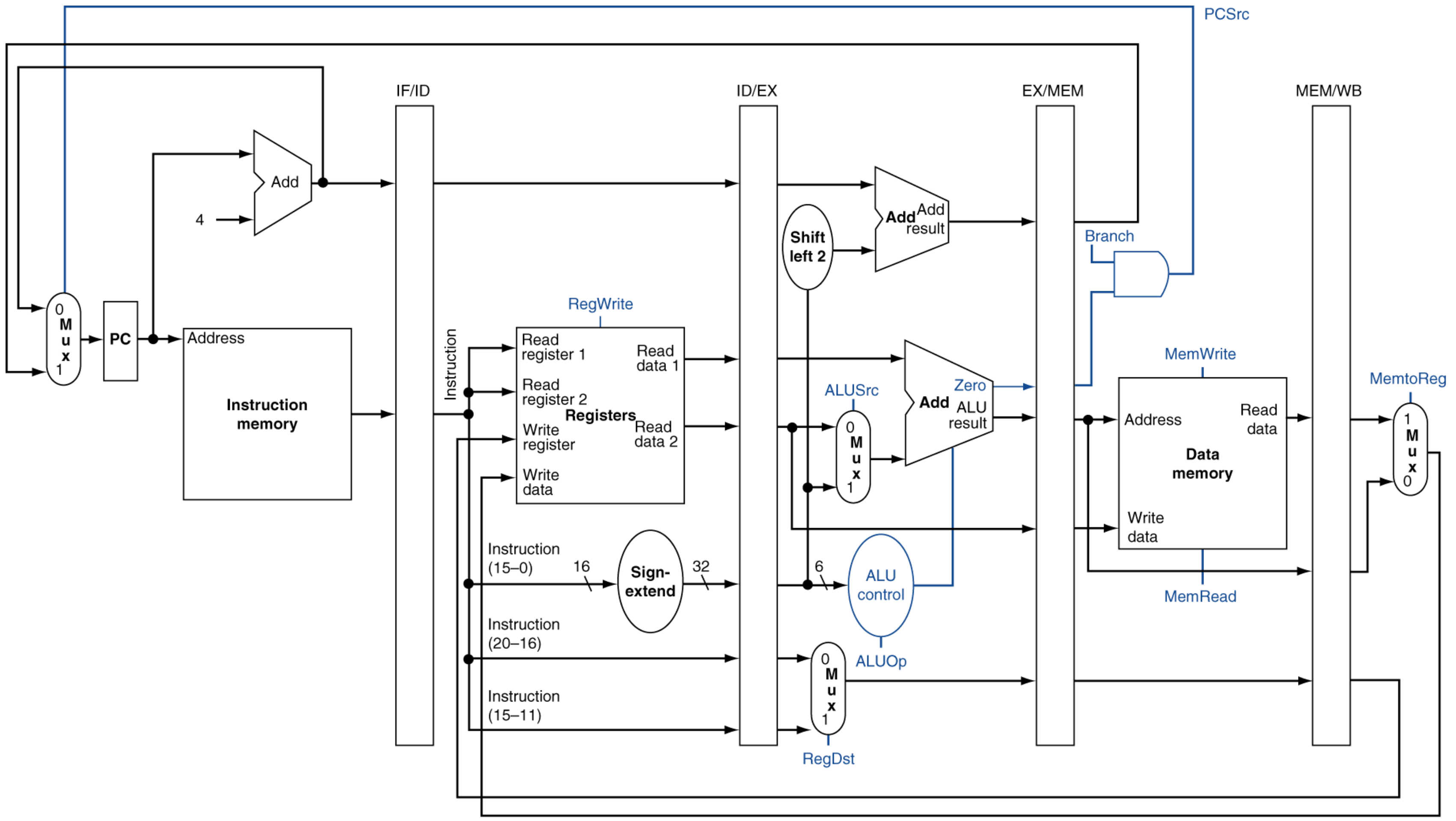
Pipeline Control

What are they?

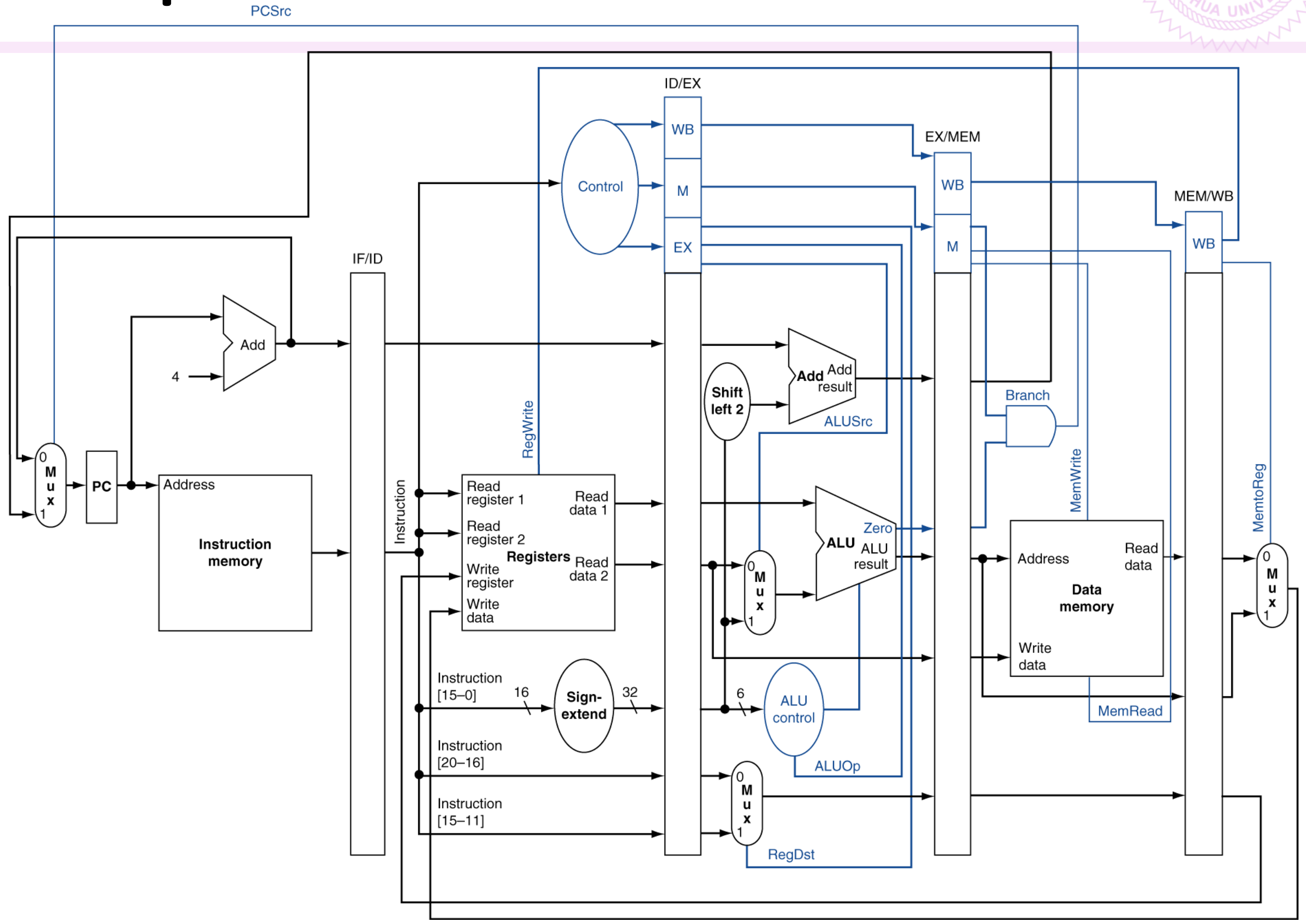
WB	• •
M	• • •
EX	• •



Datapath without Control

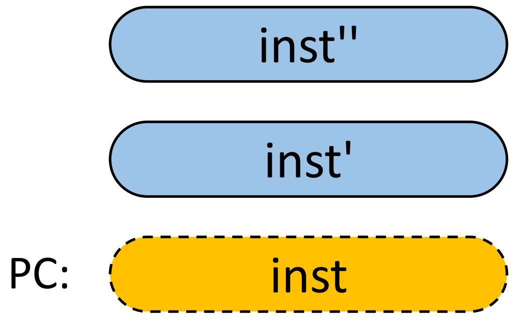


Datapath with Control

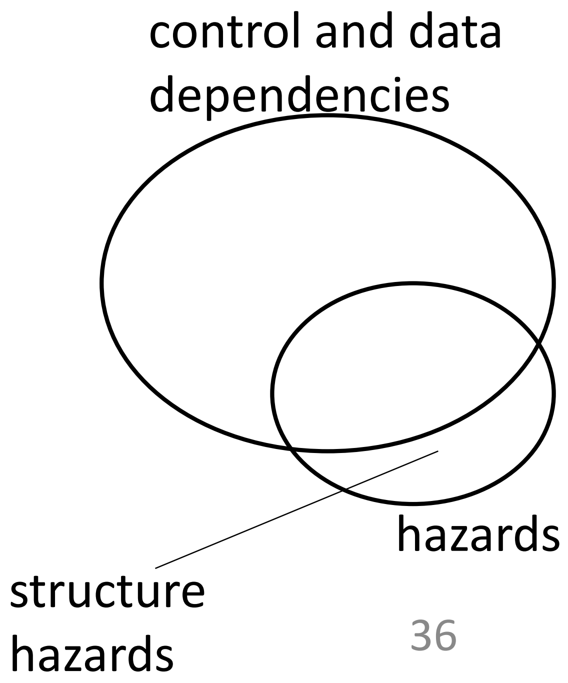


Dependencies and Hazards

- Dependencies
  - Operand of the current instruction depends on a previous instruction's outcome

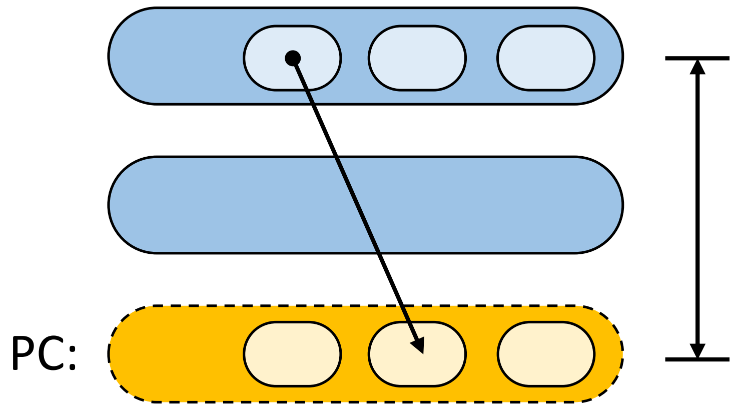


- Hazards
  - Current PC cannot be executed right now
  - Pipeline needs to postpone executing current PC (i.e., **stall**) for some cycles
  - Dependencies may result in hazards
  - Pipeline hardware design can resolve some hazards
  - Compiler can avoid some hazards



Anatomy of Data Dependencies

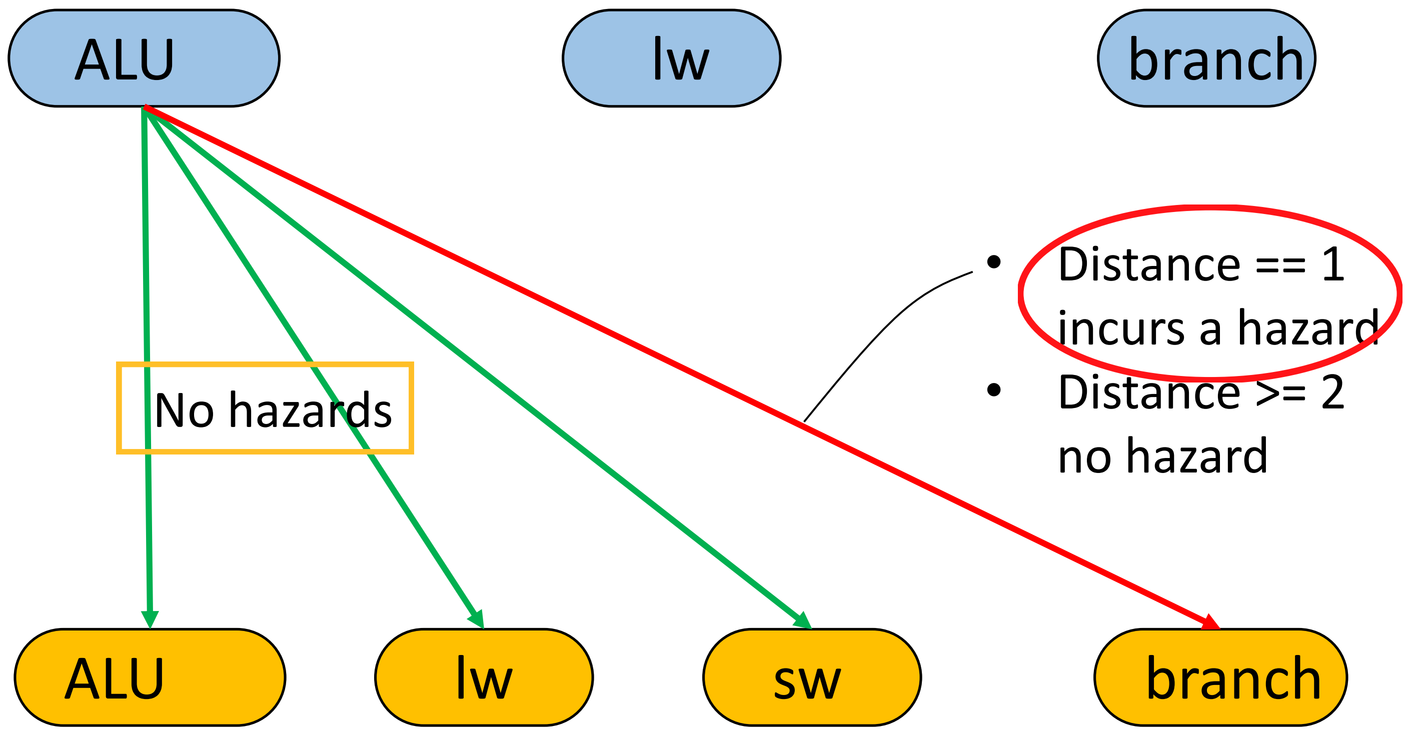
- Source operand, rd or rt



- Source instruction type, e.g., R-type, lw, & branch
- Distance, e.g., 1~5
- Destination instruction type, e.g., R-type, lw, sw, & branch

- Destination operand, rd

Results



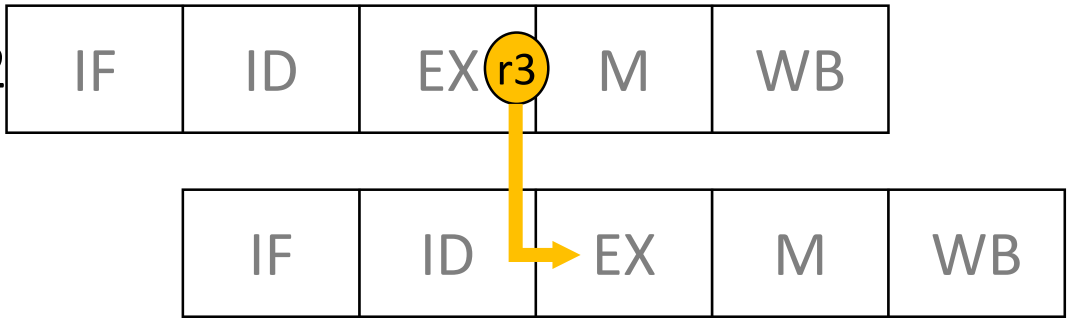
- Distance == 1 incurs a hazard
- Distance >= 2 no hazard

Detailed Analysis

- ALU - ALU

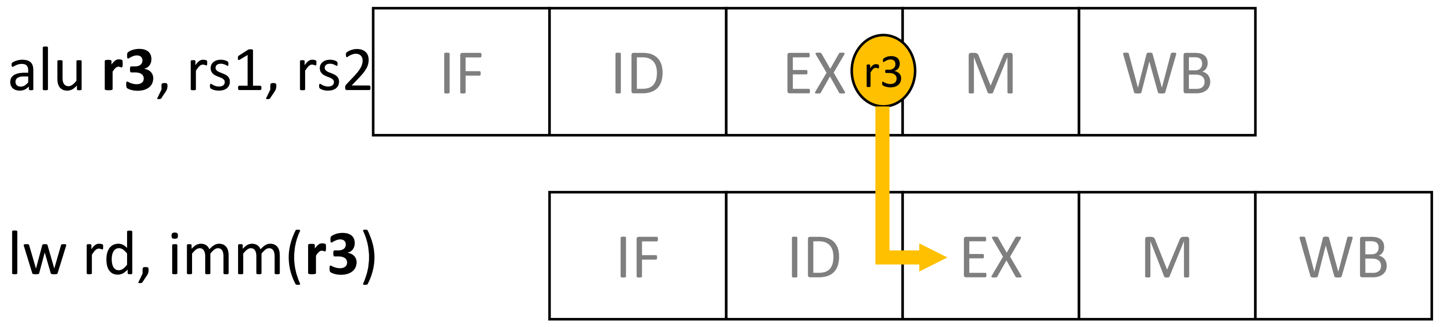
alu **r3**, rs1, rs2

alu rd, **r3**, r4



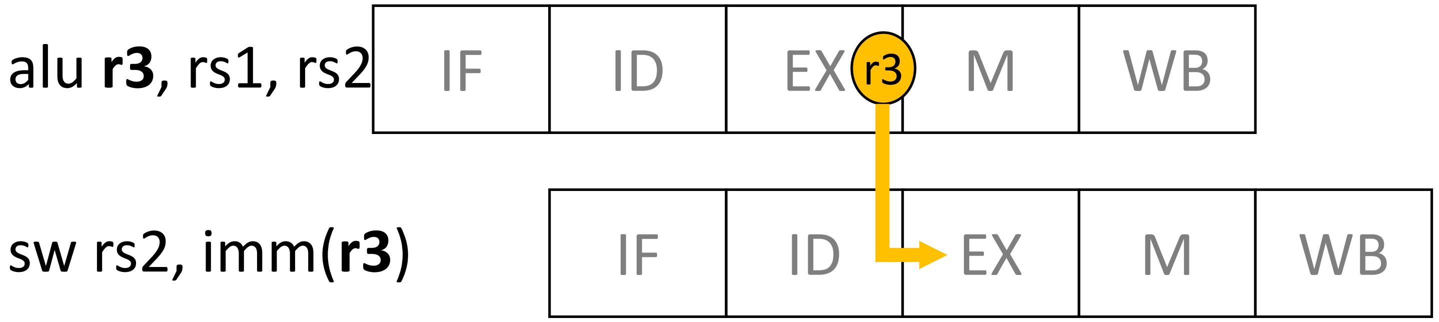
Detailed Analysis

- ALU - LW



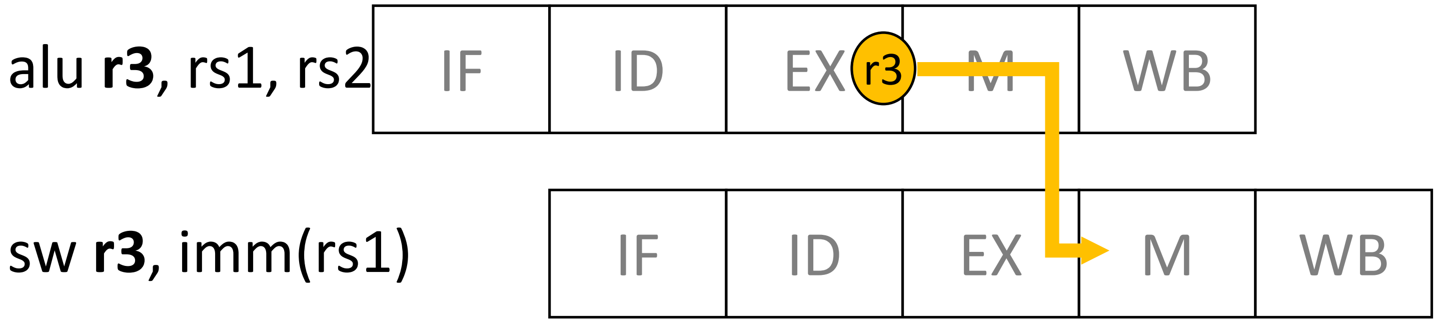
Detailed Analysis

- ALU - SW



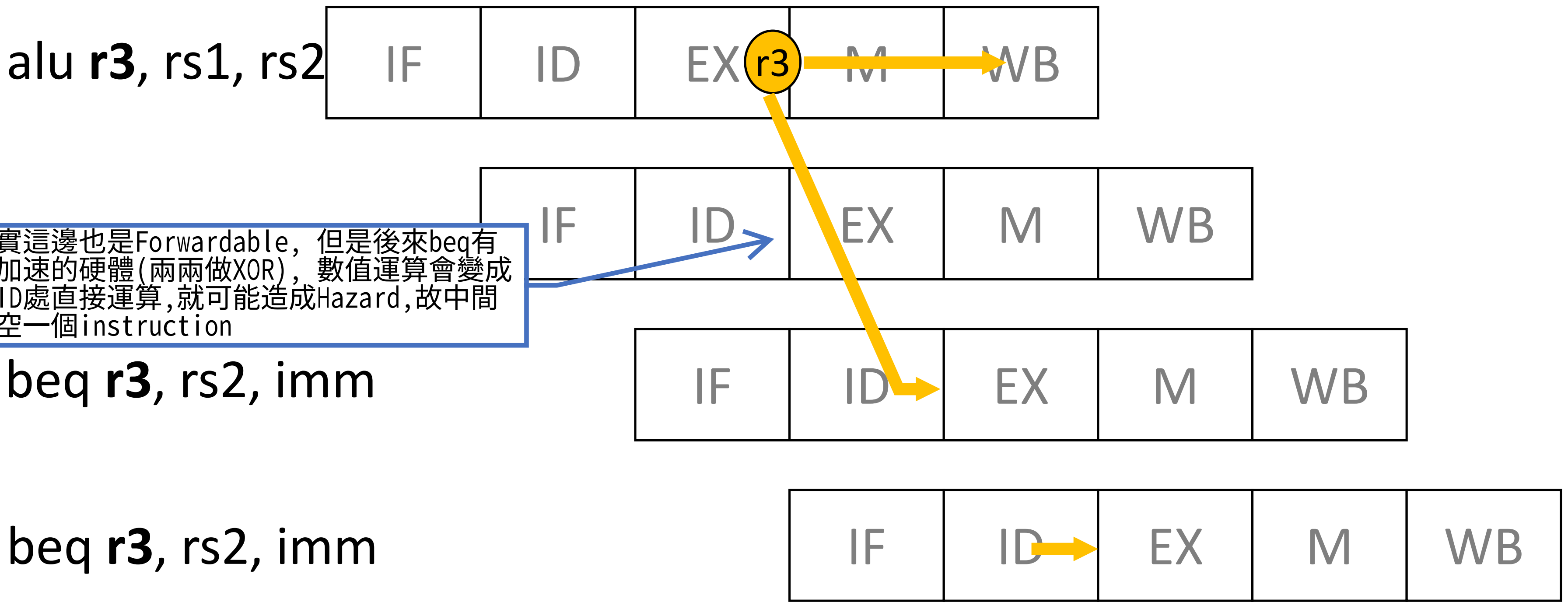
Detailed Analysis

- ALU - SW



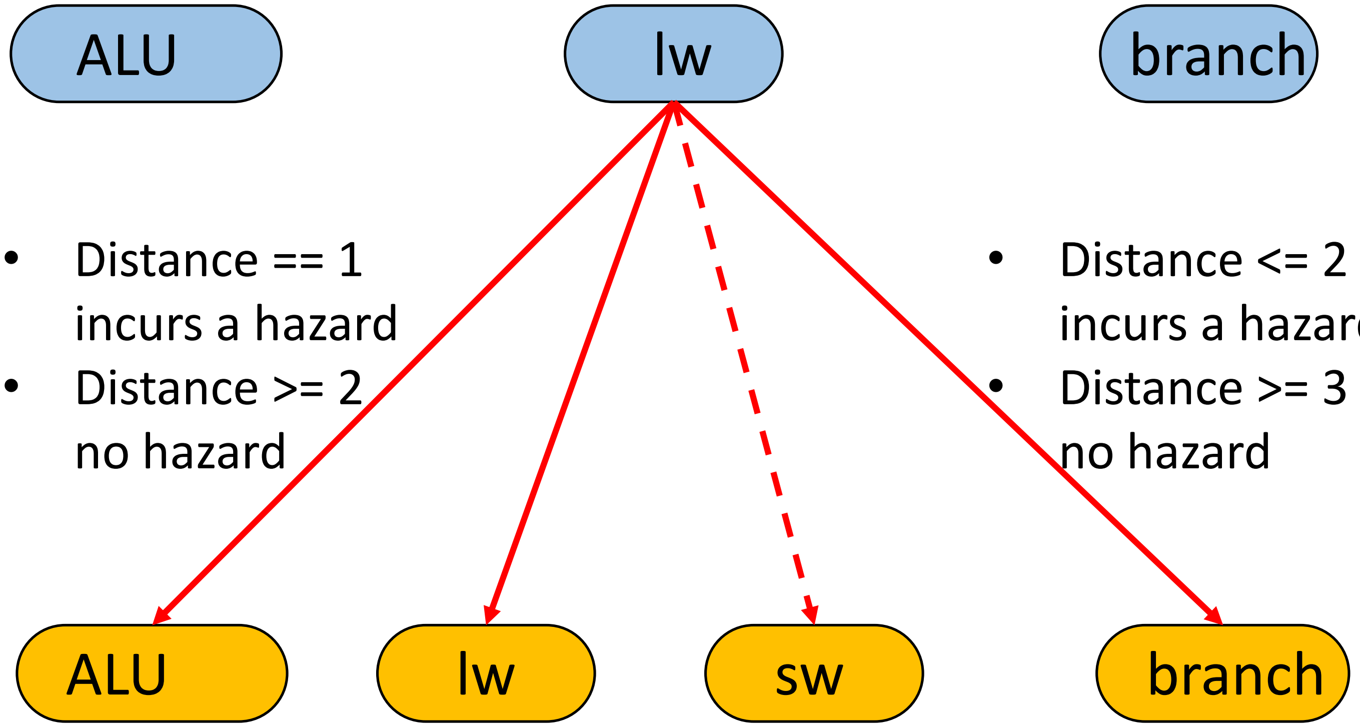
Detailed Analysis

- ALU - Branch



其實這邊也是Forwardable, 但是後來beq有做加速的硬體(兩兩做XOR), 數值運算會變成在ID處直接運算,就可能造成Hazard,故中間要空一個instruction

Results



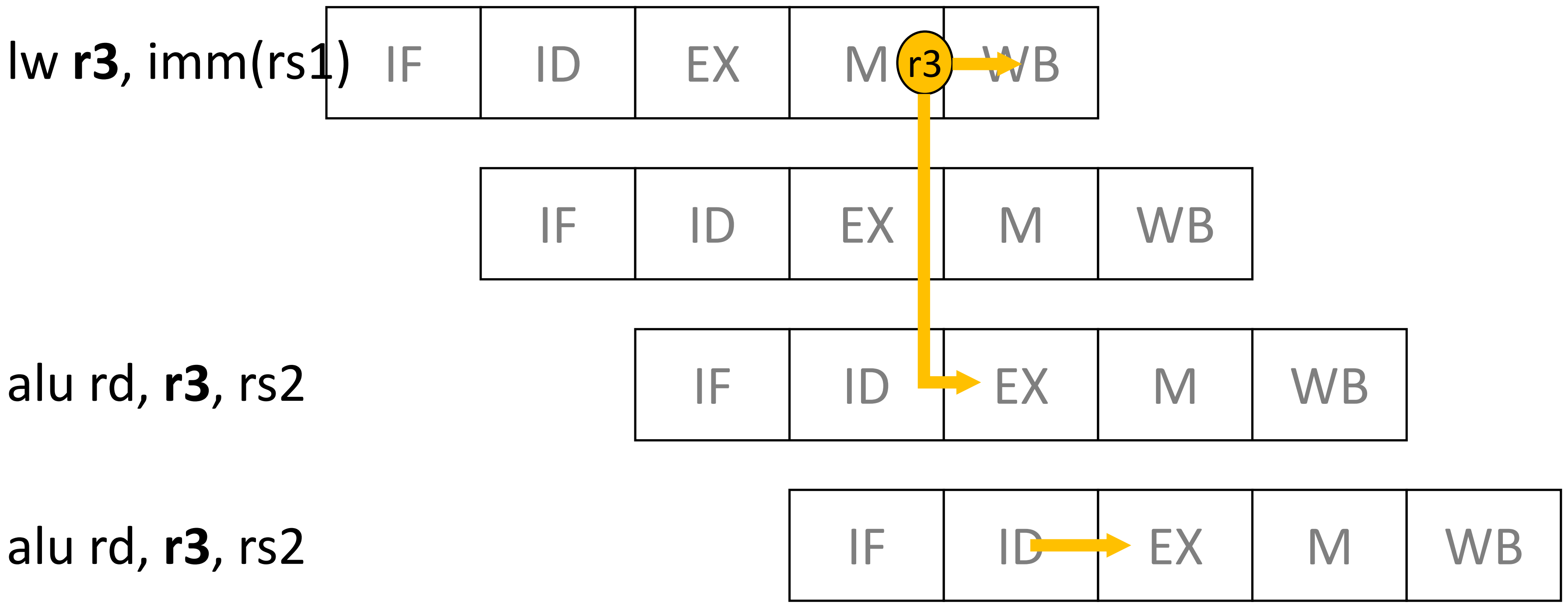
- Distance == 1 incurs a hazard
- Distance >= 2 no hazard

- Distance <= 2 incurs a hazard
- Distance >= 3 no hazard

- rs2 can be forwarded (no hazard)
- rs1 dependency causes no hazard if distance >= 2 (similar to lw-R-type)

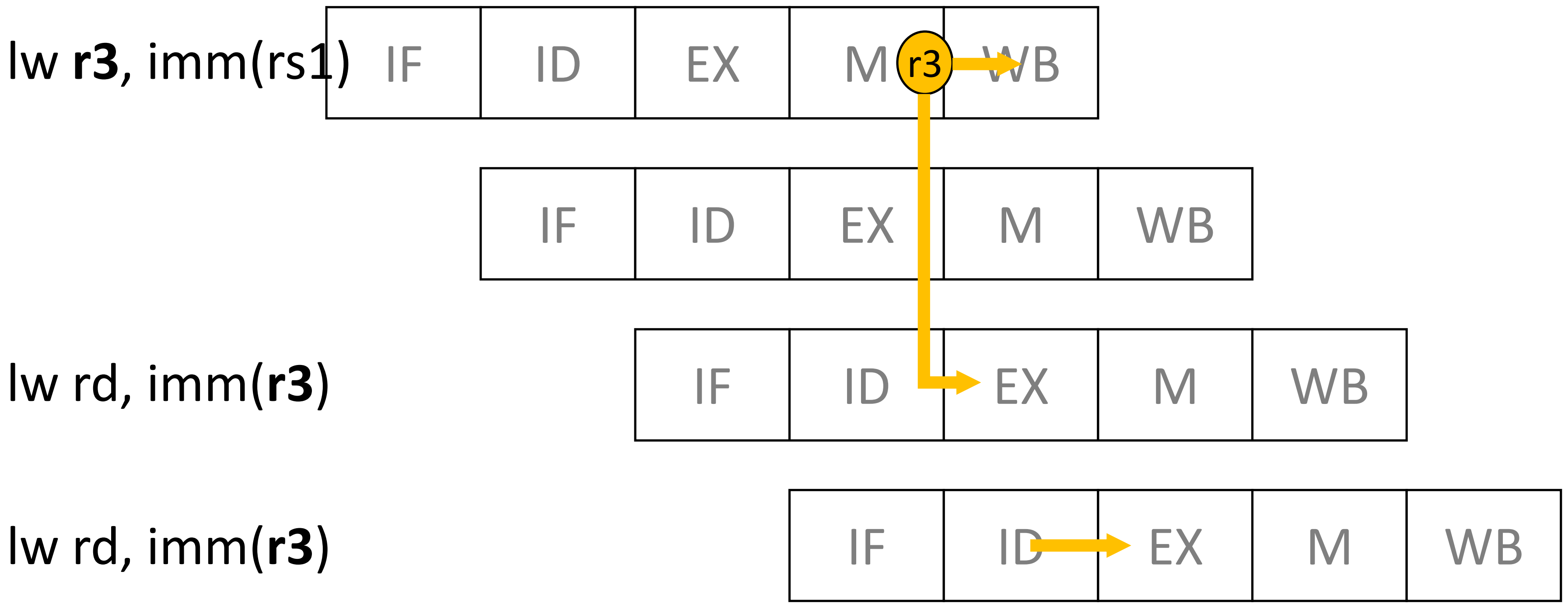
Analysis

- LW - ALU



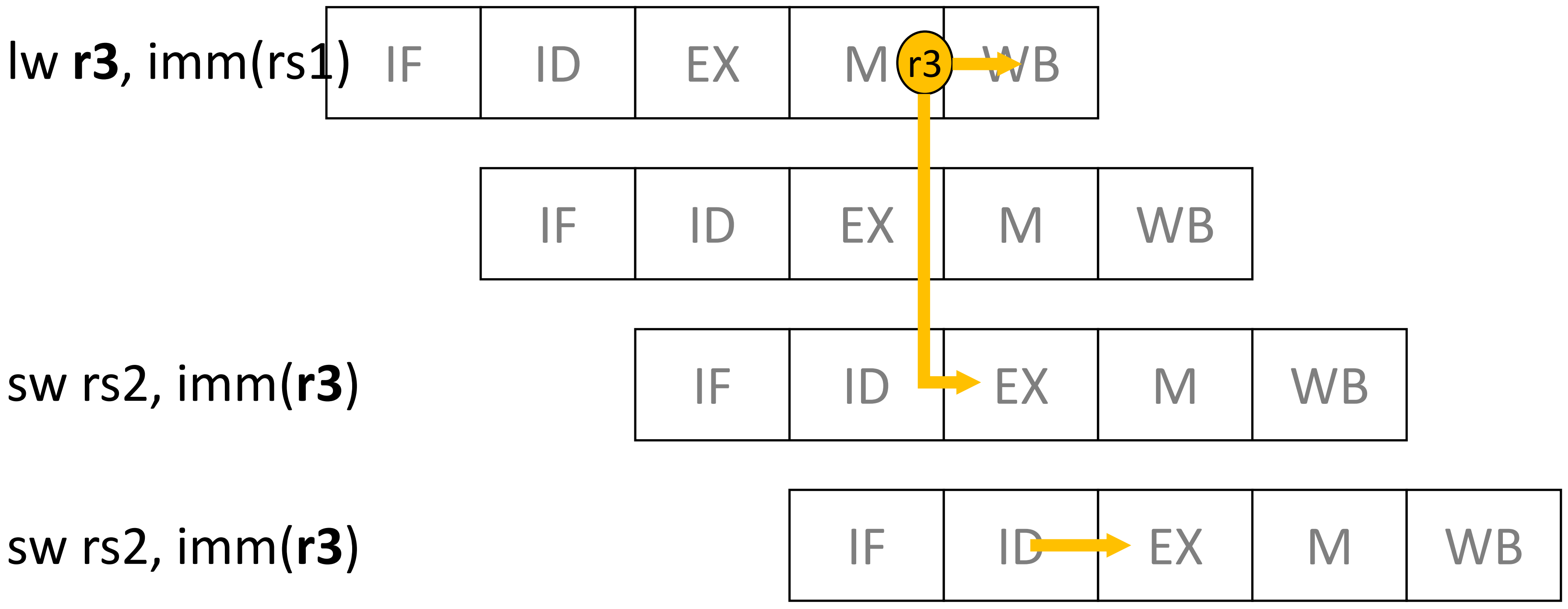
Analysis

- LW - LW



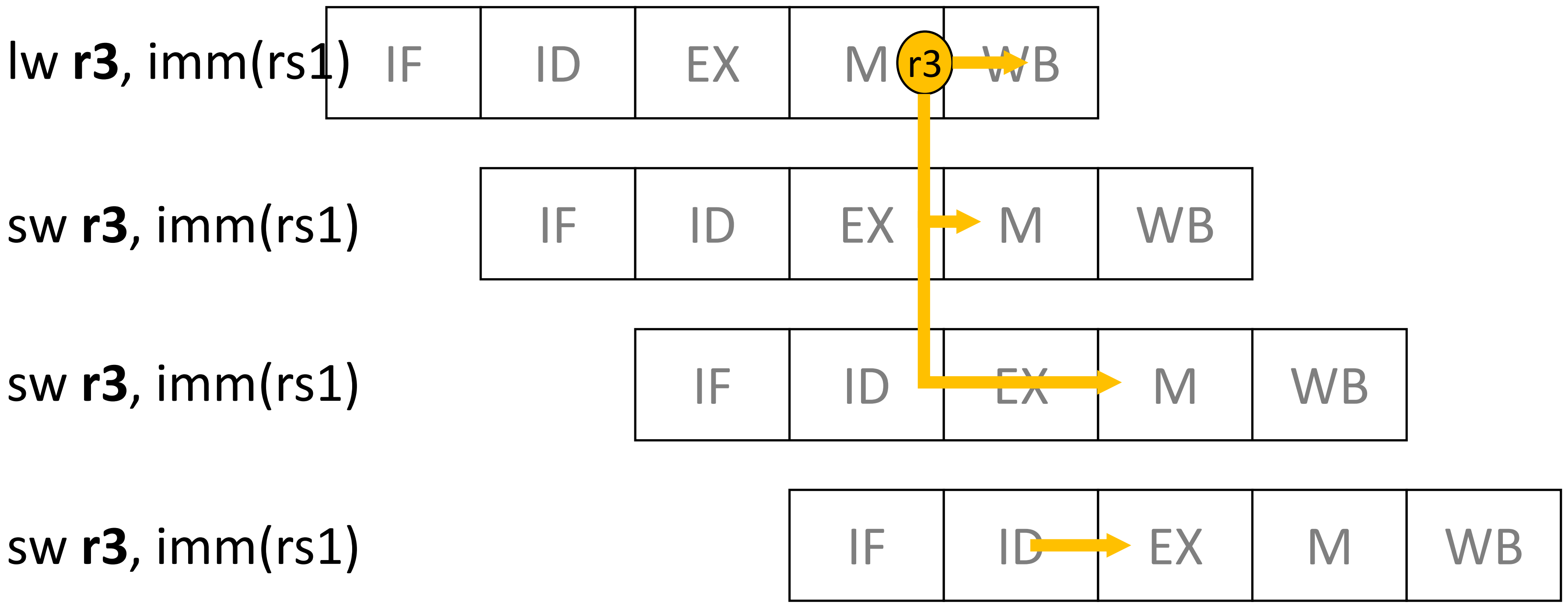
Analysis

- LW - SW



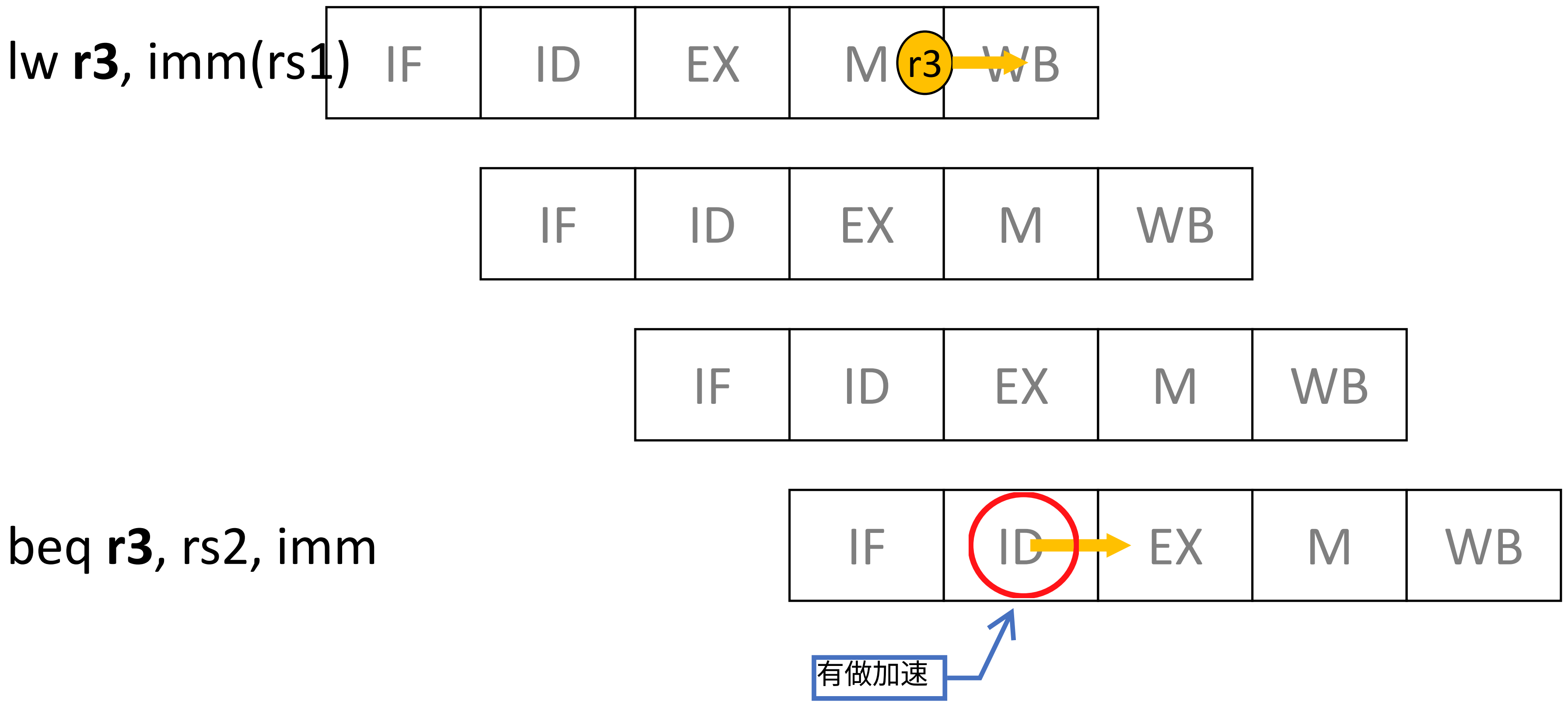
Analysis

- LW - SW

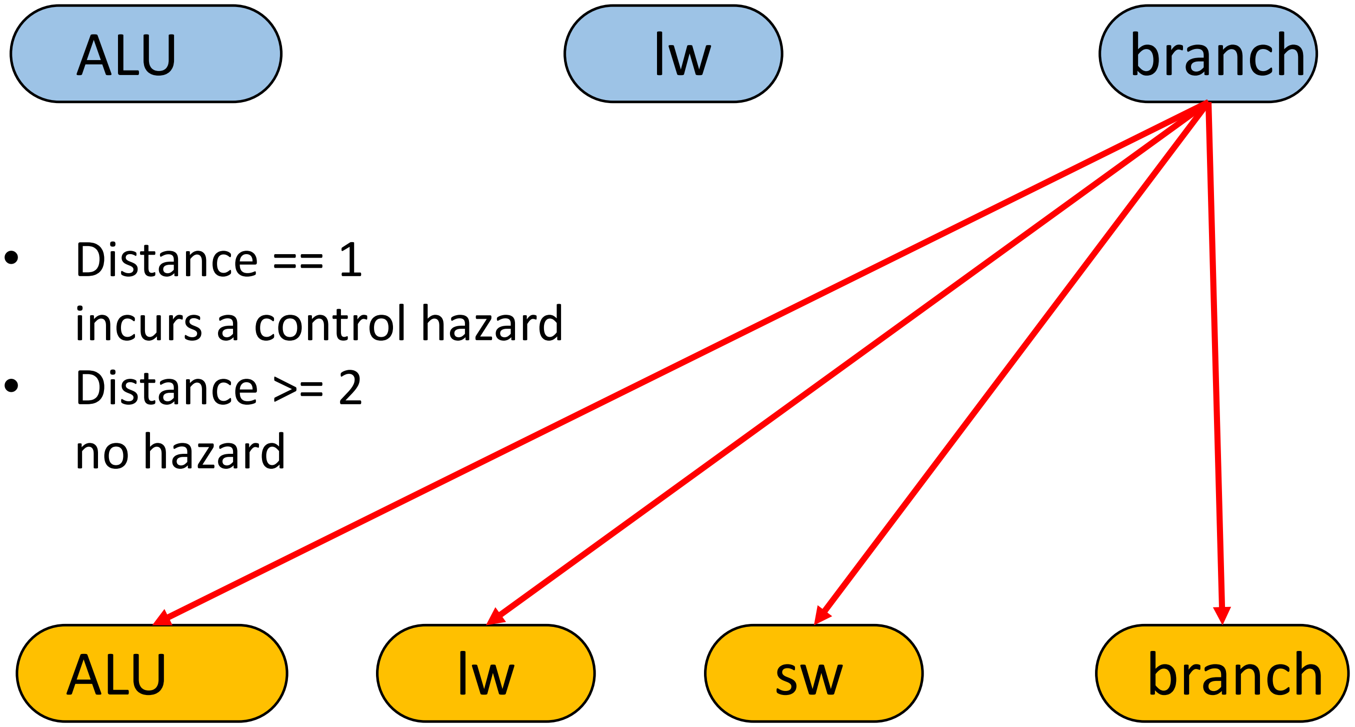


Analysis

- LW - SW

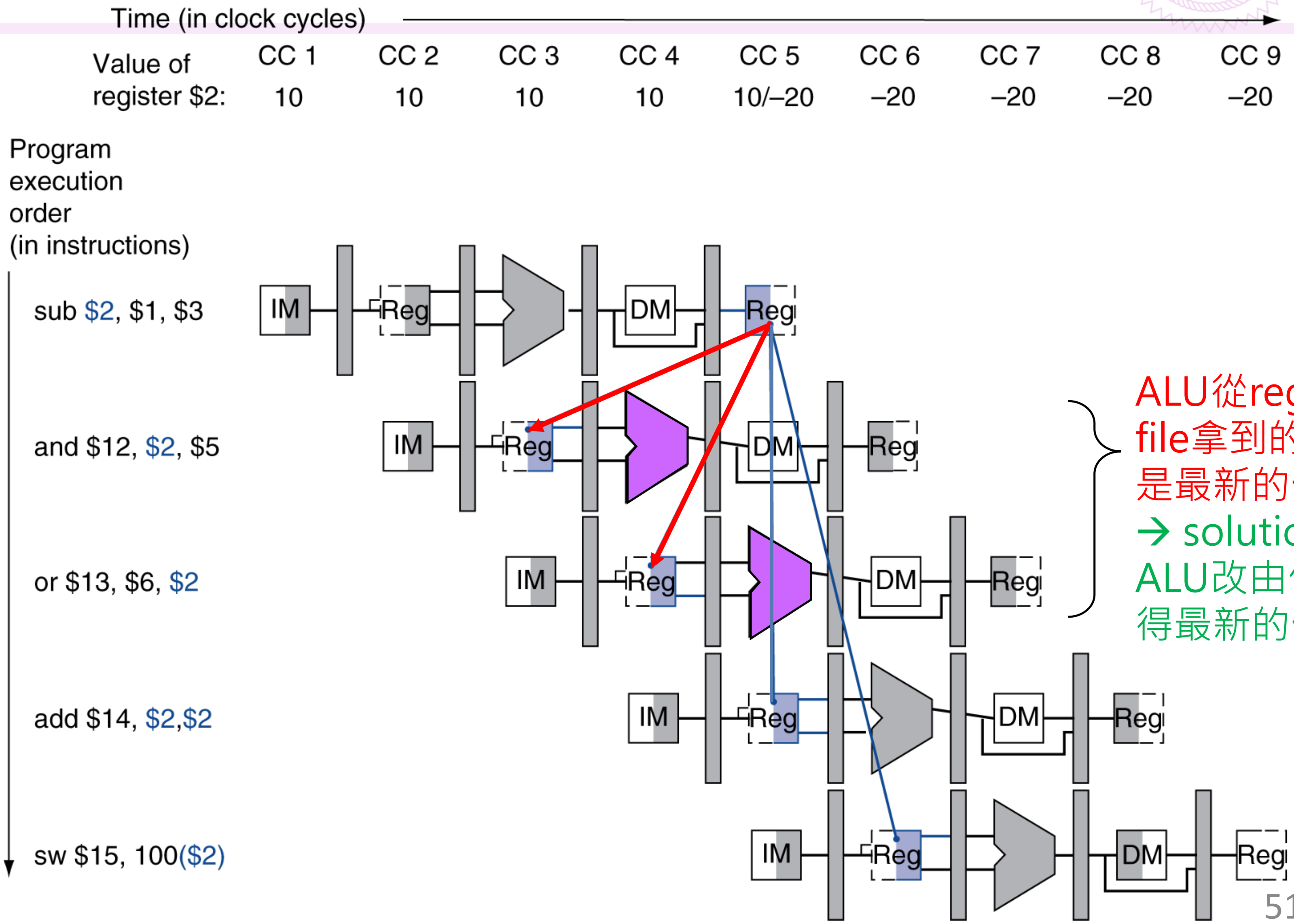


Results



- Distance == 1
  incurs a control hazard
- Distance >= 2
  no hazard

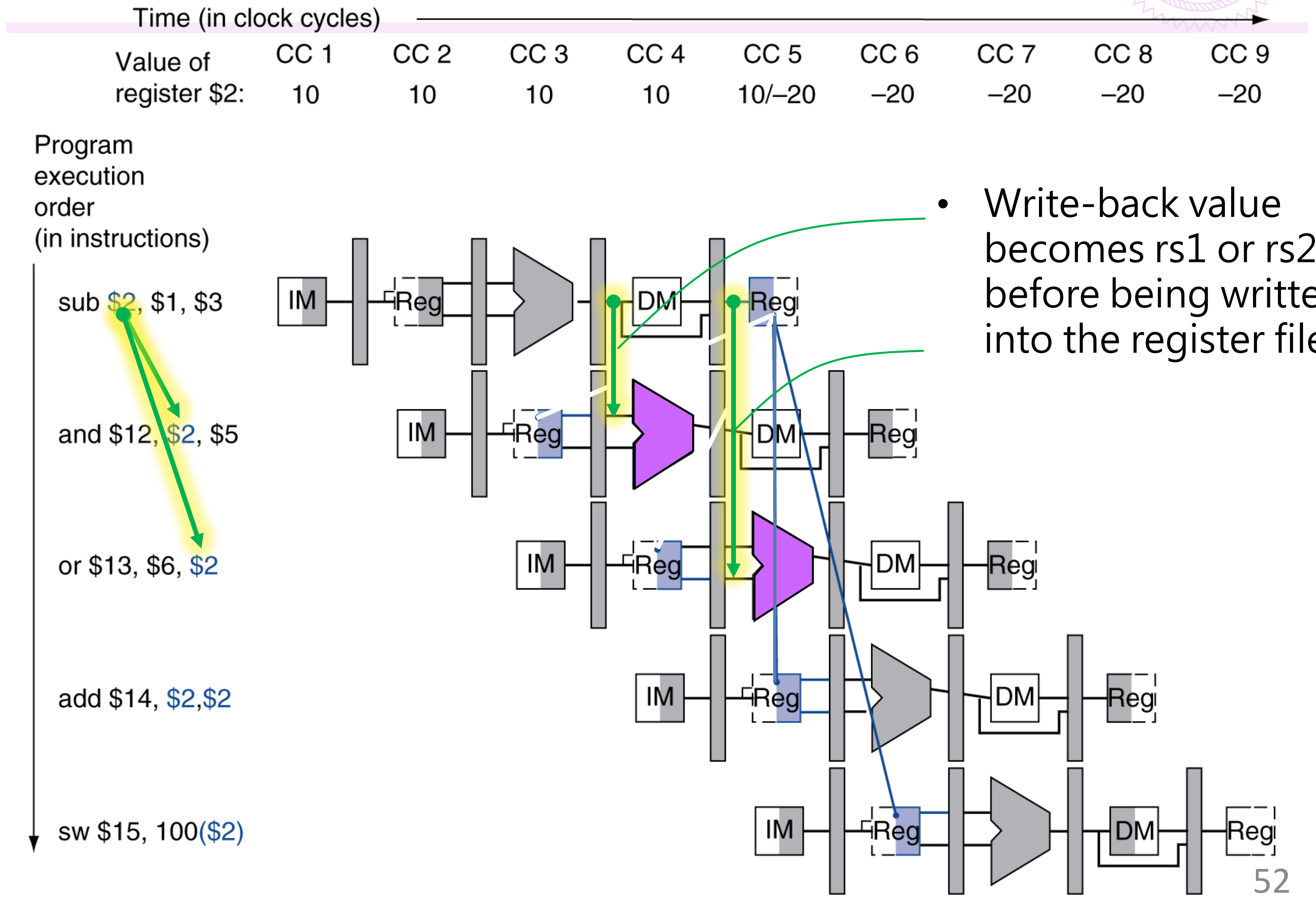
Hazard Example



ALU從register file拿到的，不是最新的值

→ solution: ALU改由他處得最新的值

Forwarding (Bypassing)

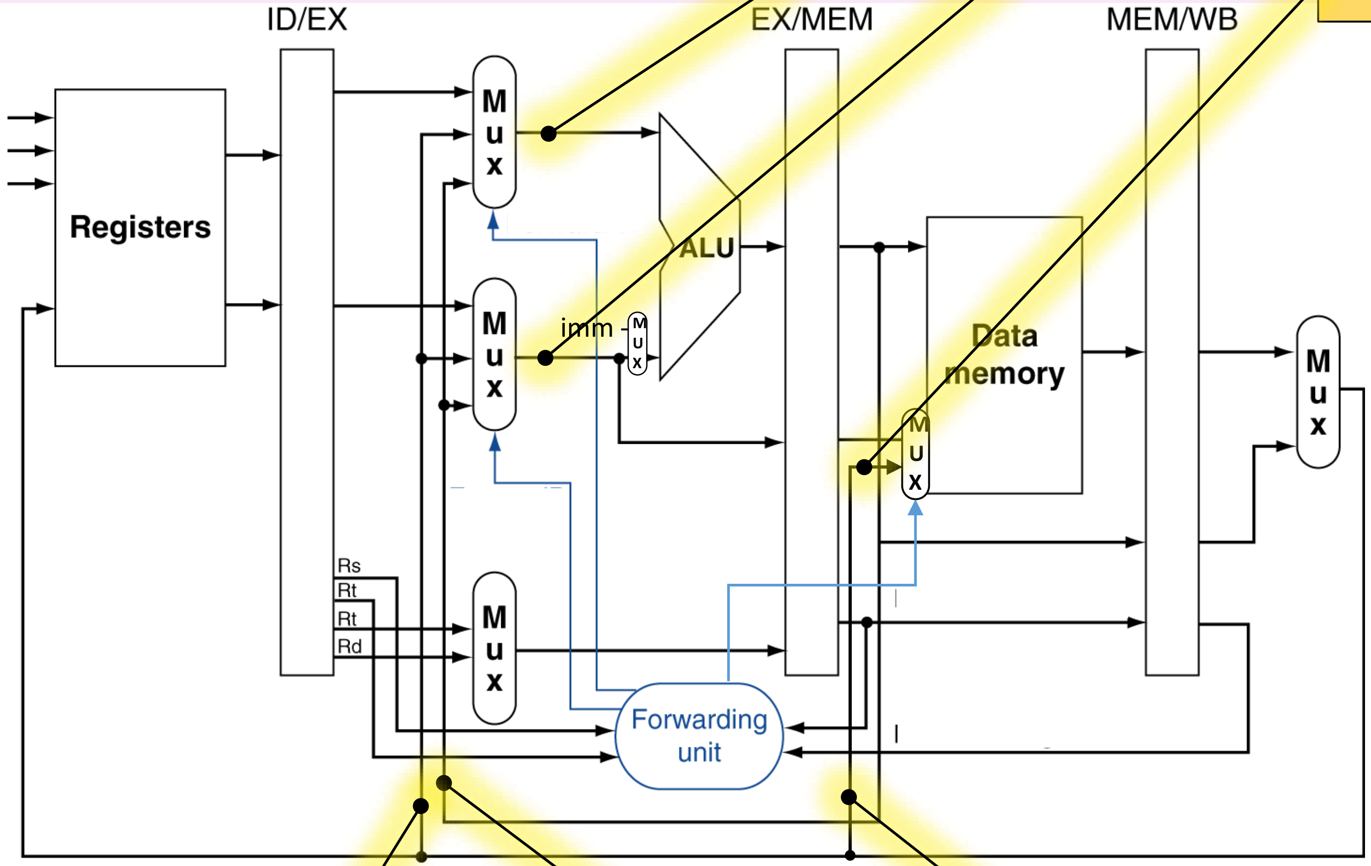


- Write-back value becomes rs1 or rs2 before being written into the register file

Forwarding Paths

- For a 5-stage pipeline
  - Forwarding destinations
    - rs1 of ALU, beq, and SW
    - rs2 of ALU and beq
    - rs2 of SW
  - Forwarding sources
    - rd of the 1-cycle-earlier ALU
    - rd of the 2-cycle-earlier ALU
    - rd of the 1-cycle-earlier LW
    - rd of the 2-cycle-earlier LW

Forwarding Paths



- rs1 of ALU, beq, and SW

- rs2 of R-type and beq

- rs2 of SW

- rd of the 2-cycle-earlier ALU and LW

- rd of the 1-cycle-earlier ALU

- rd of the 1-cycle-earlier ALU and LW

Forwarding Control

- Take special care of
  - Don't forward any result to \$0, which is always zero

```
add $0,$1,$2
add $3,$0,$4
```

  - Forward the latest value for double dependencies

```
add $1,$1,$2
add $1,$1,$3
add $1,$1,$4
```

Forwarding Control

- Fwd x to rs = f_1(inst, inst', inst'')

- Fwd inst'.rd to rs
  - inst ∈ {ALU, ~~beq,~~ lw, sw} and
  - inst' ∈ {ALU} and
  - inst.rs == inst'.rd and
  - inst.rs != 0

- Fwd inst''.rd to rs
  - inst ∈ {ALU, ~~beq,~~ lw, sw} and
  - inst'' ∈ {ALU, lw} and
  - inst.rs == inst''.rd and
  - inst.rs != 0 and
  - Fwd inst'.rd to ALU_in is false // not double dependency

Forwarding Control

- Fwd inst''.rd to inst'.rs2 = f_2(inst, inst', inst'')

- Fwd inst''.rd to inst'.rs2
  - inst' ∈ {sw} and
  - inst'' ∈ {lw} and
  - inst'.rs2 == inst''.rd and
  - inst'.rs2 != 0

Outline

- Background
- Single-cycle design
- Pipelined design
    - Pipeline concepts and MIPS's pipeline
    - Cost and issues of pipelining
- Detailed pipelined datapath and control
    - Trace the pipeline
    - Dependencies, hazards, and forwarding
    - Stalls and exceptions