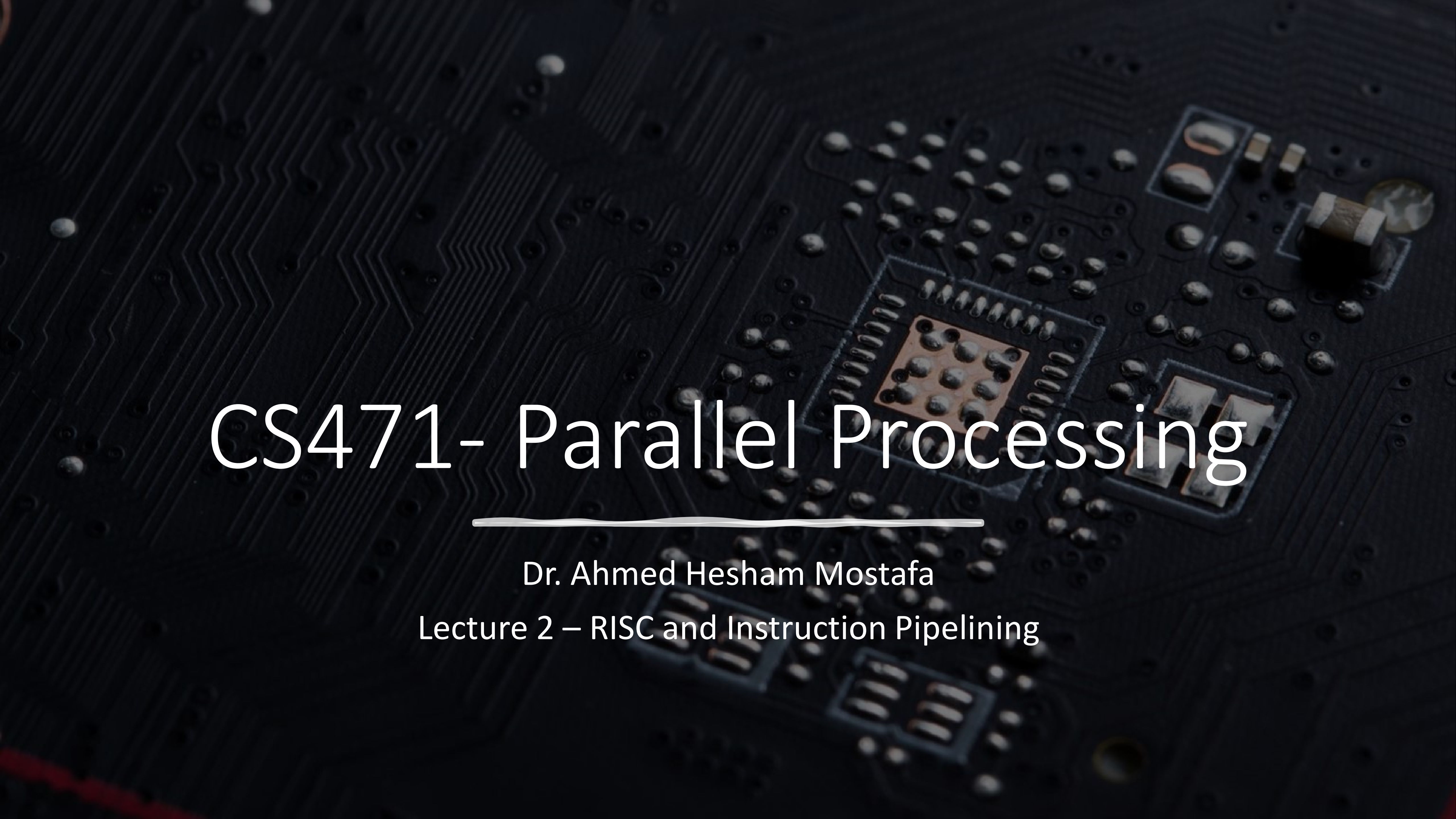


CS471- Parallel Processing

Dr. Ahmed Hesham Mostafa

Lecture 2 – RISC and Instruction Pipelining

Instruction Pipelining

- is a technique for implementing instruction-level parallelism within a single processor. Pipelining attempts to keep every part of the processor busy with some instruction by dividing incoming instructions into a series of sequential steps.
- is a process of arrangement of hardware elements of the CPU such that its overall performance is increased. Simultaneous execution of more than one instruction takes place in a pipelined processor.
- A processor is said to be fully pipelined if it can fetch an instruction on every cycle. Thus, if some instructions or conditions require delays that inhibit fetching new instructions, the processor is not fully pipelined.
- Pipeline is also called Instruction Level Parallelism (ILP).

Pipeline Stages :

- The number of dependent steps varies with the machine architecture. For example:
- The 1956–1961 IBM Stretch project proposed the terms Fetch, Decode, and Execute that have become common.
- The classic RISC pipeline comprises:
    - Instruction fetch
    - Instruction decode and register fetch
    - Execute
    - Memory access
    - Register write back
- The Atmel AVR and the PIC microcontroller each have a two-stage pipeline.

CISC V.S RISC

- **CISC is a complex instruction** , which can perform multiple operations and uses a complex hardware:
- Example MAC R0,R1,R3 which means R0 = (R1 * R3) + R0.
- Example ADD [R0],[R1],[R3] which means add the value present in the address in the R1 to the value present in the address in the R3 and store the result in the address present in the R0

CISC V.S RISC programs

- **RISC is a reduced instruction set** , which can only perform simple instructions and uses simple hardware.
- Example ADD R0,R1,R3 which means R0 = R1 + R3;
- Example LD R0,[R1] which means move R0 register the value present in the address that present in the R1.

RISC: Reduced Instruction Set Computer

Properties:

- All operations on data apply to data in registers and typically change the entire register (32 or 64 bits per reg);
- Only load and store operations affect memory;

  load: move data from mem to reg;

  store: move data from reg to mem;
- Only a few instruction formats; all instructions typically being one size.

RISC: Reduced Instruction Set Computer

- RISC has 32 registers and 3 classes of instructions.
- **The first class is ALU instruction.**
- It usually operates on two registers and stores the result in a third register.
- Typical ALU instructions include add, subtract, and logical operations such as AND, OR.

RISC: Reduced Instruction Set Computer

- **The second class is Load and Store instructions** that affect memory.
- They first get the memory address by adding the base register and an offset. The load instruction uses a second register operand as the data destination while the store instruction uses a second register as the data source.
- Load (LD) and store (SD) instructions

  operands: base register + offset;

  the sum (called *effective address*) is used as a memory address;

RISC: Reduced Instruction Set Computer

- **The third class is branches and jumps** that make conditional transfers of control.
- A branch instruction consists of two phases. The first phase specifies the branch condition to decide whether the branch should take effect.
- It’s just like an if-else in programming language.
- The branch condition can be verified via a set of condition bits or comparison between two registers.
- If the branch condition is satisfied, the second phase decides the branch destination. That is, where to fetch the next instruction.

RISC: Reduced Instruction Set Computer

- at most **5** clock cycles per instruction

IF ID EX MEM WB

- **Instruction Fetch cycle**
- The first cycle is IF for fetching the current instruction from memory.
- To do so, the processor sends the program counter to memory, fetches the current instruction from there and then increments the program counter by 4. (as each mem cell 8 bit and instruction is 32 bit so instruction need 4 mem cells)
- PC = PC + 4; //each instr is 4 bytes

RISC: Reduced Instruction Set Computer

- at most **5** clock cycles per instruction

IF ID EX MEM WB

- **Instruction Decode/register fetch cycle**
- decode the instruction;
- read the registers (corresponding to register source specifiers);

RISC: Reduced Instruction Set Computer

- at most **5** clock cycles per instruction

IF ID EX MEM WB

- **Execution/effective address cycle**
- The third cycle is EX for calculating the effective memory address.
- ALU operates on the operands fetched in the ID cycle.
- There are 3 classes of functions according to the instruction type. **The first class is memory reference**, ALU adds base register and offset to form effective address.

RISC: Reduced Instruction Set Computer

- at most **5** clock cycles per instruction

IF ID EX MEM WB

- **Execution/effective address cycle**
- **The second class is register-register ALU instruction**: in this case, ALU performs the operation specified by opcode on the values in the register file.

RISC: Reduced Instruction Set Computer

- at most **5** clock cycles per instruction

IF ID EX MEM WB

- **Execution/effective address cycle**
- **The third class is register-immediate ALU instruction**. For this type, ALU operates on the first value read from the register file and the sign-extended immediate.

RISC: Reduced Instruction Set Computer

- at most **5** clock cycles per instruction

IF ID EX MEM WB

- **MEMory access**
- The fourth cycle is MEM for memory access. there are two types of instructions affecting memory.
- For load instruction, the memory does a read using the effective address.
- For store instruction, the memory writes some data using the effective address.

RISC: Reduced Instruction Set Computer

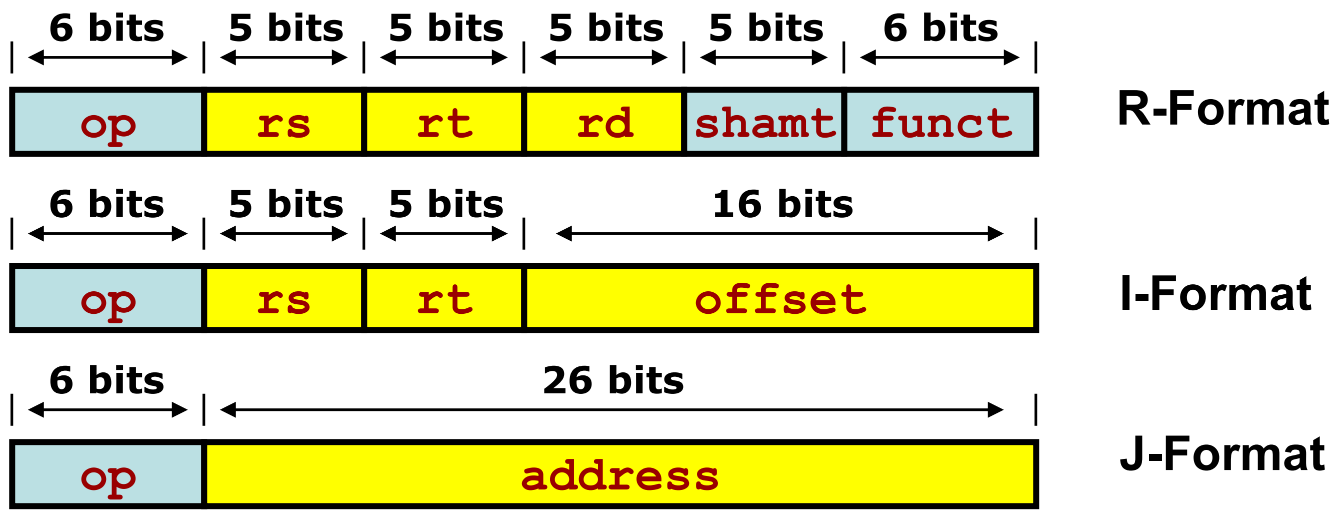
- at most **5** clock cycles per instruction

IF ID EX MEM WB

- **Write-Back cycle**
- The fifth cycle is WB for writing the result into the register file.
- The result may come from the memory if it’s a load instruction or from the ALU if it’s an ALU instruction.

RISC Instruction Format

- Instructions are divided into three types: R (register), I (immediate), and J (jump).
- Every instruction starts with a 6-bit opcode.
- In addition to the opcode, R-type instructions specify three registers,a shift amount field, and a function field;
- I-type instructions specify two registers and a 16-bit immediate value; J-type instructions follow the opcode with a 26-bit jump target.



Set of Instructions

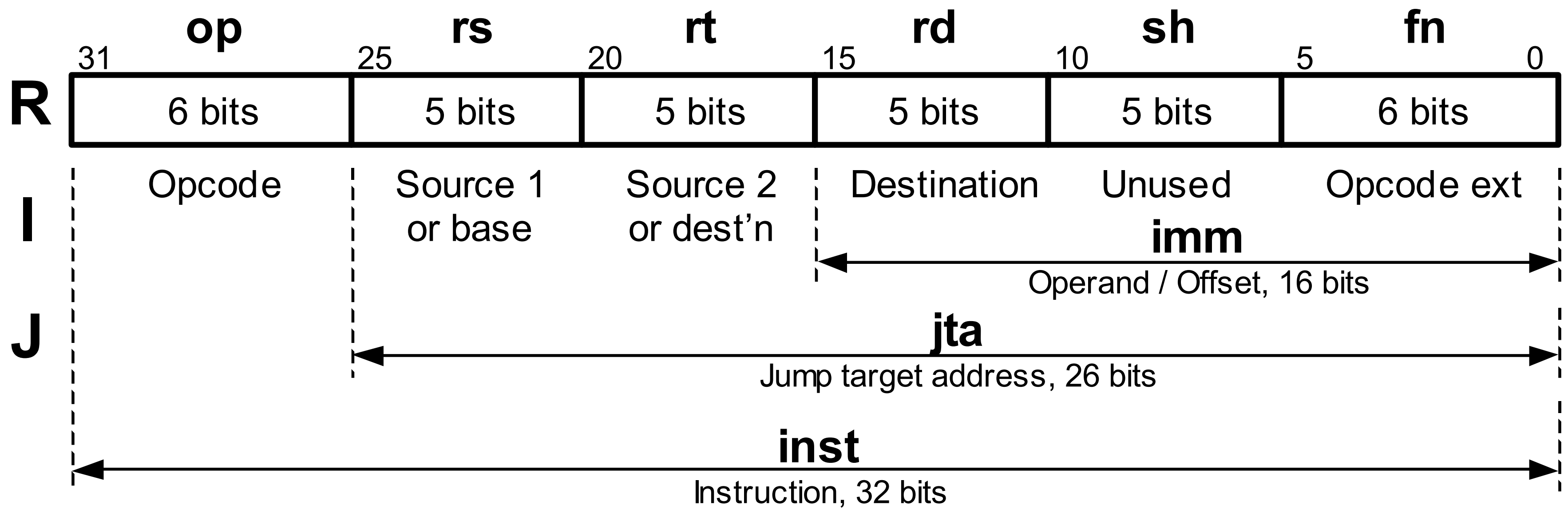


Fig. 13.1 MicroMIPS instruction formats and naming of the various fields.

We will refer to this diagram later

Seven R-format ALU instructions (`add`, `sub`, `slt`, `and`, `or`, `xor`, `nor`)
Six I-format ALU instructions (`lui`, `addi`, `slti`, `andi`, `ori`, `xori`)
Two I-format memory access instructions (`lw`, `sw`)
Three I-format conditional branch instructions (`bltz`, `beq`, `bne`)
Four unconditional jump instructions (`j`, `jr`, `jal`, `syscall`)

The MicroMIPS Instruction Set

Including `la rt,imm(rs)` (load address) makes it easier to write useful programs

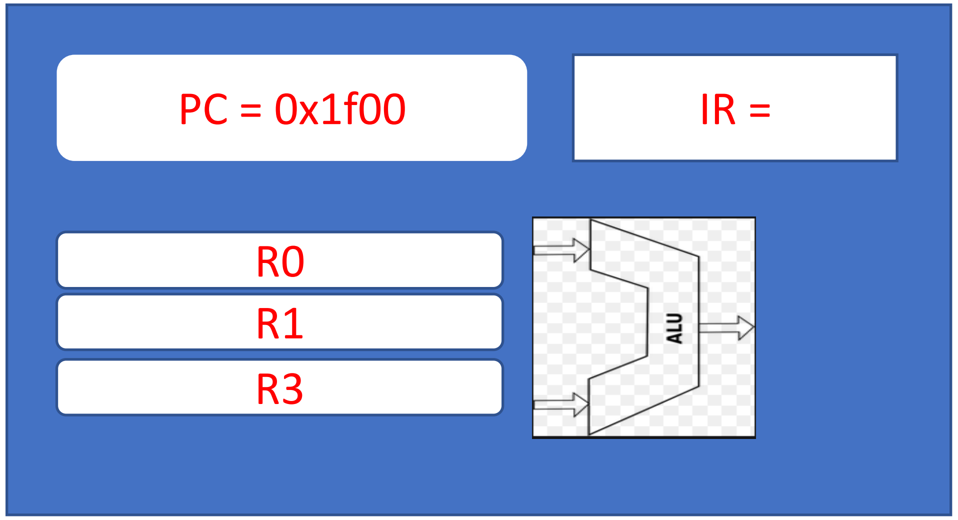
Table 13.1

	Instruction	Usage	op	fn
Copy	Load upper immediate	`lui rt,imm`	15	
Arithmetic	Add	`add rd,rs,rt`	0	32
	Subtract	`sub rd,rs,rt`	0	34
	Set less than	`slt rd,rs,rt`	0	42
	Add immediate	`addi rt,rs,imm`	8	
	Set less than immediate	`slti rd,rs,imm`	10	
Logic	AND	`and rd,rs,rt`	0	36
	OR	`or rd,rs,rt`	0	37
	XOR	`xor rd,rs,rt`	0	38
	NOR	`nor rd,rs,rt`	0	39
	AND immediate	`andi rt,rs,imm`	12	
	OR immediate	`ori rt,rs,imm`	13	
	XOR immediate	`xori rt,rs,imm`	14	
Memory access	Load word	`lw rt,imm(rs)`	35	
	Store word	`sw rt,imm(rs)`	43	
Control transfer	Jump	`j L`	2	
	Jump register	`jr rs`	0	8
	Branch less than 0	`bltz rs,L`	1	
	Branch equal	`beq rs,rt,L`	4	
	Branch not equal	`bne rs,rt,L`	5	
	Jump and link	`jal L`	3	
	System call	`syscall`	0	12

A 5-Stage Pipeline (ADD instruction)

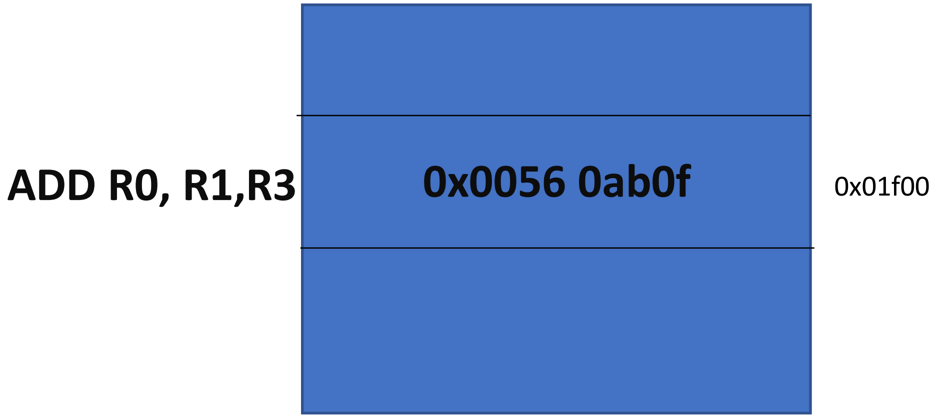
ADD R0, R1,R3

IF	ID	Execute	Memory	Store

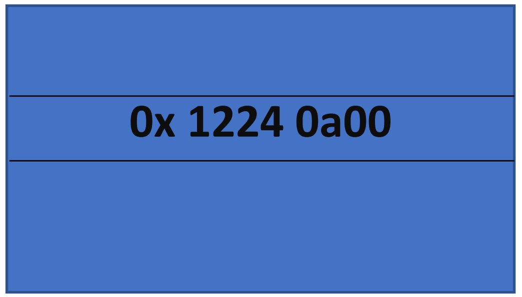


CPU

Program Memory

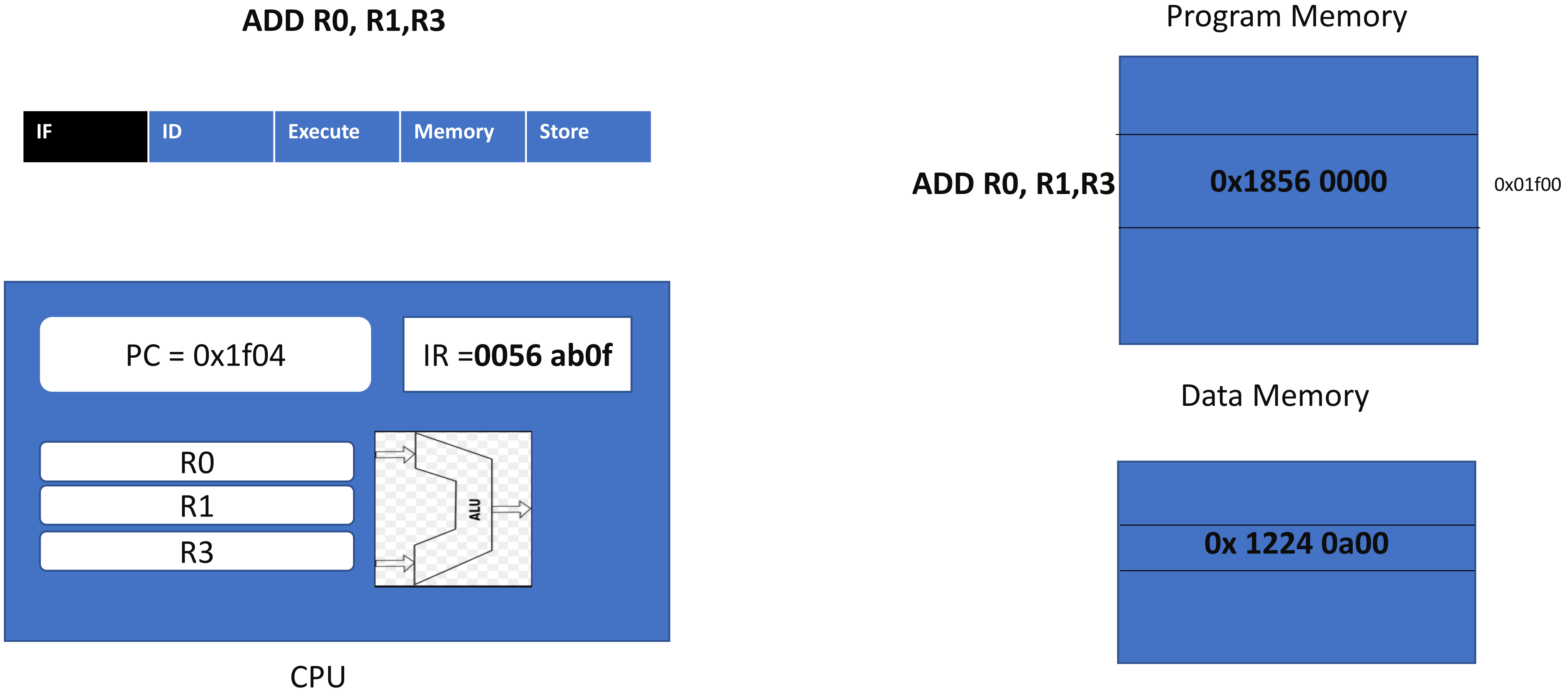


Data Memory



A 5-Stage Pipeline (ADD instruction)

ADD R0, R1,R3



A 5-Stage Pipeline (ADD instruction)

ADD R0, R1,R3

IF	ID	Execute	Memory	Store

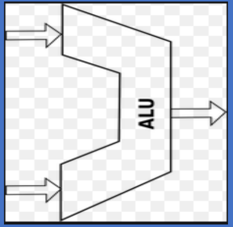
PC = 0x1f04

IR =0056 ab0f

R0

R1 to ALU latch

R3 to ALU latch



CPU

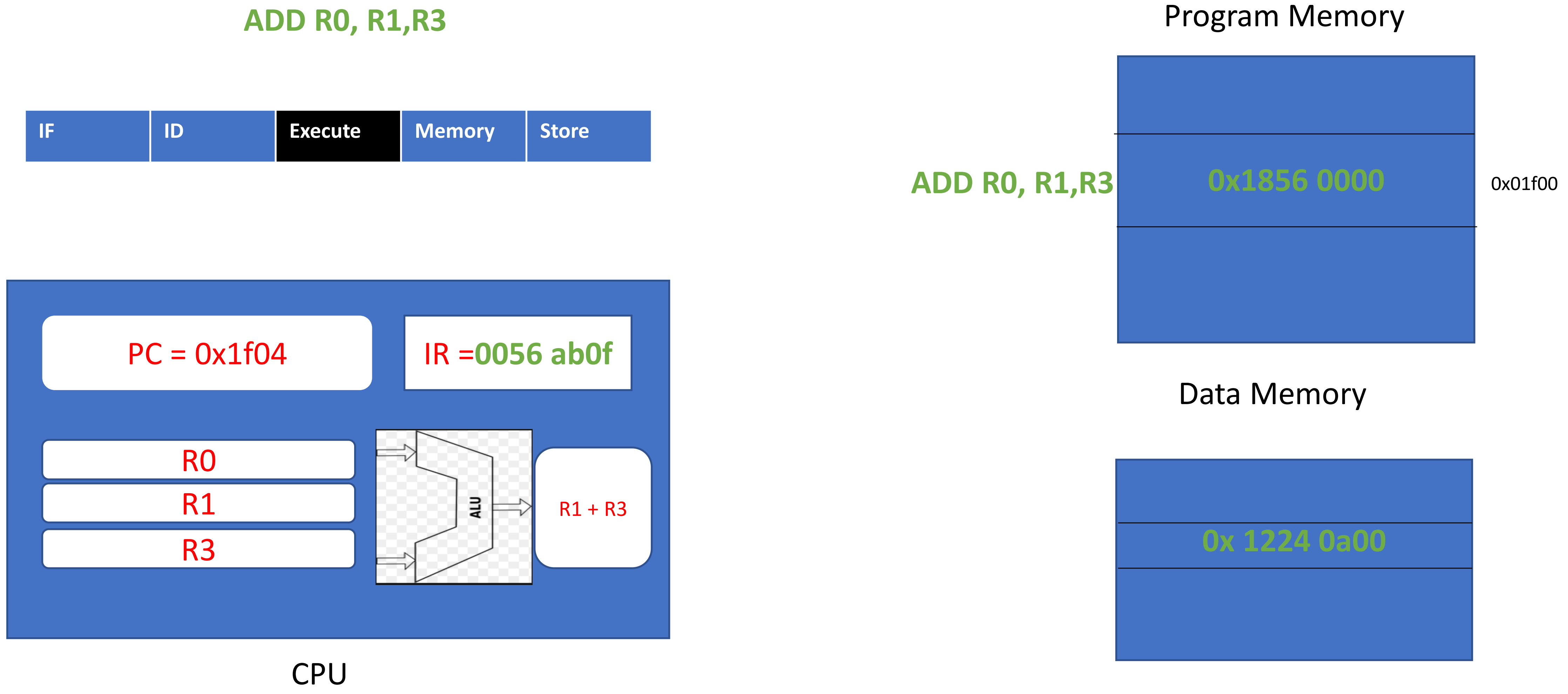
Program Memory

ADD R0, R1,R3 | 0x1856 0000 | 0x01f00

Data Memory

0x 1224 0a00

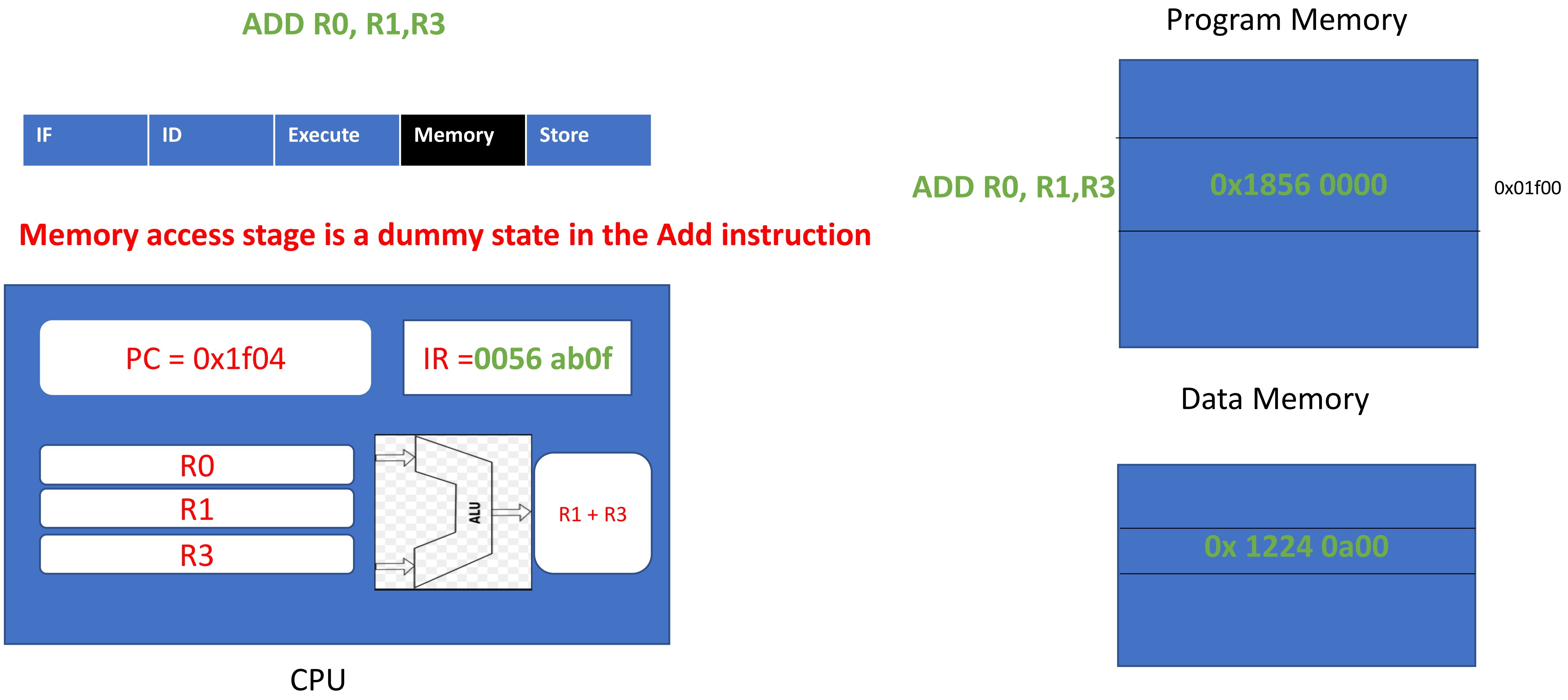
A 5-Stage Pipeline (ADD instruction)



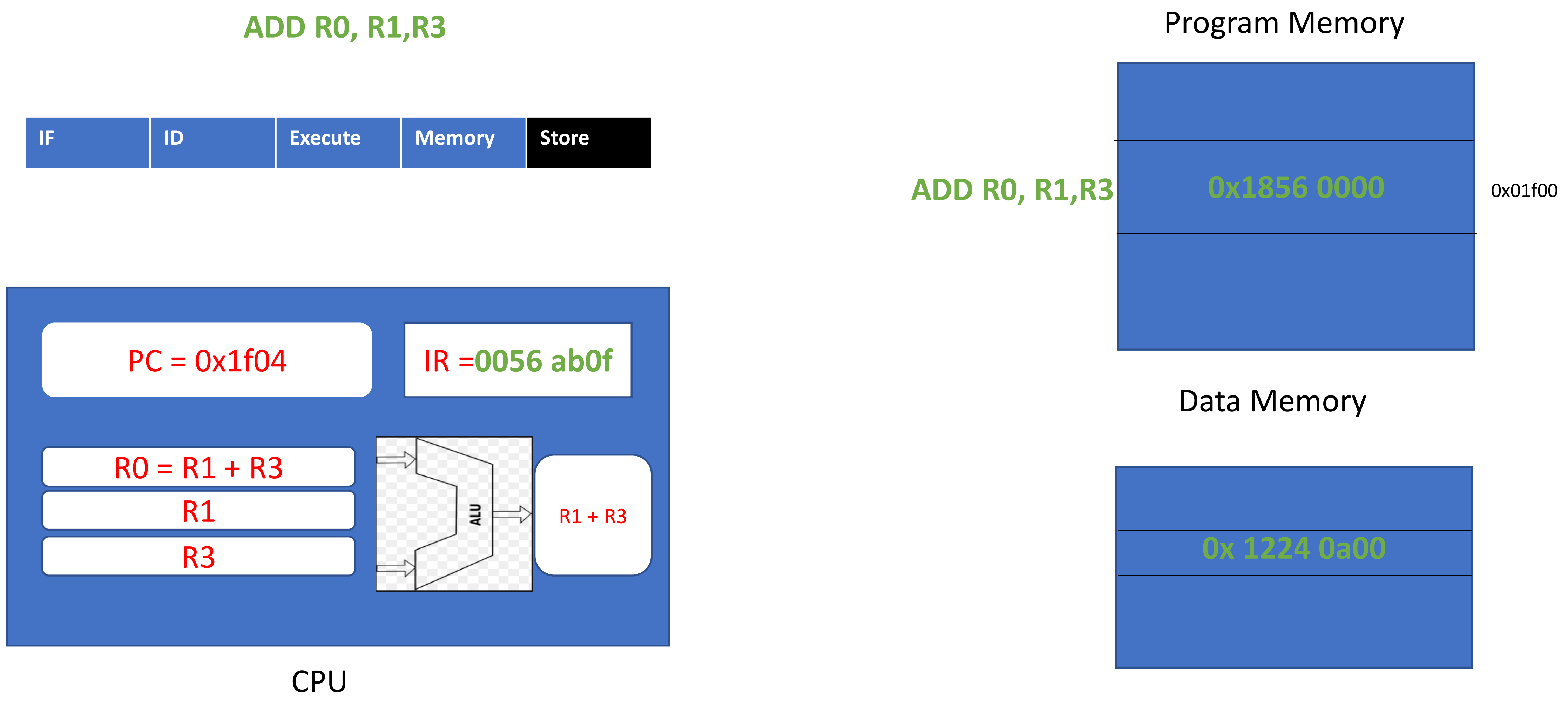
A 5-Stage Pipeline (ADD instruction)

ADD R0, R1,R3

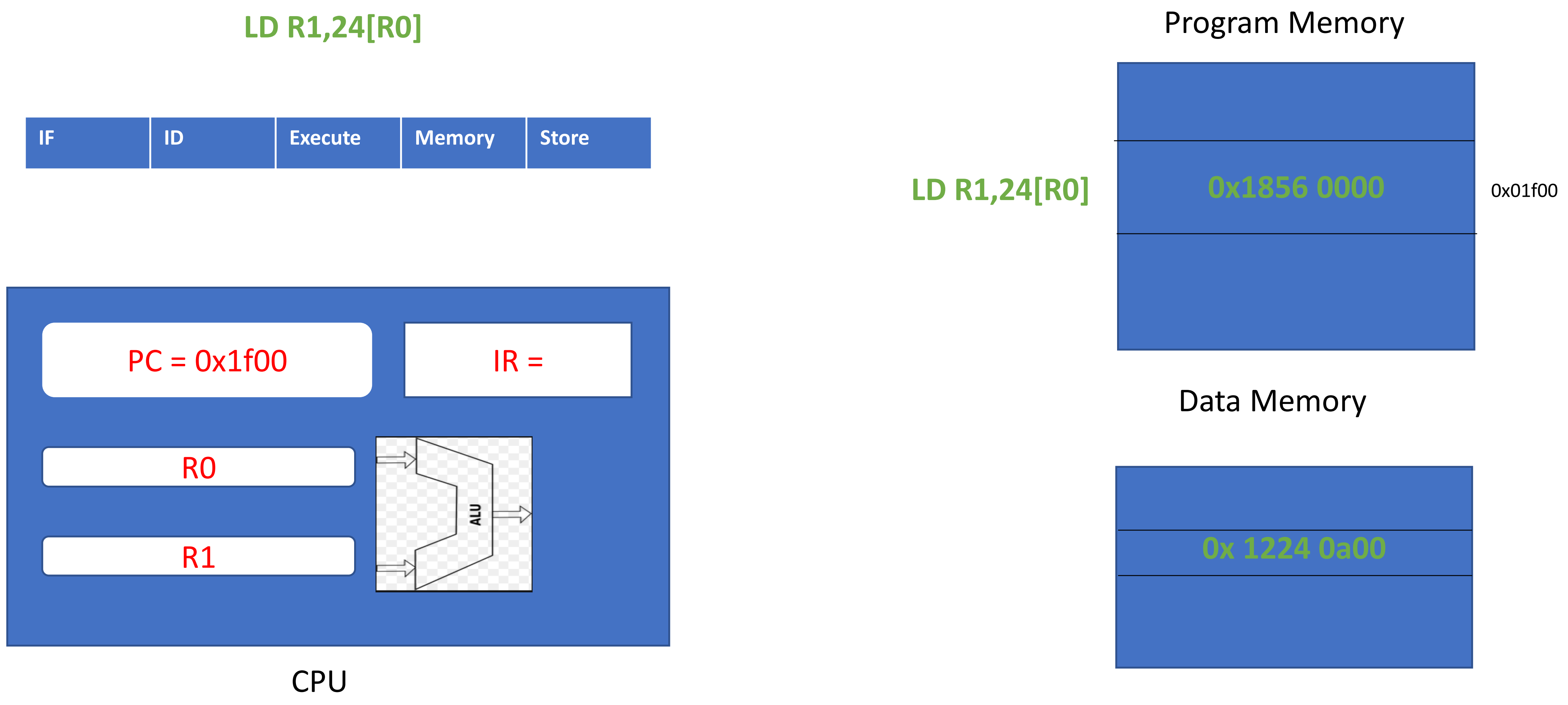
Memory access stage is a dummy state in the Add instruction



A 5-Stage Pipeline (ADD instruction)



A 5-Stage Pipeline (LD instruction)



A 5-Stage Pipeline (LD instruction)

LD R1,24[R0]

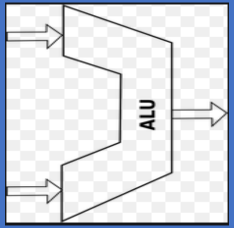
IF	ID	Execute	Memory	Store

PC = 0x1f04

IR =1856 0000

R0

R1



CPU

Program Memory

LD R1,24[R0] — 0x1856 0000 — 0x01f00

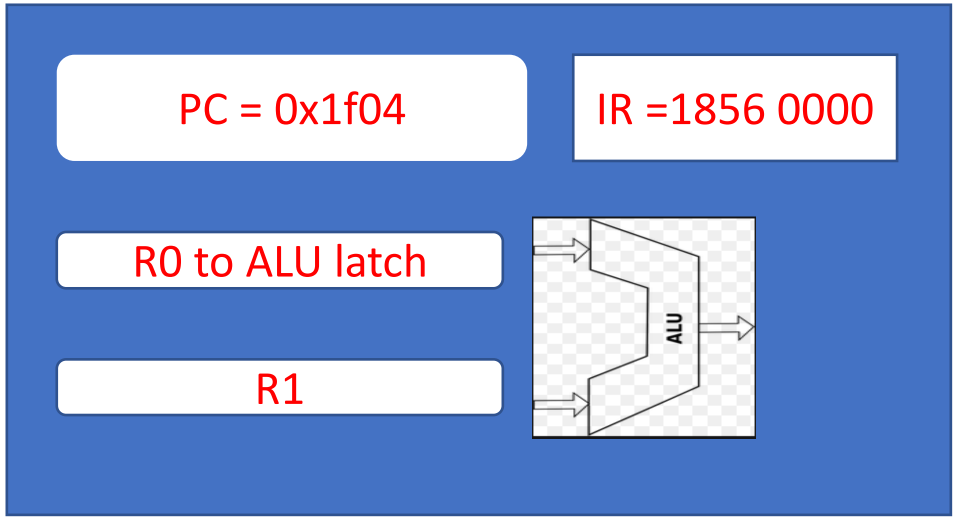
Data Memory

0x 1224 0a00

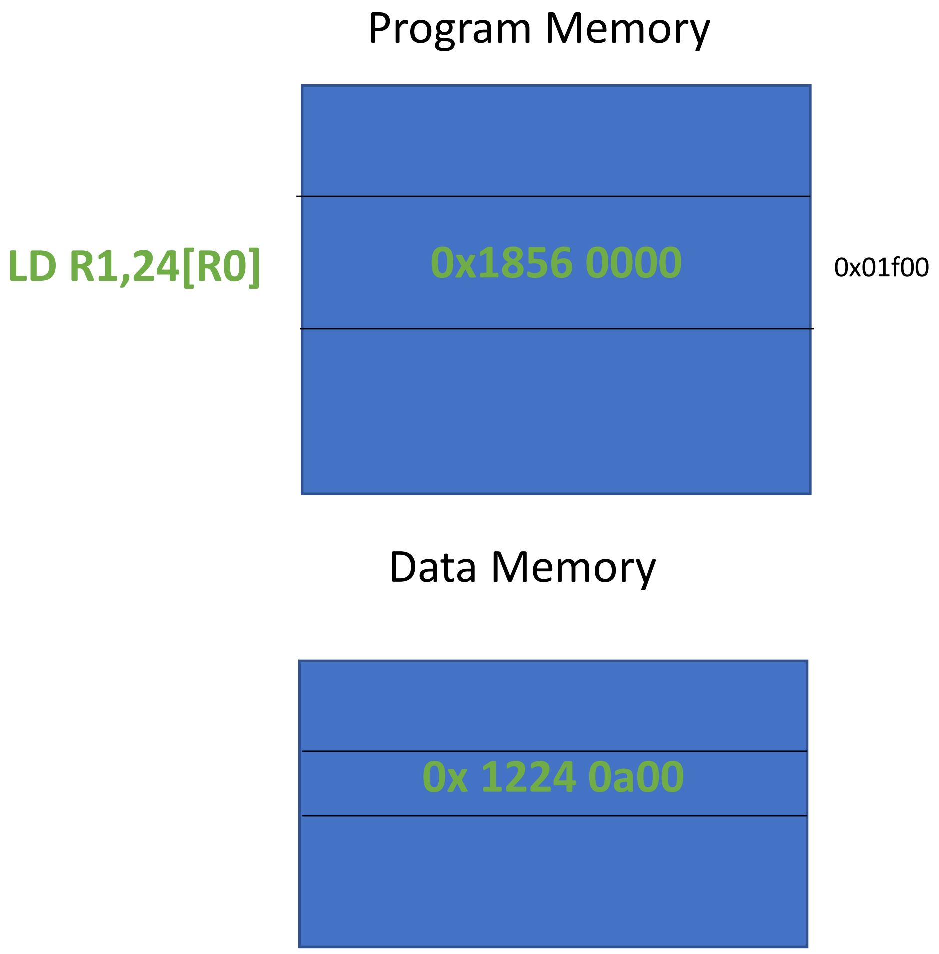
A 5-Stage Pipeline (LD instruction)

LD R1,24[R0]

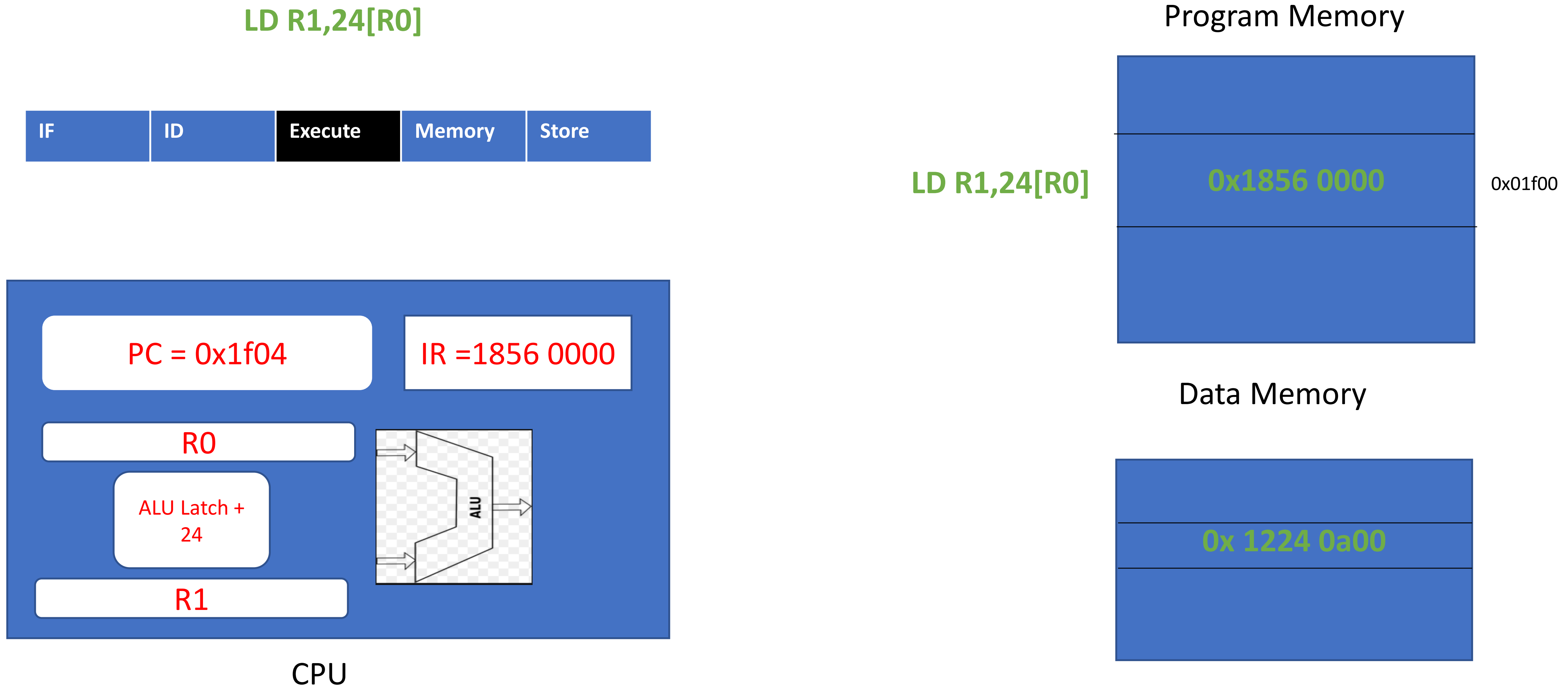
IF	ID	Execute	Memory	Store



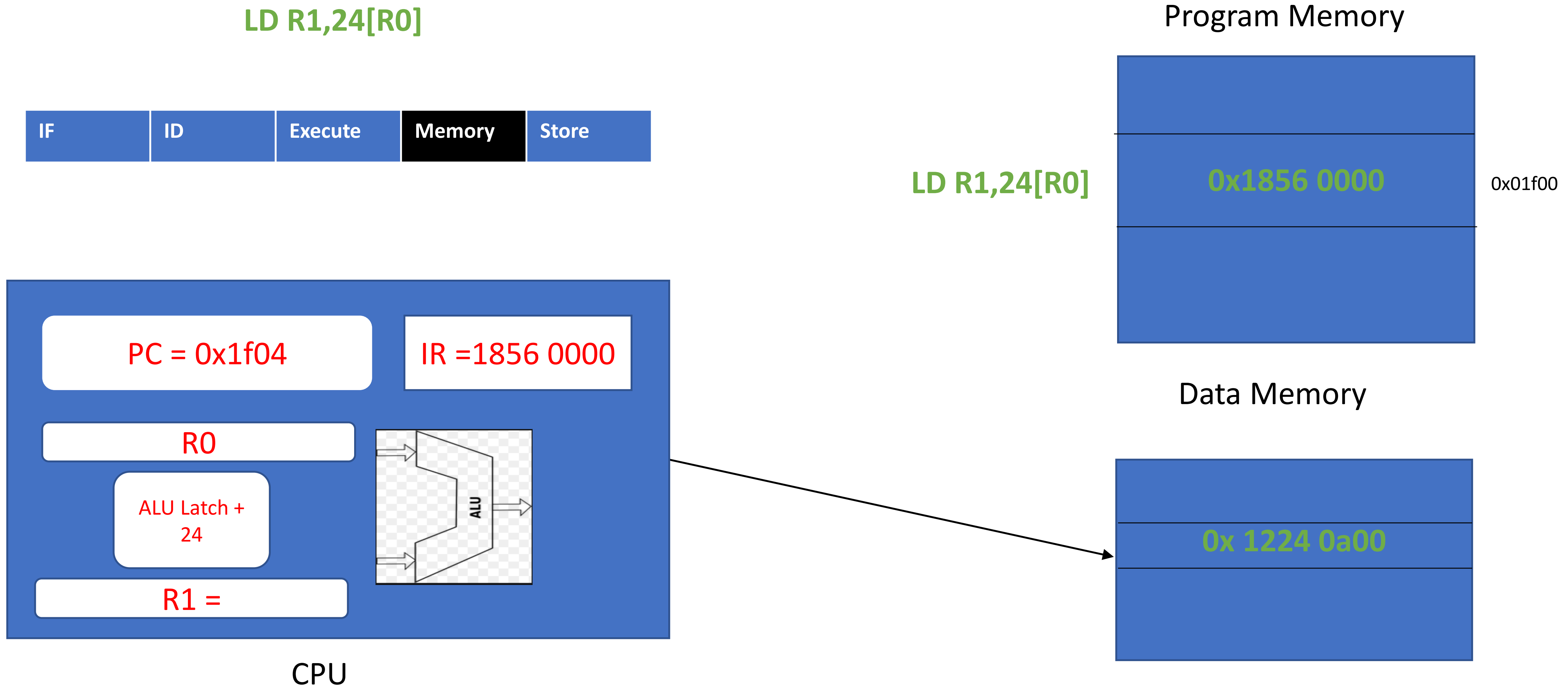
CPU



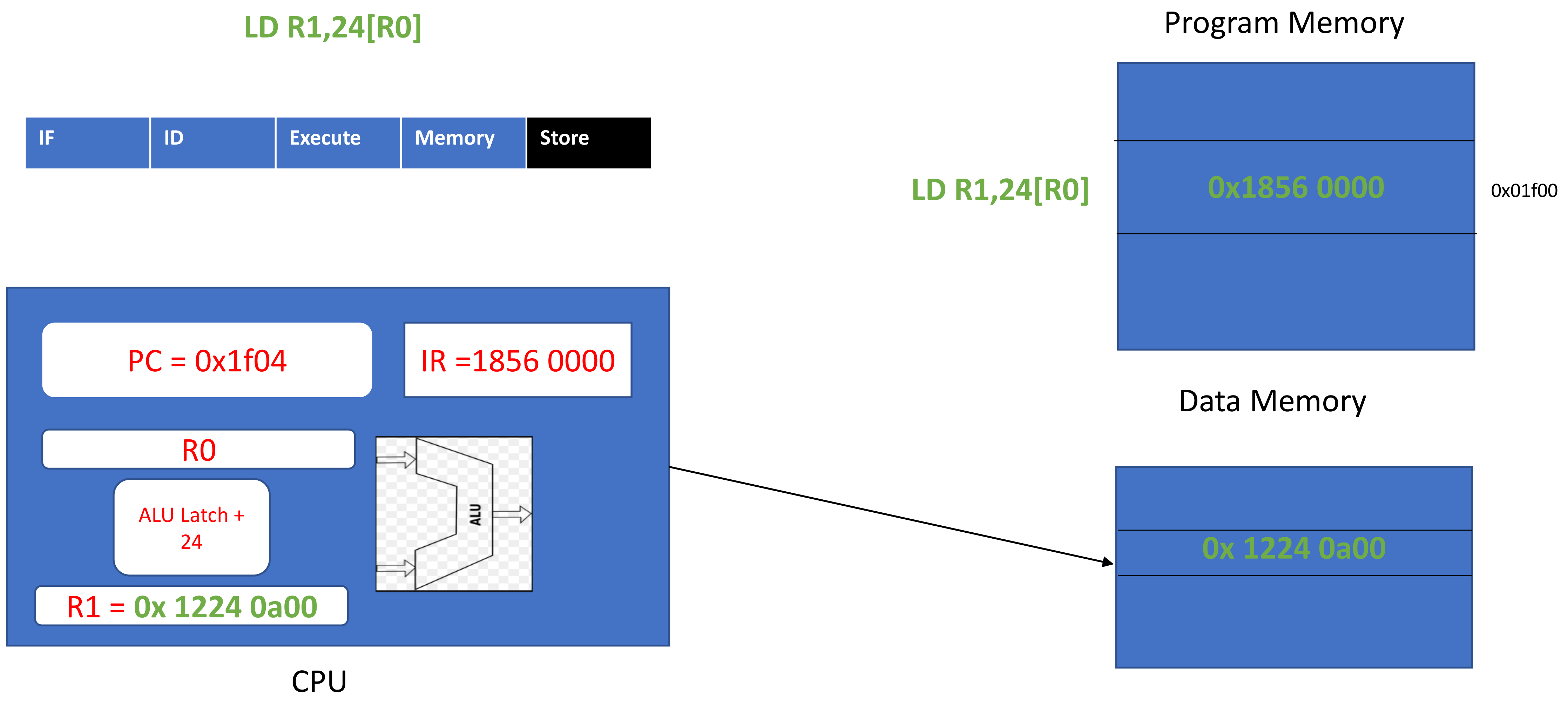
A 5-Stage Pipeline (LD instruction)



A 5-Stage Pipeline (LD instruction)



A 5-Stage Pipeline (LD instruction)



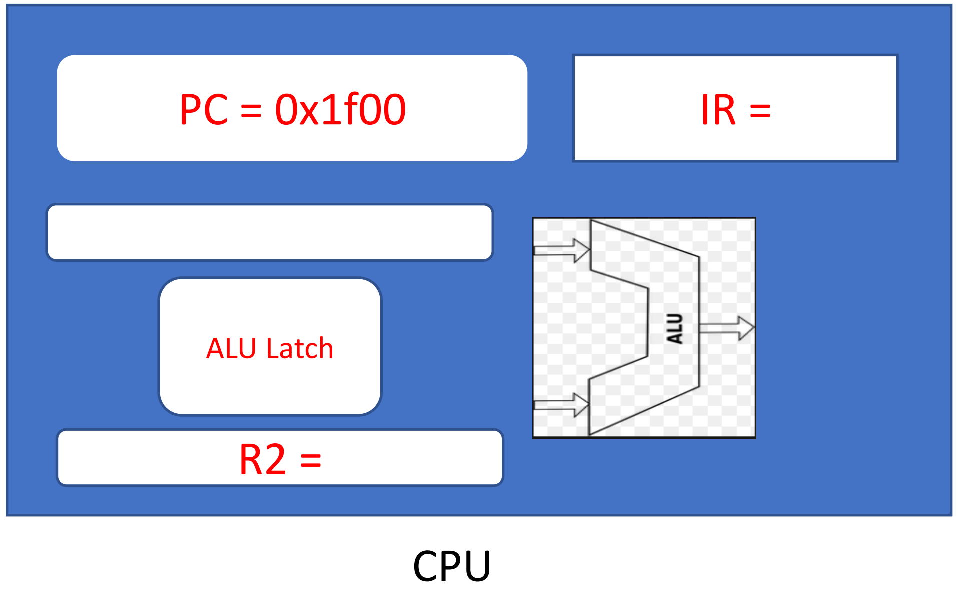
A 5-Stage Pipeline (Branch instruction)

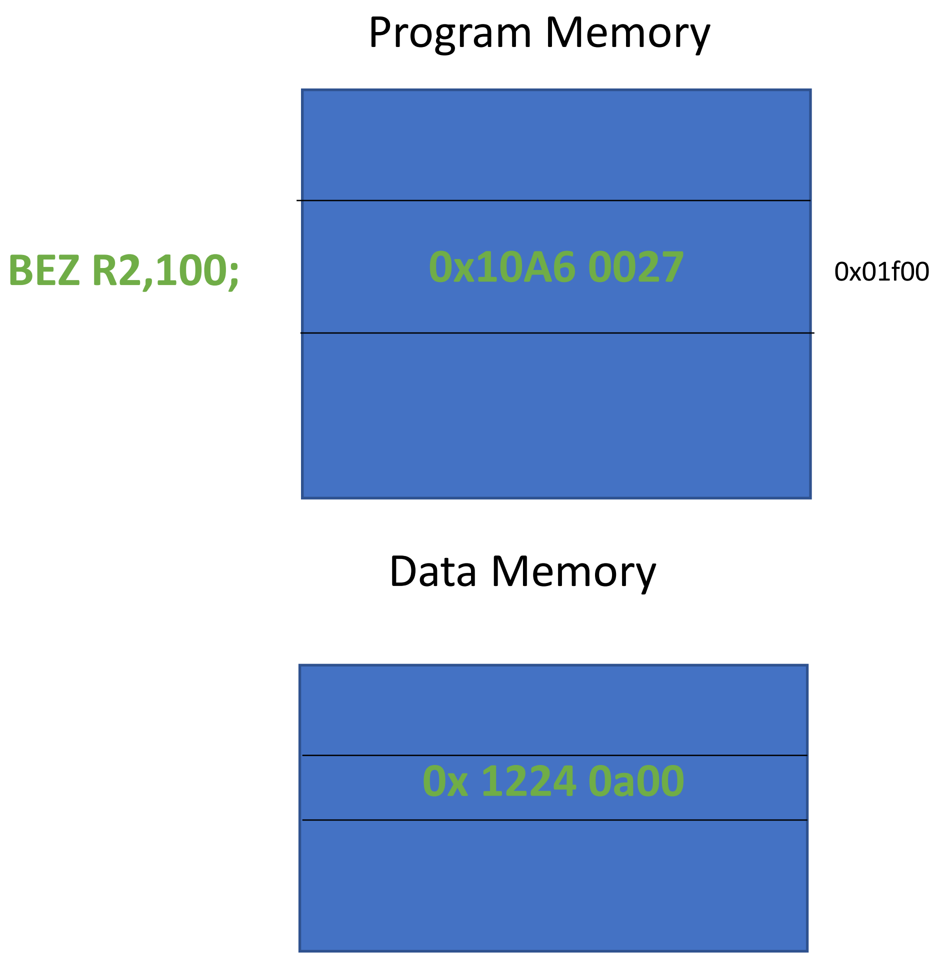
- If the instruction decoded was a branch or jump, the target address of the branch or jump was computed in parallel with reading the register file.
- The branch condition is computed after the register file is read, and if the branch is taken or if the instruction is a jump, the PC predictor in the first stage is assigned the branch target, rather than the incremented PC that has been computed.
- So the branch instruction and call instruction are completed in two cycles.

A 5-Stage Pipeline (Branch instruction)

BEZ R2,100; // If R2 is zero branch to location PC+100.

IF	ID	Execute	Memory	Store

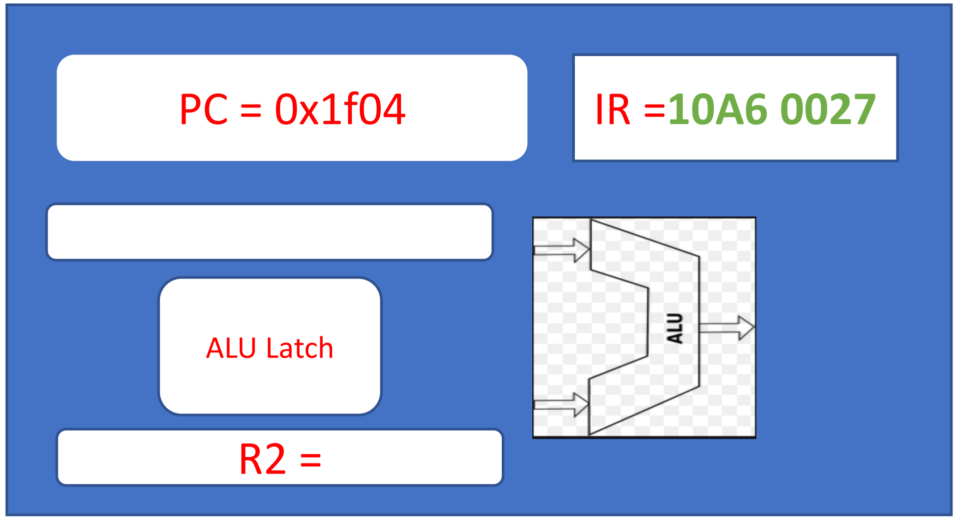




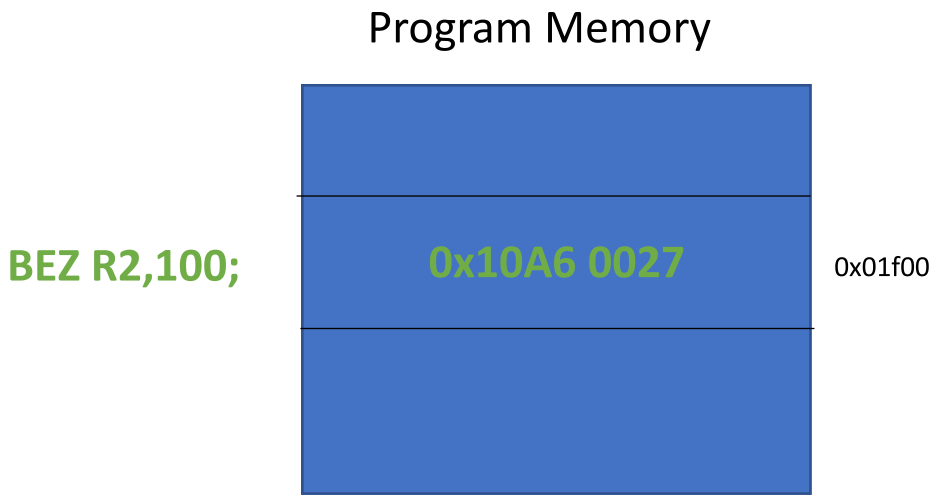
A 5-Stage Pipeline (Branch instruction)

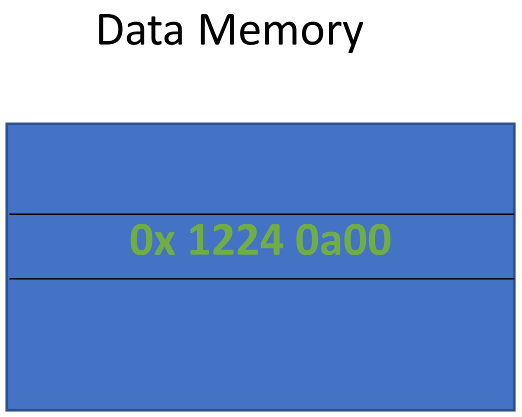
BEZ R2,100; // If R2 is zero branch to location PC+100.

IF	ID	Execute	Memory	Store



CPU





A 5-Stage Pipeline (Branch instruction)

BEZ R2,100; // If R2 is zero branch to location PC+100.

IF	ID	Execute	Memory	Store

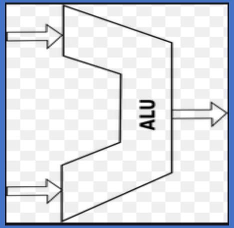
PC = 0x1f04

IR =10A6 0027

TB = PC + 100

ALU Latch Compare against zero

R2 to ALU Latch



CPU

Program Memory

BEZ R2,100;	0x10A6 0027	0x01f00

Data Memory

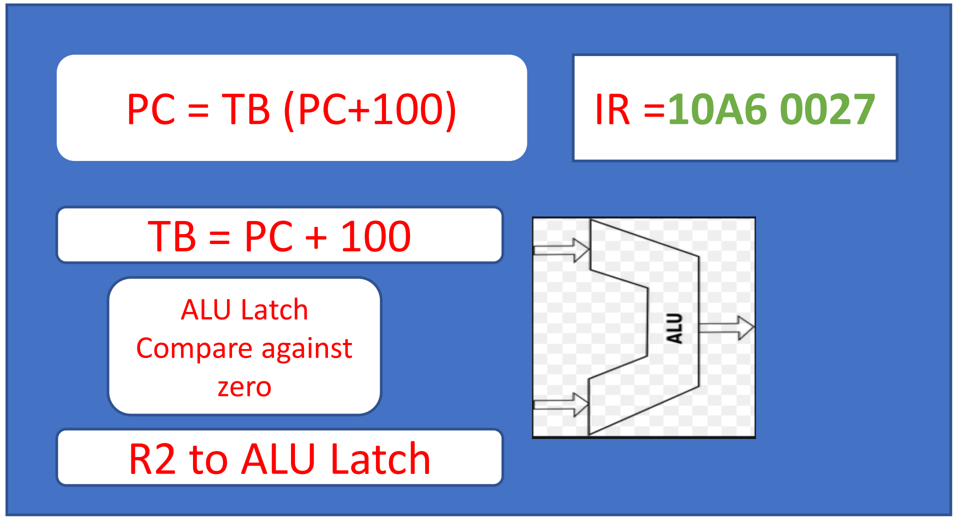
0x 1224 0a00

A 5-Stage Pipeline (Branch instruction)

BEZ R2,100; // If R2 is zero branch to location PC+100.

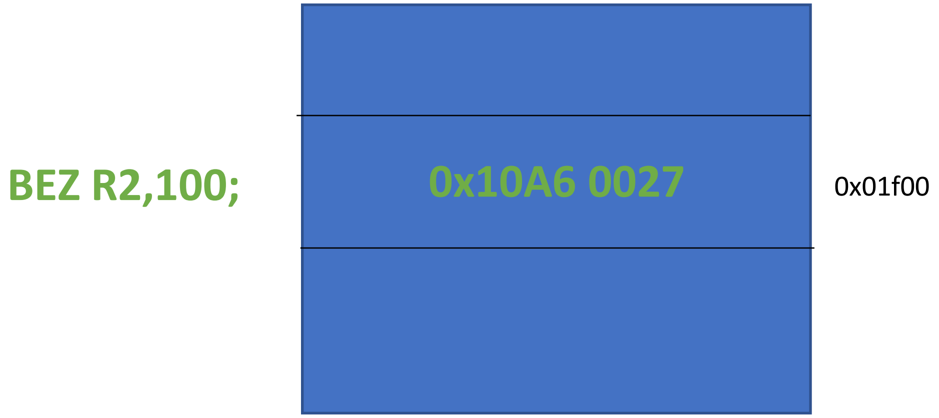
IF	ID	Execute	Memory	Store

If the Branch to be taken , the PC is updated



CPU

Program Memory



Data Memory

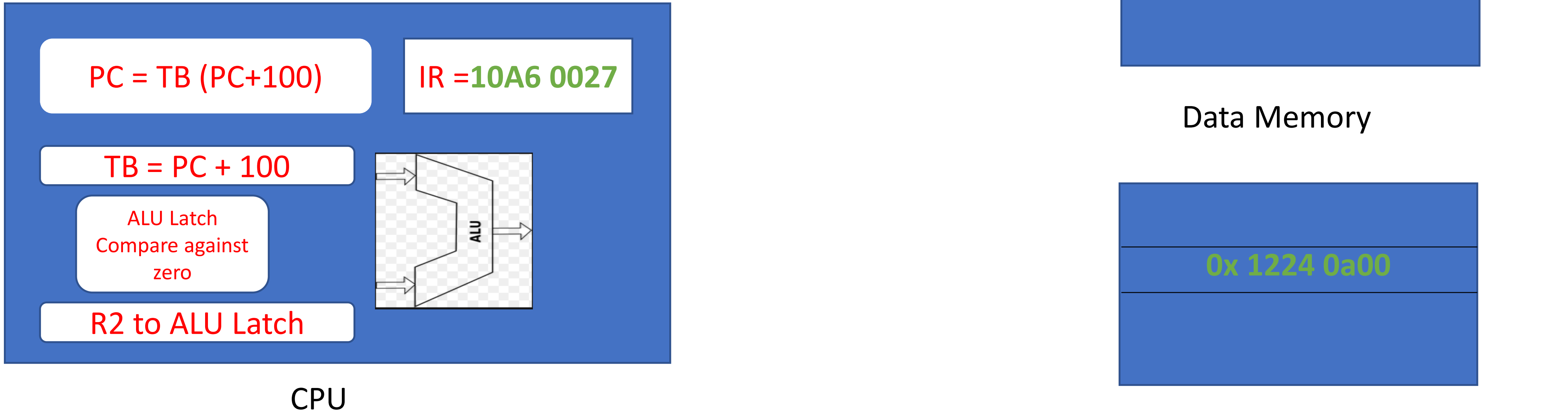
0x 1224 0a00

A 5-Stage Pipeline (Branch instruction)

BEZ R2,100; // If R2 is zero branch to location PC+100.

IF	ID	Execute	Memory	Store

The Execute stage has no additional work to do.



A 5-Stage Pipeline (Branch instruction)

BEZ R2,100; // If R2 is zero branch to location PC+100.

IF	ID	Execute	Memory	Store

The Memory stage has no additional work to do.

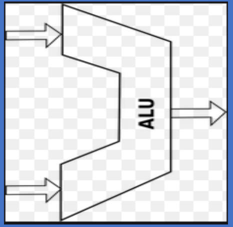
PC = TB (PC+100)

IR =10A6 0027

TB = PC + 100

ALU Latch
Compare against zero

R2 to ALU Latch



CPU

Program Memory

BEZ R2,100;	0x10A6 0027	0x01f00

Data Memory

0x 1224 0a00

A 5-Stage Pipeline (Branch instruction)

BEZ R2,100; // If R2 is zero branch to location PC+100.

IF	**ID**	**Execute**	**Memory**	**Store**

The Store stage has no additional work to do.

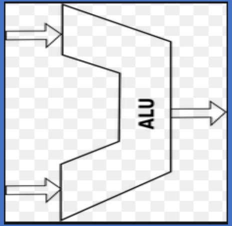
PC = TB (PC+100)

IR =10A6 0027

TB = PC + 100

ALU Latch
Compare against
zero

R2 to ALU Latch



CPU

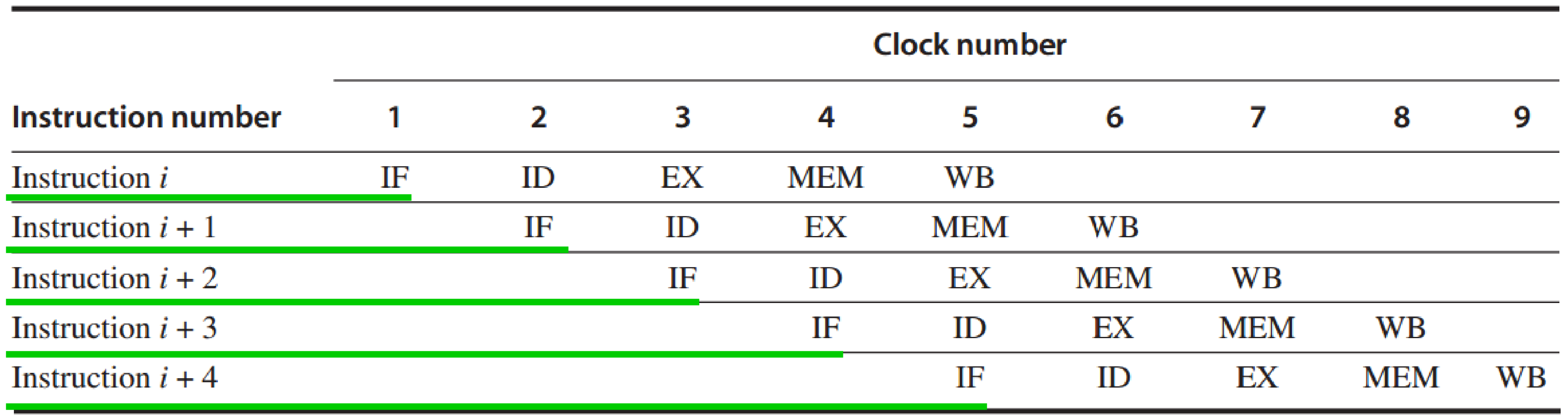
Program Memory

BEZ R2,100; — 0x10A6 0027 — 0x01f00

Data Memory

0x 1224 0a00

RISC: Five-Stage Pipeline



	Clock number								
Instruction number	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Instruction *i*	IF	ID	EX	MEM	WB				
Instruction *i* + 1		IF	ID	EX	MEM	WB			
Instruction *i* + 2			IF	ID	EX	MEM	WB		
Instruction *i* + 3				IF	ID	EX	MEM	WB	
Instruction *i* + 4					IF	ID	EX	MEM	WB

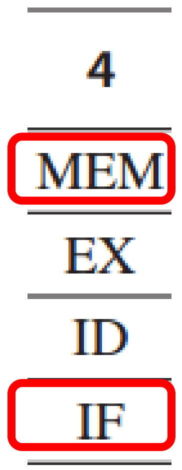
Simply start a new instruction
on each clock cycle;
Speedup = 5.

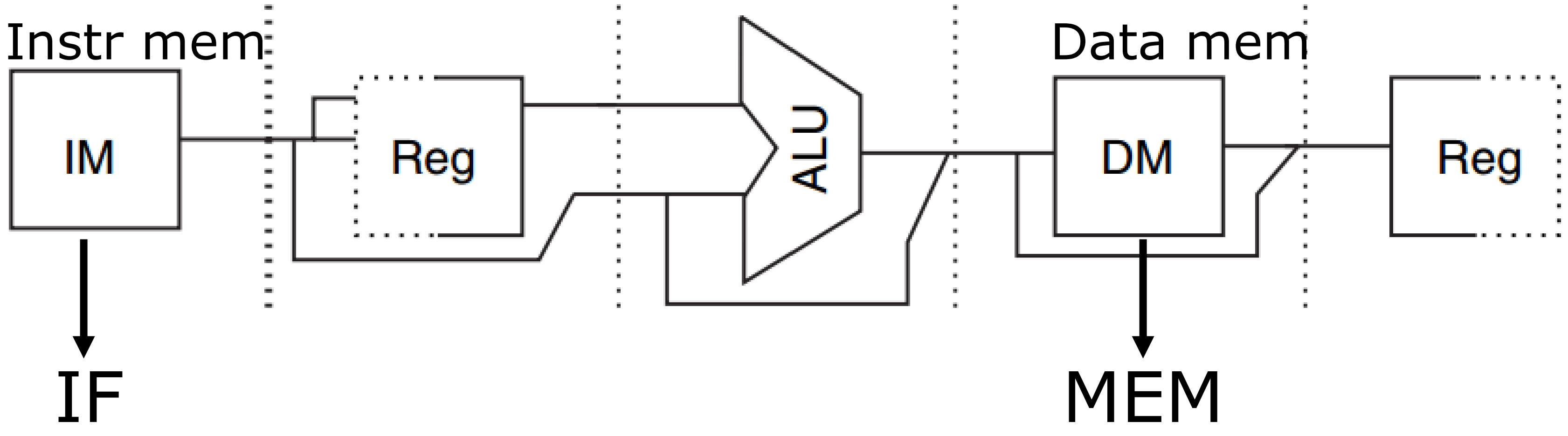
RISC: Five-Stage Pipeline

- To achieve the pipelining, we need separate instruction memory and data memory.
- Because both IF and MEM require memory access, if there is only one single memory, IF and MEM will have memory conflicts during the pipelining process.

RISC: Five-Stage Pipeline

separate instruction and data mems to eliminate conflicts for a single memory between instruction fetch and data memory access.


4
MEM
EX
ID
IF

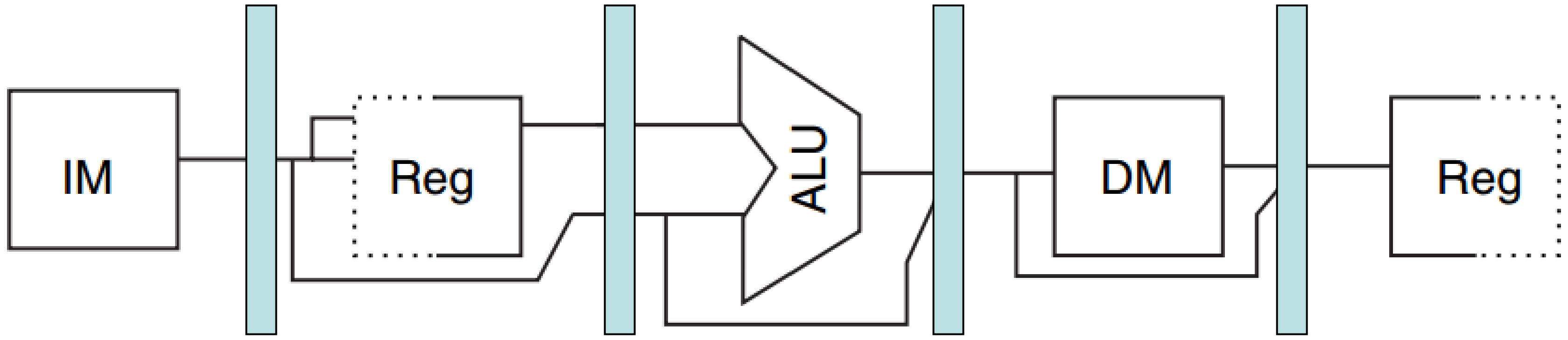


RISC: Five-Stage Pipeline

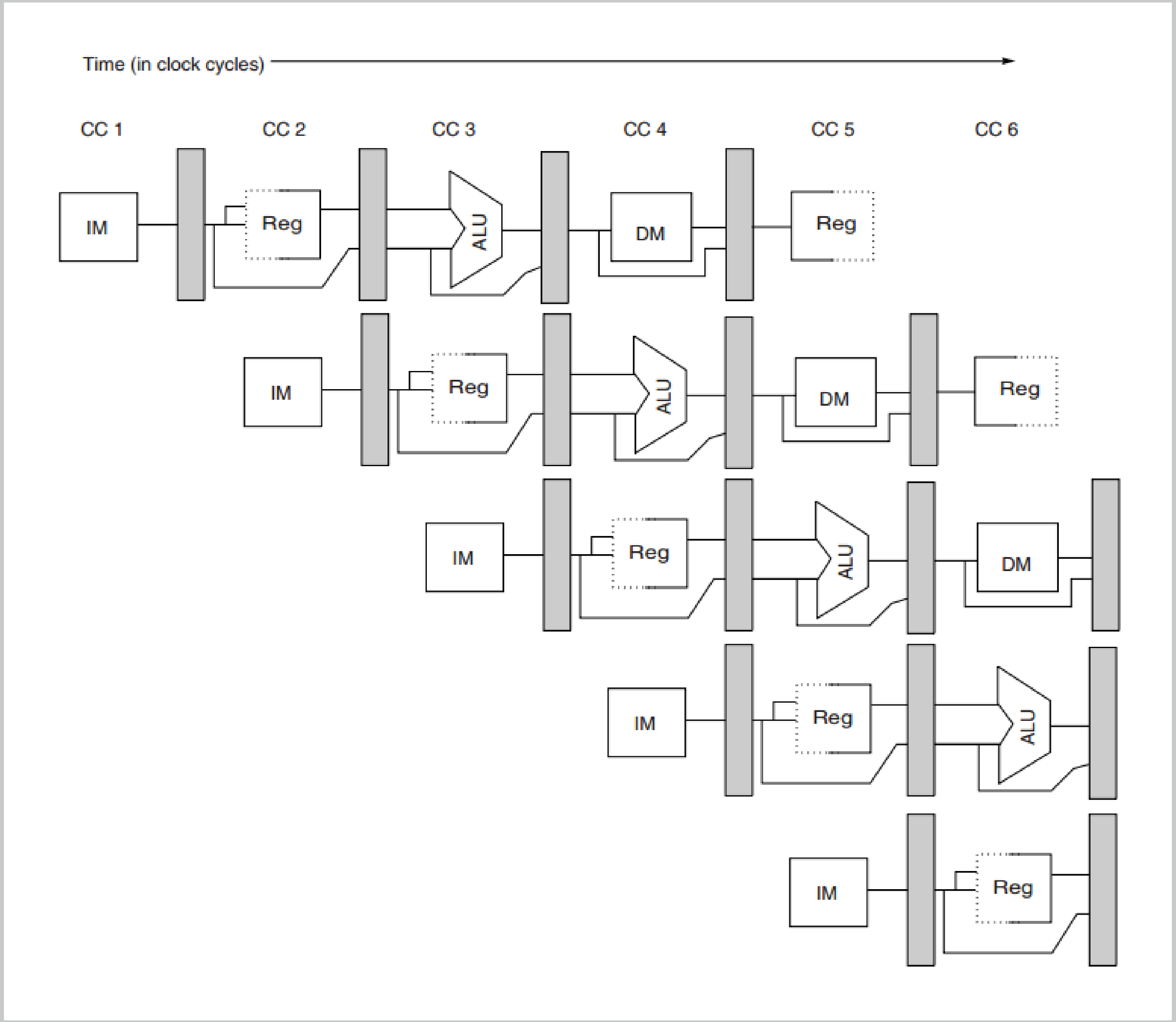
- How it works

  introduce pipeline registers between successive stages;

  pipeline registers store the results of a stage and use them as the input of the next stage.

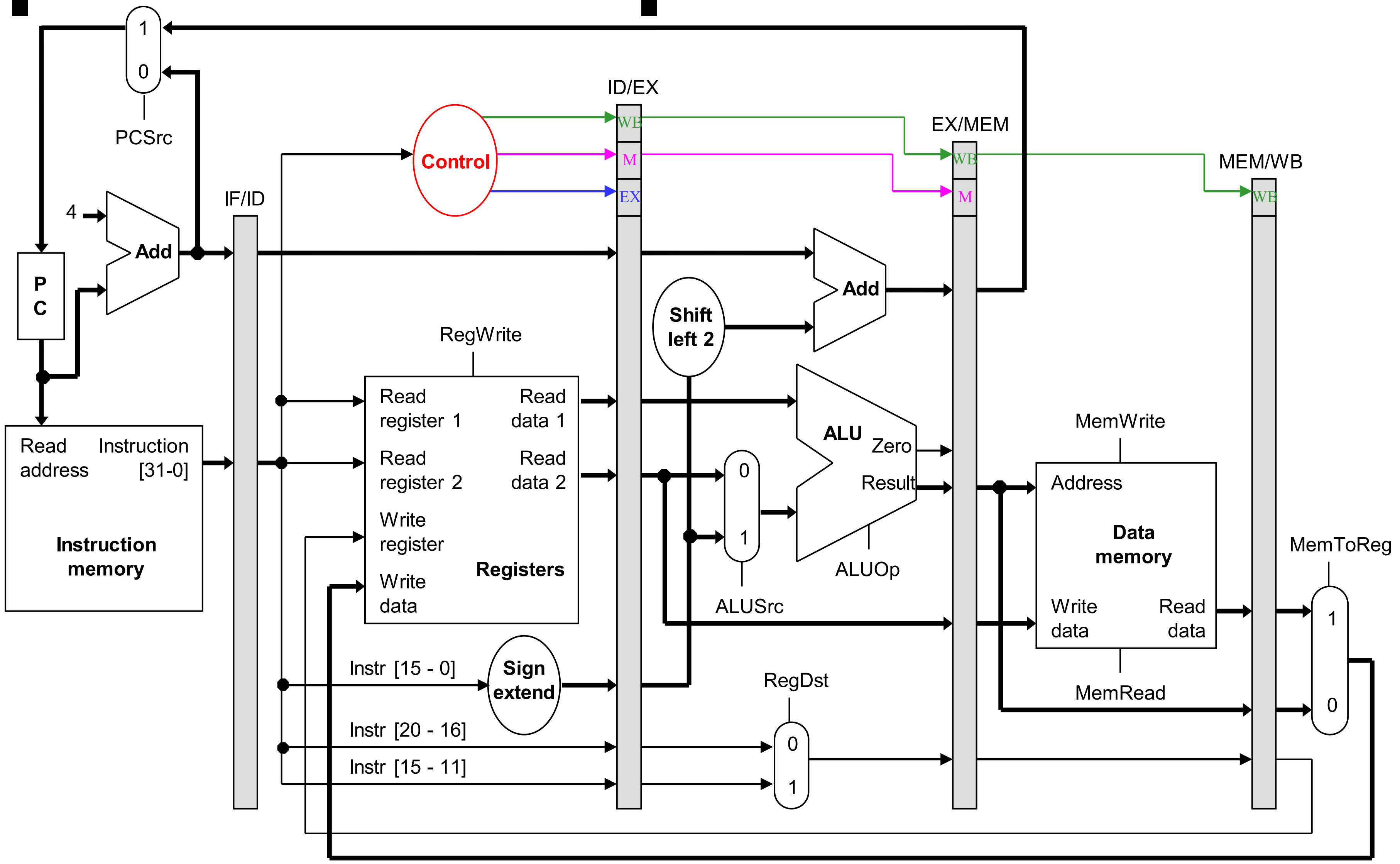


RISC: Five-Stage Pipeline



- high level paradigm of RISC pipelining

Pipelined datapath and control



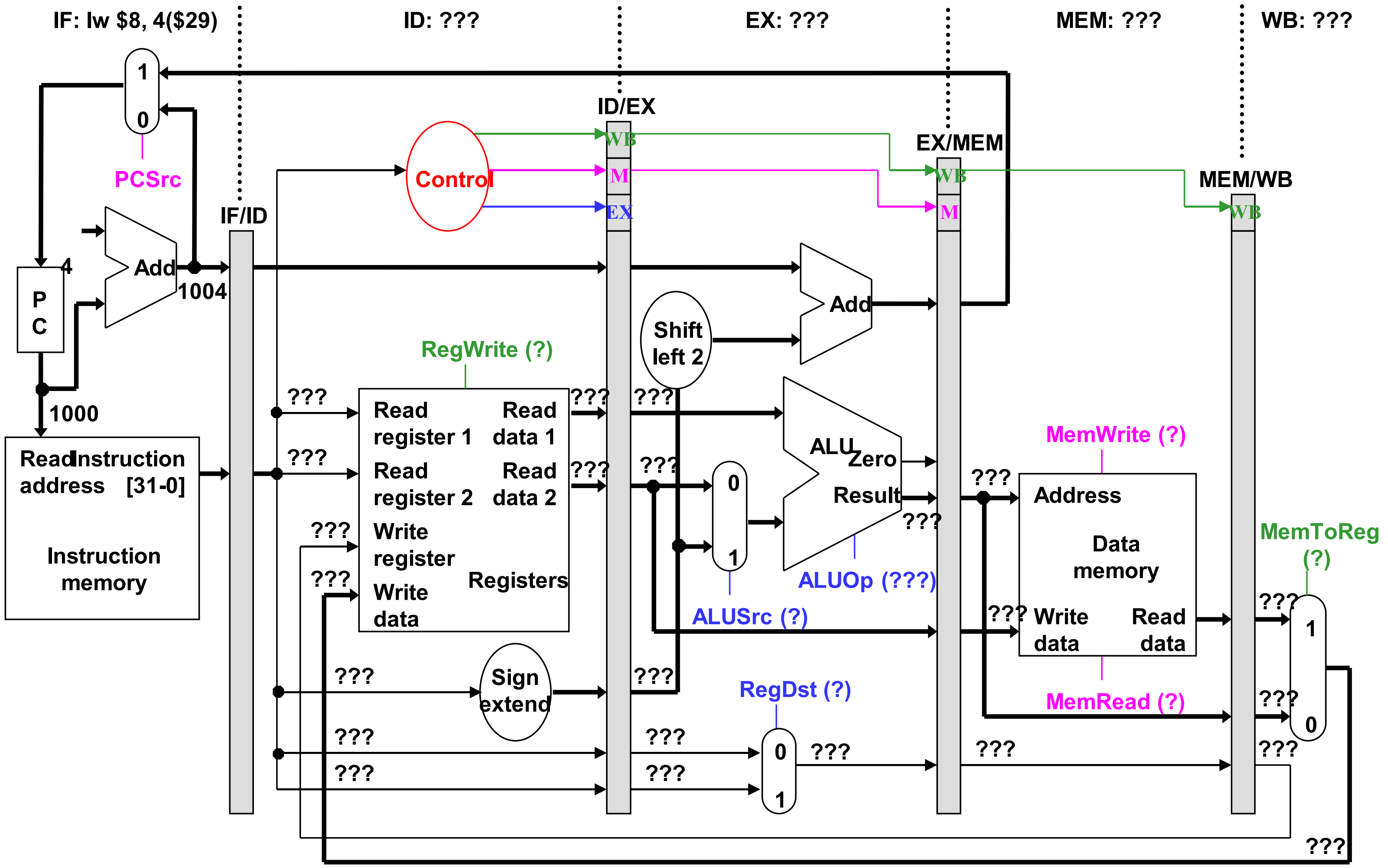
An example execution sequence

- Here’s a sample sequence of instructions to execute

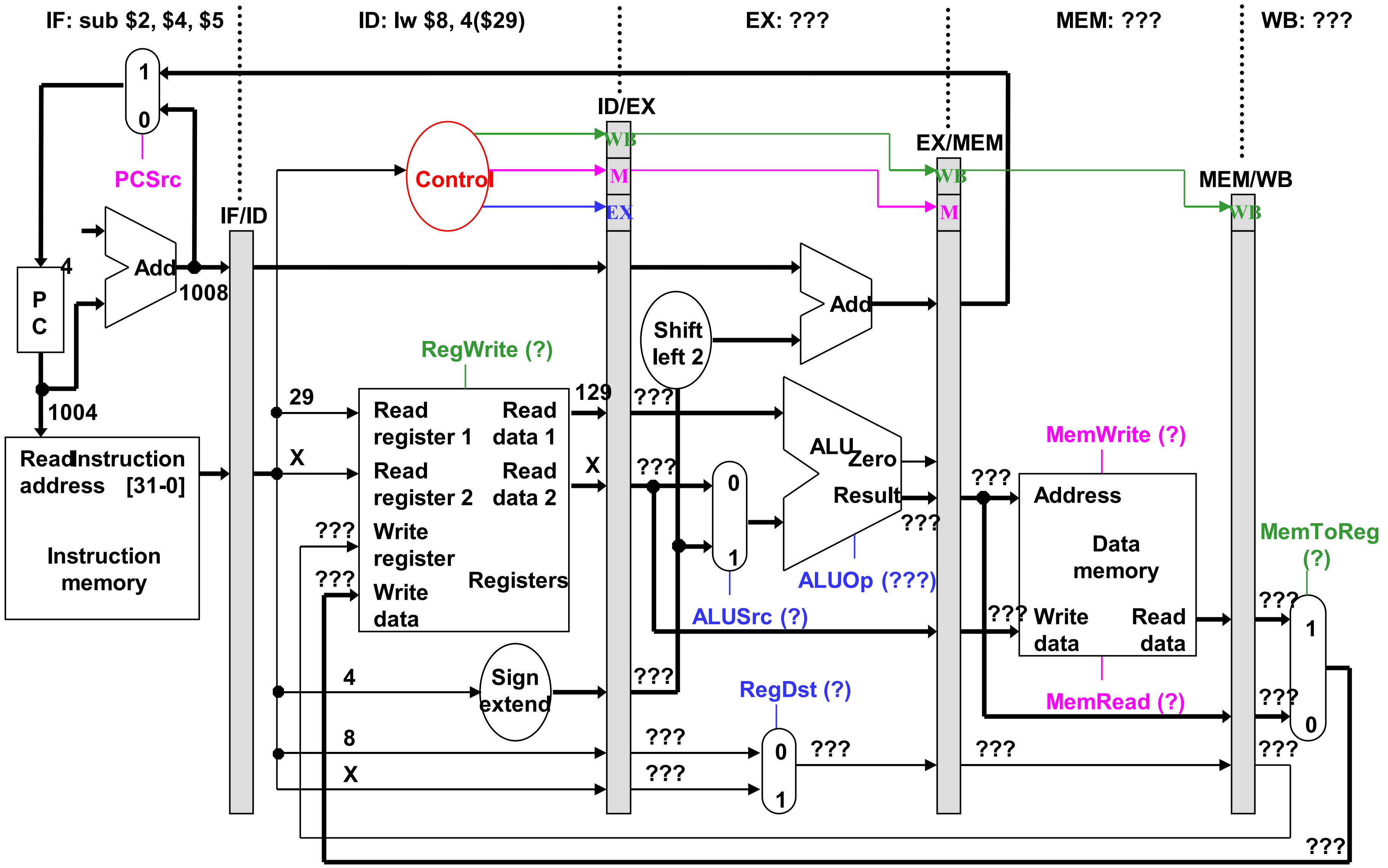
```
1000: lw  $8, 4($29)
1004: sub $2, $4, $5
1008: and $9, $10, $11
1012: or  $16, $17, $18
1016: add $13, $14, $0
```

- We’ll make some assumptions, just so we can show actual data values:
  - Each register contains its number plus 100. For instance, register $8 contains 108, register $29 contains 129, etc.
  - Every data memory location contains 99
- Our pipeline diagrams will follow some conventions:
  - An X indicates values that aren’t important, like the constant field of an R-type instruction
  - Question marks ??? indicate values we don’t know, usually resulting from instructions coming before and after the ones in our example

Cycle 1 (filling)



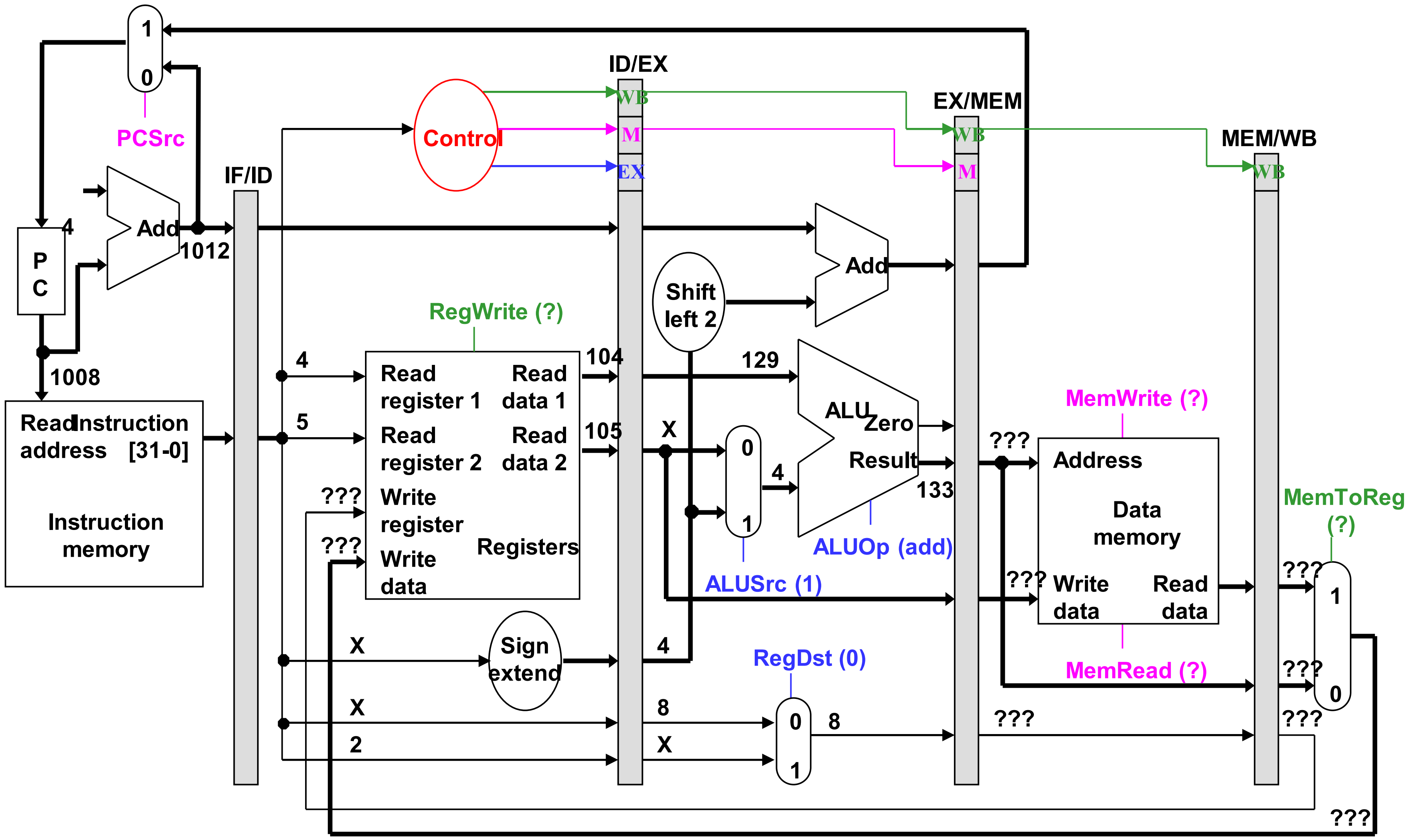
Cycle 2



IF: sub $2, $4, $5
ID: lw $8, 4($29)
EX: ???
MEM: ???
WB: ???
1
0
PCSrc
ID/EX
WB
M
EX
Control
EX/MEM
WB
M
MEM/WB
WB
IF/ID
4
Add
P
C
1008
Add
Shift
left 2
RegWrite (?)
29
129
???
1004
Read
register 1
Read
data 1
ALU
Zero
Result
MemWrite (?)
X
Read
register 2
Read
data 2
X
???
0
1
???
???
Address
Read
address
Instruction
[31-0]
Instruction
memory
???
Write
register
???
Write
data
Registers
ALUSrc (?)
ALUOp (???)
Data
memory
???
Write
data
Read
data
MemToReg
(?)
???
1
0
4
Sign
extend
???
RegDst (?)
MemRead (?)
???
8
???
0
1
???
???
???
X
???
???

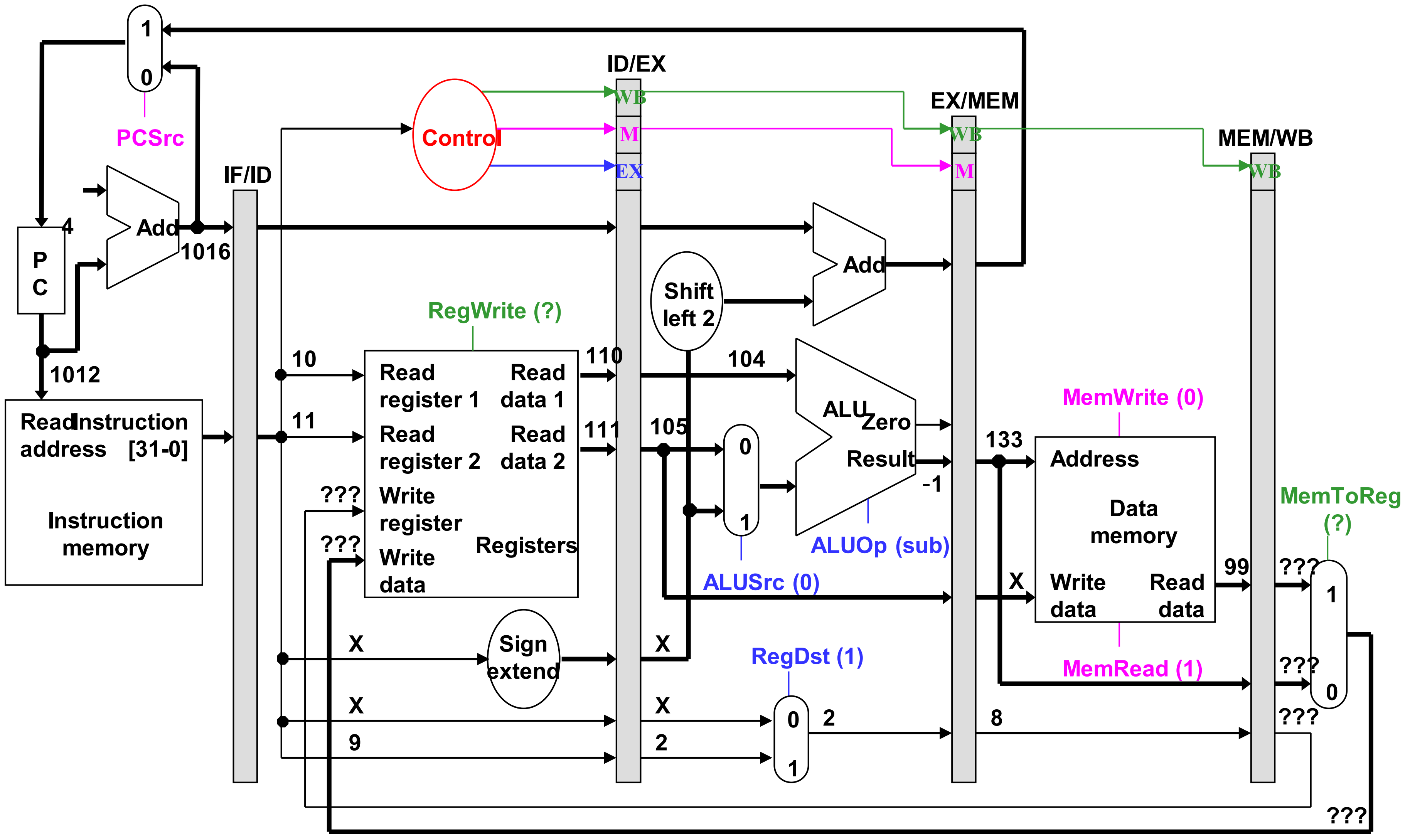
Cycle 3

IF: and \$9, \$10, \$11 ID: sub \$2, \$4, \$5 EX: lw \$8, 4(\$29) MEM: ??? WB: ???

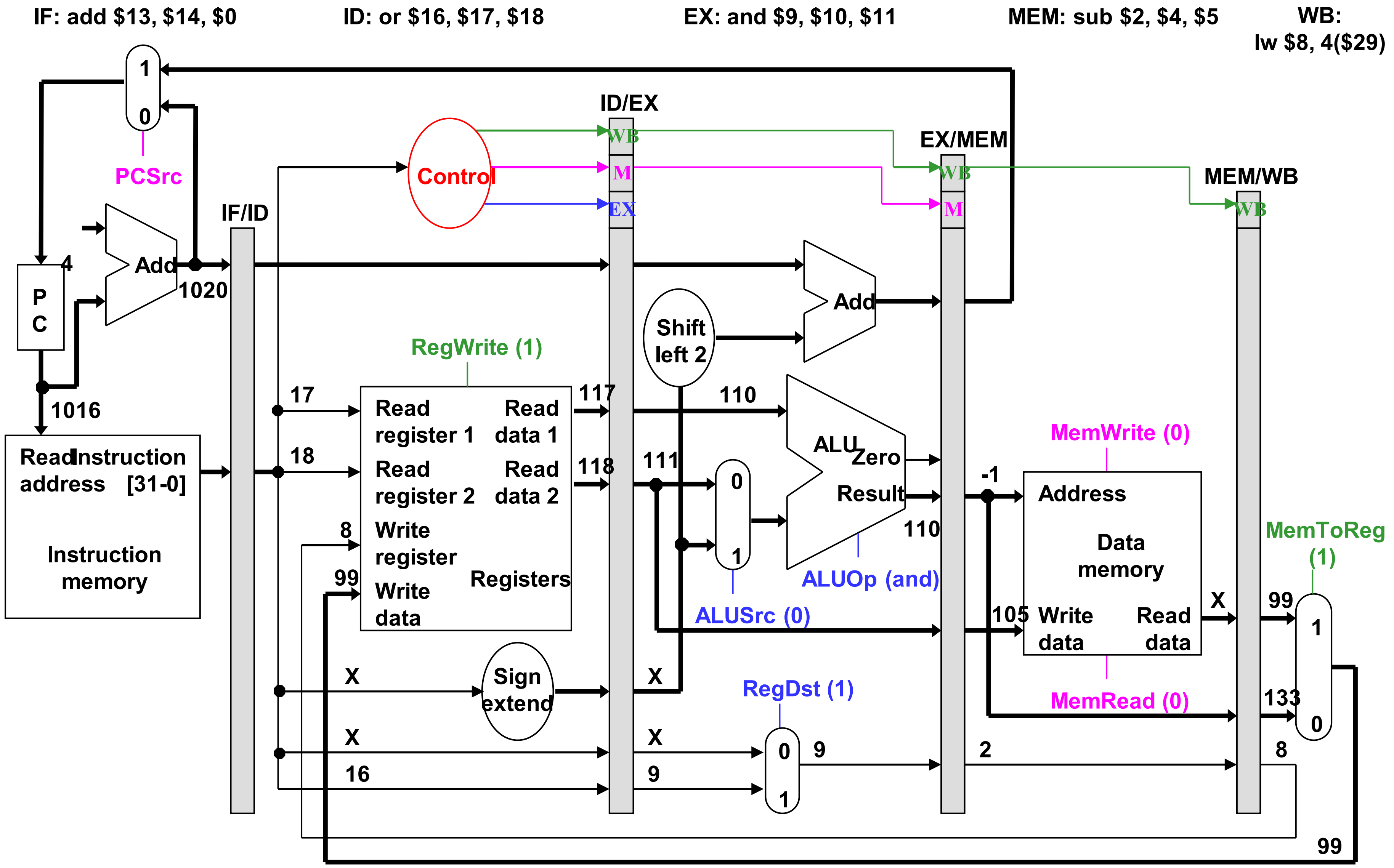


Cycle 4

IF: or \$16, \$17, \$18 ID: and \$9, \$10, \$11 EX: sub \$2, \$4, \$5 MEM: lw \$8, 4(\$29) WB: ???

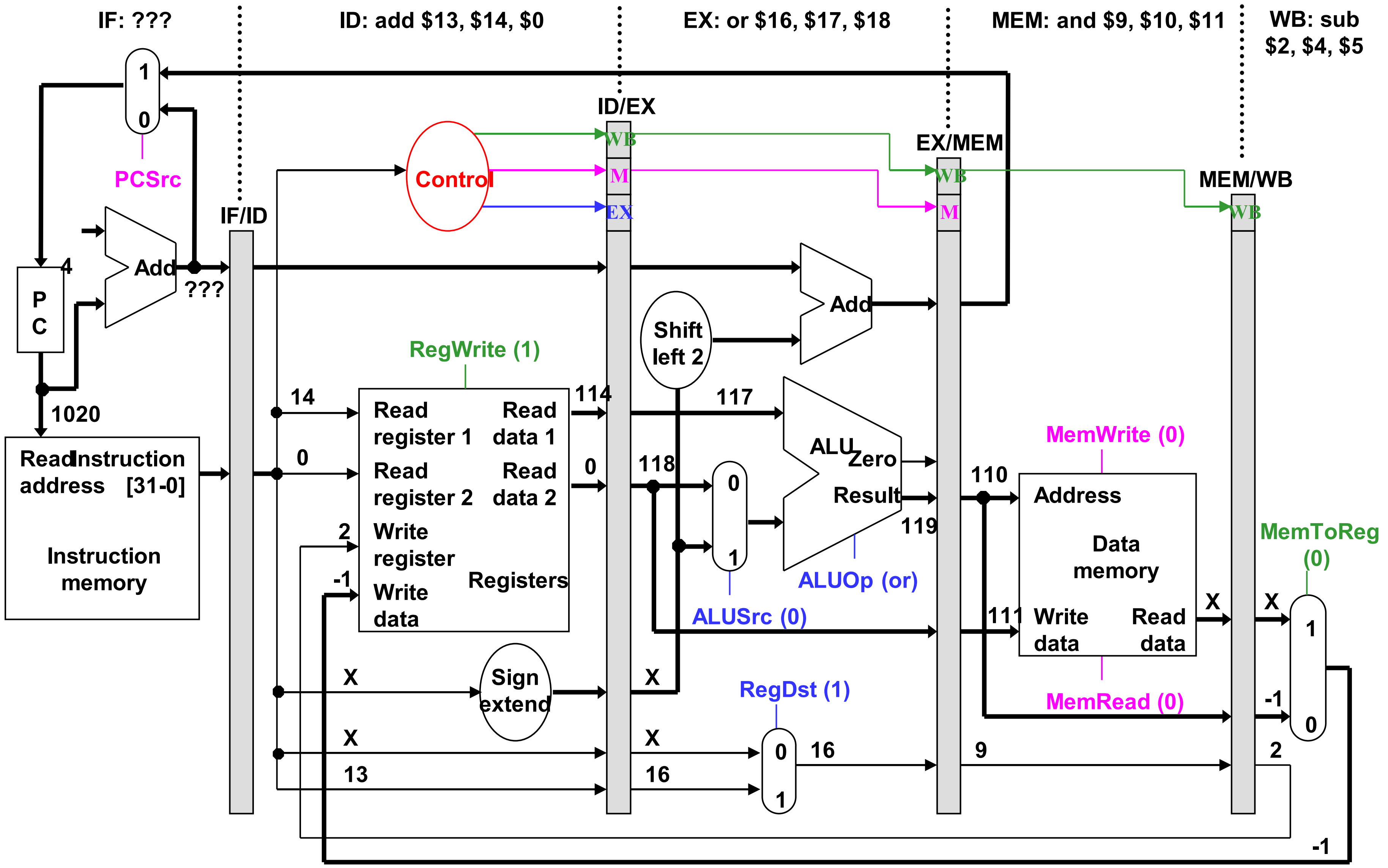


Cycle 5 (full)

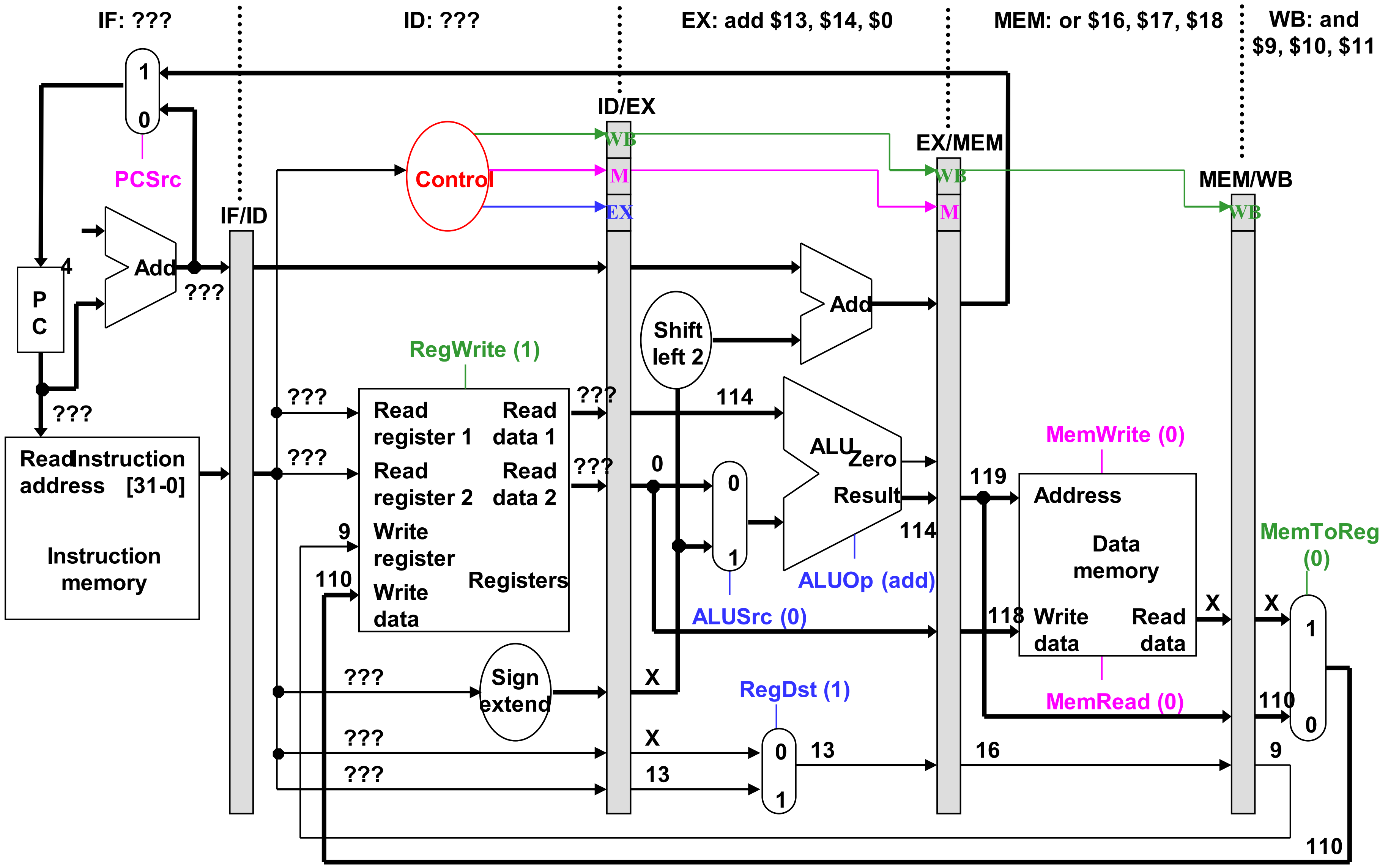


IF: add $13, $14, $0
ID: or $16, $17, $18
EX: and $9, $10, $11
MEM: sub $2, $4, $5
WB: lw $8, 4($29)
1
0
PCSrc
Control
ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
IF/ID
4
Add
1020
P C
1016
Read address
Instruction [31-0]
Instruction memory
Add
Shift left 2
RegWrite (1)
17
18
8
99
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
117
118
110
111
0
1
ALU
Zero
Result
110
ALUSrc (0)
ALUOp (and)
-1
MemWrite (0)
Address
Data memory
Write data
Read data
105
MemRead (0)
X
99
MemToReg (1)
1
0
133
8
X
Sign extend
X
X
X
16
9
RegDst (1)
0
1
9
2
99

Cycle 6 (emptying)

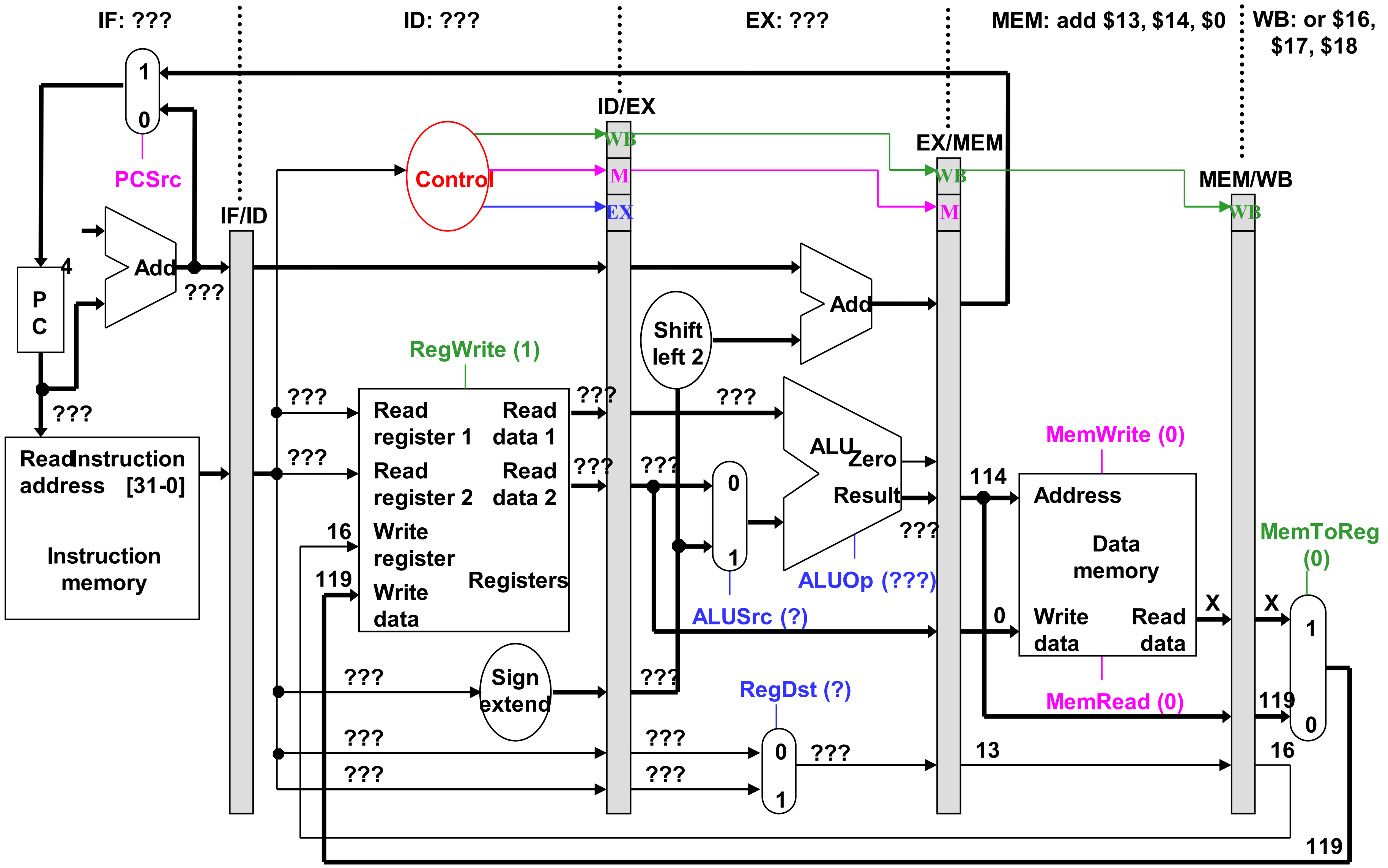


Cycle 7



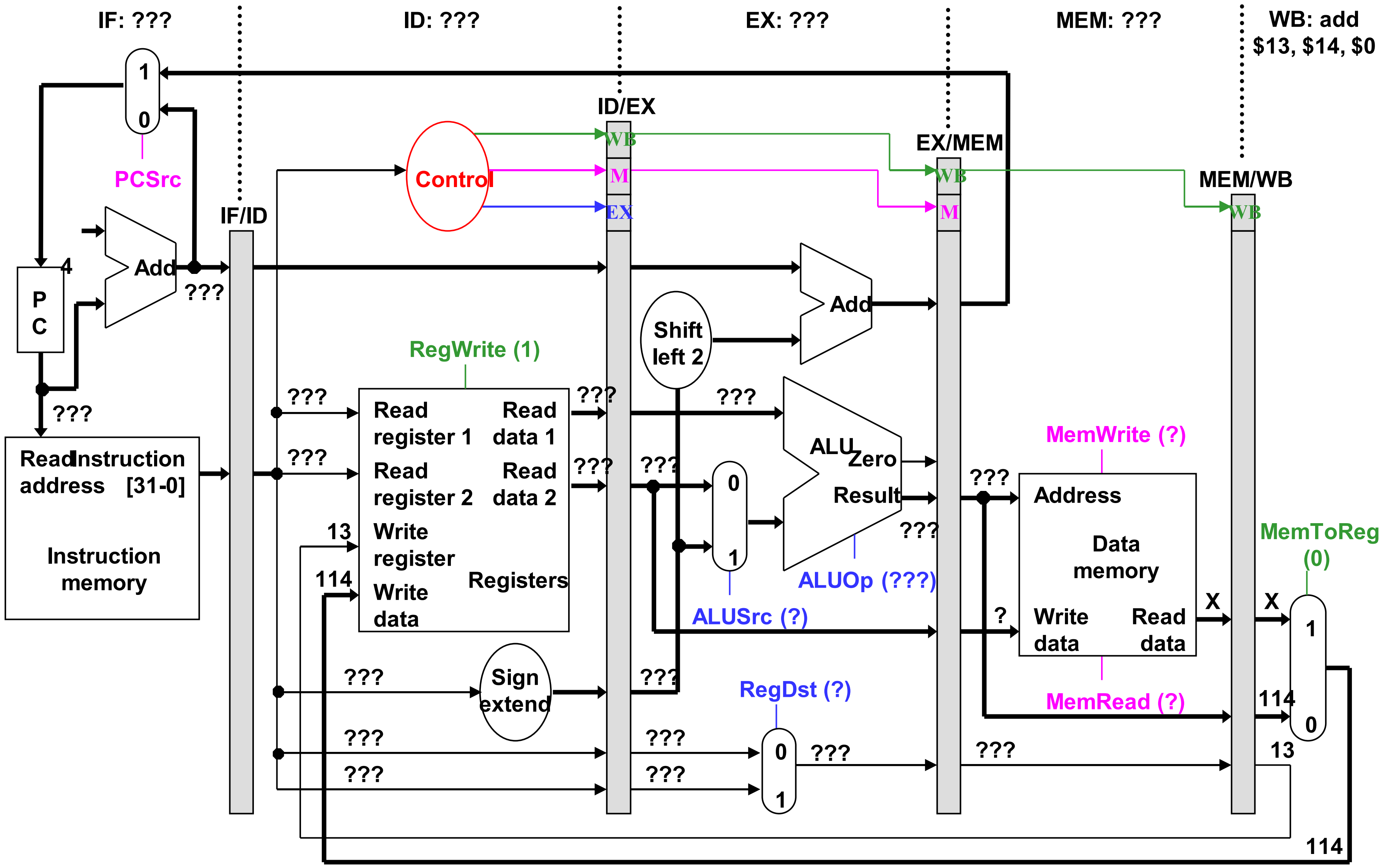
IF: ???
ID: ???
EX: add $13, $14, $0
MEM: or $16, $17, $18
WB: and $9, $10, $11
1
0
PCSrc
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
WB
M
WB
4
Add
???
P C
???
Read address
Instruction [31-0]
Instruction memory
RegWrite (1)
???
???
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
9
110
???
???
Shift left 2
Add
114
ALU
Zero
Result
0
0
1
ALUSrc (0)
ALUOp (add)
114
119
MemWrite (0)
Address
Data memory
Write data
Read data
118
MemRead (0)
X
X
MemToReg (0)
1
0
110
Sign extend
???
???
???
X
X
13
RegDst (1)
0
1
13
16
9
110

Cycle 8



IF: ???
ID: ???
EX: ???
MEM: add $13, $14, $0
WB: or $16, $17, $18
1
0
PCSrc
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
WB
M
WB
4
Add
???
P C
???
Read address
Instruction [31-0]
Instruction memory
RegWrite (1)
???
???
Read register 1
Read register 2
Read data 1
Read data 2
16
Write register
119
Write data
Registers
???
???
???
Shift left 2
Add
???
ALU
Zero
Result
???
0
1
ALUSrc (?)
ALUOp (???)
???
Sign extend
???
???
???
???
???
RegDst (?)
0
1
???
114
MemWrite (0)
Address
Data memory
0
Write data
Read data
MemRead (0)
X
X
MemToReg (0)
1
0
119
13
16
119

Cycle 9



IF: ???
ID: ???
EX: ???
MEM: ???
WB: add $13, $14, $0
1
0
PCSrc
IF/ID
ID/EX
EX/MEM
MEM/WB
Control
WB
M
EX
WB
M
WB
4
Add
???
P C
???
Read address
Instruction [31-0]
Instruction memory
RegWrite (1)
???
???
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
13
114
???
???
???
???
Shift left 2
Add
???
ALU
Zero
Result
0
1
ALUSrc (?)
ALUOp (???)
???
???
MemWrite (?)
Address
Data memory
Write data
Read data
MemRead (?)
?
X
X
MemToReg (0)
1
0
114
13
114
???
Sign extend
???
???
???
???
???
RegDst (?)
0
1
???
???

That’s a lot of diagrams there

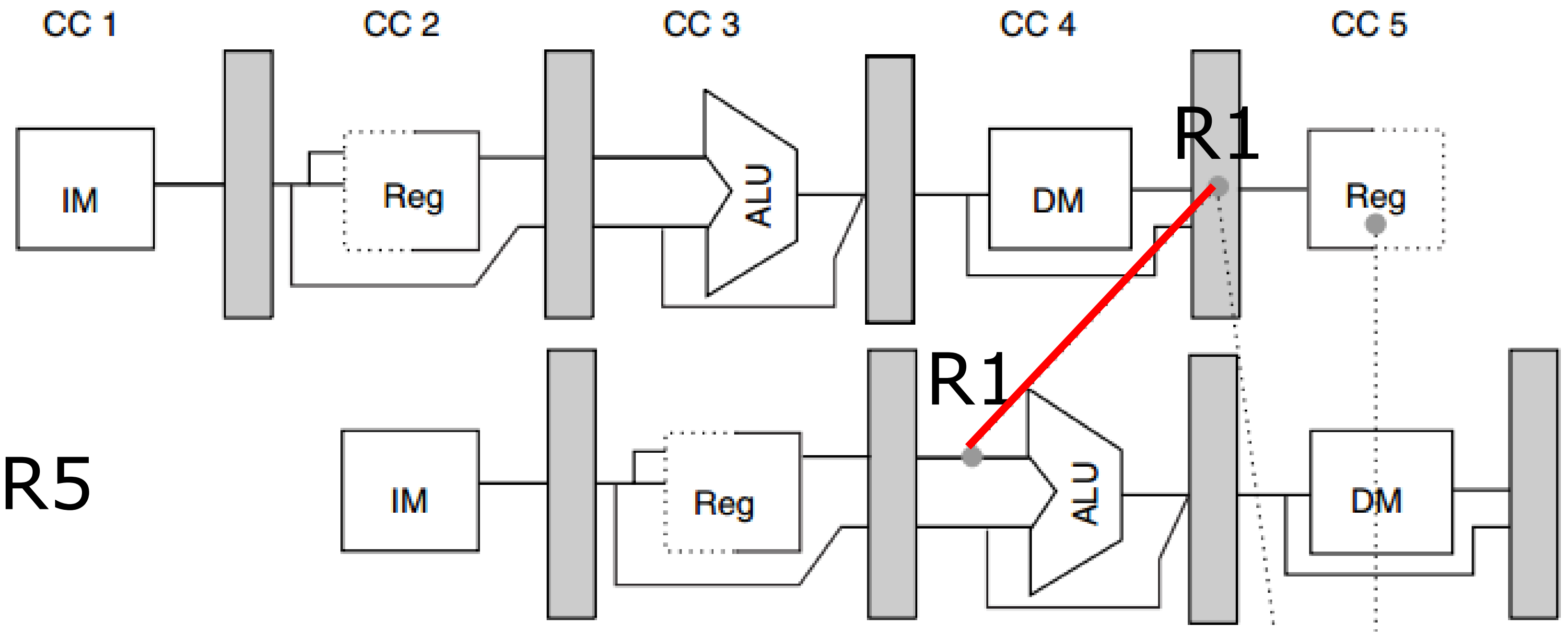
		Clock cycle								
		1	2	3	4	5	6	7	8	9
lw	$t0, 4($sp)	IF	ID	EX	MEM	WB				
sub	$v0, $a0, $a1		IF	ID	EX	MEM	WB			
and	$t1, $t2, $t3			IF	ID	EX	MEM	WB		
or	$s0, $s1, $s2				IF	ID	EX	MEM	WB	
add	$t5, $t6, $0					IF	ID	EX	MEM	WB

– You can see how instruction executions are overlapped
– Each functional unit is used by a *different* instruction in each cycle
– The pipeline registers save control and data values generated in previous clock cycles for later use
– When the pipeline is full in clock cycle 5, all of the hardware units are utilized. This is the ideal situation, and what makes pipelined processors so fast

When Pipeline Is Stuck

LD R1, 0(R2)

SUB R4, R1, R5



Actually, in many cases, instructions cannot be fully pipelined.
In this example, R1 is ready by load instruction at the end of clock cycle 4;
But subtract instruction requires R1 at the beginning of clock cycle 4; in this case, the processor can not fully pipeline the two instructions as we expect.

References

- Computer System Architecture (3rd Edition) by M. Morris Mano
- Computer organization and architecture designing for performance (8th edition) by William Stallings
- Computer Architecture: A Quantitative Approach 5th Edition by John L. Hennessy
- Patterson & Hennessy, “Computer Organization &Design”
- Lecture Notes, Lecture 4: Pipelining Basics & Hazards, Kai Bu, kaibu@zju.edu.cn
- EmbeededNotes Pipeline , https://www.mediafire.com/file/yl157ng7eby7989/Video_6___-_PipeLine_-_1_Intro.pptx/file
- https://www.youtube.com/watch?v=X84uO3H83bA
- Lecture Notes, Introduction to Computer Architecture, at the University of California, Santa Barbara. © Behrooz Parhami

Thanks