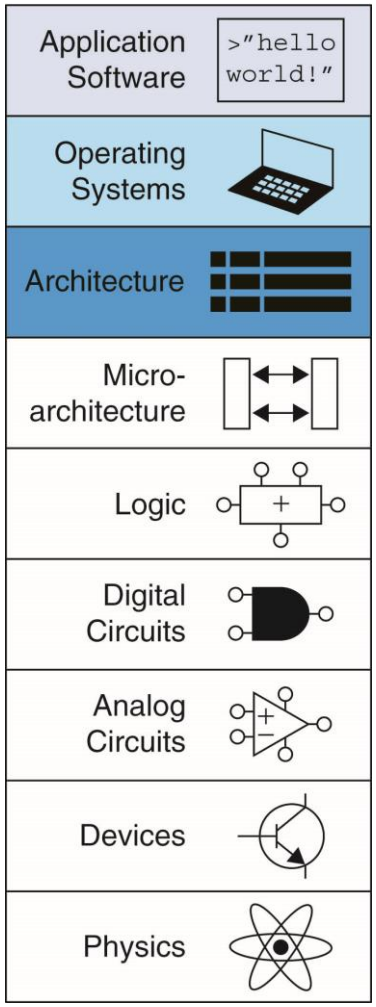
MICROARCHITECTURE

Microarchitecture

MIPS

Computer Architecture/Microarchitecture

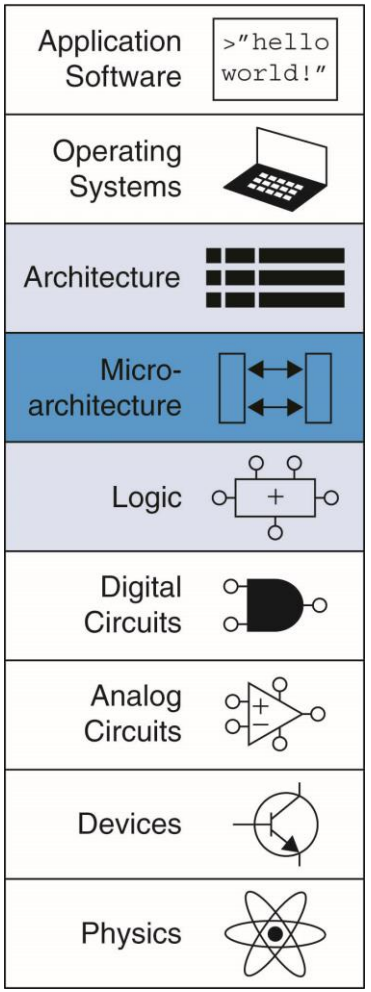


Architecture: the programmer's view of the computer

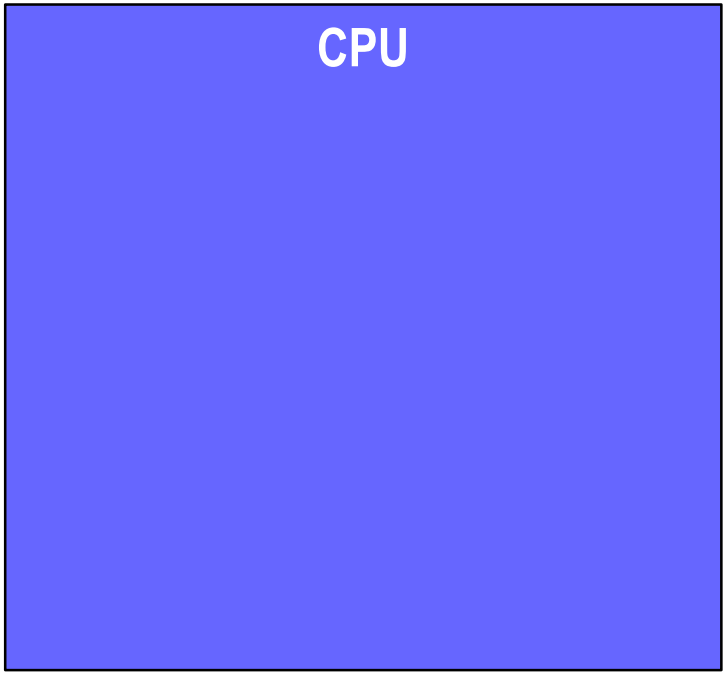
- Defined by instructions (operations) and operand locations

Microarchitecture: how to implement an architecture in hardware

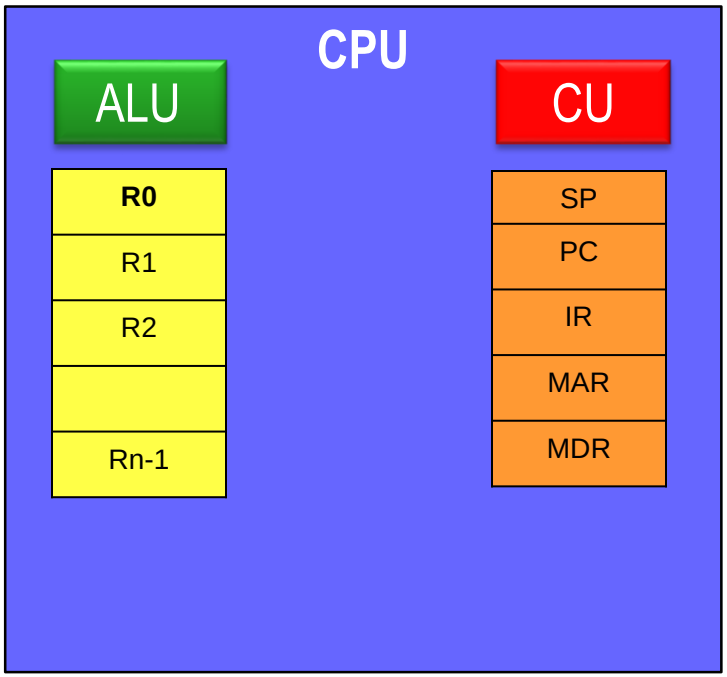
- The microarchitecture is built out of "logic" circuits and memory elements



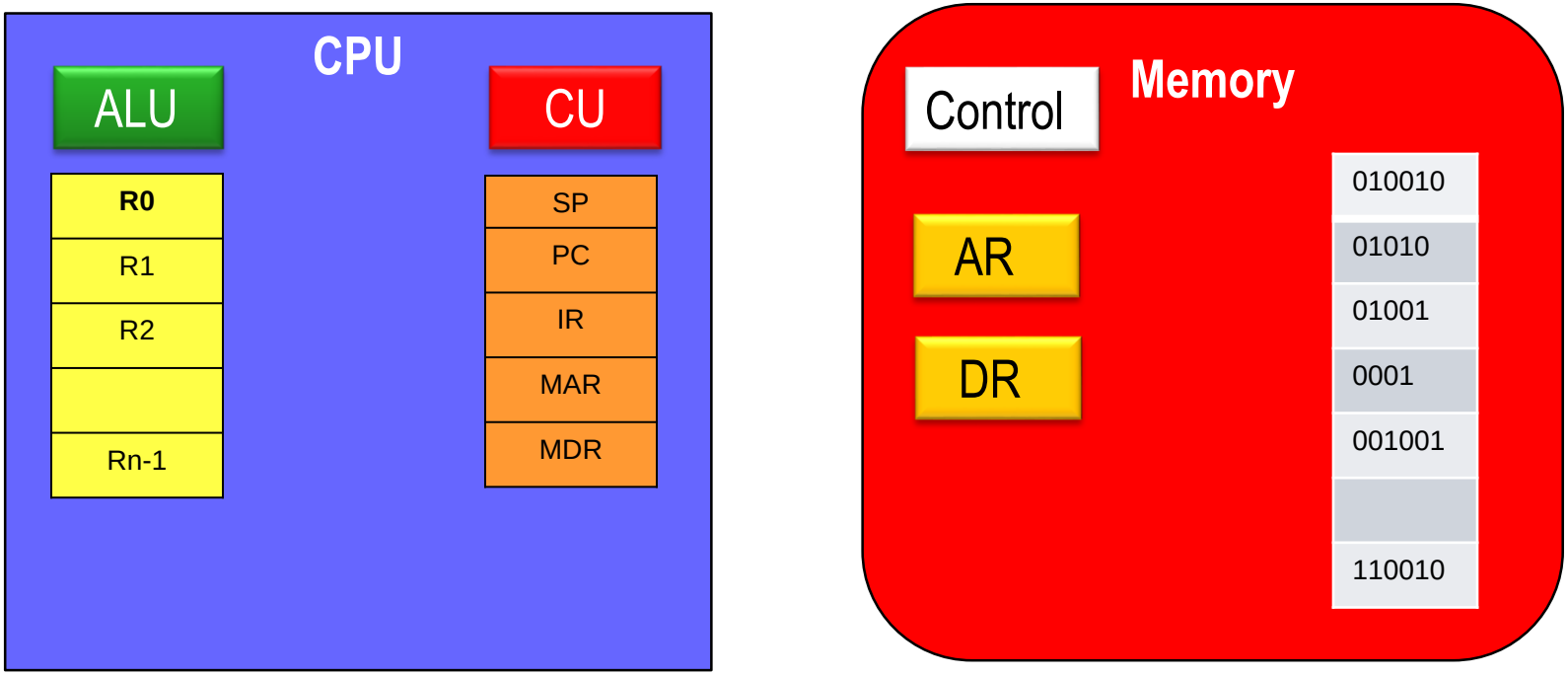
Computer System Architecture (CPU)



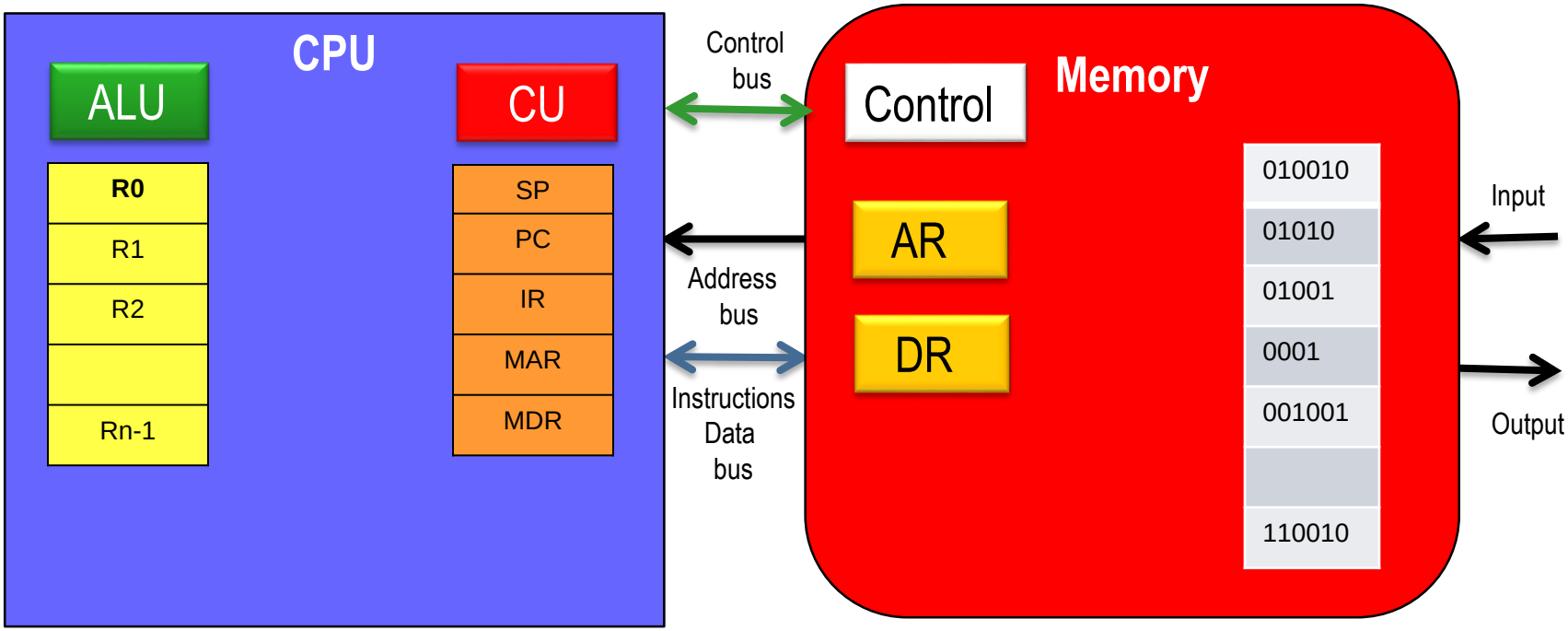
Computer System Architecture (CPU)



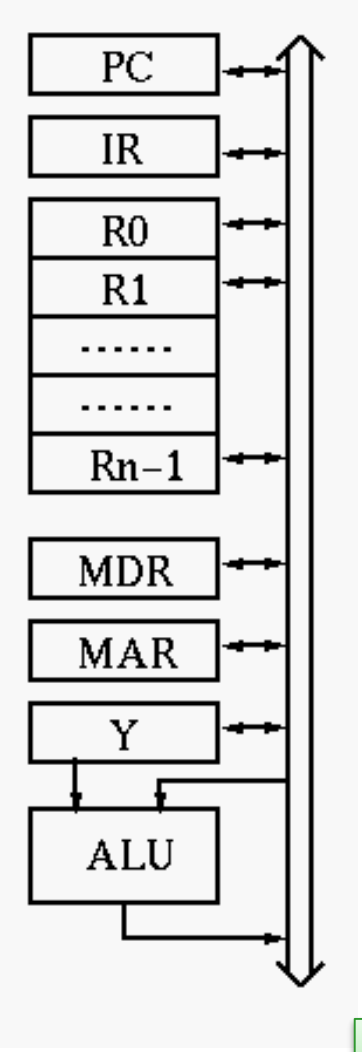
Computer System Architecture (CPU+M)

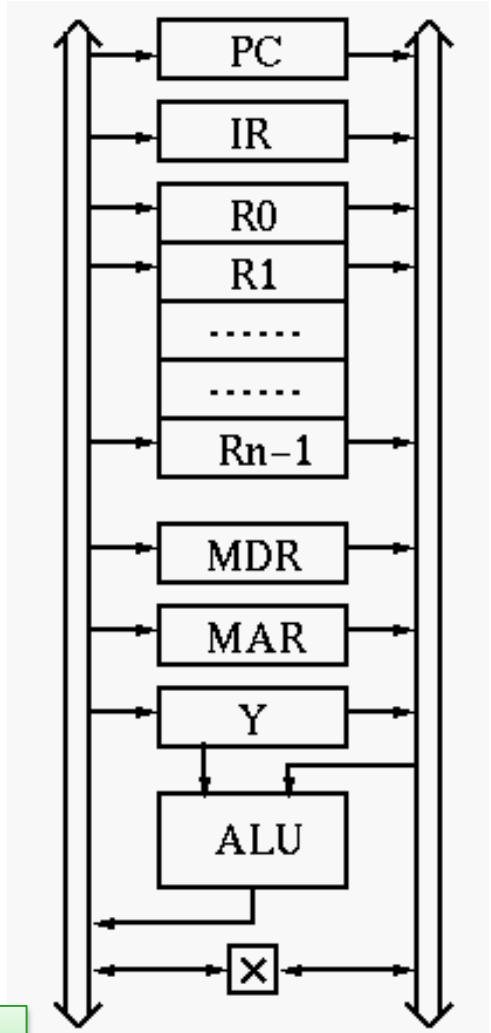


Computer System Architecture

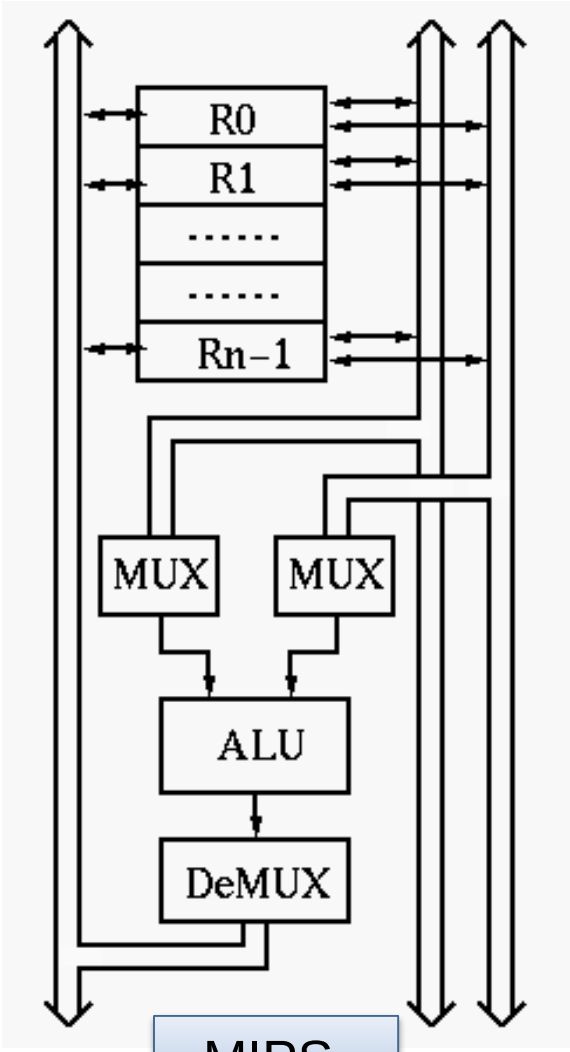


Buses



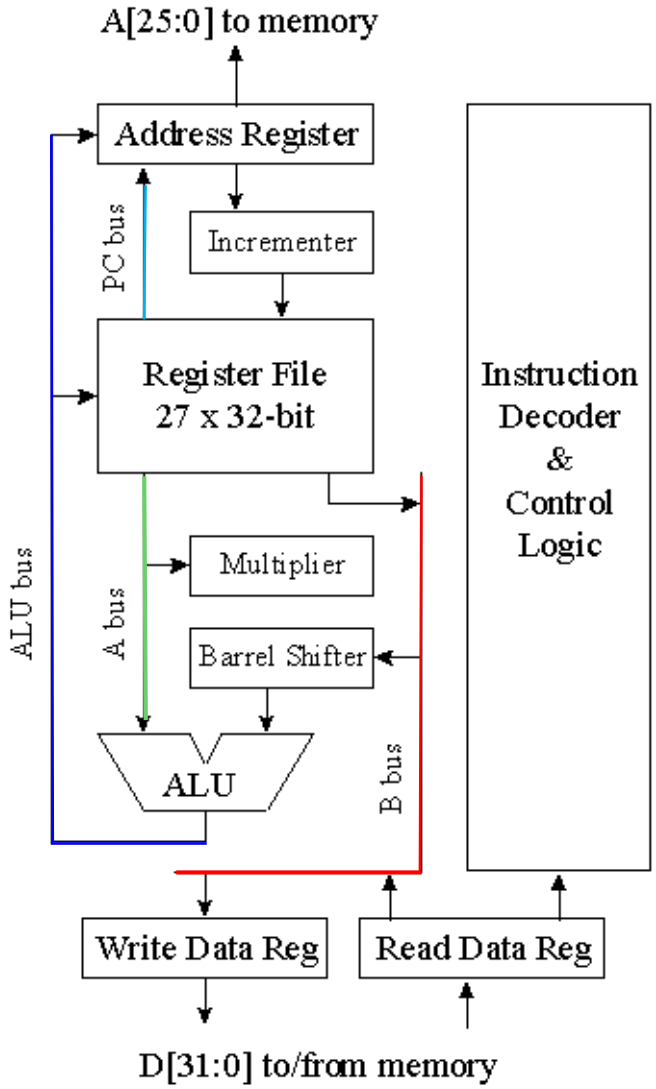


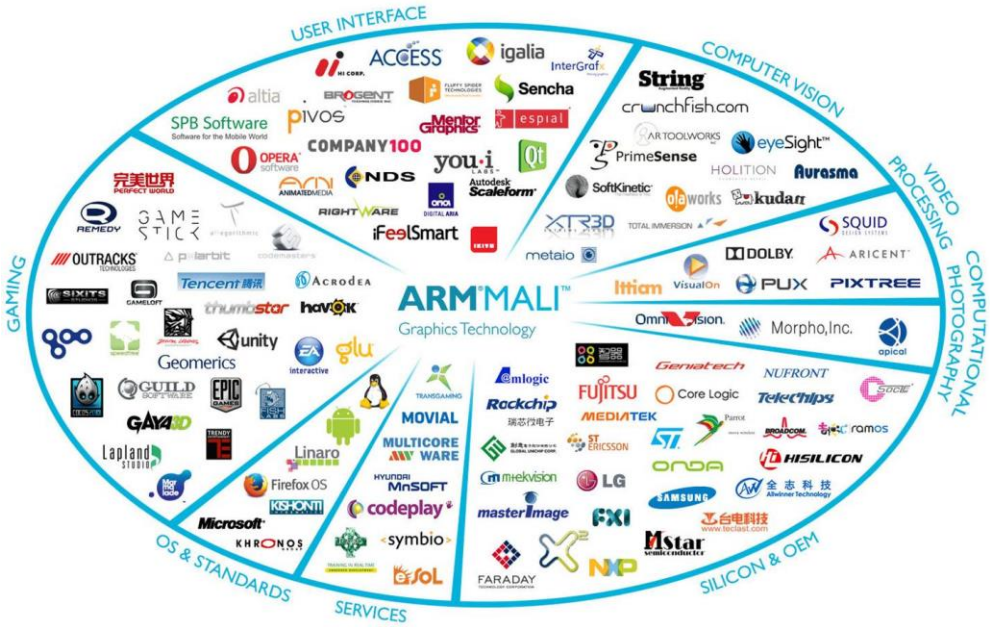
M68000



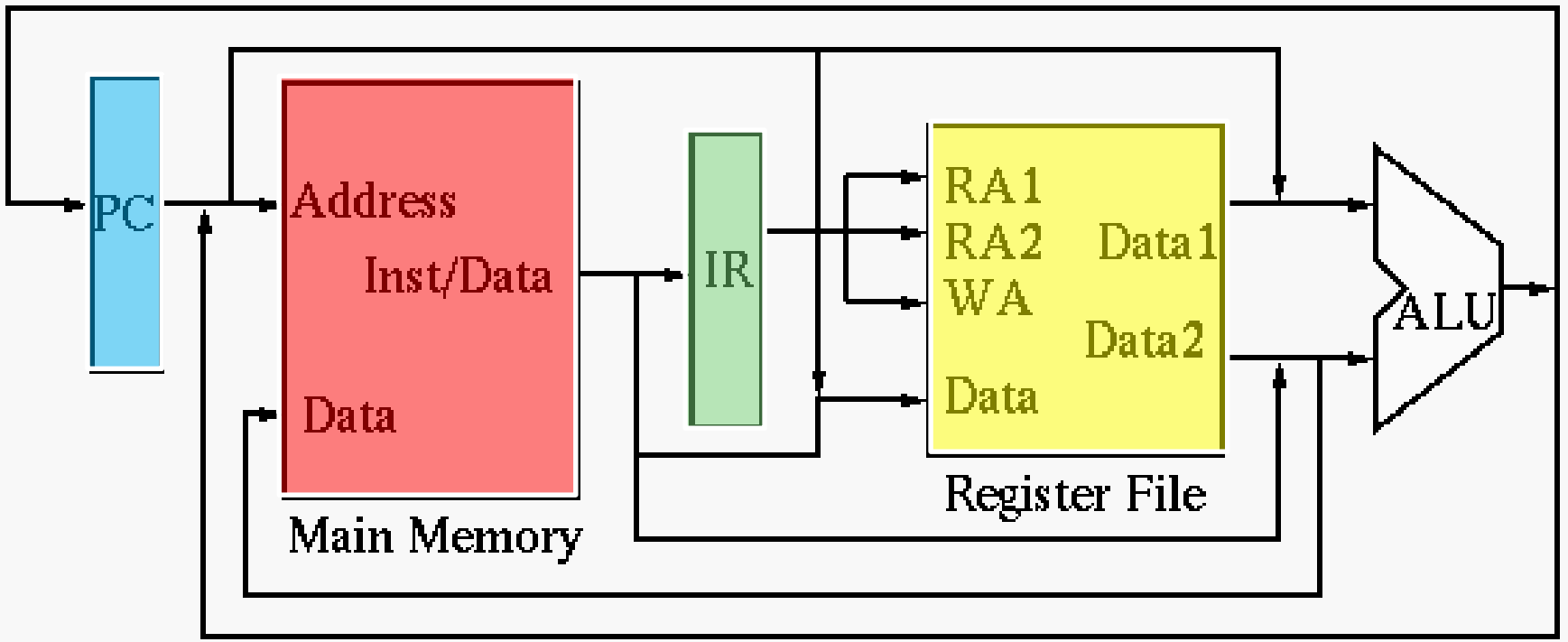
MIPS

ARM2 (1996) Datapath





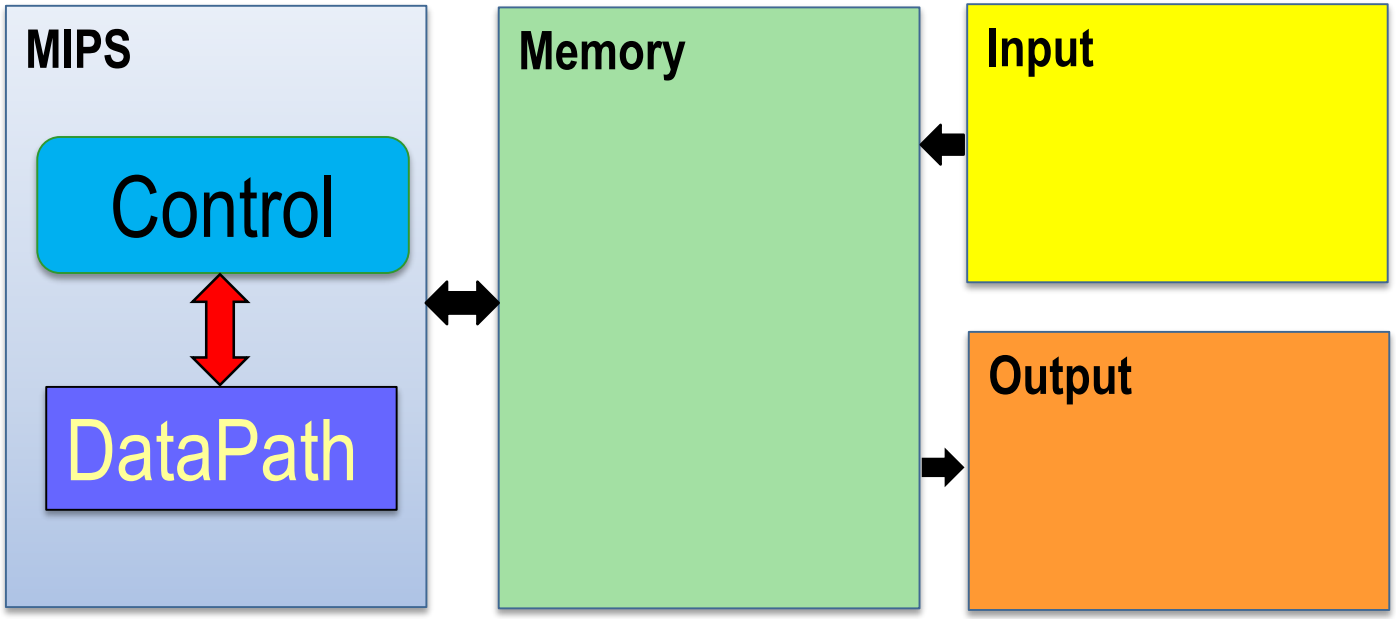
MIPS Architecture



Components of the MIPS System

- DataPath (Registers … ALU)
  - Major functional hardware components (connected with buses)
- Control Unit
  - Controls the hardware components of the dataPath
  - Control signals (sent to hardware components)
  - Condition signals (receives from the components)
  - Selects the MUX lines to switch between buses
- Memory
- I/O

Components of the MIPS System

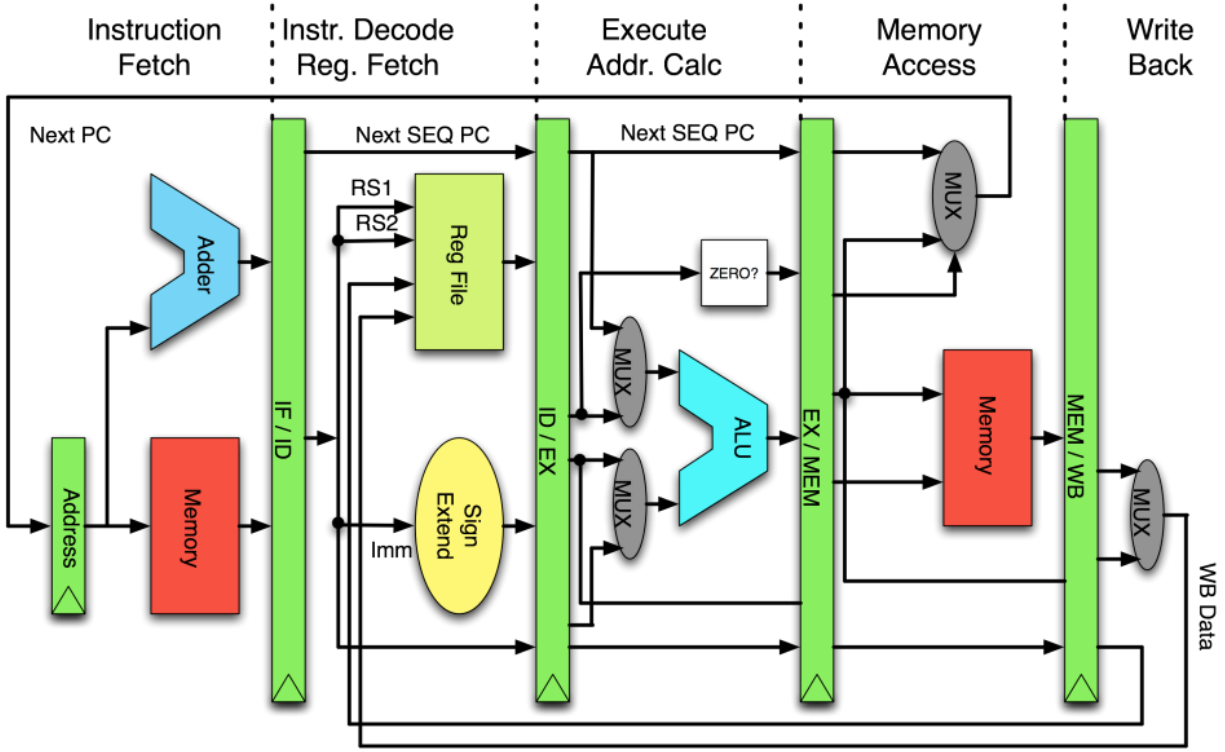


DataPath components and Memory

- Program Counter (PC)
- Instruction-Register (IR)
- Register-File (RegF)
- Arithmetic Logic Unit (ALU)
- Memory
  - Instruction memory (IM)
    - Stores the machine-code program
  - Data memory (DM)
    - Stores data

DataPath and Control Unit

- DataPath; It is the path of instructions and data in the Microprocessor
- DataPath hardware follows the cycle:
  1. Fetch Instruction
  2. Decode
  3. Execute
  4. Memory Access
  5. Write-back
- Control Unit
  - Controls the DataPath hardware



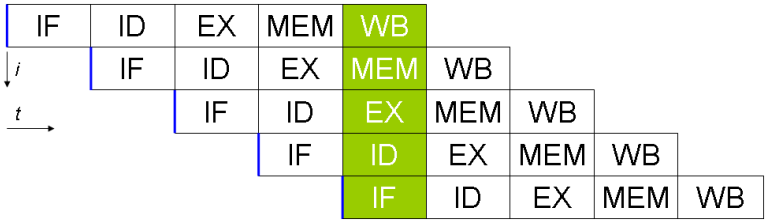
MIPS Processor

Microprocessor without **I**nterlocked **P**ipeline **S**tages

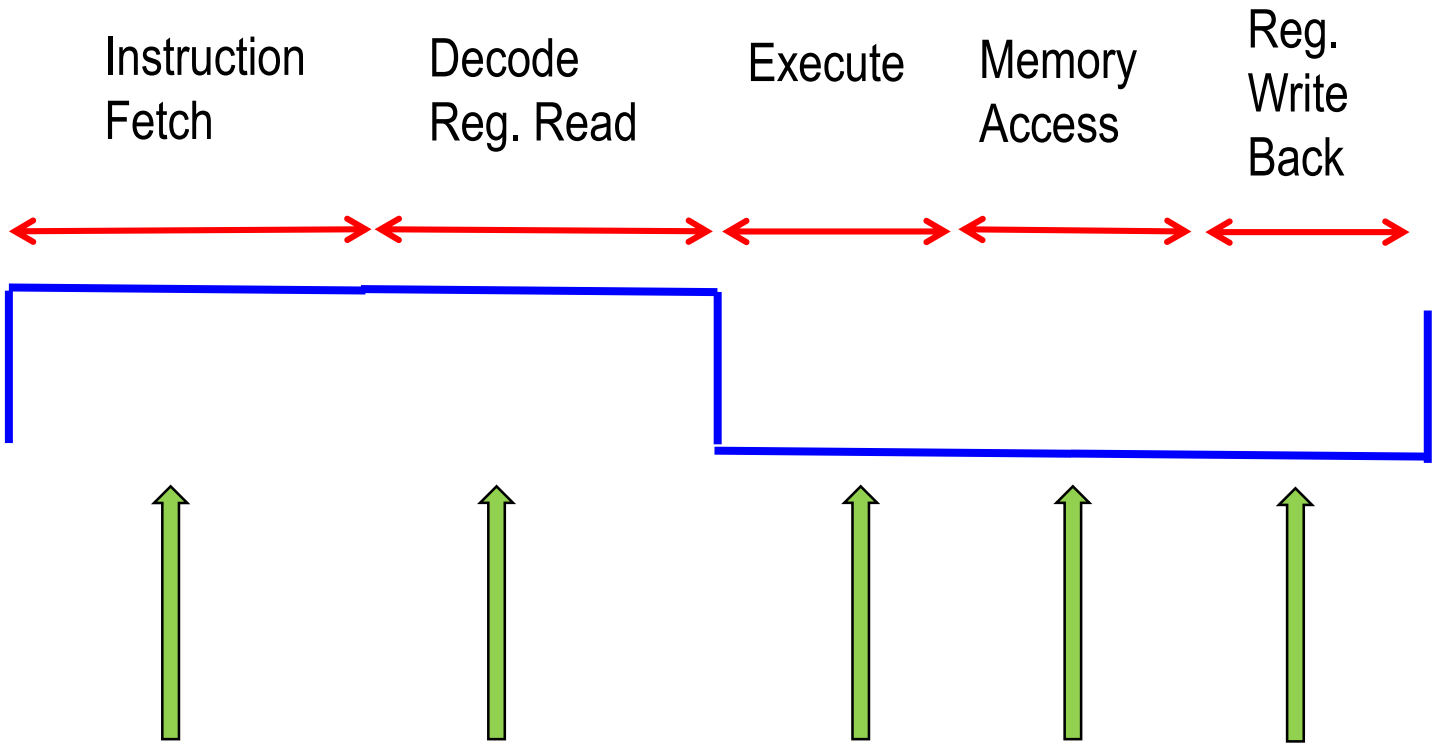
5 Cycles

Microarchitecture

- Single-Cycle: each instruction executes in a single CPU cycle
- Multiple-Cycle: each instruction is broken into series of shorter steps
- Pipelined: each instruction broken up into series of steps & multiple instructions execute at once:

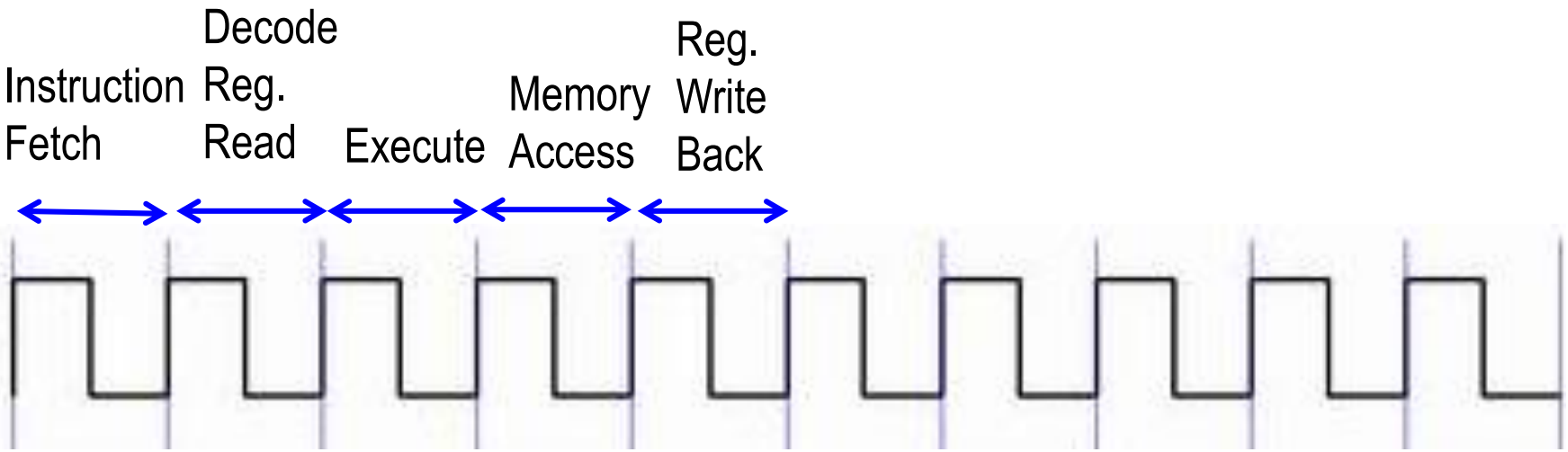


Single Cycle CPU



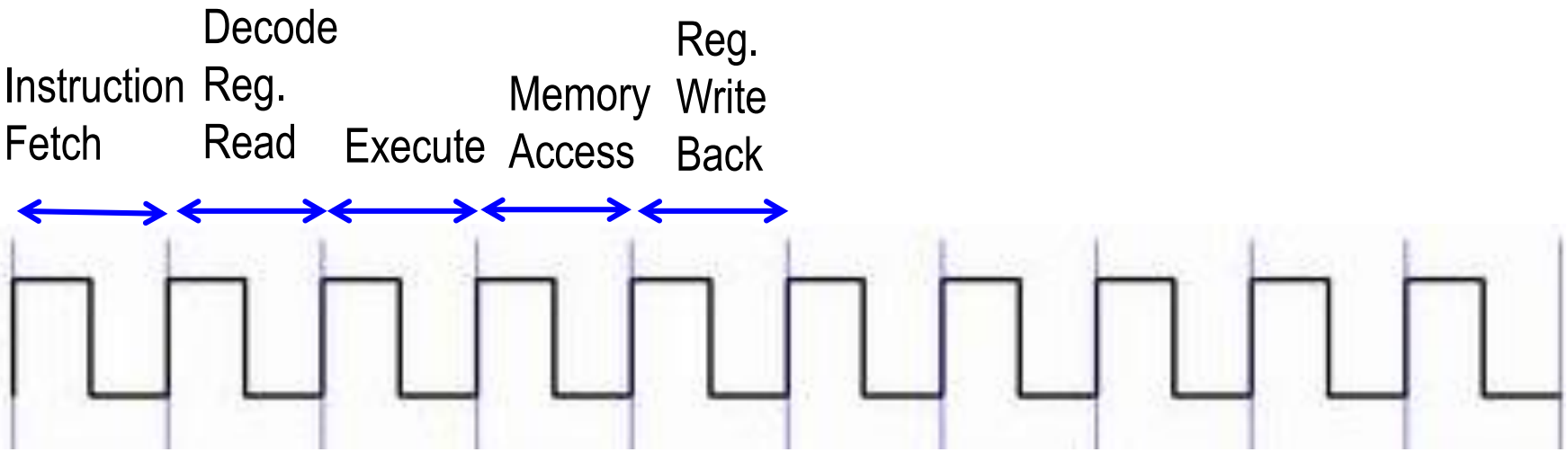
Each instruction executes in a single cycle

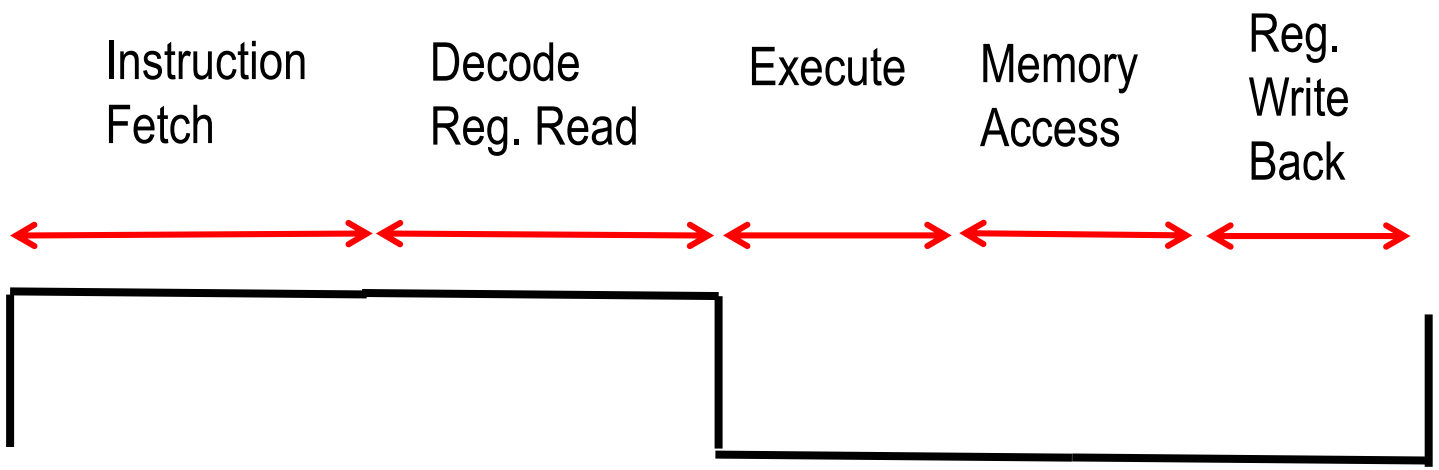
Multiple Cycle



Only one stage of instruction per clock cycle

Multiple and Single Cycle

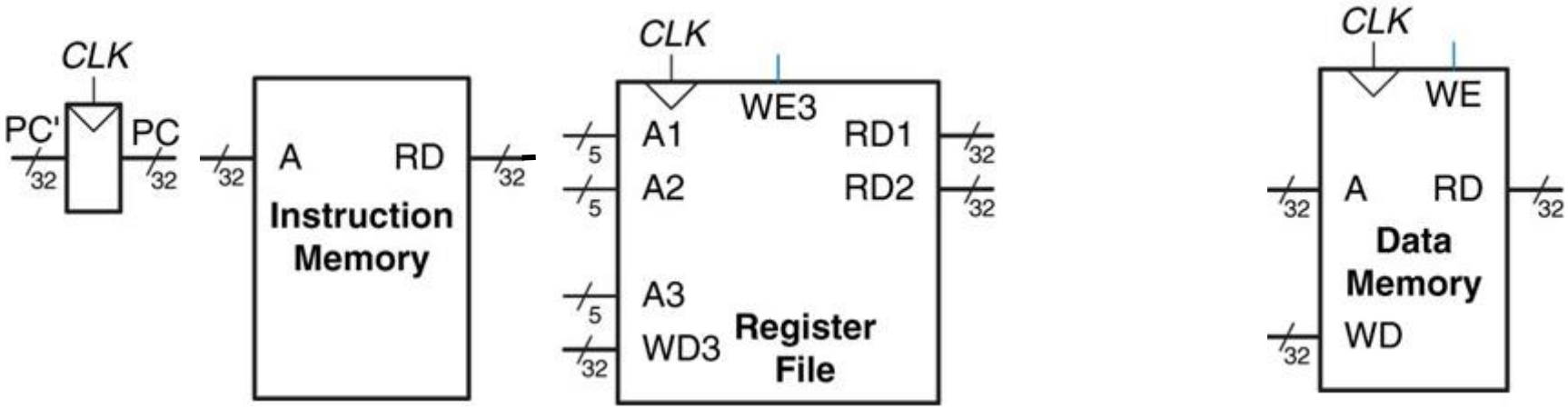




Combinational and State logic elements

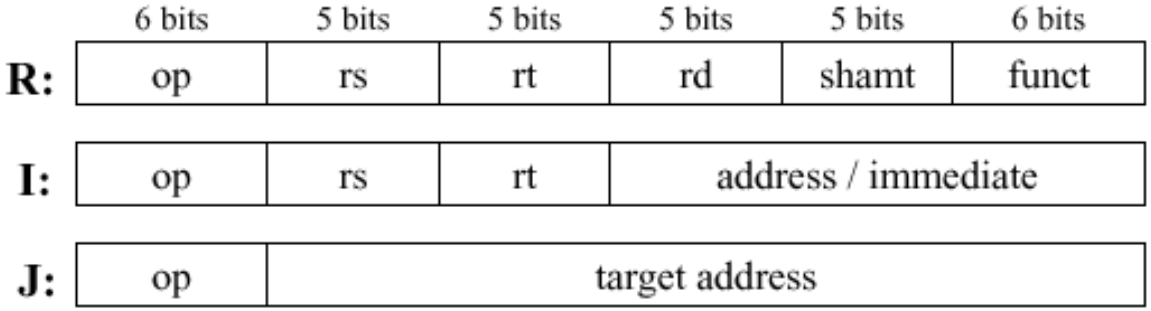
- Combinational
  - $\mathcal{F}$ [logic inputs]
  - (gates)
- State
  - $\mathcal{F}$ [logic Inputs, time (state)]
  - (flip-flops)

State elements of MIPS Processor



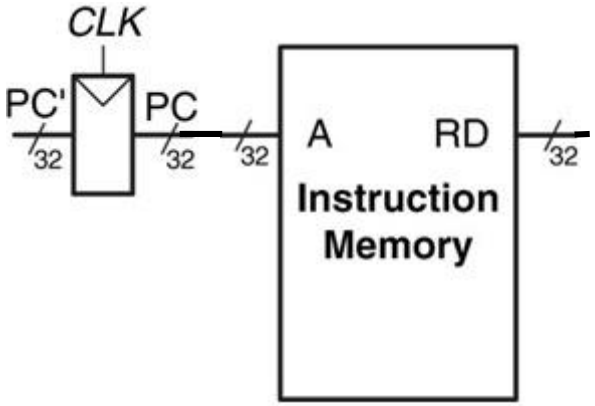
MIPS Instructions types

- R-Type: Register operands
- I-Type: Immediate operand
- J-Type: for Jumping



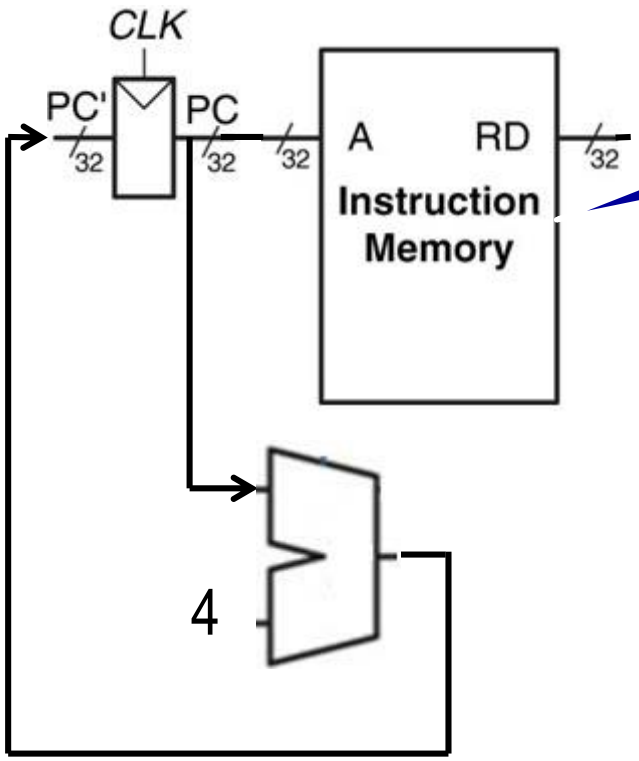
op: basic operation of the instruction (opcode)
rs: first source operand register
rt: second source operand register
rd: destination operand register
shamt: shift amount
funct: selects the specific variant of the opcode (function code)
address: offset for load/store instructions ($+/-2^{15}$)
immediate: constants for immediate instructions

Instruction Fetch



Program Counter (PC) Register stores the address of the next instruction

Instruction Fetch



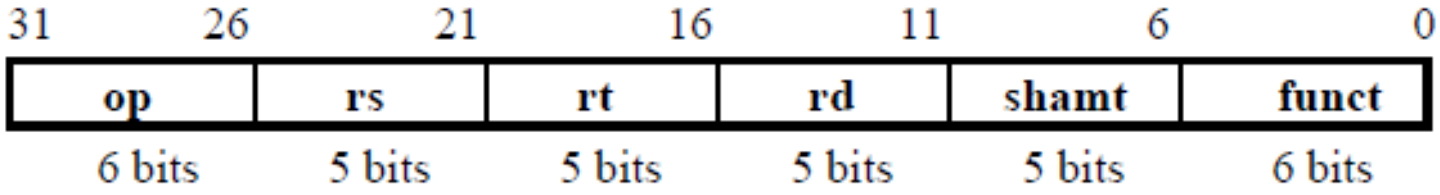
Instruction: [Instruction Register: Holds the instruction currently being processed (decoded)]

```
 3                .data
 4  Array:        .space 8
 5                .text
 6                #-------------
 7                la $t0, Array
 8                #-------------
 9                li $t1, 5
10                sw $t1, 0($t0)
11                li $t1, 6
12                sw $t1, 4($t0)
13                #-------------
14                lw $t2, 0($t0)
15                move $a0, $t2
16                li $v0, 1
17                syscall
18
19                lw $t3, 4($t0)
20                move $a0, $t3
21                li $v0, 1
22                syscall
23                #-------------
24                li $v0, 10
25                syscall
26
```

The address of the next instruction is: PC + 4 (32/8 = 4)

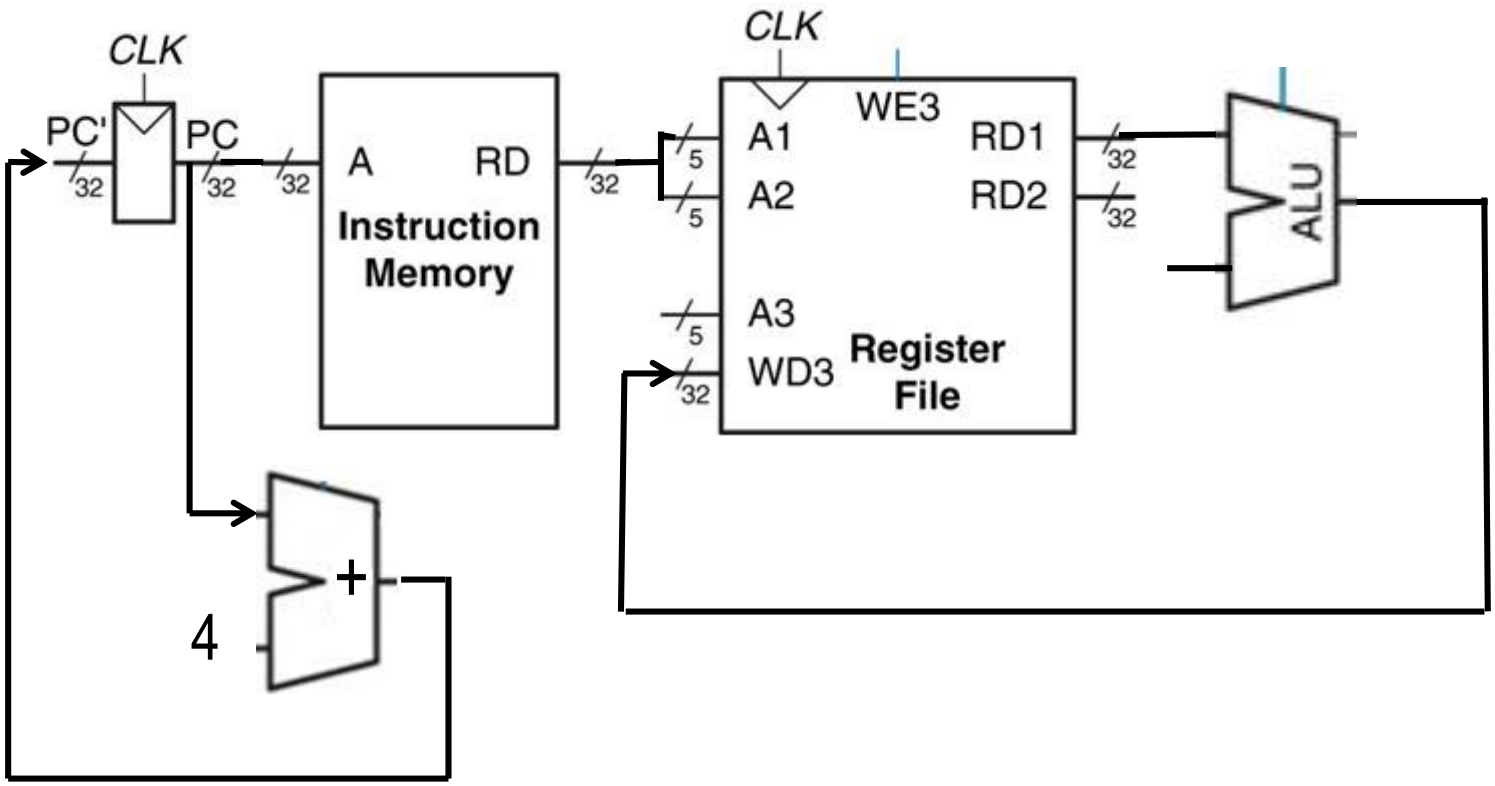
R-Type Arithmetic/Logic Instructions

6 Fields	Opcode	src reg 1	src reg 2	dst reg	shamt	funct
# bits	6 (31-26)	5 (25-21)	5 (20-16)	5 (15-11)	5 (10-6)	6 (5-0)



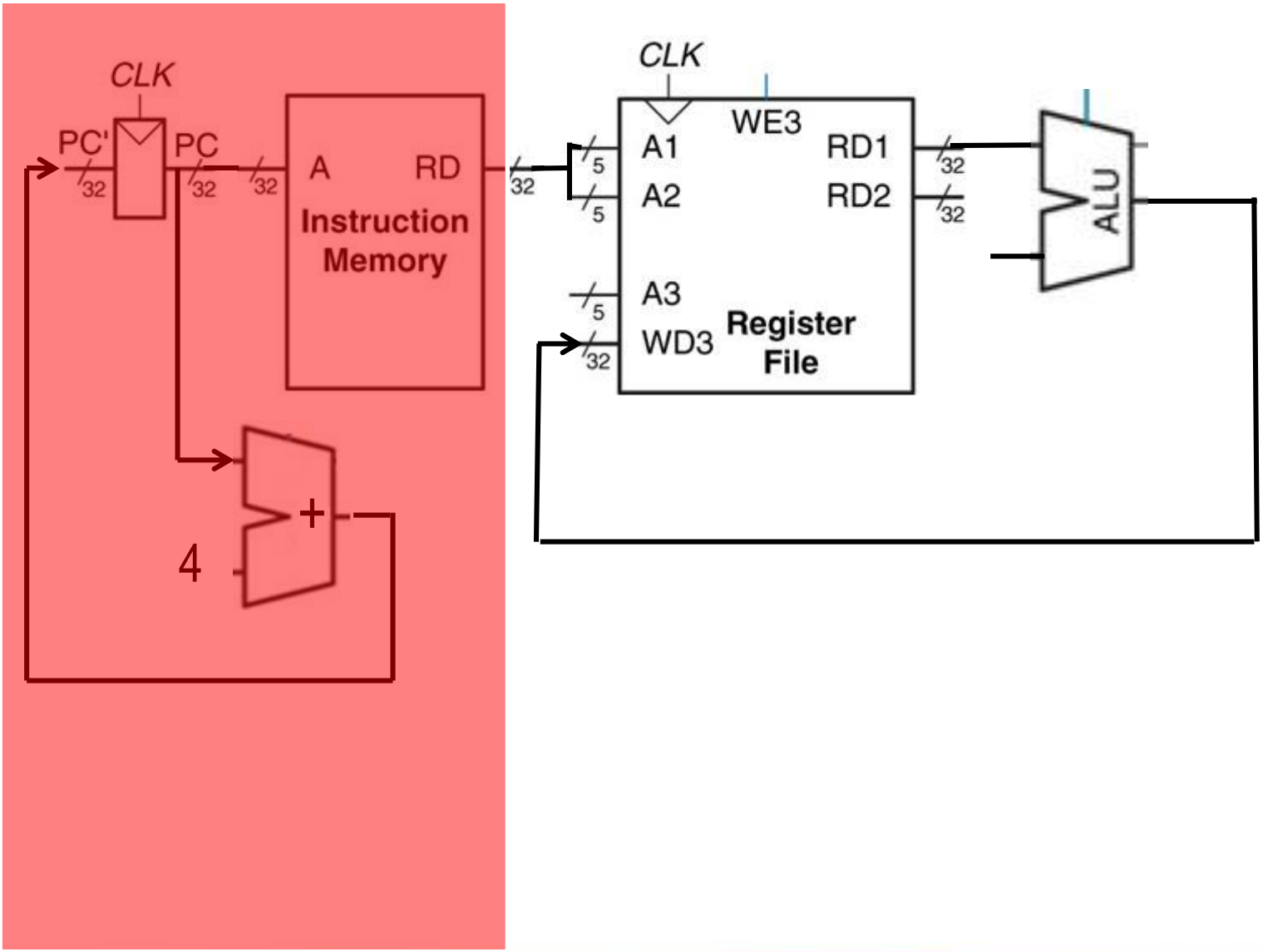

```
add rd, rs, rt        # r[d] = r[s] + r[t]
sub rd, rs, rt        # r[d] = r[s]  - r[t]
```

R-Type Arithmetic/Logic Operations

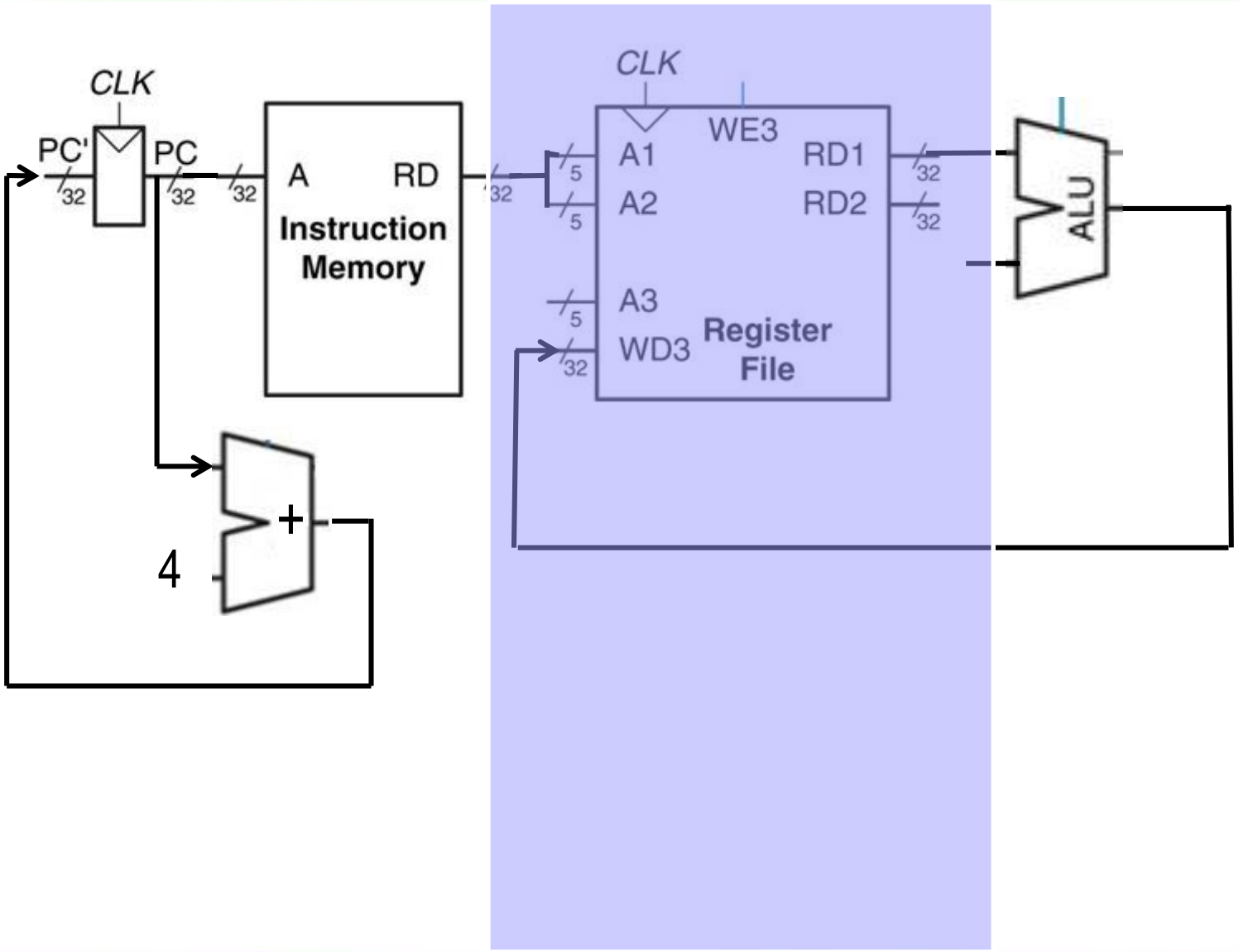



```
add rd, rs, rt
sub rd, rs, rt
```

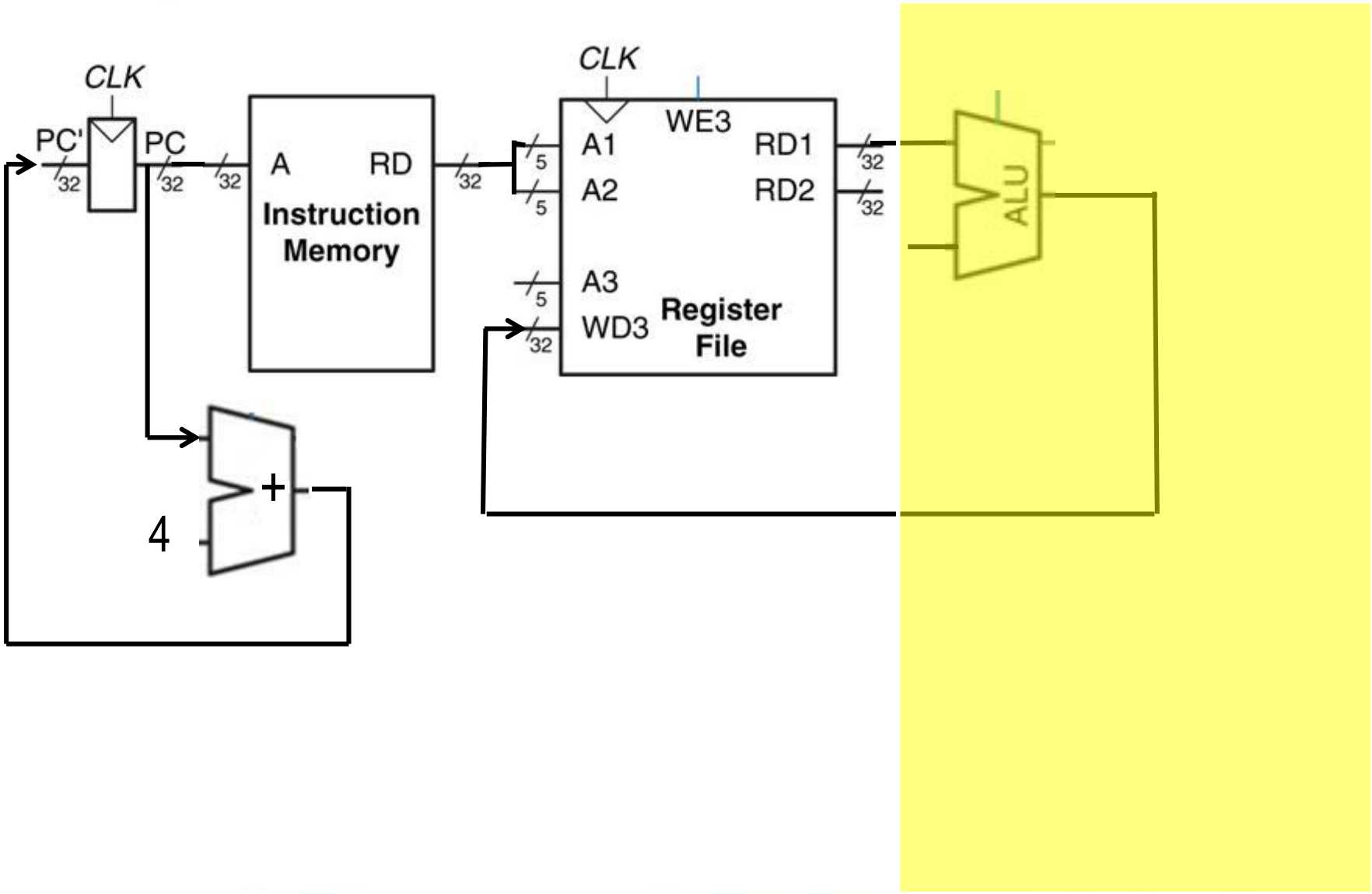
Fetch


CLK
PC'
PC
32
32
32
A
RD
Instruction
Memory
4
+
CLK
WE3
A1
A2
A3
WD3
RD1
RD2
5
5
5
32
32
32
32
Register
File
ALU

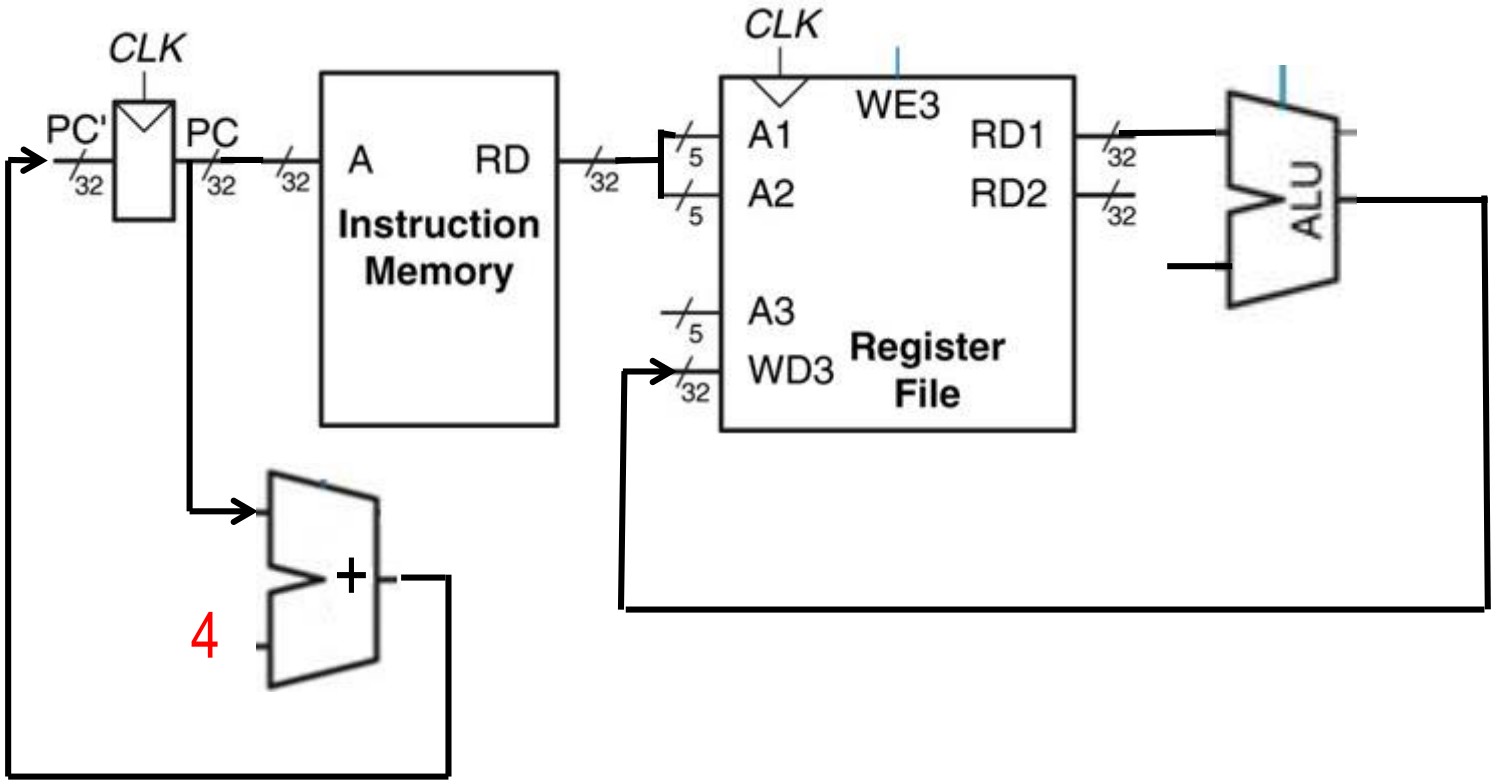
Decode


CLK
PC'
PC
32
32
32
A
RD
Instruction
Memory
4
+
CLK
WE3
A1
A2
A3
WD3
RD1
RD2
5
5
5
32
32
32
32
Register
File
ALU

Execute


CLK
PC'
PC
32
32
32
A
RD
Instruction Memory
32
CLK
WE3
A1
A2
A3
WD3
RD1
RD2
Register File
5
5
5
32
32
32
ALU
+
4

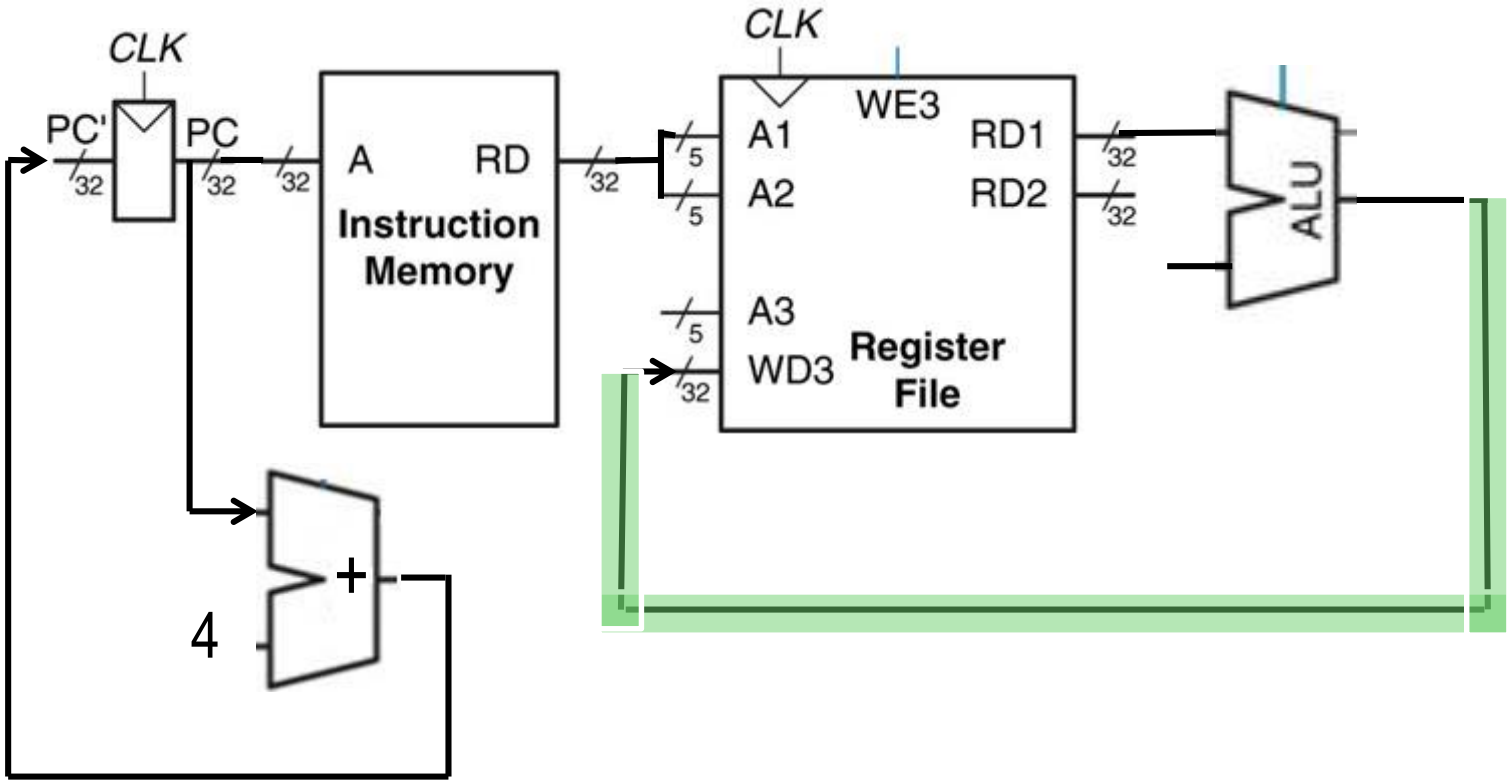
No Memory access



`add rd, rs, rt`

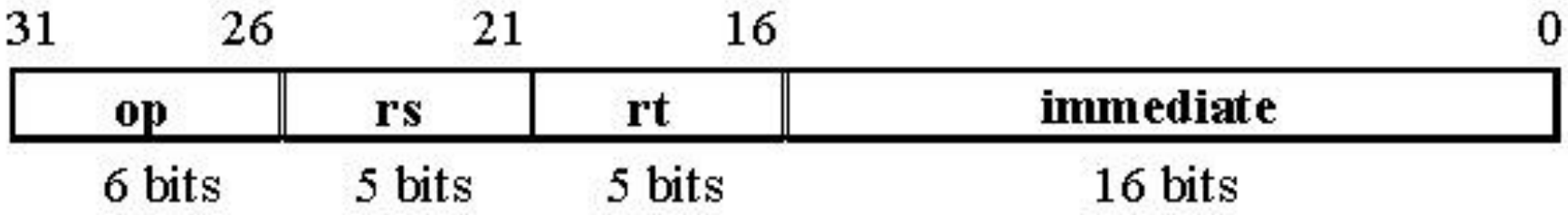
- mem[PC] — Fetch the instruction from memory
- R[rd]: R[rs] + R[rt] — The actual operation
- PC: PC + 4 — Calculate the next instruction's address

Write Back to Register-File

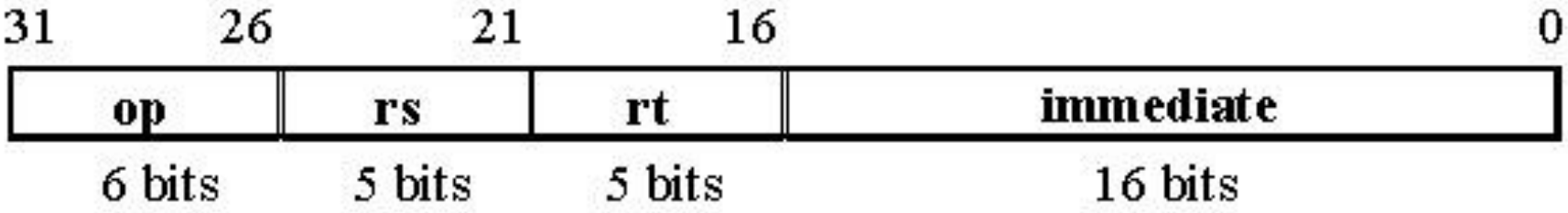


I-type: Load and Store instructions

- **Load word from Memory:**
  - `lw rt,rs, imm16` `# rt = Memory[rs+offset]`
  - `lw rt, imm16(rs)`
- **Store word in Memory:**
  - `sw rt,rs, imm16` `# Memory[rs+offset] = rt`
  - `sw rt,imm16(rs)`

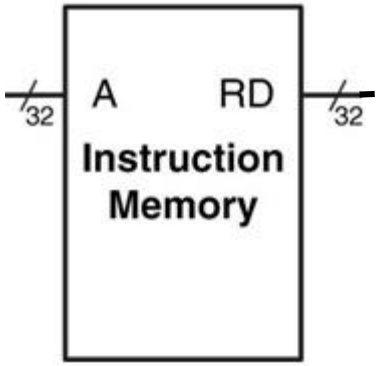


lw rt,rs, imm16

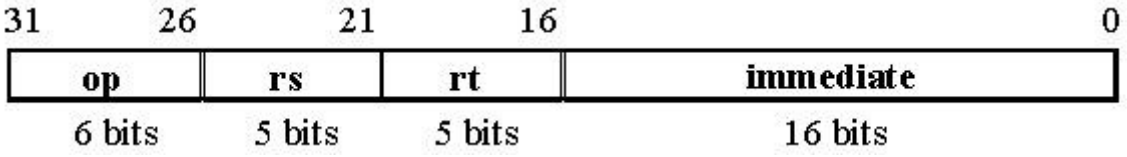


Load (Read) data from memory and write it back to Register-File

Single-Cycle Datapath

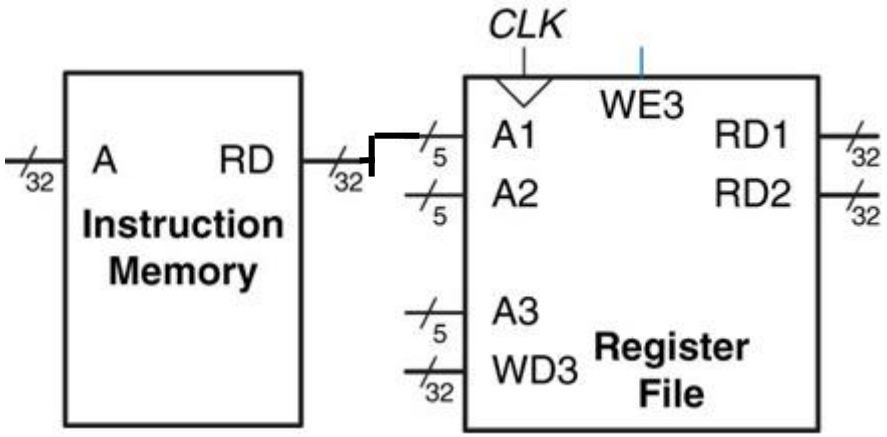


`lw rt,rs, imm16`



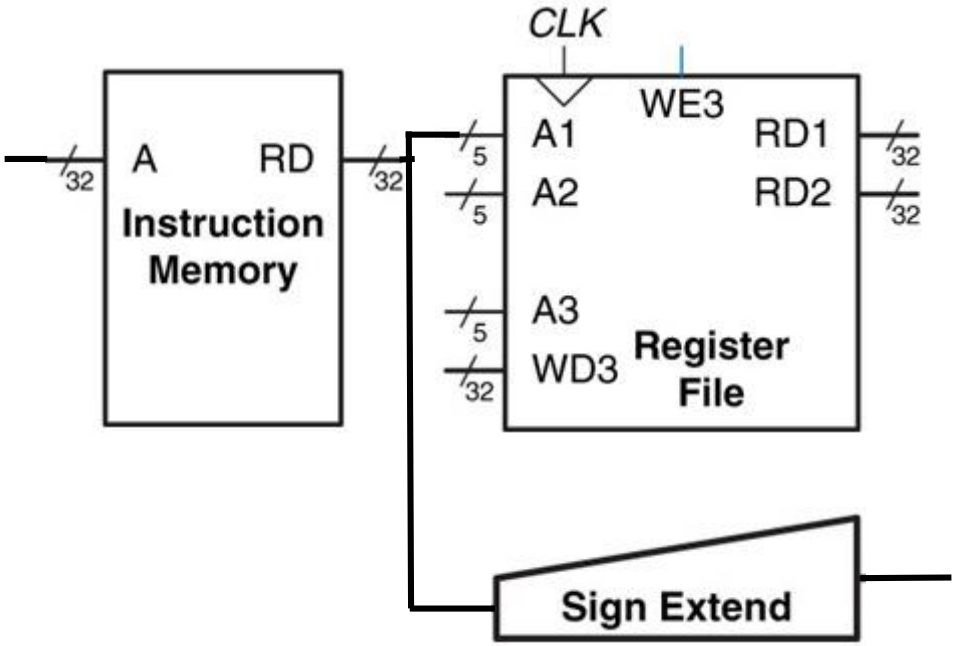
Fetch Instruction (`lw`)

Single-Cycle Datapath

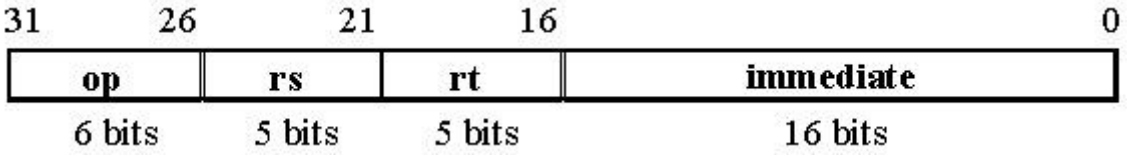


Read source operands from the Register-File (Register read)

Single-Cycle Datapath

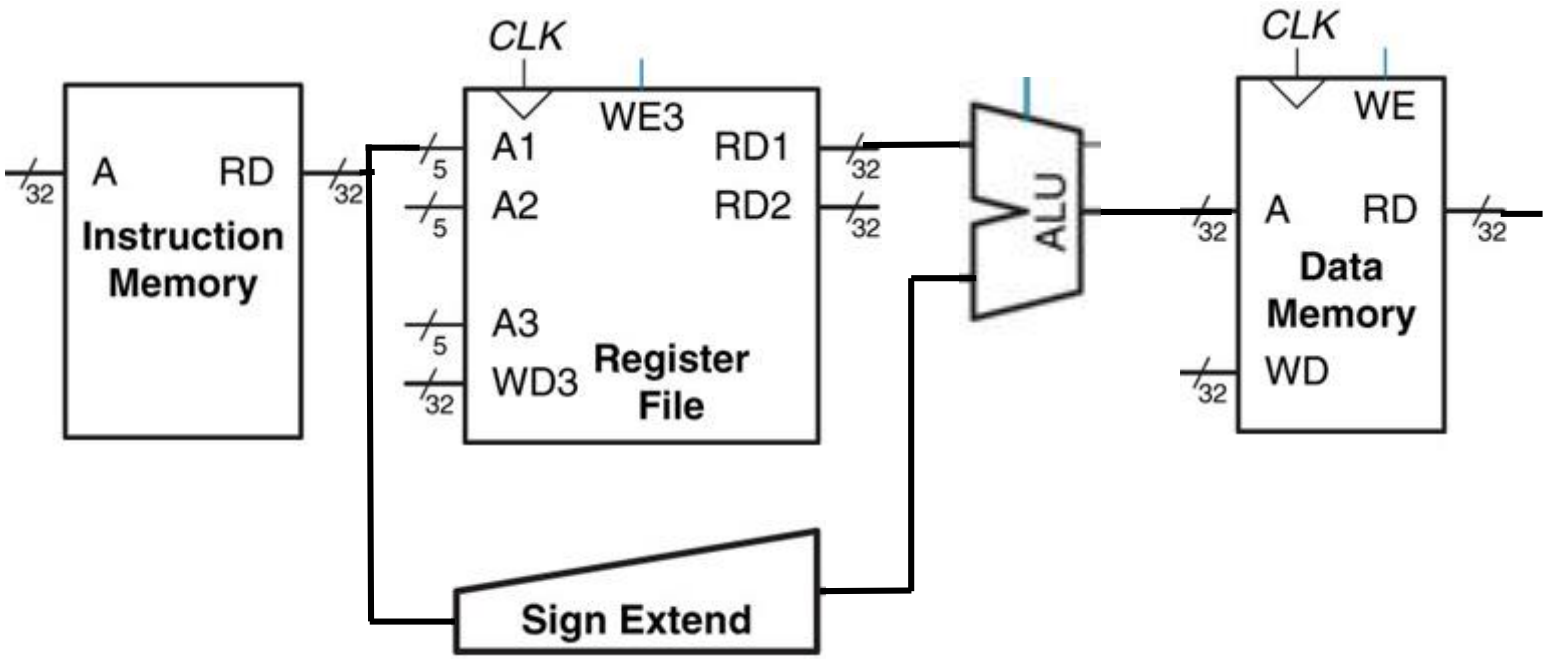


`lw rt,rs, imm16`



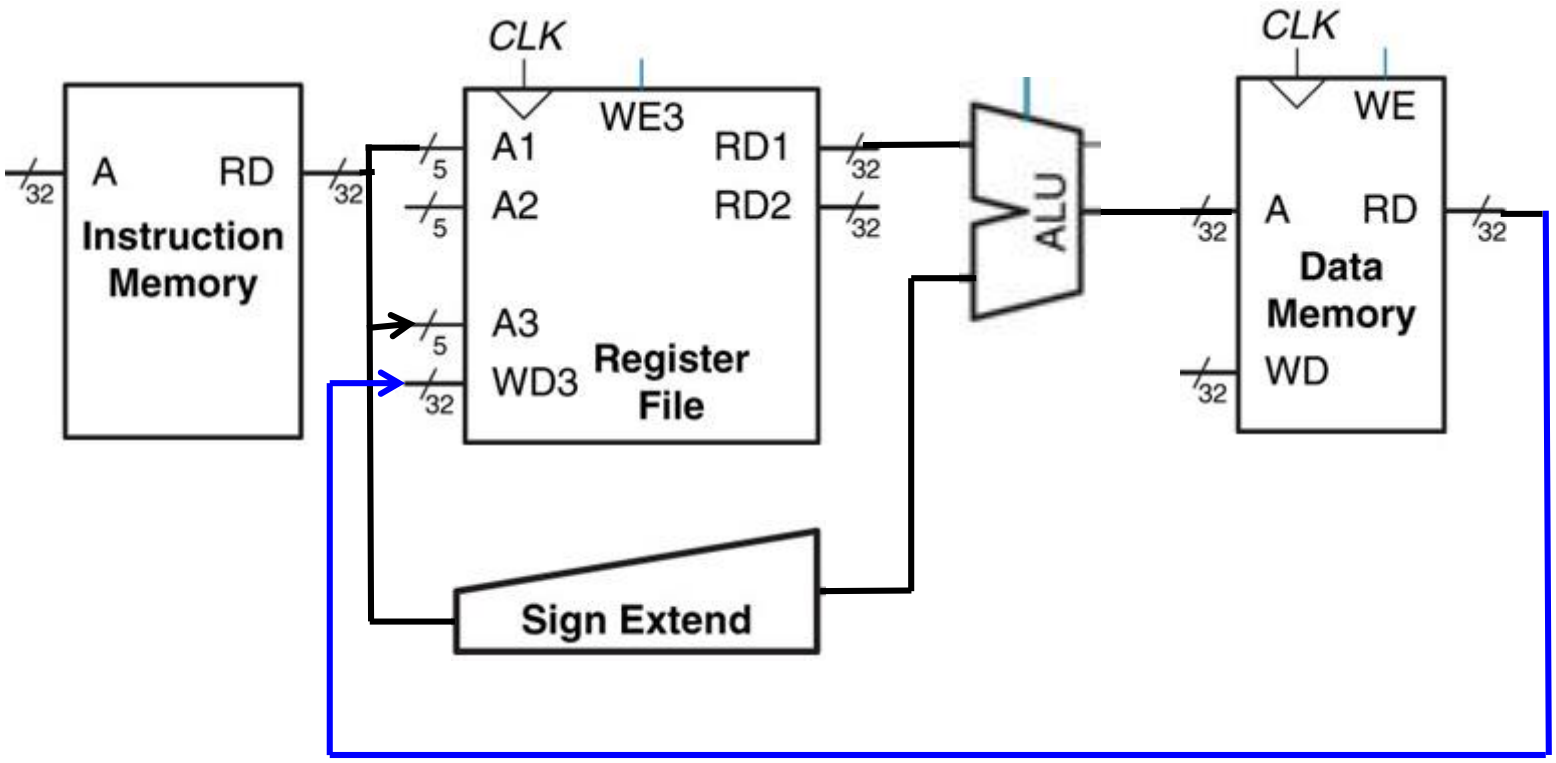
Sing-extend the immediate (`lw` Immediate)

Single-Cycle Datapath



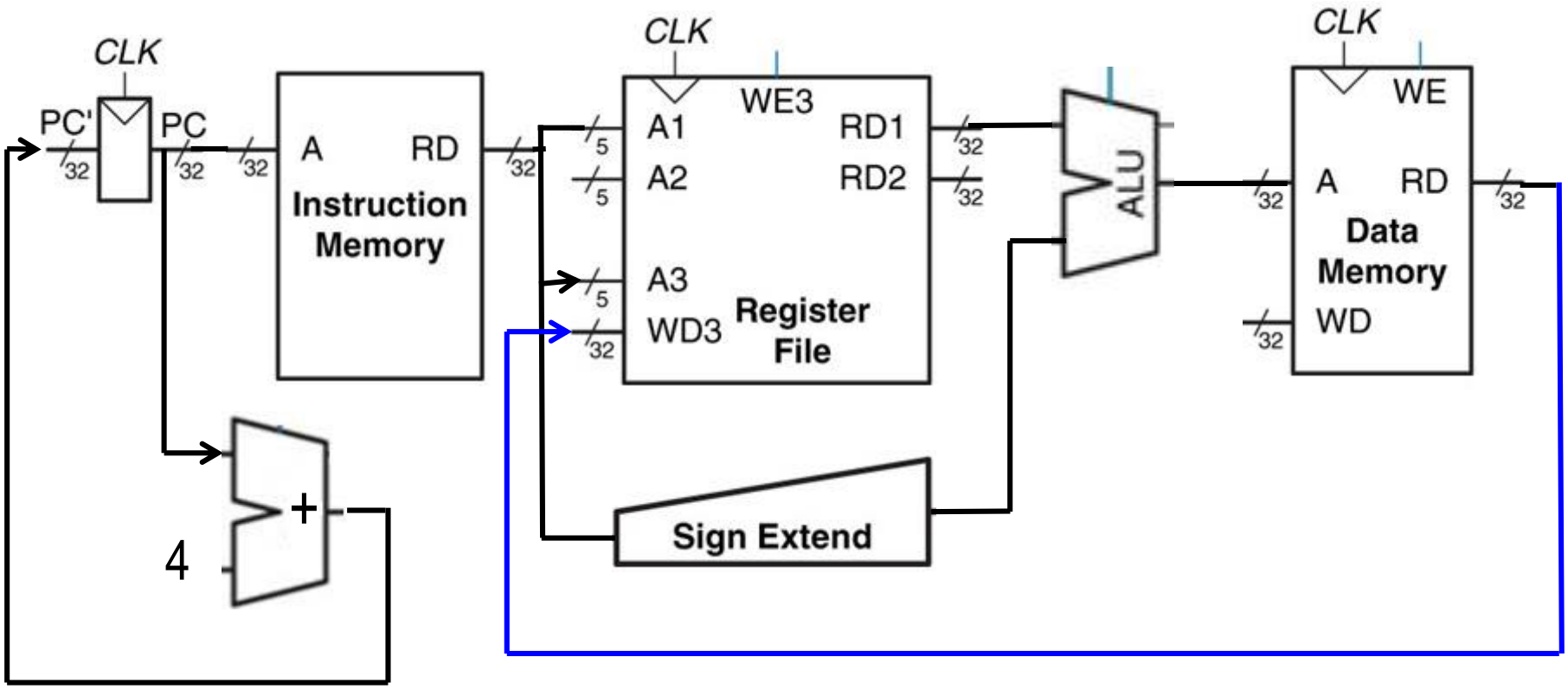
Compute the memory address (`lw` address)

Single-Cycle Datapath



Read data from memory and write it back to Register-File (`lw` Memory Read)

Plus the Fetch-part



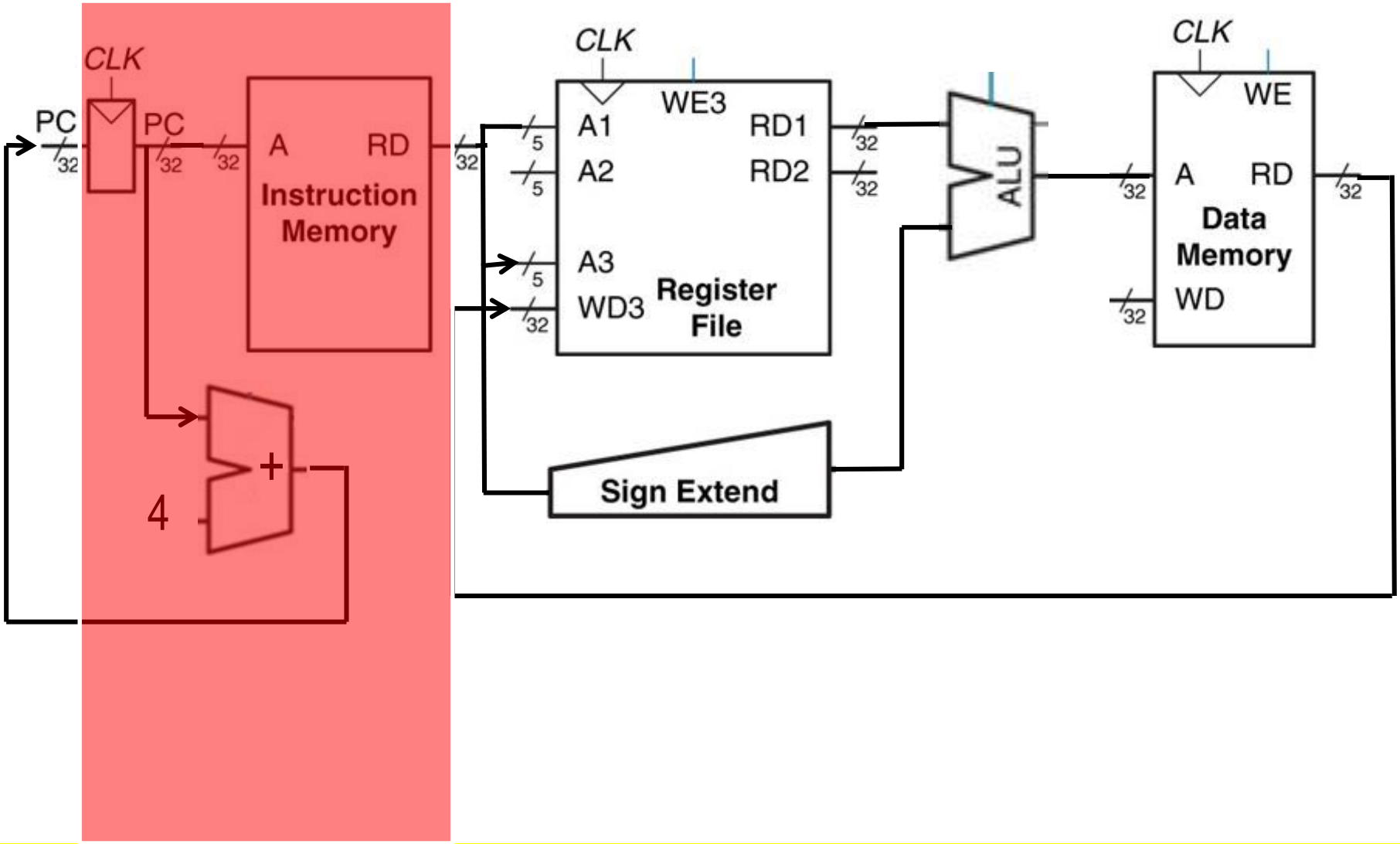
Read data from memory and write it back to Register-File (`lw` Memory Read)

Instruction execution cycle

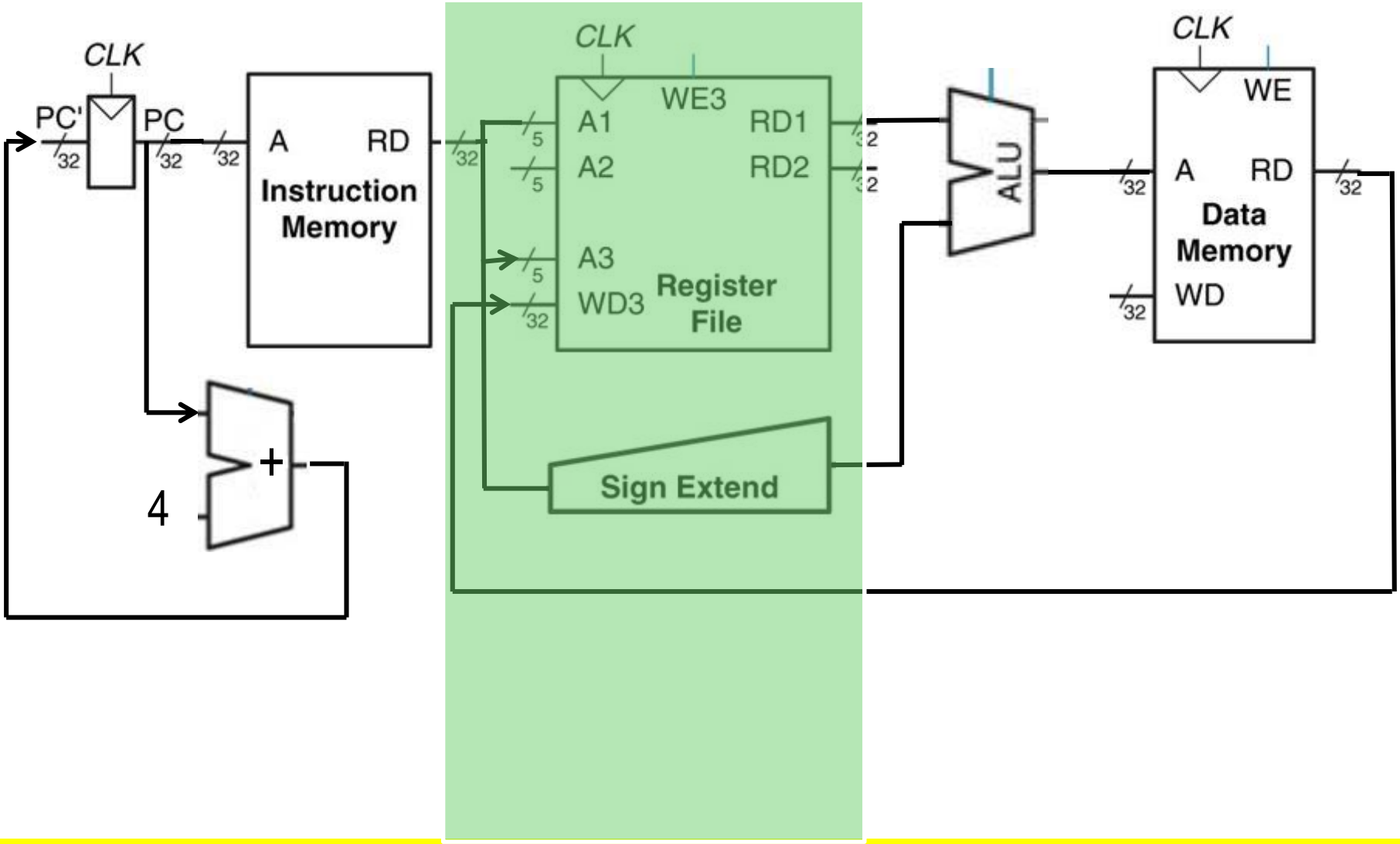
1. (IF): Fetch Instruction
2. (ID): Decode
3. (EX): Execute
4. (MEM): Memory Access
5. (WB): Write-back

- IF; **I**nstruction **F**etch machine code retrieved from memory
- ID; **I**nstruction **D**ecode set control signals, load source registers
- EX; **Ex**ecute Perform ALU operation
- MEM; **Mem**ory Read/write from/to memory
- WB; **W**rite-**B**ack Result written back into register file

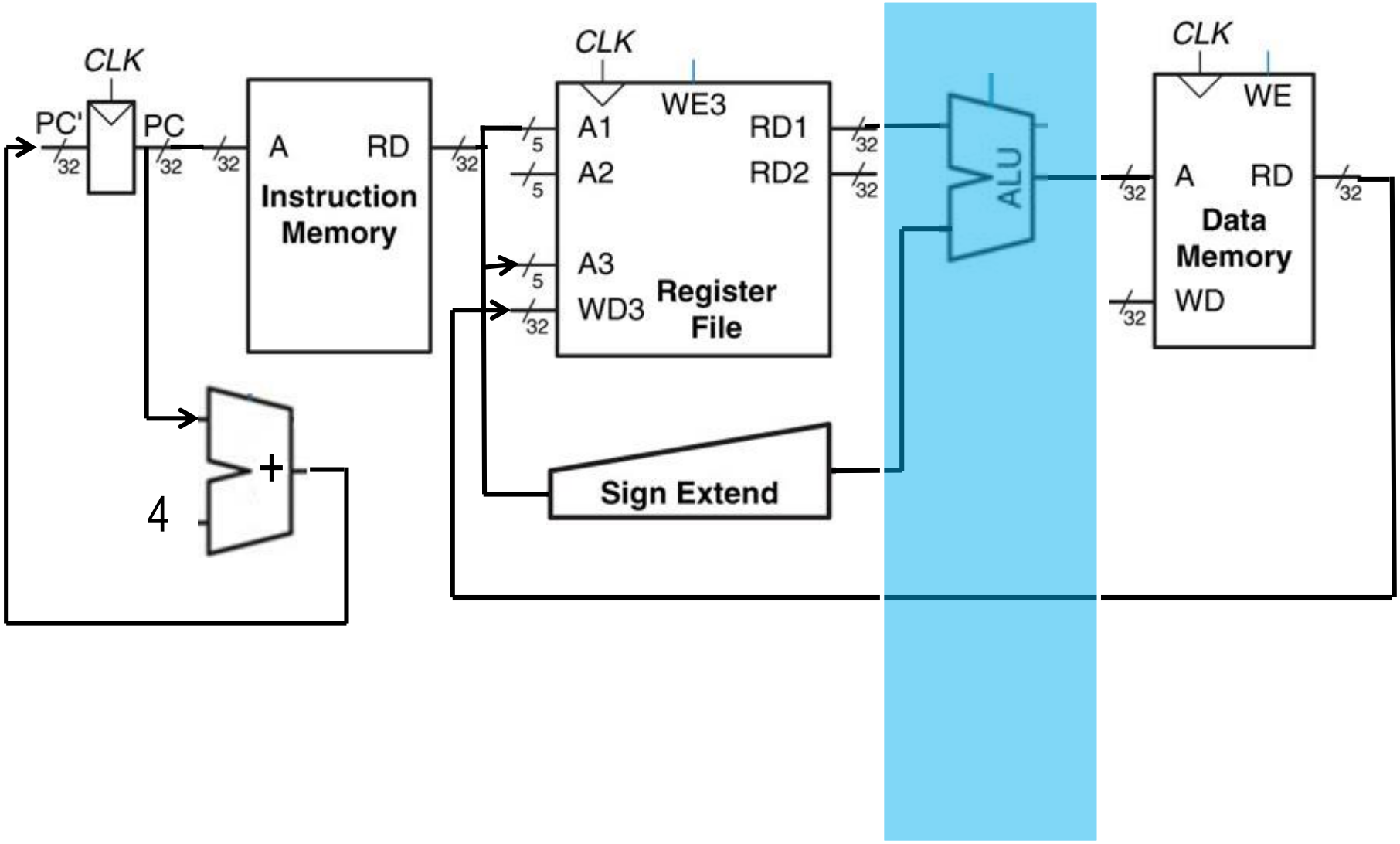
Fetch instruction



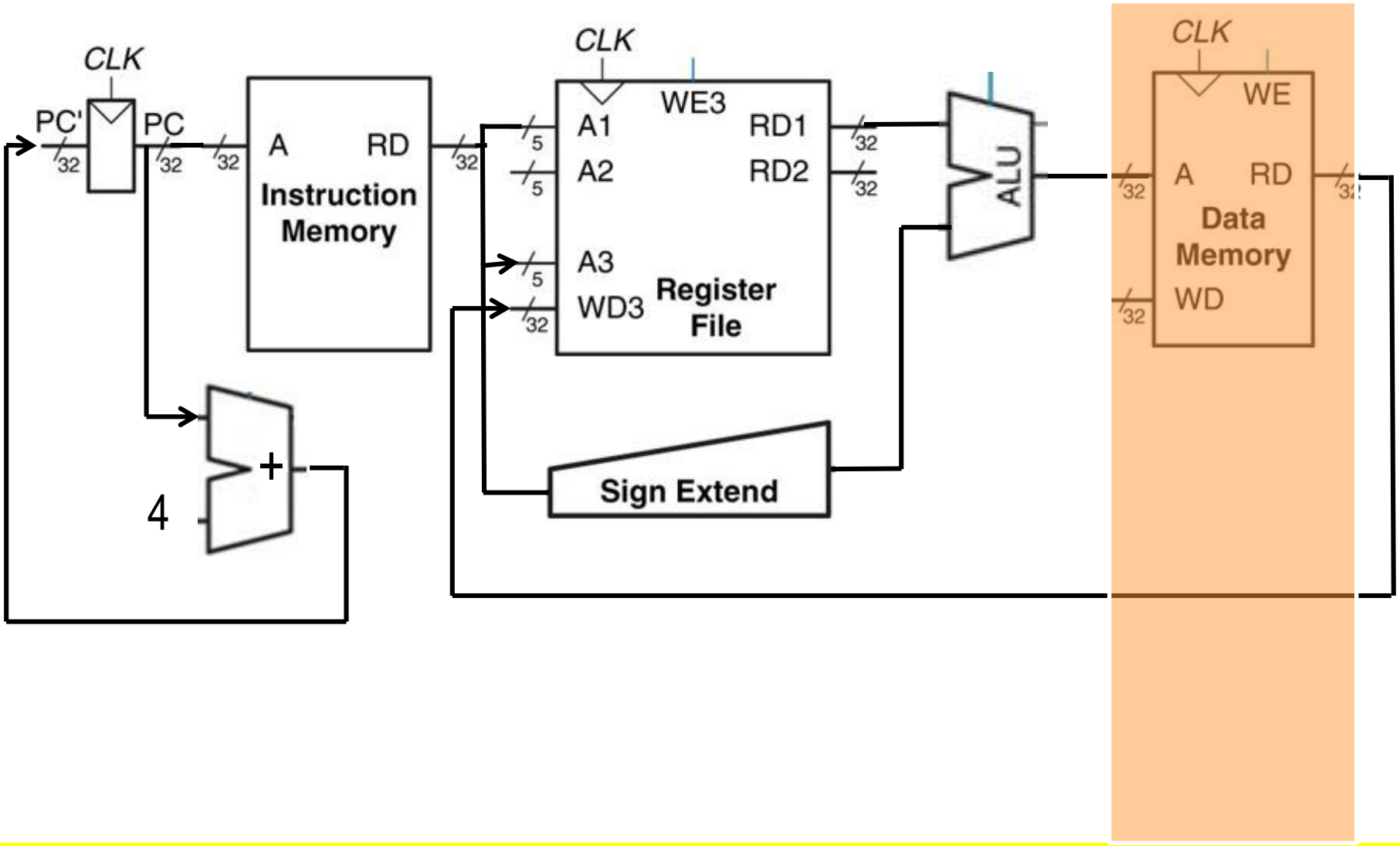
Decode


CLK
PC'
PC
32
32
32
A
RD
Instruction Memory
4
+
CLK
WE3
A1
A2
A3
WD3
RD1
RD2
5
5
5
32
32
32
32
Register File
Sign Extend
ALU
CLK
WE
A
RD
Data Memory
WD
32
32
32

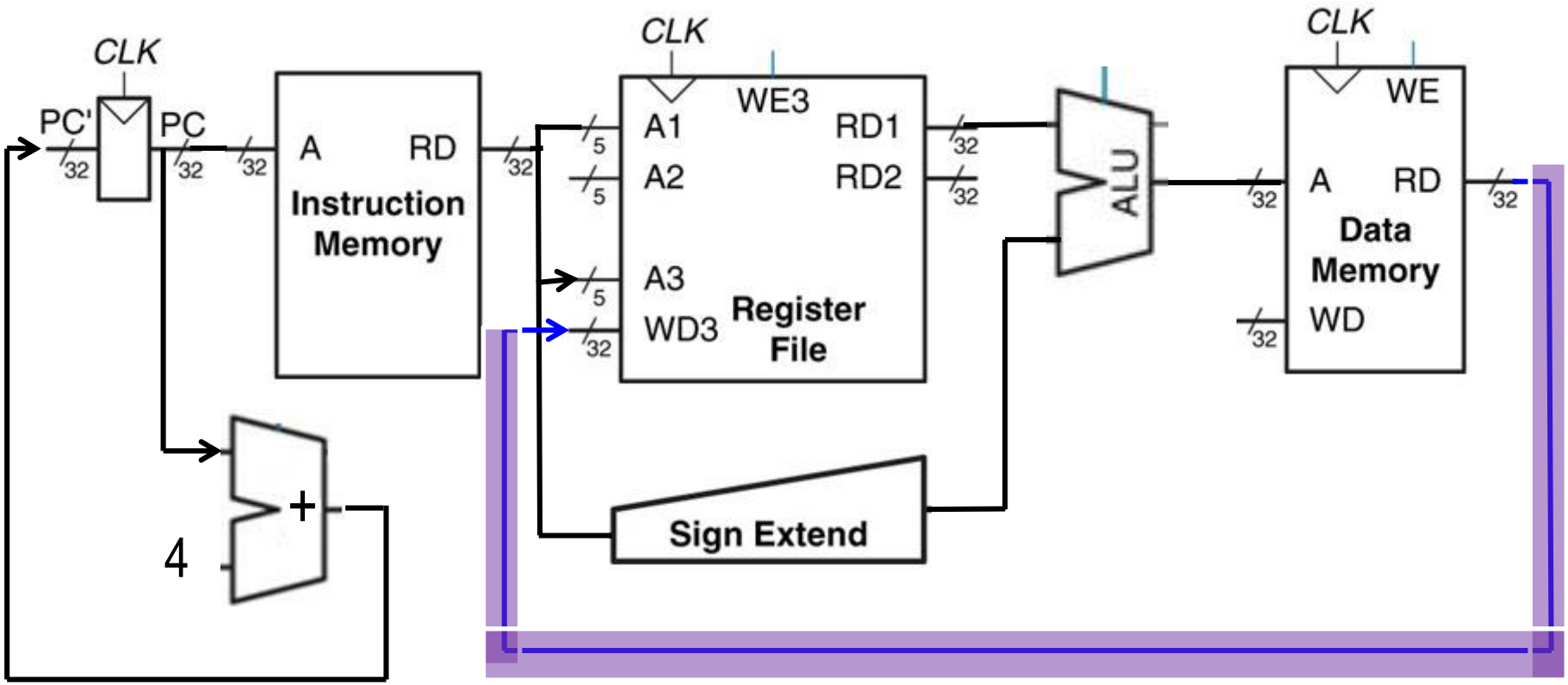
Execute


CLK
PC'
PC
32
32
32
A
RD
Instruction Memory
32
CLK
WE3
A1
A2
A3
WD3
RD1
RD2
Register File
5
5
5
32
32
32
ALU
CLK
WE
A
RD
Data Memory
WD
32
32
32
+
4
Sign Extend

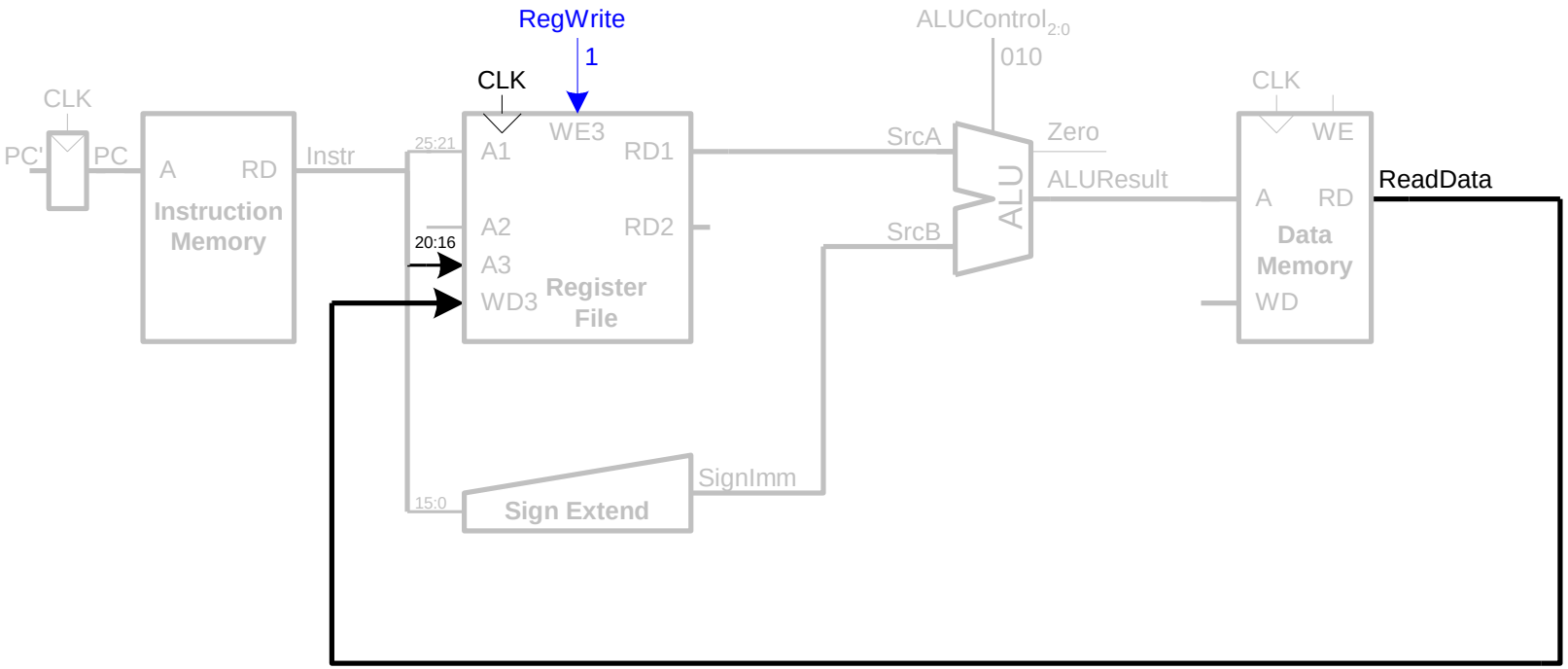
Memory Access



Write Back to Register-File

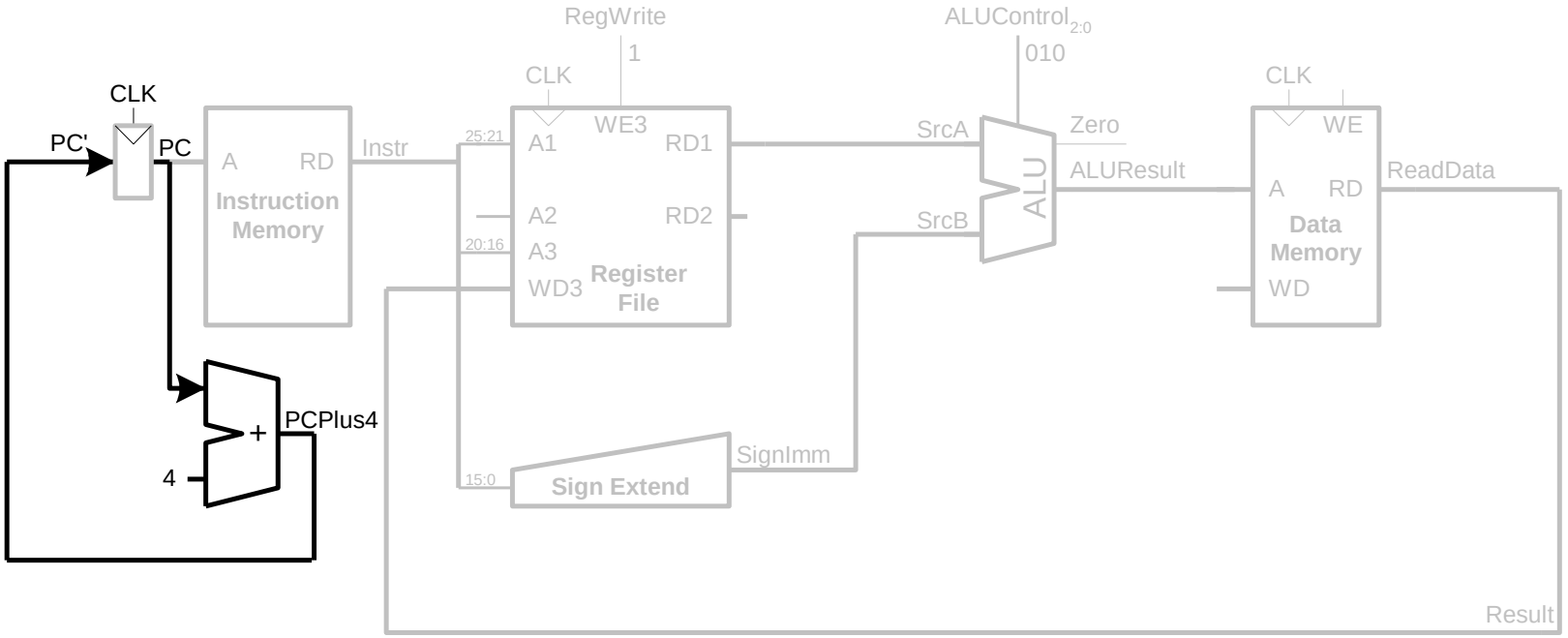


Microarchitecture: Another view

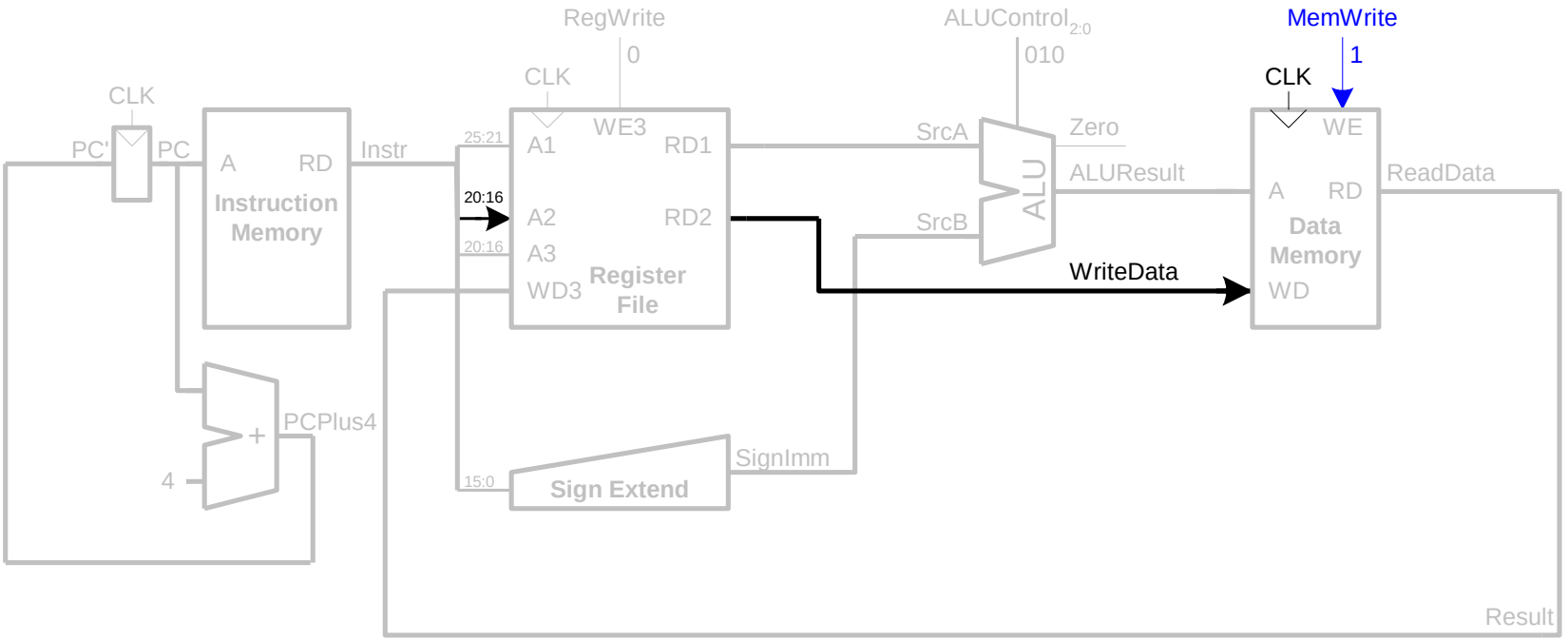


lw rt,rs, imm16

Next (second) instruction; PC increment

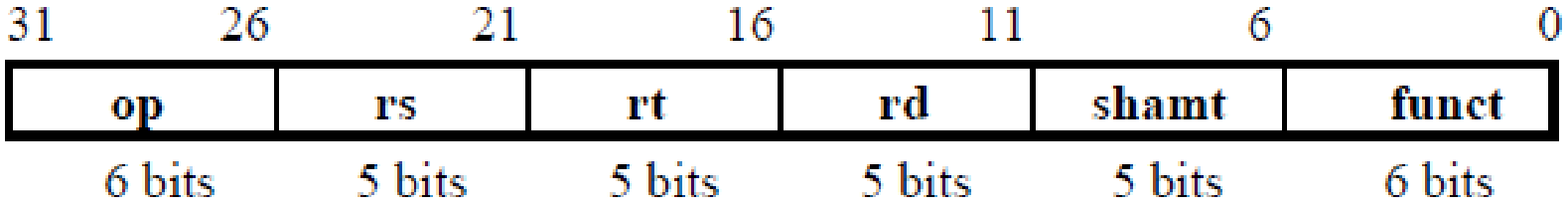


Write data in `rt` to memory



`sw rt,rs, imm16`

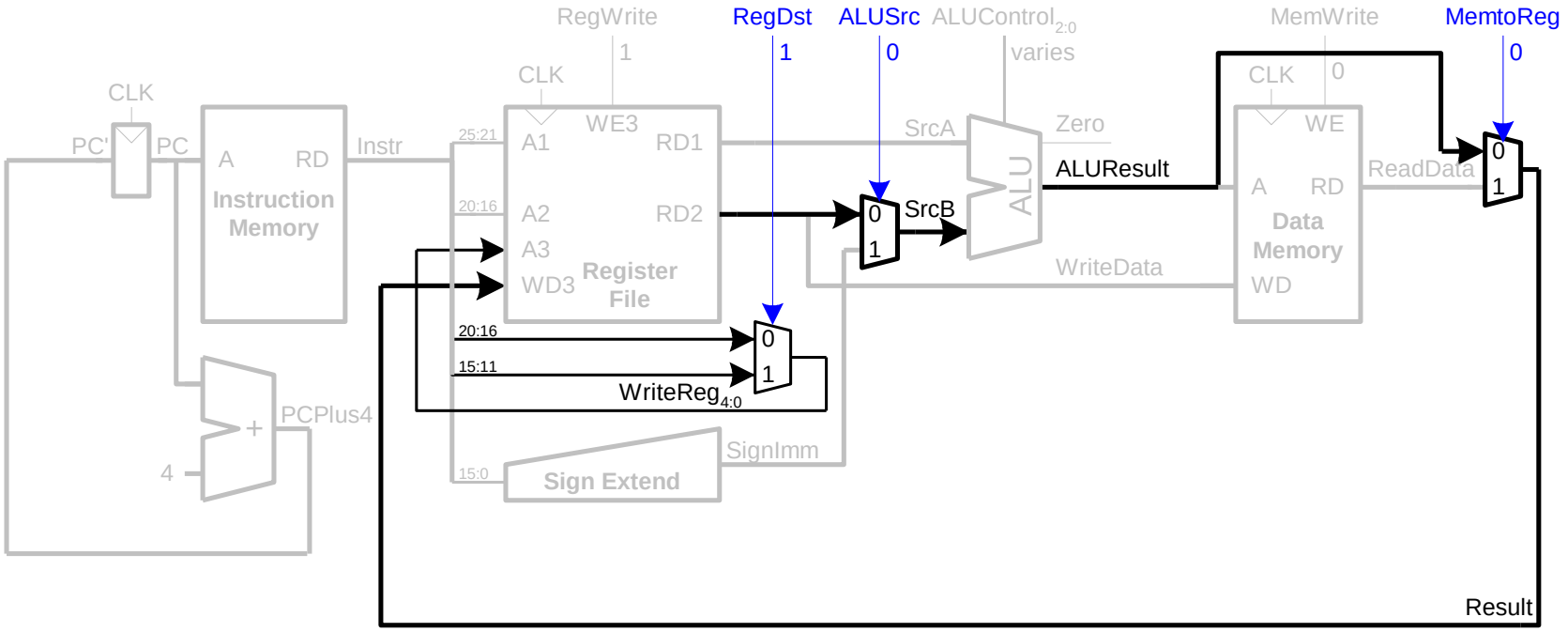
(R-type) instructions



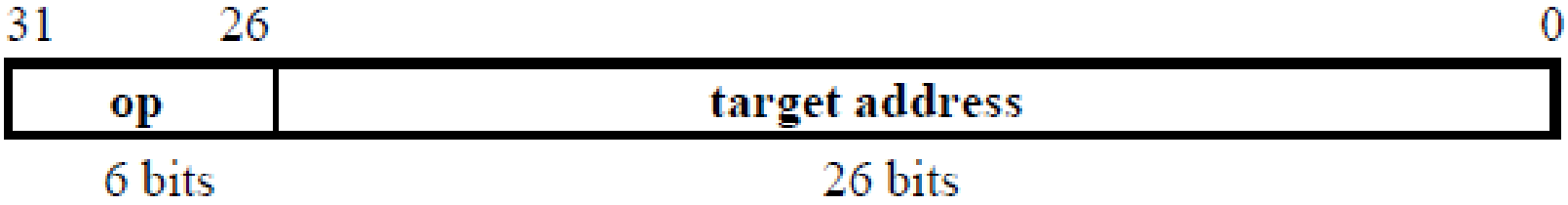

```
add rd, rs, rt          # r[d] = r[s] + r[t]

sub rd, rs, rt          # r[d] = r[s]- r[t]
```

Datapath (R-type): `add`

- Read from `rs` and `rt`
- Write *ALUResult* to register file
- Write to `rd` (instead of `rt`)



(J-type) instructions



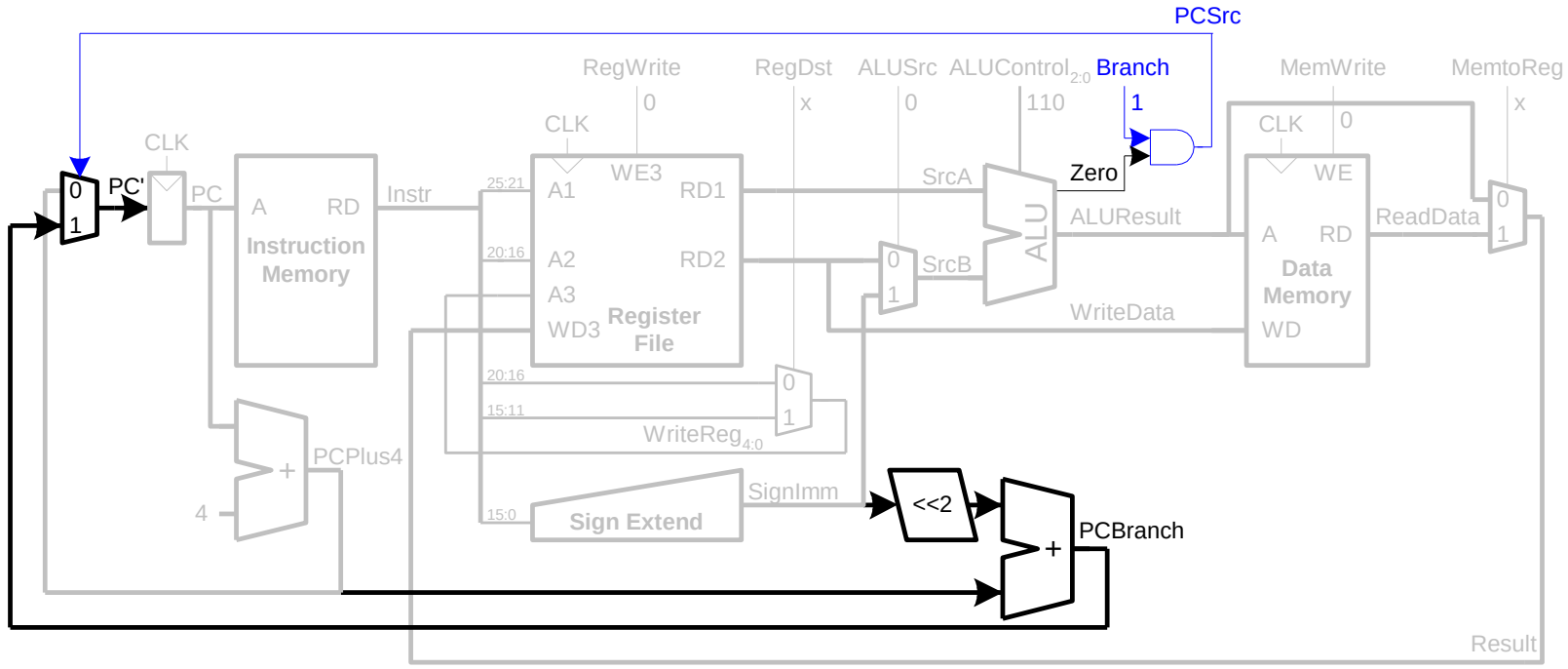
`beq rs,rt,imm16` # if $rs = $rt, $pc = imm16

Means the next instruction is at the address specified by label (imm16)

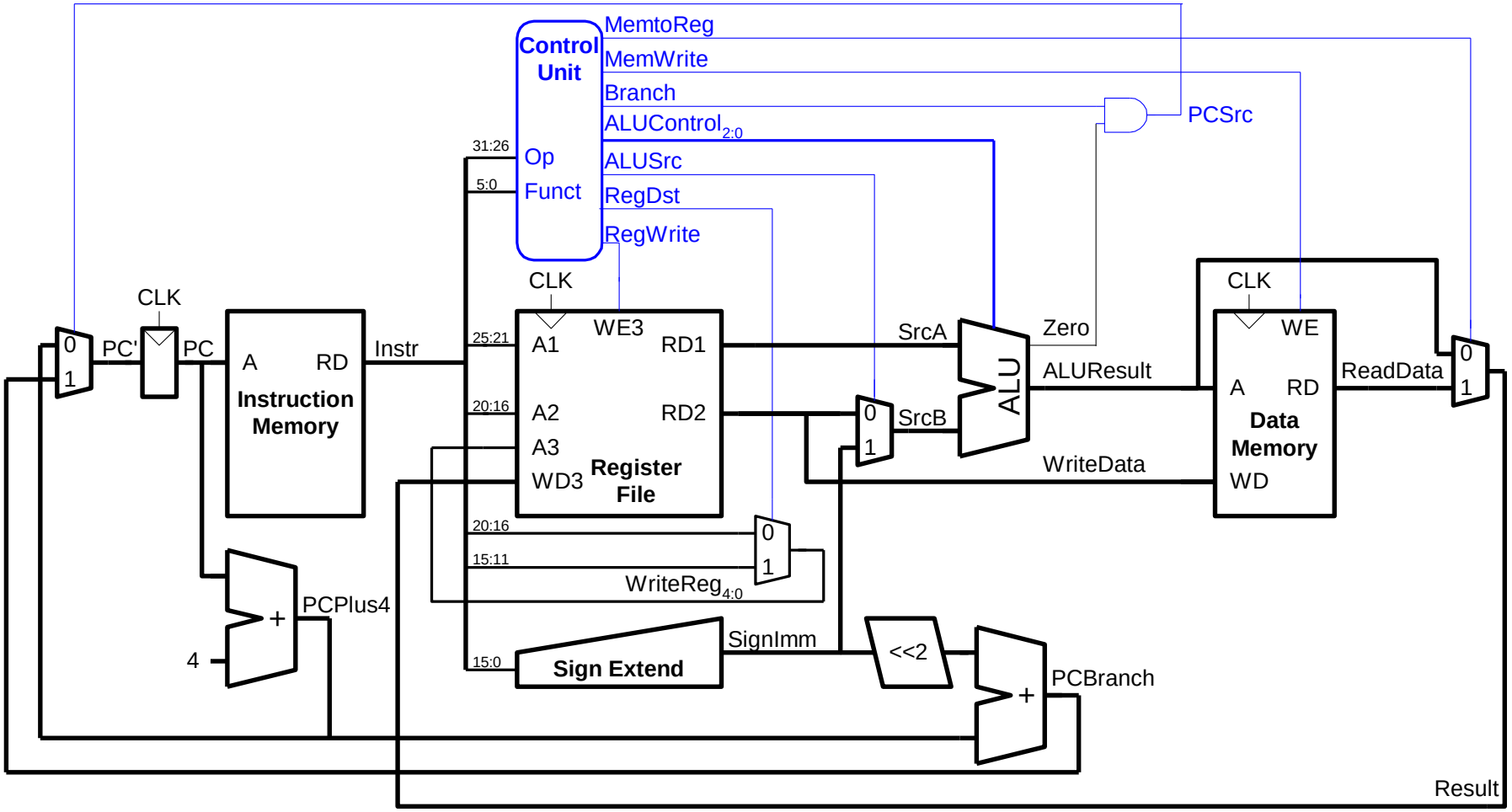
Datapath (J-type): beq

- Determine whether values in **rs** and **rt** are equal
- Calculate branch target address (BTA):

  BTA = (sign-extended immediate << 2) + (PC+4)



Single-Cycle Processor



MIPS is a **Load/Store** Architecture

- Since the MIPS is a Load/Store architecture, all instructions except Load and Store get their operands from CPU registers and store their result in a CPU register
- Hence, the instruction cycle for all instructions except, load and store, is somewhat simpler. Therefore ... it is easier …
  - To implement (hardware)
  - To understand Computer Assembly/Architecture
  - To educate.


THE END