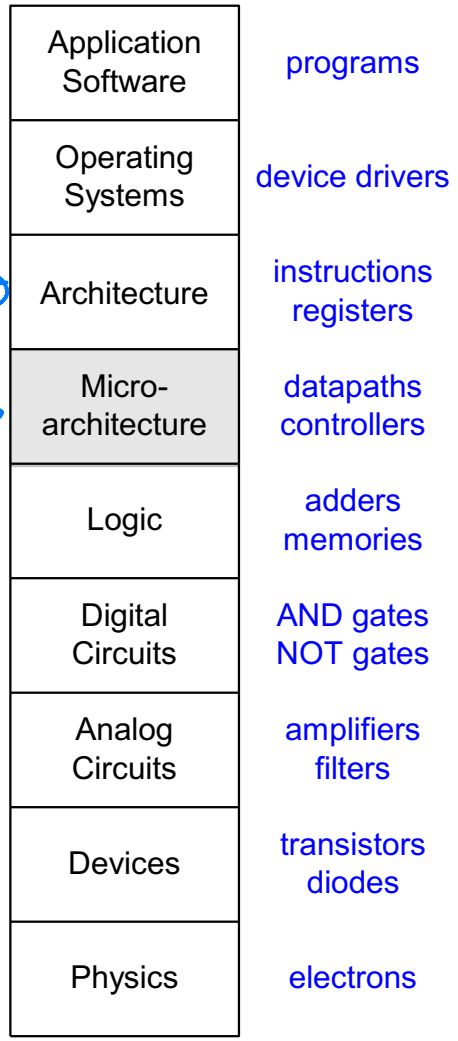
Chapter 7 :: Microarchitecture

Digital Design and Computer Architecture

Adapted from David Money Harris and Sarah L. Harris' book

Introduction

- Microarchitecture: how to implement an architecture in hardware
- Processor:
  - Datapath: functional blocks
  - Control: control signals



Microarchitecture

- Multiple implementations for a single architecture:
  - Single-cycle
    - Each instruction executes in a single cycle
  - Multicycle
    - Each instruction is broken up into a series of shorter steps
  - Pipelined
    - Each instruction is broken up into a series of steps
    - Multiple instructions execute at once.

modern processors are pipelined

7.2 Processor Performance

- Program execution time

  Execution Time = (# instructions)(cycles/instruction)(seconds/cycle)

- Definitions:
  - Cycles/instruction = CPI
  - Seconds/cycle = clock period
  - 1/CPI = Instructions/cycle = IPC
- Challenge is to satisfy constraints of:
  - Cost
  - Power
  - Performance

7.3 Single Cycle Processor:

- We consider a subset of MIPS instructions:
  - R-type instructions: `and`, `or`, `add`, `sub`, `slt`
  - Memory instructions: `lw`, `sw`
  - Branch instructions: `beq`
- Later consider adding `addi` and `j`

Architectural State

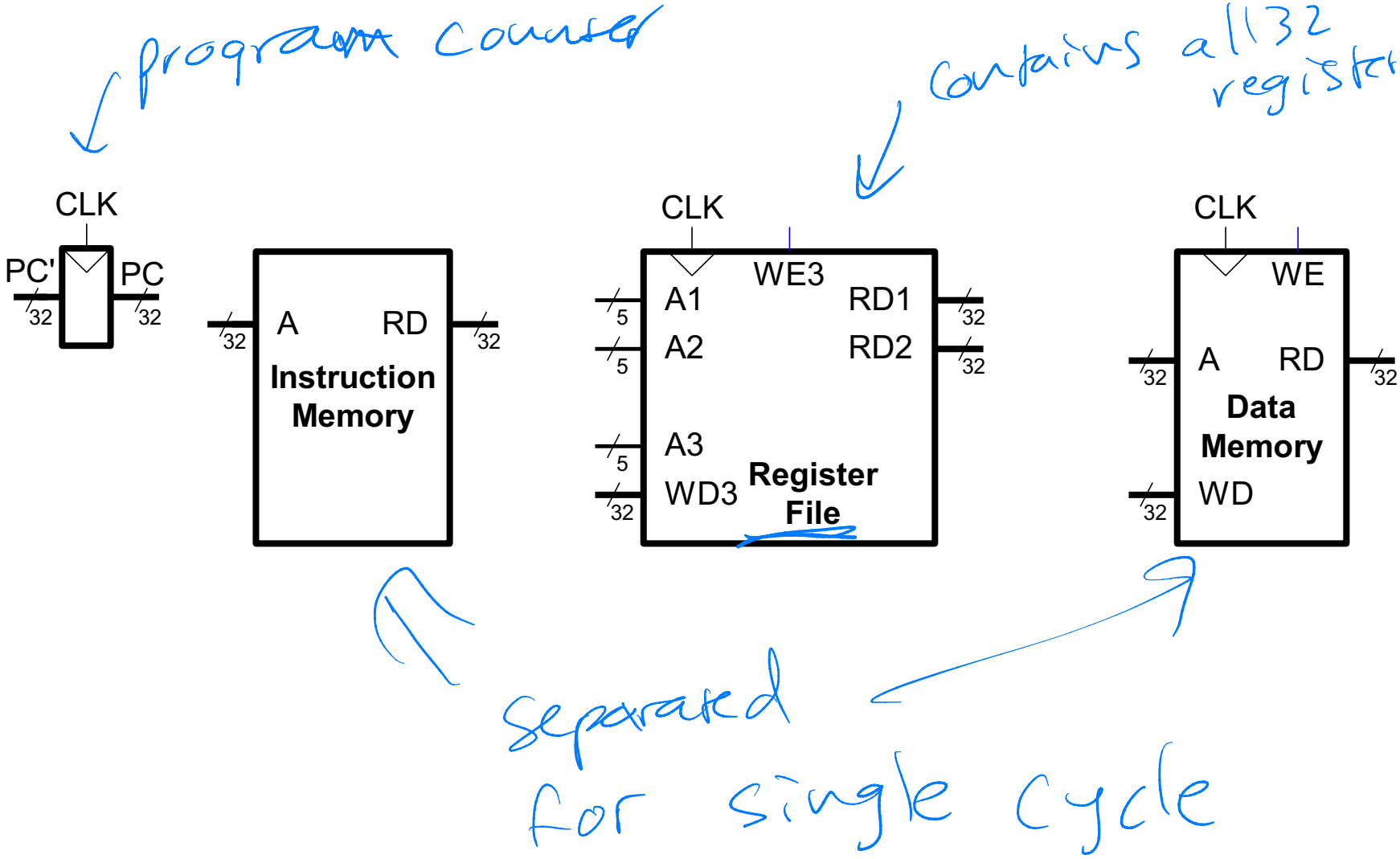
- Determines everything about a processor:
  - PC → program counter
  - 32 registers
  - Memory

the state of

where I am in program execution

→ every instruction updates arch. state.

MIPS State Elements



7.3.1 Single-Cycle MIPS Processor

- Datapath
- Control

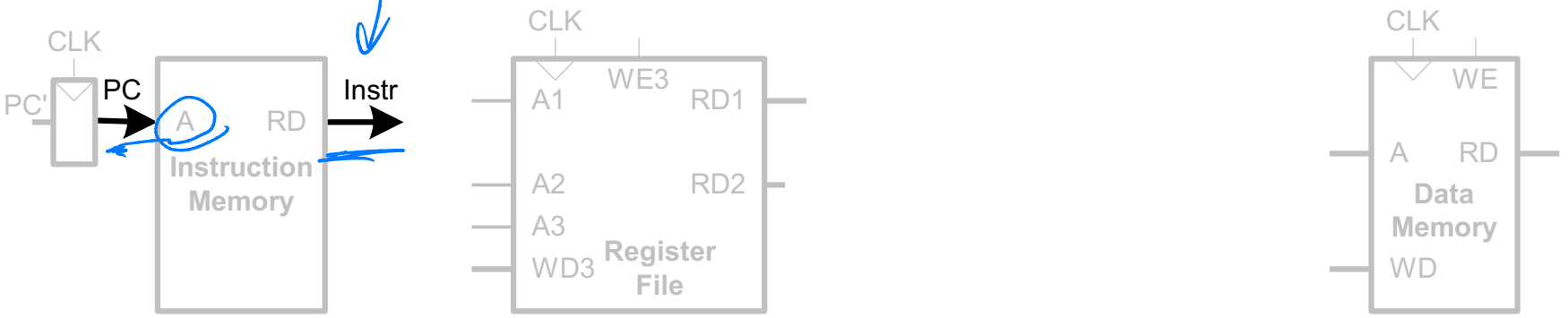
fetch - decode - execute - store

repeat

Single-Cycle Datapath: `lw` fetch

- First consider executing `lw`
- **STEP 1:** Fetch instruction

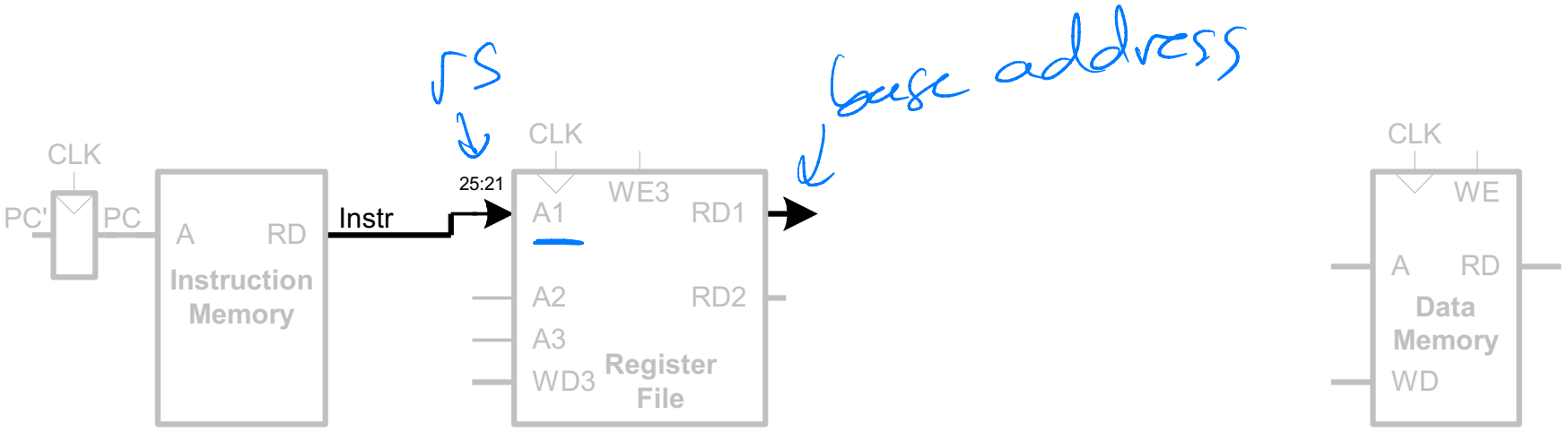
machine code



$ unless specified, all busses 32 bits

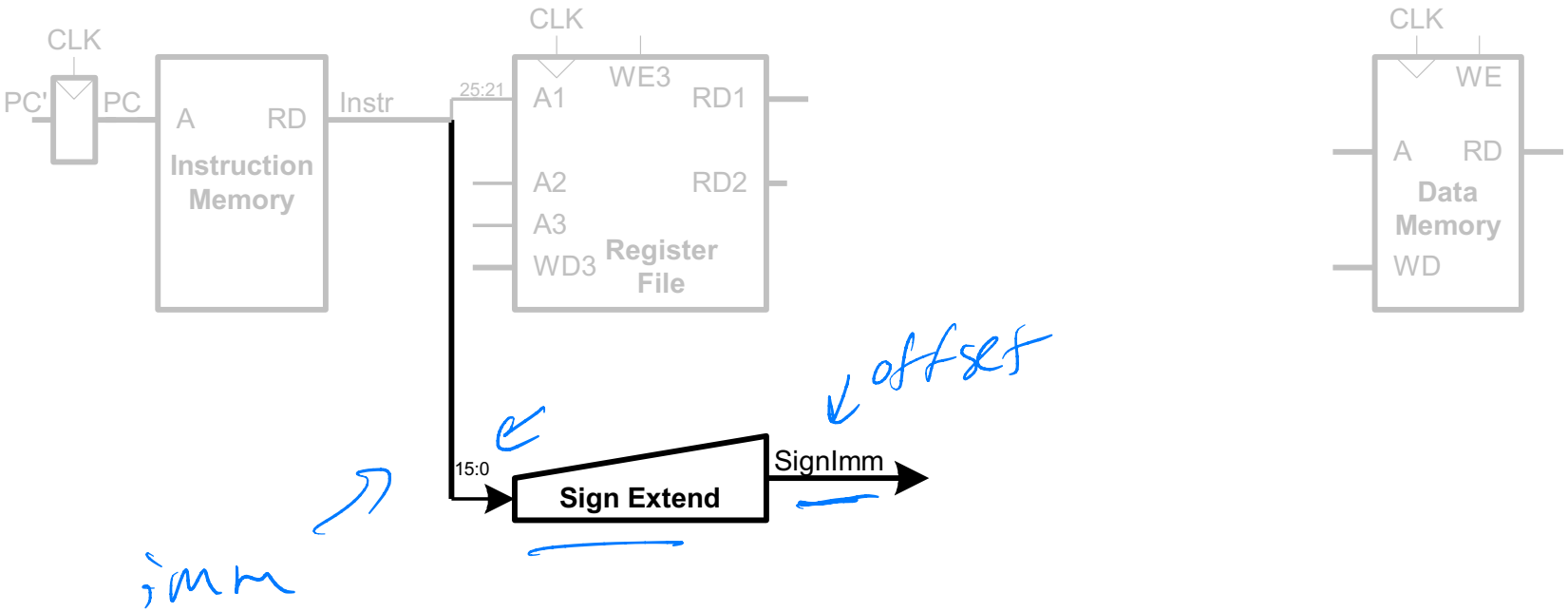
Single-Cycle Datapath: `lw` register read

- **STEP 2:** Read source operands from register file



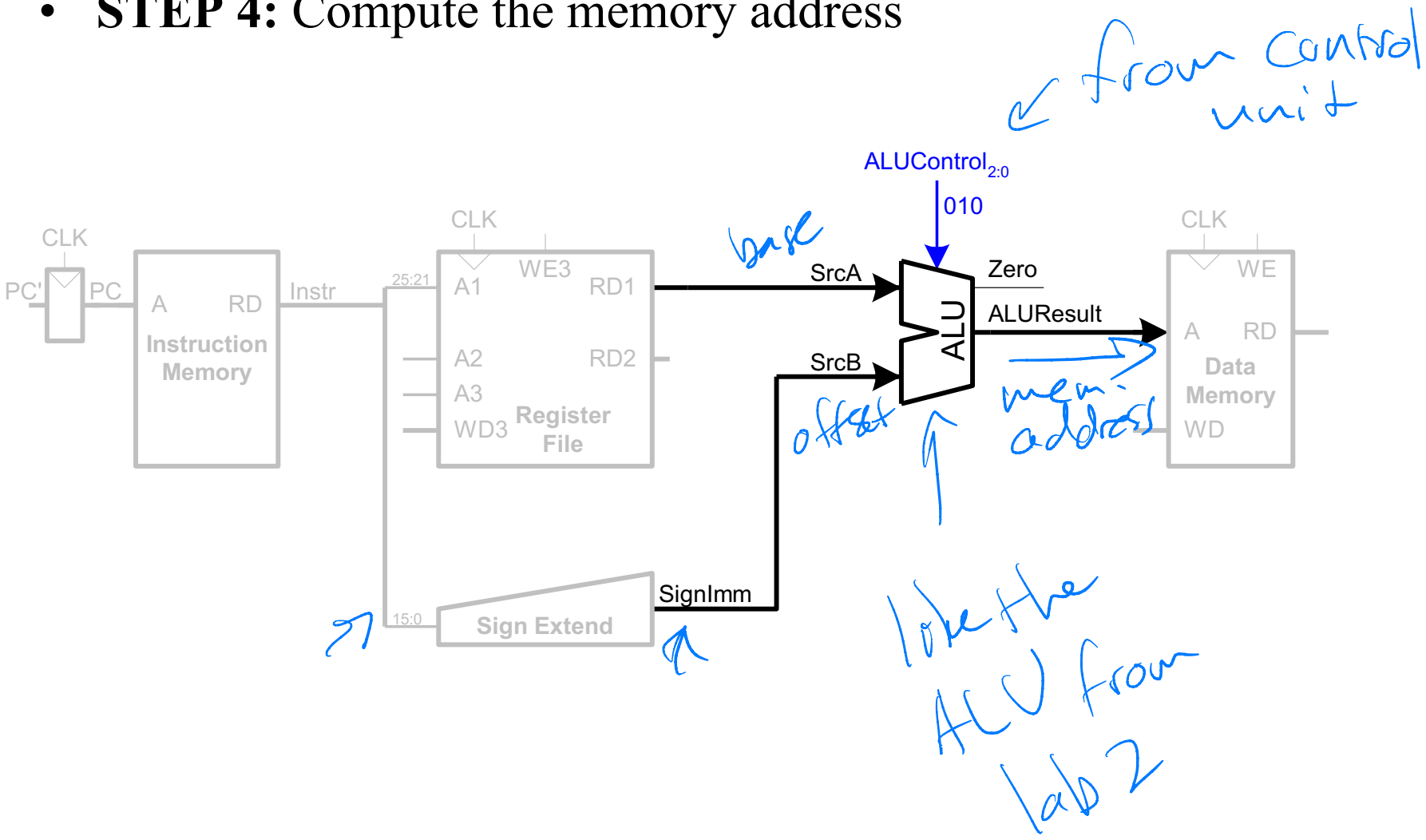
Single-Cycle Datapath: `lw` immediate

- **STEP 3:** Sign-extend the immediate



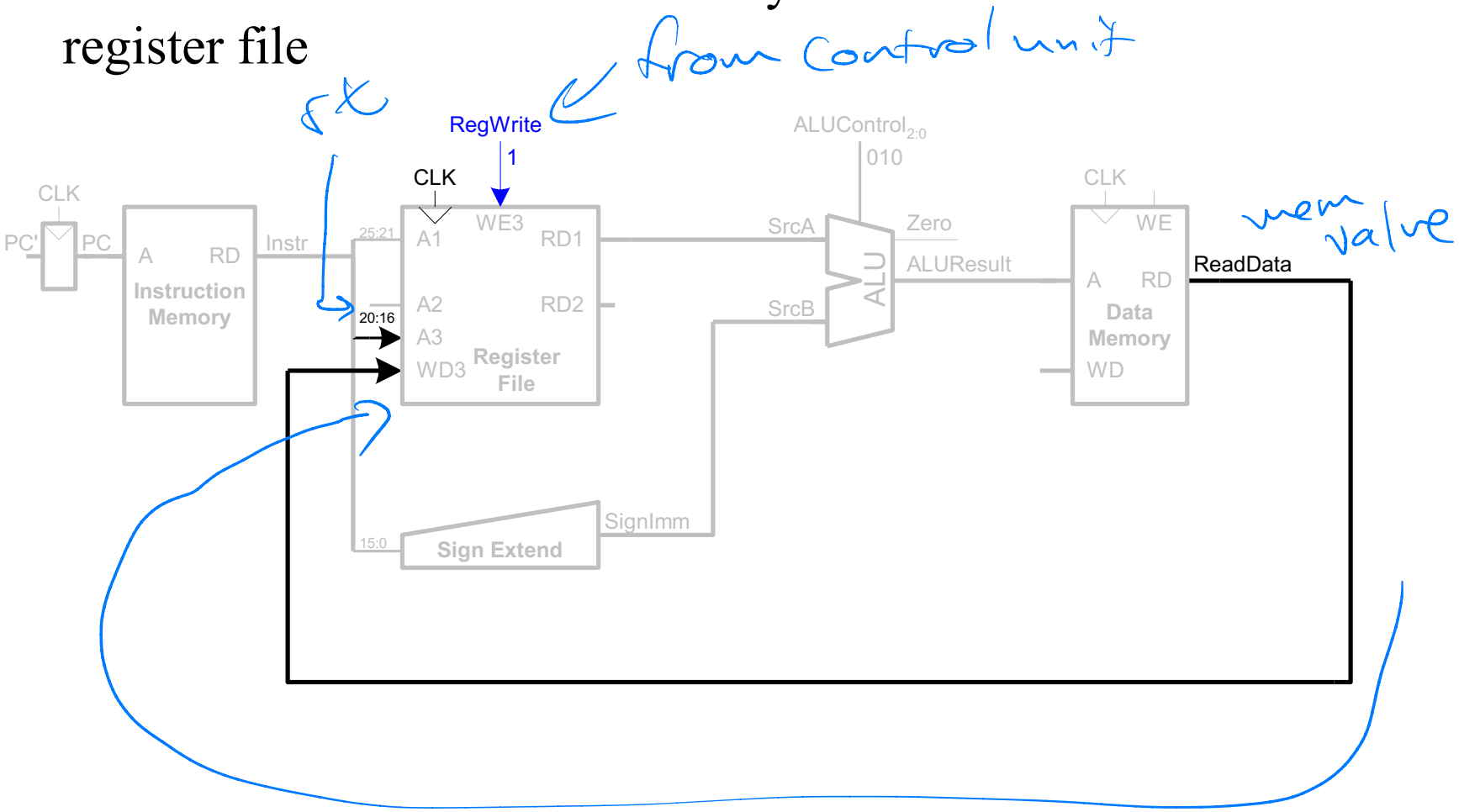
Single-Cycle Datapath: `lw` address

- **STEP 4:** Compute the memory address



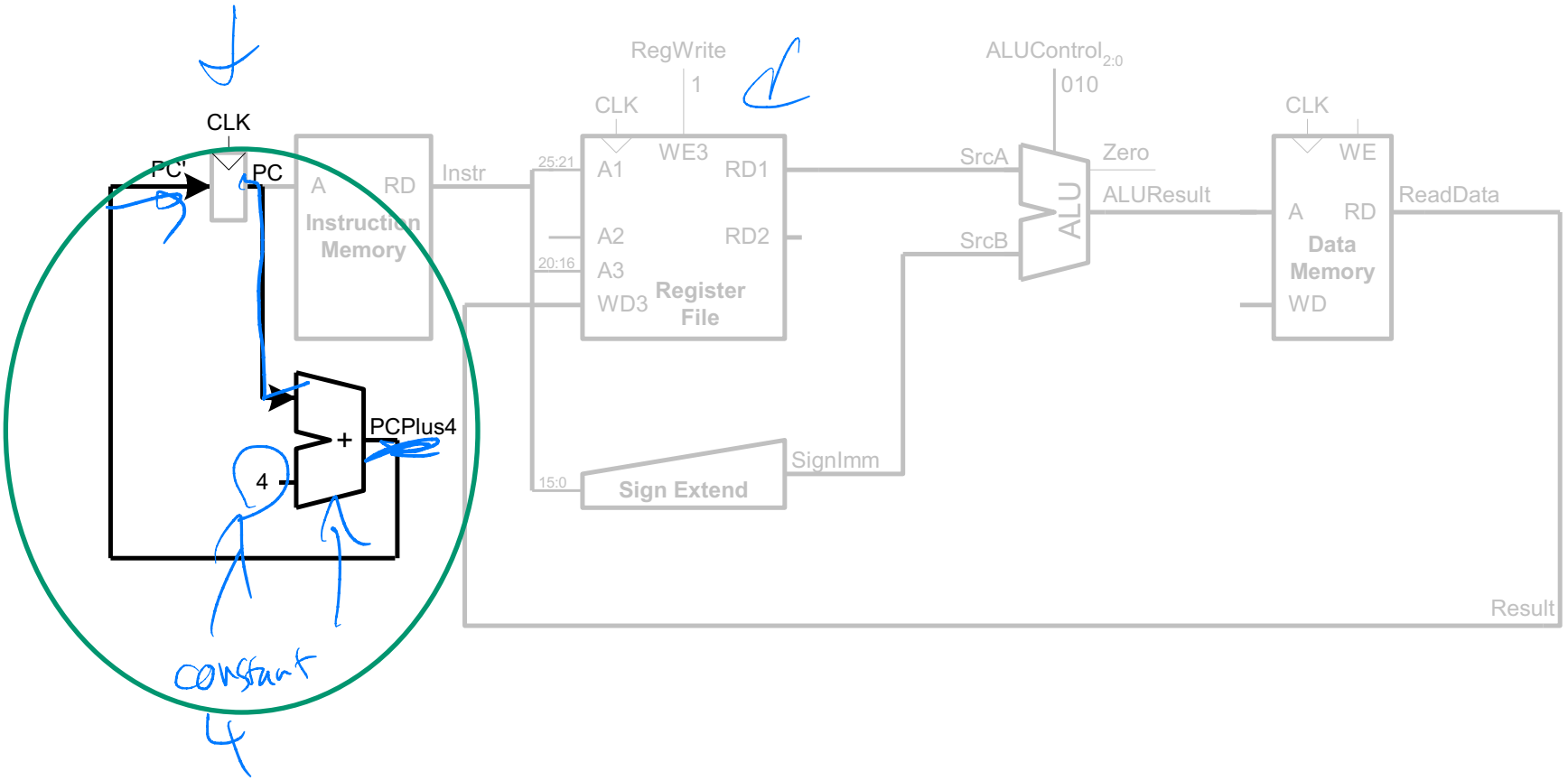
Single-Cycle Datapath: `lw` memory read

- **STEP 5:** Read data from memory and write it back to register file

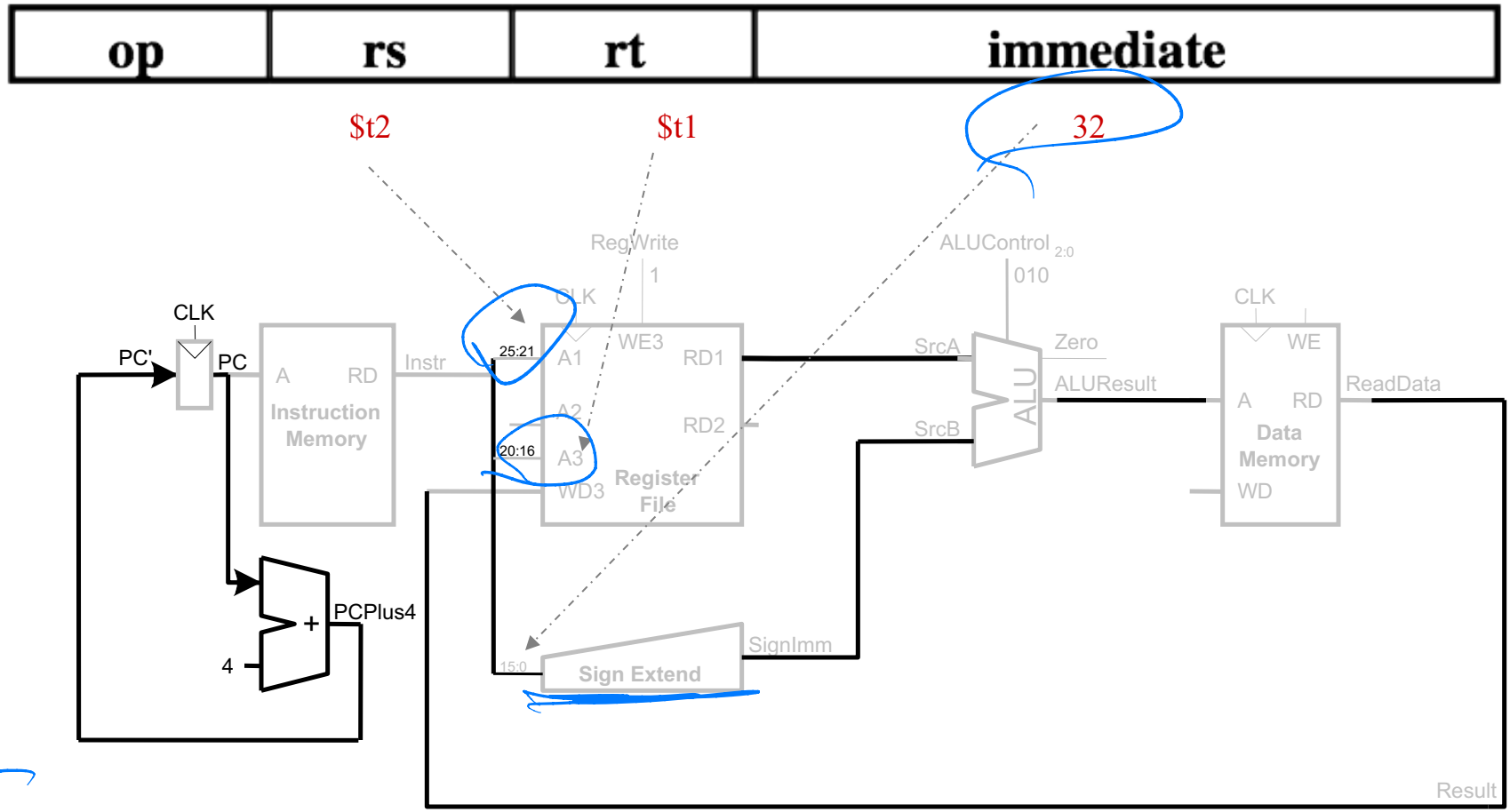


Single-Cycle Datapath: `lw` PC increment

- **STEP 6:** Determine the address of the next instruction



Single-Cycle Datapath: LW $t1, 32($t2) :load data in Data Memory address ($t2+32) to reg $t1 in Register File

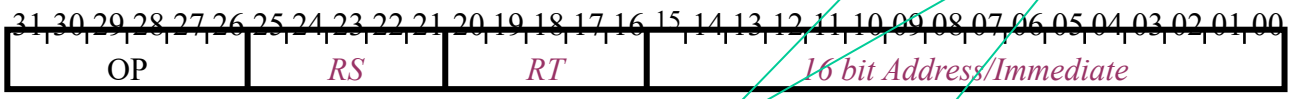


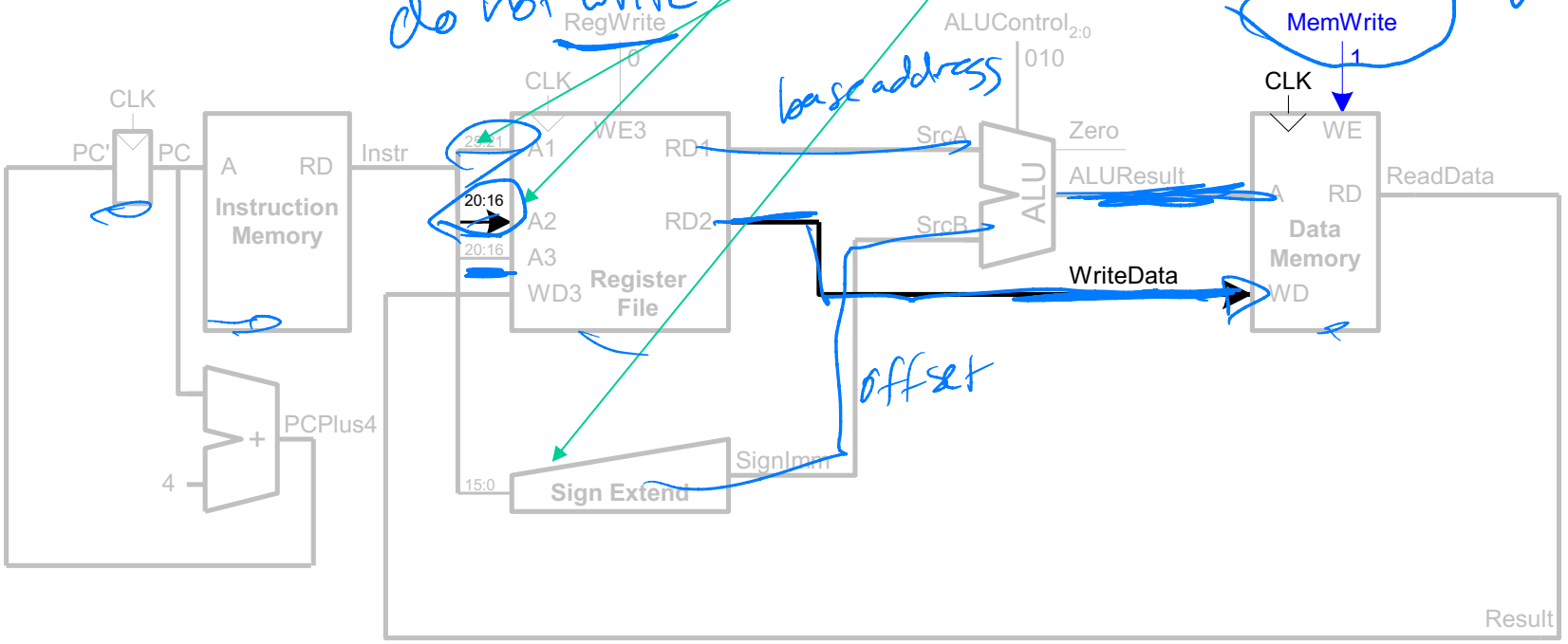
Syntax: LW $t1, 32($t2)

Action: $t1 = Mem[$t2 + 32] .Fetch from memory address[$t2+32]. And load to register $t1 in register file

Single-Cycle Datapath: `sw $s0, 0($t0)`

- Write data in `rt` to memory





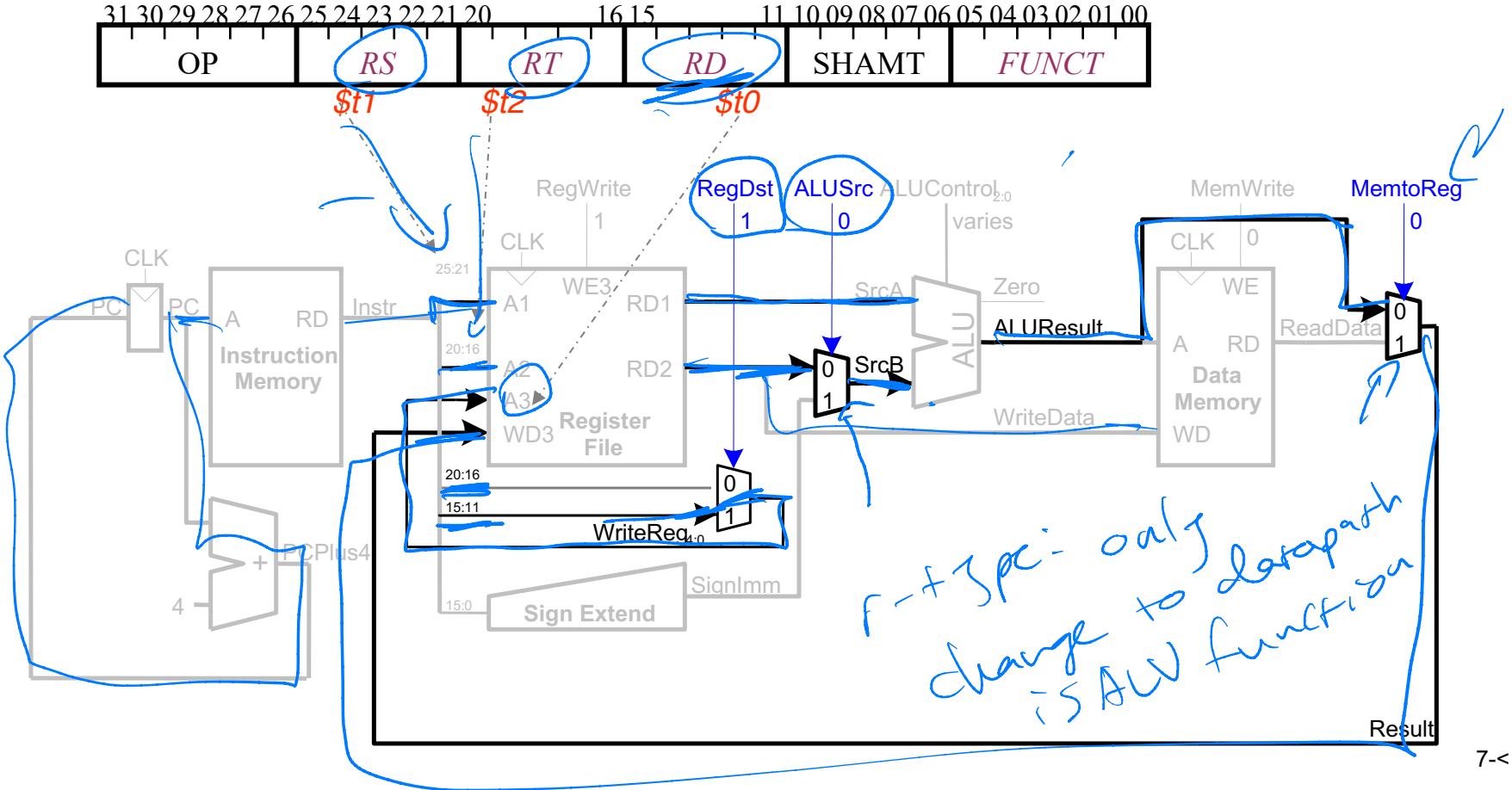
Chapter 7 :: Microarchitecture

Digital Design and Computer Architecture

Adapted from David Money Harris and Sarah L. Harris' book

Single-Cycle Datapath: R-type : ADD $t0 $t1 $t2

- Read from `rs` and `rt`
- Write *ALUResult* to register file
- Write to `rd` (instead of `rt`)
- **Important: Add multiplexers**(lw and R type use same A3: destination reg, but from different source;Use same srcB, but from different source, LW from imm field;use same WD3, but from different source)

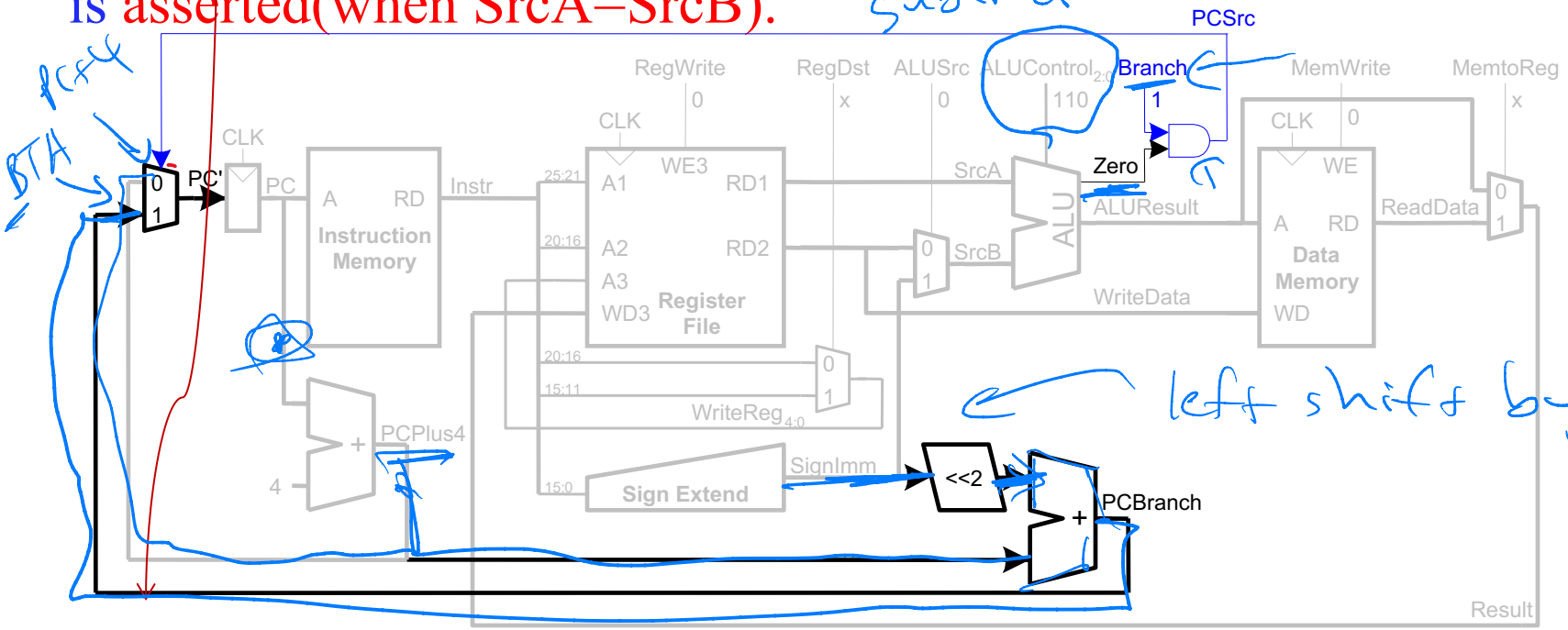


Single-Cycle Datapath: `beq`

- Determine whether values in `rs` and `rt` are equal
- Calculate branch target address:

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

- The immediate must be sign-extended and multiplier by 4 (left shift by 2, since address are multiple of 4)
- *PCBranch* is selected if the instruction is a branch and the *ZERO* flag is asserted(when SrcA=SrcB).



Addressing Modes

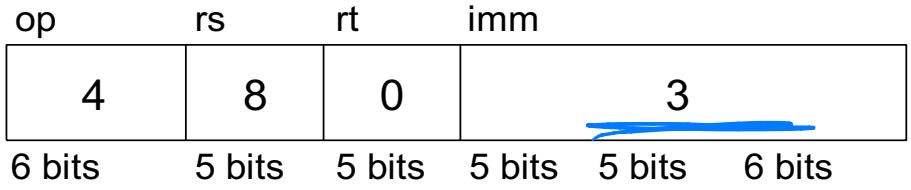
PC-Relative Addressing

```
0x10              beq   $t0, $0, else
                      #else=3: (0x20-0x14)/4
0x14              addi  $v0, $0, 1
0x18              addi  $sp, $sp, i
0x1C              jr          $ra
0x20        else: addi  $a0, $a0, -1
0x24              jal   factorial
```

Assembly Code

```
beq $t0, $0, else
(beq $t0, $0, 3)
```

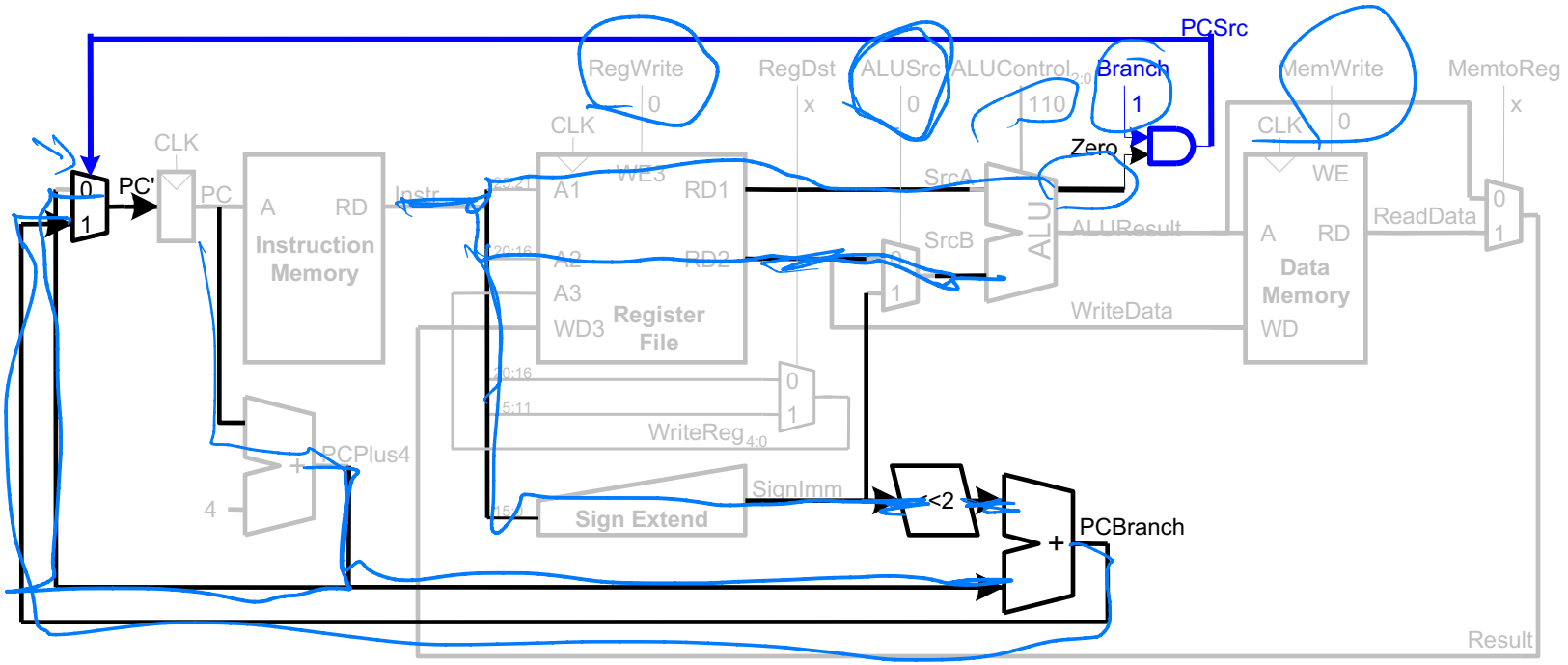
Field Values

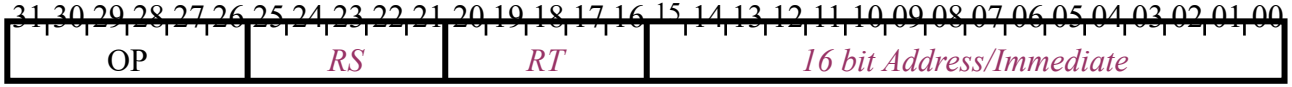


Single-Cycle Datapath: `beq $s1, $s2, 12`

- Determine whether values in `rs` and `rt` are equal
- Calculate branch target address:

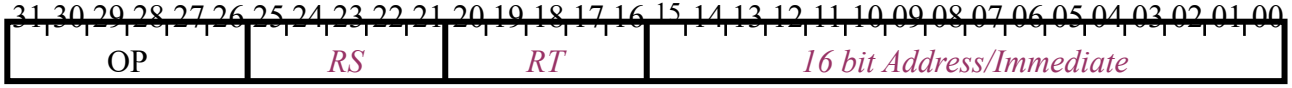
  BTA = (sign-extended immediate << 2) + (PC+4)



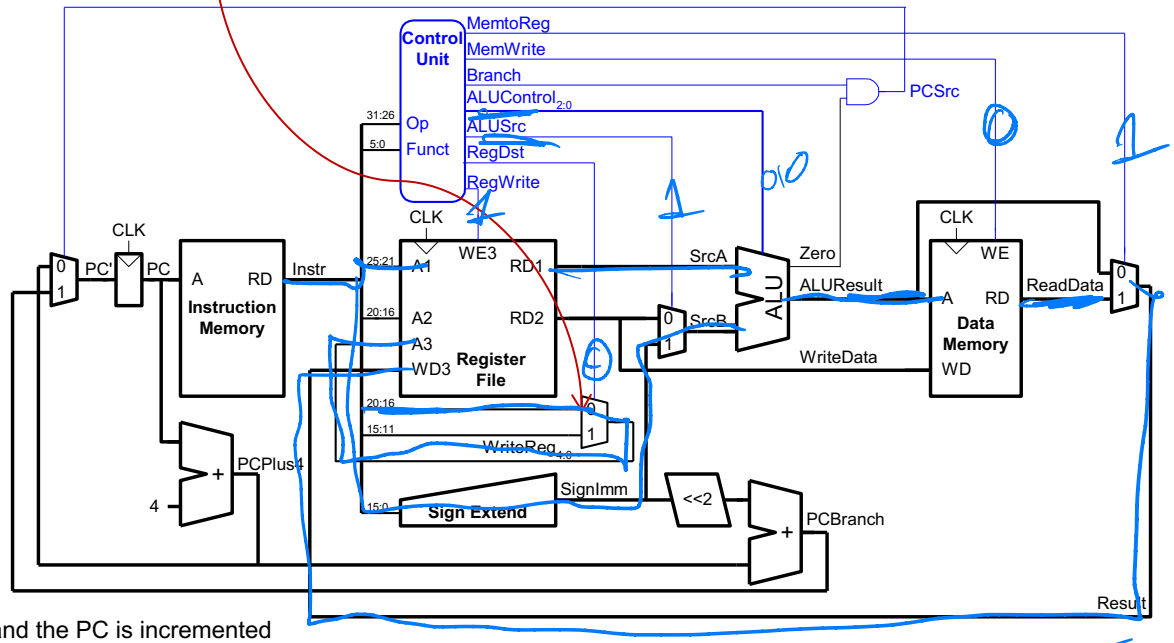


Syntax: BEQ $1, $2, 12
Action: If ($1 != $2), PC = PC + 4
Action: If ($1 == $2), PC = PC + 4 + 12x4

Control Unit: Load Word



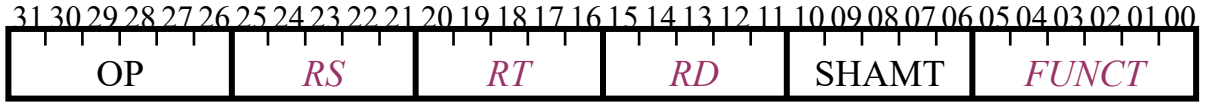

```
Load Instruction: lw rt, imm(rs)
Instruction = Mem[PC];
Addr = Reg[rs] + SignExtend(imm);
Reg[rt] = Mem[Addr];
PC = PC + 4;
```



1.The instruction is fetched, and the PC is incremented
2. A register value is read from the register file
3.The ALU computes the sum of the value read from the read from the register file and the sign-extended, lower 16 bits of the instruction(offset)
4. The sum from the ALU is used as the address of the data memory file
5. The data from the memory unit is written into the register file; the register destination is given by bits 20:16 of the instruction.

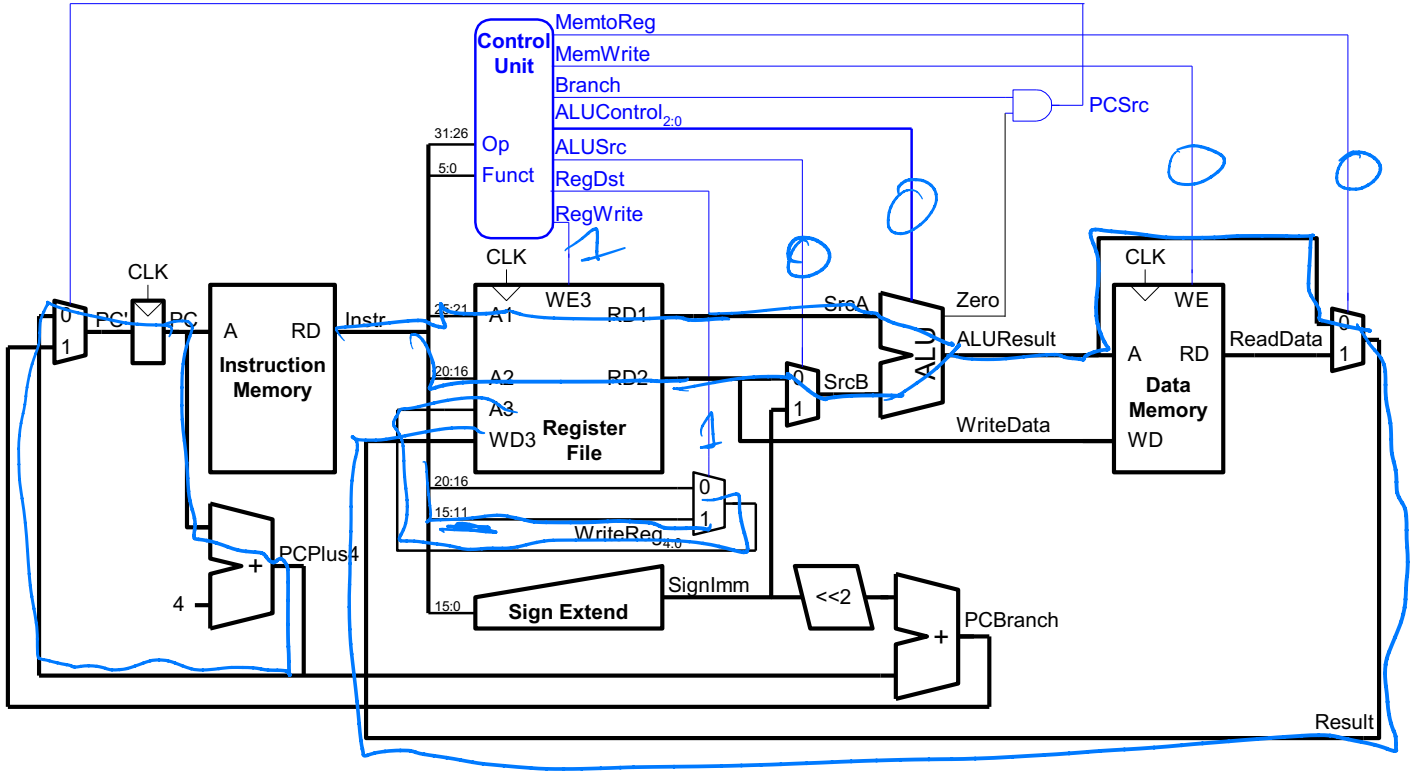
Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
lw	100011	1	0	1	0	0	1	00
								7-<22>

Control Unit:R-type



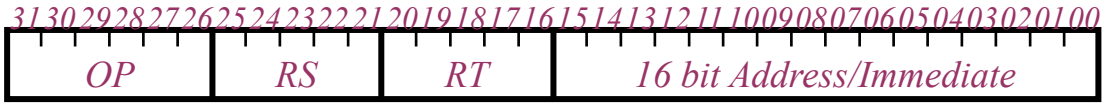
Add instruction:

```
add rd, rs, rt
Instruction = Mem[PC];
Reg[rd] = Reg[rs] + Reg[rt]
PC = PC + 4;
```



Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
R-type	000000	1	1	0	0	0	0	10

Control Unit: branch




```
Branch Instruction: beq rs, rt, imm
Instruction = Mem[PC];
Cond = (Reg[rs] - Reg[rt]) == 0;
                // Test equality
if (Cond) PC = PC + 4 + SignExtend(imm)*4;
       // Neg for backward; *4: LSbits == 00
else PC = PC + 4;
```

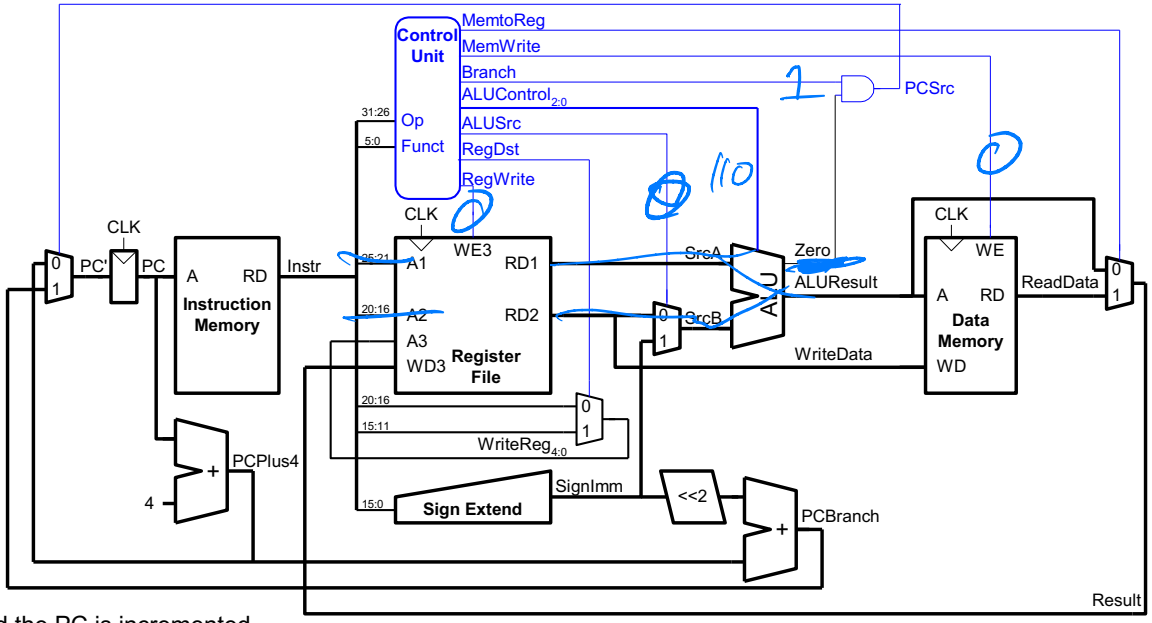
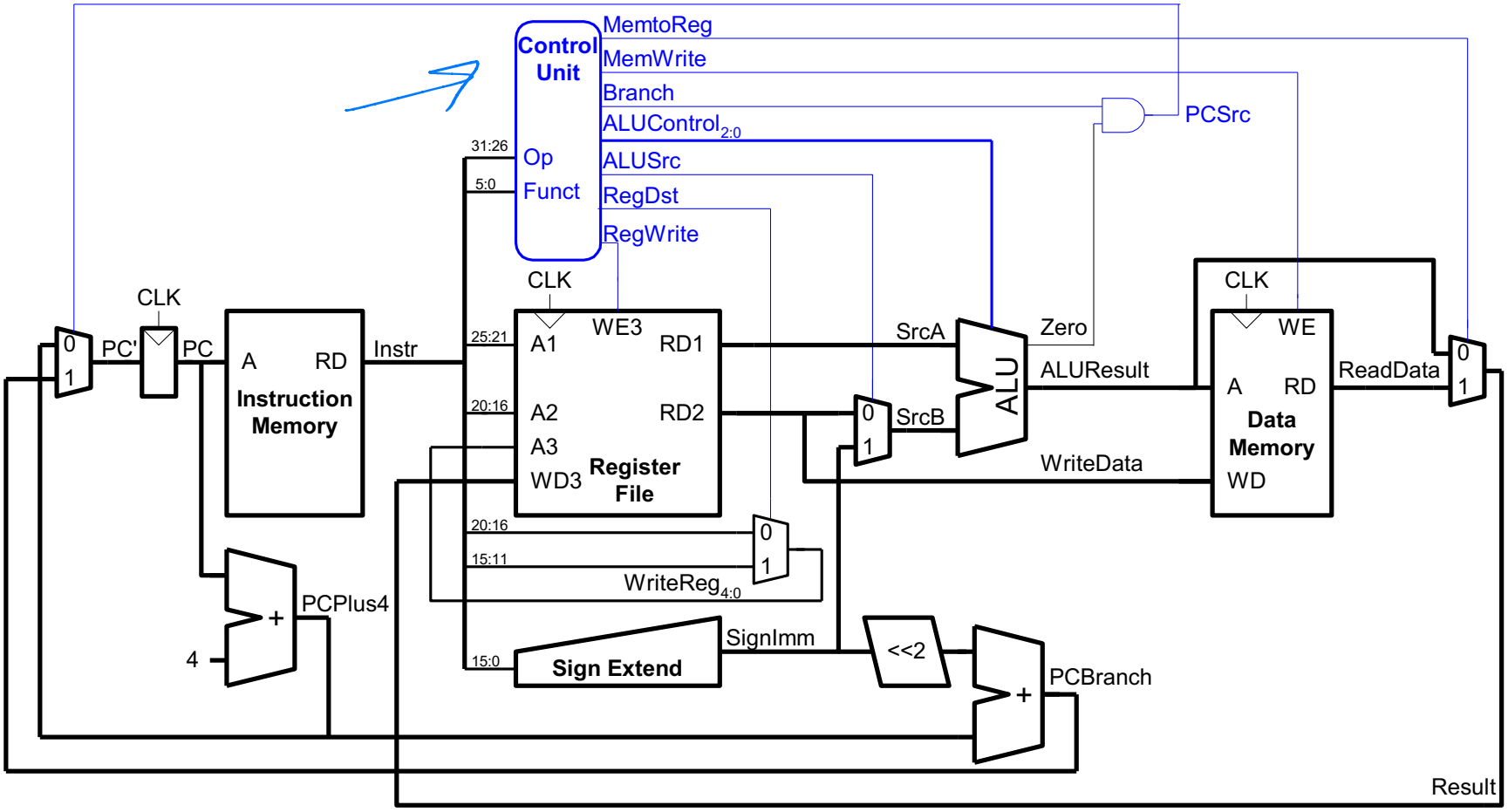


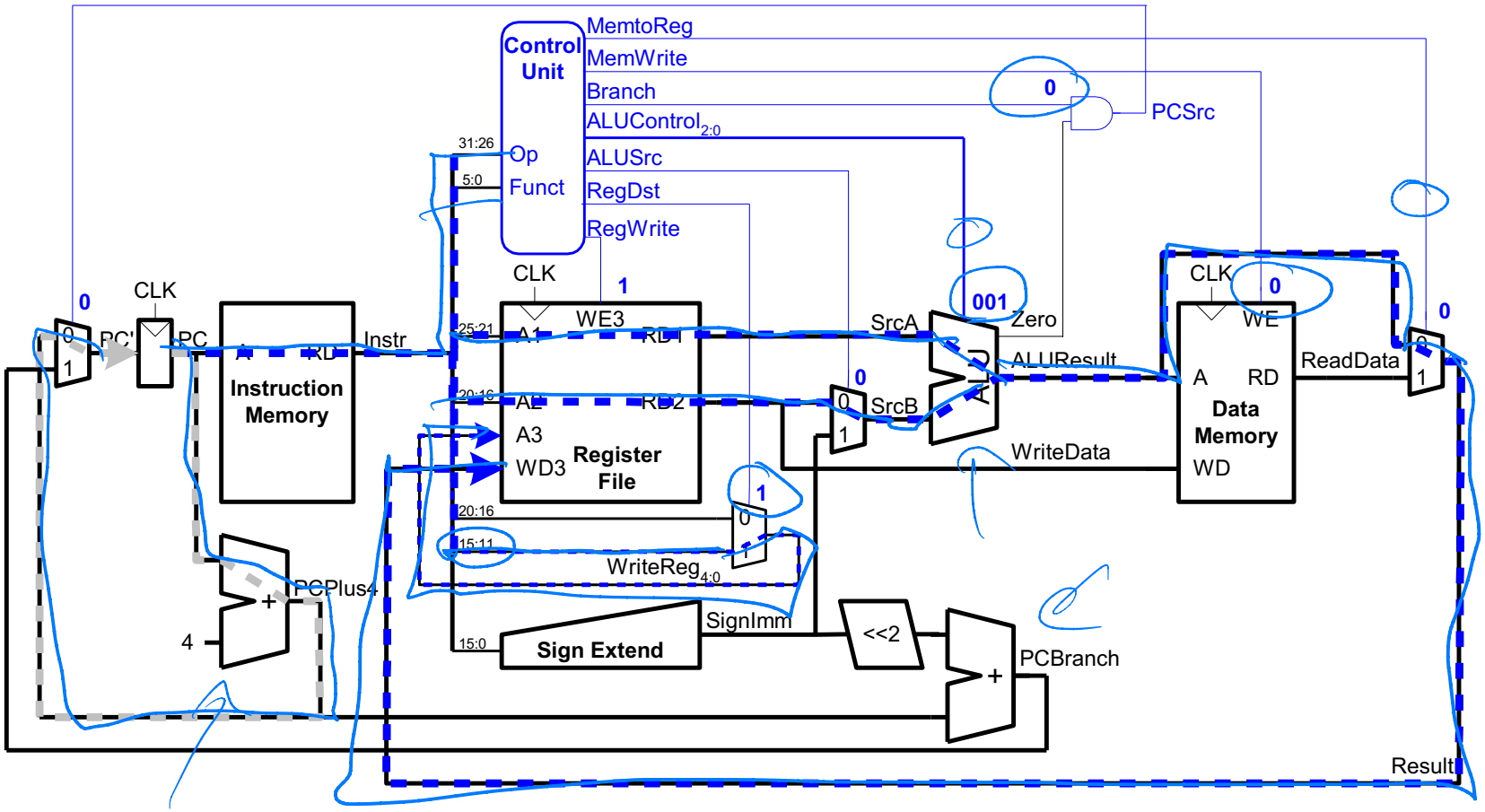
Fig.5.21
1.The instruction is fetched, and the PC is incremented
2. Two registers are read from the register file
3.The ALU performs a subtract on the data values read from the register file. The value of PC +4 is added to the sign-extended, lower 16 bits of the instruction (offset) shifted left by two; the result is the branch target address
4. The zero result from the ALU is used to decide which adder result to store into the PC

Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$
beq	000100	0	X	0	1	0	X	01
								7-<24>

Complete Single-Cycle Processor



Single-Cycle Datapath Example: `or`



Multicycle MIPS Processor

- Single-cycle microarchitecture:
  + simple
  - cycle time limited by longest instruction (`lw`)
  - two adders/ALUs and two memories
- Multicycle microarchitecture:
  + higher clock speed
  + simpler instructions run faster
  + reuse expensive hardware on multiple cycles
  - sequencing overhead paid many times
- Same design steps: datapath & control