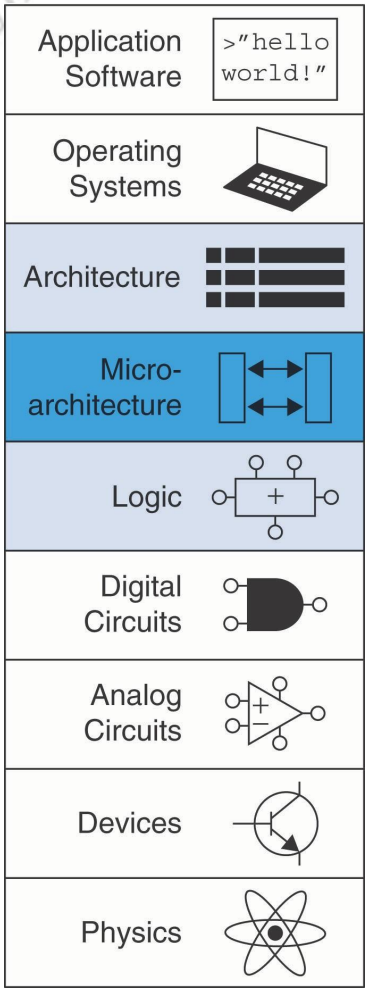
RISC V Microarchitecture

Dr. Girish H
Professor
Department of ECE
Cambridge Institute of Technology
&
Kavinesh
Research Staff
CCCIR
Cambridge Institute of Technology

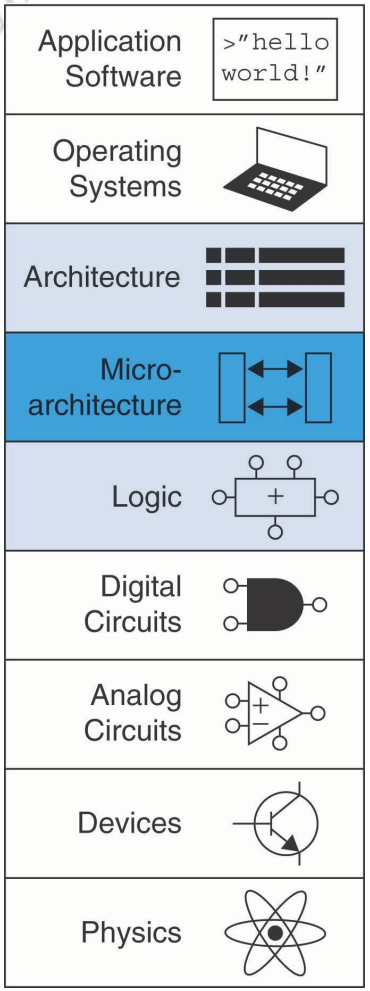
Microarchitecture

- **Introduction**
- **Single-Cycle Processor**



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 series of shorter steps
  - **Pipelined:** Each instruction broken up into series of steps & multiple instructions execute at once

Processor Performance

- **Program execution time**

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

- **Definitions:**
  - CPI: Cycles/instruction
  - clock period: seconds/cycle
  - IPC: instructions/cycle = IPC
- **Challenge is to satisfy constraints of:**
  - Cost
  - Power
  - Performance

RISC-V Processor

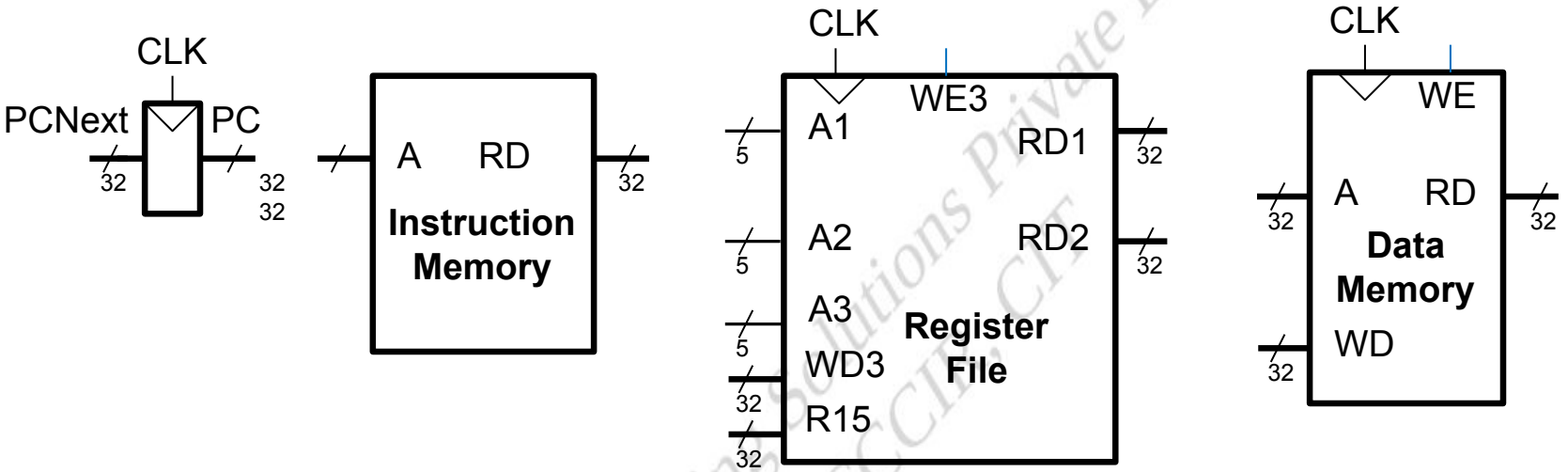
- Consider **subset** of RISC-V instructions:
  - **R-type instructions:**
    - `add`, `sub`, `and`, `or`, `slt`
  - **I-type instruction:**
    - `lw`
  - **S-type instruction:**
    - `sw`
  - **B-type instructions:**
    - `beq`

Architectural State Elements

Determines everything about a processor:

- Architectural state:
  - 32 registers
  - PC
  - Memory

RISC-V Architectural State Elements



Single-Cycle RISC-V Processor

- Datapath
- Control

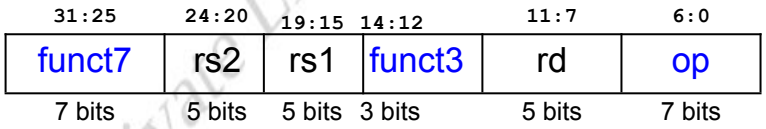
Single-Cycle RISC-V Processor

- **Datapath**
- Control

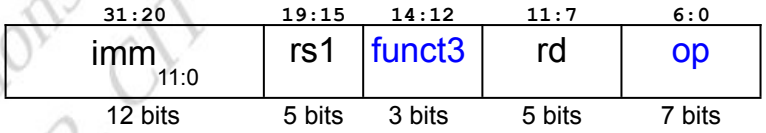
RISC-V Processor

- **R-type instructions:**
  - `add, sub, and, or, slt`
- **I-type instruction:**
  - `lw`
- **S-type instruction:**
  - `sw`
- **B-type instructions:**
  - `beq`

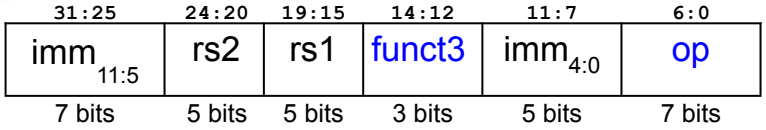
R-Type



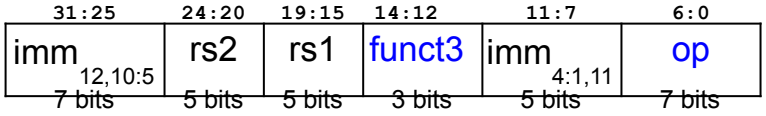
I-Type



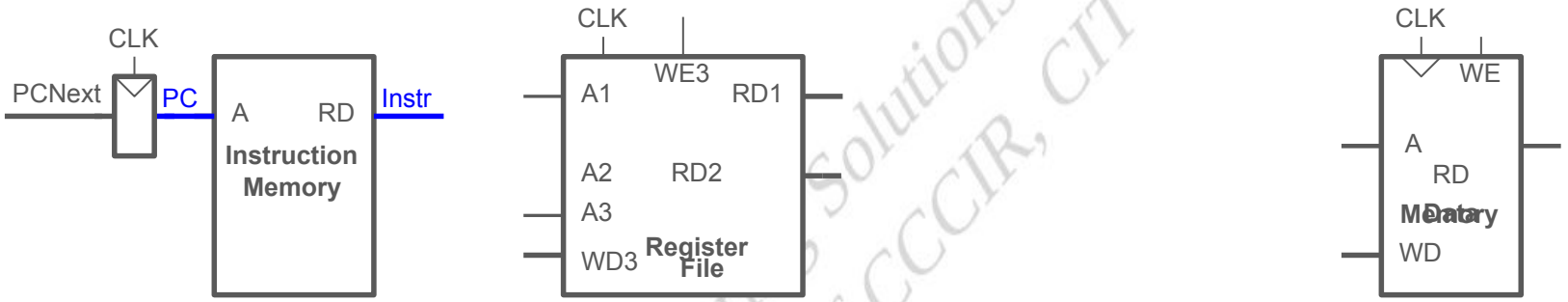
S-Type



B-Type



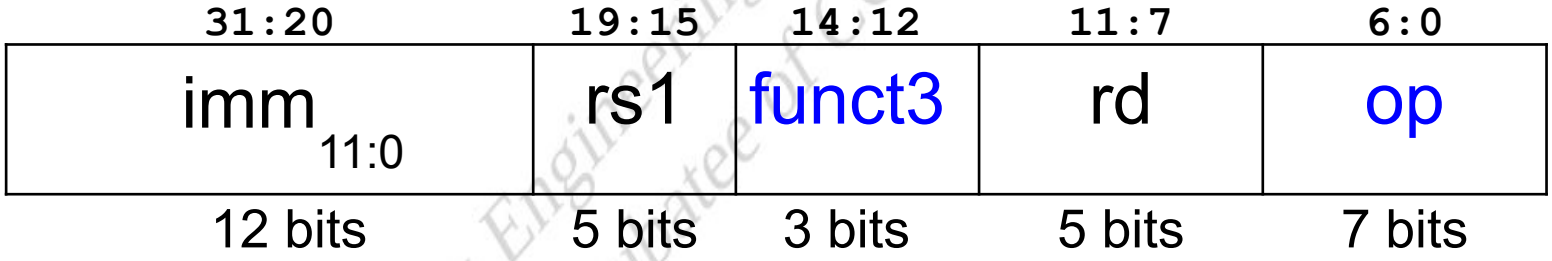
Single-Cycle RISC-V Datapath



Single-Cycle RISC-V Processor

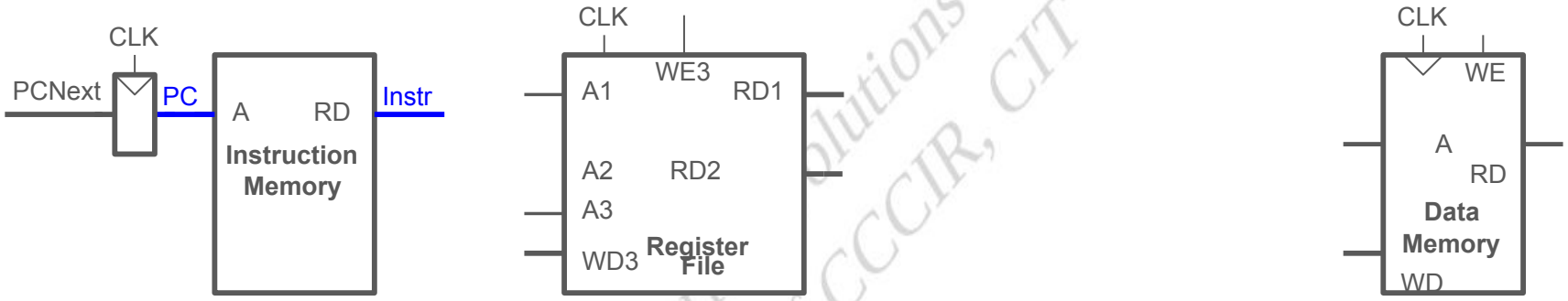
- **Datapath:** start with `lw` instruction `lw t2, -8(s3)`
  - **Example:** **`lw rd, imm(rs1)`**

I-Type



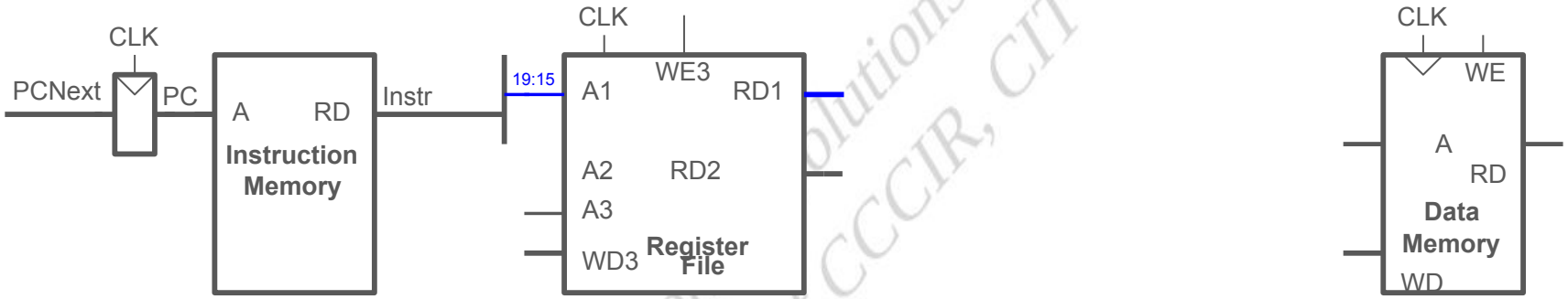
Single-Cycle Datapath: `lw` fetch

STEP 1: Fetch instruction

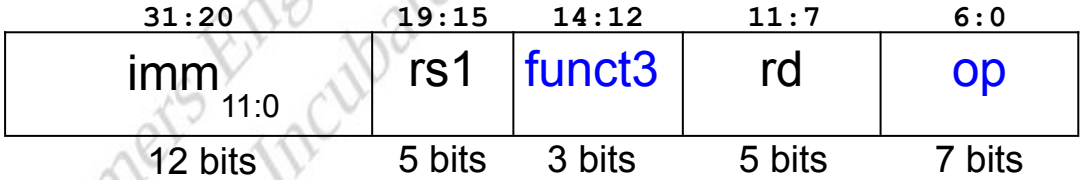


Single-Cycle Datapath: `lw` Reg Read

STEP 2: Read source operand (**rs1**) from RF



I-Type

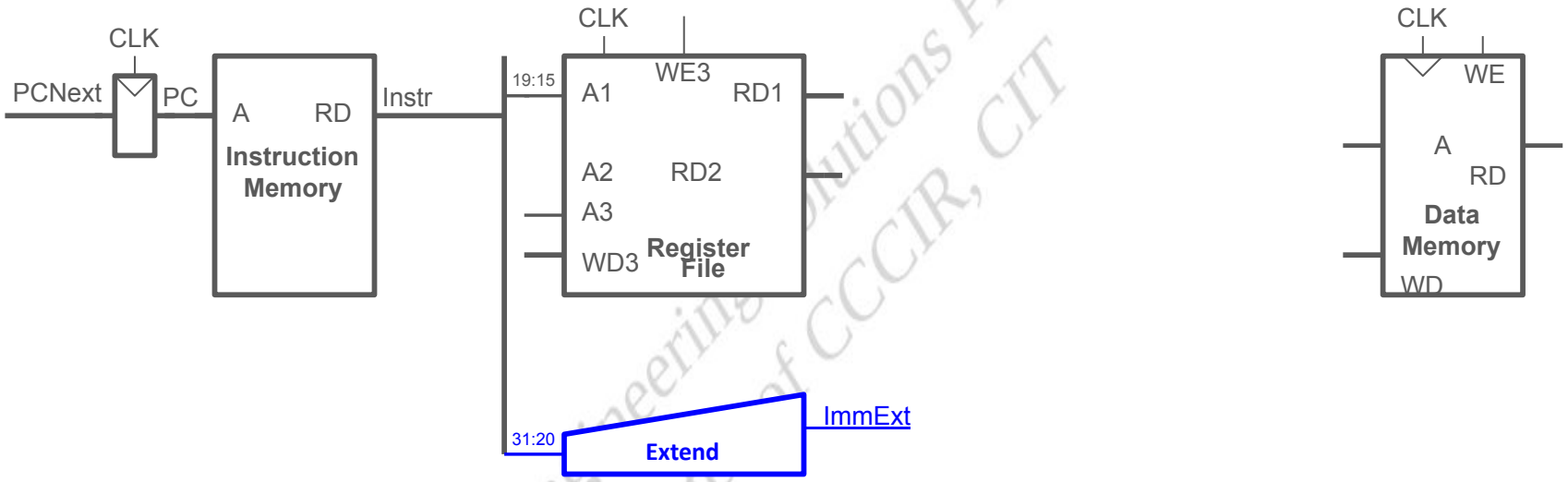



```
lw rd, imm(rs1)
```

Single-Cycle Datapath: `lw`

Immediate

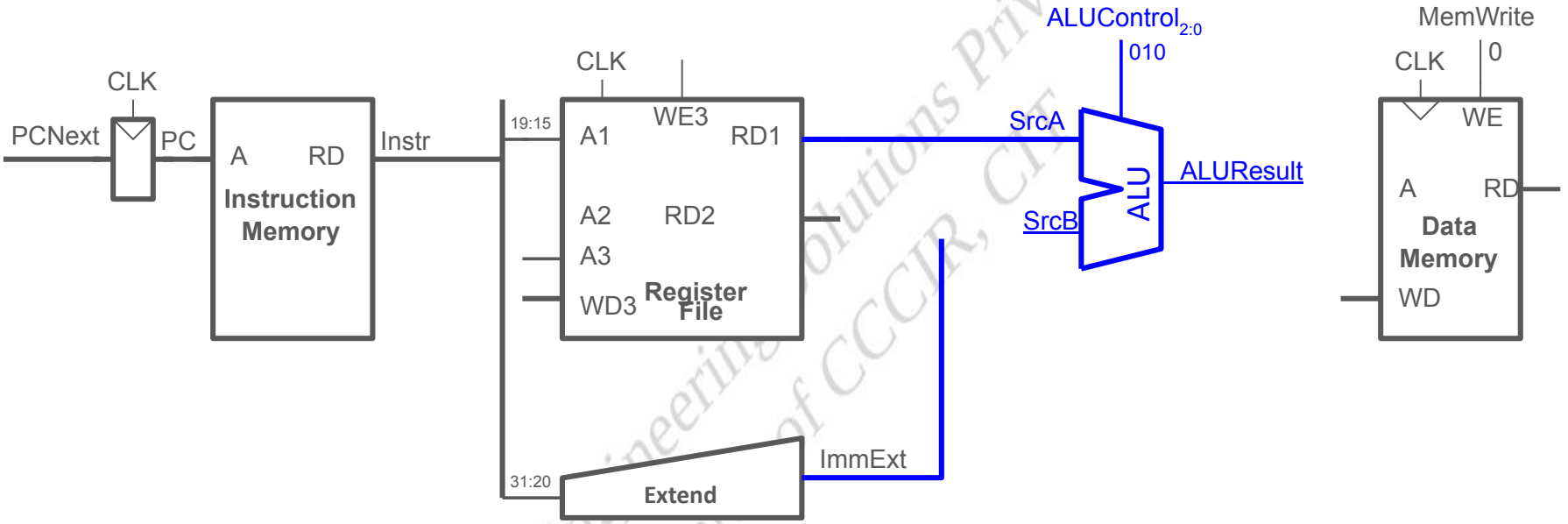
STEP 3: Extend the immediate



Single-Cycle Datapath: `lw`

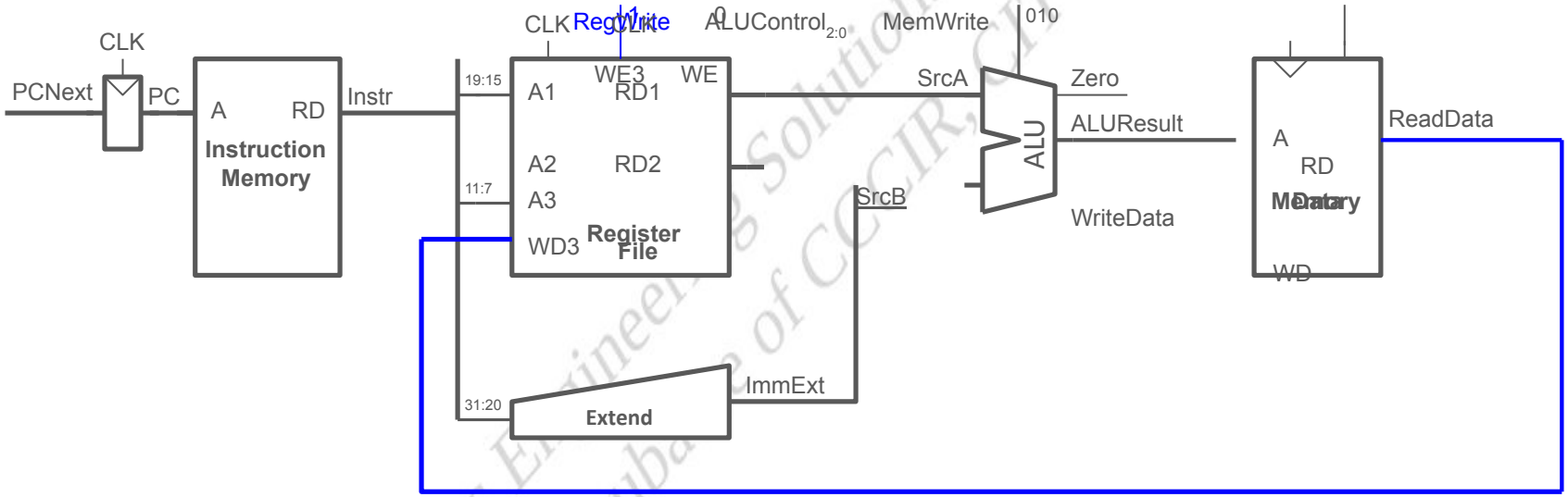
Address

STEP 4: Compute the memory address



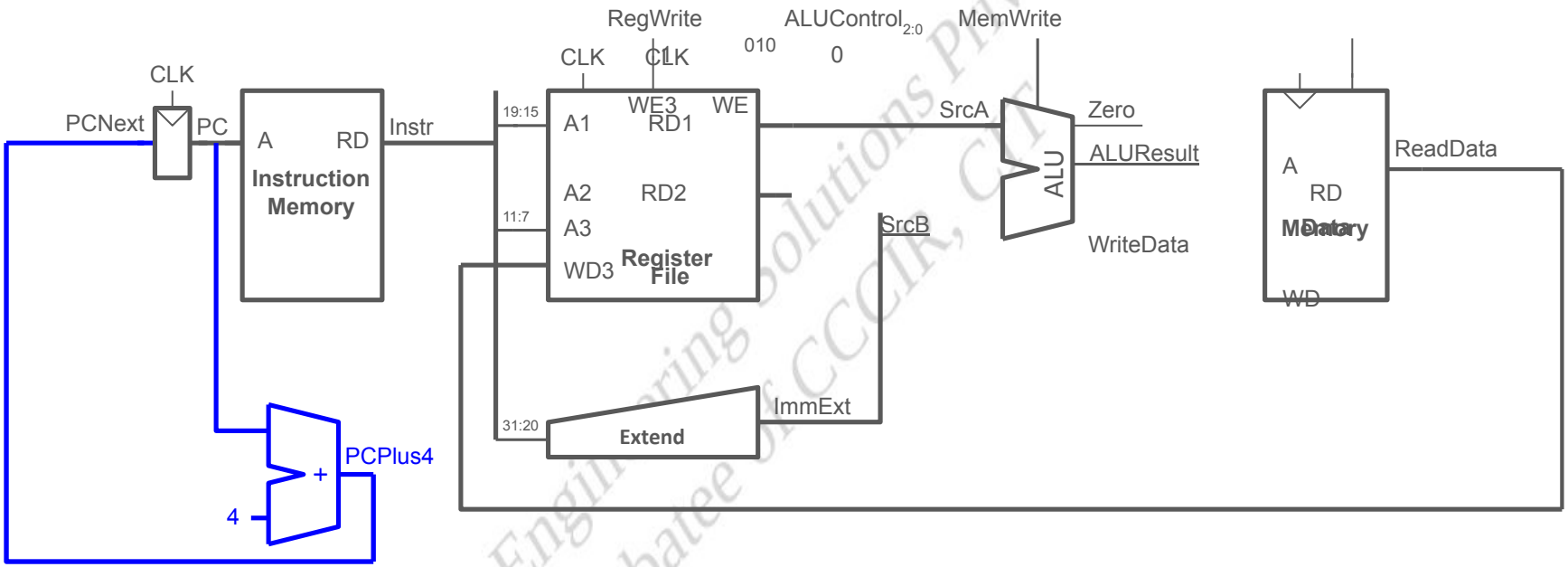
Single-Cycle Datapath: LDR Mem Read

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



Single-Cycle Datapath: PC Increment

STEP 6: Determine address of next instruction



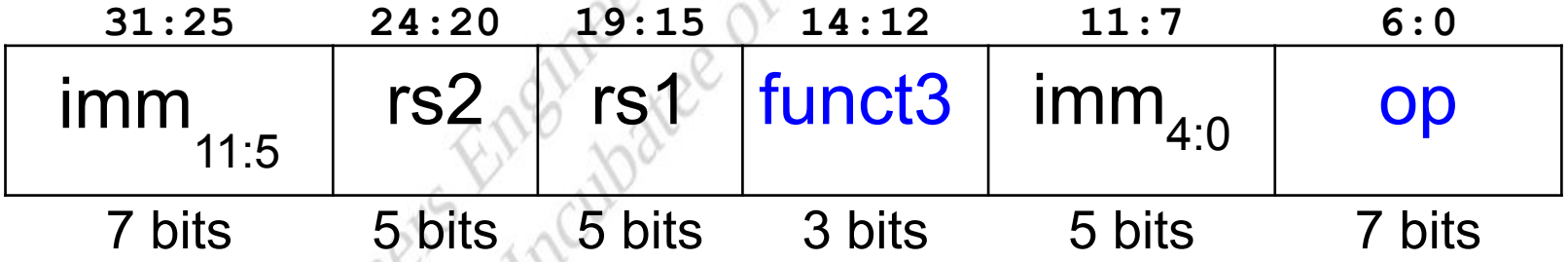
Single-Cycle Datapath: `sw`

Expand datapath to handle `sw`:

- Write data in `rs2` to memory
- **Example:** `sw t2, 0xc(s3)`

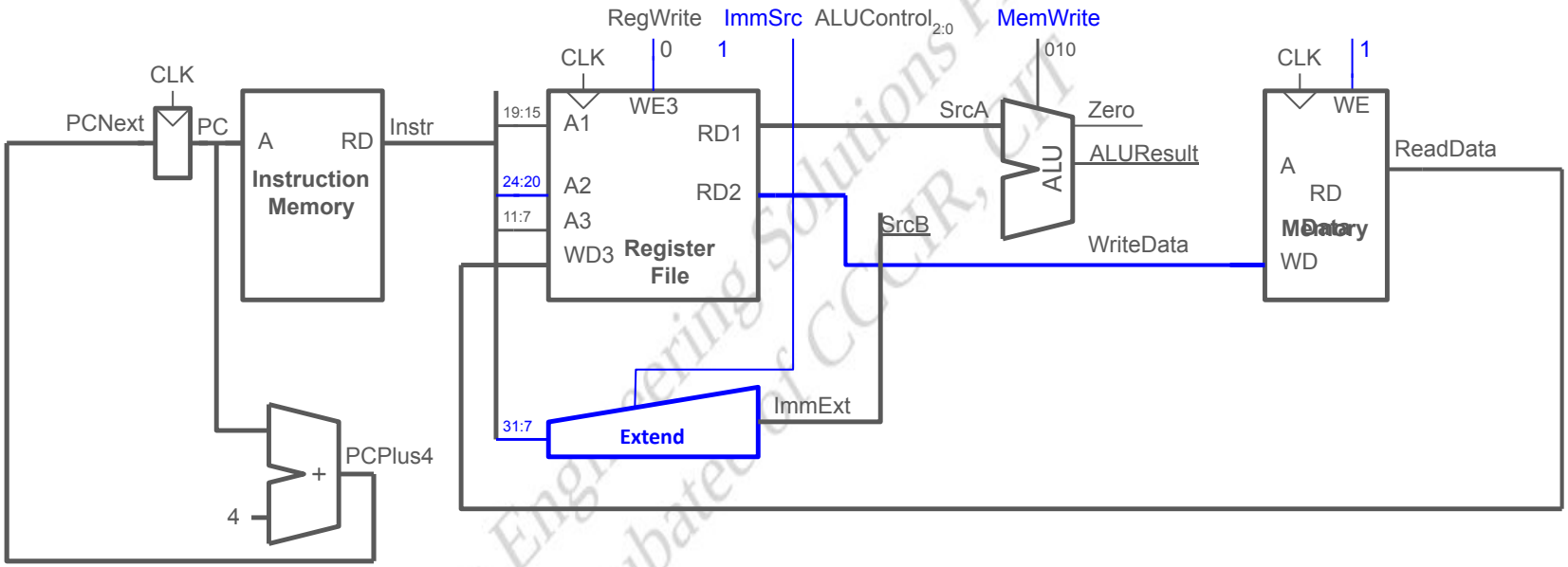
  `sw rs2, imm(rs1)`

S-Type



Single-Cycle Datapath: Data-processing

- **Immediate:** now in {instr[31:25], instr[11:7]}
- **Add control signals:** ImmSrc, MemWrite



S-Type

31:25	24:20	19:15	14:12	11:7	6:0
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op

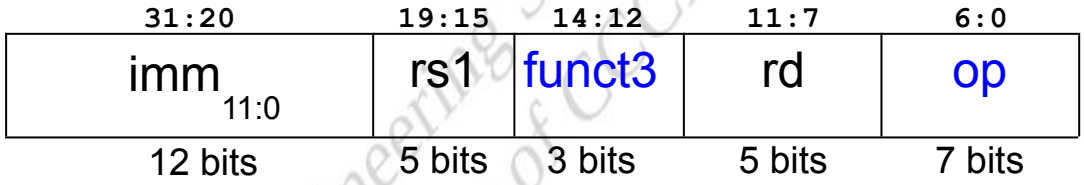
7 bits 5 bits 5 bits 3 bits 5 bits 7 bits

```
sw rs2, imm(rs1)
```

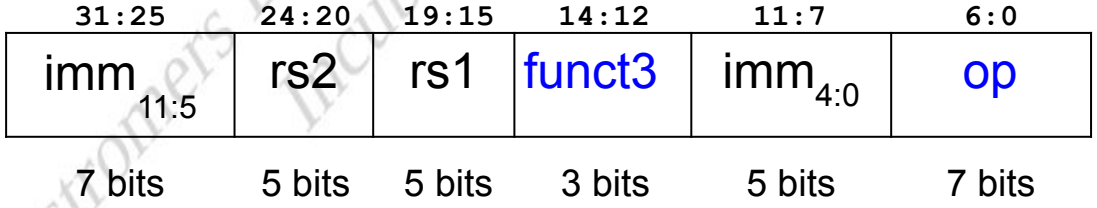
Single-Cycle Datapath: Immediate

ImmSrc	ImmExt	Instruction Type
0	{{20{instr[31]}}, instr[31:20]}	I-Type
1	{{20{instr[31]}}, instr[31:25], instr[11:7]}	S-Type

I-Type



S-Type

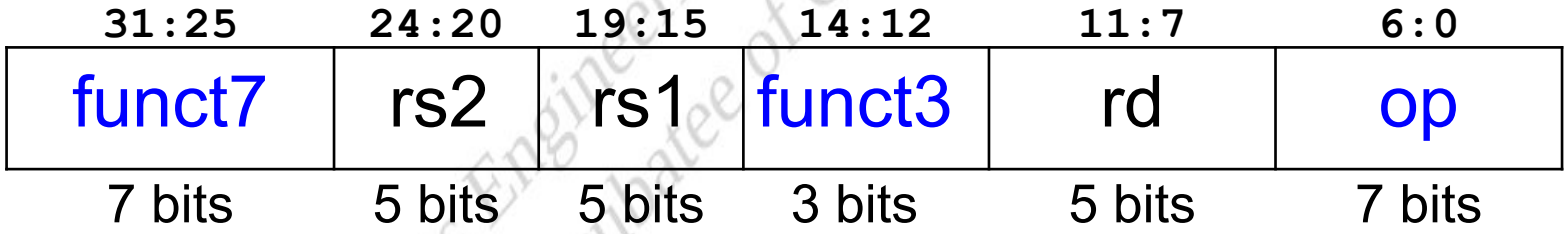


Single-Cycle Datapath: R-Type

- **Instructions:** `add`, `sub`, `and`, `or`, `slt`,
- **Example:** `add s1, s2, s3`

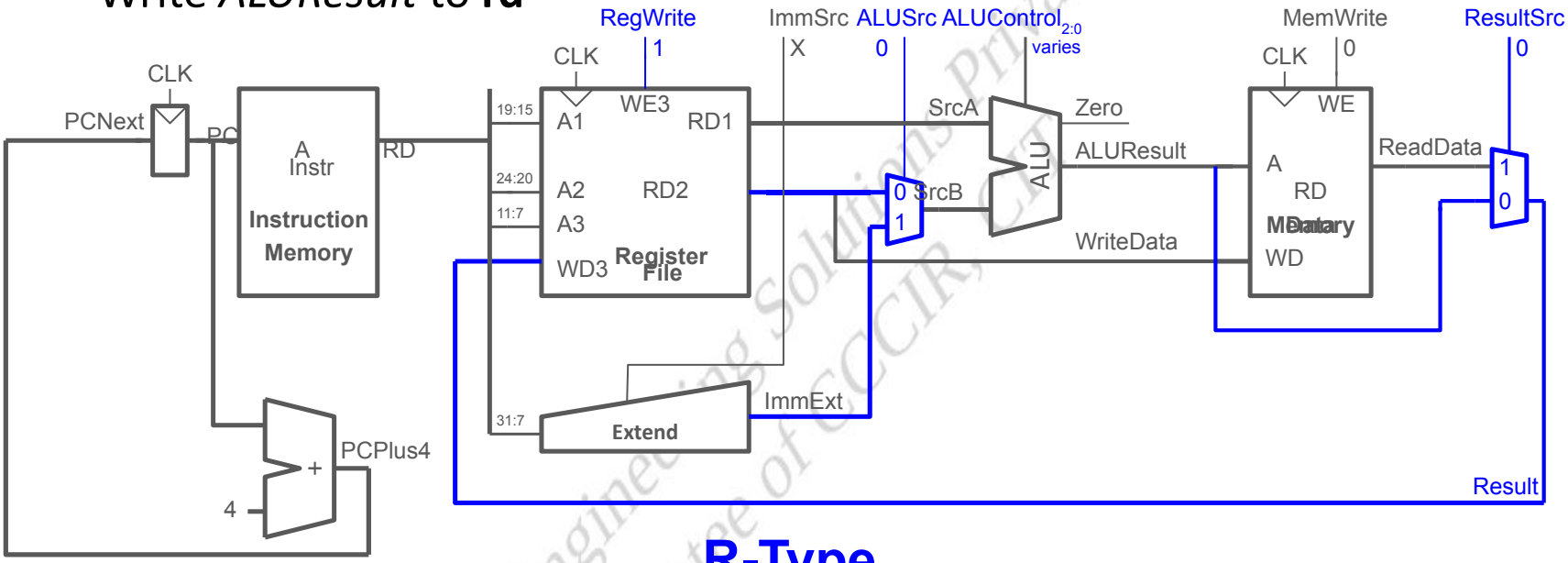
  `op rd, rs1, rs2`

R-Type



Single-Cycle Datapath: R-Type

- Read from **rs1** and **rs2** (instead of **imm**)
- Write *ALUResult* to **rd**



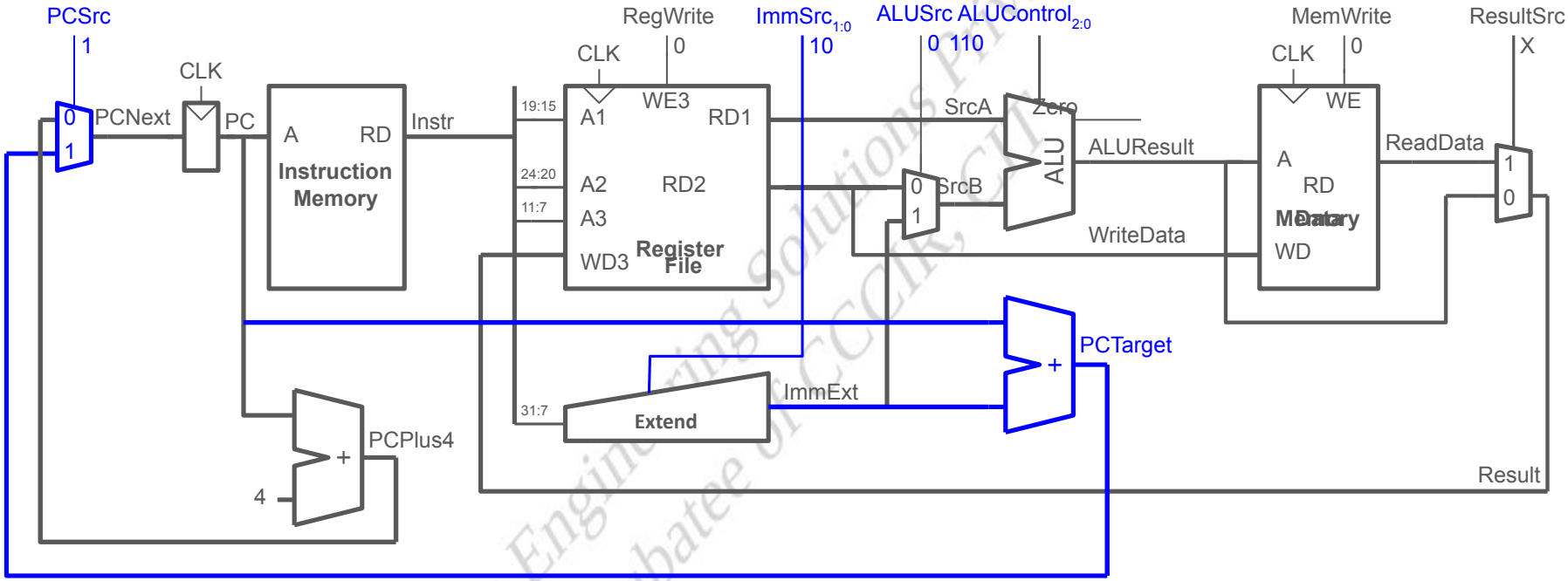
R-Type

31:25	24:20	19:15	14:12	11:7	6:0
funct7	rs2	rs1	funct3	rd	op
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

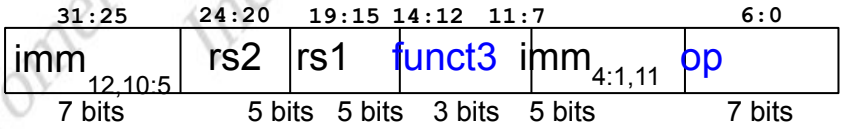
```
add rd, rs1, rs2
```

Single-Cycle Datapath: beq

Calculate branch target address: PCTarget = PC + imm



B-Type

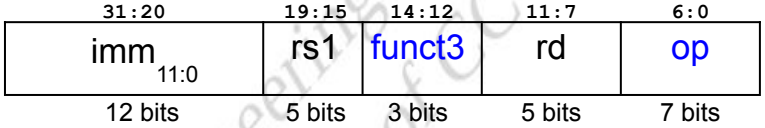



```
beq rs1, rs2, Label
```

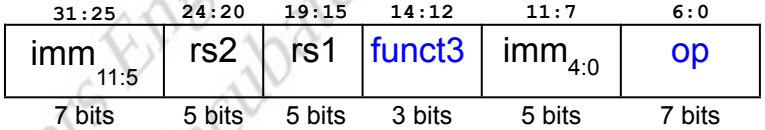
Single-Cycle Datapath: ImmExt

$ImmSrc_{1:0}$	ImmExt	Instruction Type
00	{{20{instr[31]}}, instr[31:20]}	I-Type
01	{{20{instr[31]}}, instr[31:25], instr[11:7]}	S-Type
10	{{19{instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1'b0}	B-Type

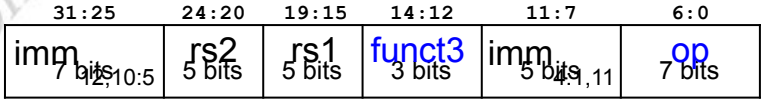
I-Type



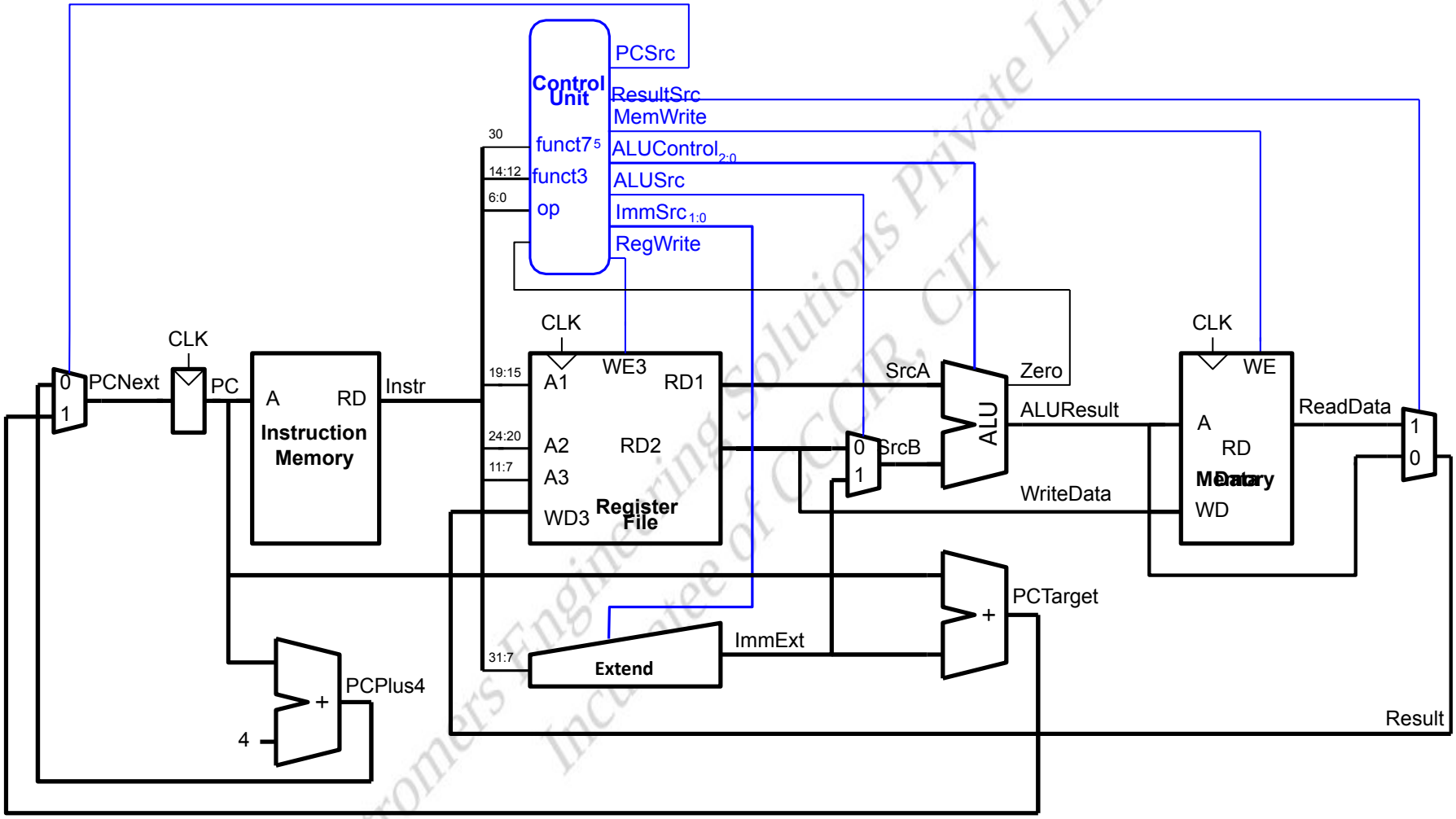
S-Type



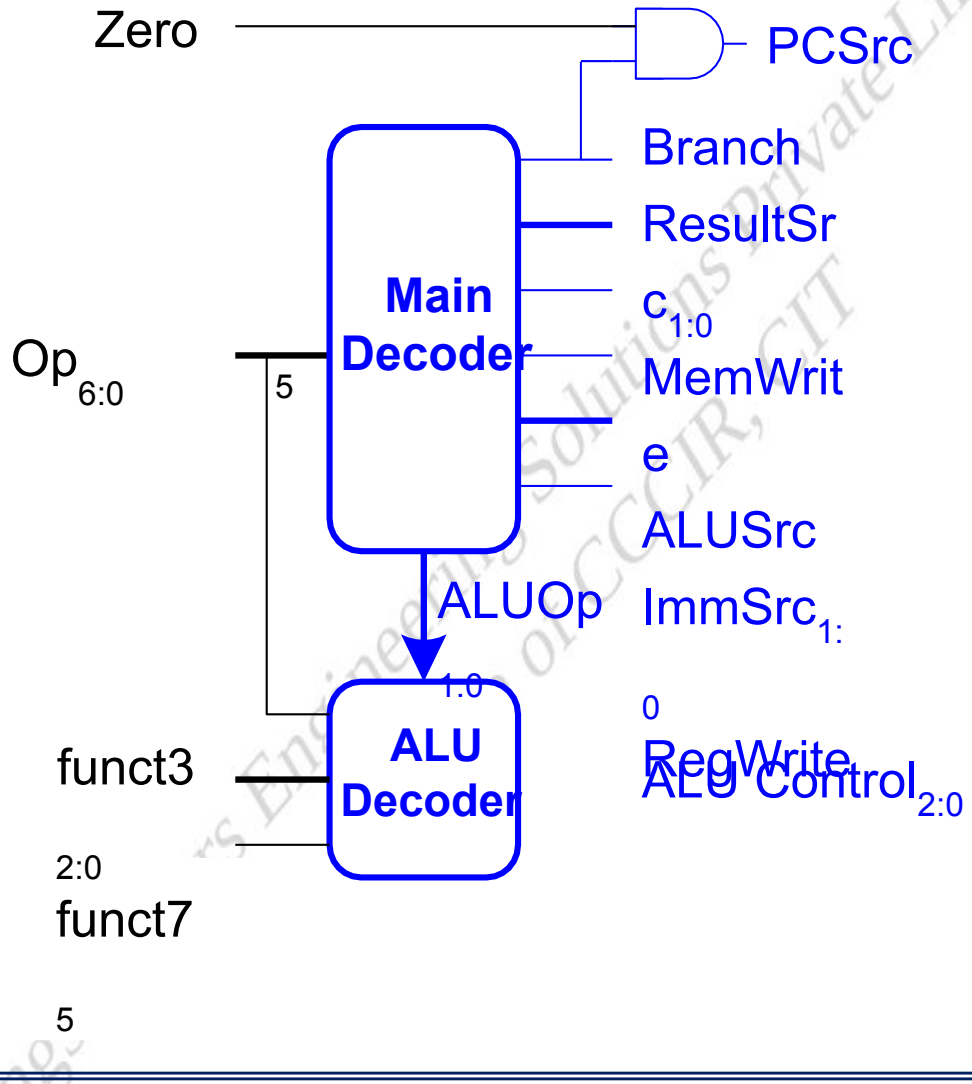
B-Type



Single-Cycle RISC-V Processor

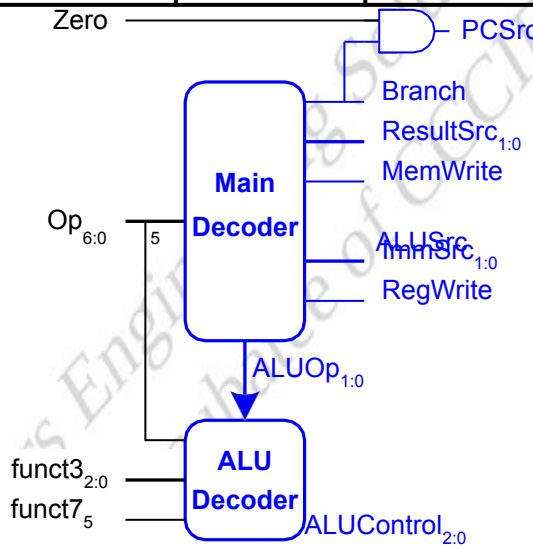


Single-Cycle Control

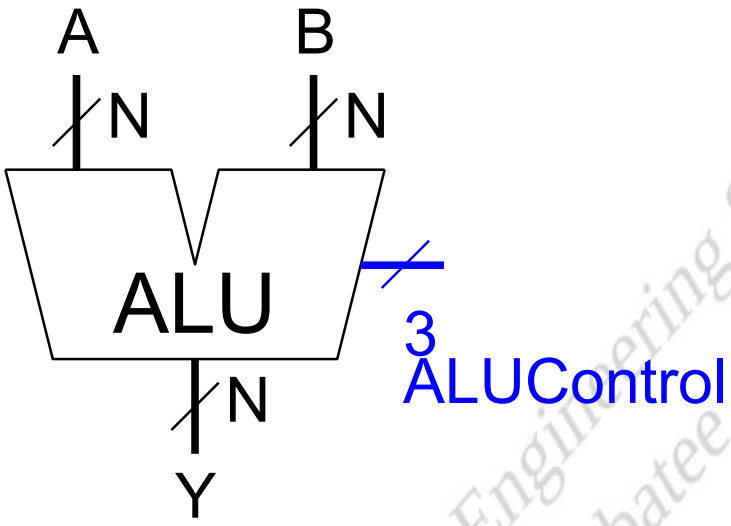


Control Unit: Main Decoder

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	`lw`	1	00	1	0	1	0	00
35	`sw`	0	01	1	1	X	0	00
51	**R-type**	1	XX	0	0	0	0	10
99	`beq`	0	10	0	0	X	1	01

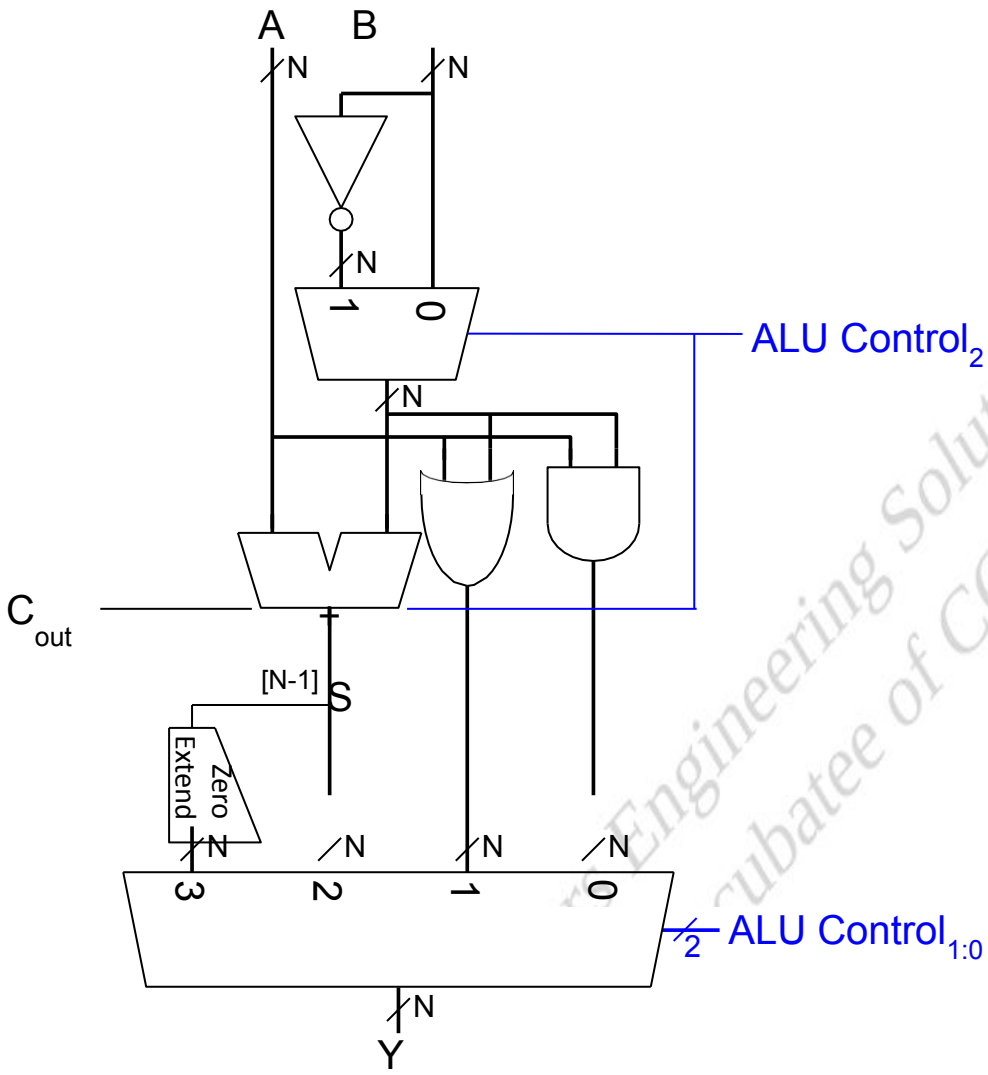


Review: ALU



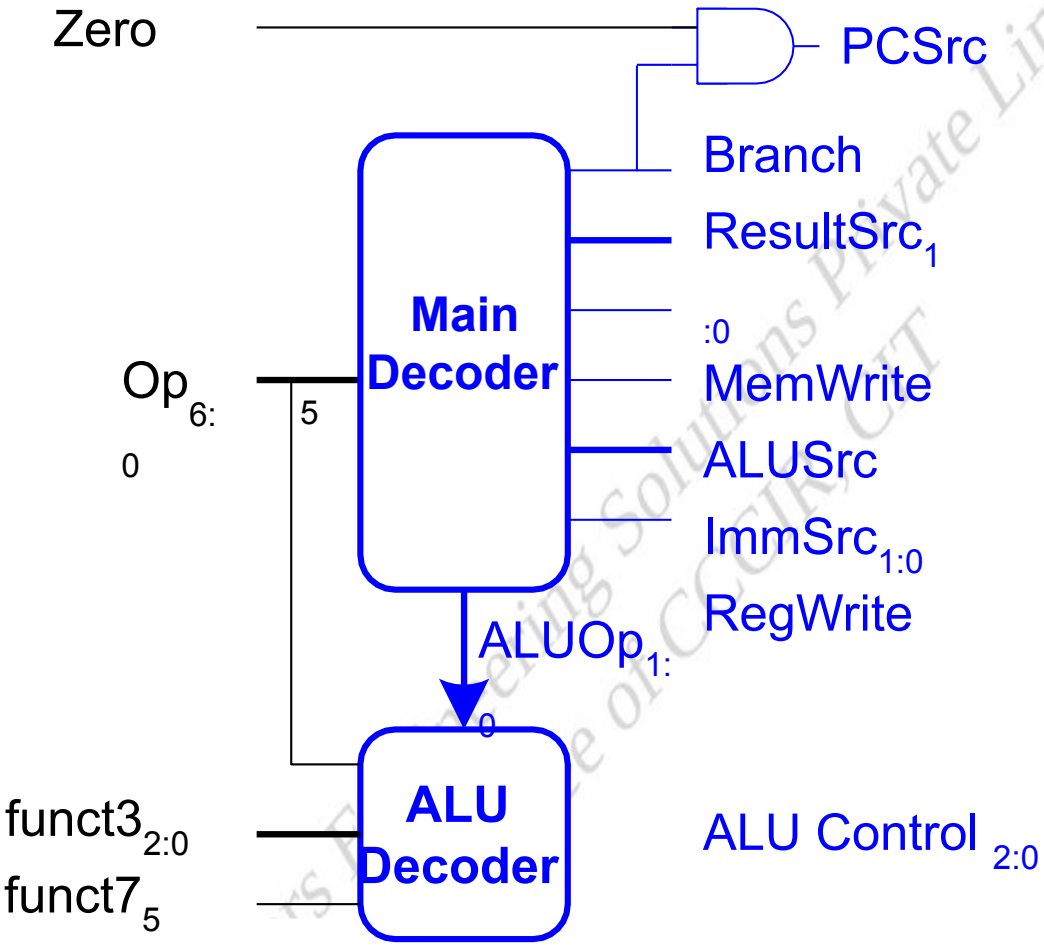
$ALUControl_{2:0}$	Function
000	A & B
001	A | B
010	A + B
110	A - B
111	SLT

Review: ALU



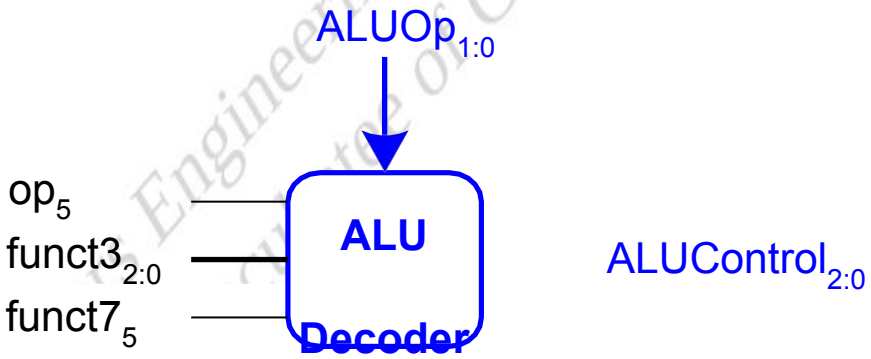
ALU Control$_{2:0}$	Function
000	A & B
001	A | B
010	A + B
110	A - B
111	SLT

Single-Cycle Control: ALU Decoder



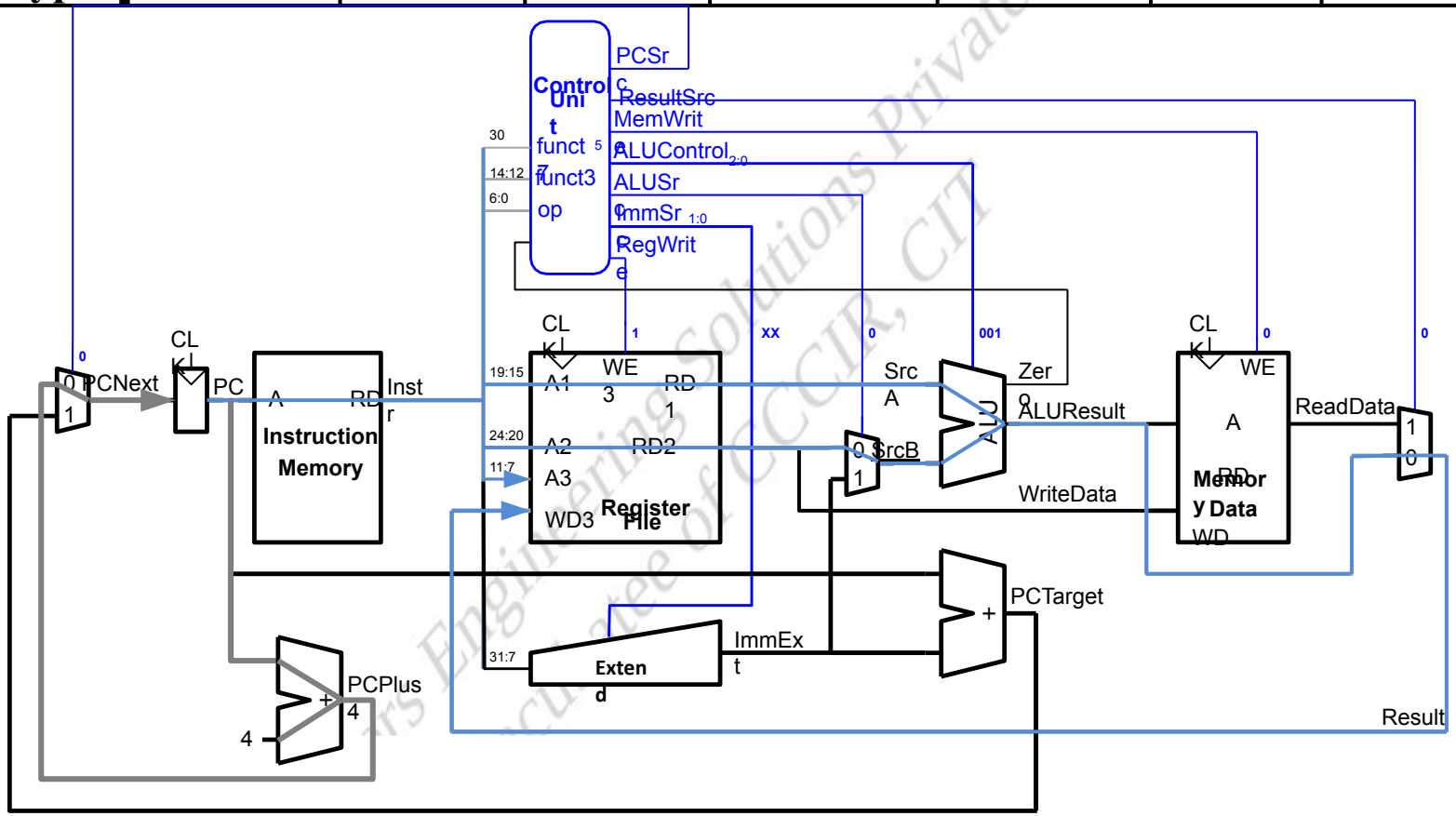
Control Unit: ALU Decoder

ALUOp	op_5	funct3	$funct7_5$	Instruction	$ALUControl_{2:0}$
00	X	X	X	`lw, sw`	010 (add)
01	X	X	X	`beq`	110 (subtract)
10	X	000	0	`add`	010 (add)
	1	000	1	`sub`	110 (subtract)
	X	010	0	`slt`	111 (set less than)
	X	110	0	`or`	001 (or)
	X	111	0	`Slt`	000 (and)

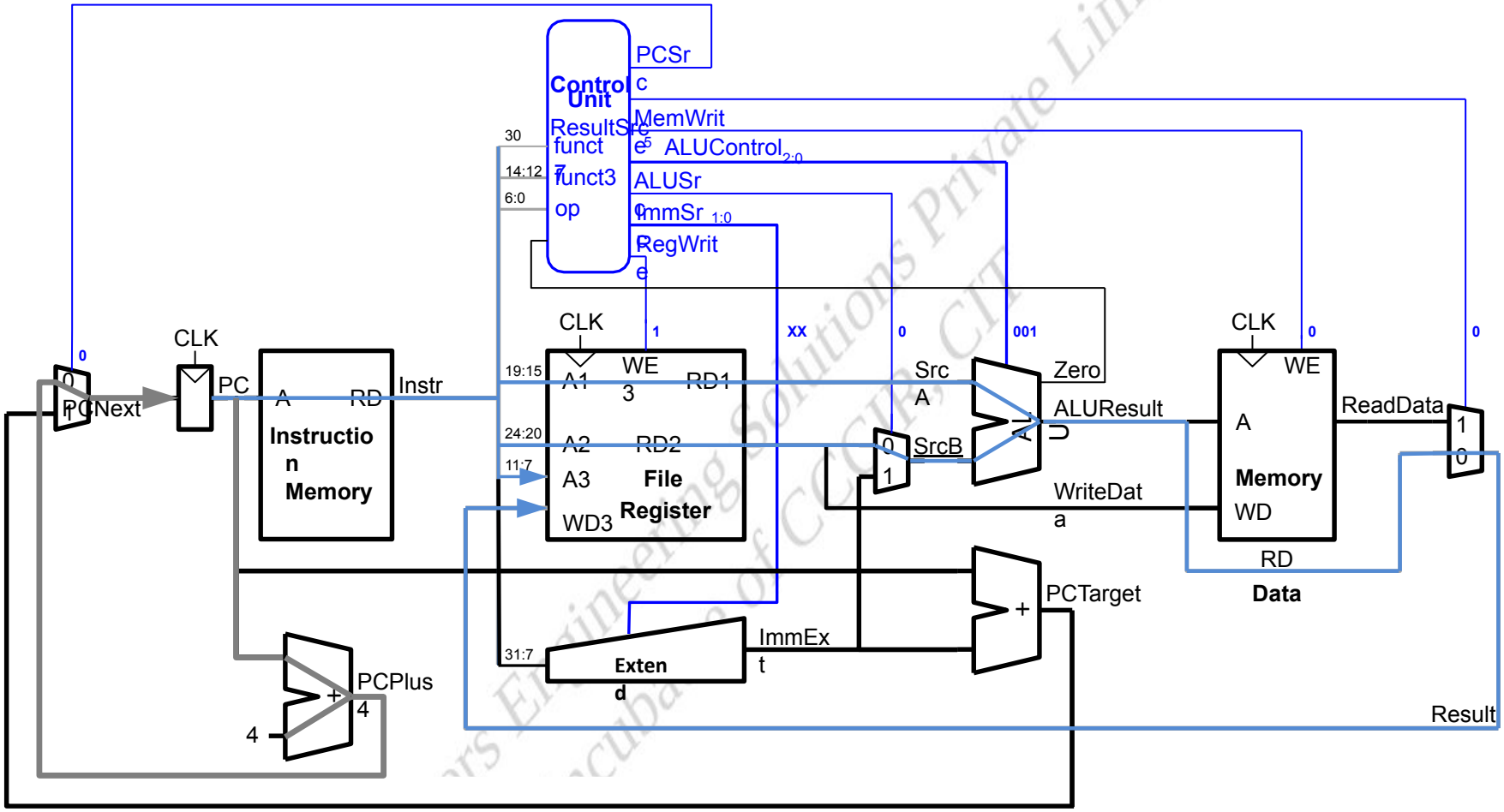


Example: `or`

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
51	**R-type**	1	XX	0	0	0	0	10



Example: `or`

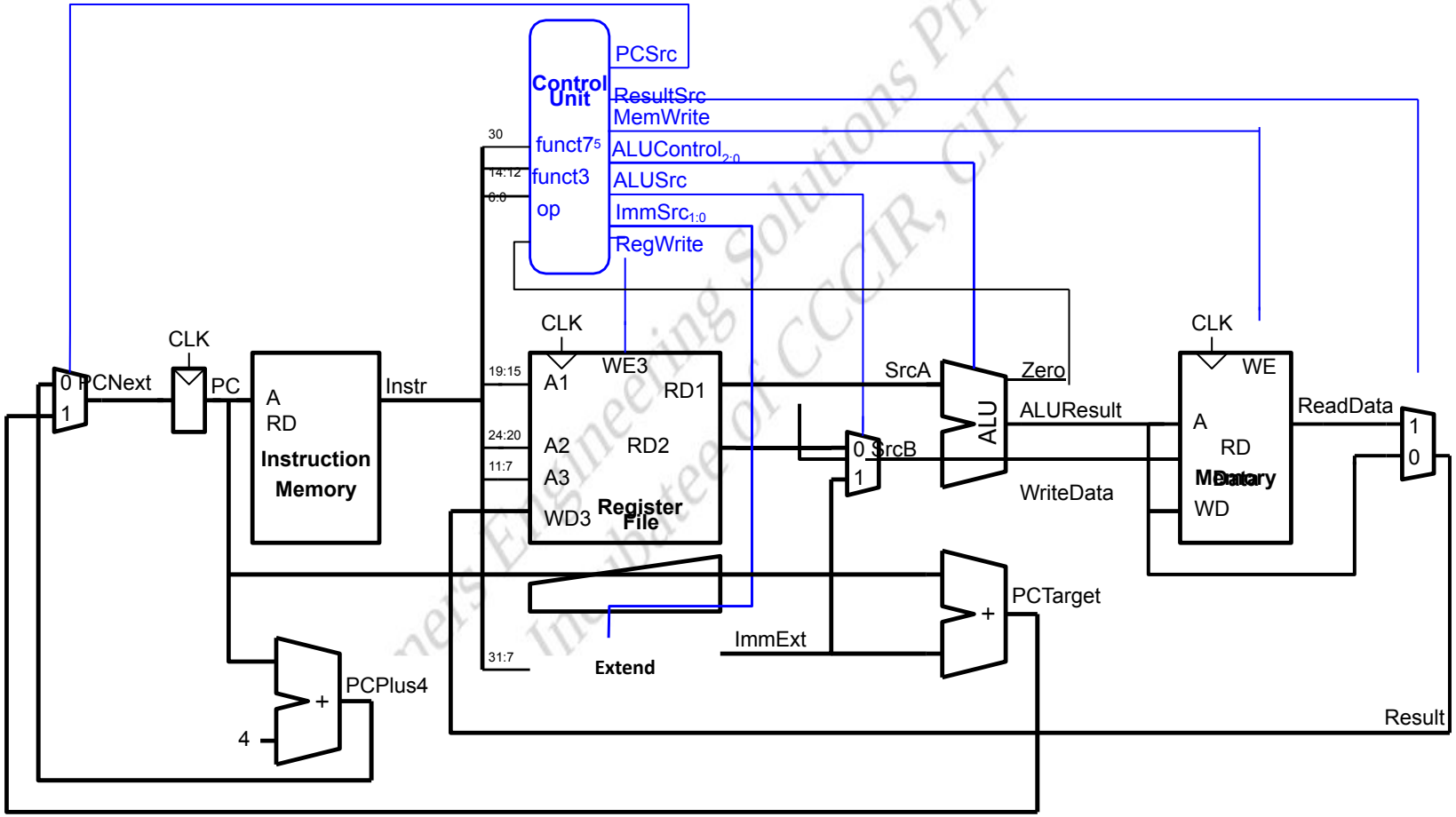


Extended Functionality: `addi`

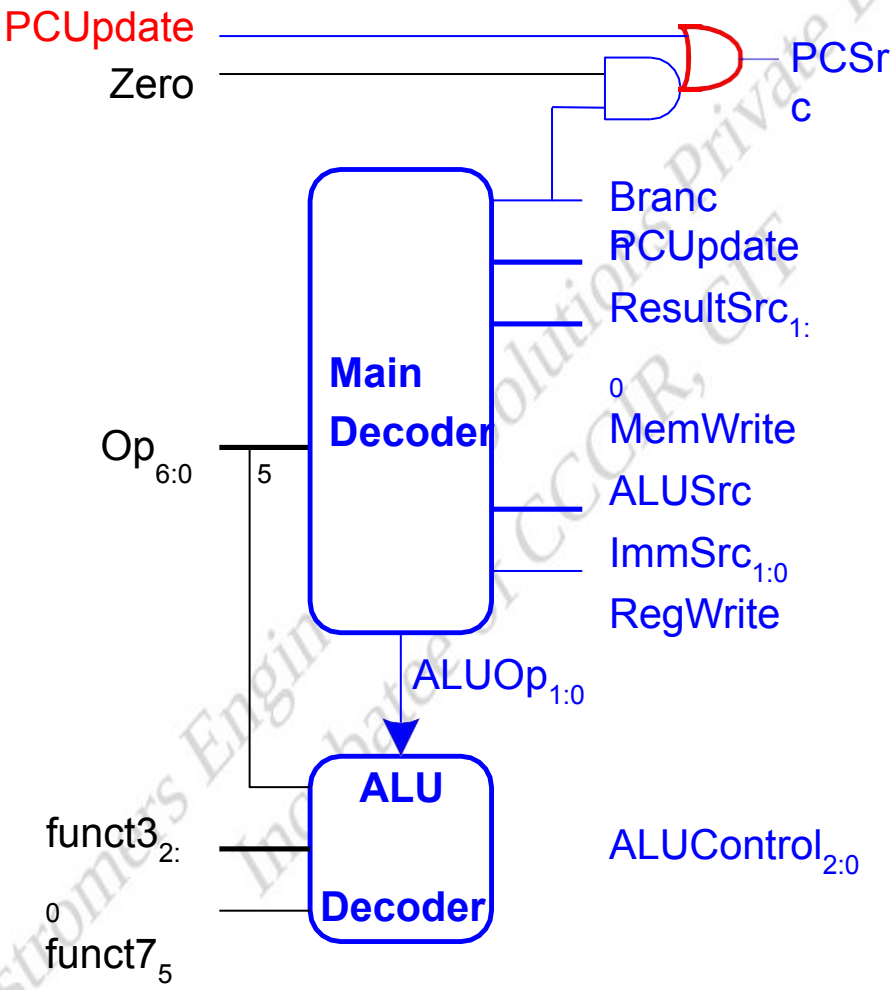
op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
3	**`lw`**	1	00	1	0	1	0	00
35	**`sw`**	0	01	1	1	X	0	00
51	**R-type**	1	XX	0	0	0	0	10
99	**`beq`**	0	10	0	0	X	1	01
19	**`addi`**	**1**	**00**	**1**	**0**	**0**	**0**	**10**

Extended Functionality: addi

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp
19	addi	1	00	1	0	0	0	10



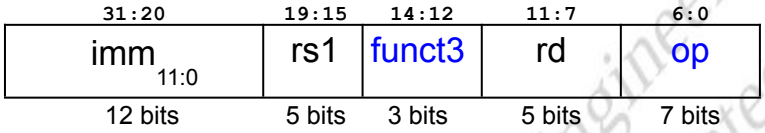
Extended Functionality: `jal`



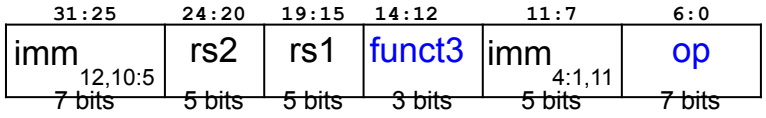
Single-Cycle Datapath: ImmExt

$ImmSrc_{1:0}$	ImmExt	Instruction Type
00	{{20{instr[31]}}, instr[31:20]}	I-Type
01	{{20{instr[31]}}, instr[31:25], instr[11:7]}	S-Type
10	{{19{instr[31]}}, instr[31], instr[7], instr[30:25], instr[11:8], 1'b0}	B-Type
11	**{{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0}**	**J-Type**

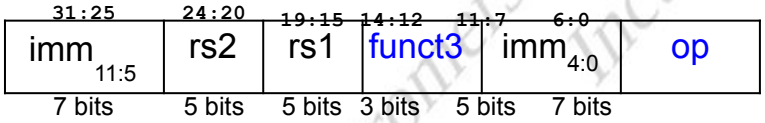
I-Type



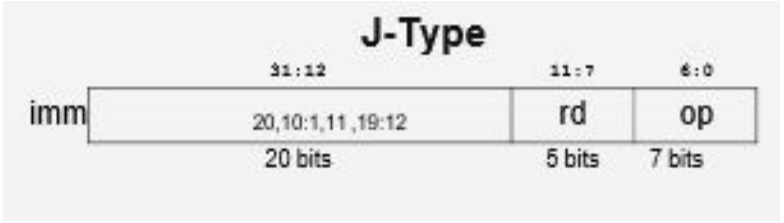
B-Type



S-Type



J-Type



Extended Functionality: `jal`

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	PCUpdate
3	`lw`	1	00	1	0	10	0	00	**0**
35	`sw`	0	01	1	1	XX	0	00	**0**
51	**R-type**	1	XX	0	0	01	0	10	**0**
99	`beq`	0	10	0	0	XX	1	01	**0**
19	`addi`	1	00	1	0	01	0	10	**0**
111	`jal`	**0**	**11**	**X**	**0**	**00**	**0**	**XX**	**1**

Extended Functionality: `jal`

op	Instruct	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	PCUpdate
111	**jal**	0	11	X	0	00	0	XX	1