ECE 281

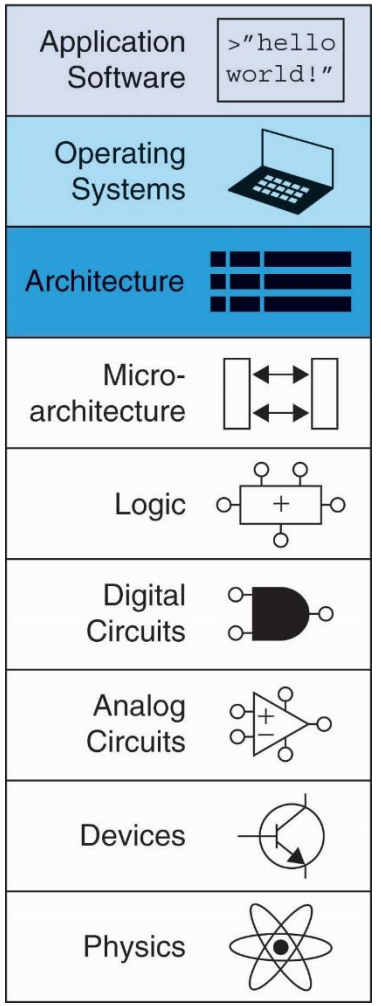
Lesson 21:
RV32I Registers

Digital Design and Computer Architecture Lecture Notes

Modified for use by USAF Academy, 2025

Introduction

- **Architecture:** programmer's view of computer
  - Defined by instructions & operand locations
- **Microarchitecture:** how to implement an architecture in hardware
- **Instructions :** commands in a computer's language
  - **Assembly language:** human-readable format of instructions
  - **Machine language:** computer-readable format (1's and 0's)



•2: Instructions
•3: Operands
•15: Machine Language: R-Type Instruction Formats
•16: Machine Language: I, S/B, U/J-Type Instr. Formats

Operands

- **Operand location:** physical location in computer
  - Registers
    - RV32I has thirty-two 32-bit registers
  - Memory
  - Constants (also called *immediates*)

RISC-V Register Set

Name	Register Number	Usage
zero	x0	Constant value 0
ra	x1	Return address
sp	x2	Stack pointer
gp	x3	Global pointer
tp	x4	Thread pointer
t0-2	x5-7	Temporaries
s0/fp	x8	Saved register / Frame pointer
s1	x9	Saved register
a0-1	x10-11	Function arguments / return values
a2-7	x12-17	Function arguments
s2-11	x18-27	Saved registers
t3-6	x28-31	Temporaries

Chapter 6: Architecture

Machine Language

Machine Language

- Binary representation of instructions
- Computers only understand 1's and 0's
- 32-bit instructions
  - Simplicity favors regularity: 32-bit data & instructions
- **4 Types of Instruction Formats:**
  - R-Type
  - I-Type
  - S/B-Type
  - U/J-Type

Review: Instruction Formats

7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
funct7	rs2	rs1	funct3	rd	op	**R-Type**
$imm_{11:0}$		rs1	funct3	rd	op	**I-Type**
$imm_{11:5}$	rs2	rs1	funct3	$imm_{4:0}$	op	**S-Type**
$imm_{12,10:5}$	rs2	rs1	funct3	$imm_{4:1,11}$	op	**B-Type**
$imm_{31:12}$				rd	op	**U-Type**
$imm_{20,10:1,11,19:12}$				rd	op	**J-Type**
20 bits				5 bits	7 bits	

R-Type Examples

Assembly	Field Values						Machine Code						
	funct7	rs2	rs1	funct3	rd	op	funct7	rs2	rs1	funct3	rd	op	
`add s2, s3, s4` `add x18, x19, x20`	0	20	19	0	18	51	0000 000	10100	10011	000	10010	011 0011	(0x01498933)
`sub t0, t1, t2` `sub x5, x6, x7`	32	7	6	0	5	51	0100 000	00111	00110	000	00101	011 0011	(0x407302B3)
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	

```
Assembly =   add x18, x19, x20

  Binary =   0000 0001 0100 1001 1000 1001 0011 0011
```

```
Assembly =   sub x5, x6, x7

  Binary =   0100 0000 0111 0011 0000 0010 1011 0011
```

rvcodec.js · RISC-V Instruction Encoder/Decoder

I-Type Examples

Assembly	imm$_{11:0}$	rs1	funct3	rd	op	Machine Code	
`addi s0, s1, 12` `addi x8, x9, 12`	12	9	0	8	19	0000 0000 1100 01001 000 01000 001 0011	(0x00C48413)
`addi s2, t1, -14` `addi x18,x6, -14`	-14	6	0	18	19	1111 1111 0010 00110 000 10010 001 0011	(0xFF230913)
`lw t2, -6(s3)` `lw x7, -6(x19)`	-6	19	2	7	3	1111 1111 1010 10011 010 00111 000 0011	(0xFFA9A383)
`lh s1, 27(zero)` `lh x9, 27(x0)`	27	0	1	9	3	0000 0001 1011 00000 001 01001 000 0011	(0x01B01483)
`lb s4, 0x1F(s4)` `lb x20,0x1F(x20)`	0x1F	20	0	20	3	0000 0001 1111 10100 000 10100 000 0011	(0x01FA0A03)
	12 bits	5 bits	3 bits	5 bits	7 bits	12 bits 5 bits 3 bits 5 bits 7 bits	

```
Assembly =   addi x18, x19, 20

  Binary =   0000 0001 0100 1001 1000 1001 0001 0011
```

```
Assembly =   add x18, x19, x20

  Binary =   0000 0001 0100 1001 1000 1001 0011 0011
```

← R-Type

Interpreting Machine Code

- Write in binary
- Start with **op**: tells how to parse rest
- Extract fields
- **op**, **funct3**, and **funct7** fields tell operation
- **Ex:** 0x41FE83B3 and 0xFDA58393

x31 x7

`0x41FE83B3: 0100 0001 1111 1110 1000 0011 1011 0011`

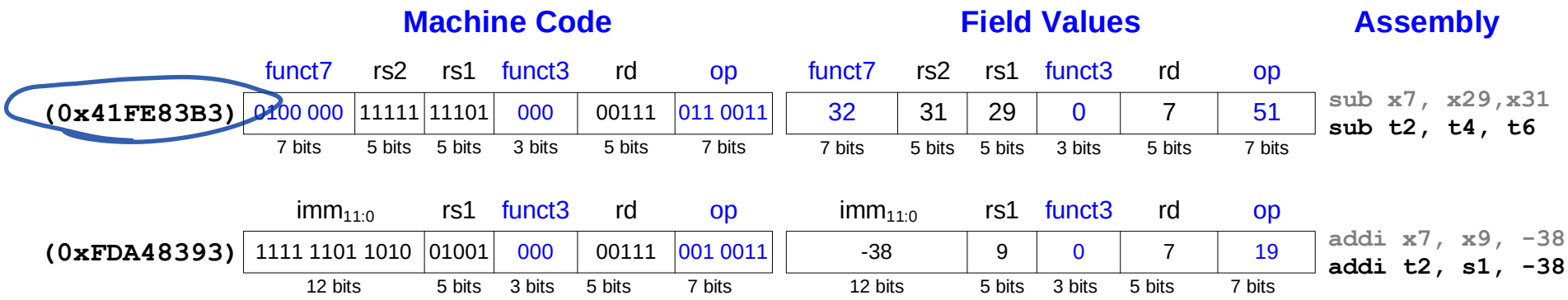
op = 51, funct3 = 0: add or sub (R-type)
funct7 = 0100000: sub

`0xFDA48393: 1111 1101 1010 0100 1000 0011 1001 0011`

op = 19, funct3 = 0: `addi` (I-type)

Interpreting Machine Code

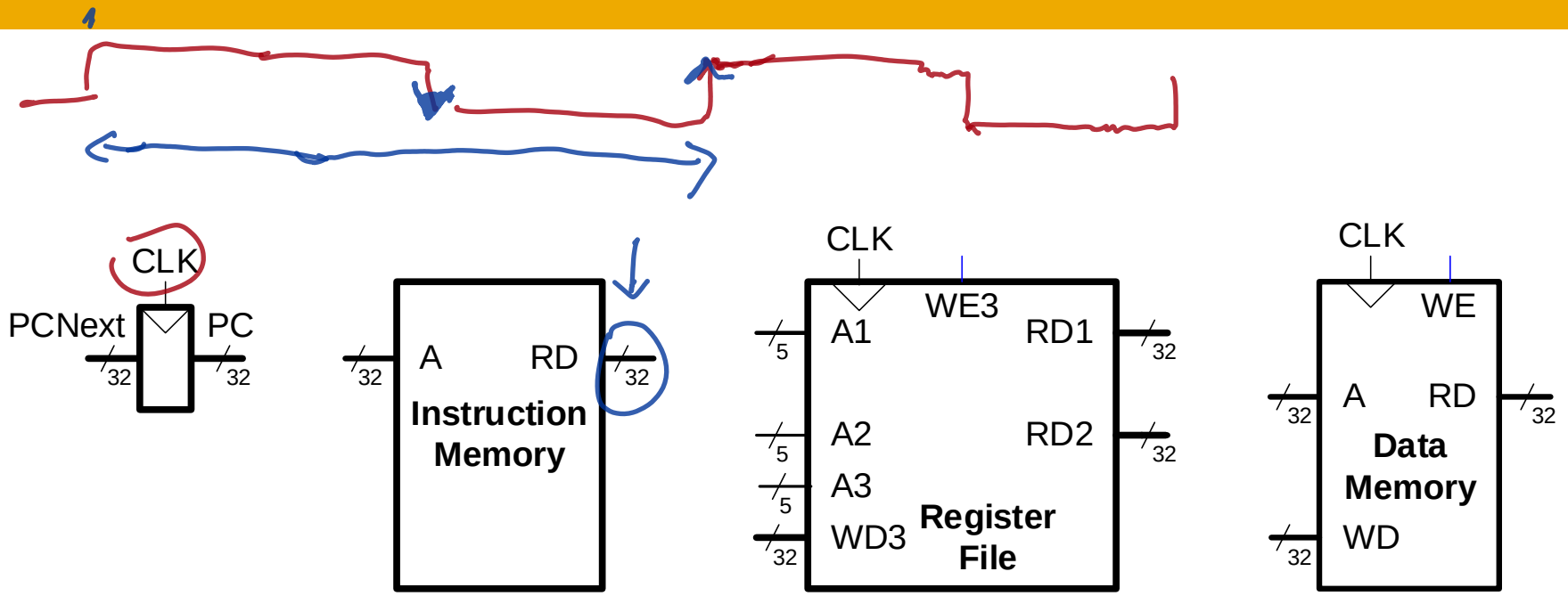
- Write in binary
- Start with **op**: tells how to parse rest
- Extract fields
- **op**, **funct3**, and **funct7** fields tell operation
- **Ex:** 0x41FE83B3 and 0xFDA58393



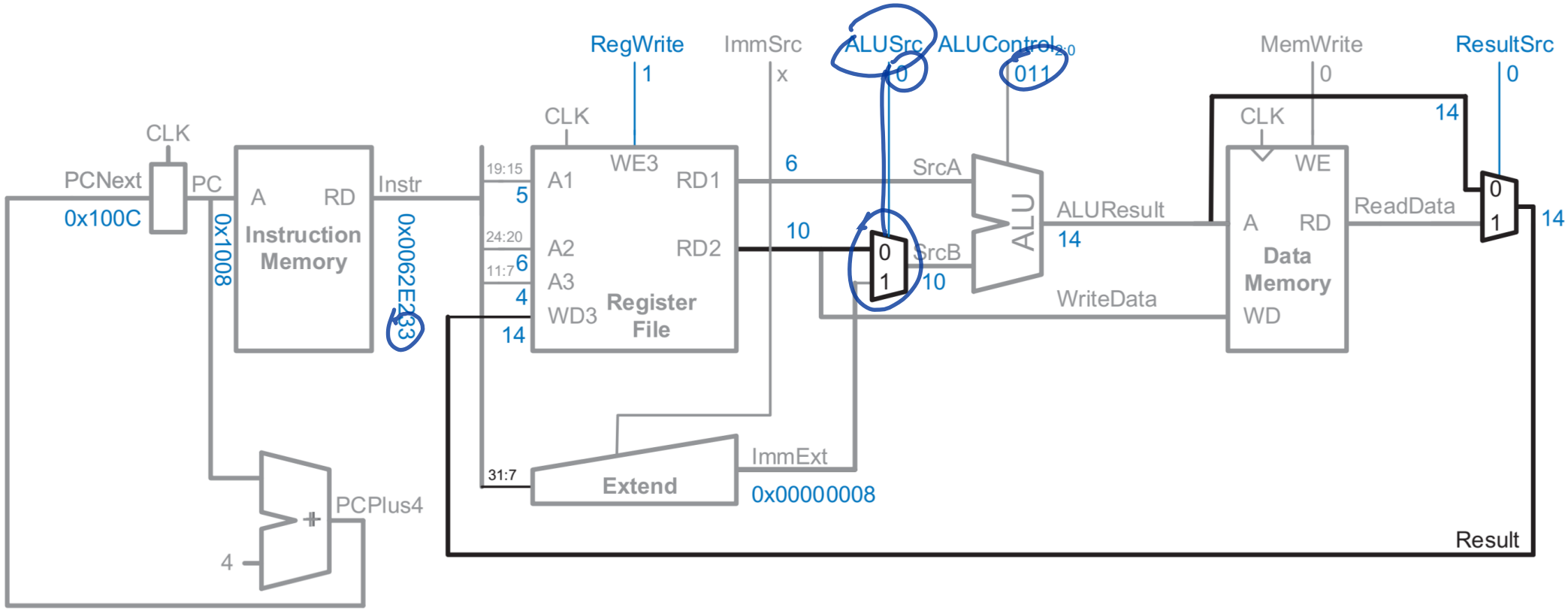
Chapter 6: Architecture

Microarchitecture

RISC-V Architectural State Elements



RV32I Single Cycle Processor (Partial)

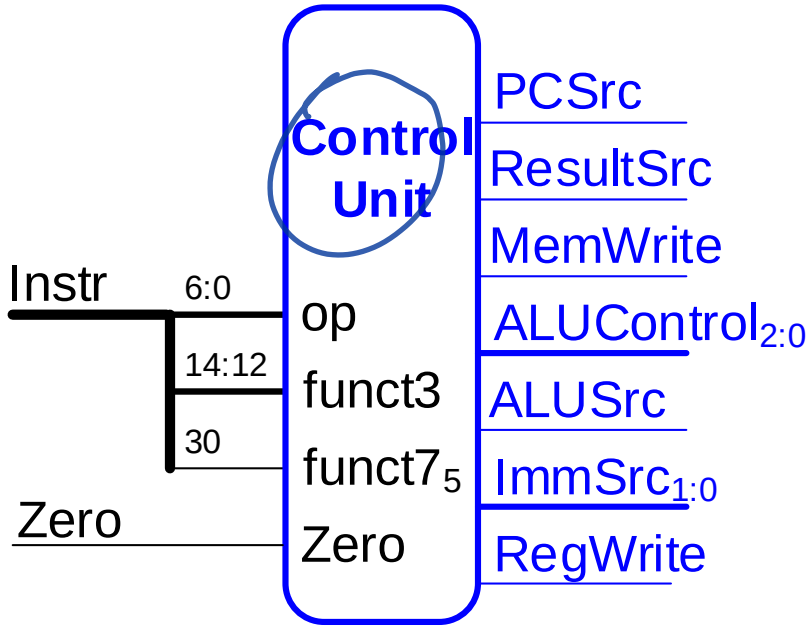


Address	Instruction		Type	Fields					Machine Language
				funct7	rs2	rs1	f3	rd	op	
0x1008	or	x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233

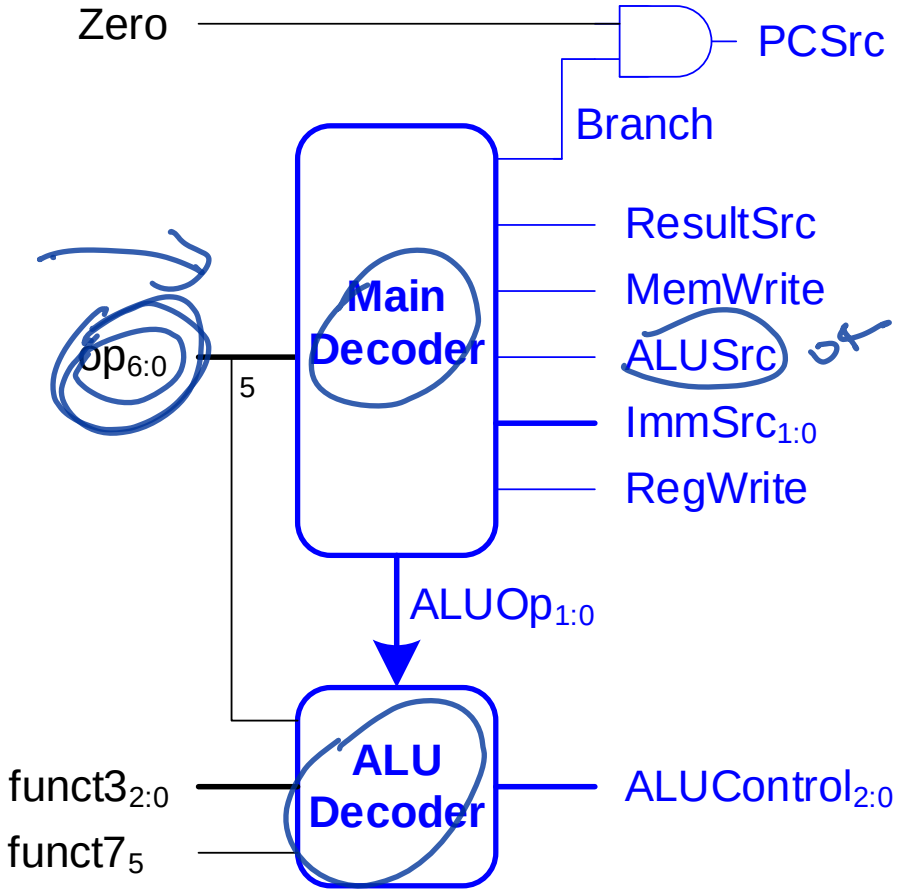
Figure 7.10 Datapath enhancements for R-type instructions

Single-Cycle Control

High-Level View



Low-Level View

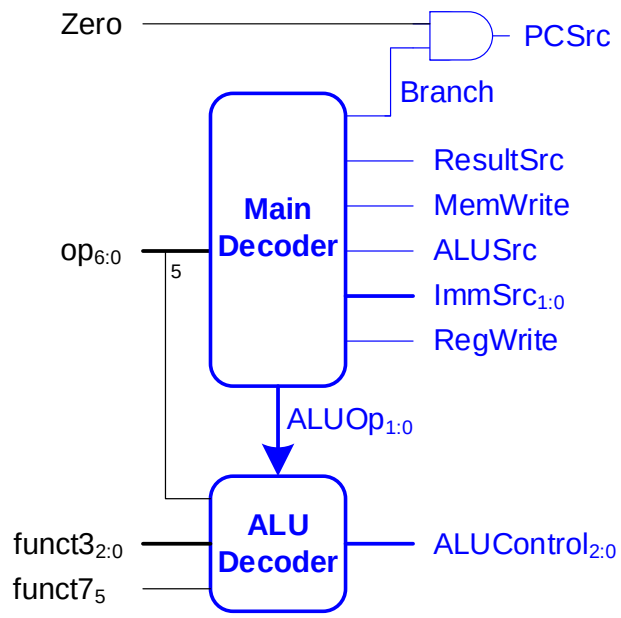


Single-Cycle Control

How does I-Type vs. R-Type influence Main decoder?

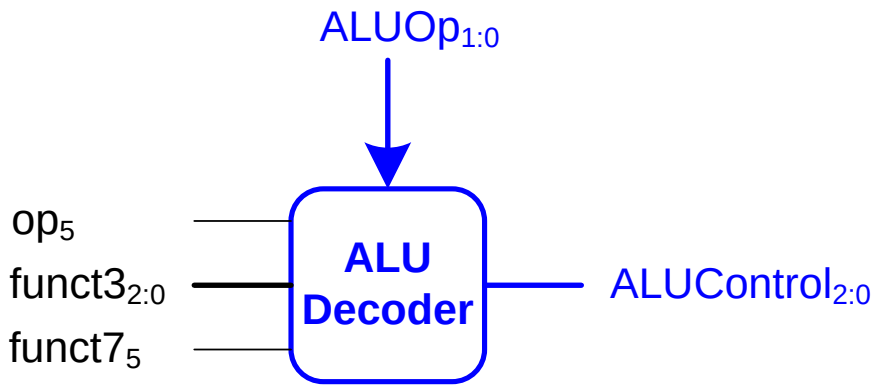
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
funct7	rs2	rs1	funct3	rd	op	R-Type
imm	imm	rs1	funct3	rd	op	I-Type

```
or x4, x5, x6    0000 0000 0110 0010 1110 0010 0011 0011
ori x4, x5, 6    0000 0000 0110 0010 1110 0010 0001 0011
```

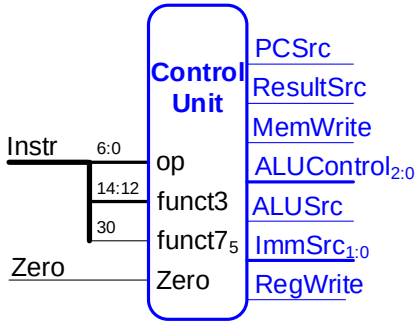


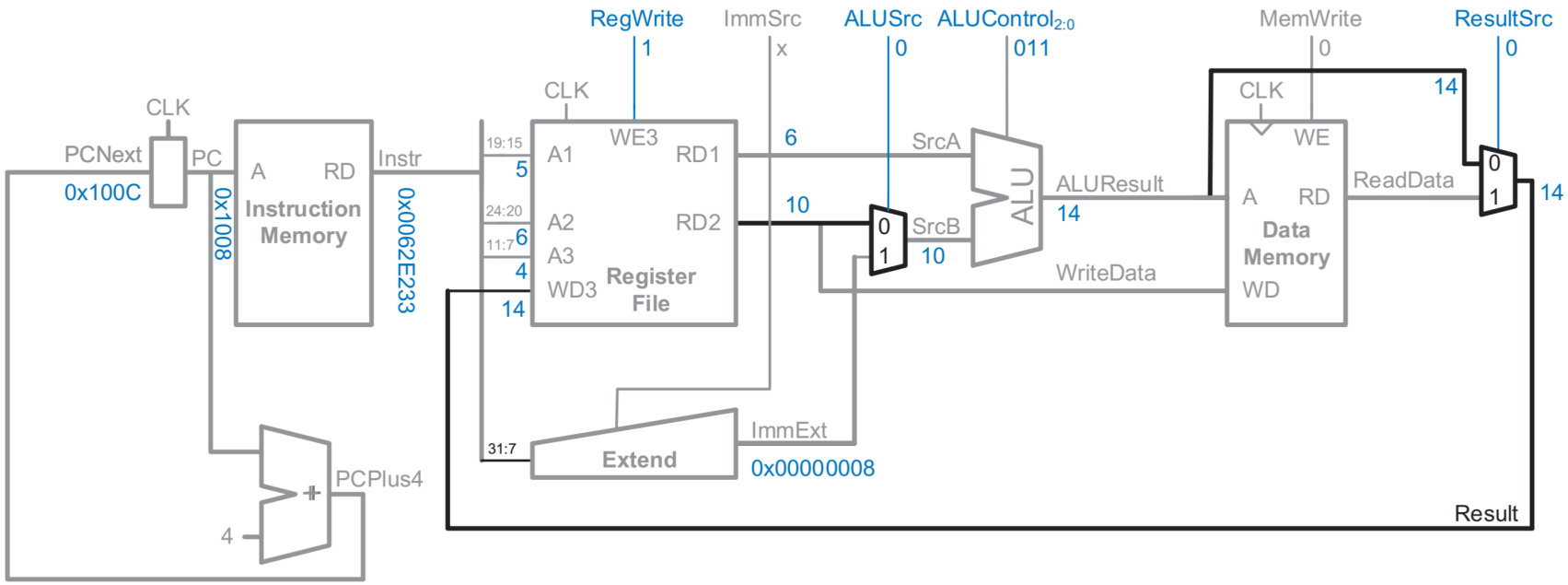
Single-Cycle Control: ALU Decoder

ALUOp	funct3	op_5, $funct7_5$	Instruction	$ALUControl_{2:0}$
00	x	x	`lw, sw`	000 (add)
01	x	x	`beq`	001 (subtract)
10	000	00, 01, 10	`add`	000 (add)
	000	11	`sub`	001 (subtract)
	010	x	`slt`	101 (set less than)
	110	x	`or`	011 (or)
	111	x	`and`	010 (and)



RV32I Single Cycle Processor (Partial)





Address	Instruction		Type	Fields					Machine Language	
				funct7	rs2	rs1	f3	rd	op	
0x1008	or	x4, x5, x6	R	0000000	00110	00101	110	00100	0110011	0062E233

Figure 7.10 Datapath enhancements for R-type instructions

Practice

Machine decoding

Write the assembly code from this machine code: `0x00b54533`

Check your work at rvcodec.js

Xor x10 x10 x11

7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
funct7	rs2	rs1	funct3	rd	op	**R-Type**
$imm_{11:0}$		rs1	funct3	rd	op	**I-Type**

Constant initialization

Write the machine code (in hexadecimal) for placing -58 into register `t7`

7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
funct7	rs2	rs1	funct3	rd	op	**R-Type**
$imm_{11:0}$		rs1	funct3	rd	op	**I-Type**

Instruction	op	funct3	Funct7	Type
`addi`	0010011 (19)	000 (0)	-	I-Type