

COEN-2710 Microprocessors - Lecture 3

Processor Part 1: Datapath and Control (Ch.4)

Cris Ababei
Marquette University
Department of Electrical and Computer Engineering

1

1

Goals of this Chapter

- ❖ Design a **datapath** and **control** that implement the RISC-V instruction set architecture (ISA).
- ❖ By the end of this chapter, you should:
 - ◆ Be able to design a datapath for an instruction set
 - ◆ Be able to design a control logic for the datapath
 - ◆ Understand the importance of the clocking methodology on the processor design
- ❖ We will examine two RISC-V implementations
 - ◆ A simplified version
 - ◆ A more realistic pipelined version

2

2

What are datapath and control?

❖ Datapath

- ◆ The path the “data” follow and undergo computations.
- ◆ Realized by the hardware components connected in a way to perform operations on data such that machine instructions are implemented.

❖ Control

- ◆ Control is the sequential logic that reconfigures the Datapath to allow the “data” to flow properly through the hardware components.
- ◆ Responsible with the generation of all control signals to “orchestrate” the correct flow of data through Datapath!
- ◆ Can be implemented as finite state-machine(s), FSMs.
- ◆ Can also be implemented as a computer inside of a computer (microcode).

3

3

Design Process

- 1. Select a subset of the instruction set to implement. Simple subset, shows most aspects**
 - ◆ **Memory reference:** ld, sd
 - ◆ **Arithmetic/logical:** add, sub, and, or
 - ◆ **Control transfer:** beq
- 2. Order the steps within instruction cycle (performed during instruction execution)**
- 3. Select the hardware components.**
- 4. Connect the hardware components.**
- 5. Design the control to make the components work together properly.**

4

4

Order the steps: **FIDE**

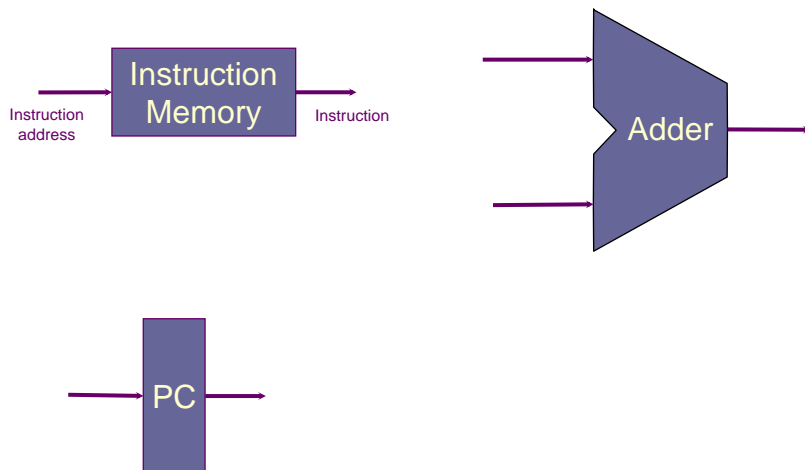
❖ **FIDE** – the sequence of activities that happens during instruction execution

1. **Fetch** (the instruction)
2. **“Increment”** (the Program Counter)
3. **Decode** (the Instruction Register)
4. **Execute** (using datapath hardware)

5

5

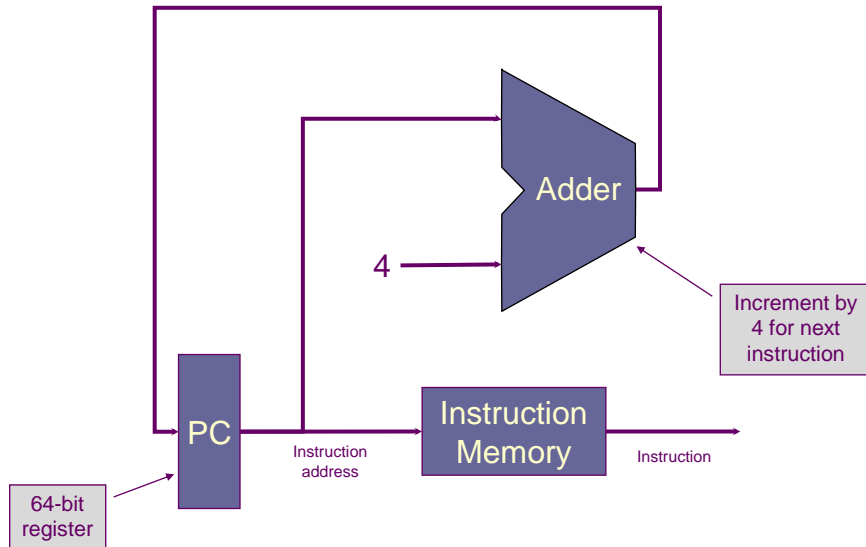
Fetch and Increment Hardware



6

6

Fetch and Increment Connections



7

Decode and Execute

❖ Decode

- ◆ Takes the Instruction Register (IR) and computes the bits needed to control the datapath (R/W flags, enables, mux selects, etc.)
- ◆ We will work more on the control later

❖ Execute

- ◆ Take the inputs specified by the instruction, and complete the required operation

8

8

Instruction Execution

- ❖ PC → instruction memory, fetch instruction
- ❖ Register numbers → register file, read registers
- ❖ Depending on instruction class
 - ◆ Use ALU to calculate
 - Arithmetic result
 - Memory address for load/store
 - Branch comparison
 - ◆ Access data memory for load/store
 - ◆ PC ← target address or PC + 4

9

9

R-format Example: “ADD” Instruction

funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

General form: add rd, rs1, rs2

Example: add x9, x20, x21

0	21	20	0	9	51
---	----	----	---	---	----

0000000	10101	10100	000	01001	0110011
---------	-------	-------	-----	-------	---------

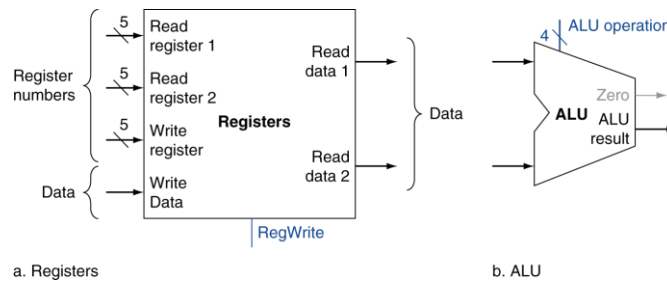
- ❖ For the execute step of FIDE, what hardware do we need?

10

10

R-Format Instructions - Hardware

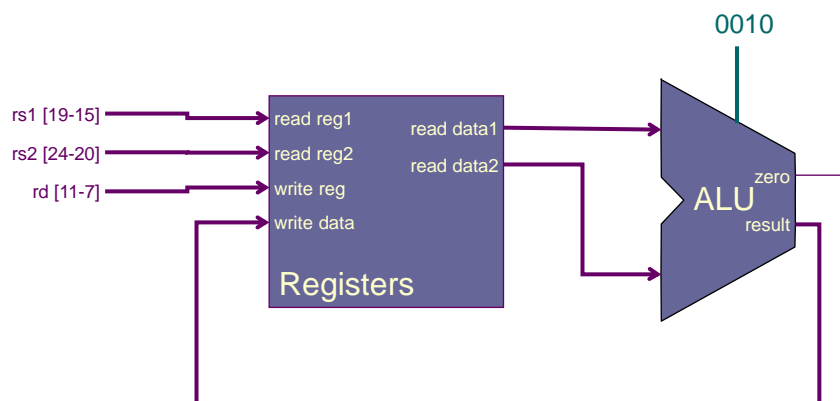
- ❖ Read two register operands
- ❖ Perform arithmetic/logical operation
- ❖ Write register result



11

11

With Hardware Connections



12

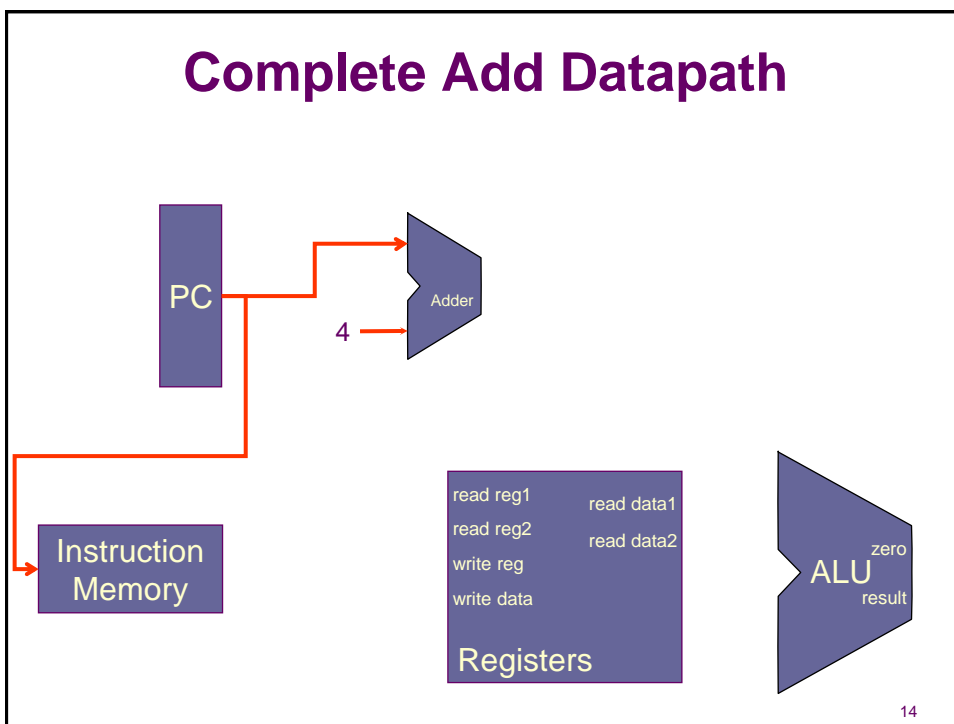
12

Complete “Add” Datapath



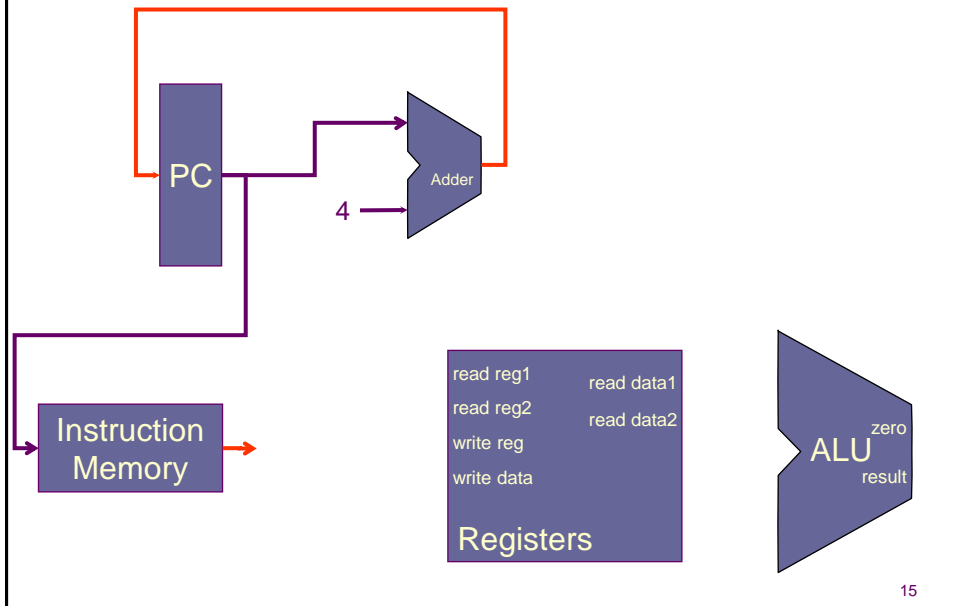
13

Complete Add Datapath



14

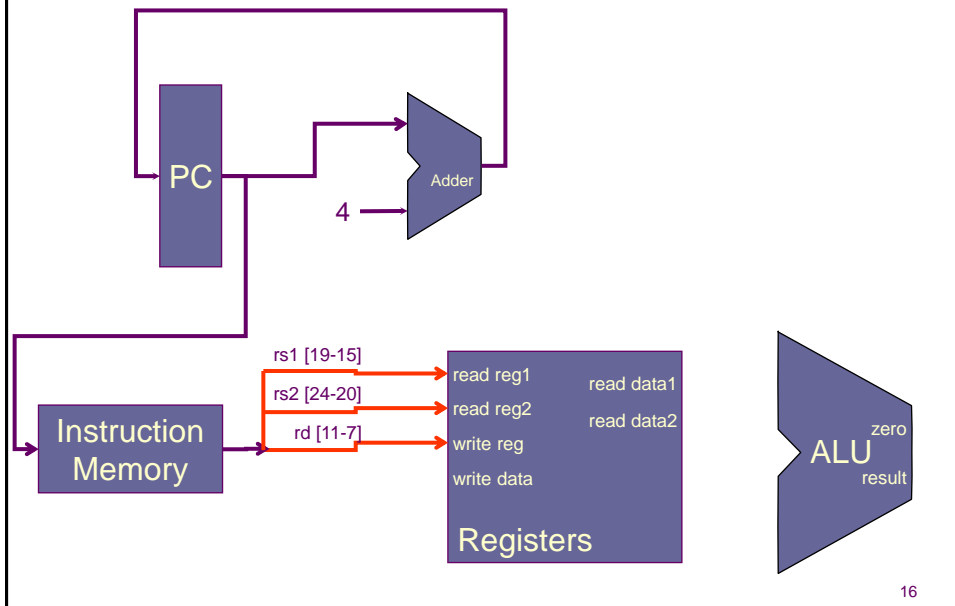
Complete Add Datapath



15

15

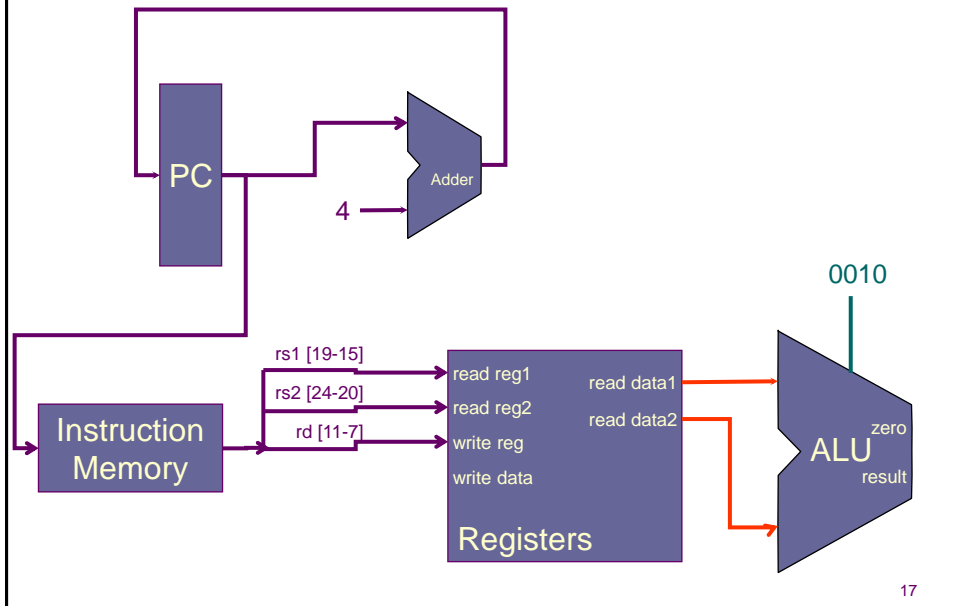
Complete Add Datapath



16

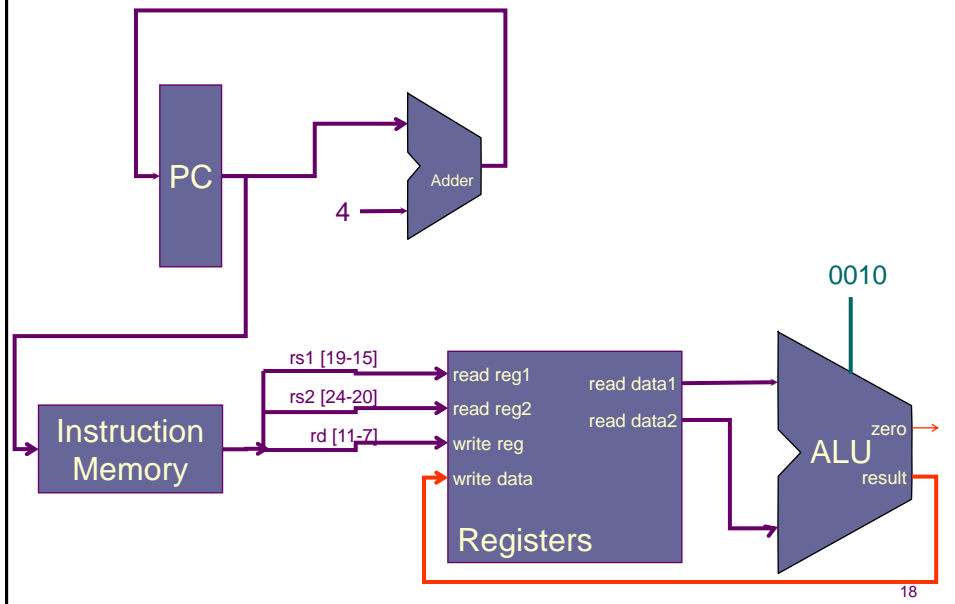
16

Complete Add Datapath



17

Complete Add Datapath



18

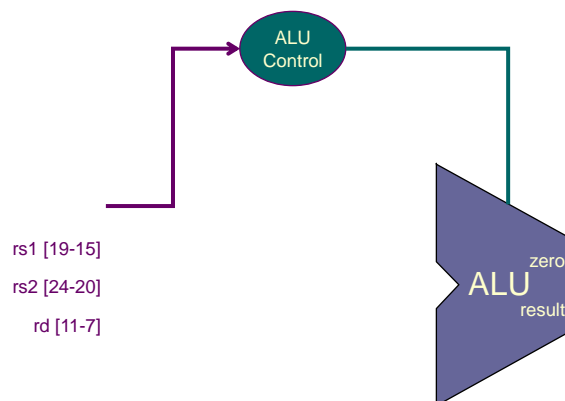
Control of R-format Instructions

- ❖ **Simplicity favors regularity!**
 - ◆ and, or, add, subtract, set-on-less-than all use the same **datapath**
- ❖ **Need to decode the instructions to **control** the ALU.**
 - ◆ Input: **Function codes** for each (recall from Chapter 2)
 - ◆ Output: **ALU control lines** (will look at later)

19

19

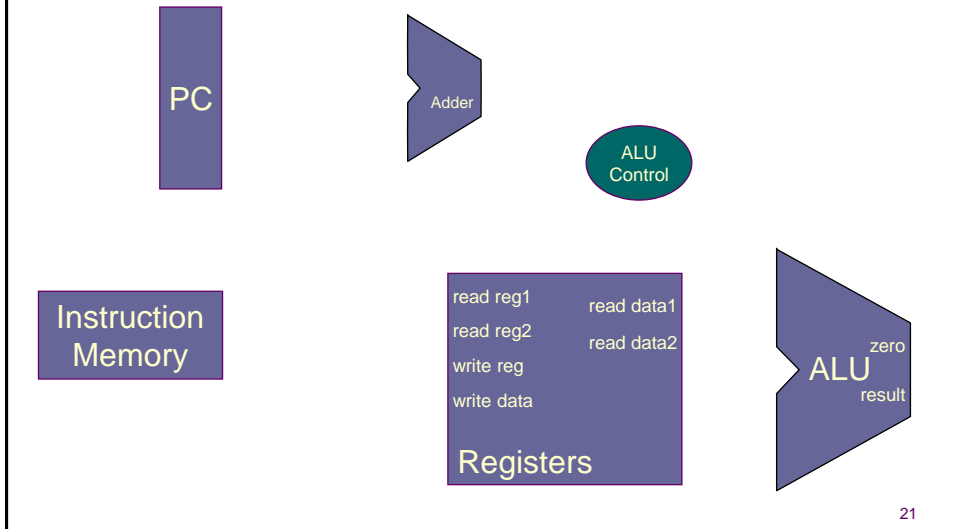
ALU Decoding for R-format



20

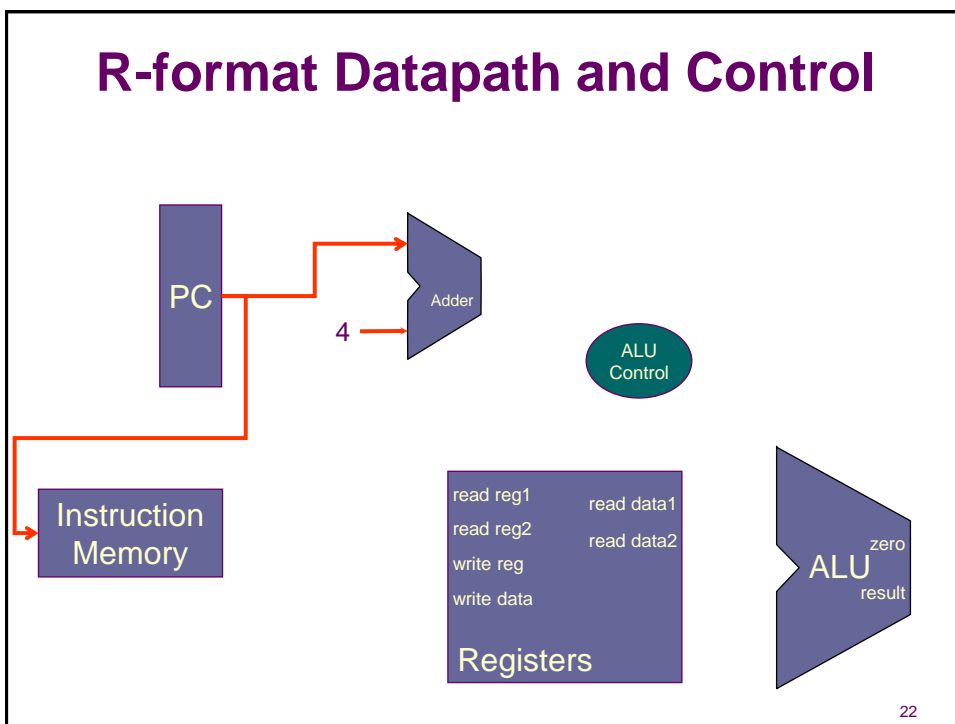
20

R-format Datapath and Control



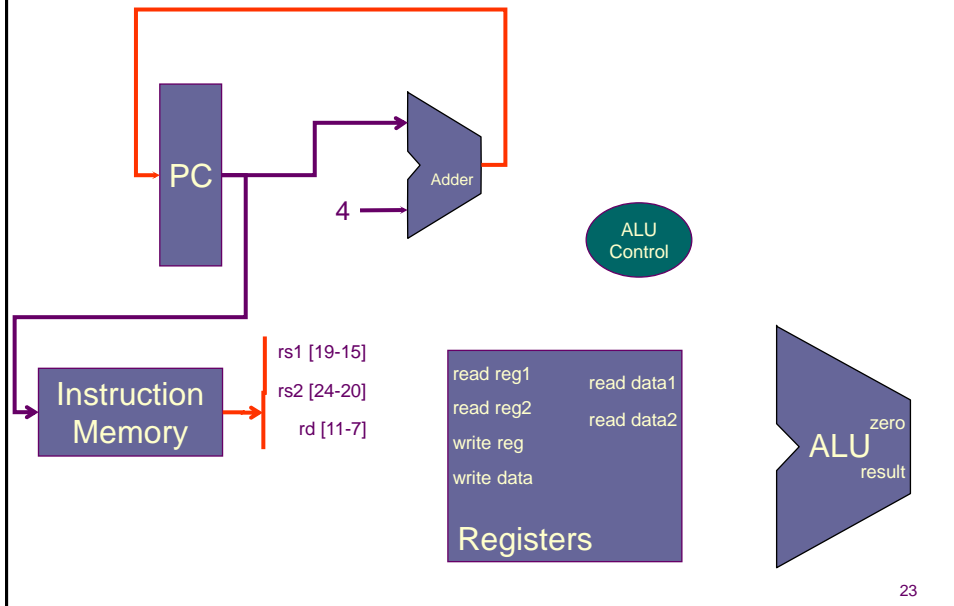
21

R-format Datapath and Control



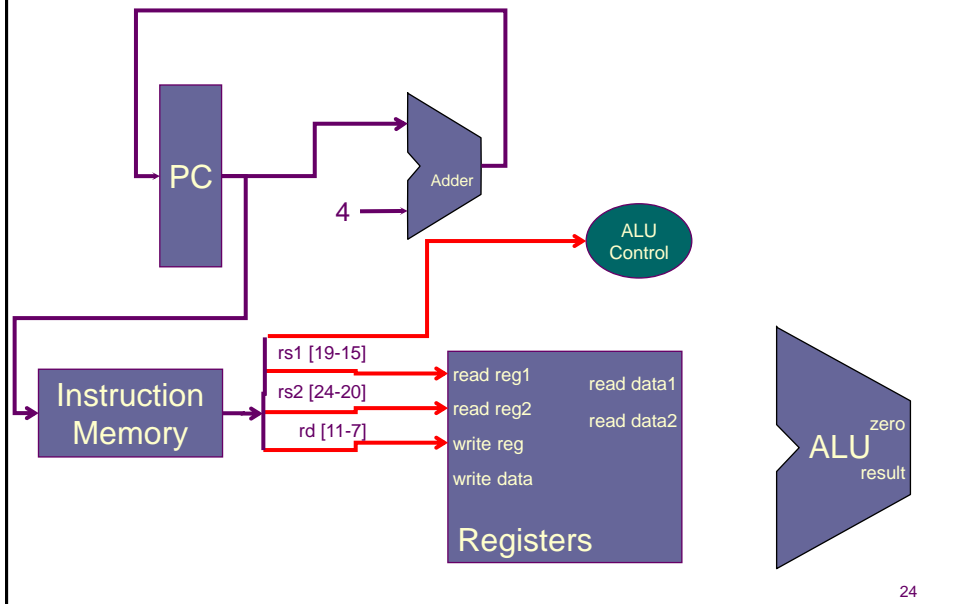
22

R-format Datapath and Control



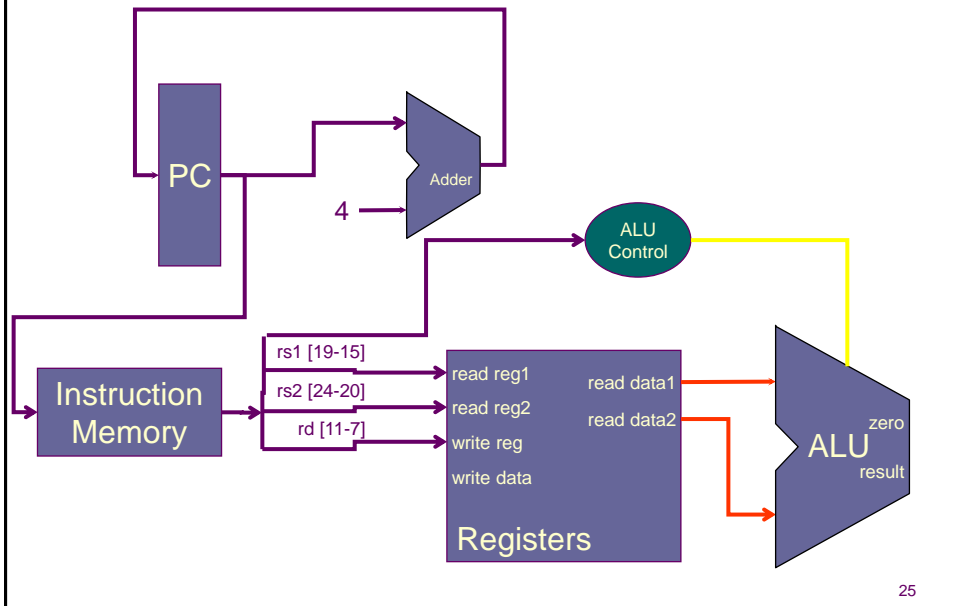
23

R-format Datapath and Control



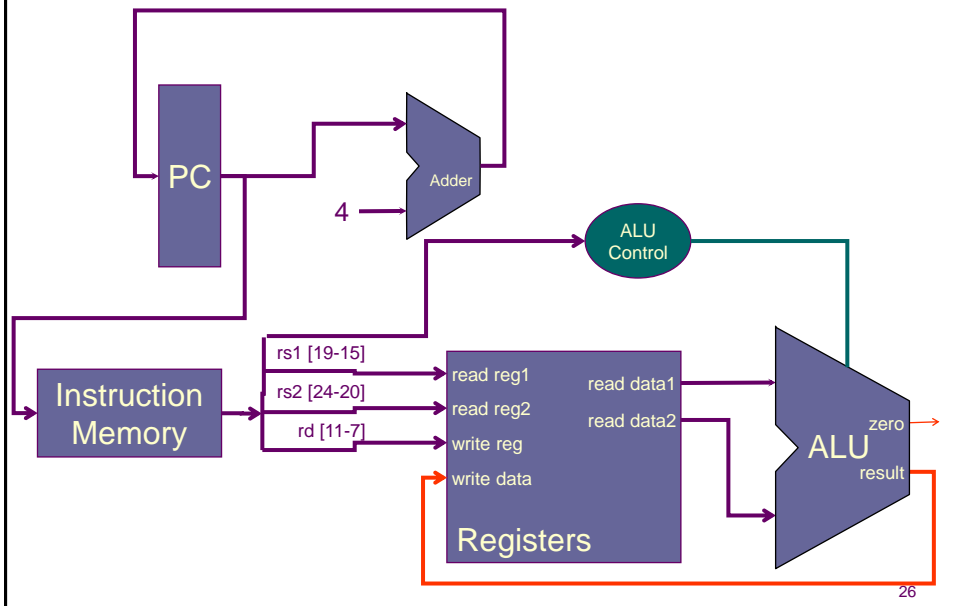
24

R-format Datapath and Control



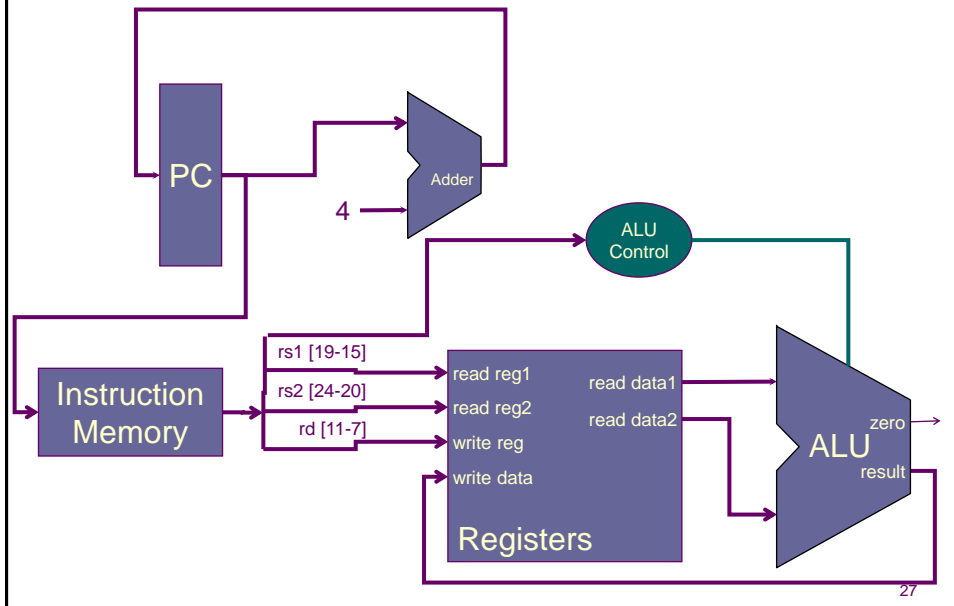
25

R-format Datapath and Control



26

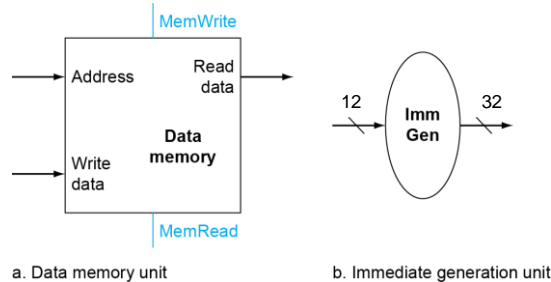
R-format Datapath and Control



27

Load/Store Instructions - Hardware

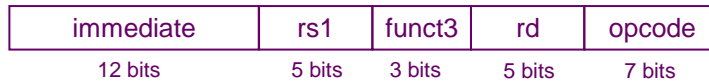
- ❖ Read register operands
- ❖ Calculate address using 12-bit offset
 - ◆ Use ALU, but sign-extend offset
- ❖ Load: Read memory and update register
- ❖ Store: Write register value to memory



28

28

I-format Instructions



❖ Immediate arithmetic and load instructions

- ◆ rs1: source or base address register number
- ◆ immediate: constant operand, or offset added to base address
 - 2s-complement, sign extended

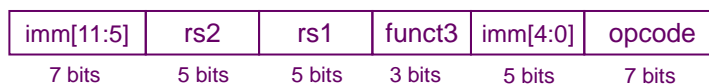
General form: `addi rd, rs1, imm`

General form: `lw rd, imm(rs1)`

29

29

S-format Instructions



❖ Different immediate format for store instructions

- ◆ rs1: base address register number
- ◆ rs2: source operand register number
- ◆ immediate: offset added to base address
 - Split so that rs1 and rs2 fields always in the same place

General form: `sw rs2, imm(rs1)`

30

30

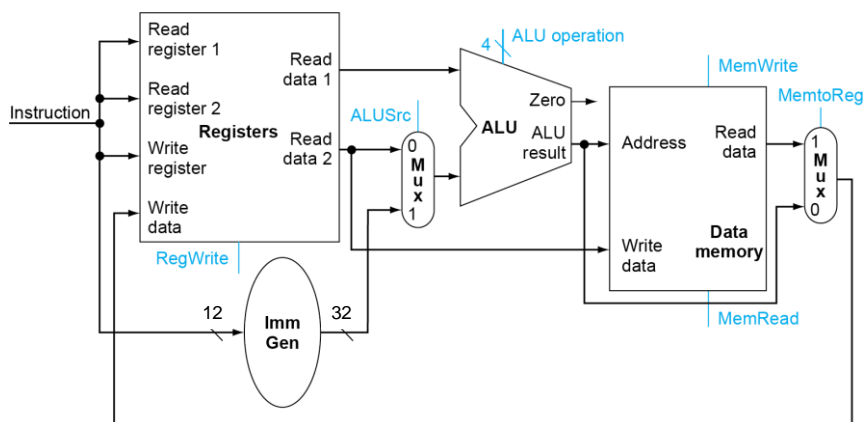
Composing the Elements

- ❖ Datapath does an instruction in one clock cycle
 - ◆ Each datapath element can only do one function at a time
 - ◆ Hence, we need separate instruction and data memories
- ❖ Use multiplexers where alternate data sources are used for different instructions

31

31

R-Type/Load/Store Datapath



32

32

Branch Instructions

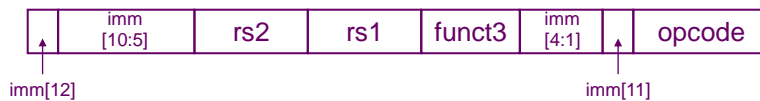
- ❖ Read register operands
- ❖ Compare operands
 - ◆ Use ALU, subtract and check Zero output
- ❖ Calculate target address
 - ◆ Sign-extend displacement
 - ◆ Shift left 1 place (halfword displacement)
 - ◆ Add to PC value

33

33

SB-format - Branch Addressing

❖ SB format:



■ PC-relative addressing

- Target address = PC + immediate $\times 2$

❖ Branch to a labeled instruction if condition is true

- ◆ Otherwise, continue sequentially

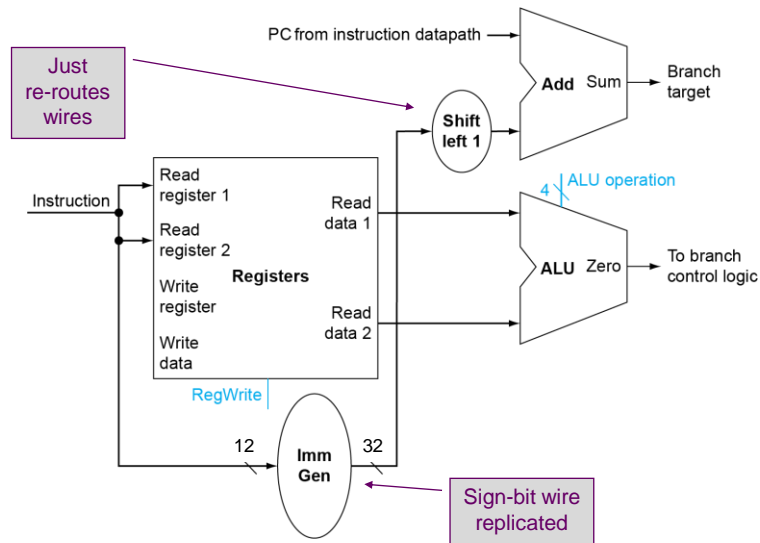
❖ beq rs1, rs2, L1

- ◆ if (rs1 == rs2) branch to instruction labeled L1

34

34

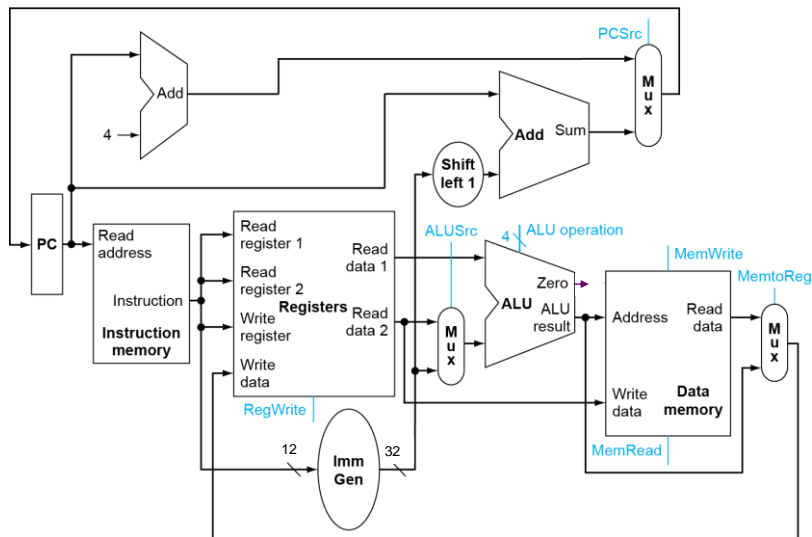
Branch Instructions



35

35

Full Datapath (Without Control Shown)



36

36

Overall Control

❖ Split into two Controllers (for now...)

❖ Divide and Conquer

1. “ALU Control” Unit

- Uses 2-bit ALUOp generated by Main Control unit
- Uses also Funct7 and Funct3 fields from Instruction
- Generates control signals ALUOperation (4 bits) that control directly the function executed by the ALU

2. “Main Control” Unit

- Control signals derived from instruction (Opcode)
- Generates a 2-bit ALUOp used by ALU Control

37

37

1) “ALU Control” Unit

❖ Generates “ALUOperation” control signals (4 bits)

◆ Based on inputs: ALUOp, Funct7, and Funct3

❖ ALUOp (2 Bits) derived from opcode by “Main Control” unit

❖ Can be implemented by simple combinational logic

◆ By logic synthesis from Truth Table on next slide

ALUOp Values

Instr.	ALUOp	Operation
ld	00	load register
sd	00	store register
beq	01	branch on equal
R-type	10	add
		subtract
		AND
		OR

38

38

ALU Control – Truth Table

❖ **Input signals:**

- ◆ **ALUOp (2 bits), Funct7 (7 bits, Instruction[31-25], Funct3 (3 bits, Instruction [14-12])**

❖ **Output signals:**

- ◆ **ALUOperation control signals (4 bits)**

Truth Table

ALUOp		Funct7 field							Funct3 field			ALUOperation
ALUOpI	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

❖ Table is from Figure 4.13 in Textbook

39

39

2) “Main Control” Unit

❖ **Generates control signals for: Register file, data memory, multiplexers, AND gate (branch related), ALUOp (2 bits), etc.**

❖ **Control signals derived from instruction Opcode**

Name (Bit position)	Fields					
	31:25	24:20	19:15	14:12	11:7	6:0
(a) R-type	funct7	rs2	rs1	funct3	rd	opcode
(b) I-type	immediate[11:0]		rs1	funct3	rd	opcode
(c) S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode
(d) SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode

40

40

Main Control – Truth Table

❖ Input signals:

- ◆ Opcode (7 bits, Instruction [6-0])

❖ Output signals:

- ◆ ALUOp (2 bits, used by ALU Control), ALUSrc, MemtoReg, RegWrite, etc.

Truth Table

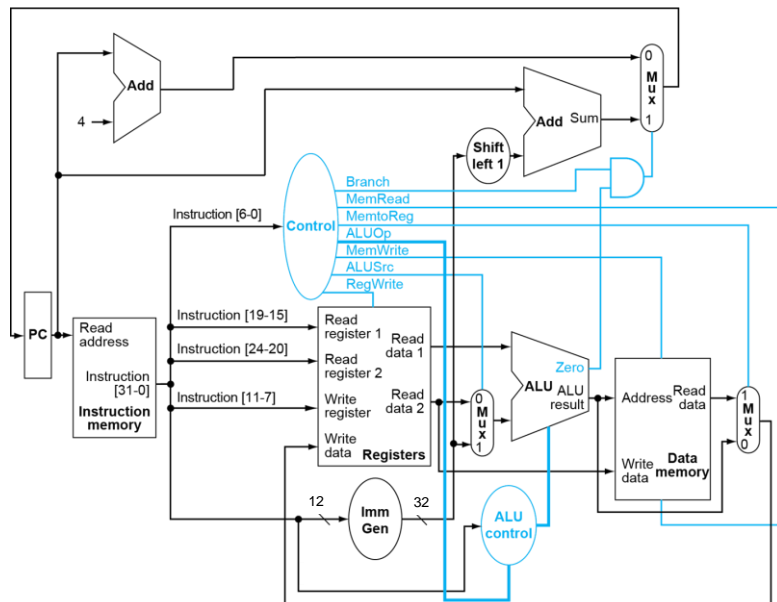
ALUOp Values			Input or output	Signal name	R-format	lw	sw	beq
Instr.	ALUOp	Operation	Inputs	I[6]	0	0	0	1
ld	00	load register		I[5]	1	0	1	1
sd	00	store register		I[4]	1	0	0	0
beq	01	branch on equal		I[3]	0	0	0	0
R-type	10	add		I[2]	0	0	0	0
		subtract		I[1]	1	1	1	1
		AND		I[0]	1	1	1	1
		OR	ALUSrc	0	1	1	0	
			MemtoReg	0	1	X	X	
		RegWrite	1	1	0	0		
		MemRead	0	1	0	0		
		MemWrite	0	0	1	0		
		Branch	0	0	0	1		
		ALUOp1	1	0	0	0	0	
		ALUOp0	0	0	0	0	1	

❖ Table is from Figure 4.26 in Textbook

41

41

Datapath With Control



42

42

R-Type Instruction

The diagram illustrates the execution of an R-Type instruction in a processor. The components and their interactions are as follows:

- PC (Program Counter):** Provides the address for the instruction memory.
- Instruction Memory:** Receives the PC address and outputs the instruction. The instruction is divided into fields:
 - Instruction [31-0]:** The full instruction.
 - Instruction [19-15]:** Read register 1.
 - Instruction [24-20]:** Read register 2.
 - Instruction [11-7]:** Write register.
 - Instruction [6-0]:** Control field.
- Control:** Receives the control field from the instruction and outputs control signals to the ALU, Registers, and Memory. The control signals are:
 - Branch
 - MemRead
 - MemtoReg
 - ALUOp
 - MemtoWrite
 - ALUSrc
 - RegWrite
- Registers:** Receives the write register and write data (from the ALU result) and outputs the read data 1 and read data 2. It also receives the read register 1 and read register 2 addresses.
- ALU (Arithmetic Logic Unit):** Receives the ALUOp control signal and the ALUSrc control signal. It performs the operation on the read data 1 and read data 2. The ALU result is output to the Registers and the ALU control.
- ALU control:** Receives the ALU control signal and outputs the ALU control signal to the ALU.
- Imm Gen (Immediate Generator):** Receives the 12-bit immediate field from the instruction and outputs a 32-bit immediate value to the ALU.
- Shift left 1:** A shift register that shifts the ALU result left by one bit.
- Add:** An adder that adds the PC value and the 4-bit branch offset from the instruction to calculate the next PC.
- Mux (Multiplexer):** Receives the ALU result and the shifted ALU result. It selects between them based on the Branch control signal. The output of the Mux is the ALU result, which is then used to calculate the next PC.
- PC:** Receives the next PC value from the Mux and outputs it to the Instruction Memory.

The diagram shows the flow of data and control signals between these components, highlighting the execution of the R-Type instruction.

Load Instruction

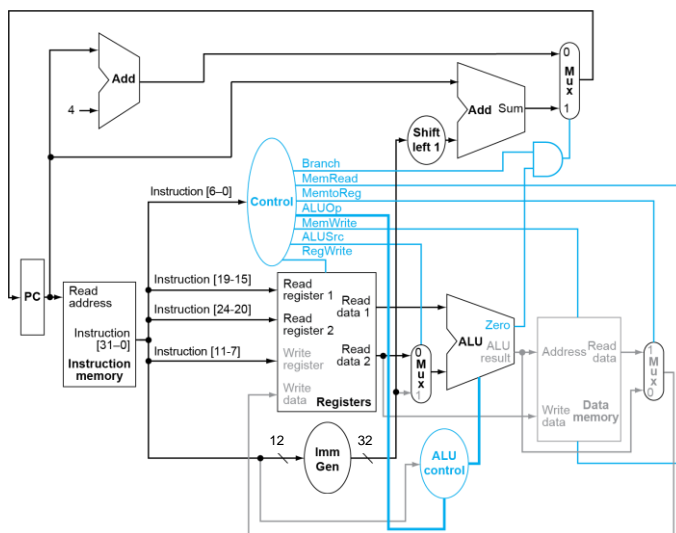
The diagram illustrates the internal components and data paths of a processor during a Load Instruction. Key components include:

- PC (Program Counter):** Provides the address for Instruction Memory.
- Instruction Memory:** Provides instruction fields: Instruction [31-0], Instruction [24-20] (Read register 2), Instruction [19-15] (Read register 1), and Instruction [11-7] (Write register).
- Control:** Receives Instruction [6-0] and outputs control signals: Branch, MemRead, MemtoReg, ALUOp, MemWrite, ALUSrc, and RegWrite.
- Registers:** Contains Read register 1, Read register 2, and Write register. It also has Read data 1, Read data 2, and Write data outputs.
- Imm Gen (Immediate Generator):** Takes a 12-bit immediate from the instruction and produces a 32-bit value.
- ALU (Arithmetic Logic Unit):** Takes two 32-bit operands (Read data 1 and Imm Gen output) and performs operations based on ALUOp. It outputs the ALU result and a Zero flag.
- ALU control:** Receives ALUOp and provides control to the ALU.
- Data Memory:** Takes an Address (Read data 2) and provides Data (Read data) and Write data outputs.
- Multiplexers (MUX):** Select between different data sources for the ALU operands, ALU control, and the final register write data.
- Shift left 1:** A barrel shifter that shifts the ALU result left by one bit.
- Add:** An adder that adds the PC value to the ALU result (shifted left by one) to calculate the next PC value.

Blue lines highlight the data paths for a Load Instruction, showing the selection of register 1 for reading, the ALU result being zero, and the data being read from memory and placed into register 2.

Page 22

BEQ Instruction

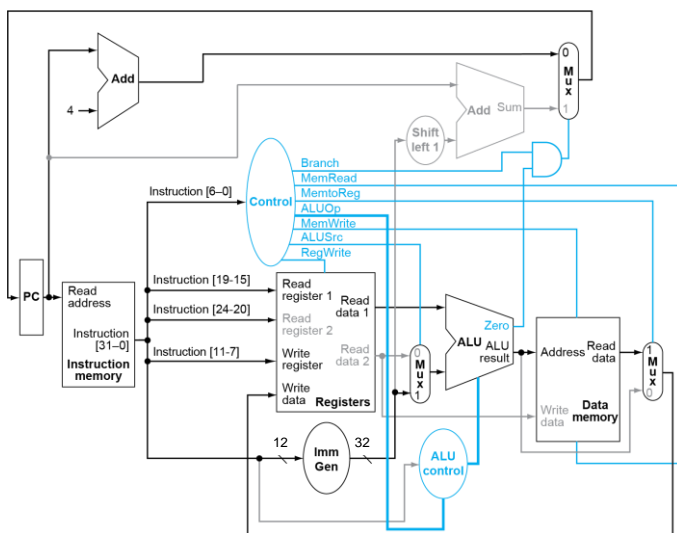


45

45

subgtz rd, rs1, rs2

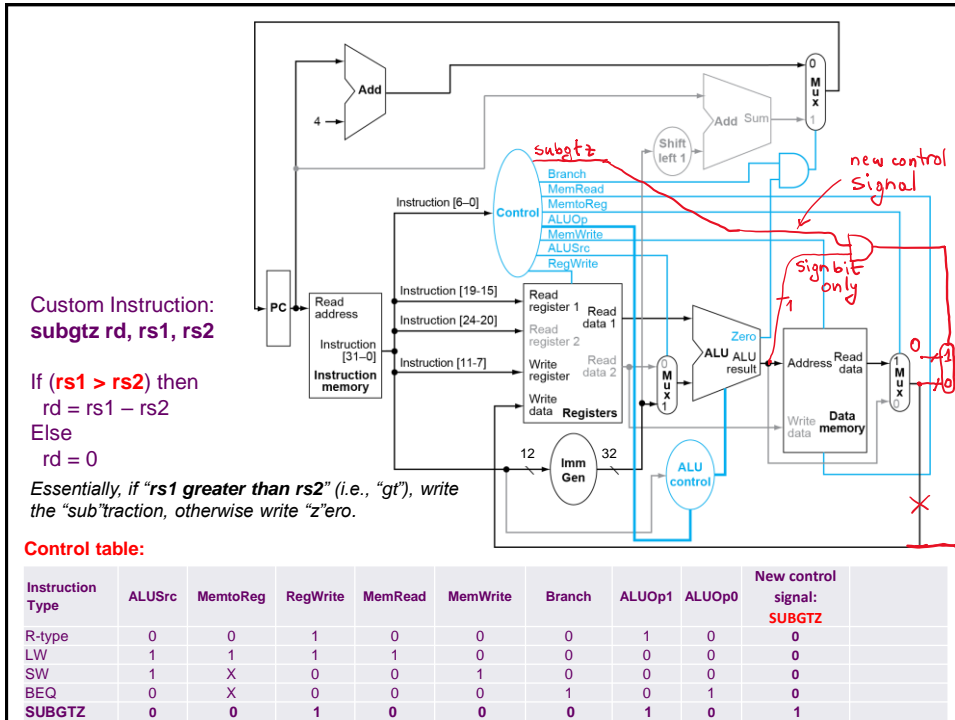
If (rs1 < rs2) then
rd = rs1 - rs2
Else
rd = 0



Control table:

Instruction Type	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0	New control signal:
R-type	0	0	1	0	0	0	1	0	
LW	1	1	1	1	0	0	0	0	
SW	1	X	0	0	1	0	0	0	
BEQ	0	X	0	0	0	1	0	1	
SUBGTZ									SUBGTZ

46

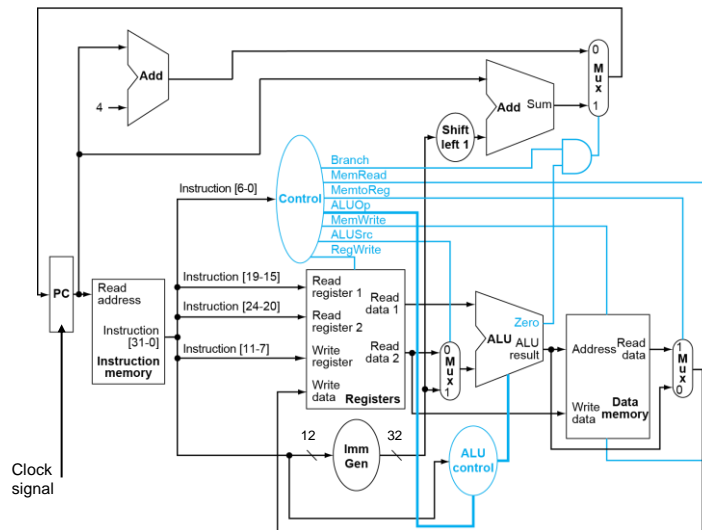


47

Practice – Performance Analysis

❖ Calculate cycle time assuming:

- ◆ Memory (2ns), ALU and adders (2ns), Register file access (1ns), MUX (0ns)



48

48

Limitations of single-cycle operations

- ❖ Each instruction uses the ENTIRE datapath, until it finishes.
 - ◆ Clock cycle time based on slowest instruction
 - ◆ What if we add more stuff, like floating-point?
 - Worst-case time delay drastically exceeds average
 - Need really long cycle time to accommodate
- ❖ Goal: Use as much of the hardware as much of the time as possible
 - ◆ PIPELINING: Break the datapath into smaller chunks, and let new instructions start while others are finishing

49