Digital Design with Verilog

RISC-V : Building Single Cycle RISC-V Processor

Lecture 29 : RISC-V Part 4

Disclaimer

I do not own all the slides which I am going to present. The content is mostly from

- GIAN course titled "**Next-Generation Semiconductors: RISC-V, AI, TL-Verilog"** by Steeve Hoover. https://github.com/silicon-vlsi/gian-course-2024-IITBBS
- Introduction to Computer Architecture by Hasan Baig, https://www.hasanbaig.com
  - Most of the slides are copied/directly from his course content, please visit his site for reference.
- Digital Design and Computer Architecture, RISC-V Edition by Sarah L. Harris and David Money Harris.
- Other online publicly available sources. I tried my best to acknowledge the source wherever possible. I too might have missed some sources too ☺

RISC-V Instruction Summary

Simple subset that shows most aspects

- Arithmetic/logical: add, sub, and, or
- Memory reference: lw, sw
- Branch: beq

Type	Instruction	Opcode	Funct3	Funct7
R-Type	add	011 0011	000	000 0000
R-Type	sub	011 0011	000	010 0000
R-Type	and	011 0011	111	000 0000
R-Type	or	011 0011	110	000 0000
I-Type	lw	000 0011		
S-Type	sw	010 0011		
SB-Type	beq	110 0011		

RISC-V Instruction Summary

Fields are at the same location in all encoding formats

opcode, funct3, rs1, rs2, rd

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

How are the instructions executed?

Instruction Execution

- What are the steps to execute instructions?

```
add     rd, rs1, rs2        # sub/and/or
lw      rd, offset(rs1)
sw      rs2, offset(rs1)
beq     rs1, rs2, offset
```

- Start from PC, which is updated at the beginning of a cycle
  - How does the processor get the instruction?
  - How does the processor get operands?
  - How does the processor generate result?
  - How does the processor save the result?

Instruction Execution

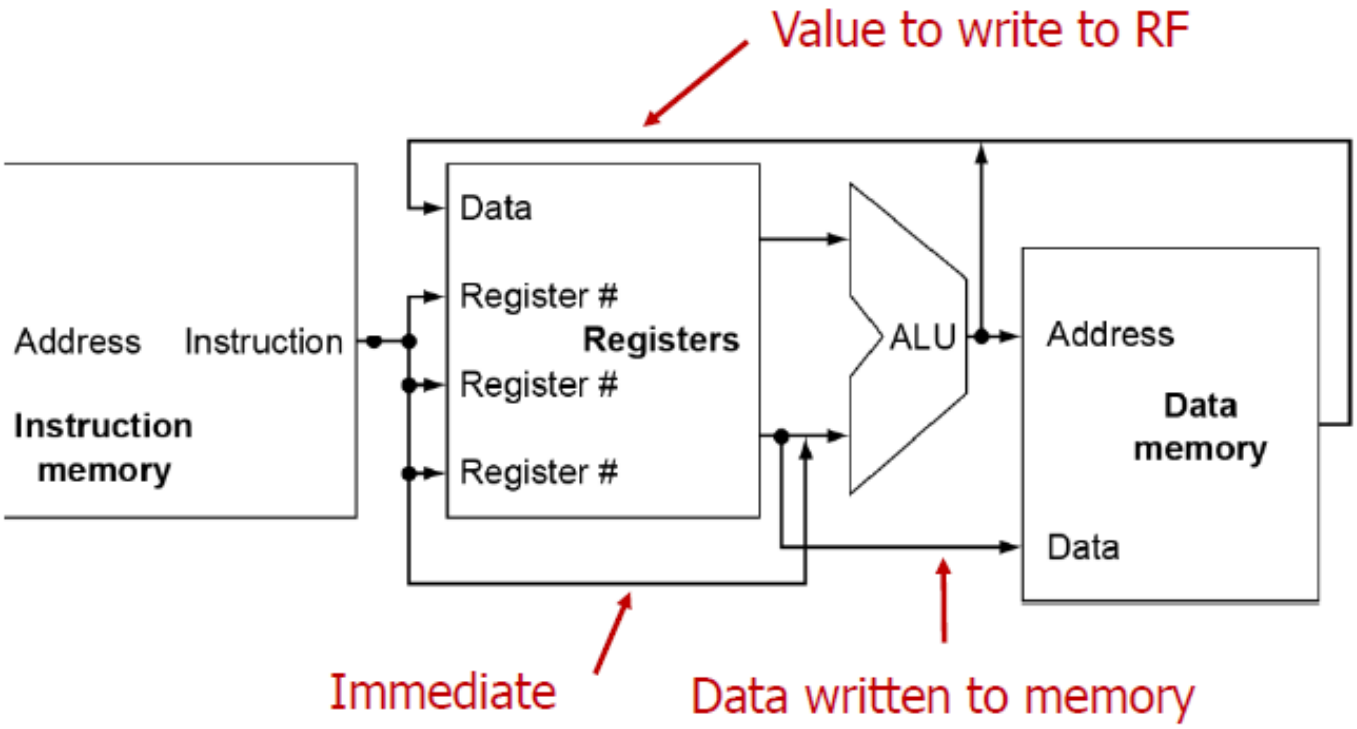
Hardware	R-type	Load/Store	Branch
I-Mem	✓	✓	✓
RF	✓	✓	✓
ALU	Compute result	Compute address	Compare
Data Memory		Read/write	Update PC
RF	Write	Write (load)	

Does not use Data Memory

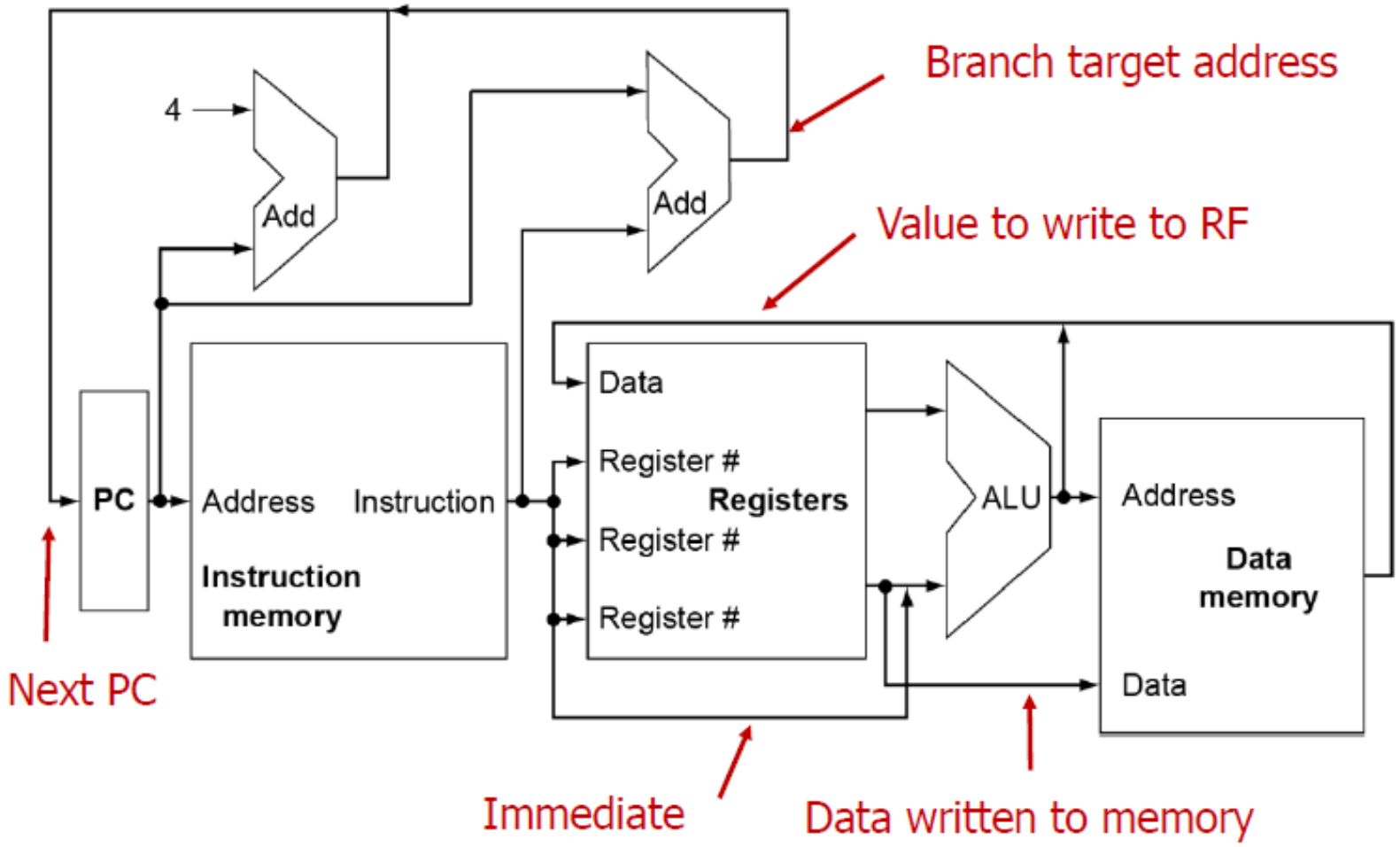
Example:

- ADD instruction is fetched from I-Mem, using the address in PC
- Register rs1 and rs2 are read from the Register File (RF)
- ALU performs addition on the contents of rs1 and rs2
- The result is saved into register rd in the RF

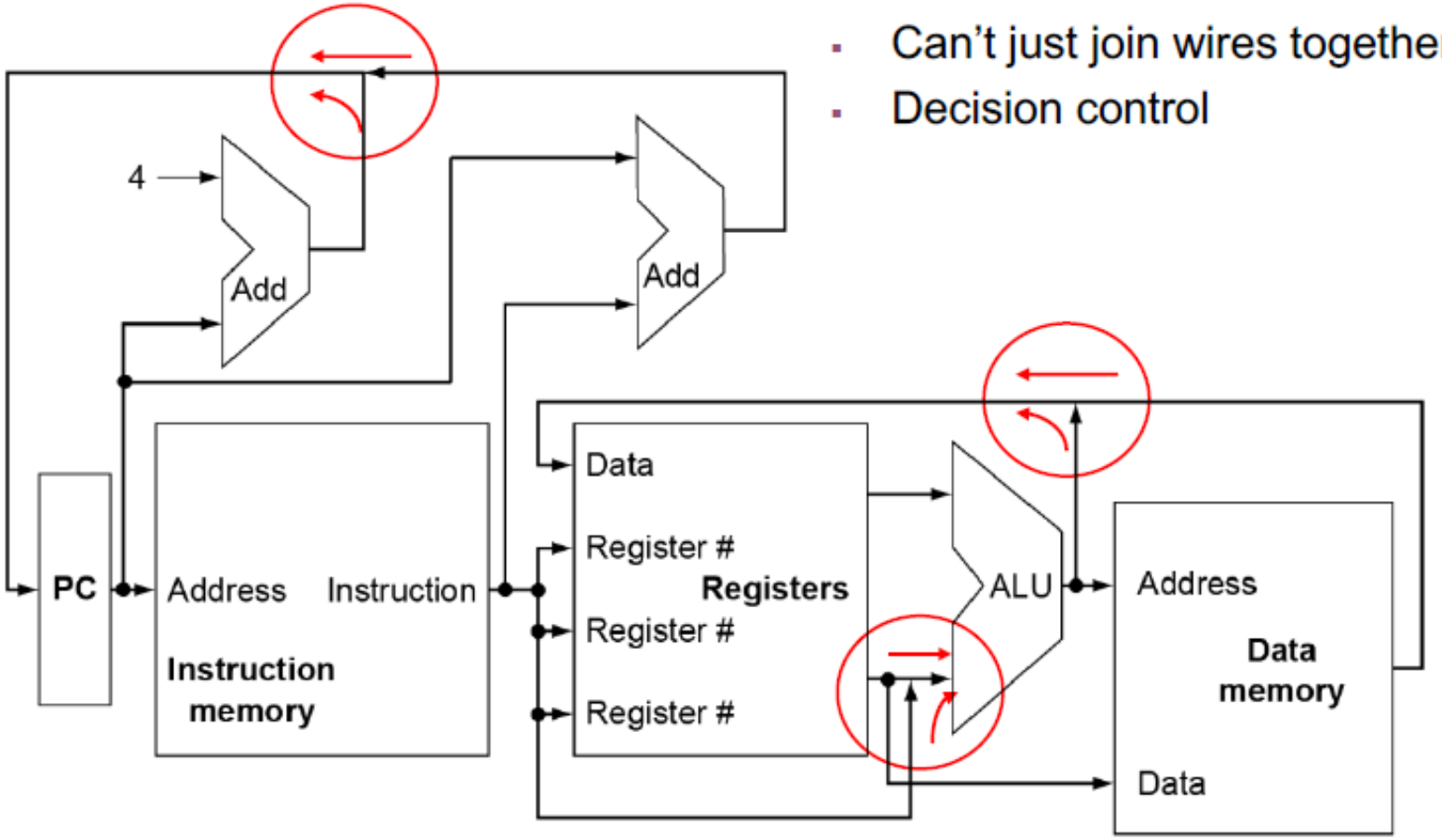
CPU Overview



CPU Overview



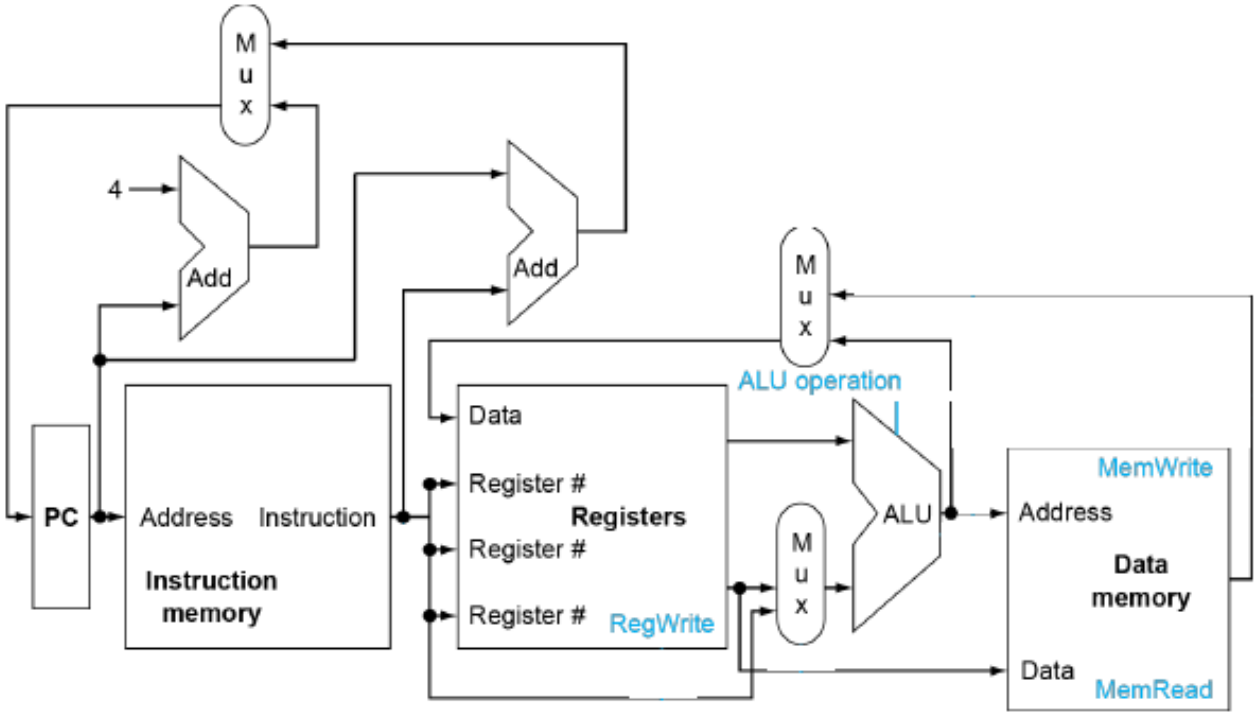
CPU Overview: Missing Components



- Can't just join wires together
- Decision control

What component we can use at these locations ?

CPU Overview: Control Unit



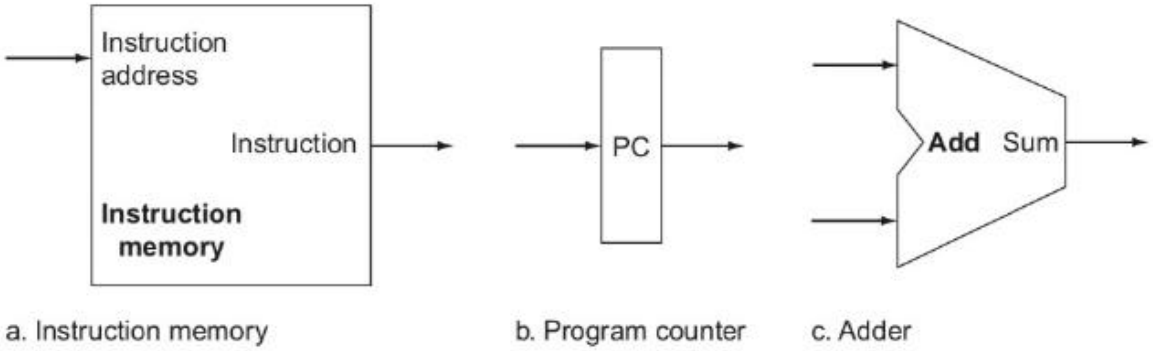
Building a Datapath

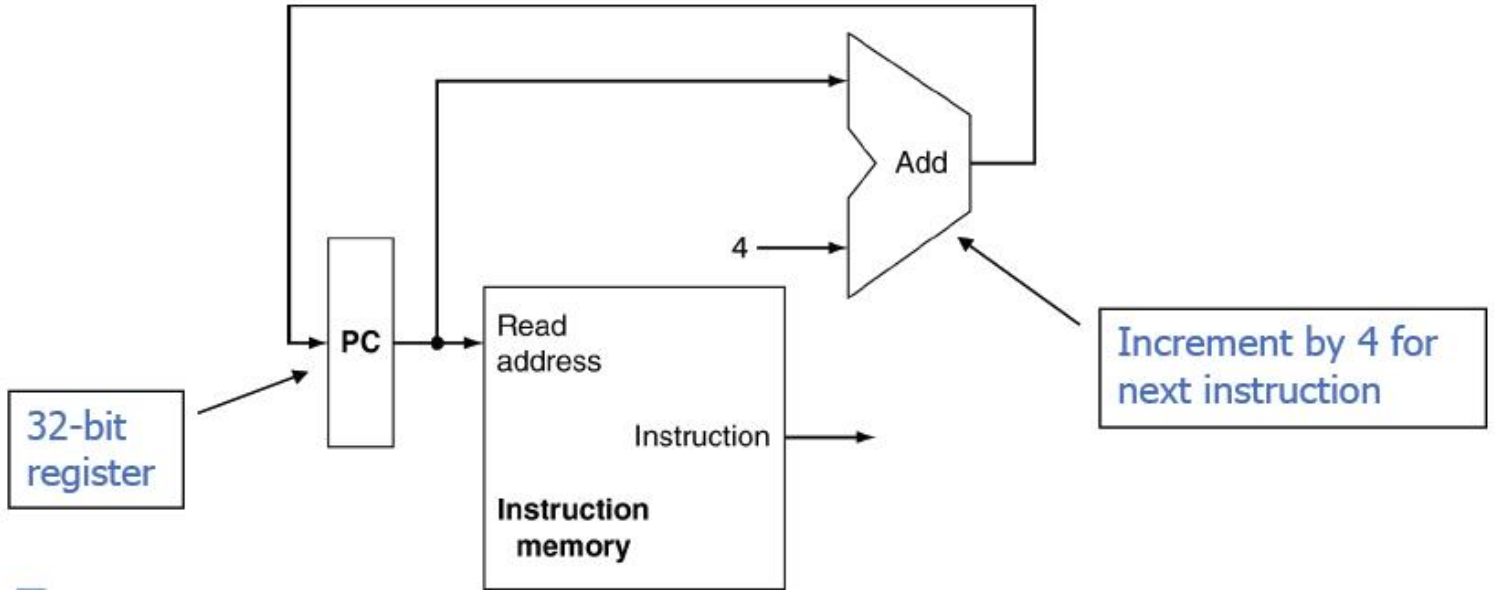
- Let us build a RISC-V datapath incrementally
  - Refining the overview design

- Datapath: Elements that process data and addresses in the CPU
  - Registers, ALUs, MUXes, memories, …

Pay attention to details !

Building a Datapath: Instruction Fetch



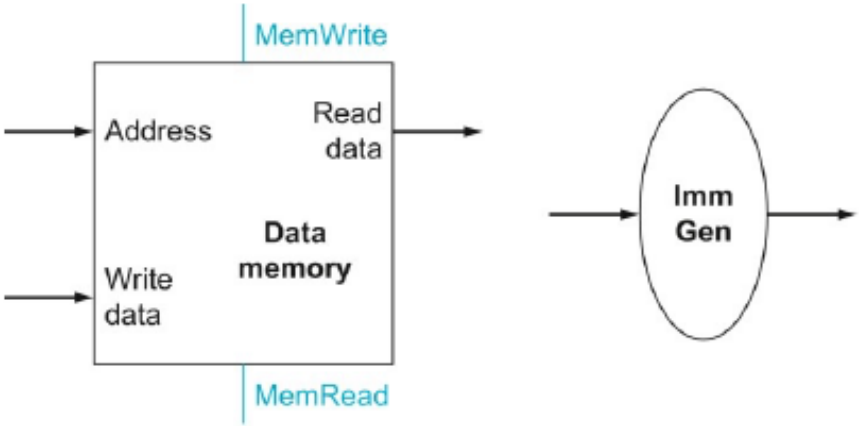


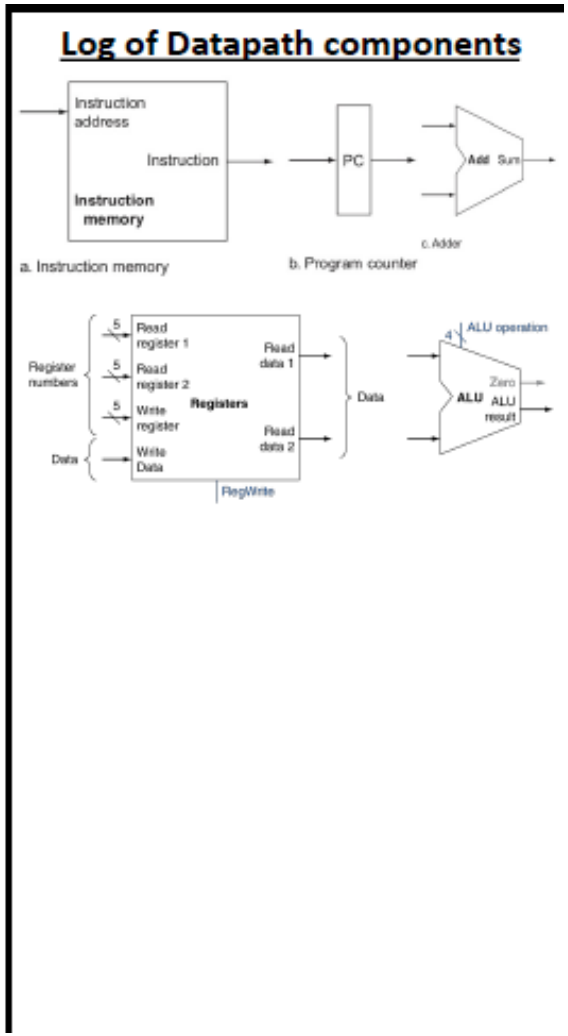
Building a Datapath: R-Format Instructions

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

Building a Datapath: Load/Store Instructions

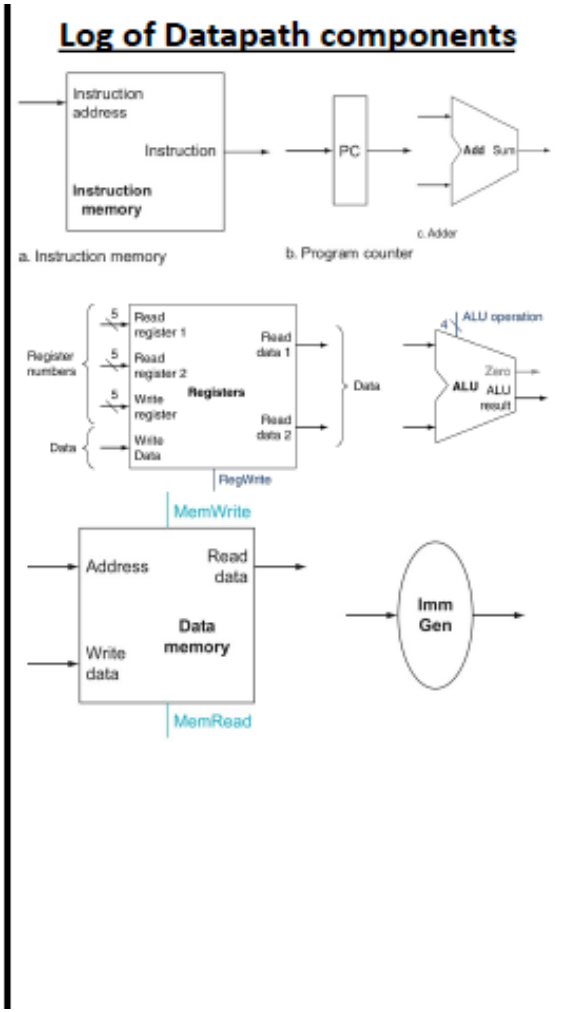
- Load: Read memory and update register
  - lw x1, offset(x2)
- Store: Write register value to memory
  - sw x1, offset(x2)
- Read register operands
- Calculate address using 12-bit offset
  - Use ALU, but sign-extend offset





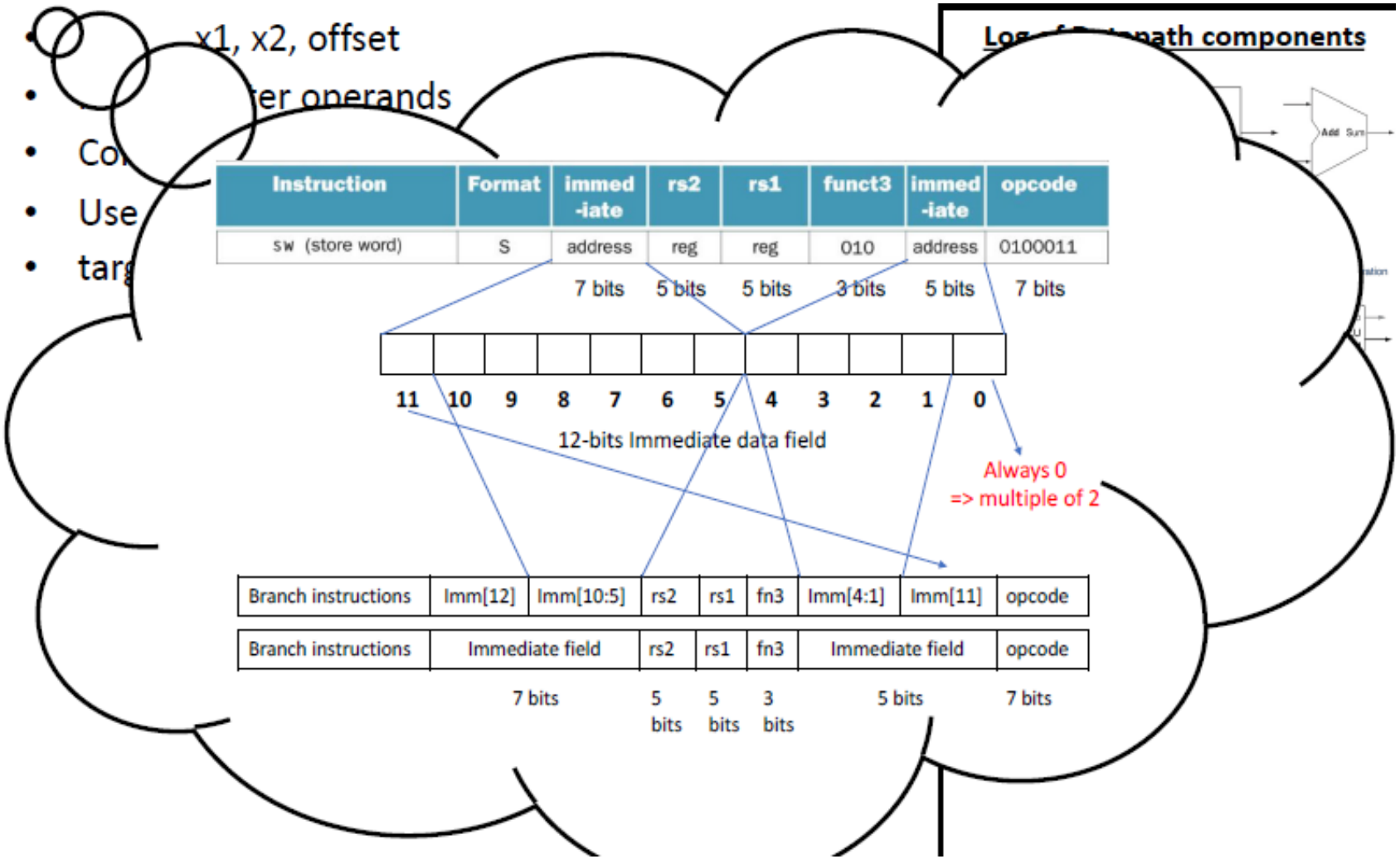
Building a Datapath: Branch Instructions

- beq x1, x2, offset
- Read register operands
- Compare operands
- Use ALU, subtract and check Zero output
- target address
  - Sign-extend displacement
  - Shift left 1 place (halfword displacement)
  - Add to PC value

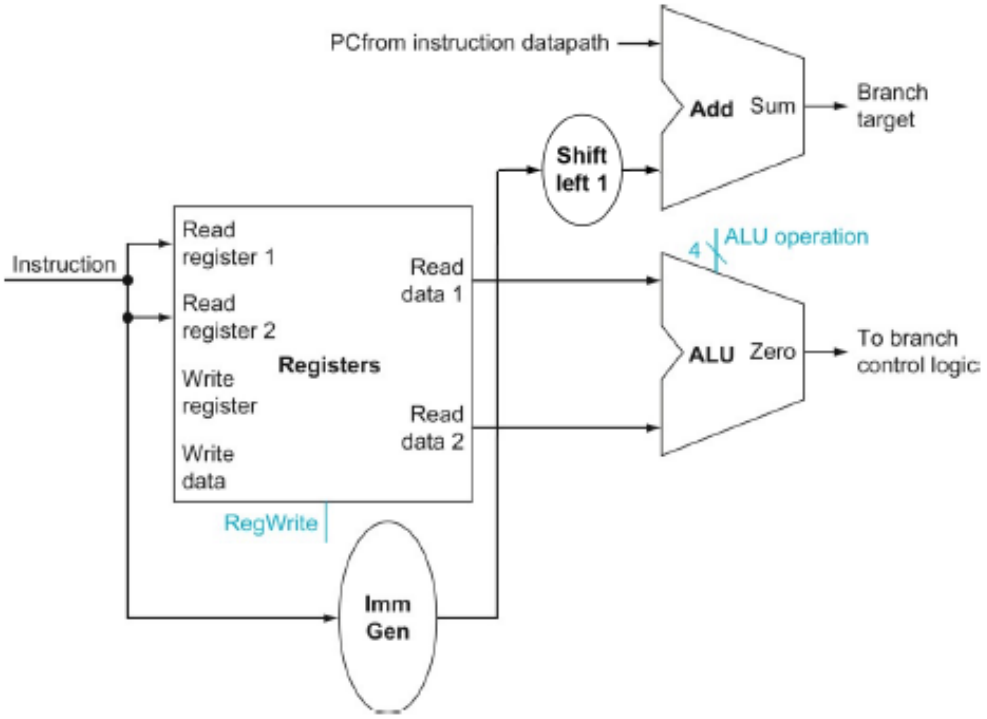


Building a Datapath: Branch Instruction

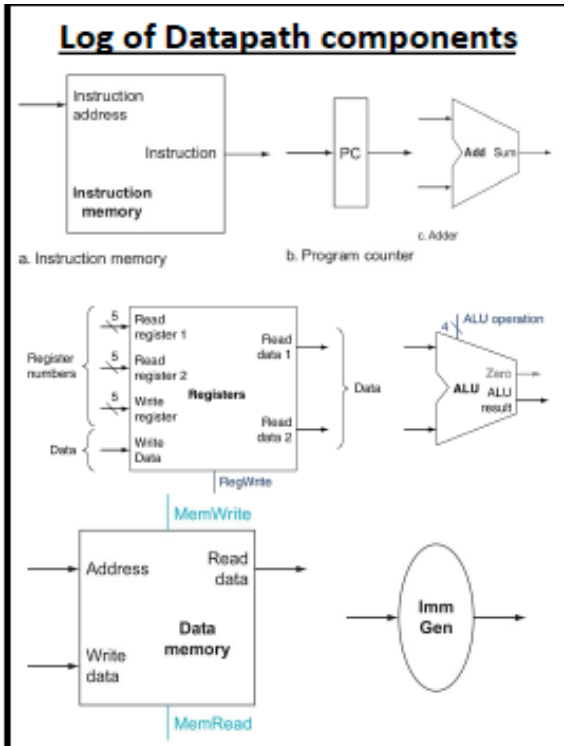
- x1, x2, offset
- [illegible]er operands
- Co[illegible]
- Use
- targ[illegible]



Building a Datapath: Branch Instruction



Equal operands → Branch target address = New PC → **Branch taken**
Else → PC + 4 = PC → **Branch not taken**

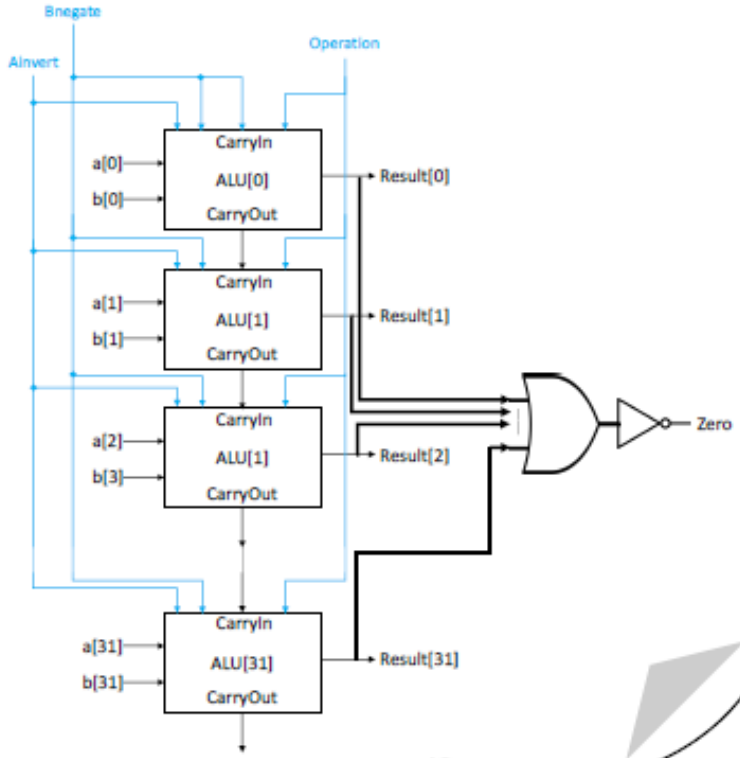


Building a Datapath: Branch Instruction

For branch instruction

Can execute A – B to check if the result = 0 ?

The condition satisfy or not is indicated by **Zero** signal

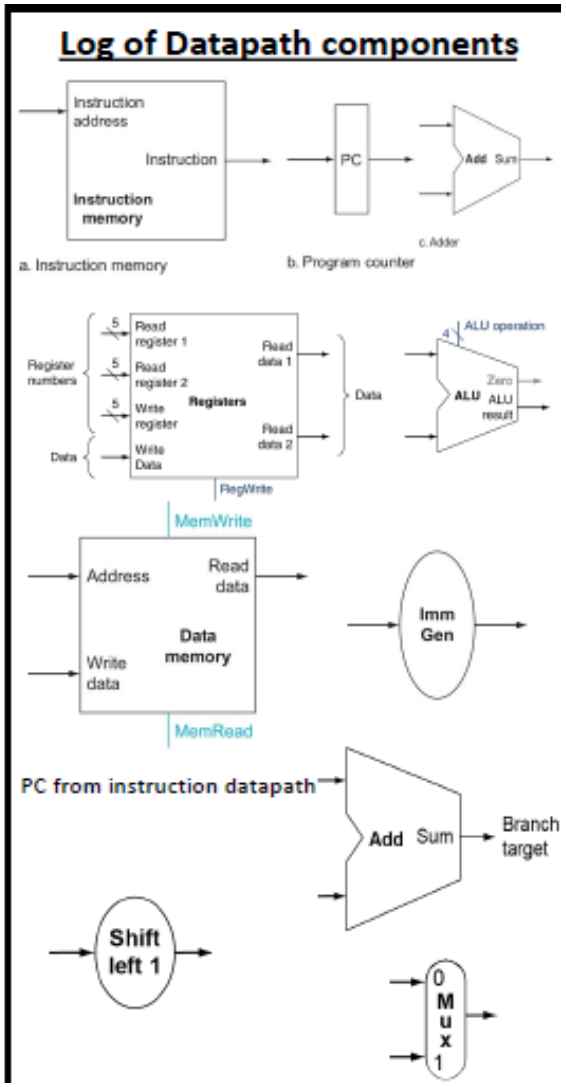


Equal operands
Else

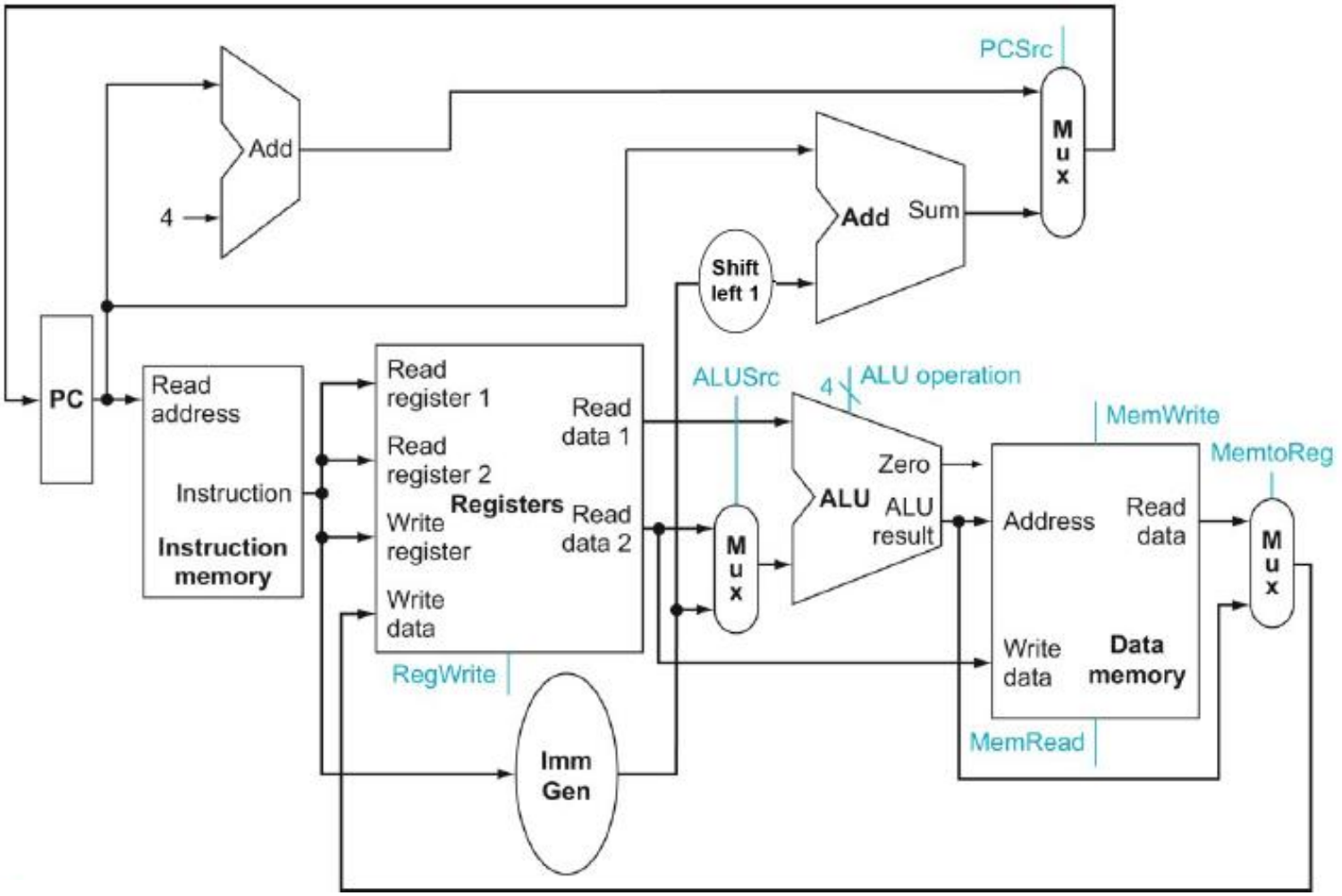
Building a Datapath: Composing the Elements

The operations of R-Type instructions and the memory instructions datapath are quite similar

- R-Type:
  - Use ALU with inputs coming from two registers
  - The value stored into a destination register comes from the ALU
- Memory:
  - Use ALU to calculate the memory address: one input from register, other input from 12-bit offset field
  - The value stored into a destination register comes from the memory (for a load)



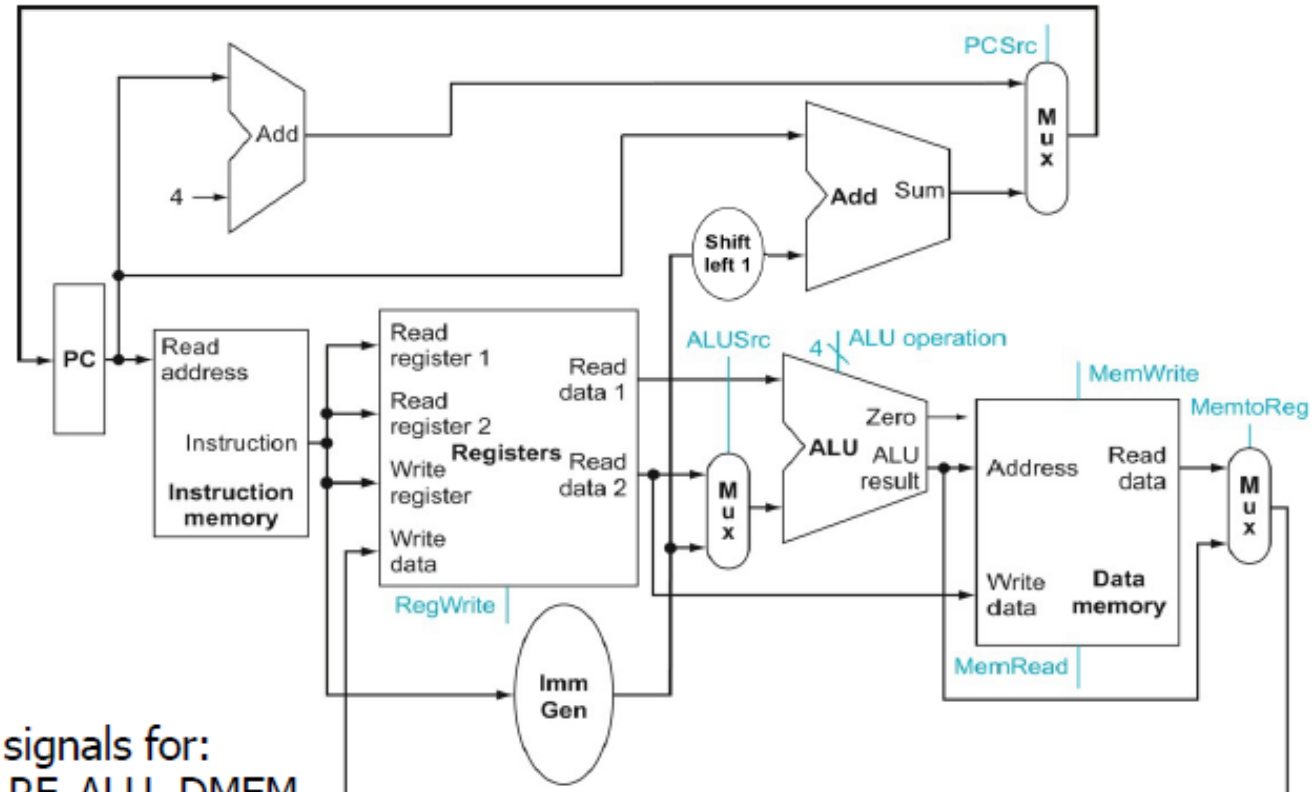
Full Datapath without Control



Datapath Summary

- Datapath can execute an instruction in one clock cycle
- Each datapath element, e.g., RF and ALU, can only do one function at a time
  - They cannot be used twice
  - Hence, we need separate instruction and data memories
- Use multiplexers where alternate data sources are used for different instructions
- We still need to generate the colored signals via Control unit

Full Datapath without Control



Control signals for:
MUXes, RF, ALU, DMEM

ALU operation is not generated by the main control directly. We generate ALU operation first.

ALU Control

The simple implementation of a RISC-V ALU subset covers the following instructions

- Load(lw) / Store(sw) → F = add
- Branch (beq) → F = subtract
- R-type → F depends on the opcode function field

ALU control lines	Function
0010	AND
0011	OR
0000	add
0001	subtract

Checking Opcode

ALU operation is generated in two steps

- We could generate it directly from opcode, funct3, and funct7

The main control checks opcode (and opcode only) and generates a 2-bit signal ALUOp that indicates the instruction type

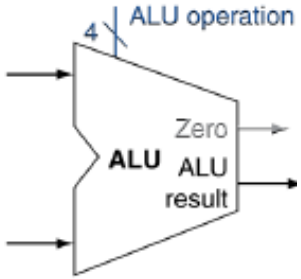
- Check opcode only (not funct3 or funct7) is faster

ALU control uses 2-bit ALUOp, instead of opcode directly

Instruction	ALUOp
lw	00
sw	00
beq	01
R-type	10

ALU Control

- Generating a 4-bit ALU control signal based on 2-bit ALUOp
- ALUOp is derived from opcode



Instruction opcode	ALUOp	Operation	funct7	funct3	ALU function	ALU operation
lw	00	load word	xxx xxxx	xxx	add	0000
sw	00	store word	xxx xxxx	xxx	add	0000
beq	01	branch if equal	xxx xxxx	xxx	subtract	0001
R-type	10	add	000 0000	000	add	0000
		subtract	010 0000	000	subtract	0001
		and	000 0000	111	and	0010
		or	000 0000	110	or	0011

ALU Control

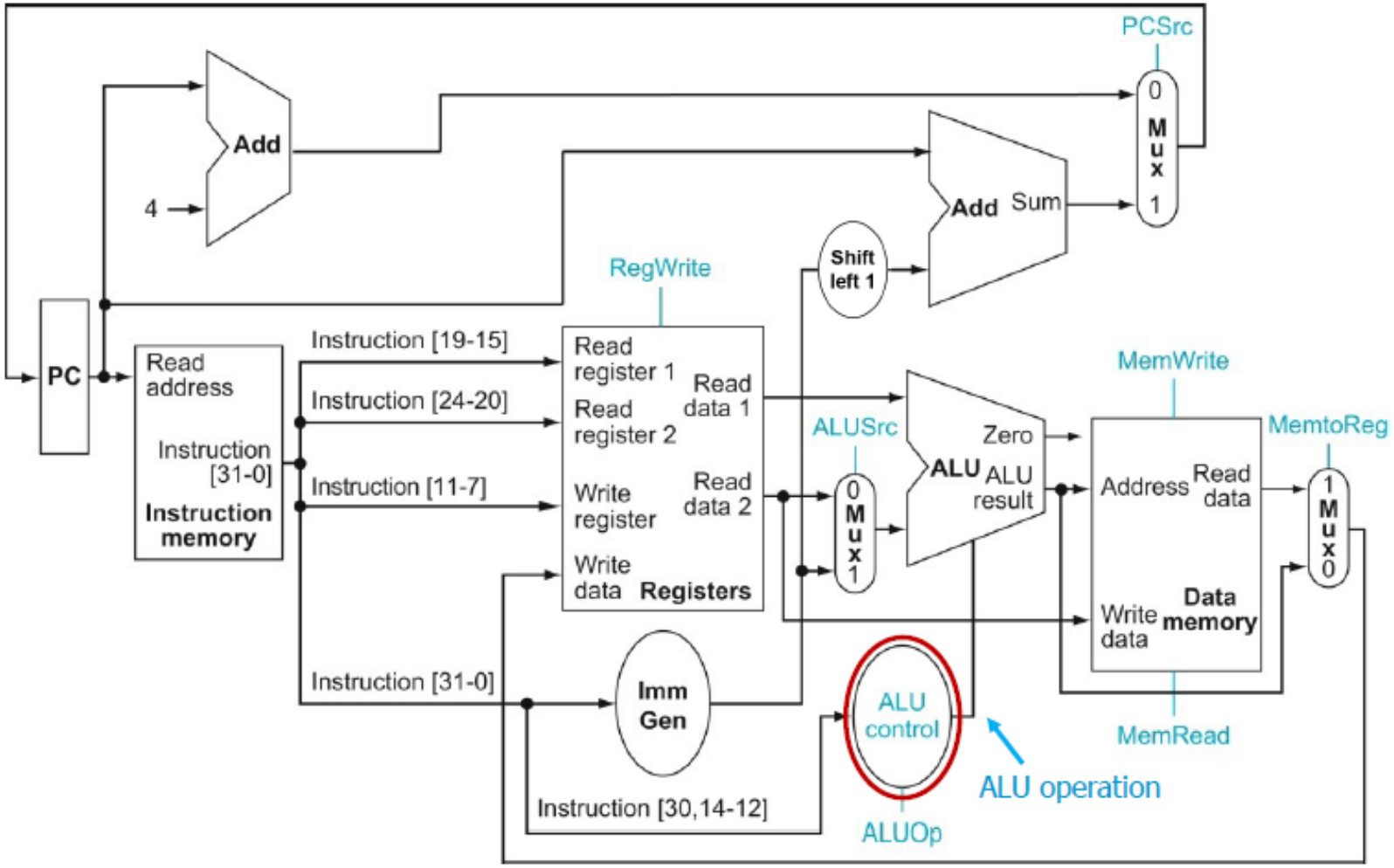
Instruction opcode	ALUOp	Operation	funct7	funct3	ALU function	ALU operation
lw	00	load word	xxx xxxx	xxx	add	0000
sw	00	store word	xxx xxxx	xxx	add	0000
beq	01	branch if equal	xxx xxxx	xxx	subtract	0001
R-type	10	add	000 0000	000	add	0000
		subtract	010 0000	000	subtract	0001
		and	000 0000	111	and	0010
		or	000 0000	110	or	0011

ALUOp		Funct7 field							Funct3 field			
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	Operation
0	0	X	X	X	X	X	X	X	X	X	X	0000
X	1	X	X	X	X	X	X	X	X	X	X	0001
1	X	0	0	0	0	0	0	0	0	0	0	0000
1	X	0	1	0	0	0	0	0	0	0	0	0001
1	X	0	0	0	0	0	0	0	1	1	1	0010
1	X	0	0	0	0	0	0	0	1	1	0	0011

ALUOp != 0b11

Need 1 bit in funct7
and 3 bits in funct3

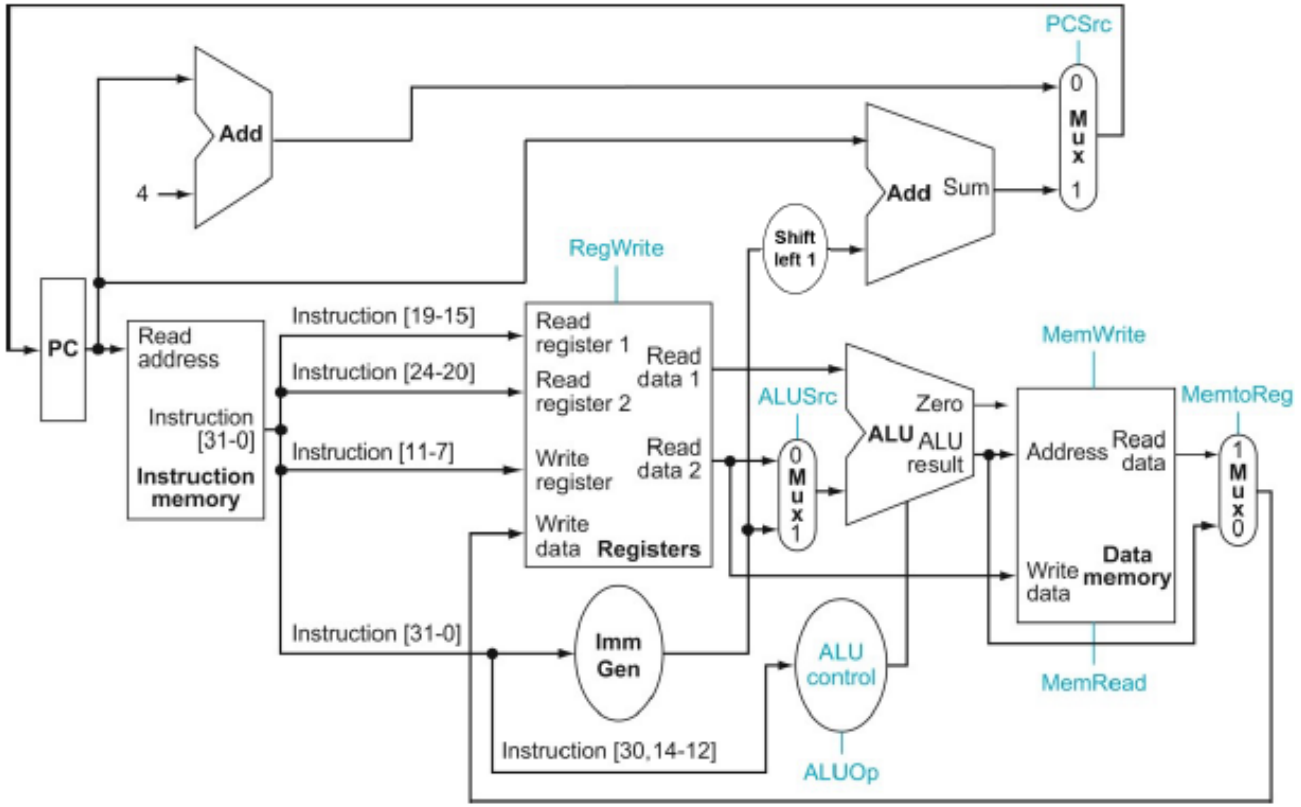
Datapath with ALU Control



Main Control Unit

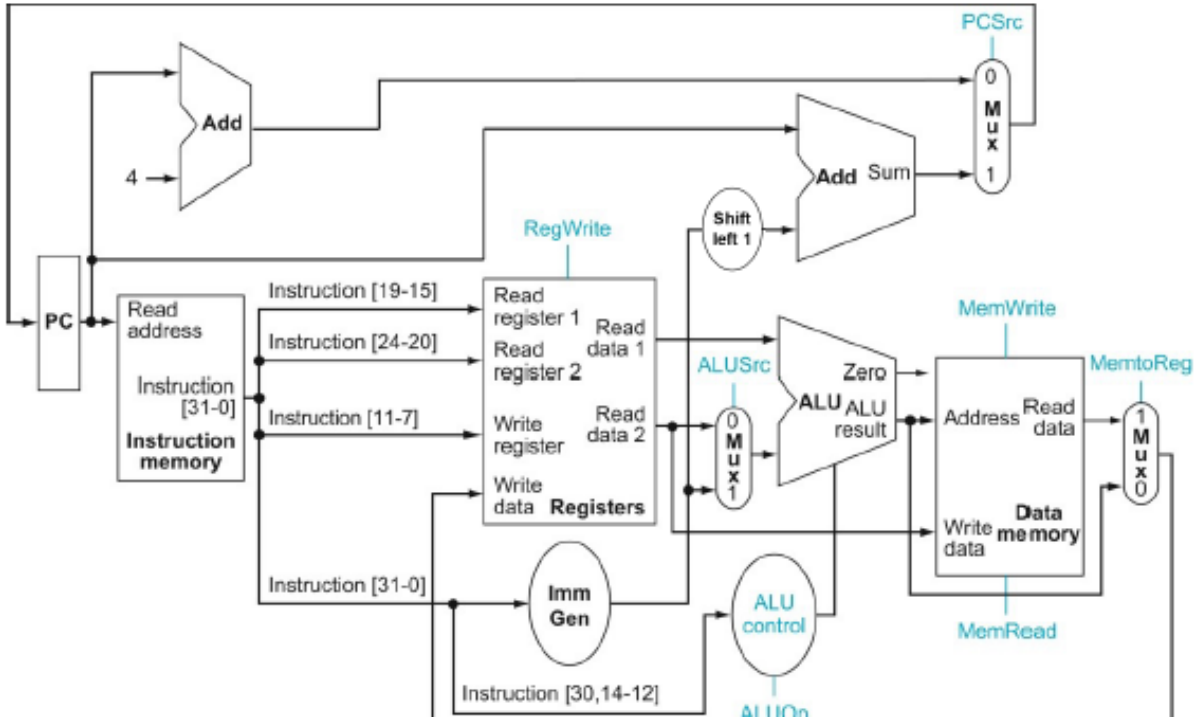
- We have worked on the ALU control signals
- We can now focus on the remaining control signals which are

- ALUOp
- RegWrite
- ALUSrc
- PCSrc
- MemWrite
- MemRead
- MemtoReg



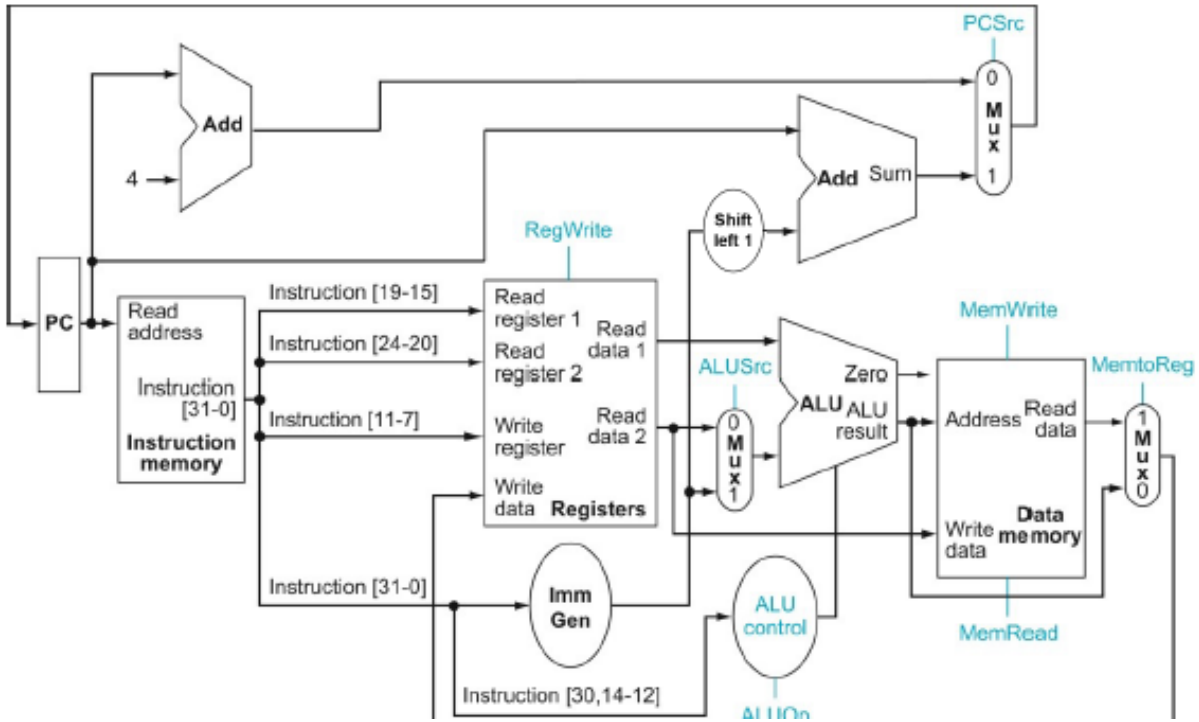
Main Control Unit: Control Signals

Signal name	Effect when deasserted	Effect when asserted
RegWrite	None.	The register on the Write register input is written with the value on the Write data input.
ALUSrc	The second ALU operand comes from the second register file output (Read data 2).	The second ALU operand is the sign-extended, 12 bits of the instruction.



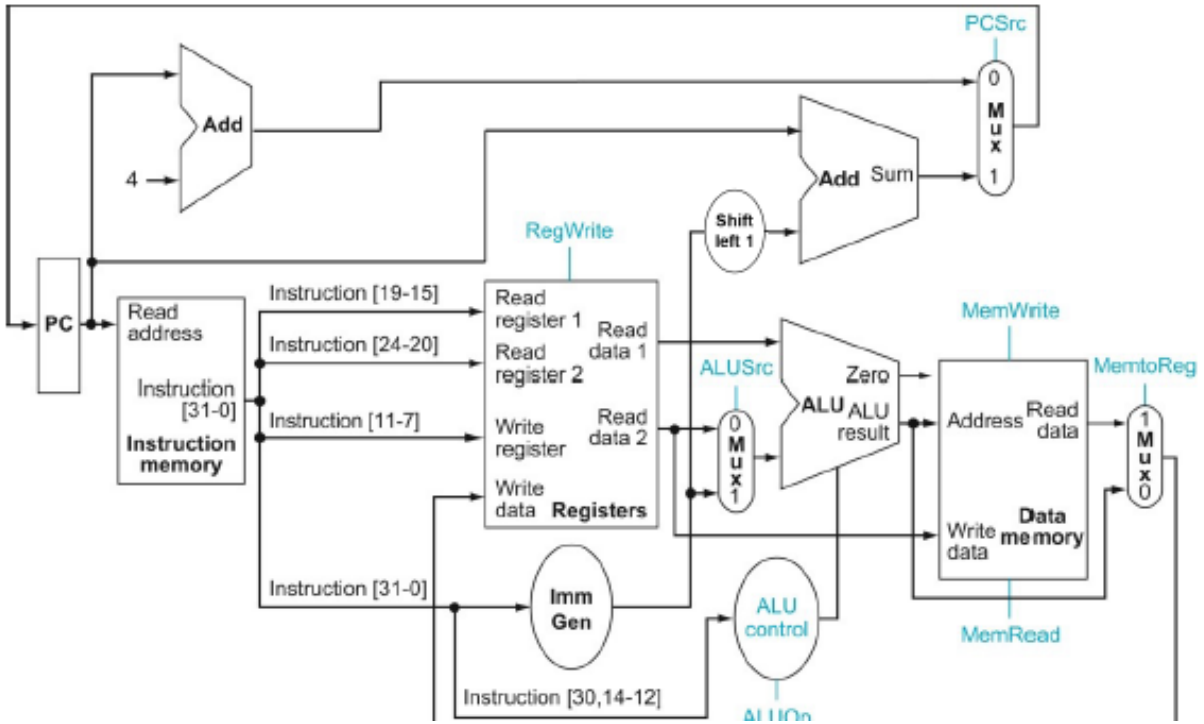
Main Control Unit: Control Signals

Signal name	Effect when deasserted	Effect when asserted
PCSrc	The PC is replaced by the output of the adder that computes the value of PC + 4.	The PC is replaced by the output of the adder that computes the branch target.
MemRead	None.	Data memory contents designated by the address input are put on the Read data output.

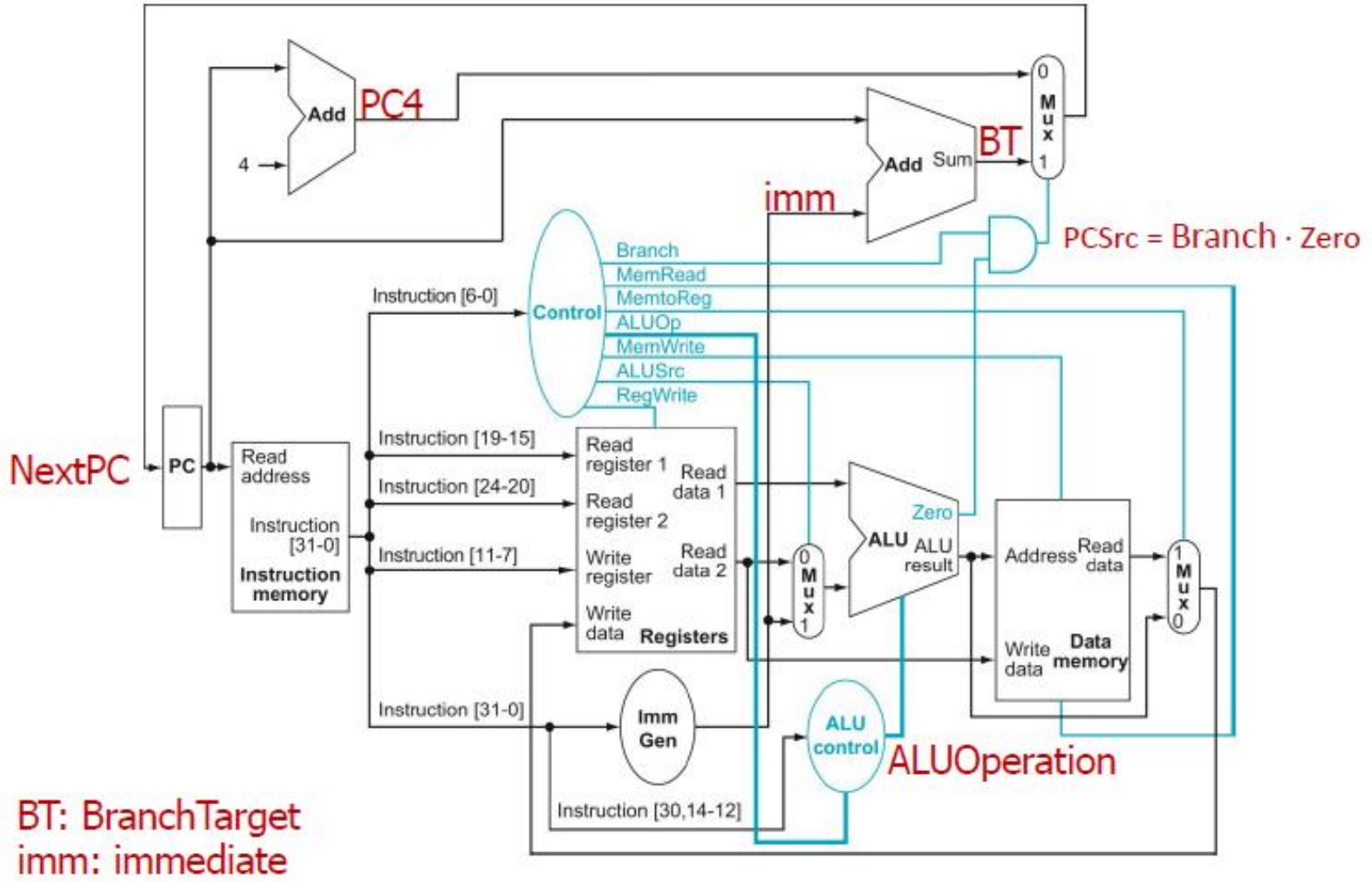


Main Control Unit: Control Signals

Signal name	Effect when deasserted	Effect when asserted
MemWrite	None.	Data memory contents designated by the address input are replaced by the value on the Write data input.
MemtoReg	The value fed to the register Write data input comes from the ALU.	The value fed to the register Write data input comes from the data memory.

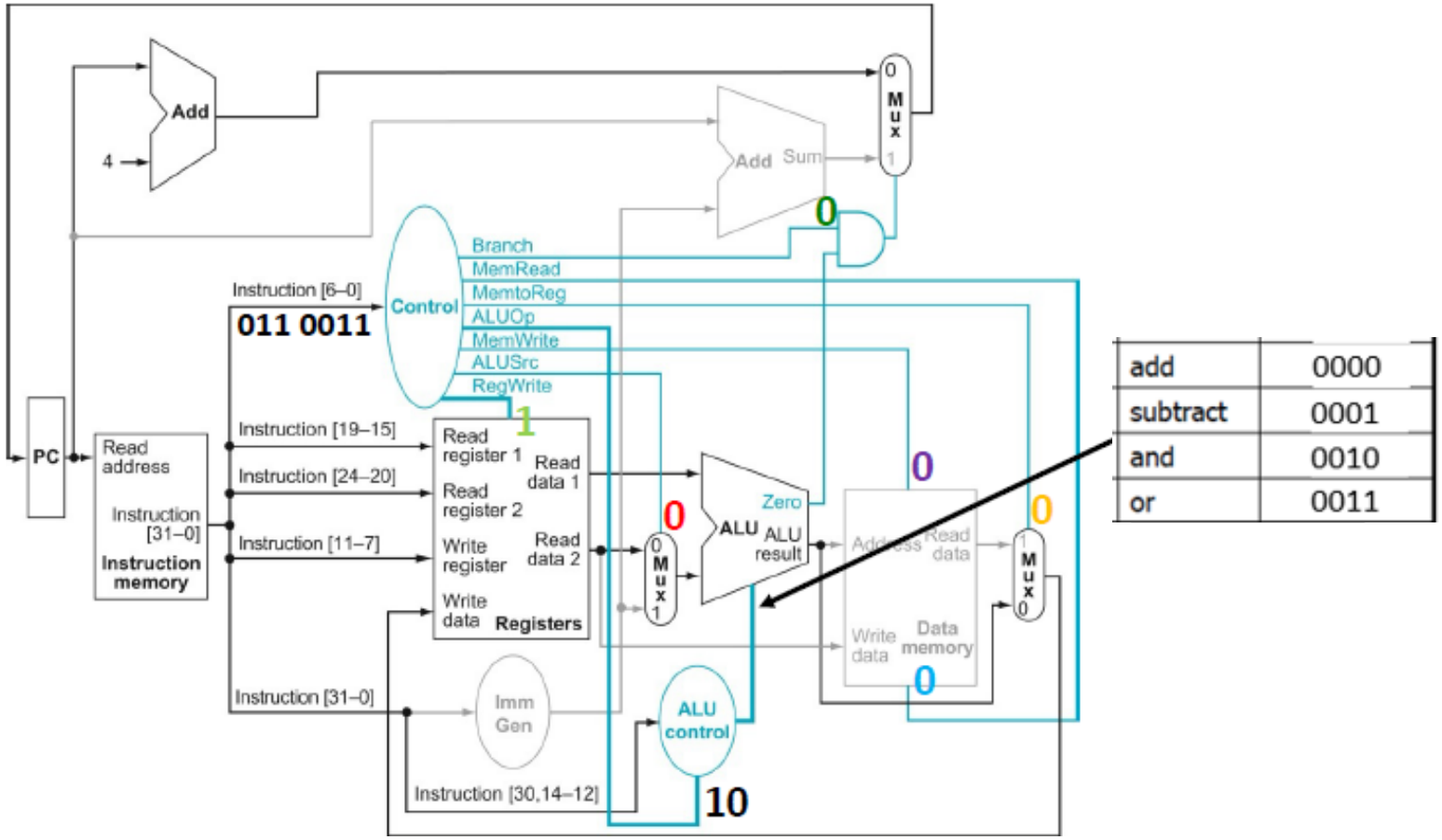


Full Datapath with Control



Operation of Datapath: R-Type Instruction

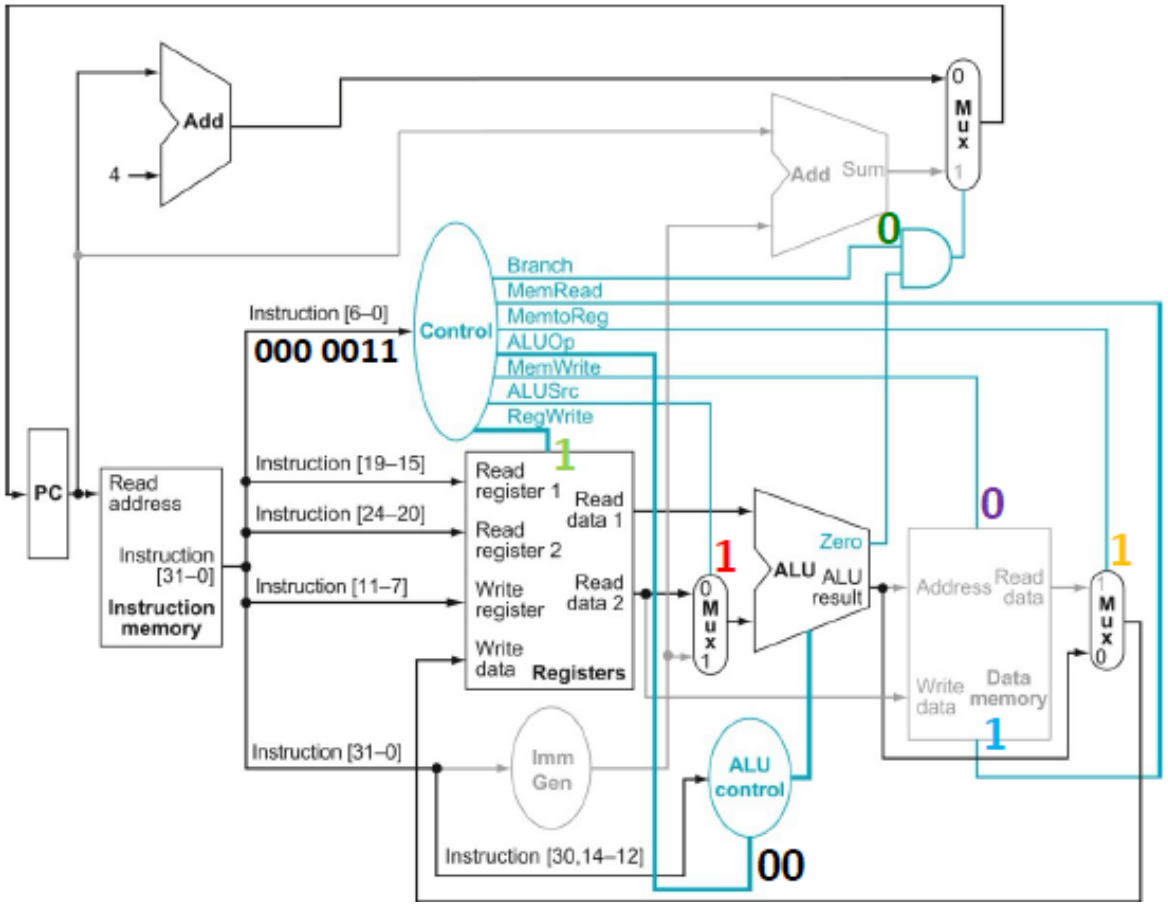
Inst.	Opcode	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch
R-Type	011 0011						



add	0000
subtract	0001
and	0010
or	0011

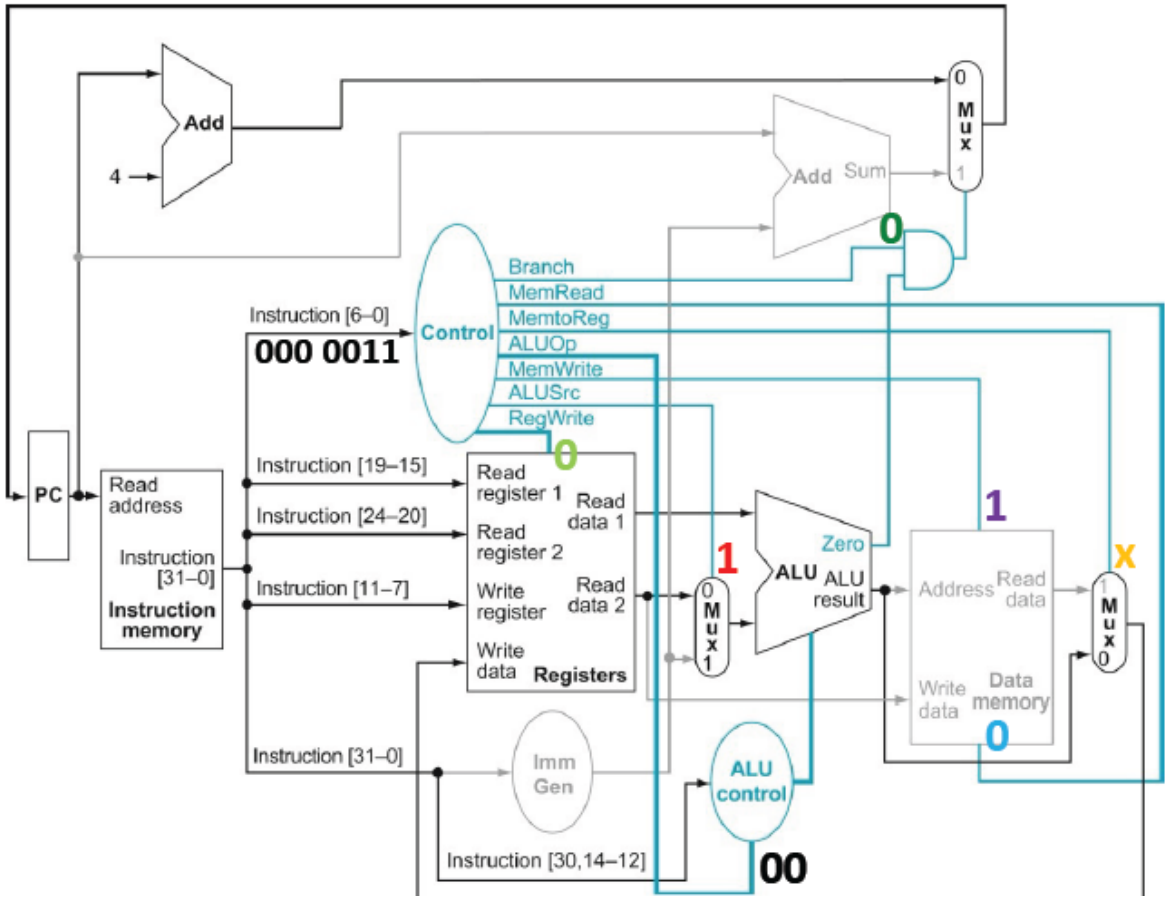
Operation of Datapath: Load (lw)

Inst.	Opcode	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch
lw	000 0011						



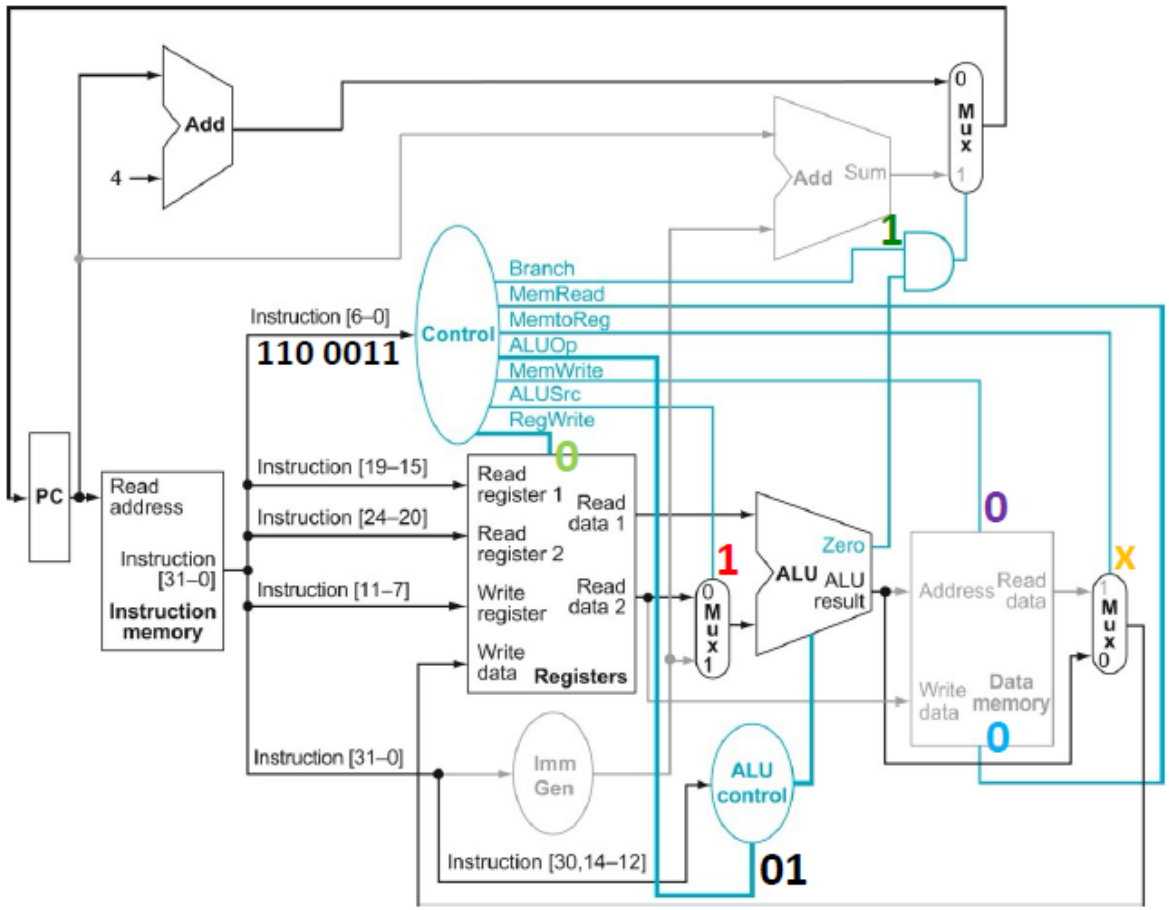
Operation of Datapath: Store (sw)

Inst.	Opcode	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch
sw	010 0011						



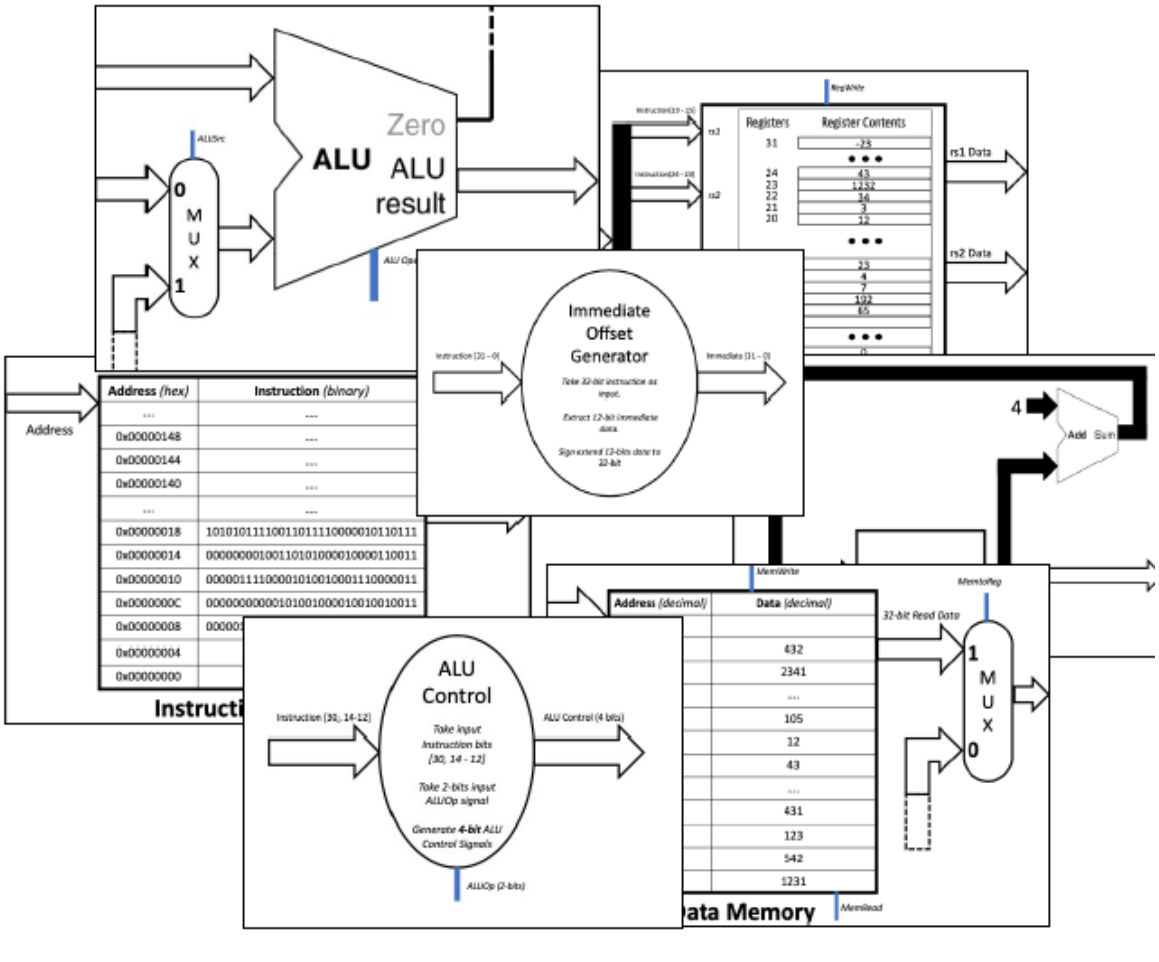
Operation of Datapath: Branch (beq)

Inst.	Opcode	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch
beq	110 0011						



Group Activity

- Construct RISC-V Processor with the following components
- Execute first two instructions (one after the other) pointed by PC
- Write corresponding values at each signal
- Update register and memory contents for both instructions



These extra signals are provided to you for making connections

Thank you