The Processor: Single-Cycle Implementation

Z. Jerry Shi
Department of Computer Science and Engineering
University of Connecticut

CSE3666: Introduction to Computer Architecture

Outline

We will first implement a simple single-cycle RISC-V processor

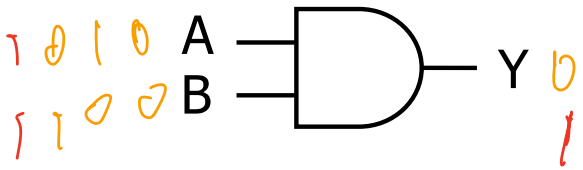
We will then improve it to a more realistic pipelined version (next week)

(multi-cycle) view

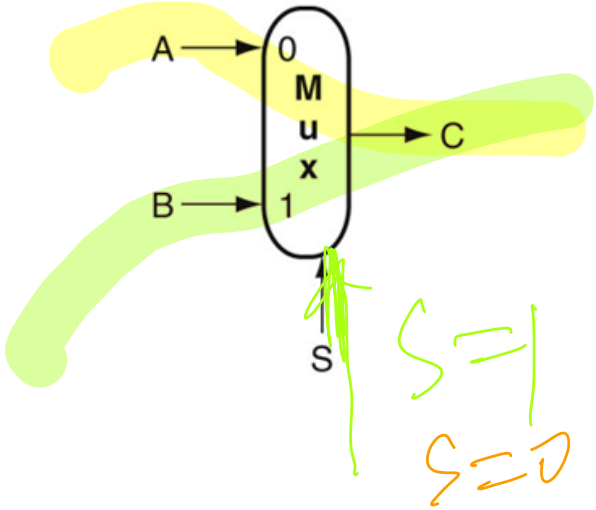
- Overview, after review
- Datapath
- Control
- Performance

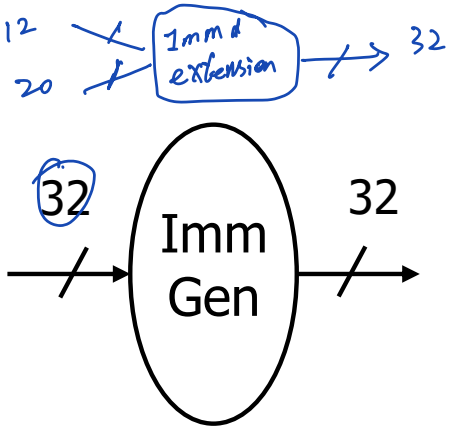
Reading: Chapter 4.1 – 4.4

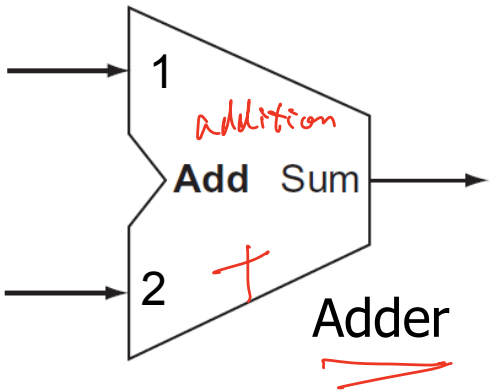
Review of building blocks

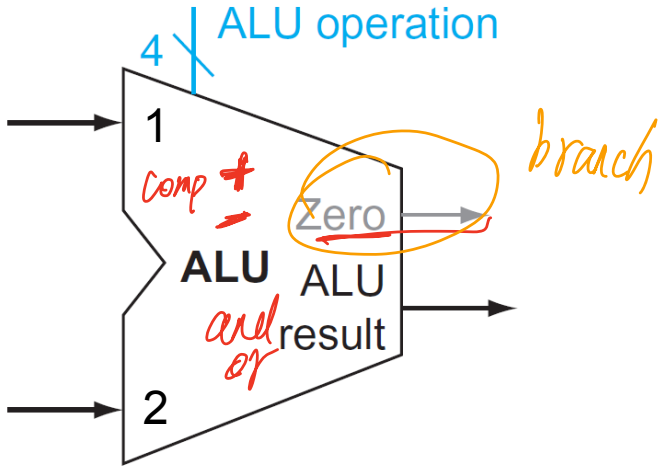


AND gate



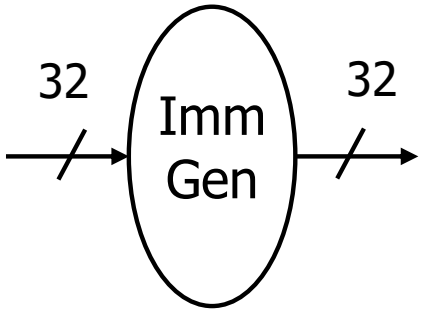






ImmGen

- ImmGen generates 32-bit immediate for I, S, SB, U, and UJ types
- We can generate each bit individually



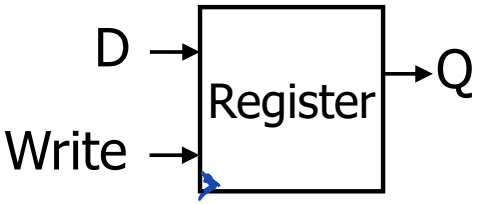
7	6	5	4	3	2	1	0	
i27	i26	i25	i24	i23	i22	i21	i20	I
"	"	"	i11	i10	i9	i8	i7	S
"	"	"	"	"	"	"	0	SB
"	"	"	"	"	"	"	"	U
0	0	0	0	0	0	0	"	UJ
2	2	2	3	3	3	3	3	unique inputs

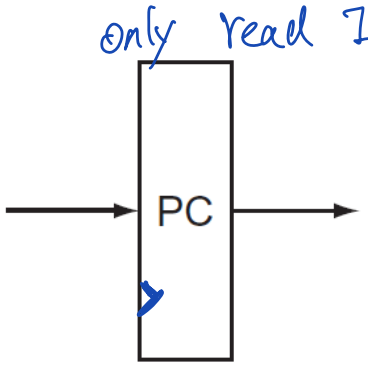
- How about R-type?

The Lower 8 bits from Figure 4.18
Figures 4.17 and 4.18 explain why they make immediate so complicated

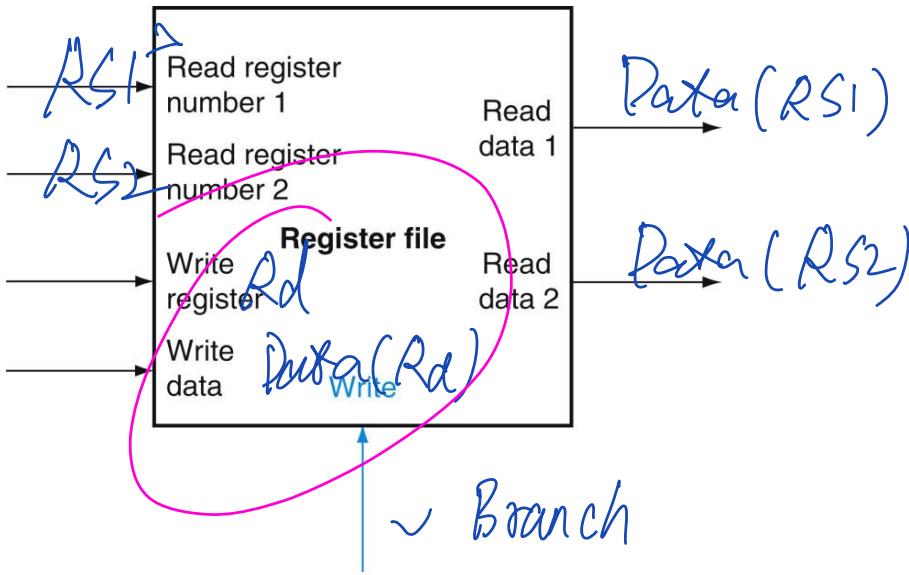
Sequential elements

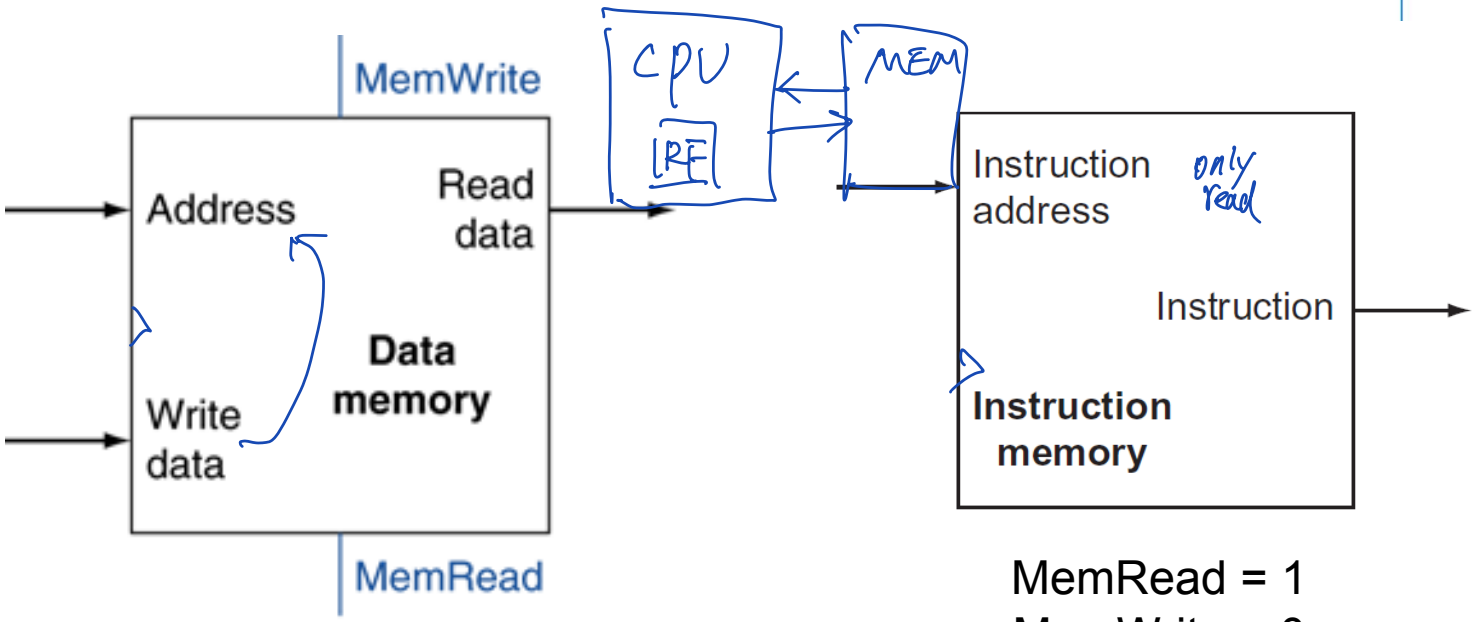
Clock signal is not shown!





Write = 1



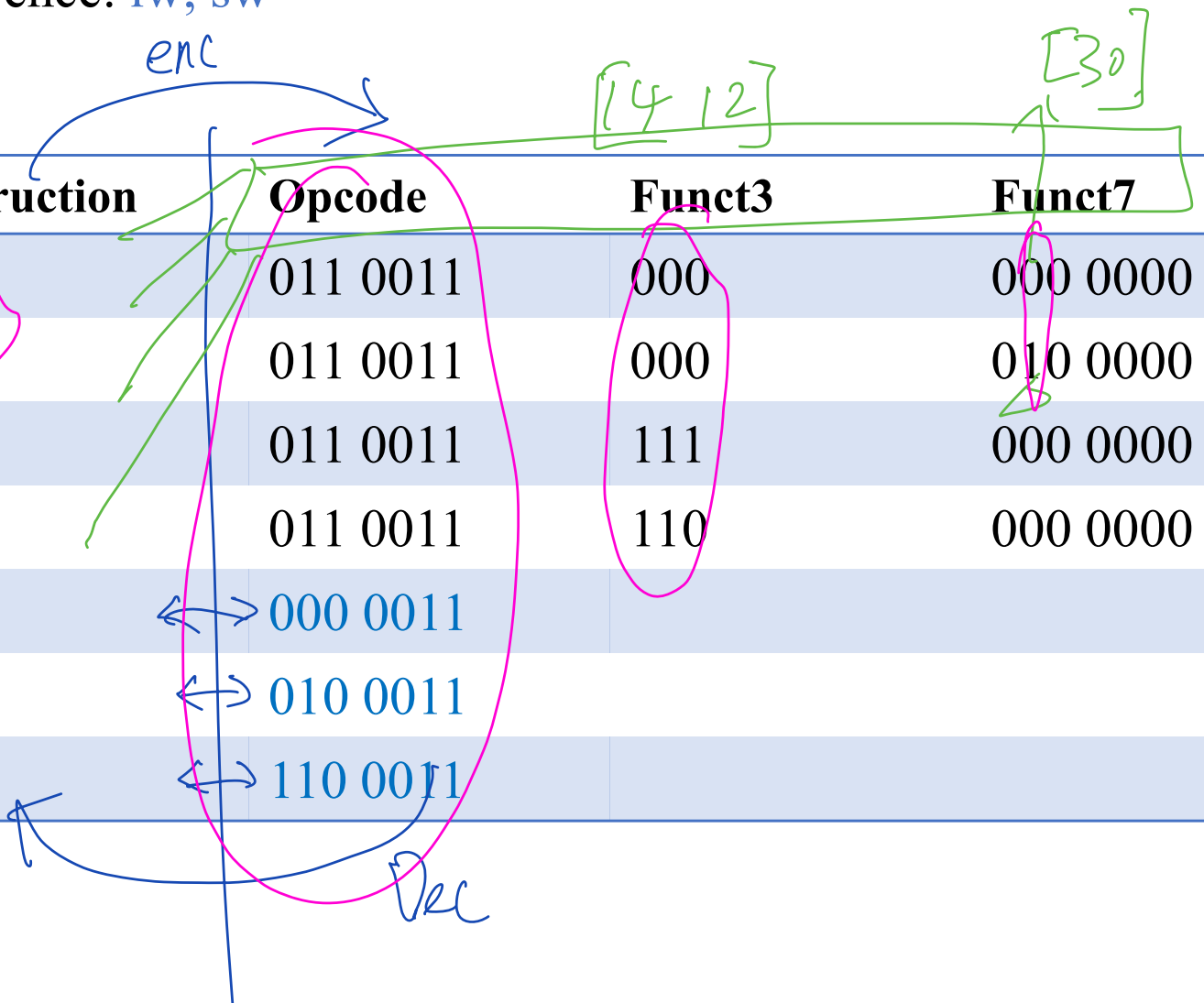


MemRead = 1
MemWrite = 0

A Subset of RISC-V Instructions

- Simple subset that shows most aspects
  - Arithmetic/logical: add, sub, and, or
  - Memory reference: lw, sw
  - Branch: beq

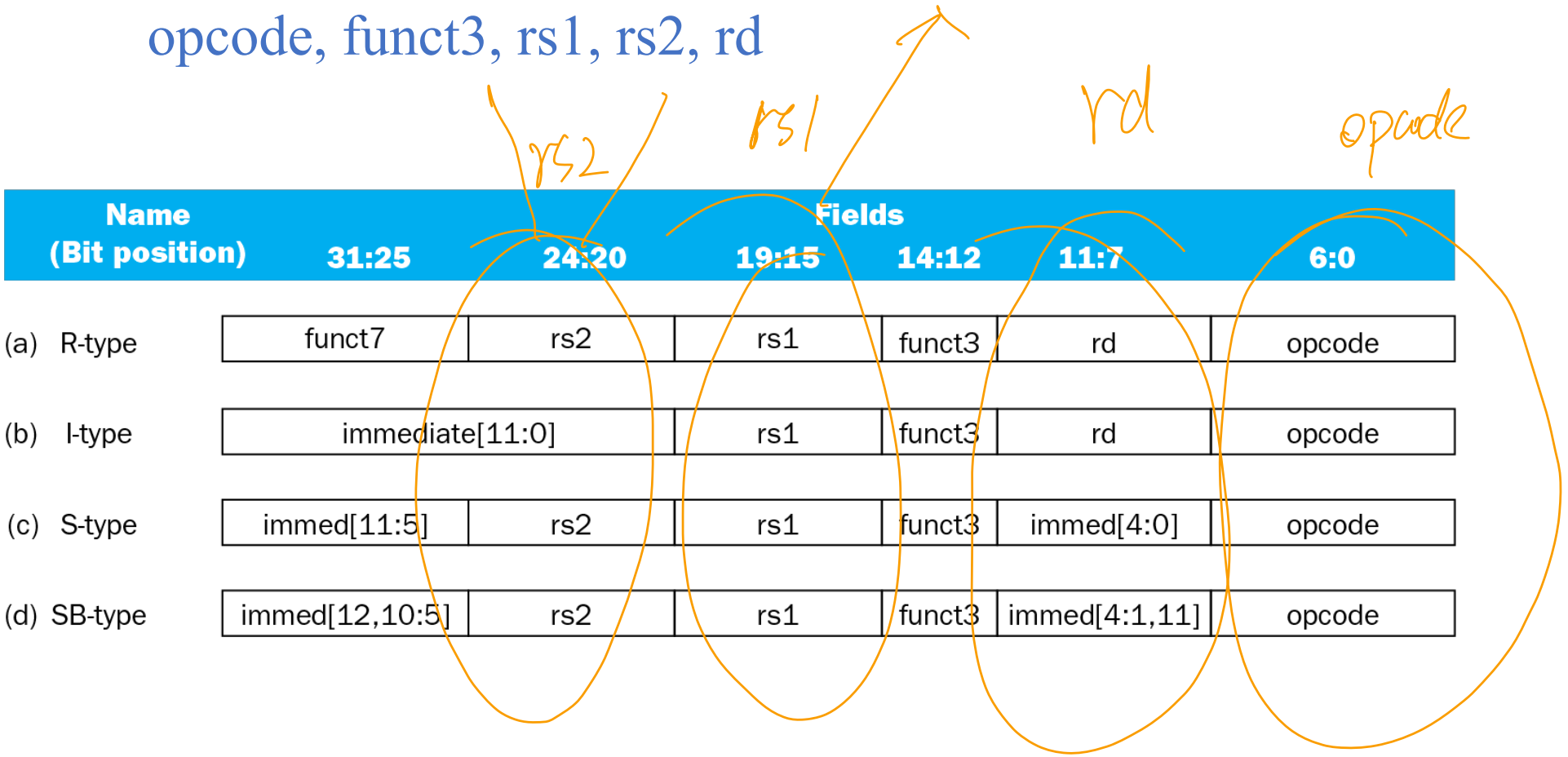
Type	Instruction	Opcode	Funct3	Funct7
R-Type	add	011 0011	000	000 0000
R-Type	sub	011 0011	000	010 0000
R-Type	and	011 0011	111	000 0000
R-Type	or	011 0011	110	000 0000
I-Type	lw	000 0011		
S-Type	sw	010 0011		
SB-Type	beq	110 0011		



Instruction encoding

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?

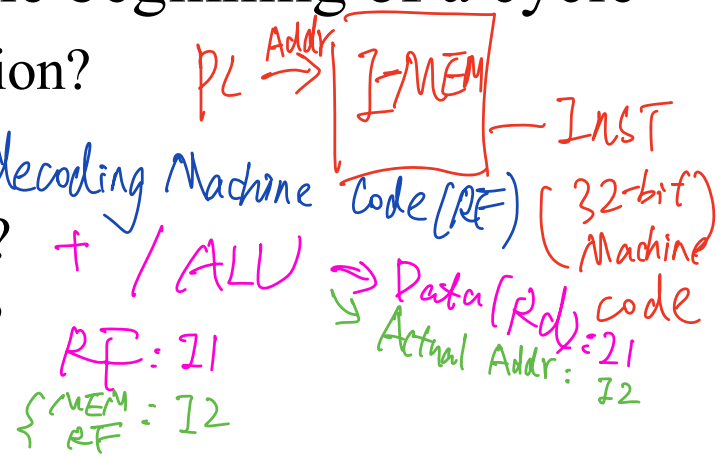
Execution of instructions

- What are the steps to execute instructions?

```
I1    add     rd, rs1, rs2     # sub/and/or
I2    lw      rd, offset(rs1)
I3    sw      rs2, offset(rs1)
I4    beq     rs1, rs2, offset
```

rs1 - rs2 == 0?

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



Steps in 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)	

Not using 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 two register values Reg[rs1] and Reg[rs2]
The result is saved into register rd in the RF

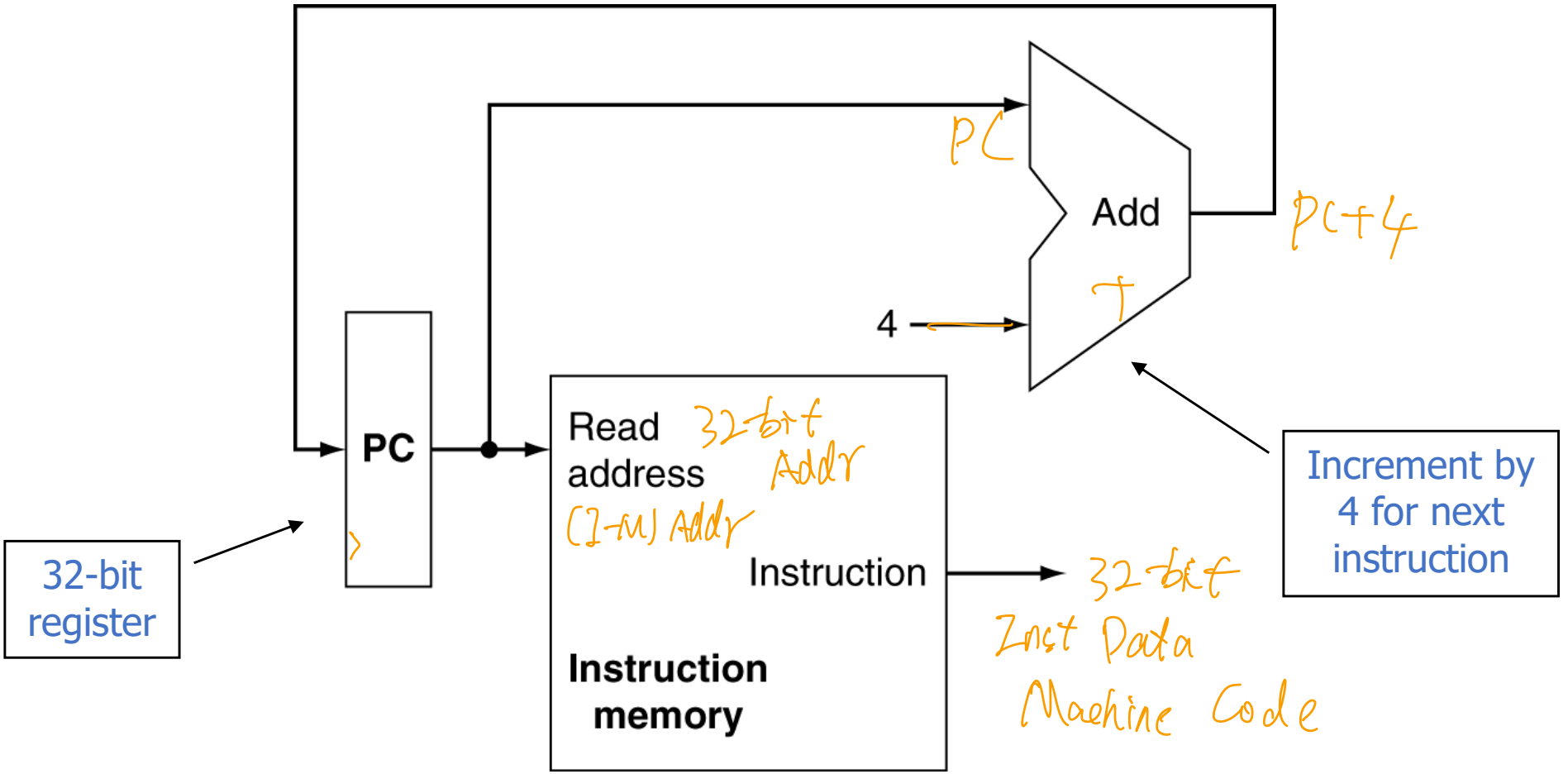
Building a Datapath

- 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 !

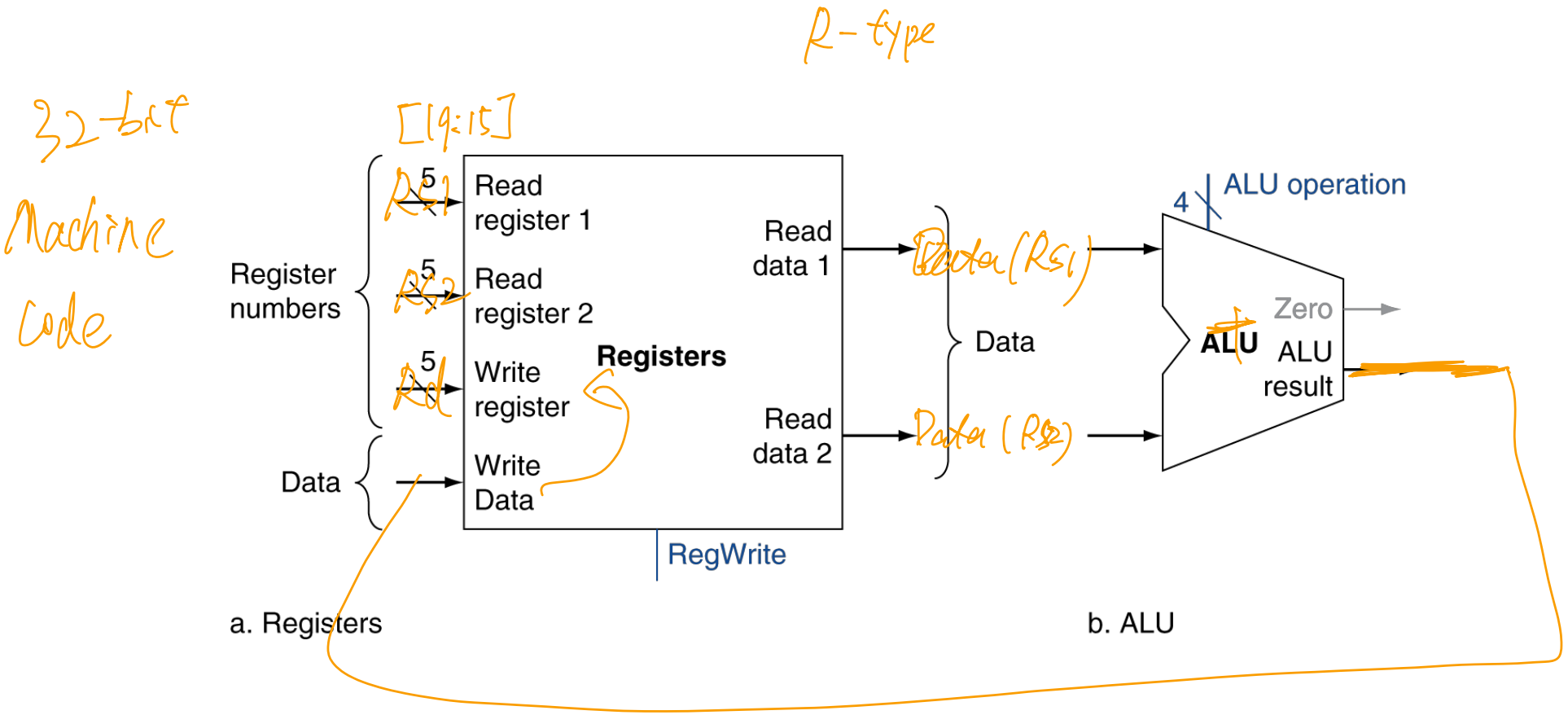
Instruction Fetch



If we remove the instruction memory, did we see similar circuit before?

R-Type Format Instructions

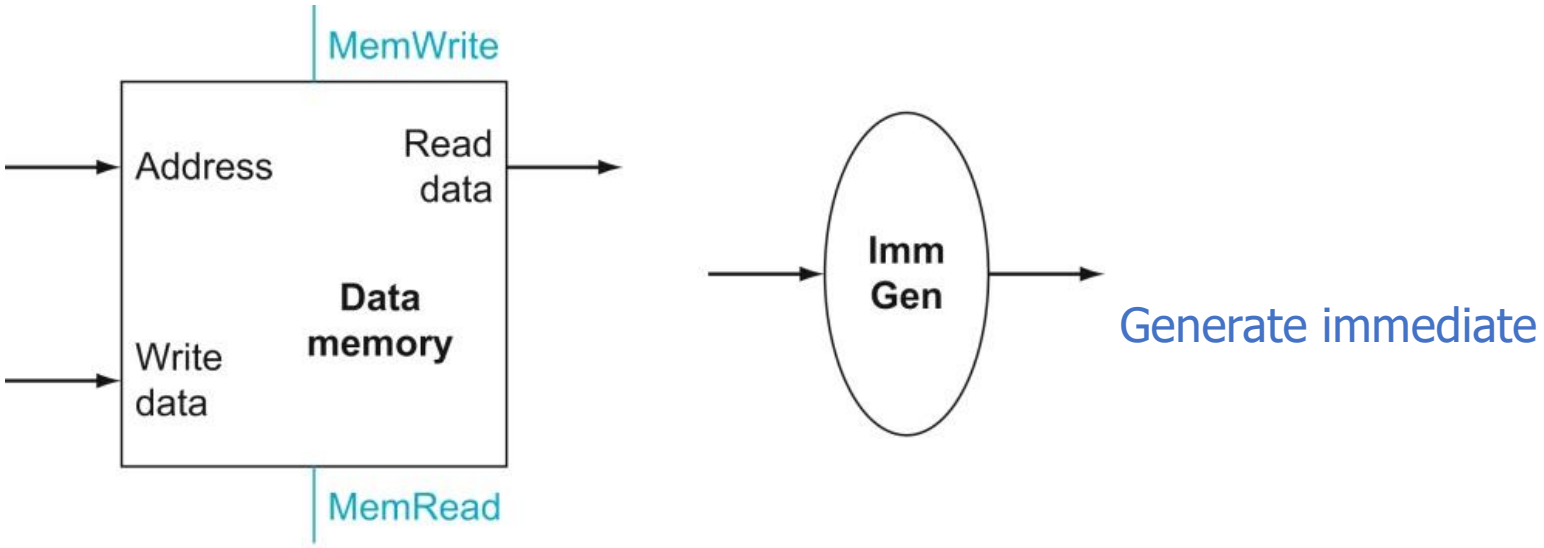
- Read two register operands
- Perform arithmetic/logical operation
- Write register result



Load/Store Instructions

- Read register operands
  - One or two registers are needed?
- Use ALU to calculate address using sign-extended 12-bit offset

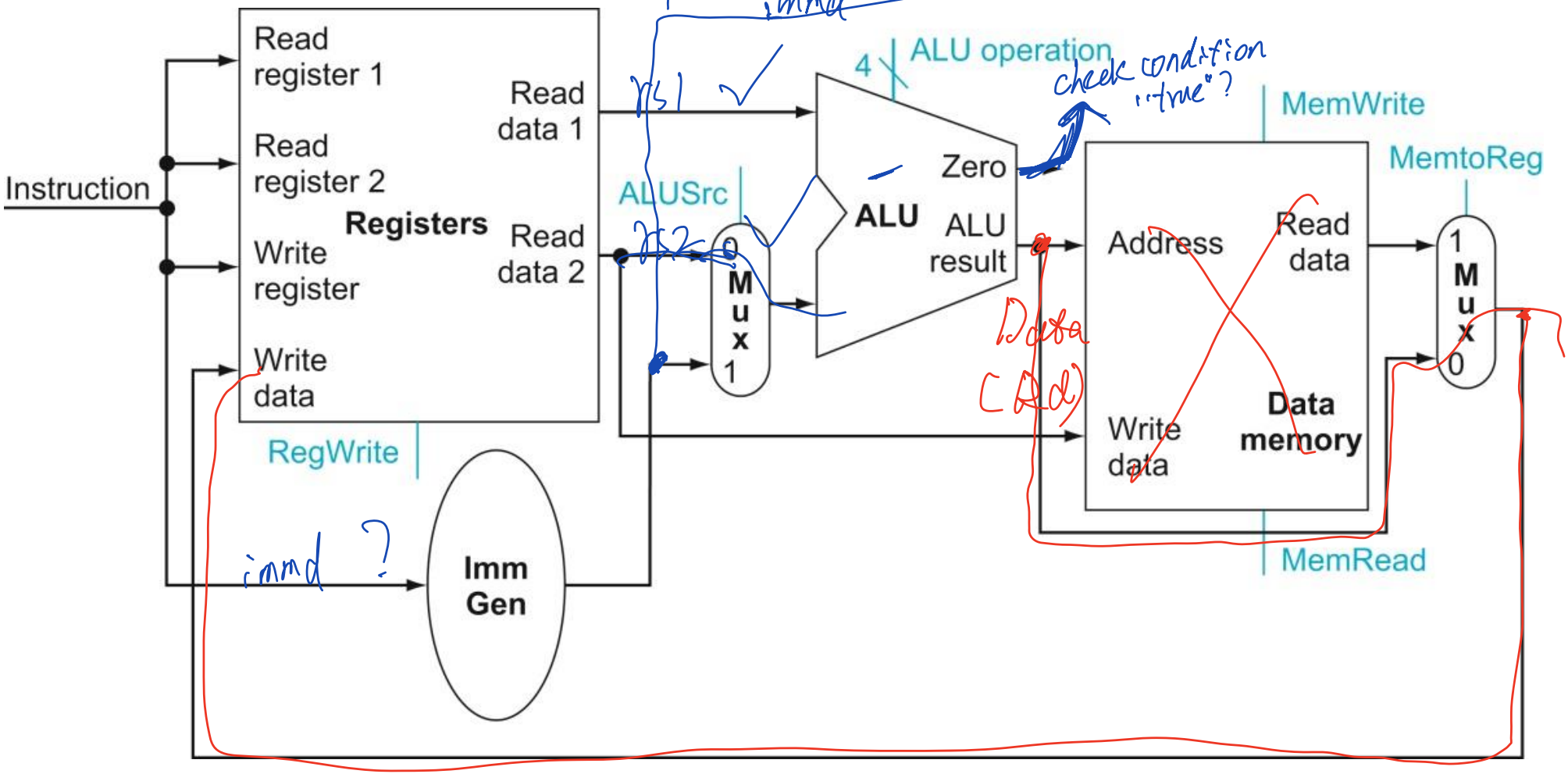
- Load: Read memory and update register
- Store: Write register value to memory



a. Data memory unit

b. Immediate generation unit

R-Type/Load/Store Datapath



Branch Instructions

- Read two register operands
- Compare them with ALU
  - Do subtraction and check Zero
- Calculate target address: PC + immediate
  - Additional adder (because ALU is used for comparison)

Branch Instructions

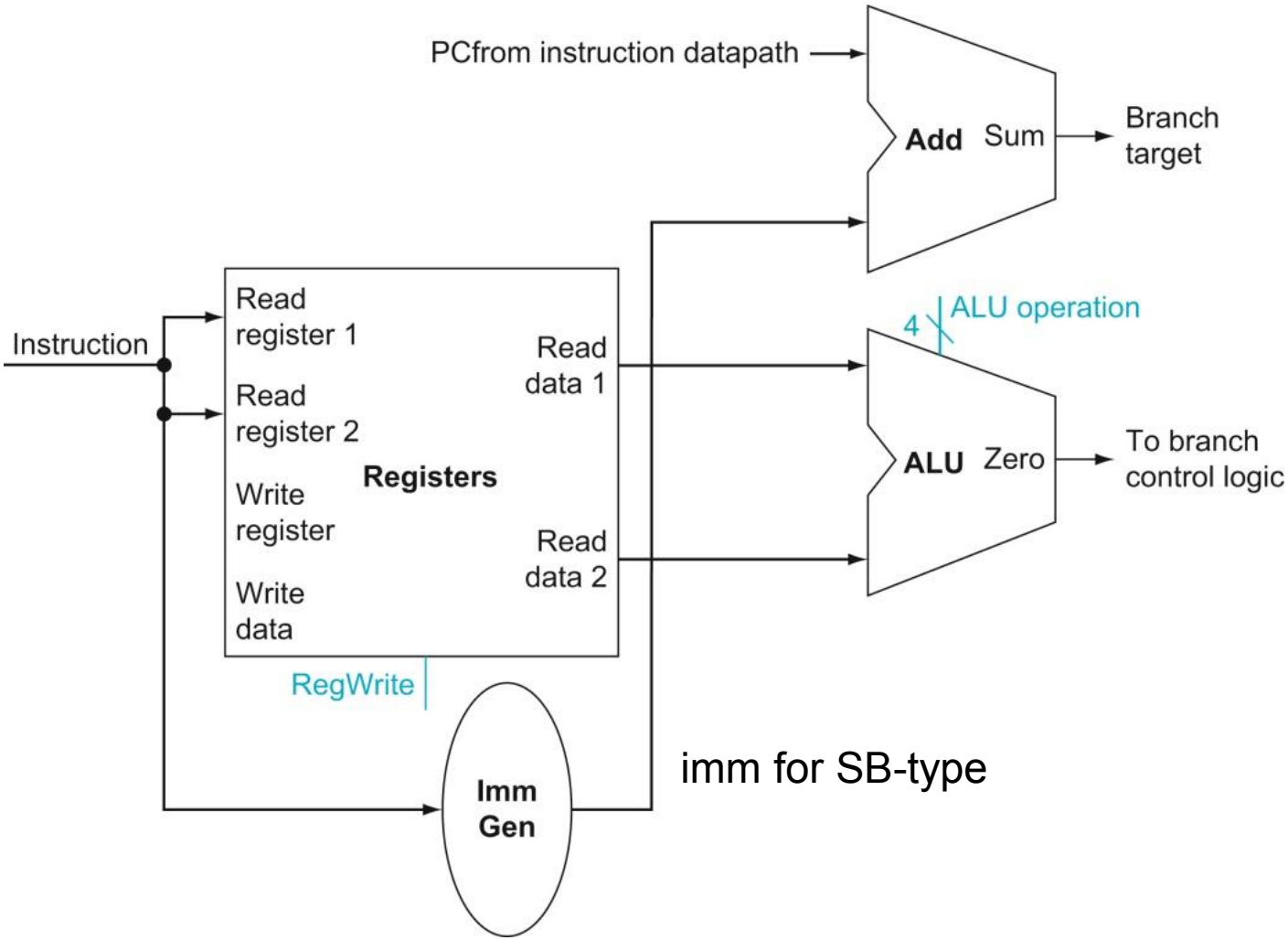


Figure 4.9

Full Datapath (w/o control)

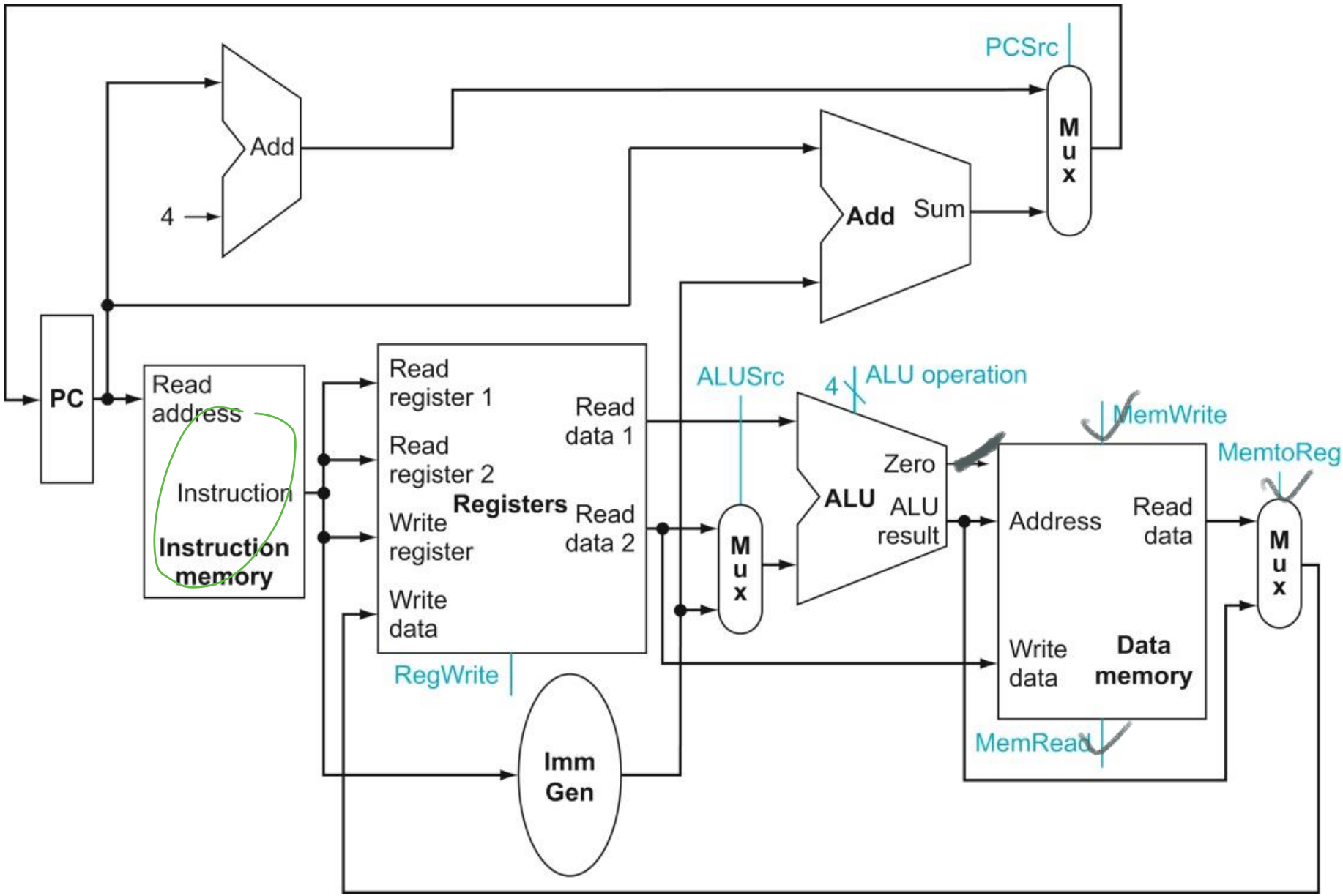


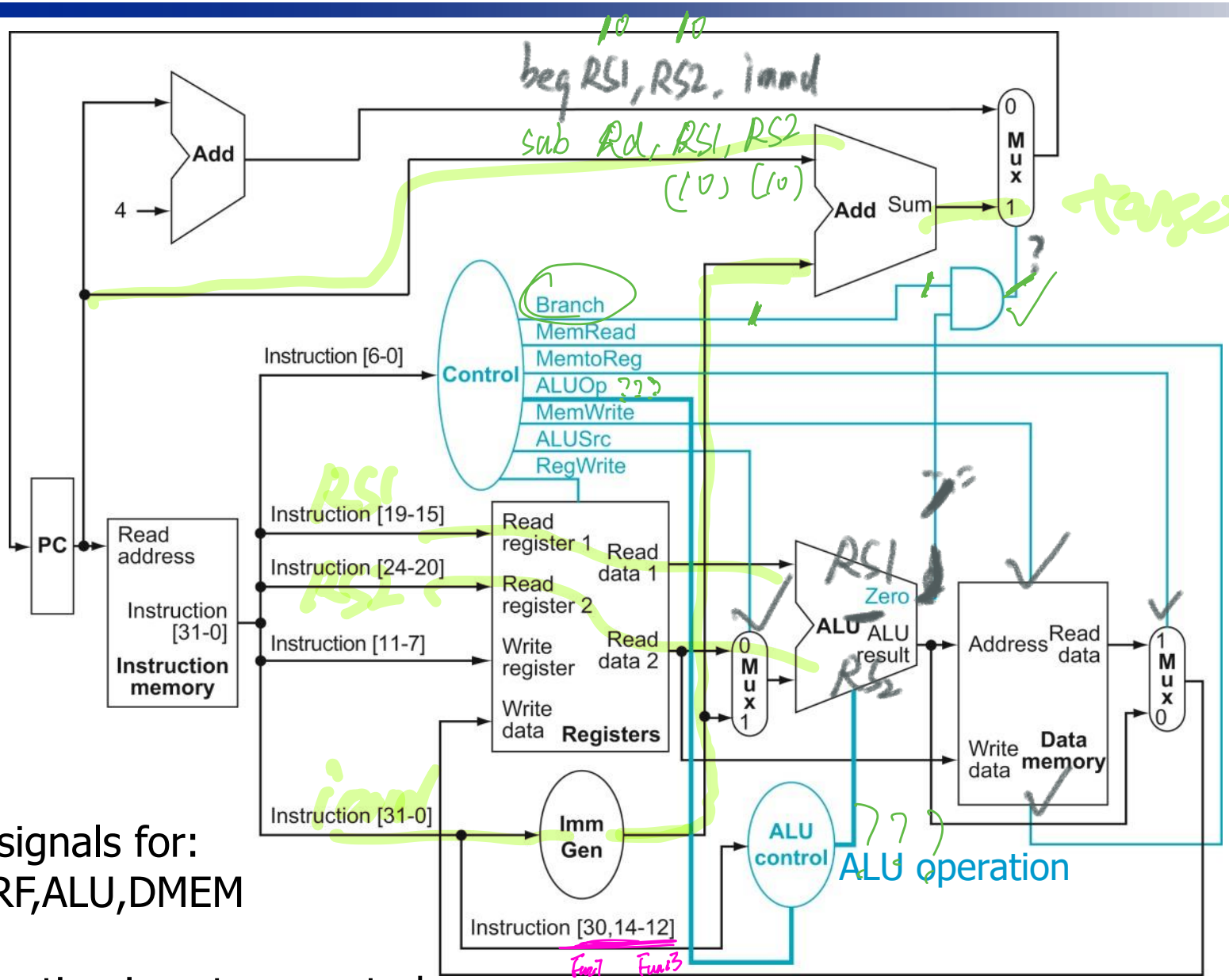
Figure 4.11

Datapath Summary

- First-cut 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 multiplexors where alternate data sources are used for different instructions

We still need to generate the colored signals

Datapath with control (preview)



Control signals for:
MUXes,RF,ALU,DMEM

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

ALU and its functions

- ALU performs functions specified by 4-bit ALU operation
- Design a combinational circuit to generate ALU operation
  - We call the module ALU Control

ALU operation	Function
0000	and
0001	or
0010	add
0110	subtract

This is the table we had after the ALU design.
Whoever design ALU provides the table.

Design ALU Control

- For each instruction,
  - What operations do we want ALU to perform?
  - How do we identify it from fields/bits in the machine code?

Instruction	Operation to perform on ALU	Fields in machine code
add (R)	+	opcode, Funct3, Funct7
and (R)	and	opcode, Funct3, Funct7
or (R)	or	opcode, Funct3, Funct7
lw (I)	+	opcode
sw (S)	+	opcode
beq (SB)	−	opcode

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 Input and Output

- If ALUOp is 10 (R-type), check more bits in funct3 and funct7

If we do not check funct7, which operation cannot be done? add, subtract

Instruction opcode	ALUOp	Operation	funct7	funct3	ALU function	ALU operation
lw	00	load word	xxx xxxx	xxx	add	0010
sw	00	store word	xxx xxxx	xxx	add	0010
beq	01	branch if equal	xxx xxxx	xxx	subtract	0110
R-type	10	add	000 0000	000	add	0010
		subtract	010 0000	000	subtract	0110
		and	000 0000	111	and	0000
		or	000 0000	110	or	0001

Implementation of ALU Control (Hardware)

We only need four bits from funct fields (bits 30, 14, 13, and 12)

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	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

Input

Output

ALUOp != 0b11

Need 1 bit in funct7 and
3 bits in funct3

I[30, 14, 13, 12] ⟷ I[30, 14-12]

Write the logic expression for each bit in Operation

For each bit, write a product term for each row where the operation bit is 1, and then OR the product terms together. Then simplify the expression.

Sum of product

Implementation of ALU Control (Software)

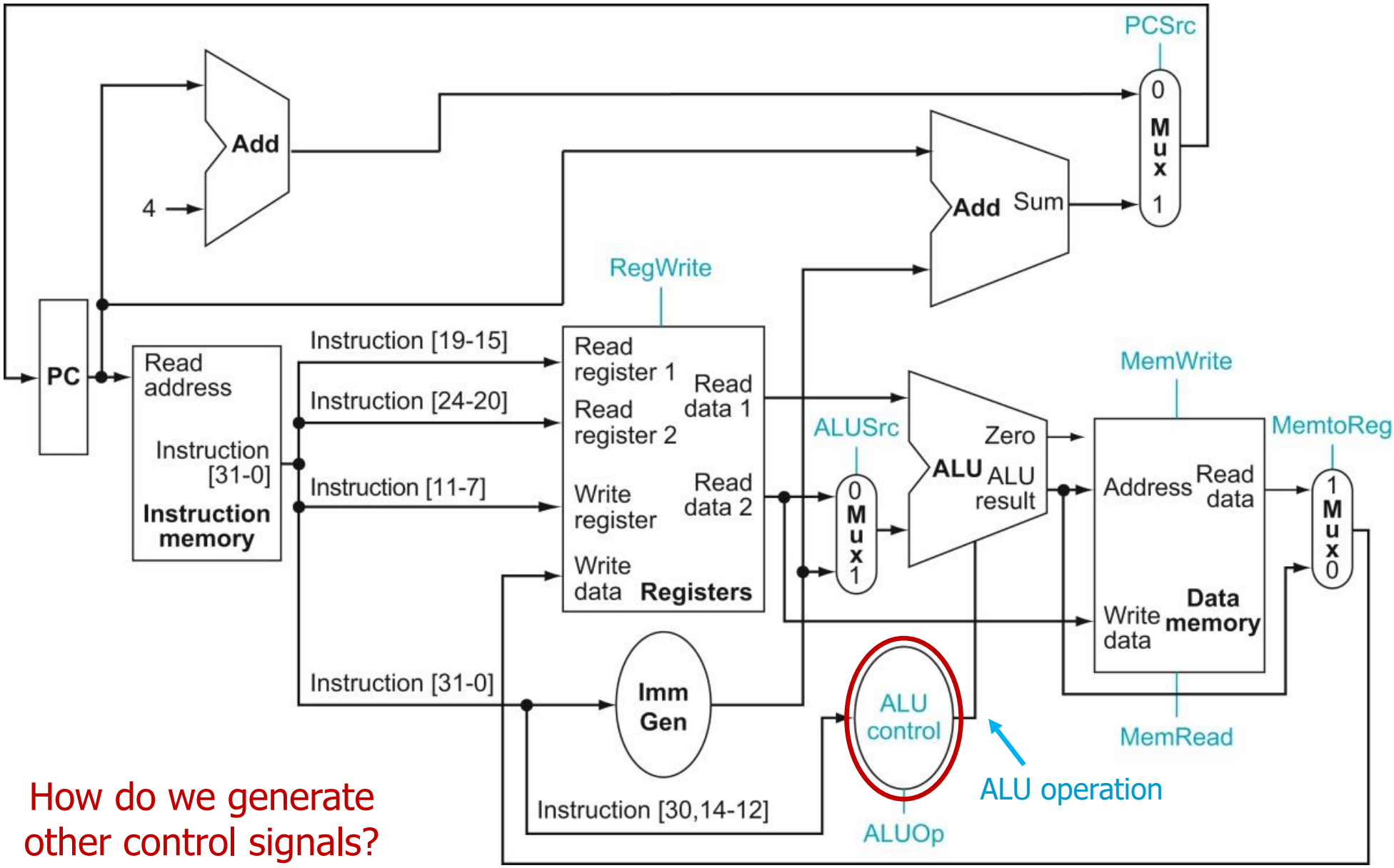
- In HDL/simulation, we can just describe the behavior

For example, in MyHDL,

```
if ALUOp == 0b00:
    ALUOperation.next = 0b0010
elif ALUOp == 0b01:
    ALUOperation.next = 0b0110
# more cases
```

Software can also generate each bit using logic expressions

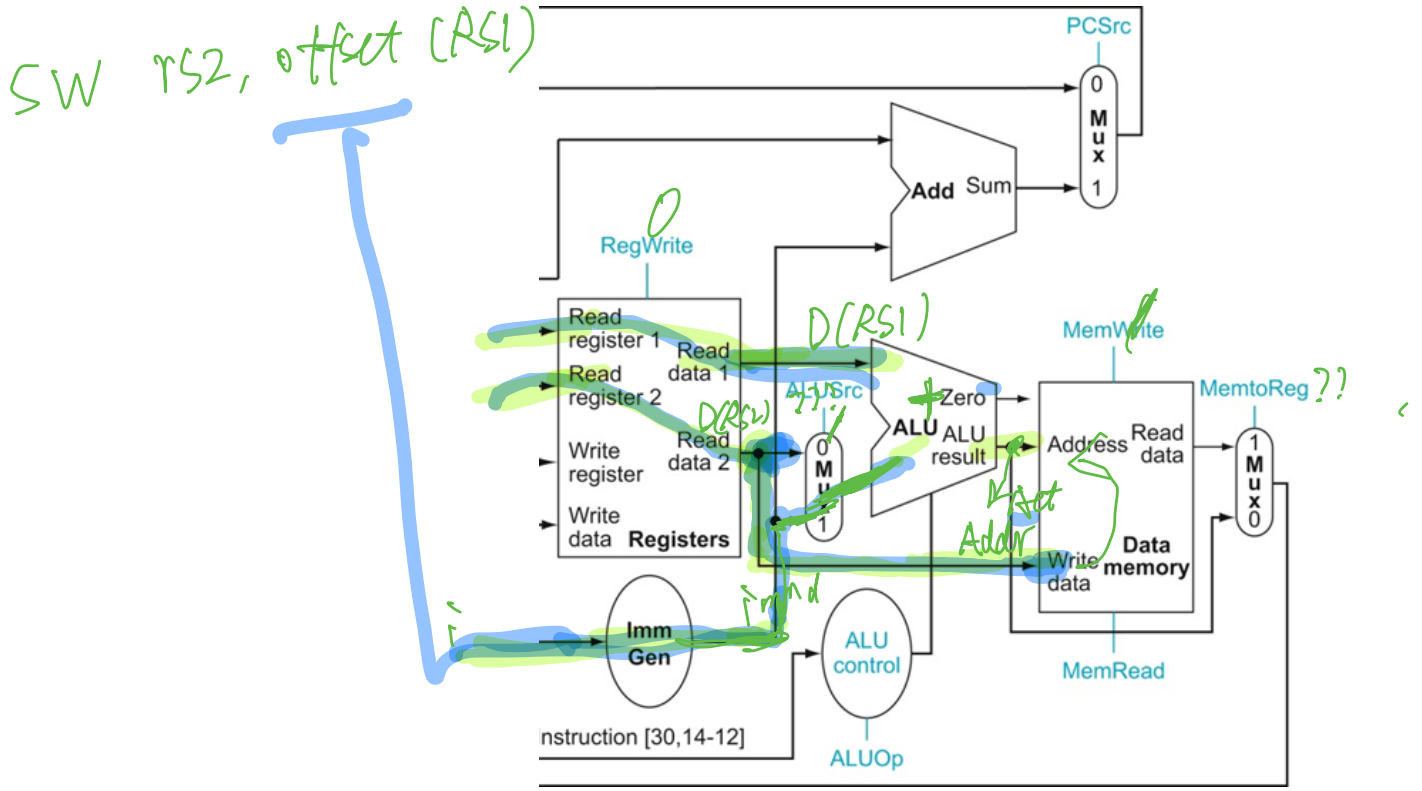
Datapath with ALU control



How do we generate other control signals?

Generating control signals from opcode

Inst.	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op
R-type							10
lw							00
sw	1	X	0	0	1	0	00
beq							01



Generating control signals from opcode

Inst.	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op
R-type	0	0	1	0 X wrong	0 X wrong	0	10
lw	1	1	1	1	0	0	00
sw	1	X	0	0	1	0	00
beq	0	X	0	0	0	1	01

X means don't care. It can be 0 or 1. Designers can pick a value to optimize the circuit.

MemRead and MemWrite should be always set, either 0 or 1.

Figure out the values yourself from the diagram (not just memorizing them).

Example: generating control signals from opcode

Inst.	Opcode	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch
R-type	011 0011	0	0	1	0	0	0
lw	000 0011	1	1	1	1	0	0
sw	010 0011	1	X	0	0	1	0
beq	110 0011	0	X	0	0	0	1

Op6, Op5, …, Op0 are bit 6, bit 5, … , and bit 0 in the opcode.

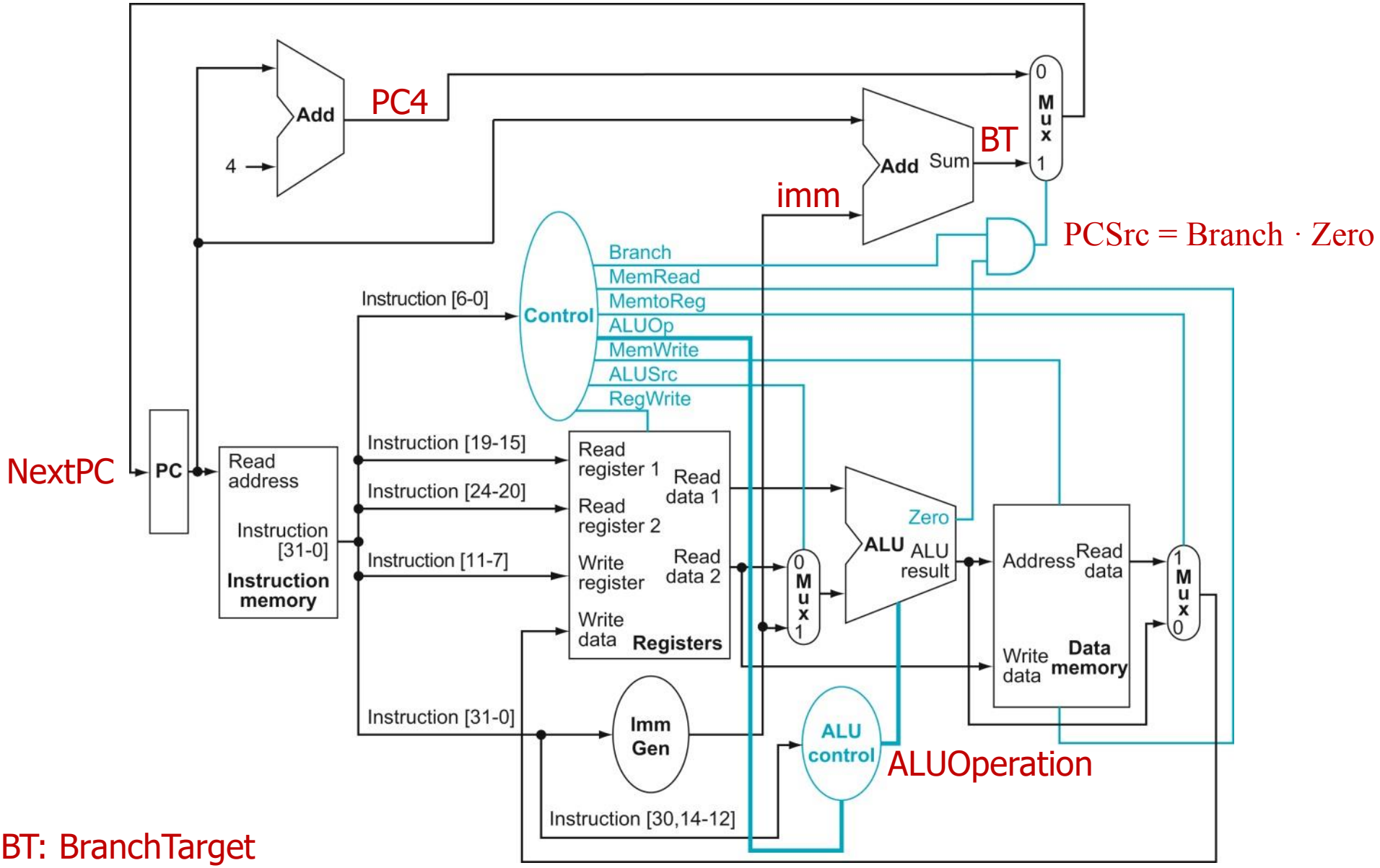
$$\text{RType} = \overline{\text{Op6}} \cdot \text{Op5} \cdot \text{Op4}$$
$$\text{Load} = \overline{\text{Op6}} \cdot \overline{\text{Op5}} \cdot \overline{\text{Op4}}$$
$$\text{Store} = \overline{\text{Op6}} \cdot \text{Op5} \cdot \overline{\text{Op4}}$$
$$\text{Branch} = \text{Op6} \cdot \text{Op5} \cdot \overline{\text{Op4}}$$

Opcode[3:0] are the same

$$\text{ALUSrc} = \text{Load} + \text{Store}$$
$$\text{RegWrite} = \text{RType} + \text{Load}$$

Study the truth table in Figure 4.26

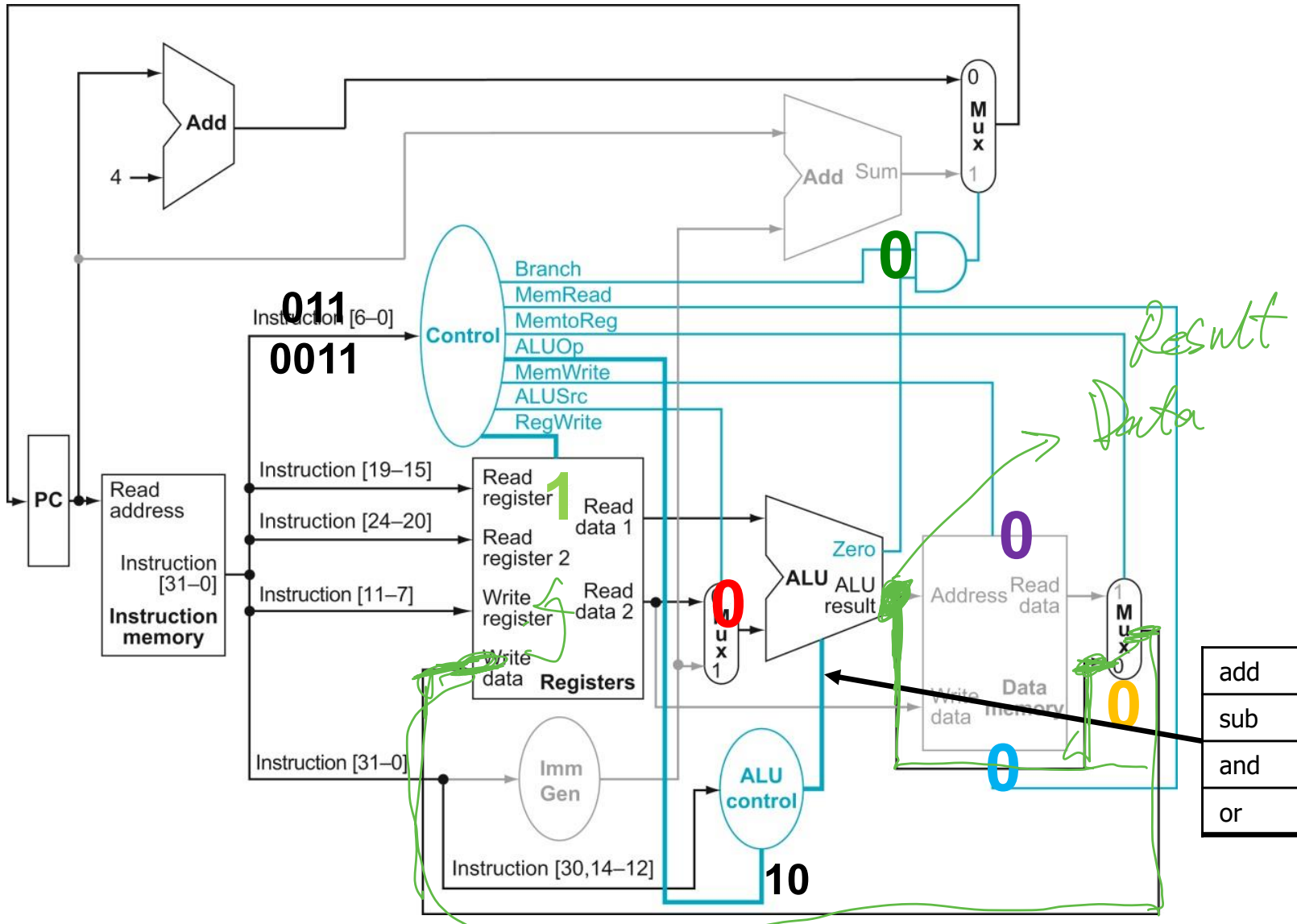
Datapath with control



BT: BranchTarget
imm: immediate

Operation of Datapath: R-Type Instruction

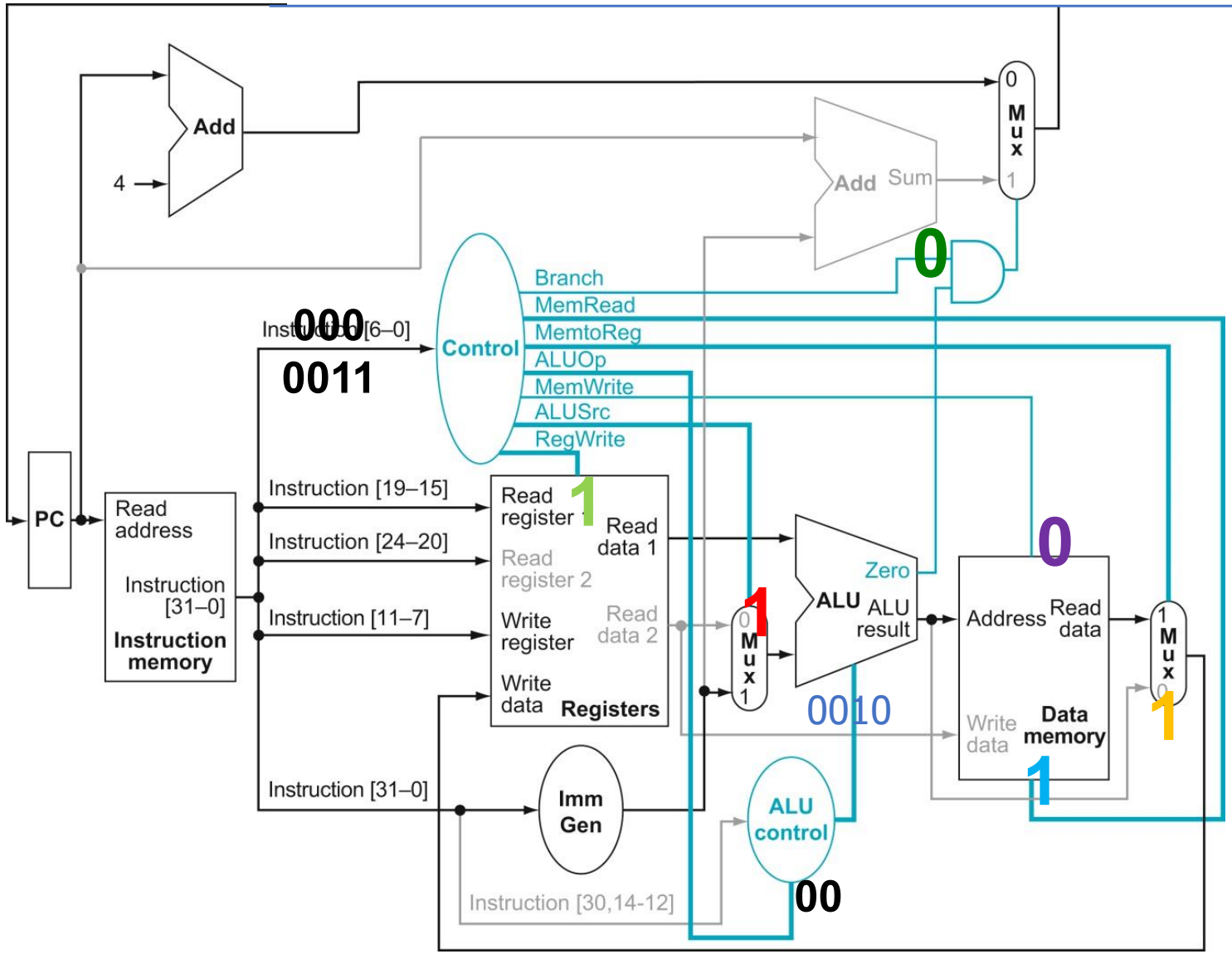
Inst.	Opcode	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch
R-type	011 0011	0	0	1	0	0	0



add	0010
sub	0110
and	0000
or	0001

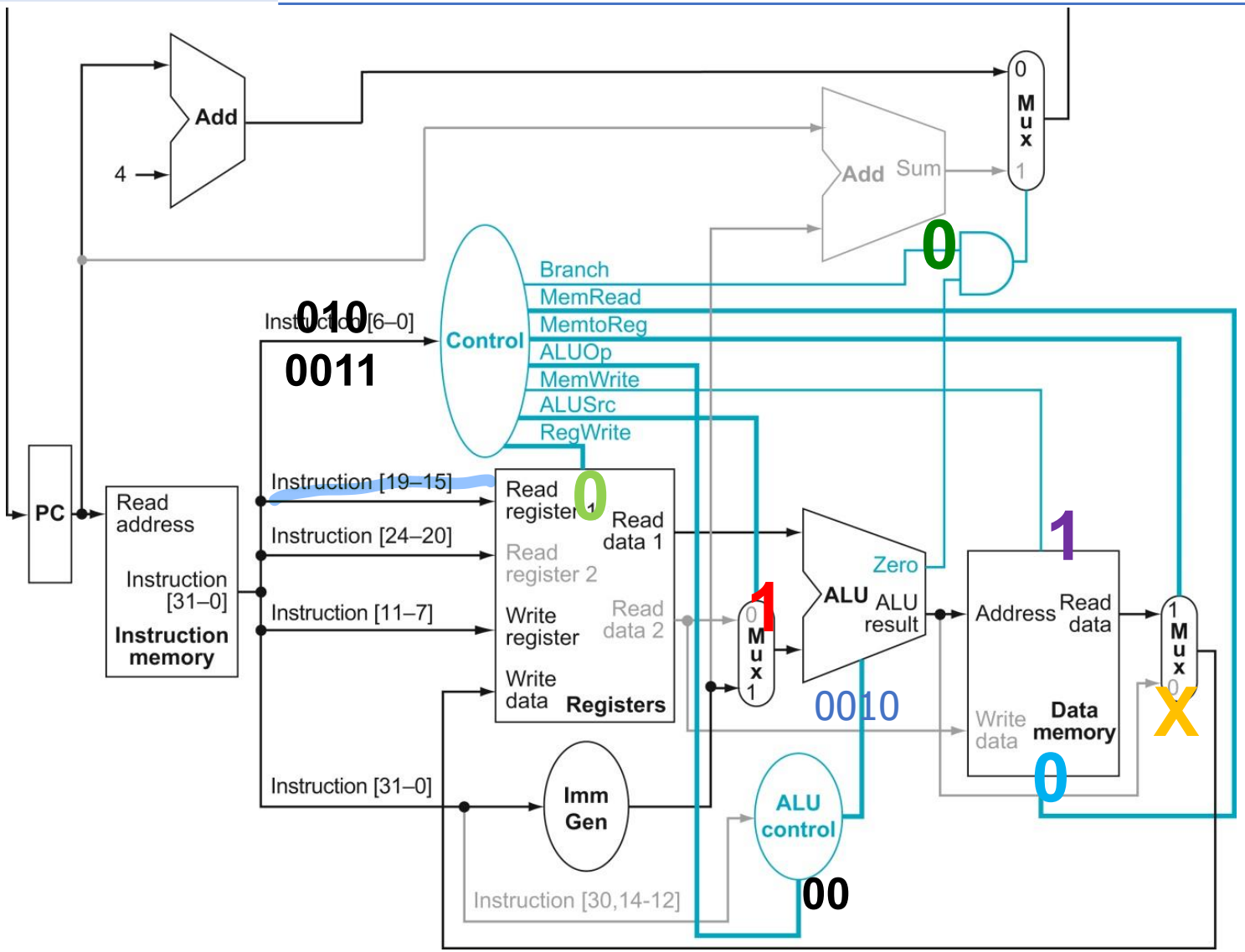
Operation of Datapath: load

Inst.	Opcode	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch
lw	000 0011	1	1	1	1	0	0



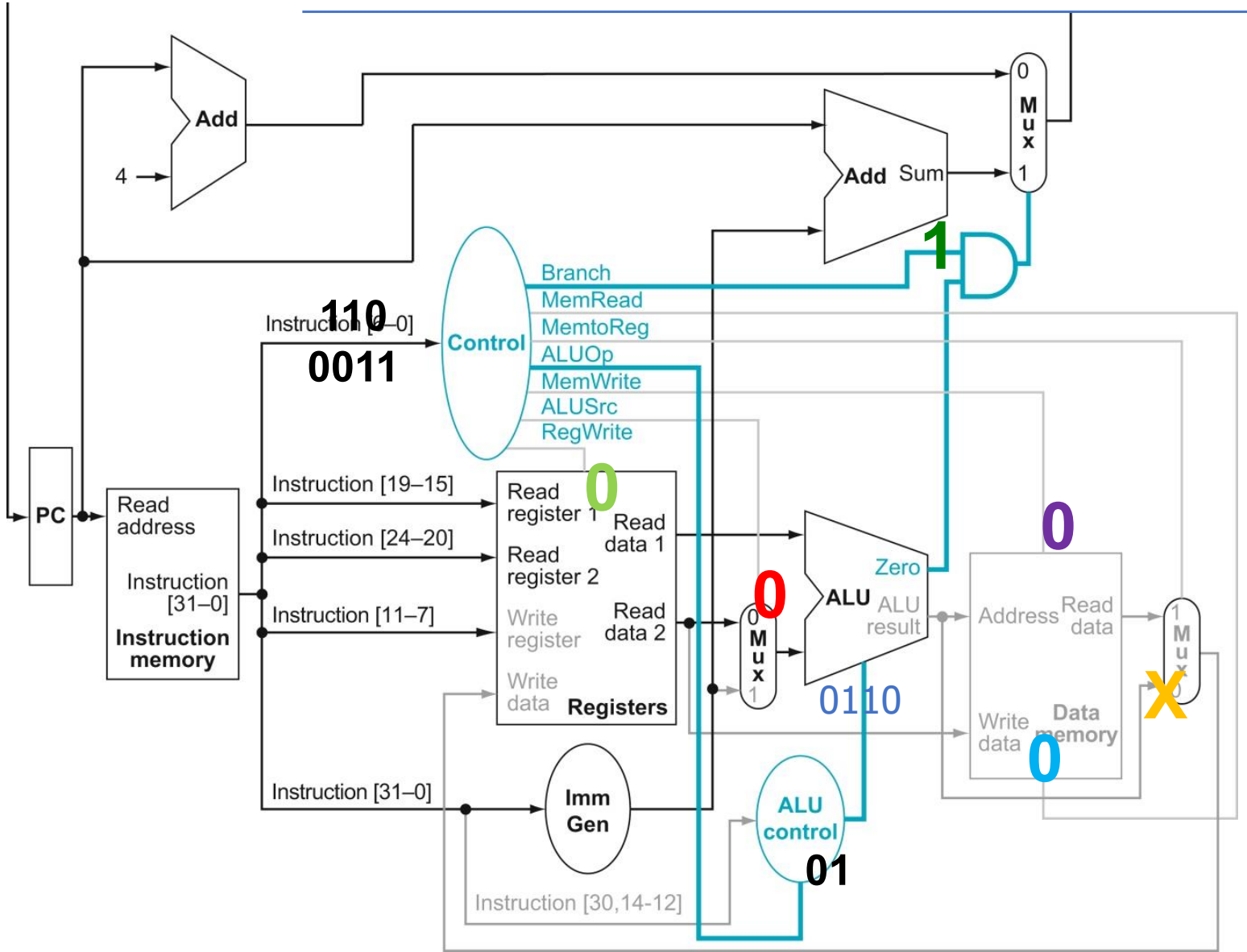
Operation of Datapath: store

Inst.	Opcode	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch
sw	010 0011	1	X	0	0	1	0

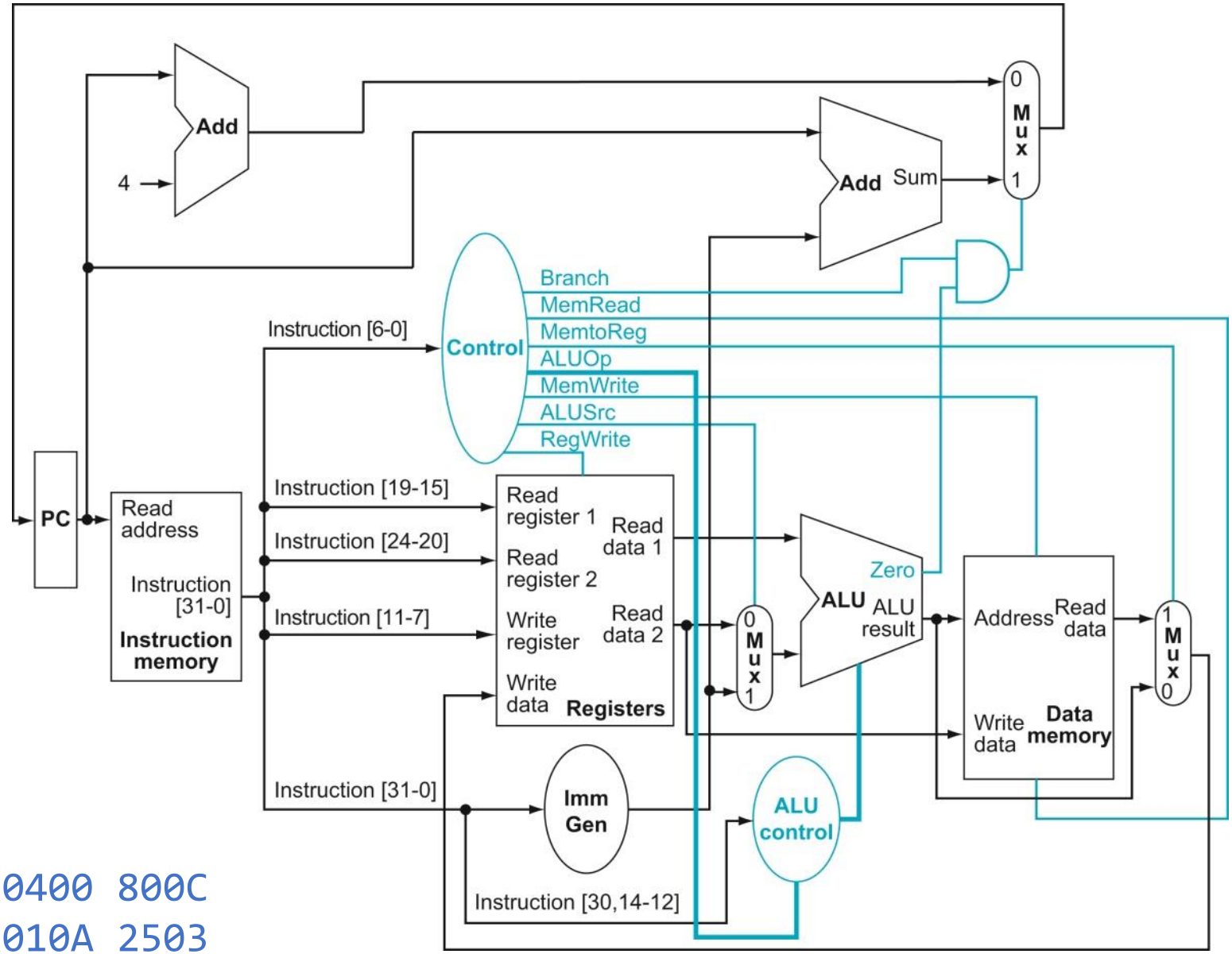


Operation of Datapath: beq

Inst.	Opcode	ALU Src	Memto Reg	Reg Write	Mem Read	Mem Write	Branch
beq	110 0011	0	X	0	0	0	1



Signal values




```
Addr: 0x0400 800C
Instr:0x010A 2503
lw x10, 16(x20)
```

Find the signal values

Start from PC and propagate signal values through the diagram

- rs1, rs2, rd =
- immediate =
- Control signals

Inst.	ALUSrc	Memto Reg	Reg Write	Mem Read	Mem Write	Branch	ALU Op
lw							

- PC4 =
- branch target address =
- PCSrc =
- next PC =

```
Addr: 0x0400 800C
Instr:0x010A 2503
lw x10, 16(x20)
```

Support more instructions

- How would you add more instructions?
  - What existing blocks are used?
  - Which new functional blocks is needed (if any)?
  - What new signals need to be added (if any)?

  `xor`
  `addi`
  `bne`
  `lui`

- Homework

  `jal`
  `jalr`

How fast can this processor run?

- Assume the following delays
  - 80 ps for decoding
  - 100 ps for register read or write
  - 200ps for ALU
  - 300ps for memory
  - Ignore other delays (e.g., propagation delay of registers and MUXes)

Instr	Instr fetch	Register read	ALU	Memory access	Register write	Total time
lw	300ps	100ps	200ps	300ps	100ps	1000ps
sw	300ps	100ps	200ps	300ps		900ps
R-type	300ps	100ps	200ps		100ps	700ps
beq	300ps	100ps	200ps			600ps

Why is single-cycle implementation not used today?

Execution time = Instruction Count × CPI × Cycle Time

Instruction Count is determined by ISA

CPI = 1

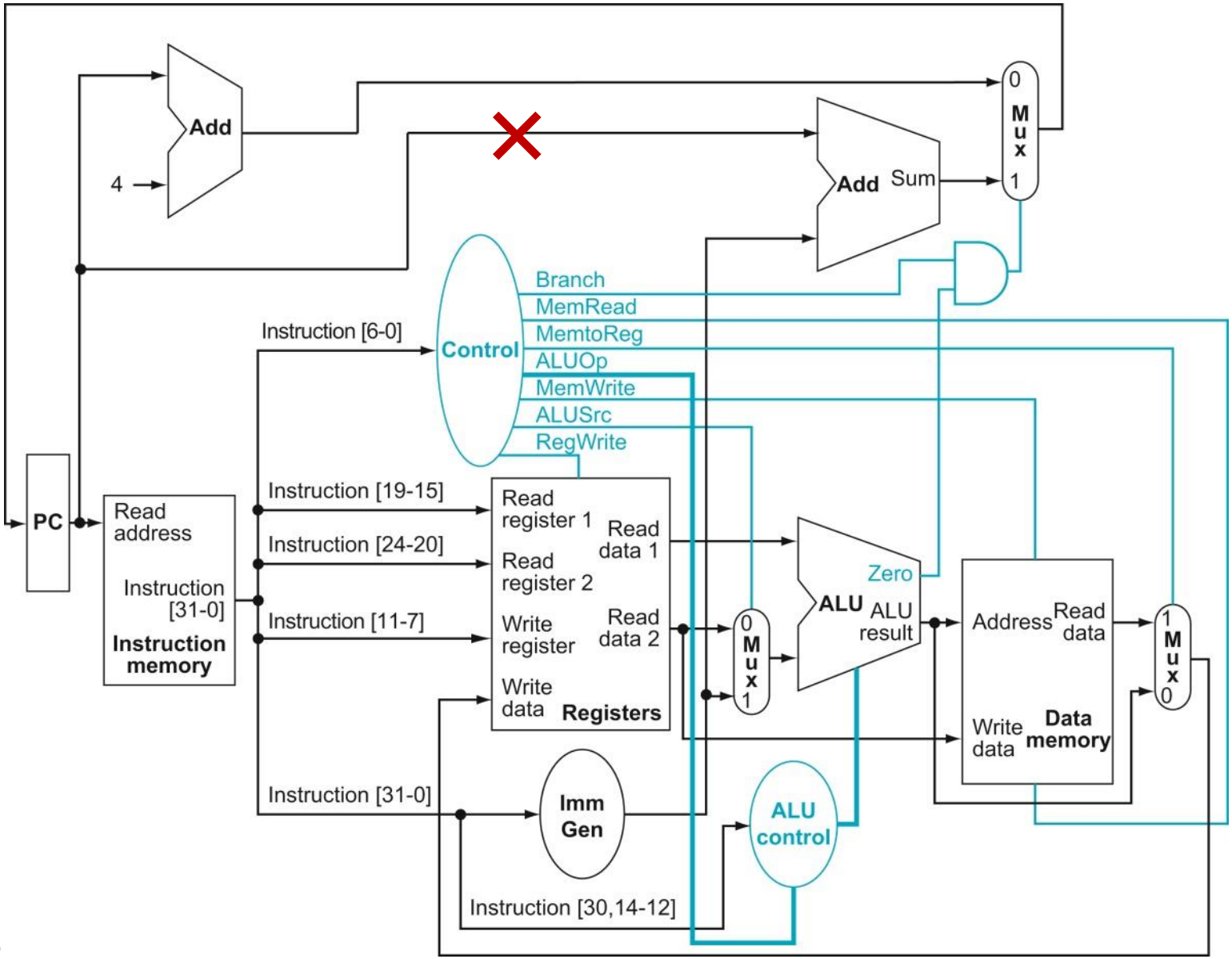
Cycle Time is decided by the slowest instruction (which one?)

Make common case fast!

Steps in Instruction Execution

- Use address in PC to fetch instruction from instruction memory
- Decode the instructions and read register file (RF)
- Arithmetic or logical operations
  - Use ALU to perform the operation
  - Set the correct signals for updating the destination register
- Load/Store
  - Use ALU to calculate memory address
  - Access data memory for load/store
  - Set the correct signals for updating the destination register, for load
- Branches
  - Use ALU to compare
  - Calculate branch target address, using a separate adder
  - Select proper address for the next instruction

Which instruction will be affected?



A. add
B. beq
C. load
D. store
E. More than 1

Description of control signals

The main control unit generates control signals from opcode

The table can be easily seen from the diagram.

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.
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.
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.

ALU Control generating ALU operation[1]

```
Operation[1]
= (~ALUOp1 & ~ALUOp0) | ALUOp0
        | (ALUOp1 & ~I[30] & ~I[14] & ~I[13] & ~I[12])
        | (ALUOp1 &  I[30] & ~I[14] & ~I[13] & ~I[12])
= ~ALUOp1 | ALUOp0 | (ALUOp1 & ~I[14] & ~I[13] & ~I[12])
= … = ~ALUOp1 | ALUOp0 | ~I[14]
```

ALUOp1	ALUOp0	I[30]	I[14]	I[13]	I[12]	Operation
0	0	X	X	X	X	0010
X	1	X	X	X	X	0110
1	X	0	0	0	0	0010
1	X	1	0	0	0	0110
1	X	0	1	1	1	0000
1	X	0	1	1	0	0001

ALU Control generating ALU operation [2, 0]

```
Operation[2]
= (ALUOp0) | (ALUOp1 & I[30] & ~I[14] & ~I[13] & ~I[12])
= …                 # continue to simplify

Operation[0]
= ALUOp1 & ~I[30] & I[14] & I[13] & ~I[12]
= …
```

ALUOp1	ALUOp0	I[30]	I[14]	I[13]	I[12]	Operation
0	0	X	X	X	X	0010
X	1	X	X	X	X	0110
1	X	0	0	0	0	0010
1	X	1	0	0	0	0110
1	X	0	1	1	1	0000
1	X	0	1	1	0	0001