Lecture 4 – Microarchitecture-Design

(ECE M116C- CS M151B) Computer Architecture Systems

Nader Sehatbakhsh
Department of Electrical and Computer Engineering
University of California, Los Angeles

HW1

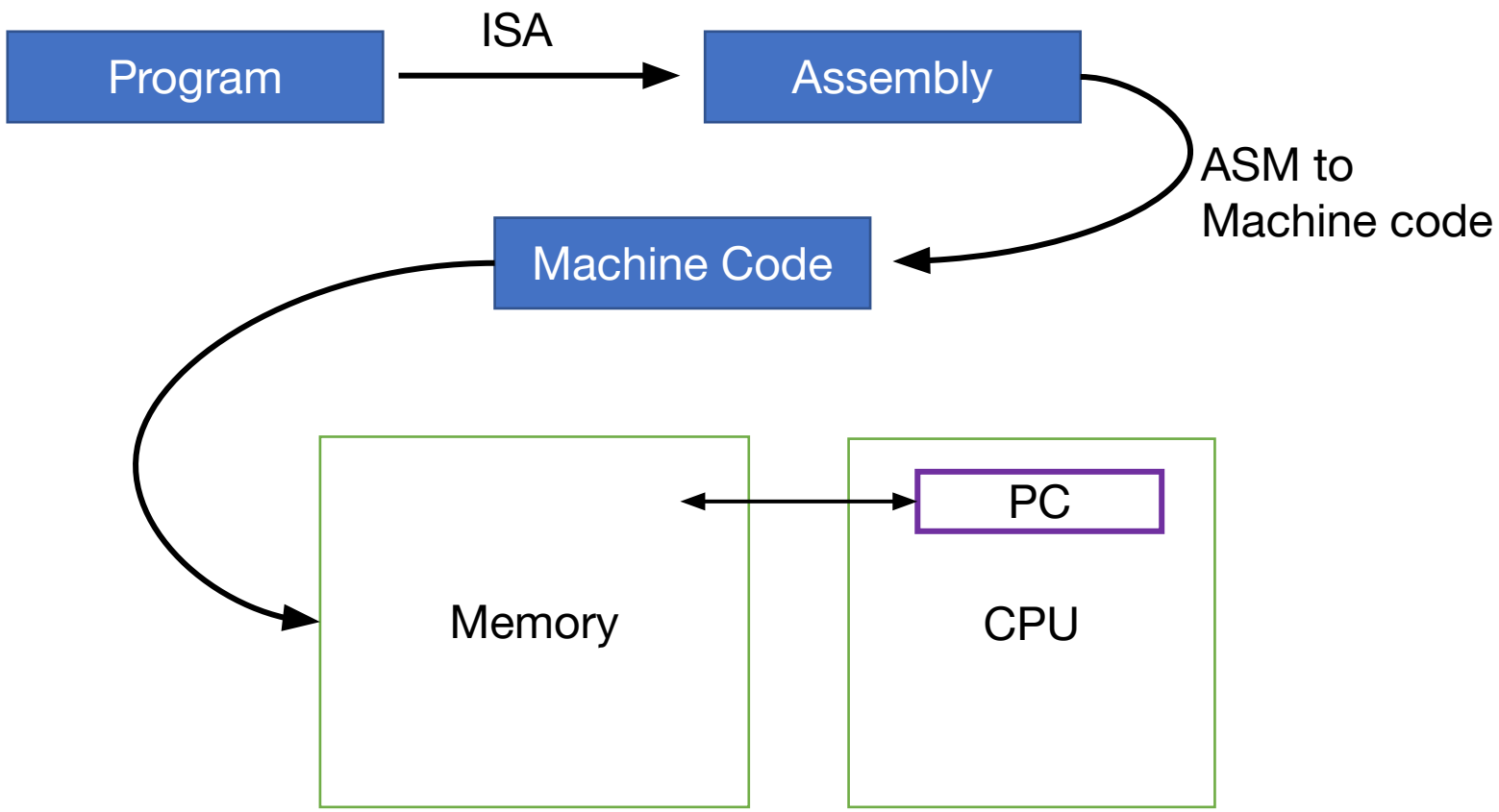
RV32I Base Instruction Set (MIT 6.004 subset)

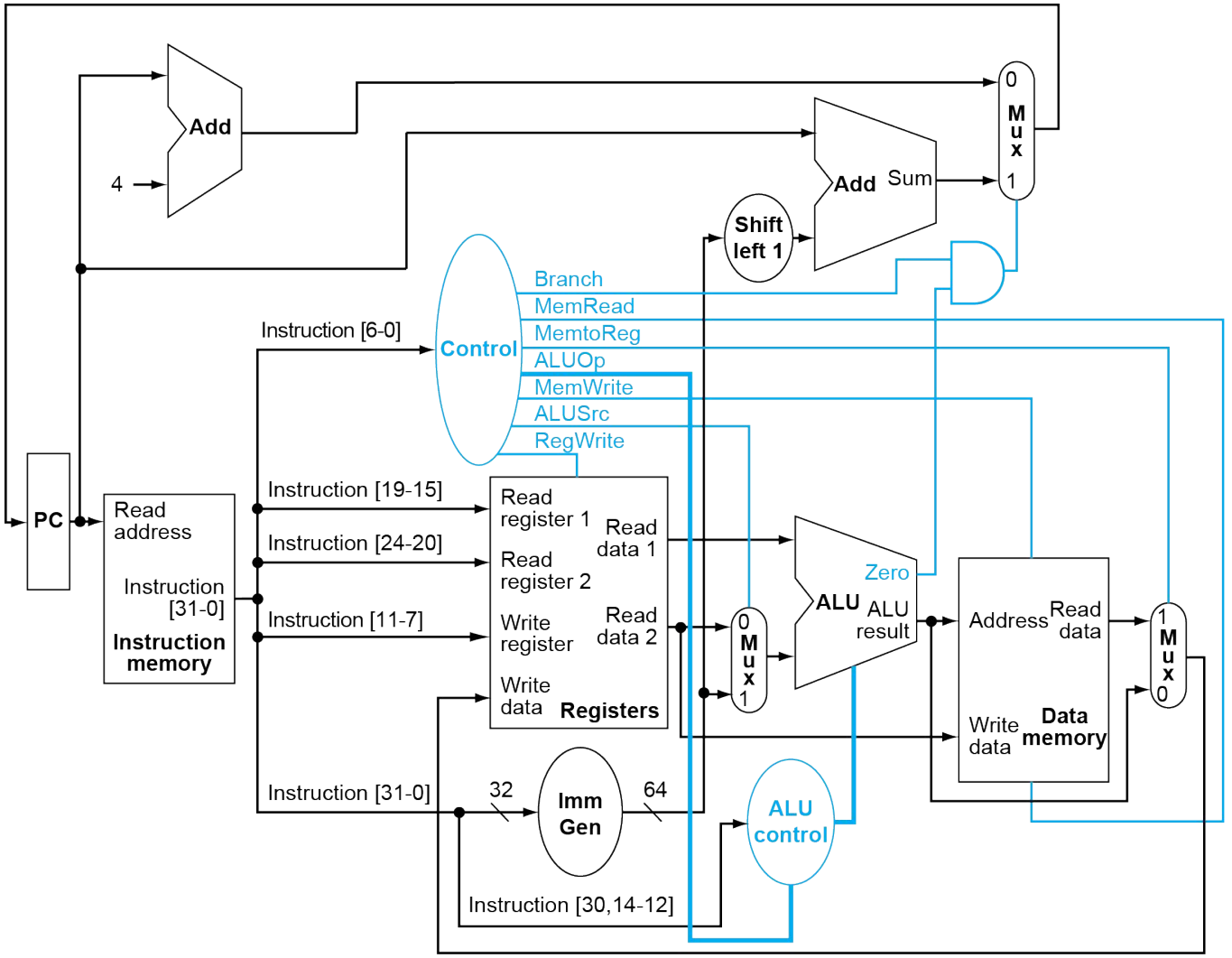
imm[31:12]				rd	0110111	LUI
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

Executing Instruction on HW

(von Neuman model)





*Images were taken from Hennessy Patterson Book [1].

Datapath + Controller

Lifecycle of an instruction

Lifecycle of an instruction

1- Instruction is read (fetch) from the (instruction) memory

Lifecycle of an instruction

1- Instruction is read (fetch) from the (instruction) memory

2- Operands should be loaded
- RegFile, (data) Memory, Imm.
  - Need register number/ address for each operand.

Lifecycle of an instruction

1- Instruction is read (fetch) from the (instruction) memory

2- Operands should be loaded
- RegFile, (data) Memory, Imm.
  - Need register number/ address for each operand.

3- Operation should be executed
- Arithmetic (which type?), data movement, control-flow.

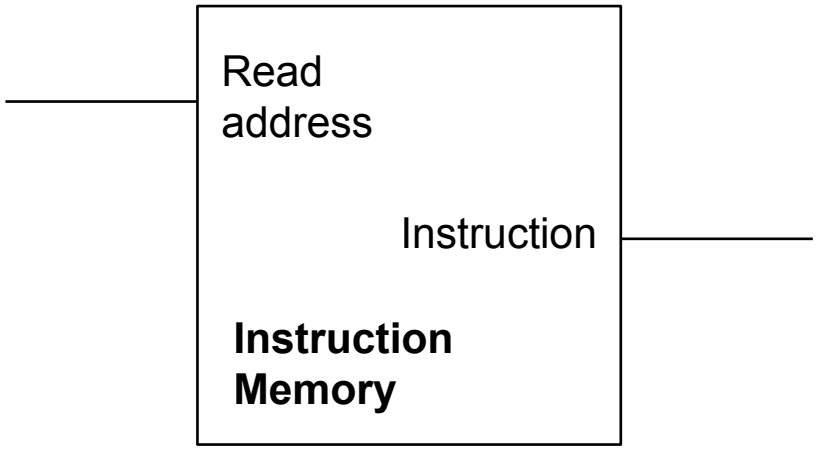
Lifecycle of an instruction

1- Instruction is read (fetch) from the (instruction) memory
2- Operands should be loaded
   - RegFile, (data) Memory, Imm.
     - Need register number/ address for each operand.
3- Operation should be executed
   - Arithmetic (which type?), data movement, control-flow
4- Results should be stored
   - RegFile or memory?

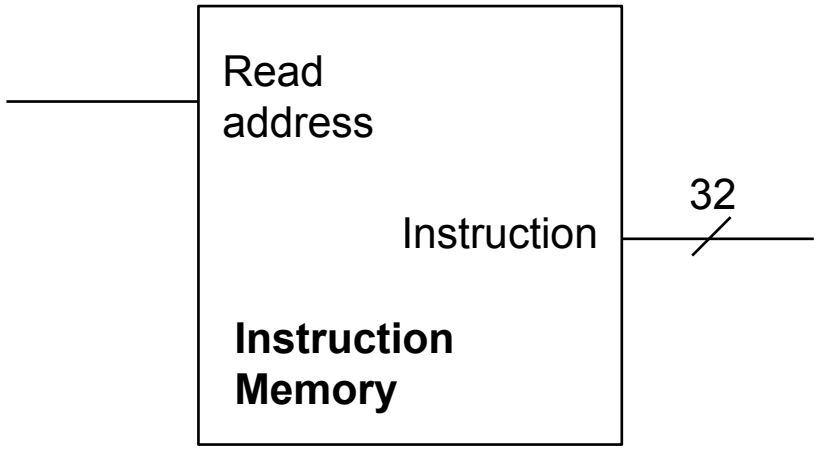
Lifecycle of an instruction

1- Instruction is read (fetch) from the (instruction) memory
2- Operands should be loaded
   - RegFile, (data) Memory, Imm.
     - Need register number/ address for each operand.
3- Operation should be executed
   - Arithmetic (which type?), data movement, control-flow.
4- Results should be stored
   - RegFile or memory?
5- PC should be updated
   - Sequential, jump, or branch?

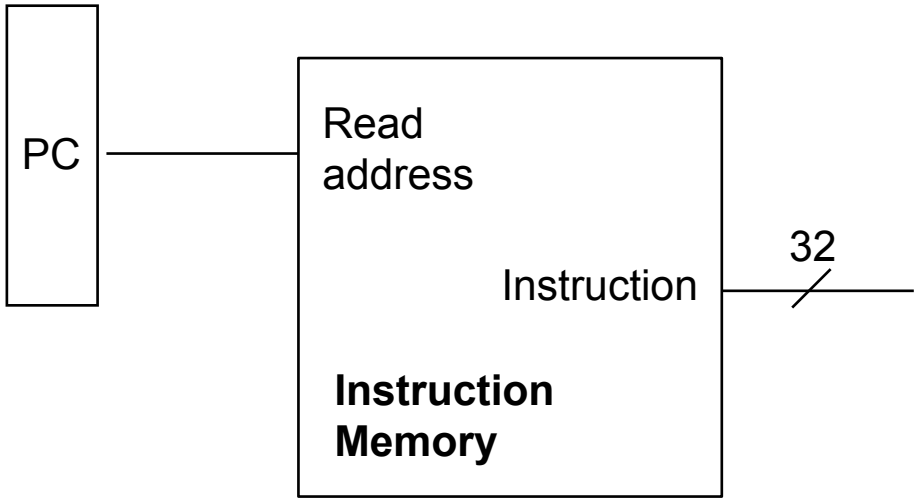
Instruction Fetch



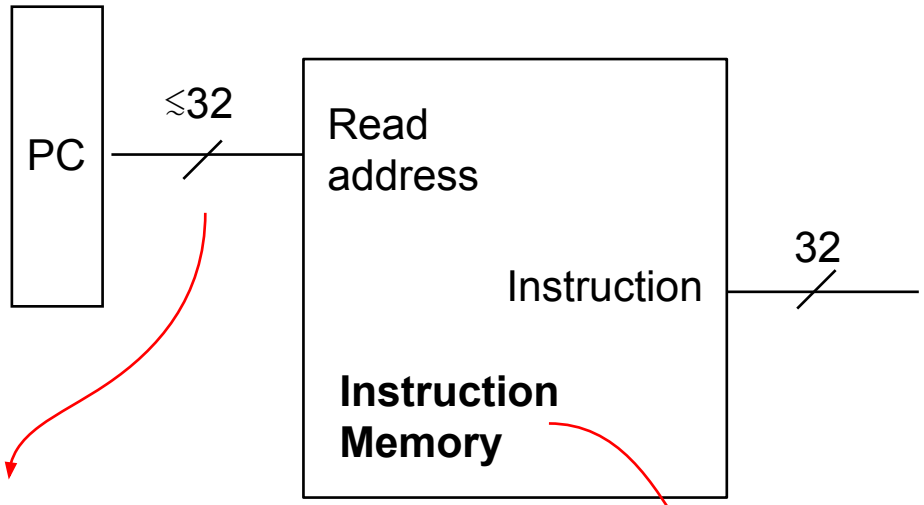
Instruction Fetch



Instruction Fetch



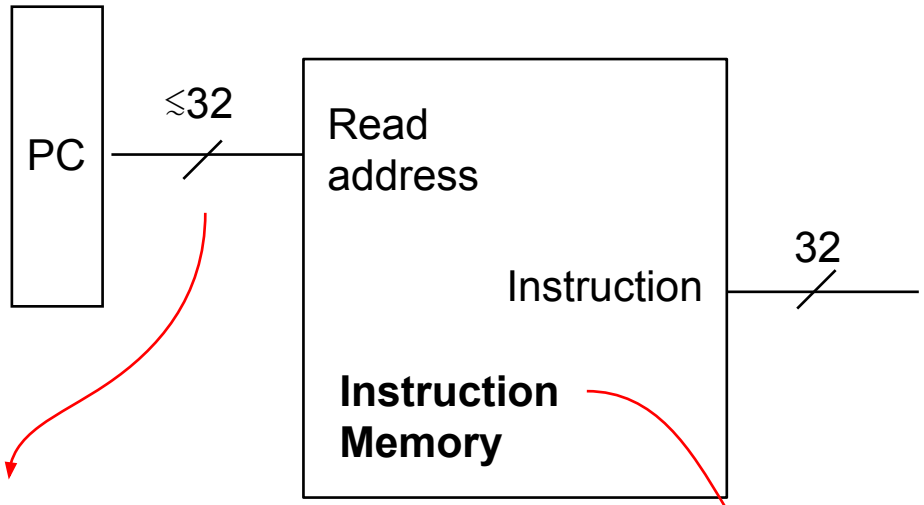
Instruction Fetch



We may not need ***all*** bits of PC for the address.

Memories are large, so we need to use SRAM/DRAM

Instruction Fetch



We may not need ***all*** bits of PC for the address.

Byte vs. word (why PC+4?)

Lifecycle of an instruction

1- Instruction is read (fetch) from the (instruction) memory

2- Operands should be loaded
- RegFile, (data) Memory, Imm.
  - Need register number/ address for each operand.

3- Operation should be executed
- Arithmetic (which type?), data movement, control-flow

4- Results should be stored
- RegFile or memory?

5- PC should be updated
- Sequential, jump, or branch?

Lifecycle of an instruction

1- Instruction is read (fetch) from the (instruction) memory

2- Operands should be loaded
- RegFile, (data) Memory, Imm.
  - Need register number/ address for each operand.

3- Operation should be executed
- Arithmetic (which type?), data movement, control-flow

4- Results should be stored
- RegFile or memory?

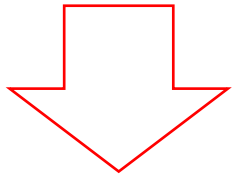
5- PC should be updated
- Sequential, jump, or branch?

Three types of operands.

Loading Operands

Immediate

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type



Read address

Instruction

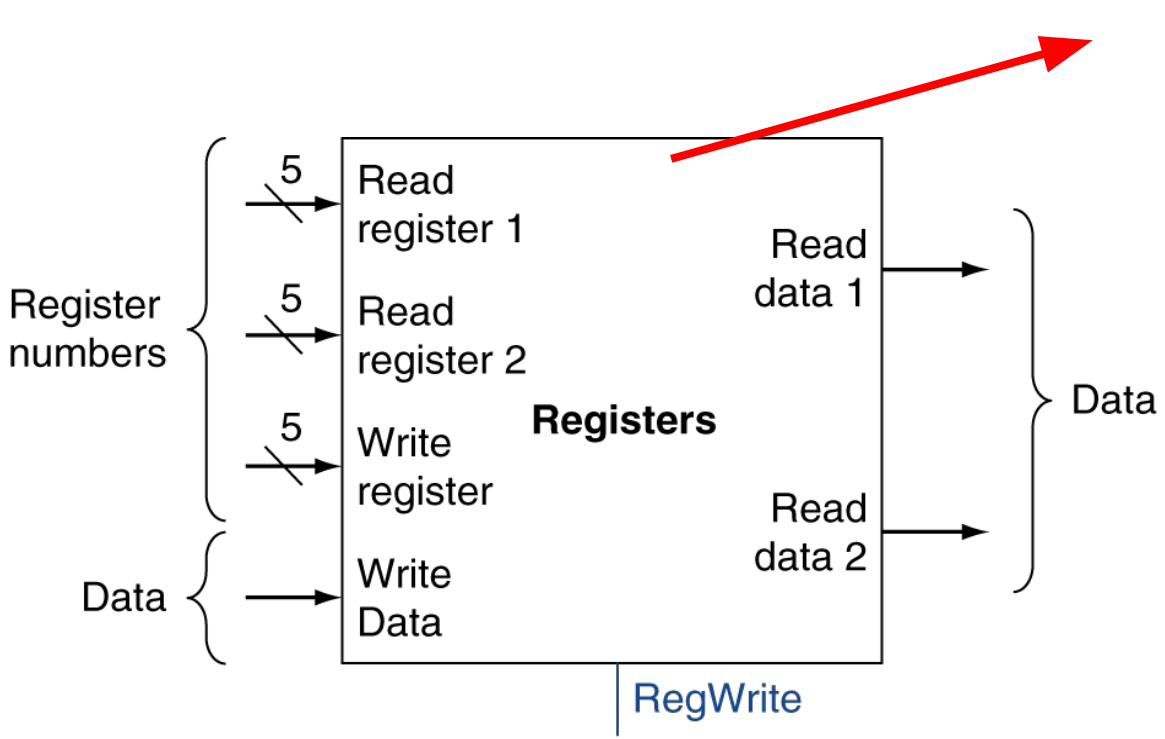
Instruction Memory

Immediate values are already stored/embed in the instruction.

Loading Operands

Register

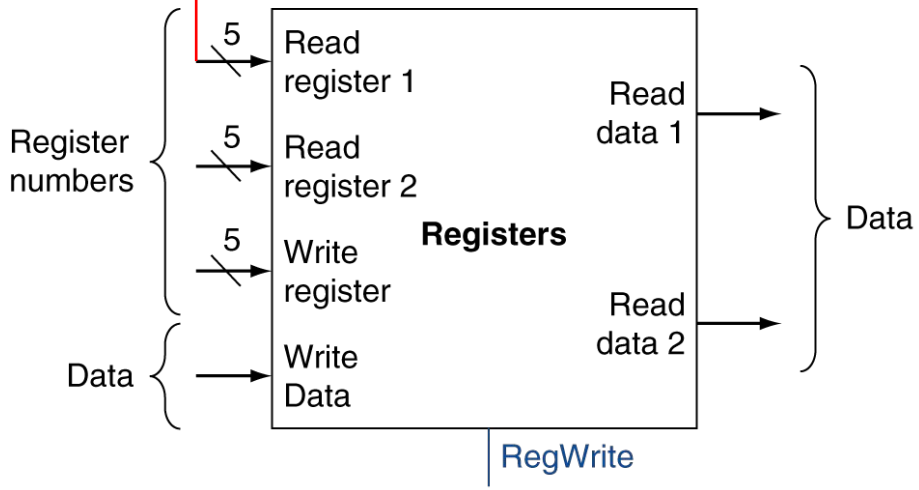
This is called Register File.



Loading Operands

Register

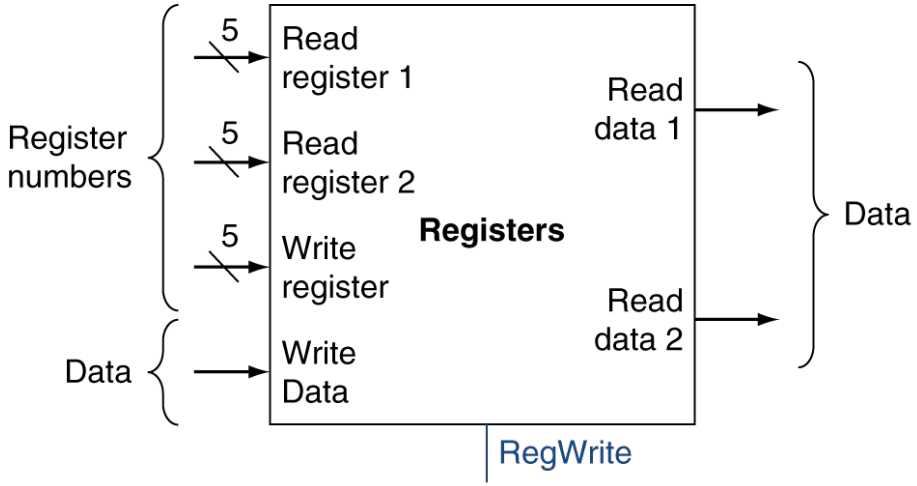
31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type



Loading Operands

Register

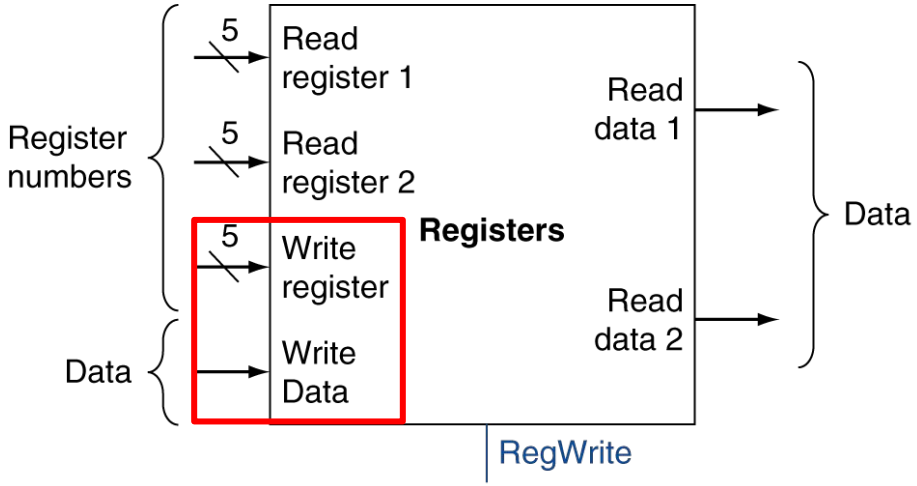
31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type



Loading Operands

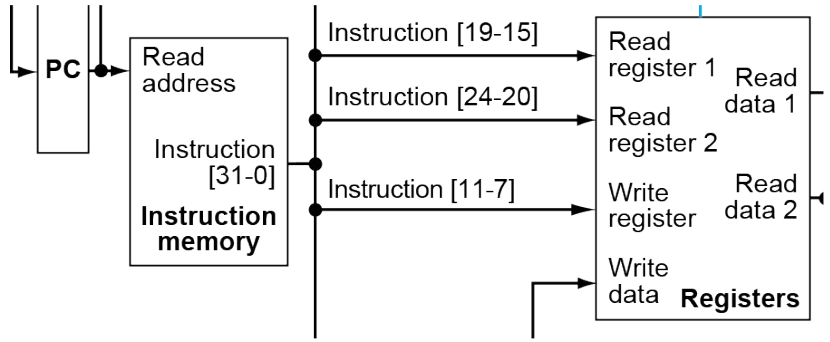
Register

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type



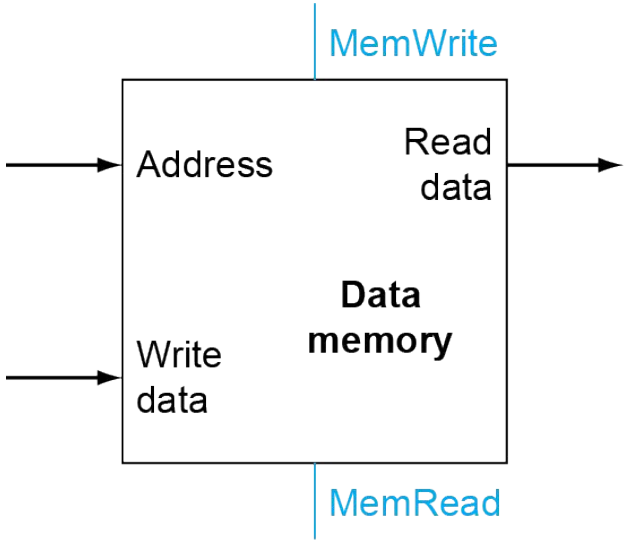
Connecting IMEM and RegFile

32-bit instruction is broken down into chunks and connected to different parts of the datapath.



Loading Operands
Memory

- *Address*
  - Different addressing modes
- *Read or Write (Load/Store)*
  - Need to write data for store (e.g., *rd*)



★ For now, let's assume that data and instruction memories are **separate**.

Loading Operands

Imm, Reg, or Memory?

- Depending on which one(s) are needed, we can load them.

Loading Operands

Imm, Reg, or Memory?

- Depending on which one(s) are needed, we can load them.

- but, *how could we know which instruction is this?*

Loading Operands

Imm, Reg, or Memory?

- Depending on which one(s) are needed, we can load them.

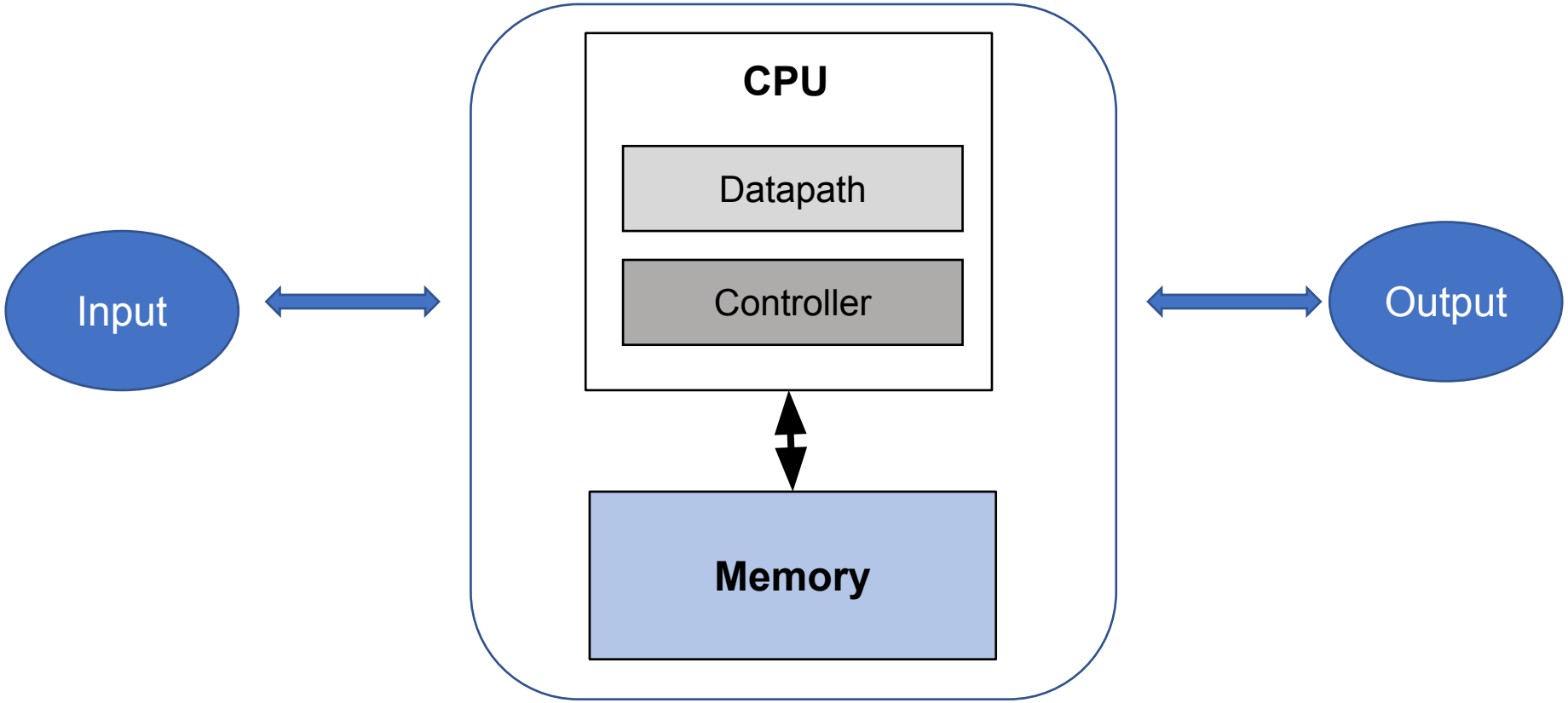
- but, *how could we know which instruction is this?*

➔ *We need a controller!*

Datapath and Controller

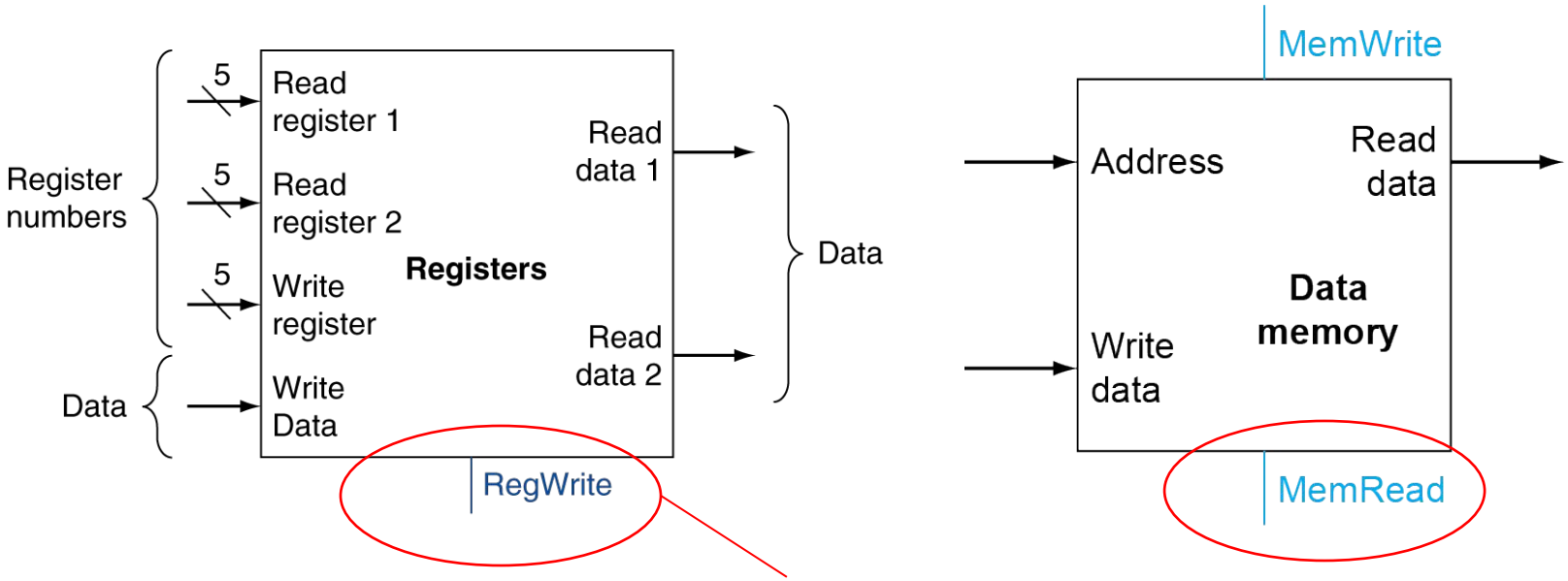
- Every CPU microarchitecture consists of *two* parts:
  - ***Datapath:*** a collection of *functional* units that process the data and create the data-flow.
  - ***Controller:*** a unit that directs the operation/activities on the datapath (e.g., which operands to load).

Von-Neumann Model



Loading Operands
Controller

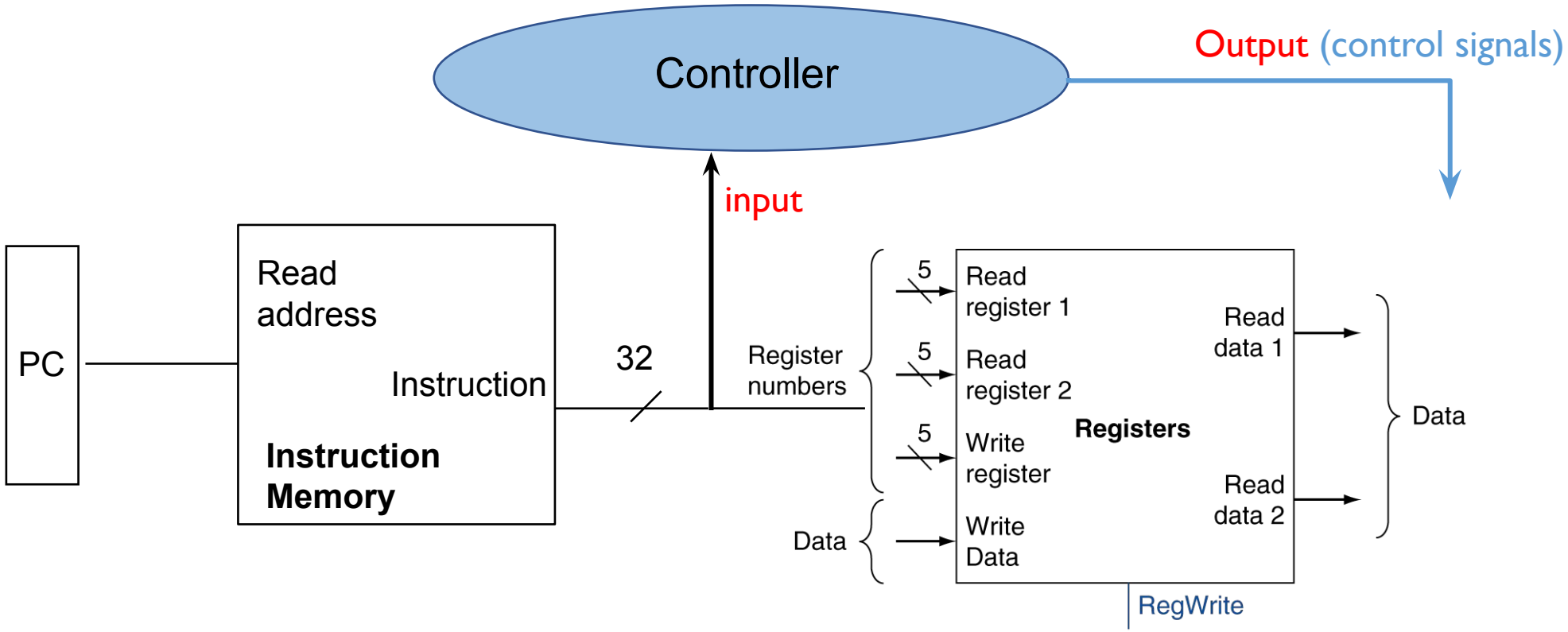
- Controller *decides* which one to use or what to do.



These are coming from the controller

Controller

Directing the operations for each Instruction



How to build a controller?

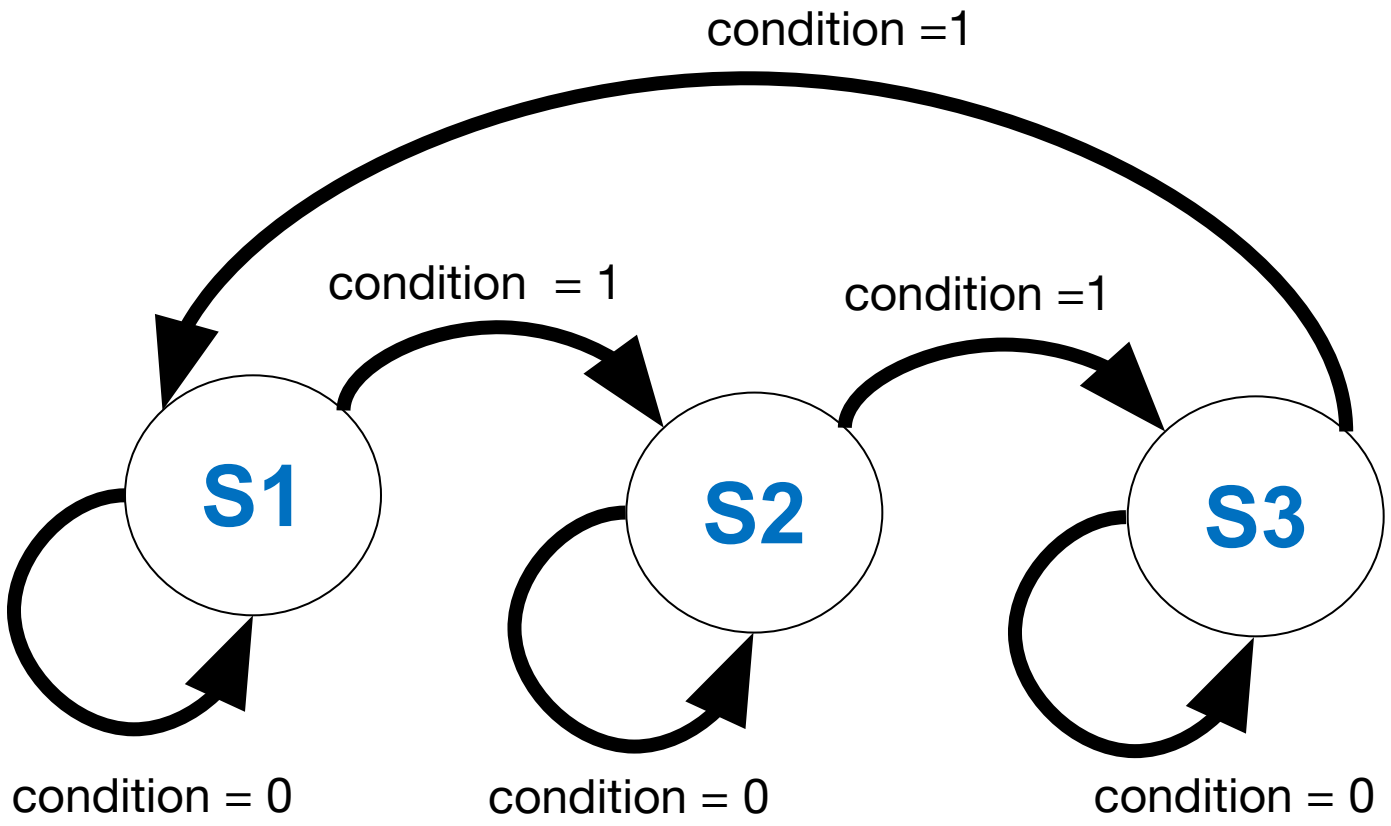
How to build a controller?

– *Use finite state machines!*

Fine-State Machine (FSM)

- A *mathematical* model of computation.
- At each point, the system *can be only at one* of the several, but ***finite***, states.
- FSM shows all the states and how they transit to each other.
- FSM also includes finite number of inputs and outputs.

State Machine Diagram



FSM/Controller for a Processor

- A giant FSM with many states!

FSM/Controller for a Processor

- A giant FSM with many states!
- Lifecycle of each instruction in the controller:
  - *Initial State:* reading the instruction and decoding it.
  - *Next State:* each instruction has its own state.
  - *Future State:* depending on the instruction, there could be several states (e.g., reading registers, loading memory, arithmetic operations, etc.).

FSM/Controller for a Processor

- A giant FSM with many states!
- Lifecycle of each instruction in the controller:
  - *Initial State:* reading the instruction and decoding it.
  - *Next State:* each instruction has its own state.
  - *Future State:* depending on the instruction, there could be several states (e.g., reading registers, loading memory, arithmetic operations, etc.).

➔ Once all the activities are done, the controller should go back to the *initial state*, and repeat this process for the next instruction.

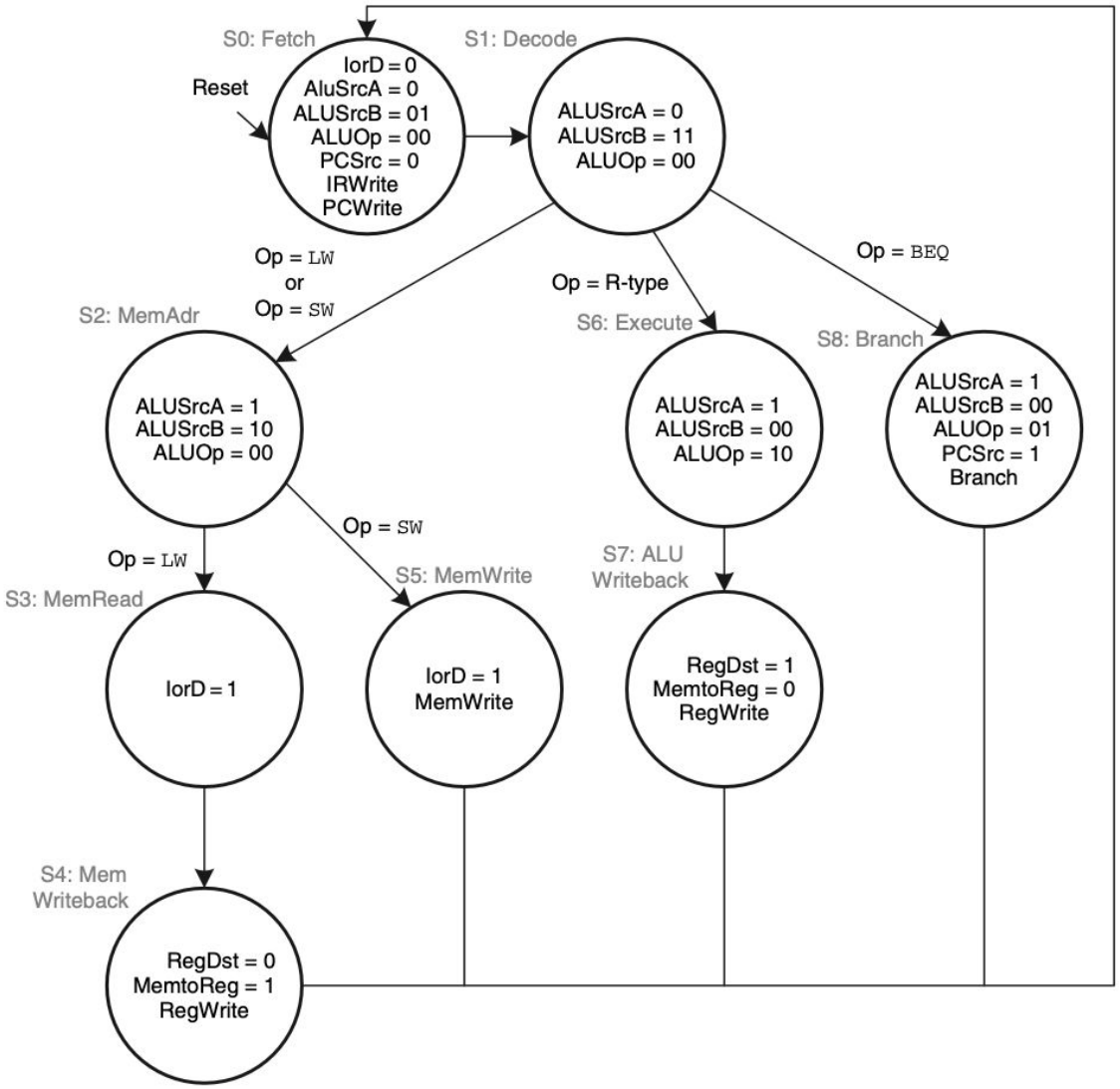
Single-Cycle Design

➔ Once all the activities are done, the controller should go back to the *initial state*, and repeat this process for the next instruction.

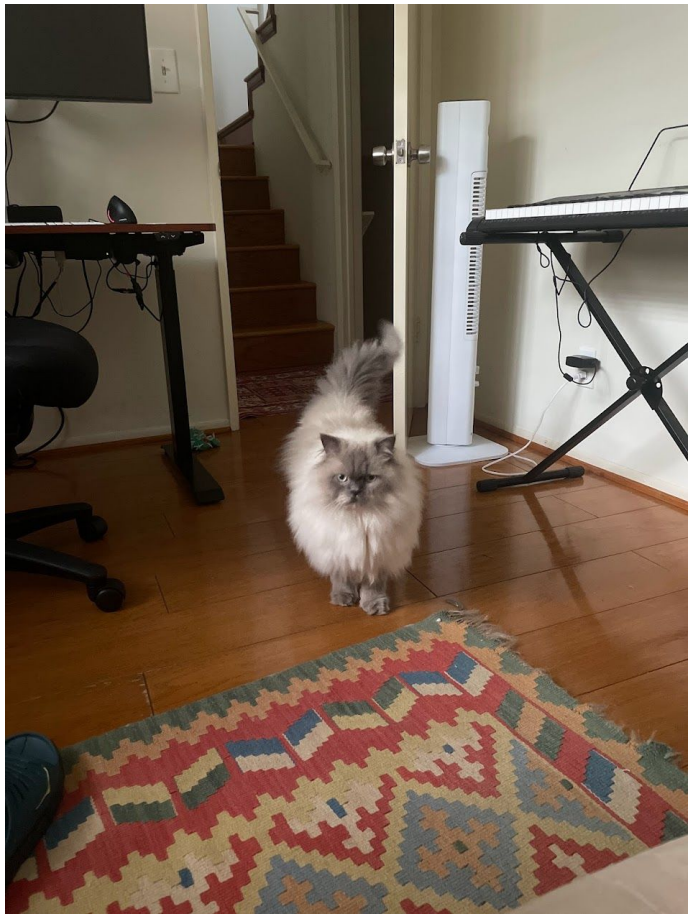
We call this single-cycle design!

FSM Full View

- Ignore the names and values for now!



Ok, where were we?



Nader Sehatbakhsh <nsehat@ee.ucla.edu>

Lifecycle of an instruction

1- Instruction is read (fetch) from the (instruction) memory

2- Operands should be loaded
- RegFile, (data) Memory, Imm.
  - Need register number/ address for each operand.

3- Operation should be executed
- Arithmetic (which type?), data movement, control-flow

4- Results should be stored
- RegFile or memory?

5- PC should be updated
- Sequential, jump, or branch?

Lifecycle of an instruction

1- Instruction is read (fetch) from the (instruction) memory

2- Operands should be loaded
- RegFile, (data) Memory, Imm.
  - Need register number/ address for each operand.

3- Operation should be executed
- Arithmetic (which type?), data movement, control-flow

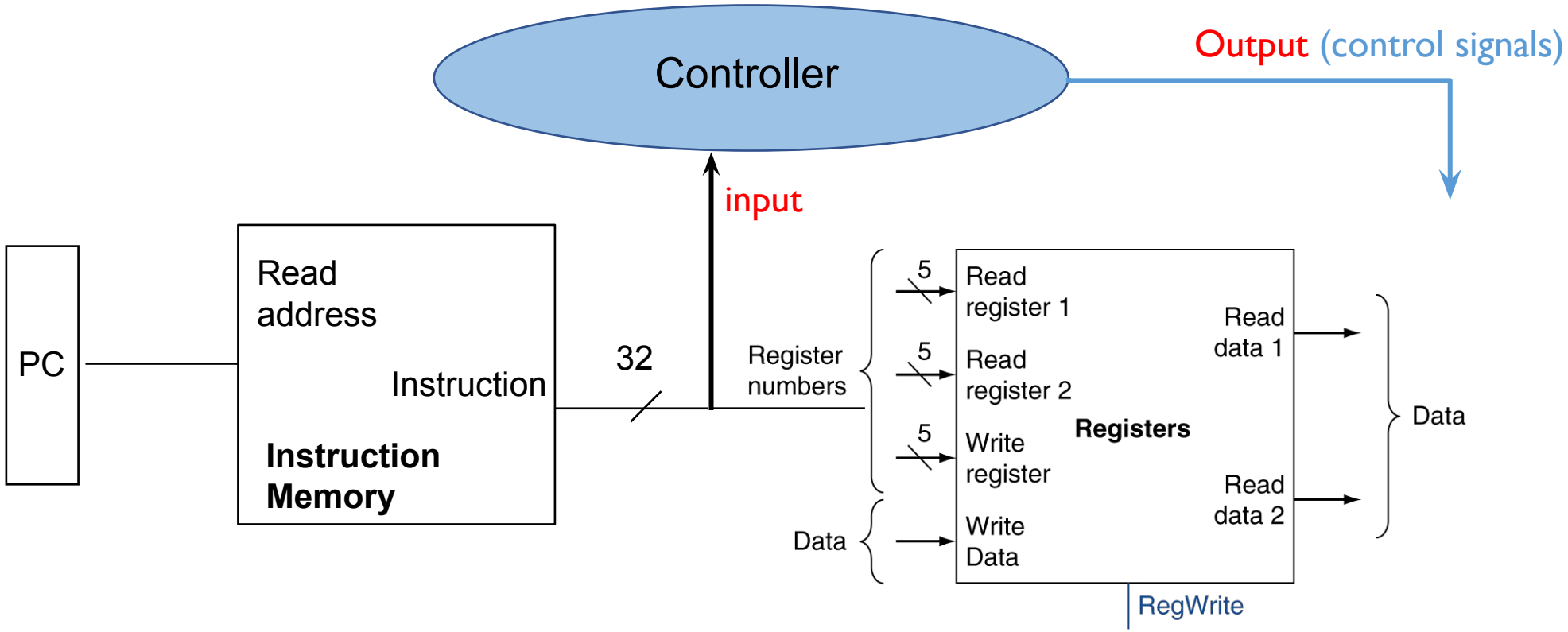
4- Results should be stored
- RegFile or memory?

5- PC should be updated
- Sequential, jump, or branch?

Three types of operands.

Controller

Directing the operations for each Instruction



Building a Simple CPU

- RISC-V ISA, 32-bit
- Handful of instructions – total of 10
  - add, sub, and, or
  - lw, sw
  - beq
- Immediate or register operands
  - addi, andi,ori

Instructions

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

imm[11:0]	rs1	000	rd	0010011	ADDI
imm[11:0]	rs1	110	rd	0010011	ORI
imm[11:0]	rs1	111	rd	0010011	ANDI

imm[11:0]		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ

Instructions

opcode, funct3, funct7

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

imm[11:0]	rs1	000	rd	0010011	ADDI
imm[11:0]	rs1	110	rd	0010011	ORI
imm[11:0]	rs1	111	rd	0010011	ANDI

imm[11:0]		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ

Nader Sehatbakhsh <nsehat@ee.ucla.edu>

Instructions

rs1, rs2, rd

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

imm[11:0]	rs1	000	rd	0010011	ADDI
imm[11:0]	rs1	110	rd	0010011	ORI
imm[11:0]	rs1	111	rd	0010011	ANDI

imm[11:0]		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ

Nader Sehatbakhsh <nsehat@ee.ucla.edu>

Instructions

imm

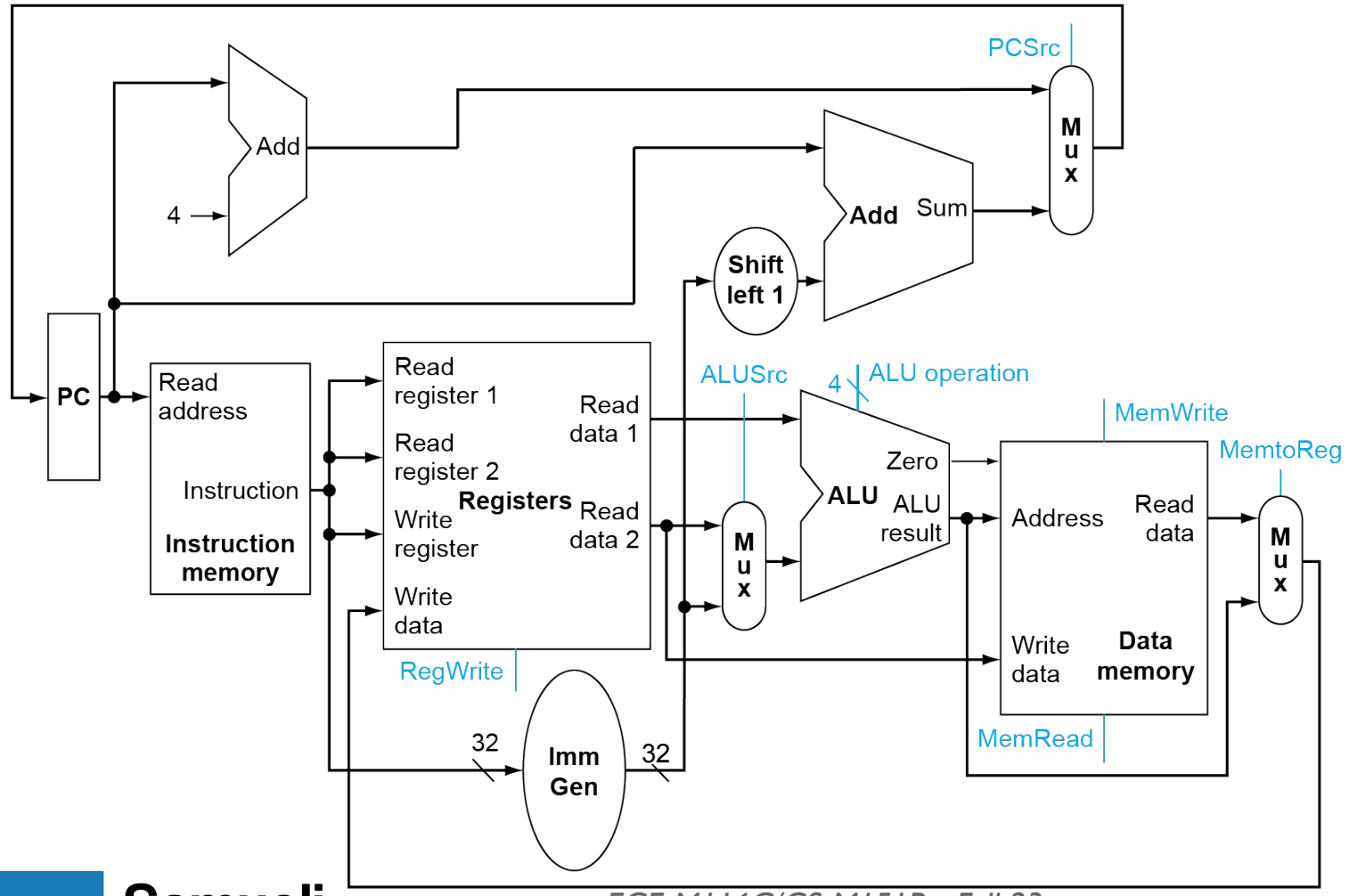
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

imm[11:0]	rs1	000	rd	0010011	ADDI
imm[11:0]	rs1	110	rd	0010011	ORI
imm[11:0]	rs1	111	rd	0010011	ANDI

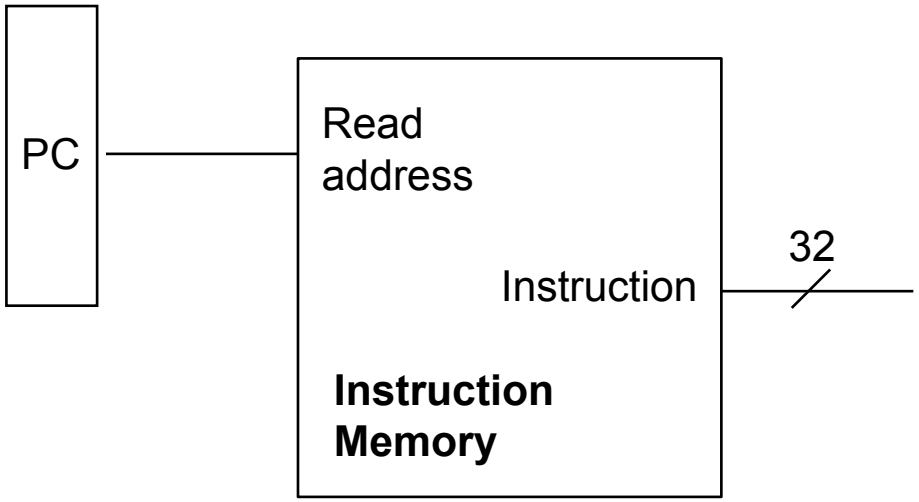
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ

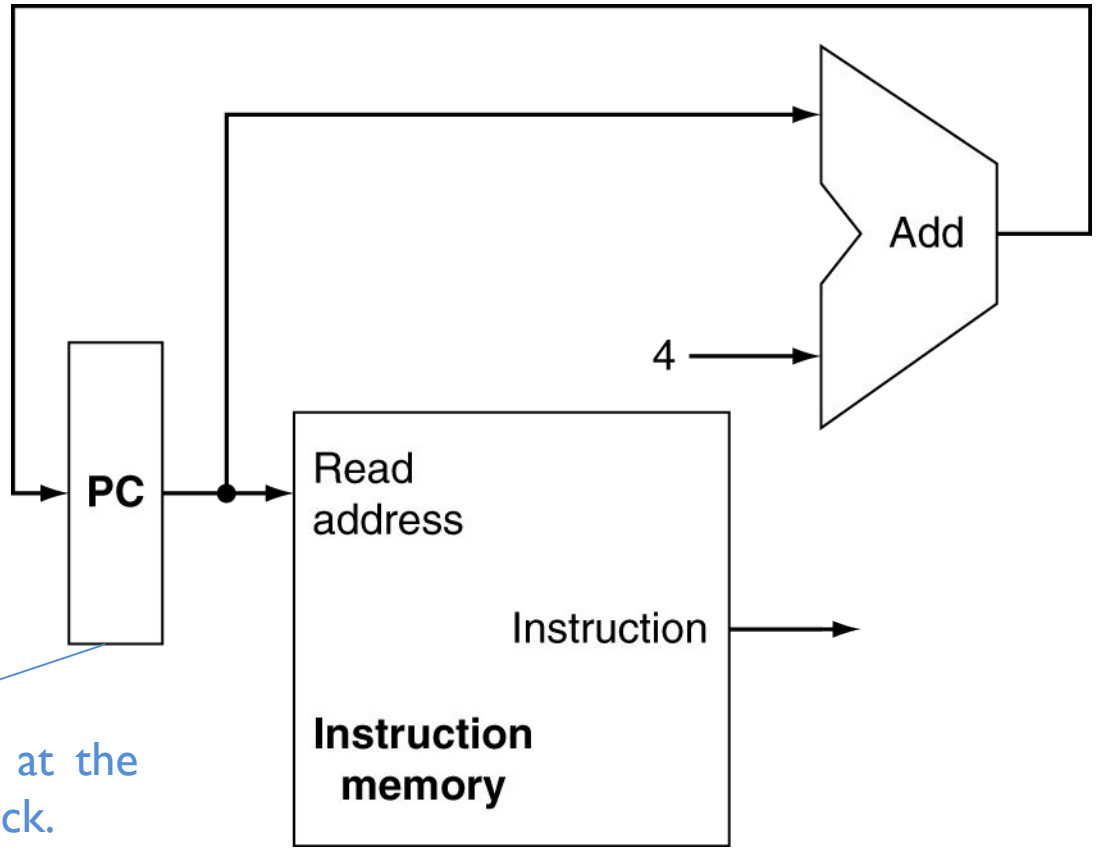
Datapath


PCSrc
Add
4
M
u
x
Add
Sum
Shift
left 1
PC
Read
address
Instruction
Instruction
memory
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU operation
M
u
x
ALU
Zero
ALU
result
MemWrite
MemtoReg
Address
Read
data
Write
data
Data
memory
MemRead
M
u
x
32
Imm
Gen
32

Datapath

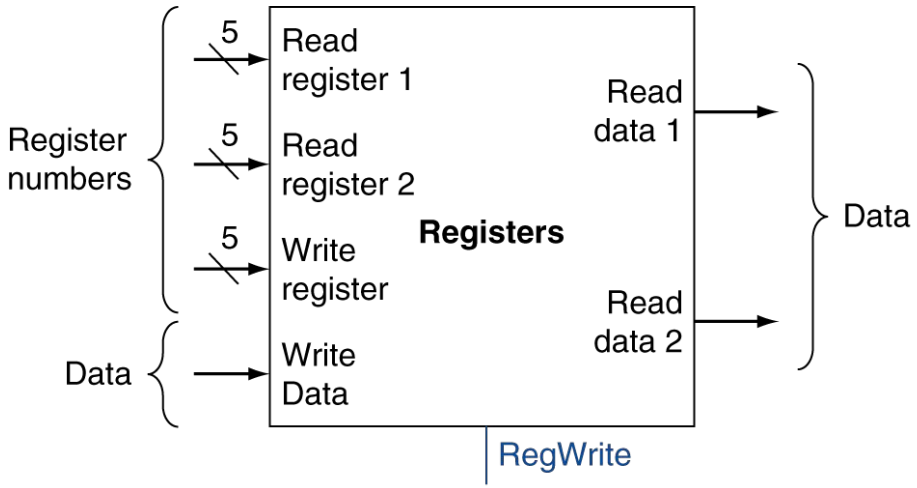


Datapath
Creating nextPC

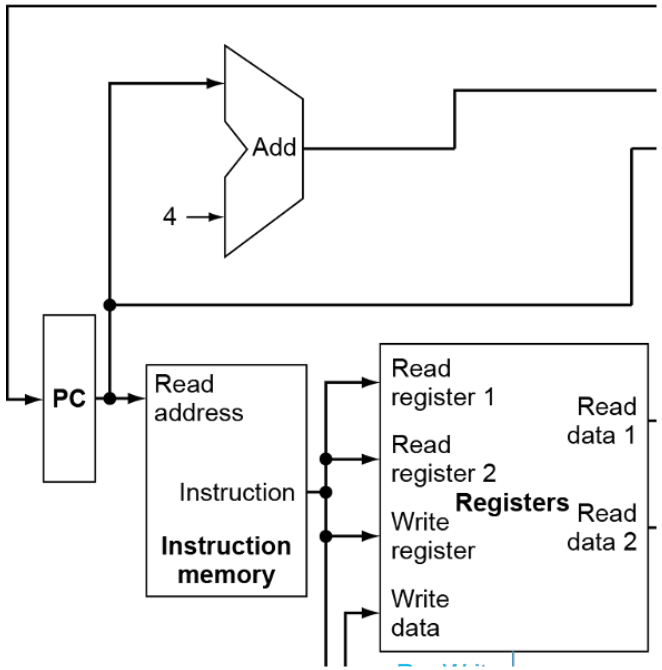


PC gets its new value at the positive edge of the clock.

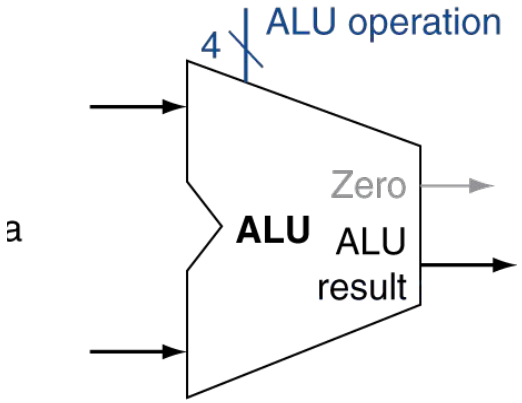
Datapath
Register File



Inst Mem, NextPC, RegFile

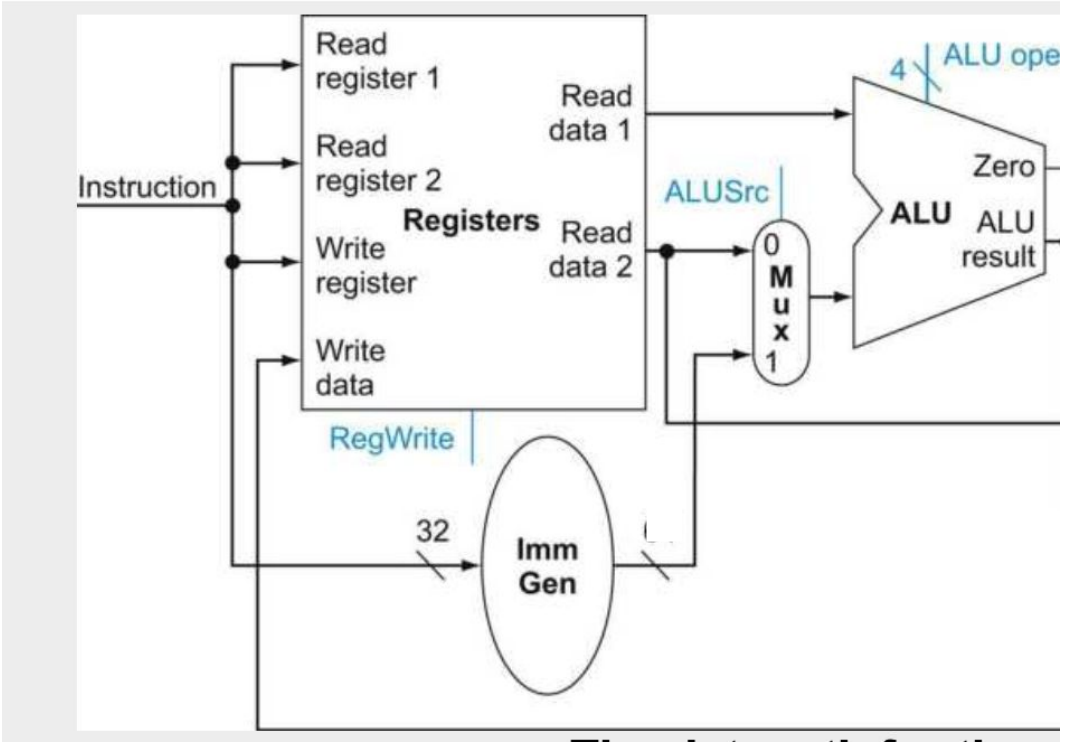


Datapath
ALU

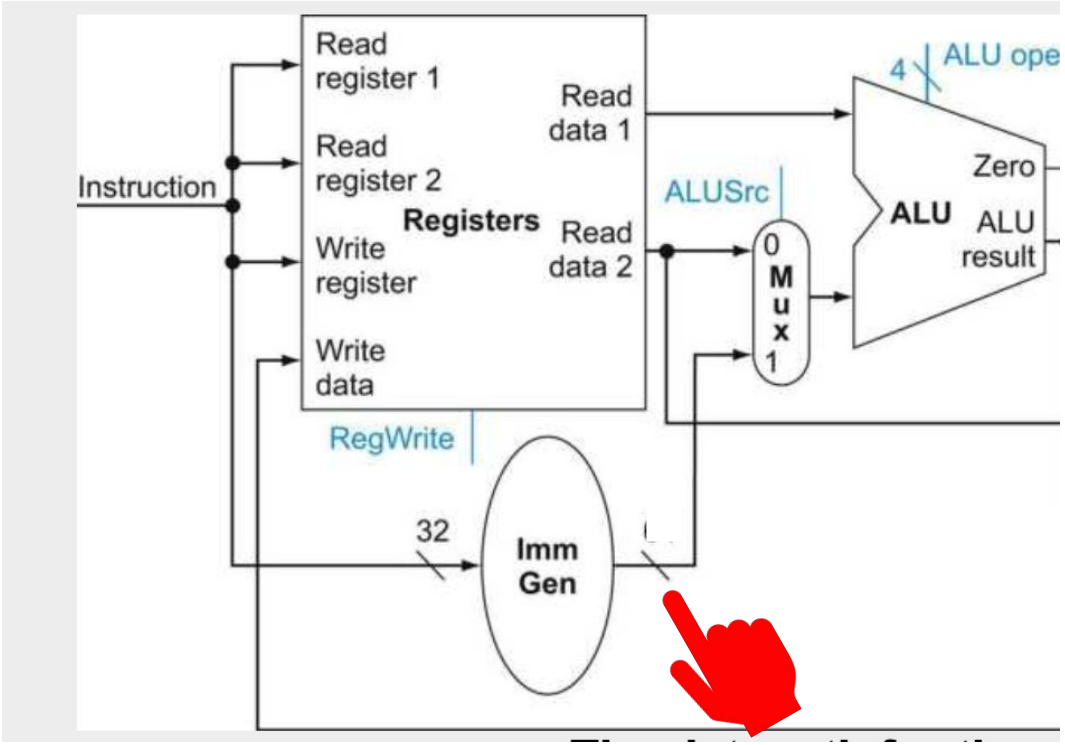


Pay attention to the blue (control signals) too!

RegFile to ALU

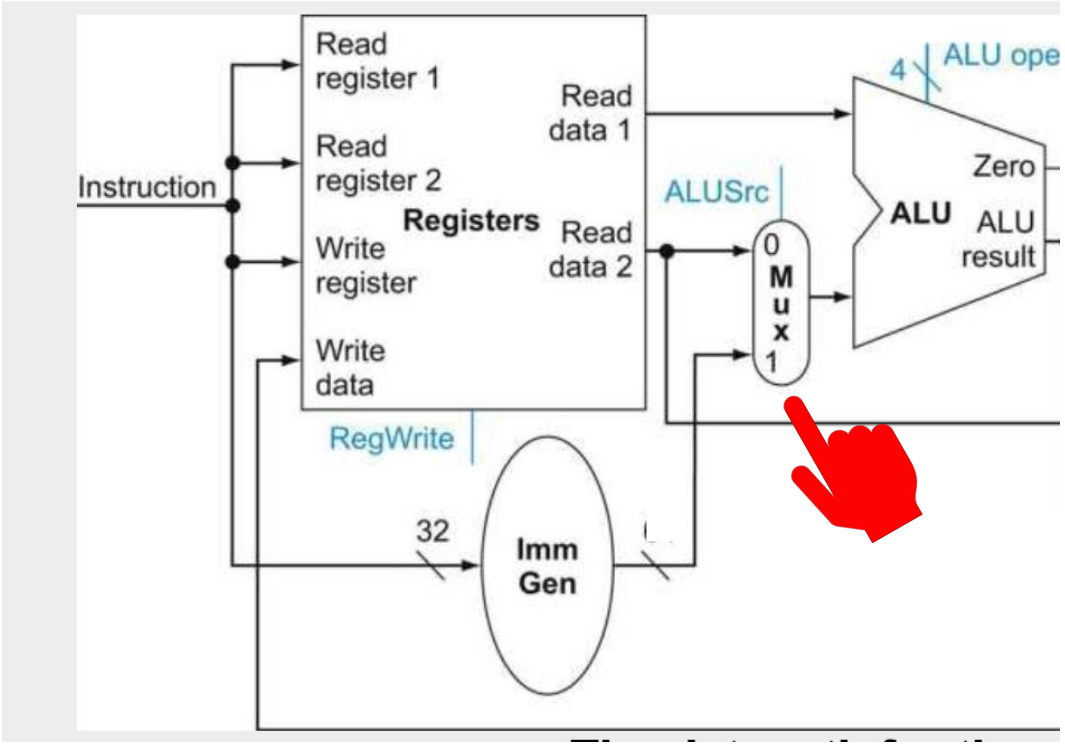


RegFile to ALU

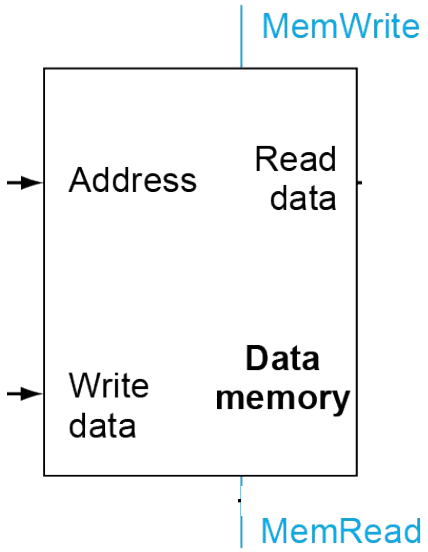


RISC-V opcode bit 6 happens to be 0 for data transfer instructions and 1 for conditional branches, and RISC-V opcode bit 5 happens to be 0 for load instructions and 1 for store instructions. Thus, bits 5 and 6 can control a 3:1 multiplexor inside the immediate generation logic that selects the appropriate 12-bit field for load, store, and conditional branch instructions.

RegFile to ALU

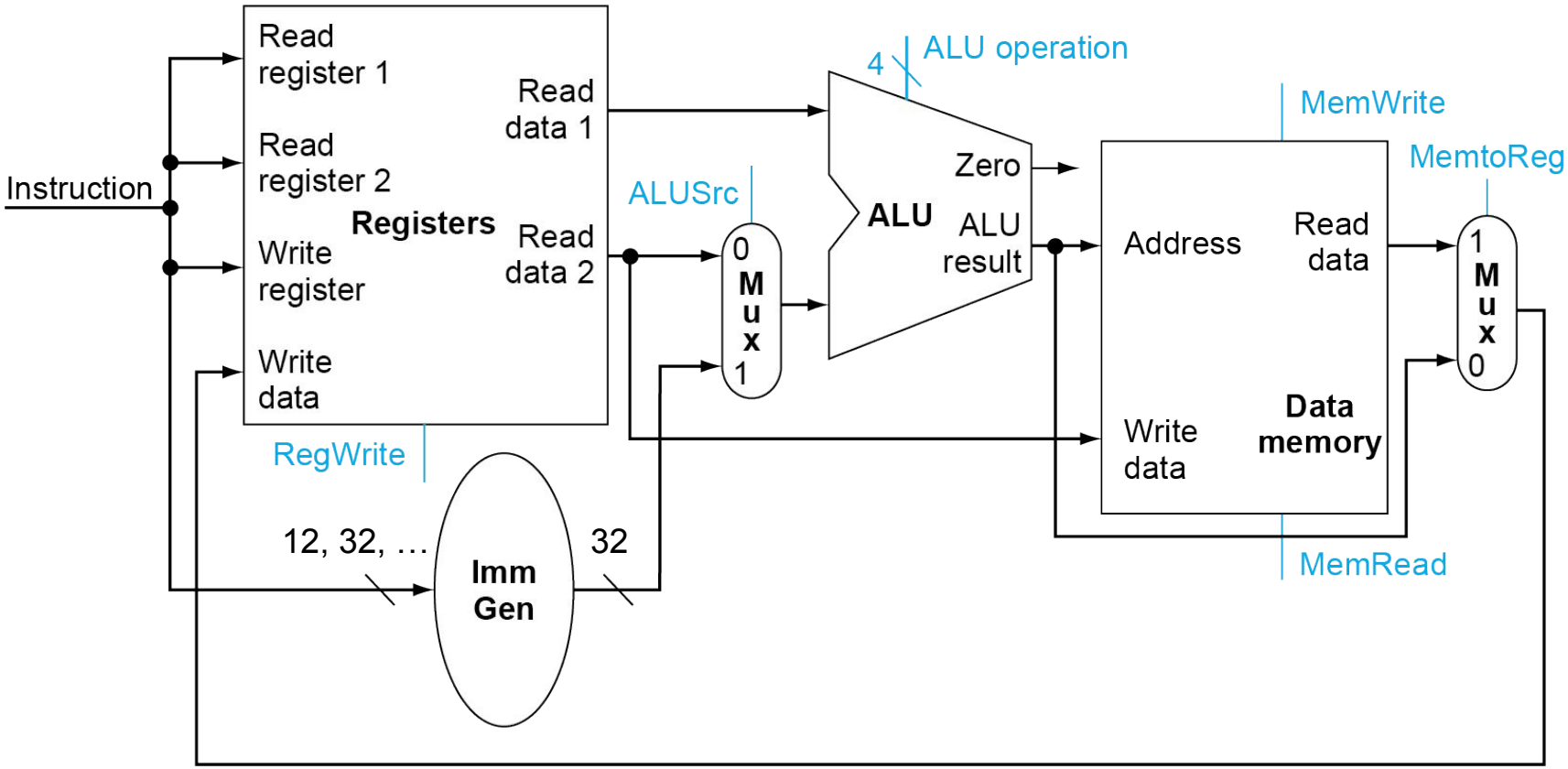


Datapath
Adding Memory

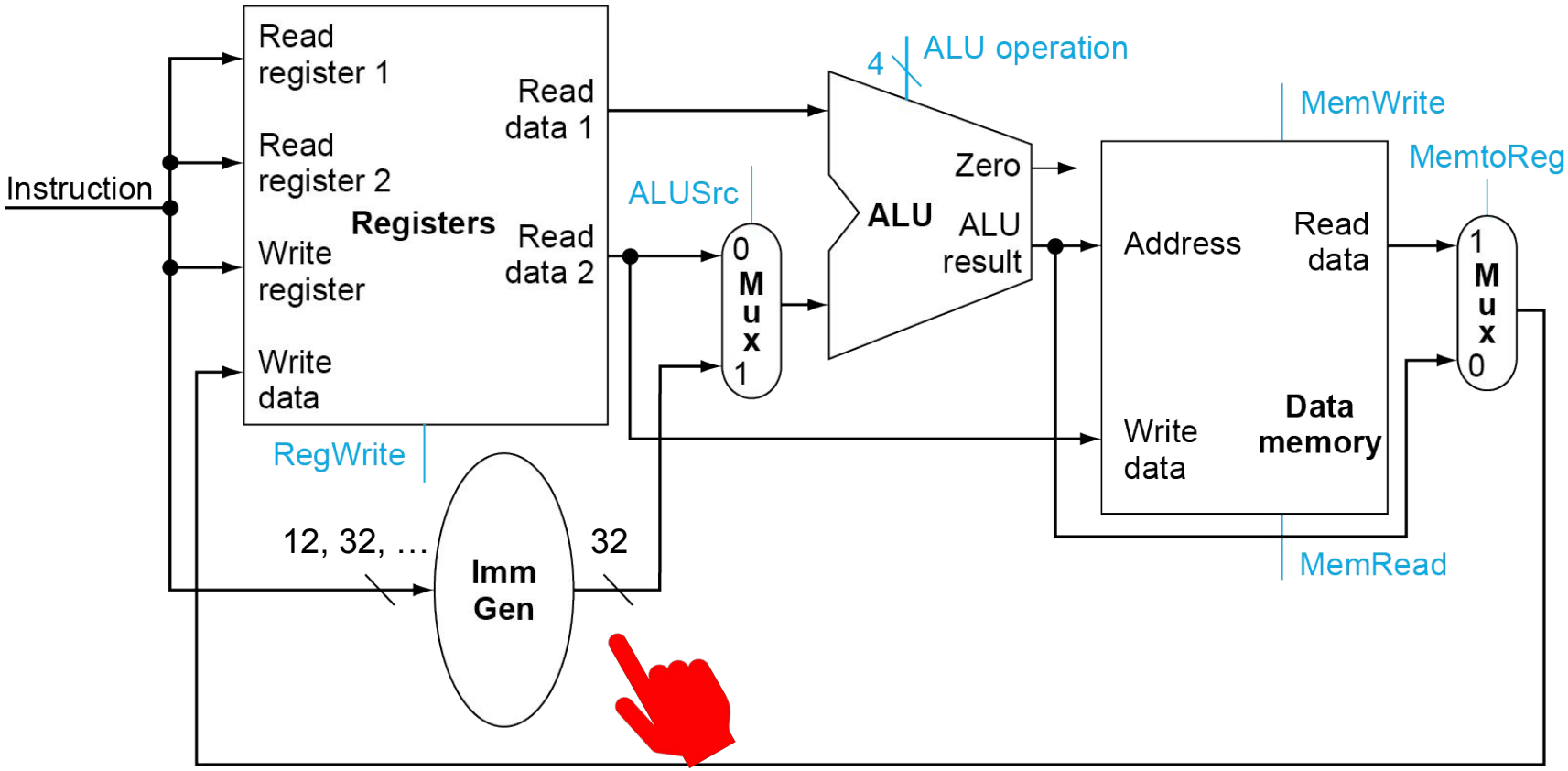


Datapath

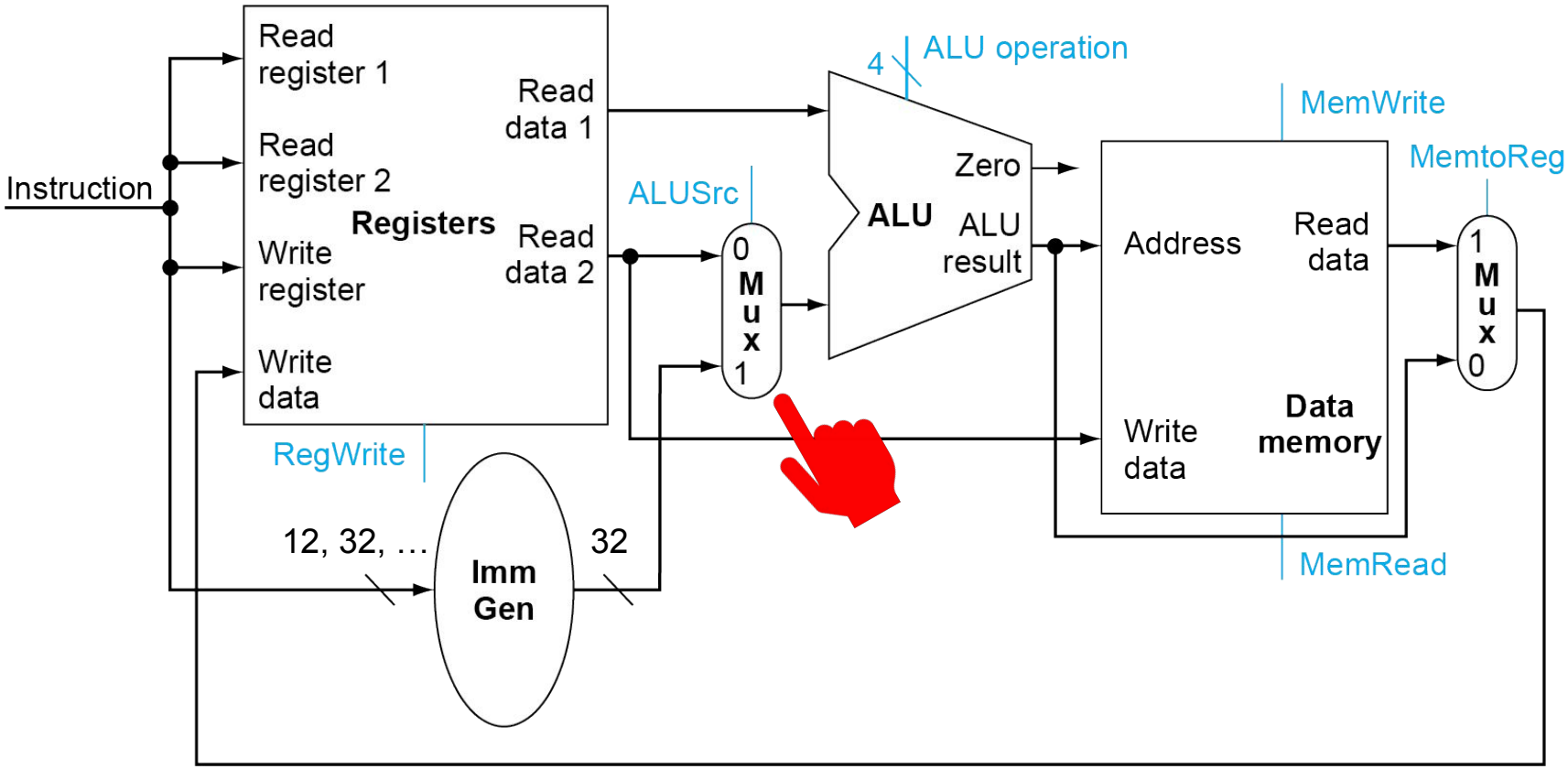
Putting all together



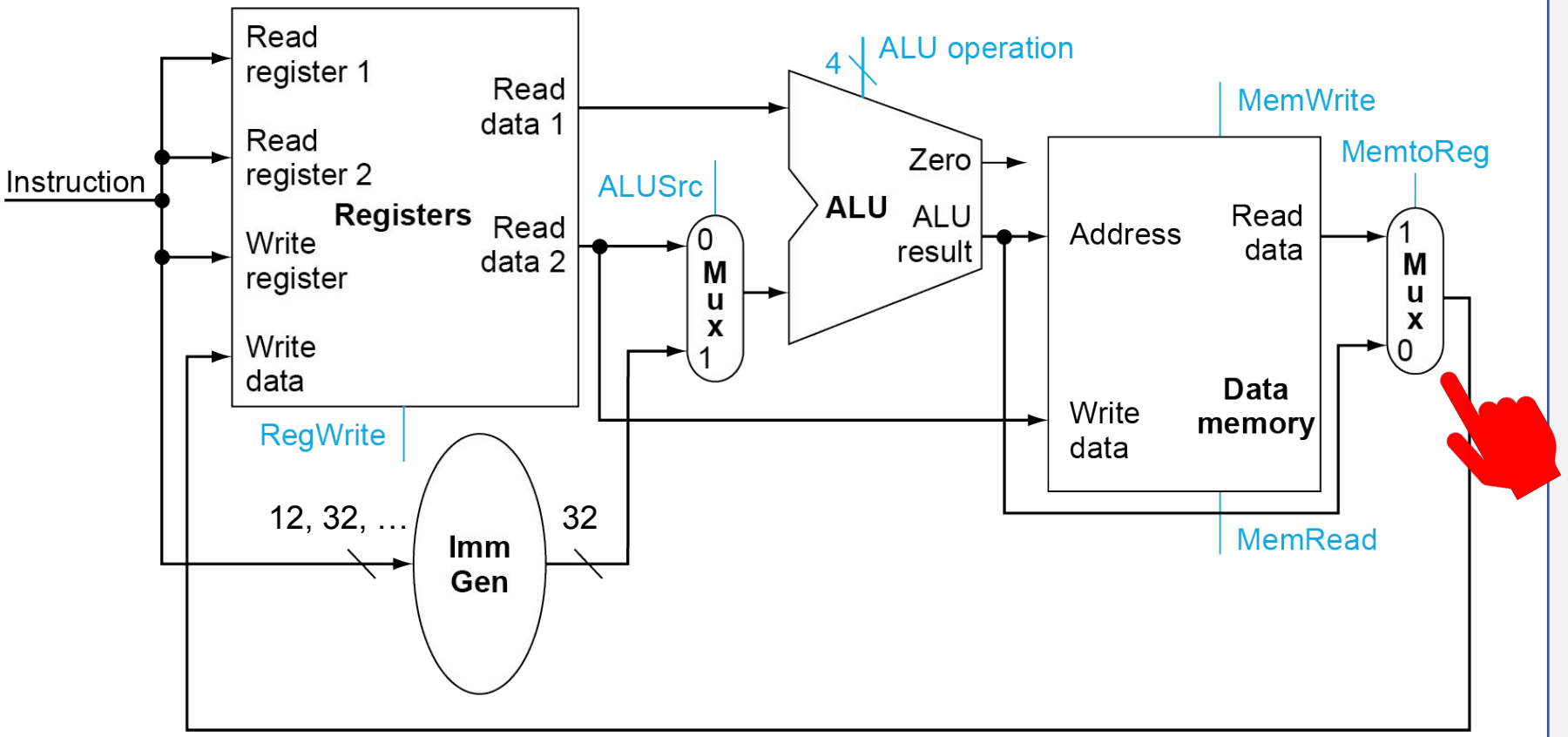
Datapath
Imm Gen.



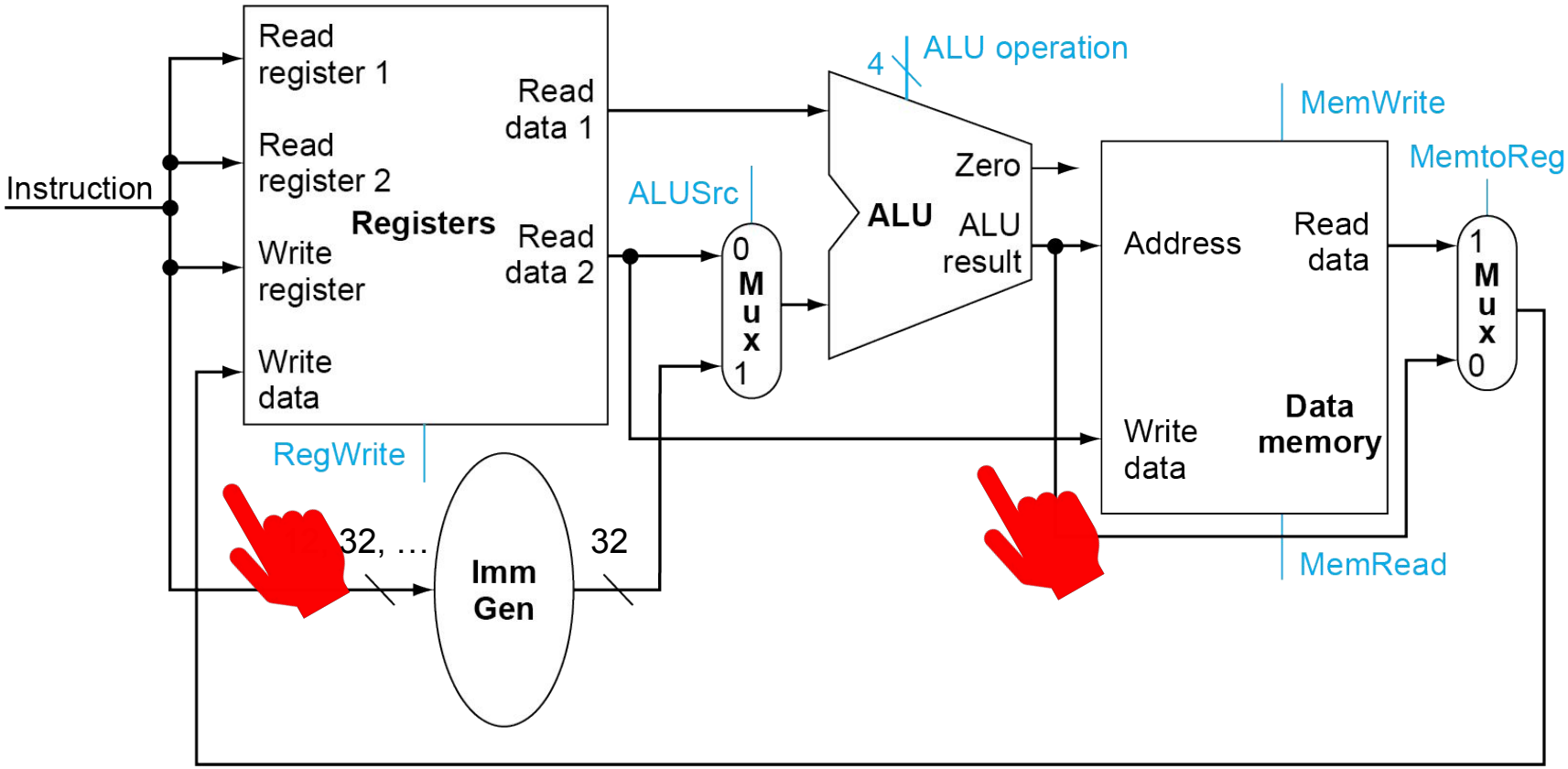
Datapath
ALUSrc.



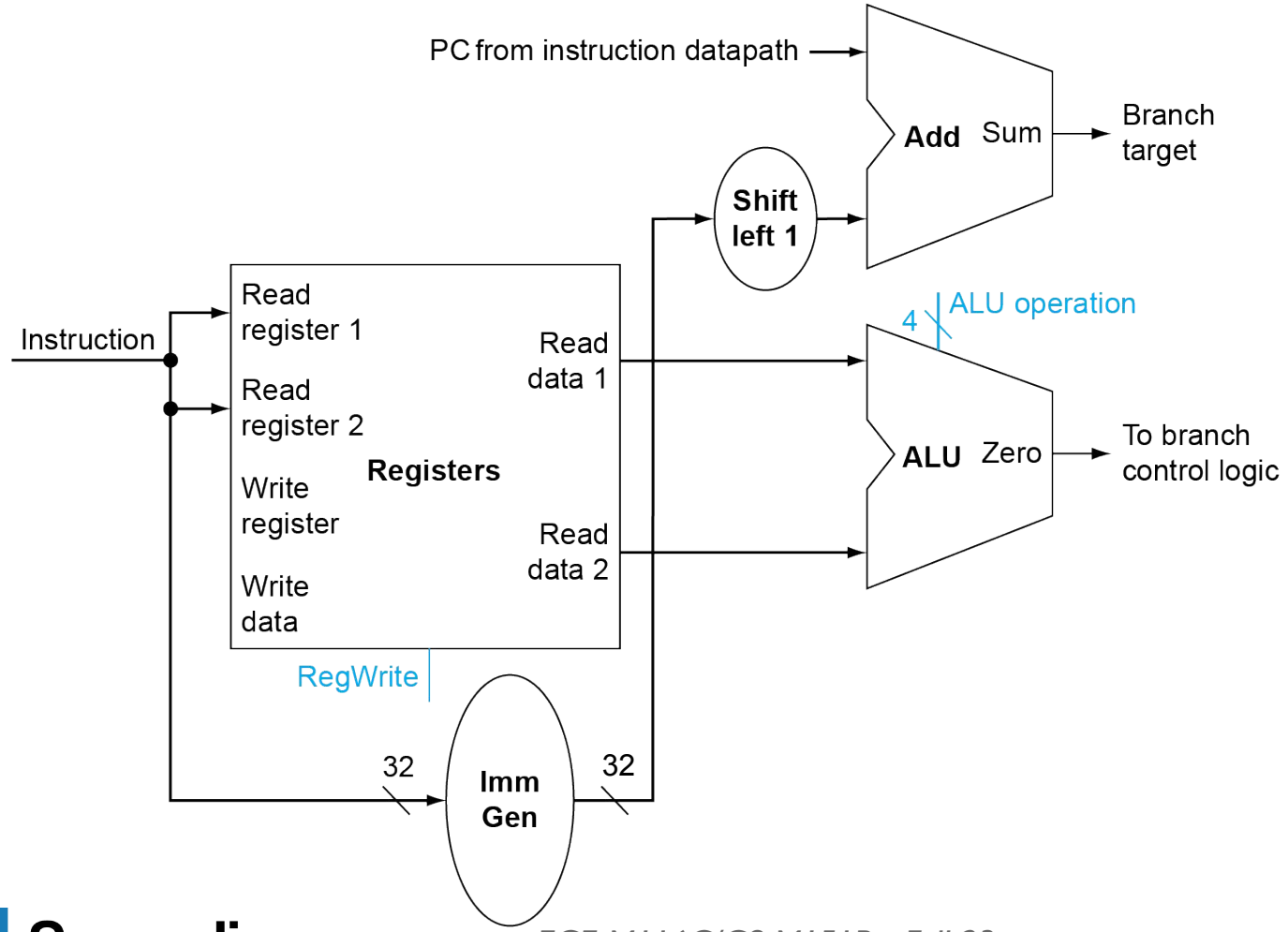
Datapath
MemtoReg.



Datapath
Storing Data

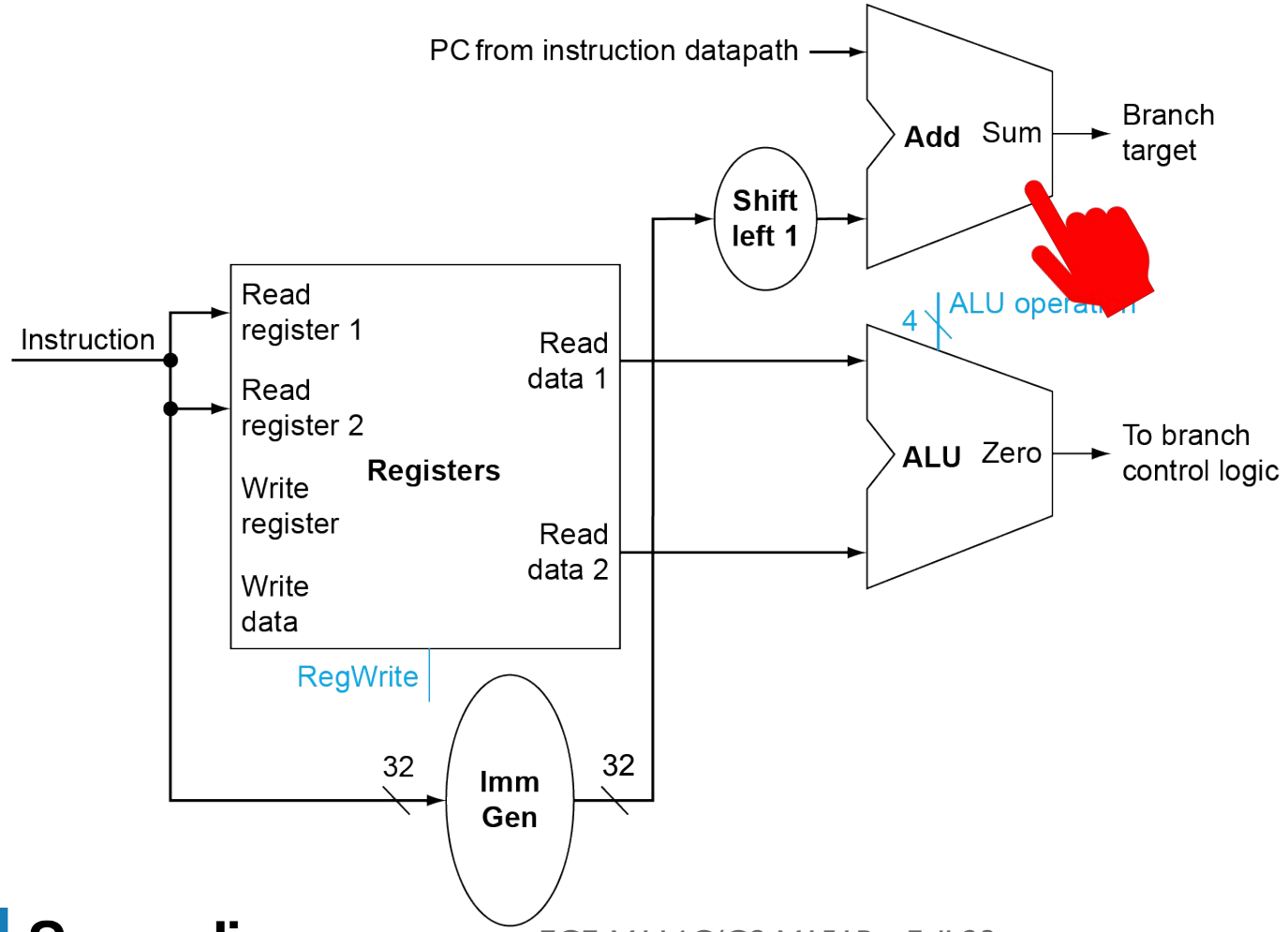


Branch Instruction



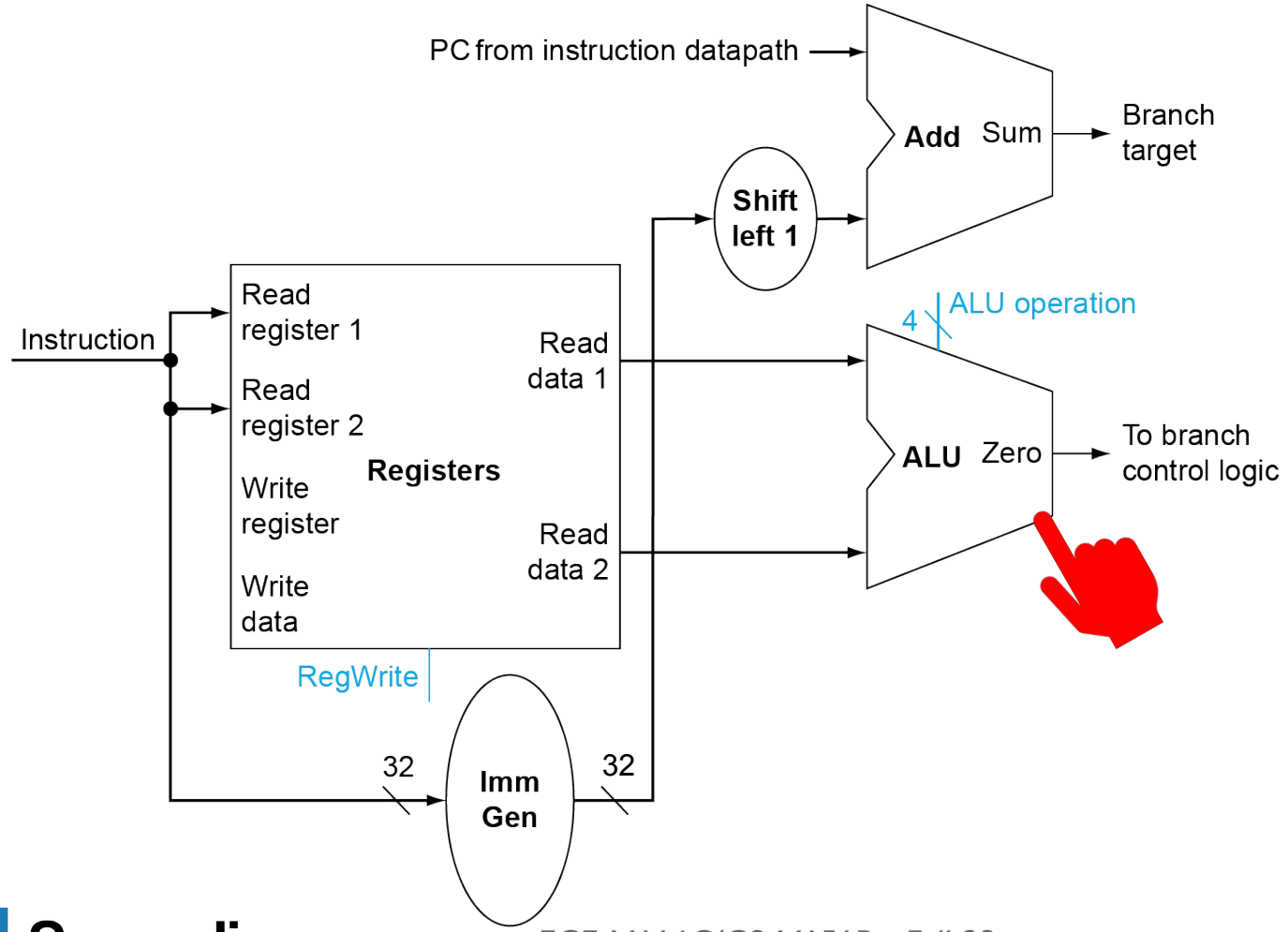
Branch Instruction

Target Addr.

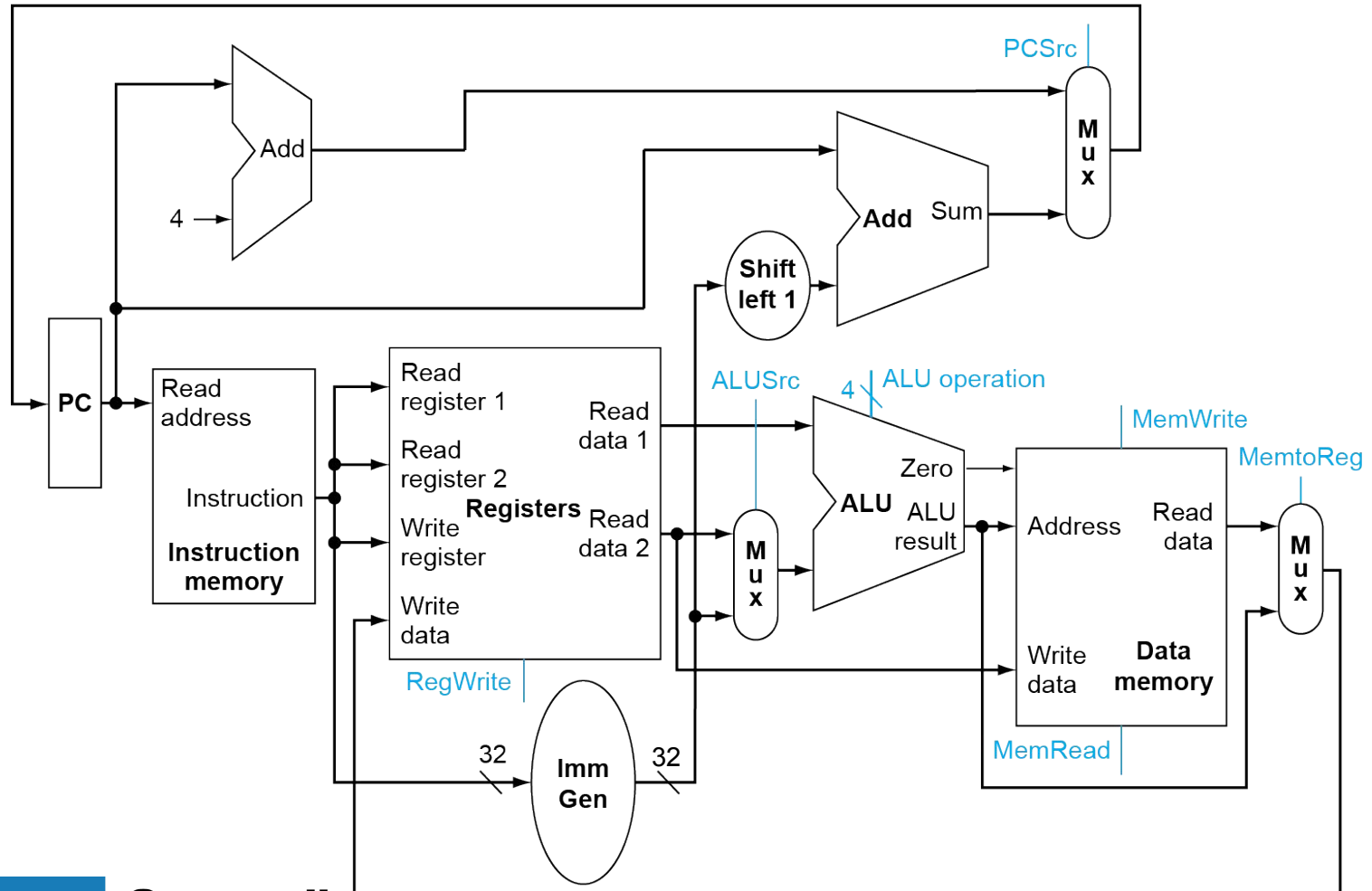


Branch Instruction

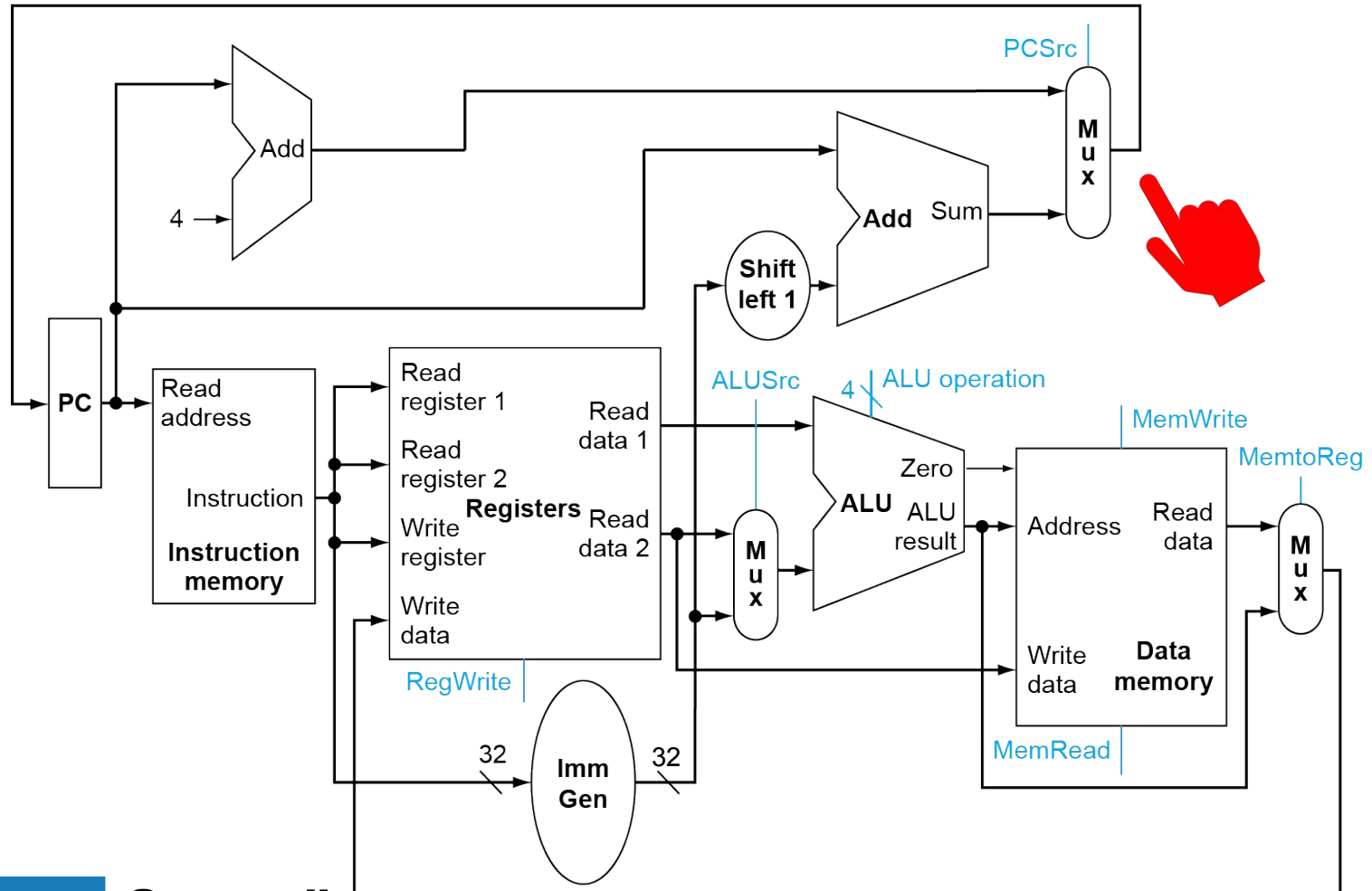
Branch Cond.



Datapath - *Combined*



Datapath - *Combined*



Control Signals

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

Instructions

0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

imm[11:0]	rs1	000	rd	0010011	ADDI
imm[11:0]	rs1	110	rd	0010011	ORI
imm[11:0]	rs1	111	rd	0010011	ANDI

imm[11:0]		rs1	010	rd	0000011	LW
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ

How to create an FSM?

- Everything should be first assigned to zero.
- Based on the instruction (opcode), and current state, some control signals should be changed.

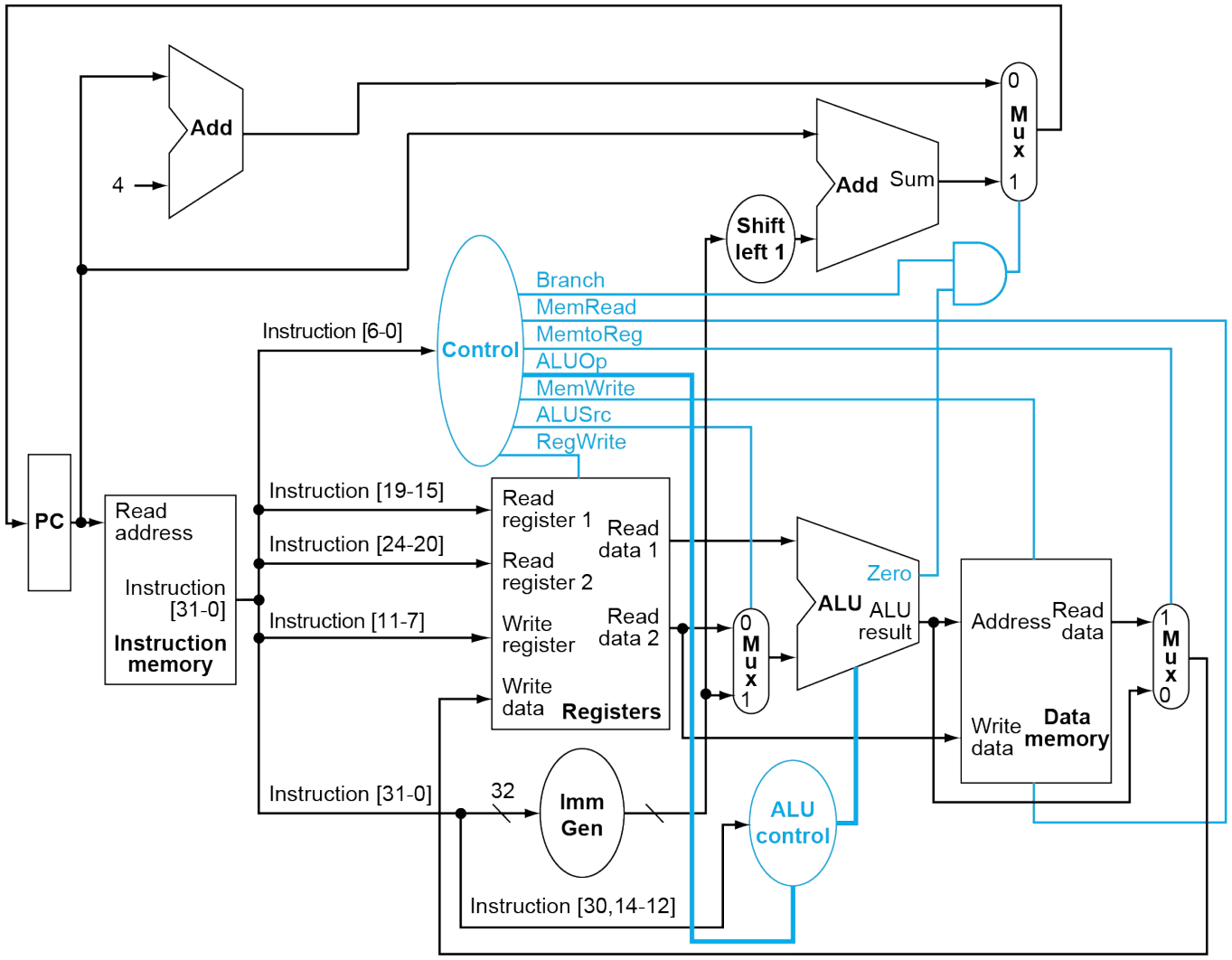
Active low vs. Active high!

How to create an FSM?

- We need to first find out which instruction is this.
  - We use opcode for that.
  - Think about it as a big switch case.
  - Some opcodes (e.g., R-type instructions), have multiple candidates. We need to use funct3 and funct7 to further decode the instruction.
- Once we know the instruction, we issue proper control signals (based on the list).

Control Signals

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



*Images were taken from Hennessy Patterson Book [1].

Datapath + Controller

Single-Cycle Design

Instruction	Opcode	RegWrite	AluSrc	Branch	MemRe	MemWr	MemtoReg	ALUOp
R-type	0110011							
I-type	0010011							
lw	0000011							
sw	0100011							
beq	1100011							

***I-type** in this table refers to all I-type instructions except load.

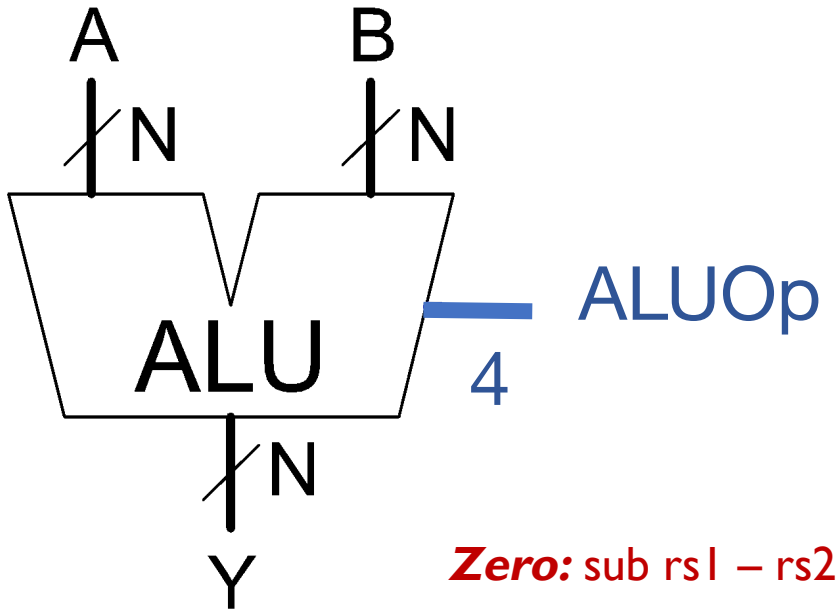
Single-Cycle Design

Instruction	Opcode	RegWrite	AluSrc	Branch	MemRe	MemWr	MemtoReg	ALUOp
R-type	0110011	1	0	0	0	0	0	-
I-type	0010011	1	1	0	0	0	0	-
lw	0000011	1	1	0	1	0	1	-
sw	0100011	0	1	0	0	1	0	-
beq	1100011	0	0	1	0	0	0	-

***I-type** in this table refers to all I-type instructions except load.

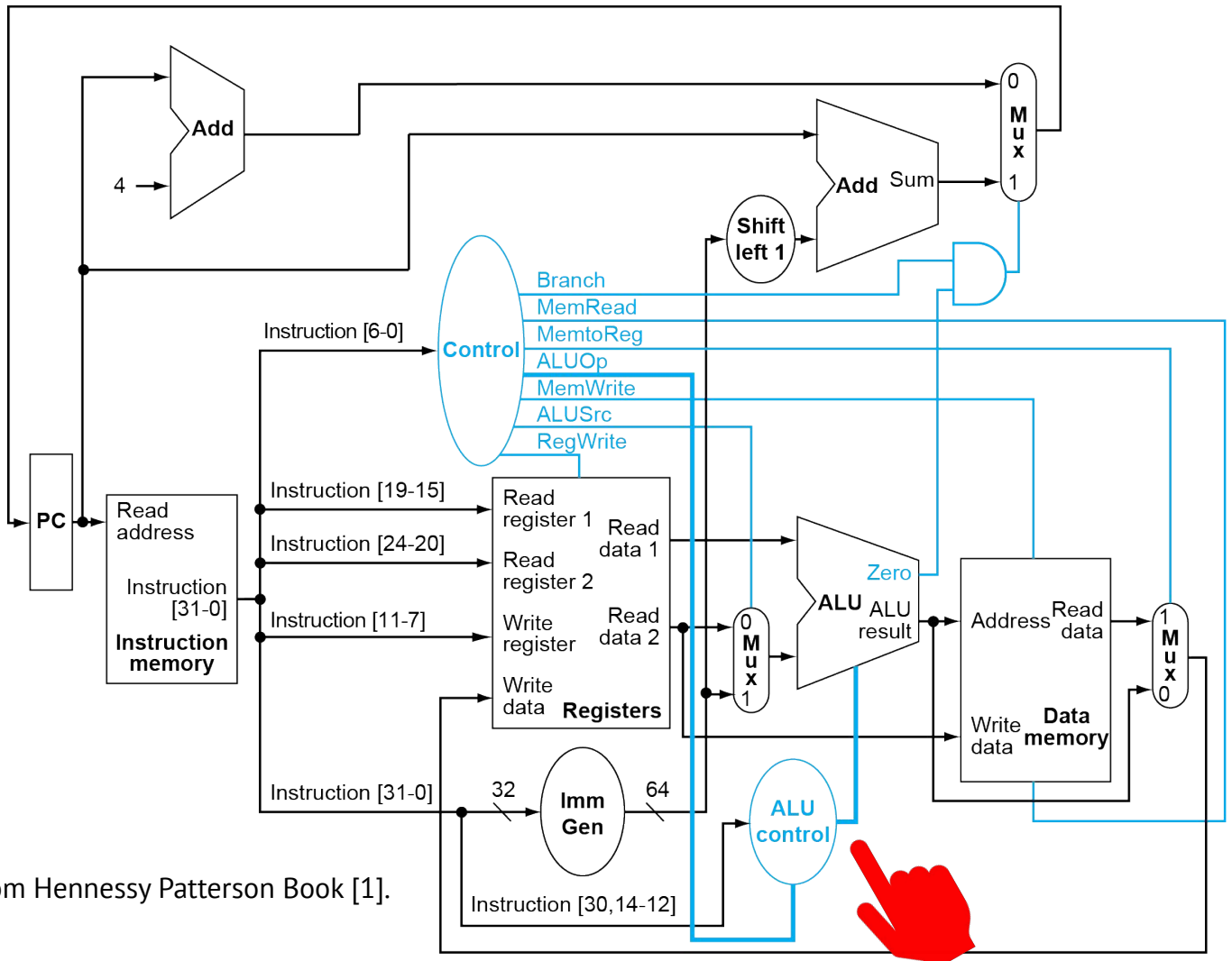
Single-Cycle Design
ALUOp

Instr.	ALU function	ALU Op
lw	add	0010
sw	add	0010
beq	subtract	0110
R-type/ I-type	add	0010
	subtract	0110
	AND	0000
	OR	0001



Zero: sub rs1 – rs2

Nader Sehatbakhsh <nsehat@ee.ucla.edu>

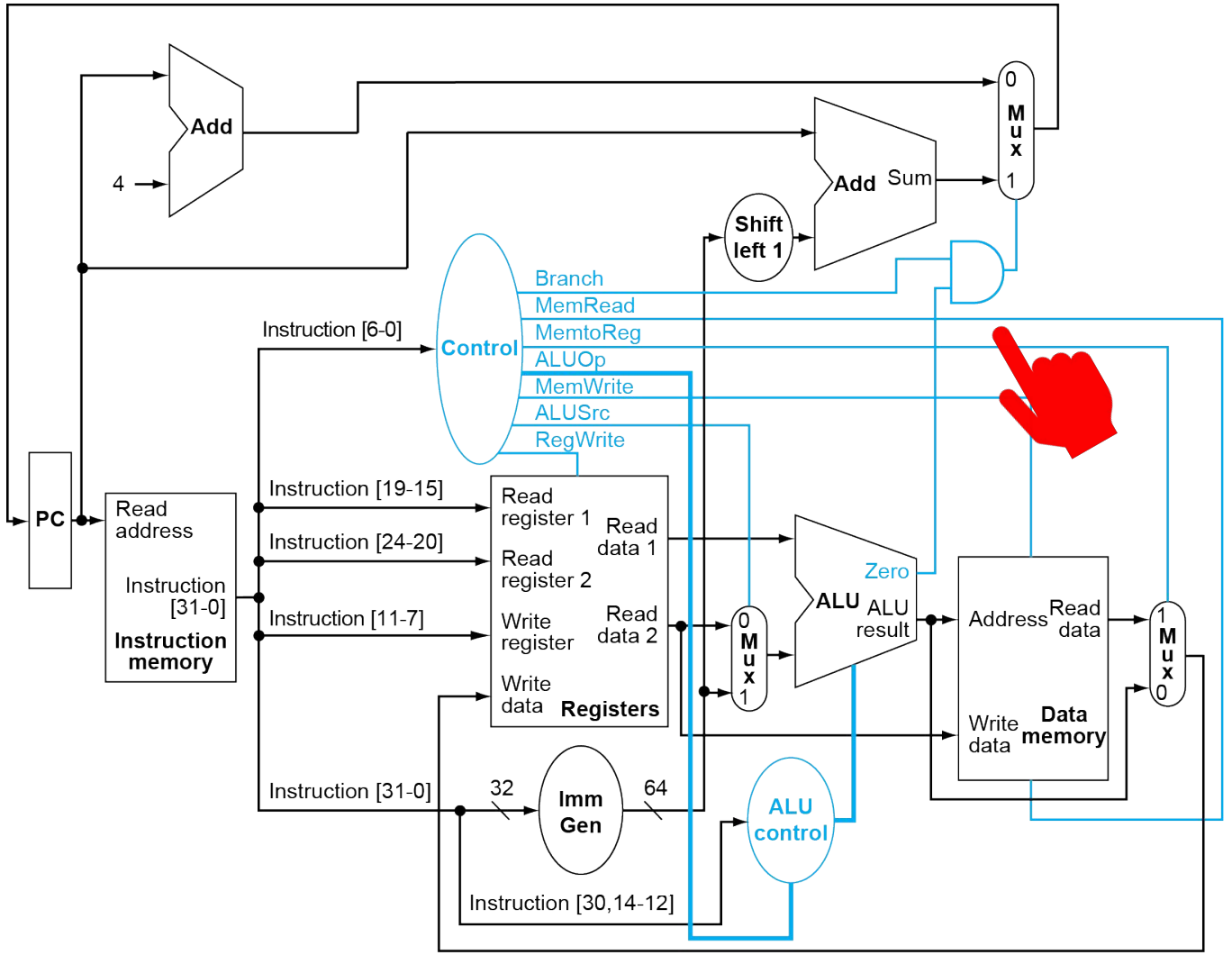


Instruction opcode	ALUOp
ld	00
sd	00
beq	01
R-type	10
R-type	10
R-type	10
R-type	10

Funct7 field	Funct3 field	Desired ALU action	ALU control input
XXXXXXX	XXX	add	0010
XXXXXXX	XXX	add	0010
XXXXXXX	XXX	subtract	0110
0000000	000	add	0010
0100000	000	subtract	0110
0000000	111	AND	0000
0000000	110	OR	0001

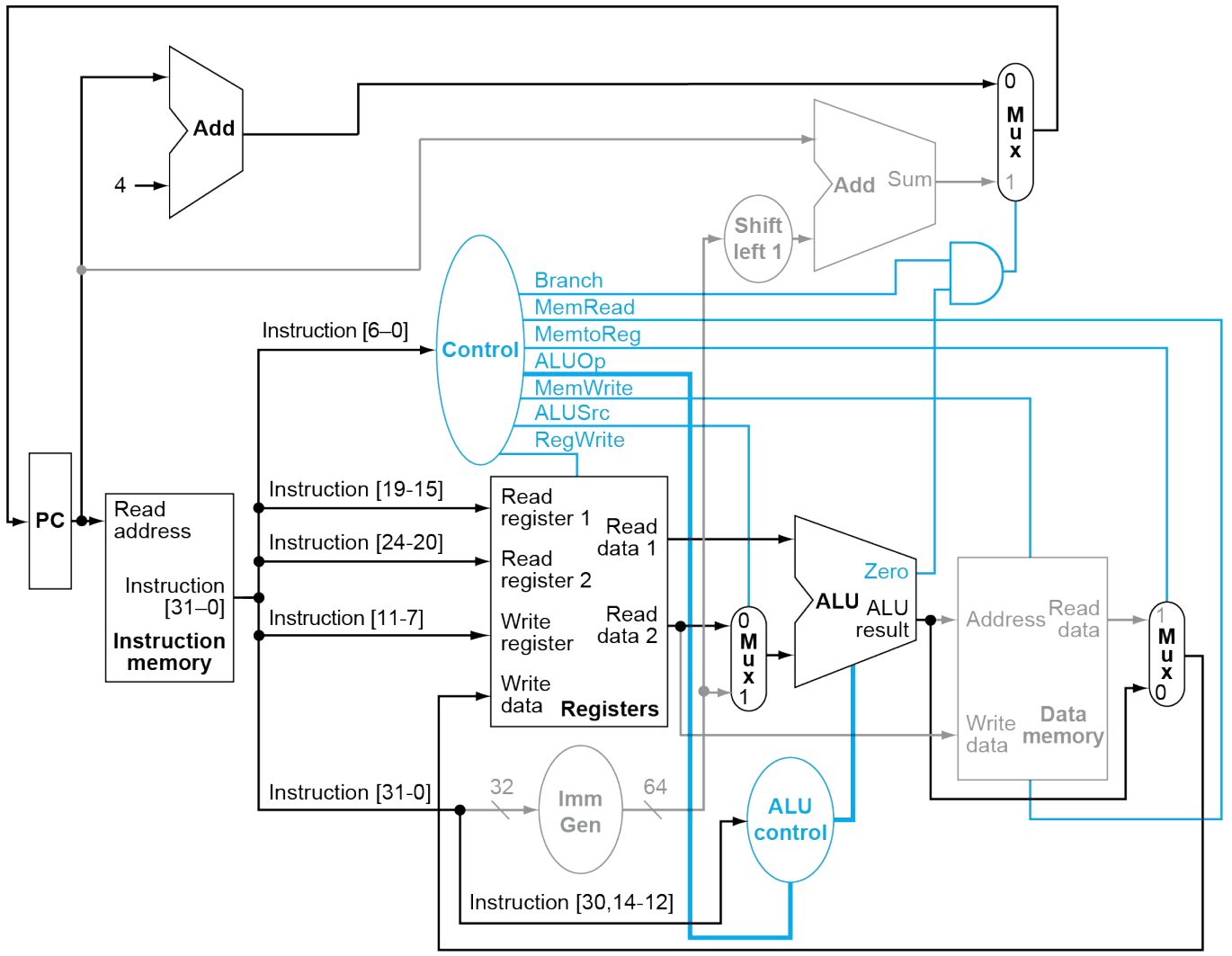
For subtract we already have 01 thus:
00 → add, 01→ subtract 10→ check rtype (except add/sub) 11→ check i-type

*Images were taken from Hennessy Patterson Book [1].


Add
4
Add
Sum
0
M
u
x
1
Shift
left 1
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [6-0]
PC
Read
address
Instruction
[31-0]
Instruction
memory
Instruction [19-15]
Instruction [24-20]
Instruction [11-7]
Read
register 1
Read
register 2
Write
register
Write
data
Read
data 1
Read
data 2
Registers
Zero
ALU
ALU
result
Address
Read
data
Write
data
Data
memory
1
M
u
x
0
Instruction [31-0]
32
Imm
Gen
64
ALU
control
Instruction [30,14-12]

Dataflow

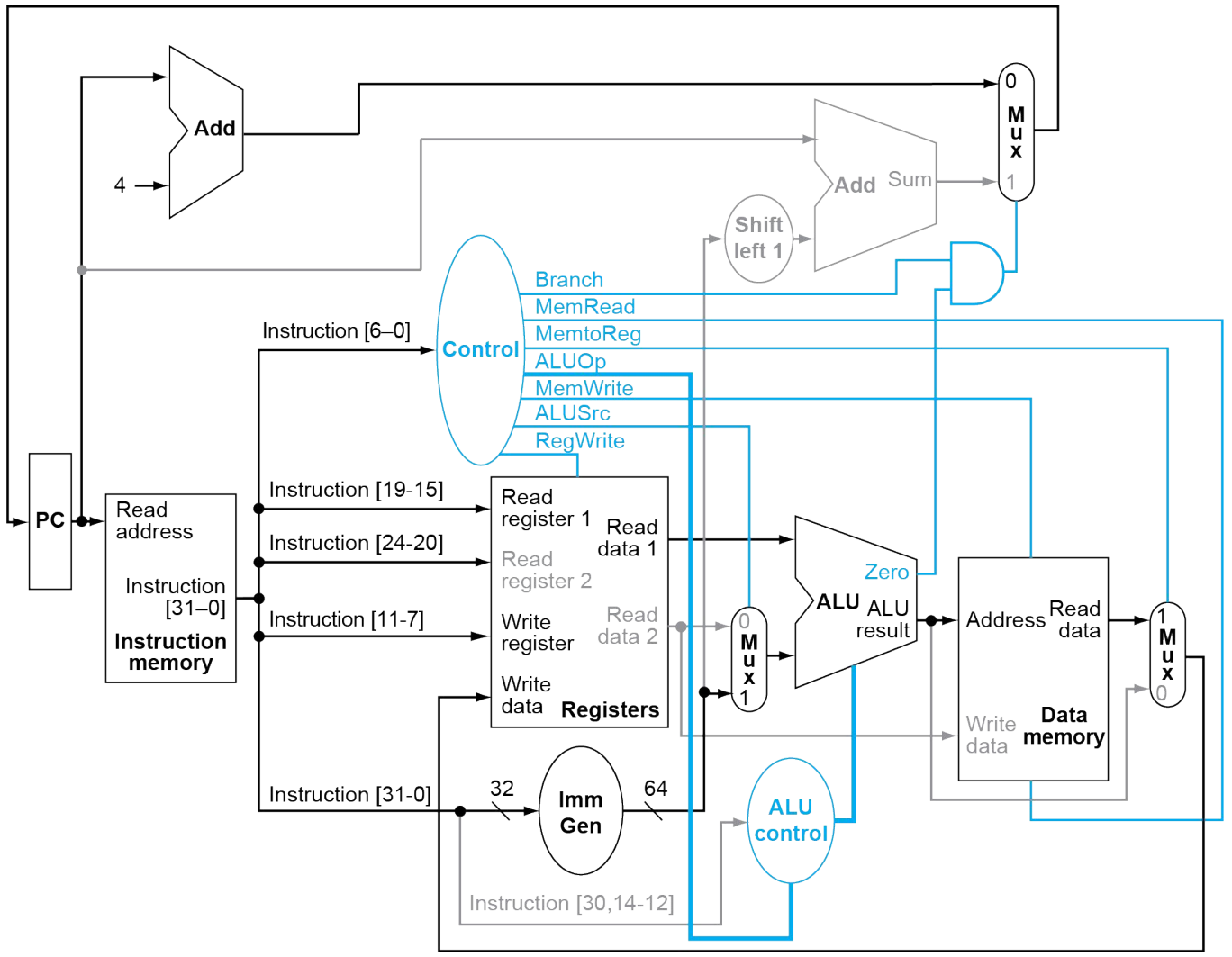
R-type



*Images were taken from Hennessy Patterson Book [1].

Dataflow

LW



*Images were taken from Hennessy Patterson Book [1].

Recap

Participation

- Link: https://gavel-for-nader.web.app/lo
  or Scan the QR code.

- Use this password: [redacted]

Summary

Nader Sehatbakhsh <nsehat@ee.ucla.edu>

End of Presentation

Acknowledgement

- This course is partly inspired by the following courses created by my colleagues:
  - CS152, Krste Asanovic (UCB)
  - 18-447, James C. Hoe (CMU)
  - CSE141, Steven Swanson (UCSD)
  - CIS 501, Joe Devietti (Upenn)
  - CS4290, Tom Conte (Goergia Tech)
  - 252-0028-00L, Onur Mutlu (ETH)