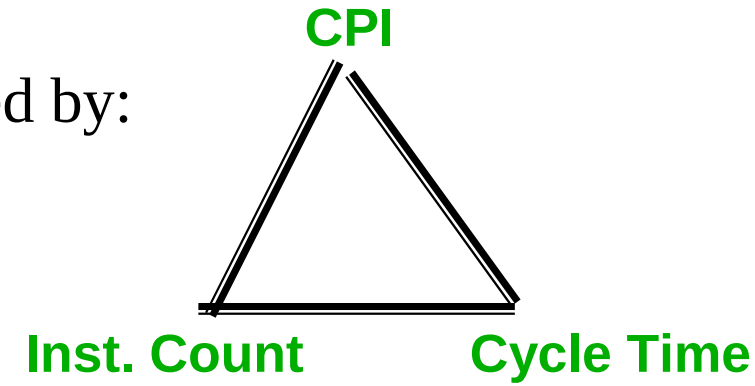
Chapter 4 The processor: Datapath & Control

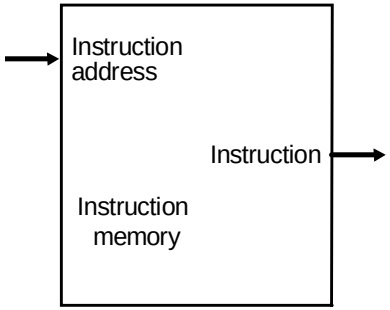
- Two implementations of RISC-V ISA
  - Single cycle implementation
    - Datapath & Control
  - Pipeline implementation
    - Datapath & Control
    - Data dependence/Hazard
- Consider only the core of RISC-V inst. Set
  - memory-reference inst. ld, sd
  - ALU inst. add, sub, and, or -- R-type
  - branch inst. beq, jal

The Performance Perspective

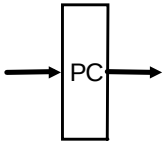


- Performance of a machine is determined by:
  - Instruction count
  - cycle time
  - CPI
- **Processor design (datapath and control) will affect:**
  - **cycle time**
  - **CPI**
- Single cycle processor:
  - A simple start to understand more complicated pipeline implementation

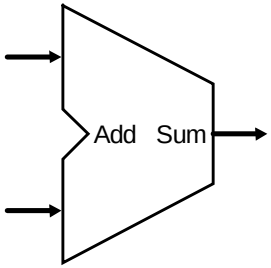
Functional Units for Implementing instructions



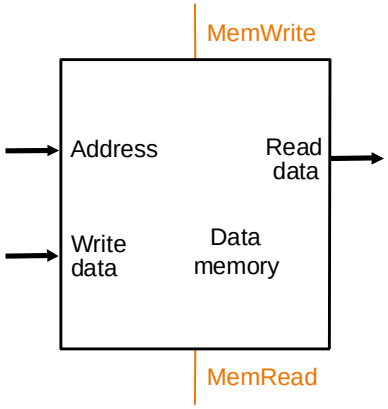
a. Instruction memory



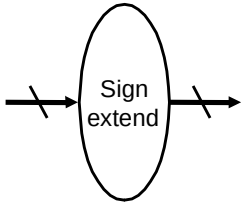
b. Program counter



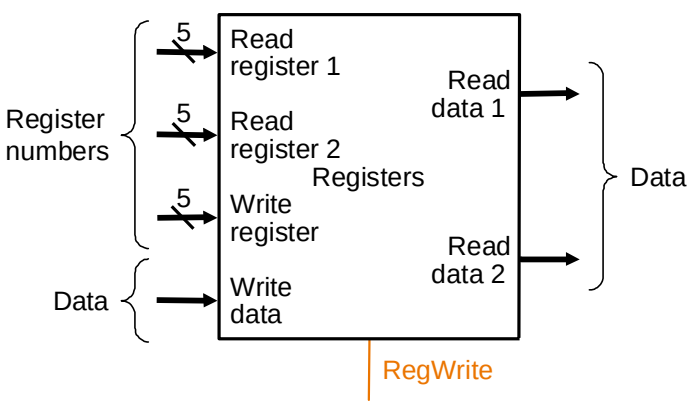
c. Adder



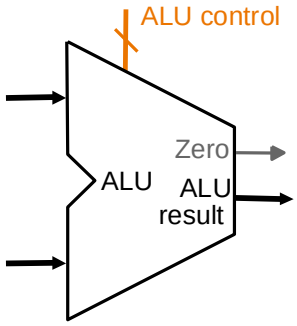
a. Data memory unit



b. Sign-extension unit



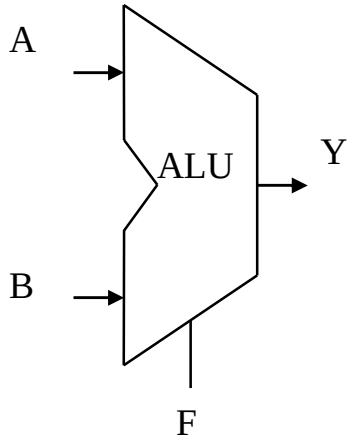
a. Registers



b. ALU

RISC-V ALU Functions

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract



Zero flag:

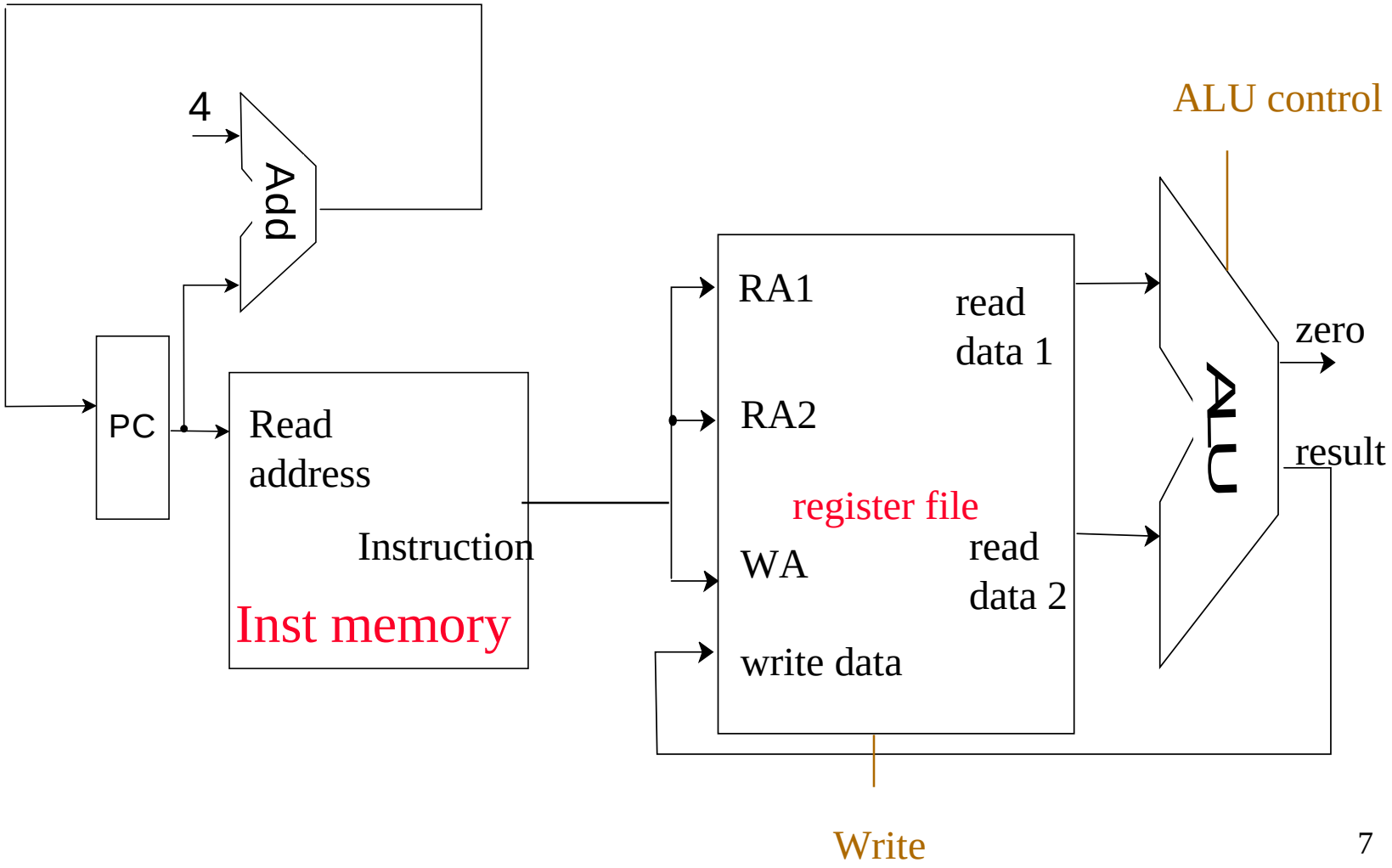
Basic Idea of Datapath Design

- Design a datapath for each type of instruction
  - R
  - ld
  - sd
    - Combine ld and sd
  - beq
- Combine all
- Immediate Generation Unit

Datapath for R-type

- Example: add x3, x4, x5
- Basic steps
  - fetch instruction
  - select registers (rs, rt),
  - ALU operation on two data, need ALU
  - write back registers

Single-Cycle: Datapath for R-type



Datapath for Load Word

- Basic steps: example: `ld x3, 64(x4)`
  - fetch instruction
  - select a register (rs1)
  - calculate address, need ALU
  - access memory (read memory)
  - write register file (rd)

Datapath for Store Word

- Basic steps: example: `sd x3, 64(x4)`
  - fetch instruction
  - select two registers (rs1, rs2)
  - calculate address, need ALU
  - access memory (write memory)

Combining datapaths of `ld & sd`

- Memory address
  - `Ld`: in fcunct7 and rs2
  - `sw`: in fcunct7 and rd
- Register write control signal.

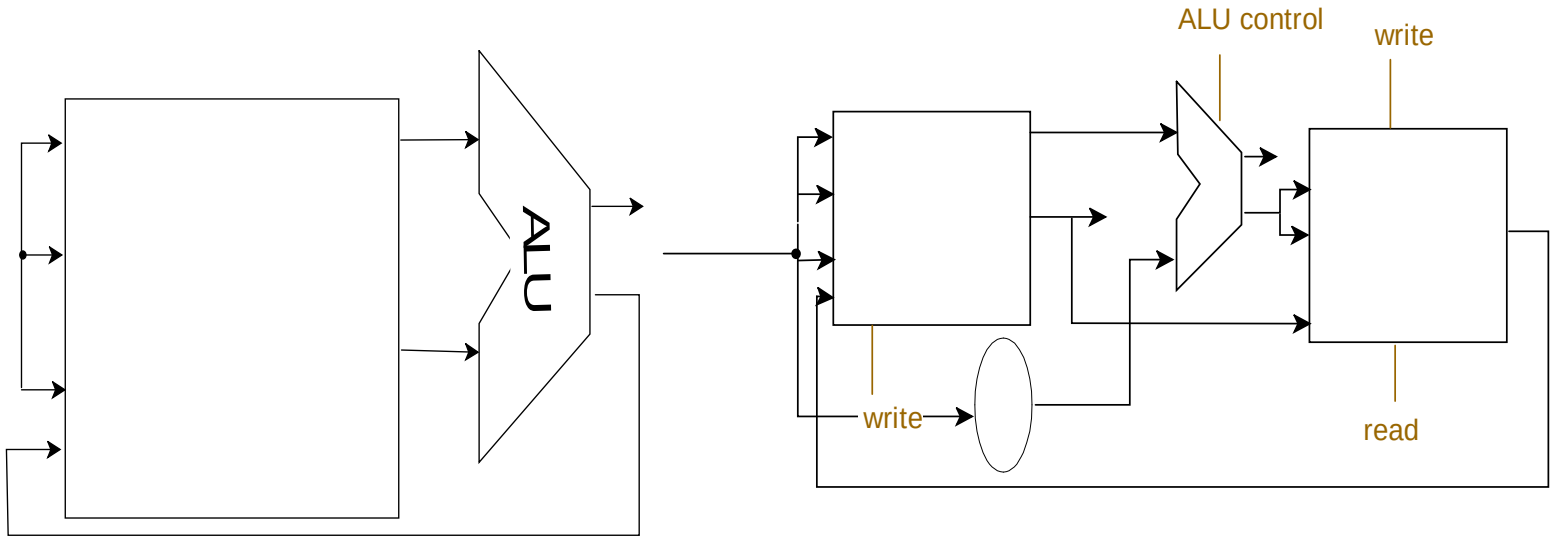
Datapath for `beq`

- Basic steps: beq x1, x2, offset
  - fetch the instruction
  - select registers
  - test condition, calculate branch address (need additional ALU)

- ALU for branch condition
- Adder for branch address
- Zero: control logic to decide if branch.

Creating a Single Datapath for All Inst.

- Combine datapaths for R-type, memory inst. and branch inst.
  - using multiplexers
  - without duplicating common functional units.
- As an example, combine R-type and memory type datapaths
- Final goal: combine all types of datapaths



R-type data path

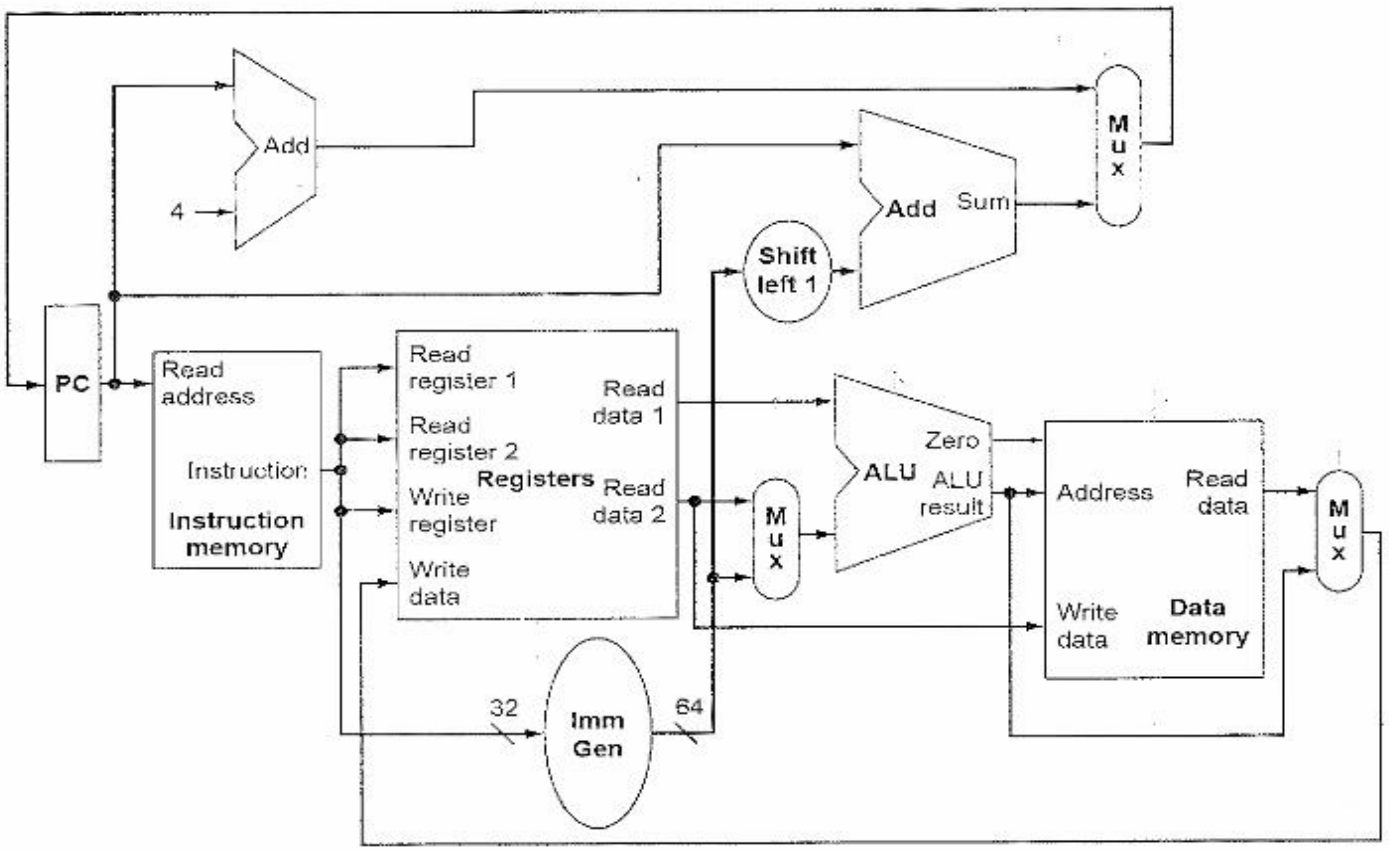
Memory type datapath

Key Differences:

2nd input to ALU

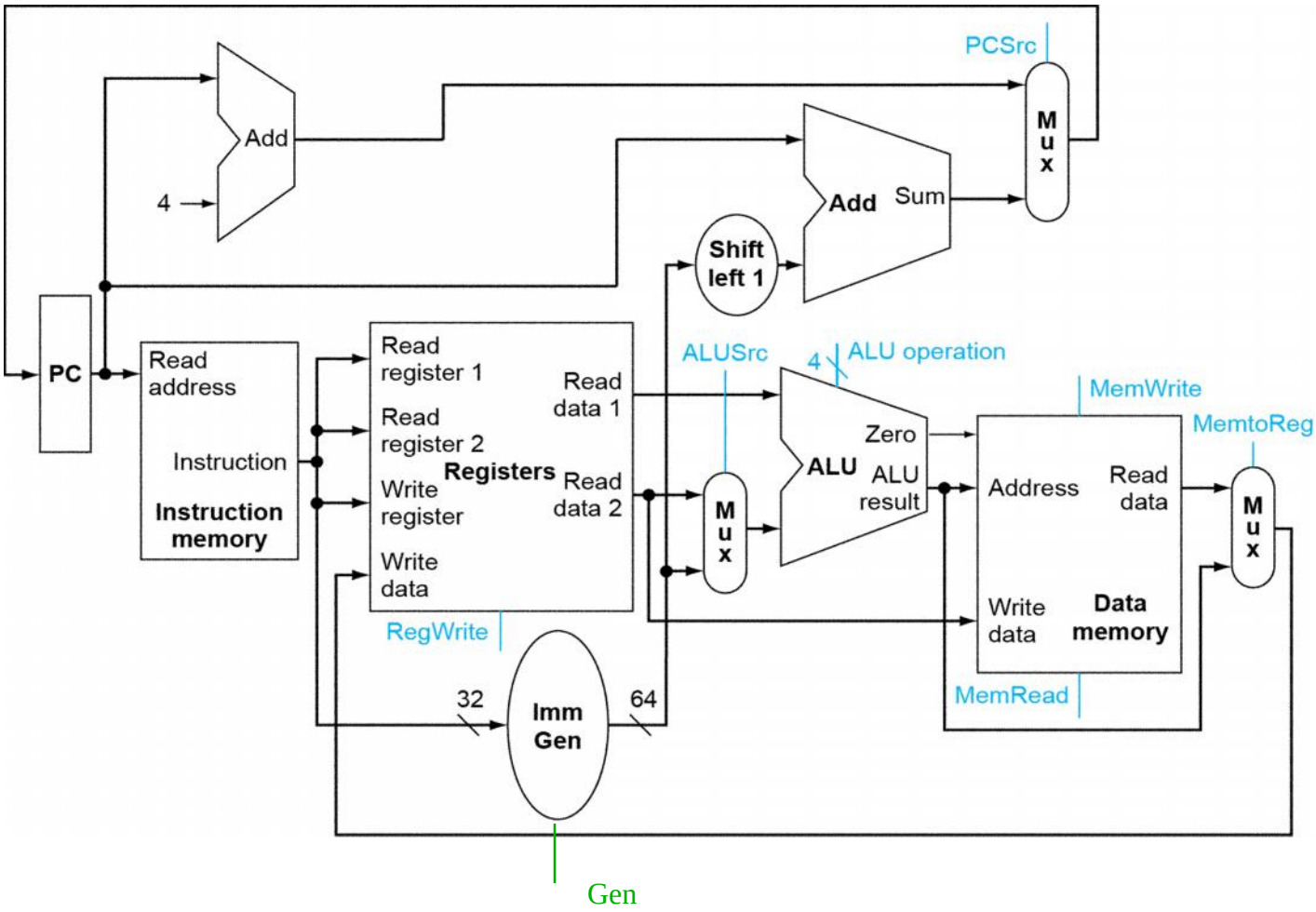
Value written into register file

Simple Datapath for Single Cycle



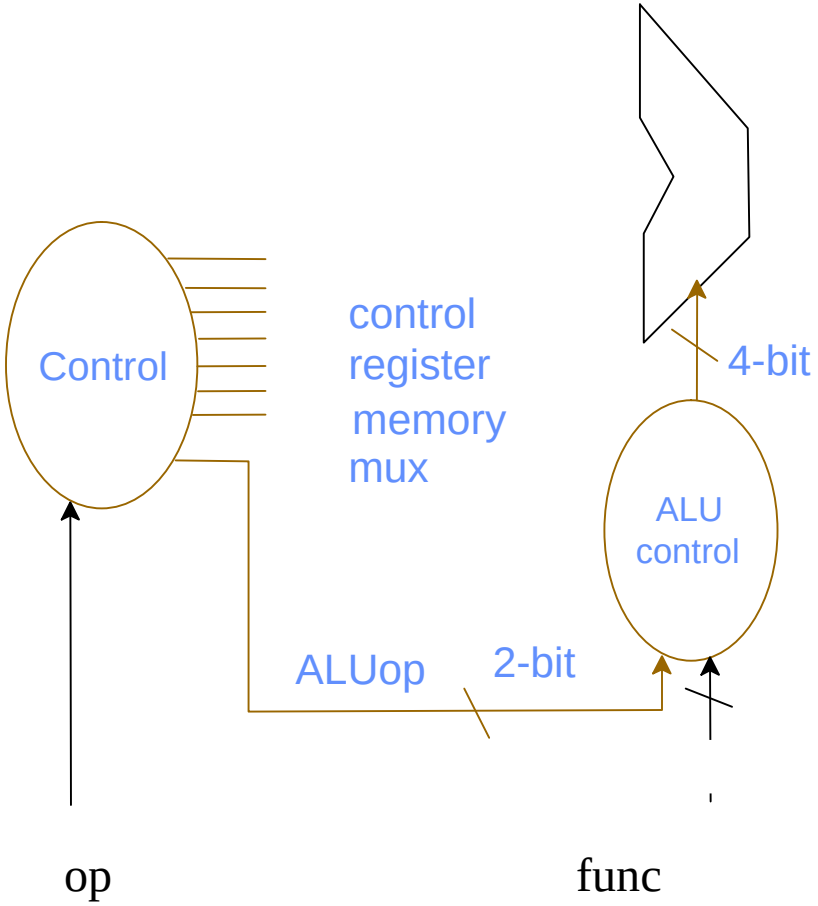
Can execute basic instructions in a single clock cycle

No resource can be used more than once during a single cycle

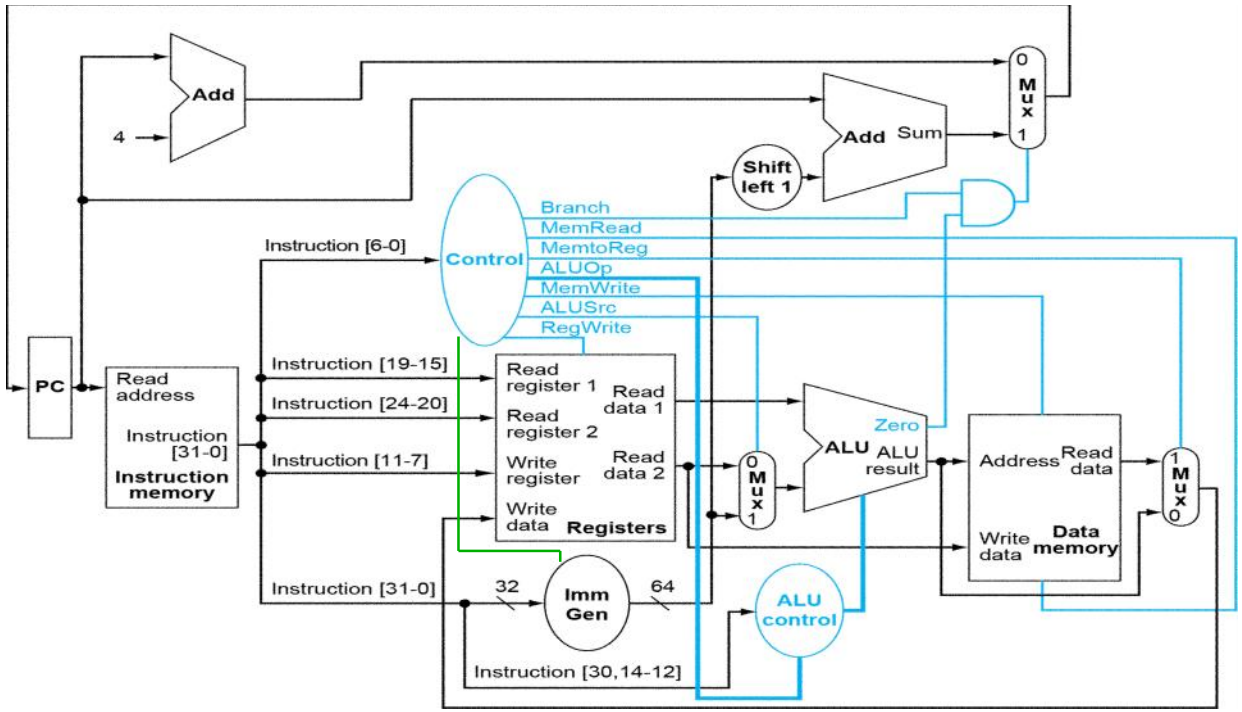

PCSrc
Add
4
M
u
x
Add
Sum
Shift
left 1
PC
Read
address
Instruction
Instruction
memory
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
RegWrite
ALUSrc
4
ALU operation
M
u
x
ALU
Zero
ALU
result
MemWrite
MemtoReg
Address
Read
data
Write
data
Data
memory
MemRead
M
u
x
32
Imm
Gen
64
Gen

Single-Cycle: Control

- Main control:
  - input: 7-bit from opcode
  - output: 9+ control lines
- ALU control:
  - input: ALUop + 4-bit (funct7 + funct3)
  - output: 4 lines
  - for non R type, ALU control depends on only ALUop
- Control for imm gen ?



Control



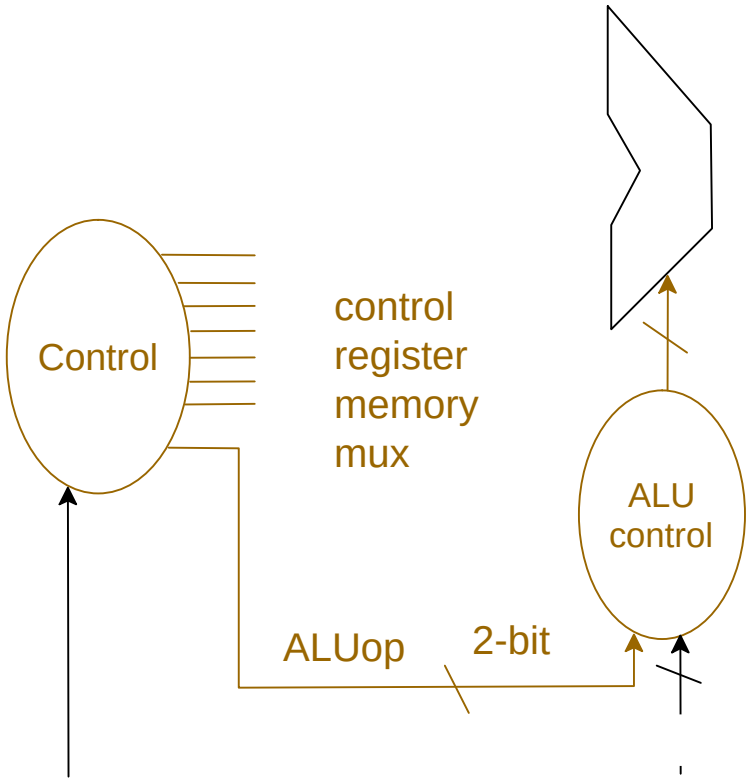
Instruction	RegWrt	Memto-Reg	Brch	Mem Read	Mem Write	ALUSrc	ALUOp1	ALUp0	Gen
R-format									
ld									
sd									
beq									

Main Control Unit -- Definitions of Control Signals

Signal Name	Effect when deasserted	Effect when asserted
MemRead	None	Data put on read dataoutput
MemWrite	None	Write data into memory
RegWrite	None	Write register
ALUSrc	2nd ALUinput from register file	2nd ALUinput from imm gen
PCSrc	PC = PC+4	PC = branch address
MemtoReg	result of ALU is sent	data in memory is sent

PCsrc = branch AND zero

Single cycle control



ALU Control

- ALU used for
  - Load/Store: F = add
  - Branch: F = subtract
  - R-type: F depends on opcode

ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract

ALU Control

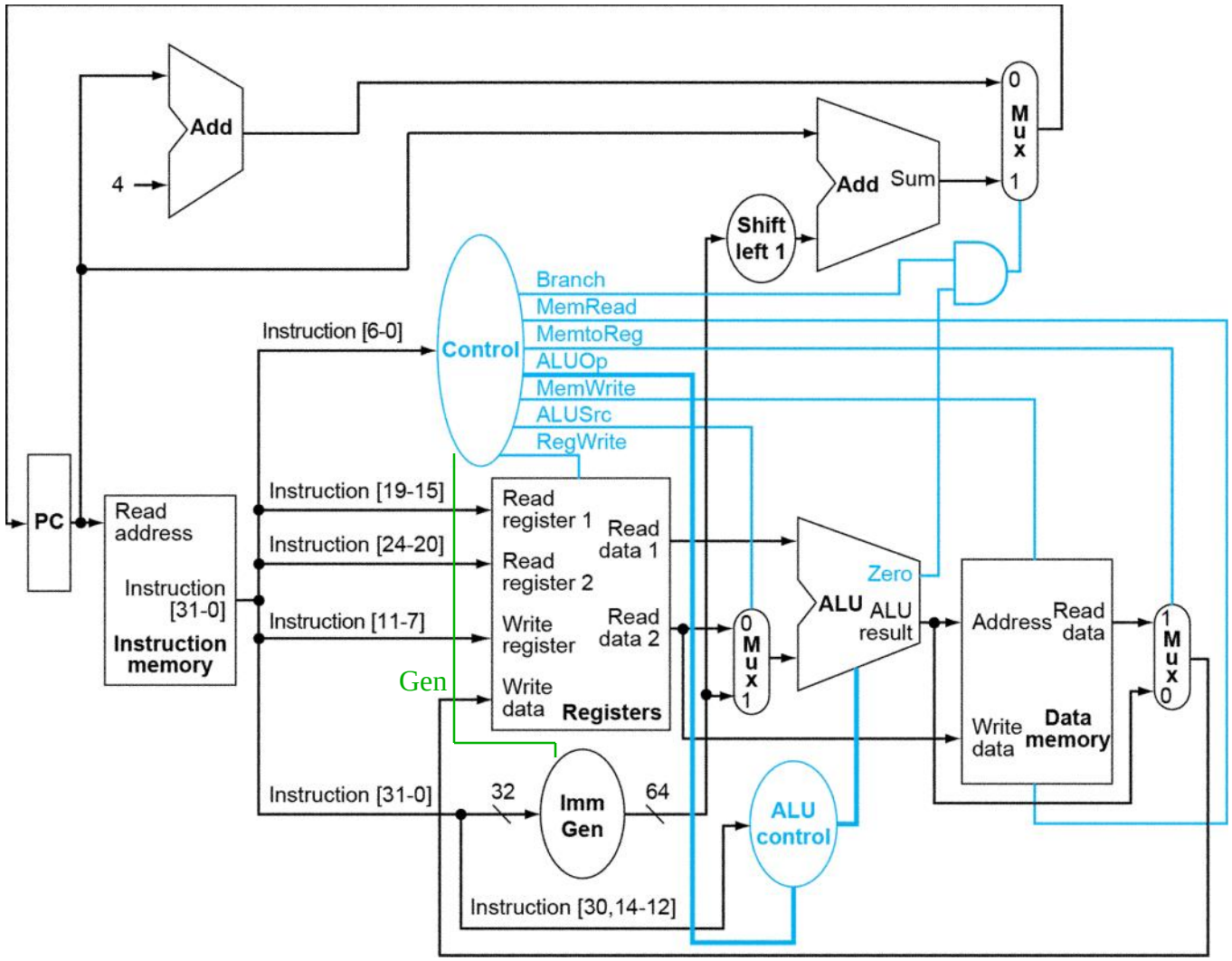
- Assume 2-bit ALUOp derived from opcode
  - Combinational logic derives ALU control

opcode	ALUOp	Operation	Fuct7+funct3	ALU action	ALU control
ld	00	load register	XXXXXXX XXX	add	0010
sd	00	store register	XXXXXXX XXX	add	0010
beq	01	branch on equal	XXXXXXX XXX	subtract	0110
R-type	10	add	0100000 000	add	0010
		subtract	0100010 000	subtract	0110
		AND	0100100 111	AND	0000
		OR	0100101 110	OR	0001

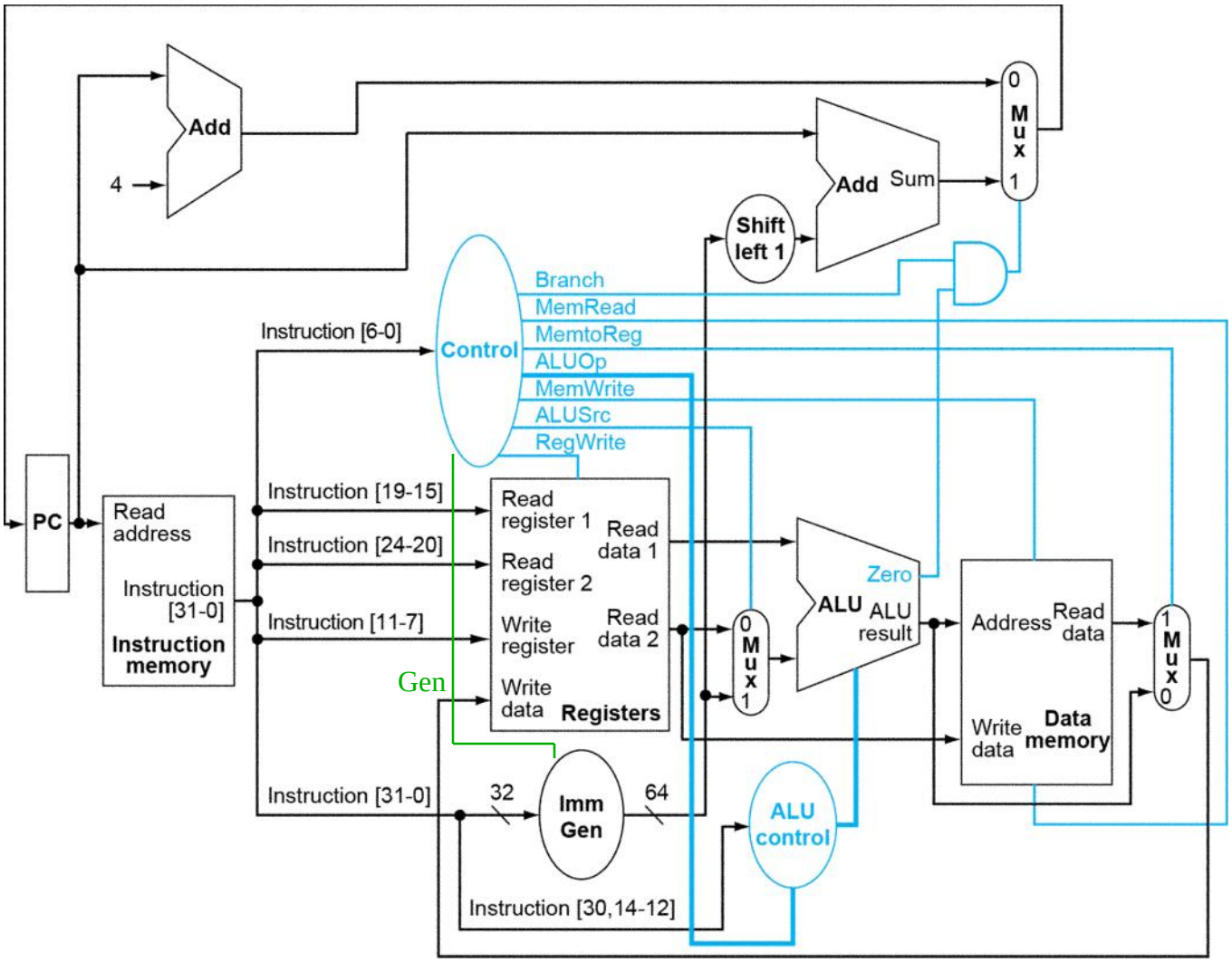
ALU Control Truth Table

ALUOp	Inst[30][14:12]	ALU control output
00	X XXX	0010
00	X XXX	0010
01	X XXX	0110
10	1 000	0010
	1 000	0110
	1 111	0000
	1 110	0001

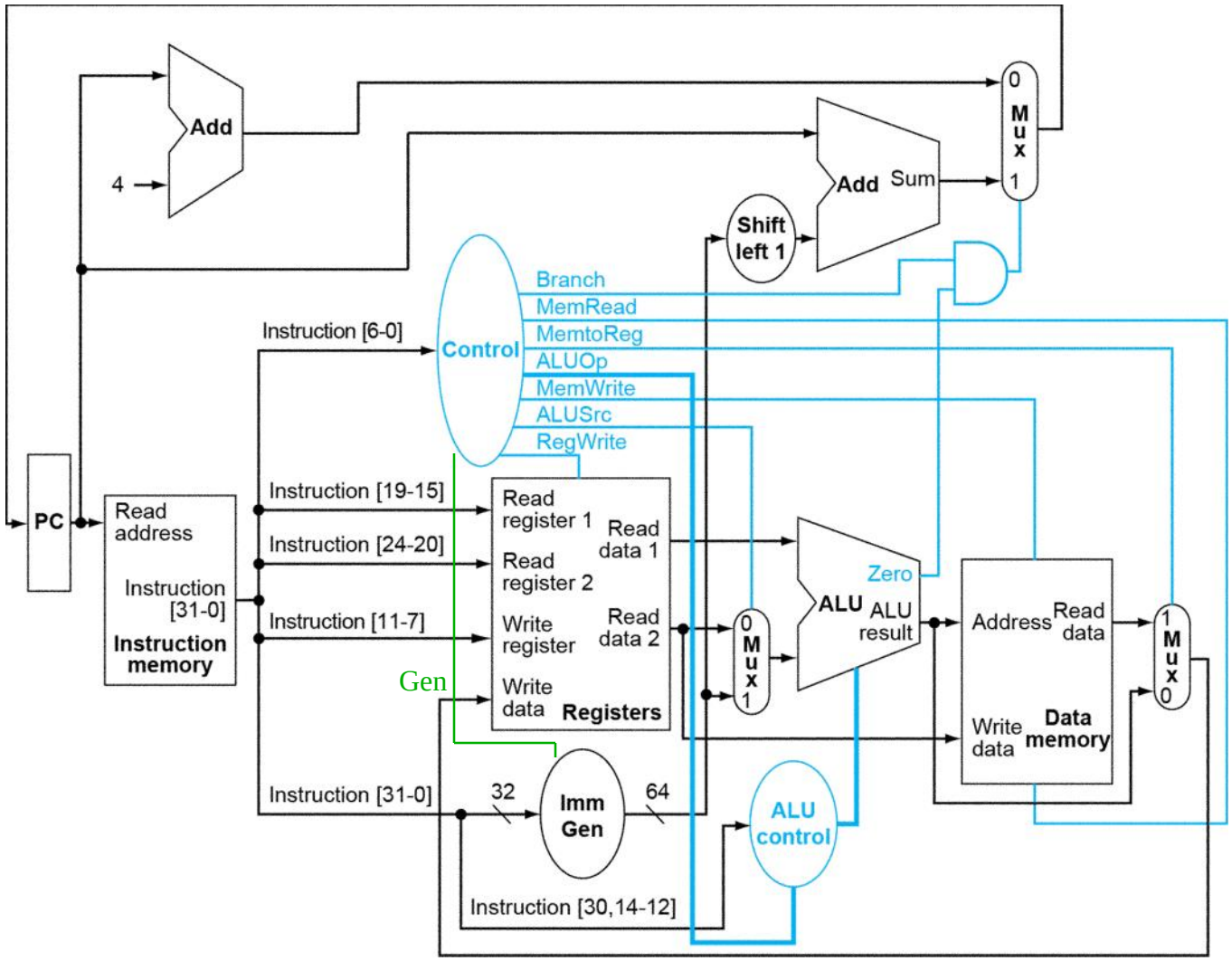
Datapath With Control



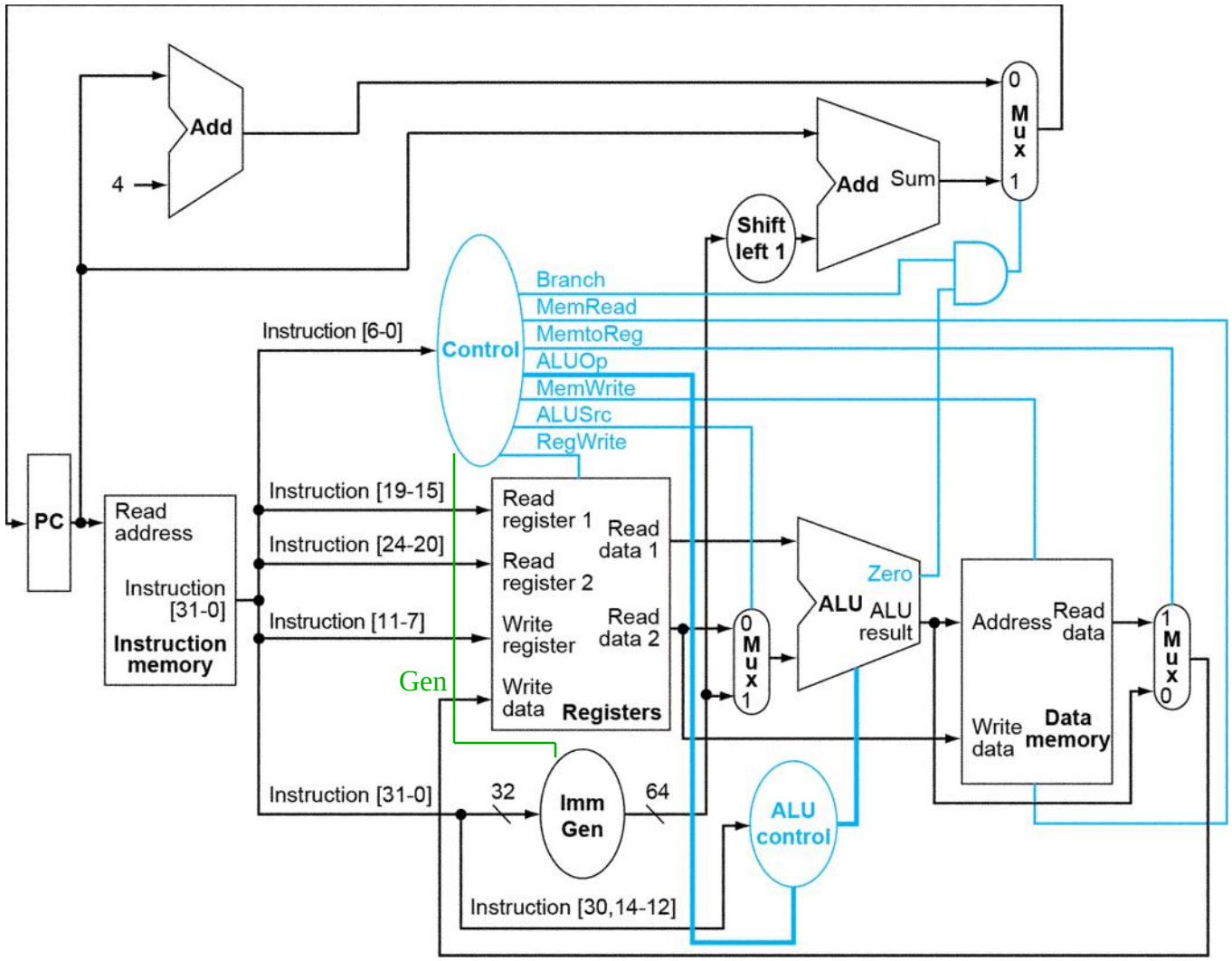
R-Type Instruction



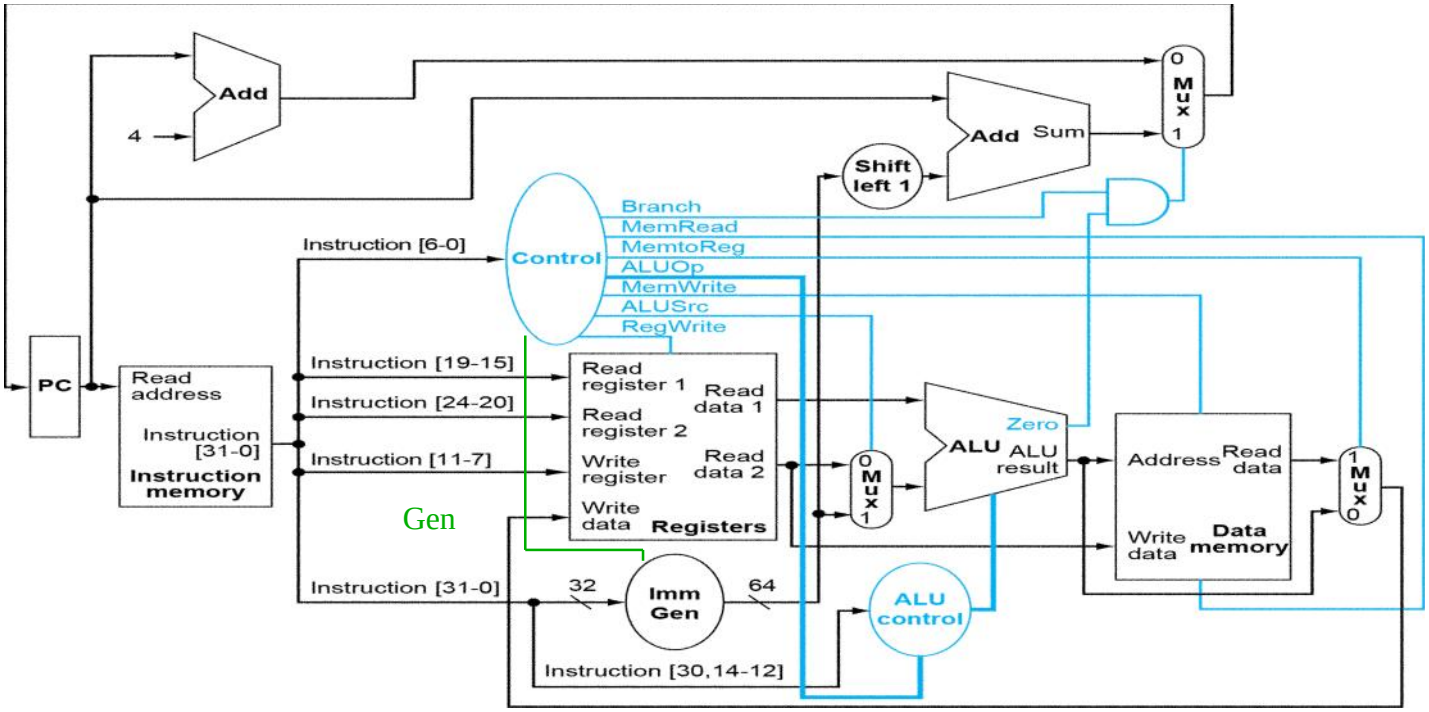
Load Instruction



BEQ Instruction



Add jal to Datapath



Instruction	RegWrt	Memto-Reg	Brch	Mem Read	Mem Write	ALUSr	ALUOp1	ALUp0	Gen
R-format									
ld									
sd									
beq									

Add jal to the datapath

- Operation 1: PC = jump address
  - Assume jump address is formed by imm gen
- Operation 2: x rd = PC

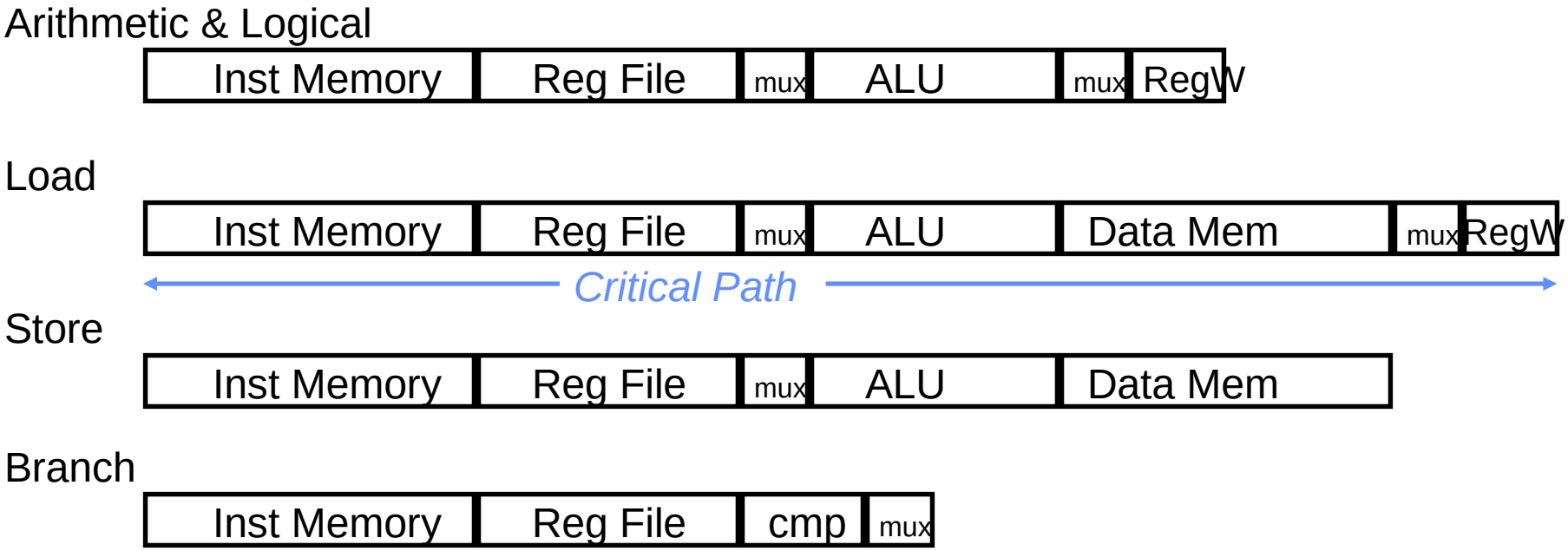
Discussion on imm gen

- Combinational circuit
- Input: the whole instruction[31:0]
- Four possible outputs
  - Immediate for ld
  - Immediate for sd
  - Immediate for beq
  - Immediate for jal

Single-Cycle: Summary

- How to construct datapath for R-type, ld/sd, beq, jal
- How to construct the control
  - ALU control
  - main control
  - imm gen
- How the datapath + control to execute instruction

Execution (cycle) Time Analysis



- Cycle time >= the critical path length
- Long Cycle Time
  - All instructions take as much time as the slowest

Exercise: Can MemtoReg be eliminated and replaced by MemRead?

	Brnch	MemRd	MemtoRg	ALUop	MemWrt	ALUSrc	RegWrt	gen
ld	0	1	1	00	0	1	1	
sd	0	0	x	00	1	1	0	
R	0	0	0	10	0	0	1	
beg	1	0	x	01	0	0	0	
Jal	0	0	x	xx	0	x	0	
	B							

Exercise: What would be the cycle time for the following cases?

- Assume
  - ALU delay=2ns,
  - Adder (PC+4) delay=x ns,
  - Adder (branch address) delay = y ns
- X=3, y=3
- x=5, y=5
- x=1, y=8