Chapter 4 - part I

Datapath and Control Single-cycle

Tien-Fu Chen

Dept. of Computer Science

National Chiao Tung Univ.

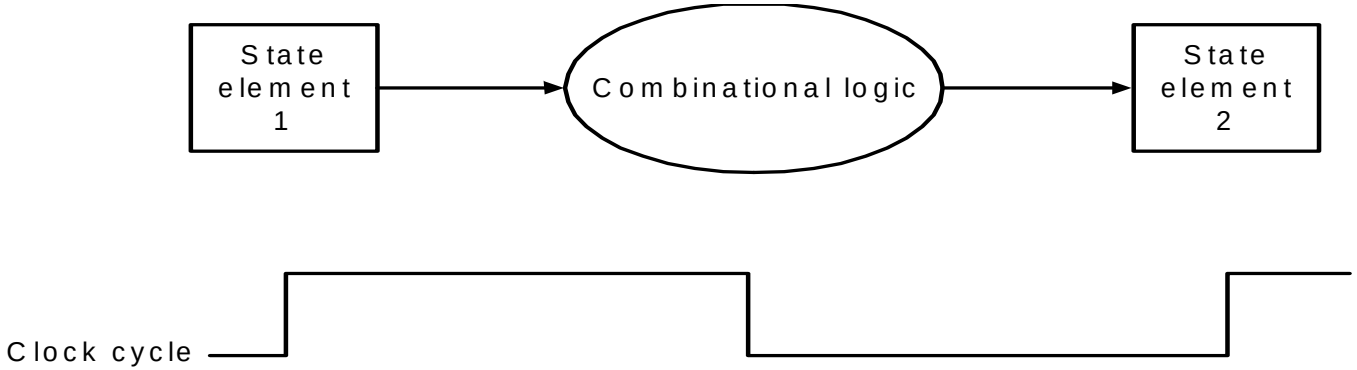
Material source:
COD RISC-V slides

Chap 4 overview

- CPU performance factors
  - Instruction count: Determined by ISA and compiler
  - CPI and Cycle time: Determined by CPU hardware
- We will examine two RISC-V implementations
  - **Chap 4-part1**: A simplified RISC-V
    - Datapath & control
  - **Chap 4-part2**: A more realistic pipelined version
    - Datapath & control
  - **Chap 4-part3**: Advanced pipelined design
- Simple subset, shows most aspects
  - Memory reference: `ld`, `sd`
  - Arithmetic/logical: `add`, `sub`, `and`, `or`
  - Control transfer: `beq`

Recap Logic Design Basics

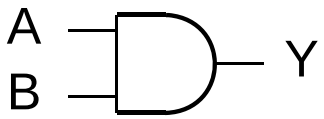
- Information encoded in binary
  - Low voltage = 0, High voltage = 1
  - One wire per bit
  - Multi-bit data encoded on multi-wire buses
- Combinational element
  - Operate on data
  - Output is a function of input
- State (sequential) elements
  - Store information



Combinational Elements

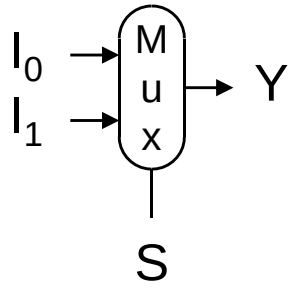
- AND-gate

  Y = A & B



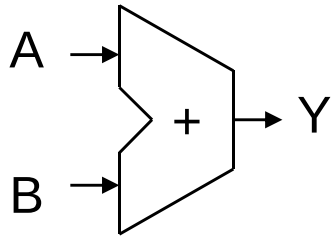
- Multiplexer

  $Y = S\ ?\ I_1 : I_0$



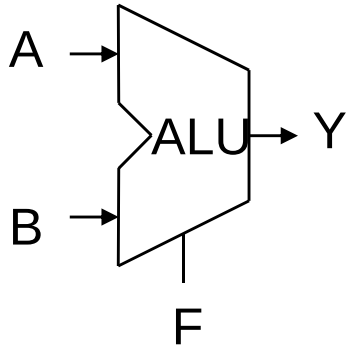
- Adder

  Y = A + B



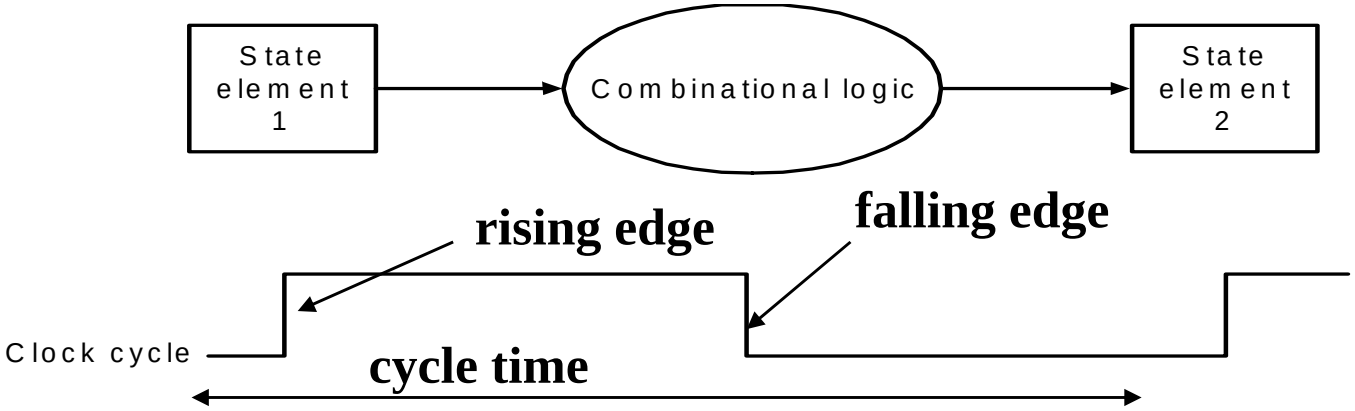
- Arithmetic/Logic Unit

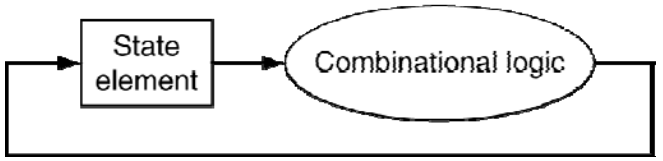
  Y = F(A, B)



Clocking Methodology

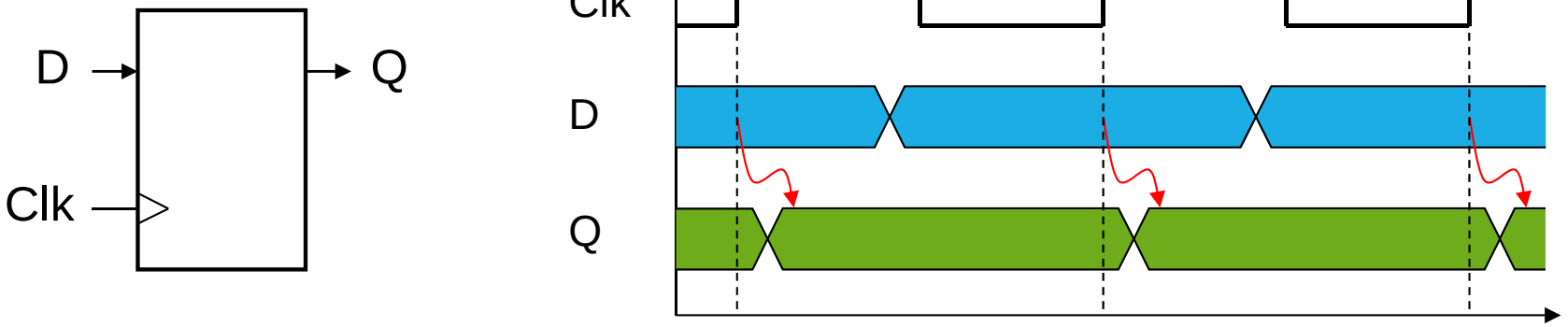
- Combinational logic transforms data during clock cycles
  - Between clock edges
  - Input from state elements, output to state element
  - Longest delay determines clock period





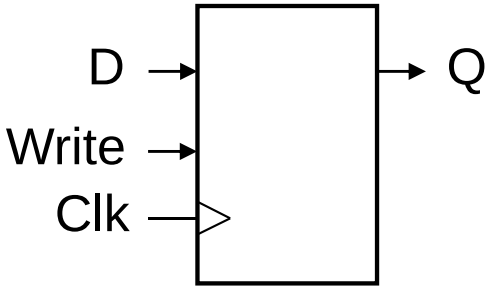
Sequential Elements

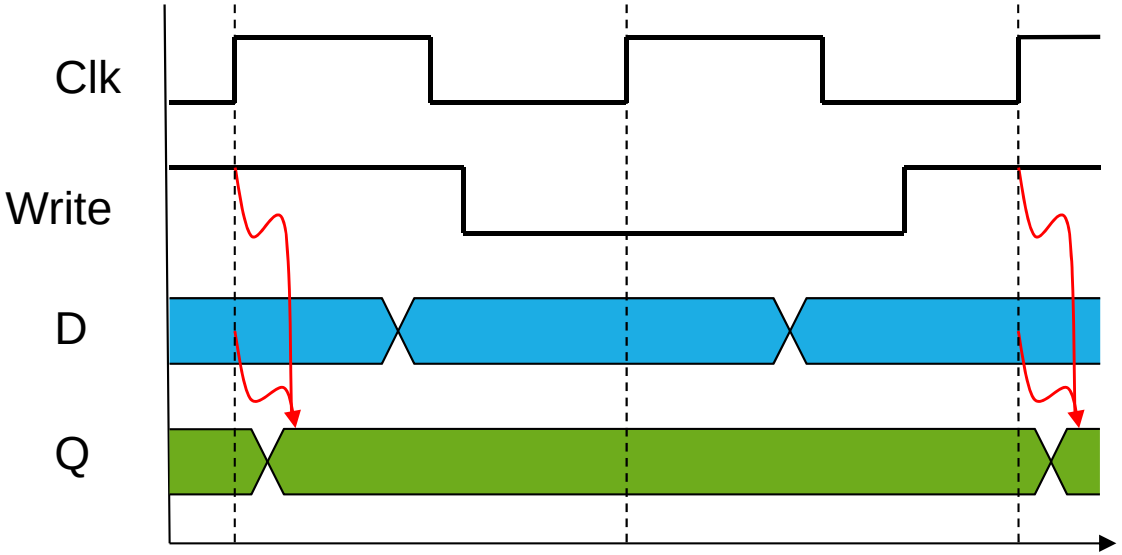
- Register: stores data in a circuit
  - Uses a clock signal to determine when to update the stored value
  - Edge-triggered: update when Clk changes from 0 to 1



Sequential Elements

- Register with write control
  - Only updates on clock edge when write control input is 1
  - Used when stored value is required later

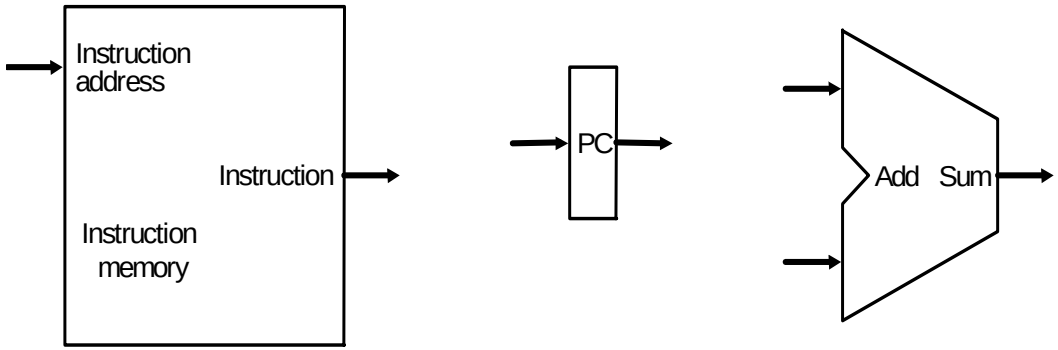




Let us start a simple RISC-V datapath

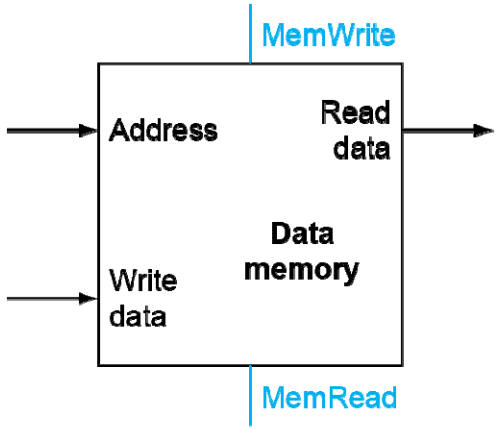
Basic components for Datapath

- Instruction Fetch



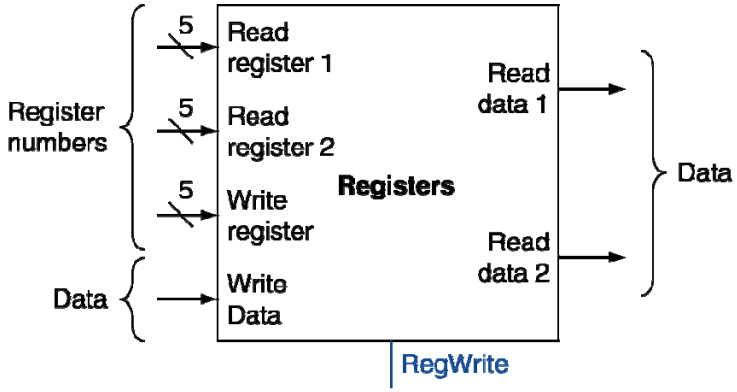
a. Instruction memory b. Program counter c. Adder

- Memory and sign extension

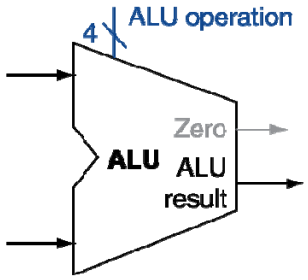


a. Data memory unit

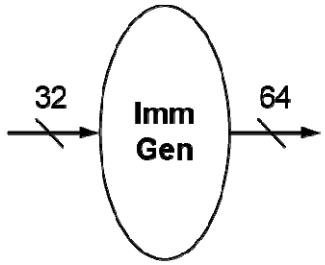
- datapath for R-type instructions



a. Registers

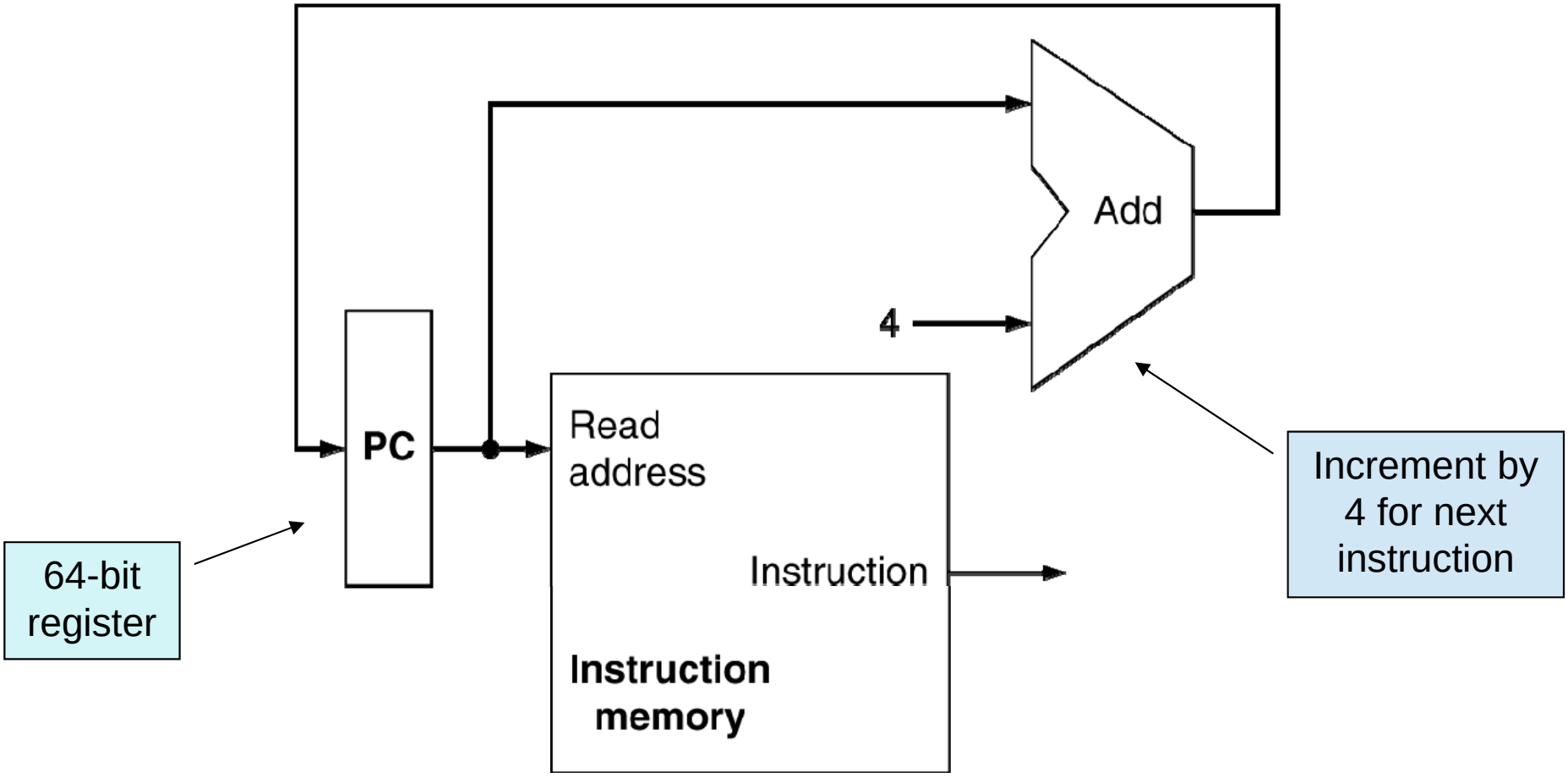


b. ALU

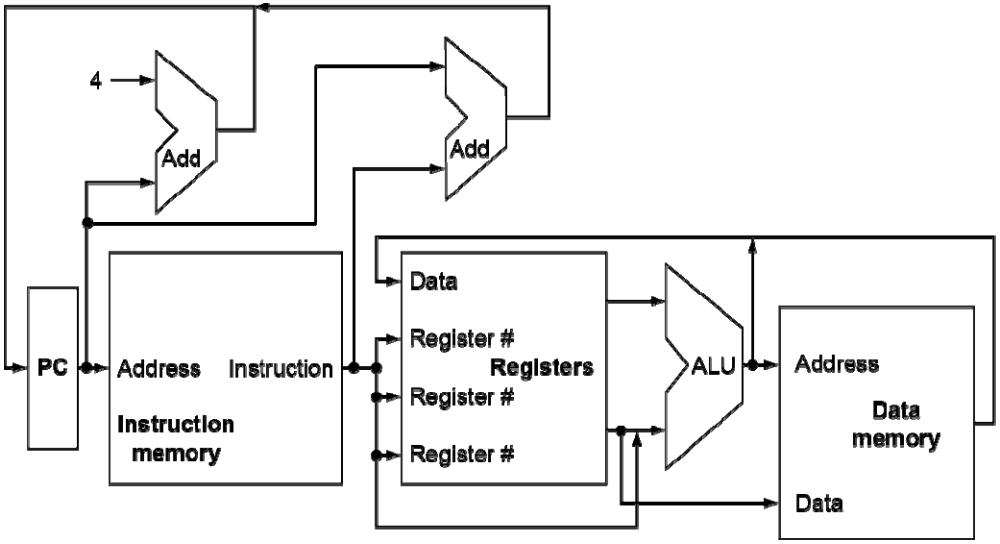


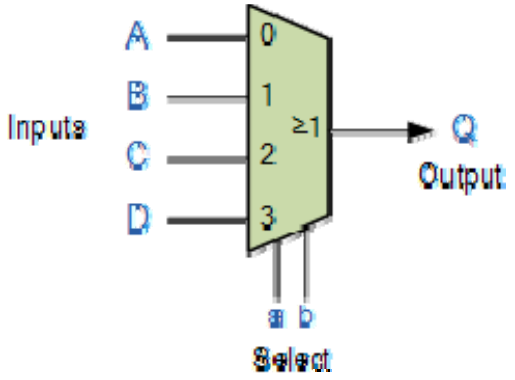
b. Immediate generation unit

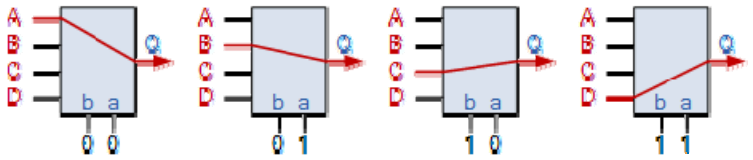
Instruction Fetch

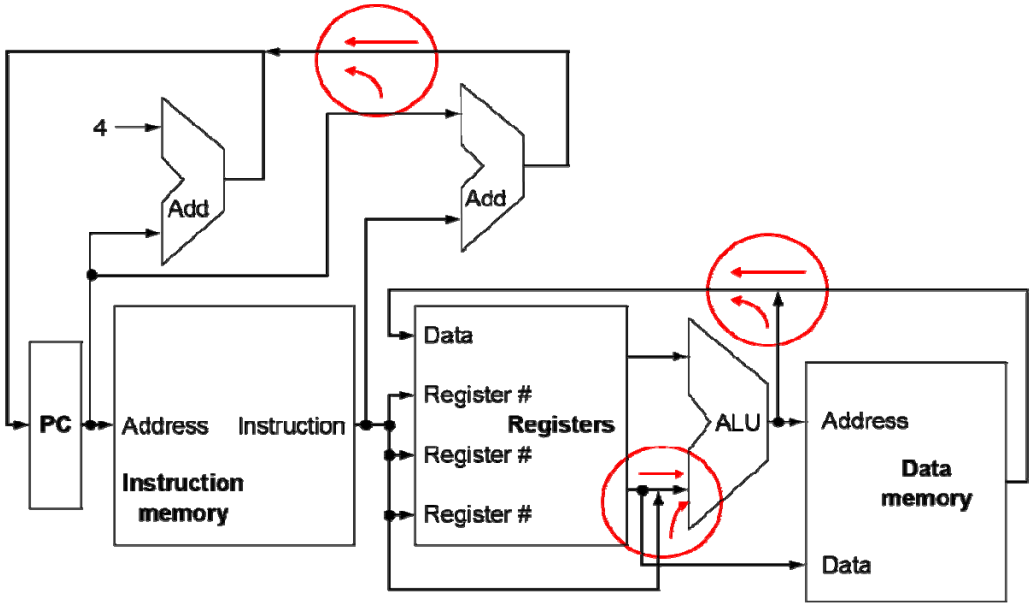


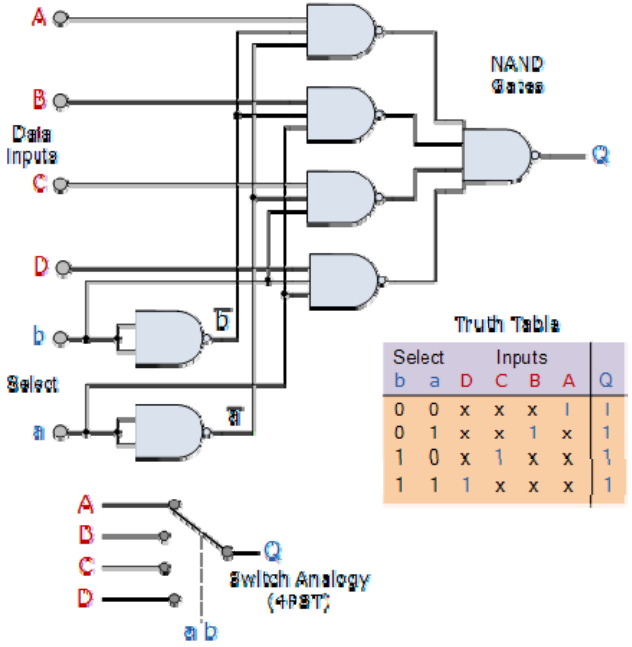
A Quick building up??







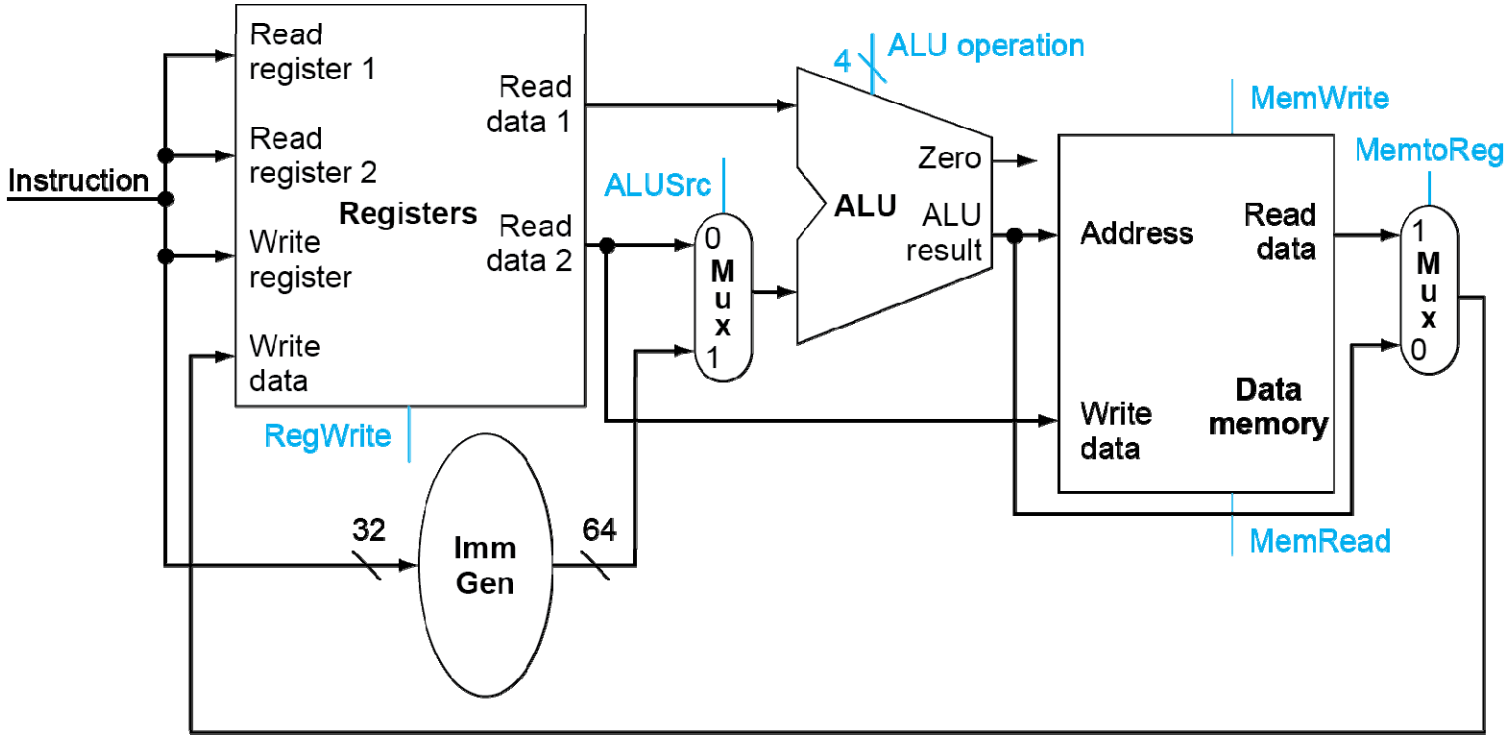




R-type: add instruction

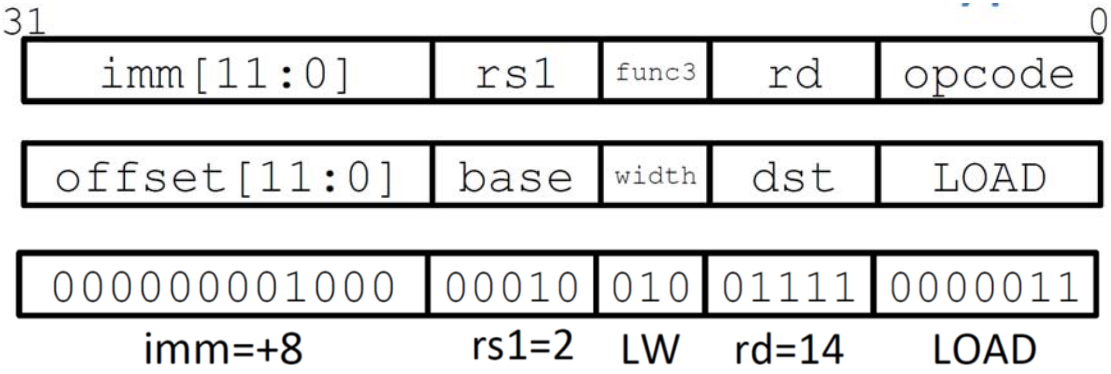
- add x9,x20,x21

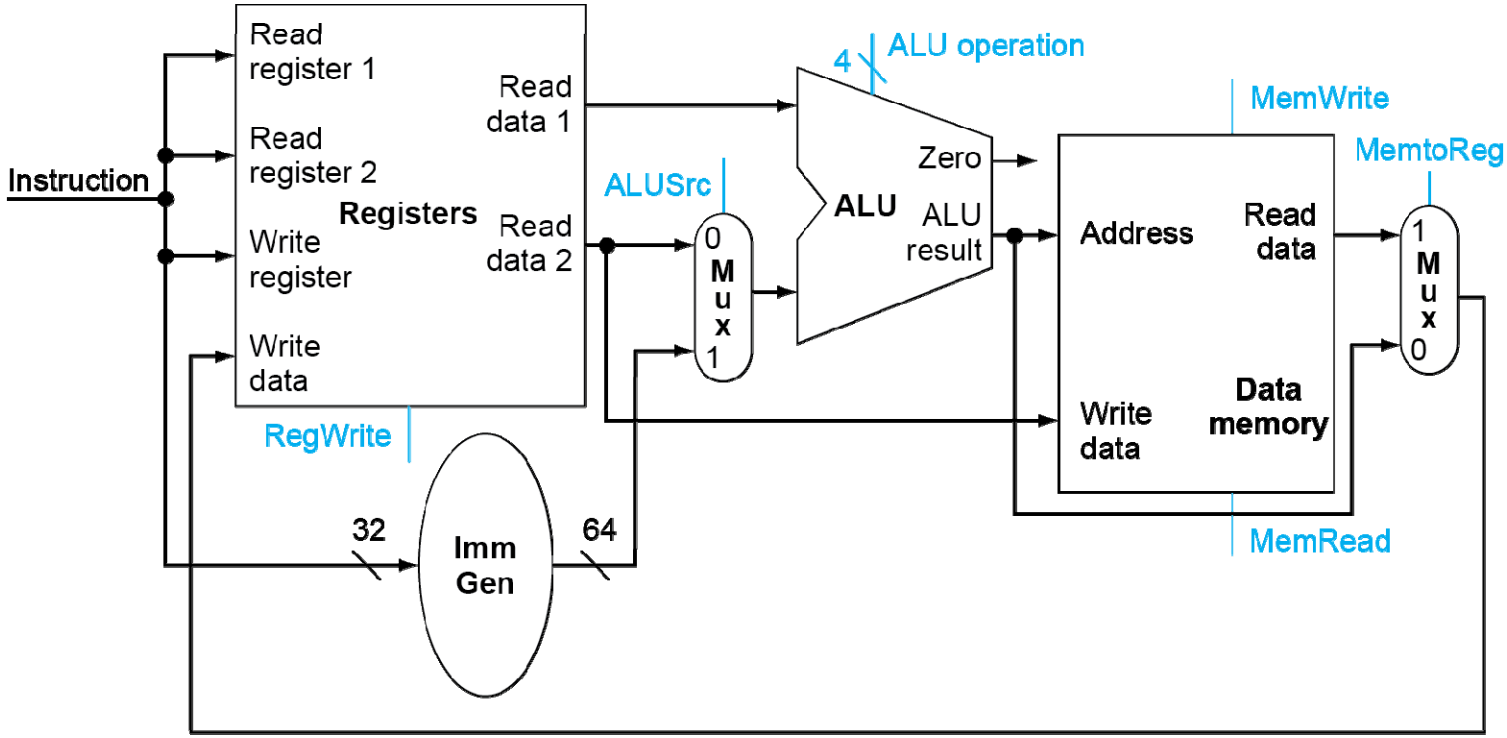
funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
0	21	20	0	9	51



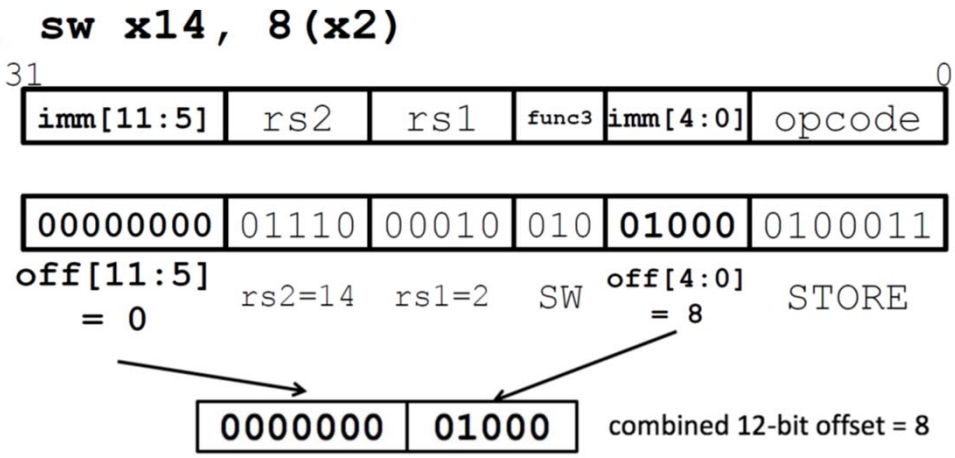
I-Format: load instruction

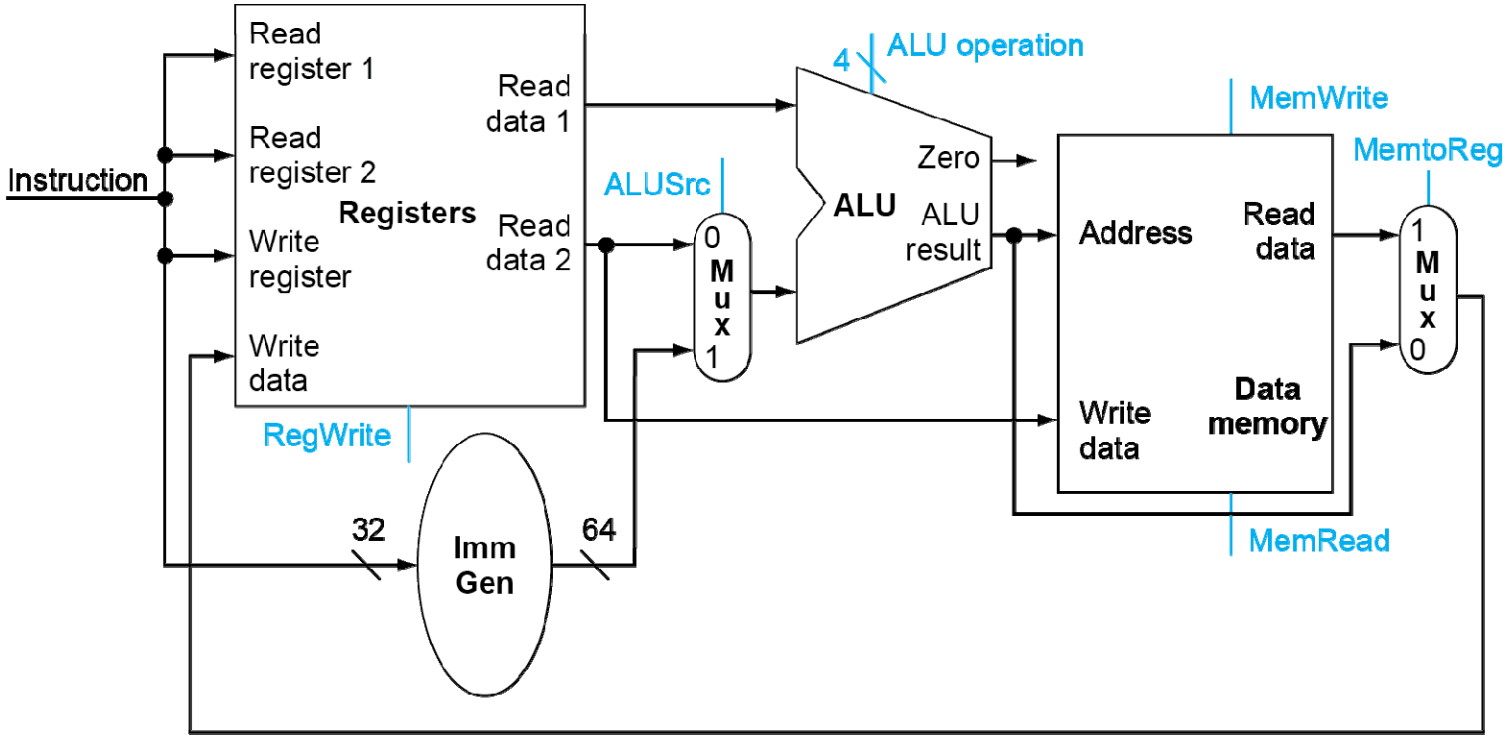
- lw x14, 8(x2)





S-Format: store instruction

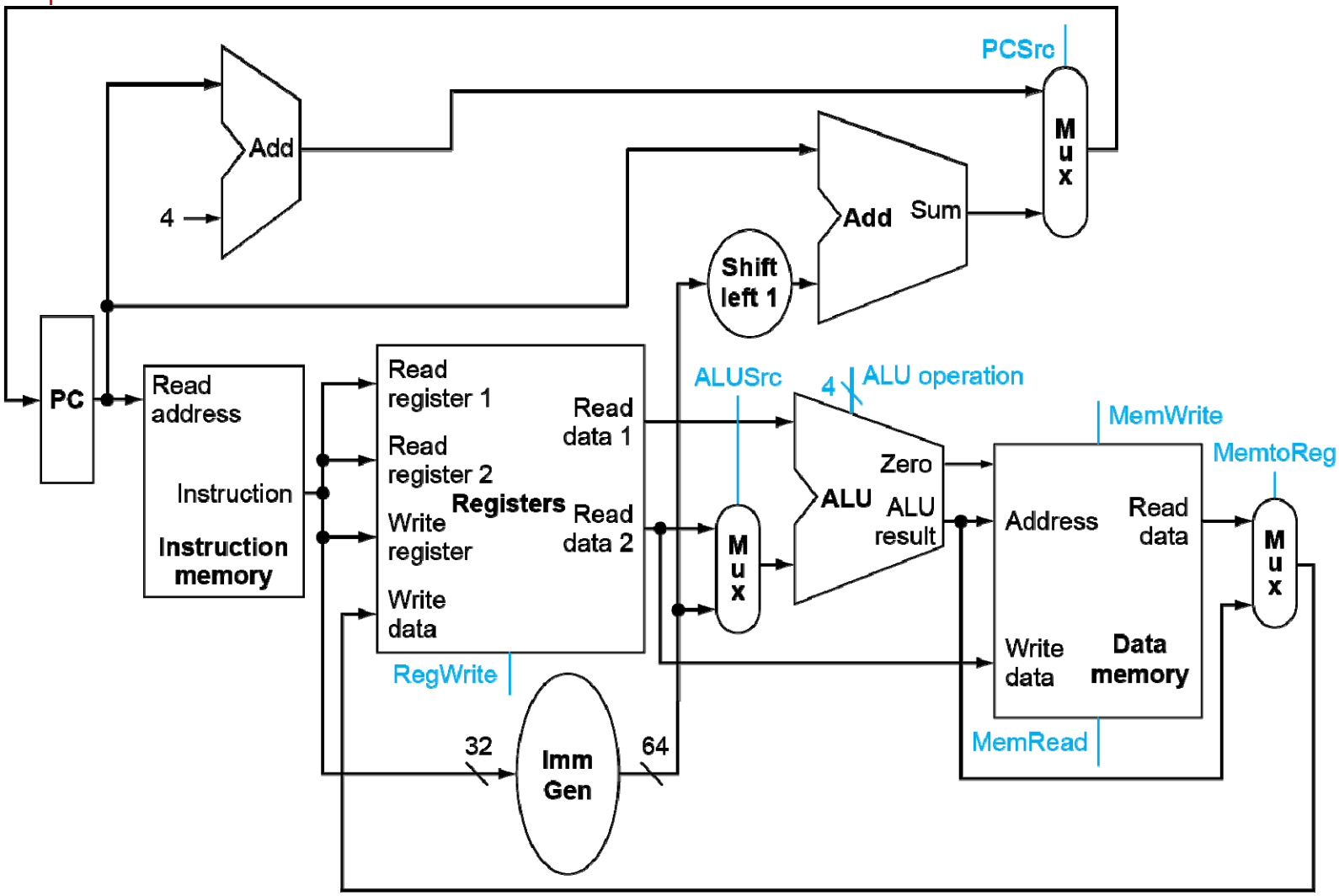




Full Datapath for R-type instruction

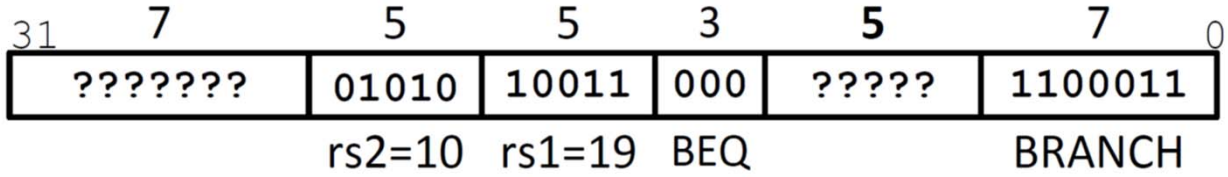
funct7	rs2	rs1	funct3	rd	opcode
7 bits	5 bits	5 bits	3 bits	5 bits	7 bits

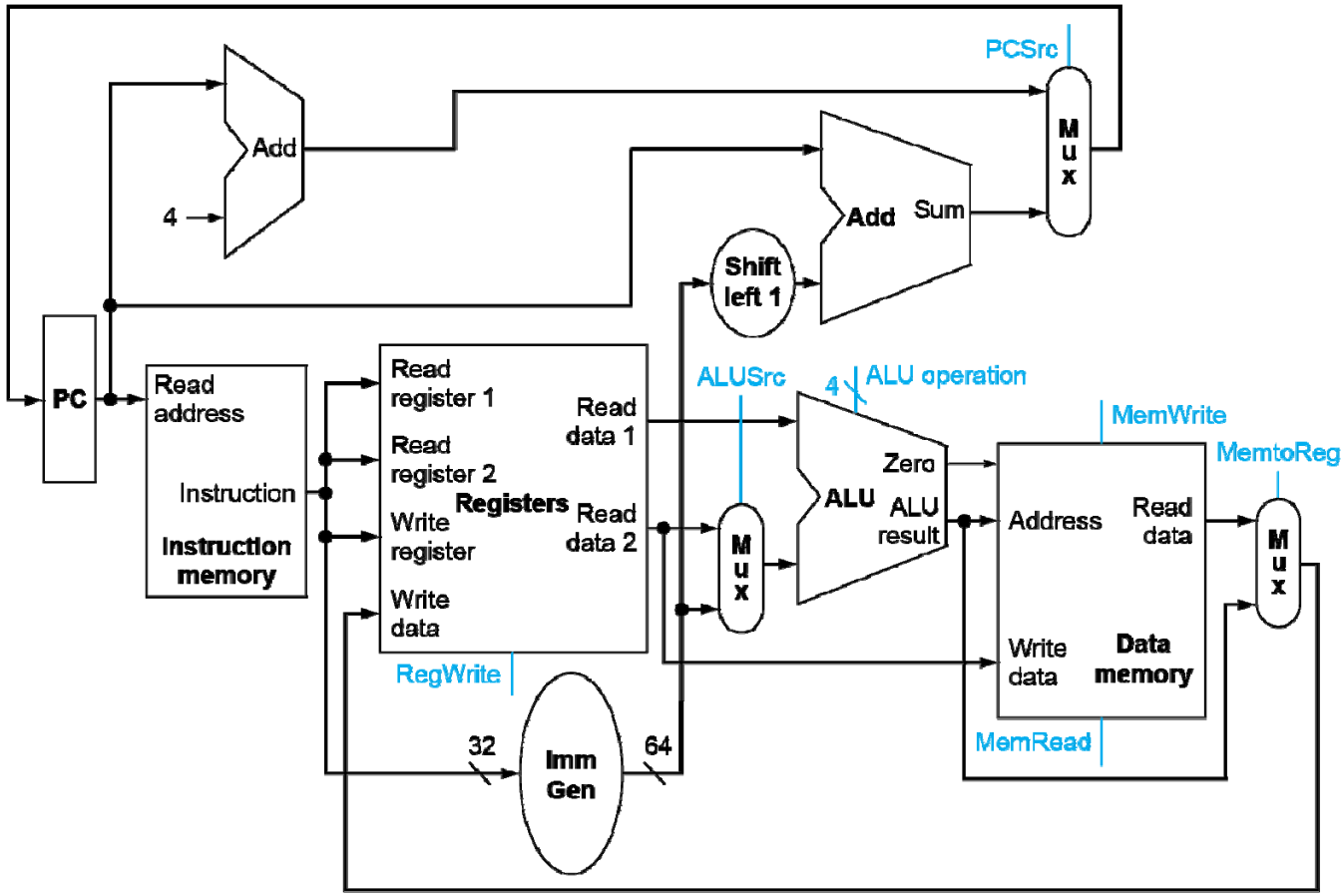
add x9,x20,x21



Branch instructions –

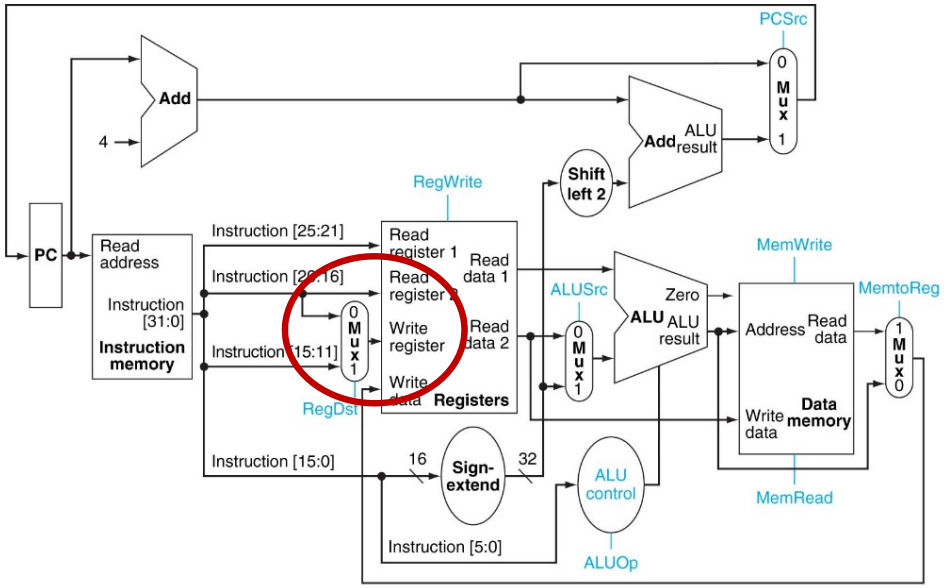
- beq x19,x10,End
  - mem[PC]
  - if ((R[rs] - R[rt]) eq 0)
    PC <- PC + sign-extend(imm13)
    else
    PC <- PC + 4



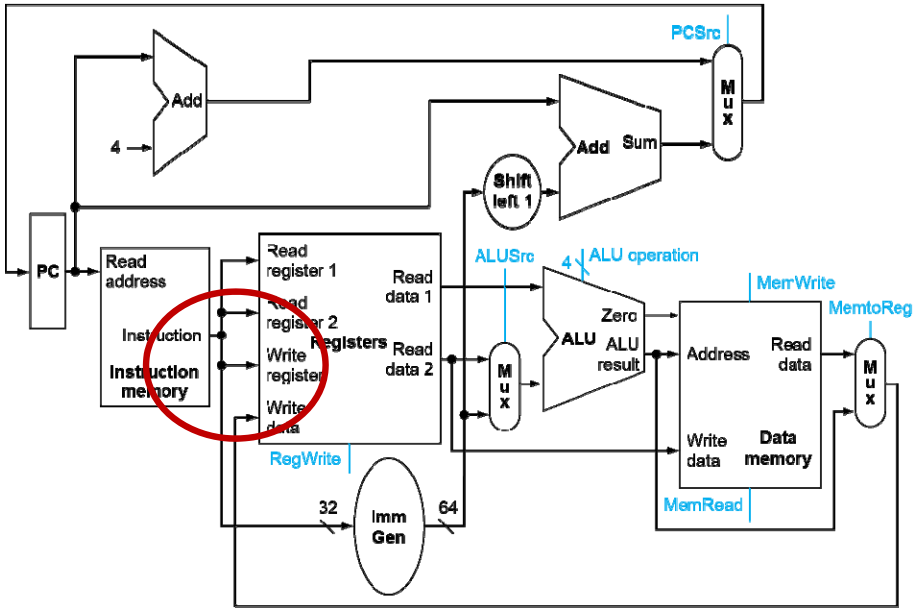


Observations between MIPS and RISC-v

MIPS



RISC-V



Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

Name (Field Size)	Field						Comments
	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits	
R-type	funct7	rs2	rs1	funct3	rd	opcode	Arithmetic instruction format
I-type	immediate[11:0]		rs1	funct3	rd	opcode	Loads & immediate arithmetic
S-type	immed[11:5]	rs2	rs1	funct3	immed[4:0]	opcode	Stores
SB-type	immed[12,10:5]	rs2	rs1	funct3	immed[4:1,11]	opcode	Conditional branch format
UJ-type	immediate[20,10:1,11,19:12]				rd	opcode	Unconditional jump format
U-type	immediate[31:12]				rd	opcode	Upper immediate format

Good design of instruction set architecture does matter!!

Let us add control to simple RISC-V

Control

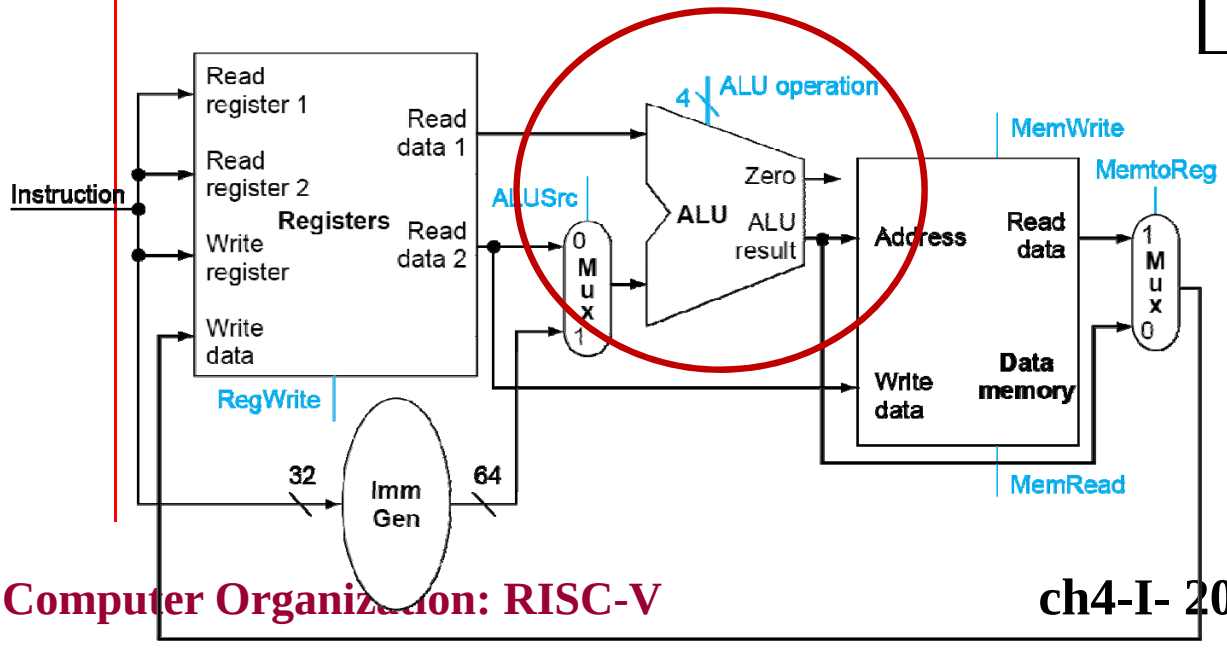
- Information comes from the 32 bits of the instruction
  - Determine control signals from 6-bit opcode field
- Selecting the operations to perform (ALU, read/write, etc.)
- Controlling the flow of data (multiplexor inputs)
  - **ALUsrc**
  - **MemtoReg**
- Enable/Disable control signals
  - **Branch, MemRead, MemWrite, RegWrite**
- ALU's operation based on instruction type and function code
  - **ALUop**

ALU Control

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

Don't confuse with add or sub instructions

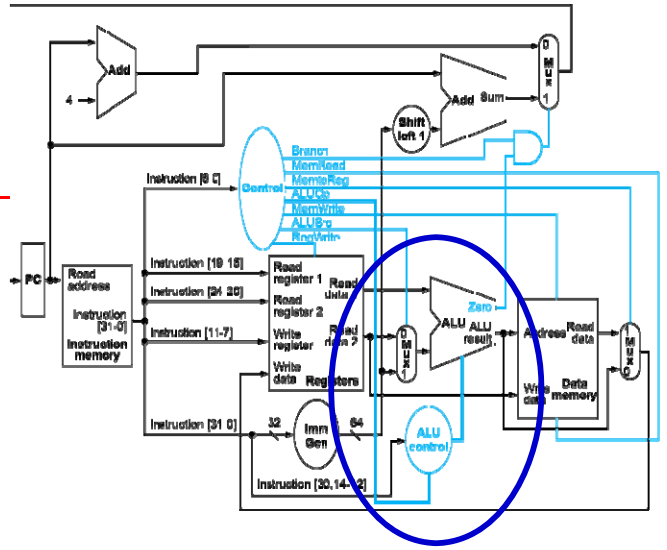
ALU control	Function
0000	AND
0001	OR
0010	add
0110	subtract

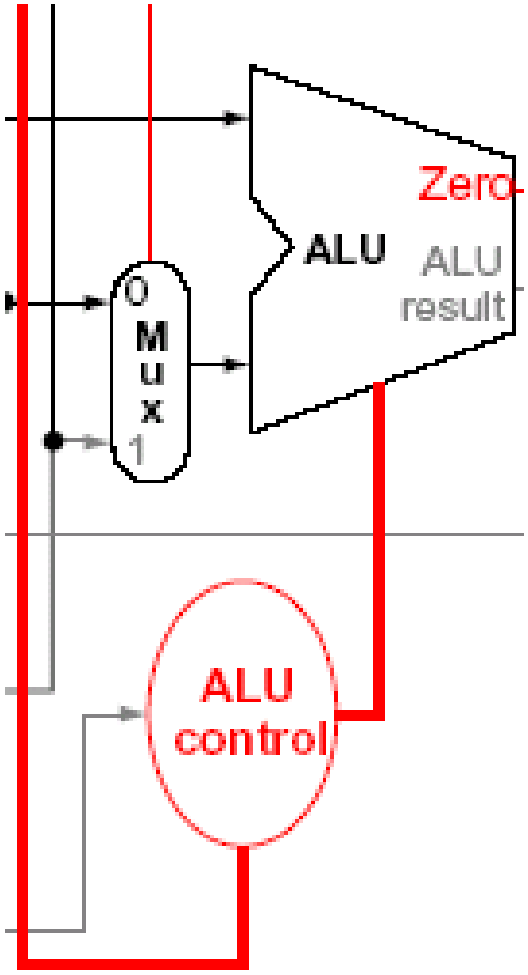


Determine ALU control

- ALU control input
  - 0000 AND
  - 0001 OR
  - 0010 add
  - 0110 subtract
  - 0111 set-on-less-than
  - 1100 NOR

- Using ALUOP to indicate instruction type
  - 00 = lw, sw
  - 01 = beq,
  - 1x = arithmetic



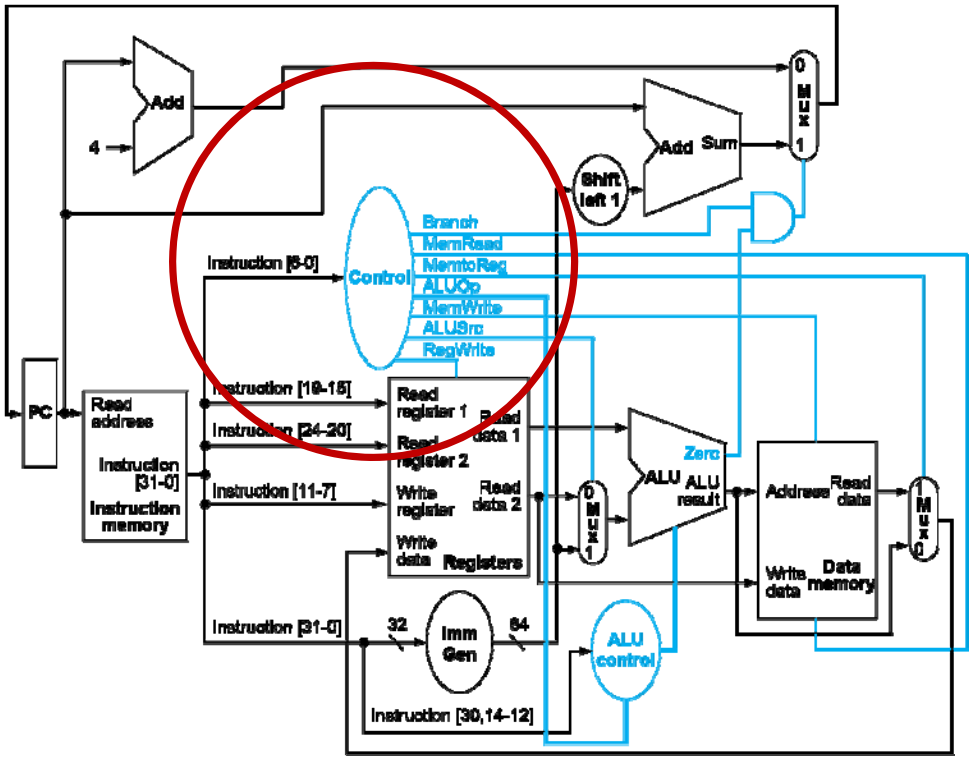


ALU Control

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

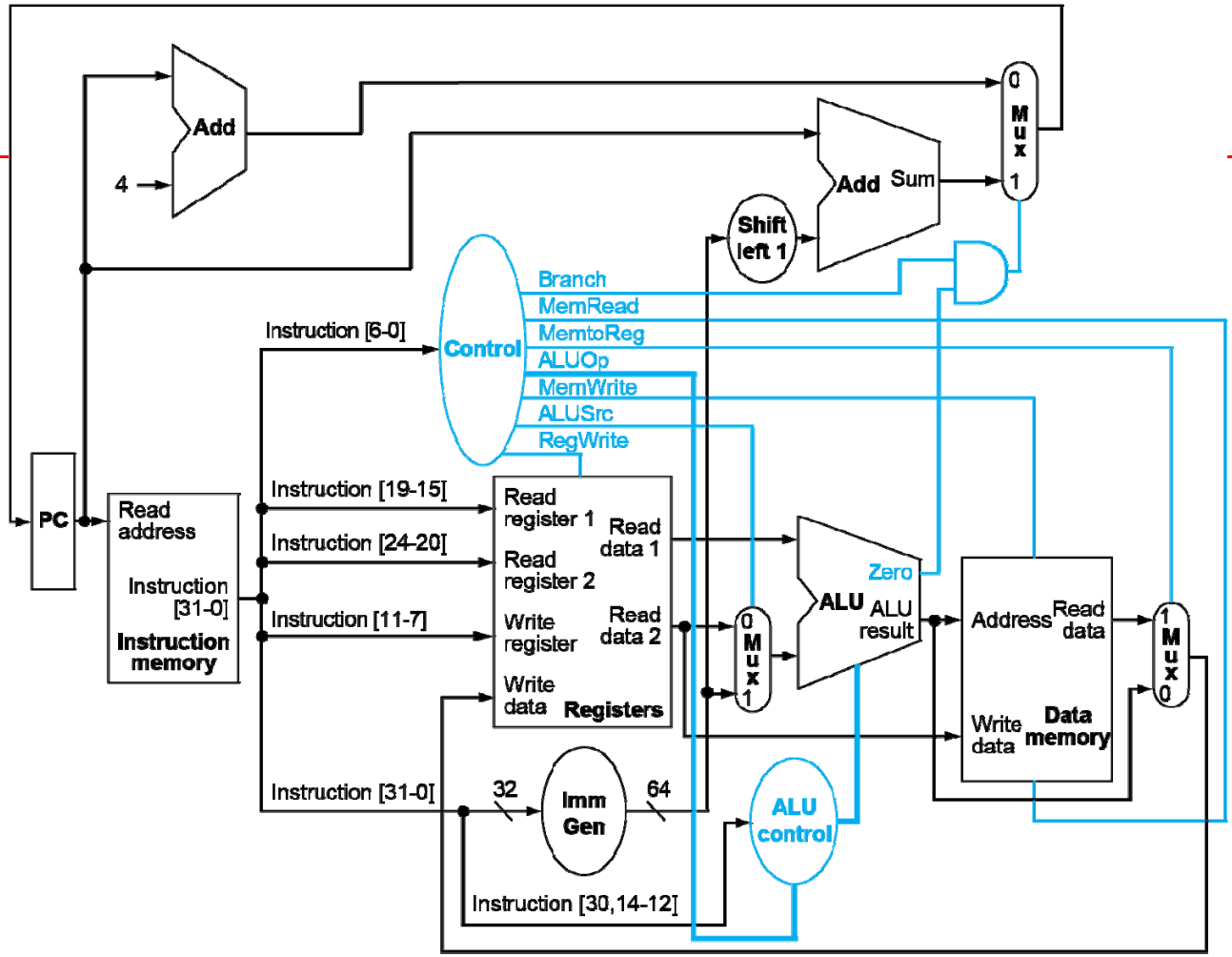
opcode	ALUOp	Operation	Opcode field	ALU function	ALU control
ld	00	load register	XXXXXXXXXXX	add	0010
sd	00	store register	XXXXXXXXXXX	add	0010
beq	01	branch on equal	XXXXXXXXXXX	subtract	0110
R-type	10	add	100000	add	0010
		subtract	100010	subtract	0110
		AND	100100	AND	0000
		OR	100101	OR	0001

Control function specified by truth table



Input or output	Signal name	R-format	ld	sd	beq
Inputs	I[6]	0	0	0	1
	I[5]	1	0	1	1
	I[4]	1	0	0	0
	I[3]	0	0	0	0
	I[2]	0	0	0	0
	I[1]	1	1	1	1
	I[0]	1	1	1	1
Outputs	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
	ALUOp0	0	0	0	1

Main control



Instruction	RegDst	ALUSrc	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALU Op0
R-format								
lw								
sw								
beq								

The Main Control Unit

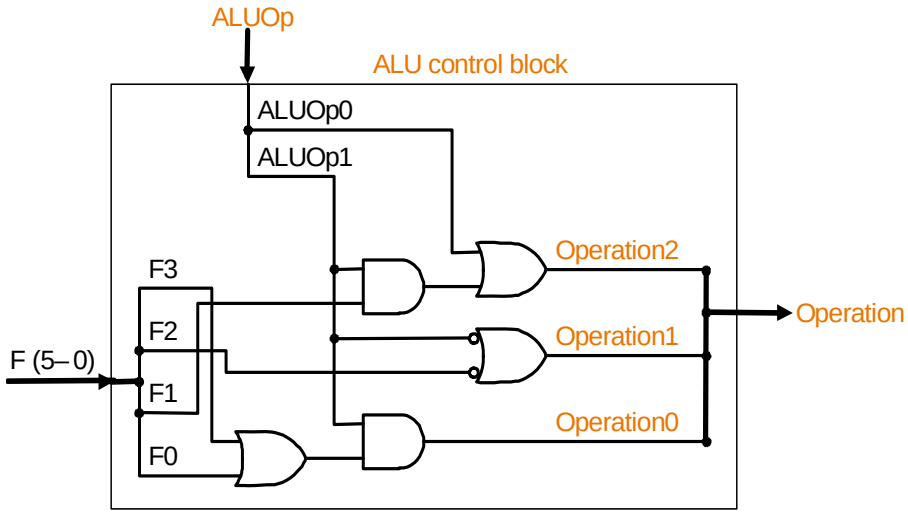
- Control signals derived from instruction

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

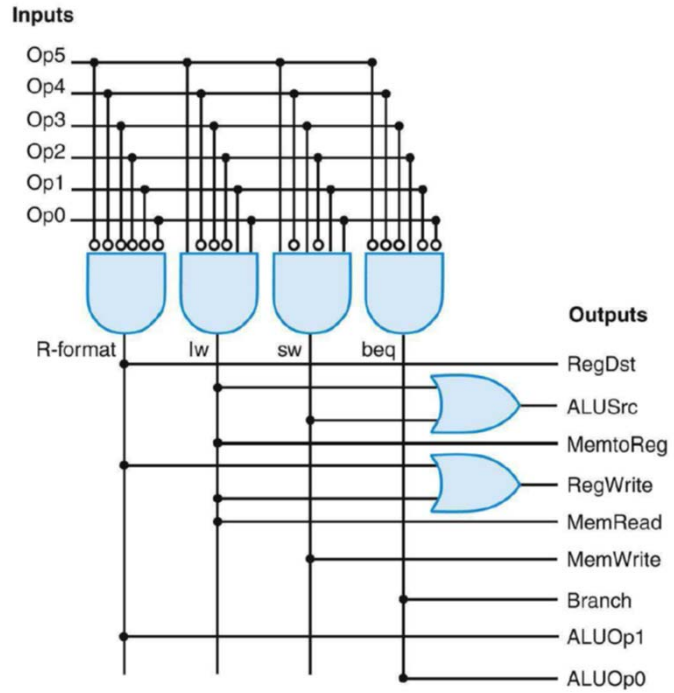
ALUOp		Funct7 field							Funct3 field			
ALUOp1	ALUOp0	I[31]	I[30]	I[29]	I[28]	I[27]	I[26]	I[25]	I[14]	I[13]	I[12]	Operation
0	0	X	X	X	X	X	X	X	X	X	X	0010
X	1	X	X	X	X	X	X	X	X	X	X	0110
1	X	0	0	0	0	0	0	0	0	0	0	0010
1	X	0	1	0	0	0	0	0	0	0	0	0110
1	X	0	0	0	0	0	0	0	1	1	1	0000
1	X	0	0	0	0	0	0	0	1	1	0	0001

Control for ALU

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



- Generate control logic using a truth table (can turn into gates):



- Implemented by programmable logic array (PLA)

Performance of Single-cycle Machine

- Calculate cycle time assuming negligible delays except:
  - memory (200ps), ALU and adders (100ps), register file access (100ps)

Inst. class	Inst mem	Reg read	ALU operatn	Data mem	Reg write	total
R-type						
LW						
SW						
branch						
Jump						

- Cycle time

Real encoding

31 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
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

32-bit R-type ALU

Reg-Reg Instruction

31 20	19 15	14 12	11 7	6 0	
imm[11:0]	rs1	funct3	rd	opcode	I-type
imm[11:0]	rs1	000	rd	0000011	LB
imm[11:0]	rs1	001	rd	0000011	LH
imm[11:0]	rs1	010	rd	0000011	LW
imm[11:0]	rs1	100	rd	0000011	LBU
imm[11:0]	rs1	101	rd	0000011	LHU

load instruction

31 25	24 20	19 15	14 12	11 7	6 0	
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

store instruction