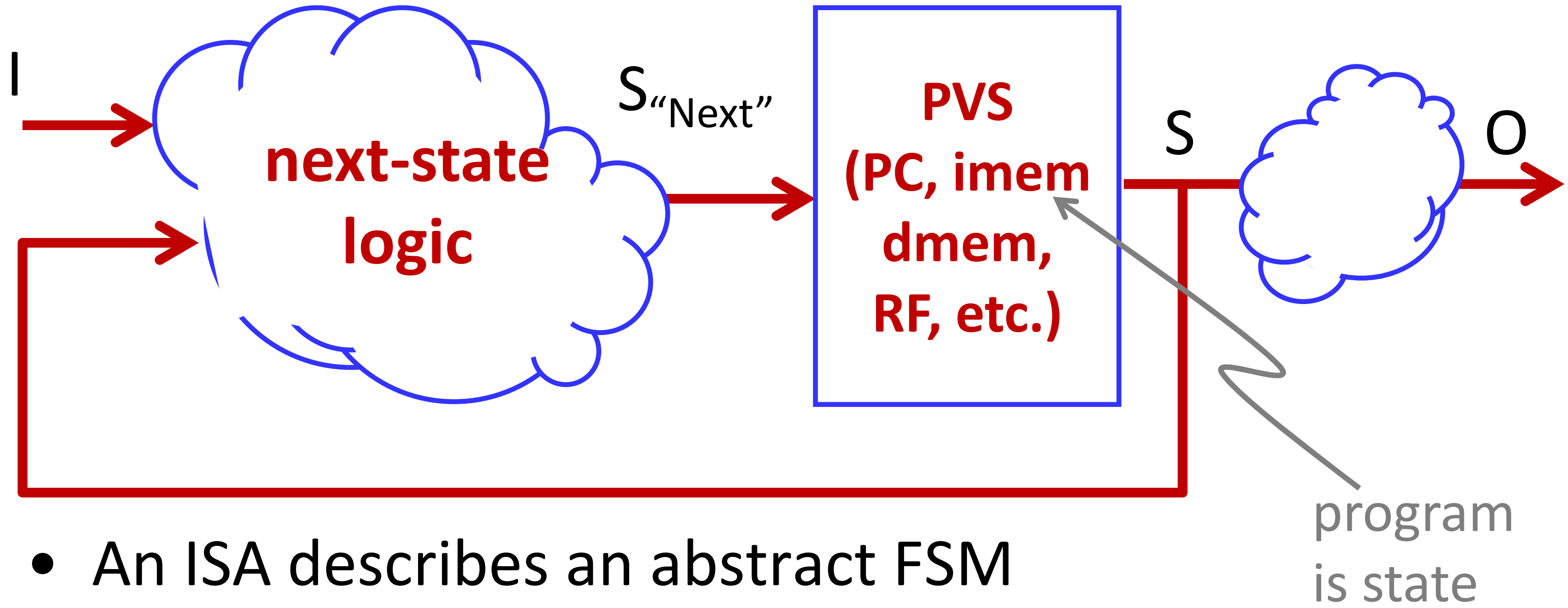
18-447 Lecture 3:
Single-Cycle Microarchitecture

James C. Hoe
Department of ECE
Carnegie Mellon University

Housekeeping

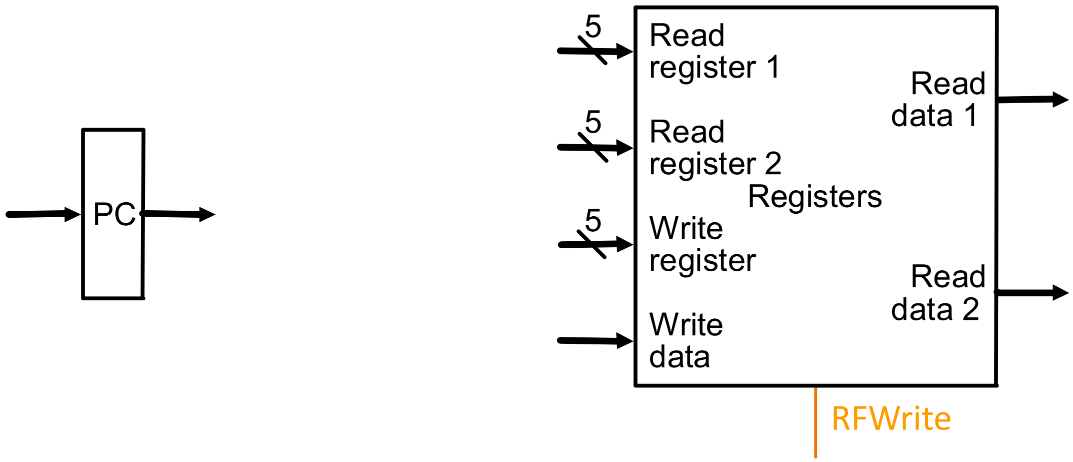
- Your goal today
  - first try at implementing the RV32I ISA
- Notices
  - Student survey on Canvas, due Wednesday
  - Handout #4: HW1, due noon 2/22
  - Lab 1, Part A, due week of 2/19
  - Lab 1, Part B, due week of 2/26
- Readings
  - P&H Ch 4.1~4.4
  - rest of P&H Ch 2 for next time

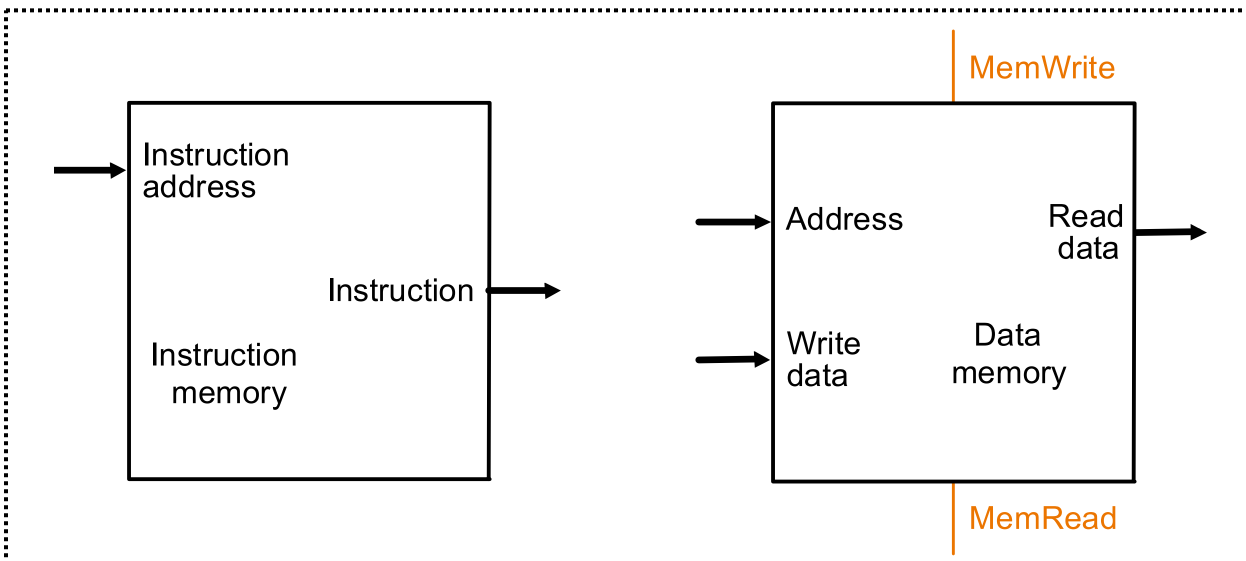
Instruction Processing FSM



- An ISA describes an abstract FSM
  - state = program visible state
  - state transition = instruction execution
- Nice ISAs have atomic instruction semantics
  - one state transition per instruction in abstract FSM
- The implementation FSM can look wildly different

Program Visible State
(aka Architectural State)





"Magic" Memory and Register File

- Combinational Read
  - output of the read data port is a combinational function of the register file contents and the corresponding read select port

- Synchronous write
  - the selected register (or memory location) is updated on the posedge clock transition when write enable is asserted

Cannot affect read output in between clock edges

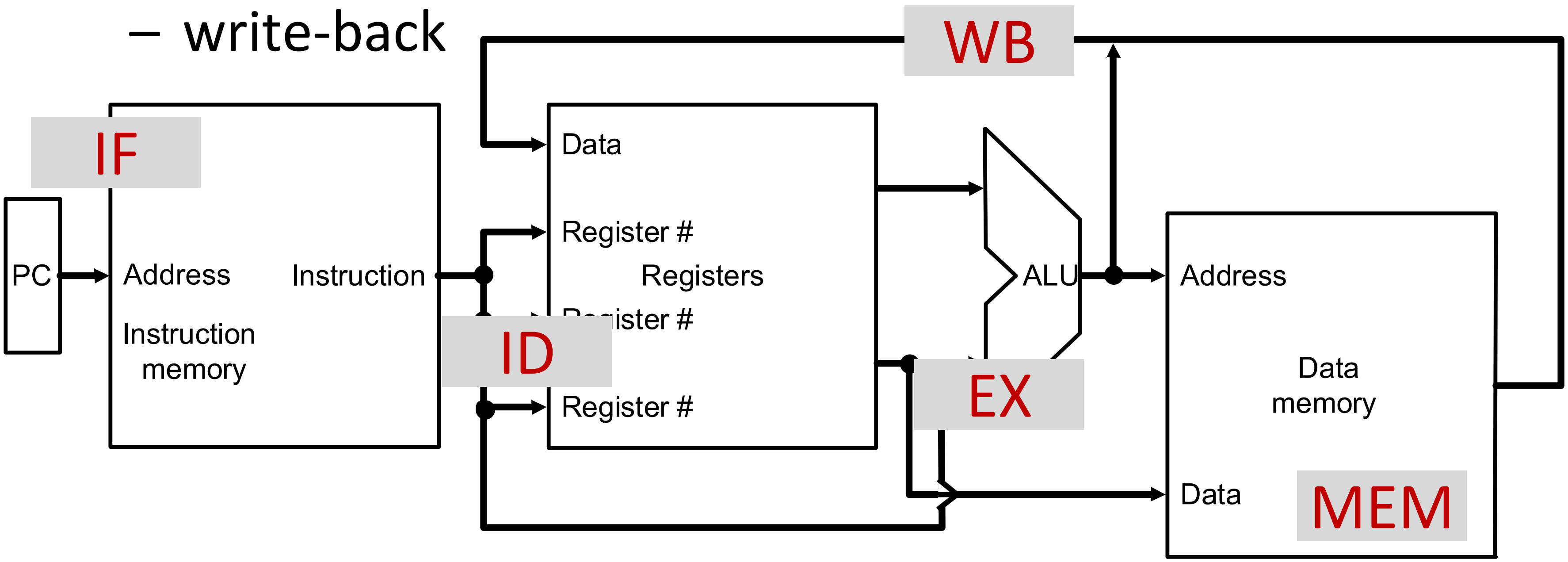
Simplifying Characteristics of "RISC"

- Simple operations
  - 2-input, 1-output arithmetic and logical operations
  - few alternatives for accomplishing the same thing
- Simple data movements
  - ALU ops are register-to-register, never memory
  - "load-store" architecture, 1 addressing mode
- Simple branches
  - limited varieties of branch conditions and targets
  - PC-offset
- Simple instruction encoding
  - all instructions encoded in the same number of bits
  - simple, fixed formats

(RISC=Reduced Instruction Set Computer)

RISC Instruction Processing

- 5 generic steps
  - instruction fetch
  - instruction decode and operand fetch
  - ALU/execute
  - memory access (not required by non-mem instructions)
  - write-back



Single-Cycle Datapath for RV32I ALU Instructions

Register-Register ALU Instructions

- Assembly (e.g., register-register addition)

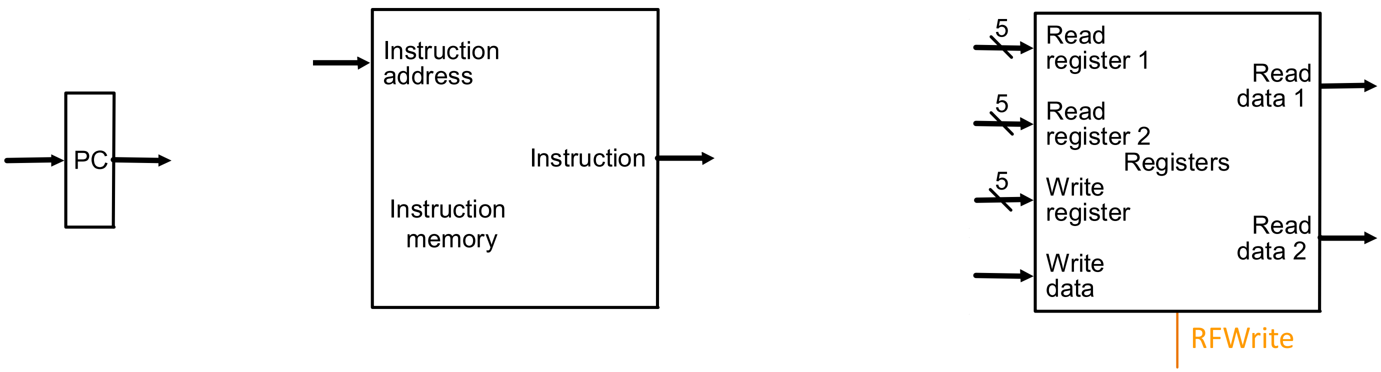
    ADD rd, rs1, rs2

- Machine encoding

0000000	rs2	rs1	000	rd	0110011
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit

- Semantics
    - GPR[rd] $\leftarrow$ GPR[rs1] + GPR[rs2]
    - PC $\leftarrow$ PC + 4
- Exceptions: none (ignore carry and overflow)
- Variations
    - Arithmetic: {ADD, SUB}
    - Compare: {signed, unsigned} x {Set if Less Than}
    - Logical: {AND, OR, XOR}
    - Shift: {Left, Right-Logical, Right-Arithmetic}

ADD rd rs1 rs2



0000000	rs2	rs1	000	rd	0110011
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit

if MEM[PC] == ADD rd rs1 rs2

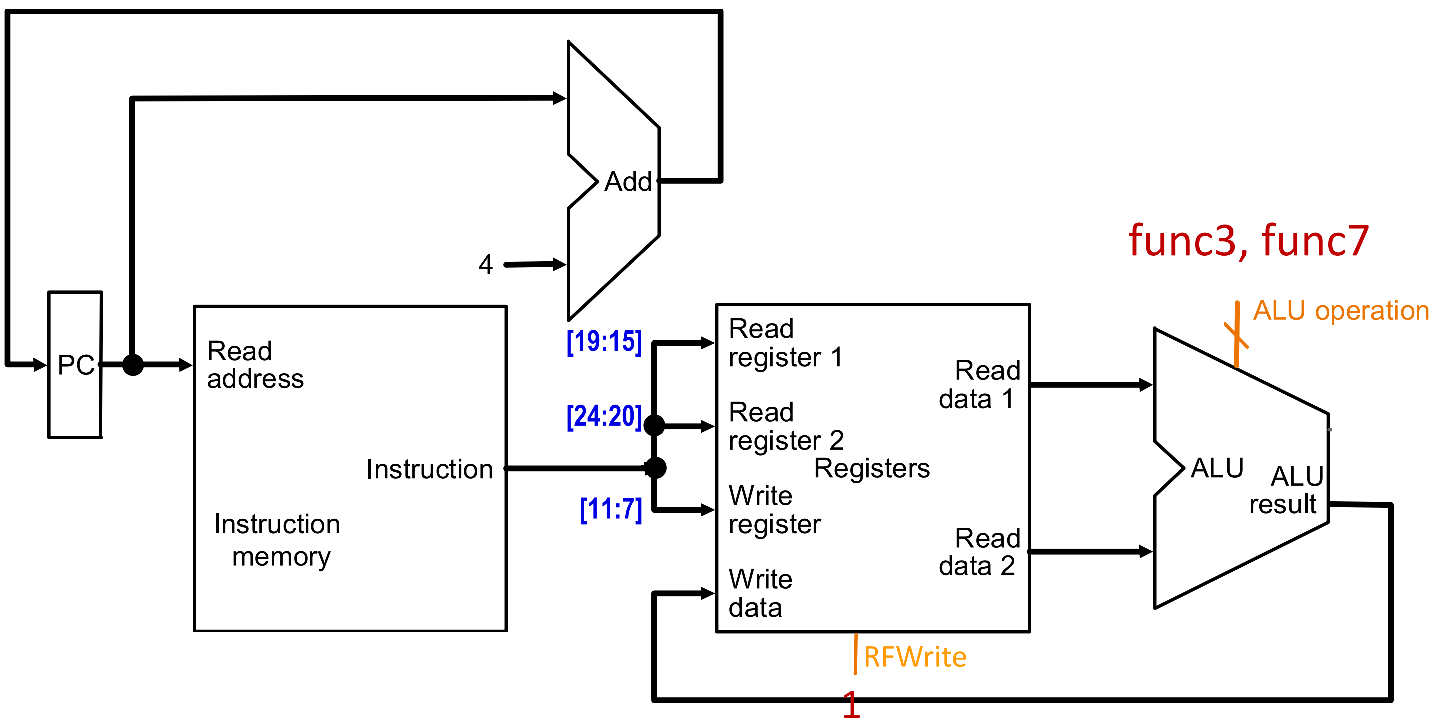
GPR[rd] $\leftarrow$ GPR[rs1] + GPR[rs2]

PC $\leftarrow$ PC + 4

IF	ID	EX	MEM	WB

Combinational state update logic

R-Type ALU Datapath



Reg-Immediate ALU Instructions

- Assembly (e.g., reg-immediate additions)

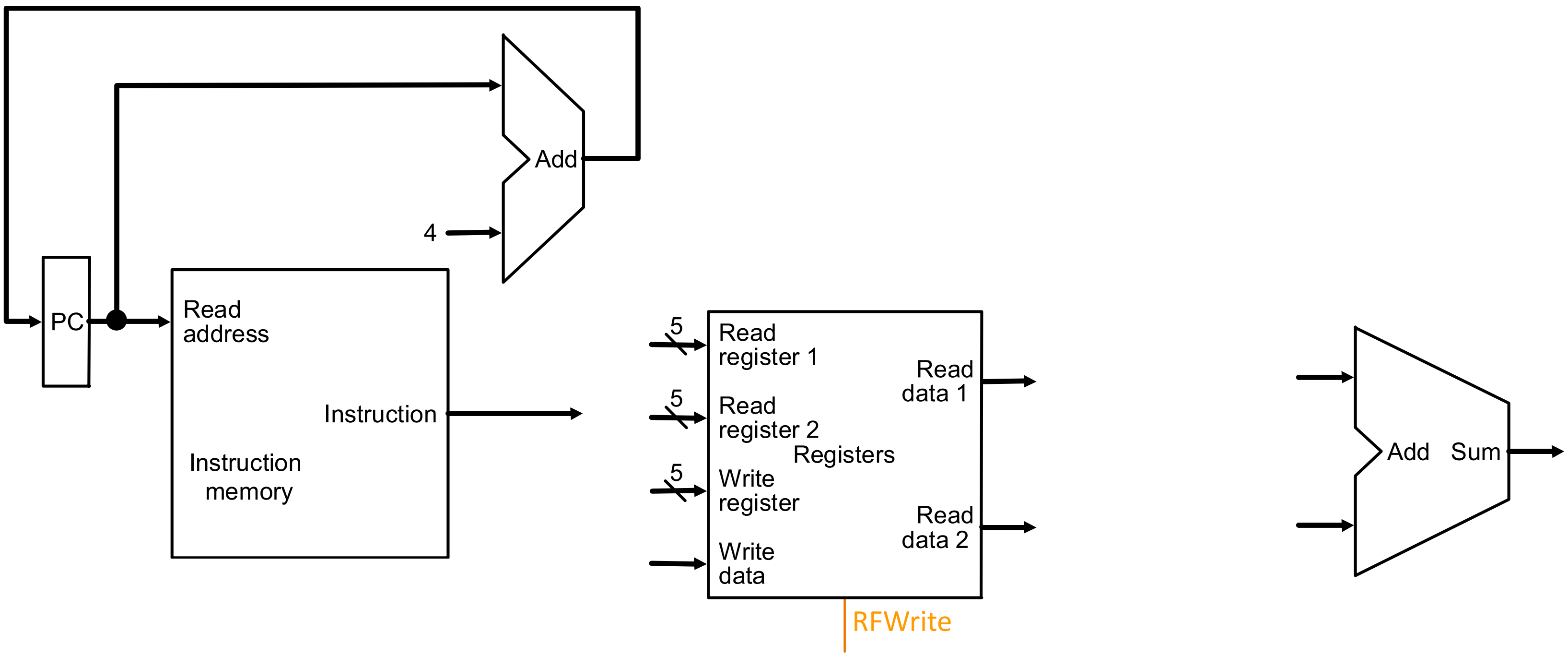
  ADDI rd, rs1, imm_{12}

- Machine encoding

imm[11:0]	rs1	000	rd	0010011
12-bit	5-bit	3-bit	5-bit	7-bit

- Semantics
  - GPR[rd] ← GPR[rs1] + sign-extend (imm)
  - PC ← PC + 4
- Exceptions: none (ignore carry and overflow)
- Variations
  - Arithmetic: {ADDI, ~~SUBI~~}
  - Compare: {signed, unsigned} x {Set if Less Than Imm}
  - Logical: {ANDI, ORI, XORI}
  - **Shifts by unsigned imm[4:0]: {SLLI, SRLI, SRAI}

ADDI rd rs1 immediate$_{12}$



imm[11:0]	rs1	000	rd	0010011
12-bit	5-bit	3-bit	5-bit	7-bit

if MEM[PC] == ADDI rd rs1 immediate

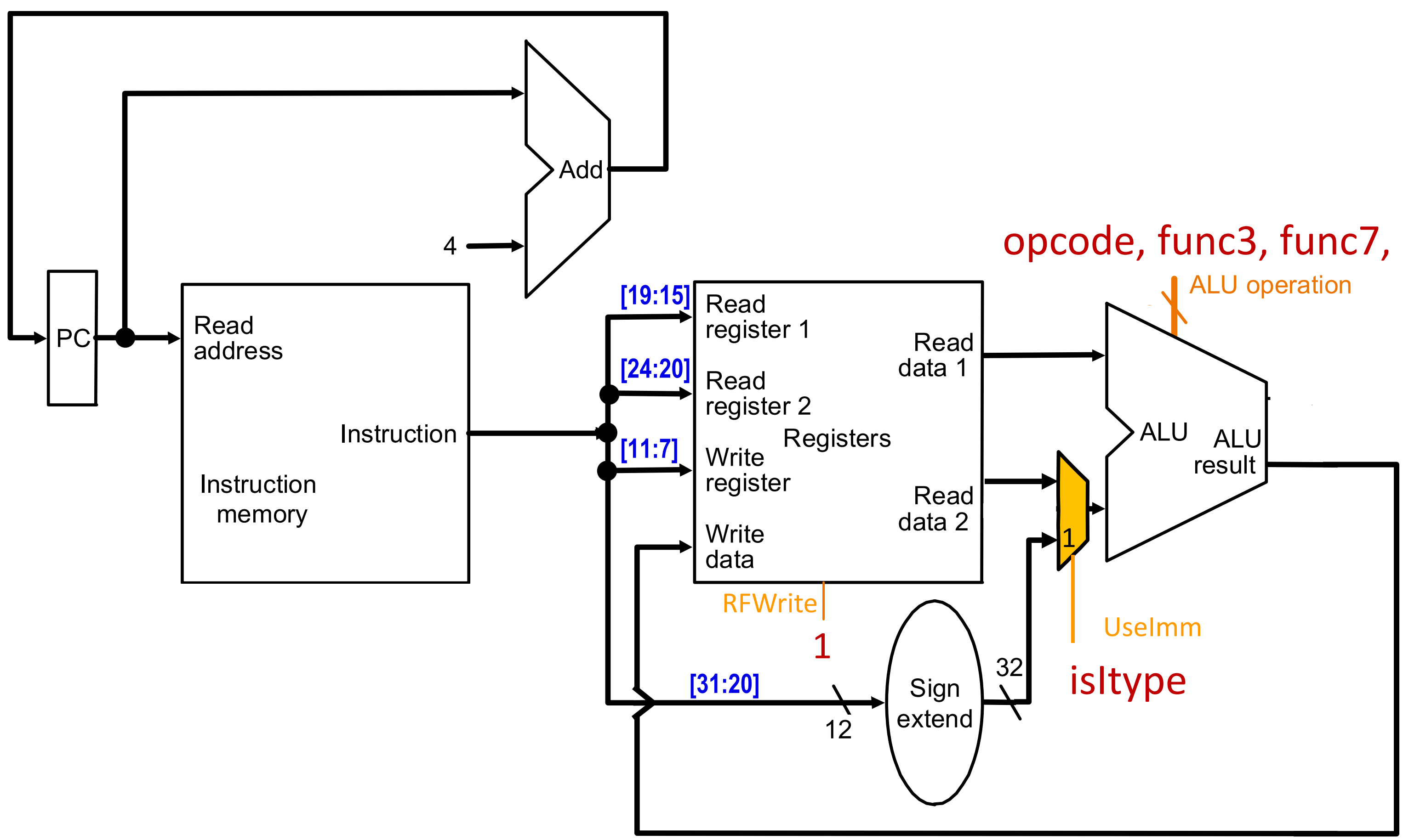
GPR[rd] ← GPR[rs1] + sign-extend (immediate)

PC ← PC + 4

IF	ID	EX	MEM	WB

Combinational state update logic

Datapath for R and I-type ALU Inst's



**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Single-Cycle Datapath for Data Movement Instructions

Load Instructions

- Assembly (e.g., load 4-byte word)

  LW rd, offset_{12}(base)

  rs1

- Machine encoding

offset[11:0]	base	010	rd	0000011
12-bit	5-bit	3-bit	5-bit	7-bit

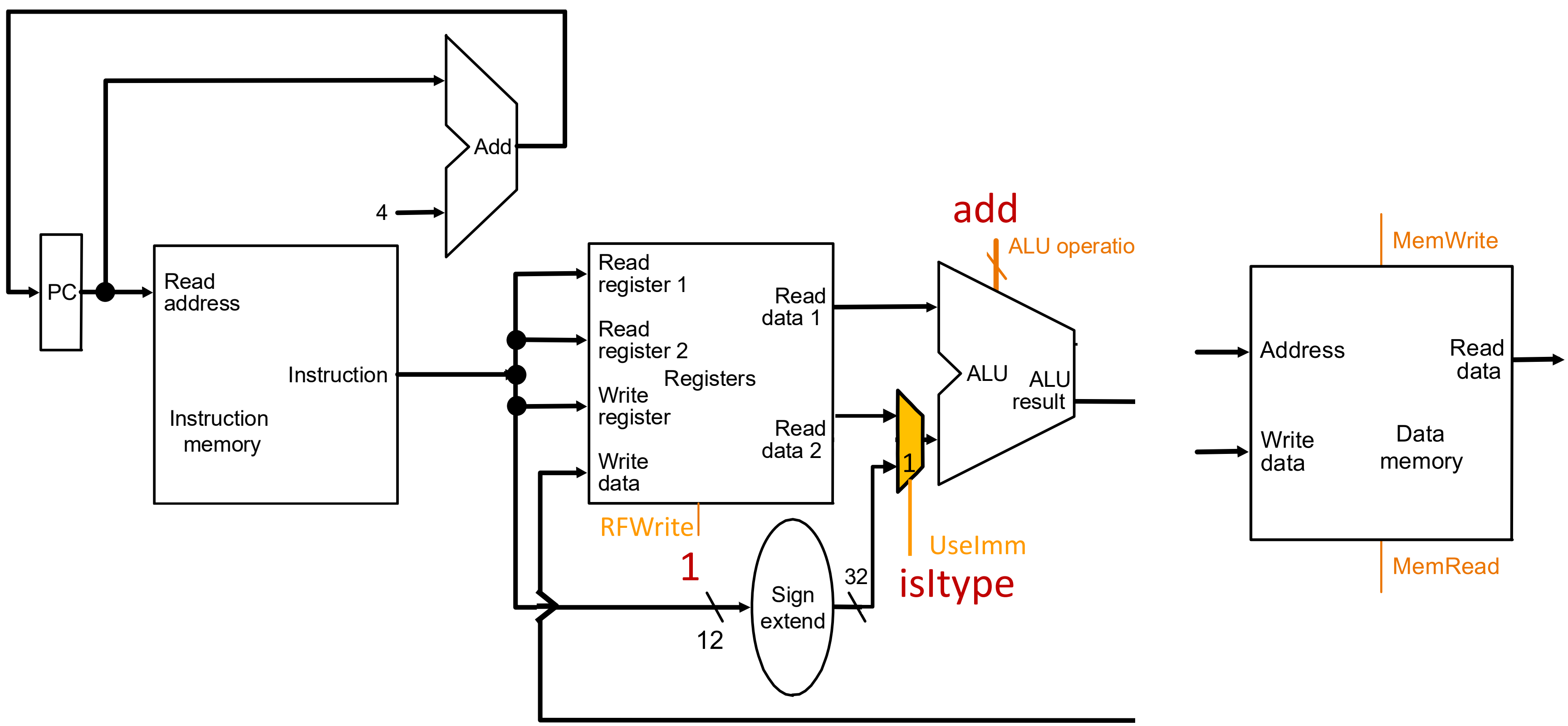
- Semantics
  - byte_address_{32} = sign-extend(offset_{12}) + GPR[base]
  - GPR[rd] $\leftarrow$ MEM_{32}[byte_address]
  - PC $\leftarrow$ PC + 4
- Exceptions: none for now
- Variations: LW, LH, LHU, LB, LBU

  e.g., LB :: GPR[rd] $\leftarrow$ sign-extend(MEM_8[byte_address])

  LBU :: GPR[rd] $\leftarrow$ zero-extend(MEM_8[byte_address])

  Note: RV32I memory is byte-addressable, little-endian

LW Datapath



if MEM[PC]==LW rd offset$_{12}$(base)
EA = sign-extend(offset) + GPR[base]
GPR[rd] ← MEM[EA]
PC ← PC + 4

IF	ID	EX	MEM	WB

Combinational
state update logic

Store Instructions

- Assembly (e.g., store 4-byte word)

  SW rs2, offset_{12}(base)

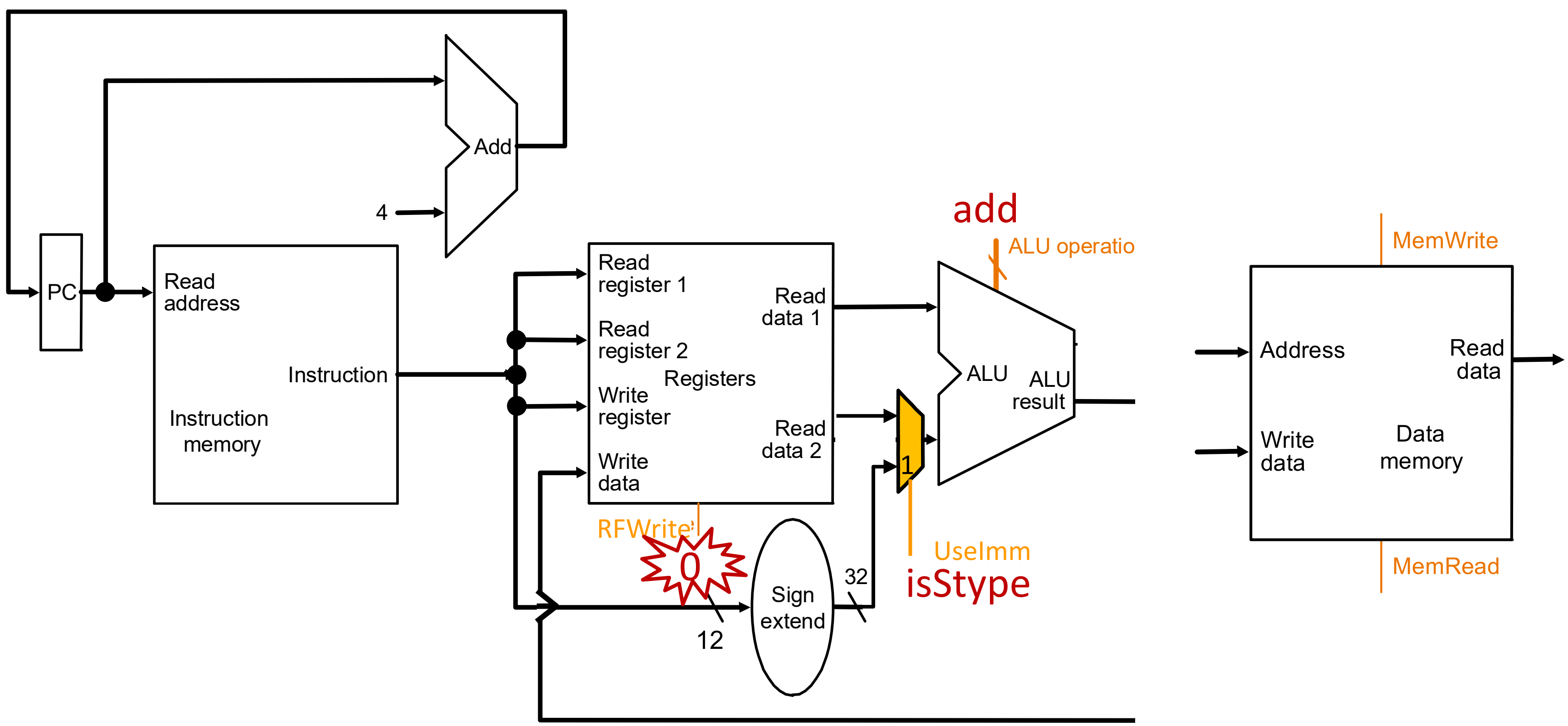
- Machine encoding

offset[11:5]	rs2	base	010	ofst[4:0]	0100011
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit

- Semantics
  - $\text{byte\_address}_{32} = \text{sign-extend}(\text{offset}_{12}) + \text{GPR}[\text{base}]$
  - $\text{MEM}_{32}[\text{byte\_address}] \leftarrow \text{GPR}[\text{rs2}]$
  - $\text{PC} \leftarrow \text{PC} + 4$
- Exceptions: none for now
- Variations: SW, SH, SB

  e.g., SB:: $\text{MEM}_{8}[\text{byte\_address}] \leftarrow (\text{GPR}[\text{rs2}])[7:0]$

SW Datapath



if MEM[PC]==SW rs2 offset$_{12}$(base)

EA = sign-extend(offset) + GPR[base]

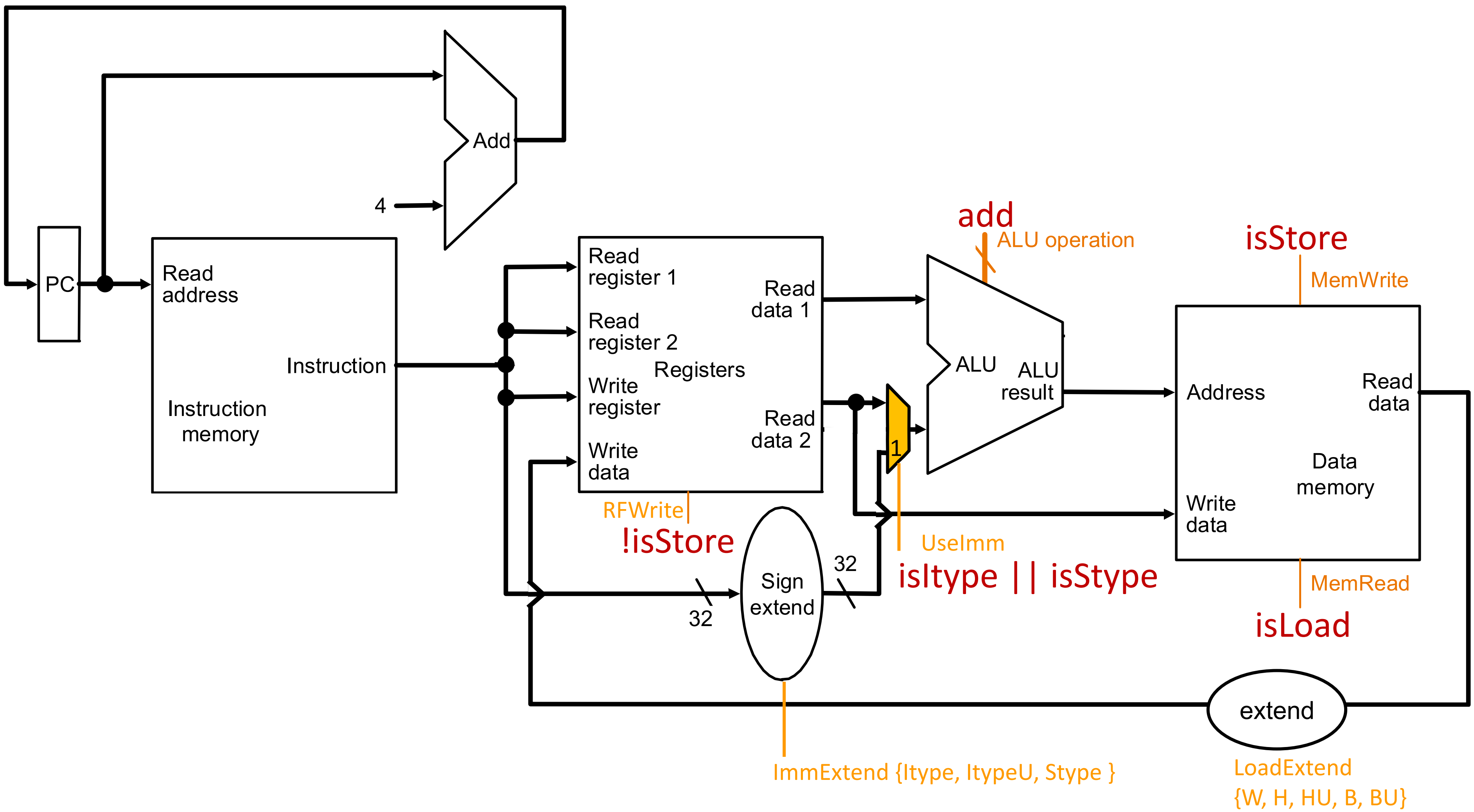
MEM[EA] ← GPR[rs2]

PC ← PC + 4

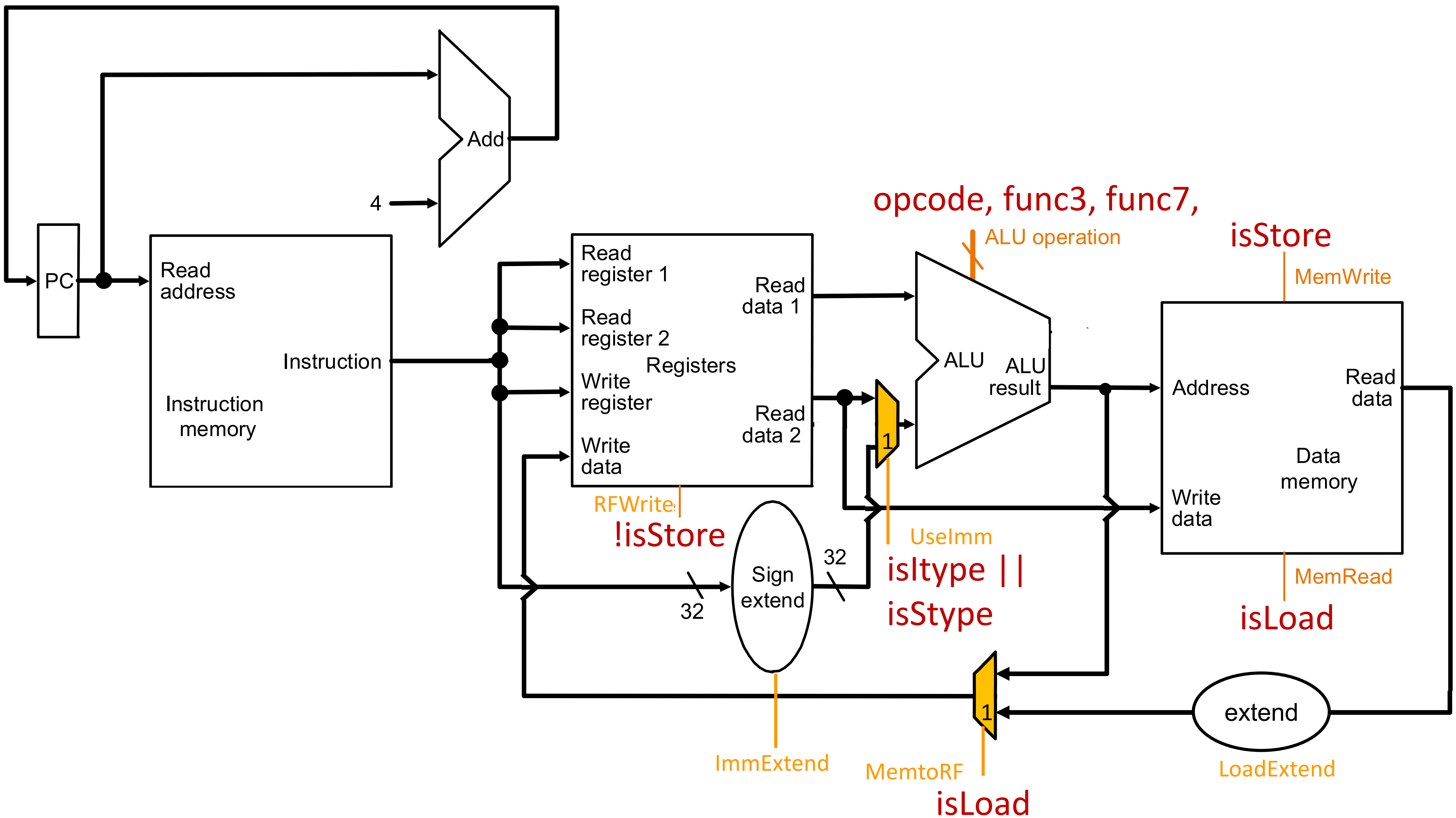
IF	ID	EX	MEM	WB

Combinational state update logic

Load-Store Datapath



Datapath for Non-Control Flow Inst's



Single-Cycle Datapath for Control Flow Instructions

Jump and Link Instruction

- Assembly

  JAL rd imm_{21} Note: implicit imm[0]=0

- Machine encoding

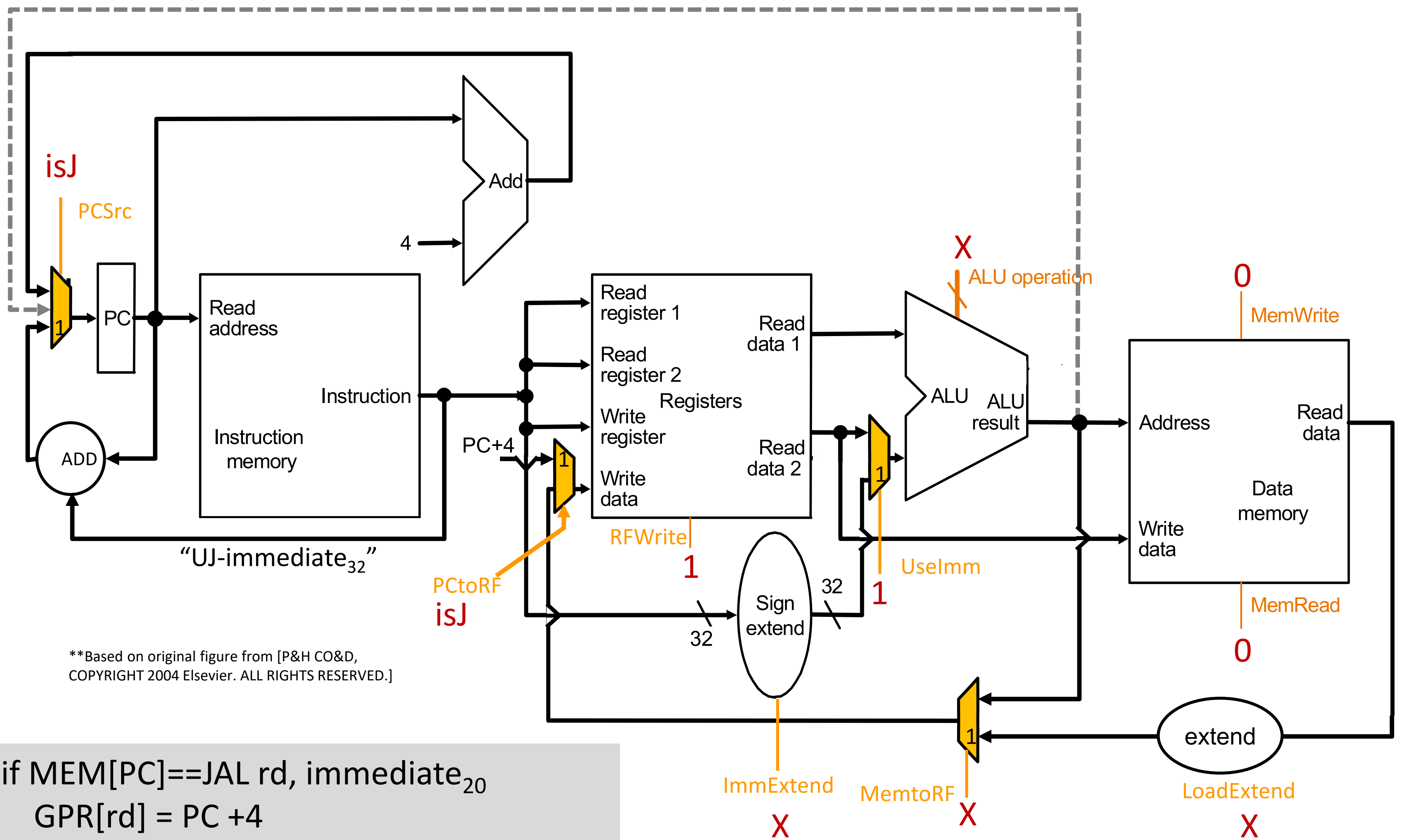
imm[20|10:1|11|19:12]	rd	1101111	UJ-type
20-bit	5-bit	7-bit	

- Semantics
  - target = PC + sign-extend(imm_{21})
  - GPR[rd] $\leftarrow$ PC + 4
  - PC $\leftarrow$ target

  How far can you jump?

- Exceptions: misaligned target (4-byte)

Unconditional Jump Datapath



**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

if MEM[PC]==JAL rd, immediate$_{20}$

GPR[rd] = PC +4

PC = PC + sign-extend(imm$_{21}$)

What about JALR?

(Conditional) Branch Instructions

- Assembly (e.g., branch if equal)

  BEQ rs1, rs2, imm_{13} Note: implicit imm[0]=0

- Machine encoding

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011
7-bit	5-bit	5-bit	3-bit	5-bit	7-bit

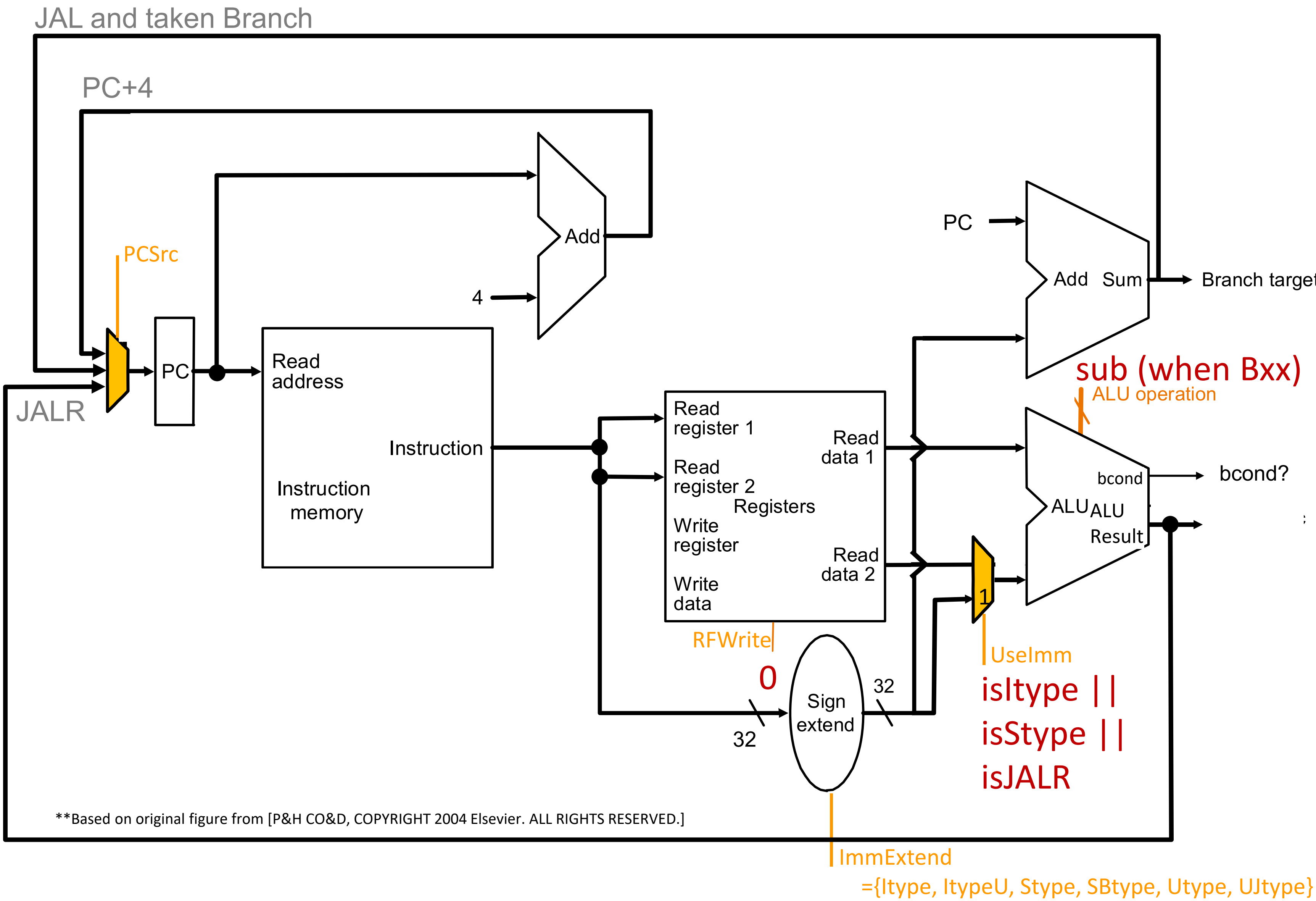
- Semantics
  - target = PC + sign-extend(imm_{13})
  - if GPR[rs1]==GPR[rs2] then PC $\leftarrow$ target

    else PC $\leftarrow$ PC + 4

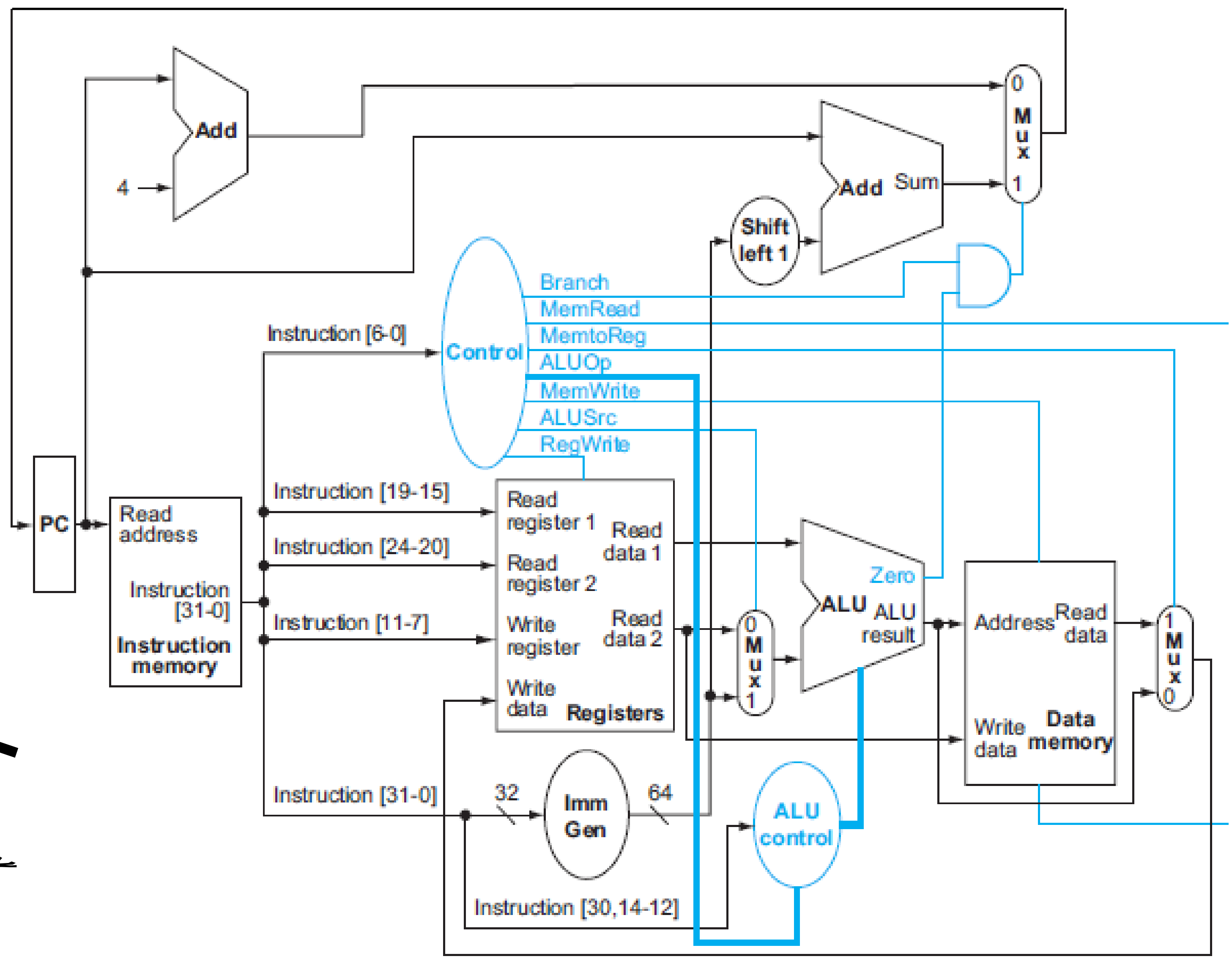
    How far can you jump?

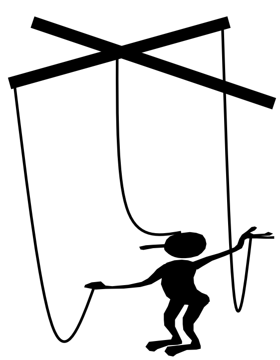
- Exceptions: misaligned target (4-byte) if taken
- Variations
  - BEQ, BNE, BLT, BGE, BLTU, BGEU

Conditional Branch Datapath

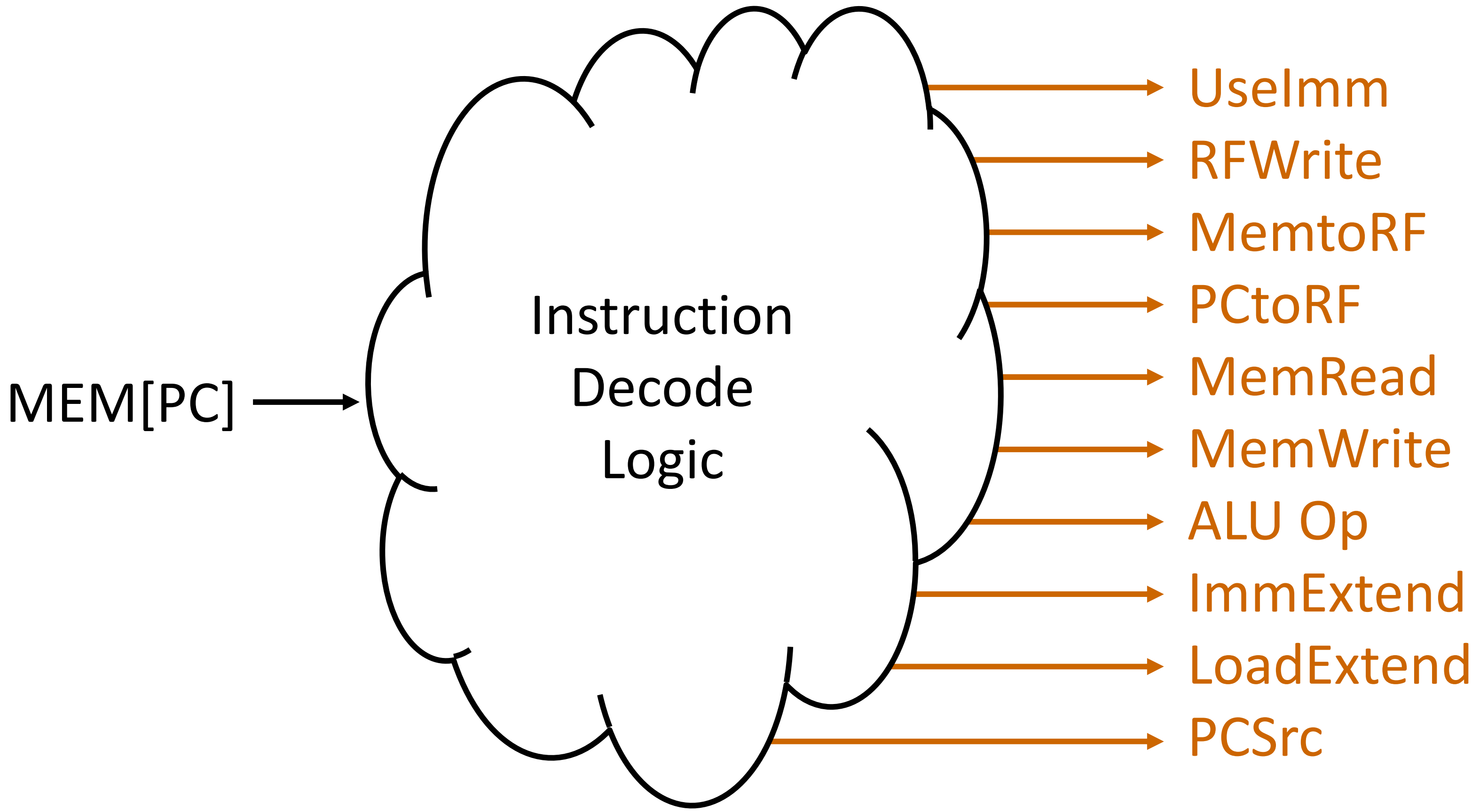


Adding Control to Datapath

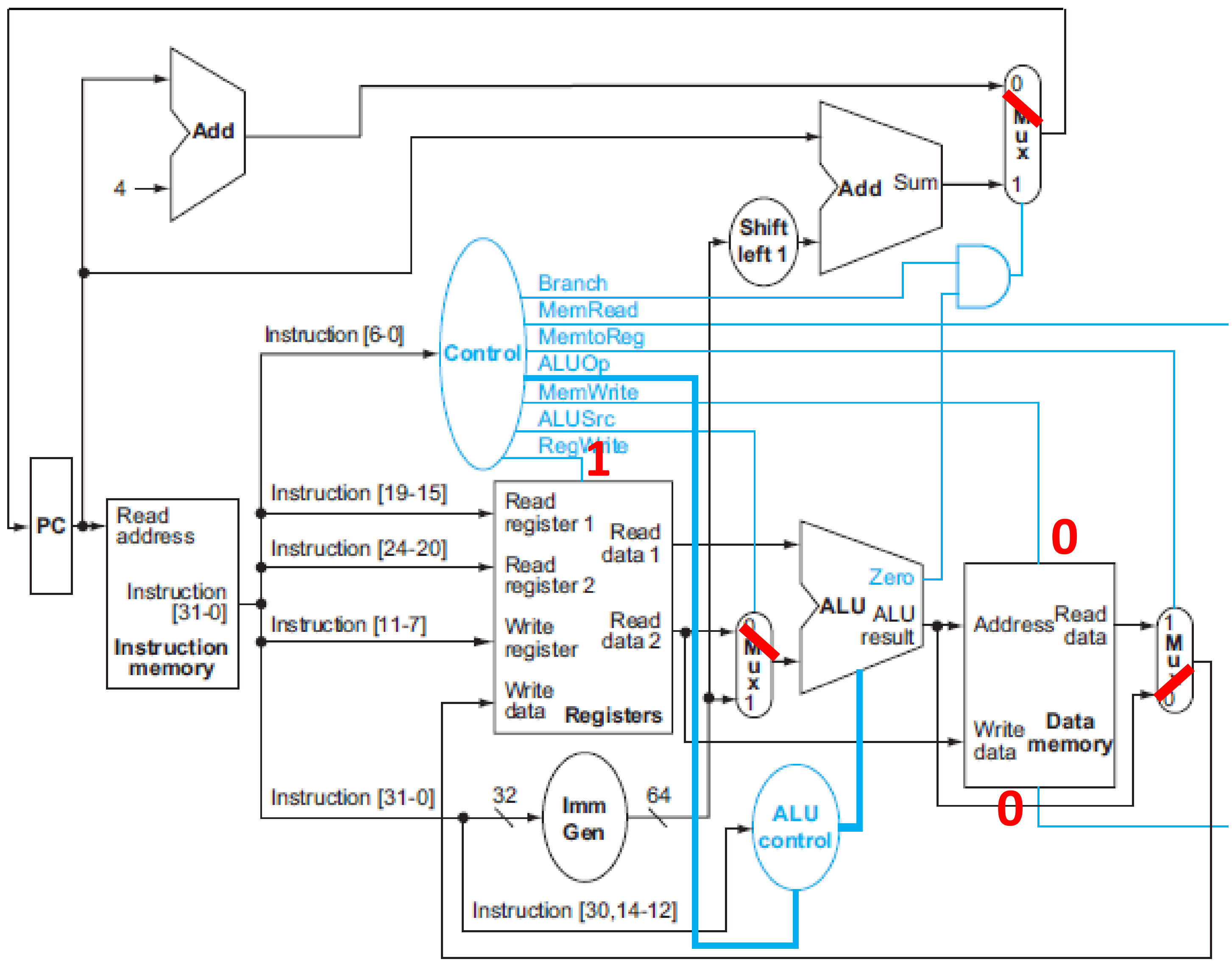




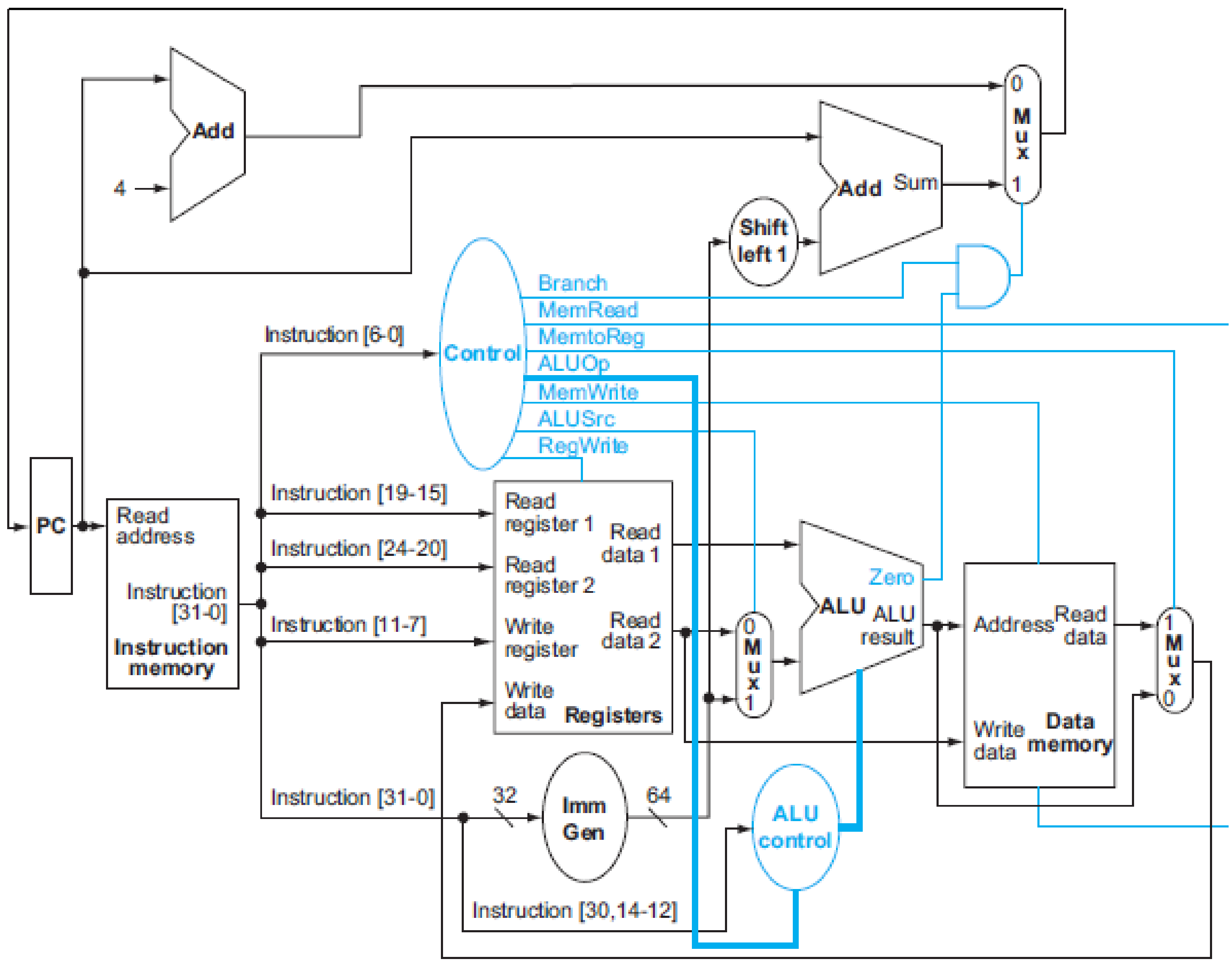
Datapath Control Generation



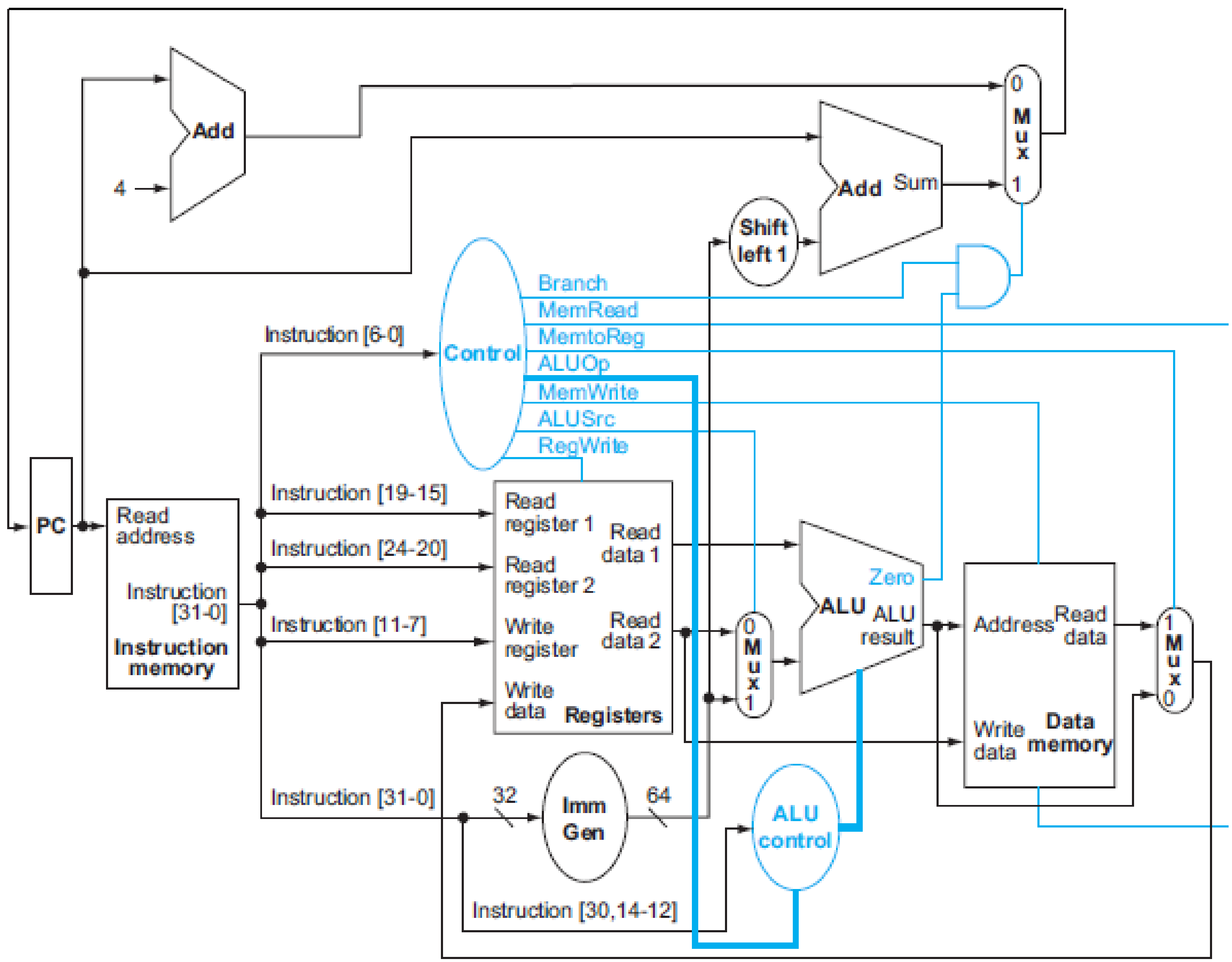
R-Type ALU Worksheet:



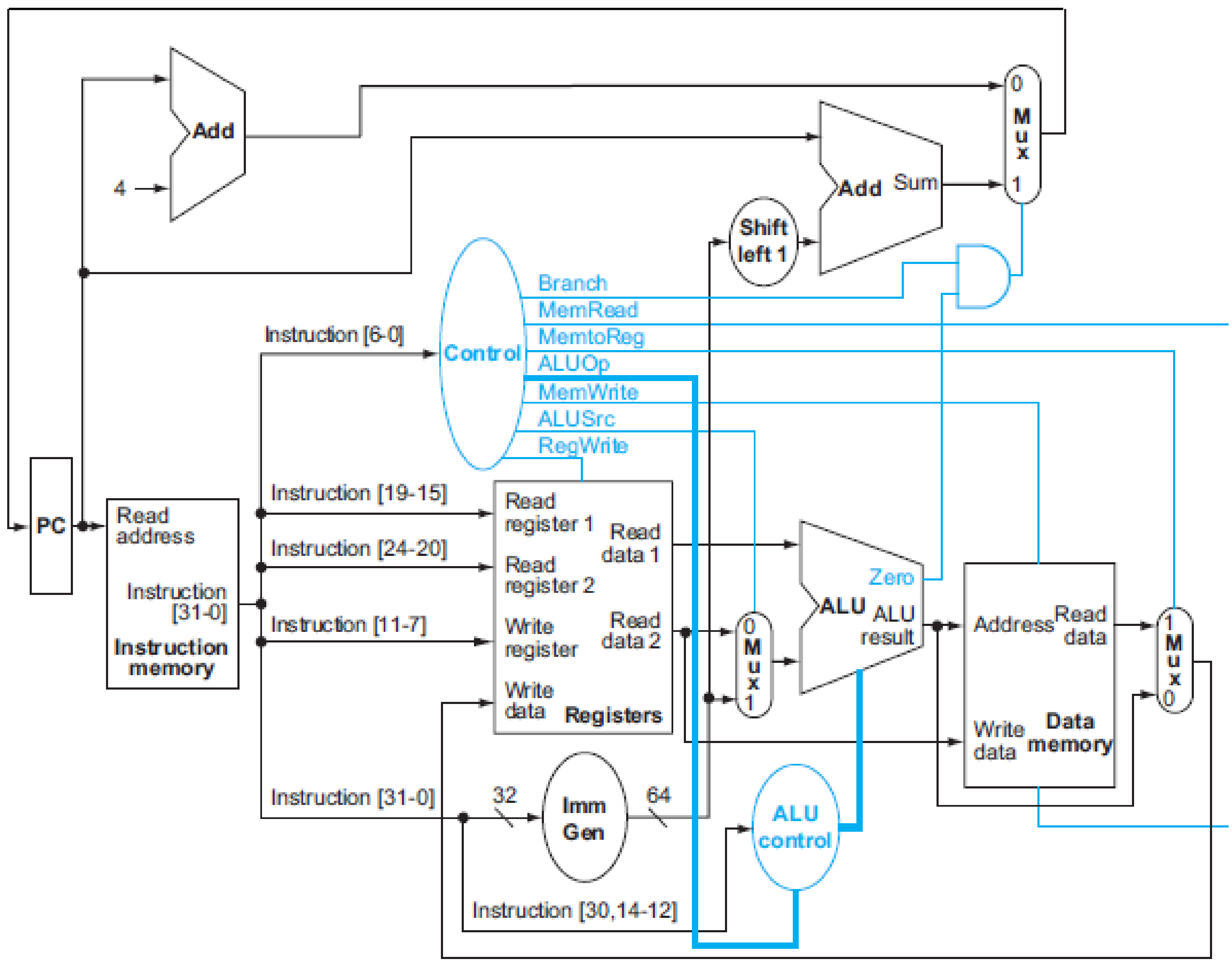
I-Type ALU Worksheet:



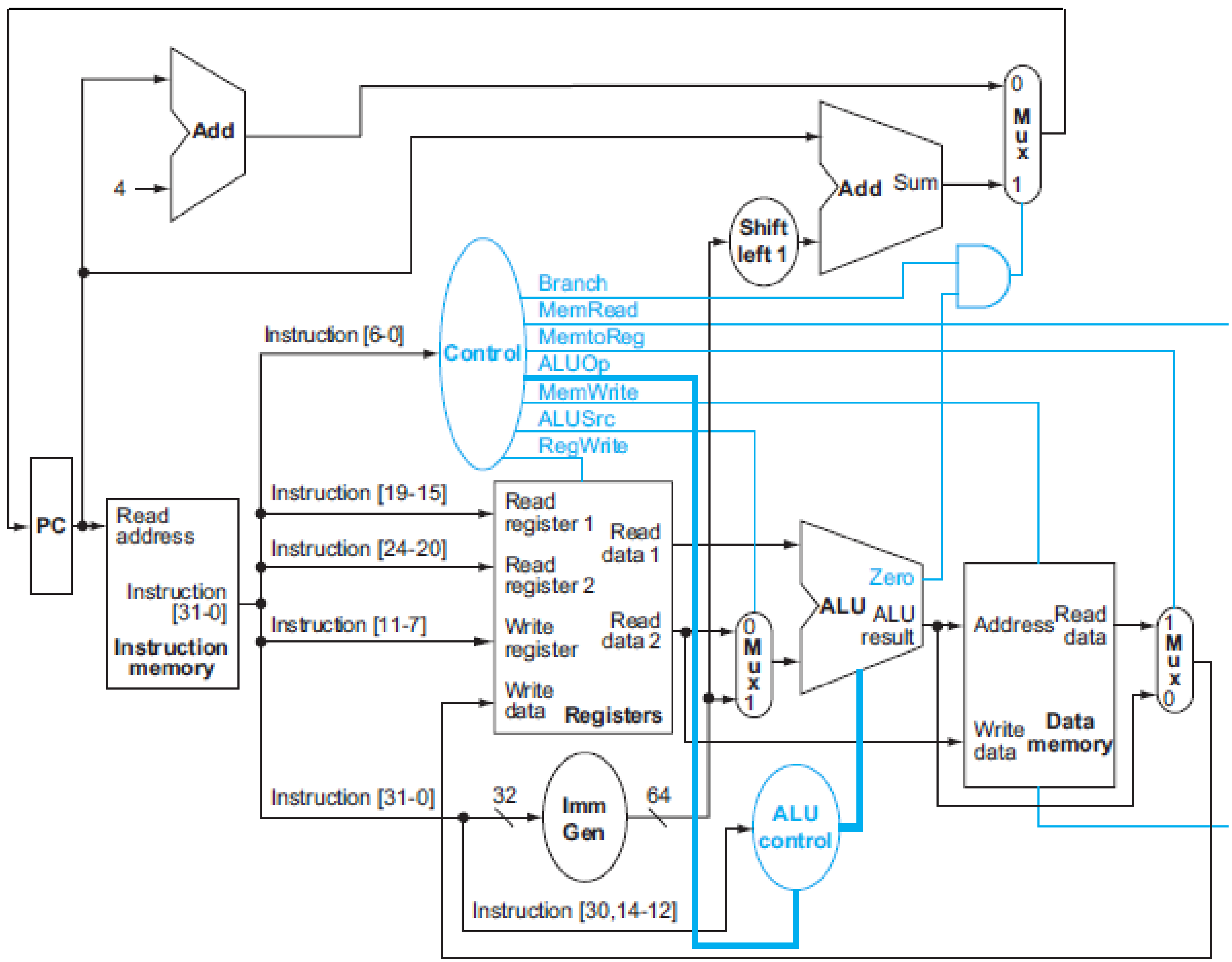
LW Worksheet:



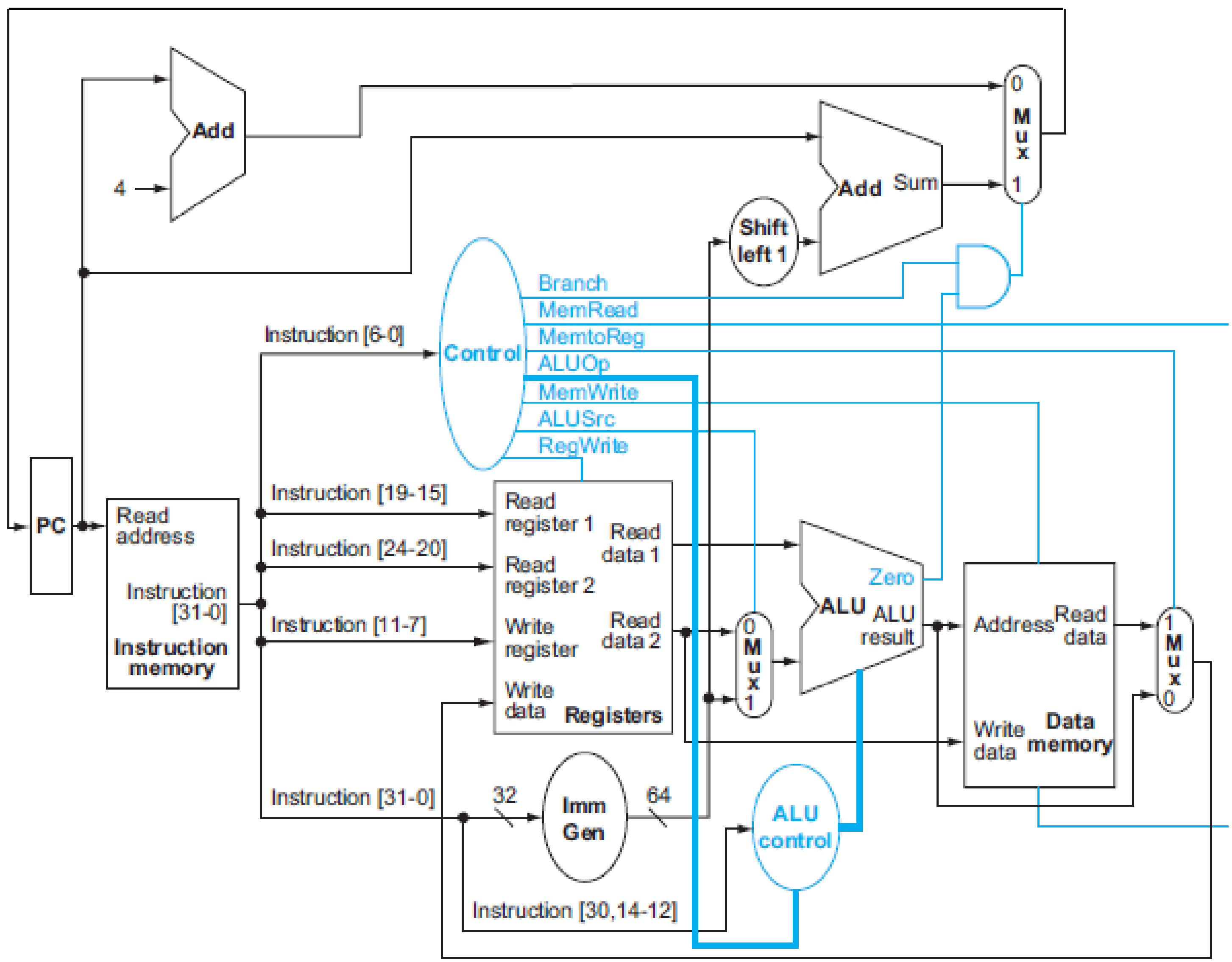
SW Worksheet:



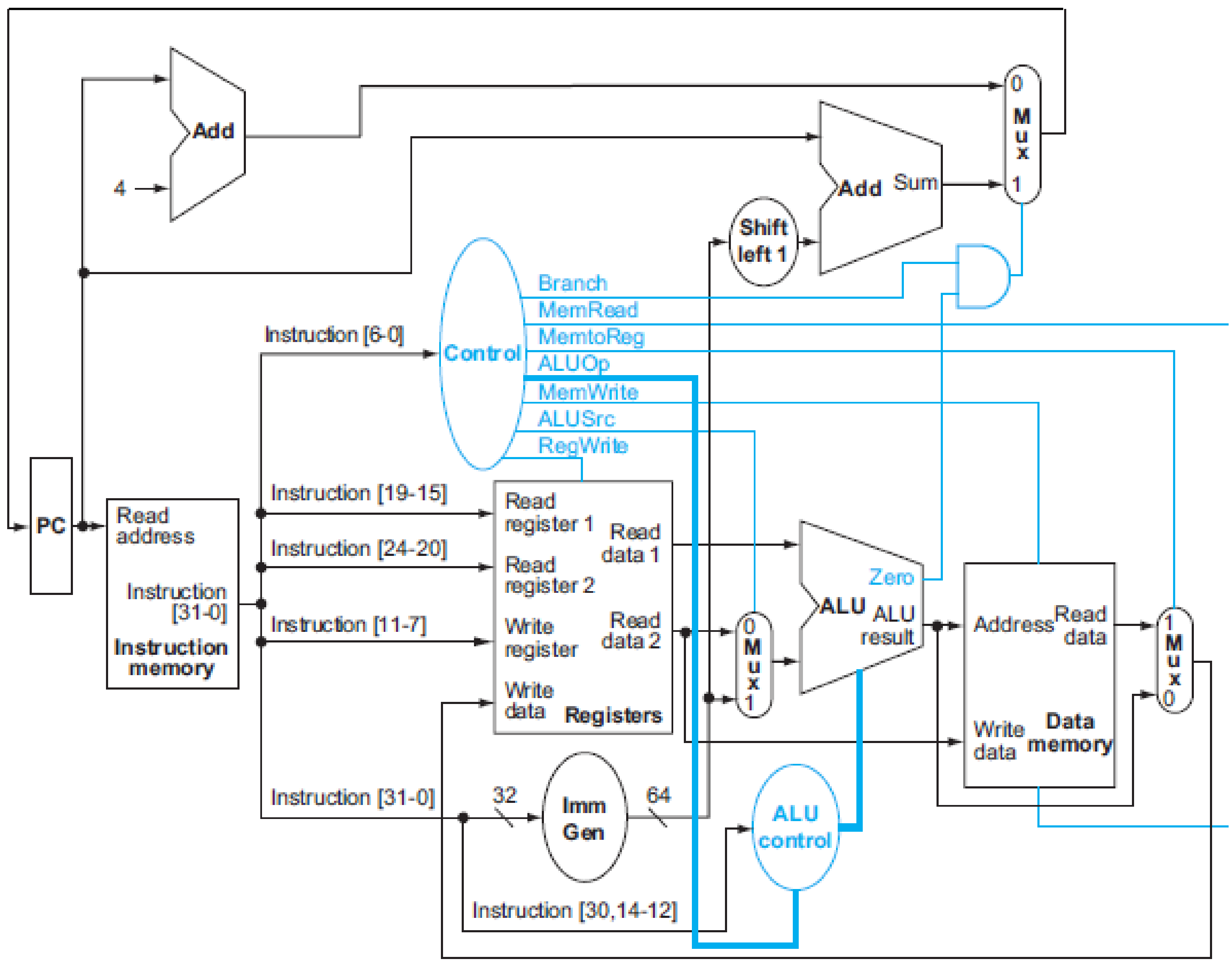
Branch Taken Worksheet:



Branch Not-Taken Worksheet:



Jump (and Link?) ALU Worksheet:



Single-Bit Control Signals

	When De-asserted	When asserted	Equation
UseImm	2^{nd} ALU input from 2^{nd} GPR read port	2^{nd} ALU input from immediate	(opcode!=IsRtype) && (opcode!=isBtype)
RFWrite	GPR write disabled	GPR write enabled	(opcode!=SW) && (opcode!=Bxx)
MemtoRF	Steer ALU result to GPR write port	steer memory load to GPR write port	opcode==LW/H/B
PCtoRF	Steer above result to GPR write port	Steer PC+4 to GPR write port	(opcode==JAL) || (opcode==JALR)
MemRead	Memory read disabled	Memory read port return load value	opcode==LW/H/B
MemWrite	Memory write disabled	Memory write enabled	opcode==SW/H/B

Multi-Bit Control Signals

	Options	Equation
ALU Op	• ADD, SUB, AND, OR, XOR, NOR, LT, and Shift • bcond: EQ, NE, GE, LT	case opcode RTypeALU: according to funct3, funct7[5] ITypeALU : according to funct3 only (except shift) LW/SW/JALR : ADD Bxx : SUB and select bcond function __ : ??
ImmExtend	Itype, ItypeU, Stype, SBtype, Utype, UJtype	• select based on instruction format type • (may want to have separate extension units for primary ALU and PC-offset adder)
PCSrc	PC+4, PCadder, ALU	case opcode JAL : PC + immediate JALR : GPR + immediate Bxx : taken?(PC + immediate):(PC + 4) __ : PC+4
LoadExtend	W,H,HU,B,BU	case func3 . . . .

Architecture vs Microarchitecture

Architecture

- Architectural Level
  - a clock has a hour hand and a minute hand, ...
  - a computer does ....????....

  You can read a clock without knowing how it works

conceptual

μarchitecture

- Microarchitecture Level
  - a particular clockwork has a certain set of gears arranged in a certain configuration
  - a particular computer design has a certain datapath and a certain control logic
- Realization Level
  - machined alloy gears vs stamped sheet metal
  - CMOS vs ECL vs vacuum tubes

physical

[Computer Architecture, Blaauw and Brooks, 1997]