

18-447 Lecture 4: Single-Cycle Microarchitecture

James C. Hoe

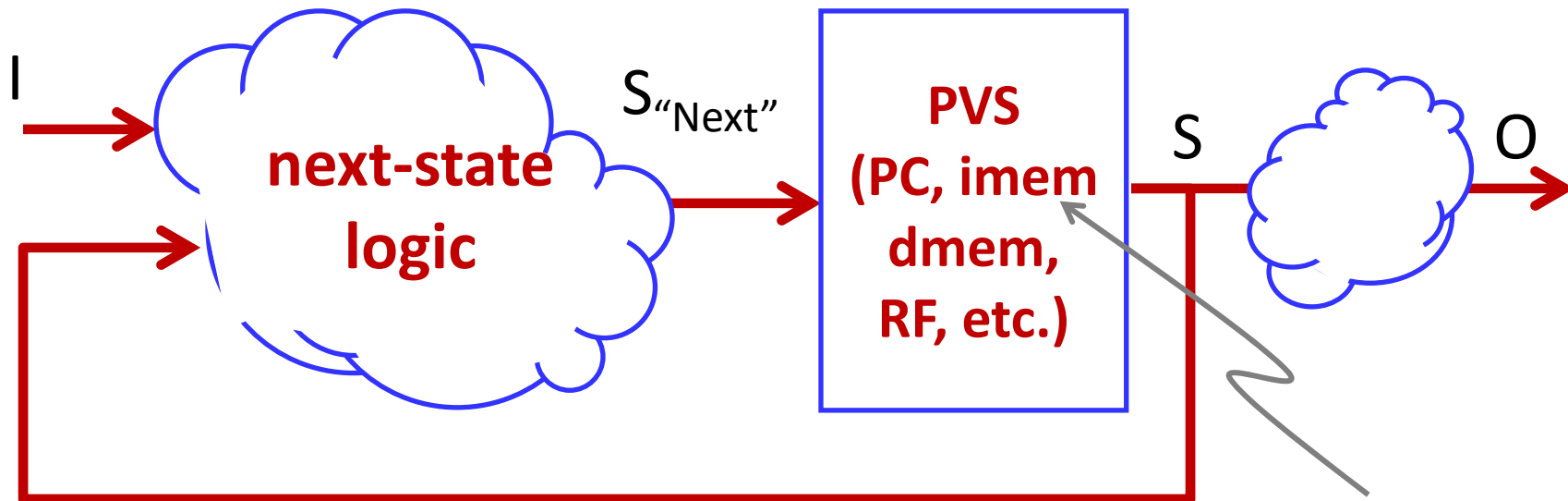
Department of ECE

Carnegie Mellon University

Housekeeping

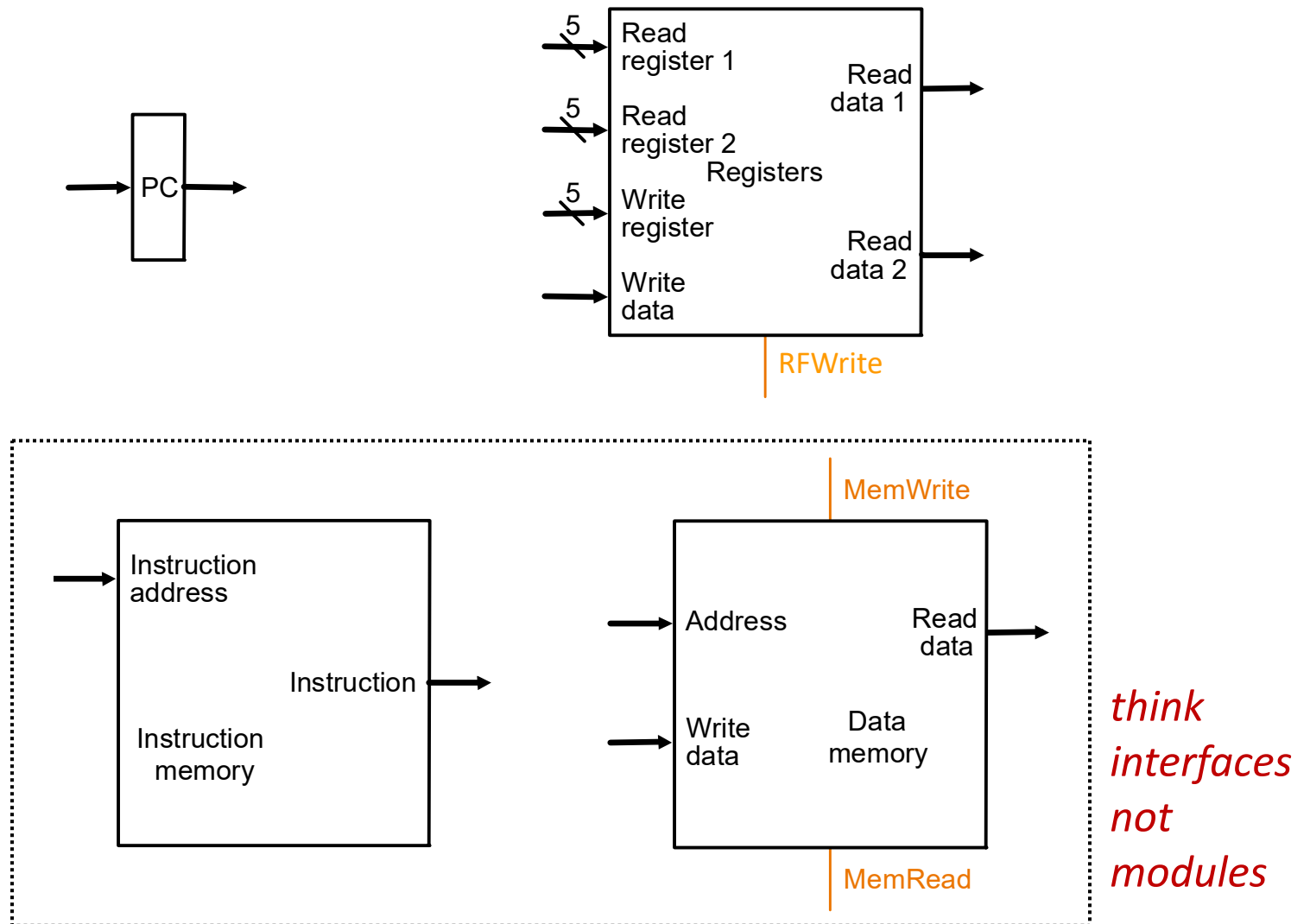
- Your goal today
 - first (correct but not good) try at implementing the RV32I ISA
- Notices
 - Student survey on Canvas, **due Wednesday**
 - Handout #4: HW1, **due noon 2/5**
 - Lab 1, Part A, **due Week 3**
 - Lab 1, Part B, **due Week 4**
- Readings
 - P&H Ch 4.1~4.4 (**for today**); Ch 1 (**for next time**)
 - finish reading P&H Ch 2 (**before Monday**)

Instruction Processing FSM



- An ISA describes an abstract FSM
 - state = programmer-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

Programmer-Visible State (aka Architectural State)



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

“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

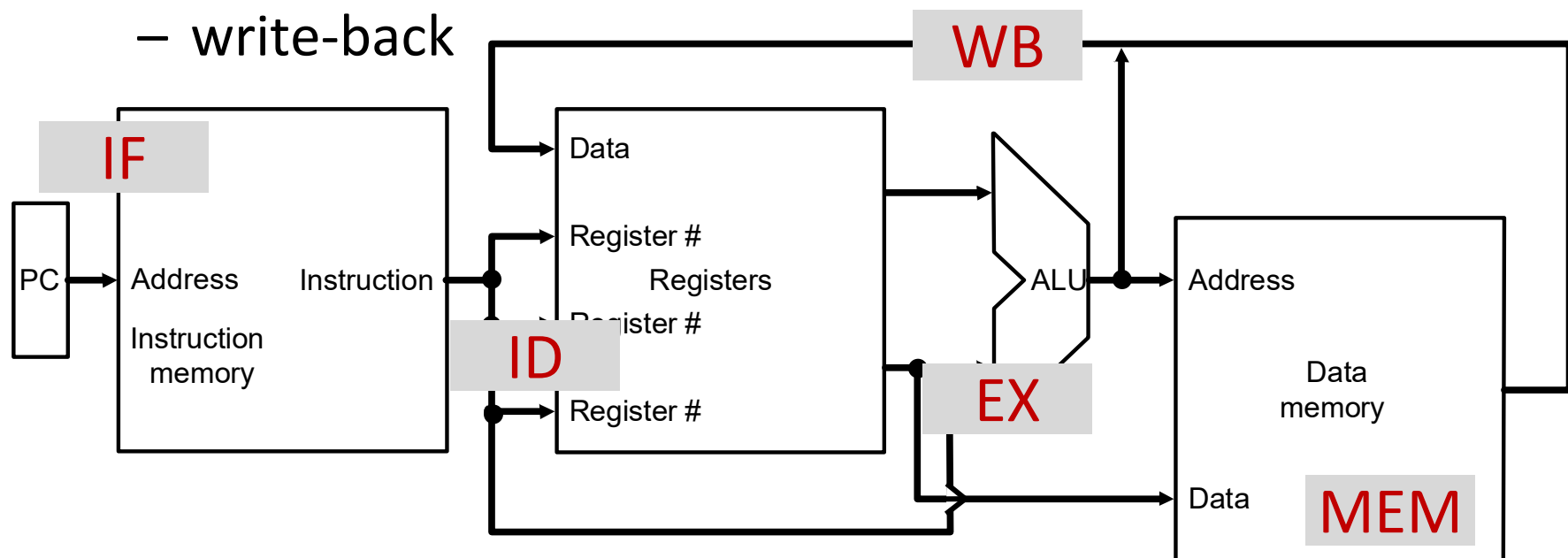
Simplifying Characteristics of “RISC”

- Simple ALU 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 uniform steps regardless of opcode
 - 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: R-type

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

- Semantics

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

what if rd is x0??

– $PC \leftarrow PC + 4$

- Exceptions: none (ignore carry and overflow)

- Variations

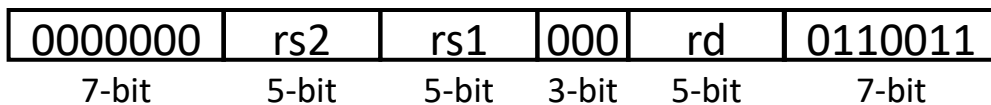
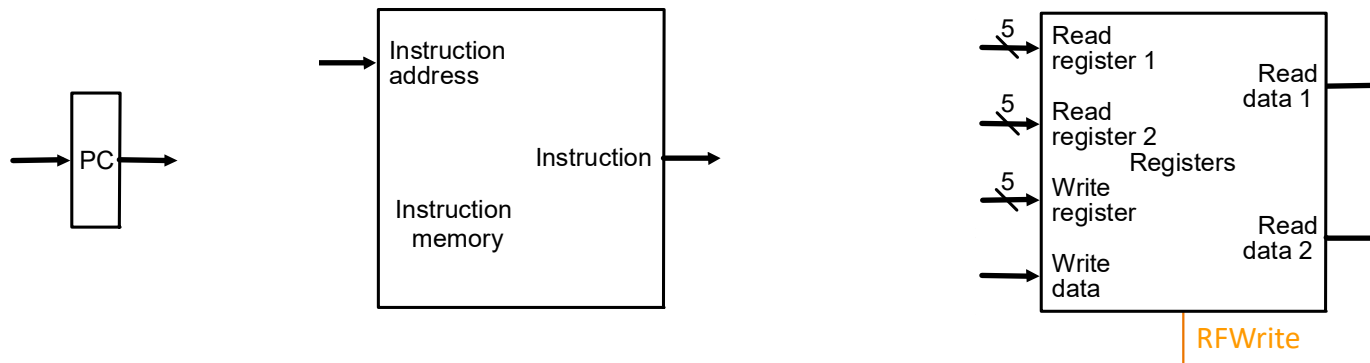
– Arithmetic: {ADD, SUB}

– Compare: {signed, unsigned} set if less than

– Logical: {AND, OR, XOR}

– Shift: {Left, Right-Logical, Right-Arithmetic}

ADD rd rs1 rs2

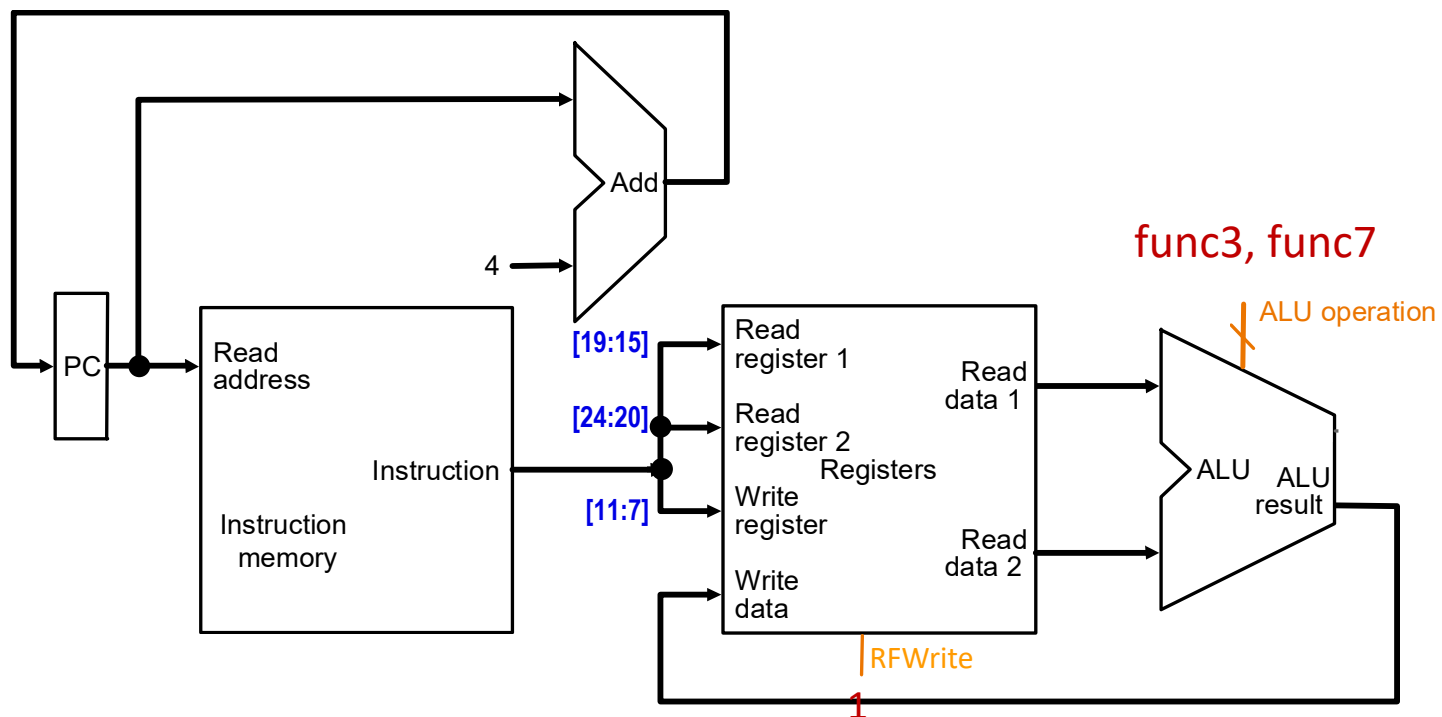


if MEM[PC] == ADD rd rs1 rs2
 $GPR[rd] \leftarrow GPR[rs1] + GPR[rs2]$
 $PC \leftarrow PC + 4$



Combinational
state update logic

R-Type ALU Datapath



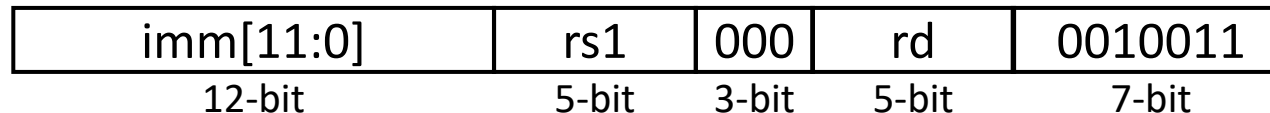
**Based on original figure from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Reg-Immediate ALU Instructions

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

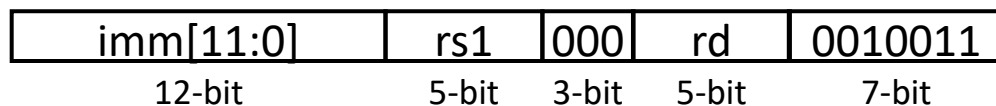
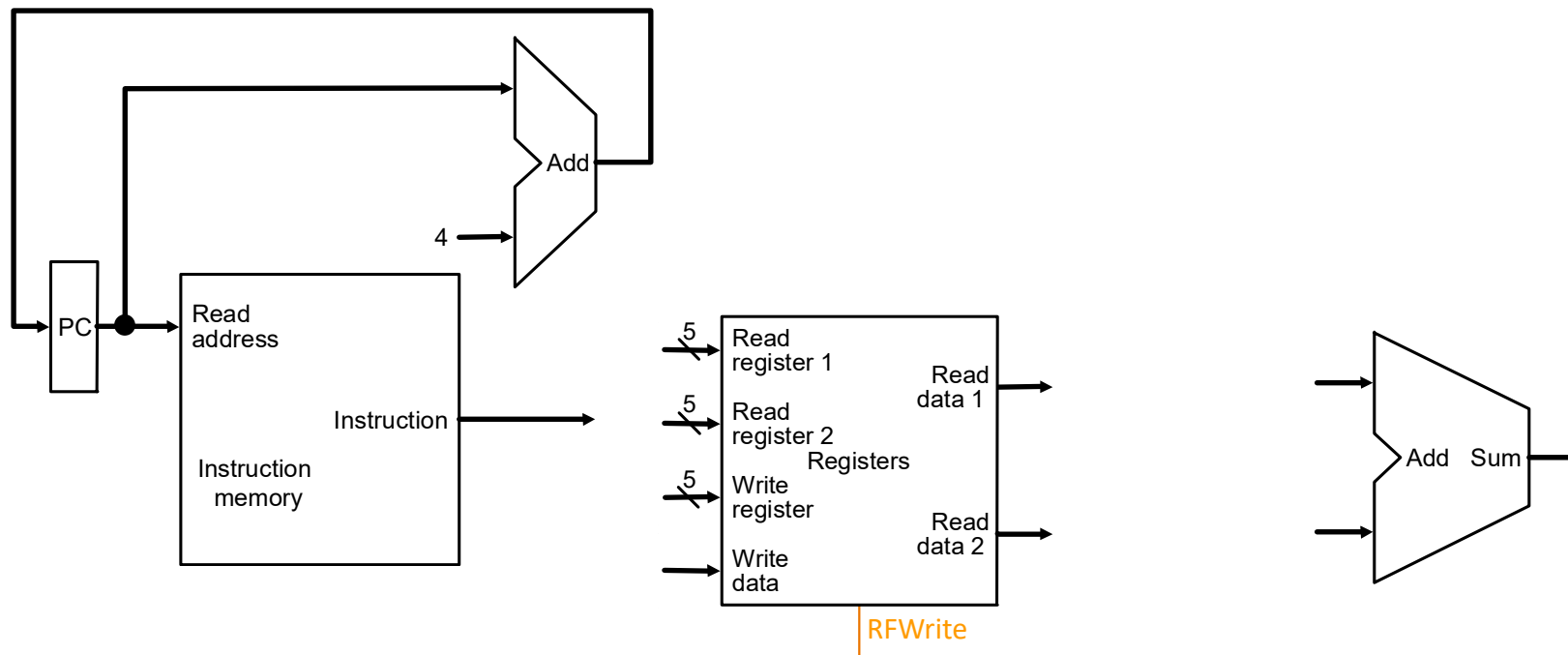
ADDI rd, rs1, imm₁₂

- Machine encoding: I-type



- Semantics
 - $\text{GPR}[\text{rd}] \leftarrow \text{GPR}[\text{rs1}] + \text{sign-extend}(\text{imm})$
 - $\text{PC} \leftarrow \text{PC} + 4$
- Exceptions: none (ignore carry and overflow)
- Variations
 - Arithmetic: {ADDI, ~~SUBI~~}
 - Compare: {signed, unsigned} set if less than
 - Logical: {ANDI, ORI, XORI}
 - **Shifts by unsigned imm[4:0]: {SLLI, SRLI, SRAI}

ADDI rd rs1 immediate₁₂

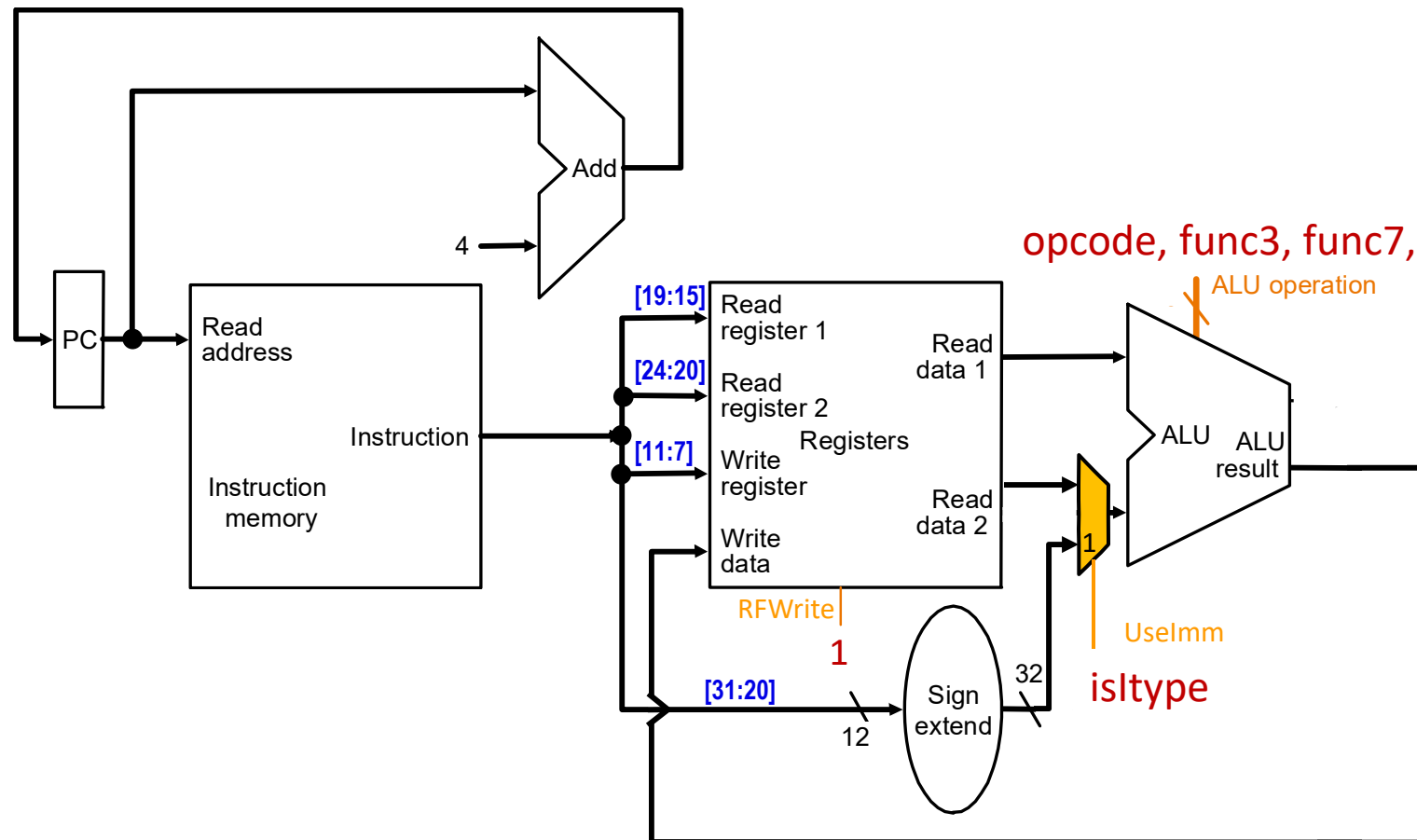


if MEM[PC] == ADDI rd rs1 immediate
 GPR[rd] ← GPR[rs1] + sign-extend (immediate)
 PC ← PC + 4



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 (i.e., Loads and Stores)

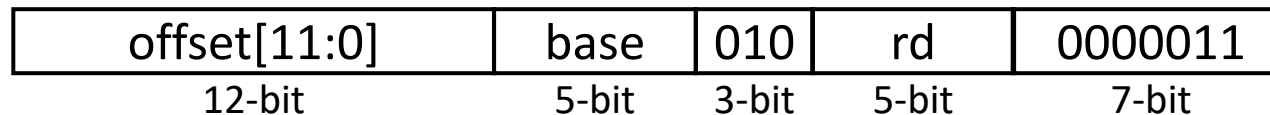
Load Instructions

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

`LW rd, offset12(base)`

rs1

- Machine encoding: I-type



- Semantics

- $\text{byte_address}_{32} = \text{sign-extend}(\text{offset}_{12}) + \text{GPR}[\text{base}]$
- $\text{GPR}[\text{rd}] \leftarrow \text{MEM}_{32}[\text{byte_address}]$
- $\text{PC} \leftarrow \text{PC} + 4$

- Exceptions: none for now

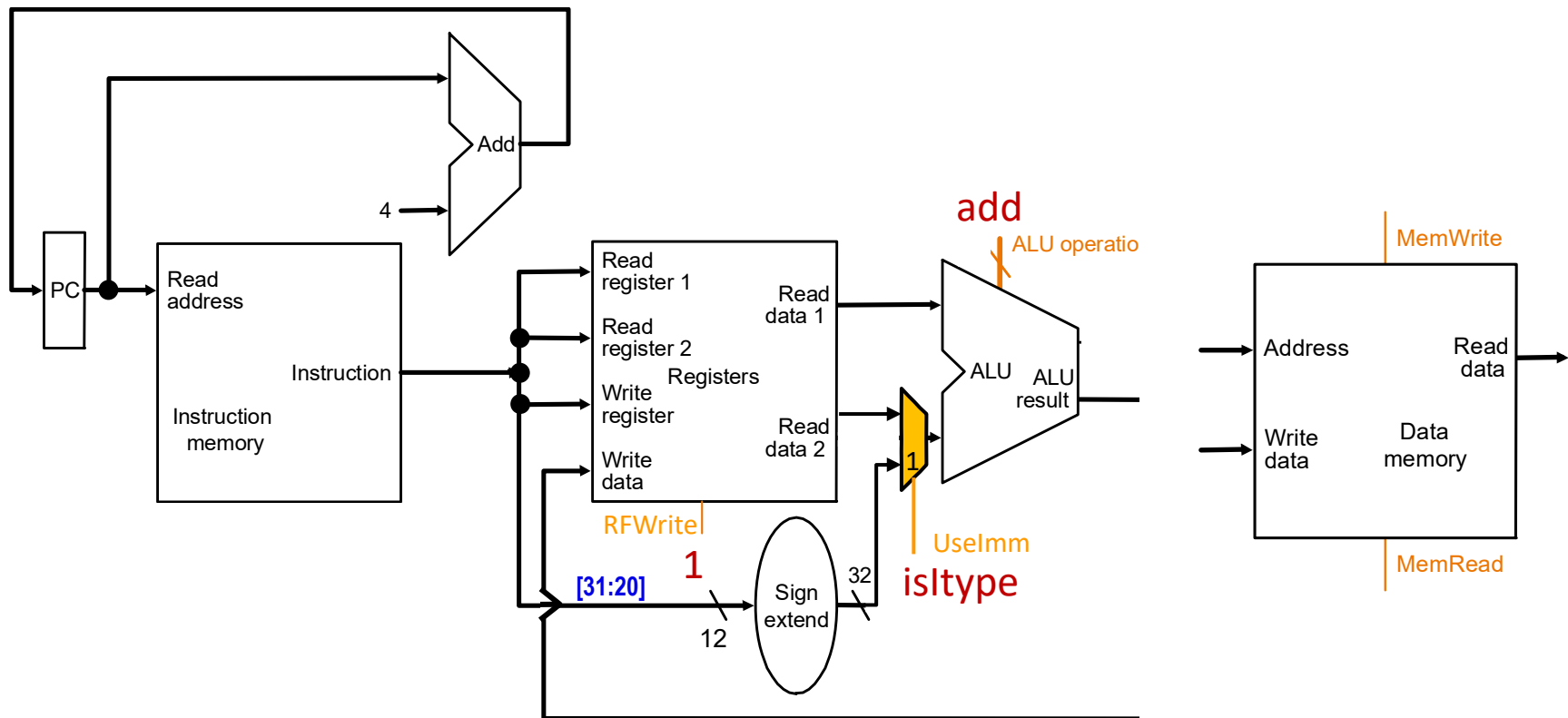
- Variations: LW, LH, LHU, LB, LBU

e.g., LB :: $\text{GPR}[\text{rd}] \leftarrow \text{sign-extend}(\text{MEM}_8[\text{byte_address}])$

LBU :: $\text{GPR}[\text{rd}] \leftarrow \text{zero-extend}(\text{MEM}_8[\text{byte_address}])$

RV32I is byte-addressable, little-endian (until v20191213)

LW Datapath



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

IF ID EX MEM WB

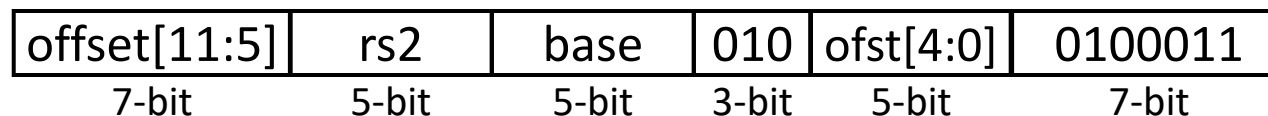
Combinational
state update logic

Store Instructions

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

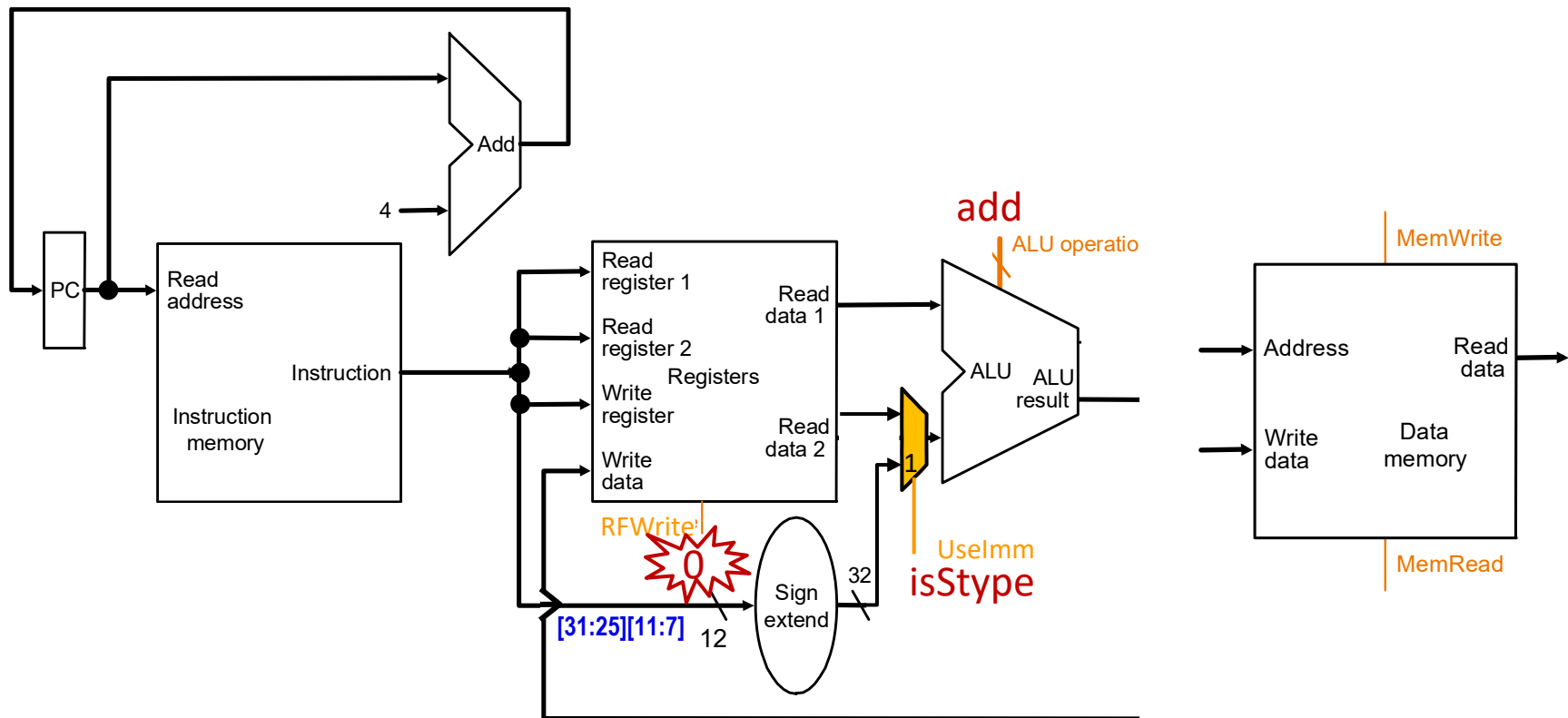
SW $rs2, offset_{12}(base)$

- Machine encoding: S-type



- Semantics
 - $byte_address_{32} = \text{sign-extend}(offset_{12}) + \text{GPR}[base]$
 - $MEM_{32}[byte_address] \leftarrow \text{GPR}[rs2]$
 - $PC \leftarrow PC + 4$
- Exceptions: none for now
- Variations: SW, SH, SB
 - e.g., SB:: $MEM_8[byte_address] \leftarrow (\text{GPR}[rs2])[7:0]$

SW Datapath



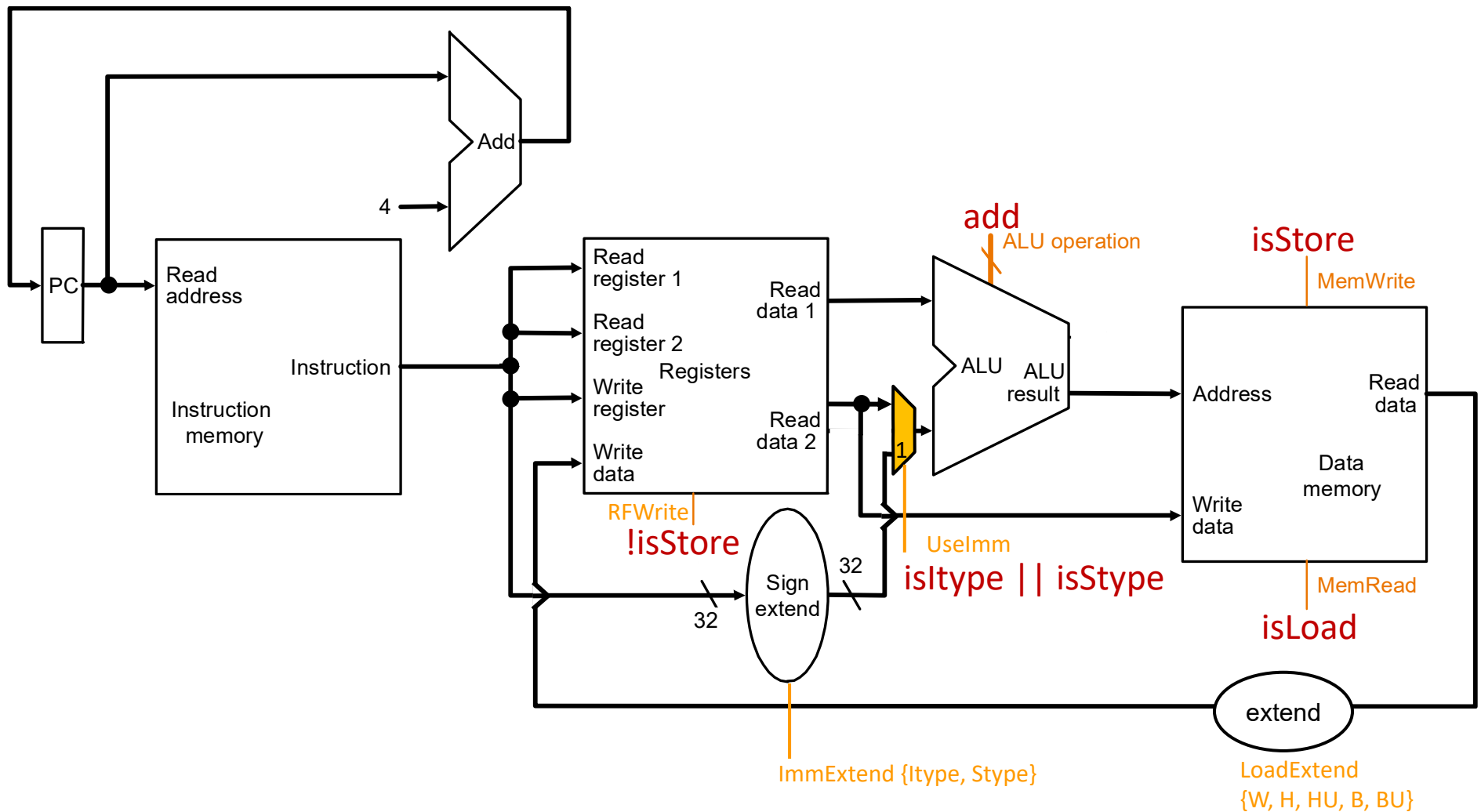
if MEM[PC]==SW rs2 offset₁₂(base)
 EA = sign-extend(offset) + GPR[base]
 MEM[EA] ← GPR[rs2]
 PC ← PC + 4

IF ID EX MEM WB

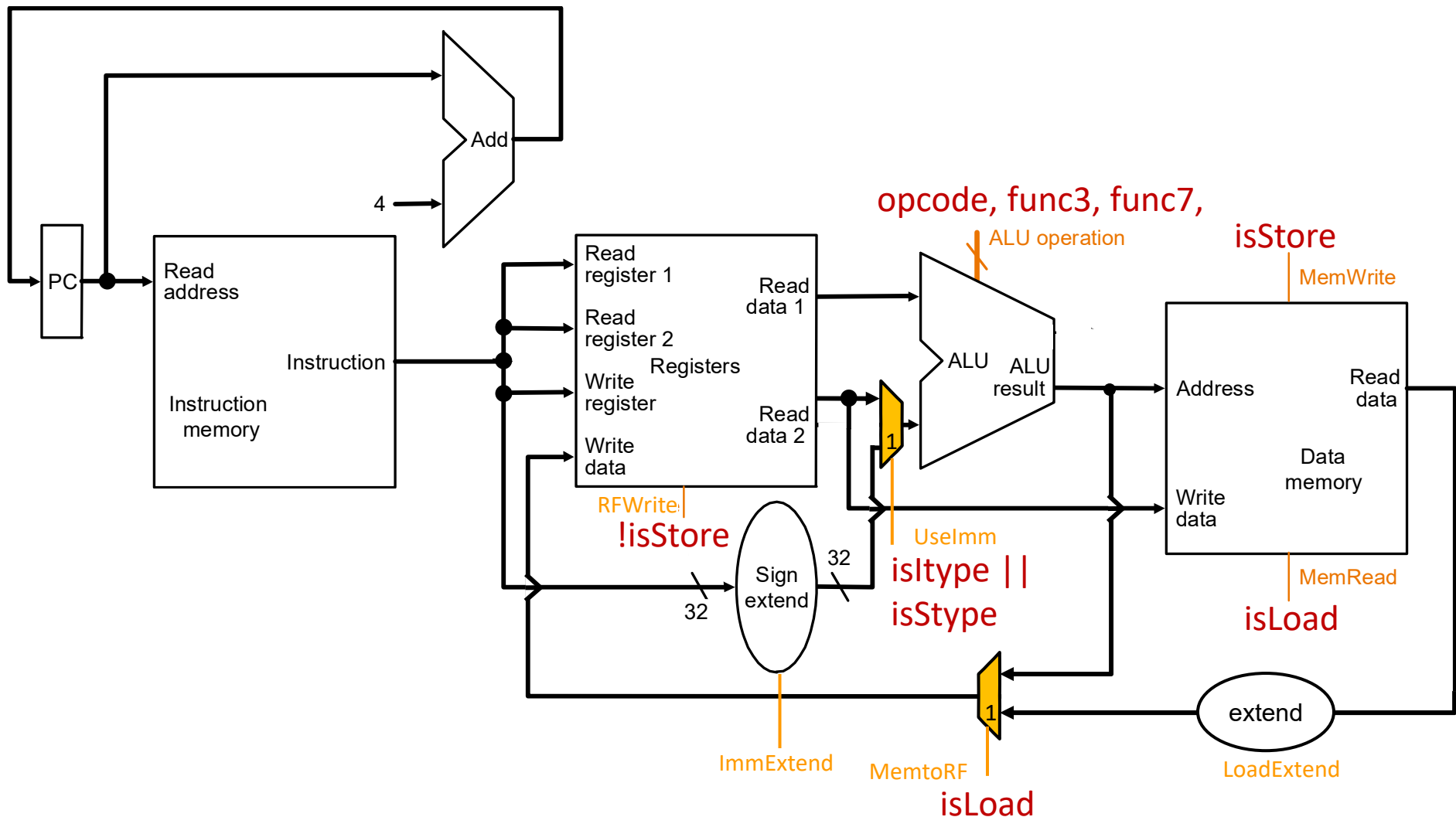


Combinational
state update logic

Load-Store Datapath



Datapath for Non-Control Flow Inst's



Single-Cycle Datapath for Control Flow Instructions

Jump and Link Instruction

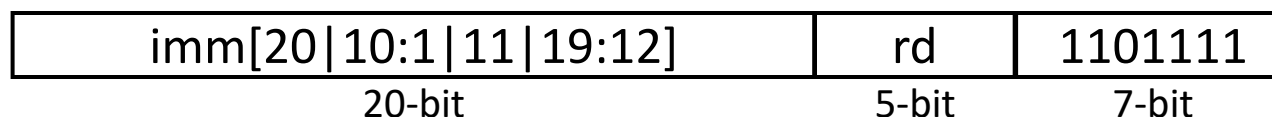
- Assembly

JAL rd imm₂₁

Note: implicit imm[0]=0

Note: real assembler expects a target label or address

- Machine encoding: J-type



- Semantics

– target = PC + sign-extend(**imm₂₁**)

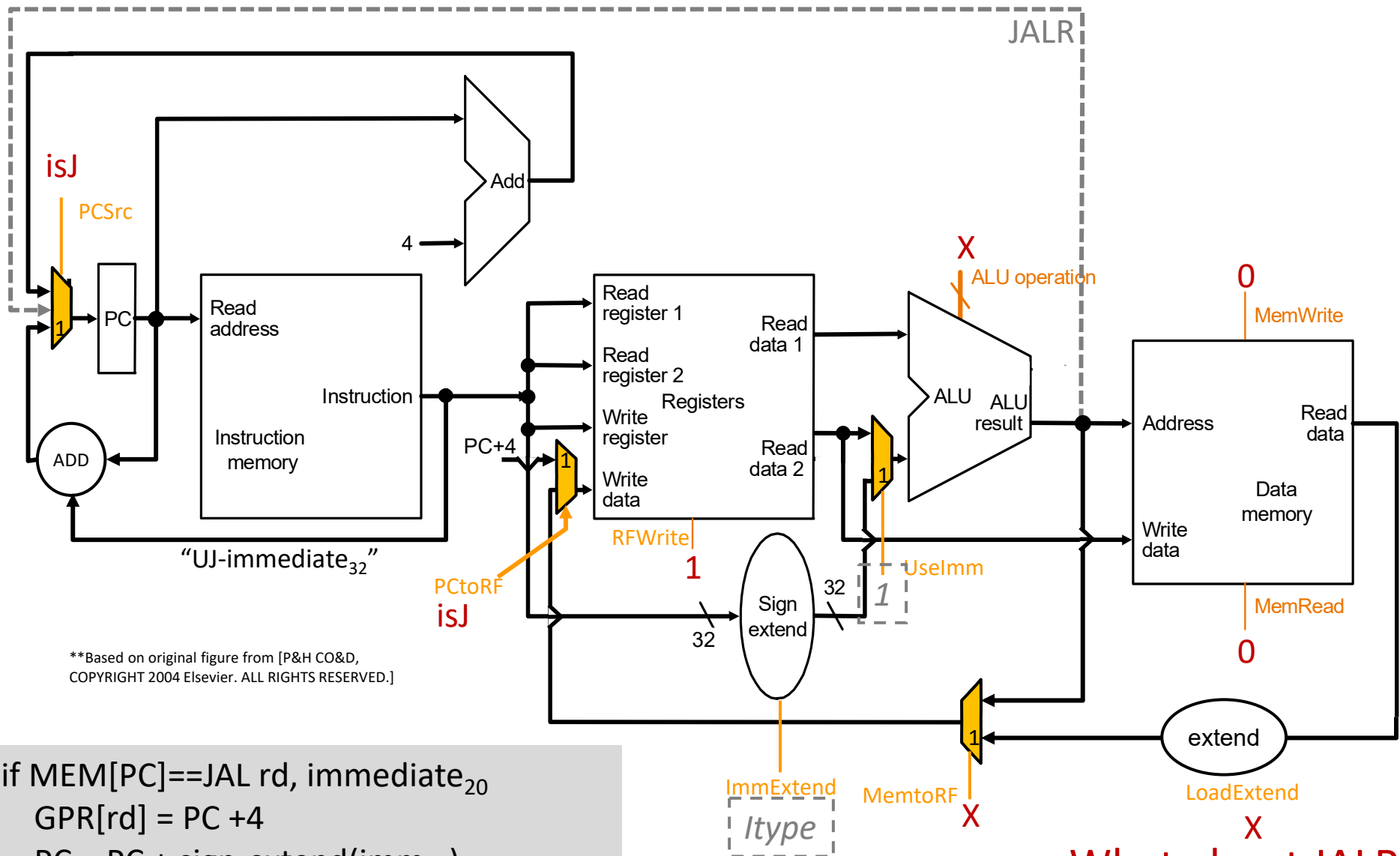
– GPR[**rd**] ← PC + 4

– PC ← target

How far can you jump?

- Exceptions: misaligned target (4-byte)

Unconditional Jump Datapath



What about JALR?

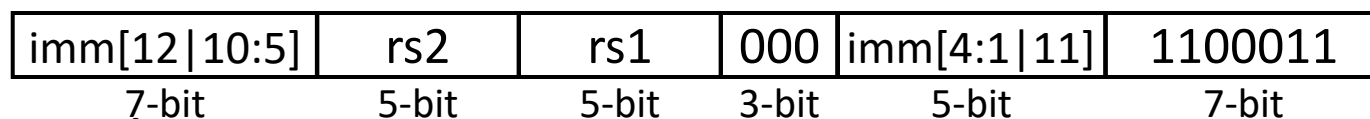
(Conditional) Branch Instructions

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

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

Note: real assembler expects a target label or address

- Machine encoding: B-type



- Semantics

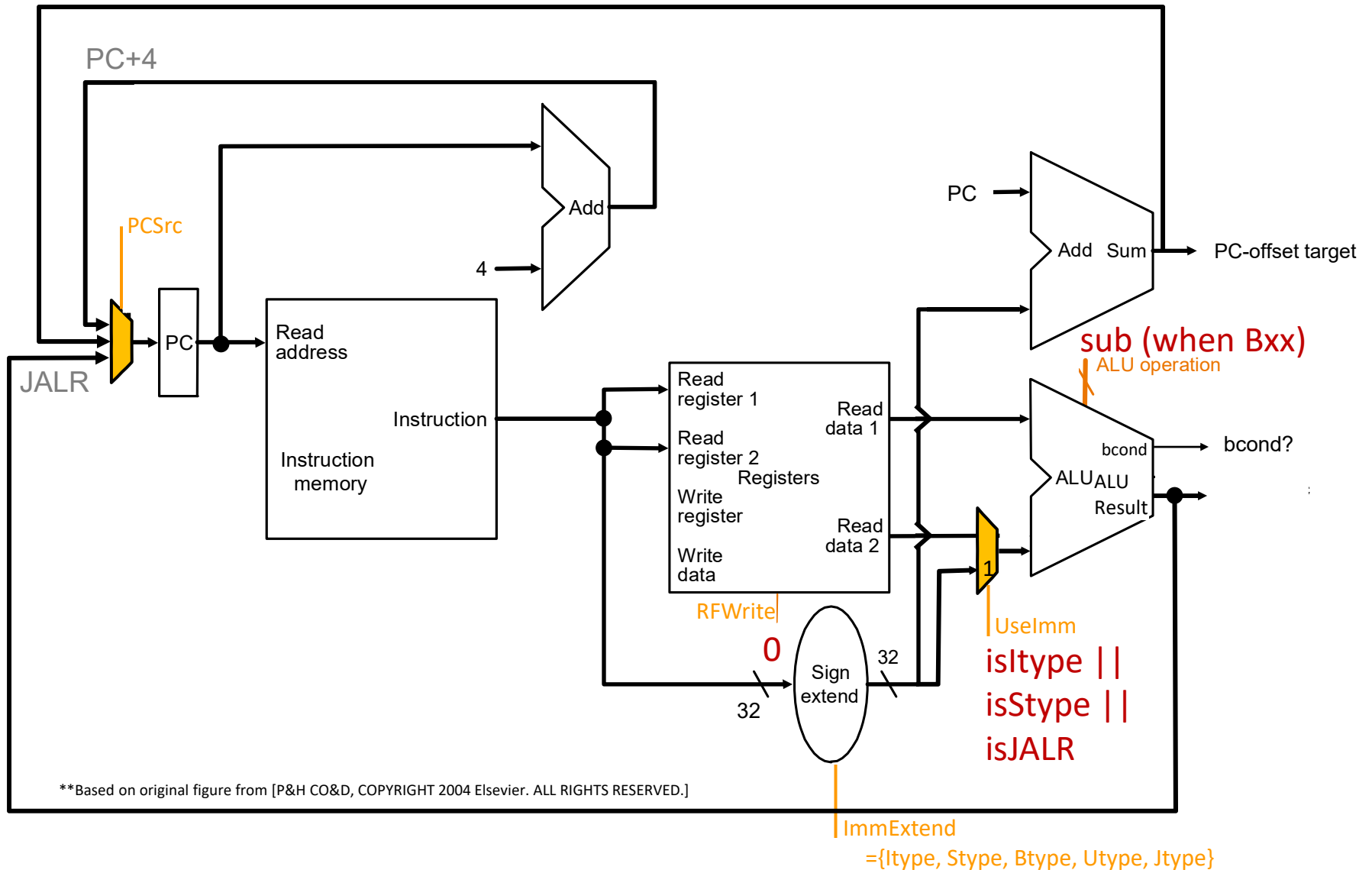
- target = PC + sign-extend(imm_{13})
- if $GPR[rs1] == GPR[rs2]$ then PC ← target
else PC ← PC + 4

How far can you jump?

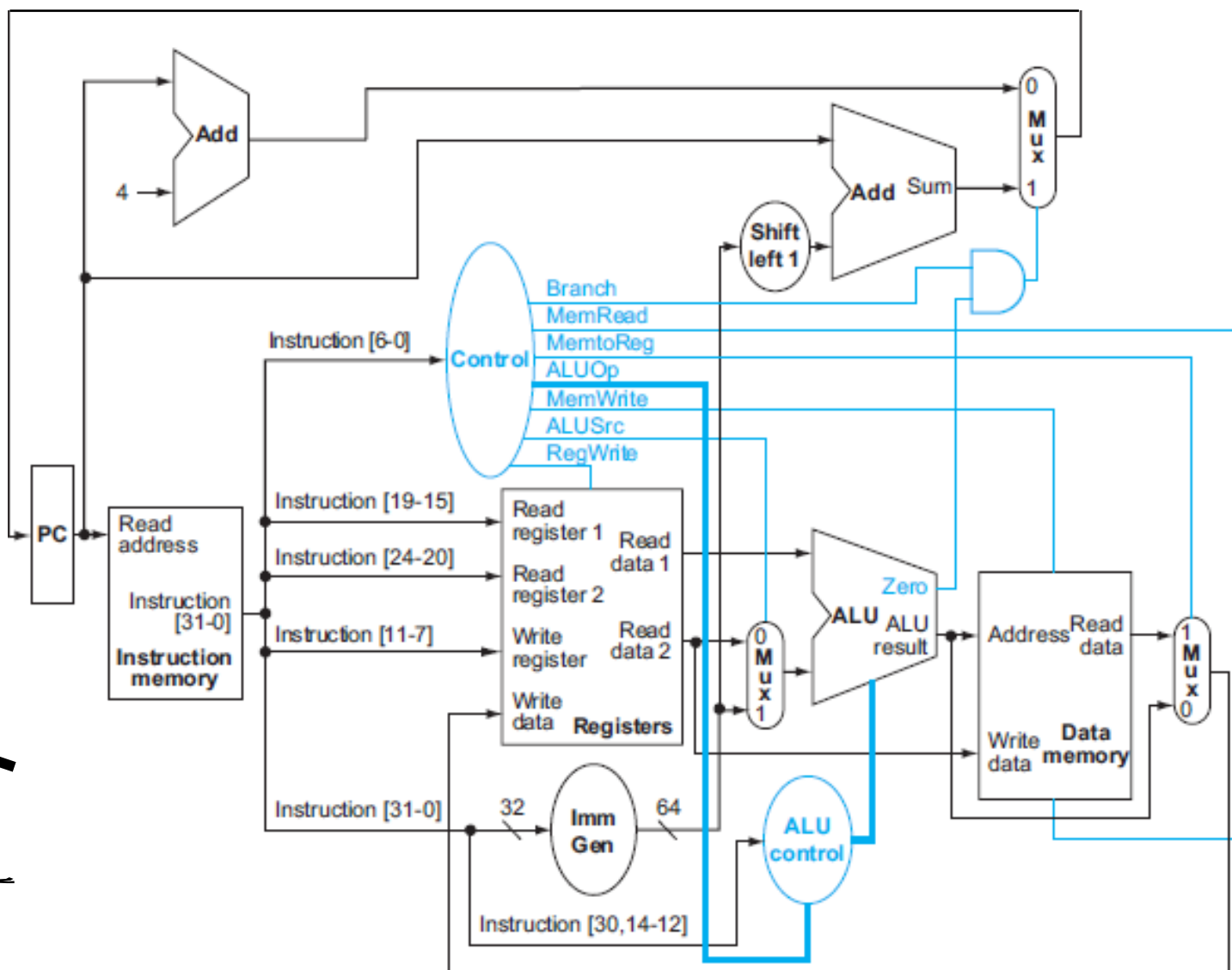
- Exceptions: misaligned target (4-byte) if taken
- Variations
 - BEQ, BNE, BLT, BGE, BLTU, BGEU

Conditional Branch Datapath

JAL and taken Branch

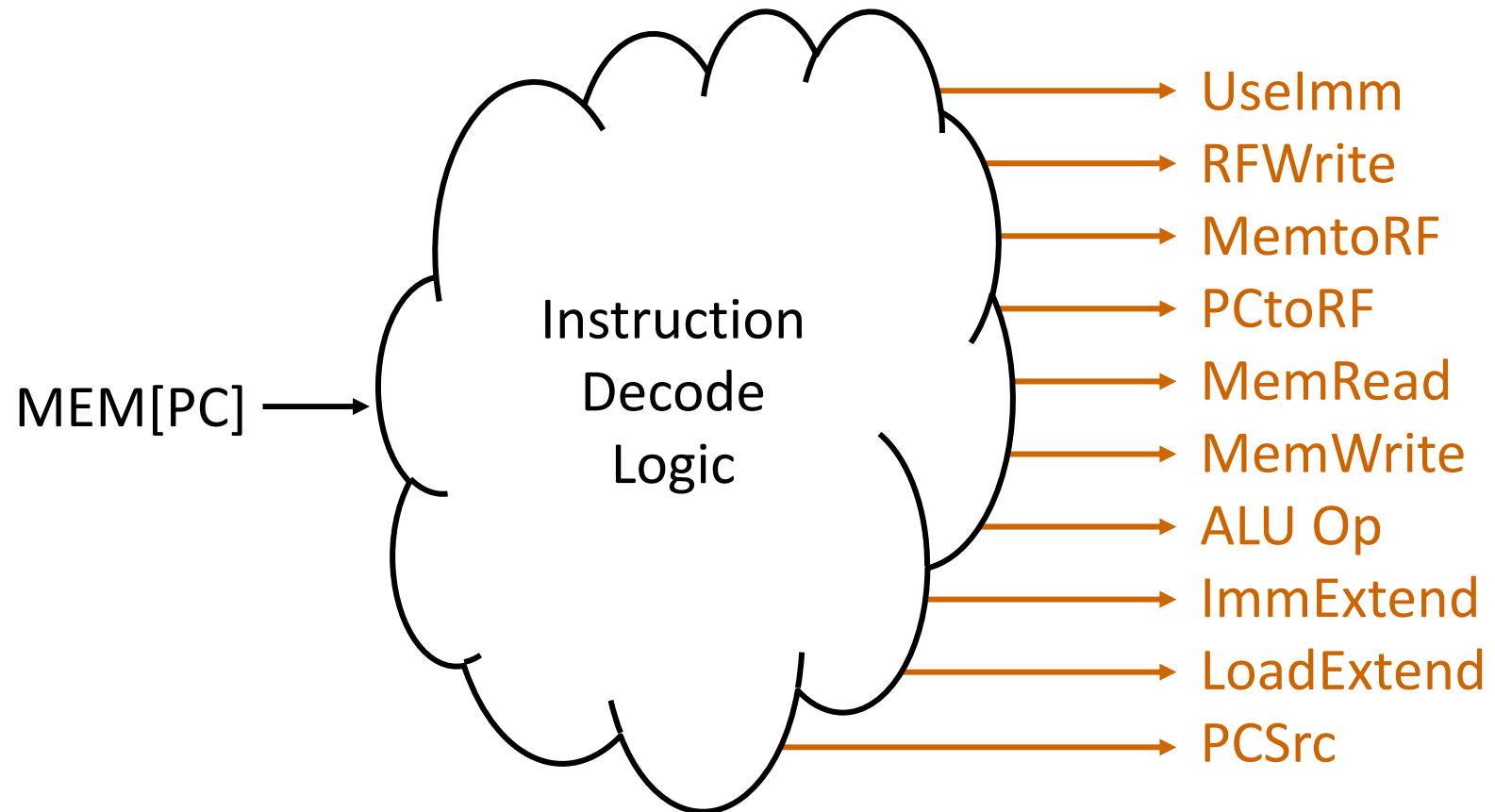


Adding Control to Datapath

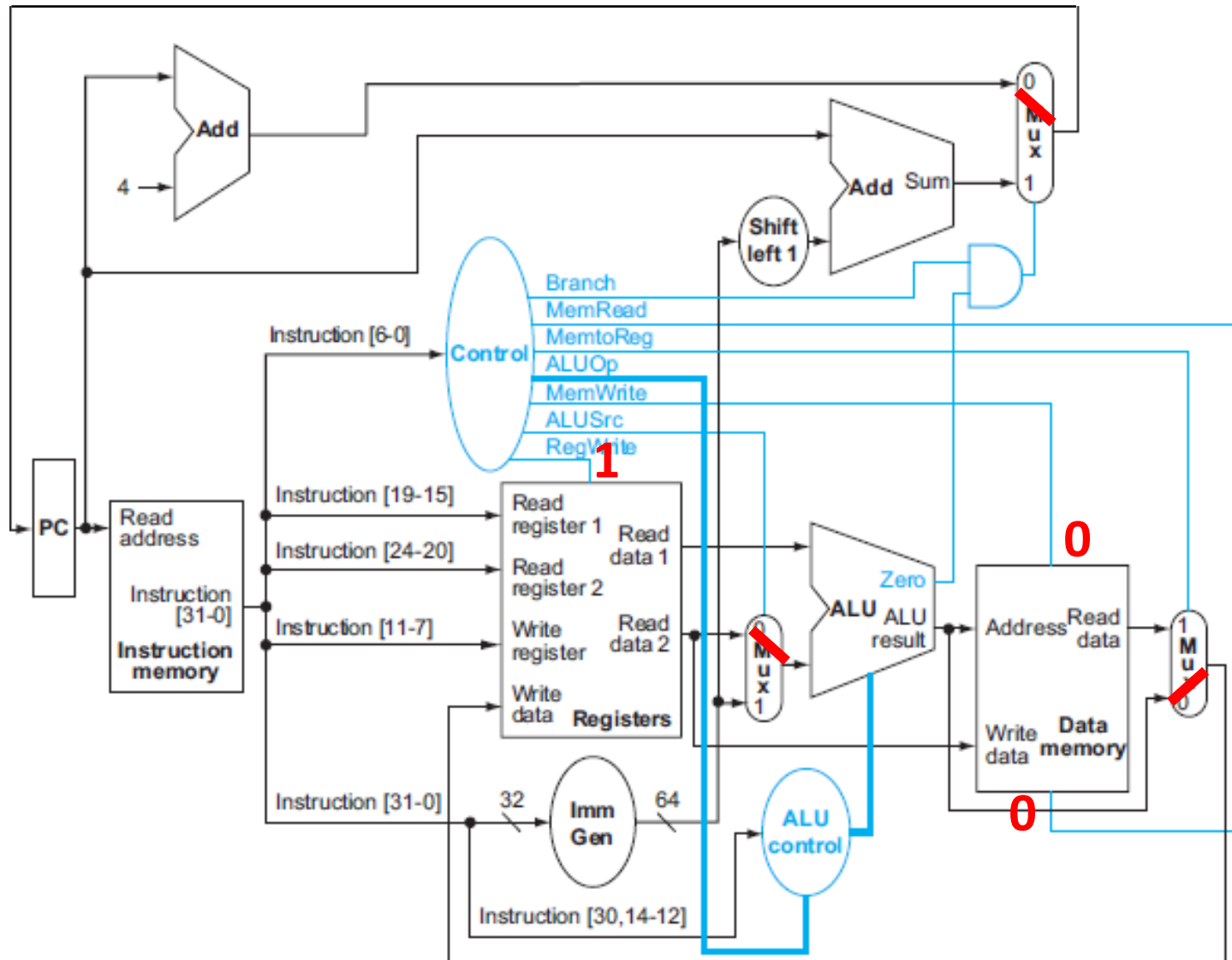


[Figure 4.17 from book, Copyright © 2018 Elsevier Inc. All rights reserved.]

Datapath Control Generation

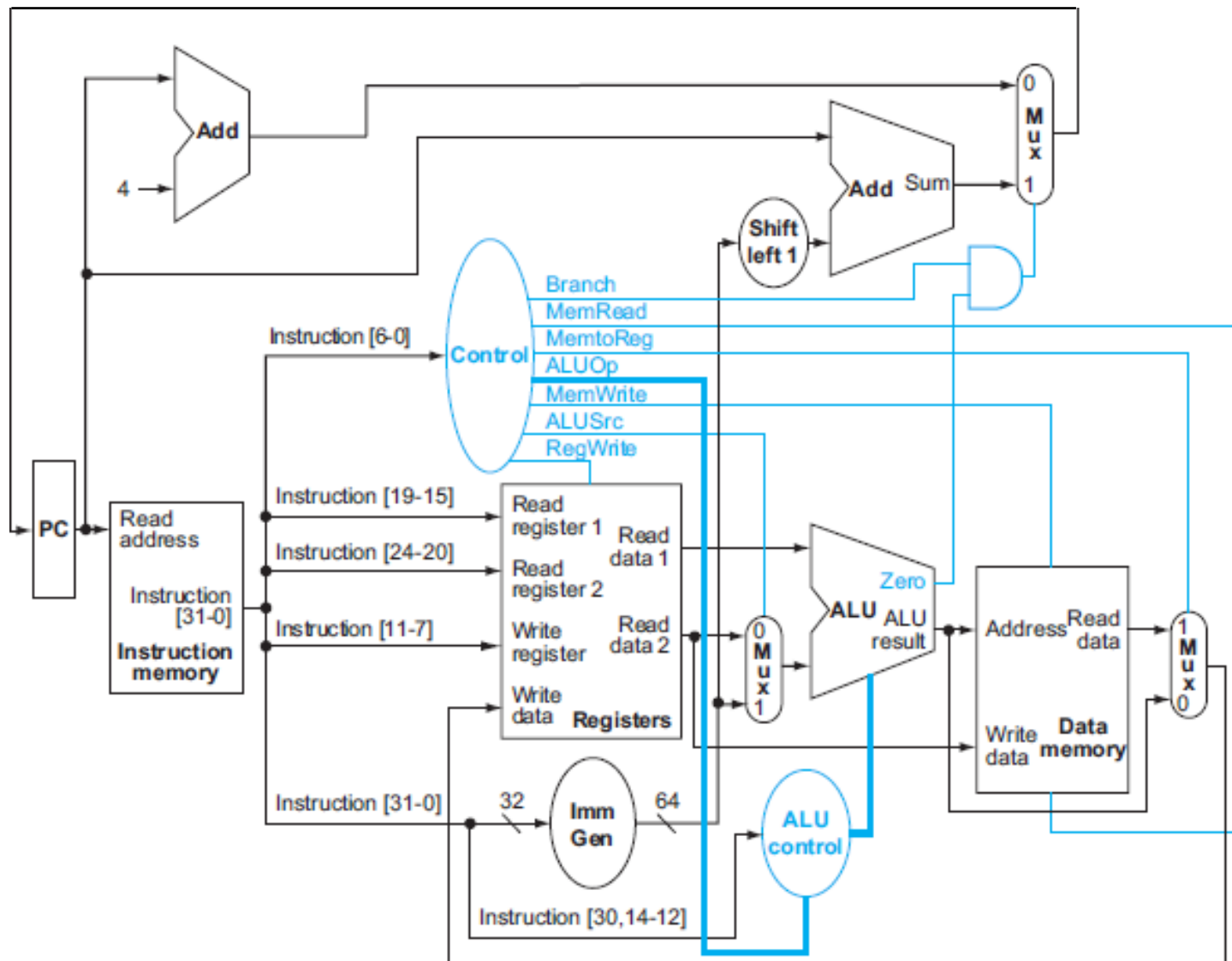


R-Type ALU Worksheet:



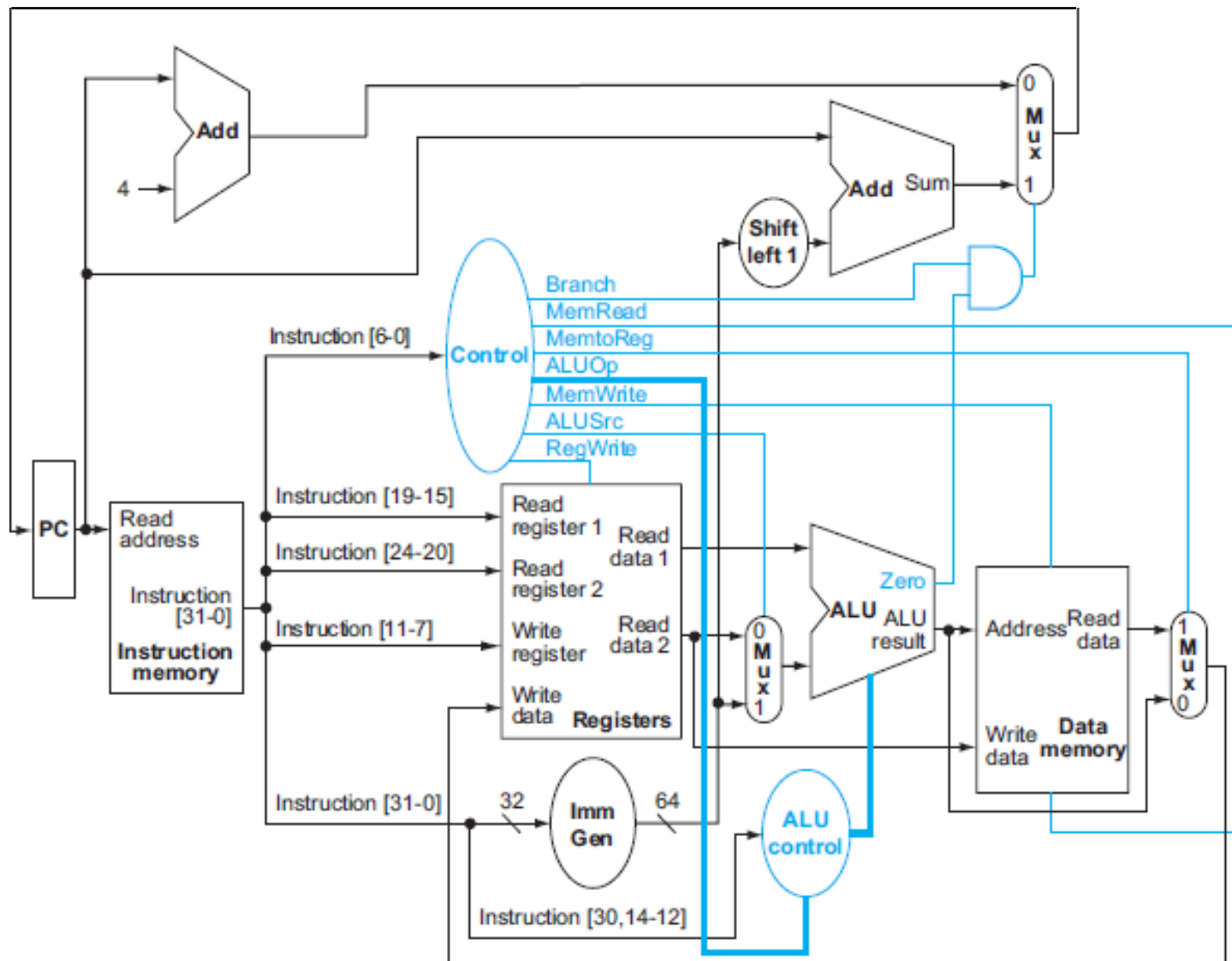
[Figure 4.17 from book, Copyright © 2018 Elsevier Inc. All rights reserved.]

I-Type ALU Worksheet:



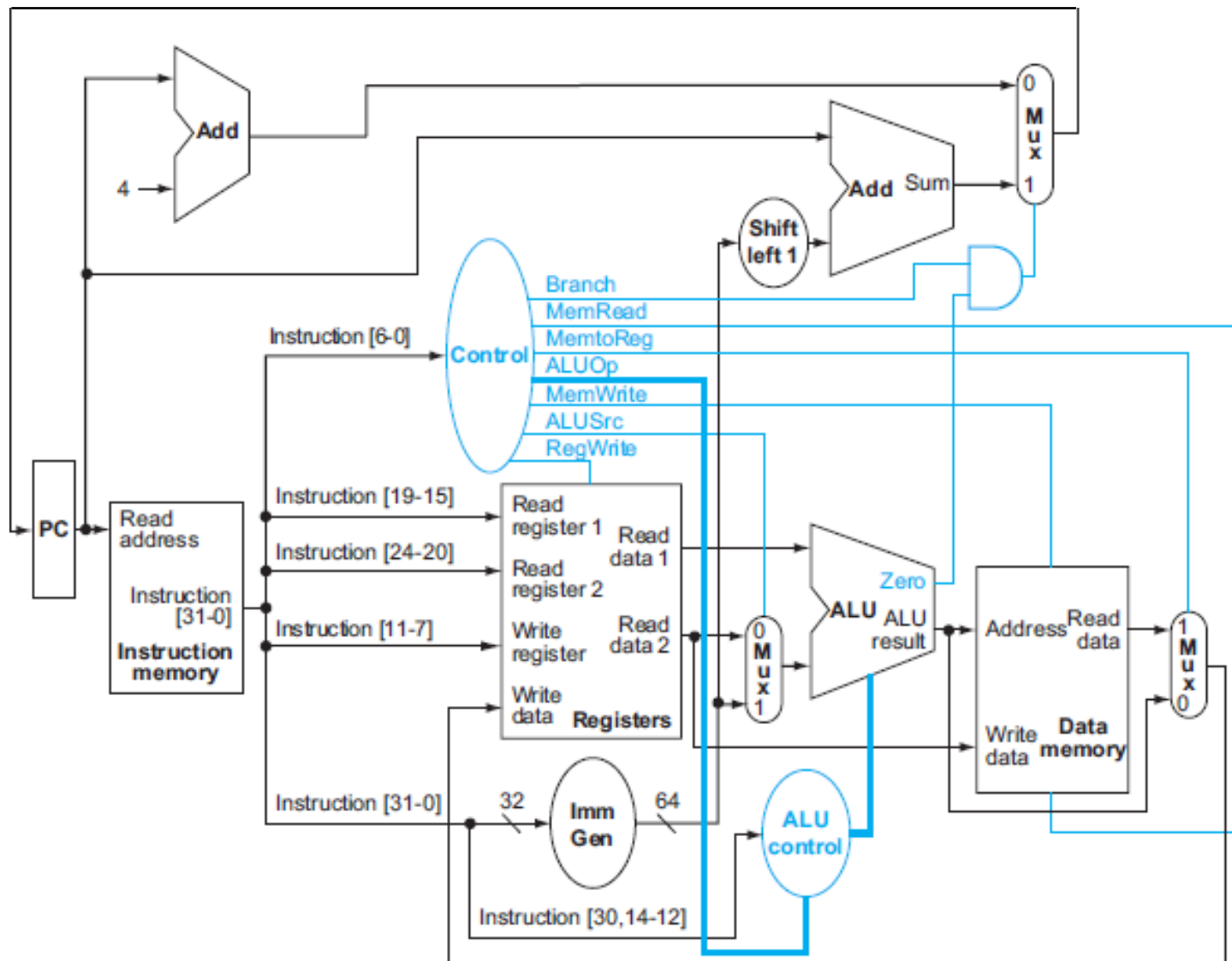
[Figure 4.17 from book, Copyright © 2018 Elsevier Inc. All rights reserved.]

LW Worksheet:



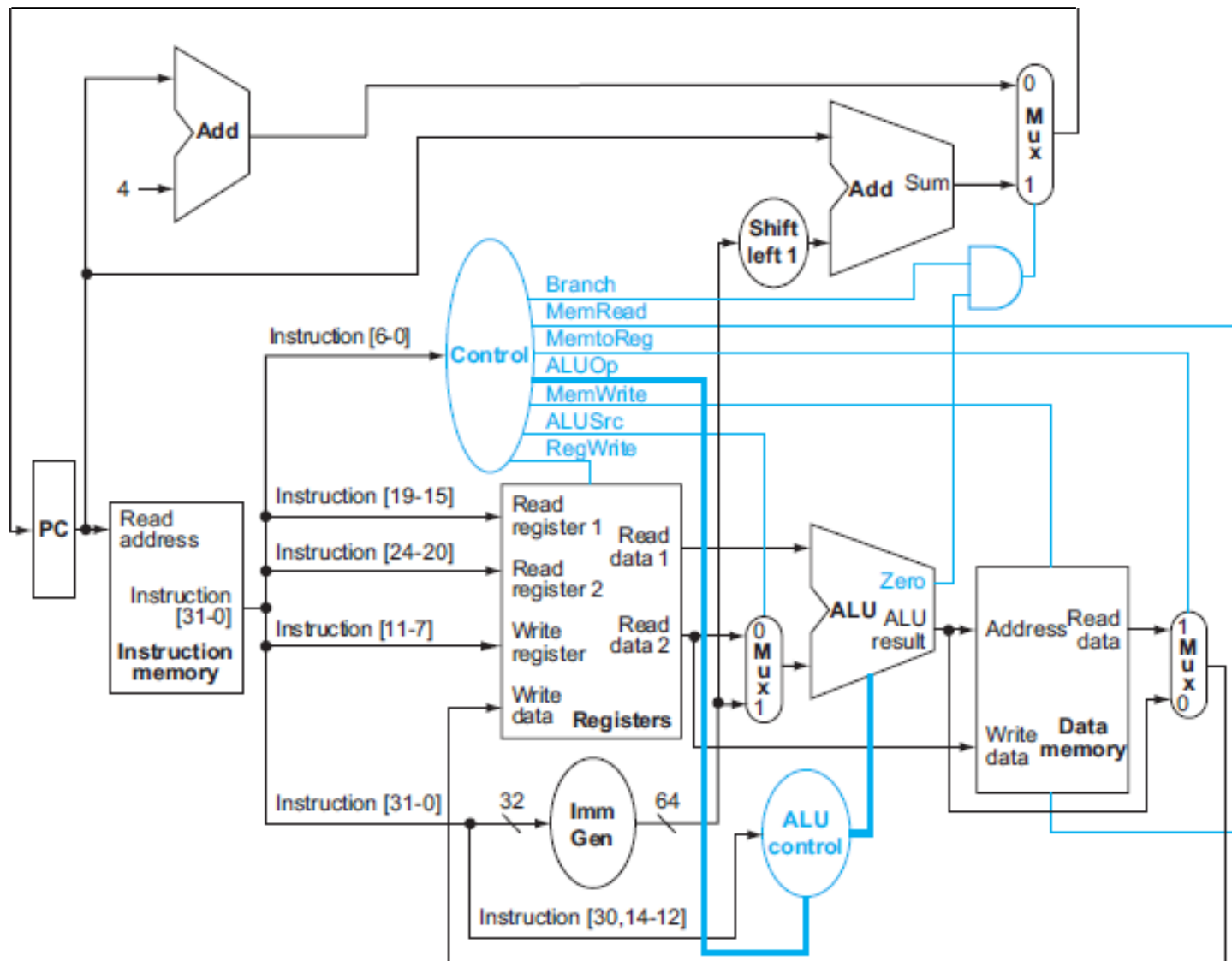
[Figure 4.17 from book, Copyright © 2018 Elsevier Inc. All rights reserved.]

SW Worksheet:



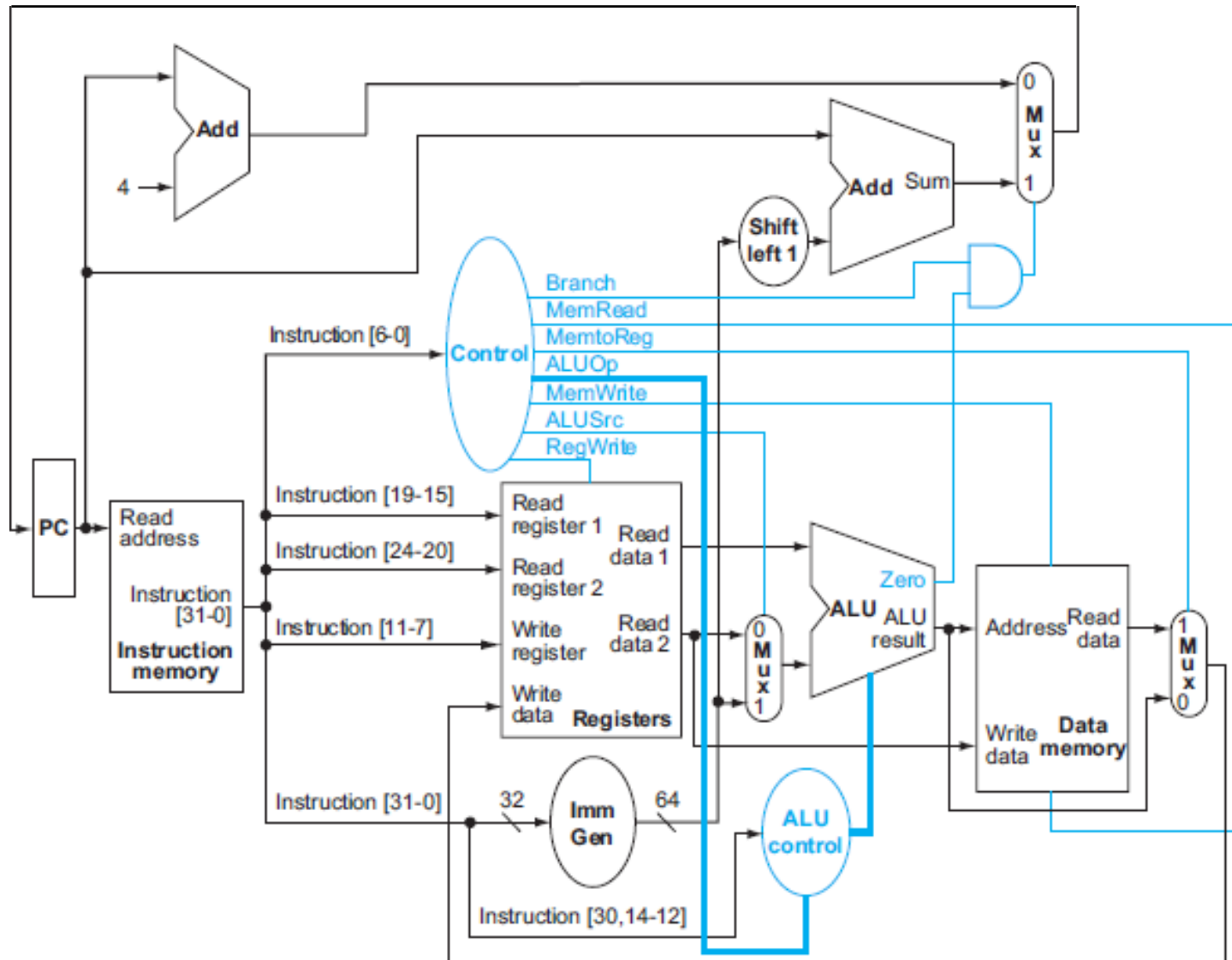
[Figure 4.17 from book, Copyright © 2018 Elsevier Inc. All rights reserved.]

Branch Taken Worksheet:



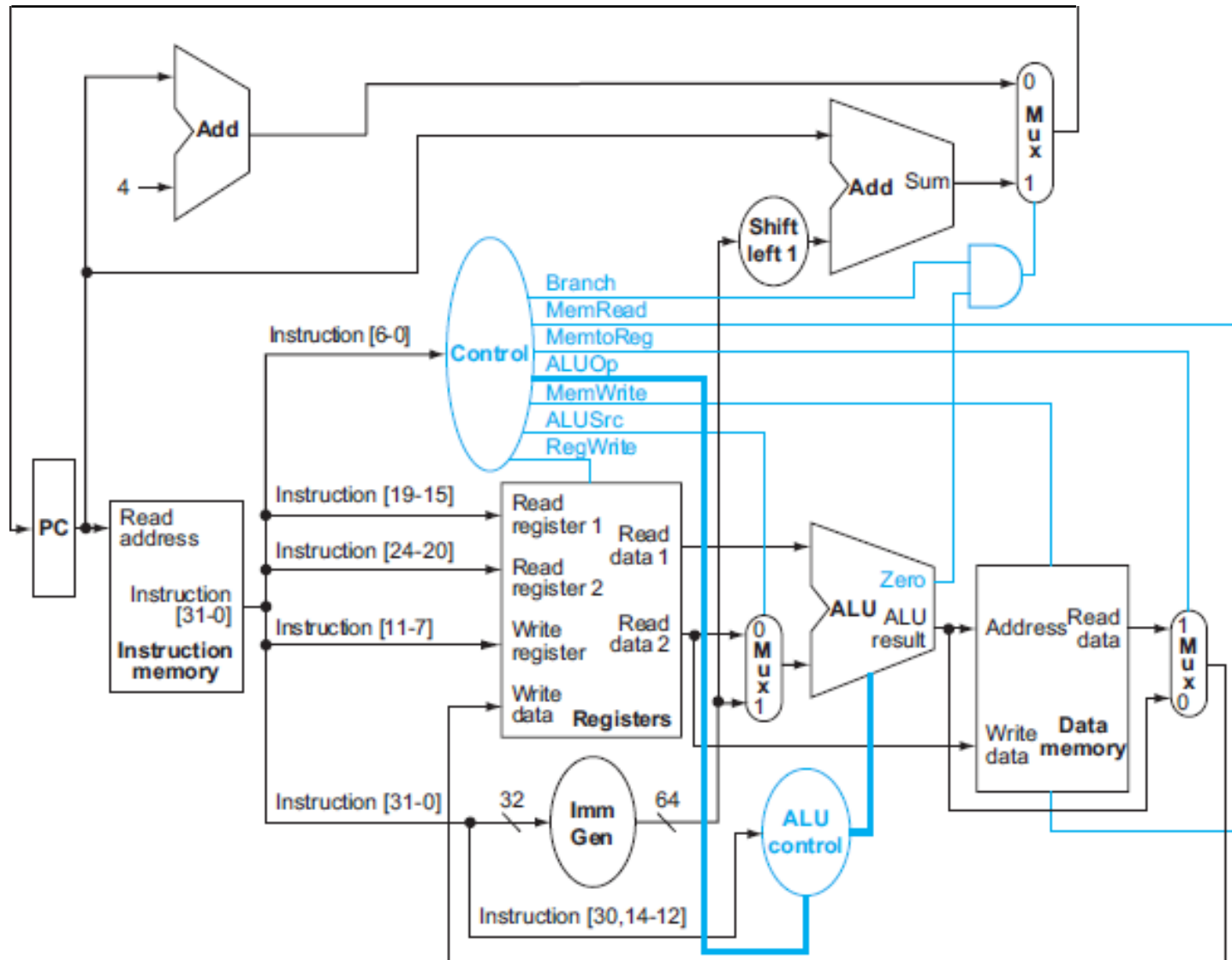
[Figure 4.17 from book, Copyright © 2018 Elsevier Inc. All rights reserved.]

Branch Not-Taken Worksheet:



[Figure 4.17 from book, Copyright © 2018 Elsevier Inc. All rights reserved.]

Jump (and Link?) ALU Worksheet:



[Figure 4.17 from book, Copyright © 2018 Elsevier Inc. All rights reserved.]

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	$(\text{opcode} \neq \text{IsRtype}) \ \&\& \ (\text{opcode} \neq \text{isBtype})$
RFWrite	GPR write disabled	GPR write enabled	$(\text{opcode} \neq \text{SW}) \ \&\& \ (\text{opcode} \neq \text{Bxx})$
MemtoRF	Steer ALU result to GPR write port	steer memory load to GPR write port	$\text{opcode} == \text{LW/H/B}$
PCtoRF	Steer above result to GPR write port	Steer PC+4 to GPR write port	$(\text{opcode} == \text{JAL}) \ \parallel \ (\text{opcode} == \text{JALR})$
MemRead	Memory read disabled	Memory read port return load value	$\text{opcode} == \text{LW/H/B}$
MemWrite	Memory write disabled	Memory write enabled	$\text{opcode} == \text{SW/H/B}$

Multi-Bit Control Signals

	Options	Equation
ALU Op	<ul style="list-style-type: none"> ADD, SUB, AND, OR, XOR, NOR, LT, and Shift bcond: EQ, NE, GE, LT 	case <code>opcode</code> RTypeALU: according to <code>funct3</code> , <code>funct7[5]</code> ITypeALU : according to <code>funct3</code> only (except shift) LW/SW/JALR : ADD Bxx : SUB and select bcond function — : ??
ImmMode	Itype, Stype, Btype, Utype, Jtype	<ul style="list-style-type: none"> 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 <code>opcode</code> JAL : PC + immediate JALR : GPR + immediate Bxx : taken?(PC + immediate):(PC + 4) — : PC+4
LoadExtend	W,H,HU,B,BU	case <code>func3</code>

Architecture vs Microarchitecture

Architecture

- Architectural Level



a clock has a hour hand and a minute hand,



a computer does?????....

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

Special Notices about Labs

- Lab 1 fully rolling
 - RISC-V simulator (C code)
 - single-cycle RISC-V (RTL Verilog)
- To get yourself rolling **(even if waitlisted)**
 - get a GitHub account
 - find lab partners
- Please observe
 - lab assignments **MUST** be done in groups of 2 or 3
 - entire group **MUST** be present during check-off
 - 10% per day penalty for late labs, capped at 50%
 - **all labs MUST be checked off to pass the course**