Lecture 5:
CPU : Single-Cycle Implementation

Jangwoo Kim (Seoul National University)
jangwoo@snu.ac.kr

Announcement

- HW #1
  - Are you doing OK?
- Lab #1
  - Are you doing OK?
- Q&A board
  - You can use "Korean"!
  - I will then make all key information available in English.

Review: MIPS ISA

- 3 simple formats
  - R-type, 3 register operands

0	rs	rt	rd	shamt	funct
6-bit	5-bit	5-bit	5-bit	5-bit	6-bit

R-type

  - I-type, 2 register operands and 16-bit immediate operand

opcode	rs	rt	immediate
6-bit	5-bit	5-bit	16-bit

I-type

  - J-type, 26-bit immediate operand

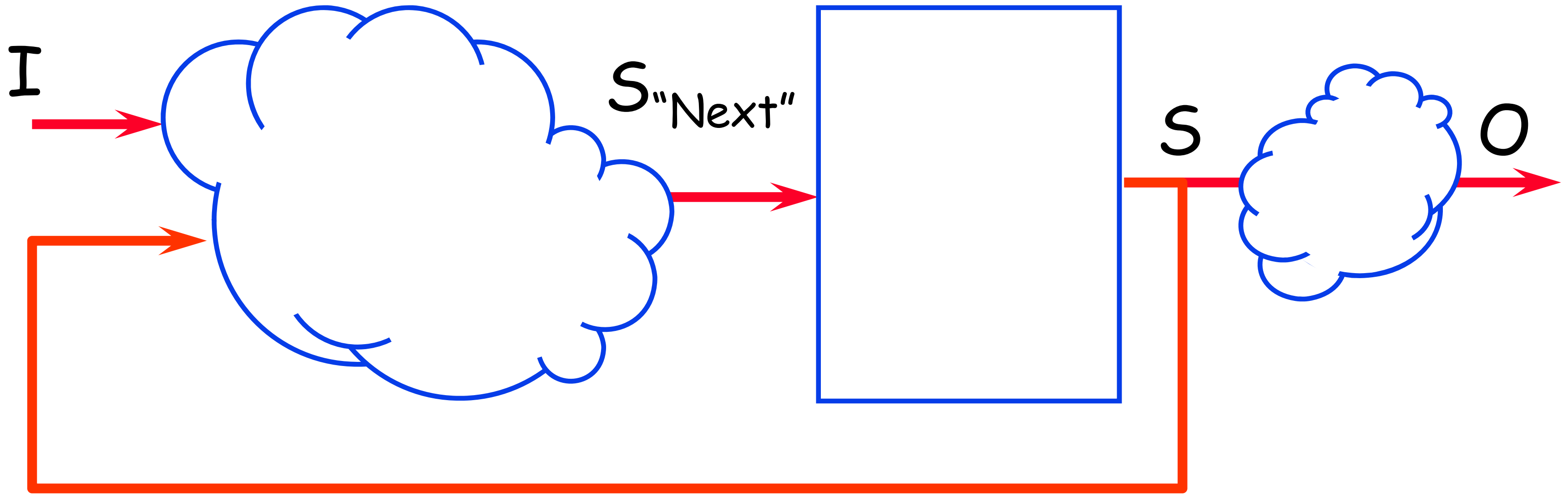
opcode	immediate
6-bit	26-bit

J-type

- Simple Decoding
  - 4 bytes per instruction, regardless of format
  - must be 4-byte aligned (2 lsb of PC must be 2b'00)

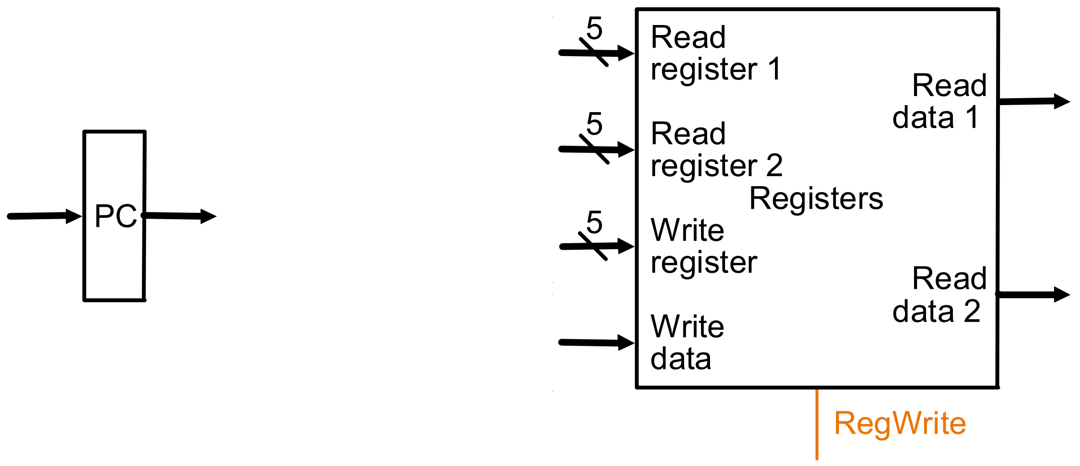
MIPS-IV ISA on the course website.

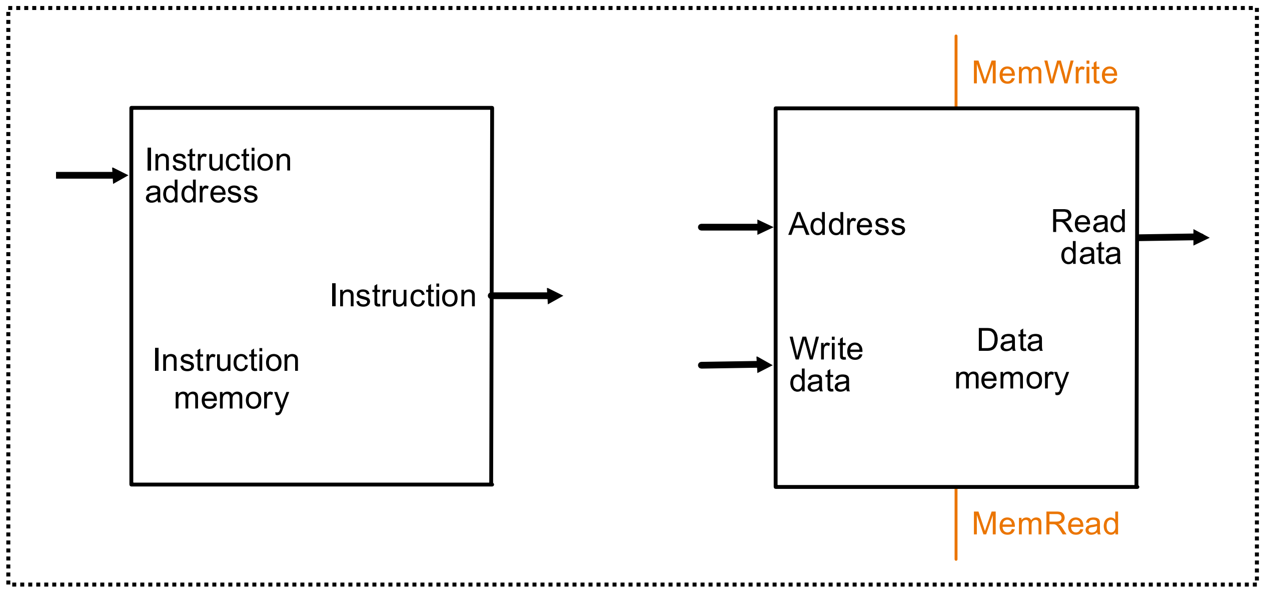
Instruction Processing FSM



- An ISA describes an abstract finite-state machine (FSM)
  - State = program visible state
  - Next-state logic = instruction execution
- Nice ISAs have atomic instruction semantics
  - One state transition per instruction in abstract FSM
- Implementation of FSMs can vary

Program Visible State



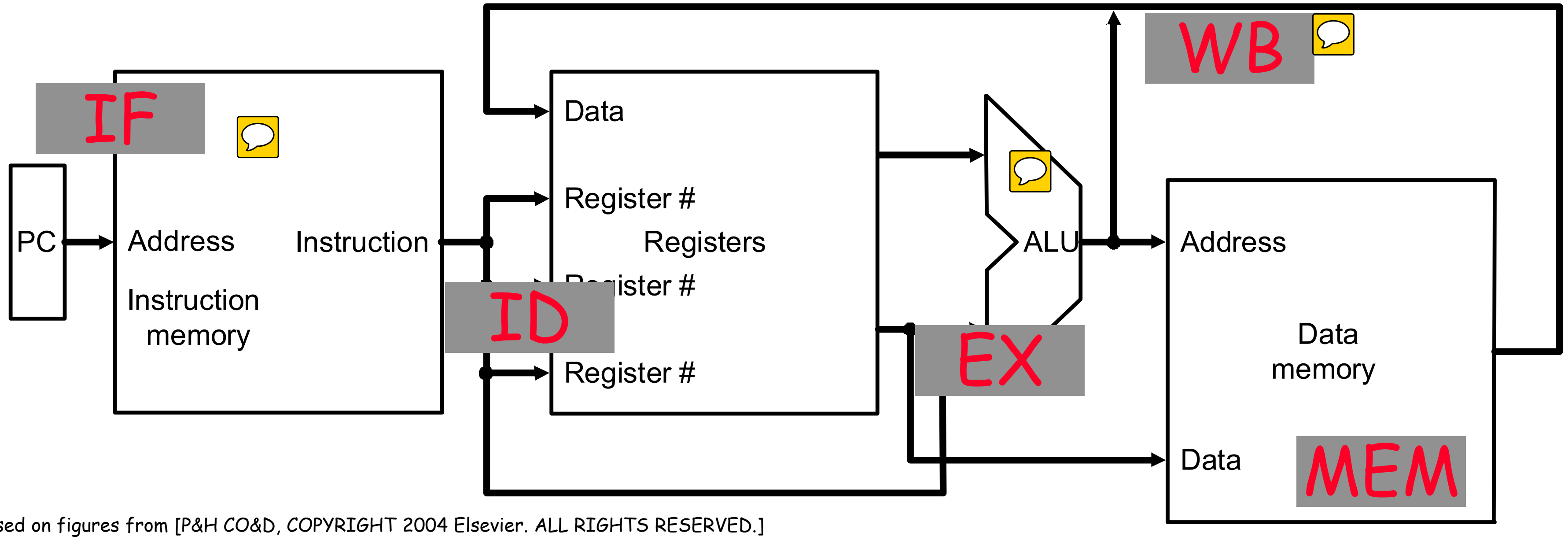


"Magic" Memory and Register File

- Combinational Read
  - The 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 is updated on the positive-edge clock transition when write enable is asserted

Instruction Processing

- 5 generic steps
  - IF: Instruction fetch
  - ID: Instruction decode and operand fetch
  - EX: ALU/execute
  - MEM: Memory access (only for load & store instructions)
  - WB: Write-back



Single-Cycle Datapath for Arithmetic and Logical Instructions

R-Type ALU Instructions

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

  ADD rd_{reg} rs_{reg} rt_{reg}

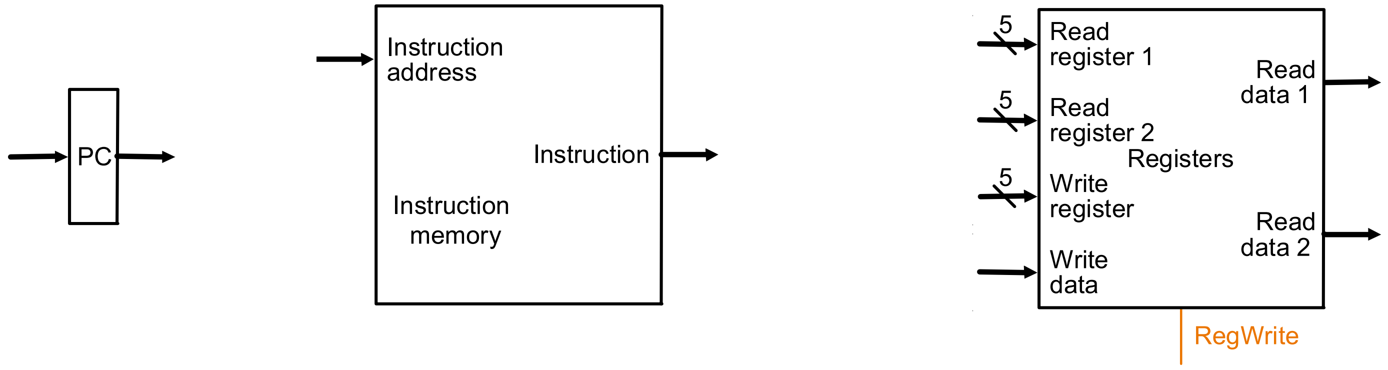
- Machine encoding

0	rs	rt	rd	0	ADD	R-type
6-bit	5-bit	5-bit	5-bit	5-bit	6-bit	

- FSM transition semantics

```
if MEM[PC] == ADD rd rs rt
        GPR[rd] ← GPR[rs] + GPR[rt]
        PC ← PC + 4
```

ADD rd rs rt



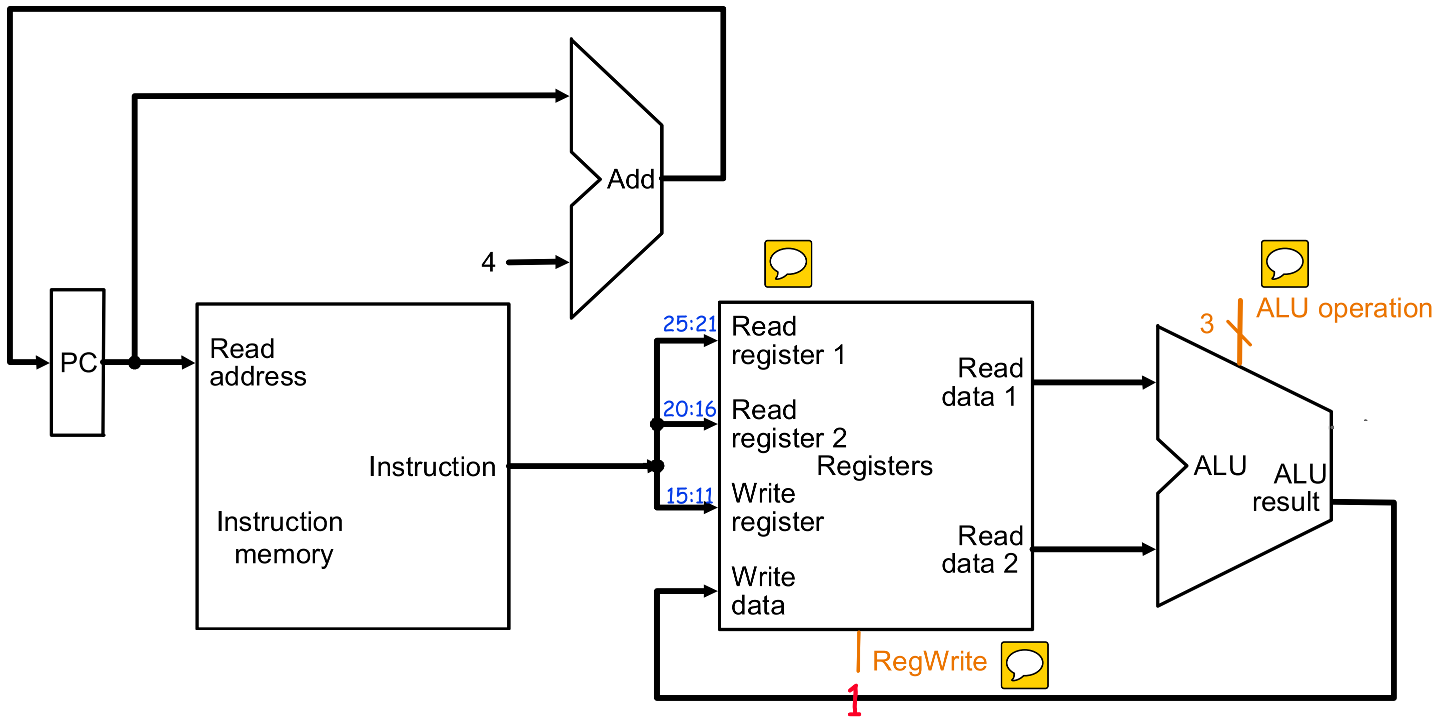
```
if MEM[PC] == ADD rd rs rt
    GPR[rd] ← GPR[rs] + GPR[rt]
        PC ← PC + 4
```

→

IF	ID	EX	MEM	WB

Combinational state update logic

ALU Datapath



I-Type ALU Instructions

- Assembly (e.g., register-immediate signed additions)

  ADDI rt_{reg} rs_{reg} $immediate_{16}$

- Machine encoding

ADDI	rs	rt	immediate
6-bit	5-bit	5-bit	16-bit

I-type

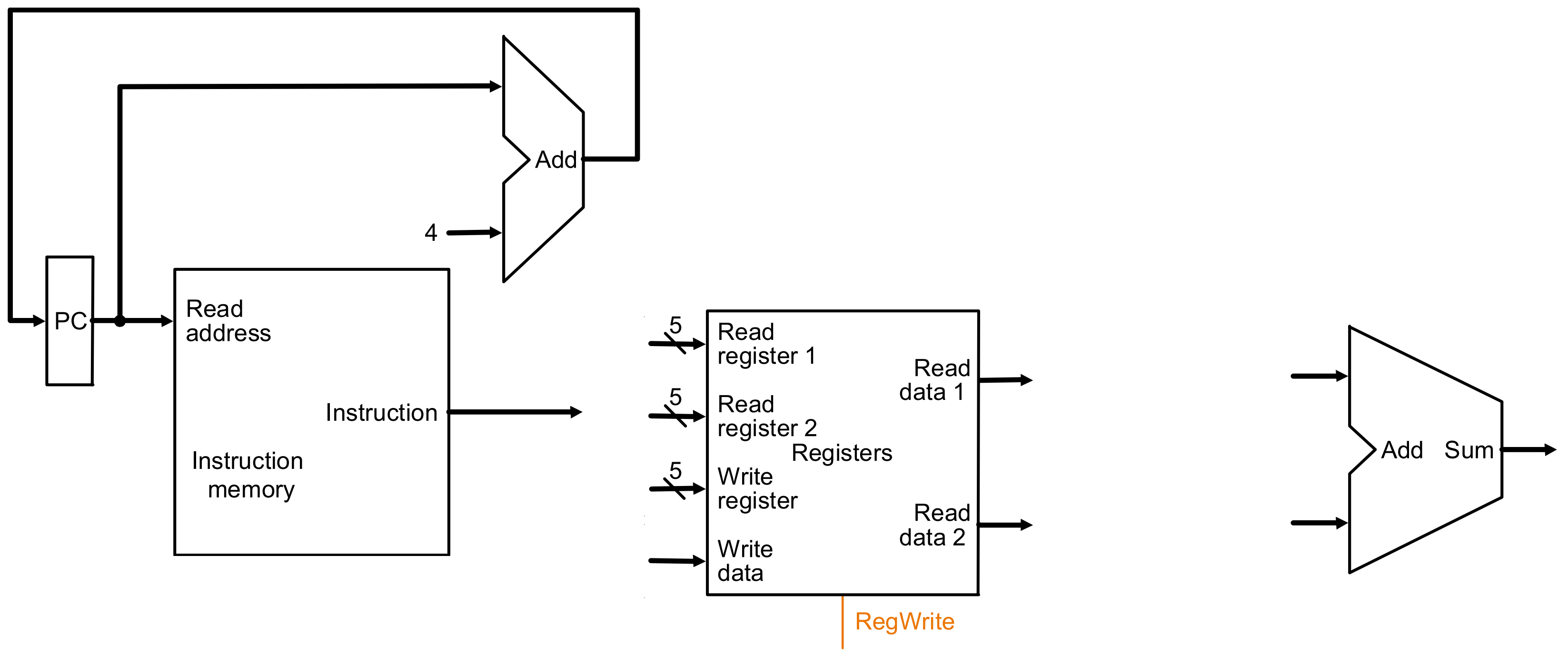
- FSM transition semantics

  if MEM[PC] == ADDI rt rs immediate

  GPR[rt] $\leftarrow$ GPR[rs] + sign-extend (immediate)

  PC $\leftarrow$ PC + 4

ADDI rt rs immediate$_{16}$

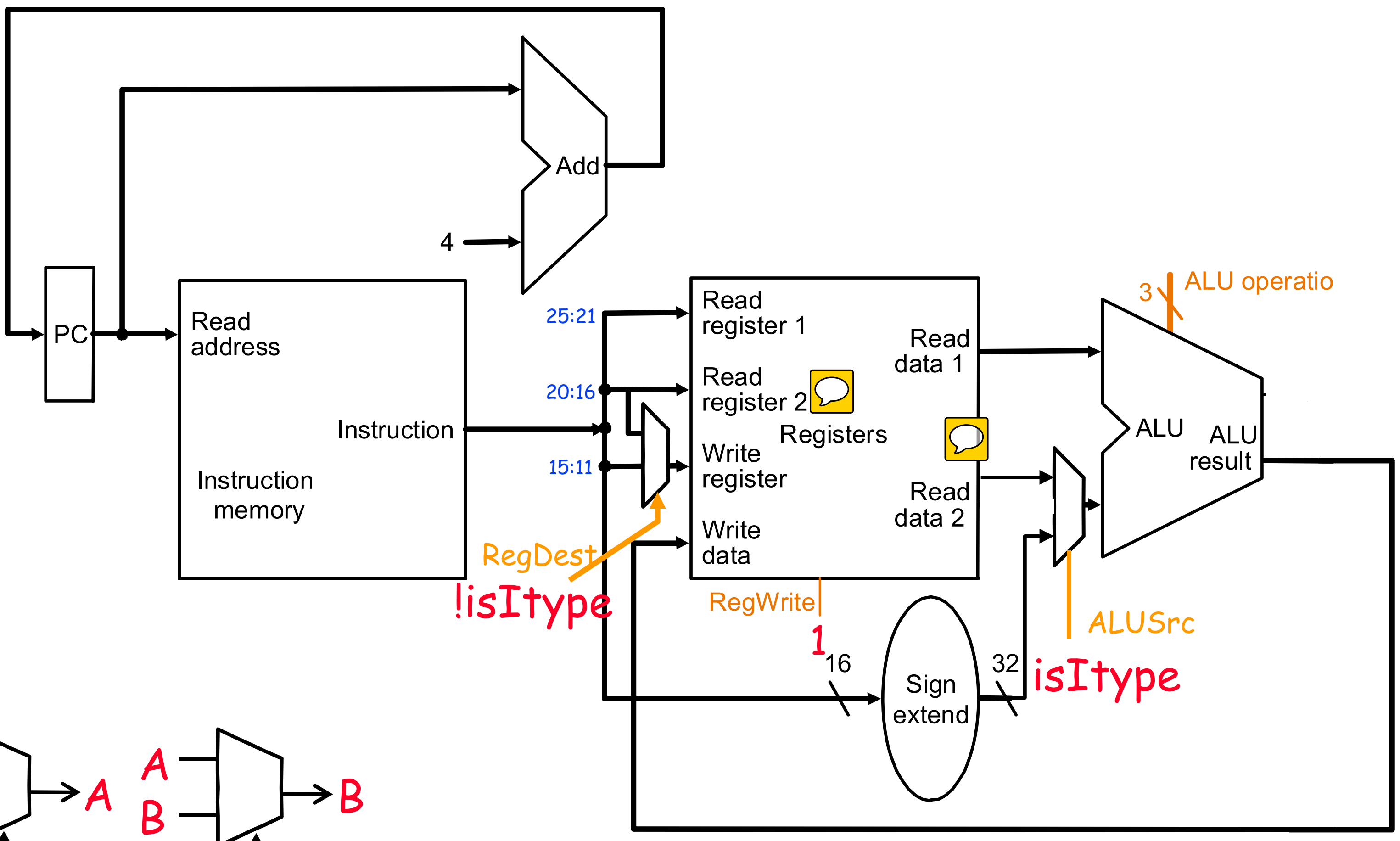


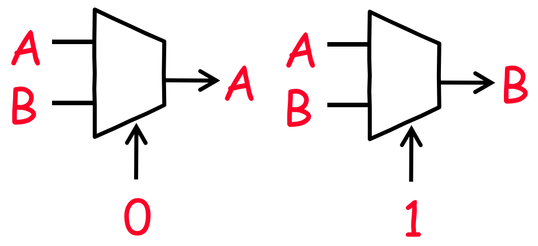
if MEM[PC] == ADDI rt rs immediate
GPR[rt] ← GPR[rs] + sign-extend (immediate)
PC ← PC + 4

IF	ID	EX	MEM	WB

Combinational
state update logic

Datapath for R and I-Type ALU Instructions





Single-Cycle Datapath for Data Movement Instructions

Load Instructions

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

  LW rt_{reg} offset_{16} (base_{reg})

- Machine encoding

LW	base	rt	offset	I-type
6-bit	5-bit	5-bit	16-bit	

- FSM transition semantics

  if MEM[PC]==LW rt offset_{16} (base)

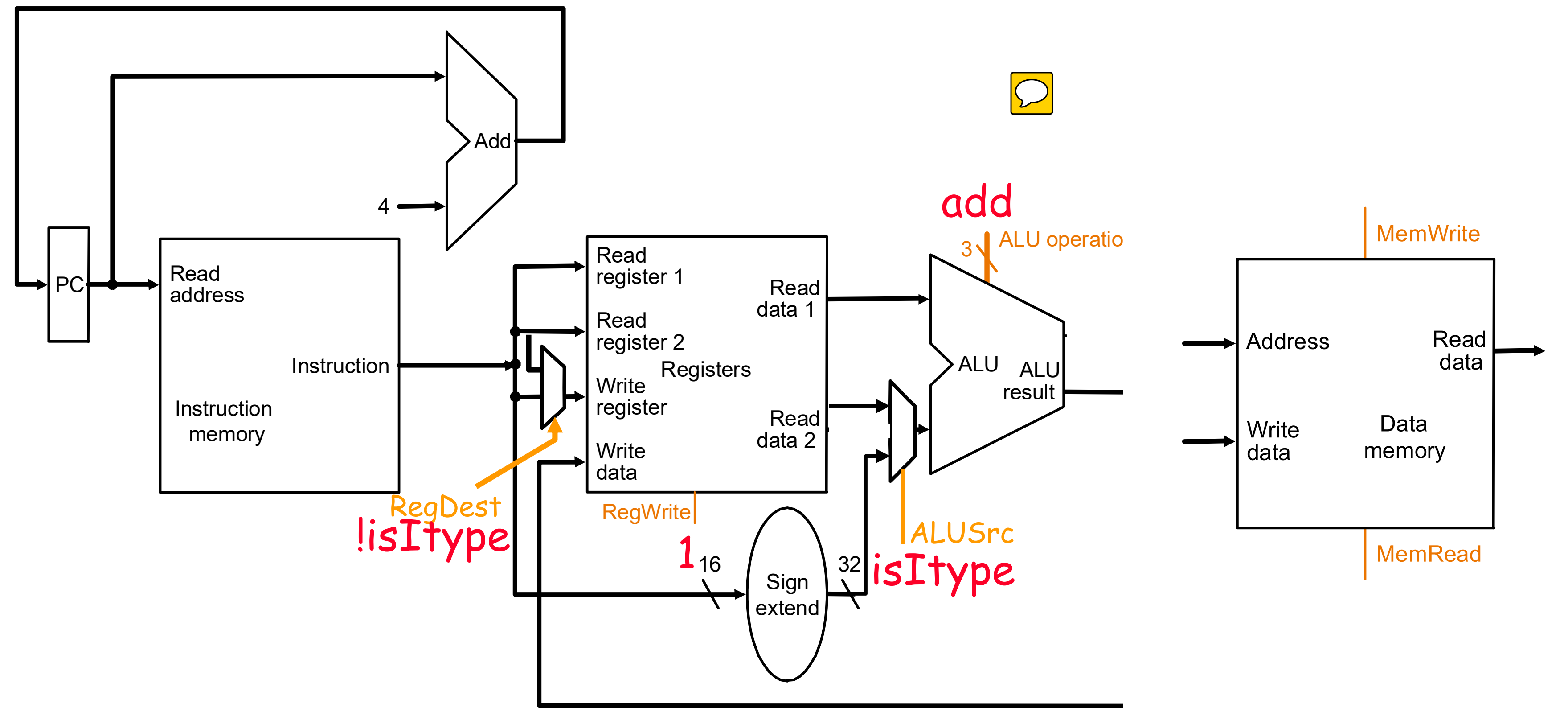
  EA = sign-extend(offset) + GPR[base]

  GPR[rt] $\leftarrow$ MEM[translate(EA)]

  PC $\leftarrow$ PC + 4

Let's not worry about delay slots here

LW Datapath



if MEM[PC]==LW rt offset$_{16}$ (base)
EA = sign-extend(offset) + GPR[base]
GPR[rt] $\leftarrow$ MEM[translate(EA)]
PC $\leftarrow$ PC + 4

IF	ID	EX	MEM	WB

Combinational
state update logic

Store Instructions

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

  SW rt_{reg} offset_{16} (base_{reg})

- Machine encoding

SW	base	rt	offset
6-bit	5-bit	5-bit	16-bit

I-type

- FSM transition semantics

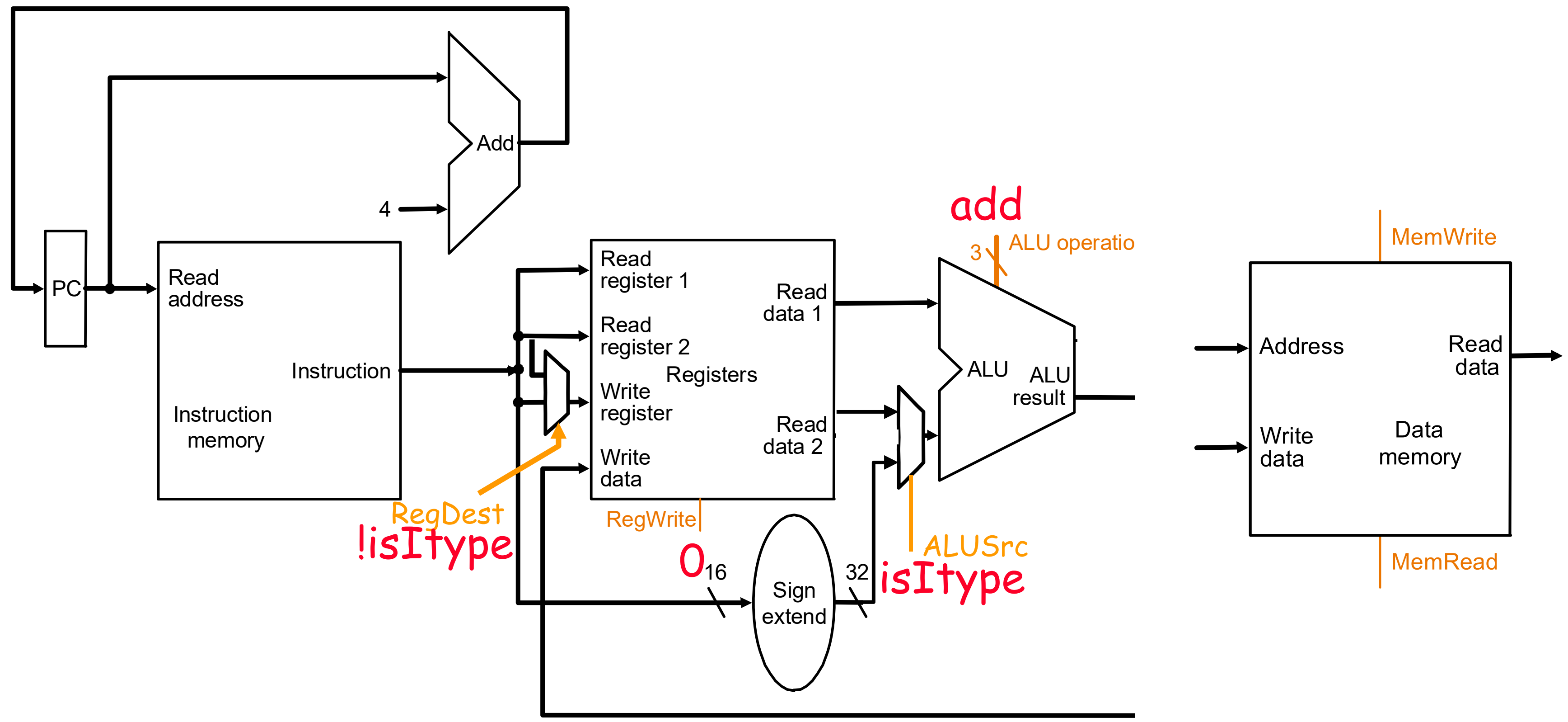
  if MEM[PC]==SW rt offset_{16} (base)

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

  MEM[translate(EA)] $\leftarrow$ GPR[rt]

  PC $\leftarrow$ PC + 4

SW Datapath

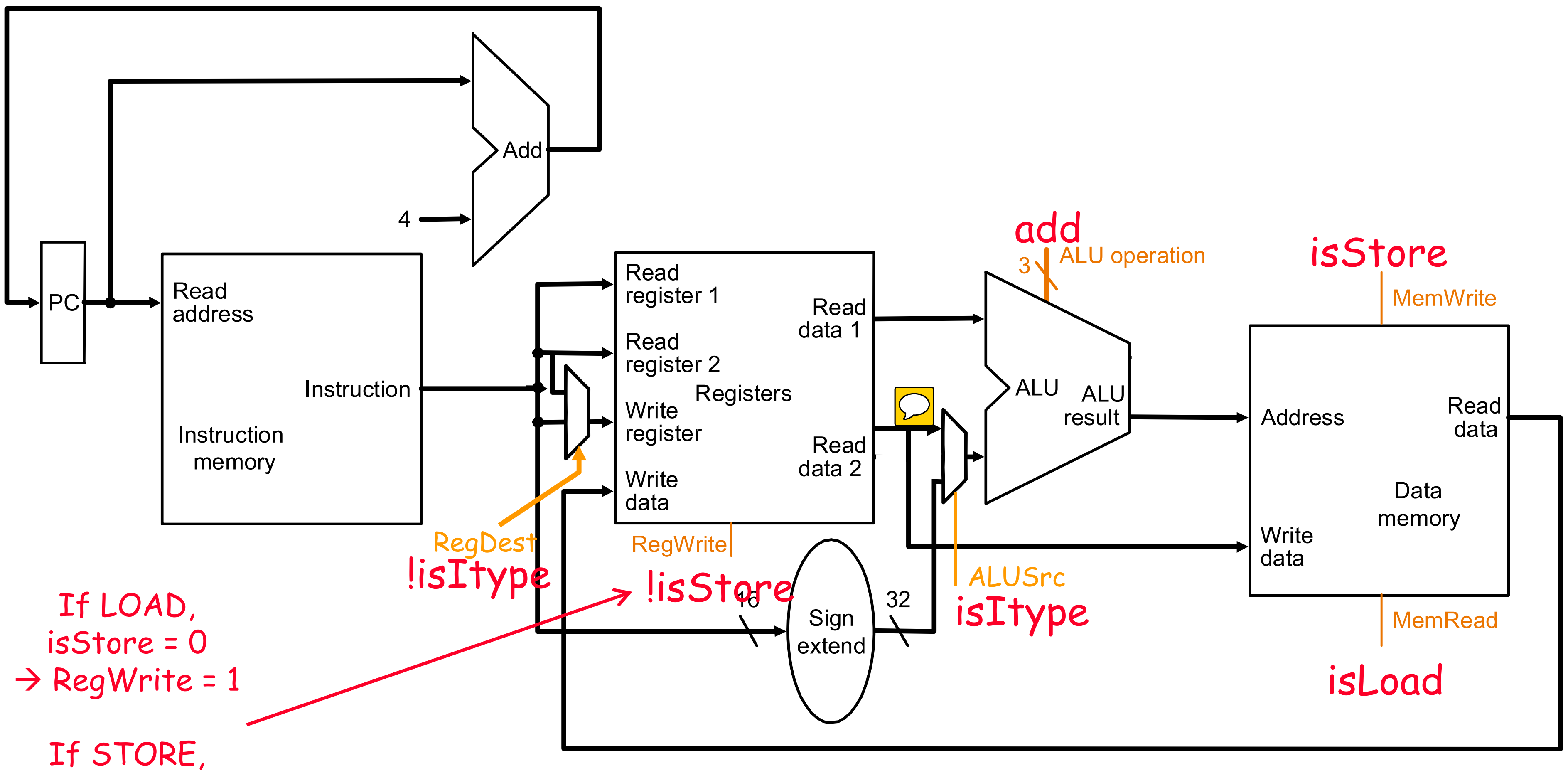


if MEM[PC]==SW rt offset$_{16}$ (base)
EA = sign-extend(offset) + GPR[base]
MEM[translate(EA)] $\leftarrow$ GPR[rt]
PC $\leftarrow$ PC + 4

IF	ID	EX	MEM	WB

Combinational
state update logic

Load-Store Datapath

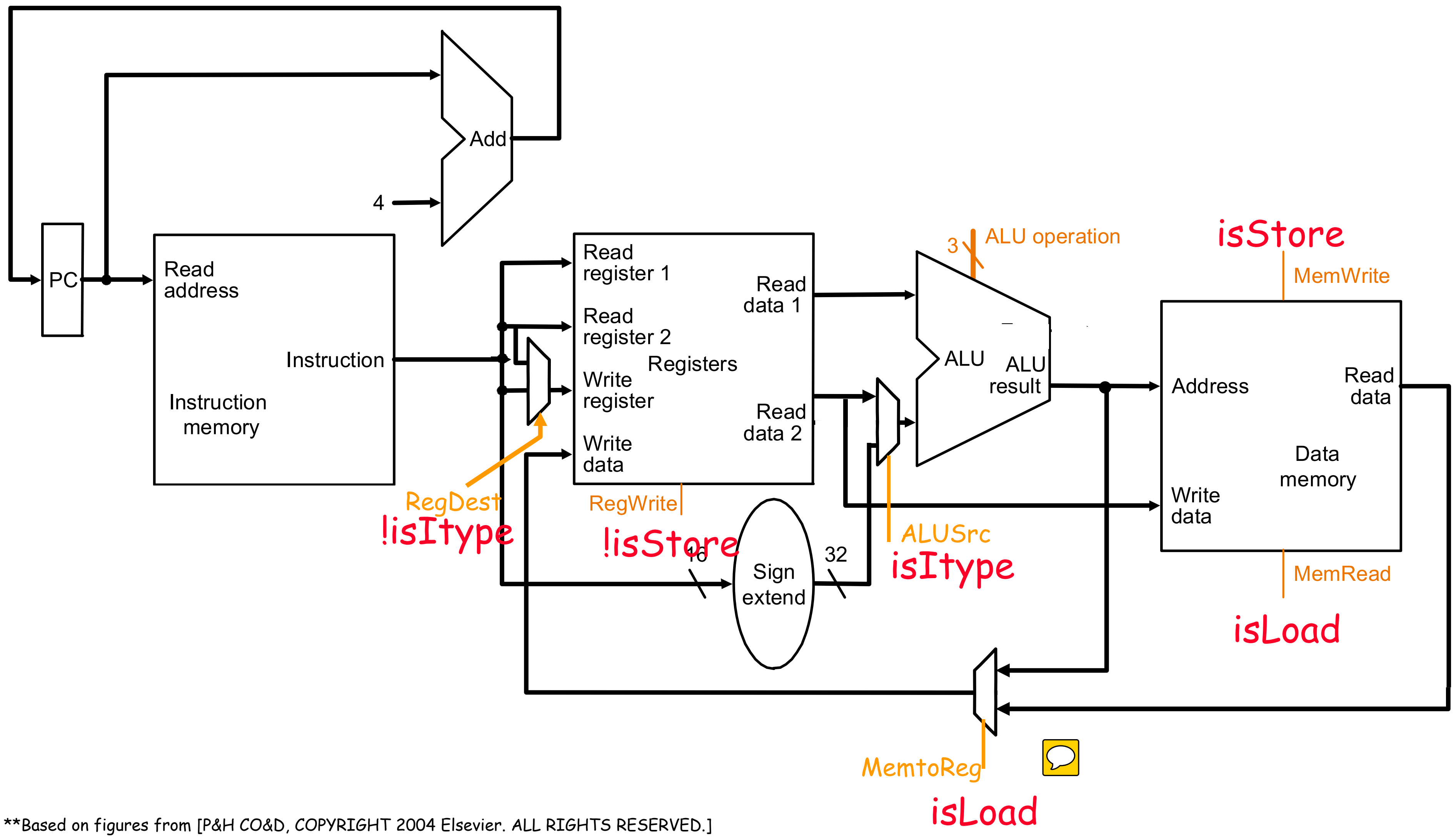


If LOAD,
isStore = 0
→ RegWrite = 1

If STORE,
isStore = 1
→ RegWrite = 0

How to uphold the delayed load semantics?

Datapath for Non-Control Flow Instructions



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

Single-Cycle Datapath for Control Flow Instructions

Unconditional Jump Instructions

- Assembly

  J immediate_{26}

- Machine encoding

J	immediate	J-type
6-bit	26-bit	

- FSM transition semantics

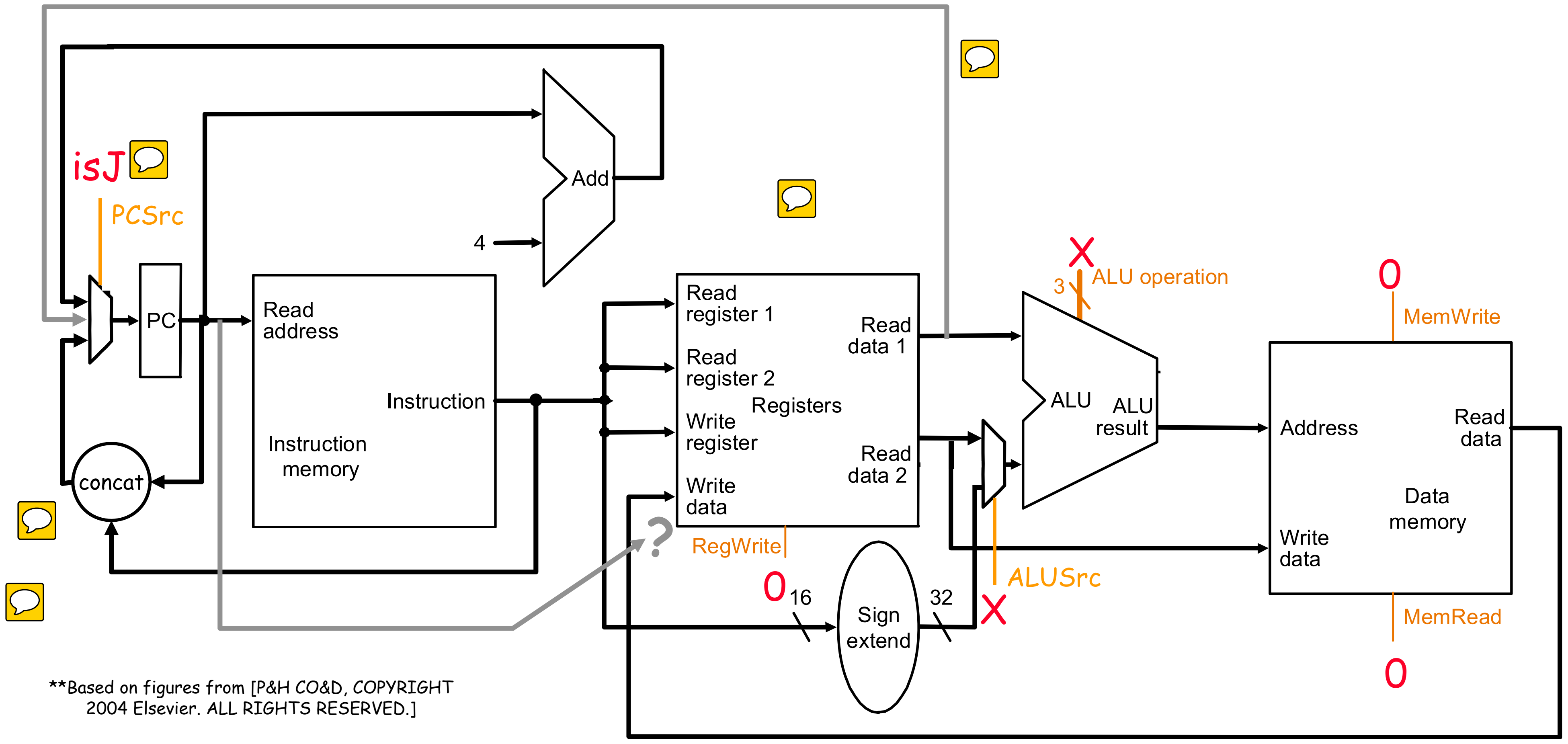
  if MEM[PC]==J immediate_{26}

  target = { PC[31:28], immediate_{26}, 2'b00 }

  PC $\leftarrow$ target

Let's not worry about delay slots here

Unconditional Jump Datapath



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

if MEM[PC]==J immediate26
PC = { PC[31:28], immediate26, 2'b00 }

What about JR, JAL, JALR?

Conditional Branch Instructions

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

  BEQ rs_{reg} rt_{reg} $immediate_{16}$

- Machine encoding

BEQ	rs	rt	immediate
6-bit	5-bit	5-bit	16-bit

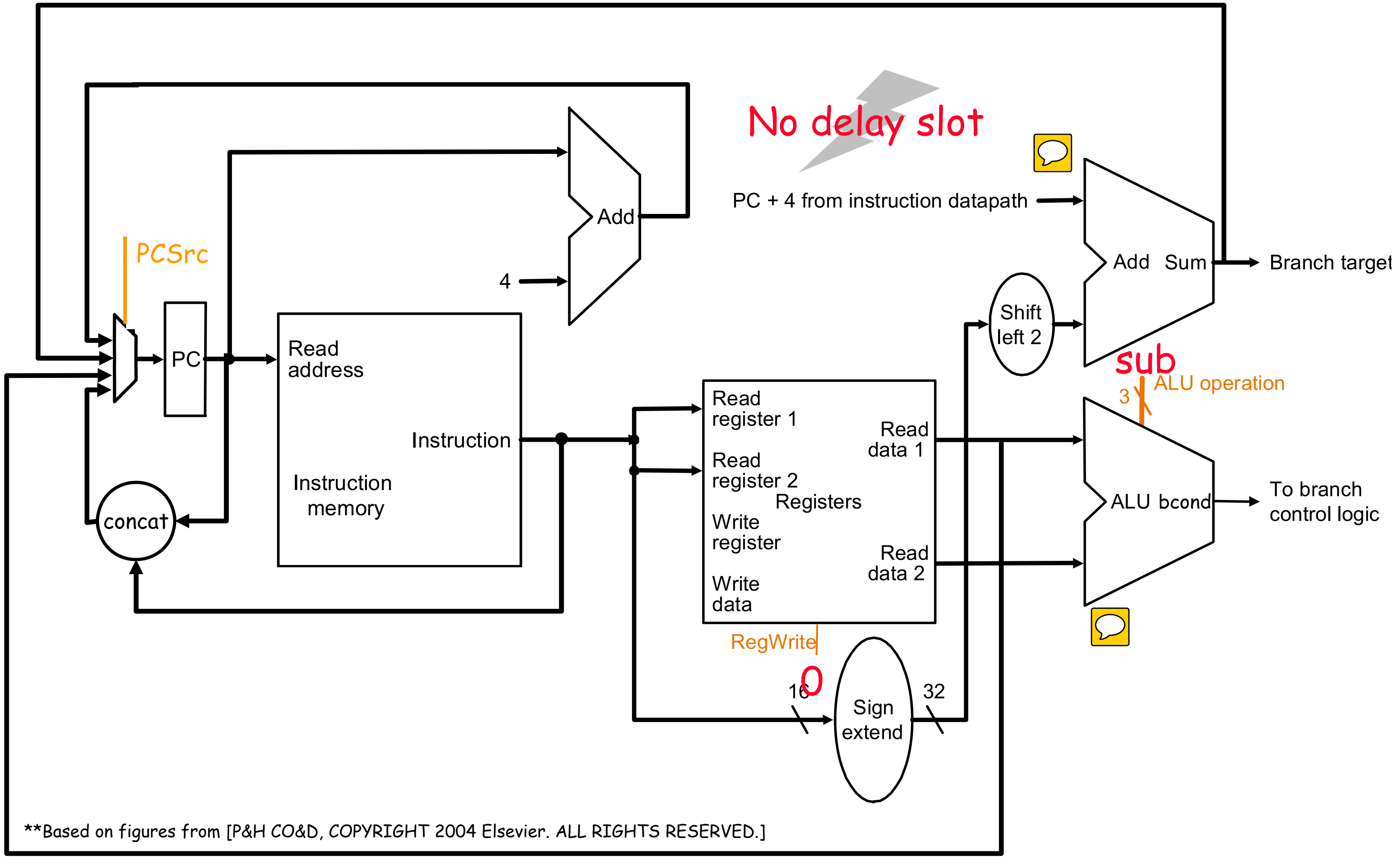
I-type

- FSM transition semantics

```
if MEM[PC]==BEQ rs rt immediate_16
    target = PC + sign-extend(immediate) x 4 +4
    if GPR[rs]==GPR[rt] then    PC ← target
                        else    PC ← PC + 4
```

Let's not worry about delay slots here

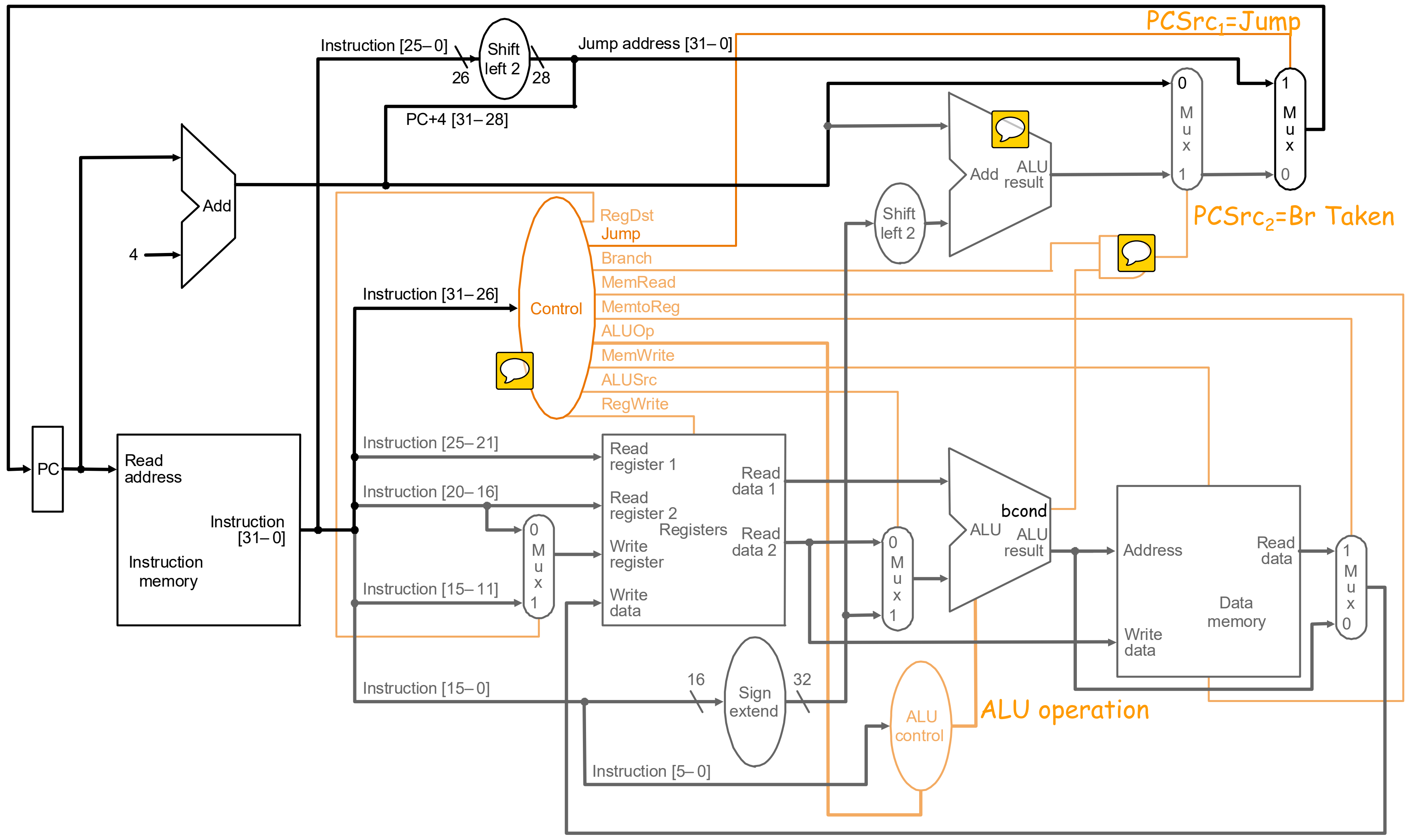
Conditional Branch Datapath



How to uphold the delayed branch semantics?

Control

Putting It All together

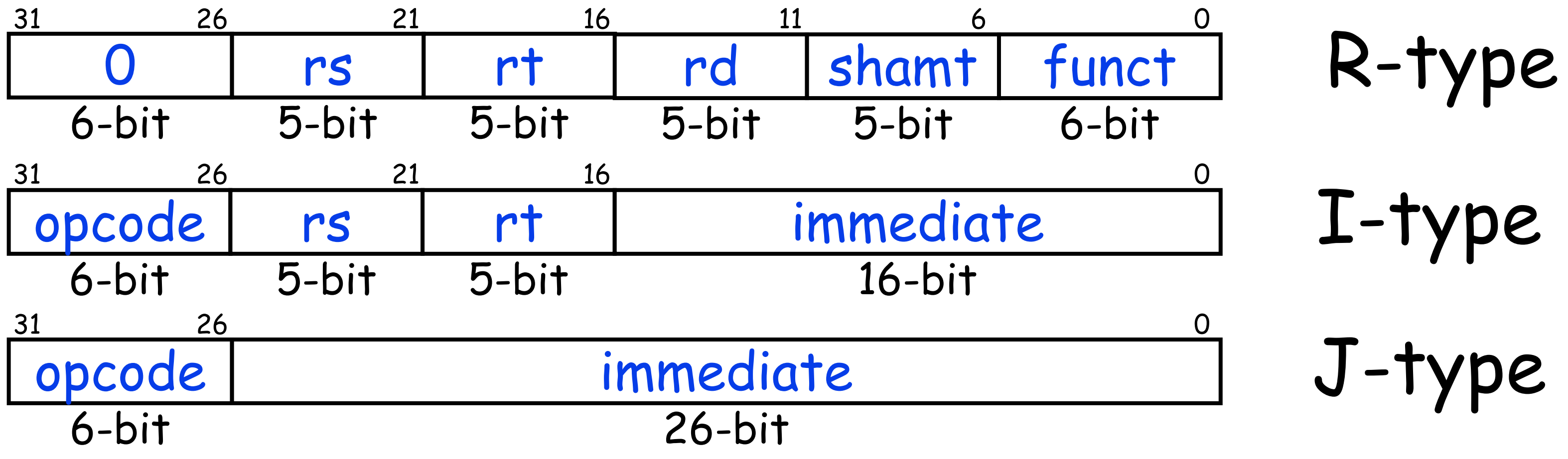


**Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

JAL, JR, JALR omitted

Single-Cycle Control

- As combinational function of Inst=MEM[PC]



- Consider
  - All R-type and I-type ALU instructions
  - LW and SW
  - BEQ, BNE, BLEZ, BGTZ
  - J, JR, JAL, JALR

Single-Bit Control Signals

	When De-asserted	When asserted	Equation
RegDest	GPR write select according to rt, i.e., inst[20:16]	GPR write select according to rd, i.e., inst[15:11]	opcode==0
ALUSrc	2nd ALU input from 2nd GPR read port	2nd ALU input from sign-extended 16-bit immediate	(opcode!=0) && (opcode!=BEQ) && (opcode!=BNE)
MemtoReg	Steer ALU result to GPR write port	steer memory load to GPR wr. port	opcode==LW
RegWrite	GPR write disabled	GPR write enabled	(opcode!=SW) && (opcode!=Bxx) && (opcode!=J) && (opcode!=JR))

JAL and JALR require additional RegDest and MemtoReg options

Single-Bit Control Signals

	When De-asserted	When asserted	Equation
MemRead	Memory read disabled	Memory read port return load value	opcode==LW
MemWrite	Memory write disabled	Memory write enabled	opcode==SW
$PCSrc_1$	According to $PCSrc_2$	next PC is based on 26-bit immediate jump target	(opcode==J) || (opcode==JAL)
$PCSrc_2$	next PC = PC + 4	next PC is based on 16-bit immediate branch target	(opcode==Bxx) && "bcond is satisfied"

JR and JALR require additional PCSrc options

ALU Control Table Lookup

- Case opcode

  '0' ⇒ select operation according to funct

  'ALUi' ⇒ selection operation according to opcode

  'LW' ⇒ select addition

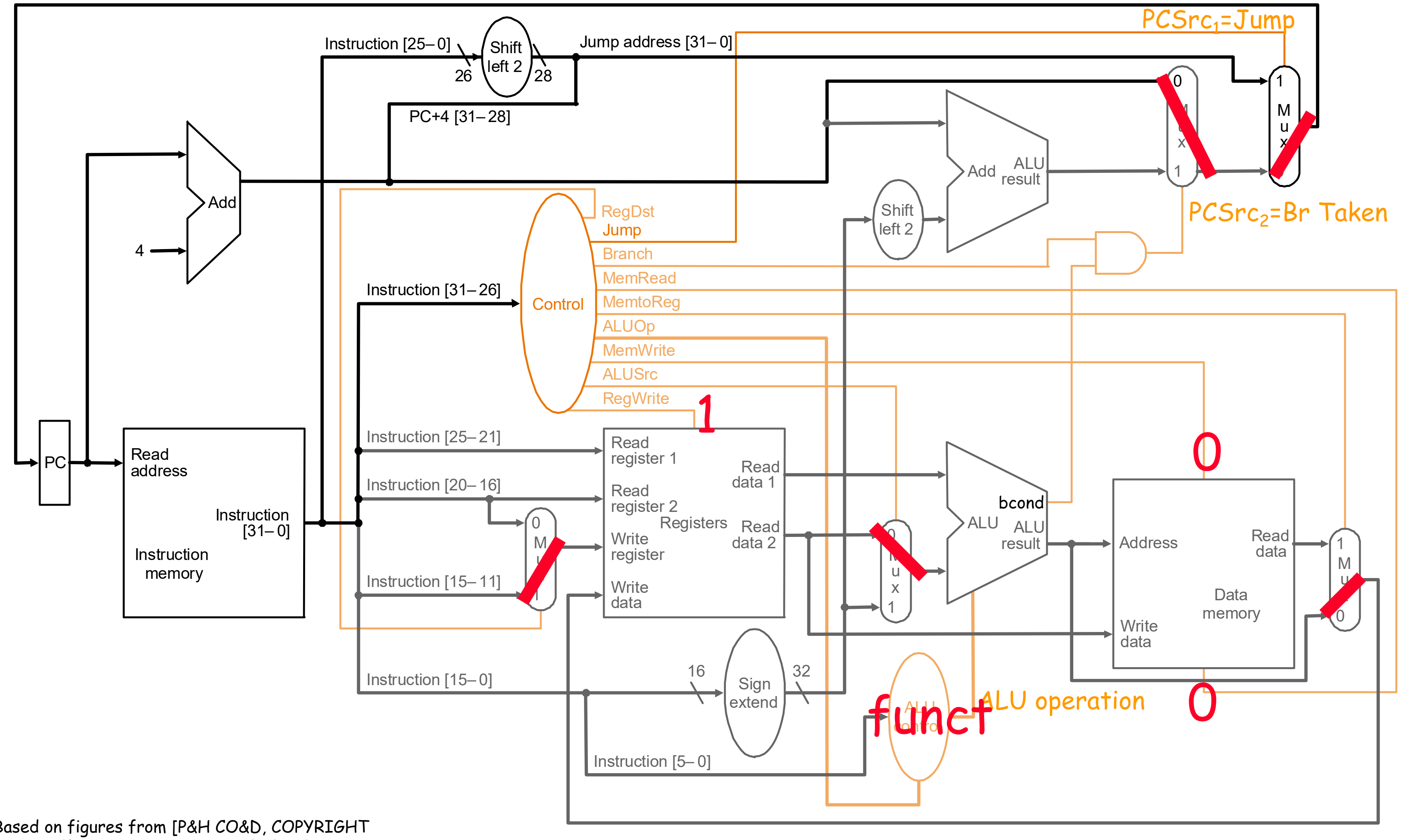
  'SW' ⇒ select addition

  'Bxx' ⇒ select bcond generation function

  __ ⇒ don't care

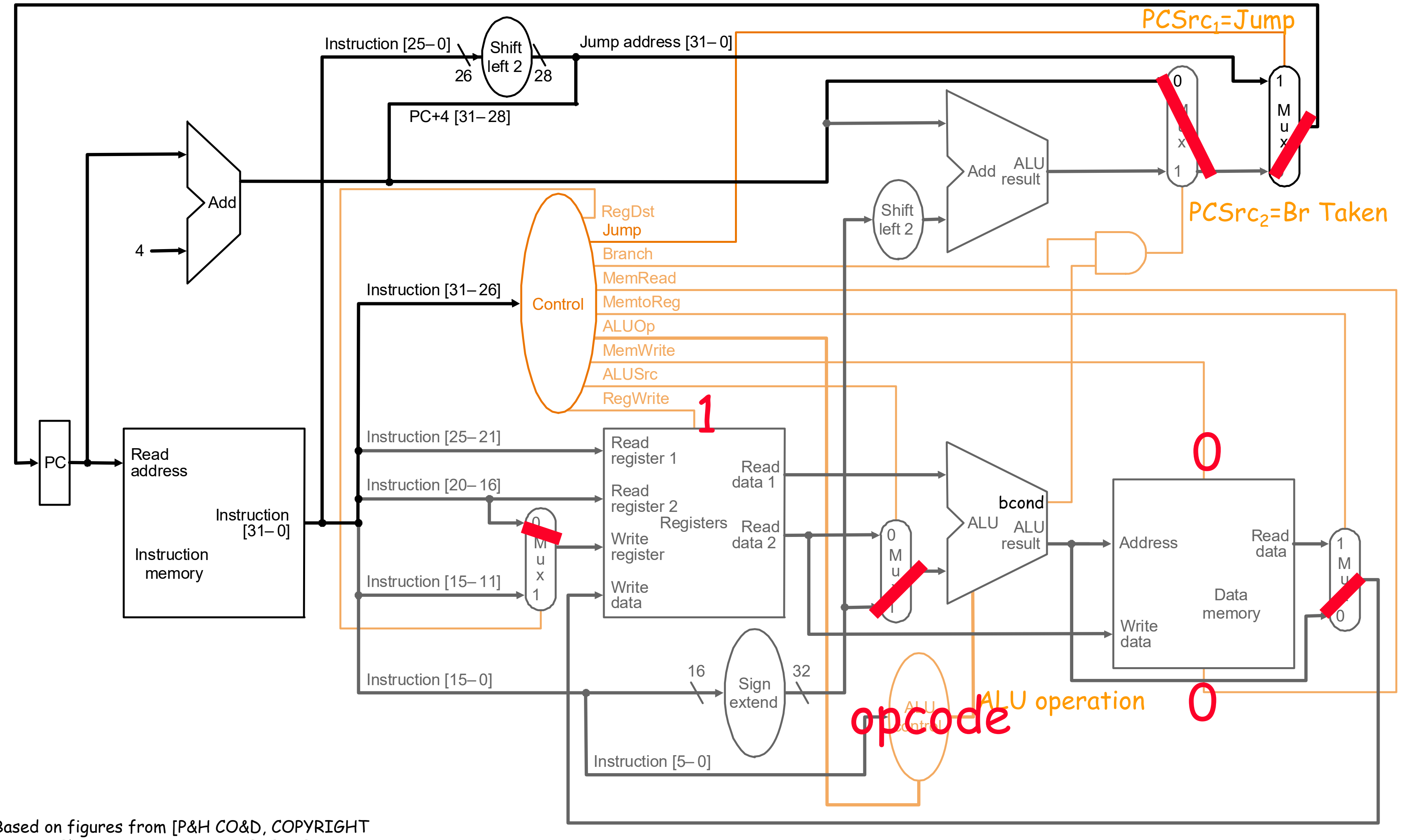
- Example ALU operations
  - ADD, SUB, AND, OR, XOR, NOR, etc.
  - bcond on equal, not equal, LE zero, GT zero, etc.

R-Type ALU



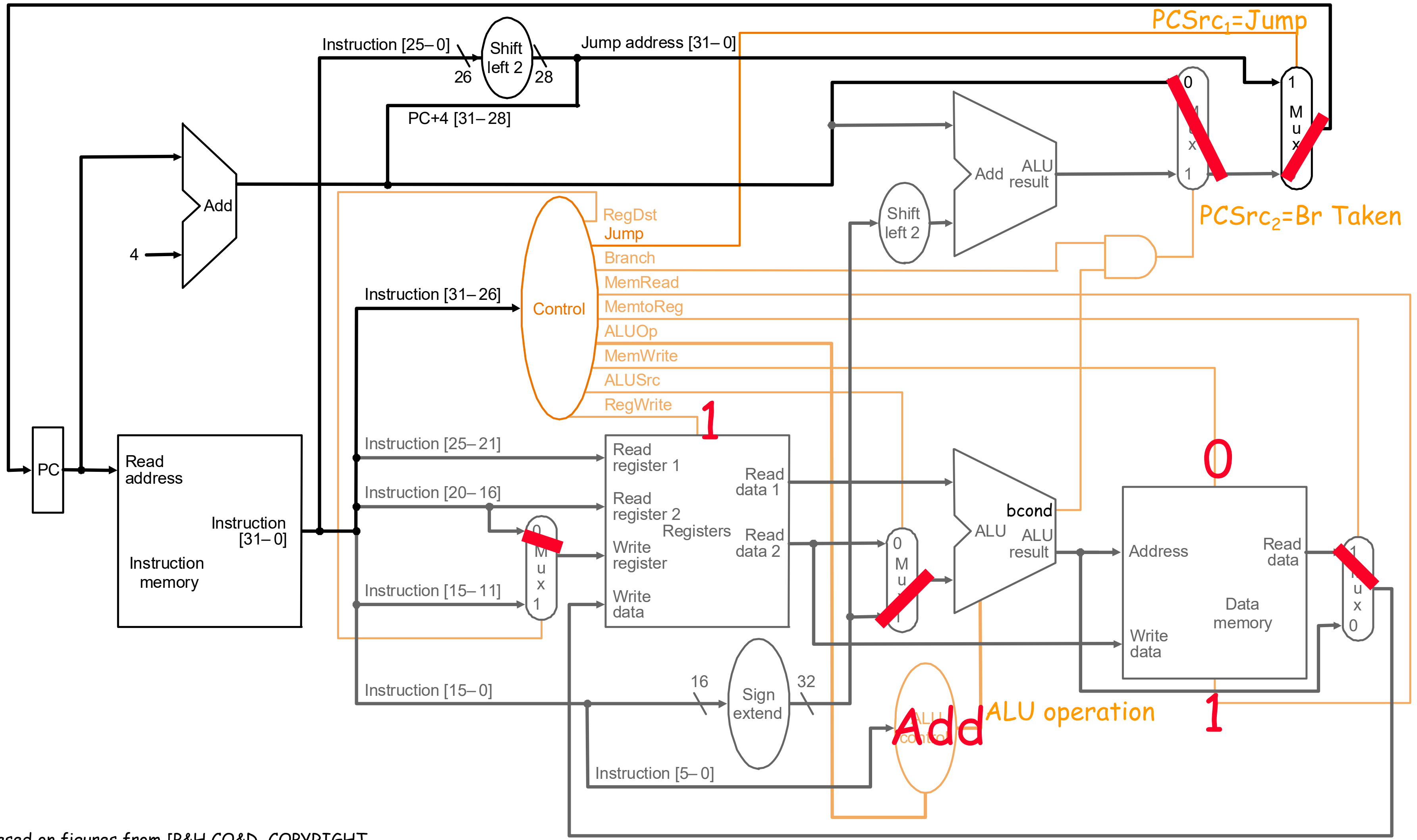
**Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

I-Type ALU



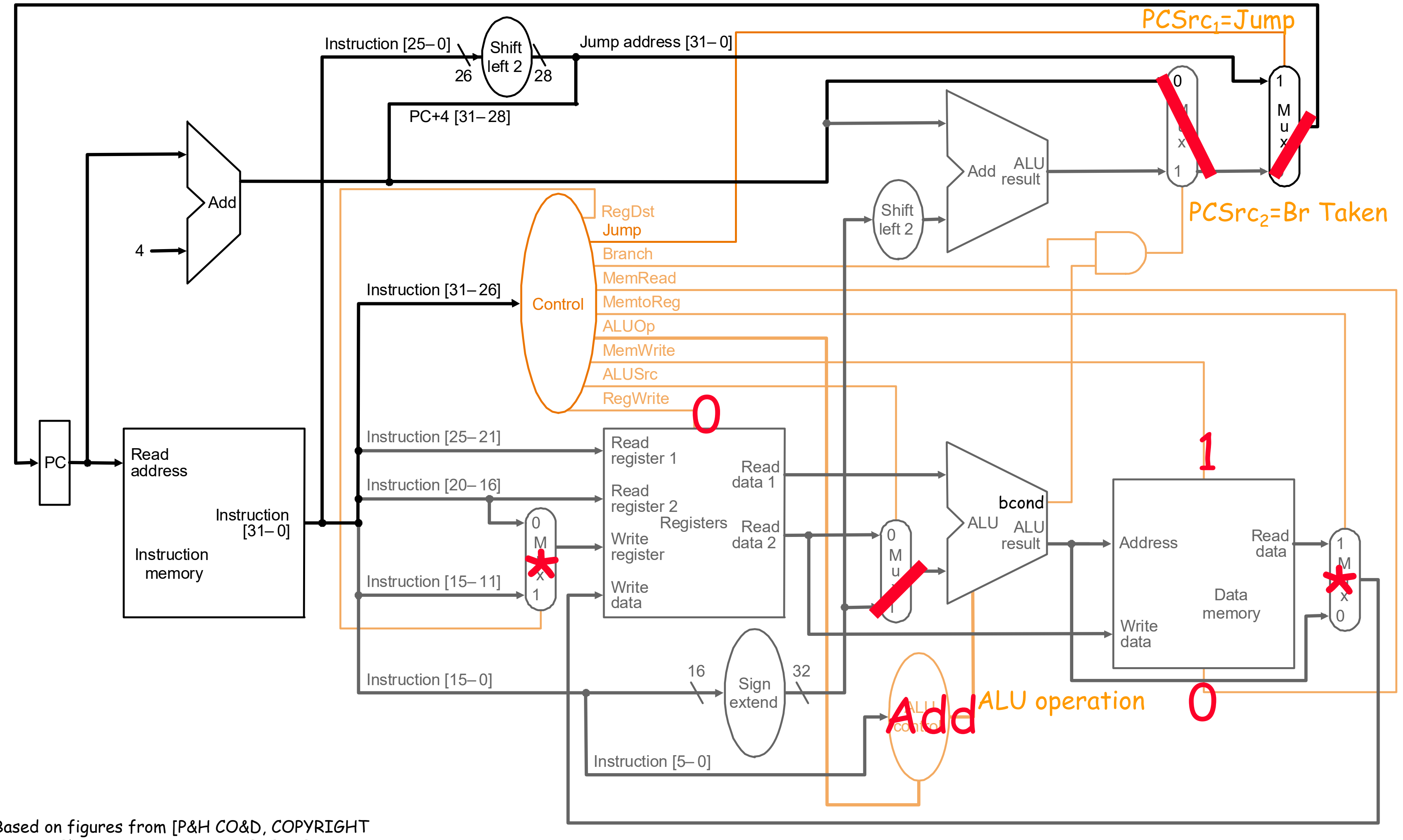
**Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

LW



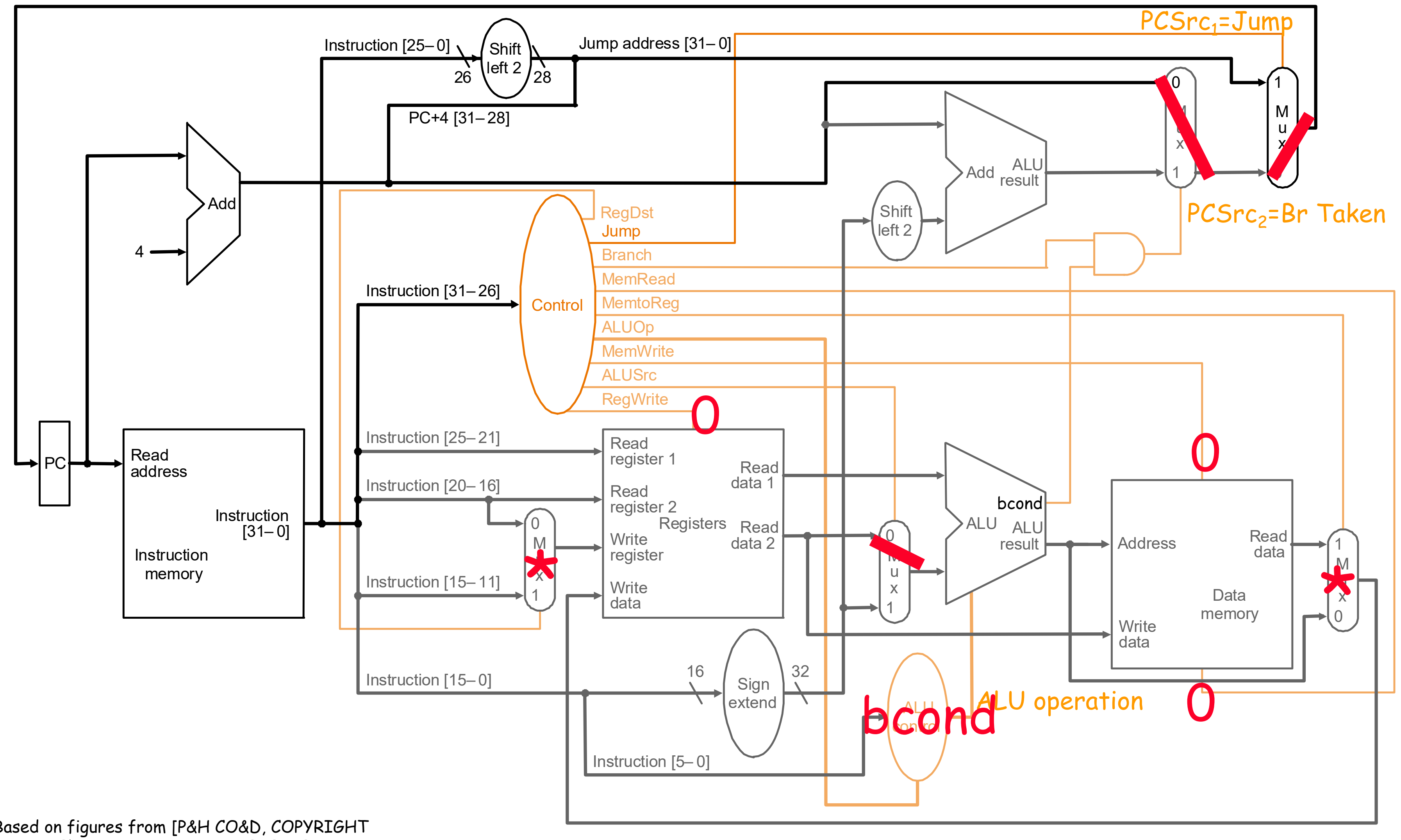
**Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

SW



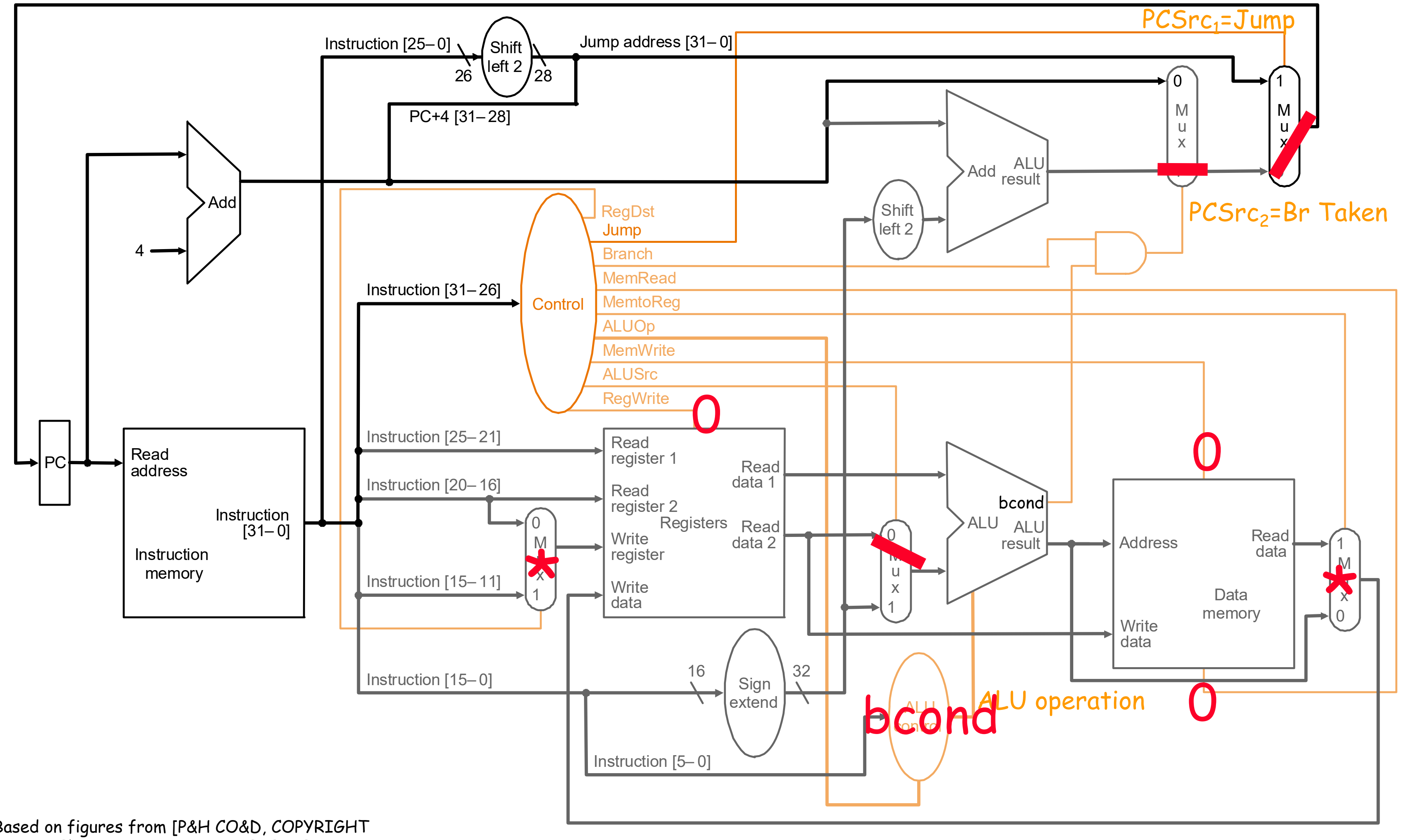
**Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Branch Not Taken



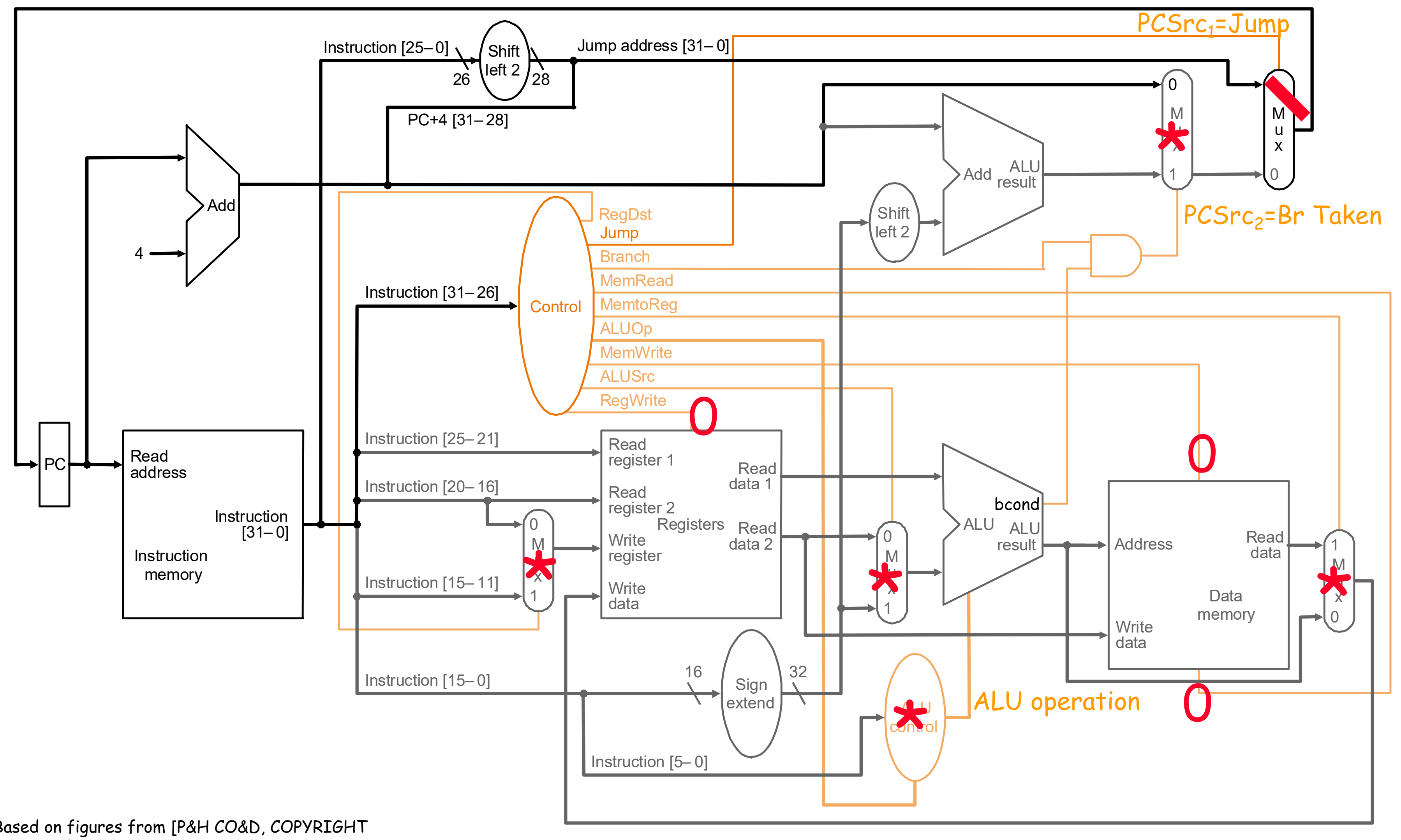
**Based on figures from [P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

Branch Taken



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

Jump



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

Now you know how to design a single-cycle MIPS CPU !!

However, this is a very slow CPU.

Multi-cycle CPU implementations (fast!) in the next lecture.

Question?

Announcements:

Reading:	*Finish reading P&H Ch.4*
Handouts:	*None*