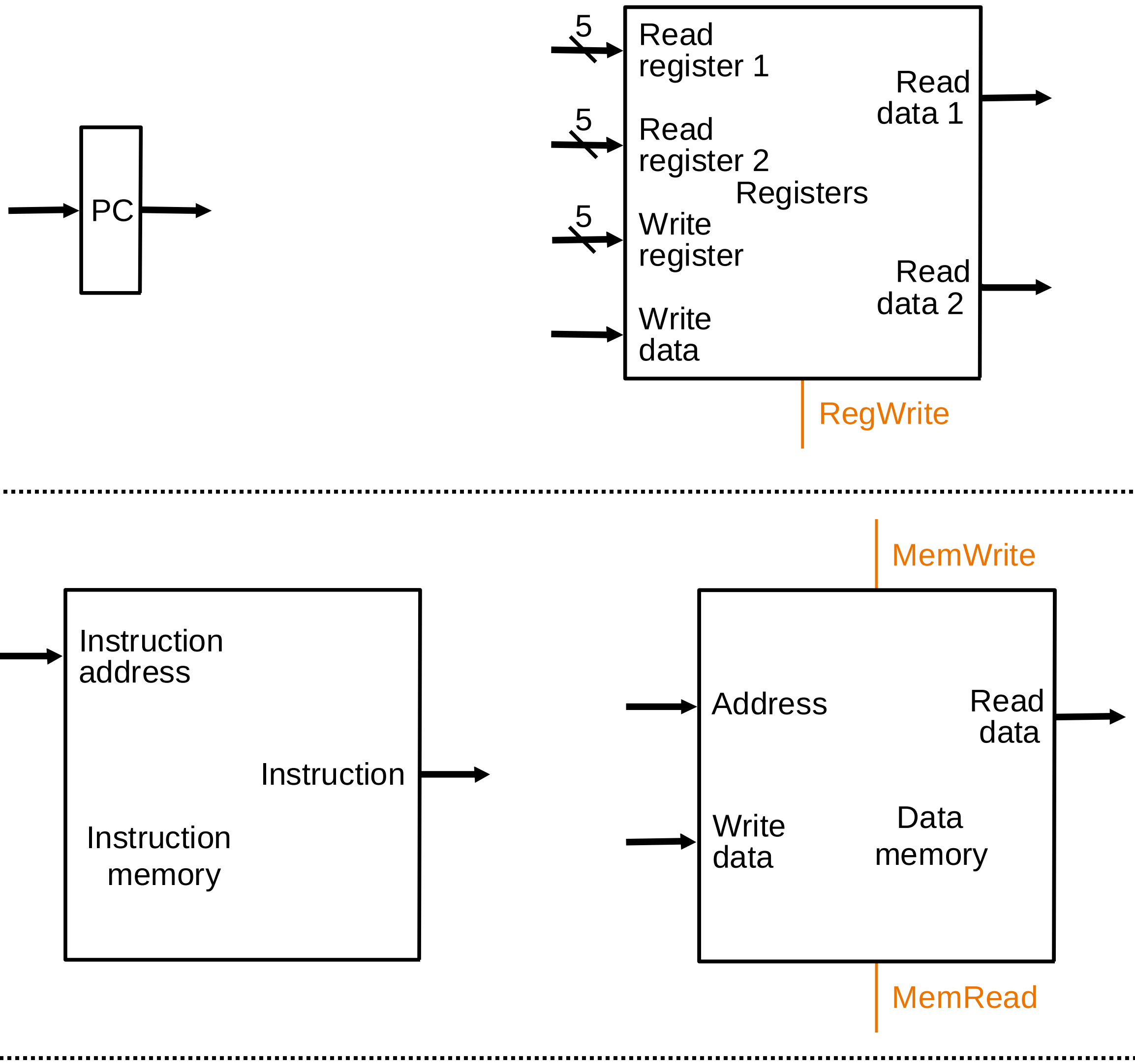
Computer Architecture
Course code: 0521292B

05. MIPS单周期微架构

Jianhua Li
College of Computer and Information
Hefei University of Technology

CPU设计中的状态元素



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

一些假设 (not practical)

- “Magic” memory and register file
- Combinational read
- Synchronous write
- Single-cycle, synchronous memory

一些假设 (not practical)

- “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
- Single-cycle, synchronous memory

一些假设 (not practical)

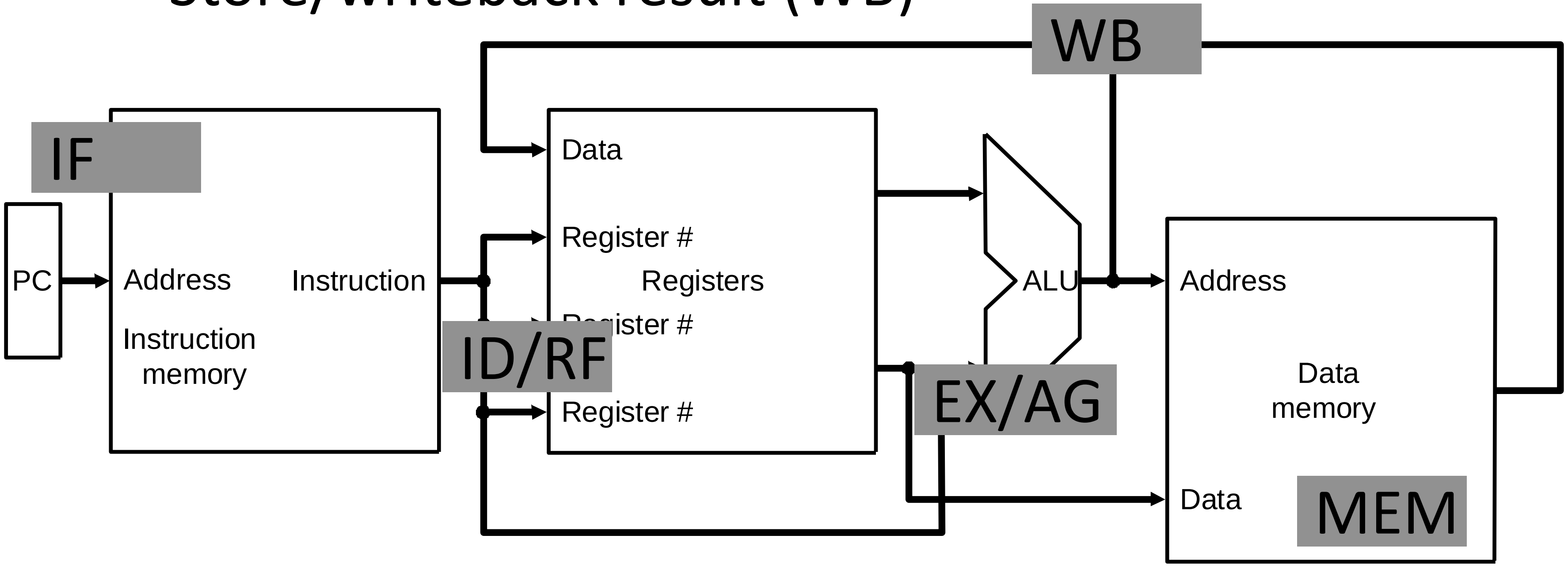
- “Magic” memory and register file
- Combinational read
  - output of the read data port is a **combinational function** of the register file contents and the corresponding read select port
- Synchronous write
  - the selected register is updated on the positive edge clock transition when write enable is asserted
    - Cannot affect read output between clock edges
- Single-cycle, synchronous memory

一些假设 (not practical)

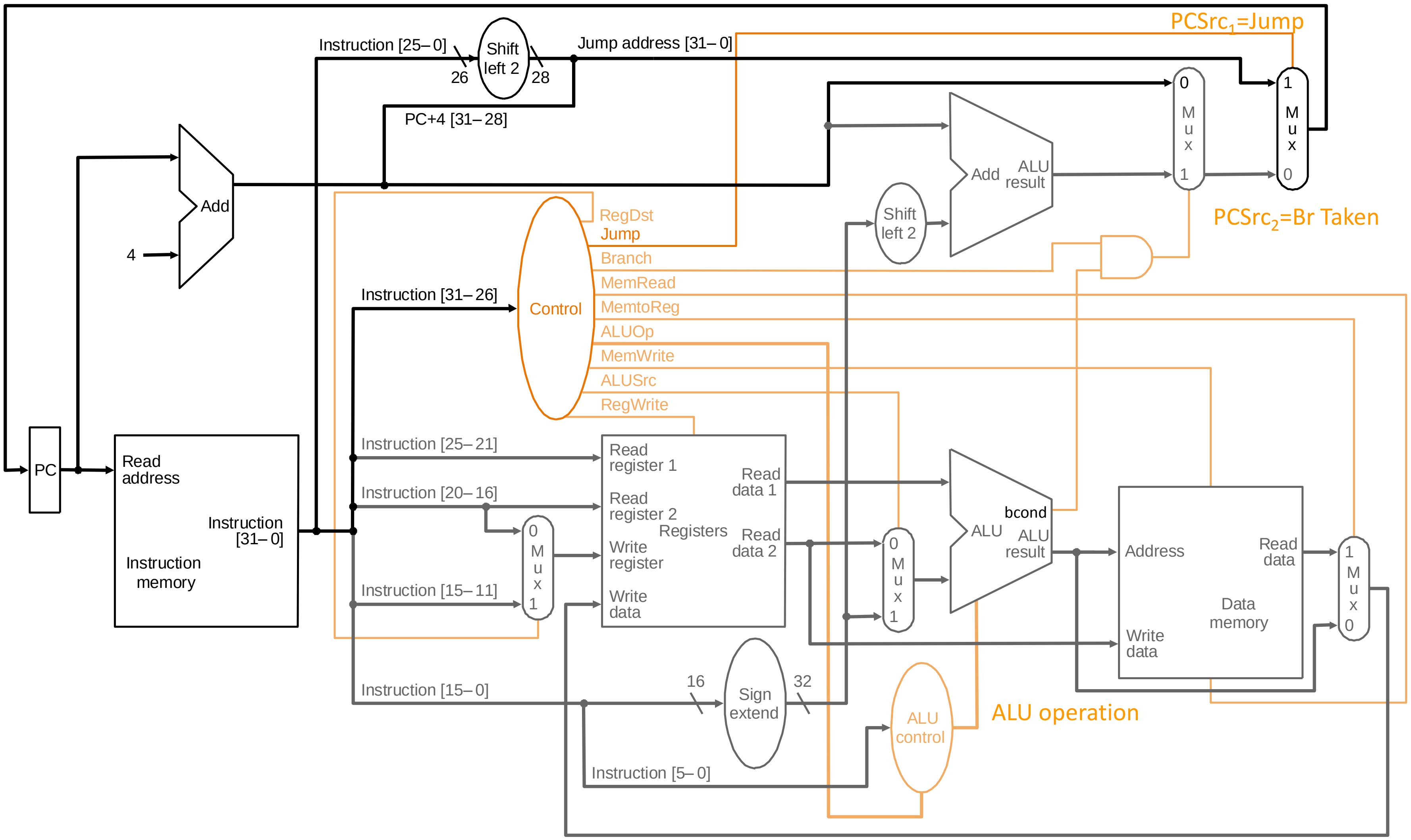
- “Magic” memory and register file
- Combinational read
  - output of the read data port is a **combinational function** of the register file contents and the corresponding read select port
- Synchronous write
  - the selected register is updated on the positive edge clock transition when write enable is asserted
    - Cannot affect read output between clock edges
- Single-cycle, synchronous memory
  - Contrast this with memory that tells when the data is ready
  - i.e., Ready bit: indicating the read or write is done

MIPS的指令处理过程

- 5 generic steps
  - Instruction fetch (IF)
  - Instruction decode and register operand fetch (ID/RF)
  - Execute/Evaluate memory address (EX/AG)
  - Memory operand fetch (MEM)
  - Store/writeback result (WB)



完整的MIPS数据通路



JAL, JR, JALR omitted

算术和逻辑指令的单周期数据通路

R-Type ALU Instructions

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

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

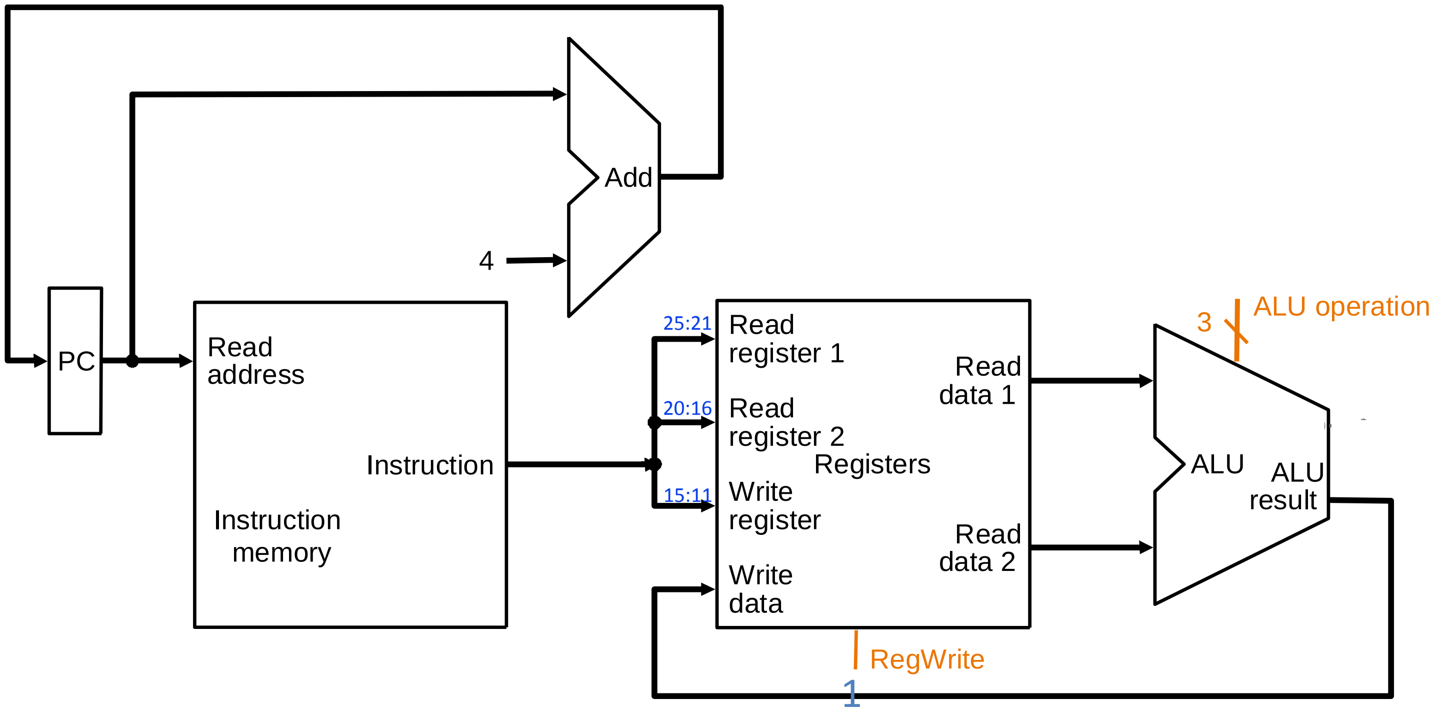
- Machine encoding

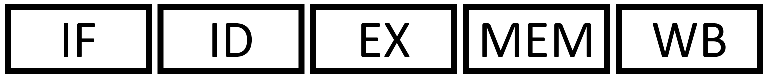
0	rs	rt	rd	0	ADD	R-type
6-bit	5-bit	5-bit	5-bit	5-bit	6-bit	

- Semantics

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

ALU Datapath





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

Combinational state update logic

I-Type ALU Instructions

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

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

- Machine encoding

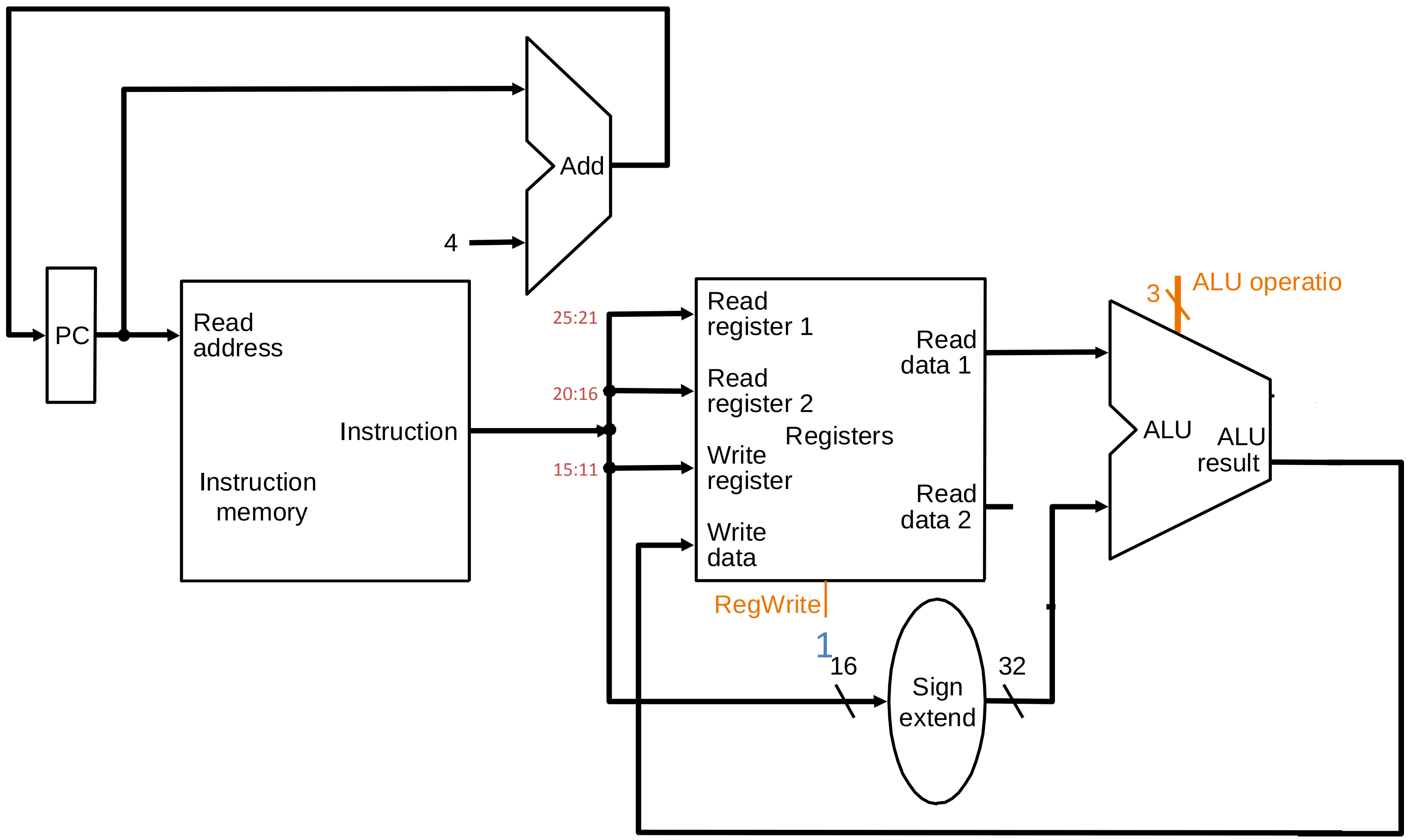
ADDI	rs	rt	immediate
6-bit	5-bit	5-bit	16-bit

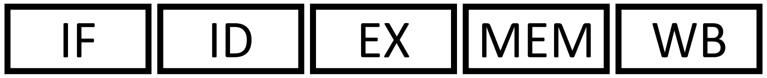
I-type

- Semantics

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

Datapath for R and I-Type ALU Insts.

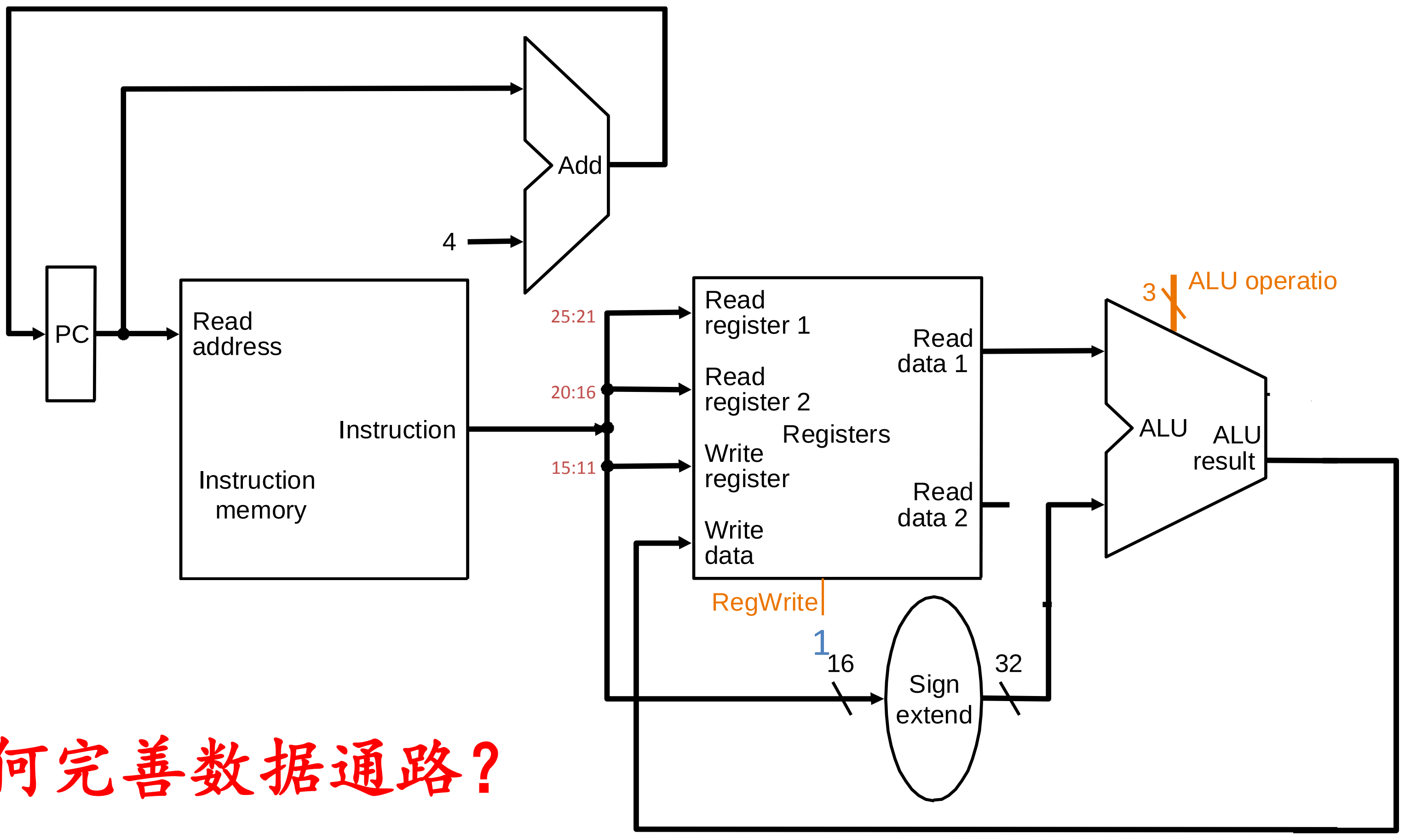




```
if MEM[PC] == ADDI rt rs immediate
    GPR[rt] ← GPR[rs] + sign-extend (immediate)
    PC ← PC + 4
```

Combinational state update logic

Datapath for R and I-Type ALU Insts.



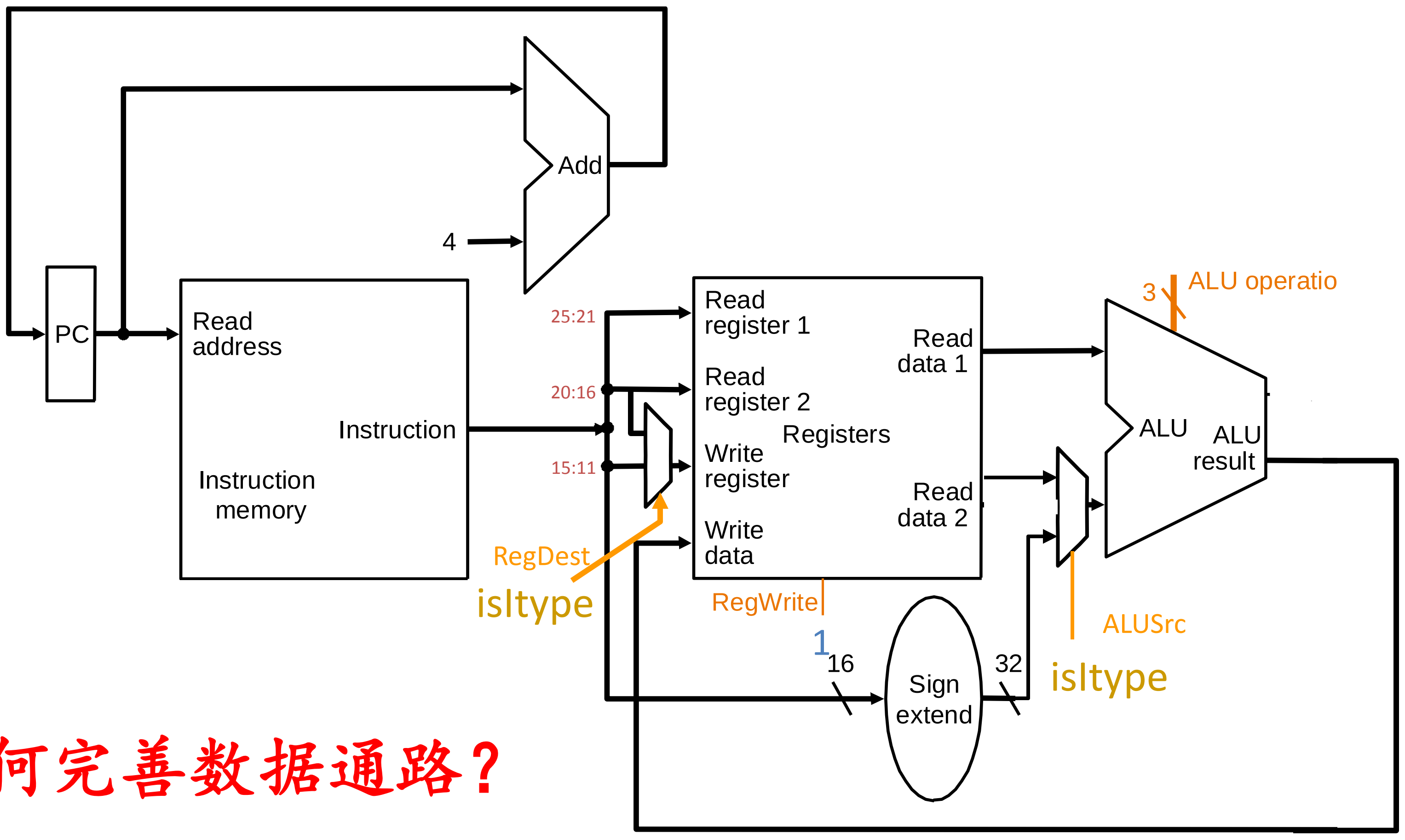
如何完善数据通路？

IF	ID	EX	MEM	WB

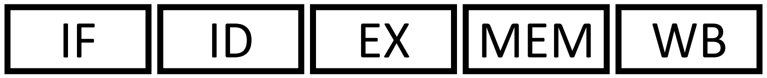
```
if MEM[PC] == ADDI rt rs immediate
    GPR[rt] ← GPR[rs] + sign-extend (immediate)
    PC ← PC + 4
```

→ Combinational state update logic

Datapath for R and I-Type ALU Insts.



如何完善数据通路？



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

→ Combinational state update logic

访存指令的
单周期数据通路

Load Instructions

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

  LW rt_{reg} offset_{16} $(\text{base}_{\text{reg}})$

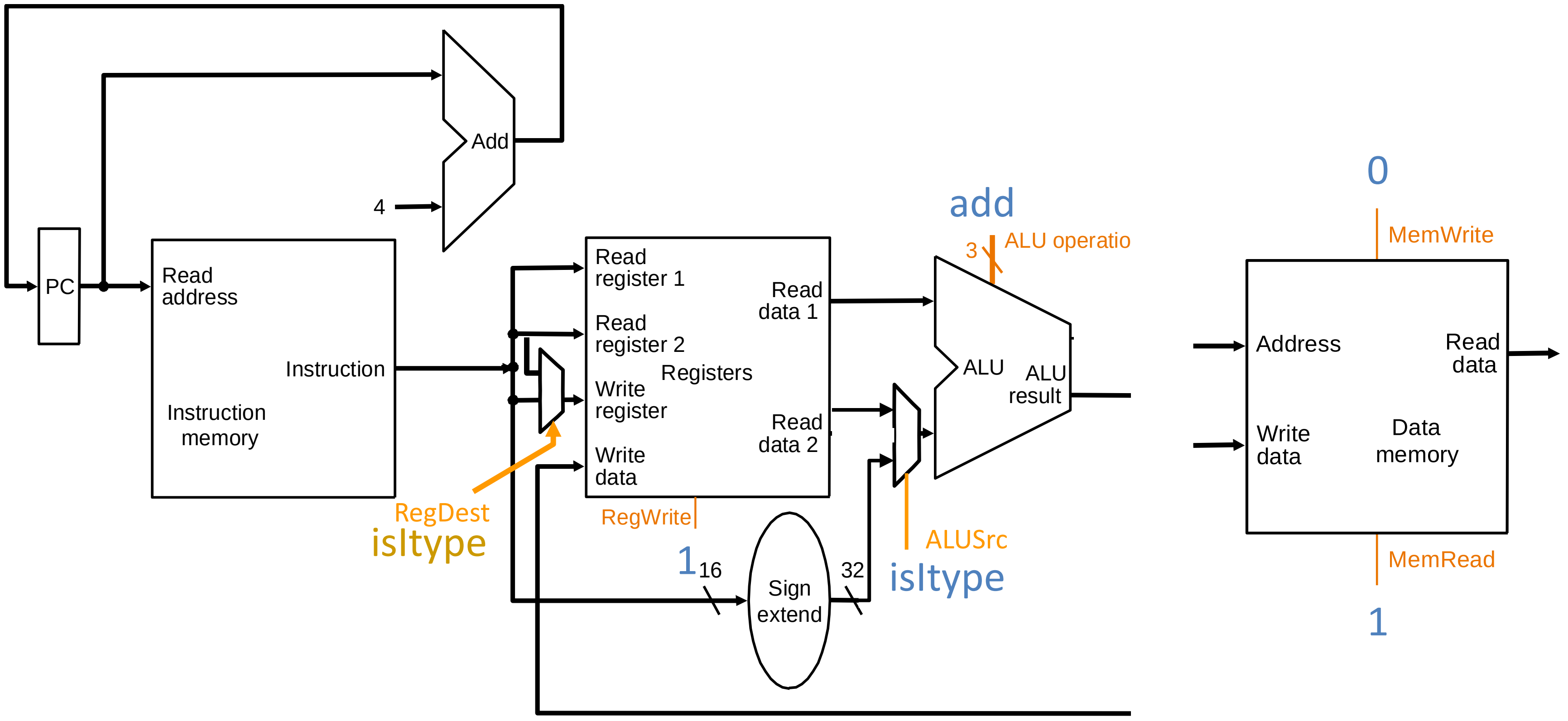
- Machine encoding

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

- Semantics

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

LW数据通路

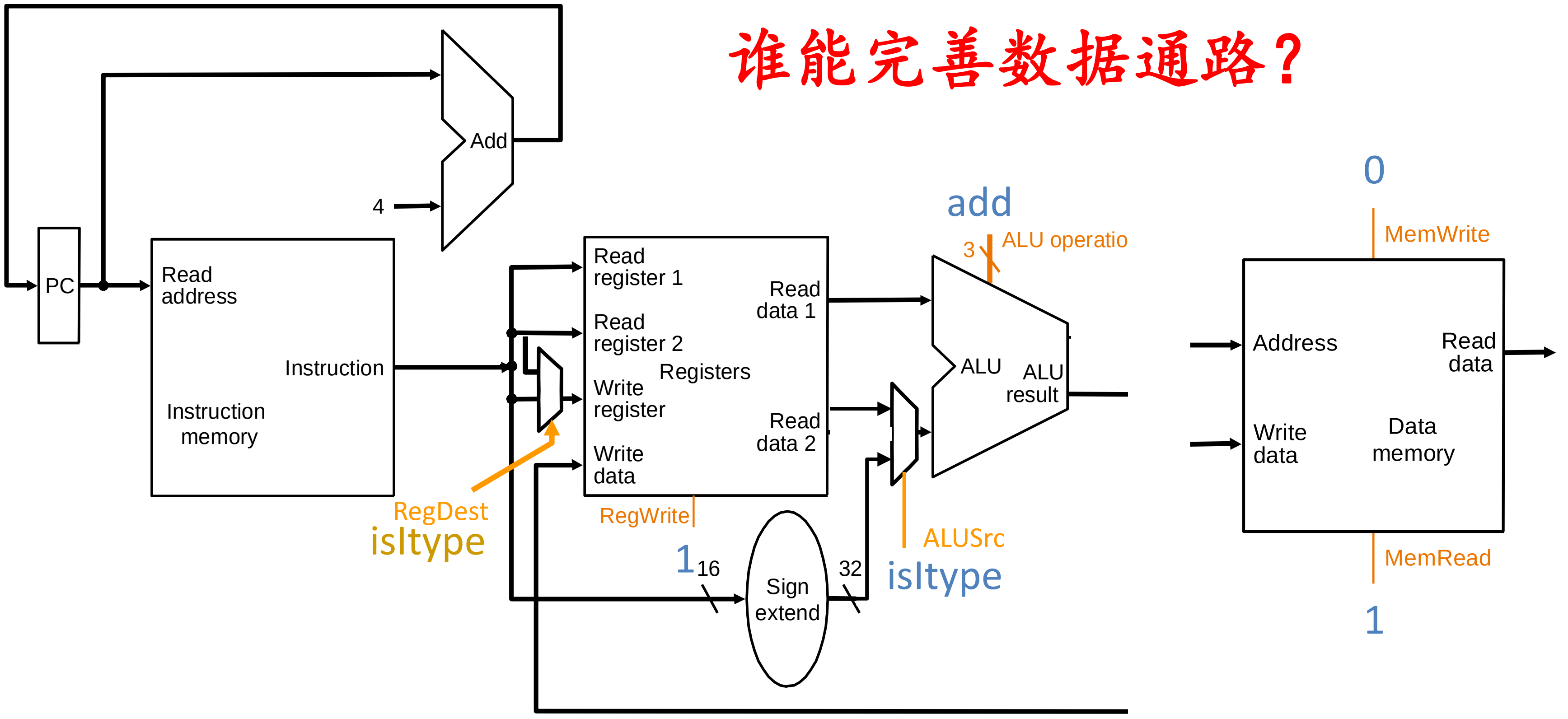


if MEM[PC]==LW rt offset$_{16}$ (base)
    EA = sign-extend(offset) + GPR[base]
    GPR[rt] ← MEM[translate(EA)]
    PC ← PC + 4

IF	ID	EX	MEM	WB

→ Combinational state update logic

LW数据通路



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

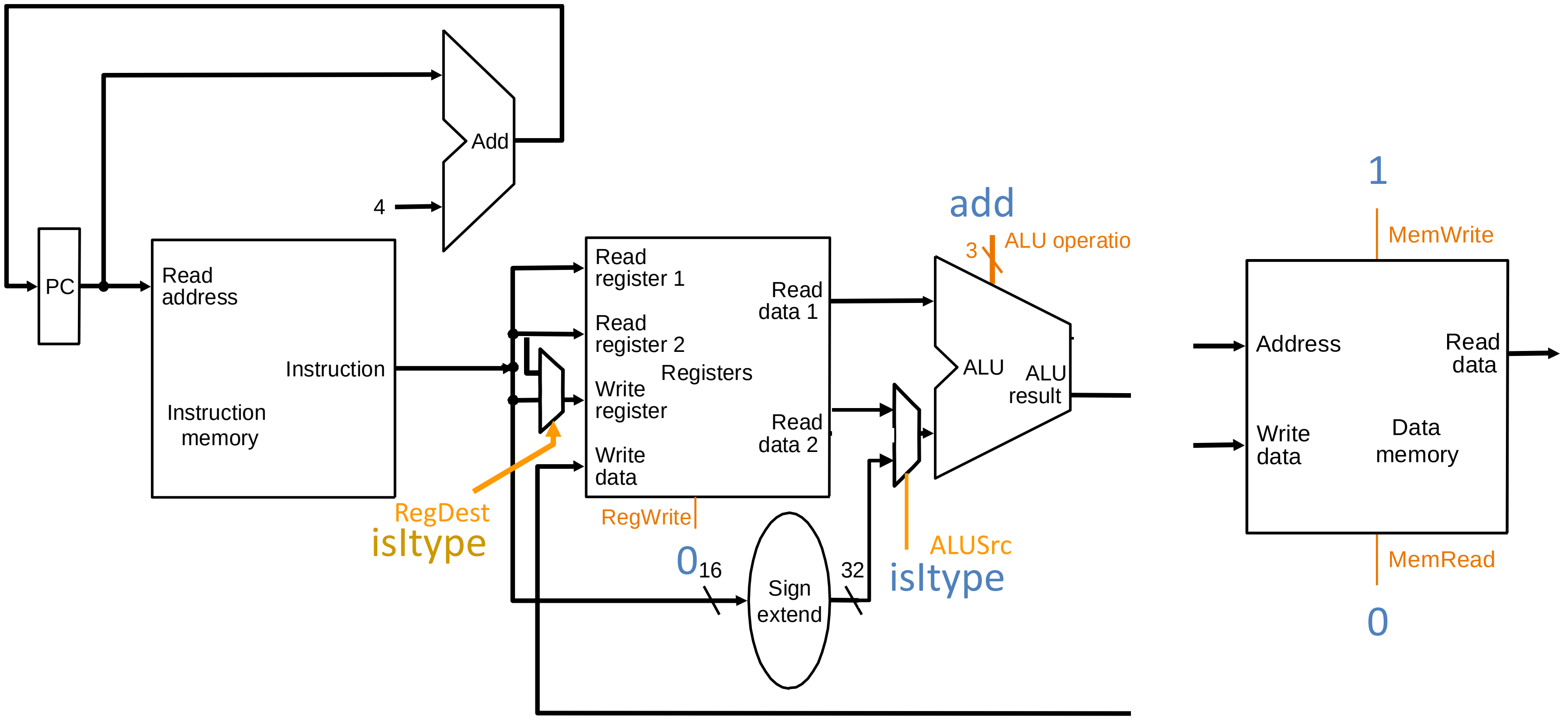
I-type

- 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数据通路

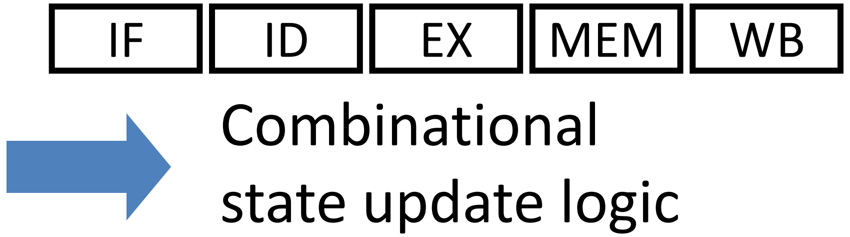


if MEM[PC]==SW rt offset$_{16}$ (base)

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

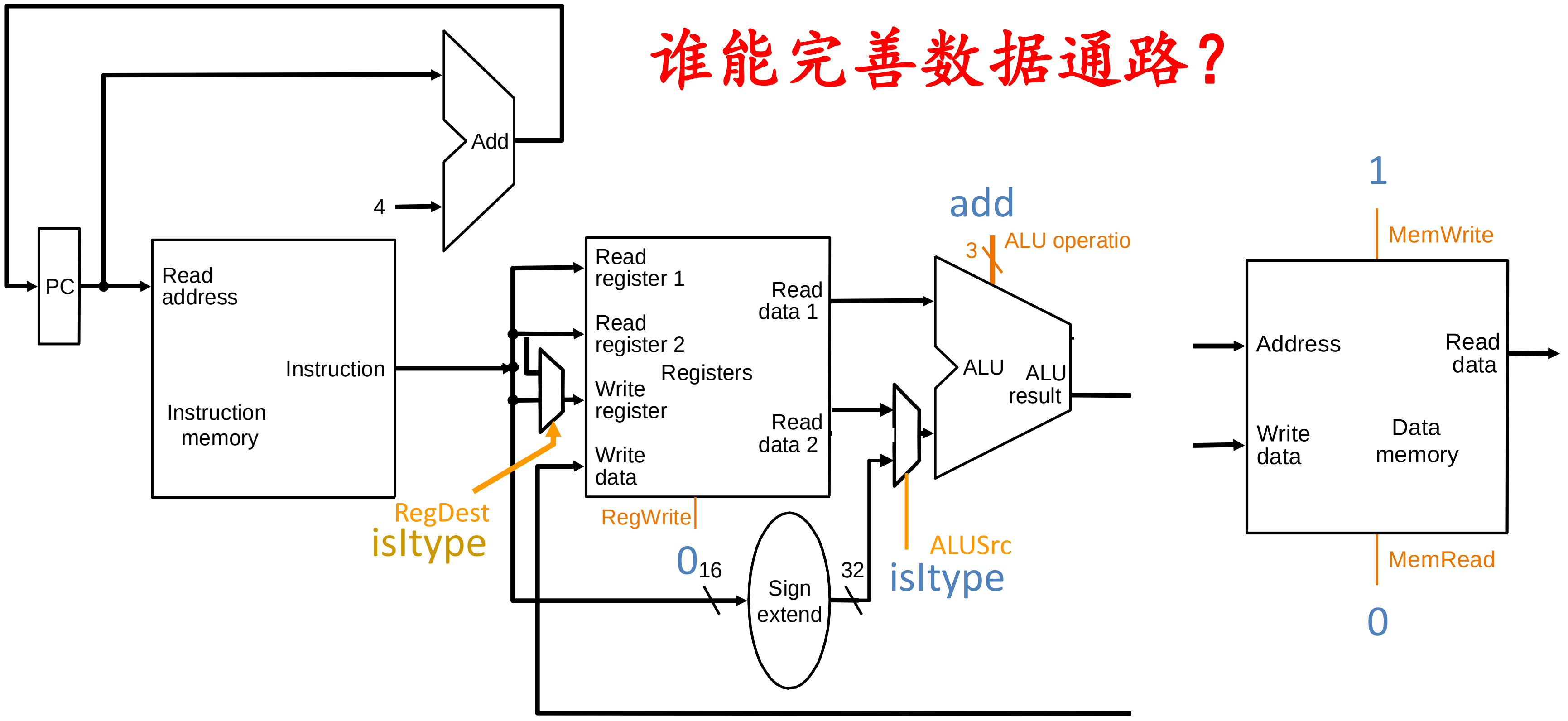
MEM[translate(EA)] ← GPR[rt]

PC ← PC + 4



Combinational state update logic

SW数据通路

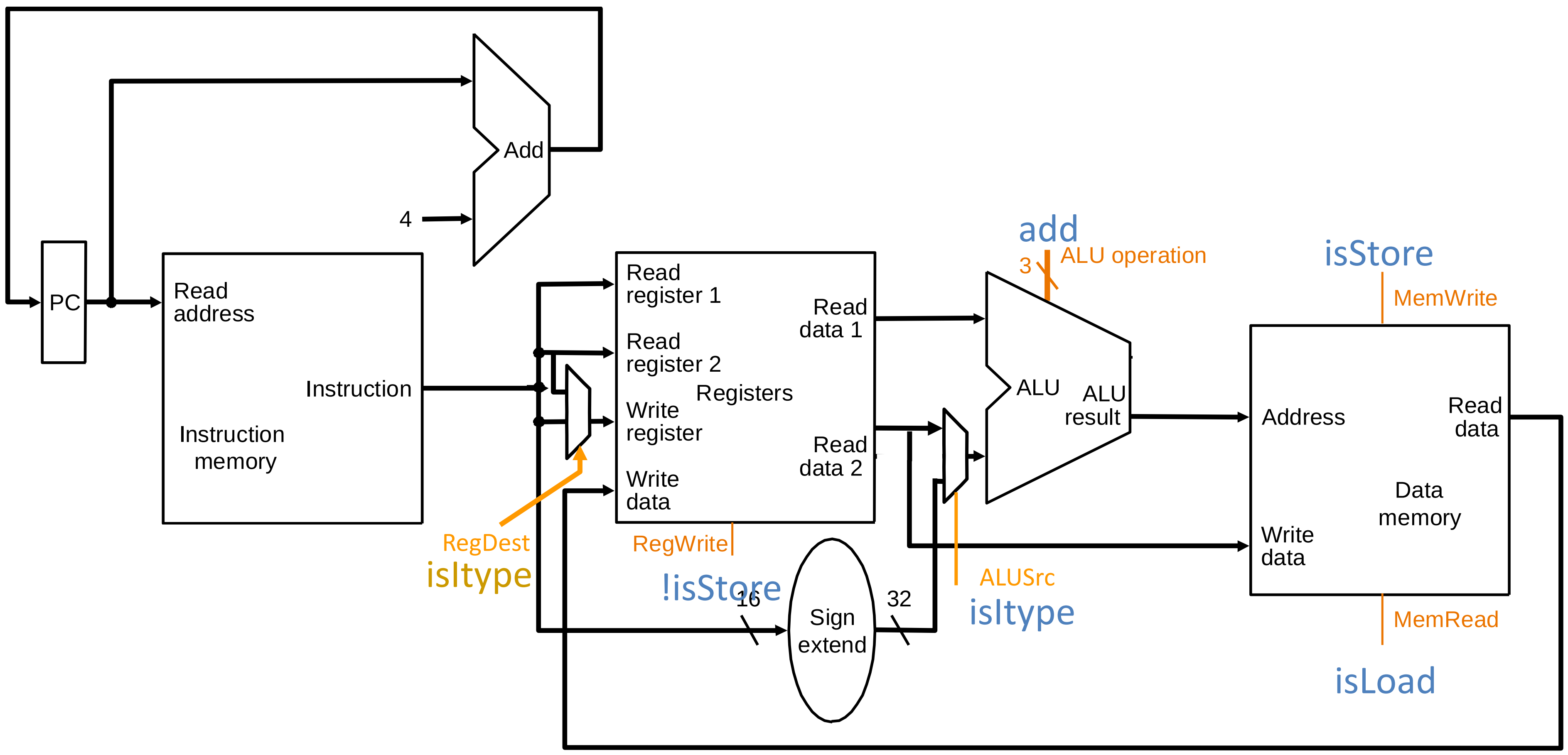


if MEM[PC]==SW rt offset$_{16}$ (base)
    EA = sign-extend(offset) + GPR[base]
    MEM[translate(EA)] ← GPR[rt]
    PC ← PC + 4

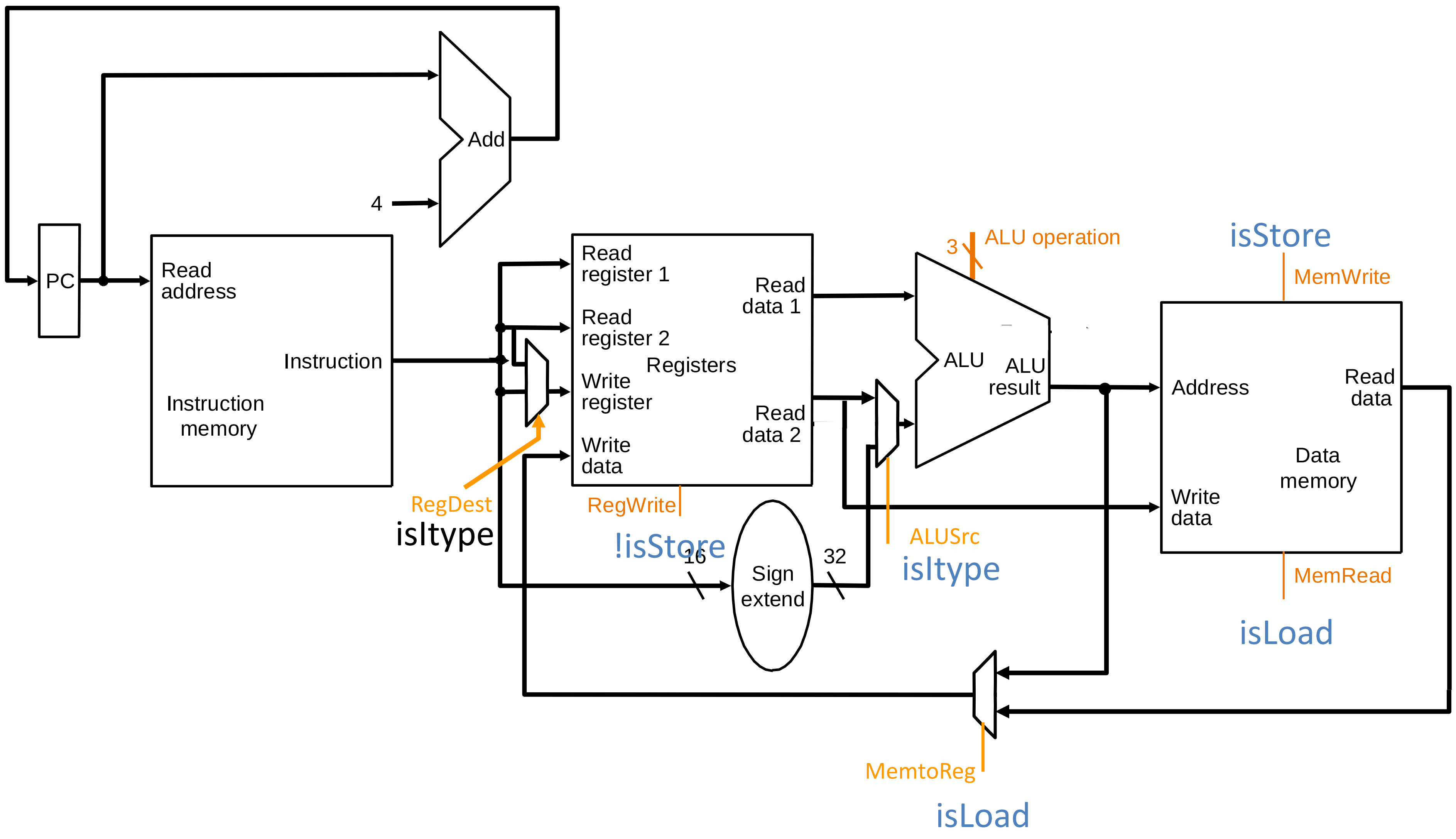
IF	ID	EX	MEM	WB

→ Combinational state update logic

Load-Store数据通路



非控制流指令的数据通路



控制流指令的
单周期数据通路

无条件跳转指令

- Assembly

  J immediate$_{26}$

- Machine encoding

J	immediate
6-bit	26-bit

J-type

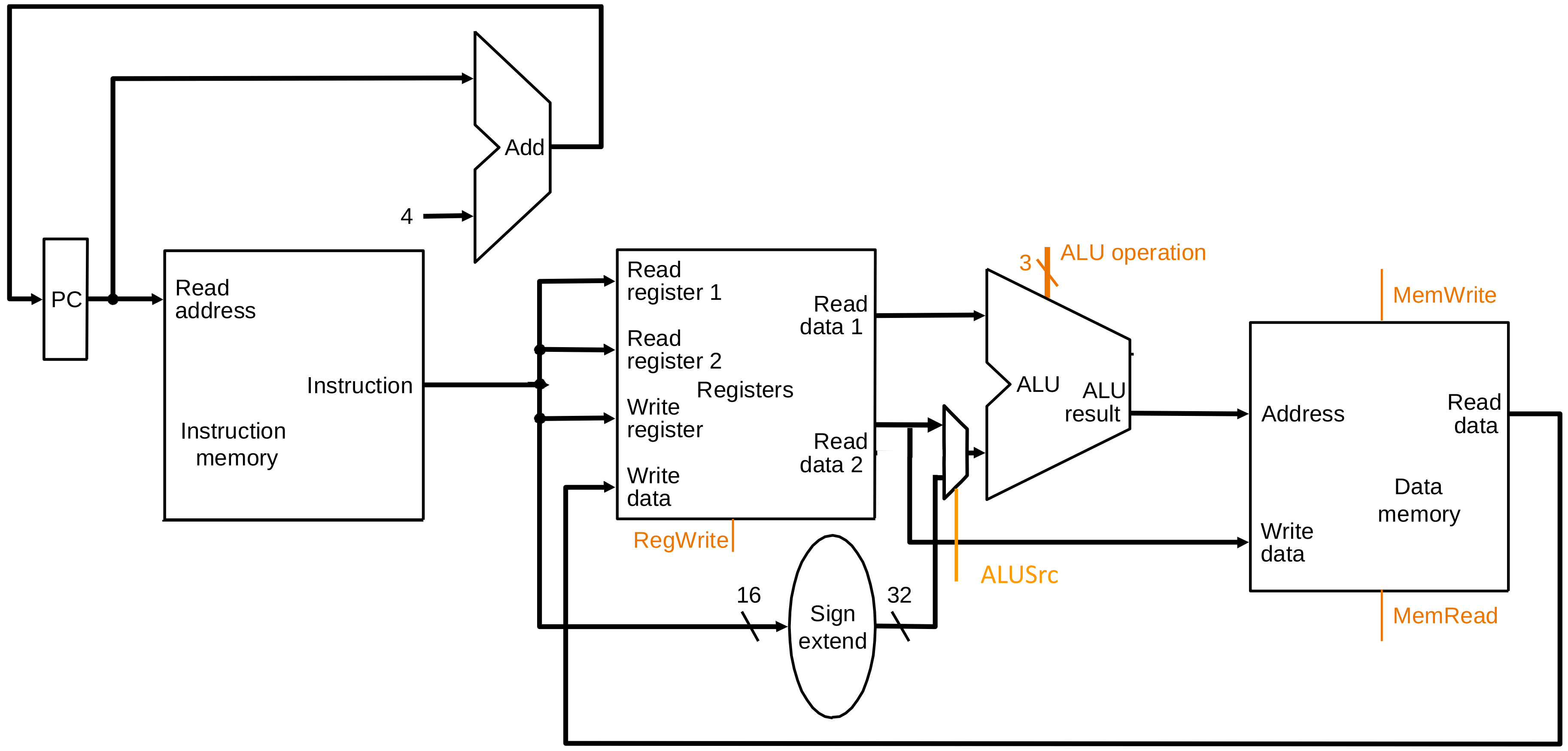
- Semantics

  if MEM[PC]==J immediate$_{26}$

      target = { PC[31:28], immediate$_{26}$, 2’ b00 }

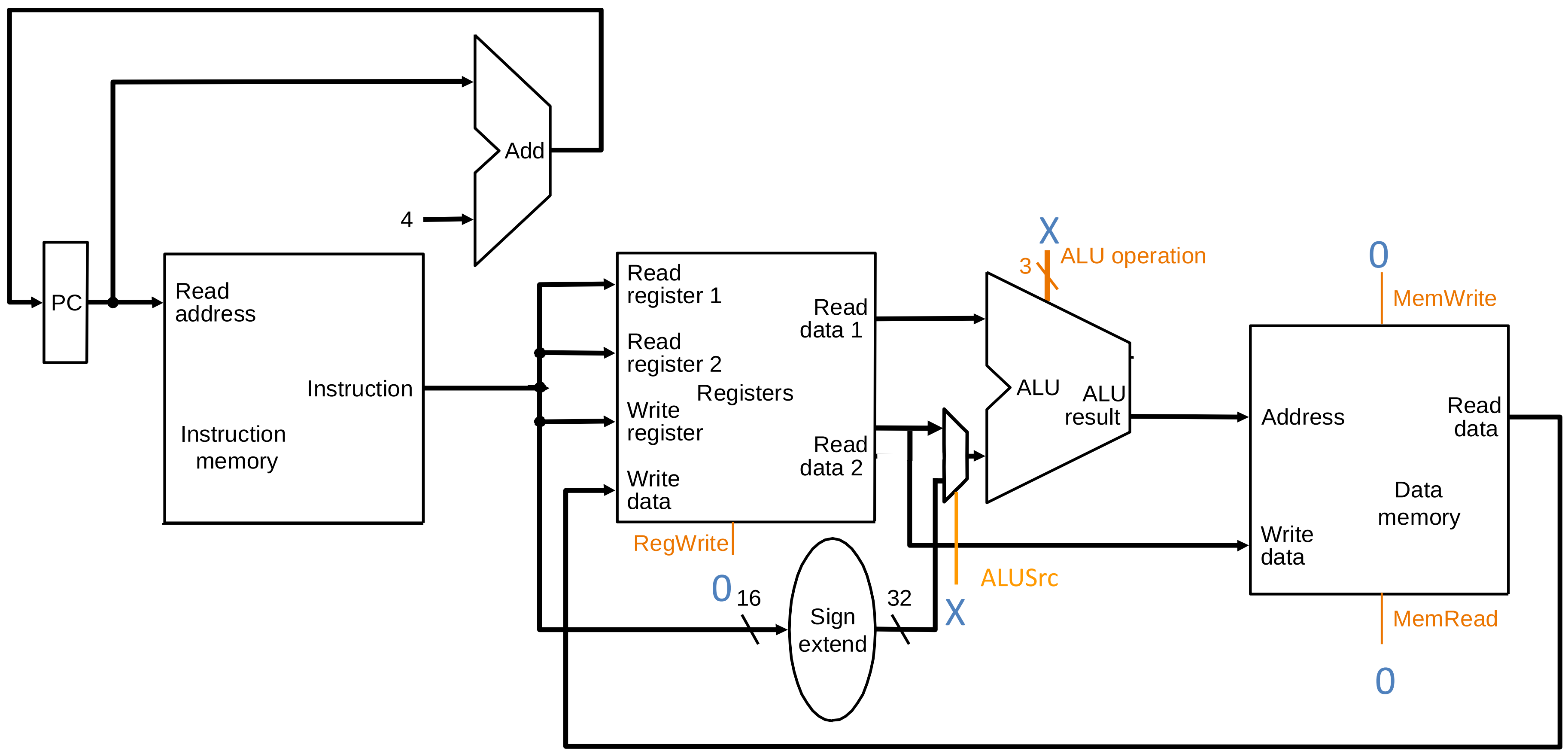
      PC $\leftarrow$ target

无条件跳转指令的数据通路



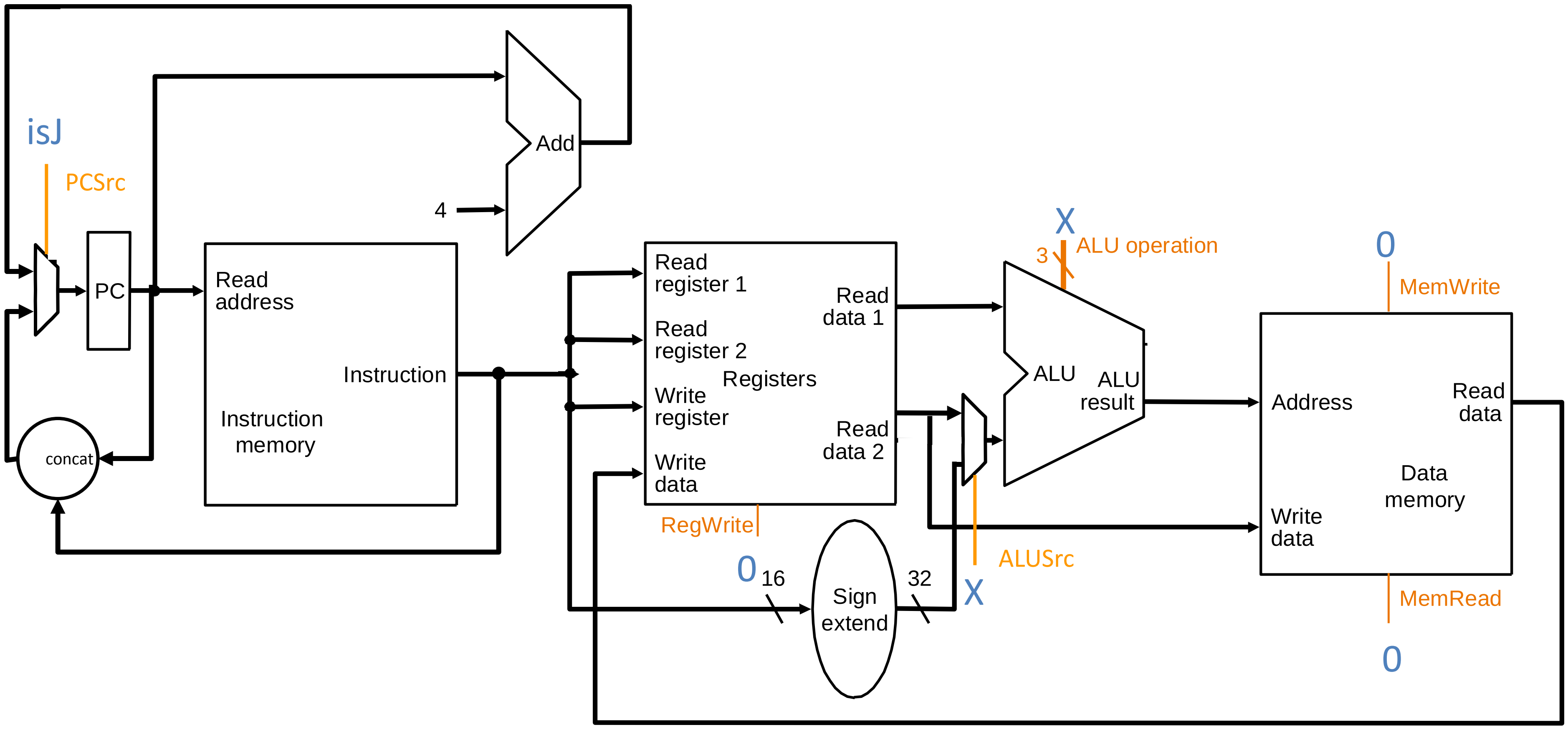
```
if MEM[PC]==J immediate26
   PC = { PC[31:28], immediate26, 2’b00 }
```

无条件跳转指令的数据通路



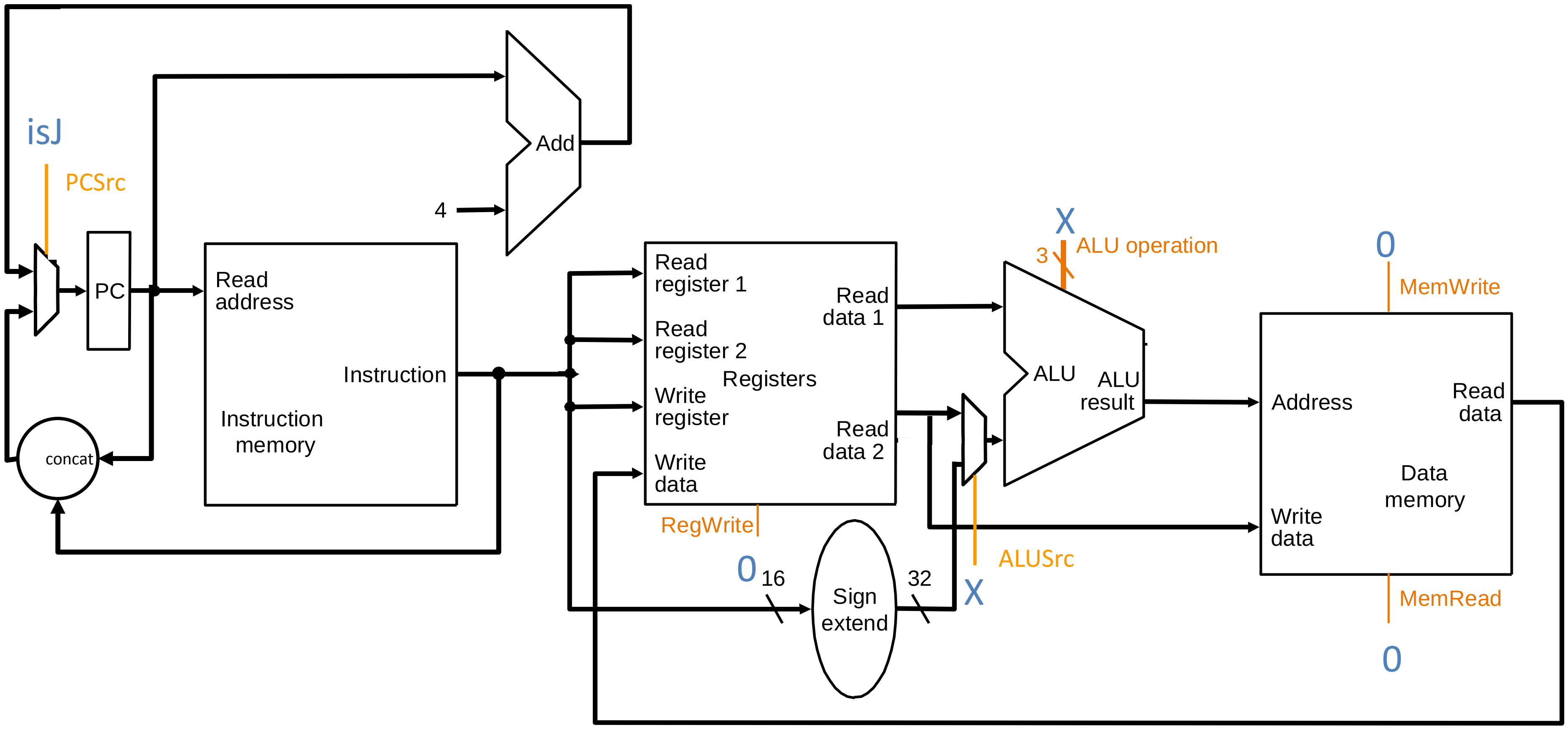
if MEM[PC]==J immediate26
    PC = { PC[31:28], immediate26, 2’b00 }

无条件跳转指令的数据通路



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

无条件跳转指令的数据通路



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

What about JR, JAL, JALR?

条件分支指令

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

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

- Machine encoding

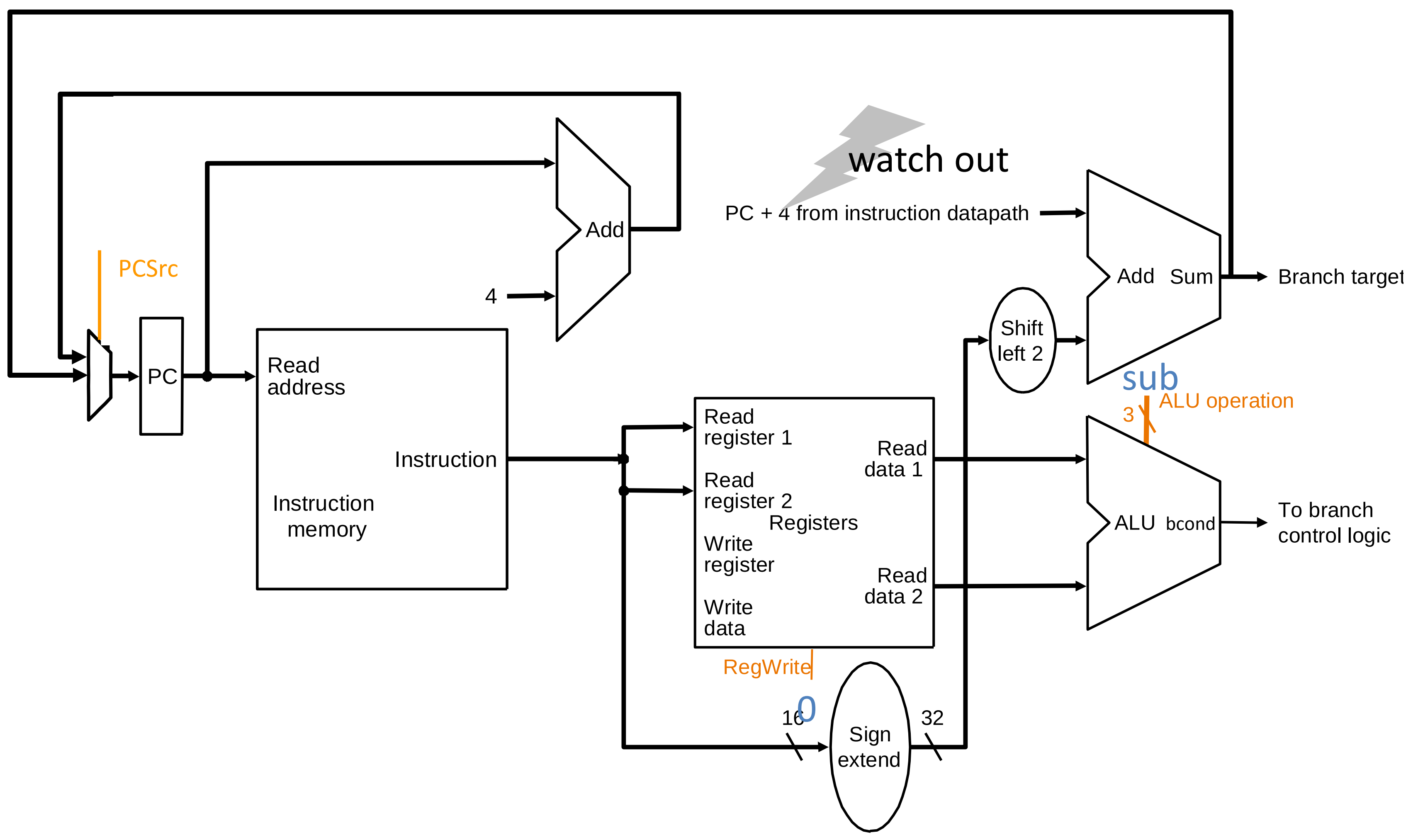
BEQ	rs	rt	immediate
6-bit	5-bit	5-bit	16-bit

I-type

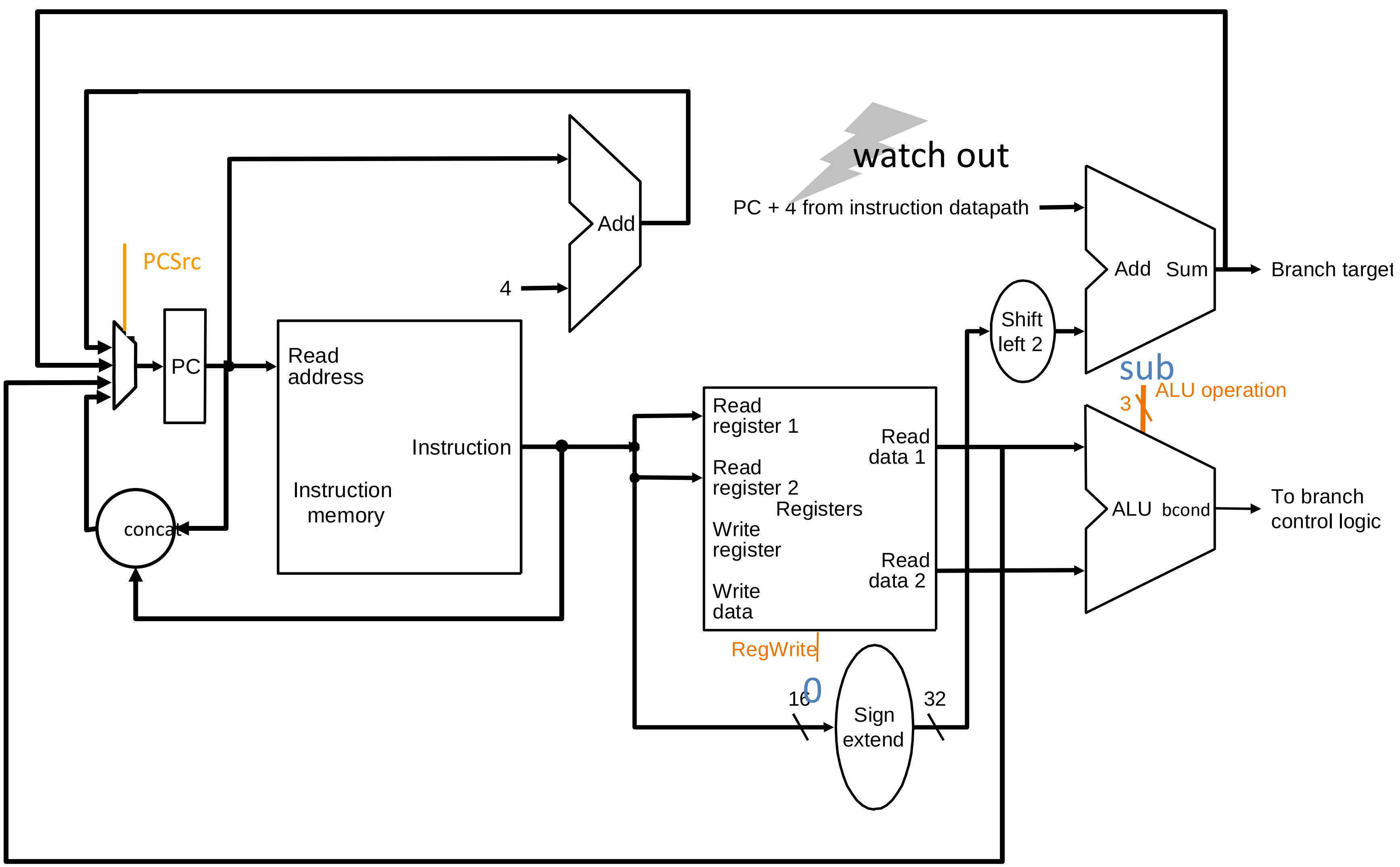
- Semantics (assuming no branch delay slot)

```
if MEM[PC]==BEQ rs rt $immediate_{16}$
    target = PC + 4 + sign-extend(immediate) x 4
    if GPR[rs]==GPR[rt] then    PC $\leftarrow$ target
                        else    PC $\leftarrow$ PC + 4
```

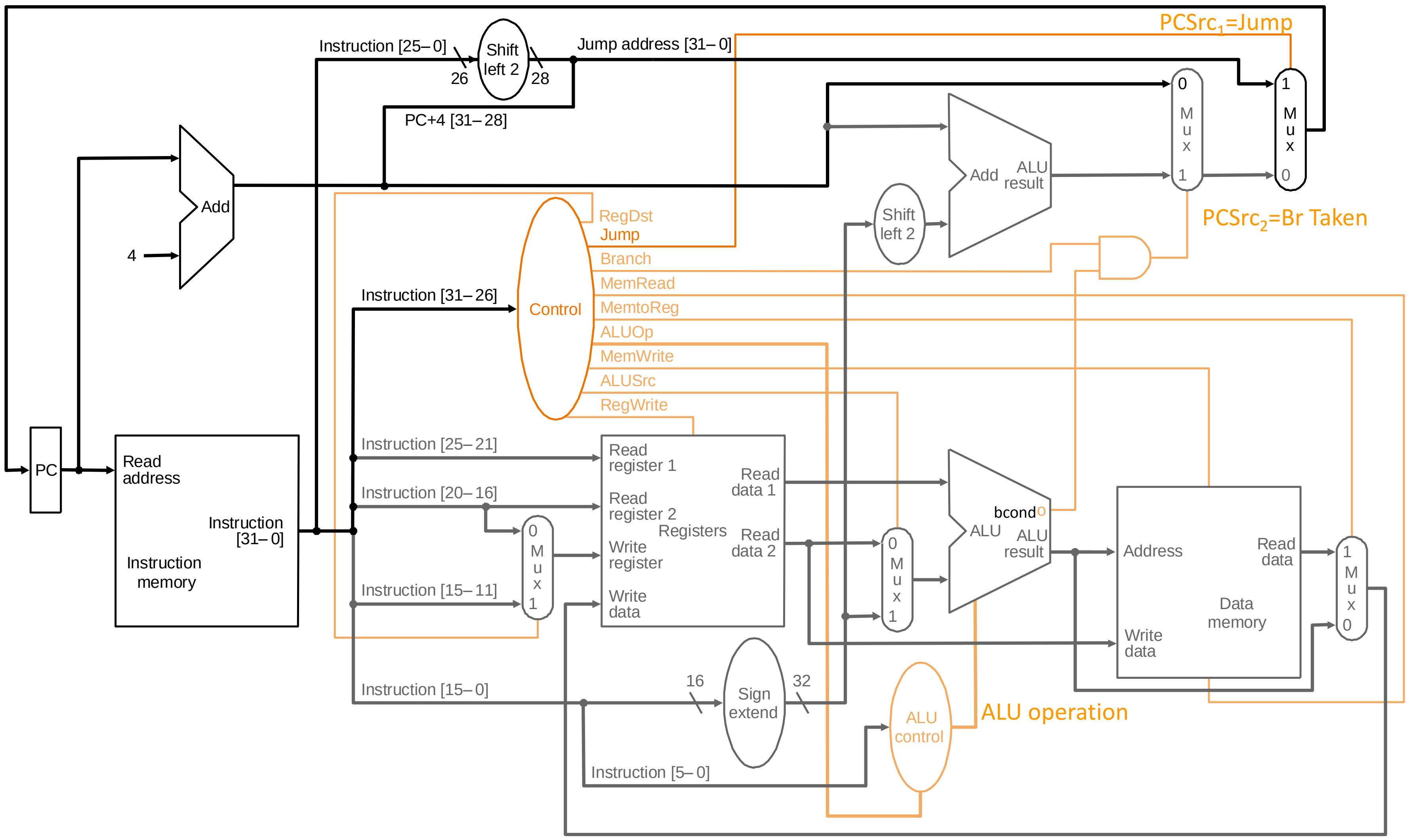
Conditional Branch Datapath (for you to finish)



Conditional Branch Datapath (for you to finish)



Putting It All Together

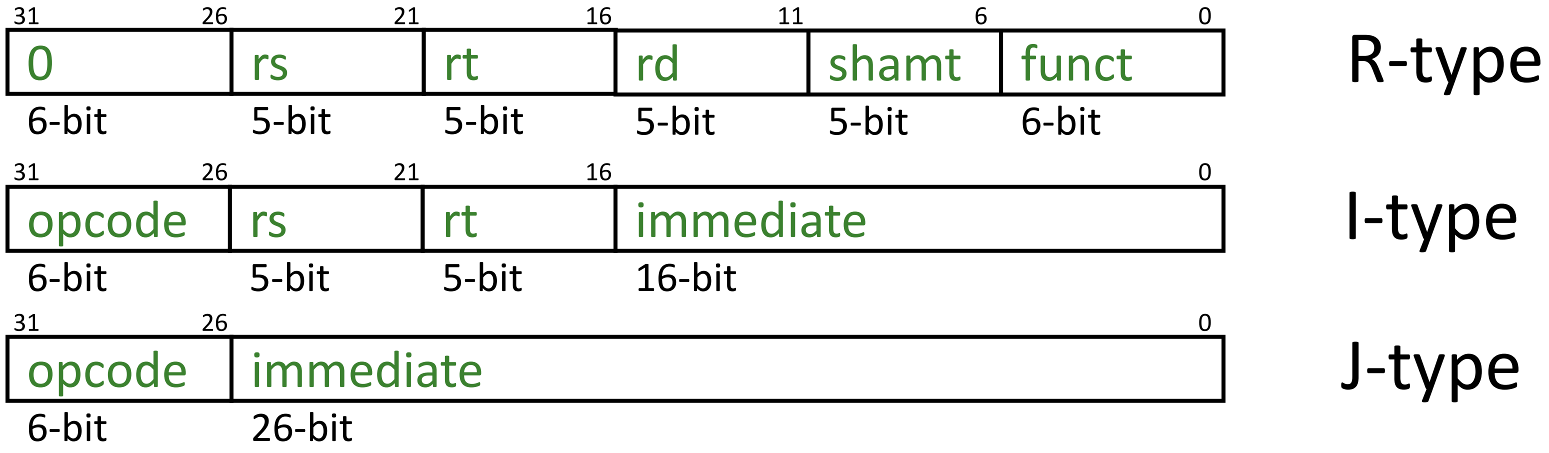


JAL, JR, JALR omitted

单周期控制逻辑

单周期硬布线控制

- 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

单个比特控制信号

	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

单个比特控制信号

	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

- case opcode

  '0' ⇒ select operation according to funct

  'ALUi' ⇒ selection operation according to opcode

  'LW' ⇒ select addition

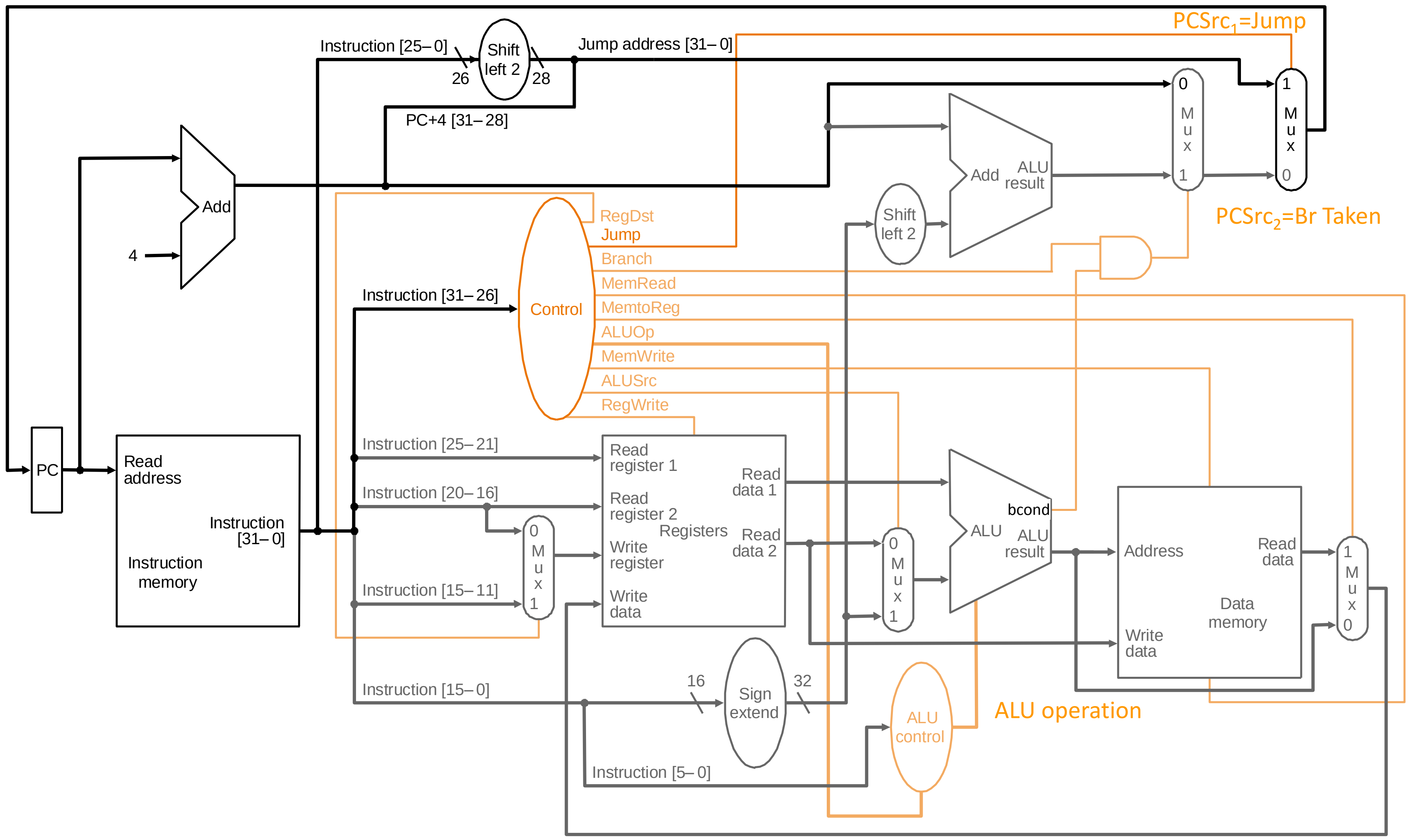
  'SW' ⇒ select addition

  'Bxx' ⇒ select bcond generation function

  __ ⇒ don't care

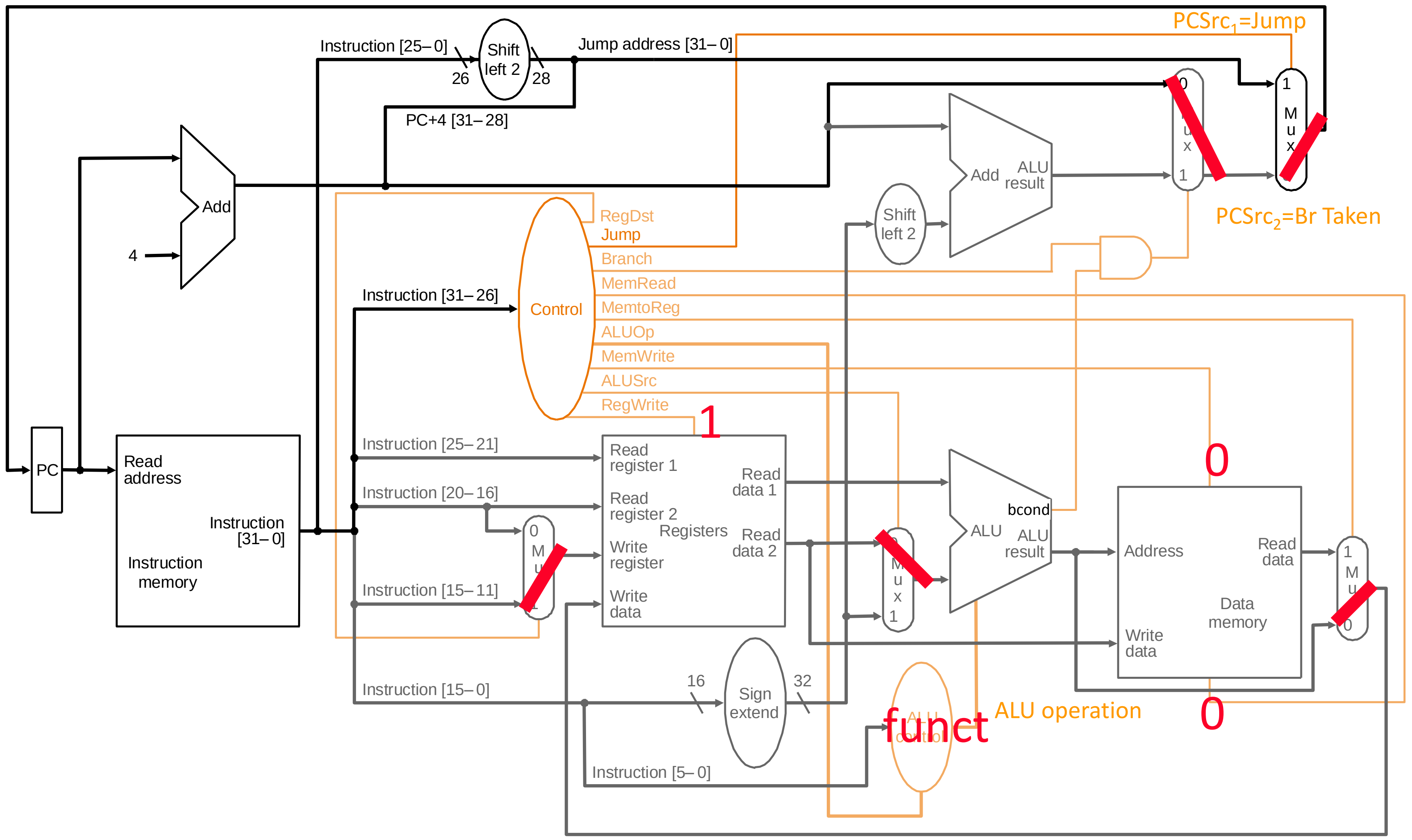
- Example ALU operations
  - ADD, SUB, AND, OR, XOR, NOR, etc.
  - bcond on equal, not equal, LE zero, GT zero, etc.

R-Type ALU

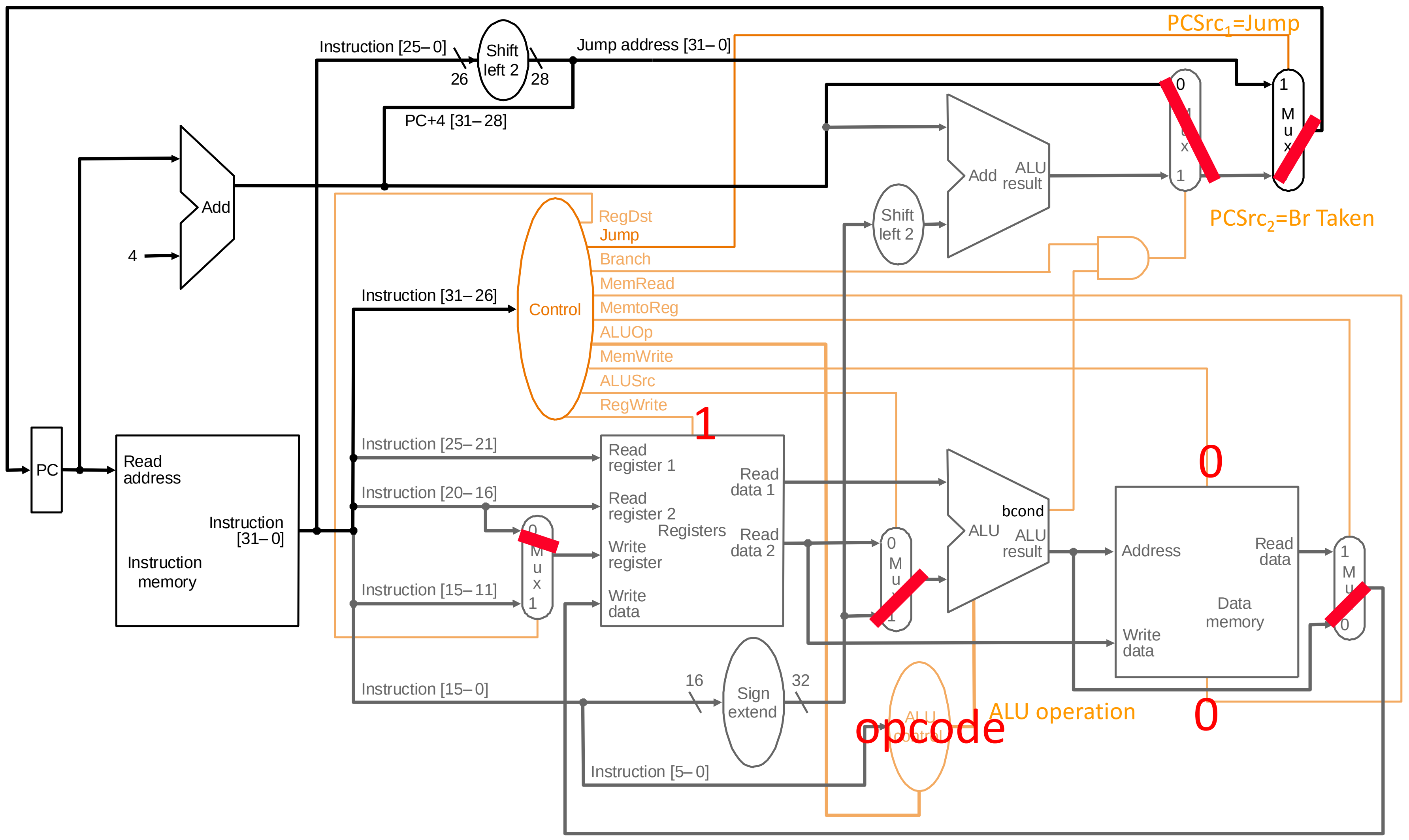


PCSrc$_1$=Jump
PCSrc$_2$=Br Taken
ALU operation
Instruction [25–0]
26
Shift left 2
28
Jump address [31–0]
PC+4 [31–28]
Add
4
Add
ALU result
0 M u x 1
1 M u x 0
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
PC
Read address
Instruction [31–0]
Instruction memory
0 M u x 1
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
16
Sign extend
32
0 M u x 1
ALU
bcond
ALU result
ALU control
Address
Read data
Write data
Data memory
1 M u x 0

R-Type ALU

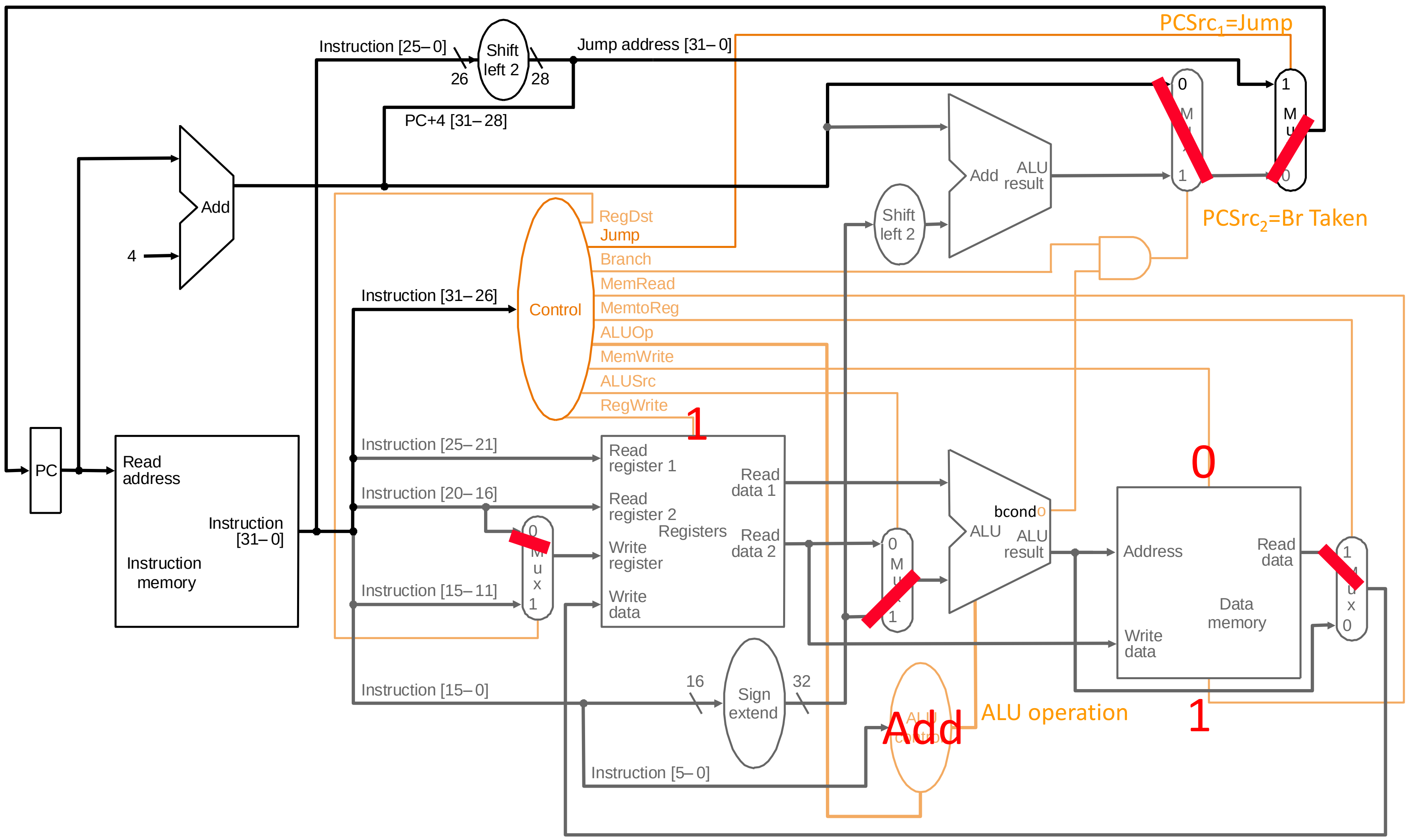

PCSrc$_1$=Jump
Instruction [25–0]
26
Shift left 2
28
Jump address [31–0]
PC+4 [31–28]
Add
4
0
M
u
x
1
1
M
u
x
ALU
Add result
PCSrc$_2$=Br Taken
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
1
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
bcond
ALU
ALU result
0
Address
Read data
Data memory
Write data
Instruction [15–0]
16
Sign extend
32
Instruction [5–0]
funct
ALU control
ALU operation
0

I-Type ALU



PCSrc$_1$=Jump
Instruction [25–0]
26
Shift left 2
28
Jump address [31–0]
PC+4 [31–28]
Add
4
Add
ALU result
Shift left 2
0
M
u
x
1
1
M
u
x
0
PCSrc$_2$=Br Taken
Instruction [31–26]
Control
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
1
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
bcond
ALU
ALU result
0
Address
Read data
Data memory
Write data
0
Instruction [15–0]
16
Sign extend
32
Instruction [5–0]
ALU control
opcode
ALU operation
0

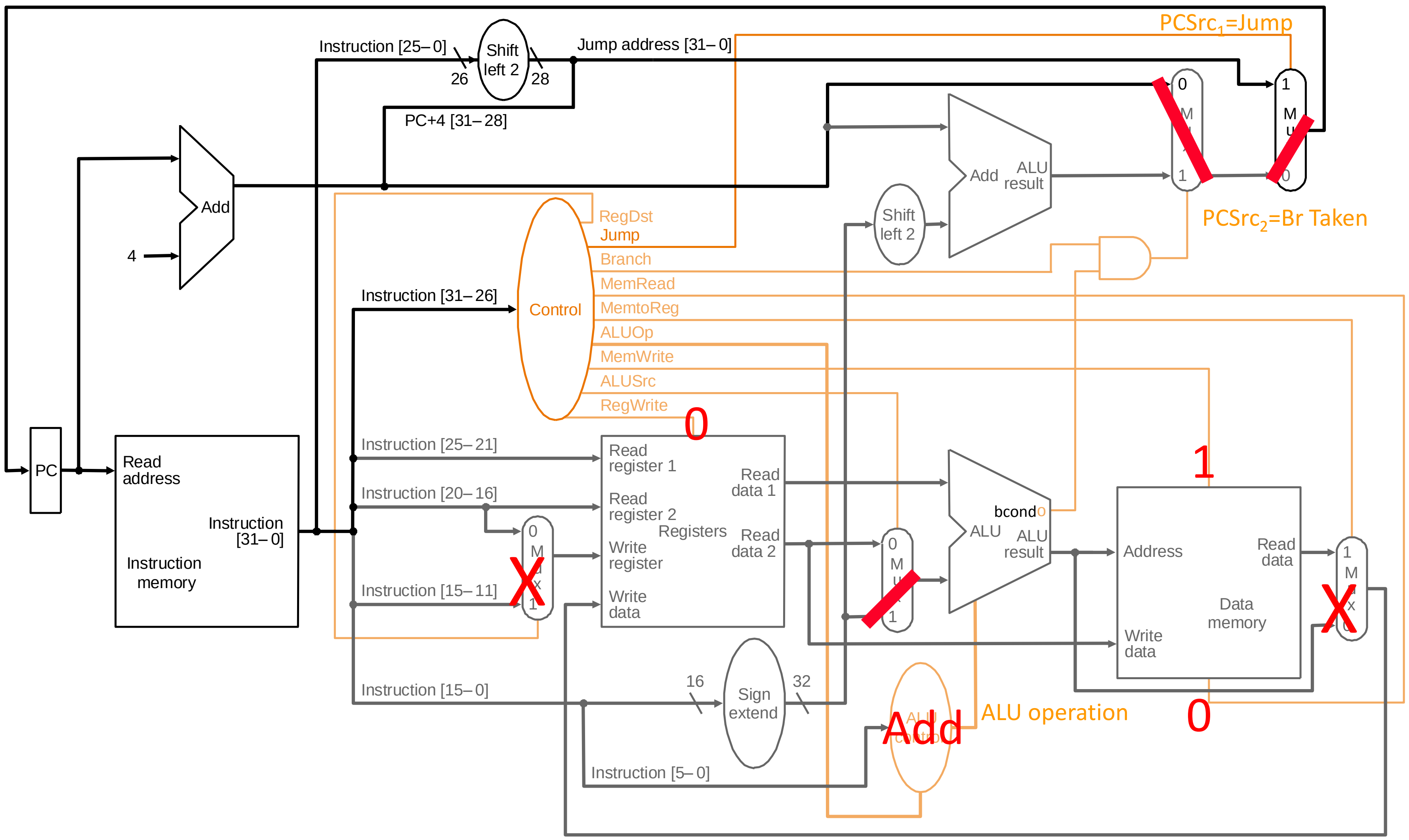
LW



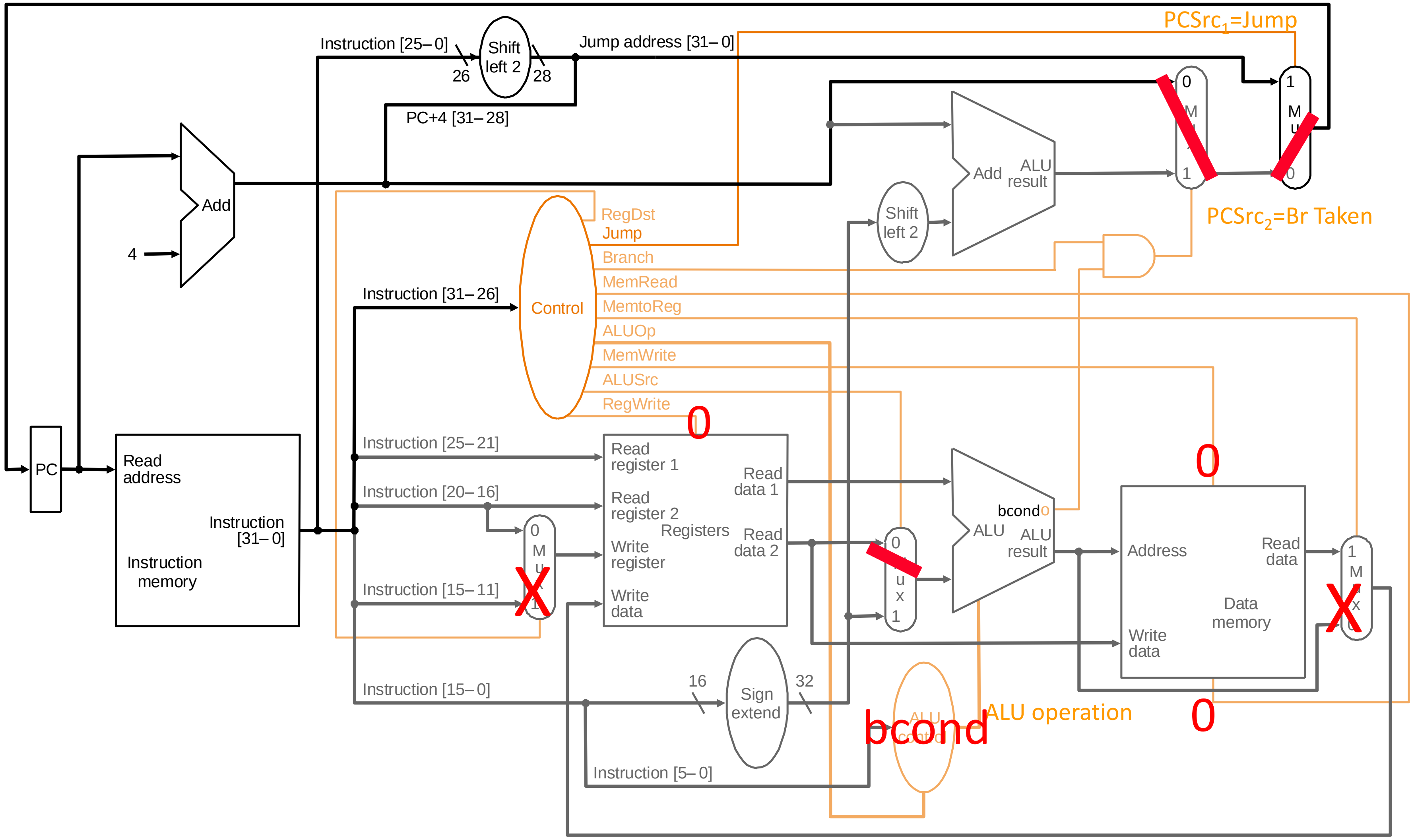
PCSrc$_1$=Jump
Instruction [25–0]
26
Shift left 2
28
Jump address [31–0]
PC+4 [31–28]
Add
4
PC
Read address
Instruction [31–0]
Instruction memory
Add
ALU result
Shift left 2
PCSrc$_2$=Br Taken
M u x
0
1
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Instruction [5–0]
1
Read register 1
Read register 2
Write register
Write data
Read data 1
Read data 2
Registers
bcond
ALU
ALU result
0
Address
Read data
Write data
Data memory
16
Sign extend
32
ALU control
Add
ALU operation
1

]

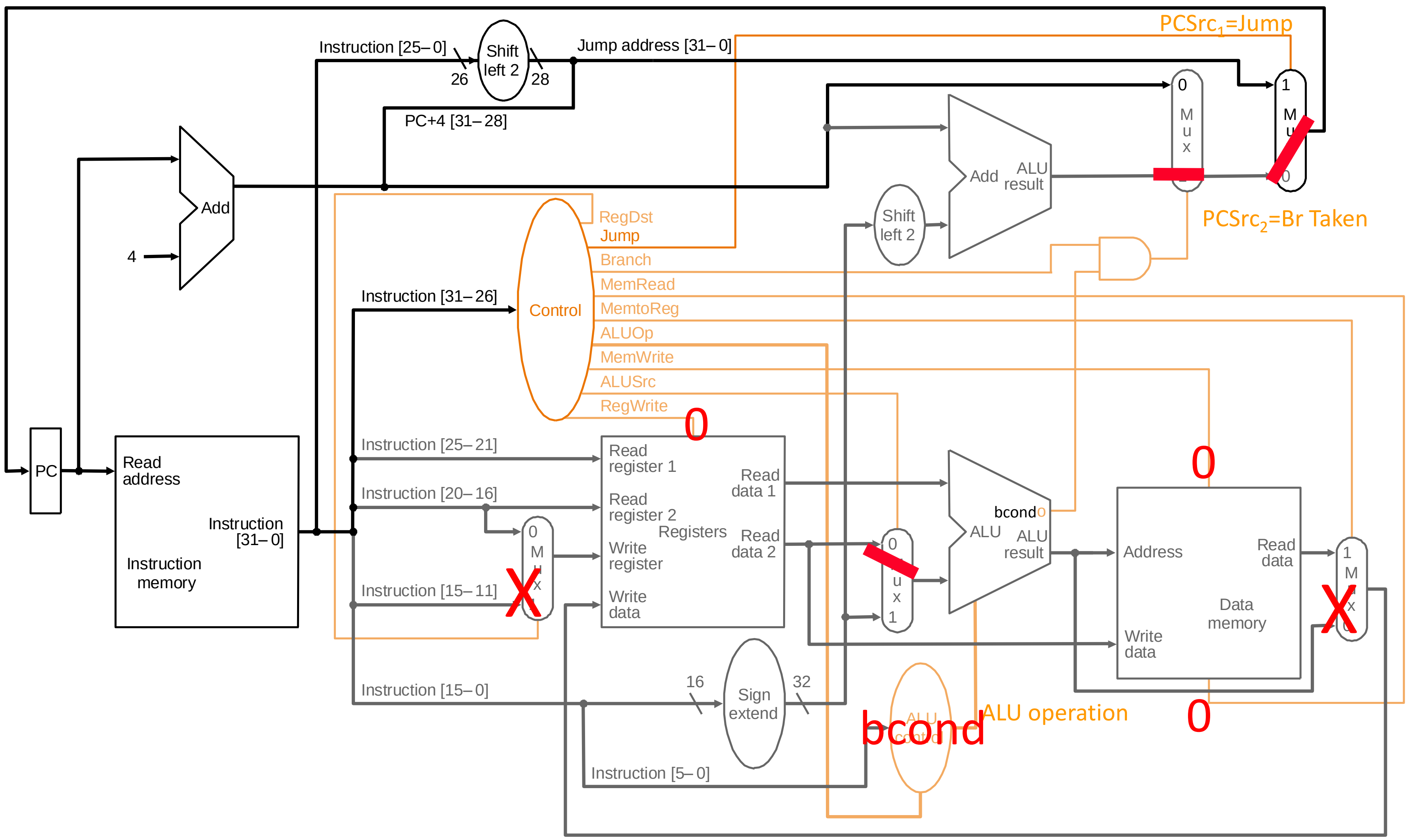
SW


PCSrc$_1$=Jump
Instruction [25–0]
26
Shift left 2
28
Jump address [31–0]
PC+4 [31–28]
Add
4
Add ALU result
0 M u x 1
1 M u x 0
PCSrc$_2$=Br Taken
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
0 M u x 1
X
0
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
0 M u x 1
bcond
ALU
ALU result
1
Address
Read data
Write data
Data memory
1 M u x 0
X
Instruction [15–0]
16
Sign extend
32
Instruction [5–0]
ALU control
Add
ALU operation
0

Branch (Not Taken)

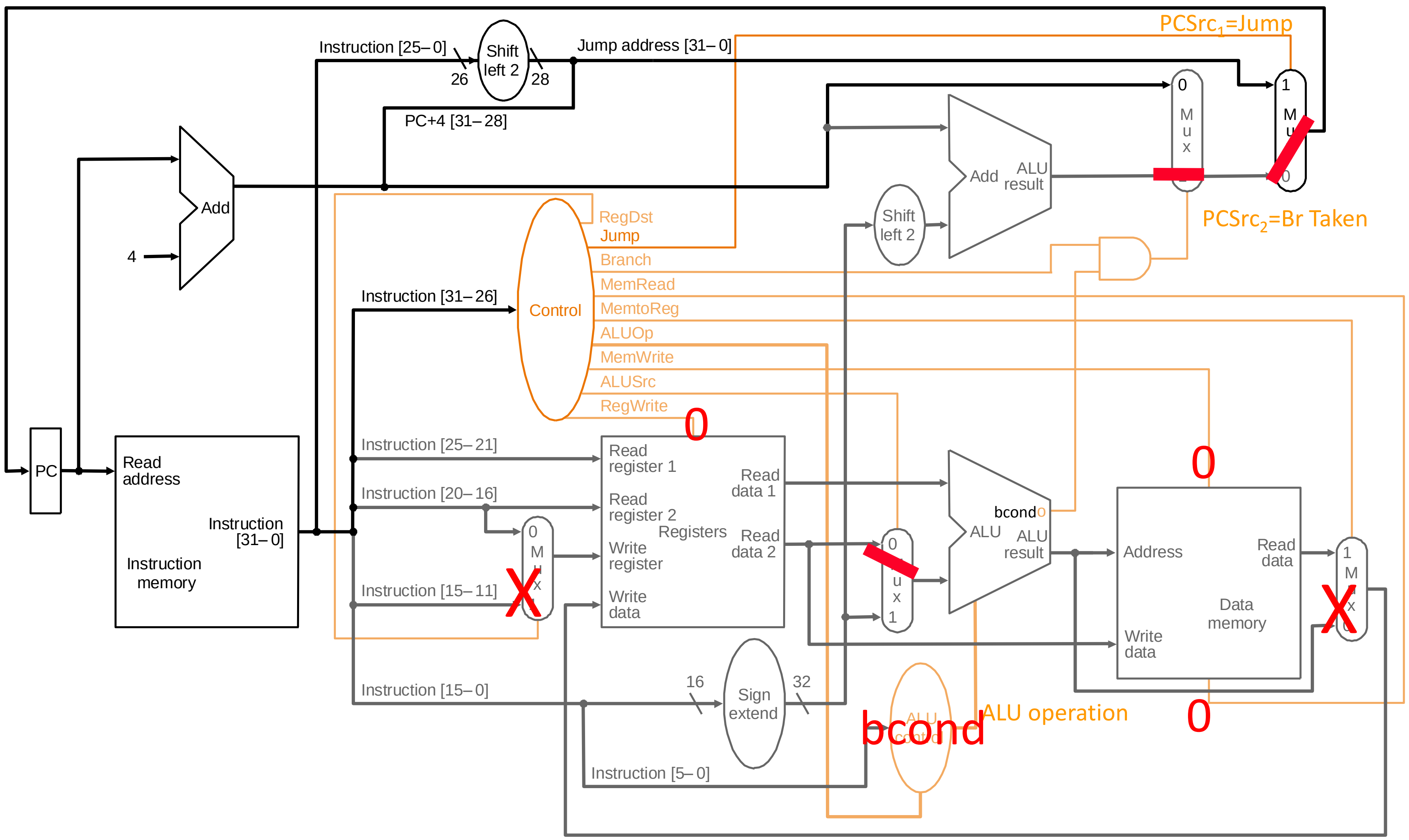


Branch (Taken)

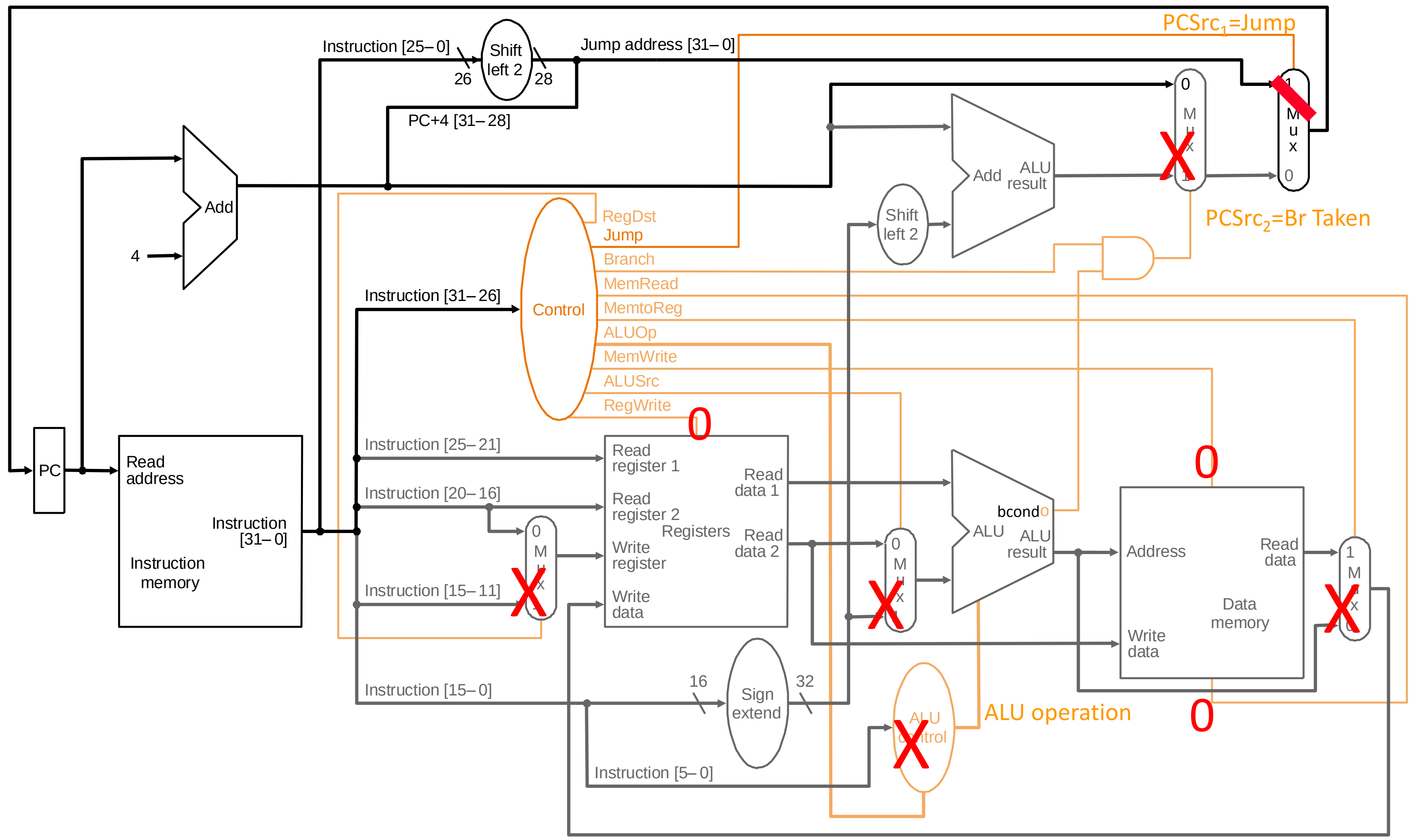


Branch (Taken)

Some control signals are dependent on the processing of data



Jump



Instruction [25–0]
26
Shift left 2
28
Jump address [31–0]
PC+4 [31–28]
PCSrc$_1$=Jump
PCSrc$_2$=Br Taken
Add
4
Add ALU result
Shift left 2
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
PC
Read address
Instruction [31–0]
Instruction memory
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Instruction [15–0]
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
bcond
ALU
ALU result
Address
Read data
Write data
Data memory
16
Sign extend
32
Instruction [5–0]
ALU control
ALU operation
0
0
0
M u x

What is in That Control Box?

- Combinational Logic → Hardwired Control
  - Idea: Control signals generated combinationally based on instruction
  - Necessary in a single-cycle microarchitecture…

- Sequential Logic → Sequential/Microprogrammed Control

What is in That Control Box?

- Combinational Logic → Hardwired Control
  - Idea: Control signals generated combinationally based on instruction
  - Necessary in a single-cycle microarchitecture…
- Sequential Logic → Sequential/Microprogrammed Control
  - Idea: A memory structure contains the control signals associated with an instruction
  - Control Store

MIPS单周期微架构的评测

A Single-Cycle Microarchitecture: Analysis

- Every instruction takes 1 cycle to execute
  - CPI (Cycles per instruction) is strictly 1

A Single-Cycle Microarchitecture: Analysis

- Every instruction takes 1 cycle to execute
  - CPI (Cycles per instruction) is strictly 1
- How long each instruction takes is determined by how long the slowest instruction takes to execute
  - Even though many instructions do not need that long to execute

A Single-Cycle Microarchitecture: Analysis

- Every instruction takes 1 cycle to execute
  - CPI (Cycles per instruction) is strictly 1
- How long each instruction takes is determined by how long the slowest instruction takes to execute
  - Even though many instructions do not need that long to execute
- Clock cycle time of the microarchitecture is determined by how long it takes to complete the slowest instruction
  - Critical path of the design is determined by the processing time of the slowest instruction

What is the Slowest Instruction to Process?

- Let’s go back to the basics
- All five phases of the instruction processing cycle take a single machine clock cycle to complete
  - Instruction fetch (IF)
  - Instruction decode and register operand fetch (ID/RF)
  - Execute/Evaluate memory address (EX/AG)
  - Memory operand fetch (MEM)
  - Store/writeback result (WB)

What is the Slowest Instruction to Process?

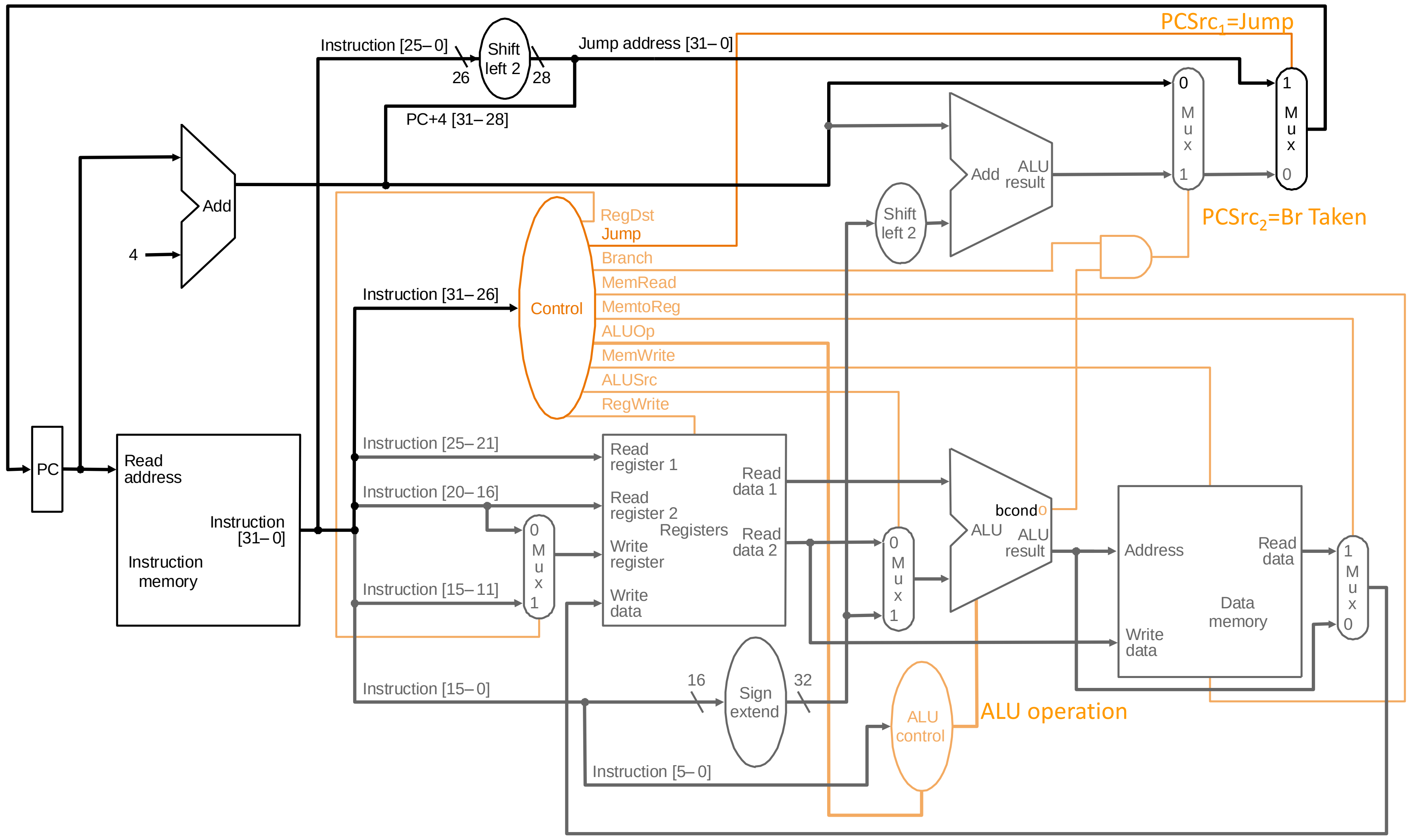
- Let's go back to the basics
- All five phases of the instruction processing cycle take a single machine clock cycle to complete
  - Instruction fetch (IF)
  - Instruction decode and register operand fetch (ID/RF)
  - Execute/Evaluate memory address (EX/AG)
  - Memory operand fetch (MEM)
  - Store/writeback result (WB)
- Do each of the above phases take the same time (latency) for all instructions?

单周期数据通路分析

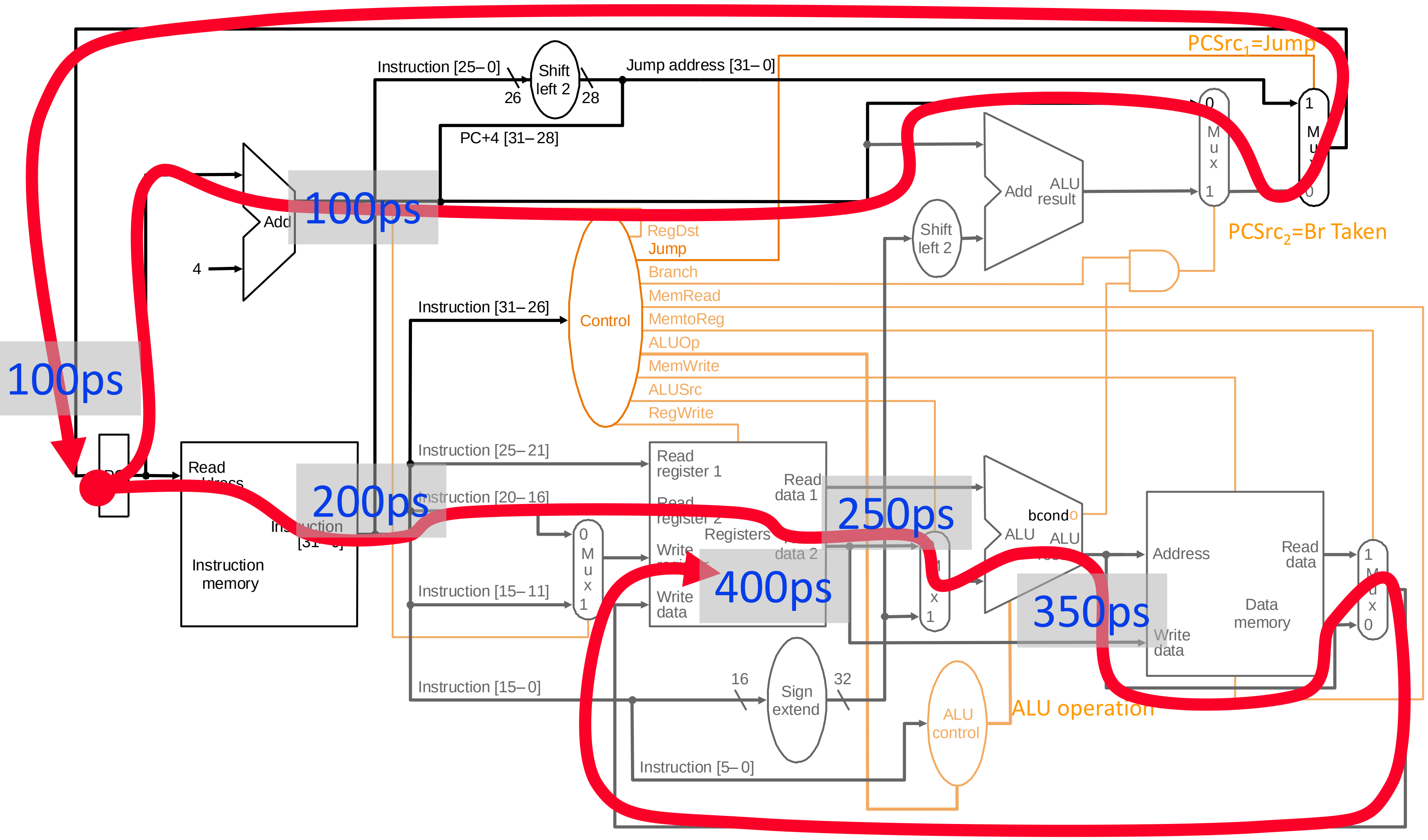
- Assume
  - memory units (read or write): 200 ps
  - ALU and adders: 100 ps
  - register file (read or write): 50 ps
  - other combinational logic: 0 ps

steps	IF	ID	EX	MEM	WB	Delay
resources	mem	RF	ALU	mem	RF	
R-type	200	50	100		50	400
I-type	200	50	100		50	400
LW	200	50	100	200	50	600
SW	200	50	100	200		550
Branch	200	50	100			350
Jump	200					200

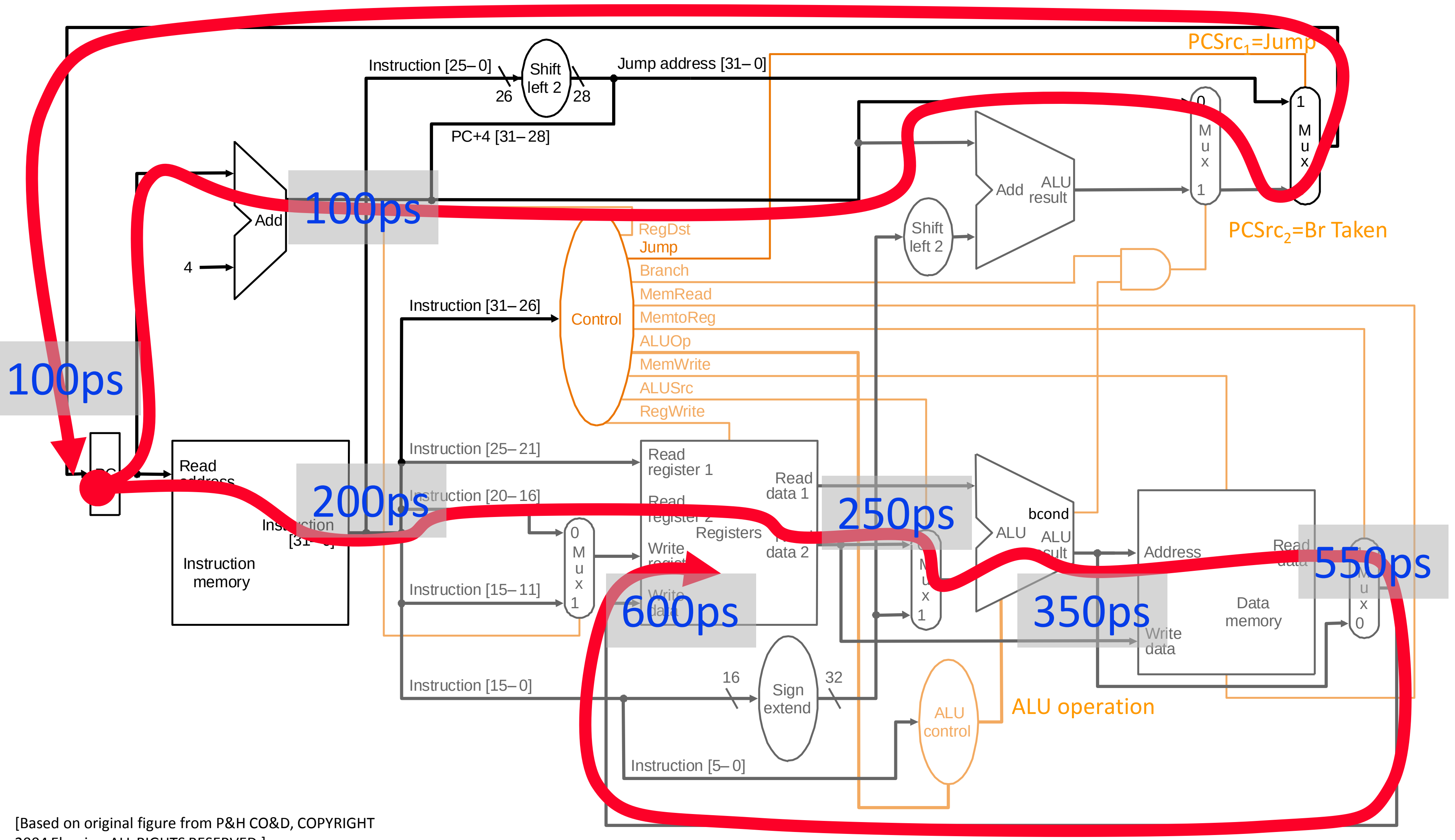
找出关键路径


PCSrc$_1$=Jump
Instruction [25– 0]
26
Shift left 2
28
Jump address [31– 0]
PC+4 [31– 28]
Add
4
Add
ALU result
Shift left 2
PCSrc$_2$=Br Taken
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Instruction [31– 26]
Control
PC
Read address
Instruction [31– 0]
Instruction memory
Instruction [25– 21]
Instruction [20– 16]
Instruction [15– 11]
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
bcond
ALU
ALU result
Address
Read data
Data memory
Write data
M u x
0
1
16
Sign extend
32
Instruction [15– 0]
ALU control
ALU operation
Instruction [5– 0]

R-Type and I-Type ALU

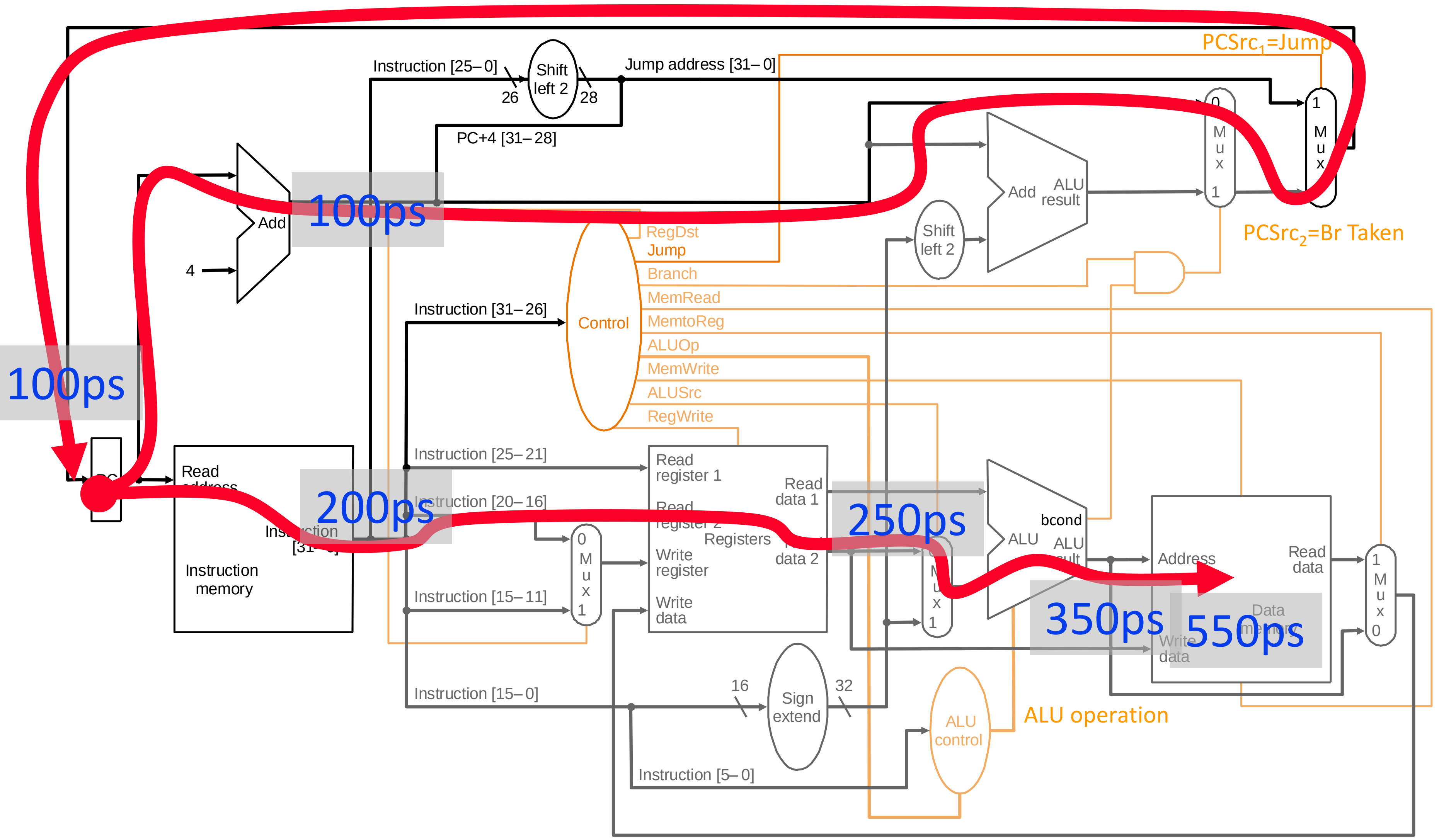


LW



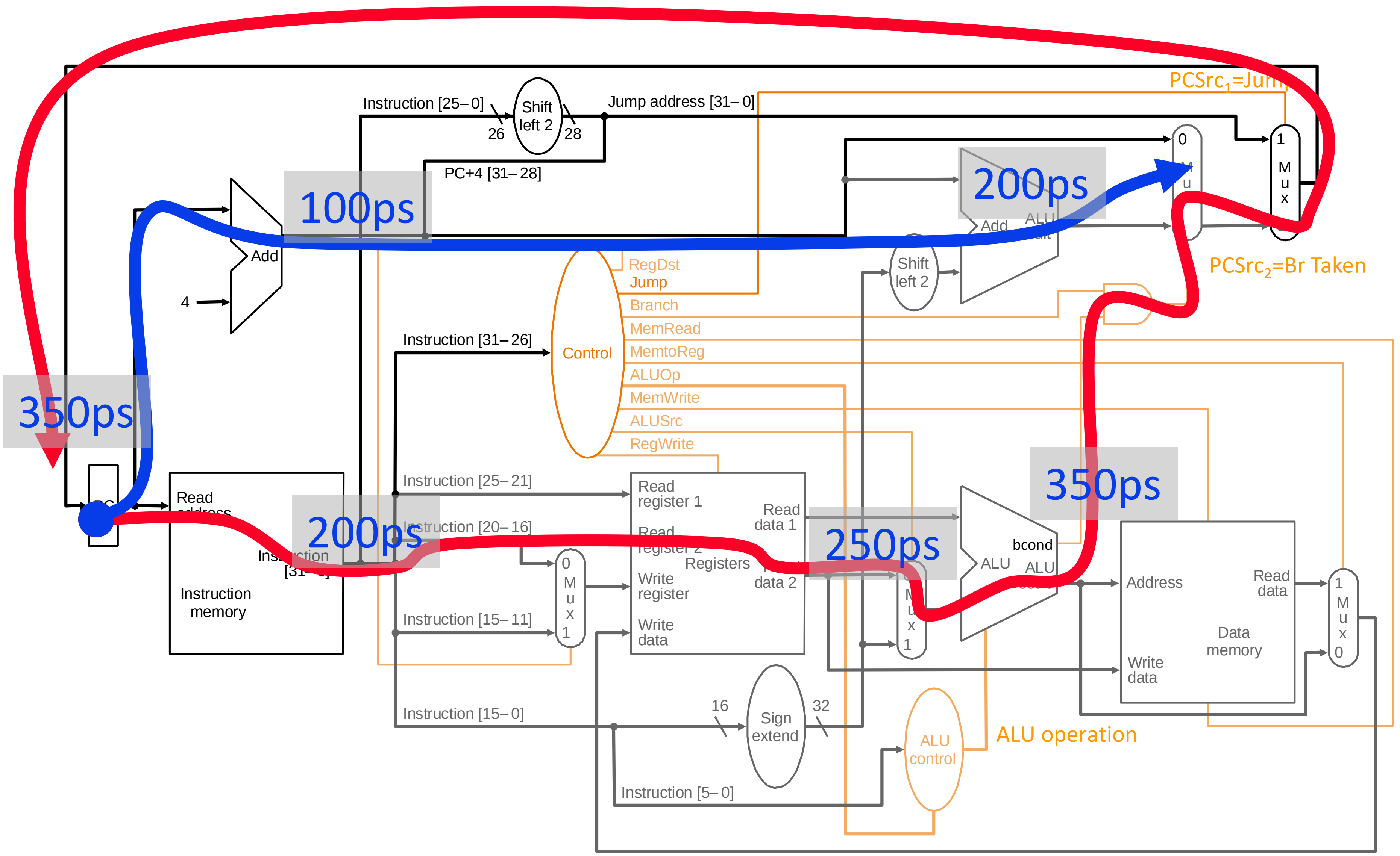
[Based on original figure from P&H CO&D, COPYRIGHT 2004 Elsevier. ALL RIGHTS RESERVED.]

SW

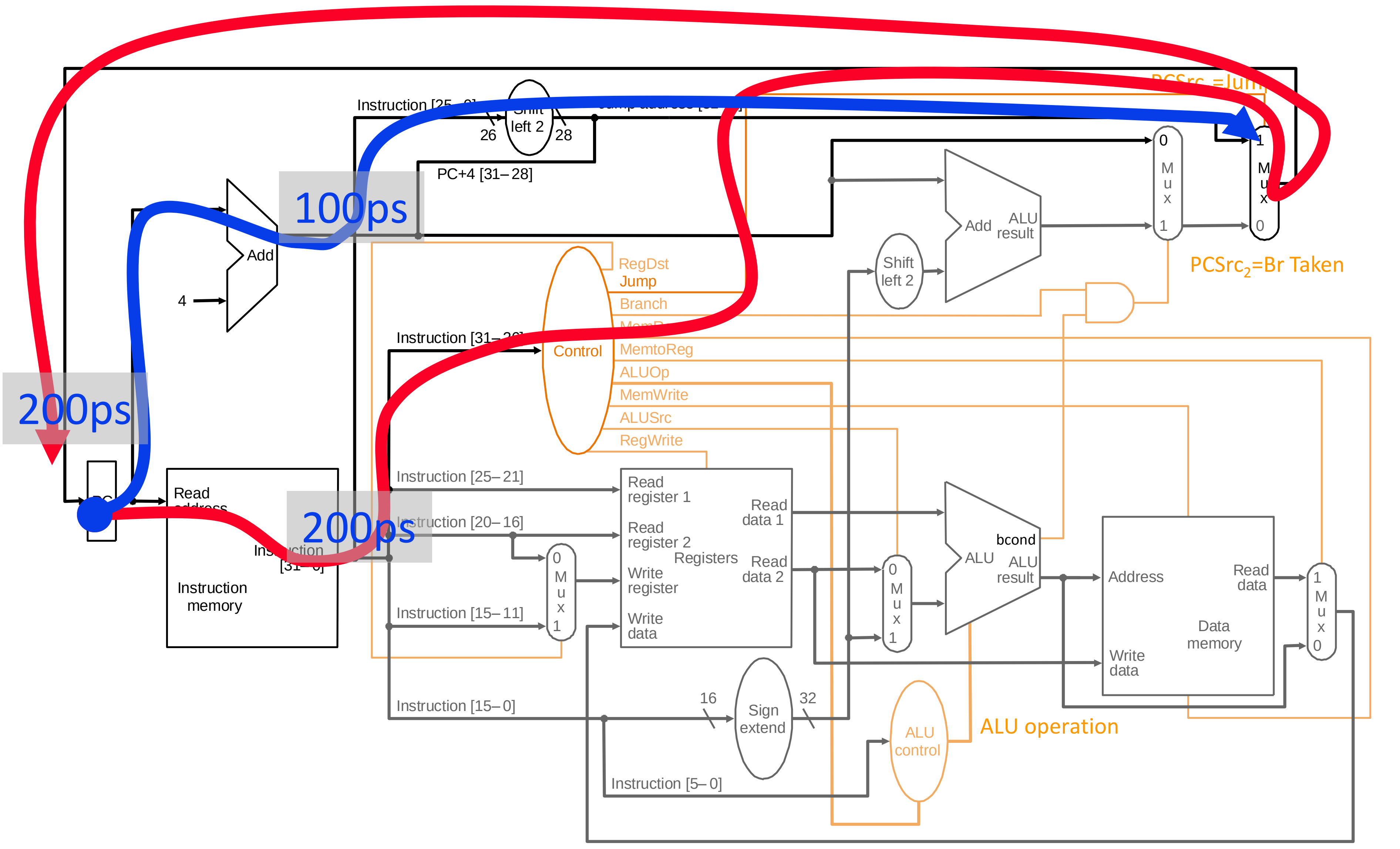


PCSrc1=Jump
Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC+4 [31–28]
1
M u x
0
M u x
1
Add
ALU result
100ps
Add
4
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
Shift left 2
PCSrc2=Br Taken
100ps
PC
Read address
Instruction [31–0]
Instruction memory
200ps
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
0 M u x 1
Read register 1
Read register 2
Write register
Write data
Registers
Read data 1
Read data 2
250ps
M u x 1
ALU
bcond
ALU result
350ps
Address
Read data
Data memory
Write data
550ps
1 M u x 0
Instruction [15–0]
16
Sign extend
32
ALU control
ALU operation
Instruction [5–0]

Branch Taken



Jump



PCSrc1=Jump
Instruction [25–0]
Shift left 2
26
28
Jump address [31–0]
PC+4 [31–28]
100ps
Add
4
PCSrc2=Br Taken
RegDst
Jump
Branch
MemRead
MemtoReg
ALUOp
MemWrite
ALUSrc
RegWrite
Control
Instruction [31–26]
200ps
PC
Read address
Instruction [31–0]
Instruction memory
200ps
Instruction [25–21]
Instruction [20–16]
Instruction [15–11]
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
0 M u x 1
Shift left 2
Add
ALU result
bcond
ALU
ALU result
Address
Read data
Data memory
Write data
1 M u x 0
Instruction [15–0]
16
Sign extend
32
Instruction [5–0]
ALU control
ALU operation

Next Topic

Pipelining & Hazards