多周期执行步骤

- 指令执行 3 到 5 个时钟周期

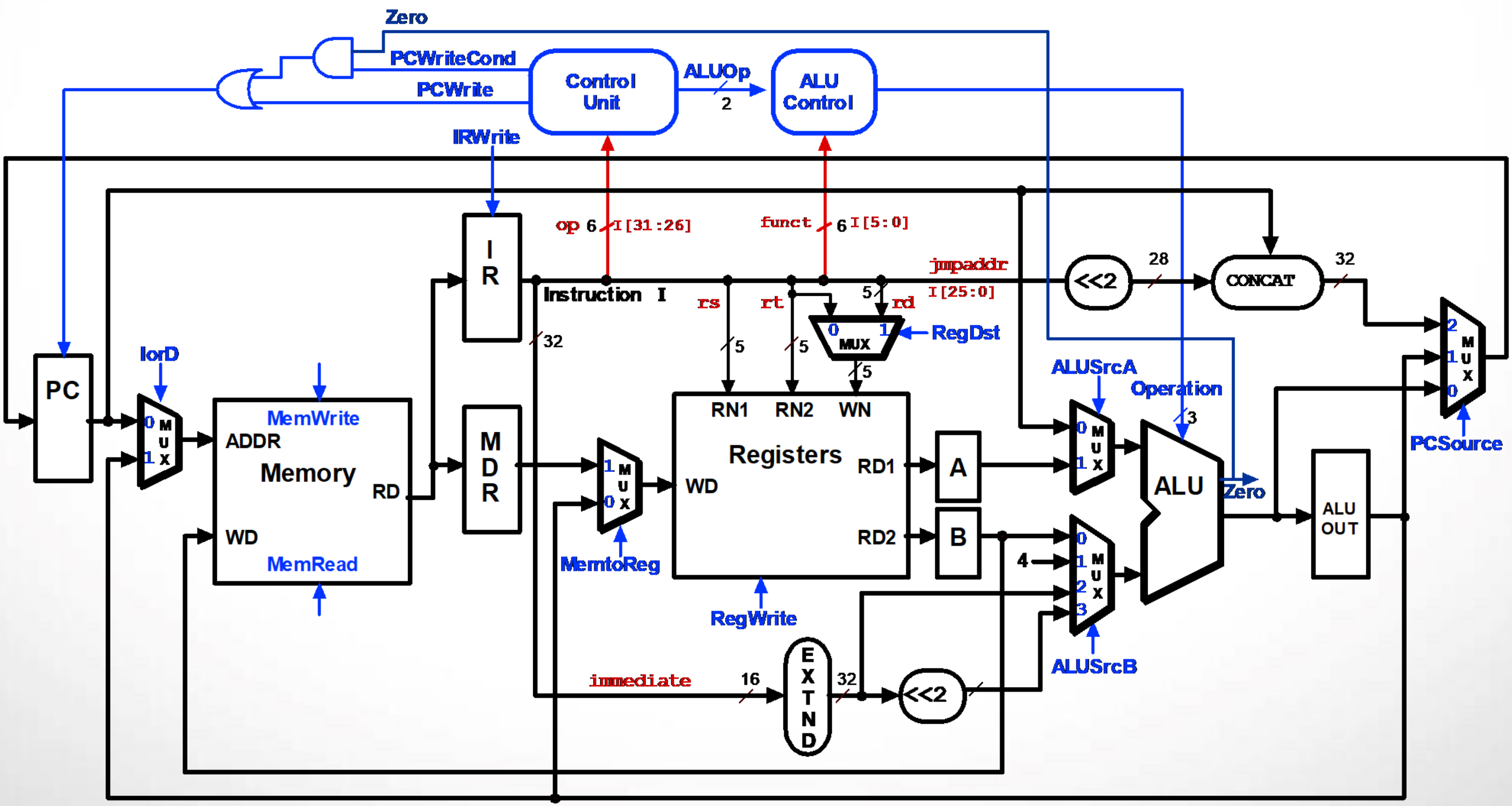
Step name	Action for R-type instructions	Action for memory-reference instructions	Action for branches	Action for jumps

Instruction fetch	IR = Memory[PC]
	PC = PC + 4			

Instruction decode/register fetch	A = Reg [IR[25-21]]
	B = Reg [IR[20-16]]
	ALUOut = PC + (sign-extend (IR[15-0]) << 2)			

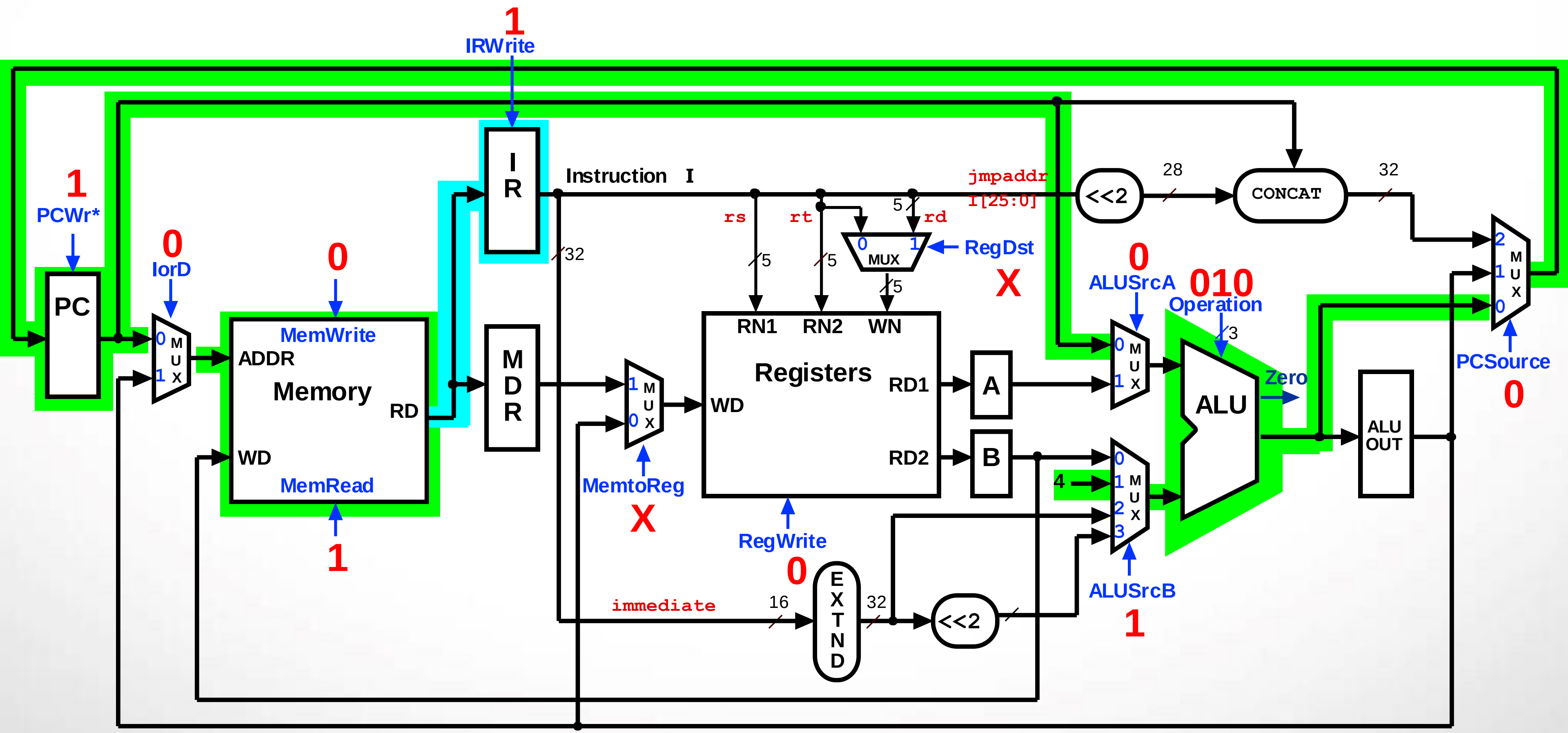
Execution, address computation, branch/ jump completion	ALUOut = A op B	ALUOut = A + sign-extend (IR[15-0])	if (A ==B) then PC = ALUOut	PC = PC [31-28] II (IR[25-0]<<2)
Memory access or R-type completion	Reg [IR[15-11]] = ALUOut	Load: MDR = Memory[ALUOut] or Store: Memory [ALUOut] = B		
Memory read completion		Load: Reg[IR[20-16]] = MDR		

完整数据通路 & 控制


Zero
PCWriteCond
PCWrite
Control Unit
ALUOp
2
ALU Control
IRWrite
op 6 I[31:26]
funct 6 I[5:0]
jmpaddr
I[25:0]
<<2
28
CONCAT
32
I R
Instruction I
32
rs
rt
rd
5
5
5
5
0 MUX 1
RegDst
PC
IorD
0 M U X 1
MemWrite
ADDR
Memory
RD
WD
MemRead
M D R
1 M U X 0
MemtoReg
RN1
RN2
WN
Registers
WD
RD1
RD2
RegWrite
A
B
ALUSrcA
0 M U X 1
Operation
3
ALU
Zero
4
0 1 2 3 M U X
ALUSrcB
immediate
16
E X T N D
32
<<2
ALU OUT
2 1 0 M U X
PCSource

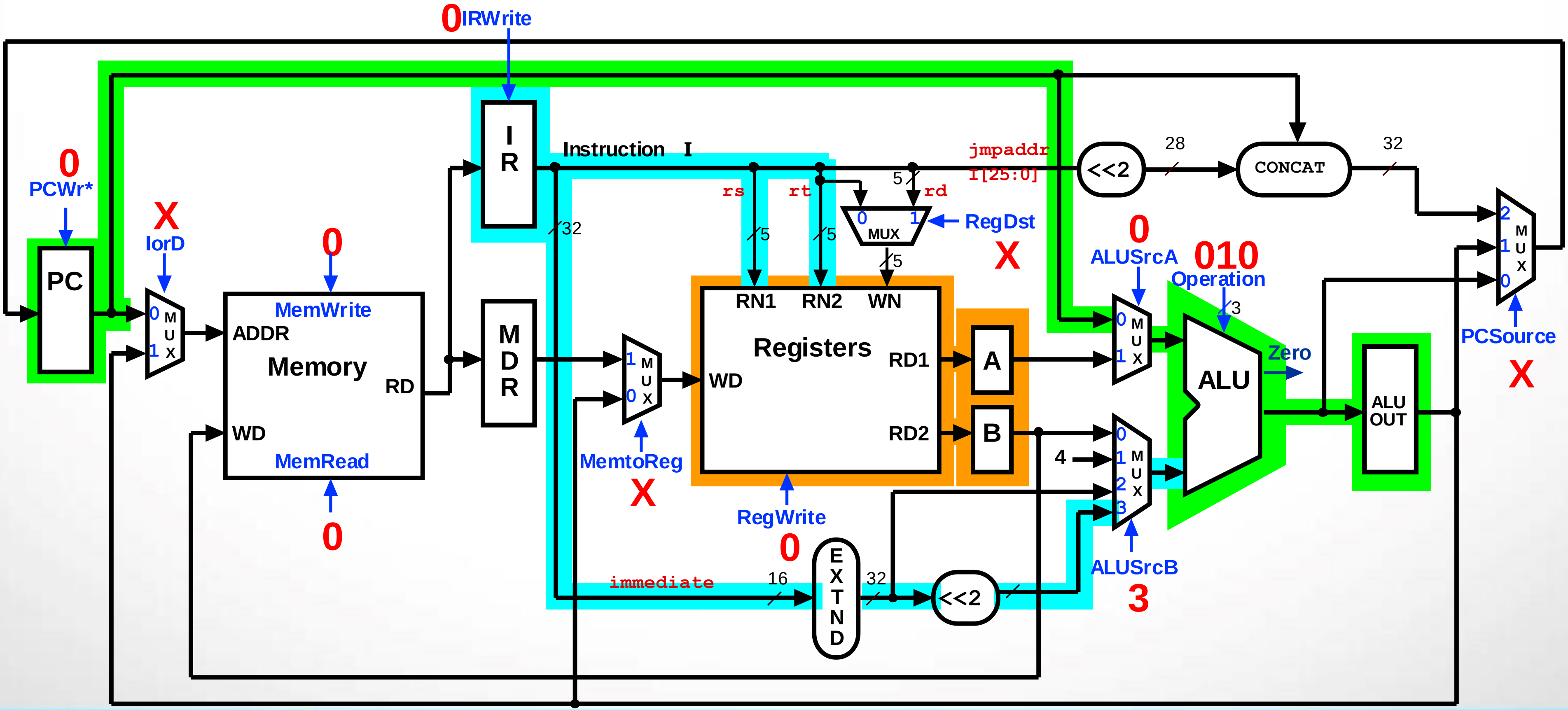
多周期执行步骤 (1)： R-Type

```
IR = Memory[PC];
PC = PC + 4;
```



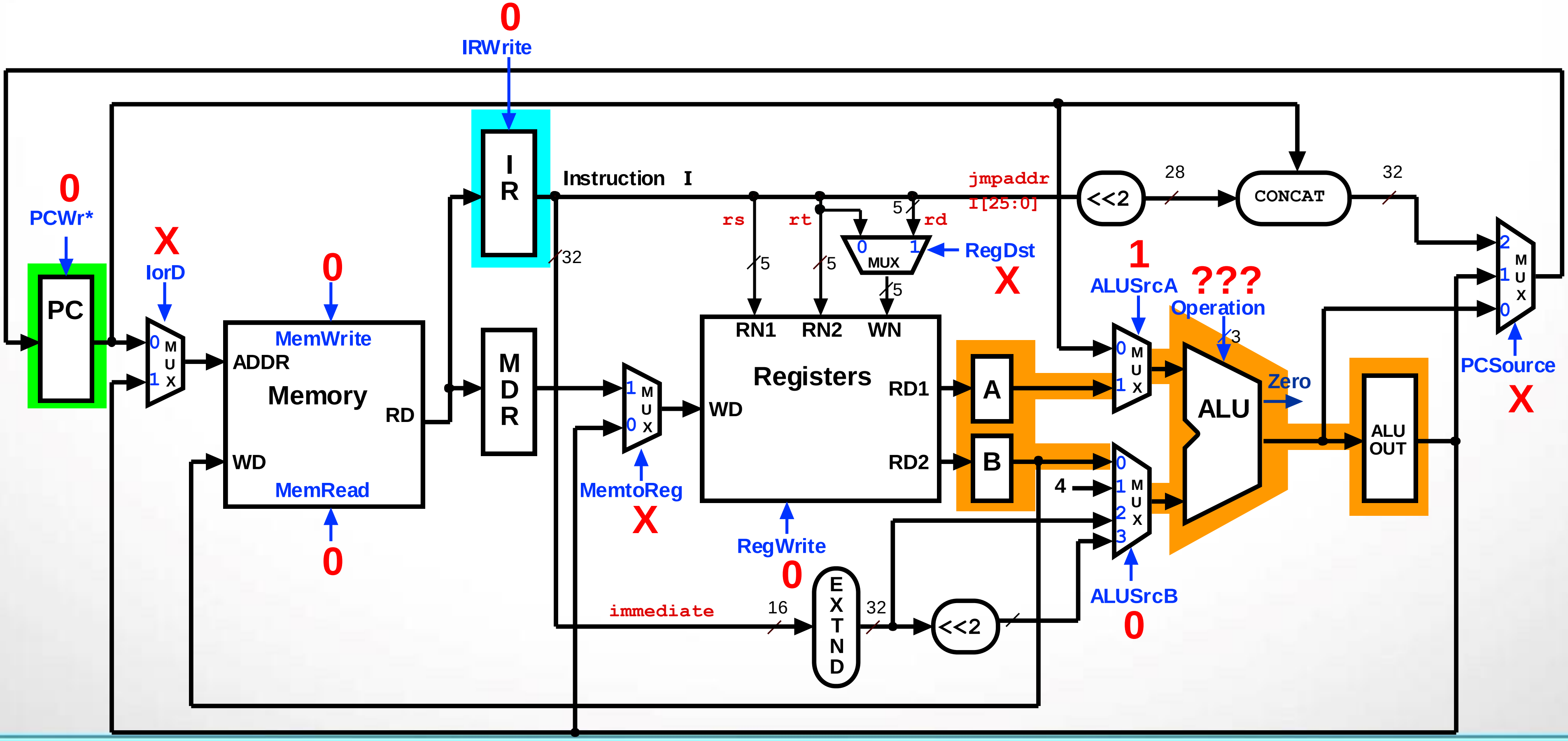
多周期执行步骤 (2)： R-Type

```
A = Reg[IR[25-21]];              (A = Reg[rs])
B = Reg[IR[20-15]];              (B = Reg[rt])
ALUOut = (PC + sign-extend(IR[15-0]) << 2)
```



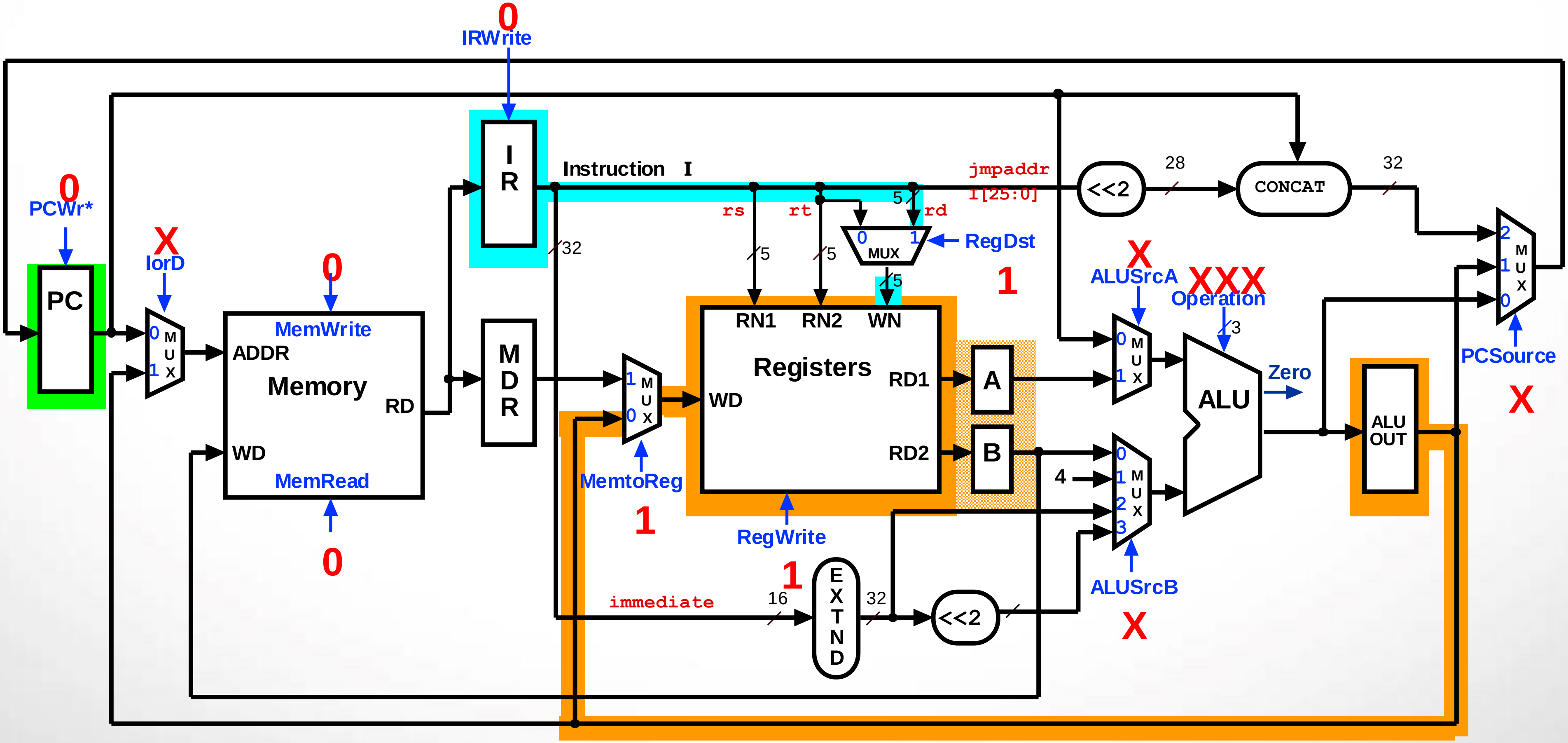
多周期执行步骤 (3)： R-Type

```
ALUOut = A op B
```



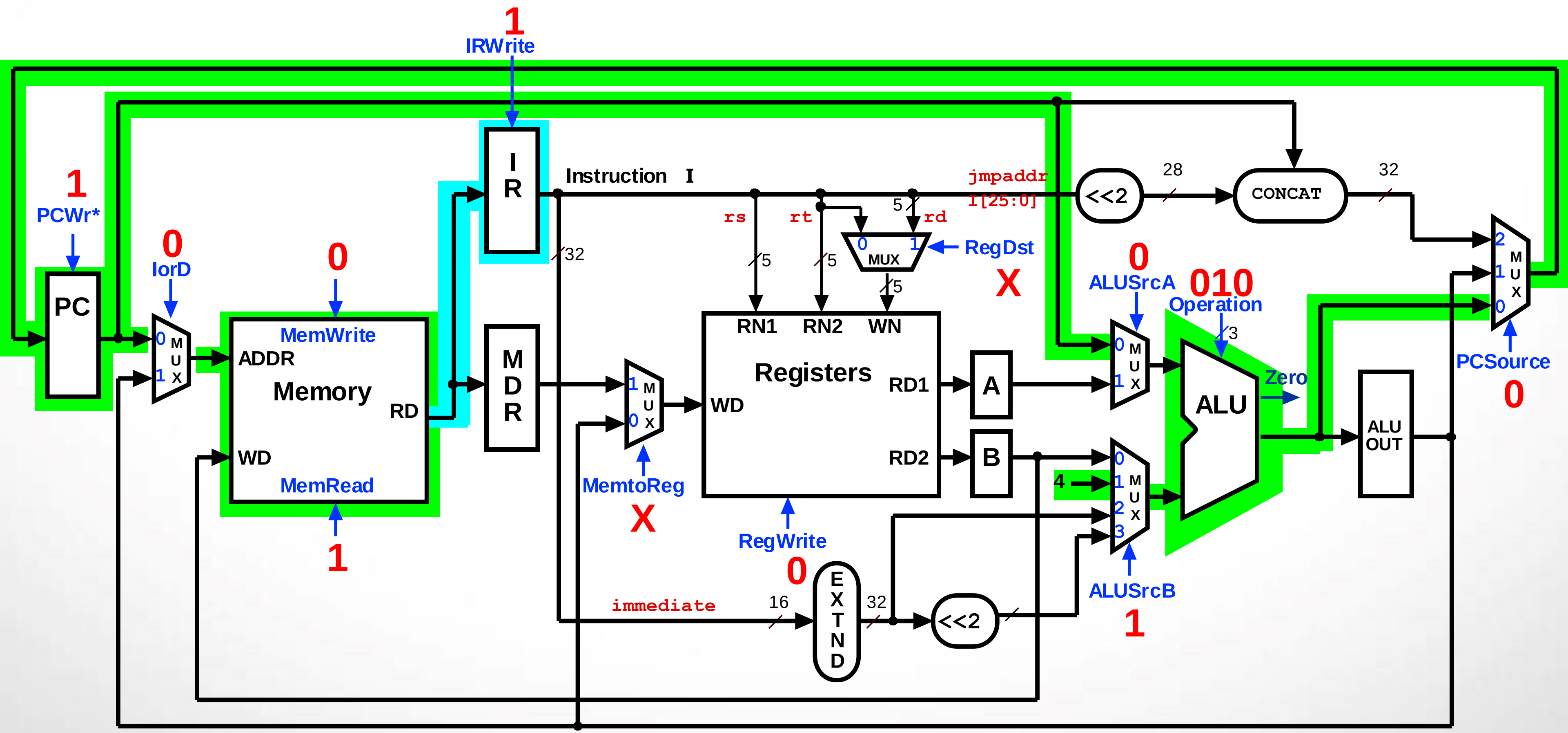
多周期执行步骤 (4)：R-Type

```
Reg[IR[15:11]] = ALUOut;          (Reg[Rd] = ALUOut)
```



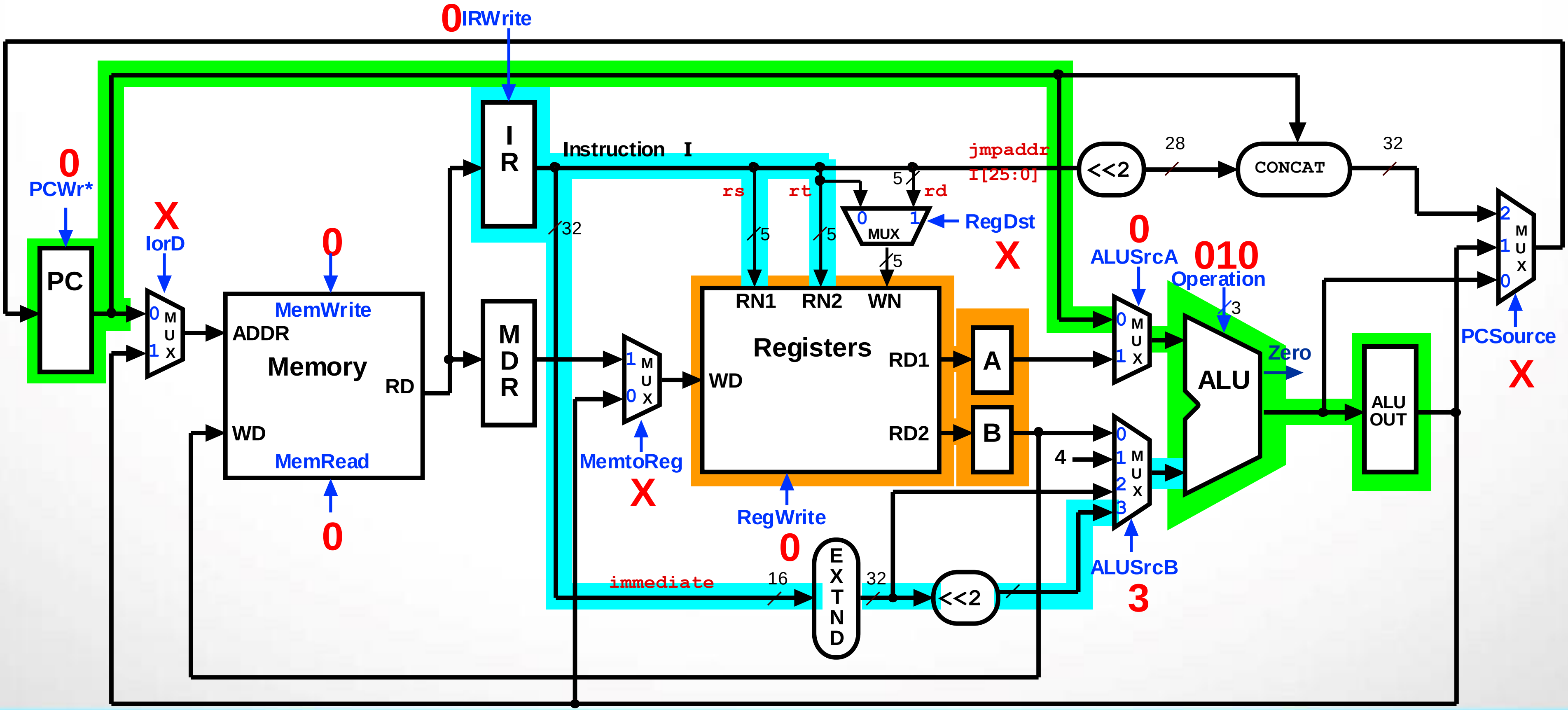
多周期执行步骤 (1):lw

```
IR = Memory[PC];
PC = PC + 4;
```



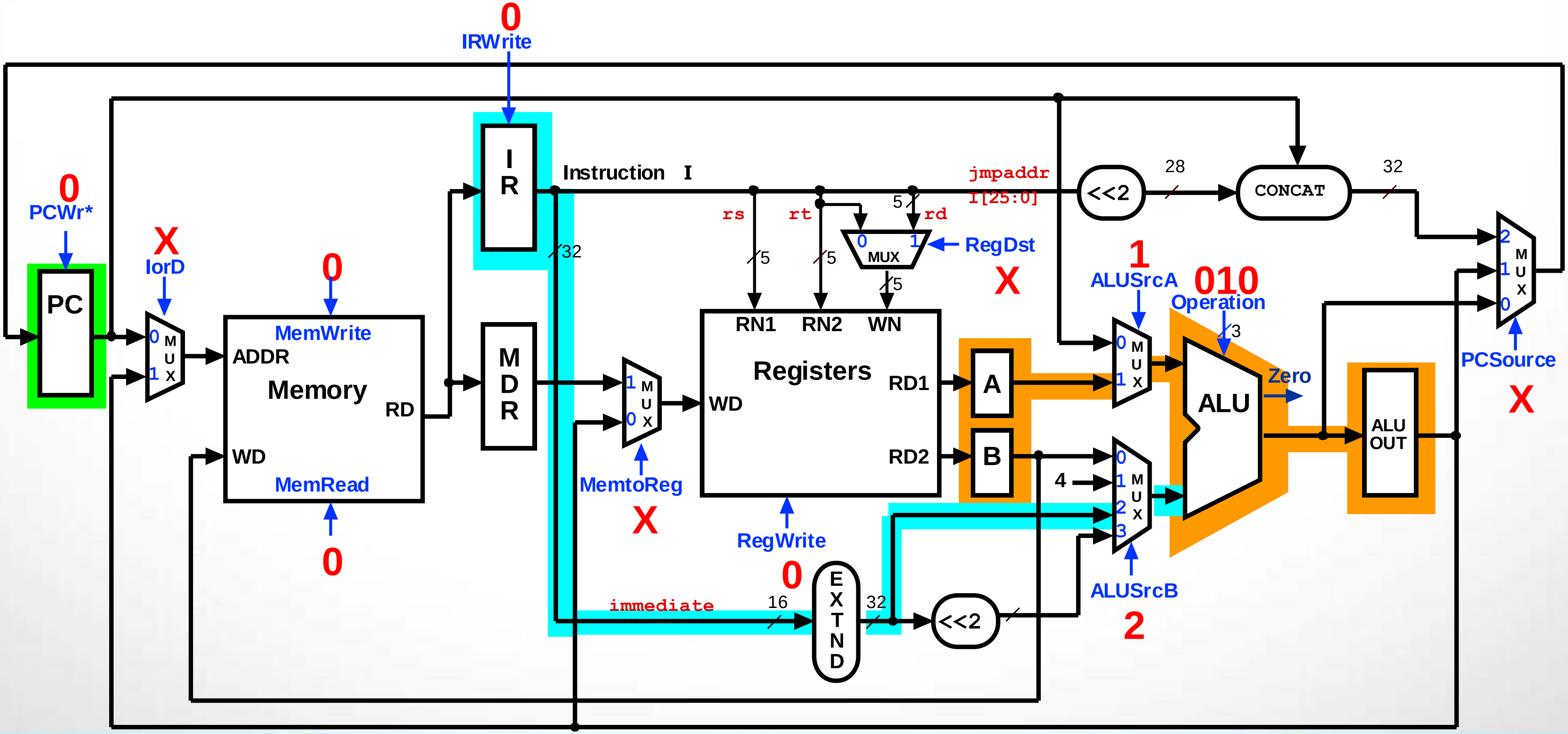
多周期执行步骤 (2):lw

```
A = Reg[IR[25-21]];            (A = Reg[rs])
B = Reg[IR[20-15]];            (B = Reg[rt])
ALUOut = (PC + sign-extend(IR[15-0]) << 2)
```



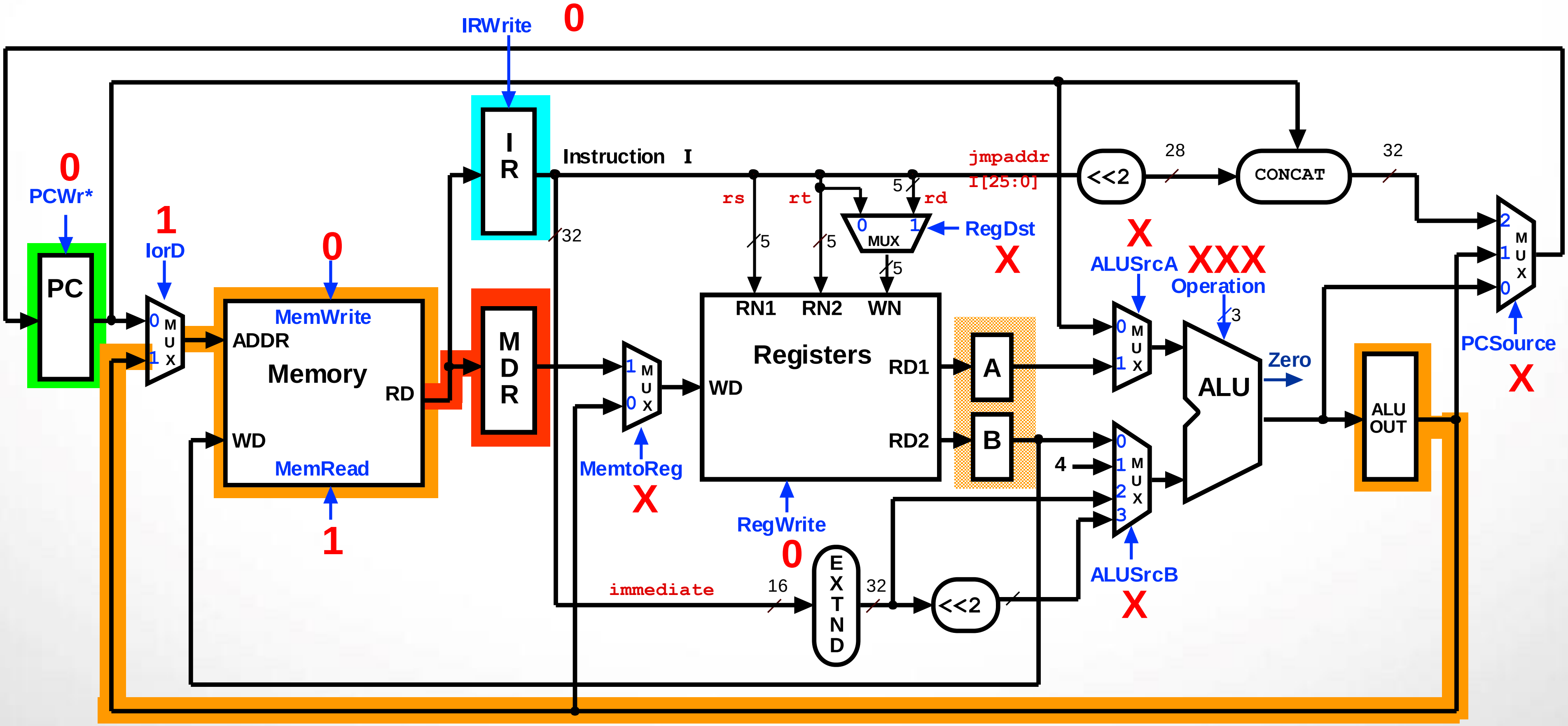
多周期执行步骤 (3):lw

```
ALUOut = A + sign-extend(IR[15-0]);
```



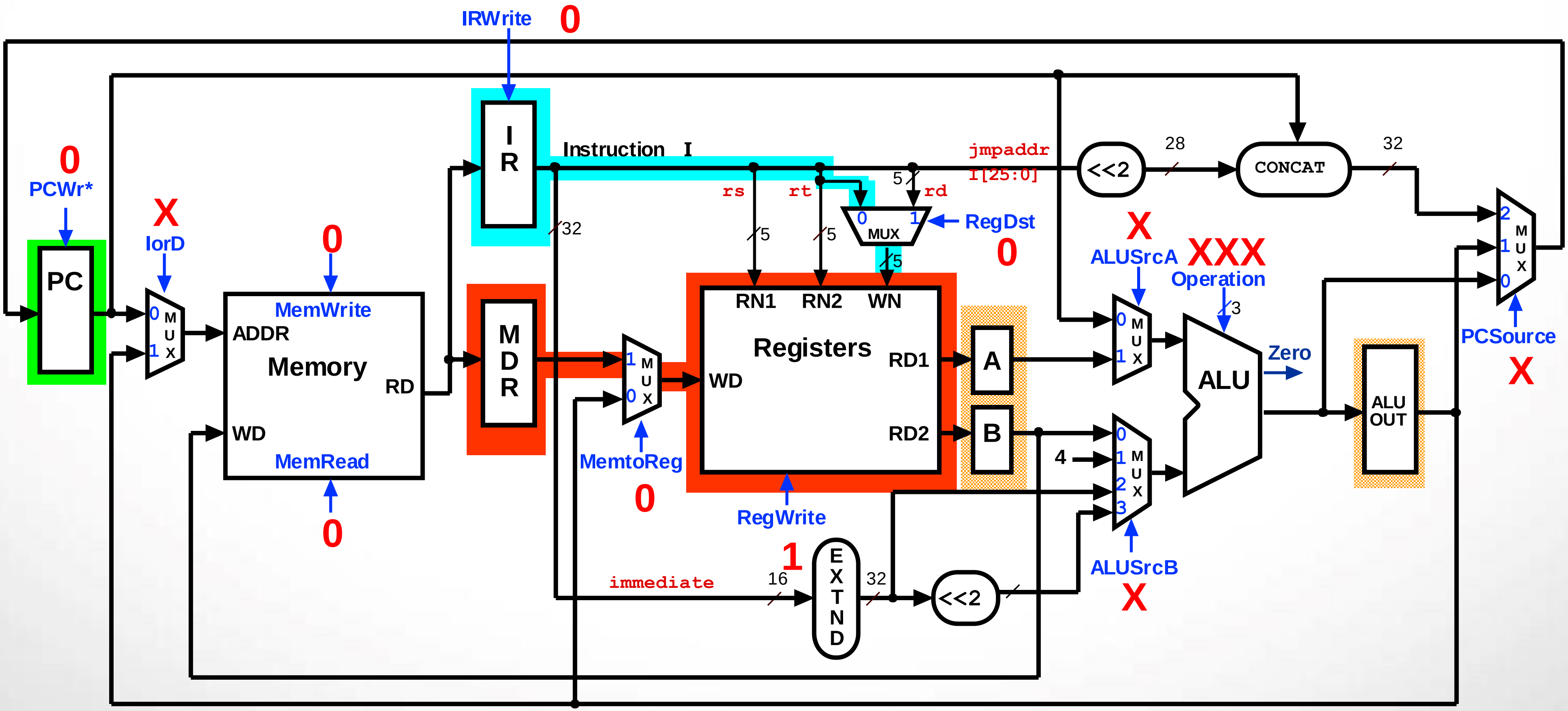
多周期执行步骤 (4)： lw

```
MDR = Memory[ALUOut];
```



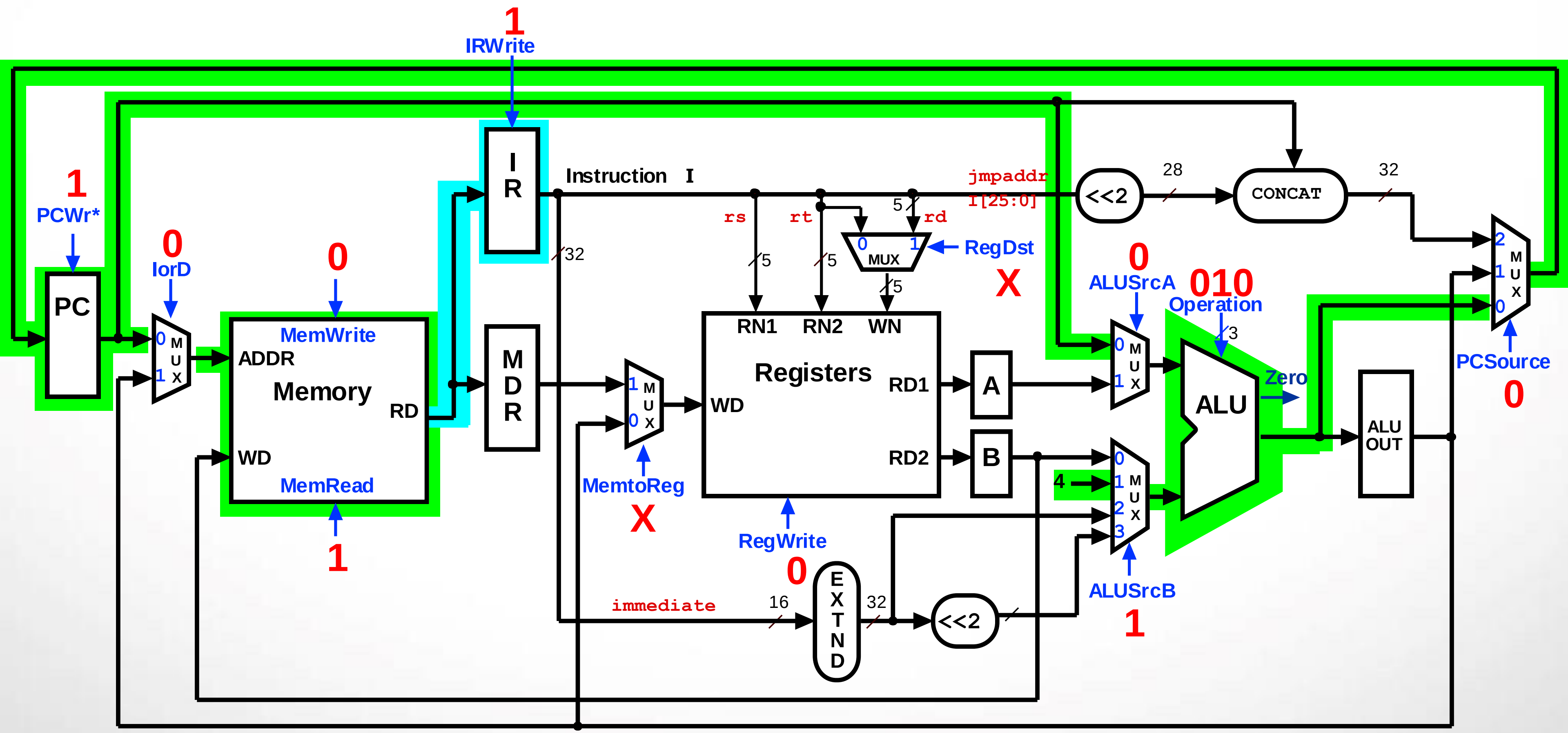
多周期执行步骤 (5)： lw

```
Reg[IR[20-16]] = MDR;
```



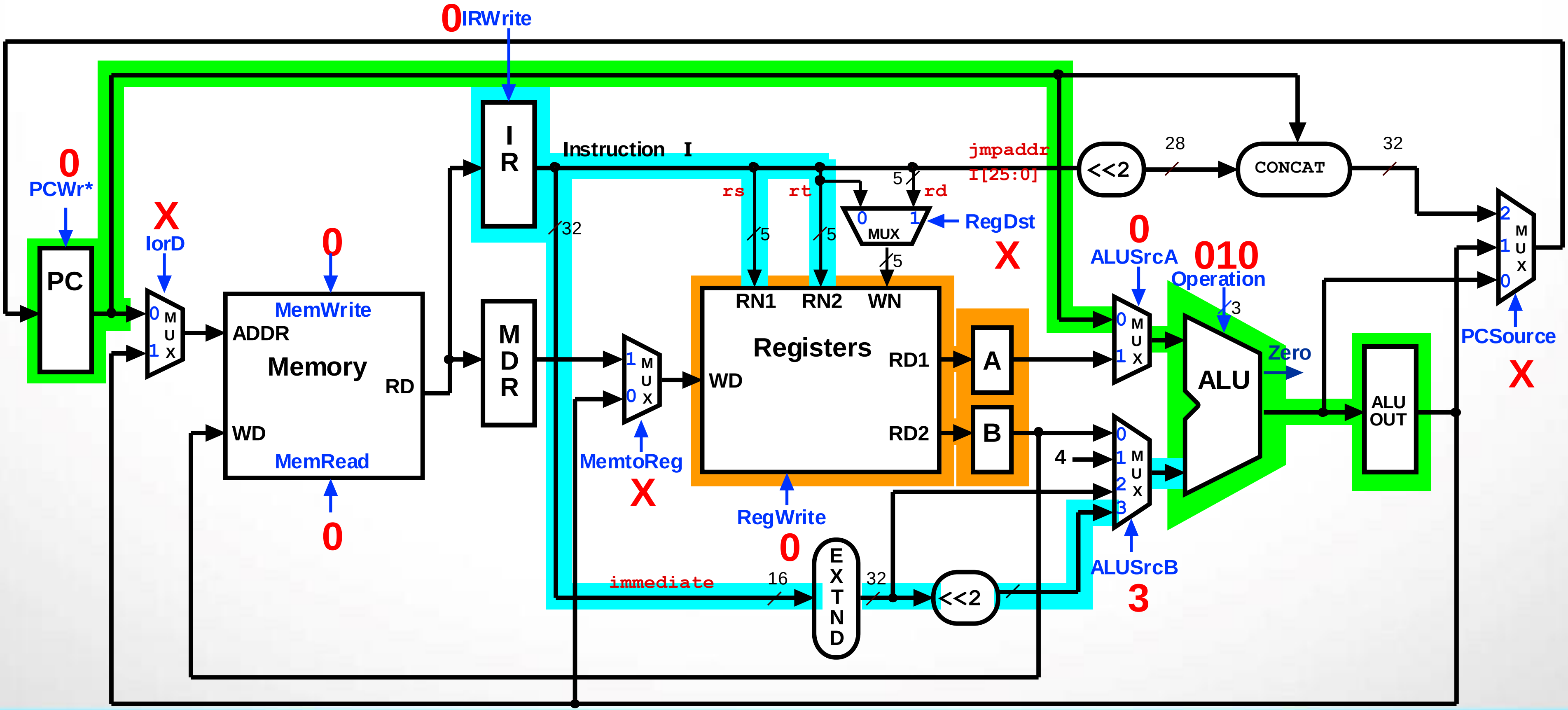
多周期执行步骤 (1):sw

```
IR = Memory[PC];
PC = PC + 4;
```



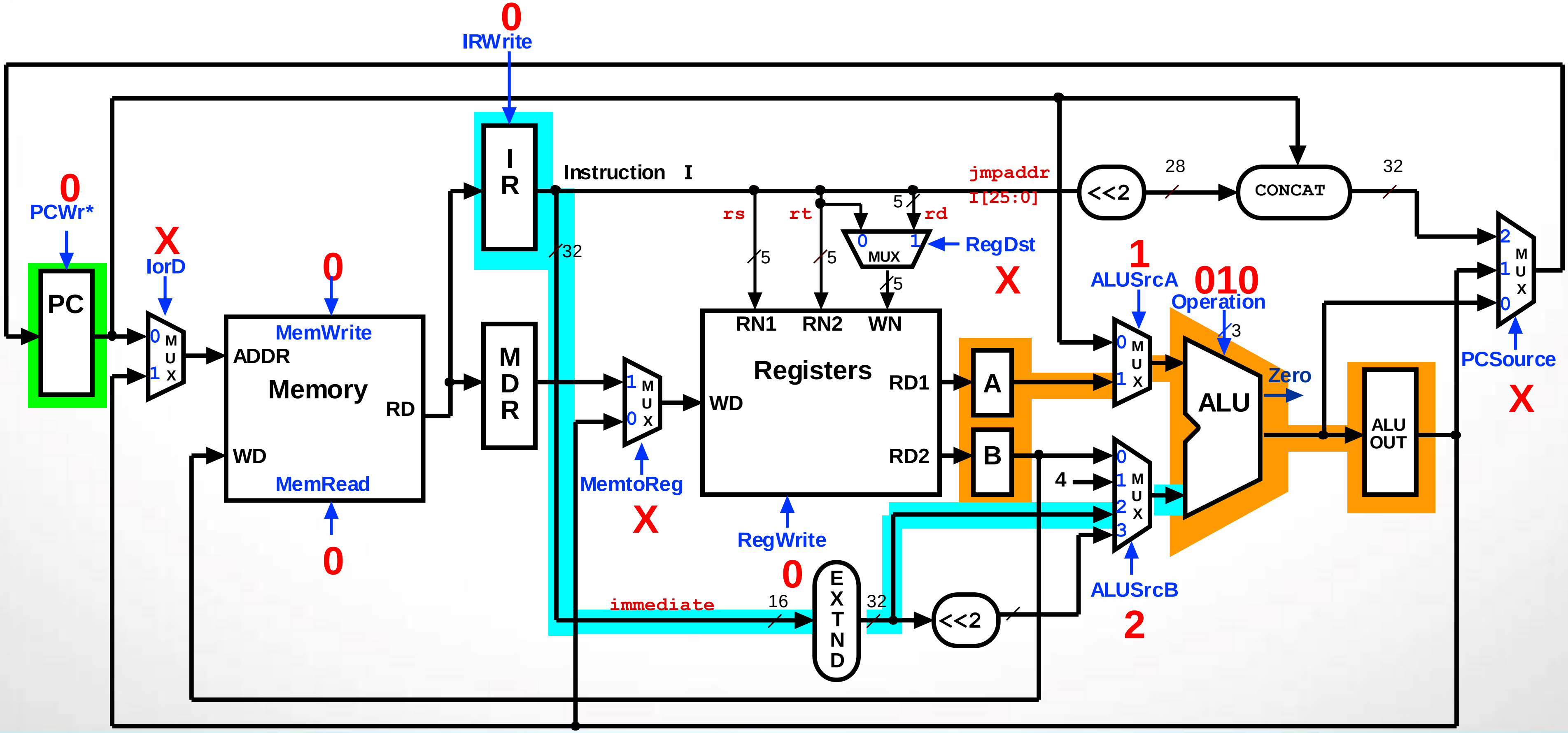
多周期执行步骤 (2):sw

```
A = Reg[IR[25-21]];                (A = Reg[rs])
B = Reg[IR[20-15]];                (B = Reg[rt])
ALUOut = (PC + sign-extend(IR[15-0]) << 2)
```



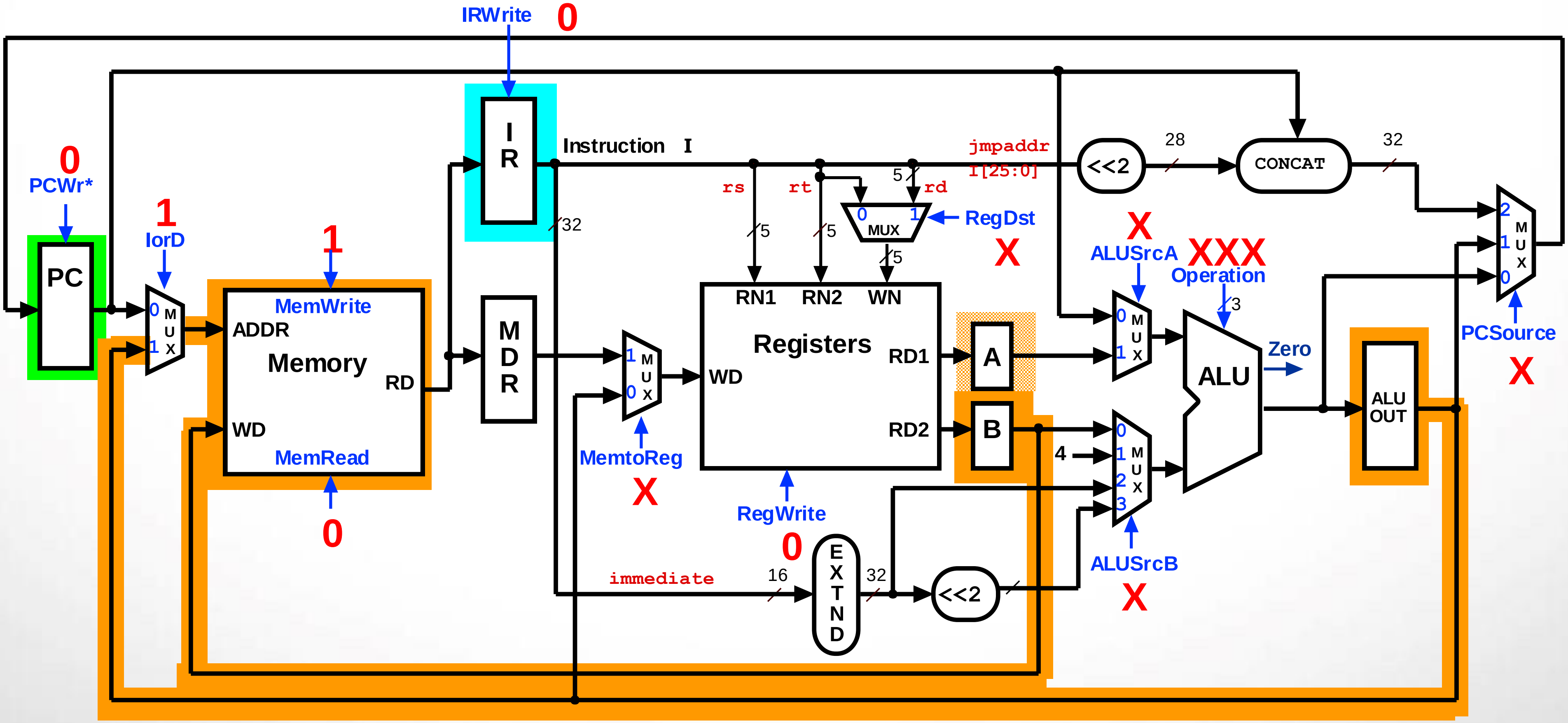
多周期执行步骤 (3):sw

```
ALUOut = A + sign-extend(IR[15-0]);
```



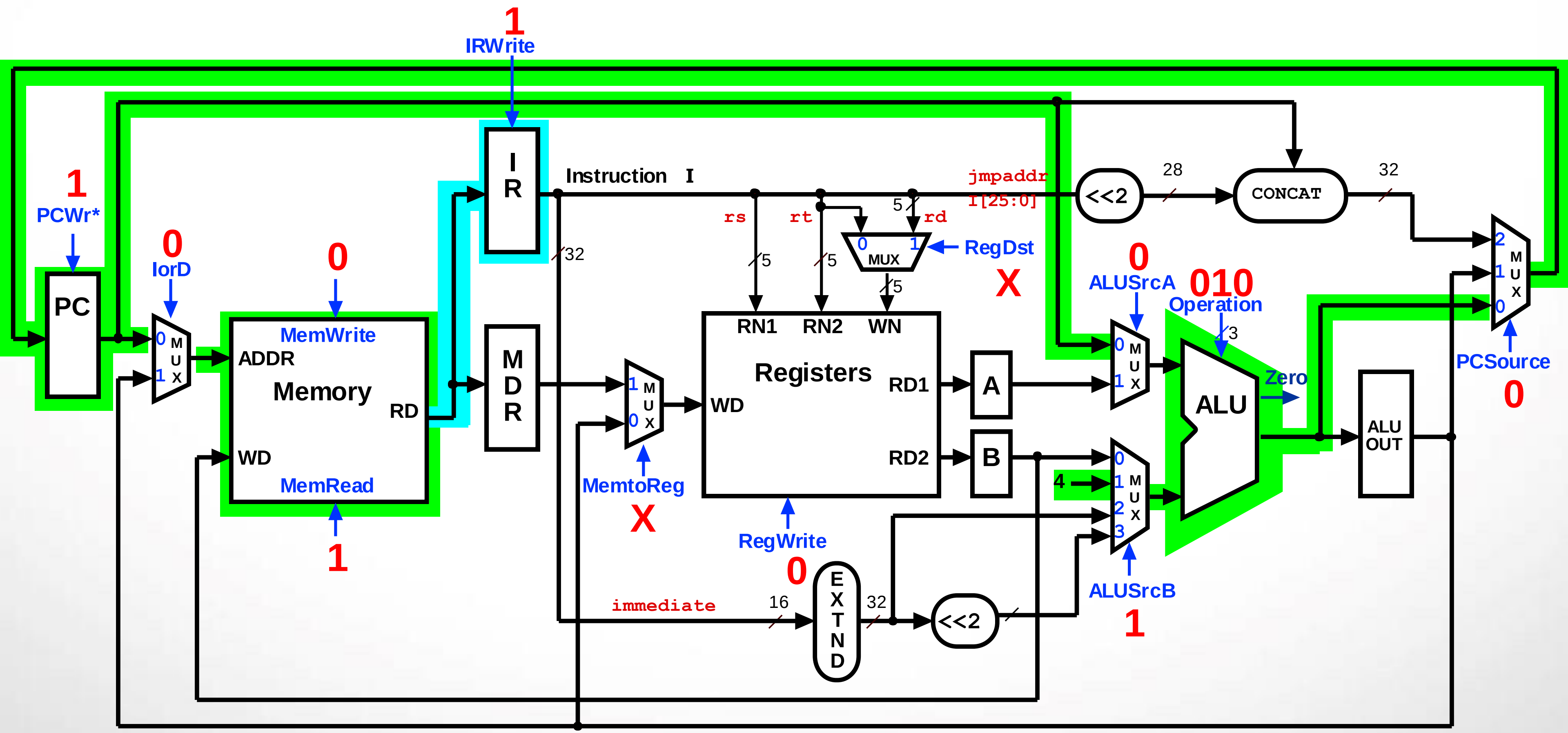
多周期执行步骤 (4)： SW

```
Memory[ALUOut] = B;
```



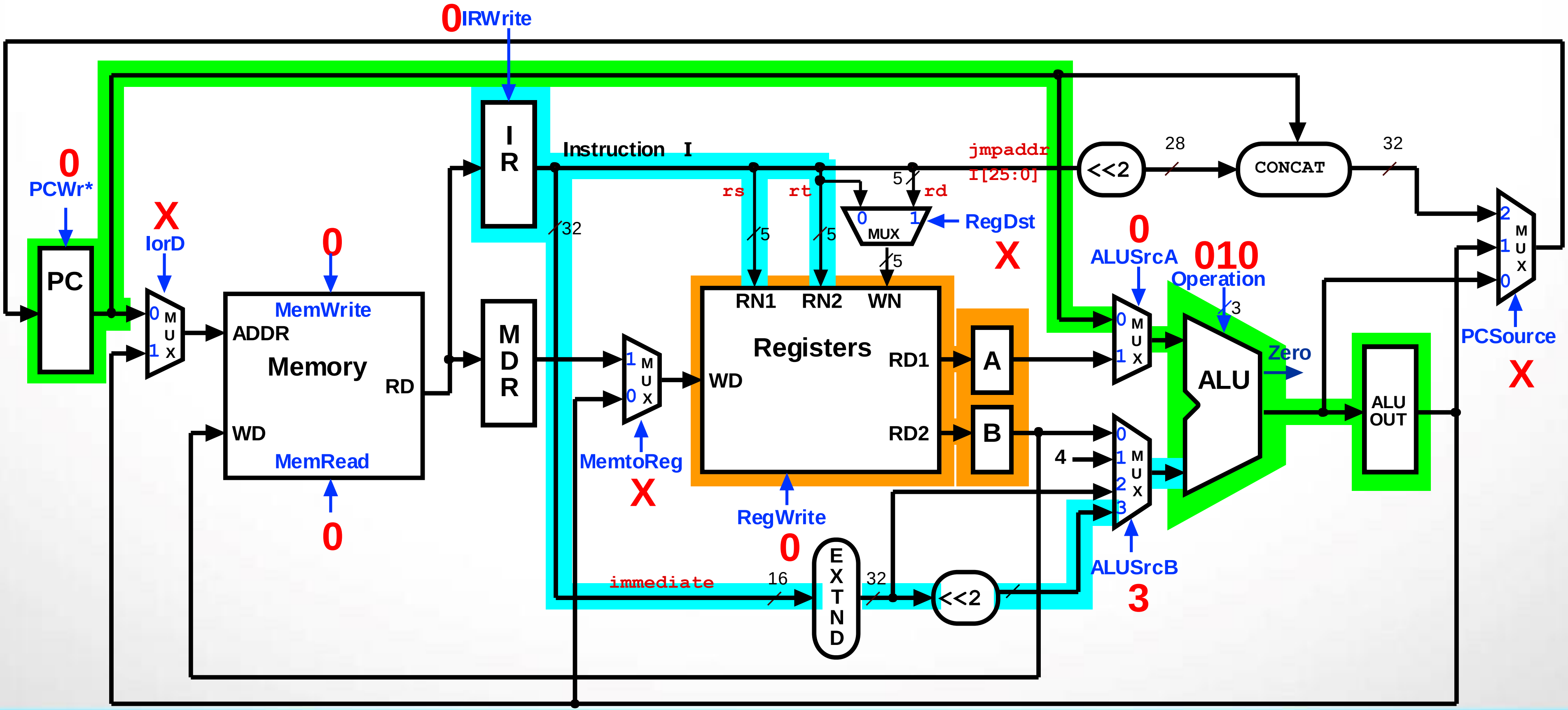
多周期执行步骤 (1)： Branch

```
IR = Memory[PC];
PC = PC + 4;
```



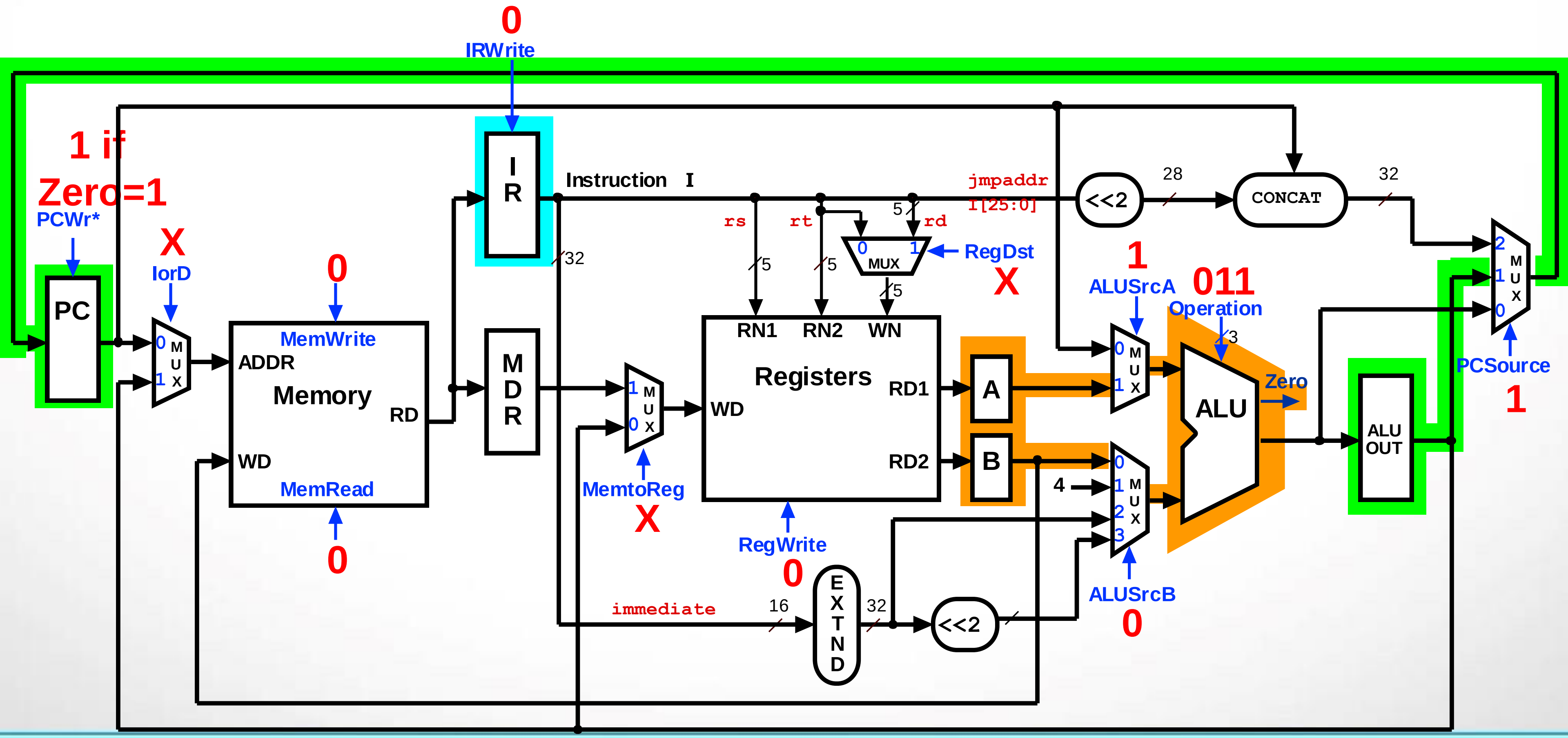
多周期执行步骤 (2)： Branch

```
A = Reg[IR[25-21]];              (A = Reg[rs])
B = Reg[IR[20-15]];              (B = Reg[rt])
ALUOut = (PC + sign-extend(IR[15-0]) << 2)
```



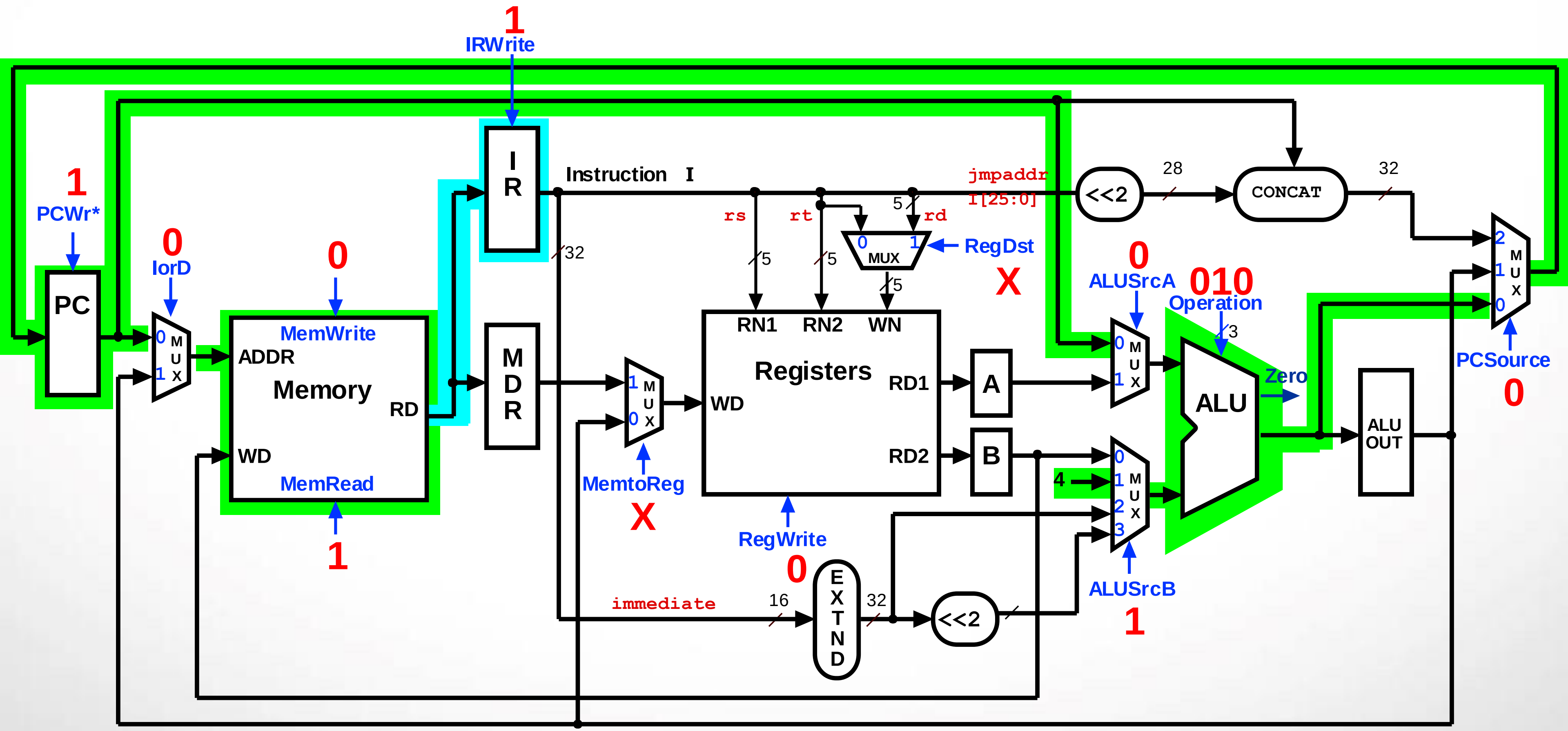
多周期执行步骤 (3)： Branch

```
if (A == B) PC = ALUOut;
```



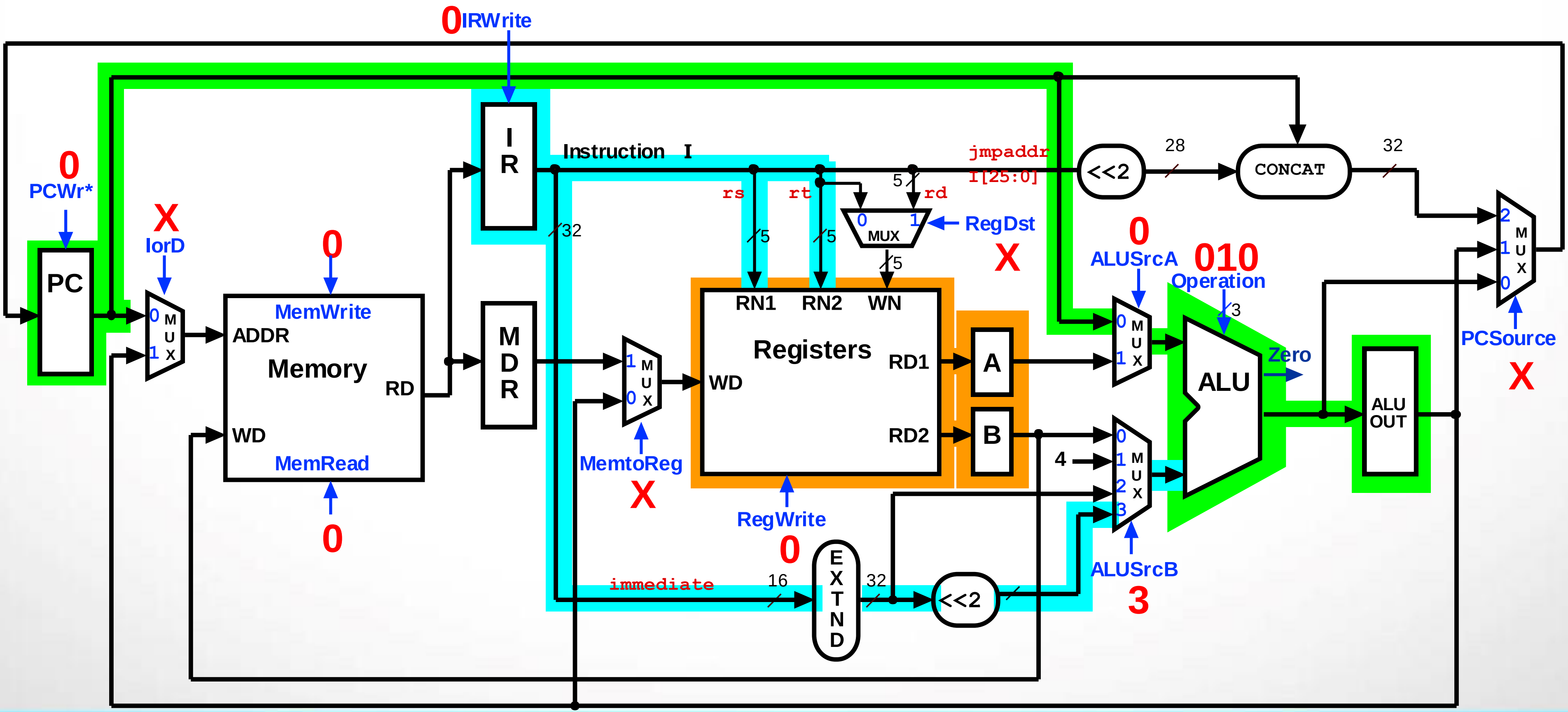
多周期执行步骤 (1)：Jump

```
IR = Memory[PC];
PC = PC + 4;
```



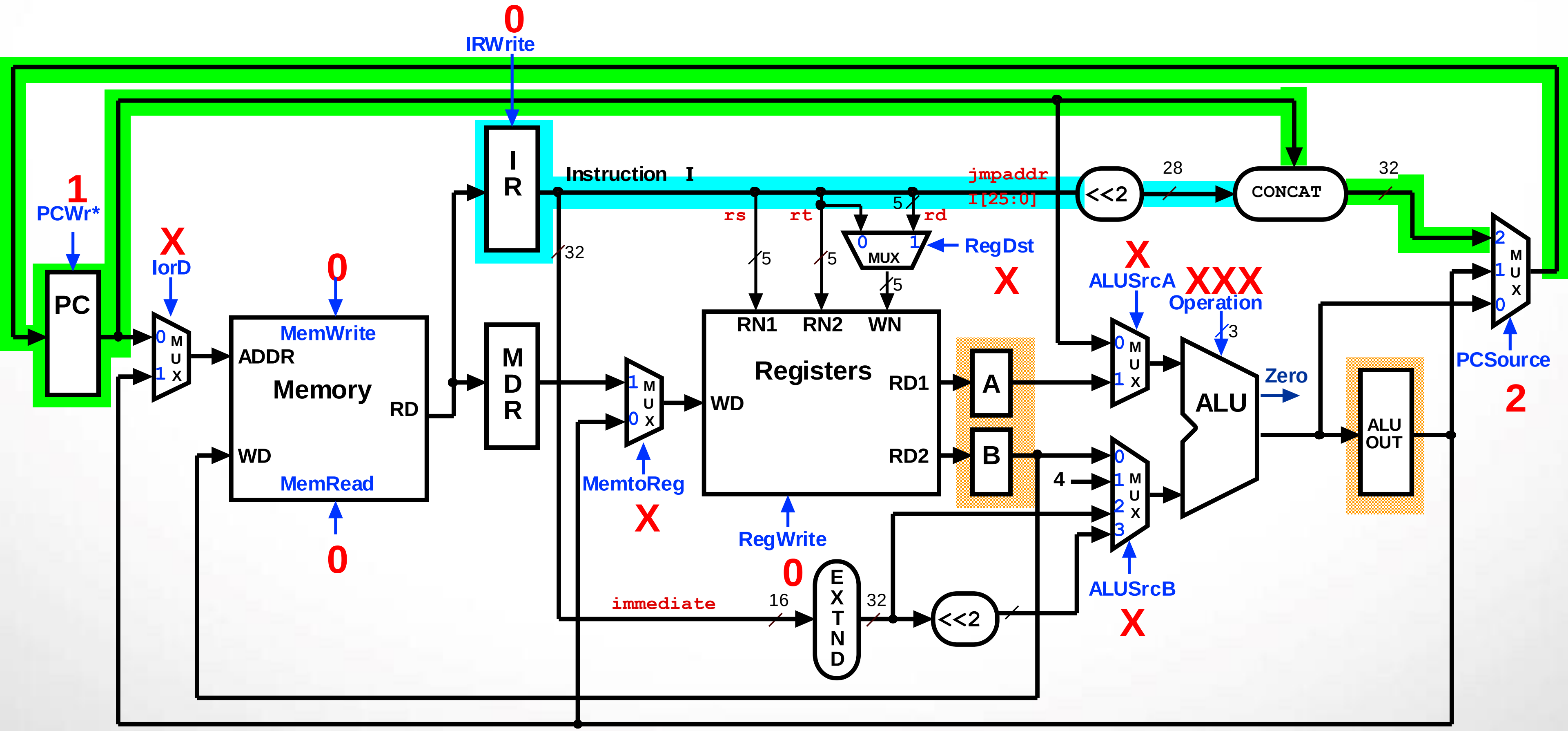
多周期执行步骤 (2)： Jump

```
A = Reg[IR[25-21]];              (A = Reg[rs])
B = Reg[IR[20-15]];              (B = Reg[rt])
ALUOut = (PC + sign-extend(IR[15-0]) << 2)
```

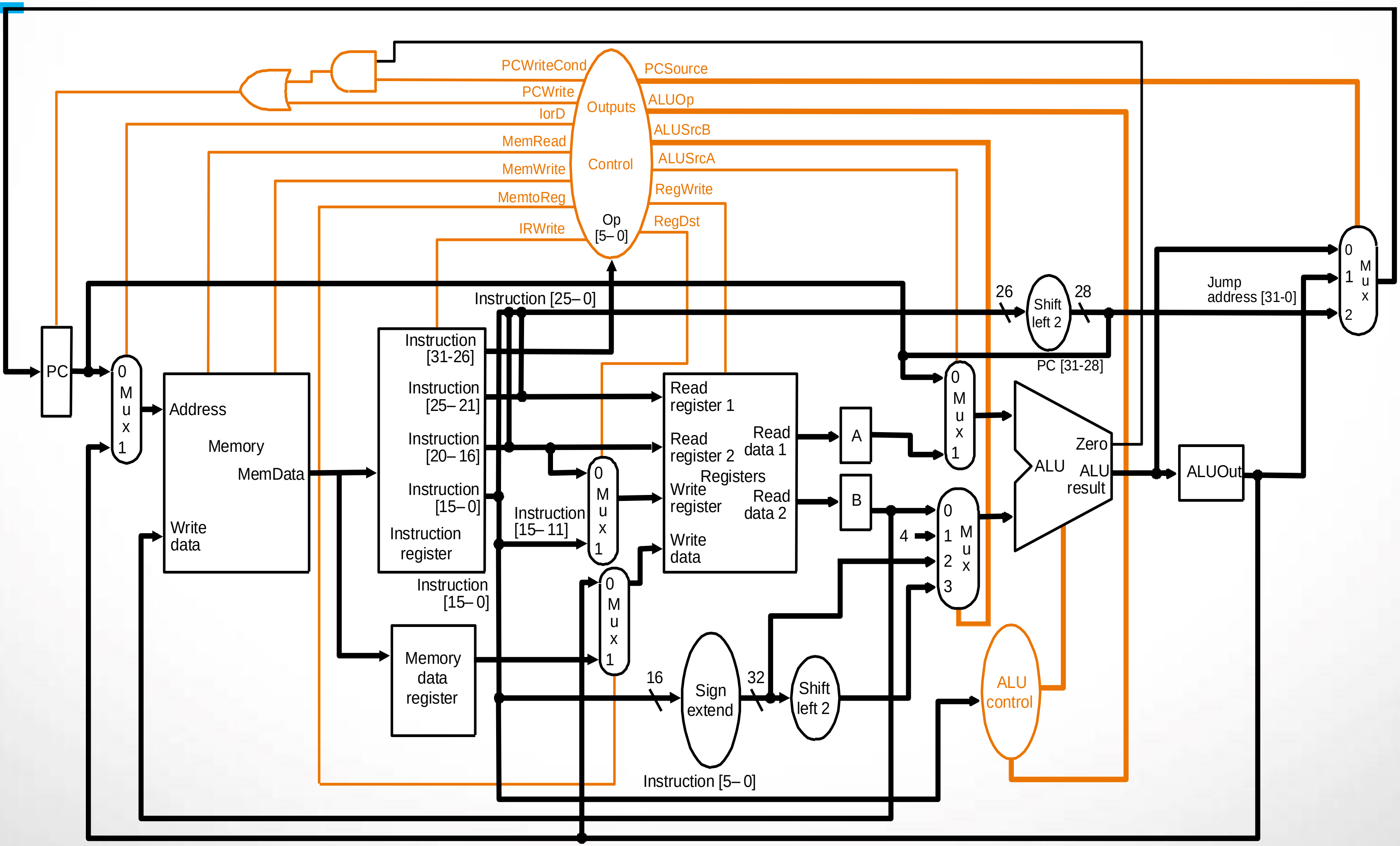


多周期执行步骤 (3)： Jump

```
PC = PC[21-28] concat (IR[25-0] << 2)
```



完整数据通路 & 控制


PCWriteCond
PCWrite
IorD
MemRead
MemWrite
MemtoReg
IRWrite
Outputs
Control
Op
[5– 0]
PCSource
ALUOp
ALUSrcB
ALUSrcA
RegWrite
RegDst
26
Shift
left 2
28
Jump
address [31-0]
0
1
M
u
x
2
Instruction [25– 0]
PC [31-28]
PC
0
M
u
x
1
Address
Memory
MemData
Write
data
Instruction
[31-26]
Instruction
[25– 21]
Instruction
[20– 16]
Instruction
[15– 0]
Instruction
register
Instruction
[15– 11]
Instruction
[15– 0]
Memory
data
register
Read
register 1
Read
register 2
Registers
Write
register
Write
data
Read
data 1
Read
data 2
A
B
4
0
1
2
3
Zero
ALU
ALU
result
ALUOut
16
Sign
extend
32
Shift
left 2
ALU
control
Instruction [5– 0]