

# **CPS104 Computer Organization**

## **Lecture 18**

### **Control for the Datapath**

**October 28 , 2009**

**Gershon Kedem**

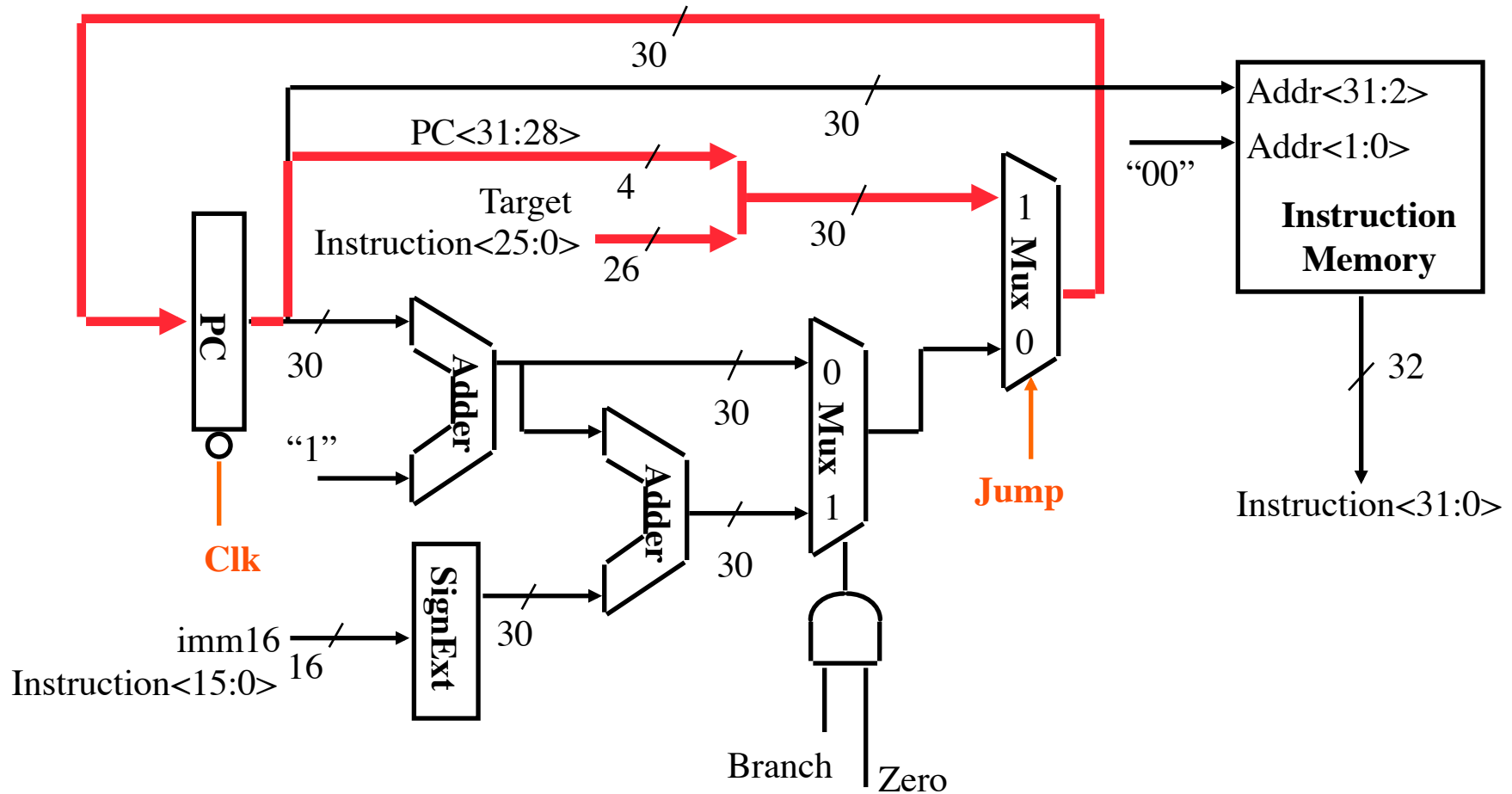
# Administratrivia

- Homework-5 is posted; **Due Monday November 2, in class**
- Reading: Chapter 4.
- Reading: Appendix C
  - ◆ Available on: . . . [cps104/Handouts/Appendix-C.pdf](#)

# Review: The Instruction Fetch Unit

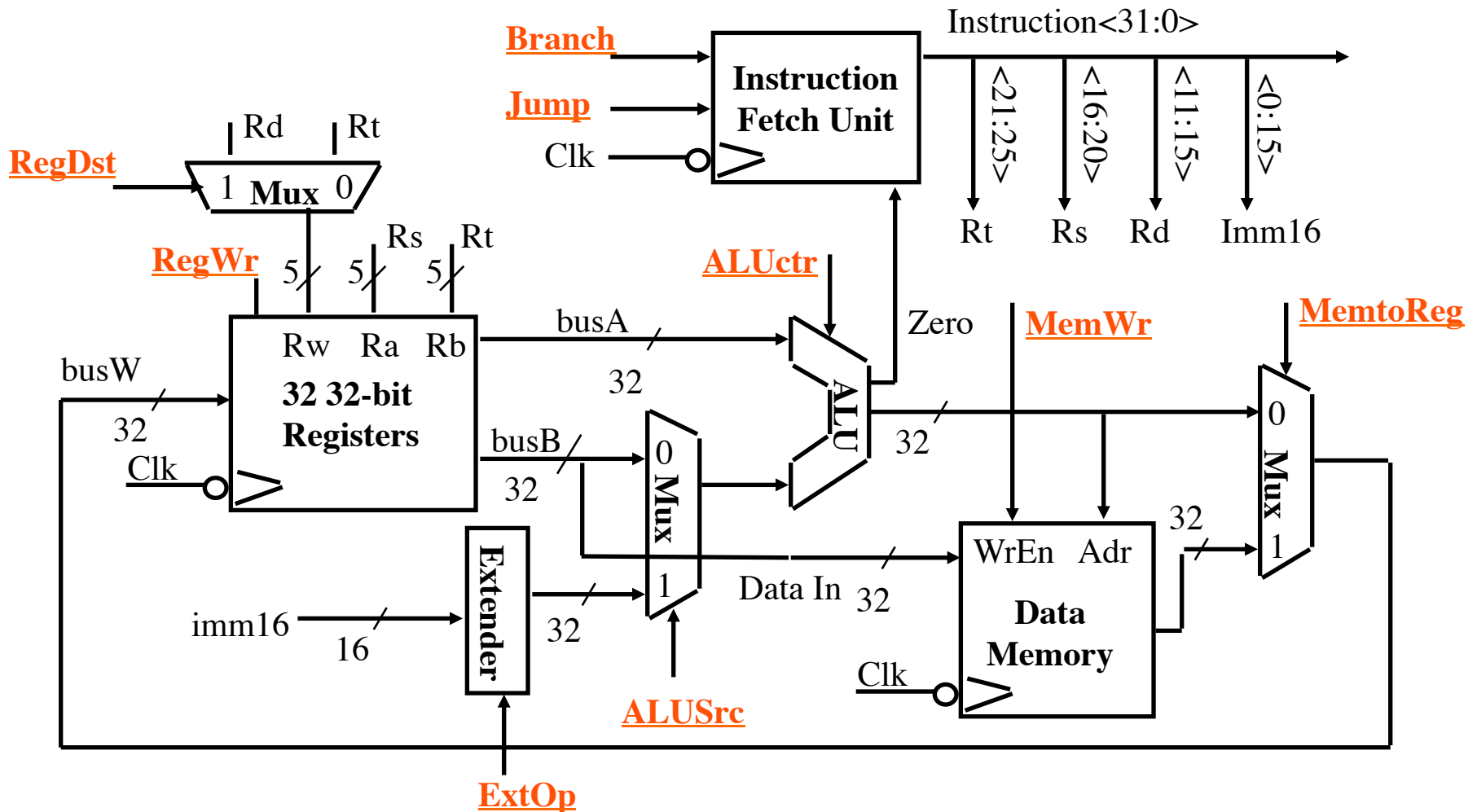
• j target

◆  $PC\langle 31:2 \rangle \leftarrow PC\langle 31:28 \rangle \text{ concat target}\langle 25:0 \rangle$



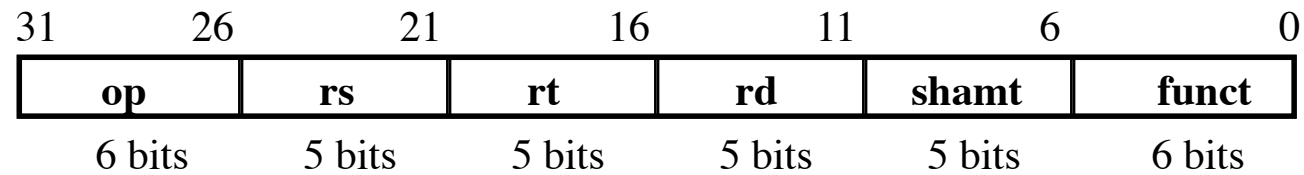
# Review: A Single Cycle Datapath

- We have everything except **control signals**.



# What about Controls?

## Review: RTL: The ADD Instruction



• **add rd, rs, rt**

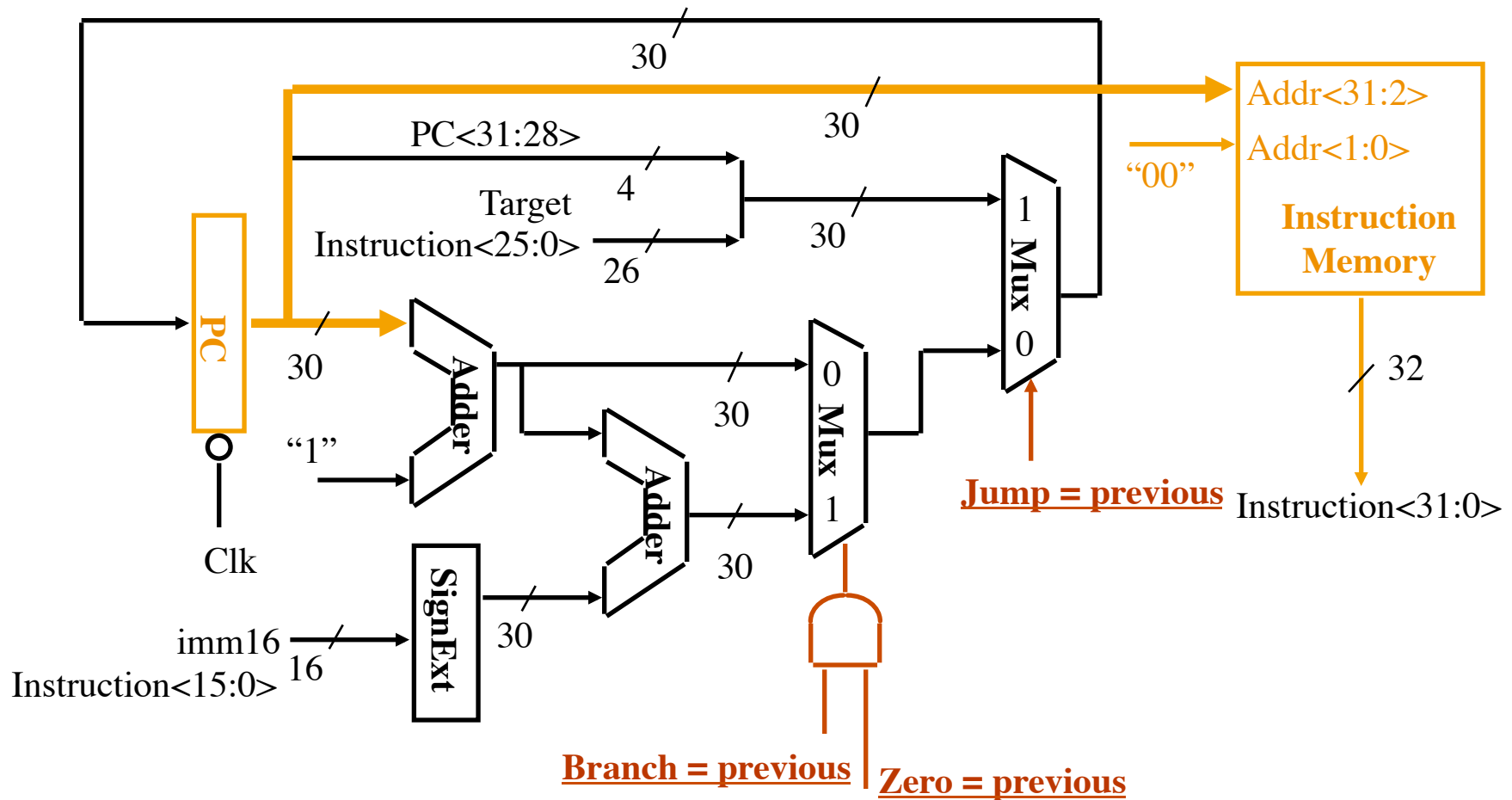
◆ **mem[PC]**                      **Fetch the instruction from memory**

◆  **$R[rd] \leftarrow R[rs] + R[rt]$**                       **The actual operation**

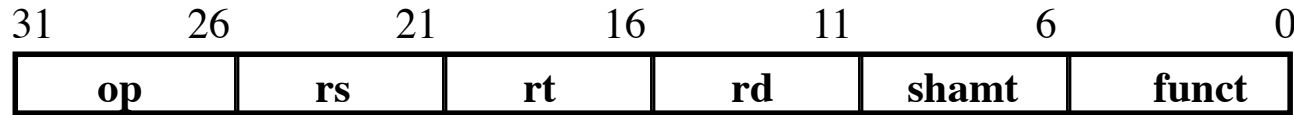
◆  **$PC \leftarrow PC + 4$**                       **Calculate the next instruction's address**

# Instruction Fetch Unit at the Beginning of Add / Subtract

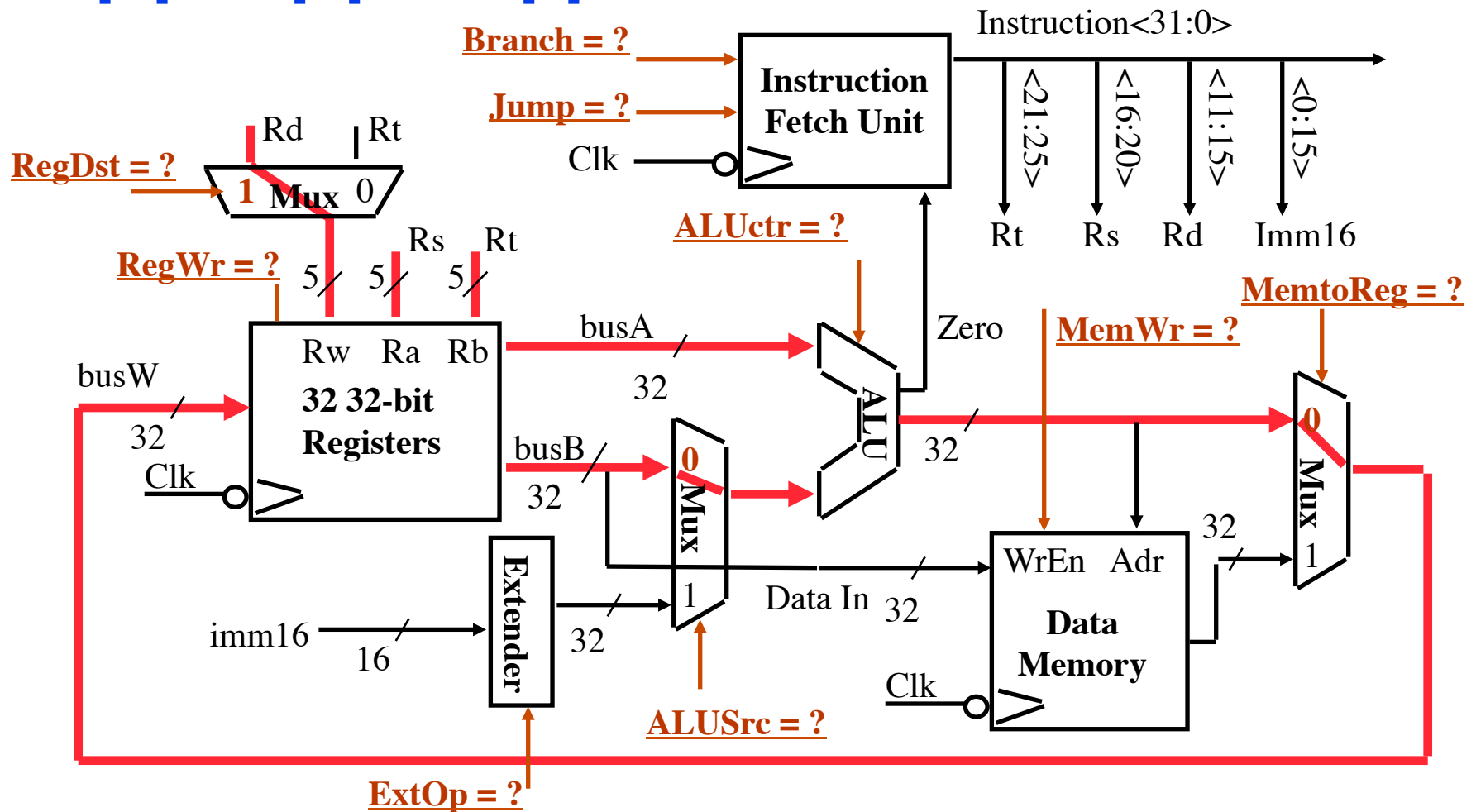
- Fetch the instruction from Instruction memory:  $\text{Instruction} \leftarrow \text{mem}[\text{PC}]$ 
  - This is the same for all instructions



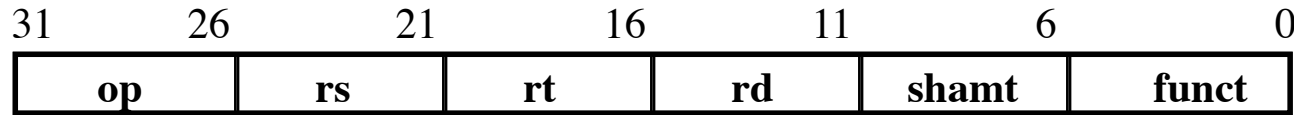
# The Single Cycle Datapath during Add and Subtract



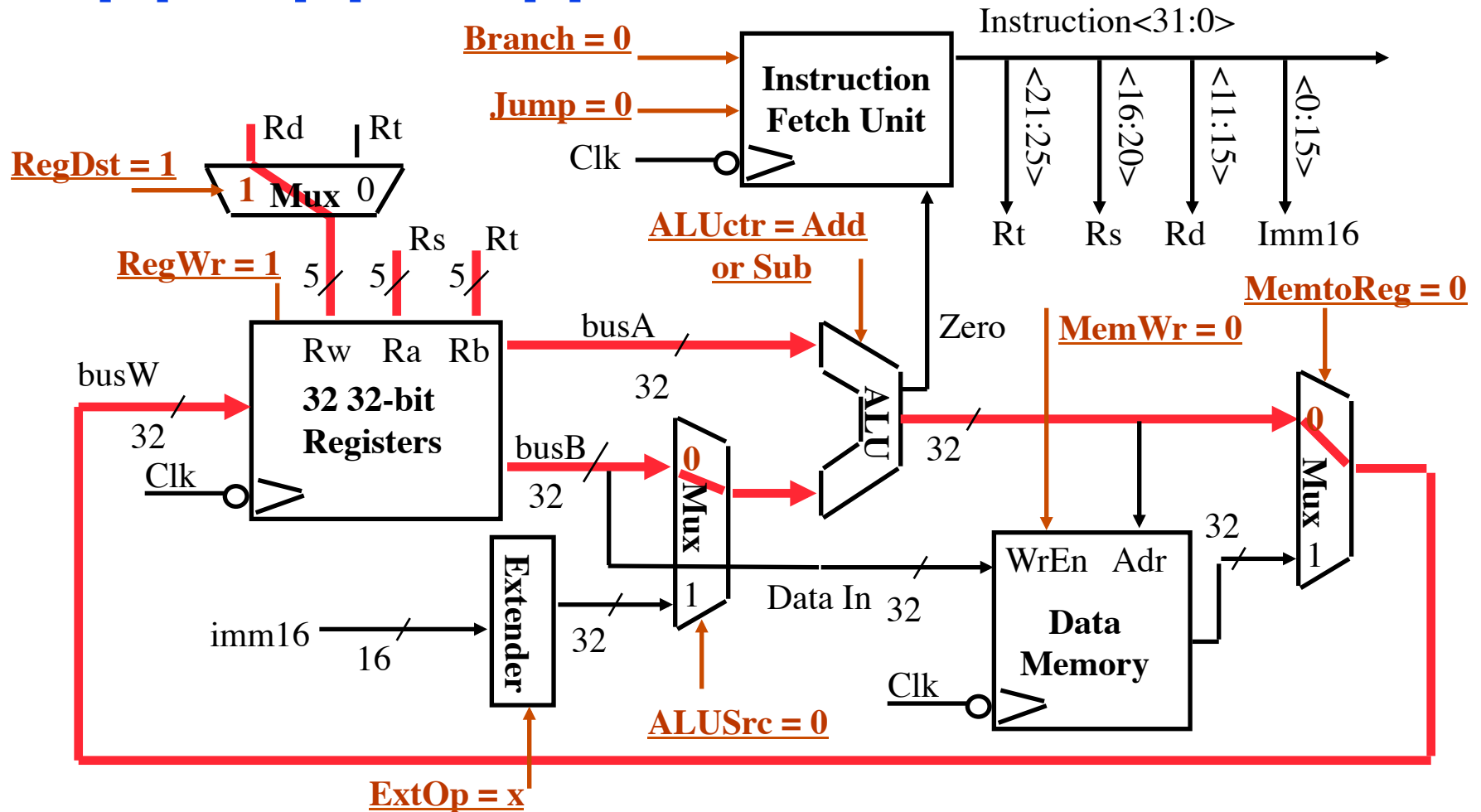
**\*  $R[rd] \leftarrow R[rs] + / - R[rt]$**



# The Single Cycle Datapath during Add and Subtract



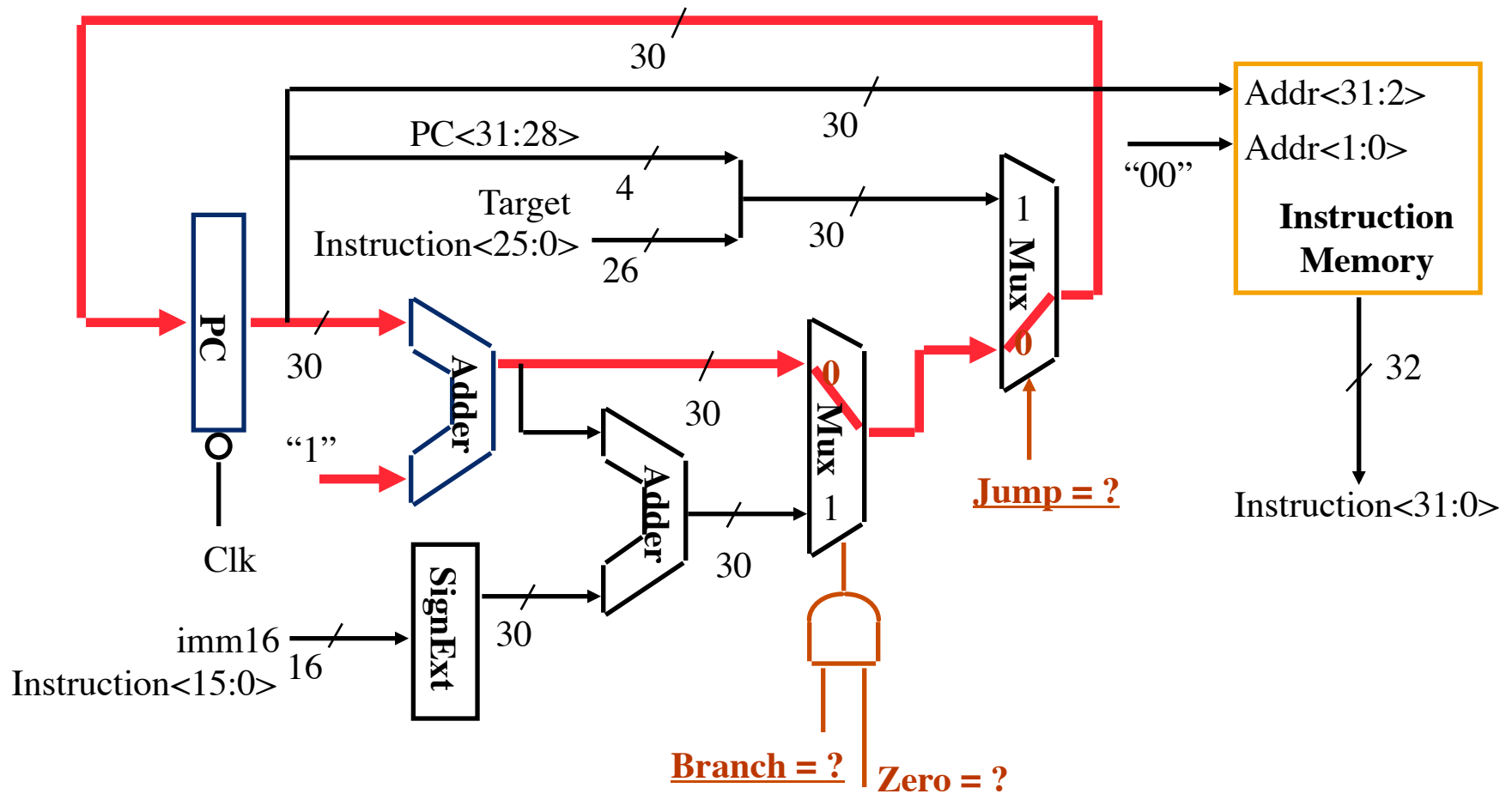
•  $R[rd] \leftarrow R[rs] + / - R[rt]$



# Instruction Fetch Unit at the End of Add and Subtract

•  $PC \leftarrow PC + 4$

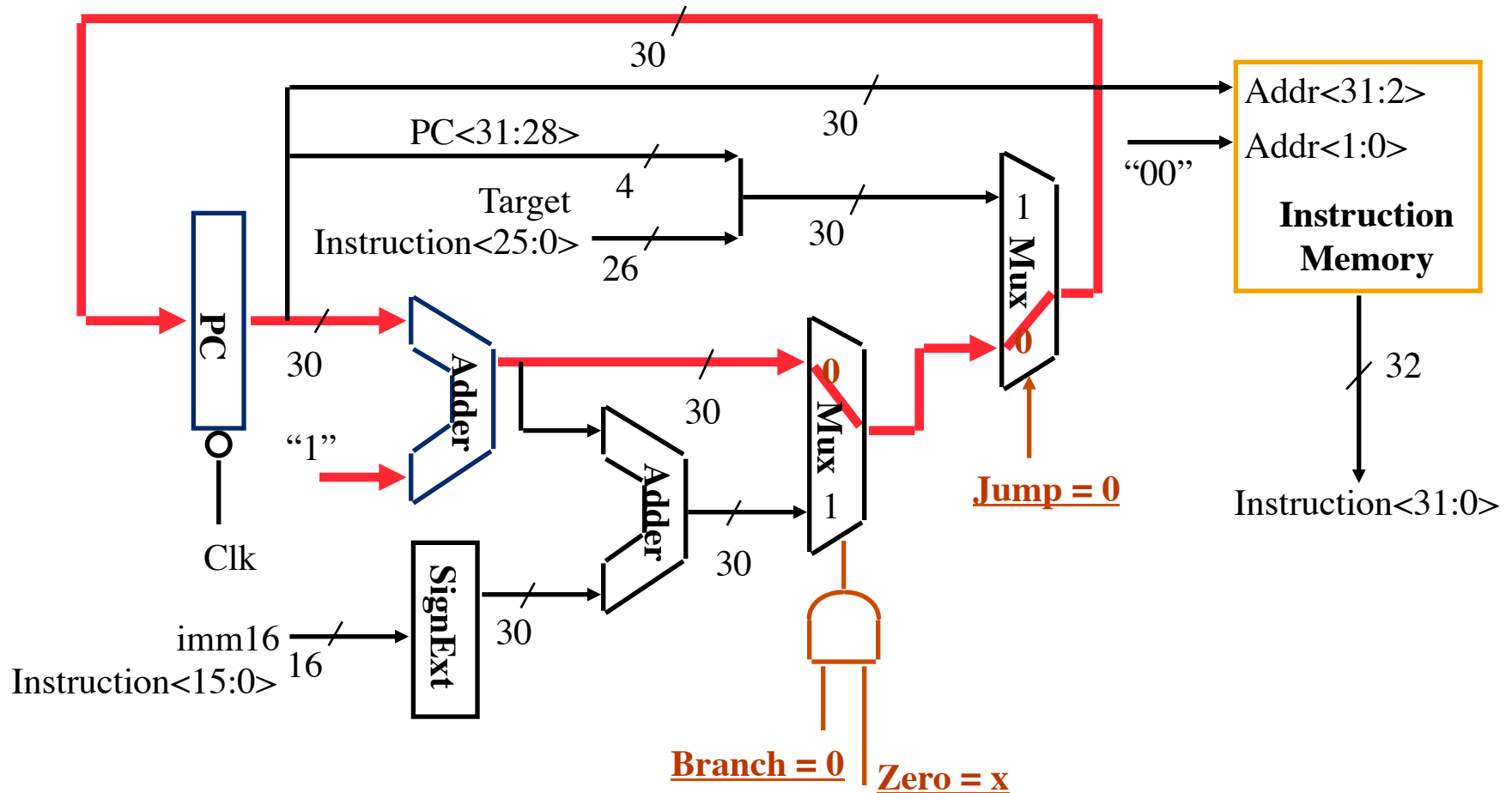
◆ This is the same for all instructions except: Branch and Jump



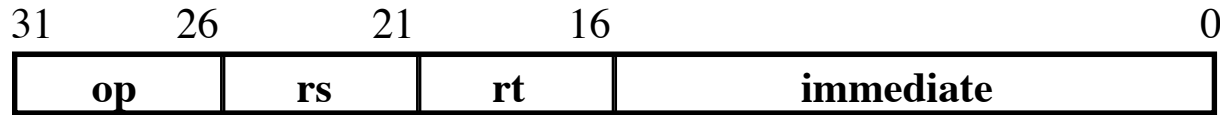
# Instruction Fetch Unit at the End of Add and Subtract

•  $PC \leftarrow PC + 4$

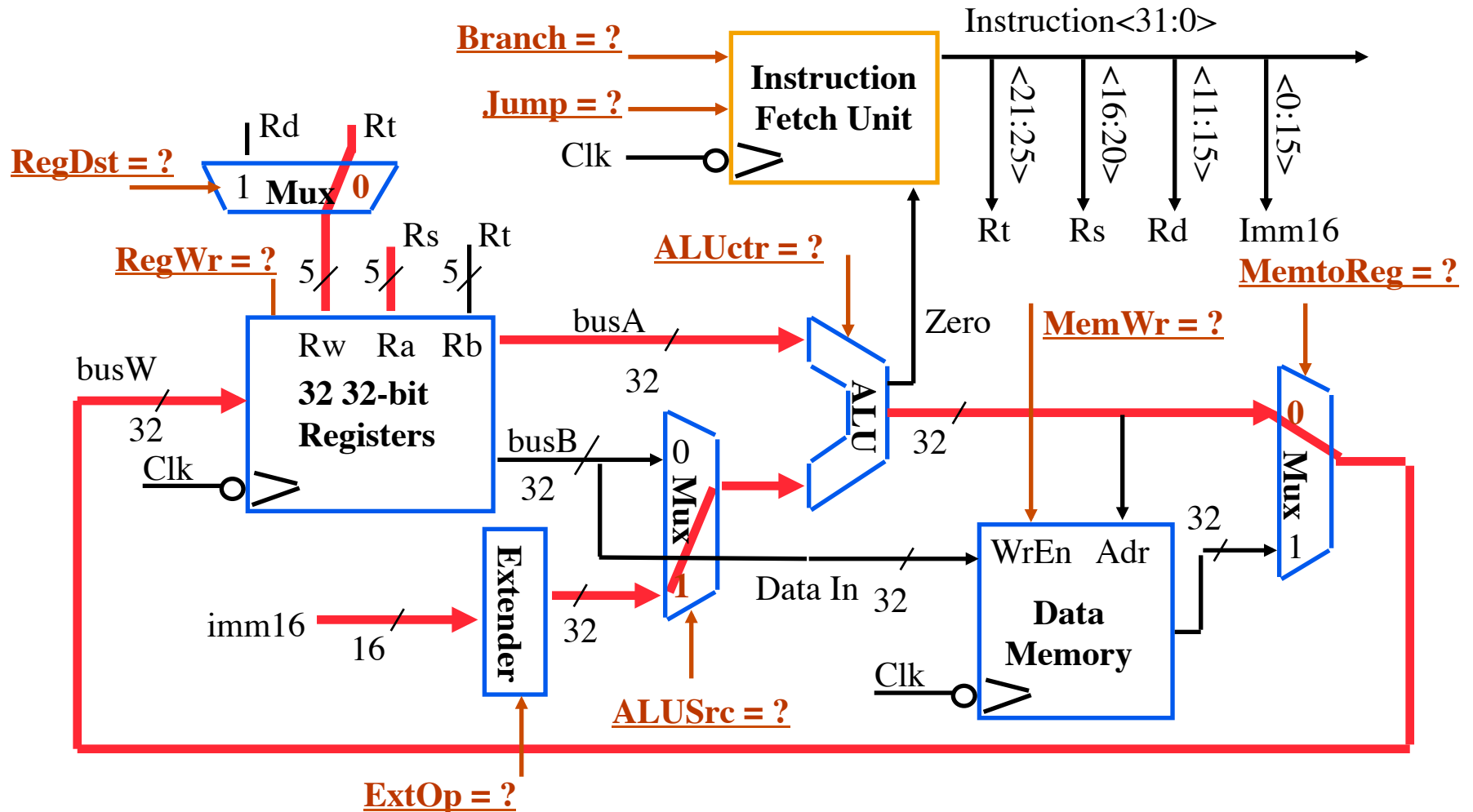
◆ This is the same for all instructions except: Branch and Jump



# The Single Cycle Datapath during Or Immediate

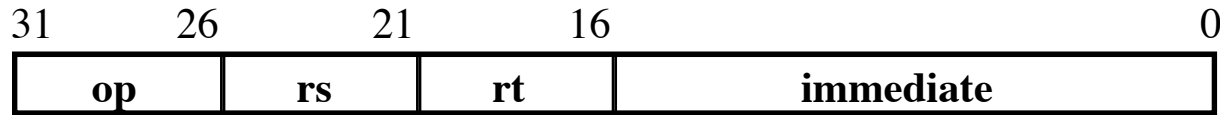


•  $R[rt] \leftarrow R[rs] \text{ or } \text{ZeroExt}[\text{Imm16}]$

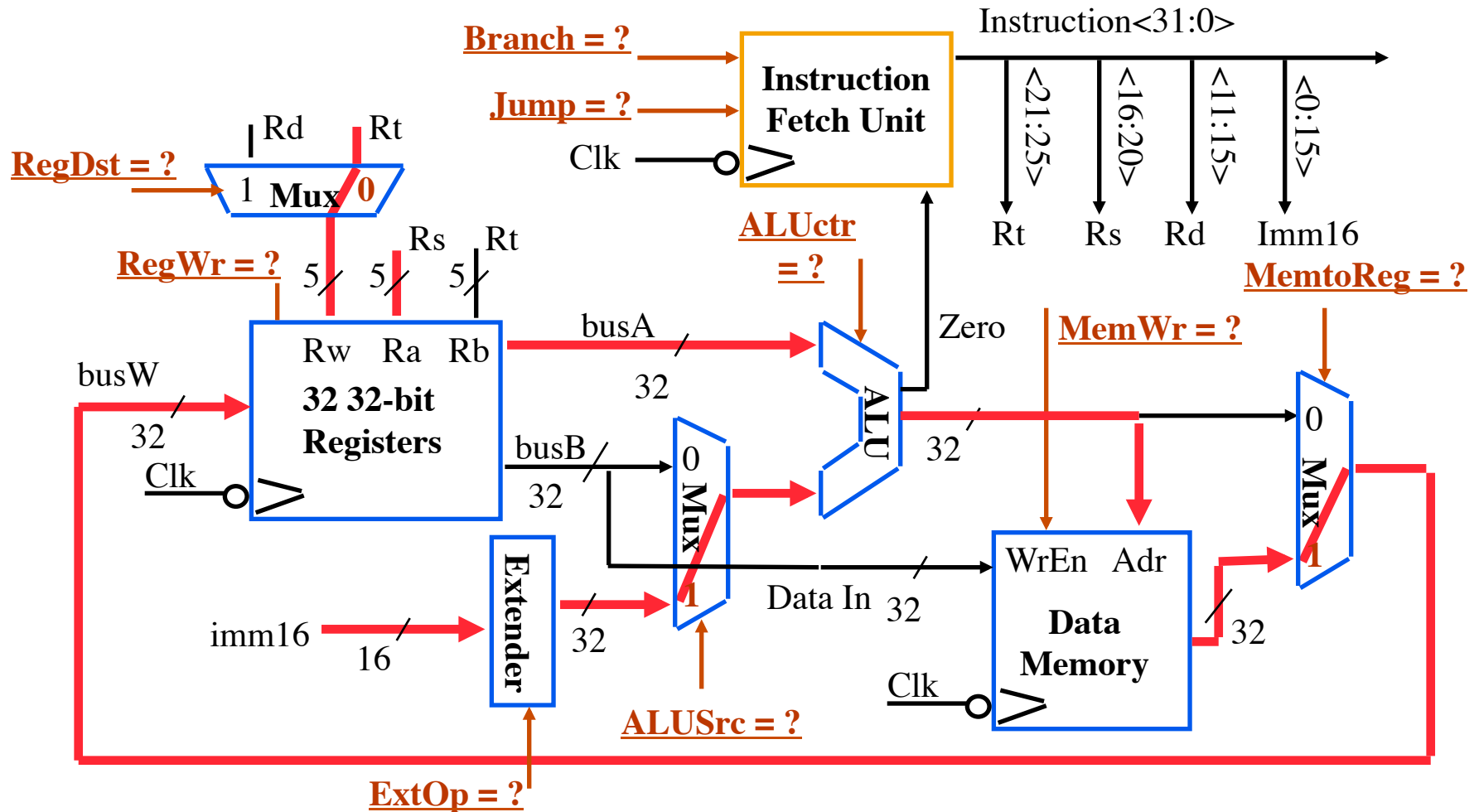




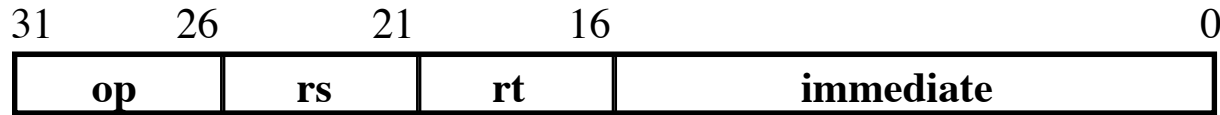
# The Single Cycle Datapath during Load



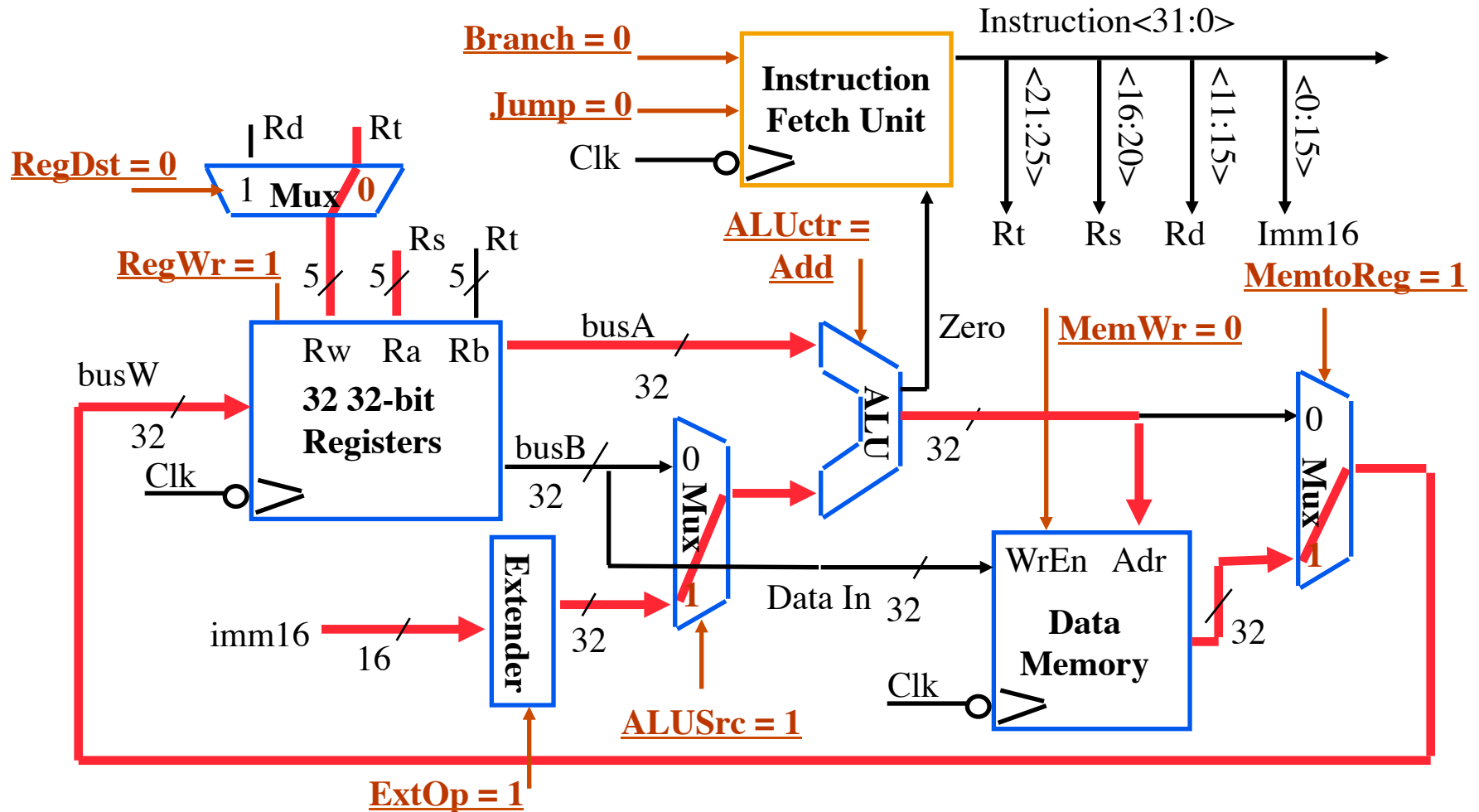
•  $R[rt] \leftarrow \text{Data Memory} \{R[rs] + \text{SignExt}[\text{imm16}]\}$



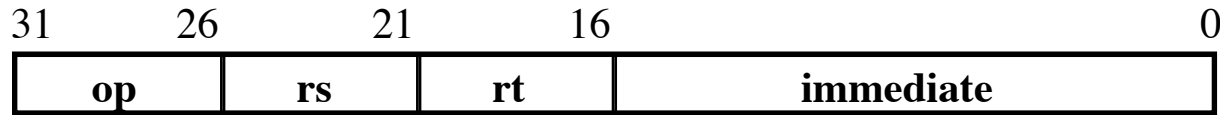
# The Single Cycle Datapath during Load



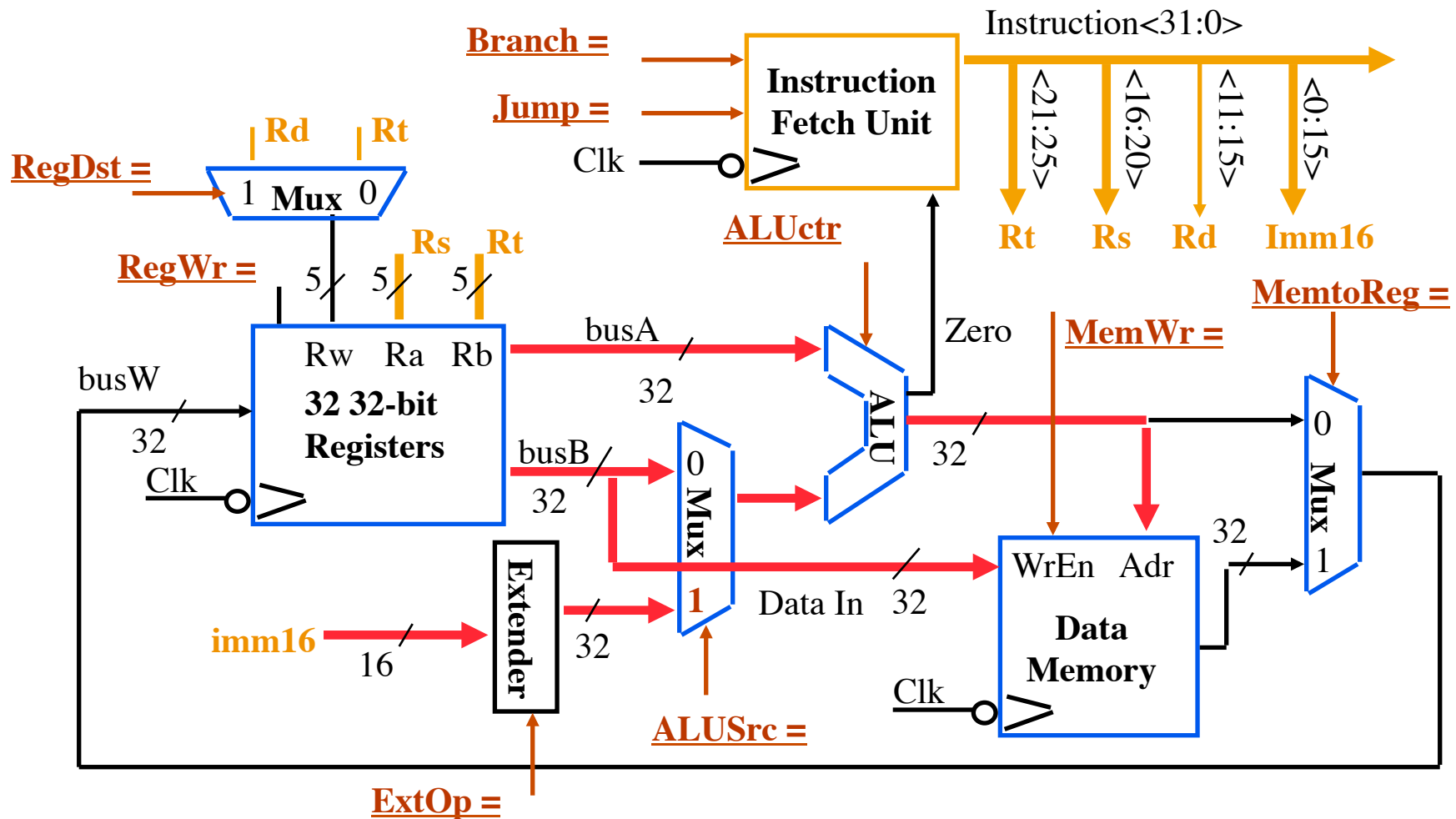
•  $R[rt] \leftarrow \text{Data Memory} \{R[rs] + \text{SignExt}[imm16]\}$



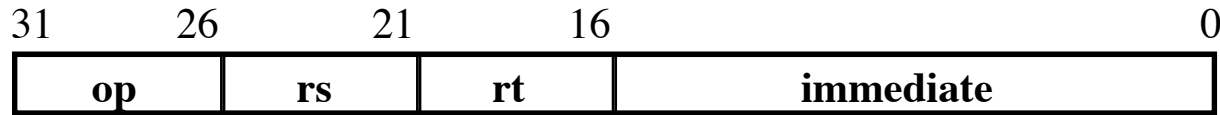
# The Single Cycle Datapath during Store



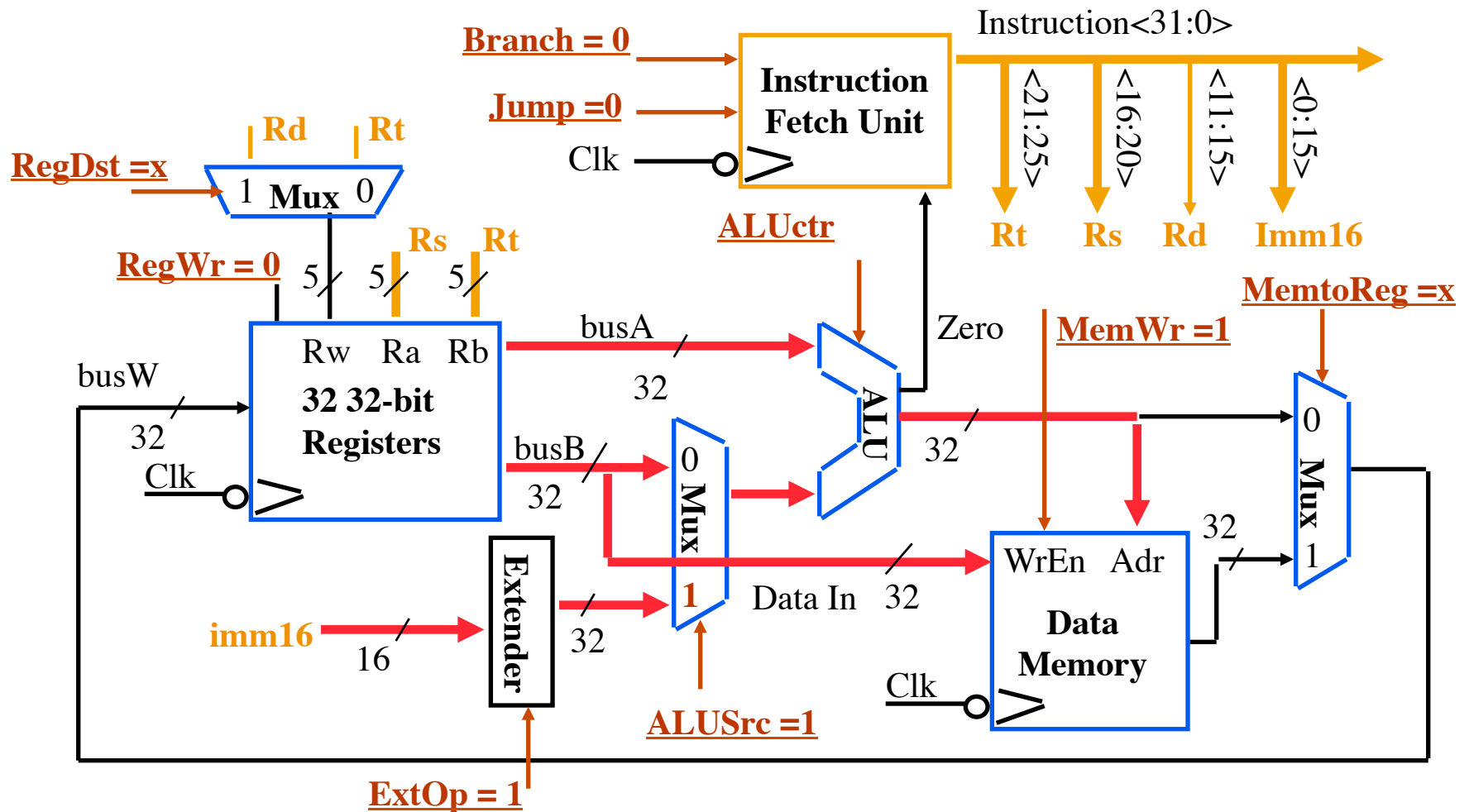
• Data Memory {R[rs] + SignExt[imm16]} ← R[rt]



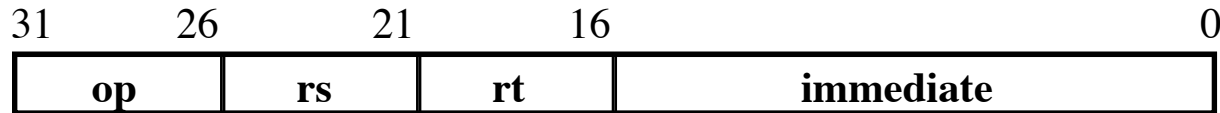
# The Single Cycle Datapath during Store



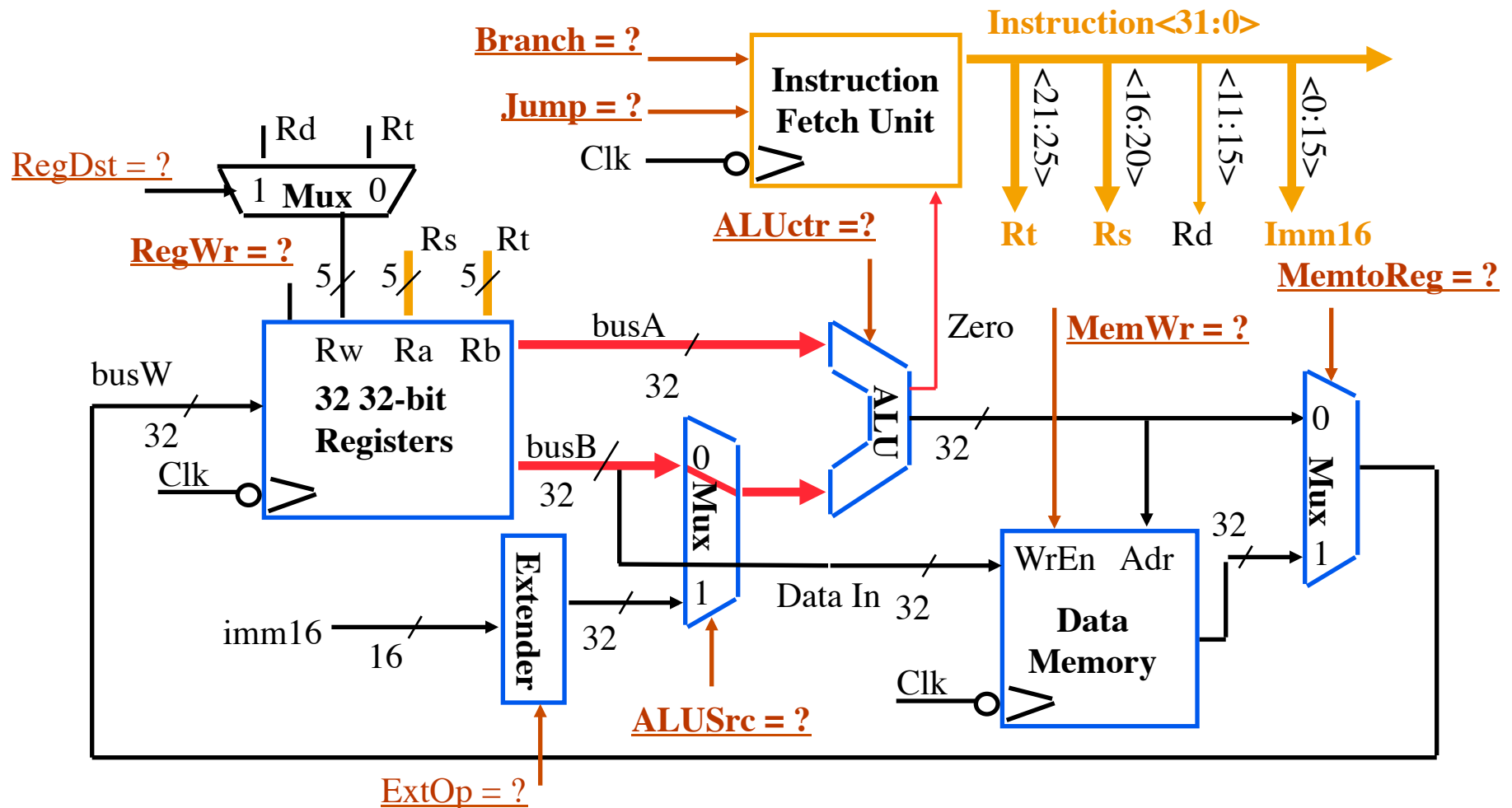
• **Data Memory {R[rs] + SignExt[imm16]} ← R[rt]**



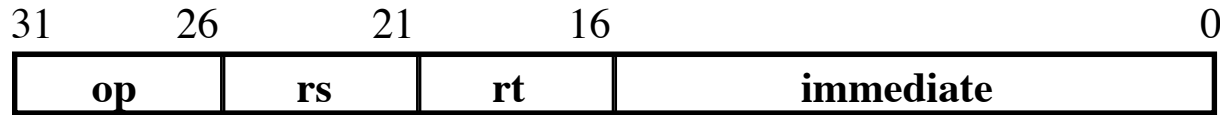
# The Single Cycle Datapath during Branch



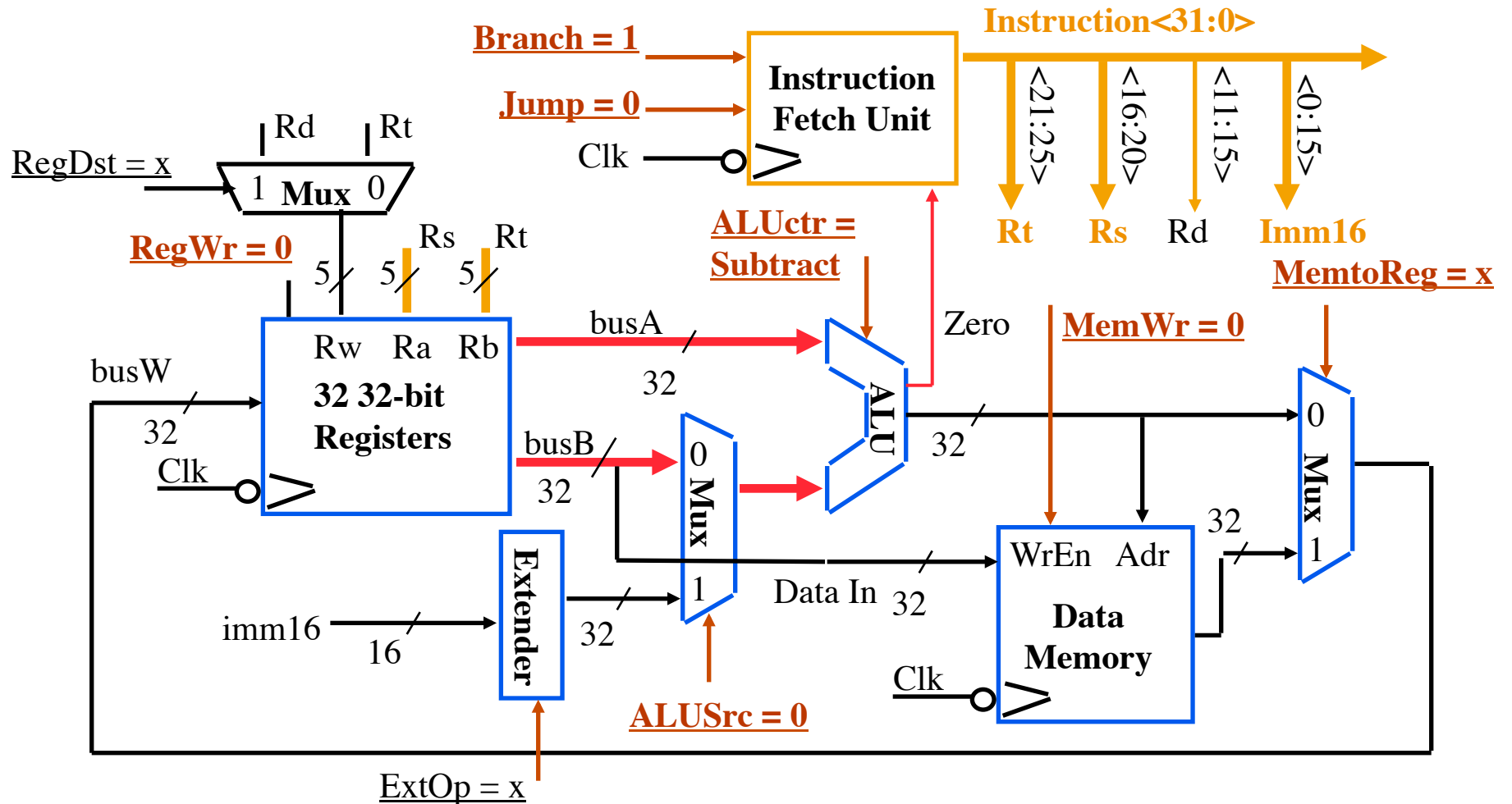
• if (R[rs] - R[rt] == 0) then Zero <- 1 ; else Zero <- 0



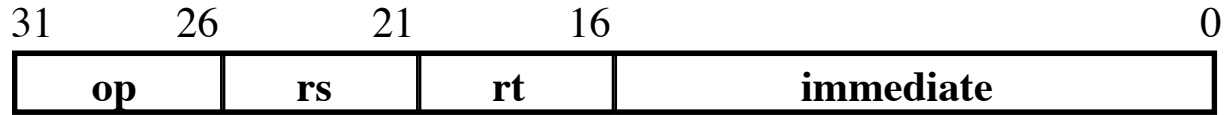
# The Single Cycle Datapath during Branch



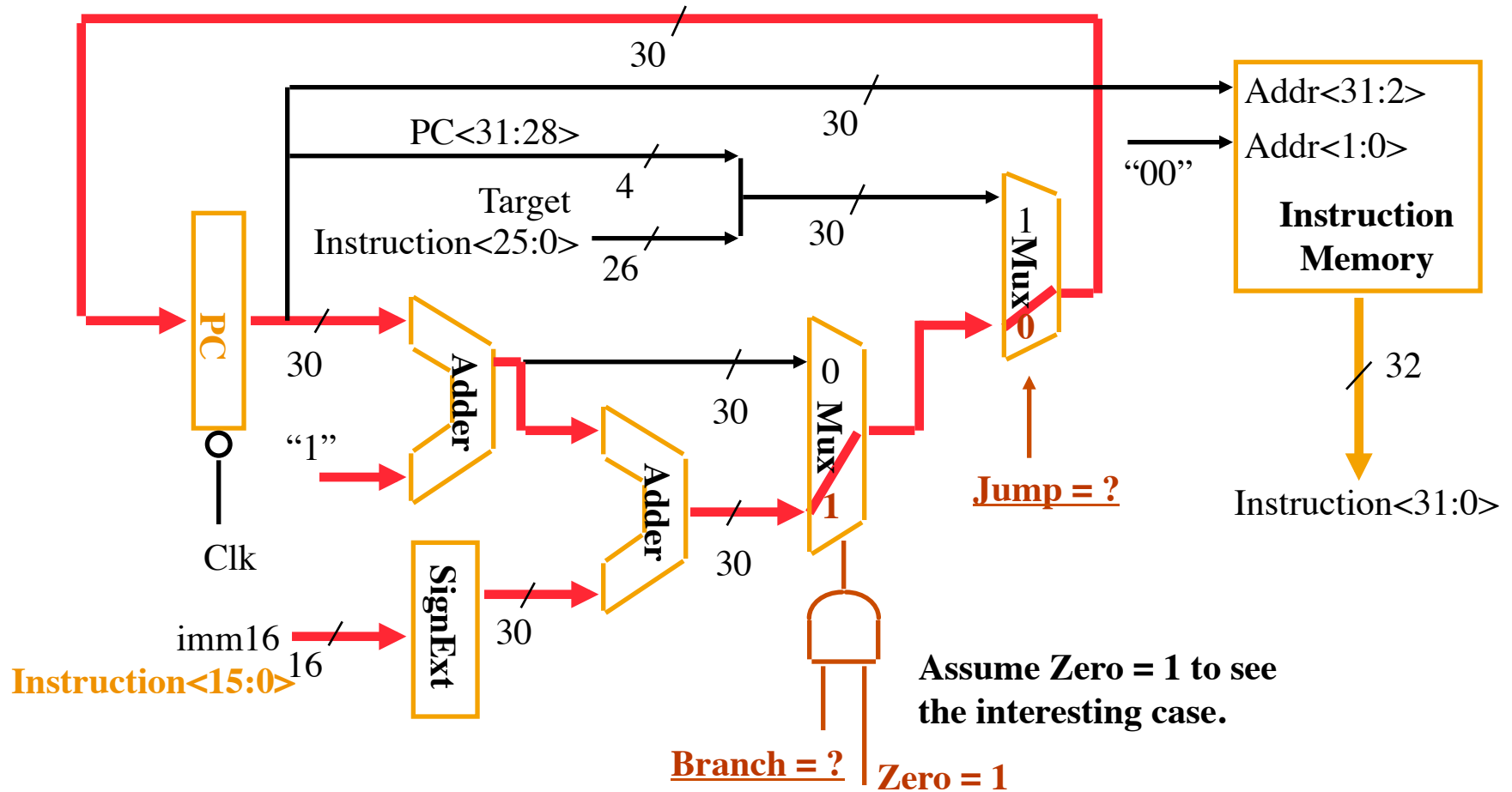
• if (R[rs] - R[rt] == 0) then Zero <- 1 ; else Zero <- 0



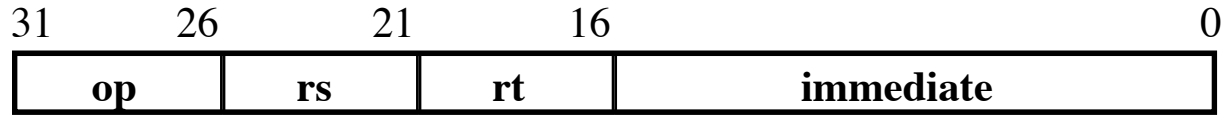
# Instruction Fetch Unit at the End of Branch



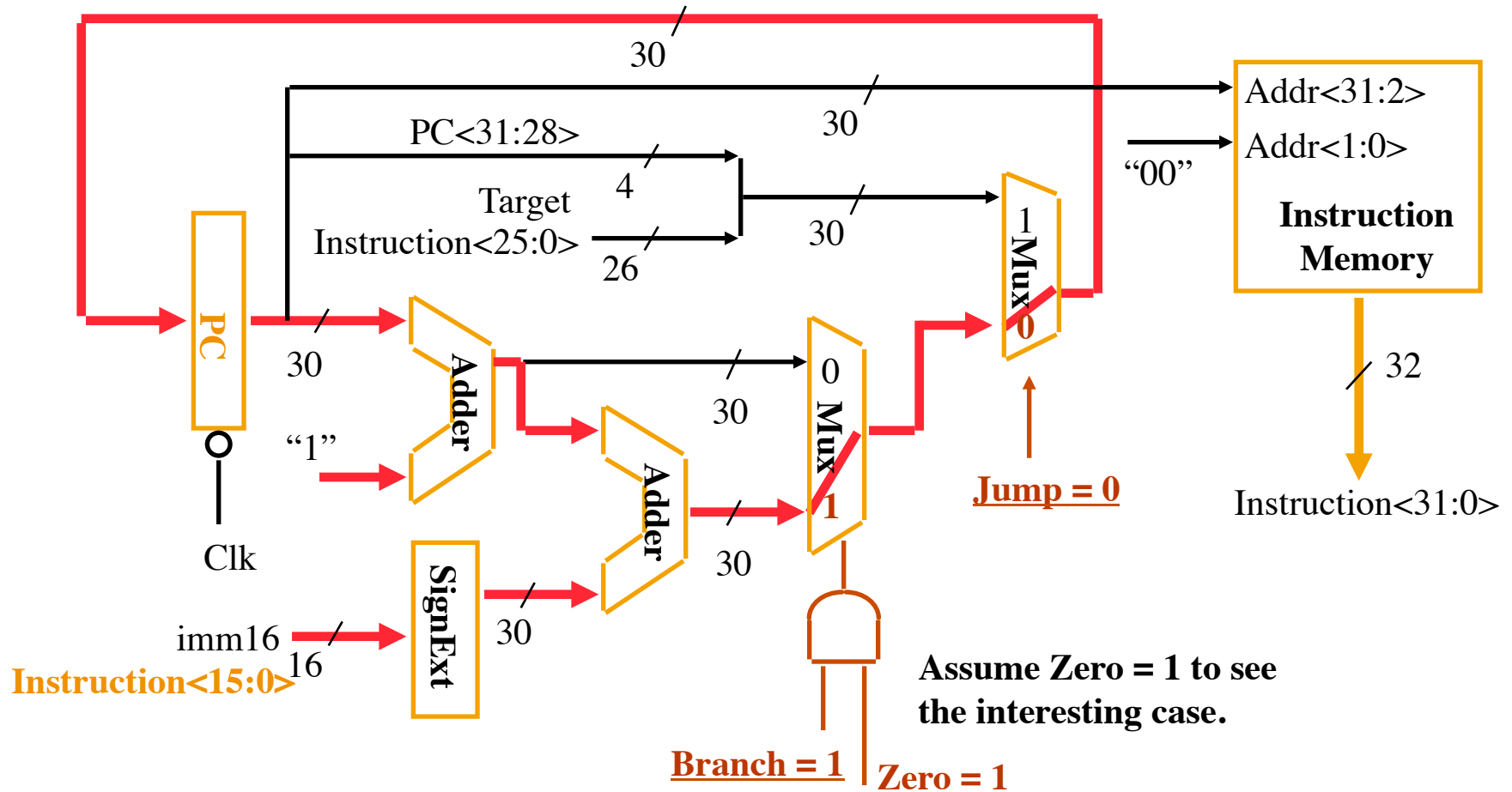
• if (Zero == 1) then  $PC = PC + 4 + \text{SignExt}[\text{imm16}] * 4$  ; else  $PC = PC + 4$



# Instruction Fetch Unit at the End of Branch



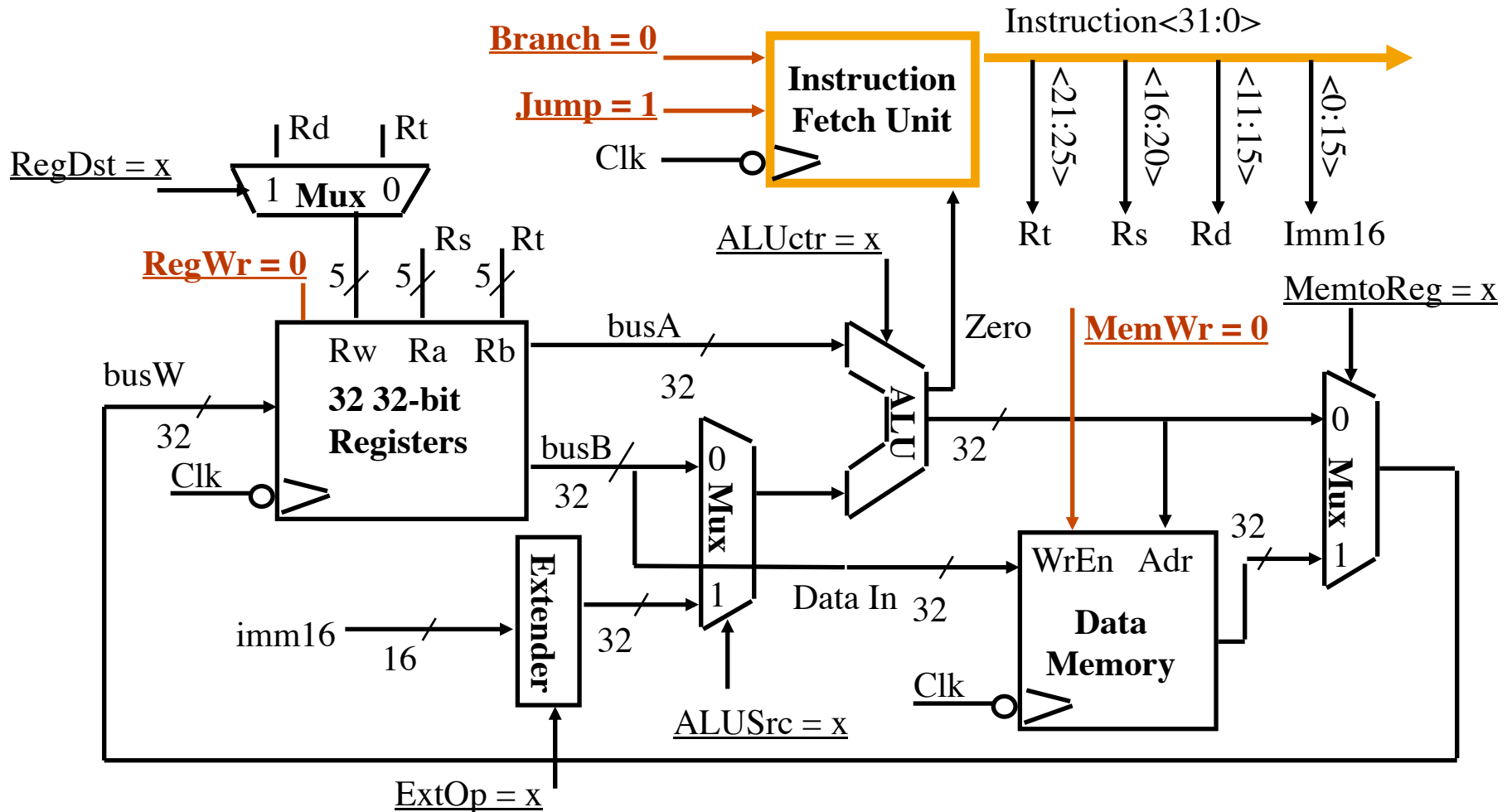
• if (Zero == 1) then  $PC = PC + 4 + \text{SignExt}[\text{imm16}] * 4$  ; else  $PC = PC + 4$



# The Single Cycle Datapath during Jump



• Nothing to do! Make sure control signals are set correctly!

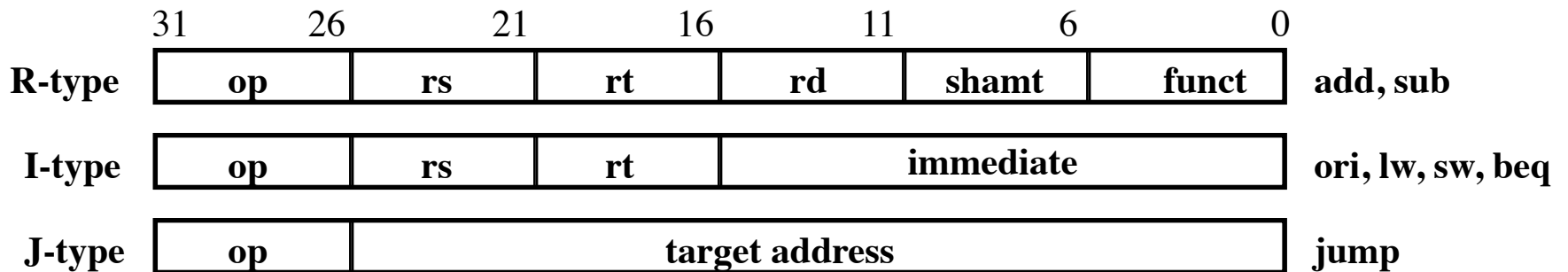




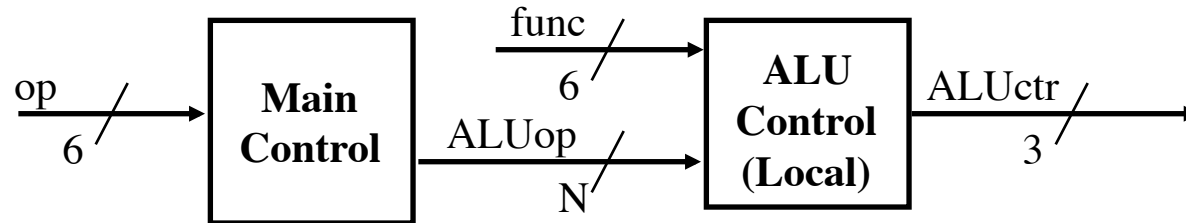
# A Summary of the Control Signals

See Appendix A → **func**  
 See Appendix A → **op**

	10 0000	10 0010	We Don't Care :-)				
	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	<b>add</b>	<b>sub</b>	<b>ori</b>	<b>lw</b>	<b>sw</b>	<b>beq</b>	<b>jump</b>
<b>RegDst</b>	1	1	0	0	x	x	x
<b>ALUSrc</b>	0	0	1	1	1	0	x
<b>MemtoReg</b>	0	0	0	1	x	x	x
<b>RegWrite</b>	1	1	1	1	0	0	0
<b>MemWrite</b>	0	0	0	0	1	0	0
<b>Branch</b>	0	0	0	0	0	1	x
<b>Jump</b>	0	0	0	0	0	0	1
<b>ExtOp</b>	x	x	0	1	1	x	x
<b>ALUctr&lt;2:0&gt;</b>	Add	Subtract	Or	Add	Add	Subtract	xxx



# The Encoding of ALUop



- \* In this exercise, ALUop has to be 2 bits wide to represent:
  - ♦ (1) “R-type” instructions
  - ♦ “I-type” instructions that require the ALU to perform:
    - (2) Or, (3) Add, and (4) Subtract
  
- \* To implement the full MIPS ISA, ALUop has to be 3 bits to represent:
  - ♦ (1) “R-type” instructions
  - ♦ “I-type” instructions that require the ALU to perform:
    - (2) Or, (3) Add, (4) Subtract, and (5) And (Example: andi)

	R-type	ori	lw	sw	beq	jump
ALUop (Symbolic)	“R-type”	Or	Add	Add	Subtract	xxx
ALUop<2:0>	1 00	0 10	0 00	0 00	0 01	xxx

# The “Truth Table” for RegWrite

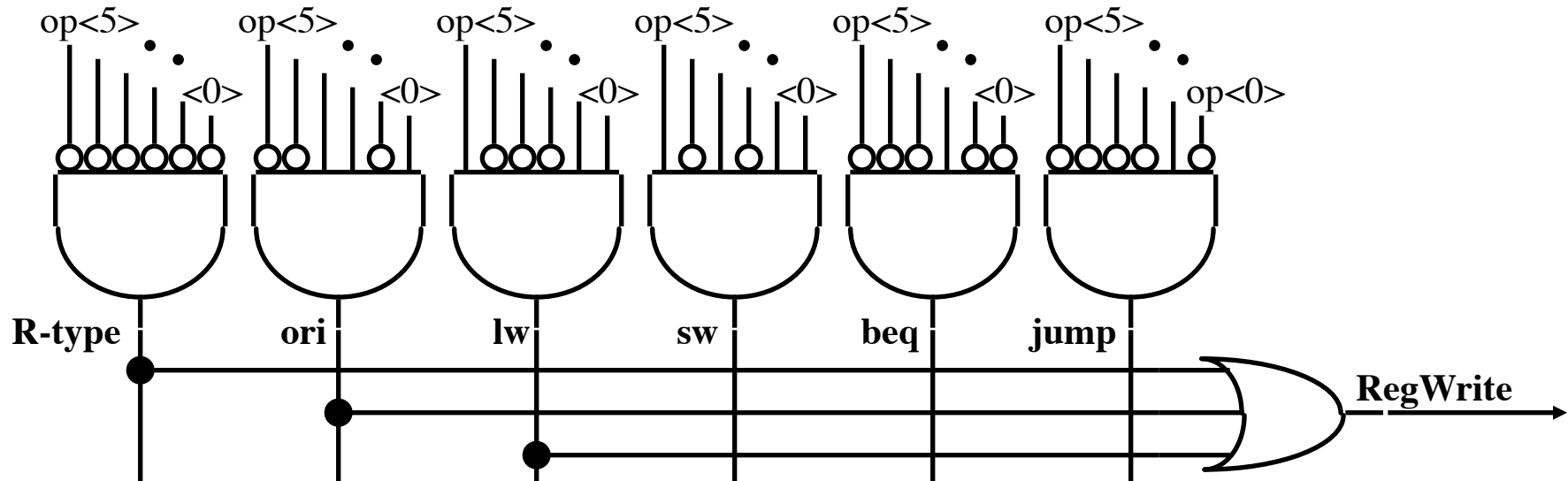
	op	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
		R-type	ori	lw	sw	beq	jump
RegWrite		1	1	1	0	0	0

\*  $\text{RegWrite} = \text{R-type} + \text{ori} + \text{lw}$

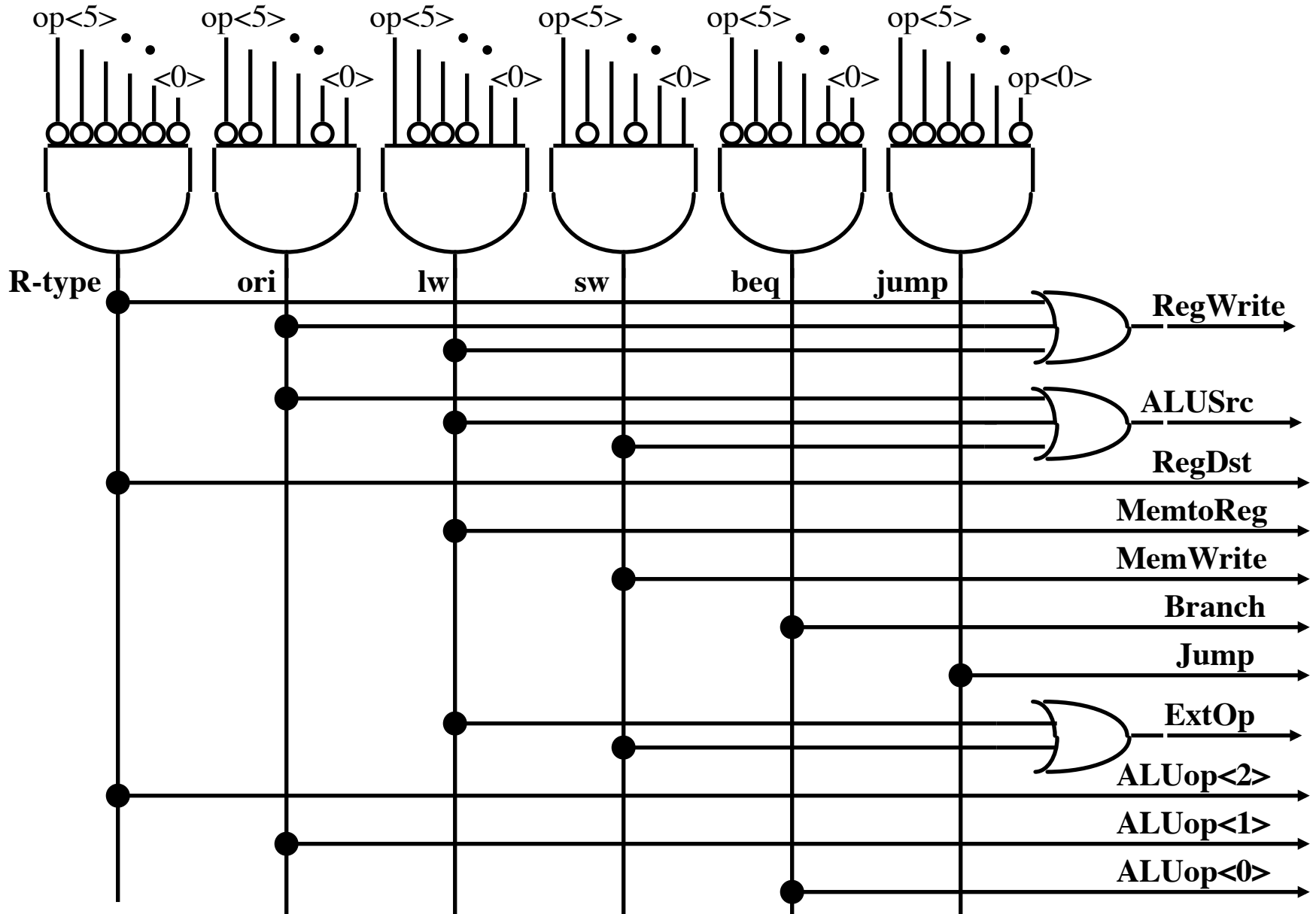
$= \text{!op<5> \& !op<4> \& !op<3> \& !op<2> \& !op<1> \& !op<0>}$  (R-type)

$+ \text{!op<5> \& !op<4> \& op<3> \& op<2> \& !op<1> \& op<0>}$  (ori)

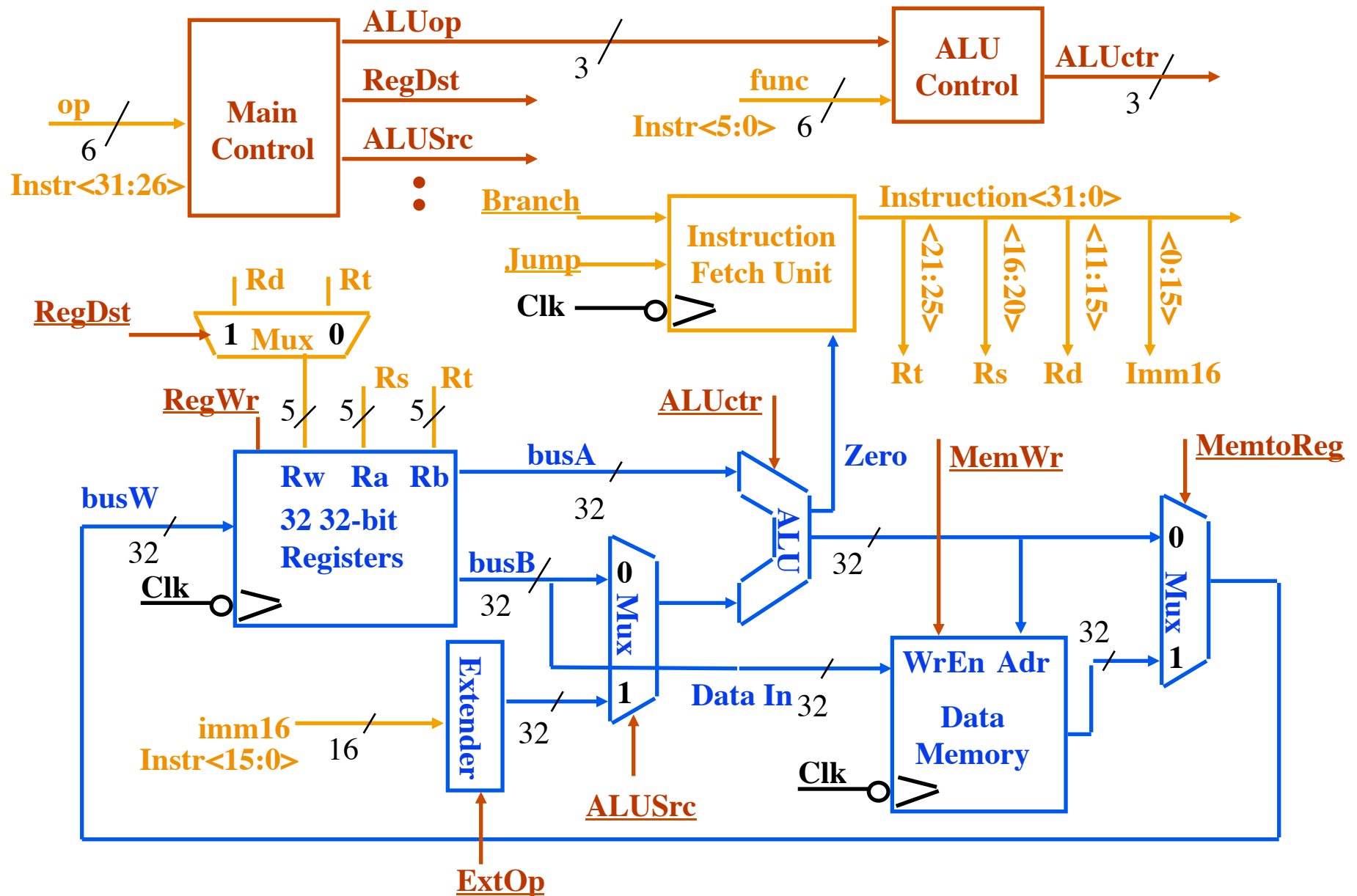
$+ \text{op<5> \& !op<4> \& !op<3> \& !op<2> \& op<1> \& op<0>}$  (lw)



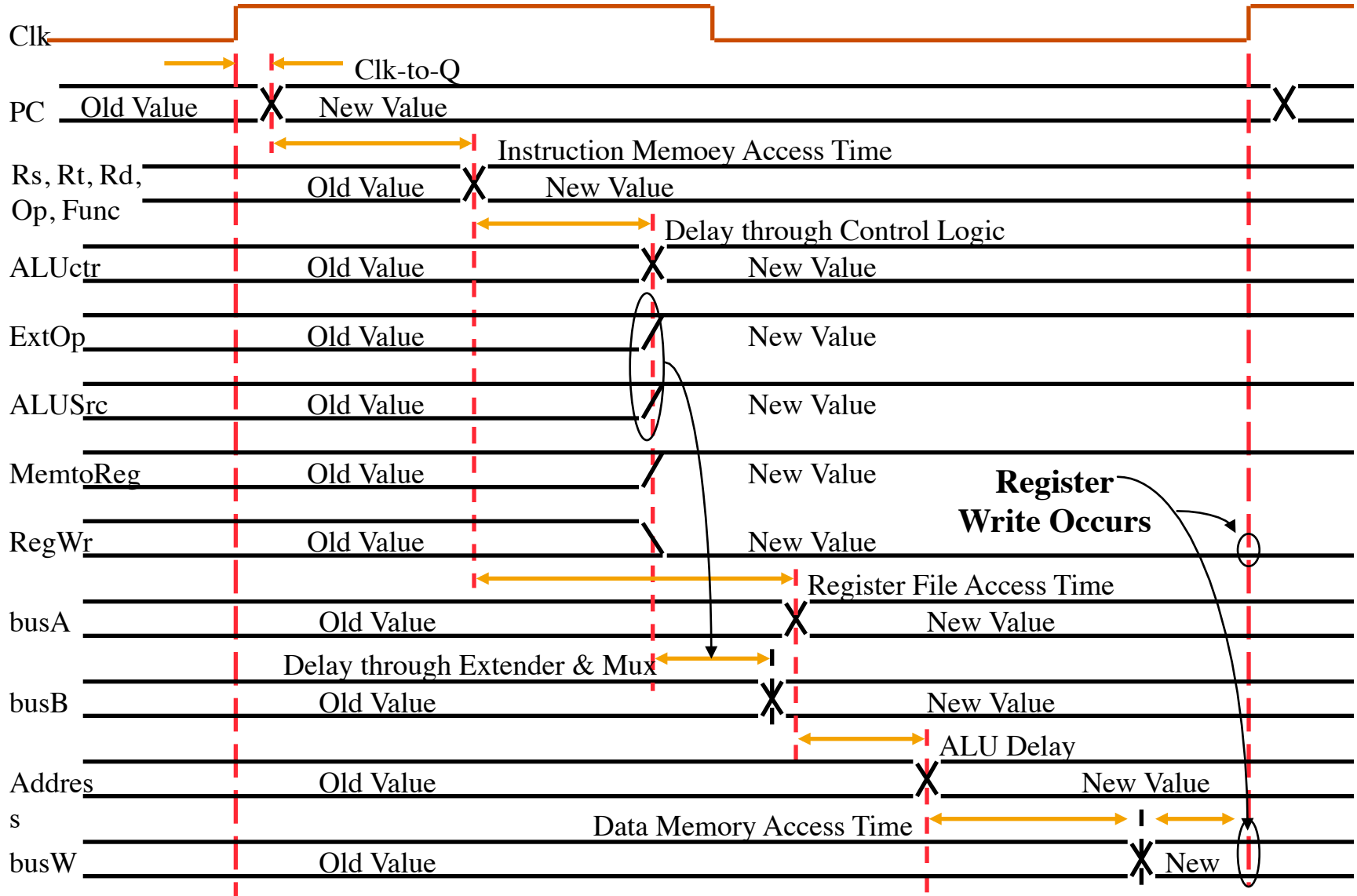
# PLA Implementation of the Main Control



# Putting it All Together: A Single Cycle Processor



# Worst Case Timing: lw \$1, \$2(offset)



# Drawback of this Single Cycle Processor

- **Long cycle time:**

- ◆ **Cycle time must be long enough for the load instruction:**

- PC's Clock -to-Q +
    - Instruction Memory Access Time +
    - Register File Access Time +
    - ALU Delay (address calculation) +
    - Data Memory Access Time +
    - Register File Setup Time +
    - Clock Skew

- **Cycle time is much longer than needed for all other instructions**