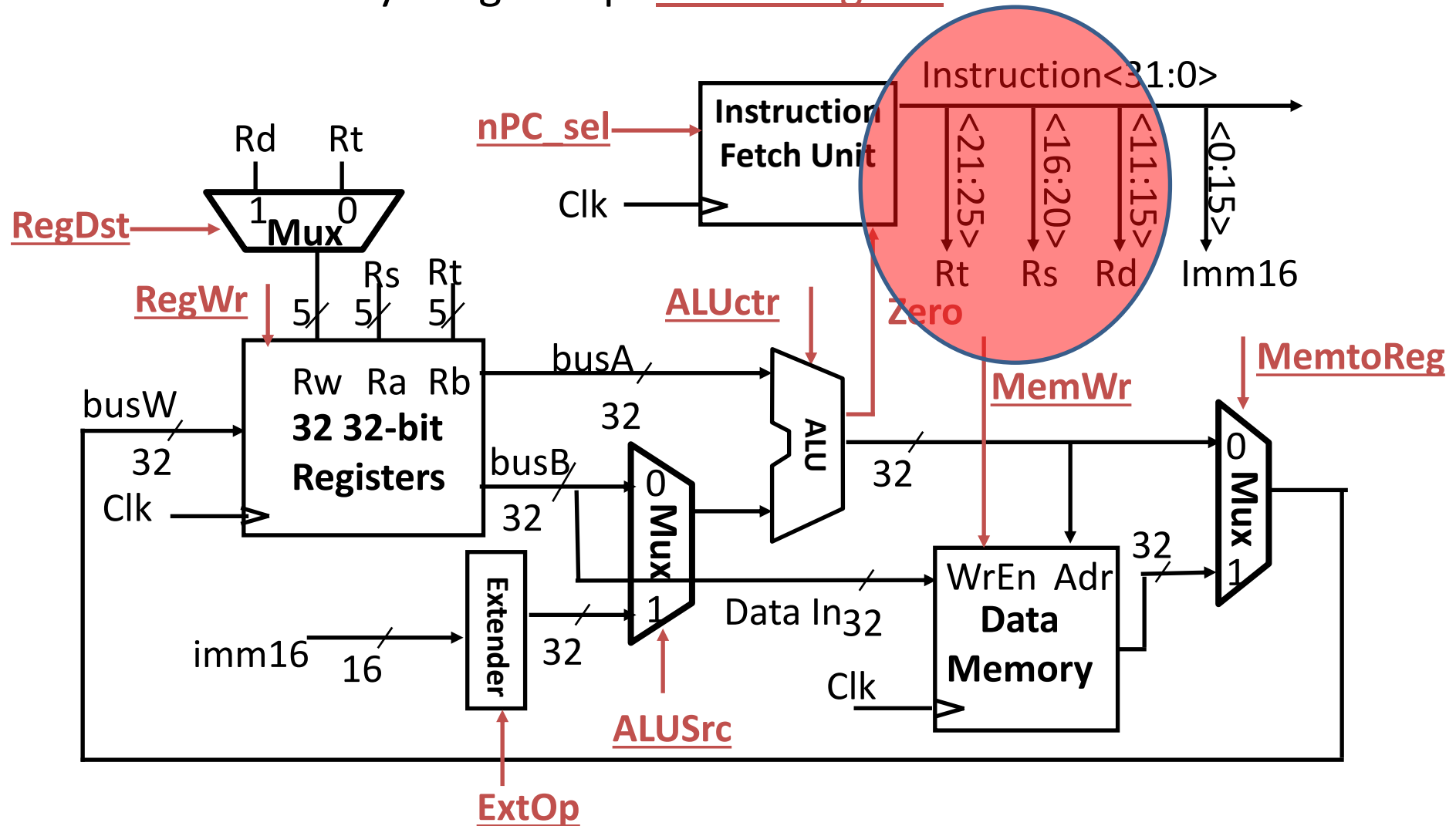


# Single Cycle CPU Control

COMP 273

# Summary: A Single Cycle Datapath

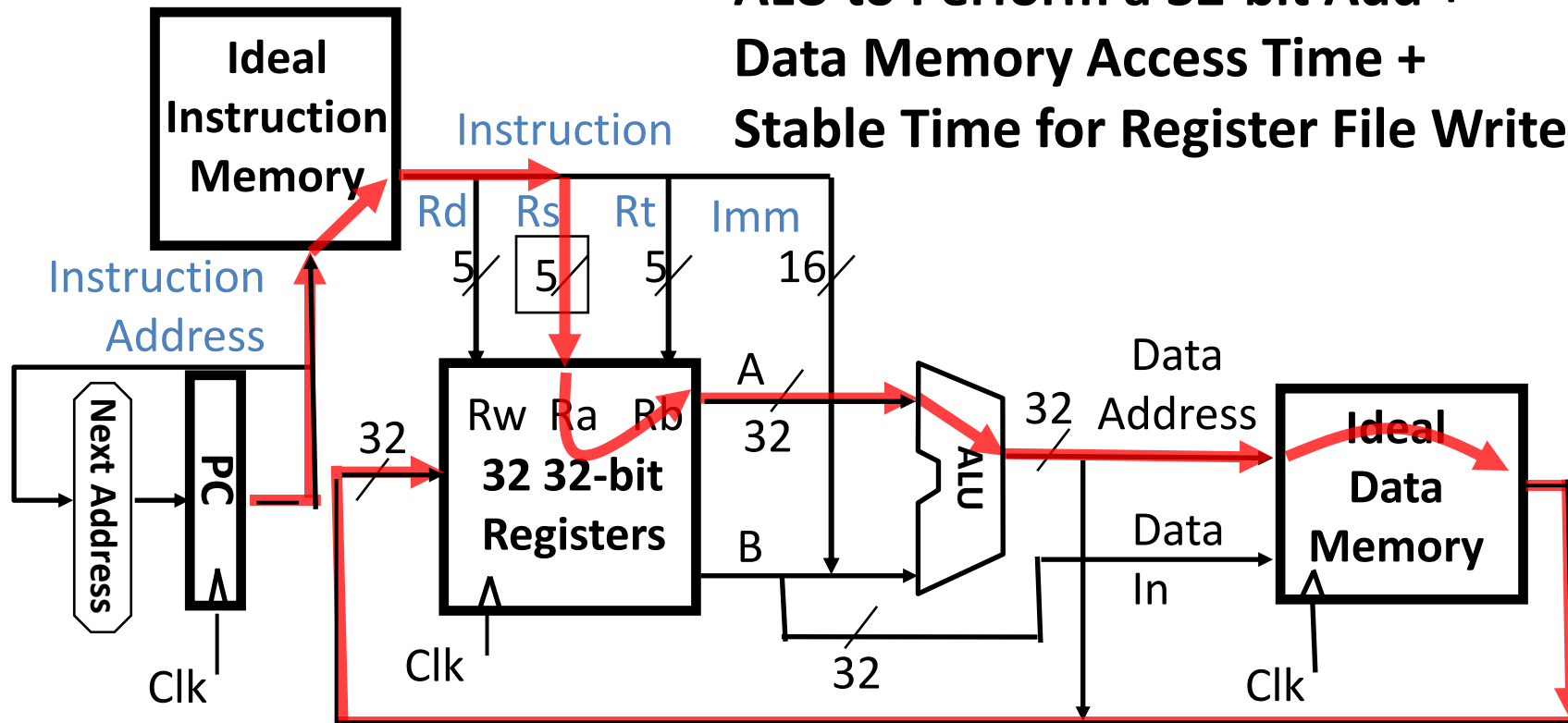
- Rs, Rt, Rd, Immed16 connected to datapath
- We have everything except control signals



# An Abstract View of the Critical Path

This affects how much you can overclock your PC!

Critical Path (Load Operation) =  
Delay clock through PC (FFs) +  
Instruction Memory's Access Time +  
Register File's Access Time, +  
ALU to Perform a 32-bit Add +  
Data Memory Access Time +  
Stable Time for Register File Write



# Recap: Meaning of the Control Signals

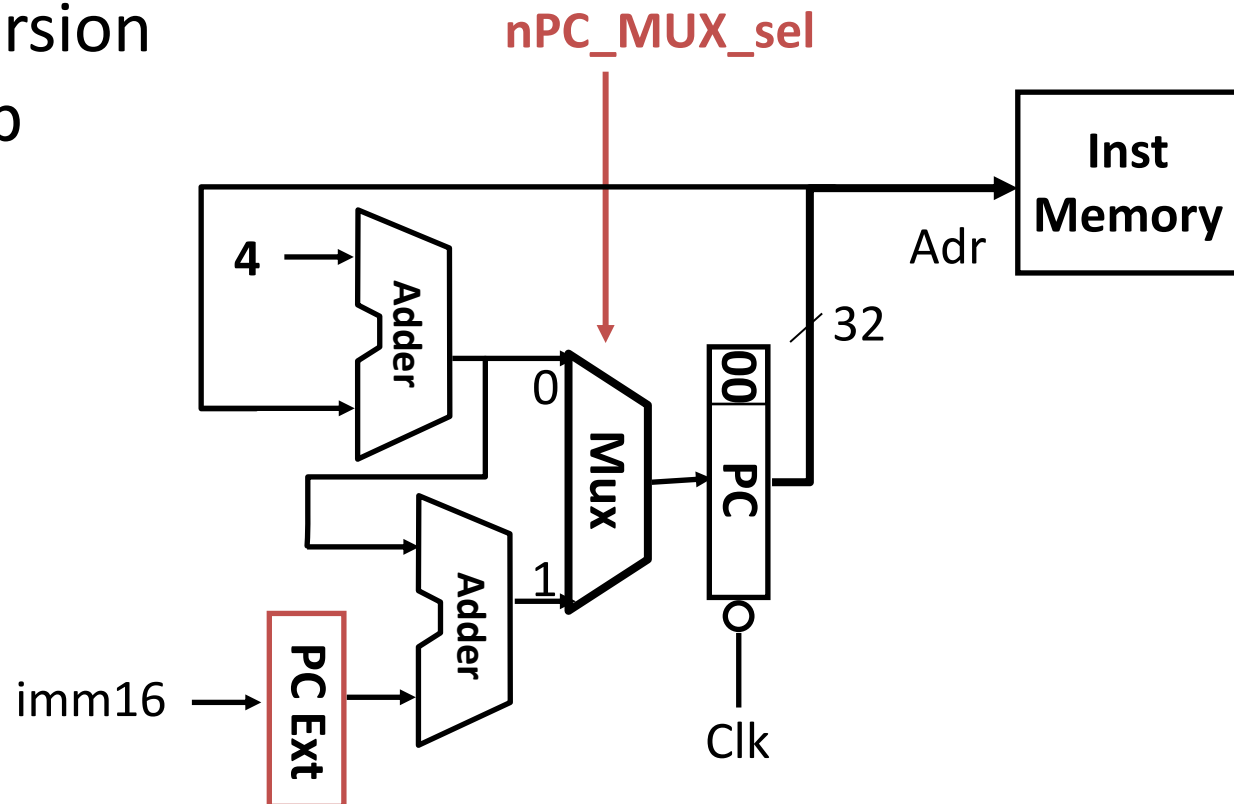
- $nPC\_MUX\_sel$ :

“n”=next

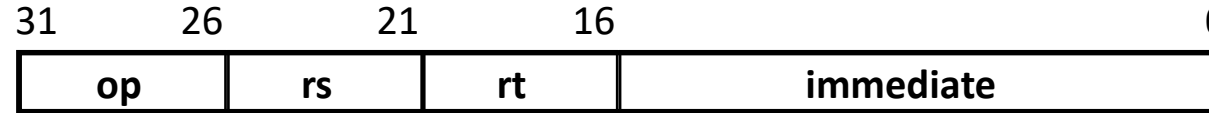
$$0 \Rightarrow PC \leftarrow PC + 4$$

$$1 \Rightarrow PC \leftarrow PC + 4 + \{SignExt(Im16), 00\}$$

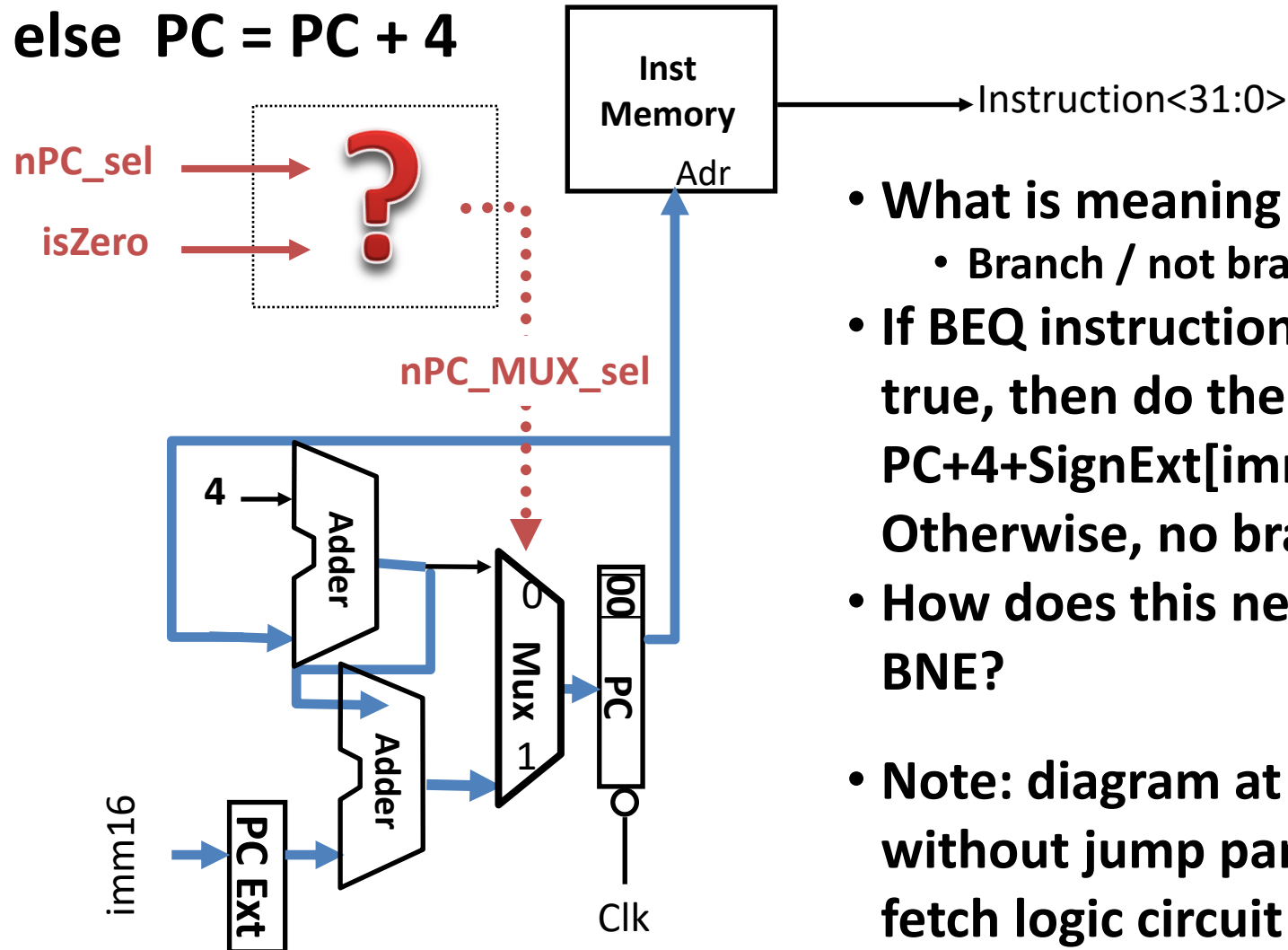
- This is the version without jump instructions



# Instruction Fetch Unit at the End of **BRANCH** (Ex. BEQ)



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



- What is meaning of nPC\_sel?
  - Branch / not branch instruction
- If BEQ instruction AND isZero are true, then do the branch (i.e.,  $PC + 4 + \text{SignExt}[\text{imm16}] * 4$ ). Otherwise, no branch (i.e.,  $PC + 4$ )
- How does this need to change for BNE?
- Note: diagram at left shown without jump part of instruction fetch logic circuit

# Recap: Meaning of the Control Signals

**ExtOp:** 0  $\Rightarrow$  “zero”;

1  $\Rightarrow$  “sign”

**ALUsrc:** 0  $\Rightarrow$  regB;

1  $\Rightarrow$  immedi

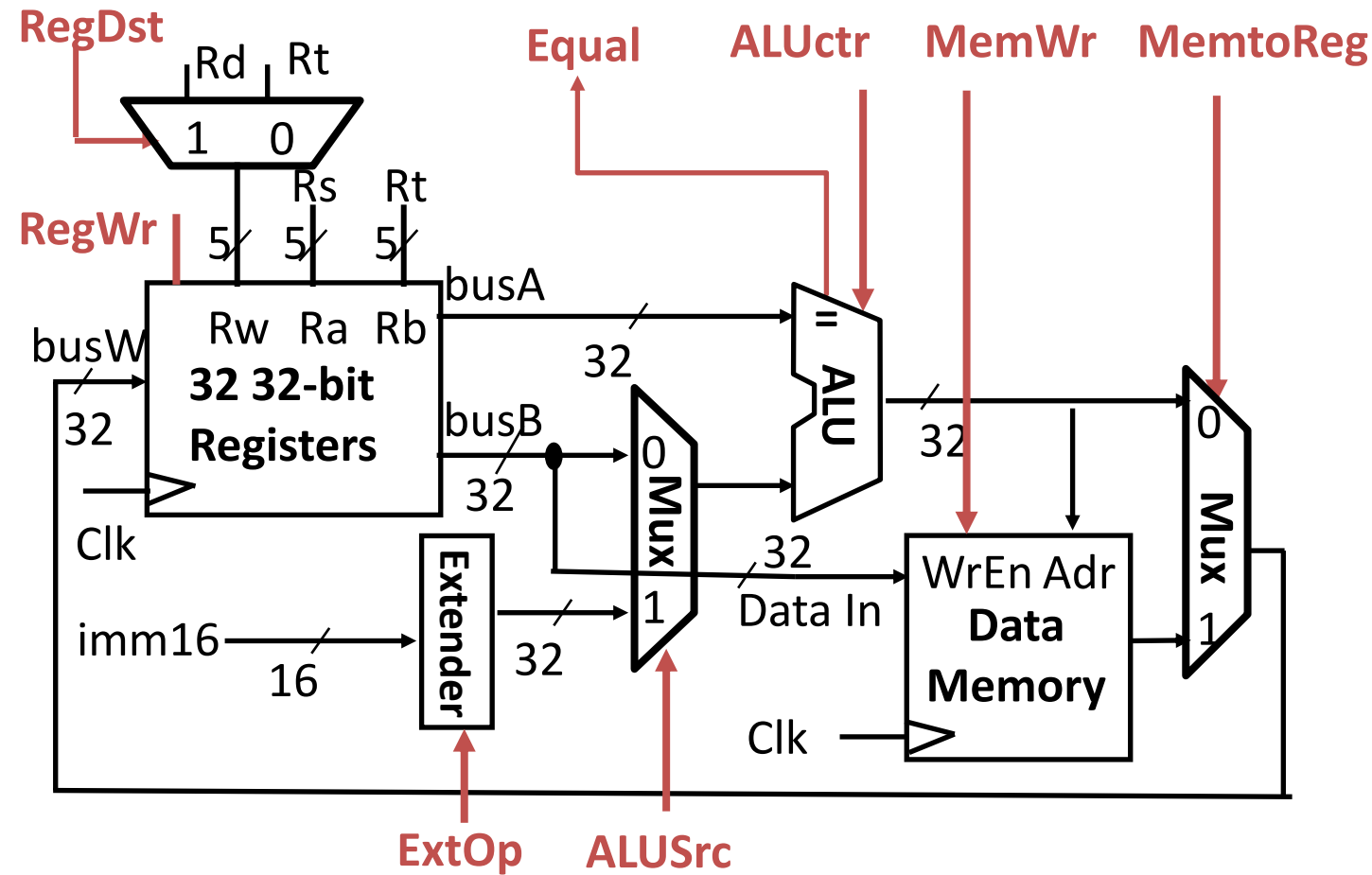
**ALUctr:** “add”, “sub”, “or”

**MemWr:** 1  $\Rightarrow$  write memory

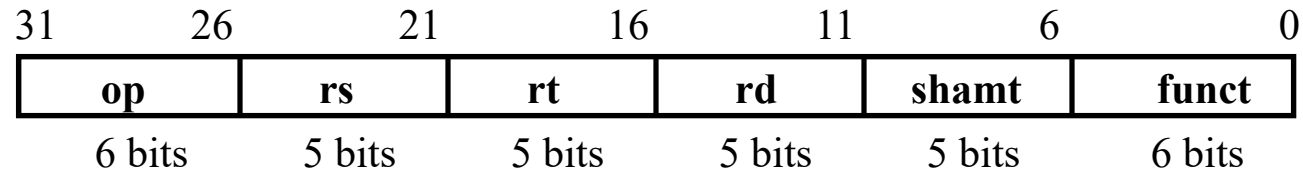
**MemtoReg:** 0  $\Rightarrow$  ALU; 1  $\Rightarrow$  Mem

**RegDst:** 0  $\Rightarrow$  “rt”; 1  $\Rightarrow$  “rd”

**RegWr:** 1  $\Rightarrow$  write register



# RTL: The **ADD** Instruction



**add rd, rs, rt**

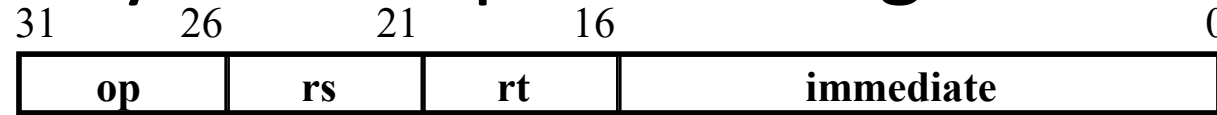
- MEM[PC]
  - Fetch the instruction from memory
- $R[rd] = R[rs] + R[rt]$ 
  - The actual operation
- $PC = PC + 4$ 
  - Calculate the next instruction's address

31	26	21	16	11	6	0
<b>op</b>	<b>rs</b>	<b>rt</b>	<b>rd</b>	<b>shamt</b>	<b>funct</b>	

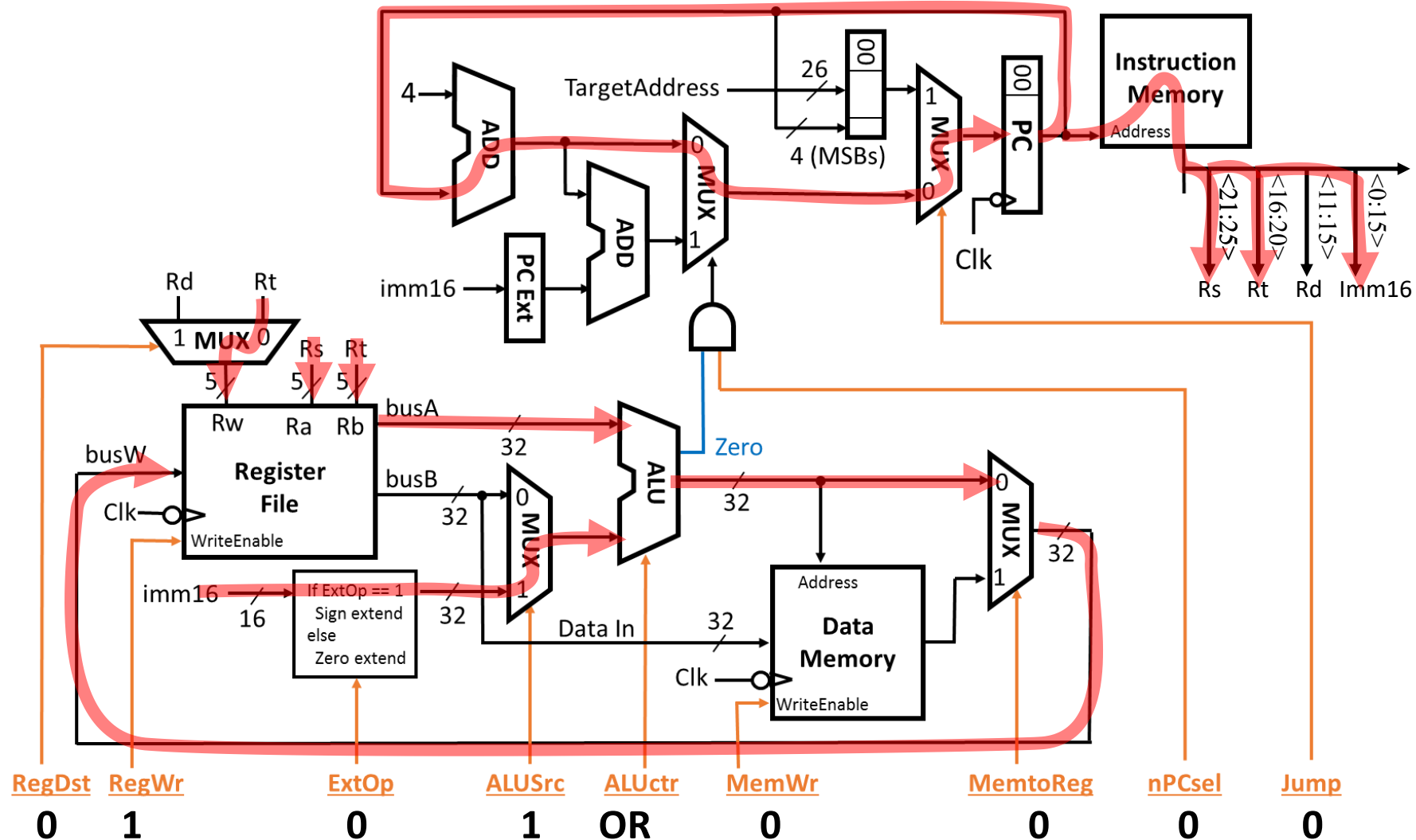
[illegible]



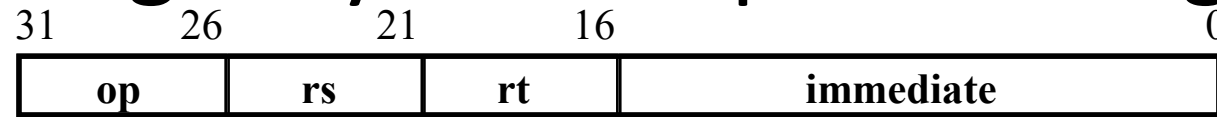
# Single Cycle Datapath during OR Immediate?



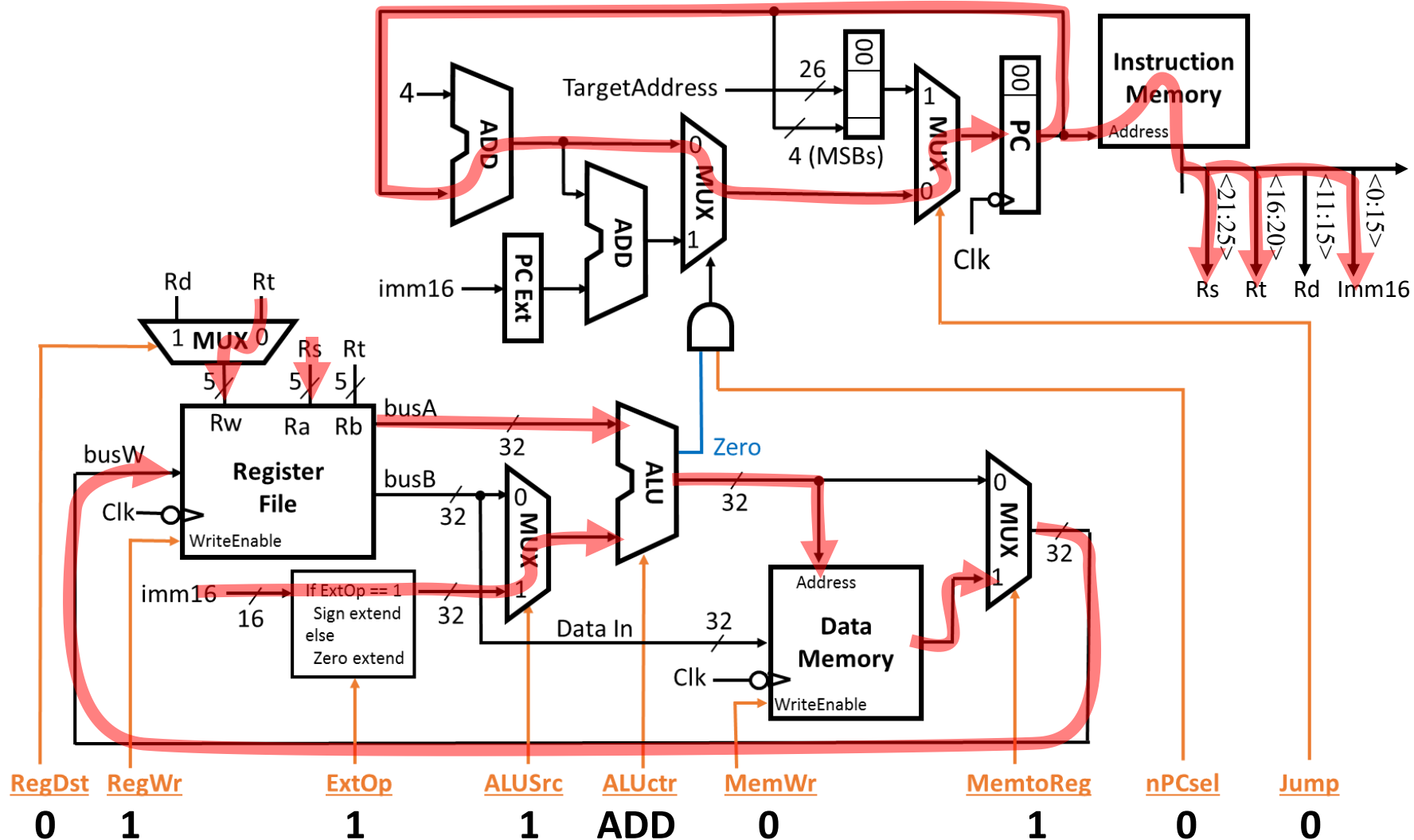
$$R[rt] = R[rs] \text{ OR } \text{ZeroExt}[\text{Imm16}]$$



# The Single Cycle Datapath during **LOAD**?



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

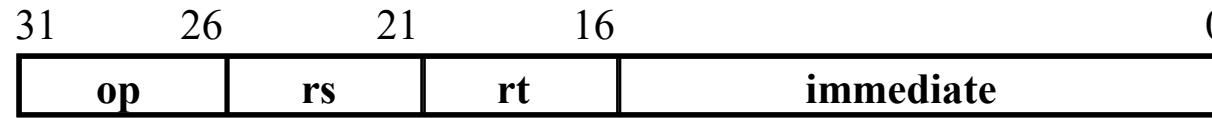


31	26	21	16	0
<b>op</b>	<b>rs</b>	<b>rt</b>	<b>immediate</b>	

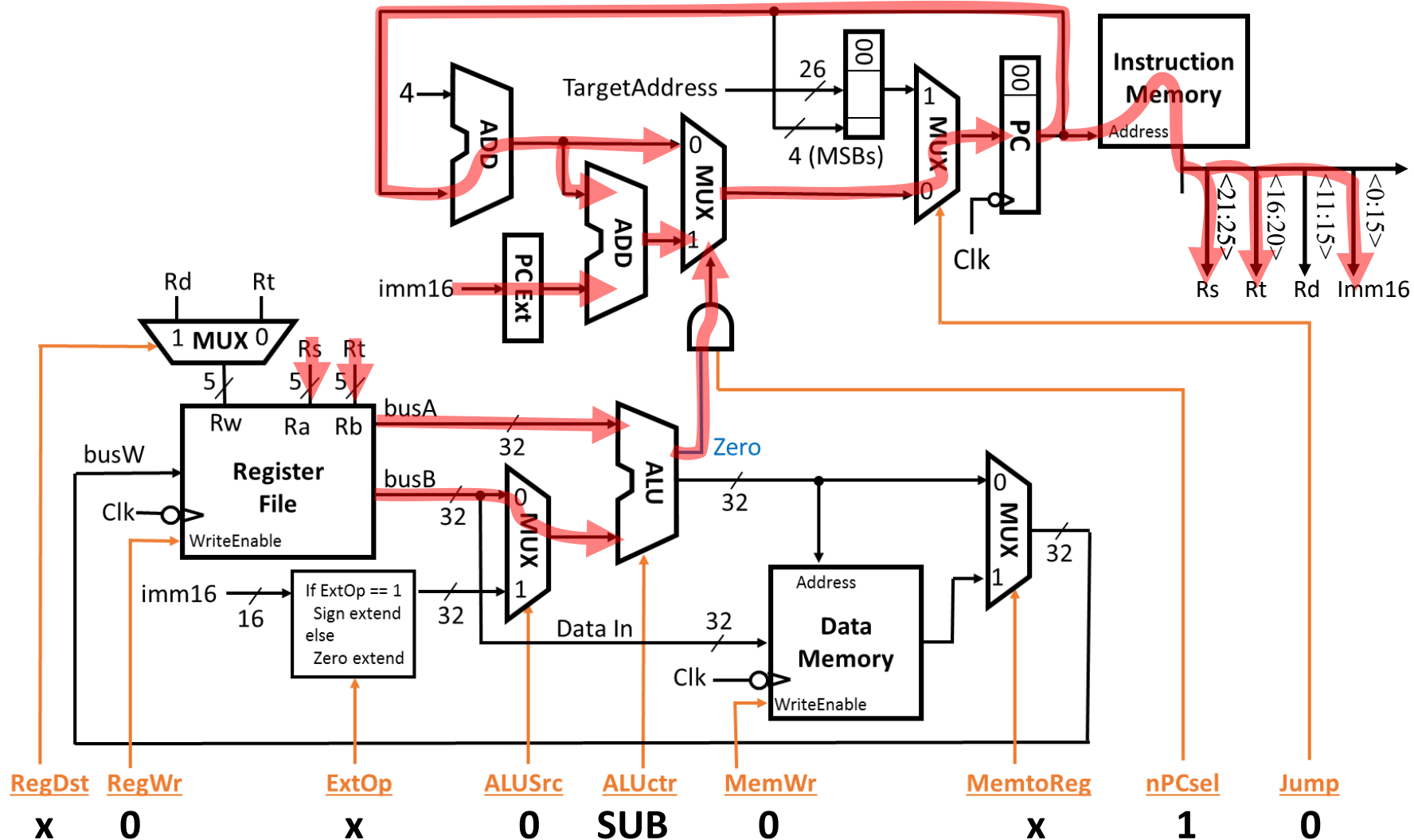
The diagram illustrates the internal structure of a processor, showing the flow of data and control signals between various components:

- Register File:** Receives `RegDst` and `RegWr` control signals. It outputs `busA` and `busB` to the ALU. It also receives `imm16` and `ExtOp` control signals. The `WriteEnable` signal is connected to the `MemWr` control signal.
- ALU:** Receives `busA` and `busB` from the Register File. It outputs a 32-bit result to the ALU MUX. The `Zero` flag is output from the ALU.
- MUXes:**
  - ALU MUX:** Selects between the ALU result and the `imm16` value (extended) based on the `ALUSrc` control signal.
  - PC MUX:** Selects between the ALU result and the `imm16` value (extended) based on the `ALUctr` control signal.
  - Instruction Memory MUX:** Selects between the ALU result and the `imm16` value (extended) based on the `MemWr` control signal.
  - Data Memory MUX:** Selects between the ALU result and the `imm16` value (extended) based on the `MemtoReg` control signal.
- PC (Program Counter):** Receives the output of the PC MUX and outputs the `TargetAddress` to the Instruction Memory.
- Instruction Memory:** Receives the `TargetAddress` and outputs the instruction to the Instruction Memory MUX.
- Data Memory:** Receives the output of the Data Memory MUX and outputs the data to the Data Memory MUX.
- Control Signals:**
  - `RegDst`: `x`
  - `RegWr`: `0`
  - `ExtOp`: `1`
  - `ALUSrc`: `1`
  - `ALUctr`: `ADD`
  - `MemWr`: `1`
  - `MemtoReg`: `x`
  - `nPCsel`: `0`
  - `Jump`: `0`

# The Single Cycle Datapath during **BRANCH** (Ex. BEQ)



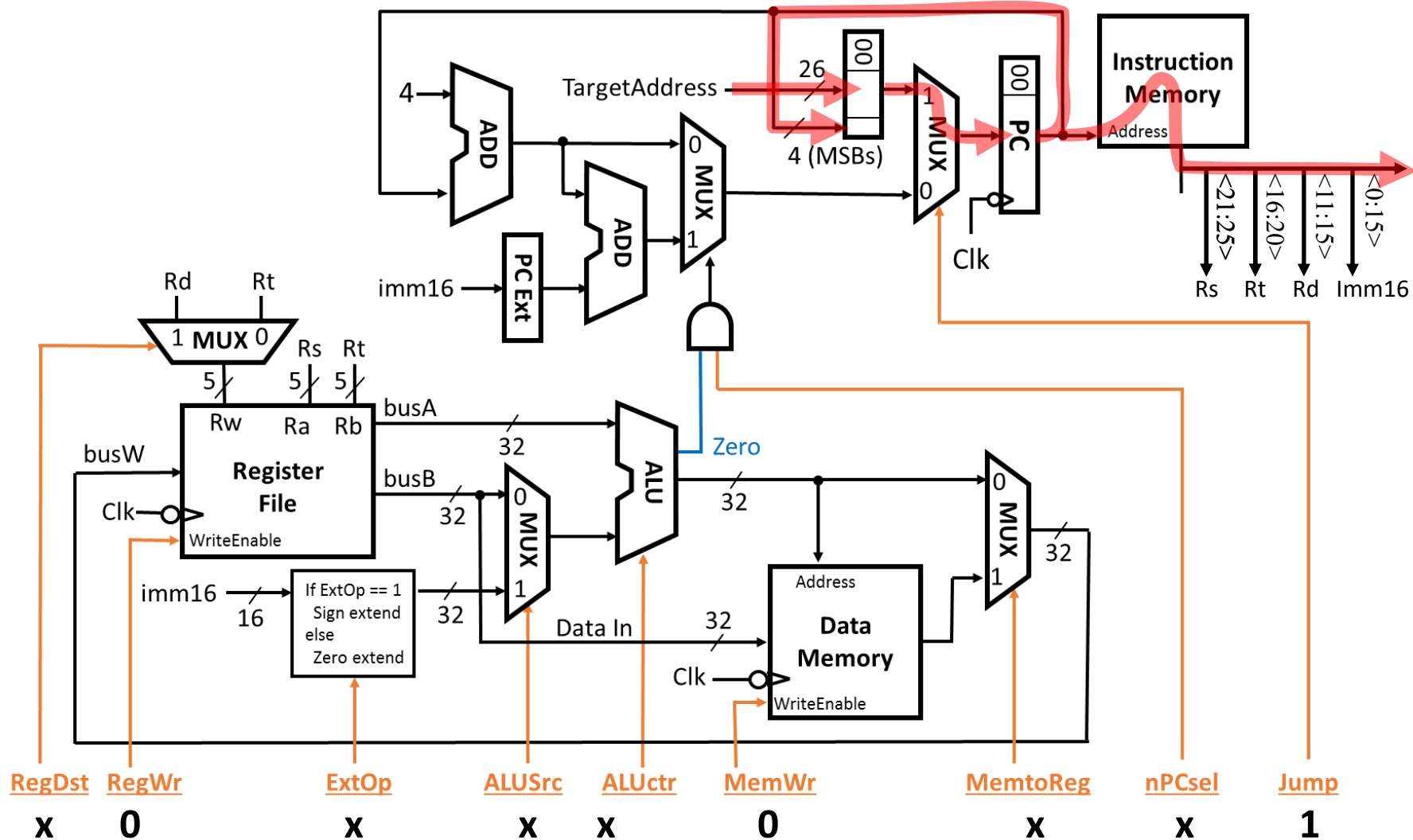
if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



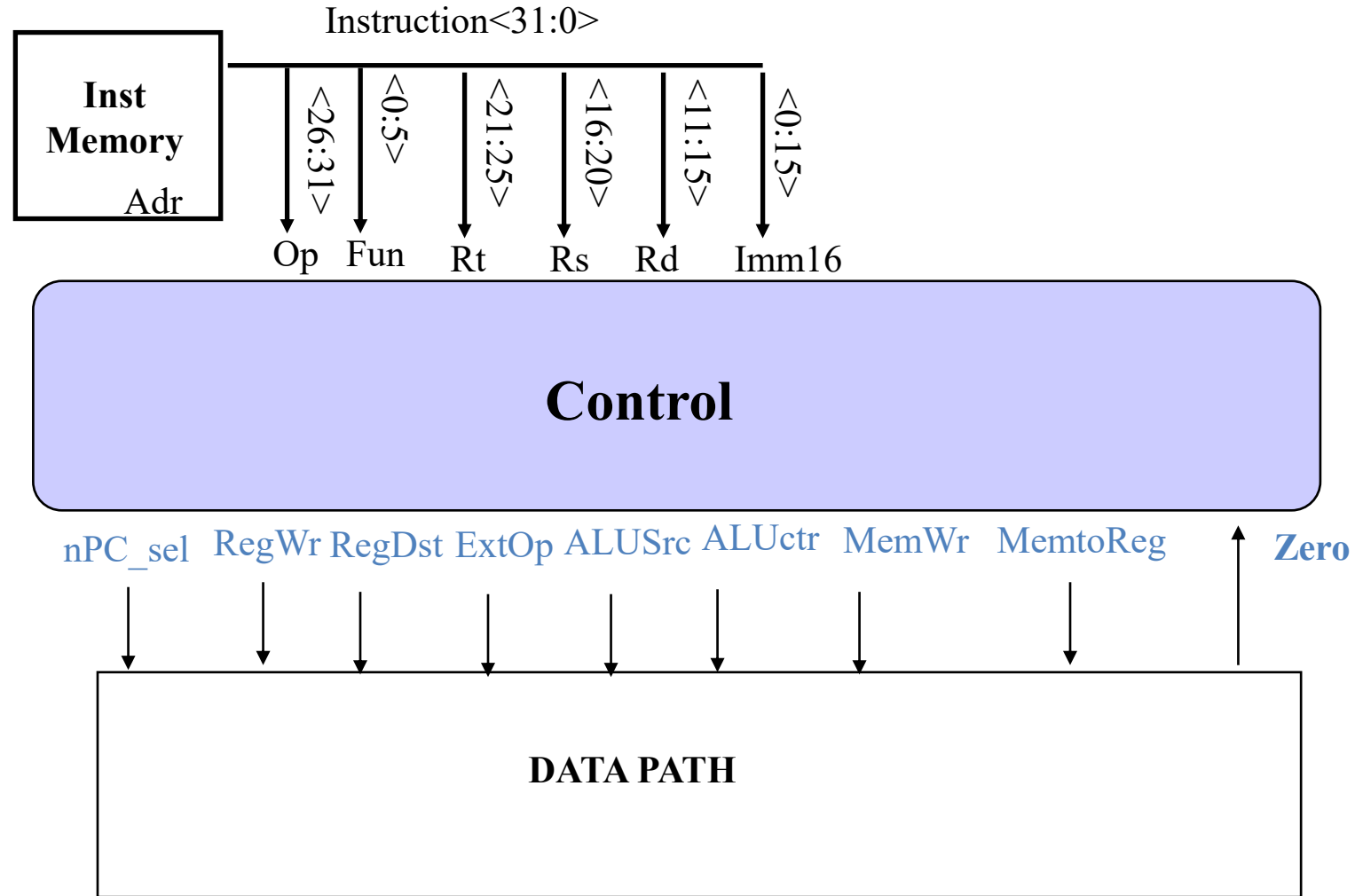
# The Single Cycle Datapath during JUMP (J)



**New PC = { PC[31..28], target address, 00 }**



# Step 4: Given Datapath: RTL $\Rightarrow$ Control



# A Summary of the Control Signals

See Appendix A.10 (MIPS  
Assembly Language)  
(or green reference sheet)

See 4.4 (A Simple  
Implementation Scheme),  
and Appendix B.5  
(Constructing a Basic  
Arithmetic Logic Unit)

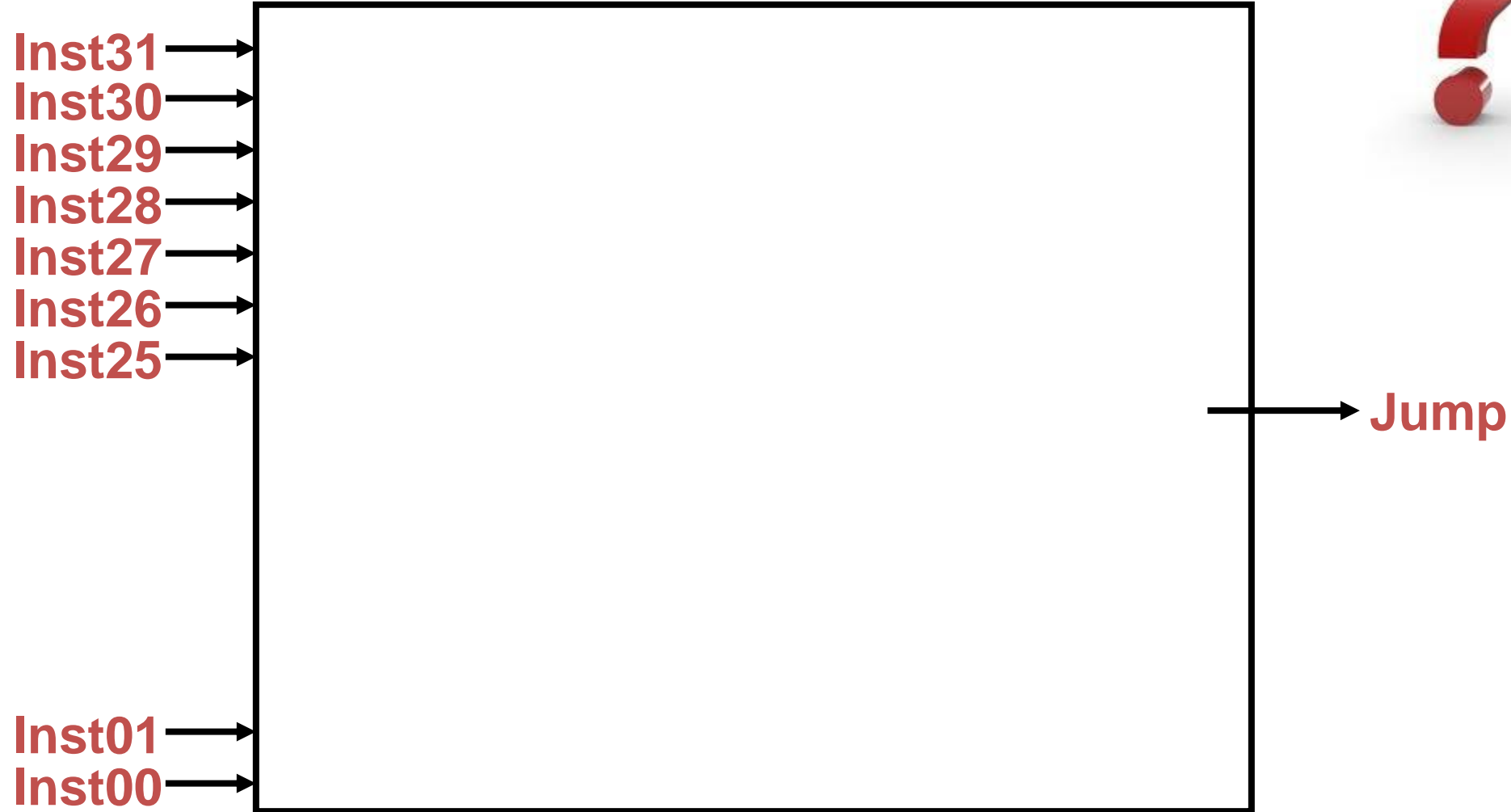
<div><div>func</div><div>op</div></div>	10 0000	10 0010	We Don't Care !				
	00 0000	00 0000	00 1101	10 0011	10 1011	00 0100	00 0010
	add	sub	ori	lw	sw	beq	jump
RegDst	1	1	0	0	x	x	x
ALUSrc	0	0	1	1	1	0	x
MemtoReg	0	0	0	1	x	x	x
RegWrite	1	1	1	1	0	0	0
MemWrite	0	0	0	0	1	0	0
nPCsel	0	0	0	0	0	1	x
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
→ ALUctr<3:0>	Add	Subtract	Or	Add	Add	Subtract	x

Inputs

Outputs

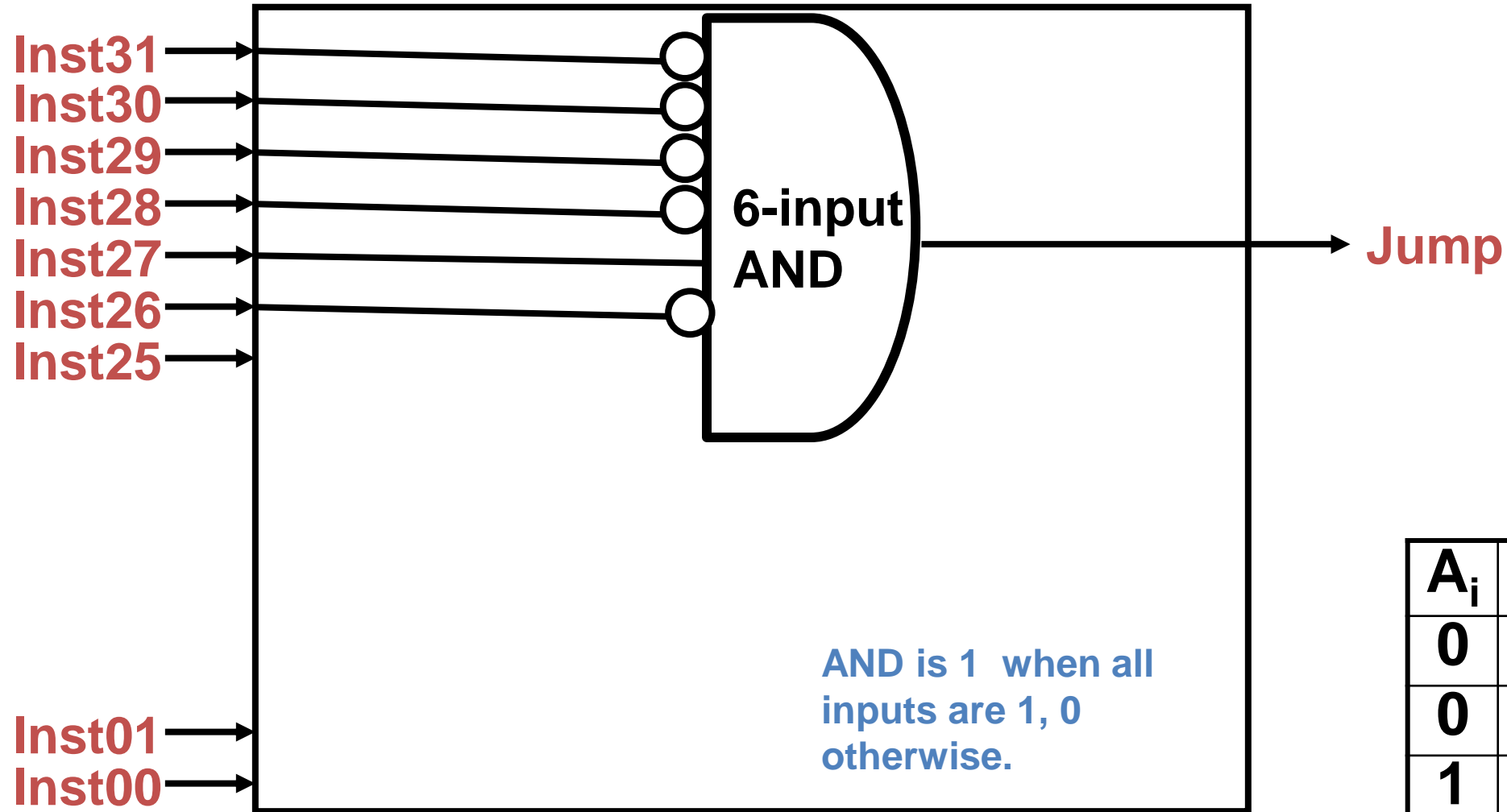
	31	26	21	16	11	6	0						
R-type	op		rs		rt		rd		shamt		funct		add, sub
I-type	op		rs		rt		immediate						ori, lw, sw, beq
J-type	op		target address										jump

# Question: Build Control Logic to implement Jump



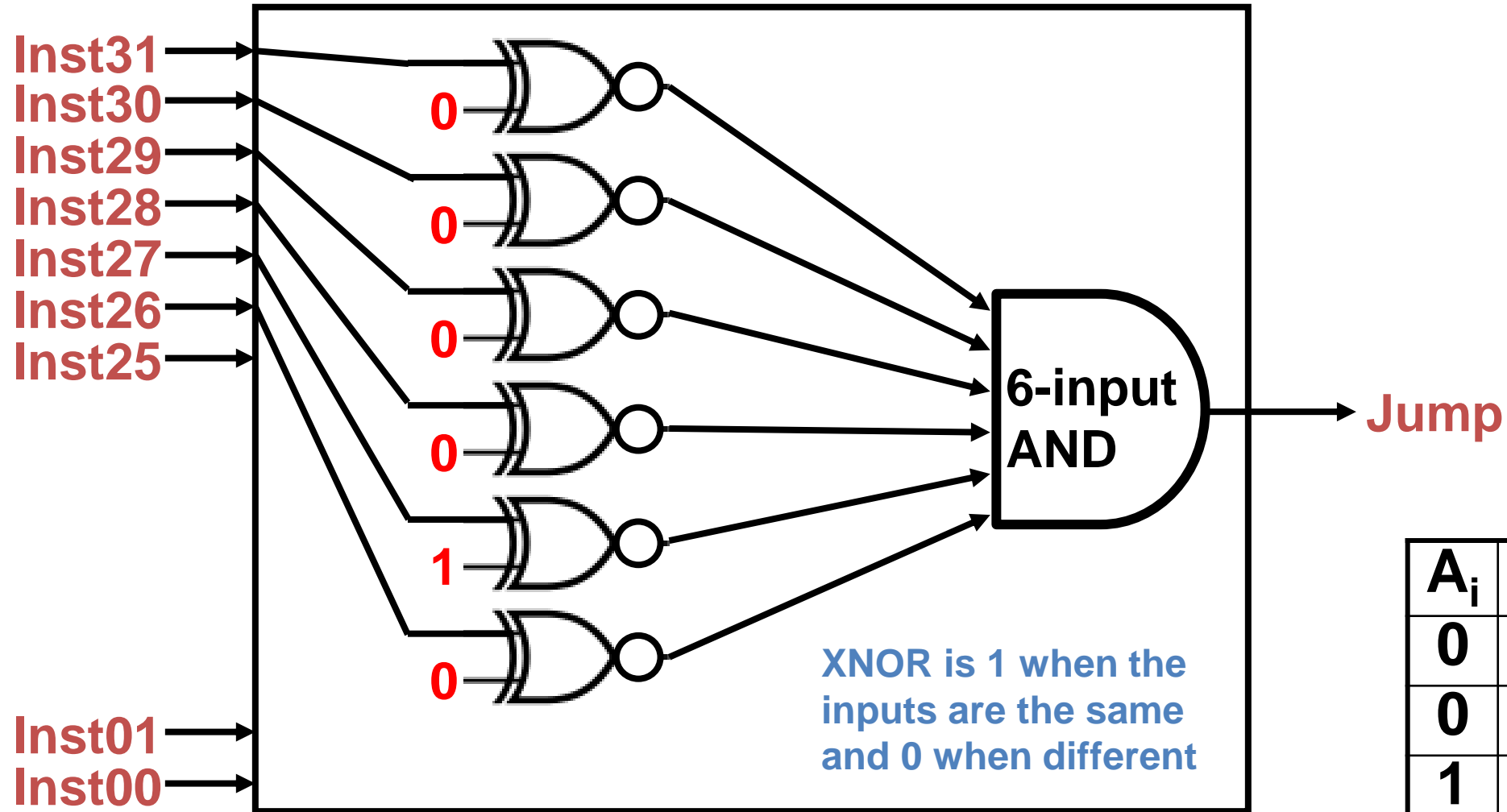


# Control Logic to implement Jump



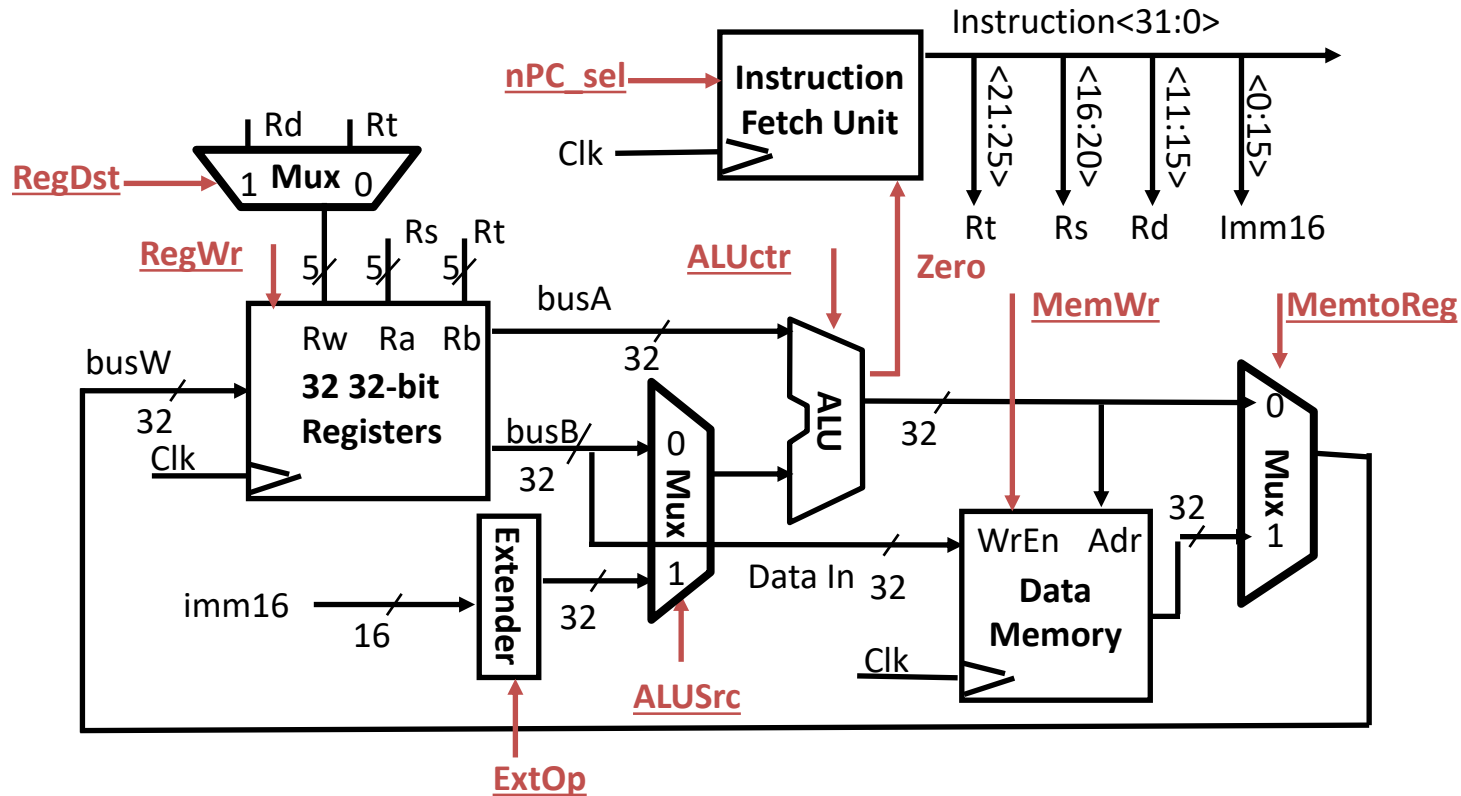
A <sub>i</sub>	B <sub>i</sub>	AND
0	0	0
0	1	0
1	0	0
1	1	1

# Control Logic to implement Jump



A <sub>i</sub>	B <sub>i</sub>	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

# Question



**ADD:**  $R[rd] = R[rs] + R[rt]$ ;  $PC = PC + 4$   
**SUB:**  $R[rd] = R[rs] - R[rt]$ ;  $PC = PC + 4$   
**ORI:**  $R[rt] = R[rs] \mid \text{zero\_ext}(\text{Imm16})$ ;  $PC = PC + 4$   
**LW:**  $R[rt] = \text{MEM}[R[rs] + \text{sign\_ext}(\text{Imm16})]$ ;  $PC = PC + 4$   
**SW:**  $\text{MEM}[R[rs] + \text{sign\_ext}(\text{Imm16})] = R[rt]$ ;  $PC = PC + 4$   
**BEQ:** if  $(R[rs] == R[rt])$  then  
            $PC = PC + 4 + (\text{sign\_ext}(\text{Imm16}) \parallel 00 \text{ (i.e., *4)})$   
       else  $PC = PC + 4$

- If MemToReg='x' & ALUctr='sub', then is the instruction **SUB** or **BEQ**?
- Which of the two signals is not the same (i.e., 1, 0, x) for ADD, LW, SW?  
**RegDst** or **ALUctr**?
- "Don't Care" signals are useful because we can simplify our implementation of the combinatorial Boolean control functions. **F** / **T**?

# Question, True or False

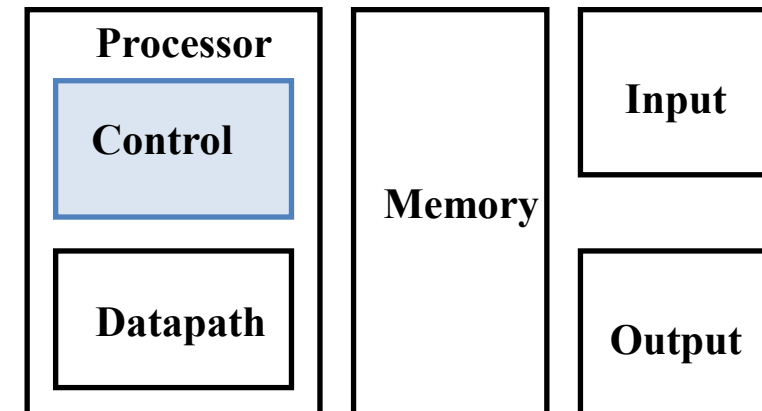


1. We should use the main ALU to compute  $PC = PC + 4$
2. The ALU is inactive for memory reads or writes.

True, false, or don't care?

# And in Conclusion... Single cycle control

- 5 steps to design a processor
  1. Analyze instruction set => datapath requirements
  2. Select set of datapath components & establish clock methodology
  3. Assemble datapath meeting the requirements
  4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
  5. Assemble the control logic
- Control is the hard part
- MIPS makes that easier
  - Instructions same size
  - Source registers always in same place
  - Immediates same size, location
  - Operations always on registers/immediates



# Review and More Information

- Textbook Section 4.4 (A Simple Implementation Scheme)