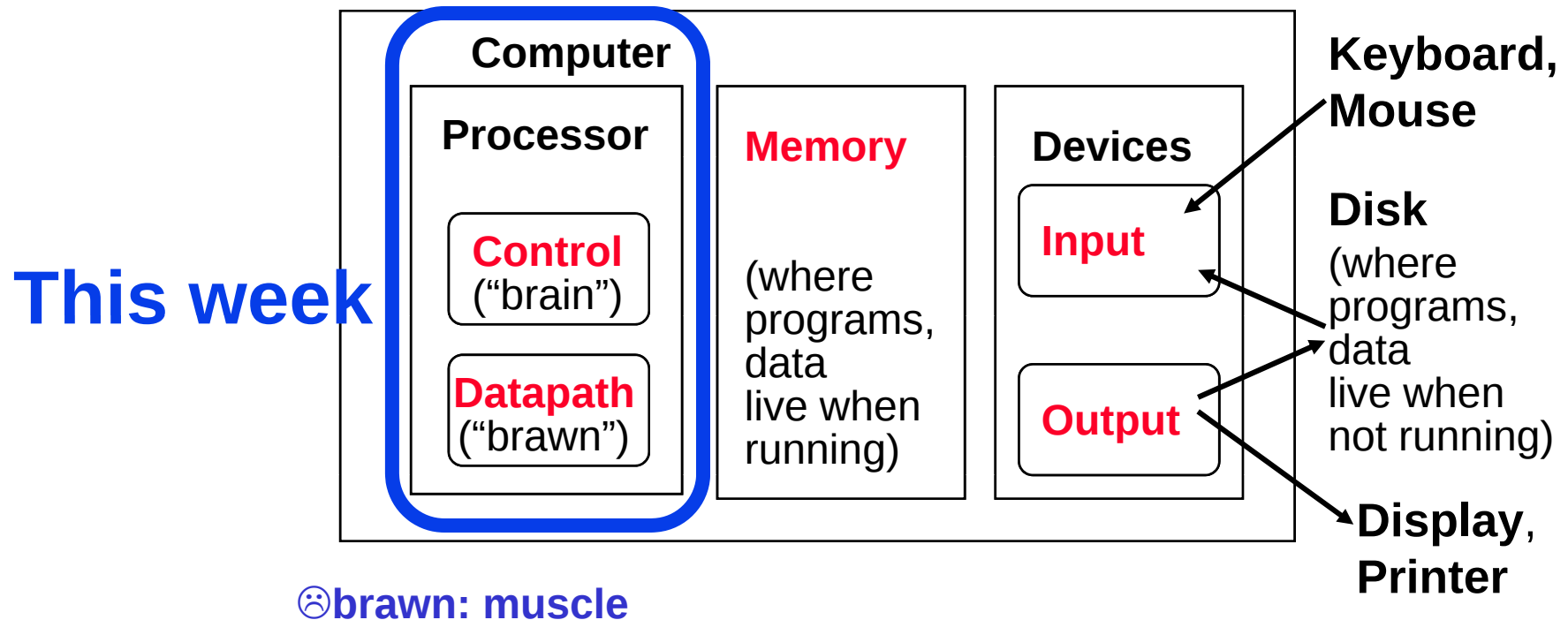
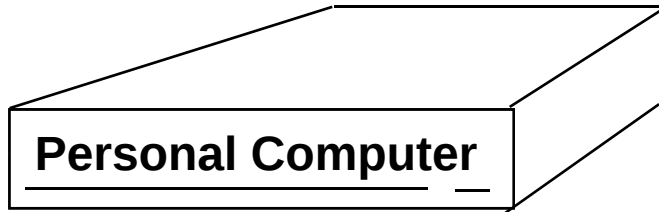


Lecture #17: CPU Design II – Control



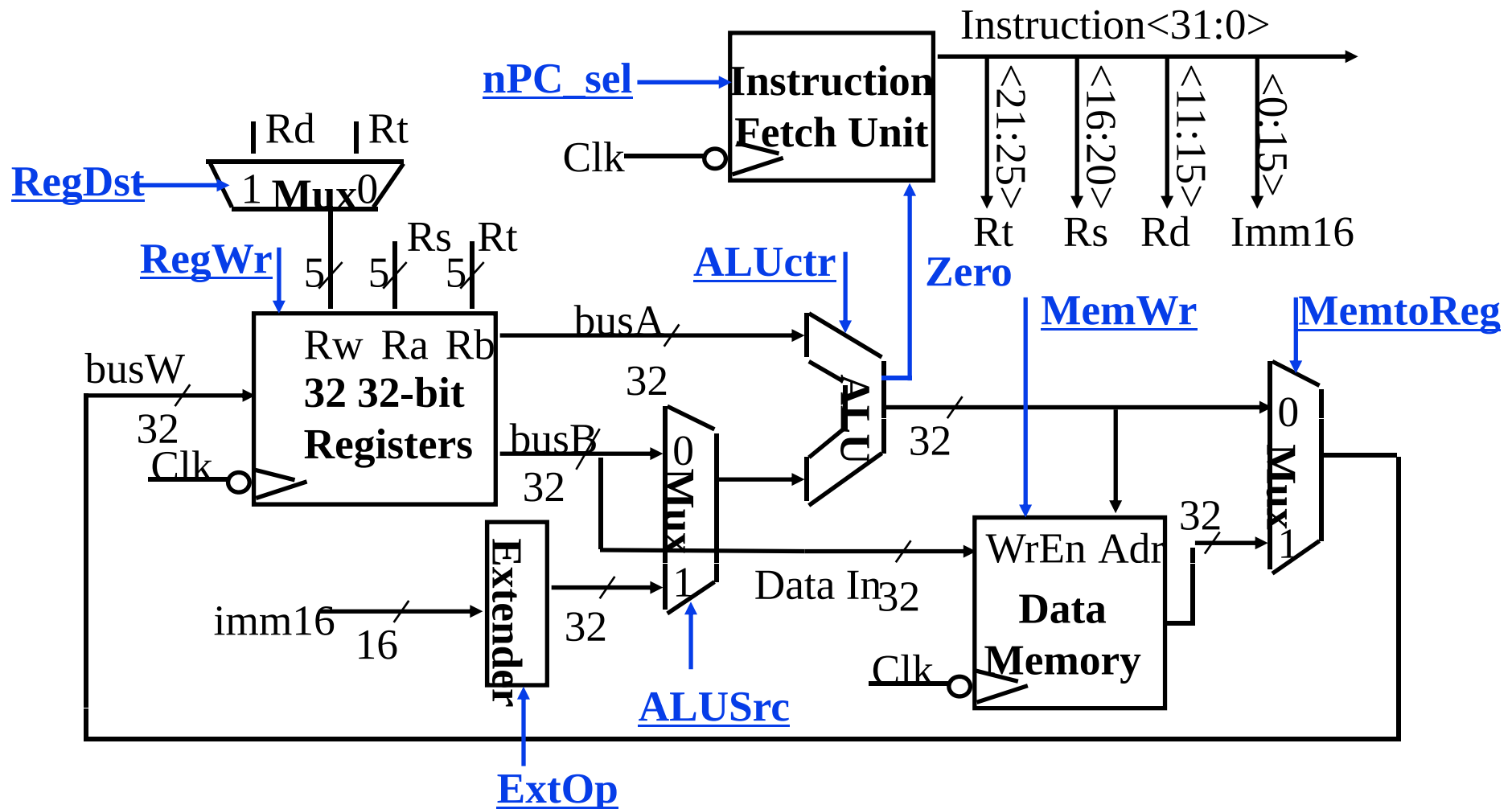
2005-07-19

Anatomy: 5 components of any Computer



Review: A Single Cycle Datapath

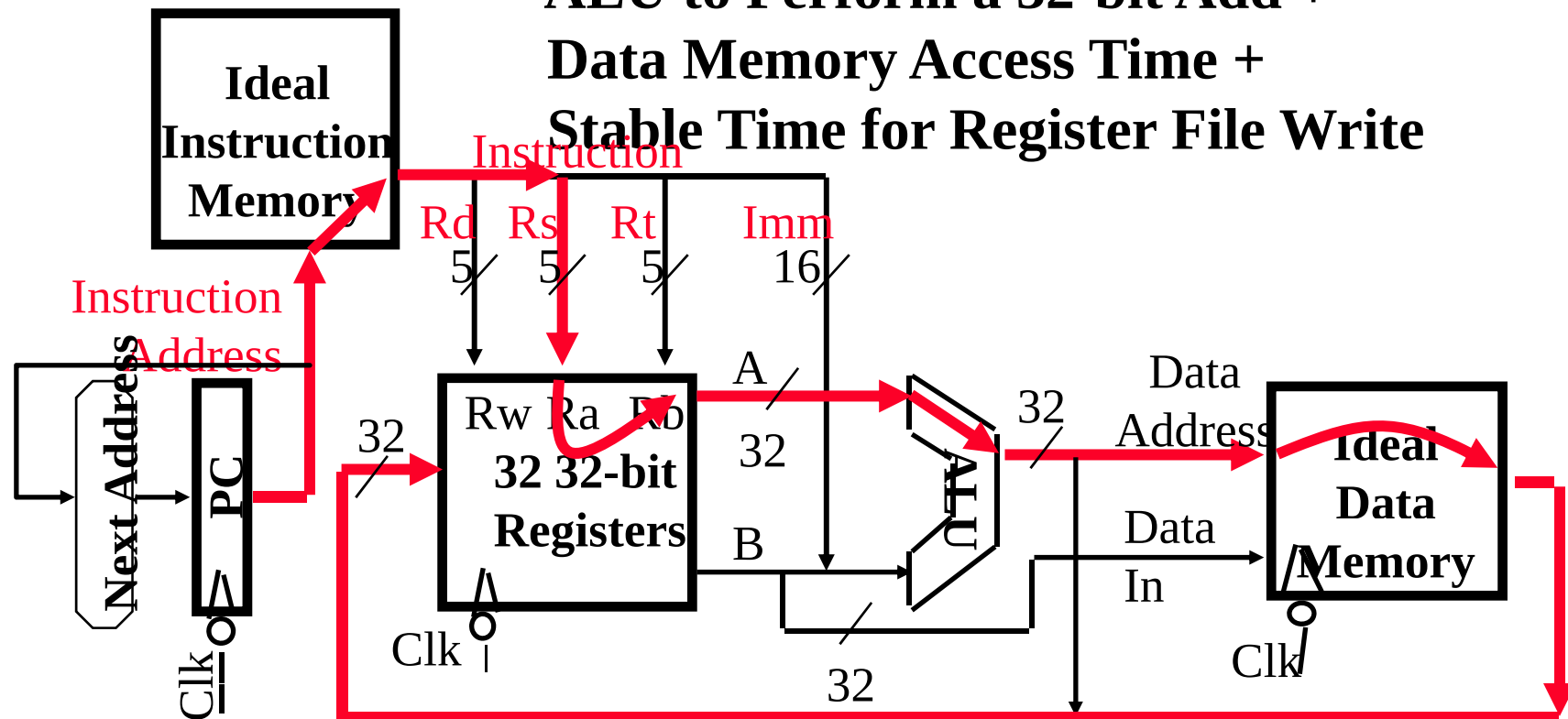
- Rs, Rt, Rd, Imed16 connected to datapath
- We have everything except control signals



An Abstract View of the Critical Path

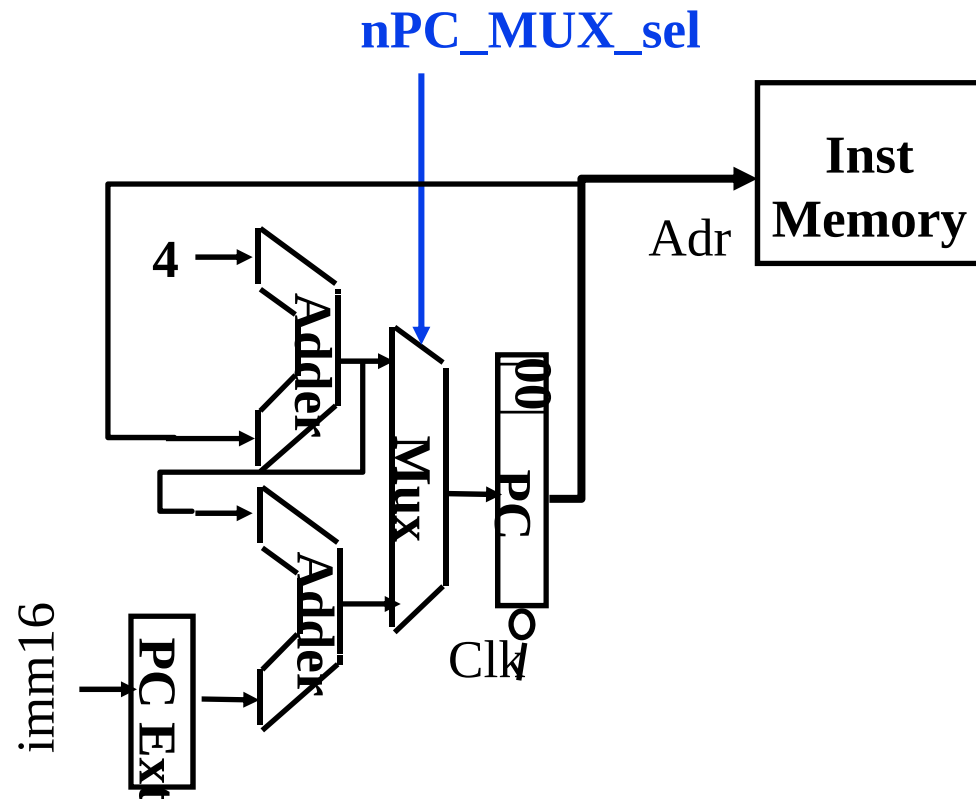
- This affects how much you can overclock your PC!

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



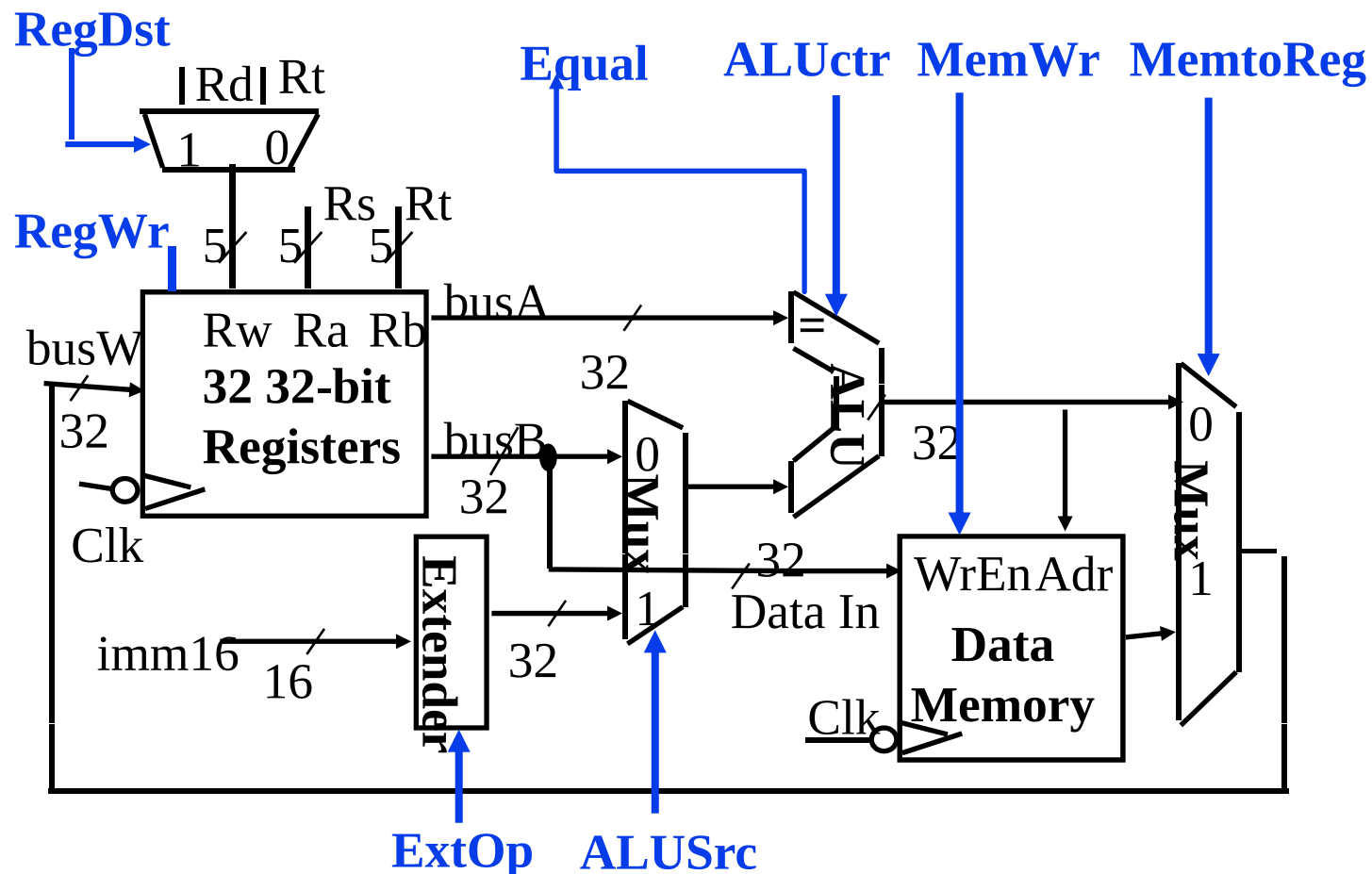
Recap: Meaning of the Control Signals

- **nPC_MUX_sel:** $0 \Rightarrow PC \leftarrow PC + 4$
 $1 \Rightarrow PC \leftarrow PC + 4 +$
 $\{ \text{SignExt(Imm16)}, 00 \}$
 “n”=next
- **Later in lecture: higher-level connection between mux and branch cond**

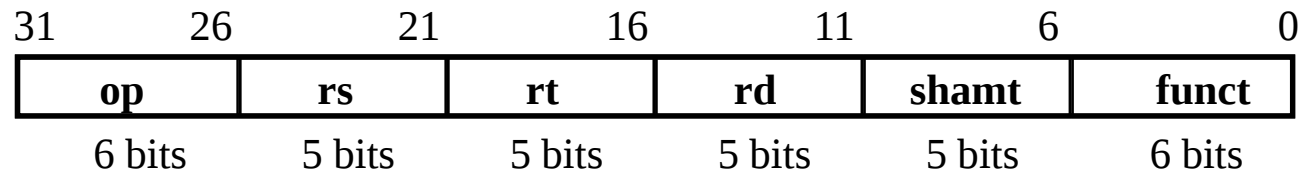


Recap: Meaning of the Control Signals

- **ExtOp:** “zero”, “sign”
Zero or sign extended
- **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



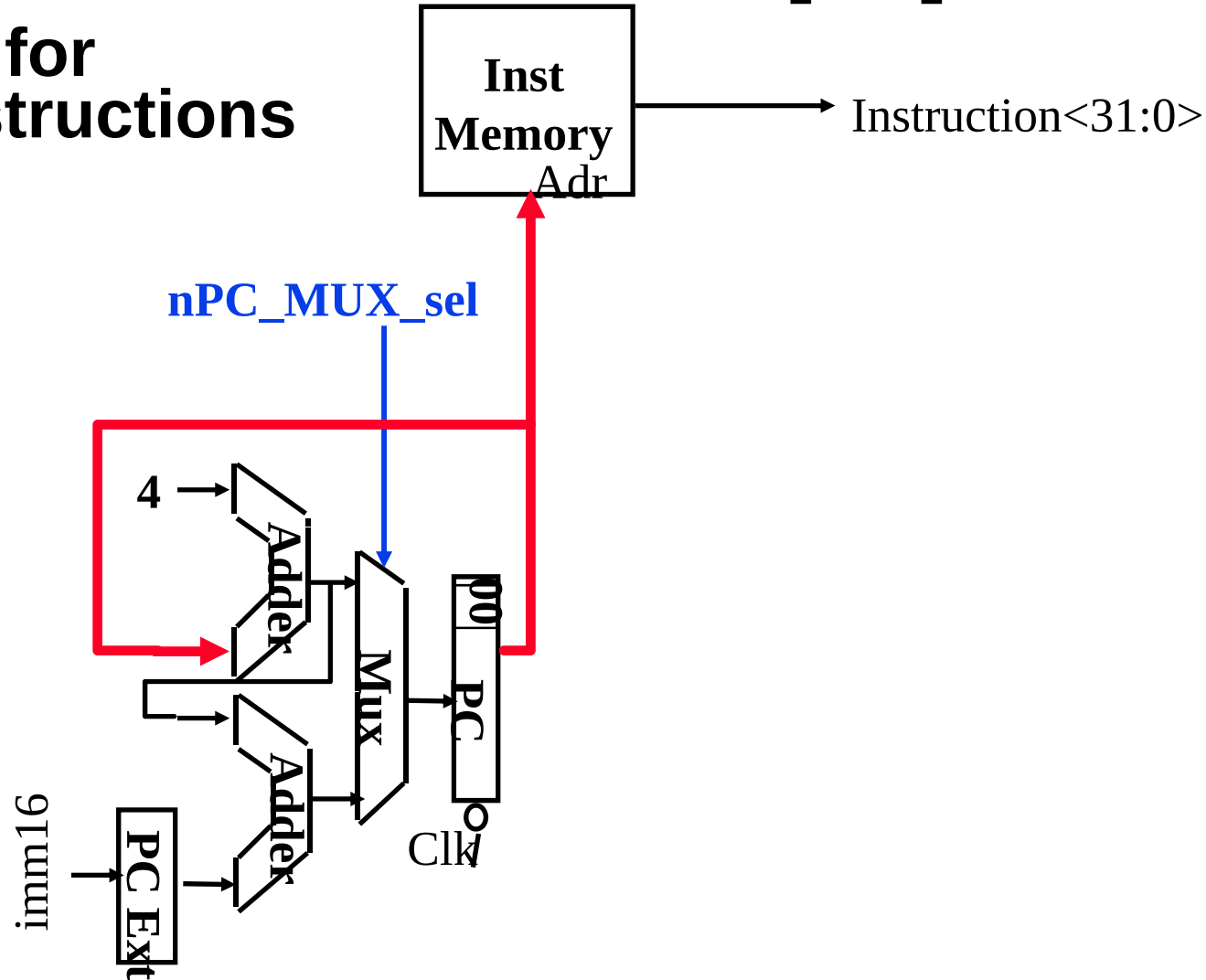
add rd, rs, rt

- **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**

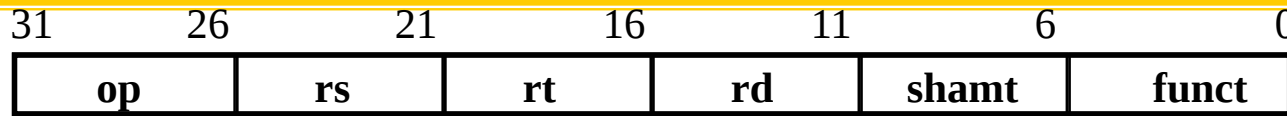
Instruction Fetch Unit at the Beginning of Add

- Fetch the instruction from Instruction memory: $\text{Instruction} = \text{MEM}[\text{PC}]$

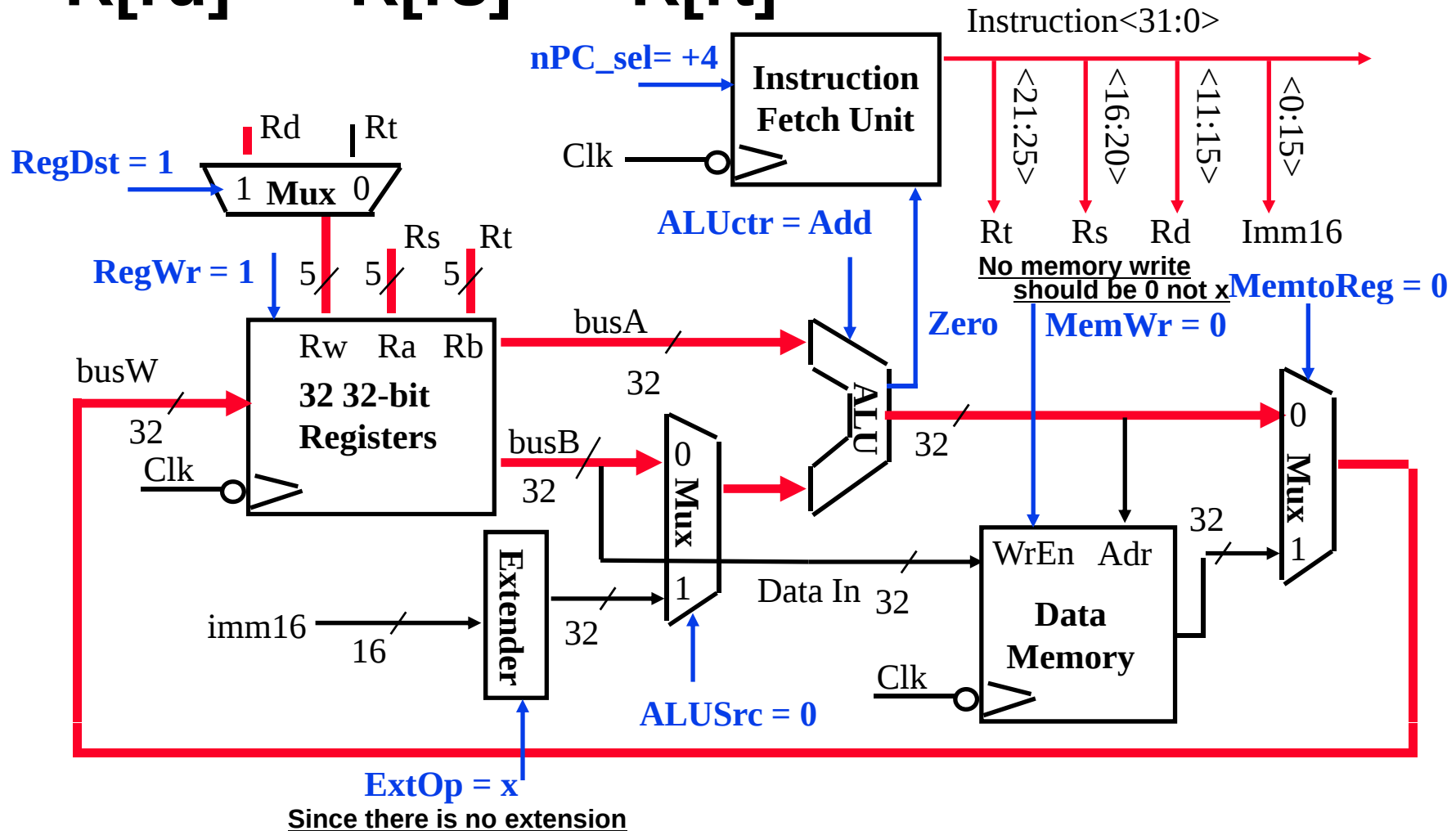
- same for all instructions



The Single Cycle Datapath during Add

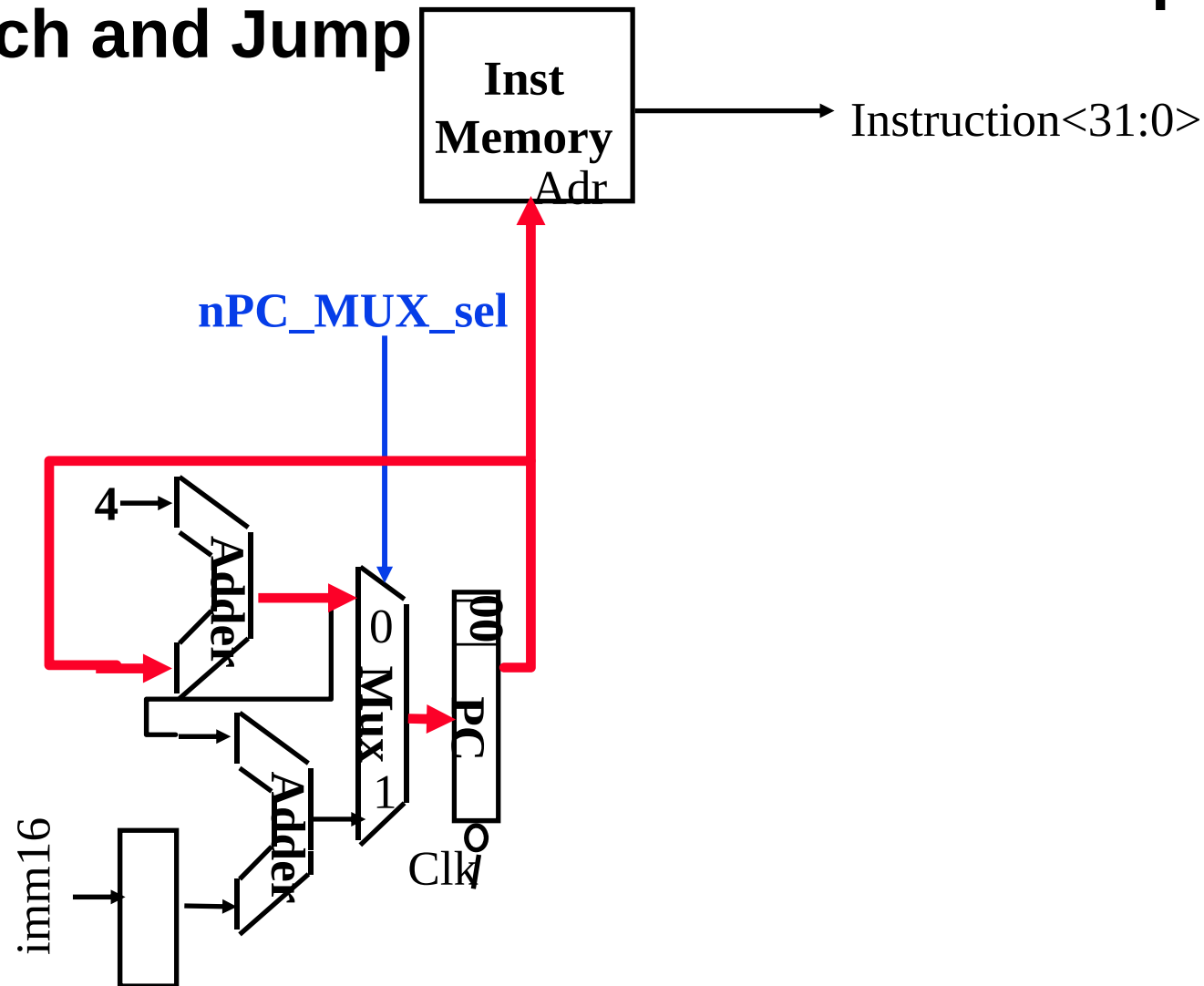


- $R[rd] = R[rs] + R[rt]$

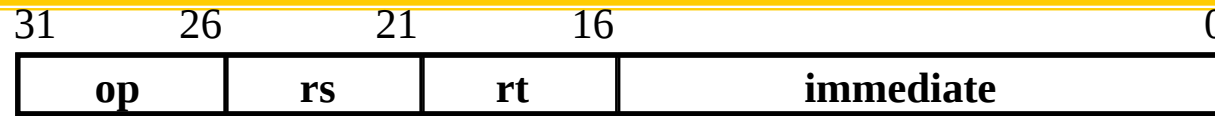


Instruction Fetch Unit at the End of Add

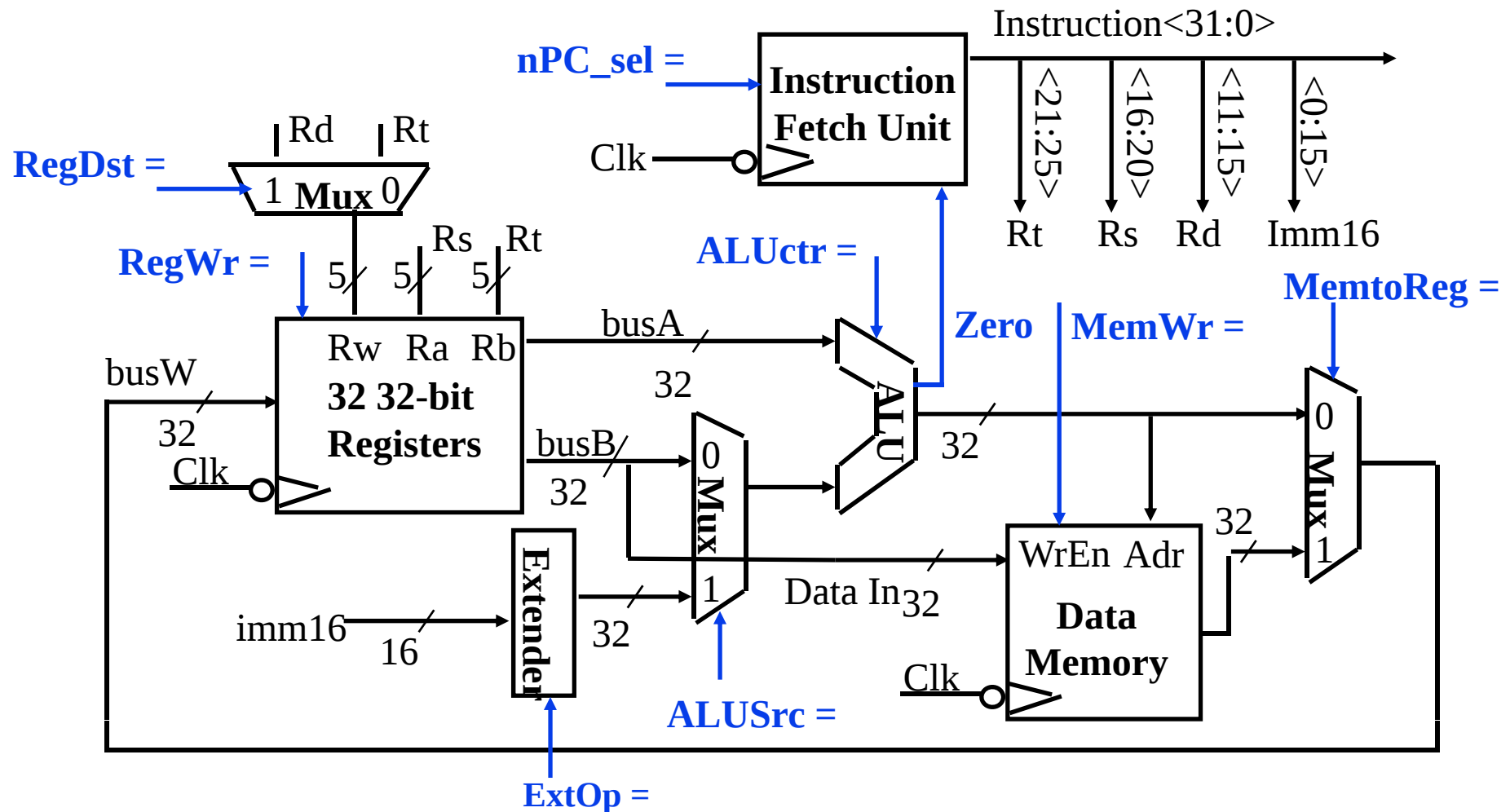
- **$PC = PC + 4$**
 - This is the same for all instructions except:
Branch and Jump



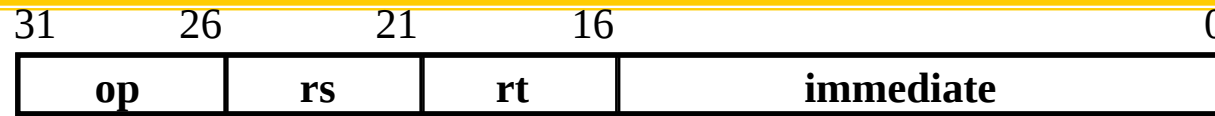
Single Cycle Datapath during Or Immediate?



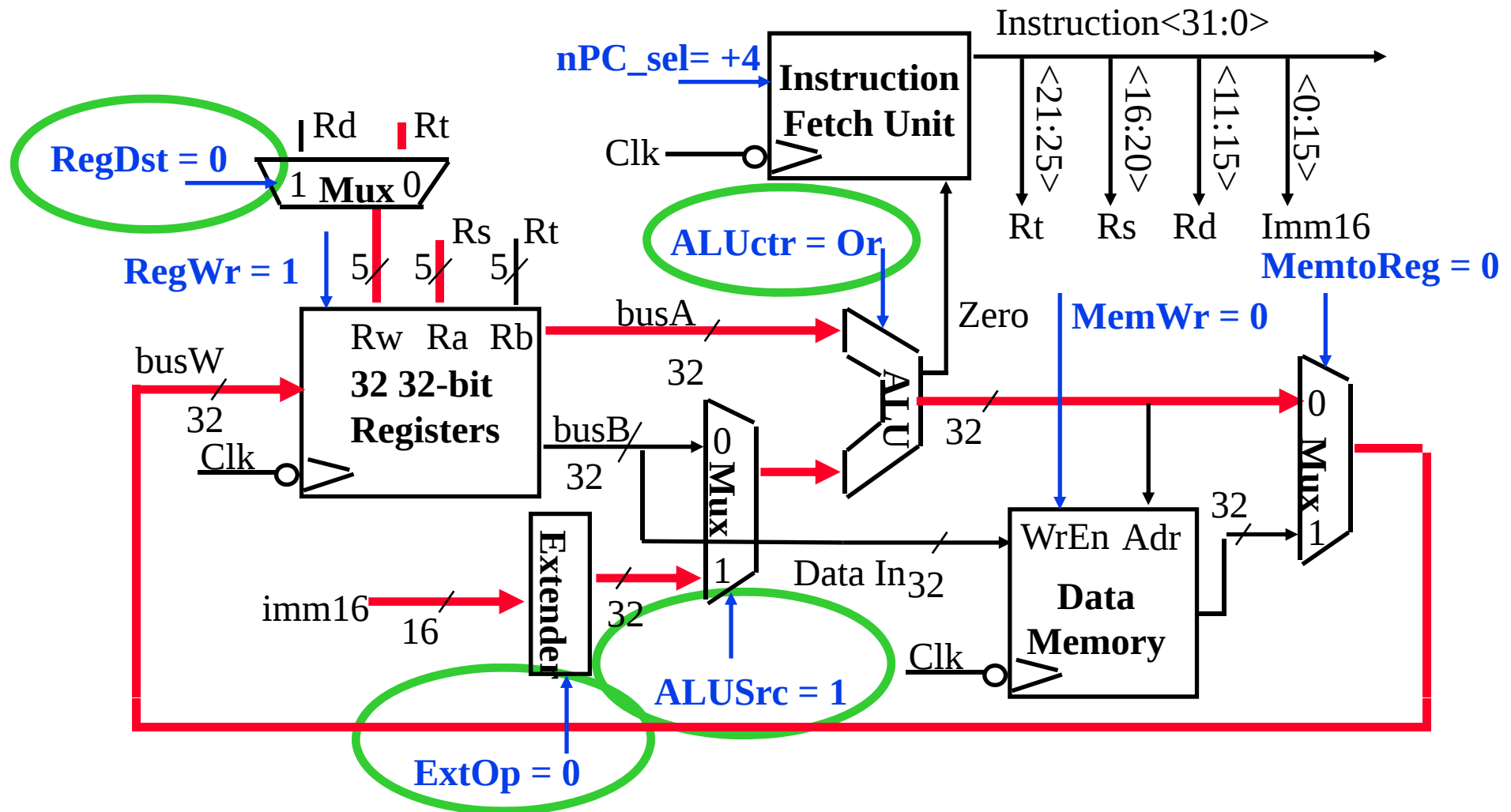
- $R[rt] = R[rs] \text{ OR } \text{ZeroExt}[\text{Imm16}]$



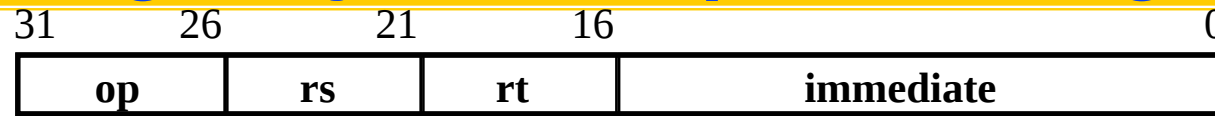
Single Cycle Datapath during Or Immediate?



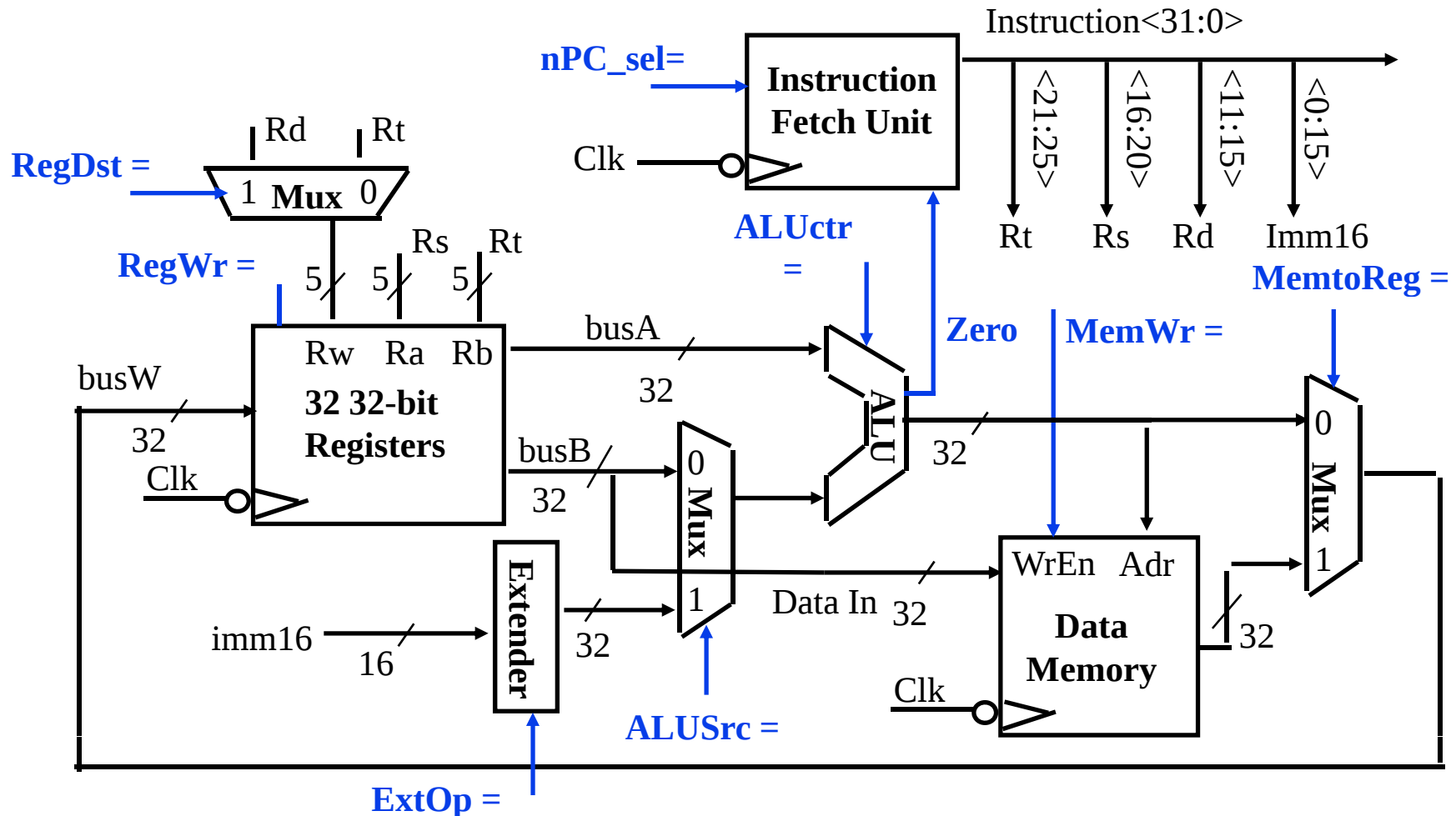
- $R[rt] = R[rs] \text{ OR } \text{ZeroExt}[\text{Imm16}]$



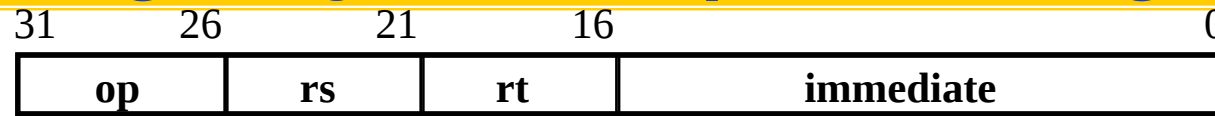
The Single Cycle Datapath during Load?



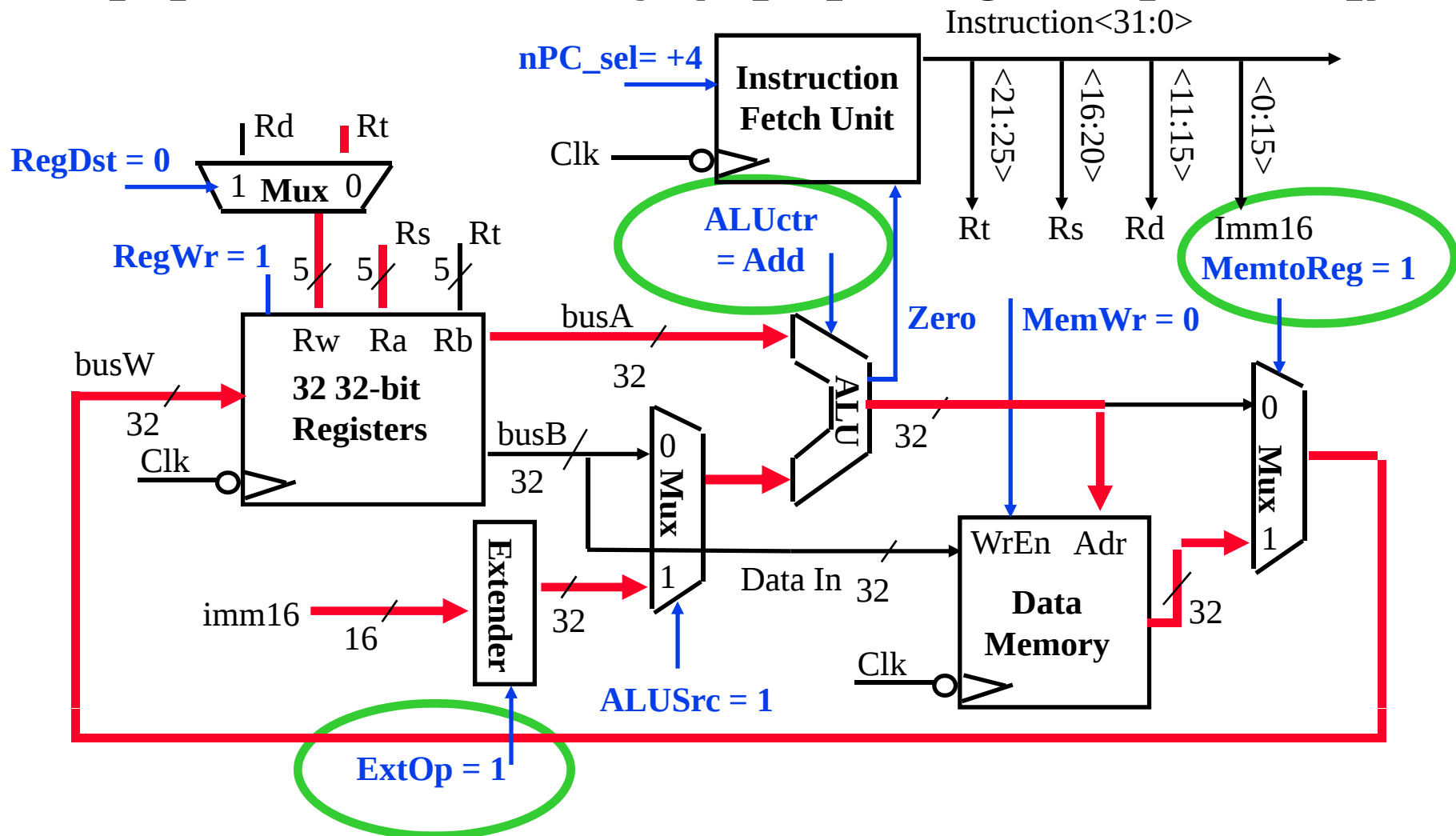
- $R[rt] = \text{Data Memory } \{R[rs] + \text{SignExt}[imm16]\}$



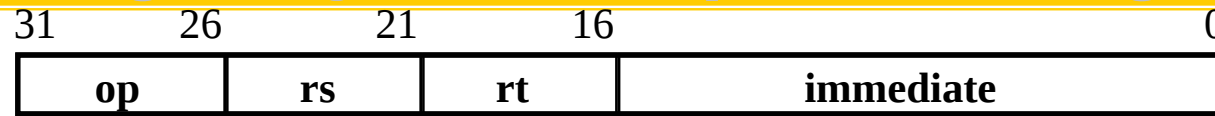
The Single Cycle Datapath during Load



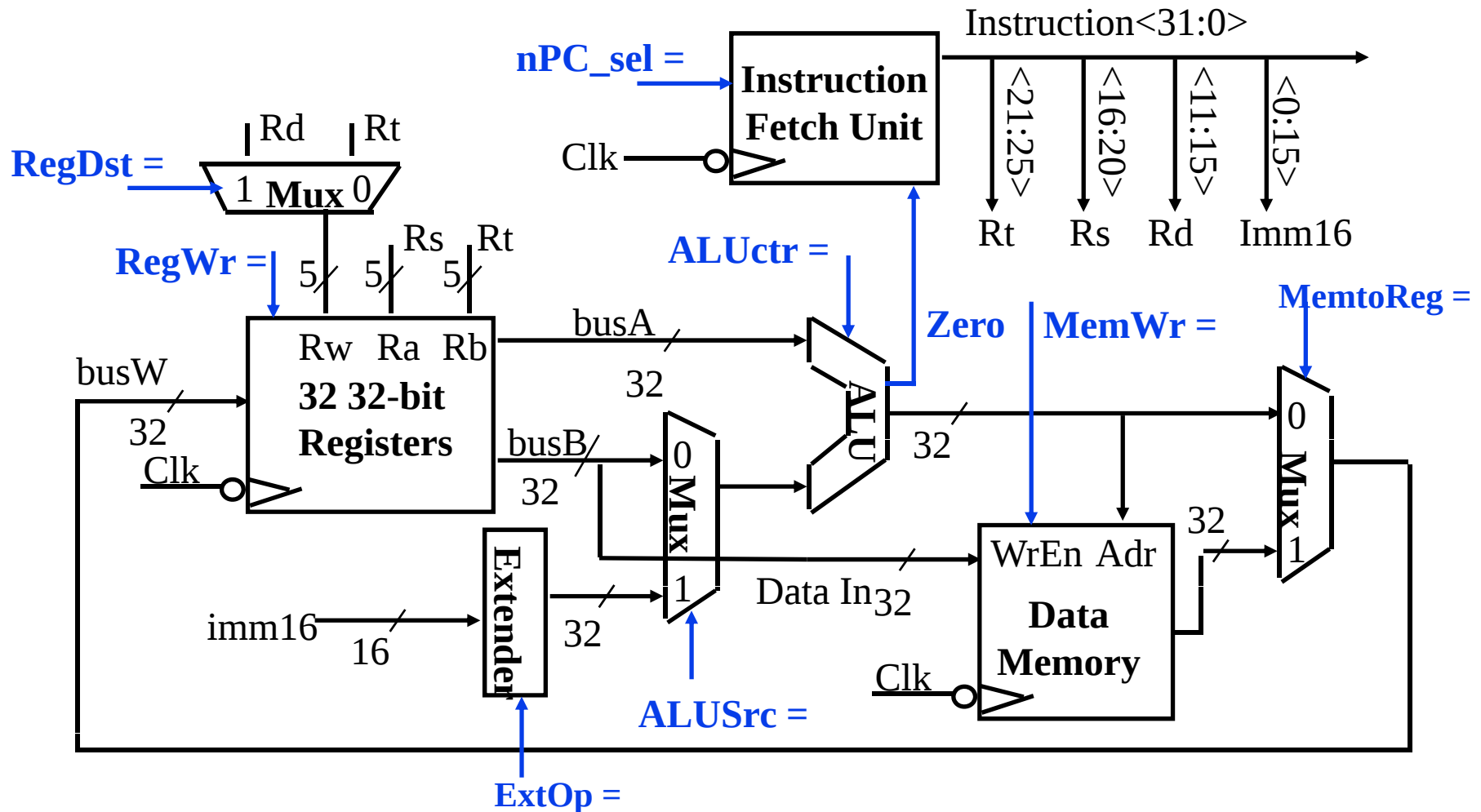
- $R[rt] = \text{Data Memory } \{R[rs] + \text{SignExt}[imm16]\}$



The Single Cycle Datapath during Store?

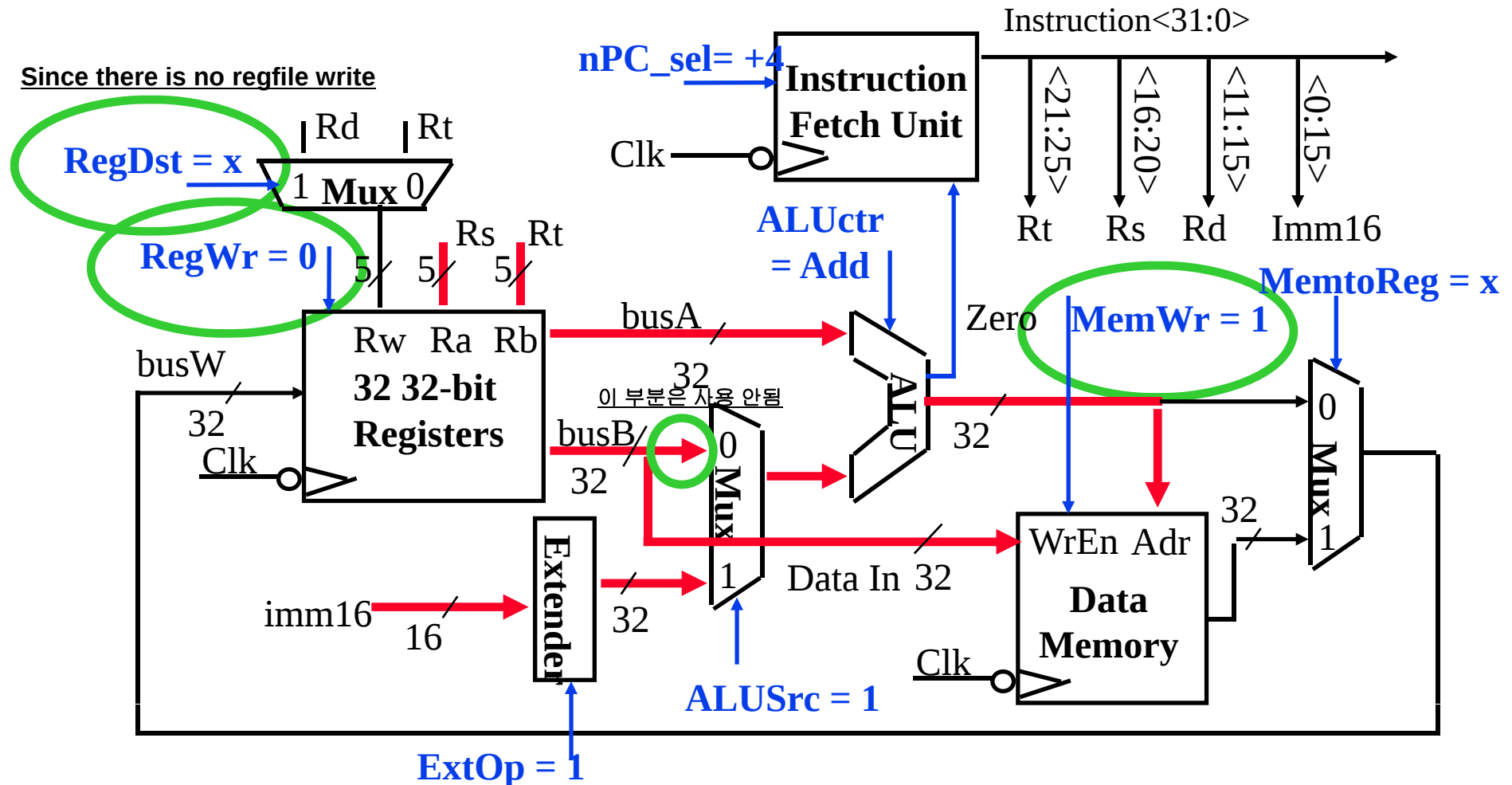


- **Data Memory {R[rs] + SignExt[imm16]} = R[rt]**

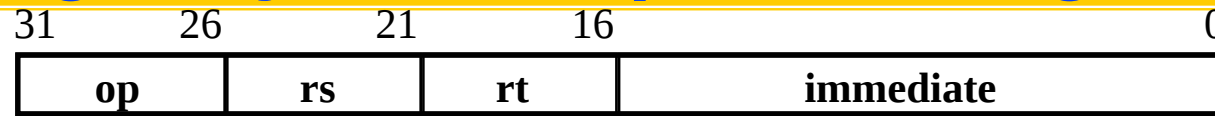


31	26	21	16	0
op	rs	rt	immediate	

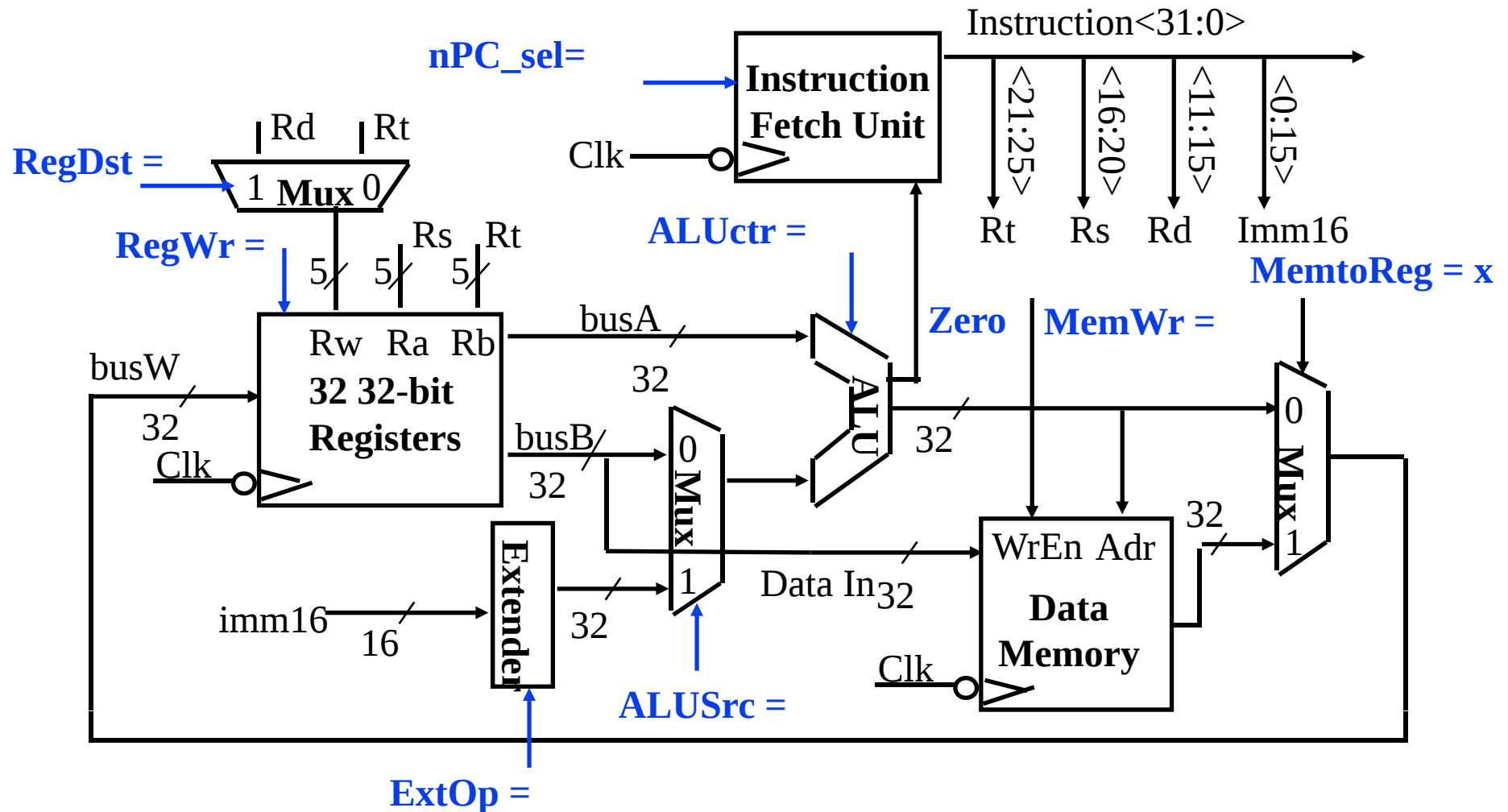
- **Data Memory $\{R[rs] + \text{SignExt}[imm16]\} = R[rt]$**



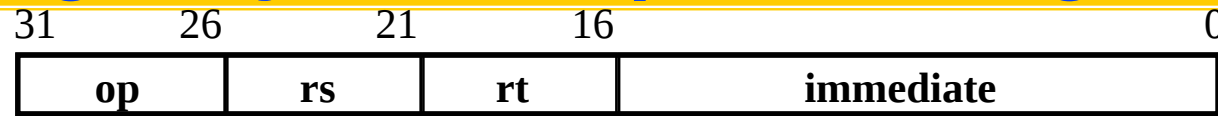
The Single Cycle Datapath during Branch?



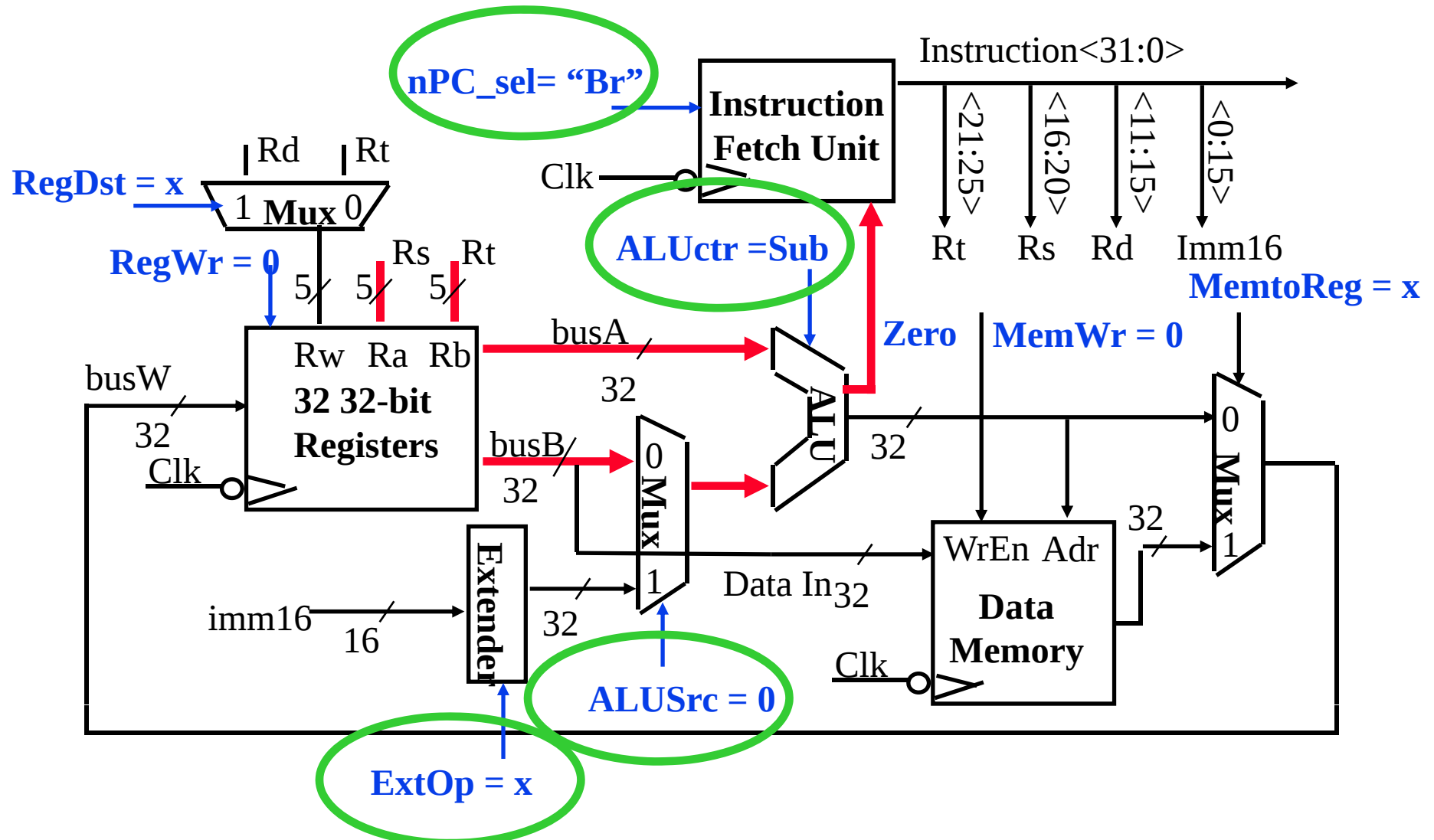
- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



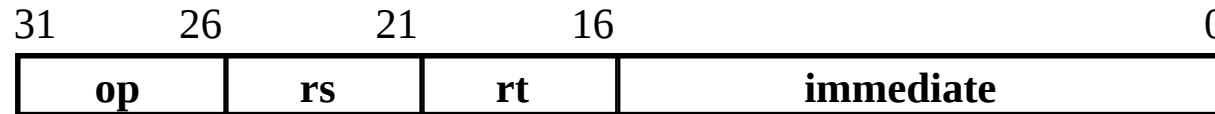
The Single Cycle Datapath during Branch



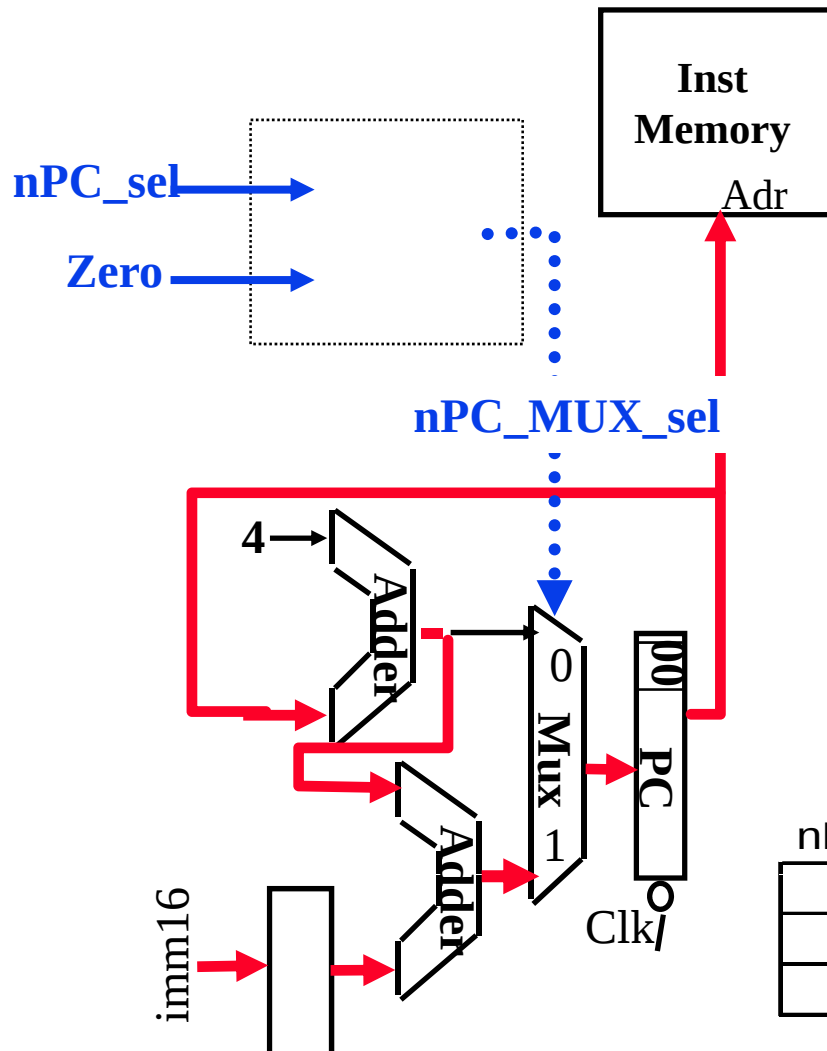
- if ($R[rs] - R[rt] == 0$) then Zero = 1 ; else Zero = 0



Instruction Fetch Unit at the End of Branch



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



• What is encoding of nPC_sel?

• Direct MUX select?

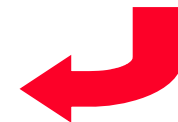
• Branch / not branch

• Let's pick 2nd option

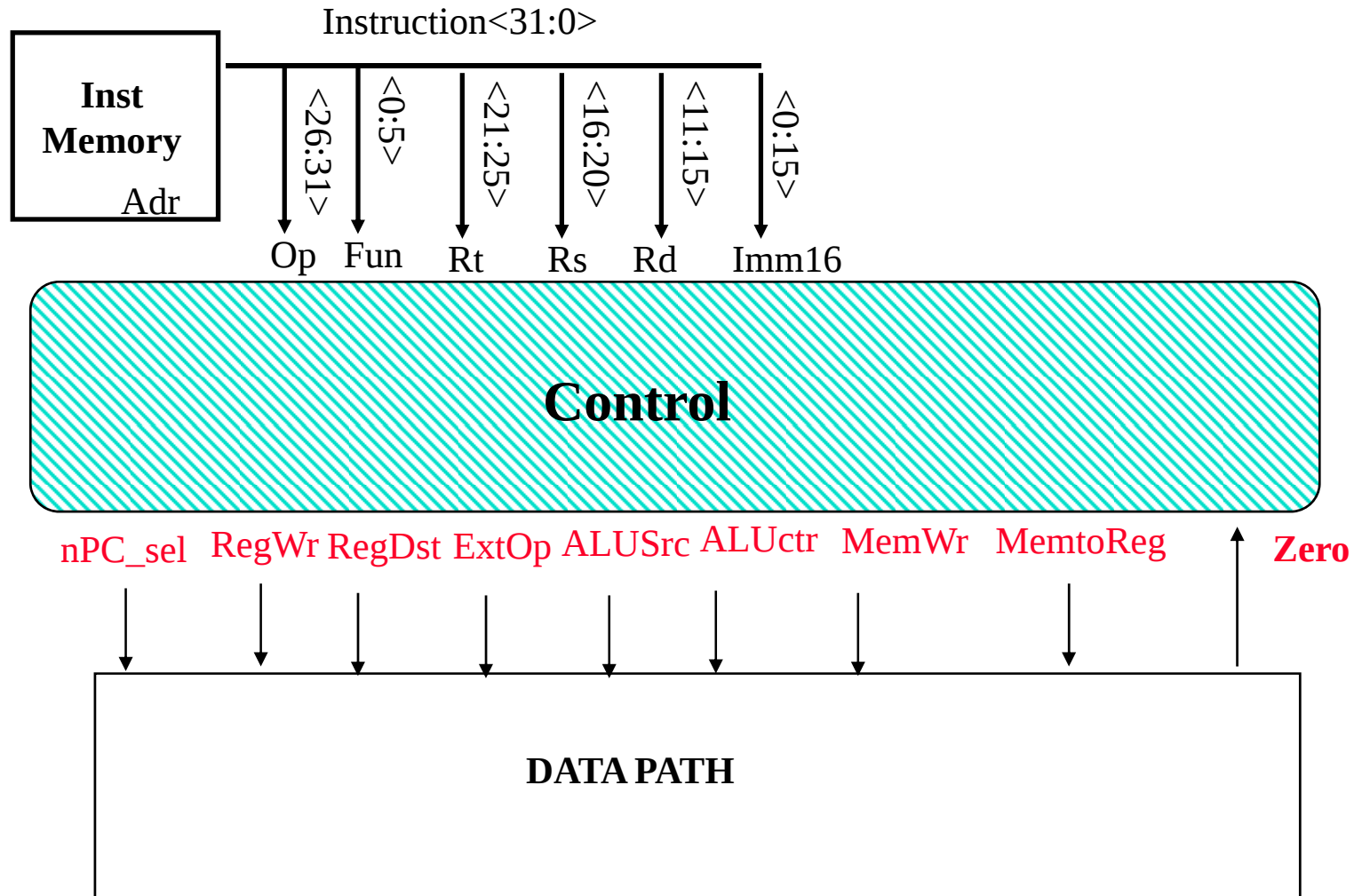
nPC_sel	zero?	<u>nPC_MUX_sel</u> MUX
0	x	0
1	0	0
1	1	1

Q: What logic gate?

AND gate



Step 4: Given Datapath: RTL -> Control



A Summary of the Control Signals (1/2)

inst Register Transfer

ADD $R[rd] \leftarrow R[rs] + R[rt];$ $PC \leftarrow PC + 4$

$ALUsrc = \text{RegB}, ALUctr = \text{"add"}, \text{RegDst} = rd, \text{RegWr}, nPC_sel = \text{"+4"}$

SUB $R[rd] \leftarrow R[rs] - R[rt];$ $PC \leftarrow PC + 4$

$ALUsrc = \text{RegB}, ALUctr = \text{"sub"}, \text{RegDst} = rd, \text{RegWr}, nPC_sel = \text{"+4"}$

ORi $R[rt] \leftarrow R[rs] + \text{zero_ext}(\text{Imm16});$ $PC \leftarrow PC + 4$

$ALUsrc = \text{Im}, \text{Extop} = \text{"Z"}, ALUctr = \text{"or"}, \text{RegDst} = rt, \text{RegWr}, nPC_sel = \text{"+4"}$

LOAD $R[rt] \leftarrow \text{MEM}[R[rs] + \text{sign_ext}(\text{Imm16})]; PC \leftarrow PC + 4$

$ALUsrc = \text{Im}, \text{Extop} = \text{"Sn"}, ALUctr = \text{"add"},$
 $\text{MemtoReg}, \text{RegDst} = rt, \text{RegWr}, \quad nPC_sel = \text{"+4"}$

STORE $\text{MEM}[R[rs] + \text{sign_ext}(\text{Imm16})] \leftarrow R[rs]; PC \leftarrow PC + 4$

$ALUsrc = \text{Im}, \text{Extop} = \text{"Sn"}, ALUctr = \text{"add"}, \text{MemWr}, nPC_sel = \text{"+4"}$

BEQ if ($R[rs] == R[rt]$) then $PC \leftarrow PC + \text{sign_ext}(\text{Imm16}) || 00$ else $PC \leftarrow PC + 4$

$nPC_sel = \text{"Br"}, ALUctr = \text{"sub"}$

A Summary of the Control Signals (2/2)

See Appendix A

func

op

	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	0
Jump	0	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x	x
ALUctr<2:0>	Add	Subtract	Or	Add	Add	Subtract	xxx

Column
by column
으로 볼것

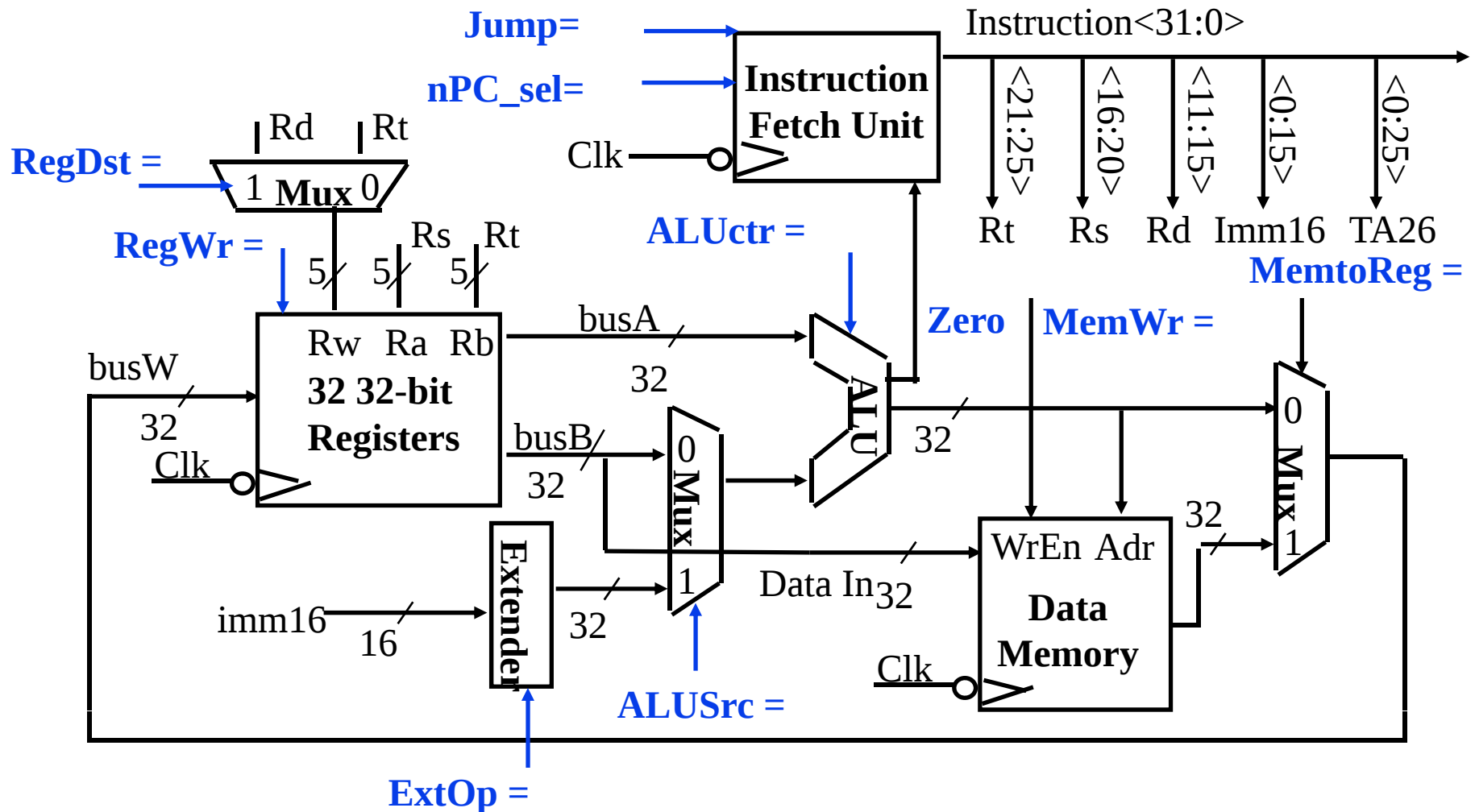
	31	26	21	16	11	6	0	
R-type	op		rs	rt	rd	shamt	funct	add, sub
I-type	op		rs	rt	immediate			ori, lw, sw, beq
J-type	op		target address				jump	

Administrivia

- **Project 2 Due Sunday**
- **HW 6 out tomorrow**
- **Slight shakeup of the schedule starting tomorrow (we're only doing one Control lecture)**
 - **This allows us to have the second midterm before the drop deadline, if that would be preferable**

The Single Cycle Datapath during Jump

- J-type instruction format:
- | | | | |
|----|----|----|----------------|
| 31 | 26 | 25 | 0 |
| op | | | target address |
- jump
- **New PC = { PC[31..28], target address, 00 }**



J-type 31 26 25 0 **jump**

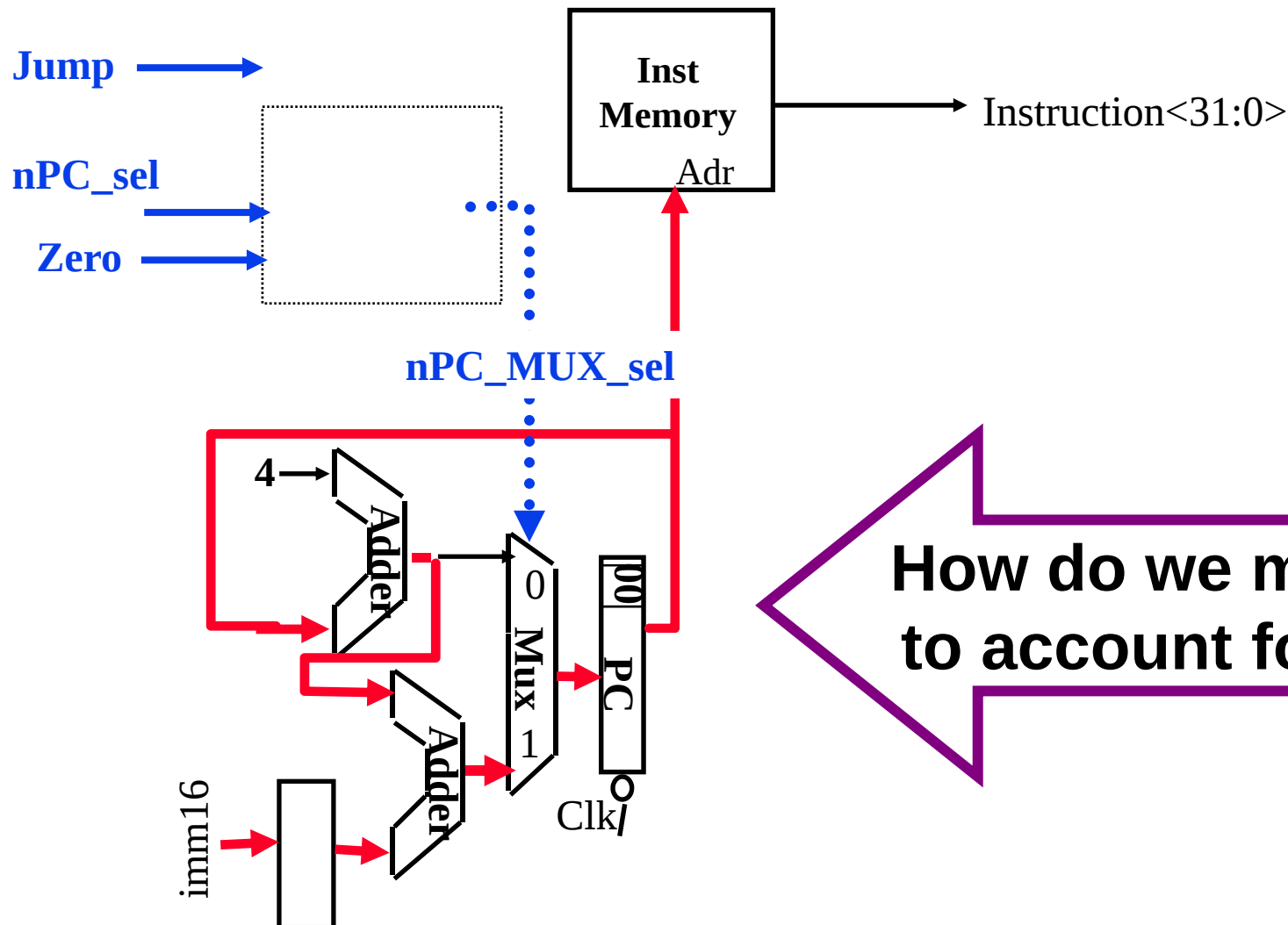
op **target address**

-
- The diagram illustrates the internal components and control signals of a processor:
- Instruction Fetch Unit:** Receives the **Instruction<31:0>** and outputs fields **<21:25>** (Rt), **<16:20>** (Rs), **<11:15>** (Rd), **<0:15>** (Imm16), and **<0:25>** (TA26). It is controlled by **Jump=1**, **nPC_sel=0**, and **Clk**.
 - 32-bit Registers:** Features **Rw**, **Ra**, and **Rb** ports. It receives **busW** (32-bit) and **Clk**. It outputs **busA** (32-bit) and **busB** (32-bit). A control signal **RegWr = 0** is shown.
 - ALU:** Receives **busA** and **busB**. It is controlled by **ALUctr = x** and **ALUSrc = x**. It outputs a 32-bit result and a **Zero** flag.
 - Data Memory:** Receives **WrEn** and **Adr** (32-bit). It outputs **Data In₃₂**. It is controlled by **Clk** and **MemWr = 0**.
 - Muxes:**
 - Mux 0:** Selects between **Rd** and **Rt** based on **RegDst = x**.
 - Mux 1:** Selects between **busA** and **busB** based on **ALUSrc = x**.
 - Mux 2:** Selects between **Zero** and **TA26** based on **MemtoReg = x**.
 - Extender:** Takes **imm16** (16-bit) and **ExtOp = x** as inputs to produce a 32-bit output for **Mux 1**.

Instruction Fetch Unit at the End of Jump



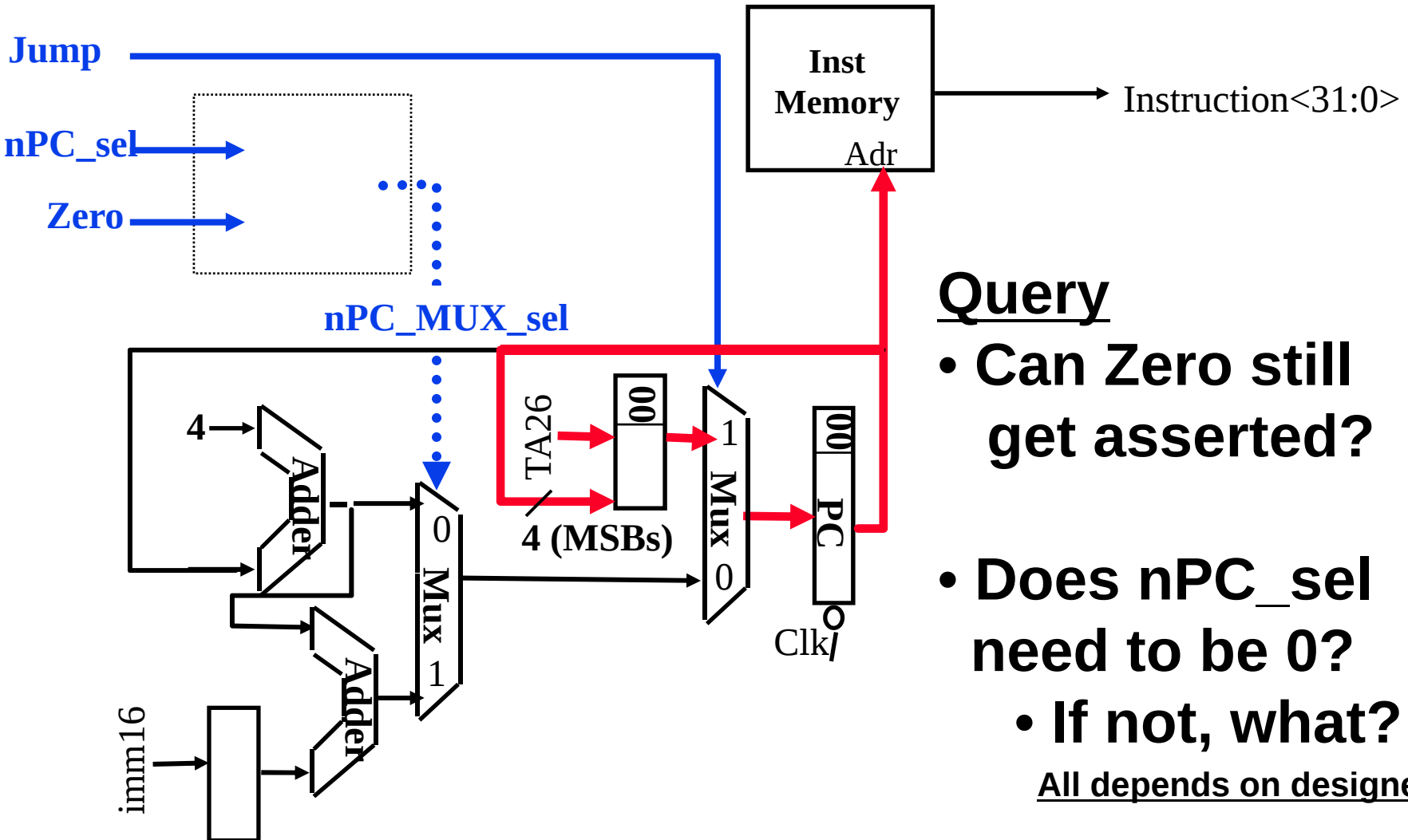
- New PC = { PC[31..28], target address, 00 }



How do we modify this to account for jumps?



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



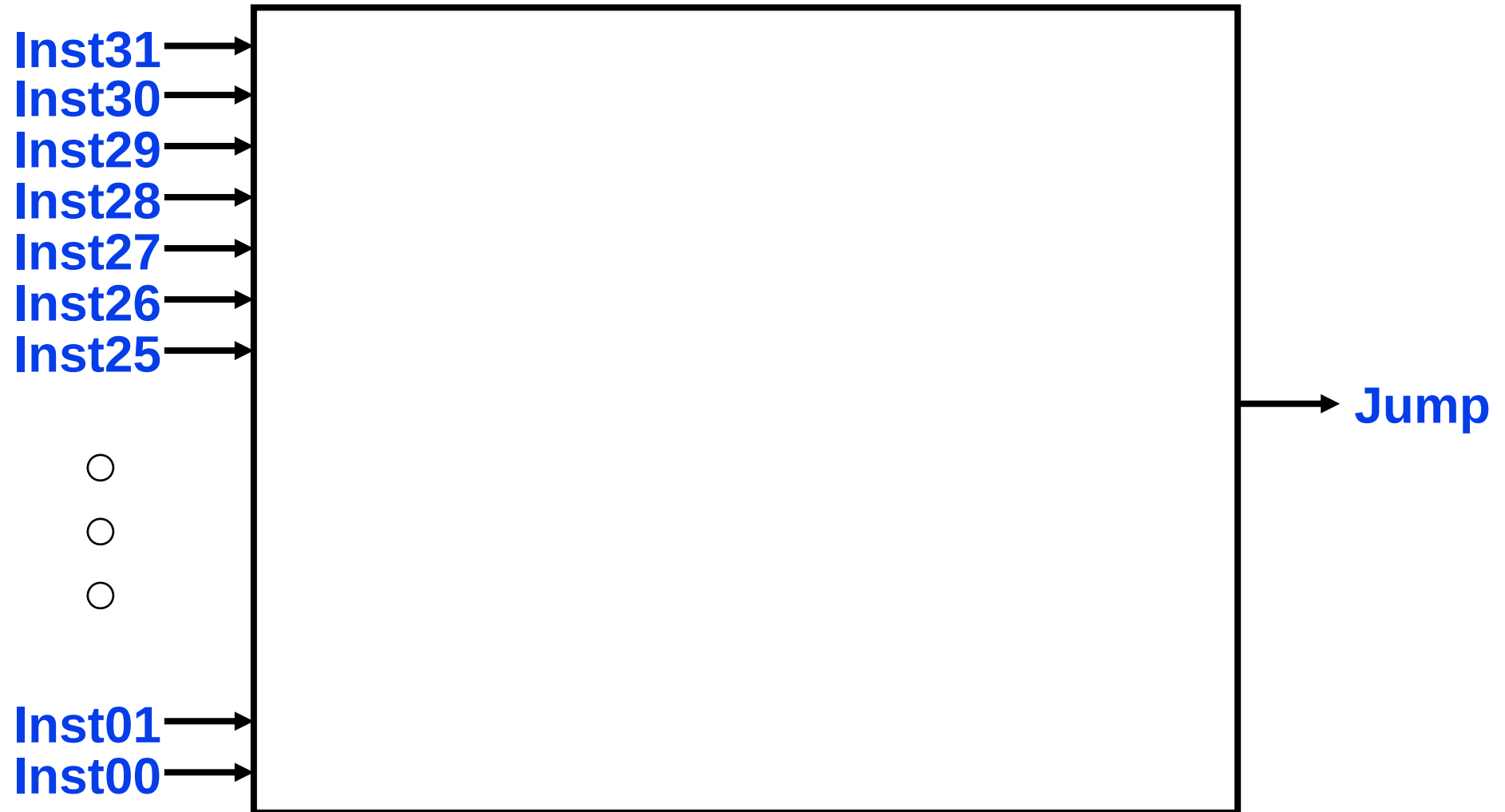
Query

- **Can Zero still get asserted?**

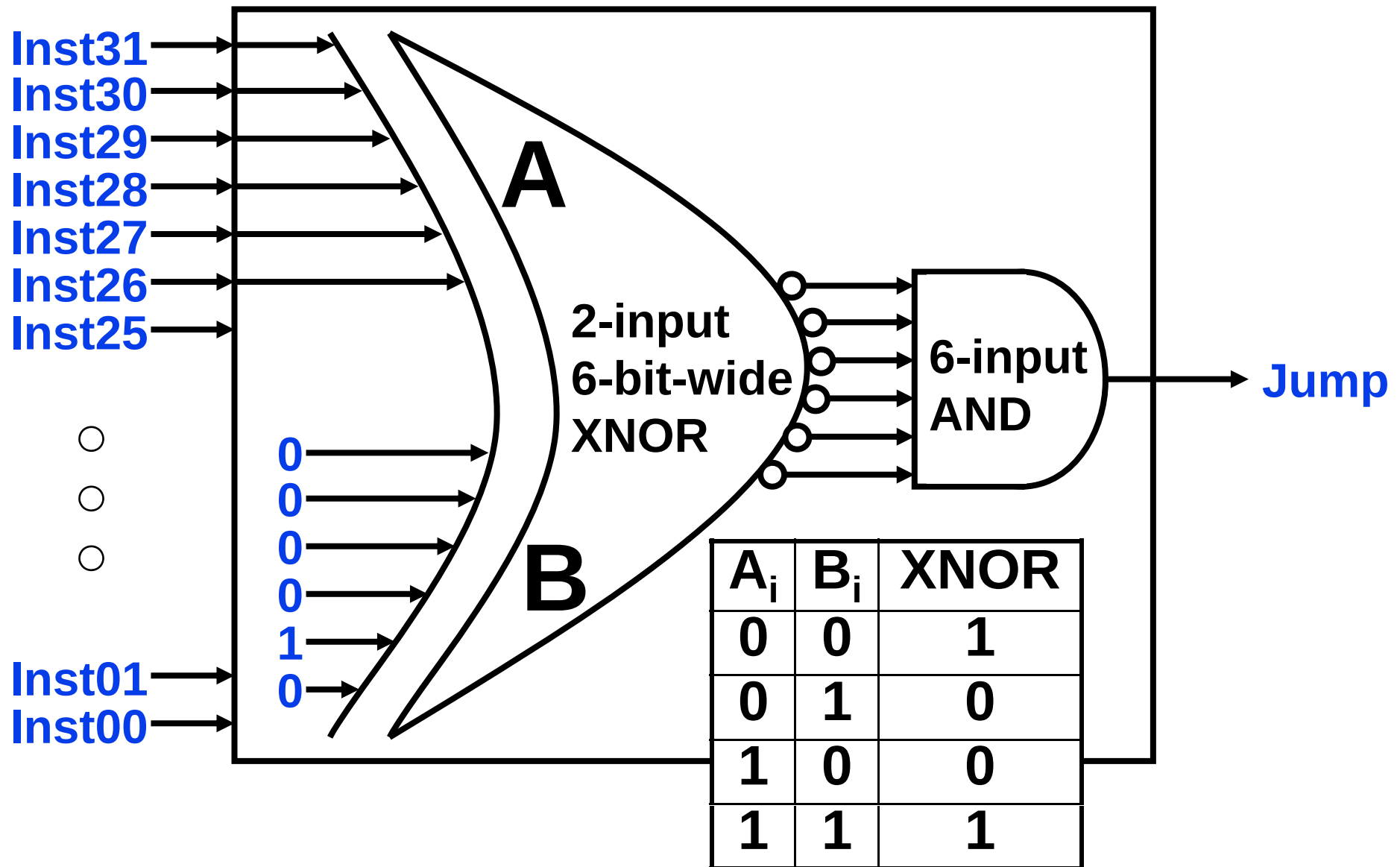
- Does nPC_sel need to be 0?
 - If not, what?

All depends on designer

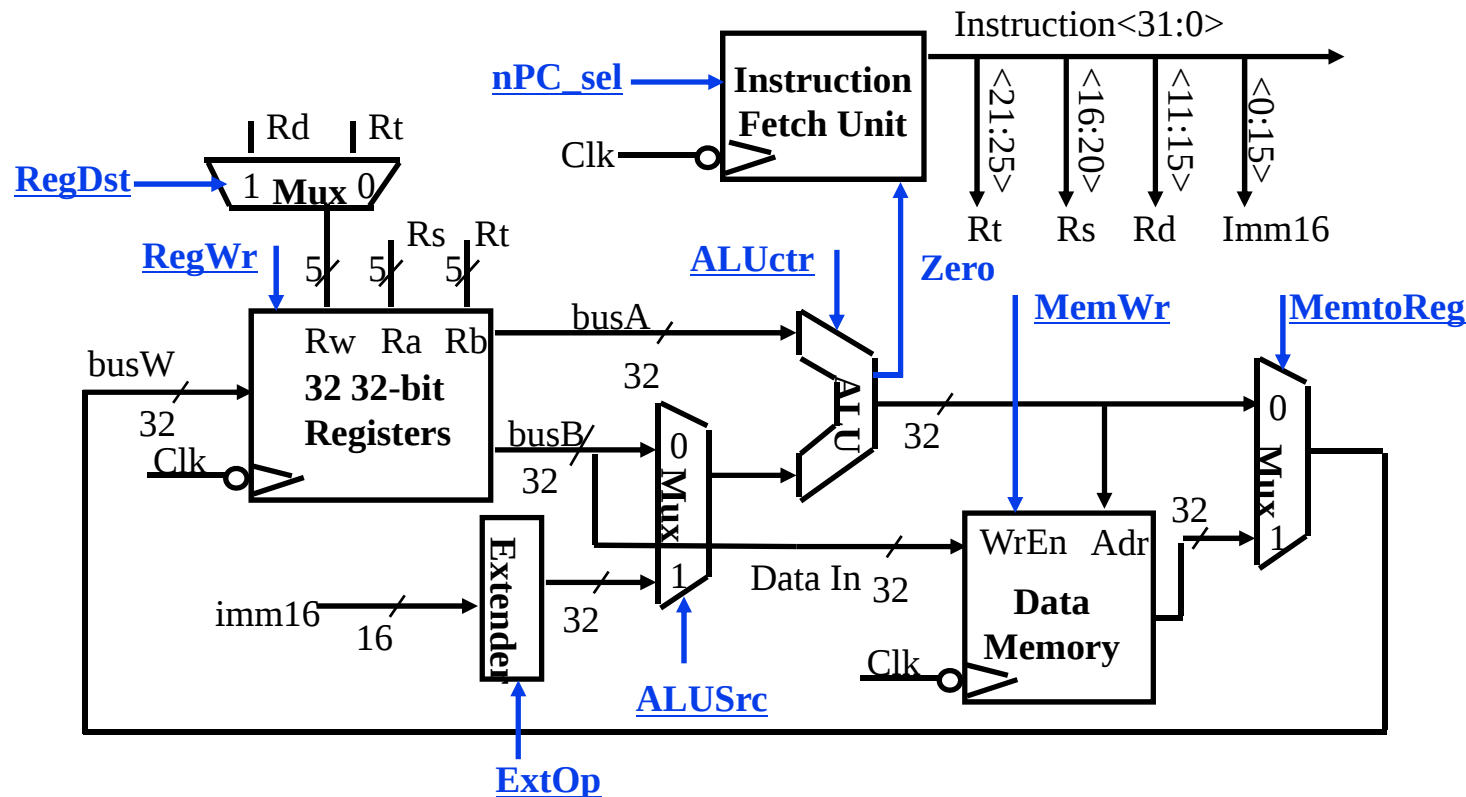
Build CL to implement Jump on paper now



Build CL to implement Jump on paper now



Peer Instruction



- MemToReg='x' & ALUctr='sub'. SUB or BEQ?
- ALUctr='add'. Which 1 signal is different for all 3 of: ADD, LW, & SW? RegDst or ExtOp?
- "Don't Care" signals are useful because we can simplify our Boolean equations?

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

