

inst.eecs.berkeley.edu/~cs61c

UC Berkeley CS61C : Machine Structures



Lecture 26 Single-cycle CPU Control

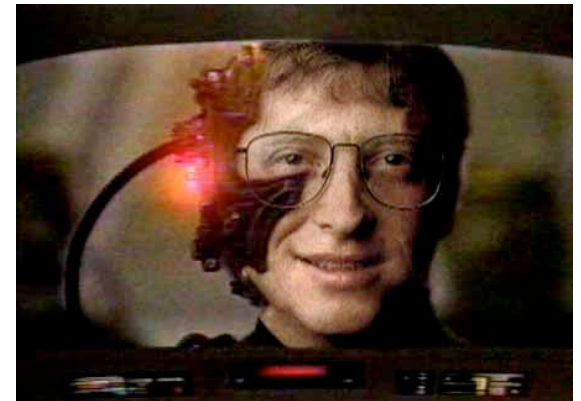
2007-03-21

Exhausted TA Ben Sussman

www.icanhascheezburger.com

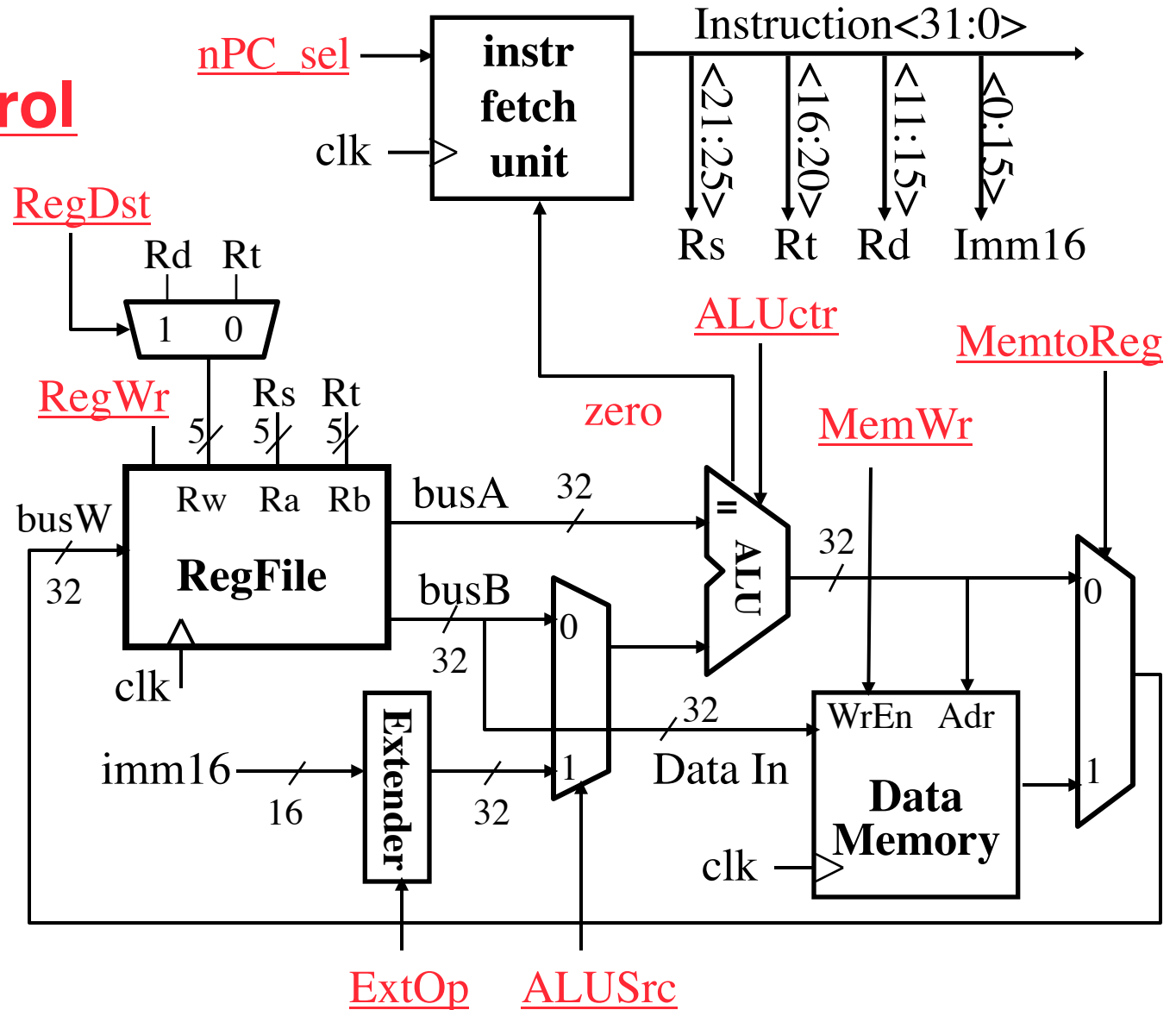
Qutrits Bring Quantum Computers Closer:
An Australian group has built and tested logic gates that convert qubits into qutrits (three-level quantum states)!

But who cares: new iPhones soon? Ben hopes so...



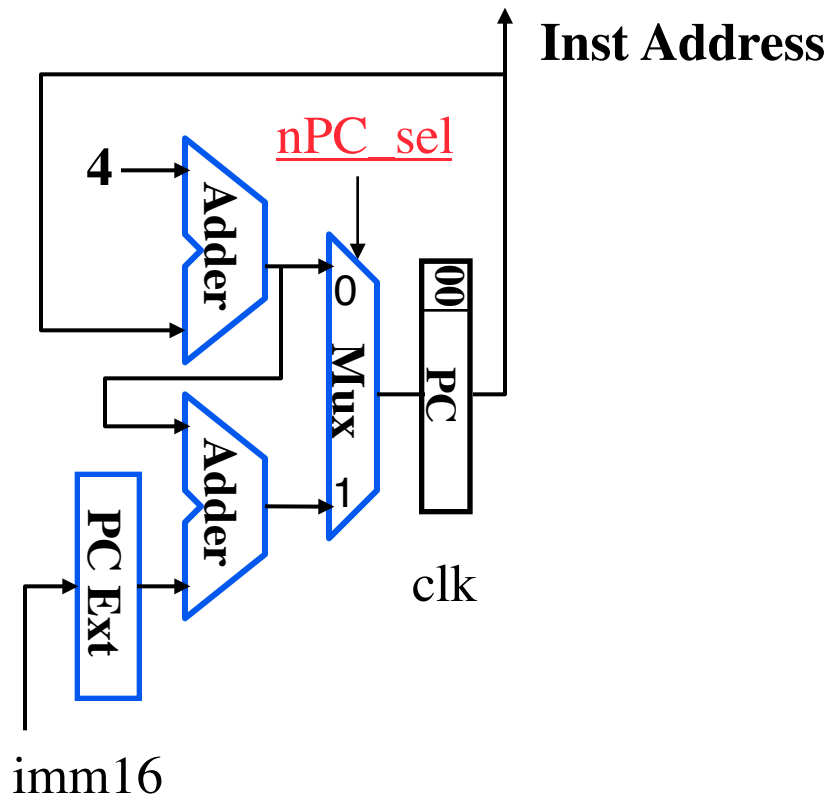
Review: A Single Cycle Datapath

- We have everything except control signals



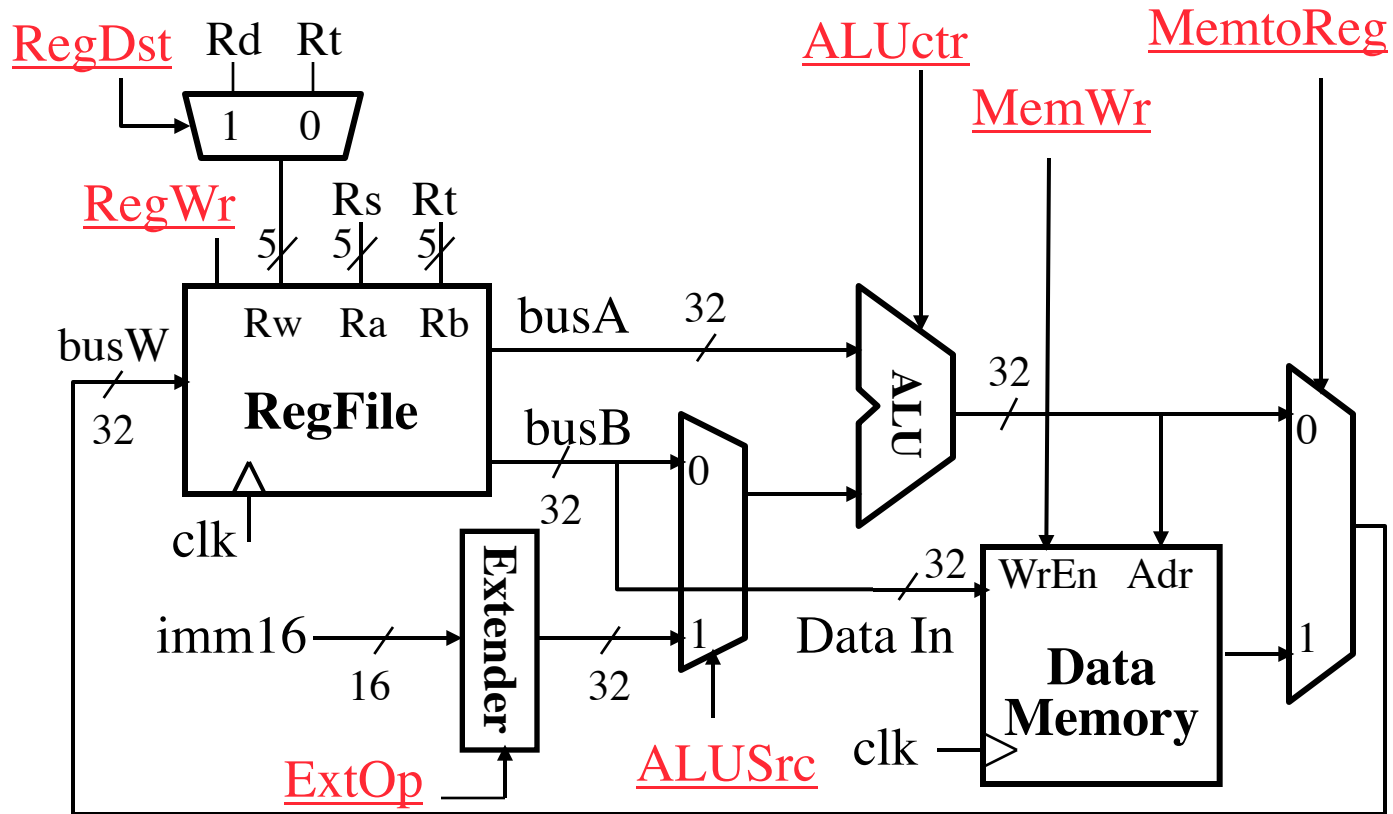
Recap: Meaning of the Control Signals

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

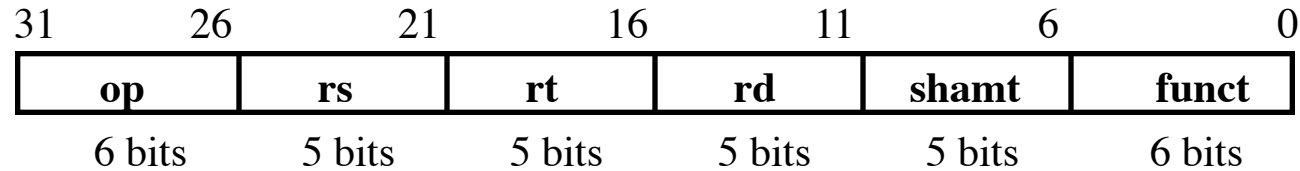


Recap: Meaning of the Control Signals

- **ExtOp:** “zero”, “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

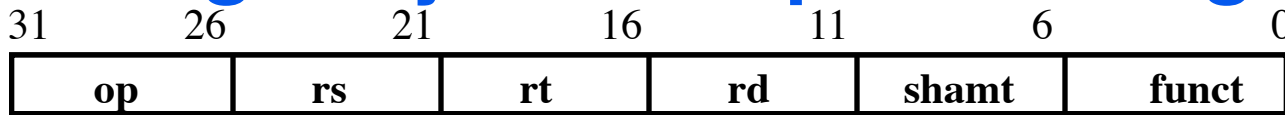


add rd, rs, rt

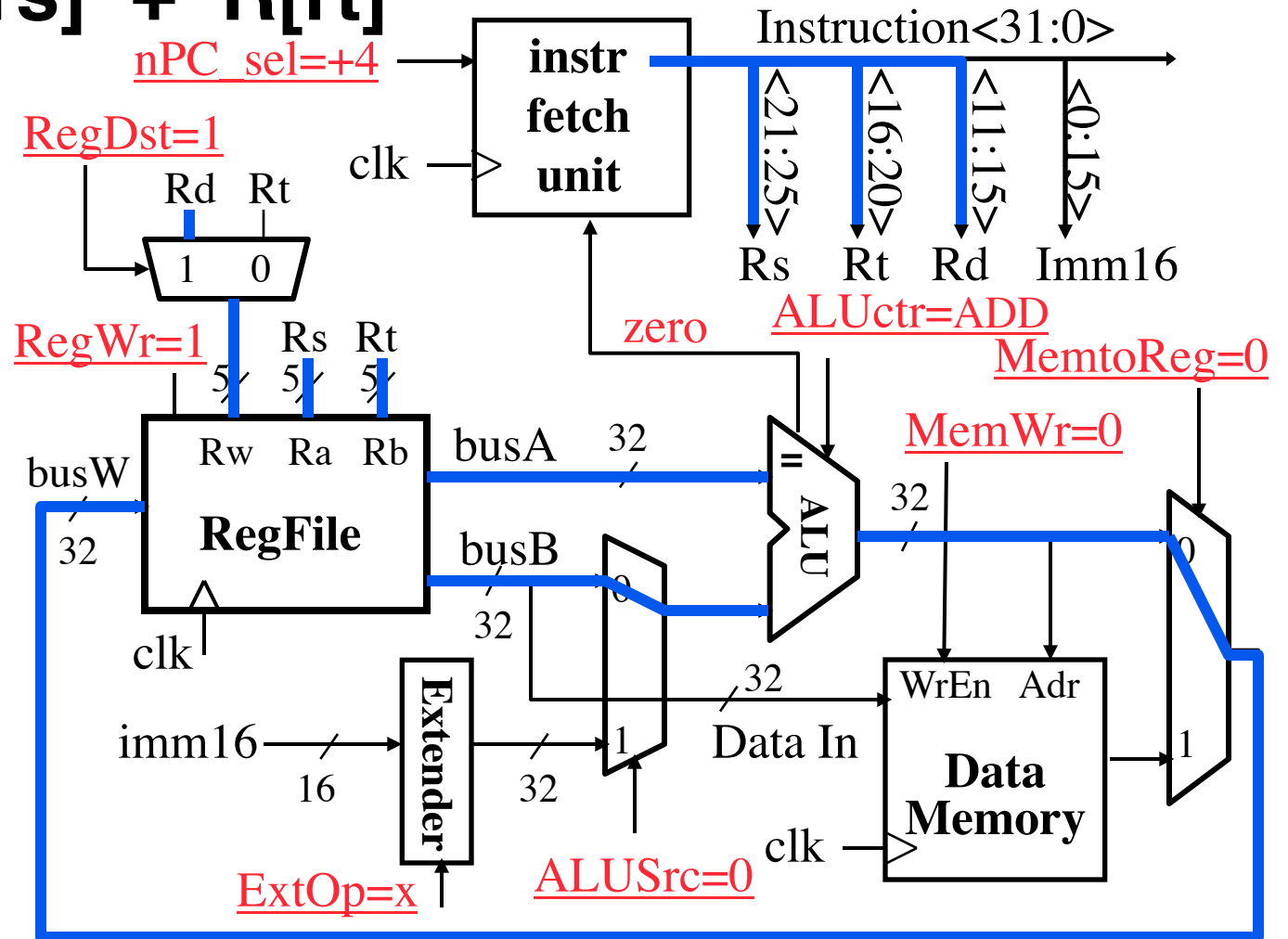
- **MEM[PC]** **Fetch the instruction from memory**
- **$R[rd] = R[rs] + R[rt]$** **The actual operation**
- **$PC = PC + 4$** **Calculate the next instruction's address**



The Single Cycle Datapath during Add

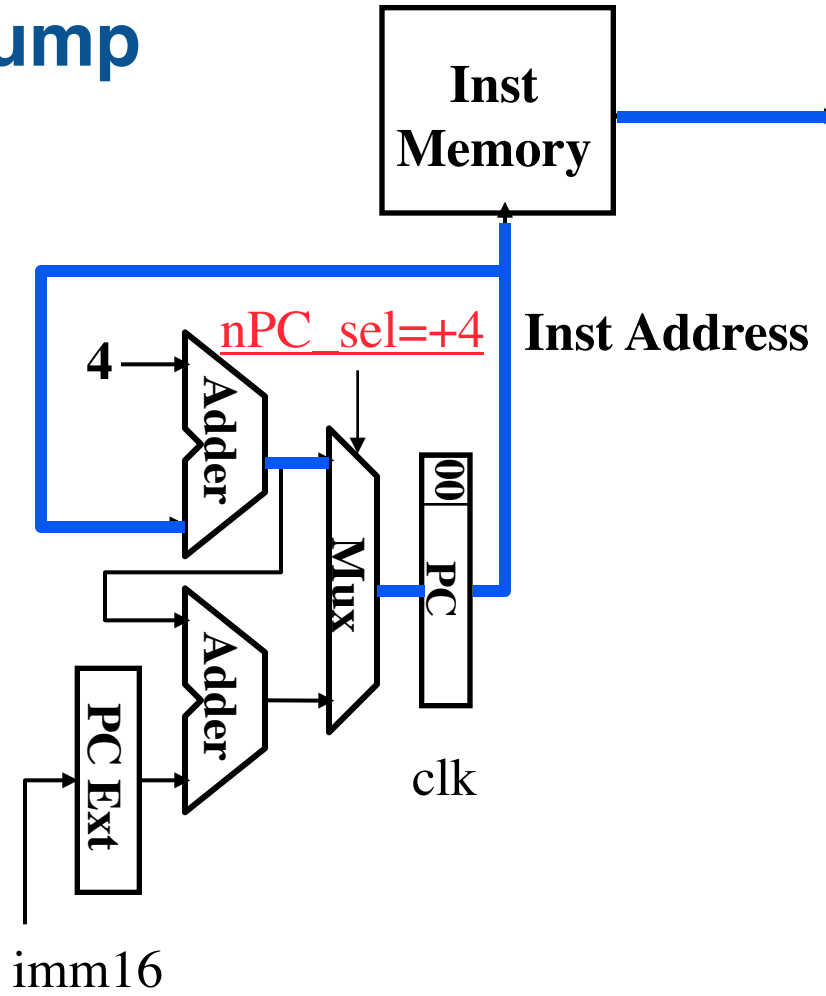


$$R[rd] = R[rs] + R[rt]$$

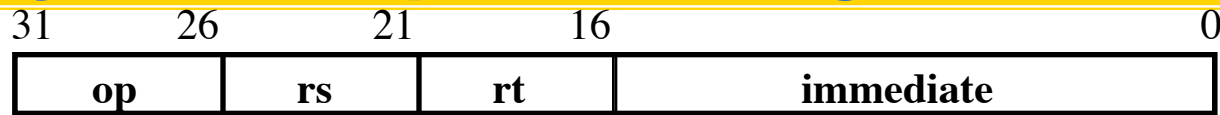


Instruction Fetch Unit at the End of Add

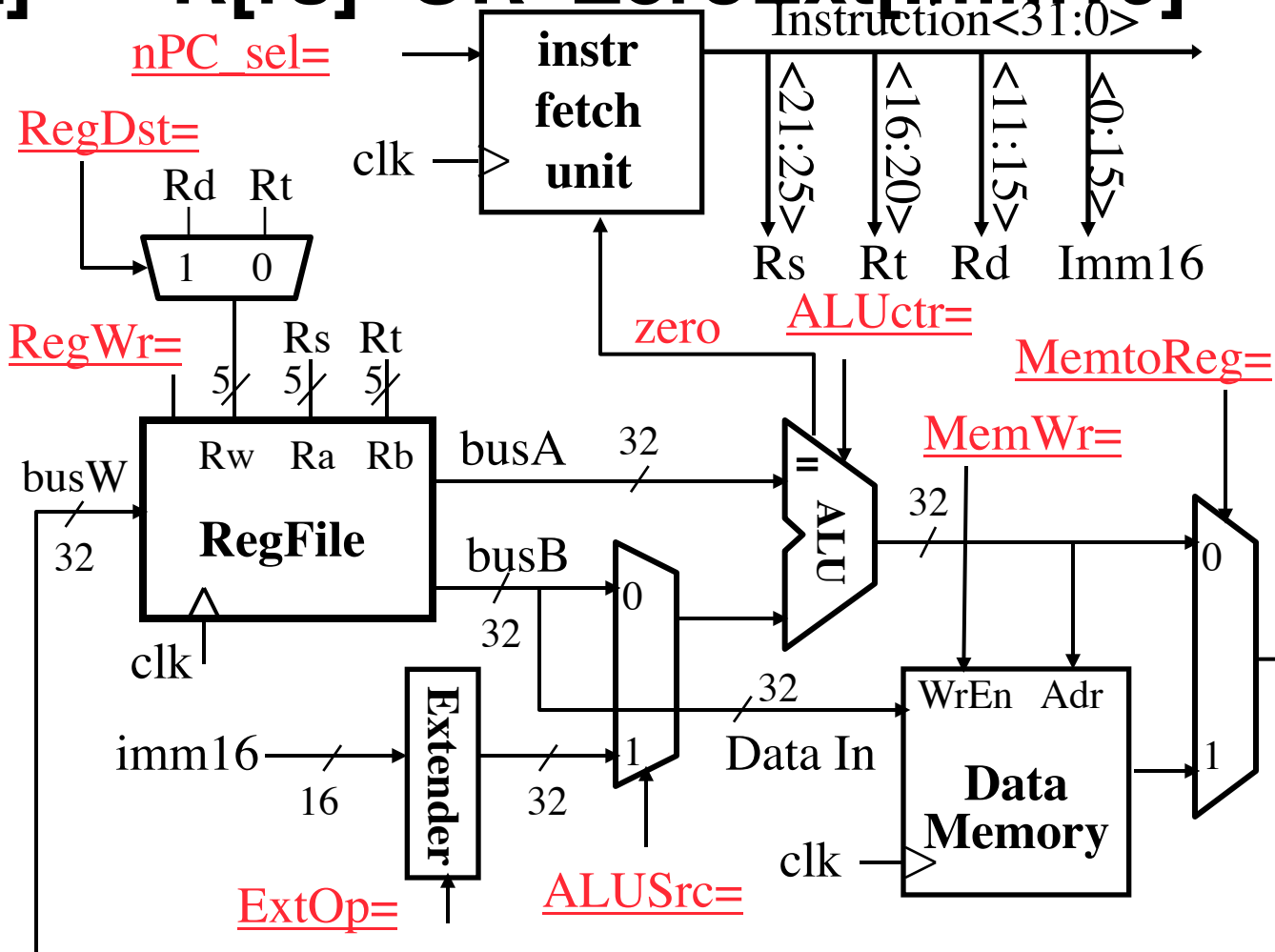
- **PC = PC + 4**
 - This is the same for all instructions except: Branch and Jump



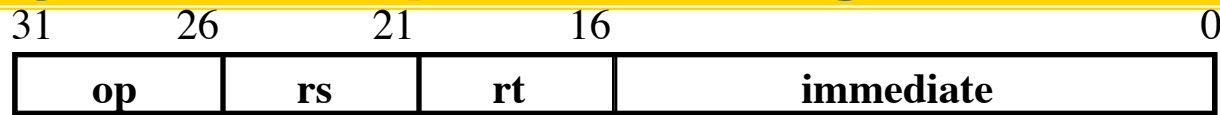
Single Cycle Datapath during Or Immediate?



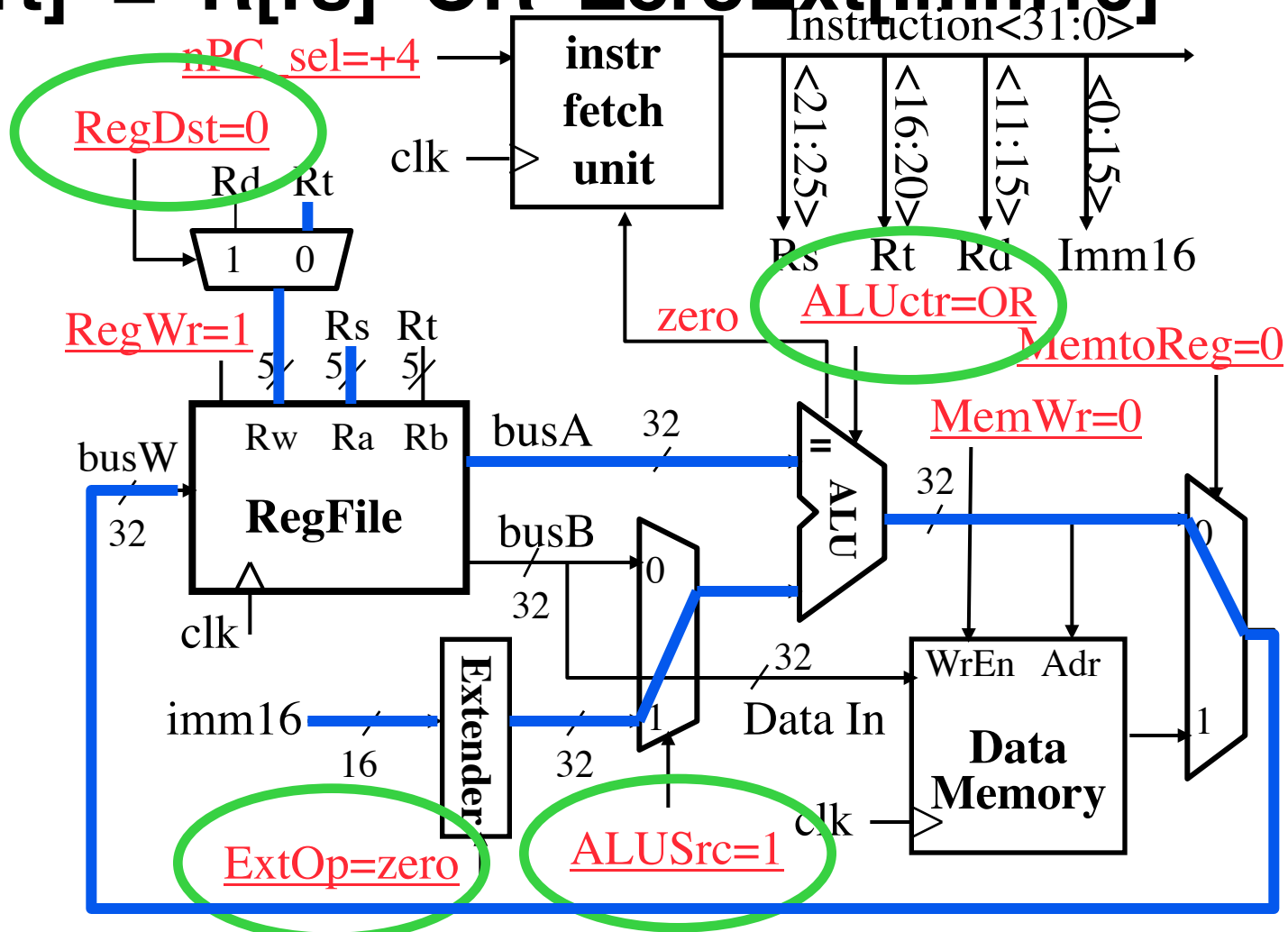
• $R[rt] = R[rs] \text{ OR } \text{ZeroExt}[\text{Imm16}]$



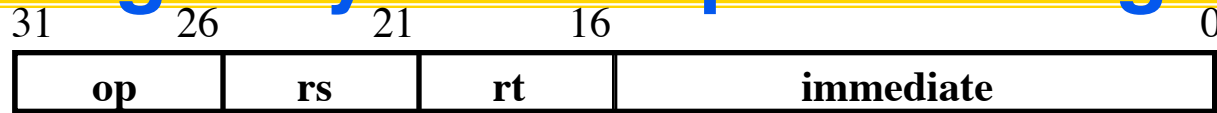
Single Cycle Datapath during Or Immediate?



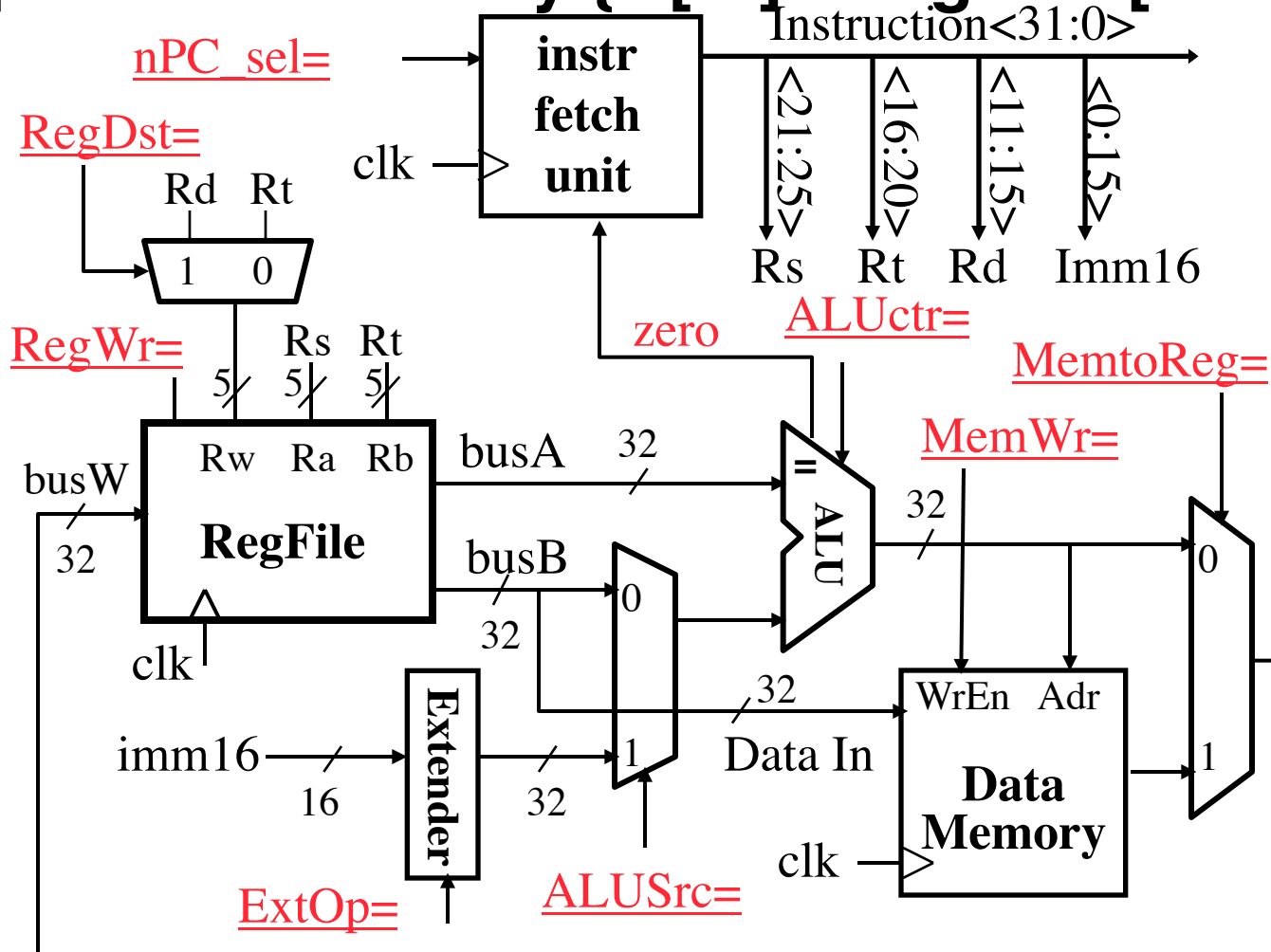
• $R[rt] = R[rs] \text{ OR } \text{ZeroExt}[\text{Imm16}]$



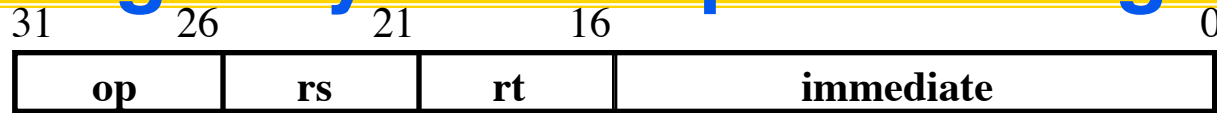
The Single Cycle Datapath during Load?



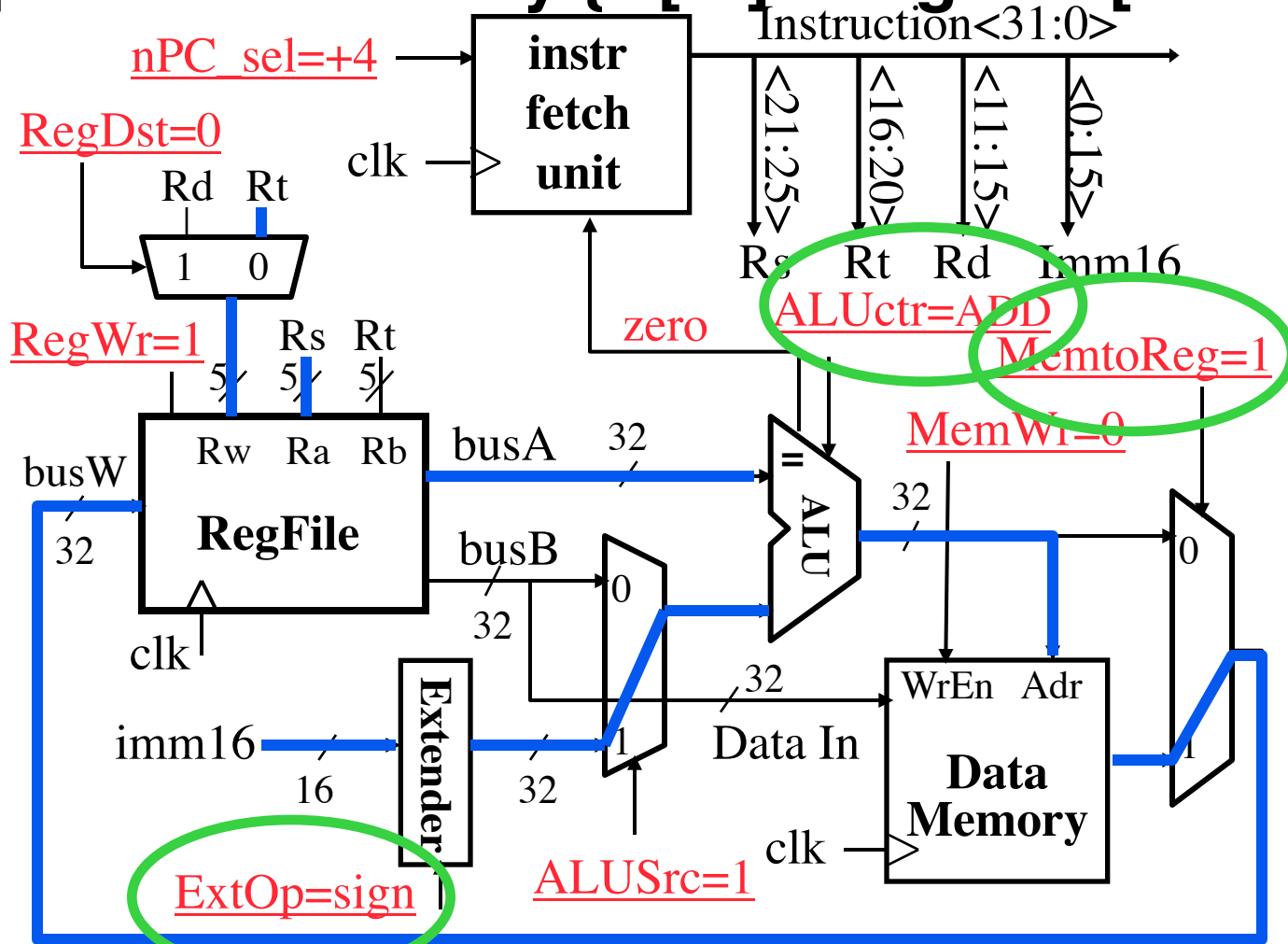
- $R[rt] = \text{Data Memory} \{R[rs] + \text{SignExt}[\text{imm16}]\}$



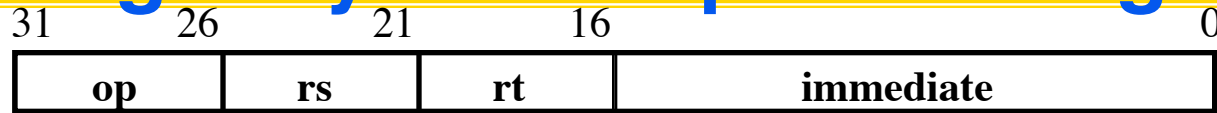
The Single Cycle Datapath during Load



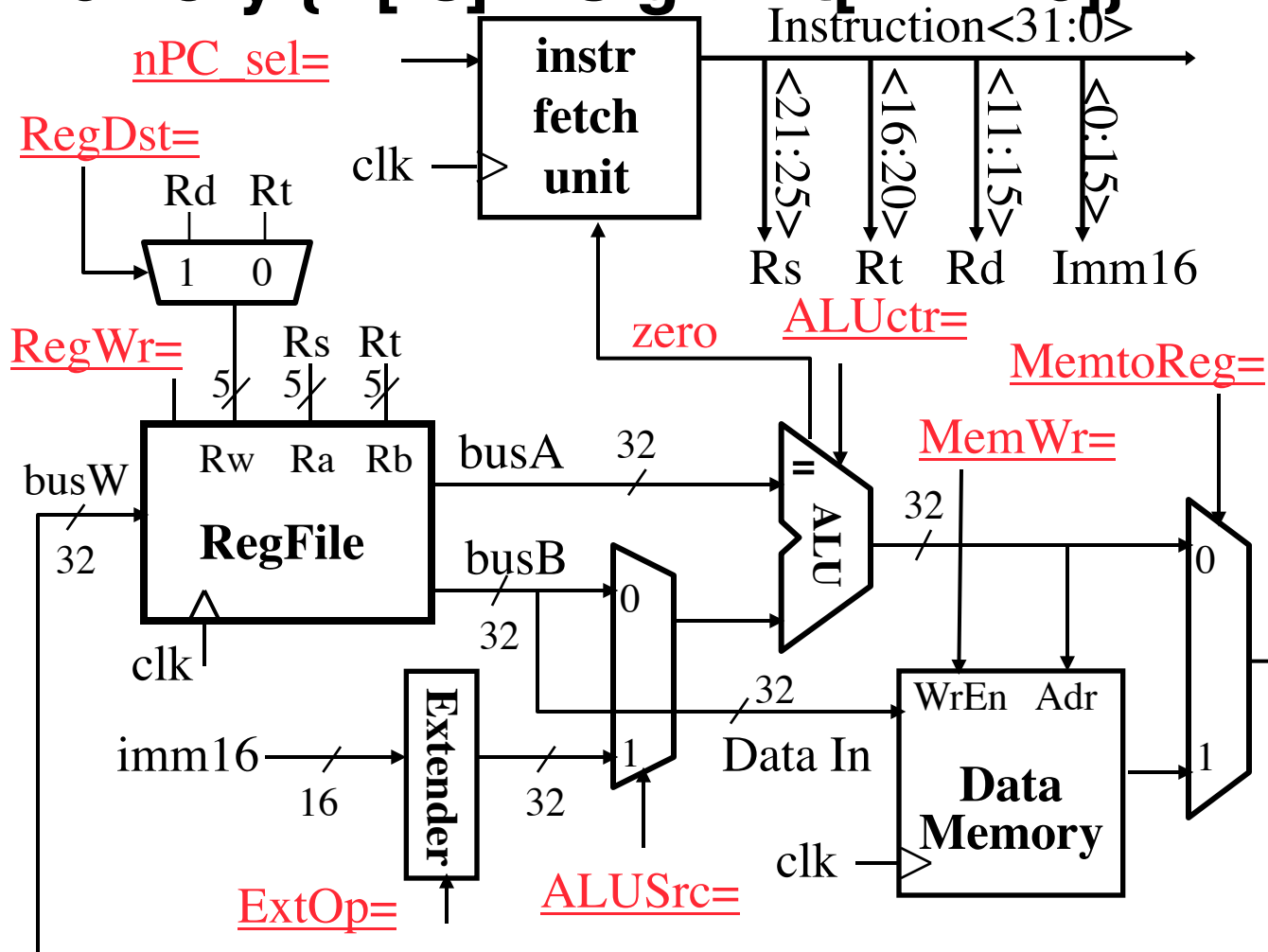
- $R[rt] = \text{Data Memory} \{R[rs] + \text{SignExt}[\text{imm16}]\}$



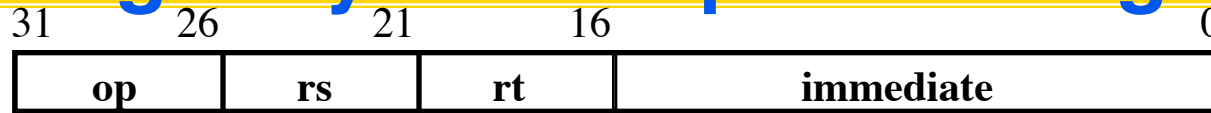
The Single Cycle Datapath during Store?



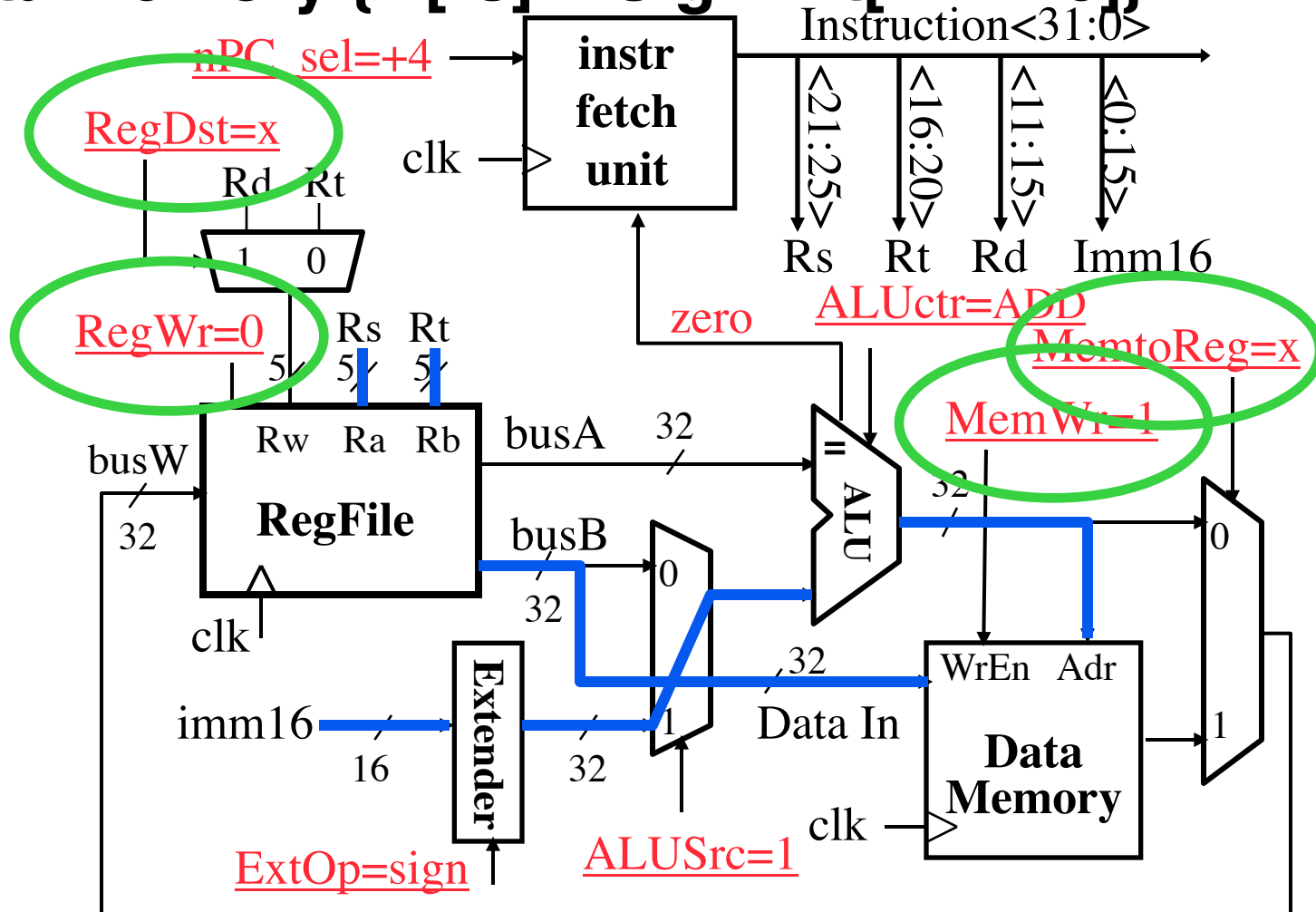
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



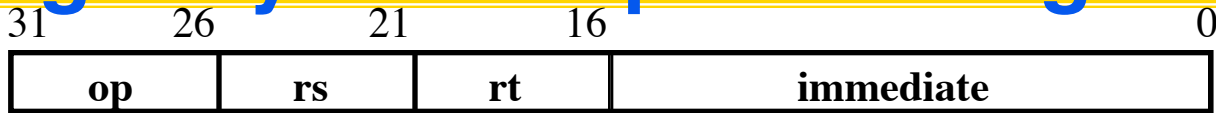
The Single Cycle Datapath during Store



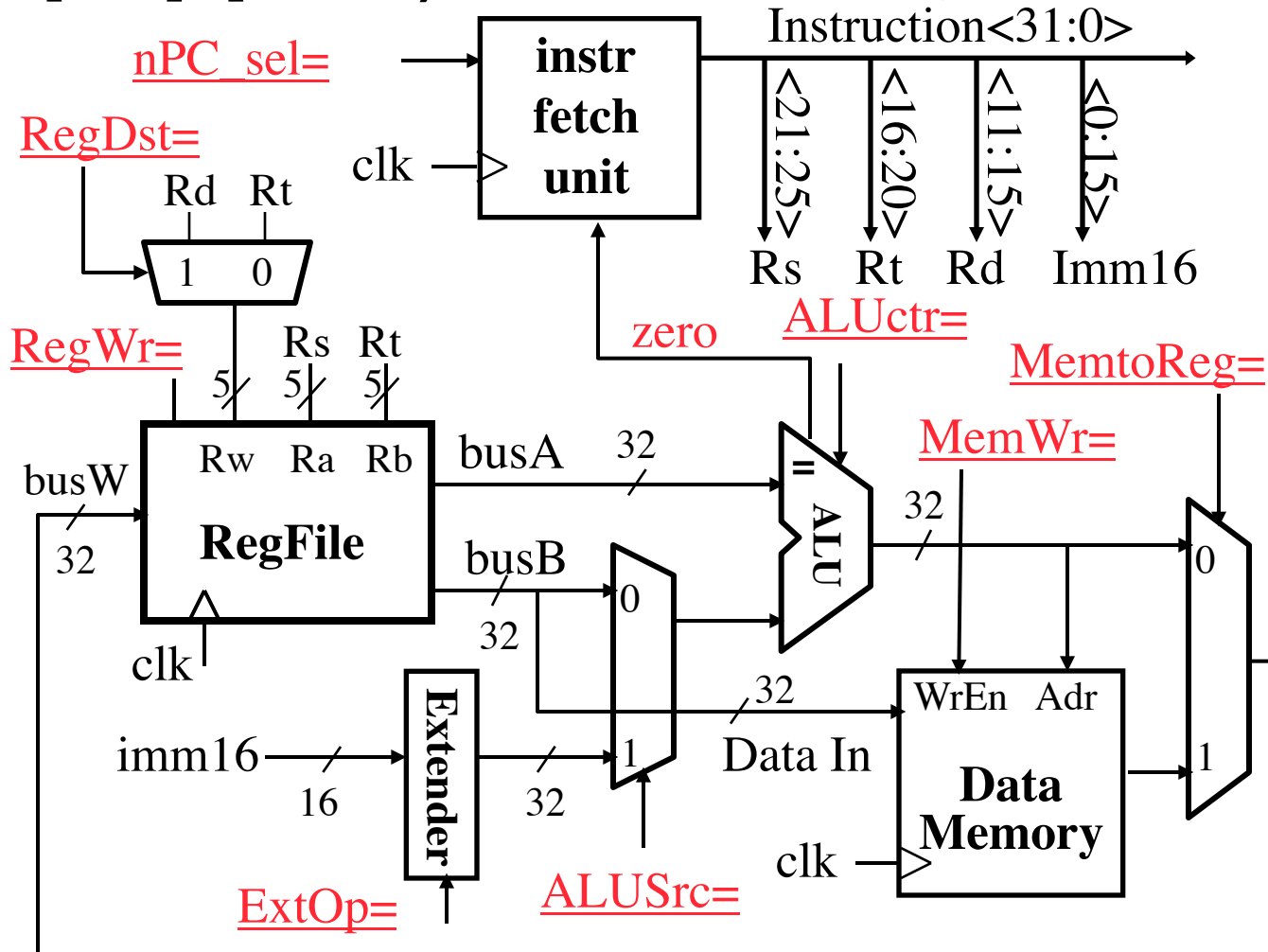
- **Data Memory {R[rs] + SignExt[imm16]} = R[rt]**



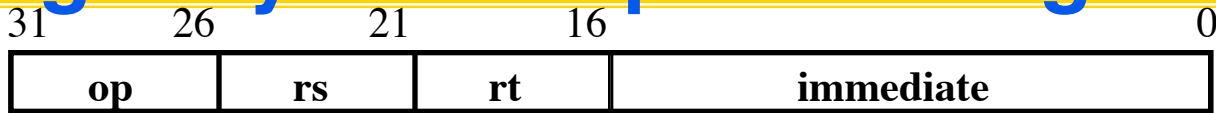
The Single Cycle Datapath during Branch?



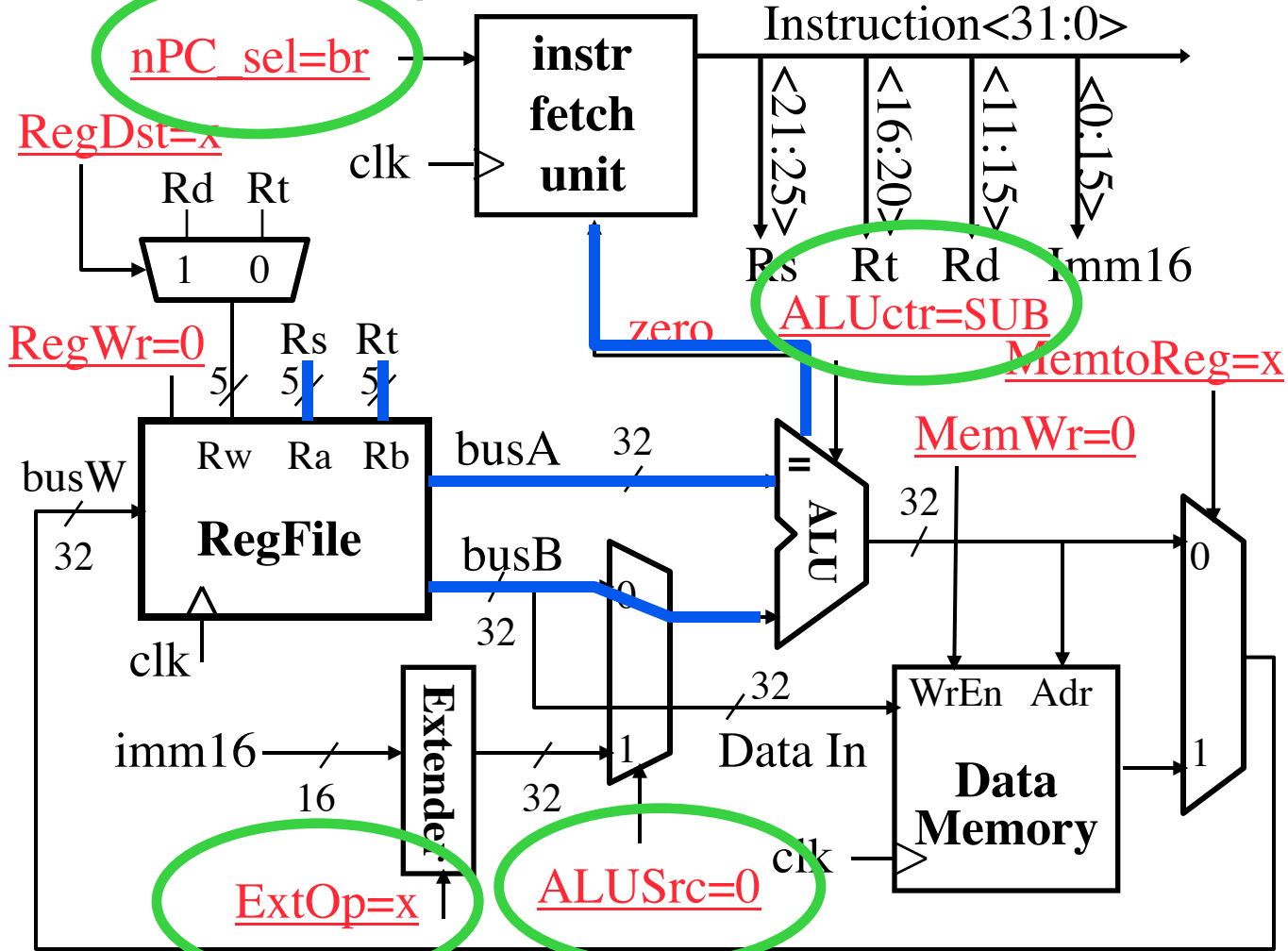
- if $(R[rs] - R[rt] == 0)$ then Zero = 1 ; else Zero = 0



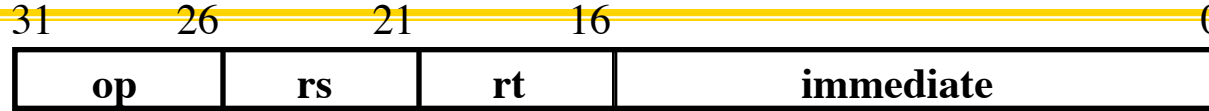
The Single Cycle Datapath during Branch



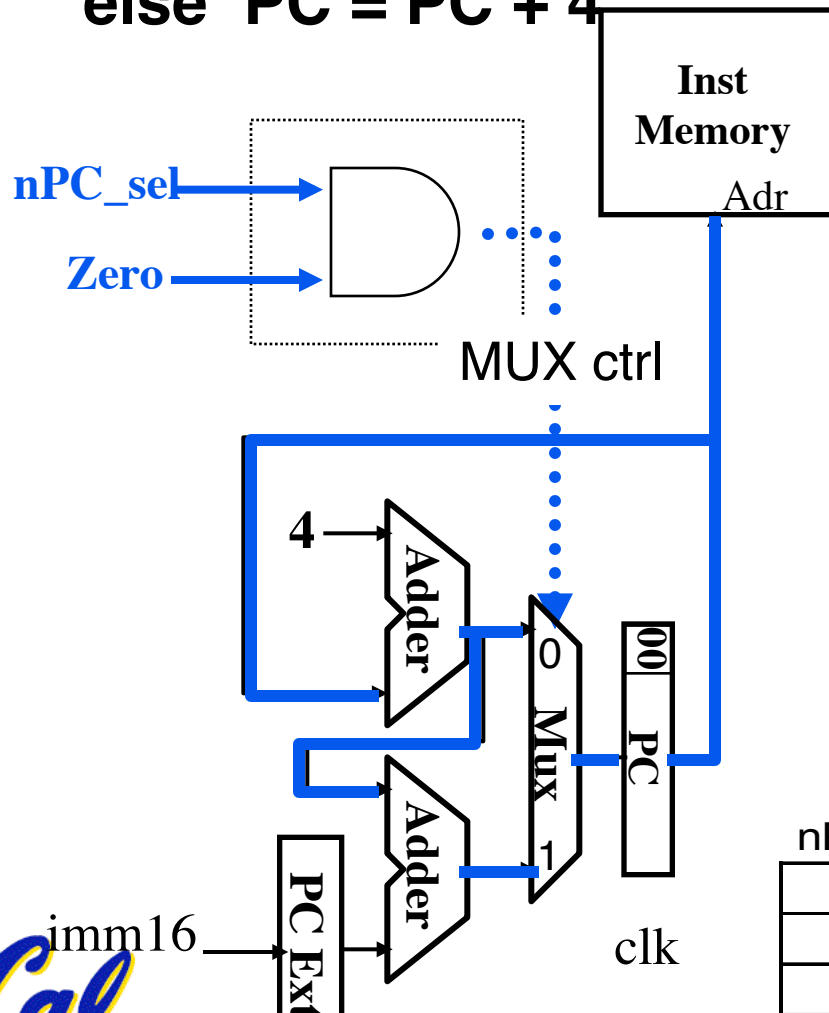
- if $(R[rs] - R[rt] == 0)$ then Zero = 1 ; else Zero = 0



Instruction Fetch Unit at the End of Branch



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



- What is encoding of nPC_sel?
 - Direct MUX select?
 - Branch inst. / not branch
- Let's pick 2nd option

nPC_sel	zero?	MUX
0	x	0
1	0	0
1	1	1

Q: What logic gate?

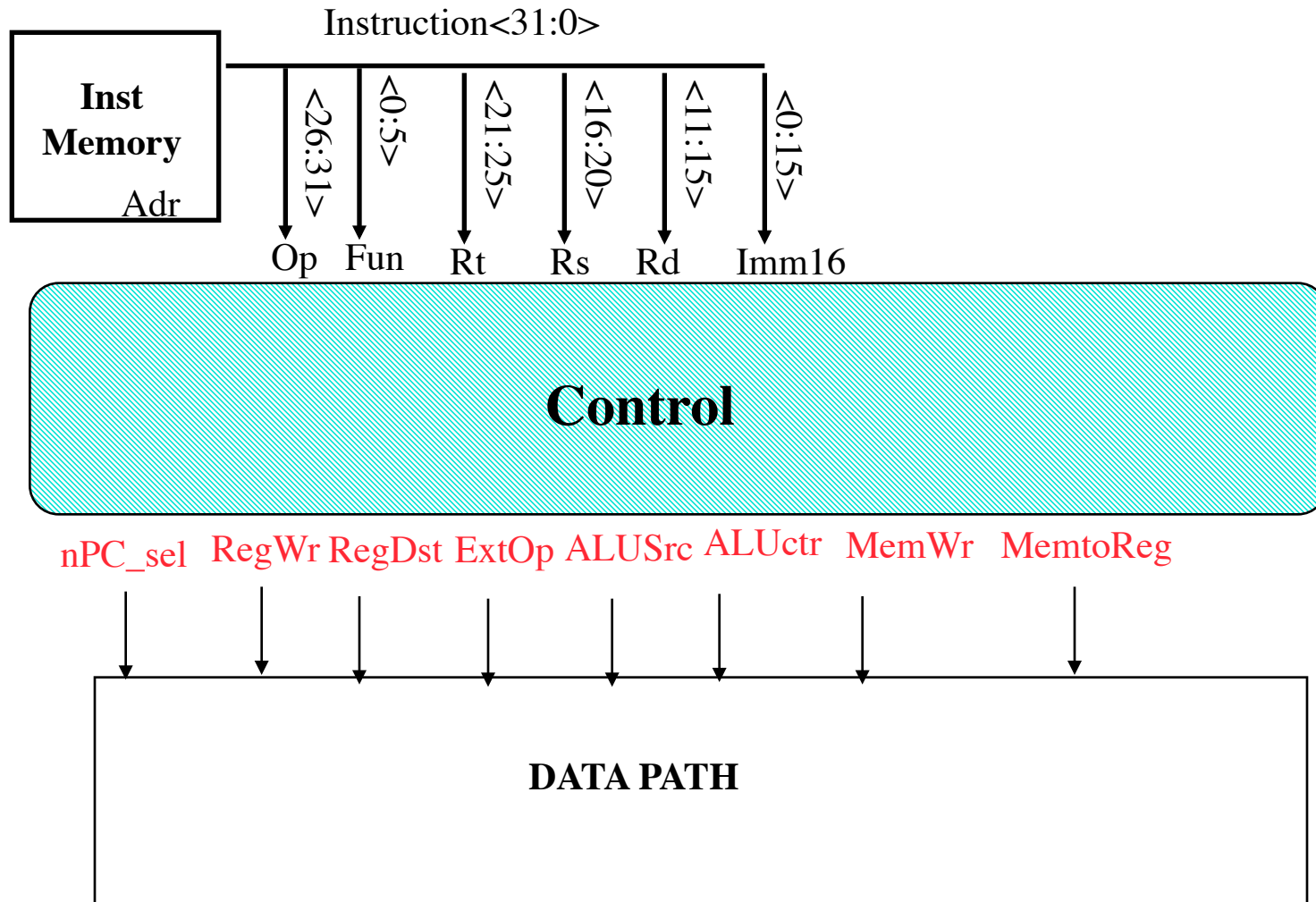


Administrivia

- **Sorry about Proj1 woes**
- **Grading is rough stuff. Don't blame Ben, he's innocent.**
- **HW6 Due imminently! Students have claimed it takes a very long time**
 - **Remember MODULAR DESIGN. This could save you a lot of time.**
- **HW7 Out Now! Get started soon.**
- **Proj3 is on its way, will be out soon after the weekend.**



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 = RegB, ALUctr = "ADD", RegDst = rd, RegWr, nPC_sel = "+4"

sub $R[rd] \leftarrow R[rs] - R[rt];$ $PC \leftarrow PC + 4$
ALUsrc = RegB, ALUctr = "SUB", RegDst = rd, RegWr, nPC_sel = "+4"

ori $R[rt] \leftarrow R[rs] + \text{zero_ext}(\text{Imm16});$ $PC \leftarrow PC + 4$
ALUsrc = Im, Extop = "Z", ALUctr = "OR", RegDst = rt, RegWr, nPC_sel = "+4"

lw $R[rt] \leftarrow \text{MEM}[R[rs] + \text{sign_ext}(\text{Imm16})];$ $PC \leftarrow PC + 4$
*ALUsrc = Im, Extop = "sn", ALUctr = "ADD",
MemtoReg, RegDst = rt, RegWr, nPC_sel = "+4"*

sw $\text{MEM}[R[rs] + \text{sign_ext}(\text{Imm16})] \leftarrow R[rs];$ $PC \leftarrow PC + 4$
ALUsrc = Im, Extop = "sn", ALUctr = "ADD", MemWr, nPC_sel = "+4"

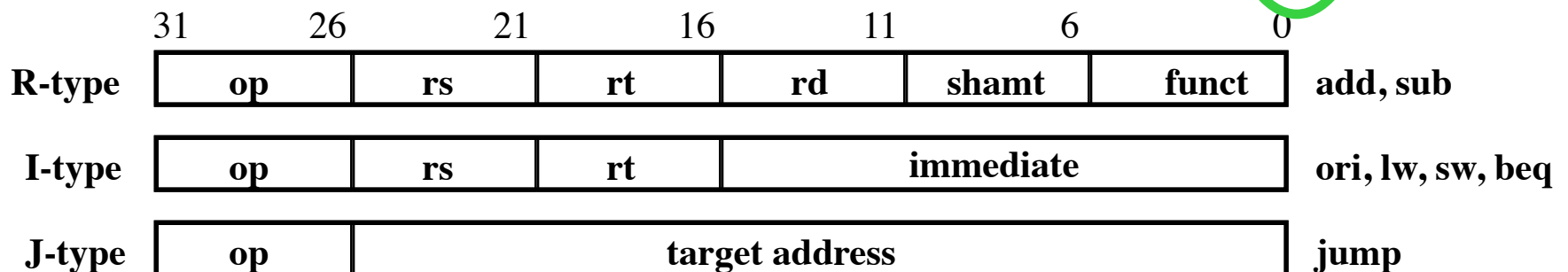
beq if ($R[rs] == R[rt]$) then $PC \leftarrow PC + \text{sign_ext}(\text{Imm16}) \parallel 00$ else $PC \leftarrow PC + 4$
nPC_sel = "br", ALUctr = "SUB"



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	?
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	x



Boolean Expressions for Controller

RegDst = add + sub

ALUSrc = ori + lw + sw

MemtoReg = lw

RegWrite = add + sub + ori + lw

MemWrite = sw

nPCsel = beq

Jump = jump

ExtOp = lw + sw

ALUctr[0] = sub + beq (assume ALUctr is 0 ADD, 01: SUB, 10: OR)

ALUctr[1] = or

where,

rtype = $\sim op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot \sim op_2 \cdot \sim op_1 \cdot \sim op_0$,

ori = $\sim op_5 \cdot \sim op_4 \cdot op_3 \cdot op_2 \cdot \sim op_1 \cdot op_0$

lw = $op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot \sim op_2 \cdot op_1 \cdot op_0$

sw = $op_5 \cdot \sim op_4 \cdot op_3 \cdot \sim op_2 \cdot op_1 \cdot op_0$

beq = $\sim op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot op_2 \cdot \sim op_1 \cdot \sim op_0$

jump = $\sim op_5 \cdot \sim op_4 \cdot \sim op_3 \cdot \sim op_2 \cdot op_1 \cdot \sim op_0$

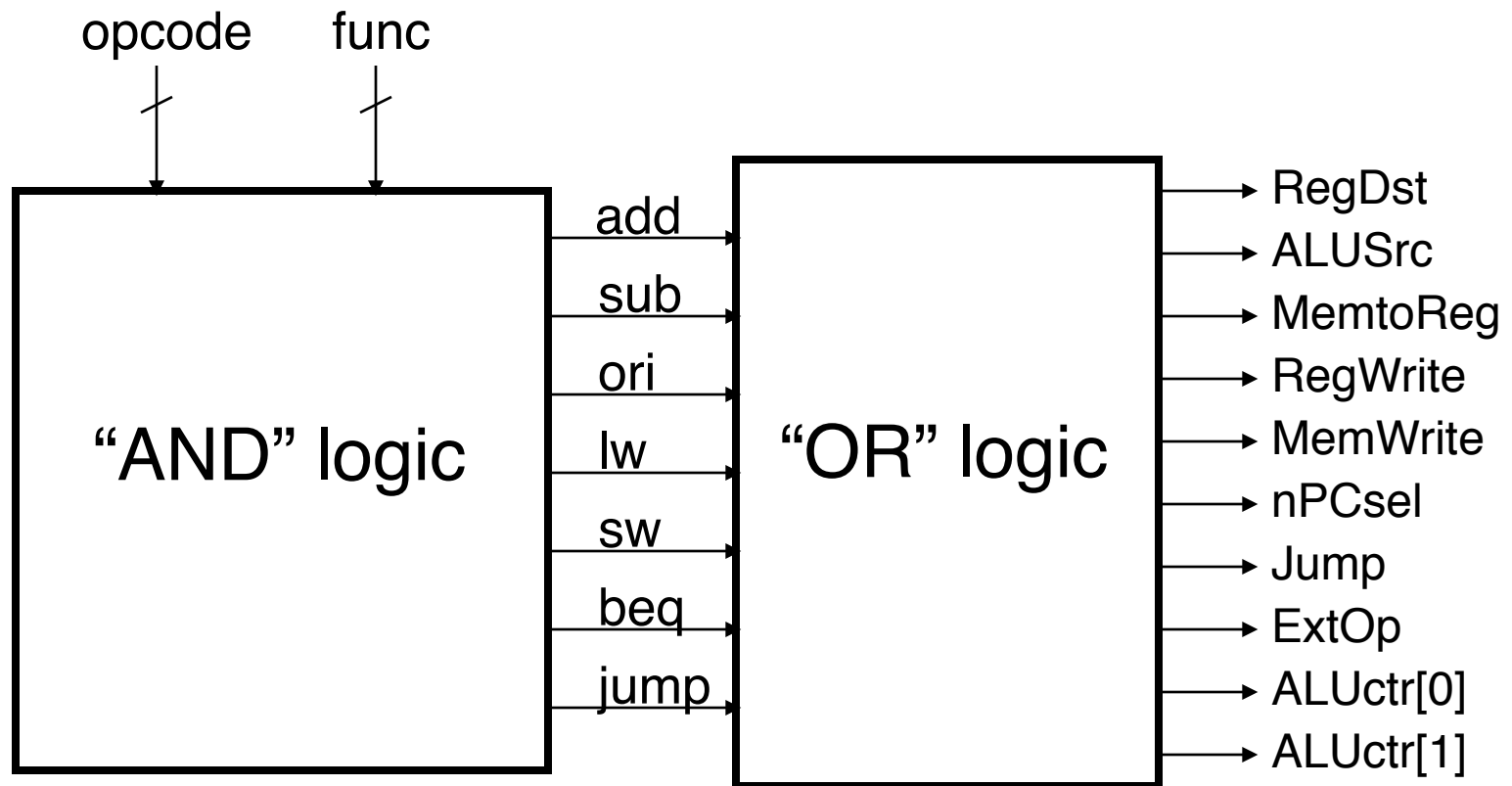
add = $rtype \cdot func_5 \cdot \sim func_4 \cdot \sim func_3 \cdot \sim func_2 \cdot \sim func_1 \cdot \sim func_0$

sub = $rtype \cdot func_5 \cdot \sim func_4 \cdot \sim func_3 \cdot \sim func_2 \cdot func_1 \cdot \sim func_0$

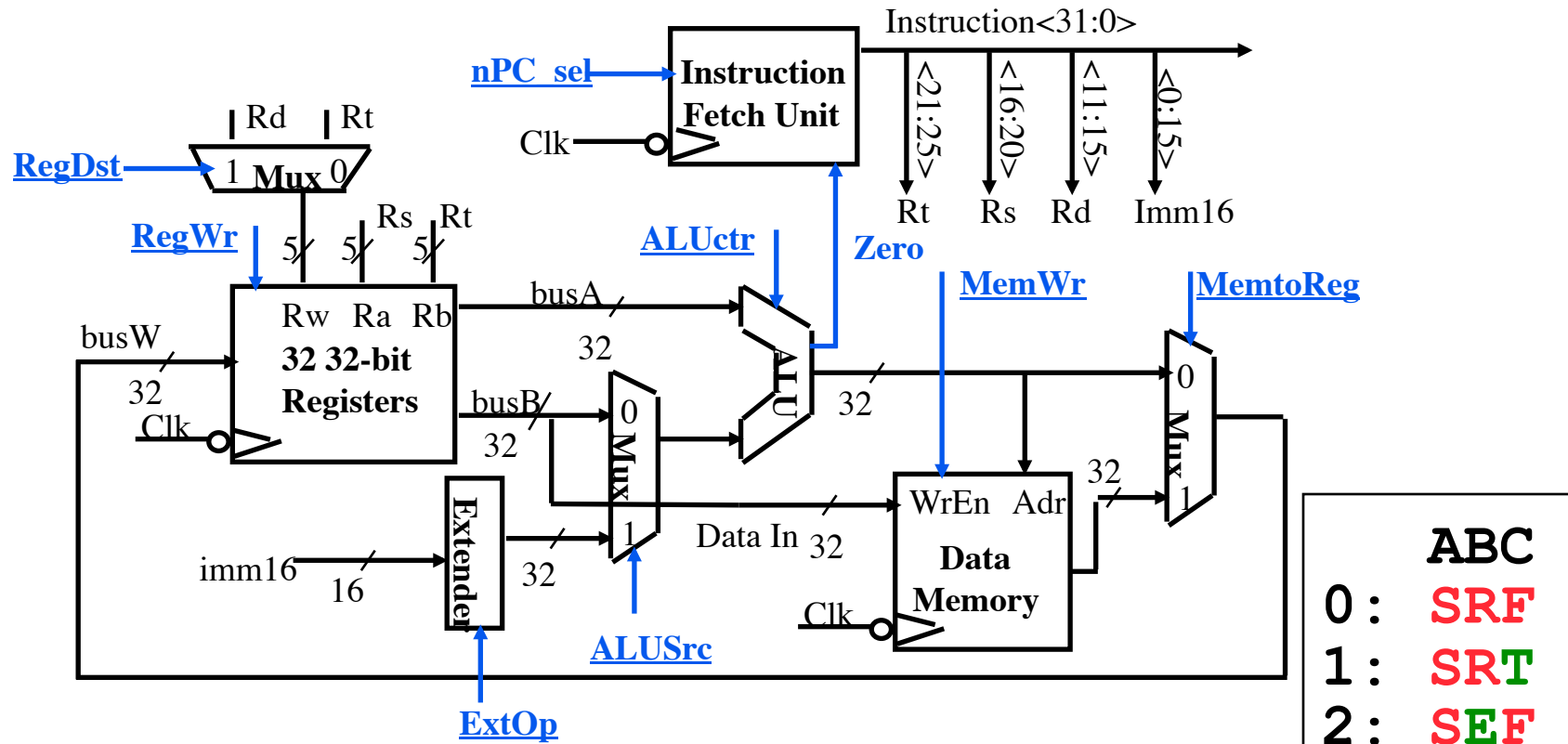
How do we
implement this in
gates?



Controller Implementation



Peer Instruction



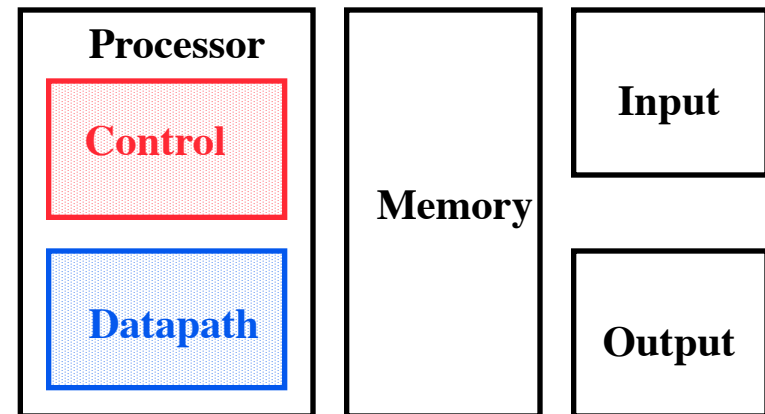
- A. MemToReg='x' & ALUctr='sub'. **SUB** or **BEQ**?
- B. ALUctr='add'. Which 1 signal is different for all 3 of: ADD, LW, & SW? **RegDst** or **ExtOp**?
- C. "Don't Care" signals are useful because we can simplify our PLA personality matrix. **F / T**?

	ABC
0 :	S R F
1 :	S R T
2 :	S E F
3 :	S E T
4 :	B R F
5 :	B R T
6 :	B E F
7 :	B E T

Summary: Single-cycle Processor

◦ 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
 - Formulate Logic Equations
 - Design Circuits



Bonus slides

- **These are extra slides that used to be included in lecture notes, but have been moved to this, the “bonus” area to serve as a supplement.**
- **The slides will appear in the order they would have in the normal presentation**

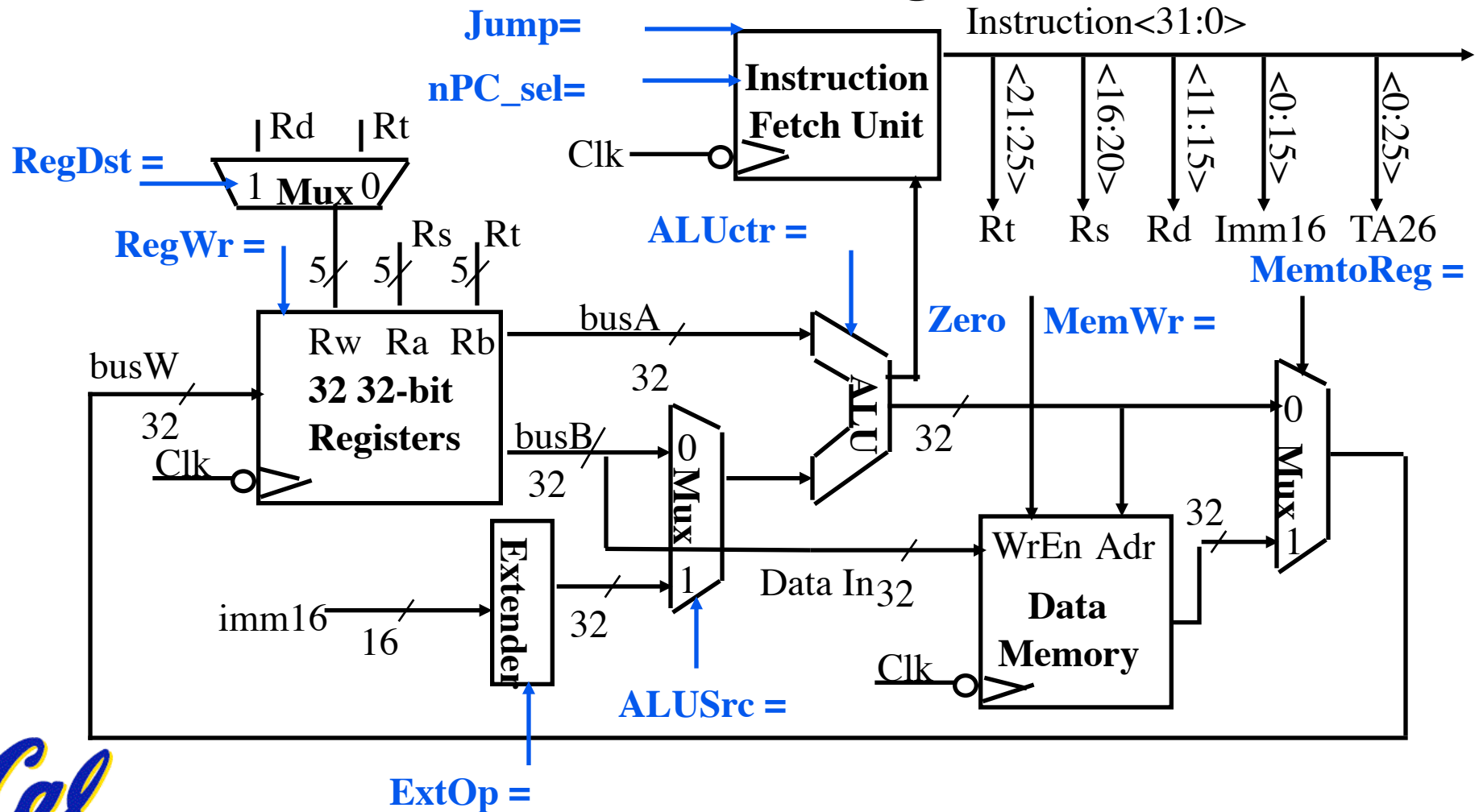
Bonus



The Single Cycle Datapath during Jump



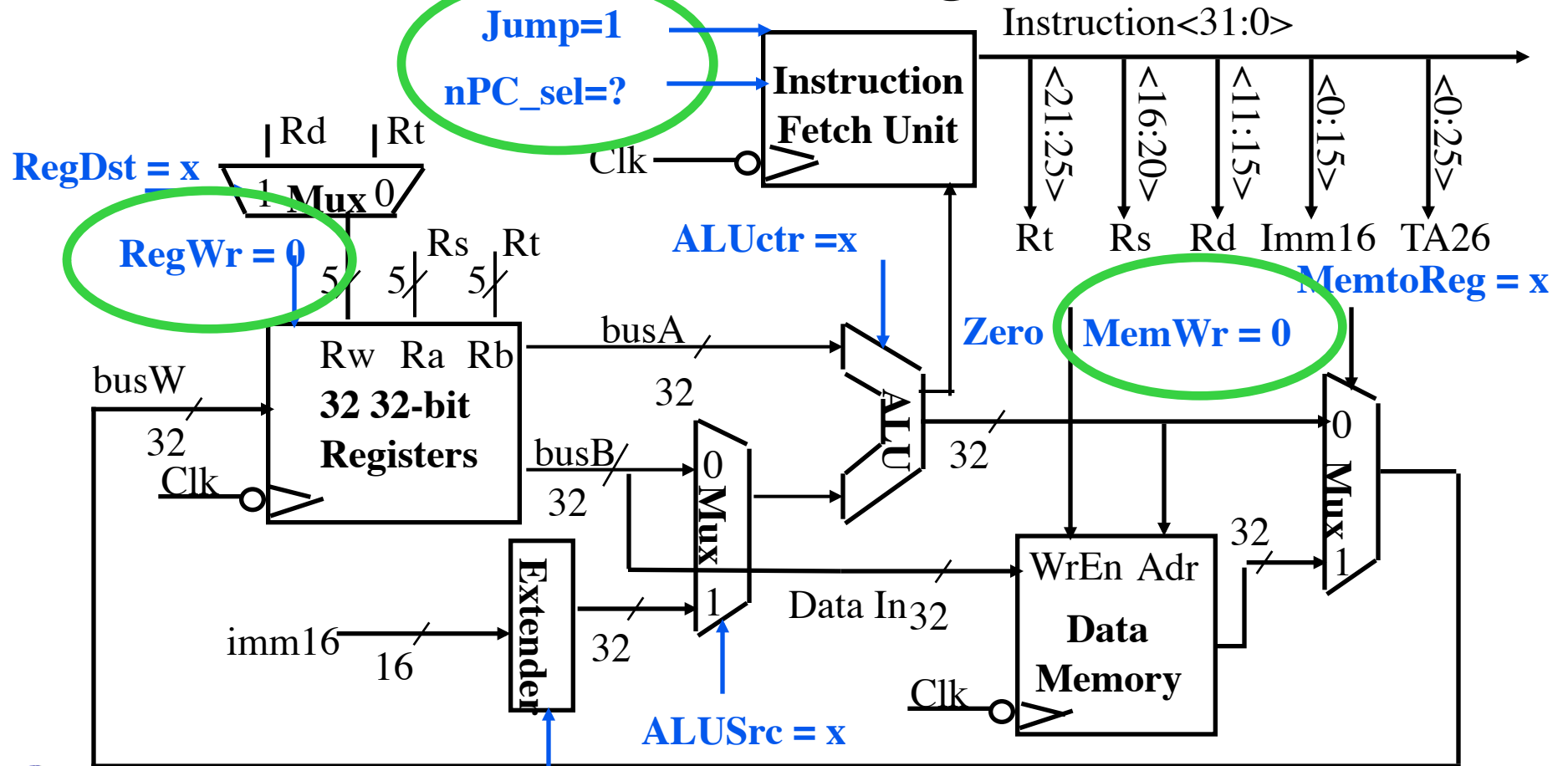
- **New PC = { PC[31..28], target address, 00 }**



The Single Cycle Datapath during Jump



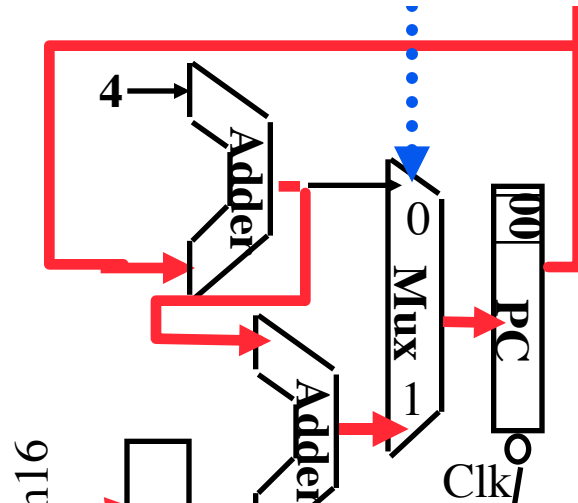
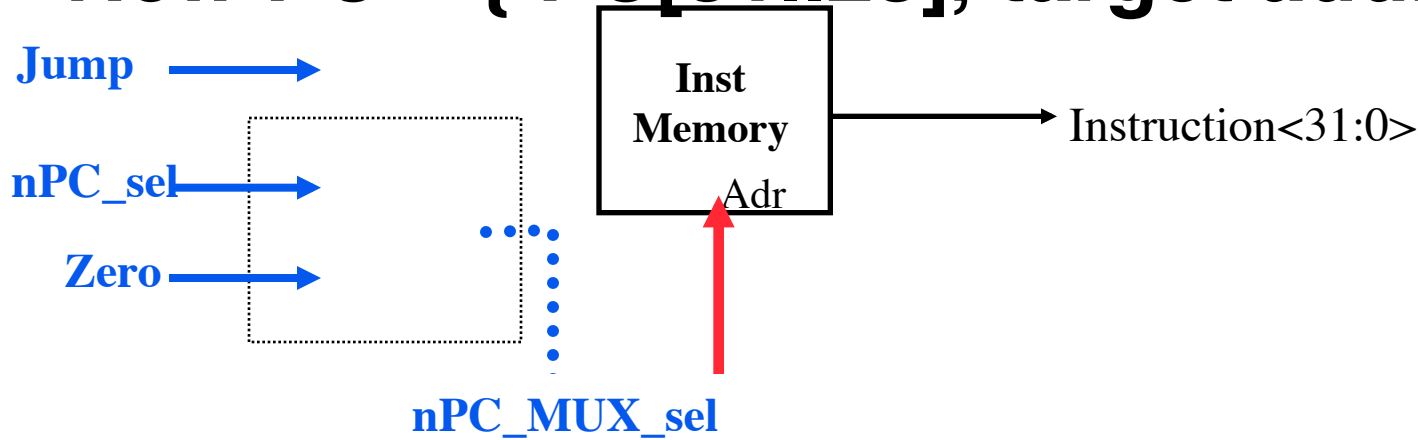
- **New PC = { PC[31..28], target address, 00 }**



Instruction Fetch Unit at the End of Jump



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



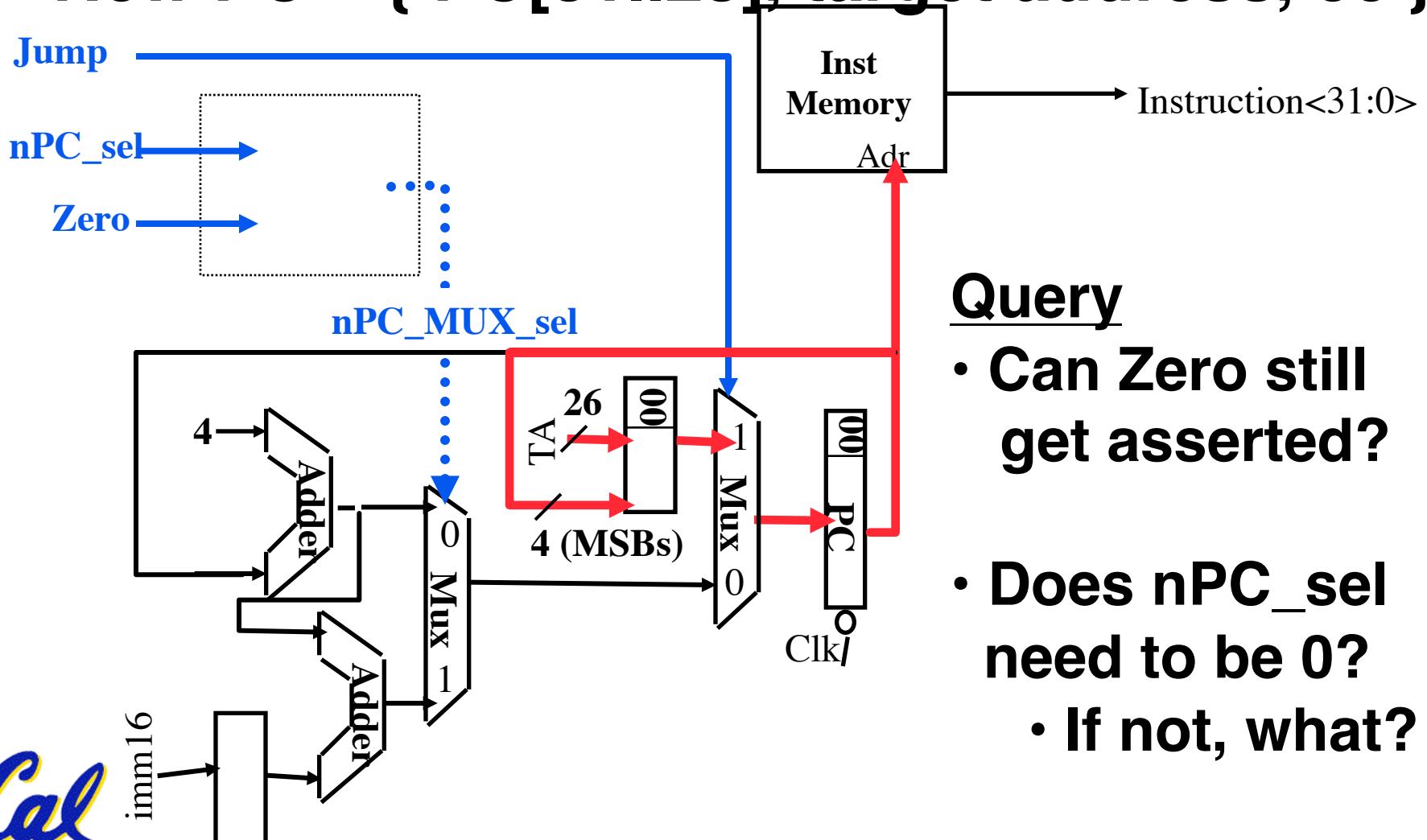
How do we modify this to account for jumps?



Instruction Fetch Unit at the End of Jump



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



Query

- Can Zero still get asserted?
- Does nPC_sel need to be 0?
 - If not, what?

