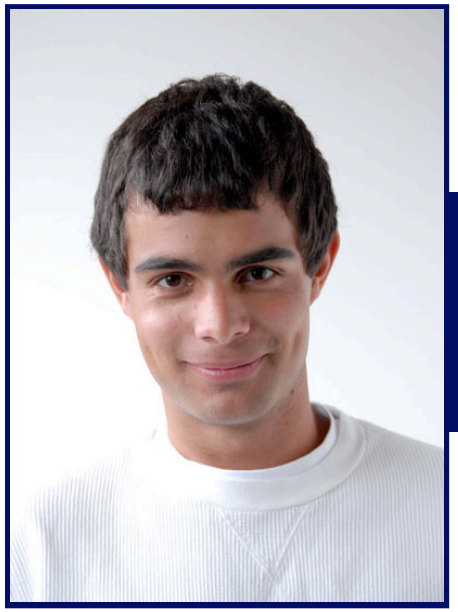
inst.eecs.berkeley.edu/~cs61c

CS61C : Machine Structures

Lecture #20 – Controlling a Single-Cycle CPU



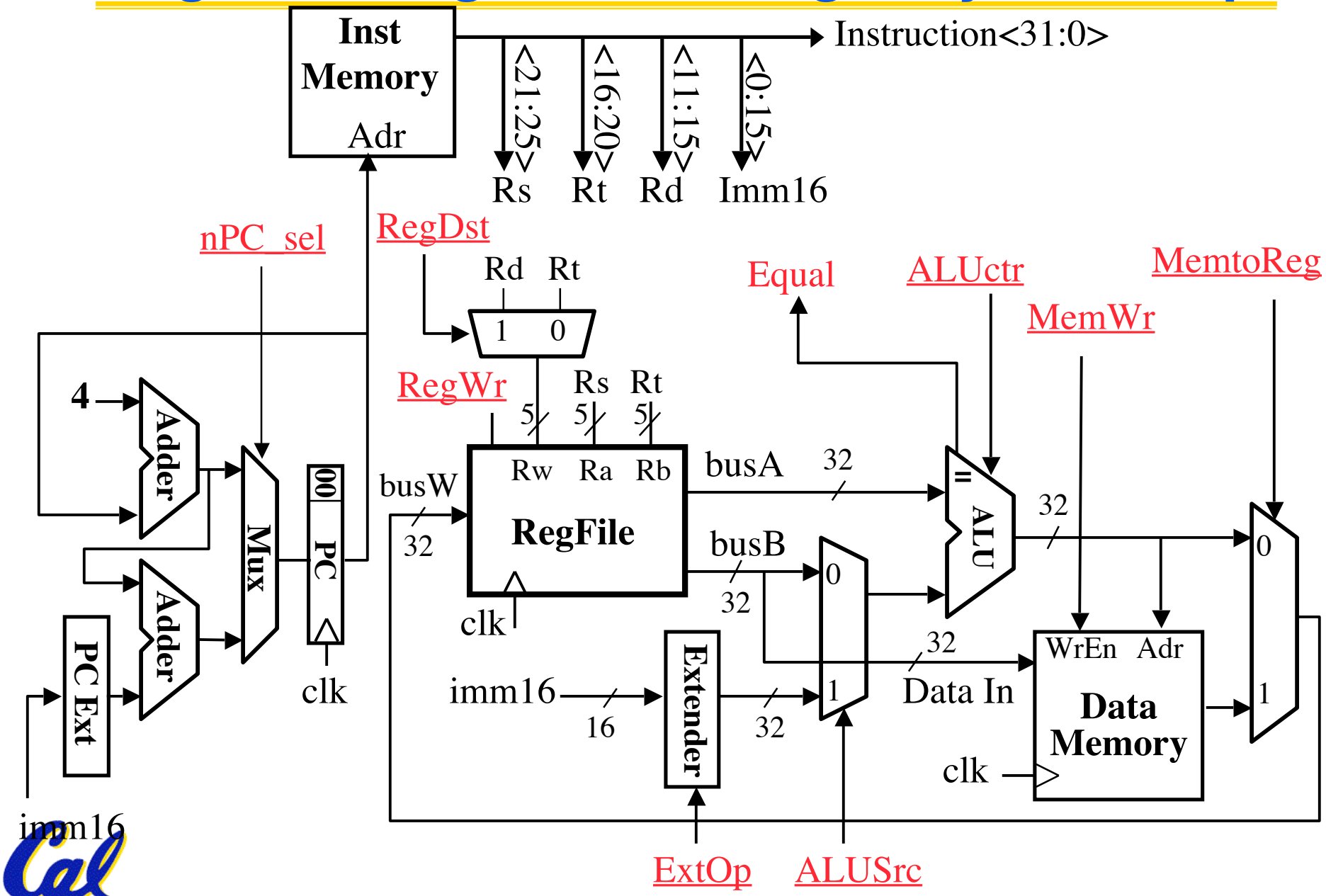
2007-7-30

Scott Beamer

Instructor

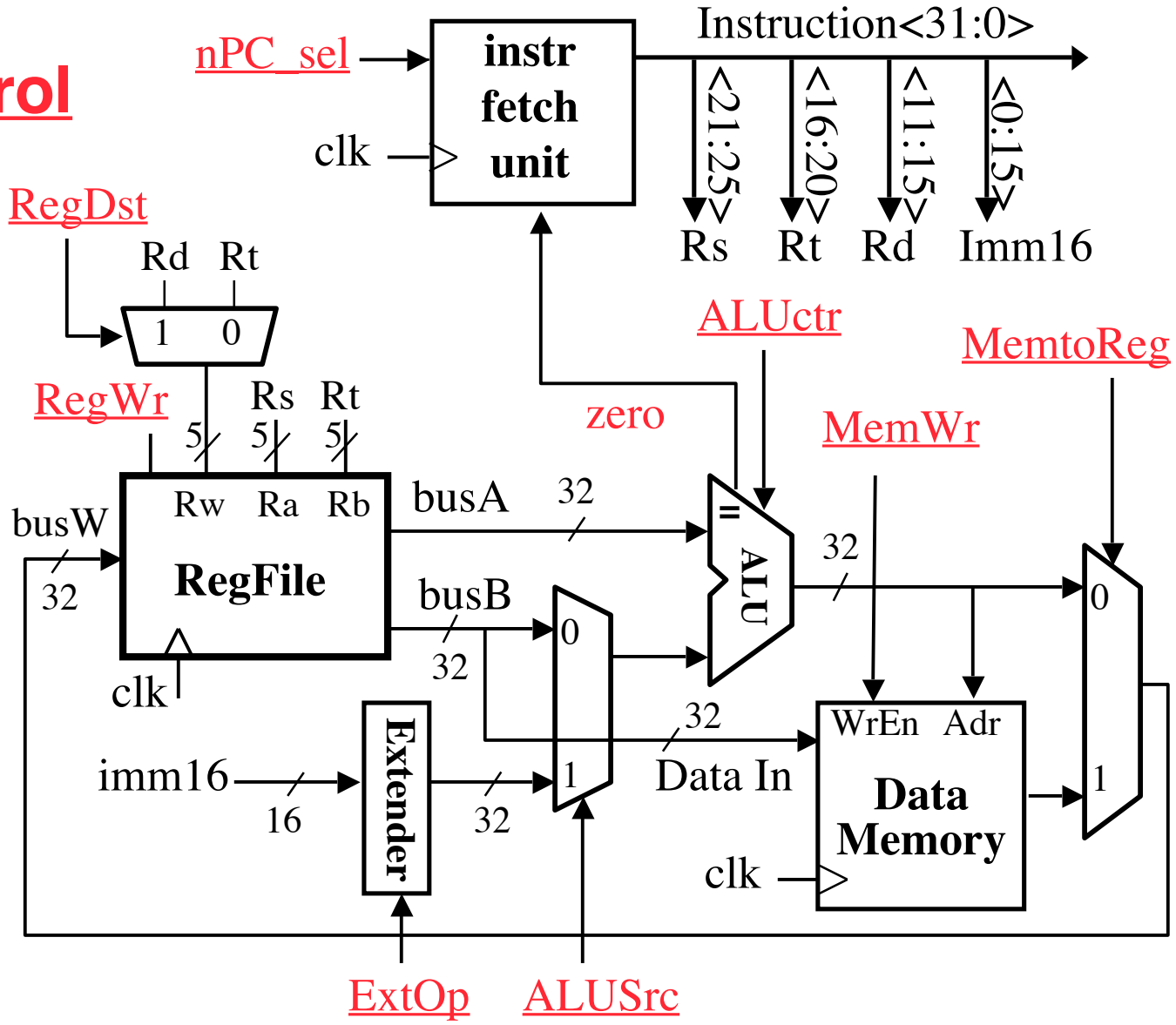
Black Hat Conference Kicks Off in Vegas

Putting it All Together:A Single Cycle Datapath

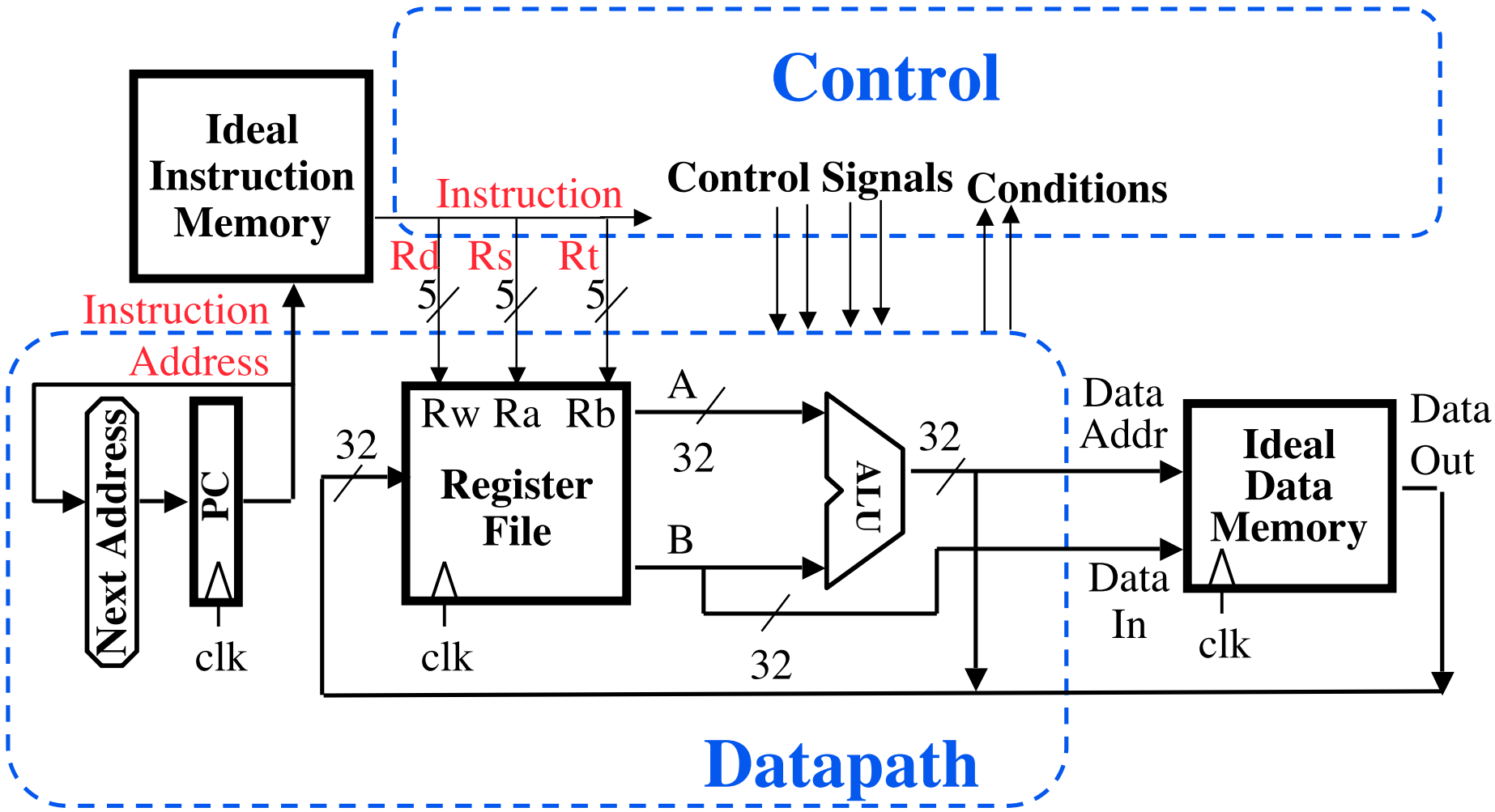


Review: A Single Cycle Datapath

- We have everything except control signals



An Abstract View of the Implementation



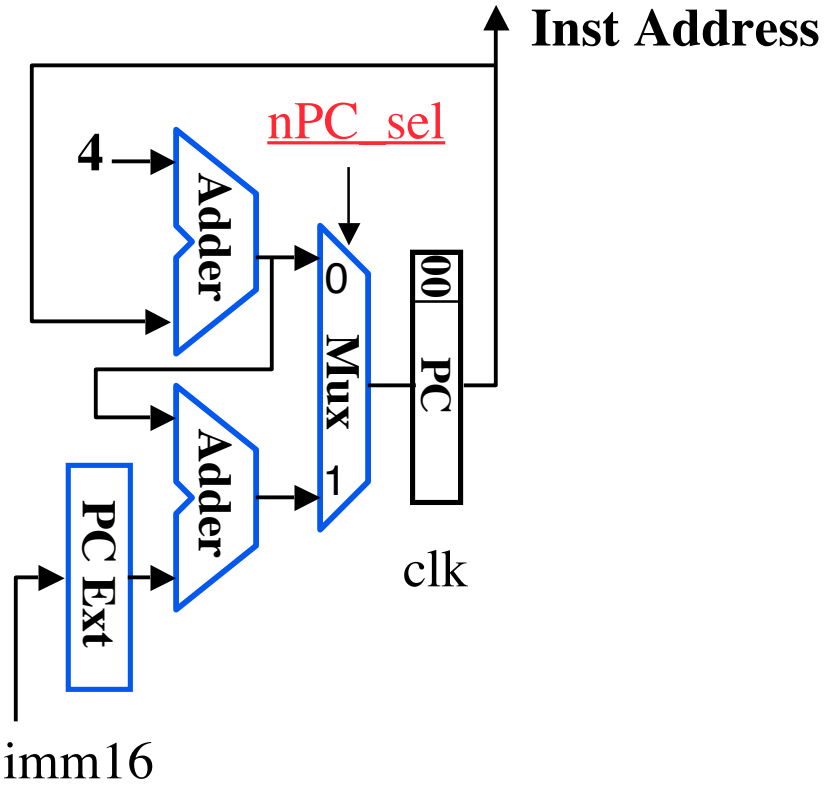
Recap: Meaning of the Control Signals

- nPC_sel:

  "n"=next

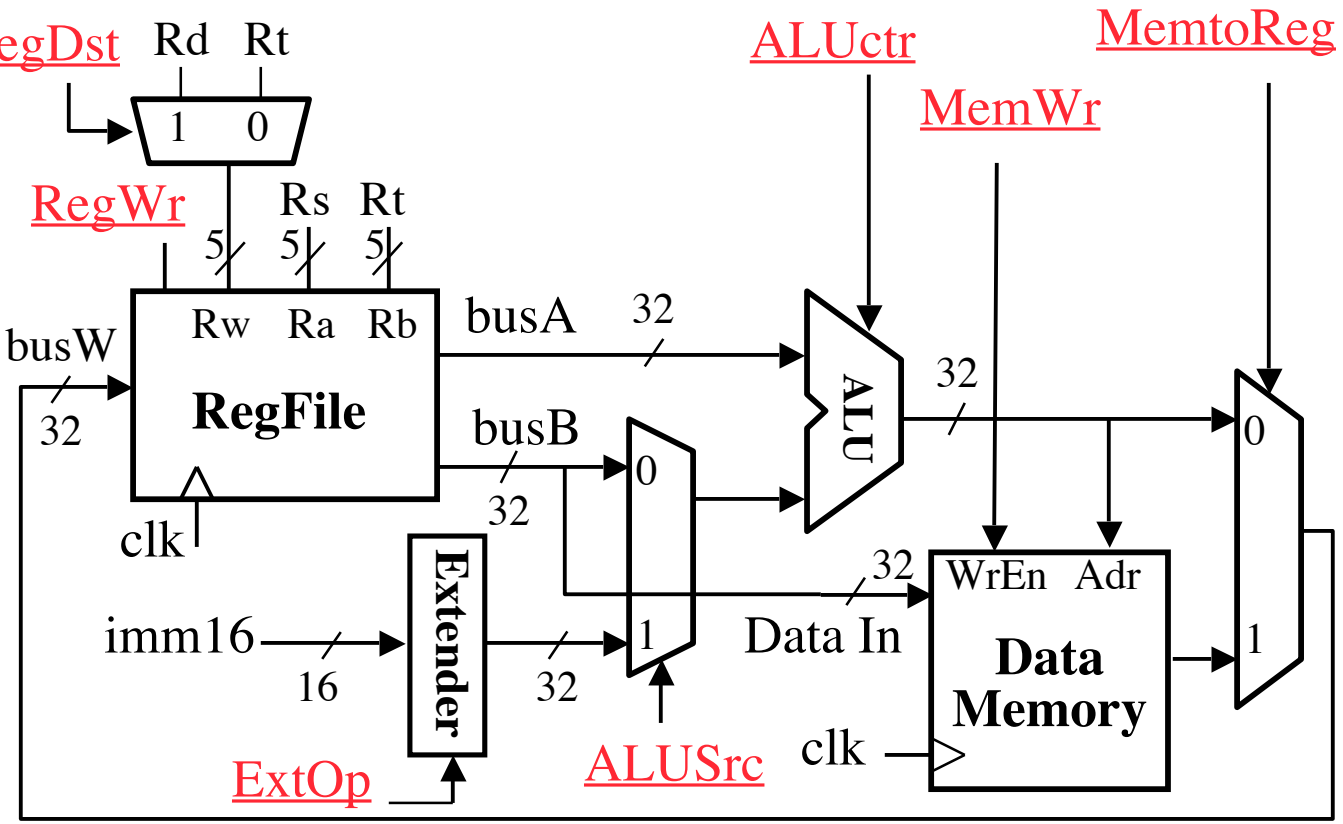
  "+4" 0 ⇒ PC <– PC + 4
  "br" 1 ⇒ PC <– PC + 4 + {SignExt(Im16) , 00 }

- Later in lecture: higher-level connection between mux and branch condition

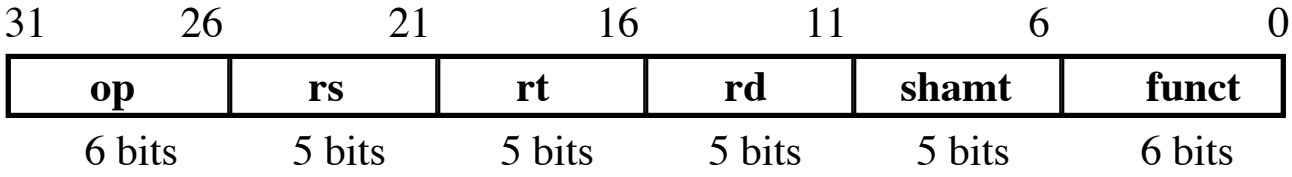


Recap: Meaning of the Control Signals

- **ExtOp:** "zero", "sign"
- **ALUsrc:** 0 ⇒ regB; 1 ⇒ immed
- **ALUctr:** "ADD", "SUB", "OR"
- **MemWr:** 1 ⇒ write memory
- **MemtoReg:** 0 ⇒ ALU; 1 ⇒ Mem
- **RegDst:** 0 ⇒ "rt"; 1 ⇒ "rd"
- **RegWr:** 1 ⇒ write register



RTL: The `Add` Instruction

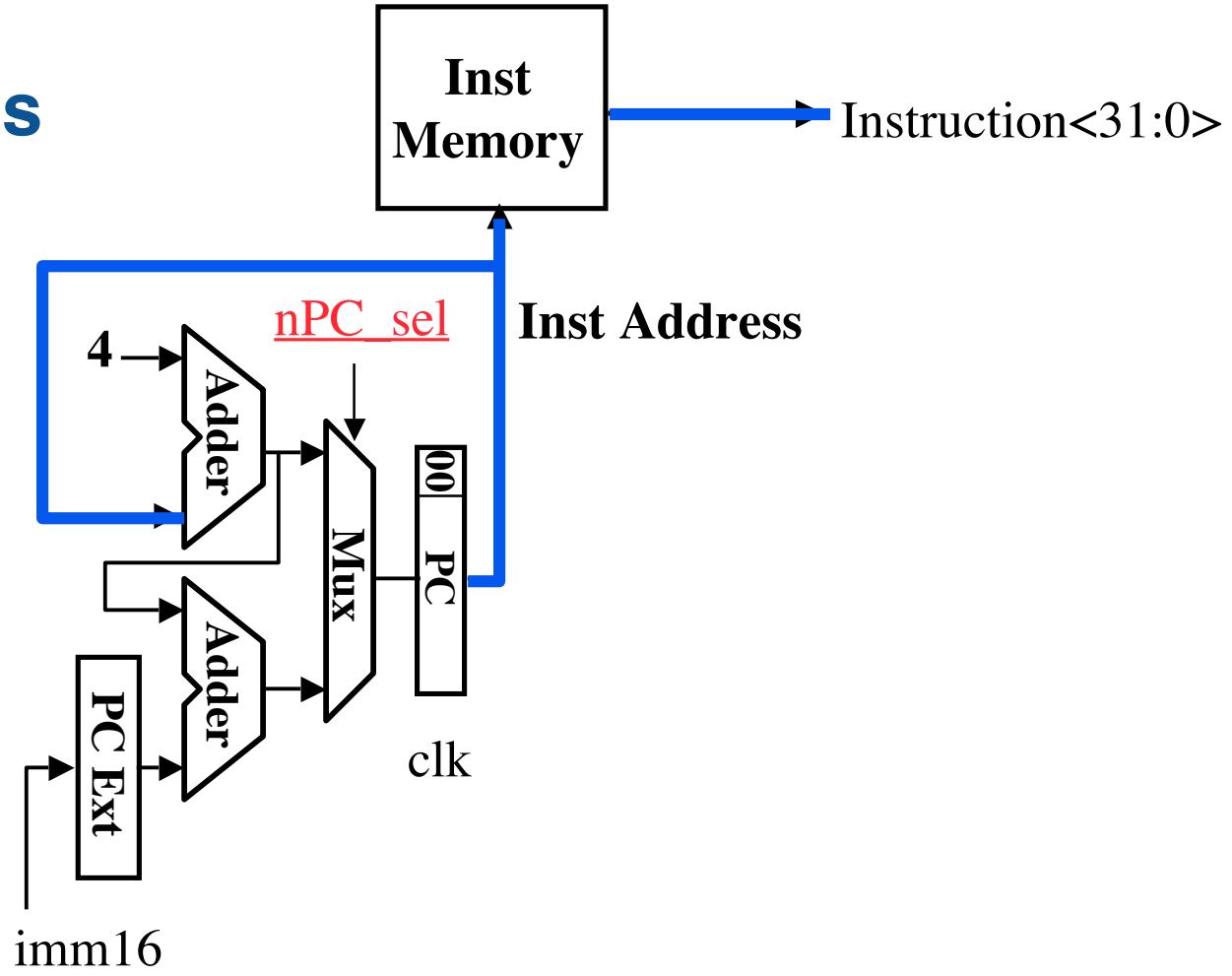



```
add rd, rs, rt
```

- 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

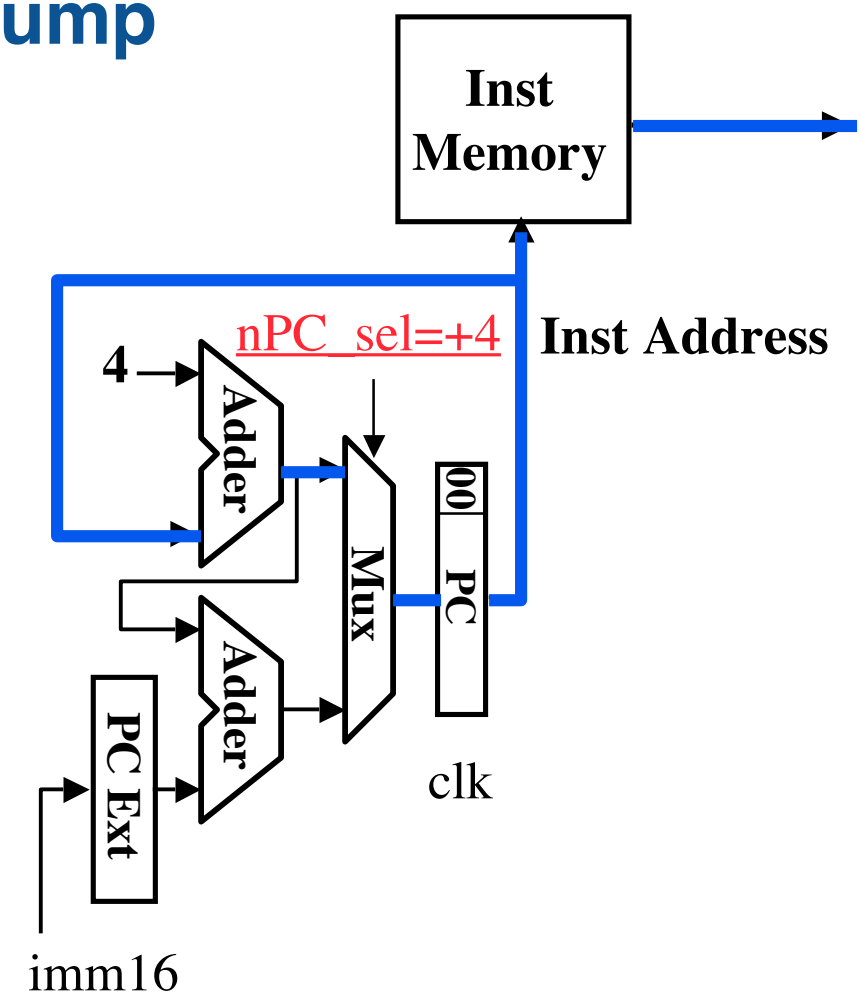
Instruction Fetch Unit at the Beginning of Add

- Fetch the instruction from Instruction memory: Instruction = MEM[PC]
  - same for all instructions

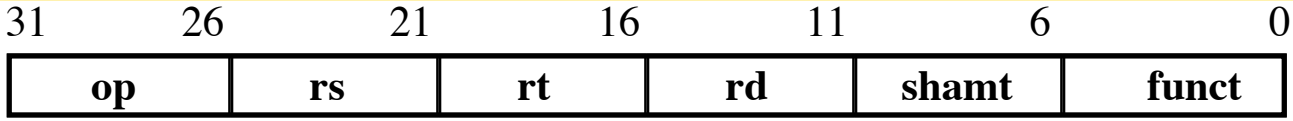


Instruction Fetch Unit at the End of `Add`

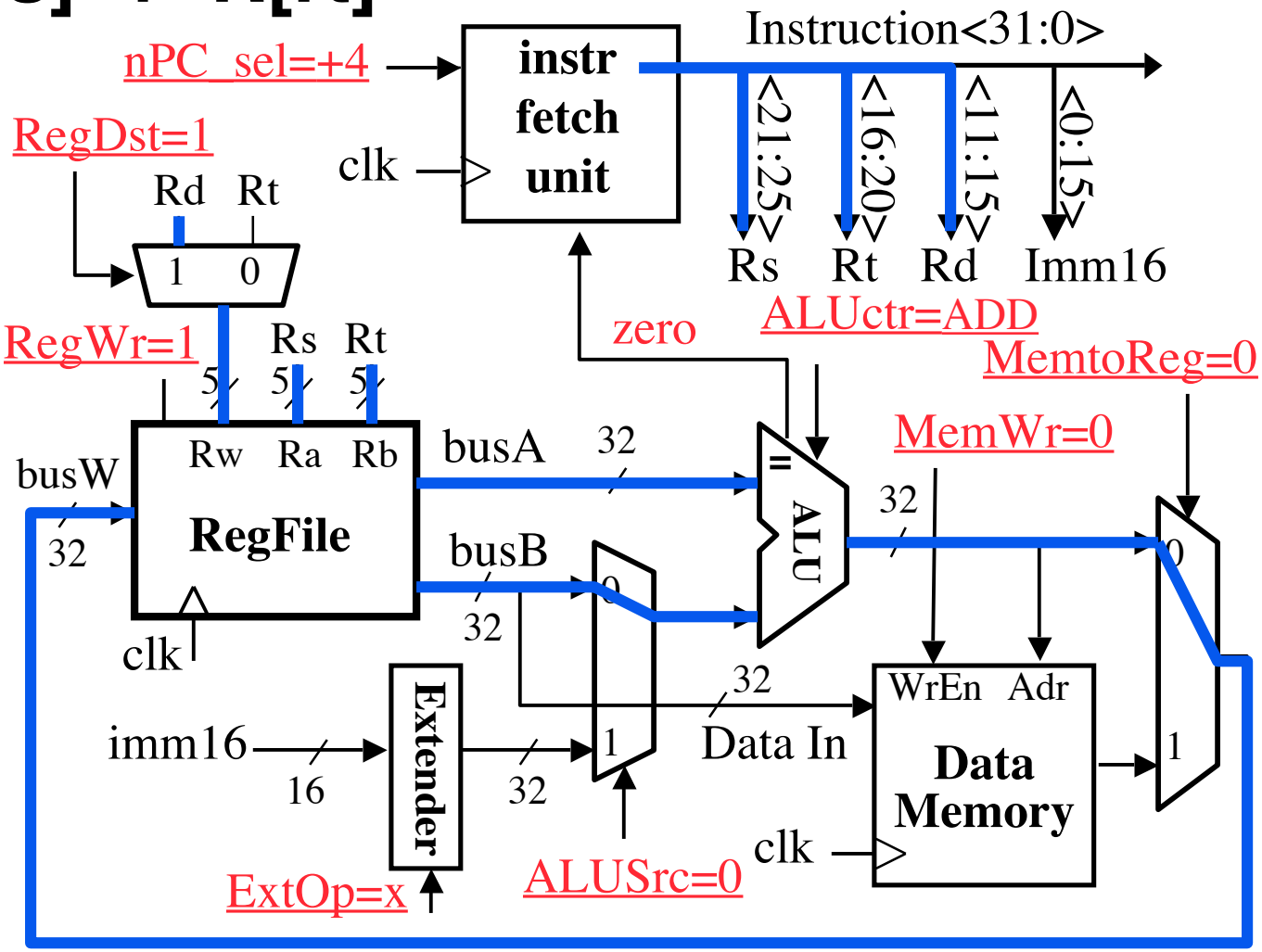
- PC = PC + 4
  - This is the same for all instructions except: Branch and Jump



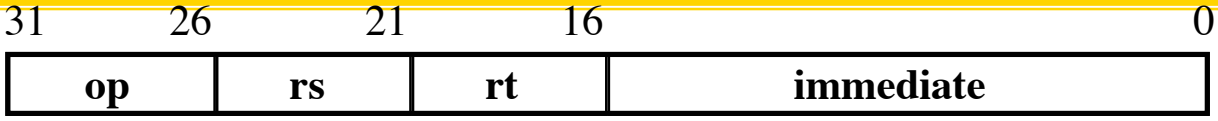
The Single Cycle Datapath during Add



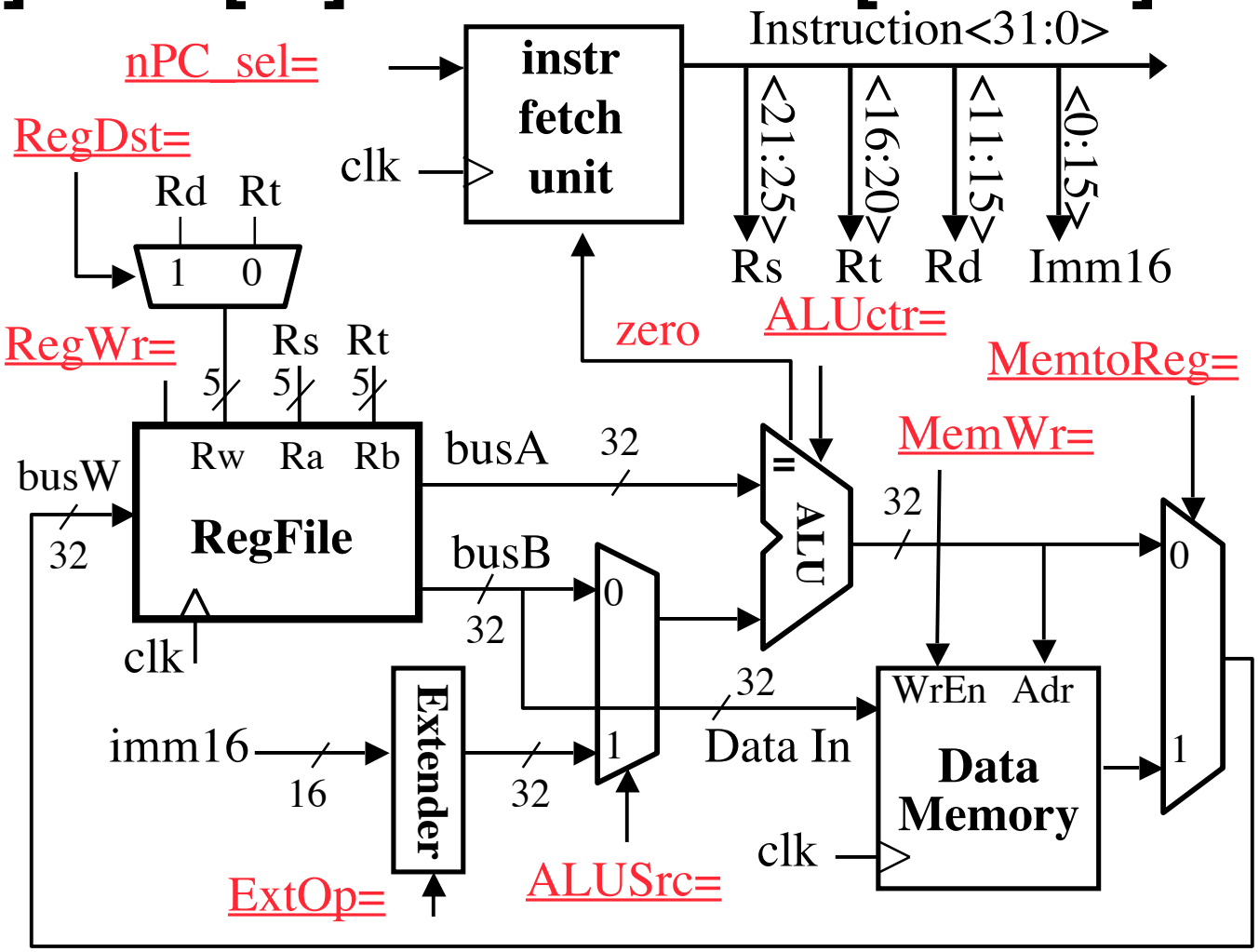
R[rd] = R[rs] + R[rt]



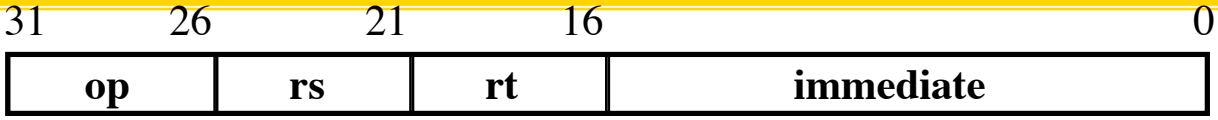
Single Cycle Datapath during Or Immediate?



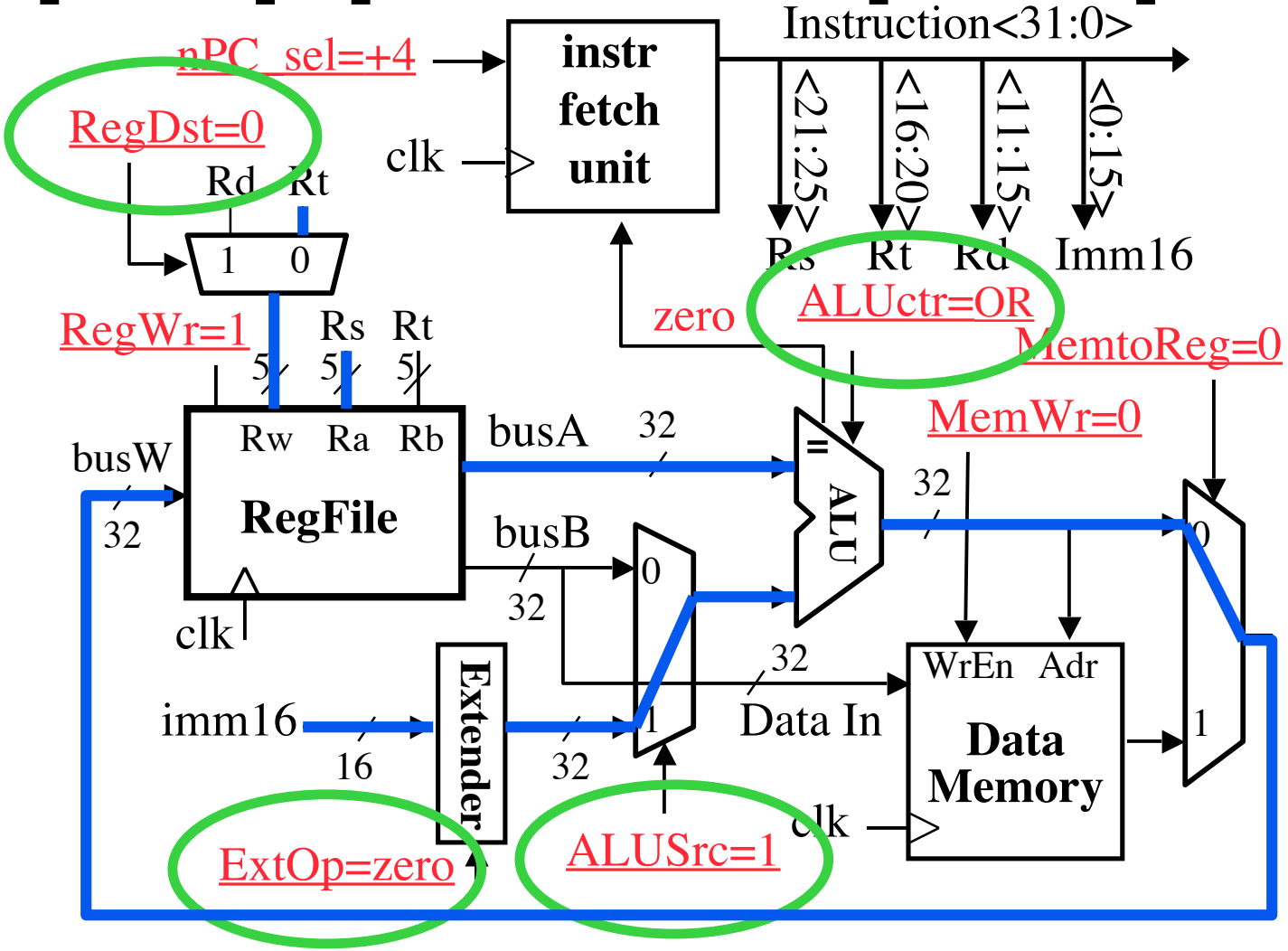
- R[rt] = R[rs] OR ZeroExt[Imm16]



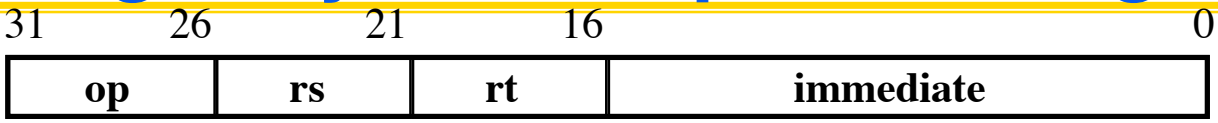
Single Cycle Datapath during Or Immediate?



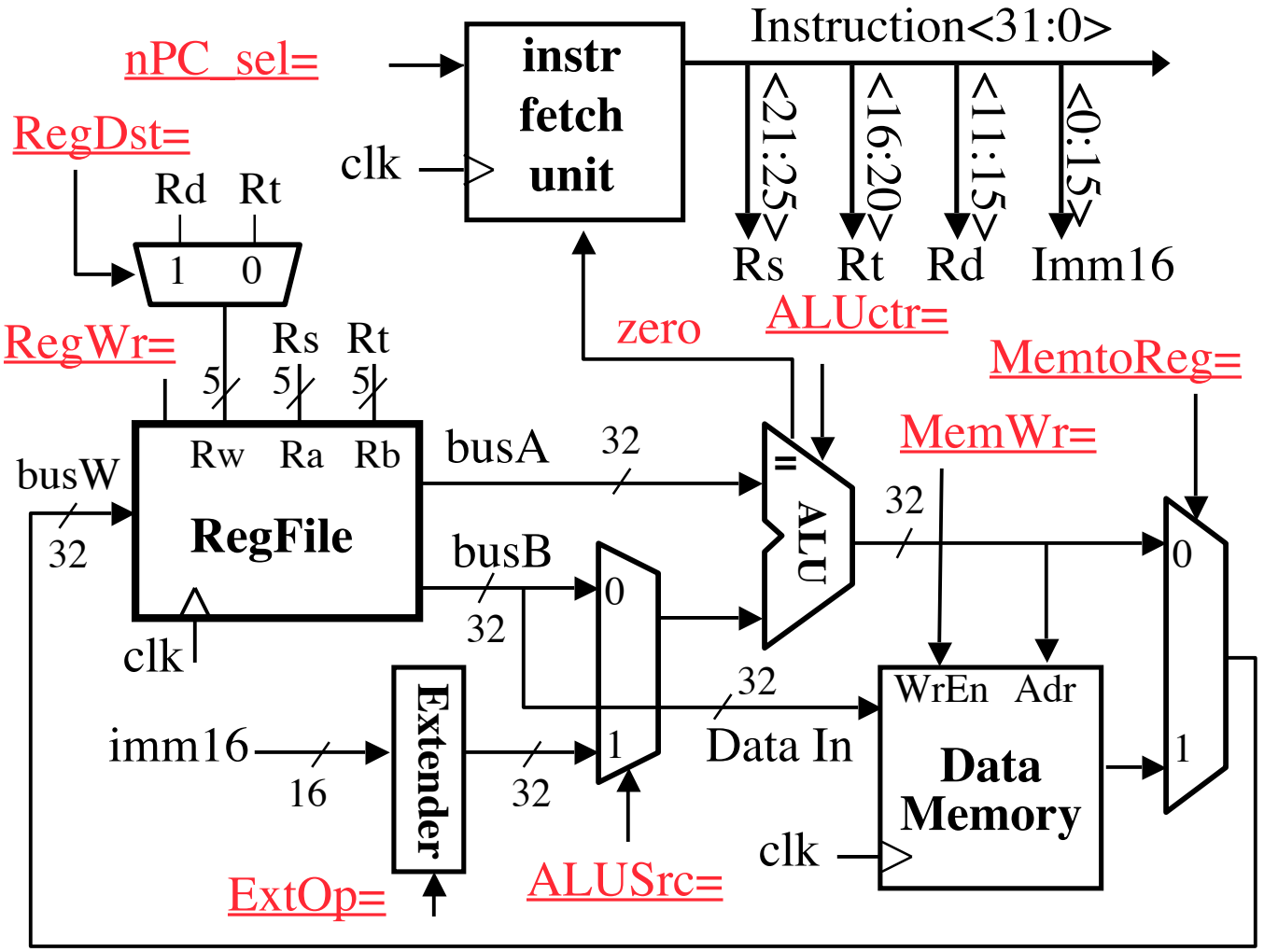
- R[rt] = R[rs] OR ZeroExt[Imm16]



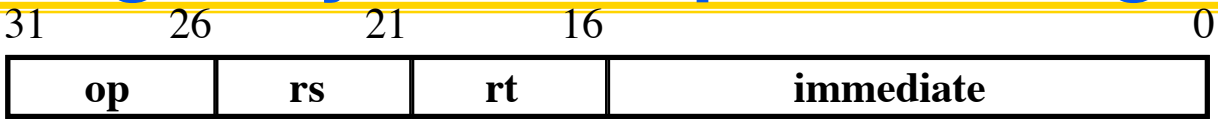
The Single Cycle Datapath during Load?



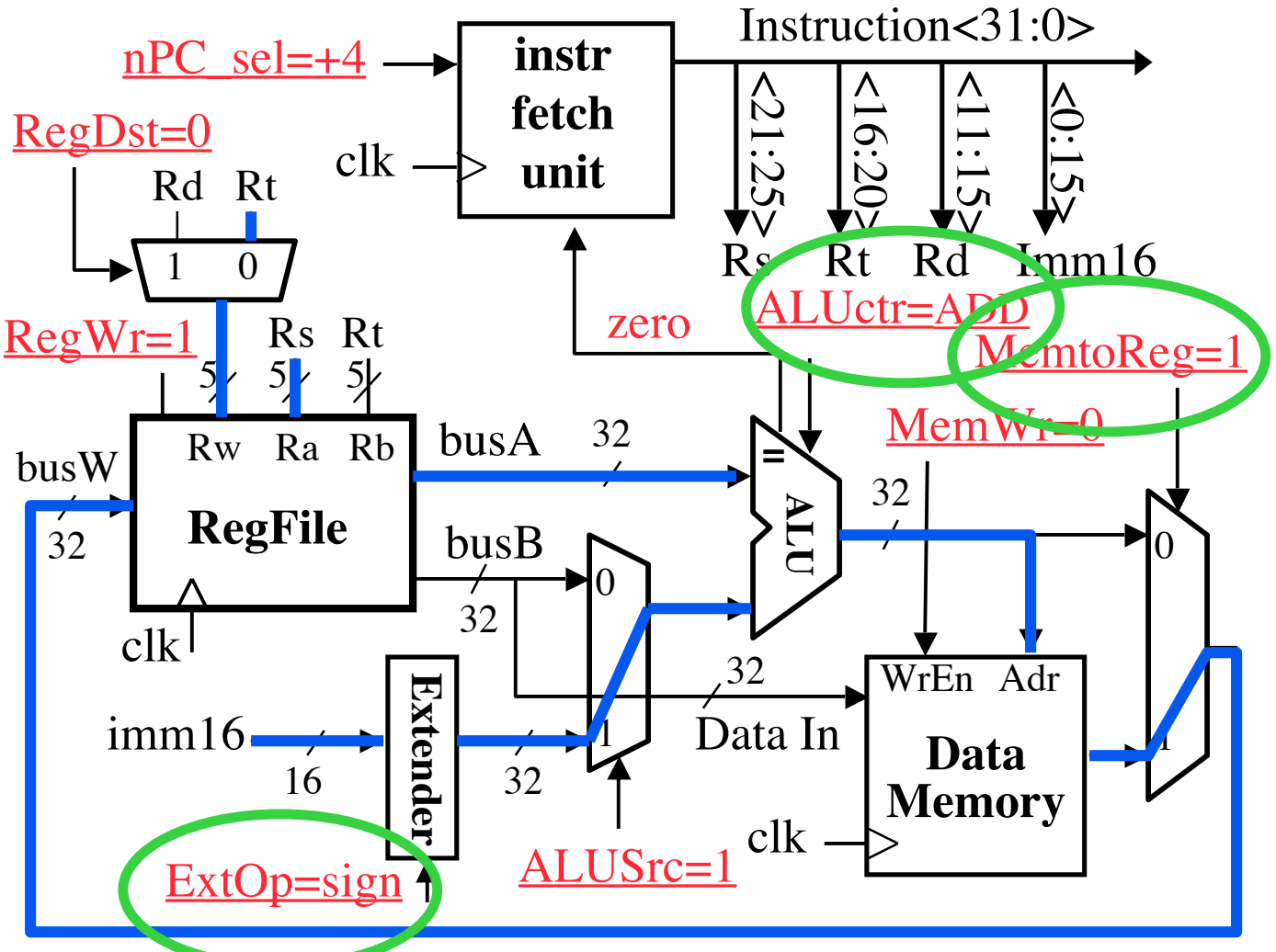
- R[rt] = Data Memory {R[rs] + SignExt[imm16]}



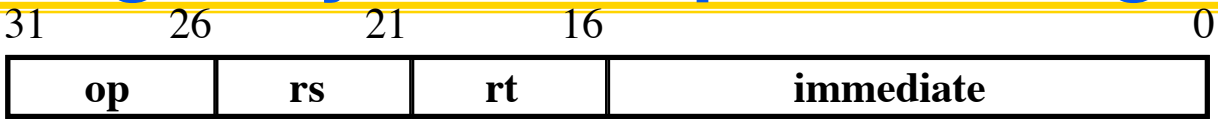
The Single Cycle Datapath during Load



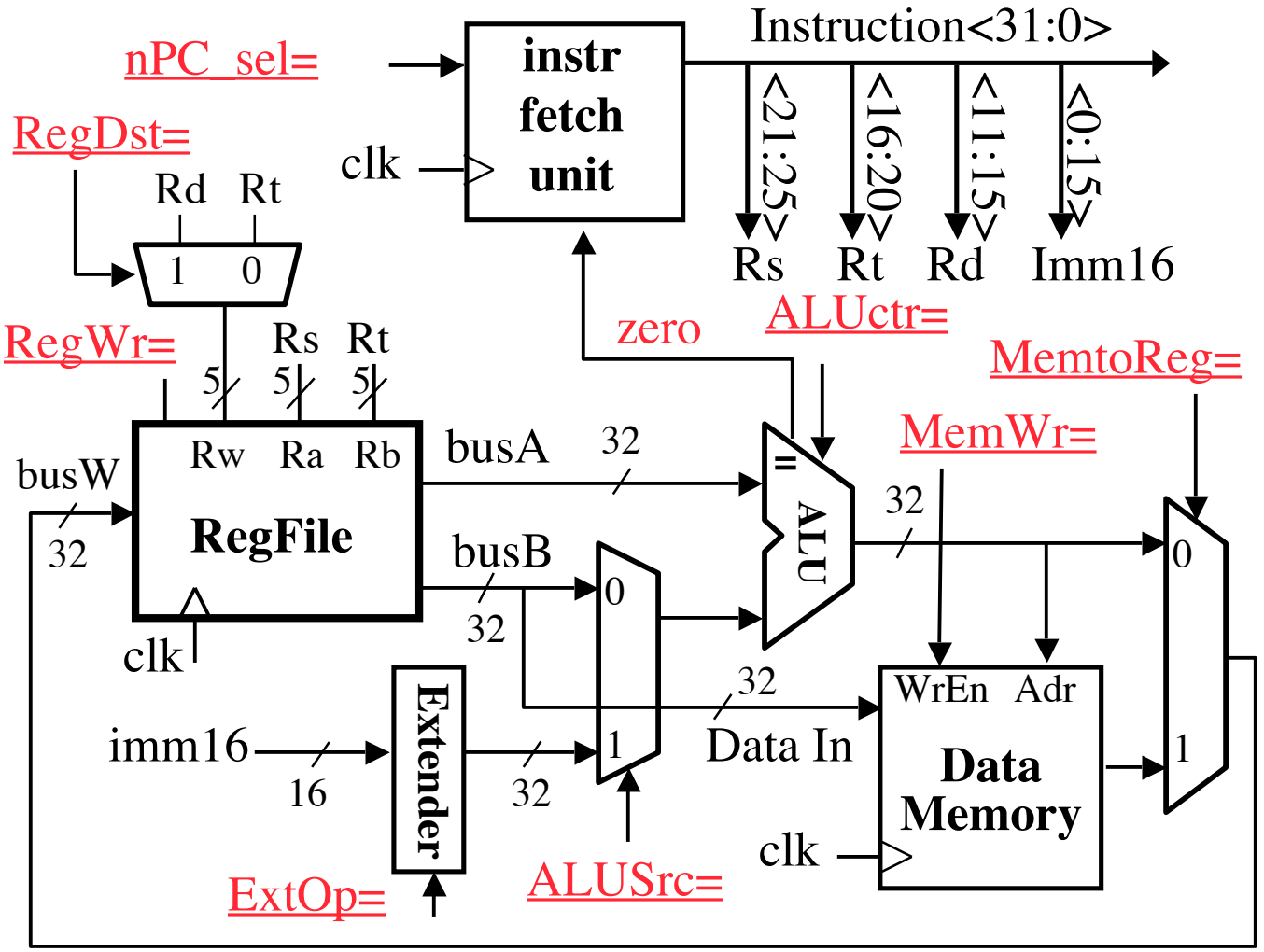
- R[rt] = Data Memory {R[rs] + SignExt[imm16]}



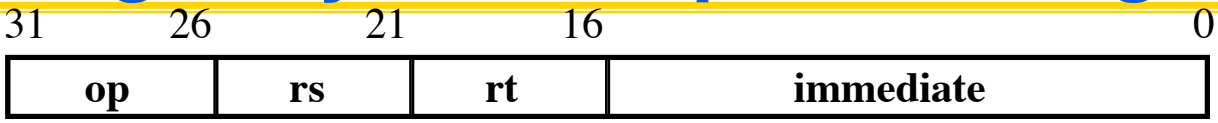
The Single Cycle Datapath during Store?



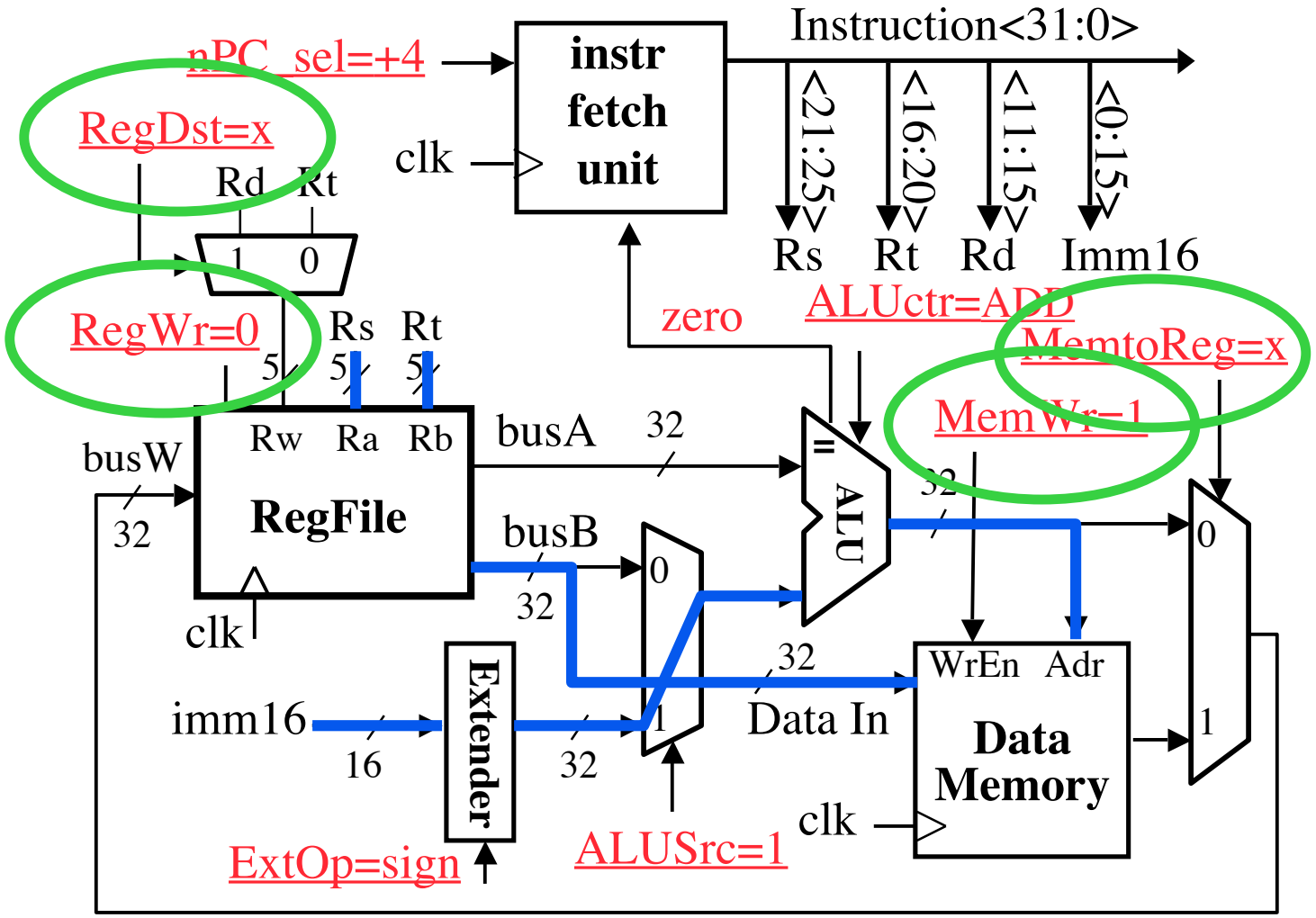
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



The Single Cycle Datapath during Store



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

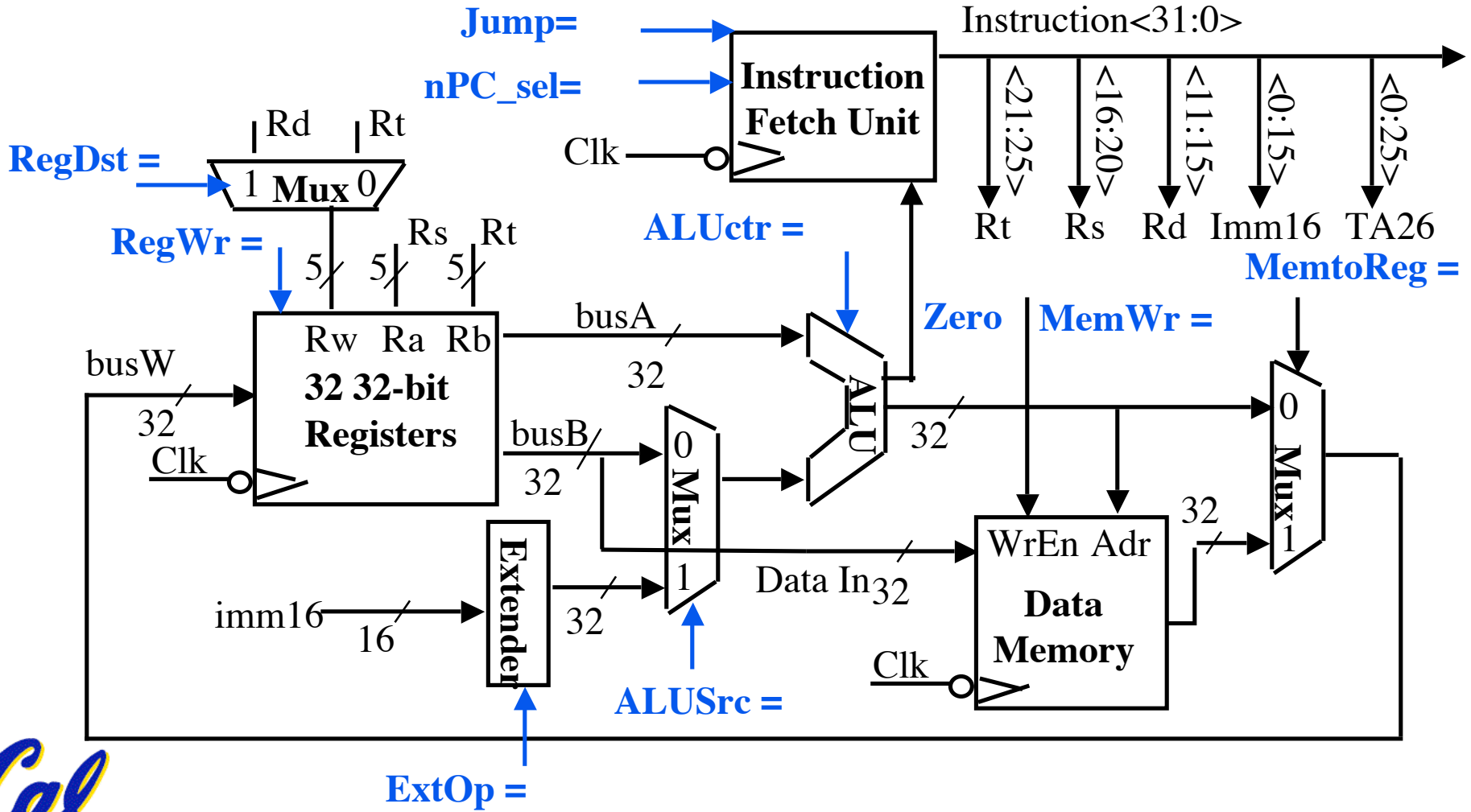


Administrivia

- Assignments
  - HW7 due 8/2
  - Proj3 due 8/5
- Assignment Grading
  - Grades should be coming in now (HW1, HW2 done, expect HW3, HW4, Proj1 soon)
  - Reader info posted on webpage
- Midterm Regrades due Wed 8/1

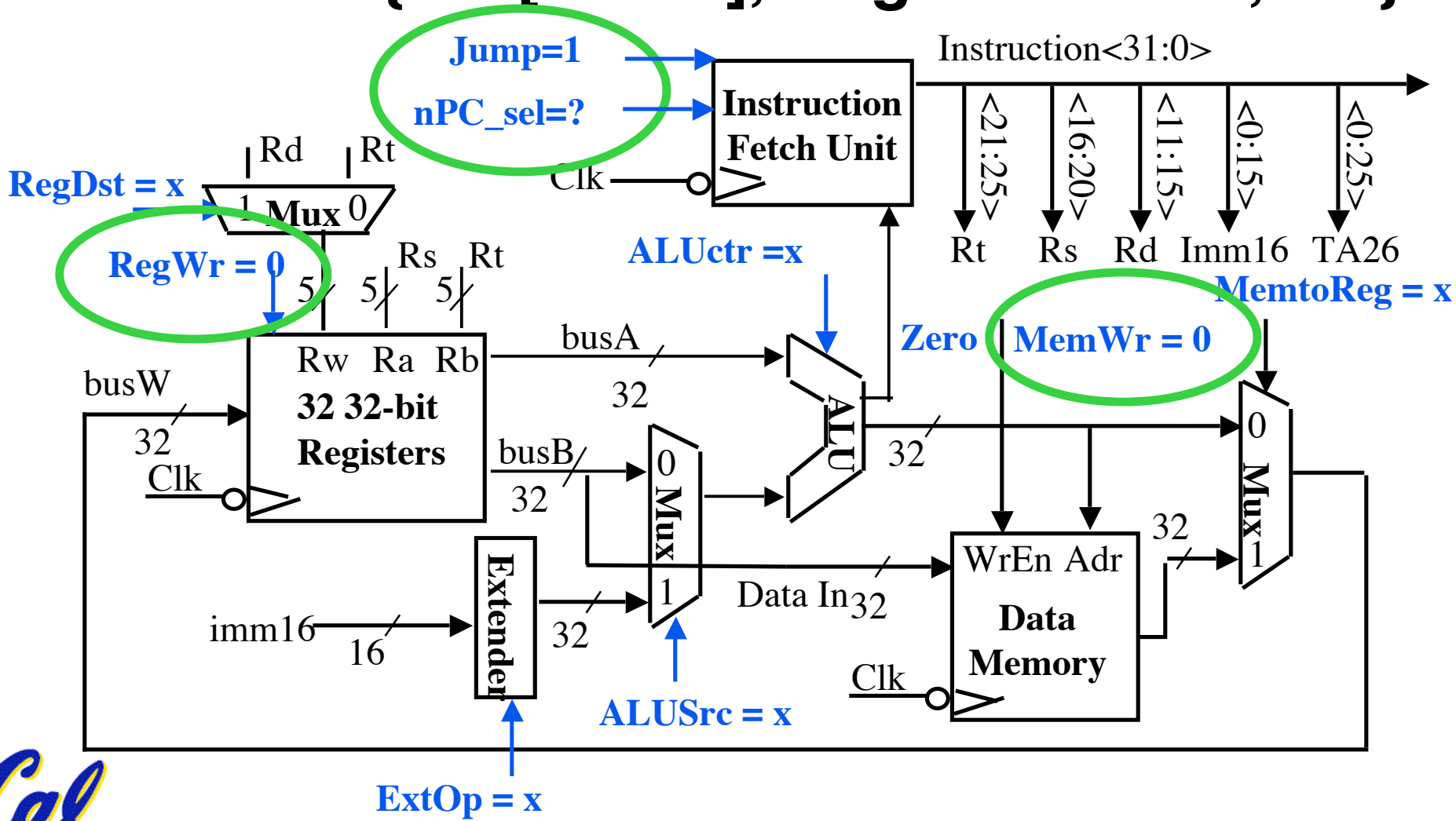
The Single Cycle Datapath during Jump

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



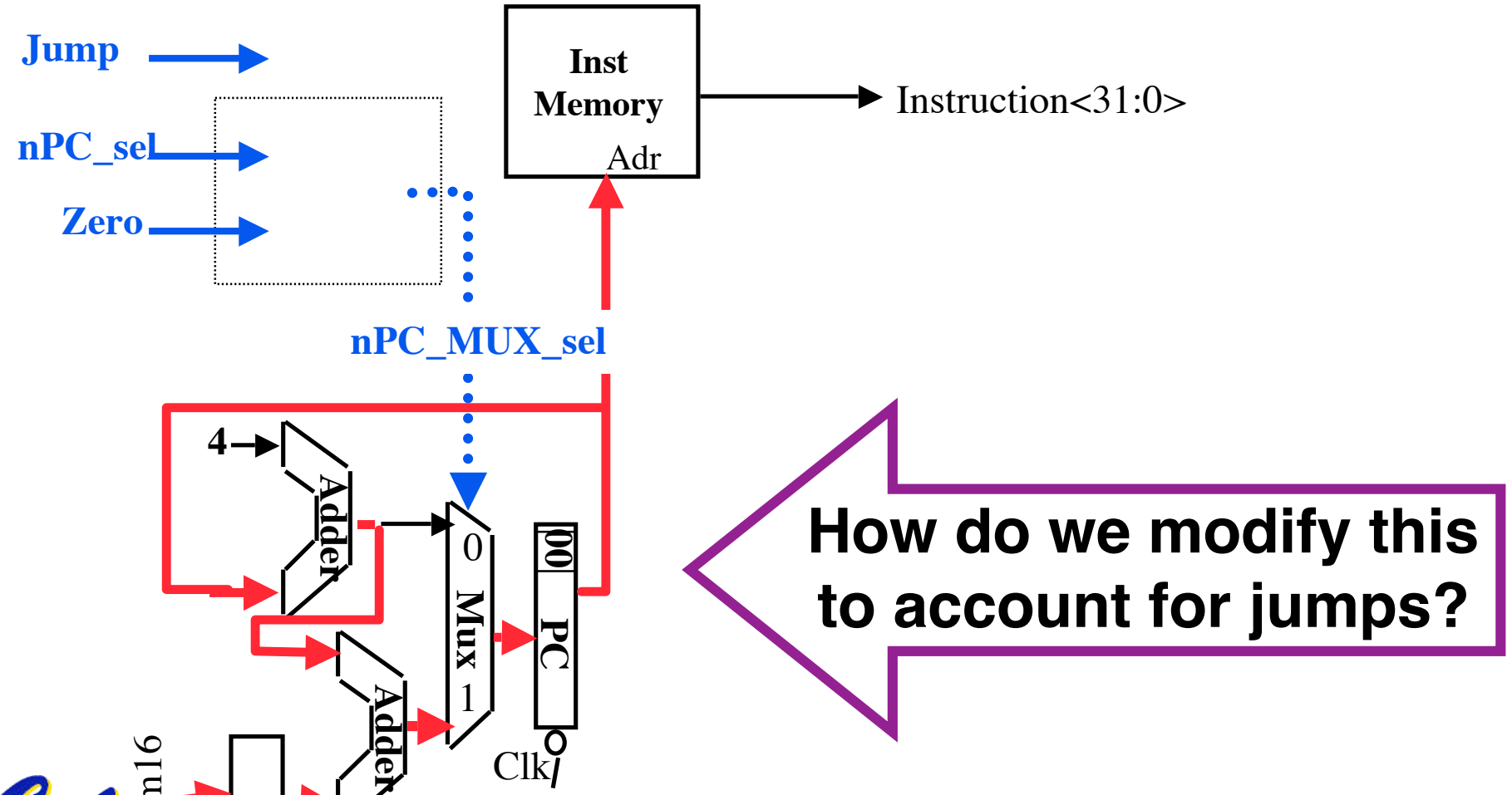
The Single Cycle Datapath during Jump

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



Instruction Fetch Unit at the End of Jump

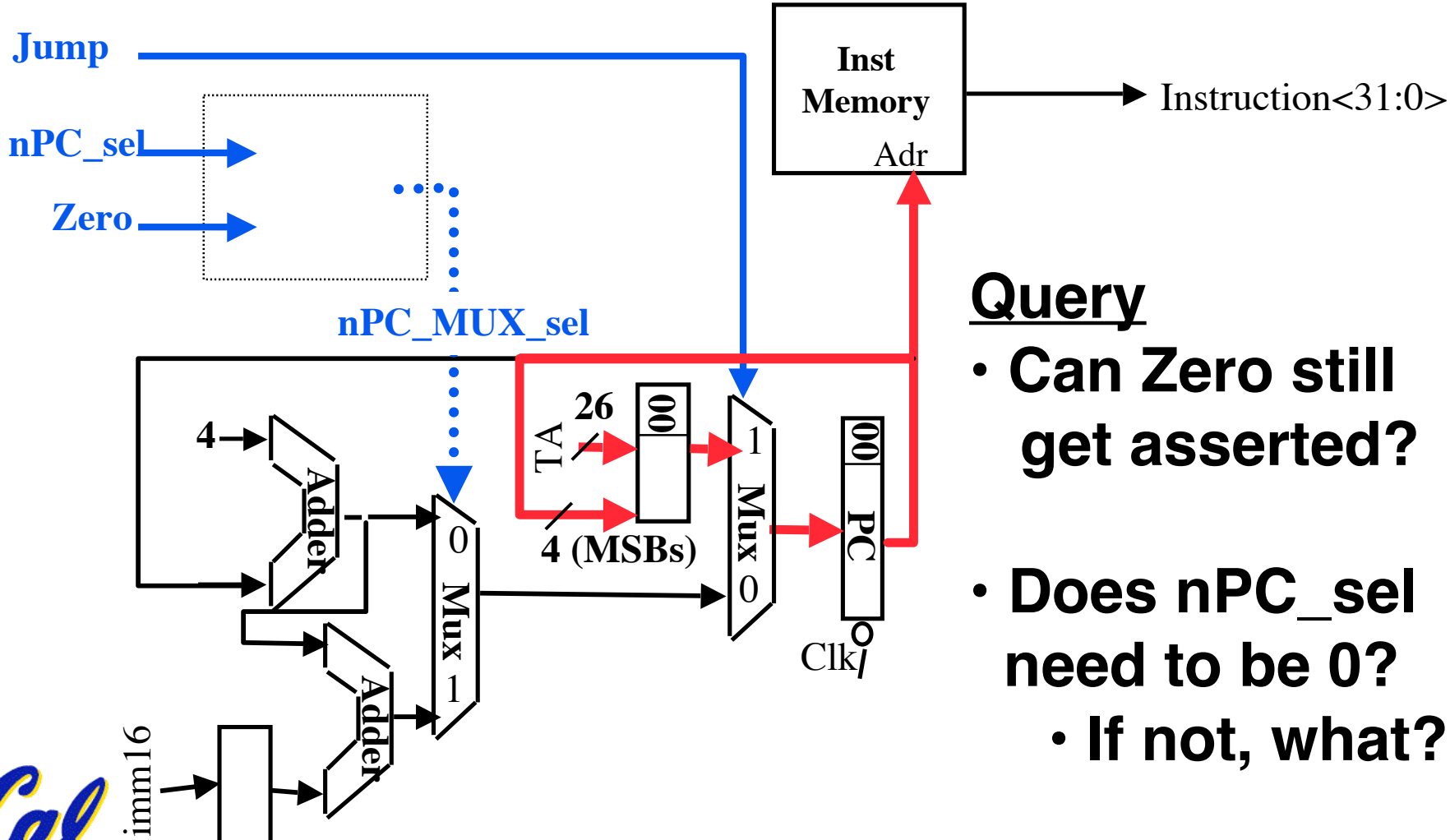
- New PC = { PC[31..28], target address, 00 }



How do we modify this to account for jumps?

Instruction Fetch Unit at the End of Jump

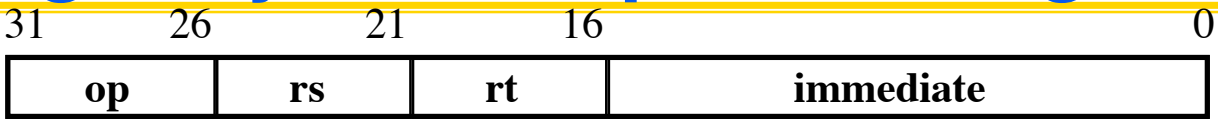
- New PC = { PC[31..28], target address, 00 }



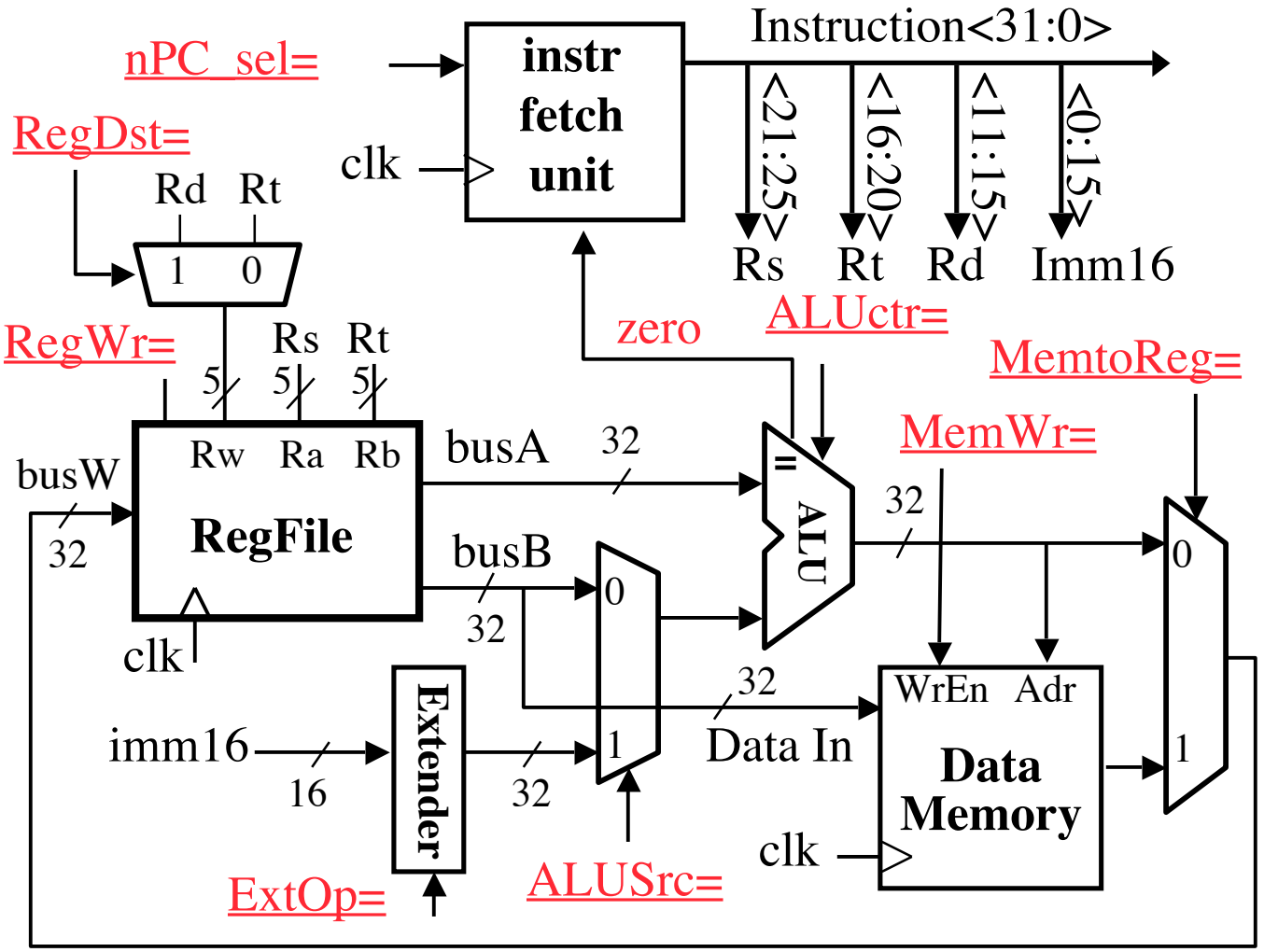
Query

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

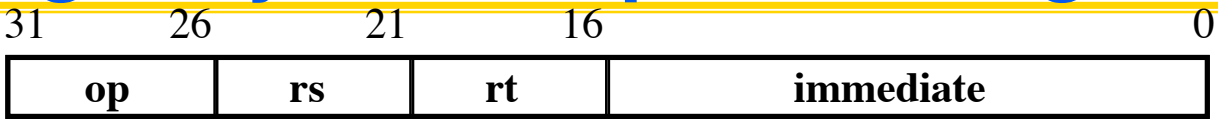
The Single Cycle Datapath during Branch?



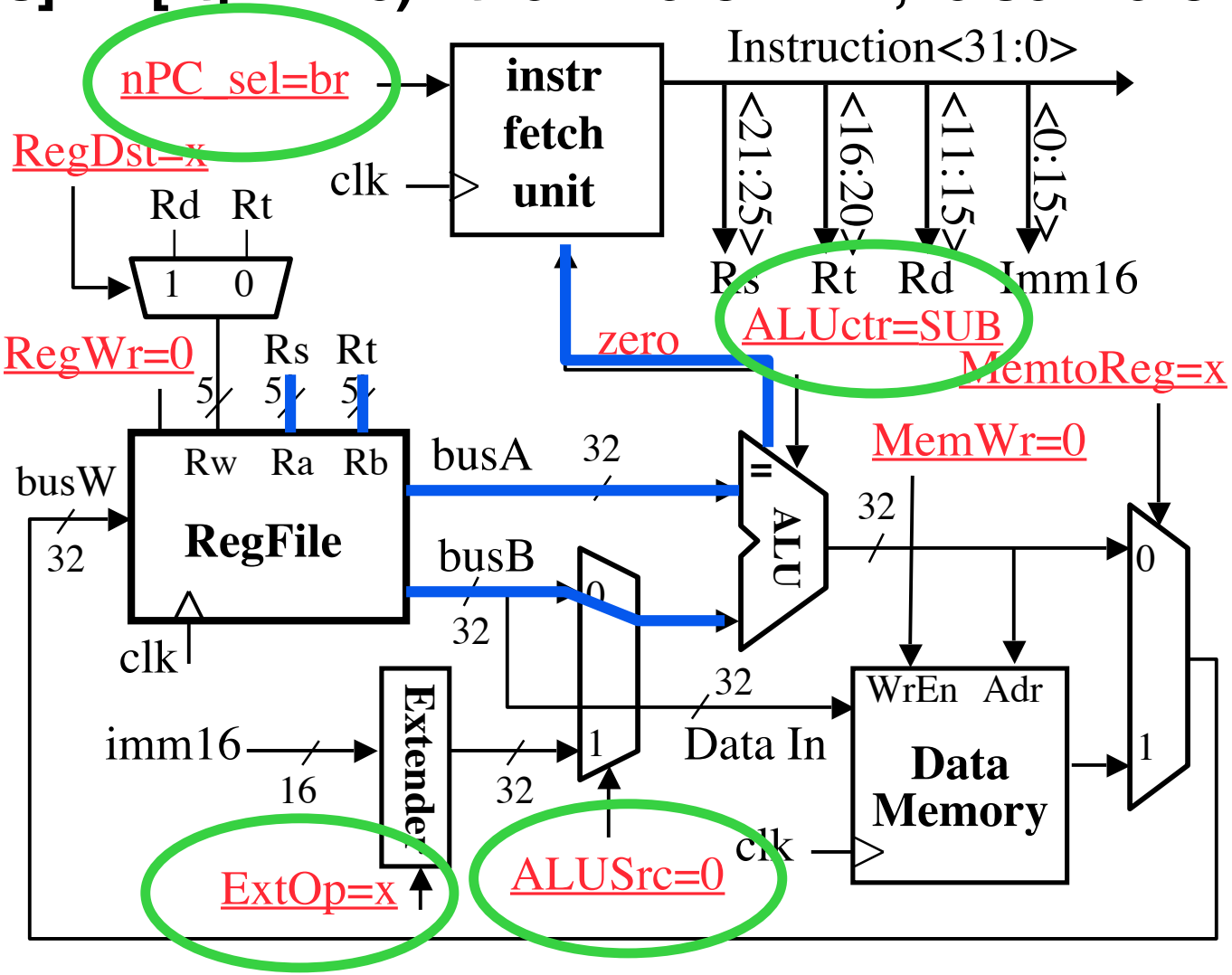
- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



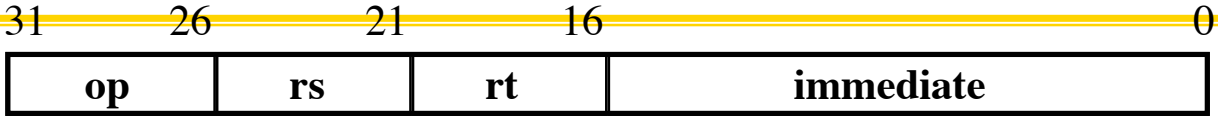
The Single Cycle Datapath during Branch



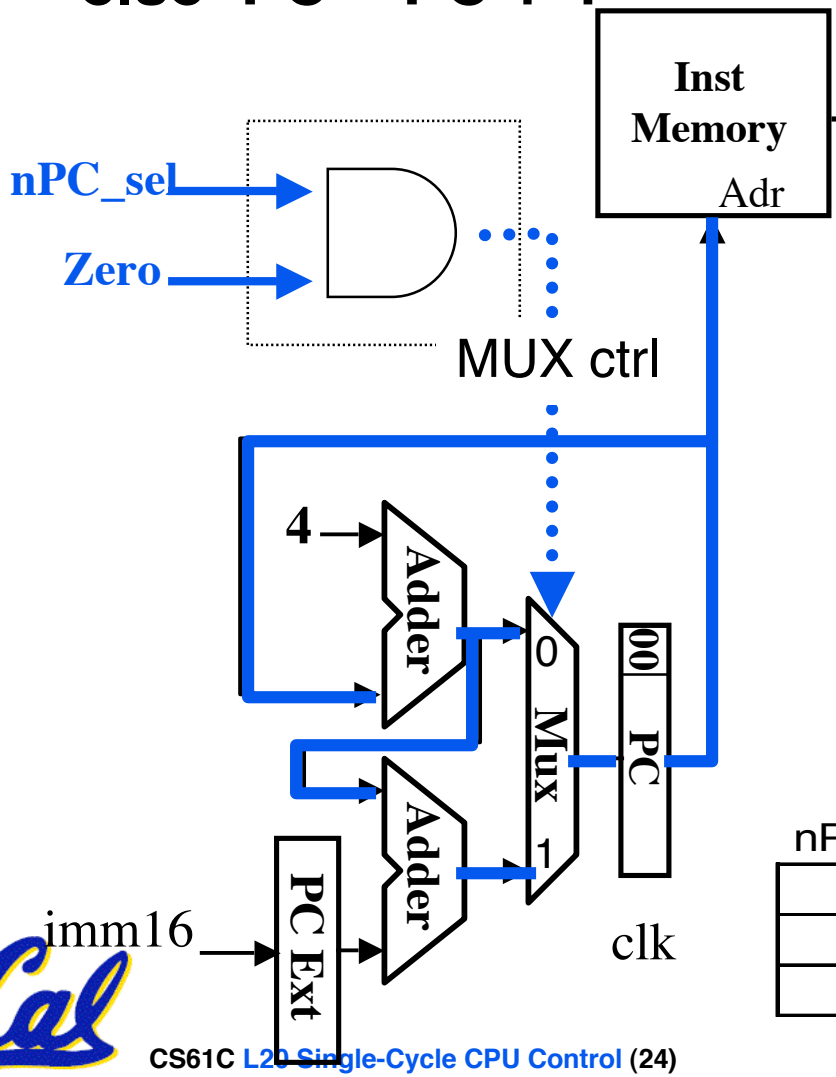
- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



Instruction Fetch Unit at the End of Branch



- if (Zero == 1) then PC = PC + 4 + SignExt[imm16]*4 ; else PC = PC + 4

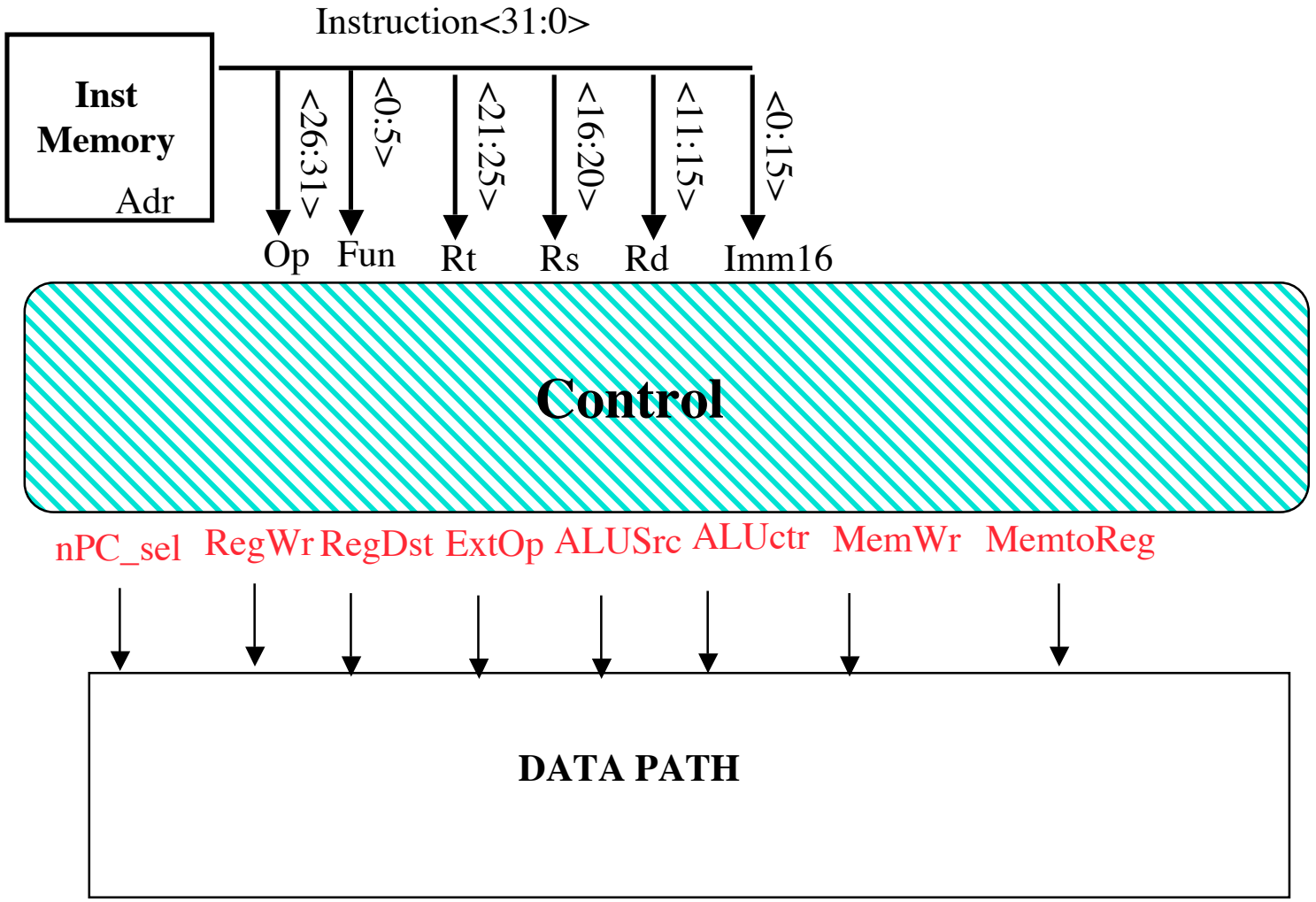


- What is encoding of nPC_sel?
  - Direct MUX select?
  - Branch inst. / not branch
- Let's pick 2nd option

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

Q: What logic gate?

Step 4: Given Datapath: RTL → Control



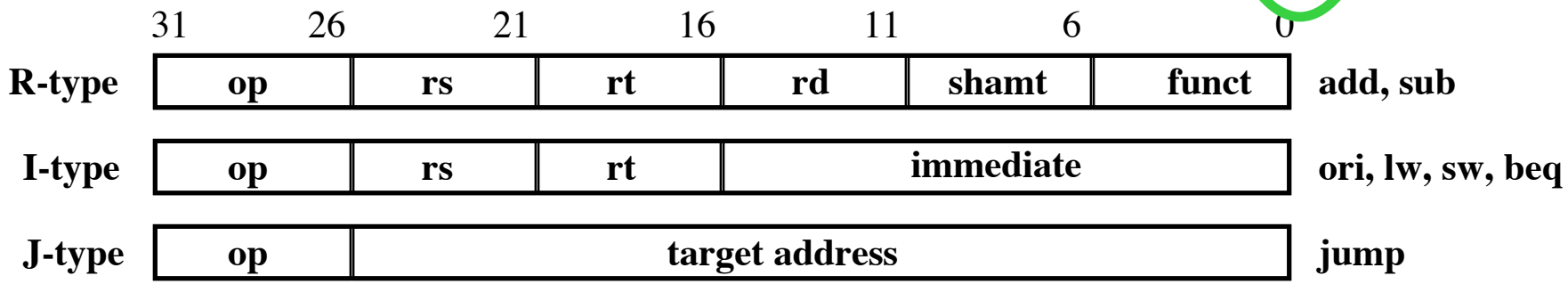
A Summary of the Control Signals (1/2)

inst	Register Transfer	
add	R[rd] ← R[rs] + R[rt];	PC ← PC + 4
	ALUsrc = RegB, ALUctr = "ADD", RegDst = rd, RegWr, nPC_sel = "+4"	
sub	R[rd] ← R[rs] – R[rt];	PC ← PC + 4
	ALUsrc = RegB, ALUctr = "SUB", RegDst = rd, RegWr, nPC_sel = "+4"	
ori	R[rt] ← R[rs] + zero_ext(Imm16);	PC ← PC + 4
	ALUsrc = Im, Extop = "Z",ALUctr = "OR", RegDst = rt,RegWr, nPC_sel ="+4"	
lw	R[rt] ← MEM[R[rs] + sign_ext(Imm16)];	PC ← PC + 4
	ALUsrc = Im, Extop = "sn", ALUctr = "ADD", MemtoReg, RegDst = rt, RegWr,	nPC_sel = "+4"
sw	MEM[R[rs] + sign_ext(Imm16)] ← R[rs];	PC ← PC + 4
	ALUsrc = Im, Extop = "sn", ALUctr = "ADD", MemWr, nPC_sel = "+4"	
beq	if (R[rs] == R[rt]) then PC ← PC + sign_ext(Imm16)] || 00 else PC ← PC + 4	
	nPC_sel = "br", ALUctr = "SUB"	

A Summary of the Control Signals (2/2)

See Appendix A

	func	10 0000	10 0010	We Don't Care :-)				
	op	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

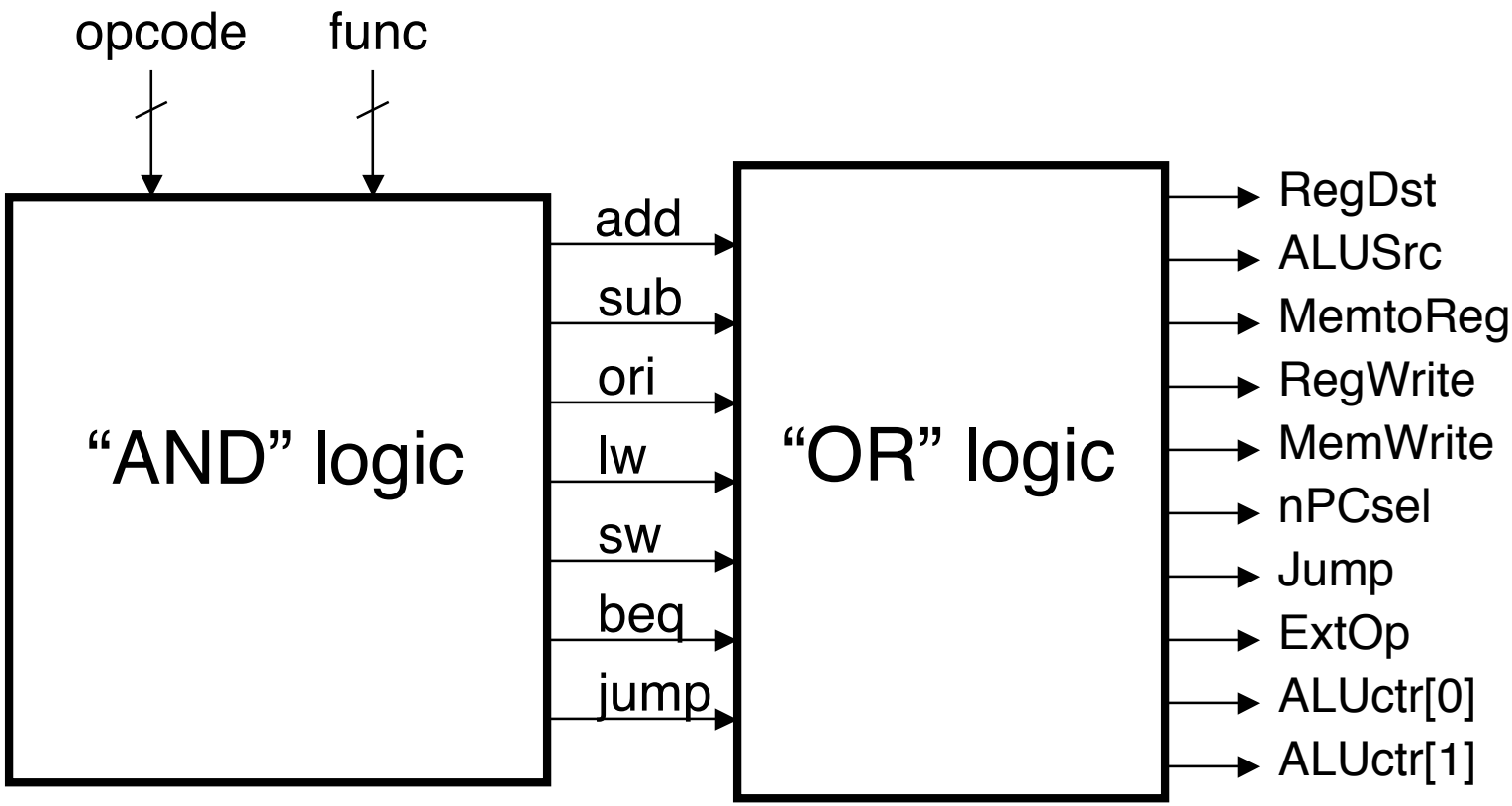
where,

$$\text{rtype} = \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet \sim op_1 \bullet \sim op_0,$$
$$\text{ori} = \sim op_5 \bullet \sim op_4 \bullet op_3 \bullet op_2 \bullet \sim op_1 \bullet op_0$$
$$\text{lw} = op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet op_1 \bullet op_0$$
$$\text{sw} = op_5 \bullet \sim op_4 \bullet op_3 \bullet \sim op_2 \bullet op_1 \bullet op_0$$
$$\text{beq} = \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet op_2 \bullet \sim op_1 \bullet \sim op_0$$
$$\text{jump} = \sim op_5 \bullet \sim op_4 \bullet \sim op_3 \bullet \sim op_2 \bullet op_1 \bullet \sim op_0$$

$$\text{add} = \text{rtype} \bullet func_5 \bullet \sim func_4 \bullet \sim func_3 \bullet \sim func_2 \bullet \sim func_1 \bullet \sim func_0$$
$$\text{sub} = \text{rtype} \bullet func_5 \bullet \sim func_4 \bullet \sim func_3 \bullet \sim func_2 \bullet func_1 \bullet \sim func_0$$

How do we implement this in gates?

Controller Implementation



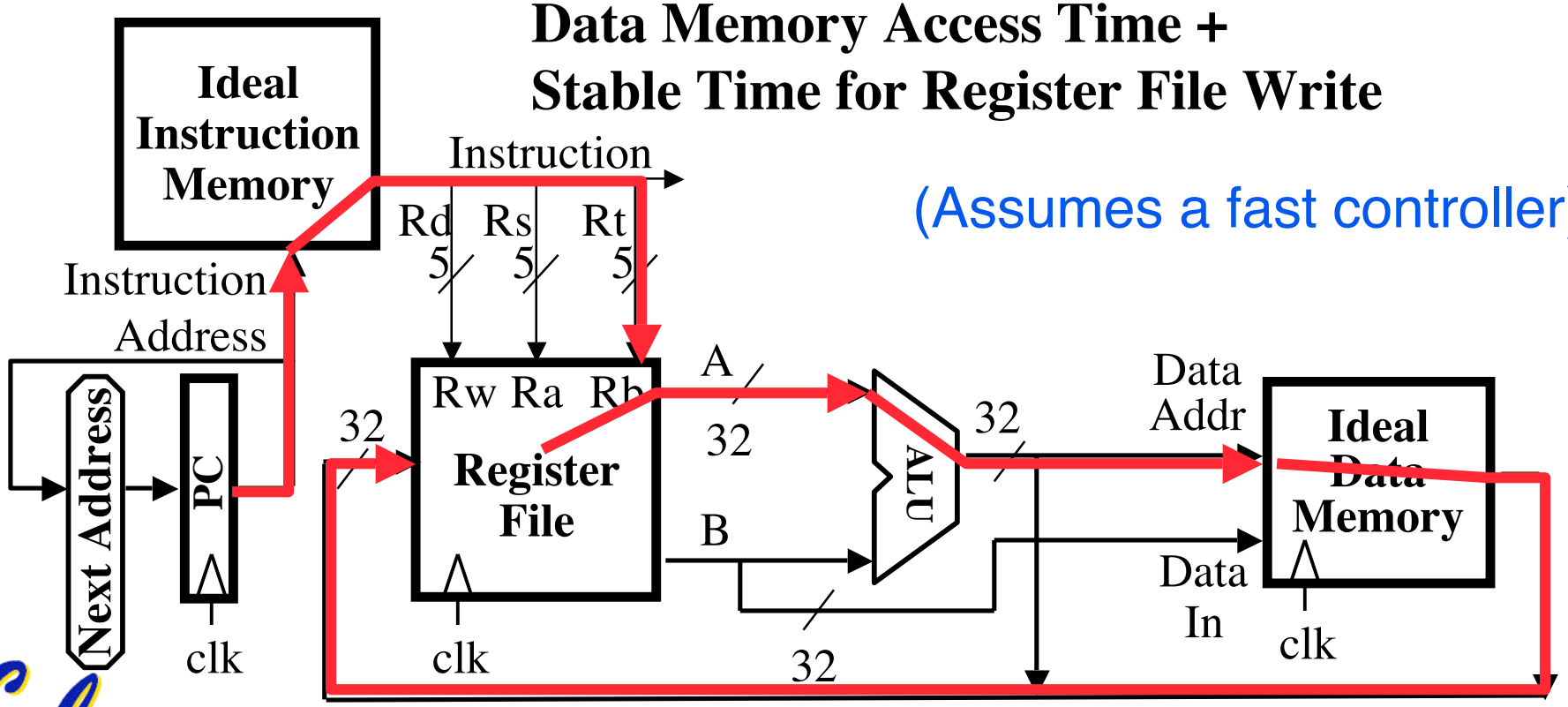
Other Programmable Logic Arrays

- There are other types of PLAs which can be reprogrammed on the fly

- The most common is called a Field Programmable Gate Array (FPGA)
  - made up of configurable logic blocks (CLBs) and flip-flops which can be programmed by software
  - Berkeley has on-going research into reconfigurable computing with FPGAs
    - Check out RAMP and BEE3 projects

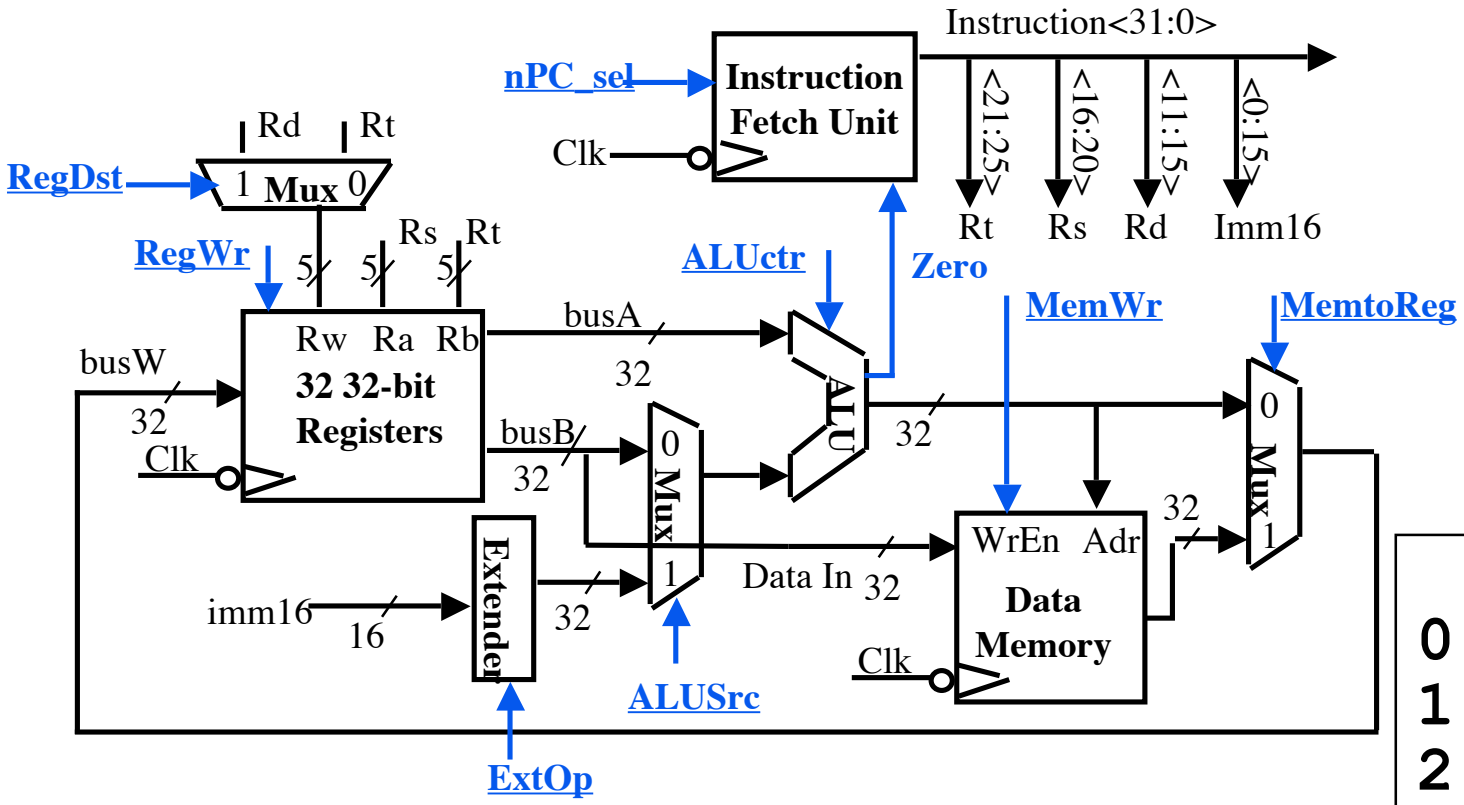
An Abstract View of the Critical Path

Critical Path (Load Instruction) =
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



(Assumes a fast controller)

Peer Instruction



	ABC
0:	SRF
1:	SRT
2:	SEF
3:	SET
4:	BRF
5:	BRT
6:	BEF
7:	BET

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?

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