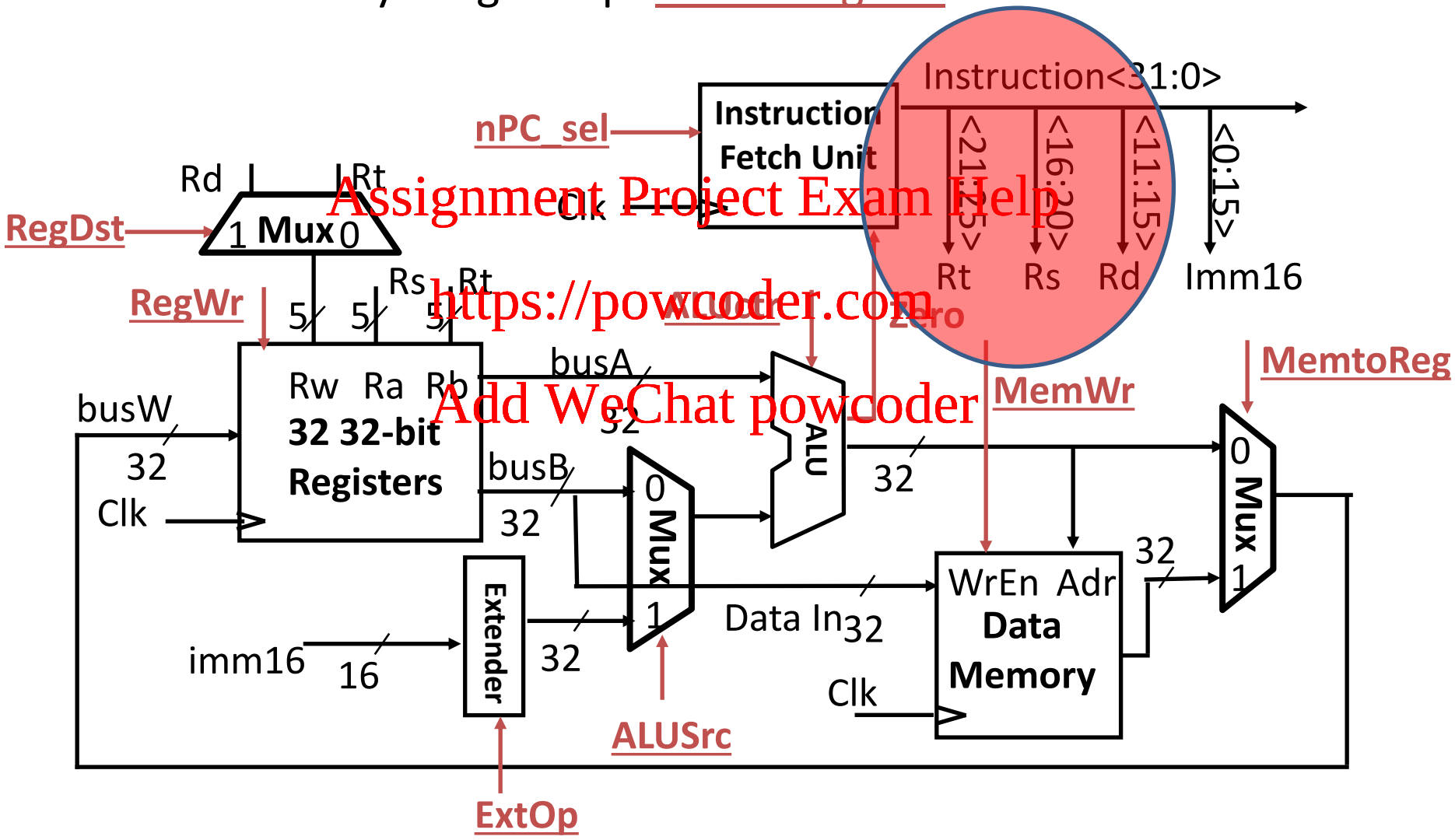
Single Cycle CPU Control

COMP 273

Summary: A Single Cycle Datapath

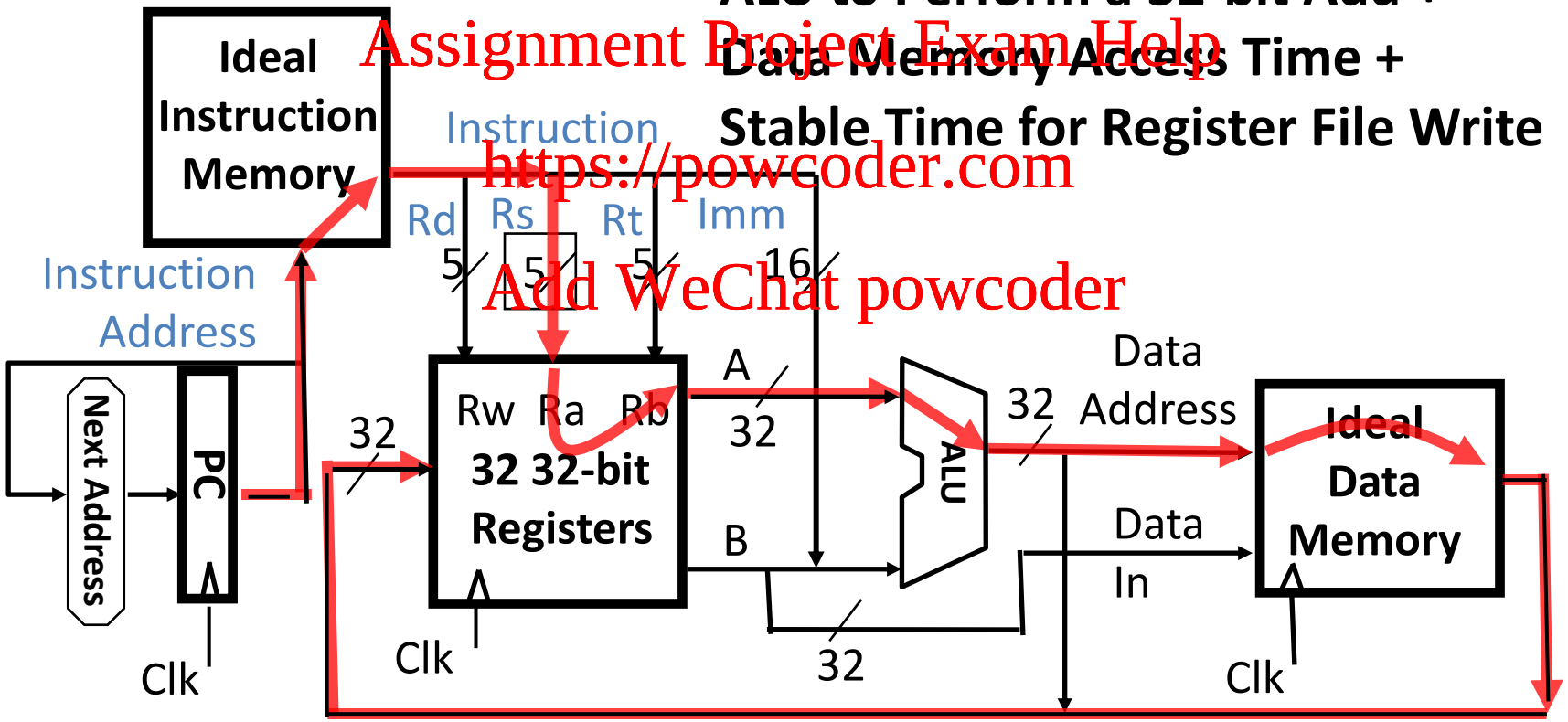
- Rs, Rt, Rd, Imed16 connected to datapath
- We have everything except control signals



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



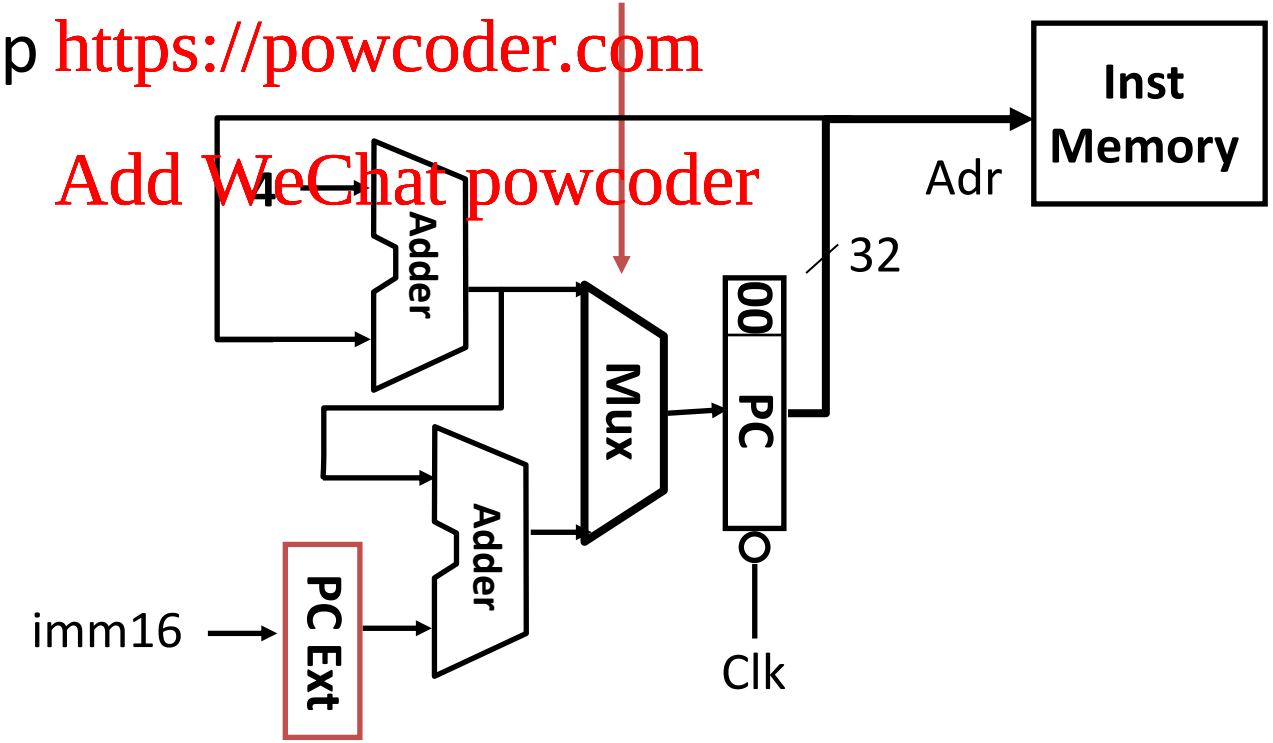
Recap: Meaning of the Control Signals

- nPC_MUX_sel:
  "n"=next

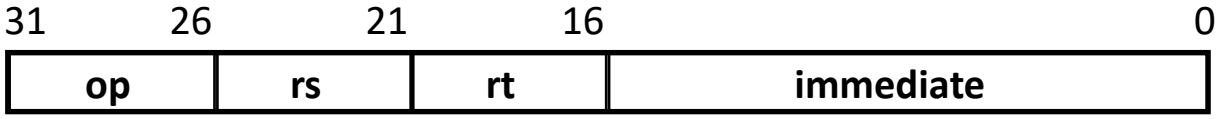
  0 ⇒ PC <– PC + 4

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

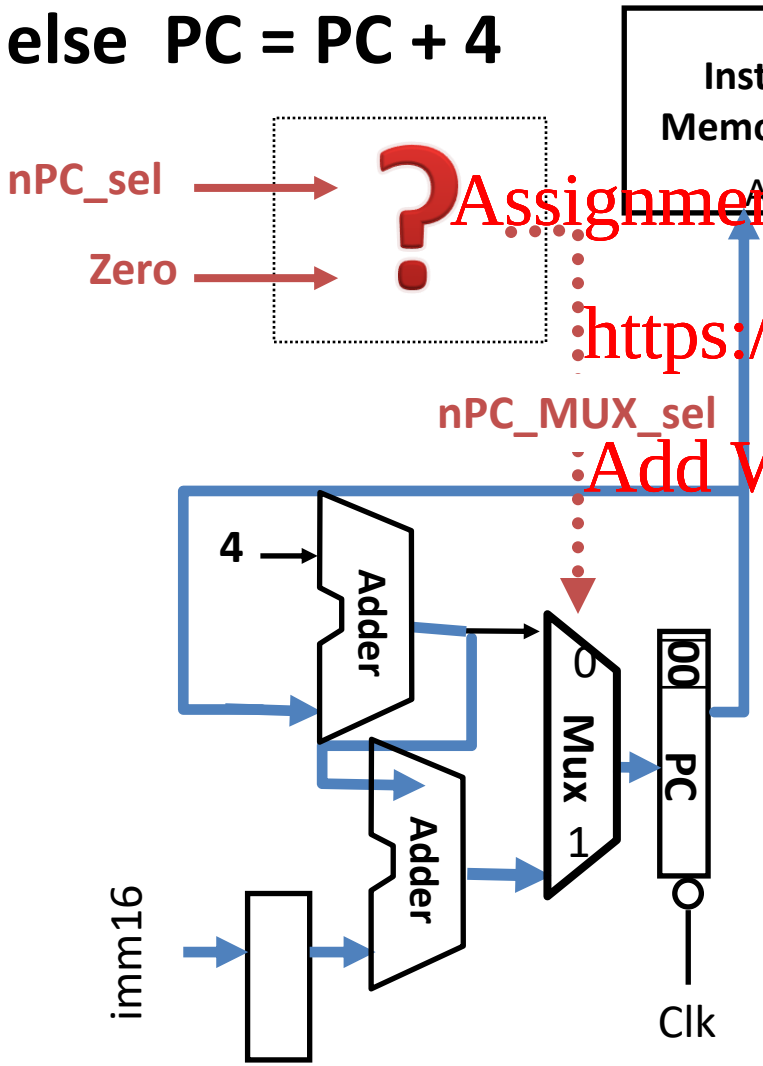
- This is the version without jump instructions



Instruction Fetch Unit at the End of **BRANCH**



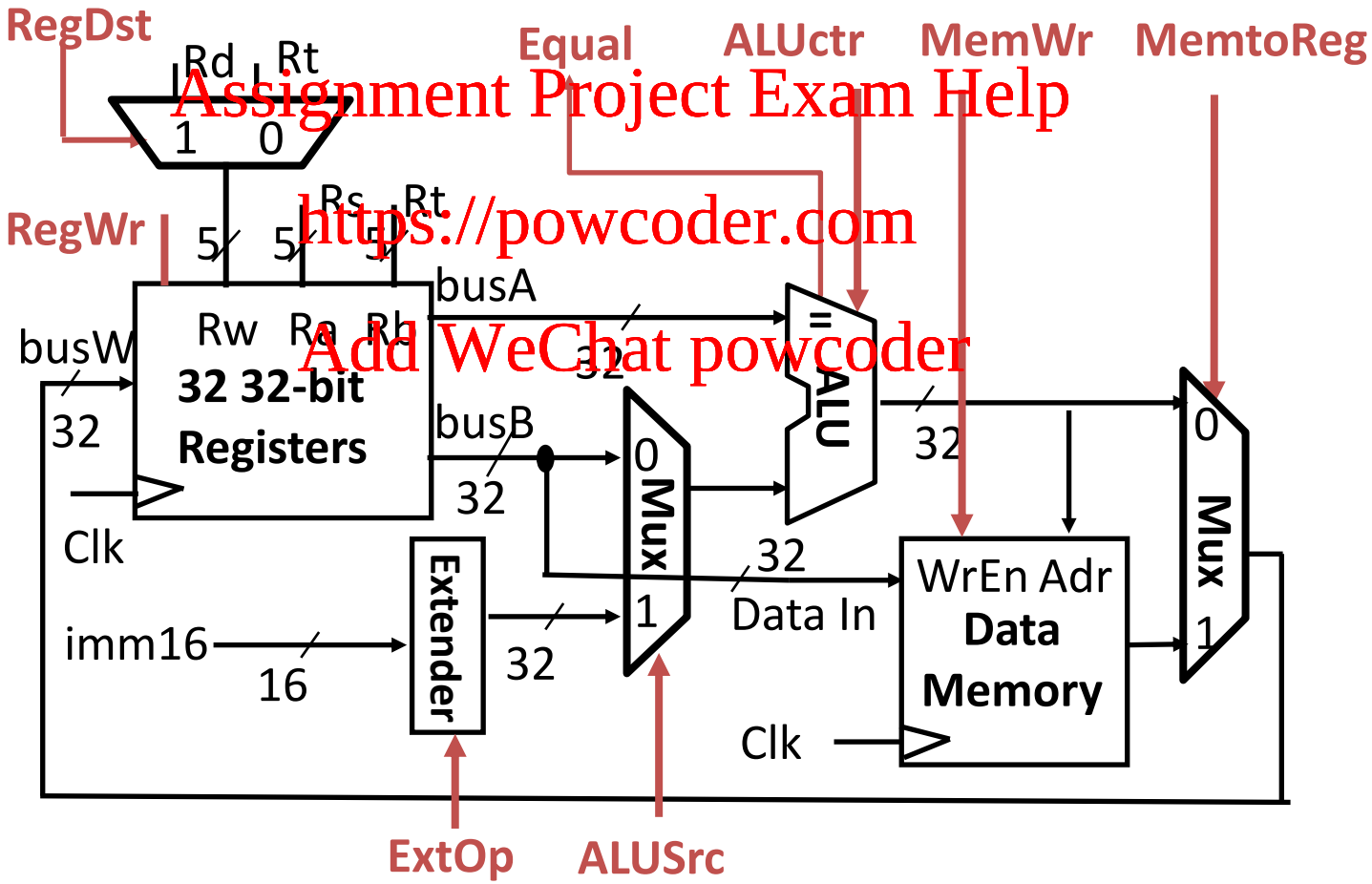
- **if (Zero == 1) then PC = PC + 4 + SignExt[imm16]*4 ;**
  else PC = PC + 4



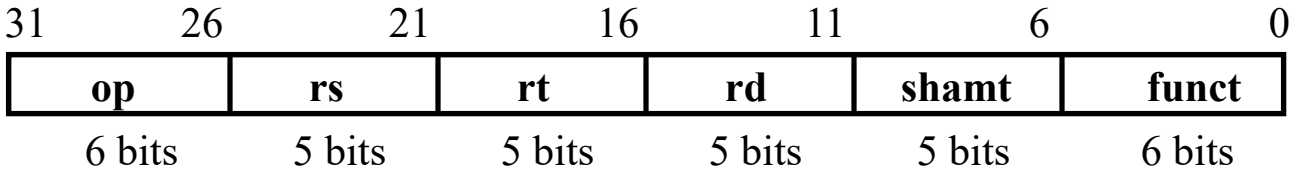
- **What is meaning of nPC_sel?**
  - **Direct MUX select?**
  - **Branch / not branch**
- **We've picked 2nd option, and implemented control logic for BEQ using an AND gate**
- **How does this need to change for BNE?**
- **Note: diagram at left shown without jump part of instruction fetch logic circuit**

Recap: Meaning of the Control Signals

ExtOp:	**"zero", "sign"**	**MemWr:**	**1 ⇒ write memory**
ALUsrc:	**0 ⇒ regB;**	**MemtoReg:**	**0 ⇒ ALU; 1 ⇒ Mem**
	1 ⇒ immed	**RegDst:**	**0 ⇒ "rt"; 1 ⇒ "rd"**
ALUctr:	**"add", "sub", "or"**	**RegWr:**	**1 ⇒ write register**



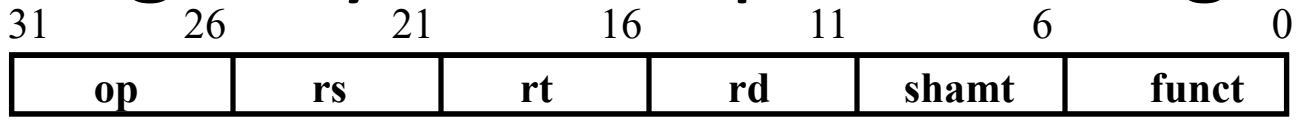
RTL: The **ADD** Instruction



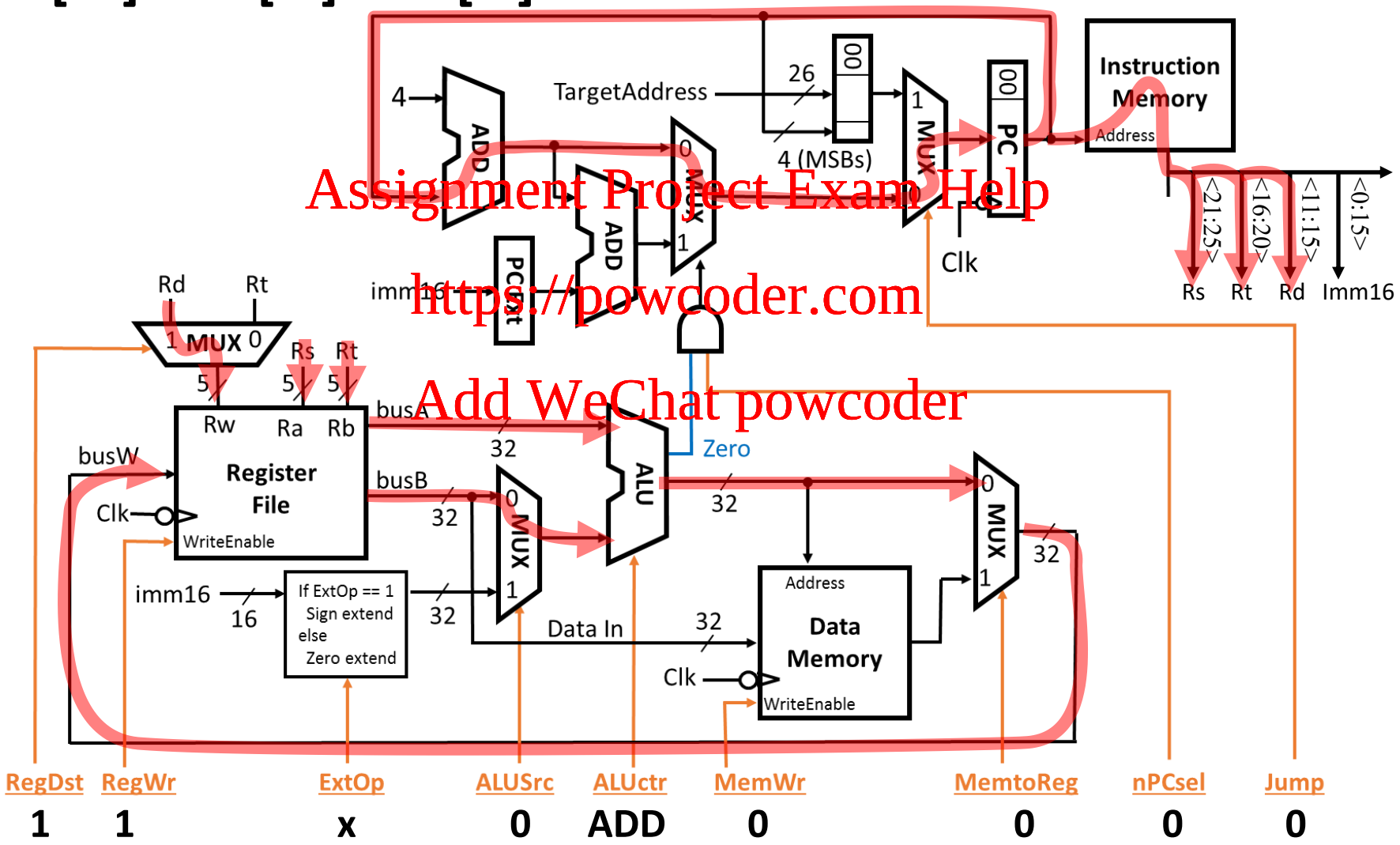

```
add rd, rs, rt
```

- 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

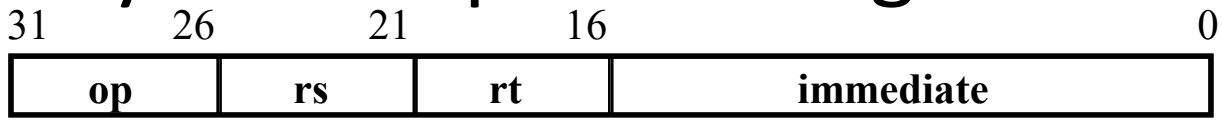
The Single Cycle Datapath during **ADD**



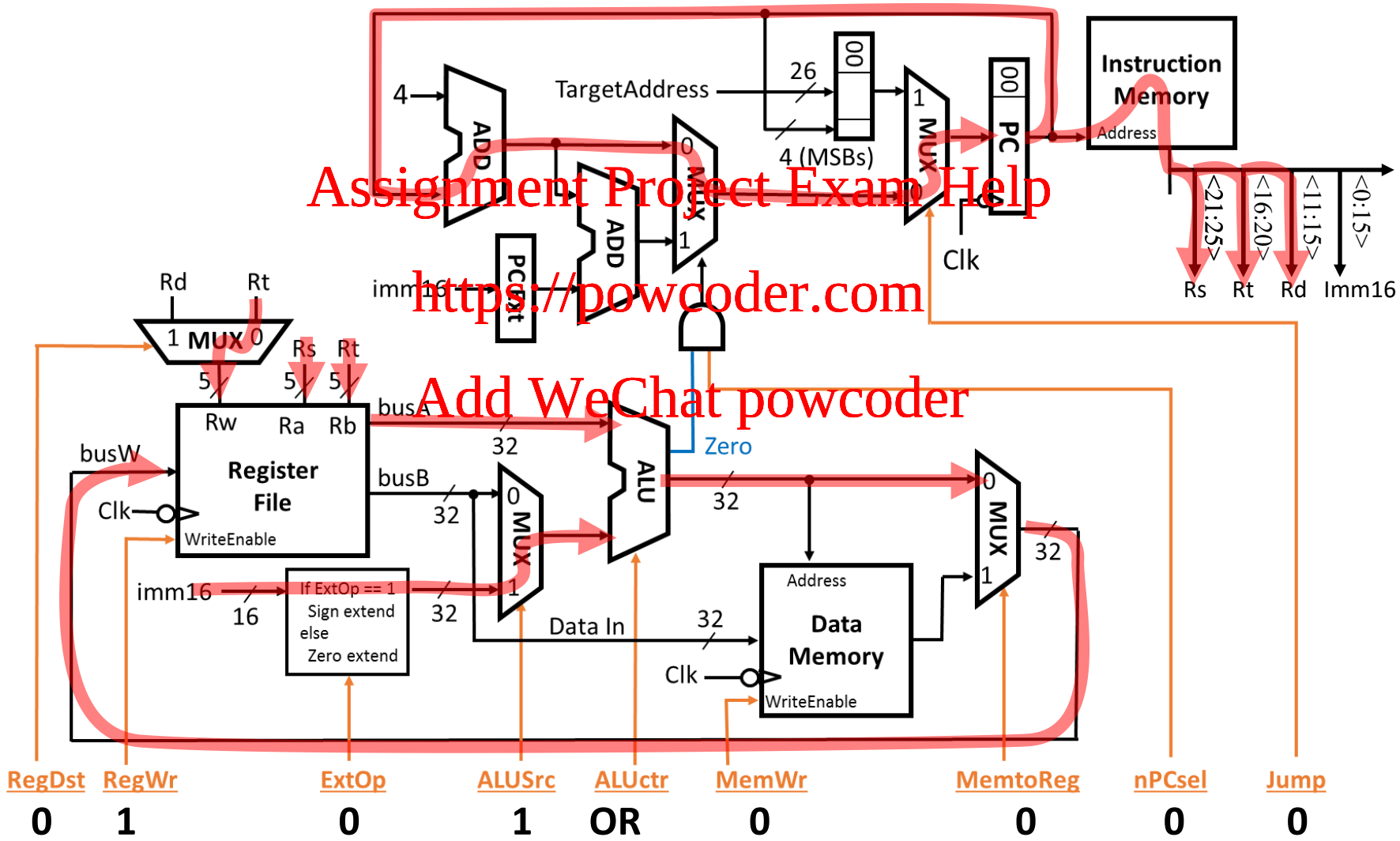
R[rd] = R[rs] + R[rt]



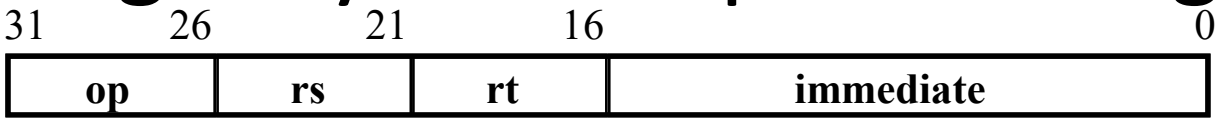
Single Cycle Datapath during **OR** Immediate?



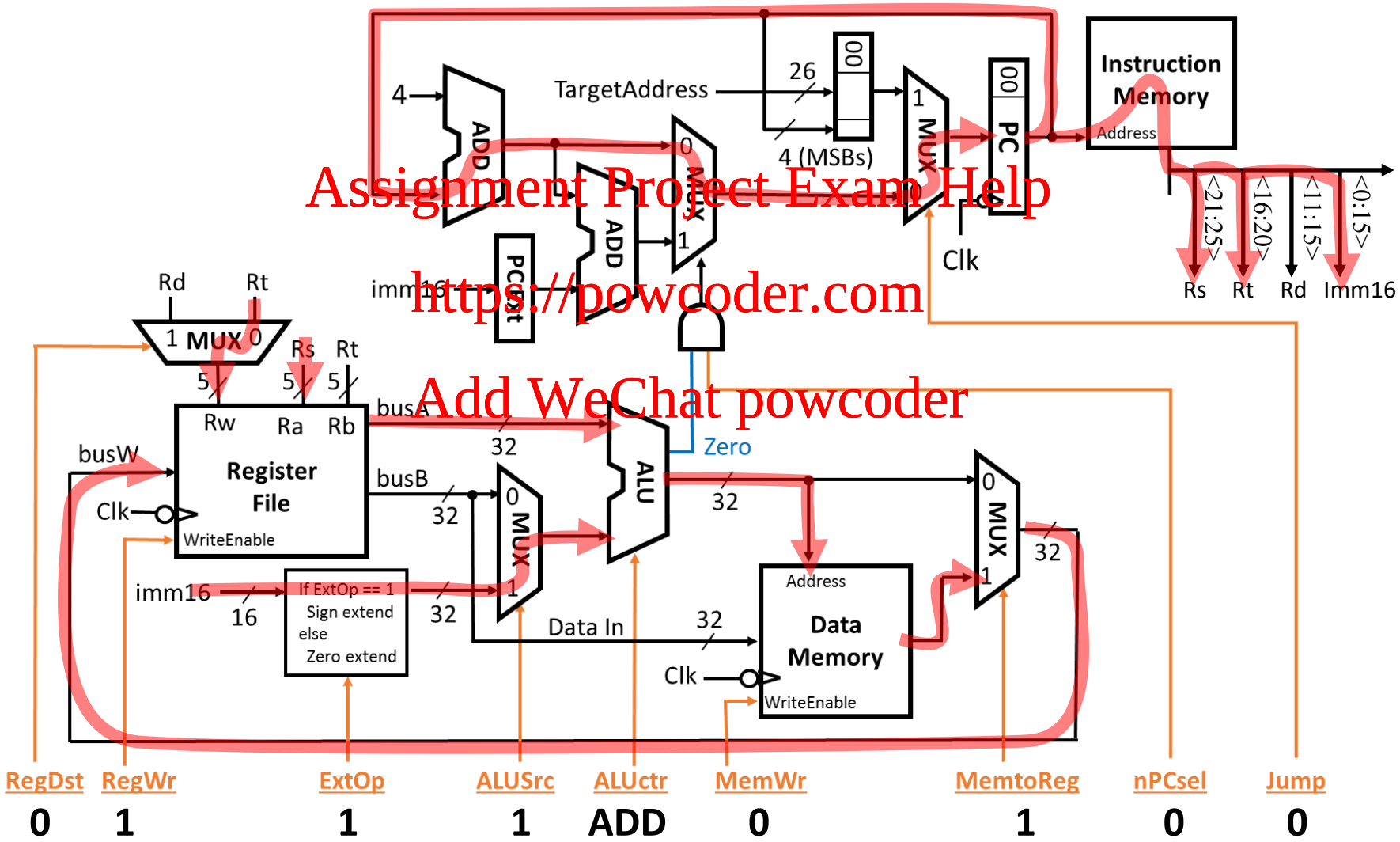
R[rt] = R[rs] OR ZeroExt[Imm16]



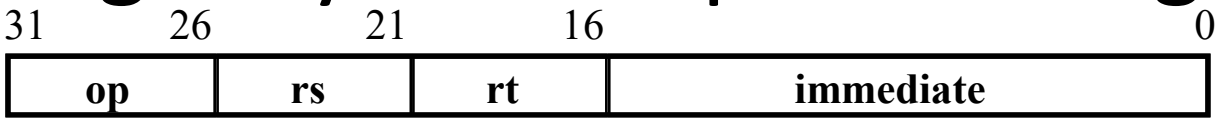
The Single Cycle Datapath during **LOAD**?



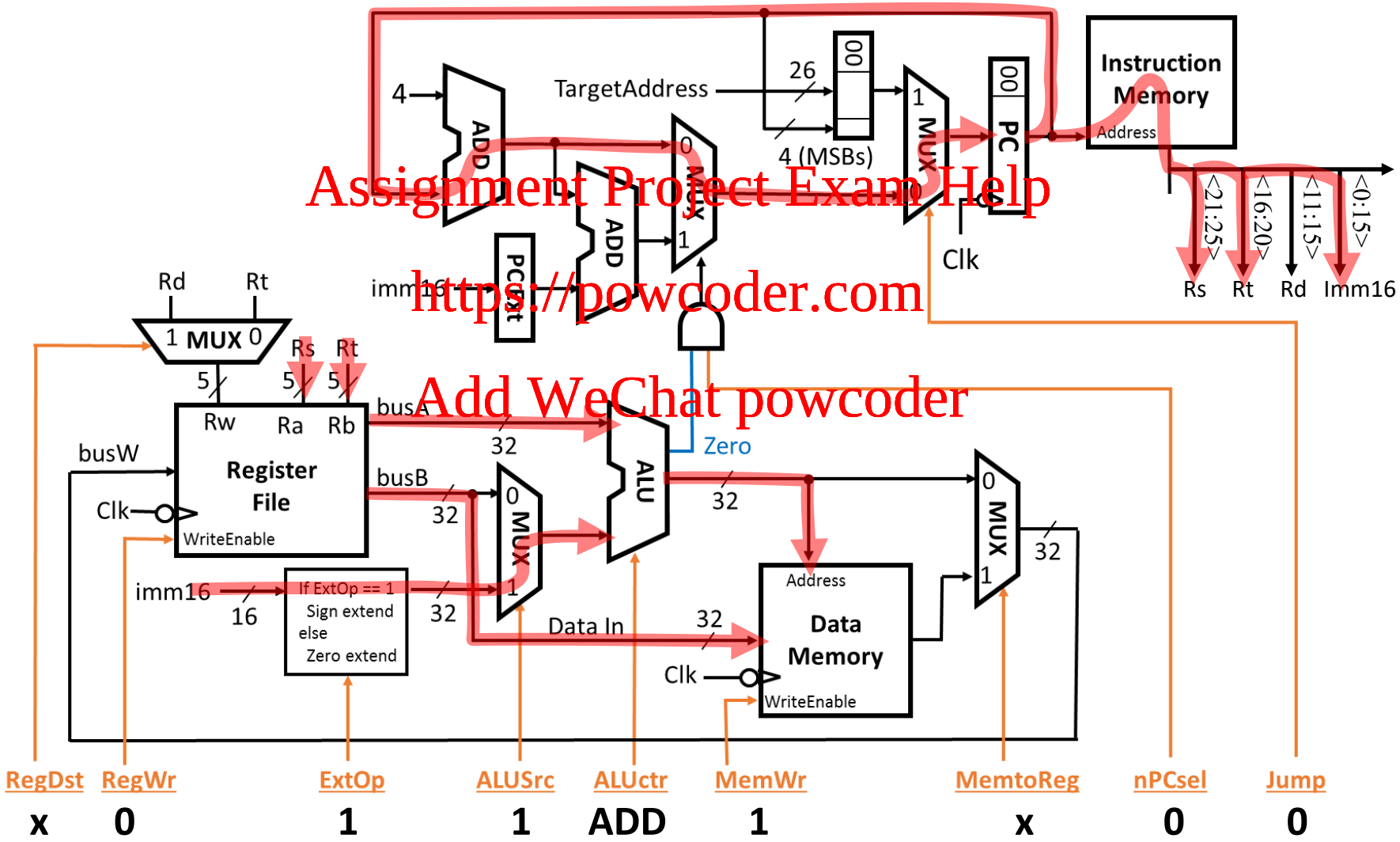
R[rt] = Data Memory {R[rs] + SignExt[imm16]}



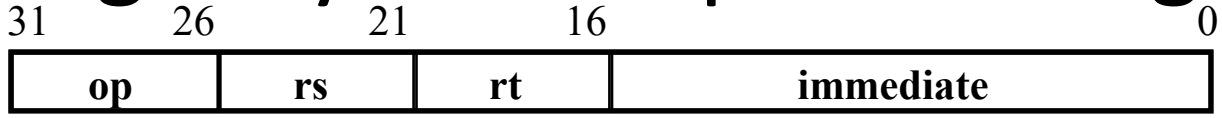
The Single Cycle Datapath during **STORE**?



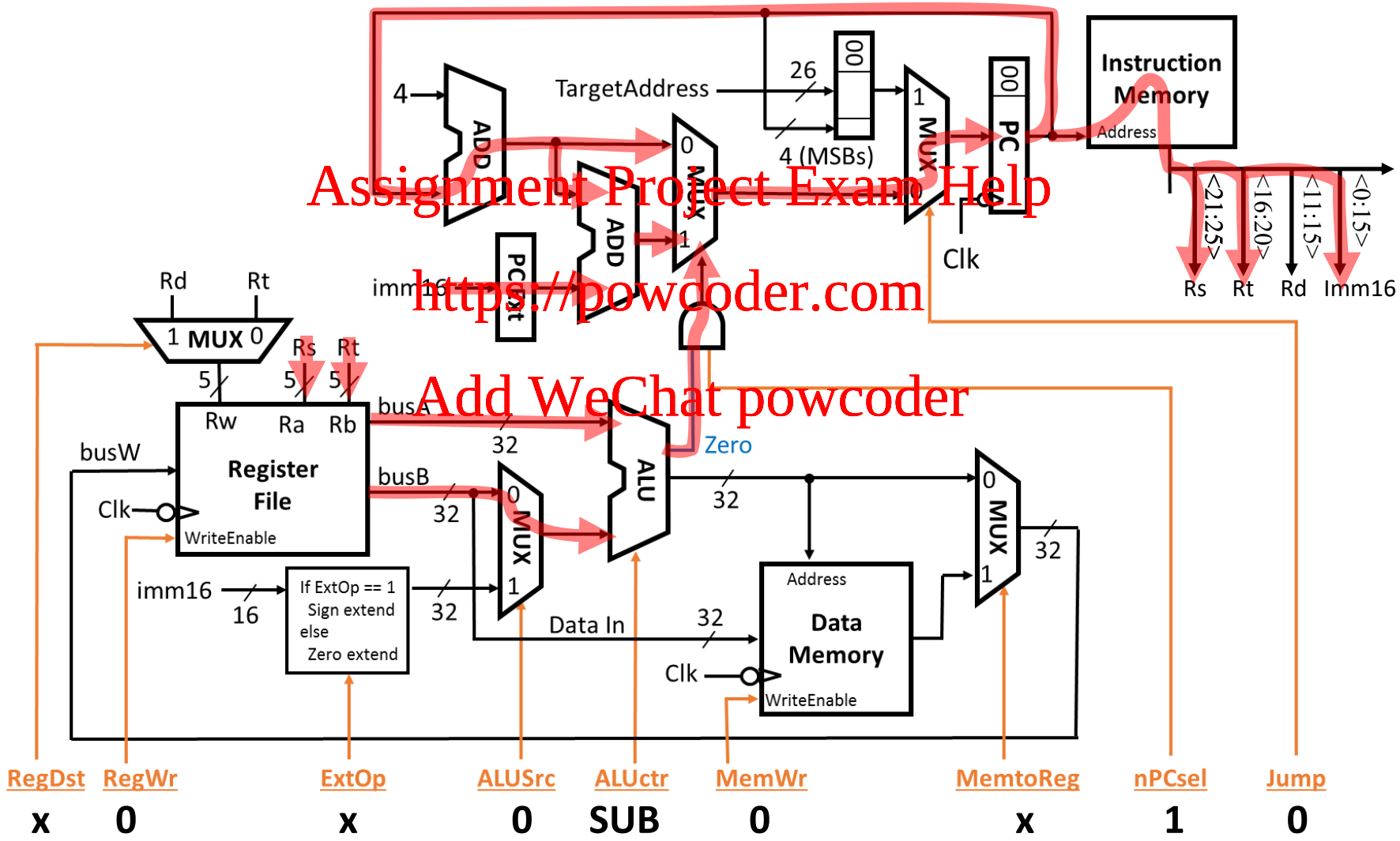
Data Memory {R[rs] + SignExt[imm16]} = R[rt]



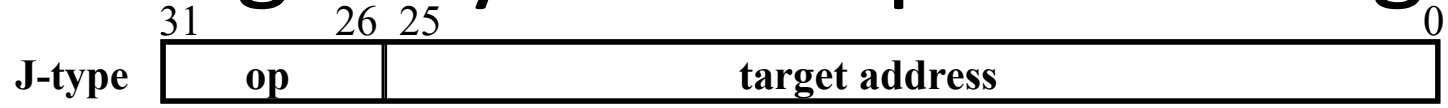
The Single Cycle Datapath during **BRANCH**



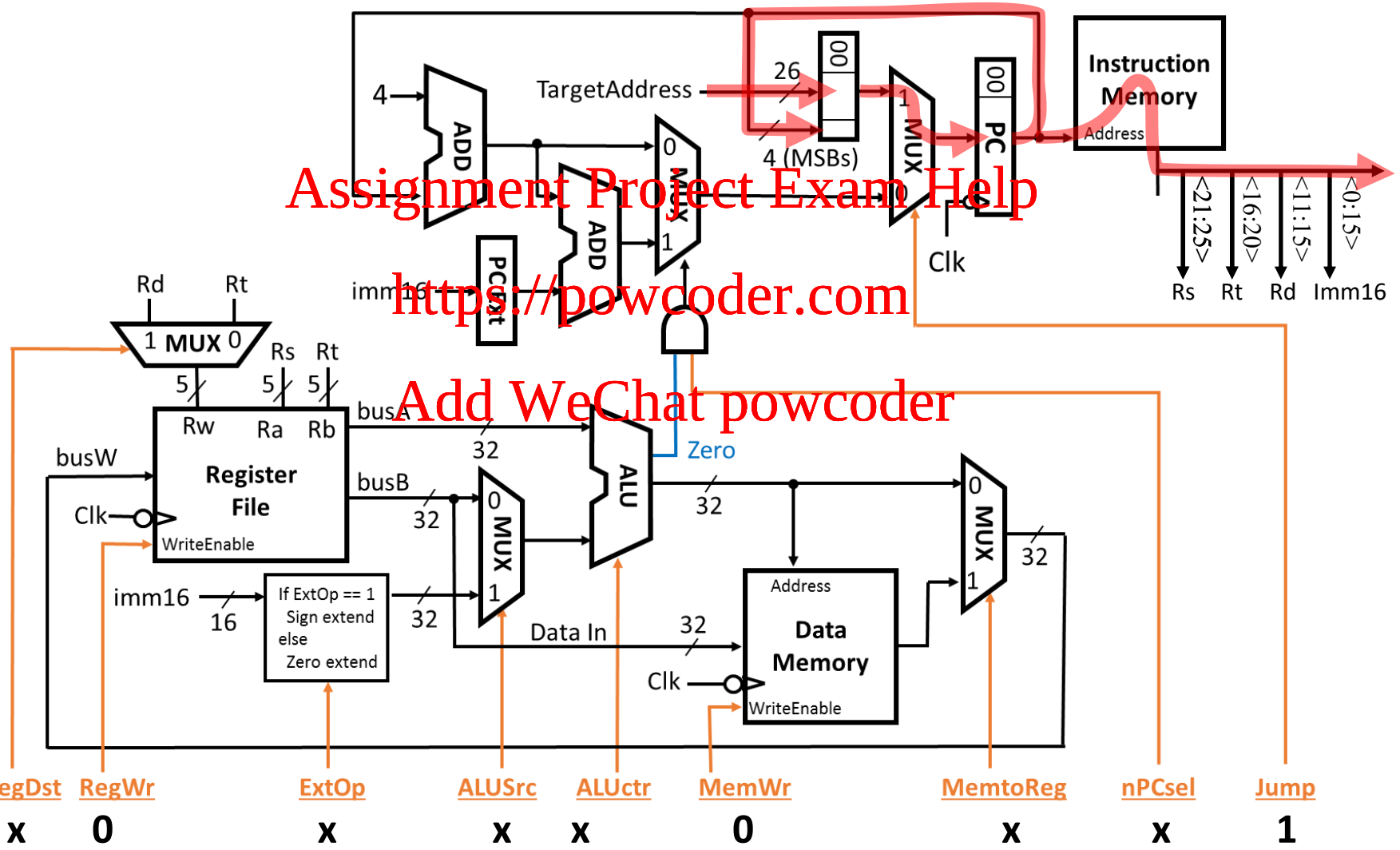
if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



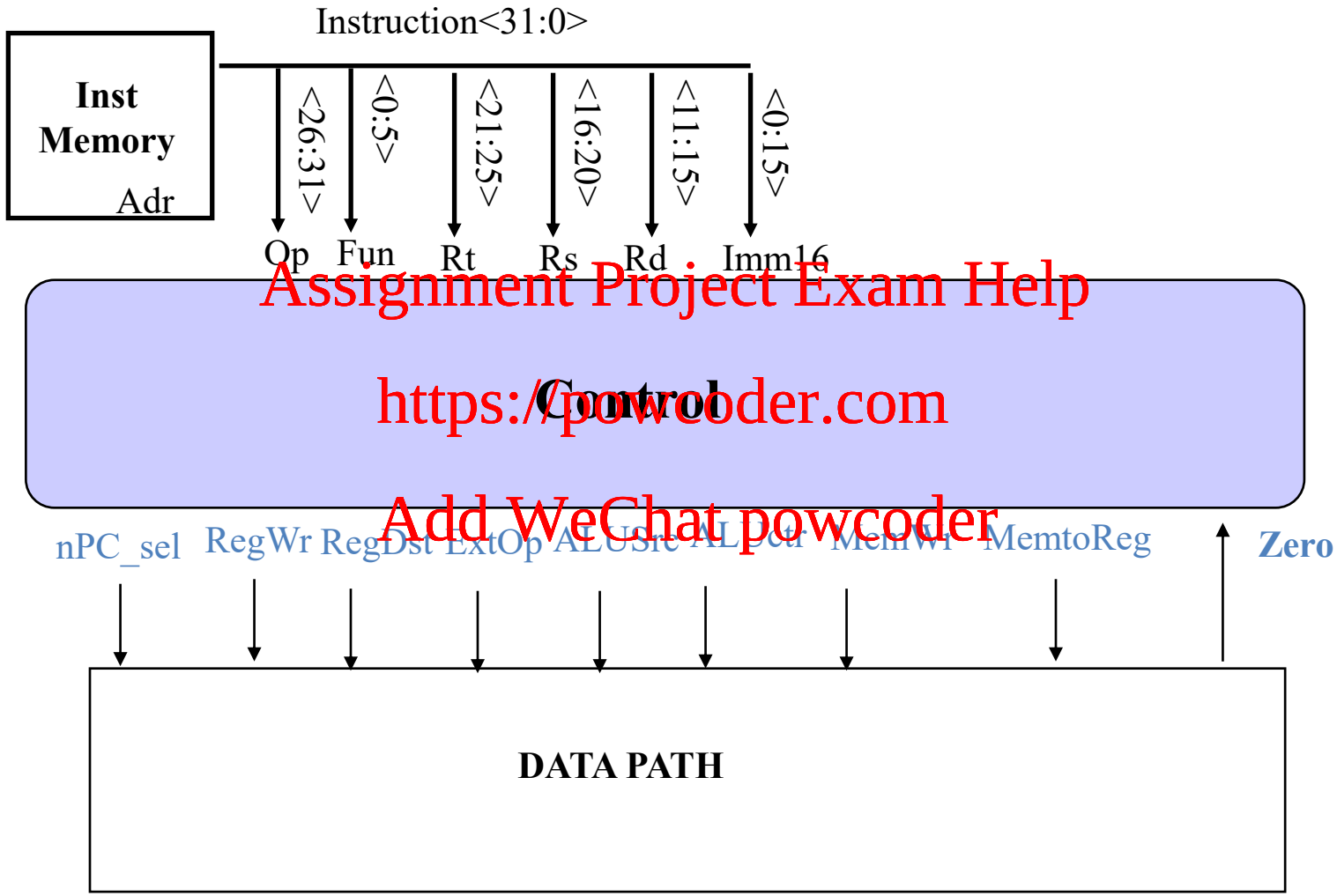
The Single Cycle Datapath during **JUMP**



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



Step 4: Given Datapath: RTL ⇒ Control

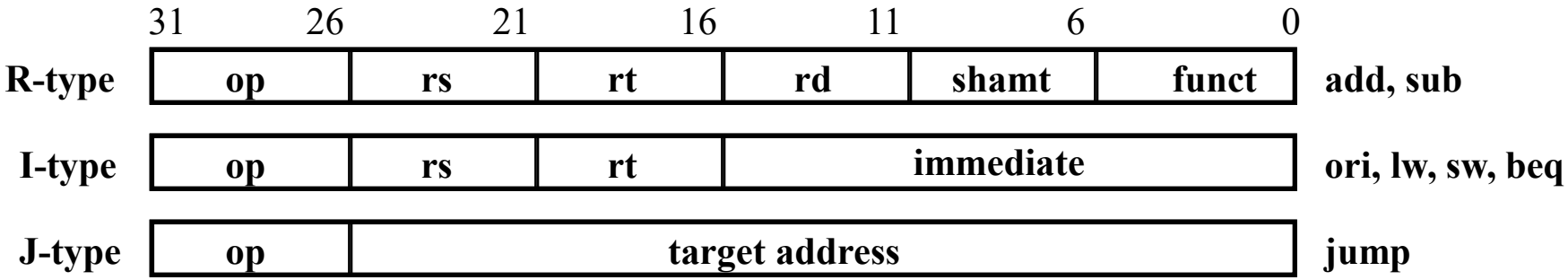


A Summary of the Control Signals

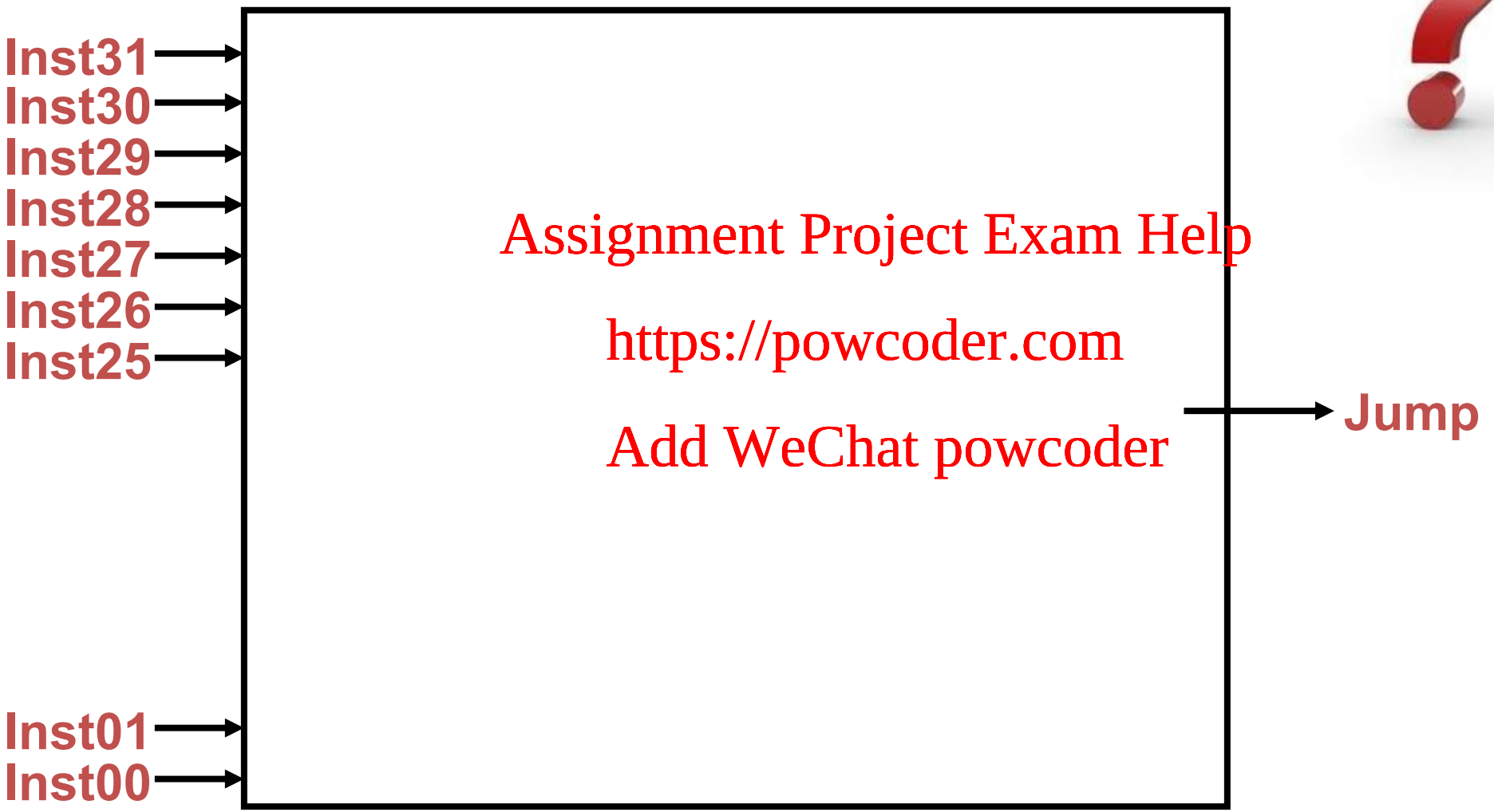
See Appendix B
Page 50 onward
(or green reference sheet)

	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	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

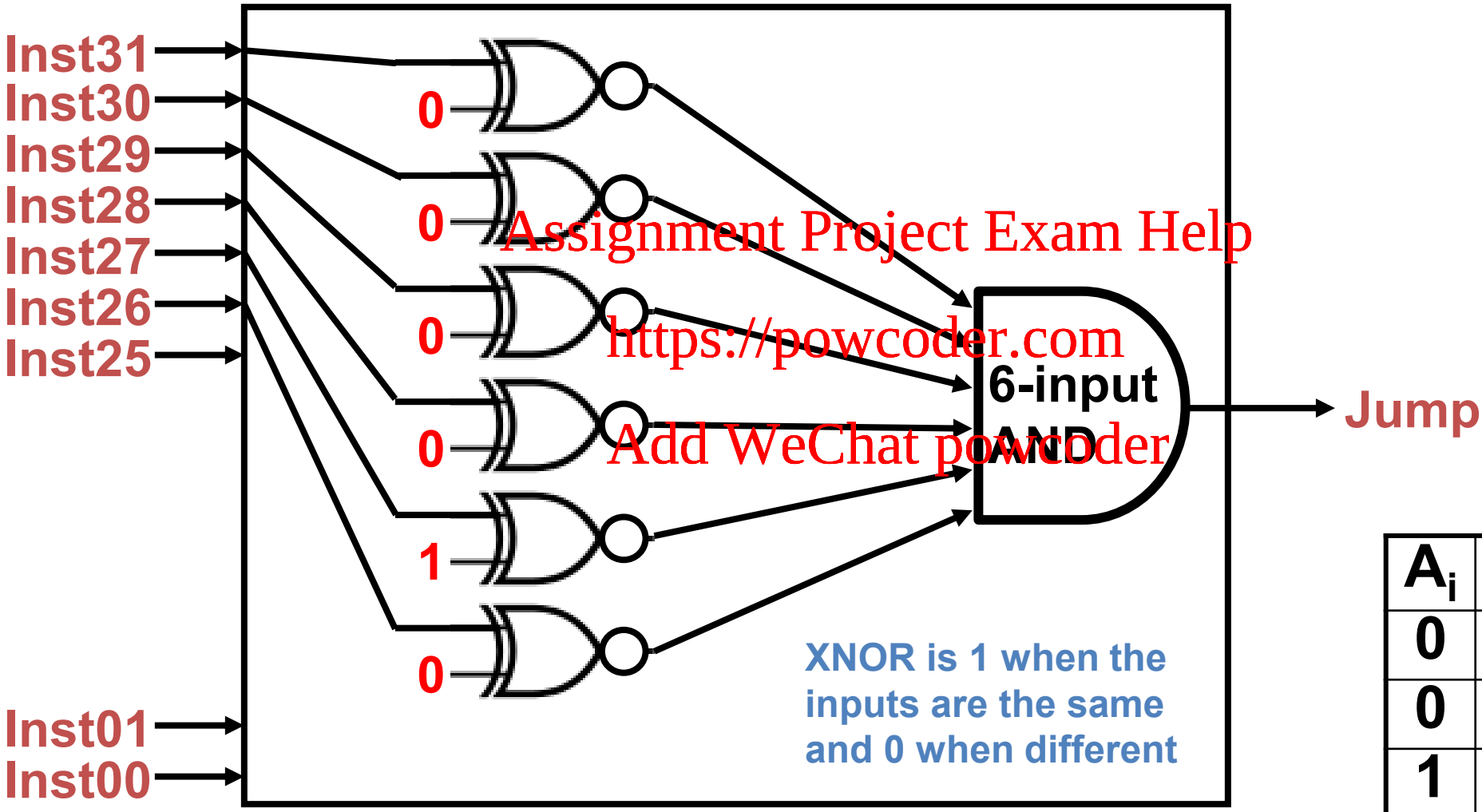
See 4.4, and Appendix C



Question: Build Control Logic to implement Jump



Control Logic to implement Jump



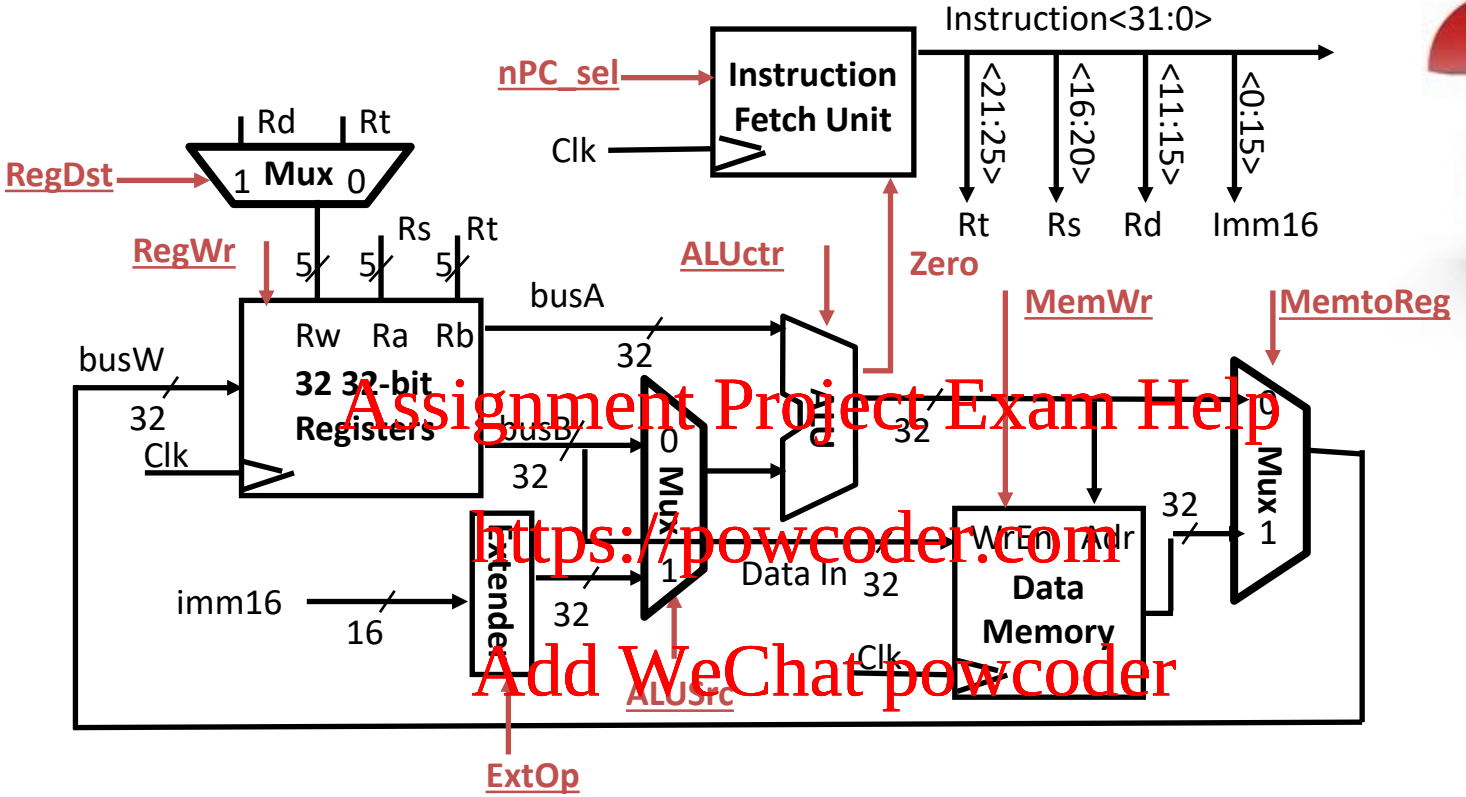
A_i	B_i	XNOR
0	0	1
0	1	0
1	0	0
1	1	1

Question, True or False

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

True, false, or don't care?

Question



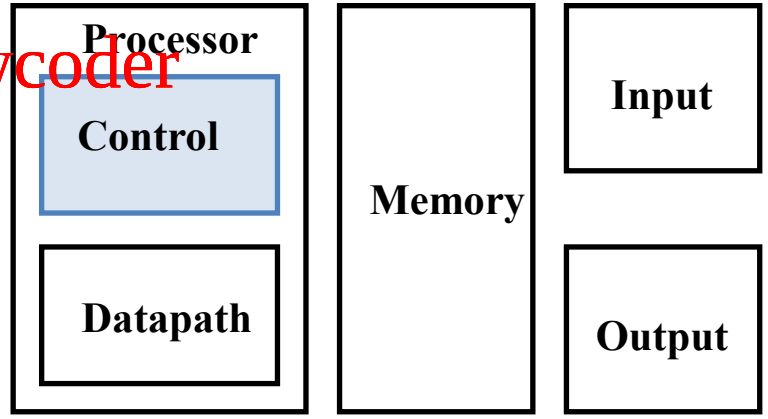
A. MemToReg='x' & ALUctr='sub'. SUB or BEQ?

B. Which of the two signals is not the same (i.e., 1, 0, x) for ADD, LW, SW? RegDst or ALUctr?

C. "Don't Care" signals are useful because we can simplify our implementation of the combinatorial Boolean control functions. F / T?

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