CSE 31
Computer Organization

Lecture 24 – CPU Design (4)

Announcement

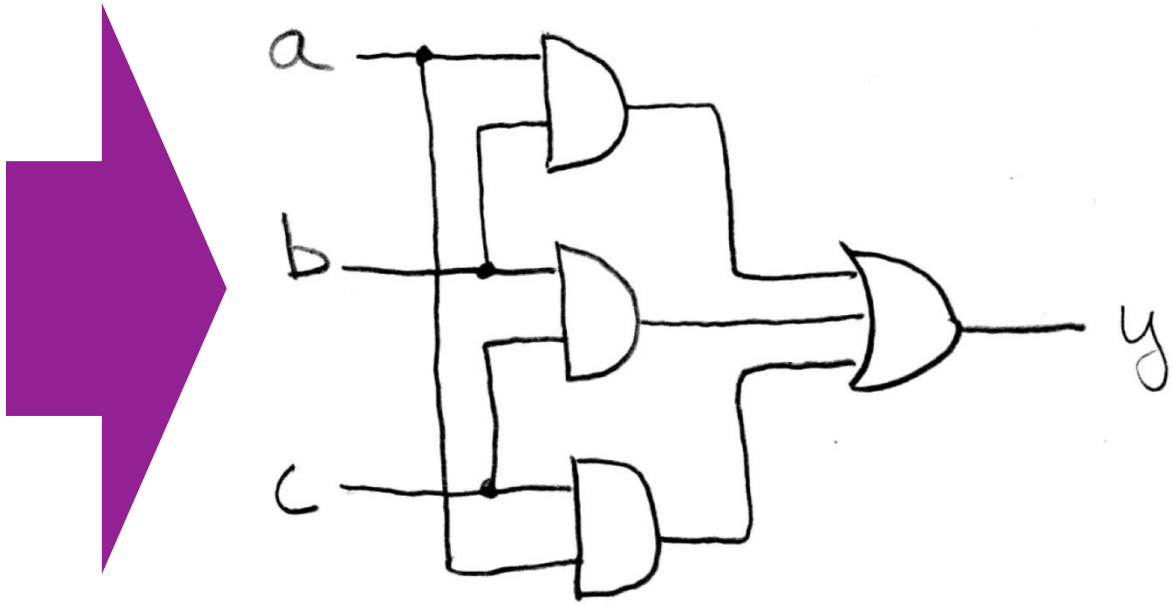
- Lab #10
  - Due this week
- HW #7 in CatCourses
  - Due Monday (5/6) at 11:59pm
- HW #8 in zyBooks (Through CatCourses)
  - Due Saturday (5/11) at 11:59pm
- Course evaluation online by 5/9

Announcement

- zyBooks assignment Re-dos
  - Re-submit at most 5 reading assignments **or** HW (**zyBooks only**)
  - Email to me (not your TAs)
    - Include your name, assignment numbers
    - (Monday) 5/13 at 11:59pm, no extension
  - Fill out online evaluation by 5/9, Thursday (70% of class)
- Final Exam
  - 5/11 (Saturday), 11:30 – 2:30pm
  - Cover all
  - Practice exam in CatCourses
  - Closed book
  - 2 sheet of note (8.5” x 11”)
  - MIPS reference sheet will be provided
  - Review: 5/10 (Friday) 2-4pm, COB2 140

TT ⇒ Gates (e.g., majority circ.)

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Canonical forms (1/2)

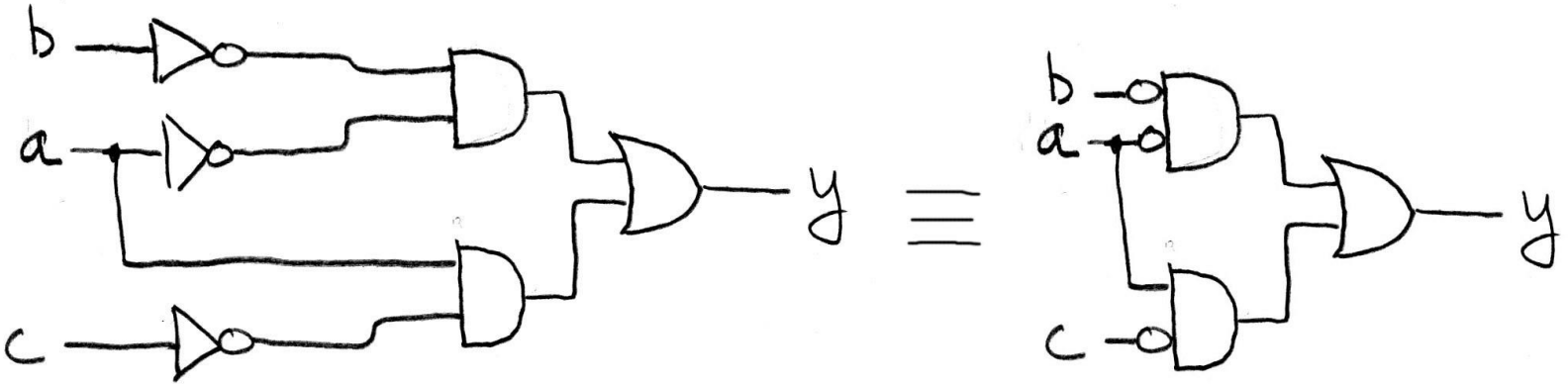
	abc	y
$\overline{a} \cdot \overline{b} \cdot \overline{c}$	000	1
$\overline{a} \cdot \overline{b} \cdot c$	001	1
	010	0
	011	0
$a \cdot \overline{b} \cdot \overline{c}$	100	1
	101	0
$a \cdot b \cdot \overline{c}$	110	1
	111	0

$$y = \overline{a}\overline{b}\overline{c} + \overline{a}\overline{b}c + a\overline{b}\overline{c} + ab\overline{c}$$

Sum-of-products
Sum-of-minterms
(ORs of ANDs)

Canonical forms (2/2)

$$
\begin{aligned}
y &= \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + a\bar{b}\bar{c} + ab\bar{c} \\
&= \bar{a}\bar{b}(\bar{c} + c) + a\bar{c}(\bar{b} + b) && \textit{distribution} \\
&= \bar{a}\bar{b}(1) + a\bar{c}(1) && \textit{complementarity} \\
&= \bar{a}\bar{b} + a\bar{c} && \textit{identity}
\end{aligned}
$$



Canonical forms example

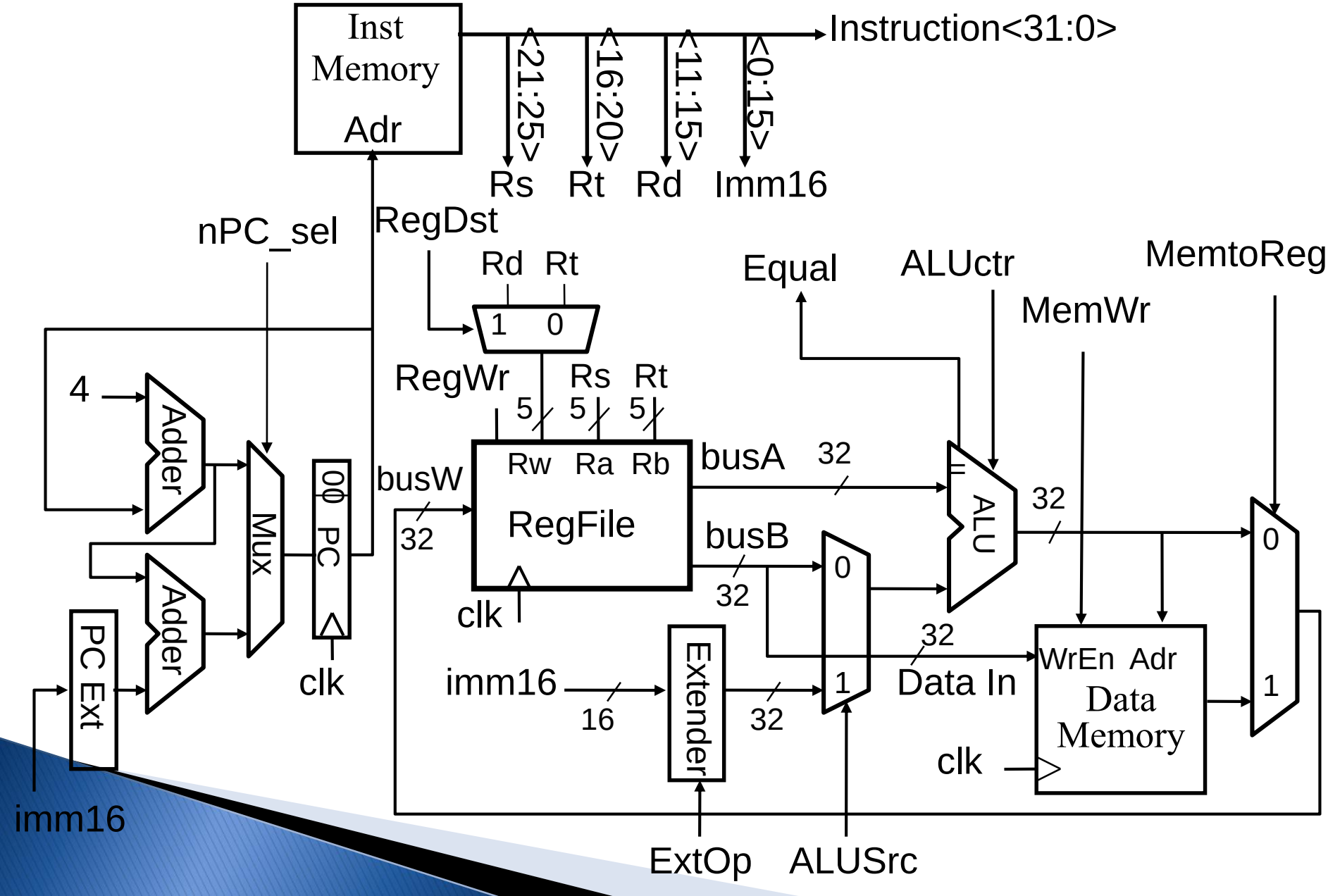
A	B	C	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

$= \bar{a}\bar{b}c + \bar{a}bc + ab\bar{c}$

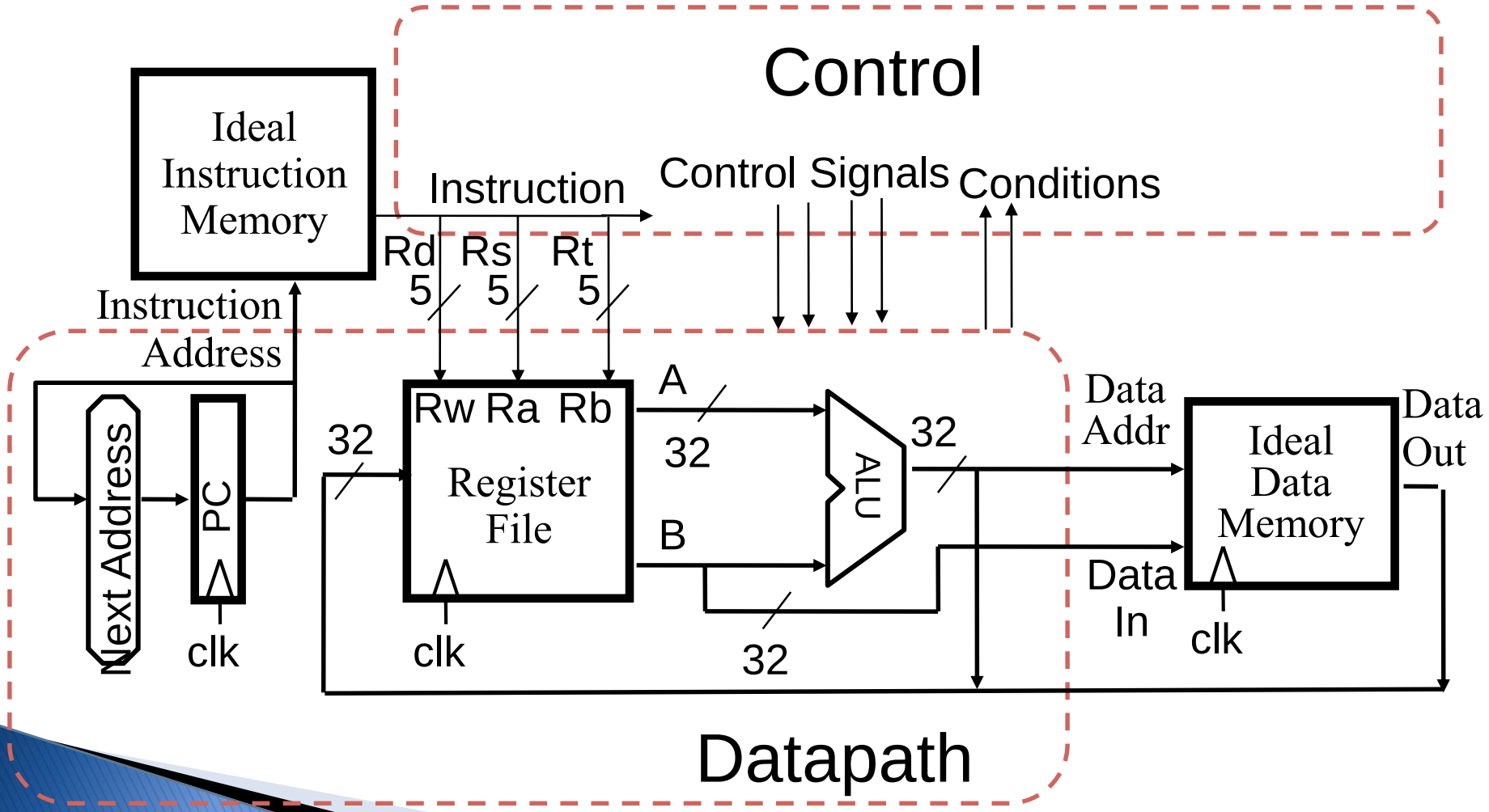
$= \bar{a}c(\bar{b} + b) + ab\bar{c}$

$= \bar{a}c + ab\bar{c}$

Single Cycle Datapath



Abstract View of the Implementation

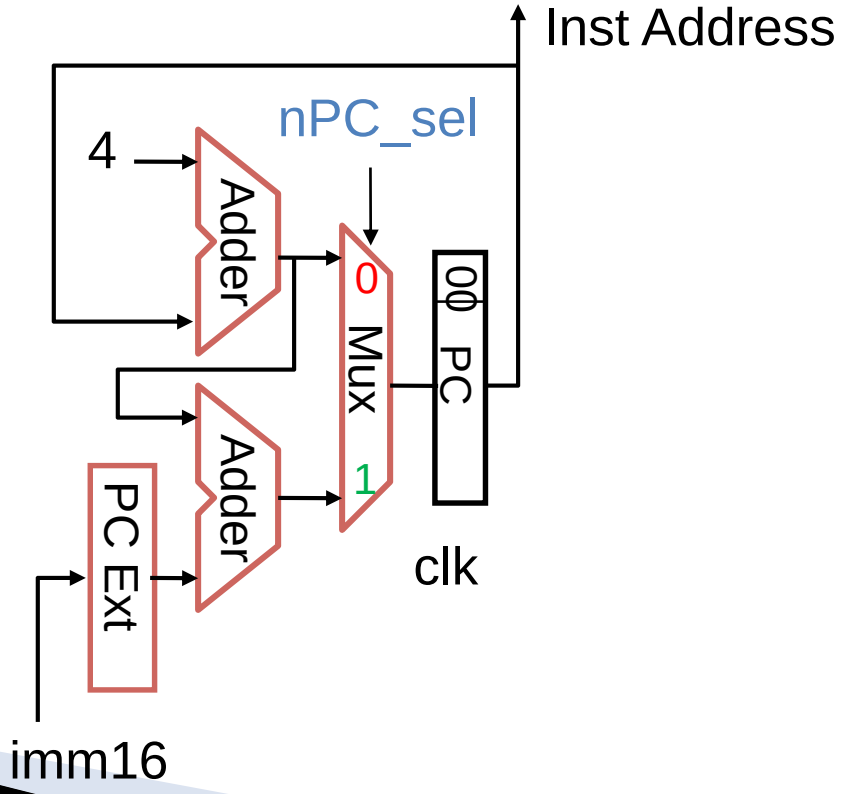


Meaning of the Control Signals

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

  "n"=next

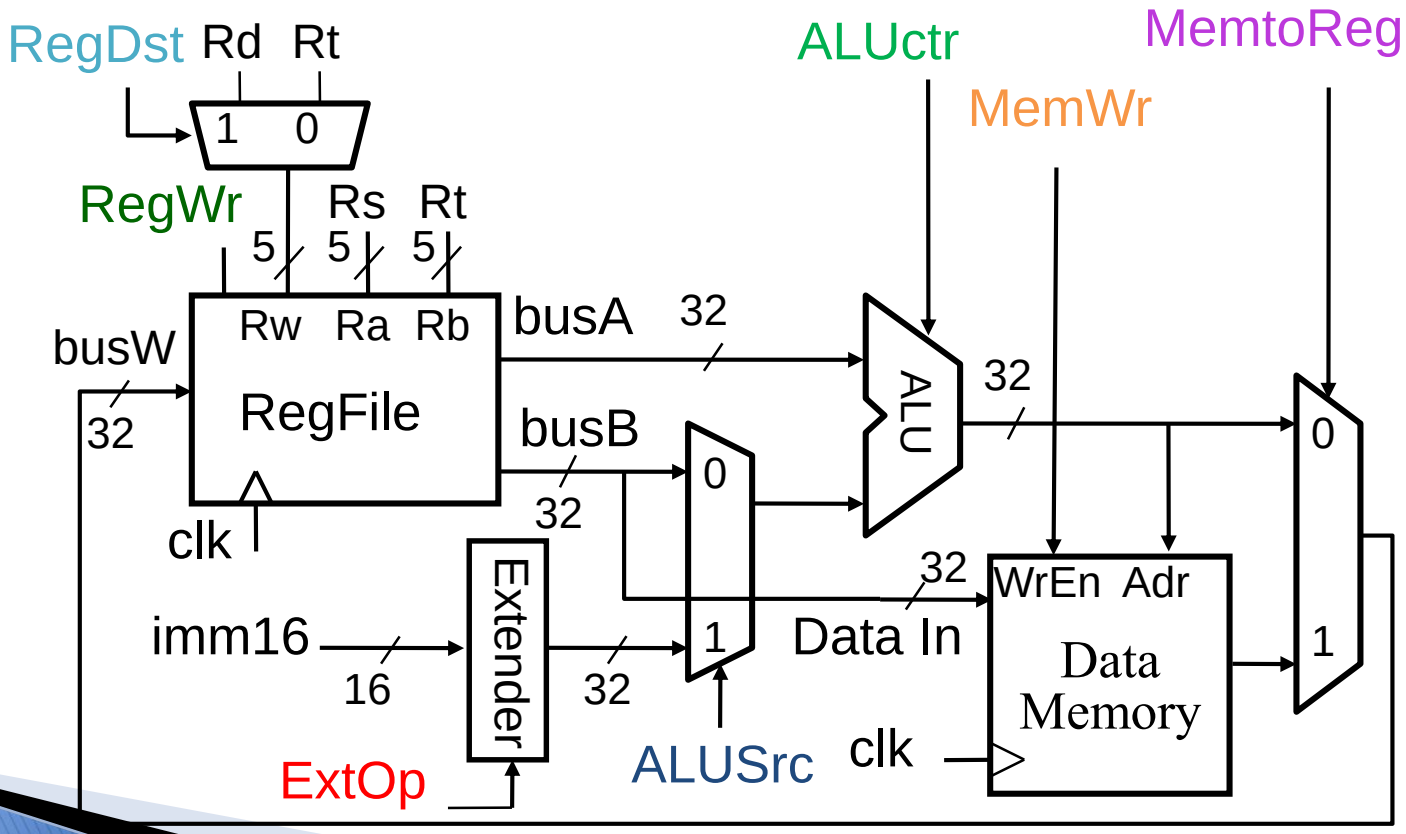
- Later in lecture: higher-level connection between mux and branch condition



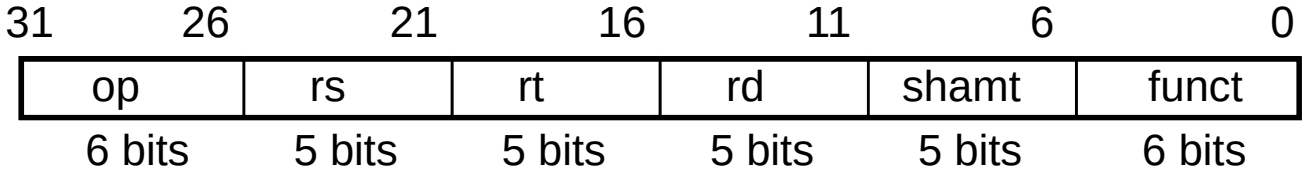
Meaning of the Control Signals

- 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



The Add Instruction



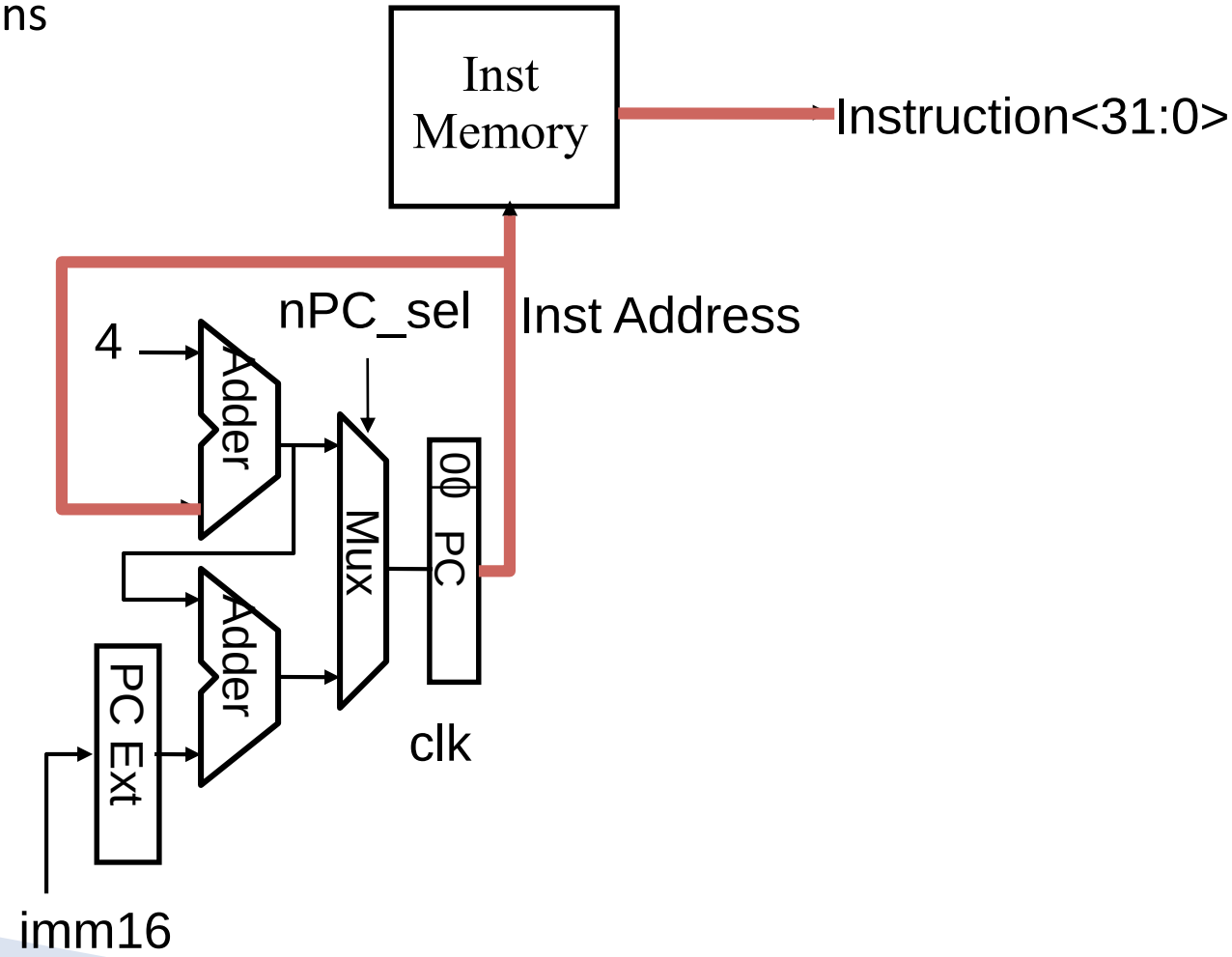

```
add rd, rs, rt
```

- 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

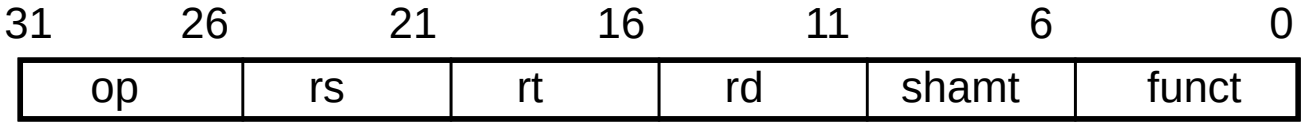
Instruction Fetch Unit start of Add

- Fetch the instruction from Instruction memory:

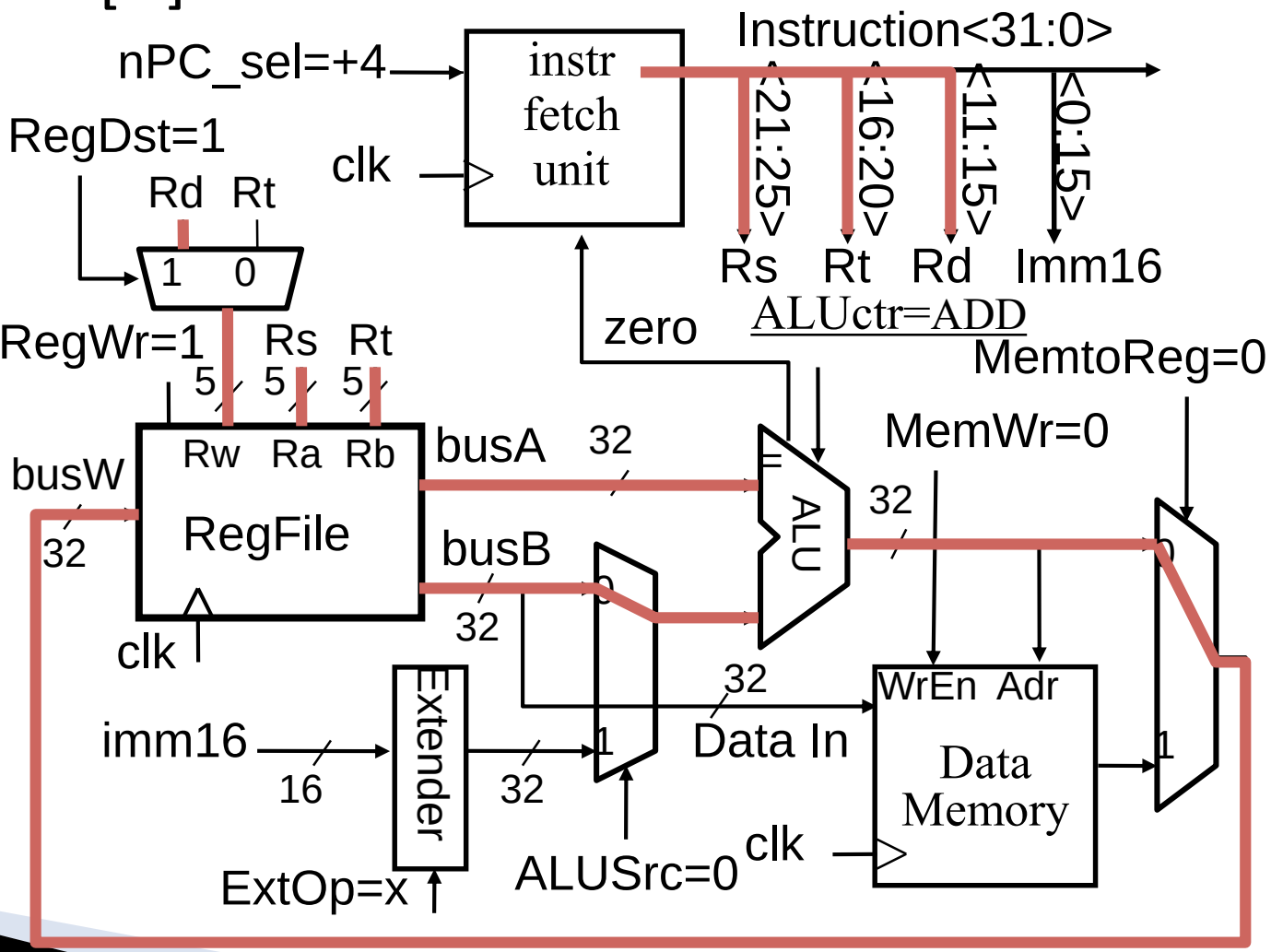
  Instruction = MEM[PC]
  - same for all instructions



The Single Cycle Datapath during Add

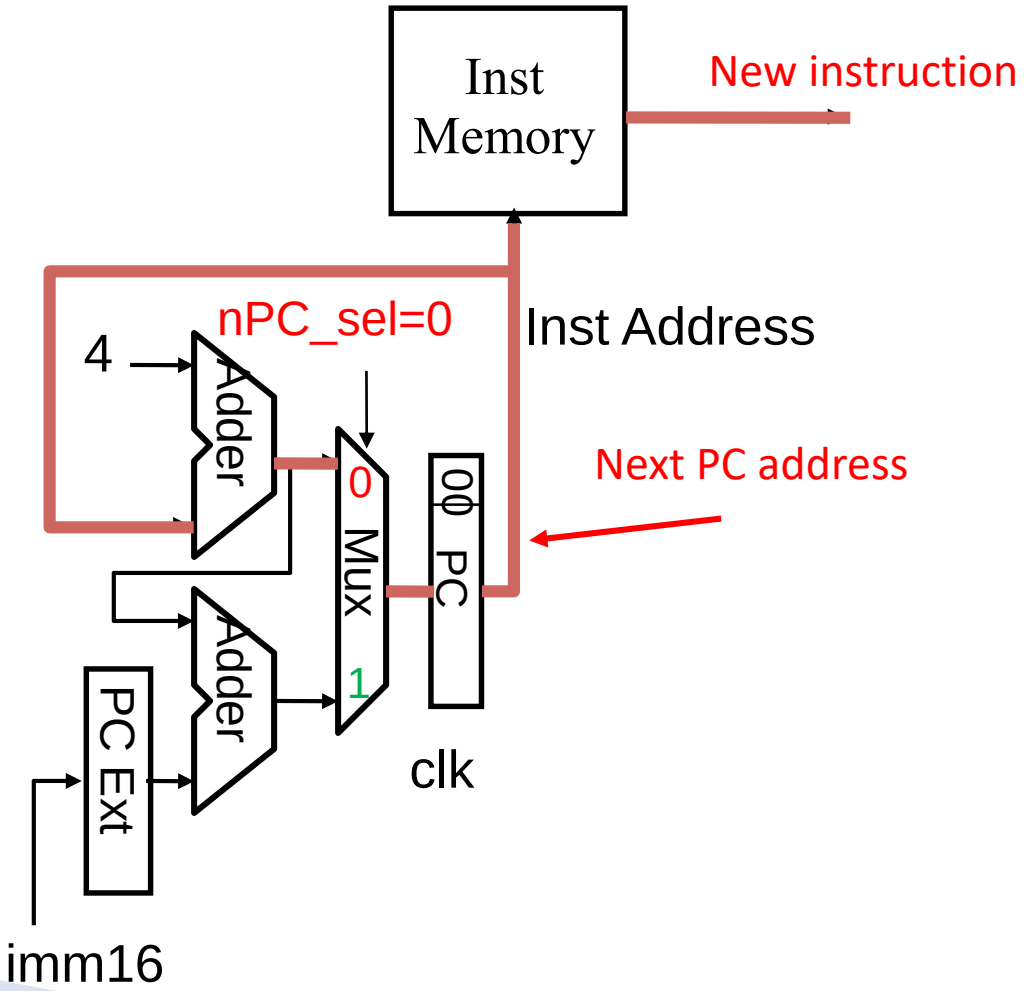


R[rd] = R[rs] + R[rt]

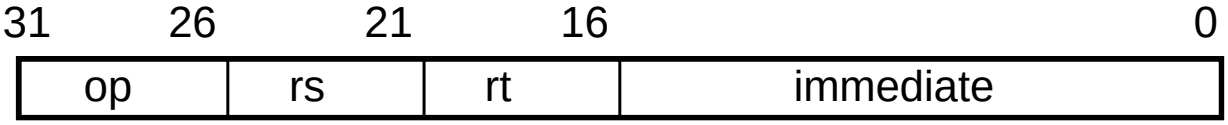


Instruction Fetch Unit end of Add

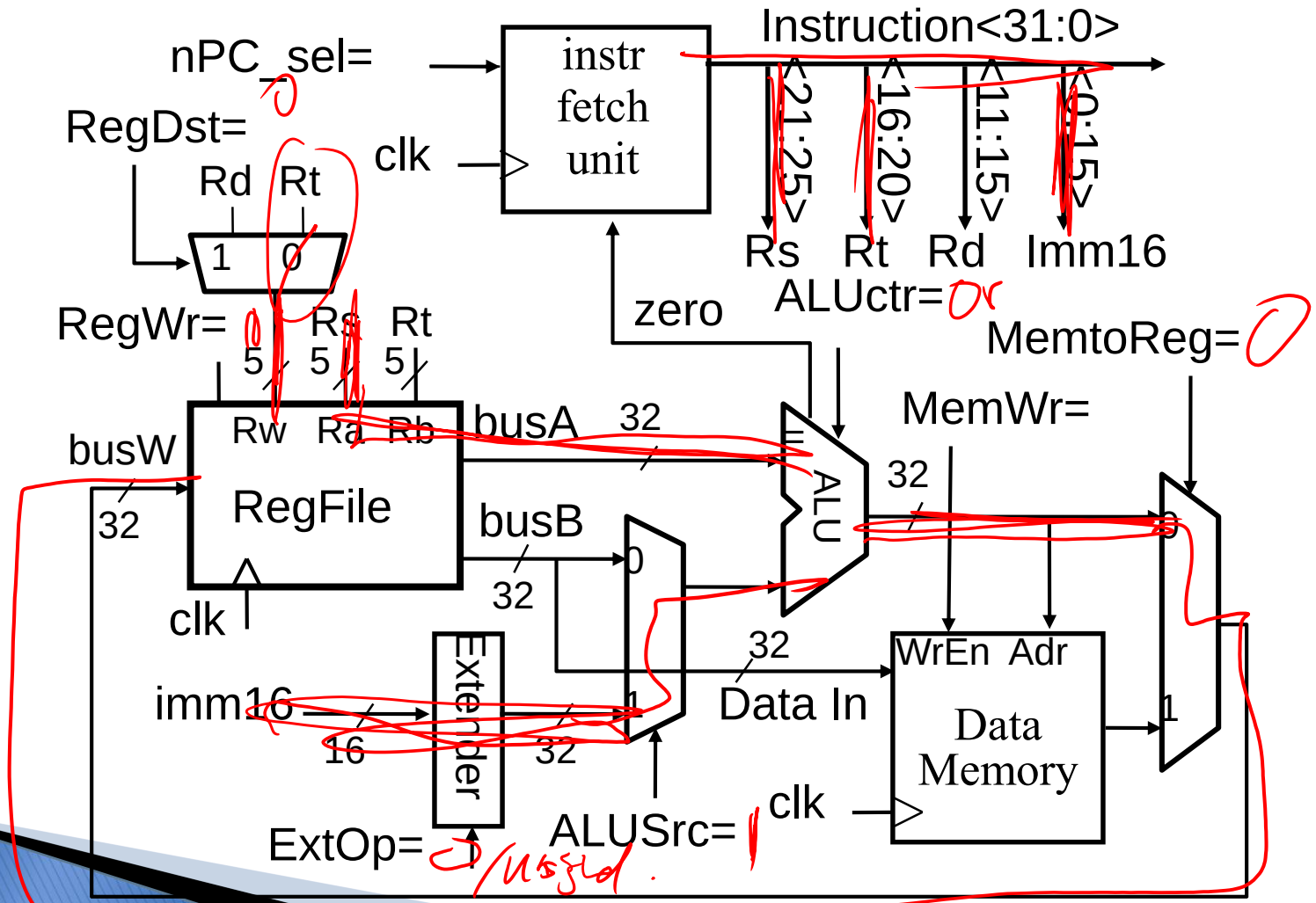
- PC = PC + 4
  - This is the same for all instructions except: Branch and Jump



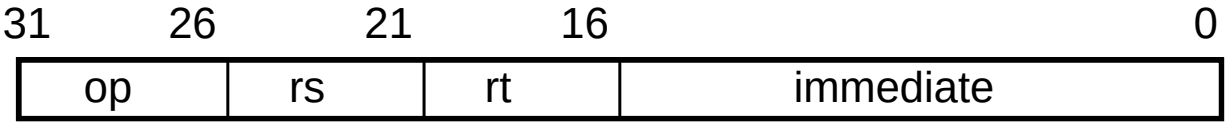
Single Cycle Datapath for Ori



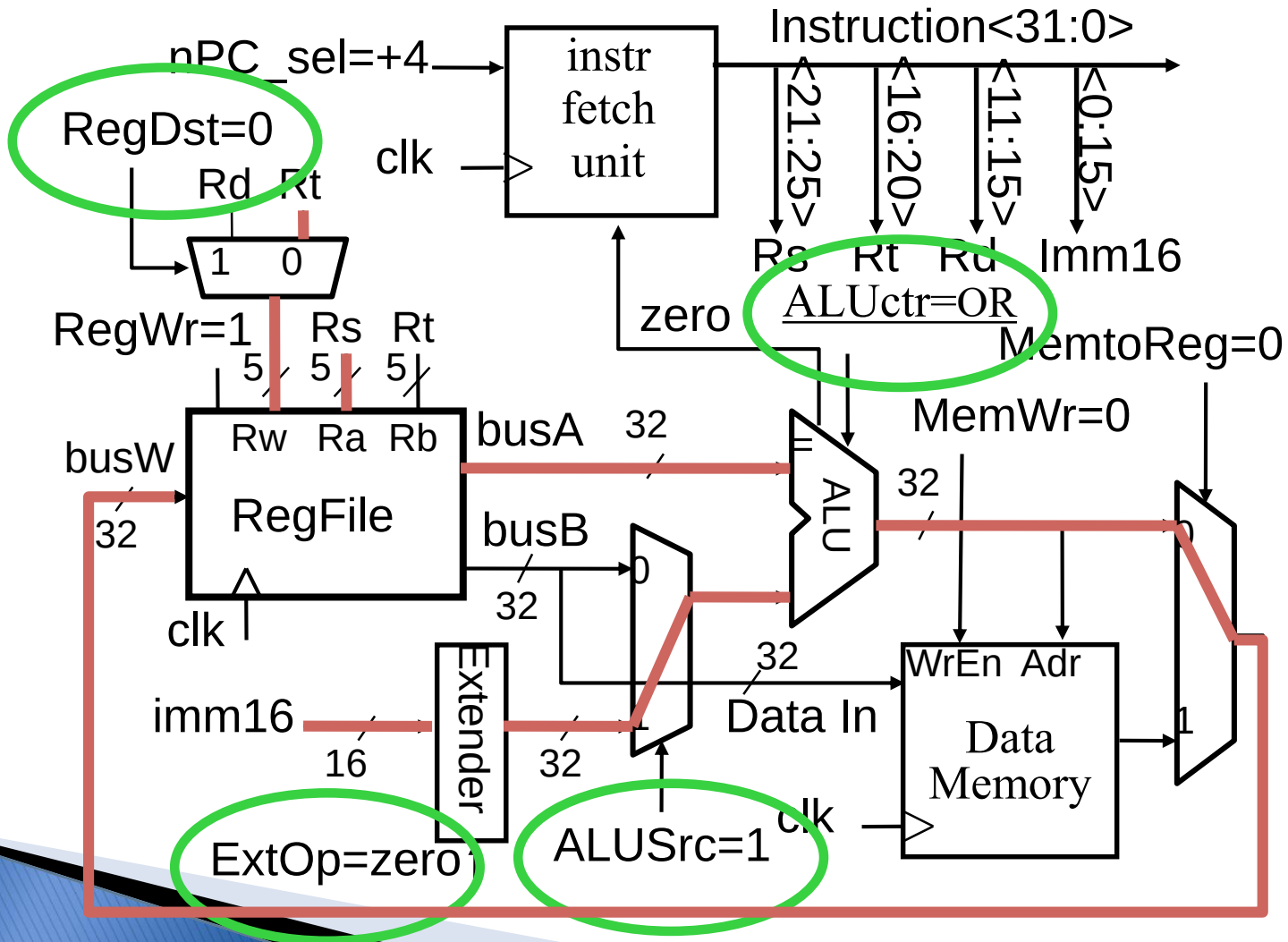
- R[rt] = R[rs] OR ZeroExt[Imm16]



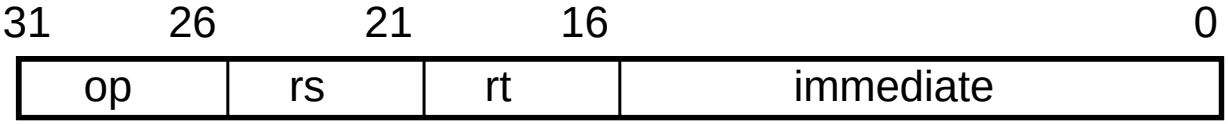
Single Cycle Datapath for Ori



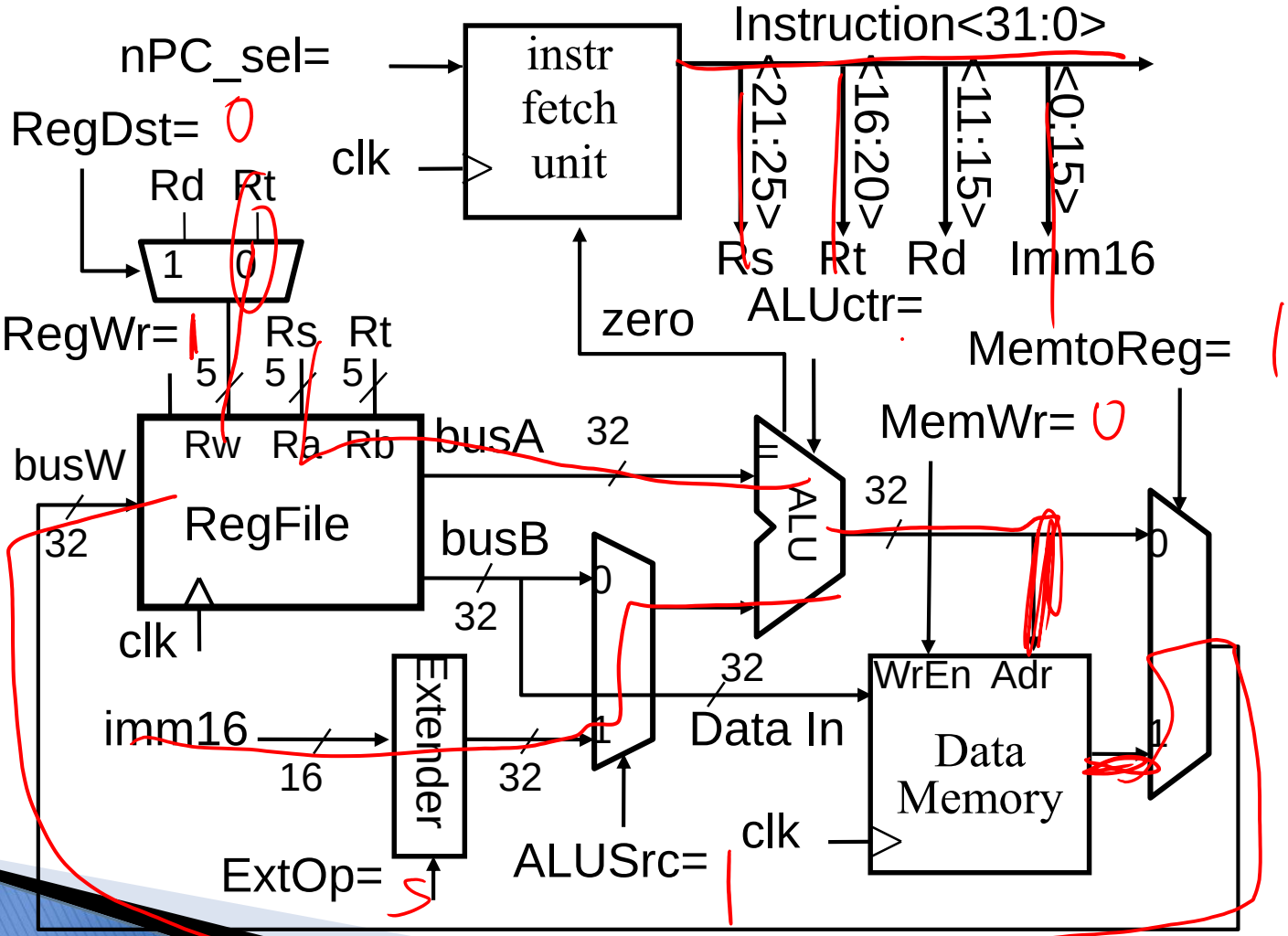
- R[rt] = R[rs] OR ZeroExt[Imm16]



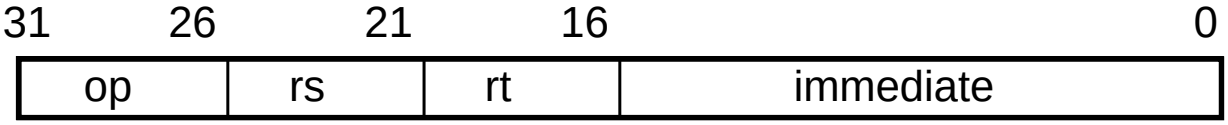
Single Cycle Datapath for LW



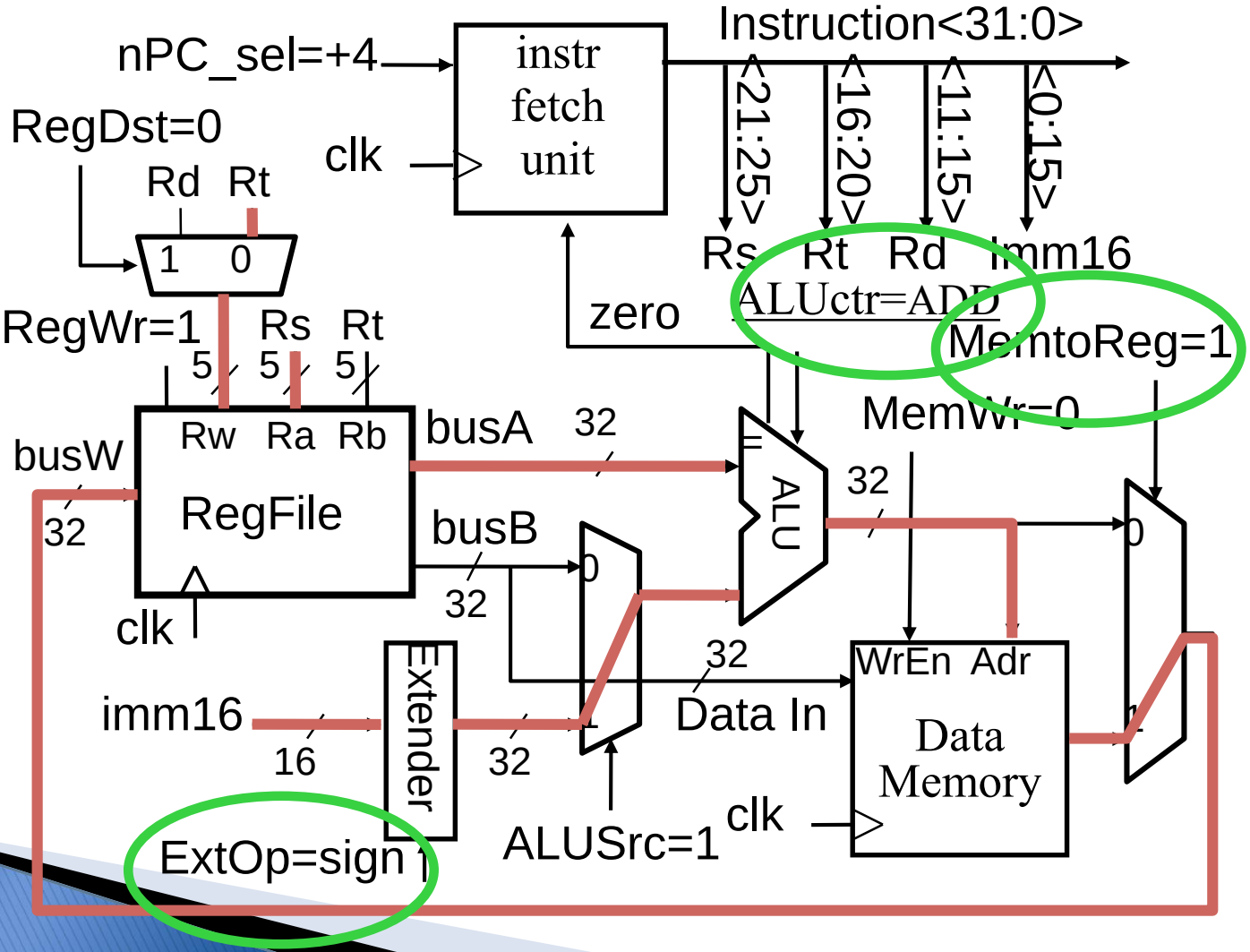
- R[rt] = Data Memory {R[rs] + SignExt[imm16]}



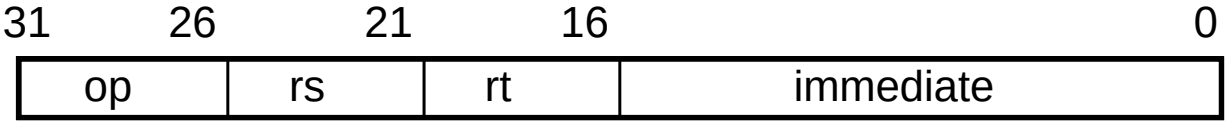
Single Cycle Datapath for LW



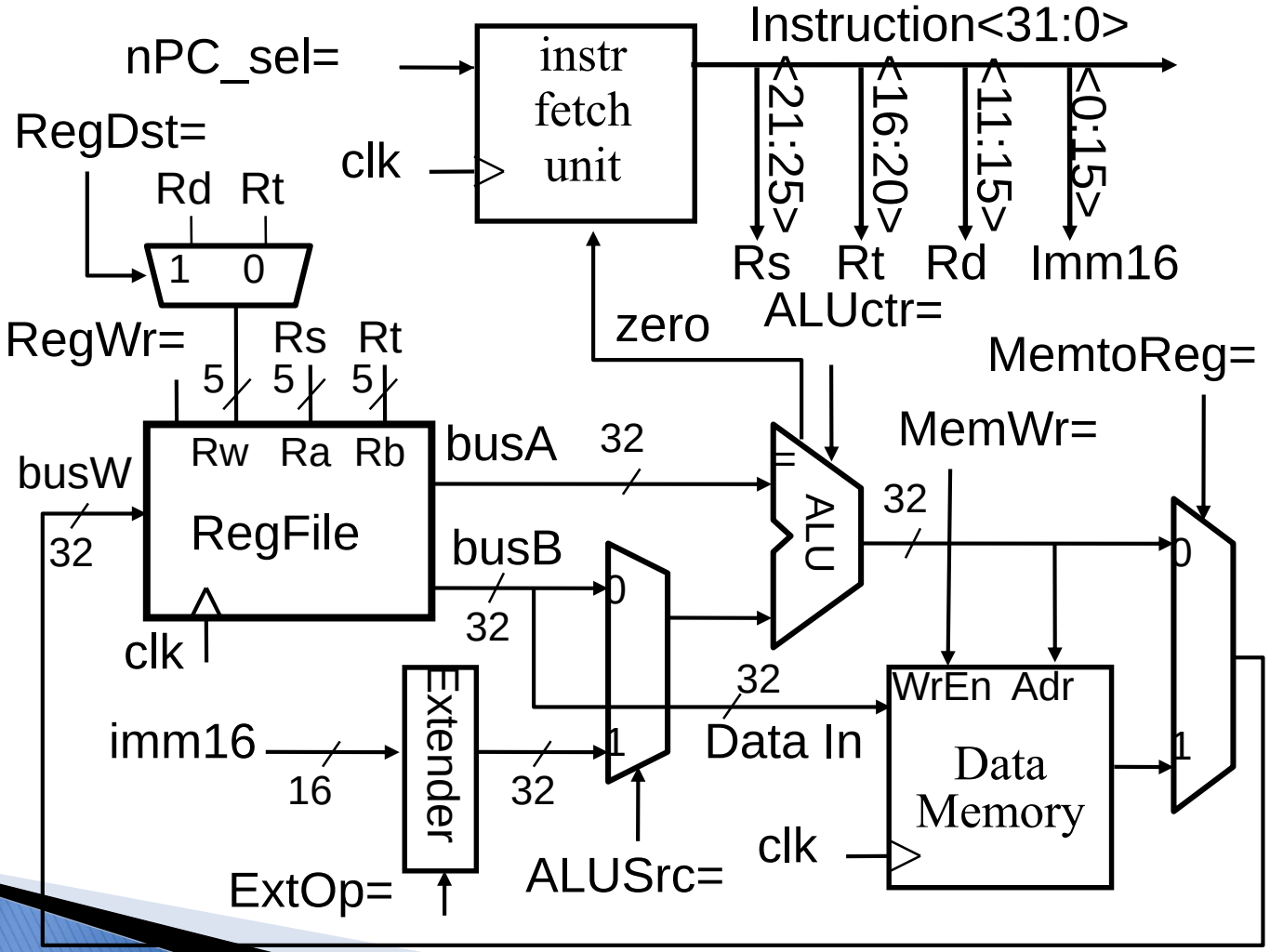
- R[rt] = Data Memory {R[rs] + SignExt[imm16]}



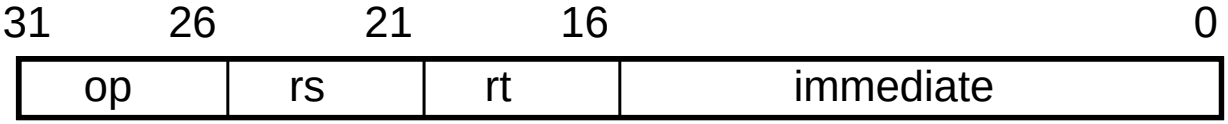
Single Cycle Datapath for SW



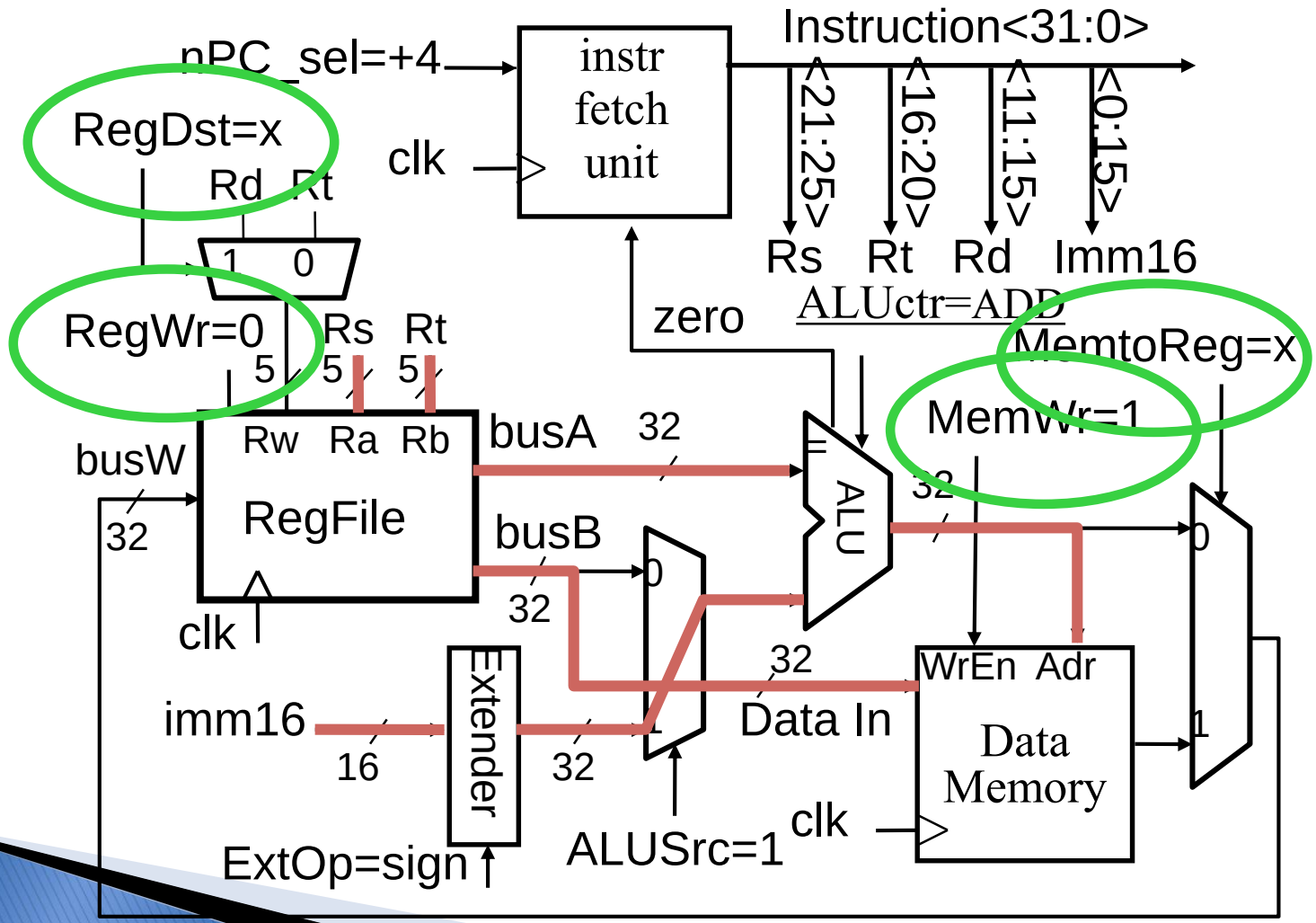
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



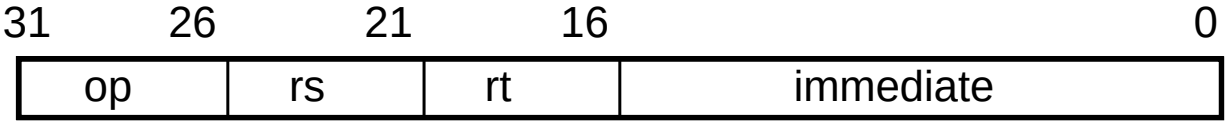
Single Cycle Datapath for SW



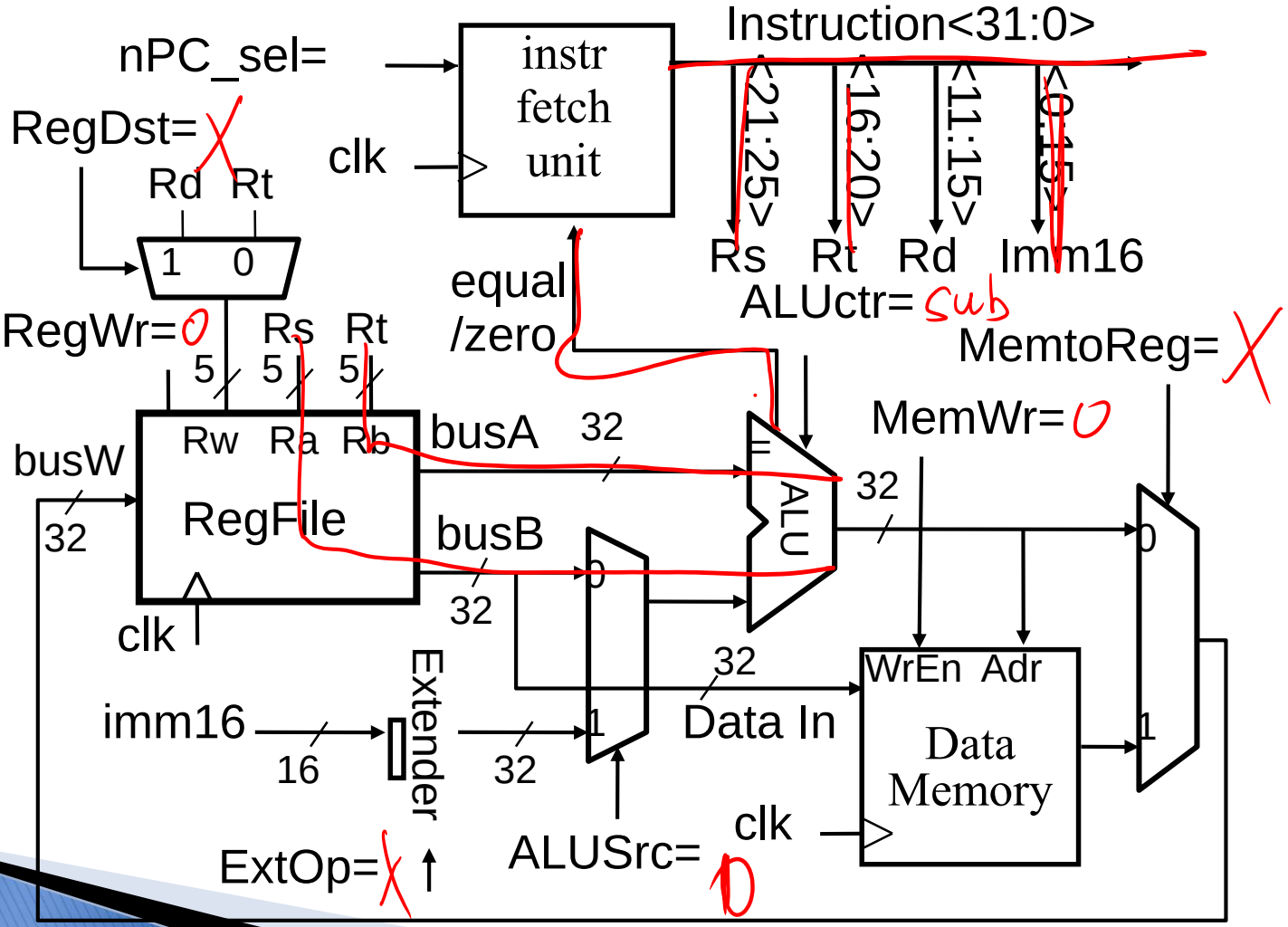
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



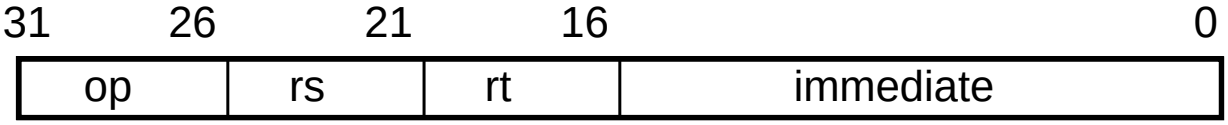
Single Cycle Datapath for Branch



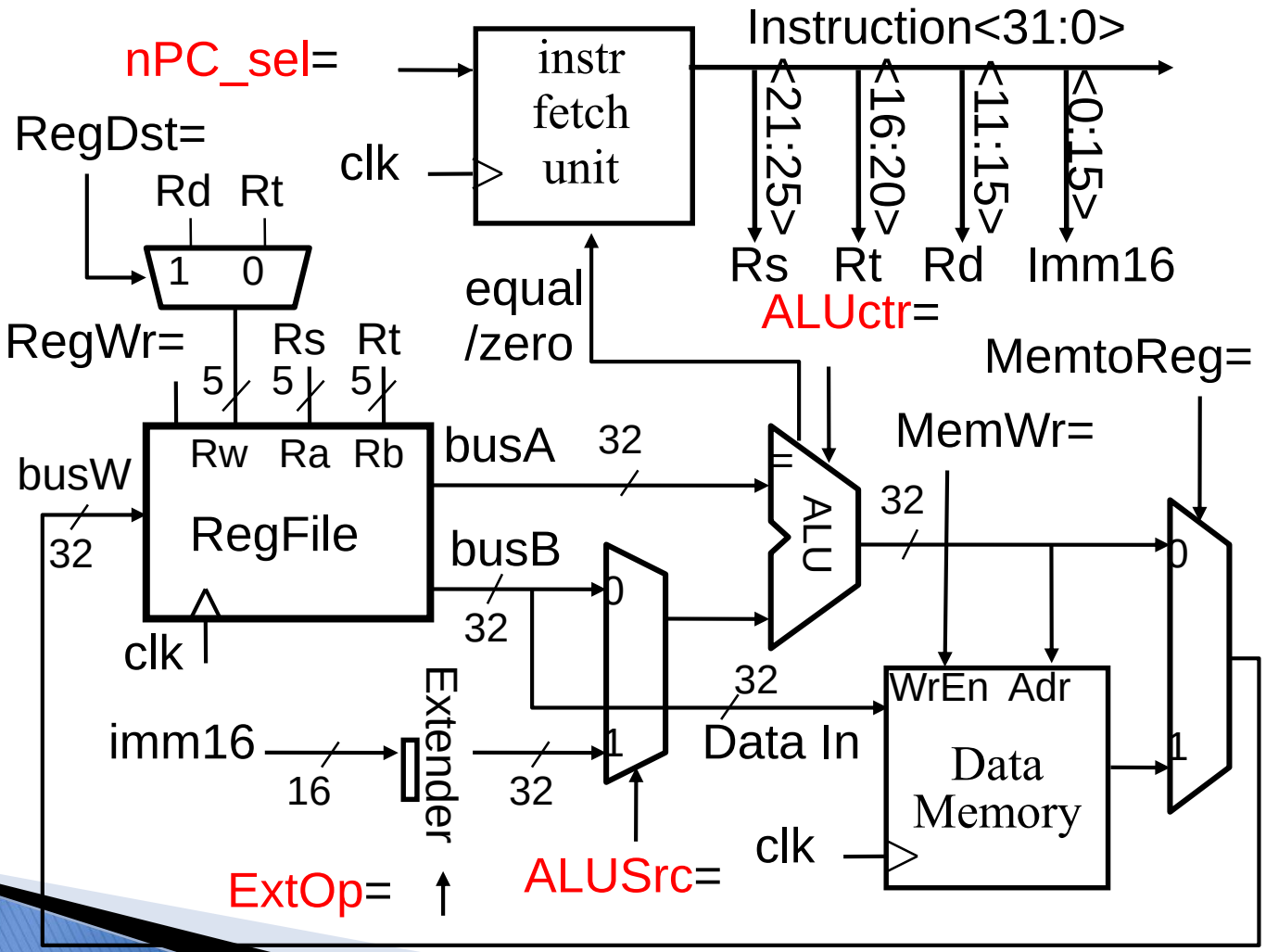
- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



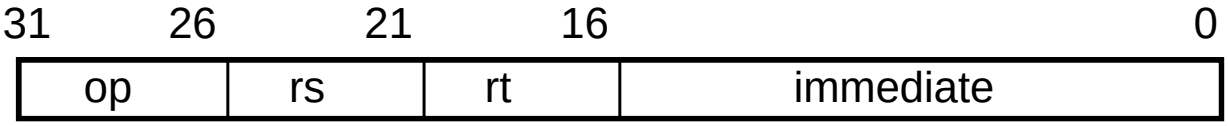
Single Cycle Datapath for Branch



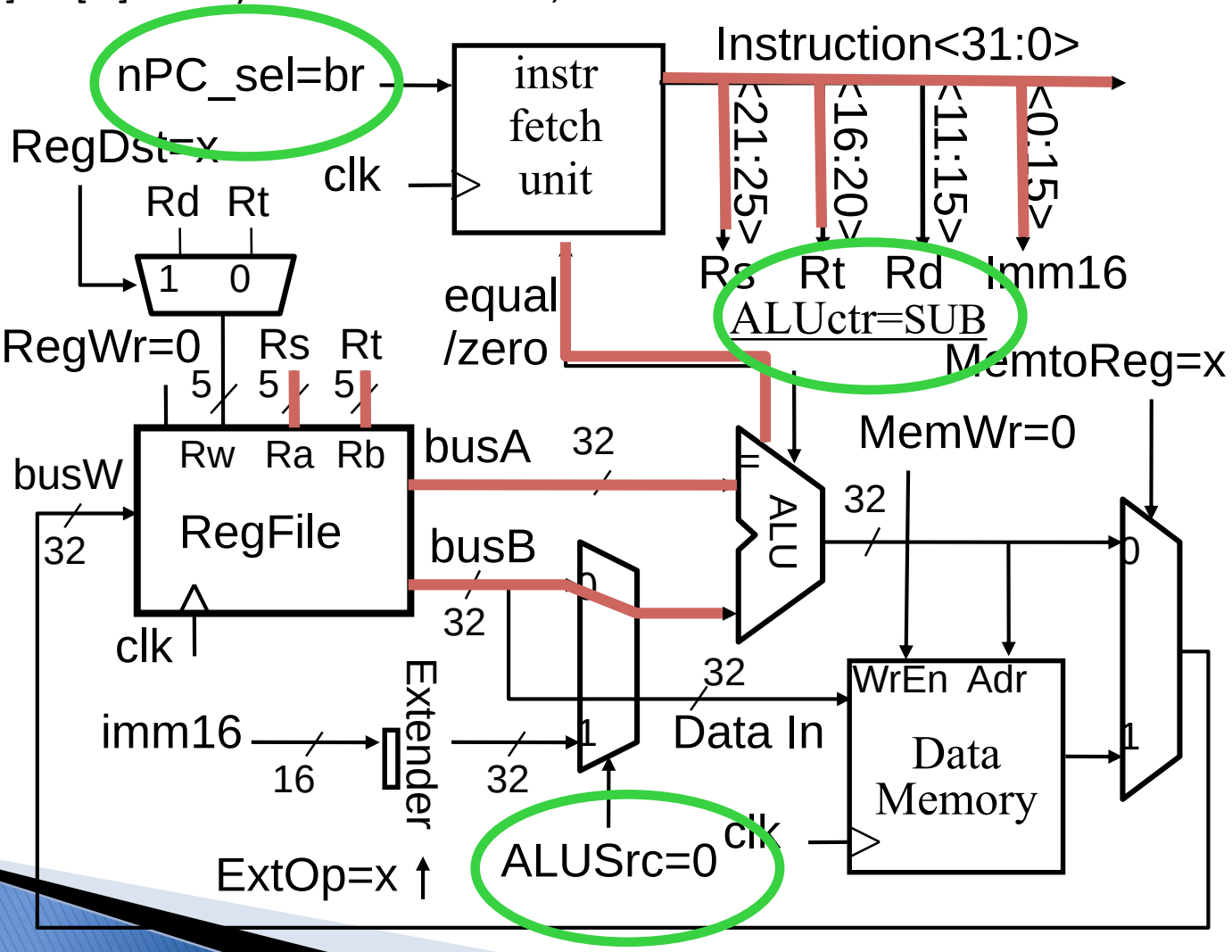
- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



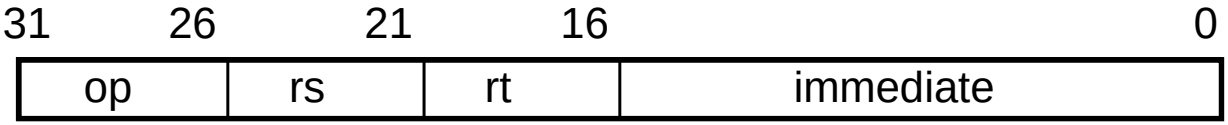
Single Cycle Datapath for Branch



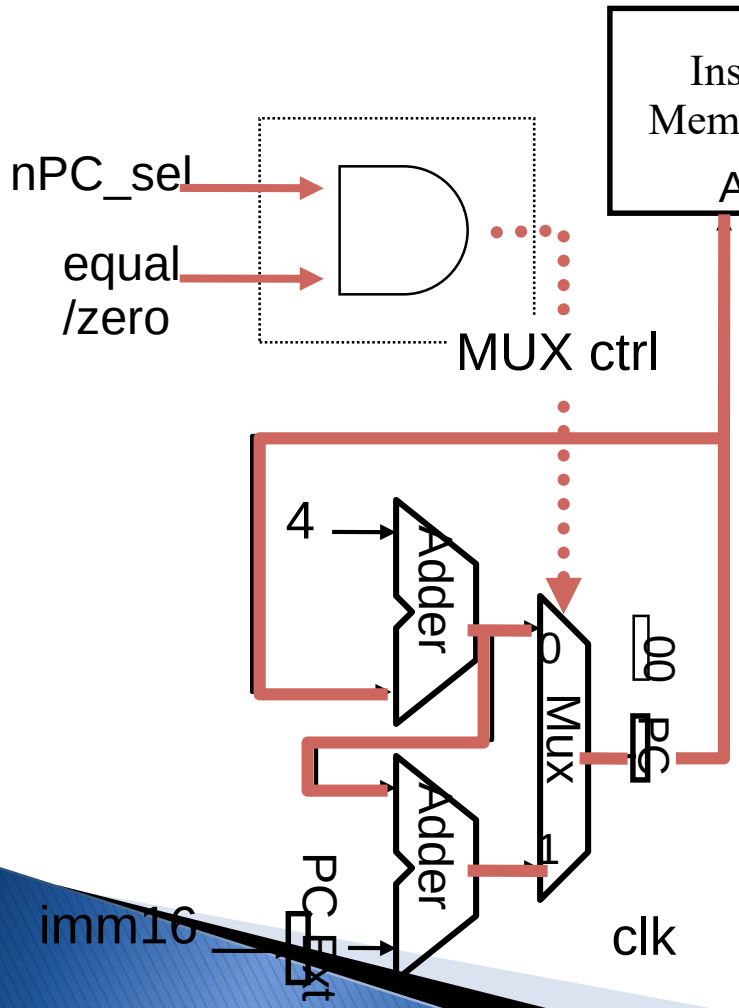
- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



Instruction Fetch Unit end of Branch



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



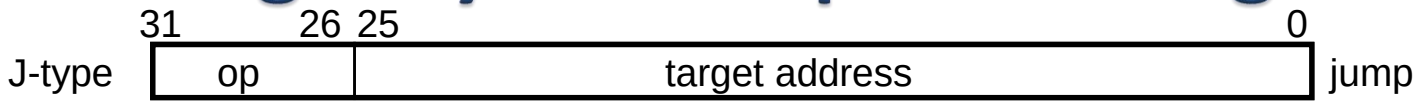
- What is encoding of nPC_sel?
  - **Direct MUX select?**
  - **Branch inst. / not branch**
- Let's pick 2nd option

is a branch statement

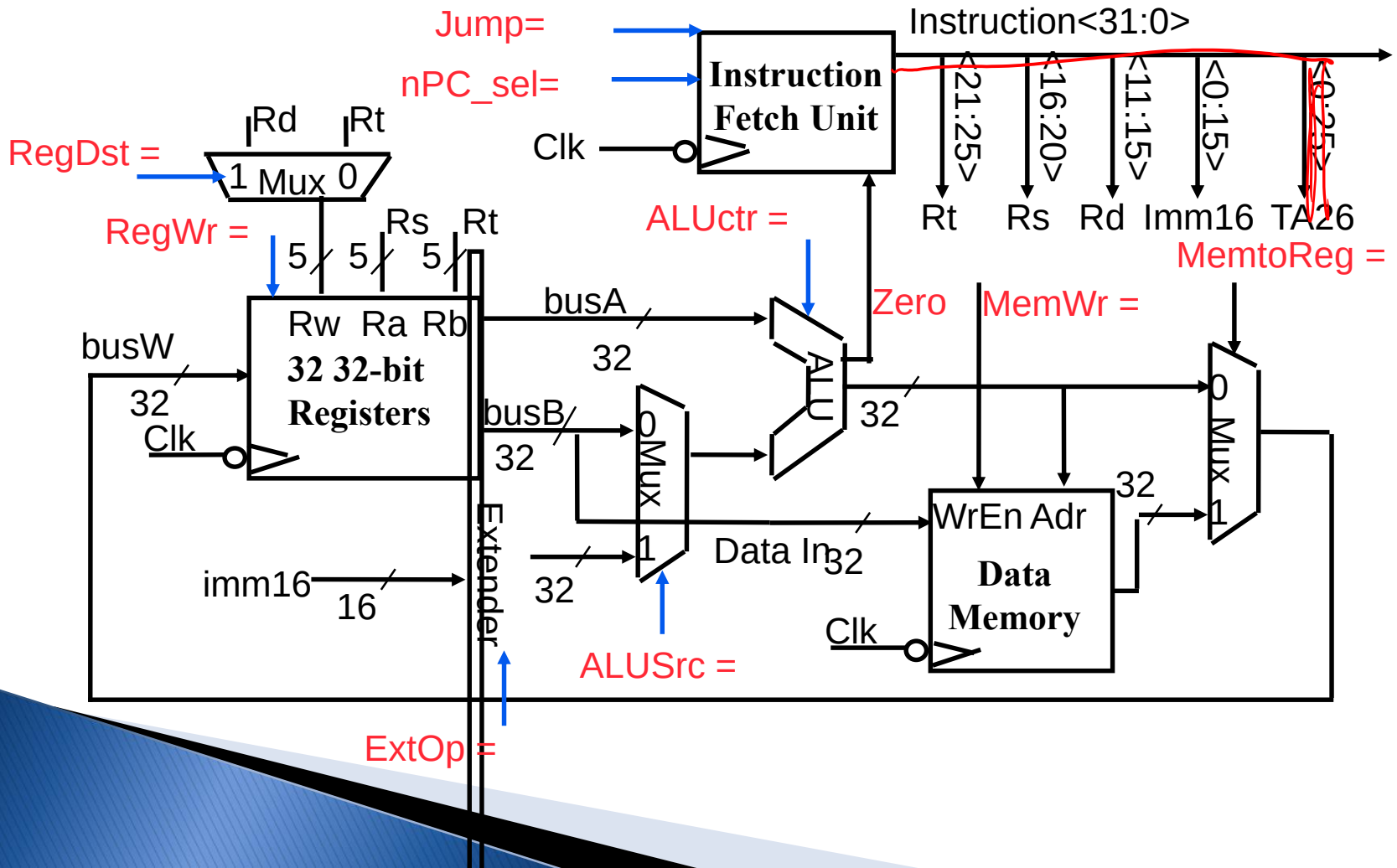
nPC_sel	zero?	MUX
0	x	0
1	0	0
1	1	1

Q: What logic gate?

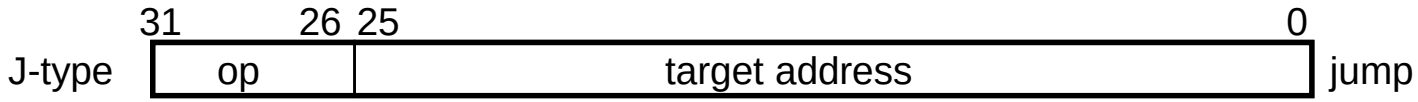
The Single Cycle Datapath during Jump



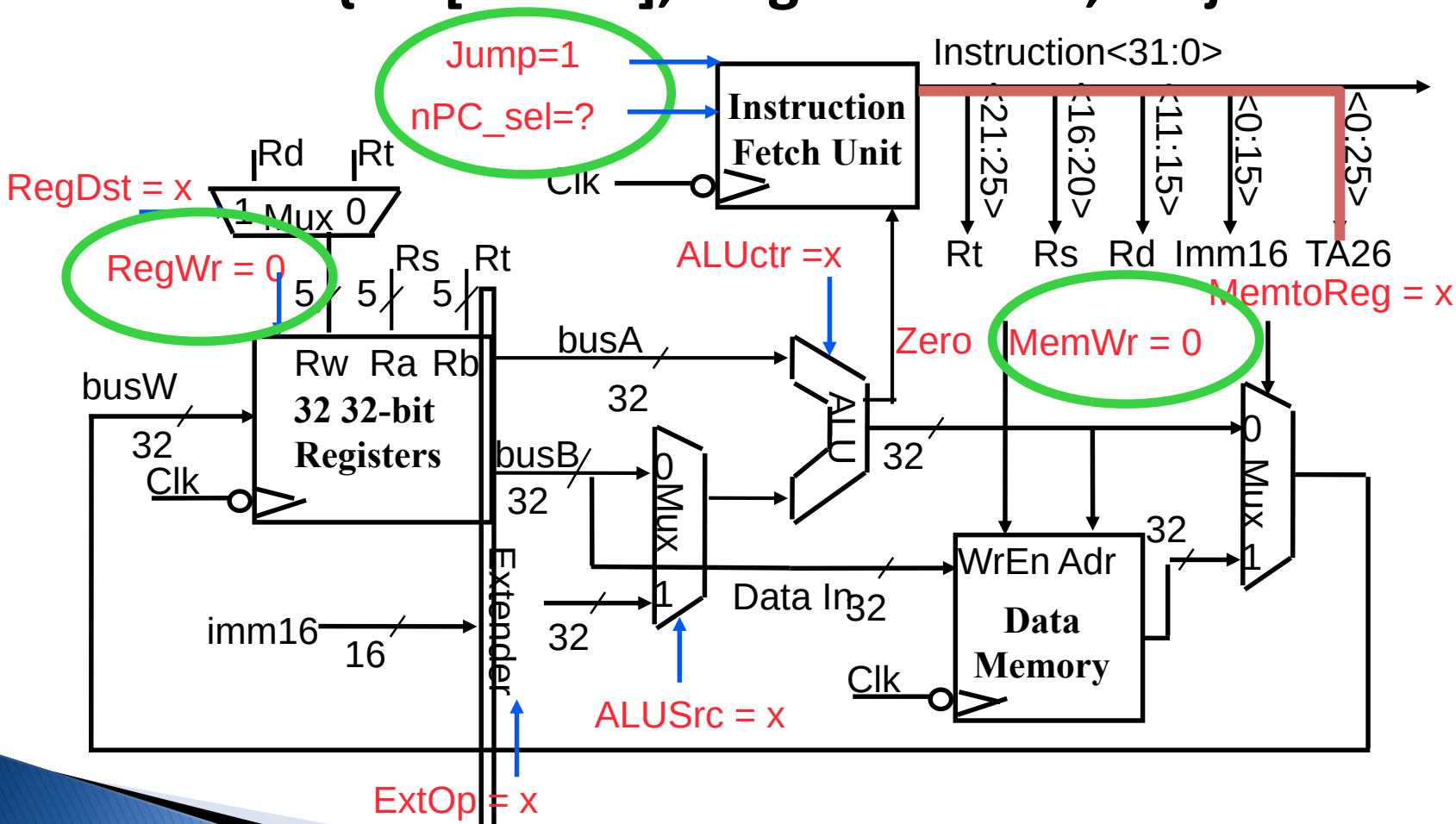
- **New PC = { PC[31..28], target address, 00 }**



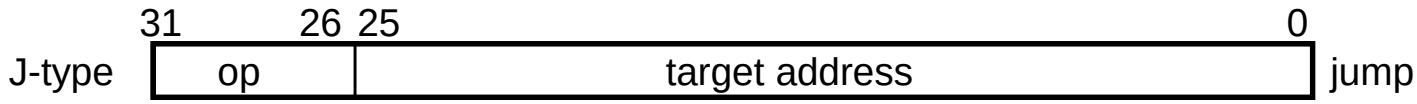
The Single Cycle Datapath during Jump



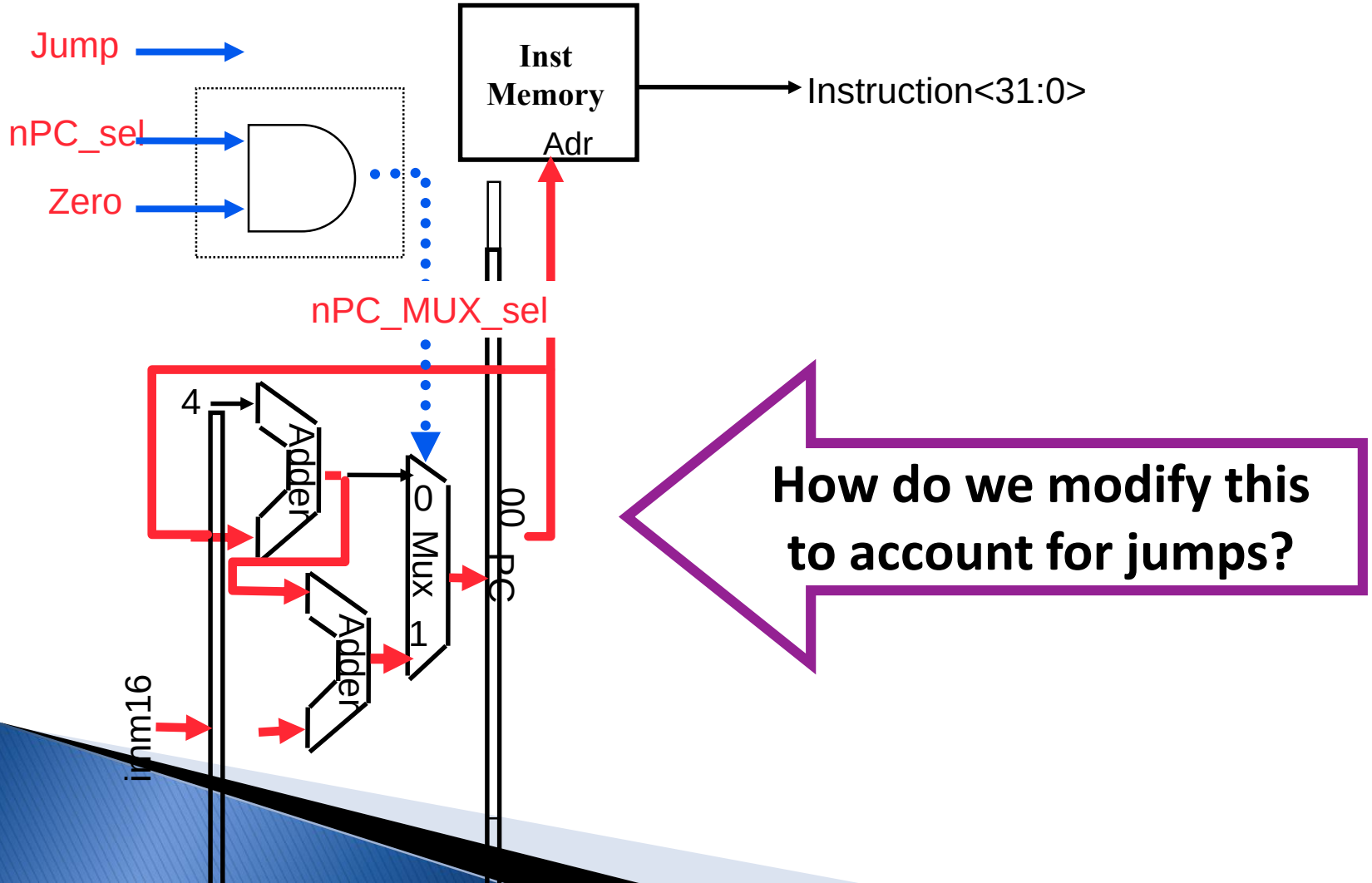
- **New PC = { PC[31..28], target address, 00 }**



Instruction Fetch Unit at the End of Jump

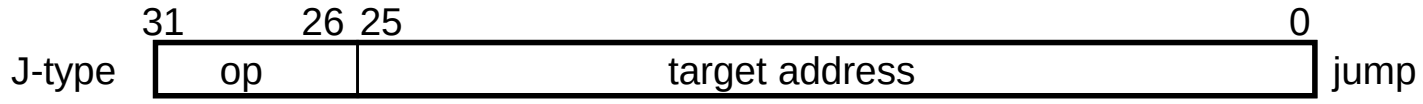


- **New PC = { PC[31..28], target address, 00 }**



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 }