# CSE 31
# Computer Organization

Lecture 26 – CPU Design (wrap up), Boolean Algebra, Gates and Circuits

# Announcements

- Labs
  - Lab 10 grace period* ends this week
    - » Demo is **NOT REQUIRED** to receive full credit
- Reading assignments
  - **All** Reading (01-08) and Homework (01-06) assignments **open for submission till 09-MAY, 11:59pm**
    - » Complete **Participation**/**Challenge** Activities in each section to receive grade
    - » IMPORTANT: Make sure to submit score to CatCourses by using the link in the assignment page
    - » You may re-do past Reading/Homework assignments to improve score.
- Final Exam
  - On 10-MAY from 8:00-11:00am at COB2 130
  - Announcement and Sample Exam posted on CatCourses

* A 10% penalty will be applied for late *submissions* during the grace period.

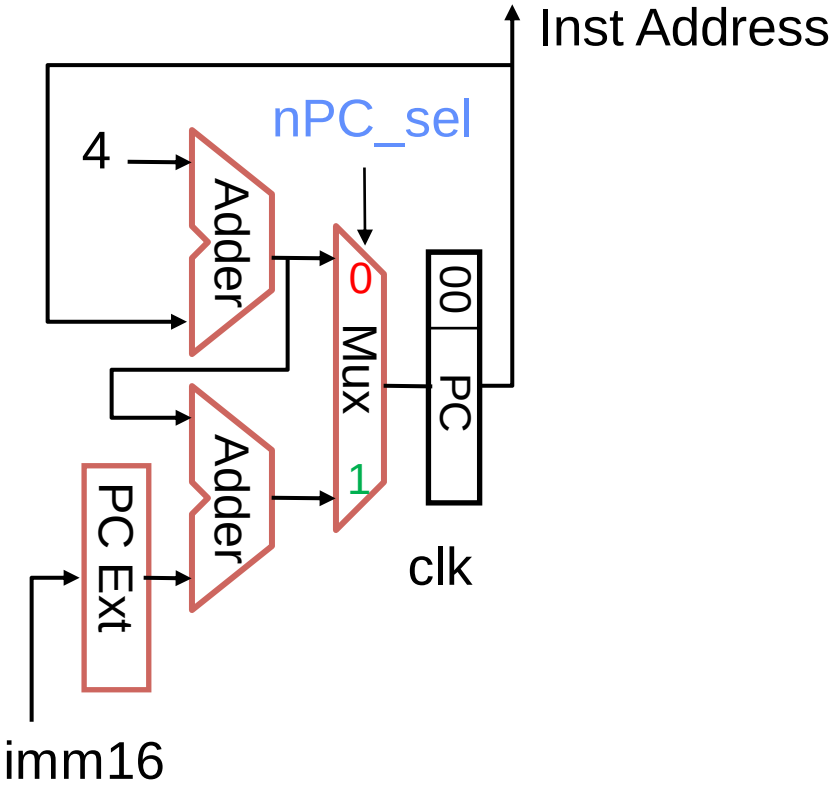
# Announcements

- Project 02
  - Due 05-MAY
  - Can work in teams of 2 students
    - » Each team member must identify teammate in “Comments…” text-box at the submission page
    - » If working in teams, each student must submit code (can be the same as teammate) and demo individually
    - » Grade can vary among teammates depending on demo
  - Demo required for project grade
    - » No partial credit for submission without demo
  - **No grace period**
    - » **Must complete submission and demo by due date.**
- **Extra Credit**
  - Up to 2% towards your overall grades
  - **Due 06-MAY, 01:59am**
  - See assignment page on CatCourses for more details
- Lab with lowest score dropped from final grade evaluation

# Meaning of the Control Signals (review)

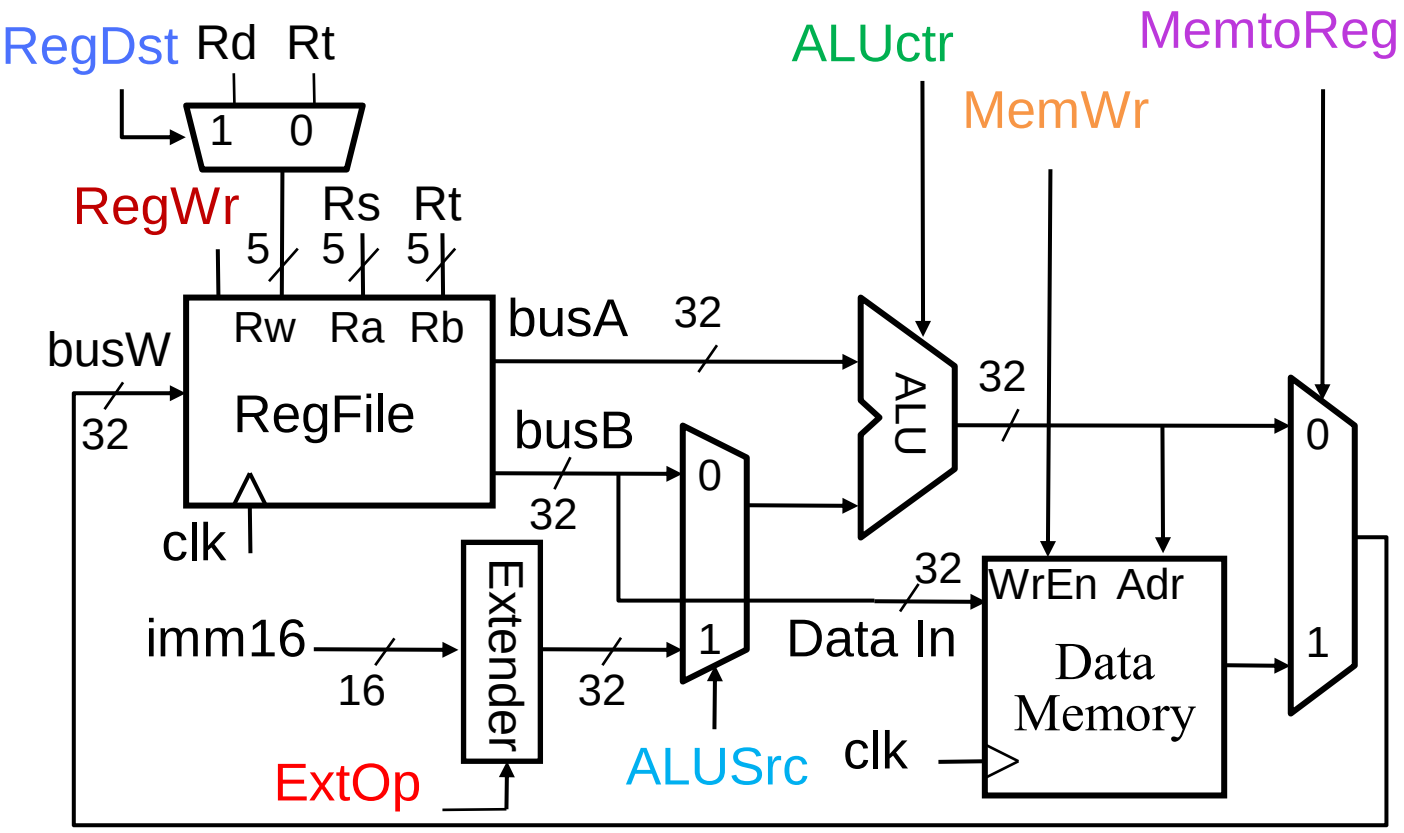
- nPC_sel: "+4": 0 ⇒ PC <– PC + 4
  "br": 1 ⇒ PC <– PC + 4 + {SignExt(Im16) , 00 }
  "n"=next

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

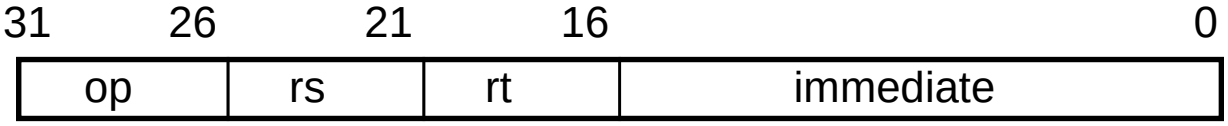


# Meaning of the Control Signals (review)

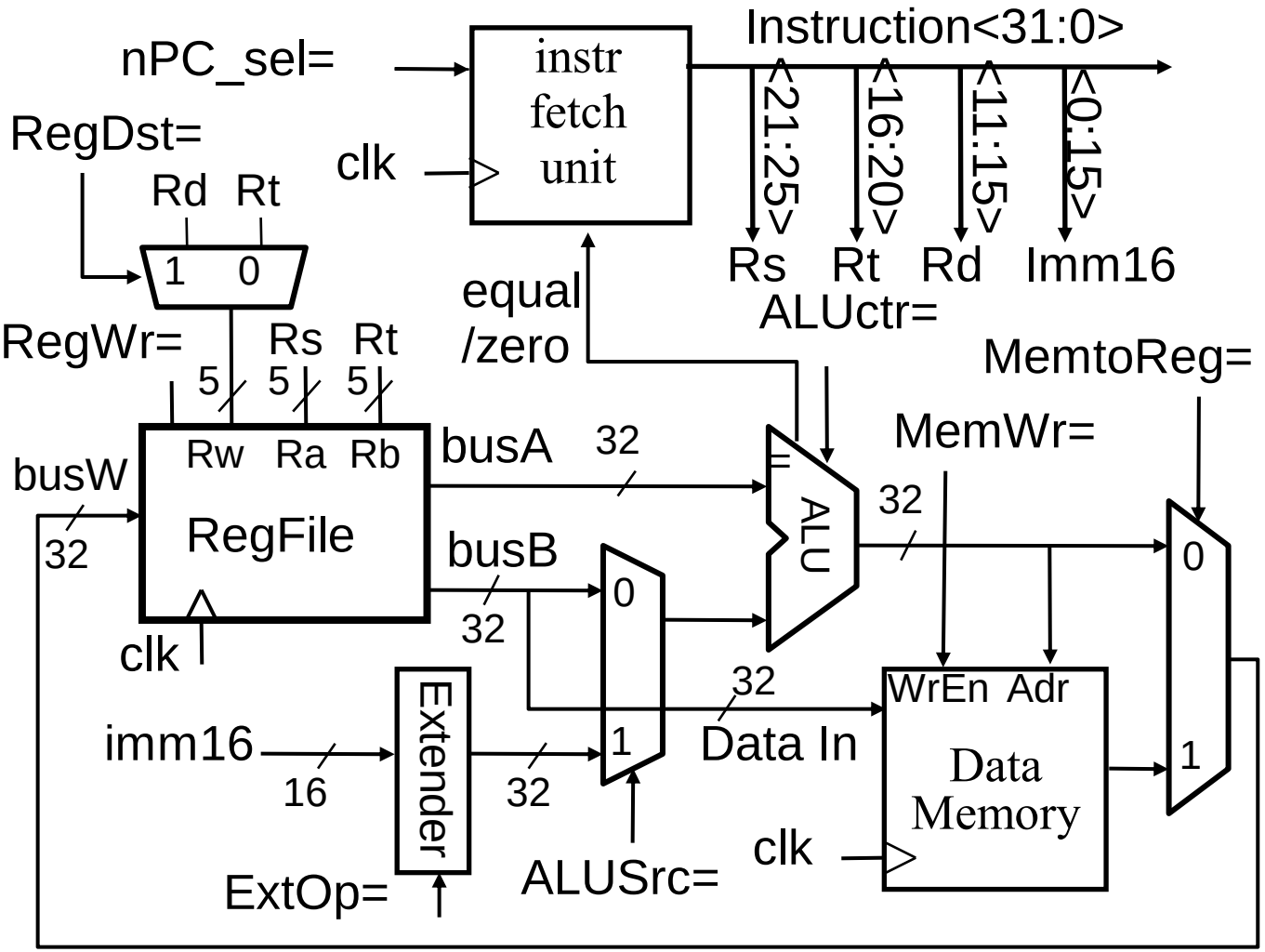
- ExtOp: 0 ⇒ zero extend
  1 ⇒ sign extend
- ALUsrc: 0 ⇒ regB;
  1 ⇒ immed
- ALUctr: "ADD", "SUB", "OR", …
- MemWr: 1 ⇒ write memory
- MemtoReg: 0 ⇒ ALU; 1 ⇒ Mem
- RegDst: 0 ⇒ "rt"; 1 ⇒ "rd"
- RegWr: 1 ⇒ write register



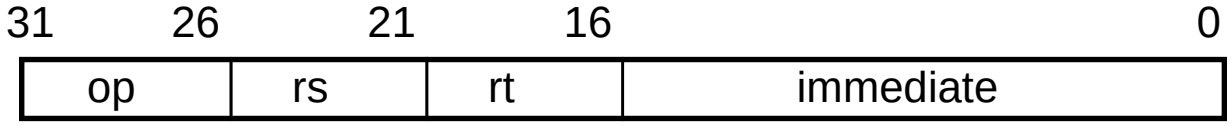
# Single Cycle Datapath for Branch



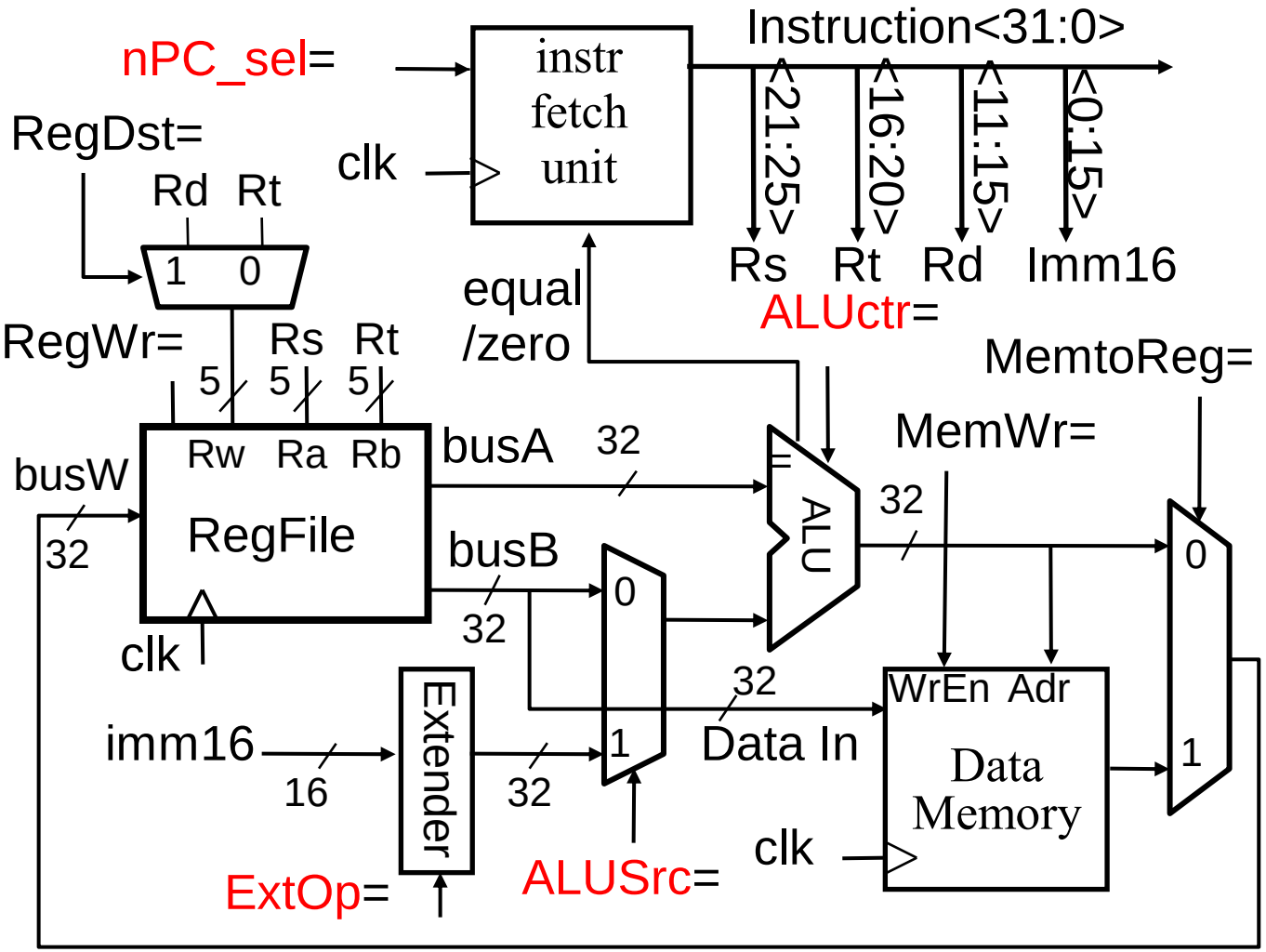

- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



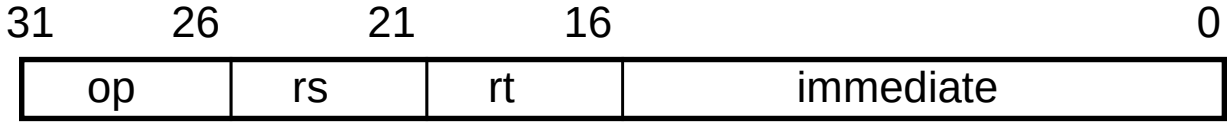
# Single Cycle Datapath for Branch



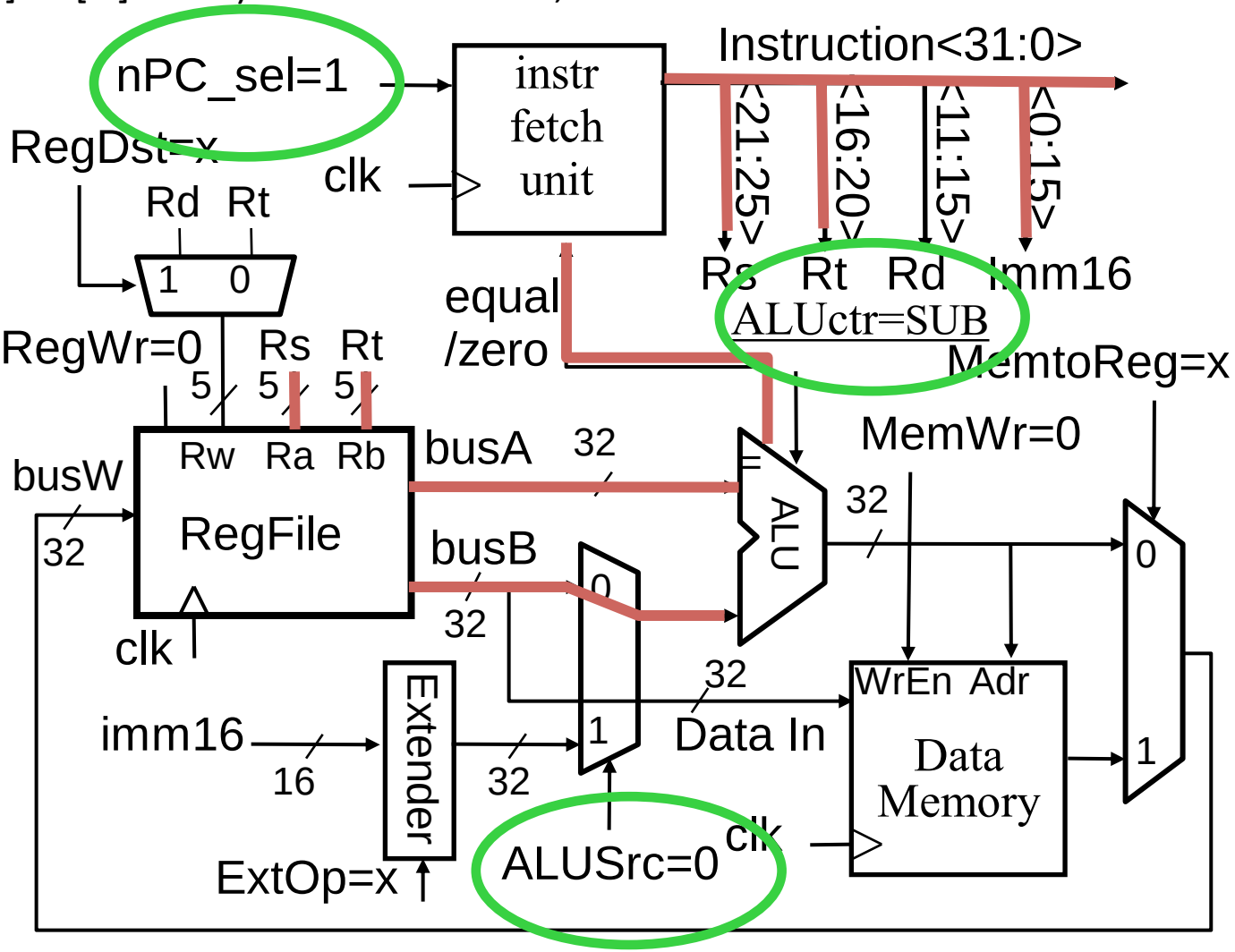

- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



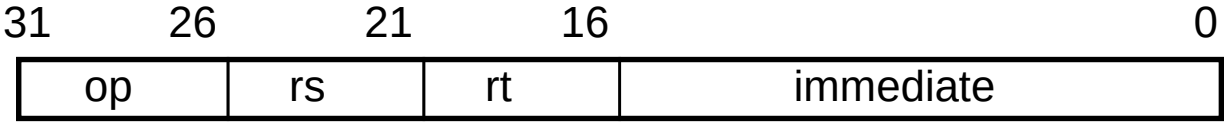
# Single Cycle Datapath for Branch



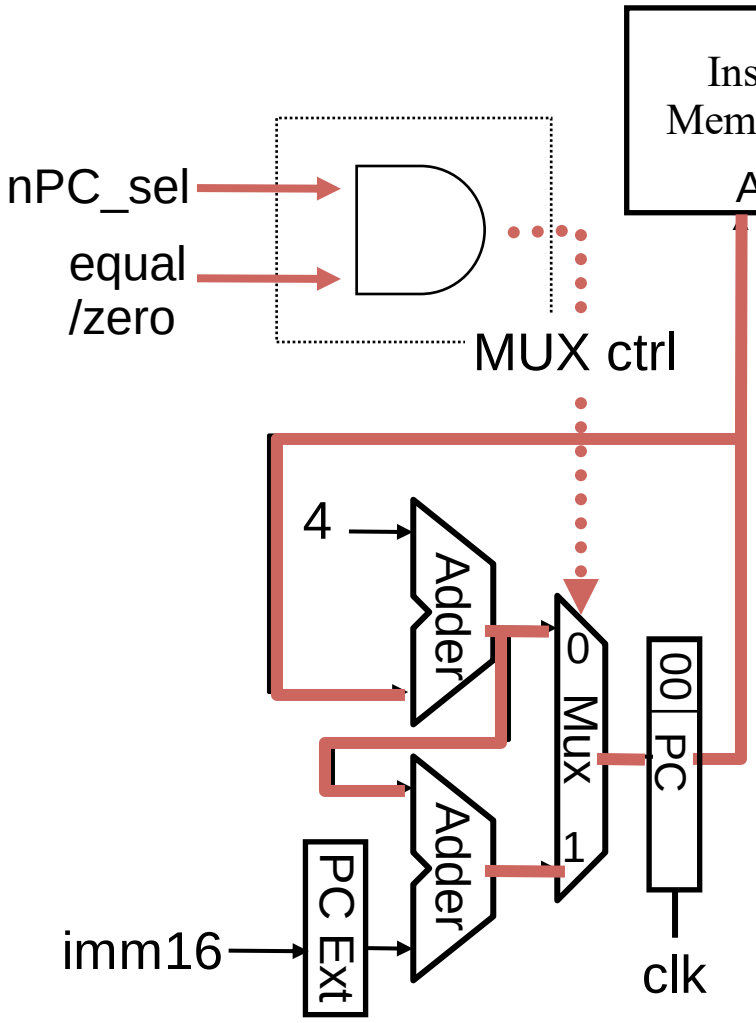

- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



# Instruction Fetch Unit end of Branch




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



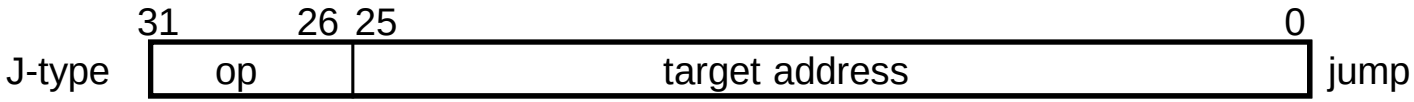

- What is encoding of nPC_sel?
  - **Direct MUX select?**
  - **Branch inst. / not branch**
- Let's pick 2nd option

Is it a branch instr?

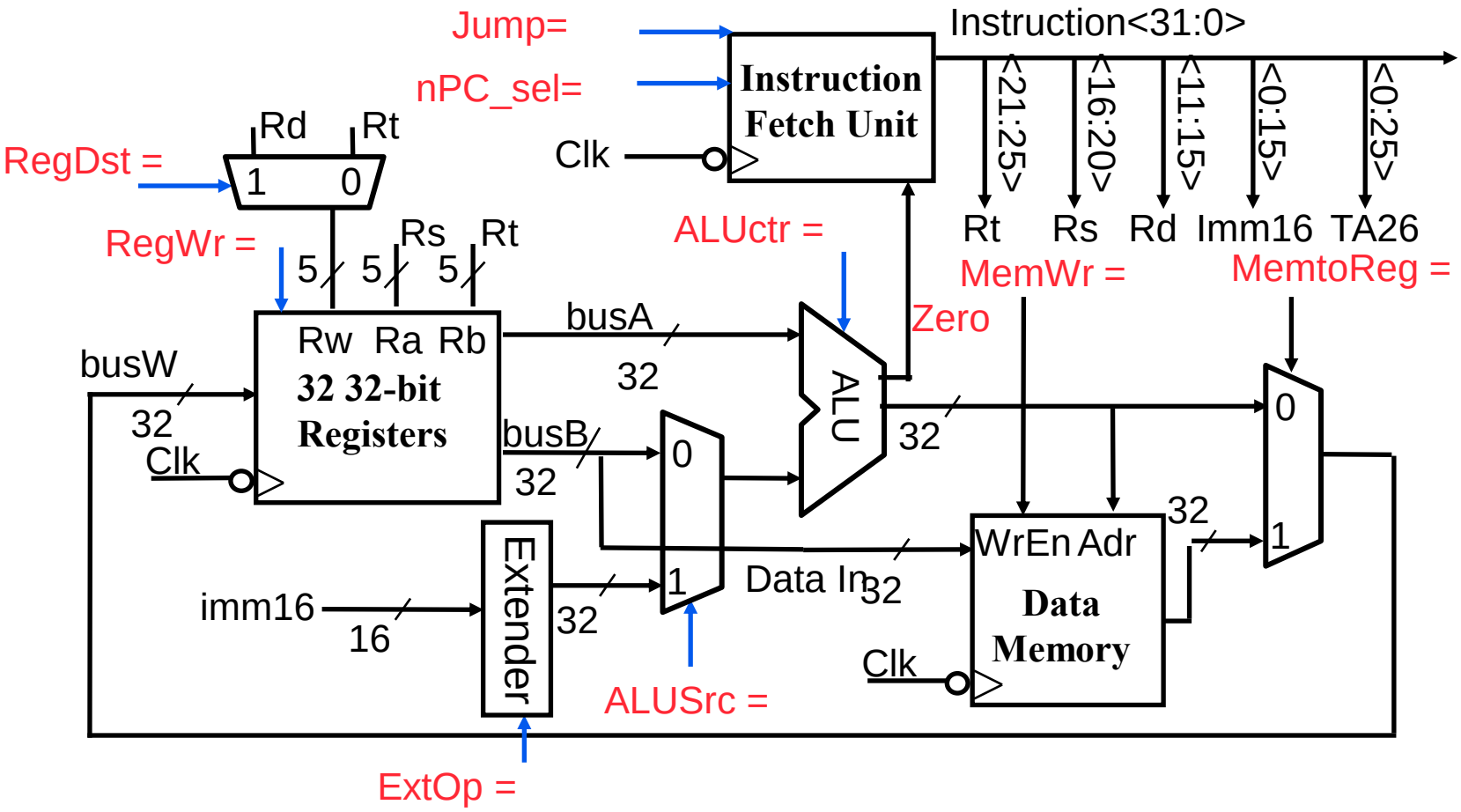
| nPC_sel | zero? | MUX |
|---|---|---|
| 0 | x | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Q: What logic gate?

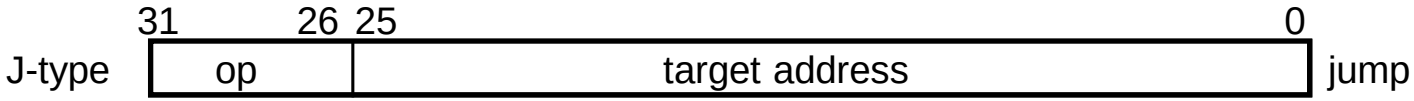
# The Single Cycle Datapath during Jump



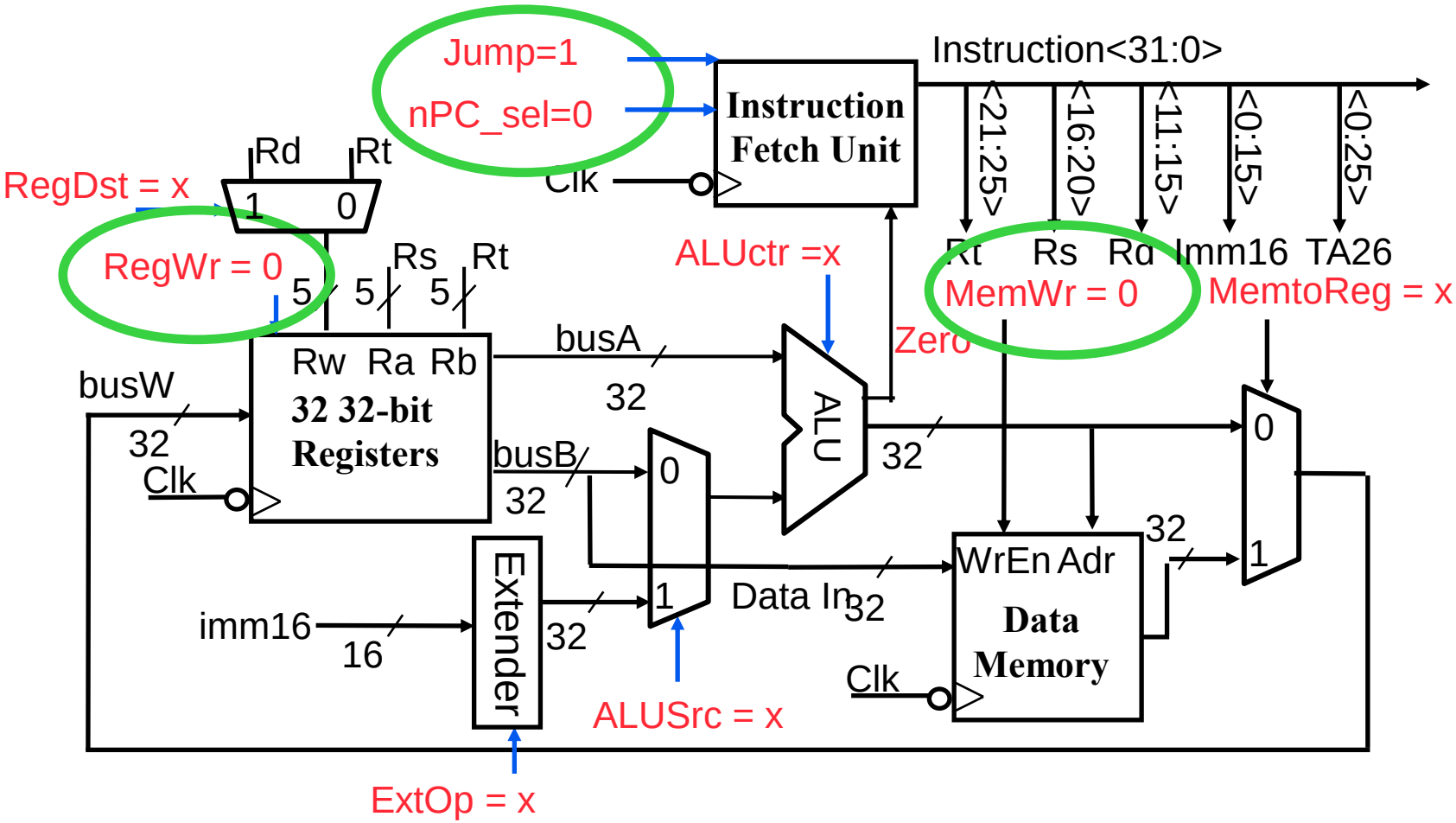

- **New PC = { PC[31..28], target address, 00 }**



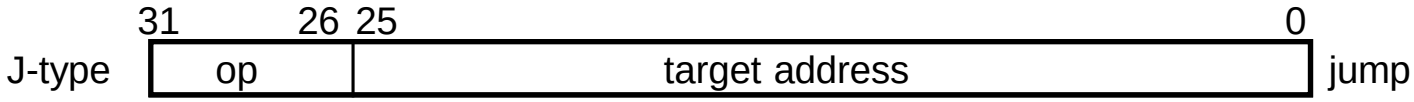
# The Single Cycle Datapath during Jump



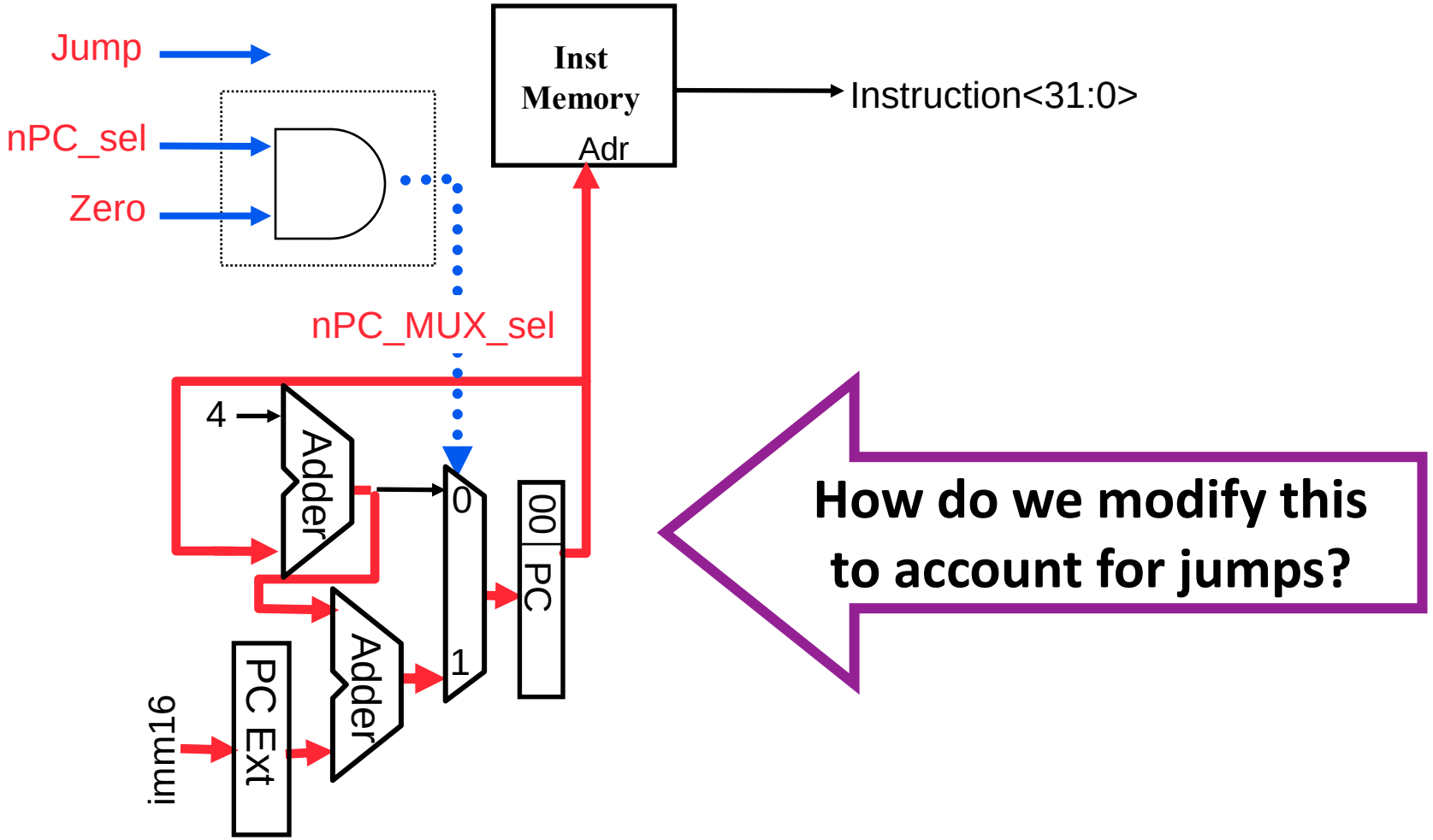

- **New PC = { PC[31..28], target address, 00 }**



# Instruction Fetch Unit at the End of Jump

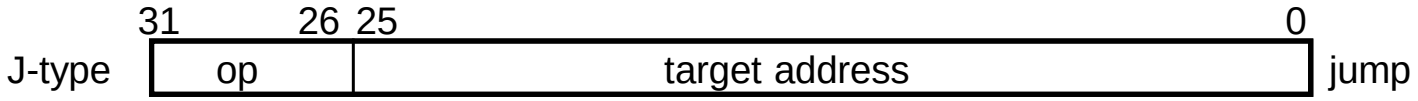



- **New PC = { PC[31..28], target address, 00 }**

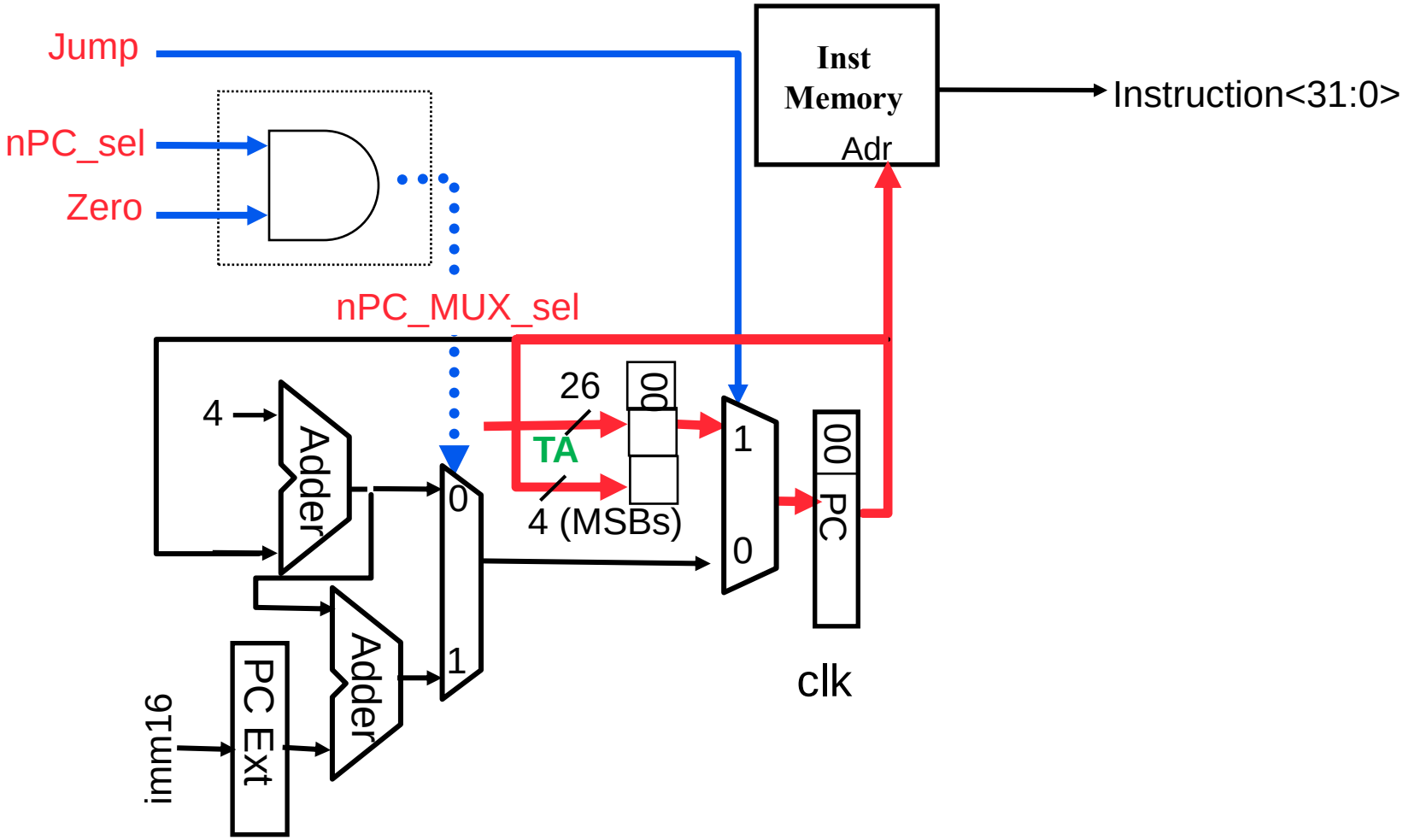



**How do we modify this to account for jumps?**

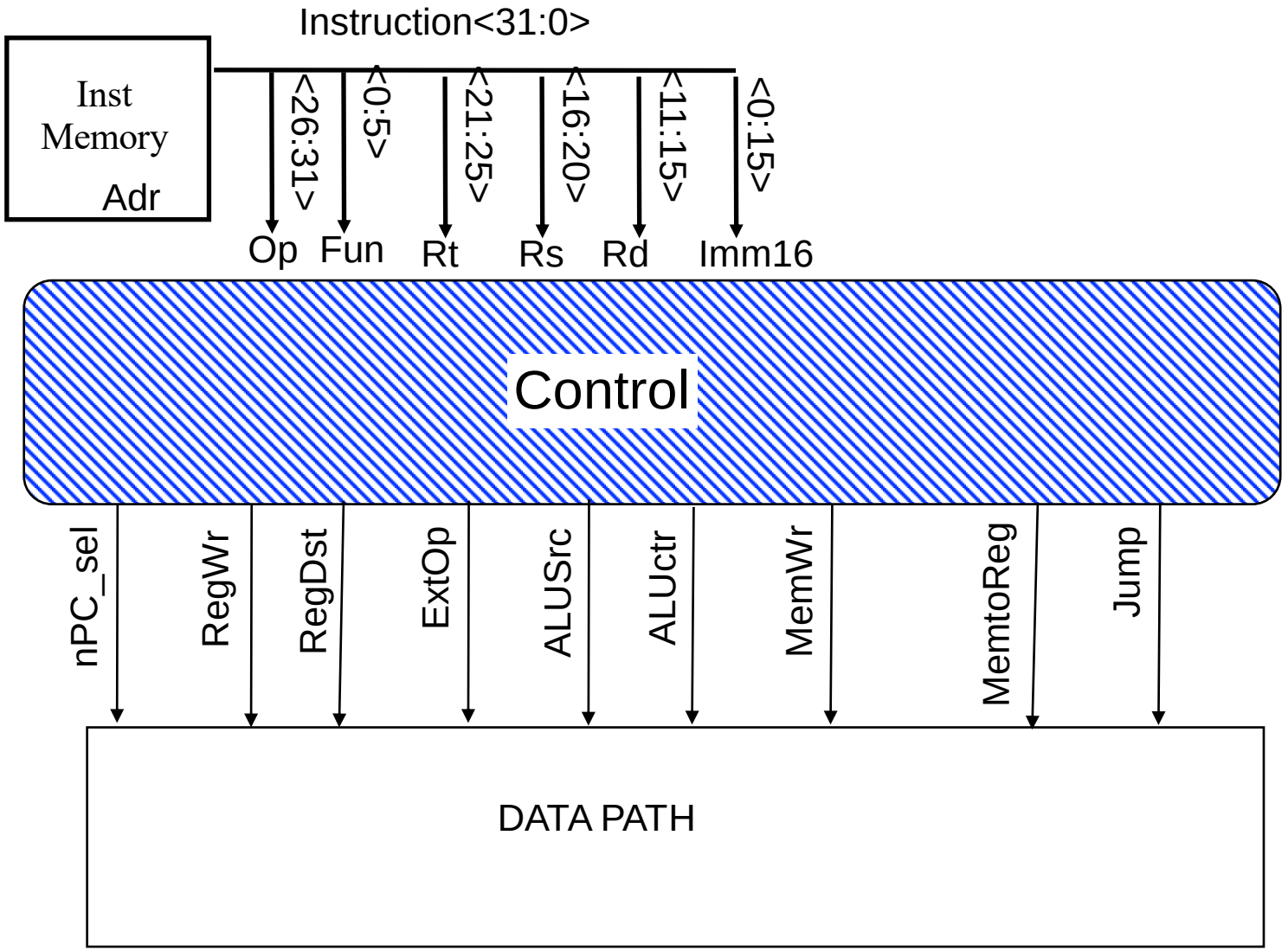
# Instruction Fetch Unit at the End of Jump




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



# Control Logic



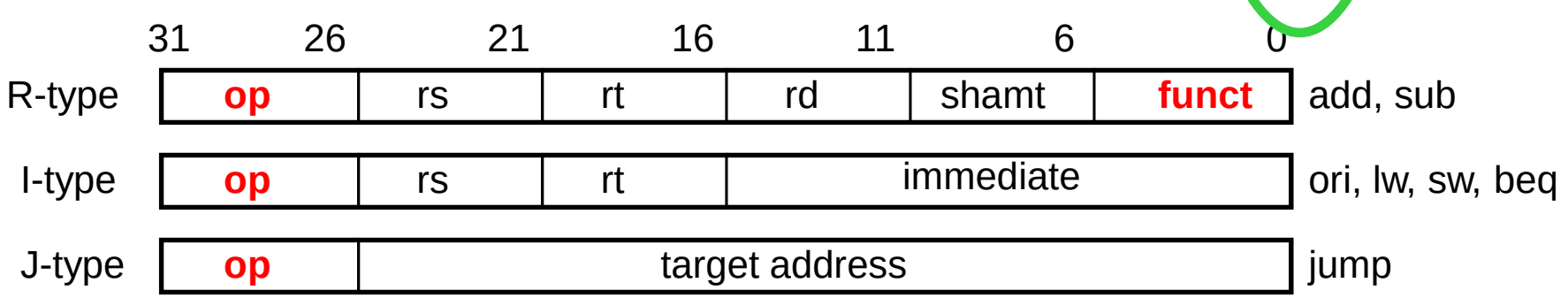
# Control Signals (1/2)

| 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" | |

# Control Signals (2/2)

See Appendix A → **func** / **op**

| | add | sub | ori | lw | sw | beq | jump |
|---|---|---|---|---|---|---|---|
| **func** | 10 0000 | 10 0010 | We Don't Care :-) | | | | |
| **op** | 00 0000 | 00 0000 | 00 1101 | 10 0011 | 10 1011 | 00 0100 | 00 0010 |
| 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 00 ADD, 01: SUB, 10: OR)
ALUctr[1] = or

+: OR, •: AND, ~: NOT

rtype = 1 when opcode is 000000

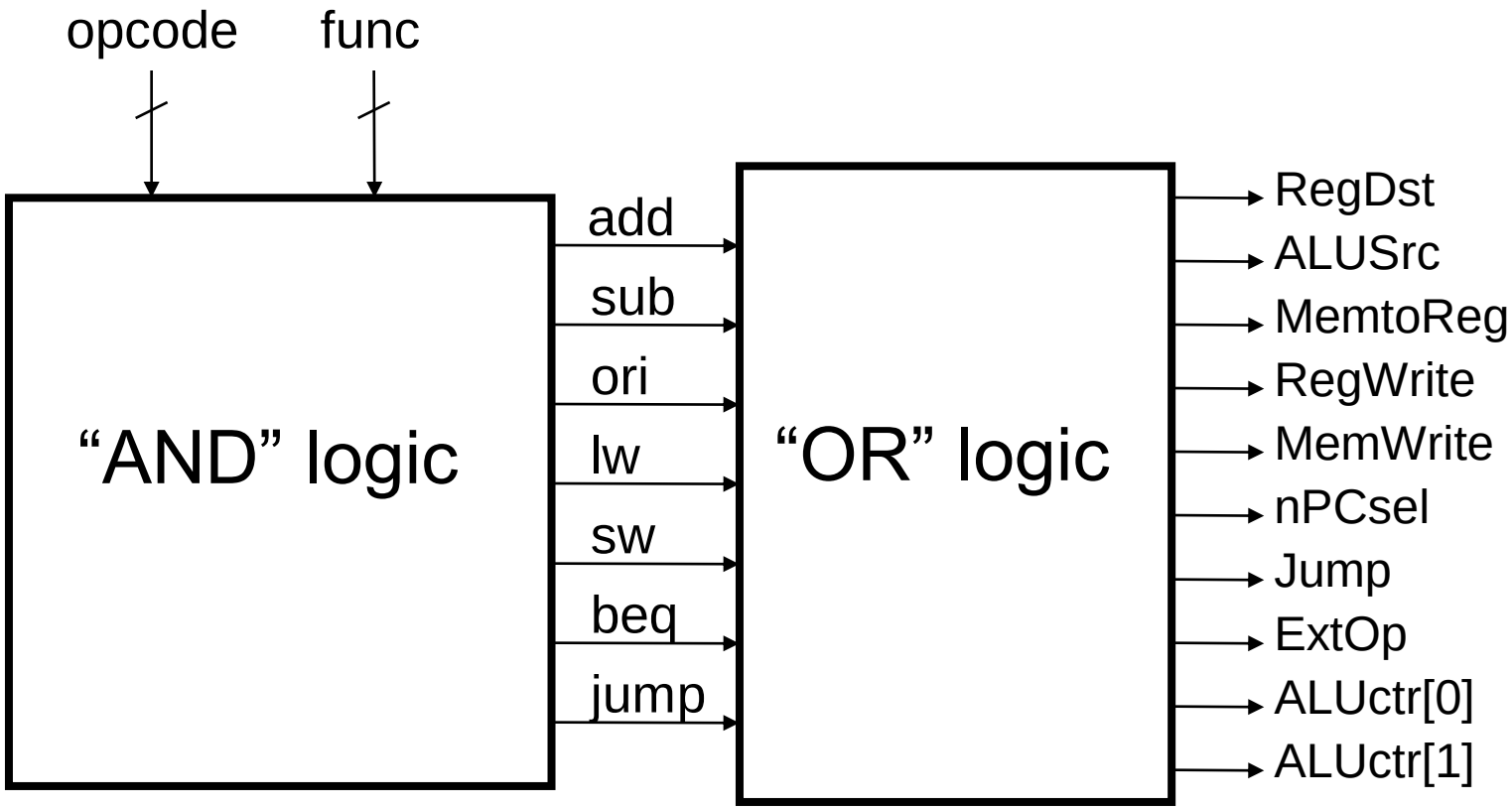
*where,*

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

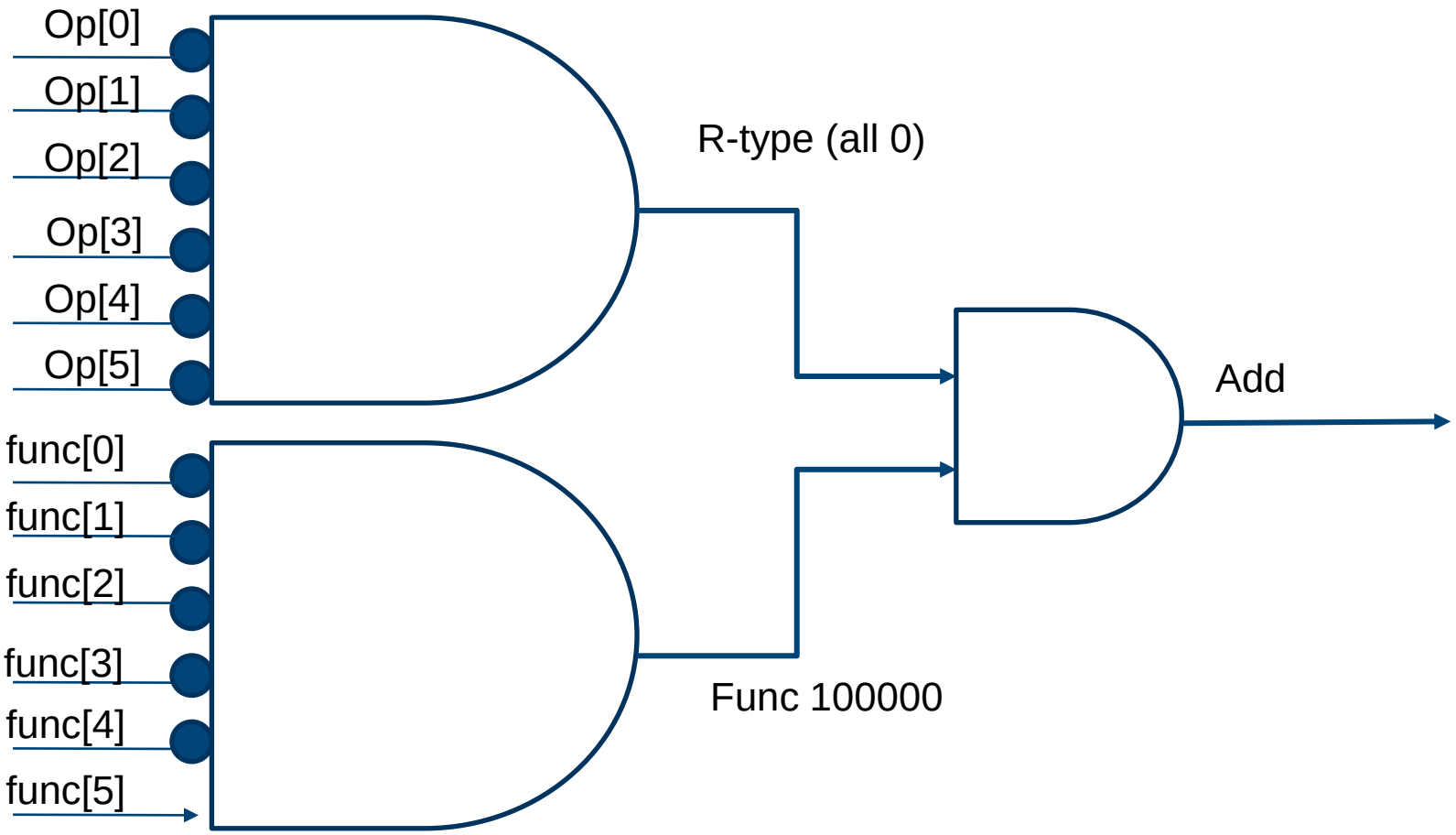
How do we implement this in gates?

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

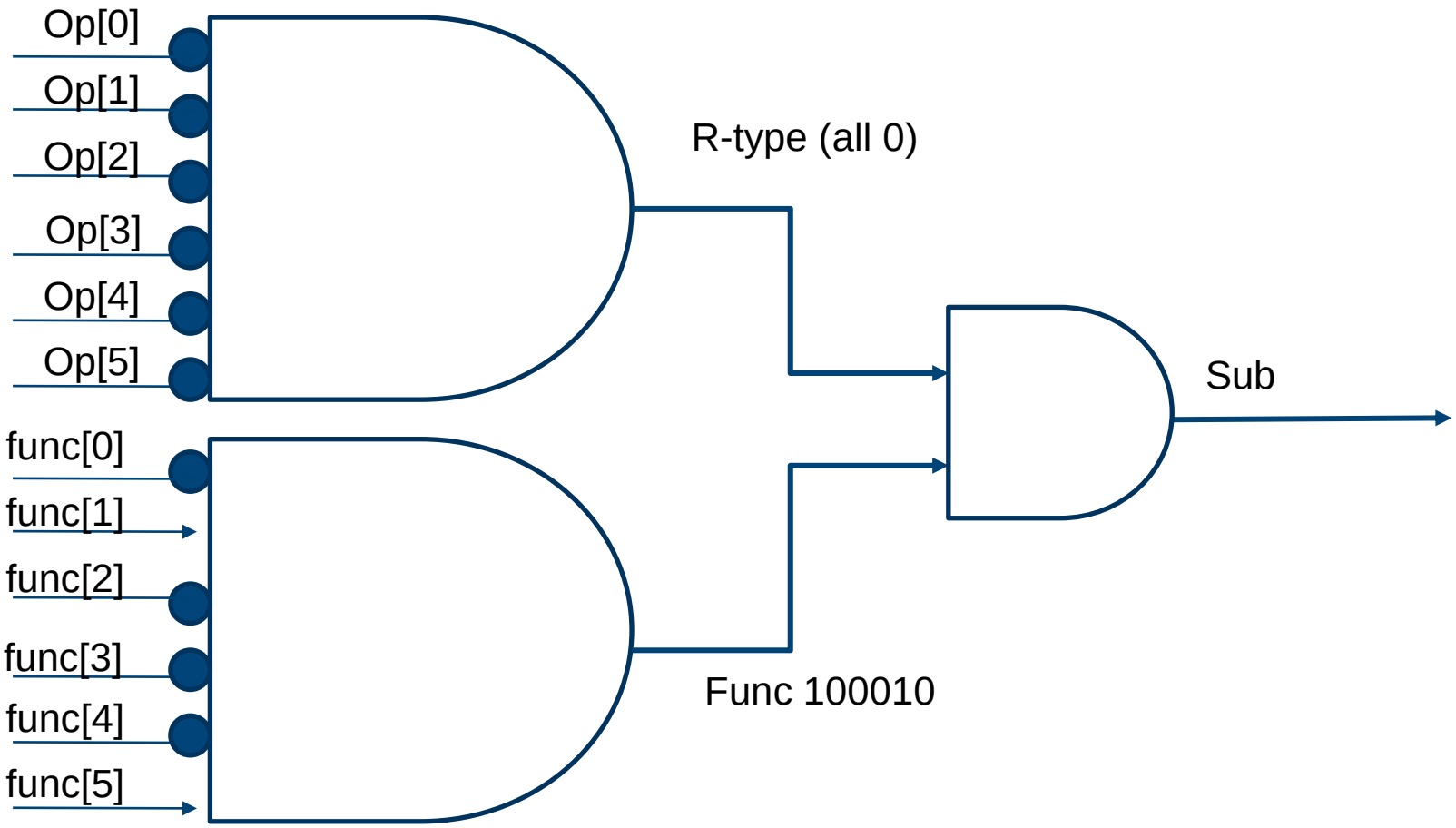
# Controller Implementation



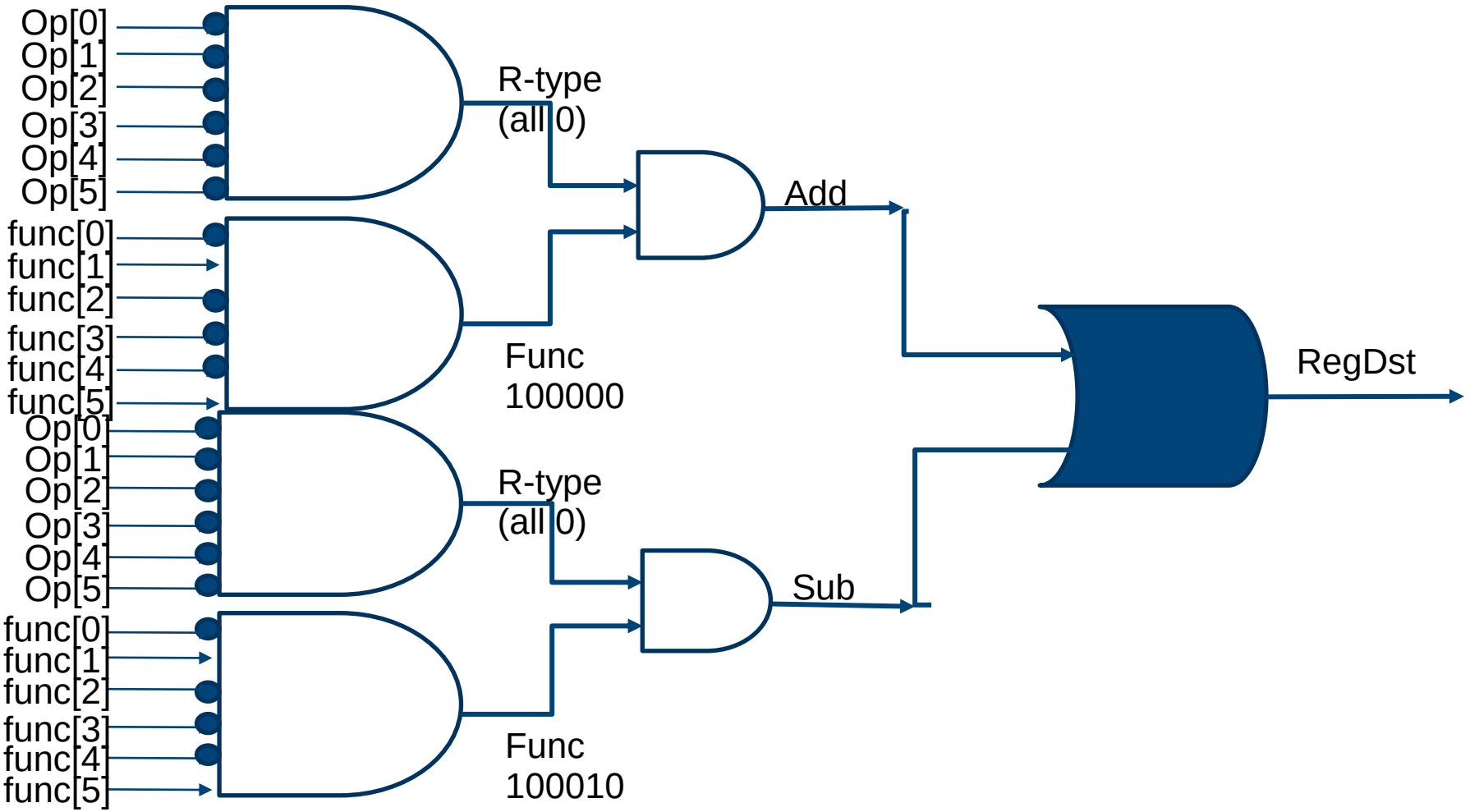
# Add (opcode = 00 0000, func = 10 0000)



# Sub (opcode = 00 0000, func = 10 0010)



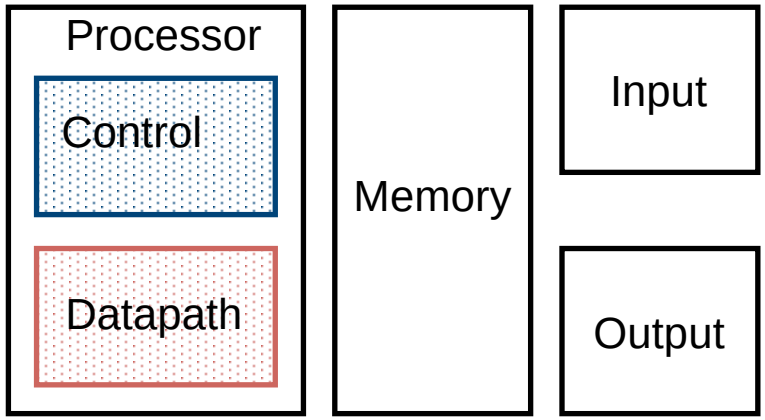
# RegDst


Op[0]
Op[1]
Op[2]
Op[3]
Op[4]
Op[5]
R-type
(all 0)
func[0]
func[1]
func[2]
func[3]
func[4]
func[5]
Add
Func
100000
RegDst
Op[0]
Op[1]
Op[2]
Op[3]
Op[4]
Op[5]
R-type
(all 0)
Sub
func[0]
func[1]
func[2]
func[3]
func[4]
func[5]
Func
100010

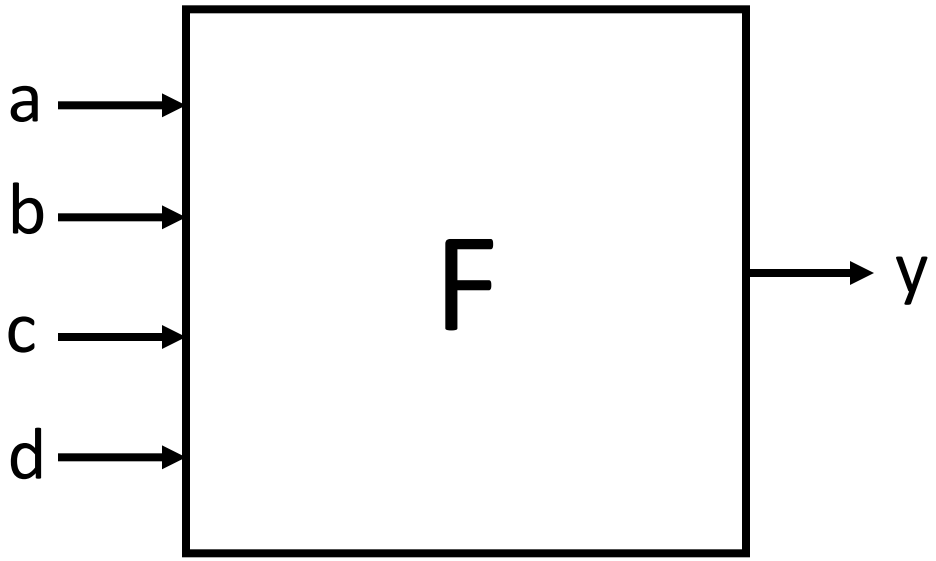
# Summary: Single-cycle Processor

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



# Truth Tables

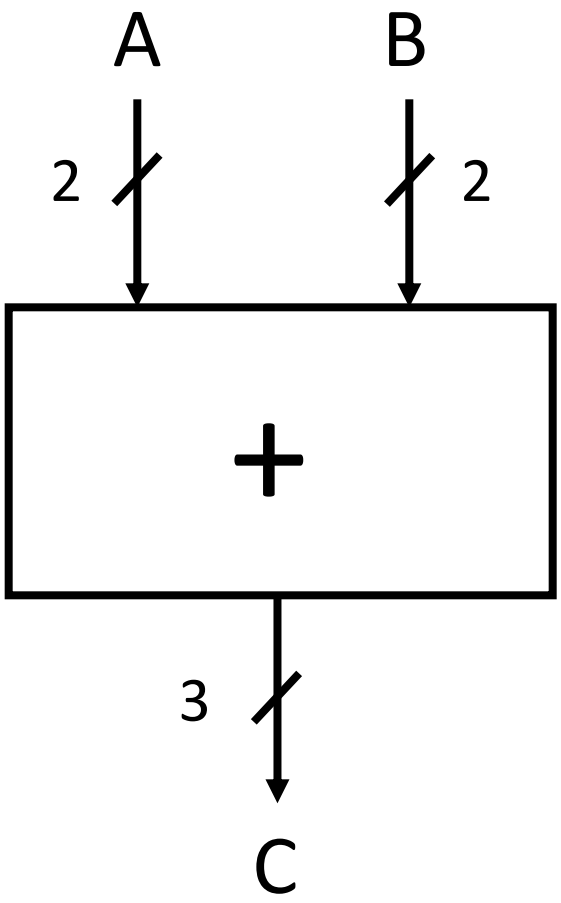



| a | b | c | d | y |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | F(0,0,0,0) |
| 0 | 0 | 0 | 1 | F(0,0,0,1) |
| 0 | 0 | 1 | 0 | F(0,0,1,0) |
| 0 | 0 | 1 | 1 | F(0,0,1,1) |
| 0 | 1 | 0 | 0 | F(0,1,0,0) |
| 0 | 1 | 0 | 1 | F(0,1,0,1) |
| 0 | 1 | 1 | 0 | F(0,1,1,0) |
| 0 | 1 | 1 | 1 | F(0,1,1,1) |
| 1 | 0 | 0 | 0 | F(1,0,0,0) |
| 1 | 0 | 0 | 1 | F(1,0,0,1) |
| 1 | 0 | 1 | 0 | F(1,0,1,0) |
| 1 | 0 | 1 | 1 | F(1,0,1,1) |
| 1 | 1 | 0 | 0 | F(1,1,0,0) |
| 1 | 1 | 0 | 1 | F(1,1,0,1) |
| 1 | 1 | 1 | 0 | F(1,1,1,0) |
| 1 | 1 | 1 | 1 | F(1,1,1,1) |

# TT #1: XOR, 1 iff a or b = 1 (not both)

| a | b | y |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# TT #2: 2-bit adder




| A $a_1a_0$ | B $b_1b_0$ | C $c_2c_1c_0$ |
|---|---|---|
| 00 | 00 | 000 |
| 00 | 01 | 001 |
| 00 | 10 | 010 |
| 00 | 11 | 011 |
| 01 | 00 | 001 |
| 01 | 01 | 010 |
| 01 | 10 | 011 |
| 01 | 11 | 100 |
| 10 | 00 | 010 |
| 10 | 01 | 011 |
| 10 | 10 | 100 |
| 10 | 11 | 101 |
| 11 | 00 | 011 |
| 11 | 01 | 100 |
| 11 | 10 | 101 |
| 11 | 11 | 110 |

How Many Rows?

# TT #3: 32-bit unsigned adder

| A | B | C |
|---|---|---|
| 000 ... 0 | 000 ... 0 | 000 ... 00 |
| 000 ... 0 | 000 ... 1 | 000 ... 01 |
| . | . | . |
| . | . | . |
| . | . | . |
| 111 ... 1 | 111 ... 1 | 111 ... 10 |

How Many Rows?

# TT #4: 3-input majority circuit

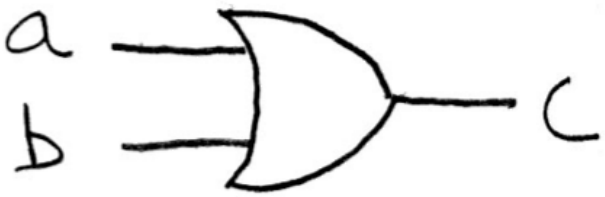
| a | b | c | y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

# Logic Gates (1/2)

AND

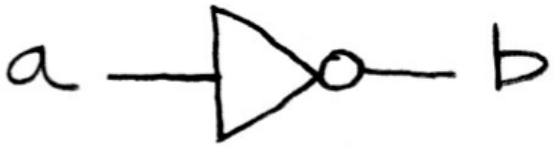

| ab | c |
|---|---|
| 00 | 0 |
| 01 | 0 |
| 10 | 0 |
| 11 | 1 |

OR




| ab | c |
|---|---|
| 00 | 0 |
| 01 | 1 |
| 10 | 1 |
| 11 | 1 |

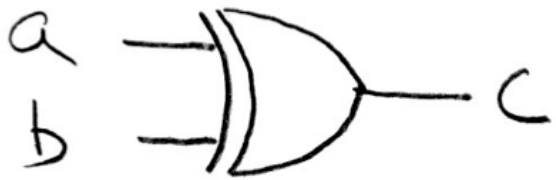
NOT




| a | b |
|---|---|
| 0 | 1 |
| 1 | 0 |

# Logic Gates (2/2)

XOR

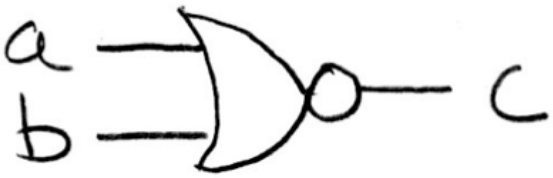



| ab | c |
|---|---|
| 00 | 0 |
| 01 | 1 |
| 10 | 1 |
| 11 | 0 |

NAND


| ab | c |
|---|---|
| 00 | 1 |
| 01 | 1 |
| 10 | 1 |
| 11 | 0 |

NOR




| ab | c |
|---|---|
| 00 | 1 |
| 01 | 0 |
| 10 | 0 |
| 11 | 0 |

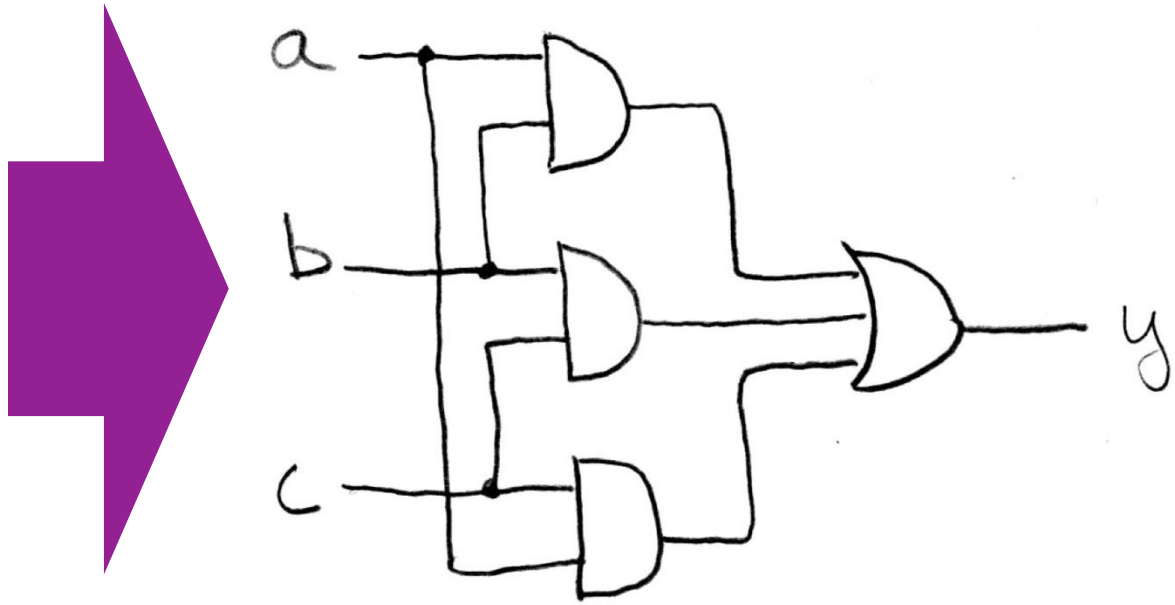
# 2-input gates extend to n-inputs

- N-input XOR is the only one which isn't so obvious
- It's simple: XOR is a 1 iff the # of 1s at its input is odd

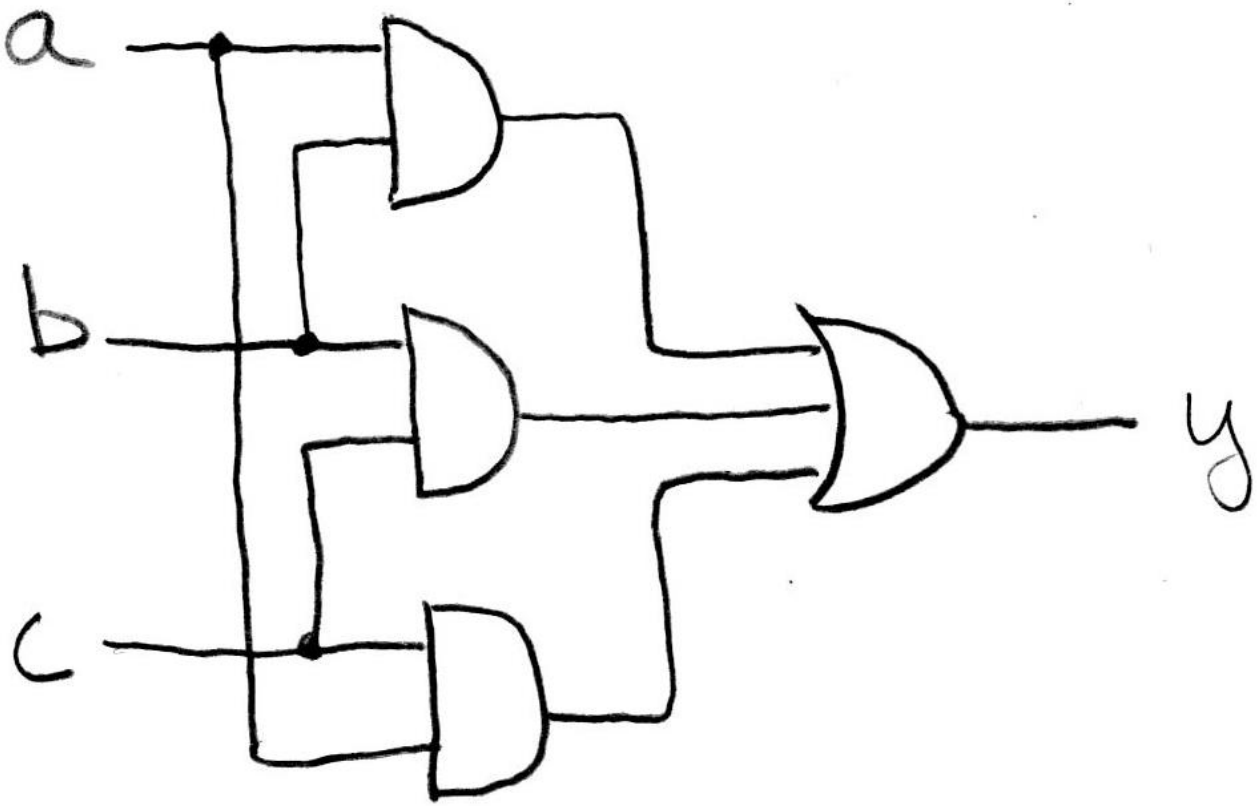
| a | b | c | y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

# 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 |



# Boolean Algebra (e.g., for majority fun.)




$y = a \bullet b + a \bullet c + b \bullet c$

$y = ab + ac + bc$

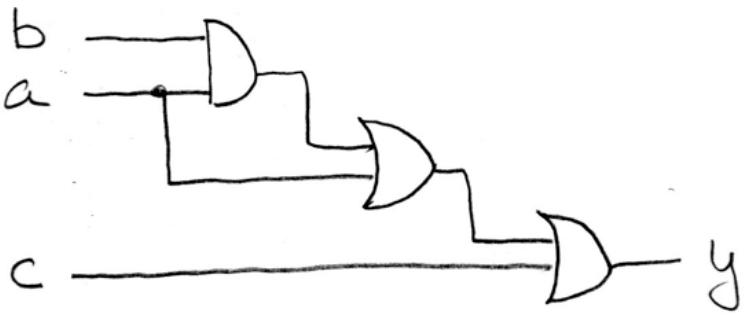
# Laws of Boolean Algebra

| | | |
|---|---|---|
| $x \cdot \overline{x} = 0$ | $x + \overline{x} = 1$ | complementarity |
| $x \cdot 0 = 0$ | $x + 1 = 1$ | laws of 0's and 1's |
| $x \cdot 1 = x$ | $x + 0 = x$ | identities |
| $x \cdot x = x$ | $x + x = x$ | idempotent law |
| $x \cdot y = y \cdot x$ | $x + y = y + x$ | commutativity |
| $(xy)z = x(yz)$ | $(x + y) + z = x + (y + z)$ | associativity |
| $x(y + z) = xy + xz$ | $x + yz = (x + y)(x + z)$ | distribution |
| $xy + x = x$ | $(x + y)x = x$ | uniting theorem |
| $\overline{x}y + x = x + y$ | $(\overline{x} + y)x = xy$ | uniting theorem v.2 |
| $\overline{x \cdot y} = \overline{x} + \overline{y}$ | $\overline{x + y} = \overline{x} \cdot \overline{y}$ | DeMorgan's Law |

# Boolean Algebraic Simplification

$$
\begin{aligned}
y &= ab + a + c \\
&= a(b + 1) + c && \textit{distribution, identity} \\
&= a(1) + c && \textit{law of 1's} \\
&= a + c && \textit{identity}
\end{aligned}
$$

# Circuit & Algebraic Simplification




original circuit

$$\downarrow$$

$$y = ((ab) + a) + c$$

equation derived from original circuit

$$\downarrow$$

$$\begin{aligned} &= ab + a + c \\ &= a(b + 1) + c \\ &= a(1) + c \\ &= a + c \end{aligned}$$

algebraic simplification

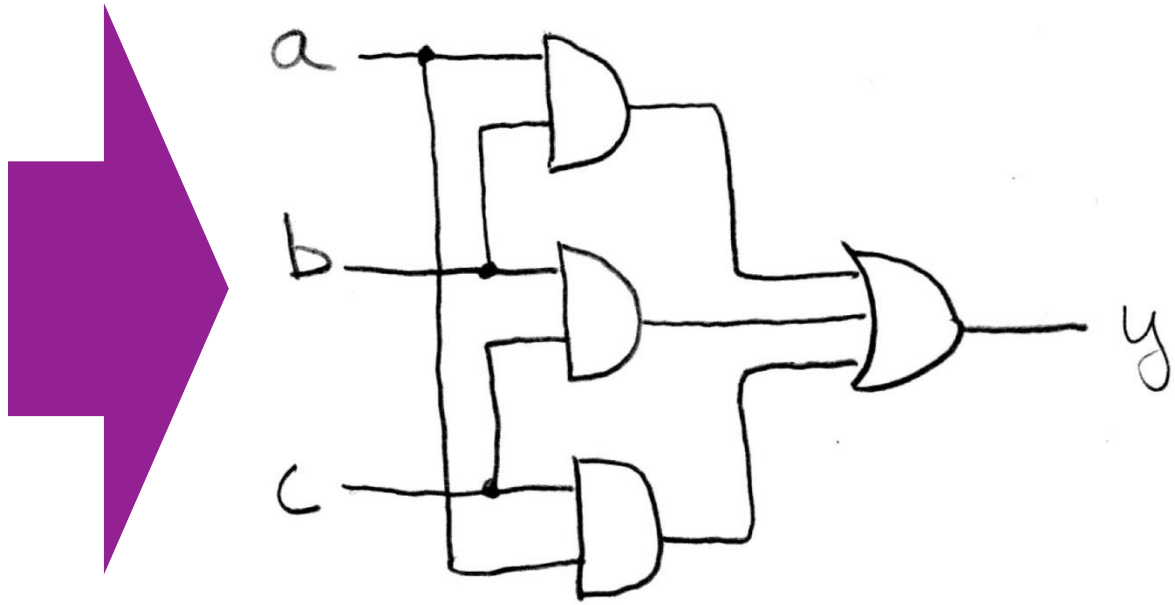
$$\downarrow$$


simplified circuit

BA also great for circuit verification
Circ X = Circ Y?
use BA to prove!

# 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)

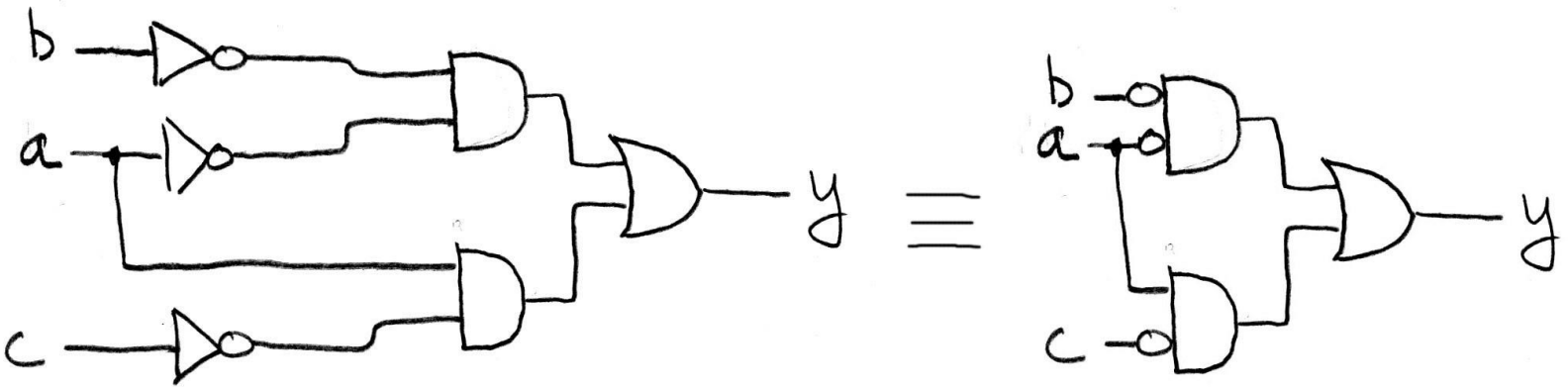
| | $abc$ | $y$ |
|---|---|---|
| $\overline{a} \cdot \overline{b} \cdot \overline{c}$ | 000 | 1 |
| $\overline{a} \cdot \overline{b} \cdot c$ | 001 | 1 |
| | 010 | 0 |
| | 011 | 0 |
| $a \cdot \overline{b} \cdot \overline{c}$ | 100 | 1 |
| | 101 | 0 |
| $a \cdot b \cdot \overline{c}$ | 110 | 1 |
| | 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 or
Sum-of-minterms
(ORs of ANDs)

# Canonical forms (2/2)

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



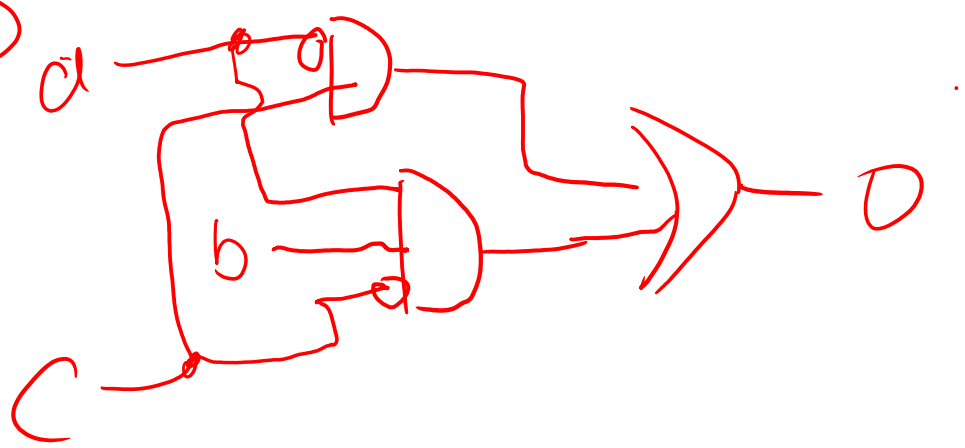
# Canonical forms example

| 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 |

$O = \bar{a}\cdot\bar{b}\cdot c + \bar{a}\cdot b\cdot c + a\cdot b\cdot\bar{c}$

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

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



**And that wraps it up!**

**Thank you all for an excellent semester.**

**All the best on the Finals.**