CSE 31 Computer Organization

Lecture 23 – CPU Design (3)

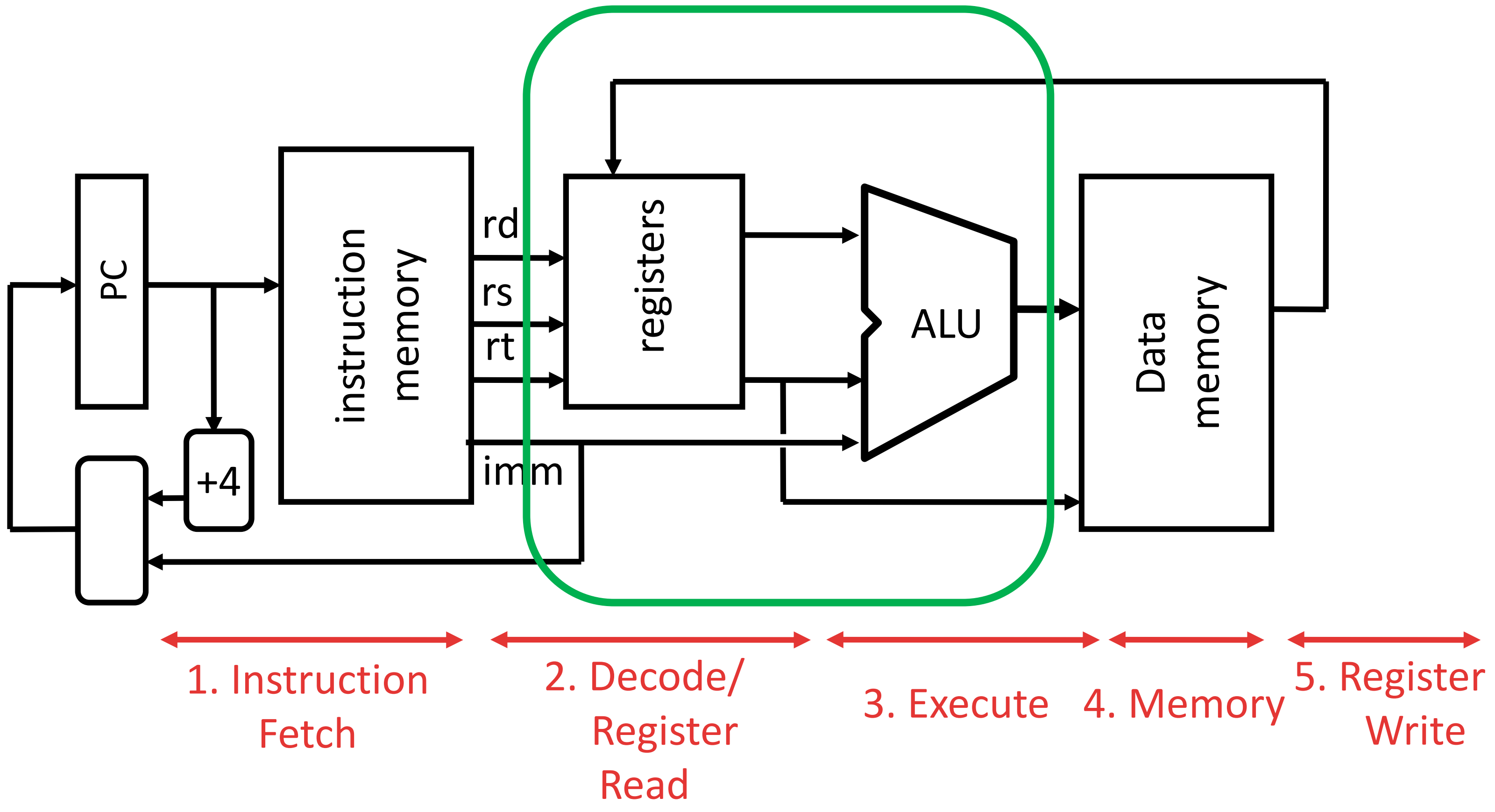
Announcement

- Lab #10
  - Due in 1 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
- Reading assignment
  - Chapter 5.7-5.11 of zyBooks
    - Make sure to do the Participation Activities
    - Due Friday (5/3) at 11:59pm
- Course evaluation online by 5/9

Announcement

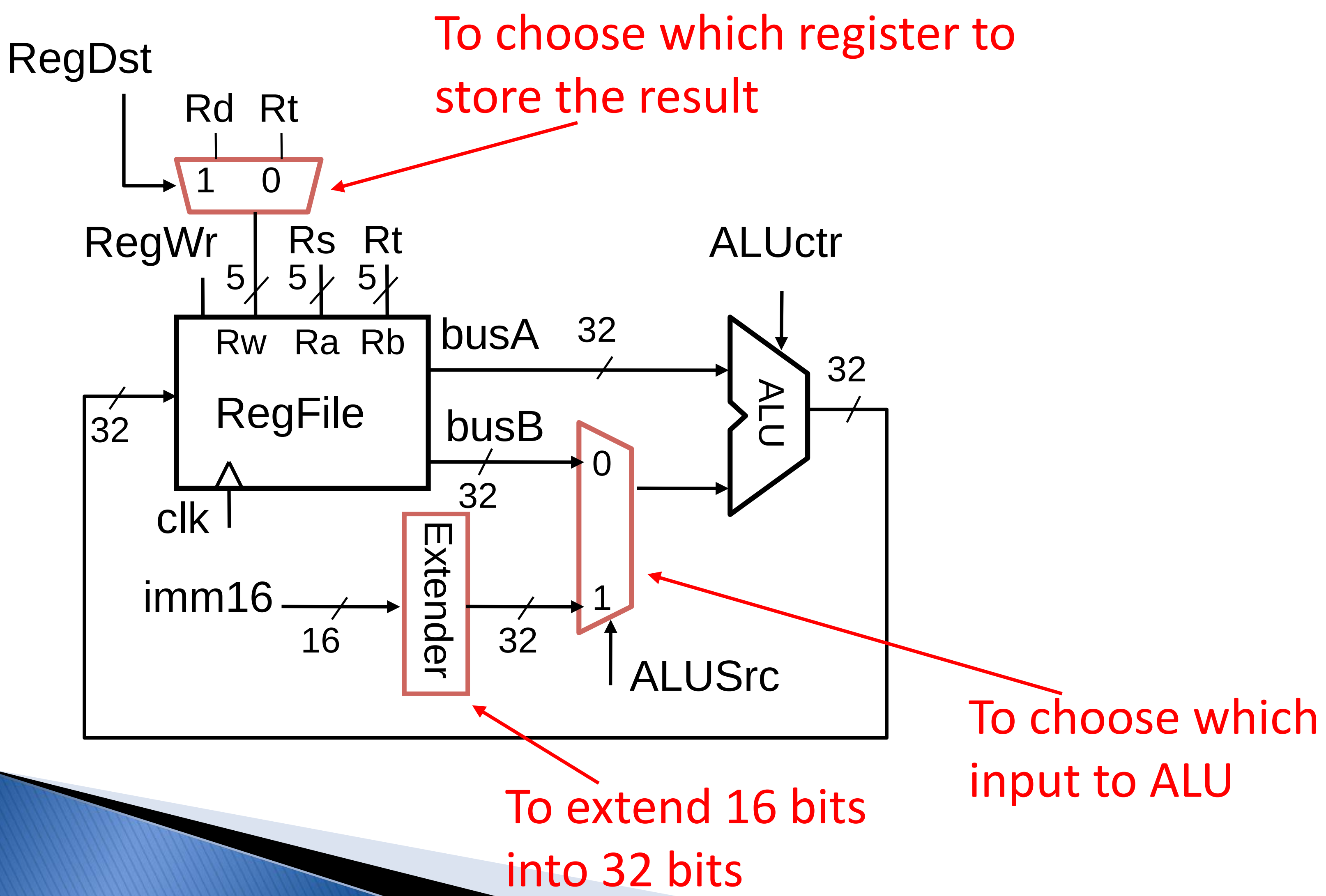
- Final Exam
  - 5/11 (Saturday), 11:30am – 2:30pm
  - Cover all
  - Practice exam in CatCourses
  - Closed book
  - 2 sheet of note (8.5” x 11”)
  - MIPS reference sheet will be provided
  - Review: 5/10 (Friday) 2-4pm, COB2 140

Generic Steps of Datapath



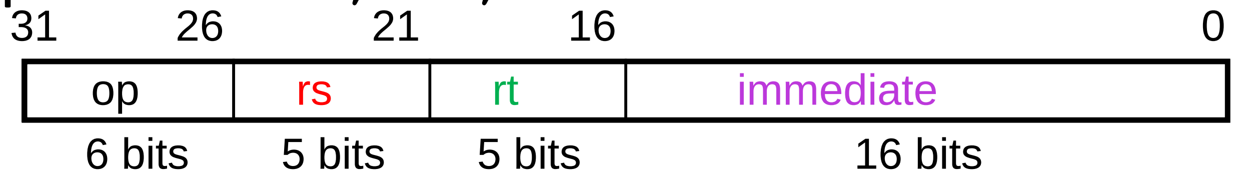
How do we handle the different register usage between r-type and i-type instructions?

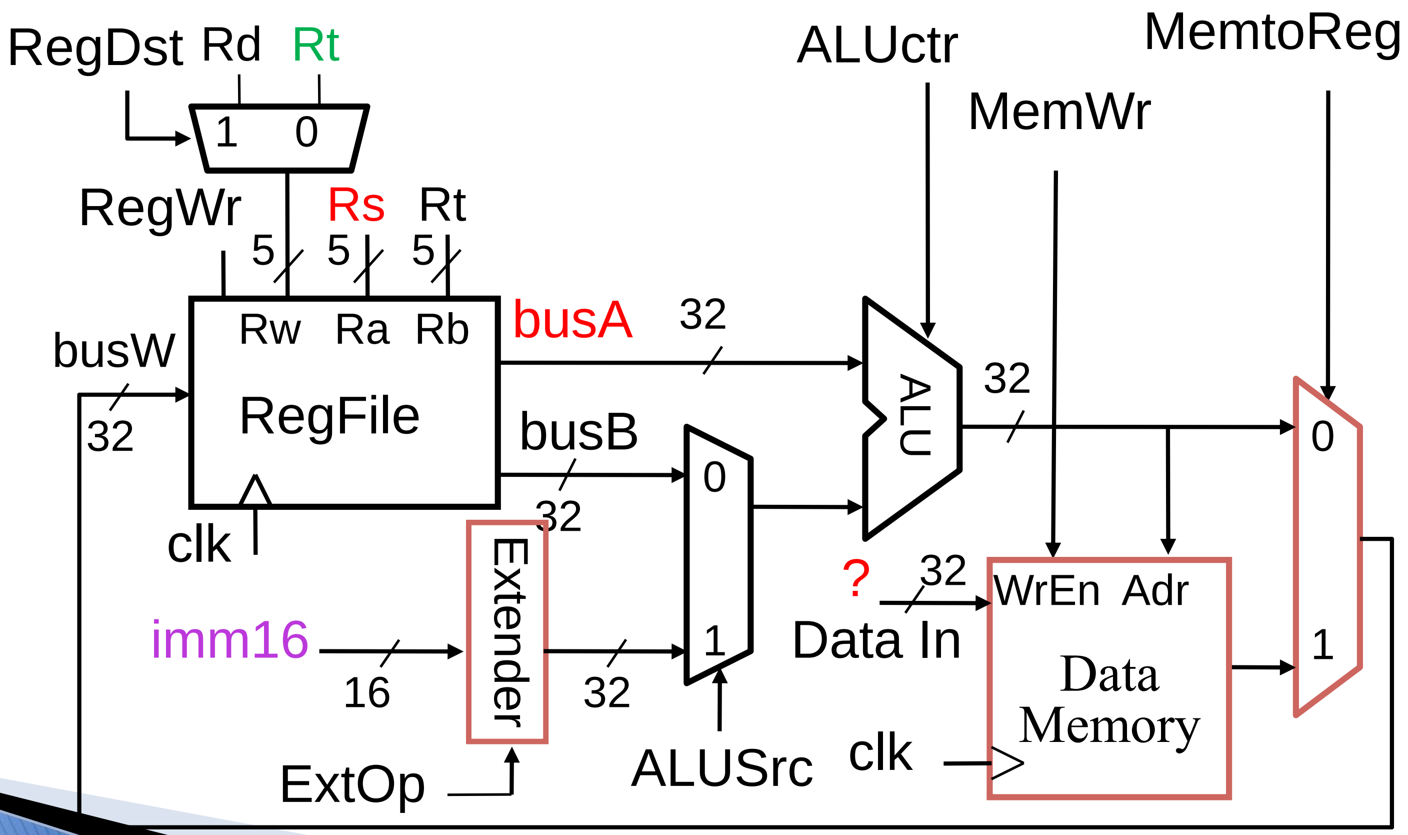
A zoomed in version of RegFile and ALU


To choose which register to store the result
RegDst
Rd
Rt
1
0
RegWr
Rs
Rt
5
5
5
ALUctr
Rw
Ra
Rb
busA
32
32
ALU
RegFile
32
busB
0
32
clk
Extender
imm16
16
32
1
ALUSrc
To choose which input to ALU
To extend 16 bits into 32 bits

Load Memory

- R[rt] = Mem[R[rs] + SignExt[imm16]]
- Example: `lw rt,rs,imm16`

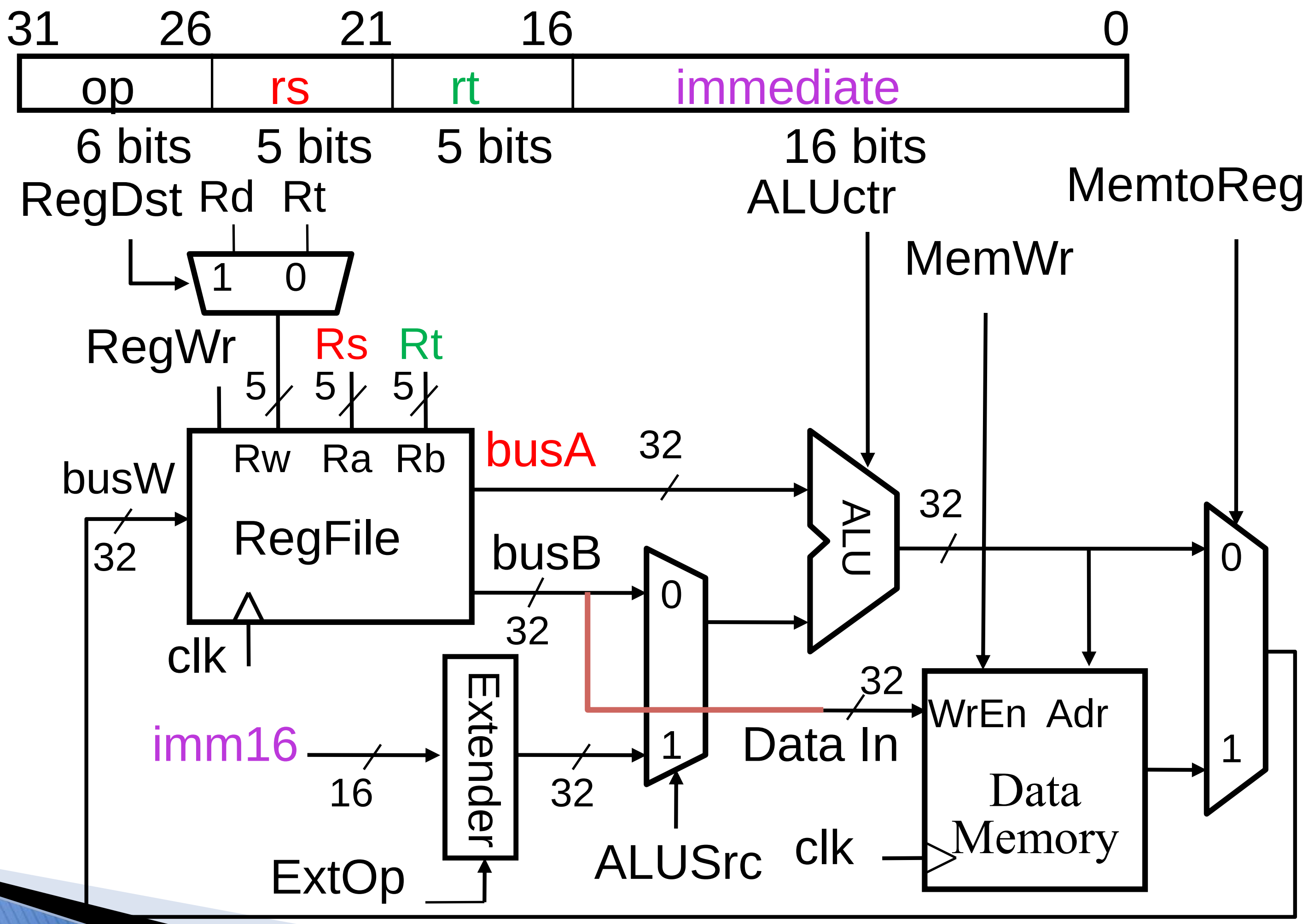




Store Memory

- Mem[R[rs] + SignExt[imm16]] = R[rt]

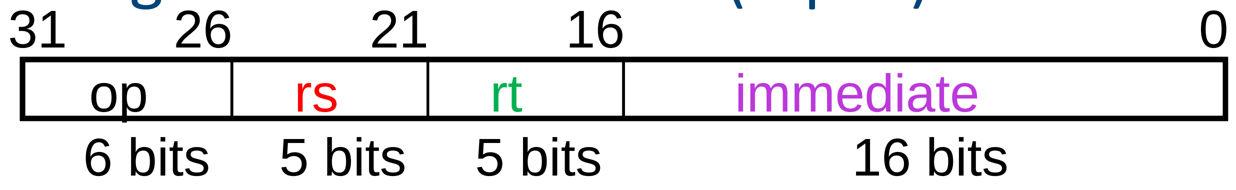
  Ex.: `sw rt, rs, imm16`

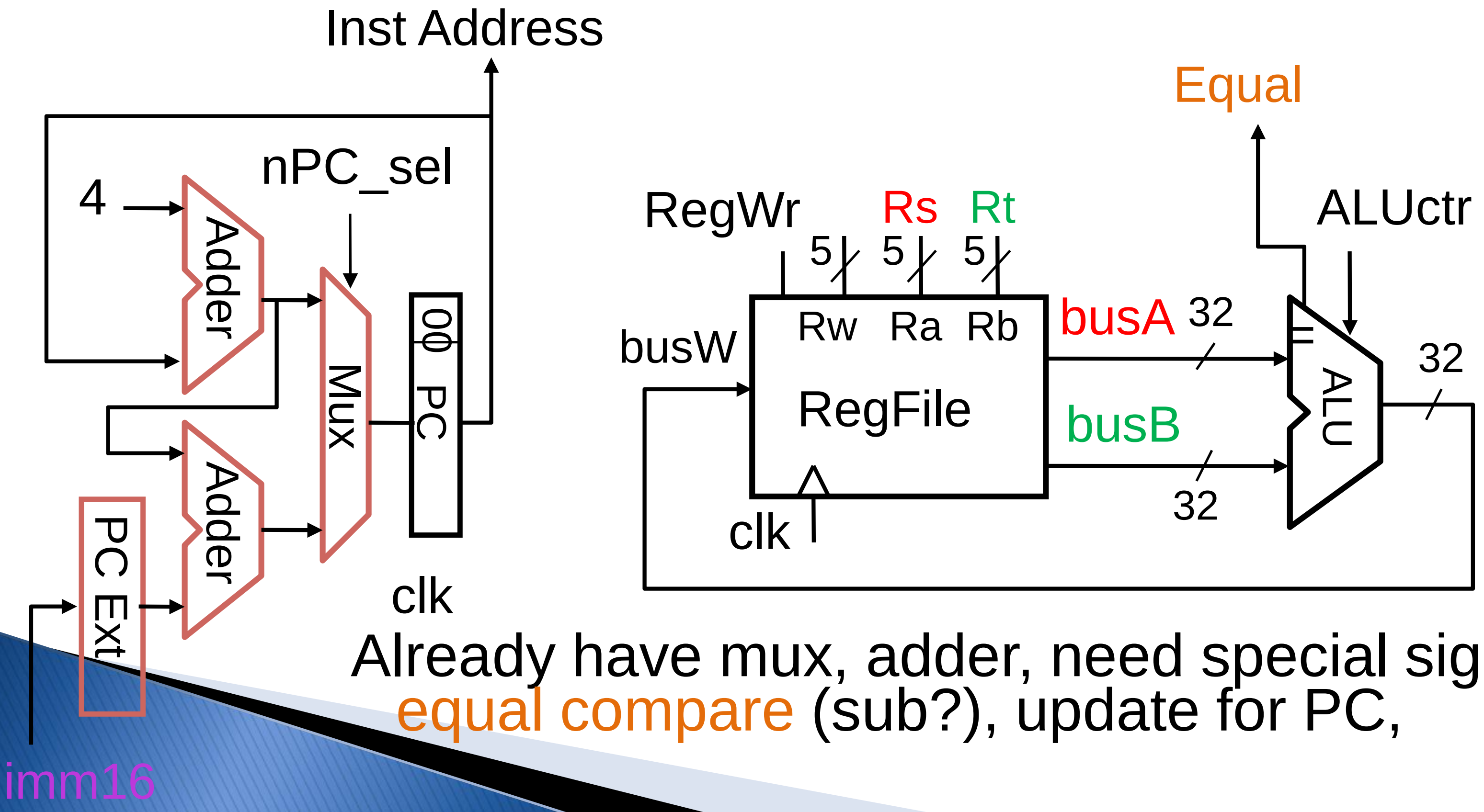


Datapath for Branch Operations

- beq rs, rt, imm16

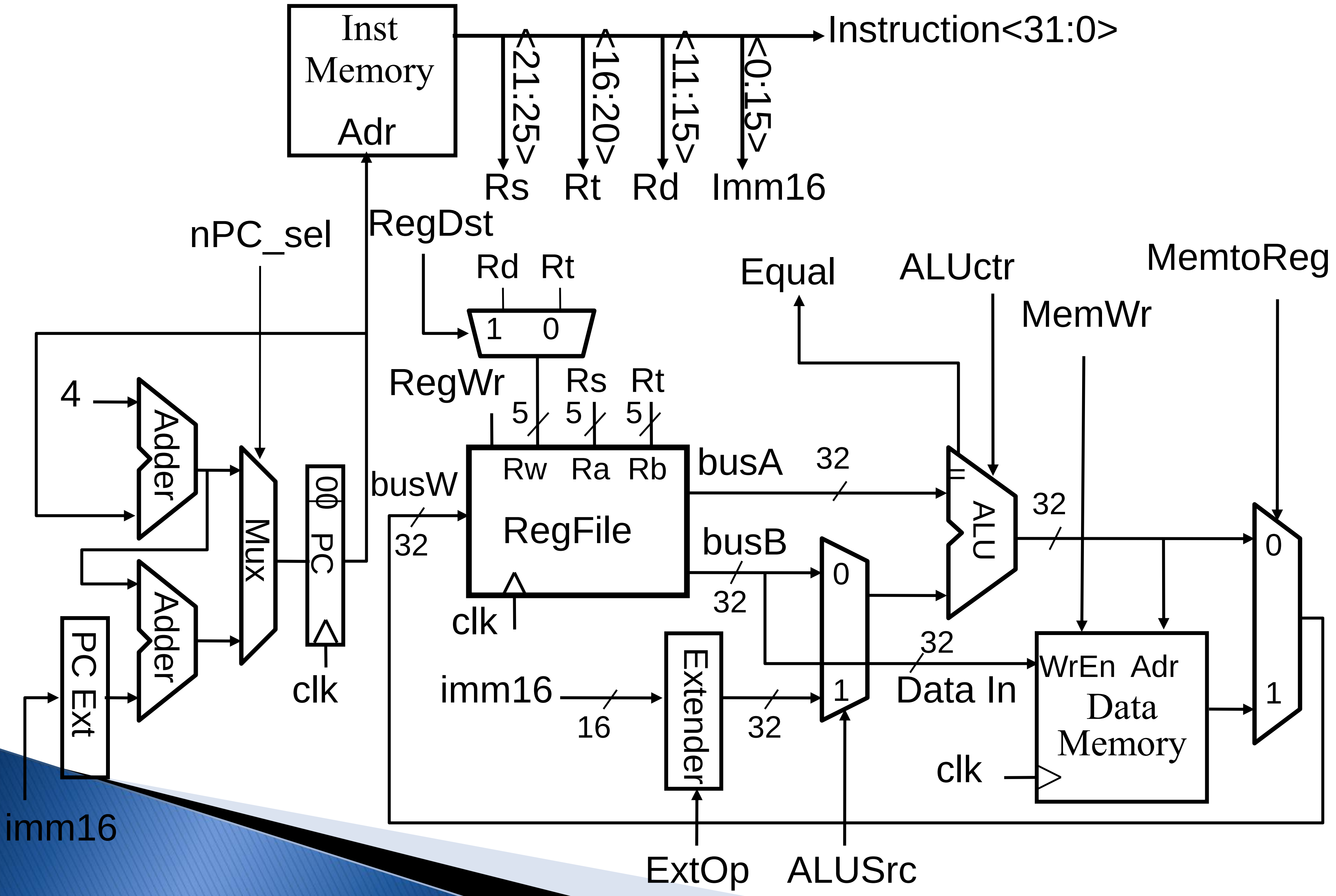
  Datapath generates condition (equal)



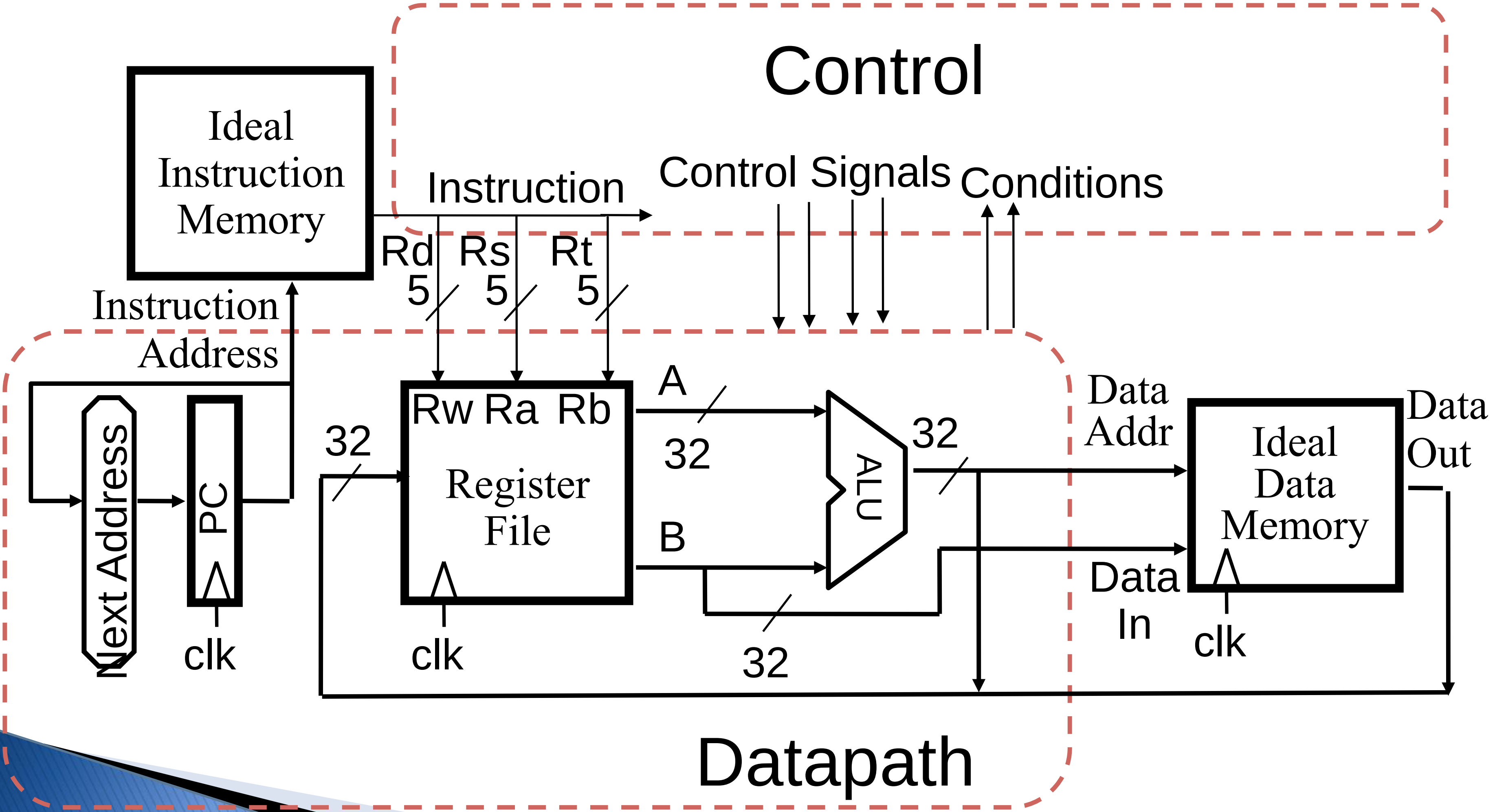


Already have mux, adder, need special sign need equal compare (sub?), update for PC,

Single Cycle Datapath

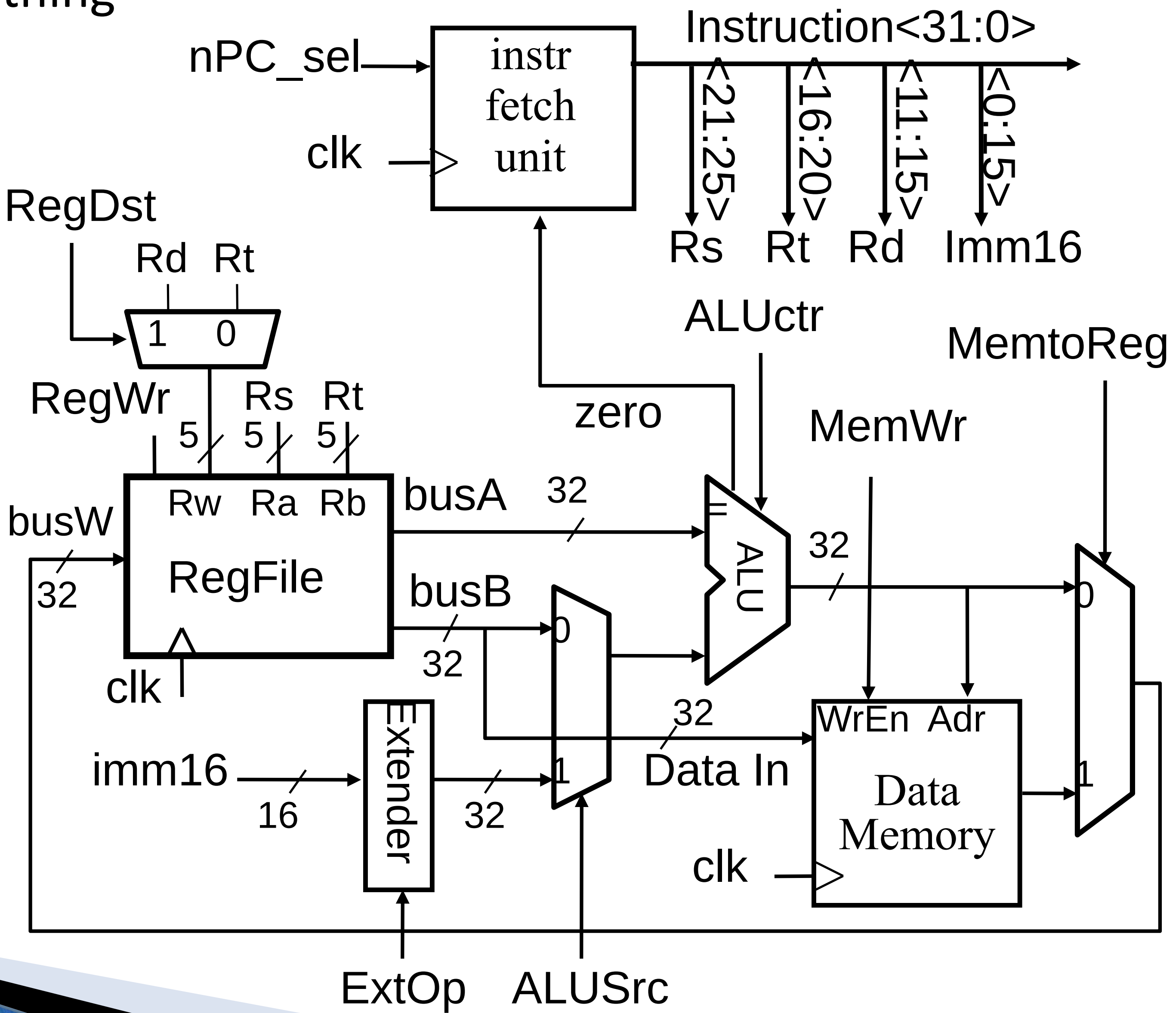


Abstract View of the Implementation



A Single Cycle Datapath

- We have everything except control signals

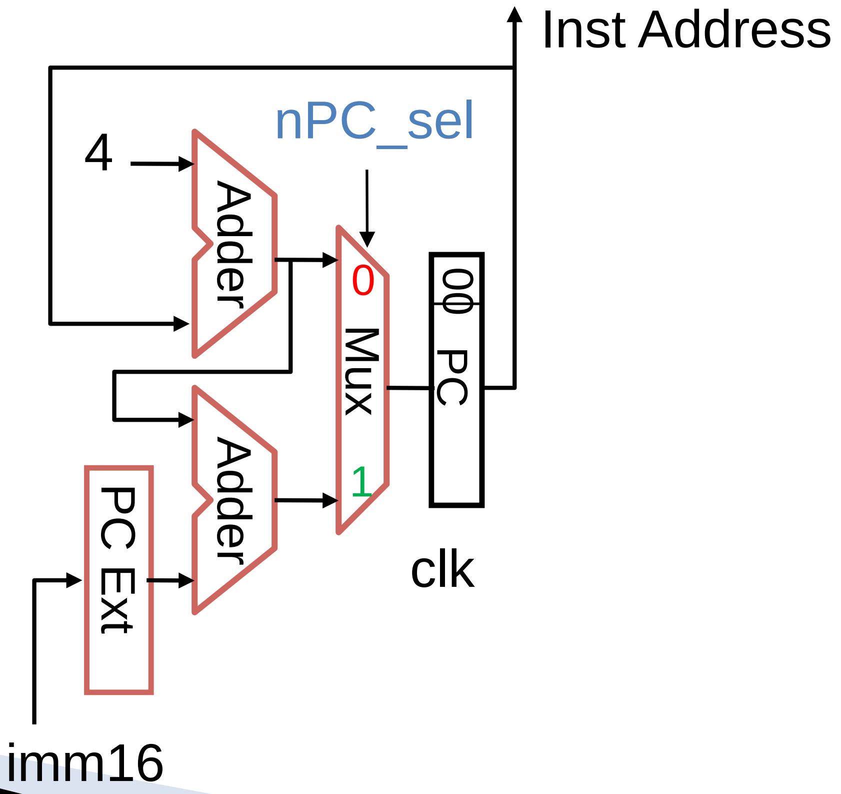


Meaning of the Control Signals

- nPC_sel: “+4”: $0 \Rightarrow PC \leftarrow PC + 4$
  “br”: $1 \Rightarrow PC \leftarrow PC + 4 + \{SignExt(Im16), 00\}$

  “n”=next

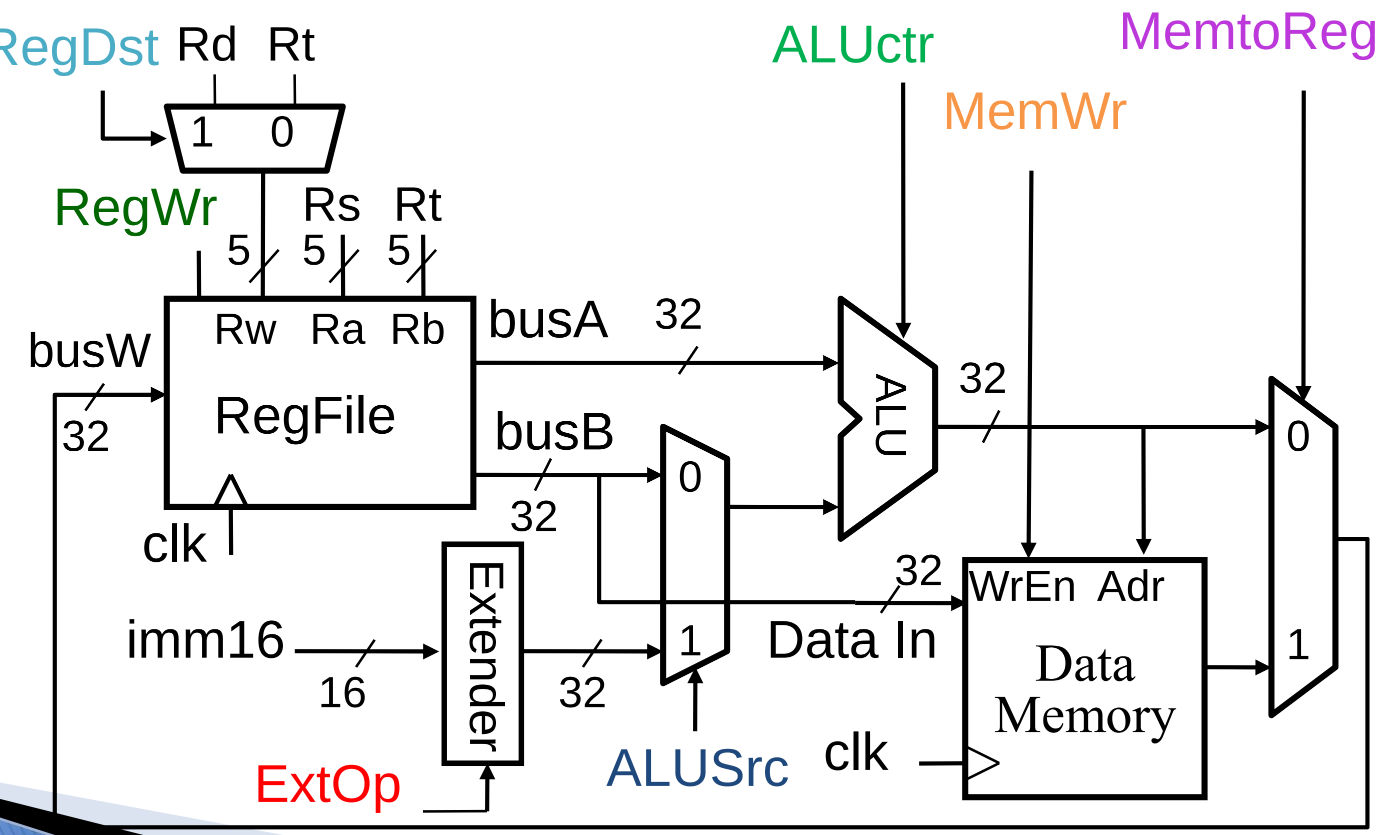
- Later in lecture: higher-level connection between mux and branch condition



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

31 26 21 16 11 6 0

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

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

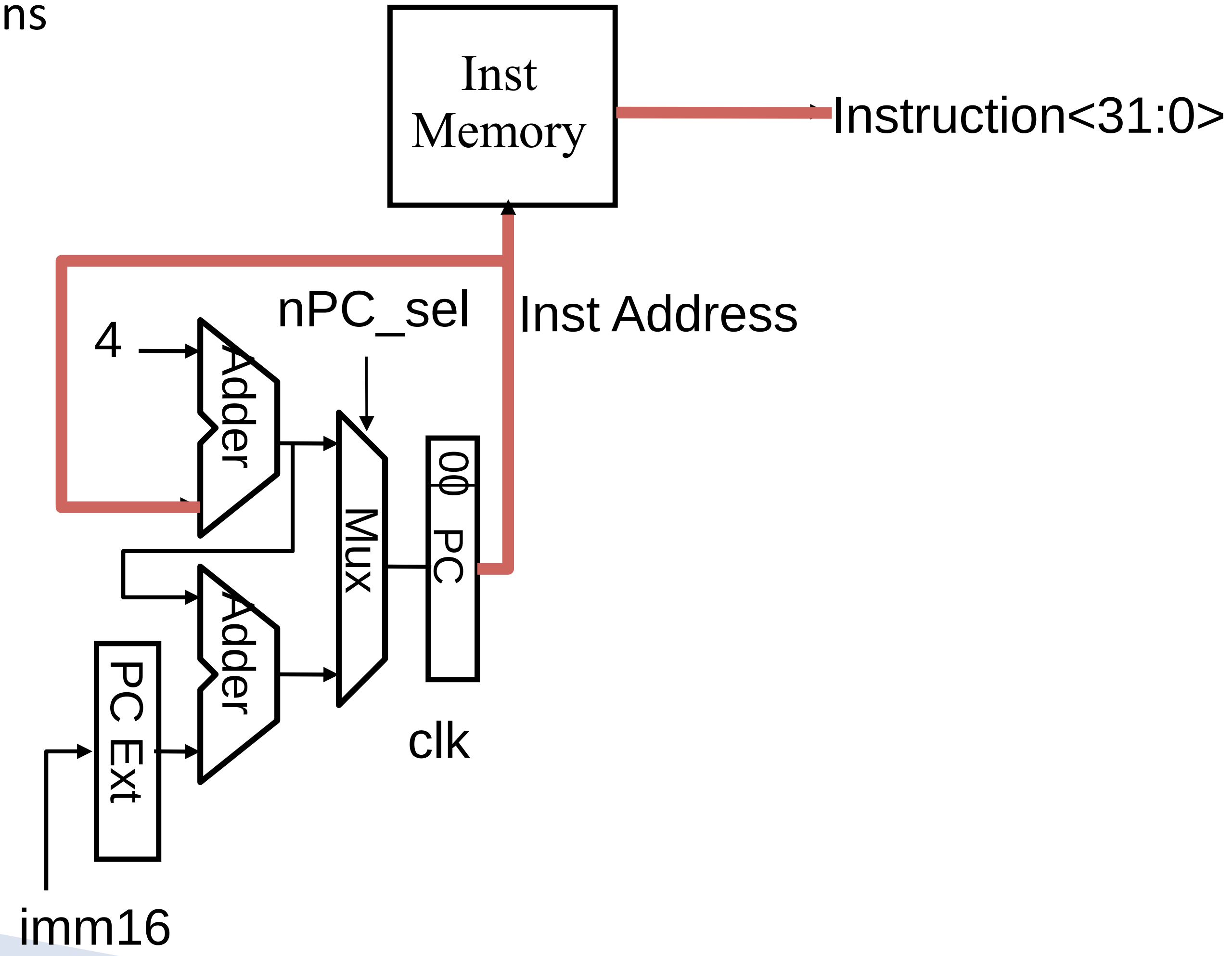
- MEM[PC] Fetch the instruction from memory
- R[rd] = R[rs] + R[rt] The actual operation
- PC = PC + 4 Calculate the next instruction’s address

Instruction Fetch Unit start of Add

- Fetch the instruction from Instruction memory:

  Instruction = MEM[PC]

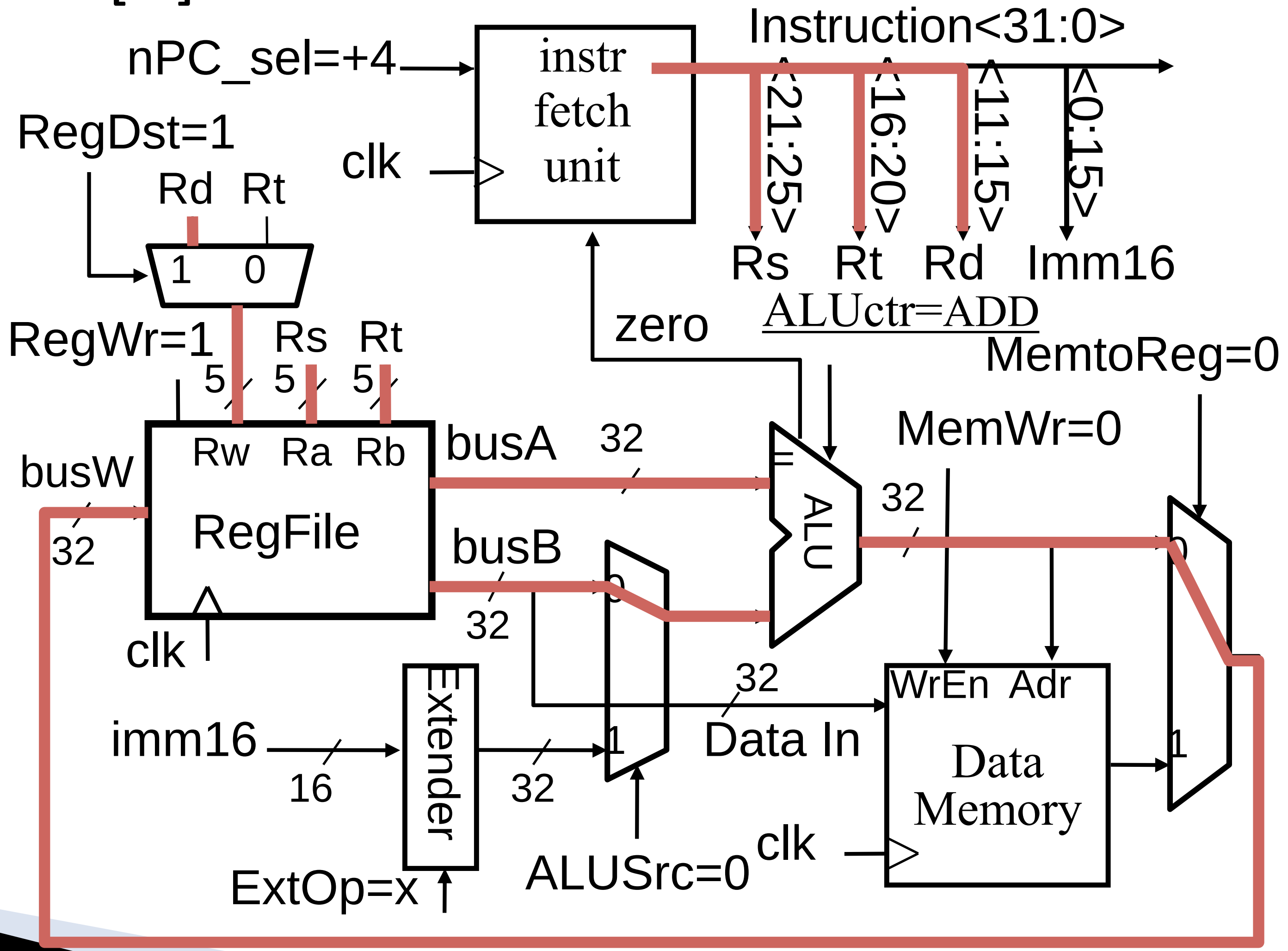
  - same for all instructions



The Single Cycle Datapath during Add

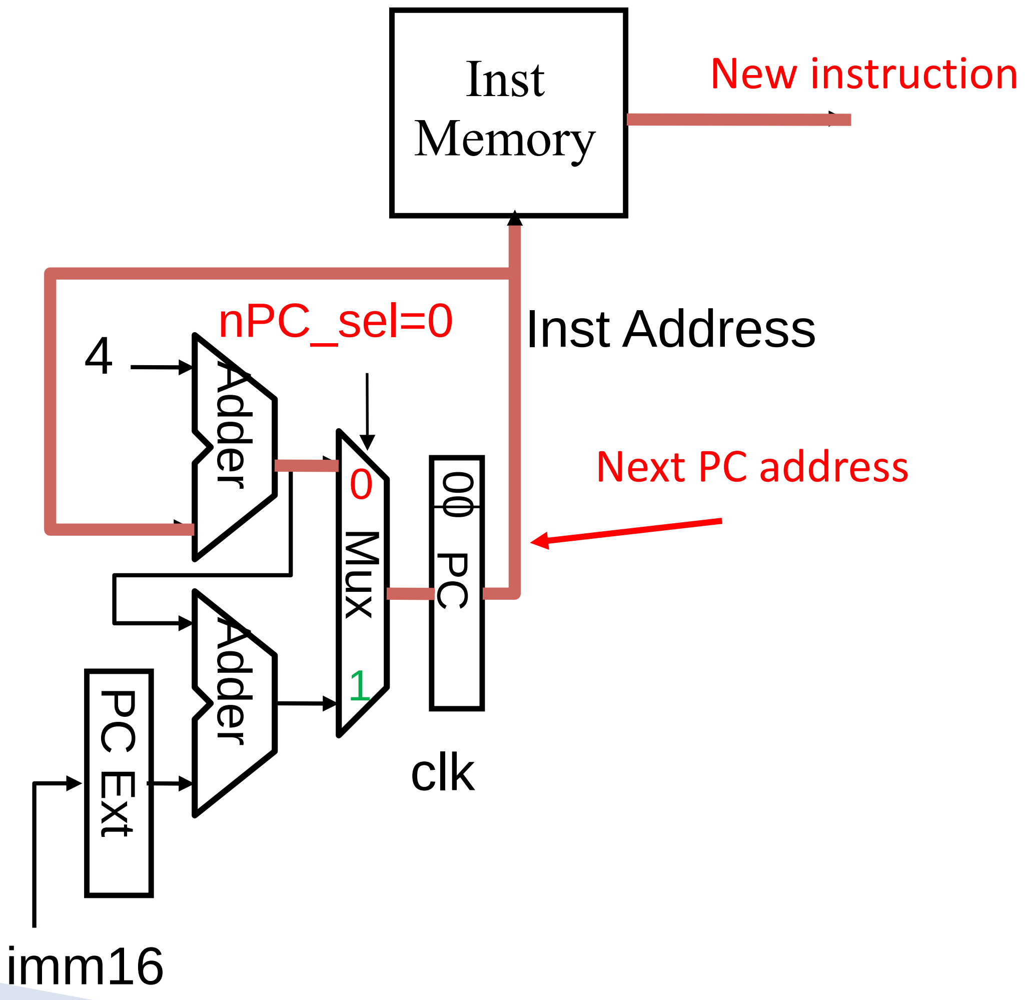
31	26	21	16	11	6	0
op	rs	rt	rd	shamt	funct	

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

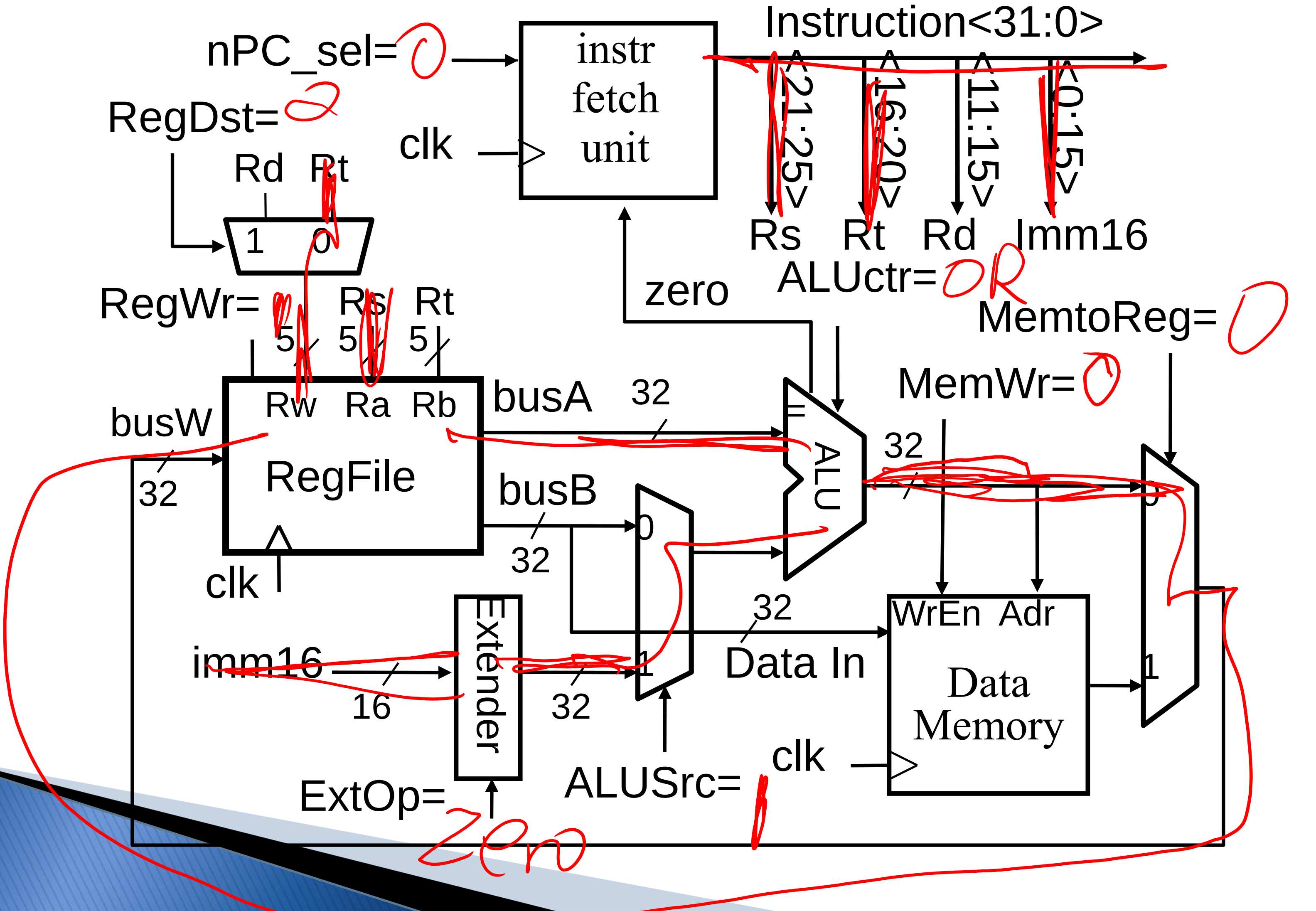


Single Cycle Datapath for Ori

31 26 21 16 0

op	rs	rt	immediate

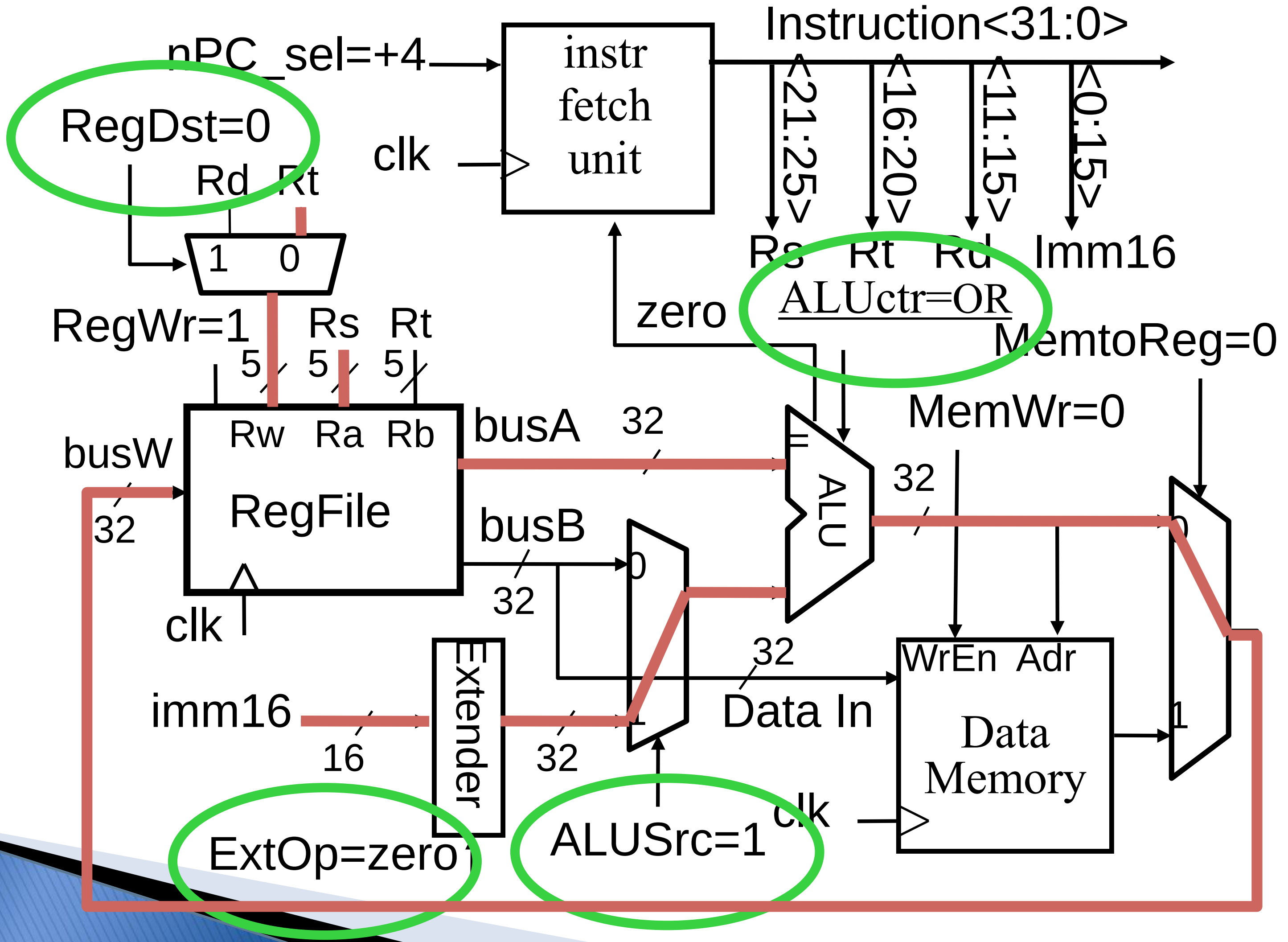
- R[rt] = R[rs] OR ZeroExt[Imm16]



Single Cycle Datapath for Ori

31	26	21	16	0
op	rs	rt	immediate	

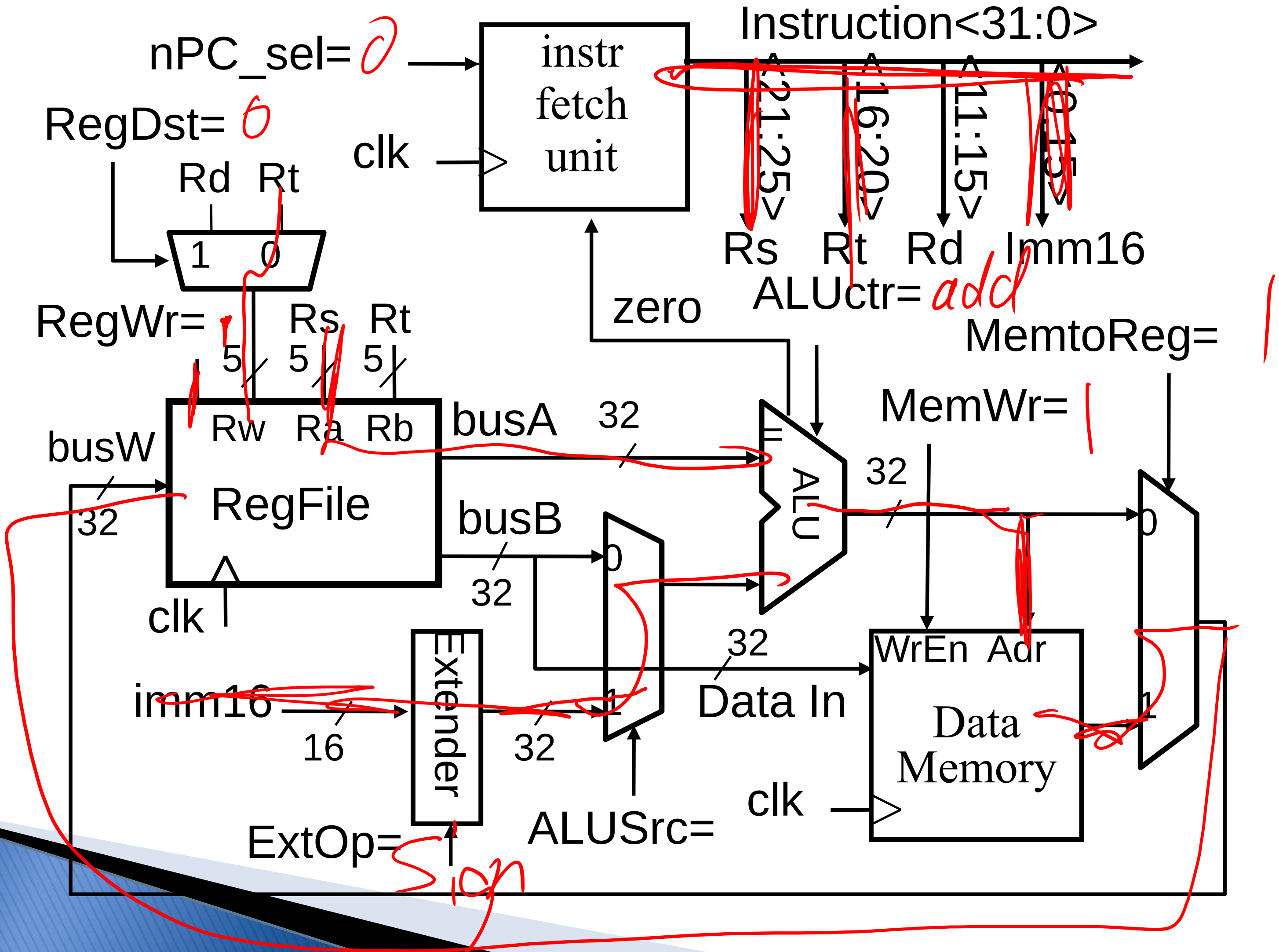
- R[rt] = R[rs] OR ZeroExt[Imm16]



Single Cycle Datapath for LW

31	26	21	16	0
op	rs	rt	immediate	

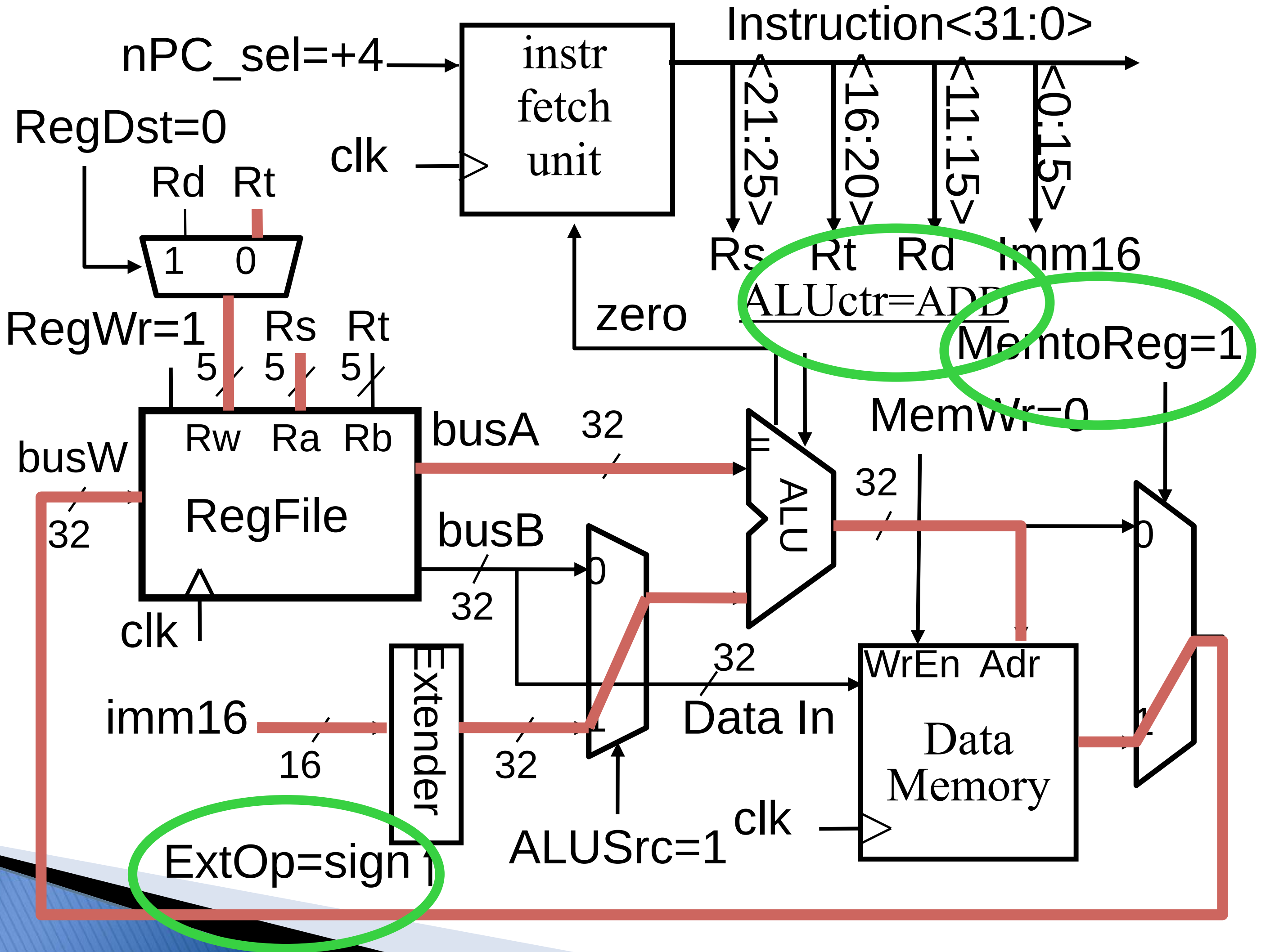
- R[rt] = Data Memory {R[rs] + SignExt[imm16]}



Single Cycle Datapath for LW

31	26	21	16	0
op	rs	rt	immediate	

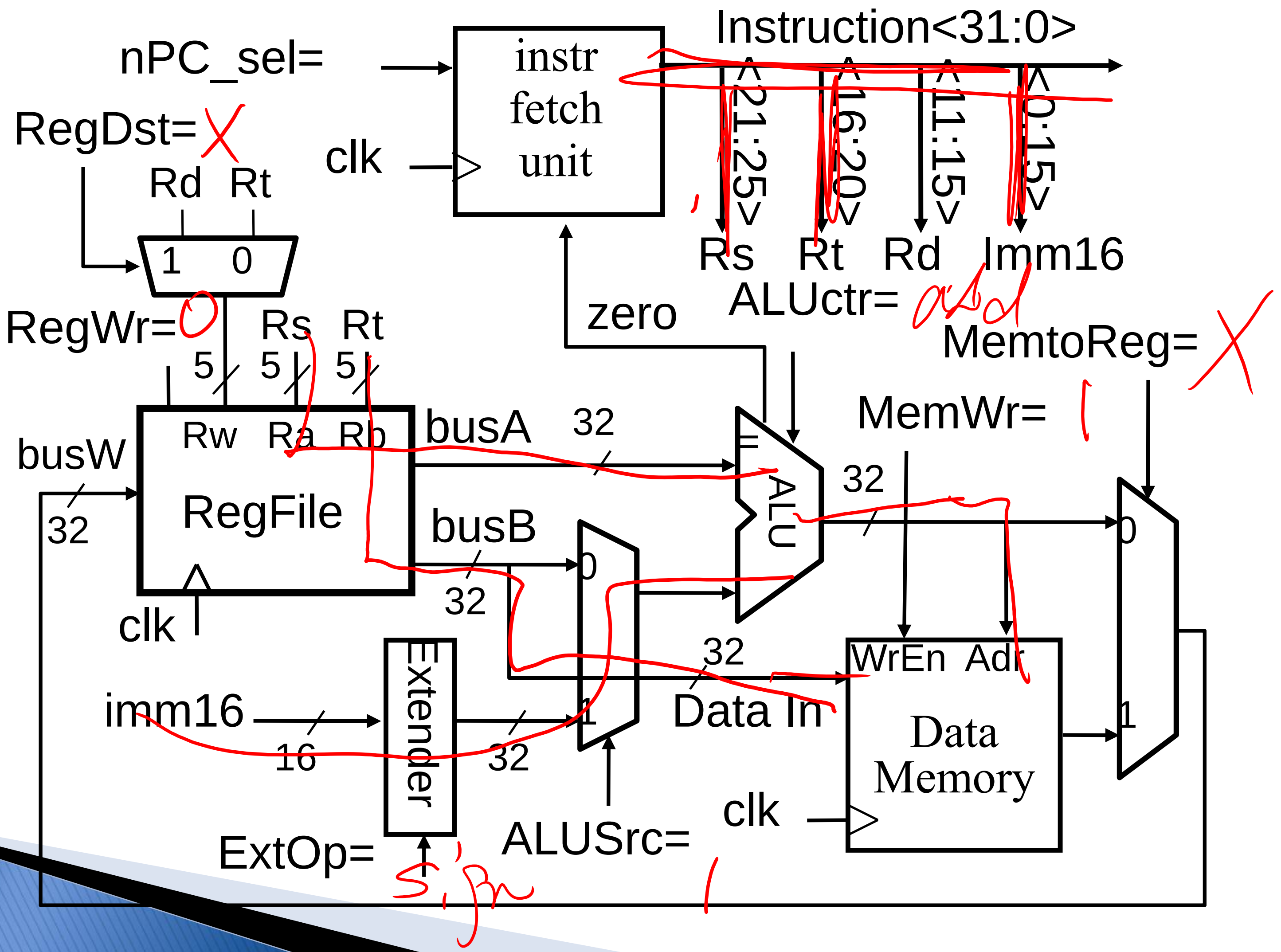
- R[rt] = Data Memory {R[rs] + SignExt[imm16]}



Single Cycle Datapath for SW

31	26	21	16	0
op	rs	rt	immediate	

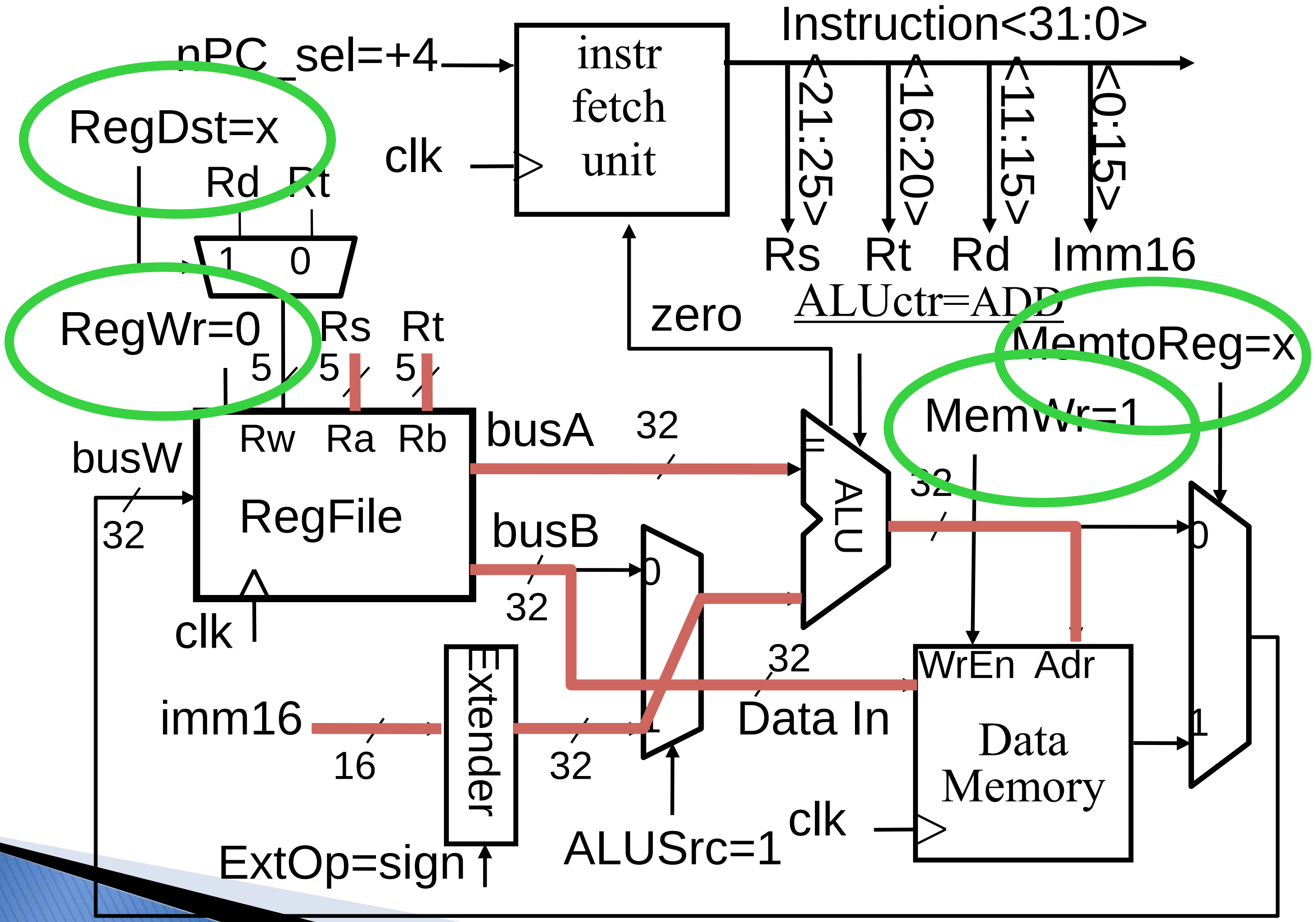
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



Single Cycle Datapath for SW

31	26	21	16	0
op	rs	rt	immediate	

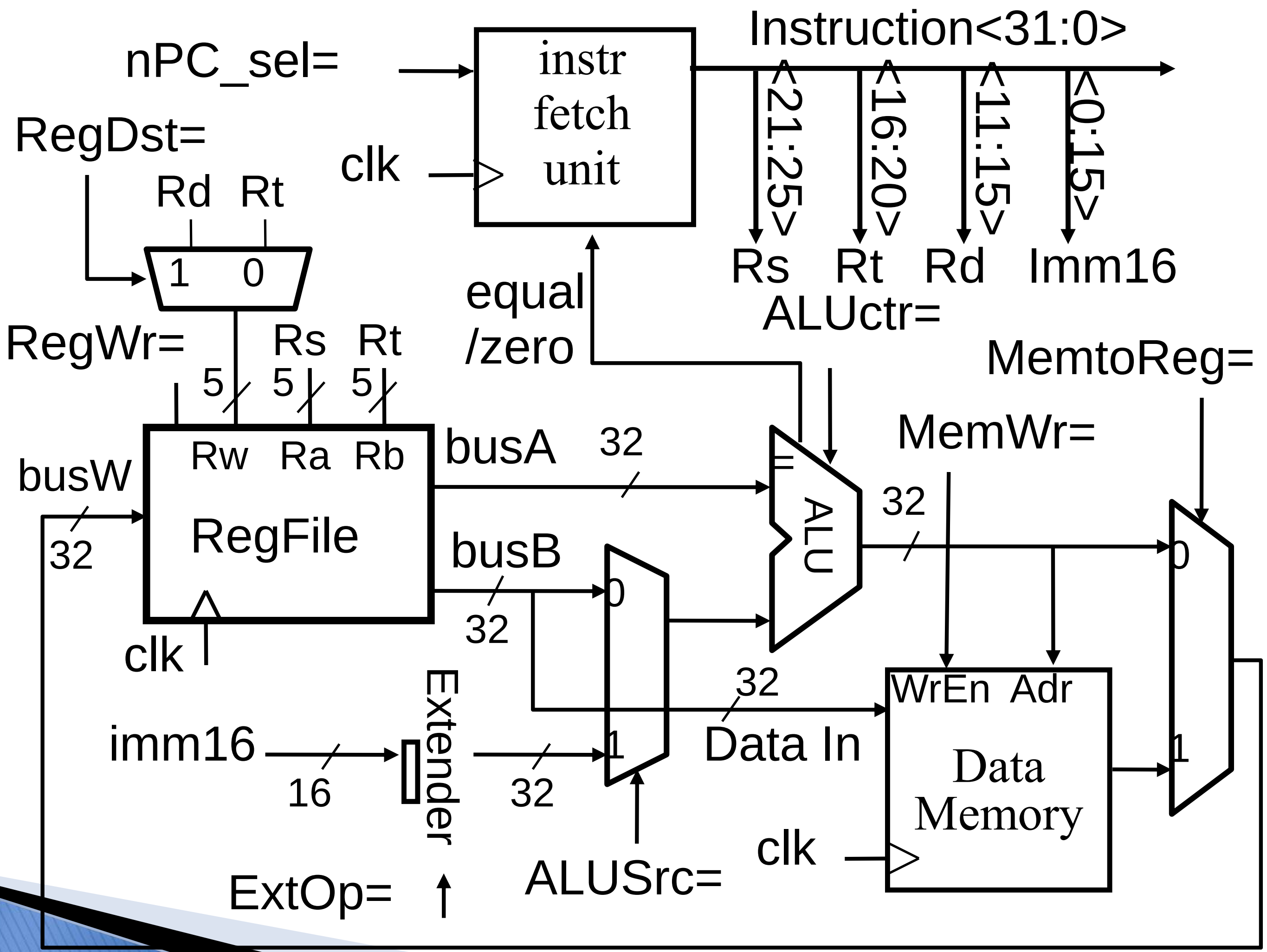
- Data Memory {R[rs] + SignExt[imm16]} = R[rt]



Single Cycle Datapath for Branch

31	26	21	16	0
op	rs	rt	immediate	

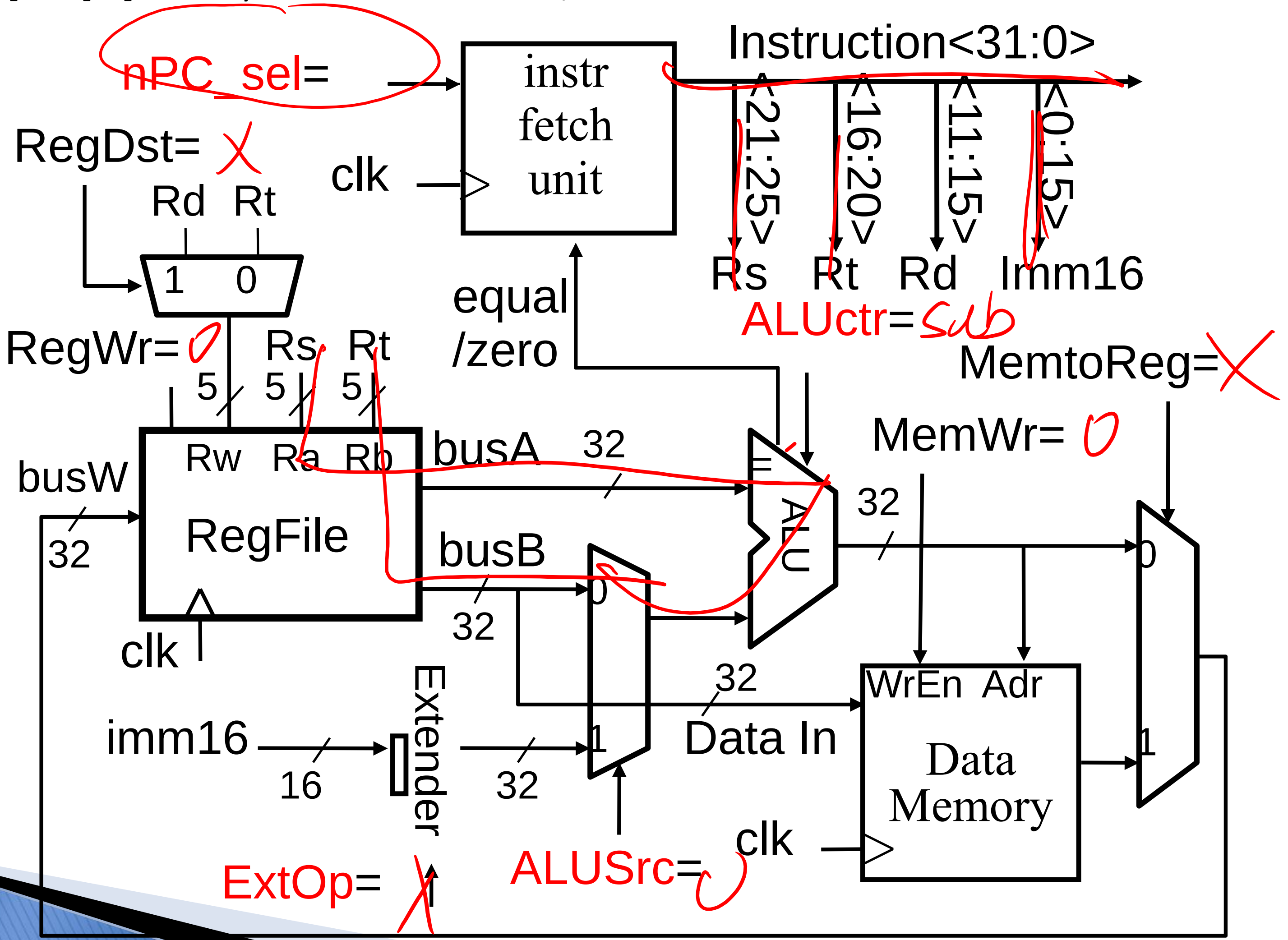
- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



Single Cycle Datapath for Branch

31	26	21	16	0
op	rs	rt	immediate	

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

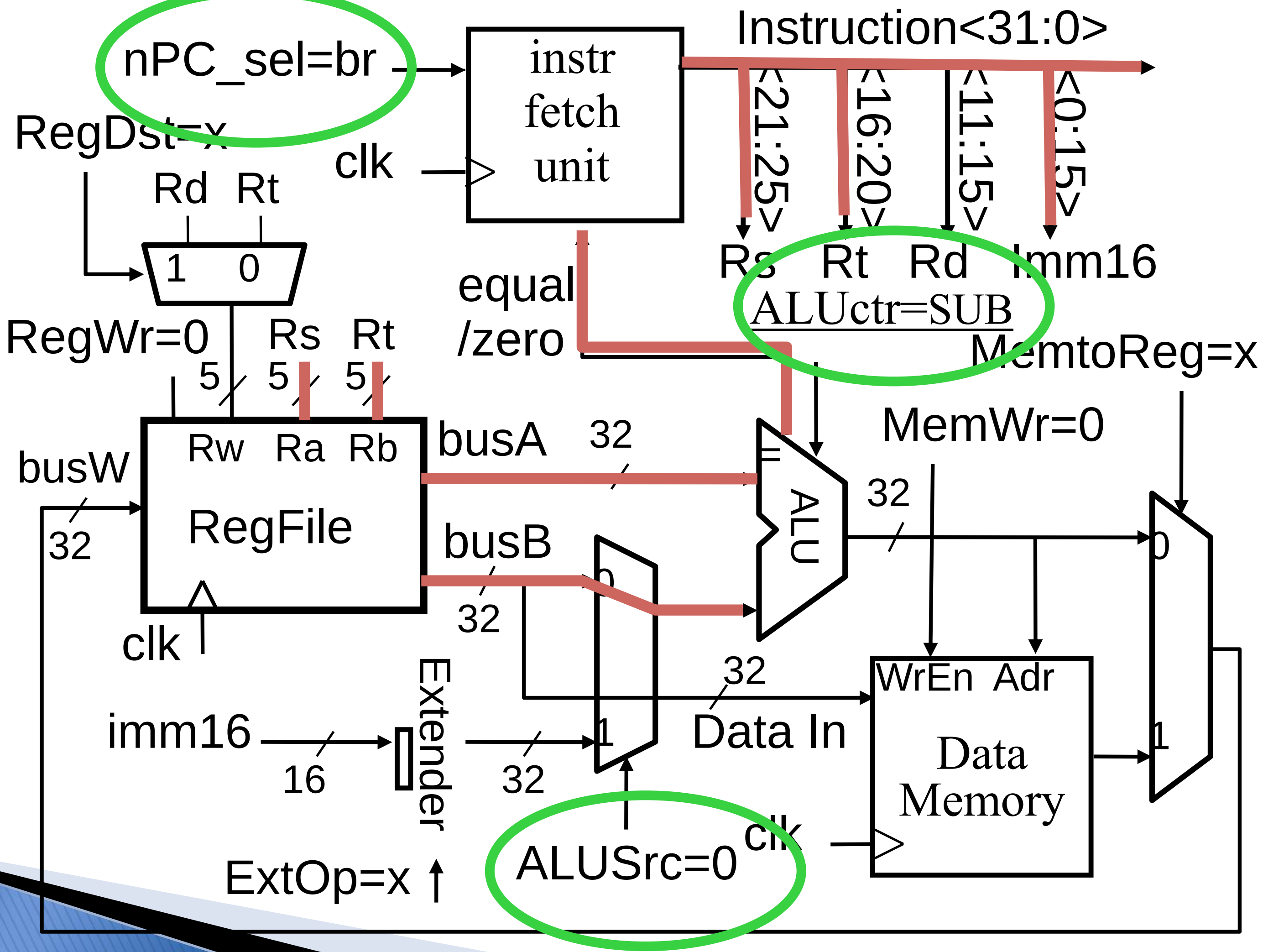


Single Cycle Datapath for Branch

31 26 21 16 0

op	rs	rt	immediate

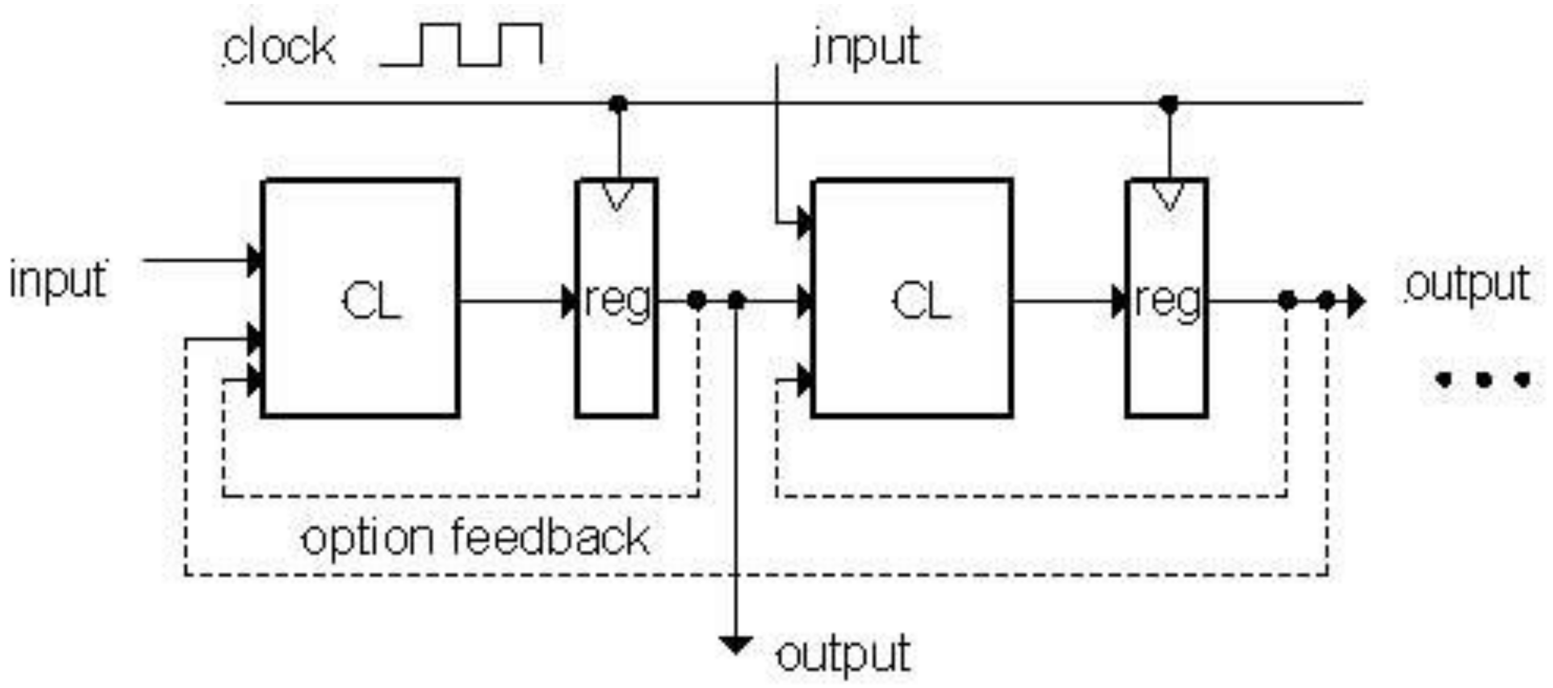
- if (R[rs] - R[rt] == 0) then Zero = 1 ; else Zero = 0



Type of Circuits

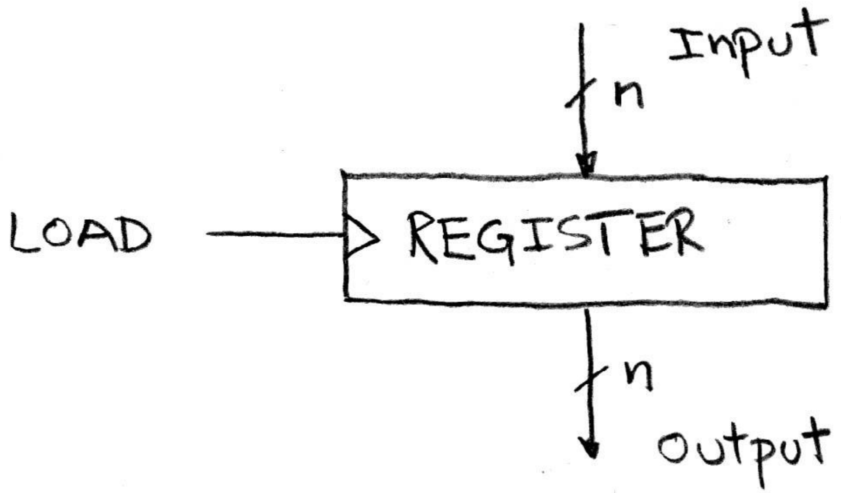
- Synchronous Digital Systems are made up of two basic types of circuits:
- Combinational Logic (CL) circuits
    - Our previous adder circuit is an example.
    - Output is a function of the inputs only.
    - Similar to a pure function in mathematics, y = f(x). (No way to store information from one invocation to the next. No side effects)
- State Elements: circuits that **store** information.

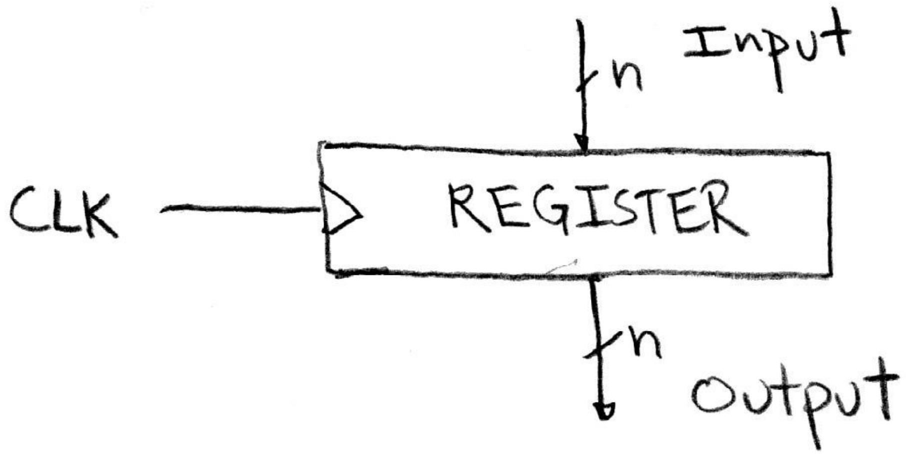
General Synchronous Systems



- Collection of CL blocks separated by registers.
- Registers may be back-to-back and CL blocks may be back-to-back.
- Feedback is optional.
- Clock signal(s) connects only to clock input of registers.

Circuits with STATE (register)

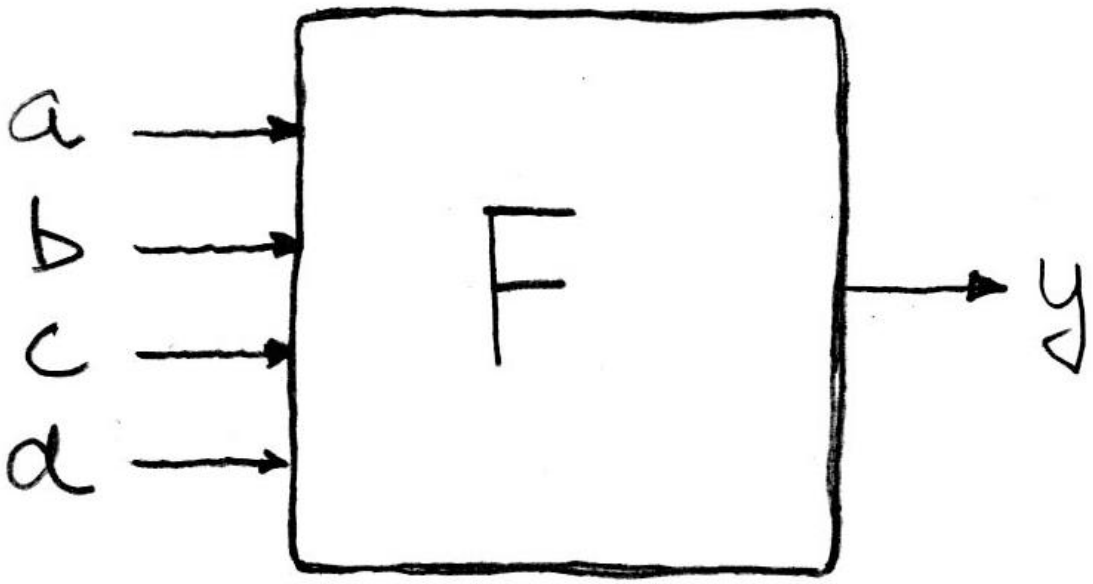




Uses for State Elements

- As a place to store values for some indeterminate amount of time:
  - Register files (like $1-$31 on the MIPS)
  - Memory (caches, and main memory)
- Help control the flow of information between combinational logic blocks.
  - State elements are used to hold up the movement of information at the inputs to combinational logic blocks and allow for orderly passage.

Truth Tables

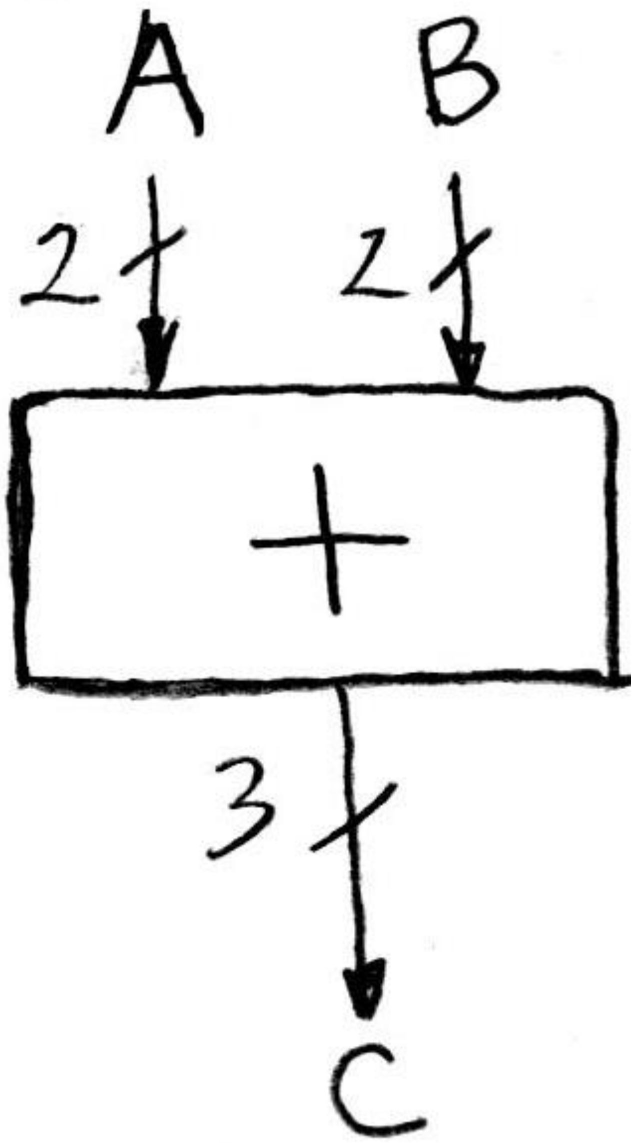


a	b	c	d	y
0	0	0	0	F(0,0,0,0)
0	0	0	1	F(0,0,0,1)
0	0	1	0	F(0,0,1,0)
0	0	1	1	F(0,0,1,1)
0	1	0	0	F(0,1,0,0)
0	1	0	1	F(0,1,0,1)
0	1	1	0	F(0,1,1,0)
0	1	1	1	F(0,1,1,1)
1	0	0	0	F(1,0,0,0)
1	0	0	1	F(1,0,0,1)
1	0	1	0	F(1,0,1,0)
1	0	1	1	F(1,0,1,1)
1	1	0	0	F(1,1,0,0)
1	1	0	1	F(1,1,0,1)
1	1	1	0	F(1,1,1,0)
1	1	1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

TT #1: XOR, 1 iff a/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	B	C
a_1a_0	b_1b_0	$c_2c_1c_0$
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

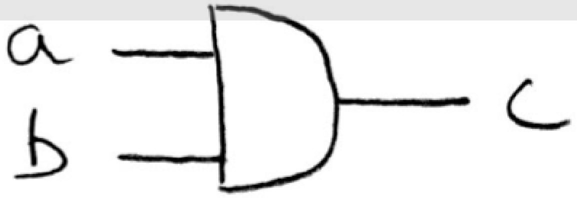
How Many Rows?

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

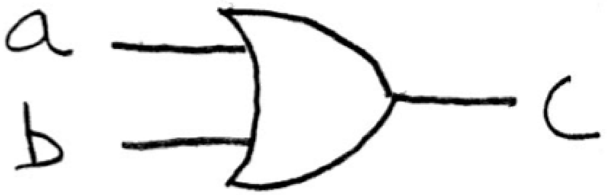
Logic Gates (1/2)

AND



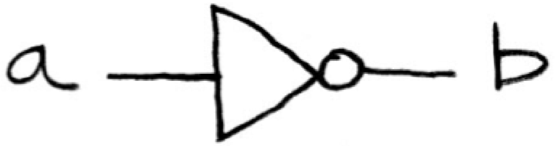
ab	c
00	0
01	0
10	0
11	1

OR



ab	c
00	0
01	1
10	1
11	1

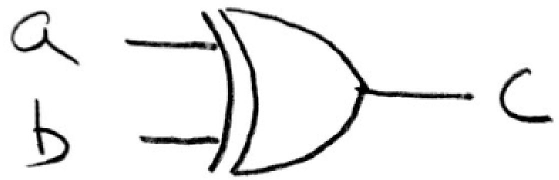
NOT



a	b
0	1
1	0

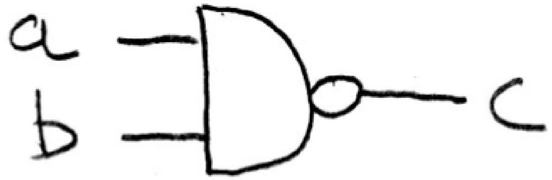
Logic Gates (2/2)

XOR



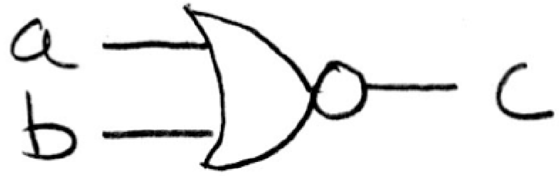
ab	c
00	0
01	1
10	1
11	0

NAND



ab	c
00	1
01	1
10	1
11	0

NOR



ab	c
00	1
01	0
10	0
11	0

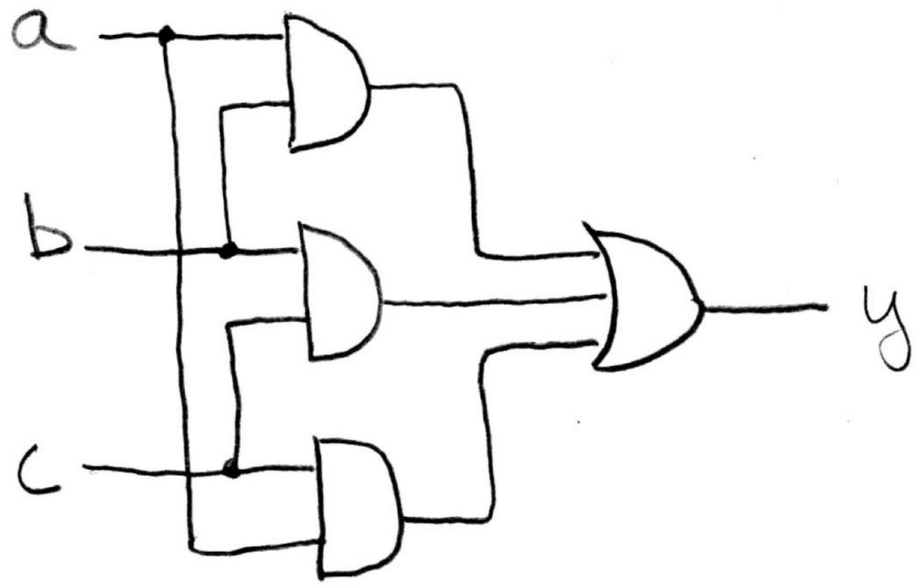
2-input gates extend to n-inputs

- 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

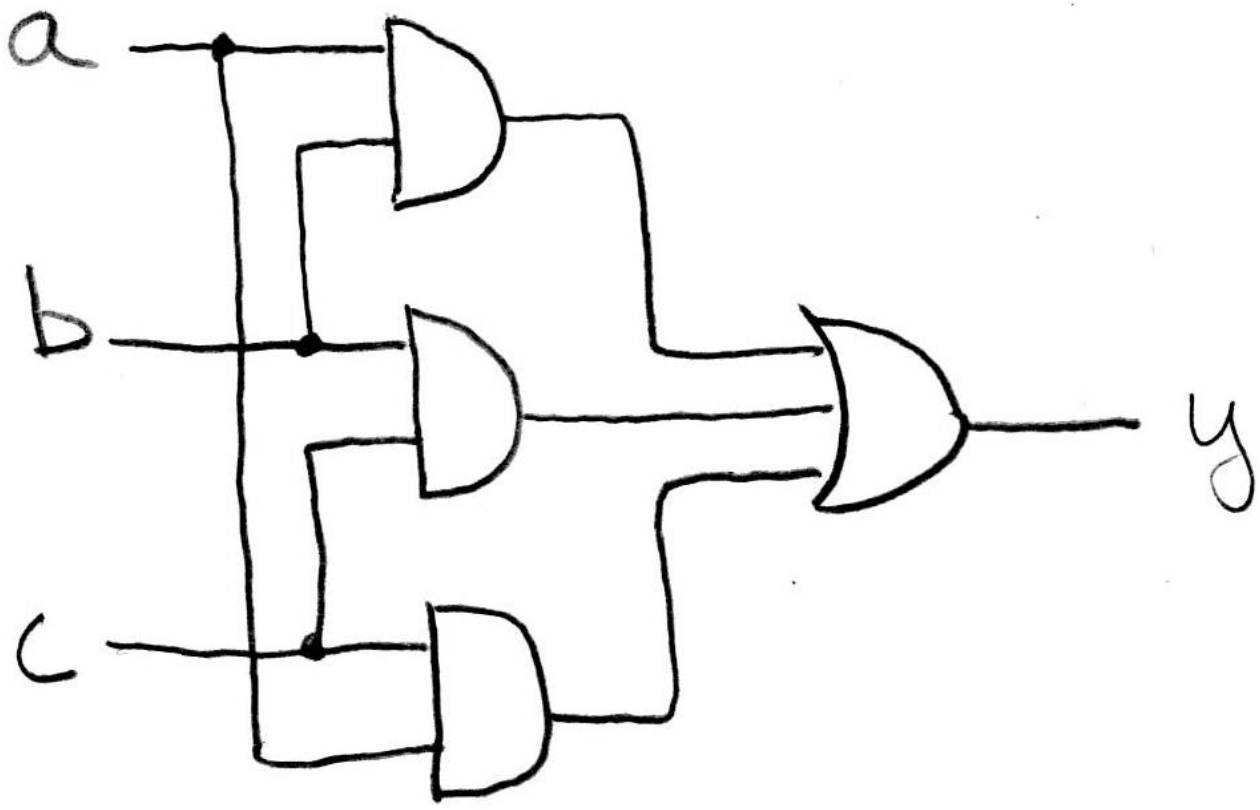
a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

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 • b + a • c + b • c

y = ab + ac + bc

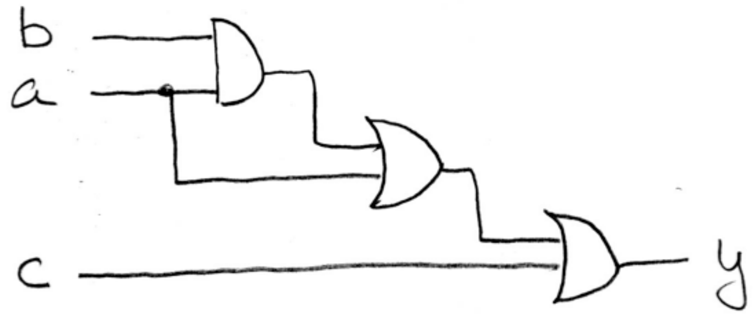
Laws of Boolean Algebra

$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$	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

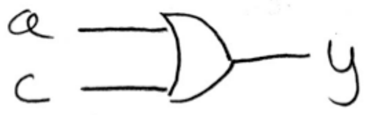
$$\downarrow$$

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

algebraic simplification

BA also great for circuit <u>verification</u>
Circ X = Circ Y?
use BA to prove!

$$\downarrow$$



simplified circuit

Canonical forms (1/2)

Sum-of-products
(ORs of ANDs)

	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

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

Canonical forms (2/2)

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