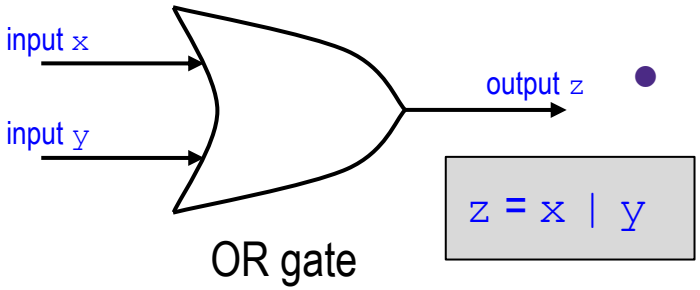
Black Box Architecture

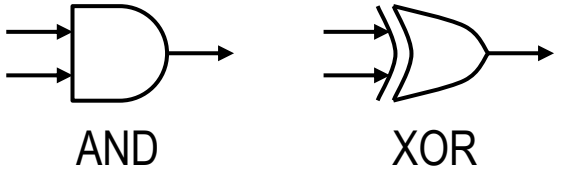
A black box is employed to abstract the function of a device.

Notation:

- **values of 0 / 1 travel along wires (lines)**
- **arrows indicate direction of travel**
- **there are inputs and outputs**



Also:

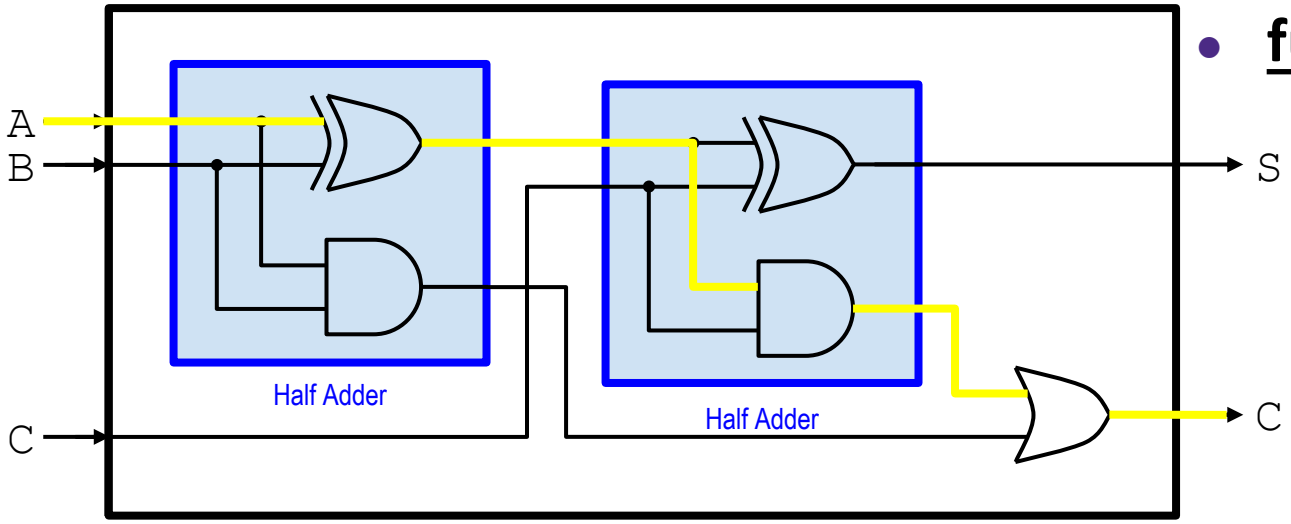


After a change in inputs, there is a time delay for outputs

=: propagation delay (t_{pd})

- **on a picosecond to nanosecond scale**
- **find longest distance from inputs to outputs**

Mystery Circuit : Full Adder

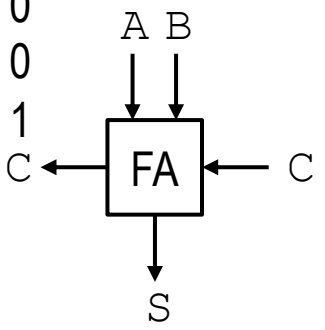


Full Adder

- **function table:**
  - basically a truth table

A B C	C S
0 0 0	0 0
0 0 1	0 1
0 1 0	0 1
0 1 1	1 0
1 0 0	0 1
1 0 1	1 0
1 1 0	1 0
1 1 1	1 1

3-bit Addition!



Q. What's the propagation delay?

- **3 gate delays (highlighted)**

Q. What does the circuit accomplish?

- **algebra:** `S = A ^ B ^;CC = (A & B) | (C & (A ^ B))`

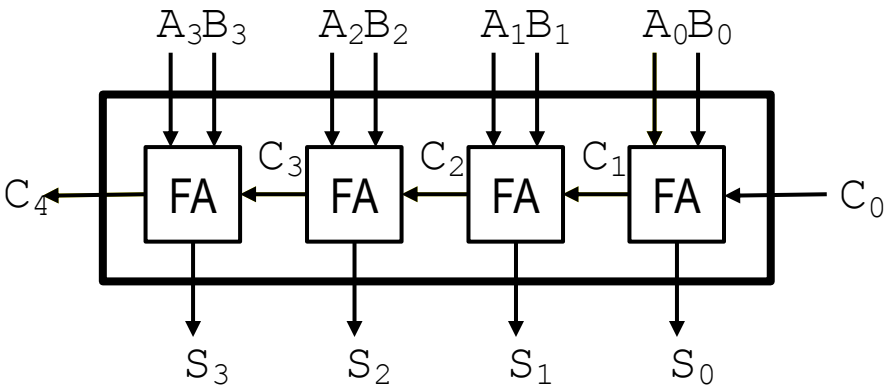
; $C = AB + C(A \oplus B)$

$S = A \oplus B \oplus C$

Computing with Combinational Circuits

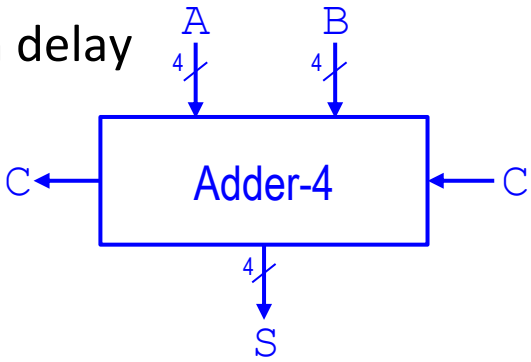
Definition: A combinational circuit computes a pure function, i.e., its outputs react only based on its inputs. There are no feedback loops and no state information (memory) is maintained.

Theorem: Every Boolean function can be implemented with NAND and NOT. Circuits are modular



… a 4-bit ripple carry adder!

- adds by columns
- propagation delay = 9 **(2*n* + 1)**



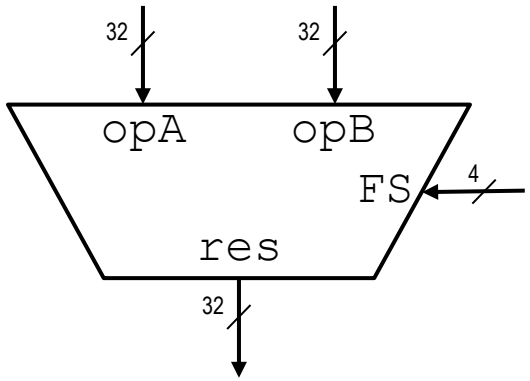
Function Unit

Hardware circuits are fixed

⇒ can't adjust wires / gates while running

⇒ build control wires to parametrize its function

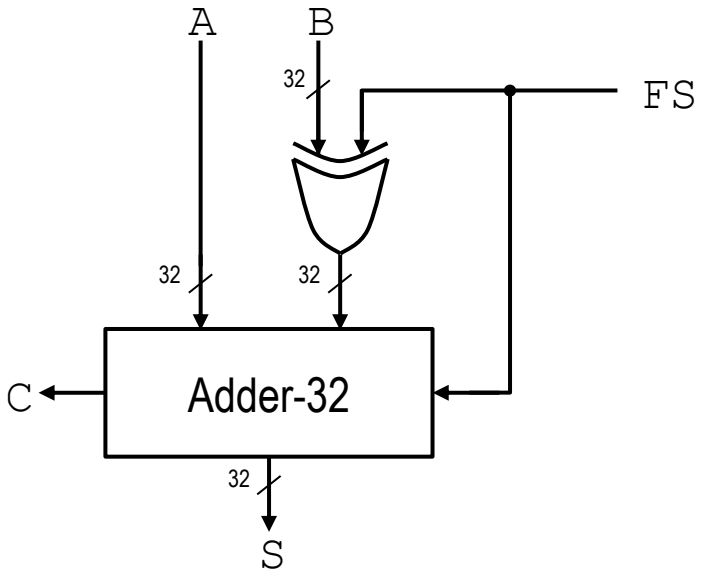
Function Unit:



Function Select:

FS	func
0001	A + B
0010	A − B
1000	A * B
0100	A ^ B
0101	A + 1
1101	B

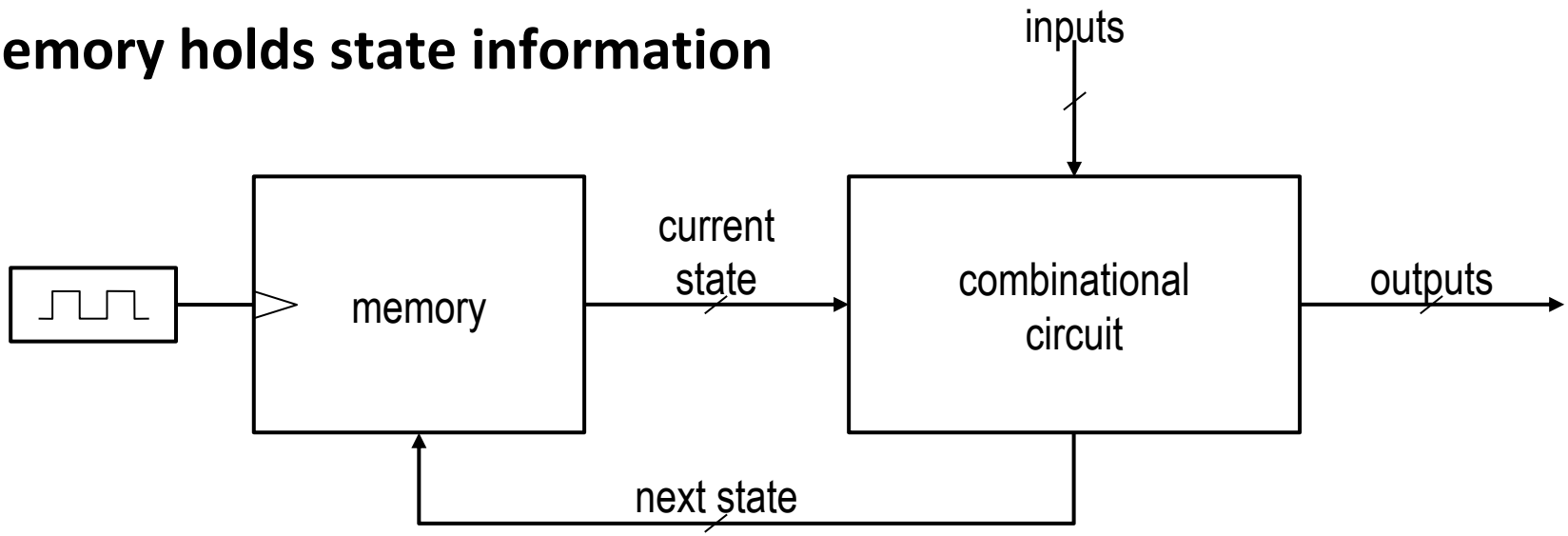
Function Unit: Adder-Subtractor



- if `FS == 0` then
  `S = A + B`
- if `FS == 1` then
  `S = A + B̄ + 1`
  `= A − B`

Digital State Machines

Memory holds state information



- **compute next state based on (current state, inputs)**
- **compute outputs based on (current state, inputs)**

Q. What does this imply about the clock period?

- **clock period must exceed (t_{pd} of combinational circuit + t_{pd} of registers)**

CPU Hardware

Goal: Given an instruction set architecture, construct a machine that reliably executes instructions.

Design choices will influence speed of instructions:

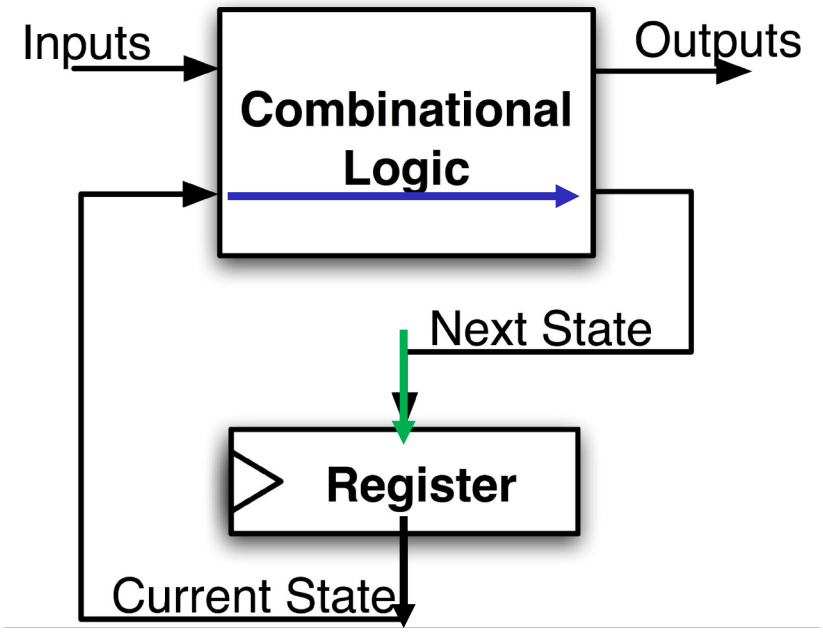
- **some instructions will be faster than others**
- **order of instructions may matter**
- **order of memory accesses may matter**

"conflicts" or "hazards"

Maximum Clock Frequency

- What is the max frequency of this circuit?
  - Limited by how much time needed to get correct Next State to Register (t_{setup} constraint)

Assumes Max Delay > Hold Time



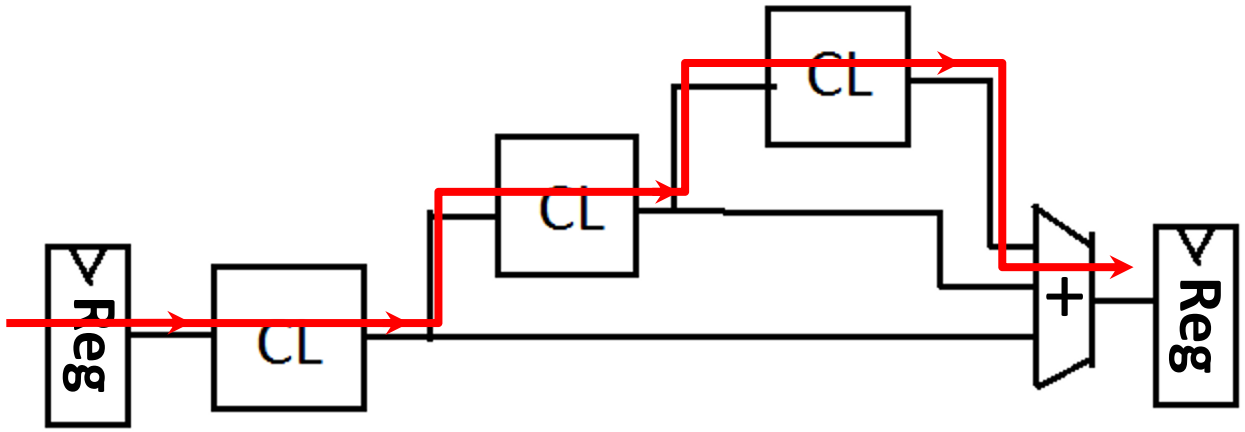
Max Delay = CLK-to-Q Delay
+ CL Delay
+ Setup Time

Min Period = Max Delay

Max Freq = 1/Min Period

The Critical Path

- The *critical path* is the longest delay between *any* two registers in a circuit
- The clock period must be *longer* than this critical path, or the signal will not propagate properly to that next register



How do we go faster?

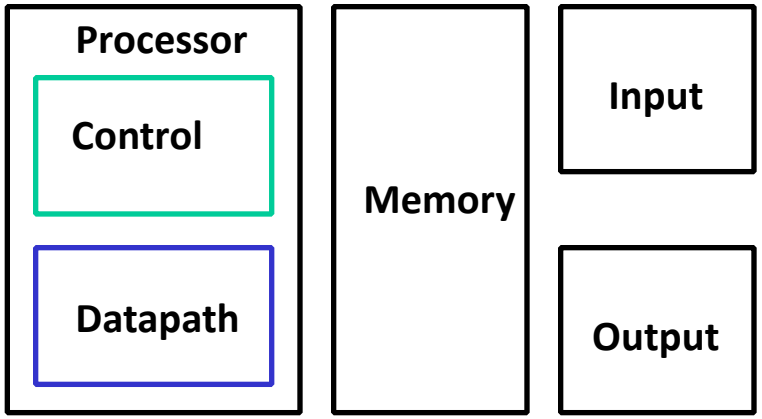
Pipelining!

- Split operation into smaller parts and add a register between each one.

RISC-V *CPU Datapath, Control Intro*

Design Principles

- Five 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



Summary !

- Universal datapath
  - Capable of executing all RISC-V instructions in one cycle each
  - Not all units (hardware) used by all instructions
- 5 Phases of execution
  - IF (Instruction Fetch), ID (Instruction Decode), EX (Execute), MEM (Memory), WB (Write Back)
  - Not all instructions are active in all phases (except for loads!)
- Controller specifies how to execute instructions
  - Worth thinking about: what new instructions can be added with just most control?

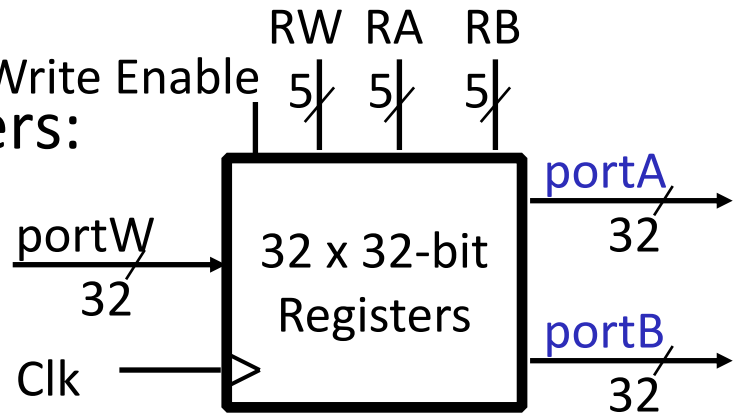
Your CPU in two parts

- ***Central Processing Unit (CPU):***
  - *Datapath:* contains the hardware necessary to perform operations required by the processor
    - Reacts to what the controller tells it! (ie. "I was told to do an add, so I"ll feed these arguments through an adder)
  - *Control:* decides what each piece of the datapath should do
    - What operation am I performing? Do I need to get info from memory? Should I write to a register? Which register?
    - Has to make decisions based on the input instruction only!

Design Principles

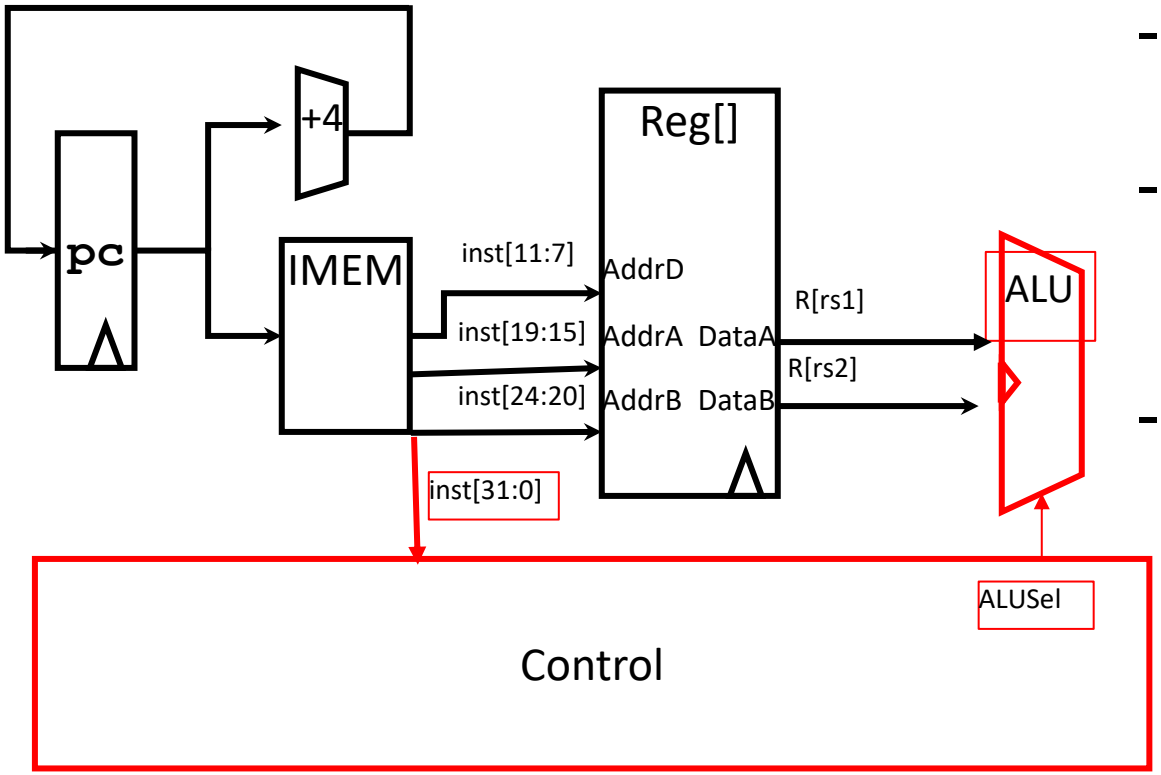
- Determining control signals
  - Any time a datapath element has an input that changes behavior, it requires a control signal (e.g. ALU operation, read/write)
  - Any time you need to pass a different input based on the instruction, add a **MUX** with a control signal as the selector (e.g. next PC, ALU input, register to write to)
- Your control signals will change based on your exact datapath
- Your datapath will change based on your ISA

Storage Element: Register File



- *Register File* consists of 31 registers:
  - Output ports portA and portB
  - Input port portW
- Register selection
  - Place data of register RA (number) onto portA
  - Place data of register RB (number) onto portB
  - Store data on portW into register RW (number) when Write Enable is 1
- Clock input (CLK)
  - CLK is passed to all internal registers so they can be written to if they match RW and Write Enable is 1

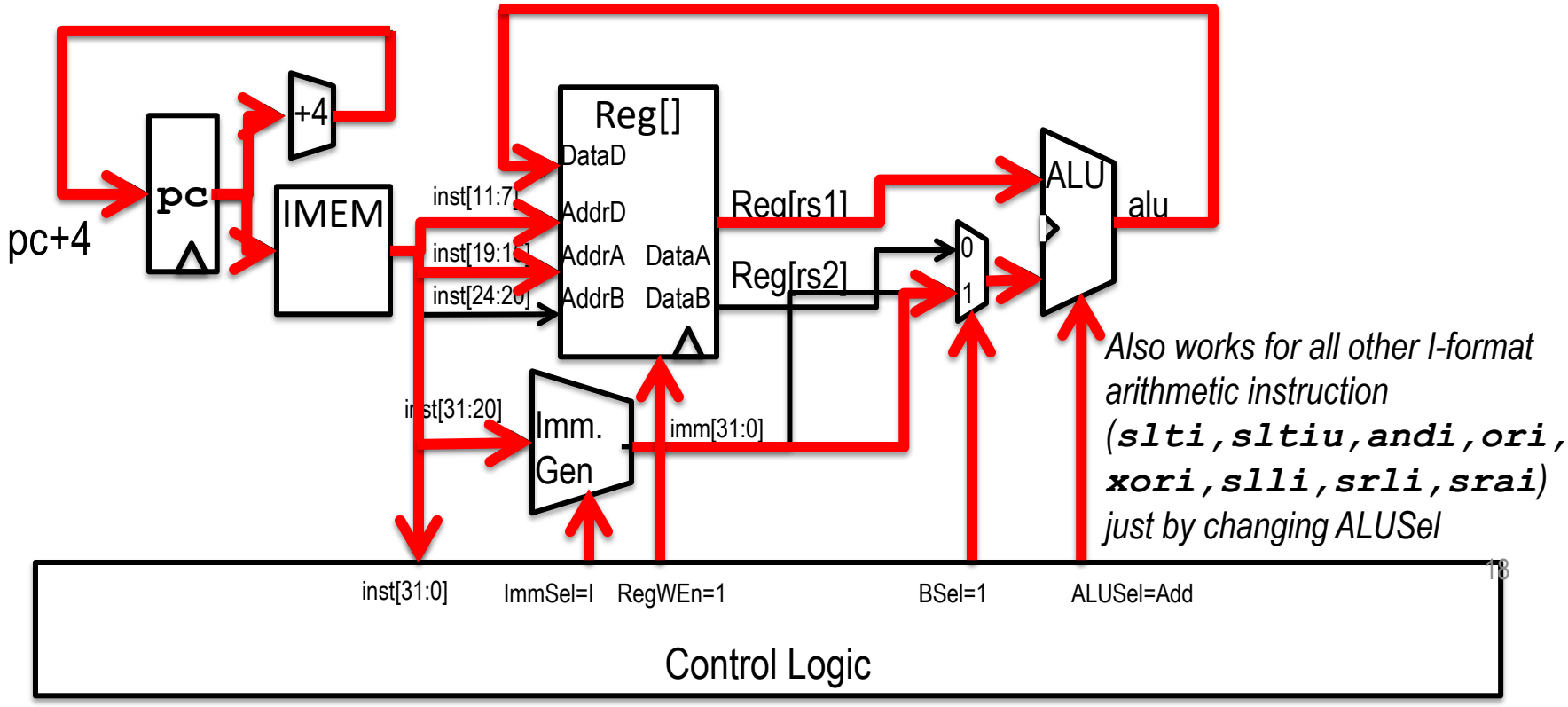
Implementing R-Types



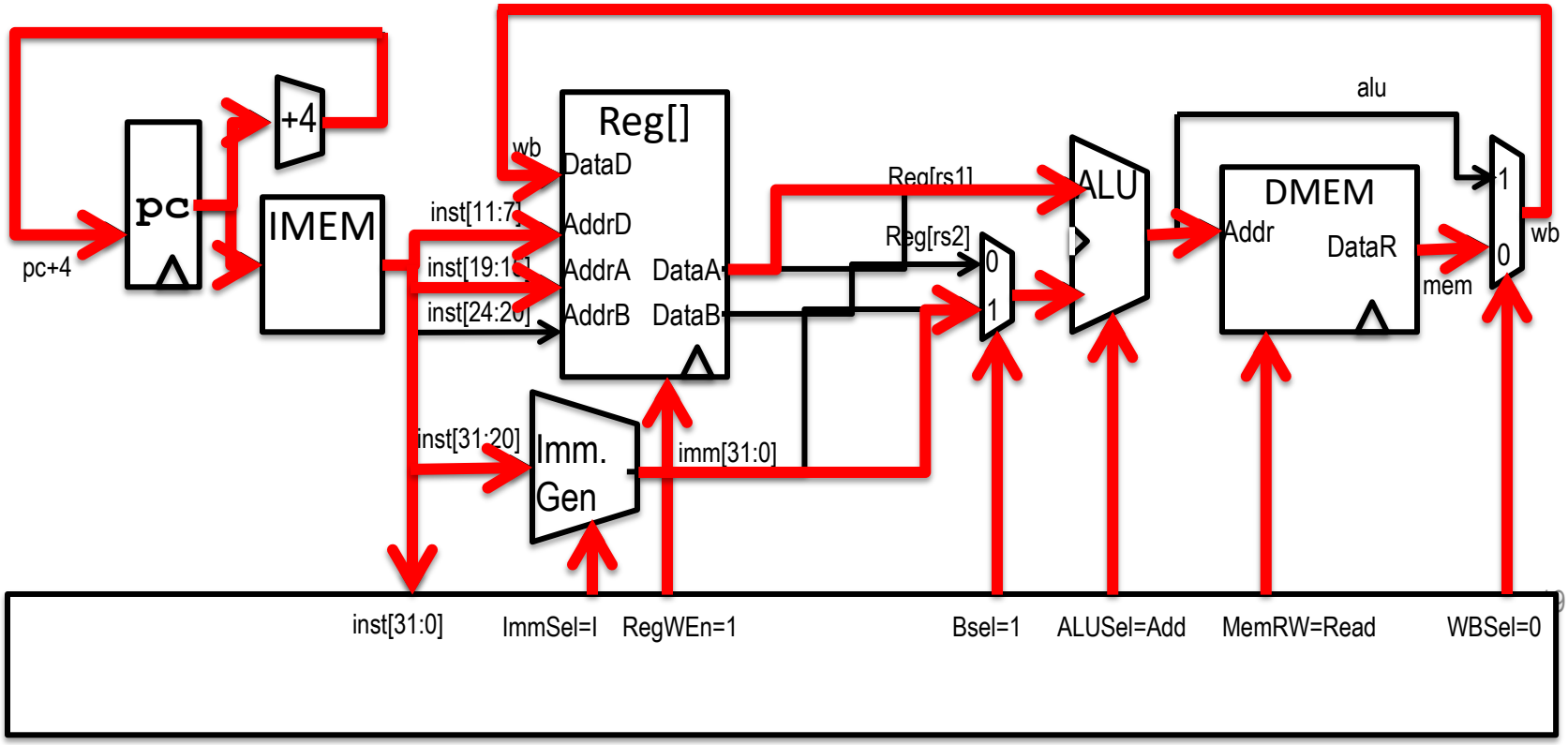
(4) Perform operation

- New hardware: ALU (Arithmetic Logic Unit)
- Abstraction for adders, multipliers, dividers, etc.
- How do we know what operation to execute?
  - Our first control bit! ALUSel(ect)

Adding `addi` to datapath

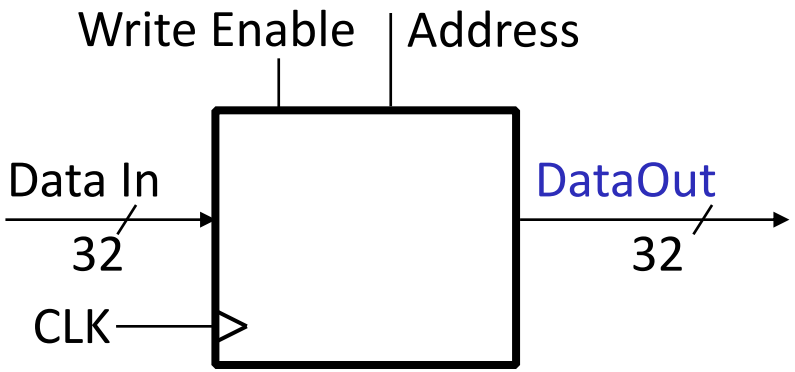


Adding `lw` to datapath

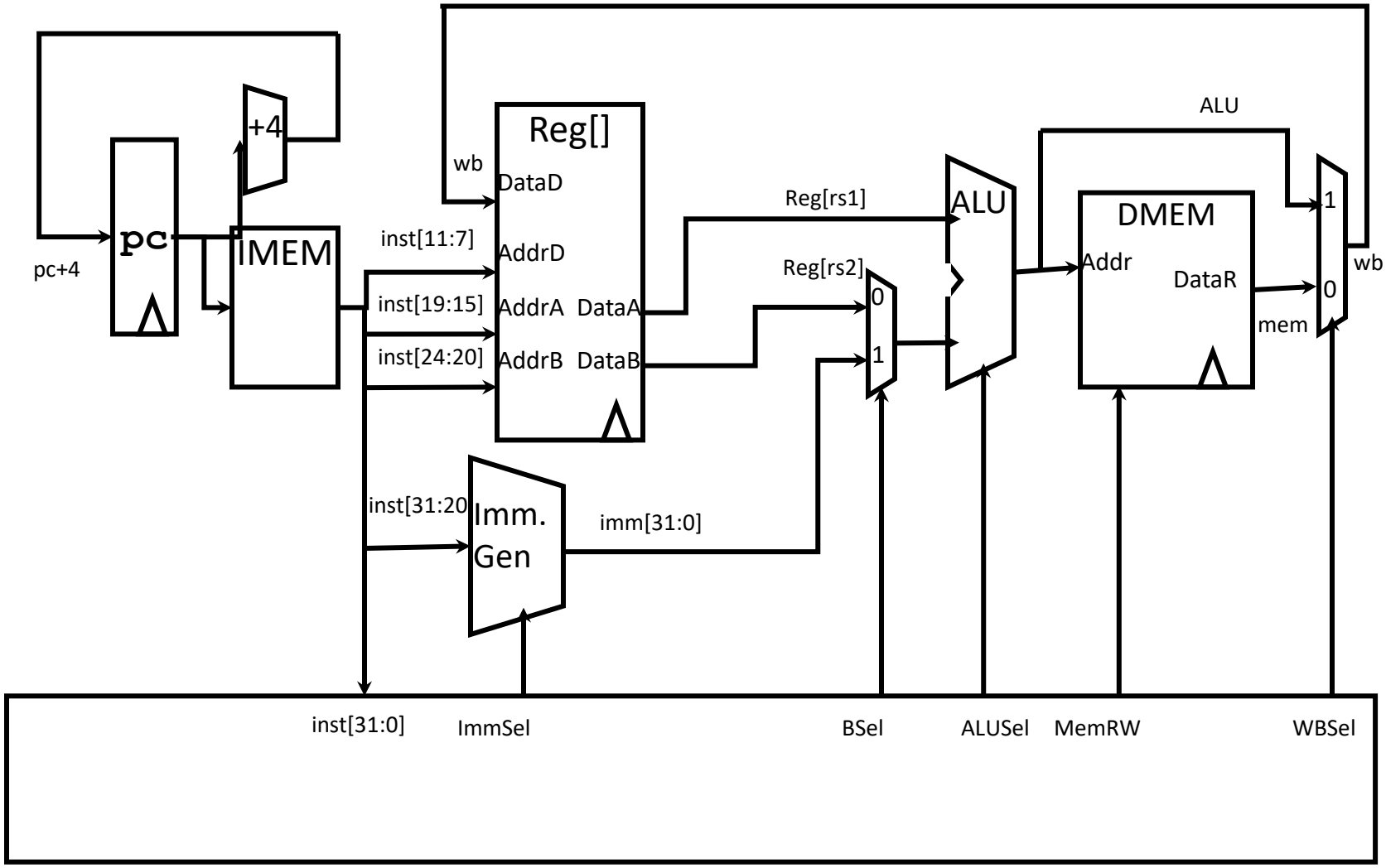


Storage Element: Idealized Memory

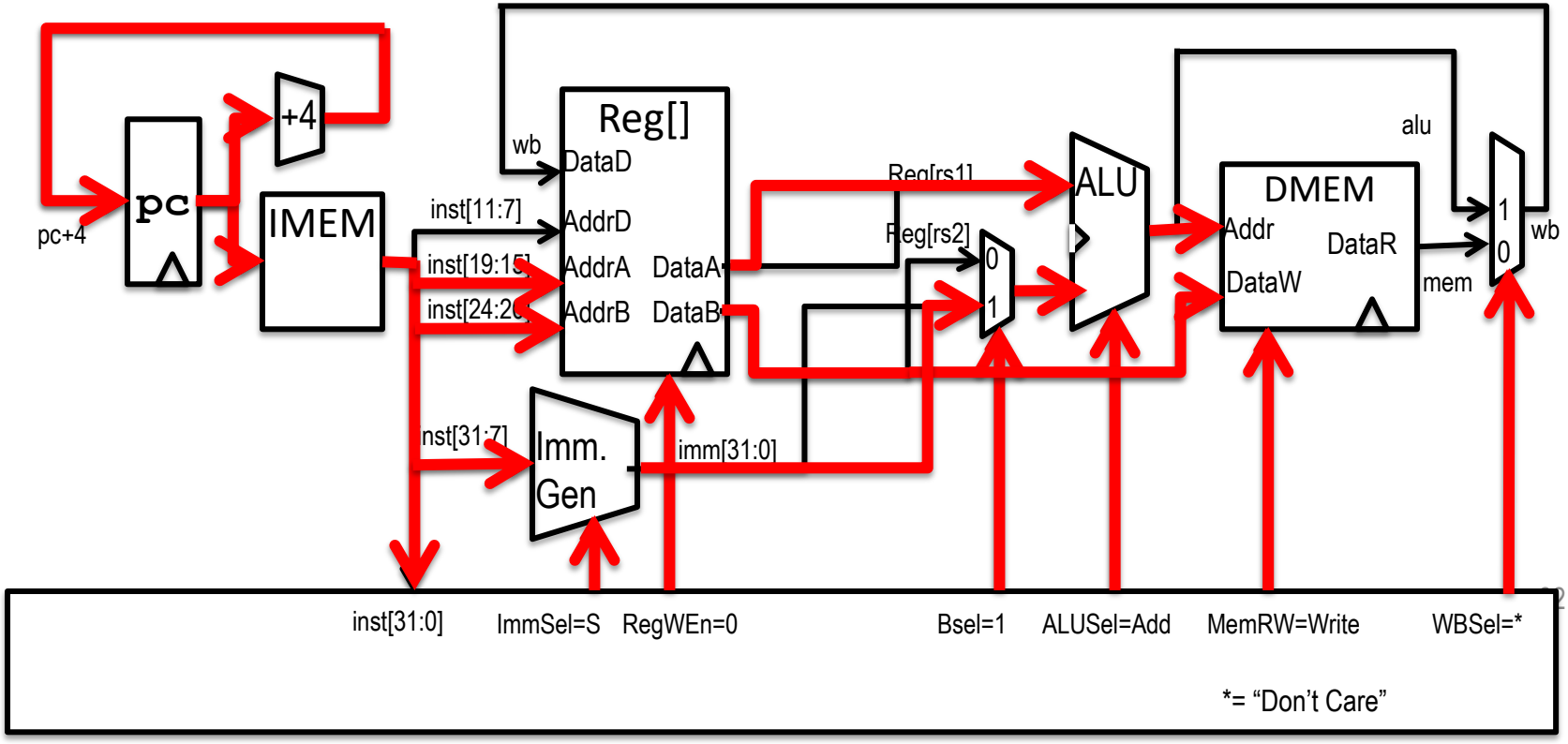
- Memory (idealized)
  - One input port: Data In
  - One output port: Data Out
- Memory access:
  - Read: Write Enable = 0, data at Address is placed on Data Out
  - Write: Write Enable = 1, Data In written to Address
- Clock input (CLK)
  - CLK input is a factor ONLY during write operation
  - During read, behaves as a combinational logic block: Address valid → Data Out valid after "access time"



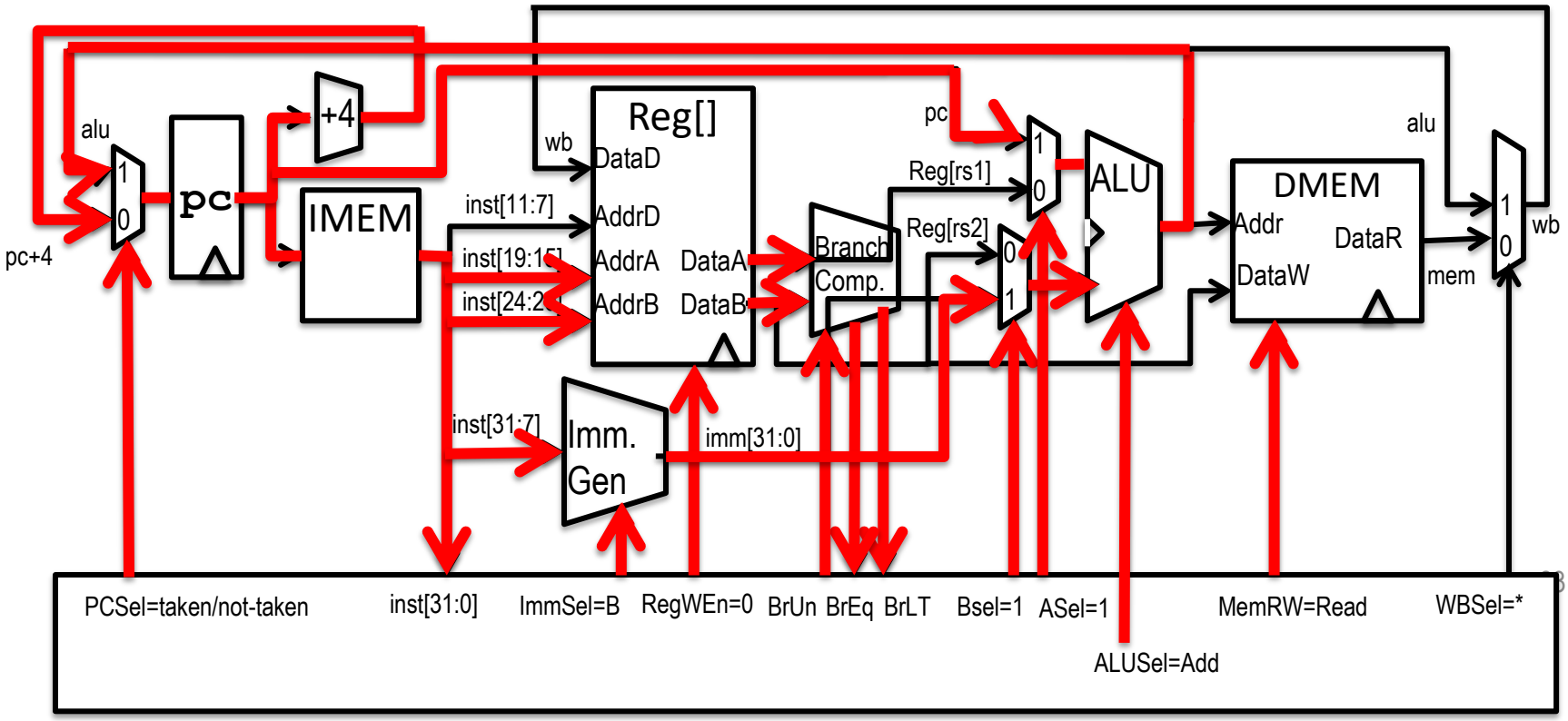
Current Datapath



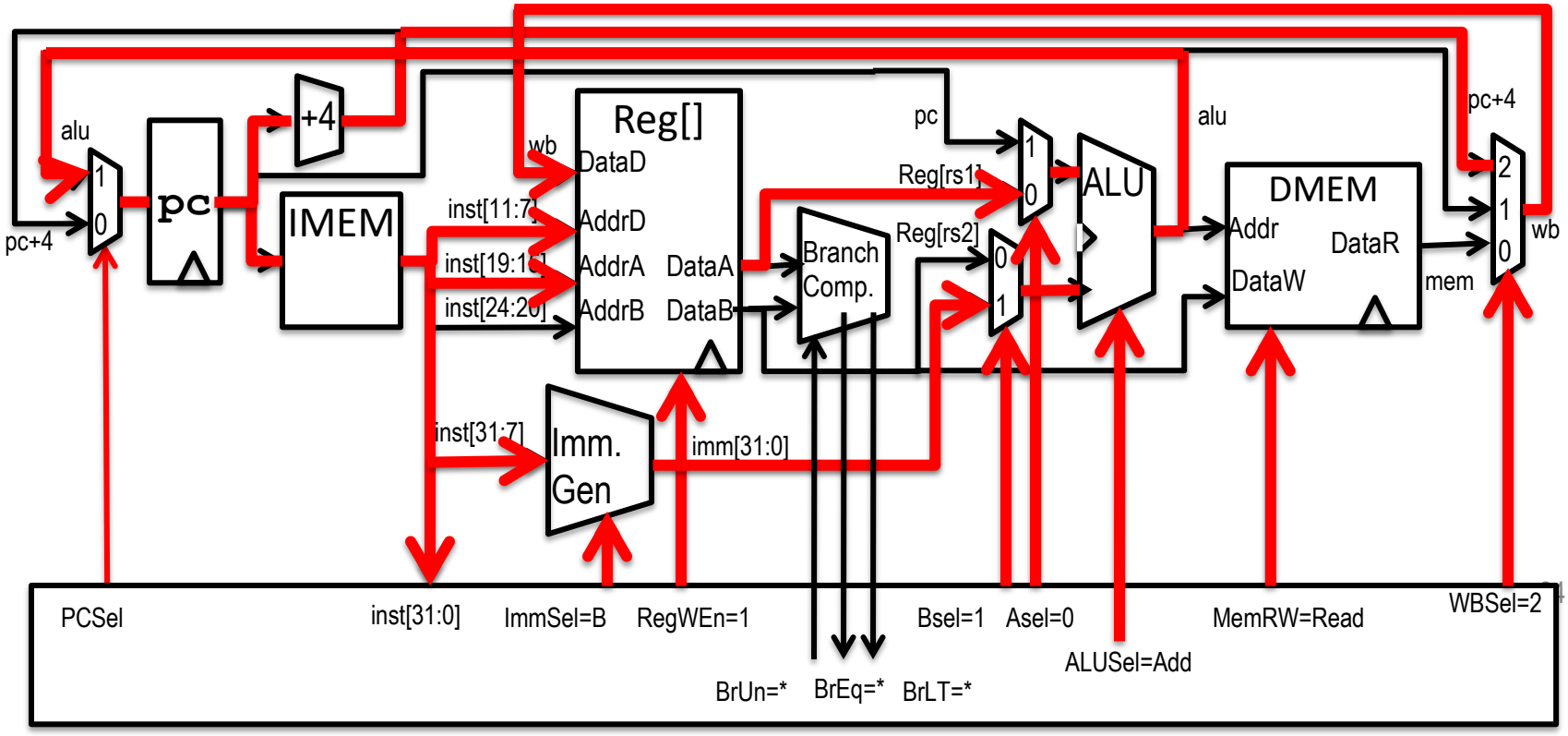
Adding sw to datapath



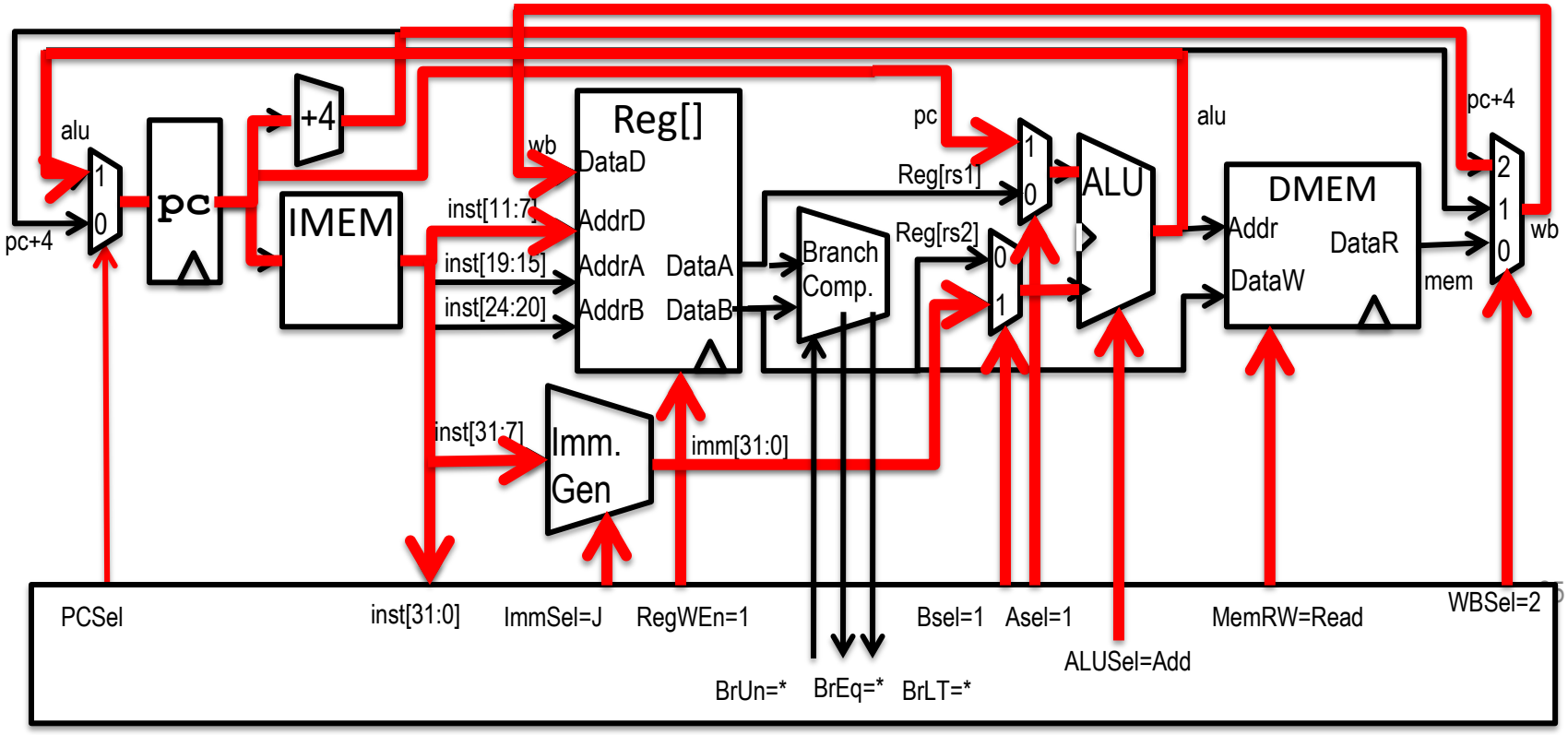
Adding branches to datapath



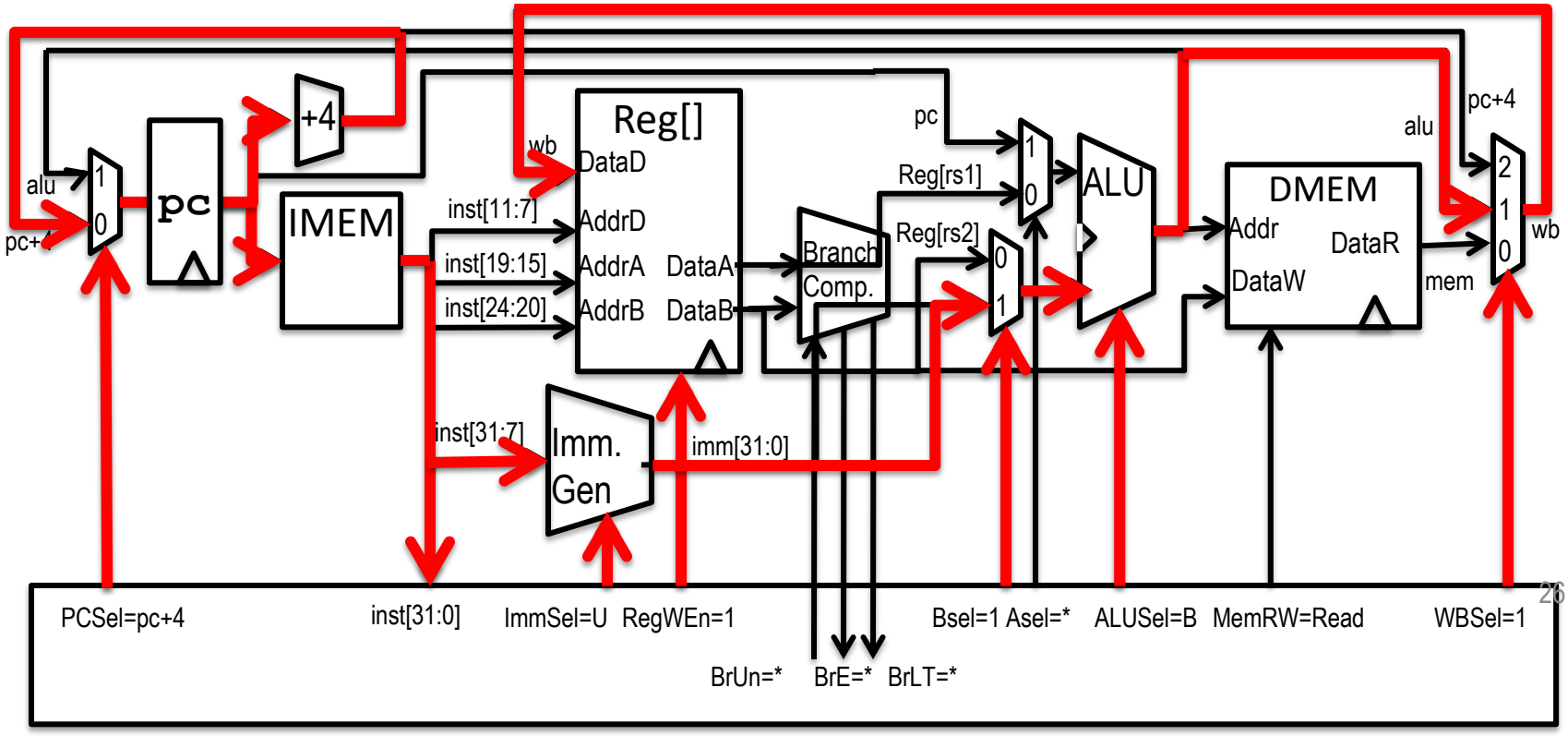
Adding `jalr` to datapath



Adding `jal` to datapath



Implementing `lui`



Implementing `auipc`