

Foundations of processor design: Single-cycle datapaths

EECS 370 – Introduction to Computer Organization – Winter 2015

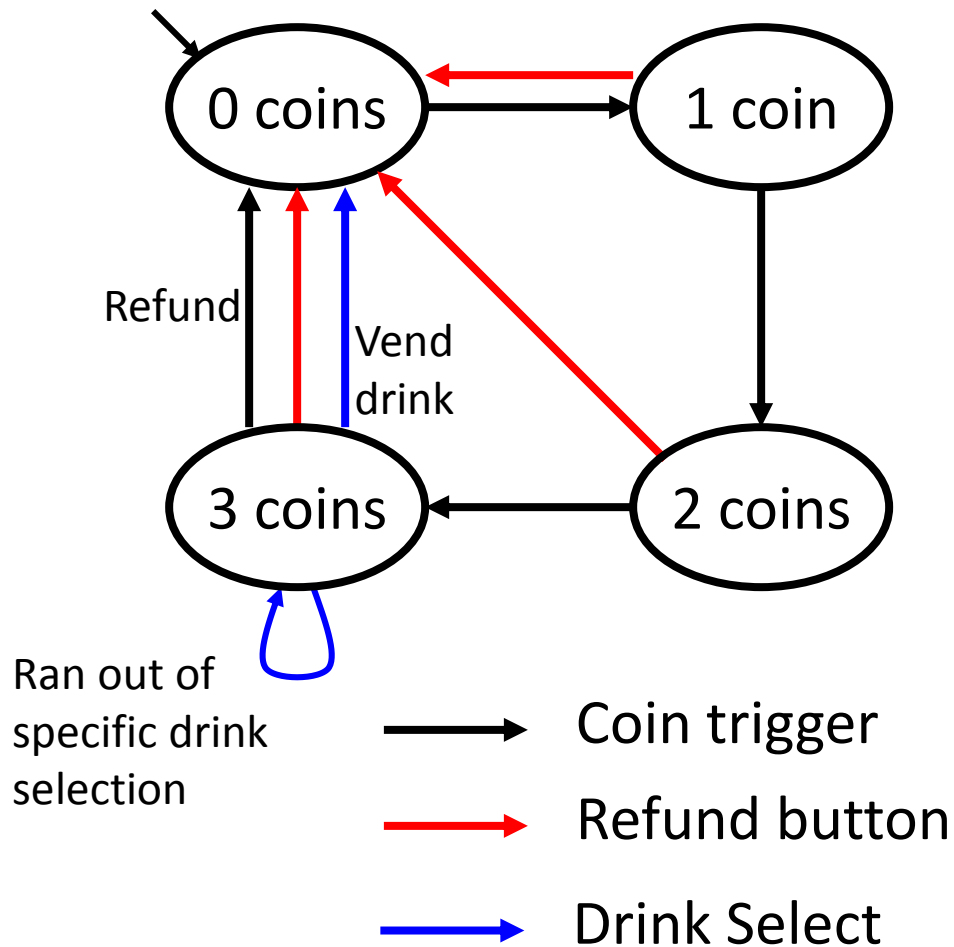
Robert Dick, Andrew Lukefahr, and Satish Narayanasamy

**EECS Department
University of Michigan in Ann Arbor, USA**

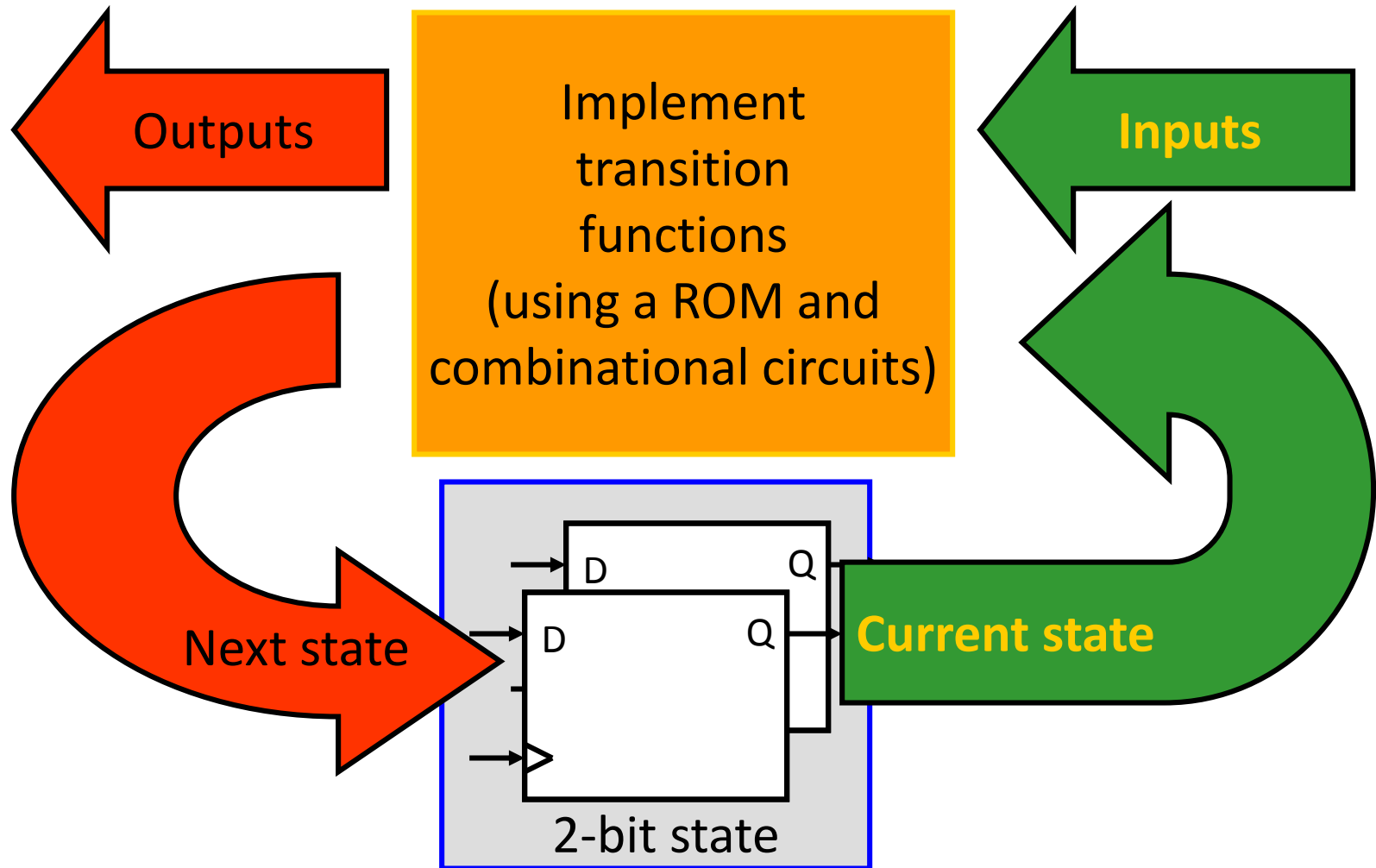
© Dick-Lukefahr-Narayanasamy, 2015

The material in this presentation cannot be
copied in any form without our written permission

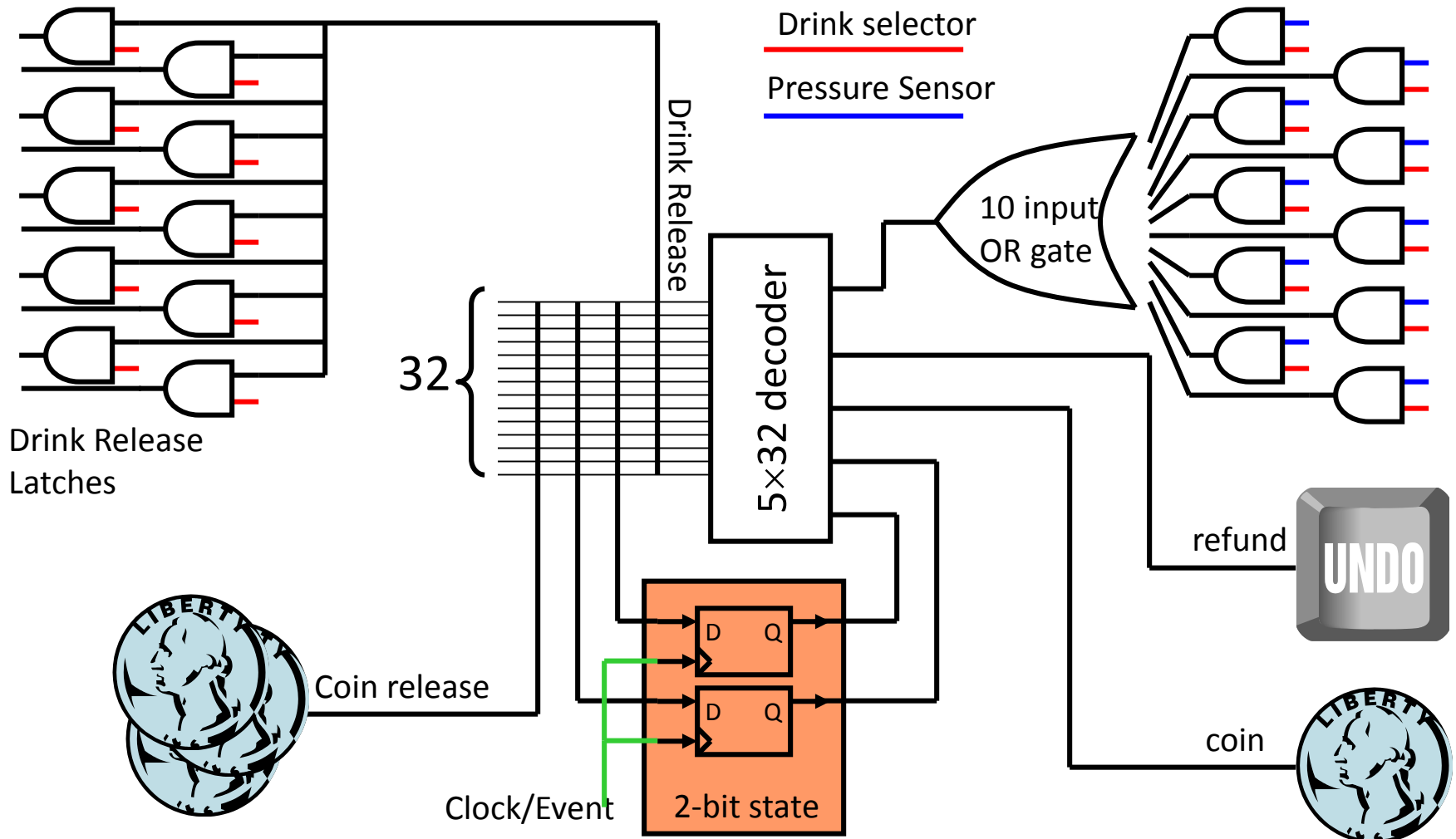
Review: FSM for Vending Machine



Review: implementing a FSM



Review: putting it all together

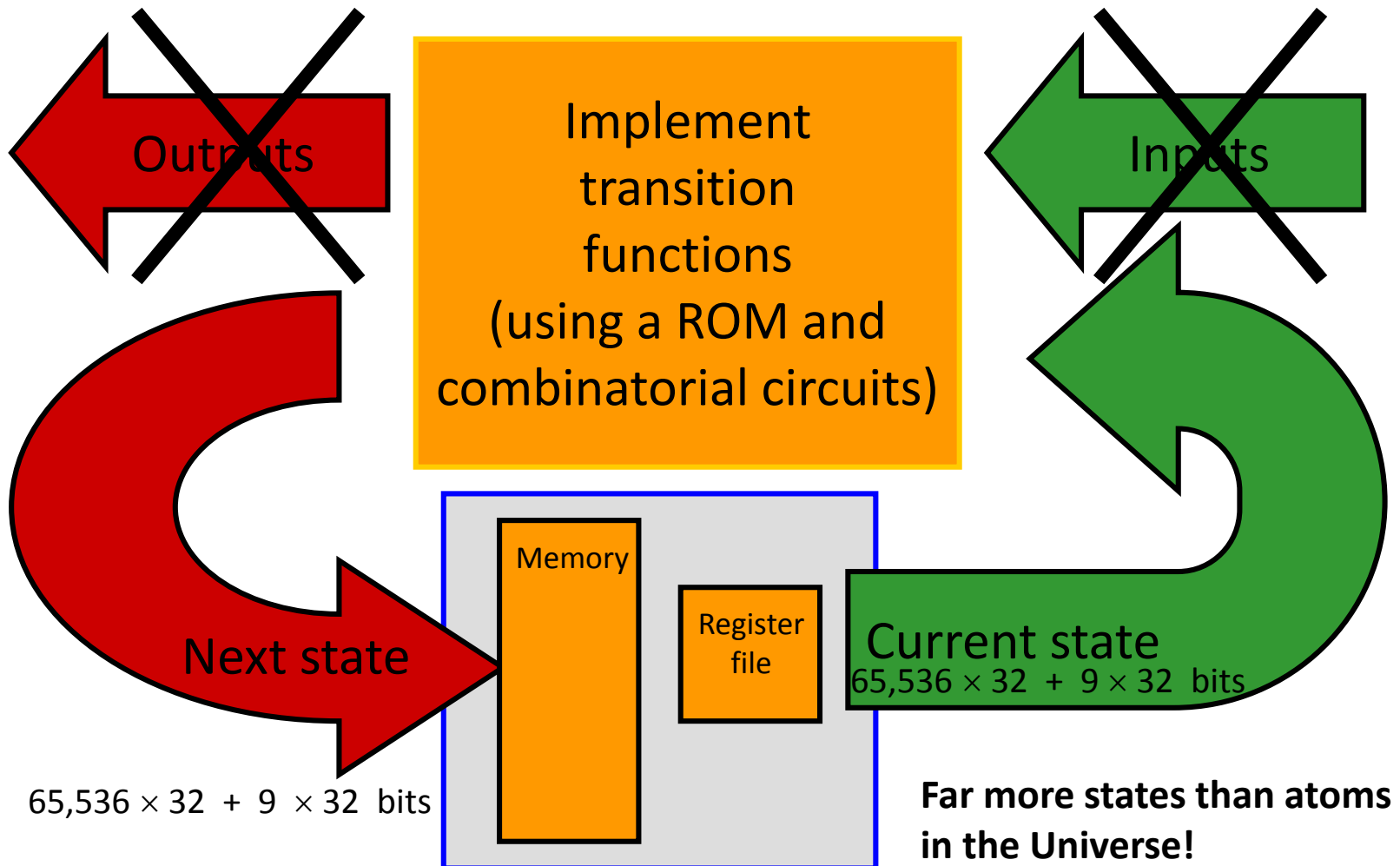


Single-Cycle Processor Design

□ General-Purpose Processor Design

- Fetch Instructions
- Decode Instructions
 - Instructions are input to control ROM
- ROM data controls movement of data
 - Incrementing PC, reading registers, ALU control
- Clock drives it all
- Single-cycle datapath: Each instruction completes in one clock cycle

LC2Kx Processor as FSM



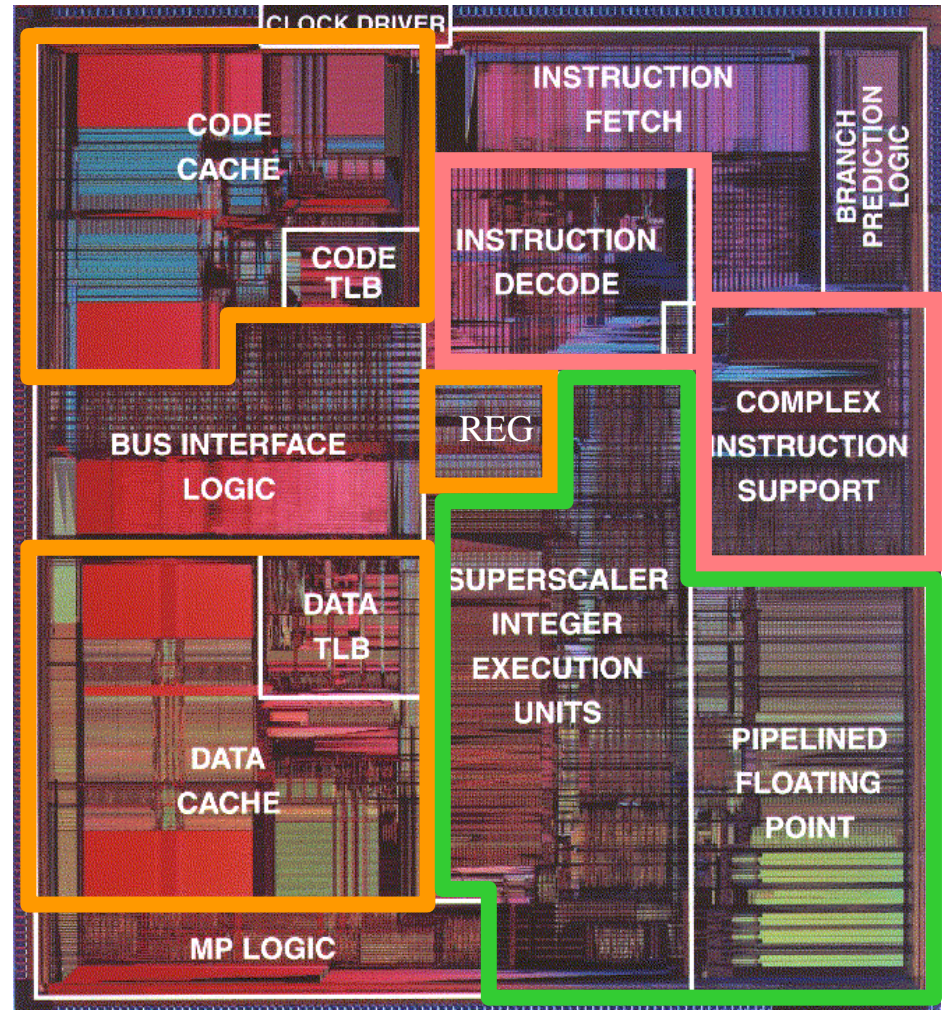
Pentium Processor Die

State

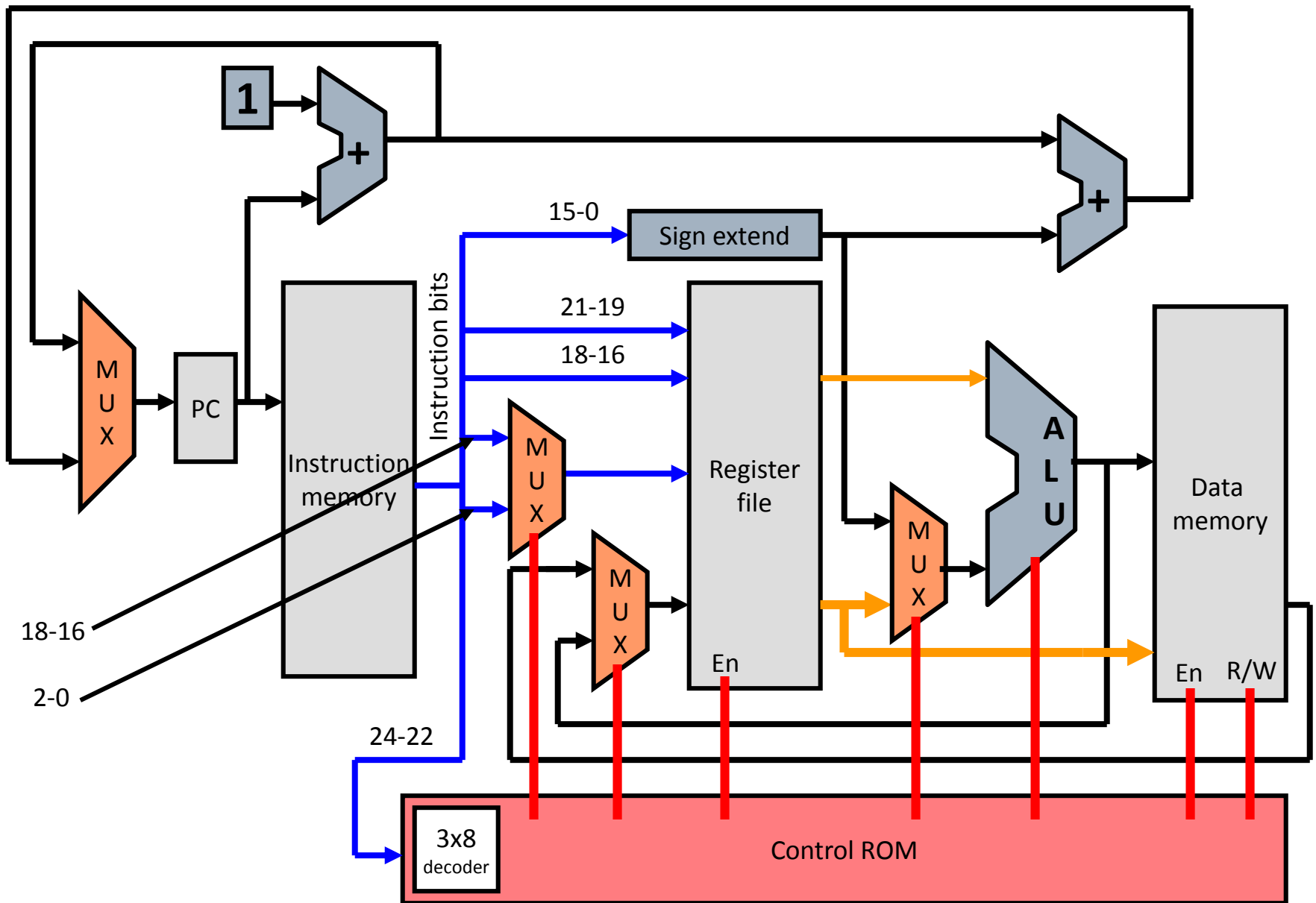
- Registers
- Memory

Control ROM

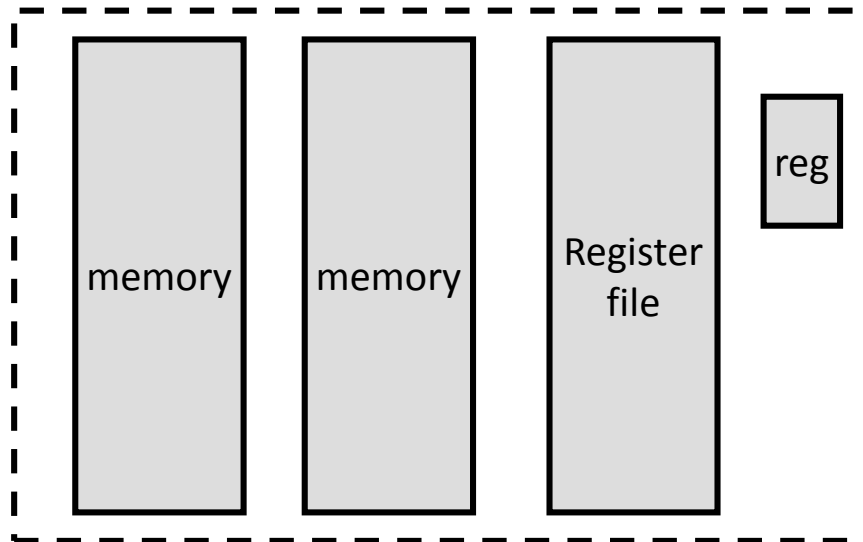
Combinational logic (Compute)



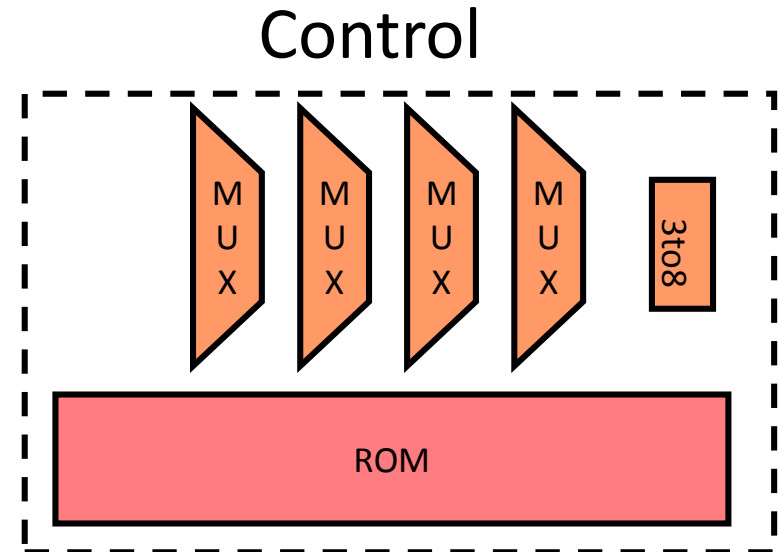
LC2Kx Datapath Implementation



Building Blocks for the LC2

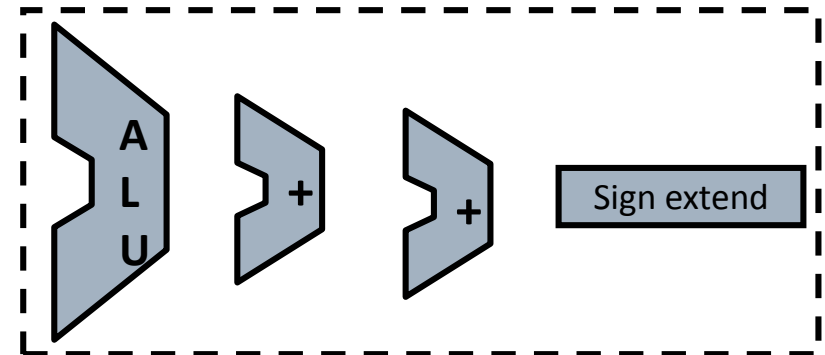


State



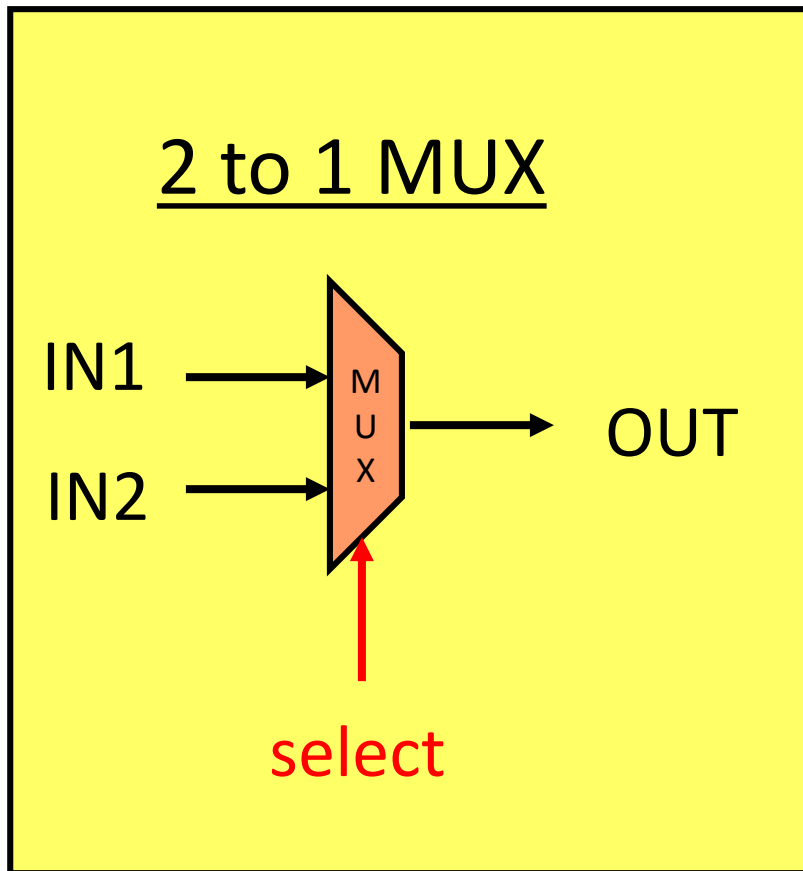
Control

Compute



Here are the pieces, go build yourself a processor!

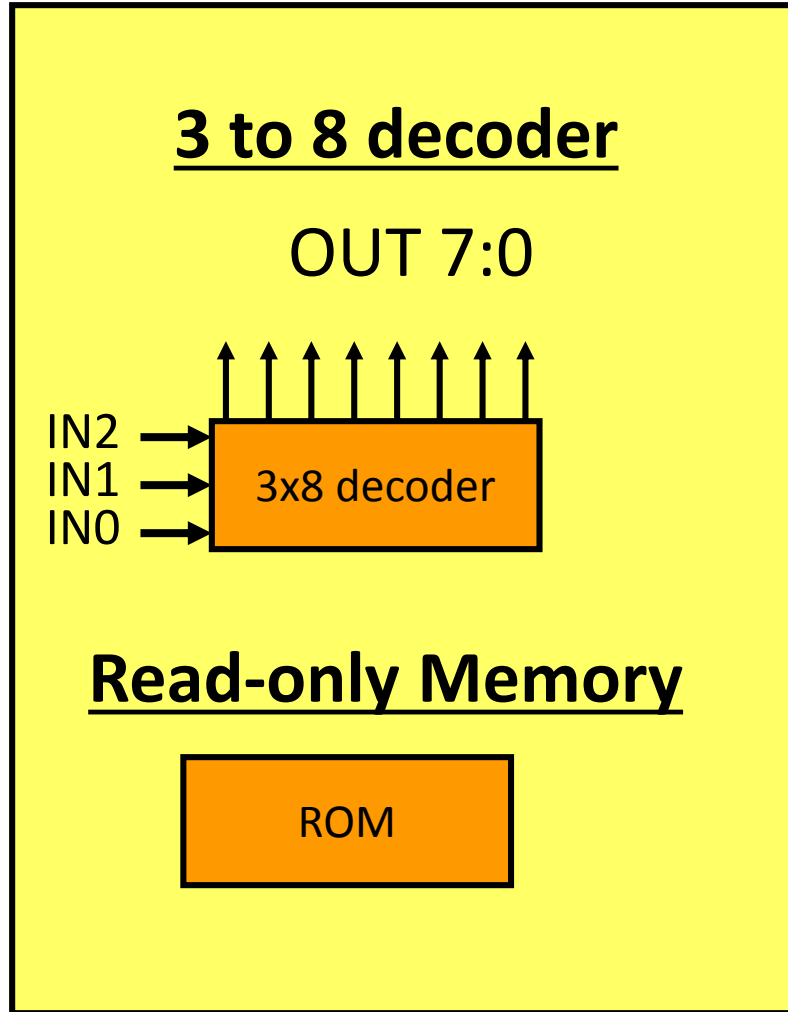
Control Building Blocks (1)



Connect one of the inputs to OUT based on the value of select

If (! select)
 OUT = IN1
Else
 OUT = IN2

Control Building Blocks (2)



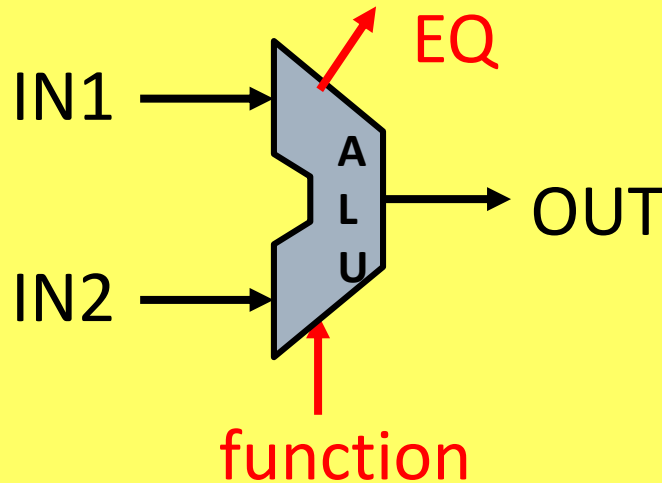
Decoder activates one of the output lines based on the input

IN	OUT
<u>210</u>	<u>76543210</u>
000	00000001
001	00000010
010	00000100
011	00001000
etc.	

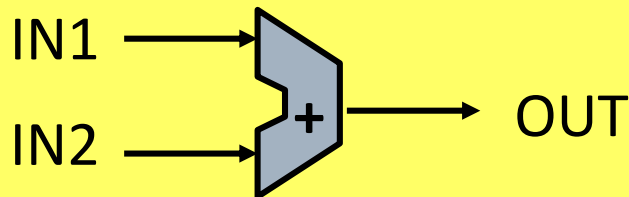
ROM stores preset data in each location.
Give address to access data.

Compute Building Blocks (1)

Arithmetic Logic Unit



Adder



Perform basic arithmetic functions

$$\text{OUT} = f(\text{IN1}, \text{IN2})$$

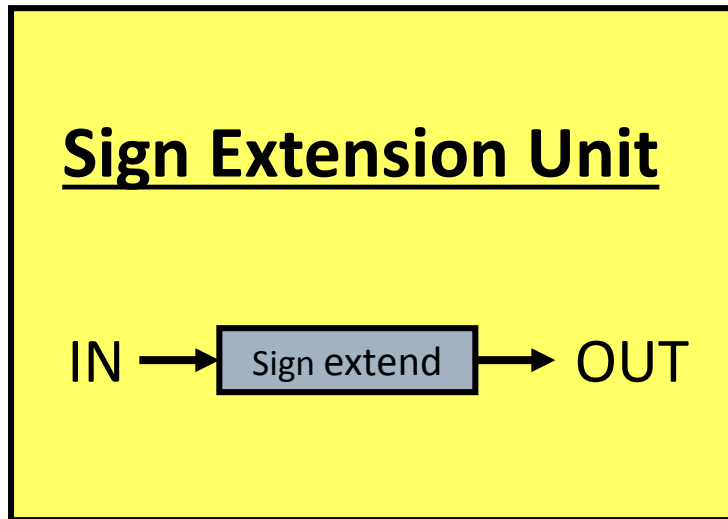
$$\text{EQ} = (\text{IN1} == \text{IN2})$$

For LC2, $f = \text{add, nand}$.

For other processors, there are many more functions.

For now, only adds

Compute Building Blocks (2)



Sign extend input by replicating the MSB to width of output

$$\text{OUT}(31:0) = \text{SE}(\text{IN}(15:0))$$

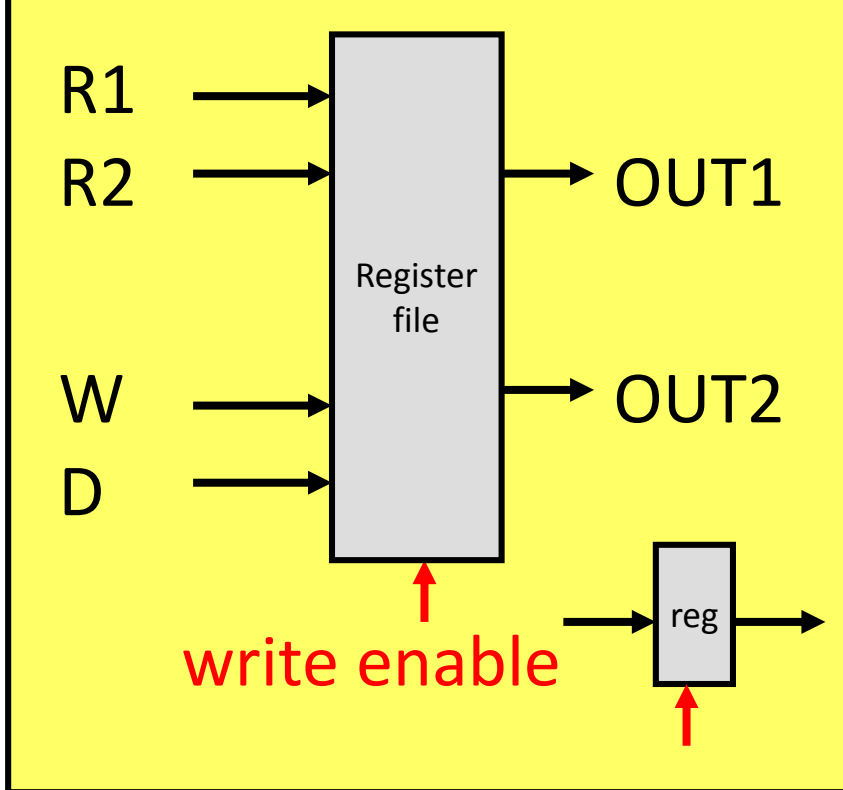
$$\text{OUT}(31:16) = \text{IN}(15)$$

$$\text{OUT}(15:0) = \text{IN}(15:0)$$

Useful when compute unit is wider than data

State Building Blocks (1)

Register File or Register



Small/fast memory to store temporary values

n entries (LC2 = 8)

r read ports (LC2 = 2)

w write ports (LC2 = 1)

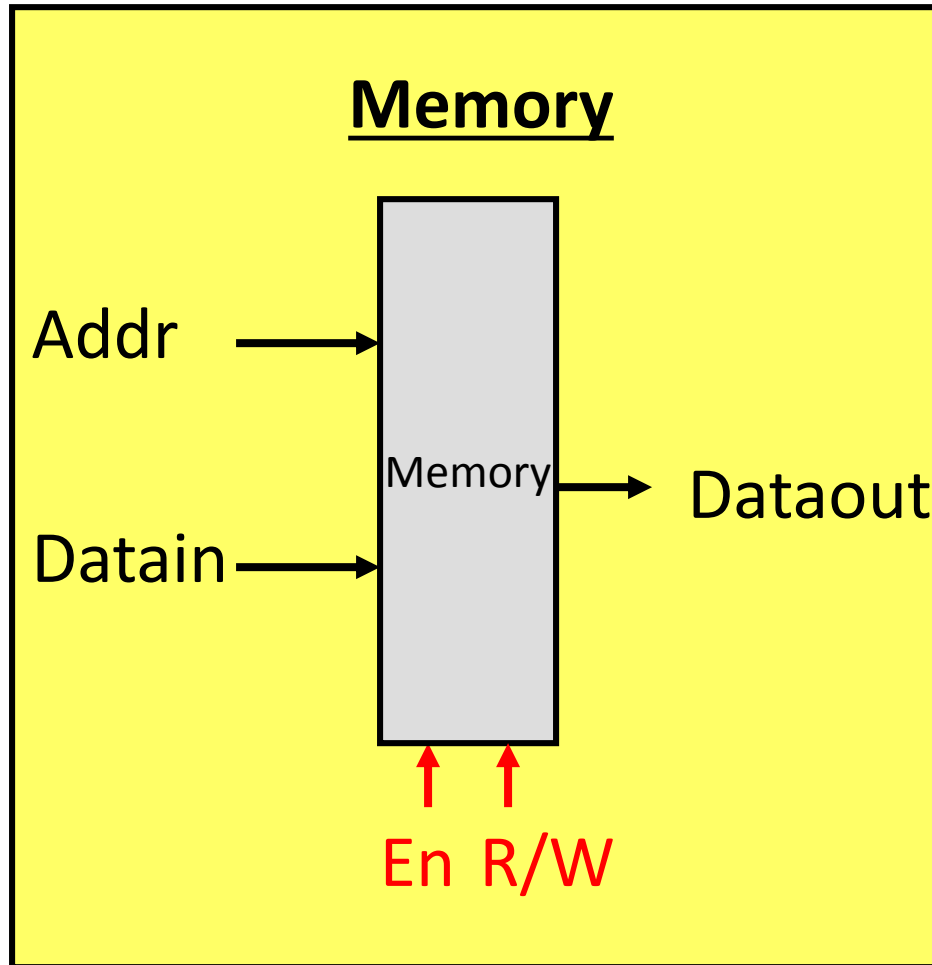
* R_i specifies register number to read

* W specifies register number to write

* D specifies data to write

How many bits are R_i and W_i in LC2?

State Building Blocks (2)

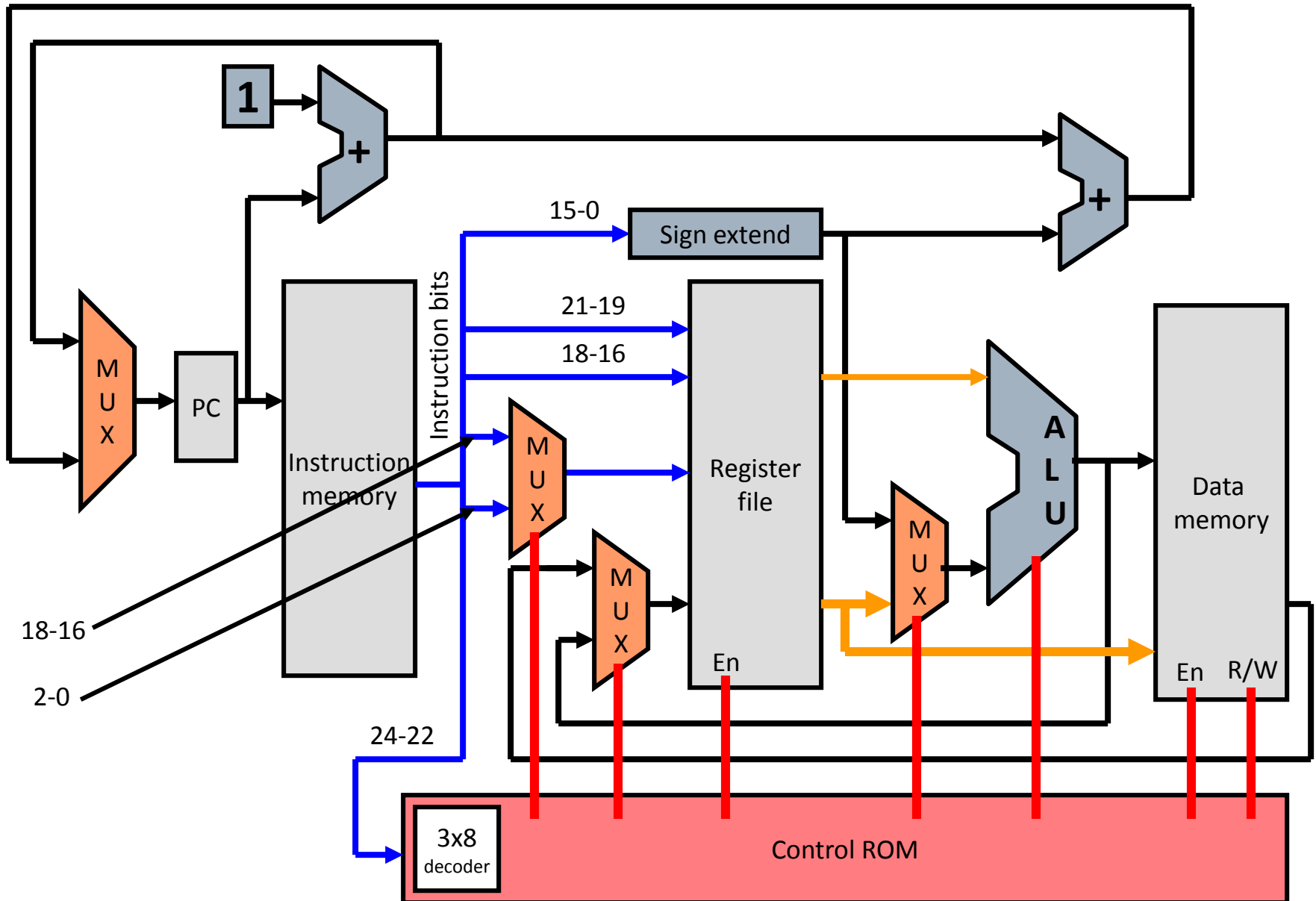


Slower storage structure to hold large amounts of stuff.

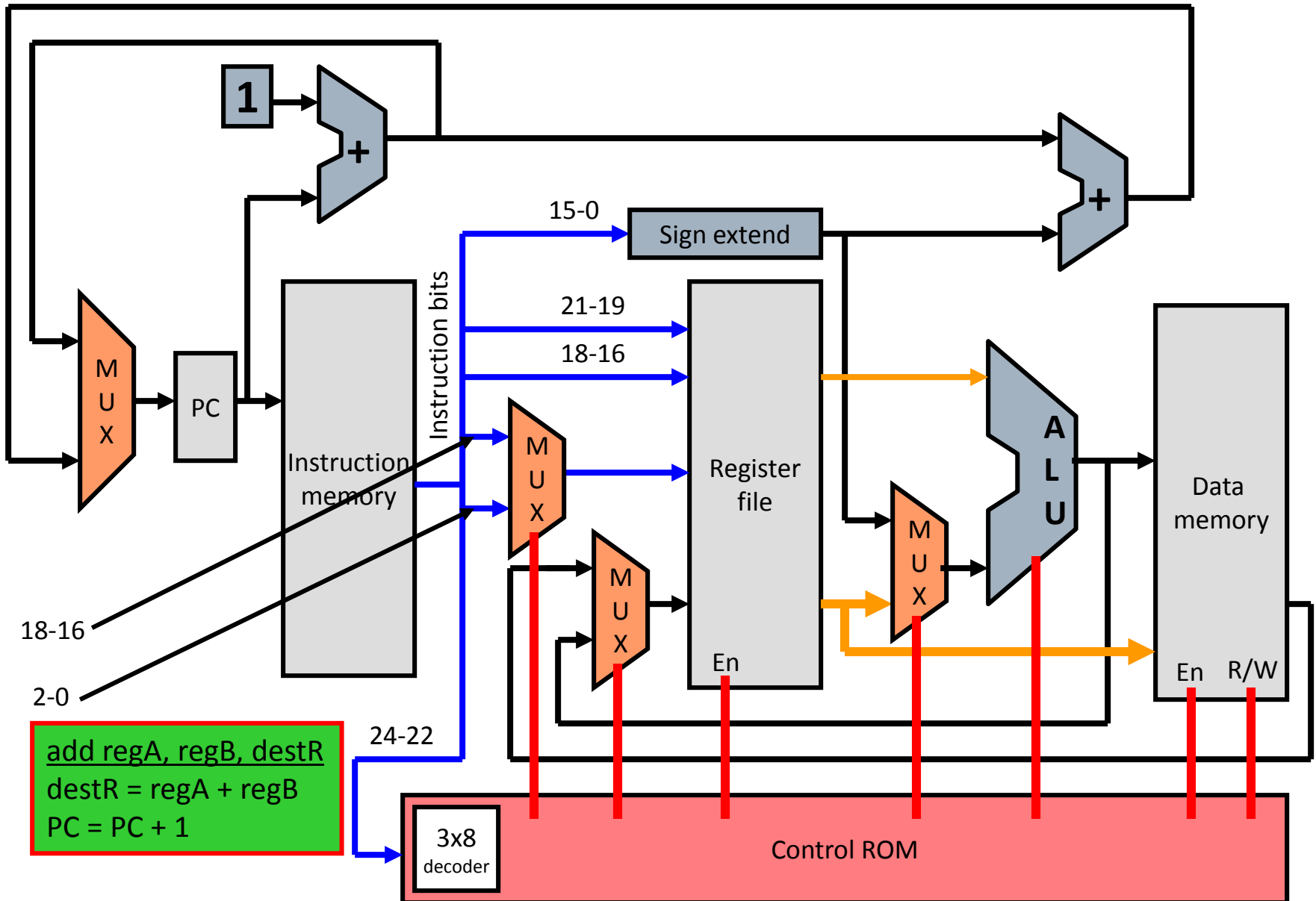
Use 2 memories for LC2

- * Instructions
- * Data
- * 65,536 total words

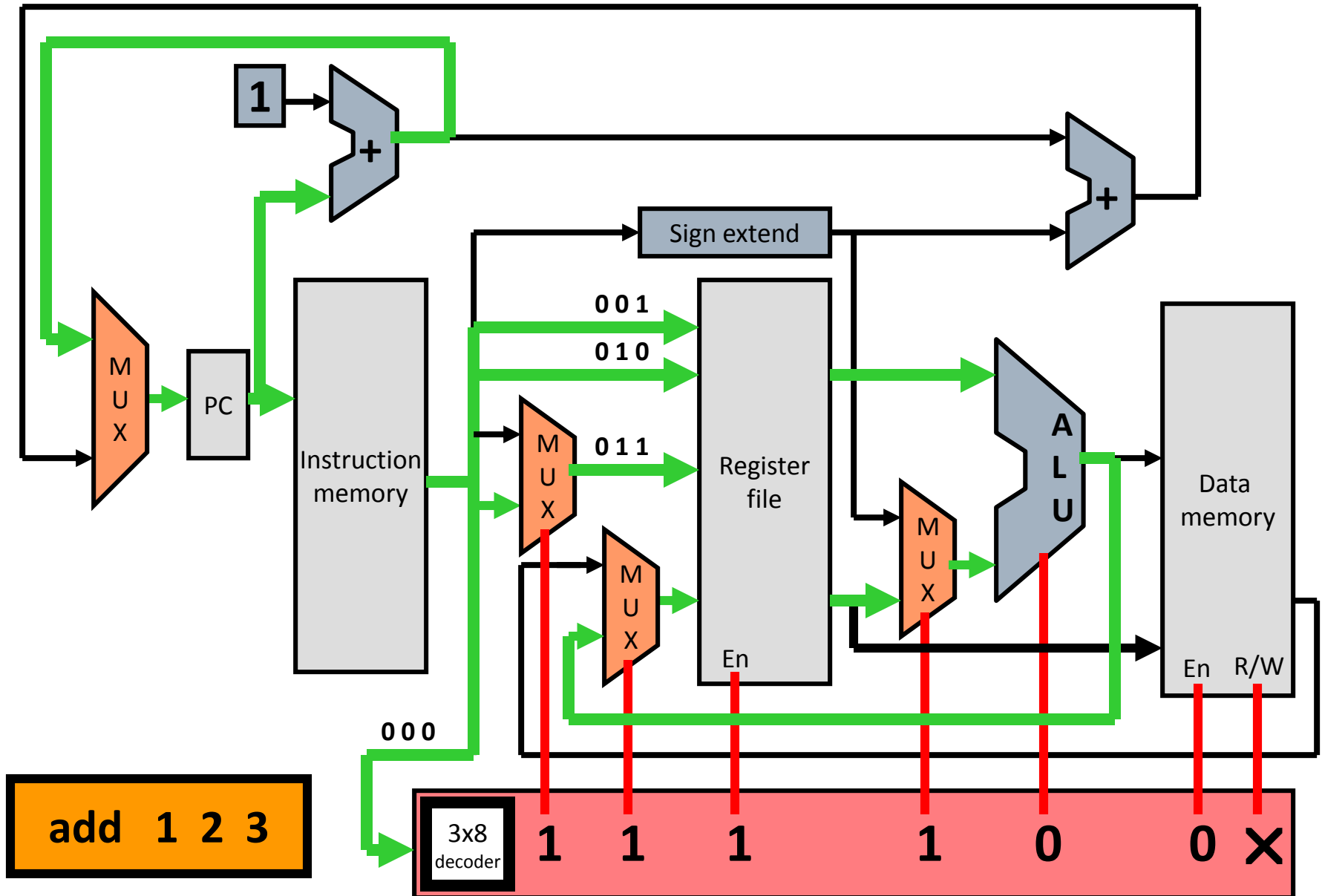
LC2Kx Datapath Implementation



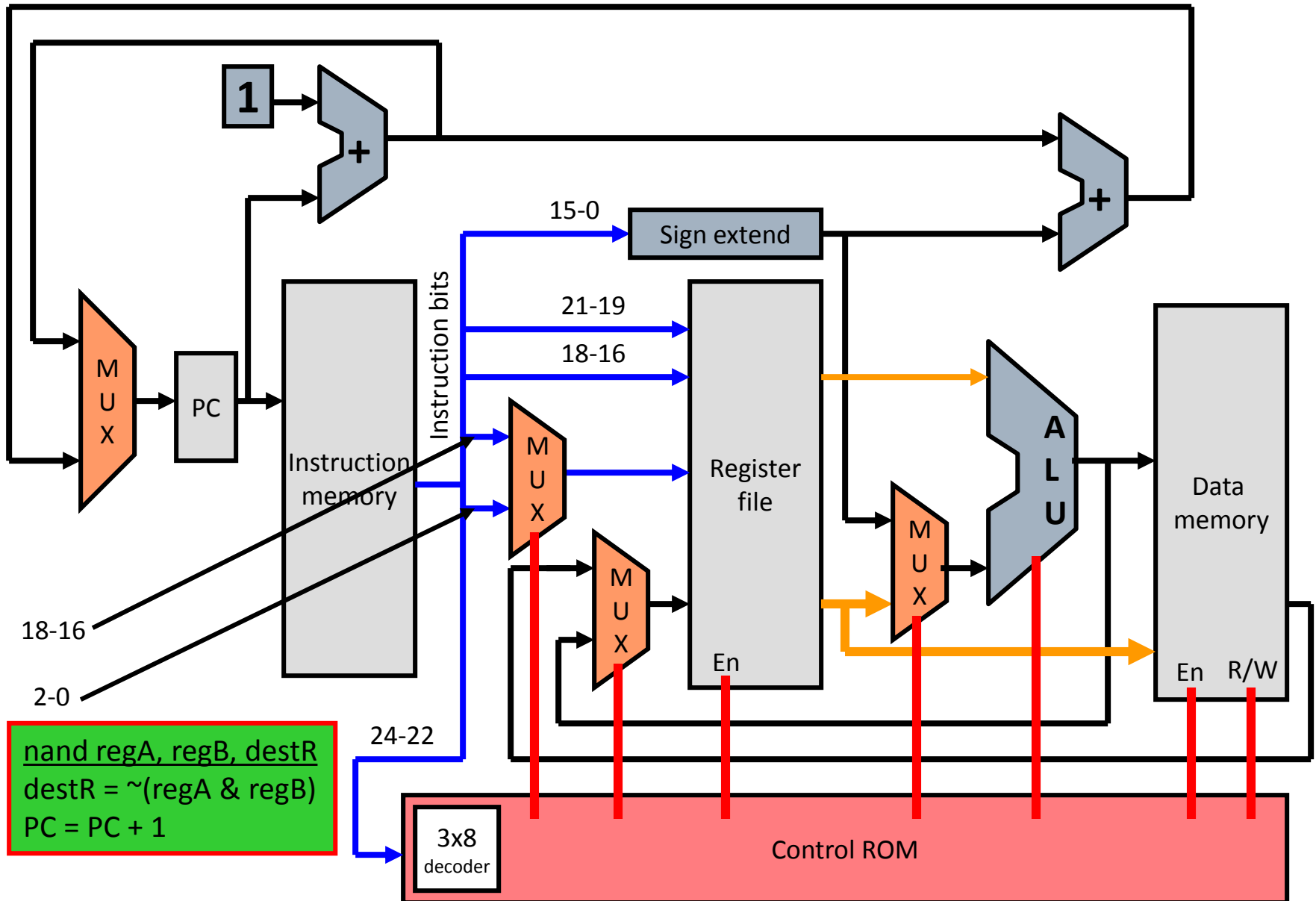
Executing an **ADD** Instruction



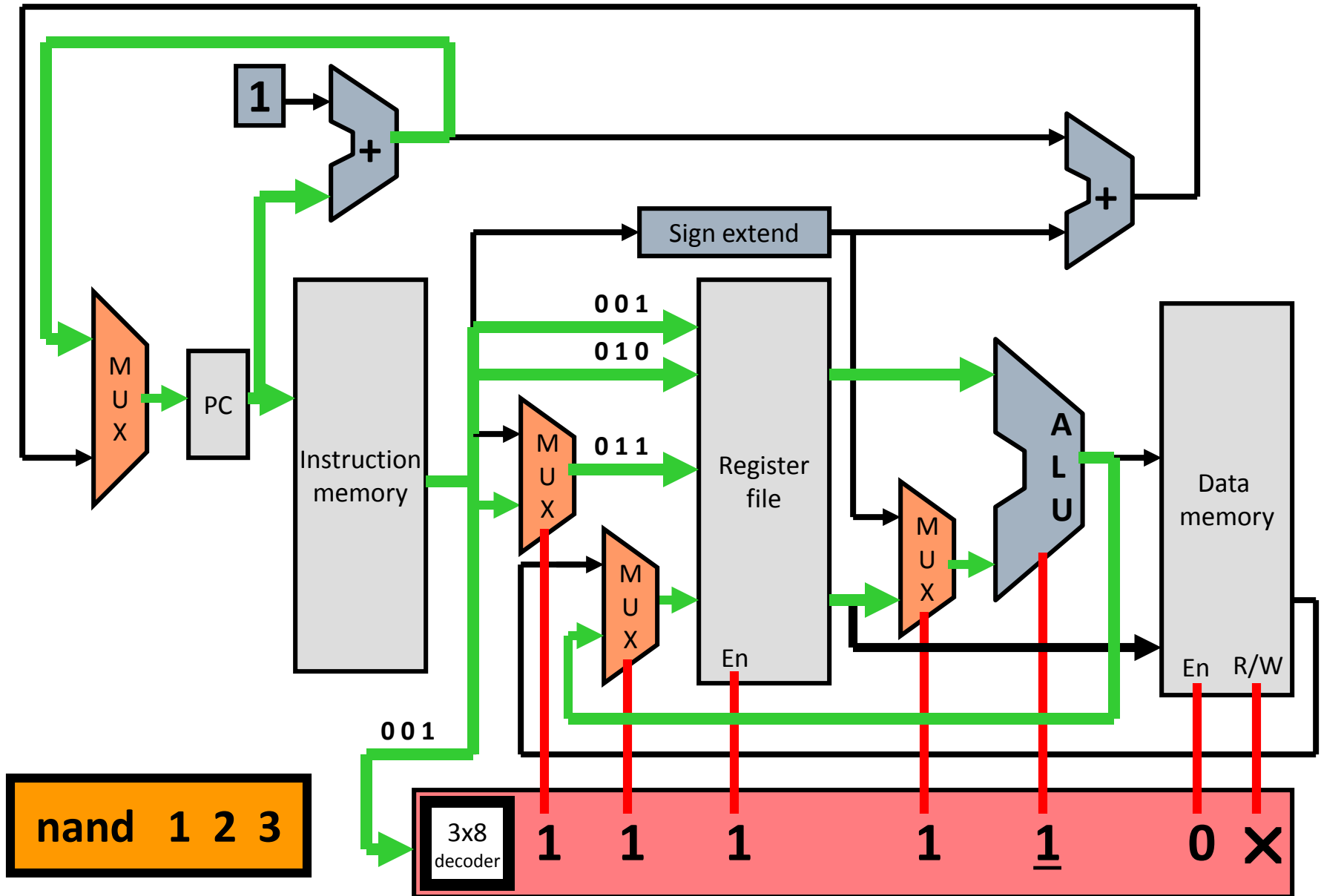
Executing an **ADD** Instruction on LC2Kx Datapath



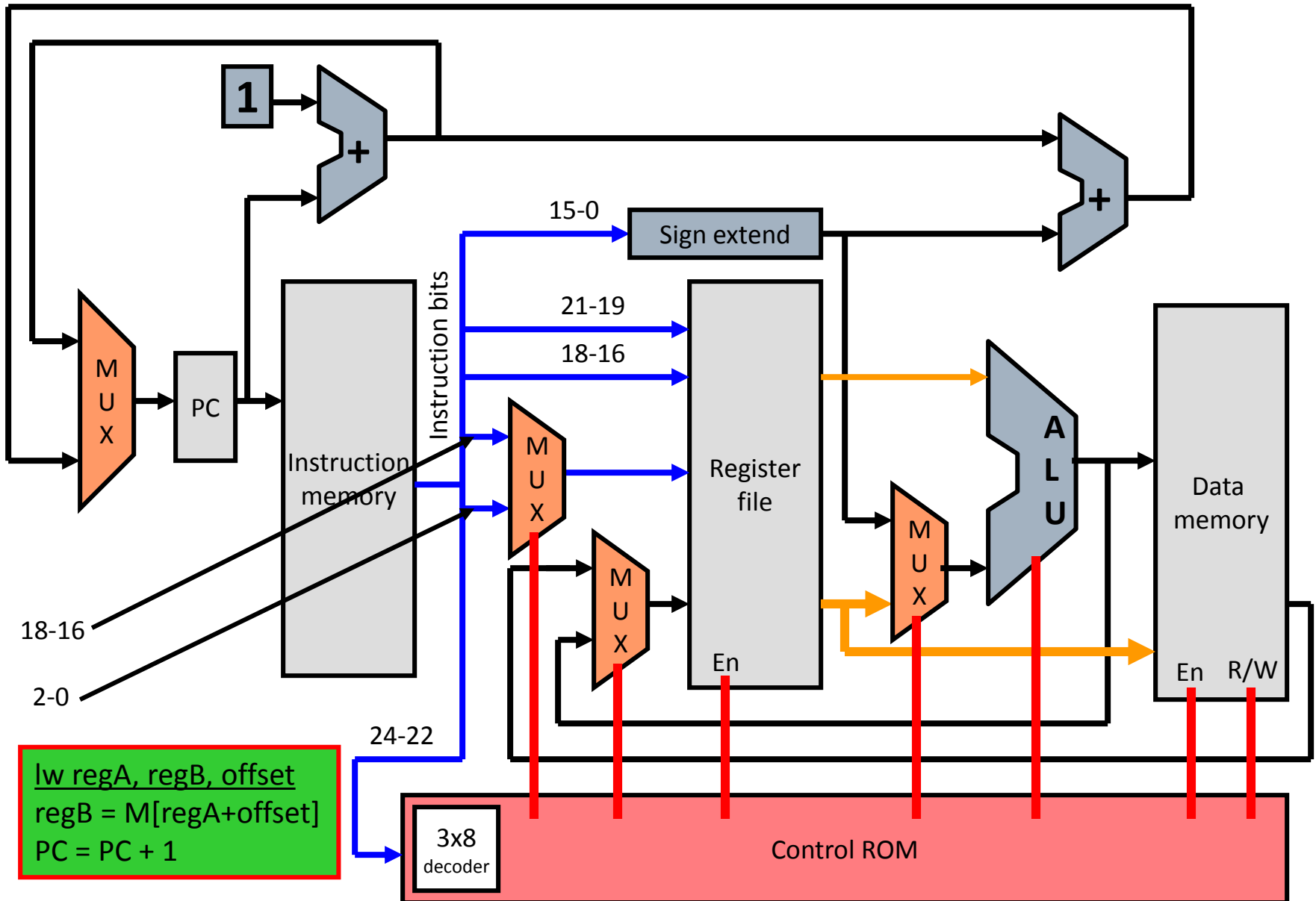
Executing a **NAND** Instruction



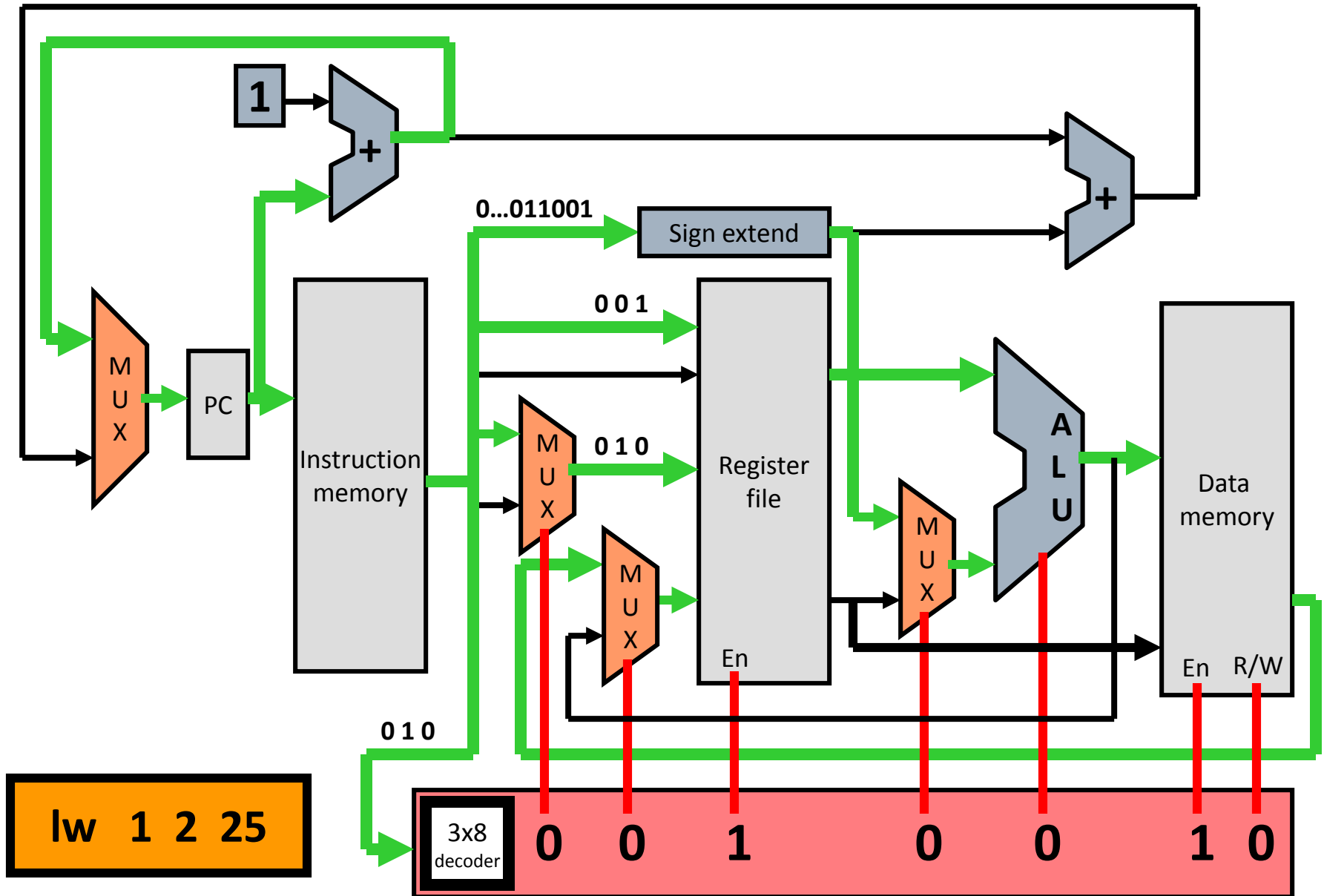
Executing **NAND** Instruction on LC2Kx Datapath



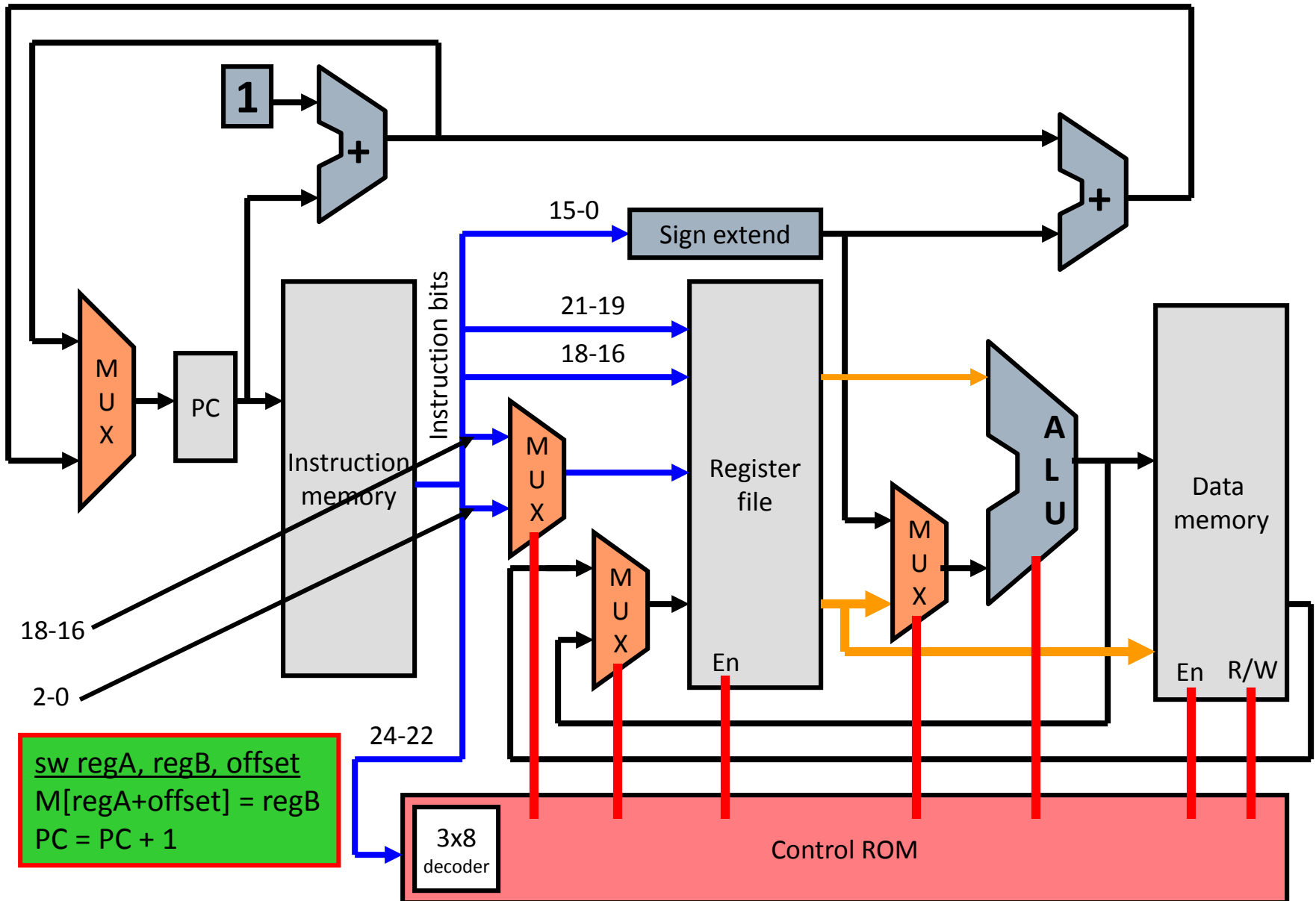
Executing a **LW** Instruction



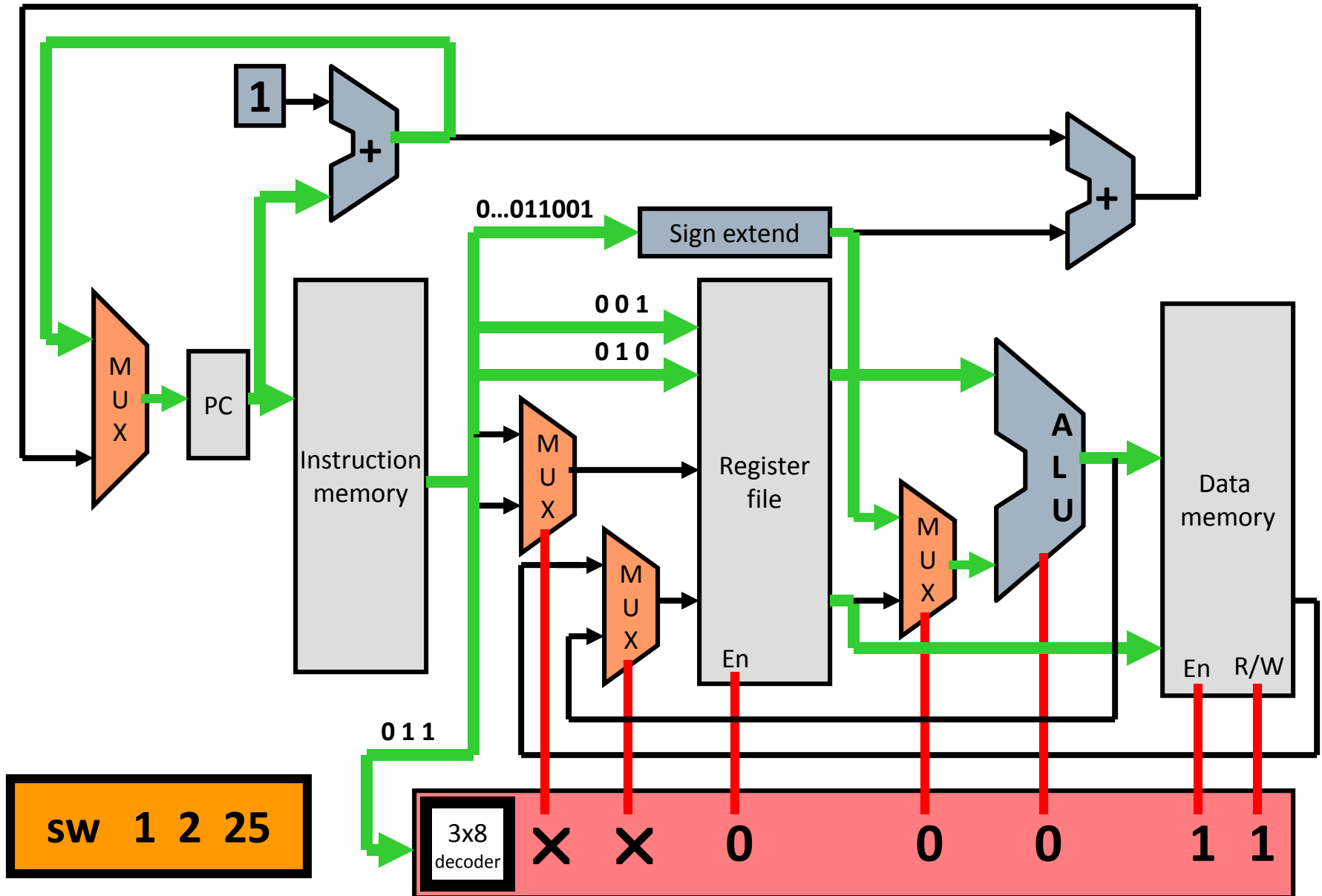
Executing a **LW** Instruction on LC2Kx Datapath



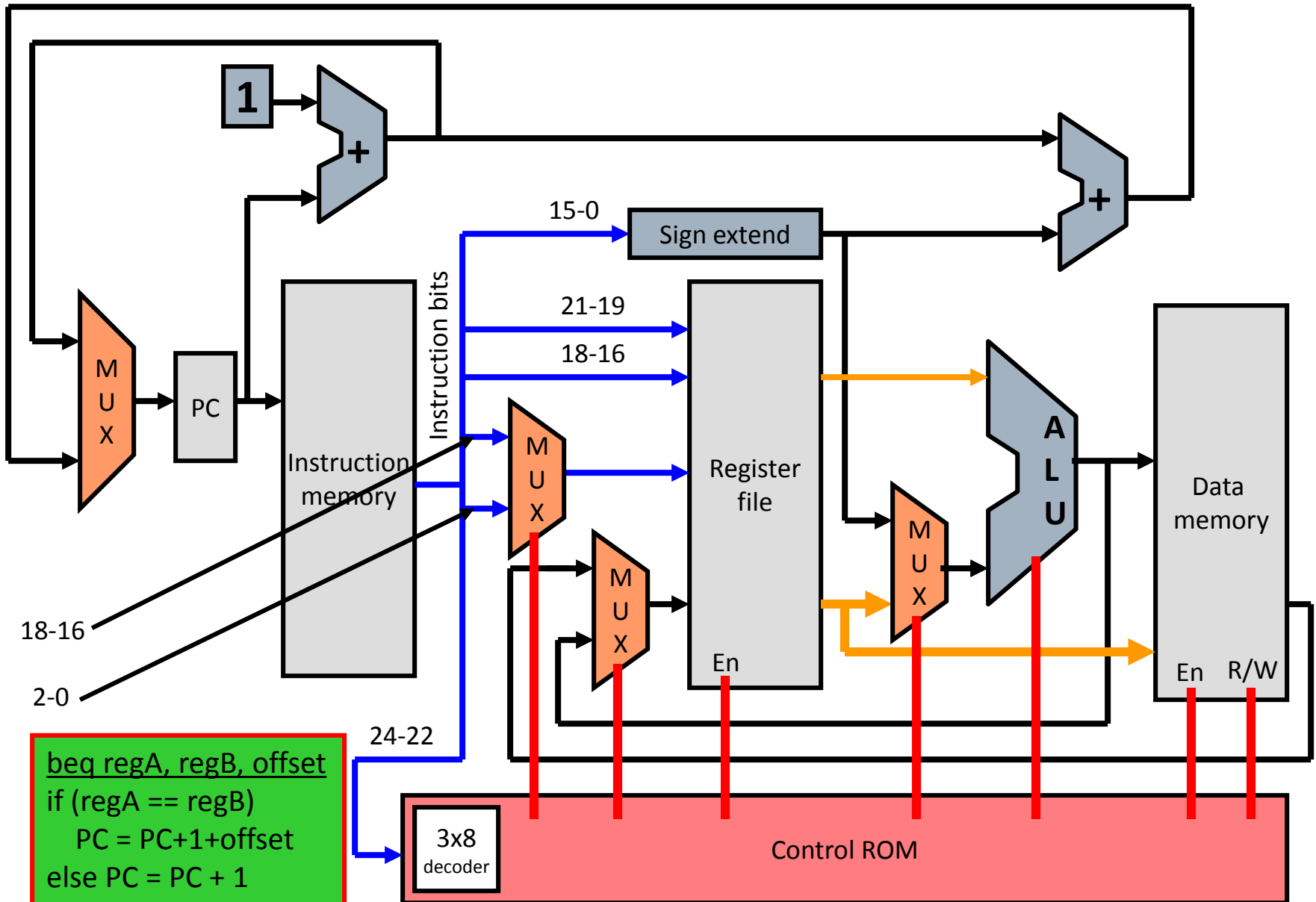
Executing a **SW** Instruction



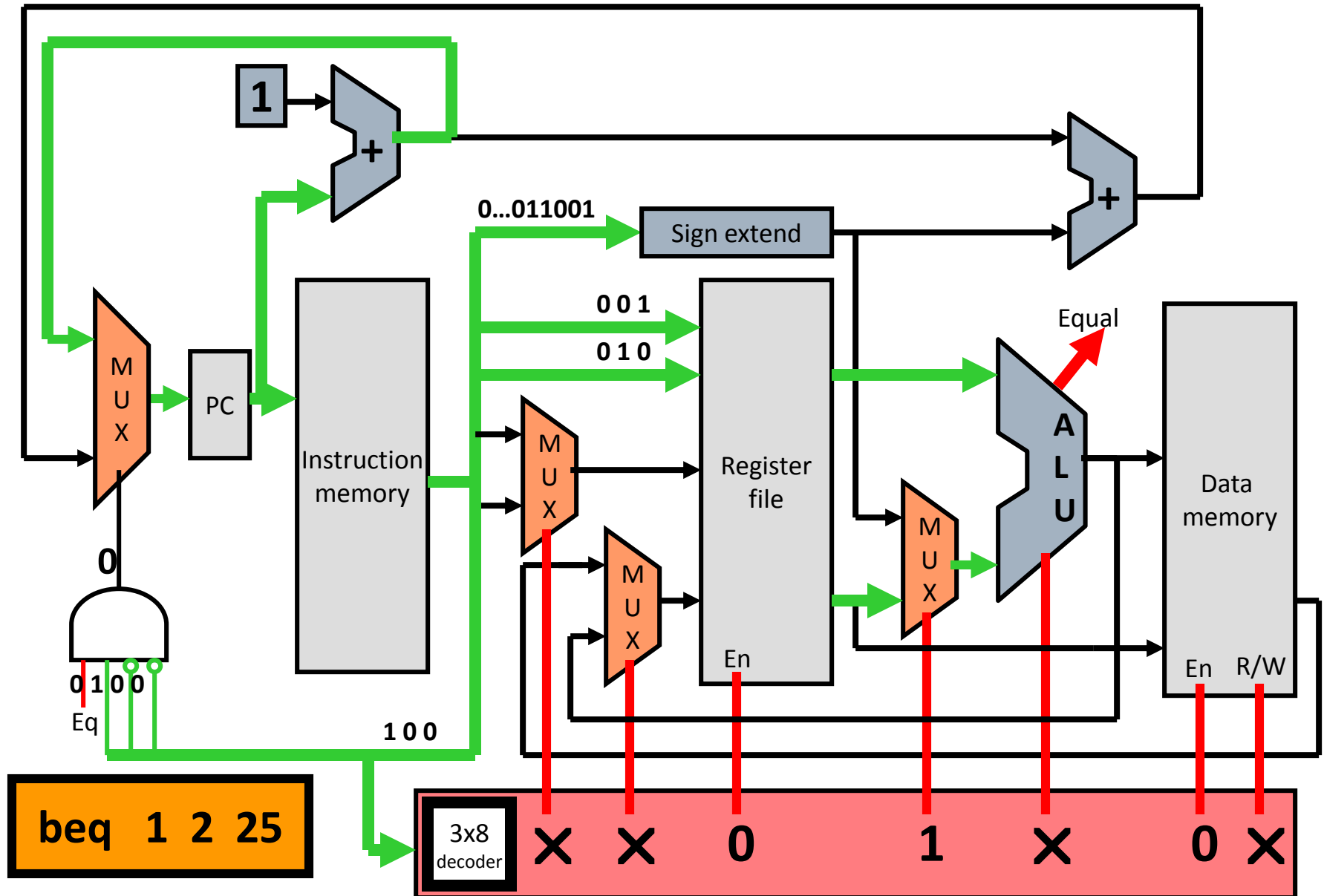
Executing a **SW** Instruction on LC2Kx Datapath



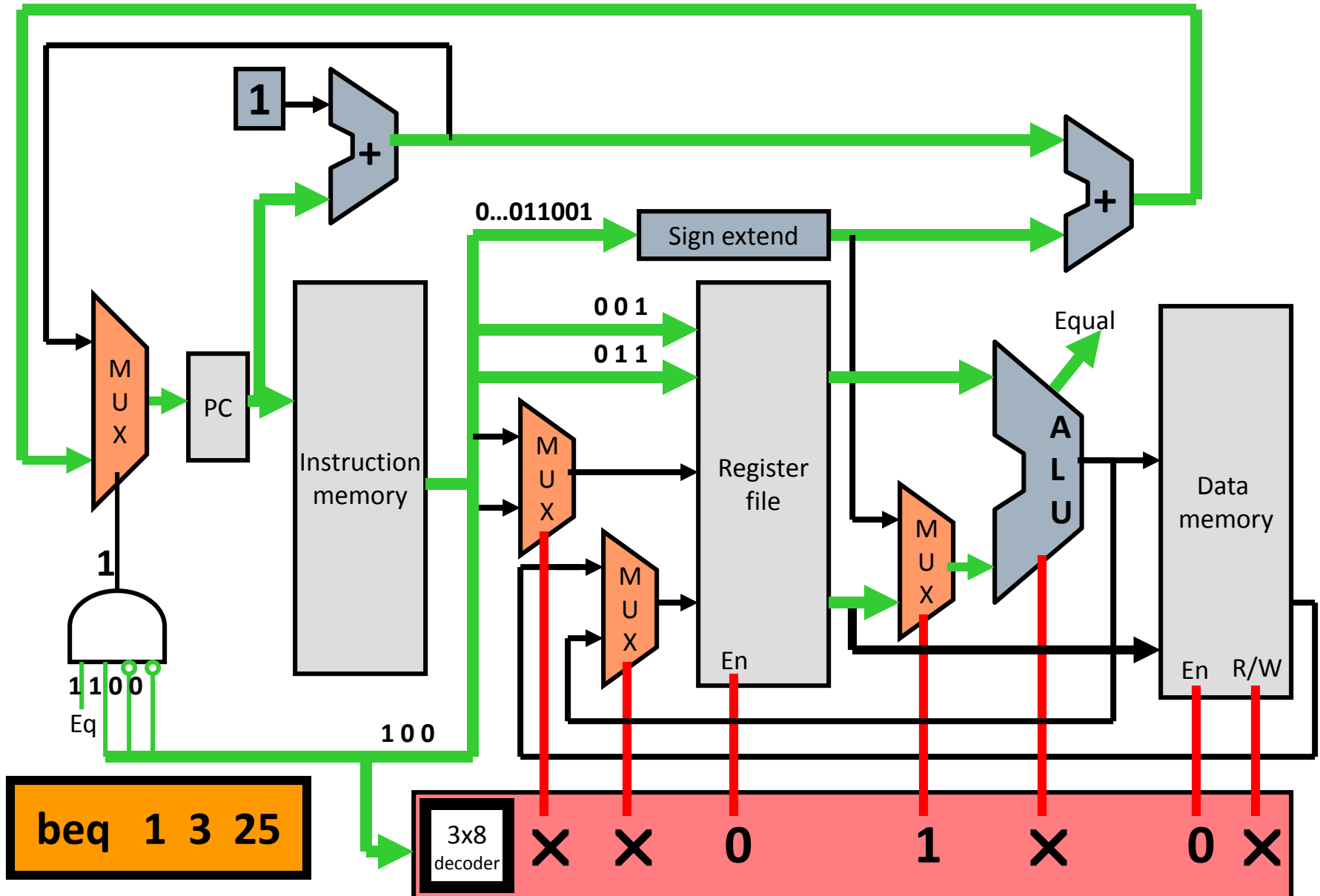
Executing a **BEQ** Instruction



Executing “not taken” **BEQ** Instruction on LC2K Datapath



Executing a “taken” BEQ Instruction on LC2K Datapath

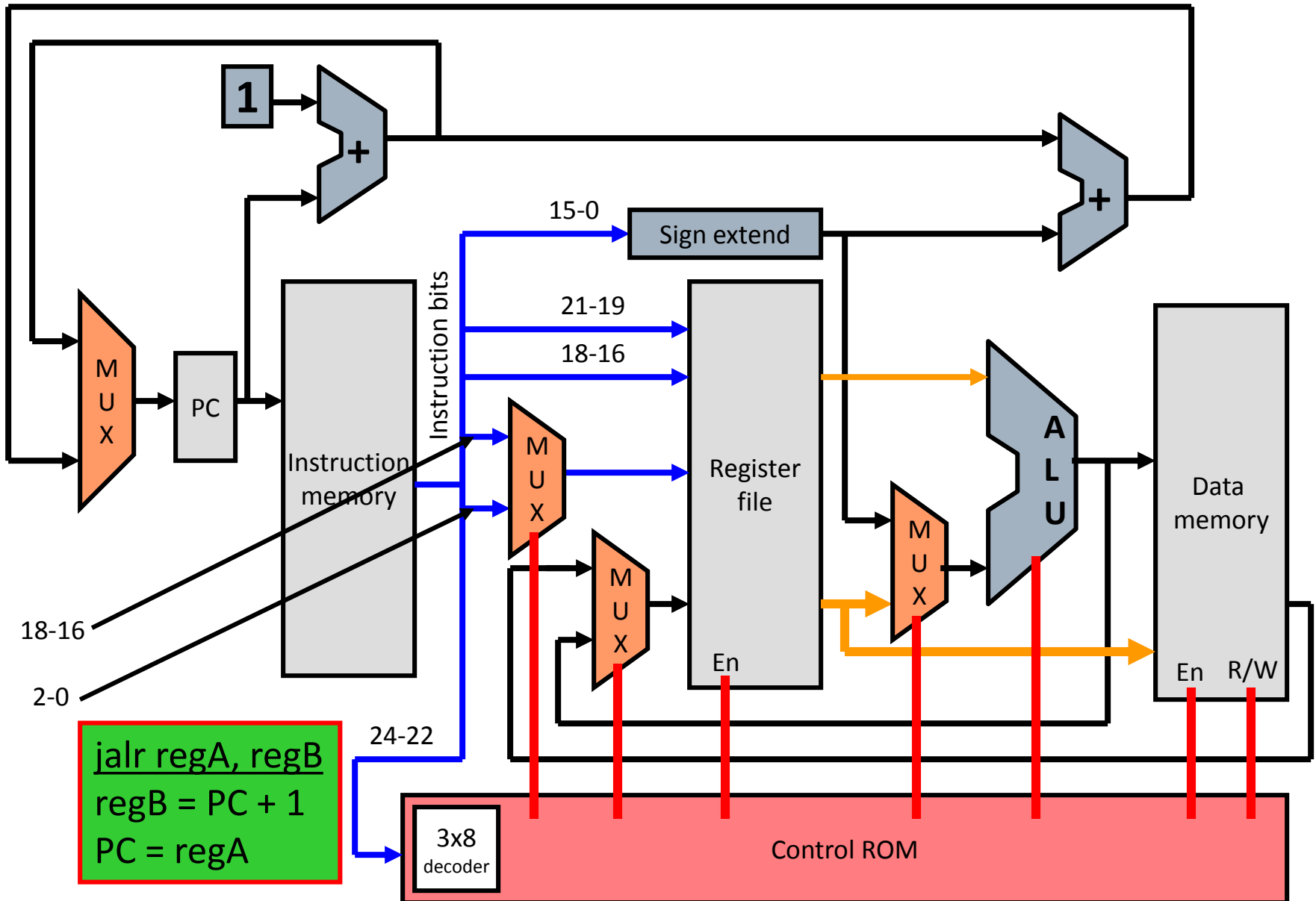


So Far, So Good

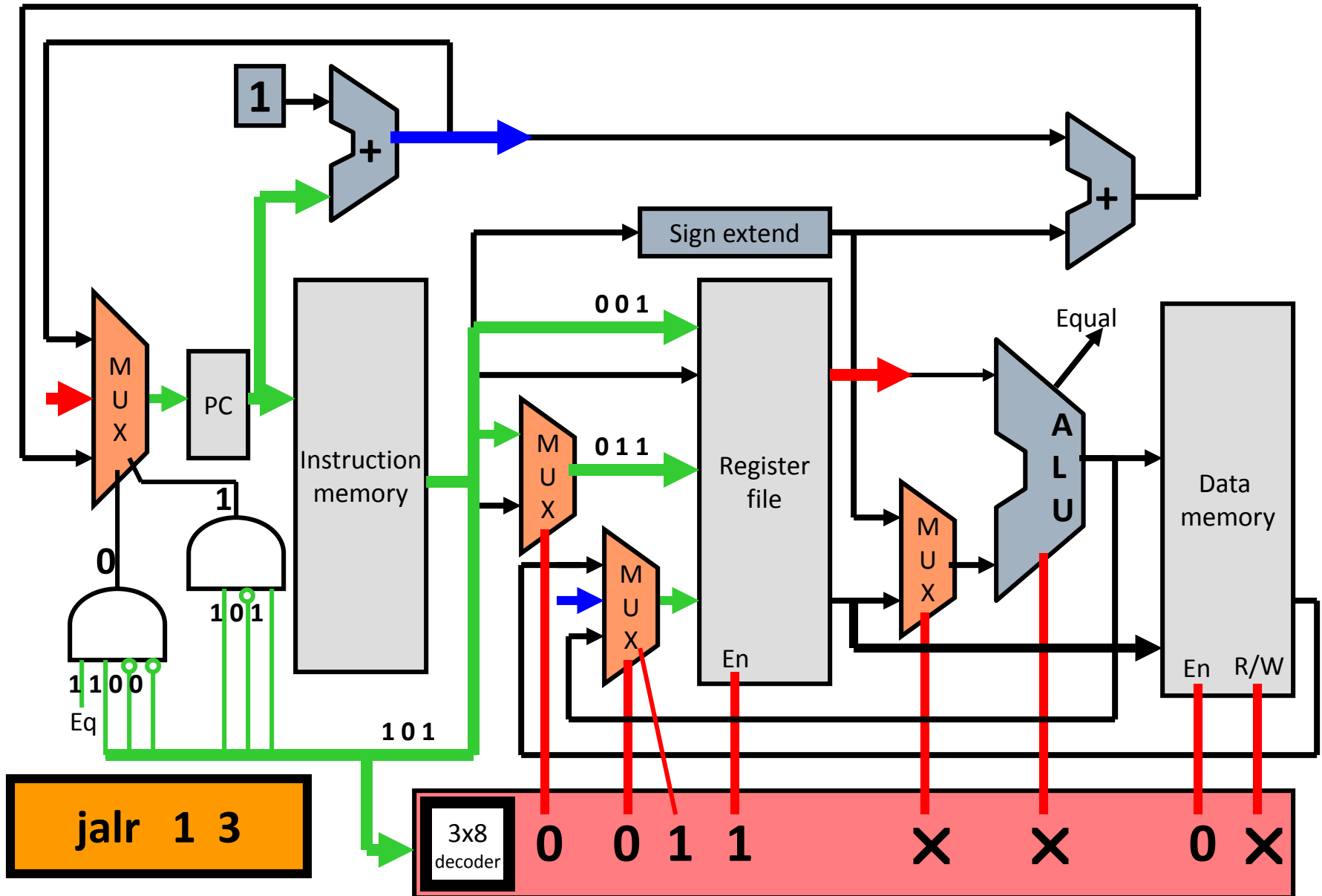
❑ Every architecture seems to have at least one ugly instruction.

- JALR doesn't fit into our nice clean datapath
- To implement JALR we need to
 - Write PC+1 into regB
 - Move regA into PC
- Right now there is:
 - No path to write PC+1 into a register
 - No path to write a register to the PC

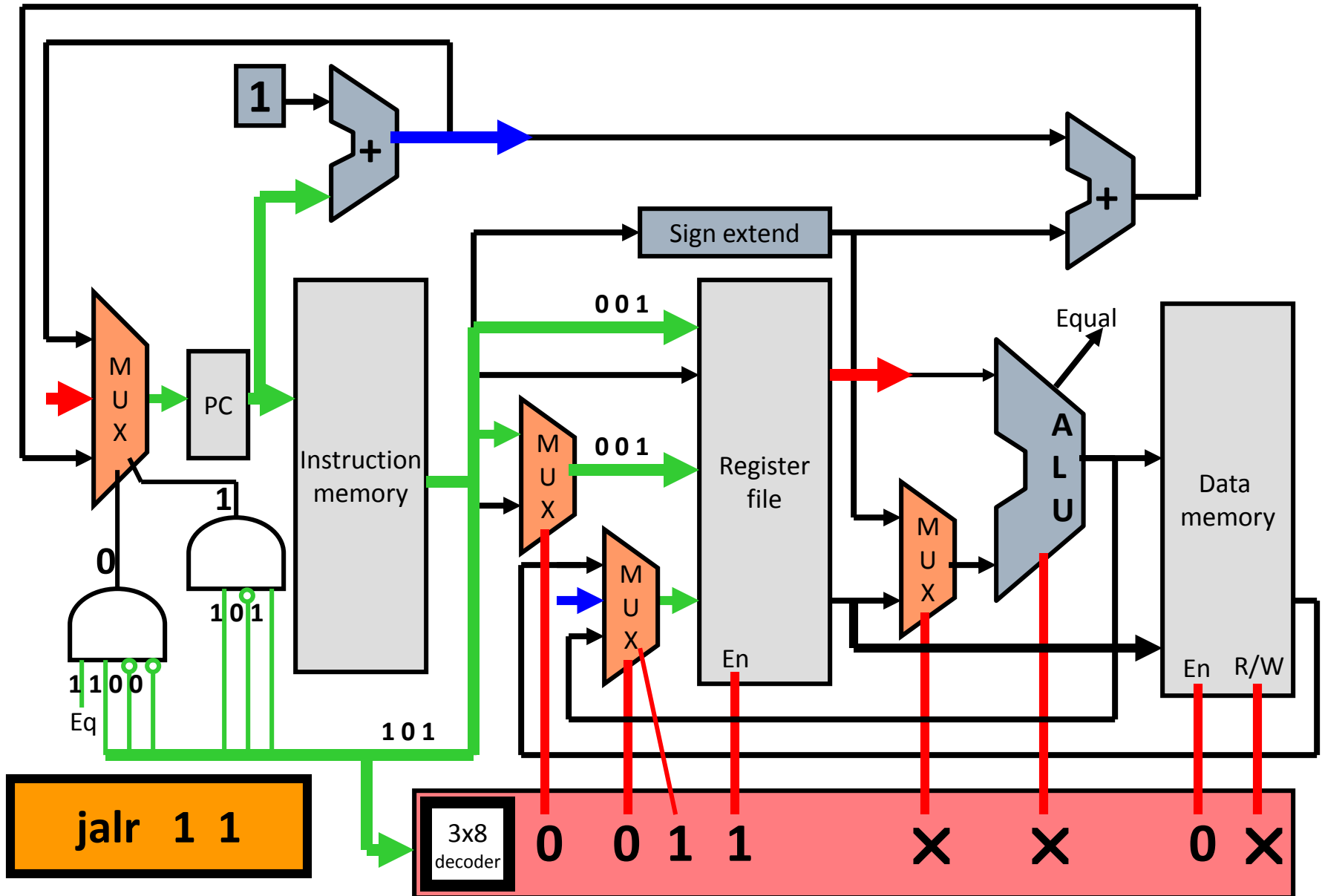
Executing a **JALR** Instruction



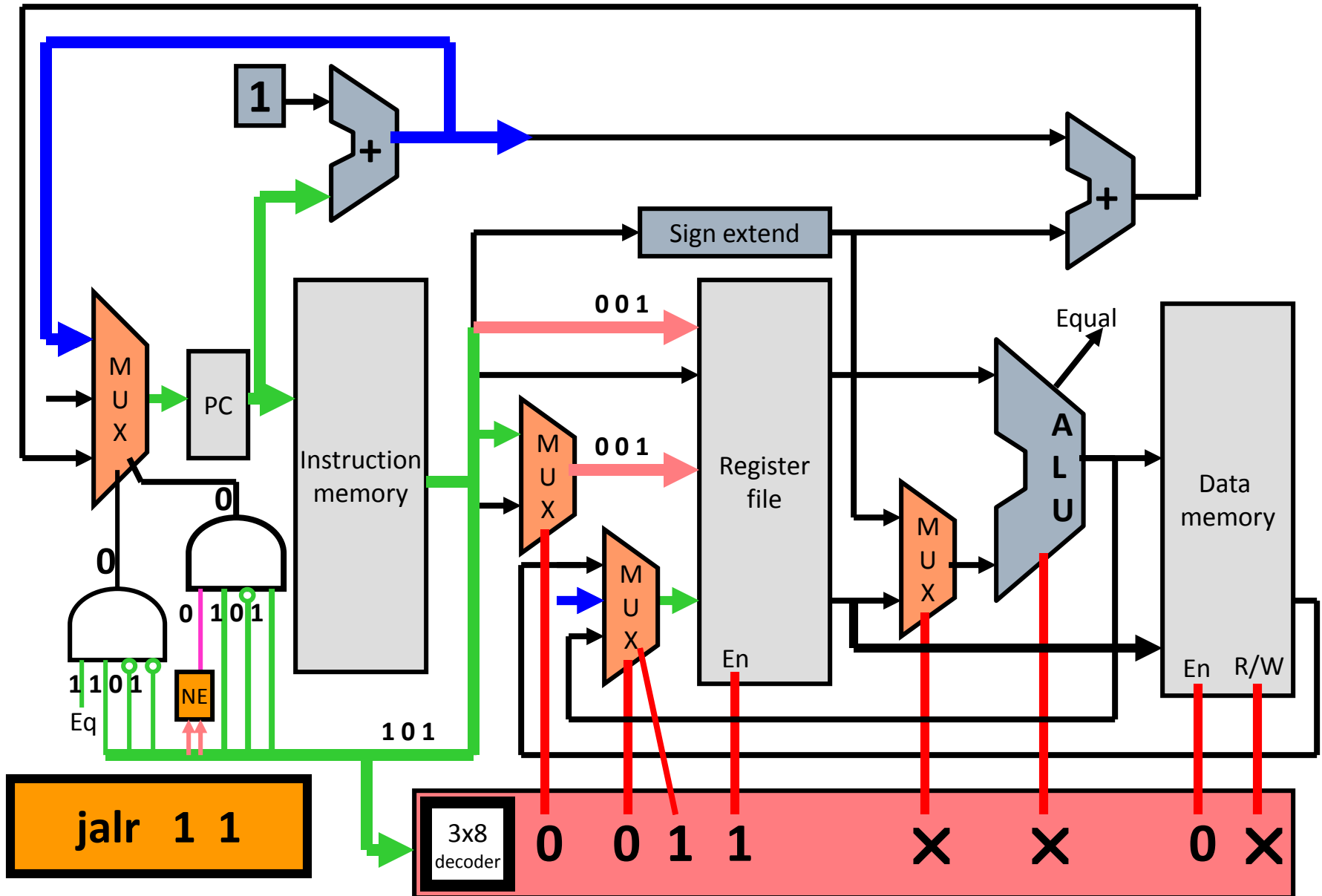
Executing a **JALR** Instruction on LC2Kx Datapath



What If $\text{regA} = \text{regB}$ for a JALR ?



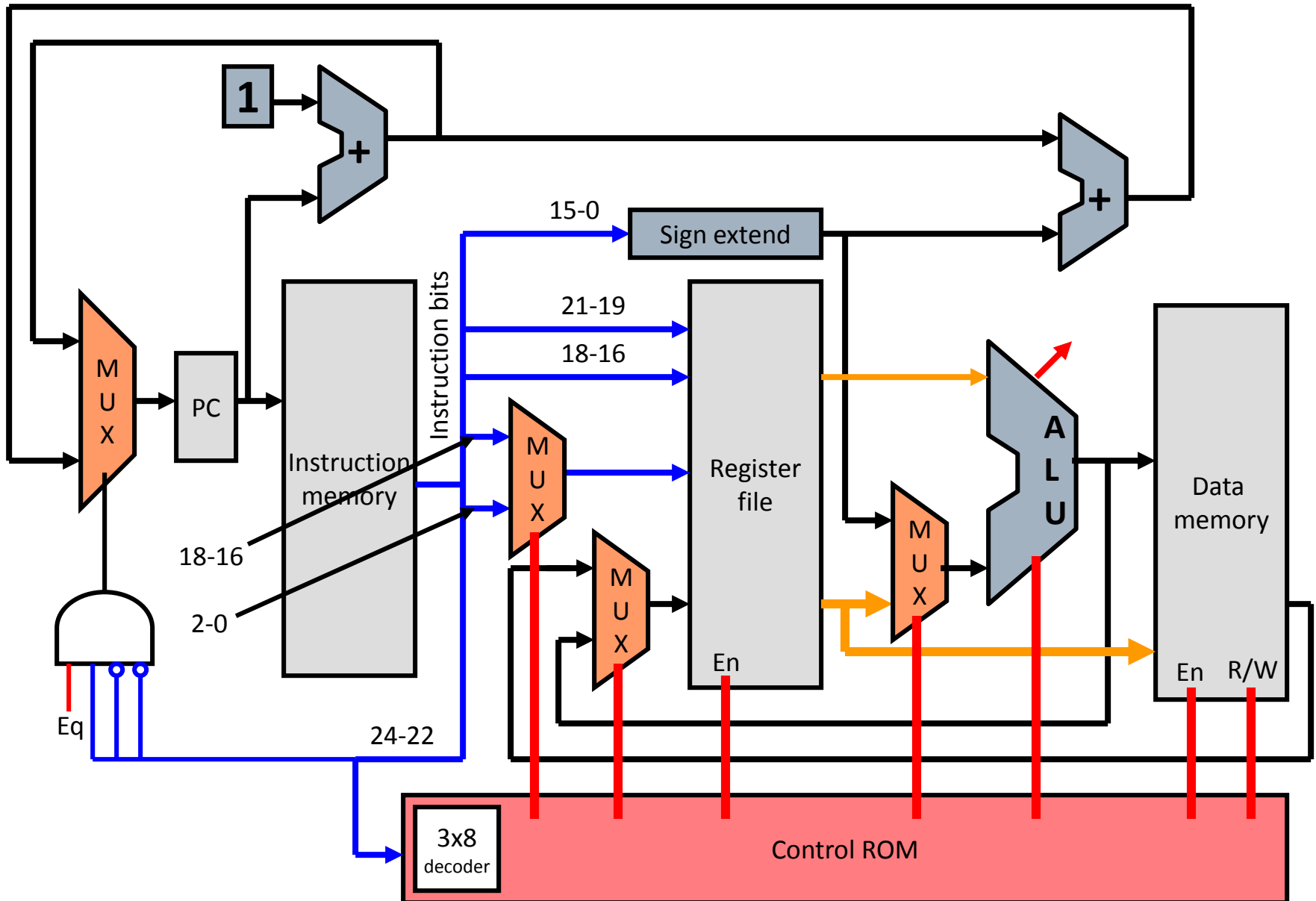
Changes for a **JALR 1 1** Instruction



Class Problem

- ❑ Extend the single cycle datapath to perform the following operation
 - `cmov regA, regB, destR`
 - `destR = regA` (if `regB != 0`)
 - `PC = PC + 1`

Class Problem (continued)



What's Wrong with Single Cycle?

- ❑ **All instructions run at the speed of the slowest instruction.**
- ❑ Adding a long instruction can hurt performance
 - What if you wanted to include multiply?
- ❑ You cannot reuse any parts of the processor
 - We have 3 different adders to calculate $PC+1$, $PC+1+offset$ and the ALU
- ❑ No benefit in making the common case fast
 - Since every instruction runs at the slowest instruction speed
 - This is particularly important for loads as we will see later

What's Wrong with Single Cycle?

- 1 ns – Register read/write time
- 2 ns – ALU/adder
- 2 ns – memory access
- 0 ns – MUX, PC access, sign extend, ROM

	Get Instr	read reg	ALU oper.	mem	write reg	
• add:	2ns	+ 1ns	+ 2ns		+ 1 ns	= 6 ns
• beq:	2ns	+ 1ns	+ 2ns			= 5 ns
• sw:	2ns	+ 1ns	+ 2ns	+ 2ns		= 7 ns
• lw:	2ns	+ 1ns	+ 2ns	+ 2ns	+ 1ns	= 8 ns

Computing Execution Time

❑ Assume: 100 instructions executed

- 25% of instructions are loads,
- 10% of instructions are stores,
- 45% of instructions are adds, and
- 20% of instructions are branches.

❑ Single-cycle execution:

= ??

❑ Optimal execution:

= ??