

Digital Building Blocks

CS31

Pascal Van Hentenryck



Overview

Digital Building Blocks

- Decoders and Multiplexers
- ALU (arithmetic and logic unit)
- ROM

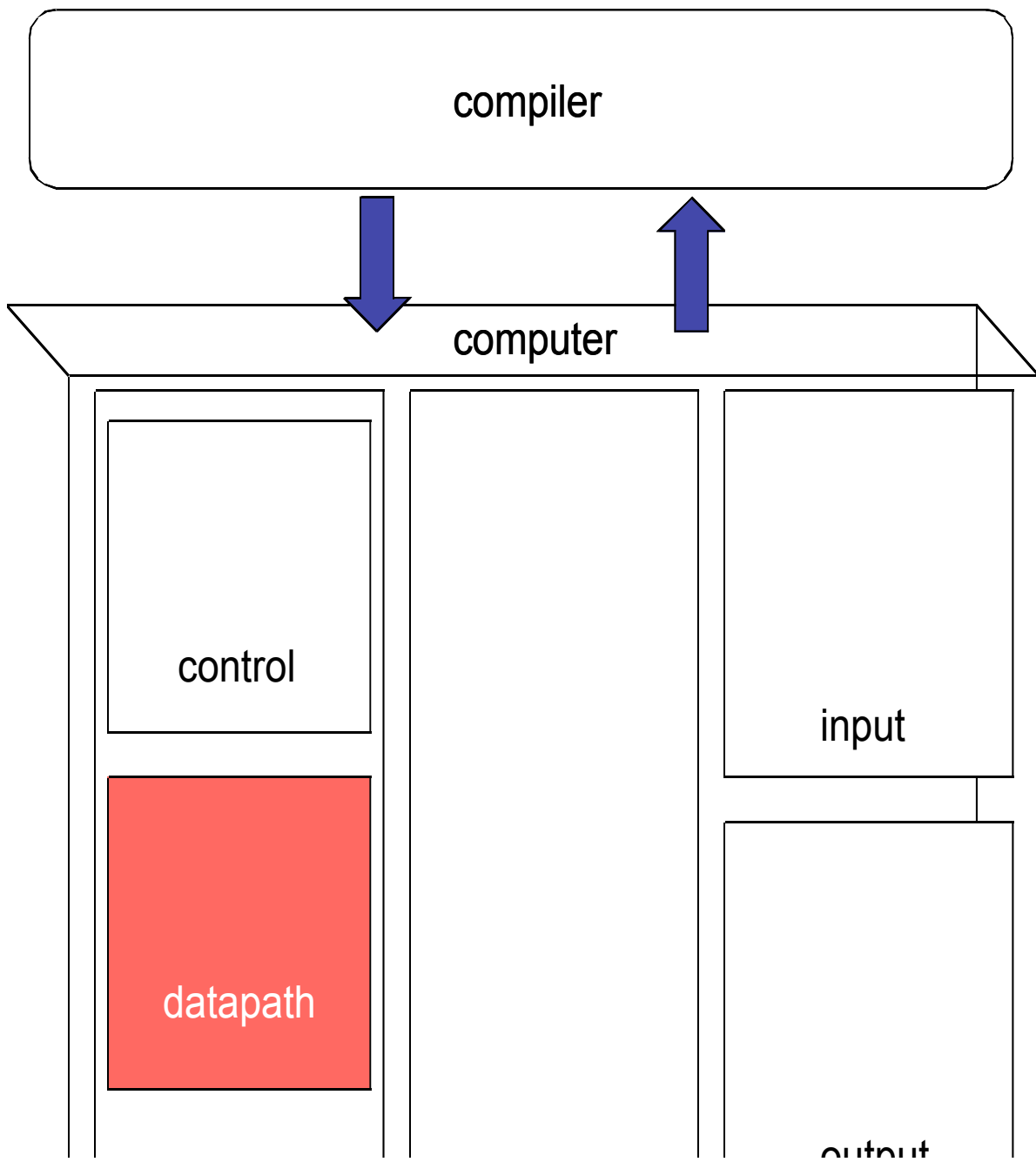
ALU

- This is the brawn of the computer

ROM

- Useful for implementing the control

The Big Picture



Abstraction Hierarchy

Programming Language

Assembly Language

Machine Language

Sequential Circuit

Combinational Circuit

Binary Value

Voltage

Two Building Blocks

Decoders

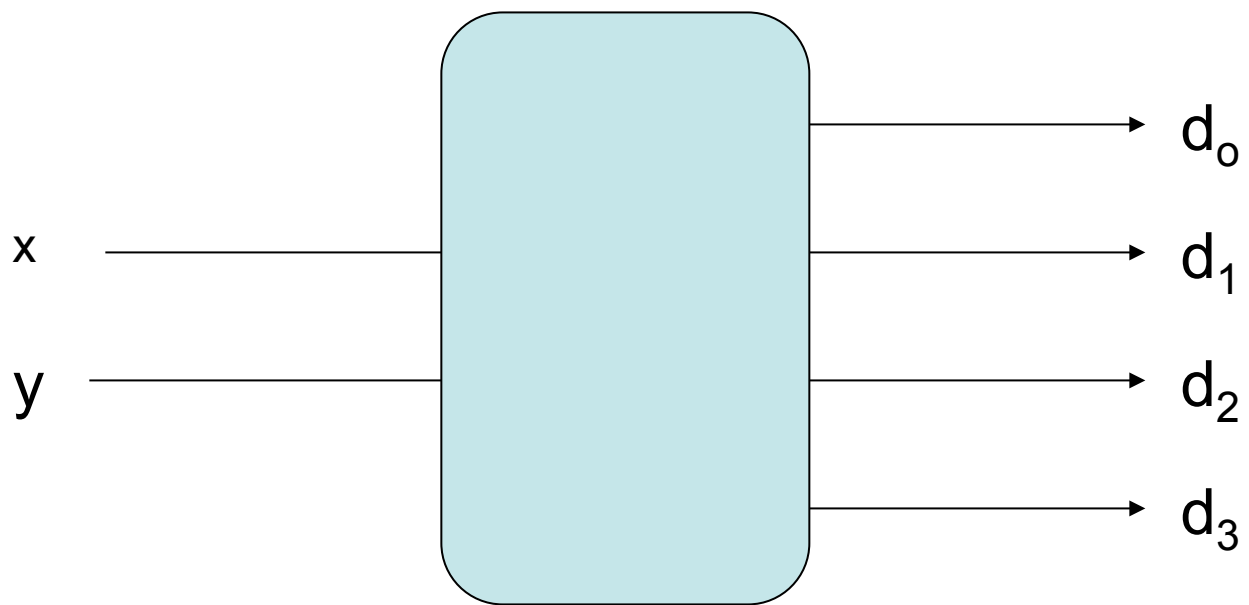
- n inputs / 2^n outputs
- Given the inputs, select a unique output
- Particularly useful for implementing memories

Multiplexers (selectors)

- 2^n data inputs / n selection inputs / 1 output
- The output is the data input selected by the selection input
- Particularly useful for implementing ALU and various other pieces of the machine

Decoders

Turn binary “coded” quantities into one unit vector for every possible value.



Decoders

Turn binary “coded” quantities into one unit vector for every possible value.

x	y	d_0	d_1	d_2	d_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Look at (x, y) as a binary number

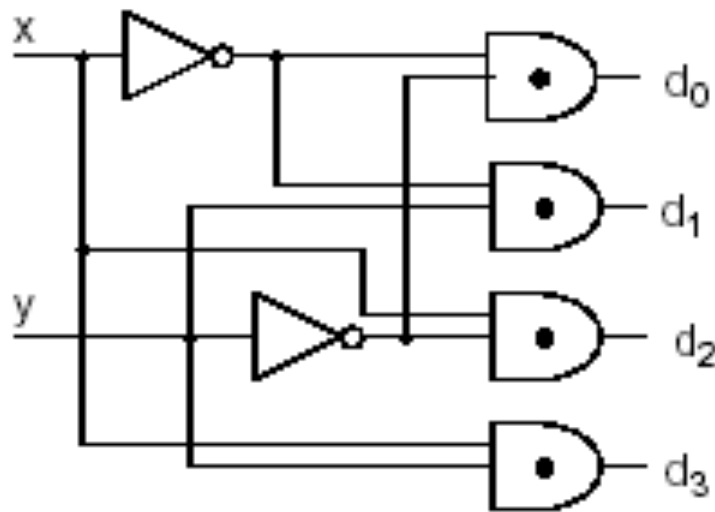
The decoder

- assign d_{2^*x+y} to 1
- all the others to zero

Decoders

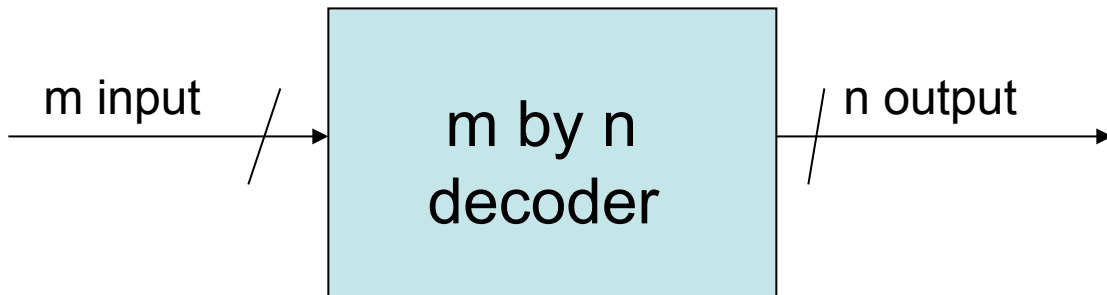
Turn binary “coded” quantities into one unit vector for every possible value.

x	y	d_0	d_1	d_2	d_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



More Decoders

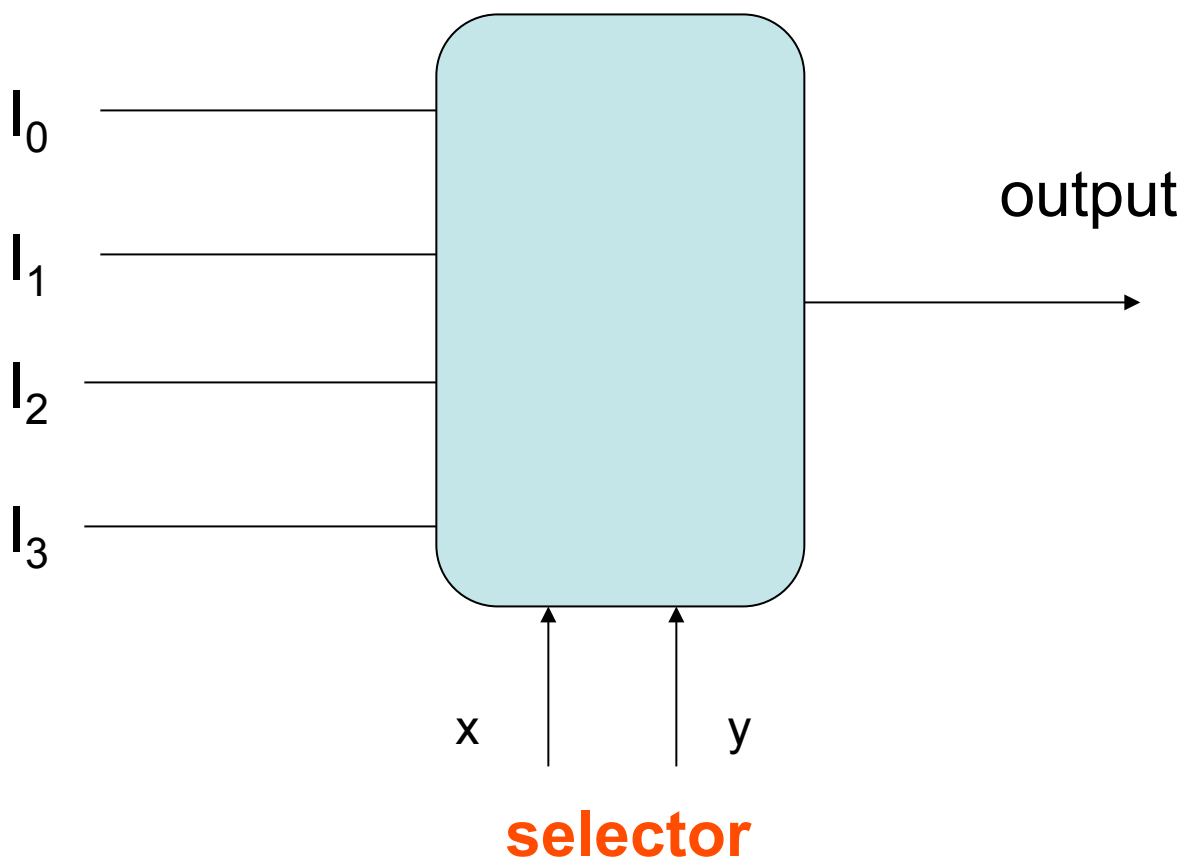
- Decoders can be in any size



- n is always 2^m
- two types of decoders: a single output 1 or a single output is 0

Multiplexers

Choose a particular input to pass through as specified by values on the selection (or address) lines.

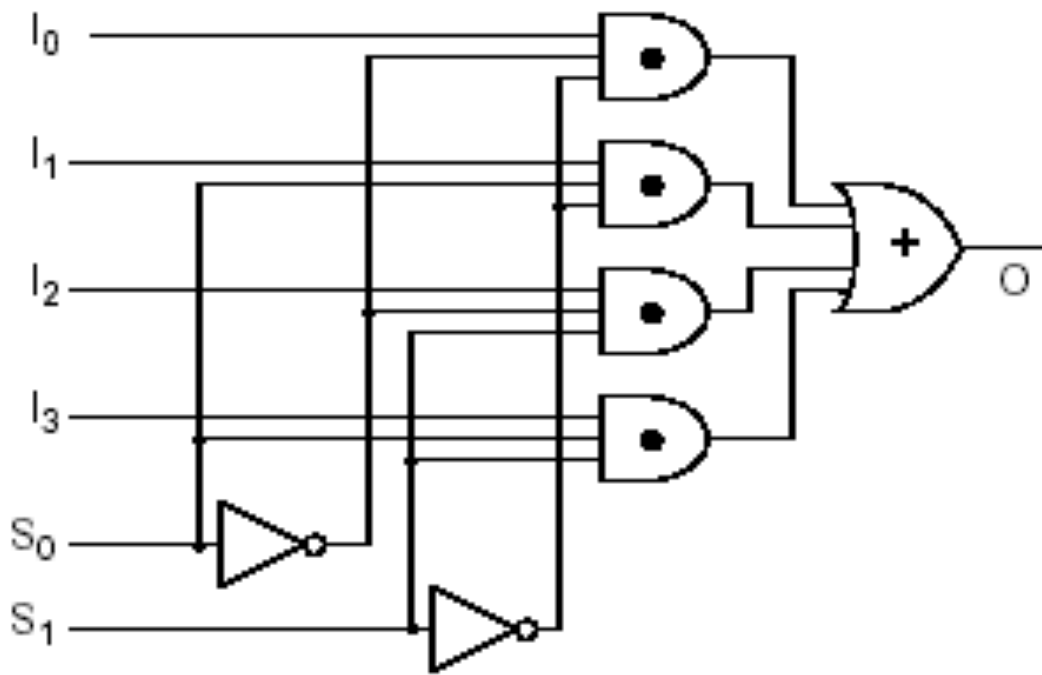


Essentially a switch in hardware

$$\text{Output} = I_{2x+y}$$

Multiplexers

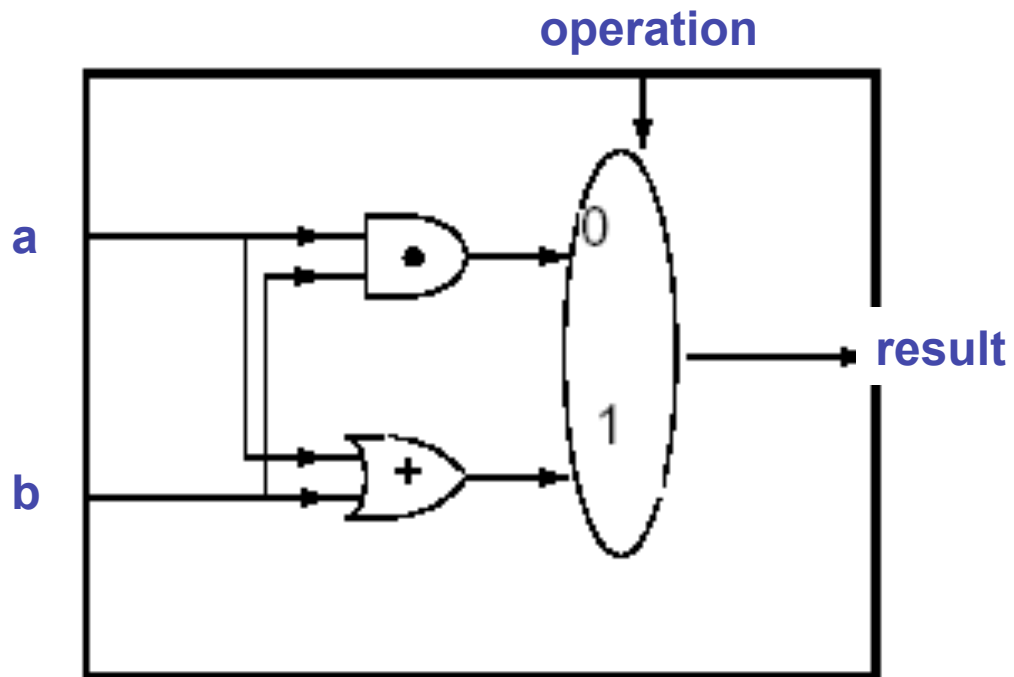
Choose a particular input to pass through as specified by values on the selection (or address) lines.



S_1	S_2	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

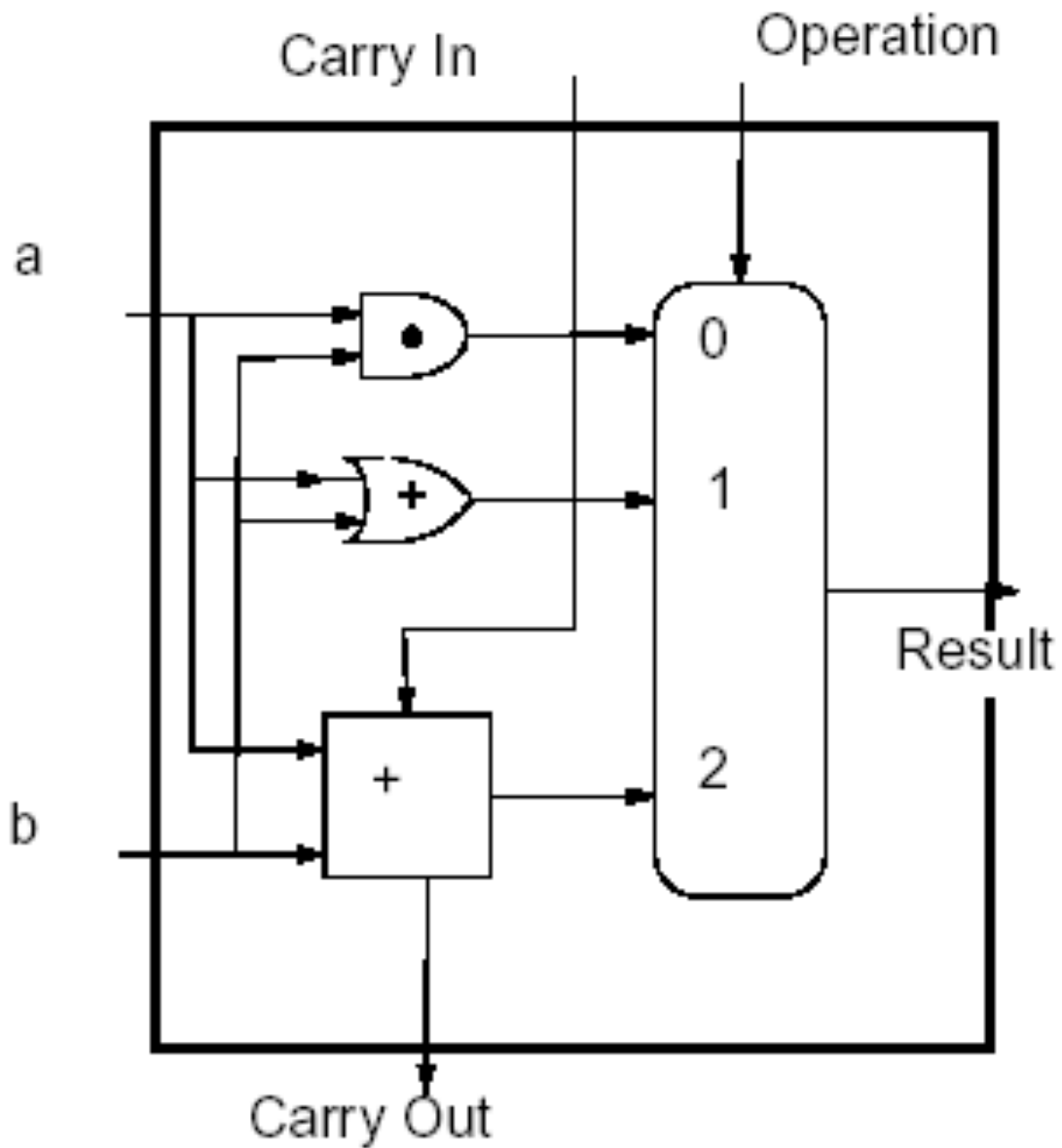
1-bit ALU

1-bit ALU with **and** and **or**

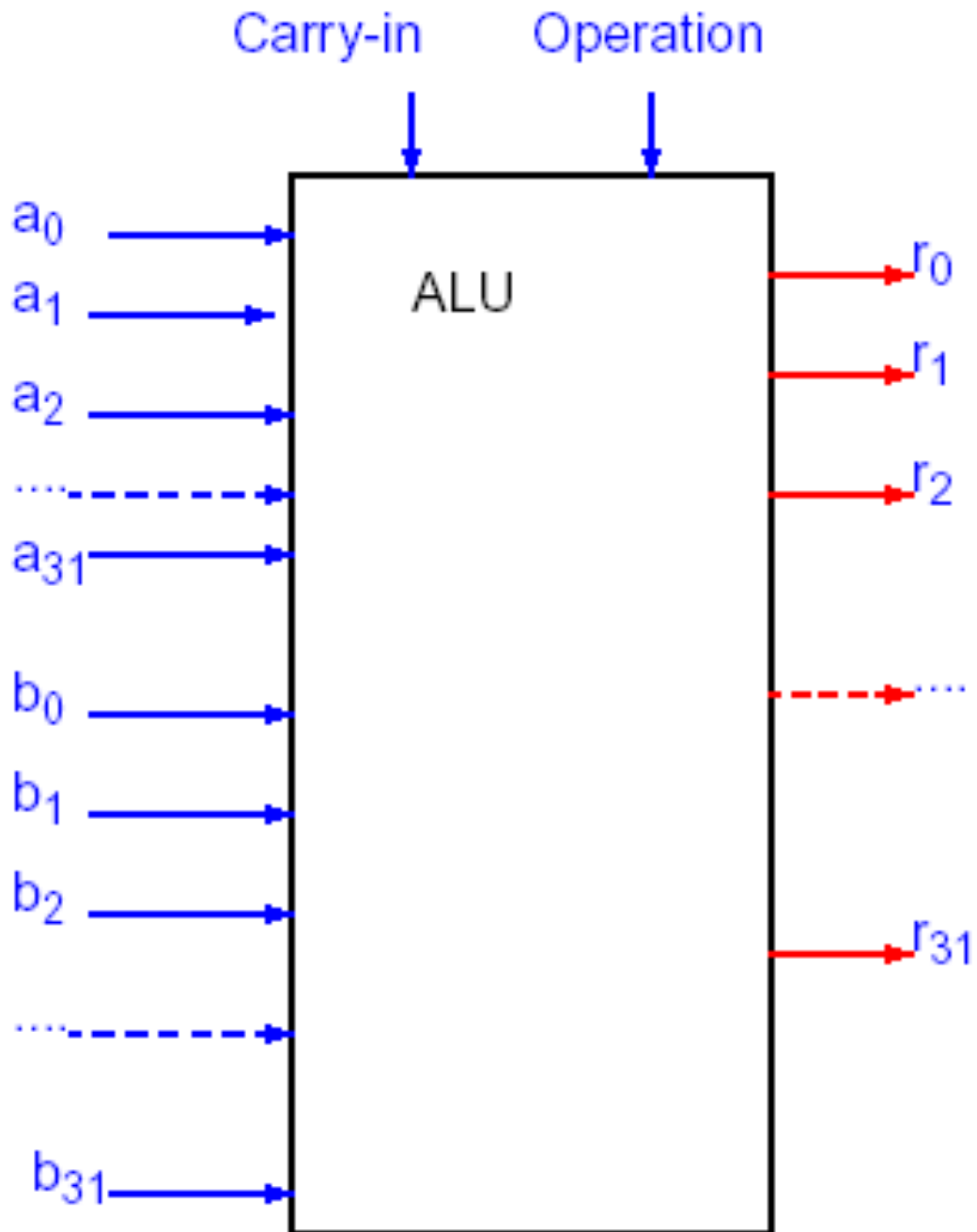


Add addition to this ALU now.

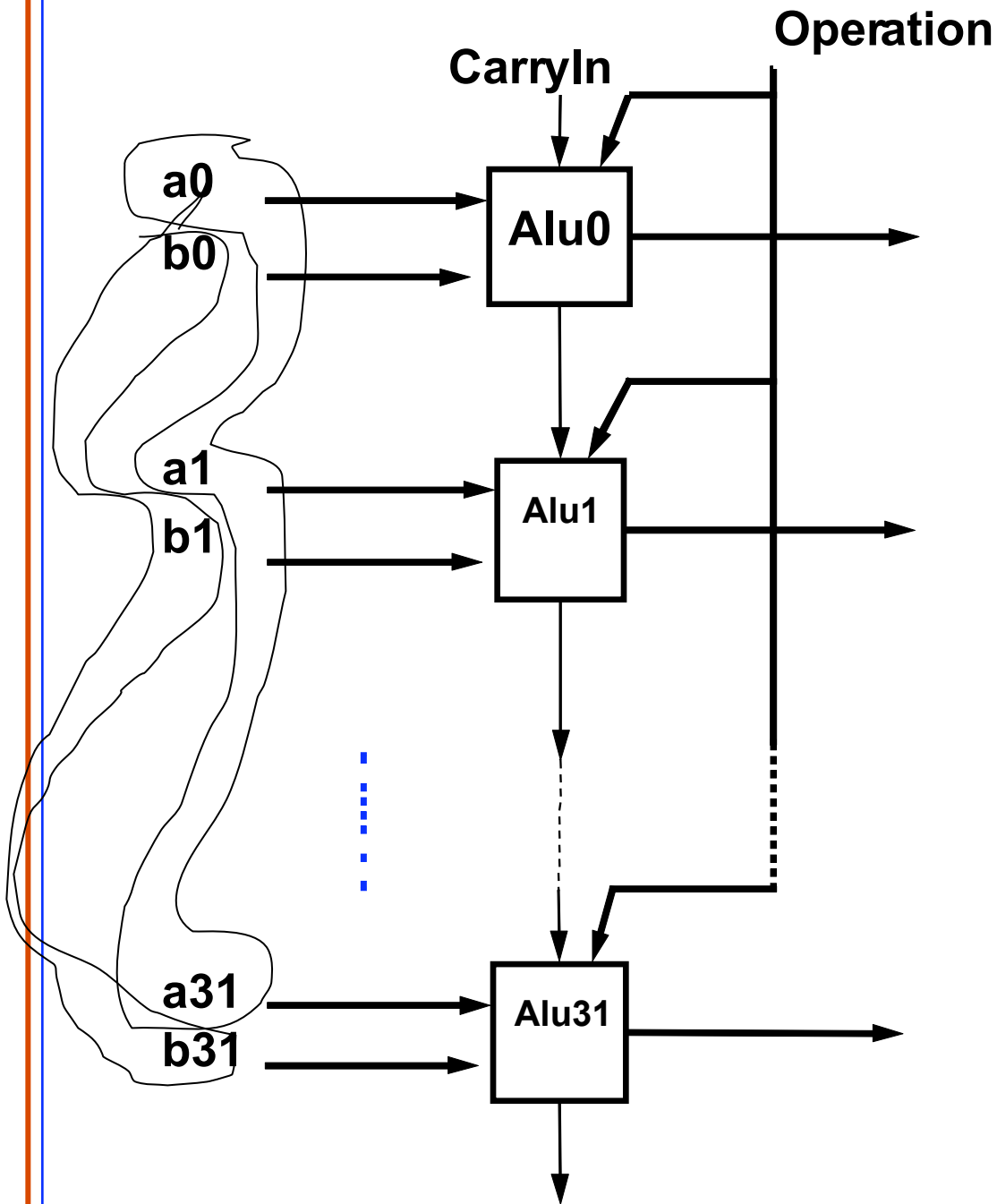
1-bit ALU



32-bit ALU



32-bit ALU



32-bit ALU

Include subtraction

- Add the negative version of b

How to obtain the negative version

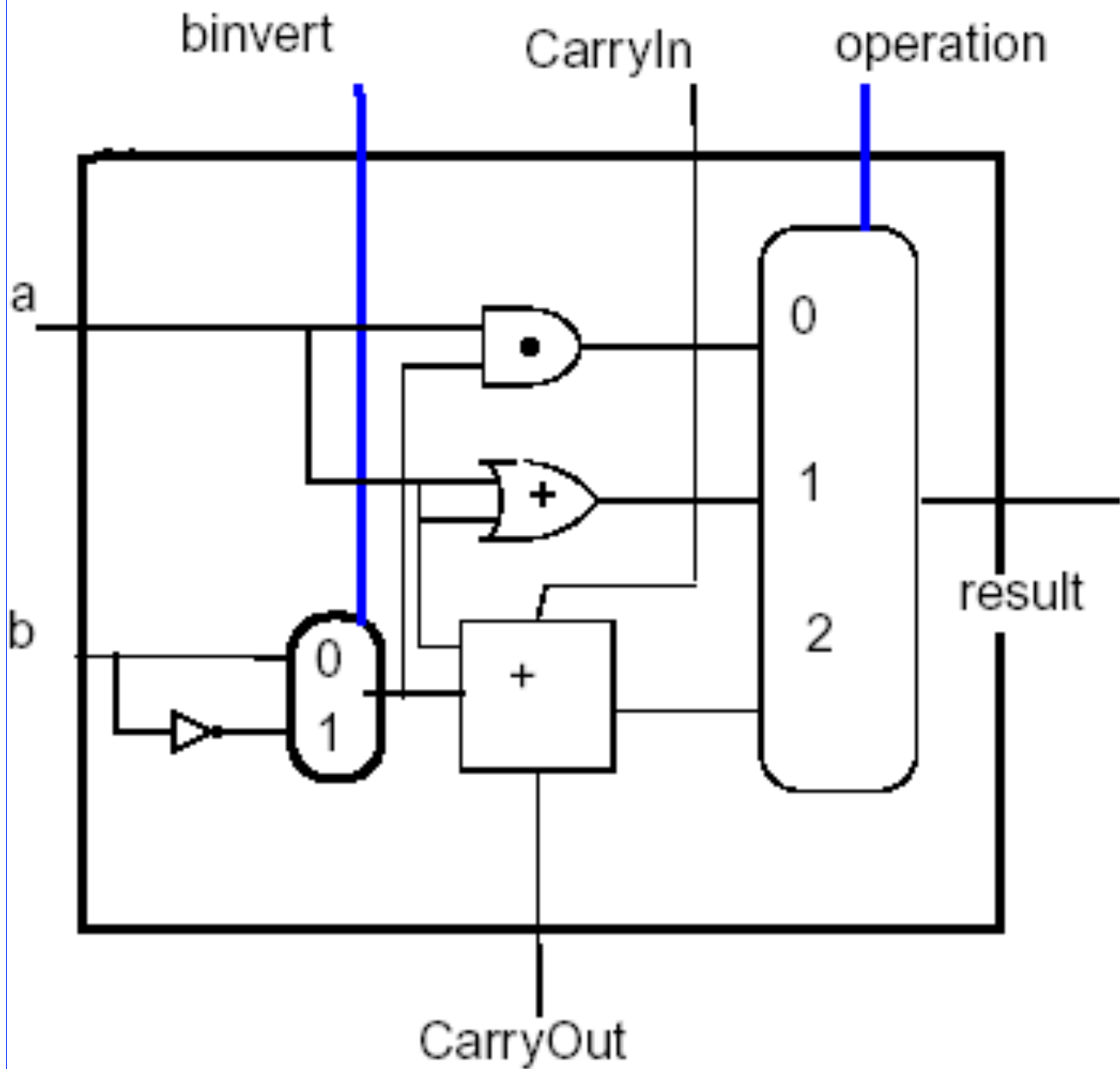
- Invert each bit
- Add 1

How do to that simply

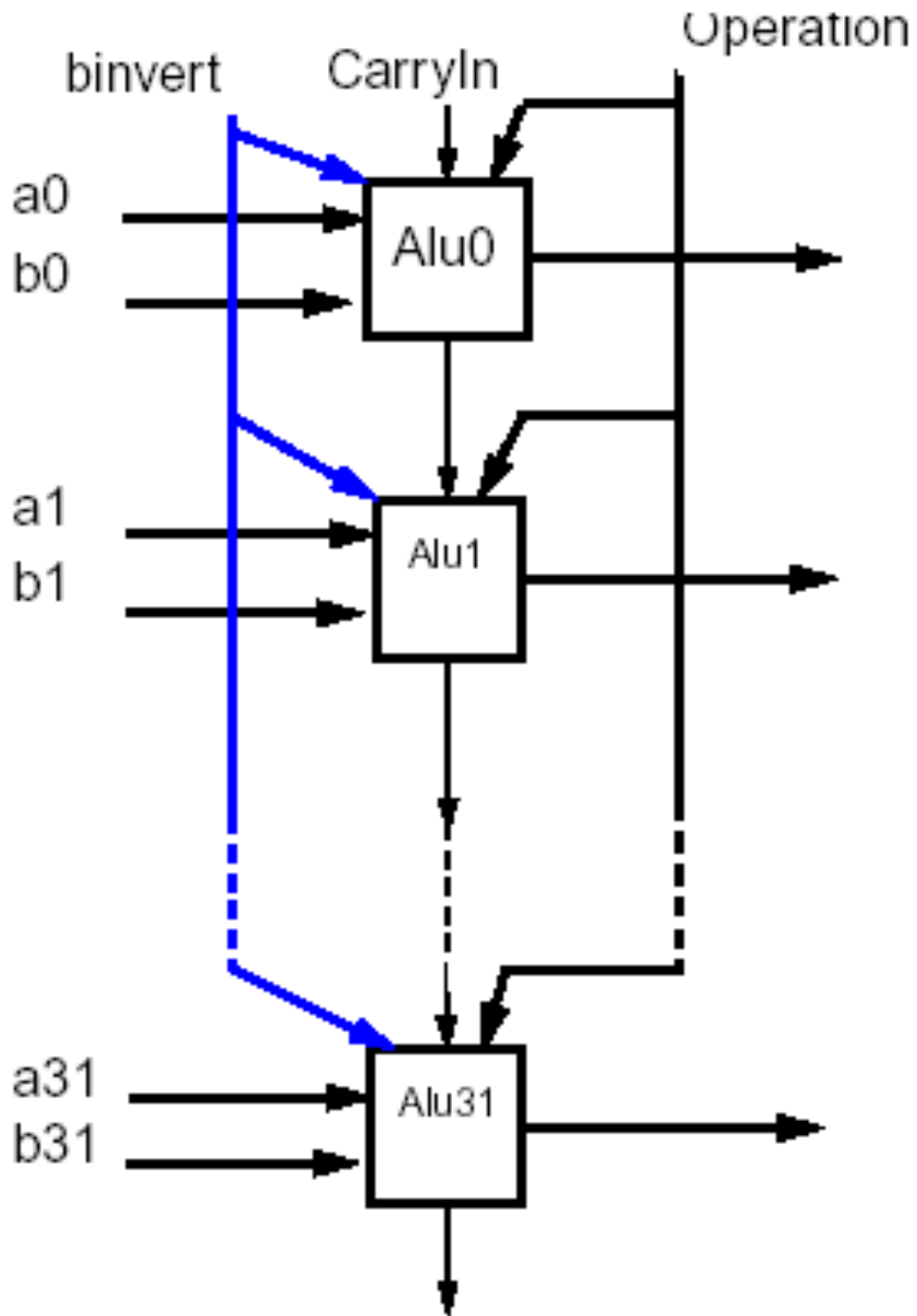
- Generalize a bit
- Use a simple trick

This shows why 2's complement is a good representation

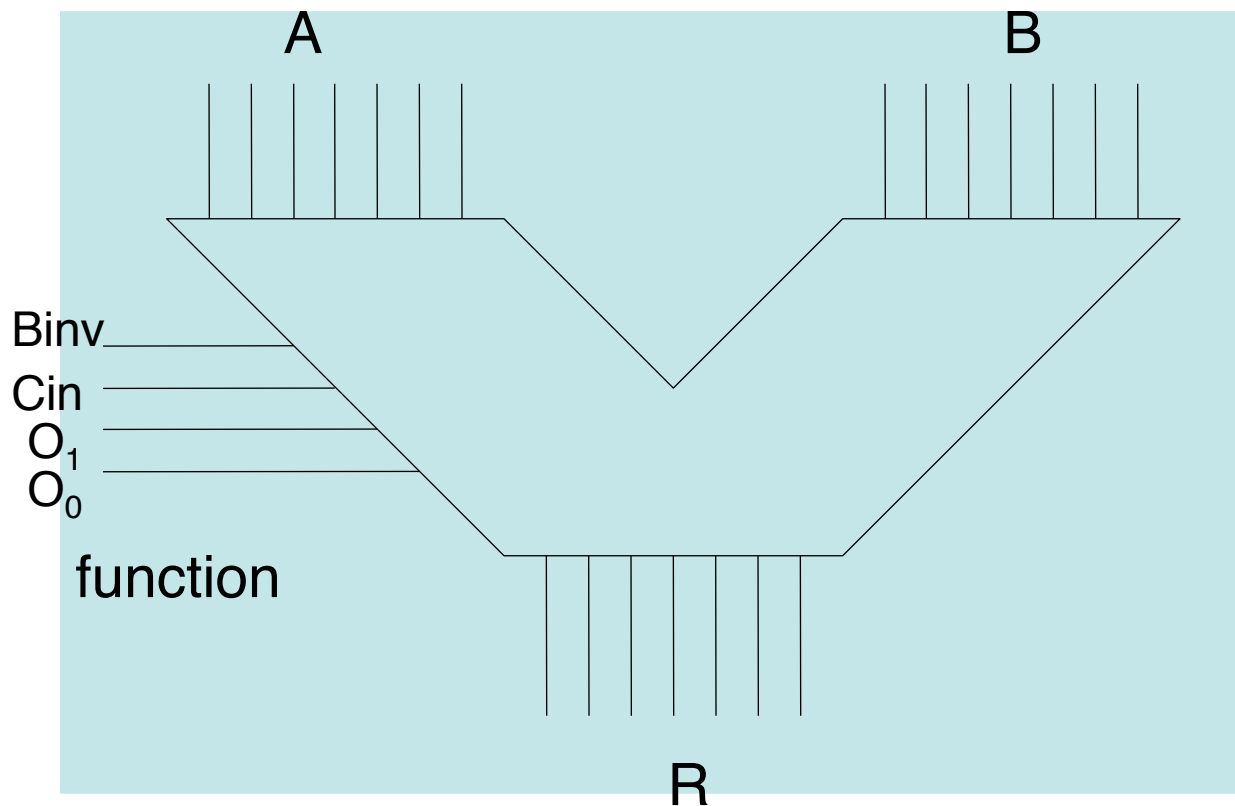
1-bit ALU (Revisited)



32-bit ALU



ALU (Absolutely Lazy Unit)



Possible Modes

Operation 0: $R = A + B$

Operation 1: $R = A - B$

Operation 2: $R = A \text{ and } B$

Operation 3: $R = A \text{ or } B$

Controlling the ALU

	O_1	O_0	B_{inv}	C_{in}
$A \wedge B$	1	0	0	x
$A \vee B$	1	1	0	x
$A + B$	0	0	0	0
$A - B$	0	0	1	1

(Look Ma! I can build an ALU)

Implementing a Logic Function

Three main possibilities

- Specific circuit
- ROM
- PLA

Specific Circuit

- when there is a lot of structure (the Karnaugh map, remember?)

ROM

- when there is almost no structure

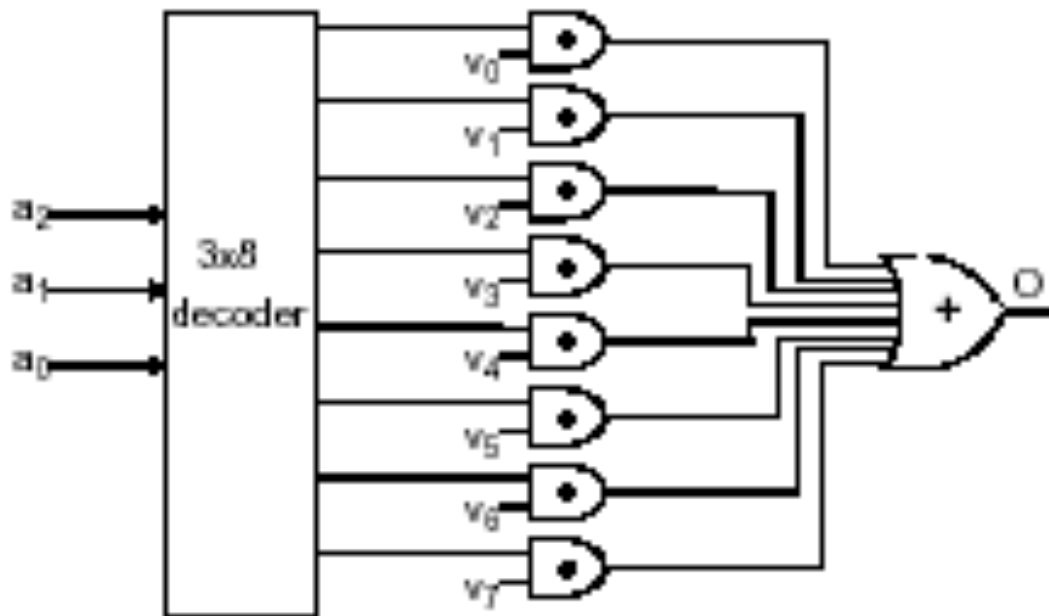
PLA

- in between

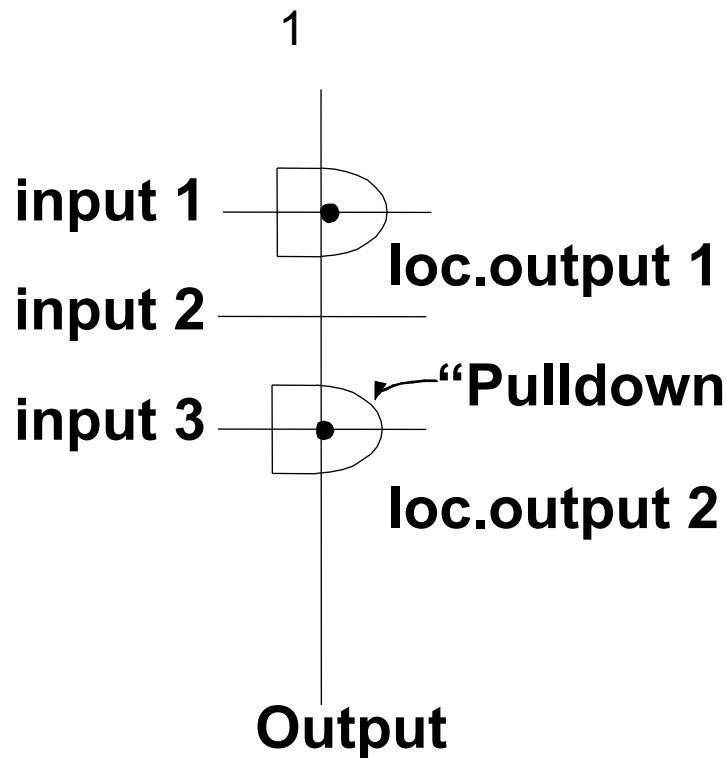
Read Only Memory (ROM)

Map addresses to fixed values

a_1	a_2	a_3	O
0	0	0	V_1
0	0	1	V_2
0	1	0	V_3
0	1	1	V_4
1	0	0	V_5
1	0	1	V_6
1	1	0	V_7
1	1	1	V_8



Modern ROMs



Local outputs are associated with each pull-down

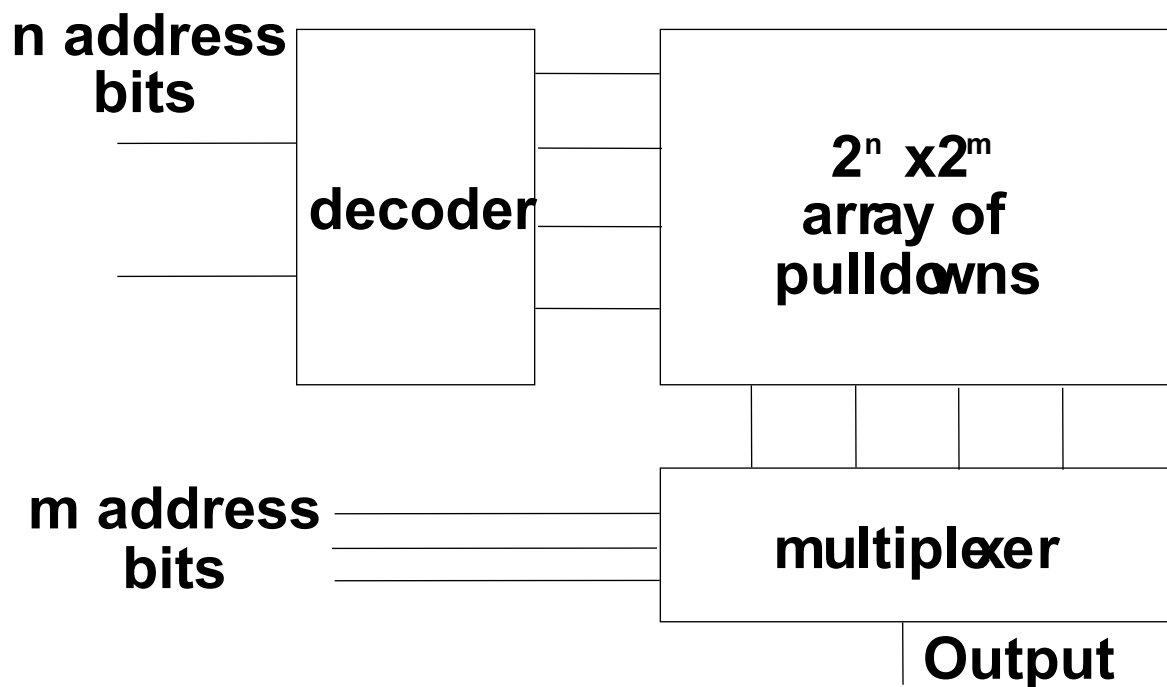
The local outputs are equal to their corresponding inputs

The output is the logical and of the local outputs

Modern ROMs

Modern ROMs are organized around

- a decoder
- a multiplexer
- an array of pulldowns



Example (stupid, I agree)

Assume that I need a ROM to store the prime numbers from 0 to 63.

Given an integer (between 0 and 63), build a circuit that returns 1 if the number is prime and 0 otherwise.

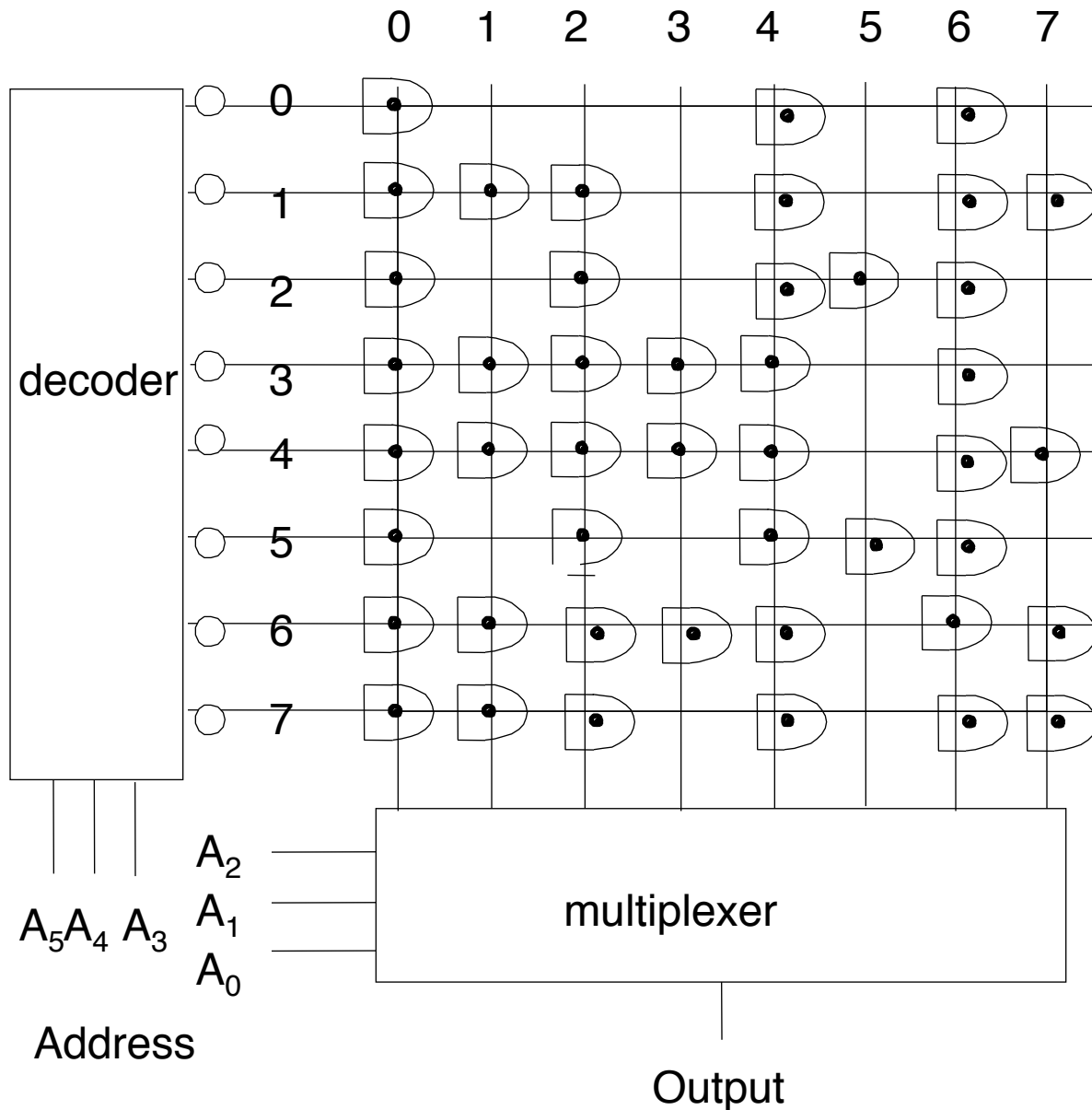
The integer is given as a sequence of bits

$$A_5 A_4 A_3 A_2 A_1 A_0$$

We use an array of 8 by 8

Storing Primes

Put a pulldown where you want a 0.



If you want more than one bit of output, just use more ROM circuits in parallel with different stored functions.