

Some Common Combinational Circuits

MUX (Multiplexor)

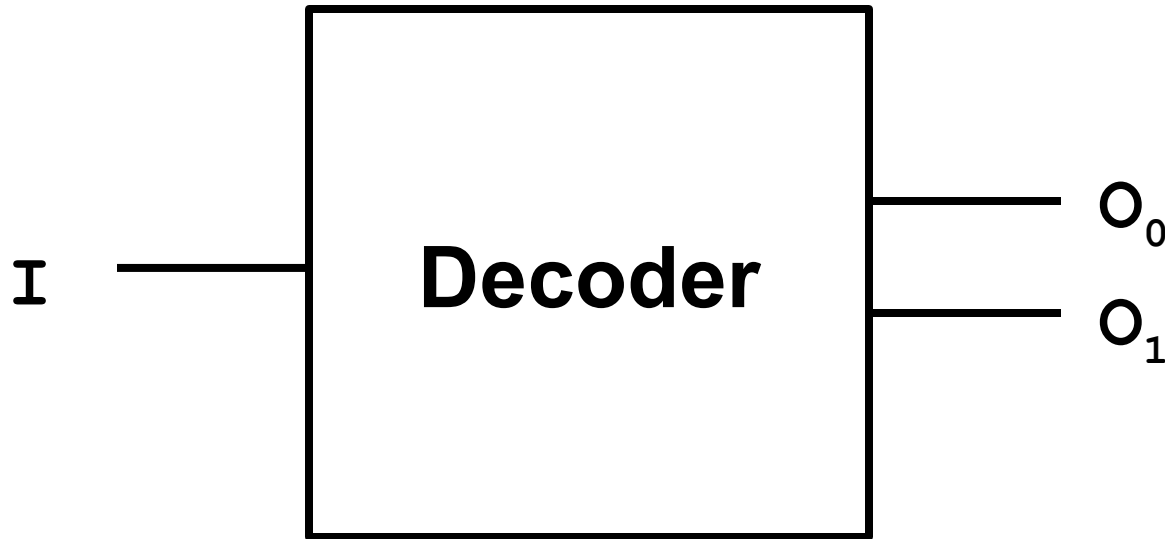
Decoder

Adder

Some commonly used components

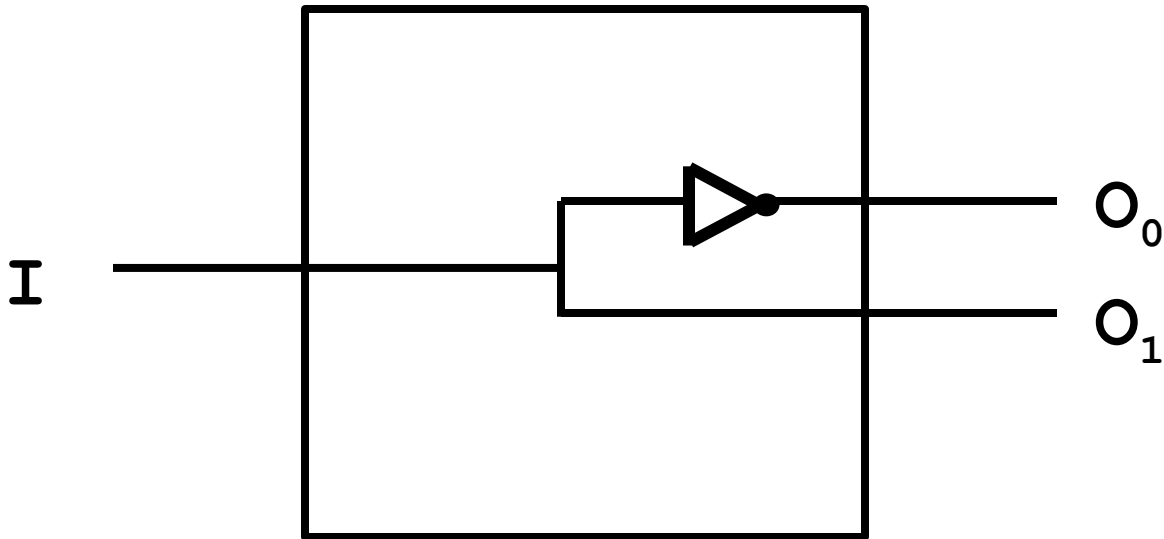
- Decoders: n inputs, 2^n outputs.
 - *the inputs are used to select which output is turned on. At any time exactly one output is on.*
- Multiplexors: 2^n inputs, n selection bits, 1 output.
 - *the selection bits determine which input will become the output.*
- Adder: $2n$ inputs, $2n$ outputs.
 - *Computer Arithmetic.*

1 input Decoder

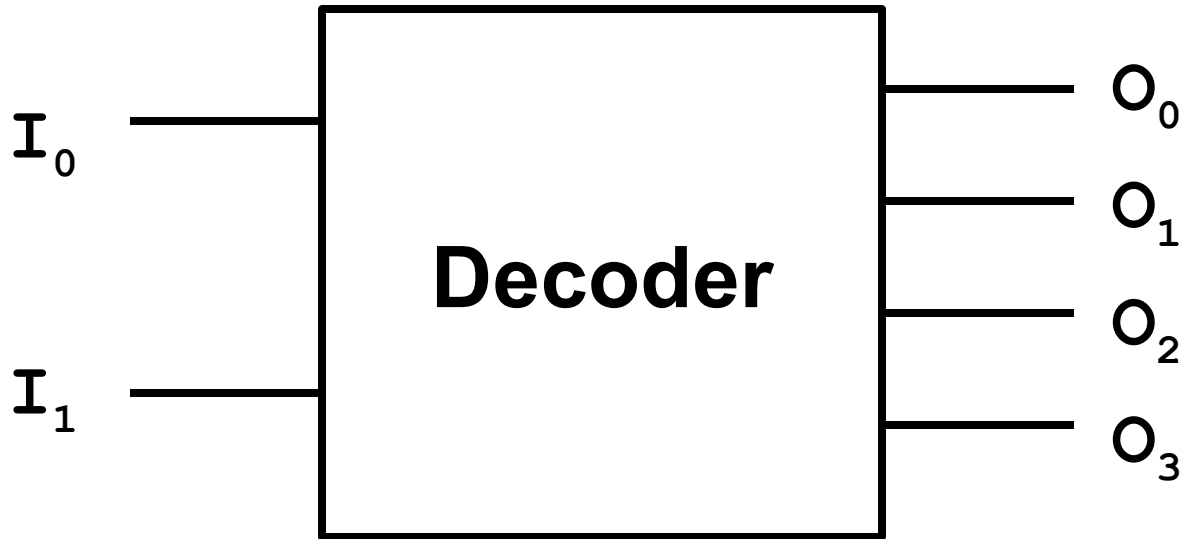


Treat \mathbf{I} as a 1 bit integer i . The \mathbf{i}^{th} output will be turned on ($\mathbf{O}_i=1$), the other one off.

1 input Decoder

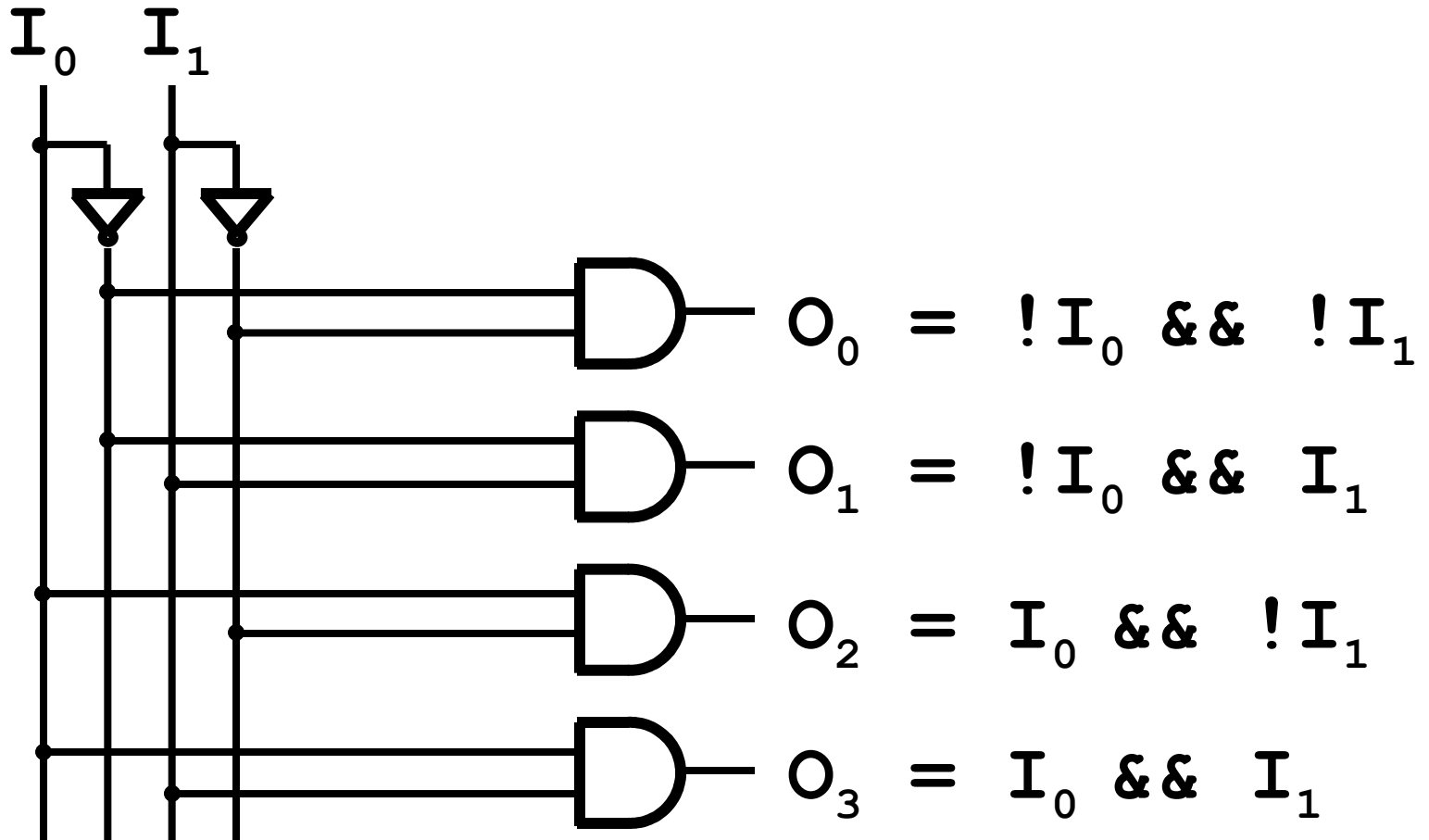


2 input Decoder

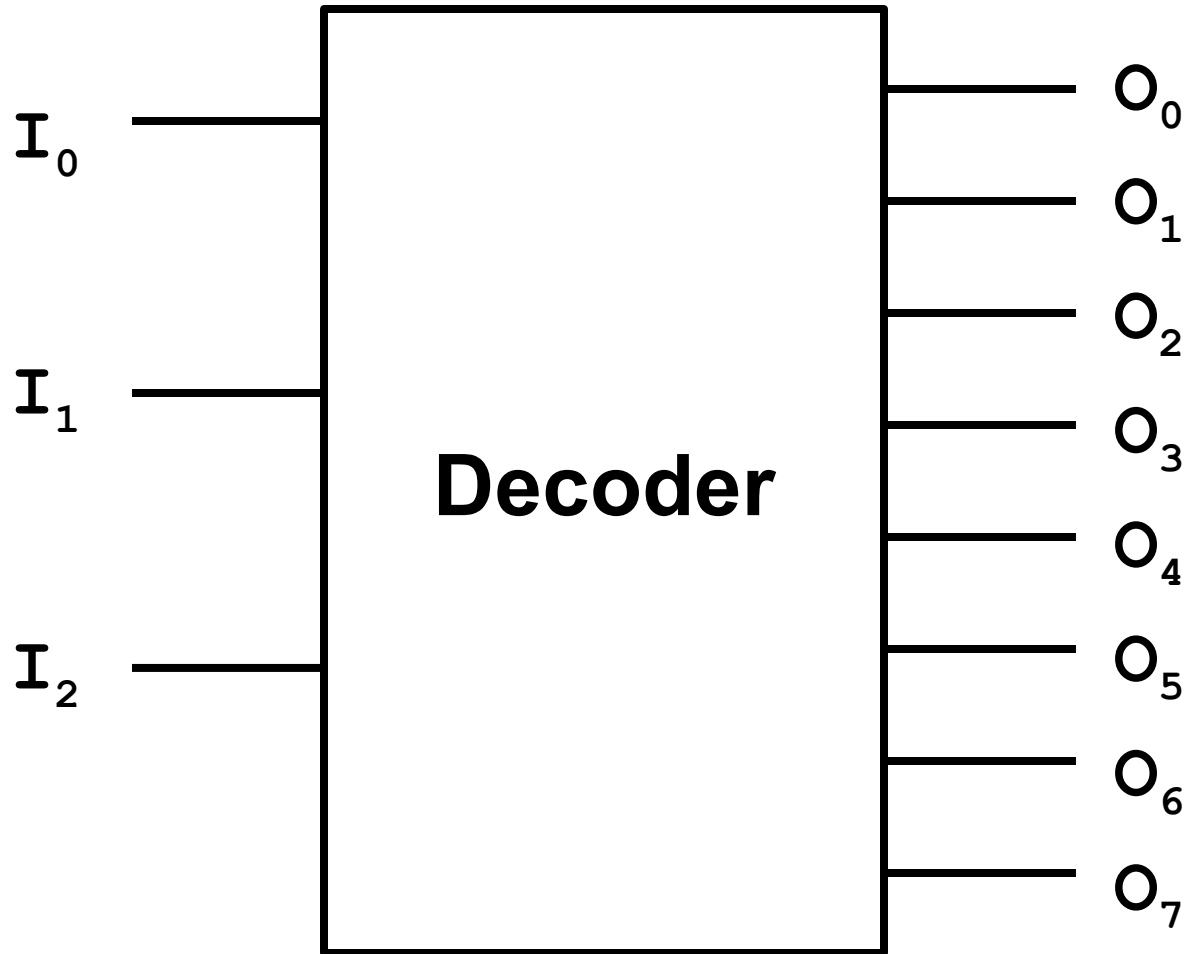


Treat I_0I_1 as a 2 bit integer i . The i^{th} output will be turned on ($O_i=1$), all the others off.

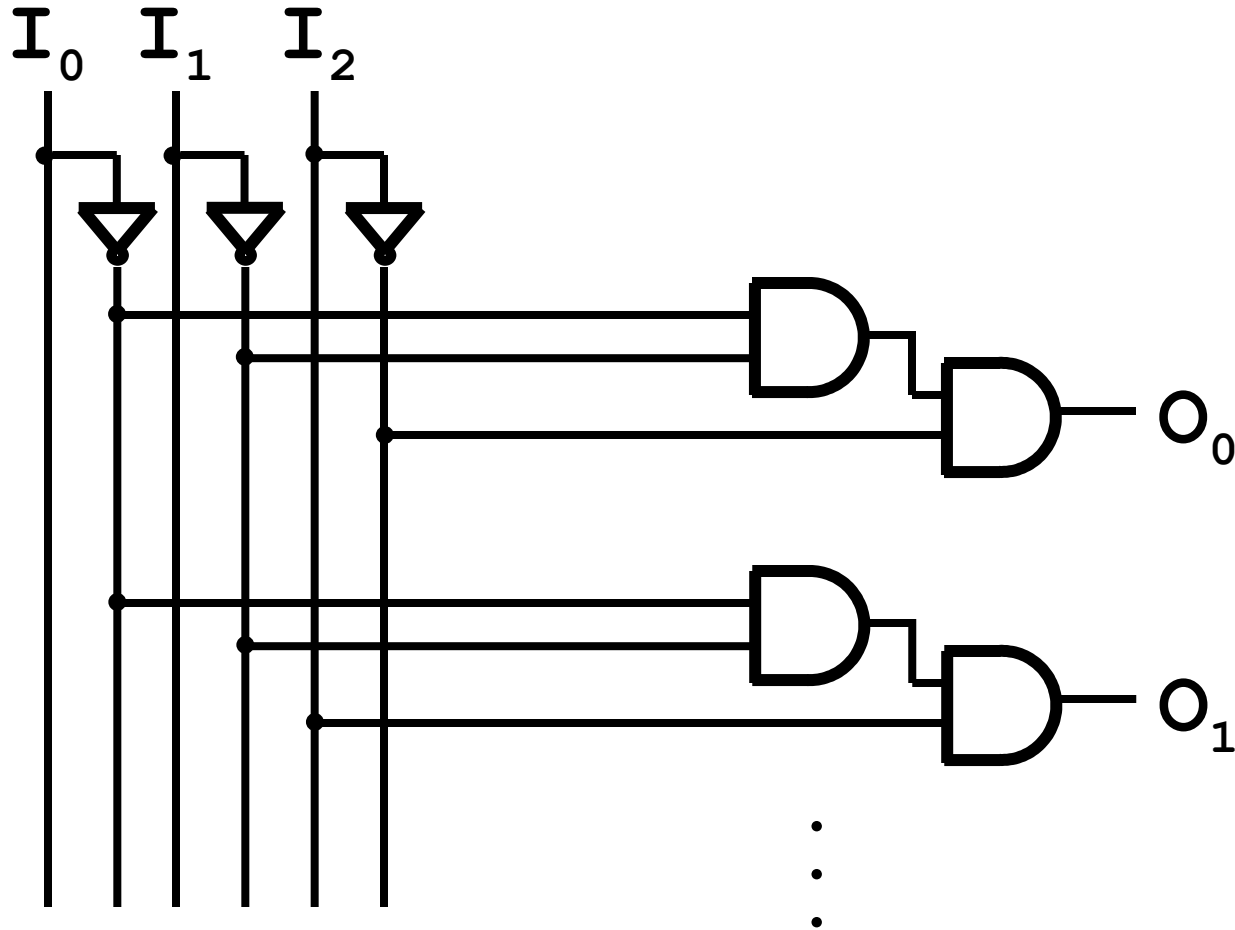
2 input Decoder



3 Input Decoder



3-Decoder Partial Implementation



2 Input Multiplexor

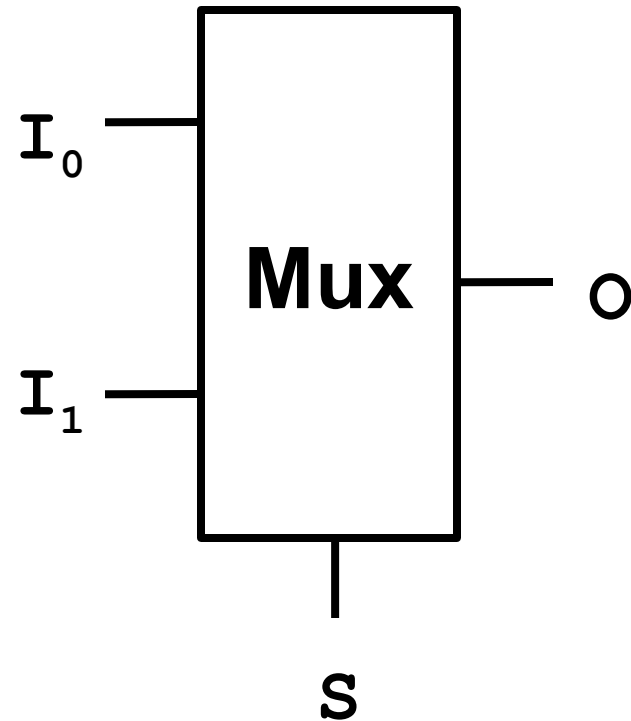
Inputs: I_0 and I_1

Selector: S

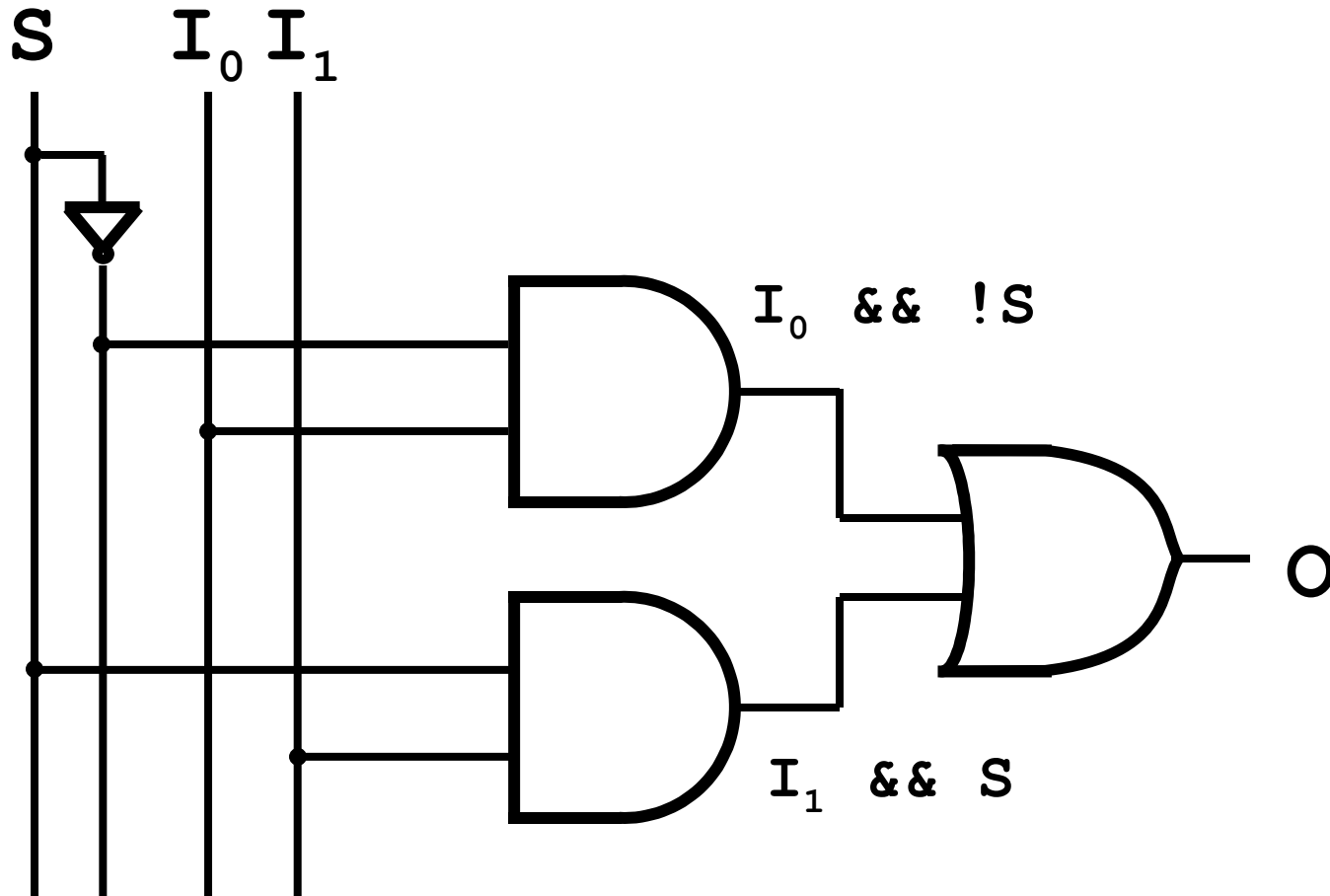
Output: O

If S is a 0: $O=I_0$

If S is a 1: $O=I_1$



2-Mux Logic Design



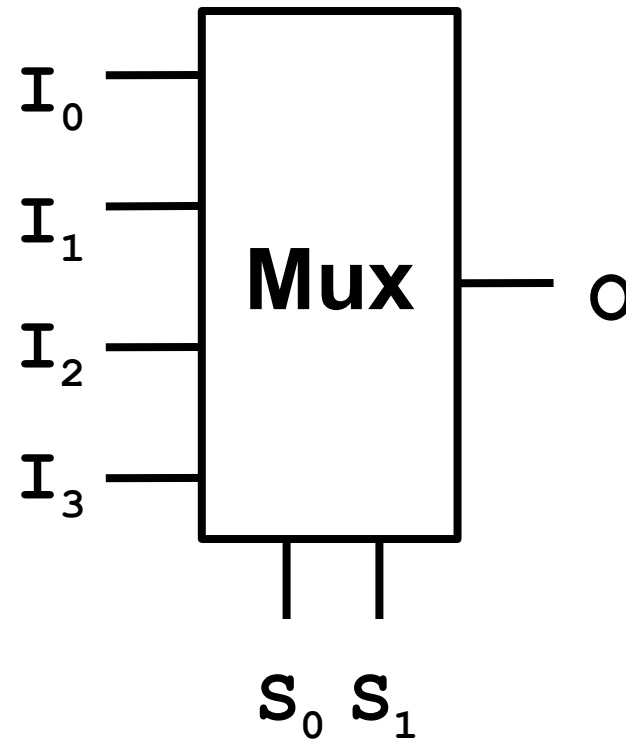
4 Input Multiplexor

Inputs: I_0 I_1 I_2 I_3

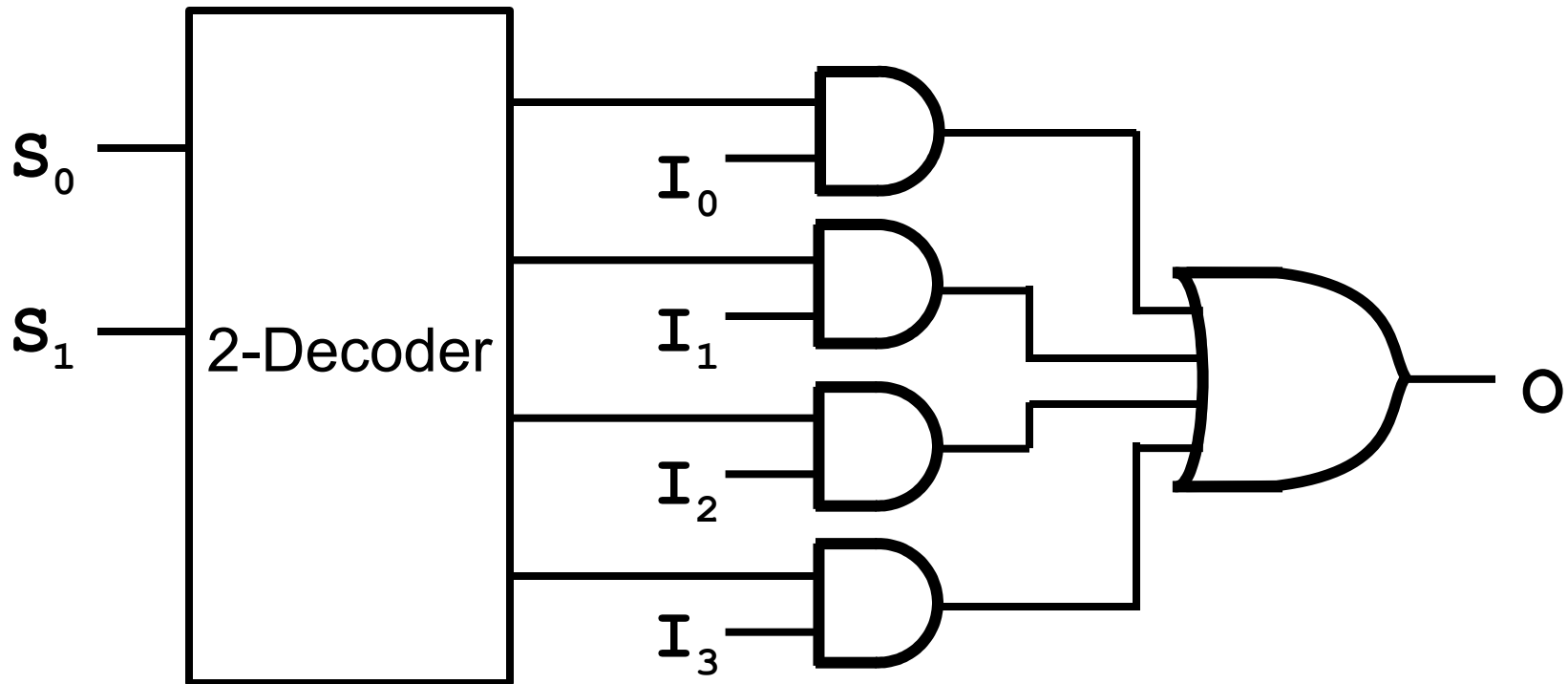
Selectors: S_0 S_1

Output: O

S_0	S_1	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3



One Possible 4-Mux



Adder

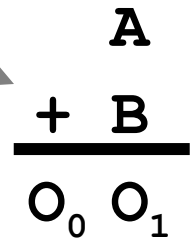
- We want to build a box that can add two 32 bit numbers.
 - Assume 2s complement representation
- We can start by building a *1 bit adder*.

Addition

- We need to build a 1 bit *adder*
 - compute binary addition of 2 bits.
- We already know that the result is 2 bits.

A	B	O_0	O_1
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

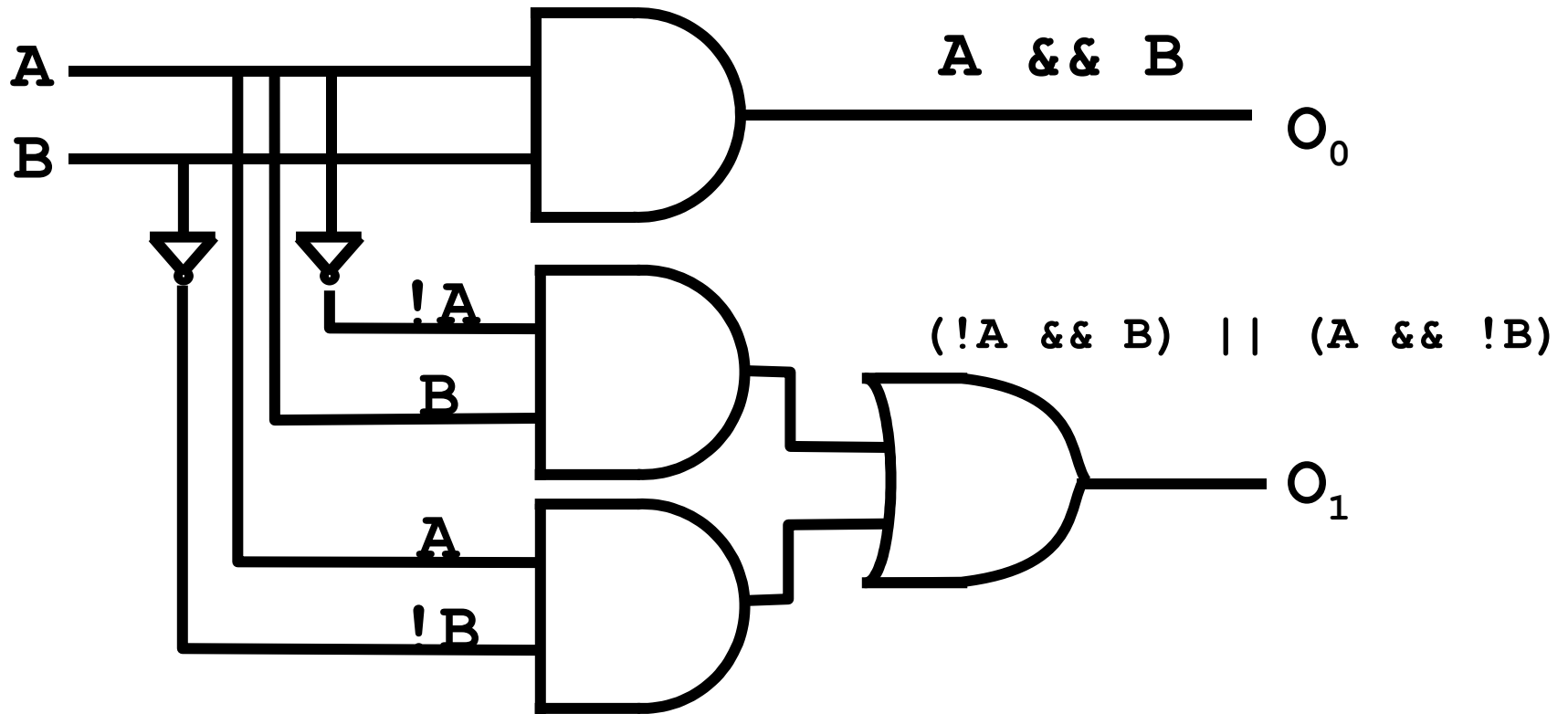
This is addition!



A diagram illustrating the addition of two bits, A and B. The inputs A and B are stacked vertically with a plus sign between them. A horizontal line is drawn below the plus sign. Below the line, the outputs O_0 and O_1 are shown. An arrow points from the text "This is addition!" to the plus sign.

$$\begin{array}{r} \mathbf{A} \\ + \mathbf{B} \\ \hline \mathbf{O}_0 \mathbf{O}_1 \end{array}$$

One Implementation



Binary addition and our *adder*

$$\begin{array}{r} \\ \\ + \\ \hline 1 \\ \end{array}$$

← Carry

What we really want is something that can be used to implement the binary addition algorithm.

- O_0 is the *carry*
- O_1 is the *sum*

What about the second column?

$$\begin{array}{r} \\ \\ + \\ \hline \end{array}$$

1 1 ← Carry

01001

01101

10110

- We are adding 3 bits
 - new bit is the *carry* from the first column.
 - The output is still 2 bits, a *sum* and a *carry*

Truth Table for Addition

A	B	Carry In	Carry Out	Sum
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

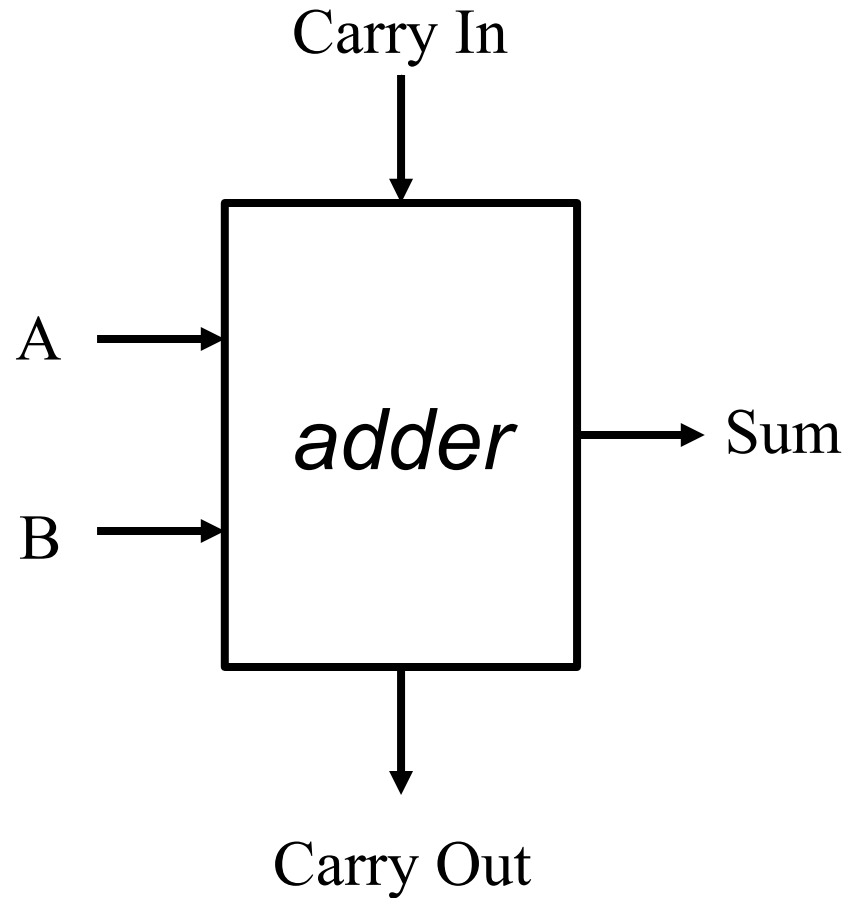
1 bit adder (3 inputs!)

- We can come up with a logic design:

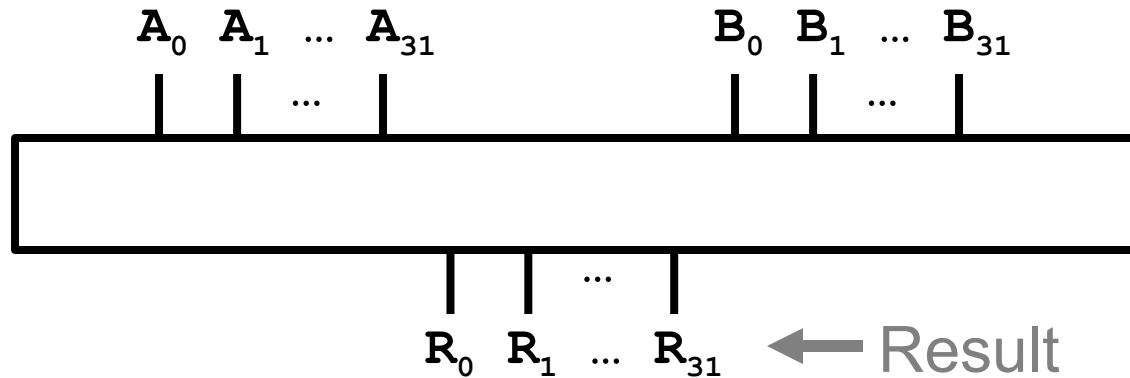
$$\text{Carry Out} = (A \&\& B) \ || \ (A \&\& \text{CarryIn}) \ || \ (B \&\& \text{CarryIn})$$

$$\begin{aligned} \text{Sum} = & (!A \ \&\& \ !B \ \&\& \ \text{CarryIn}) \ || \\ & (!A \ \&\& \ B \ \&\& \ !\text{CarryIn}) \ || \\ & (A \ \&\& \ !B \ \&\& \ !\text{CarryIn}) \ || \\ & (A \ \&\& \ B \ \&\& \ \text{CarryIn}) \end{aligned}$$

New Component: 1 Bit Adder



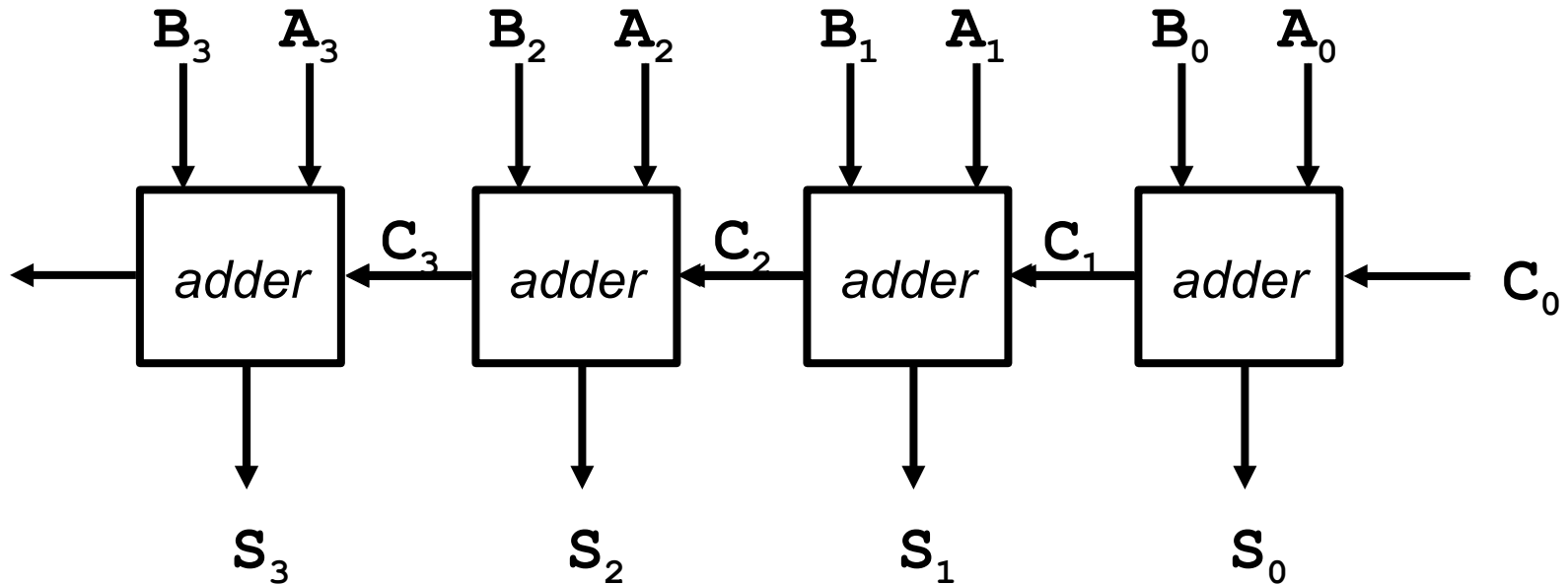
Building a 32 bit Adder



- 64 inputs
- 32 bit output

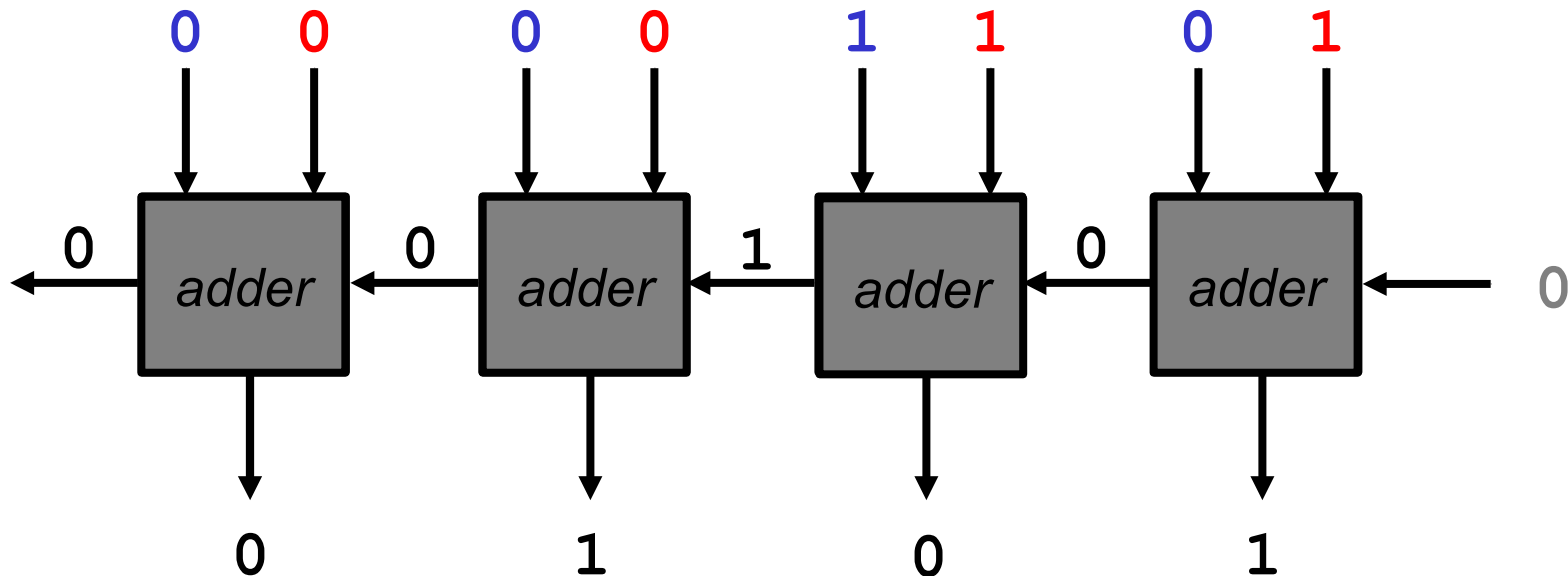
4 Bit *Ripple Carry* Adder

$$\begin{array}{r} C_3 \ C_2 \ C_1 \ C_0 \\ A_3 \ A_2 \ A_1 \ A_0 \\ + \ B_3 \ B_2 \ B_1 \ B_0 \\ \hline S_3 \ S_2 \ S_1 \ S_0 \end{array}$$



4 Bit Ripple Carry Adder

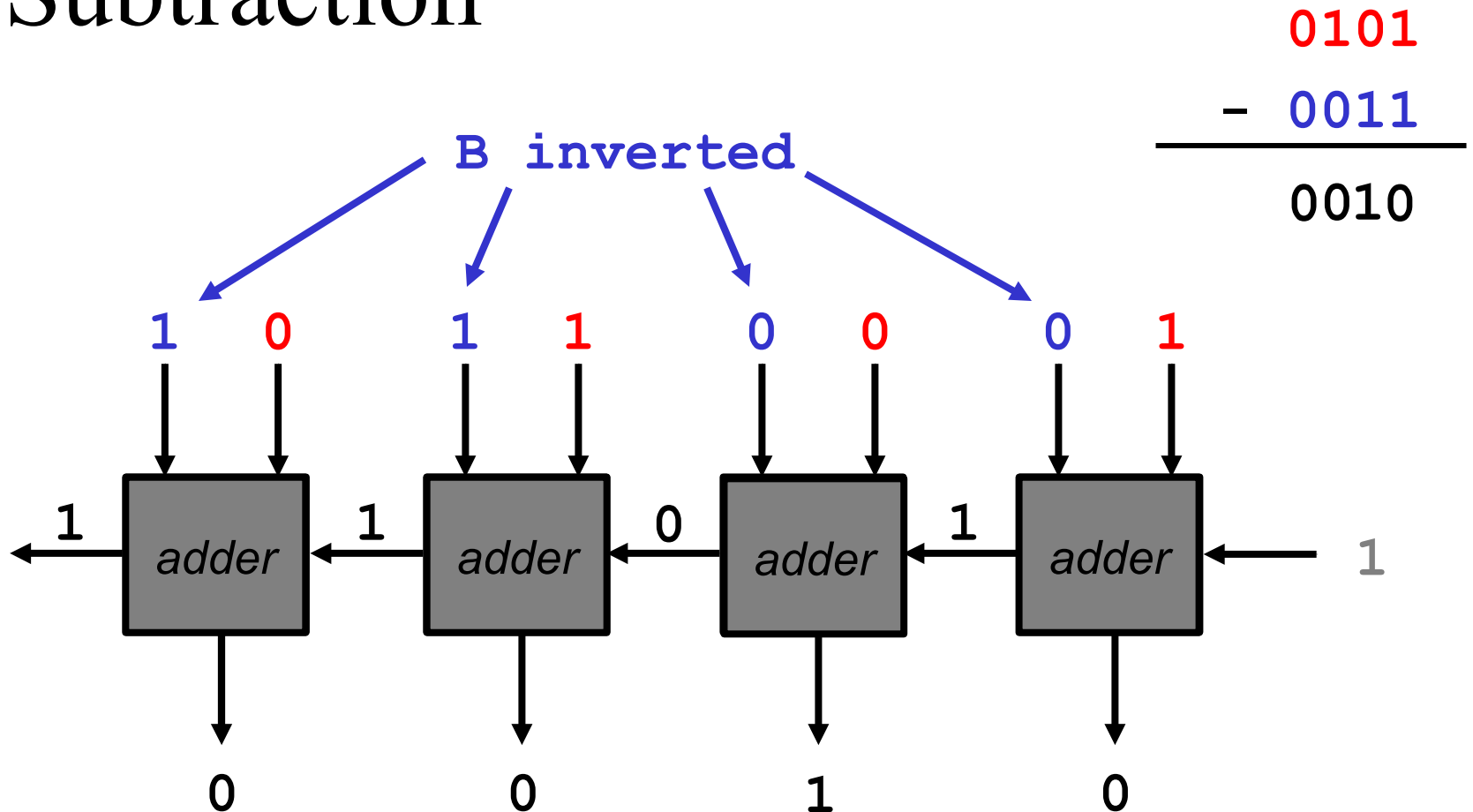
$$\begin{array}{r} 010 \\ 0011 \\ + 0010 \\ \hline 0101 \end{array}$$



Subtraction

- Compute $A-B$ as $A + (-B-1) + 1$
- $-B-1$ is just all the bits of B inverted.
- Add the $+1$ by setting C_0 to 1

Subtraction

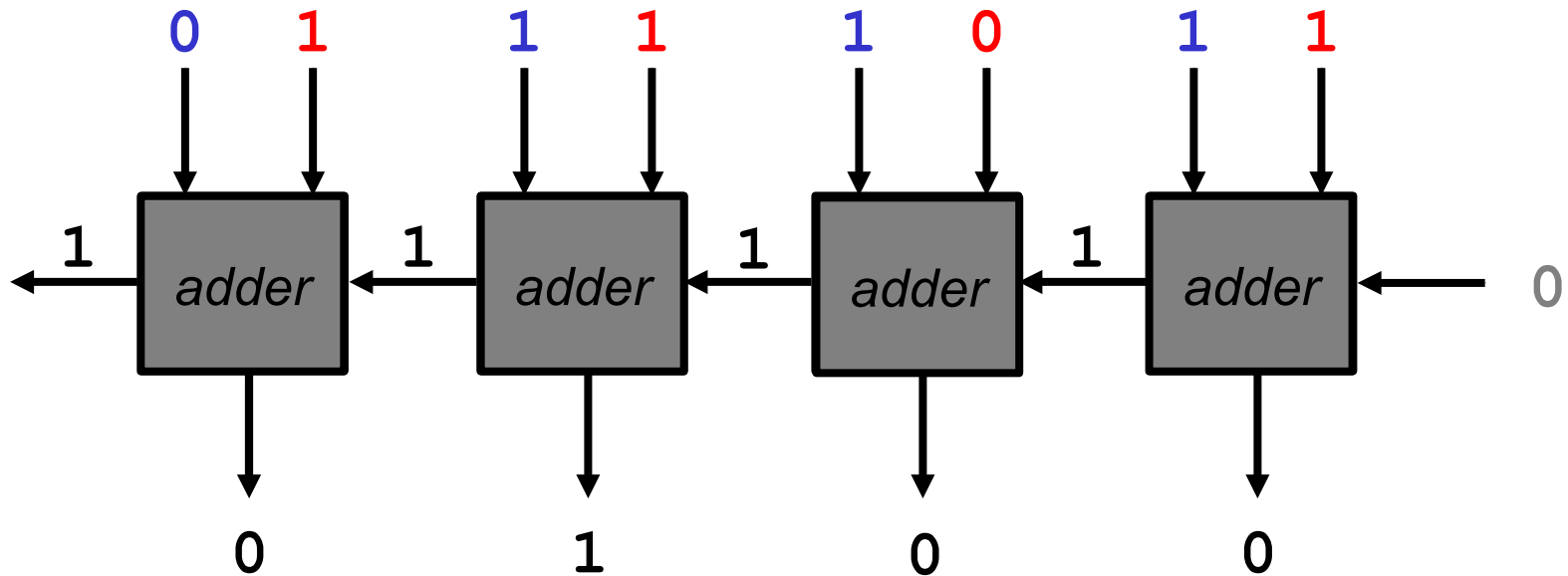


Two's Complement Numbers

- Nothing is different!
 - This is the advantage of using 2's complement representation.
- Overflow:
 - For addition: sign of the result is different than the sign of the operands (and they have the same sign).

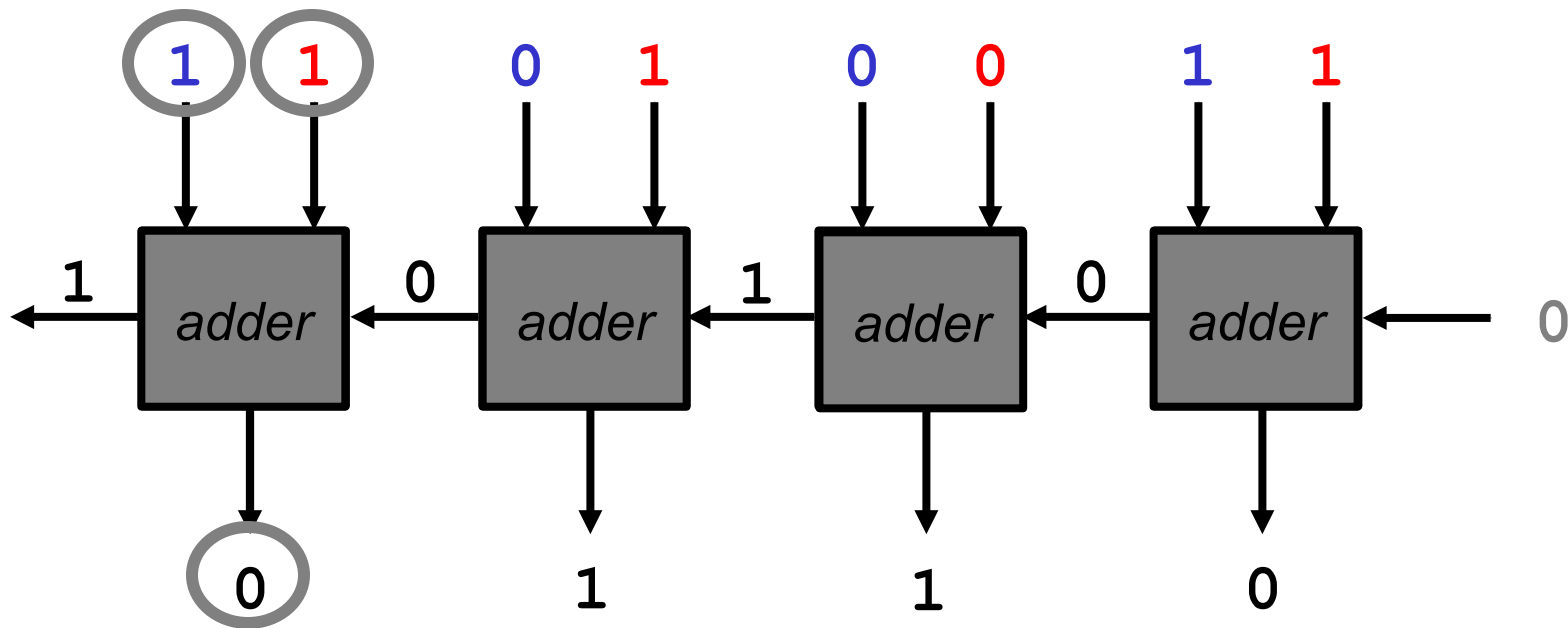
$$-3 + 7$$

$$\begin{array}{r} 111 \\ 1101 \\ + 0111 \\ \hline 0100 \end{array}$$



$$-3 + -7$$

$$\begin{array}{r} 001 \\ 1101 \\ + 1001 \\ \hline 0110 \end{array}$$



Overflow!

Ripple Carry Timing

- All the adders are actually operating all the time (they are just combinational circuits).
- We wait long enough (until the last carry has been computed) and then pay attention to the complete answer.
- It is likely that there are intermediate values that are wrong!

Carry Look-ahead

- Compute the carry bits right away.
 - As a function of the inputs A and B .
- Not possible for a large adder (32 bit), but realistic for a 4 bit adder.

4 Bit Carry Look-ahead

$$\begin{array}{r} 010 \\ 0011 \\ + 0010 \\ \hline 0101 \end{array}$$

