COMSM1302
Overview of Computer Architecture

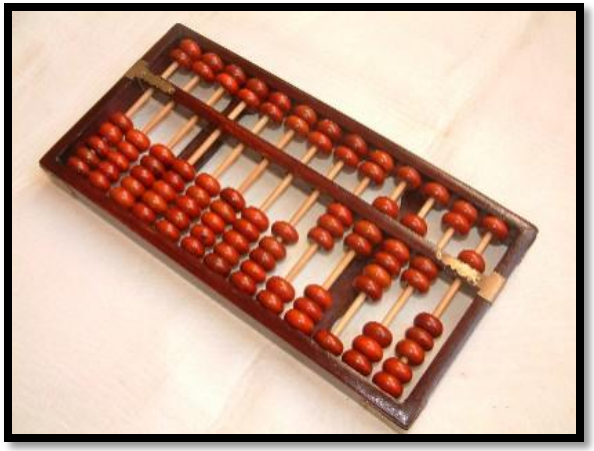
Lecture 4 – Simple devices

In this lecture

Foundations	• Data representation, logic.
Building blocks	• Transistors, transistor based logic, **simple devices**, storage.
Modules	• Hex modules, memory, simple controller and processor.
Programming	• Assembly, assembler, language, compilation phases, boot-strapping.
Bigger systems	• ARM & Thumb, I/O, protecting shared systems, memory hierarchy, multi-processors, networks.
Wrap-up	• More examples, historical computers, contemporary systems.

Today, we learn to add!

- And also…
  - Subtract
  - Select 1 signal from many
  - Distribute 1 signal to many
- The circuits shown hereafter will be drawn in Logisim.
  - You can download and try yourself.
  - And you will need to in the first lab!



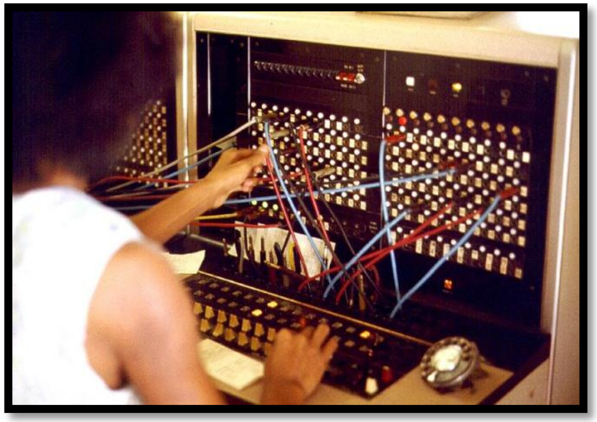


Photo by Joseph A. Carr, 1975

The simplest of binary addition

- How to add two, single-digit binary numbers?
  - Each digit is either 0 or 1
  - There are three possible results
    - 0b00, 0b01, 0b10
    - 0, 1, 2
  - Two inputs
    - A, B
  - Two outputs
    - Sum (S), carry (C)

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

With some Boolean Algebra

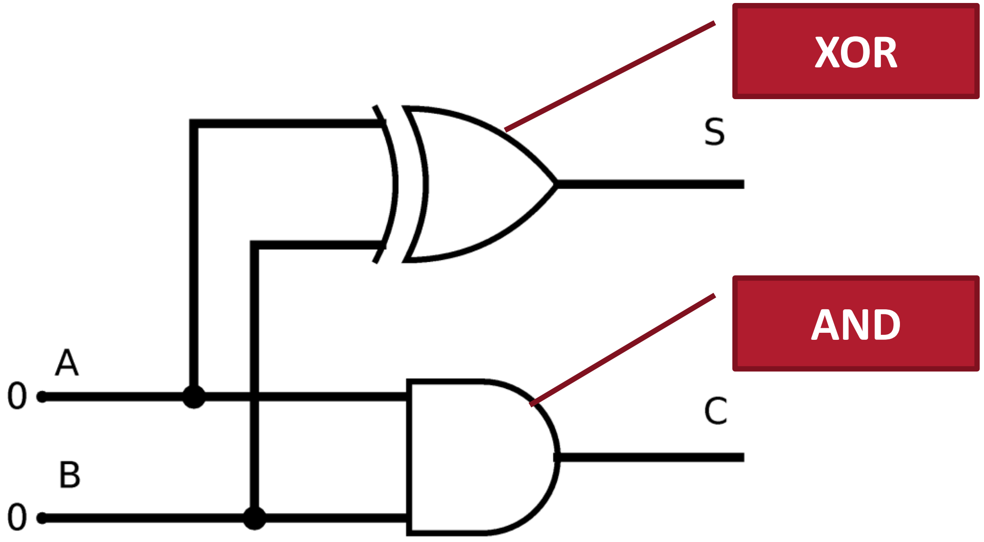
- Remember our truth tables for Boolean Algebra…
- Which operations can be used to help us generate S and C?

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

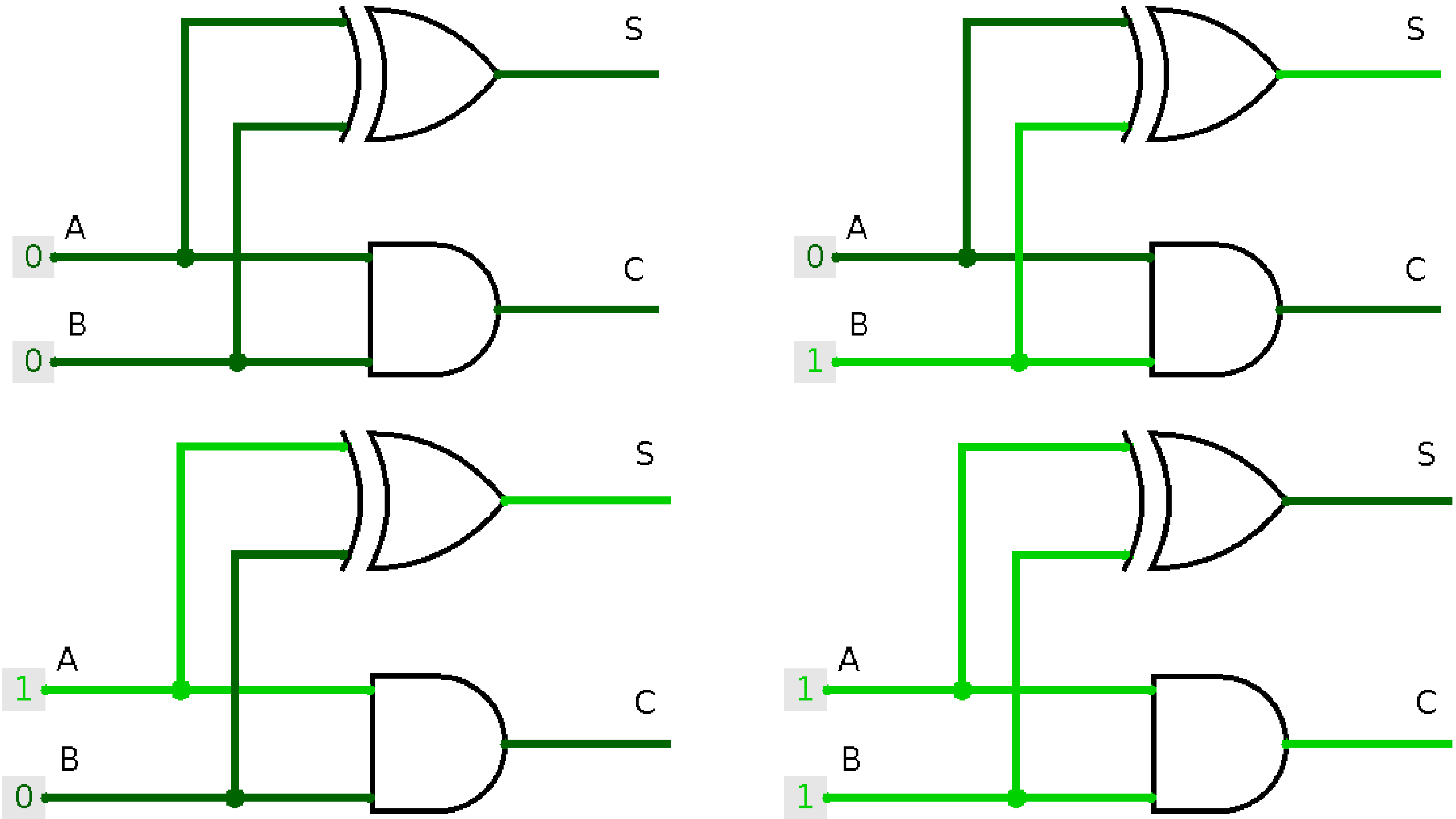
A	B	¬A	A ∧ B	A ∨ B	A ⊕ B	A → B	A≡B
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	0	0
1	1	0	1	1	0	1	1

The half-adder

- $S = A \oplus B$
- $C = A \wedge B$
- Now let's build it with logic gates.



The half-adder in action



Now what?

- We can add two bits together.
  - By generating a sum and a carry bit.
- How do we add multiple bits together?

Carry example on the board

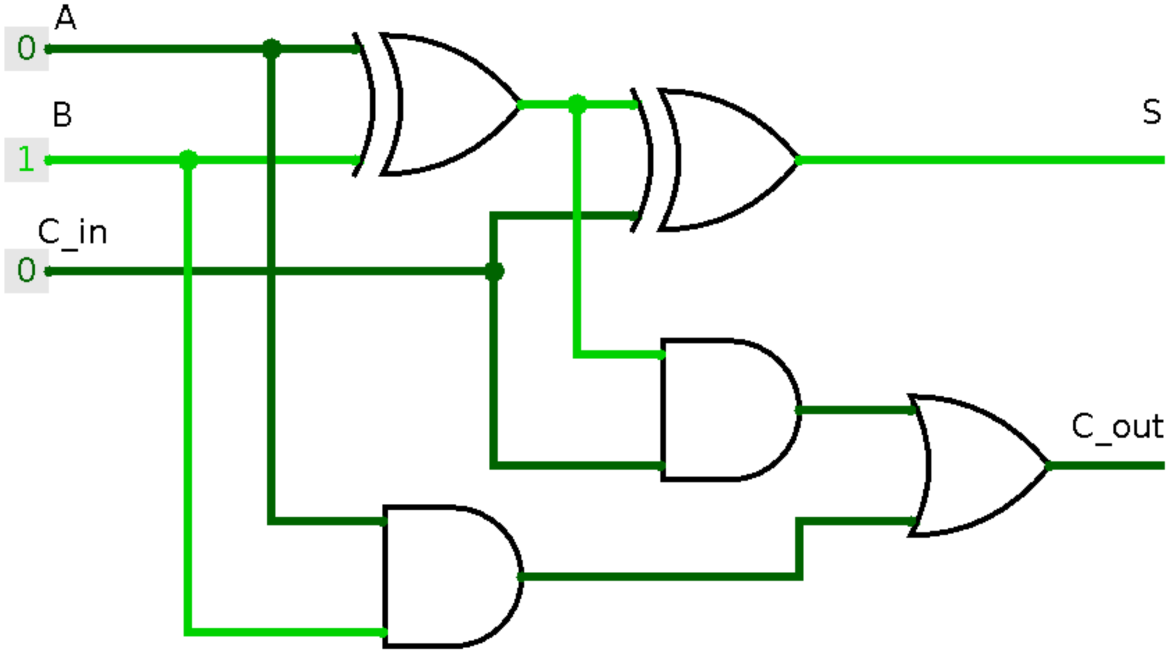
The full-adder

C_in	A	B	S	C_out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

- $S = A \oplus B \oplus C\_in$
  - **Explanation on board**
- $C\_out =$
  $(A \land B) \lor (C\_in \land (A \lor B))$
  - **Explanation on board**
- Also valid for C_out
  - $(A \land B) \lor (C\_in \land (A \oplus B))$
  - **Why?**

The full-adder

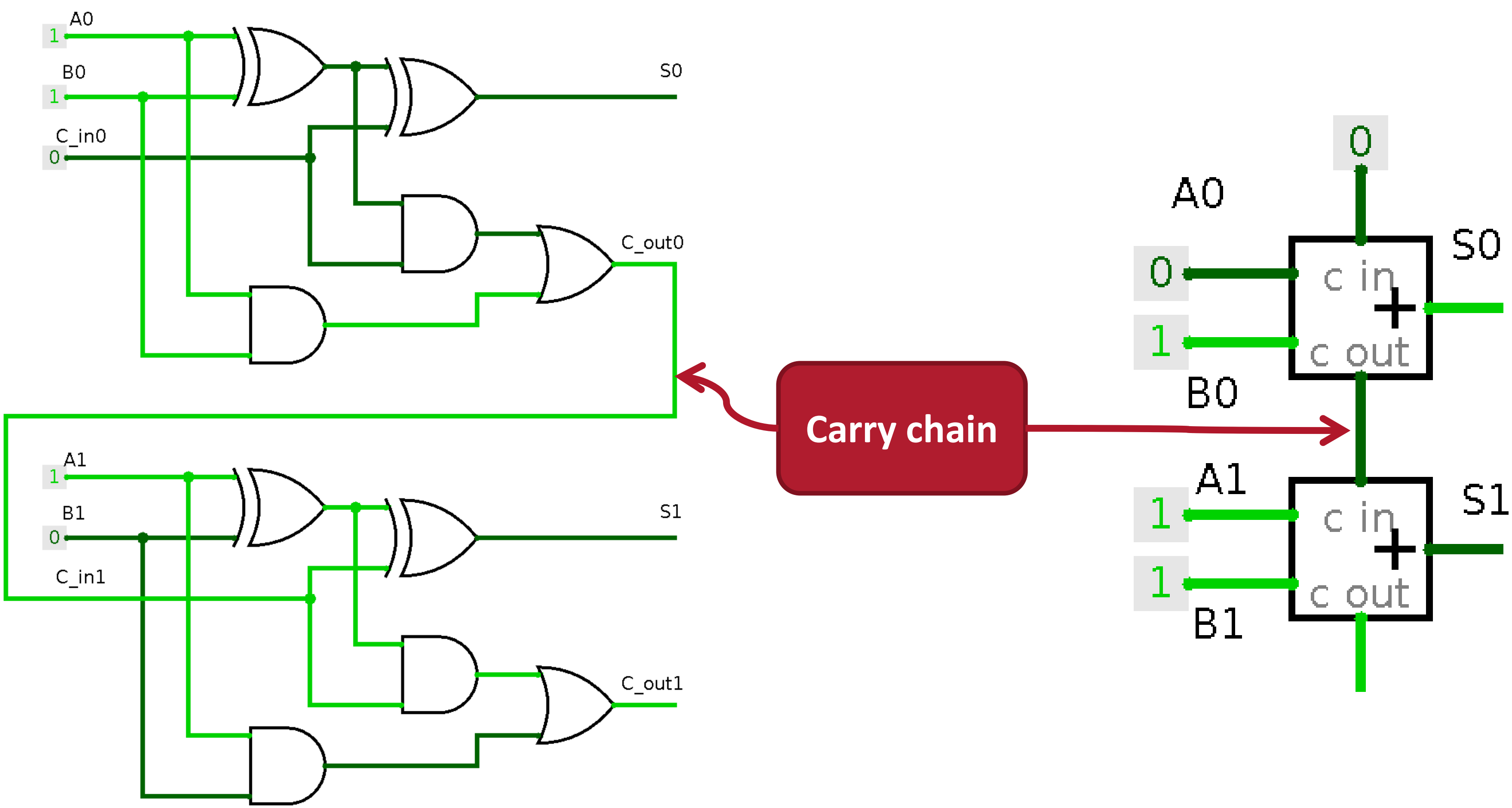
- 8 different combinations of input
- Try them yourself!



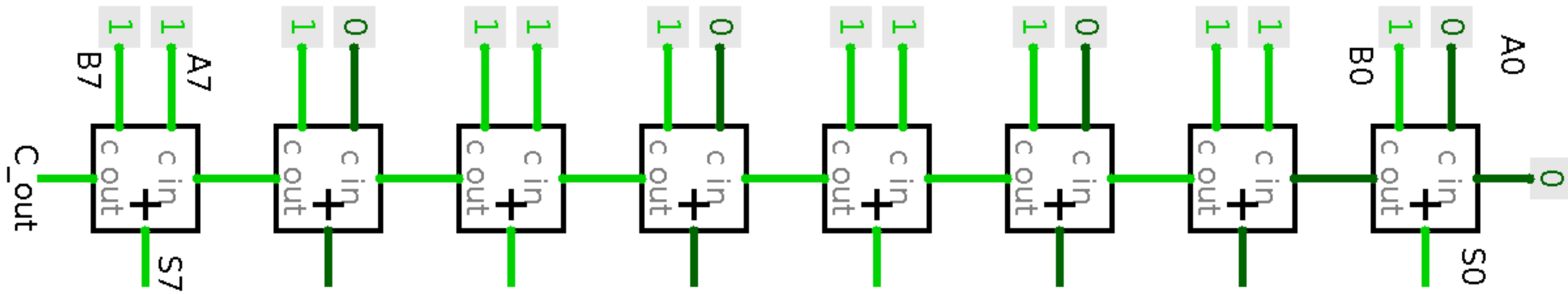
Now what?

- We can add two bits together.
  - By generating a sum and a carry bit.
- We can add three bits together
  - By accommodating a carry-in as well as our regular two inputs.
- How do we add multiple bits together?

Chaining full-adders



8-bit adder, ripple-carry adder



- Named **ripple-carry** because a carry signal generated at the LSB (Least Significant Bit… bit 0) of the device can affect the result on any/all more significant bits.
- Does this have any performance implications?

Building blocks

- To recap what we've done:
  - Used Boolean Algebra as **building blocks**.
  - To build a unit capable of **adding two bits**.
    - **Half-adder**
  - Extended it to handle **carry-in**.
    - **Full-adder**
  - Chained them together to make an adder of **arbitrary size**.
    - **Ripple-carry adder**
- Now we can add anything!
  - Modern processors typically have adders between 8 and 64 bits.
  - **Why?**

Subtraction

- Subtraction is easy if we think of it as **adding one number to a negative number**.
- So let's represent this subtraction:

  1 – 2 = -1

- As:

  1 + -2 = -1

- How to negate a number?
  – 2s complement!

Reminder: 2s complement

$$Y = -x_{N-1} \cdot 2^{N-1} + \sum_{i=0}^{N-2} x_i \cdot 2^i$$

-128	64	32	16	8	4	2	1	Base 10
0	0	0	0	1	0	1	0	**10**
1	0	0	0	1	0	1	0	**-118**

Negating in 2s complement

- Flip the bits and add one.

Example on board

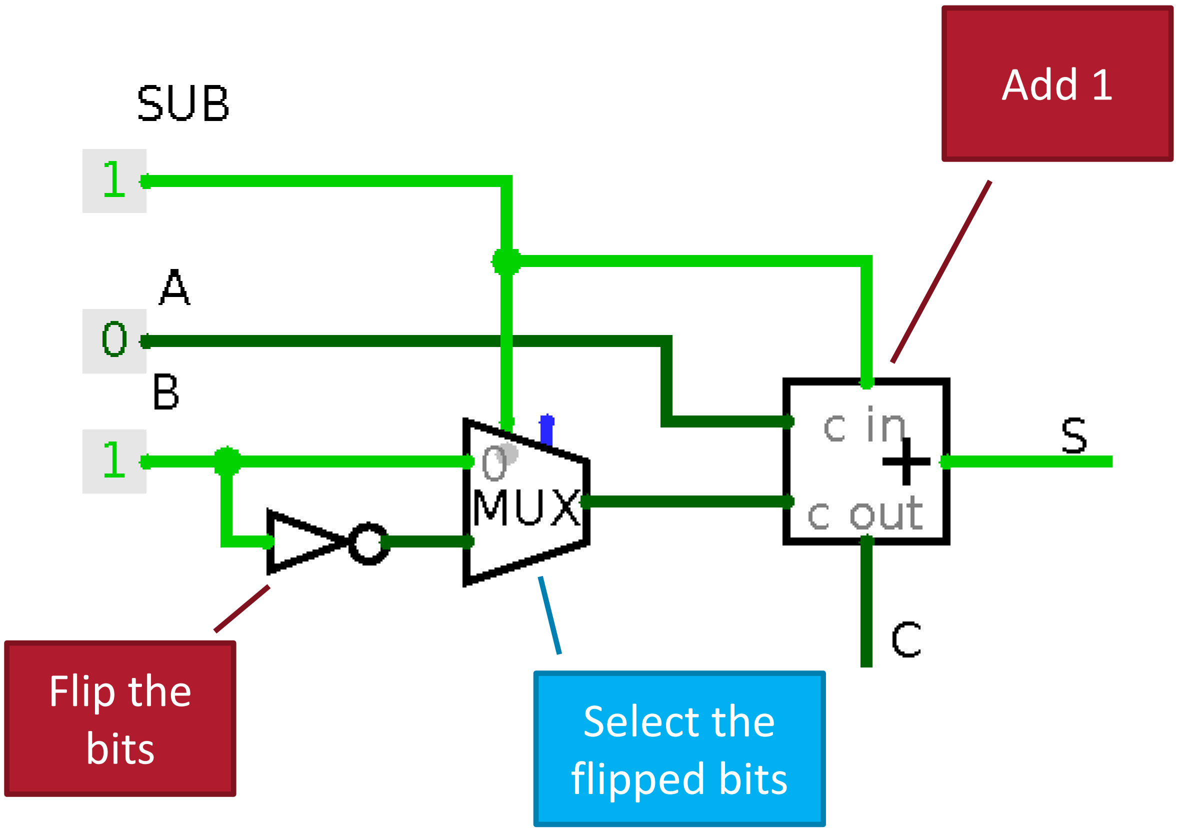
- A – B = A + (Not(B) + 1)
- We already have all the tools we need for this!
  - NOT gates to flip bits
  - An unused C_in at the beginning of our adder's carry chain.

The adder-subtractor?

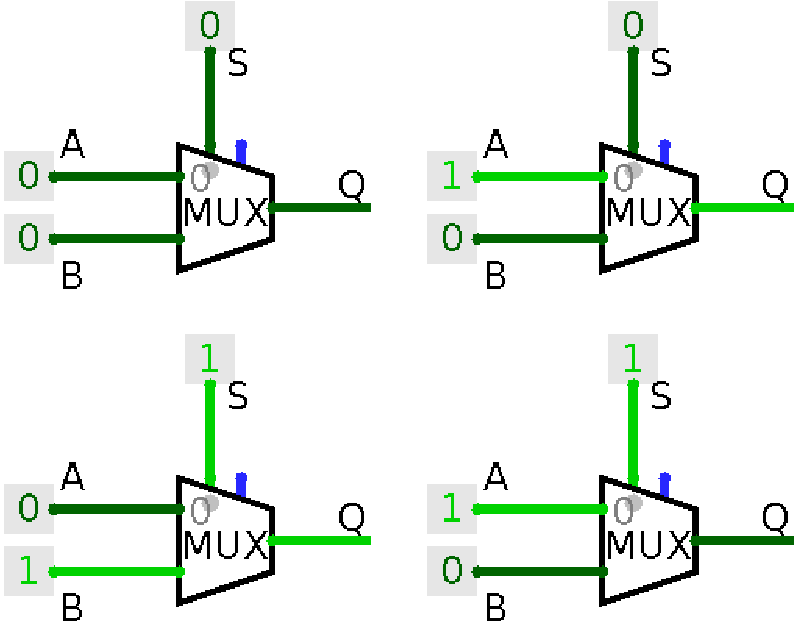
- We can build an adder **or** a subtractor.
- They are **very similar**.
- Can we build one unit that does **both**?

- Almost…

Adder-subtractor



Choosing a signal



S	A	B	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Choosing a signal

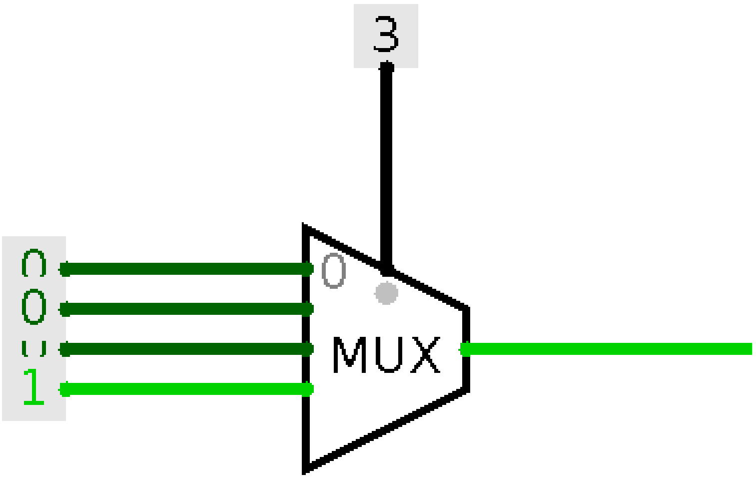
- $(A \land \neg S) \lor (B \land S)$
- Consider $S = 0$
  - $(A \land 1) \lor (B \land 0)$
  - $A \lor 0$
  - A
- Consider $S = 1$
  - $(A \land 0) \lor (B \land 1)$
  - $0 \lor B$
  - B

S	A	B	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

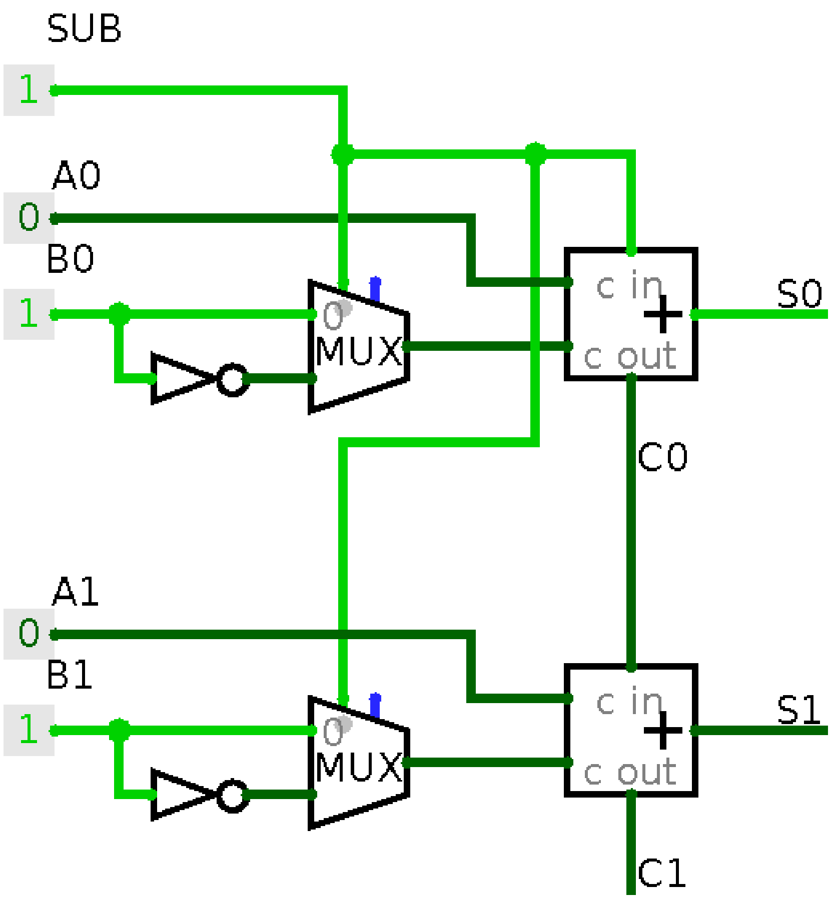
The multiplexer

- S selects which input to propagate to the output.
- 2-to-1 multiplexer
  – 1 select bit
- **N**-to-1 multiplexers also possible
  – $\text{Log}_2(N)$ select bits needed

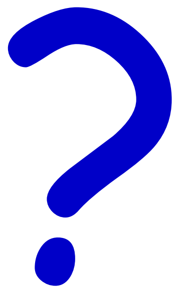
2-bit signal (imagine a bundle of two wires)



Ripple-carry adder-subtractor



Demultiplexer



- 1-to-2 demultiplexer
  - Choose **which of 2 wires** to **propagate** the input signal onto.
  - $Q0 = A \land \neg S$
  - $Q1 = B \land S$
- Instead of choosing which signal to select, we choose **where to send** a signal.
- 1-to-N demux has more logic on the select inputs.
  - That logic on its own is useful.
  - **Why?**

Summary

- Used Boolean Algebra to build **four simple devices**:
  - Demux, Mux, Adder, Subtractor.
- Combined mux, NOT gate and adder to build **adder-subtractor**.
- We can do **basic arithmetic** with a **bunch of NAND gates**!

Imagine if we could **store** the results of that arithmetic, somehow…

Next lecture

Foundations	• Data representation, logic.
Building blocks	• Transistors, transistor based logic, simple devices, **storage**.
Modules	• Hex modules, memory, simple controller and processor.
Programming	• Assembly, assembler, language, compilation phases, boot-strapping.
Bigger systems	• ARM & Thumb, I/O, protecting shared systems, memory hierarchy, multi-processors, networks.
Wrap-up	• More examples, historical computers, contemporary systems.