8. Floating point representation Combinational Logic and Adders

EECS 370 – Introduction to Computer Organization - Winter 2016

Profs. Valeria Bertacco & Reetu Das

EECS Department
University of Michigan in Ann Arbor, USA

Announcements

- Homework 2 extension: it is now due on Wednesday 2/3
- Project 1 is due Thursday 2/4

Recap:

… we mostly talked about:

- Linker and loader
- Object files
  - Symbol table
  - Relocation table
- Floating point arithmetic

CLASS PROBLEM: Convert 8.125 to floating point

Floating Point Multiplication

- Add exponents (don’t forget to account for the bias)
- Multiply significands (don’t forget the implicit **1** bits)
- Renormalize if necessary
- Compute sign bit (simple exclusive-or)

Floating Point Multiply

$$10.625_{10} = 1010.101_2$$

$$10_{10} = 1010_2$$

0	10000010	01010100000000000000000
	+	×
0	10000010	01000000000000000000000
	-127	
0	10000101	10101001000000000000000

$$\begin{array}{r} 1\,0\,1\,0\,1\,0\,1 \\ \times \qquad 1\,0\,1 \\ \hline 1\,0\,1\,0\,1\,0\,1 \\ 1\,0\,1\,0\,1\,0\,1\,0\,0 \\ \hline 1\,1\,0\,1\,0\,1\,0\,0\,1 \end{array}$$

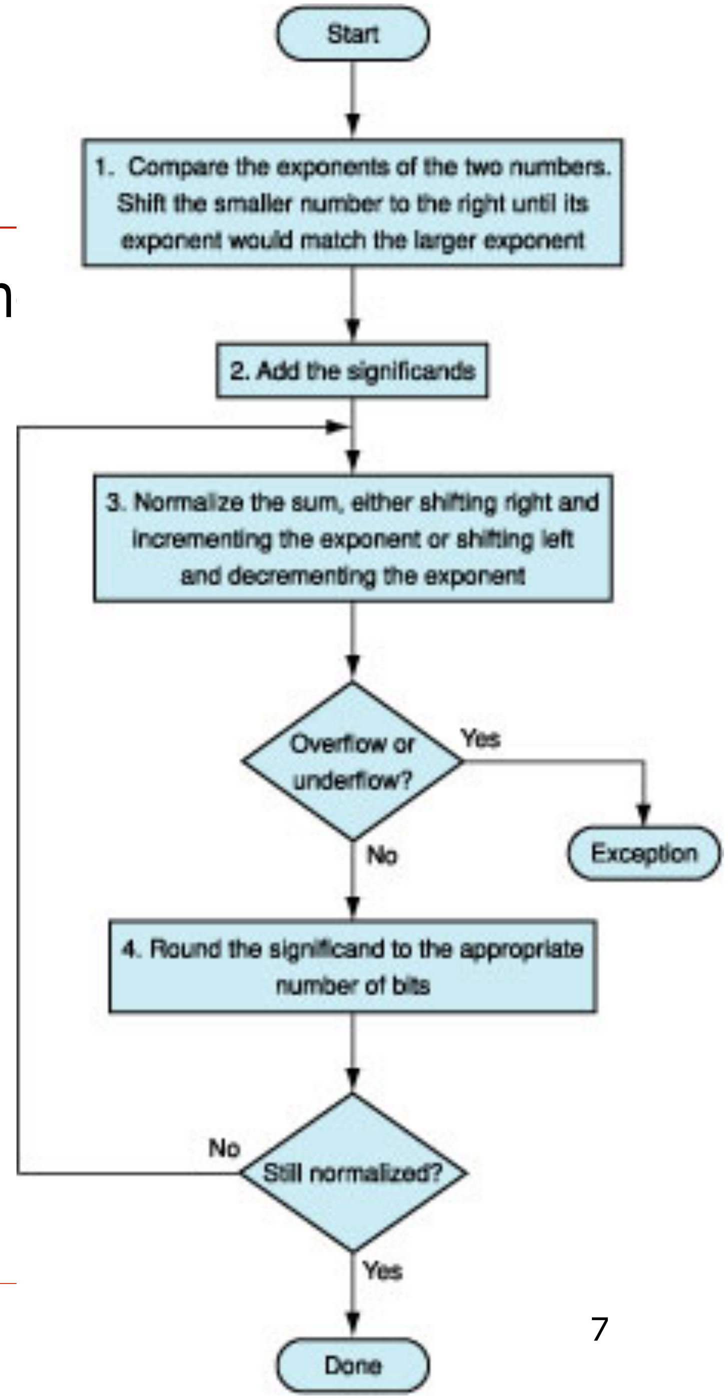
$$1101010.01_2$$
$$= 106.25_{10}$$

Floating Point Addition

- More complicated than floating point multiplication!
- If exponents are unequal, must shift the significand of the smaller number to the right to align the corresponding place values
- Once numbers are aligned, simple addition (could be subtraction, if one of the numbers is negative)
- Renormalize (which could be messy if the numbers had opposite signs; for example, consider addition of +1.5000 and – 1.4999)
- Added complication: rounding to the correct number of bits to store could denormalize the number, and require one more step

Floating Point Addition

1. Shift smaller exponent number significan right to match larger.
1. Add significands.
2. Normalize and update exponent.
3. Check for “out of range”.



Class Problem 2

Show how to add the following 2 numbers using IEEE floating point addition: 100.125 + 13.75

Class Problem 2

101.125	0	10000101	10010100100000000000000
13.75	0	10000010	10111000000000000000000

Shift by 6-3 = 3

Shift mantissa by difference in exponent

00**1**10111000000000000000

Note: When shifting to the right, the first shift should put the implicit 1, then 0's

Sum Mantissa's

```
  1 0 0 1 0 1 0 0 1
+ 0 0 1 1 0 1 1 1 0
-------------------
  1 1 0 0 1 0 1 1 1
```

Sum didn't overflow, so no re-normalization needed

0	10000101	11001011100000000000000

= 114.875

Class Problem 3

Show how to add the following 2 numbers using IEEE floating point addition: 117.125 + 13.75

Class Problem 3

117.125 | **0** | **10000101** | **11010100100000000000000**

13.75 | **0** | **10000010** | **10111000000000000000000**

Shift by 6-3 = 3

Shift mantissa by difference in exponent

00110111000000000000000

Note: When shifting to the right, the first shift should put the implicit 1, then 0's

Sum Mantissa's

```
  1 1 0 1 0 1 0 0 1
+ 0 0 1 1 0 1 1 1 0
-------------------
1 0 0 0 0 1 0 1 1 1
```

Sum overflows, re-normalize by adding one to exponent and shifting mantissa by one

0 | **10000110** | **00000101110000000000000**

= 114.875

More precision and range

We have described IEEE-754 binary32 floating point format, commonly known as “single precision” (“float” in C/C++)

24 bits precision; equivalent to about 7 decimal digits

$3.4 * 10^{38}$ maximum value

Good enough for most but not all calculations

IEEE-754 also defines a larger binary64 format, “double precision” (“double” in C/C++)

53 bits precision, equivalent to about 16 decimal digits

$1.8 * 10^{308}$ maximum value

Most accurate physical values currently known only to about 47 bits precision, about 14 decimal digits

Next 2 Lectures

1. **Combinational Logic:**
    - **Basics of electronics; logic gates, muxes, decoders**
    - **ALU design**

2. State Machines
    - Sequential logic
    - Clocks and data storage
    - Building a simple processor

Levels of abstraction

- Quantum level, solid state physics
- Conductors, Insulators, Semiconductors.
- Doping silicon to make diodes and transistors.
- Building simple gates, boolean logic, and truth tables
- Combinational logic: muxes, decoders
- Clocks
- Sequential logic: latches, memory
- State machines
- Processor Control: Machine instructions
- Computer Architecture: Defining a set of instructions

Start with the materials: conductors and insulators

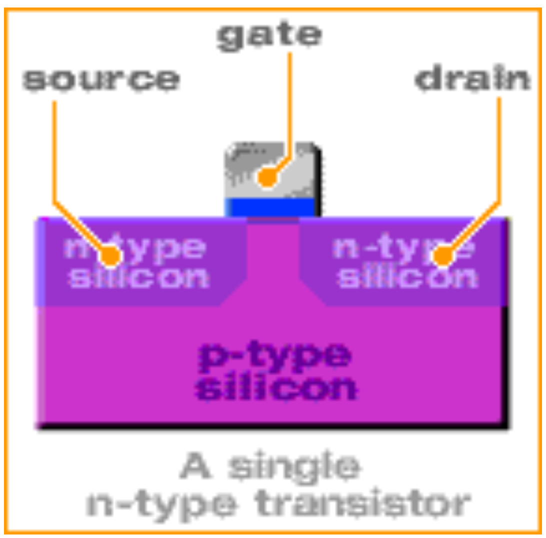
- Conductor: a material that permits electrical current to flow easily. (low resistance)
  - Lattice of atoms with free electrons

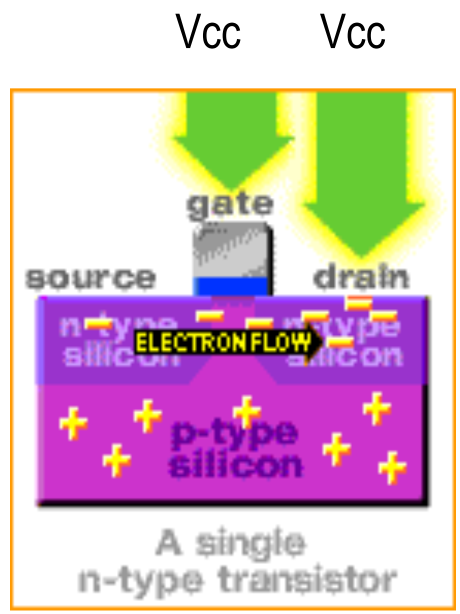
- Insulator: a material that is a poor conductor of electrical current (High resistance)
  - Lattice of atoms with strongly held electrons

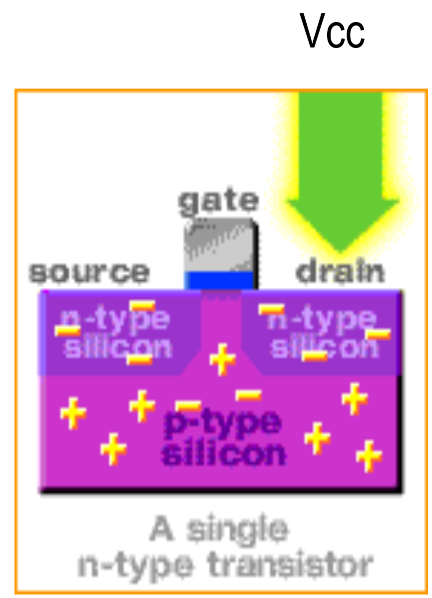
- Semi-conductor: a material that can act like a conductor or an insulator depending on conditions. (variable resistance)

Making a transistor

Transistors are electronic switches connecting the source to the drain if the gate is “on”.





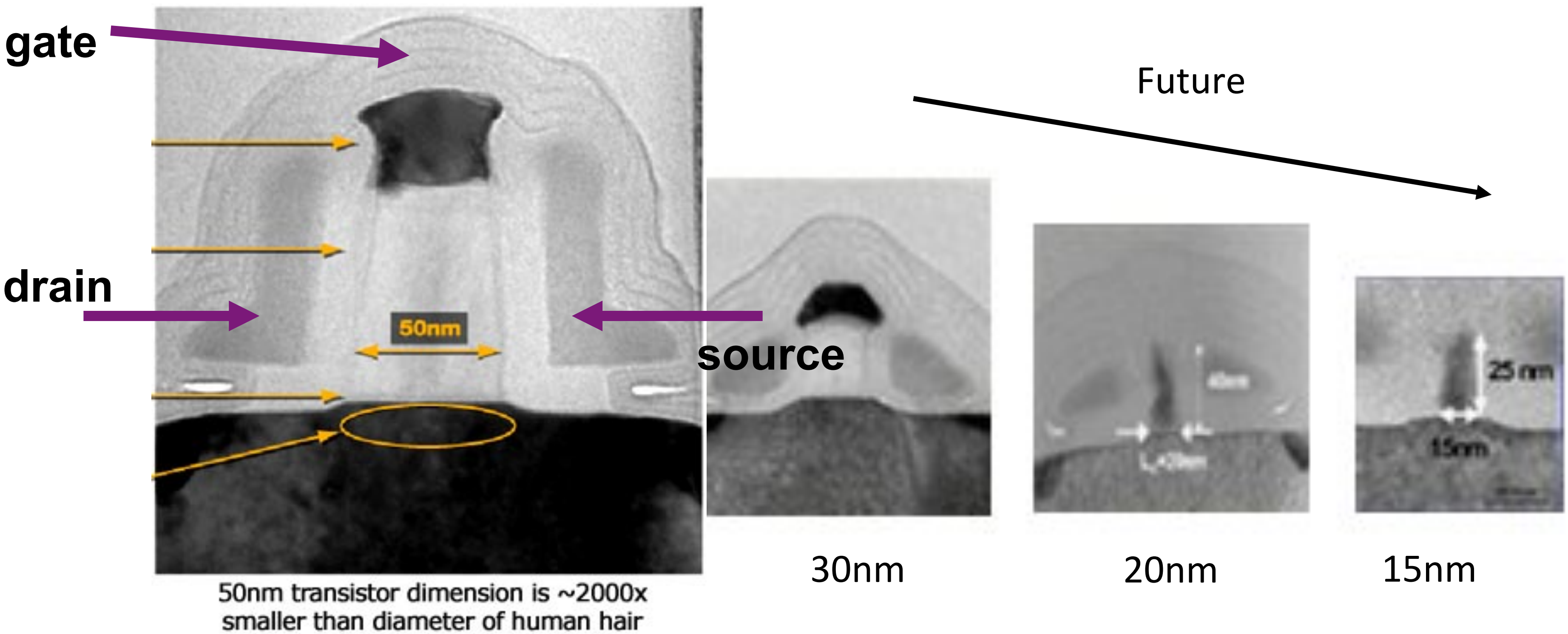


http://www.intel.com/education/transworks/INDEX.HTM

Recent pictures and the near future

Source: Intel

Today



90nm technology 2003

65nm technology 2005

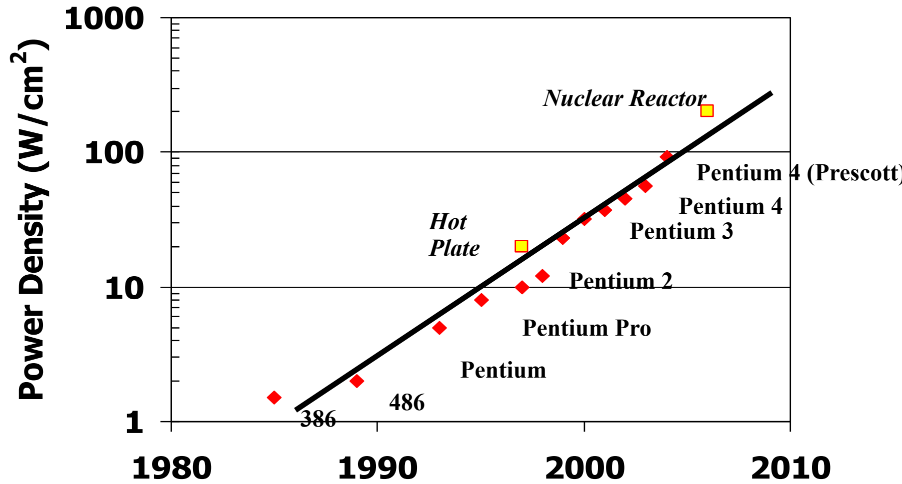
45nm technology 2008

32nm technology 2010

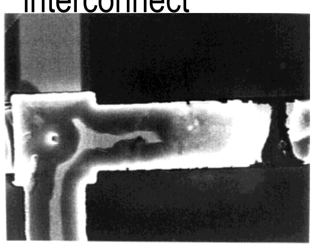
22nm technology 2012

The future carries a lot of problems…

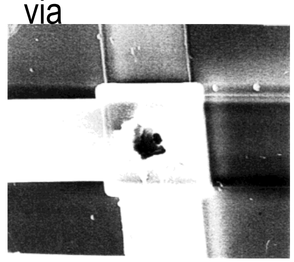
- *Area is not the biggest*
- Power density – Watts/mm^2
- Reliability (faults)

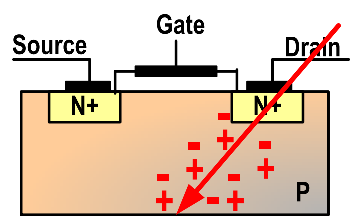


interconnect

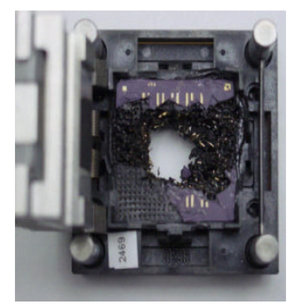


via





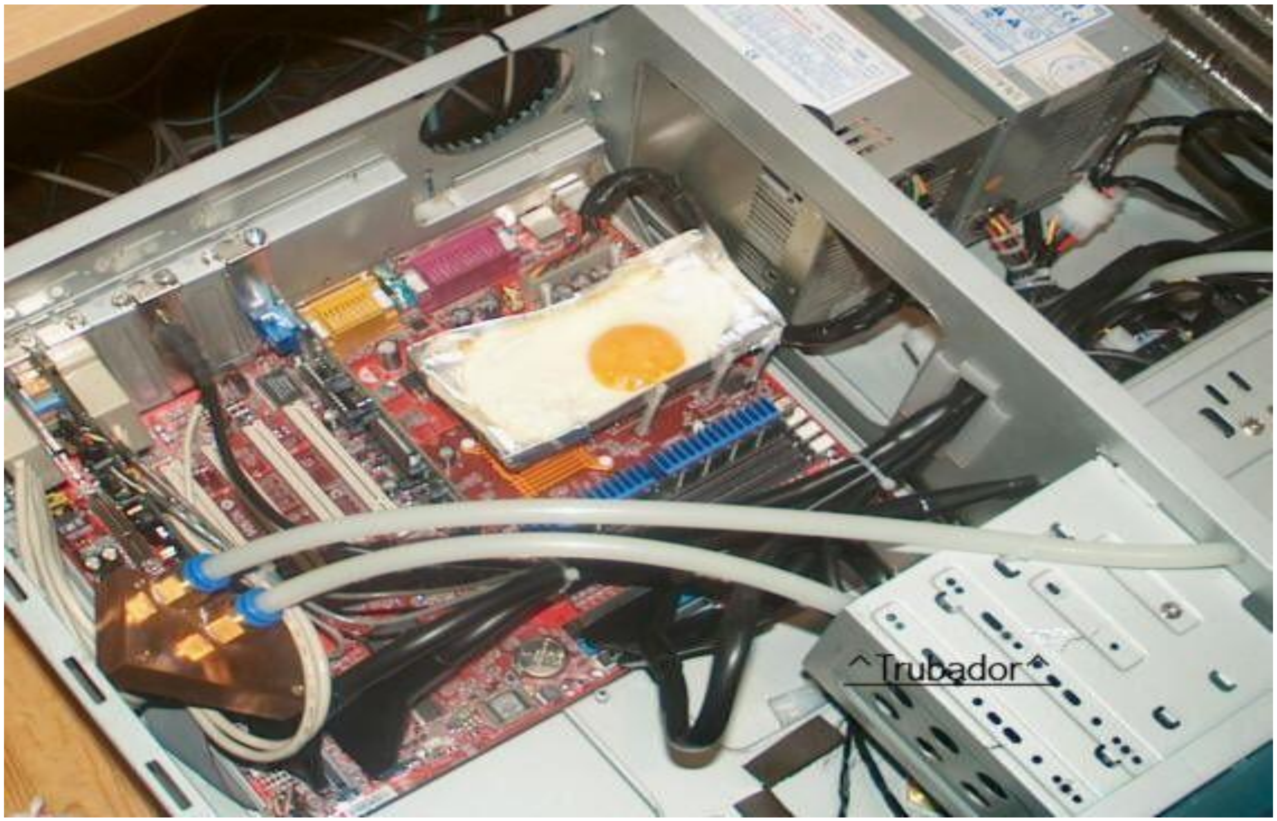
transients



Testing burn-in out

- Process variation (not all transistors are equal)
- …

As for power: Cooking-aware computing



Source: The New York Times, 25 June 2002

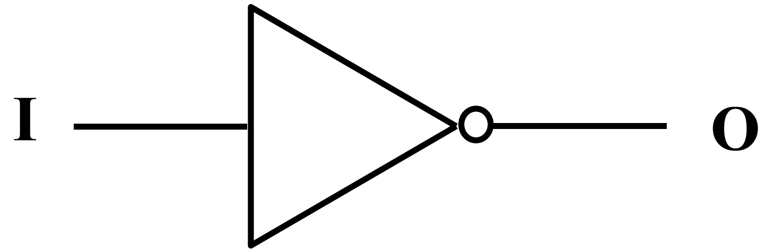
Basic gate: inverter

CS abstraction
- logic function

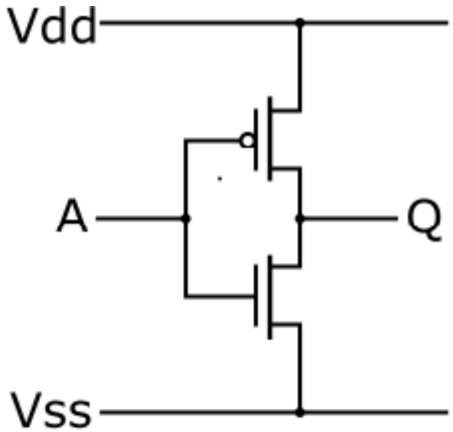
Truth Table

I	O
0	1
1	0

Schematic symbol (CS/EE)



Transistor-level schematic

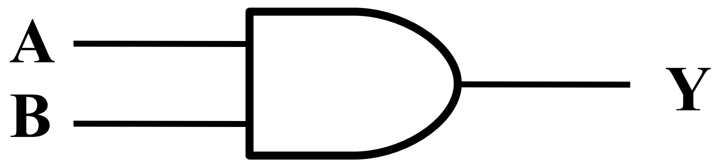


Basic gates: AND and OR

AND

Truth Table

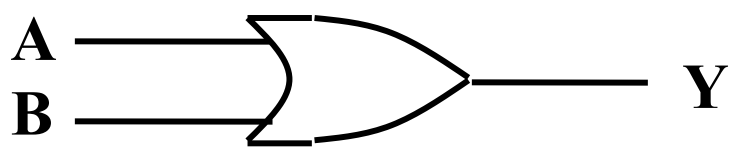
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1



OR

Truth Table

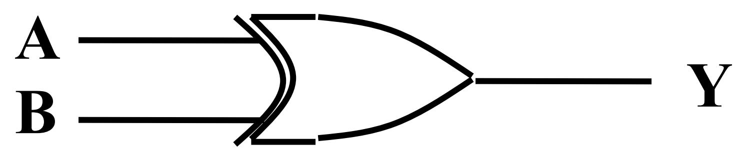
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



Basic gate: XOR (eXclusive OR)

Truth Table

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

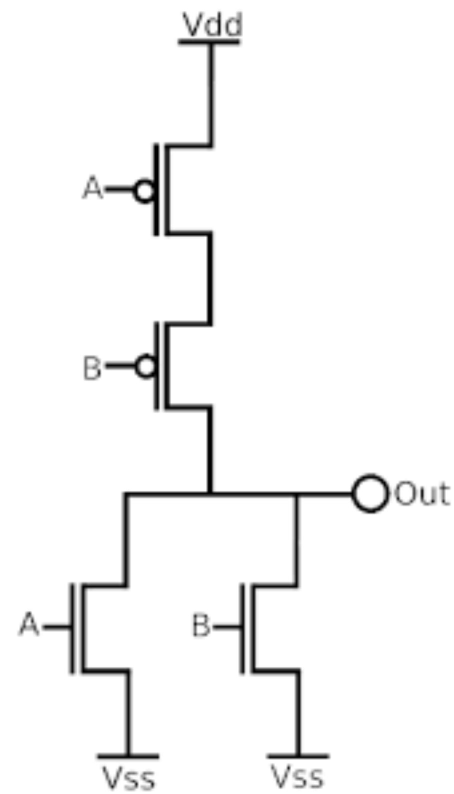


Basic gate: NOR (like the LC-2K NOR)

Truth Table

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Transistor-level schematic



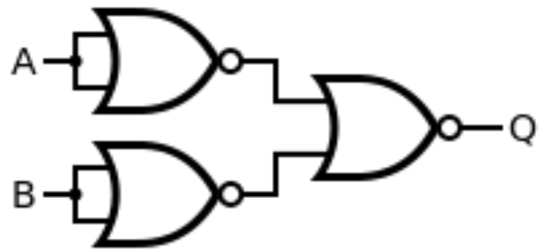
Exercise

- NOR can be used to implement *all* other logic functions!
- Exercise:
- Implement INV using only NOR gates
- Implement AND using only NOR gates
- Implement OR using only NOR gates
- Implement XOR using only NOR gates

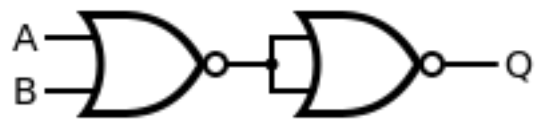
Exercise

- INV

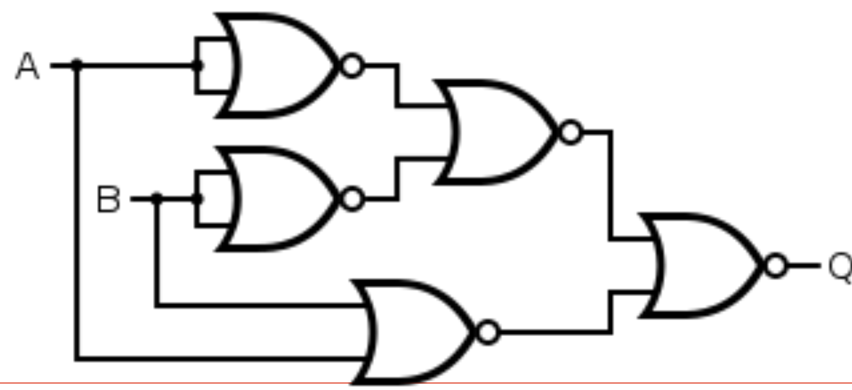
- AND



- OR



- XOR

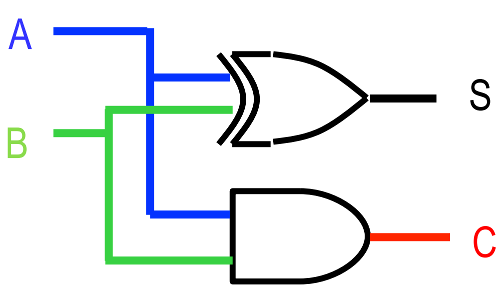


Combinational Circuits – implement Boolean expressions

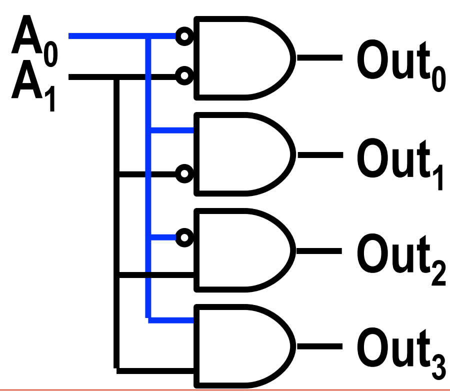
- ❑ Output is determined exclusively by the input
- ❑ **No memory**: Output is valid only as long as input is

  - Adder is the basic gate of the ALU
  - Decoder is the basic gate of indexing
  - MUX is the basic gate controlling data movement

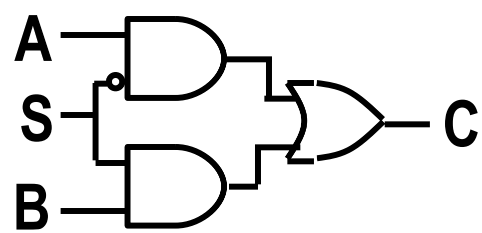
Half Adder



Decoder



MUX



Half adder

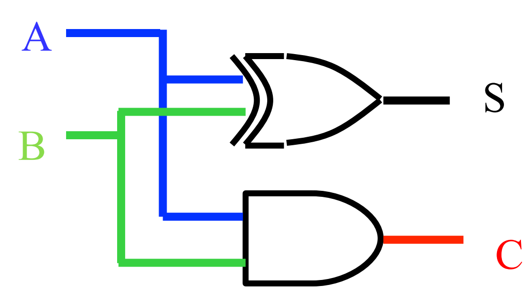
- Carry bit (C) can use an AND gate
- Sum bit (S) can use an XOR gate

Truth Table

Add 2 1-bit numbers

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

Circuit



Decoder

A_1	A_0	Out_3
0	0	0
0	1	0
1	0	0
1	1	1

Out_3 is just an AND gate

A_1	A_0	Out_2
0	0	0
0	1	0
1	0	1
1	1	0

Out_2 would be an AND gate if A_0 was inverted

A_1	A_0	Out_1
0	0	0
0	1	1
1	0	0
1	1	0

Invert A_1

A_1	A_0	Out_0
0	0	1
0	1	0
1	0	0
1	1	0

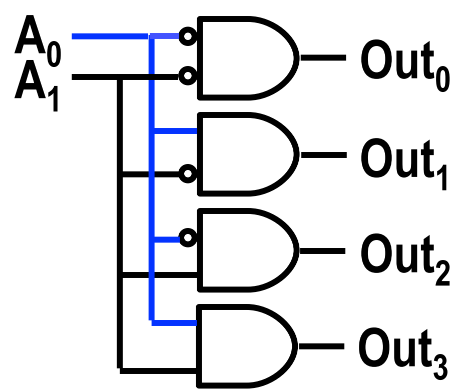
Invert A_1 and A_2

Truth Table

Select a single line
given an index

A_1	A_0	Out $_{3-0}$
0	0	0001
0	1	0010
1	0	0100
1	1	1000

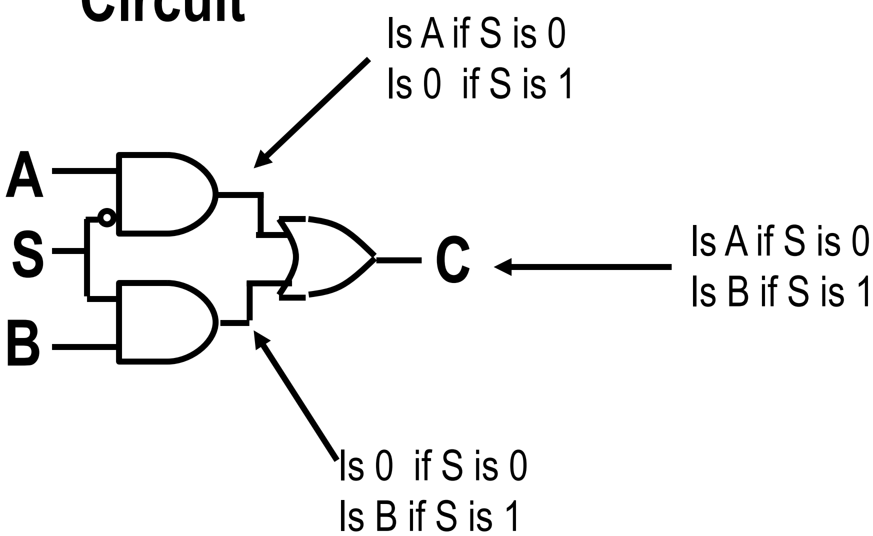
Circuit



Multiplexor (MUX)

- Input S selects either input A or input B

Circuit



Truth Table

Select one of multiple input lines to pass to the output

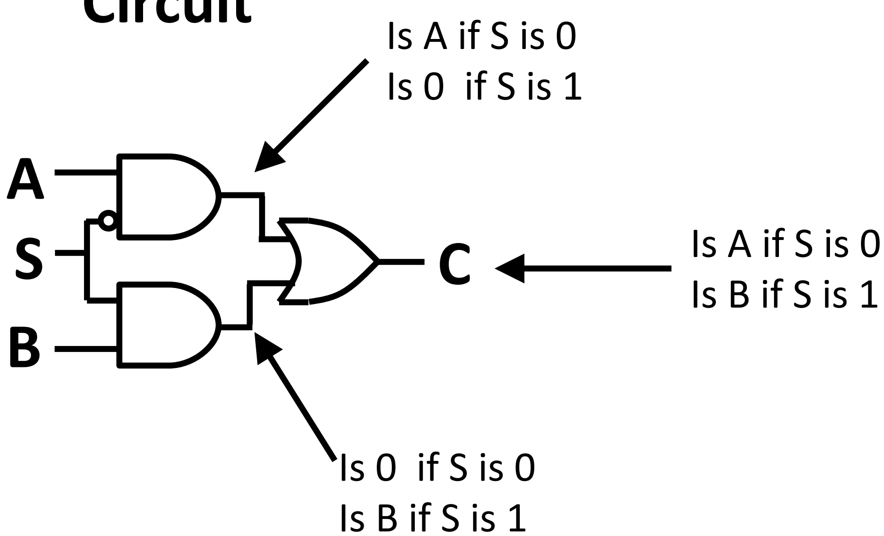
A	B	S	C
a	b	0	a
a	b	1	b

This is called a 2x1 MUX since it has 2 inputs and 1 output.
How would you build a 4x1 MUX?

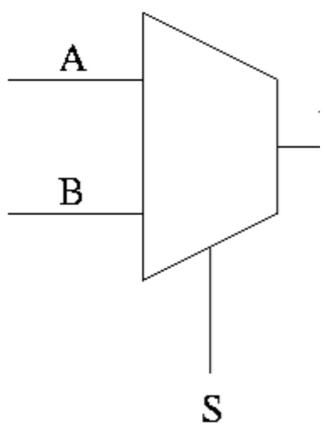
Exercise

- Build (draw) a 4x1 mux
- Hint: use 2x1 muxes and 2 S lines

Circuit

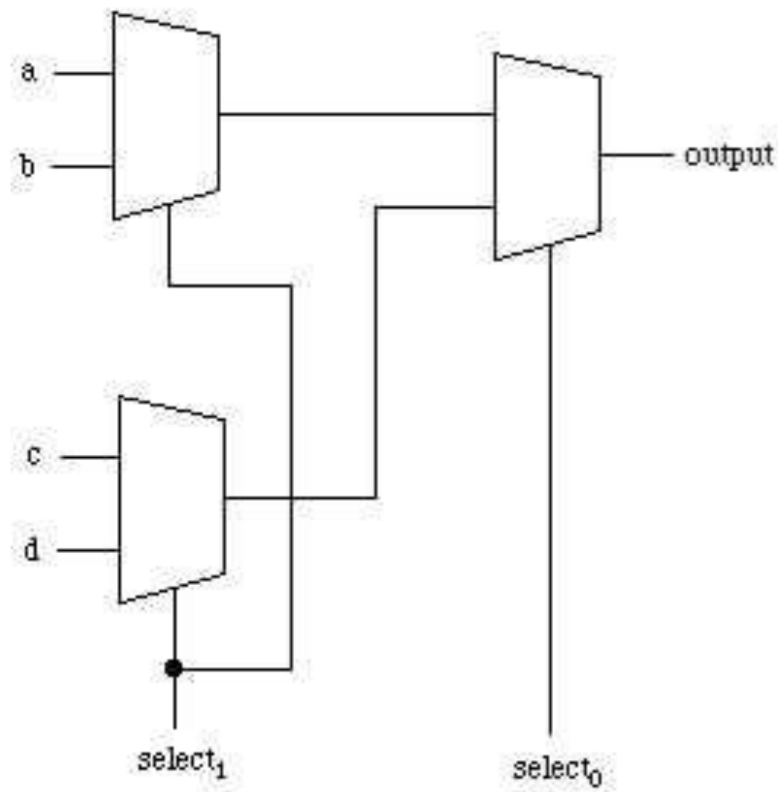


Symbol

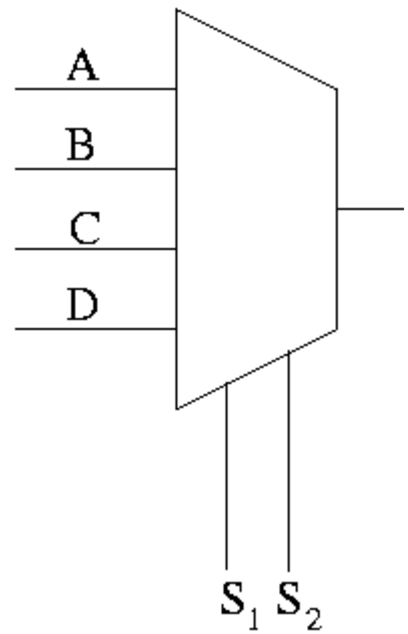


Exercise

- 4x1 mux made from 2x1 muxes



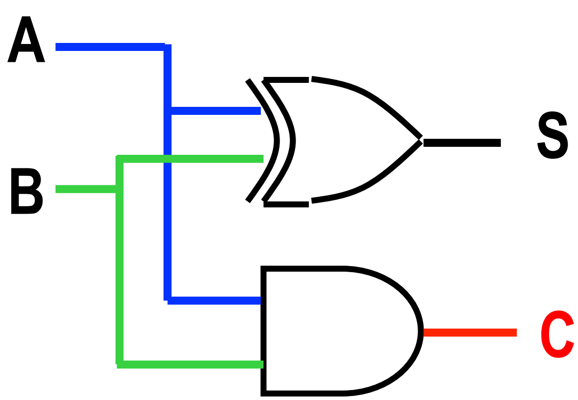
Symbol



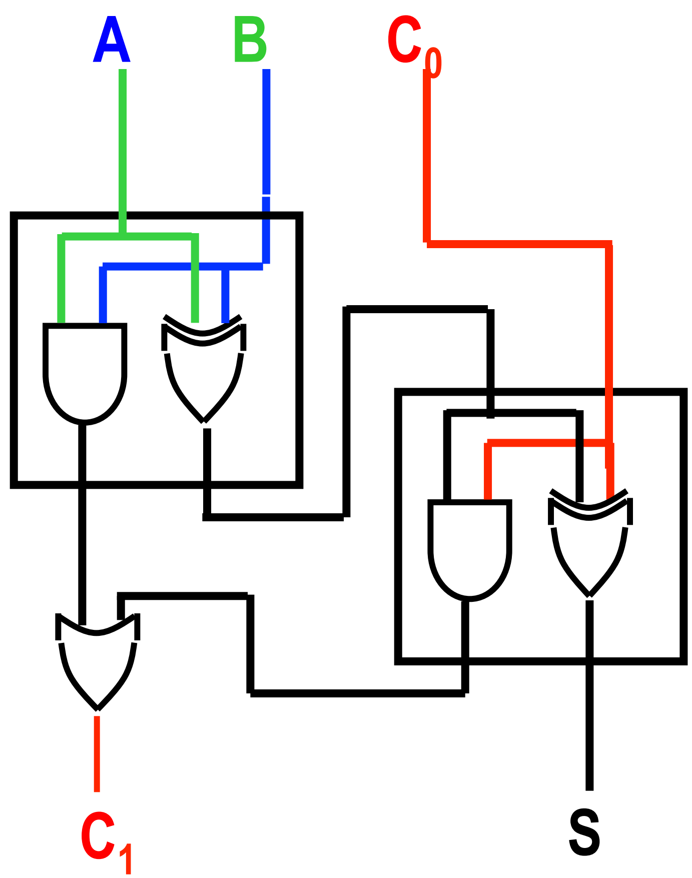
Building combinational circuits: half and full adder

Half adder

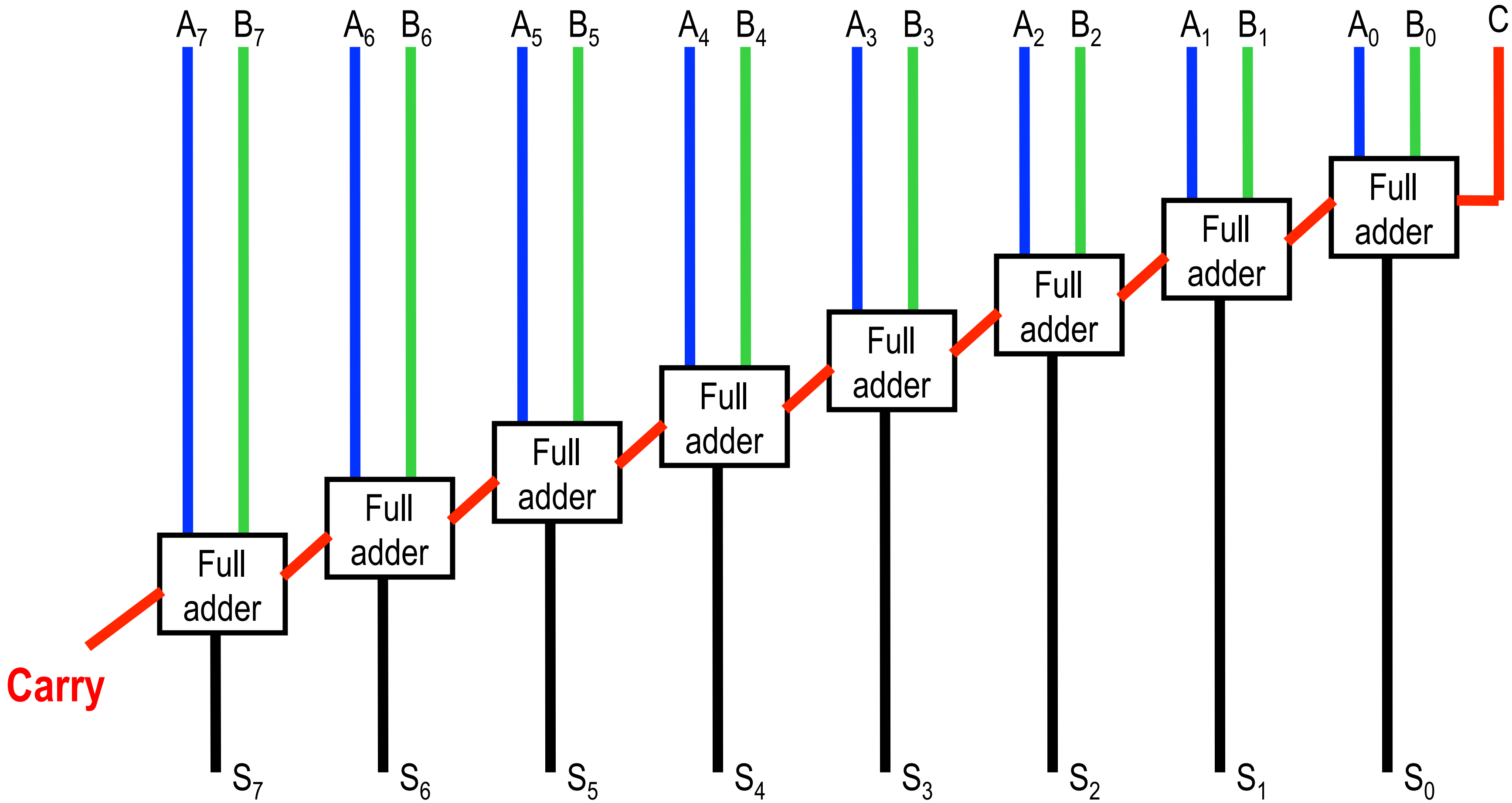
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Full adder



8-bit ripple carry adder



Unfortunately this has a very large propagation time for 32 or 64 bit adds

Problem with ripple carry adder

- The critical path is two gate delays per stage.
- Consider adding two 32-bit numbers.
- 64 gate delays.
- Too slow!
- Consider faster alternatives.
- To do this, we will use the concepts of generation and propagation.

- Generate: cout = 1 regardless of cin.
- Propagate: cout = 1 only if cin = 1.

Single bit carry propagate and generate

sum $= a \oplus b \oplus cin$

$= p \oplus cin,$

given that $p = a \oplus b$.

cout $= a\,b + a\,cin + b\,cin$

$= a\,b + cin\,(a + b)$

$= g + cin\,(a + b)$

$= g + cin\,p,$

given that $g = a\,b$.

Note that

$a \oplus b \neq a + b \rightarrow a\,b \rightarrow g.$

Generalized carry generation

$cout_i = g_i + p_i\ cout_{i-1}$.

Thus,

$cout_1 = g_1$

$cout_2 = g_2 + p_2\ cout_1 = g_2 + p_2\ g_1$

$cout_3 = g_3 + p_3\ cout_2$

$\quad = g_3 + p_3\ (g_2 + p_2\ g_1)$

$\quad = g_3 + p_3\ g_2 + p_3\ p_2\ g_1$

Within the flattened group, there is no carry chain!

Multiplier

$$
\begin{array}{r l}
01001101 & (77_{10}) \\
\times 1\,0\,1\,1\,0\,0\,1\,1 & (179_{10})
\end{array}
$$

```
                  0 1 0 0 1 1 0 1      × 1
                0 1 0 0 1 1 0 1 0      × 1

          0 1 0 0 1 1 0 1 0 0 0 0      × 1
        0 1 0 0 1 1 0 1 0 0 0 0 0      × 1

    0 1 0 0 1 1 0 1 0 0 0 0 0 0 0      × 1
    ---------------------------------------
    0 1 1 0 1 0 1 1 1 0 1 0 1 1 1          (13,783₁₀)
```

Faster multiplication

Traditional multiply

Generate partial products one at a time

Add them up as you go

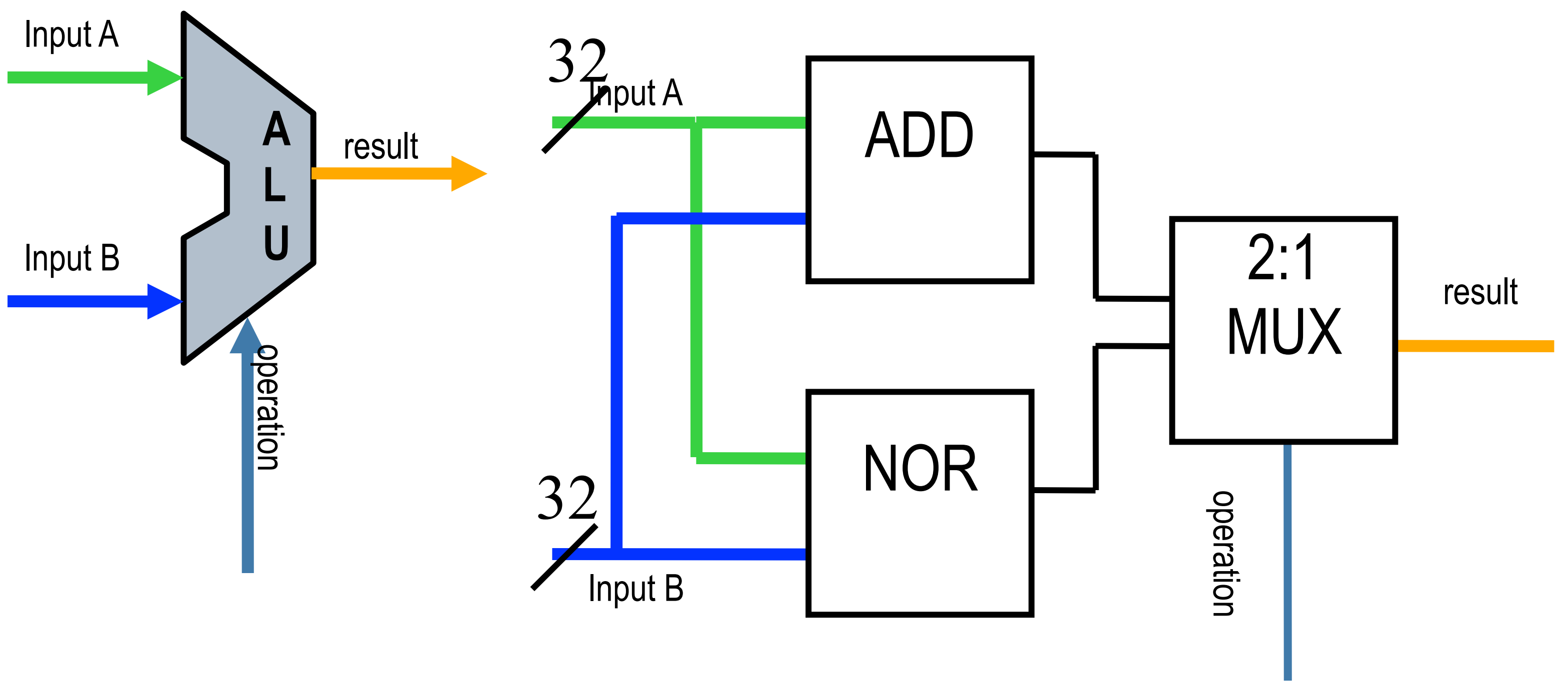
Faster way:

Generate partial products in parallel

Add them up as fast as you can

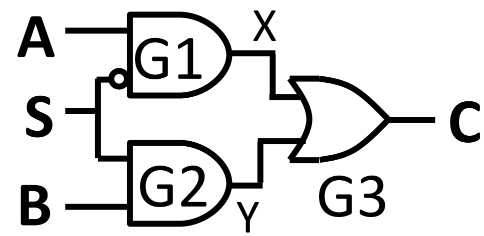
Tree structure could do it in logarithmic time rather than linear time

Building combinational circuits: LC-2K ALU example

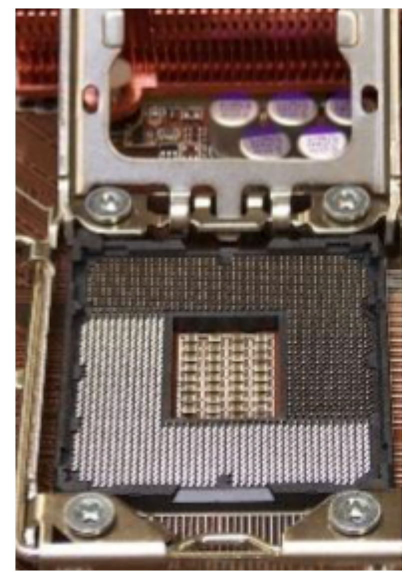


Verifying a Circuit

- How many possible inputs are there for a 2-1 mux?



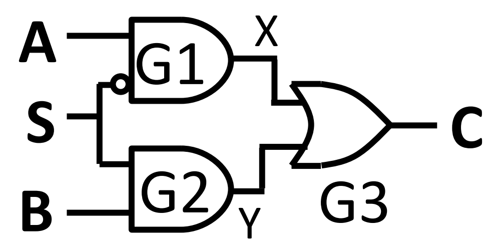
- How many possible inputs are there for a Core i7 with a 1,366 pins? For simplicity, assume all the pins are inputs.



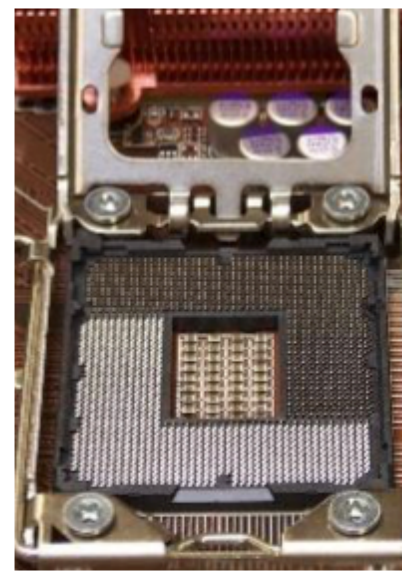
- How long would it take to try them all?

Verifying a Circuit

- How many possible inputs are there for a 2-1 mux?
  - A, B, or S could be 0 or 1, so 2^3 = 8
  - Easy to test all possible inputs



- How many possible inputs are there for a Core i7 with a 1,366 pins? For simplicity, assume all the pins are inputs.
  - 2^1366 = REALLY BIG NUMBER
  - Comparison: ~10^80 = 2^266 atoms in the universe



- How long would it take to try them all?
  - We don't have enough time!

Verifying a Circuit

- It's hard to verify combinational circuits
- *It gets worse when we add memory to the circuit*
- Next time: sequential circuits