8. Foundations of Processor Design: Combinational Logic

EECS 370 – Introduction to Computer Organization – Winter 2023

EECS Department
University of Michigan in Ann Arbor, USA

Upcoming stuff

- Project 1s and 1m
  - Due this Thursday
- HW2
  - Due Monday 2/6
  - Group part is non-trivial
- Project 2 is posted
  - 2a due Thursday 2/16
  - 2l due Thursday 2/23
  - 2c due Tuesday 3/10 (after break)
- HW3 out early next week, due Monday 2/20
- <u>Midterm: Thursday 3/9 7-9pm</u>

This week: Digital Logic

- Lectures 1-7:
  - LC2K and ARMv8/LEGv8 ISAs
  - Converting C to Assembly
  - Function Calls
  - Linking
- **Today:**
  - **Quick floating point review**
  - **Combinational Logic**
- Thursday:
  - Sequential Logic

Floating Point Representation

$10.625_{10} \Rightarrow 1010.101_{2}$

1.010101×2^{3}

This must be a 1!
So don't store it.

+/-	Exponent (3)	Significand (1010101)
1 bit	8 bits	23 bits

10.625_{10} = 0 10000010 01010100000000000000000

Class Problem

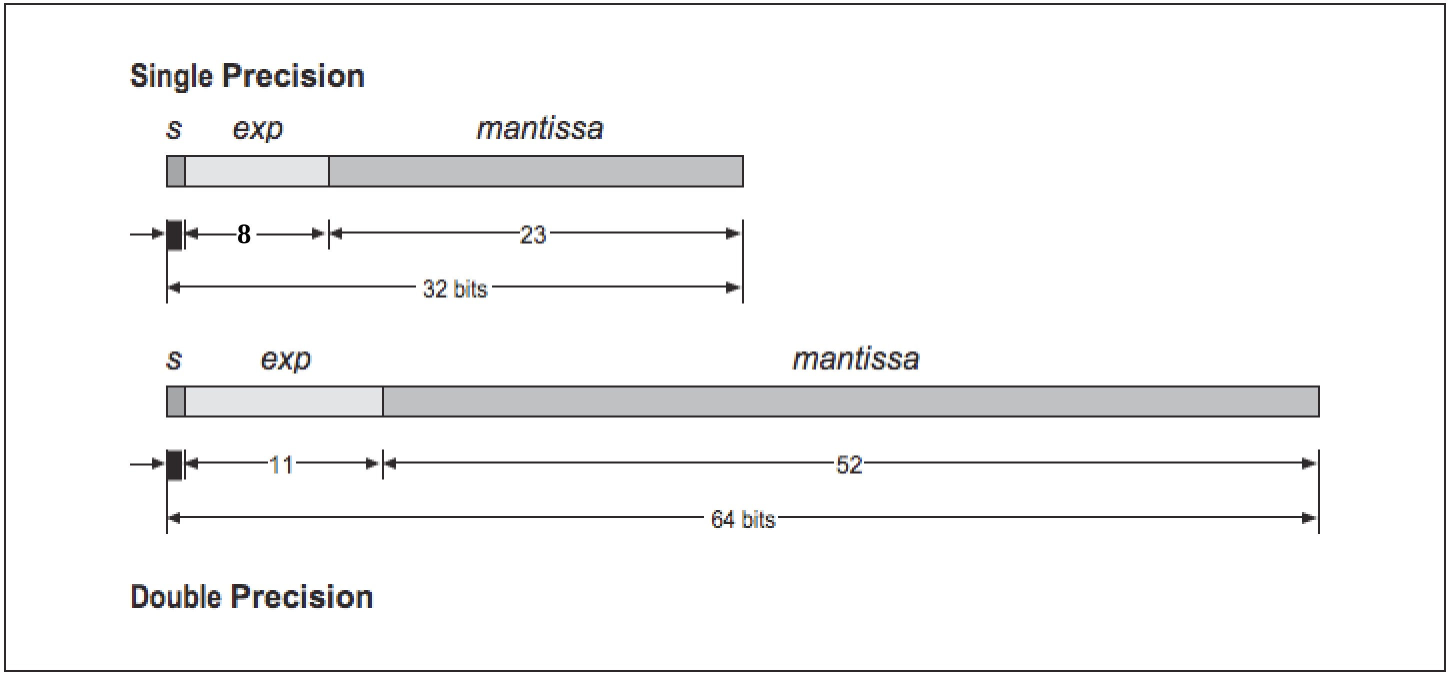
- What is the value (in decimal) of the following IEEE 754 floating point encoded number?

1	**10000101**	**01011001000000000000000**

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

Single (“float”) precision



Up Until Now…

- We've covered high-level C code to an executable
  - Compilation
  - Assembly
  - Linking
  - Loading
- Now, we'll talk about the hardware that runs this code
  - First step: the basics of digital logic

Next 3 Lectures

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

2. Sequential Logic
   - Clocks, latches, and flip-flops

3. State machines and the single-cycle computer

Levels of abstraction

- Quantum level, solid state physics
- Conductors, Insulators, Semiconductors.
- Doping silicon to make diodes and transistors.
- 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 to current flow)
  - Lattice of atoms with free electrons

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

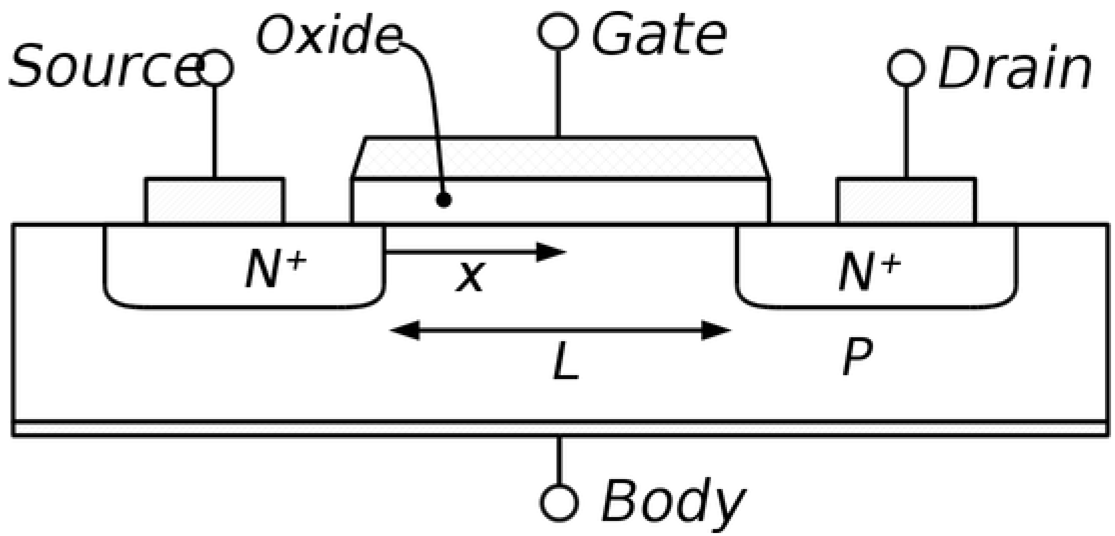
- Semi-conductor: a material that can act like a conductor or an insulator depending on conditions. (variable resistance to current flow)

Doped silicon semiconductors

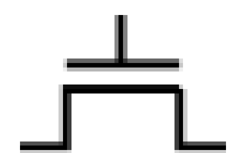
- How do we make tiny electronically controlled switches?
  - Semiconductors: control whether it conducts or not
- Basic semiconductor material of most electronics
- Start with pure crystalline silicon (4 valence electrons)
- Deliberately add a **small** amount of an impurity with a different number of valence electrons
  - Light doping: 1 in 100,000,000 atoms
  - Heavy doping: 1 in 10,000 atoms
- This small amount of impurity can drastically change the electrical properties of the silicon!

Making a transistor

Our first level of abstraction is the transistor (basically 2 diodes sitting back-to-back)

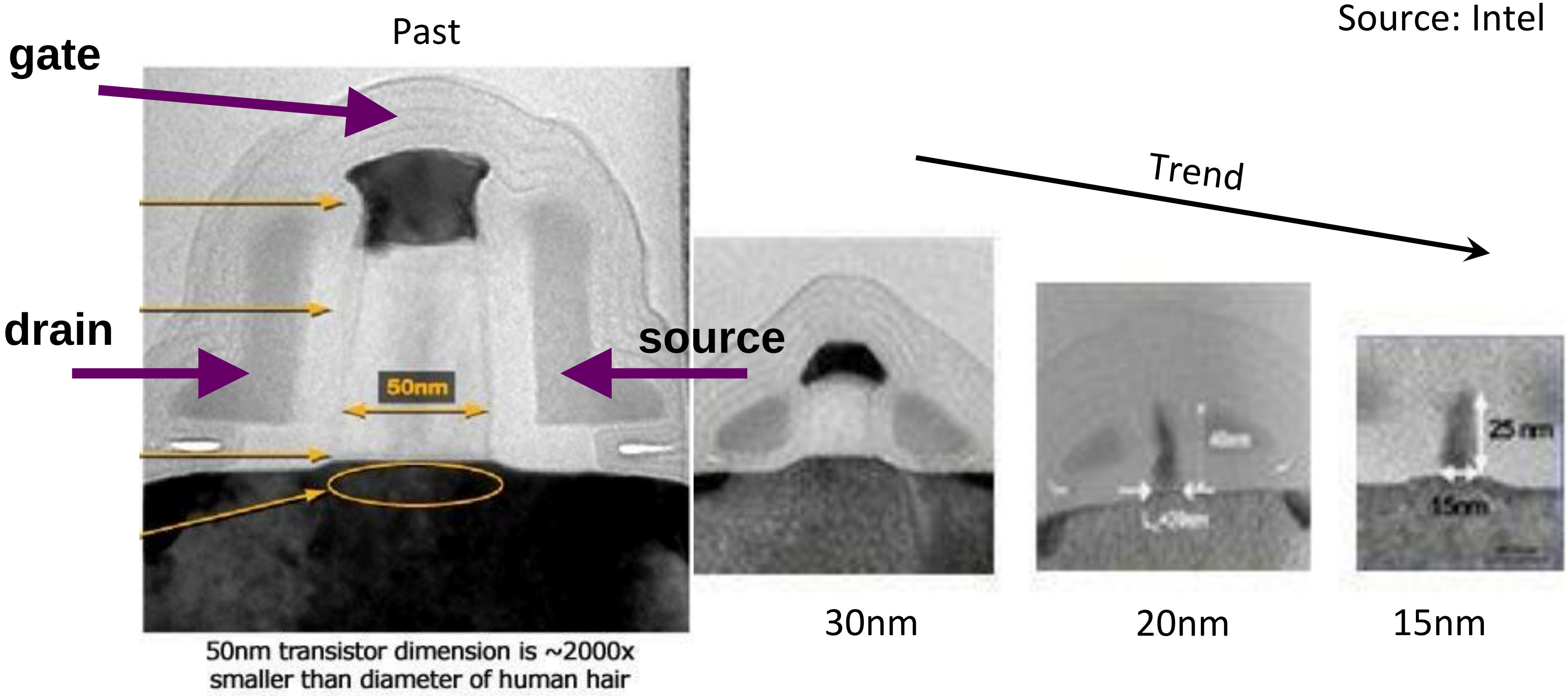


Electrical engineers use a diagram like this:



Recent pictures

Source: Intel



90nm technology
2003

65nm technology
2005

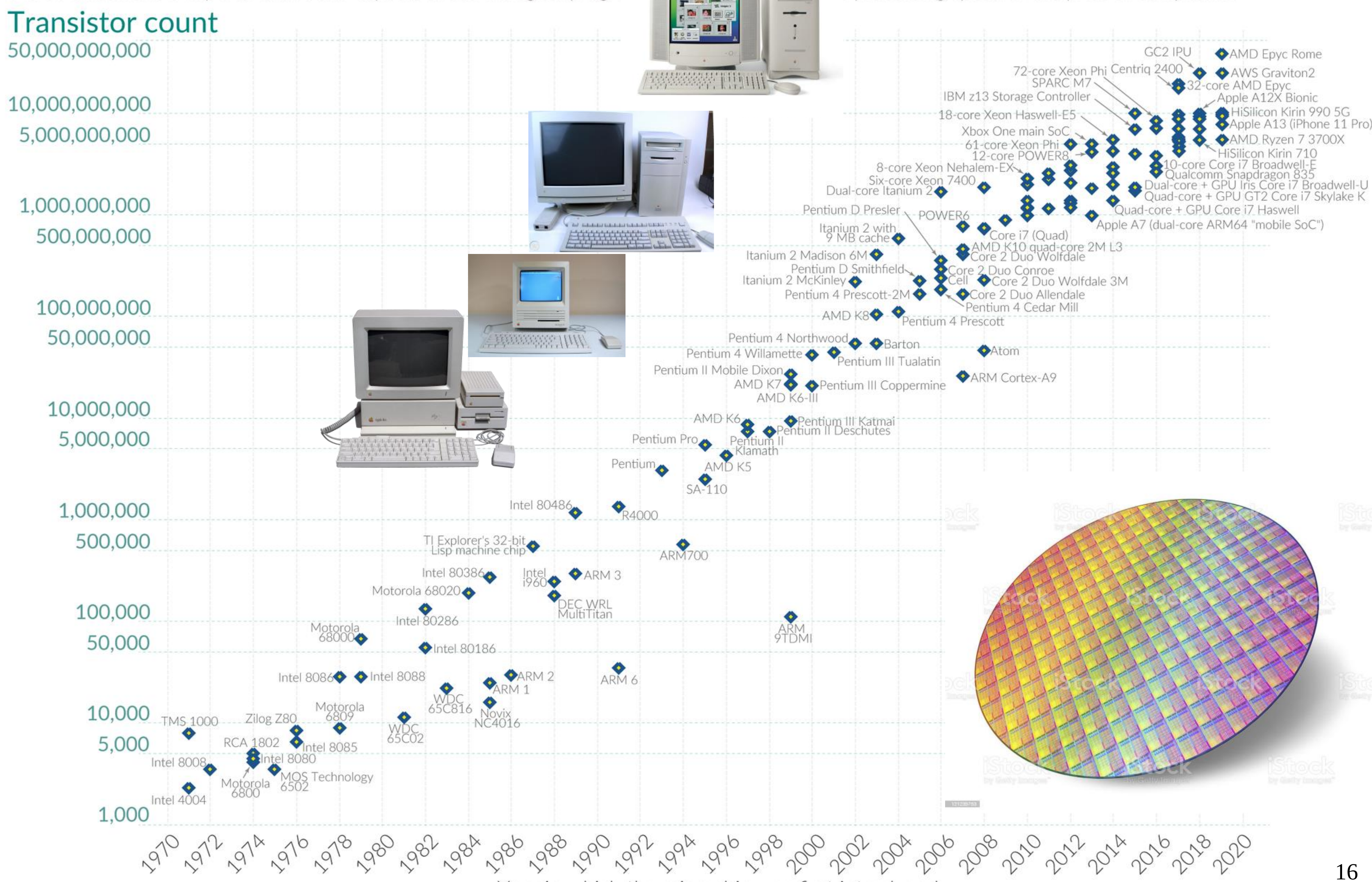
45nm technology
2008

32nm technology
2010

Moore's Law: The number of transistors on microchips doubles every two years

Our World in Data

Moore's law describes the empirical regularity that the number of transi s doubles approximately every two years.
This advancement is important for other aspects of technological progre ; processing speed or the price of computers.



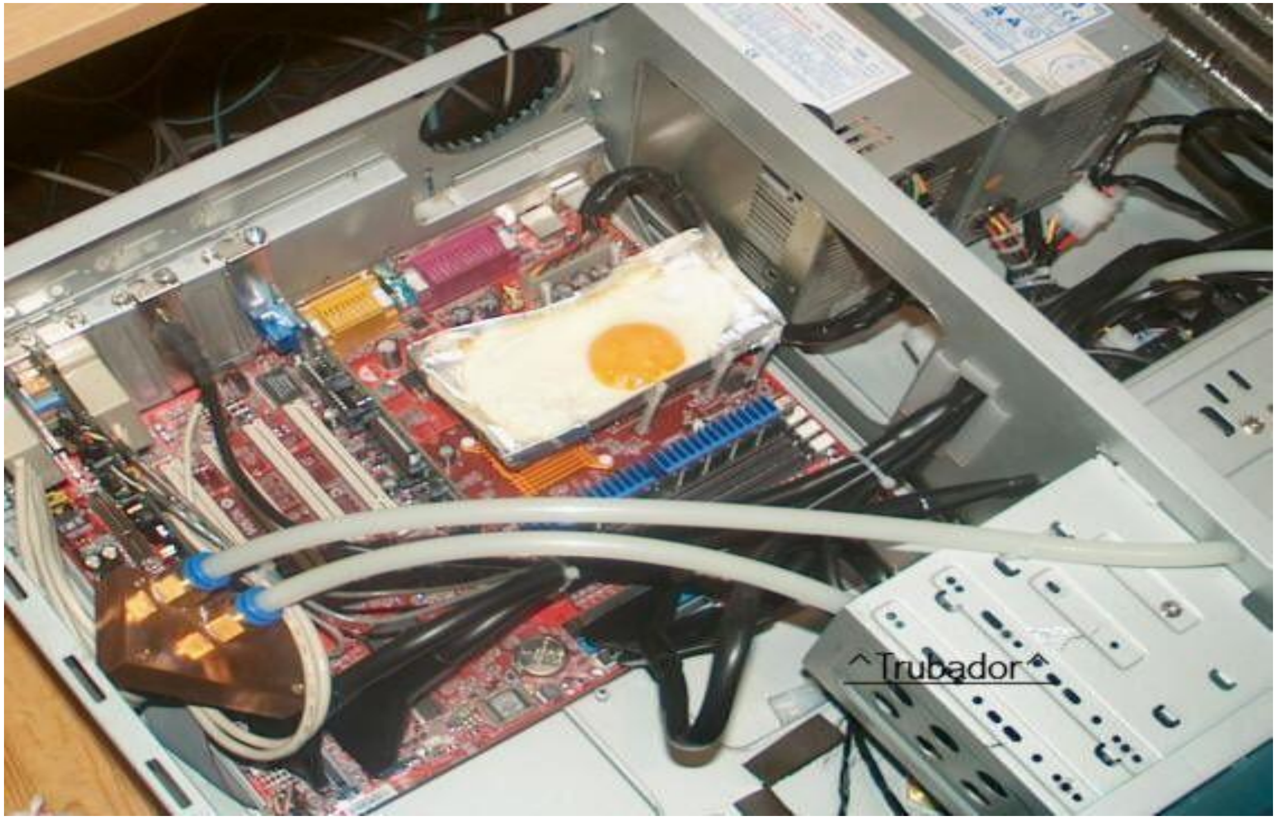
Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world's largest problems.

We have more “switches”, but they are hard to use.

- Basic theme:
  - As transistors get smaller, the power used by each transistor drops
    - But not be enough to keep the power per unit area constant.
  - So power density (power per unit area) is going up.
  - Power turns into heat
  - So the chips are getting very hot
  - So we either have to run the transistors slower (so they use less power) or only use some of them.

- In the past the power density was nearly constant (Denard scaling)
  - So Moore’s Law gave us more we could use “for free”

As for power: Cooking-aware Computing



Source: The New York Times, 25 June 2002

Let’s dig into how we use those transistors

Basic gate: Inverter

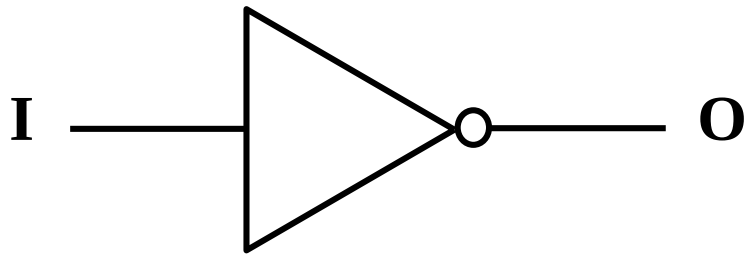
CS abstraction

- logic function

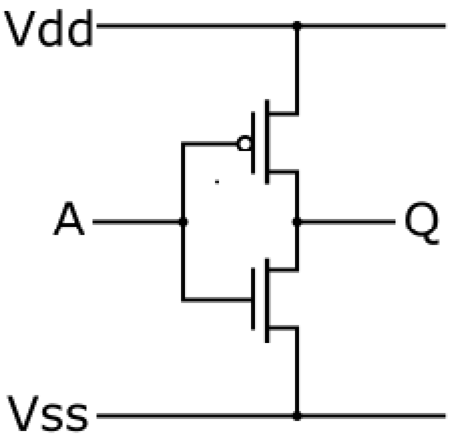
Truth Table

I	O
0	**1**
1	**0**

Schematic symbol (CS/EE)



Transistor-level schematic

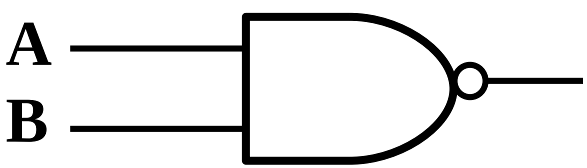


We won't focus on this part in 370. Take 270 and 312 to learn more

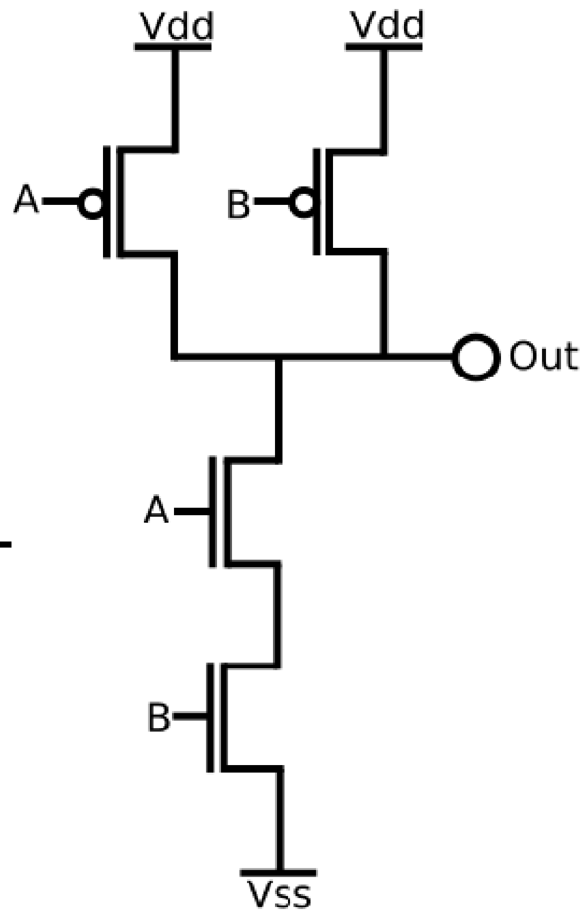
Basic gate: NAND

Truth Table

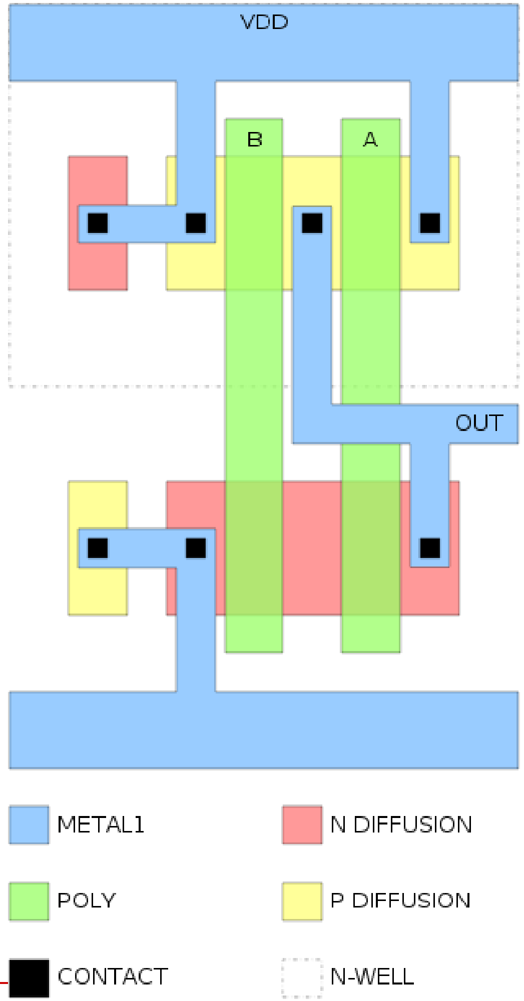
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



Transistor-level schematic



Layout schematic

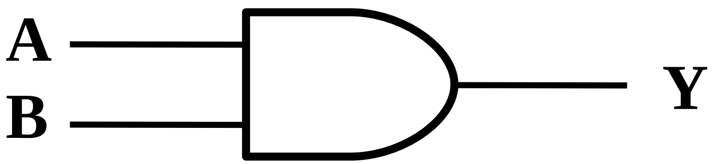


Basic gates: AND and OR

AND

Truth Table

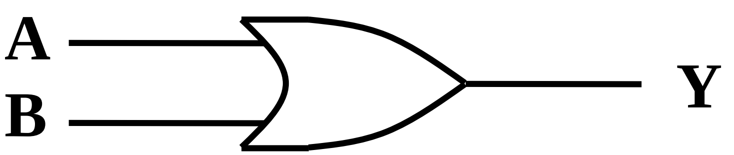
A	B	Y
0	**0**	**0**
0	**1**	**0**
1	**0**	**0**
1	**1**	**1**



OR

Truth Table

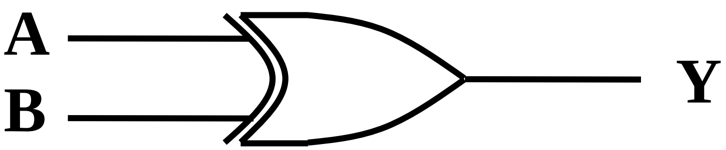
A	B	Y
0	**0**	**0**
0	**1**	**1**
1	**0**	**1**
1	**1**	**1**



Basic gate: XOR (eXclusive OR)

Truth Table

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



Exercise

- NAND is logically complete
  - This means that all gates can be implemented using only NANDs
  - NOR is also logically complete

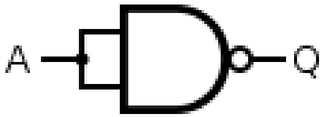
- Exercise:
  - Implement INV using only NAND gates
  - Implement AND using only NAND gates
  - Implement OR using only NAND gates
    - Hint Demorgan's Law: a || b = !(!a && !b)

Exercise: Implement each using only NAND gates

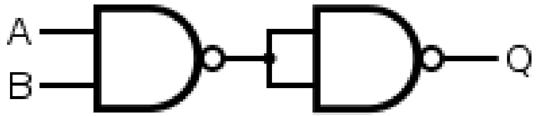
- INV
- AND
- OR

Exercise

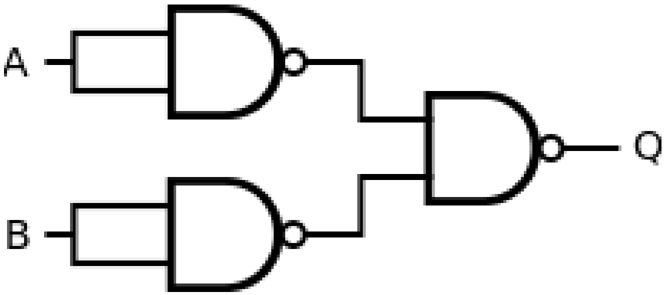
- INV



- AND



- OR

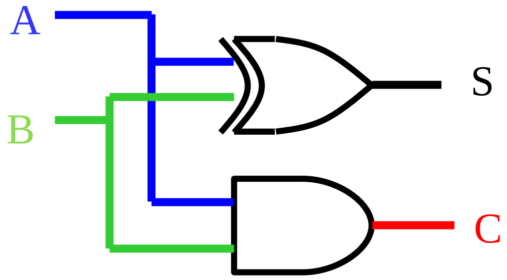


Building Complexity: Addition

- We want to design a circuit that performs binary addition
- Let's start by adding two bits
  - Design a circuit that takes two bits (A and B) as input
    - Generates a sum and carry bit (S and C)
  1. Make a truth table
  2. Design a circuit

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

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

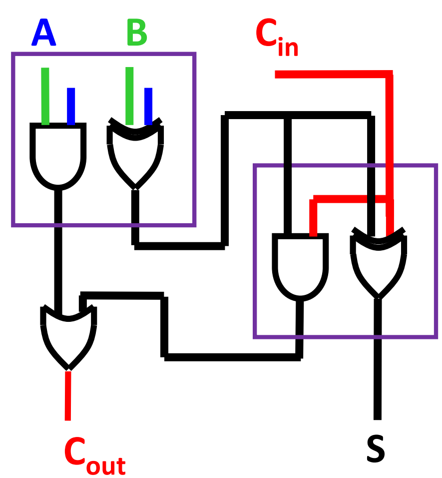


Building Complexity: Addition

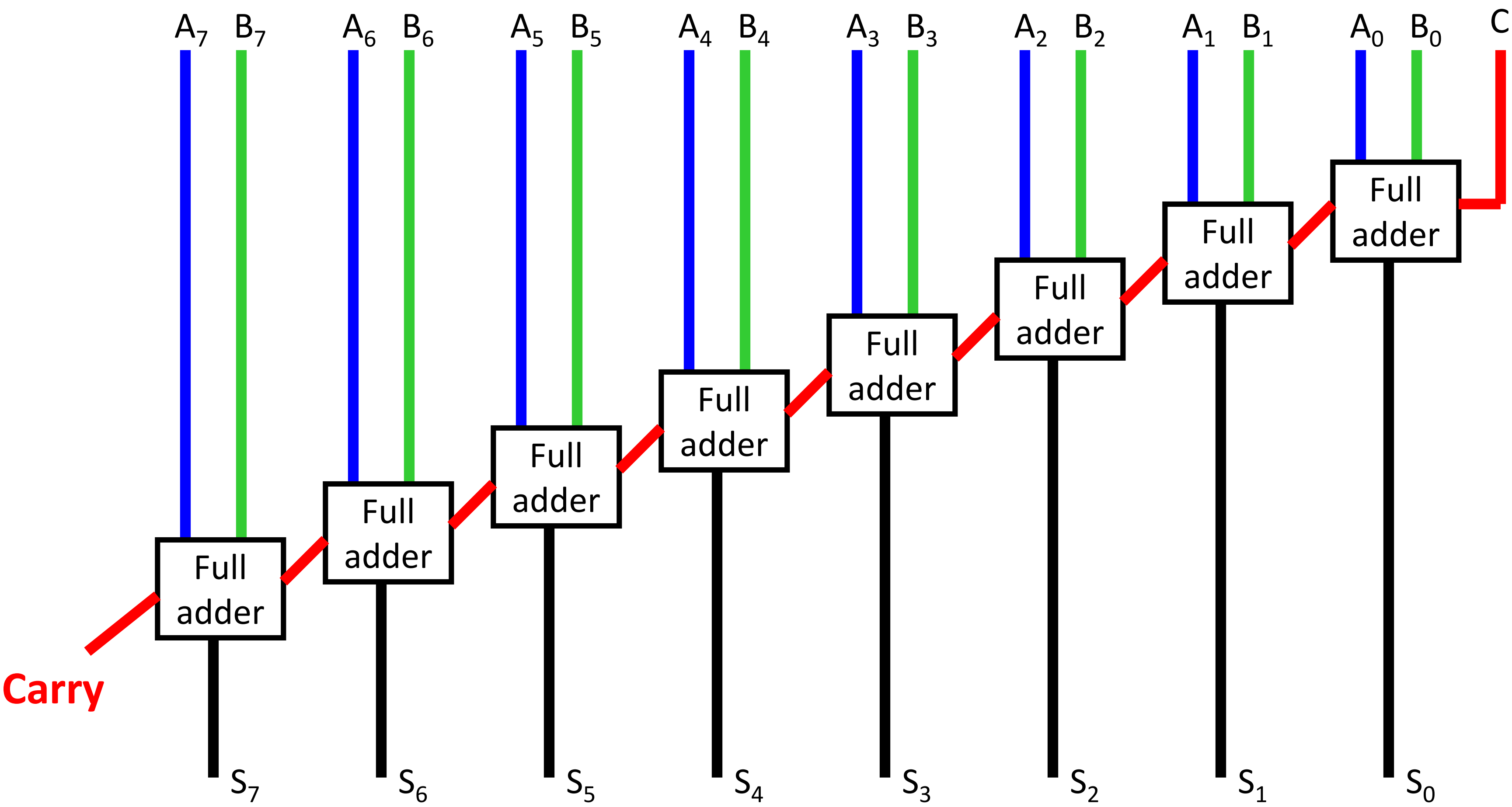
- Now we can add two bits, but how do we deal with carry bits?
  - We have to design a circuit that can add three bits
    - Inputs: A, B, Cin
    - Outputs: S, Cout
  1. Design a truth table
  2. Circuit
- How do we combine these?

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

Cin	A	B	Cout	S
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



8-bit Ripple Carry Adder



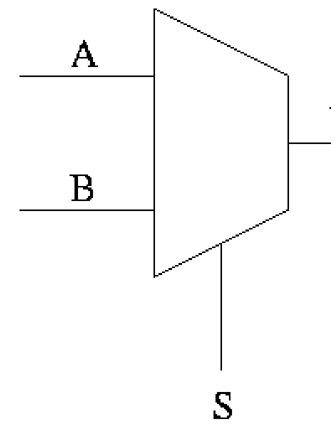
Unfortunately this has a very large propagation time for 32 or 64 bit adds

Building Complexity: Selecting

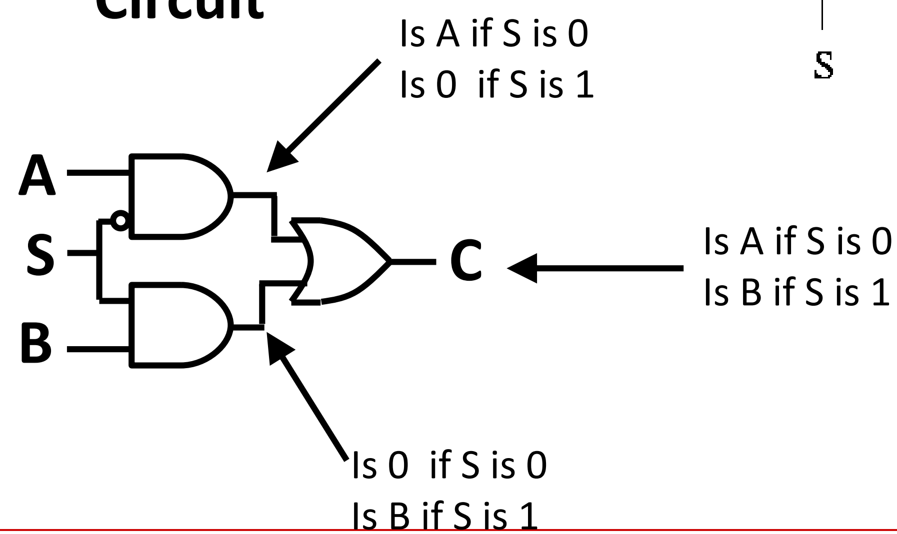
- We want to design a circuit that can select between two inputs
- Let's do a one bit version
  1. Draw a truth table

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

Symbol

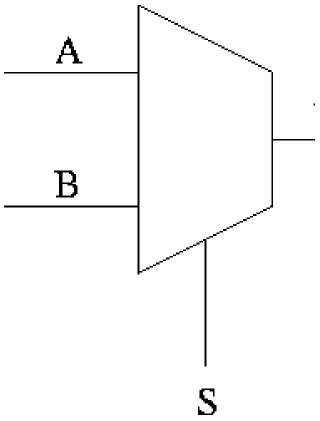


Circuit



Class Problem

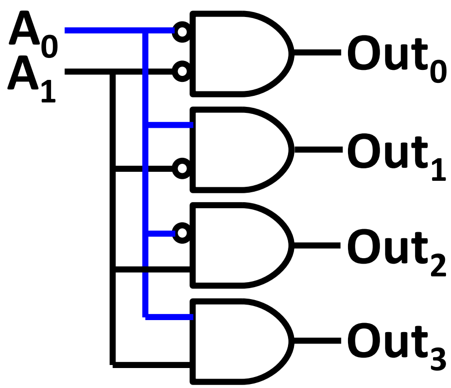
- Build (draw) a 4x1 mux
- Only use 2x1 muxes



Building Complexity: Decoding

- Another common device is a decoder
  - Input: N-bit binary number
  - Output: 2^N bits, exactly one of which will be high

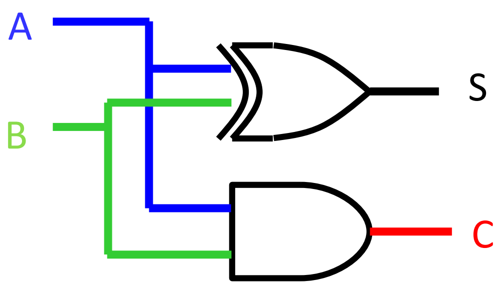
Decoder



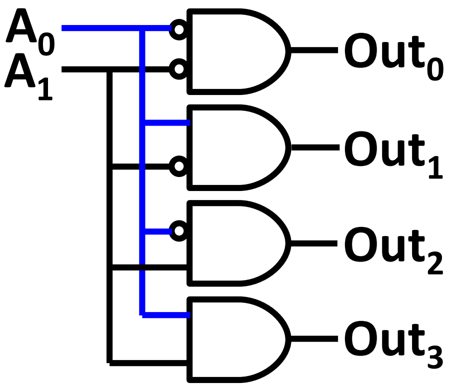
Combinational Circuits implement Boolean expressions

- 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 (Lecture 9)
  - Decoder is the basic gate of indexing
  - MUX is the basic gate controlling data movement

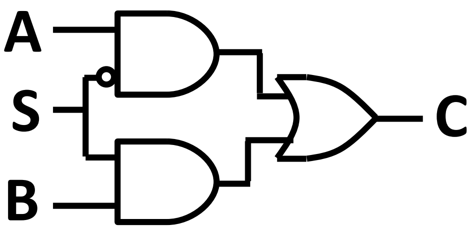
Half Adder



Decoder



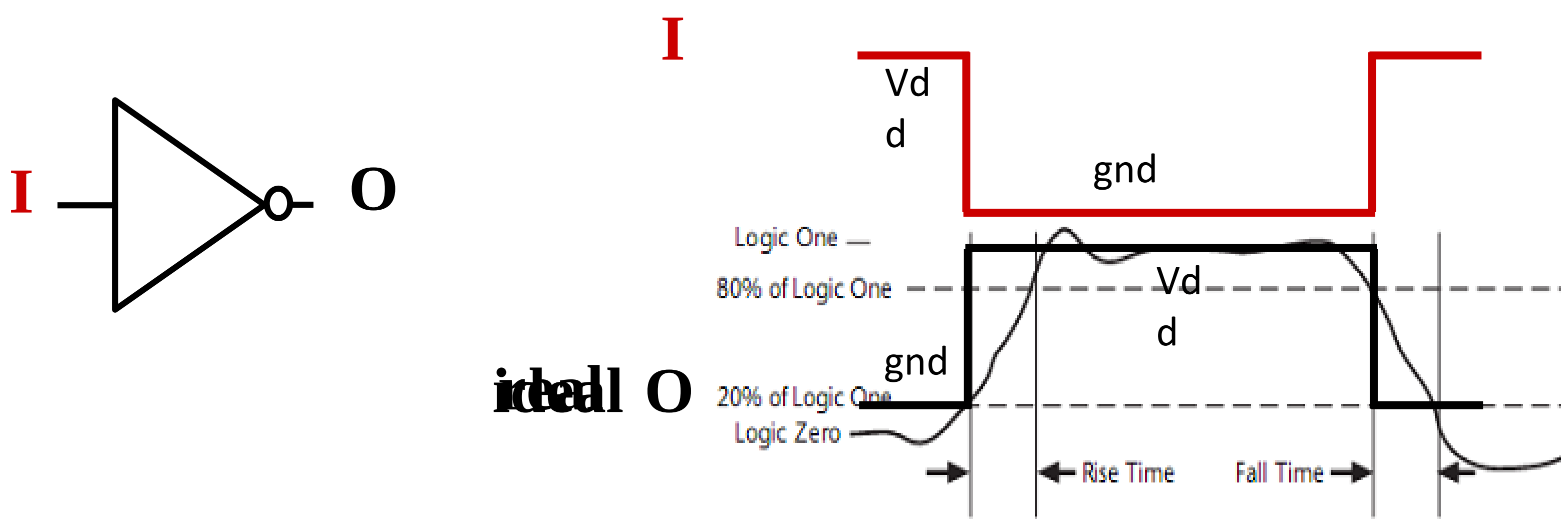
MUX



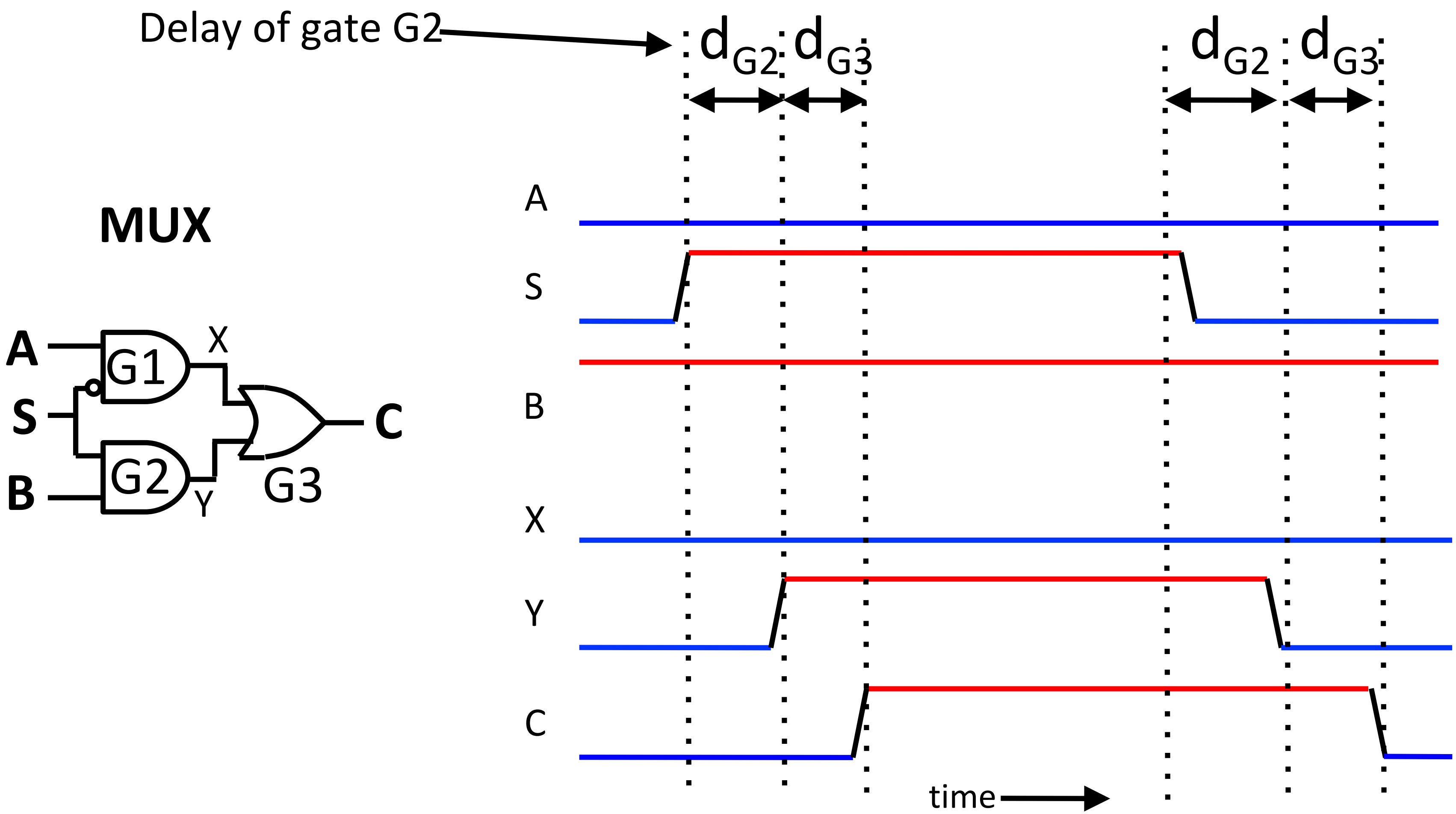
One more problem: Propagation delay in combinational gates

- Gate outputs do not change exactly when inputs do.
  - Transmission time over wires (~speed of light)
  - Saturation time to make transistor gate switch

⇒ Every combinatorial circuit has a propagation delay (time between input and output stabilization)



Timing in Combinational Circuits

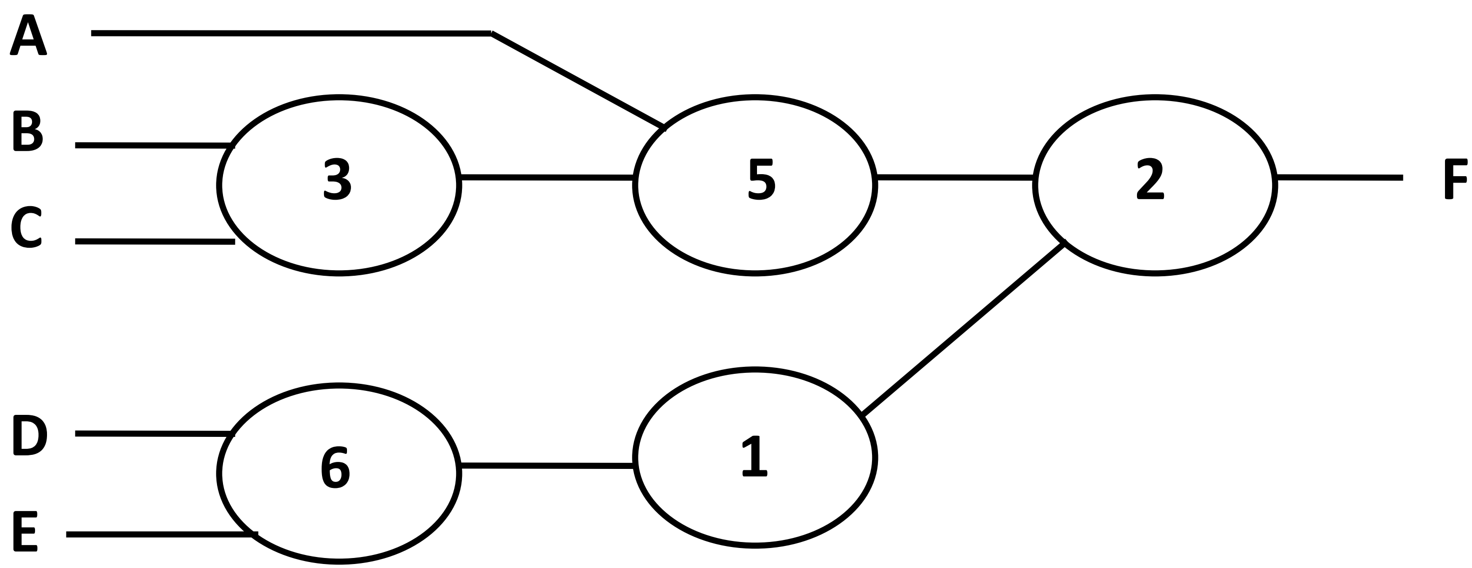


What is the input/output delay (or simply, delay) of the MUX?

Waveform viewers are part of designers' daily life

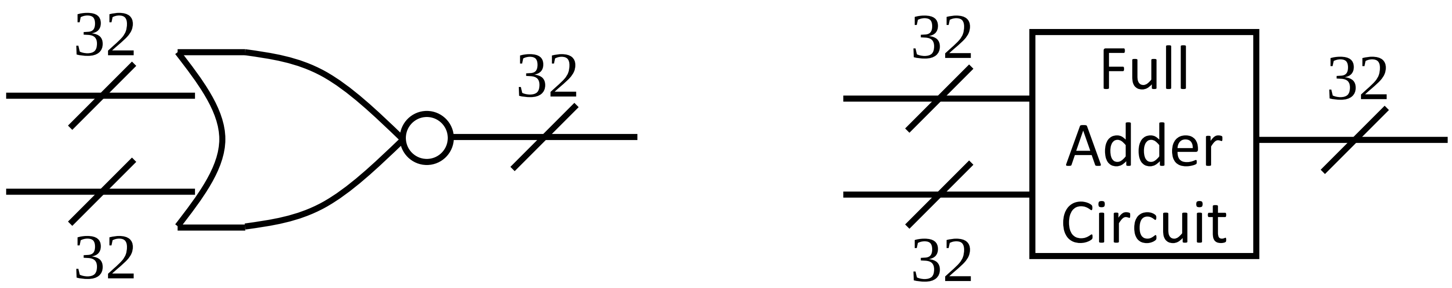
What is the delay of this Circuit?

Each oval represents one gate, the type does not matter



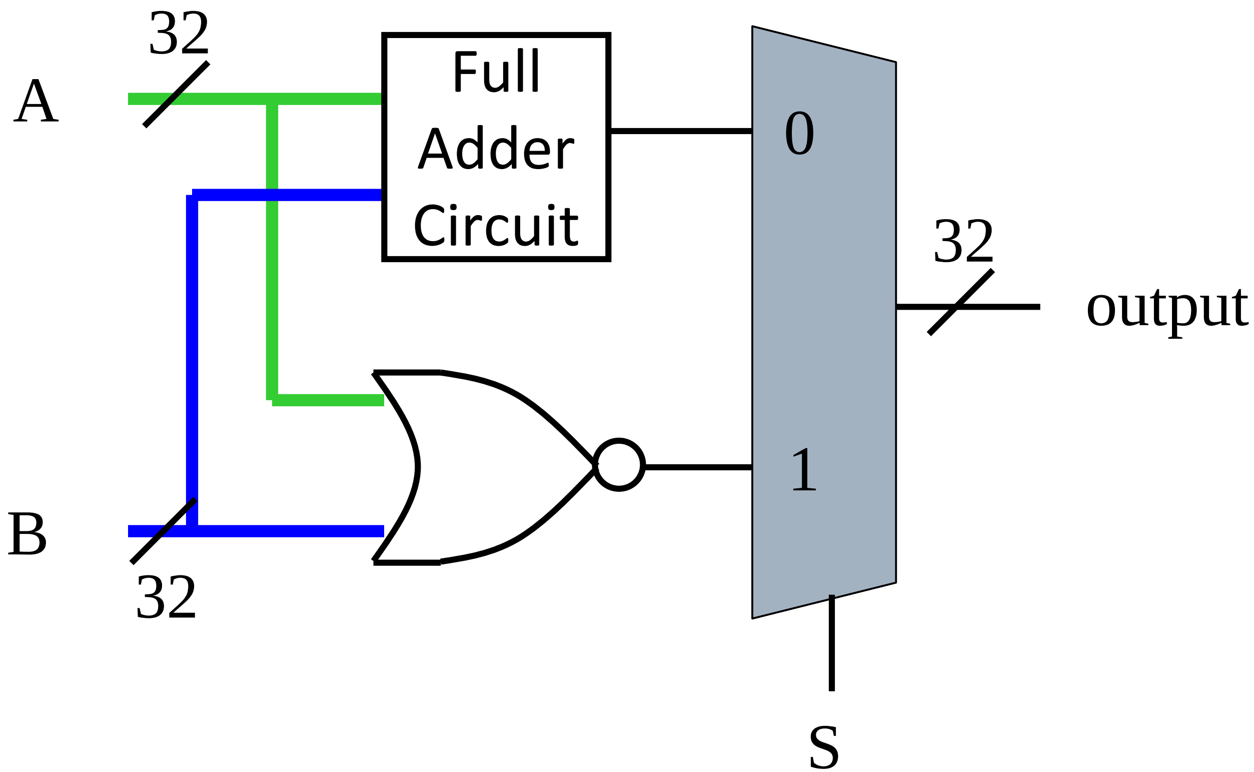
Exercise

- Use the blocks we have learned about so far (full adder, NOR, mux) to build this circuit
  - Input A, 32 bits
  - Input B, 32 bits
  - Input S, 1 bit
  - Output, 32 bits
  - When S is low, the output is A+B, when S is high, the output is NOR(a,b)
- Hint: you can express multi-bit gates like this:

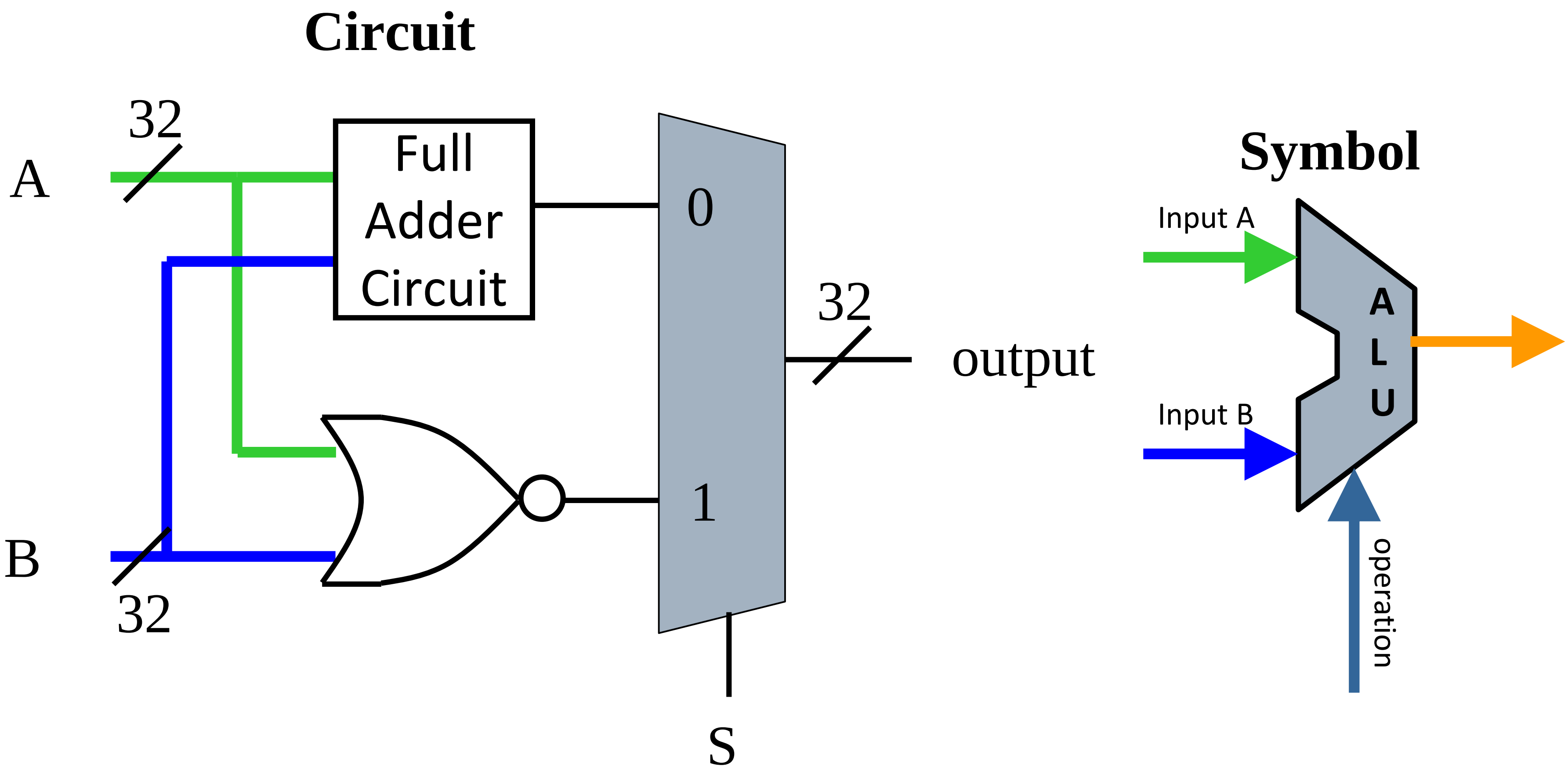


Exercise

- This is a basic ALU (Arithmetic Logic Unit)
- It is the heart of a computer processor!



LC-2K ALU


Circuit
32
A
Full
Adder
Circuit
0
32
output
1
B
32
S
Symbol
Input A
Input B
A
L
U
operation