L8_1 Floating-Point_Arithmetic

EECS 370 – Introduction to Computer Organization – Fall 2020

Learning Objectives

- To understand the algorithm for arithmetic operations using IEEE 754 floating-point values.

Floating Point Representation

Floating Point Review

$10.625_{10} \Rightarrow 1010.101_2$

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 $01010100000000000000000_2$

Floating Point - Example

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

1	10000101	01011001000000000000000

sign bit	1	- (negative)

exponent	10000101	133 – 127 = 6 (biased by 127)

significand	01011001000000000000000	add implicit 1

-1.01011001×2^6	shift radix point 6 places	-1010110.01

$$-1010110.01 = -(2^6 + 2^4 + 2^2 + 2^1 + 2^{-2}) = -(64 + 16 + 4 + 2 + ½) = -86.25_{10}$$

Floating Point Multiply - Example

Floating Point

10.625_{10}

10_{10}

Algorithm:

1. Convert to binary
2. Convert binary numbers to IEEE 754 floating-point
3. Multiply
    1. Sign bit – xor
    2. Add exponents - *mind the bias! (127)*
    3. Multiply significands

Floating Point Multiply - Example

Floating Point

$10.625_{10} = 1010.101_2$ ⇨

$10_{10} = 1010_2$ ⇨

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


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

1101010.01_2

$= 106.25_{10}$

Floating Point Addition

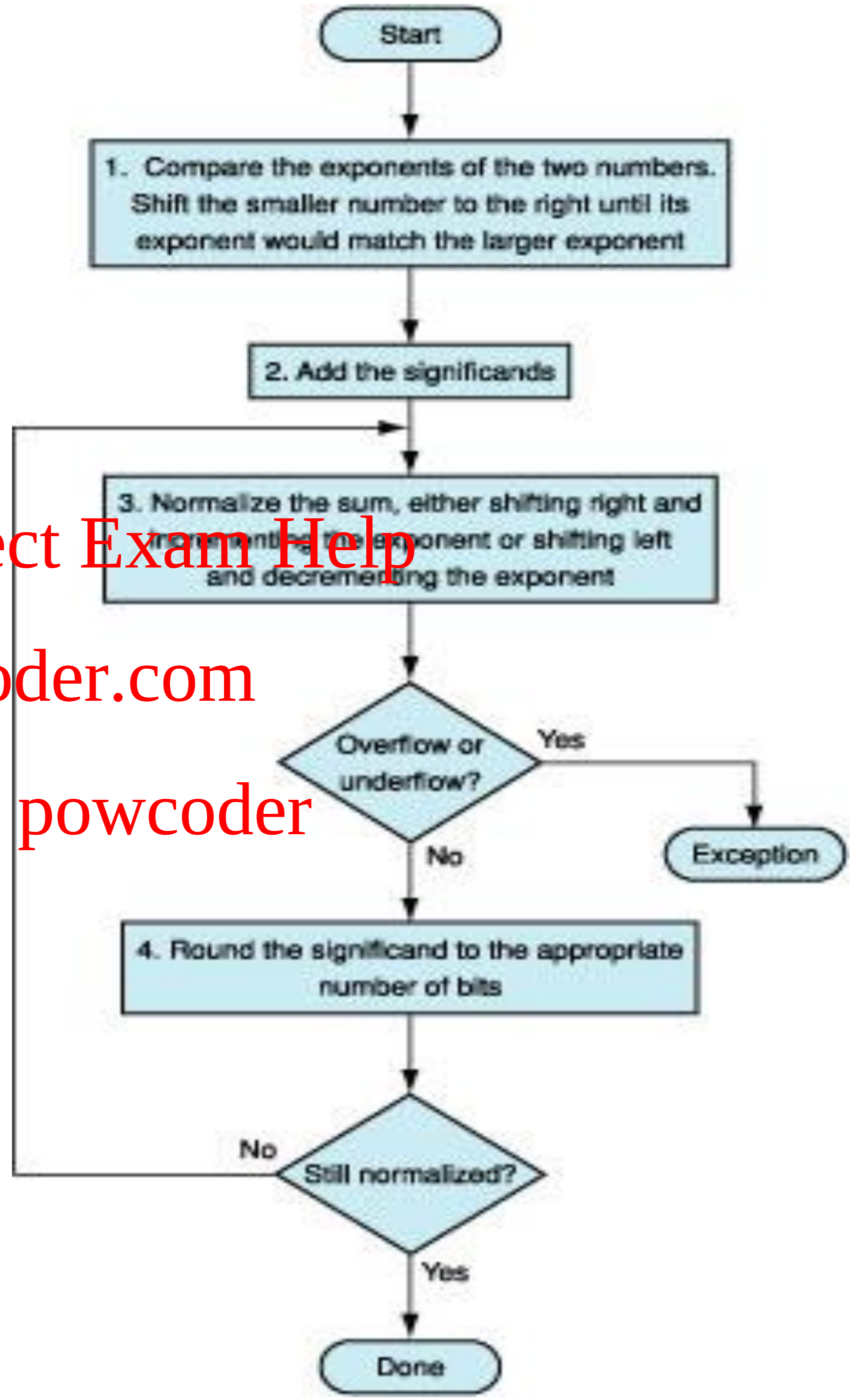
Floating Point

- 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 (shift to get back into proper "scientific notation")
- 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 right to match larger.
2. Add significands
3. Normalize and update exponent
4. Check for "out of range"

Normalize: shift significand (mantissa) for integer part to be 1 and remaining bits are fractions	Example: **1010.101_2** Normalized is **1.010101×2^3**



Floating Point Addition - Example

Floating Point

Problem: Add two numbers using IEEE floating point addition: **`101.125 + 13.75`**

1. Convert to IEEE 754 format
2. Shift smaller exponent right to match larger.
3. Add significands
4. Normalize and update exponent
5. Check for "out of range"

Floating Point Addition - Example

Floating Point

Problem: Add two numbers using IEEE floating point addition: **101.125 + 13.75**

	Sign	Exponent	Mantissa
101.125	0	10000101	10010100100000000000000
13.75	0	10000010	10111000000000000000000

Shift mantissa by difference in exponent

Shift by 6 - 3 = 3

	Sign	Exponent	Mantissa
13.75	0	10000101	00110111000000000000000
	0	10000101	11001011100000000000000

Sum Significands

```
  1100101001
+ 0001101110
------------
  1110010111
```

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

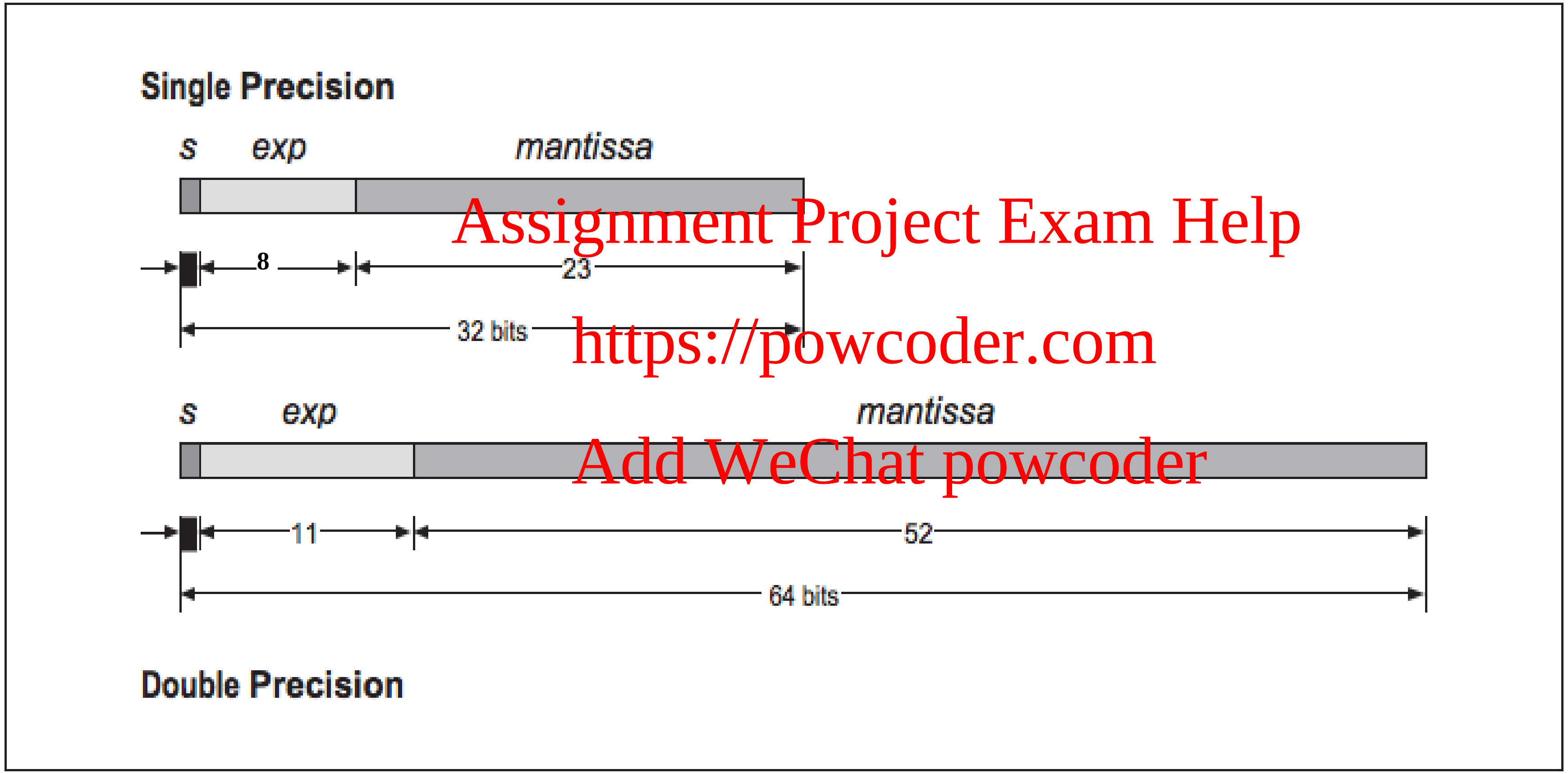
$= 1.110010111_2 \times 2^6 \quad = 114.875_{10}$

More Precision and Range

Floating Point

- 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 many but not all calculations
- IEEE-754 also defines a larger binary64 format, "double precision" (`double` data type 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

Floating Point Precision



Logistics

- There are 3 videos for lecture 8
  - L8_1 – IEEE_Floating-Point_Arithmetic
  - L8_2 – Basic-Electronics_Logic-Gates
  - L8_3 – Combinational-Logic
- There is one worksheet for lecture 8
  1. Logic gates – complete at the end of all 3 videos.

L8_2 Basic-Electronics_Logic-Gates

EECS 370 – Introduction to Computer Organization – Fall 2020

Office Hours

- Drop by office hours, ask questions, say ‘hi’
- Just in case you did not see them on the calendar:

Tuesdays 11am to 12 noon (EST) – individual meetings

Group office hours:

Tuesdays 4pm to 4:30 pm

Thursdays 9:45 am to 10:15 am

Thursdays 2:30 pm to 3:00 pm

https://umich.zoom.us/j/92153246345

Learning Objectives

- To identify logic gates used in combinational logic circuits and describe their operations.
- Be able to create the functionality of any logic gate with the NOR gate, (and therefore, the nor instruction in LC-2K).

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

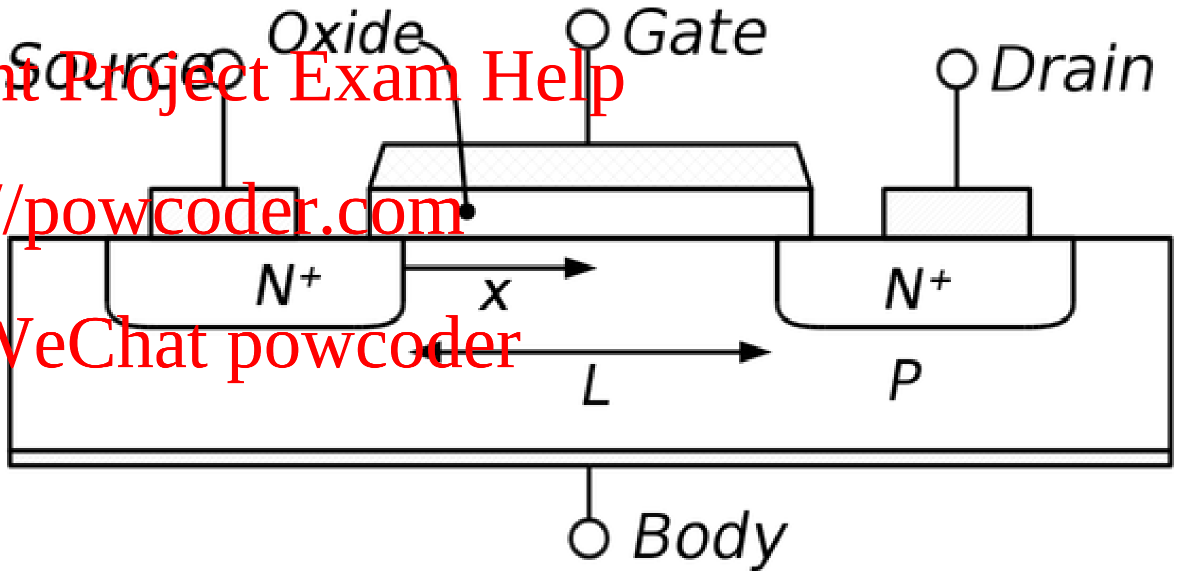
- Conductor: a material that permits electrical current to flow easily. (low resistance to current flow)
  - Lattice of atoms with free electrons

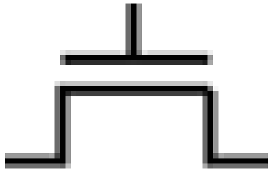
- Insulator: a material that is a poor conductor of electrical current (High resistance to current flow)
  - Lattice of atoms with strongly held electrons

- Semi-conductor: a material that can switch between an (okay) conductor and an (okay) insulator
  - Controlled via an external voltage
  - Basis for "logical switches" that make up digital circuits

Making a Transistor

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



- Electrical engineers use a symbol like this: 

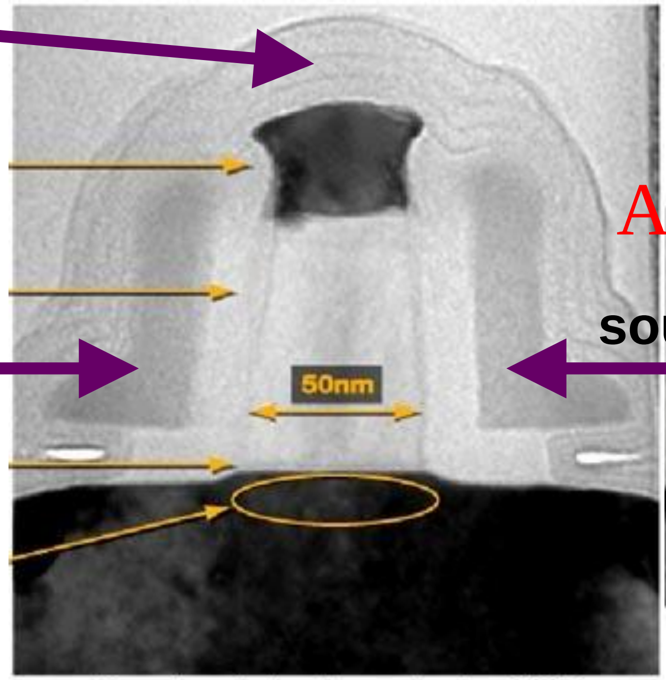
Recent Pictures and the Near Future

Source: Intel

gate

drain

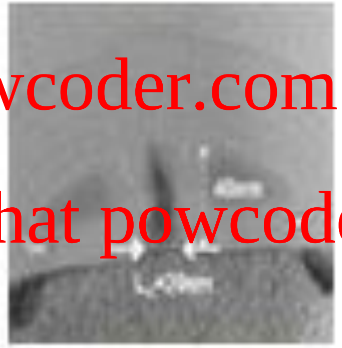
source



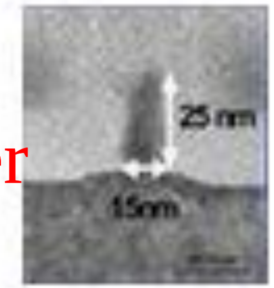
50nm transistor dimension is ~2000x smaller than diameter of human hair

Trend

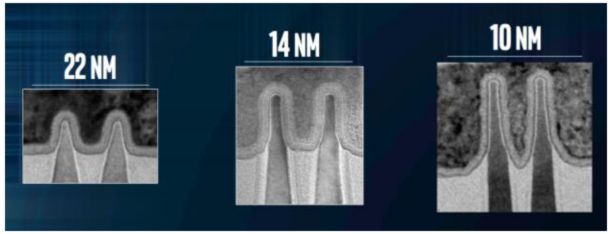
30nm



20nm



15nm



90nm technology 2003

65nm technology 2005

45nm technology 2008

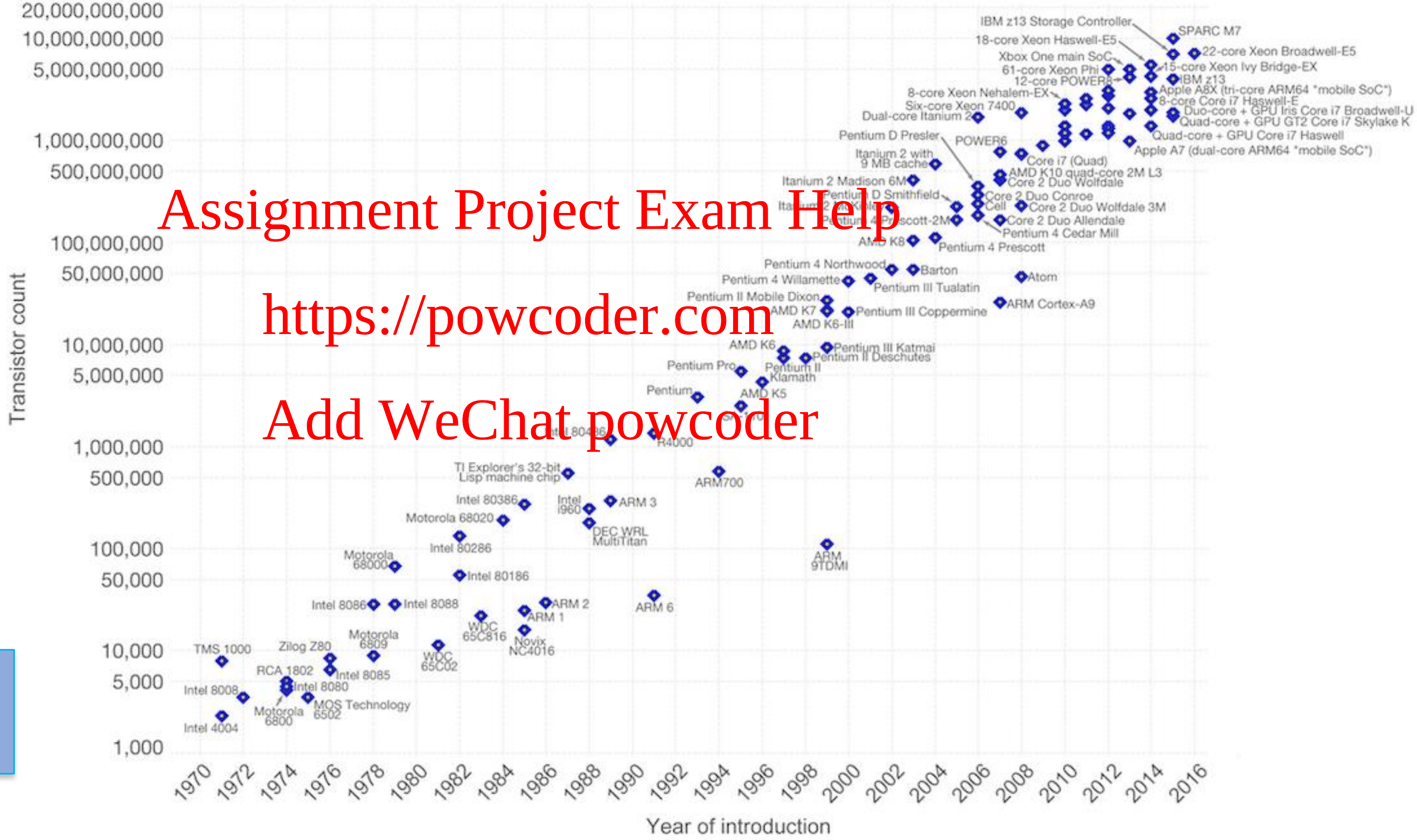
32nm technology 2010

14nm technology 2014

7nm technology 2018

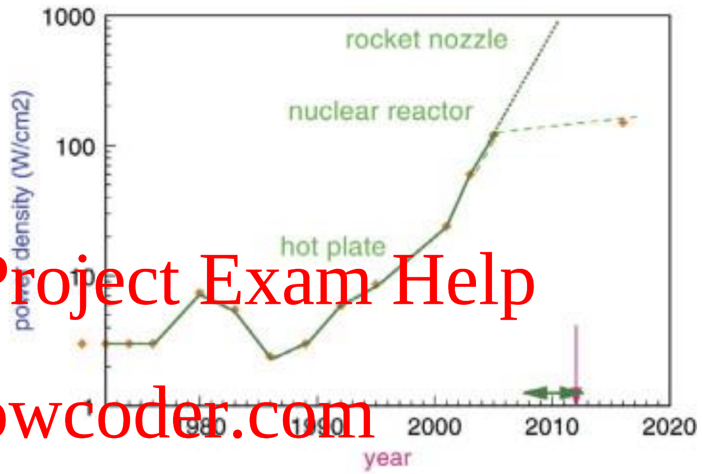
5nm technology 2020

Transistor Count



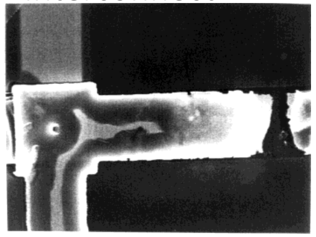
2020 - NVIDIA RTX 3090
28 Billion Transistors

Present and Future Problems

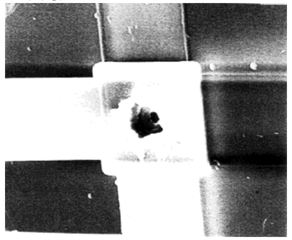


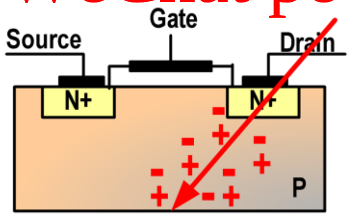
- Area is the least of them
- Power density – Watts/mm2
- Leakage current
- Reliability (faults)

interconnect

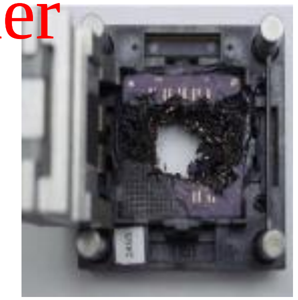


via





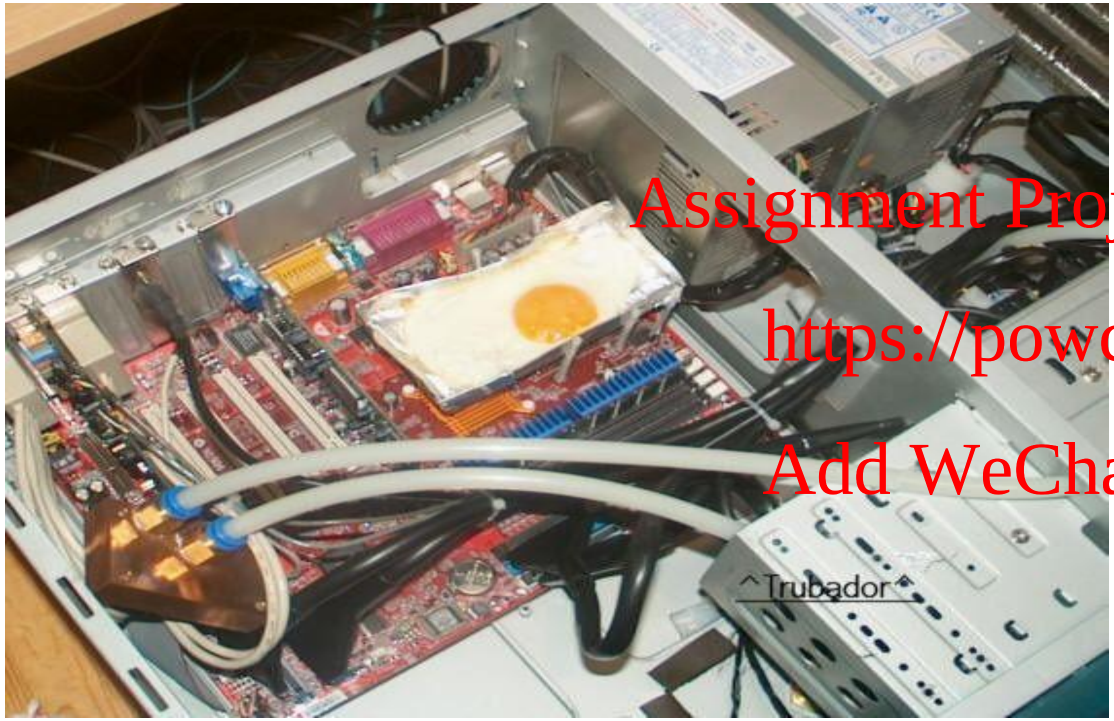
transients



Testing burn-in

- Process variation (not all transistors are equal)

As for power: Cooking-aware Computing



Source: The New York Times, 25 June 2002

Liquid Nitrogen Cooling

Basic Gate: Inverter

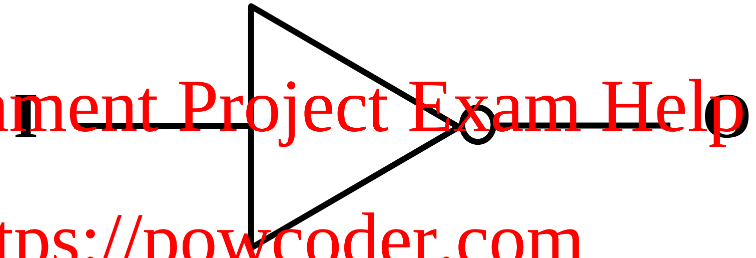
Logic Gates

CS abstraction
- logic function

Truth Table

I	O
0	1
1	0

Schematic symbol (CS/EE)



Logic Gates

Basic Gate: NAND

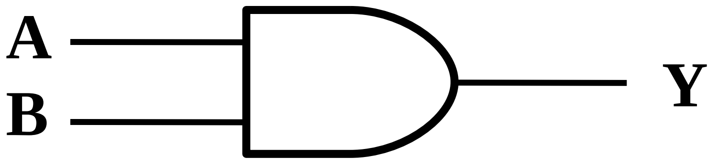
Truth Table

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

Logic Gates

Basic Gates: AND, OR, XOR

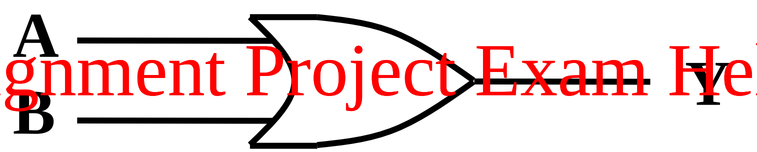
AND



Truth Table

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

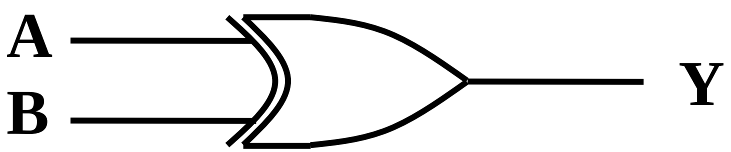
OR



Truth Table

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

XOR



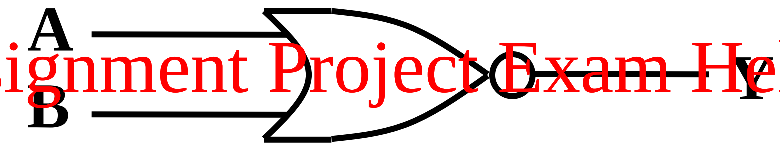
Truth Table

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

Logic Gates

Basic Gates: NOR

NOR



Truth Table

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

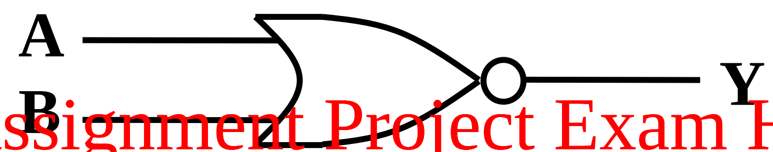
Logic Gate Exercise

- NOR is logically complete
  - This means that all gates can be implemented using only NORs
    - All gates can be implemented in LC2K
  - NAND is also logically complete
- Exercise:
  - Implement INV using only NOR gates
  - Implement AND using only NOR gates
  - Implement OR using only NOR gates
  - Hint Demorgan’s Law:
    - A || B = !(!A && !B)
    - !(A || B) = !A && !B

Logic Gate Exercise – INV (!) using NOR

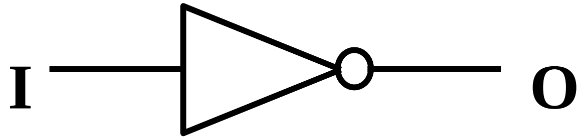
```
!(A || B) = A NOR B
```
substitute A for B
```
!(A || A) = A NOR A
!(A)      = A NOR A
!A        = A NOR A
```

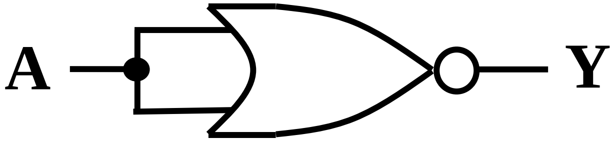
NOR



Truth Table

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





Truth Table

I	O
0	1
1	0

Logic Gate Exercise – AND using NOR

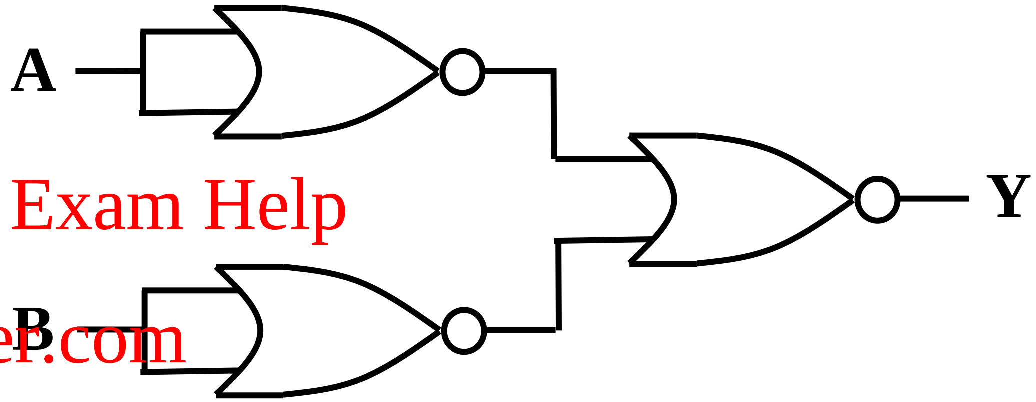
`!(A || B) = !A && !B = A NOR B`

`!A NOR !B = A && B`

substitute A NOR A for !A

substitute B NOR B for !B

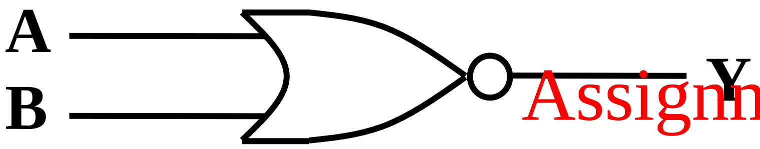
`A && B = (A NOR A) NOR (B NOR B)`



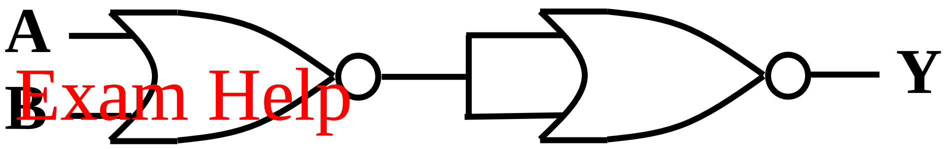
A	B	A && B	Y
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

Logic Gate Exercise – OR using NOR

NOR



OR



Truth Table

A	B	A||B	Y	!Y
0	0	0	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	0	1

Logistics

- There are 3 videos for lecture 8
  - L8_1 – IEEE_Floating-Point_Arithmetic
  - L8_2 – Basic-Electronics_Logic-Gates
  - L8_3 – Combinational-Logic
- There is one worksheet for lecture 8
  1. Logic gates – complete at the end of all 3 videos.

L8_3 Combinational-Logic

EECS 370 – Introduction to Computer Organization – Fall 2020

Learning Objectives

- To create circuits using combinations of basic gates.

Building Complexity: Addition (1)

Comb. Logic

GOAL: We want to design a circuit that performs binary addition

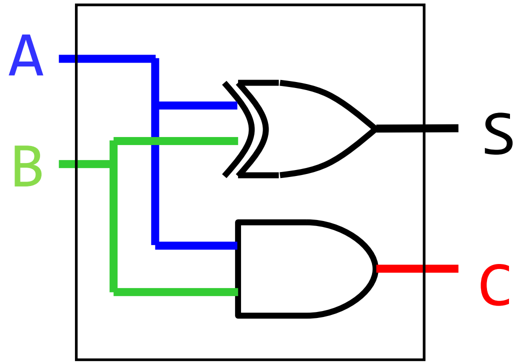
```
   0 1 1 0
  1 0 1 1 1
+ 0 0 1 1 0
-----------
  1 1 1 0 1
```

Let us start by adding two bits

- Design a circuit that takes two bits as input (A and B)
- Generates a sum and carry bit (S and C)

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

1. Make a truth table
2. Design a circuit



Half-Adder

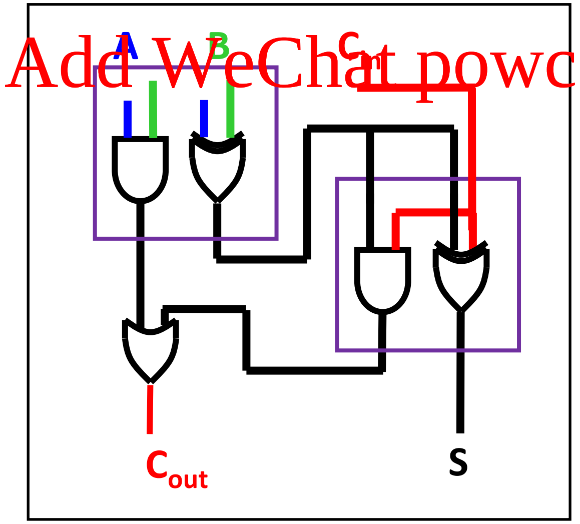
Comb. Logic

Building Complexity: Addition (1)

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

- Now we can add two bits, but how do we deal with carry bits?
  - We must 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?

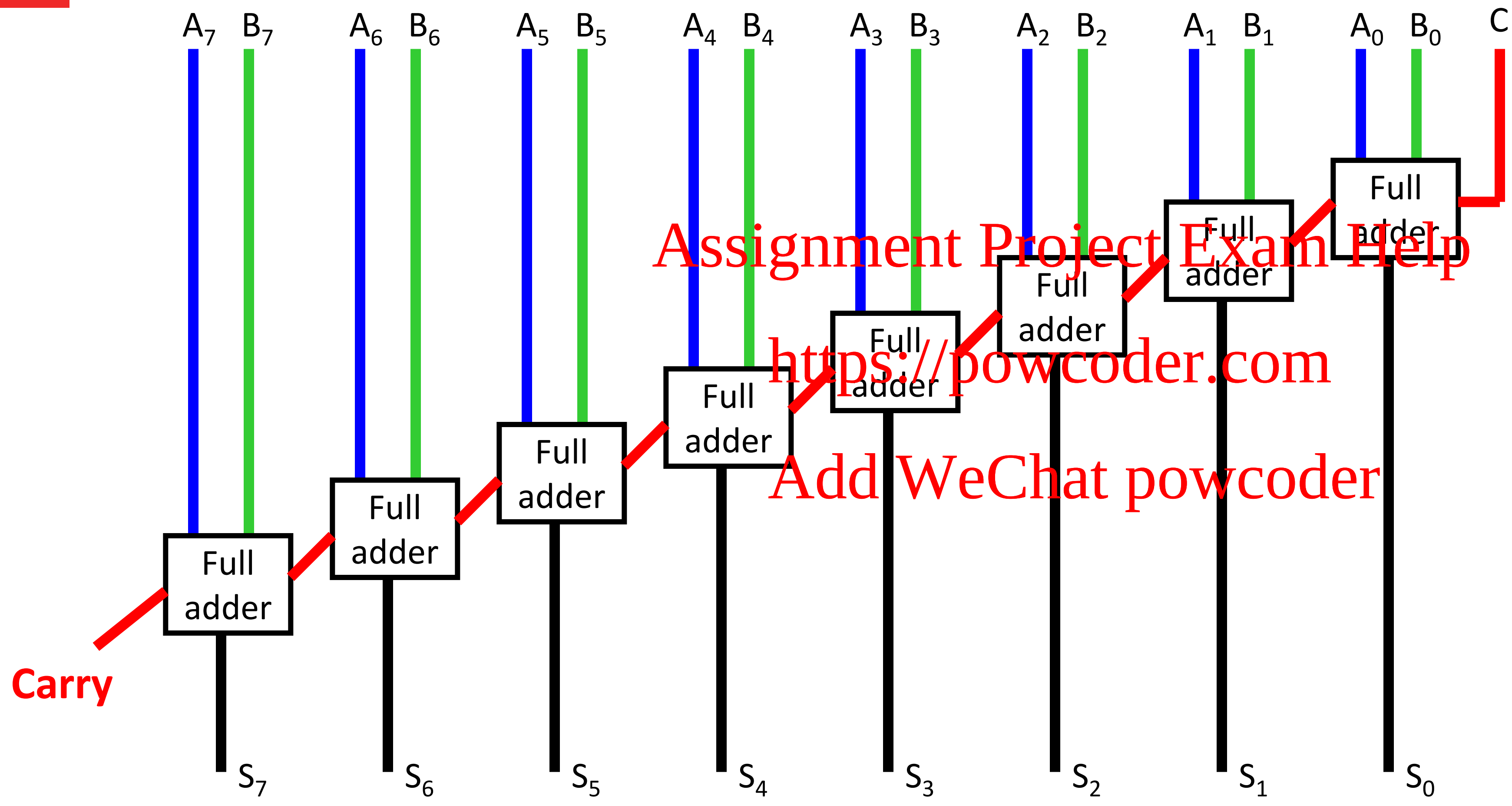
Full-Adder



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

Comb. Logic



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

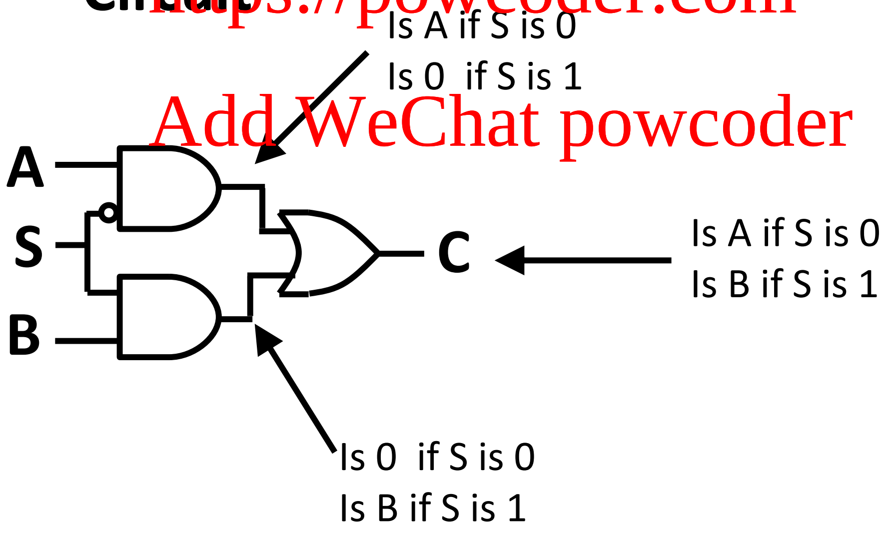
Building Complexity: Selecting

- We want to design a circuit that can select between two inputs - Let us start with a one-bit version

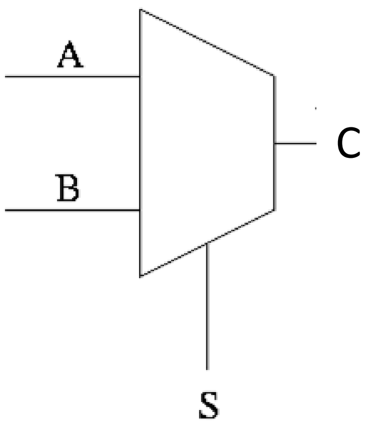
A	B	S	C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

1. Draw a truth table

Circuit



Symbol

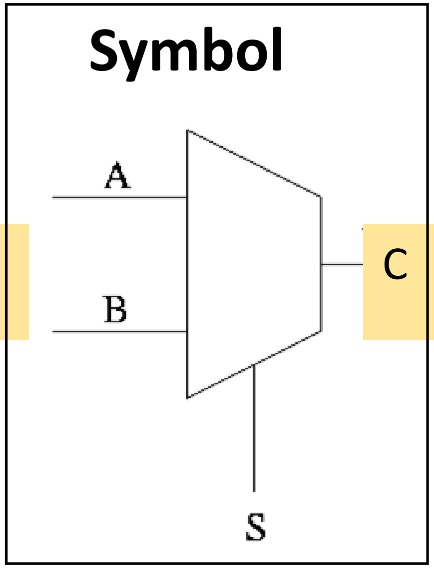


Multiplexer (Mux)

Multiplexer - Example

Comb. Logic

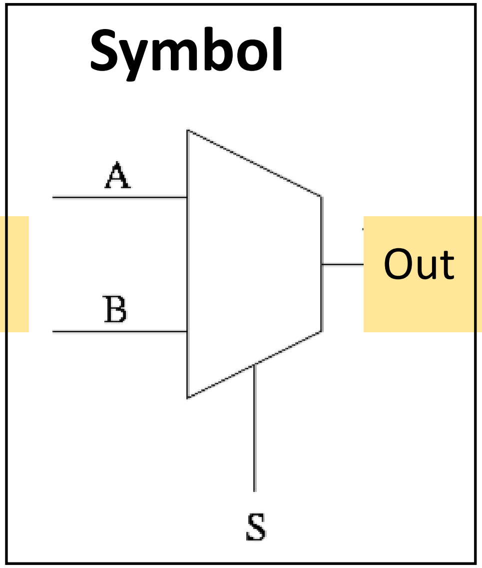
Problem: Build a 4x1 mux using only 2x1 muxes

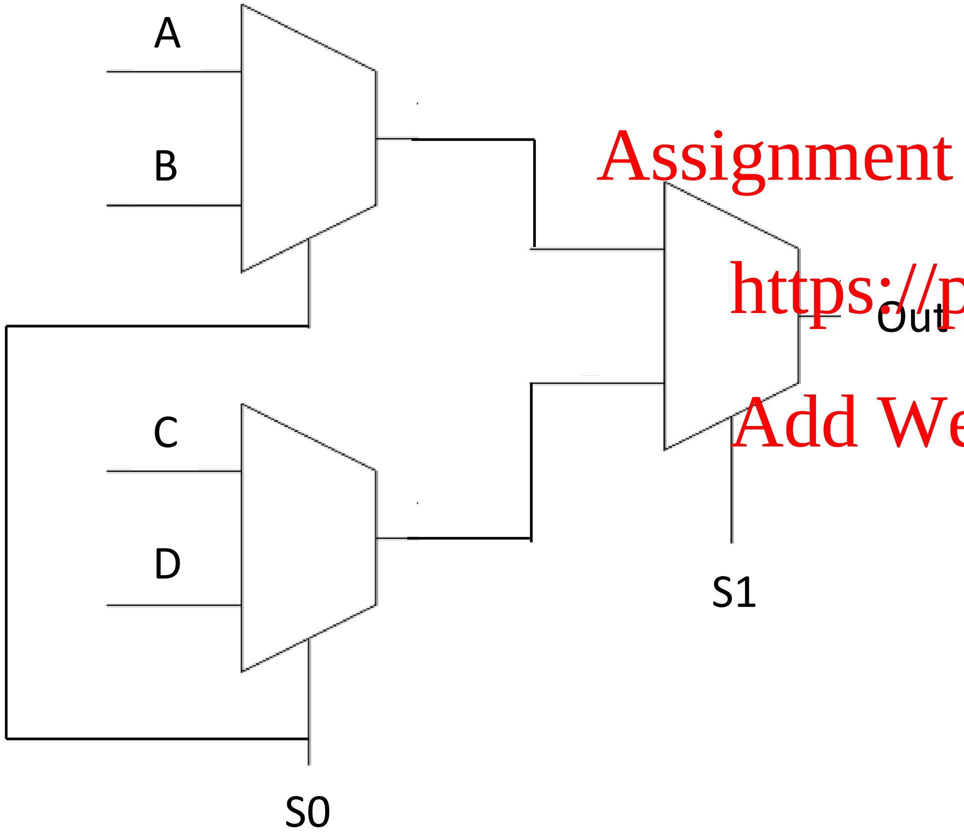


Multiplexer - Example

Comb. Logic

Problem: Build a 4x1 mux using only 2x1 muxes



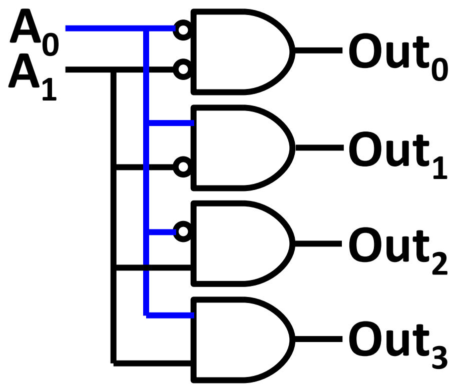


S1	S0	Out
0	0	A
0	1	B
1	0	C
1	1	D

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

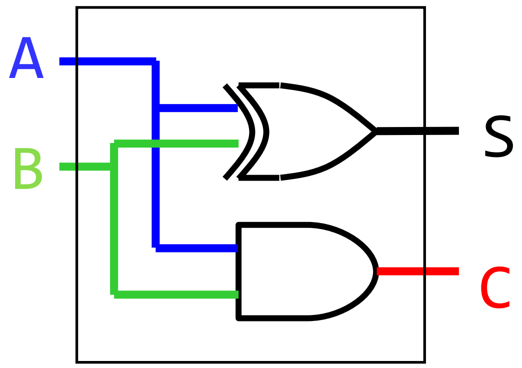
Decoder



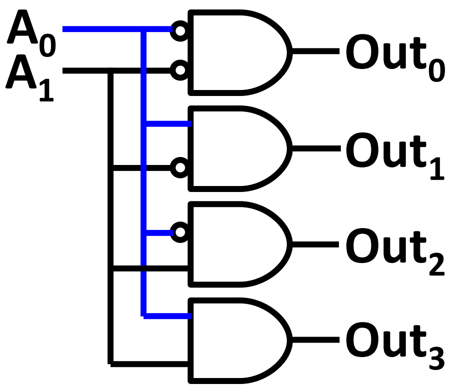
Combinational Circuits Implement Boolean Expressions

- 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 (Future lecture)
  - Decoder is the basic gate of indexing
  - MUX is the basic gate controlling data movement

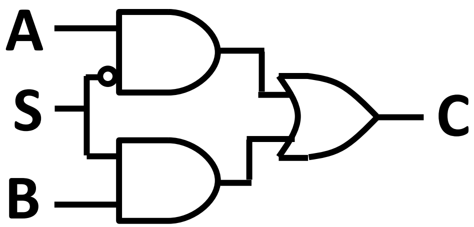
Half-Adder



Decoder



Mux



Logistics

- There are 3 videos for lecture 8
  - L8_1 – IEEE_Floating-Point_Arithmetic
  - L8_2 – Basic-Electronics_Logic-Gates
  - L8_3 – Combinational-Logic
- There is one worksheet for lecture 8
  1. Logic gates – complete at the end of all 3 videos.