Schedule…

Date	Day	Class No.	Title	Chapters	HW Due date	Lab Due date	Exam
12 Nov	Wed	21	Boolean Algebra	13.2 – 13			EXAM 2
13 Nov	Thu						
14 Nov	Fri		Recitation				
15 Nov	Sat						
16 Nov	Sun						
17 Nov	Mon	22	Combinational Logic	13.3 – 13.5		LAB 10	
18 Nov	Tue						
19 Nov	Wed	23	Sequential Logic	14.1			

Hardened or Softened by Afflictions

Alma 62:41

41 But behold, because of the exceedingly great length of the war between the Nephites and the Lamanites **many had become hardened**, because of the exceedingly great length of the war; and **many were softened because of their afflictions**, insomuch that they did humble themselves before God, even in the depth of humility.

Lecture 21 – Binary Numbers & Boolean Algebra

Signed Binary Integers

3 common representations for **signed** integers:

1. Sign magnitude
2. 1's compliment
3. 2's compliment

Most common for computers

For all 3 the **MSB** encodes the sign:

0 = +

1 = –

Sign-Magnitude

Range: $-\left(2^{n-1}-1\right) \longrightarrow \left(2^{n-1}-1\right)$

Representations

- 01111_{binary} => $15_{decimal}$
- 11111 => -15
- 00000 => 0
- 10000 => -0

The **MSB** encodes the sign:
`0` = +
`1` = –

Problem

- Difficult addition/subtraction
  - check signs
  - convert to positive
  - use adder or subtractor as required
- How to add two sign-magnitude numbers?
  - Ex: 1 + (-4)

1's Complement

Range: $\boxed{-\left(2^{n-1}-1\right) \longrightarrow \left(2^{n-1}-1\right)}$

Representations

- 00110_{binary} => $6_{decimal}$
- 11001 => -6
- 00000 => 0
- 11111 => -0

Problem

- Difficult addition/subtraction
  - no need to check signs as before
  - cumbersome logic circuits
    - *end-around-carry*
- How to add to one's complement numbers?
  - Ex: 4 + (-3)

To negate a number, Invert it, bit-by-bit.

MSB still encodes the sign:

0 = +

1 = –

Two's Complement

- Problems with sign-magnitude and 1's complement
  - two representations of zero (+0 and –0)
  - arithmetic circuits are complex
- ***Two's complement*** representation developed to make circuits easy for arithmetic.
  - only one representation for zero
  - just **ADD** the two numbers to get the right answer (regardless of sign)

Two's Complement

Range: $\boxed{-\left(2^{n-1}\right) \longrightarrow \left(2^{n-1}-1\right)}$

Representation:

- If number is **positive** or **zero**,
  - normal binary representation, zeroes in upper bit(s)
- If number is **negative**,
  - start with positive number
  - flip every bit (i.e., take the one's complement)
  - then add one

MSB still encodes the sign:
`0` = +
`1` = –

```
  00101   (5)
  11010   (1's comp)
+     1
  -----
  11011   (-5)
```

```
  01001   (9)
  10110   (1's comp)
+     1
  -----
  10111   (-9)
```

2's Complement

- Positional number representation with a twist
  - **MSB** has a *negative* weight

$$-2^{n-1} \quad 2^{n-2} \quad \cdots \quad 2^{1} \quad 2^{0}$$

0110 = 2^2 + 2^1 = 6

1110 = -2^3 + 2^2 + 2^1 = -2

2's Complement

- Positional number representation with a twist
  - **MSB** has a *negative* weight

```
0110 = 2² + 2¹ = 6
1110 = -2³ + 2² + 2¹ = -2
```

$$-2^{n-1} \quad 2^{n-2} \quad \cdots \quad 2^{1} \quad 2^{0}$$

$$-2^{n-1} \quad 2^{n-2} \quad \cdots \quad 2^{1} \quad 2^{0}$$

–MSB

2's Complement

- Positional number representation with a twist
  - **MSB** has a *negative* weight

```
0110 = 2² + 2¹ = 6
1110 = -2³ + 2² + 2¹ = -2
```

$$-2^{n-1} \quad 2^{n-2} \quad \cdots \quad 2^{1} \quad 2^{0}$$

$$-2^{n-1} \quad 2^{n-2} \quad \cdots \quad 2^{1} \quad 2^{0}$$

–MSB + remaining bits

2's Complement

- Positional number representation with a twist
  - **MSB** has a *negative* weight

$$-2^{n-1} \quad 2^{n-2} \quad \cdots \quad 2^{1} \quad 2^{0}$$

```
0110 = 2² + 2¹ = 6
1110 = -2³ + 2² + 2¹ = -2
```

$$-2^{n-1} \quad 2^{n-2} \quad \cdots \quad 2^{1} \quad 2^{0}$$

–MSB + remaining bits

```
11111111
```

2's Complement

- Positional number representation with a twist
  - **MSB** has a *negative* weight

$$0110 = 2^2 + 2^1 = 6$$

$$1110 = -2^3 + 2^2 + 2^1 = -2$$

$$\boxed{-2^{n-1} \quad 2^{n-2} \quad \cdots \quad 2^1 \quad 2^0}$$

$$\boxed{-2^{n-1} \quad 2^{n-2} \quad \cdots \quad 2^1 \quad 2^0}$$

–MSB + remaining bits

11111111

2's Complement

- Positional number representation with a twist
  - **MSB** has a *negative* weight

$$0110 = 2^2 + 2^1 = 6$$

$$1110 = -2^3 + 2^2 + 2^1 = -2$$

$$-2^{n-1} \quad 2^{n-2} \quad \cdots \quad 2^1 \quad 2^0$$

$$-2^{n-1} \quad 2^{n-2} \quad \cdots \quad 2^1 \quad 2^0$$

–MSB + remaining bits

$$11111111 = -2^7 + 2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0$$

$= -1$

Two's Complement Shortcut

- To take the two's complement of a number:
  1. copy bits from right to left until (and including) the first **`1`**
  2. flip remaining bits to the left

```
  011010000
  100101111   (1's comp)
+         1
 __________
  100110000
```

```
0110|10000
(flip)  (copy)
1001|10000
```

Two's Complement

<u>Example3</u>: What is $\mathbf{0110101}_2$ in decimal?

What is it's 2's complement?

Two's Complement

<u>Example3</u>: What is $\mathbf{0110101}_2$ in decimal?
What is it's 2's complement?

$$\mathbf{0110101}_2 = 0\times2^6 + 1\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$$

$$= 53_{10}$$

Two's Complement

<u>Example3</u>: What is $\mathbf{0110101}_2$ in decimal?
What is it's 2's complement?

$$\mathbf{0110101}_2 = 0\times2^6 + 1\times2^5 + 1\times2^4 + 0\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$$

$$= 53_{10}$$

```
 0110101        (53)
 1001010        (1's comp)
+      1
 -------
 1001011        (-53)
```

Two's Complement Negation

- To negate a number, invert all the bits and add 1 (or use shortcut)

Number	Decimal Value	Negated Binary Value
0110	6	1010
0111	7	1001
0000	0	0000
1111	-1	0001
0100	4	1100
1000	-8	1000 (??)

Signed Binary Numbers

Binary	Sign-magnitude	1's compliment	2's complement
0 0 0	0	0	0
0 0 1	1	1	1
0 1 0	2	2	2
0 1 1	3	3	3
1 0 0	-0	-3	-4
1 0 1	-1	-2	-3
1 1 0	-2	-1	-2
1 1 1	-3	-0	-1

Decimal to Binary Conversion

Positive numbers

- ❶ start with empty result
- ❷ if decimal number is odd, prepend ‘**`1`**’ to result
  else prepend ‘**`0`**’
- ❸ divide number by 2, throw away fractional part (INTEGER divide)
- ❹ if number is non-zero, go back to ❷ else you are done

Negative numbers

- do above for **positive version** of number and **negate result.**

Decimal to Binary Conversion

Number	Binary Value
5	0101
6	0110
123	01111011
35	00100011
-35	1011101
1007	01111101111

Hexadecimal Notation

- Binary is hard to read and write by hand
- Hexadecimal is a common alternative
  - 16 digits are **0123456789ABCDEF**

```
0100 0111 1000 1111 = 0x478F
1101 1110 1010 1101 = 0xDEAD
1011 1110 1110 1111 = 0xBEEF
1010 0101 1010 0101 = 0xA5A5
```

0x is a common prefix for writing numbers which means *hexadecimal*

Binary	Hex	Dec
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

1. Separate binary code into groups of 4 bits (starting from the right)
2. Translate each group into a single hex digit

Binary to Hex Conversion

- Every four bits is a hex digit.
  - start grouping from right-hand side

011	1010	1000	1111	0100	1101	0111
⇓	⇓	⇓	⇓	⇓	⇓	⇓
3	**A**	**8**	**F**	**4**	**D**	**7**

This is not a new machine representation, just a convenient way to write the number.

Boolean Algebra

Boolean Algebra

<u>Boolean Algebra</u>: the mathematics associated with binary numbers

- Developed by George Boole in 1854

Variables in boolean algebra can take only one of two possible values:

`0` → **`FALSE`**

`1` ⟶ **`TRUE`**

Logic Functions

3 different ways to represent logic functions:

1. **<u>Equation</u>**: a mathematical representation of a logic function

A bar over a variable represent an inverting or a **NOT** operation

$$out = \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab$$

Final logic output

Each letter variable represents a top-level input to the logic function

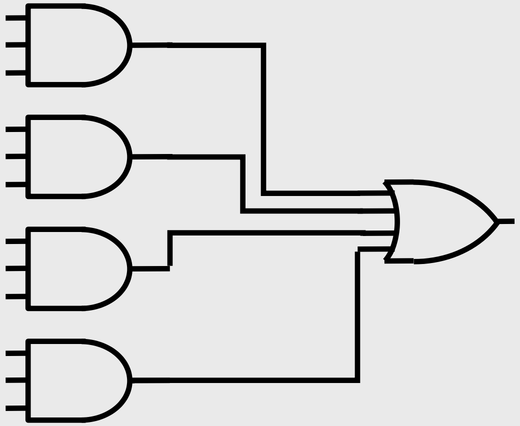
Mathematical operations (i.e. addition and multiplication) are boolean algebra operations

Logic Functions

3 different ways to represent logic functions:

2. **<u>Gates</u>**: a visual block representation of the function

Top-level inputs



Final output

Four 3-input **AND** gates feeding into one 4-input **OR** gate

Logic Functions

3 different ways to represent logic functions:

3. **<u>Truth Table</u>**: indicates what the output will be for every possible input combination

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

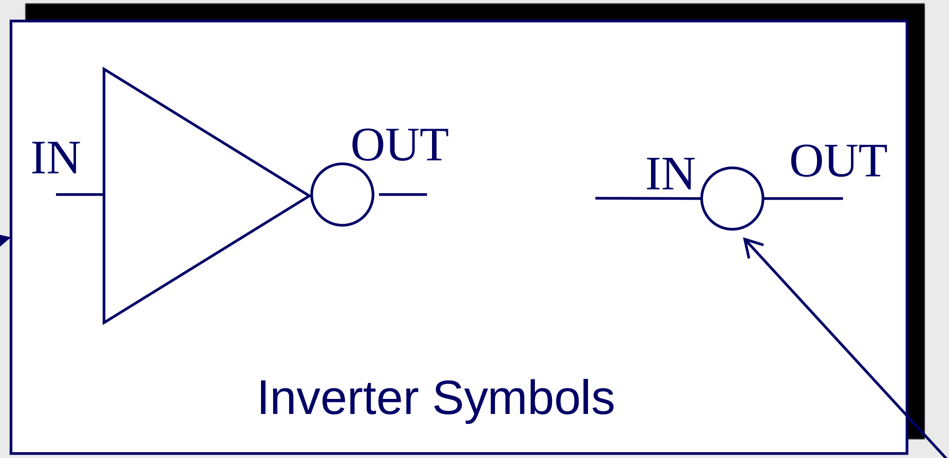
If there are n inputs (left-hand columns) there will be 2^n entries (rows) in the table
<u>EX</u>: 3 inputs require $2^3 = 8$ rows

There will always be at least one output (right-hand columns)

For each input combination (row) outputs will be either **`0`** or **`1`**

The Inverter

Gate



Inverter Symbols

Usually abbreviated with just a 'bubble' next to another gate input/output

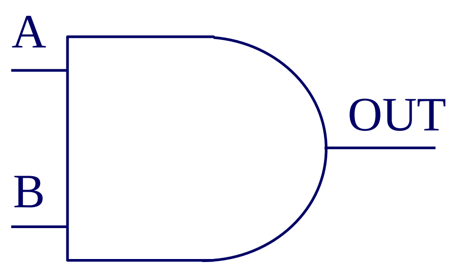
Truth-table

IN	OUT
0	1
1	0

$$OUT = \overline{IN}$$

Equation

The AND Gate



AND Symbol

Gate

Truth Table

A	B	OUT
0	0	0
0	1	0
1	0	0
1	1	1

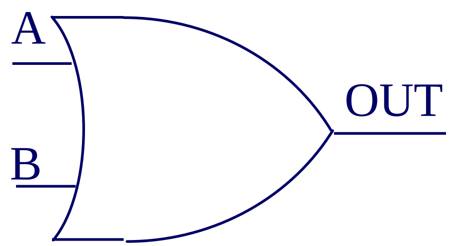
$$OUT = A \cdot B$$

Equation

NB: multiplication operation → AND

The OR Gate

Gate



OR Symbol

Truth Table

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	1

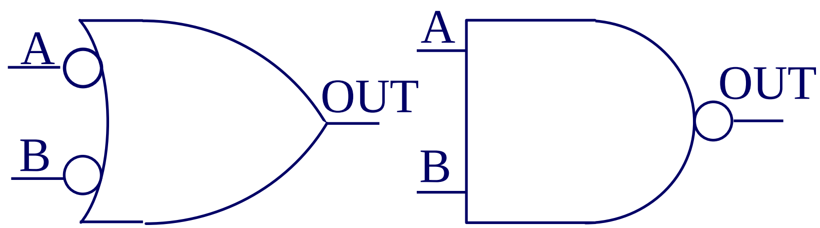
$$OUT = A + B$$

Equation

NB: addition operation → OR

The NAND Gate (NOT-AND)

Gate



NAND Symbols

Truth Table

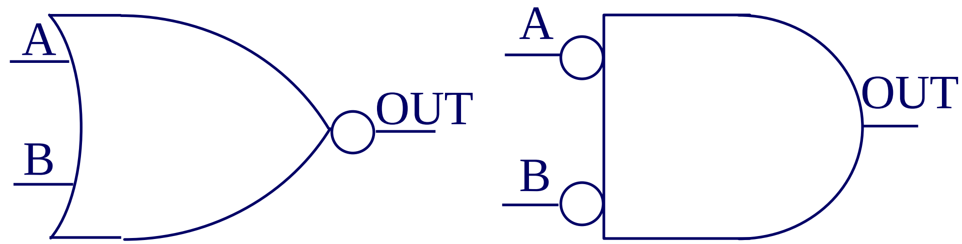
A	B	OUT
0	0	1
0	1	1
1	0	1
1	1	0

Equation

$$OUT = \overline{A \cdot B}$$

The NOR Gate (NOT-OR)

Gate



NOR Symbols

Truth Table

A	B	OUT
0	0	1
0	1	0
1	0	0
1	1	0

$$OUT = \overline{A+B}$$

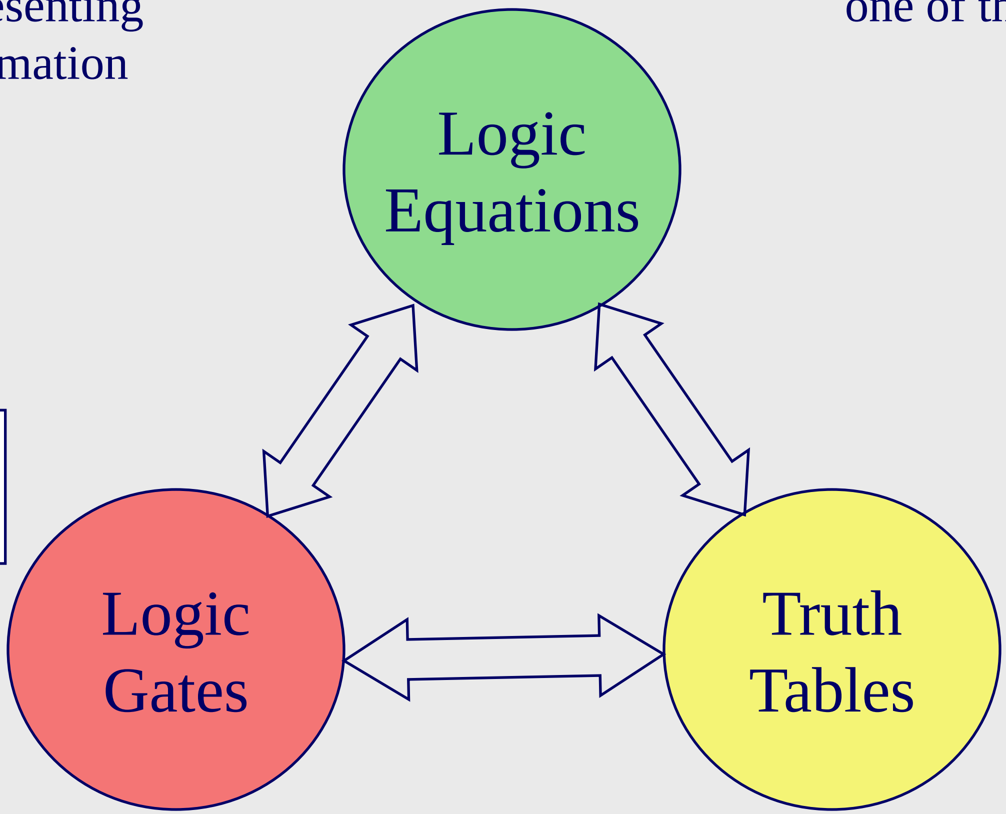
Equation

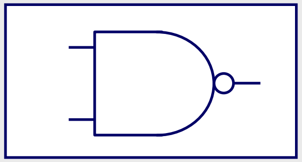
You should know how to Translate

These are three different ways of representing logical information

$$\overline{A \cdot B}$$

You can convert any one of them to any other





A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	1

Equations to Gates

$$y = \mathrm{NOT}(s) \;\; \mathrm{AND} \;\; a \;\; \mathrm{AND} \;\; \mathrm{NOT}(b)$$

OR

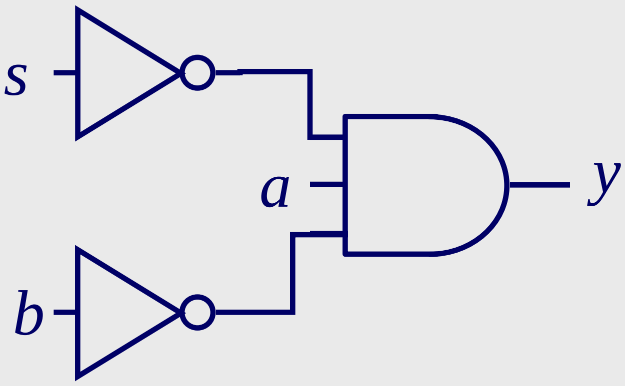
$$y = \overline{s} \cdot a \cdot \overline{b}$$

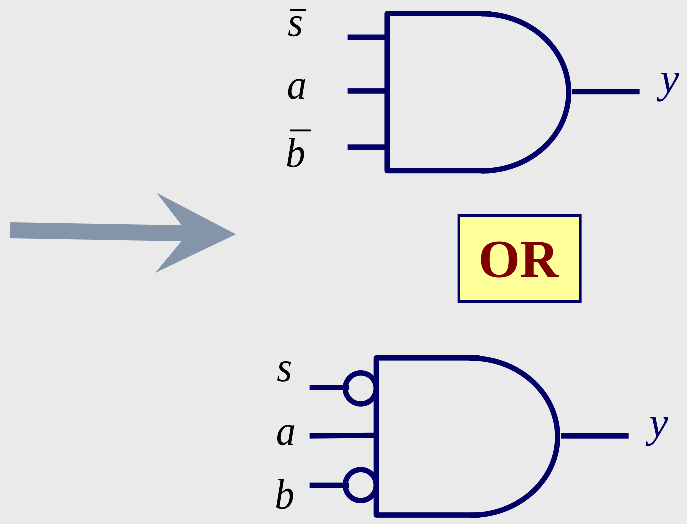
Equations to Gates

$$y = \text{NOT}(s) \;\; \text{AND} \;\; a \;\; \text{AND} \;\; \text{NOT}(b)$$

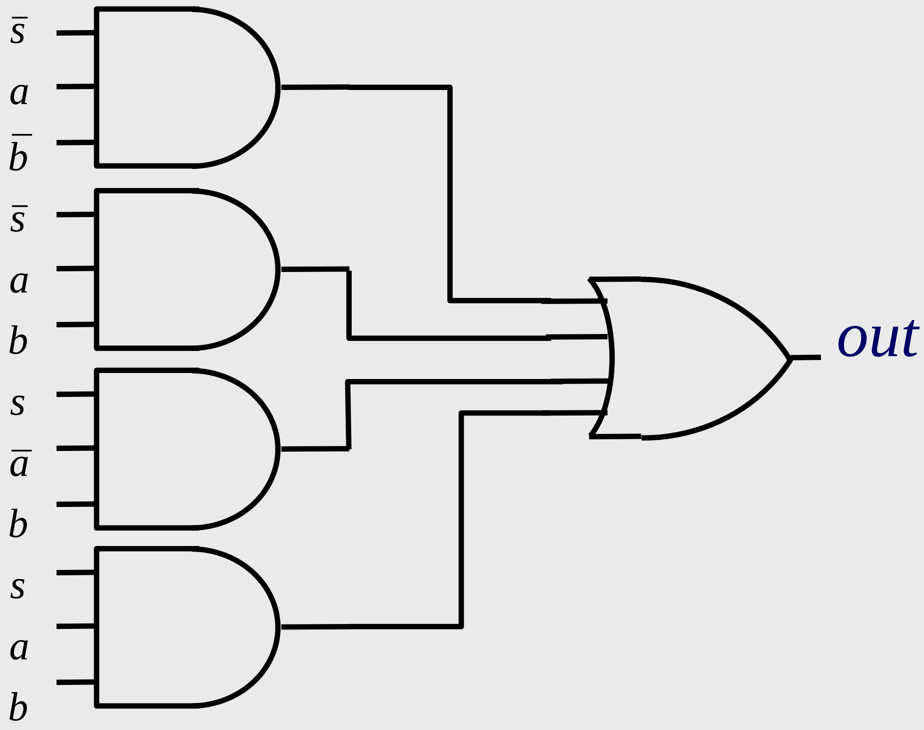
OR

$$y = \bar{s} \cdot a \cdot \bar{b}$$

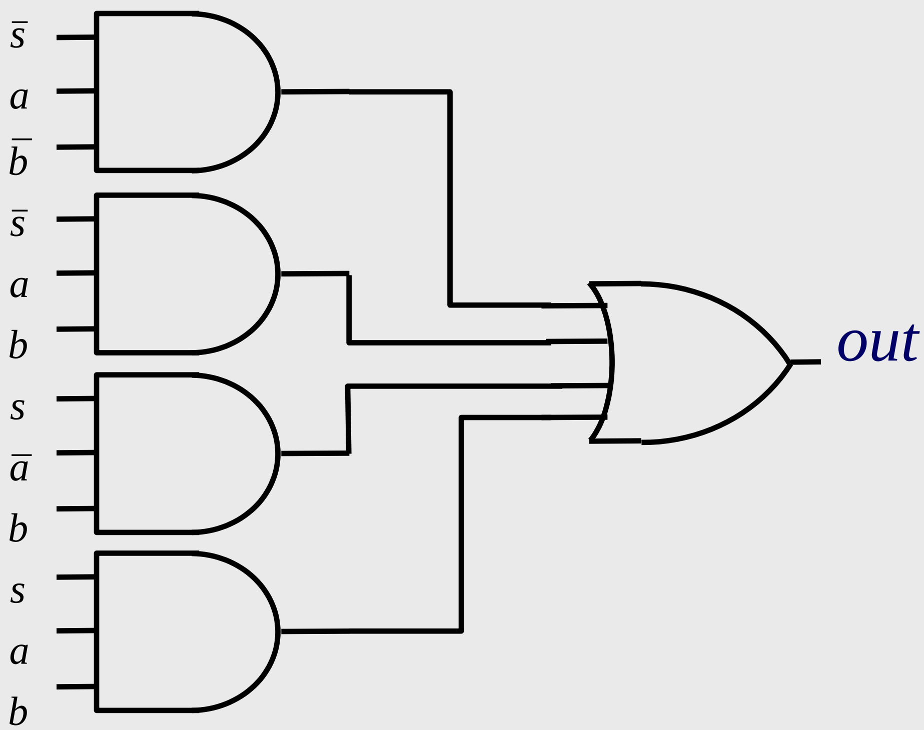




Gates to Equations



Gates to Equations



$$out = (\bar{s} \cdot a \cdot \bar{b}) + (\bar{s} \cdot a \cdot b) + (s \cdot \bar{a} \cdot b) + (s \cdot a \cdot b)$$

OR

$$out = \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab$$

Truth Tables to Gates

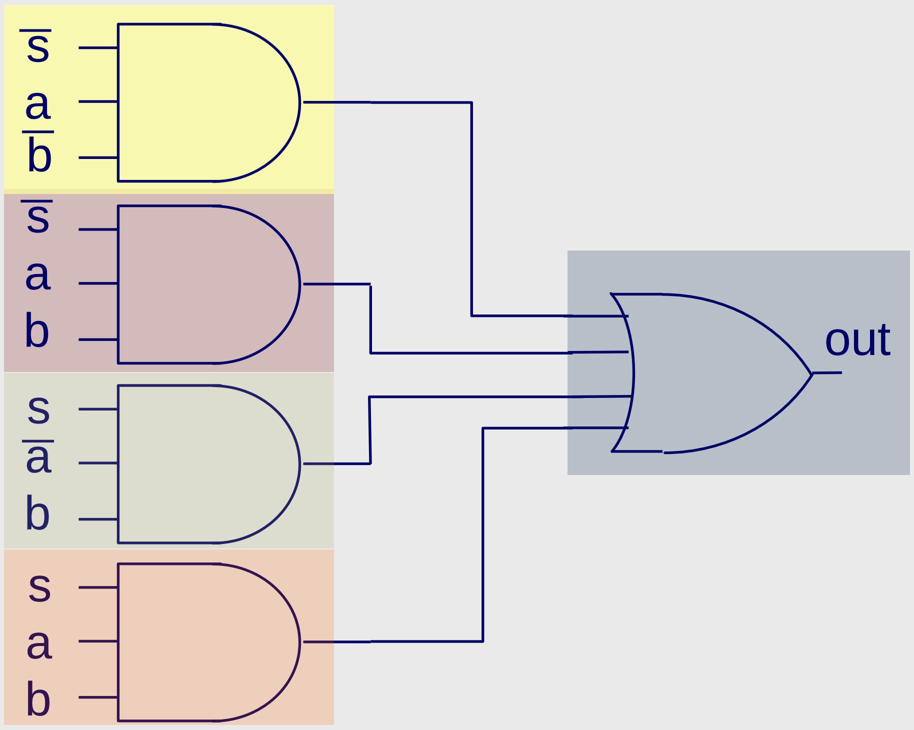
- Each row of truth table is an **AND** gate
- Each output column is an **OR** gate

S	A	B	OUT
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

Truth Tables to Gates

- Each row of truth table is an **AND** gate
- Each output column is an **OR** gate

S	A	B	OUT
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



Truth Table to Equations

- Write out truth table a combination of **AND**'s and **OR**'s
  - equivalent to gates
  - easily converted to gates

S	A	B	OUT
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

Truth Table to Equations

- Write out truth table a combination of **AND**'s and **OR**'s
  - equivalent to gates
  - easily converted to gates

S	A	B	OUT
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

$$out = \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab$$

Equations to Truth Tables

- For each **AND** term
  - fill in the proper row on the truth table

$$out = \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab$$

Equations to Truth Tables

- For each **AND** term
  - fill in the proper row on the truth table

$$out = \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab$$

S	A	B	OUT
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