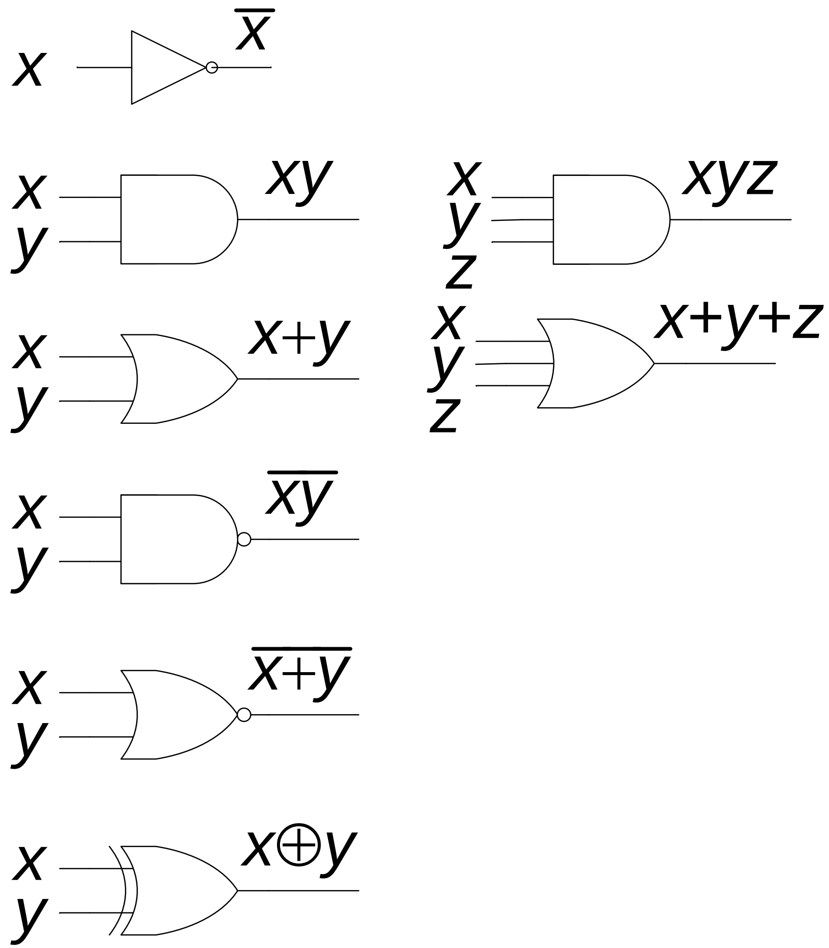
Boolean Algebra

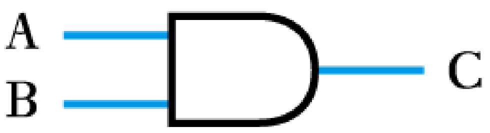
Logic Gates

Basic logic gates

- Not
- And
- Or
- Nand
- Nor
- Xor



The AND gate



(a) Circuit symbol

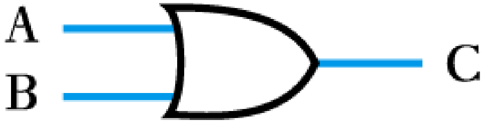
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

$$C = A \cdot B$$

(c) Boolean expression

The OR gate



(a) Circuit symbol

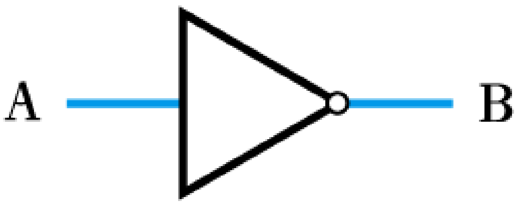
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

$$C = A + B$$

(c) Boolean expression

The NOT gate (or inverter)



(a) Circuit symbol

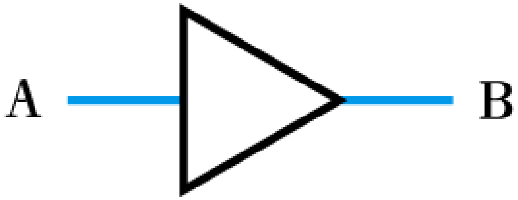
A	B
0	1
1	0

(b) Truth table

$$B = \overline{A}$$

(c) Boolean expression

A logic buffer gate



(a) Circuit symbol

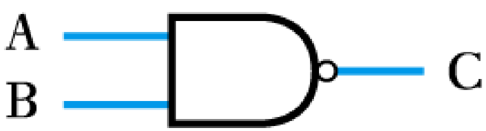
A	B
0	0
1	1

(b) Truth table

$$B = A$$

(c) Boolean expression

The NAND gate



(a) Circuit symbol

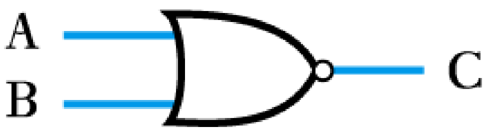
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

(b) Truth table

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

(c) Boolean expression

The NOR gate



(a) Circuit symbol

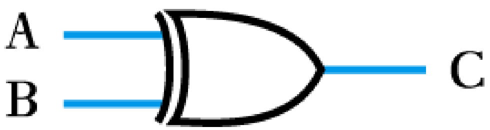
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

(b) Truth table

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

(c) Boolean expression

The Exclusive OR gate



(a) Circuit symbol

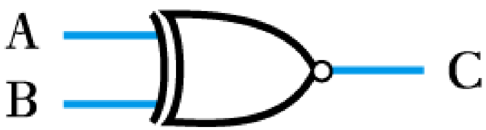
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

(b) Truth table

$$C = A \oplus B$$

(c) Boolean expression

The Exclusive NOR gate



(a) Circuit symbol

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

(b) Truth table

$$C = \overline{A \oplus B}$$

(c) Boolean expression

Boolean Algebra

- **Boolean Constants**
  – these are ‘0’ (false) and ‘1’ (true)
- **Boolean Variables**
  – variables that can only take the vales ‘0’ or ‘1’
- **Boolean Functions**
  – each of the logic functions (such as AND, OR and NOT) are represented by symbols as described above
- **Boolean Theorems**
  – a set of **identities** and **laws** – see text for details

Boolean laws

$$AB = BA$$

$$A + B = B + A$$

$$A + AB = A$$

$$A(A + B) = A$$

$$A(B + C) = AB + BC$$

$$A + BC = (A + B)(A + C)$$

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

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

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

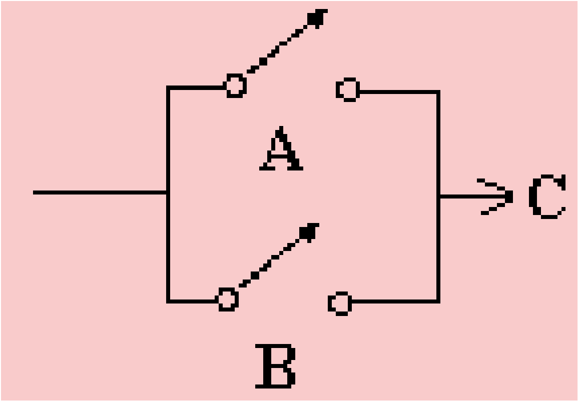
$$A(\overline{A} + B) = AB$$

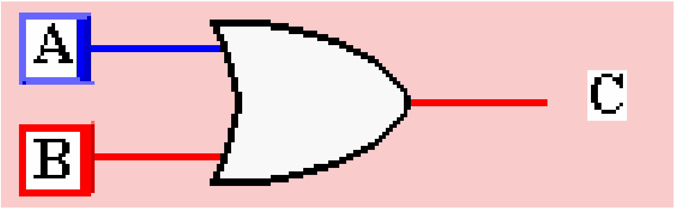
$$A(BC) = (AB)C$$

$$A + (B + C) = (A + B) + C$$

OR Gate

- Current flows if either switch is closed
  - Logic notation A + B = C



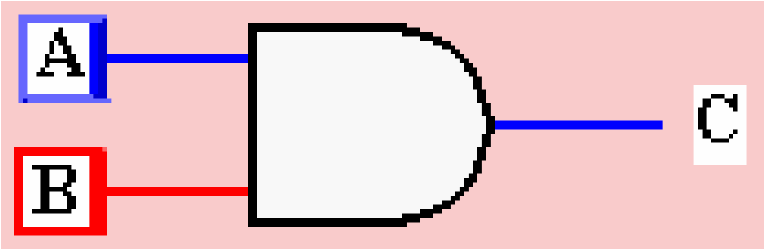


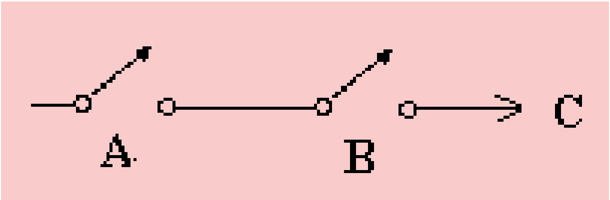
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

AND Gate

- In order for current to flow, both switches must be closed
  - Logic notation A•B = C

    (Sometimes AB = C)

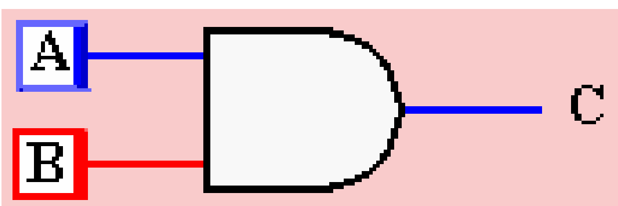




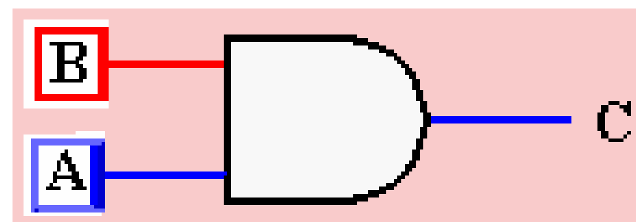
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

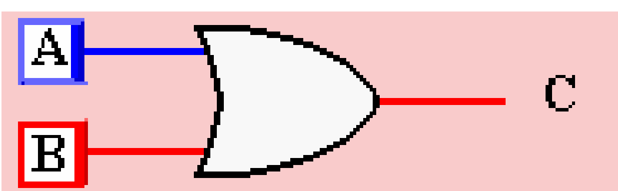
Properties of AND and OR

- Commutation
  - A + B = B + A
  - A • B = B • A

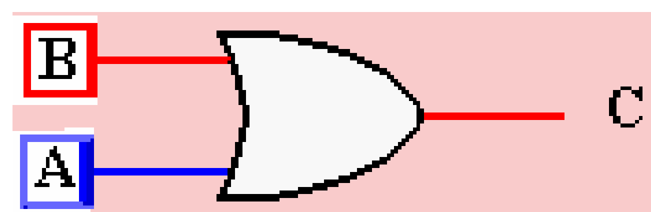


Same as

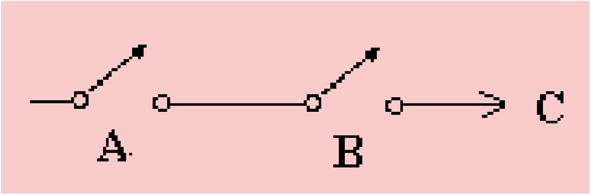




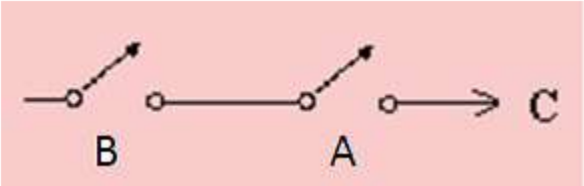
Same as



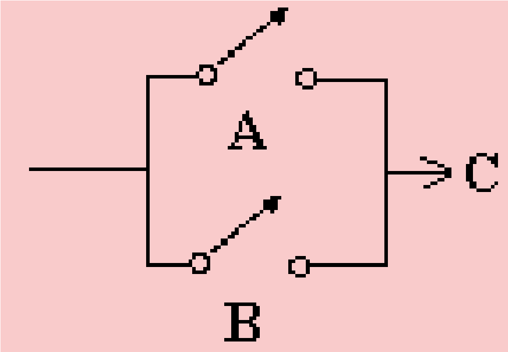
Commutation Circuit



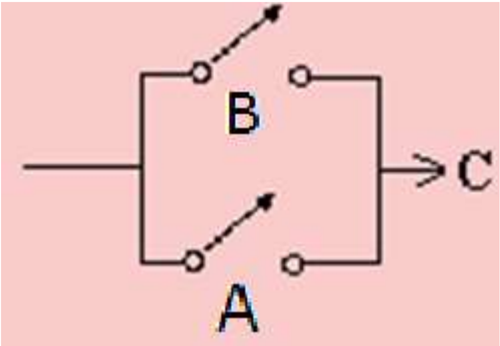
A • B



B • A



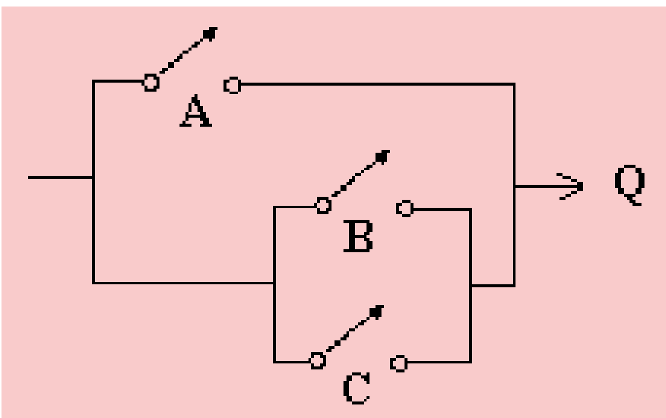
A + B



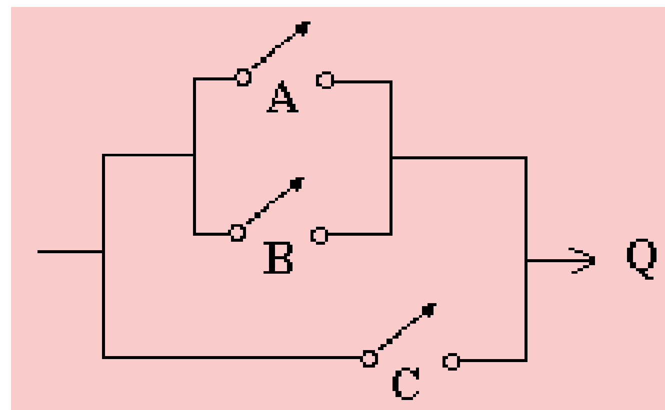
B + A

Properties of AND and OR

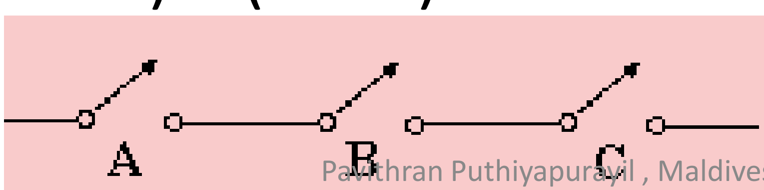
- Associative Property
  - ❖A + (B + C) = (A + B) + C



=

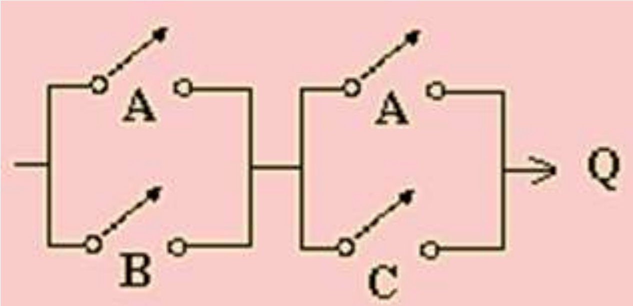


  - ❖A • (B • C) = (A • B) • C



Distributive Property

(A + B) • (A + C)



A	B	C	Q
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

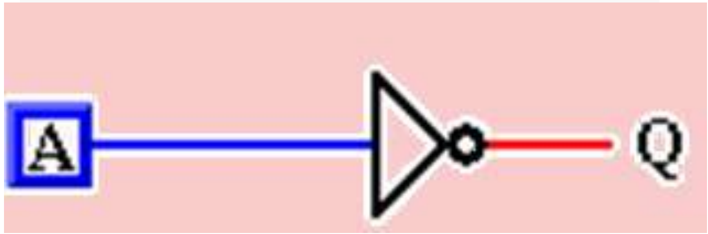
Binary Addition

A	B	S	C(arry)
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

Notice that the carry results are the same as AND

$$C = A \bullet B$$

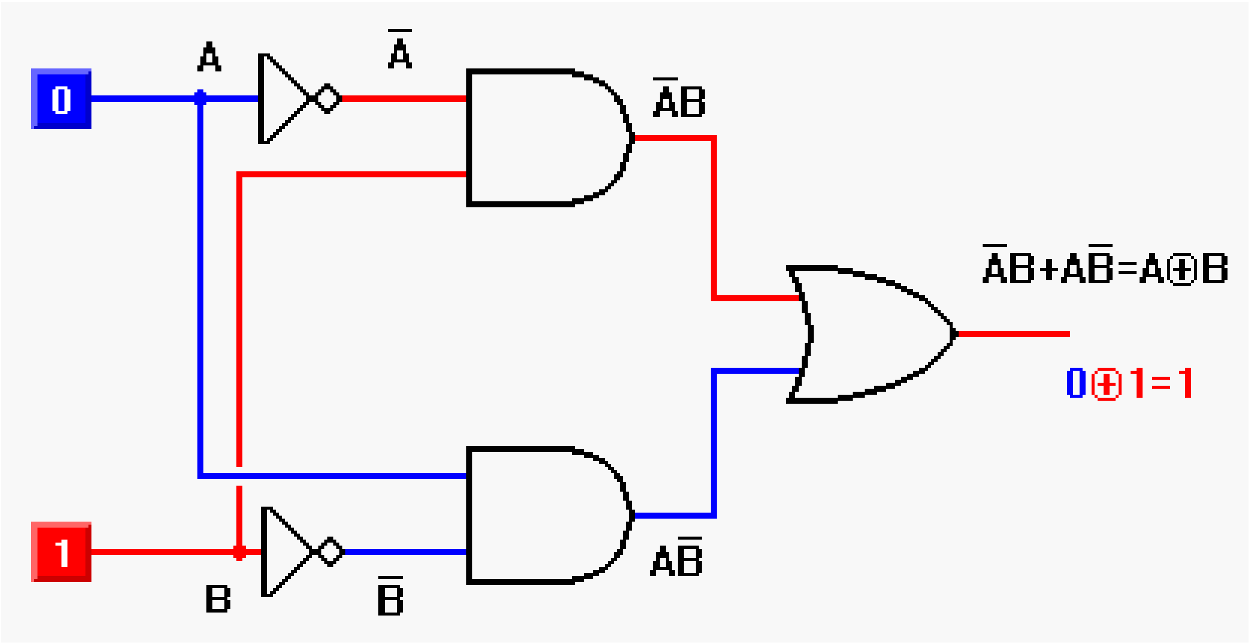
Inversion (NOT)



Logic: $Q = \overline{A}$

A	Q
0	1
1	0

Circuit for XOR

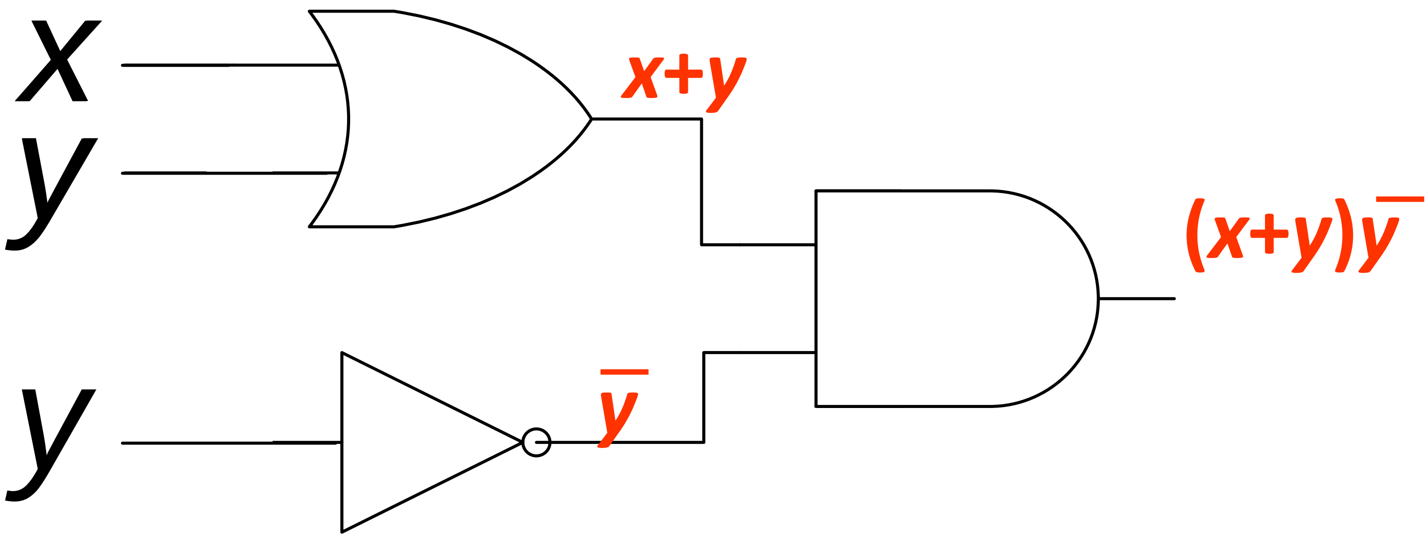


$$A \oplus B = \overline{A} \cdot B + A \cdot \overline{B}$$

Accumulating our results: Binary addition is the result of XOR plus AND

Converting between circuits and equations

- Find the output of the following circuit

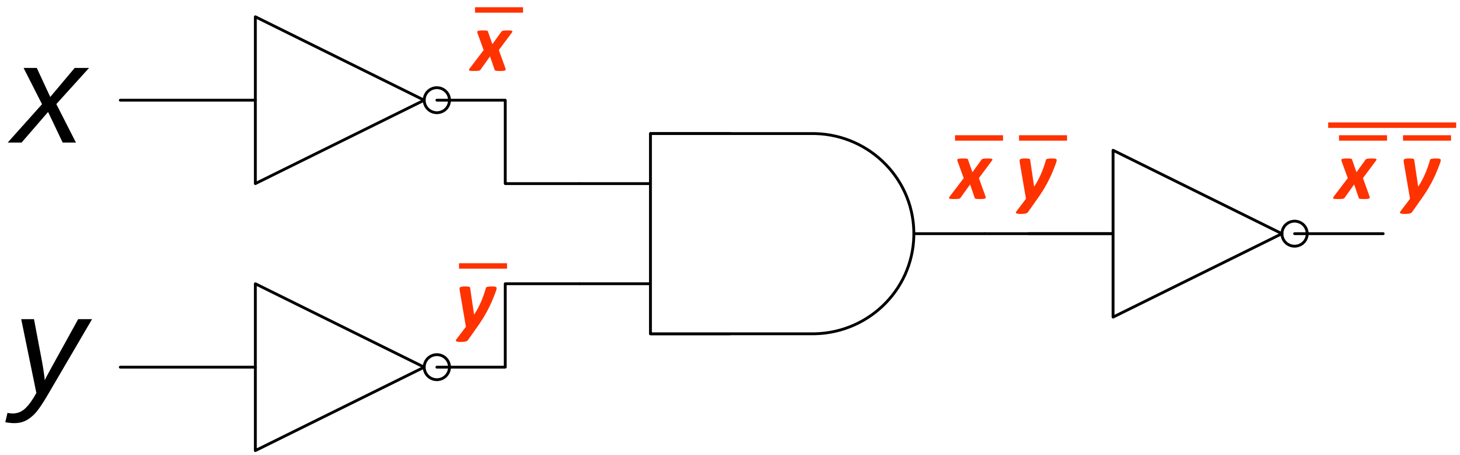


- Answer: $(x+y)\overline{y}$

Or $(x \vee y) \wedge \neg y$

Converting between circuits and equations

- Find the output of the following circuit

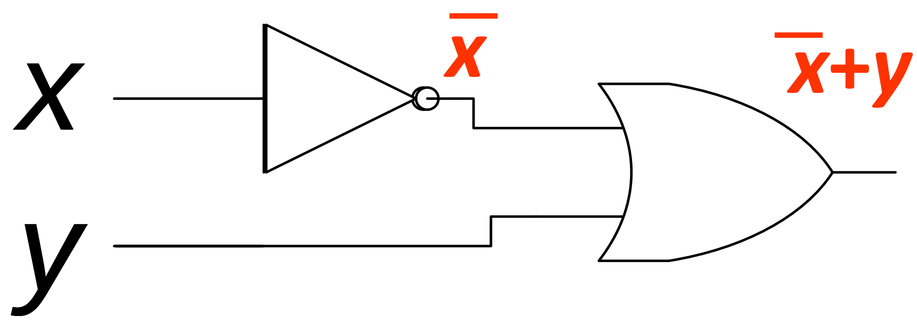


- Answer: $\overline{\bar{x}\bar{y}}$
  - Or $\neg(\neg x \wedge \neg y) \equiv x \vee y$

Converting between circuits and equations

- Write the circuits for the following Boolean algebraic expressions

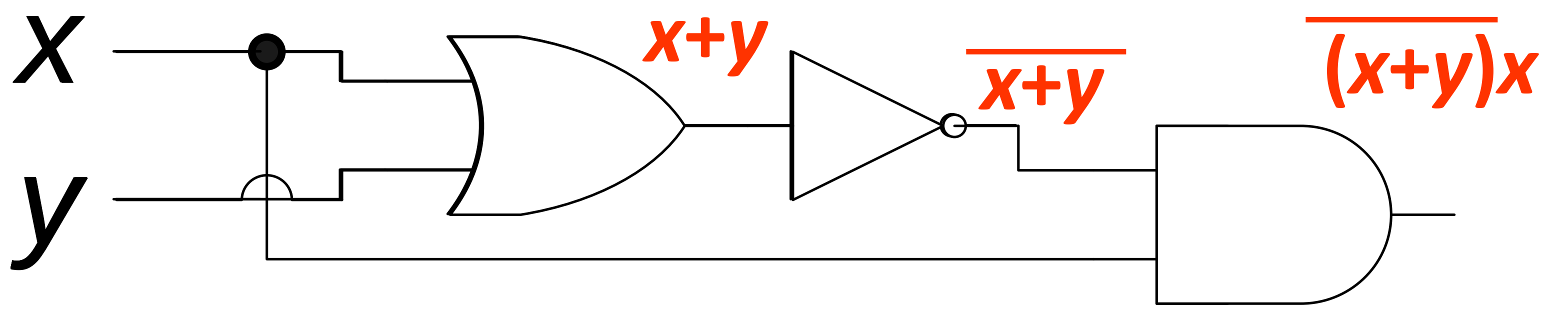
a) $\overline{x}+y$



Converting between circuits and equations

- Write the circuits for the following Boolean algebraic expressions

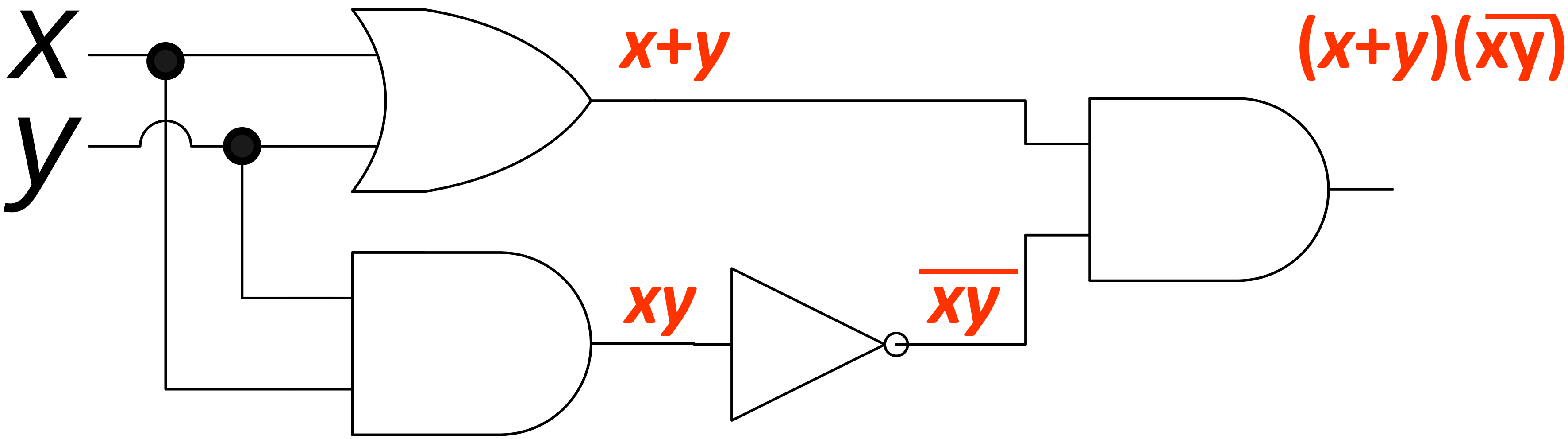
b) $\overline{(x+y)}x$



Writing xor using and/or/not

- $p \oplus q \equiv (p \vee q) \wedge \neg(p \wedge q)$
- $x \oplus y \equiv (x + y)(\overline{xy})$

x	y	$x \oplus y$
1	1	0
1	0	1
0	1	1
0	0	0



Converting decimal numbers to binary

- 53 = 32 + 16 + 4 + 1

  $= 2^5 + 2^4 + 2^2 + 2^0$

  $= 1*2^5 + 1*2^4 + 0*2^3 + 1*2^2 + 0*2^1 + 1*2^0$

  = 110101 in binary

  = 00110101 as a full byte in binary

- 211= 128 + 64 + 16 + 2 + 1

  $= 2^7 + 2^6 + 2^4 + 2^1 + 2^0$

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

  = 11010011 in binary

Converting binary numbers to decimal

- What is 10011010 in decimal?

10011010 $= 1*2^7 + 0*2^6 + 0*2^5 + 1*2^4 + 1*2^3 +$
$0*2^2 + 1*2^1 + 0*2^0$
$= 2^7 + 2^4 + 2^3 + 2^1$
$= 128 + 16 + 8 + 2$
$= 154$

- What is 00101001 in decimal?

00101001 $= 0*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 1*2^3 +$
$0*2^2 + 0*2^1 + 1*2^0$
$= 2^5 + 2^3 + 2^0$
$= 32 + 8 + 1$
$= 41$

A note on binary numbers

- In this slide set we are only dealing with non-negative numbers
- The book (section 1.5) talks about two's-complement binary numbers
  - Positive (and zero) two's-complement binary numbers is what was presented here
  - We won't be getting into negative two's-complmeent numbers

How to add binary numbers

- Consider adding two 1-bit binary numbers *x* and *y*
  - 0+0 = 0
  - 0+1 = 1
  - 1+0 = 1
  - 1+1 = 10

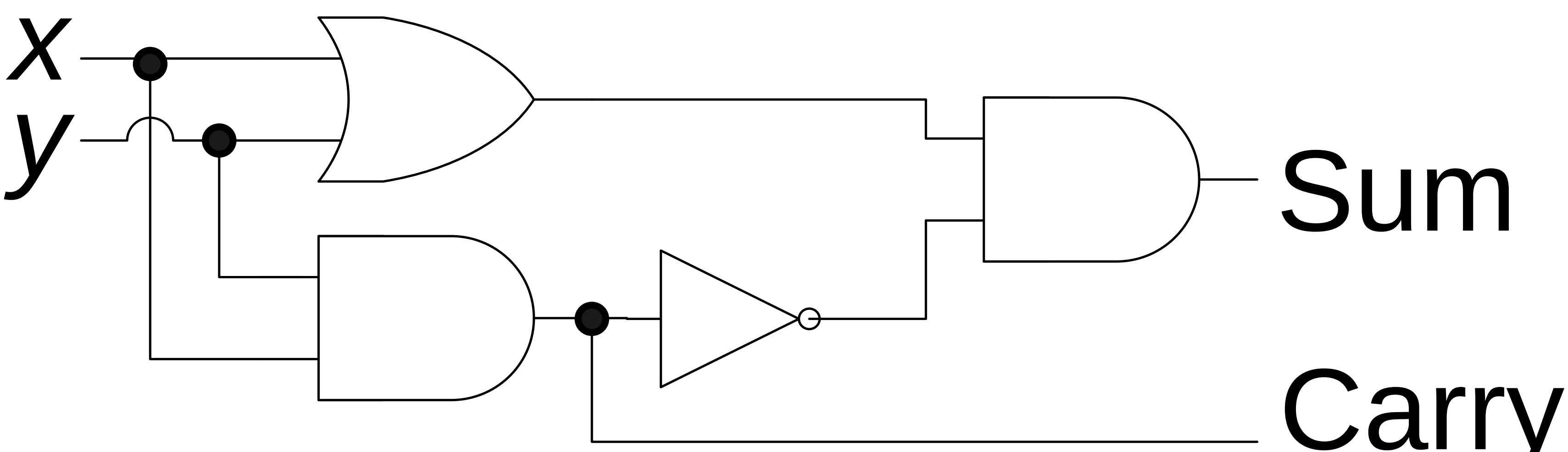
x	*y*	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Carry is *x* AND *y*
- Sum is *x* XOR *y*
- The circuit to compute this is called a half-adder

The half-adder

- Sum = *x* XOR *y*
- Carry = *x* AND *y*

x	*y*	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Using half adders

- We can then use a half-adder to compute the sum of two Boolean numbers

$$
\begin{array}{r}
\color{blue}{1\ 0\ 0} \\
1\ 1\ 0\ 0 \\
+\,1\ 1\ 1\ 0 \\
\hline
?\ 0\ 1\ 0
\end{array}
$$

How to fix this

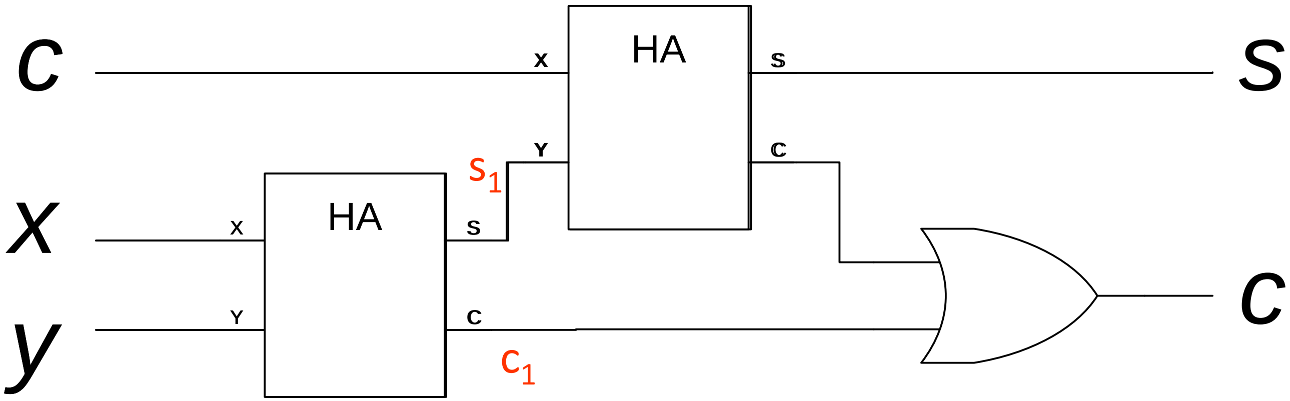
- We need to create an adder that can take a carry bit as an additional input
  - Inputs: *x*, *y*, carry in
  - Outputs: sum, carry out
- This is called a full adder
  - Will add *x* and *y* with a half-adder
  - Will add the sum of that to the carry in
- What about the carry out?
  - It's 1 if either (or both):
  - *x*+*y* = 10
  - *x*+*y* = 01 and carry in = 1

x	*y*	*c*	carry	sum
1	1	1	1	1
1	1	0	1	0
1	0	1	1	0
1	0	0	0	1
0	1	1	1	0
0	1	0	0	1
0	0	1	0	1
0	0	0	0	0

The full adder

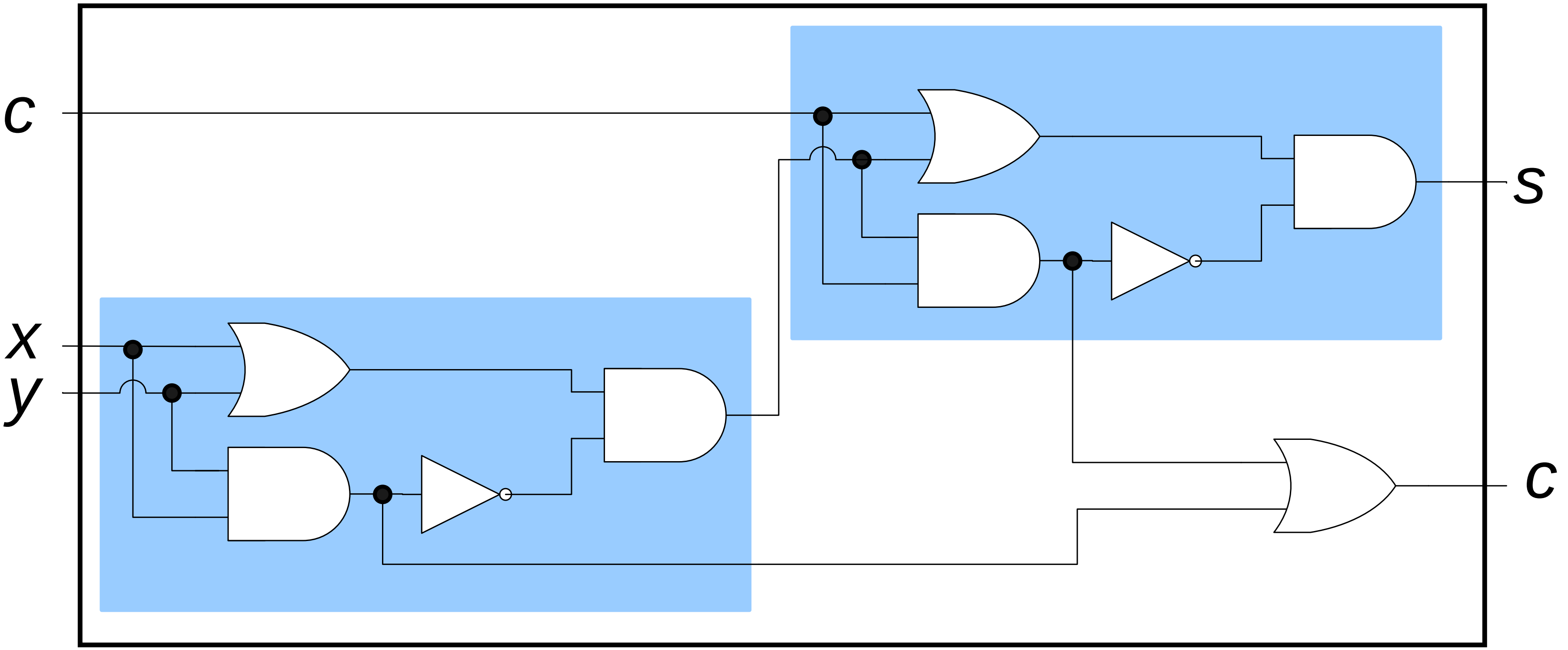
- The "HA" boxes are half-adders

x	y	c	s_1	c_1	carry	sum
1	1	1	0	1	1	1
1	1	0	0	1	1	0
1	0	1	1	0	1	0
1	0	0	1	0	0	1
0	1	1	1	0	1	0
0	1	0	1	0	0	1
0	0	1	0	0	0	1
0	0	0	0	0	0	0



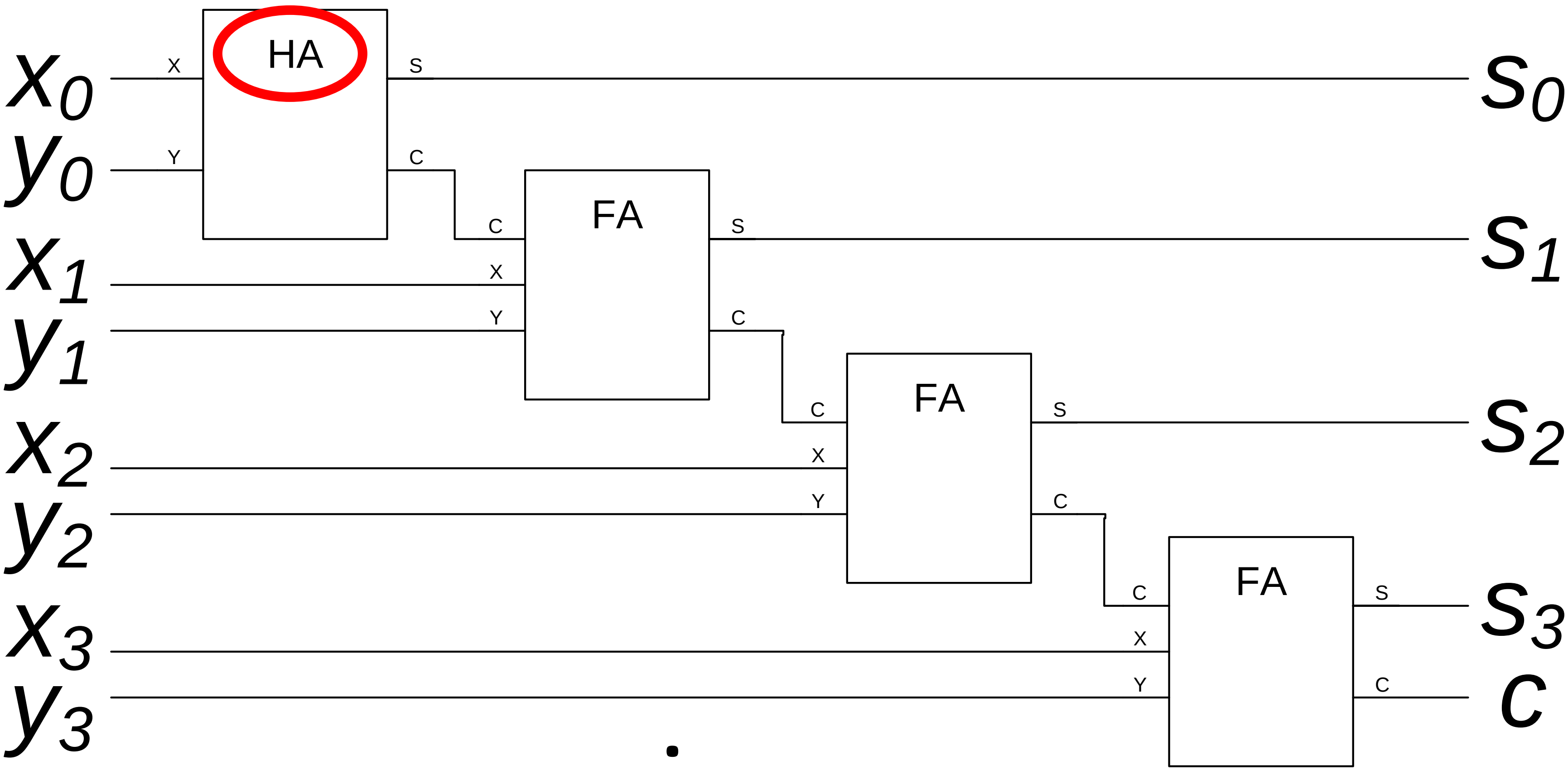
The full adder

- The full circuitry of the full adder



Adding bigger binary numbers

- Just chain full adders together



Adding bigger binary numbers

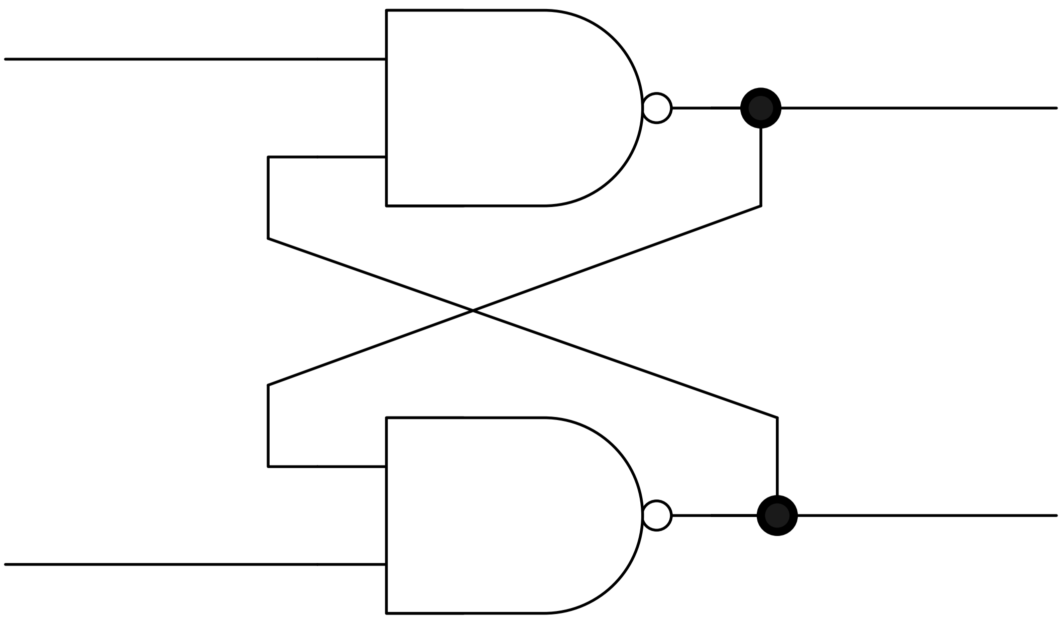
- A half adder has 4 logic gates
- A full adder has two half adders plus a OR gate
  - Total of 9 logic gates
- To add *n* bit binary numbers, you need 1 HA and *n*-1 FAs
- To add 32 bit binary numbers, you need 1 HA and 31 FAs
  - Total of 4+9*31 = 283 logic gates
- To add 64 bit binary numbers, you need 1 HA and 63 FAs
  - Total of 4+9*63 = 571 logic gates

More about logic gates

- To implement a logic gate in hardware, you use a transistor
- Transistors are all enclosed in an “IC”, or integrated circuit
- The current Intel Pentium IV processors have 55 million transistors!

Flip-flops

- Consider the following circuit:



- What does it do?

Memory

- A flip-flop holds a single bit of memory
  - The bit “flip-flops” between the two NAND gates
- In reality, flip-flops are a bit more complicated
  - Have 5 (or so) logic gates (transistors) per flip-flop
- Consider a 1 Gb memory chip
  - 1 Gb = 8,589,934,592 bits of memory
  - That’s about 43 million transistors!
- In reality, those transistors are split into 9 ICs of about 5 million transistors each

Hexadecimal

- A numerical range from 0-15
  - Where A is 10, B is 11, … and F is 15
- Often written with a ‘0x’ prefix
- So 0x10 is 10 hex, or 16
  - 0x100 is 100 hex, or 256
- Binary numbers easily translate:

Table 1.5.3

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