Logic Gates

Epp, setions 1.4 and 1.5

Review of Boolean algebra

- Just like Boolean logic
- Variables can only be 1 or 0
  - Instead of true / false

Review of Boolean algebra

- Not is a horizontal bar above the number
  - $\bar{0} = 1$
  - $\bar{1} = 0$
- Or is a plus
  - 0+0 = 0
  - 0+1 = 1
  - 1+0 = 1
  - 1+1 = 1
- And is multiplication
  - 0*0 = 0
  - 0*1 = 0
  - 1*0 = 0
  - 1*1 = 1

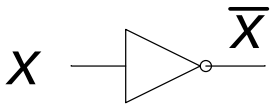
Review of Boolean algebra

- Example: translate $(x+y+z)(x\bar{y}\bar{z})'$ to a Boolean logic expression
  - $(x \vee y \vee z) \wedge (\neg x \wedge \neg y \wedge \neg z)$
- We can define a Boolean function:
  - F(x,y) = $(x \vee y) \wedge (\neg x \wedge \neg y)$
- And then write a “truth table” for it:

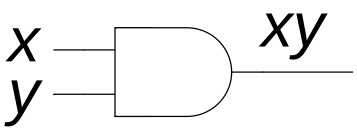
x	*y*	*F(x,y)*
1	1	0
1	0	0
0	1	0
0	0	0

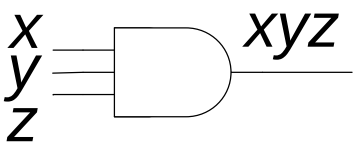
Basic logic gates

- Not

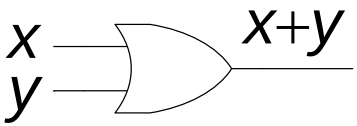


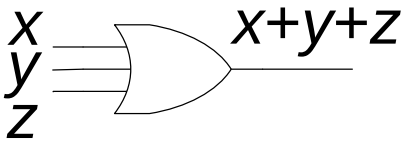
- And



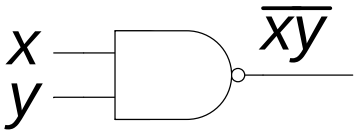


- Or

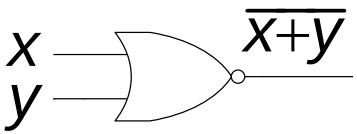




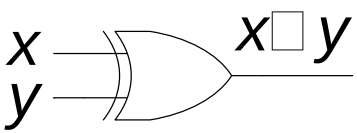
- Nand



- Nor

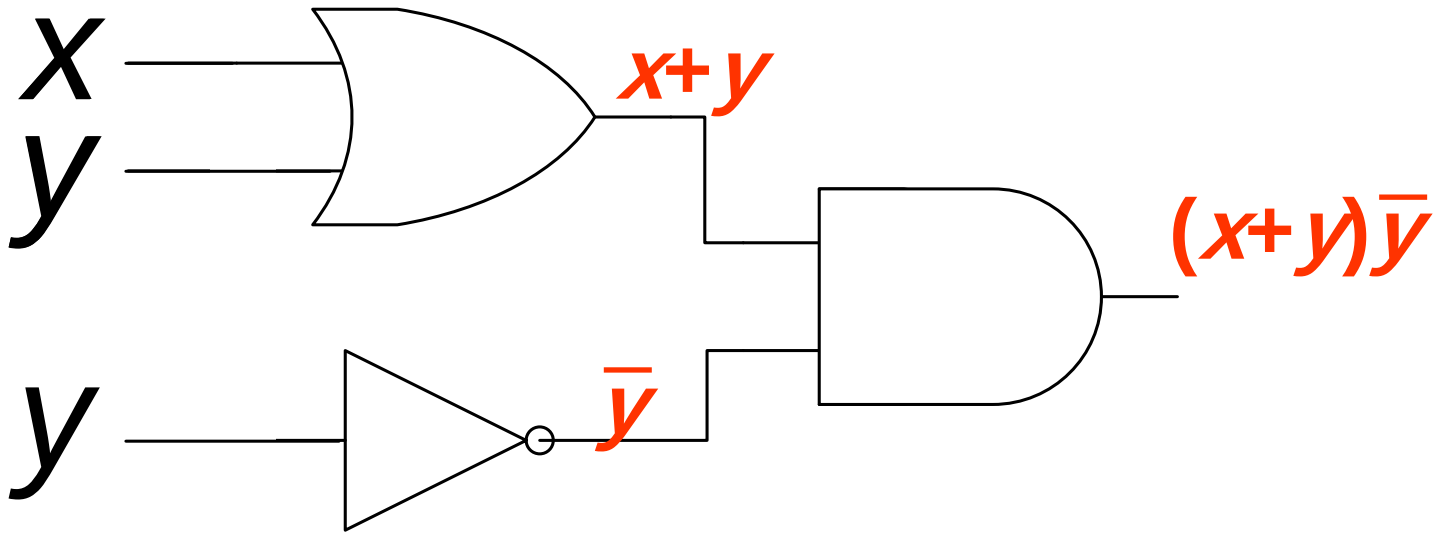


- Xor



Converting between circuits and equations

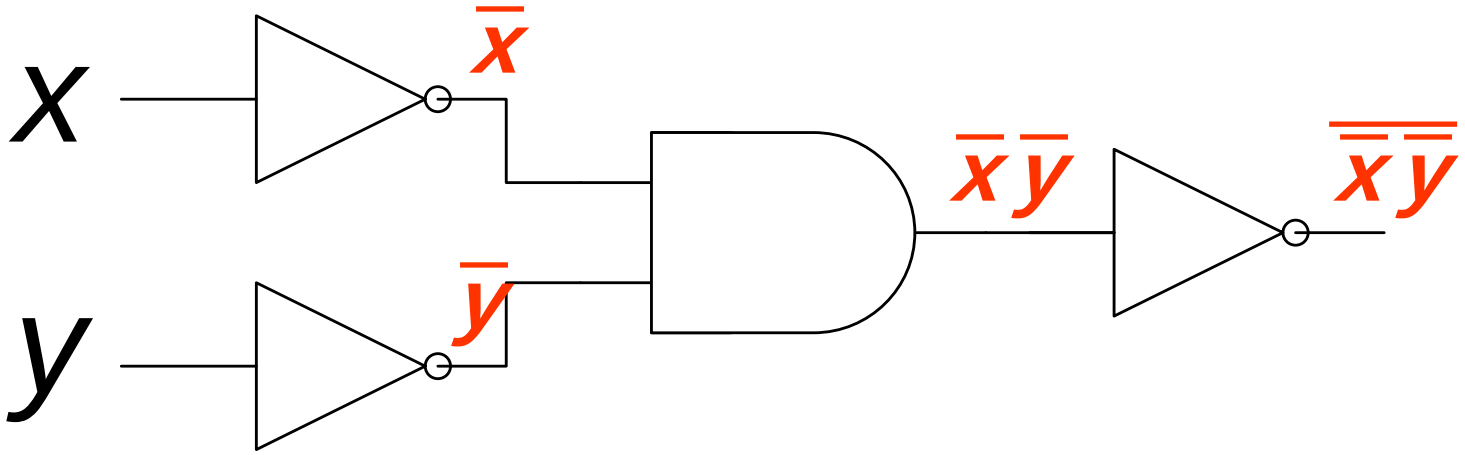
- Find the output of the following circuit



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

Converting between circuits and equations

- Find the output of the following circuit

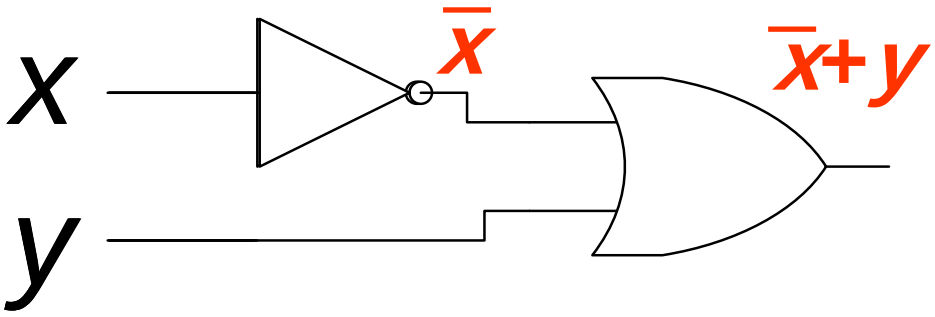


- Answer: $\overline{\bar{x}\bar{y}}$
  - Or $\neg(\neg x \wedge \neg y) \equiv x \vee y$

Converting between circuits and equations

- Write the circuits for the following Boolean algebraic expressions

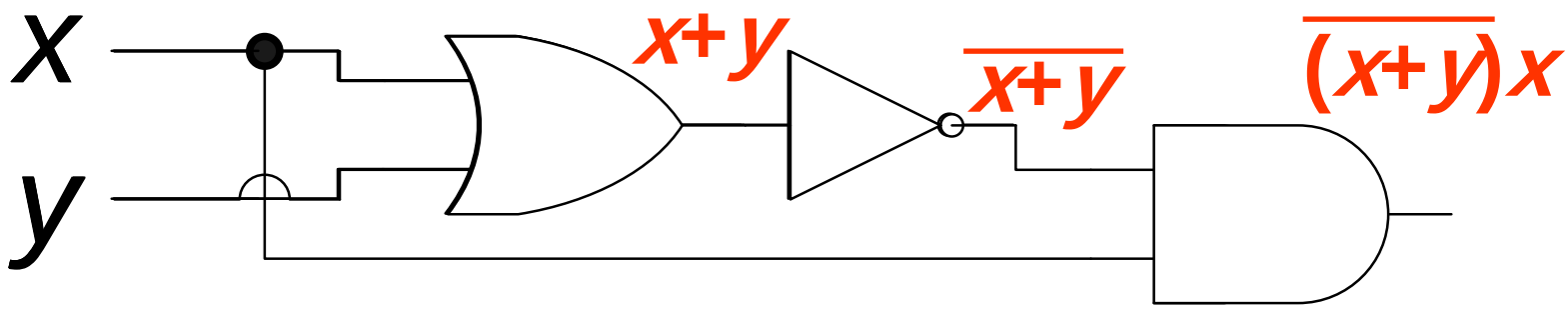
a) $\overline{x}+y$



Converting between circuits and equations

- Write the circuits for the following Boolean algebraic expressions

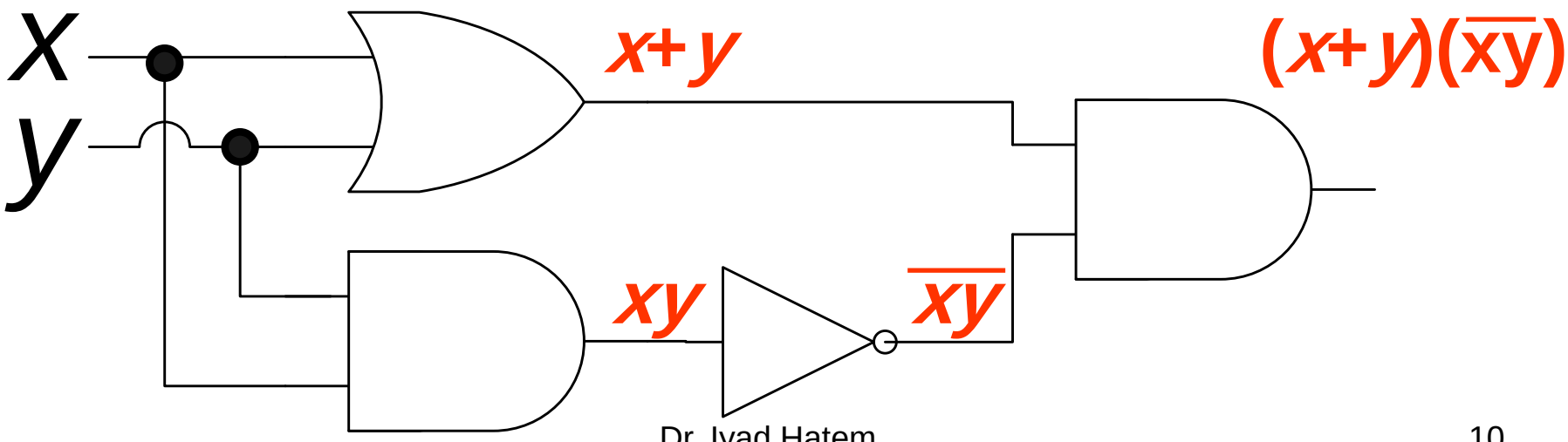
b) $\overline{(x+y)}x$



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

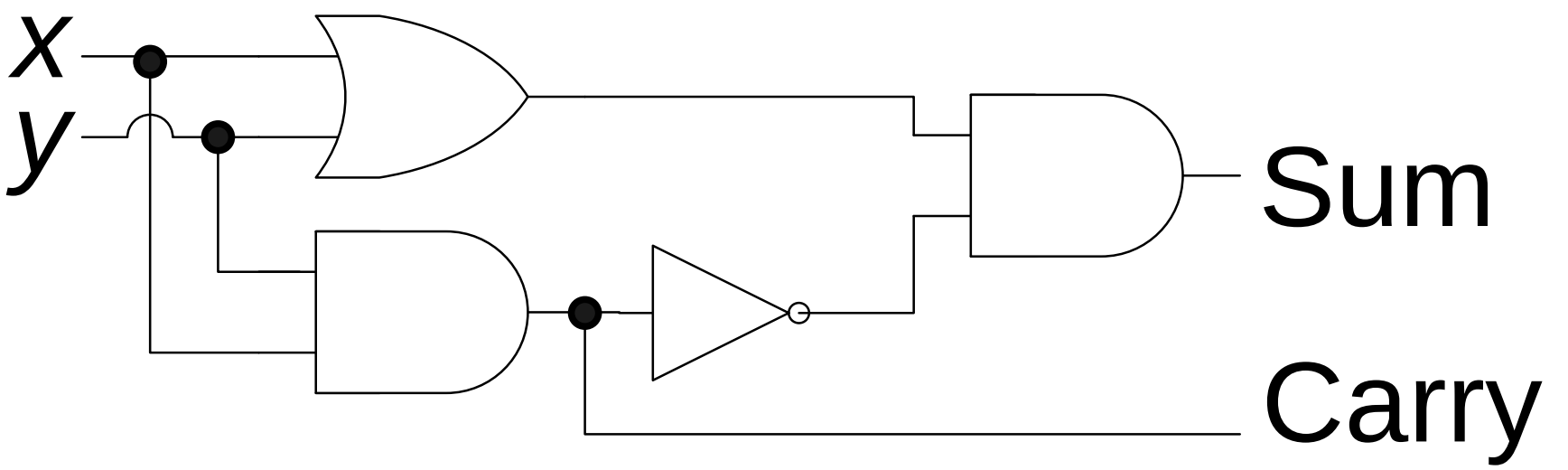
x	*y*	Carry	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

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

The half-adder

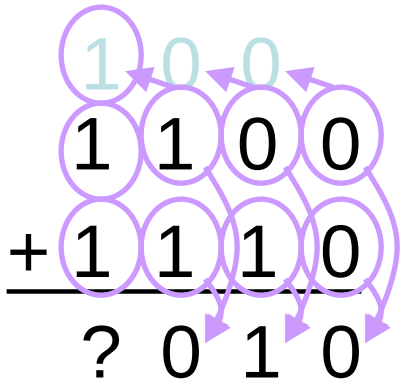
- Sum = *x* XOR *y*
- Carry = *x* AND *y*

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



How to fix this

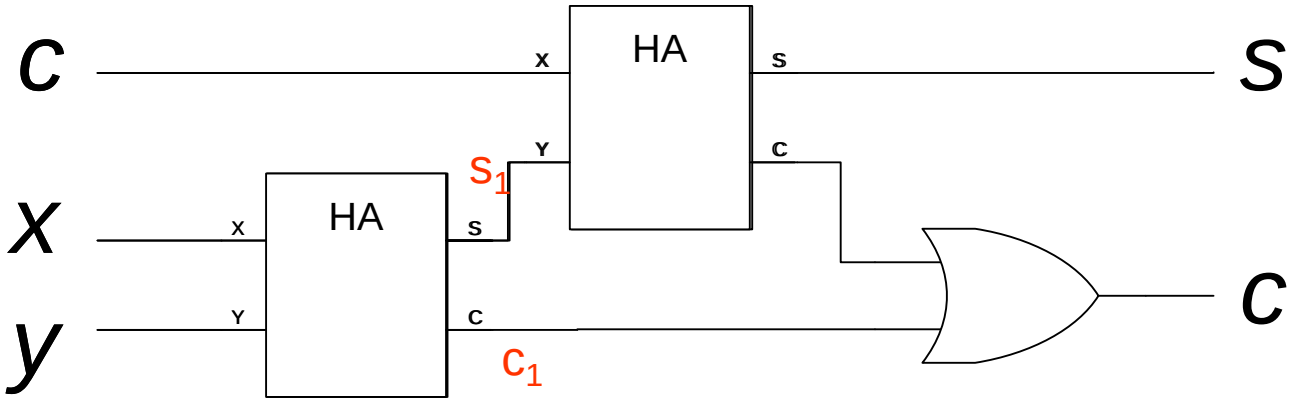
- We need to create an adder that can take a carry bit as an additional input
  - Inputs: *x*, *y*, carry in
  - Outputs: sum, carry out
- This is called a full adder
  - Will add *x* and *y* with a half-adder
  - Will add the sum of that to the carry in
- What about the carry out?
  - It's 1 if either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

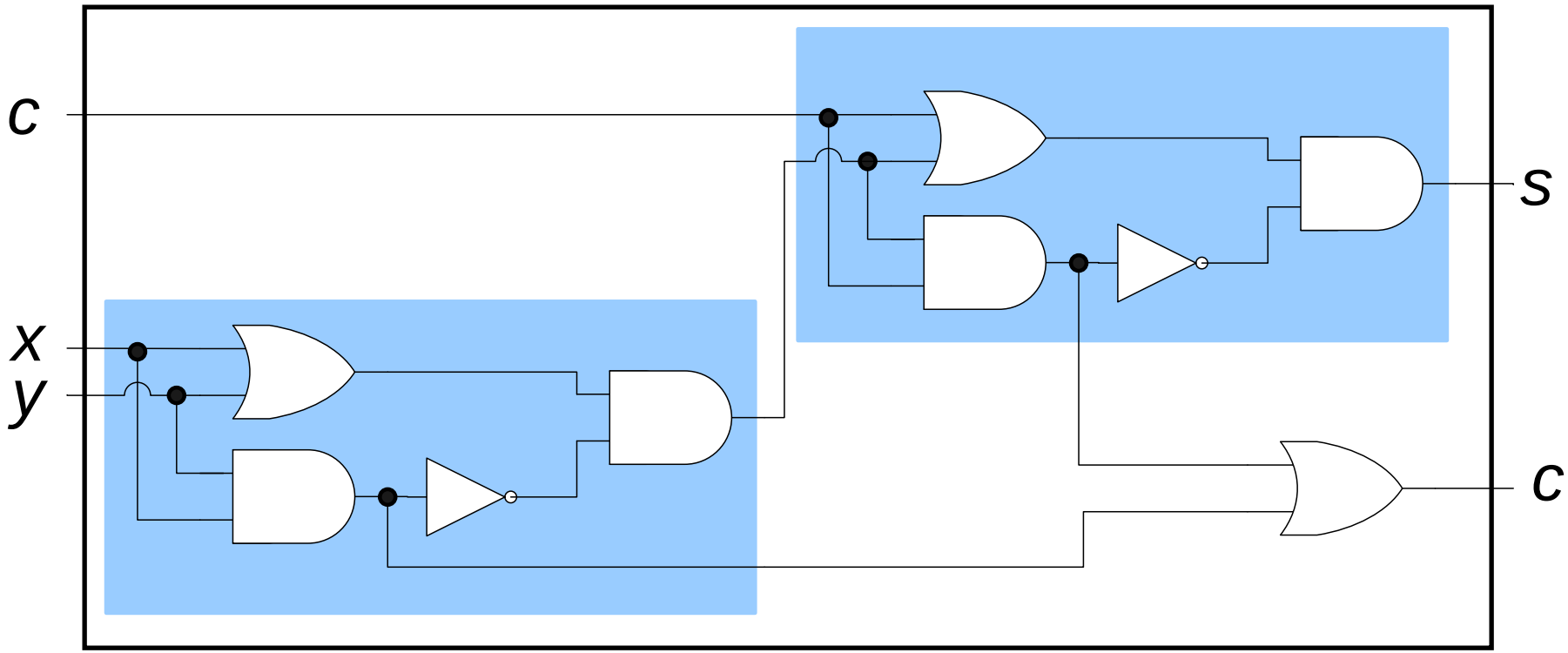
- The “HA” boxes are half-adders

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



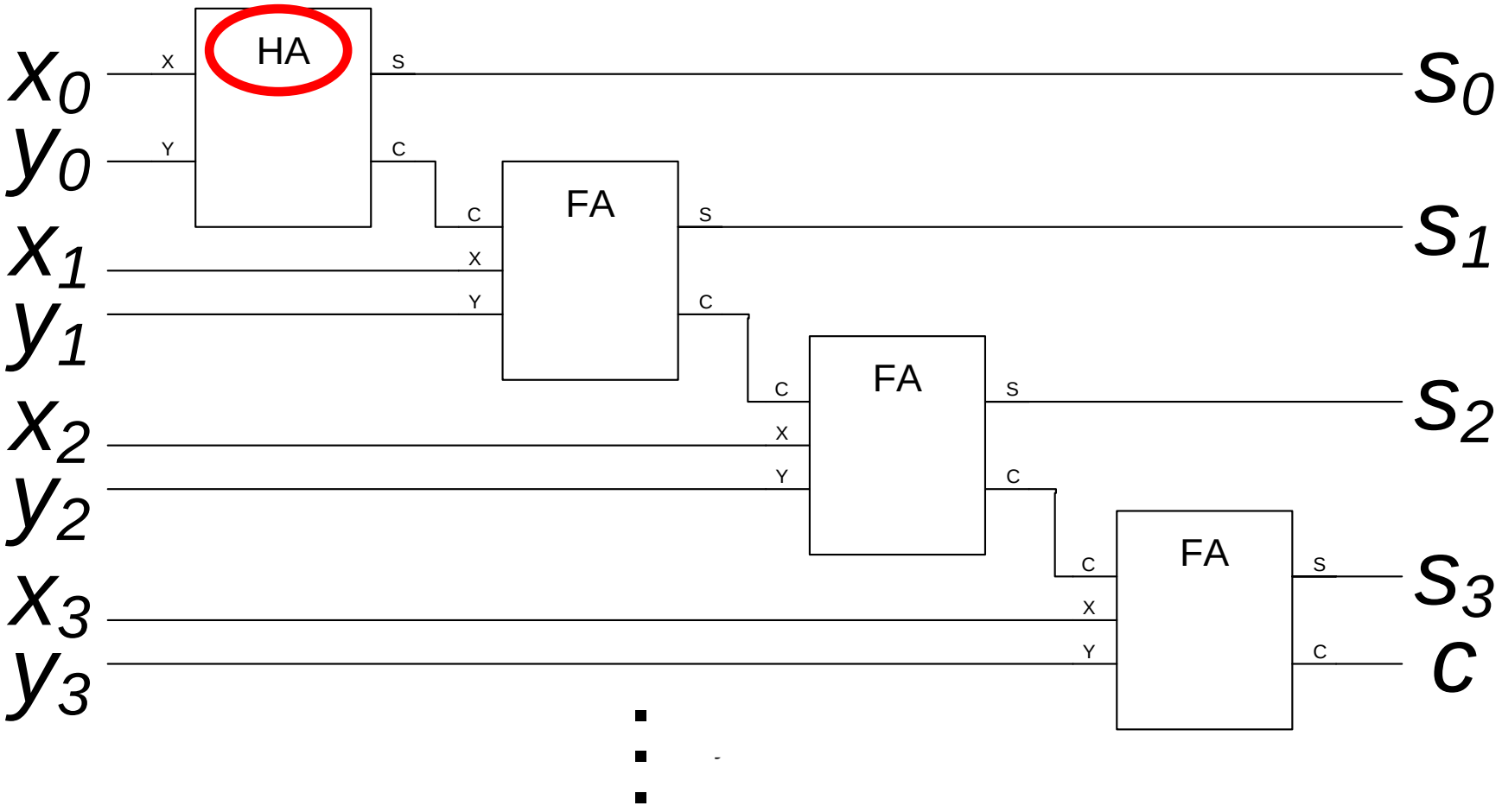
The full adder

- The full circuitry of the full adder



Adding bigger binary numbers

- Just chain full adders together



Adding bigger binary numbers

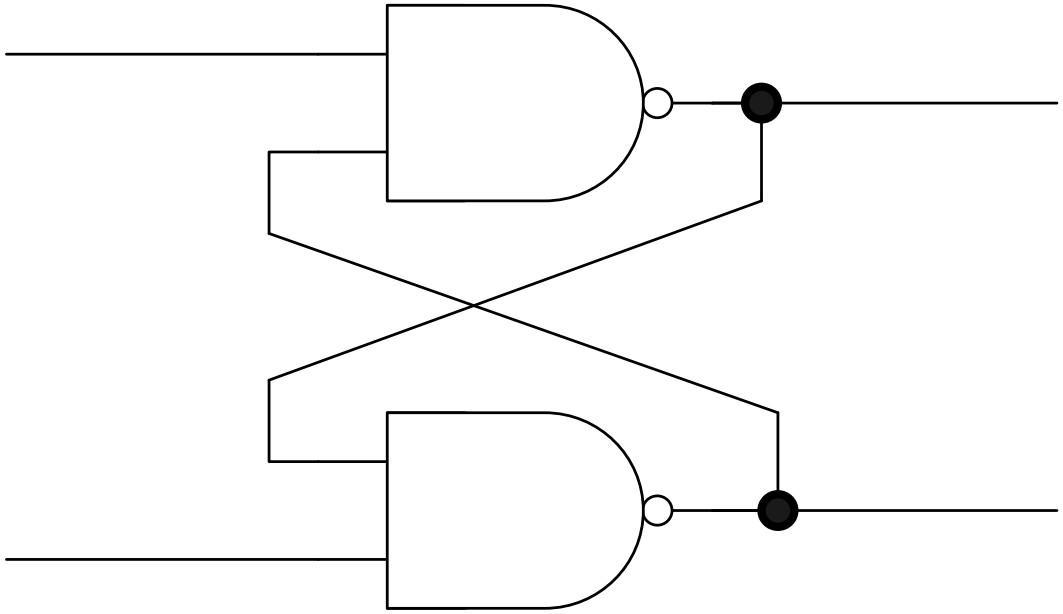
- A half adder has 4 logic gates
- A full adder has two half adders plus a OR gate
  - Total of 9 logic gates
- To add *n* bit binary numbers, you need 1 HA and *n*-1 FAs
- To add 32 bit binary numbers, you need 1 HA and 31 FAs
  - Total of 4+9*31 = 283 logic gates
- To add 64 bit binary numbers, you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

Hexadecim

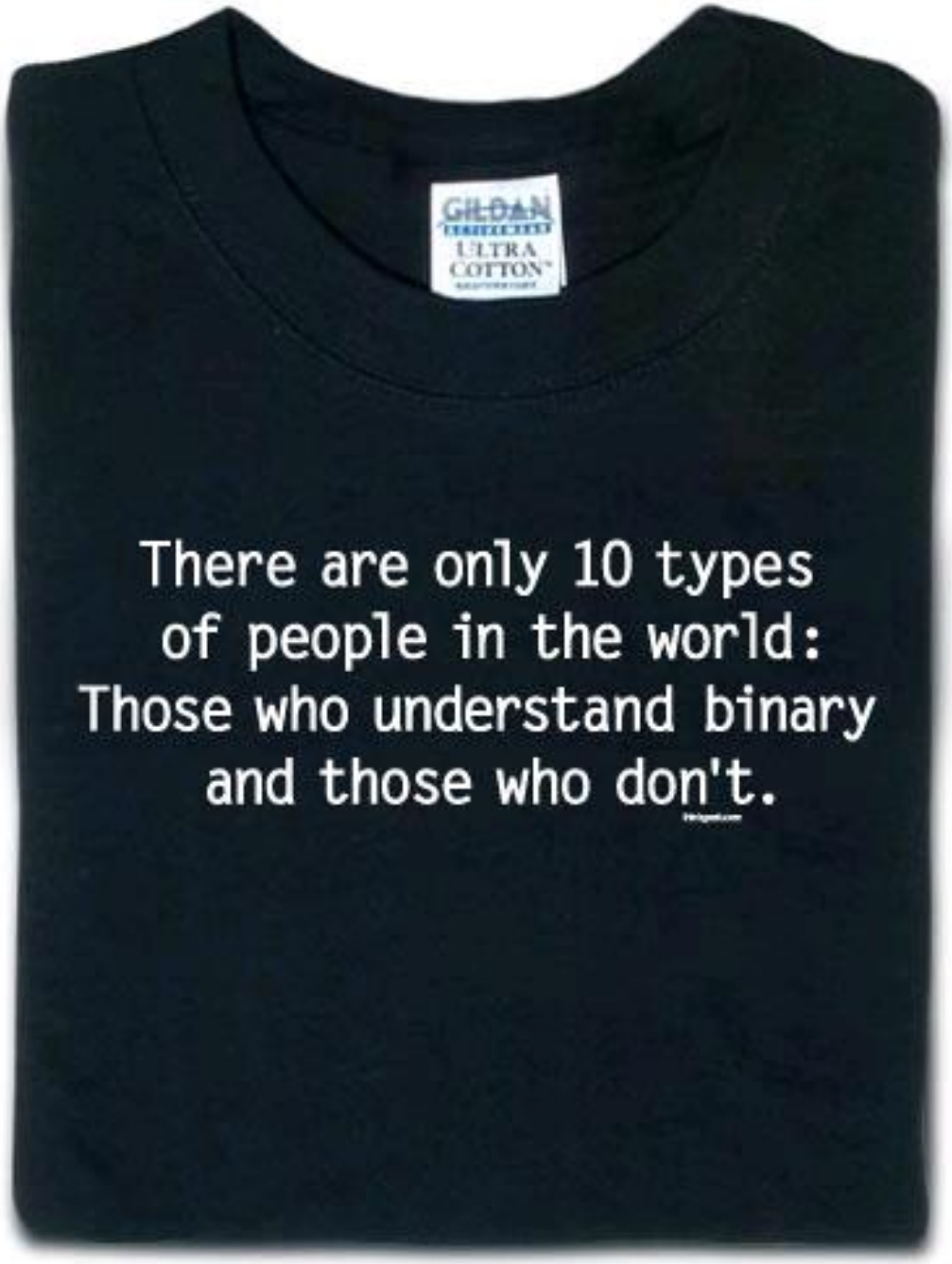
- A numerical range from 0-15
  - Where A is 10, B is 11, … and F is 15
- Often written with a ‘0x’ prefix
- So 0x10 is 10 hex, or 16
  - 0x100 is 100 hex, or 256
- Binary numbers easily translate:

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

From ThinkGeek

(http://www.thinkgeek.com



Also from ThinkGeek

(http://www.thinkgeek.com)

DEADBEEF

- Many IBM machines would fill allocated (but uninitialized) memory with the hexa-decimal pattern 0xDEADBEEF
  - Decimal -21524111
  - See http://www.jargon.net/jargonfile/d/DEADBEEF.html

- Makes it easier to spot in a debugger

Also also from ThinkGeek

(http://www.thinkgeek.com)

- 0xDEAD = 57005
- Now add one to that...