

The Building Blocks: Binary Numbers, Boolean Logic, and Gates

Chapter 4

Purpose of Chapter

- Learn how computers represent and store information.
- Learn why computers represent information that way.
- Learn what the basic building devices in a computer are, and how those devices are used to store information.
- Learn how to build more complex devices using the basic devices.

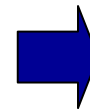
External Representation of Information

- When we communicate with each other, we need to represent the information in an understandable notation, e.g.
 - We use digits to represent numbers.
 - We use letters to represent text.
- Same applies when we communicate with a computer:
 - We enter text and numbers on the keyboard,
 - The computers displays text, images, and numbers on the screen.
- We refer to this as an external representation.
 - But how do humans/computers store the information “internally”?

External Representation of Information

- Humans:

Text, numbers,
images, sounds



Text, numbers,
sounds

- Computers:

Text, numbers,
images, sounds



Text, numbers,
images, sounds

Binary Numbers

Information we need to represent

- Numbers
 - Integers (234, 456)
 - Positive/negative value (-100, -23)
 - Floating point numbers (12.345, 3.14159)
- Text
 - Characters (letters, digits, symbols)
- Other
 - Graphics, Sound, Video, ...

Numbering Systems

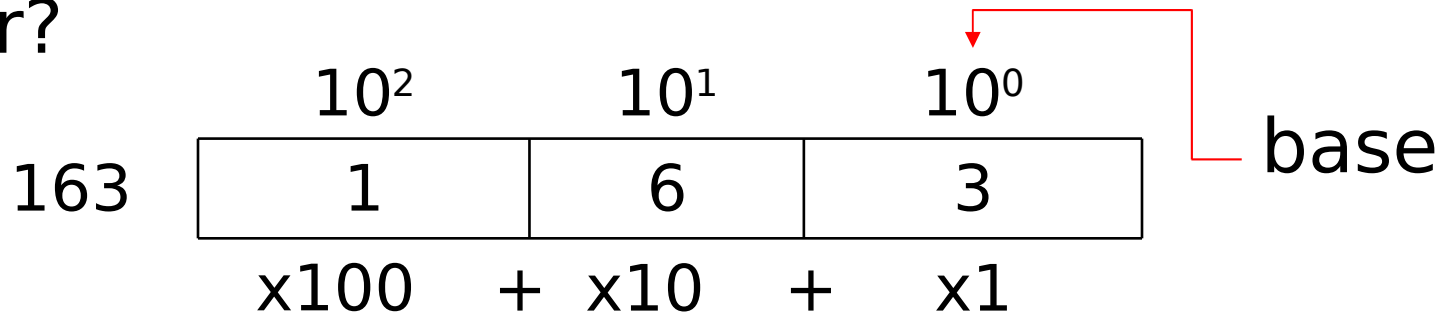
- We use the decimal numbering system
 - 10 digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
 - For example: 12
- Why use 10 digits (symbols)?
 - Roman: I (=1) V (=5) X (=10) L (=50), C(=100)
 - XII = 12, Pentium IV
- What if we only had one symbol?
 - IIIII IIIII II = 12

The Binary Numbering System

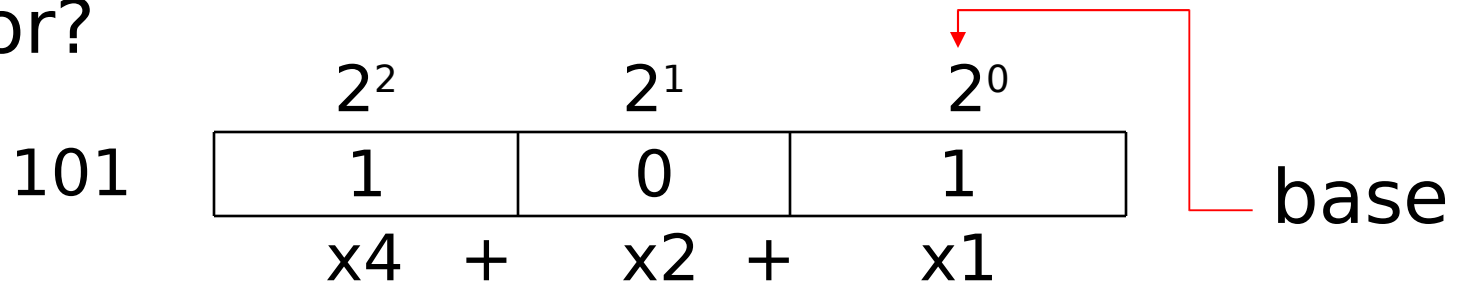
- All computers use the binary numbering system
 - Only two digits: 0, 1
 - For example: 10, 10001, 10110
- Similar to decimal, except uses a different base
 - Binary (base-2): 0, 1
 - Decimal (base-10): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
 - Octal (base-8): 0, 1, 2, 3, 4, 5, 6, 7
 - Hexadecimal (base-16): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F (A=10, ..., F=15)
- What do we mean by a base?

Decimal vs. Binary Numbers

- What does the decimal value 163 stand for?



- What does the binary value 101 stand for?



Binary-to-Decimal Conversion Table

Decimal	Binary	Decimal	Binary	Decimal	Binary	Decimal	Binary
0	0	8	1000	16	10000	24	11000
1	1	9	1001	17	10001	25	11001
2	10	10	1010	18	10010	26	11010
3	11	11	1011	19	10011	27	11011
4	100	12	1100	20	10100	28	11100
5	101	13	1101	21	10101	29	11101
6	110	14	1110	22	10110	30	11110
7	111	15	1111	23	10111	31	11111

Converting from Binary to Decimal

- What is the decimal value of the binary value 101 ?

	2^2		2^1		2^0	
101	1		0		1	
	x4	+	x2	+	x1	
	4	+	0	+	1	= 5

- What is the decimal value of the binary value 1110 ?

	2^3		2^2		2^1		2^0	
1110	1		1		1		0	
	x8	+	x4	+	x2	+	x1	
	8	+	4	+	2	+	0	= 14

Bits

- The two binary digits 0 and 1 are frequently referred to as bits.
- How many bits does a computer use to store an integer?
 - Intel Pentium PC = 32 bits
- What if we try to compute a larger integer?
 - If we try to compute a value larger than the computer can store, we get an arithmetic overflow error.

Representing Unsigned Integers

- How does a 16-bit computer represent the value 14?

0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

- What is the largest 16-bit integer?

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

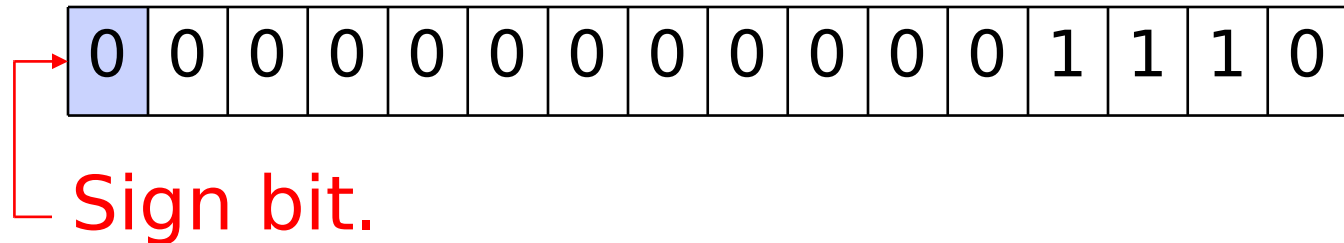
$$= 1 \times 2^{15} + 1 \times 2^{14} + \dots + 1 \times 2^1 + 1 \times 2^0 = 65,535$$

Representing Signed Integers

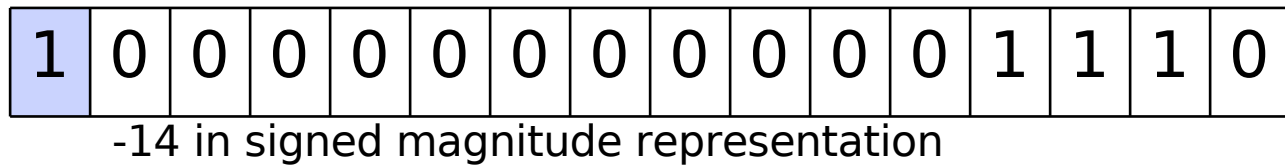
- How does the computer represent signed values?
 - Three schemes have been used:
 - Signed magnitude
 - 1's complement
 - 2's complement
 - While unsigned binary is in some sense “natural” signed binary arithmetic is a human construct
 - There *is* a best way - which is what we all use today

Representing Signed Integers

- Signed Magnitude
 - First bit is the *sign* bit: 0=> +ve 1=> -ve
 - Remainder is the *magnitude* (i.e. numeric value)



+14 in signed magnitude representation



- Problems
 - Arithmetic logic is complex
 - There are two zeros:
 - 0000000000000000
 - 1000000000000000

Representing Signed Integers

- 1's Complement
 - There is no *sign* bit but an *algorithm*:
 - To convert from +ve to -ve and vice versa:
 - invert ('flip') all the bits

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

+14 in 1's complement representation

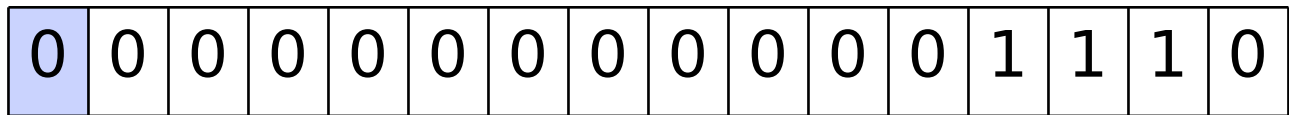
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

-14 in 1's complement representation

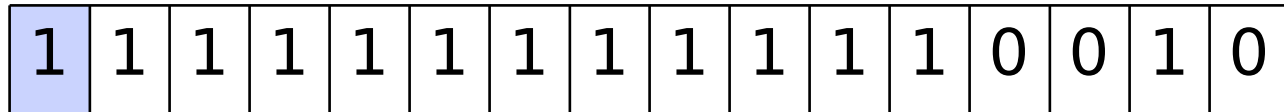
- Problems
 - Although arithmetic is much simpler, there are two zeros:
 - 0000000000000000
 - 1111111111111111

Representing Signed Integers

- 2's Complement
 - There is no *sign* bit but an *algorithm*:
 - To convert from +ve to -ve and vice versa:
 - invert ('flip') all the bits
 - then add 1



+14 in 2's complement representation



-14 in 2's complement representation

- Simple arithmetic (like 1's complement)
- Only one zero:

```
0000000000000000    "+0", convert:
1111111111111111
+ 1
-----
0000000000000000
```

Representing Floating Point Numbers

- How do we represent floating point numbers like 5.75 and -143.50?
- Three step process:
 - Convert the decimal number to a binary number.
 - Write binary number in “normalized” scientific notation.
 - Store the normalized binary number.
- Look at an example:
 - How do we store the number 5.75?

1. Convert decimal to binary

...	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	...
	8	4	2	1	$\frac{1}{2}$	$\frac{1}{4}$	

$$4 + 1 + \frac{1}{2} + \frac{1}{4} = 5.75$$

	0	1	0	1	1	1	
--	---	---	---	---	---	---	--

5.75 decimal = 101.11 binary

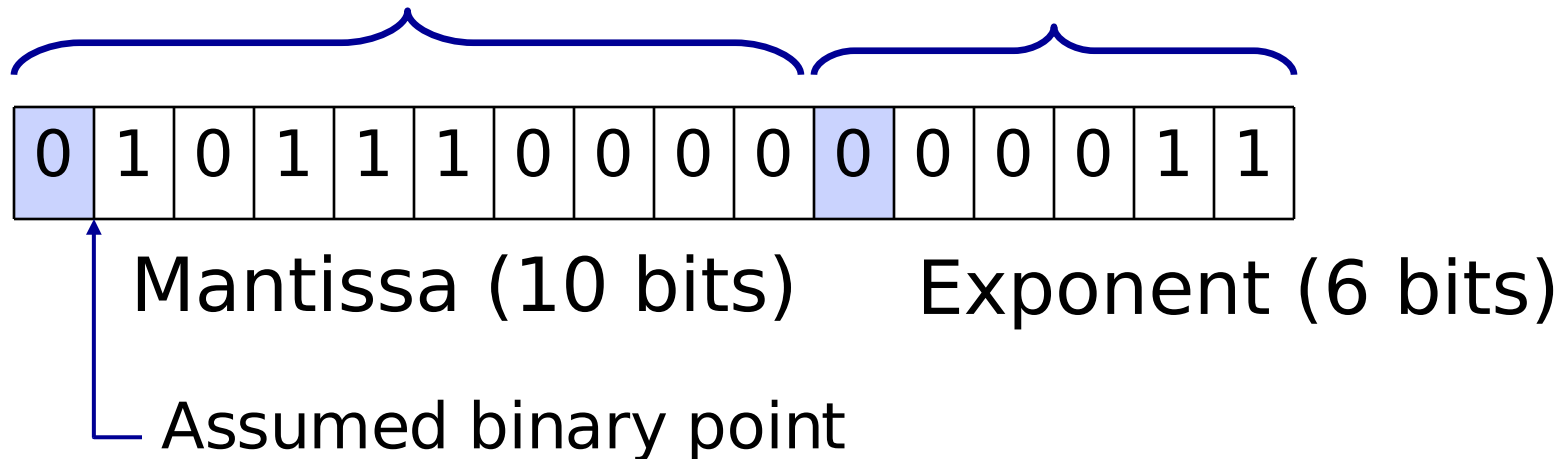
2. Using normalized scientific notation

- Scientific notation: $\pm M \times B^{\pm E}$
 - B is the base, M is the mantissa , E is the exponent.
 - Example: (decimal, base=10)
 - $3 = 3 \times 10^0$ (e.g. $3 * 1$)
 - $2050 = 2.05 \times 10^3$ (e.g. $2.05 * 1000$)
- Easy to convert to scientific notation:
 - 101.11×2^0
- Normalize to get the “.” in front of first (leftmost) 1 digit
 - Increase exponent by one for each location “.” moves left (decreases if we have to move right)
 - $101.11 \times 2^0 = .10111 \times 2^3$

3. Store the normalized number

Base 2 implied, not stored

$$+ .10111 \times 2^{+3}$$



Representing Text

- How can we represent text in a binary form?
 - Assign to each character a positive integer value (for example, A is 65, B is 66, ...)
 - Then we can store the numbers in their binary form!
- The mapping of text to numbers
 - Code mapping
- Need standard code mappings (why?):
 - ASCII (American Standard Code for Information Interchange) => each letter 8-bits
 - only 256 different characters can be represented (2^8)
 - Unicode => each letter 16-bits (sometimes)

ASCII Code mapping Table

Char	Integer	Binary	Char	Integer	Binary
	32	00100000	A	65	01000001
!	33	00100001	B	66	01000010
“	34	00100010	C	67	01000011
...
0	48	00110000	x	120	01111000
1	49	00110001	y	121	01111001
2	50	00110010	z	122	01111010
...

Example of Representing Text

- Representing the word “Hello” in ASCII
 - Look the value for each character up in the table
 - (Convert decimal value to binary)

H	e	l	l	o
72	101	108	108	111
01001000	01100101	01101100	01101100	01101111

example

CSc

01000011 01010011 01100011



C

S

c

112

00110001 00110001 00110010



1

1

2

Notice the difference between ASCII '112' and binary representation of 112_{10} : 0111 0000

Hexadecimal

Decimal	Binary	Hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7

Decimal	Binary	Hex
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

$$\text{ex: } 112 = 0111\ 0000 = 70_{16}$$

$$126 = 0111\ 1110 = 7E_{16}$$

Why use Binary Numbers?

- Why not use the decimal systems, like humans?
- The main reason for using binary numbers is:
 - Reliability
- Why is that?
 - Electrical devices work best in a bistable environment, that is, there are only two separate states (e.g. on/off).
 - When using binary numbers, the computers only need to represent two digits: 0 and 1

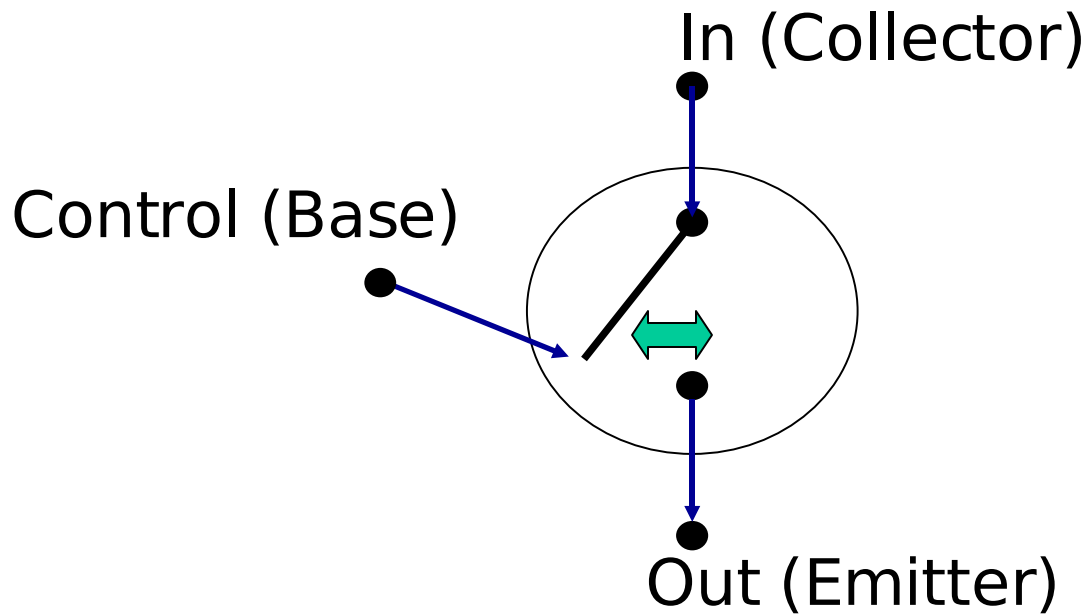
Binary Storage Devices

- We could, in theory at least, build a computer from any device:
 - That has two stable states (one would represent the digit 0, the other the digit 1)
 - Where the two states are “different” enough, such that one doesn’t accidentally become the other.
 - It is possible to sense in which state the device is in.
 - That can switch between the two states.
- We call such devices binary storage devices
 - Can you think of any?

Transistor

- The binary storage device computers use is called a transistor:
 - Can be in a stable On/Off state (current flowing through or not)
 - Can sense in which state it is in (measure electrical flow)
 - Can switch between states (takes < 10 billionths of a second!)
 - Are extremely small (can fit > 10 million/cm² , shrinking as we speak)
- Transistors are build from materials called semi-conductors
 - e.g. silicon
- The transistor is the elementary building block of computers, much in the same way as cells are the elementary building blocks of the human body!

Transistor – Conceptual Model



- The control line (base) is used to open/close switch:
 - If voltage applied then switch closes, otherwise is open
- Switch decides state of transistor:
 - Open: no current flowing through (0 state)
 - Closed: current flowing through (1 state)

Future Development?

- Transistors
 - Technology improving, allowing us to pack the transistors more and more densely (VLSI, ULSI, ...)
- Can we invent more efficient binary storage devices?
 - Past: Magnetic Cores, Vacuum Tubes
 - Present: Transistors
 - Future: ?
- Quantum Computing?

Boolean Logic and Gates

Boolean Logic

- Boolean logic is a branch of mathematics that deals with rules for manipulating the two logical truth values true and false.
- Named after George Boole (1815-1864)
 - An English mathematician, who was first to develop and describe a formal system to work with truth values.
- Why is Boolean logic so relevant to computers?
 - Direct mapping to binary digits!
 - 1 = true, 0 = false

Boolean Expressions

- A Boolean expression is any expression that evaluates to either true or false.
- Is $1+3$ a Boolean expression?
 - No, doesn't evaluate to either true or false.
- Examples of Boolean expressions:

$X > 100$

$X < Y$

$A = 100$

$2 > 3$

Boolean Operators

- We use the three following operators to construct more complex Boolean expressions

AND

OR

NOT

- Examples:

$X > 100$ AND $X < 250$

$A == 0$ OR $B > 100$

Truth Table for AND

Let a and b be any Boolean expressions, then

a	b	a AND b
False	False	False
False	True	False
True	False	False
True	True	True

Examples

X is 10 and Y is 15

X > 0 AND X < 20 True

X = 10 AND X > Y False

Truth Table for OR

Let a and b be any Boolean expressions, then

a	b	$a \text{ OR } b$
False	False	False
False	True	True
True	False	True
True	True	True

Examples

X is 10 and Y is 15

$X > 0 \text{ OR } X < 20$ True

$X = 10 \text{ OR } X > Y$ True

Truth Table for NOT

Let a be any Boolean expression, then

a	NOT a
False	True
True	False

Examples

NOT $X > 0$

NOT $X > Y$

X is 10 and Y is 15

False

True

Boolean Operators (cont.)

- Assume X is 10 and Y is 15.
- What is the value of the Boolean expression?

$X=10 \text{ OR } X=5 \text{ AND } Y<0$

$(X=10 \text{ OR } X=5) \text{ AND } Y<0$	False
$X=10 \text{ OR } (X=5 \text{ AND } Y<0)$	True

We should use parentheses to prevent confusion!

Examples of Boolean Expressions

- Assuming $X=10$, $Y=15$, and $Z=20$.
- What do the following Boolean expressions evaluate to?

$((X==10) \text{ OR } (Y==10)) \text{ AND } (Z>X)$ true

$(X==Y) \text{ OR } (\text{NOT } (X>Z))$ true

$\text{NOT } ((X>Y) \text{ AND } (Z>Y) \text{ AND } (X<Z))$ true

$((X==Y) \text{ AND } (X==10)) \text{ OR } (Y<Z)$ true

Gates

- A gate is an electronic device that operates on a collection of binary inputs to produce a binary output.
- We will look at three different kind of gates, that implement the Boolean operators:
 - AND
 - OR
 - NOT

Alternative Notation

- When we are referring to gates, we use a different notation from Boolean expressions:

a AND b a · b a && b a ∧ b

a OR B a + b a || b - a ∨ b

NOT a 'a ¬a !a

- The functionality of the operators is the same, just a different notation.

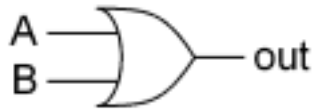
Gates

- Types of gates

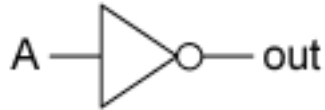
- AND



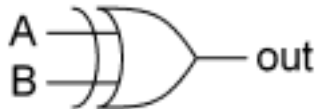
- OR



- NOT



- XOR



- NAND



- NOR



Gates vs. Transistors

- We can build the AND, OR, and NOT gates from transistors.
- Now we can think of gates, instead of transistors, as the basic building blocks:
 - Higher level of abstraction, don't have to worry about as many details.
 - Can use Boolean logic to help us build more complex circuits. (But not in this course)

Summary so far

- Representing information
 - External vs. Internal representation
- Computers represent information internally as
 - Binary numbers
- We saw how to represent as binary data:
 - Numbers (integers, negative numbers, floating point)
 - Text (code mappings as ASCII and Unicode)
 - (Graphics, sound, ...)

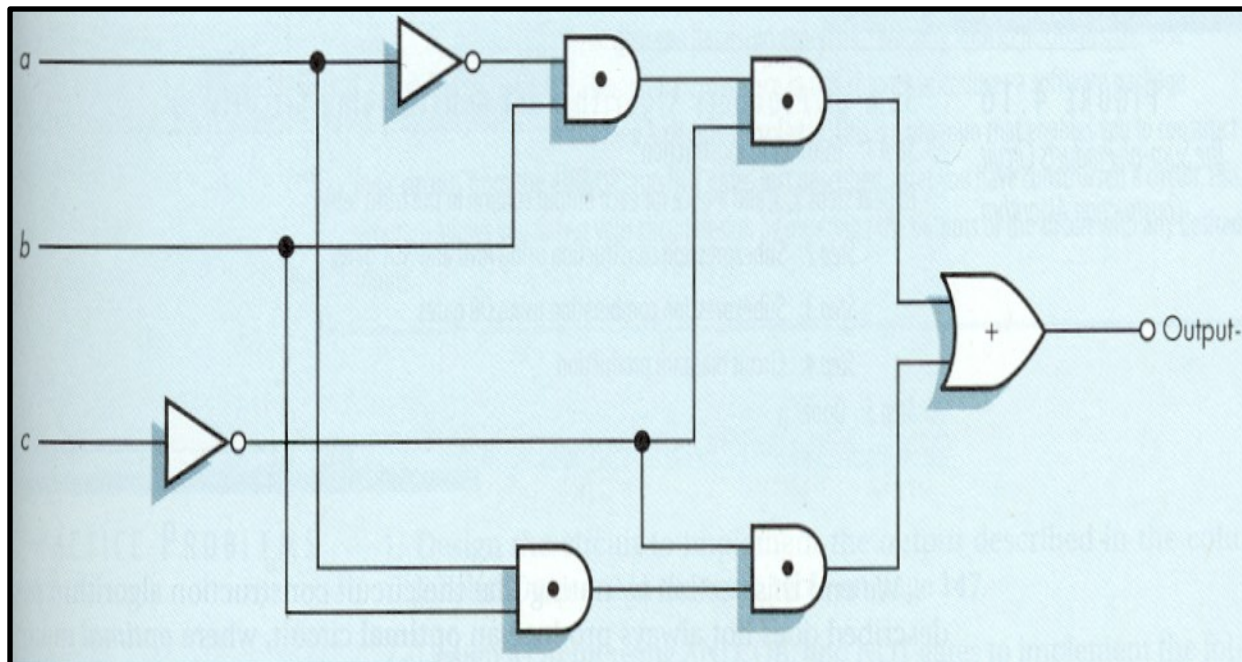
Summary so far (cont.)

- Why do computers use binary data?
 - Reliability
 - Electronic devices work best in a bistable environment, that is, where there are only 2 states.
- Can build a computer using a binary storage device:
 - Has two different stable states, able to sense in which state device is in, and easily switch between states.
- Fundamental binary storage device in computers:
 - Transistor

Summary so far (cont.)

- Boolean Logic
 - Boolean expressions are expressions that evaluate to either true or false.
 - Can use the operators AND, OR, and NOT
- Learned about gates
 - Electronic devices that work with binary input/output.
- Next we will talk about:
 - Circuits built from gates.

Computer Circuits



Purpose

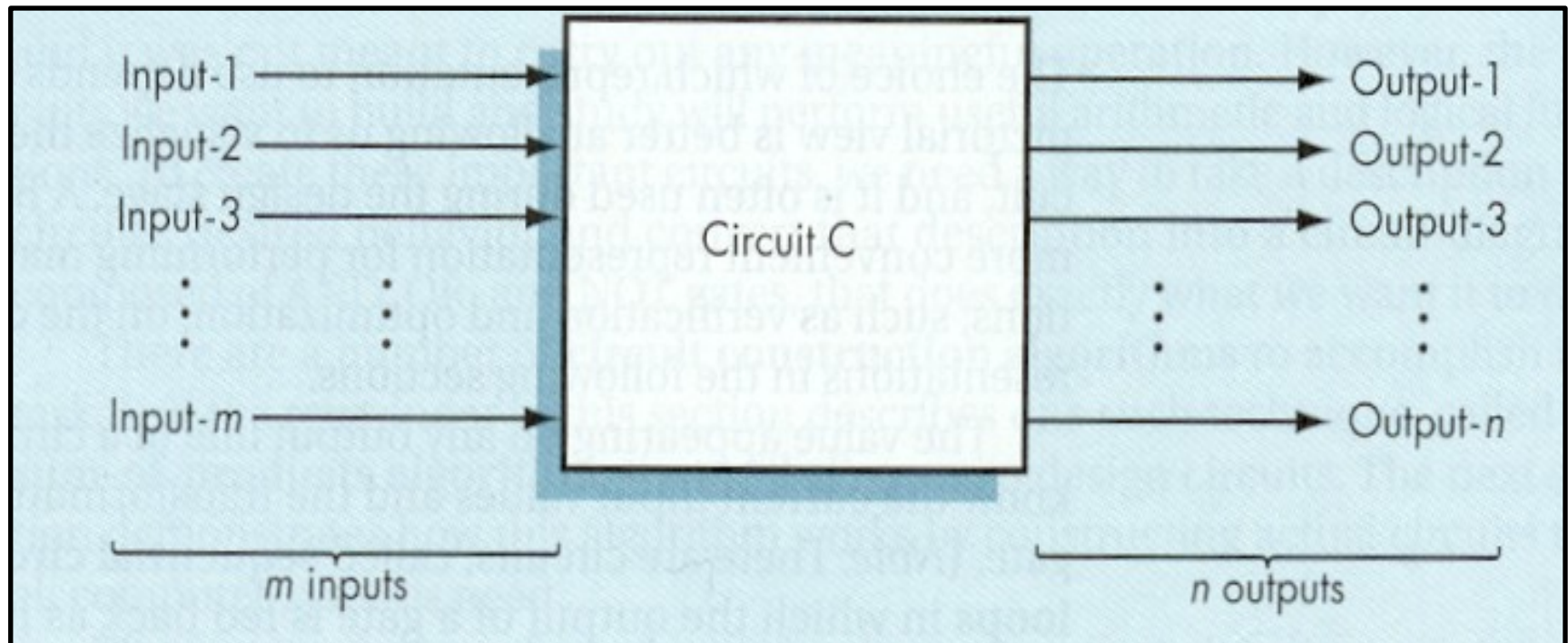
- We have looked at so far how to build logic gates from transistors.
- Next we will look at how to build circuits from logic gates, for example:
 - A circuit to check if two numbers are equal.
 - A circuit to add two numbers.
- Gates will become our new building blocks:
 - Human body: cells->organs->body
 - Computers: gates->circuits->computer

Circuit

- A **circuit** is a collection of interconnected logic gates:
 - that transforms a set of binary inputs into a set of binary outputs, and
 - where the values of the outputs depend only on the current values of the inputs
- These kind of circuits are more accurately called **combinatorial circuits**.

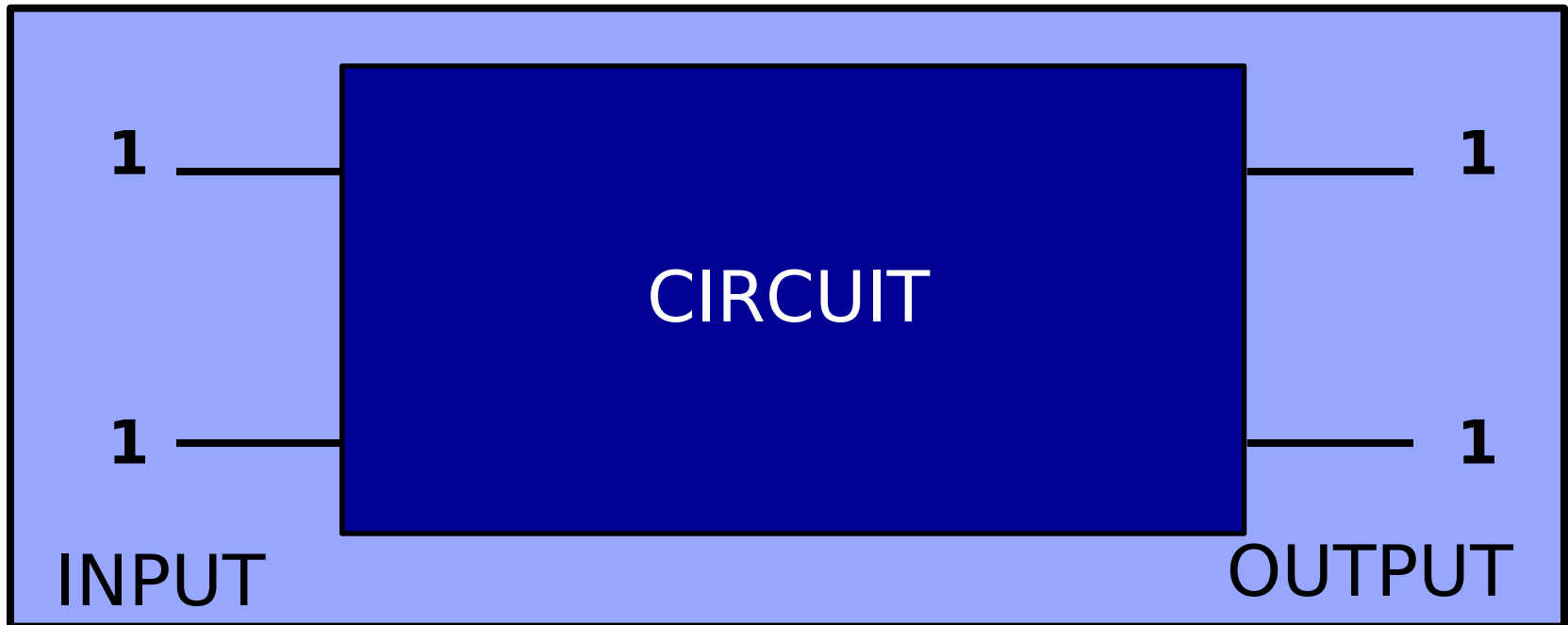
Circuit (external view)

- A circuit can have any number of inputs and outputs:
 - Number of inputs and outputs can differ.
 - The inputs and outputs are either 0 or 1.



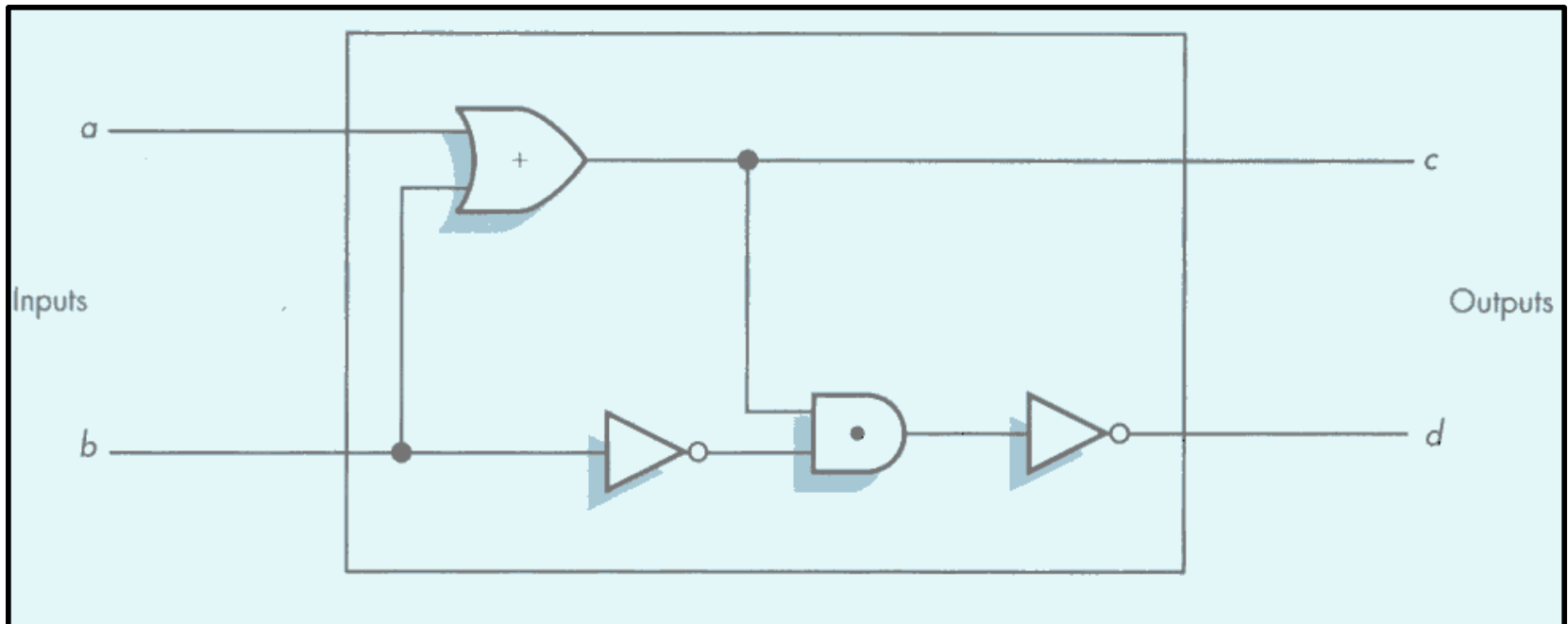
Circuit (external view cont.)

- Output depends only on current input values
 - Each set of input always generates the same output.
 - Different sets of input can generate identical output.



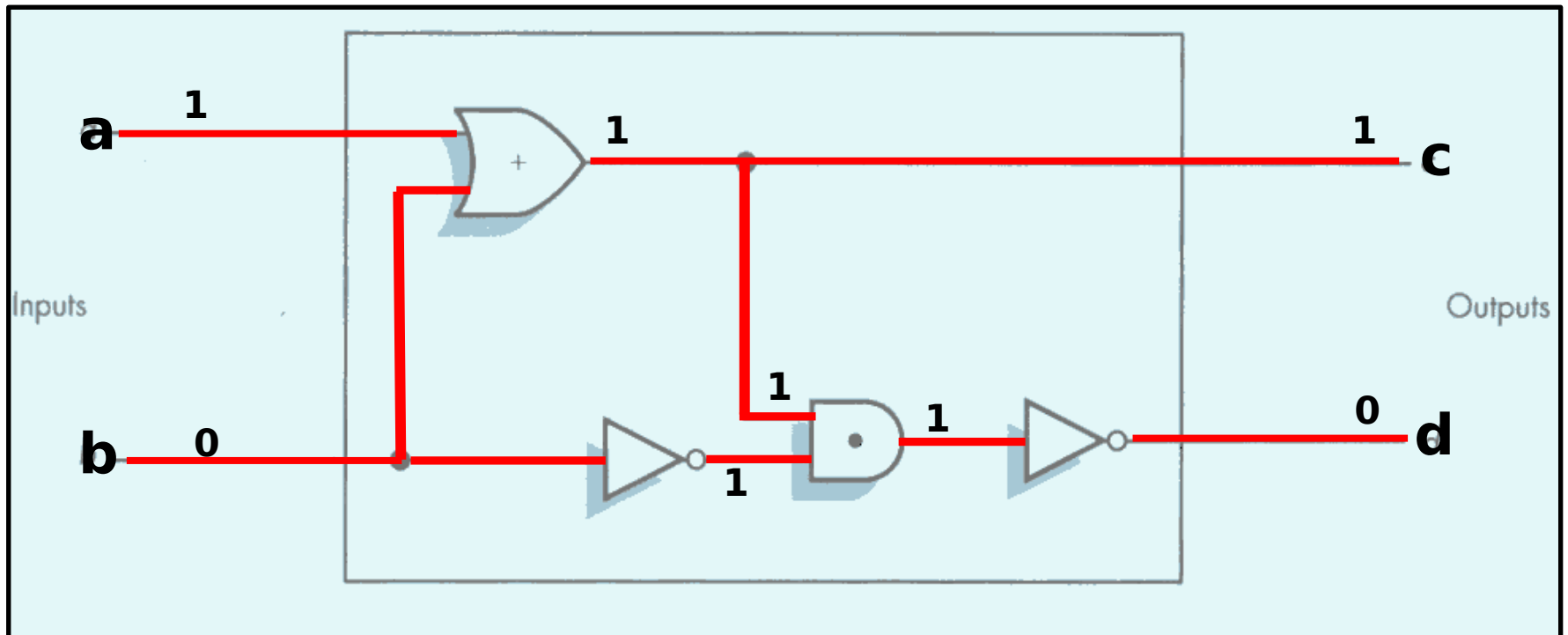
Circuit (internal view)

- Circuits are built from interconnected **AND**, **OR** and **NOT** gates, in a way such that each input combination produces the desired output.



Example

- What are the output values c and d given input values $a=1$, $b=0$?



Circuit Diagrams and Boolean Expr

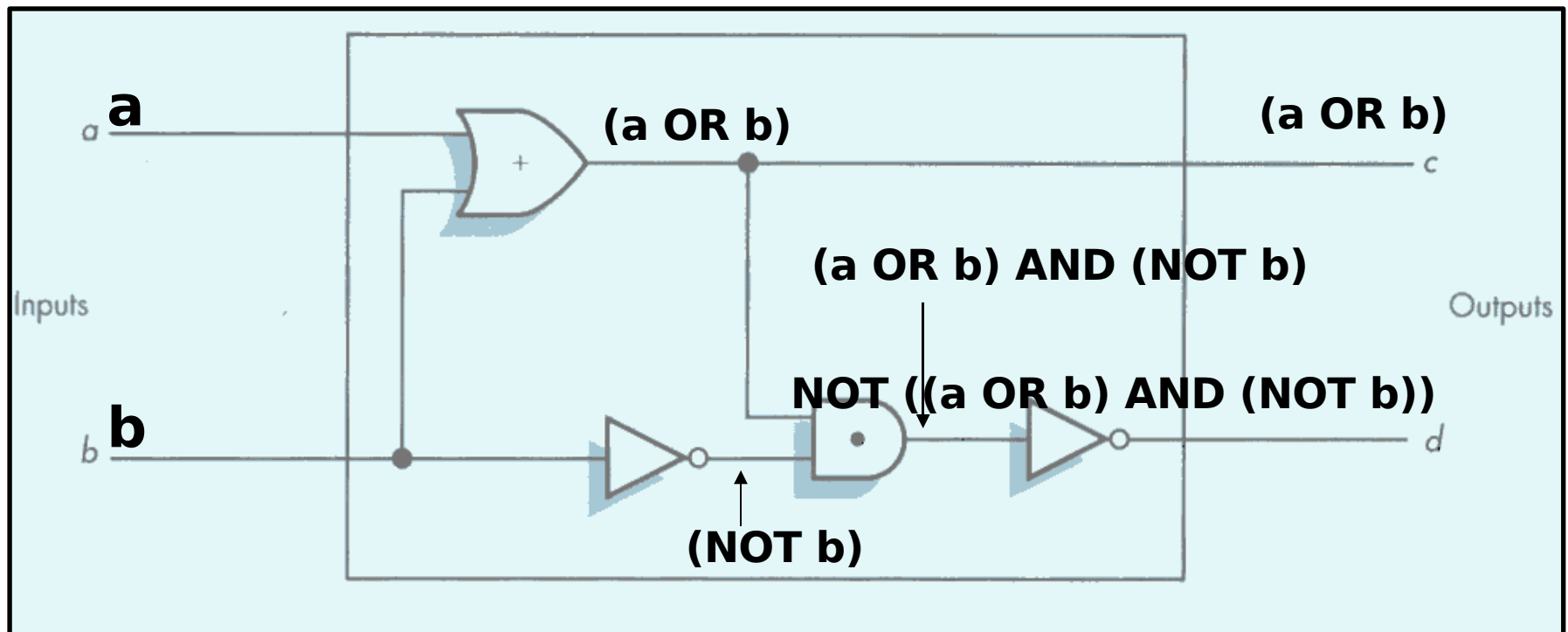
- The diagrams we were looking at are called circuit diagrams.
- Relationship between circuit diagrams and Boolean expressions:
 - Every Boolean expression can be represented pictorially as a circuit.
 - Every output in a circuit diagram can be written as a Boolean expression.
- Example (output values c and d from previous diagram):

$$c = (a \text{ OR } b)$$

$$d = \text{NOT} ((a \text{ OR } b) \text{ AND } (\text{NOT } b))$$

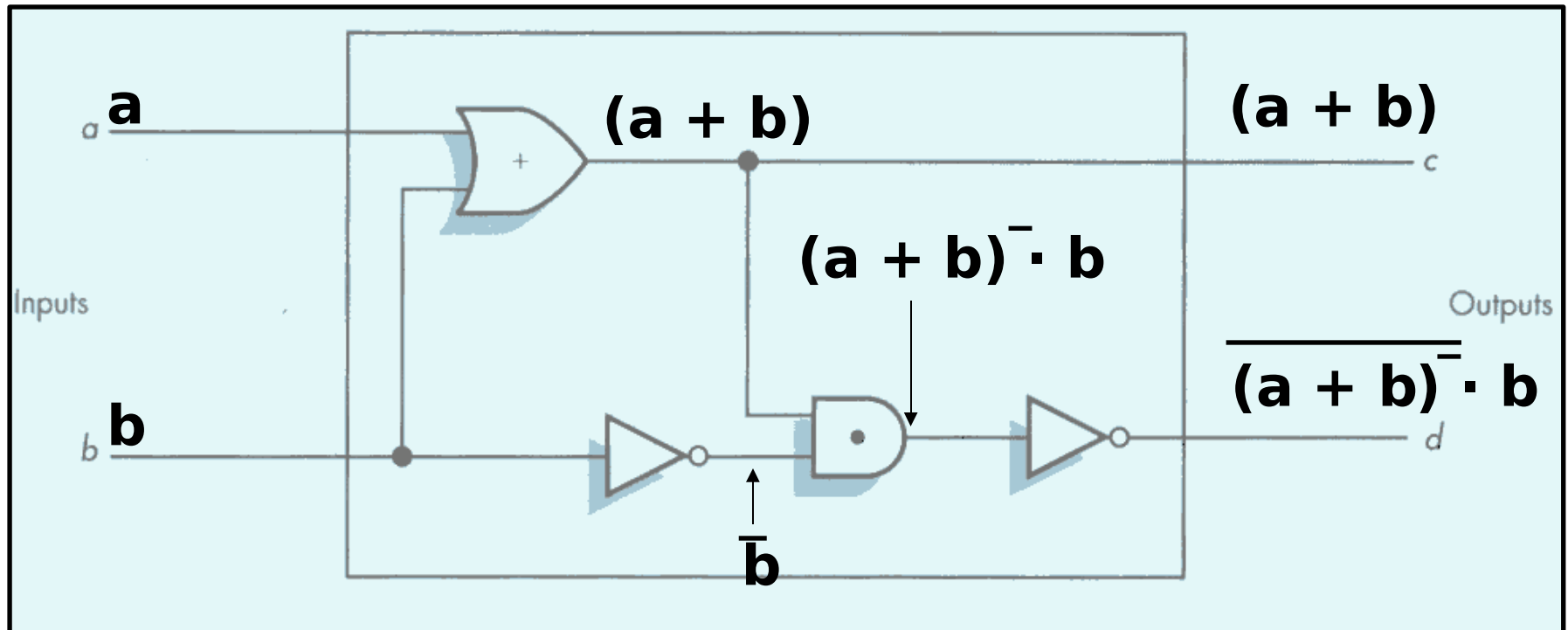
Circuits Diagram and Boolean Expr

- Deriving Boolean expressions for the output.



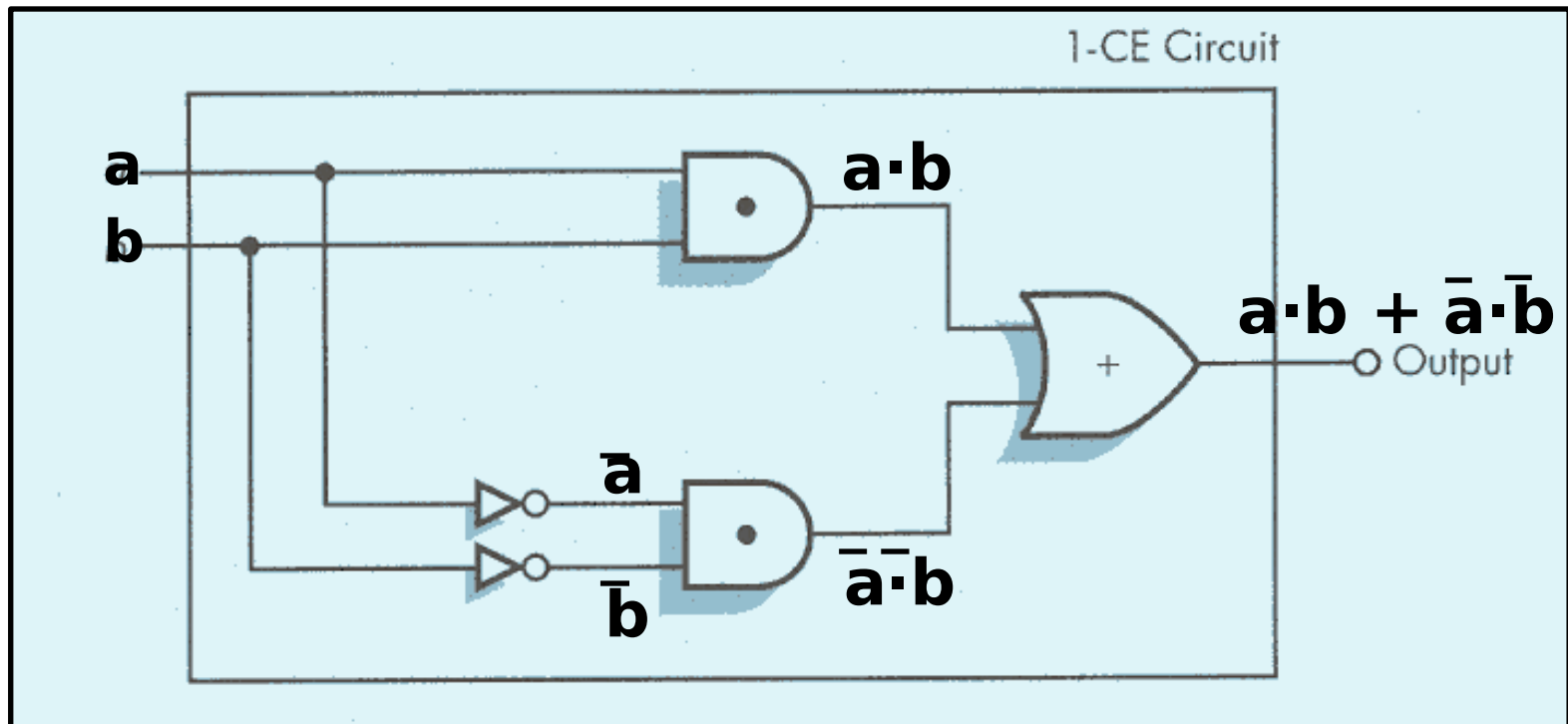
Circuits Diagram and Boolean Expr

- Remember, when writing Boolean expressions for circuit diagrams, we can use a different notation!



Example

- What Boolean expression describes the output?



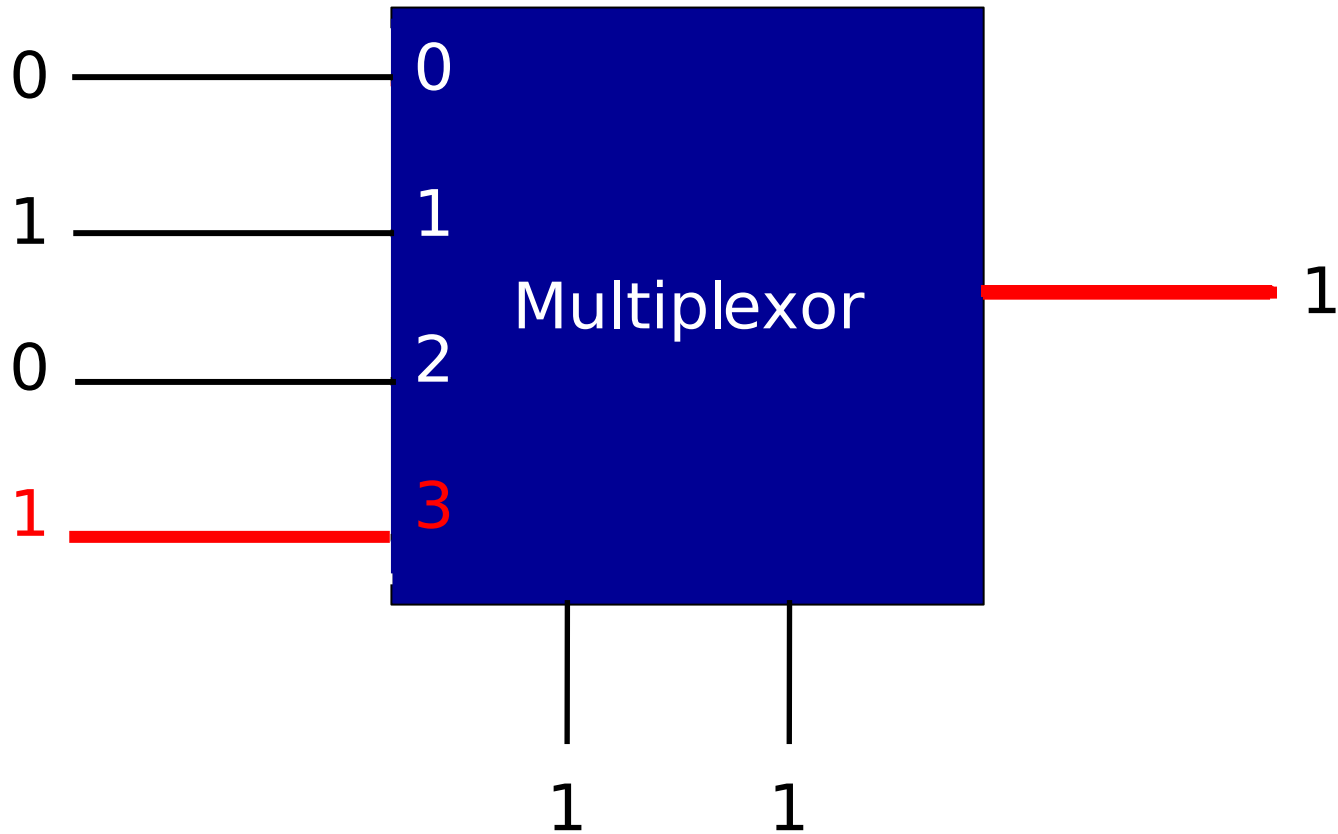
Control Circuits

- So far we have seen two types of circuits:
 - Logical (is $a = b$?)
 - Arithmetic ($c = a + b$)
- Computers use many different logical ($>$, $<$, $>=$, $<=$, $!=$, ...), and arithmetic ($+$, $-$, $*$, $/$) circuits.
- There are also different kind of circuits that are essential for computers → control circuits
 - We will look at two different kind of control circuits, multiplexors and decoders.

Multiplexor

- A multiplexor circuit has:
 - 2^N input lines (numbered $0, \dots, 2^N-1$)
 - 1 output line
 - N selector lines
- The selector lines are used to choose which of the input signals becomes the output signal:
 - Selector lines are interpreted as an N-bit integer
 - The signal on the input line with the corresponding number becomes the output signal.

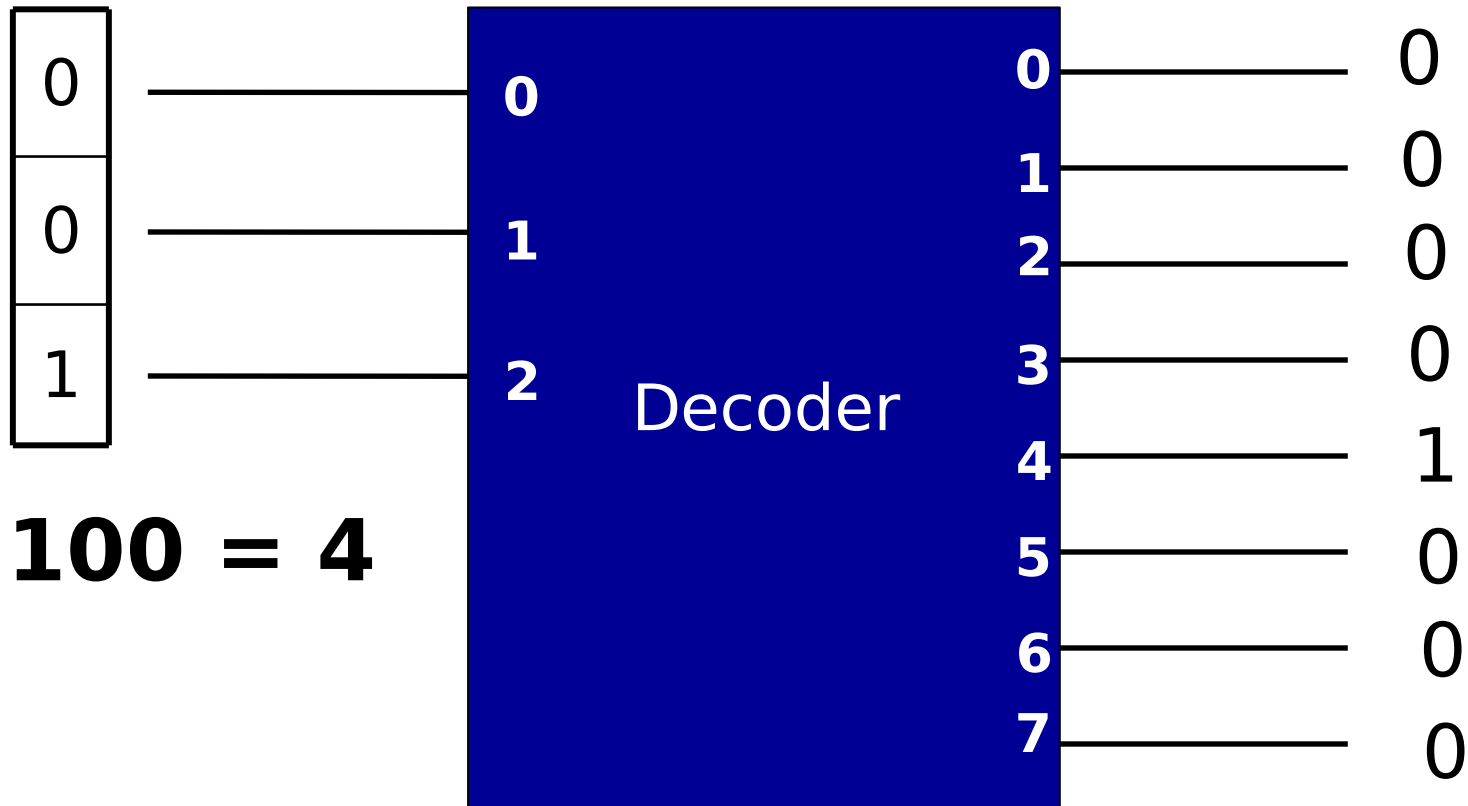
Multiplexor (cont.)



Decoder

- A decoder circuit has:
 - N input lines (numbered $0, 1, \dots, N-1$)
 - 2^N output line (numbered $0, 1, \dots, 2^N-1$)
- Works as follows:
 - The N input lines are interpreted as a N -bit integer value.
 - The output line corresponding to the integer value is set to 1, all other to 0

Decoder (cont.)



Summary

- We looked at how computers represent data:
 - Internal vs External Representation
 - Basic storage unit is a binary digit – bit
 - Data is represented internally as binary data.
 - Use the binary number system.
- We learned why computers use binary data:
 - Main reason is reliability
 - Electronic devices work best in bi-stable environment.

Summary (cont.)

- We looked at the basic building blocks used in computers:
 - Binary Storage Device = Transistor
- We saw how to build logic gates (AND, OR, NOT):
 - Transistors
 - Gates
 - Boolean logic