

Lecture 3

Data representation

Lecture 3 - Data Representation

- **Number systems in the world and history**
 - Non-positional / additive number systems:
 - Egyptian
 - Roman
 - Greek
 - Positional number systems
 - Babylonian (a 2-digit, sexagesimal)
 - Celtic, Maya, Inuit (vigesimal)
 - Hindu-Arabic (decimal)
 - Hybrid
 - Chinese

Lecture 3 - Data Representation

- **Number systems in the world and history**

- Non-positional / additive number systems:

- Egyptian
 - Roman
 - Greek

- Positional number systems

- Babylonian (a 2-digit, sexagesimal)
 - Celtic, Maya, Inuit (vigesimal)
 - Hindu-Arabic (decimal)



- Hybrid

- Chinese

Lecture 3 - Data Representation

- **Positional Number System**

- Needs to define a “base” B
- from B depend:
 - set of digits = $\{0, 1, \dots, B-1\}$
 - positional weight of i -th digit = B^i
 - representation of natural numbers with N digits

$$a_{N-1} \ a_{N-2} \ \dots \ a_2 \ a_1 \ a_0$$

$$B^{N-1} \cdot a_{N-1} + B^{N-2} \cdot a_{N-2} + \dots + B^2 \cdot a_2 + B^1 \cdot a_1 + B^0 \cdot a_0$$

$$A = \sum_{i=0}^{N-1} B^i \cdot a_i$$

Lecture 3 - Data Representation

- **Positional Number System** $B^{N-1} \cdot a_{N-1} + B^{N-2} \cdot a_{N-2} + \dots + B^2 \cdot a_2 + B^1 \cdot a_1 + B^0 \cdot a_0$
 - set of digits = $\{0, 1, \dots, B-1\}$

$$A = \sum_{i=0}^{N-1} B^i \cdot a_i \quad \longrightarrow \quad A_{max} = \sum_{i=0}^{N-1} B^i (B - 1) = \sum_{i=0}^{N-1} B^{i+1} - \sum_{i=0}^{N-1} B^i$$

$$= B^1 + B^2 + B^3 + \dots + B^{N-3} + B^{N-2} + B^{N-1} + B^N \\ -(1 + B^1 + B^2 + B^3 + \dots + B^{N-3} + B^{N-2} + B^{N-1})$$

$$\rightarrow A_{max} = B^N - 1$$

Lecture 3 - Data Representation

- Positional Number System

- example: $B = 10$

- digits = $\{0, 1, 2, \dots, 9\}$

$a_{N-1} a_{N-2} \dots a_2 a_1 a_0$

$$B^{N-1} \cdot a_{N-1} + \dots + B^1 \cdot a_1 + B^0 \cdot a_0$$

1784

$$10^3 \cdot 1 + 10^2 \cdot 7 + 10^1 \cdot 8 + 10^0 \cdot 4$$

Lecture 3 - Data Representation

- Binary System

- Base = 2
- digits = {0, 1}
- BIT = Binary DigiT

- Example: $101_2 = 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 4 + 0 + 1 = 5_{10}$

Lecture 3 - Data Representation

- Binary System

- Base = 2
- digits = {0, 1}
- BIT = Binary DigiT

- Example: $101_2 = 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 4 + 0 + 1 = 5_{10}$

Def:
Most Significant Bit
(MSB)

Def:
Least Significant Bit
(LSB)

Lecture 3 - Data Representation

- **Binary System**

- Natural Numbers conversion

- **Binary to Decimal:** apply the definition

- Example:

$$1101_2 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 8 + 4 + 0 + 1 = 13_{10}$$

Lecture 3 - Data Representation

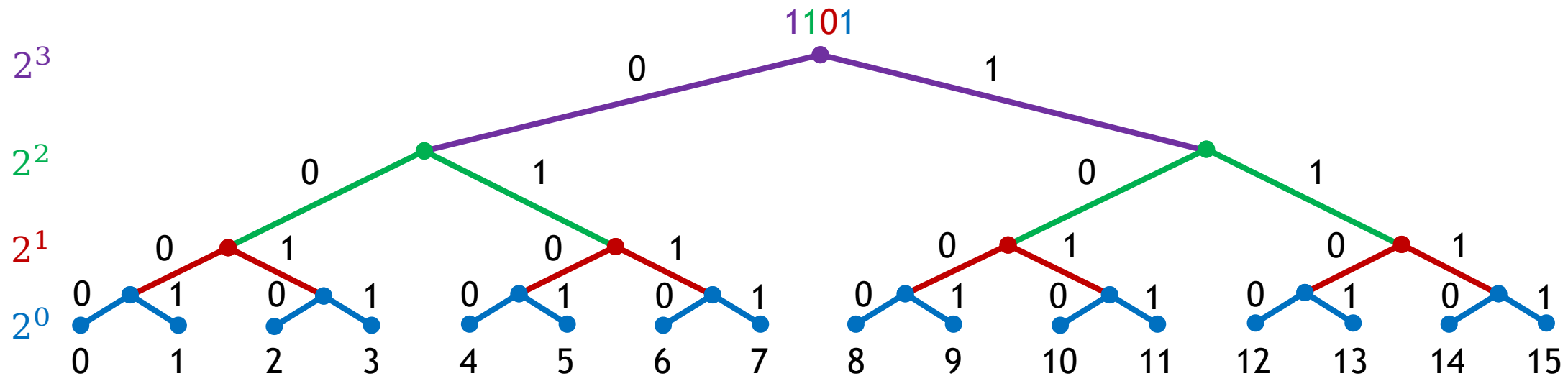
- Binary System

- Natural Numbers conversion

- Binary to Decimal: apply the definition

- Example:

$$1101_2 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 8 + 4 + 0 + 1 = 13_{10}$$



Lecture 3 - Data Representation

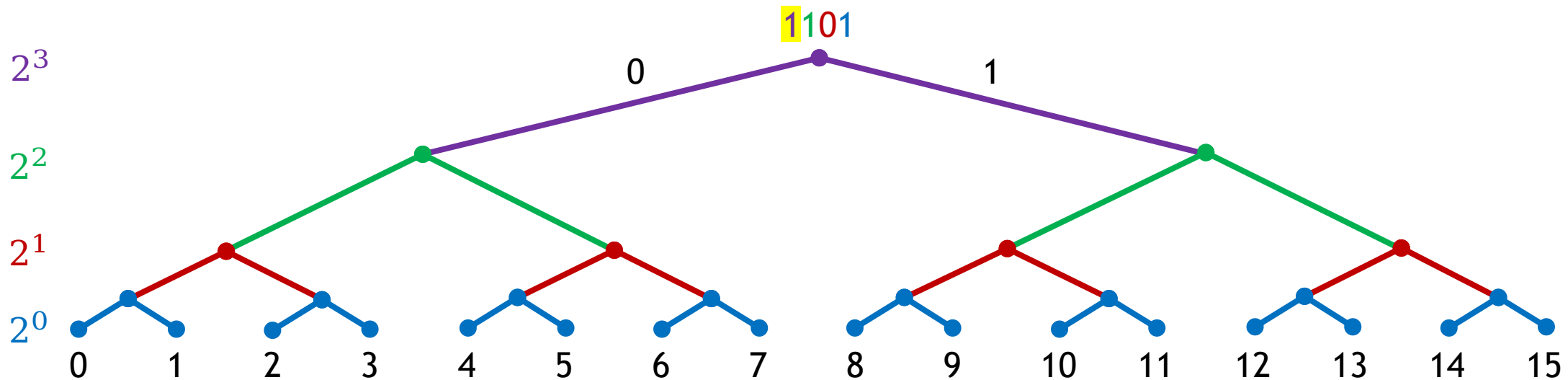
- Binary System

- Natural Numbers conversion

- Binary to Decimal: apply the definition

- Example:

$$1101_2 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 8 + 4 + 0 + 1 = 13_{10}$$



Lecture 3 - Data Representation

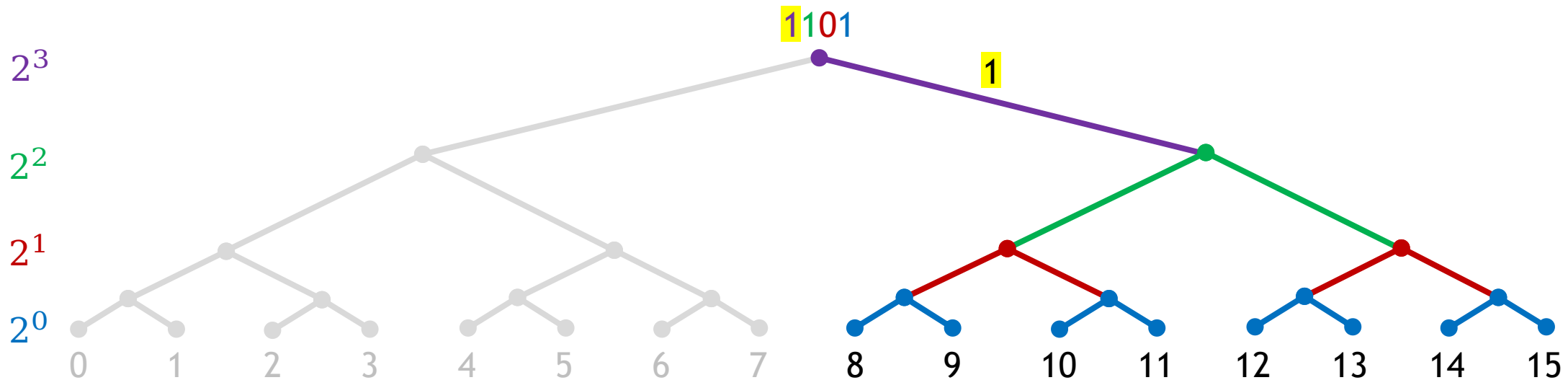
- Binary System

- Natural Numbers conversion

- Binary to Decimal: apply the definition

- Example:

$$1101_2 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 8 + 4 + 0 + 1 = 13_{10}$$



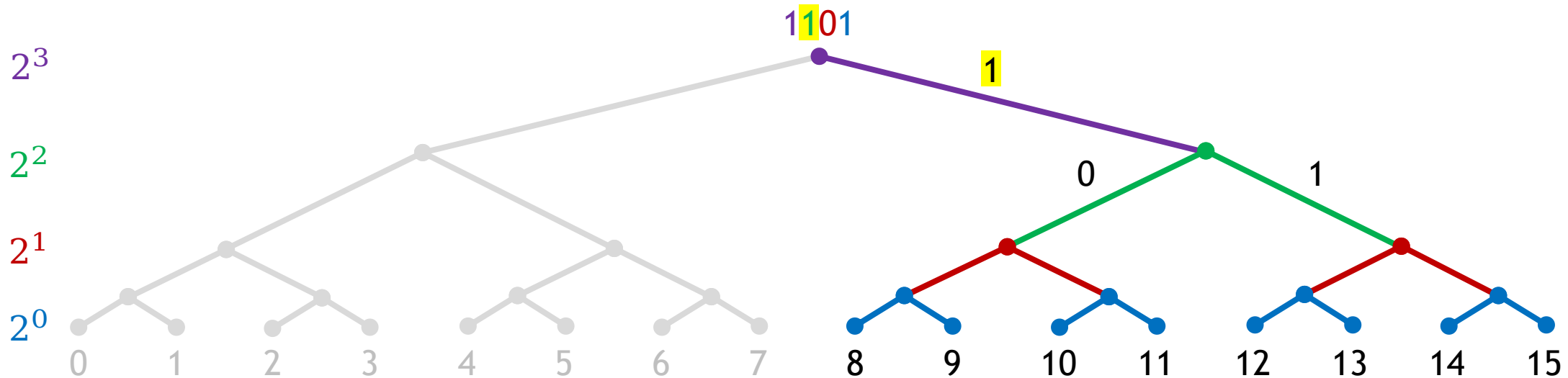
Lecture 3 - Data Representation

- **Binary System**

- Natural Numbers conversion

- **Binary to Decimal:** apply the definition
- Example:

$$1101_2 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 8 + 4 + 0 + 1 = 13_{10}$$



Lecture 3 - Data Representation

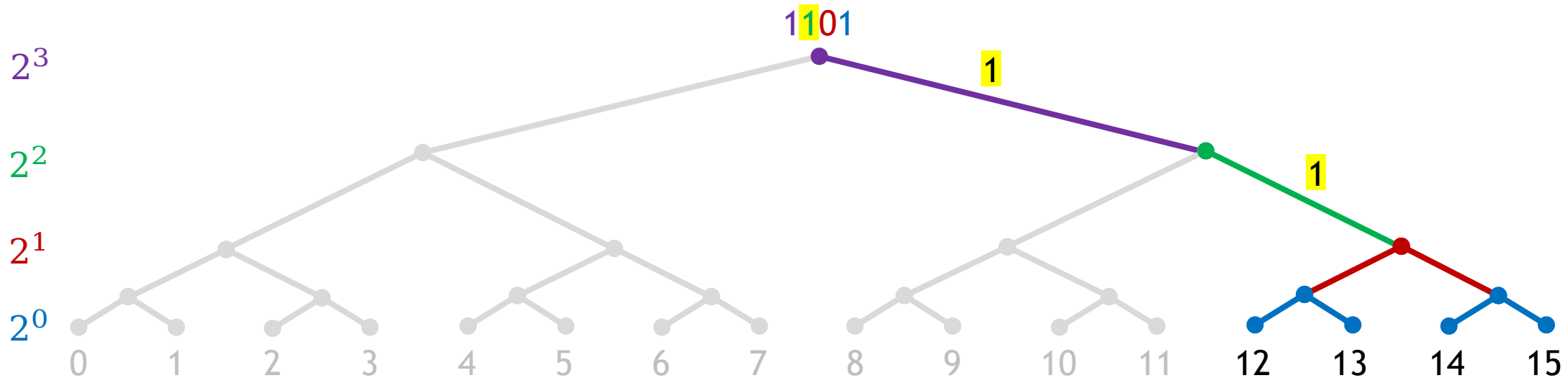
- Binary System

- Natural Numbers conversion

- Binary to Decimal: apply the definition

- Example:

$$1101_2 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 8 + 4 + 0 + 1 = 13_{10}$$



Lecture 3 - Data Representation

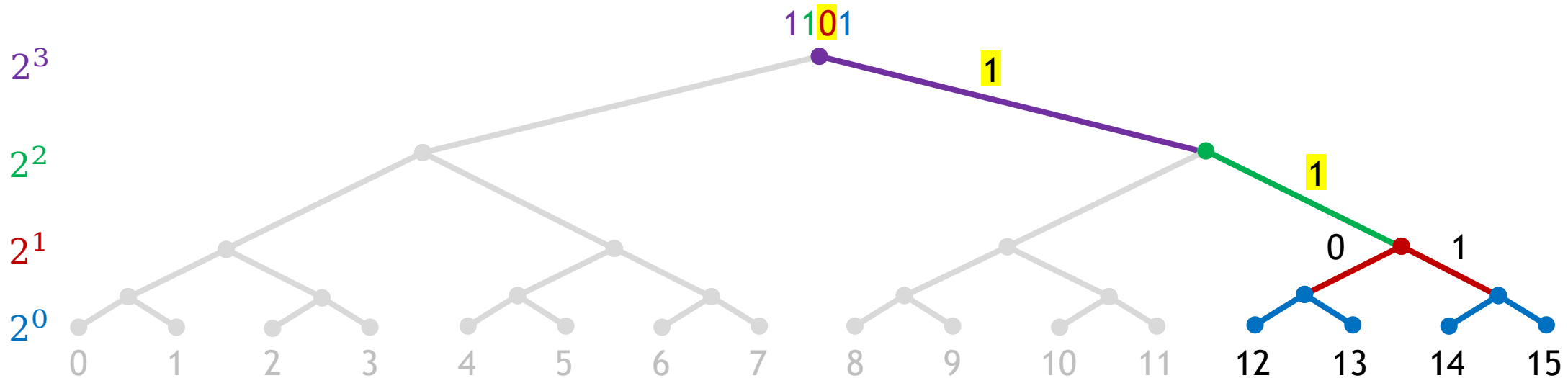
- Binary System

- Natural Numbers conversion

- Binary to Decimal: apply the definition

- Example:

$$1101_2 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 8 + 4 + 0 + 1 = 13_{10}$$



Lecture 3 - Data Representation

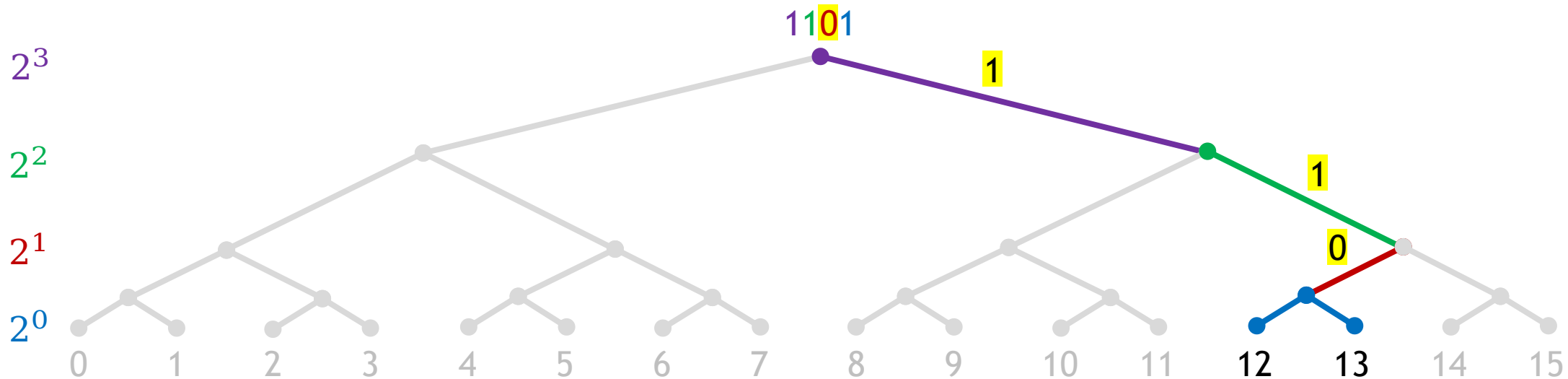
- Binary System

- Natural Numbers conversion

- Binary to Decimal: apply the definition

- Example:

$$1101_2 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 8 + 4 + 0 + 1 = 13_{10}$$



Lecture 3 - Data Representation

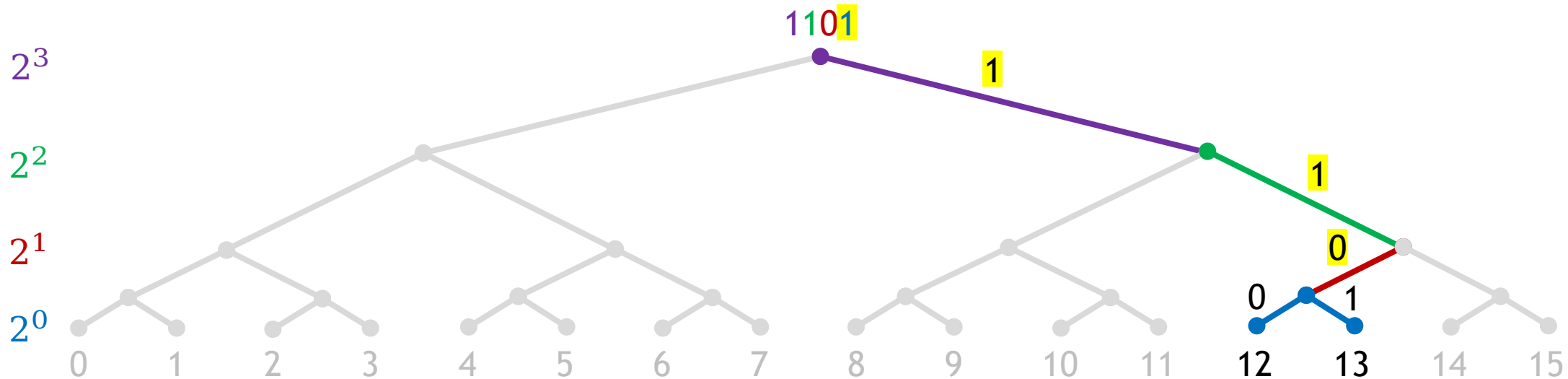
- Binary System

- Natural Numbers conversion

- Binary to Decimal: apply the definition

- Example:

$$1101_2 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 8 + 4 + 0 + 1 = 13_{10}$$



Lecture 3 - Data Representation

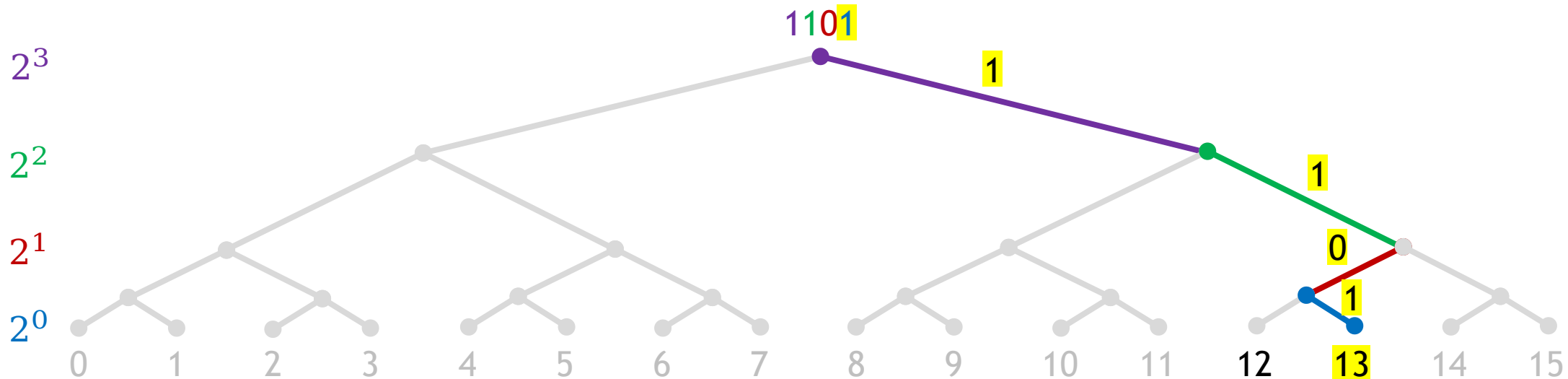
- Binary System

- Natural Numbers conversion

- Binary to Decimal: apply the definition

- Example:

$$1101_2 = 2^3 \cdot 1 + 2^2 \cdot 1 + 2^1 \cdot 0 + 2^0 \cdot 1 = 8 + 4 + 0 + 1 = 13_{10}$$



Lecture 3 - Data Representation

- **Binary System**

- Natural Numbers conversion

- Decimal to Binary:

- Successive divisions by 2
 - Take the remainders in reverse order

- Example:

13_{10}	2
6	1
3	0
1	1
0	1

Lecture 3 - Data Representation

- **Binary System**

- Natural Numbers conversion

- Decimal to Binary:

- Successive divisions by 2
 - Take the remainders in reverse order

- Example:

13_{10}	2
6	1
3	0
1	1
0	1



$$13_{10} = 1101_2$$

Lecture 3 - Data Representation

- **Binary System: Limits to the representation of Natural Numbers**

- 1 bit $\rightarrow$ 2 values $[0, 1]_2 \leftrightarrow [0, 1]_{10}$
- 2 bits $\rightarrow$ 4 values $[00, 01, 10, 11]_2 \leftrightarrow [0, 1, 2, 3]_{10}$
- 3 bits $\rightarrow$ 8 values $[000, 001, 010, 011, 100, 101, 110, 111]_2 \leftrightarrow [0, 1, 2, 3, 4, 5, 6, 7]_{10}$
- ...

- General rule for Natural Numbers with N bits:

$$0 \leq b \leq 2^N - 1$$

- Corresponding to 2^N unique combinations

Lecture 3 - Data Representation

- Hexadecimal system (abbr. H, X or hex)
 - Base = 16
 - digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
 - often used for compact writing of binary code (4:1)
- $16 = 2^4 \rightarrow$ a hex digit is equivalent to 4 bits: $0 \leq x \leq 16^N - 1 = 2^{4N} - 1$
- $\rightarrow$ any binary number can be converted in hexadecimal replacing the corresponding hex digit to each 4-digits block of the binary number

Lecture 3 - Data Representation

- Hexadecimal system (abbr. H, X or hex)

- Base = 16
- digits = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}
- often used for compact writing of binary code (4:1)

- $16 = 2^4 \rightarrow$ a hex digit is equivalent to 4 bits: $0 \leq x \leq 16^N - 1 = 2^{4N} - 1$
- $\rightarrow$ any binary number can be converted in hexadecimal replacing the corresponding hex digit to each 4-digits block of the binary number
- example:

10111001₂

B 9 16

Often written 0xB9
to specify it is Hexadecimal

Lecture 3 - Data Representation

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Lecture 3 - Data Representation

- **Terminology**

- bit → single digit
- byte → sequence of 8 bit (unit of memory storage)
- word → aggregation of bytes (1, 2, 4, 8, a fixed-size group of bits that a processor is designed to handle as a single unit)

Lecture 3 - Data Representation

- Sum in binary

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0 \text{ (carry=1)}$$

Example:

$$\begin{array}{rcccccc} & 1 & & 1 & & \\ 0 & 1 & 1 & 0 & + & \\ 0 & 1 & 1 & 1 & = & \\ \hline 1 & 1 & 0 & 1 & & \end{array}$$

Lecture 3 - Data Representation

- Subtraction in binary

$$0 - 0 = 0$$

$$0 - 1 = 1 \text{ (borrow=1)}$$

$$1 - 0 = 1$$

$$1 - 1 = 0$$

Example:

$$\begin{array}{r} \overset{0}{\cancel{1}} \overset{1}{0} 1 1 + \\ 0 1 0 1 = \\ \hline 0 1 1 0 \end{array}$$

Lecture 3 - Data Representation

- **Overflow**

- Error: the result of an operation cannot be represented using the same encoding and number of bits as the operands.
- Overflow is a “dynamic” condition: it only occurs as the result of an operation.
- Example:

$$\begin{array}{r} \\ \\ \\ \\ \hline \end{array}$$

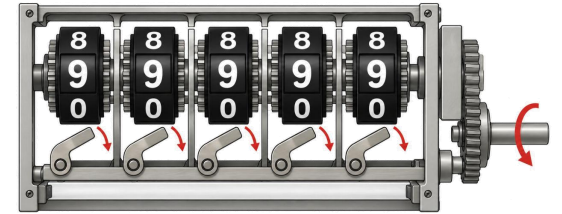
Lecture 3 - Data Representation

- **Overflow and programming languages: where is the problem?**

- real limit of the data representation
- In some languages, the memory space reserved for a value is fixed and depends on the type
- Example: C Language
 - `int` is `int32` → for an integer the system reserves 32 bytes of memory
 - Max integer value = $2^{32} - 1$
 - increasing this value by +1 we lose the carry value → = 0

```
x = 2 ** 32 - 1
x = x + 1      # x = 0 !!!
```

- Python avoids overflows, using an arbitrary-precision integer representation



$$\begin{array}{r} 1 \quad 1 \quad 1 \\ 1 \quad 1 \quad 1 \quad + \\ 0 \quad 0 \quad 1 \quad = \\ \hline 0 \quad 0 \quad 0 \end{array}$$

Data Representation

How to store data in the memory

Lecture 3 - Data Representation

- Why do computer use binary?

Lecture 3 - Data Representation

- Why do computer use binary?

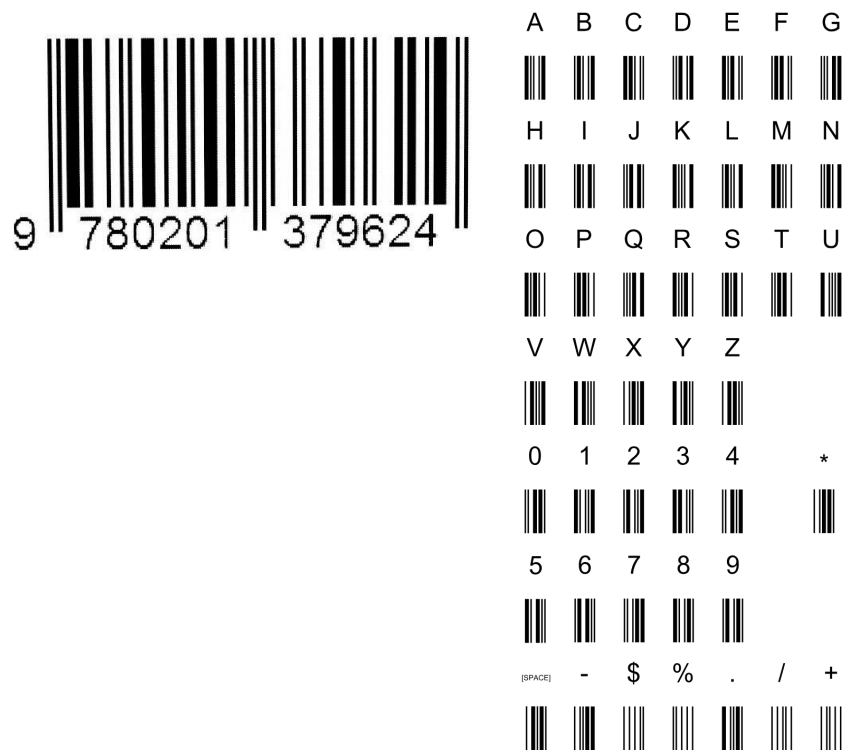
- All modern digital technology is based on **transistors**
- This basic electronic components can reliably distinguish between two states:
 - ON → 1
 - OFF → 0
- At a very high (inaccurate) level, it can be considered as a switch, turning on and off a bulb
- A programmer is someone who sends a light signal by turning on and off a set of bulbs



Binary sistem is used not because it is superior, but because it fits naturally with digital electronic hardware

Lecture 3 - Data Representation

- Where do we find binary encoding?
 - It may look old school, but it's at the core of modern life.



codes



Pits in burned DVDs

Lecture 3 - Data Representation

- From bits to data

- Sequences of bits can encode different kinds of information
- The interpretation depends on the convention (meaning) defined between sender and receiver
- *At the hardware level, everything is ultimately represented as bits; the meaning of those bits depends on how they are interpreted.*
- Examples:
 - Characters (text)
 - Numbers
 - Color of a pixel (image)
 - Sound
 - Instruction

01101000	↔	h
01100101	↔	e
01101100	↔	l
01101100	↔	l
01101111	↔	o
00100000	↔	space
01110100	↔	t
01101000	↔	h
01100101	↔	e
01110010	↔	r
01100101	↔	e
00100001	↔	!

Lecture 3 - Data Representation

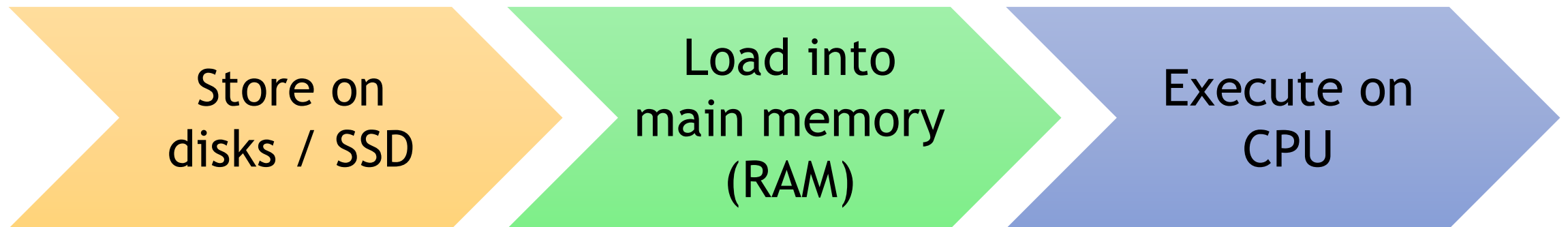
Where is the information stored?

Main Memory (RAM)

- fast
- directly used while programs are running
- volatile (deleted if not alimented)

Secondary (Mass) Storage

- slow
- SSD / Hard disks - long term
- persistent

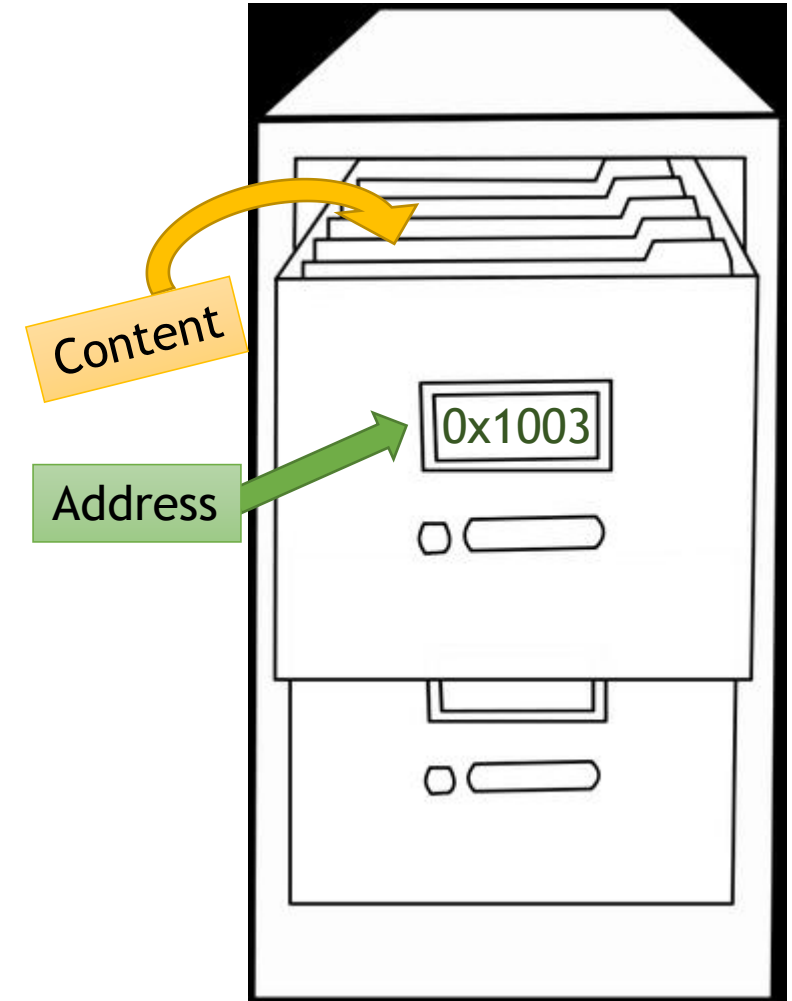


Lecture 3 - Data Representation

- How is memory organized?

- Main memory can be viewed, at a high level, as a sequence of storage locations
- Each location has an **address**, which uniquely identifies its position.
- Modern computers are typically **byte-addressable**: each address identifies one byte.

Address	Content	
0x1000	01010101	[8 bits]
0x1001	11000010	[8 bits]
0x1002	00001101	[8 bits]
0x1003	10110001	[8 bits]
...		



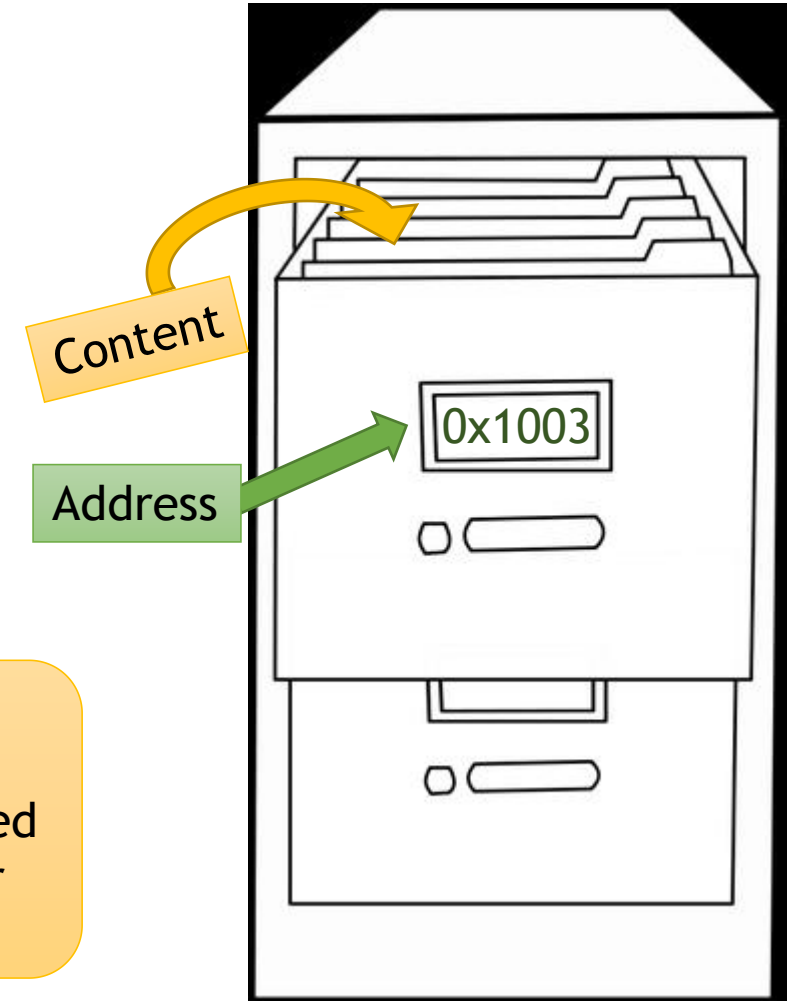
Lecture 3 - Data Representation

- How is memory organized?

- Main memory can be viewed, at a high level, as a sequence of storage locations
- Each location has an **address**, which uniquely identifies its position.
- Modern computers are typically **byte-addressable**: each address identifies one byte.

Address	Content	
0x1000	01010101	[8 bits]
0x1001	11000010	[8 bits]
0x1002	00001101	[8 bits]
0x1003	10110001	[8 bits]
...		

NOTE: Several consecutive bytes (*Word*) can be handled together as a larger unit of data.



Lecture 3 - Data Representation

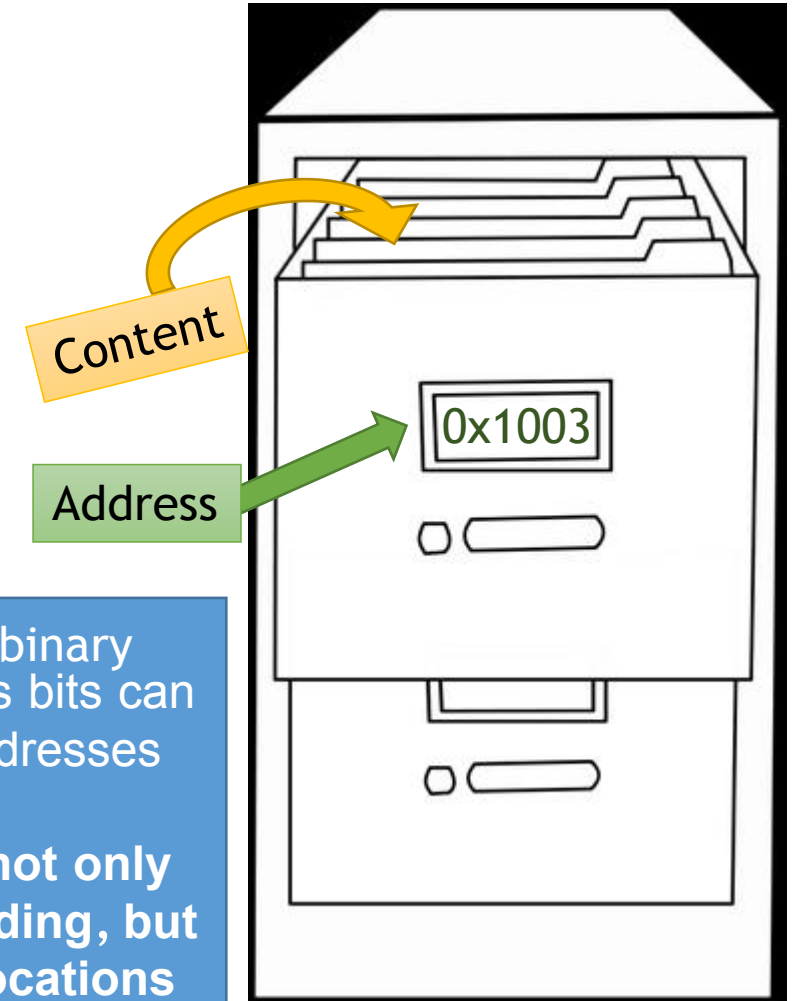
- How is memory organized?

- Main memory can be viewed, at a high level, as a sequence of storage locations
- Each location has an **address**, which uniquely identifies its position.
- Modern computers are typically **byte-addressable**: each address identifies one byte.

Address	Content	
0x1000	01010101	
0x1001	11000010	[8 bits]
0x1002	00001101	[8 bits]
0x1003	10110001	[8 bits]
...		

Memory addresses are binary numbers too → N address bits can represent 2^N different addresses

Binary representation not only determines values encoding, but also identify memory locations



Lecture 3 - Data Representation

- **Data types and memory**

- Different kinds of data require different amounts of memory
- A value has a ***type*** → tell the system how to interpret the data and how much space is needed
- These sizes are architecture- and language-dependant
- Larger data types (like 4-byte integers or 8-byte pointers) span multiple sequential addresses. The address used points to the first byte of that group.

Data type	Example	Typical size in low-level languages (C-like)
Boolean	True, False	1 byte
Character	'A', '4', '?'	1-4 bytes
Integer	42	4 bytes
Floating point	3.14	8 bytes

Lecture 3 - Data Representation

- Data types and memory

Variables	Type	Address	Content
name	string	0x1000	0x4A - 'J'
		0x1001	0x6F - 'o'
		0x1002	0xE3 - 'ã'
		0x1003	0x6F - 'o'
age	int	0x1004	
height	float	0x1005	0x3FE3D70A - 1.78
city	string	...	
		0x100C	
		0x100D	0x46 - 'F'
		0x100E	0x61 - 'a'
		0x100F	0x72 - 'r'
		0x1010	0x6F - 'o'
		0x1011	0x0000001C - 28
		...	
		0x1014	

Data type	Example	Typical size in low-level languages (C-like)
Boolean	True, False	1 byte
Character	'A', '4', '?'	1-4 bytes
Integer	42	4 bytes
Floating point	3.14	8 bytes

Lecture 3 - Data Representation

- **Data types and memory:** What about Python?



- In languages such as C, basic data types often have a relatively fixed memory size.
- Python is more flexible: variables do not directly contain fixed-size values. They refer to objects, whose memory usage can vary (managed automatically).
- Example:
 - `x = 10`
 - `y = 10**100`
 - Both types are integers, but not necessarily need the same size of memory

Representation of signed integers

Lecture 3 - Data Representation

- How to represent integer numbers with sign?
 - Sign can be either positive (+) or negative (-)



Lecture 3 - Data Representation

- How to represent integer numbers with sign?
 - Sign can be either positive (+) or negative (-)
 - Easy representation in binary... but not always the easy way is the best!
 - 2 main solutions:
 - sign-and-magnitude representation
 - two's complement

Lecture 3 - Data Representation

- How to represent integer numbers with sign?
 - **sign-and-magnitude representation** (with N bits):
 - a bit for the sign (0 for +, 1 for -)
 - $N-1$ bits for the magnitude (absolute value)



$+3_{10}$	→	0011
-3_{10}	→	1011

- $-(2^{N-1} - 1) \leq x \leq (2^{N-1} - 1)$
- es. 8 bits → [-127 ... +127]

Lecture 3 - Data Representation

- How to represent integer numbers with sign?
 - **sign-and-magnitude representation** (with N bits): Limits
 - Double 0 (+0, -0)

$+0_{10}$	$\rightarrow$	0000
-0_{10}	$\rightarrow$	1000
 - More complex operations

+	$A > 0$	$A < 0$
$B > 0$	$A + B$	$B - A $
$B < 0$	$A - B $	$-(A + B)$

Lecture 3 - Data Representation

- How to represent integer numbers with sign?

- two's complement representation** (with N bits)

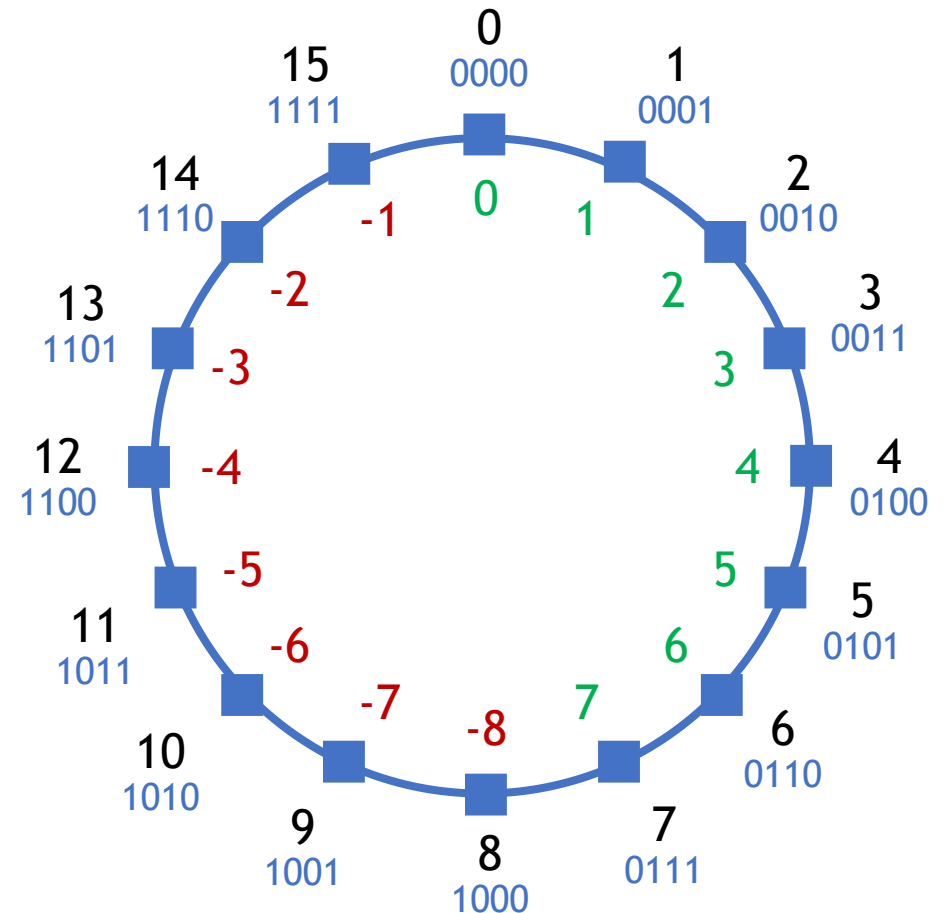
- MSB has negative value
- All the remaining positions have positive value

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

-2^3 2^2 2^1 2^0

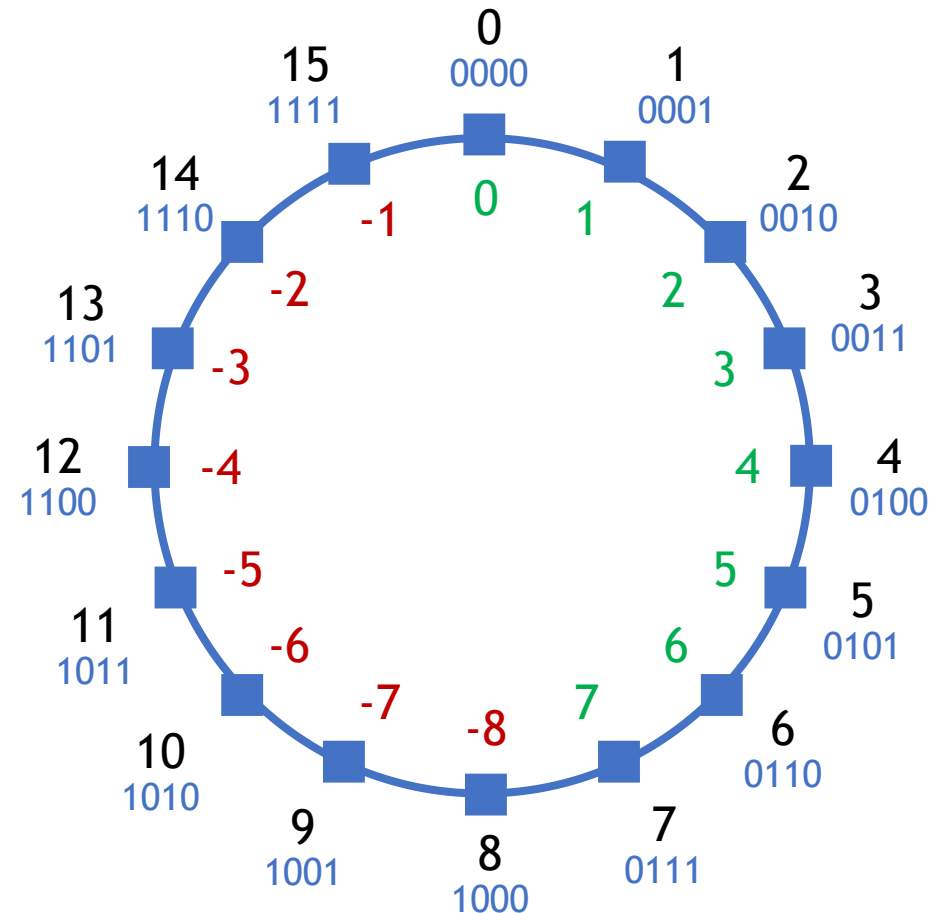
$$-8 \quad +2 \quad +1 \quad = -5_{10}$$

- Note:** as Sign and Mag., MSB indicates the sign
- $-(2^{N-1}) \leq x \leq (2^{N-1} - 1)$
- es. 8 bits $\rightarrow [-128 \dots +127]$



Lecture 3 - Data Representation

- How to represent integer numbers with sign?
 - **two's complement representation** (with N bits)
 - Simplified operations (simple circuits)
 - most common representation in modern computers
 - Simple rule to convert a signed number N_{10} to 2'sC:
 - if MSB = 0 $\rightarrow$ apply normal conversion (Positive)
 - if MSB = 1
 - 1. Convert module of N_{10} in binary N_2
 - 2. take the two's complement ($0 \rightarrow 1$, $1 \rightarrow 0$) of N_2
 - 3. add +1

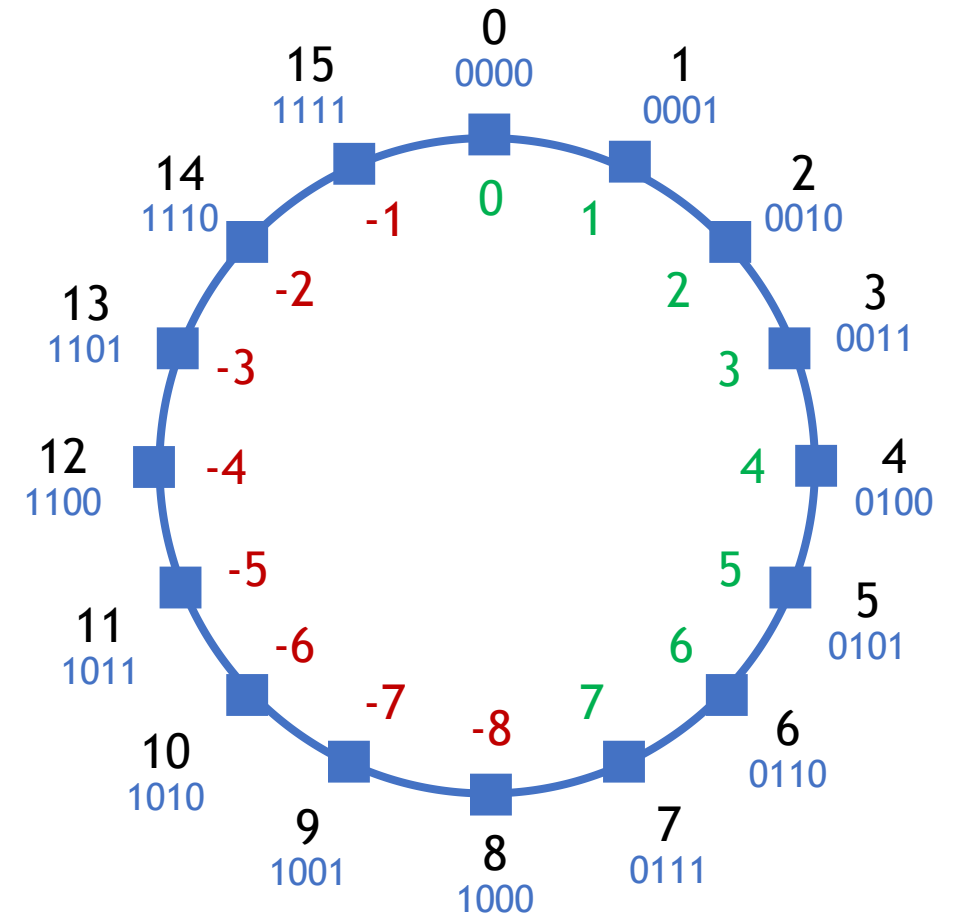


Lecture 3 - Data Representation

- How to represent integer numbers with sign?
 - **two's complement representation**
 - Example of signed integer to 2's Complement for -5:

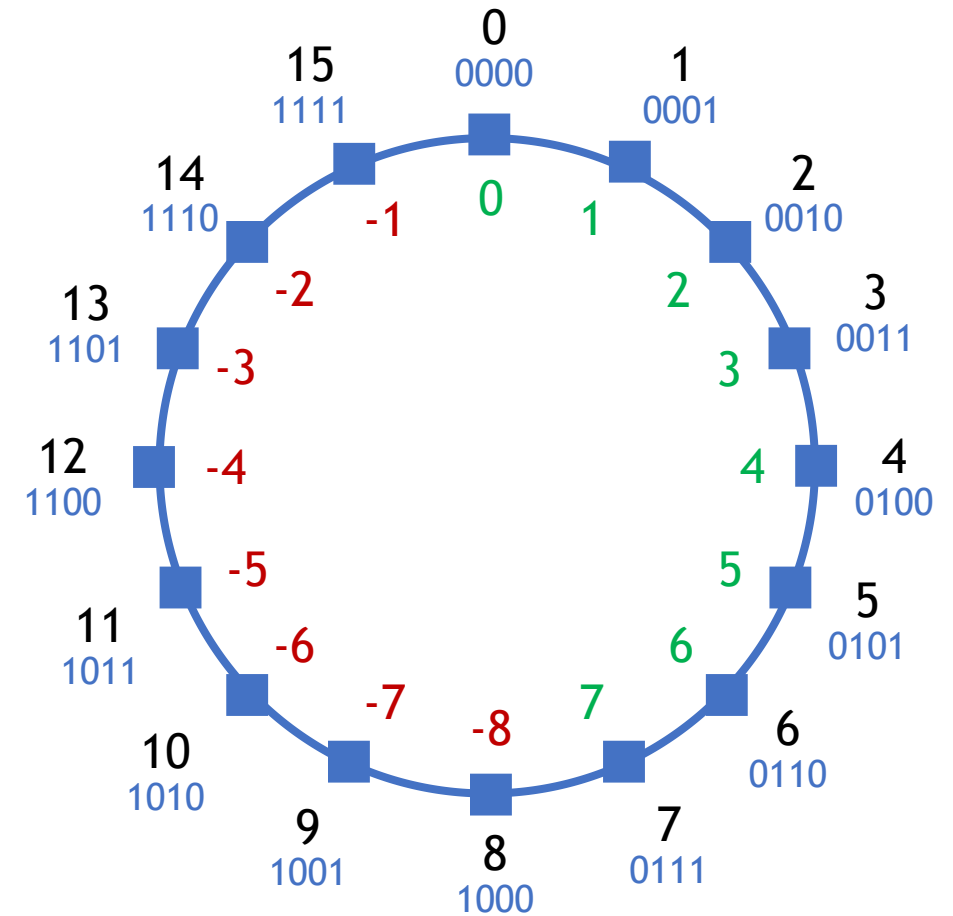
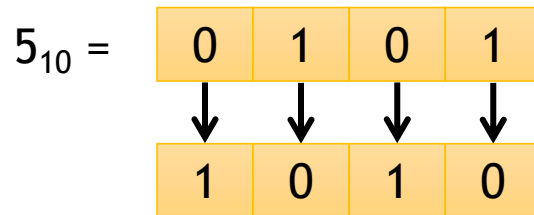
$5_{10} =$

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



Lecture 3 - Data Representation

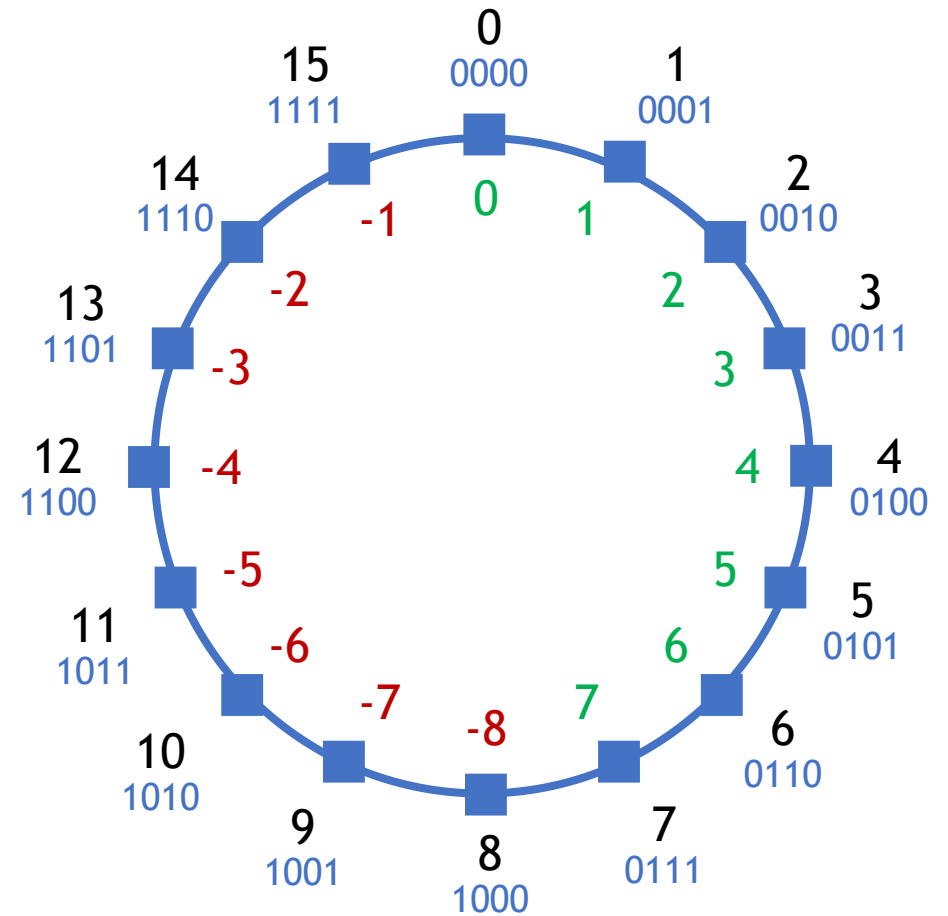
- How to represent integer numbers with sign?
 - two's complement representation
 - Example of signed integer to 2's Complement for -5:



Lecture 3 - Data Representation

- How to represent integer numbers with sign?
 - **two's complement representation**
 - Example of signed integer to 2's Complement for **-5**:

$$\begin{array}{r} 5_{10} = \begin{array}{|c|c|c|c|} \hline 0 & 1 & 0 & 1 \\ \hline \end{array} \\ \downarrow \downarrow \downarrow \downarrow \\ \begin{array}{|c|c|c|c|} \hline 1 & 0 & 1 & 0 \\ \hline \end{array} + \\ \begin{array}{|c|c|c|c|} \hline 0 & 0 & 0 & 1 \\ \hline \end{array} \\ \hline \\ \begin{array}{|c|c|c|c|} \hline 1 & 0 & 1 & 1 \\ \hline \end{array} = -5_{2C} \end{array}$$



Lecture 3 - Data Representation

- **Sum in 2C:**

- Sum is applied directly
- Example: $4 + (-7) = -3$

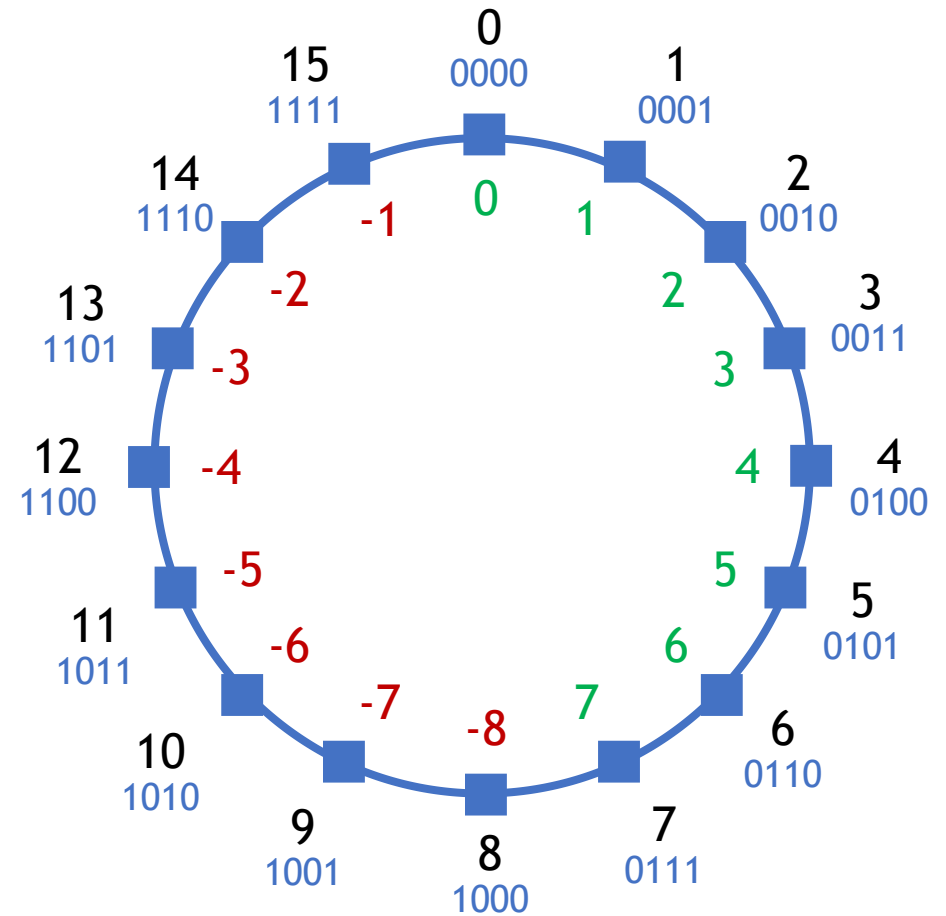
$$\begin{array}{r} 0100 + \\ 1001 = \\ \hline 1101 \end{array}$$

- **Subtraction in 2C:**

- Equivalent to the sum of the additive inverse sign
- inverse obtained computing two's complement (invert all bits and add 1)
- Example: $4 - (-7) = 4 + 7$

- **Overflow in the sum:**

- Never occurs with same-sign addends
- May occur with opposit-sign addends (however, carry out of the MSB is discarded)



Representation of real numbers

Dense numbers and finite representation

Lecture 3 - Data Representation

- How to represent real numbers?

- **Fixed Point**

- allocate a fixed number M of bits to the integer and $M-1$ bits to the fractional parts

0	0	1	1	.	1	1	1	0
---	---	---	---	---	---	---	---	---

$$\begin{array}{ccccccc} & & 2^1 & 2^0 & & 2^{-1} & 2^{-2} & 2^{-3} \\ = & 2 & + & 1 & + & 0.5 & + & 0.25 & + & 0.125 & = & 3.875 \end{array}$$

Always positional

- What for 3.14? → cannot be represented exactly
 - most close approximation is: 0011.0010 = 3.125

Lecture 3 - Data Representation

- How can we increase precision?

$$3.14 \approx \begin{array}{|c|c|c|c|} \hline 0 & 0 & 1 & 1 \\ \hline \end{array} . \begin{array}{|c|c|c|c|} \hline 0 & 0 & 1 & 0 \\ \hline \end{array}$$
$$\begin{array}{ccc} & 2^1 & 2^0 \\ & = 2 & + 1 \end{array} \quad \begin{array}{ccc} & 2^{-3} \\ & + 0.125 \end{array} = 3.125$$

Lecture 3 - Data Representation

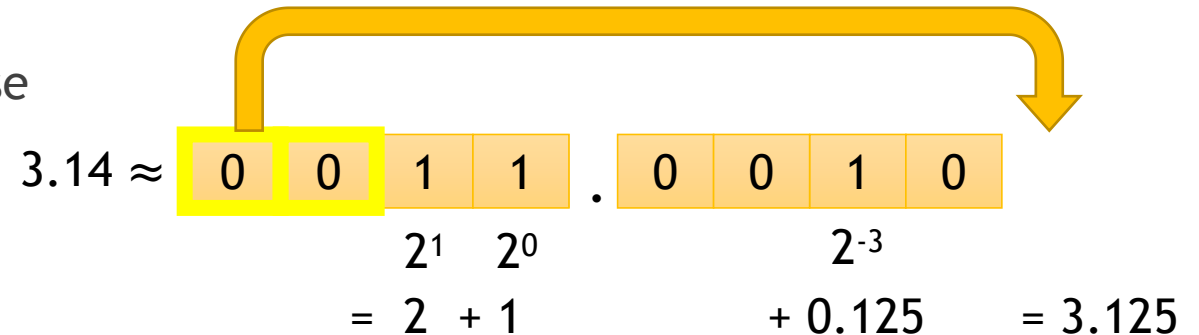
- How can we increase precision?
 - we may have unused positions...

$$3.14 \approx \begin{array}{|c|c|c|c|} \hline 0 & 0 & 1 & 1 \\ \hline \end{array} . \begin{array}{|c|c|c|c|} \hline 0 & 0 & 1 & 0 \\ \hline \end{array}$$

$\begin{array}{cc} 2^1 & 2^0 \\ = 2 & + 1 \end{array} \quad \begin{array}{c} 2^{-3} \\ + 0.125 \end{array} = 3.125$

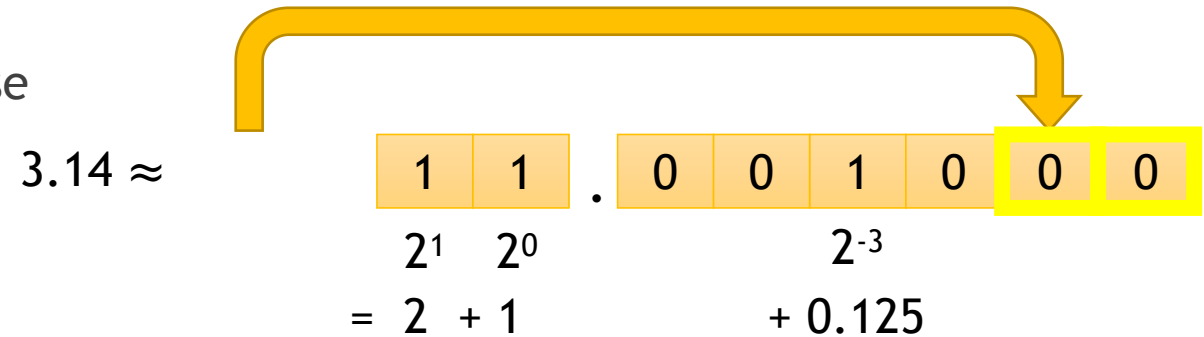
Lecture 3 - Data Representation

- How can we increase precision?
 - we may have unused positions...
 - If we could shift the decimal point, we could use the unused positions to increase precision



Lecture 3 - Data Representation

- How can we increase precision?
 - we may have unused positions...
 - If we could shift the decimal point, we could use the unused positions to increase precision

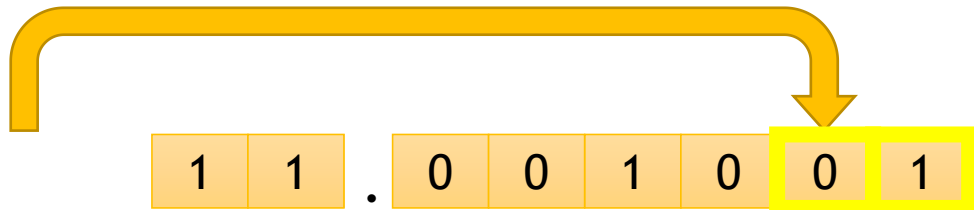


Lecture 3 - Data Representation

- How can we increase precision?
 - we may have unused positions...
 - If we could shift the decimal point, we could use the unused positions to increase precision

3.14 $\approx$

1	1	.	0	0	1	0	0	1
2^1	2^0				2^{-3}			2^{-6}
= 2	+ 1				+ 0.125			+ 0.015625
= 3.140625								



Lecture 3 - Data Representation

- How to represent real numbers?
 - **Floating Point** (float) i.e. exponential notation in binary
 - sign (S) - exponent (E) - significand (mantissa M)

$$X = (-1)^S M \times 2^{E-b}$$

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

S *E* *M*

Lecture 3 - Data Representation

- How to represent real numbers?
 - **Floating Point** (float) i.e. exponential notation in binary
 - sign (S) - exponent (E) - significand (mantissa M)

$$X = (-1)^S M \times 2^{E-b}$$


S E M

fixed (known) bias to avoid writing negative numbers. Just an interpretation.
mapping for example $b = 3$ we have exponent:

000	001	010	011	100	101	110	111
0	1	2	3	4	5	6	7
↓	↓	↓	↓	↓	↓	↓	↓
-3	-2	-1	-0	1	2	3	4

Lecture 3 - Data Representation

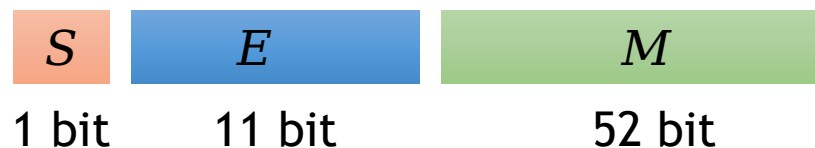
- **Floating Point** (float) i.e. exponential notation in binary

- Format IEEE-754:

- float



- double

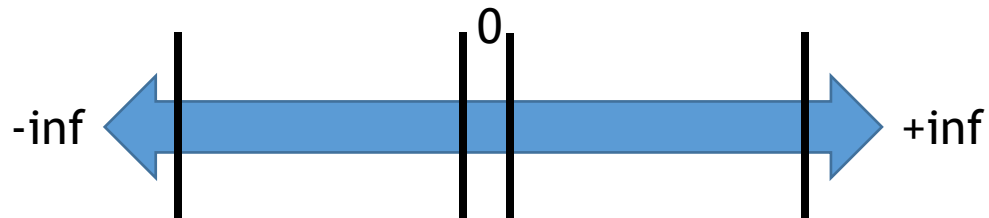


$$X = (-1)^S M \times 2^{E-b}$$

<https://float.exposed/>

Lecture 3 - Data Representation

- The precision limit is still real
 - Some numbers are not represented exactly (not only weird numbers, e.g. 0.1, 0.6)
 - Strong consequences:
 - Floating-point **addition is not associative** → Rounding errors can make the result depend on the order in which operations are performed
 - Min and Max representable values, values too large become “ $\pm\text{inf}$ ” (*overflow*), values too small become “0” (*underflow*)



```
x = 0.1
print(f"{x:.20f}")
# 0.1000000000000000000555
```

```
a = 1e16
b = -1e16
c = 1.0

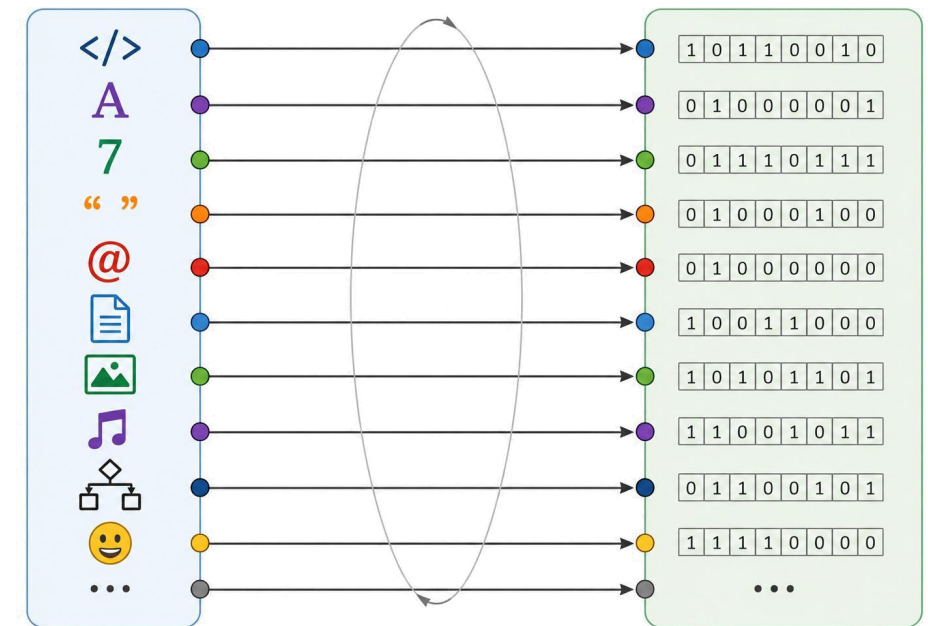
print((a + b) + c)  # 1.0
print(a + (b + c))  # 0.0
```

Representing non-numeric data

From numbers to text, media and much more

Lecture 3 - Data Representation

- A computer can manage only numbers (binary code)
→ to represent general “objects” we need to assign a unique code
 - Code instructions
 - Files
 - Operations...
 - In the end, are represented by a code to be interpreted



Lecture 3 - Data Representation

- How many object can be represented with N bits?
 - N bits $\rightarrow 2^N$ numbers ($0 \dots 2^N - 1$)
 - if we have M objects $\rightarrow$ we need $N = \lceil \log_2 M \rceil$ bits
 - that is, the smallest natural N s.t. $2^N \geq M$

0	0	0
0	0	1
0	1	0
0	1	1

A

B

C

D

1	0	0
1	0	1
1	1	0
1	1	1

E

F

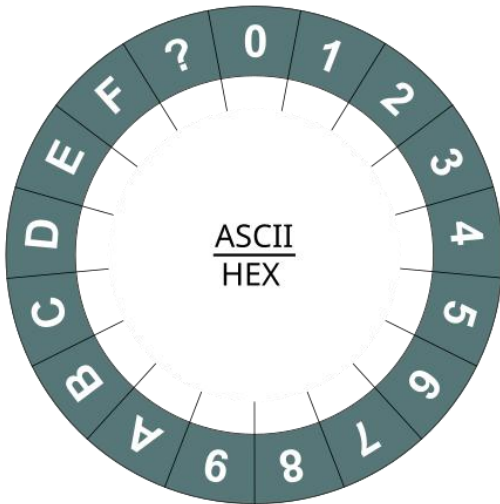
G

H

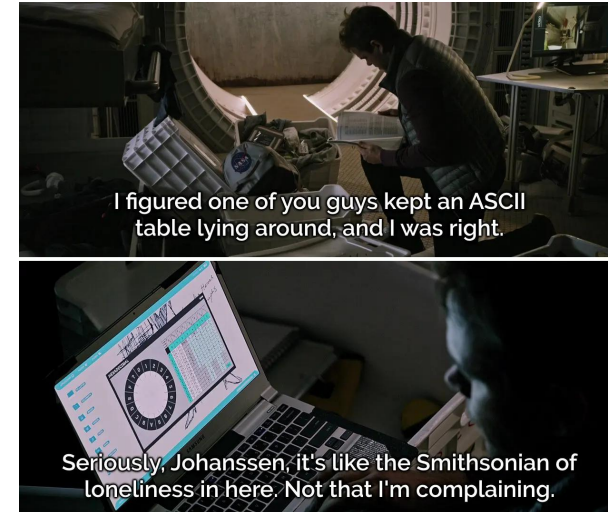
$$N = 3 \leftrightarrow M = 2^3 = 8$$

Lecture 3 - Data Representation

- Encoding of characters
 - ASCII (8 bit)



	0	0	0	0	1	1	1	1
	0	1	0	1	0	1	0	1
	0	1	2	3	4	5	6	7
0	NUL	DLE	SPACE	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EQT	DC4	%	4	D	T	d	t
5	ENQ	NAK	\$	5	E	U	e	u
6	ACK	SYN	%	6	F	V	f	v
7	BEL	ETB	&	7	G	W	g	w
8	BS	CAN	'	8	H	X	h	x
9	HT	EM	(	9	I	Y	i	y
A	LF	SUB	)	:	J	Z	j	z
B	VT	ESC	*	;	K	[	k	(
C	FF	FS	+	←	L	\	l	
D	GR	GS	,	=	M	]	m	)
E	SO	RS	-	→	N	^	n	.
F	SI	US	/	?	O	_	o	del



Lecture 3 - Data Representation

- Encoding of characters

- ASCII – American Standard Code for Information Interchange

- 7 bit → 128 symbols
 - It represents:
 - Letters: A-Z, a-z
 - Digits: 0-9
 - Punctuation and symbols: !, ?, @, #, ...
 - Control characters: newline, tab, etc.

- In general written in Hexadecimal:

- 'A' = $41_{16} = 56_{10} = 1000001_2$
 - 'Z' = $5A_{16} = 90_{10} = 1011010_2$

→	0	0	0	0	1	1	1	1
→	0	0	1	1	0	0	1	1
→	0	1	0	1	0	1	0	1
	0	1	2	3	4	5	6	7
0	NUL	DLE	SPACE	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EQT	DC4	%	4	D	T	d	t
5	ENQ	NAK	\$	5	E	U	e	u
6	ACK	SYN	%	6	F	V	f	v
7	BEL	ETB	&	7	G	W	g	w
8	BS	CAN	'	8	H	X	h	x
9	HT	EM	(	9	I	Y	i	y
A	LF	SUB	)	:	J	Z	j	z
B	VT	ESC	*	;	K	[	k	(
C	FF	FS	+	←	L	\	l	
D	GR	GS	,	=	M	]	m	)
E	SO	RS	-	→	N	^	n	.
F	SI	US	/	?	O	_	o	del

Lecture 3 - Data Representation

- Encoding of characters

- Unicode

- Extends ASCII, designed to represent characters from virtually all writing systems
 - Assigns each character a unique code point written as U+XXXX
 - Example: A → U+0041, ã → U+00E3, € → U+20AC
 - Unicode defines which number identifies each character, not directly how it is stored as bytes

- UTF-8

- A widely used encoding of Unicode code points (Python standard)
 - Uses a variable number of bytes per character: 1 (ASCII) to 4 bytes
 - Examples:
 - A → 41
 - ã → C3 A3
 - € → E2 82 AC
 - 💡 → F0 9F 98 80

<https://home.unicode.org/>

<https://symbl.cc/en/unicode-table/>

Lecture 3 - Data Representation

- Recall: ASCII/UTF-8 code is **not** formatted text like in Word
- We can save text in two ways:
 - **Not formatted:** only characters composing the text are saved
 - **Formatted:** requires byte sequences for the aspect (font, spacing...)

Other kind of data

Encoding audio, video and so on

NOT IN THIS COURSE

Lecture 3 - Data Representation

- Encoding audio/video is much more articulated
- We need to represent many dimensions: color, intensity, tone, frequency...
- However, the basic idea is the same: to associate a unique code to an object
- Example (Image):
 - The color of each pixel is encoded with 8 bit for each channel (R, G, B)
 - $\rightarrow 256 \times 256 \times 256 = 16777216$ colors
 - often written in hex: yellow = `#ffff00`

