CPS 188

Computer Programming Fundamentals

Prof. Alex Ufkes

Topic 1.2: Data Representation

Toronto Metropolitan University

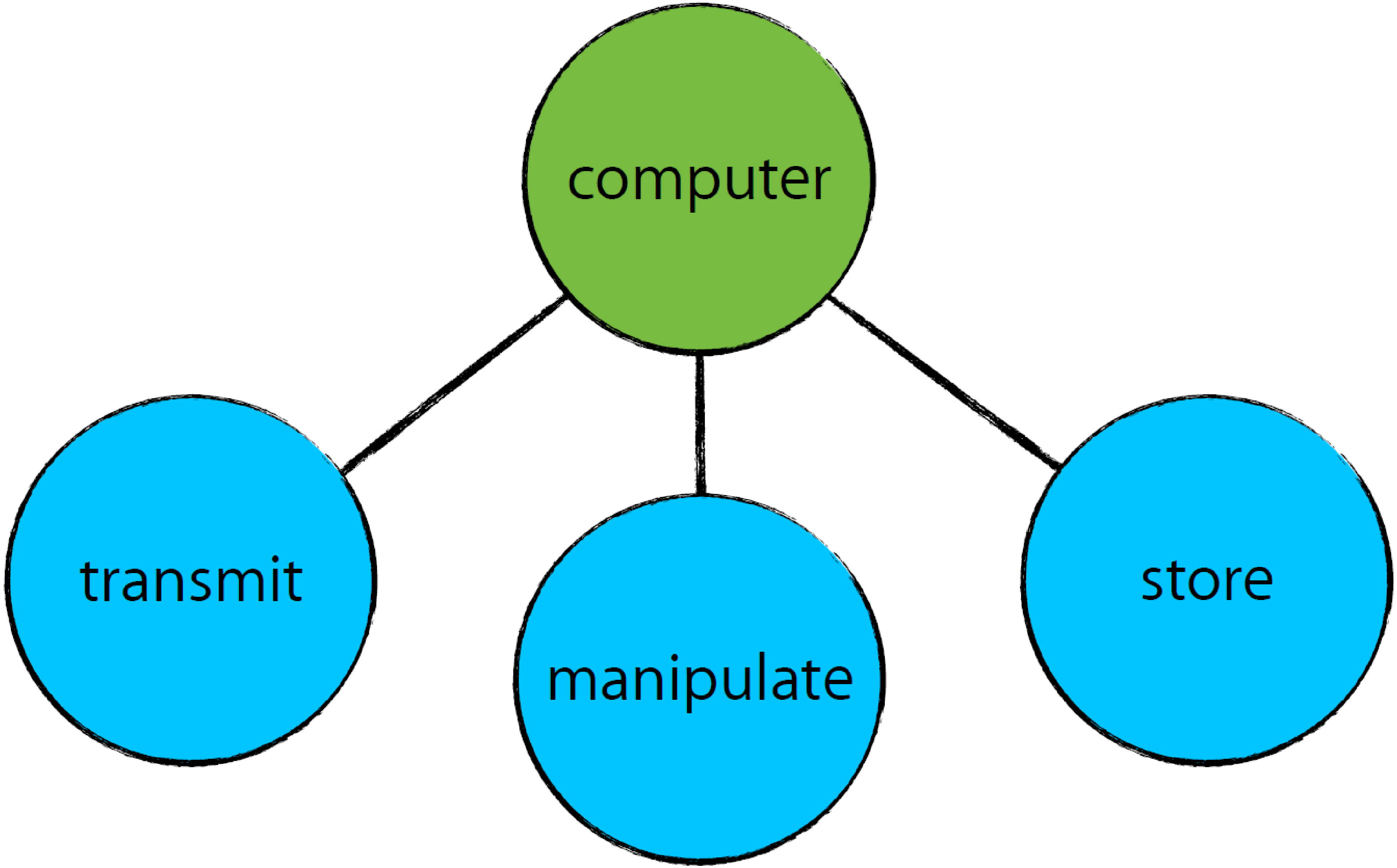
Notice!

Obligatory copyright notice in the age of digital delivery and online classrooms:

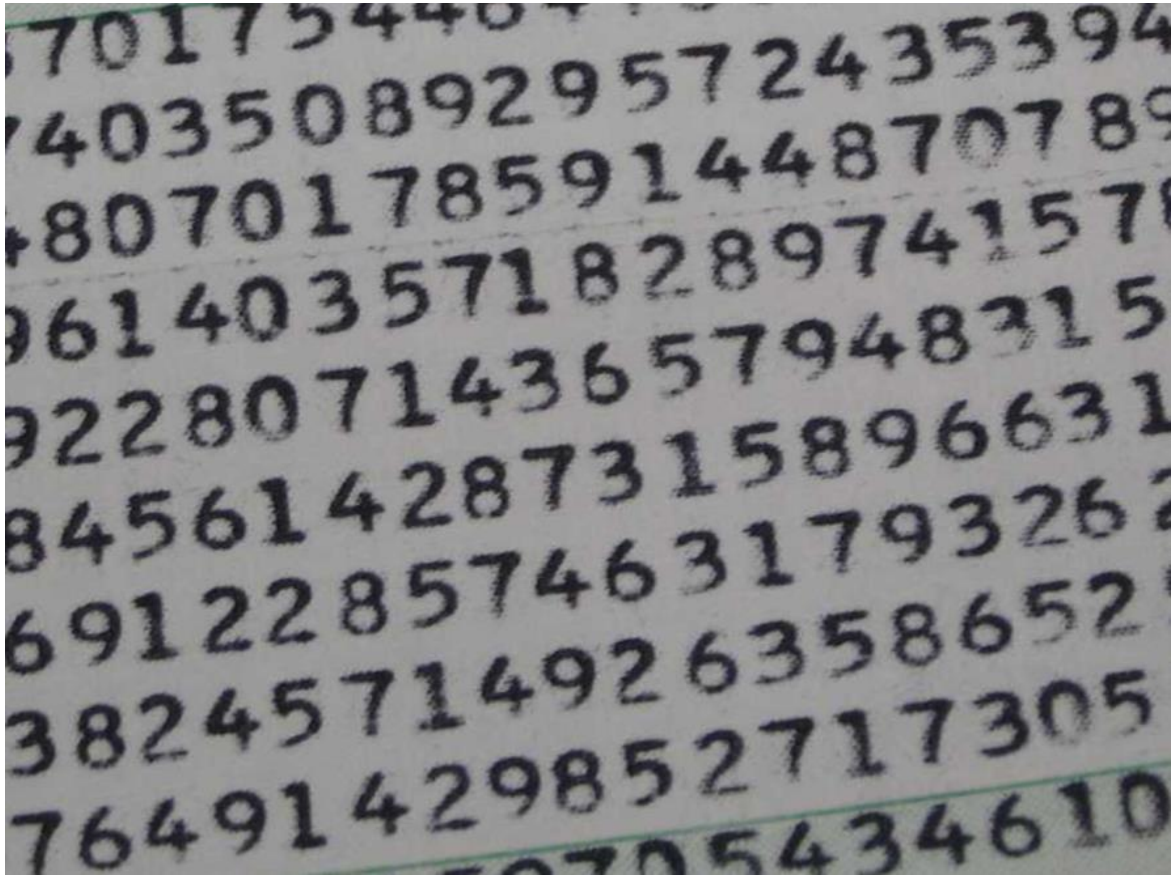
Today

Data Representation:

- Binary: Bits and bytes
- Integers, two’s complement
- Characters
- IEEE-754 Floating point


computer
transmit
manipulate
store

Data Sources



Data Sources

Data Sources

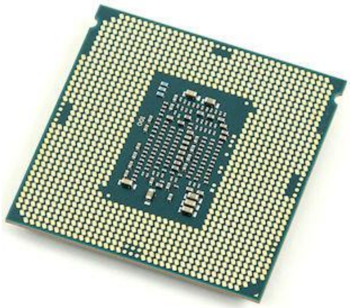
- As far as the physical computer system is concerned, there is no such thing as imagery, audio, text, or even numbers.
- The CPU has no idea what a JPG or MP3 is.
- At the hardware level, we deal exclusively in binary (0, 1)
- Different data types are encoded in different ways.
- A Bitmap image is encoded (in binary) differently than a JPG image, and so on.

(Even 0 and 1 are abstractions over voltage levels – 0V/GND for 0, 1.7/3.3/5.0v for 1, depending on the circuit)

Computer Memory

Volatile

Not volatile



A few general-purpose registers (8-16) + Cache (<100Mb)

Main memory (RAM): R*arely* higher than 64GB. (1GB = 1 billion bytes)

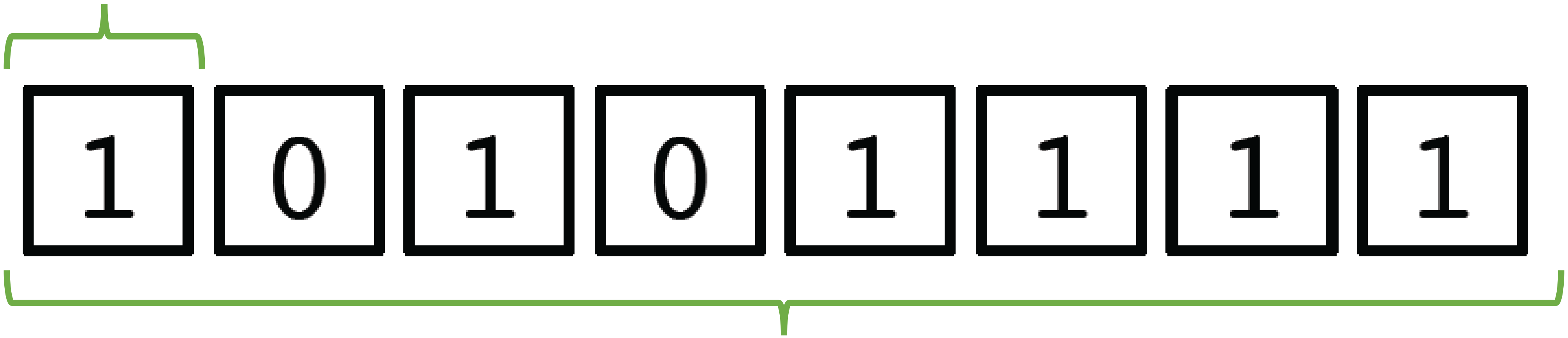
Secondary Storage

Hard drives, SSDs, multiple Terabyte (1TB = 1 trillion bytes)

Wait… What's a *byte*?

Binary: Bits & Bytes

Bit (0 or 1)



Byte (group of 8 bits)

Binary: Words

1 0 1 0 1 1 1 1

- A *word* is a group of bytes, length depends on architecture.
- In a 32-bit system, a word is 32 bits (4 bytes).
- Corresponds to the size of registers on the CPU.

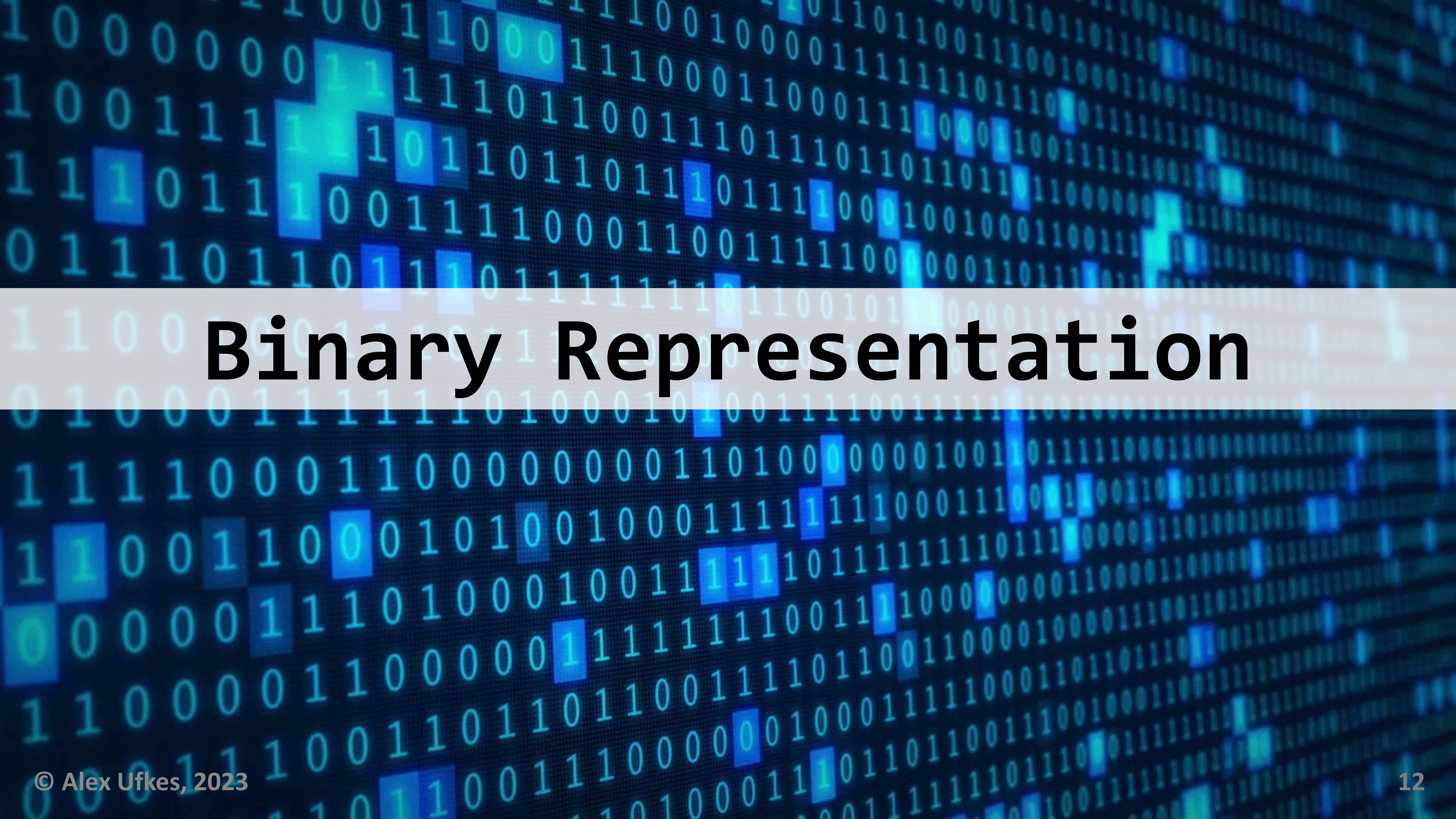
Addressing

Every byte has an address.

Bits are not addressed individually.

Byte address

2052	10100011
2053	00001000
2054	11001100
2055	10001100
2056	10001101
2057	11001101
2058	00101010
2059	10001100



Binary Representation

Binary Representation

- How to represent images or audio in binary are beyond the scope of this course.
- However, it’s easy enough to represent numbers and characters.
- These are the two most important data types for this course.

Whole numbers (integers):

- `0, 1, 2, 3, -76, 896543233,` and so on.

Floating Point numbers:

- `0.0, 3.141592, -7.6, 1.23e4,` and so on.

Characters? Individual letters and symbols:

- `‘A’, ‘x’, ‘Q’, ‘4’, ‘+’, ‘=’, ‘ ’,` etc.

Unsigned Integers

- All data in memory is represented in binary.
- The simplest type of data to understand is the humble unsigned integer.
- Unsigned? Greater than or equal to zero.
- Understand them the same way we understand positive base-10 numbers.

Understanding Binary Integers

Decimal: Base 10 number system we're all used to

10^5	10^4	10^3	10^2	10^1	10^0
5	5	1	9	3	2

551932_{10}

Indicates base

10^5	10^4	10^3	10^2	10^1	10^0	
5	5	1	9	3	$\boxed{2}$	$_{10}$

$2 \times 10^0 = \quad 2$

10^5	10^4	10^3	10^2	10^1	10^0	
$\boxed{5}$	$\boxed{5}$	$\boxed{1}$	$\boxed{9}$	$\boxed{3}$	$\boxed{2}$	$_{10}$

$$2 \times 10^0 = 2$$

$$3 \times 10^1 = 30$$

10^5	10^4	10^3	10^2	10^1	10^0	
5	5	1	$\boxed{9}$	3	2	$_{10}$

$$2 \times 10^0 = 2$$

$$3 \times 10^1 = 30$$

$$9 \times 10^2 = 900$$

10^5	10^4	10^3	10^2	10^1	10^0	
5	5	1	9	3	2	$_{10}$

$$
\begin{aligned}
2 \times 10^0 &= 2 \\
3 \times 10^1 &= 30 \\
9 \times 10^2 &= 900 \\
1 \times 10^3 &= 1000
\end{aligned}
$$

10^5	10^4	10^3	10^2	10^1	10^0	
5	5	1	9	3	2	$_{10}$

$$2 \times 10^0 = 2$$

$$3 \times 10^1 = 30$$

$$9 \times 10^2 = 900$$

$$1 \times 10^3 = 1000$$

$$5 \times 10^4 = 50000$$

10^5	10^4	10^3	10^2	10^1	10^0	
$\boxed{5}$	$\boxed{5}$	$\boxed{1}$	$\boxed{9}$	$\boxed{3}$	$\boxed{2}$	$_{10}$

$$
\begin{array}{lr}
2 \times 10^0 = & 2 \\
3 \times 10^1 = & 30 \\
9 \times 10^2 = & 900 \\
1 \times 10^3 = & 1000 \\
5 \times 10^4 = & 50000 \\
5 \times 10^5 = & 500000
\end{array}
$$

$$10^5 \quad 10^4 \quad 10^3 \quad 10^2 \quad 10^1 \quad 10^0$$

$$\boxed{5}\,\boxed{5}\,\boxed{1}\,\boxed{9}\,\boxed{3}\,\boxed{2}_{10}$$

$$
\begin{aligned}
2 \times 10^0 &= 2 \\
3 \times 10^1 &= 30 \\
9 \times 10^2 &= 900 \\
1 \times 10^3 &= 1000 \\
5 \times 10^4 &= 50000 \\
5 \times 10^5 &= \underline{500000} \\
& 551932
\end{aligned}
$$

Binary-to-decimal

Binary: Base **2** number system

2^5	2^4	2^3	2^2	2^1	2^0
1	0	0	1	0	1

$_2$ Indicates base

2^5	2^4	2^3	2^2	2^1	2^0	
1	0	0	1	0	$\boxed{1}$	$_2$

$1 \times 2^0 = \quad 1$

2^5	2^4	2^3	2^2	2^1	2^0	
1	0	0	1	$\boxed{0}$	1	$_2$

$1 \times 2^0 = \quad 1$

$0 \times 2^1 = \quad 0$

$$
\begin{array}{cccccc}
2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\
\boxed{1} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{1}
\end{array}_2
$$

$1 \times 2^0 = 1$
$0 \times 2^1 = 0$
$1 \times 2^2 = 4$

$$\begin{array}{cccccc} 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \boxed{1} & \boxed{0} & \boxed{0} & \boxed{1} & \boxed{0} & \boxed{1} \end{array}_2$$

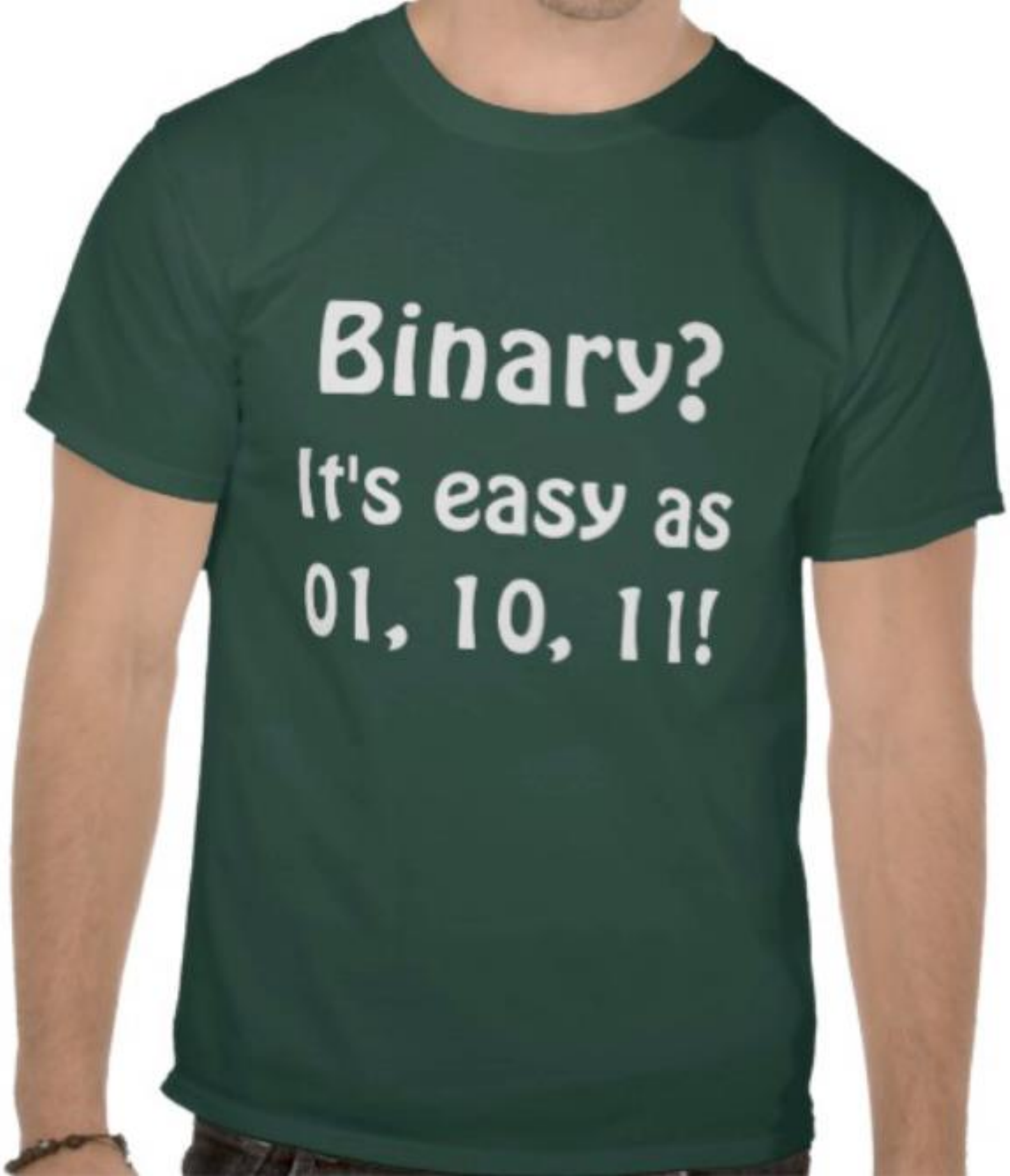
$1 \times 2^0 = \quad 1$
$0 \times 2^1 = \quad 0$
$1 \times 2^2 = \quad 4$
$0 \times 2^3 = \quad 0$

2^5	2^4	2^3	2^2	2^1	2^0	
1	$\boxed{0}$	0	1	0	1	$_2$

$$1 \times 2^0 = 1$$
$$0 \times 2^1 = 0$$
$$1 \times 2^2 = 4$$
$$0 \times 2^3 = 0$$
$$0 \times 2^4 = 0$$

2^5	2^4	2^3	2^2	2^1	2^0	
1	0	0	1	0	1	$_2$

$$
\begin{aligned}
1 \times 2^0 &= 1\\
0 \times 2^1 &= 0\\
1 \times 2^2 &= 4\\
0 \times 2^3 &= 0\\
0 \times 2^4 &= 0\\
1 \times 2^5 &= \underline{32}\\
& 37
\end{aligned}
$$


Binary?
It's easy as
01, 10, 11!

Other Bases: Hexadecimal

In computing:

- Hexadecimal is very common (base-16), as is octal (base-8)
- Decimal goes from 0-9, Hexadecimal goes from 0 to F.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10

Hexadecimal <-> Binary

1. Divide the binary number into sets of 4 bits
2. Convert each set of bits into a hex digit

$$0001000011111001_2$$

$$\underset{1}{0001}\ \underset{0}{0000}\ \underset{F}{1111}\ \underset{9}{1001}_2$$

$10F9_{16}$ or 0x10F9

Octal <-> Binary

- Base-8 (oct = 8, octagon), digits go from 0-7
- Conversion follows same procedure as Hex, with one difference:
- Divide binary number into groups of 3 bits instead of 4 bits.

$$1010000111101100_2$$

Note leading zeroes!

001	010	000	111	101	100
1	2	0	7	5	4

$$\mathbf{120754_8}$$

Binary: Negative Integers? Characters?

<u>We've seen:</u>

- Number systems (decimal, binary, hex, octal)
- unsigned integer (positive, whole numbers)

2^5	2^4	2^3	2^2	2^1	2^0	
1	0	0	1	0	1	$_2$

<u>Next we will look at:</u>

- Signed integer (whole number, can be positive or negative)
- Floating point (real numbers, positive or negative)
- Characters ('A', 'b', '=‘, '#', any <u>single</u> character)

Integers

- In practice, integers are 8, 16, 32, or 64 bits.
- The most common in this course is 32-bit

2^5	2^4	2^3	2^2	2^1	2^0	
1	0	0	1	0	1	$_2$

Only 6 bits here
How to convert to 32?

Add leading zeroes: 00000000000000000000000000100101

In base 10, 578 is the same number as 000000578. Same applies in binary.

Signed Integers

Two options:

1. Sign-and-Magnitude
2. Two's Complement

Sign and magnitude:

- Use the left-most bit as a sign bit
- 1 for negative, 0 for positive

Major Drawback? Two representations for zero:

`+0 = 0000 0000`
`-0 = 1000 0000`

Two's Complement

1) Write the positive number in binary. If the number *is* positive, stop here.
2) If the number is negative, Invert the bits (1s becomes 0s, 0s become 1s)
3) Add 1 (add 1 to the **VALUE** of the number, do **NOT** *append* 1)

Write -9 in two's complement format.

+9:	0	0	0	0	1	0	0	1
invert :	1	1	1	1	0	1	1	0
–9:	1	1	1	1	0	1	1	1

Two's Complement

To convert a two's complement binary number back to decimal, simply perform the process again in the same order:

	$\boxed{1}$ 1 1 1 0 1 1 1	Negative! first bit is 1
Invert:	0 0 0 0 1 0 0 0	
Add 1:	0 0 0 0 0 0 0 1	
	0 0 0 0 1 0 0 1	

At this point, convert to decimal using previously described method.
But remember! The final number is negative because the first bit was 1.

Integer to Binary

Example: Convert -138 into binary

- Number is negative, so we need to perform two's comp operation.
- First though, we just convert 138 to binary.

1. Find the highest power of two less than or equal to 138
   - It's 128, or 2^7. Thus, the bit in position 7 is 1

		1							
2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Integer to Binary

Example: Convert -138 into binary

1. Find the highest power of two less than or equal to 138
   - It's 128, or 2^7. Thus, the bit in position 7 is 1
2. Subtract 128 from 138, 10 remains. Repeat 1.
   - Highest power of two less than or equal to 10 is 8, or 2^3
   - The bit in position 3 is 1, positions >3 are zero.

		1	0	0	0	1			
2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Integer to Binary

Example: Convert -138 into binary

1. Find the highest power of two less than or equal to 138
   - It's 128, or 2^7. Thus, the bit in position 7 is 1
2. Subtract highest power of 2 from remainder. Repeat 1.
   - Keep repeating until nothing remains.
3. Since the original number was negative, we apply 2s-comp

0	0	1	0	0	0	1	0	1	0
2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

Integer to Binary

Example: Convert -138 into binary

3. Since the original number was negative, we apply 2s-comp

	0	0	1	0	0	0	1	0	1	0
Invert:	1	1	0	1	1	1	0	1	0	1
Add 1:	1	1	0	1	1	1	0	1	1	0

Integer to Binary

Example: Convert -138 into binary

$$-138_{10} = 1101110110_2$$

To extend this to 32 (or 64) bits, we add leading **1s**

1	1	0	1	1	1	0	1	1	0

Two's Complement: Another Way

Write -148_{10} in two's comp format:

Write the positive number in binary using repeated division by 2:

0	0	1	0	0	1	0	1	0	0

$_2$

148 / 2 = 74	Remainder? NO	9 / 2 = 4.~~5~~	Remainder? YES
74 / 2 = 37	Remainder? NO	4 / 2 = 2	Remainder? NO
37 / 2 = 18.~~5~~	Remainder? YES	2 / 2 = 1	Remainder? NO
18 / 2 = 9	Remainder? NO	1 = 1	1 goes on the left

Two's Complement: Another Way

Write -148_{10} in two's comp format:

Write the positive number in binary using repeated division by 2:

0	0	1	0	0	1	0	1	0	0

Invert, add 1:

`1101101100`

Floating-Point (Real) Numbers

Real numbers are represented using a **sign bit**, a **mantissa**, and an **exponent**.

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}
0	1	0	1	.	1	1	1
8	4	2	1		$^1/_2$	$^1/_4$	$^1/_8$

IEEE-754

Most widely used standard for floating point computation

32-bit:

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

sign bit

Exponent (8 bits)

Mantissa (23 bits)

64-bit:

0 1 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 …

sign bit

Exponent (11 bits)

Mantissa (52 bits)

Decimal to IEEE-754

Learn by example: Convert -9_{10} to 32-bit floating point format.

1) Convert the absolute value of the number to binary

$$\mathbf{9_{10} = 1001.0_2}$$

(Note the added .0)

2) Append exponent indicator:

$$\mathbf{9 = 1001.0 \times 2^0}$$

3) Normalize (increment exponent):

$$\mathbf{9 = 1.\boxed{0010} \times 2^{\boxed{3}}}$$

Decimal moves left, exponent goes up (scientific notation)

Mantissa

Decimal to IEEE-754

4) Add bias to exponent:

   Bias is 127 for 32-bit, **1023** for 64-bit

   From 3) **$9 = 1.0010 \times 2^3$ -> exponent is $3 + 127 = 130$**

5) Convert exponent to binary:

   $$\mathbf{130_{10} = 1\ 0\ 0\ 0\ 0\ 0\ 1\ 0_2}$$

6) Set sign bit (negative), place mantissa and exponent:

$$\underset{\substack{\uparrow \\ \text{sign} \\ \text{bit}}}{1} \quad \underbrace{1\ 0\ 0\ 0\ 0\ 0\ 1\ 0}_{\text{Exponent (8 bits)}} \quad \underbrace{0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{Mantissa (23 bits)}}$$

Another Decimal to IEEE-754

Convert 18.75_{10} to 32-bit floating point format.

2^{-1} 2^{-2} 2^{-3}

. 1 1 1

$^1/_2$ $^1/_4$ $^1/_8$

1) **18.75_{10} = 10010.11_2** Notice the right side of the decimal! Recall:

2-4) Exponent: **$10010.11_2 \times 2^0 = 1.001011_2 \times 2^4$, 4 + 127 = 131**

5) $131_{10} = 1000\ 0011_2$

6) Place components

$\underset{\text{sign bit}}{0}\ \underbrace{1\ 0\ 0\ 0\ 0\ 0\ 1\ 1}_{\text{Exponent (8 bits)}}\ \underbrace{0\ 0\ 1\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0}_{\text{Mantissa (23 bits)}}$

Yet Another Decimal to IEEE-754

Write 9.625_{10} in 32-bit floating point format.

Step 1) Convert absolute value to binary (both sides of decimal!)

1	0	0	1	.	1	0	1	0
8	4	2	1		1/2	1/4	1/8	1/16

0.625	x2	= 1.25
0.25	x2	= 0.5
0.5	x2	= 1.0
0.0	x2	= 0.0

$9.0_{10} = 1001.0_2$

$0.625_{10} = 0.101_2$

Yet Another Decimal to IEEE-754

Write 9.625_{10} in 32-bit floating point format.

Step 2) Append exponent indicator:

$$\mathbf{1001.101 \times 2^0}$$

Step 3) Normalize (increment exponent):

$$1001.101 \times 2^0 = \mathbf{1.001101 \times 2^{\boxed{3}}}$$

Step 4) Add bias to exponent:

$$3 + 127 = 130$$

Yet Another Decimal to IEEE-754

Write 9.625_{10} in 32-bit floating point format.

Step 5) Convert exponent to binary:

$$130_{10} = \boxed{1000\ 0010_2}$$

Step 6) Set sign bit, place mantissa and exponent:

From step 3) $1.\boxed{001101} \times 2^3$

$$\underset{\text{sign bit}}{0}\quad \underbrace{1\,0\,0\,0\,0\,0\,1\,0}_{\text{Exponent (8 bits)}}\quad \underbrace{0\,0\,1\,1\,0\,1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0}_{\text{Mantissa (23 bits)}}$$

Convert Back? IEEE-754 to Decimal

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

Reverse everything:

Convert exponent to decimal, *subtract* bias

$\mathbf{1000\ 0010_2 = 130_{10}}$

$\mathbf{130 - 127 = 3}$

Add leading 1 to mantissa

$(1\ .\ 0\ 0\ 1\ 1\ 0\ 1 \times 2^3)$

Apply exponent.

$(1\ 0\ 0\ 1\ .\ 1\ 0\ 1 \times 2^0)$

Convert both sides back to decimal

$\mathbf{1001_2 = 9_{10}}$

$\mathbf{0.101_2 = 0.625_{10}}$

Characters

Expressed using the ASCII table

ASCII Table

NUL ^@ 0	SOH ^A 1	STX ^B 2	ETX ^C 3	EOT ^D 4	ENQ ^E 5	ACK ^F 6	BEL ^G 7	BS ^H 8	TAB ^I 9	LF ^J 10	VT ^K 11	FF ^L 12	CR ^M 13	SO ^N 14	SI ^O 15
DLE ^P 16	DC1 ^Q 17	DC2 ^R 18	DC3 ^S 19	DC4 ^T 20	NAK ^U 21	SYN ^V 22	ETB ^W 23	CAN ^X 24	EM ^Y 25	SUB ^Z 26	ESC ^[27	FS ^\ 28	GS ^] 29	RS ^^ 30	US ^? 31
32	! 33	" 34	# 35	$ 36	% 37	& 38	' 39	(40	) 41	* 42	+ 43	, 44	- 45	. 46	/ 47
0 48	1 49	2 50	3 51	4 52	5 53	6 54	7 55	8 56	9 57	: 58	; 59	< 60	= 61	> 62	? 63
@ 64	A 65	B 66	C 67	D 68	E 69	F 70	G 71	H 72	I 73	J 74	K 75	L 76	M 77	N 78	O 79
P 80	Q 81	R 82	S 83	T 84	U 85	V 86	W 87	X 88	Y 89	Z 90	[91	\ 92	] 93	^ 94	_ 95
` 96	a 97	b 98	c 99	d 100	e 101	f 102	g 103	h 104	i 105	j 106	k 107	l 108	m 109	n 110	o 111
p 112	q 113	r 114	s 115	t 116	u 117	v 118	w 119	x 120	y 121	z 122	{ 123	| 124	} 125	~ 126	DEL 127

Characters are 1 byte (8 bits) each

```
'A' = 65 = 0100 0001
'a' = 97 = 0110 0001
'$' = 36 = 0010 0100
'1' = 49 = 0011 0001
```

NUL ^@ 0	SOH ^A 1	STX ^B 2	ETX ^C 3	EOT ^D 4	ENQ ^E 5	ACK ^F 6
DLE ^P 16	DC1 ^Q 17	DC2 ^R 18	DC3 ^S 19	DC4 ^T 20	NAK ^U 21	SYN ^V 22
32	! 33	" 34	# 35	$ 36	% 37	& 38
0 48	1 49	2 50	3 51	4 52	5 53	6 54
@ 64	A 65	B 66	C 67	D 68	E 69	F 70
P 80	Q 81	R 82	S 83	T 84	U 85	V 86
` 96	a 97	b 98	c 99	d 100	e 101	f 102
p 112	q 113	r 114	s 115	t 116	u 117	v 118

Notice: The character ‘1’ does not have the ASCII value of 1. Rather, its value is 49. The difference between the integer 1 and the character ‘1’ is **VERY** important!

Questions?