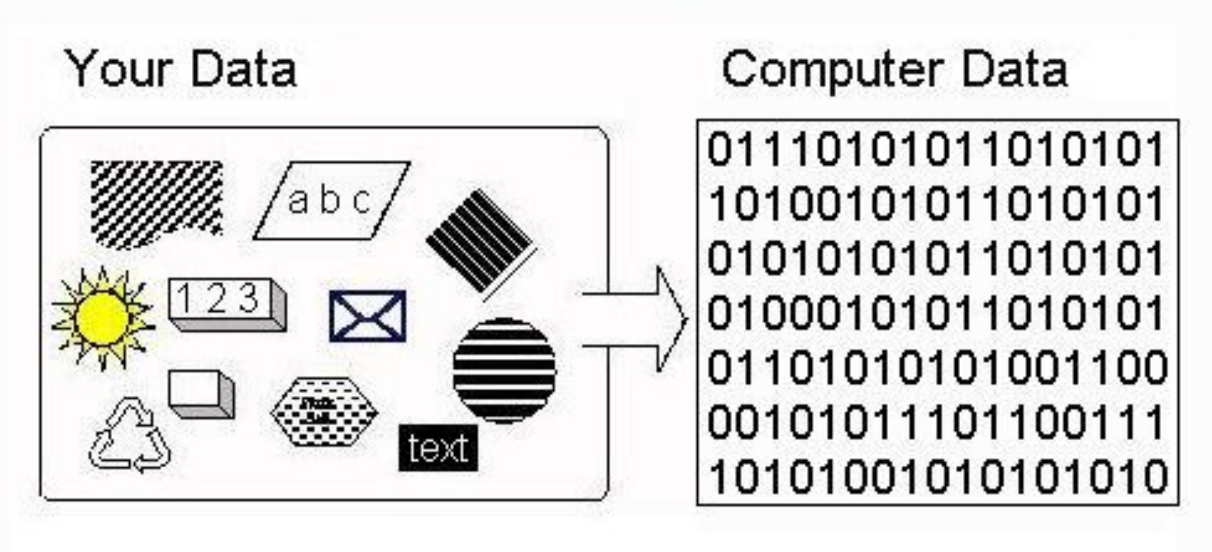
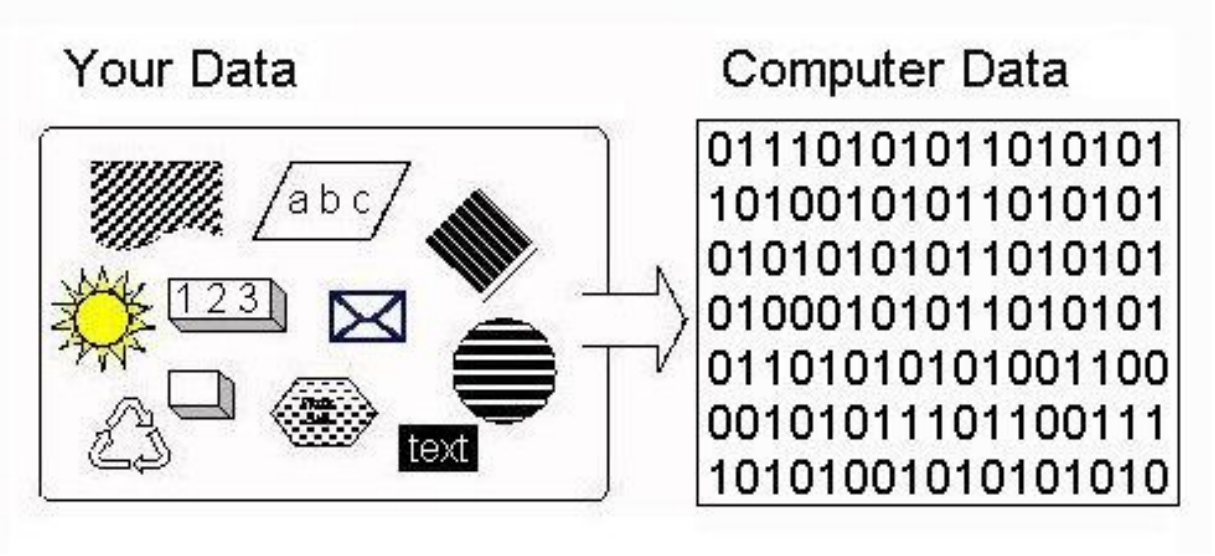
COSC 122: Computer Fluency

How do Computers Represent Data?



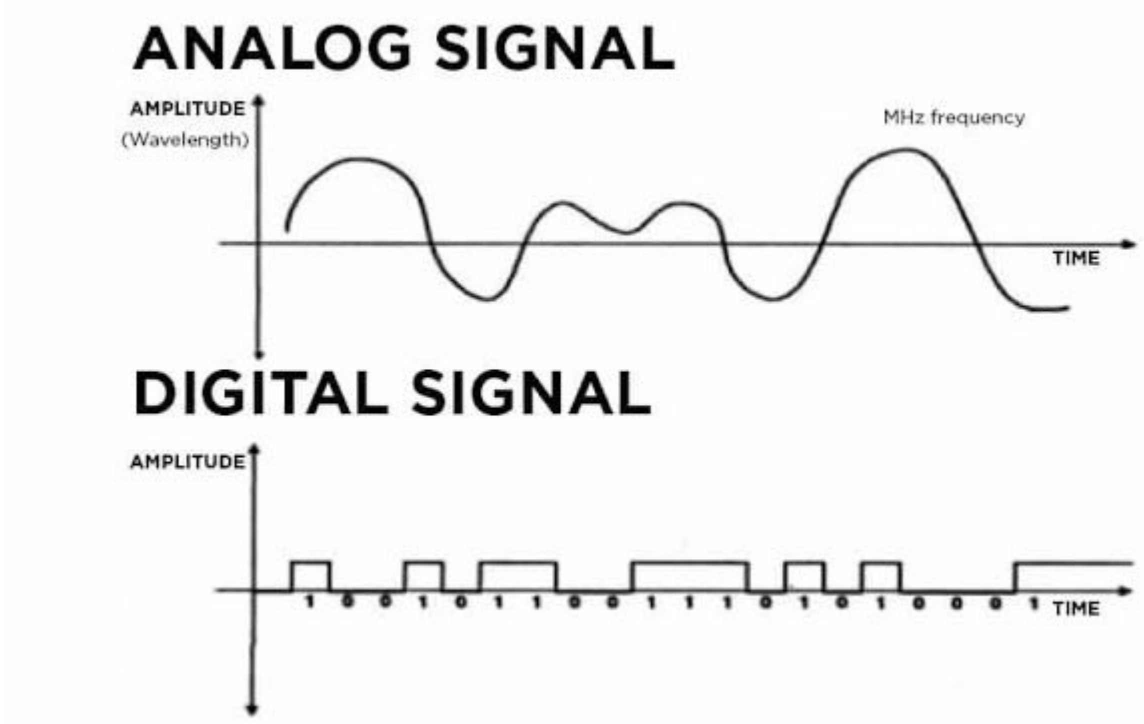
How do Computers Represent Data?



analog data

digital (binary) data

Visual Difference

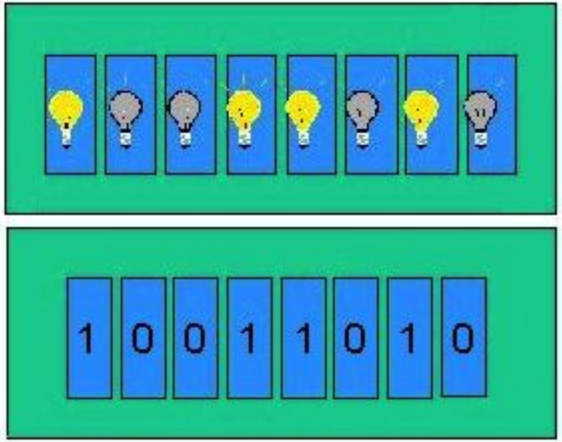


continuous

discrete

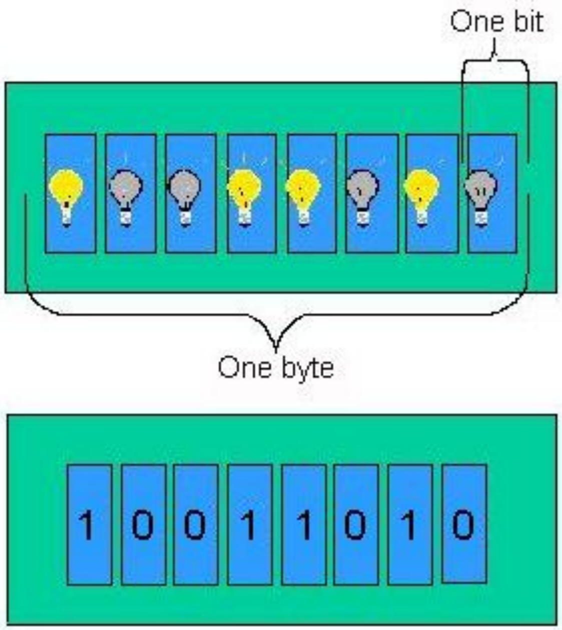
Why Binary?

- Computers use a binary system (two digits: 0 and 1)
- System is ideal because it can be physically represented by electrical states
    - Transistors act as switches
    - An "on" transistor represents a 1, allowing electrical current to flow.
    - An "off" transistor represents a 0, blocking the electrical current.



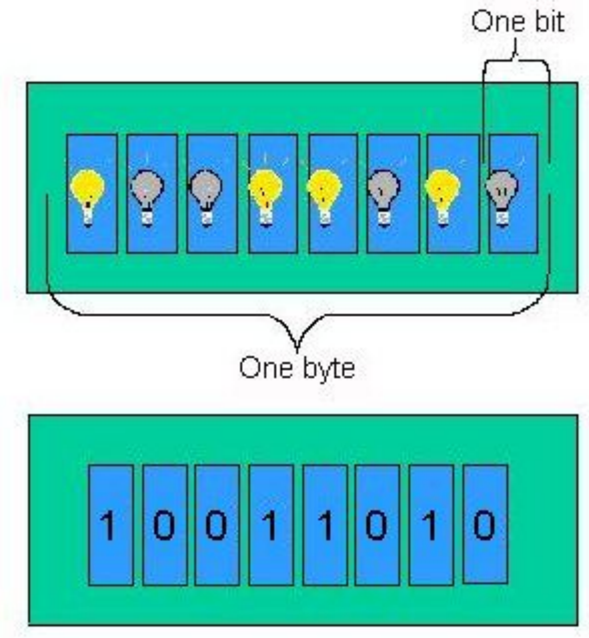
Binary Representation

- Combinations of these binary digits are used to represent more complex information
- Each digit is called a bit
- Can we count beyond 2?
    - 0 → 0
    - 1 → 1
    - 2 → ?



Binary Representation

- Combinations of these binary digits are used to represent more complex information
- Each digit is called a bit
- Can we count beyond 2?
  - 0 → 0
  - 1 → 1
  - 2 → ?
- Put more digits together (next slide)
- Each chunk of 8 digits is called a byte



Decimal to Binary

- Our number system is called decimal (base 10)
  - Each digit (position in a number) represents a power of 10
  - E.g. 345 = (3 x 100) + (4 x 10) + (5 x 1)
    = (3 x 10^2) + (4 x 10^1) + (5 x 10^0)
  - Larger number corresponds to multiplying 10 with a higher exponent

…

Decimal to Binary

- Our number system is called decimal (base 10)
  - Each digit (position in a number) represents a power of 10
  - E.g. 345 = (3 x 100) + (4 x 10) + (5 x 1)
    = $(3 \times 10^2) + (4 \times 10^1) + (5 \times 10^0)$
  - Larger number corresponds to multiplying 10 with a higher exponent
- What about binary numbers?
  - E.g. $0 = (0 \times 2^0) = 0 \times 1 \rightarrow 0$
    $1 = (1 \times 2^0) = 1 \times 1 \rightarrow 1$

Decimal to Binary

- Our number system is called decimal (base 10)
  - Each digit (position in a number) represents a power of 10
  - E.g. $345 = (3 \times 100) + (4 \times 10) + (5 \times 1)$
    $= (3 \times 10^2) + (4 \times 10^1) + (5 \times 10^0)$
  - Larger number corresponds to multiplying 10 with a higher exponent
- What about binary numbers?
  - E.g. $0 = (0 \times 2^0) = 0 \times 1 \rightarrow 0$
    $1 = (1 \times 2^0) = 1 \times 1 \rightarrow 1$
    $10 = (1 \times 2^1) + (0 \times 2^0) = 2 + 0 \rightarrow 2$

Decimal to Binary

- Our number system is called decimal (base 10)
  - Each digit (position in a number) represents a power of 10
  - E.g. $345 = (3 \times 100) + (4 \times 10) + (5 \times 1)$
    $= (3 \times 10^2) + (4 \times 10^1) + (5 \times 10^0)$
  - Larger number corresponds to multiplying 10 with a higher exponent
- What about binary numbers?
  - E.g. $0 = (0 \times 2^0) = 0 \times 1 \rightarrow 0$
    $1 = (1 \times 2^0) = 1 \times 1 \rightarrow 1$
    $10 = (1 \times 2^1) + (0 \times 2^0) = 2 + 0 \rightarrow 2$
    $11 = (1 \times 2^1) + (1 \times 2^0) = 2 + 1 \rightarrow 3$

Decimal to Binary

- Our number system is called decimal (base 10)
  - Each digit (position in a number) represents a power of 10
  - E.g. 345 = (3 x 100) + (4 x 10) + (5 x 1)
    $= (3 \times 10^2) + (4 \times 10^1) + (5 \times 10^0)$
  - Larger number corresponds to multiplying 10 with a higher exponent
- What about binary numbers?
  - E.g. $0 = (0 \times 2^0) = 0 \times 1 \rightarrow 0$
    $1 = (1 \times 2^0) = 1 \times 1 \rightarrow 1$
    $10 = (1 \times 2^1) + (0 \times 2^0) = 2 + 0 \rightarrow 2$
    $11 = (1 \times 2^1) + (1 \times 2^0) = 2 + 1 \rightarrow 3$
    $100 = (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) = 4 + 0 + 0 \rightarrow 4$
    …

Common Numbers

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111

Decimal	Binary
8	1000
10	1010
20	10100
40	101000
80	1010000
100	1100100
1000	1111101000
…	

How to Convert?

- **Exercise: Given 1011, what is this number in decimal?**

How to Convert?

- **Exercise: Given 1011, what is this number in decimal?**
- Answer: 11
- Show your work:

  $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 2 + 1 = 11$

How to Convert?

- **Exercise: Given 1011, what is this number in decimal?**
- Answer: 11
- Show your work:

  $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 2 + 1 = 11$
- **Exercise: Given 00101010, what is this number in decimal?**

How to Convert?

- **Exercise: Given 1011, what is this number in decimal?**
- Answer: 11
- Show your work:

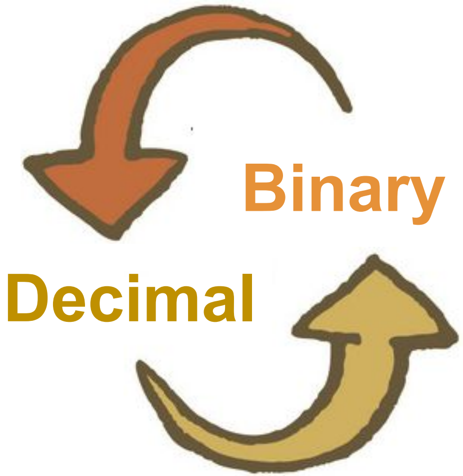
  $1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 8 + 2 + 1 = 11$
- **Exercise: Given 00101010, what is this number in decimal?**
- Answer: 42
- Show your work:

  $0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$

  $= 1 \times 2^5 + 1 \times 2^3 + 1 \times 2^1$

  $= 32 + 8 + 2 = 42$

Conversion Rule from Binary to Decimal



- Identify the binary number to convert
- Assign powers of 2 from right to left
  - Rightmost digit is 2^0
  - The next digit to the left is 2^1
  - The next digit to the left is 2^2
  - etc.
- Multiply each digit of the binary number with the corresponding power of 2
- Sum up the results

The Other Direction?

- **Exercise: Given 13, what is this number in binary?**

The Other Direction?

- **Exercise: Given 13, what is this number in binary?**
- Answer: 1101
- Show your work:

  13 ÷ 2 = 6 remainder 1

  6 ÷ 2 = 3 remainder 0

  3 ÷ 2 = 1 remainder 1

  1 ÷ 2 = 0 remainder 1

  Write remainders in reverse: 1101
- Check: $1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 8 + 4 + 1 = 13$

The Other Direction?

- **Exercise: Given 13, what is this number in binary?**
- Answer: 1101
- Show your work:

  13 ÷ 2 = 6 remainder 1

  6 ÷ 2 = 3 remainder 0

  3 ÷ 2 = 1 remainder 1

  1 ÷ 2 = 0 remainder 1

  Write remainders in reverse: 1101
- Check: $1 \times 2^3 + 1 \times 2^2 + 1 \times 2^0 = 8 + 4 + 1 = 13$

Another Example

- **Exercise: Given 123, what is this number in binary?**

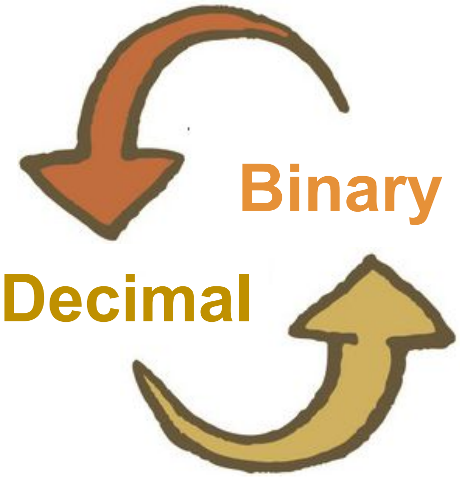
Another Example

- **Exercise: Given 123, what is this number in binary?**
- Answer: 111 1011
- Show your work:
  123 ÷ 2 = 61 remainder 1
  61 ÷ 2 = 30 remainder 1
  30 ÷ 2 = 15 remainder 0
  15 ÷ 2 = 7 remainder 1
  7 ÷ 2 = 3 remainder 1
  3 ÷ 2 = 1 remainder 1
  1 ÷ 2 = 0 remainder 1
  Write remainders in reverse: 1111011

Another Example

- **Exercise: Given 123, what is this number in binary?**
- Answer: 111 1011
- Show your work:
  123 ÷ 2 = 61 remainder 1
  61 ÷ 2 = 30 remainder 1
  30 ÷ 2 = 15 remainder 0
  15 ÷ 2 = 7 remainder 1
  7 ÷ 2 = 3 remainder 1
  3 ÷ 2 = 1 remainder 1
  1 ÷ 2 = 0 remainder 1
  Write remainders in reverse: 1111011
- Check: $1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^1 + 1 \times 2^0$
  = 64 + 32 + +16 + 8 + 2 + 1 = 123

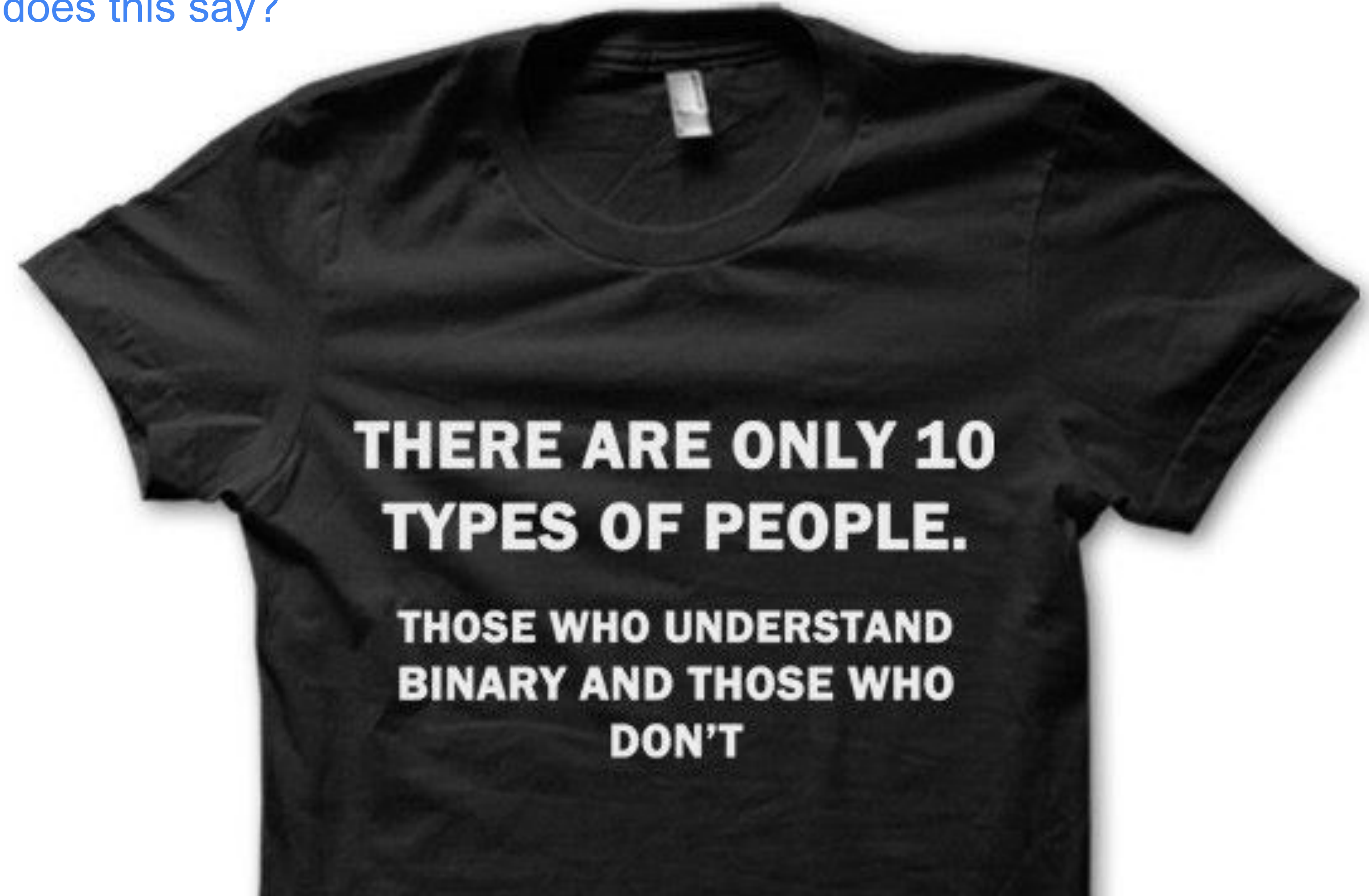
Conversion Rule from Binary to Decimal



- Identify the decimal number to convert
- Repeatedly divide the number by 2 until answer is 0
- Keep track the remainder each time (0 or 1)
- Write the remainders in reverse

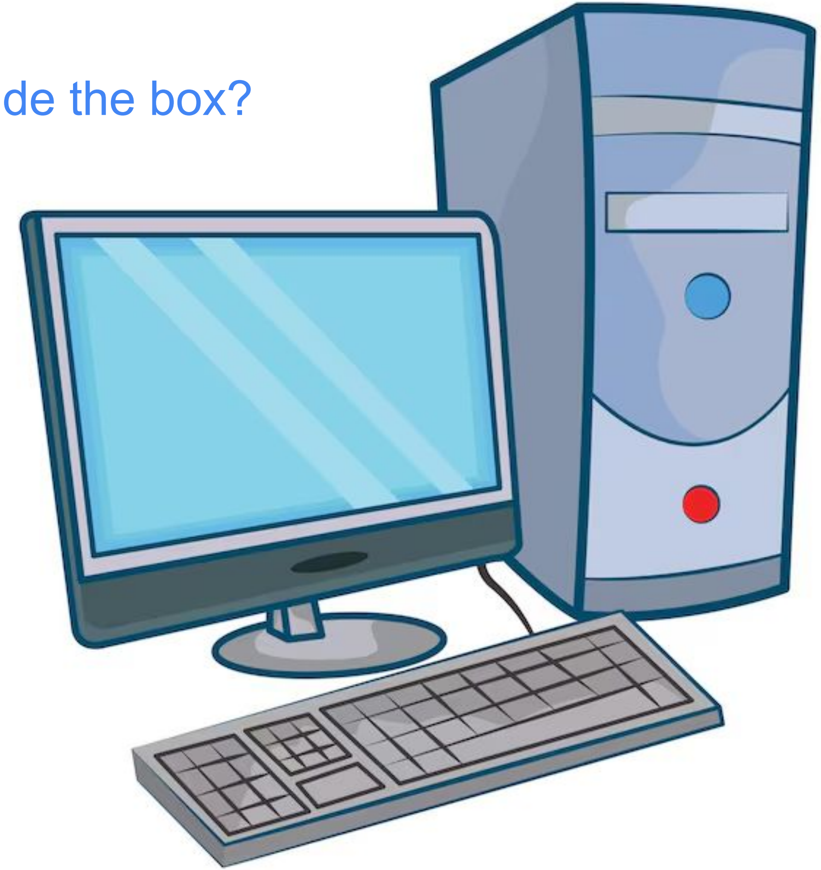
What does this say?

What does this say?

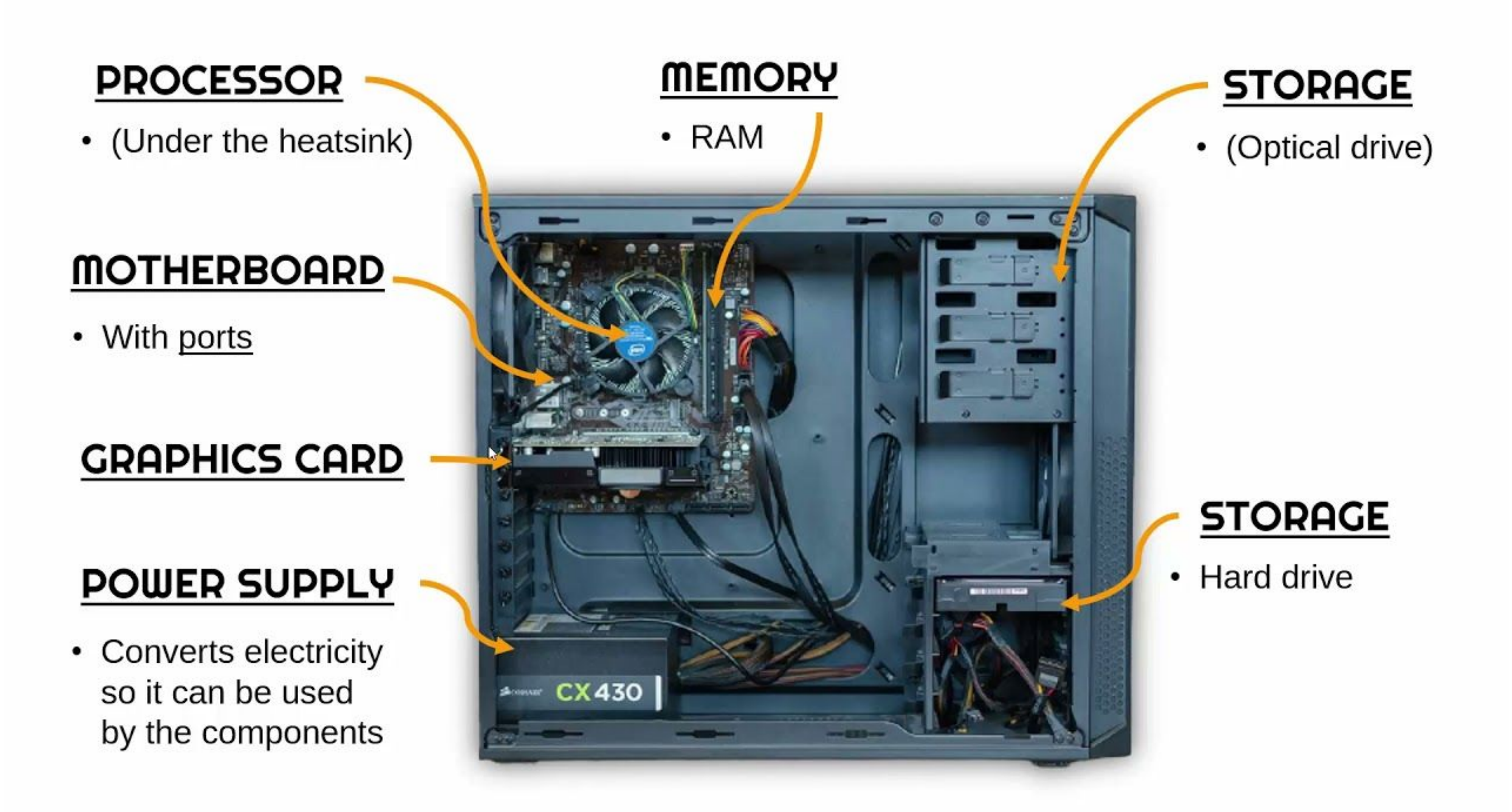


Taking a Step Back …

What's inside the box?



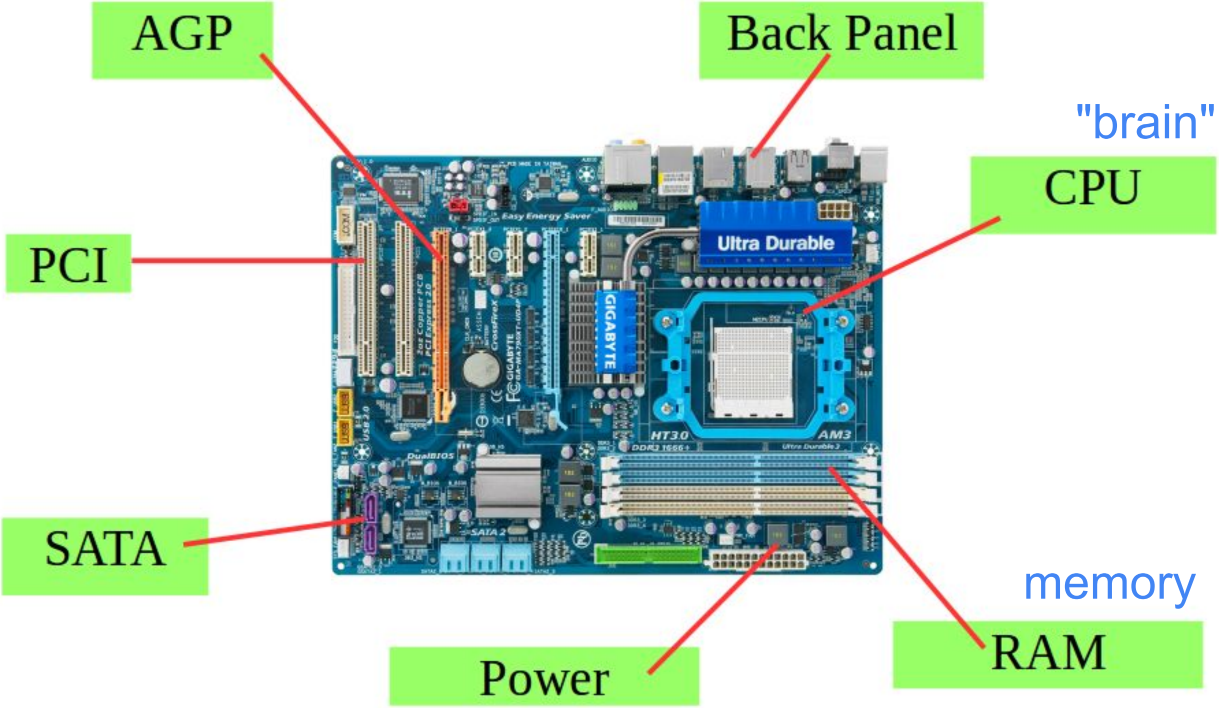
What's Inside the Computer Box?



Zooming into the Motherboard

Connects to monitor, speakers, keyboard, mouse

various slots to connect to other hardware

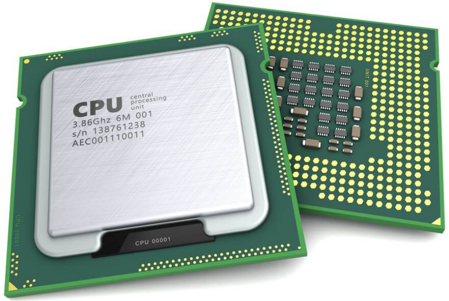


"brain"

memory

provides power to connected component

Using the Numbers



- CPU (central processing unit) is the computer's brain
- Basic operations:
  - Arithmetic (add, subtract, multiply, divide)
  - Logical operations (and, or, not, xor)
  - Data movement (load, store, move)
  - Bit shifting and rotation (shift left, shift right, wrap around)
  - Comparisons (equal, less than, greater than)
  - Control flows (jump/branch, call/return)
- Fundamental concepts to all programming

Adding Binary Numbers

- Works the same way as addition with decimal numbers
- Example:

```
  1 0 0 1 0 1 1 1
+ 0 1 1 0 0 1 1 0
_________________
```

- Add one digit at a time from right to left
- If adding two values is bigger than the base max value, carry 1 to next digit

Adding Binary Numbers

- Works the same way as addition with decimal numbers
- Example:

```
        1 1           (carries)
  1 0 0 1 0 1 1 1
+ 0 1 1 0 0 1 1 0
-----------------
  1 1 1 1 1 1 0 1
```

- Add one digit at a time from right to left
- If adding two values is bigger than the base max value, carry 1 to next digit

Adding Binary Numbers

- Works the same way as addition with decimal numbers
- Example:

```
        1 1             (carries)
  1 0 0 1 0 1 1 1
+ 0 1 1 0 0 1 1 0
-----------------
  1 1 1 1 1 1 0 1
```

How about?

```
  1011
+ 0110
------
```

- Add one digit at a time from right to left
- If adding two values is bigger than the base max value, carry 1 to next digit

Adding Binary Numbers

- Works the same way as addition with decimal numbers
- Example:

```
      1 1              (carries)
  1 0 0 1 0 1 1 1
+ 0 1 1 0 0 1 1 0
-----------------
  1 1 1 1 1 1 0 1
```

How about?

```
   1011
 + 0110
 ------
  10001
```

- Add one digit at a time from right to left
- If adding two values is bigger than the base max value, carry 1 to next digit

Number Systems

Number System	Base	Digits/Letters used
Binary Number System	Base: 2	0, 1
Decimal Number System	Base: 10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Hexadecimal Number System	Base: 16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9 A, B, C, D, E, F

Hexadecimal Representation (Base 16)

- Compact and more human-friendly representation of binary data
- Hex is short for hexadecimal
- 16 values: 0 1 2 3 4 5 6 7 8 9 A B C D E F

Hexadecimal Representation (Base 16)

- Compact and more human-friendly representation of binary data
- Hex is short for hexadecimal
- 16 values: 0 1 2 3 4 5 6 7 8 9 A B C D E F
- Comparison: With 8 digits, how big of a number can the system represent?

Hexadecimal Representation (Base 16)

- Compact and more human-friendly representation of binary data
- Hex is short for hexadecimal
- 16 values: 0 1 2 3 4 5 6 7 8 9 A B C D E F
- Comparison: With 8 digits, how big of a number can the system represent?

Binary:	2^8	=	256
Decimal:	10^8	=	100,000,000
Hexadecimal:	16^8	=	4,294,967,296

Hexadecimal Representation (Base 16)

- Compact and more human-friendly representation of binary data
- Hex is short for hexadecimal
- 16 values: 0 1 2 3 4 5 6 7 8 9 A B C D E F
- Comparison: With 8 digits, how big of a number can the system represent?

  Binary: 2^8 = 256

  Decimal: 10^8 = 100,000,000

  Hexadecimal: 16^8 = 4,294,967,296
- Physical computer addresses use hexadecimal representation: 04-33-C2-F8-EA-7C

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
- Problem: long sequences of 0's and 1's is tedious and error prone
- Convert each 4-bit group to hex to get shorter instruction:

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
- Problem: long sequences of 0's and 1's is tedious and error prone
- Convert each 4-bit group to hex to get shorter instruction:

0000

0

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
- Problem: long sequences of 0's and 1's is tedious and error prone
- Convert each 4-bit group to hex to get shorter instruction:

1010 0000

A 0

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
- Problem: long sequences of 0's and 1's is tedious and error prone
- Convert each 4-bit group to hex to get shorter instruction:

0011 1010 0000

3 A 0

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
- Problem: long sequences of 0's and 1's is tedious and error prone
- Convert each 4-bit group to hex to get shorter instruction:

1010 0011 1010 0000

A 3 A 0

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
- Problem: long sequences of 0's and 1's is tedious and error prone
- Convert each 4-bit group to hex to get shorter instruction:

1000 1010 0011 1010 0000

8 A 3 A 0

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
- Problem: long sequences of 0's and 1's is tedious and error prone
- Convert each 4-bit group to hex to get shorter instruction:

1101 1000 1010 0011 1010 0000
D 8 A 3 A 0

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
- Problem: long sequences of 0's and 1's is tedious and error prone
- Convert each 4-bit group to hex to get shorter instruction:

1110	1101	1000	1010	0011	1010	0000
E	D	8	A	3	A	0

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
- Problem: long sequences of 0's and 1's is tedious and error prone
- Convert each 4-bit group to hex to get shorter instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
  8 E D 8 A 3 A 0

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
- Problem: long sequences of 0's and 1's is tedious and error prone
- Convert each 4-bit group to hex to get shorter instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
  8 E D 8 A 3 A 0
  → 8E D8 A3 A0
- Hex is easier to read and write

From Binary to Hexadecimal

- The 32 bits here represent a computer instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
- Problem: long sequences of 0's and 1's is tedious and error prone
- Convert each 4-bit group to hex to get shorter instruction:
  1000 1110 1101 1000 1010 0011 1010 0000
  8 E D 8 A 3 A 0
  → 8E D8 A3 A0
- Hex is easier to read and write
- Each hex digit corresponds to a 4-big binary sequence
  e.g. 11 (decimal) = 1011 (binary) = B (hex)

Table of 3 Systems

- Mapping across decimal, binary, and hexadecimal

Decimal (base 10)	Binary (base 2)	Hexadecimal (base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Table of 3 Systems

- Mapping across decimal, binary, and hexadecimal
- What is 32 (decimal) in binary?

Decimal (base 10)	Binary (base 2)	Hexadecimal (base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Table of 3 Systems

- Mapping across decimal, binary, and hexadecimal
- What is 32 (decimal) in binary?

  32 ÷ 2 = 16 remainder 0

  … etc.

  →100000

Decimal (base 10)	Binary (base 2)	Hexadecimal (base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Table of 3 Systems

- Mapping across decimal, binary, and hexadecimal
- What is 32 (decimal) in binary?

  32 ÷ 2 = 16 remainder 0

  … etc.

  →100000
- What is 32 (decimal) in hex?

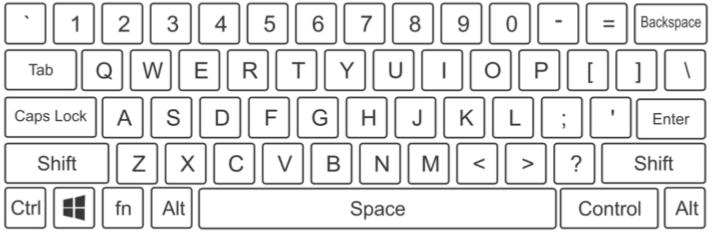
Decimal (base 10)	Binary (base 2)	Hexadecimal (base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Table of 3 Systems

- Mapping across decimal, binary, and hexadecimal
- What is 32 (decimal) in binary?
  32 ÷ 2 = 16 remainder 0
  … etc.
  →100000
- What is 32 (decimal) in hex?
  10 0000 (binary)
  0010 0000 (padding)
  → 20

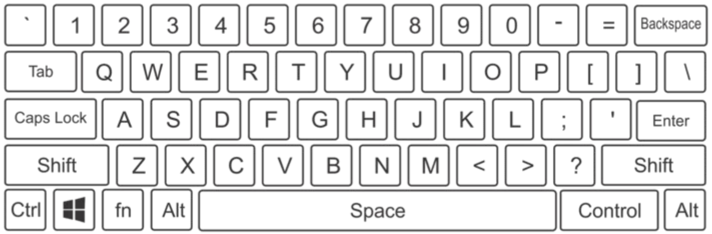
Decimal (base 10)	Binary (base 2)	Hexadecimal (base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

ASCII Table



- In fact, all other input characters get their own special mapping!

ASCII Table



- In fact, all other input characters get their own special mapping!
- ASCII = the American standard code for information interchange
- Invented in 1963
- Advantages of a standard:
    - Computer parts built by different manufacturers can be connected
    - Programs can create data and store it so that other programs can process it later
- 7-bit ASCII encoding handles:
    - All basic letters (upper and lower case)
    - All numbers, punctuation marks, and special characters

ASCII Table Mapping

- Uses 7-bit encoding
- 7 bits gives $2^7 = 128$ possible symbols

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

ASCII Table Mapping

- Uses 7-bit encoding
- 7 bits gives $2^7 = 128$ possible symbols
- 1981, IBM extended this to 8-bit encoding
- To allow more space to represent characters from other languages

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

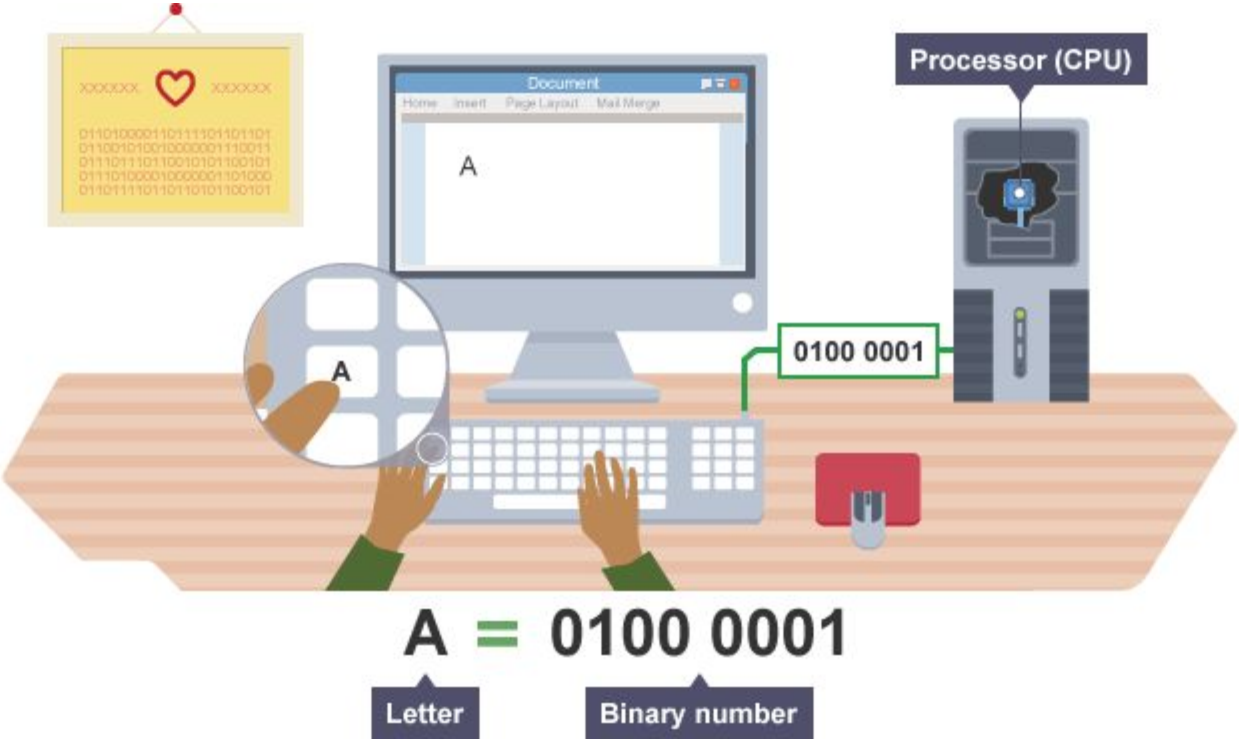
ASCII Table Mapping

- Uses 7-bit encoding
- 7 bits gives $2^7 = 128$ possible symbols
- 1981, IBM extended this to 8-bit encoding
- To allow more space to represent characters from other languages
- 1991: Unicode was introduced as the universal character encoding system (supports 2^{16} characters)

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

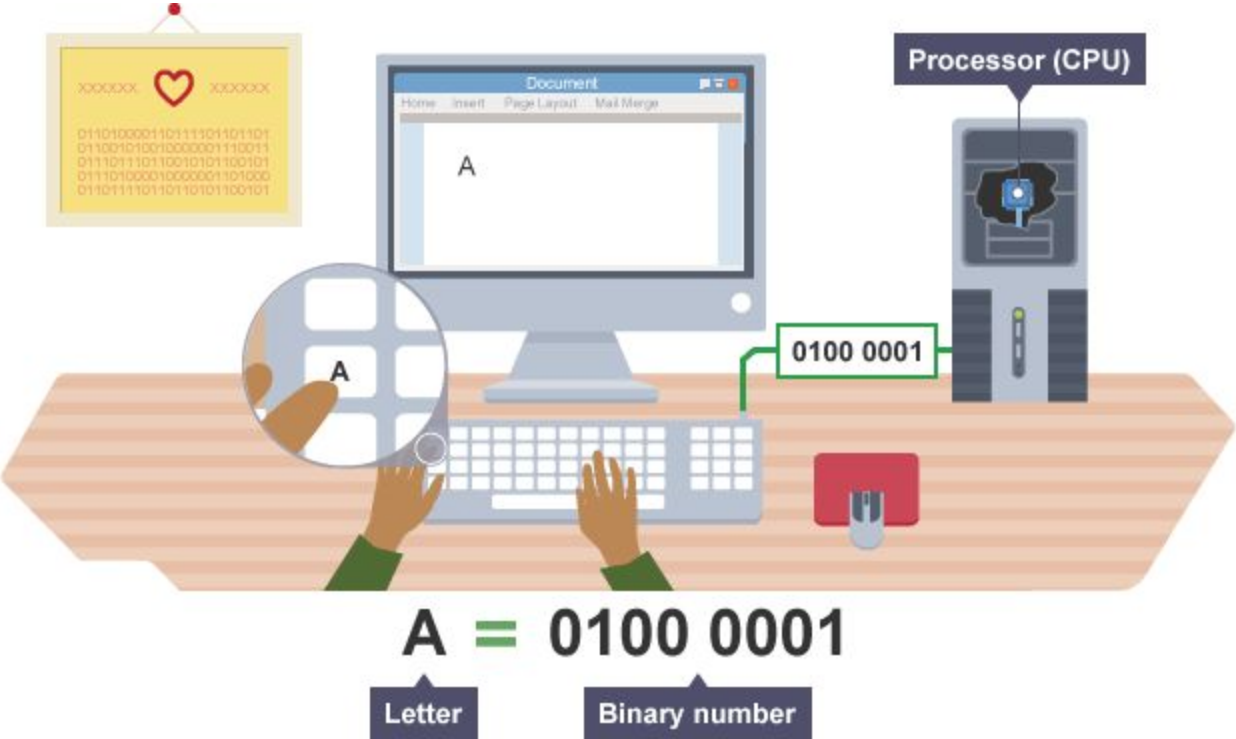
Encoding Higher-Level Information

- ASCII or Unicode tells us how to map text to binary representation



Encoding Higher-Level Information

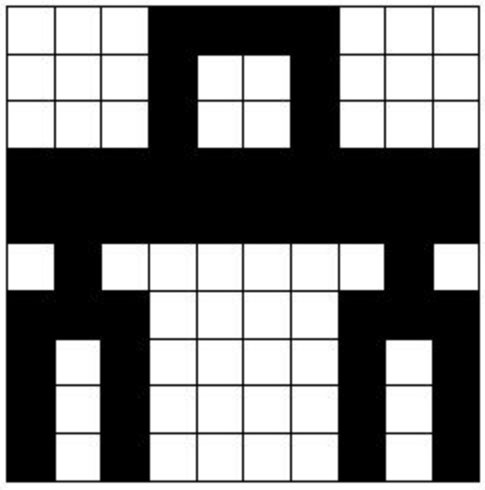
- ASCII or Unicode tells us how to map text to binary representation



What about images and sounds?

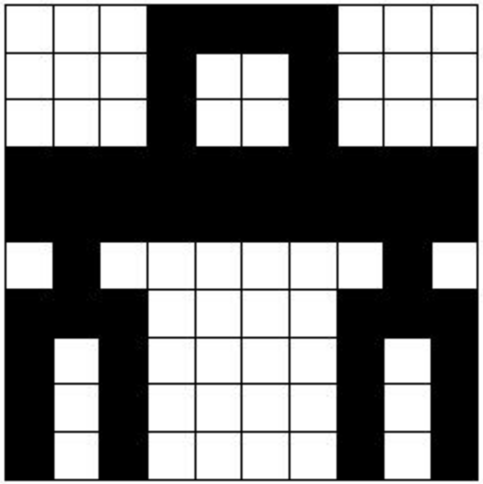
Representing Images

- Images are made up of pixels
- Consider black and white images



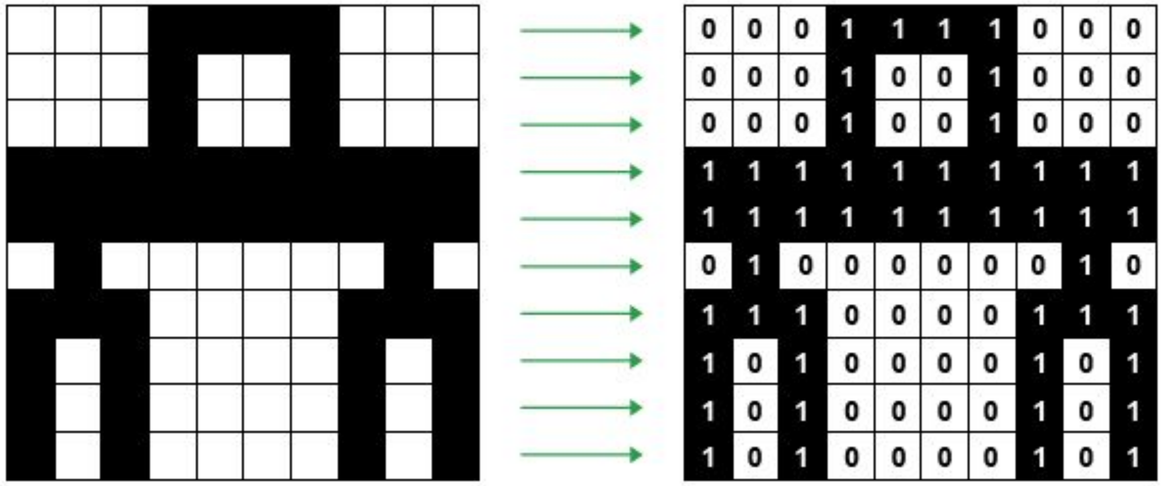
Representing Images

- Images are made up of pixels
- Consider black and white images
- Mapping: 1 is black (or on) and 0 is white (or off)



Representing Images

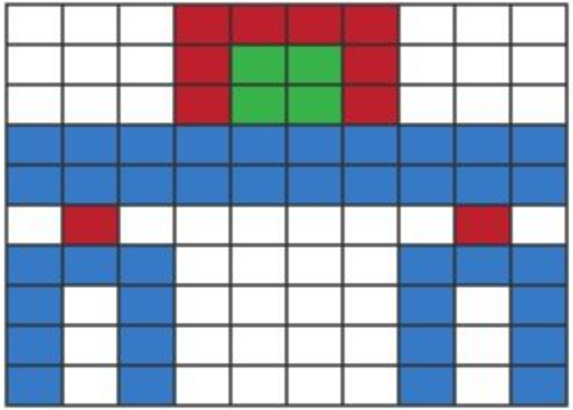
- Images are made up of pixels
- Consider black and white images
- Mapping: 1 is black (or on) and 0 is white (or off)



Representing Colours

- Suppose computers use RGB (red, green, blue)

  How many bits do we need to represent these colours?



Representing Colours

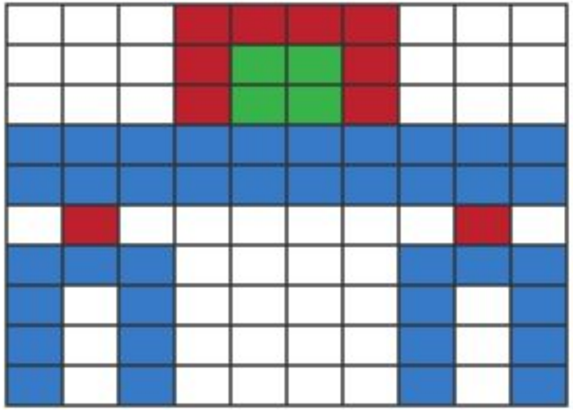
- Suppose computers use RGB (red, green, blue)
- We can use a 2-bit mapping for 4 colours:

  00 = white

  01 = blue

  10 = green

  11 = red



Representing Colours

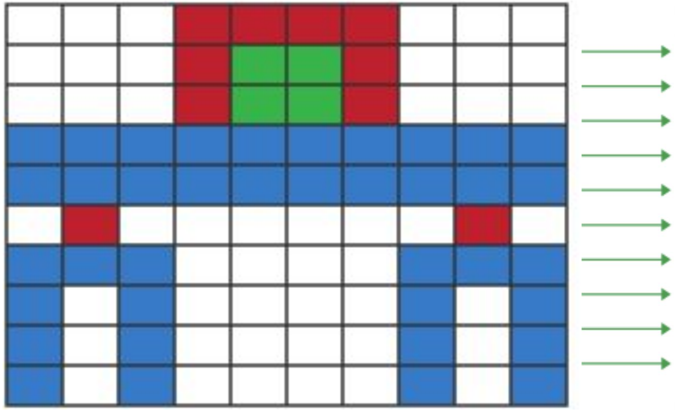
- Suppose computers use RGB (red, green, blue)
- We can use a 2-bit mapping for 4 colours:

  00 = white

  01 = blue

  10 = green

  11 = red



00	00	00	11	11	11	11	00	00	00
00	00	00	11	10	10	11	00	00	00
00	00	00	11	10	10	11	00	00	00
01	01	01	01	01	01	01	01	01	01
01	01	01	01	01	01	01	01	01	01
00	11	00	00	00	00	00	00	11	00
01	01	01	00	00	00	00	01	01	01
01	00	01	00	00	00	00	01	00	01
01	00	01	00	00	00	00	01	00	01
01	00	01	00	00	00	00	01	00	01

Representing Colours

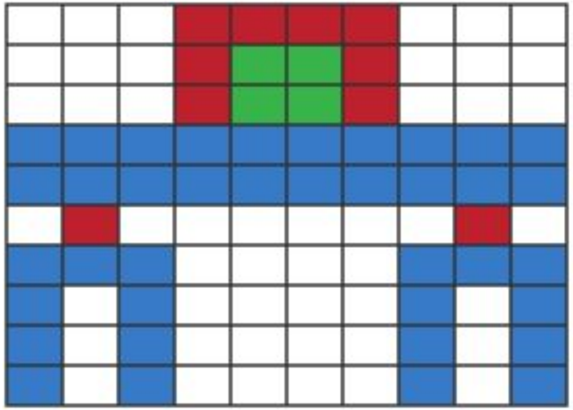
- Suppose computers use RGB (red, green, blue)
- We can use a 2-bit mapping for 4 colours:

00 = white

01 = blue

10 = green

11 = red

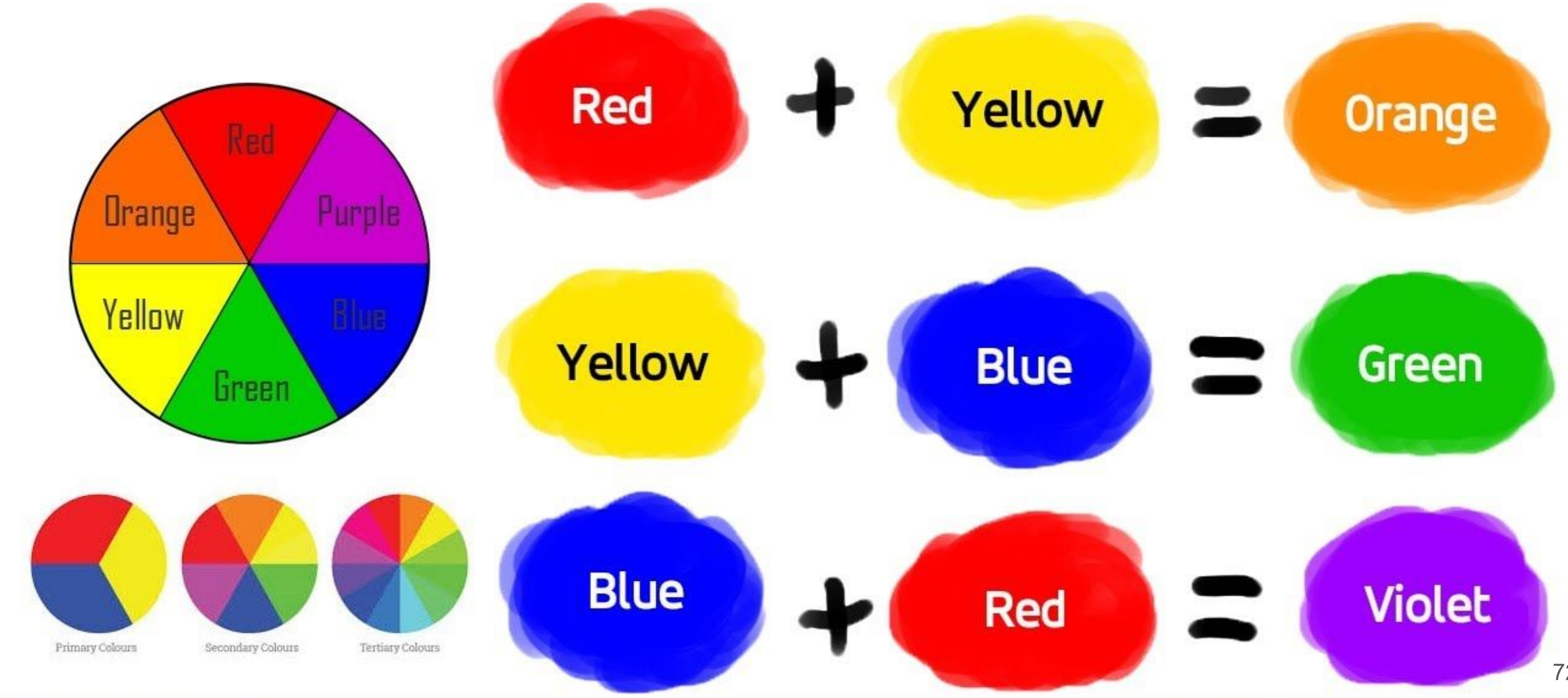


00	00	00	11	11	11	11	00	00	00
00	00	00	11	10	10	11	00	00	00
00	00	00	11	10	10	11	00	00	00
01	01	01	01	01	01	01	01	01	01
01	01	01	01	01	01	01	01	01	01
00	11	00	00	00	00	00	00	11	00
01	01	01	00	00	00	00	01	01	01
01	00	01	00	00	00	00	01	00	01
01	00	01	00	00	00	00	01	00	01
01	00	01	00	00	00	00	01	00	01

How many colours do we have, in general?

How many bits do we need, in general?

Remember Painting?

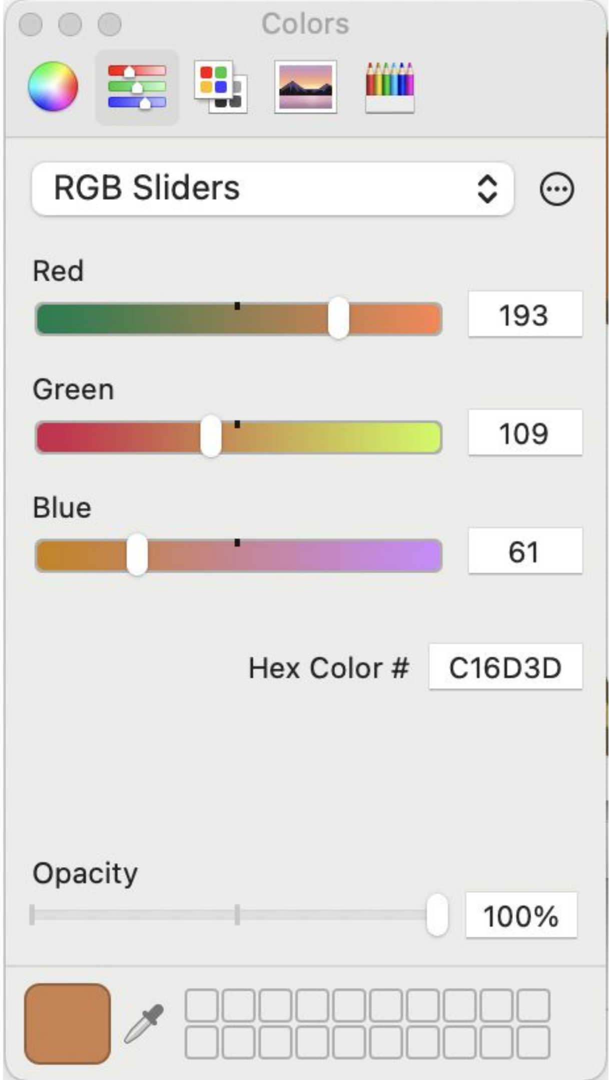


Colour Systems

- **Red-Yellow-Blue** is a traditional pigment-based (subtractive) system used in painting for mixing physical paints
- **Red-Green-Blue** is an light-based (additive) system used for digital screens and light
  - Combines red, green, and blue light in various intensities
  - Creates a wide range of colors
  - All colors mixing to white
  - Based on how the human eye cone cells perceive colors

Reading Colour Pickers

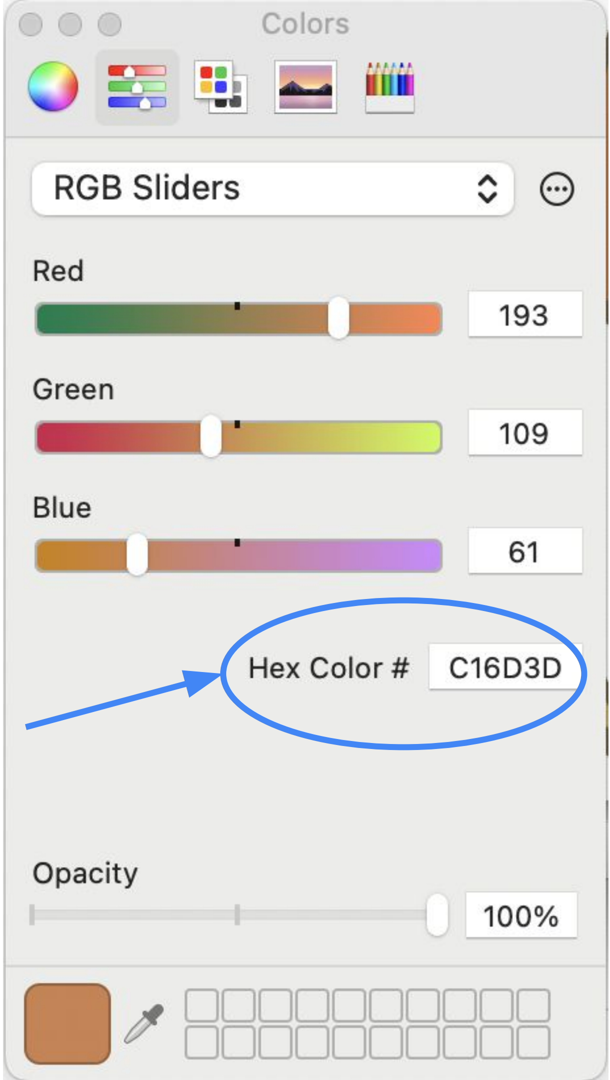
- Common format: Each of R, G, and B is stored as an 8-bit number
  - Pure red = (255, 0, 0)
  - Pure green = (0, 255, 0)
  - Pure blue = (0, 0, 255)
  - 0 means no light (dark)
    Black = (0, 0, 0)
  - 255 means full brightness
    White = (255, 255, 255)
- Gives 256^3 = 16,777,216 possible colors



Reading Colour Pickers

- Common format: Each of R, G, and B is stored as an 8-bit number
    - Pure red = (255, 0, 0)
    - Pure green = (0, 255, 0)
    - Pure blue = (0, 0, 255)
    - 0 means no light (dark)
      Black = (0, 0, 0)
    - 255 means full brightness
      White = (255, 255, 255)
- Gives 256^3 = 16,777,216 possible colors

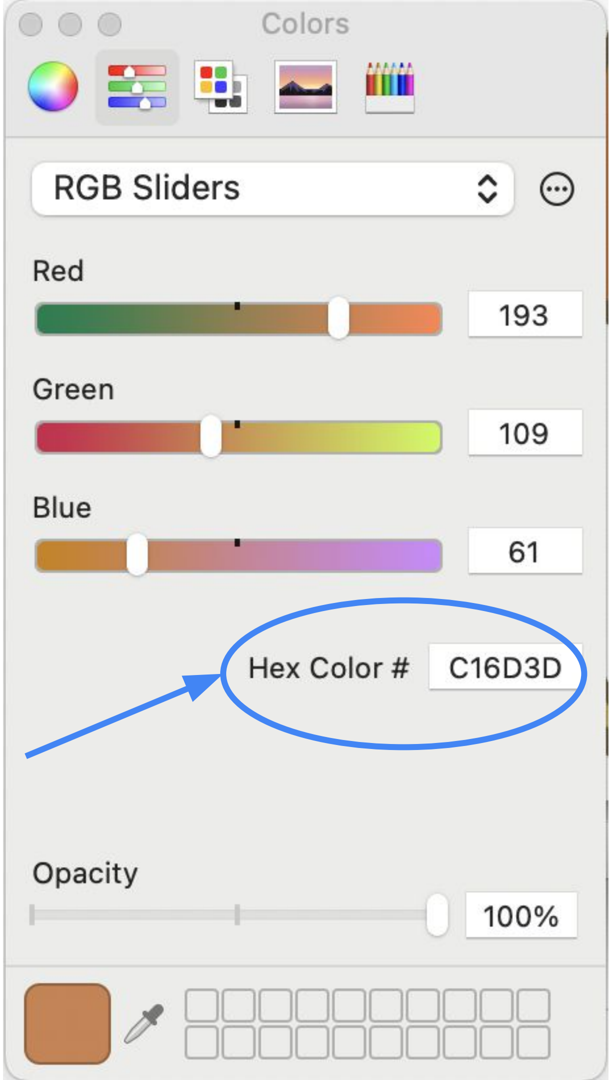
Why 6 digits?



Reading Colour Pickers

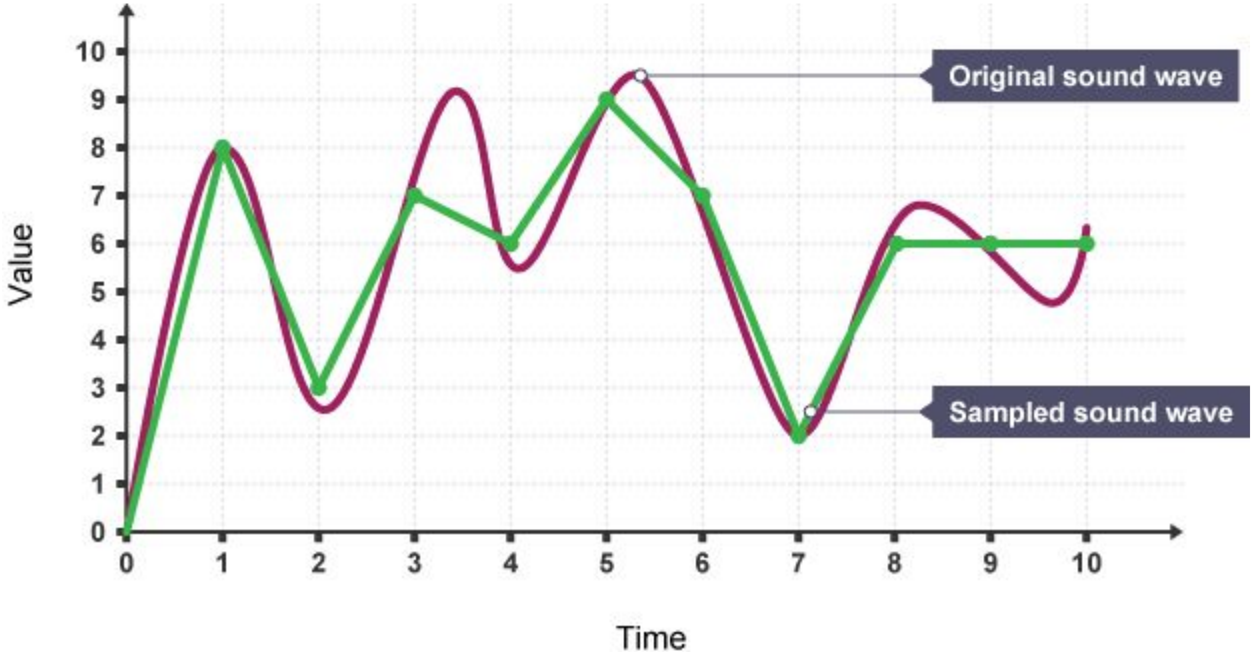
- Common format: Each of R, G, and B is stored as an 8-bit number
  - Pure red = (255, 0, 0)
  - Pure green = (0, 255, 0)
  - Pure blue = (0, 0, 255)
  - 0 means no light (dark)
    Black = (0, 0, 0)
  - 255 means full brightness
    White = (255, 255, 255)
- Gives 256^3 = 16,777,216 possible colors

What is 6D?
What about FF?



Representing Sounds

- Sound waves are captured and sampled at regular discrete points in time
- These "samples" represent the amplitude (strength) of the sound wave



Other Systems of Data Representations

- QR Codes
- NATO broadcast alphabet
- Morse code

  …
- Different systems used for different purposes
- Starts with basic units, then combine to create larger data items