LG 467 Computers in Linguistics

[1-2021] Topic 1: Encoding Language

Sakol Suethanapornkul

Previously...

Goal: Make computers process & understand human language

Utilities: Do useful things with language for us

- Search engines (Google, Bing, DuckDuckGo)
- Virtual assistants (Siri, Alexa, Cortana, etc.)
- Chatbots
- Voice & speech recognition softwares
- Translation systems

But, here's the wrinkle...

For computers to be able to process language...

- they need to have knowledge of language, which means
- they must be able to encode language

How can computers actually do this?

A "byte"-size Intro

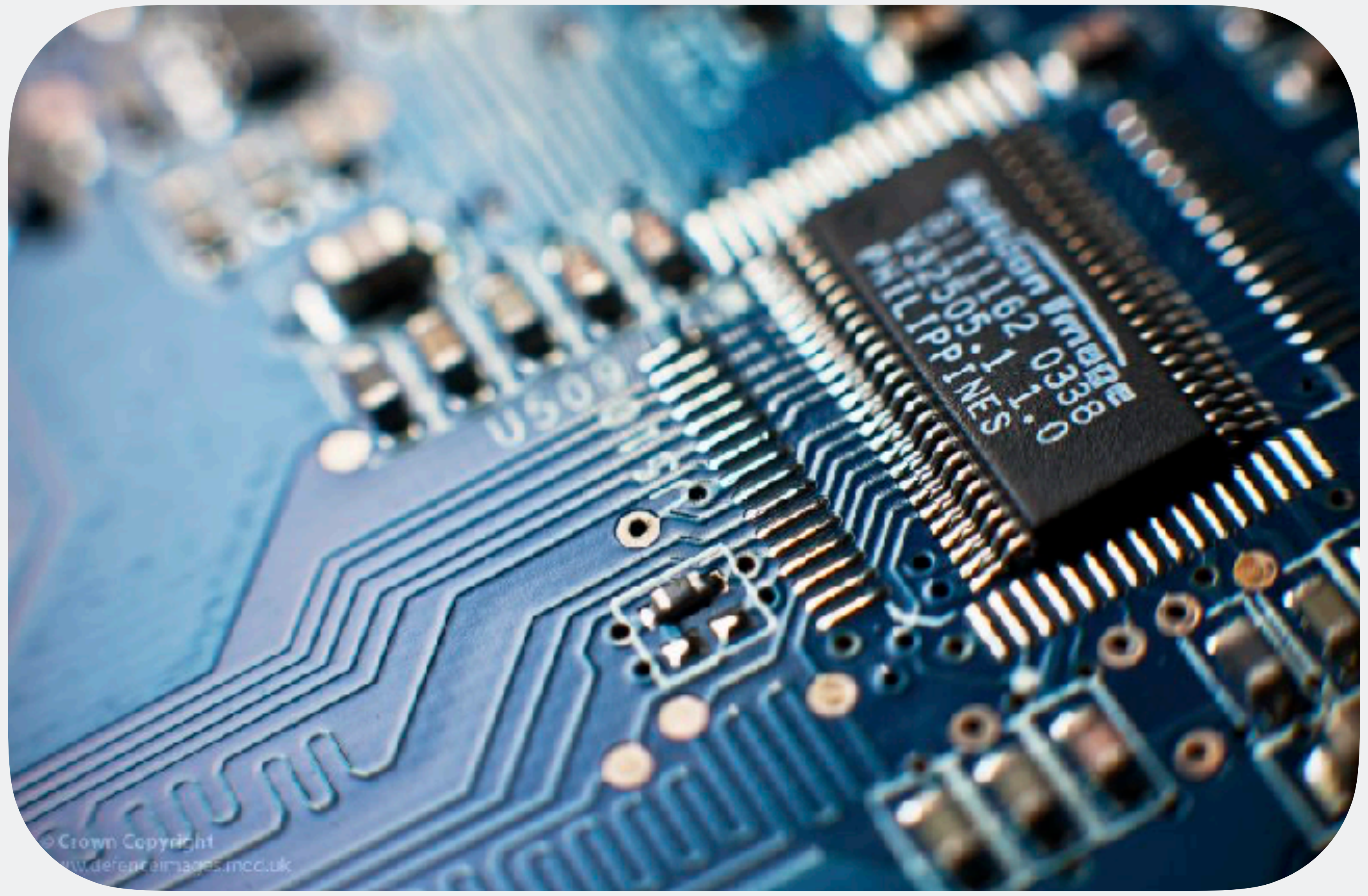
Computer 101

To a computer, human language is just binary data, the 0s and 1s

Hello, there!	→	01001000 01100101 01101100 01101100 01101111 00100000 01110100 01101000 01100101 01110010 01100101 00100001
สวัสดี	→	11100000 10111000 10101010 11100000 10111000 10100111 11100000 10111000 10110001 11100000 10111000 10101010 11100000 10111000 10010100 11100000 10111000 10110101

Computer 101

Why is information stored and represented this way?



- Information "travels" over wires.
- Is "information" passing through a wire?:
  - 1 = Yes
  - 0 = No

Image source: Flickr

Computer 101

- ***Bits*** (binary digits) = most basic unit of information (on-off state)
  - Logical state: 1 (= ON; YES; TRUE) or 0 (= OFF; NO; FALSE)
  - Each bit (= wire) doesn't carry much information, but…
- ***Bytes*** (each is equal to ***8 bits***) = basic addressable unit
  - Bytes can represent much more information (e.g., characters, numbers, etc.)

Decimal number: 199 ➔ 11000111

Binary number system

- Decimal notation (numbers 0-9)

	10^4	10^3	10^2	10^1	10^0
45,250 →	4	5	2	5	0

Binary number system

- Binary notation (numbers: 0 and 1)

		2^4	2^3	2^2	2^1	2^0
10110	➔	1	0	1	1	0

- What is this number in decimal?

Binary number system

- Binary notation (numbers: 0 and 1)

		2^4	2^3	2^2	2^1	2^0
10110	➡	1	0	1	1	0

- Answer: $(1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 22$

Binary number system

- Going from decimal numbers to binary (division method)

Decimal	Remainder?	Binary
25/2 = 12	Yes	1
12/2 = 6	No	01
6/2 = 3	No	001
3/2 = 1	Yes	1001
1/2 = 0	Yes	11001

Encoding Texts

Encoding written language

- Representing characters (and texts) with 0s and 1s:

Hello!

010010000110010101101100011011000110111100100001

- Each character is mapped to a ***code point*** (i.e., a unique integer)
  - H → 72_{dec}
  - e → 101_{dec}

Adapted from: Na-Rae Han's LING 1330/2330 lecture

Encoding written language

- Each code point is represented as a binary number, using **a fixed number of bits**

- Say: 8 bits (= 1 byte)
  - H → 72_{dec} → 01001000 ($2^6 + 2^3$ → 64 + 8 = 72)
  - e → 101_{dec} → 01100101 ($2^6 + 2^5 + 2^2 + 2^0$ → 64 + 32 + 4 + 1 = 101)

- One byte can represent 256 different characters ($2^8 = 256$)
  - 00000000 → 0_{dec} 11111111 → 255_{dec}

Adapted from: Na-Rae Han's LING 1330/2330 lecture

ASCII

- ASCII: American Standard Code for Information Interchange
  - One of the first character encoding standards (out in 1963)

- ASCII uses 7 bits (2^7 = 128 possible code points, from 0 to 127)
  - Code points for control characters (start of text, end of text, etc.)
  - 32 to 127 for **printable characters**
    - Space, punctuations (. , ; ! ?), symbols (< > & $ *), numbers
    - Uppercase letters and lowercase letters

ASCII

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C		
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Image source: Quara

ASCII

- So, what is this in English? (Note: 7 bits and spaces)

  1101000 1101111 1101101 1100101

ASCII

- So, what is this in English? (Note: 7 bits and spaces)

1101000 1101111 1101101 1100101

ANSWER: 1101000 → 104 → h

1101111 → 111 → o

1101101 → 109 → m

1100101 → 101 → e

- In an 8-bit representation, "insert" 0 as the highest bit

Encoding different systems

- ASCII is well and good for English characters
  - but how about diacritics (e.g., résumé)?
  - and all other scripts (Arabic, Thai, Japanese, etc.)?
- One solution: Extend ASCII to 8-bit encoding (256 characters) and use code points 128-255 with non-English characters
  - ISO 8859 has 16 implementations
    - ISO 8859-1: French, German, Spanish, etc.
    - ISO 8859-7: Greek alphabet

Adapted from: Na-Rae Han's LING 1330/2330 lecture

Encoding different systems

- Problems: Two different encodings
  - same code points for different characters
  - different code points for same characters
- We'd like to work with characters from any existing writing system
  - and also, in today's world, emojis 🐶 📱 💔 🤪 👏🏼
- Let's talk about Unicode…

Adapted from: Language & Computers' online material set

Unicode

- Unicode Consortium, founded in 1991, develops a single representation for every possible character

"Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language."

(www.unicode.org)

Adapted from: Language & Computers’ online material set

Unicode

- Unicode 13.0 contains 143,859 characters (incl. 55 new emojis)
  - 14.0 (beta; Sept 2021) adds new scripts & emojis 🫠 🫃 🪷
- Unicode uses 32 bits (4 bytes), meaning we can store:
  - 2^{32} = 4,294,967,296 possible characters!
- Wait, does this mean everything is encoded with 32 bits? Isn't this wasteful for some characters?

Unicode

- Let's take one example:

h	→	ASCII (7 bits)	1101000
	→	(8 bits; 1 byte)	01101000
	→	(16 bits; 2 bytes)	0000000001101000
	→	(32 bits; 4 bytes)	0000000000000000 0000000001101000

Adapted from: Na-Rae Han's LING 1330/2330 lecture

Unicode

- Unicode has three versions
  - UTF-32 (32 bits): direct representation
  - UTF-16 (16 bits): $2^{16} = 65,536$
  - UTF-8 (8 bits): $2^{8} = 256$

- One awesome fact: You can represent 2^{32} code points with UTF-8

---

* UTF is short for Unicode Transformation Format.

Unicode

- UTF-8 (or utf8) uses a variable-width encoding
  - Encode characters in as few bytes as possible but will use multiple bytes if needed

Example: home

01101000 01101111 01101101 01100101

Unicode

- UTF-8 (or utf8) uses a variable-width encoding
  - Encode characters in as few bytes as possible but will use multiple bytes if needed

Example: home

01101000	01101111	01101101	01100101

Unicode

- UTF-8 (or utf8) uses a variable-width encoding
  - Encode characters in as few bytes as possible but will use multiple bytes if needed

Example: home

01101000	01101111	01101101	01100101

Unicode

- UTF-8 (or utf8) uses a variable-width encoding
  - Encode characters in as few bytes as possible but will use multiple bytes if needed

Example: สกล

11100000 10111000 10101010
11100000 10111000 10000001
11100000 10111000 10100101

Unicode

- UTF-8 (or utf8) uses a variable-width encoding
  - Encode characters in as few bytes as possible but will use multiple bytes if needed

Example: สกล

11100000 10111000 10101010
11100000 10111000 10000001
11100000 10111000 10100101

Unicode

- UTF-8 (or utf8) uses a variable-width encoding
  - Encode characters in as few bytes as possible but will use multiple bytes if needed

Example: สกล

11100000 10111000 10101010 → 111000101010 → $E2A_{hex}$

11100000 10111000 10000001 → 111000000001

11100000 10111000 10100101 → 111000100101

Unicode

- UTF-8 (or utf8) uses a variable-width encoding
  - So, let's return to our earliest example

สวัสดี → 11100000 10111000 10101010 11100000 10111000 10100111 11100000 10111000 10110001 11100000 10111000 10101010 11100000 10111000 10010100 11100000 10111000 10110101

Unicode

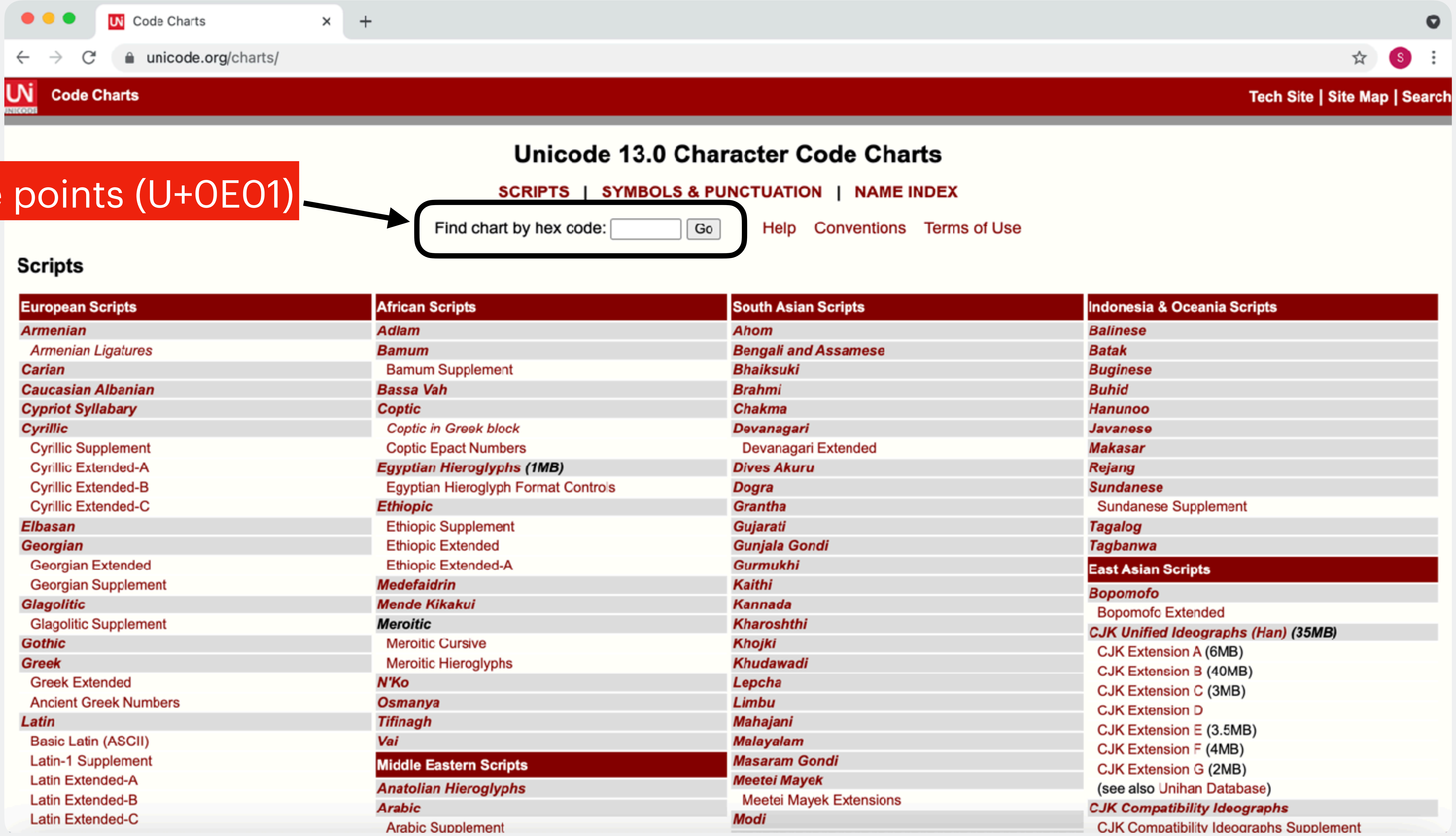
- Reading Unicode code points

U+0041	Latin capital letter A
U+006A	Latin small letter j

- U+ means "Unicode"
- 0041_{hex} and $006A_{hex}$ are code points
  - hex digits for easy byte conversion (and readability)

Unicode

Look up code points (U+0E01)



Unicode

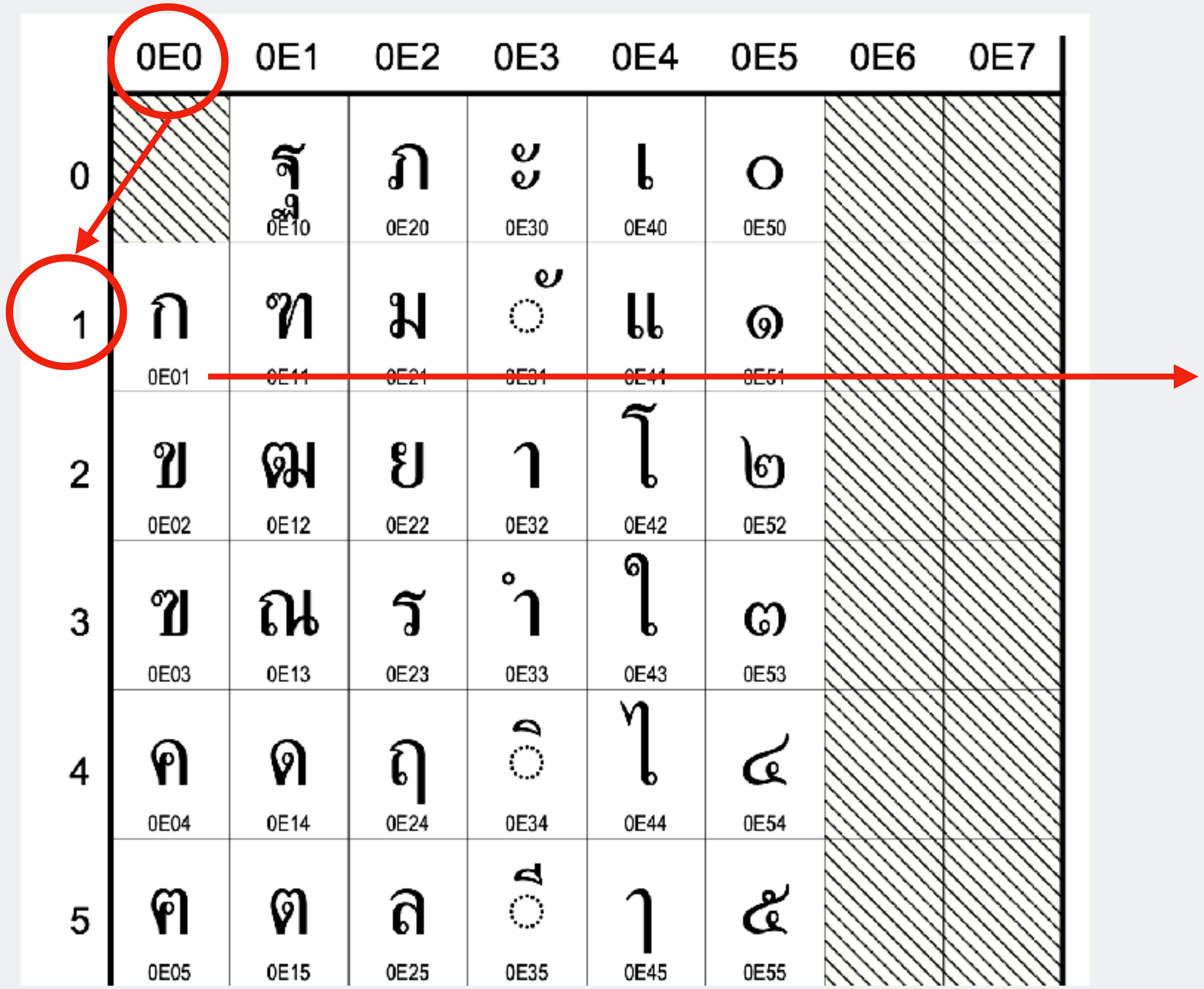
- C0 Controls and Basic Latin

	000	001	002	003	004	005	006	007
0	NUL 0000	DLE 0010	SP 0020	0 0030	@ 0040	P 0050	` 0060	p 0070
1	SOH 0001	DC1 0011	! 0021	1 0031	A 0041	Q 0051	a 0061	q 0071
2	STX 0002	DC2 0012	" 0022	2 0032	B 0042	R 0052	b 0062	r 0072
3	ETX 0003	DC3 0013	# 0023	3 0033	C 0043	S 0053	c 0063	s 0073
4	EOT 0004	DC4 0014	$ 0024	4 0034	D 0044	T 0054	d 0064	t 0074
5	ENQ 0005	NAK 0015	% 0025	5 0035	E 0045	U 0055	e 0065	u 0075

U+0072

Unicode

- Thai



	0E0	0E1	0E2	0E3	0E4	0E5	0E6	0E7
0		ฐ 0E10	ภ 0E20	ะ 0E30	เ 0E40	๐ 0E50		
1	ก 0E01	ฑ 0E11	ม 0E21	ั 0E31	แ 0E41	๑ 0E51		
2	ข 0E02	ฒ 0E12	ย 0E22	า 0E32	โ 0E42	๒ 0E52		
3	ฃ 0E03	ณ 0E13	ร 0E23	ำ 0E33	ใ 0E43	๓ 0E53		
4	ค 0E04	ด 0E14	ฤ 0E24	ิ 0E34	ไ 0E44	๔ 0E54		
5	ฅ 0E05	ต 0E15	ล 0E25	ี 0E35	ๅ 0E45	๕ 0E55		

U+0E01

Unicode

- Hexadecimal or hex (base-16)
  - hex digits = 0 1 2 3 4 5 6 7 8 9 A B C D E F
  - a hex digit is equivalent to 4 bits; two hex digits = 8 bits

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

Why did you put me through this?

- Webpages, emails, etc. rely on this standard
- It’s good to know a bit about this, although we don’t have to deal with bits and bytes directly in our work!