WTF-8

Mistakes were made…

Daniel Frey, 2024-06-05

Words are Hard

Language and Computers

- Fun walk-around in history, both ancient and modern
- This talk simplifies and skips over a lot of stuff
- Language and how it is constructed is super interesting!
- Relevant for computer systems, hence for computer scientists, engineers, ...
- Not meant to point fingers and bash people, just understand how history unfolds
- Enjoy the rabbit hole by googling it yourself after the talk

Babylon

Spoken Language

Let's talk about it...

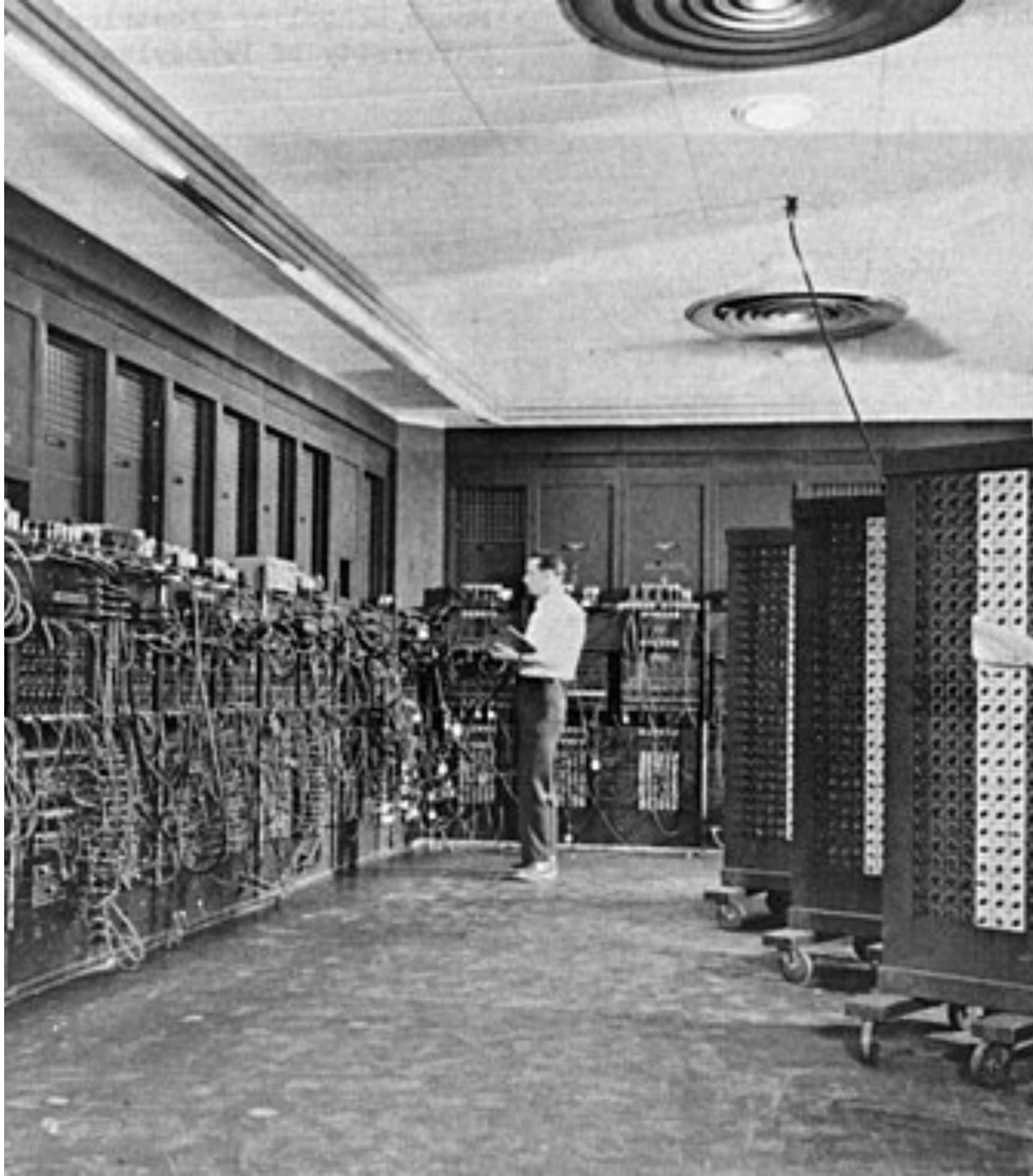
- Before written language, there was spoken language (Duh!)
- This is where things went south already (or north?)
- Large distances, separated communities
- Dialects became languages
- Having no internet and smartphones means no long distance communication (fun fact: Yodeling was long distance communication in the mountains)
- So... no problem!

Written Language

Hello مرحبًا नमस्ते привет こんにちは Γειά σου 你好 안녕하세요

- Written languages were invented multiple times, independently
- First recorded language was Sumerian, ca. 3400 B.C.
  (fast forward: Arabic, English, Elvish, Klingon… there are tons of languages out there)
- (Modern) writing systems are build upon alphabets or syllabaries
- Graphemes are the smallest functional unit of a writing system
- Graphemes are abstract, similar to the notion of a character in a computer system
- Glyphs are a specific shape, design, or representation of a character (a, *a*, a, **a**, **a**, ***a***)

Babylon 2.0



“I think there is a world market for maybe five computers”

Thomas Watson, president of IBM, 1943

The History of Computers

- Previous title slide showed ENIAC, constructed 1943-45
  (Electronic Numerical Integrator and Computer)
- 30 tons, 200 kW, 18.000+ vacuum tubes, …
- World’s first Turing-complete electronic computer
- Other honorable mentions are:
  - Zuse Z3, 1938-41, first working electro*mechanical* computer, *kinda* Turing-complete
  - Colossus, 1943-44, broke Enigma, first electronic computer, *not* Turing-complete

The 8-bit Era

The best era, period.

- Transistors: Proposed 1926, first working device 1947, MOSFETs in 1959
- First integrated circuits (ICs), first microprocessors (4004 in 1970, 6502 in 1975)
- Computers weren’t limited to governments and big companies anymore
- Started the Home Computer Revolution (aka the 8-bit era)
- Normal people had access to computers
- Next milestone: Connect them up!

Connecting the World

Networking is not a Soft Skill

- ARPANET created by DARPA in 1966-71
- First TCP/IP based wide area network, TCP/IP v4 in 1983, connected to universities in 1986
- This became the Internet, the original ARPANET was formally decommissioned in 1990
- ASCII first standardised in 1963
- Uses 7 bits, limited to 127 characters
- 8-bit variants added later, but which one do you want to use?

Codepages

ISO-8859-1 ≠ ISO-8859-15

- ASCII is nice for the Latin alphabet, but only modern English uses a pure Latin alphabet
- Many other languages add diacritics (ä, å, á, ç, …) and/or digraphs (ß, æ, …)
- What about other alphabets, e.g. Greek, Cyrillic, Chinese, you-name-it…?
- Codepages were a band-aid, not a solution
- We need something better

Finally... Unicode!

Solve all you problems, forever.

- Work began ca. 1987 at Xerox, the Unicode Consortium was incorporated in January 1991
- First Unicode standard was published in October 1991
- Initially code points were limited to 16 bits
- Unicode 2.0 was published on 1996
- Introduces a surrogate character mechanism to encode over a million code points
- Code points need to be mapped to the real world... fun with encodings!

Early Adopters

Early Adopters

- Early adopters tended to use UCS-2
  (the fixed-length two-byte obsolete precursor to UTF-16)
- Windows NT, July 1993 adopted UCS-2/UTF-16
- Java, 1994 also used UCS-2/UTF-16
- Oracle, May 1995 used UCS-2
- Unix-like systems adopted UTF-8
- The internet uses UTF-8 almost exclusively (97.9% as of April 2023)

Encodings

UTF-16

Simple, right?

- Ignore endianness (ain’t nobody got time for that)
- Neither will we discuss UTF-32 (ain’t nobody got space for that)
- Surrogate character mechanism to enhance the available code points beyond 16 bit
- `0x0000-0xCFFF` and `0xE000-0xFFFF` directly represents a code point
- `0xD800-0xDBFF` is a high surrogate, `0xDC00-0xDFFF` is a low surrogate
- Must occur as surrogate pairs, one high surrogate followed by one low surrogate

UTF-16

Simple, right?... RIGHT?

- High surrogate (`0xD800-0xDBFF`) -> `0b110110YYYYYYYYYY`
- Low surrogate (`0xDC00-0xDFFF`) -> `0b110111XXXXXXXXXX`
- Map to a code point in the `0x10000-0x10FFFF` range
  - Code point = `0bYYYYYYYYYYXXXXXXXXXX` + `0x10000`
- Handling of surrogate pairs is often not thoroughly tested
- This leads to persistent bugs and potential security holes, e.g. CVE-2008-2938, -2012-2135

UTF-8

Who uses UTF-16 anyways?

- `0b0XXXXXXX`
  One byte, compatible with ASCII, encodes code points `0x00-0x7F`

- `0b110XXXXX 0b10XXXXXX`
  Two bytes, encodes code points `0x80-0x7FF`

- `0b1110XXXX 0b10XXXXXX 0b10XXXXXX`
  Three bytes, encodes code points `0x800-0xFFFF`

- `0b11110XXX 0b10XXXXXX 0b10XXXXXX 0b10XXXXXX`
  Four bytes, encodes code points `0x10000-0x10FFFF`

UTF-8

UTF-8 rulez!

- Not all byte sequences are valid UTF-8 strings
- You must use the shortest possible UTF-8 encoding of a code point, no overlong encodings
  - Example: Code point 32 (space) could be encoded as `0b11000000 0b10100000`, but also as `0b00100000`. The latter is shorter, therefore the *only* correct UTF-8 encoding
- UTF-8 strings do not contain any reserved surrogate code points (`0xD800-0xDFFF`)

Mistakes

Mistakes everywhere!

CESU-8

Compatibility Encoding Scheme for UTF-16: 8-Bit

- “Popular” with early adopters, especially Oracle
- Encodes surrogate pairs (from UCS-2/UTF-16) to UTF-8 separately
  - Generates a three byte UTF-8 encoding for the high surrogate, and…
  - …generates a three byte UTF-8 encoding for the low surrogate…
  - …instead of a four byte UTF-8 encoding of the code-point
- Leads to `0b11101101 0b1010XXXX 0b10XXXXXX 0b11101101 0b1011XXXX 0b10XXXXXX` instead of `0b11110XXX 0b10XXXXXX 0b10XXXXXX 0b10XXXXXX`

CESU-8

Compatibility Encoding Scheme for UTF-16: 8-Bit

- Actually a Unicode standard now
- Decode a UTF-8 code point, if it is a high surrogate:
  - Decode the next UTF-8 code point
  - Check it is a low surrogate, combine both into the *actual* code point
- Supporting CESU-8 in HTML documents is prohibited by the W3C, as it would present a cross-site scripting vulnerability!

MUTF-8

Modified UTF-8

- Originated in the Java programming language
- Modified UTF-8 is an “enhancement” of CESU-8, adding a special overlong encoding of the `NUL` character (`U+0000`), represented as a two-byte sequence: `0xC0 0x80`
- A Modified UTF-8 string therefore never contain any actual null bytes, but can contain all Unicode code points including `U+0000`, which allows such strings (with a null byte appended) to be processed by traditional null-terminated string functions
- All known MUTF-8 implementations are also CESU-8 (but not vice versa)

WTF-8

Wobbly Transformation Format – 8-bit

- Encodes surrogate code points even if they are not in a pair
- Must only be used internally and converted to a Unicode encoding at a system's boundary before being emitted
- Except the WTF-8 specification doesn't really say how the conversion is done
- Why was WTF-8 even created? What is the use case?

WTF-8

Wobbly Transformation Format – 8-bit

- Why? That's why:
  - In ECMAScript (aka JavaScript), a `String` value is defined as a sequence of 16-bit integers that *usually* represent UTF-16 text but may or may not be well-formed
  - Windows applications normally use UTF-16, but the file system treats paths and file names as an opaque sequence of `WCHAR`s (16-bit code units)
- We say that strings in these systems are encoded in potentially ill-formed UTF-16 or WTF-16

Thank you!

Questions?