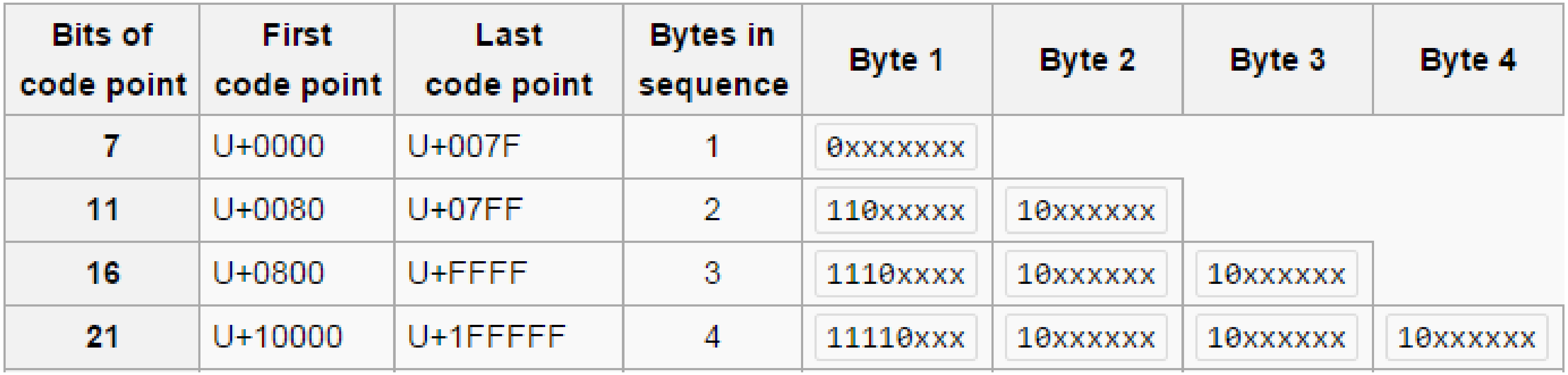
Quick Aside on ASCII and Unicode

- ASCII: 7-bit representations of characters for text. (Including non-print characters like NULL.) So only 128 symbols in total.
- Historically: various local attempts to expand to 8- or 16-bit representations to accommodate more.
- Almost universal now: Unicode (which represents even more than a 16-bit system could)…It has "code points" from U+000000 to U+10FFFF (i.e., $2^{20} + 2^{16}$, or 1,114,112 possible characters, the majority of which code points are still unassigned)
  - Actually $2^{20} + 2^{16} - 2^{11} = 1{,}112{,}064$, since there are 2,048 illegal codepoints
- Why can you still get away with ASCII?

UTF-8 Encoding of Unicode



Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4
7	U+0000	U+007F	1	0xxxxxxx			
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx		
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx	
21	U+10000	U+1FFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Source: Wikipedia

- First 128 code points correspond to ASCII
- Beginnings of bytes tell the role of the byte…0 = ASCII, 10 = continuation byte, 110=start of two-byte sequence, etc.
- No code pointed above U+10FFFF actually allowed

Internal Representation of Unicode

- Many languages still use UTF-16 representations when they store unicode text in memory (wchar_t)
- These are a similar (but more complex) scheme that requires either two bytes or four bytes depending on the character
- UTF -16 is less efficient for English (2 bytes vs 1). More efficient for many East Asian languages (2 bytes vs 3)
- No codepoints are allowed above U+10FFFF or in the range U+D800 to U+DFFF to keep compatible with UTF-16