Activity 10: Characters
Recorder's Report

Manager: Reader:

Recorder: Driver:

Date: Score: Satisfactory / Not Satisfactory

Record your team's answers to the key questions (marked with ) below.

a) Model 1, Question #5

b) Model 3, Question #15

c) Model 4, Question #35

Activity 10: Characters

In our discussion of encoding (lesson 1) we noted that characters (including letters, numbers, and punctuation) could be represented as a glyph (using ink on paper), as a gesture (using American Sign Language), as a combination of short and long tones (Morse Code), or as raised dots on paper (Braille). At the end of the lesson on subtraction (lesson 6) we noted that anything that can be mapped to discrete values, like the form of the British invasion ("one if by land, two if by sea"), could be encoded as binary numbers.
In this lesson we look at encoding printable characters as numeric codes.

Content Learning Objectives

After completing this activity, students should be able to:

- Identify the range of codes used for ASCII;
- Describe the types of characters included in Unicode; and,
- Describe the difference between a character codepoint and an encoding.

Process Skill Goals

During the activity, students should make progress toward:

- Translate characters to and from ASCII given a chart.

Sources

- Figure 1: https://en.wikipedia.org/wiki/File:ASCII-Table-wide.svg
- Figure 2: https://en.wikipedia.org/wiki/ISO/IEC_8859-1
- Unicode: https://en.wikipedia.org/wiki/Unicode
- UTF-8: https://en.wikipedia.org/wiki/UTF-8

Model 1 Character Codes

Four our purposes, a character is a letter, number, punctuation, or other symbol used in writing. Each character is an abstraction that can be represented in print (ink on paper or pixels on a screen) using a glyph from a font. So 'A' and 'A' are the same character represented with a different font (Times New Roman and Arial respectively). The same character can also be represented as a gesture (using American Sign Language), as a combination of short and long tones (Morse Code), or as raised dots on paper (Braille).
A number is written as a sequence of digits (characters) in a base such as decimal, octal, or hexadecimal (see lesson 3). A character such as 'F' can be used as a letter (in the name "Foster") or as a digit (in the hexadecimal number "0xFF").

Refer to Model 1 above as your team develops consensus answers to the questions below.

Questions (5 min) **Start time:**

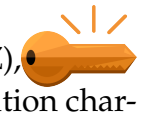
1. How many digits (characters) are in the number 99324?

2. How many bits are required to specify codes for the decimal digits (0-9)? (Hint: not ten!)

3. How many bits are required to specify codes for the uppercase English letters (A-Z)?

4. How many bits are required to specify codes for the uppercase English letters (A-Z), the lowercase English letters (a-z), and the decimal digits (0-9)?

5. How many bits are required to specify codes for the uppercase English letters (A-Z), the lowercase English letters (a-z), the decimal digits (0-9), and a dozen or so punctuation characters?



6. How many bits are in a byte?

7. How many unique codes (values) can be stored in 8 bits?

Model 2 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]

Figure 1: ASCII Table

The American Standard Code for Information Interchange (ASCII) was formalized in 1967 and is the standard set of character codes to represent English letters (uppercase and lowercase), Arabic numerals, basic punctuation, and a number of control codes.
Refer to Model 2 above as your team develops consensus answers to the questions below.

Questions (5 min) Start time:

8. How many codes are defined in ASCII? What is the range (start to end) of values?

9. How many bits are required to represent one ASCII code?

10. What are the hexadecimal codes for "ASCII"?

11. What characters are represented by the hexadecimal codes 54 61 62 6C 65?

12. Give an example of a control character (0x00 to 0x20 and 0x7F) that would commonly appear in regular text.

Model 3 Code Pages

While ASCII was fine in America (it is, after the American Standard . . .), other countries had reason to need character encoding as well. Even English-speaking countries, like England, would want to use a currency symbol ('£') that is not provided in ASCII.

Refer to Model 3 above as your team develops consensus answers to the questions below.

Questions (10 min) **Start time:**

13. Are there any codes are available in a byte that are not used by ASCII (hint: see questions 7 & 8)? If so, how many? What is their range?

In the closing decades of the 20th century codes were defined for a variety of geographical regions. ISO/IEC 8859 defined fifteen code sets or "Parts" that each keep ASCII in the lower half of the 8-bit range and in the upper half add just under 100 region-specific characters.

Part 1	Latin-1 Western European
Part 2	Latin-2 Central European
Part 3	Latin-3 South European
Part 4	Latin-4 North European
Part 5	Latin/Cyrillic
Part 6	Latin/Arabic
Part 7	Latin/Greek
Part 8	Latin/Hebrew
Part 9	Latin-5 Turkish
Part 10	Latin-6 Nordic
Part 11	Latin/Thai
Part 13	Latin-7 Baltic Rim
Part 14	Latin-8 Celtic
Part 15	Latin-9
Part 16	Latin-10 South-Eastern European

Part 1 has characters for the following Western European languages: Danish, Dutch, English, Faeroese, Finnish, French, German, Icelandic, Irish, Italian, Norwegian, Portuguese, Rhaeto-Romantic, Scottish Gaelic, Spanish, Catalan, and Swedish and is shown in the Figure 2.

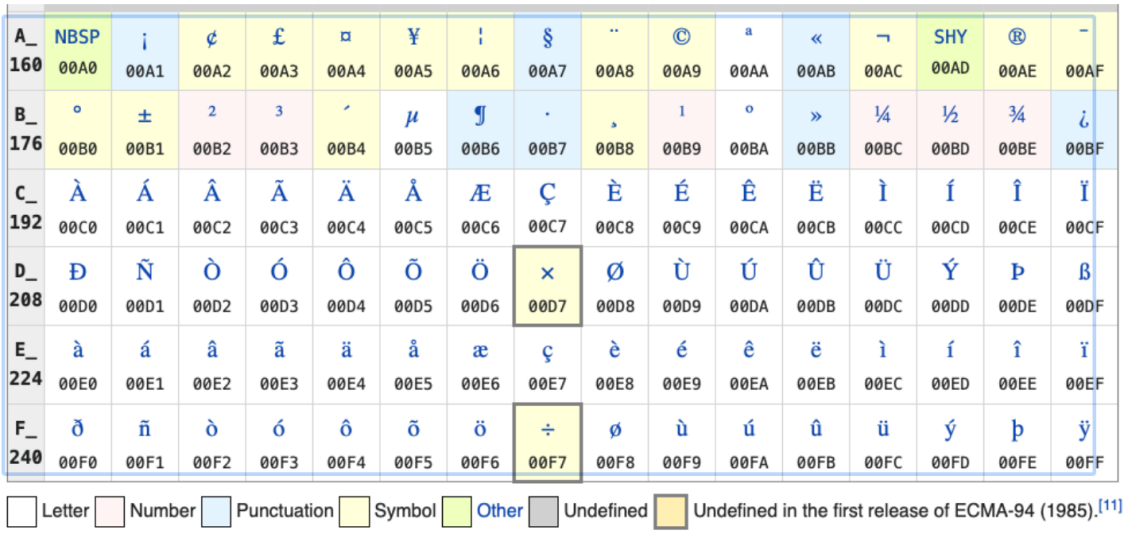


Figure 2: ISO 8859-1

14. What does the inclusion of Hebrew (Part 8) in this list say about the number of characters in the Hebrew alphabet? What is a rough estimate of the upper limit?

15. While the code page approach allows many languages to be represented, each

Part can use the same code for a different character, so 0xAA is 'Ē' in Part 4 and 'Š' in Part 10. So, while documents written in Part 8 (Hebrew) can be read by other users in Israel, the document would be gibberish to someone in the United States (using Part 1), even if that person was fluent in Hebrew because the computer would present the wrong character for a particular code. When using Code Pages, what information needs to accompany every string of text (document, email, etc.)?

16. Because each Part uses ASCII for the first 128 values, (American) English can be shared with another language (that is, Part 6 supports English and Arabic while Part 8 supports English and Hebrew). But a computer can generally run in only one mode at a time (using a specific Part to describe its character set). What problem does this present?

17. China, Japan, and Korea had as much or more need for computer support as many of the countries supported by ISO/IEC 8859 (Japan was—and is—more technologically advanced than, say, Turkey or Thailand), yet there was no eight-bit encoding for their alphabets. What characteristic of their alphabets would make 8-bit encoding impossible?

18. Because memory is typically addressed in units of a byte, data is typically coded into multiples of a byte. How many bits are in two bytes? What is the range of values that can be encoded in two bytes?

Model 4 Unicode

Unicode was created to provide a unique, unified, universal (notice the repetition of 'u'!) encoding of all characters in common use in all modern languages. The goal was to support any text used in a recently published newspaper or magazine, and this was expected to be far below 2^{14} or 16,384 so would fit comfortably in 16 bits. Not surprisingly, more and more characters were added and Unicode 14.0 (March 2020) specifies 143,859 characters, each with its own unique number or codepoint. A codepoint is denoted as U+0000 through U+10FFFF ("U+" plus the code point value in hexadecimal, prepended with leading zeros as necessary to result in a minimum of four digits, e.g., U+00F7 for the division sign, ÷, versus U+13254 for the Egyptian hieroglyph designating a reed shelter or a winding wall: 𓍊).
The first 256 codepoints match ISO/IEC 8859 Part 1 Latin-1 Western European (introduced above), so the first 128 codepoints match ASCII.

Refer to Model 4 above as your team develops consensus answers to the questions below.

Questions (30 min) Start time:

19. How many unique codes can be represented in 2 bytes (feel free to use a calculator)? Is two bytes sufficient to represent a full Unicode character?

20. Computer manufacturers tend to use powers of two for words (processor byte sizes). This allows backwards compatibility (a single four-byte register holds exactly two 16-bit values, while a three-byte register could hold only one 16-bit value). The following shows the history of Intel's CPU architecture:

a) 8-bit words in the Intel 8008 (1972)

b) 16-bit words in the Intel 8086 (1978)

c) 32-bit words in the Intel 30368 (1985)

d) 64-bit words in the Intel Pentium 4 (2004)

Which word size from the above list would be the most compact form to hold any Unicode codepoint?

21. Given a document containing approximately 10,000 ASCII characters, about how kilobytes (thousand bytes) will the file size be if stored as one-byte characters?

22. Unicode's "Base Multilingual Plane" (BMP) uses codepoints from U+0000 to U+FFFF and contains characters for almost all modern languages, including Chinese, Japanese, and Korean characters. How many bytes are required to represent a character in the BMP?

23. If each codepoint were saved as a two-byte value, about how big (in KB) would a file be that had approximately 10,000 characters? If those characters all happened to be in the ASCII range, how much space would be "wasted" with zero-filled bytes?

24. If each codepoint were saved as a four-byte value, about how big (in KB) would a file be that had approximately 10,000 characters? If those characters all happened to be in the ASCII range, how much space would be "wasted" with zero-filled bytes?

If four bytes were used to store each character, a great deal of space would be wasted, particularly in the United States where most text is ASCII (but also in the rest of the world where the BMP requires only two bytes). Unicode defines various encodings that can be used to store character's codepoint in memory. Note that there is a difference between a codepoint (the number for a character) and the encoding of the codepoint (how it is represented in memory).
As discussed in Lesson 1, a common way to get more codes from a limited range is to use an escape code. An escape code indicates that the subsequent item is to be interpreted in an alternate manner. For example, in Braille, one code isn't an actual character but is used to indicate that the character that follows is uppercase (instead of the default of lowercase).

25. If an ‘a’ takes one Braille space, how many spaces does an ‘A’ take? If approximately 2% of text is uppercase, about how many spaces would be taken by a document with 10,000 characters?

26. If you were given an offset into a Braille document for a letter that might be uppercase, what would you need to do to determine if it was uppercase or lowercase?

27. What is the range of codes used by the BMP (hint: see question 22)?

28. What is the range of codes for all of Unicode (hint: see the beginning of this section)?

29. What is the range of codes not in the BMP?

30. What is the size of the non-BMP range in hexadecimal? (Hint: max – min + 1)

31. How many bits are required to represent the non-BMP range (not codepoints)?

32. Unicode reserves 2048 codepoints in the BMP (from 0xD800 to 0xDFFF) that are not used as characters but as surrogates for characters outside the BMP. Convert the minimum and maximum surrogate codepoints to binary:

33. What bit values and positions indicates that a codepoint is not a character but a surrogate?

34. In a two-byte word, how many bits does that leave for the surrogate (non-BMP) codepoint?

35. If a non-BMP codepoint were stored in two surrogate words, we could use one bit to indicate whether this is the first (0) or second (1) surrogate word. How many bits per surrogate does that leave for the non-BMP codepoint? If two surrogates are used to encode a non-BMP codepoint, how many bits are available? How does this compare to the number of bits needed (see question 31)?

UTF-16 is a variable-length Unicode encoding that uses 16 bits for BMP codepoints (including ASCII), storing the codepoint as data, and 32 bits for non-BMP codepoints, breaking the codepoint over two surrogate words. It is used internally by Java, JavaScript, and (until recently) Microsoft Windows.
A similar variable-length approach has been used to define UTF-8.

Layout of UTF-8 byte sequences

Number of bytes	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	U+0000	U+007F	0xxxxxxx			
2	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	U+10000	[nb 2]U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Figure 3: UTF-8 Encoding

36. In UTF-8, how many bytes are required to encode the full range of ASCII? (Hint: see question 8.)

37. Recall that ISO 8859-1 defines one-byte character codes that support much of Western Europe. In UTF-8, how many bytes are required to encode non-ASCII codes from ISO 8859-1?

38. In UTF-8, how many bytes are required to encode the majority of BMP codepoints?

39. In UTF-8, how many bytes are required to encode non-BMP codepoints?

40. Is there any size difference between encoded ASCII and the same characters in UTF-8?

41. If most of your characters were ASCII, which (if any) would be more compact, UTF-8 or UTF-16?

42. If most of your characters were Western European, which (if any) would be more compact, UTF-8 or UTF-16?

43. If most of your characters were East Asian (Chinese, Japanese, or Korean), which (if any) would be more compact, UTF-8 or UTF-16?

44. In UTF-8, what distinguishes a single-byte character from a multi-byte character?

45. In UTF-8, what distinguishes the first byte of a multi-byte character from the subsequent bytes?

46. If you were given an arbitrary byte offset into a UTF-8 encoded string, how would you determine the number of bytes?

UTF-8 accounts for the vast majority of all web pages (over 95%) and is the default for use in Microsoft Windows (as of 2019).