Till A. Heilmann

Reciprocal Materiality and the Body of Code. A Close Reading of the American Standard Code for Information Interchange (ASCII)

2015

https://doi.org/10.25969/mediarep/678

Veröffentlichungsversion / published version
Zeitschriftenartikel / journal article

Empfohlene Zitierung / Suggested Citation:
Heilmann, Till A.: Reciprocal Materiality and the Body of Code. A Close Reading of the American Standard Code for Information Interchange (ASCII). In: *Digital Culture & Society*, Jg. 1 (2015), Nr. 1, S. 39–52. DOI: https://doi.org/10.25969/mediarep/678.

Erstmalig hier erschienen / Initial publication here:
http://digicults.org/files/2016/11/I.2-Heilmann_2015_The-body-of-code.pdf

Reciprocal Materiality and the Body of Code

A Close Reading of the American Standard Code for Information Interchange (ASCII)

Till A. Heilmann

Abstract

Materiality has often been a neglected factor in discussions of digitally encoded information. While a lot of early works in media studies suffered from this shortcoming, questions regarding the materiality of digital technology and artefacts have slowly gained prominence in recent debates. Matthew Kirschenbaum's concept of "forensic" and "formal" materiality has proven particularly useful to the study of digital artefacts, differentiating the (routinely overlooked) physical existence of digital data from their (commonly discussed) logical character. However, analyses concerning the materiality of digital artefacts are often one-sided, focussing on the physicality of the medium in which digital data are inscribed. To counter this bias, I present the concept of a 'reciprocal materiality' of digital data: It is not only that digital data are always inscribed in some material substrate (Kirschenbaum's 'forensic' dimension of data); conversely, the materiality of the medium inscribes itself into the structure of digital data (its 'formal' level). The 'body of code' is shaped by the material framework it inhabits. I will illustrate this using as an example one of the most important encoding schemes in the history of digital technology: the American Standard Code for Information Interchange (ASCII). A 'close reading' of the technical specifications of ASCII – a standard designed in the early 1960s to work across multiple technological platforms – will reveal the extent to which this code incorporates the materiality of media such as punched tape and teletype terminals.

Introduction

At least since Friedrich Kittler (1997) infamously claimed that "There Is No Software", the materiality of digital media has been deemed one of the main theoretical and empirical objects of media studies. Still, in-depth analyses of computing hardware (from a media theoretical or historical perspective) have

DOI 10.14361/dcs-2015-0104

been the exception rather than the rule.[1] In fact, the field of media studies has recently seen a move away from hardware-centric considerations of specific platforms, machines and components, and towards the investigation of algorithms, codes and applications in what is called Software Studies by its proponents (cf. Fuller 2008; Manovich 2013).

Probably the most compelling account of the material character of digital data has been given by Matthew G. Kirschenbaum. In his 2008 landmark monograph *Mechanisms. New Media and the Forensic Imagination*, Kirschenbaum introduced the distinction between "formal" and "forensic" materiality in order to describe the peculiar dual nature of digitally encoded and stored data: On the one hand, bits are physical marks inscribed in media; on the other hand, they serve as bodiless symbols in the process of computation. "Formal materiality" is the name Kirschenbaum gives to the latter phenomenon and to those principles and properties commonly associated with computers and new media. Describing matter(s) on the level of abstract symbol manipulation, formal materiality designates the seemingly "*immaterial* behavior" of digital technology (Kirschenbaum 2008: 11). The illusion of immateriality, however, is grounded in what Kirschenbaum calls the forensic materiality of digital technology: the particular configuration of hardware and the concrete existence of the software stored, transmitted and processed by the hardware. On the microscopic level of forensic materiality, digital data are distinctive, physical marks, each mark an individual and unique inscription in a given medium (cf. Kirschenbaum 2008: 61-63). The forensic materiality of digital data manifests itself in diverse forms (nanometre scale strips of electromagnetic flux reversals on metal platters in hard disk drives, voltage levels in the logic gates of solid-state drive transistors etc.) but these forms are always, by necessity, irreducibly physical facts.

Kirschenbaum's distinction between formal and forensic materiality is a major conceptual contribution to the study of digital media, and his analyses of computer games and interactive fiction stored on hard and floppy disks are highly original. Nevertheless, Kirschenbaum's modelling of materiality suffers, I think, from a small but important limitation. In *Mechanisms*, the relation between formal and forensic materiality is portrayed as asymmetrical. Formal materiality results from forensic materiality. Physicality – the 'actual' materiality of digital technology – is located on the level of forensic materiality, whereas materiality of the formal kind is an "abstract projection" or "illusion" (cf. Kirschenbaum 2008: 11). Furthermore, the (illusionary) formal existence of digital data is unaffected by the forensic character of their physical reality (a bit only ever has the logical or numerical value of 0 or 1, no matter in what form the bit is actually stored, transmitted and processed). Kirschenbaum's conceptual distinction between formal and forensic materiality suggests a factual separation of the two dimensions. Foregrounding the physical reality of forensic materiality, formal materiality appears as an illusionary phenomenon; foregrounding

1 See, for example, Dennhardt (2009), and the books in the Platform Studies series by Montfort and Bogost (2009) and Maher (2012).

the logical reality of formal materiality, forensic materiality appears as a mere medium of inscription. Either way, data are 'actually' or 'ultimately' inscribed into matter and the (forensic) materiality of the storage medium serves as a 'passive' recipient for the inscriptive act.

But what if we consider the reverse possibility? What if, in Kirschenbaum's terms, the formal materiality of digital data is not simply an abstraction but also (at least in part) a rather direct reflection of data's forensic materiality? What if digital data are not only recorded as a series of physical marks in some material substrate but are themselves 'marked' by the materiality of their technological framework and media? What if the (forensic) materiality of digital technology has a determining influence on the (logical) forms of data? I will argue that the relation between the actual physical existence of digitally encoded and stored data and the logical form of data is symmetric in so far as the materiality of digital technology acts a medium of inscription both passively and actively. Code is inscribed into materiality and materiality, conversely, inscribes itself into code. This is what I call reciprocal materiality. The 'body' of code, accordingly, means two distinct but related aspects of the same thing: the code's physical form of inscription (what Kirschenbaum calls forensic materiality) and its logical form of representation (i.e. the overall structure and the individual elements of the code as a system). By 'code', I refer, in the most general way, to any kind of data that is digitally encoded for and processed by computing machinery, as well as to any norm that controls or regulates the encoding and processing of data, i.e. to "raw data", network protocols, character sets, file formats, program source code etc.

In the remainder of the paper, I will illustrate my argument through a close reading of one of the most influential codes in the history of computing: the American Standard Code for Information Interchange, better known under its acronym ASCII. This code seems to be especially suitable for an analysis of reciprocal materiality because ASCII was invented, as the name indicates, for the exchange of data among different communication and processing systems and, consequently, was designed to abstract as much as possible from any particular machine. Also, its creators started from scratch, so to speak, and ASCII was not made to be backward compatible with earlier codes (and the requirements of their machinery, respectively). Therefore, one might think that the 'body' of ASCII would be mostly unaffected by the functional and physical characteristics of hardware. However, our analysis will reveal a considerable influence of the materiality of digital technology on the code's structure and content.

The American Standard Code for Information Interchange

ASCII is a character-encoding scheme, i.e. a standardized way of indicating characters from a predefined repertoire by using code numbers or bit patterns. Developed at the beginning of the 1960s by a committee with representatives from large US computer and telecommunication companies like IBM, NCR,

Bell and RCA, and the Department of Defense, the first version of ASCII was approved and published by the American Standards Association in 1963, with a major revision in 1967 (cf. American Standards Association 1963; Smith 1967). Although it would become nearly ubiquitous in the world of mini- and microcomputers, ASCII was not designed "necessarily for internal use in information processing equipment" (American Standards Association 1963: 10) or for interfacing with computer operators. Its original goal was to mediate between the variety of incompatible character encodings in use for data processing and for telegraphic communication systems at the time.[2] The propagation of ASCII as a computer code leading to its dominance in the PC sector coincided with several momentous technological changes: the advent of commercial transistorized minicomputers in the 1960s, the triumph of microcomputers for home and business users from the late 1970s on, and the global spread of the Internet in the 1980s and 1990s (cf. Ceruzzi 2003: 133, 152). Only since the early 2000s, ASCII is being superseded by the universal Unicode system.

In the first five decades of its existence, ASCII has seen a lot of computing hardware come and go. This pertains especially to the media of storage, which have evolved from the punched cards of the early days to the vast disks arrays and tape libraries in today's data centres. ASCII encoded data has been recorded on hard drives, flash drives, DVDs and CDs, ZIP disks, floppy disks and (through formats like the Kansas City standard) even on compact cassettes. When the standard was first specified at the beginning of the 1960s, however, the principal storage medium was perforated paper tape (cf. American Standards Association 1963: 3). The use of punched tape (see fig. 1) for controlling mechanical and electromechanical devices originated from automatic telegraphs and typesetting machines in the late 19th century and the practice was adopted for computers with the very first machines in the 1940s and 1950s. Before raster graphics and video displays became the norm in the late 1970s, the typical interface to a computer system was a terminal device: a specially outfitted teleprinter or electric typewriter. Devices like the Teletype Model 33 ASR (see fig. 2), an ASCII-compatible teleprinter and one of the most popular computer terminals in the 1960s and 1970s, also had tape punches and readers for automated input and output of data. The early materiality of ASCII encoded data was not one of pixels on high-resolution screens, invisible electromagnetic tracks on hard disks and modern short travel keyboards but one of paper, chad, ink, the loud clacking of printing mechanisms, the smell of hot oil and ozone and the whirring noise of electric motors.

2 IBM, General Electric/Honeywell and Burroughs used (mutually exclusive) variants of the Binary-Coded Decimal (BCD) scheme derived from IBM's punched card code while UNIVAC computers operated on a custom version of the military FIELDATA code. Telecommunication companies used ITA2 (derived from Emile Baudot's 5-bit telegraphic code) for their teleprinters.

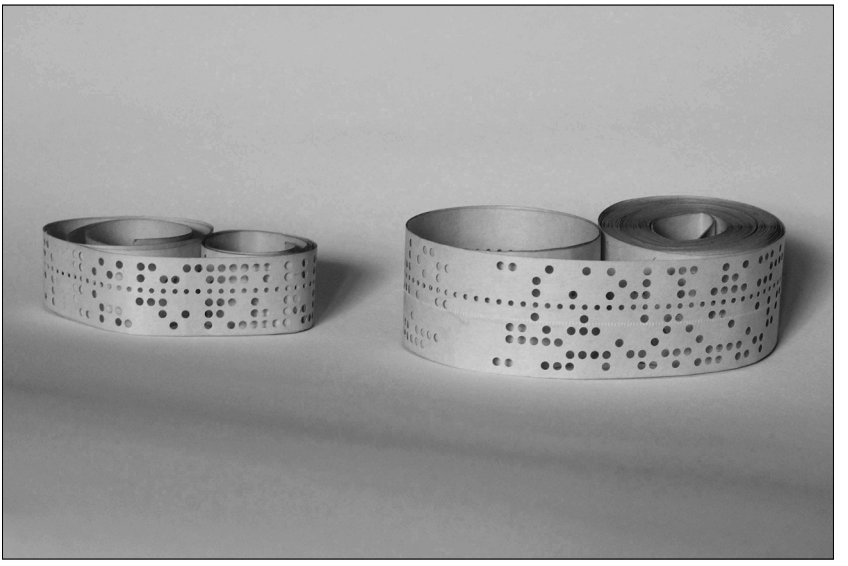

Fig. 1: Punched paper tape (five and eight hole) (http://en.wikipedia.org/wiki/Punched_tape#/media/File:PaperTapes-5and8Hole.jpg; Public Domain)

Fig. 2: Teletype Model 33 ASR (http://commons.wikimedia.org/wiki/File:Teletype-IMG_7292.jpg; CC ASL 2.0)

Set Size

Let us now take a closer look at ASCII and investigate the 'body' of the code in more detail. The documents I will refer to are the first edition of ASCII from 1963 (American Standards Association 1963), the commentary by Fred W. Smith (an engineer with Western Union and member of the ASA committee that developed ASCII) on this first edition (Smith 1964) and Smith's notes on the revised version of the standard from 1967 (Smith 1967).

ASCII is a character set encoded with seven bits, allowing for 27 permutations of the bit pattern or 128 code points (see fig. 3). The unit length of the code, also called the "set size", is a first hint of the power of reciprocal materiality. Other set sizes would have been possible and the committee originally made plans for a unit length of six or of eight bits. While many early encoding schemes, like IBM's BCD, used only six bits (or even just five, like the code used in Lyon's LEO, the world's first commercial computer), this set size was considered too small. Because ASCII was designed for communications and information interchange, it had to incorporate special characters to control the operation of telecommunications equipment like 'carriage return', 'line feed' and 'horizontal tab' (cf. American Standards Association 1963: 7). Six bits were not enough to code such characters in addition to the alphanumerics and other regular symbols without resorting to the use of a 'shift' character signalling a switch between two different interpretations of the same code points (as it was done with ITA2, a predecessor of ASCII; cf. Smith 1964: 51). A set size of eight bits, on the other hand, was rejected because "it provides far more characters than are now needed in general applications" (American Standards Association 1963: 7). But this is only half the truth. According to Smith, "it was decided that an 8-bit code would be too wasteful for most users" (1964: 51). Before advances in the fabrication of integrated circuits made large primary memories possible (and affordable) in the 1970s, the memory capacity of computing machines was small, typically comprising only a few hundred kilobytes.[3] Every single bit was precious and not to be wasted. Restricting the set size of ASCII to seven bits reflects this crucial material constraint of computing in the 1960s.[4]

3 The IBM 704, the company's first mass-produced computer with magnetic core memory introduced in 1954, initially had 147'456 bits or 18 kilobytes of memory (eventually extended to just over one hundred kilobytes). For comparison: Apple's iPhone 6 starts at 16 gigabytes of RAM.

4 Also, saving one bit allowed the use of an additional eighth bit either for internationalization of the code or as a parity check bit, a very simple form of error detecting (cf. Smith 1964: 55).

USASCII code chart

Bits b_4	b_3	b_2	b_1	Row ↓ / Column →	0 (000)	1 (001)	2 (010)	3 (011)	4 (100)	5 (101)	6 (110)	7 (111)
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Fig. 3: Revised US American Standard Code for Information Interchange, X3.4 – 1967 (http://commons.wikimedia.org/wiki/File:ASCII_Code_Chart-Quick_ref_card.png; Public Domain)

The Graphic Subset

After the set size, the structure of ASCII is the next facet to reveal more about the code's technological framework at the time of its specification. ASCII consists of two character subsets: one for "graphics", i.e. printing characters like alphanumerics, and one for "controls", i.e. non-printing characters used for inband-signaling.

The graphic subset is the part of ASCII that regular computer users of today will be familiar with – simply because it is that part of the code you can easily input on a keyboard and then see on the screen (or printed on paper). The graphics originally comprised a total of 64 characters: the basic English letters A through Z, the Arabic digits 0 through 9 and those symbols for punctuation, mathematical expressions and business use (like comma, period, plus, minus, the dollar and per cent sign and the ampersand) which the committee considered "most useful" (American Standards Association 1963: 7). While the first edition of ASCII from 1963 specified only one case of letters (rendered as uppercase by teleprinters and typewriters) and left twenty-eight code positions unassigned, the revised version from 1967 extended ASCII to cover both uppercase and lowercase letters (cf. American Standards Association 1963: 6; Smith 1967: 186-187). The graphics also include the "word separator" or space character, which normally is not printed but, since it occupies an area on a

printed page, counts as a printing character. In short: ASCII's graphics are (with a few exceptions) the characters one would find on the keys of a standard QWERTY keyboard from the 1950s. Of the 95 graphics in the code, 83 make up the complete set of symbols in the traditional typewriter keyboard layout.[5] Shortly after the first version of the code was finalised, teleprinters and terminal devices with ASCII-compliant keyboards came to the market, in particular the aforementioned Model 33. The main material influence on ASCII's graphics, one can conclude, is the keys one presses on a typewriter terminal or teleprinter to generate the according glyphs.

While this statement may seem obvious (or even outright trivial), a few remarks on the graphics subset of ASCII are in order. First, the code's printing characters do not belong to just any keyboard layout. They match a US keyboard layout (ASCII is, after all, an *American* code). Consequently, 7-bit ASCII as it was specified in 1963 and revised in 1967 does not know about German umlauts, French accented letters,[6] the Scandinavian Å, Spanish inverted question and exclamation marks etc. This, expectably, led to problems when ASCII was adopted for non-English alphabets, which require additional characters. The committee had anticipated this difficulty and provided several options to adapt ASCII for national uses outside the US according to an ISO standard (unspecified at the time) that would not change the set size or compromise the interoperability of the code.[7] These officially sanctioned methods proved insufficient and unpopular, however, and the original 7-bit version of ASCII was soon extended

5 The exceptions are the "less than" and "greater than" symbols < and >, the "brackets" [and], the "up arrow" (for the mathematical operation of exponentiation), the "left arrow" (for the logical connective "implies"), the "reverse slant" \ (to form, together with the slash /, the boolean operators \/ and /\), and, since the 1967 revision of ASCII, the "circumflex" ^, the "overline" ~, the "vertical line" | (for the logical operator "or" or to designate absolute values) and the "braces" { and } (probably to code the ALGOL words "begin" and "end"; cf. American Standards Association 1963: 6, 8; Bemer 1959, 1972: 20; Smith 1967: 187).

6 French accented letters can be constructed using 7-bit ASCII by combining the apostrophe, grave and circumflex graphic with a vowel (cf. Smith 1967: 186).

7 The 1963 specification mentioned the five graphics immediately following the letter Z (i.e. the brackets, reverse slant and up and left arrow) and the code point right before the letter A (i.e. the @-sign) as candidates for character substitution in European alphabets while the two graphics following the digit 9 (the colon and semicolon) could be replaced by 10 and 11 for "use of the Sterling monetary system or duodecimal arithmetic" (cf. American Standards Association 1963: 10). The 1967 revision of ASCII reserved the graphics @, [, /,], {, | and } for special "national use", permitted the interpretation of the graphics ", ‘, , ("comma"), `, and ~ as diacritical marks (diaresis, acute accent, cedilla, grave accent, and tilde), allowed the "number sign" (#) to be used as the pound sign (£), and declared the "circumflex", "grave accent" and "overline" graphics as code points for substitution by additional characters (cf. Smith 1967: 186-187). More generally, the control characters "shift in" and "shift out" were introduced to change to an alternate set of graphics (cf.

into many language- or platform-specific 8-bit variants (using the eighth bit to designate 127 additional characters), which were incompatible with each other.[8]

The placement of the graphics and their internal order also tell us about the code's materiality. Both the 1963 original edition and the 1967 revision organized ASCII's 128 code points into a table with 16 rows (identified by the four "low order" bits) and 8 columns (identified by the three "high-order" bits). The ideal would have been to place the graphics subset and the control subset side by side as two 'dense' code blocks of four adjacent columns each and to have the non-printing characters in the first four columns of the table. This way, a simple check could have been made on bit 7 to determine whether a character belonged to the graphic subset or not (cf. American Standards Association 1963: 7). Smith (1964: 53) notes that this examination would have only required "relatively simple circuitry", stressing the importance of ASCII processing hardware. But consideration of another rather material aspect of ASCII processing rendered this solution impossible. The very first code point (at the beginning of the first column) and the very last one (at the end of the eighth column) had to be reserved for the non-printing "null" and "delete" characters. The reason for this was the most common storage medium at the time: perforated paper tape. On tape, bits are indicated by the presence or absence of holes in the paper. A bit value of one corresponds to a punched hole while a bit value of zero corresponds to 'no hole'. The first code point, then, is the 'all-zeroes' character represented by blank paper and the last code point is the 'all-ones' character with all seven holes punched. Assigning the binary code 0000000 to the "null" character (which basically means 'do nothing' or 'ignore') permitted "continuing the traditional use of blank perforated tape as a leader at the beginning of a message" (Smith 1964: 53). Assigning 1111111 to the "delete" character was simply a necessity: On perforated tape, there is no way to correct a wrong character since one cannot 'un-punch' an erroneously punched hole. But one can always 'overwrite' any character's bit pattern on the tape by punching all holes, thus 'rubbing out' wrong characters. Because the "null" and "delete" characters were non-printing and had to be put in the first and last column respectively, it was decided to place the subset of printing characters in the four middle columns of the code table.[9] (In the revised version of ASCII from 1967, the graphic subset stretches into the last, eighth column containing the "delete" character, thereby violating the principle of a 'dense' block of printing characters only.) The materiality of paper tape, in other words, dictated the use of the first and the last code point in ASCII (and, by consequence, the placement of the graphic subset).

Smith 1964: 54) but they would also be used to change the typeface of the printer or the color of the typewriter ribbon.

8 For example, Atari home computers used ATASCII, Commodore's 8-bit machines used PETSCII, IBM introduced dozens of national "code pages" for the PC platform, and the ISO published a 16-part encoding scheme to cover all major writing systems except the scripts of East Asian languages (Chinese, Japanese, and Korean).

9 This still provided a fairly easy way to identify the graphic subset because in each of the four middle columns, bit 7 is different from bit 6 (cf. Smith 1964: 53).

Next, the graphic subset's arrangement is meaningful in some non-trivial ways. Unsurprisingly, the digits were coded in the sequence of the natural numbers they represent (including the "0" which comes before "1", not after "9" as on typewriter keyboards) and the letters were arranged in alphabetical order. Collating and sorting of ASCII encoded data thus translates into straightforward comparisons of binary codes easy to implement in hardware and software (cf. American Standards Association 1963: 8; Smith 1964: 53). To conform with conventional collating practices, common word separators like the space, comma and period were put before digits and letters (so that "Johns" comes before "Johnson", and "West, W." before "Weston."; cf. Smith 1964: 53). Also, the digits were placed so as to form a 4-bit numeric subset of 'natural' binary coded decimals (where each digit is coded separately with the binary representation of its decimal numeric value, i.e. "0" is represented by 0000, "1" by 0001, "2" by 0020, "123" by 0001 0010 0011 and so on).

While none of this betrays physical needs or aspects of ASCII processing, the arrangement of digits, letters and additional symbols in relation to each other is significant. As mentioned previously, the structure of ASCII was influenced by "the needs of typewriter-like devices" (American Standards Association 1963: 8). The design of the code sought to facilitate its implementation in hardware. In particular, this meant that there is "only a common 1-bit difference between characters normally paired on keytops" (American Standards Association 1963: 8). In ASCII's original edition from 1963, the bit-pairing of characters mainly affected the digits and matching symbols as they appeared on the top row of US mechanical typewriter keyboards: 2/", 3/#, 4/$ etc. When the revised version of 1967 added lowercase letters, these were naturally bit-paired with their uppercase complements. For modern electronic keyboards, the relation between code points of paired characters is less relevant. The typewriter terminals and teleprinters of the 1960s, however, had to implement character shifting by electromechanical means. With paired characters shifted by exactly one or two columns in the code table, pressing the shift key on the keyboard could simply toggle the appropriate bit (bit 5 for digits and symbols, bit 6 for uppercase and lowercase letters).[10]

The Controls Subset

Like the graphics, the controls subset of ASCII is deeply influenced by the materiality of its supporting framework – maybe even more so. This is almost self-explanatory: Controls are defined as those characters required "for the control of terminal devices, input and output devices, format, or transmission and switching" (American Standards Association 1963: 8). They are a direct reflec-

10 Not all characters in ASCII were shifted according to the traditional US typewriter layout. The placement of the "space" character and the digit "0", among others, prevented a complete match between typewriter keytops and bit-paired characters (cf. American Standards Association 1963: 8).

tion of the hardware ASCII-encoded software 'inhabits'. The controls were placed in the first two columns of the code table and divided into four groups (communication controls, format effectors, device controls and information separators) with nine "miscellaneous" characters belonging to no group (cf. Smith 1967: 188). The structure of the controls subset and the names and definitions of some characters changed considerably from the first edition in 1963 to the revised version of ASCII in 1967. The following remarks refer to the 1967 version.

Control characters are not unique to ASCII. They were invented for automatic telegraph systems. Probably the first control characters in history were the aforementioned "null" and "delete" which stem from the code that French telegraph operator Émile Baudot designed for his printing telegraph patented in 1874 (cf. Heath 1972: 83; Beauchamp 2001: 394-395). When Baudot's code was adapted by New Zealand journalist Donald Murray at the beginning of the 20th century for use with his new tele-typewriter system (the prototype of commercial teleprinters), Murray added two more controls: "carriage return" and "line feed". This was necessary because the codes received in Murray's system were no longer printed in one continuous line on paper tape but were automatically translated into graphics and printed on a roll of standard typewriter paper. Something had to signal to the printing mechanism that the end of a line of text had been reached and that the typewriter's carriage should now return to the right and the paper should be pushed up so the printing of the message could continue on a new line. Thus, the materiality of standard typewriter paper brought about a new class of codes.

Western Union bought the rights to Murray's system and used his code with a few modifications for its global communications network until the 1950s. In 1931, it was standardized by the International Telecommunication Union as the International Telegraph Alphabet No. 2 (ITA2) (cf. Mackenzie 1980: 62-64). ASCII inherited the "null", "delete", "carriage return" and "line feed" controls from ITA2 and added some more to deal with the increased complexity of communications systems and devices. The transmission or communication controls (e.g. "enquiry", "acknowledge", "negative acknowledge", "end of transmission") show no signs of influence from the materiality of the equipment. Rather, they embody some fairly general principles of telecommunication. The same holds for the information separators, which were intended to indicate the logical structure of data as assembled into "files", "groups", "records" and "units".

Of more relevance to questions of materiality are the device controls and format effectors. While the exact use of the four device control characters was never defined, it seems they were intended to turn auxiliary equipment like tape readers and punches on teleprinters on and off (cf. Smith 1964: 54; Russell 1989: 3). The most direct reflection of ASCII's material environment shows in the group of so-called format effectors. These controls are those characters "[...] which are used to organize printed data on a page, such as carriage return, line feed, and vertical tabulation [...]" (Smith 1964: 54). The functions of the "carriage return" and "line feed" characters have already been discussed. "Horizontal tabulation" moved the carriage (or print head) to the next tab stop, usually

defined as every eighth character in a row.[11] This control was used mostly to facilitate the display of data in tables (it also saved valuable memory space since it could replace several "space" characters). "Vertical tabulation" advanced the paper by a specified number of lines and "form feed" advanced the paper to the beginning of the next page. Apart from the (obvious) case of graphic characters like letters and digits, this group of control characters is where the materiality of the printed page is inscribed most clearly in the body of ASCII.[12] Two of the unclassified "miscellaneous" controls also mirror some material aspects of computing in the early 1960s: As its name implies, "bell" was designed to ring a bell (or sound an alarm) on the receiving teleprinter device to alert the operator. Equally telling is the name of the "end of medium" character which signals that there is no place left on the physical medium of storage (e.g. punched tape).

Today, most of ASCII's 32 control characters are obsolete. But some remain in use. Of the format effectors, "carriage return" and "line feed" still terminate lines in text files,[13] and "horizontal tabulation" designates a move to the next tab stop. In Unix systems, a few communication controls are still meaningful: "End of text" (input by pressing C while holding down the <Ctrl> key) interrupts the job currently running in the foreground of a terminal, "end of transmission" (<Ctrl>-d) commonly exits a program or shell and "negative acknowledge" (<Ctrl>-u) deletes all characters from the cursor back to the beginning of the line. Of the "miscellaneous" controls, "substitute" (<Ctrl>-z) pauses the current job in Unix shells and "bell" (<Ctrl>-g or echo -e \a on the command line) is still interpreted by most terminal emulations as an acoustic or visual signal.

Conclusion

ASCII is not an abstract representation of 'pure' information (as is commonly assumed of digital codes), not only an 'immaterial' model of an idealized writing system. It is also a testament to significant material conditions of early digital technology like sparse primary memory (with its 7-bit set size and space-saving characters like "horizontal tabulation"), uncomplicated and inexpensive circuitry (with its structure for easy identification of the graphics and controls subsets), punched tape and cards (with the "null" and "delete" characters represented by 'all-zeroes' and 'all-ones' codes, and the "end of medium" control),

11 In card systems, "horizontal tabulation" skipped the current card.

12 "Backspace" which moves the carriage (or head) of the printing mechanism one position back is also classified as a format effector. It was not only meant to 'rub out' a previously typed character (by 'overwriting' it with the "delete" character). It could also be used to add diacritic marks (like the "acute", "grave accent" or "circumflex") or to emphasize words (with "underline" characters).

13 Different platforms handle these controls differently. Windows and MS-DOS use the "carriage return" (CR) in combination with the "line feed" (LF), Unix uses just the LF character and classic Mac OS and many 8-bit home computers used just the CR character.

typewriter keyboards (with its set of graphics for basic English letters and its shift-paired characters) and printed pages (with its format effectors like "carriage return" and "line feed"). The reciprocal materiality of digital technology shaped the 'body' of the code.

Since the days of ASCII's original design at the beginning of the 1960s, the technological framework supporting the code has undergone a drastic transformation. Meanwhile, the code has changed (in its physical form) and remained unchanged (in its logical form) at the same time. Teletypes, transistorized minicomputers and perforated paper tape are a thing of the past. And although it is still used in the US version and many extended variants (in network protocols like HTTP and the Internet's Domain Name System, for example), ASCII too is being superseded slowly but surely by Unicode. For reasons of backward compatibility, however, ASCII is preserved within the new standard. The first 128 of Unicode's 1,114,112 code points correspond exactly to the 7-bit code space of the 1967 revision of ASCII. That is why, even on the most modern computers, you can still, by calling code point 7 of Unicode, ring a teleprinter's "bell".

References

American Standards Association (1963): "X3.4-1963 American Standard Code for Information Interchange", (http://worldpowersystems.com/projects/codes/X3.4-1963/).

Beauchamp, Ken (2001): History of Telegraphy, London: The Institution of Electrical Engineers.

Bemer, Robert W. (1959): "A Proposal for a Generalized Card Code for 256 Characters." In: Communications of the ACM 2/9, pp. 19-23.

Bemer, Robert W. (1972): "A View of the History of the ISO Character Code." In: Honeywell Computer Journal 6/4, pp. 274-286.

Ceruzzi, Paul E. (2003): A History of Modern Computing, 2nd ed., Cambridge: MIT Press.

Dennhardt, Robert (2009): Die Flipflop-Legende und das Digitale, Berlin: Kadmos.

Fuller, Matthew (ed.) (2008): Software Studies. A Lexicon, Cambridge, MA: MIT Press.

Heath, F. G. (1972): "Origins of the Binary Code." In: Scientific American 227/2, pp. 76-83.

Kirschenbaum, Matthew G. (2008): Mechanisms. New Media and the Forensic Imagination, Cambridge, MA–London: MIT Press.

Kittler, Friedrich (1997): "There Is No Software." In: John Johnston (ed.), Literature, Media, Information Systems, Amsterdam: G + B Arts, pp. 147-155.

Mackenzie, Charles E. (1980): Coded Character Sets. History and Development, Reading, MA: Addison-Wesley.

Maher, Jimmy (2012): The Future Was Here: Commodore Amiga, Cambridge, MA: MIT Press.

Manovich, Lev (2013): Software Takes Command, New York: Bloomsbury Academic (http://issuu.com/bloomsburypublishing/docs/9781623566722_web?e=3257035/4651685).

Montfort, Nick/Bogost, Ian (2009): Racing the Beam: The Atari Video Computer System, Cambridge, MA: MIT Press.

Russell, David (1989): The Principles of Computer Networking, Cambridge: Cambridge University Press.

Smith, Fred W. (1964): "New American Standard Code for Information Interchange." In: Western Union Technical Review, April, 50-58.

Smith, Fred W. (1967): "Revised U.S.A. Standard Code for Information Interchange." In: Western Union Technical Review, November, pp. 184-191.