The TRUTH *of the* TECHNOLOGICAL WORLD

Essays on the Genealogy of Presence

FRIEDRICH A. KITTLER

With an Afterword by Hans Ulrich Gumbrecht

Translated by Erik Butler

STANFORD UNIVERSITY PRESS

Stanford, California

16 There Is No Software

"The Eastern world is exploding," Barry McGuire sang. The first time, he did so in the wild 1960s—to talk all his friends (via vinyl or eight-track tape) out of the belief that we are *not* standing on "The Eve of Destruction." The second time, after a brilliant electronic remake that made his vinyl track into the digital chart-topper of the Armed Forces Network in Dharan, his words (via ultra short waves) sought to talk the warriors of Operation Desert Storm out of the belief that they—or we—still face destruction . . .[1]

McGuire (or rather the digital signal processor that erased his phonographically immortalized negation without a trace) was still right the second time, but only because explosions did not count at all. It is unimportant whether oil derricks or Scud missiles (those direct descendants [*Reichsunmittelbare Enkel*] of the V-2) fly into the air. The East can go ahead and explode. All that matters is what happens in the Western world at the present moment: first and foremost, the implosion of high technologies—and as a result, the implosion of a set of signifiers [*Signifikantenszene*] that otherwise would still be called "World Spirit" [*Weltgeist*]. Without computing technology, there would be no deconstruction, Derrida declared in Siegen. Writings and texts no longer exist in perceptible times and spaces, but rather in the transistor cells of computers. And since, in the last three decades, the heroic deeds of Silicon Valley have managed to reduce the dimensions of transistor cells to the submicron level (that is, to less than a micrometer), our present-day scene of writing can only be described by way of fractal geometry: as self-similarity of letters over some six decades that extend from corporate billboards the size of a house down to a bitmap the size of a transistor. At the alphabetic beginnings of history itself, a mere two and a half decades separated a camel and the Hebrew characters designating the animal. Now that all signs have been miniaturized to a molecular scale, the act of writing has vanished.

As everyone knows, even if no one wants to say it, nobody writes anymore. Writing—that peculiar kind of software—long toiled at the incurable confusion between use and reference [*Gebrauch und Erwähnung*]. Up to, and including, the time when Friedrich Hölderlin composed his hymns, the mere mention of lightning was evidence enough that it might be used for poetry.[2] Today, in contrast, after the transformation of this same lightning into electricity, human writing occurs through inscriptions that are not just burned into silicon by means of electron beam lithography, but rather—and in contrast to all writing implements of history—are themselves able to read and write.

The final act of writing in history, then, may have occurred in the late 1970s, when a team of Intel engineers, under the direction of Dr. Marcian E. Hoff, spread a few dozen square meters[3] of drawing paper on the floor of an empty garage in Santa Clara and drew up the hardware architecture for the first integrated microprocessor. In a second step—mechanical this time—the manual layout of two thousand transistors and their connections was reduced to the size of a thumbnail on a real chip. Third, electro-optical equipment wrote the design onto silicon. Fourth, after the end product—the 4004, which has provided the prototype for all microprocessors ever since—was employed in the new adding machine of Intel's Japanese client, our postmodern scene of writing could begin.

Meanwhile, given the complexity of hardware in present-day microprocessors, manual design techniques no longer have a chance at all. In order to develop the next generation of computers, engineers do not use drawing but rather Computer-Aided Design: the geometrical capacities of the most recent generation of calculators are just enough to map out the topology of their successors. In this way, "the feet" of those who "will carry you out also" once more "stand at the door" (as the biblical phrase has it).

All the same, Hoff's primitive blueprints provided an almost perfect example of a Turing machine. Since Turing's 1937 dissertation, every act of calculation—whether performed by human beings or by machines—can be formalized as a countable quantity of commands that work through an infinitely long band of paper and its discrete signs. Turing's concept of this kind of paper machine,[4] whose operations encompass only writing and reading, movement forward and backward, has proven the mathematical equivalent of all calculable functions and seen to it that machinic, literal meaning [*der maschinelle Wortsinn*] has pushed aside the innocent professional designation of "computer."[5] Universal Turing machines need only be fed with the description (the

program) of any other machine to imitate this machine in its effects. And because ever since Turing it is possible to abstract from the differences in hardware between two devices, the so-called Church-Turing conjecture amounts, in its strictest—that is, physical—form, to declaring nature itself a Universal Turing Machine.

As such, this affirmation has doubled the implosion of hardware with the implosion of software. Ever since it has been possible to build computers—since 1943 with vacuum tubes, and since 1949 with transistors—the problem has existed of somehow describing and reading these universal writing-reading machines, which are in fact illegible. As is well known, the solution is called "software," that is, the development of higher programming languages. The ancient monopoly—whereby everyday languages functioned as their own metalanguages, therefore admitting no Other for the Other—has collapsed and given way to a new hierarchy of programming languages. This postmodern Tower of Babel[6] now extends, in the meanwhile, from simple command codes, whose linguistic extension is still a configuration of hardware, over assemblers, which are the extension of these same command codes, up to so-called standard languages [*Hochsprachen*], whose extensions—by way of innumerable detours through interpreters, compilers, and linkers—are also called "assemblers." Writing today, as it occurs in software development, is an infinite series of the self-similarities discovered by fractal geometry; except that, in contrast to the mathematical model, it remains mathematically impossible, in physical/physiological terms, to have access to [*erreichen*] all these layers [*Schichten*]. Modern media technology—ever since the invention of film and gramophones—is fundamentally arranged to undermine sensory perception. We can simply no longer know what our writing is doing, and least of all when we are programming.

The situation may be illustrated in everyday terms—for example, with the word-processing program from which the words here derive. May the *genius loci* of Palo Alto, which produced both the first and the most elegant operating systems, forgive a subject of the Microsoft Corporation for limiting the examples to the dumbest of all operating systems.

In order to process texts—that is, to become oneself a paper machine on an IBM AT under Microsoft DOS—the purchase of a commercial software package is the first item of business. Secondly, some of the files in this package must have the extension .EXE or .COM; otherwise, word processing cannot start under DOS. Executable files, namely, entertain a singular strange relationship with their proper names. On the one hand, they bear magniloquent

titles such as "WordPerfect"; on the other hand, they are more or less cryptic acronyms (because vowels are missing) like "WP." For all that, the full name serves only the advertising strategies of software manufacturers; the latter still employ everyday languages as a matter of course because the Disk Operating System, aka DOS, cannot read file names with more than eight letters. That is why unpronounceable abbreviations freed of vowels—acronyms that revoke the elementary innovation of ancient Greece—are both necessary for postmodern writing and perfectly up to the task. Indeed, for the first time since the alphabet was invented, these abbreviations seem to have endowed it with magical powers. "WP," that is, does exactly what it says. Unlike both the signifier "WordPerfect" and empty old-European names like "Spirit" or, indeed, "Word," executable computer files comprehend all the routines and data that are necessary for them to achieve realization. The act of writing—typing "W," "P," and "Enter"—does not make the Word perfect, but it does make WordPerfect run. Software affords many small triumphs of this sort.

The more or less inflationary literature that accompanies software—so as not to fall too short of the command line—doubles these magic powers. Typically, software manuals, because they must bridge the abyss between everyday languages, electronics, and literature, present the program package as a linguistic agent with the power to command, absolutely, system resources, address spaces, and hardware parameters of the computer in question: WP, activated by command-line argument *X*, switches the screen from mode *A* to mode *B*, starts in setting *C*, finally returns to *D*, and so on.[7]

However, all the actions that, according to what is written on paper, Agent WP performs are entirely virtual, because each one of them has to run "under" DOS (as it is so aptly put). But in factual terms, only the operating system—or more precisely, its shell—is at work: COMMAND.COM searches the keyboard buffer for an 8-byte file name, translates the relative addresses of the file it (perhaps) finds into absolute ones, loads this modified version from external bulk memory into silicon RAM, and finally assigns the execution of the program (which occurs for a limited time) to the first lines of code belonging to a slave named "WordPerfect."

At the same time, the same command-line argument can also be turned against DOS, because, in the final analysis, the operating system works as a simple expansion of a basic input/output system called BIOS. No single application, nor even the underlying microprocessor system, could ever start if a few elementary functions—which have been burned into silicon for secu-

rity reasons and therefore form part of the indelible hardware—did not, so to speak, possess Baron von Münchhausen's ability to pull themselves up out of the marsh by their own hair [*Zopf*].[8] Each material transformation of entropy into information, from a million dozing transistor cells into electric voltage differences, necessarily presumes a material event called "Reset."

In principle, this descent from software to hardware—from higher to lower levels of observation—could run over as many orders of magnitude as one wishes. Even elementary code operations, notwithstanding their metaphorical promises (e.g., "call" or "return"), amount to strictly local manipulations of signs and therefore (more's the pity, Lacan) to signifiers of varying electric potentials. All formalization—as defined by David Hilbert—effectively abolishes theory, simply because "the theory" at issue "is no longer a system of meaningful propositions, but one of sentences as sequences of words, which are in turn sequences of letters. We say by reference to the form alone which combinations of words are sentences, which sentences are axioms, and which sentences follow as immediate consequences of others."[9]

When meanings shrink down to sentences, sentences to words, and words to letters, then no software exists either. Or rather, it would not exist if computer systems did not need—at least until now—to coexist with an environment of everyday languages. This environment, however, has consisted, ever since a famous, twofold Greek invention,[10] of written characters and coins; that is, of "letters" and "litter." In the meanwhile, compelling economic reasons have fundamentally done away with the modesty of an Alan Turing—who, during the Stone Age of the Technical Era, preferred reading machine output in binary numbers to decimal computations.[11] The so-called philosophy of the so-called computing community places all its stock in hiding hardware behind software, and electronic signifiers behind human/machine interfaces. In a duly philanthropic spirit, handbooks for high-level programming languages warn of the mental breakdown that would result from writing trigonometric functions in assembler code.[12] In all compassion, BIOS (Basic Input/Output System) guidebooks (and their professional authors) take it upon themselves to "hide the particulars controlling the underlying hardware from your program."[13] Taken to a logical conclusion—and without much difference from the gradations in medieval hierarchies of angels—operating-system functions such as COMMAND.COM would hide the BIOS altogether; application programs like WordPerfect would conceal the operating system entirely; and so on.

Now fundamental changes in computer design (i.e., in the way the Penta-

gon conceives science) have led this whole system of secrecy to be perfected. First—and on an intentionally superficial level—graphic interfaces were developed for use that, because they conceal operations [*Schreibakte*] necessary for programming, deprive users of the machine as a whole. The IBM-authorized compendium of computer graphics does not even pretend that its user interfaces make system programming faster or more efficient than simple command lines would be.[14] Secondly—in immediate conjunction with Ada,[15] the Pentagon's programming language, but also on the microscopic level of the hardware itself—a new operating mode called "Protected Mode" was developed. According to Intel's *Microprocessor Programming Manual*, it has the sole purpose of keeping "untrusted programs" and "untrusted users" from all access to system resources such as input/output channels and the core of the operating system.[16] In a technical sense, however, all users are untrustworthy; in Protected Mode (as it prevails under UNIX, for example), they are not permitted to control their machines at all anymore.

The uninterrupted victory march of software represents a strange reversal of Turing's proof that there can be no problems calculable in a mathematical sense that a simple machine could not solve. At the precise location of this machine, the physical Church-Turing conjecture, by equating physical hardware with the algorithms for calculating it, created a blank that software could successfully occupy—and from whose obscurity it benefits.

After all, high-level programming languages—the higher their Tower of Babel grows and the more everyday it becomes—operate just like the so-called one-way functions of the newest mathematical cryptography.[17] In their standard form, such functions may be calculated with a justifiable time investment; for example, when the length of operations [*Maschinenzeit*] only increases for polynomial expressions of functional complexity. On the other hand, however, the time cost for the inverse operation—that is, the matter of calculating input parameters on the basis of a function's results—would increase in exponential, and therefore untenable, relation to the function's complexity. In other words, one-way functions protect algorithms from their own results.

This cryptographic feature is, as it were, made to order for software. It offers a comfortable way to avoid what Turing's proof shows: that the concept of intellectual property has become impossible—and especially where algorithms are concerned. Yet the very fact that software has no existence independent of machines has only increased insistence on the commercial (or American) quality of the medium. All licenses, dongles, or patents that have been registered for

WP—or WordPerfect—prove the functionality of one-way functions. American courts, in contempt of all traditions of mathematical honor, have even confirmed copyright claims to algorithms.

And so, it is not surprising that recently and on the highest level—at IBM, that is—the hunt has opened for mathematical formulas that might determine the difference in complexity (the Kolmogorov equation) between an algorithm and its output. In the good old days of Shannon's theory of information, maximum information coincided with maximum noise to some degree.[18] In contrast, the new IBM measure of logical depth is defined as follows:

> The value of a message . . . appears to reside not in its information (its absolutely unpredictable parts) nor in its obvious redundancy (verbatim repetitions, unequal digit frequencies), but rather in what might be called its buried redundancy—parts predictable only with difficulty, things the receiver could in principle have figured out without being told, but only at considerable cost in money, time, or computation. In other words, the value of a message is the amount of mathematical or other work plausibly done by its originator, which its receiver is saved from having to repeat.[19]

IBM's measure of logical depth, in its mathematical rigor, might also replace the antiquated, necessarily imprecise everyday terms of "originality," "authorship," and "copyright"—and thereby make them legally enforceable as well. Unfortunately, however, the algorithm for calculating the originality of algorithms in general is incalculable even by Turing's methods [*Nur leider ist gerade der Algorithmus zur Originalitätsberechnung von Algorithmen überhaupt selber turing-unberechenbar*].[20]

In this tragic situation, penal law—at least in Germany—has given up the concept of intellectual property for software (which is just as immaterial as "the Law" itself) and, instead, defined software as a "thing" [*Sache*]. The ruling of the Federal Constitutional Court [*Bundesgerichtshof*], according to which no computer program would ever run without corresponding electrical charges in silicon circuitry,[21] proves yet again that the virtual undecidability between software and hardware is hardly based—as systems theorists would so gladly believe—on a change of the observer's perspective.[22] On the contrary, there are good reasons to find for the indispensability, and therefore the precedence, of hardware.

A machine with unlimited resources of time and space, with an endless paper supply and unlimited calculating speed, has only ever existed once: in Turing's paper "On Computable Numbers with an Application to the Entscheid-

ungsproblem." In contrast, all physically constructible machines are limited by strict parameters within their very code. The inability of Microsoft DOS to recognize file names longer than eight characters (e.g., "WordPerfect") does not just illustrate, in its own trivial and obsolete way, a problem that has entailed more and more incompatibility between the different generations of 8-bit, 16-bit, and 32-bit microprocessors. It also points to the impossibility, as a matter of definition, of digitalization—that is, of calculating the body of real numbers, which used to be called "nature."[23]

That means, however, in the words of the Los Alamos National Laboratory, that

> We use digital computers whose architecture is given to us in the form of a physical piece of machinery, with all its artificial constraints. We must reduce a continuous algorithmic description to one codable on a device whose fundamental operations are countable, and we do this by various forms of chopping up into pieces, usually called discretization. Using finite differences, elements, or some similar scheme, an algorithm with an operation count belonging to N is constructed and is translated into some high level language. The compiler then further reduces this model to a binary form determined largely by machinic constraints.
>
> The outcome is a discrete and synthetic microworld image of the original problem, whose structure is arbitrarily fixed by a differencing scheme and computational architecture chosen at random. The only remnant of the continuum is the use of radix arithmetic, which has the property of weighing bits unequally, and for nonlinear systems is the source of spurious singularities.
>
> This is what we actually do when we compute up a model of the physical world with physical devices. This is not the idealized and serene process that we imagine when usually arguing about the fundamental structures of computation, and very far from Turing machines.[24]

And so, it is no longer a matter of further pursuing the Church-Turing hypothesis and "inject[ing] an algorithmic character into the behavior of the physical world for which there is not evidence."[25] If the world does not arise from God playing dice, the algorithmic behavior of rain clouds or waves in the sea does not exclude, but rather includes, the fact that their molecules operate as computers of their own activity. On the contrary, it would all be a matter of calculating the "price of programmability" itself. This decisive capacity of computers clearly has nothing to do with software; it depends only on the degree to which a given item of hardware can house something like a writing system.

In 1937, when Claude Shannon—"in the most momentous [*folgenreichste*] master's thesis that was ever written"[26]—provided proof that simple telegraph

relays could implement Boolean algebra as a whole, such a recording system [*Aufschreibesystem*] was established. And when the integrated circuit, derived from William Shockley's transistor in the early seventies, combined, on one and the same chip, the controllable resistance of silicon with its own oxide as a near perfect insulator, the programmability of matter could, as Turing had prophesied, "take control."[27] And so, software—if it even existed at all—would simply be a billion-dollar business revolving around one of the cheapest elements on earth. Connected on a chip, silicon and silicon oxide yield almost perfect hardware. On the one hand, millions of switching elements work under the same physical conditions—which is decisive, above all, for the critical parameter of chip temperature and prevents exponentially increasing deviations of transistor resistance. On the other, these millions of switching elements remain electrically isolated from each other. Only this paradoxical relation between two physical parameters—thermal continuity and electrical discretization—makes it possible for integrated digital circuitry not just to be finite machines [*Automaten*], like so many other devices on earth, but to approximate the Universal Discrete Machine that has long since swallowed up the name "Turing."

This structural difference can be illustrated quite easily. For example,

> a combination lock is a finite automaton, but it is not ordinarily decomposable into a base set of elementary type components that can be reconfigured to simulate an arbitrary physical system. As a consequence it is not structurally programmable, and in this case it is effectively programmable only in the limited sense that its state can be set for achieving a limited class of behaviors. In contrast, a digital computer used to simulate a combination lock is structurally programmable since the behavior is achieved by synthesizing it from a canonical set of primitive switching components.[28]

Switching components, however—be they telegraph relays, electron tubes, or finally, silicon transistors—pay a price for their decomposability [*Zerlegbarkeit*] or discretization. Apart from the trivial (i.e., discrete) case of word processing—which all but fades away in view of all the other scientific, military, and industrial areas where computers are employed—digital calculators, as the sole "all-or-none organs" in the strict sense of the word,[29] continue to face a continuous environment of clouds, waves, and wars. This avalanche of enormous, and real, numbers, as Ian Hacking would say, can only be mastered by adding more and more switching elements—until the 2,000 transistors of the Intel 4004 have turned into the 1.2 million of the current Intel flagship, the 80486. However, it can be mathematically demonstrated that the growth rate

of possible connections between these elements—that is, the computing power as such—has a square-root function as its upper limit. The system, in other words, "cannot keep up with polynomial growth rates in problem size,"[30] to say nothing of exponential rates. The same isolation between digital or discrete elements that secures its ability to function—at least under conditions that are not tropical or arctic—also limits the dimension of possible connections to the local environment of a given chip. Under conditions of global interaction [*Wechselwirkungen*], however, which digital chips experience only in thermal terms, connectivity "according to current force laws"[31] and following combinatory logic could rise to an upper limit set by the squared value of all elements involved.

Precisely this optimal connectivity—on the other, physical hand—distinguishes nonprogrammable systems. On the basis of their global interaction, such systems, whether they are waves or beings, can display polynomial growth rates in complexity; therefore, however, they could only be calculated by machines that would not have to pay the price of programmability themselves. Clearly, this hypothetical—but sorely needed—type of machine would be pure hardware: a physical apparatus working in an environment of any number of physical devices and subject only to the same limitations of resources to which they are also subject. Software, in the conventional sense of an abstraction that may be realized, would no longer exist. The procedures for such a machine, even though they would remain open for algorithmic scripting, would essentially have to operate on a material substrate whose connectivity would permit continuous reconfiguration of its cells. And although "the substrate can also be described in algorithmic terms, by means of simulation," its "characterization is of such immense importance for . . . effectiveness and so closely connected with the choice of hardware"[32] that programming it would have little in common with that of approximated Turing machines.

Such badly needed—and none-too-distant—machines, which are already being discussed in current computer science and have already been approximated by the chip industry,[33] might tempt the eyes of some observers to discern the familiar visage of "man" [*das vertraute Antlitz des Menschen*], whether evolutionarily disguised or not, in them. That may be. At the same time, however, our equally familiar silicon hardware is already following many of the demands placed on highly connected, unprogrammable systems. Between its million transistor cells, a million times a million interactions have always already been occurring: electron diffusion and quantum-mechanical tunnel ef-

fects occur over the entire chip—except that the manufacturing technology of today treats such interactions as system limitations, physical side effects, sources of interference, and so on. All this noise, which cannot be avoided, is at least to be minimized: that is the price that the computer industry must pay for structurally programmable machines. The inverse strategy—maximizing noise—would not just lead back from IBM to Shannon; it would also offer the sole path to that body of real numbers, which once was called "Chaos."

"Can't you understand what I'm tryin' to say," goes the original version of "Eve of Destruction."

the 80286 recognizes it), cf. Norbert Juffa and Peter Siering, "Wege über die Mauer. Loadall—Extended Memory im Real Mode des 80286," *c't* 11 (1990), 262–66.

24. Löwe, *VHSIC*, 70.

Chapter 16

1. Thanks to Wolfgang Hagen (Radio Bremen), who performed the textual comparison of the two versions of "Eve of Destruction" for listeners in a live broadcast.

2. Cf. Thomas Hafki, *Franklin—Frankenstein. Zum Verhältnis von Elektrizität und Literatur*, Master's thesis [*Magisterarbeit*], Bochum, 1993.

3. In 1978, when the Intel 8086 processor was being designed, the blueprints are said to have filled sixty-four square meters of graph paper. Cf. Klaus Schrödl, "Quantensprung," *DOS* 12 (1990): 102f.

4. Cf. Alan M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society* 42 (1937): 230–65.

5. The extent of such repression is demonstrated by the authors (or typesetters) of Intel's reference manuals: for example, the floating point instruction *f2xm1*—that is, an input quantity squared, minus one—does not become, once translated into everyday English, "Compute 2^x-1," but rather "Computer 2^x-1." Cf. Intel, *387 DX User's Manual. Programmer's Reference* (Santa Clara, 1989), 4–9; as well as Intel, *i486 Microprocessor. Programmer's Reference Manual* (Santa Clara, 1990), 26–72.

6. Cf. Wolfgang Hagen, "Die verlorene Schrift. Skizzen zu einer Theorie der Computer," *Arsenale der Seele. Literatur- und Medienanalyse seit 1870*, ed. Friedrich A. Kittler and Georg Christoph Tholen (Munich: Fink, 1989), 221:

> Therefore, the linguistic structure of von-Neumann machine logic already establishes, in principle, the divergence of software and software manuals, and so, from 1945 on, a Babylonian tower of computer performances has piled ever higher; using them has nothing more to do with the meaningful arrangement of a machine language. A tower of software with undocumented errors, hopelessly confused dialects, and a mass of linguistic acts that no one at all can follow.

In an image that for being less precise is all the more desperate, a UNIX expert has written: "Almost all operating systems are marked, after a certain age, by a high 'degree of pollution.' They grow wild in all directions and give the impression of being ruins that can only be held together with great effort" (Horst Drees, *UNIX. Ein umfassendes Kompendium für Anwender und Systemspezialisten* [Haar: Markt & Technik, 1988], 19). The UNIX expert is too polite to weave the proper name of a company such as Microsoft into the welter.

7. It is no accident that the only counterexample of which I am aware comes

from Richard Stallman's Free Software Foundation, which has declared war—a struggle that is just as heroic as it is doomed—on software copyright in general. This counterexample goes: "When we say that '*C-n* moves down vertically one line' we are glossing over a distinction that is irrelevant in ordinary use but is vital in understanding how to customize Emacs. It is the function *next-line* that is programmed to move down vertically. *C-n* has this effect because it is bound to that function. If you rebind *C-n* to the function *forward-word* then *C-n* will move forward by words instead" (Richard Stallman, *GNU Emacs Manual* [Cambridge, Mass., 1988], 19).

8. May this be accepted as a free translation for "booting."

9. Stephen C. Kleene, quoted in Robert Rosen, "Effective Processes and Natural Law," *The Universal Turing Machine. A Half-Century Survey*, ed. Rolf Herken (New York: Springer, 1995), 489.

10. Cf. Johannes Lohmann, "Die Geburt der Tragödie aus dem Geiste der Musik," *Archiv für Musikwissenschaft* (1980), 174.

11. Cf. Andrew Hodges, *Alan Turing: The Enigma* (New York: Simon & Schuster, 1983), 399.

12. Cf. *TOOL Praxis: Assembler-Programmierung auf dem PC, Ausgabe 1* (Würzburg, 1989), 9.

13. Nahajyoti Barkalati, *The Waite Group's Macroassembler Bible* (Carmel, Ind.: Howard H. Sams, 1989), 528.

14. Cf. James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes, *Computer Graphics. Principles and Practice* (Reading/Mass., 1990), 398.

15. On the connection between the Pentagon, Ada, and Intel's iAPX 432, the first microprocessor in Protected Mode—which, when it failed economically, gave rise to the industry standard from the 80286 up to the 80486—cf. Glenford J. Myers, "Overview of the Intel iAPX 432 Microprocessor," *Advances in Computer Architecture* (New York: John Wiley & Sons, 1982), 335–44 (with thanks to Ingo Ruhmann, Bonn). Whoever wishes to understand the failure should meditate the following: "The 432 can be characterized as a three-address-storage-to-storage architecture, there are no registers visible to programs" (342).

16. Cf. Chapter 15.

17. On the following, cf. Patrick Horster, *Kryptologie* (Mannheim, Vienna, Zurich: Bibliographisches Institut Wissenschaftsverlag, 1985), 23–27.

18. See Chapter 12.

19. Charles H. Bennett, "Logical Depth and Physical Complexity," *Universal Turing Machine*, 210.

20. Thanks to Oswald Wiener, Dawson City.

21. Cf. M. Michael König, "Sachlich sehen. Probleme bei der Überlassung von Software," *c't* 3 (1990): 73.

22. One might sooner—as it stands in a letter from Dirk Baecker (15 April 1991)—

> suppose that the distinction between hardware and software is a distinction that is supposed to escort [*betreuen*] the reentry of the distinction between programmability and non-programmability back into the realm of what can be programmed. It stands, so to speak, for the calculability of technology, and in this sense, for technology itself. This can be only because the "unity" of the program can only be achieved if equation and calculation are, respectively, distributed on two sides, so that only one of the sides is ever operationally available and the other one can be kept constant.

23. Consequently, I am at a loss to explain how Turing's famous paper could declare in the first sentence that "the 'computable numbers' may be described briefly as the real numbers whose expressions as a decimal are calculable by finite means" (Turing, "On Computable Numbers," 230), then define the set of calculable numbers as countable, and finally, call π, as "the limit of a computably convergent sequence," a computable number (256).

24. Brosl Hasslacher, "Beyond the Turing Machine," *Universal Turing Machine*, 391.

25. Ibid., 389.

26. Friedrich-Wilhelm Hagemeyer, *Die Entstehung von Informationskonzepten in der Nachrichtentechnik. Eine Fallstudie zur Theoriebildung in der Technik in Industrie- und Kriegsforschung*, Diss. phil. (typescript), Berlin, 1979, 432.

27. Alan Turing, "Intelligent Machinery: A Heretical Theory," *The Essential Turing: Writings in Computing, Logic, Philosophy, Artificial Intelligence, and Artificial Life*, ed. B. Jack Copeland (Oxford: Oxford University Press, 2004), 475.

28. Michael Conrad, "The Price of Programmability," *Universal Turing Machine*, 264–65.

29. Cf. John von Neumann, "General and Logical Theory of Automata," *Collected Works* (Oxford: Pergamon, 1963), V, 296ff.

30. Conrad, "Price of Programmability," 268.

31. Ibid., 265.

32. Ibid., 279.

33. Thus, the first integrated neural network—from, of all places, Intel's discrete chip empire (and as far as I can see, for the second time in the entire history of the firm after the relatively hybrid i2920 signal processor)—went back to straightforward analog operational amplifiers.