Programmed Visions

Software Studies

Matthew Fuller, Lev Manovich, and Noah Wardrip-Fruin, editors

Expressive Processing: Digital Fictions, Computer Games, and Software Studies
Noah Wardrip-Fruin, 2009

Code/Space: Software and Everyday Life
Rob Kitchin and Martin Dodge, 2011

Programmed Visions: Software and Memory
Wendy Hui Kyong Chun, 2011

Contents

Series Foreword vii
Preface: Programming the Bleeding Edge of Obsolescence xi

Introduction: Software, a Supersensible Sensible Thing 1

***You* 13**

I Invisibly Visible, Visibly Invisible 15

1 On Sourcery and Source Codes 19

***Computers that Roar* 55**

2 Daemonic Interfaces, Empowering Obfuscations 59

II Regenerating Archives 97

3 Order from Order, or Life According to Software 101

***The Undead of Information* 133**

4 Always Already There, or Software as Memory 137

Conclusion: In Medias Res 175

Epilogue: In Medias Race 179

***You, Again* 181**

Notes 183
Index 233

1 On Sourcery and Source Codes

The spirit speaks! I see how it must read,
And boldly write: "In the beginning was the Deed!"
—Johann Wolfgang Goethe[1]

Software emerged as a thing—as an iterable textual program—through a process of commercialization and commodification that has made code *logos*: code as source, code as true representation of action, indeed, code as conflated with, and substituting for, action.[2] Now, in the beginning, is the word, the instruction. Software as logos turns *program* into a noun—it turns process in time into process in (text) space. In other words, Manfred Broy's software "pioneers," by making software easier to visualize, not only sought to make the implicit explicit, they also created a system in which the intangible and implicit drives the explicit. They thus obfuscated the machine and the process of execution, making software the end all and be all of computation and putting in place a powerful logic of sourcery that makes source code—which tellingly was first called pseudocode—a fetish.[3]

This chapter investigates the implications of code as logos and the ways in which this simultaneous conflation and separation of instruction from execution, itself a software effect, is constantly constructed and undone, historically and theoretically. This separation is crucial to understanding the power and thrill of programming, in particular the nostalgic fantasy of an all-powerful programmer, a sovereign neoliberal subject who magically transforms words into things. It is also key to addressing the nagging doubts and frustrations experienced by programmers: the sense that we are slaves, rather than masters, clerks rather than managers—that, because "code is law," the code, rather than the programmer, rules. These anxieties have paradoxically led to the romanticization and recuperation of early female operators of the 1946 Electronic Numerical Integrator and Computer (ENIAC) as the first programmers, for they, unlike us, had intimate contact with and knowledge of the machine. They did not even need code: they engaged in what is now called "direct programming," wiring connections

and setting values. Back then, however, the "master programmer" was part of the machine (it controlled the sequence of calculation); computers, in contrast, were human. Rather than making programmers and users either masters or slaves, code as logos establishes a perpetual oscillation between the two positions: every move to empower also estranges.

This chapter, however, does not call for a return to direct programming or hardware algorithms, which, as I argue in chapter 4, also embody logos. It also does not endorse such a call because the desire for a "return" to a simpler map of power drives source code as logos. The point is not to break free from this sourcery, but rather to play with the ways in which logos also invokes "spellbinding powers of enchantment, mesmerizing fascination, and alchemical transformation."[4] The point is to make our computers more productively spectral by exploiting the unexpected possibilities of source code as fetish. As a fetish, source code produces surprisingly "deviant" pleasures that do not end where they should. Framed as a re-source, it can help us think through the machinic and human rituals that help us imagine our technologies and their executions. The point is also to understand how the surprising emergence of code as logos shifts early and still-lingering debates in new media studies over electronic writing's relation to poststructuralism, debates that the move to software studies has to some extent sought to foreclose.[5] Rather than seeing technology as simply fulfilling or killing theory, this chapter outlines how the alleged "convergence" between theory and technology challenges what we thought we knew about logos. Relatedly, engaging source code as fetish does not mean condemning software as immaterial; rather, it means realizing the extent to which software, as an "immaterial" relation become thing, is linked to changes in the nature of subject-object relations more generally. Software as thing can help us link together minute machinations and larger flows of power, but only if we respect its ability to surprise and to move.

Source Code as Logos

To exaggerate slightly, software has recently been posited as the essence of new media and knowing software a form of enlightenment. Lev Manovich, in his groundbreaking *The Language of New Media*, for instance, asserts: "New media may look like media, but this is only the surface. . . . To understand the logic of new media, we need to turn to computer science. It is there that we may expect to find the new terms, categories, and operations that characterize media that become programmable. *From media studies, we move to something that can be called 'software studies'—from media theory to software theory.*"[6] This turn to software—to the logic of what lies beneath—has offered a solid ground to new media studies, allowing it, as Manovich argues, to engage presently existing technologies and to banish so-called "vapor theory"—theory that fails to distinguish between demo and product, fiction and reality—to the margins.[7]

This call to banish vapor theory, made by Geert Lovink and Alexander Galloway among others, has been crucial to the rigorous study of new media, but this rush away from what is vapory—undefined, set in motion—is also troubling because vaporiness is not accidental but rather essential to new media and, more broadly, to software. Indeed, one of this book's central arguments is that a rigorous engagement with software makes new media studies more, rather than less, vapory. Software, after all, is ephemeral, information ghostly, and new media projects that have never, or barely, materialized are among the most valorized and cited.[8] (Also, if you take the technical definition of information seriously, information increases with vapor, with entropy). This turn to computer science also threatens to reify knowing software as truth, an experience that is arguably impossible: we all know some software, some programming languages, but does anyone really "know" software? What could this knowing even mean? Regardless, from myths of all-powerful hackers who "speak the language of computers as one does a mother tongue"[9] or who produce abstractions that release the virtual[10] to perhaps more mundane claims made about the radicality of open source, knowing (or using the right) software has been made analogous to man's release from his self-incurred tutelage.[11] As advocates of free and open source software make clear, this critique aims at political, as well as epistemological, emancipation. As a form of enlightenment, it is a stance of how not to be governed like that, an assertion of an essential freedom that can only be curtailed at great cost.[12]

Knowing software, however, does not simply enable us to fight domination or rescue software from "evil-doers" such as Microsoft. Software, free or not, is embedded and participates in structures of knowledge-power. For instance, using free software does not mean escaping from power, but rather engaging it differently, for free and open source software profoundly privatizes the public domain: GNU copyleft—which allows one to use, modify, and redistribute source code and derived programs, but only if the original distribution terms are maintained—seeks to fight copyright by spreading licences everywhere.[13] More subtly, the free software movement, by linking freedom and freely accessible source code, amplifies the power of source code both politically and technically. It erases the vicissitudes of execution and the institutional and technical structures needed to ensure the coincidence of source code and its execution. This amplification of the power of source code also dominates critical analyses of code, and the valorization of software as a "driving layer" conceptually constructs software as neatly layered.

Programmers, computer scientists, and critical theorists have reduced software to a recipe, a set of instructions, substituting space/text for time/process. The current common-sense definition of *software* as a "set of instructions that direct a computer to do a specific task" and the OED definition of software as "the programs and procedures required to enable a computer to perform a specific task, as opposed to the physical components of the system" both posit software as cause, as what drives

computation. Similarly, Alexander Galloway argues, "code draws a line between what is material and what is active, in essence saying that writing (hardware) cannot *do* anything, but must be transformed into code (software) to be effective. . . . Code is a language, but a very special kind of language. *Code is the only language that is executable* . . . code is the first language that actually does what it says."[14] This view of software as "actually doing what it *says*" (emphasis added) both separates instruction from, and makes software substitute for, execution. It assumes no difference between source code and execution, between instruction and result. That is, Galloway takes the principles of executable layers (application on top of operating system, etc.) and grafts it onto the system of compilation or translation, in which higher-level languages are transformed into executable codes that are then executed line by line. By doing what it "says," code is surprisingly logos. Like the King's speech in Plato's *Phaedrus*, it does not pronounce knowledge or demonstrate it—it transparently pronounces itself.[15] The hidden signified—meaning—shines through and transforms itself into action. Like Faust's translation of logos as "deed," code is action, so that "in the beginning was the Word, and the Word was with God, and the Word was God."[16]

Not surprisingly, many scholars critically studying code have theorized code as performative. Drawing in part from Galloway, N. Katherine Hayles in *My Mother Was a Computer: Digital Subjects and Literary Texts* distinguishes between the linguistic performative and the machinic performative, arguing:

> Code that runs on a machine is performative in a much stronger sense than that attributed to language. When language is said to be performative, the kinds of actions it "performs" happen in the minds of humans, as when someone says "I declare this legislative session open" or "I pronounce you husband and wife." Granted, these changes in minds can and do reach in behavioral effects, but the performative force of language is nonetheless tied to the external changes through complex chains of mediation. By contrast, code running in a digital computer causes changes in machine behavior and, through networked ports and other interfaces, may initiate other changes, all implemented through transmission and execution of code.[17]

The independence of machine action—this autonomy, or automatic executability of code—is, according to Galloway, its material essence: "The material substrate of code, which must always exist as an amalgam of electrical signals and logical operations in silicon, however large or small, demonstrates that code exists first and foremost as commands issued to a machine. Code essentially has no other reason for being than instructing some machine in how to act. One cannot say the same for the natural languages."[18] Galloway thus concludes in "Language Wants to Be Overlooked: On Software and Ideology," "to see code as subjectively performative or enunciative is to anthropomorphize it, to project it onto the rubric of psychology, rather than to understand it through its own logic of 'calculation' or 'command.'"[19]

To what extent, however, can source code be understood outside of anthropomorphization? Does understanding voltages stored in memory as commands/code not

already anthropomorphize the machine? The title of Galloway's article, "Language *Wants* to Be Overlooked" (emphasis mine), inadvertently reveals the inevitability of this anthropomorphization. How can code/language want—or most revealingly *say*—anything? How exactly does code "cause" changes in machine behavior? What mediations are necessary for this insightful yet limiting notion of code as inherently executable, as conflating meaning and action?

Crafty Sources

To make the argument that code is automatically executable, the process of execution itself not only must be erased, but source code must also be conflated with its executable version. This is possible, Galloway argues, because the two "layers" of code can be reduced to each other: "uncompiled source code is *logically* equivalent to that same code compiled into assembly language and/or linked into machine code. For example, it is absurd to claim that a certain value expressed as a hexadecimal (base 16) number is more or less fundamental than that same value expressed as binary (base 2) number. They are simply two expressions of the same value."[20] He later elaborates on this point by drawing an analogy between quadratic equations and software layers:

> One should never understand this "higher" symbolic machine as anything empirically different from the "lower" symbolic interactions of voltages through logic gates. They are complex aggregates yes, but it is foolish to think that writing an "if/then" control structure in eight lines of assembly code is any more or less machinic than doing it in one line of C, just as the same quadratic equation may swell with any number of multipliers and still remain balanced. The relationship between the two is *technical*.[21]

According to Galloway's quadratic equation analogy, the difference between a compact line of higher-level programming code and eight lines written in assembler equals the difference between two equations, in which one contains coefficients that are multiples of the other. The solution to both equations is the same: one equation is the same as the other.

This reduction, however, does not capture the difference between the various instantiations of code, let alone the empirical difference between the higher symbolic machine and the lower interactions of voltages (the question here is: where does one make the empirical observation?). To state the obvious, one cannot run source code: it must be compiled or interpreted. This compilation or interpretation—this making executable of code—is not a trivial action; the compilation of code is not the same as translating a decimal number into a binary one. Rather, it involves instruction explosion and the translation of symbolic into real addresses. Consider, for example, the instructions needed for adding two numbers in PowerPC assembly language, which is one level higher than machine language:

```
li      r3,1          *load the number 1 into register 3
li      r4,2          *load the number 2 into register 4
add     r5,r4,r3      *add r3 to r4 and store the result in r5
stw     r5,sum(rtoc)  *store the contents of r5 (i.e., 3) into the memory location
                      *called "sum" (where sum is defined elsewhere)
blr                   *end of this snippet of code[22]
```

This explosion is not equivalent to multiplying both sides of a quadratic equation by the same coefficient or to the difference between E and 15. It is, instead, a breakdown of the steps needed to perform a simple arithmetic calculation; it focuses on the movement of data within the machine. The relationship between executable and higher-level code is not that of mathematical identity but rather logical equivalence, which can involve a leap of faith. This is clearest in the use of numerical methods to turn integration—a function performed fluidly in analog computers—into a series of simpler, repetitive arithmetical steps.

This translation from source code to executable is arguably as involved as the execution of any command, and it depends on the action (human or otherwise) of compiling/interpreting and executing. Also, some programs may be executable, but not all compiled code within that program is executed; rather, lines are read in as necessary. Software is "layered" in other words, not only because source is different from object, but also because object code is embedded within an operating system.

So, to spin Galloway's argument differently, a technical relation is far more complex than a numerical one. Rhetoric was considered a *technê* in antiquity. Drawing on this Paul Ricoeur explains, "*technê* is something more refined than a routine or an empirical practice and in spite of its focus on production, it contains a speculative element."[23] A technical relation engages art or craft. A technical person is one "skilled in or practically conversant with some particular art or subject."[24] Code does not always or automatically do what it says, but it does so in a crafty, speculative manner in which meaning and action are both created. It carries with it the possibility of deviousness: our belief that compilers simply expand higher-level commands—rather than alter or insert other behaviors—is simply that, a belief, one of the many that sustain computing as such. This belief glosses over the fact that *source code only becomes a source after the fact*. Execution, and a whole series of executions, belatedly makes some piece of code a source, which is again why source code, among other things, was initially called pseudocode.

Source code is more accurately a *re-source*, rather than a source. Source code becomes the source of an action only after it—or more precisely its executable substitute—expands to include software libraries, after its executable version merges with code burned into silicon chips; and after all these signals are carefully monitored, timed,

and rectified. Source code becomes a source only through its destruction, through its simultaneous nonpresence and presence.[25] (Thus, to return to the historical difficulties of analyzing software outlined by Mahoney, every software run is to some extent a reconstruction.) Source code as *technê*, as a generalized writing, is spectral. It is neither dead repetition nor living speech; nor is it a machine that erases the difference between the two. It, rather, puts in place a "relation between life and death, between present and representation, between two apparatuses."[26] As I elaborate throughout this book, information—through its capture in memory—is undead.

Source Code, after the Fact

Early on, the difficulties of code as source were obvious. Herman H. Goldstine and John von Neumann emphasized the dynamic nature of code in their "Planning and Coding of Problems for an Electronic Computing Instrument." In it, they argued that coding, despite the name, is not simply the static translation of "a meaningful text (the instructions that govern solving the problem under consideration) from one language (the language of mathematics, in which the planner will have conceived the problem, or rather the numerical procedure by which he has decided to solve the problem) into another language (that of our code)."[27] Because code does not unfold linearly, because its value depends on intermediate results, and because code can be modified as it is run (self-modifying code), "it will not be possible in general to foresee in advance and completely the actual course of C [the sequence of codes]." Therefore, "coding is . . . the technique of providing a dynamic background to control the automatic evolution of a meaning."[28] Code as "dead repetition," in other words, has always been regenerative and interactive; every iteration alters its meaning. Even given the limits to iterability that Hayles has presciently outlined in *My Mother Was a Computer*—limits due to software as axiomatic—coding still means producing a mark, a writing, open to alteration/iteration rather than an airtight anchor.[29]

Much disciplinary effort has been required to make source code readable as the source. Structured programming, which I examine in more detail later, sought to rein in "goto crazy" programmers and self-modifying code. A response to the much-discussed "software crisis" of the late 1960s, its goal was to move programming from a craft to a standardized industrial practice by creating disciplined programmers who dealt with abstractions rather than numerical processes.[30]

Making code the source also entails reducing hardware to memory and thus erasing the existence and possibility of hardware algorithms. Code is also not always the source because hardware does not need software to "do something." One can build algorithms using hardware. Figure 1.1, for instance, is the logical statement: if notB and notA, do CMD1 (state P); if notB and notA and notZ OR B and A (state Q) then command 2.

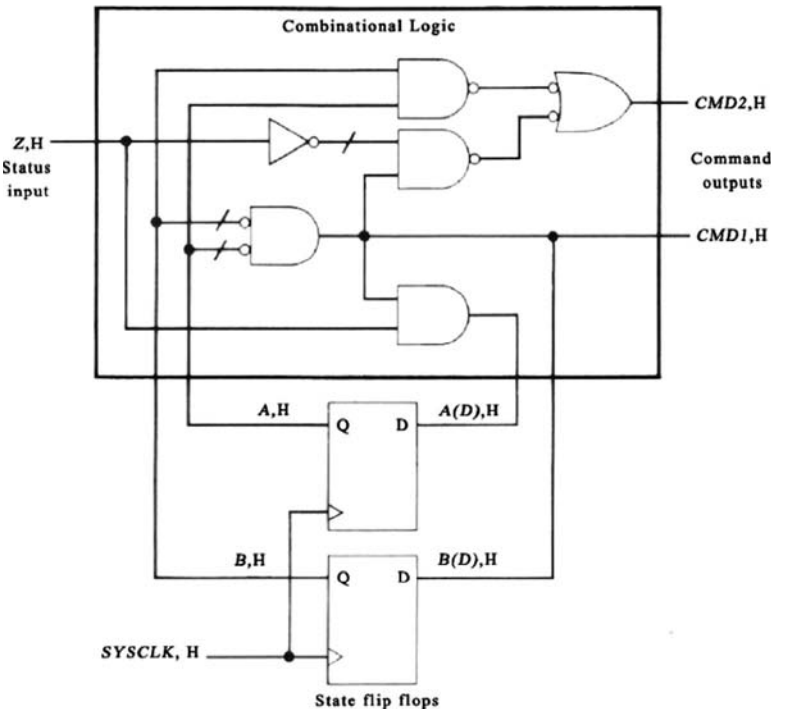

Figure 1.1
Logic diagram for a hardware algorithm

To be clear, I am not valorizing hardware over software, as though hardware naturally escapes this drive to make space signify time. Crucially, this schematic is itself an abstraction. Logic gates can only operate "logically"—as logos—if they are carefully timed. As Philip Agre has emphasized, the digital abstraction erases the fact that gates have "directionality in both space (listening to its inputs, driving its outputs) and in time (always moving toward a logically consistent relation between these inputs and outputs)."[31] When a value suddenly changes, there is a brief period in which a gate will give a false value. In addition, because signals propagate in time over space, they produce a magnetic field that can corrupt other nearby signals (known as *crosstalk*). This schematic erases all these various time- and distance-based effects by rendering space blank, empty, and banal. Thus hardware schematics, rather than escaping from the logic of sourcery, are also embedded within this structure. Indeed, as chapter 4 elaborates, John von Neumann, the generally acknowledged architect of the stored-memory digital computer, drew from Warren McCulloch and Walter Pitts's conflation of neuronal activity with its inscription in order to conceptualize modern computers. It is perhaps appropriate then that von Neumann, who died from a cancer stemming

from his work at Los Alamos, spent the last days of his life reciting from memory *Faust Part 1*.[32] At the source of stored program computing lies the Faustian erasure of word for action.

The notion of source code as source coincides with the introduction of alphanumeric languages. With them, human-written, nonexecutable code becomes source code and the compiled code, the object code. Source code thus is arguably symptomatic of human language's tendency to attribute a sovereign source to an action, a subject to a verb.[33] By converting action into language, source code emerges. Thus, Galloway's statement, "To see code as subjectively performative or enunciative is to anthropomorphize it, to project it onto the rubric of psychology, rather than to understand it through its own logic of 'calculation' or 'command,'" overlooks the fact that to use higher-level alphanumeric languages is already to anthropomorphize the machine. It is to embed computers in "logic" and to reduce all machinic actions to the commands that supposedly drive them. In other words, the fact that "code is law"—something legal scholar Lawrence Lessig emphasizes—is hardly profound.[34] After all, code is, according to the OED, "a systematic collection or digest of the laws of a country, or of those relating to a particular subject." What is surprising is the fact that software is code; that code is—has been made to be—executable, and this executability makes code not law, but rather every lawyer's dream of what law should be: automatically enabling and disabling certain actions, functioning at the level of everyday practice.[35]

Code is executable because it embodies the power of the executive, the power of enforcement that has traditionally—even within classic neoliberal logic—been the provenance of government.[36] Whereas neoliberal economist and theorist Milton Friedman must concede the necessity of government because of the difference between "the day-to-day activities of people [and] the general customary and legal framework within which these take place," code as self-enforcing law "privatizes" this function, further reducing the need for government to enforce the rules by which we play.[37] In other words, if as Foucault argues neoliberalism expands judicial interventions by reducing laws to "the rules for a game in which each remains master regarding himself and his part," then "code is law" reins in this expansion by moving enforcement from police and judicial functions to software functions.[38] "Code is law," in other words, automatically brings together disciplinary and sovereign power through the production of self-enforcing rules that, as von Neumann argues, "govern" a situation.

"Code is law" makes clear the desire for sovereign power driving both source code and performative utterances more generally. David Golumbia—looking more generally at widespread beliefs about computers—has insightfully claimed: "The computer encourages a Hobbesian conception of this political relation: one is either the person who makes and gives orders (the sovereign), or one follows orders."[39]

This conception, which crucially is also constantly undone by modern computation's twinning of empowerment with ignorance, depends, I argue, on this conflation of code with the performative. As Judith Butler has argued in *Excitable Speech*, Austinian understandings of performative utterances as simply doing what they say posit the speaker as "the judge or some other representative of the law."[40] It resuscitates fantasies of sovereign—that is *executive* (hence executable)—structures of power: it is "a wish to return to a simpler and more reassuring map of power, one in which the assumption of sovereignty remains secure."[41] This wish for a simpler map of power—indeed power as mappable—drives not only code as automatically executable, but also, as the next chapter contends, interfaces more generally. This wish is central to computers as machines that enable users/programmers to navigate neoliberal complexity.

Against this nostalgia, Butler, following Jacques Derrida, argues that iterability lies behind the effectiveness of performative utterances. For Butler, iterability is the process by which "*the subject who 'cites' the performative is temporarily produced as the belated and fictive origin of the performative itself*."[42] The programmer/user, in other words, is produced through the act of programming. Moreover, the effectiveness of performative utterances, Butler also emphasizes, is intimately tied to the community one joins and to the rituals involved—to the history of that utterance. Code as law—as a judicial process—is, in other words, far more complex than code as logos. Similarly, as Weizenbaum has argued, code understood as a judicial process undermines the control of the programmer:

> A large program is, to use an analogy of which Minsky is also fond, an intricately connected network of courts of law, that is, of subroutines, to which evidence is transmitted by other subroutines. These courts weigh (evaluate) the data given to them and then transmit their judgments to still other courts. The verdicts rendered by these courts may, indeed, often do, involve decisions about what court has "jurisdiction" over the intermediate results then being manipulated. The programmer thus cannot even know the path of decision-making within his own program, let alone what intermediate or final results it will produce. Program formulation is thus rather more like the creation of a bureaucracy than like the construction of a machine of the kind Lord Kelvin may have understood.[43]

Code as a judicial process is code as *thing*: the Latin term for thing, *res*, survives in legal discourse (and, as I explain later, literary theory). The term *res*, as Heidegger notes, designates a "gathering," any thing or relation that concerns man.[44] The relations that Weizenbaum discusses, these bureaucracies within the machine, as the rest of this chapter argues, mirror the bureaucracies and hierarchies that historically made computing possible. Importantly, this description of computers as following a set of rules that programmers must follow—Weizenbaum's insistence on the programmer's ignorance—does not undermine the resonances between neoliberalism and computation; if anything, it makes these resonances more clear. It also clarifies the desire

driving code as logos as a solution to neoliberal chaos. Foucault, emphasizing the rhetoric of the economy as a "game" in neoliberal writings, has argued, "both for the state and for individuals, the economy must be a game: a set of regulated activities . . . in which the rules are not decisions which someone takes for others. It is a set of rules which determine the way in which each must play a game whose outcome is not known by anyone."[45] Although small-s sovereigns proliferate through neoliberalism's empowered yet endangered subjects, it still fundamentally denies the position of the Sovereign who knows—a position that we nonetheless nostalgically desire . . . for ourselves.

Yes, Sir!

This conflation of instruction with result stems in part from software's and computing's gendered, military history: in the military there is supposed to be no difference between a command given and a command completed—especially to a computer that is a "girl." For computers, during World War II, were in fact young women with some background in mathematics. Not only were women available for work during that era, they also were considered to be better, more conscientious computers, presumably because they were better at repetitious, clerical tasks. They were also undifferentiated: they were all unnamed "computers," regardless of their mathematical training.[46] These computers produced ballistics tables for new weapons, tables designed to control servicemen's battlefield actions. Rather than aiming and shooting, servicemen were to set their guns to the proper values (not surprisingly, these tables and gun governors were often ignored or ditched by servicemen).[47]

The women who became the "ENIAC girls" (later the more politically correct "women of the ENIAC")—Kathleen/Kay McNulty (Mauchly Antonelli), Jean Jennings (Bartik), Frances Snyder (Holberton), Marlyn Wescoff (Meltzer), Frances Bilas (Spence), and Ruth Lichterman (Teitelbaum) (married names in parentheses)—were computers who volunteered to work on a secret project (when they learned they would be operating a machine, they had to be reassured that they had not been demoted). Programmers were former computers because they were best suited to prepare their successors: they thought and acted like computers. One could say that programming became programming and software became software when the command structure shifted from commanding a "girl" to commanding a machine. Kay Mauchly Antonelli described the "evolution" of computing as moving from female computers using Marchant machines to fill in fourteen-column sheets (which took forty hours to complete the job), to using differential analyzers (fifteen minutes to do the job), to using the ENIAC (seconds).[48]

Software languages draw from a series of imperatives that stem from World War II command and control structures. The automation of command and control, which

Paul Edwards has identified as a perversion of military traditions of "personal leadership, decentralized battlefield command, and experience-based authority,"[49] arguably started with World War II mechanical computation. Consider, for instance, the relationship between the volunteer members of the Women's Royal Naval Service (called Wrens), and their commanding officers at Bletchley Park. The Wrens also (perhaps ironically) called *slaves* by the mathematician and "founding" computer scientist Alan Turing (a term now embedded within computer systems), were clerks responsible for the mechanical operation of the cryptanalysis machines (the Bombe and then the Colossus), although at least one of the clerks, Joan Clarke (Turing's former fiancé), became an analyst. Revealingly, I. J. Good, a male analyst, describes the Colossus as enabling a man–machine synergy duplicated by modern machines only in the late 1970s: "the analyst would sit at the typewriter output and call out instructions to a Wren to make changes in the programs. Some of the other uses were eventually reduced to decision trees and were handed over to the machine operators (Wrens)."[50] This man–machine synergy, or interactive real-time (rather than batch) processing, treated Wrens and machines indistinguishably, while simultaneously relying on the Wrens' ability to respond to the mathematician's orders. This "interactive" system also seems evident in the ENIAC's operation: in figure 1.2, a male analyst issues commands to a female operator.

The story of the initial meeting between Grace Murray Hopper (one of the first and most important programmer-mathematicians) and Howard Aiken would also seem to buttress this narrative. Hopper, with a PhD in mathematics from Yale, and a former mathematics professor at Vassar, was assigned by the U.S. Navy to program the Mark 1, an electromechanical digital computer that made a sound like a roomful of knitting needles. According to Hopper, Aiken showed her "a large object with three stripes . . . waved his hand and said: 'That's a computing machine.' I said, 'Yes, Sir.' What else could I say? He said he would like to have me compute the coefficients of the arc tangent series, for Thursday. Again, what could I say? 'Yes, Sir.' I didn't know what on earth was happening, but that was my meeting with Howard Hathaway Aiken."[51] Computation depends on "Yes, Sir" in response to short declarative sentences and imperatives that are in essence commands. Contrary to Neal Stephenson, in the beginning—marking the possibility of a beginning—was the command rather than the command line.[52] The command line is a mere operating system (OS) simulation. Commands have enabled the slippage between programming and action that makes software such a compelling yet logically "trivial" communications system.[53] Commands lie at the core of the cybernetic conflation of human with machine.[54] I. J. Good's and Hopper's recollections also reveal the routinization at the core of programming: the analyst's position at Bletchley Park was soon replaced by decision trees acted on by the Wrens. Hopper, self-identified as a mathematician (not programmer), became an advocate of automatic programming. Thus routinization or automation lies at the

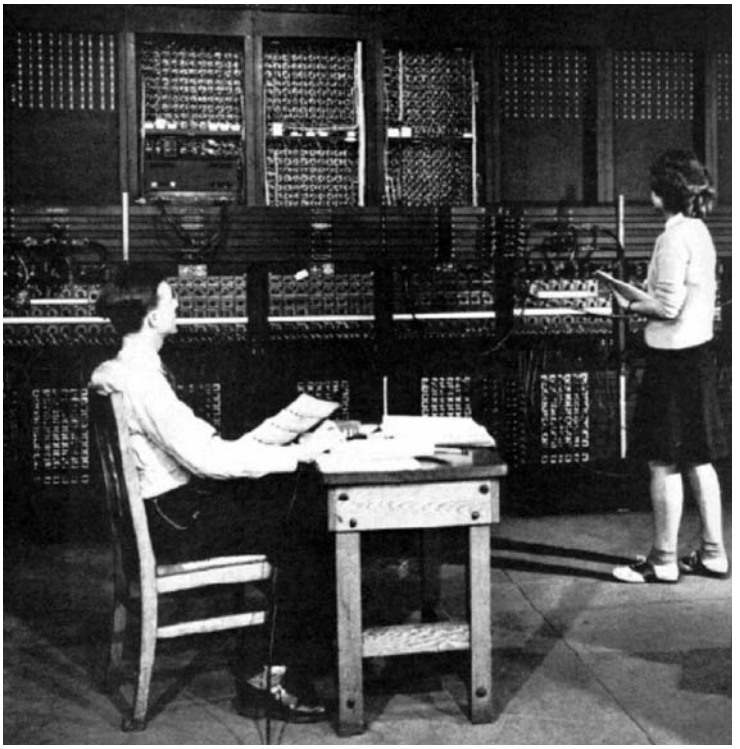

Figure 1.2
ENIAC programmers, late 1940s. U.S. military photo, Redstone Arsenal Archives, Huntsville, Alabama.

core of a profession that likes to believe it has successfully automated every profession but its own.[55]

This narrative of the interchangeability of women and software, however, is not entirely true: the perspective of the master, as Hegel famously noted, is skewed. (Tellingly, Mephistopheles offers to be Faust's servant.)[56] The master depends on the slave entirely, and it is the slave's actions that make possible another existence. Execution is never simple. Hopper's "Yes, Sir" actually did follow in the military command tradition. It was an acceptance of responsibility; she was not told how to calculate the trajectory. Also, the "women of the ENIAC," although an afterthought, played an important role in converting the ENIAC into a stored-program computer and in determining the trade-off between storing values and instructions: they did not simply operate the machine, they helped shape it and make it functional.[57] Users of the ENIAC usually were divided into pairs: one who knew the problem and one who knew the

machine "so the limitations of the machine could be fitted to the problem and the problem could be changed to fit the limitations."[58] Programming the ENIAC—that is, wiring the components together in order to solve a problem—was difficult, especially since there were no manuals or exact precedents.[59] To solve a problem, such as how to determine ballistics trajectories for new weapons, ENIAC "programmers" had first to break down the problem logically into a series of small yes/no decisions; "the amount of work that had to be done before you could ever get to a machine that was really doing any thinking," Bartik relates, was staggering and annoying.[60] The unreliability of the hardware and the fact that engineers and custodians would unexpectedly change the switches and program cables compounded the difficulty.[61]

These women, Holberton in particular, developed an intimate relation with the "master programmer," the ENIAC's control device. Although Antonelli first figured out how to repeat sections of the program, using the master programmer, Holberton, who described herself as a logician, specialized in controlling its operation.[62] As Bartik explains:

> We found it very easy to learn that you do this step, step one, then you do step two, step three, but I think the thing that was the hardest for us to learn was transfer of control which the ENIAC did have through the master programmer, so that you would be able to repeat pieces of program. So, the techniques for dividing your program into subroutines that could be repeated and things of this kind was the hardest for us to understand. I certainly know it was for me.[63]

Because logic diagrams did not then exist, Holberton developed a four-color pencil system to visualize the workings of the master programmer.[64] This drive to visualize also extended to the machine as a whole. To track the calculation, holes were drilled in the panels over the accumulators so that "when you were doing calculations these lights were flashing as the numbers built up and as you transferred numbers and things of this kind. So you had the feeling of excitement."[65] These lights not only were useful in tracking the machine, they also were invaluable for the demonstration. Even though the calculation for the demonstration was itself buggy, the flashing lights, the cards being read and written, gave the press a (to them) incomprehensible visual display of the enormity and speed of the calculation being done. In what would become a classic programming scenario, the problem was "debugged" the day after the demonstration. According to Holberton:

> I think the next morning, I woke up and in the middle of the night thinking what that error was. I came in, made a special trip on the early train that morning to look at a certain wire, and you know, it's the same kind of programming error that people make today. It's the, the decision on the terminal end of a do loop, speaking Fortran language, had the wrong value. Forgetting that zero was also one setting and the setting of the switch was one off. And I'll never forget that because there it was my first do loop error. But it went on that way and I remember telling Marlyn, I said, "If anybody asks why it's printing out that way, say it's supposed to be that way." [Laughter][66]

Programming enables a certain duplicity, as well as the possibility of endless actions that animate the machine. Holberton, described by Hopper as the best programmer she had known, would also go on to develop an influential SORT algorithm for the UNIVAC 1 (the Universal Automatic Computer 1, a commercial offshoot of the ENIAC).[67] Indeed, many of these women were hired by the Eckert–Mauchly company to become the first programmers of the UNIVAC, and were transferred to Aberdeen to train more ENIAC programmers.

Drawing from the historical importance of women and the theoretical resonances between the feminine and computing (parallels between programming and what Freud called the quintessentially feminine invention of weaving, between female sexuality as mimicry and Turing's vision of computers as universal machines/mimics) Sadie Plant has argued that computing is essentially feminine. Both software and feminine sexuality reveal the power that something that cannot be seen can have.[68] Women, Plant argues, "have not merely had a minor part to play in the emergence of digital machines. . . . Theirs is not a subsidiary role which needs to be rescued for posterity, a small supplement whose inclusion would set the existing records straight. . . . Hardware, software, wetware—before their beginnings and beyond their ends, women have been the simulators, assemblers, and programmers of the digital machines."[69] Because of this and women's early (forced) adaptation to "flexible" work conditions, Plant argues, women are best prepared to face our digital, networked future: "sperm count," she writes, "falls as the replicants stir and the meat learns how to learn for itself. Cybernetics is feminisation."[70] Responding to Plant's statement, Alexander Galloway has argued, "the universality of [computer] protocol can give feminism something that it never had at its disposal, the obliteration of the masculine from beginning to end."[71] Protocol, Galloway asserts, is inherently antipatriarchy. What, however, is the relationship between feminization and feminism, between so-called feminine modes of control and feminism? What happens if you take seriously Grace Murray Hopper's claims that the term *software* stemmed from her description of compilers as "layettes" for computers and the claim of J. Chuan Chu, one of the hardware engineers for the ENIAC, that software is the "daughter" of Frankenstein (hardware being the son)?[72]

To address these questions, we need to move beyond recognizing these women as programmers and the resonances between computers and the feminine. Such recognition alone establishes a powerful sourcery, in which programming is celebrated at the exact moment that programmers become incapable of "understanding"—of seeing through—the machine. The move to reclaim the ENIAC women as the first programmers in the mid- to late-1990s occurred when their work as operators—and the visual, intimate knowledge of machine operations this entailed—had become entirely incorporated into the machine and when women "coders" were almost definitively pushed out of the workplace. It is love at last (and first) sight, not just for these women but also for these interfaces, which really were transparent holes, in which inside and

outside coincided. Also, reclaiming these women as the first programmers and as feminist figures glosses over the hierarchies within programming—among operators, coders, and analysts—that defined the emergence of programming as a profession and as an academic discipline.[73] To put Hopper and the "ENIAC girls" together is to erase the difference between Hopper, a singular hero who always defined herself as a mathematician, and nameless disappearing computer operators. It is also to deny personal history: Hopper, a social conservative from a privileged background, stated many times that she was not a feminist, and Hopper's stances could be perceived as antifeminist (while the highest-ranking female officer in the Navy, she argued that women were incapable of serving in combat duty).[74] Not accidentally, Hopper's dream, her drive for automatic computing, was to put the programmer inside the computer and thus to rehumanize the mathematician: pseudocode was to free the mathematician and her brain from the shackles of programming.[75]

Bureaucracies within the Machine

TROPP: We talked about Von Neumann and I would like to talk about how you saw people like John Mauchly and the role that they played, and Goldstine and Burks and others that you came in contact with [including] Clippinger, and Frankel, and how, how they looked from your vantage point?
HOLBERTON: Well, we were lowly programmers, so I looked up to all these gentlemen.
TROPP: [Laughter][76]

The conflation of instruction with action, which makes computers understood as software and hardware machines such a compelling model of neoliberal governmentality and which resuscitates dreams of sovereign power, depends on incorporating historical programming hierarchies within the machine.

Programming, even at what has belatedly been recognized as its origin, was a hierarchical affair. Herman H. Goldstine and John von Neumann, in "Planning and Coding of Problems for an Electronic Computing Instrument," separated the task of planning (dealing with the dynamic nature of code through extensive flow charting) from that of coding (the microproduction of the actual instructions). Regarding dynamic or macroscopic aspects, they argued, "every mathematician, or every moderately mathematically trained person should be able to do this in a routine manner, if he has familiarized himself with the main examples that follow in this report, or if he has had some equivalent training in this method." Regarding the static or microscopic work, they asserted, "we feel certain that a moderate amount of experience with this stage of coding suffices to remove from it all difficulties, and to make it a perfectly routine operation."[77] The dropping of the pronoun *he* was not accidental: as Nathan Ensmenger and William Aspray note, the dynamic analysis was to be performed by "the 'planner,' who was typically the scientific user and overwhelmingly often was

male; the sixth task was to be carried out by 'coders'—almost always female."[78] Although this separation between operators, coders, and planners was not immediately accepted everywhere—the small Whirlwind group viewed itself more as a "model shop" in which coding, programming, and operations were mixed together—this hierarchical separation between what Philip Kraft calls the "head and the hand" became dominant as programming became a mass, commercial enterprise.[79]

SAGE (the Semi-Automatic Ground Environment) air defense system, widely considered the first large software project, was programmed by the Systems Development Corporation (SDC), an offshoot of the RAND Corporation. SDC had expanded from a few programmers to more than eight hundred by the late 1950s, making it by far the largest employer of programmers. Because its programmers went on to form the industry (it was dubbed the "university of programmers"), SAGE had a wide impact on the field's development. SAGE, however, not only taught people how to code but also inculcated a strict division of programming in which senior programmers (later systems analysts), who developed program specifications, were separated from programmers, who worked on coding specifications; they in turn were separated from the coders who turned coding specifications into documented machine code.[80] This separation, as Kraft has recorded, was still thriving in the 1970s.[81] This separation was also gendered. As Herbert D. Benington, one of the managers of SAGE, later narrated, "women turned out to be very good for the administrative programs. One reason is that these people tend to be fastidious—they worry how all the details fit together while still keeping the big picture in mind. I don't want to sound sexist, but one of our strongest groups had 80 percent women in it; they were doing the right kind of thing. The mathematicians were needed for some of the more complex applications."[82] Not accidentally, the SDC was spun off from the System Training Program, a group comprised of RAND psychologists focused on producing more effective groups.[83]

Buttressing this hierarchy was a strict system of control, "tools of a very complex nature" that did not survive SAGE. As Benington explains, these tools enabled managers to track and punish coders: "You could assign an individual a job, you could control the data that that individual had access to, you could control when that individual's program operated, and you could find out if that individual was playing the game wrong and punish that person. So we had a whole set of tools for design, for controlling of the team, for controlling of the data, and for testing the programs that were really quite advanced."[84] Because of this system of control, Benington viewed symbolic addressing and other moves to automate programming as "dangerous because they couldn't be well-disciplined." However, although automatic programming has been linked to empowerment, it has also led to the more thorough (because subtle and internalized) disciplining of programmers, which simultaneously empowers and disempowers programmers.

Indeed, this overt system of control and punishment was replaced by a "softer" system of structured programming that makes source code source. As Mahoney has argued, structured programming emerged as a "means both of quality control and of disciplining programmers, methods of cost accounting and estimation, methods of verification and validation, techniques of quality assurance."[85] Kraft targets structured programming as de-skilling: through it, programming was turned from a craft to an industrialized practice in which workers were reduced to interchangeable detail workers.[86] Structured programming limits the logical procedures coders can use and insists that the program consist of small modular units, which can be called from the main program. Structured programming (also generally known as "good programming" when I was growing up) hides, and thus secures, the machine. It focuses on and enables abstraction—and abstraction from the specific uses of and for the machine—thereby turning programming from a numerical- to a problem-based task.

Not surprisingly, having little to no contact with the actual machine enhances one's ability to think abstractly rather than numerically. Edsger Dijkstra, whose famous condemnation of "goto" statements has encapsulated to many the fundamental tenets of structured programming, believes that he was able to "pioneer" structured programming precisely because he began his programming career by coding for ghosts: for machines that did not yet exist.[87] In "Go To Statement Considered Harmful," Dijkstra argues, "the quality of programmers is a decreasing function of the density of go to statements in the programs they produce" because goto statements work against the fundamental tenet of what Dijkstra considered to be good programming, namely, the necessity to "shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible."[88] This is important because, if a program suddenly halts because of a bug, gotos (statements that tell a program to go to a specific line if a condition is met) make it difficult to find the place in the program that corresponds to the buggy code. Gotos make difficult the conflation of instruction with its product—the reduction of process to command—that grounds the emergence of software as a concrete entity and commodity. That is, gotos make it difficult for the source program to act as a legible source.[89] As this example makes clear, structured programming moves away from issues of program efficiency—the time it takes to run a program—and more toward the problem of minimizing all the costs involved in producing and maintaining large programs. This move also makes programming an "art." As Dijkstra argues in his letter justifying structured programming, "it is becoming most urgent to stop to consider programming primarily as the minimization of cost/performance ratio. We should recognize that already now programming is much more an intellectual challenge: the art of programming is the art of organizing complexity, of mastering multitude and avoiding its bastard chaos as effectively as possible."[90] Again, this depends on making "the

structure of the program text [reflect] the structure of the computation."[91] It means moving away from assembly and other languages that routinely offer bizarre exits and self-modifying code to languages that feature clear and well-documented repetitions (while . . . do . . .) that end in one clear place, that return control to the main program.

Structured programming languages "save" programmers from themselves by providing good security, where security means secure from the programmer (increasingly, "securing" the machine means making sure programmers cannot access or write over key systems).[92] Indeed, structured programming, which emphasizes programming as a problem of flow, is giving way to data abstraction, which views programming as a problem of interrelated objects, and hides far more than the machine. Data abstraction depends on information hiding, on the nonreflection of changeable facts in software. As John V. Guttag, a "pioneer" in data abstraction explains, data abstraction is all about forgetting, about hiding information about how a type is implemented behind an interface.[93] Rather than "polluting" a program by enabling invisible lines of contact between supposedly independent modules, data abstraction presents a clean or "beautiful" interface by confining specificities, and by reducing the knowledge and power of the programmer. Knowledge, Guttag insists, is dangerous: "'Drink deep, or taste not the Pierian Spring,' is not necessarily good advice. Knowing too much is no better, and often worse, than knowing too little. People cannot assimilate very much information. Any programming method or approach that assumes that people will understand a lot is highly risky."[94] Abstraction—the "erasure of difference in the service of likeness or equality"—also erases, or "forgets," knowledge, rendering it, like the machine, ghostly.[95]

Thus abstraction both empowers the programmer and insists on his/her ignorance—the dream of a sovereign subject who knows and commands is constantly undone. Because abstraction exists "in the mind of the programmer," abstraction gives programmers new creative abilities. Computer scientist David Eck argues, "every programming language defines a virtual machine, for which it is the machine language. Designers of programming languages are creating computing machines as surely as the engineer who works in silicon and copper, but without the limitations imposed by materials and manufacturing technology."[96] However, this abstraction—this move away from the machine specificities—hands over, in its virtual separation of machine into software and hardware, the act of programming to the machine itself. Mildred Koss scoffed at the early notion of computers as brains because "they couldn't think in the way a human thinks, but had to be given a set of step-by-step machine instructions to be executed before they could provide answers to a specific problem"—at that time software was not considered to be an independent object.[97] The current status of software as a commodity, despite the nonrivalrous nature of "instructions," indicates the triumph of the software industry, an industry that first struggled not only financially but also conceptually to define its product. The rise of software

depends both on historical events, such as IBM's unbundling of its services from its products, and on abstractions enabled by higher-level languages. Guttag's insistence on the unreliability and incapability of human beings to understand underscores the cost of such an abstraction. Abstraction is the computer's game, as is programming in the strictest and newest sense of the word: with "data-driven" programming, for instance, machine learning/artificial intelligence (computers as source of source code) has become mainstream.

Importantly, this stratification and disciplining of labor has a much longer history: human computing itself, as David Grier has documented, moved from an art to a routinized procedure through a separation of planners from calculators.[98] Whereas the mathematician Alexis-Claude Clairaut called on two of his colleagues/friends, Joseph Lalande, Nicole-Reine Lapaute, in 1757 to calculate the date of Halley's comet's 1758 return, Gaspard Clair François Marie Riche de Prony, director of the Bureau du Cadastre, devised a system of intellectual labor to calculate metric tables in 1791. Not accidentally, the tables were part of a revolutionary governmental project: the move to the metric system by the National Assembly in order to gain control of the French economy.[99] De Prony, inspired by Adam Smith, divided the group into manual workers (unemployed pre-Revolutionary wig makers or servants who had basic arithmetic skills) and planners (experienced computers who planned the calculation). This system in turn inspired Charles Babbage's difference and analytic engines, in which the engines would replace the manual workers: according to Grier, de Prony's system showed Babbage that "the division of labor was not restricted to physical work but could be applied to 'some of the sublimest investigations of the human mind,' including the work of calculation."[100] This routinized calculation was not smoothly adopted; for a long time within the United States, such a model was resisted and, even during World War I, computers were graduate students and young assistant professors. In order to produce calculations necessary for governmental projects (such as eugenics, census, navigation, weapons, etc.) in the twentieth century, however, mass computation became the norm.

The U.S. wholesale embrace of mass calculation also coincides with a governmental project. Begun during the Great Depression as a way to put unemployed high school graduates to work, the Work Progress Administration's (WPA) Math Tables Project (MTP) produced some of the finest error-free tables in the world.[101] Indeed, it was not until the Roosevelt administration and the New Deal that the United States became seriously involved in producing mathematical tables. Since it was a WPA project, many established academics refused to be involved with it. To gain credibility, those in charge (themselves "less desirable" or unconventional PhDs) were determined to produce the most accurate tables possible. Gertrude Blanch, who ran the program with Milton Abramowitz, insists that most of the people they hired were qualified.[102] In contrast, Ida Rhodes, another PhD hired by the MTP, claims: "[Most] of the people

[who] came to us really knew nothing at all about mathematics or [even] arithmetic. Gertrude Blanch says that they were all High School graduates, and they may have been. I never checked on that. But if they were, very few of them had remembered anything about the arithmetic or the algebra or whatever mathematics they had [studied]."[103] By the end, however, they were transformed. According to Rhodes, Blanch performed miracles, "welding a malnourished, dispirited crew of people, coming from [the] Welfare Rolls, [into] a group that Leslie J. Comrie said was the 'mightiest computing team the world had ever seen.'"[104] To Rhodes, the social work involved in this project—"[salutary benefit conferred on] the spirit of those people [by] raising them from abject and self-despising people into a team that [acquired] a magnificent esprit de corps"—has been overlooked.[105] As Rhodes's rhetoric indicates, this was a patronizing if admirable project, run by "saints." Rhodes, herself partially deaf, would become an advocate for including physically challenged people in programming work. (Blanch interestingly had a more edgy view of sainthood. Describing Rhodes, she remarked, "if there are saints on earth, she's one of them. Saints may be difficult to live with but . . . it's nice to have a few around").[106]

This saintly salutary work comprised dividing the group into four categories, listed in ascending ability—the adders, the multipliers, the dividers, and the checkers—and creating worksheets so that "people who knew nothing about mathematics could [do advanced functions] by just following one step at a time."[107] The flawlessness of these tables stemmed both from these worksheets, created by Blanch, and from the degree to which these tables were checked (the Bessel function, for instance, was checked more than twenty-two times). Since the goal of the project was to keep these people busy, as well as to produce tables, accuracy was stressed over expediency and over sophistication of numerical techniques. Accuracy, according to Rhodes, became an obsession."[108]

Not surprisingly, though, the MTP computers were sometimes suspicious of their oversight. Rhodes relates, "we had impressed upon our workers over and over and over again that we were not watching them. We were not counting their output." Rather, "the only thing we asked of them is complete accuracy." This accuracy was also inscribed in the worksheets themselves in a nontransparent, repetitive manner. Rhodes and Blanch created worksheets, "in which every operation had to be done at least twice" and in which this duplicity was hidden. Rhodes explains, "for example, if we added a and b we wouldn't immediately say: add b and a. But some time later we saw to it that b got added to a, and we had arrows connecting the answers saying that these two answers should agree to, say one or two [units in] the last place. If they did not get such an agreement, then they were to [erase the pertinent portion] and [re-compute it]."[109] Again, the fact that these tables were largely unnecessary—and hence not time-sensitive—made this emphasis on accuracy over timeliness possible.

According to Rhodes, only two girls did not internalize the accuracy-ethic and cheated.[110] Rhodes revealingly narrates the dishonesty of the "colored" girl who joined the group after claiming that she was being discriminated against in another project:

> [Being] a softy, [I swallowed her story.] I should have checked with [her] boss and found out why she was not liked. But I didn't. And so I asked Gertrude's permission and she said, "All right, let's give her a chance." [And] she started working for us.
>
> Well, she hadn't been with us long enough apparently to absorb that feeling of accuracy, although, of course, we also gave her the [same] lecture that we gave everybody else. She must have thought that the more she produces, the more we will think of her and the more anxious we will be to keep her. [Her checker] reported to us that the girl was a whiz, she handed in many more sheets than anyone else; and I began to feel very proud of myself, thinking, oh, I got [me a] good girl, working so hard.
>
> You see, all that the [checker did was to examine the values, connected by the] arrows and if they agreed within one or two [units,] he was satisfied. In her case he once mentioned, "It's remarkable, they agree to the very last place." That should have given me an idea, but I was too busy with other things. Well, one evening Gertrude and I sat down to do our regular job of checking the sheets, and [when] we got [to] hers, [no values] differenced, absolutely nothing differenced. That was something we couldn't believe. How could [they] not difference? The arrows showed perfect agreement—too perfect, as a matter of fact.
>
> Well, lots of things can happen. First of all, the formula can be wrong. [Or we] could have made a mistake [in breaking down] the formula [while preparing] the worksheet. [Or] we could have made a mistake in [a sign.] We could have made a mistake in a constant. It happened to be my worksheet, so I checked [it] over: no mistake there. [She had to] copy certain information from other Tables. Maybe [I] gave her the wrong tables. [An examination showed] that she copied the correct Tables. What else could have happened? The point [is] that we were so innocent and so trusting, it never occurred to us that what really happened [could have occurred.] What had happened was that she would get the first answer, and then when she got to [it] the second time — where the arrows showed that they had to agree — and [they] didn't agree, she merely erased the [second] answer and copied down the first [one.] We found that out [when] Gertrude and I recomputed all her sheets.[111]

This remarkable story reveals the contradictions in this disciplinary system: although Rhodes denies that they judged performance by speed, she thinks she got herself a "good girl" when the "colored girl" performs quickly. Also, although math presumably requires some intellectual labor, intelligence is condemned. The "colored girl"'s ability to figure out the system, the algorithm, is denounced as cheating, and the managers' faith in their own nontransparent plans described as "trusting." These worksheets were an early form of programming: a breakdown of a complex operation into sequence of simple operations that depends on accurate and single-minded calculation. As this example makes clear, such programming depended on mind-numbingly repetitive operations by the "dumb" and the downtrodden, whose inept or deceitful actions could disrupt the task at hand. Modern computing replaces these with vacuum tubes and transistors.

As Alan Turing contended, "the class of problems capable of solution by the machine can be defined fairly specifically . . . [namely] those problems which can be solved by human clerical labour, working to fixed rules, and without understanding."[112]

Source code become "thing"—the erasure of execution—follows from the mechanization of these power relations, the reworking of subject-object relations through automation as both empowerment and enslavement and through repetition as both mastery and hell. Embedded within the notion of instruction as source and the drive to automate computing—relentlessly haunting them—is a constantly repeated narrative of liberation and empowerment, wizards and (ex-)slaves.

Automation as Sourcery

Automatic programming, what we could call programming today, reveals the extent to which automation and the history of programming cannot be considered a simple deskilling (Kraft's argument) or a march toward greater human power. Rather, through automation, expertise is both created and called into question: it is something that coders did not simply fear, but also appreciated and drove.

Automatic programming arose from a desire to reuse code and to recruit the computer into its own operation—essentially, to transform singular instructions into a language a computer could write. As Koss, an early UNIVAC programmer, explains:

> Writing machine code involved several tedious steps—breaking down a process into discrete instructions, assigning specific memory locations to all the commands, and managing the I/O buffers. After following these steps to implement mathematical routines, a sub-routine library, and sorting programs, our task was to look at the larger programming process. We needed to understand how we might reuse tested code and have the machine help in programming. As we programmed, we examined the process and tried to think of ways to abstract these steps to incorporate them into higher-level language. This led to the development of interpreters, assemblers, compilers, and generators—programs designed to operate on or produce other programs, that is, automatic programming.[113]

Automatic programming is an abstraction that allows the production of computer-enabled human-readable code—key to the commodification and materialization of software and to the emergence of higher-level programming languages.

Higher-level programming languages, unlike assembly language, explode one's instructions and enable one to forget the machine. In them, simple operations often call a function, making it a metonymic language par excellence. These languages also place everyone in the position of the planner, without the knowledge of the coder. They enable one to run a program on more than one machine—a property now assumed to be a "natural" property of software ("direct programming" led to a unique configuration of cables; early machine language could be iterable but only on the same machine—assuming, of course, no engineering faults or failures). In order to emerge

as a language or a source, software and the "languages" on which it relies had to become iterable. With programming languages, the product of programming would no longer be a running machine but rather this thing called software—something theoretically (if not practically) iterable, repeatable, reusable, no matter who wrote it or what machine it was destined for; something that inscribes the absence of both the programmer and the machine in its so-called writing.[114] Programming languages enabled the separation of instruction from machine, of imperative from action, a move that fostered the change in the name of source code itself, from "pseudo" to "source." Pseudocode intriguingly stood both for the code as language and for the code as program (i.e., source code). The manual for UNIVAC's A-2 compiler, for instance, defines pseudocode as "computer words other than the machine (C-10) code, design [sic] with regard to facilitating communications between programmer and computer. Since a pseudo-code cannot be directly executed by the computer, there must be programmed a modification, interpretation or translation routine which converts the pseudo-codes to machine instruction and routines."[115] Pseudocode, which enables one to move away from machine specificity, is called "information"—what later would become a ghostly immaterial substance—rather than code.

According to received wisdom, these first attempts to automate programming—the "pseudo"—were resisted by "real" programmers.[116] John Backus, developer of FORTRAN, claims that early machine language programmers were engaged in a "black art"; they had a "chauvinistic pride in their frontiersmanship and a corresponding conservatism, so many programmers of the freewheeling 1950s began to regard themselves as members of a priesthood guarding skills and mysteries far too complex for ordinary mortals."[117] Koss similarly argues, "without these higher-level languages and processes . . . , which democratized problem solving with the computer, I believe programming would have remained in the hands of a relatively small number of technically oriented software writers using machine code, who would have been essentially the high priests of computing."[118]

This story of a "manly" struggle against automatic programming resonates with narratives of mechanical computing itself as "feminizing" numerical analysis. Whirlwind team member Bob Everett offers the following summary of a tale describing two different ways of approaching automatic computing, which was told at Aiken's mid-1940s meeting: "One was the woman who gets married, and that's fine, and she looks ahead to a life-time of three meals a day, 365 days a year, and dishes to wash after each one of them. Her husband brings her home from the honeymoon, and she discovers he's bought her an automatic dishwasher. That's one way. The other way is the guy who decides to climb a mountain, and he buys all the rope, pitons, and one thing and another, and he goes to the mountain and finds that somebody has built a funicular railway."[119] According to this description, automatic computing is feminine or emasculating: an escape from domestic drudgery or the automation of a properly

masculine enterprise. Thus, it is not just the introduction of automatic programming that inspired narratives of masculine expertise under siege, but also the introduction of—or, more properly, the appreciation of—the (automatic) computer.

In a related manner, Hopper (and perhaps only Hopper) experienced the U.S. Navy, in particular her initial training as a thirty-seven-year-old woman, as "the most complete freedom I'd ever had." Whereas her younger counterparts rebelled "against the uniforms and the regulations," she embraced the Navy's strict structure as a release from domestic duties. As she relates, "All of a sudden I didn't have to decide anything, it was all settled. I didn't even have to bother to decide what I was going to wear in the morning, it was there. I just picked it up and put it on. So for me all of a sudden I was relieved of all minor decisions. . . . I didn't even have to figure out what I was going to cook for dinner." The difficulties of domestic life and sacrifice during World War II colored Hopper's enthusiasm, since "housekeeping had gotten to be quite a chore by then to figure out how much meat you could have and could you give dad some sugar 'cause he loved it and you might have some extra points. That's when I learned to drink most of my drinks without any sugar in them so that dad could have it. And we had very little gasoline and we had to have a car and you had to plan every trip very carefully. Well, all of a sudden I'm in midshipmen's school and all of a sudden you don't have to do any of it.[120] Importantly, though, this release was also an insertion into a well-defined system, in which one both gave and received commands. When a *Voice of America* interviewer asked, "You are supposed to command, but also to conform and obey. How do you come to terms with those two extremes?" Hopper replied, "The essential basic principle of the Navy is leadership. And leadership is a two-way street. It is loyalty up and loyalty down. Respect your superior, keep him informed of what you are doing, and take care of your crew. That is everyone's responsibility."[121]

Automatic programming, seen as freeing oneself from both drudgery and knowledge, thus calls into question the simple narrative of it as dispersing a reluctant "priesthood" of machine programmers. This narrative of resistance assumes that programmers naturally enjoyed tedious and repetitive numerical tasks and as well as developing singular solutions for their clients. The "mastery" of computing can easily be understood as "suffering." Indeed, Hopper called her early days with the Harvard Mark 1 her "sufferings" and argued, "experienced programmers are always anxious to make the computer carry out as much routine work as they can."[122] Harry Reed, an early ENIAC programmer, relays, "the whole idea of computing with the ENIAC was a sort of *hair-shirt* kind of thing. Programming for the computer, whatever it was supposed to be, was a redemptive experience—one was supposed to *suffer* to do it." According to Reed, programmers were actively trying to convince people to write small programs for themselves. In the 1970s, he "actually had to take my Division and sit everybody down who hadn't taken a course in FORTRAN, because, by God, they were going to write their own programs now. We weren't going to get computer

specialists to write simple little programs that they should have been writing.[123] Also, the first programmers were the first writers of reusable subroutines. Holberton, for instance, developed the first SORT generator to save her colleagues' time, "I felt for all the work that Betty Jean and I had done on sorting methods, it was a shame for people to have to sit down and re-do and re-code that same thing even though they could use the books to do it, if it could be done by a machine. And that's the reason, and it only took six months to program the thing. That's six more months."[124] Thus, rather than programmers circling the wagons to protect their positions, it would seem that many programmers themselves welcomed and contributed to the success of automatic programming.

As well, since programmers were in incredible demand in the 1950s through the 1960s, the need to create boundaries to protect jobs seems odd. Although compilers and interpreters may not have been accepted immediately, especially by those already trained in machine programming, the resistance may have stemmed more from the work environment than from personal arrogance. Coders were under great pressure to be as efficient as possible. As Holberton and Bartik relay in a 1973 interview, early coders often developed a persecution complex, because machine time was the most important and expensive thing:

BARTIK: The worst sin that you could commit was to waste that machine time. So that we really became paranoid.
HOLBERTON: Mhm. Efficiency.
BARTIK: We thought everybody was after us.
TROPP: [Laughter]
BARTIK: For our inefficiency.
HOLBERTON: You wasted one add time, you were being inefficient.
BARTIK: So it was fine for us to struggle for two days to cut off the slightest amount on that machine.[125]

Compilers were arguably accepted because the demand for programmers meant a loss in quality (an ever widening recruitment)—programming efficiently in machine language therefore became a mark of expertise. In this sense, the introduction of automatic programming, which set a certain standard of machine efficiency, helped to produce the priesthood it was supposedly displacing.

Corporate and academic customers, for whom programmers were orders of magnitude cheaper per hour than computers, do seem to have resisted automatic programming. Jean Sammet, an early female programmer, relates, in her influential *Programming Languages: History and Fundamentals*, that customers objected to compilers on the ground that they "could not turn out object code as good as their best programmers. A significant selling campaign to push the advantages of such systems was underway at that time, with the spearhead being carried for the numerical scientific languages (i.e., FORTRAN) and for 'English-language-like' business data-processing languages by

Remington Rand (and Dr. Grace Hopper in particular)."[126] This selling campaign not only pushed higher-level languages (by devaluing humanly produced programs), it also pushed new hardware: to run these programs, one needed more powerful machines. The government's insistence on standardization, most evident in the development and widespread use of COBOL, itself a language designed to open up programming to a wider range of people, fostered the general acceptance of higher-level languages, which again were theoretically, if not always practically, machine independent or iterable. The hardware-upgrade cycle was normalized in the name of saving programming time.

This "selling campaign" led to what many have heralded as the democratization of programming, the opening of the so-called priesthood of programmers. In Sammet's view, this was a partial revolution

> in the way in which computer installations were run because it became not only possible, but quite practical to have engineers, scientists, and other people actually programming their own problems without the intermediary of a professional programmer. Thus the conflict of the open versus closed shop became a very heated one, often centering [on] the use of FORTRAN as the key illustration for both sides. This should not be interpreted as saying that all people with scientific numerical problems to solve immediately sat down to learn FORTRAN; this is clearly not true but such a significant number of them did that it has had a major impact on the entire computer industry. One of the subsidiary side effects of FORTRAN was the introduction of FORTRAN Monitor System [IB60]. This made the computer installation much more efficient by requiring less operator intervention for the running of the vast number of FORTRAN (as well as machine language) programs.[127]

The democratization or "opening" of computing, which gives the term *open* in *open source* a different resonance, would mean the potential spread of computing to those with scientific numerical problems to solve and the displacement of human operators by operating systems. But the language of priests and wizards has hardly faded and scientists have always been involved with computing, even though computing has not always been considered to be a worthy scientific pursuit. The history of computing is littered with moments of "computer liberation" that are also moments of greater obfuscation.[128]

Higher level programming languages—automatic programming—may have been sold as offering the programmer more and easier control, but they also necessitated blackboxing even more the operations of the machine they supposedly instructed. Democratization did not displace professional programmers but rather buttressed their position as professionals by paradoxically decreasing their real power over their machines, by generalizing the engineering concept of information.

So what are we to do with these contradictions and ambiguities? As should be clear by now, these many contradictions riddling the development of automatic programming were key to its development, for the automation of computing is both an

acquisition of greater control and freedom, and a fundamental loss of them. The narrative of the "opening" of programming reveals the tension at the heart of programming and control systems: are they control systems or servomechanisms (Norbert Wiener's initial name for them)? Is programming a clerical activity or an act of Hobbesian mastery? Given that the machine takes care of "programming proper"—the sequence of events during execution—is programming programming at all? What is after all compacted in the coinciding changes in the titles of "operators" to "programmers" and of "mathematicians" to "programmers"? The notion of the priesthood of programming erases this tension, making programming always already the object of jealous guardianship, and erasing programming's clerical underpinnings.[129]

Programming in the 1950s does seem to have been fun and fairly gender balanced, in part because it was so new and in part because it was not as lucrative as hardware design or even sales: the profession was gender neutral in hiring if not pay because it was not yet a profession.[130] The "ENIAC girls" were first hired as subprofessionals, and some had to acquire more qualifications in order to retain their positions. As many female programmers quit to have children or get married, men (and compilers) took their increasingly lucrative positions. Programming's clerical and arguably feminine underpinnings—both in terms of personnel and of command structure—became buried as programming sought to become an engineering and academic field in its own right.[131] Democratization did not displace professional programmers but rather buttressed their position as professionals by paradoxically decreasing their real power over their machines. It also, however, made programming more pleasurable.

Causal Pleasure

The distinction between programmers and users is gradually eroding. With higher-level languages, programmers are becoming more like simple users. Crucially, though, the gradual demotion of programmers has been offset by the power and pleasure of programming. To program in a higher-level language is to enter a magical world—it is to enter a world of logos, in which one's code faithfully represents one's intentions, albeit through its blind repetition rather than its "living" status.[132] Edwards argues, "programming can produce strong sensations of power and control" because the computer produces an internally consistent if externally incomplete microworld, a "simulated world, entirely within the machine itself, that does not depend on instrumental effectiveness. That is, where most tools produce effects on a wider world of which they are only a part, the computer contains its own worlds in miniature. . . . In the microworld, as in children's make-believe, the power of the programmer is absolute."[133] Joseph Weizenbaum, MIT professor, creator of ELIZA (an early program that imitated a Rogerian therapist) and member of the famed MIT AI (Artifical Intelligence) lab, similarly contends:

> The computer programmer . . . is a creator of universes for which he alone is the lawgiver. So, of course, is the designer of any game. But universes of virtually unlimited complexity can be created in the form of computer programs. Moreover, and this is a crucial point, systems so formulated and elaborated *act out* their programmed scripts. They compliantly obey their laws and vividly exhibit their obedient behavior. No playwright, no stage director, no emperor, however powerful, has ever exercised such absolute authority to arrange a stage or a field of battle and to command such unswervingly dutiful actors or troops.[134]

The progression from playwright to stage director to emperor is telling: programming languages, like neoliberal economics, model the world as a "game."[135] To return to the notion of "code is law," programming languages establish the programmer as a sovereign subject, for whom there is no difference between command given and command completed. As a lawgiver more powerful than a playwright or emperor, the programmer can "say" "let there be light" and there is light. Iterability produces both language and subject. Importantly, Weizenbaum views the making performative or automatically executable of words as the imposition of instrumental reason, inseparable from the process of "enlightenment" critiqued by the Frankfurt school.[136] Instrumental reason, he argues, "has made out of words a fetish surrounded by black magic. And only the magicians have the rights of the initiated. Only they can say what words mean. And they play with words and they deceive us."[137]

Programming languages offer the lure of visibility, readability, logical if magical cause and effect. As Brooks argues, "one types the correct incantation on the keyboard, and a display screen comes to life, showing things that never were nor could be."[138] One's word creates something living. Consider this ubiquitous "hello world" program written in C++ ("hello world" is usually the first program a person will write):

```
// this program spits out "hello world"
#include <iostream.h>
int main ()
{
  cout << "Hello World!";
  return 0;
}
```

The first line is a comment line, explaining to the human reader that this program spits out "Hello World!." The next line directs the compiler's preprocessor to include iostream.h, a standard file to deal with input and output to be used later. The third line, "int main (),"begins the main function of the program; "cout << 'Hello World!';" prints "Hello World!" to the screen ("cout" is defined in iostream.h); "return 0" terminates the main function and causes the program to return a 0 if it has run correctly.

Although not immediately comprehensible to someone not versed in C++, this program nonetheless seems to make some sense, and seems to be readable. It comprises a series of imperatives and declaratives that the computer presumably understands and obeys. When it runs, it follows one's commands and displays "Hello World!."

It is no accident that "hello world" is the first program one learns because it is easy, demonstrating that we can produce results immediately. This ease, according to Weizenbaum, is what makes programming so seductive and dangerous:

> It happens that programming is a relatively easy craft to learn. . . . And because programming is almost immediately rewarding, that is, because a computer very quickly begins to behave somewhat in the way the programmer intends it to, programming is very seductive, especially for beginners. Moreover, it appeals most to precisely those who do not yet have sufficient maturity to tolerate long delays between an effort to achieve something and the appearance of concrete evidence of success. Immature students are therefore easily misled into believing that they have truly mastered a craft of immense power and of great importance when, in fact, they have learned only its rudiments and nothing substantive at all.[139]

The seeming ease of programming hides a greater difficulty—executability leads to unforeseen circumstances, unforeseen or buggy repetitions. Programming offers a power that, Weizenbaum argues, corrupts as any power does.[140] *What corrupts, Weizenbaum goes on to explain, however, is not simply ease, but also this combination of ease and difficulty*. Weizenbaum argues that programming creates a new mental disorder: the compulsion to program, which he argues hackers, who "hack code" rather than "work," suffer from (although he does note that not all hackers are compulsive programmers).[141]

To explain this addiction, Weizenbaum explains the parallels between "the magical world of the gambler" and the magical world of the hacker—both entail megalomania and fantasies of omnipotence, as well as a "pleasureless drive for reassurance."[142] Like gambling, programming can be compulsive because it both rewards and challenges the programmer. It is driven by "two apparently opposing facts: first, he knows that he can make the computer do anything he wants it to do; and second, the computer constantly displays undeniable evidence of his failures to him. It reproaches him. There is no escaping this bind. The engineer can resign himself to the truth that there are some things he doesn't know. But the programmer moves in a world entirely of his own making. The computer challenges his power, not his knowledge."[143] According to Weizenbaum, because programming engages power rather than truth, it can induce a paranoid megalomania in the programmer.[144] Because this knowledge is never enough, because a new bug always emerges, because an unforeseen wrinkle causes divergent unexpected behavior, the hacker can never stop. Every error seems correctable; every error points to the hacker's lack of foresight; every error leads to another. Thus, unlike the "useful programmer," who "works" by solving the problem at hand and carefully documents his code, the hacker aimlessly hacks code: programming

becomes a technique, a game without a goal and thus without an end. Hackers' skills are thus "disembodied" and this disembodiment transforms their physical appearance: Weizenbaum describes them as "bright young men of disheveled appearance, often with sunken glowing eyes . . . sitting at computer consoles, their arms tensed and waiting to fire their fingers, already poised to strike, at the buttons and keys on which their attention seems to be as riveted as a gamer's on the rolling dice.[145]

Although Weizenbaum is quick to pathologize hackers as pleasureless pitiful creatures, hackers themselves emphasize programming as pleasurable—and their lack of "usefulness" can actually be what is most productive and promising about programming. Linus Torvalds, for instance, argues that he, as an eternal grad student, decided to build the Linux operating system core just "for fun." Torvalds further views the decisions programming demands as rescuing programming from becoming tedious. "Blind obedience on its own, while initially fascinating," he writes, "obviously does not make for a very likable companion. In fact, that part gets boring fairly quickly. What makes programming so engaging is that, while you can make the computer do what you want, you have to figure out *how*."[146] Richard Stallman, who fits Weizenbaum's description of a hacker (and who was in the AI lab, probably building those indispensable functions) likewise emphasizes the pleasure, but more important the "freedom" and "freeness" associated with programming—something that stems from programming as not simply the production of a commercial (or contained) product. Hacking reveals the extent to which source code can become a fetish: something endless that always leads us pleasurably, as well as anxiously, astray.

Source Code as Fetish

Source code as source means that software functions as an axiom, as "a self-evident proposition requiring no formal demonstration to prove its truth, but received and assented to as soon as it is mentioned."[147] In other words, whether or not source code is only a source after the fact or whether or not software can be physically separated from hardware,[148] software is always posited as already existing, as the self-evident ground or source of our interfaces. Software is axiomatic. As a first principle, it fastens in place a certain neoliberal logic of cause and effect, based on the erasure of execution and the privileging of programming that bleeds elsewhere and stems from elsewhere as well.[149] As an axiomatic, it, as Gilles Deleuze and Félix Guattari argue, artificially limits decodings.[150] It temporarily limits what can be decoded, put into motion, by setting up an artificial limit—the artificial limit of programmability—that seeks to separate information from entropy, by designating some entropy information and other "non-intentional" entropy noise. Programmability, discrete computation, depends on the disciplining of hardware and programmers, and the desire for a programmable axiomatic code. Code, however, is a medium in the full sense of the word. As a

medium, it channels the ghost that we imagine runs the machine—that we see as we don't see—when we gaze at our screen's ghostly images.

Understood this way, source code is a fetish. According to the OED, a fetish was originally an ornament or charm worshipped by "primitive peoples . . . on account of its supposed inherent magical powers."[151] The term *fetisso* stemmed from the trade of small wares and magic charms between the Portuguese merchants and West Africans; Charles de Brosses coined the term *fetishism* to describe "primitive religions" in 1757. According to William Pietz, Enlightenment thinkers viewed fetishism as a "false causal reasoning about physical nature" that became "the definitive mistake of the pre-enlightened mind: it superstitiously attributed intentional purpose and desire to material entities of the natural world, while allowing social action to be determined by the . . . wills of contingently personified things, which were, in truth, merely the externalized material sites fixing people's own capricious libidinal imaginings."[152] That is, fetishism, as "primitive causal thinking," derived causality from "things"—in all the richness of this concept—rather than from reason:

> Failing to distinguish the intentionless natural world known to scientific reason and motivated by practical material concerns, the savage (so it was argued) superstitiously assumed the existence of a unified causal field for personal actions and physical events, thereby positing reality as subject to animate powers whose purposes could be divined and influenced. Specifically, humanity's belief in gods and supernatural powers (that is, humanity's unenlightenment) was theorized in terms of prescientific peoples' substitution of imaginary personifications for the unknown physical causes of future events over which people had no control and which they regarded with fear and anxiety.[153]

A fetish allows one to visualize what is unknown—to substitute images for causes. Fetishes allow the human mind both too much and not enough control by establishing a "unified causal field" that encompasses both personal actions and physical events. Fetishes enable a semblance of control over future events—a possibility of influence, if not an airtight programmability—that itself relies on distorting real social relations into material givens.

This notion of fetish as false causality has been most important to Karl Marx's diagnosis of capital as fetish. Marx famously argued:

> the commodity-form . . . is nothing but the determined social relation between men themselves which assumes here, for them, the phantasmagoric form of a relation between things. In order, therefore, to find an analogy we must take a flight into the misty realm of religion. There the products of the human head appear as autonomous figures endowed with a life of their own, which enter into relations both with each other and with the human race. So it is in the world of commodities with the products of men's hands. I call this the . . . fetishism.[154]

The capitalist thus confuses social relations and the labor activities of real individuals with capital and its seemingly magical ability to reproduce. For, "it is in interest-

bearing capital . . . that capital finds its most objectified form, its pure fetish form. . . . Capital—as an entity—appears here as an independent source of value; a something that creates value in the same way as land [produces] rent, and labor wages."[155] Both these definitions of fetish also highlight the relation between things and men: men and things are not separate, but rather speak with and to one another. That is, things are not simply objects that exist outside the human mind, but are rather tied to events, to the timing of events.

The parallel to source code seems obvious: we "primitive folk" worship source code as a magical entity—as a source of causality—when in truth the power lies elsewhere, most importantly, in social and machinic relations. If code is performative, its effectiveness relies on human and machinic rituals. Intriguingly though, in this parallel, Enlightenment thinking—a belief that knowing leads to control, to a release from tutelage—is not the "solution" to the fetish, but, rather, what grounds it, for source code historically has been portrayed as the solution to wizards and other myths of programming: machine code provokes mystery and submission; source code enables understanding and thus institutes rational thought and freedom. Knowledge, according to Weizenbaum, sustains the hacker's aimless actions. To offer a more current example of this logic than the FORTRAN one cited earlier, Richard Stallman, in his critique of nonfree software, has argued that an executable program "is a mysterious bunch of numbers. What it does is secret."[156] Against this magical execution, source code supposedly enables an understanding and a freedom—the ability to map and know the workings of the machine, but, again, only through a magical erasure of the gap between source and execution, an erasure of execution itself. If we consider source code as fetish, the fact that source code has hardly deprived programmers of their priestlike/wizard status makes complete sense. If anything, such a notion of programmers as superhuman has been disseminated ever more and the history of computing—from direct manipulation to hypertext—has been littered by various "liberations."

But clearly, source code can do and be things: it can be interpreted or compiled; it can be rendered into machine-readable commands that are then executed. Source code is also read by humans and is written by humans for humans and is thus the source of some understanding. Although Ellen Ullman and many others have argued, "a computer program has only one meaning: what it does. It isn't a text for an academic to read. Its entire meaning is its function," source code must be able to function, even if it does not function—that is, even if it is never executed.[157] Source code's readability is not simply due to comments that are embedded in the source code, but also due to English-based commands and programming styles designed for comprehensibility. This readability is not just for "other programmers." When programming, one must be able to read one's own program—to follow its logic and to predict its outcome, whether or not this outcome coincides with one's prediction.

This notion of source code as readable—as creating some outcome regardless of its machinic execution—underlies "codework" and other creative projects. The Internet artist Mez, for instance, has created a language called mezangelle that incorporates formal code and informal speech. Mez's poetry deliberately plays with programming syntax, producing language that cannot be executed, but nonetheless draws on the conventions of programming language to signify.[158] Codework, however, can also work entirely within an existing programming language. Graham Harwood's perl poem, for example, translates William Blake's nineteenth-century poem "London" into London.pl, a script that contains within it an algorithm to "find and calculate the gross lung-capacity of the children screaming from 1792 to the present."[159] Regardless of whether or not it can execute, code can be—must be—worked into something meaningful. Source code, in other words, may be the source of things other than the machine execution it is "supposed" to engender.

Source code as fetish, understood psychoanalytically, embraces this nonteleological potential of source code, for the fetish is a deviation that does not "end" where it should. It is a genital substitute that gives the fetishist nonreproductive pleasure. It allows the child to combat castration—his inscription within the world of paternal law and order—for both himself and his mother, while at the same time accommodating to his world's larger oedipal structure. It both represses and acknowledges paternal symbolic authority. According to Freud, the fetish, formed the moment the little boy discovers his mother's "lack," is "a substitute for the woman's (mother's) phallus which the little boy once believed in and does not wish to forego."[160] As such, it both fixes a singular event—turning time into space—and enables a logic of repetition that constantly enables this safeguarding. As Pietz argues, "the fetish is always a meaningful fixation of a singular event; it is above all a 'historical' object, the enduring material form and force of an unrepeatable event. This object is 'territorialized' in material space (an earthly matrix), whether in the form of a geographical locality, a marked site on the surface of the human body, or a medium of inscription or configuration defined by some portable or wearable thing."[161] Even though it fixes a singular event, the fetish works only because it can be repeated, but again, what is repeated is both denial and acknowledgment, since the fetish can be "the vehicle both of denying and asseverating the fact of castration."[162] Slavoj Žižek draws on this insight to explain the persistence of the Marxist fetish:

> When individuals use money, they know very well that there is nothing magical about it—that money, in its materiality, is simply an expression of social relations . . . on an everyday level, the individuals know very well that there are relations between people behind the relations between things. The problem is that in their social activity itself, in what they are *doing*, they are *acting* as if money, in its material reality is the immediate embodiment of wealth as such. They are fetishists in practice, not in theory. What they "do not know,"

> what they misrecognize, is the fact that in their social reality itself—in the act of commodity exchange—they are guided by the fetishistic illusion.[163]

Fetishists, importantly, know what they are doing—knowledge, again, is not an answer to fetishism, but rather what sustains it. The knowledge that source code offers is no cure for source code fetishism: if anything, this knowledge sustains it. As the next chapter elaborates, the key question thus is not "what do we know?" but rather "what do we do?"

To make explicit the parallels, source code, like the fetish, is a conversion of event into location—time into space—that does affect things, although not necessarily in the manner prescribed. Its effects can be both productive and nonexecutable. Also, in terms of denial and acknowledgment, we know very well that source code in that state and without the intercession of other "layers" is not executable, yet we persist in treating it as so. And it is this glossing over that makes possible the ideological belief in programmability.

Code as fetish means that computer execution deviates from the so-called source, as source program does from programmer. Turing, in response to the objection that computers cannot think because they merely follow human instructions, contends:

> Machines take me by surprise with great frequency. . . . The view that machines cannot give rise to surprises is due, I believe, to a fallacy to which philosophers and mathematicians are particularly subject. This is the assumption that as soon as a fact is presented to a mind all consequences of that fact spring into the mind simultaneously with it. It is a very useful assumption under many circumstances, but one too easily forgets that it is false. A natural consequence of doing so is that one then assumes that there is no virtue in the mere working out of consequences from data and general principles.[164]

This erasure of the vicissitudes of execution coincides with the conflation of data with information, of information with knowledge—the assumption that what is most difficult is the capture, rather than the analysis, of data. This erasure of execution through source code as source creates an intentional authorial subject: the computer, the program, or the user, and this source is treated as the source of meaning. The fact that there is an algorithm, a meaning intended by code (and thus in some way knowable), sometimes structures our experience with programs. When we play a game, we arguably try to reverse engineer its algorithm or at the very least link its actions to its programming, which is why all design books warn against coincidence or random mapping, since it can induce paranoia in its users. That is, because an interface is programmed, most users treat coincidence as meaningful. To the user, as with the paranoid schizophrenic, there is always meaning: whether or not the user knows the meaning, s/he knows that it regards him or her. To know the code is to have a form of "X-ray vision" that makes the inside and outside coincide, and the act of revealing sources or connections becomes a critical act in and of itself.[165] Code

as source leads to that bizarre linking of computers to visual culture, to transparency, which constitutes the subject of chapter 2.

Code as fetish thus underscores code as thing: code as a "dirty window pane," rather than as a window that leads us to the "source." Code as fetish emphasizes code as a set of relations, rather than as an enclosed object, and it highlights both the ambiguity and the specificity of code. Code points to, it indicates, something both specific and nebulous, both defined and undefinable. Code, again, is an abstraction that is haunted, a source that is a re-source, a source that renders the machinic—with its annoying specificities or "bugs"—ghostly. As Thomas Keenan argues, "haunting can only be thought as the difficult (simultaneous and impossible) movement of remembering and forgetting, inscribing and erasing, the singular and the different."[166] Embracing software as thing, in theory and in practice, opens us to the ways in which the fact that we cannot know software can be an enabling condition: a way for us to engage the surprises generated by a programmability that, try as it might, cannot entirely prepare us for the future.

Computers that Roar

Computers, like other media, are metaphor machines: they both depend on and perpetuate metaphors. More remarkably, though, they—through their status as "universal machines"—have become metaphors for metaphor itself.

From files to desktops, windows to spreadsheets, metaphors dominate user interfaces. In the 1990s (and even today), textbooks of human–computer interface (HCI) design described metaphors as central to "user-friendly" interfaces. Metaphors make abstract computer tasks familiar, concrete, and easy to grasp, since through them we allegedly port already existing knowledge to new tasks (for instance, experience with documents to electronic word processing). Metaphors proliferate not only in interfaces, but also in computer architecture: from memory to buses, from gates to the concept of architecture itself. Metaphors similarly structure software: viruses, UNIX daemons, monitors, back orifice attacks (in which a remote computer controls the actions of one's computer), and so on. At the contested "origin" of modern computing lies an analogy turned metaphor: John von Neumann deliberately called the major components of modern (inhuman) computers "organs," after cybernetic understandings of the human nervous system. Drawing from the work of Alan Turing and Charles Babbage, Jon Agar has argued that the computer, understood as consisting of software and hardware, is a "government machine." Like the British Civil Service, it is a "general-purpose 'machine' governed by a code."[1]

The role of metaphor, however, is not simply one way. Like metaphor itself, it moves back and forth. Computers have become metaphors for the mind, for culture, for society, for the body, affecting the ways in which we experience and conceive of "real" space: from the programmed mind running on the hard-wired brain to reprogrammable culture versus hard-wired nature, from neuronal networks to genetic programs. Paul Edwards has shown how computers as metaphors and machines were crucial to the Cold War and to the rise of cognitive psychology, an insight developed further by David Golumbia in his analysis of computationalism. As cited earlier, Joseph Weizenbaum has argued that computers have become metaphors for all "effective procedures," that is, for anything that can be solved in a prescribed number of steps,

such as gene expression and clerical work.[2] Weizenbaum also notes that the power of computer as metaphor is itself based on "only the vaguest understanding of a difficult and complex scientific concept. . . . The public vaguely understands—but is nonetheless firmly convinced—that any effective procedure can, in principle, be carried out by a computer . . . it follows that a computer can at least imitate man, nature, and society in all their procedural aspects."[3] Crucially, this means that, at least in popular opinion, the computer is a machine that can imitate, and thus substitute for, all others based on its programming. This vaguest understanding—software as thing—is neither accidental to nor a contradiction of the computer as metaphor, but rather grounds its appeal.

Because computers are viewed as universal machines, they have become metaphors for metaphor itself: they embody a logic of substitution, a barely visible conceptual system that orders and disorders. Metaphor is drawn from the Greek terms *meta* (change) and *phor* (carrying): it is a transfer that transforms. Aristotle defines metaphor as consisting "in giving the thing a name that belongs to something else; the transference being either from genus to species, or from species to genus, or from species to species, or on grounds of analogy."[4] George Lakoff and Mark Johnson argue, "*The essence of metaphor is understanding and experiencing one kind of thing in terms of another*."[5] Metaphor is necessary "because so many of the concepts that are important to us are either abstract or not clearly delineated in our experience (the emotions, ideas, time, etc.), we need to get a grasp on them by means of other concepts that we understand in clearer terms (spatial orientations, objects, etc.)."[6] Lakoff and Johnson argue that we live by metaphors (such as "argument is war," "events are objects," and "happy is up"), that they serve as the basis for our thoughts and our actions.[7] Metaphors govern our actions because they are also "grounded in our constant interaction with our physical and cultural environments."[8] That is, the similarities that determine a metaphor are based on our interactions with various objects—it is therefore no accident that metaphors are thus prominent in "interactive" design. Crucially, metaphors do not simply conceptualize a preexisting reality; they also create reality.[9] Thus, they are not something we can "see beyond," but rather things necessary to seeing. Even to see beyond certain metaphors, they argue, we need others.[10] Metaphor is an "imaginative rationality": "Metaphor . . . unites reason and imagination. Reason, at the very least, involves categorization, entailment, and inference. Imagination, in one of its many aspects, involves seeing one kind of thing in terms of another kind of thing—what we have called metaphorical thought."[11] This imaginative seeing one kind of thing in terms of another thing also involves hiding: a metaphor, Thomas Keenan argues, means that "something . . . shows itself by hiding itself, by announcing itself as something else or in another form."[12]

Paul Ricoeur, focusing more on metaphor as a linguistic entity, similarly stresses the centrality and creative power of metaphor. To Ricoeur, metaphor grounds the possibility of logical thought. Ricoeur, drawing from Aristotle's definition, argues that change, movement, and transposition (and thus deviation, borrowing, and substitution) characterize metaphor.[13] By transposing an "alien" name, metaphor is a "categorical transgression . . . a kind of deviance that threatens classification itself."[14] Since metaphor, however, also "'conveys learning and knowledge through the medium of the genus,'" Ricoeur contends, "metaphor destroys an order only to invent a new one; and that the category-mistake is nothing but the complement of a logic of discovery."[15] It is a form of making, of poesis, that grounds all forms of classification.[16] This disordering that is also an ordering, a dismantling that is also a redescription, is also instructive and pleasurable—it offers us "the pleasure of understanding that follows surprise."[17] This movement from surprise to understanding is mirrored in metaphor itself, which is a mode of animation, of change—it makes things visible, alive, and actual by representing things in a state of activity.[18]

Computers, understood as universal machines, stand in for substitution itself. Allegedly making possible the transformation of anything into anything else via the medium of information, they are transference machines. They do not simply change X into Y, they also animate both terms. They create a new dynamic reality: the files they offer us are more alive; the text that appears on their screens invites manipulation, addition, animation. Rather than stable text on paper, computers offer information that is flexible, programmable, transmissible, and ever-changing. Even an image that appears stably on our screen is constantly refreshed and regenerated. Less obviously, computers—software in particular—also concretize Lakoff and Johnson's notion of metaphors as concepts that govern, that form consistent conceptual systems: software is an invisible program that governs, that makes possible certain actions. But if computers are metaphors for metaphors, they also (pleasurably) disorder, they animate the categorical archival system that grounds knowledge.

If theories of metaphor regularly assume that the vehicle (the image expressly used) makes the abstract tenor (the idea represented) concrete—that one makes something unfamiliar familiar through a known concrete vehicle—software as metaphor combines what we only vaguely understand with something equally vague. It is not simply, then, that one part of the metaphor is "hidden," but rather that both parts—tenor and vehicle—are invisibly visible. This does not mean, however, that software as metaphor fails. It is used regularly all the time because it succeeds as a way to describe an ambiguous relation between what is visible and invisible, for invisible laws as driving visible manifestations. Key to understanding the power of software—software as power—is its very ambiguous thingliness, for it grounds

software's attractiveness as a way to map—to understand and conceptualize—how power operates in a world marked by complexity and ambiguity, in a world filled with things we cannot fully understand, even though these things are marked by, and driven by, rules that should be understandable, that are based on understandability. Software is not only necessary for representation; it is also endemic of transformations in modes of "governing" that make governing both more personal and impersonal, that enable both empowerment and surveillance, and indeed make it difficult to distinguish between the two.

4 Always Already There, or Software as Memory

Software—as instructions and information (the difference between the two being erased by and in memory)—not only embodies the always already there, it also grounds it. It enables a logic of "permanence" that conflates memory with storage, the ephemeral with the enduring. Through a process of constant regeneration, of constant "reading," it creates an enduring ephemeral that promises to last forever, even as it marches toward obsolescence/stasis. The paradox: what does not change does not endure, yet change—progress (endless upgrades)—ensures that what endures will fade. Another paradox: digital media's memory operates by annihilating memory.

Remarkably, digital media has been heralded as "saving" analog media from destruction and obscurity. Many users, blind to the limitations of electromagnetic materials, assume that one can actually "store" things in memory. They assume that data saved on their DVDs, hard drives, and jump drives will always be there, that disk failure and the loss of memory it threatens are accidents instead of eventualities. Digitization surprisingly emerged as a preservation method in the 1990s by becoming a major form of "reformatting," a procedure designed to save intellectual content threatened by decaying materials—such as acidic wood-pulp paper and silver-nitrate film—by reproducing it.[1] Indeed, the National Endowment for the Humanities' 1988 "Brittle Books Program," which microfilmed millions of books in peril of "slow burn," viewed digitization as the preferred preservation method, even given a computer file's five-year shelf life. This celebration of the digital as archives' salvation stems in part from how digital files address another key archival issue: access. From the Library of Congress's early attempt to digitize its collections, the American Memory Pilot program (1990–1994), to Google's plan to digitize over ten million unique titles through its Book Search Program (announced in 2004), digitization has been trumpeted as a way for libraries finally to fulfill their mission: to accumulate and provide access to human knowledge. Digital archives are allegedly H. G. Wells's "World Brain" and André Malraux's museum without walls, among other dreams, come true.

At the same time, however, computer archives have been targeted as *the source* of archival decay and destruction, their liquidity threatening both the possibility and the

authenticity of cultural memory. Digital media disrupt the archive because they themselves are difficult to archive or have not been properly archived or both. The 1999 Modern Languages Association (MLA) report, "Preserving Research Collections: A Collaboration between Librarians and Scholars," summarizes the dual challenges of the hard and the soft: "Imagine a historian opening a late nineteenth-century text and helplessly watching as the title page breaks in her hand. Imagine another scholar, ten years from now, inserting a disk containing an important document into the computer and reading only a "fatal error" message on his screen. These two examples illustrate the Janus-like preservation challenge faced by research libraries today: fragility of the print past and the volatility of the future."[2] The material limits of materials not only cause the future to be volatile, but also, again, so do the ever-updating, ever-proliferating, and increasingly incompatible soft and hard technologies—the challenges to the historical preservation of software outlined in the introduction to this book. Moreover, digital imaging potentially destabilizes authenticity. If libraries and archives, as Abby Smith has argued, "serve not only to safeguard that information [which has long-term value], but also to provide evidence of one type or another of the work's provenance, which goes to establishing the authenticity of that work," this function is seriously undermined by electronic images and documents, which are easily changed or falsified.[3] The sheer plethora of digital files also calls into question the importance of the libraries' and archives' traditional gatekeeping function. This is most clear in the Internet Wayback Machine (IWM)'s approach to selection: this site creates a "library of the Internet" by backing up all accessible sites. If libraries and archives traditionally distinguished between materials of enduring value and "other bits of recorded information, like laundry lists and tax returns," which were allowed to vanish, the IWM has solved the extremely time-consuming task of selecting the enduring from the ephemeral by saving everything. (Although it originally tried to save only "significant" material, it soon became an automatic archive of everything.) In addition to all these difficulties, attempts to digitize content have been frustrated by copyright issues, with rights holders demanding compensation or refusing permission. Digital copies—allegedly defined by their immateriality—are, as the introduction has emphasized, more closely regulated than their material counterparts, especially since their use can be controlled by private contracts rather than by copyright or patents.

As this discussion makes clear, digital media's promise is also its threat; the two cannot be neatly divided into the good and the bad. Digital media, if it "saves" anything, does so by transforming storage into memory, by making what decays slowly decay more quickly, by proliferating what it reads. By animating the inanimate—crossing the boundary between the live and the dead—digital media poses new challenges and opportunities for "the archive."

Taking up the intertwining of the biological and the technological addressed previously, this chapter investigates how something as admittedly "soft" (and vapory) as

software hardened into something that allegedly guarantees heredity, and permanence. Looking in particular at von Neumann's early formulation of stored-memory computer architecture, chapter 4 argues that memory became conflated with storage through analogies to analogies: through analogies to cybernetic neurons, to genetic programs, to what would become "analog" media itself. Through these analogies (and their erasure), the new and the different have been reduced to the familiar. I uncover these differences and analogies not to attribute blame, but rather to reveal the dreams and hopes driving these misreadings: the desire to expunge volatility, obliterate ephemerality, and neutralize time itself, so that our computers can become synonymous with archives.[4] These desires are key to stabilizing hardware so that it can contain, regenerate, and thus reproduce what it "stores." Further, they are central to the twin emergence of neoliberalism and computer programs as strategic games.

These analogies also ground one of the fundamental axioms of digital media, namely that the digital reduces the analog—the real world—to 1s and 0s. By doing so the digital allegedly releases and circulates information that before clung stubbornly to material substances, effectively erasing the importance of context and embodiment. The fact that this has become an axiom should make us pause, especially since the evidence against it is substantial: the digital has proliferated, not erased, media types; what has become the analog is not the opposite, but rather the "ground" of the digital; and last, information is not naturally or inherently binary. Rather than making everything universally equivalent, the digital has exploded differences among media formats. Proprietary and nonproprietary electronic file formats such as jpeg, gif, mp3, QuickTime, doc, txt, rtf, and so on, not only distinguish between image, sound, and text, but also introduce ever more numerous differences among them. This explosion is not accidental to the digital, but rather, as I argue later, central to it. Also, the term *analog*, based on the word *analogy*, does not simply refer to what is real. After the emergence of electronic, arithmetically based computers, the term *analog* was adopted to describe computers that solved problems using similar physical models, rather than numerical methods. And finally, information is not simply digital, for information stems from the transmission of continuous electronic signals. The information traveling through computers is not 1s and 0s; beneath binary digits and logic lies a messy, noisy world of signals and interference. Information—if it exists—is always embodied, whether in a machine or an animal. To make information appear disembodied requires a lot of work, work that is glossed over if we just accept the digital as operating through 1s and 0s.

Revising the working thesis of chapters 2 and 3—software as axiomatic—chapter 4 contends that the digital is axiomatic. The digital emerges as a clean, precise logic through an analogy to an analogy, which posits the analog as real/continuous. Looking at the differences between analog and digital computers, this chapter reveals how discrete logical devices work by restricting possibilities and possible decodings.

It also examines how the development of these devices drives the need for "memory," a regenerating and degenerating archive that paradoxically, as Geoffrey C. Bowker notes, annihilates memory by substituting generalized patterns for particular memories.[5] This does not simply erase human agency, however, but rather fosters new dreams of human intervention, action, and incantation. It does not absolve us of responsibility, but instead calls on us to respond constantly, to save actively, if we are to save at all.

Biological Abstractions

John von Neumann's mythic, controversial, and incomplete 1945 "First Draft of a Report on the EDVAC" introduced the concept of stored program computing and memory to the U.S. military and the academic "public." This report is remarkably abstract: rather than describing actually existing components, such as vacuum tubes and mercury delay lines, it offers "hypothetical elements." According to von Neumann, it does so because, although dealing with real elements such as vacuum tubes would be ideal, such specificity would derail the process by introducing specific radio engineering questions at too early a stage. Thinking concretely in terms of types and sizes of vacuum tubes and other circuit elements "would produce an involved and opaque situation in which the preliminary orientation which we are now attempting would be hardly possible." To avoid this, von Neumann bases his consideration "on a hypothetical element, which functions essentially like a vacuum tube—e.g., like a triode with an appropriate associated RLC-circuit—but which can be discussed as an isolated entity, without going into detailed radio frequency electromagnetic considerations."[6] The vagaries of the machinery (vacuum tubes etc.), which are not necessarily digital but can be made to act digitally, threaten the clean schematic logic needed to design this clean, logical machine. Von Neumann describes this deferral as "only temporary."[7] However, J. Presper Eckert and John Mauchly, the original patent holders of stored program computing, would allege that von Neumann did not touch on the "true electromagnetic nature" of the devices because it was outside his purview: von Neumann, they contended, merely translated their concrete ideas into formal logic.[8] In fact, rather than a temporary omission, abstractness was von Neumann's modus operandi, central to the "axiomatic" (blackboxing) method of his general theory of natural and artificial automata and consonant with his game theory work.

This fateful abstraction, this erasure of the vicissitudes of electricity and magnetism, surprisingly depends on an analogy to the human nervous system. As cited earlier, von Neumann specifies the major components of the EDVAC as corresponding to different neurons: "The three specific parts CA [central arithmetic], CC [central control] (together C) and M [memory] correspond to the associative neurons in the human

nervous system. It remains to discuss the equivalents of the *sensory* or *afferent* and the *motor* or *efferent* neurons. These are the *input* and the *output* organs of the device."[9] These neurons, however, are not simply borrowed from the human nervous system. They are the controversial, hypothetical neurons postulated by Warren McCulloch and Walter Pitts in their "A Logical Calculus of Ideas Immanent in Nervous Activity," a text McCulloch claims von Neumann saved from obscurity.[10] (Von Neumann would later describe these neurons as "extremely amputated, simplified, idealized.")[11] In accordance with McCulloch and Pitts, von Neumann expunges the messy materiality of these "neurons":

> Following W. S. McCulloch and W. Pitts . . . we ignore the more complicated aspects of neuron functioning: thresholds, temporal summation, relative inhibition, changes of the threshold by after-effects of stimulation beyond the synaptic delay, etc. It is, however, convenient to consider occasionally neurons with fixed thresholds 2 and 3, that is, neurons which can be excited only by (simultaneous) stimuli on 2 or 3 excitatory synapses (and none on an inhibitory synapse). . . . It is easily seen that these simplified neuron functions can be imitated by telegraph relays or by vacuum tubes. Although the nervous system is presumably asynchronous (for the synaptic delays), precise synaptic delays can be obtained by using synchronous setups.[12]

This analogy thus depends on and enables a reduction of both technological and biological components to blackboxes. In this simplified analogy, the effects of time are ignored to the extent that the synchronous can substitute for the asynchronous and interactions or "after effects" are erased.

So: to what extent are these abstractions and analogies necessary? What did and do they make possible? Clearly, this blackboxing, by divorcing symbolic analysis from material embodiment, has fostered a belief in information as immaterial, but more is at stake in this move to "biology." Notably, Claude Shannon's influential 1936 masters thesis, which showed that relay and switching can be symbolically analyzed (and designed) using Boolean logic, did not rely on an analogy between relays and neurons.[13] In *A Symbolic Analysis of Relay and Switching Circuits*, Shannon develops a means for simplifying and systematizing the development of complex electrical systems. He argues, "Any circuit is represented by a set of equations, the terms of the equations corresponding to the various relays and switches in the circuit." He then goes on to develop a calculus "for manipulating these equations by simple mathematical processes, most of which are similar to ordinary algebraic algorisms."[14] Shannon neither turns to biology nor elaborates on the material details of switches to ground his symbolic analysis. So why should the formal schematic of an automatic stored-memory computer be biologically inflected? And, why does a logical calculus—Boolean, digital logic—necessitate the erasure of the actual functioning of elements, such as vacuum tubes? To respond to these questions, I begin with another: How exactly are analog and digital related in electronic computing?

Nothing but Analog, All the Way Down

According to von Neumann in his 1948 "General and Logical Theory of Automata," a text that intriguingly reverses his initial analogy between vacuum tubes and neurons, the difference between "analogy and digital machines" lies in the ways they produce errors. Analogy machines, von Neumann contends, treat numbers as physical quantities. In order to perform a calculation, they thus find "various natural processes which act on these quantities in the desired way," such as wheel and disk integrators (which lie at the heart of the first computer mice). According to von Neumann, the guiding principle of analogy machines is the classic signal/information-to-noise ratio, a concept Shannon addresses in his *Mathematical Theory of Information*. That is, "the critical question with every analogy procedure is this: How large are the uncontrollable fluctuations of the mechanism that constitute the 'noise,' compared to the significant 'signals' that express the numbers on which the machine operates?"[15] If the calculation to be performed is complex and multistepped, such as the solving of partial differential equations, noise is amplified at every juncture, making it difficult to separate error from answer. Digital machines, in contrast, treat numbers as "aggregates of digits," rather than as physical quantities or signals. Because of this, they are not subject to noise constraints and offer the possibility of absolute precision, although von Neumann points out that round-off errors (now largely addressed by floating-point arithmetic) limit a digital machine's accuracy. Regardless, "the real importance of the digital procedure lies in its ability to reduce the computational noise level to an extent which is completely unobtainable by any other (analogy) procedure."[16]

Crucially, this reduction in noise occurs by ignoring the "analogy" aspect of digital components, for almost every element is a mixture of analogy and digit, as von Neumann acknowledges in "General and Logical Theory of Automata." In opposition to his "First Draft," this later article treats "living organisms as if they were purely digital automata." Responding to objections to this treatment, such as the fact that neurons do not simply work in an all-or-none fashion, he contends:

> In spite of the truth of these observations, it should be remembered that they may represent an improperly rigid critique of the concept of an all-or-none organ. The electromechanical relay, or the vacuum tube when properly used, are undoubtedly all-or-none organs. Indeed, they are the prototypes of such organs. Yet both of them are in reality complicated analogy mechanisms, which upon appropriately adjusted stimulation respond continuously, linearly or non-linearly, and exhibit the phenomena of "breakdown" or "all-or-none" response only under very particular conditions of operation.[17]

The digit, in other words, often treats a quantity as a discrete number, its accuracy resulting from a cut in a signal. The circularity of this passage, in which vacuum tubes are declared prototypes for all-or-none machines, is remarkable. Based on an analogy to computing elements, neurons, which themselves grounded computing elements as

digital, are declared digital: an initial analogy is reversed and turned into ontology. At the base of this logic lies a redefinition of analogy itself as a complicated mechanism that operates on continuous quantities, rather than on discrete units.

This redefinition of analog as continuous, still present with us today whenever we refer to film and other media as "analog media," reveals a fundamental ambiguity at the core of what would become known as analog machines: does the analogy take place at the level of the machine architecture or at the level of signal? Analog as model emphasizes analogous differential equations and thus nonobvious analogous effects; analog as continuous buries these likenesses and privileges data over process. According to Thomas D. Truitt and A. E. Rogers in their 1960 *Basics of Analog Computers*:

> The word "analog" (or "analogue") has been used and misused. It has one meaning to some people, and a variety of uses to others. Webster speaks of a thing which maintains "a relation of likeness with another, consisting in the resemblance not of the things themselves, but of two or more attributes, or effects. . . . It is important to recognize that while *analog computer* refers most commonly to this one specific type of analog computer [general purpose d-c electronic analog computer], it can just as well refer to certain mechanical and hydraulic devices, to general purpose a-c electronic computers, and to a variety of special purpose computers. All of these have one characteristic in common—that the components of each computer or device are assembled to permit the computer to perform as a model, or in a manner analogous to some other physical system.[18]

Truitt and Rogers contend that similarities in system behavior, rather than resemblances between individual components, are key. In this sense, analog machines are simulation machines par excellence. Analog computers are based on similar physical relationships between mechanical and electronic systems and emphasize quantities over numbers. That is, the "signal" operated on and the result measured is a physical quantity, such as the intensity of an electrical current, or the rotation of a disk. Importantly, the notion of these machines as "analogy" machines only became apparent after the introduction of what would become digital computers, simulacra par excellence.

Analog to What?

Analog elements, even as they "ground" digital ones such as transistors and neurons, are not simple predecessors to digital computers. Analog and digital machines both thrived in the 1940s through the 1960s. Analog computers were used regularly in nuclear reactors for real-time data processing, as part of real-time control systems, such as flight simulators, and to simulate guided missiles in 3D (they were used to build the intercontinental ballistic missiles, which made the SAGE (Semi-Automatic Ground Environment) air defense system obsolete by the time it was completed).[19] So-called analog computers were popular because of their speed: they could solve

problems in parallel, rather than serially (one step at a time), and although digital machines could complete one operation (such as subtraction) much more quickly than analog machines, they were not necessarily faster at complex operations. Early analog machines, as argued earlier, also offered a real-time graphical display that allowed engineers to see immediately how changing a coefficient or variable would alter a problem. Last, the fact that analog computers offered fewer decimal points in their solutions than their digital counterparts was often not important, since the accuracy of the calculation was frequently limited by other factors (measuring input, inadequate equations, etc.) and since early digital computers had significant digit control problems.

Not only were analog computers not viewed or accepted as stepping-stones toward digital ones, but also the division itself between analog and digital electronic computers was not clear. Electronic differential analyzers such as MADDIDA (Magnetic Drum Digital Differential Analyzer), which operated using Boolean algebra and digital electronic circuits, yet treated the signals to be operated as quantities rather than numerical entities, muddied the boundary between analog and digital machines —a boundary that arguably did not then exist. Indeed, analyzers only became analog computers rather than "mechanical mathematical" machines after electronics had displaced electromechanics in the production of discrete and nondiscrete machines.[20]

Electronics arguably marked a "break" between newer and older calculating machines in the 1940s as significant as the difference between digital and "analogy." In the May 1946 press release announcing the ENIAC (Electronic Numerical Integrator and Computer—the first working electronic digital computer), the U.S. War Department introduced it as the first "all-electronic general purpose computer," and underscored its "electronic methods."[21] Electronics marked the ENIAC's difference from both the mechanical "analog" differential analyzer and the "digital" (yet electromechanical) Harvard Automatic Sequence Calculator (Mark 1). In Vannevar Bush's 1945 Franklin Institute article introducing the electromechanical Rockefeller Differential Analyzer (RDA, built in 1942)[22] and in the press releases circulated that year, the RDA is never described by its makers/promoters as an analog machine, but rather as a "machine approach" to mathematics,[23] a "computing machine which marks a significant advance in the field of mechanized mathematics"[24] or, more colloquially, as an "electromechanical giant,"[25] a "tireless ally of science."[26] In response to these publications and to the War Department's announcing the ENIAC, newspapers reported on the machines together, calling them both "Magic Brains"[27] and "Mathematical Robots."[28]

Electronic devices were an important breakthrough because of their speed, and because they were built using nonspecialized labor. Mechanical differential analyzers required trained operators to be present at all times and inadvertently "taught" calculus to its "uneducated" operators. Bush claimed that the integraph (an early

electronic version of the differential analyzer) enabled operators/students to cope with difficult mathematical questions by providing "the man who studies it a grasp of the innate meaning of the differential equation." For such a man, "one part at least of formal mathematics will become a live thing."[29] Seeing wheel and disk integrators in action makes calculus "live," moving it from formal writing to actual experience. According to Larry Owens, differential analyzers offered engineering students a graphic way to "think straight in the midst of complexity"—a type of thinking indebted to an engineering "graphical idiom," which operated as a universal language.

At the core of early analog analyzers lie ordinary differential equations. Similar ordinary differential questions describe seemingly disparate and unrelated electrical, electromechanical, mechanical, and chemical phenomena, all of which can be understood as closed "circuits." Analog machines, in this sense, work because ordinary differential equations are universal at a large scale, and because Newton's laws describing force can also describe electrical charge and water capacity.[30] For instance, the mechanical spring circuit represented in figure 4.1 corresponds to the RLC circuit in figure 4.2:

The mechanical spring system corresponds to the following formula:

$m(\mathrm{d}^2x/\mathrm{d}t^2)$ = F [force] – kx [oscillating force of spring] – $D(\mathrm{d}x/\mathrm{d}t)$ [dissipative force of friction]

The electrical system of figure 4.2 has the following analogous differential equation (see table 4.1 for the corresponding quantities):

$L(\mathrm{d}^2q/\mathrm{d}t^2)$ = V [voltage] – $1/Cq$ [oscillating capacitor charge] – $R(\mathrm{d}q/\mathrm{d}t)$ [charge lost over resistor]

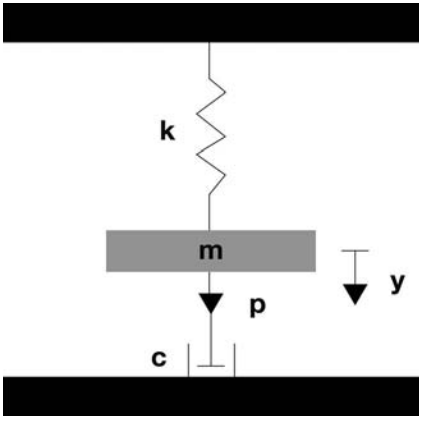

Figure 4.1
Mechanical spring circuit

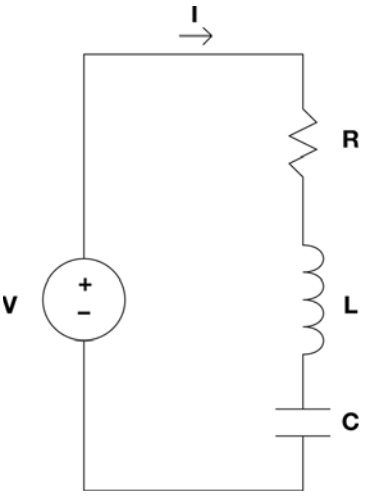

Figure 4.2
RLC circuit

Table 4.1
Analogous entities in the two systems

Mechanical	Electrical
force F	voltage V
mass m	inductance L
friction coefficient D	resistance R
displacement x	charge q
velocity dx/dt	current I
spring coefficient k	reciprocal of capacity $1/C$

All these equations could be put in the form

$$D^{n-1}y/dx^{n-1} = \int d^n y/dx^n dx.$$

For the mechanical spring system, this would be

$$dx/dt \text{ [velocity]} = (1/m) \int (F - kx - D(dx/dt))dt.$$

These equations are not usually solvable using normal analytic methods, but can be solved using numerical methods (desk calculators generally produced tables of solutions to differential equations before the popularization of machinic computers). MIT's differential analyzers employed a wheel and disc integrator to solve these differential

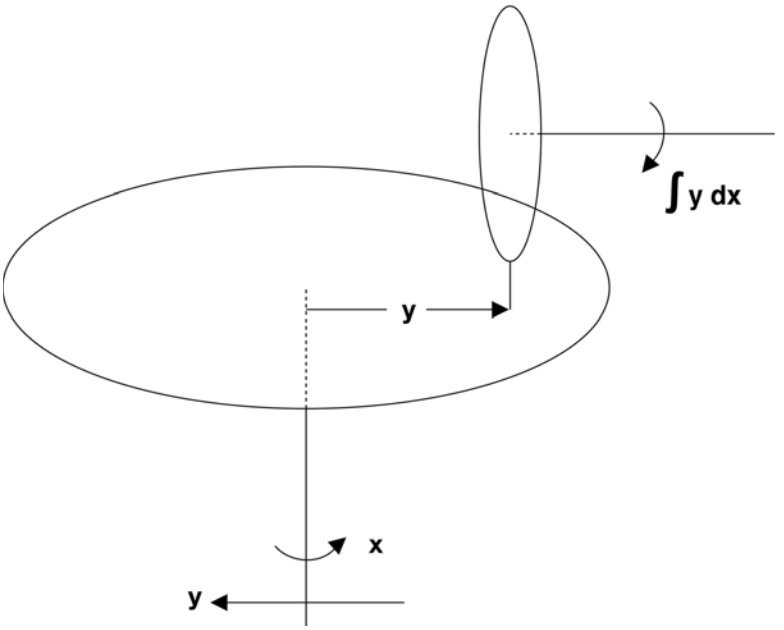

Figure 4.3
Schematic of a basic wheel and disk integrator

equations mechanically, using feedback to solve for values, which appeared on both sides of the equation sign. Figure 4.3 gives the basic design and principle of the integrator.

As figure 4.4 makes clear, the distance y is not a static value, but rather a function given determined by the rotation of another shaft.

So that

$W = k\int_{v1}^{v} U\mathrm{d}V.$

To schematically represent the various operations, Bush used the following symbols (see figure 4.5):

So, using the equation $\mathrm{d}^2y/\mathrm{d}x^2 = \mathrm{f}(x)$, in which case $\mathrm{f}(x)$ is known in order to solve for y, one would build the setup outlined in figure 4.6.

Crucially, the differential analyzer employed "generative" functions—that is, the output could feed into itself. It could thus solve for variables on both sides of the equation. For instance, consider the solution for $\mathrm{d}^2y/\mathrm{d}x^2 = \mathrm{f}(y)$, which is shown in figure 4.7.

These generative functions mark a fundamental difference between digital machines, which solve problems step by step, and analog machines.

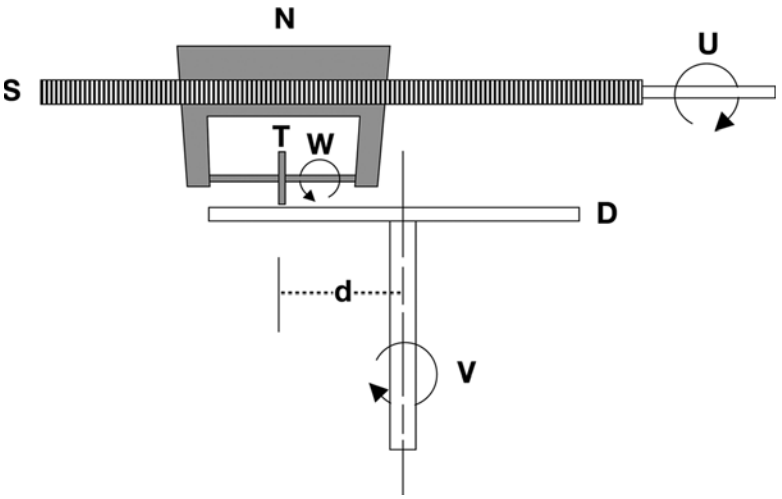

Figure 4.4
Integrator geometry

Because of this mechanical yet "live," analogous relationship, analog machines have generally been conceptualized as more transparent and intuitive than digital ones. Samuel Caldwell, director of MIT's Center of Analysis, stated, "There is a vividness and directness of meaning of the electrical and mechanical processes involved . . . [whereas] a Digital Electronic computer is bound to be a somewhat abstract affair, in which the actual computational processes are fairly deeply submerged."[31] Historian Paul Nyce has argued this mechanical mirroring made the move from analogy to essence or ontology difficult: one always dealt with—made visible—two analogous situations, rather than a universal solution. Nyce contends that analog devices

> belong to a long tradition of scientific instruments, starting in the seventeenth century, that "made visible what could not be seen" . . . Unlike most scientific instruments, however, analog devices supported both understanding (literally by measurement and number, like an astrolabe) and investigation for they, like an orrery, were "models" of phenomena. . . . What also made them persuasive is that they were both statements about and direct imitations of the things they represented. Mimesis is "hidden" or absent in digital machines: analog machines represent phenomena vividly and directly."[32]

Intriguingly, direct representation—or more accurately correspondence—makes analog machines live, vivid, and direct. It is a representation that always is tethered to another "source," which it does not try to hide. The differential analyzer was not, as the digital computer would be, amenable to notions of "universal" disembodied information. The differential analyzer simulated other phenomena, whereas digital computers, by

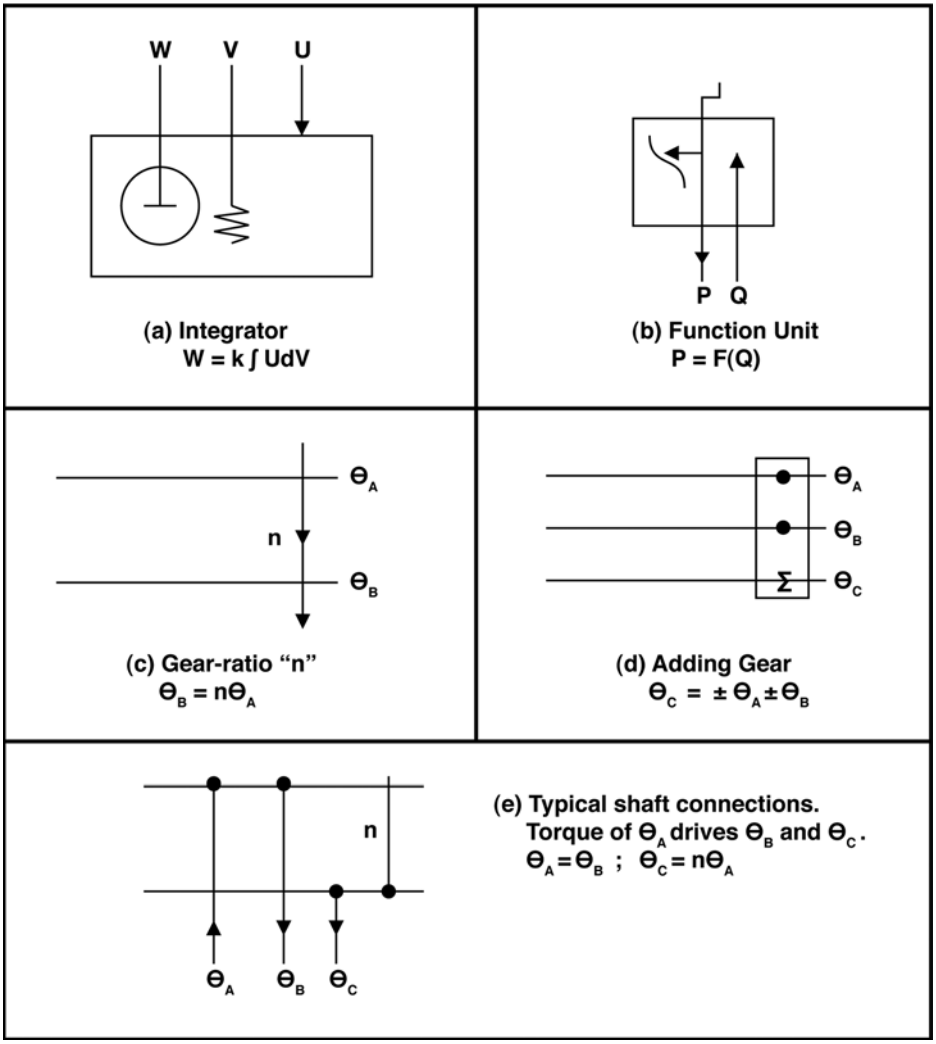

Figure 4.5
Symbols used in connection diagrams for a differential analyzer

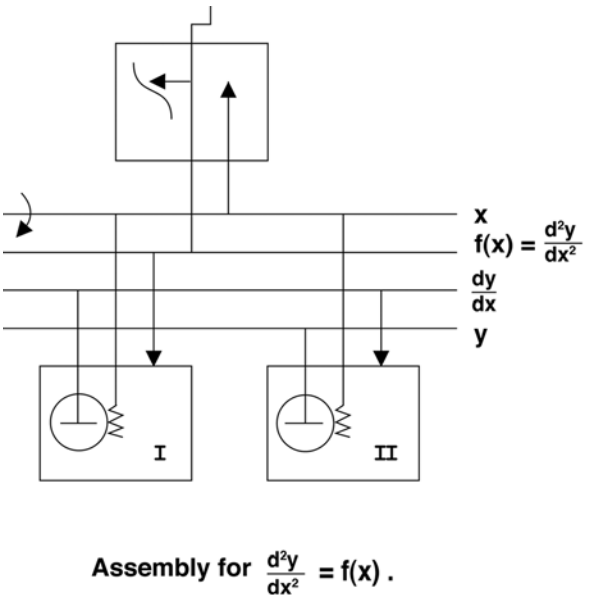

Figure 4.6
$d^2y/dx^2 = f(x)$

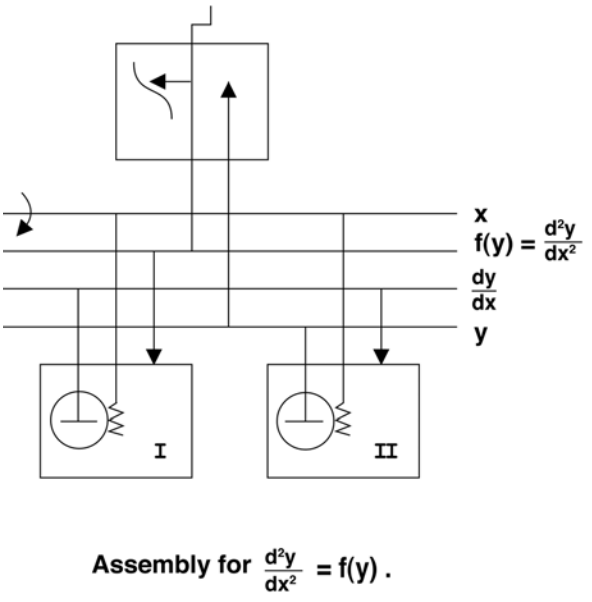

Figure 4.7
$d^2y/dx^2 = f(y)$

hiding mimesis, could simulate any other machine. That is, while both digital and analog computers depend on analogy, digital computers, through their analogy to the human nervous system (which we will see stemmed from a prior analogy between neurons and Turing machines), simulate other computing machines using numerical methods, rather than recreating specific mechanical/physical situations. They move us from "artificial representation" or mechanical analysis (description) to simulacra or "information" (prescription). They move us from solving a problem by defining its parameters to solving it by laying out a procedure to be followed step by step. Depending on one's perspective, analog computers either offer a more direct, "intuitive," and, according to Vannevar Bush, "soul-satisfying" way of solving differential equations or they are imprecise and noisy devices, which add extra steps—the translation of real numbers into physical entities.[33] The first, the engineer's perspective, views computers as models and differential equations as approximations of real physical processes; the second, the mathematician's perspective, treats equations as predictors, rather than descriptors of physical systems—the computer becomes a simulacrum, rather than a simulation.

To be clear, though, analog machines did not simply operate via analogy; again, the notion that they operated through analogy would only be apparent later. They dealt with "signals," from which the notion and the theory of information would emerge, and further Vannevar Bush, as an electrical engineer, considered electricity to be a universal principle.[34] As well, to return to the question of electronics, all analog machines are not large, "intuitively understood," "live" mechanical devices. The electronic machines of the 1950s and 60s differed significantly from their mechanical predecessors. We thus need to be careful not to base arguments about analog machines as a whole on Vannevar Bush's early machines.[35] Indeed, Bush and Caldwell argued that one benefit of the electromechanical RDA was the fact that a trained operator was not necessary. As they explained, the user no longer had to "keep up "with the machine.[36] Op-amps as integrators, or even multipliers, were not "seeable" and graspable in the same fashion as wheel and disks.[37] Last, analog computational structures do not have to coincide perfectly with the problem to be solved: one can reuse an integrator in the same way that one can reuse an adder.

The move to electronics not only deskilled operators, it also made computers mass producible. The mechanical differential analyzers were steeped in the "odor" and the specialized labor of the machine lab, and they used special cams hand-crafted by highly skilled mechanics (the University of Pennsylvania Moore School Differential Analyzer was a WPA [Works Progress Administration] project, designed to employ mechanics). B. Holbrook, who worked at Bell Labs, argued that wire-wound potentiometers "offered the possibility of getting a completely new and relatively trainable type of labor into the manufacture of these things instead of the very high precision

mechanics that were necessary by using the prior method."[38] Electronic analog and digital computers used mass-produced vacuum tubes and later transistors. Thus, both electronic analog and digital "machines" participate in Fordist logic: they automate calculation and production and make invisible the mathematics or calculations on which they rely.

Digital machines, however, are more profoundly Fordist than analog ones. The War Department ENIAC press release states that the ENIAC will eliminate expensive design processes: "Many electrical manufacturing firms, for instance, spend many thousands of dollars yearly in building 'analogy' circuits when designing equipment."[39] Most significant, they are more Fordist because their programming breaks down problems into simple, repeatable discrete steps. It is in programming, or to be more precise, programming in opposition to coding, that analog and digital machines most differ. Douglass Hartree, in his 1949 *Calculating Instruments and Machines*, reserves the terms *programming* and *coding* for digital machines, even though the RDA used tapes to specify the required interconnections between the various units, the values of ratios for the gearboxes, and the initial displacements of the integrators.[40] These tapes, unlike ones used for digital electronic computers, did not contain instructions necessary for sequencing a calculation; like von Neumann, Hartree describes programming as the "drawing up [of a] schedule of [the] sequence of individual operations required to carry out the calculation," and coding as the "process of translating operations into instructions in the particular form in which they are read by the machine."[41] Digital and analog electronic programming both retained the iconographic language of the differential analyzers, and in this sense were both grounded in mechanical methods or in their simulation. However, whereas digital flow charts produce a sequence of individual operations, analog programming produces a "circuit" diagram of systematic relations (see figure 4.8). These differences in programming also point toward key internal differences in representation, namely numbers versus quantities. Coded digital machines are much easier to follow. At a certain level then, analog machines (especially mechanical ones) were not simply more visual or transparent, but rather more complicated.

This complexity made it unlikely that analog computers could spawn or support code as logos. Code as logos—code as the machine—is intimately linked to digital design, which enables a strict step-by-step procedure that neatly translates time into space. Although later it would threaten to reduce all hardware to memory devices in the minds of most of its users, code as logos depended on a certain "hard" digital logic. This logic turns neurons and vacuum tubes themselves into logos and produces an insatiable need for memory, understood as regenerative circuits. This logic again stems from "biology," or, rather, from technologically enhanced biology: cybernetics.

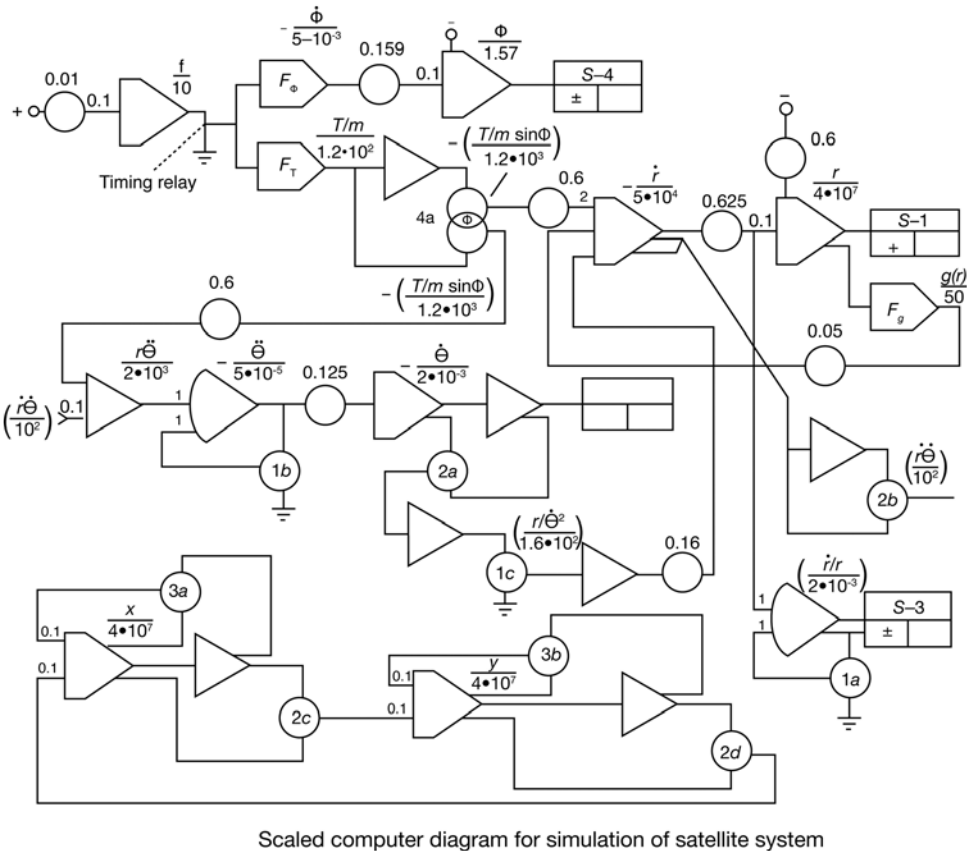

Figure 4.8
An analog program diagram, based on an image from Albert S. Jackson, *Analog Computation* (New York: McGraw-Hill Book Company, 1960), 266

In the Beginning Was Logos (Again)

In "A Logical Calculus of the Ideas Immanent in Nervous Activity," McCulloch and Pitts seek to explain the operation of the brain in logical terms. This paper is part of McCulloch's larger project of "experimental epistemology," his effort to explain "how we know what we know . . . in terms of the physics and chemistry, the anatomy and physiology, of the biological system."[42] This experimental epistemology did not shun theory, but rather sought to weave together philosophy and neurophysiology. At its heart lies the equation of "the 'all-or-none' character of nervous activity" with propositional logic. It reduces a neuronal action to a statement capable of being true or false, "to a proposition which proposed its adequate stimulus."[43] This equation once more

conflates word with action: in this particular case, the firing of a neuron with the proposition that "made" it fire. (Not surprisingly, McCulloch describes his examination of the human mind as a "quest of the Logos.")[44] This equation also concretizes the mind and ideas: "With the determination of the net," McCulloch and Pitts write, "the unknowable object of knowledge, the 'thing itself,' ceases to be unknowable."[45]

As the quotations around "all-or-none" imply, this description is a simplification, one coupled with assumptions such as: "a certain fixed number of synapses must be excited within a period of latent addition in order to excite a neuron at any time, and this number is independent of previous activity and position on the neuron."[46] Despite this, they argue that the all-or-none behavior of neurons makes them the fundamental psychic units or "psychons," which can be compounded "to produce the equivalents of more complicated propositions" in a causal manner.[47] Indeed, the goals of McCulloch and Pitts's logical calculus are to calculate the behavior of any neural net and to find a neural net that will behave in a specified way.[48] Remarkably, their method to "know the unknowable" not only simplifies nervous activity, it also does not engage the actual means by which inhibition or excitation occurs. This is because their method considers circuits equivalent if their result—their perceived behavior—is the same (as I explain later, this was crucial to cybernetic memory). Further, they erase actual alterations that occur during facilitation and extinction (antecedent activity temporarily alters responsiveness to subsequent stimulations of same part of the net) and learning (activities concurrent at some previous time alters the net permanently) via fictitious nets composed of ideal neurons whose connections and thresholds are unaltered.[49] Even though they state that formal equivalence does not equal factual explanation, they also insist that the differences between actual and idealized action do not affect the conclusions that follow from their formal treatment, namely the discovery/generation of a logical calculus of neurons.

Importantly, this logic of equivalence between neural nets and propositional logic was grounded, for McCulloch, in the nature of numbers themselves. In "What Is a Number, that Man May Know It, and a Man, that He May Know a Number?," he draws from David Hume to argue that only numbers truly can be equal. McCulloch's definition of numbers is Bertrand Russell's, "a number is the class of all those classes that can be put into one-to-one correspondence to it."[50] McCulloch's logical calculus, in other words, could only be digital with 1s and 0s corresponding to true and false. McCulloch later made this explicit, in his 1951 "Why the Mind Is in the Head," distinguishing the nervous system from sense organs in terms of digital versus analog. "In so-called logical, or digital contrivances," he writes, "a number to be represented is replaced by a number of things—as we may tally grain in a barn by dropping a pebble in a jug for each sheaf . . . the nervous system is par excellence a logical machine."[51] To McCulloch, logical equals digital because they both rely on numbers. Although analog machines also imply and are based on one-to-one models, McCulloch,

focusing on signals rather than on the machine, claims, "in so-called analogical contrivances a quantity of something, say a voltage or a distance, is replaced by a number of whatnots or conversely, quantity replaces the number. Sense organs and effectors are analogical."[52] In this schema, analog to digital conversion takes place at the level of data—the difference in machine technology is completely erased through a logic of equivalence.

By calling the cortex a digital machine, McCulloch sought to displace the then popular theory of the mind as functioning mimetically. According to Seymour Papert, McCulloch liberated the theory of perception from "the idea that there must be in the brain some sort of genetically faithful representation of the outside world."[53] This is most clearly seen in his 1959 "What the Frog's Eye Tells the Frog's Brain," (an article with J. Y. Lettvin, H. R. Maturana, and W. H. Pitts). In it, they argue that because a frog's eye does not transmit a copy of what it sees but rather detects certain patterns of light and their changes in time, the "eye speaks to the brain in a language already highly organized and interpreted, instead of transmitting some more or less accurate copy."[54] Even earlier, though, and before von Neumann's preliminary draft, the cortex for McCulloch was a Turing machine. In "A Logical Calculus," McCulloch and Pitts state, "Every net, if furnished with a tape, scanners connected to afferents and suitable efferents to perform necessary motor-operations, can compute only such numbers as can a Turing machine."[55] Neural nets are inspired by and aspire to be Turing machines.[56] Von Neumann's use of McCulloch and Pitts's analysis is thus an odd and circular way of linking stored-memory digital computers to computing machines—once more, an over-determined discovery of a linkage between biology and computer technology, yet another turn of the double helix (before, of course, there was a double helix).

This linkage not only establishes a common formal logic, it also enables the emergence of computer "memory." Moving away from ideas of field-based, analogical notions of memory, McCulloch's neural nets produce transitory memories and ideas through circular loops. Drawing from Wiener's definition of information as order (negative entropy), McCulloch argues that ideas are information: they are regularities or invariants that conserve themselves as other things transform.[57] McCulloch contentiously claims that this stability is produced by reverberating "positive-feedback" circuits, that is, transitory memory (reverberatory memory cannot survive a "shut down," such as a deep sleep or narcosis).[58] These reverberatory circuits, though, even as they enable memory, also render "reference indefinite as to time past,"[59] for what is retained is the memory, not all the events that led to that memory. In this sense, they threaten to become "eternal ideas," separated from context. This separation, combined with the fact that the neural nets can specify the next but not the previous state, means that "our knowledge of the world, including ourselves, is incomplete as to space and indefinite as to time."[60] Causality runs only one way: one cannot decisively "reverse engineer" a neural net's prior state.

This emergence of memory is thus, as Bowker notes, also a destruction of memory. Thinking through cybernetician Ross Ashby's claim that "memory is a metaphor needed by a 'handicapped' observer who cannot see a complete system," Bowker writes, "The theme of the destruction of memory is a complex one. It is not that past knowledge is not needed; indeed, it most certainly is in order to make sense of current actions. However, a *conscious* holding of the past in mind was not needed: the actant under consideration—a dog, a person, a computer—had been made sufficiently different that, first, past knowledge was by definition retained and sorted and, second, only useful past knowledge survived."[61] What is truly remarkable is that this destruction of memory has spawned the seemingly insatiable need for computer memory. Memories are rendered into context-free circuits freed from memory, circuits that are necessary to the operation of the animal/machine.

Although the past may not be determinable from the present, memories—as context-free invariant patterns—ground our ability to predict the future. This prediction—causality—according to McCulloch (drawing from Hume) is only a "suspicion"[62] that there is "some law compelling the world to act hereafter as it did of yore."[63] Like those of ideas, these predictive circuits persist. Indeed, McCulloch argues, "the earmark of every predictive circuit is that if it has operated long uniformly it will persist in activity, or overshoot; otherwise it could not project regularities from the known past upon the unknown future."[64] The endurance of these circuits, however, threatens closure, threatens to make the unknown imperceptible, something that McCulloch "as a scientist . . . dread[s] most, for as our memories become stored, we become creatures of our yesterdays—mere has-beens in a changing world. This leaves no room for learning."[65] Memory, then, which enables a certain causality as well as an uncertainty as to time and place, threatens to overwhelm the system, creating networks that crowd out the new. A neural circuit, if it persists—programmability—makes prediction possible. It, however, also puts in jeopardy what for McCulloch is most interesting and vital about humanity: the ability to learn and adapt to the unknown, that is, the future as future.

This notion of memory as circuit/signal underscores McCulloch's difference from cognitive psychology, which, following developments in computer technology, would consider the brain hardware and the mind software.[66] In McCulloch's system, the mind and body are intimately intertwined, with the mind becoming less "ghostly"—more concrete—perhaps paradoxically by becoming signal.[67] Signals bridge mind and brain because they have a double nature; they are both physical events and symbolic values.[68] They are both statement and result. The logic of computers as logos stems from the disciplining, the axiomatizing, of hardware. This in turn "solidifies" instructions into things in and of themselves. Notably, McCulloch in his later work did address software, or programs, but referred to them as instructions to be operated on by data in memory, rather than as stored themselves in memory.[69] Instructions, in

other words, did not drive the system—the logic, the logos, happened at the level of firing neurons.

Thus, by turning to McCulloch and Pitts rather than to Shannon, von Neumann gains a particular type of abstraction or logical calculus: an axiomatic abstraction and schematic design that greatly simplifies the behavior of its base components. Von Neumann also gains a parallel to the human nervous system, key to his later work on "general automata." Last, he "gains" the concept of memory—a concept that he would fundamentally alter by asserting the existence of biological organs not known to exist. Through this hypothetical "memory organ," and his discussion of the relationship between orders and data, his model would profoundly affect the development of cognitive science and artificial intelligence (AI) and life (AL). Through this memory organ, von Neumann would erase the difference between storage and memory, and also open up a different relationship between man and machine, one that would incorporate instructions—as a form of heredity—into the machine, making software fundamental. If word (as description) becomes event in McCulloch and Pitts's theory, in von Neumann's theory event once again becomes word, word becomes instruction.

Memories to Keep in Mind

Von Neumann's work with natural and artificial automata in general reverses the arrow of the analogy established in "First Draft." Rather than explaining computers in terms of the human nervous system, he elucidates the brain and its functioning in terms of computational processes. This is most clear in von Neumann's discussion of memory, which he considered to be a "much more critical and much more open" issue than logical processing.[70] In computer systems, memory was the bottleneck, for the limitations of memory on the machine created an "abnormal economy," in which the computer is forced to store all the information it needs to solve a problem on the equivalent of one page.[71]

The term *memory organ* clearly borrows from biology. This borrowing, however, was not necessary. Prior to "First Draft," mechanisms designed to store numbers and functions necessary for computing were called storage devices or "the store," following Babbage's terminology. J. Presper Eckert's 1944 "Disclosure of Magnetic Calculating Machine," used as evidence in the patent trial, refers concretely to the disks or tapes used to store data; his 1946 patent application, in contrast, employs the term *electrical memory*. This movement from storage to memory lies at the heart of the computer as archive, the computer as saving us from the past, from repetition through repetition.

Computer storage devices as memory is no simple metaphor, since it asserts the existence of an undiscovered biological organ. Although von Neumann initially

viewed memory as comprising afferent neurons, he soon changed his mind, based on his own experience with computers, in particular with the number of vacuum tubes needed to create the types of reverberatory circuits McCulloch and Pitts described. In a reverse move, he postulated human memory as something unknown but logically necessary, making clear that his first analogy was based on a leap of faith. In *The Computer and the Brain*, written ten years after "First Draft," von Neumann writes, "the presence of a memory—or, not improbably, of several memories—within the nervous system is a matter of surmise and postulation, but one that all our experience with artificial automata suggests and confirms." Von Neumann goes on to emphasize our ignorance regarding this memory:

> It is just as well to admit right at the start that all physical assertions about the nature, embodiment, and location of [human memory] are equally hypothetical. We do not know where in the physically viewed nervous system a memory resides; we do not know whether it is a separate organ or a collection of specific parts of other already known organs, etc. It may well be residing in a system of specific nerves, which would then have to be a rather large system. It may well have something to do with the genetic mechanism of the body. We are as ignorant of its nature and position as were the Greeks, who suspected the location of the mind in the diaphragm. The only thing we know is that it must be a rather large-capacity memory, and that it is hard to see how a complicated automaton like the human nervous system could do without one.[72]

This passage reveals how quickly the computer moved from a system modeled on ideal neurons to a concrete model for more complex biological phenomena. This statement, which seems to be so careful and qualified—we basically do not know what the memory is or where it resides—at the same time asserts the existence of a memory organ or set of organs based on an analogy to computers: "The only thing we know is that it must be a rather large-capacity memory, and that it is hard to see how a complicated automaton like the human nervous system could do without one." This guess regarding capacity assumes that the brain functions digitally, that it stores information as bits, which are then processed by the brain, rather than functioning more continuously in a "field-based" manner. Again, this assumption was by no means accepted whole-heartedly by biologists. Dr. Lashley, among others, responded to von Neumann's difficulty with neuronal capacity by arguing that the memory was more dynamic rather than static and that "the memory trace is the capacity of many neurons to work together in certain permutations."[73]

Neurons as switching elements drive von Neumann's "logical" guess regarding memory capacity, as well as his confusion over its location:

> In the human organism, we know that the switching part is composed of nerve cells, and we know a certain amount about their functioning. As to the memory organs, we haven't the faintest idea where or what they are. We know that the memory requirements of the human organism are large, but on the basis of any experience that one has with the subject, it's not likely that the memory sits in the nervous system, and it's really very obscure what sort of thing it is.[74]

Digital switching devices, based on the reduction of all processes to true/false propositions, insatiably demand memoryless memory. As von Neumann explains in "First Draft," the need for memory increases as problems are broken down into long and complicated sequences of operations (described in chapter 1 of this book by Bartik and Holberton). Digital computation needs to store and have access to intermediate values, instructions, specific functions, initial conditions and boundary conditions, etc. Prior to the EDVAC, these were stored in an outside recording medium such as a stack of paper cards. The EDVAC was to increase the speed of calculation by putting some of those values inside the memory organ, making porous the boundaries of the machine. Memory instituted "*a prosthesis of the inside*."[75] Memory was not simply sequestered in the "organ"; it also bled into the central arithmetic unit, which, like every unit in the system, needed to store numbers in order to work.

To contain or localize memory, von Neumann organized it hierarchically: there were to be many memory organs, defined by access time rather than content. For instance, in the 1946 work "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," von Neumann and colleagues divide memory into two conceptual forms—numbers and orders, which can be stored in the same organ if instructions are reduced numbers—and into two types—primary and secondary.[76] The primary memory consists of registers, made of flip-flops or trigger circuits, which need to be accessed quickly and ideally randomly. Primary memory, however, is very expensive and cumbersome. A secondary memory or storage medium supplements the first, holding values needed in blocks for a calculation. Besides being able to store information for periods of time, such a memory needs to be controllable automatically (without the help of a person), easily accessed by the machine, and preferably rewriteable. Interestingly, the devices listed as possible secondary memories are other forms of media: for instance, teletype tapes, magnetic wire or tapes, and movie film. (The primary media was also another medium—the Selectron was a vacuum tube similar to one used for television.)[77] This gives a new resonance to McLuhan's assertion that new media do not make preexisting media obsolete but merely change their use.[78] Von Neumann and colleagues also outlined a third form of memory, "dead storage," which is an extension of secondary memory, since it is not initially integrated with the machine. Not surprisingly, input and output devices eventually become part of "dead storage." As von Neumann argues later in *The Computer and the Brain*, "the very last stage of any memory hierarchy is necessarily the outside world, that is, the outside world as far as the machine is concerned, i.e. that part of it with which the machine can directly communicate, in other words the input and the output organs of the machine."[79] In this last step, the borders of the organism and the machine explode. Rather than memory comprising an image of the world in the mind, memory comprises the whole world itself as it becomes "dead."

This last step renders the world dead by conflating memory—which is traditionally and initially regenerative and degenerative—with other more stable forms of media such as paper storage, a comparison that is still with us today at the level of both memory (files) and interface (pages and documents). This conflation both relied on and extended neurophysiological notions of memory as a trace or inscription, like the grooves of a gramophone record. McCulloch, for instance, in 1951, in response to objections posed by von Neumann over memory as reverberatory circuits, outlined a hierarchical memory system that resonated with von Neumann's schema. There are first temporary reverberations, and second, nervous nets that alter with use (central to conditioned behaviors). The third type of memory, which he sees as an informational bottleneck, however, leaves him unhappily stumped; he is at a loss to describe its location and its operation:

> I don't see how we can tell where we have to look as yet, because in many of the experiments in which there are lesions made in brains, we have had large amounts of territory removed. However, usually we fail to destroy most fixed memories: therefore, we cannot today locate the filing cabinets. I think that sooner or later answers to the question of those filing cabinets, or whatever it is on which is printed "photographic records" and what not, will have to be found.[80]

The term *filing cabinet* is drawn from von Neumann's own terminology. In his response to McCulluch's paper, von Neumann, perhaps informed by psychoanalytical arguments that memories never die (one of von Neumann's uncles introduced psychoanalysis to Hungary and von Neumann apparently loved to analyze jokes) or by his personal experience (he allegedly had a photographic memory and could recall conversations word for word), presents the following "negative" and not entirely "cogent" argument against memory as residing in the neurons:

> There is a good deal of evidence that memory is static, unerasable, resulting from an irreversible change. (This is of course the very opposite of a "reverberating," dynamic, erasable memory.) Isn't there some physical evidence for this? If this is correct, then no memory, once acquired, can be truly forgotten. Once a memory-storage place is occupied, it is occupied forever, the memory capacity that it represents is lost; it will never be possible to store anything else there. What appears as forgetting is then not true forgetting, but merely the removal of that particular memory-storage region from a condition of rapid and easy availability to one of lower availability. It is not like the destruction of a system of files, but rather like the removal of a filing cabinet into the cellar. Indeed, this process in many cases seems to be reversible. Various situations may bring the "filing cabinet" up from the "cellar" and make it rapidly and easily available again.[81]

Von Neumann's "negative argument" relies on files and the human mind as the owner/manipulator—or, to return to Cornelia Vismann's argument outlined in chapter 2, chancellor—of files. It also depicts the human brain as surprisingly nonplastic: easily used up and unerased, hence once more the need for great storage. It also moves away from memory as based on erasable "regenerative" traces toward fantasies of traces

that do not fade: immortality within the mortal machine.[82] This is a far cry from Vannevar Bush's description of the human mind in chapter 2 as fundamentally ephemeral and prone to forgetting. The digital paradoxically produces memory as storage, in part because logical algorithms need to read and write values. An entire process can fail if one variable is erased.

Memory as storage also allows von Neumann to describe genes as a form of human memory. In *The Computer and the Brain*, he writes, "another form of memory, which is obviously present, is the genetic part of the body; the chromosomes and their constituent genes are clearly memory elements which by their state affect, and to a certain extent determine, the functioning of the entire system."[83] With this move toward genes as memory—necessary for his theory of self-reproducing formula—neurons would not stand in for words (true or false propositions), but words (instructions) would come to stand in for neurons.

Descriptions that Can

The deed is everything, the Glory naught.
—*Faust*, Part II

According to William Poundstone, the last anecdote of von Neumann's "total recall" concerns his last days, when he lay dying of cancer at Walter Reed Hospital, a cancer caused by his work on nuclear weapons (the drive for nuclear weapons also powered the development of digital electronic computers; American computers and neoliberalism are both reactions to Nazism).[84] His brother Michael read *Faust* in the original German to von Neumann and, "as Michael would pause to turn the page, von Neumann would rattle off the next few lines from memory."[85] Converting to Catholicism before his death, von Neumann was deeply influenced by the work of Goethe, *Faust* in particular. Said his brother Nicholas, "We studied *Faust* in school very thoroughly, both parts, in original and in Hungarian translation. And we discussed it for years and rereading it occasionally thereafter, throughout our respective lifetimes."[86] One of the three passages Nicholas describes as particularly important to his brother was Faust's grappling with logos: "Faust's monologue at the opening of the First Part: 'In the beginning was the Act,' and the corresponding statement in Part II: 'The deed is everything, the Glory naught.' This we discussed in the context of the redeeming value of action."[87] According to Nicholas, this passage led "ultimately to John's views emphasizing the redeeming value of practical applications in his profession."[88] John von Neumann as an unredeemed (although not yet fallen) Faust.

This passage, however, has other resonances, intersecting with the question of logos weaving through this book. Faust, seeking to translate the Bible into German pauses over "in the beginning was the Word":

I'm stuck already! I must change that; how?

Is then "the word" so great and high a thing?

There is some other rendering,

Which with the spirit's guidance I must find.

We read: "In the beginning was the Mind."

Before you write this first phrase, think again;

Good sense eludes the overhasty pen.

Does "mind" set worlds on their creative course?

It means: "In the beginning was the Force."

So it should be—but as I write this too,

Some instinct warns me that it will not do.

The spirit speaks! I see how it must read,

And boldly write: "In the beginning was the Deed!"[89]

Faust, after a failed encounter with a spirit he conjured but cannot control, replaces Word with Deed, which, rather than Word, Force, or Mind, creates and rules the hour. Ironically, Faust, of course, is later saved by the Word—a technicality regarding his statement of satisfaction. Regardless, this substitution of Word with Deed sums up von Neumann's axiomatic approach to automata and his attraction to McCulloch and Pitts's work. It also leads him to conceive of memory as storage: as a full presence that does not fade, even though it can be misplaced. What is intriguing, again, is that this notion of a full presence stems from a bureaucratic metaphor: filing cabinets in the basement. This reconceptualization of human memory bizarrely offers immortality through "dead" storage: information as undead.

McCulloch and Pitts's methodology again depends on axiomatizing idealized neurons, where, according to von Neumann, "axiomatizing the behavior of the elements means this: We assume that the elements have certain well-defined, outside, functional characteristics; that is, they are to be treated as 'black boxes.' They are viewed as automatisms, the inner structure of which need not be disclosed, but which are assumed to react to certain unambiguously defined stimuli, by certain unambiguously defined responses."[90] This controversial axiomatization, which von Neumann would employ later in his theory of self-reproducing automata, reduces all neuronal activities to true/false statements.[91] Neurons follow a propositional logic. Von Neumann contends that this axiomatizing and the subsequent logical calculus it allows means that McCulloch and Pitts have proven that "any functioning . . . which can be defined at all logically, strictly, and unambiguously in a finite number of words

can also be realized by such a formal neural network. . . . It proves that anything that can be exhaustively and unambiguously described, anything that can be completely and unambiguously put into words, is ipso facto realizable by a suitable finite neural network."[92] Words that describe objects, in other words, can be replaced by mechanisms that act, and all objects and concepts, according to von Neumann, can be placed in this chain of substitution. "There is no doubt," he asserts, "that any special phase of any conceivable form of behavior can be described 'completely and unambiguously' in words. This description may be lengthy, but it is always possible. To deny it would amount to adhering to a form of logical mysticism which is surely far from most of us."[93] This does not mean, however, that such a description is simple; indeed, von Neumann stresses that McCulloch and Pitts's theorizing is important for its reverse meaning: "there is a good deal in formal logics to indicate that the description of the functions of an automaton is simpler than the automaton itself, as long as the automaton is not very complicated, but that when you get to high complications, the actual object is simpler than the literary description."[94]

This notion of an actual object is not outside of language, even if it is outside "literary description," for, to von Neumann, producing an object and describing how to build it were equivalent. For instance, he argues that the best way to describe a visual analogy may be to describe the connections of the visual brain.[95] According to this logic, the instructions to construct a machine can substitute for the machine itself, to the extent that it can produce all the behaviors of the machine.

This logic is most clear in von Neumann's earliest model of self-reproduction, which Arthur Burks later dubbed a "robot" or "kinematic" model.[96] In this model, "constructing automata" *A* are placed in a "reservoir in which all elementary components in large numbers are floating."[97] Automaton *A* "when furnished the description of [an] other automaton in terms of appropriate functions will construct that entity." This description "will be called an instruction and denoted by a letter *I*. . . . All [*A*s] have a place for an instruction *I*."[98] In this system, instruction drives construction. In addition to automata *A*, there are also automata *B*, which can copy any instruction *I* given to them. The decisive step, von Neumann argues, is the following instruction to the reader about embedding instructions:

> Combine the automata *A* and *B* with each other, and with a control mechanism *C* which does the following. Let *A* be furnished with an instruction *I*. . . . Then *C* will first cause *A* to construct the automaton, which is described by this instruction I. Next *C* will cause *B* to copy the instruction *I* referred to above, and insert the copy into the automaton referred to above, which has just been constructed by *A*. Finally, *C* will separate this construction from the system *A* + *B* + *C* and "turn it loose" as an independent entity.[99]

This independent entity is to be called *D*. Von Neumann then argues, "In order to function, the aggregate *D* = *A* + *B* + *C* must be furnished with an instruction *I*, as described above. This instruction, as pointed out above, has to be inserted into *A*. Now form an

instruction I_D, which describes this automaton D, and insert I_D into A within D. Call the aggregate which now results E. E is clearly self-reproductive."[100] This instruction I_D (which nicely resonates with ID and id), he claims, is roughly equivalent to a gene. He also contends that B "performs the fundamental act of reproduction, the duplication of the genetic material, which is clearly the fundamental operation in the multiplication of living cells." This analogy fails, however, because "the natural gene does probably not contain a complete description of the object whose construction its presence stimulates. It probably contains only general pointers, general cues."[101] Thus, the memory of the system—here postulated as a more vibrant form of memory than "paper tape"—becomes the means by which the automaton can self-reproduce.[102]

This description is amazing for several reasons. In it, von Neumann transforms McCulloch and Pitts's schematic neural networks, in which there is no separation of software from hardware, into the basis for code as logos for the instructions replace the machine. What becomes crucial, in other words, and encapsulates the very being of the machine, are the instructions needed to construct it. Furthermore, and inseparable from the translation of event into instruction, this description—as a set of instructions itself—contains a bizarre, almost mystical, address. For, when von Neumann says, "Now form an instruction I_D, which describes this automaton D, and insert I_D into A within D," or "Combine the automata A and B with each other, and with a control mechanism C," who will do this forming and combining; who will perform these crucial steps and how? What mystical force will respond to this call? Like Faust before Mephistopheles arrives, are we to incant spells to create spirits? The transformation of description into instruction leaves open the question: who will do this? Who will create the magical description that goes inside? Remarkably, this call makes clear the fact that humans are indistinguishable from automata, something that bases von Neumann's game theory as well.

Games and Universes

This replacement of descriptions by instructions (or choices among instructions) also grounds von Neumann's work in game theory, which corresponds to his work on automata in many ways, as Arthur Burks has pointed out. "There is a striking parallel," Burks writes, "between von Neumann's proposed automata theory and his theory of games. Economic systems are natural competitive systems; games are artificial competitive systems. The theory of games contains the mathematics common to both kinds of competitive systems, just as automata theory contains the mathematics common to both natural and artificial automata."[103] This comparison, however, not only occurs at the level of mathematics or mathematization, but also at the level of heuristics, descriptions, and strategies. Game theory, which has been a key tool of neoliberal economic theory, seeks to understand the problem of exchange through

the perspective of a "game of strategy," in which participants create strategies in response to others' moves, the rules of the game, and (objective) probabilities.[104] Similar to von Neumann's "First Draft," von Neumann and Oskar Morgenstern's 1944 *Theory of Games and Economic Behavior* (their preliminary discussion of game theory) serves as a *heuristic*, a "phase of transition from unmathematical plausibility considerations to the formal procedure of mathematics."[105] Also like his theory of automata, and indeed like most of von Neumann's mathematical work, game theory is based on an axiomatic method. Most importantly, von Neumann and Morgenstern introduce the notion of *strategy* to replace or simplify detailed description. Describing the process of giving an exact description of what comprises a game, they write, "we reach—in several successive steps—a rather complicated but exhaustive and mathematically precise scheme." Their key move is "to replace the general scheme by a vastly simpler one, which is nevertheless equivalent to it. Besides, the mathematical device which permits this simplification is also of an immediate significance for our problem: It is "the introduction of the exact concept of a strategy."[106] A strategy is a complete plan that "specifies what choices [the player] will make in every possible situation, for every possible actual information which he may possess at that moment in conformity with the pattern of information which the rules of the game provide for him for that case."[107] This replacement of a complete description with a strategy is not analogous to the replacement of machine code with a higher-level programming language, or what von Neumann calls "short code." This "equivalence" is not based on a simplification through the creation of a language that reduces several events into one statement, but rather on a fundamental transformation of a step-by-step description of events into a description of the premises—the rules and related choices—driving the player's actions. This strategy, which game theory remarkably assumes every player possesses before the game, is analogous to a program—a list of instructions to be followed based on various conditions. A player's strategy is not a summary of the rules of the game, but rather a list of choices to be followed—it is, to return to a distinction introduced in chapter 1, a product of "programming" rather than coding. Or, to put it slightly differently, understanding game strategy as a program highlights the fact that a program does not simply establish a universe as Weizenbaum argues; it is one possible strategy devised within an overarching structure of rules (a programming language). A strategy/program thus emphasizes the programming/economic agent as freely choosing between choices.[108]

This program/strategy has been the basis of much of the criticism directed against game theory, such as Gregory Bateson's contention:

> What applications of the theory of games do is to reinforce the player's acceptance of the rules and competitive premises, and therefore make it more and more difficult for the players to conceive that there might be other ways of meeting and dealing with each other. . . . Von Neumann's "players" differ profoundly from people and mammals in that those robots totally

> lack humor and are totally unable to "play" (in the sense in which the word is applied to kittens and puppies).[109]

Bateson is absolutely correct in his assessment: in outlining such a comprehensive version of a strategy, game theory assumes a player who could only be—or later would become—an automaton. Furthermore, von Neumann admits that game theory is prescriptive rather than descriptive. He writes, "the immediate concept of a solution is plausibly a set of rules for each participant which will tell him how to behave in every situation which may conceivably arise."[110] Thus, game theory presumes a strategy and the production of a strategy, as well as the replacement of a detailed description of every action with a more general procedural one. A strategy is something an automaton—or more properly a programmer—working non-"interactively" with a computer has. Game theory's assumptions again resonate with those of neoliberalism (Milton Friedman, to take one example, theorizes the day-to-day activities of people as analogous to those of "the participants in a game when they are playing it").[111]

Words, as instructions that stand in for deeds, are also crucial to von Neumann's desire to make his machines "universal." Von Neumann approaches the concept of universality through an interpretation of Alan Turing's "On Computable Numbers, with an Application to the Entscheidungsproblem," the 1936 paper that initially inspired McCulloch and Pitts.[112] In this paper, Turing shows that Hilbert's *Entscheidungsproblem* (the decision problem) cannot have a solution through theoretical machines, analogous to a "man," that can compute any number. He also posits the existence of a "universal machine," "a single machine which can be used to compute any computable sequence."[113] Von Neumann, in a rather historically dubious move, equates abstract or universal Turing machines with higher-level languages.

To make this argument, von Neumann separates codes into two types: complete and short. In computing machines, complete codes "are sets of orders, given with all necessary specifications. If the machine is to solve a specific problem by calculation, it will have to be controlled by a complete code in this sense. The use of a modern computing machine is based on the user's ability to develop and formulate the necessary complete codes for any given problem that the machine is supposed to solve."[114] Short codes, in contrast, are based on Turing's work, in particular his insight that "it is possible to develop code instruction systems for a computing machine which cause it to behave as if it were another, specified, computing machine."[115] Importantly, Turing himself did not refer to short or complete codes, but rather to instructions and tables to be mechanically—meaning faithfully—followed. Despite this, von Neumann argues that a code following Turing's schema must do the following:

> It must contain, in terms that the machine will understand (and purposively obey), instructions (further detailed parts of the code) that will cause the machine to examine every order it gets and determine whether this order has the structure appropriate to an order of the second

> machine. It must then contain, in terms of the order system of the first machine, sufficient orders to make the machine cause the actions to be taken that the second machine would have taken under the influence of the order in question.
>
> The important result of Turing's is that in this way the first machine can be caused to imitate the behavior of *any* other machine.[116]

Thus, in a remarkably circular route, von Neumann establishes the possibilities of source code as logos: as something iterable and universal. Word becomes action becomes word becomes the alpha and omega of computation.

Enduring Ephemeral

Crucially, memory is not a static but rather an active process. A memory must be held in order to keep it from moving or fading. Again, memory does not equal storage. Although one can conceivably store a memory, *storage* usually refers to something material or substantial, as well as to its physical location: a store is both what is stored and where it is stored. According to the OED, to store is to furnish, to build stock. Storage or stocks always look toward the future. In computer speak, one reverses common language, since one stores something in memory. This odd reversal and the conflation of memory and storage gloss over the impermanence and volatility of computer memory. Without this volatility, however, there would be no memory. To repeat, memory stems from the same Sanskrit root for *martyr*. Memory is an act of commemoration—a process of recollecting or remembering.

This commemoration, of course, entails both the permanent and the ephemeral. Memory is not separate from questions of representation or enduring traces. Memory, especially artificial memory, traditionally has brought together the permanent and the ephemeral; for instance, the wax tablet with erasable letters (the inspiration for classical mnemotechnics). As Frances A. Yates explains, the rhetorician treated architecture as a writing substrate onto which images, correlating to objects to be remembered, were inscribed. Summarizing the *Rhetorica Ad Herennium*, the classic Latin text on rhetoric, she states:

> The artificial memory is established from places and images . . . the stock definition to be forever repeated down the ages. A *locus* is a place easily grasped by the memory, such as a house, an intercolumnar space, a corner, an arch, or the like. Images are forms, marks or simulacra . . . of what we wish to remember. For instance, if we wish to recall the genus of a horse, of a lion, of an eagle, we must place their images on a definite *loci*.
>
> The art of memory is like an inner writing. Those who know the letters of the alphabet can write down what is dictated to them and read out what they have written. Likewise those who have learned mnemonics can set in places what they have heard and deliver it from memory. "For the places are very much like wax tablets or papyrus, the images like the letters, the arrangement and disposition of the images like the script, and the delivery is like the reading."[117]

Visiting these memorized places, one revives the fact to be recalled. This discussion of memory offers a different interpretation of the parallels between human and computer memory. The rhetorician was to recall a physical space within her mind—the image is not simply what is projected upon a physical space, but also the space for projection. Similarly, computer memory (which, too, is organized spatially) is a storage medium *like* but not quite paper. Both degenerate, revealing the limitations of the simile.

Memory as active process is seen quite concretely in early forms of "regenerative memory," from the mercury delay line to the Williams tube, the primary memory mentioned earlier. The serial mercury delay line (figure 4.9) took a series of electrical pulses and used a crystal to transform them into sound waves, which would make their way relatively slowly down the mercury tube. At the far end, the sound waves would be amplified and reshaped.[118] One tube could usually store about a thousand binary bits at any given moment.

Another early memory device, the Williams tube (figure 4.10), derived from developments in cathode ray tubes (CRTs); the television set is not just a computer screen, but was also once its memory. The Williams tube takes advantage of the fact that a beam of electrons hitting the phosphor surface of a CRT not only produces a spot of light, but also a charge. This charge will persist for about 0.2 seconds before it leaks

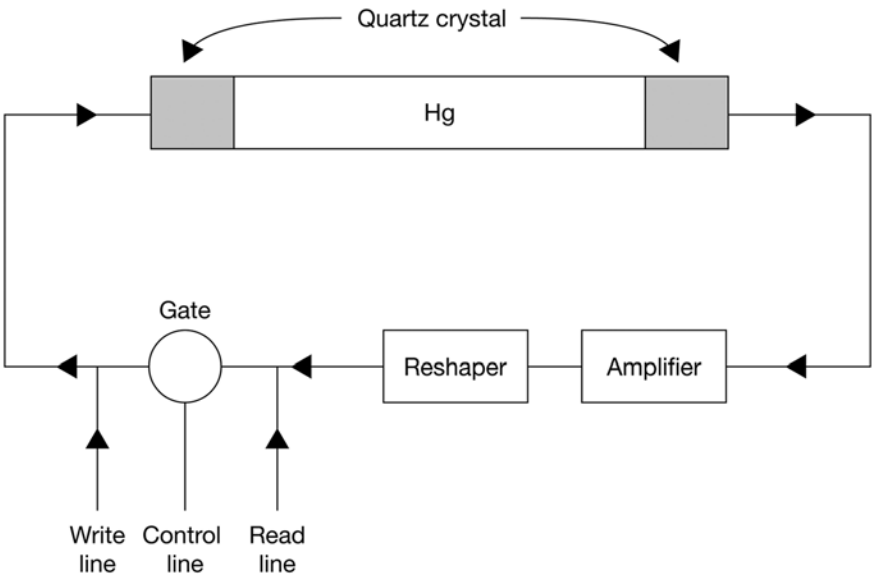

Figure 4.9
Schematic of the mercury delay line

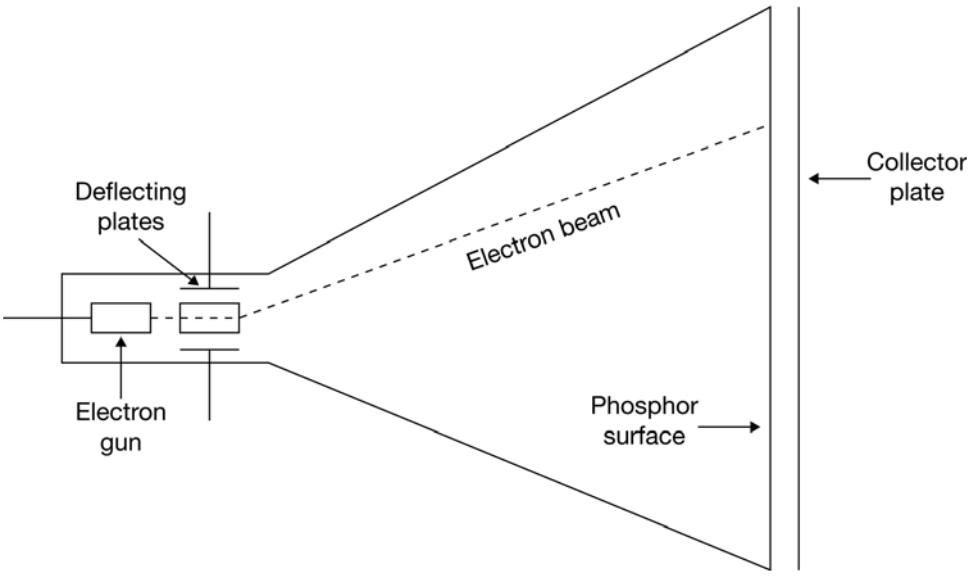

Figure 4.10
Schematic of the Williams tube

away and can be detected by a parallel collector plate. Thus, if this charged spot can be regenerated at least five times per second, memory can be produced in the same manner as the mercury delay tube. Current forms of computer memory also require regeneration.

Today's RAM is mostly volatile and based on flip-flop circuits and transistors and capacitors, which require a steady electrical current. Although we do have forms of nonvolatile memory, such as flash memory, made possible by better-insulated capacitors, they have a limited read-write cycle. Memory traces, to repeat Derrida's formulation, "produce the space of their inscription only by acceding to the period of their erasure."[119]

Thus, as Wolfgang Ernst has argued, digital media is truly *time-based media*, which, given a screen's refresh cycle and the dynamic flow of information in cyberspace, turns images, sounds, and text into a discrete moment in time. These images are frozen for human eyes only.[120] Information is dynamic, however, not only because it must move in space on the screen, but also, and more important, because it must move within the computer and because degeneration traditionally has made memory possible while simultaneously threatening it. Digital media, allegedly more permanent and durable than other media (film stock, paper, etc.), depends on a degeneration so actively denied and repressed. This degeneration, which engineers would like to divide into

useful versus harmful (erasability versus signal decomposition, information versus noise), belies and buttresses the promise of digital computers as permanent memory machines. If our machines' memories are more permanent, if they enable a permanence that we seem to lack, it is because they are constantly refreshed—rewritten—so that their ephemerality endures, so that they may "store" the programs that seem to drive them. To be clear, this is not to say that information is fundamentally immaterial; as Matthew Kirschenbaum has shown in his insightful *Mechanisms: New Media and the Forensic Imagination*, information (stored to a hard drive) leaves a trace that can be forensically reconstructed, or again, as I've argued elsewhere, for a computer, to read is to write elsewhere.[121] This is to say that if memory is to approximate something so long lasting as storage, it can do so only through constant repetition, a repetition that, as Jacques Derrida notes, is indissociable from destruction (or in Bush's terminology, forgetting).[122]

This enduring ephemeral—a battle of diligence between the passing and the repetitive—also characterizes content today. Internet content may be available 24/7, but 24/7 on what day? Further, if things constantly disappear, they also reappear, often to the chagrin of those trying to erase data. When A3G (article III groupie), the gossipy conservative and supposedly female author of underneaththeirrobes.blogs.com—a blog devoted to Supreme Court personalities—came out as a thirty-year-old Newark-based U.S. attorney named David Lat in an interview with the *New Yorker*, his site was temporarily taken down by the U.S. government.[123] Archives of his site—and of every other site that does not reject robots—however, are available at the Internet Wayback Machine (IWM, web.archive.org) with a six-month delay.

Like search engines, the Internet Wayback Machine comprises a slew of robots and servers that automatically and diligently, and in human terms, obsessively, back up most web pages. Also like search engines, they collapse the difference between the Internet, whose breadth is unknowable, and their backups; however, unlike search engines, the IWM does not use the data it collects to render the Internet into a library, but rather use these backups to create what the creators of the IWM call a "library of the Internet." The library the IWM creates, though, certainly is odd, for it has no coherent shelving system: the IWM librarians do not offer a card catalog or a comprehensive, content-based index.[124] This is because the IWM's head librarian is a machine, only capable of accumulating differing texts. That is, its automatic power of discrimination only detects updates within a text. The IWM's greatest oddity, however, stems from its recursive nature: the IWM diligently archives itself, including its archives, within its archive.

The imperfect archives of the IWM are considered crucial to the ongoing relevance of libraries. The IWM's creators state: "Libraries exist to preserve society's cultural artifacts and to provide access to them. If libraries are to continue to foster education and scholarship in this era of digital technology, it's essential for them to extend those

functions into the digital world."[125] The need for cultural memory drives the IWM and libraries more generally. Noting the loss of early film archives due to the recycling of early film stock, the archivists describe the imperative of building an "internet library":

> Without cultural artifacts, civilization has no memory and no mechanism to learn from its successes and failures. And paradoxically, with the explosion of the Internet, we live in what Danny Hillis has referred to as our "digital dark age."
>
> The Internet Archive is thus working to prevent the Internet—a new medium with major historical significance—and other "born-digital" materials from disappearing into the past. Collaborating with institutions including the Library of Congress and the Smithsonian, we are working to preserve a record for generations to come.[126]

The IWM is necessary because the Internet, which is in so many ways *about* memory, has, as Ernst argues, no memory—at least not without the intervention of something like the IWM.[127] Other media do not have a memory, but they do age and their degeneration is not linked to their regeneration. As well, this crisis is brought about because of this blinding belief in digital media as cultural memory. This belief, paradoxically, threatens to spread this lack of memory everywhere and plunge us negatively into a way-wayback machine: the so-called "digital dark age." The IWM thus fixes the Internet by offering us a "machine" that lets us control our movement between past and future by regenerating the Internet at a grand scale. The Internet Wayback Machine is appropriate in more ways than one: because web pages link to, rather than embed, images, which can be located anywhere, and because link locations always change, the IWM preserves only a skeleton of a page, filled with broken—rendered—links and images (figure 4.11). The IWM, that is, only backs up certain data types. These "saved"

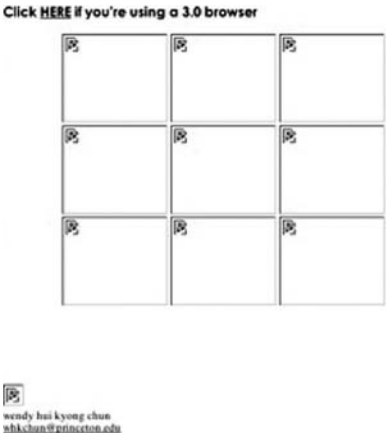

Figure 4.11
Screenshot of IWM backup of <http://www.princeton.edu/~whkchun/index.html>

pages are not quite dead, but not quite alive either, for their proper commemoration requires greater effort. These gaps not only visualize the fact that our constant regenerations affect what is regenerated, but also the fact that these gaps—the irreversibility of this causal programmable logic—are what open the World Wide Web as archive to a future that is not simply stored upgrades of the past.

Repetition and regeneration open the future by creating a nonsimultaneous new that confounds the chronological time these processes also enable. Consider, for instance, the temporality of weblogs (also known as *blogs*). Blogs seem to follow the timing of newspapers in their plodding chronology, but blogs contain within themselves archives of their posts, making the blog, if anything, like the epistolary novel. Unlike the epistolary novel, which, however banal, was focused on a plot or a moral, the blog entries are tied together solely by the presence of the so-called author. What makes a blog "uninteresting" is not necessarily its content, which often reads like a laundry list of things done or to do, but rather its immobility. The ever-updating, inhumanly clocked time in which our machines and memories are embedded and constantly refreshed makes the blog's material stale. The chronology, seemingly enabled by this time, is also compromised by these archives and the uncertainty of their regular reception. An older post can always be "discovered" as new; a new post is already old. This nonsimultaneousness of the new, this layering of chronologies, means that the gap between illocutionary and perlocutionary in high-speed telecommunications may be dwindling, but—because everything is endlessly repeated—response is demanded over and over again. The new is sustained by this constant demand to respond to what we do not yet know, by the goal of new media czars to continually create desire for what one has not yet experienced.

Digital media networks are not based on the regular obsolescence or disposability of information, but rather on the resuscibility or the undead of information. Even text messaging, which seems to be about the synchronous or the now, enables the endless circulation of forwarded messages, which are both new and old. Reliability is linked to deletion: a database is considered to be unreliable (to contain "dirty data"), if it does not adequately get rid of older, inaccurate information. Also, this repetition, rather than detracting from the message, often attests to its importance. Repetition becomes a way to measure scale in an almost inconceivably vast communications network.

Rather than getting caught up in speed then, what we must analyze, as we try to grasp a present that is always degenerating, are the ways in which ephemerality is made to endure. Paul Virilio's constant insistence on speed as distorting space-time and on real-time as rendering us susceptible to the dictatorship of speed has generated much good work in the field, but it can blind us to the ways in which images do not simply assault us at the speed of light.[128] Just because images flash up all of a sudden does not mean that response or responsibility is impossible, or that scholarly

analysis is no longer relevant. As the news obsession with repetition reveals, an image does not flash up only once. The pressing questions are: why and how is it that the ephemeral endures? And what does the constant repetition and regeneration of information effect? What loops and what instabilities does the enduring ephemeral introduce to the logic of programmability? What is surprising is not that digital media fades, but rather that it stays at all and that we remain transfixed at our screens as its ephemerality endures.

Notes

Preface

1. As I state in "Did Anyone Say New Media?," this demise does not coincide with the demise of media once called new, but rather with industry's quest to survive and thrive after the "new economy" and after new media's wide acceptance. Does it, after all, make sense to have a New Media Group within Apple in 2004? See *New Media, Old Media: A History and Theory Reader*, ed. Wendy Hui Kyong Chun and Thomas Keenan (New York: Routledge, 2006), 1–2.

Introduction

1. John Godfrey Saxe, "Blind Men and the Elephant," <http://www.wordinfo.info/words/index/info/view_unit/1/?letter=B&spage=3>, accessed 6/1/2008.

2. See Paul N. Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America* (Cambridge, Mass.: MIT Press, 1996).

3. The geneticists Luigi Luca and Francesco Cavalli-Sforza, for instance, argue: "It would be very difficult to change our hardware, our genetic makeup. It is much easier to try improving our software, our culture" in *The Great Human Diasporas: The History of Diversity and Evolution* (New York: Basic Books, 1995), 245.

4. Edwards, *Closed World*; Joseph Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation* (San Francisco: W. H. Freeman, 1976), 157.

5. Manfred Broy, "Software Engineering—From Auxiliary to Key Technology," in *Software Pioneers: Contributions to Software Engineering,* ed. Manfred Broy and Ernst Denert (Berlin: Springer, 2002), 11, 12.

6. Michael Mahoney, "The History of Computing in the History of Technology," *IEEE Annals of the History of Computing* 10, no. 2 (1988): 121.

7. As I argue later, however, all software is a reconstruction, a repetition.

8. Adrian Mackenzie, *Cutting Code: Software and Sociality* (New York: Peter Lang Pub. Inc., 2006), 169.

9. Herman H. Goldstine and John von Neumann, "Planning and Coding of Problems for an Electronic Computing Instrument," "Report on the Mathematical and Logical Aspects of an Electronic Computing Instrument," Part II, Volume I (Princeton, N.J.: Institute for Advanced Study, 1947), 2.

10. Paul Ceruzzi, *A History of Modern Computing*, 2nd ed. (Cambridge, Mass.: MIT Press, 2003), 80.

11. Friedrich Kittler, "There Is No Software," ctheory.net, October 18, 1995, <http://www.ctheory.net/text_file.asp?pick=74>, accessed 8/5/2010.

12. Mahoney, "History of Computing," 121.

13. Kathleen Broome Williams, *Grace Murray Hopper: Admiral of the Cyber Sea* (Annapolis, Md.: Naval Institute Press, 2004), 89; J. C. Chu, "Computer Development at Argonne National Laboratory," in *A History of Computing in the Twentieth Century*, ed. N. Metropolis et al. (New York: Academic Press, 1980), 345.

14. Martin Campbell-Kelly in *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, Mass.: MIT Press, 2003) divides the software industry into three period-based sectors: the software contractor (software as a service), software products, and mass-market products (3).

15. Herbert D. Benington, "Production of Large Computer Programs," *IEEE Annals of the History of Computing* 5, no. 4 (1983): 353.

16. This notion of programming is even evident in Fred Brooks's influential 1975 work *The Mythical Man-Month*, an analysis of the pitfalls of software programming based on his disastrous experiences with IBM's System/360. In it, he argues, "the purpose of a programming system is to make a computer easy to use"; in Frederick P. Brooks, *The Mythical Man-Month: Essays on Software Engineering,* 20th Anniversary Edition (New York: Addison-Wesley Professional, 1995), 43.

17. *Gottschalk v. Benson*, 409 U.S. 63 (1972); Pamela Samuelson, "Benson Revisited: The Case Against Patent Protection for Algorithms and Other Computer Program-Related Inventions," *Emory Law Journal* 39 (Fall 1990): 1053.

18. See Margaret Jane Radin, "Information Tangibility," in *Economics, Law, and Intellectual Property: Seeking Strategies for Research and Teaching in a Developing Field*, ed. Ove Granstrand (Dordrecht, The Netherlands: Kluwer Academic Publishers, 2003), 397.

19. *In re Alappat*, 33 F.3d at 1545, 31 USPQ2d at 1558 (Fed. Cir. 1994).

20. The U.S. Copyright Act of 1976 copyright states, "In no case does copyright protection for an original work of authorship extend to any idea, procedure, process, system, method of operation, concept, principle, or discovery, regardless of the form in which it is described, explained, illustrated, or embodied in such work."

21. Radin, "Information Tangibility," 407.

22. Rep. No. 473, 94th Cong., 1st Sess. 54 (1975).

23. 17 U.S.C. §102 (a).

24. Matthew Kirschenbaum, *Mechanisms: New Media and the Forensic Imagination* (Cambridge, Mass.: MIT Press, 2008).

25. Radin, "Information Tangibility," 406.

26. Ibid.

27. Ibid., 397.

28. She writes, "The compromise marries information to the external realm of objects. The structure of compromise involves the process of externalizing the internal ideas and embodying them in an object—a book. That is, the creative work starts out internal to the person, hence unpropertizable, but becomes embodied in an external object, hence propertizable" (Ibid., 398).

29. *The Oxford English Dictionary* (OED), 2nd ed., S.V. "information, *n.*"

30. Martin Heidegger, "The Thing," *Poetry, Language, Thought* (New York: HarperCollins, 1971), 176–177.

31. Colin Gordon, "Governmental Rationality: An Introduction," in *The Foucault Effect: Studies In Governmentality*, ed. Graham Burchell et al. (Chicago: Chicago University Press, 1991), 3. As Gordon explains, "Modern governmental rationalities consist, precisely, in the . . . 'daemonic' coupling of 'city-game' and 'shepherd-game': the invention of a form of secular political pastorate which couples 'individuation' and 'totalization'" (Ibid., 8). Foucault viewed sexuality, or, more broadly, technologies of the self, as crucial to the dual focus of both governmentality and his own research.

32. Foucault writes,

> If I deploy the word "liberal," it is first of all because this governmental practice in the process of establishing itself is not satisfied with respecting this or that freedom, with guaranteeing this or that freedom. More profoundly, it is a consumer of freedom. It is a consumer of freedom inasmuch as it can only function insofar as a number of freedoms actually exist: freedom of the market, freedom to buy and sell, the free exercise of property rights, freedom of discussion, possible freedom of expression, and so on. The new governmental reason needs freedom, therefore the new art of government consumes freedom. It consumes freedom, which means that it must produce it. It must produce it, it must organize it. The new art of government therefore appears as the management of freedom, not in the sense of the imperative: "be free," with the immediate contradiction that this imperative may contain. The formula of liberalism is not "be free." Liberalism formulates simply the following: I am going to produce what you need to be free. I am going to see to it that you are free to be free. And so, if this liberalism is not so much the imperative of freedom as the management and organization of the conditions in which one can be free, it is clear that at the heart of this liberal practice is an always different and mobile problematic relationship between the production of freedom and that which in the production of freedom risks limiting and destroying it. . . . Liberalism as I understand it, the liberalism we can describe as the art of government formed in the eighteenth century, entails at its heart a productive / destructive relationship [with]* freedom. . . . Liberalism must produce freedom, but this very act entails the establishment of limitations, controls, forms of coercion, and obligations relying on threats etcetera. (Michel Foucault, *The Birth of Biopolitics: Lectures at the College de France, 1978–1979*, trans. Graham Burchell [Basingstoke, England, and New York: Palgrave Macmillan, 2008], 63–64)

Foucault's analysis of liberalism thus resonates strongly with my previous argument, in *Control and Freedom: Power and Paranoia in the Age of Fiber Optics* (Cambridge, Mass.: MIT Press, 2006), that computers have been intimately linked to the bizarre reduction of freedom to control and

control technologies; but freedom, that book argued, cannot be reduced to control: freedom makes control possible, necessary, and never enough.

33. Foucault writes,

> By this word [governmentality] I mean three things. First, by "governmentality" I understand the ensemble formed by the institutions, procedures, analyses and reflections, calculations and tactics that allow the exercise of this very specific, albeit very complex, power that has as its target population, political economy as its major form of knowledge, and apparatuses of security as it essential technical instrument. Second, by "governmentality," I understand the tendency, the line of force that, for a long time, and throughout the West, has constantly led towards the pre-eminence over all other types of power—sovereign, discipline, and so on—of the type of power which may be termed government, and which has led to the development of a series of specific governmental apparatuses . . . on the other hand . . . to the development of a series of knowledges. Finally, by "governmentality," I think we should understand the process, or rather, the result of the process, by which the state of justice of the Middle Ages became the administrative state during fifteenth and sixteenth centuries and was gradually "governmentalized." (Michel Foucault, *Security, Territory, Population: Lectures at the College de France, 1977–1978*, trans. Graham Burchell [Basingstoke, England, and New York: Palgrave Macmillan, 2007], 108–109)

34. For more on this, see Martin Campbell-Kelly and William Aspray, *Computer: A History of the Information Machine* (New York: Basic Books, 1996), 20–30.

35. Ibid., 22.

36. See David Alan Grier, *When Computers Were Human* (Princeton, N.J.: Princeton University Press, 2005).

37. David Harvey, *A Brief History of Neoliberalism* (Oxford: Oxford University Press, 2005), 2.

38. Foucault, *Birth of Biopolitics*, 117.

39. Friedman, *Capitalism and Freedom*, 4.

40. As quoted by David Harvey, *Brief History of Neoliberalism*, 23.

41. Foucault, *Birth of Biopolitics*, 252.

42. Ibid., 147.

43. Friedman, *Capitalism and Freedom*, 13; emphasis in original.

44. In this system, innovation "is nothing other than the income of . . . human capital, that is to say, of the set of investments we have made at the level of man himself" (Ibid., 231).

45. An intriguing symptom of this is the popularity of the *You* book series by Michael F. Roizen and Mehmet Oz. The first book is called *You: The Owner's Manual: An Insider's Guide to the Body that Will Make You Healthier and Younger* (New York: Harper, 2005).

46. Catherine Malabou, *What Should We Do with Our Brains*, trans. Sebastian Rand (New York: Fordham University Press, 2008), 44.

47. Ben Shneiderman, "Direct Manipulation: A Step Beyond Programming Languages," in *The New Media Reader*, ed. Noah Wardrip-Fruin and Nick Montfort (Cambridge, Mass.: MIT Press, 2003), 486.

48. As quoted in Fiona Morrow, "The Future Catches Up with Novelist William Gibson," *The Globe and Mail*, last updated April 3, 2009, <http://www.theglobeandmail.com/news/arts/article786109.ece>, accessed 9/9/2009.

49. See David Glovin and Christine Harper, "Goldman Trading-Code Investment Put at Risk by Theft," last updated July 6, 2009, *bloomberg.com*, <http://www.bloomberg.com/apps/news?pid=20601087&sid=a_6d.tyNe1KQ>, accessed 9/9/2009.

50. Joseph Weizenbaum has argued,

> A large program is, to use an analogy of which Minsky is also fond, an intricately connected network of courts of law, that is, of subroutines, to which evidence is transmitted by other subroutines. These courts weigh (evaluate) the data given to them and then transmit their judgments to still other courts. The verdicts rendered by these courts may, indeed, often do, involve decisions about what court has "jurisdiction" over the intermediate results then being manipulated. The programmer thus cannot even know the path of decision-making within his own program, let alone what intermediate or final results it will produce. Program formulation is thus rather more like the creation of a bureaucracy than like the construction of a machine of the kind Lord Kelvin may have understood. (*Computer Power and Human Reason: From Judgment to Calculation* [San Francisco: W. H. Freeman, 1976], 234)

51. Bill Brown, "Thing Theory," *Critical Inquiry* 28, no. 1, Things (Autumn 2001): 4–5.

52. Ibid., 5.

53. Ibid., 3.

54. Ibid., 4.

55. Ibid., 5.

56. Ibid., 4.

57. Brown, "Thing Theory," 16; Heidegger, "The Thing," 165.

I Invisibly Visible, Visibly Invisible

1. As quoted by John Schwartz, "Privacy Fears Erode Support for a Network to Fight Crime," *New York Times*, March 15, 2004, C1. Seisint is a corporation developing The Matrix, a computer system described later in this section.

2. For more on this debate, see Thomas Elsaesser, "Early Film History and Multi-Media: An Archaeology of Possible Futures?," in *New Media, Old Media: A History and Theory Reader*, ed. Wendy Hui Kyong Chun and Thomas Keenan (New York: Routledge, 2006), 13–25; and Don E. Tomlinson, "One Technological Step Forward and Two Legal Steps Back: Digitalization and Television Newspictures as Evidence and as Libel," *Loyola of Los Angeles Entertainment Law Journal* 9 (1989): 237.

3. Roland Barthes, *Camera Lucida* (New York: Hill & Wang, 1981), 88.

4. Mary Ann Doane, *The Emergence of Cinematic Time: Modernity, Contingency, the Archive* (Cambridge, Mass., and London, England: Harvard University Press, 2002), 223.

5. *Crime Scene Investigation* (*CSI*) is a popular U.S. television program (also syndicated internationally), which features forensic investigations of crimes. Similarly, scientific digital animations of subcellular activity are the closest thing we have to true representations of activities that can be gleaned but not seen.

6. Jean Baudrillard. *The Ecstasy of Communication*, trans. Bernard and Caroline Schutze (Brooklyn, N.Y.: Autonomedia, 1988), 21–22; emphasis in original.

7. Information Awareness Office, Total Information Awareness Program (TIA) System Description Document, version 1.1 (July 19, 2002), <http://www.epic.org/privacy/profiling/tia/tiasystemdescription.pdf>, accessed 2/1/2007.

8. As quoted by Seth Finkelstein, "Google's Surveillance Is Taking Us Further Down the Road to Hell," *The Guardian* (March 26, 2009), <http://www.guardian.co.uk/technology/2009/mar/26/seth-finkelstein-google-advertising>, accessed 4/1/2009.

9. Joshua Gomez, Travis Pinnick, and Ashkan Soltani, "KnowPrivacy," June 1, 2009, <http://www.knowprivacy.org/report/KnowPrivacy_Final_Report.pdf>, 4, accessed 7/1/2009.

10. Paul Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America* (Cambridge, Mass.: MIT Press, 1996); Joseph Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation* (San Francisco: W. H. Freeman, 1976), 157.

11. Ibid.

12. Ibid., 234.

13. What, for instance, is Microsoft Word? Is it the encrypted executable—residing on a CD or one's hard drive—or its source code, or is it its execution? These, importantly, are not the same, even though they are all called Word: not only are they in different locations, but also two are programs while the other is a process. Structured programming, as discussed later, has been key to the conflation of program with process.

1 On Sourcery and Source Codes

1. Johann Wolfgang Goethe, "6. Faust's Study (i)," *Faust, Part One*, trans. David Luke (Oxford and New York: Oxford University Press, 1987), line 1236–1237.

2. As Barbara Johnson notes in her explanation of Jacques Derrida's critique of logocentrism, logos is the "image of perfectly self-present meaning . . . , the underlying ideal of Western culture. Derrida has termed this belief in the self-presentation of meaning, 'Logocentrism,' for the Greek word *Logos* (meaning speech, logic, reason, the Word of God)" in Translator's Introduction, *Dissemination*, trans. Barbara Johnson (Chicago: University of Chicago, 1981), ix.

3. For instance, *The A-2 Compiler System Operations Manual* prepared by Richard K. Ridgway and Margaret H. Harper under the direction of Grace M. Hopper (Philadelphia: Remington Rand, 1953) explains that a pseudo-code drives its compiler, just as "C-10 Code tells UNIVAC how to proceed. This pseudo-code is a new language which is much easier to learn and much shorter

and quicker to write. Logical errors are more easily found in information than in UNIVAC coding because of the smaller volume" (1).

4. Jacques Derrida, "Plato's Pharmacy," *Dissemination*, trans. Barbara Johnson (Chicago: University of Chicago, 1981), 115.

5. For example, George Landow, *Hypertext 2.0: The Convergence of Contemporary Literary Theory and Technology* (Baltimore, Md.: Johns Hopkins University Press, 1997); Sherry Turkle, *Life on the Screen: Identity in the Age of the Internet* (New York: Simon and Schuster, 1997); and Greg Ulmer, *Applied Grammatology: Post(e)-Pedagogy from Jacques Derrida to Joseph Beuys* (Baltimore, Md.: Johns Hopkins University Press, 1984). In her important work, N. Katherine Hayles has investigated the differences between poststructuralist writing and code in *My Mother Was a Computer: Digital Subjects and Literary Texts* (Chicago: University of Chicago Press, 2005).

6. Lev Manovich, *The Language of New Media* (Cambridge, Mass.: MIT Press, 2001), 48; emphasis in original.

7. *Vapor theory* is a term coined by Peter Lunenfeld and used by Geert Lovink to designate theory so removed from actual engagement with digital media that it treats fiction as fact. This term, however, can take on a more positive resonance, if one takes the nonmateriality of software seriously. See Geert Lovink, "Enemy of Nostalgia, Victim of the Present, Critic of the Future Interview with Peter Lunenfeld," July 31, 2000, <http://www.nettime.org/Lists-Archives/nettime-l-0008/msg00008.html>, accessed 2/1/2007.

8. Consider, for instance, the effectiveness of Chris Csikszentmihályi's largely unrealized *Afghan Explorer Robot*—covered by numerous news sources, it raised fundamental questions about restrictions on reporters covering war.

9. Alexander Galloway, *Protocol: How Power Exists after Decentralization* (Cambridge, Mass.: MIT Press, 2004), 164.

10. McKenzie Wark, "A Hacker Manifesto," version 4.0, <http://subsol.c3.hu/subsol_2/contributors0/warktext.html>, accessed 1/1/2010.

11. See Richard Stallman, "The Free Software Movement and the Future of Freedom; March 9th 2006," <http://fsfeurope.org/documents/rms-fs-2006-03-09.en.html>, accessed 6/1/2008. Immanuel Kant famously described the Enlightenment as "mankind's exit from its self-incurred immaturity" in "An Answer to the Question: What Is Enlightenment," in *What Is Enlightenment? Eighteenth-Century Answers and Twentieth-Century Questions*, ed. James Schmidt (Berkeley: University of California Press, 1996), 58.

12. For more on enlightenment as a stance of how not to be governed like that, see Michel Foucault, "What Is Critique?," in *What Is Enlightenment?*, ed. Schmidt, 382–398.

13. GNU copyleft/Public Licence (GPL) is symptomatic of the move in contemporary society away from the public/private dichotomy to that of open/closed. As Niva Elkin-Koren notes in "Creative Commons: A Skeptical View of a Worthy Project," the Creative Commons strategy "does not aim at creating a public domain, at least not in the strict legal sense of a regime that

is free of any exclusive proprietary rights. The strategy is entirely dependent upon a proprietary regime and drives its legal force from its existence. The normative framework assumes that it is possible to replace existing practices of producing and distributing informational works by relying on the existing proprietary regime. The underlying assumption is that if intellectual property rights remain the same, but rights are exercised differently by their owners, free culture would emerge" (<http://www.hewlett.org/NR/rdonlyres/6D4BFD1E-09BB-4F89-9208-7C1E4B141F2A/0/Creative_Commons_Amsterdam_final2006.pdf>, accessed 6/1/2008). Although Elkin-Koren is writing about Creative Commons in this passage, she makes it clear that this strategy of extending and revising intellectual property rights is drawn from the Free Software Movement's GPL.

14. Galloway, *Protocol*, 165–166; emphasis in original. Given that the adjective *executable* applies to anything that "can be executed, performed, or carried out" (the first example of *executable* given by the OED is from 1796), this is a strange statement.

15. See Derrida's analysis of *The Phaedrus* in "Plato's Pharmacy," 165–166.

16. John 1:1, *Bible, King James Version*.

17. Hayles, *My Mother Was a Computer*, 50. Hayles's argument immediately poses the question: What counts as internal versus external to the machine, especially given that, in John von Neumann's foundational description of stored program computing, the input and output (the outside world to the machine) was a form of memory?

18. Alexander Galloway, "Language Wants to Be Overlooked: Software and Ideology," *Journal of Visual Culture* 5, no. 3 (2006): 326.

19. Ibid.

20. Galloway, *Protocol*, 167.

21. Galloway, "Language Wants to Be Overlooked," 321.

22. This example draws from the *PowerPC Assembly Language Beginners Guide*, <http://www.lightsoft.co.uk/Fantasm/Beginners/Chapt1.html>, accessed 7/1/2009.

23. Paul Ricoeur, *The Rule of Metaphor*, trans. Robert Czerny et al. (London and New York: Routledge, 2003), 31.

24. *The Oxford English Dictionary* (OED), 2nd ed., S.V. "technical, *a*.(*n*.)"

25. Jacques Derrida stresses the disappearance of the origin that writing represents: "To repeat: the disappearance of the good-father-capital-sun is thus the precondition of discourse, taken this time as a moment and not as a principle of *generalized* writing. . . . The disappearance of truth as presence, the withdrawal of the present origin of presence, is the condition of all (manifestation of) truth. Nontruth is the truth. Nonpresence is presence. Differance, the disappearance of any originary presence, is *at once* the condition of possibility *and* the condition of impossibility of truth. At once" ("Plato's Pharmacy," 168).

26. Jacques Derrida, "Freud and the Scene of Writing," *Writing and Difference*, trans. Alan Bass (Chicago: University of Chicago Press, 1978), 228.

27. Herman H. Goldstine and John von Neumann, "Planning and Coding of Problems for an Electronic Computing Instrument," "Report on the Mathematical and Logical Aspects of an Electronic Computing Instrument," Part II, Volume I (Princeton, N.J.: Institute for Advanced Study, 1947), 2. They also state, "A coded order stands not simply for its present contents at its present location, but more fully for any succession or passages of C through it, in connection with any succession of modified contents to be found by C there, all of this being determined by other orders of the sequence."

28. Ibid.

29. Jacques Derrida in "Signature Event Context," in *Limited Inc.*, trans. Samuel Weber and Jeffrey Mehlman (Evanston, Ill.: Northwestern University Press, 1988 [1977]), argues, "every sign . . . can be *cited*, put between quotation marks; in so doing it can break with every given context, engendering an infinity of new contexts in a manner which is absolutely illimitable. This does not imply that the mark is valid outside of a context, but on the contrary that there are only contexts without any center or absolute anchoring [*ancrage*]. This citationality, duplication or duplicity, this iterability of the mark is neither an accident nor an anomaly, is that (normal/abnormal) without which a mark could not even have a function called 'normal.' What would a mark be that could not be cited? Or one whose origins would not get lost along the way?" (12). Against this, Hayles, in *My Mother Was a Computer*, contends that computer code is not iterable because "the contexts are precisely determined by the level and nature of the code. Code may be rendered unintelligible if transported into a different context—for example, into a different programming language. . . . Only at the high level of object-oriented languages such as C++ does code recuperate the advantages of citability and iterability . . . and thus become 'grammatological'" (48).

30. For more on the software crisis and its relationship to software engineering, see Martin Campbell-Kelly and William Aspray, *Computer: A History of the Information Machine* (New York: Basic Books, 1996),196–203; Paul Ceruzzi, *A History of Modern Computing*, 2nd ed. (Cambridge, Mass.: MIT Press, 2003), 105; and Frederick P. Brooks's *The Mythical Man-Month: Essays on Software Engineering*, 20th Anniversary Edition (New York: Addison-Wesley Professional, 1995).

31. Philip E. Agre, *Computation and Human Experience* (Cambridge: Cambridge University Press, 1997), 92.

32. Norman Macrae, *John von Neumann* (New York: Pantheon Books, 1992), 378.

33. According to Friedrich Nietzsche, "there is no 'being' behind the doing, effecting, becoming: 'the doer' is merely a fiction added to the deed—the deed is everything." In *The Birth of Tragedy & The Genealogy of Morals*, trans. Francis Golffing (Garden City, N.Y.: Doubleday, 1956), 178–179.

34. Lawrence Lessig, *Code: And Other Laws of Cyberspace* (New York: Basic Books, 1999).

35. Code as automatically enabling and disabling actions is also code as police. I elaborate more on this in the "Crisis, Crisis, Crisis, or the Temporality of Networks" chapter of *Imagined Networks* (work in progress).

36. Milton Friedman, for instance, argues "the role of government just considered is to do something that the market cannot do for itself, namely, to determine, arbitrate, and enforce the rules of the game." See *Capitalism and Freedom*, Fortieth Anniversary Edition (Chicago: Chicago University Press, 2002), 27.

37. Ibid., 25.

38. Michel Foucault, *The Birth of Biopolitics: Lectures at the Collège de France, 1978–1979*, trans. Graham Burchell (Basingstoke, England, and New York: Palgrave Macmillan, 2008), 175.

39. David Golumbia, *The Cultural Logic of Computation* (Cambridge, Mass.: Harvard University Press, 2009), 224.

40. Judith Butler, *Excitable Speech: A Politics of the Performative* (New York: Routledge, 1997), 48.

41. Ibid., 78.

42. Ibid., 49; emphasis in original.

43. Joseph Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation* (San Francisco: W. H. Freeman, 1976), 234.

44. Martin Heidegger, "The Thing," *Poetry, Language, Thought* (New York: HarperCollins, 1971), 176–177.

45. Foucault, *Birth of Biopolitics*, 173.

46. None were classified as mathematicians, even though many of these women, hired to calculate ballistic trajectories using Marchant machines and even the differential analyzer, did have college degrees in mathematics (the "ENIAC girls," however, were promoted quickly to "mathematicians" as the war progressed). Special training courses in calculus and other "higher" maths were offered to those without college degrees by instructors such as Adele Goldstine, wife of Herman Goldstine and documenter of the ENIAC, as well as Mary Mauchly, wife of John Mauchly. This division between mathematicians and computers also existed during World War I, with graduate students and young assistant professors labeled as "computers," although the relations between them were less rigid. See David Grier, *When Computers Were Human* (Princeton, N.J.: Princeton University Press, 2005).

This notion of women as ideal programmers also persisted after the war with SAGE (discussed later) and projects such as Pacific Mutual's "Project Helpmate," a project that treated wives as interchangeable labor units to meet short-run staffing needs:

> To do that, we had to make sure we didn't say that Ann is better than Sue and Sue is better than Jane. We employed them for the same salary amount, no matter what their experience, and for the same deal, which was that they would get so much a month and if they stayed til the end of the agreed upon time they got a bonus of so much. This was "project Helpmate." It was the only way we could get people, because we needed a lot of people for a limited period of time, and then wouldn't need them. Not only we wouldn't need them, but we thought we would need fewer of our other staff. That whole process is another story in itself. (Richard. D. Dotts, "Interview," May 21, 1973, *Computer Oral History Collection 1969–1973, 1977*, Archives Center, National Museum of American History, Smithsonian Institution [box 82], 23)

Women were also coders presumably because they would be more accepting of a job in which advancement was not clearly defined. As Dotts notes, programmers were viewed as easily expendible.

47. Jean J. Bartik, Frances E. Snyder Holberton, and Henry S. Tropp, "Interview," April 27, 1973, *Computer Oral History Collection, 1969–1973, 1977* (Washington, D.C.: Archives Center, National Museum of American History), <http://invention.smithsonian.org/downloads/fa_cohc_tr_bart730427.pdf>, accessed 8/8/2009, 12–13.

48. See "Women of the ENIAC: An Oral History," February 23, 2005 <http://www.witi.com/center/aboutwiti/videos/eniac_kay_qt_hi.php>, accessed 8/8/2009.

49. Paul N. Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America* (Cambridge, Mass.: MIT Press, 1996), 71.

50. I. J. Good, "Pioneering Work on Computers at Bletchley," in *A History of Computing in the Twentieth Century: A Collection of Essays*, ed. N. Metropolis et al. (New York: Academic Press, 1980), 31–46.

51. Quoted in Ceruzzi, *A History of Modern Computing*, 82.

52. See Neal Stephenson, *In the Beginning . . . Was the Command Line* (New York: HarperCollins Publishers Inc., 1999).

53. See Heinz von Foerster, "Epistemology of Communication," in *The Myths of Information: Technology and Postindustrial Culture*, ed. Kathleen Woodward (Madison, Wisc.: Coda Press, 1980), 18–27.

54. See Norbert Wiener, "Cybernetics in History," *The Human Use of Human Beings* (New York: Da Capo Press, 1950), 15–27.

55. See Michael S. Mahoney, "Finding a History for Software Engineering," *IEEE Annals of the History of Computing* 26, no. 1 (January–March 2004): 8–19.

56. Johann Wolfgang Goethe, "7. Faust's Study (ii)," *Faust, Part One*, trans. David Luke (Oxford and New York: Oxford University Press, 1987), line 1648. The older translation by Philip Wayne uses the term *slave* (Baltimore, Md.: Penguin Books, 1949), 86.

57. According to Antonelli, they were hired after construction was well under way because it was feared that the assembled crew would disband soon (the machine was only operational after World War II).

58. Bartik, Holberton and Tropp, "Interview," 68.

59. To learn how to operate it, they went to Aberdeen to learn IBM punch card and plug board technology, and they studied the block diagrams fastidiously.

60. Bartik, Holberton and Tropp, "Interview," 54–55.

61. Ibid., 51.

62. Ibid., 38.

63. Ibid., 31.

64. Ibid., 33. Through this, she sought "to design something [so] that we could visualize what we were doing and also to devise something so that someone else could understand what we were doing to check it."

65. Ibid., 62. In essence, if you could take a motion picture of the machine, you would have been able to reproduce the numbers. The term *breakpoint* stems from the fact that the ENIAC programmers would actually pull the wire to stop the programs and read the accumulators (56).

66. Ibid., 56.

67. W. Barkley Fritz, "The Women of ENIAC," *IEEE Annals of the History of Computing* 18, no. 3 (September 1996): 21.

68. As argued earlier, although software produces visible effects, software itself cannot be seen. Luce Irigaray, in *This Sex Which is Not One*, trans. Catherine Porter (Ithaca, N.Y.: Cornell University Press, 1985), has similarly argued that feminine sexuality is nonvisual: "her sexual organ represents *the horror of nothing to see"* (emphasis in original, 26).

69. Sadie Plant, *Zeros + Ones: Digital Women + The New Technoculture* (New York: Doubleday, 1997), 37.

70. As quoted by Galloway, *Protocol*, 188.

71. Ibid.

72. Kathleen Broome Williams, *Grace Murray Hopper: Admiral of the Cyber Sea* (Annapolis, Md.: Naval Institute Press, 2004), 89; J. C. Chu, "Computer Development at Argonne National Laboratory," in *History of Computing*, ed. N. Metropolis et al., 345.

73. For example, see Fritz, "The Women of ENIAC," 13–28; Jennifer Light, "When Computers Were Women," *Technology and Culture* 40, no. 3 (July 1999): 455–483.

74. She nonetheless insisted that a woman who wanted to pursue mathematics in the Navy was a different story since "women have always done mathematics, since the days of the Greeks" (Hopper, "The Captain is a Lady," *60 Minutes Transcript*, March 1983, 11).

75. She states, "The novelty of inventing programs wears off and degenerates into the dull labor of writing and checking programs . . . this duty . . . looms as an imposition on the human brain." See "The Education of a Computer," *IEEE Annals of the History of Computing* 9, no. 3 (July–December 1987): 273. This rehumanizing of the mathematician was also to coincide with the mechanization (or perhaps, given the legacy of the ENIAC's "master programmer," the remechanization) of the professional computer.

Thus by considering the professional programmer (not the mathematician), as an integral part of the computer, it is evident that the memory of the programmer and all information and data to which he can refer is available to the computer subject only to translation into suitable language. And it is further evident that the

computer is fully capable of remembering and acting upon any instructions once presented to it by the programmer.

With some specialized knowledge of more advanced topics, UNIVAC at present has a well-grounded mathematical education fully equivalent to that of a college sophomore, and it does not forget and does not make mistakes. It is hoped that its undergraduate course will be completed shortly and it will be accepted as a candidate for a graduate degree. (Ibid., 280–281)

The compiler was thus to enfold professional programmers within the machine itself, by dumping their memory into the machine.

76. Bartik, Holberton, and Tropp, "Interview," 105–106.

77. Goldstine and von Neumann, "Planning and Coding," section 7.9, 20, <http://www.admin.ias.edu/library/hs/da/ECP/Planning_Coding_Problems_v2p1.pdf>, accessed 8/1/2009.

78. Nathan Ensmenger and William Aspray, "Software as Labor Process," in *Proceedings of the International Conference on History of Computing: Software Issues* (New York: Springer-Verlag, 2000), 158.

79. Henry S. Tropp et al., "A Perspective on SAGE: Discussion," *IEEE Annals of the History of Computing* 5, no. 4 (October 1983): 387.

80. Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog: A History of the Software Industry* (Cambridge, Mass.: MIT Press, 2004), 68–69. There was some resistance, as Irwin Greenwald points out, to this method: "We always separated the machine function from the procedures function or the programming function or what have you, which was very difficult for some people that we hired to get accustomed to. Stan Rothman, for example, was used to doing the whole job himself. I guess the years he was there he never did get to the point where he could accept our style. I know three and four different people who tried to work with him and gave up because he insisted on doing it the way he was used to, which was good, but not our style" (Greenwald and Robina Mapstone, "Interview," April 3, 1973, *Computer Oral History Collection 1969–1973, 1977, Archives Center, National Museum of American History, Smithsonian Institution*, Box 8, Folder 10, 17–18).

81. Philip Kraft, *Programmers and Managers: The Routinization of Computer Programming in the United States* (New York: Springer-Verlag, 1984), 15–16.

82. Tropp et al., "A Perspective on SAGE," 386.

83. Ibid., 385.

84. Ibid., 387.

85. Mahoney, "Finding a History," 15.

86. Kraft, *Programmers and Managers*, 99.

87. Edsger W. Dijkstra, "EWD 1308: What Led to 'Notes on Structure Programming,'" in *Software Pioneers: Contributions to Software Engineering*, ed. Manfred Broy and Ernst Denert (Berlin: Springer, 2002), 342.

88. Edsger W. Dijkstra, "Go To Statement Considered Harmful," in *Software Pioneers*, ed. Broy and Denert, 352.

89. Structured programming was introduced as a way to make the programs rather than the programmer priest the source, although the term *programmer priest* complicates the notion of *source:* is the source the programmer or some mythic power she mediates?

90. Edsger Dijkstra, "Notes on Structured Programming," April 1970, <http://www.cs.utexas.edu/~EWD/ewd02xx/EWD249.PDF>, accessed 8/8/2009, 7.

91. Ibid., 21.

92. John V. Guttag, "Abstract Data Types, Then and Now," in *Software Pioneers*, ed. Broy and Denert, 444.

93. Ibid.

94. Ibid., 445.

95. Thomas Keenan, "The Point Is to (Ex)Change It: Reading *Capital* Rhetorically," in *Fetishism as Cultural Discourse*, ed. Emily Apter and William Pietz (Ithaca, N.Y.: Cornell University Press, 1993), 165.

96. David Eck, *The Most Complex Machine: A Survey of Computers and Computing* (Natick, Mass.: A. K. Peters, 2000), 329, 238.

97. Adele Mildred Koss, "Programming on the Univac 1: A Woman's Account," *IEEE Annals of the History of Computing* 25, no. 1 (January–March 2003): 49.

98. Grier, *When Computers Were Human*.

99. Ibid., 33.

100. Ibid., 40.

101. After the Depression, it evolved into a more sophisticated unit, which used more complex numerical methods and Marchant machines to perform the basic mathematics.

102. Gertrude Blanch and Henry Tropp, "Interview," May 16, 1973, *Computer Oral History Collection 1969–1973, 1977, Archives Center, National Museum of American History, Smithsonian Institution,* <http://invention.smithsonian.org/downloads/fa_cohc_tr_blan730516.pdf>, accessed 8/8/2009, 3.

103. Ida Rhodes and Henry Tropp, "Interview," March 21, 1973, *Computer Oral History Collection 1969–1973, 1977, Archives Center, National Museum of American History, Smithsonian Institution*, <http://invention.smithsonian.org/downloads/fa_cohc_tr_rhod730321.pdf>, accessed 8/8/2009, 2.

104. Ibid.

105. Ibid., 3.

106. Blanch and Tropp, "Interview," 32.

107. Rhodes and Tropp, "Interview," 9.

108. "We were obsessed—especially Gertrude, who had fallen in love with the marvelous Tables of the British—with the idea that nothing but accuracy counts. And for this reason we had made it very clear to everybody that it was all [we] asked of them—complete accuracy" (Ibid., 12).

109. Ibid.

110. Blanch and Tropp, "Interview," 10.

111. Ibid., 12–13.

112. Alan Turing, "Proposal for Development in the Mathematics Division of an Automatic Computing Engine (ACE)," in *A. M. Turing's ACE Report of 1946 and Other Papers*, ed. B. E. Carpenter and R. W. Doran (Cambridge, Mass.: MIT Press, 1986), 38–39.

113. Koss, "Programming on the Univac 1," 56.

114. See Derrida, "Signature Event Context," 1–23.

115. *The A-2 Compiler System Operations Manual* prepared by Richard K. Ridgway and Margaret H. Harper under the direction of Grace M. Hopper (Philadelphia: Remington Rand, 1953), 53. This description makes it clear that pseudocode is not source. This manual, rather, refers to pseudo-code as information:

> The A-2 Compiler is a programming system for UNIVAC which produces, as its output, the complete coding necessary for the solution of a specific problem. If the problem has been correctly described to the compiler the coding will be correct and checked (by UNIVAC) and the program tape may be immediately run without any debugging. . . . The coding necessary for the solution of a specific problem is ordered by the programmer in pseudo-code. This pseudo-code is called "information" and it is the information which tells the compiler how to proceed just as C-10 Code tells UNIVAC how to proceed. This pseudo-code is a new language which is much easier to learn and much shorter and quicker to write. Logical errors are more easily found in information than in UNIVAC coding because of the smaller volume. (Ibid., 1; emphasis in original)

This passage nicely highlights the difference that pseudo-code makes: rather than dealing with the minutiae of machine programming (which in the case of UNIVAC was much easier than other machines because its native C-10 used alphanumerics rather than binary numbers), pseudo-code enables the programmer to address the logical problem at hand. Pseudocode was thus not conceived initially as "source code," but rather as an artifice—a supplement—that produced intermediate code that could be debugged.

116. Hardware was also an important limiting factor: the languages produced by the UNIVAC team did not penetrate widely because the UNIVAC did not; the Laining and Zierler system was limited to the Whirlwind.

117. John Backus, "Programming in America in the 1950s—Some Personal Impressions," in *History of Computing*, ed. N. Metropolis et al., 127.

118. Koss, "Programming on the Univac 1," 58.

119. Bob Everett, "Interview," May 29, 1970, *Computer Oral History Collection 1969–1973, 1977*, Archives Center, National Museum of American History, Smithsonian Institution (box 7, folder 2), 23.

120. Hopper goes on to say, "I should have studied a little harder but it wasn't that much that weighed you down and I just promptly relaxed into it like a featherbed and gained weight and had a perfectly heavenly time whereas the youngsters were very busy rebelling against the uniforms and the regulations and having to eat what was put in front of them and everything and I was just tickled to pieces by that time to have a beautiful meal put in front of me and I'd eat it." See Hopper and Tropp, "Interview," July 1, 1968, *Computer Oral History Collection 1969–1973, 1977*, Archives Center, National Museum of American History, Smithsonian Institution, <http://invention.smithsonian.org/downloads/fa_cohc_tr_hopp680700.pdf>, 26, accessed 9/9/2009. This freedom would make all the difference between her career trajectory (as a divorced woman with no children) and that of the other females around her who mostly left computing when they married or became pregnant.

121. United States Information Agency, *Voice of America Interviews with Eight American Women of Achievement* (College Park, Md.: National Archives and Records Administration, 1984), 9. Emblematically, Hopper loved drill, which she compared to dancing, and she became a battalion commander (Williams, *Grace Murray Hopper*, 22). The promotion to battalion commander also explains why she enjoyed the Navy so much. Midshipmen's school, after all, entailed both learning how to follow and how to give orders: "Thirty days to learn how to take orders, and thirty days to learn how to give orders, and you were a naval officer" (as quoted in Williams, *Grace Murray Hopper*). This learning to take and give orders made it possible for Hopper to reconcile discipline with leadership.

122. Relaying a visit to the Mark I, she notes, "nobody understood about computers, and they showed him how that they put the cards in the card reader and then they told this poor guy that those cards traveled down in that big round thing, and went up in the counter, and the poor guy believed it. They used to tell people the most horrible things, and I'm afraid we were at least partially responsible for some of the mythology that grew up around computers." See "Interview," *Computer Oral History Collection 1969–1973, 1977*, Archives Center, National Museum of American History, Smithsonian Institution, January 7, 1969, <http://invention.smithsonian.org/downloads/fa_cohc_tr_hopp690107.pdf>, accessed 9/9/2009, 12. For her "sufferings," see Williams, *Grace Murray Hopper*, 31. For her comment regarding experienced programmers, see G. M. Hopper and H. W. Mauchly, "Influence of Programming Techniques on the Design of Computers," *Proceedings of the IRE* 41, no. 10 (October 1953): 1250.

123. Harry Reed, "My Life with the ENIAC: A Worm's Eye View," in *Fifty Years of Army Computing: From ENIAC to MSRC*, ed. Thomas Bergin (U.S. Army Research Laboratory, 2000), 158; emphasis in original; accessible online at <http://ftp.arl.mil/~mike/comphist/harry_reed.pdf>.

124. Bartik, Holberton, and Tropp, "Interview," 121.

125. Ibid., 93.

126. Jean E. Sammet, *Programming Languages: History and Fundamentals* (Englewood Cliffs, N.J.: Prentice-Hall, 1969), 144.

127. Ibid., 148.

128. See, for instance, Theodor Nelson, *Computer Lib; Dream Machines* (Redmond, Wash.: Tempus Books of Microsoft Press, 1987).

129. David Golumbia's otherwise insightful analysis of computationalism also seeks to divide computing into two clear perspectives—the sovereign or the slave: "From the perspective we have been developing here, the computer encourages a Hobbesian conception of this political relation: one is either the person who makes and gives orders (the sovereign), or one follows orders. There is no room in this picture for exactly the kind of distributed sovereignty on which democracy itself would seem to be predicated" (*Cultural Logic of Computation*, 224).

130. See "Anecdotes: How Did You First Get into Computing?" *IEEE Annals of the History of Computing* 25, no. 4 (October–December 2003): 48–59.

131. Although not addressed here, there has always been tension between business and academic computing.

132. Frederick P. Brooks, while responding to the disaster that was OS/360, also emphasizes the magical powers of programming. Describing the joys of the craft, Brooks writes:

> Why is programming fun? What delights may its practitioner expect as his reward?
>
> First is the sheer joy of making things.
>
> Second is the pleasure of making things that are useful to other people.
>
> Third is the fascination of fashioning complex puzzle-like objects of interlocking moving parts and watching them work in subtle cycles, playing out the consequences of principles built in from the beginning.
>
> Fourth is the joy of always learning, which springs from the nonrepeating nature of the task.
>
> Finally there is the delight of working in such a tractable medium. The programmer, like the poet, works only slightly removed from thought-stuff. He builds his castles in the air, from air, creating by exertion of the imagination. . . . Yet the program construct, unlike the poet's words, is real in the sense that it moves and works, producing visible outputs separate from the construct itself. It prints results, draws pictures, produces sounds, moves arms. The magic of myth and legend has come true in our time. One types the correct incantation on a keyboard, and a display screen comes to life, showing things that never were nor could be. (Brooks, *Mythical Man-Month*, 7–8).

133. Paul Edwards, "The Army and the Microworld: Computers and the Politics of Gender Identity," *Signs* 16, no. 1 (Autumn 1990): 108–109.

134. Weizenbaum, *Computer Power and Human Reason*, 115; italics in original.

135. Foucault, *Birth of Biopolitics*, 120.

136. Indeed, he cites Horkheimer's critique:

> Concepts have been reduced to summaries of the characteristics that several specimens have in common. By denoting similarity, concepts eliminate the bother of enumerating qualities and thus serve better to organize

the material of knowledge. They are thought of as mere abbreviations of the items to which they refer. Any use transcending auxiliary, technical summarization of factual data has been eliminated as a last trace of superstition. Concepts have become "streamlined," rationalized, labor-saving devices . . . thinking itself [has] been reduced to the level of industrial processes . . . in short, made part and parcel of production." (Weizenbaum, *Computer Power and Human Reason*, 249)

And he argues that this critique applies directly to programming languages: "no one who does not know the technical basis of the systems we have been discussing can possibly appreciate what a chillingly accurate account of them this passage is. It was written by the philosopher-sociologist Max Horkheimer in 1947, years before the forces that were even then eclipsing reason, to use Horkheimer's own expression, came to be embodied literally in machines . . . As we see so clearly in the various systems under scrutiny, meaning has become entirely transformed into function" (Weizenbaum, *Computer Power and Human Reason*, 250).

137. Ibid., 255.

138. Brooks, *Mythical Man-Month*, 8.

139. Weizenbaum, *Computer Power and Human Reason*, 277.

140. Ibid., 115.

141. Ibid., 118.

142. Ibid., 124, 122, 121.

143. Ibid., 119.

144. For more on the relationship between paranoia and knowledge, rather than truth, see Wendy Hui Kyong Chun, *Control and Freedom: Power and Paranoia in the Age of Fiber Optics* (Cambridge, Mass.: MIT Press, 2006).

145. Ibid., 116.

146. Linus Torvalds and David Diamond, *Just for Fun: The Story of an Accidental Revolutionary* (New York: HarperCollins, 2001), 73.

147. *The Oxford English Online Dictionary* (Oxford and New York: Oxford University Press, 1992), <http://dictionary.oed.com/entrance.dtl>, accessed 3/1/2007.

148. See Friedrich Kittler, "There is No Software," *Ctheory* (1995), <http://www.ctheory.net/articles.aspx?id=74>, accessed 6/1/2008.

149. Namely, twentieth-century genetics, but this is the topic of chapter 3.

150. (Interview with) Felix Guattari and Gilles Deleuze, "Capitalism: A Very Special Delirium," in "Chaosophy," ed. Sylvere Lothringer, *Autonomedia/Semiotexte* (1995), <http://www.generation-online.org/p/fpdeleuze7.htm>, accessed 8/1/2009.

151. *The Oxford English Dictionary* (OED), 2nd ed., S.V. "fetish, *n.*"

152. William Pietz, "Fetishism and Materialism," in *Fetishism as Cultural Discourse*, ed. Apter and Pietz, 138, 139.

153. Ibid., 137.

154. Karl Marx, *Capital: A Critique of Political Economy*, *Volume One*, trans. Ben Fowkes (New York: Penguin Books in association with *New Left Review* 1990–1992, 1976), 165.

155. Marx as quoted by Pietz, "Fetishism and Materialism," 149.

156. Richard Stallman, "Copyright and Globalization in the Age of Computer Networks" (2001), <http://www.gnu.org/philosophy/copyright-and-globalization.html>, accessed 6/1/2008.

157. Ellen Ullman interviewed by Scott Rosenberg, "21st: Elegance and Entropy," *Salon.com*, <http://dir.salon.com/story/tech/feature/1997/10/09/interview/>, accessed 6/1/2008.

158. See Mez's site at <http://www.hotkey.net.au/~netwurker/>, accessed 6/1/2008.

159. See <http://www.scotoma.org/notes/index.cgi?LondonPL>, accessed 6/1/2008.

160. Sigmund Freud, "Fetishism," in *Sexuality and the Psychology of Love*, ed. Philip Rieff (New York: Collier Books, 1963), 205. Freud also writes: "It is not true that the child emerges from his experience of seeing the female parts with an unchanged belief in the woman having a phallus. He retains this belief but he also gives it up; during the conflict between the deadweight of the unwelcome perception and the force of the opposite wish, a compromise is constructed such as is only possible in the realm of unconscious thought—by the primary processes. In the world of psychical reality the woman still has a penis in spite of it all, but this penis is no longer the same as it once was. Something else has taken its place, has been appointed as its successor, so to speak, and now absorbs all the interest which formerly belonged to the penis" (Ibid., 206).

161. William Pietz, "The Problem of the Fetish," pt. 1, *Res: Anthropology and Aesthetics* 9 (Spring 1985): 12.

162. Freud, "Fetishism," 208.

163. Slavoj Žižek, *The Sublime Object of Ideology* (London and New York: Verso, 1989), 31.

164. Alan Turing, "Computing Machinery and Intelligence," *Mind* 59 (1950), <http://www.loebner.net/Prizef/TuringArticle.html>, accessed 3/1/2006.

165. N. Katherine Hayles develops this theme of revealing codes in *My Mother Was a Computer*, 54–61. Importantly, some software art projects also complicate and frustrate code as Xray vision and connection as meaning, such as Golan Levin's *AxisApplet* produced for the Whitney Artport CODeDoc project.

166. Keenan, "The Point Is to (Ex)Change It," 173.

Computers that Roar

1. See John von Neumann, "The First Draft Report on the EDVAC," *Contract No. W–670–ORD–4926* between the U.S. Ordnance (USORD) and the University of Pennsylvania, June 30, 1945, <http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf>, accessed 9/7/2009; and

Jon Agar, *The Government Machine: A Revolutionary History of the Computer* (Cambridge, Mass.: MIT Press 2003), 391.

2. Paul N. Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America* (Cambridge, Mass.: MIT Press, 1997); David Golumbia, *The Cultural Logic of Computation* (Cambridge, Mass.: Harvard University Press, 2009); Joseph Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation* (San Francisco: W. H. Freeman, 1976), 157.

3. Weizenbaum, *Computer Power and Human Reason*, 157.

4. Aristotle as quoted and translated by Paul Ricoeur, *The Rule of Metaphor* (London and New York: Routledge), 13.

5. George Lakoff and Mark Johnson, *Metaphors We Live By* (Chicago: University of Chicago, 1980), 5; emphasis in original.

6. Ibid., 115.

7. They write, "Our ordinary conceptual system, in terms of which we both think and act, is fundamentally metaphorical in nature. The concepts that govern our thought are not just matters of the intellect. They also govern our everyday functioning down to the most mundane details. Our concepts structure what we perceive, how we get around in the world, and how we relate to other people. Our conceptual system thus plays a central role in defining our everyday realities" (Ibid., 3).

8. Ibid., 119.

9. Ibid., 144.

10. Ibid., 239.

11. Ibid., 193.

12. Thomas Keenan, "The Point Is to (Ex)Change It: Reading *Capital* Rhetorically," in *Fetishism as Cultural Discourse*, ed. Emily Apter and William Pietz (Ithaca, N.Y.: Cornell University Press, 1993), 157.

13. Roman Jackobson of course defined *metaphor* in terms of selection and combination—the ability to think through substitutions in "Two Aspects of Language and Two Types of Aphasic Disturbances," *Fundamentals of Language*, 2nd rev. ed. (The Hague: Mouton, 1971).

14. Ricoeur, *Rule of Metaphor*, 23.

15. Ibid., 23–24.

16. Ibid., 44. He writes, "If metaphor belongs to an heuristic of thought, could we not imagine that the process that disturbs and displaces a certain logical order, a certain conceptual hierarchy, a certain classification scheme, is the same as that from which all classification proceeds? . . . could we not imagine that the order itself is born in the same way that it changes? Is there not,

in Gadamer's terms, a 'metaphoric' at work at the origin of logical thought, at the root of all classification? . . . The idea of an initial metaphorical impulse destroys the[se] oppositions between proper and figurative, ordinary and strange, order and transgression. It suggests the idea that order itself proceeds from the metaphorical constitution of semantic fields, which themselves give rise to genus and species" (Ibid., 24).

17. Ibid., 37.

18. Ibid., 48.

2 Daemonic Interfaces, Empowering Obfuscation

1. Luc Boltanski and Eve Chiapello have outlined this encyclopedically in *The New Spirit of Capitalism*, trans. Gregory Elliott (London: Verso, 2005).

2. *Oxford English Dictionary Online*, June 2010, Draft Entry, S.V. "daemon, *n*," emphasis in original.

3. Paul Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America* (Cambridge, Mass.: MIT Press, 1996), 111, 107.

4. Ibid., 12.

5. Ibid., 13.

6. See David Mindell, *Between Human and Machine: Feedback, Control, and Computing before Cybernetics* (Baltimore, Md.: Johns Hopkins University Press, 2002).

7. See Gordon Bell, "Toward a History of (Personal) Workstations," in *A History of Personal Workstations*, ed. Adele Goldberg (New York: ACM Press; Reading, Mass.: Addison-Wesley Pub. Co., 1988), 29; Martin Campbell-Kelly and William Aspray, *Computer: A History of the Information Machine* (Boulder, Colo.: Westview Press, 2004), 168.

8. As quoted by Edwards, *Closed World*, 258.

9. John von Neumann, *Papers of John von Neumann on Computing and Computer Theory,* ed. William Aspray and Arthur Burks (Cambridge, Mass.: MIT Press, 1987), 413.

10. J. C. R. Licklider, "Man-Computer Symbiosis," in *NewMediaReader*, ed. Noah Wardrip-Fruin and Nick Montfort (Cambridge, Mass.: MIT Press, 2003), 75.

11. As quoted in Joseph Weizenbaum, *Computer Power and Human Reason: From Judgment to Calculation* (San Francisco: W. H. Freeman and Company, 1976), 246.

12. Ben Shneiderman, "Direct Manipulation: A Step Beyond Programming Languages," in *NewMediaReader,* ed. Wardrip-Fruin and Montfort, 486.

13. Michel Foucault, *The Birth of Biopolitics: Lectures at the Collège de France, 1978–1979*, trans. Grahm Burchell (Basingstoke, England, and New York: Palgrave Macmillan, 2008), 252.

14. Boltanski and Chiapello, *New Spirit of Capitalism*, 92.

15. Catherine Malabou, *What Should We Do with Our Brains*, trans. Sebastian Rand (New York: Fordham University Press, 2008), 44.

16. Ibid., 51.

17. George Lakoff and Mark Johnson, *Metaphors We Live By* (Chicago: Chicago University Press, 1980), 70. Ben Shneiderman also directly acknowledges Piaget among others in "Direct Manipulation," 492–493.

18. Lakoff and Johnson, *Metaphors We Live By*, 123.

19. Shneiderman, "Direct Manipulation," 491.

20. Indeed, Laurel argues in *Computers as Theater* (Reading, Mass.: Addison-Wesley Publishers, 1991) that interface metaphors are "dangerous" because they are always doomed to fail: they are similes rather than metaphors, like reality only different (129). This difference can become overwhelming for the user—the interface can become a visible impediment—because the interface refers to the wrong thing: to reality rather than to compelling causality or to action (131).

21. Ibid., xviii.

22. Ibid., 77.

23. Ibid., 33.

24. Ibid., 6; emphasis in original.

25. Edwards draws from Sherman Hawkins's use of the term "to define one of the major dramatic spaces in Shakespearean plays. Closed-world plays are marked by a unity of place, such as a walled city or the interior of a castle or house. Action within this space centers [on] attempts to invade and/or escape its boundaries. Its archetype is the siege , , , the closed world includes not just the sealed, claustrophobic spaces metaphorically marking its closure, but the entire surrounding field in which the drama takes place" (Edwards, *Closed World*, 12–13).

26. She contends, "There are ways in which art is "lawful"; that is, there are formal, structured, and causal dimensions that can be identified and used both descriptively and productively" (Laurel, *Computers as Theater*, 28).

27. Ibid., 67.

28. Ibid., 62.

29. Ibid., 101.

30. Ibid., 167.

31. See Wendy Hui Kyong Chun, *Control and Freedom: Power and Paranoia in the Age of Fiber Optics* (Cambridge, Mass.: MIT Press, 2006).

32. Louis Althusser, "Ideology and Ideological State Apparatuses (Notes Towards an Investigation)," in *Lenin and Philosophy and Other Essays,* trans. Ben Brewster (New York: Monthly Review Press, 2001), 109.

33. Ibid., 171 (emphasis in original).

34. See Slavoj Žižek, *The Sublime Object of Ideology* (London and New York: Verso, 1989), 11–53.

35. See Ceruzzi, *A History of Modern Computing*, 208–209; Martin Campbell-Kelly, *From Airline Reservations to Sonic the Hedgehog* (Cambridge, Mass.: MIT Press), 207–229.

36. See Thomas Y. Levin, "Rhetoric of the Temporal Index: Surveillant Narration and the Cinema of 'Real Time,'" in *CTRL Space: Rhetorics of Surveillance from Bentham to Big Brother*, ed. Thomas Y. Levin et al. (Cambridge, Mass.: MIT Press, 2002), 578–593.

37. Tara McPherson, "Reload: Liveness, Mobility and the Web," in *The Visual Culture Reader*, 2nd ed., ed. Nicholas Mirzoeff (London and New York: Routledge, 2002), 461–462.

38. Coming from film rather than from television studies and focusing more on applications than on phenomenology, Alexander Galloway in *Protocol: How Power Exists after Decentralization* (Cambridge, Mass.: MIT Press, 2004) similarly argues that continuity makes websurfing "a compelling, intuitive experience for the user":

> On the Web, the browser's movement is experienced as the user's movement. The mouse movement is substituted for the user's movement. The user looks through the screen into an imaginary world, and it makes sense. The act of "surfing the web," which, phenomenologically, should be an unnerving experience of radical dislocation—passing from a server in one city to a server in another city—could not be more pleasurable for the user. Legions of computer users live and play online with no sense of radical dislocation. (64)

39. Lev Manovich, "Generation Flash," 2002, <http://www.manovich.net/DOCS/generation_flash.doc>, accessed 8/8/2010.

40. Julian Dibbell, *My Tiny Life: Crime and Passion in a Virtual World* (New York: Holt, 1998).

41. Manovich, "Generation Flash," 2002.

42. Fredric Jameson, *Postmodernism, or the Cultural Logic of Late Capitalism* (Durham, N.C.: Duke University Press, 1991), 51.

43. However, these parallels arguably reveal the fact that our understandings of ideology are lacking precisely to the extent that they, like interfaces, rely on a fundamentally theatrical model of behavior.

44. This argument thus seeks to complicate Matthew Kirschenbaum's insightful critique in *Mechanisms: New Media and the Forensic Imagination* (Cambridge, Mass.: MIT Press, 2008) of "medial ideology"—the acceptance of the interface as the computer—rampant in new media studies (36–45). This medial ideology is so attractive not simply because we are enamored by the flickering signifiers on our screens, but also because interfaces offer us a way to "map" our larger relation to the world, to ideology itself.

45. See Jean François Lyotard, *The Postmodern Condition: A Report on Knowledge* (Minneapolis: University of Minnesota Press, 1984).

46. Jameson, *Postmodernism*, ix.

47. Ibid., x. Formally, Jameson argues, postmodern art—which erodes the barrier between high and low culture—has five characteristics: (1) a new depthlessness, in which inimitable styles are turned into surface characteristics (postmodern codes, blank parodies or "pastiche"); (2) the waning of affect (the waning of the subject and of the difference between inside and outside); (3) a weakening of historicity (of the relationship between past, present, and future); (4) a change in tone (toward the sublime); and (5) "a constitutive relationship of all this to a whole new technology, which is itself a reflection, or a way to deal with a whole new economic world" (*Postmodernism*, 6).

Following from Jameson's description of postmodernism—in particular from his likening of postmodernism to pastiche and his view of technology as reflection—Lev Manovich, in *The Language of New Media* (Cambridge, Mass.: MIT Press, 2001), has argued that GUIs are quintessentially postmodern. It is no accident, he writes, that the GUI, "which legitimized a 'cut and paste' logic, as well as media manipulation software such as Photoshop, which popularized a plug-in architecture, took place during the 1980s—the same decade when contemporary culture became 'postmodern'" (131). The GUI's cut-and-paste logic, Manovich argues, is emblematic of a culture that

> no longer tried to "make it new." Rather, endless recycling and quoting of past media content, artistic styles, and forms became the new "international style" and the new cultural logic of modern society. Rather than assembling more media recordings of reality, culture is now busy reworking, recombining, and analyzing already accumulated media material. Invoking the metaphor of Plato's cave, Jameson writes that postmodern cultural production "can no longer look directly out of its eyes at the real world but must, as in Plato's cave, trace its mental images of the world on its confining walls." In my view, this new cultural condition found its perfect reflection in the emerging computer software of the 1980s that privileged selection from ready-made elements over creating them from scratch. And to a large extent it is this software that in fact made postmodernism possible. (Ibid.)

Manovich's application of Jameson's diagnosis of postmodernism to computer interfaces is intriguing, although, given Lyotard's linking of postmodernism to new creative acts, arguably one-sided and reductive. Lyotard links postmodernism to paralogy and the sublime—a challenge to totalitarianism. More important, however, by focusing on formal likenesses, Manovich misses the larger point: rather than a symptom or a cause, GUIs, which emerged as a common cultural object much later than Jameson's and Lyotard's initial descriptions of postmodernism in the 1970s, are a *response* to the challenges of postmodernism, to the spatial challenges it posed. Interfaces, that is, are ways to navigate postmodern confusion.

48. Fredric Jameson, "Cognitive Mapping," in *Marxism and the Interpretation of Culture*, ed. Cary Nelson and Lawrence Grossberg (Champaign: University of Illinois Press, 1988), 351.

49. Ibid., 351.

50. Jameson, *Postmodernism*, 39.

51. It stems from an earlier disorientation, brought about by global imperialism, in which the truth of a subject's experience

> no longer coincides with the place in which it takes place. The truth of that limited daily experience of London lies, rather, in India or Jamaica or Hong Kong; it is bound up with the whole colonial system of the British Empire. . . . Yet those structural coordinates are no longer accessible to immediate lived experience and are often not even conceptualizable for most people.
>
> There comes into being, then, a situation in which we can say that if individual experience is authentic, then it cannot be true, and that if a scientific or cognitive model of the same content is true, then it escapes individual experience. (Jameson, "Cognitive Mapping," 349)

52. Ibid., 353.

53. Jameson argues, "the conception of capital is admittedly a totalizing or systemic concept: no one has ever seen or met the thing itself; it is either the result of scientific reduction (and it should be obvious that scientific thinking always reduces the multiplicity of the real to a small-scale model) or the mark of an imaginary and ideological vision" ("Cognitive Mapping," 354).

54. Ibid., 356.

55. Jameson, *Postmodernism*, 37.

56. Ibid., 54.

57. For an excellent analysis of free labor, see Tiziana Terranova's *Network Culture: Politics for the Information Age* (London: Pluto Press, 2004).

58. David Harvey, *A Brief History of Neoliberalism* (Oxford: Oxford University Press, 2005), 3.

59. See Foucault, *Birth of Biopolitics*, 280.

60. David Mindell similarly argues, "Our computers retain traces of earlier technologies, from telephones and mechanical analogs to directorscopes and tracking radars. . . . When we articulate a mouse to direct a machine, do we not resemble Sperry's pointer-matching human servomechanisms? When we interpret glowing images and filter out signals from noise, do we not resemble a pip-matching radar operator?" See Mindell, *Between Human and Machine*, 321.

61. See Linda C. Smith, "Memex as an Image of Potentiality Revisited," in *From Memex to Hypertext: Vannevar Bush and the Mind's Machine*, ed. James M. Nyce and Paul Kahn (Boston: Academic Press, 1991), 261–286.

62. Indeed, Vannevar Bush deliberately contrasts the memex to expensive digital computers in "Memex Revisited," in *New Media, Old Media*, ed. Wendy Hui Kyong Chun and Thomas Keenan (New York: Routledge, 2006), 86–96.

63. Vannevar Bush, "As We May Think," *The Atlantic* (July 1945), <http://www.theatlantic.com/doc/194507/bush>, accessed 9/9/2009.

64. Vannevar Bush, "Memorandum Regarding Memex," *From Memex to Hypertext,* 81.

65. Bush, "As We May Think," n.p.

66. For Nelson and Engelbart, the "gadgetry" Bush envisions guarantees freedom. Nelson, in his "As We Will Think" (in *From Memex to Hypertext: Vannevar Bush and the Mind's Machine*, ed. James M. Nyce and Paul Kahn [Boston: Academic Press, 1991]) pinpoints Bush's notion of a "trail" as hypertext, where hypertext more generally means a "text structure that cannot be conveniently printed" (Ibid., 253). Hypertext, Nelson stresses, was to liberate human thinking: "Let me suggest that such an object and system [hypertext], properly designed and administered, could have great potential for education, increasing the student's range of choices, his sense of freedom, his motivation, and his intellectual grasp." See Theodor Nelson, "A File Structure for the Complex, the Changing, and the Indeterminate," in *NewMediaReader*, ed. Wardrip-Fruin and Montfort, 144). Engelbart, focusing on the manipulation of symbols, argues, in relation to his own system, "you are quite elated by this freedom to juggle your thoughts, and by the way this freedom allows you to *work* them into shape." See Douglas C. Engelbart, "Augmenting Human Intellect: A Conceptual Framework" (October 1962), <http://www.dougengelbart.org/pubs/augment-3906.html>, accessed 8/8/2009; emphasis in original. Bush himself contends that *the* problem hindering scientists and scientific progress is access: man must "mechanize his records more fully if he is to push his experiment [human civilization] to its logical conclusion and not merely become bogged down part way there by overtaxing his limited memory" ("As We May Think," section 8, n.p.).

67. Bush, "As We May Think," section 1, n.p.

68. Ibid.

69. Bush, "Memex Revisited," 85.

70. Ibid., 91, 93, 94.

71. Bush, "As We May Think," section 6, n.p.

72. Ibid., sections 6, 8, n.p.

73. Ibid., section 4, n.p.; Bush, "Memex Revisited," 85.

74. This is Foucault's diagnosis of traditional history in *The Archaeology of Knowledge & The Discourse on Language*, trans. A. M. Sheridan Smith (New York: Pantheon Books, 1982).

75. Noah Wardrip-Fruin, "Introduction to 'As We May Think,'" in *NewMediaReader*, ed. Wardrip-Fruin and Montfort, 35; emphasis in original.

76. Jann Sapp, "The Nine Lives of Gregor Mendel," *Experimental Inquiries: Historical, Philosophical, and Social Studies of Experimentation in Science*, ed. H. E. Le Grand (Dordrecht, The Netherlands: Kluwer Academic Publishers, 1990), 137–166.

77. See Jacques Derrida, *Archive Fever: A Freudian Impression*, trans. Eric Prenowitz (Chicago: University of Chicago Press, 1996). The pleasure of forgetfulness is to some extent the pleasure of death and destruction. It is thus no accident that this "supplementing" of human memory has also been imagined as the death of the human species in so many fictions and films, and that déjà vu is the mark of the artificial in the Wachowski brothers' film *The Matrix*.

78. Douglas C. Engelbart, "Letter to Vannevar Bush and Program On Human Effectiveness," *From Memex to Hypertext*, 236.

79. Douglas C. Engelbart, "The Augmented Knowledge Workshop," in *Proceedings of the ACM Conference on the History of Personal Workstations* (New York: ACM Press, 1986), 74.

80. Engelbart, "II: Conceptual Framework," in "Augmenting Human Intellect," n. p.

81. Engelbart, "1.A, General Introduction," in "Augmenting Human Intellect," n. p.

82. Engelbart, "II: Conceptual Framework," n. p.

83. As quoted by Thierry Bardini, in *Bootstrapping: Douglas Engelbart, Coevolution, and the Origins of Personal Computing* (Stanford, Calif.: Stanford University Press, 2000), 18.

84. Ibid., 19.

85. Engelbart, "Hypothetical Description of Computer-Based Augmentation System," in "Augmenting Human Intellect," n. p.

86. For more on this see Foucault, *Birth of Biopolitics*, 243–246. See also Philip Agre's discussion of capture as making more tasks intelligible as market transactions in "Surveillance and Capture: Two Models of Privacy," *The Information Society* 10, no. 2 (1994): 101–127.

87. For more on this, see Jane Feuer, "The Concept of Live Television: Ontology as Ideology," in *Regarding Television: Critical Approaches*, ed. E. Ann Kaplan (Washington, D.C.: University Press of America, 1983), 12–22.

88. Cornelia Vismann, *Files: Law and Media Technology*, trans. Geoffrey Winthrop-Young (Stanford, Calif.: Stanford University Press, 2008), 6.

89. Ibid., 7.

90. Ibid.,10.

91. Ibid., 163.

92. The following PERL program, for instance, says hello every 5 minutes:

```
use POSIX qw(setsid);

#turns the process into a session leader, group leader, and ensures that it doesn't #have a controlling terminal

chdir '/'                 or die "Can't chdir to /: $!";

umask 0;

open STDIN, '/dev/null' or die "Can't read /dev/null: $!";

#open STDOUT, '>/dev/null' or die "Can't write to /dev/null: $!";

open STDERR, '>/dev/null' or die "Can't write to /dev/null: $!";

defined(my $pid = fork) or die "Can't fork: $!";
```

```
exit if $pid;
setsid                  or die “Can’t start a new session: $!”;
while(1) {
sleep(5);
print “Hello...\n”;
}
```

From <http://www.webreference.com/perl/tutorial/9/3.html>, accessed 9/1/2008.

93. Fernando J. Corbato, as quoted in “The Origin of the Word Daemon,” from Richard Steinberg/Mr. Smarty Pants, *The Austin Chronicle*, <http://ei.cs.vt.edu/~history/Daemon.html>, accessed 3/1/2007. This is why Neal Stephenson in *Snow Crash* (New York: Bantam Books, 1993) describes robots or servants in the Metaverse as daemons.

94. This resonates with Derrida’s analysis of writing, in contrast to “living logos,” as “orphaned” in “Plato’s Pharmacy,” *Dissemination*, trans. Barbara Johnson (Chicago: University of Chicago, 1981), 76.

95. John Schwartz, “Privacy Fears Erode Support for a Network to Fight Crime,” *New York Times*, March 15, 2004, <http://www.nytimes.com/2004/03/15/technology/15matrix.html?pagewanted=1>, accessed 4/15/2004.

96. Ibid.

97. Stephanie Clifford, “Cable Companies Target Commercials to Audience,” *New York Times*, March 3, 2009, <http://www.nytimes.com/2009/03/04/business/04cable.html>, accessed 9/9/2009.

98. “‘Personal data’ in iTunes tracks,” *BBC News Online*, June 1, 2007, <http://news.bbc.co.uk/2/hi/technology/6711215.stm>, accessed 8/1/2008.

99. Manovich, *Language of New Media*, 48.

100. Microsoft is considering such actions in its Palladium initiative.

101. See Friedrich Kittler, “There Is No Software,” *Ctheory.net*, October 18, 1995, <http://www.ctheory.net/articles.aspx?id=74>, accessed 8/8/2010.

102. See Francois Jacob, *The Logic of Life: A History of Heredity,* trans. Betty E. Spillman (New York: Pantheon Books, 1973), 247–298.

103. Richard Doyle’s concept of “rhetorical software,” developed in his *On Beyond Living: Rhetorical Transformations of the Life Sciences* (Stanford, Calif.: Stanford University Press, 1997), is emblematic of the use of *software* as a critical term in nonscientific scholarly discourse.

104. Bruce Schneier, “U.S. Enables Chinese Hacking of Google,” *CNN Opinion Online*, <http://www.cnn.com/2010/OPINION/01/23/schneier.google.hacking/index.htm>, accessed 3/28/2010.

105. Ibid.

106. "Government Information Awareness," *SourceWatch.org*, <http://www.sourcewatch.org/index.php?title=Government_Information_Awareness>, accessed 3/28/2010.

107. Google, "Explore Flu Trends Around the World," <http://www.google.org/flutrends/>, accessed June 24, 2009.

108. Adrian Mackenzie, *Cutting Code: Software and Sociality* (New York: Peter Lang Pub. Inc., 2006), 169.

109. Milton Friedman, *Capitalism and Freedom*, Fortieth Anniversary Edition (Chicago: Chicago University Press, 2002), 30.

110. Friedrich Nietzsche, "On Truth and Lie in an Extra-Moral Sense" (1873), from the *Nachlass*, trans. Walter Kaufmann and Daniel Breazeale <www.geocities.com/thenietzschechannel/tls.htm>, accessed 9/3/2009.

II Regenerating Archives

1. Wolfgang Ernst, "Art of the Archive," *Künstler.Archiv—Neue Werke zu historischen Beständen*, ed. Helen Adkins (Köln: Walter König, 2005), 99.

2. Jacques Derrida, "Freud and the Scene of Writing," *Writing and Difference*, trans. Alan Bass (Chicago: University of Chicago, 1978), 226.

3. Execution, as this book has been arguing, is arguably a more critical category, but it is constantly overlooked in favor of memory.

4. The move from calculator to computer is also the move from mere machine to human-emulator: IBM initially resisted the term *computer* because computers initially were human. To call a machine a computer implied job redundancy (Martin Campbell-Kelly and William Aspray, *Computer: A History of the Information Machine* (New York: Basic Books, 1996), 115). John von Neumann, in his mythic and controversial *The First Draft Report of the EDVAC* (1945) deliberately used the term *memory organ* rather than *store*, also in use at the time, in order to parallel biological and computing components and to emphasize the ephemeral nature of vacuum tubes. (See von Neumann, "First Draft of a Report on the EDVAC," <www.cs.colorado.edu/~zathras/csci3155/EDVAC_vonNeumann.pdf>, accessed 9/12/2003). Vacuum tubes, unlike mechanical switches, can hold values precisely because their signals can degenerate—and thus regenerate.

5. For more on this compression see David Harvey, *A Short History of Neoliberalism* (Oxford: Oxford University Press, 2005, 7).

6. See Ryan Lizza, "The YouTube Election," *New York Times*, August 20, 2006, <http://www.nytimes.com/2006/08/20/weekinreview/20lizza.html?ex=1313726400&en=a605fabfcb81eebf&ei=5088&partner=rssnyt&emc=rss>, accessed 9/1/2009. In this video clip Senator George Allen referred to an Indian American man as a "macacca" while campaigning in Virginia.

7. For more on this ideal and its incapacity to explain public behavior, see Thomas Keenan's "Publicity and Indifference (Sarajevo on Television)," *PLMA* 111, no. 1 (January 2002): 104–116.

8. Jacques Derrida, *Archive Fever: A Freudian Impression*, trans. Eric Prenowitz (Chicago: Chicago University Press, 1998), 11–12.

9. Derrida, *Archive Fever*, 84.

10. Derrida, "Freud and the Scene of Writing," 228.

11. Howard Caygill, "Meno and the Internet: Between Memory and the Archive," *History of the Human Sciences* 12 (1999): 2.

12. Derrida, *Archive Fever*, 36.

13. Ibid.

14. Ibid., 4n1.

15. For the relationship between threat and promise see Jacques Derrida, "Typewriter Ribbon," *Without Alibi* (Palo Alto, Calif.: Stanford University Press, 2002), 155.

16. Ibid., 7.

17. Ibid., 1; emphasis in original.

18. According to Ernst, archives are comprised of fundamentally discontinuous units or fragments—they are not histories (stories), but rather "discrete, isolated units and islands of discourse" that are "generally smoothed over in narrative representations based on archive research" (Ernst, "Art of the Archive," 94). These discrete units do not simply lie about, but rather are ordered—the archive "is a metonymic device that 'formalizes' experience"; "the art of the archive entails assigning names to documents—to which stories immediately adhere" (Ibid., 94, 96). In contrast to this metonymic linguistic ordering based on names and categories, digital media offers the possibility of searching based on textual and pattern-based similarities, disrupting archival ordering. This disruption undermines a fundamental function of the archive: the separation of documents into discrete units; see Wolfgang Ernst, "Dis/continuities: Does the Archive Become Metaphorical in Multi-Media Space?," in *New Media, Old Media: A History and Theory Reader*, ed. Wendy Hui Kyong Chun and Thomas Keenan (New York: Routledge, 2006), 119. Rather than an archive, one has something like a "life stream"—a time based medium that is only superficially comprised of documents. In addition, cyberspace has no memory: "the archival phantasms in cyberspace are an ideological deflection of the sudden erasure of archives (both hard- and software) in the digital world. . . . The Internet has no organized memory and no central agency, being defined rather by the circulation of discrete states. If there is memory, it operates as a radical constructivism: always just situationally built, with no enduring storage" (Ibid., 119). This constant flow and information in general, whose value is linked to entropy rather than order technically redeems (wipes out) the archive (Ibid., 97). "The archive is a given, well-defined-lot; the Internet, on the contrary, is not just a collection of unforeseen texts, but of sound and images as well, an archive of sensory data for which no genuine archival culture has been developed so far

in the occident. . . . What separates the Internet from the classical archive is that its mnemonic logic is more dynamic than cultural memory in the printed archive" (Ibid., 119–120).

3 Order from Order, or Life According to Software

1. Pierre-Simon Laplace, *A Philosophical Essay on Probabilities* (1820; reprinted, New York: Dover, 1951), preface.

2. Gilles Deleuze and Félix Guattari, *A Thousand Plateaus: Capitalism and Schizophrenia*, trans. Brian Massumi (Minneapolis: University of Minnesota Press, 1987), 143.

3. See Paul Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America* (Cambridge, Mass.: MIT Press, 1996); and Richard Boyd, "Metaphor and Theory Change: What Is a 'Metaphor' a Metaphor For?," in *Metaphor and Thought*, ed. Andrew Ortony (Cambridge: Cambridge University Press, 1993), chap. 21.

4. Richard Dawkins, *The Selfish Gene*, 2nd ed. (Oxford: Oxford University Press, 1989), v.

5. Ibid., 49.

6. François Jacob, *The Logic of Life* (New York: Pantheon, 1982), 2.

7. John von Neumann, "First Draft of a Report on the EDVAC," Contract No. W-670-ORD-4926 between the United States Army Ordnance Department and the University of Pennsylvania, June 30, 1945, <http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf>, 3, accessed 8/8/2010.

8. Herman Goldstine and John von Neumann, "Planning and Coding of Problems for an Electronic Computer Instrument," "Report on the Mathematical and Logical Aspects of an Electronic Computing Instrument," Part II, Volume I (Princeton, N.J.: Institute for Advanced Study, 1947).

9. The other factors, such as clerical and gendered bureaucratic relations, and Fordist industrial techniques, are the subjects of the other chapters.

10. Jacob, *Logic of Life*, 264–265.

11. Ibid., 1.

12. Lily Kay, *Who Wrote the Book of Life? A History of the Genetic Code* (Stanford, Calif.: Stanford University Press, 2000), xv.

13. Ibid., 85.

14. Richard Doyle, *On Beyond Living: Rhetorical Transformations of the Life Sciences* (Stanford, Calif.: Stanford University Press, 1997), 1.

15. Ibid., 13.

16. Ibid., 3.

17. Ibid., 2.

18. Ibid., 6–7.

19. David Mindell in *Between Human and Machine: Feedback, Control, and Computing Before Cybernetics* (Baltimore: Johns Hopkins University Press, 2002) argues convincingly against this claim, emphasizing the importance of electrical engineering to the development of cybernetics.

20. Norbert Wiener, *The Human Use of Human Beings: Cybernetics and Society* (Garden City, N.Y.: Doubleday, 1954), 27; emphasis in original.

21. Wiener explains, "Information is a name for the content of what is exchanged with the outer world as we adjust to it, and make our adjustment felt upon it. The process of receiving and of using information is the process of our adjusting to the contingencies of the outer environment, and of our living effectively within that environment" (*Human Use of Human Beings*, 17–18).

22. On the importance of the military, see Heinz von Foerster, "Epistemology of Communication," in *The Myths of Information: Technology and Postindustrial Culture*, ed. Kathleen Woodward (Madison, Wisc.: Coda Press, 1980), 18–27.

23. Norbert Wiener, *Cybernetics, or, Control and Communication in the Animal and the Machine* (Cambridge, Mass.: Technology Press, 1948), 118. Paul Edwards in *The Closed World* similarly distinguishes between cybernetic and artificial intelligence in terms of hardware versus software as the means for modeling brains (239).

24. Warren Weaver, "Some Recent Contributions," in Claude E. Shannon and Warren Weaver, *The Mathematical Theory of Communication* (Urbana: University of Illinois Press, 1963), 18.

25. Claude Shannon, "The Mathematical Theory of Communication," in Shannon and Weaver, *Mathematical Theory of Communication*, 31; emphasis in original.

26. Jacob, *Logic of Life*, 79, 207.

27. Ibid., 298.

28. Richard Panek, *The Invisible Century: Einstein, Freud, and the Search for Hidden Universes* (New York: Penguin, 2004); emphasis in original. Panek theorizes the invisible century in terms of the Freudian unconscious and Einsteinian relativity. In contrast, genetics and computer memory do not simply speculate about the invisible, they posit an invisible program that drives the visible world.

29. Erwin Schrödinger, *What Is Life?* with *Mind and Matter* and *Autobiographical Sketches* (Cambridge, UK: Cambridge University Press, 1992), 23; and Jacob, *Logic of Life*, 285.

30. Schrödinger, *What Is Life?*, 68–69.

31. Ibid., 4.

32. Ibid., 31.

33. Ibid., 69.

34. Ibid., 73–74.

35. Schrödinger himself rejected such a connection. See Kay, *Who Wrote the Book of Life?*, 64–65.

36. Kay, *Who Wrote the Book of Life?*, 21.

37. Schrödinger, *What Is Life?*, 23.

38. Schrödinger to E. I. Conway, October 25, 1942, quoted in E. J. Yoxen, "The Social Impact of Molecular Biology" (unpublished PhD thesis, Cambridge University, 1978), 152.

39. Schrödinger, *What Is Life?*, 23.

40. Ibid., 21–22. By making the code-script both pattern and development, Schrödinger possibly was rejecting the classical genetic separation of transmission from development, but since his code-script conflates development with the means of transmission, it hardly offers a serious biological consideration of development.

41. Kay, *Who Wrote the Book of Life?*, xvii.

42. Kay similarly argues that a poststructuralist view of writing sees writing as writing itself. My argument is slightly different: it's not the poststructuralist view but rather the view that writing and execution are conflated, although importantly, this formulation also gives great powers to anyone who controls the source.

43. Jacob, *Logic of Life*, 298.

44. François Jacob, *The Statue Within: An Autobiography* (New York: HarperCollins, 1988), 58.

45. Linus Pauling, "Schrödinger's Contributions to Chemistry and Biology," in *Schrödinger: Centenary Celebrations of a Polymath*, ed. C. W. Kilmister (Cambridge: Cambridge University Press, 1987), 225–233.

46. Kay argues, "Schrödinger's code-script was based on permutations in proteins, it neither related one system of symbols . . . to another . . . as did genetic codes after 1953, nor, most importantly did it claim to transfer information" (*Who Wrote the Book of Life?*, 61.)

47. Ibid., 161.

48. Gunther S. Stent, "Introduction: Waiting for the Paradox," in *Phage and the Origins of Molecular Biology*, ed. John Cairns et al. (Long Island, N.Y.: Cold Spring Harbor Laboratory of Quantitative Biology, 1966), 3.

49. Donald Fleming, "Émigré Physicists and the Biological Revolution," in *The Intellectual Migration*, ed. Donald Fleming and Bernard Bailyn (Cambridge, Mass.: Harvard University Press, 1969), 172.

50. Evelyn Fox Keller, "Physics and the Emergence of Molecular Biology: A History of Cognitive and Political Synergy," *Journal of the History of Biology* 23, no. 3 (Fall 1990): 390.

51. Leah Ceccarelli, *Shaping Science with Rhetoric: The Cases of Dobzhansky, Schrödinger, and Wilson* (Chicago: University of Chicago, 2001), 82–112.

52. Ibid., 67.

53. Doyle, *On Beyond Living*, 13.

54. Ibid., 28–29.

55. Ibid., 13; emphasis in original.

56. Ibid., 35. Not surprisingly, this part of Schrödinger's text has been mainly forgotten, since his hypothesis has proven so incorrect. What Schrödinger could not foresee is not only the importance of hydrogen bonds, but also the complete conflation of message with action, necessary for code-script as life.

57. Michel Foucault, *The Archaeology of Knowledge & The Discourse on Language*, trans. A. M. Sheridan Smith (New York: Pantheon Books, 1972), 129.

58. Ibid.; emphasis in original.

59. Michel Foucault, *The Order of Things: An Archaeology of the Human Sciences*, trans. A. M. Sheridan (New York: Vintage Books, 1994), ix.

60. Ibid., 131.

61. Ibid., 15.

62. Deleuze argues in *Foucault*, trans. Sean Hand (Minneapolis: University of Minnesota, 1988), "each age has its own particular way of putting language together, because of its different groupings" (56) and "each historical formation sees and reveals all it can within the conditions laid down for visibility, just as it says all it can within the conditions relating to statements" (59).

63. Foucault, *Archaeology of Knowledge*, 128, and *Order of Things*, xx.

64. Foucault, *Order of Things*, ix.

65. See Michel Foucault, "Two Lectures," in *Power/Knowledge: Selected Interviews and Other Writings, 1972–1977*, ed. Colin Gordon (New York: Pantheon, 1980), 78–108, and "What Is Critique," in *What Is Enlightenment? Eighteenth-Century Answers and Twentieth-Century Questions*, ed. James Schmidt (Berkeley, Calif.: University of California Press, 1996), 394–395.

66. Foucault, *Archaeology of Knowledge*, 106.

67. Ibid., 130.

68. See Walter Gilbert, "The RNA World," *Nature* 319 (February 1986): 618.

69. A. M. Turing, "Computing Machinery and Intelligence," *Mind* 59 (1950): 433–460.

70. Ibid., 440.

71. Kay, *Who Wrote the Book of Life?*, 106.

72. John von Neumann, "General and Logical Theory of Automata," in *Papers of John von Neumann on Computing and Computing Theory*, ed. William Aspray and Arthur Burks (Cambridge, Mass.: MIT Press, 1986), 42.

73. H. J. Muller, "Mutation," in *Eugenics, Genetics and the Family, Volume 1. Scientific Papers of the Second International Congress of Eugenics* (Baltimore, Md.: Williams & Wilkins Co., 1923), 106.

74. R. C. Punnett, *Mendelism* (London: Macmillan & Co., 1919).

75. Jan Sapp, "The Nine Lives of Gregor Mendel," in *Experimental Inquiries: Historical, Philosophical and Social Studies of Experimentation in Science*, ed. H. E. Le Grand (Dordrecht/Boston/London: Kluwer Academic Publishers, 1990), 138. The author/date citations included are as they appear in the chapter by Sapp.

76. Ibid., 141.

77. Ibid., 149.

78. Daniel J. Kevles, *In the Name of Eugenics: Genetics and the Uses of Human Heredity* (Cambridge, Mass.: Harvard University Press, 1998), 42.

79. Raphael Falk, "The Real Objective of Mendel's Paper: A Response to Monaghan and Corcos," *Biology and Philosophy* 6, no. 4 (1991): 448.

80. Punnett, *Mendelism,* 32.

81. Francis Galton, "Eugenics: Its Definition, Scope, and Aims," *The American Journal of Sociology* 10, no. 1 (July 1904): 1–6.

82. The British biometricians were more focused on the degeneration of the British race. It is telling that when Fisher changed the subtitle of the journal from "devoted to the genetic study of racial problems" to "devoted to the study of human populations," racial problems referred to problems in one's own race.

83. Kevles, *In the Name of Eugenics*, 9.

84. As quoted in Kevles, *In the Name of Eugenics*, 17.

85. Ronald Fischer, a population geneticist and eugenics advocate, is mainly heralded as breaching the differences between the biometricians and Mendelians. His "evolutionary synthesis" entailed proving that discontinuous changes were not as large as the early Mendelians believed (they were not "sports," which led to new species) and by showing that the continuous traits the biometricians tracked were actually comprised of multiple discrete ones.

86. See Raphael Falk, "What Is a Gene?" *Studies in the History and Philosophy of Science* 17, no. 2 (1986): 138.

87. The drive toward establishing genes as actual physical entities is attributed to Thomas Morgan's lab, in particular to the work of Hermann J. Muller. Morgan's lab, working with drosophila, showed that certain traits were sex-linked, that is, lay on the X or Y chromosome. For Muller—Falk argues in "What Is a Gene?"—genes were units in their own right with inherent characteristics, even if they could only be recognized by their effects. They were a "substance causing reproduction of its own specific composition, but can nevertheless mutate, and retain the property of reproducing itself in various new forms" (Falk, "What Is a Gene," 150). This

material view of the gene loosened the one-to-one correspondence between character and genetic factor, enabling a more modern and logical relationship between the gene and the trait to emerge. It was not until 1944, however, that nucleic acids rather than proteins were revealed as the material core of heredity, and not until the 1950s that there was a consensus on that subject.

88. See Raphael Falk, "The Struggle of Genetics for Independence," *Journal of the History of Biology* 28, no. 2 (1995): 239.

89. See Jan Sapp, "The Struggle for Authority in the Field of Heredity, 1900–1932: New Perspectives on the Rise of Genetics," *Journal of the History of Biology* 16, no. 3 (1983): 322.

90. George Stocking Jr., "The Turn-of-the-Century Concept of Race," *Modernism/Modernity* 1, no. 1 (January 1994): 6.

91. Ibid., 10. Intriguingly, though, this confusion of race and nation had its limitations. Nicholas Hudson argues in "From 'Nation' to 'Race': The Origin of Racial Classification in Eighteenth Century Thought," *Eighteenth-Century Studies* 29, no. 3 (1996) that although race and nation both stem from the same concept of lineage or stock (249), the Africans represented a special case, since they "roughly constituted a single 'race' even in the traditional sense of lineage" (249). As well, in the eighteenth century, the detail devoted to the Native Americans was disparaged as writers emphasized the similarities of the American race, and denied "savages" the complexity of nationality (256). Race as a scientific category more familiar to those in the twentieth century—like disciplinary techniques such as fingerprinting—was first used to describe others, and then applied to "civilized" folk.

92. William Provine, "Geneticists and Race," *American Zoologist* 26 (1986): 859–860.

93. Stocking, "Turn-of-the-Century Concept," 16.

94. Ernst Mayr, *The Growth of Biological Thought: Diversity, Evolution, and Inheritance* (Cambridge, Mass.: Belknap Press, 1982), 695–698.

95. Ibid., 700.

96. Diane Paul, "'In the Interests of Civilization': Marxist Views of Race and Culture in the Nineteenth Century," *Journal of the History of Ideas* 42, no. 1 (January–March 1981): 116–117.

97. See Evelyn Fox Keller, "Nature, Nurture, and the Human Genome Project," in *Code of Codes*, ed. D. J. Kevles and L. Hood (Cambridge, Mass.: Harvard University Press, 1991), 281–357; and Eve Kosofsky Sedgwick, *Tendencies* (New York: Routledge, 1994), 160.

98. Punnett, *Mendelism,* 167–168.

99. Ibid., 169.

100. Ibid.

101. Nils Roll-Hansen, "The Progress of Eugenics: Growth of Knowledge and Change in Ideology," *History of Science* 26 (1988): 293–331.

102. Curt Stern as quoted by Diane Paul, "From Eugenics to Medical Genetics," *Journal of Policy History* 9, no. 1 (1997): 101.

103. Garland Allen, "The Social and Economic Origins of Genetic Determinism: A Case Study of the American Eugenics Movement 1900–1940 and Its Lessons for Today," *Genetica* 99, no. 2–3 (March 1997): 77–88.

104. Michel Foucault, *History of Sexuality*, volume 1, trans. Robert Hurley (New York: Vintage, 1978), 136.

105. Ibid., 137.

106. Michel Foucault, *Security, Territory, Population: Lectures at the College de France 1977–1978*, trans. Graham Burchell (New York: Picador, 2007), 2.

107. Ibid., 79.

108. Charles B. Davenport, *Heredity in Relation to Eugenics* (New York: Arno Press & New York Times, 1972), 265–266.

109. Ibid., 266.

110. For more on genetics as human capital see Michel Foucault, *The Birth of Biopolitics: Lectures at the Collège de France, 1978–1979*, trans. Graham Burchell (Basingstoke, England, and New York: Palgrave Macmillan, 2008), 227.

111. Margaret Sanger, "Chapter XVIII: The Goal," in *Woman and the New Race* (New York: Truth Pub. Co., 1920), 229.

112. Foucault, *History of Sexuality*, 147.

113. Foucault, *Security, Territory, Population*, 100.

114. Michel Foucault, *Society Must Be Defended: Lectures at the College de France 1975–1976*, trans. David Macey (New York: Picador, 2003), 255.

115. Sapp, "Struggle for Authority," 337, 318.

116. Jacob, *Logic of Life*, 1.

117. Ibid., 338.

118. Davenport, *Heredity in Relation to Eugenics*, 1, 2, 4.

119. Ibid., 7; emphasis in original.

120. Ibid., 234; emphasis in original.

121. Ibid., 255.

122. Ibid., 260.

123. See Diane B. Paul and Hamish G. Spencer, "Did Eugenics Rest on an Elementary Mistake?," in *Thinking about Evolution: Historical, Philosophical, and Political Perspectives*, ed. Rama S. Singh et al. (Cambridge: Cambridge University Press, 2000), 103–118.

124. Provine, "Geneticists and Race," 857. Nils Roll-Hansen, responding to Provine's and Paul's arguments, takes a far more positivist position, arguing that the trend, from the new genetics of

1915 on, has been, and continues to be that the more that is known of human heredity, the more ideas of human programming are abandoned. For him, politics did have some role to play in this trajectory, but eugenics, he insists, was abandoned for scientific reasons. Roll-Hansen's argument, however, relies on a rather specious distinction between science and "mere politics" (as if politics too did not try to grapple with "facts") and relies on an odd definition of eugenics. For instance, he argues that one could support a Norwegian sterilization law that caused "legally incompetent persons to be sterilized with consent of guardian if serious mental illness, or high degree retarded or enfeebled, and reason to believe they could not support themselves or their offspring, or condition transferred to offspring" and not support eugenics ("Progress of Eugenics," 317). But what was eugenics if not the sterilization of those that could not support one's offspring, or of those with a condition that would detrimentally affect society as a whole? It was after all, the means by which the "fit" were to reproduce more than the "unfit."

125. Lily Kay, *The Molecular Vision of Life: Caltech, the Rockefeller Foundation, and the Rise of the New Biology* (Oxford: Oxford University Press, 1993), 8.

126. Ibid., 9.

127. Ibid., 17.

128. Ibid., 49.

129. Ibid., 92.

130. Ibid., 282.

131. Kay, *Who Wrote the Book of Life?*, xvi.

132. Ibid., 3.

133. Jacob, *Logic of Life*, 251.

134. Foucault, *Birth of Biopolitics*, 63–64.

135. Claire J. Tomlin and Jeffrey D. Alexrod, *Nature Reviews Genetics* 8 (May 2007): 331.

136. I. C. Weaver, N. Cervoni, F. A. Champagne, A. C. D'Alessio, S. Sharma, J. R. Seckl, S. Dymov, M. Szyf, and M. J. Meaney, "Epigenetic Programming by Maternal Behavior," *Nature Neuroscience* 7 (2004): 847–854.

137. Personal correspondence, September 18, 2009.

138. Catherine Malabou, *What Should We Do with Our Brain?*, trans. Sebastian Rand (New York: Fordham University Press, 2008), 41, 50; emphasis in original.

139. Ibid., 72.

The Undead of Information

1. See Matthew Kirschenbaum, *Mechanisms: New Media and the Forensic Imagination* (Cambridge, Mass.: MIT Press, 2008).

2. Sigmund Freud, "A Note Upon the 'Mystic Writing-Pad,'" in *The Standard Edition of the Complete Psychological Works of Sigmund Freud*, volume XIX (1923–1925): *The Ego and the Id and Other Works* (New York: W. W. Norton & Company, 1990), 230.

3. Jacques Derrida, *Archive Fever: A Freudian Impression*, trans. Eric Prenowitz (Chicago: University of Chicago Press, 1996), 19, 91–92.

4. Ibid., 15.

5. Warren J. Kurse II and Jay G. Heiser, *Computer Forensics: Incident Response Essentials* (Boston: Addison-Wesley, 2002), 77.

6. François Jacob, *Of Flies, Mice, and Men*, trans. Giselle Weiss (Cambridge, Mass.: Harvard University Press, 1998), 20–21. See Walter Gilbert, "The RNA World," *Nature* 319 (February 1986): 618.

7. Jacques Derrida, *Of Grammatology*, corrected ed., trans. Gayatri Chakravorty Spivak (Baltimore, Md.: Johns Hopkins University Press, 1997), 9.

8. Karl Marx, *Capital Volume One*, trans. Ben Fowkes (New York: Vintage, 1977), 163.

9. Thomas Keenan, "The Point Is to (Ex)Change It: Reading *Capital* Rhetorically," in *Fetishism as Cultural Discourse*, ed. Emily Apter and William Pietz (Ithaca, N.Y.: Cornell University Press, 1993), 165, 168.

4 Always Already There, or Software as Memory

1. Jutta Read-Scott, "Preserving Research Collections: A Collaboration between Librarians and Scholars," published by the Association of Research Libraries and the Modern Language Association, and the American Historical Association on behalf of the Task Force on the Preservation of the Artifact at the Annual Meeting of the Modern Languages Association, Chicago, December 27–30, 1999, <http://www.mla.org/rep_preserving_collections>, accessed 8/8/2010.

2. Ibid., "Introduction."

3. Abby Smith, *Why Digitize* (Washington, D.C.: Council on Library Resources, 1999): 3, 5, <http://www.eric.ed.gov:80/ERICDocs/data/ericdocs2sql/content_storage_01/0000019b/80/17/5d/23.pdf>, accessed 9/1/2009.

4. Archive as "always already there" is evident in Reginald Punnett's and Charles Davenport's texts discussed in chapter 3, in which the accumulation of human knowledge drives scientific and "civilized" progress. In its more dynamic form, software has also come to ground the "solution" to the traditional archive, the difficulties of access and selection outlined by Vannevar Bush in "As We May Think": hypertext as memex erases questions of reading as it focuses our attention on mapping connections. Even in its most critical form, Michel Foucault's formulation of the archive—not as something accumulated, but rather as first the law of what can be said—resonates with the logic of software. Discourse follows certain rules, its "networks" making possible certain positions of enunciation ("users").

5. See Geoffrey C. Bowker, *Memory Practices in the Sciences* (Cambridge, Mass.: MIT Press, 2005), 100.

6. John von Neumann, "First Draft of a Report on the EDVAC," Contract No. W-670-ORD-4926 between the United States Army Ordnance Department and the University of Pennsylvania, June 30, 1945, <http://qss.stanford.edu/~godfrey/vonNeumann/vnedvac.pdf>, 9, accessed 9/7/2009.

7. Von Neumann states, it is "only a transient standpoint, to make the present preliminary discussion possible. After the conclusions of the preliminary discussion, the elements will have to be reconsidered in their true electromagnetic nature. But at that time the decisions of the preliminary discussion will be available, and the corresponding alternatives accordingly eliminated" (Ibid.).

8. See William Aspray, *John von Neumann and the Origins of Modern Computing* (Cambridge, Mass.: MIT Press, 1990): 42.

9. Von Neumann, "First Draft," 3.

10. Warren S. McCulloch, "What Is a Number, that a Man May Know It, and a Man, that He May Know a Number?," in *Embodiments of Mind* (Cambridge, Mass.: MIT Press, 1965), 9.

11. Von Neumann, "Theory of Self-Reproducing Automata," *Papers of John von Neumann on Computers and Computer Theory*, Charles Babbage Institute Reprint Series for the History of Computing, vol. 12 (Cambridge, Mass.: MIT Press, 1986), 447.

12. Von Neumann, "First Draft," section 4.2.

13. Claude Shannon, "A Symbolic Analysis of Relay and Switching Circuits" (1936), in *Claude Elwood Shannon, Collected Papers*, ed. N. J. A. Sloane and Aaron D. Wyner (New York: IEEE Press, 1993).

14. Ibid., 2.

15. Von Neumann, "General and Logical Theory of Automata," *Papers of John von Neumann*, 396. Jay Forrester allegedly abandoned his initial plan to create an analog universal flight simulator precisely because the complex nature of the machine made distinguishing between noise and signal difficult.

16. Ibid., 398.

17. Ibid., 400–401.

18. Thomas D. Truitt and A. E. Rogers, *Basics of Analog Computers* (New York: Rider, 1960), 1–3; emphasis in original.

19. In the 1960s, analog computers were used in reactor engineering to solve problems related to "(1) automatic control rod drives; (2) stability of reactor power plants with internal feedbacks; (3) studies of reactivity lifetime reactor fuels; (4) dynamics of heat transfer and coolant flow" (Lawrence T. Bryant et al., *Introduction to Electronic Analogue Computing* [Chicago: Argonne National Labs, 1960], 9).

20. Lawrence T. Bryant and colleagues separate analyzers from analog computers based on the use of electronics (*Introduction to Electronic Analogue Computing*, 9). By 1948, the REAC (Reeves Analog Computer) had arrived. By 1949, Hartree noted the growing (wrongheaded) consensus in the United States to refer to (what would become) computers as analog machines and digital machines; Edmund C. Berkeley in his 1949 *Giant Brains, or Machines that Think* explained: "Machines that handle information as measurements of physical quantities are called *analogue* machines, because the measurement is *analogous* to, or like, the information" (New York: Wiley, 65). The fact that differential analyzers, along with early digital machines, were similarly dubbed "robot brains" or calculators and then became (retroactively) "computers" in the 1950s onward (the term *computer* makes far more sense in terms of numerical machines rather than differential analyzers, since both human and digital computers use numerical methods) reveals the extent to which analog and digital machines were considered to be analogous. The fact that differential analyzers would become "analogy" machines, and analog would constantly be confused with continuous, reveals the details of their coemergence and codependence.

21. U.S. War Department, "F U T U R E" press release, Bureau of Public Relations, Saturday, February 16, 1946, <http://americanhistory.si.edu/collections/comphist/pr2.pdf>, accessed 9/7/2009. The press release does go on to explain the difference between "digital" (general purpose) and "continuous signal" (specialized).

22. Vannevar Bush and Samuel H. Caldwell, "A New Type of Differential Analyzer," *Journal of the Franklin Institute,* 240 (1945): 255.

23. Ibid., 262.

24. John J. Rowlands, Director of News Service, MIT, press release, Tuesday, October 30, 1945.

25. Ibid.

26. Ibid.

27. Martha G. Morrow, "Magic Brains Spur Science and Technology: Fabulous Electronic Computers Herald Faster Planes and More Accurate Guns," in *New York World-Telegram*, March 10, 1946.

28. Herbert B. Nichols, "New Mathematical Robots Unscramble Digits to Multiply Inventions: Research Labs Calculate Devices to Bridge Years of Two Plus Two," in *Christian Science Monitor*, March 20 1946, 11.

29. Larry Owens, "Vannevar Bush and the Differential Analyzer," in *From Memex to Hypertext: Vannevar Bush and the Mind's Machine*, ed. James M. Nyce and Paul Kahn (Boston: Academic Press, 1991), 23.

30. These resemblances are not merely accidental or natural, since charge was conceived in terms of force and flow, and each concept is used to elucidate the others.

31. As quoted by Larry Owens, "Vannevar Bush and the Differential Analyzer: The Text and Context of an Early Computer," *Technology and Culture* 27, no. 1 (January 1986): 66.

32. James M. Nyce, "Nature's Machine: Mimesis, the Analog Computer, and the Rhetoric of Technology," in *Computing with Biological Metaphors*, ed. Ray Paton (London: Chapman & Hall, 1994), 417. Differential analyzers "did not reduce phenomena to any set of general principles and then treat those principles as the thing itself. In effect then, they both follow and obey the same laws . . . these machines represent a triumph over a tendency in science to formalize and typify phenomena" (418).

33. Hartree, *Calculating Instruments and Machines*, 1.

34. Nyce, "Nature's Machine," 420. Furthermore, although analog machines (as Nyce later argues) held out "a possibility that for the brain or the machine to be understood, neither has to be reduced to any one set of laws or logical principles" (422), differential analyzers and other computing machines were called a "roomful of brains" in the popular press and in MIT's own press releases. The word brain, rather than mind, highlights physical particularity, and the description of these machines as brains draws on similarities between mechanisms/inputs: electricity is like blood or food, which gives the machine energy; the gears are like cells; and the differential analyzer has only one mechanical thought: differential equations. The robot brain thus operated by imitation, rather than duplication.

35. Indeed, it is strange and revealing that many scholars who argue that analog and digital computers thrived together during the 1940s to 1960s nonetheless base their analyses solely on the differential analyzer.

36. Bush and Caldwell, "New Type of Differential Analyzer," 277.

37. Operational amplifiers, as their name implies, amplify voltage differences. The simplest op-amp, a triode amplifier, comprises an electron tube with three electrodes, a cathode, a plate, and a control grid. As Truitt and Rogers explain, the cathode is heated to a high temperature by an electrical hot-wire filament, a positive voltage is applied between the plate and the cathode, and electrons are ejected and directed to the plate. The number of electrons per second traveling to the plate, namely, the plate current, depends on the temperature of the cathode and the applied voltage. If a negative voltage is applied between the control grid (located between the cathode and the plate) and the cathode, then the value of this grid voltage exercises far greater control over the plate current than does the value of the plate voltage. A small change in grid voltage can thus cause a relatively large change in plate current (Truitt and Rogers, *Basics of Analog Computers,* 59).

Because they enable relatively low-cost signal amplification, op-amps were key to long-distance telephone communications. Their amplification, however, is noisy and nonuniform: they usually amplify a rapidly changing voltage less than a steady voltage, and the amplification is time delayed (instead of giving an immediate amplification, in other words, there is more of a curve). Negative feedback, which Harold Black "discovered" in 1927, makes op-amps less noisy and more like an on-off switch. As David Mindell argues, what was key to "discovering" negative feedback was conceptualizing the "output . . . as containing a pure, desirable component—the signal—and an impure, unwanted component—the distortion" (*Between Human and Machine*, 118).

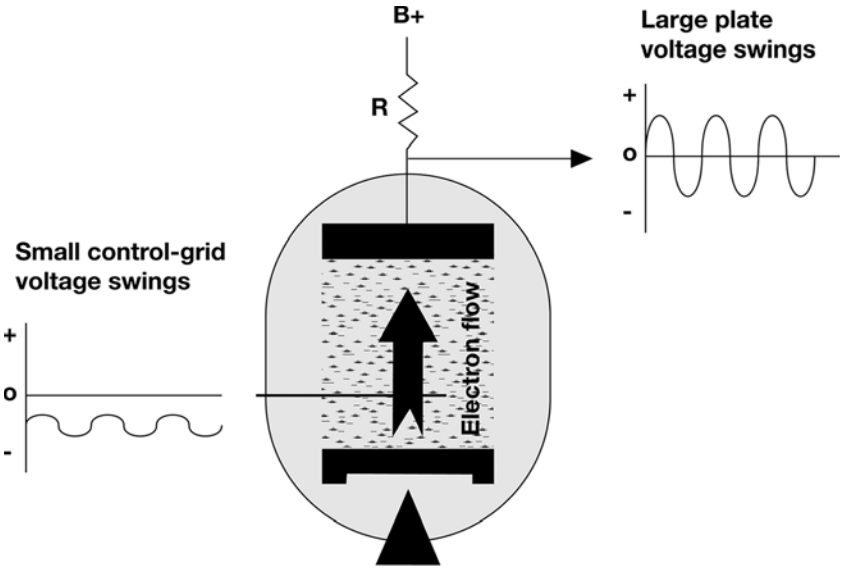

Figure 4.12
Basic triode

38. B. Holbrook and Uta C. Merzbach, "Interview," May 10, 1969, *Computer Oral History Collection, 1969–1973, 1977*, *Archives Center, National Museum of American History, Smithsonian*, <http://invention.smithsonian.org/downloads/fa_cohc_tr_holb_690510.pdf>,10, accessed 8/8/2010.

39. War Department, "F U T U R E," 3.

40. Hartree, *Calculating Instruments and Machines*, 15.

41. Ibid., 111. In a historical irony, Hartree's division between programming and coding, designed in part to separate mere coders from mathematician programmers, has disappeared, as has, arguably, the act of programming itself.

42. McCulloch, "What Is a Number," 1.

43. "Because of the 'all-or-none' character of nervous activity," they write, "neural events and the relations among them can be treated by means of propositional logic" (Walter McCulloch and Walter Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity," in *Embodiments of Mind*, 19). They also state, "Many years ago one of us, by considerations impertinent to this argument, was led to conceive of the response of any neuron as factually equivalent to a proposition which proposed its adequate stimulus. He therefore attempted to record the behavior of complicated nets in the notation of the symbolic logic of proposition. The 'all-or-none' law of nervous activity is sufficient to ensure that the activity of any neuron may be represented as a proposition. Physiological relations existing among nervous activities correspond, of course, to

relations existing among the propositions; and the utility of the representation depends upon the identity of these relations with those of the logic of propositions" (21).

44. Ibid., 3.

45. McCulloch and Pitts, "A Logical Calculus," 38.

46. Ibid., 22.

47. McCulloch, "What Is a Number," 8. In "A Logical Calculus," McCulloch and Pitts write: "To psychology, however, defined, specification of the net would contribute all that could be achieved in that field—even if the analysis were pushed to ultimate psychic units or 'psychons,' for a psychon can be no less than the activity of a single neuron. Since that activity is inherently propositional, all psychic events have an intentional, or 'semiotic,' character" (38).

48. Ibid., 24.

49. Ibid., 22.

50. Russell as quoted by McCulloch, "What Is a Number," 6–7.

51. Warren McCulloch, "Why the Mind Is in the Head," in *Embodiments of Mind*, 73.

52. Ibid.

53. Seymour Papert, "Introduction," in *Embodiments of Mind*, xvi.

54. J. Y. Lettvin, H. R. Maturana, W. S. McCulloch, and W. H. Pitts, "What the Frog's Eye Tells the Frog's Brain," in *Embodiments of Mind*, 251.

55. McCulloch and Pitts, "A Logical Calculus," 35. More strongly, McCulloch would state in 1951 that the "cortex computes" ("Why the Mind Is in the Head," 85).

56. Turing's logic itself remarkably depends on analogies, which become equivalences.

57. McCulloch, "Why the Mind Is in the Head," 72.

58. Ibid., 91.

59. McCulloch and Pitts, "A Logical Calculus," 35.

60. Ibid., 35.

61. Bowker, *Memory Practices in the Sciences*, 100.

62. Warren S. McCulloch, "Finality and Form," in *Embodiments of Mind*, 275.

63. McCulloch, "Why the Mind Is in the Head," 84.

64. Ibid.

65. Ibid.

66. For more on this see Paul Edwards, *The Closed World: Computers and the Politics of Discourse in Cold War America* (Cambridge, Mass.: MIT Press, 1996).

67. McCulloch and Pitts, "A Logical Calculus," 38.

68. Warren S. McCulloch, "Physiological Processes Underlying Psychoneuroses," in *Embodiments of Mind*, 373.

69. Warren S. McCulloch, "Toward Some Circuitry of Ethical Robots or an Observational Science of the Genesis of Social Evaluation in the Mind-Like Behavior of Artifacts," in *Embodiments of Mind*, 197.

70. John von Neumann, *Theory of Self-Reproducing Automata* (Urbana: University of Illinois, 1966), 39.

71. Ibid., 40.

72. John von Neumann, *The Computer and the Brain*, 2nd ed. (New Haven, Conn.: Yale University Press, 2000), 61.

73. McCulloch, "Why the Mind Is in the Head," in *Embodiments of Mind*, 102. Current studies of memory have moved away from this stark distinction between order and data back toward an analogy-based system of memory that insists that the brain preserves traces of events through the combination of neurons.

74. Von Neumann, *Theory of Self-Reproducing Automata*, 39.

75. Jacques Derrida, *Archive Fever: A Freudian Impression* (Chicago: University of Chicago Press, 1998), 19; emphasis in original.

76. Arthur W. Burks, Herman H. Goldstine, and John von Neumann, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," Part I, Volume 1 (Princeton, N.J.: Institute for Advanced Study, 1946).

77. Von Neumann, *An Electronic Computing Instrument*, section 4.4 (6).

78. Marshall McLuhan, *Understanding Media: The Extensions of Man* (Cambridge, Mass.: MIT Press, 1964).

79. Von Neumann, *The Computer and the Brain*, 36.

80. McCulloch, "Why the Mind Is in the Head," 133.

81. As quoted in McCulloch, "Why the Mind Is in the Head," 92–93. Von Neumann's move to files is intriguing, especially if one considers the importance of files—and of disposing files—to modern bureaucracy and state power (see Cornelia Vismann's *Files: Law and Media Technology*, trans. Geoffrey Winthrop-Young [Stanford, Calif.: Stanford University Press, 2008]).

82. As Derrida notes, a trace that cannot fade is not a trace but a full presence, the son of God (Jacques Derrida, "Freud and the Scene of Writing," *Writing and Difference*, trans. Alan Bass [Chicago: University of Chicago Press, 1978], 230).

83. Von Neumann, *The Computer and the Brain*, 65.

84. For more on the relationship between neoliberalism and Nazism, see Michel Foucault, *The Birth of Biopolitics: Lectures at the Collège de France, 1978–1979*, trans. Graham Burchell (Basingstoke, England, and New York: Palgrave Macmillan, 2008), 106–116.

85. William Poundstone, *Prisoner's Dilemma* (New York: Doubleday, 1992), 194.

86. Nicholas Vonneuman, "The Philosophical Legacy of John von Neumann, in the Light of Its Inception and Evolution in His Formative Years," paper presented at The Legacy of John von Neumann Symposium at Hoftstra University, May 30, 1988 (Meadowbrook, Pa.: N. A. Vonneuman, 1987–1988), 1.

87. Ibid., 2.

88. Ibid.

89. Johannes Wolfgang Goethe, "Faust's Study (i)," *Faust Part One,* trans. David Luke (Oxford: Oxford University Press, 1987), lines 1225–1237.

90. Von Neumann, "General and Logical Theory," 392.

91. Von Neumann writes, "I will introduce as elementary units neurons, a 'muscle,' entities which make and cut fixed contacts, and entities which supply energy, all defined with about that degree of superficiality with which the formal theory of McCulloch and Pitts describes an actual neuron by axiomatizing automata in this manner, one has thrown half of the problem out the window, and it may be the more important half. One has resigned oneself not to explain how these parts are made up of real things, specifically, how these parts are made up of actual elementary particles, or even of higher chemical molecules" ("Theory of Self-Reproducing Automata," 480).

92. Von Neumann, "General and Logical Theory," 412–413.

93. Ibid., 413. Von Neumann also offers an interpretation of Turing's "On Computable Numbers, with an Application to the Entscheidungsproblem" (see note 117) in order to justify this statement. Von Neumann writes:

> Turing observed that a completely general description of any conceivable automaton can be (in the sense of the foregoing definition) given in a finite number of words. The description will contain certain empty passages—those referring to the functions mentioned earlier . . . which specify the actual functioning of the automaton. When these empty passages are filled in, we deal with a specific automaton. As long as they are left empty, this schema represents the general definition of the general automaton. Now it becomes possible to describe an automaton, which has the ability to interpret such a definition. In other words, which, when fed the functions that in the sense described above define specific automaton, will thereupon function like the object described. The ability to do this is no more mysterious than the ability to read a dictionary and a grammar and to follow their instructions about the uses and principles of combinations of words. This automaton, which is constructed to read a description and to imitate the object described is then the universal automaton in the sense of Turing. (Von Neumann, "General and Logical Theory," 416)

94. Von Neumann, "Theory and Organization of Complicated Automata," *Papers of John von Neumann*, 450.

95. Von Neumann, "General and Logical Theory," 414.

96. Arthur Burks, "Introduction: Theory of Natural and Artificial Automata," *Papers of John von Neumann*, 374.

97. Von Neumann, "General and Logical Theory," 419.

98. Ibid.

99. Ibid., 420.

100. Ibid.

101. Ibid., 421.

102. Von Neumann, *Theory of Self-Reproducing Automata*, 101 and 113. In other models of self-reproduction, such as the cellular model, von Neumann still privileged the role of memory. It is only with a large external yet accessible memory that his cellular units can be logically universal—capable of inductive processes—and thus function as fundamental cells. In describing the function of the construction of a "tape and its control" for the cellular automata, von Neumann treats the memory as containing instructions for the creation of a secondary automata (Ibid., 202).

103. Von Neumann, *Papers of John von Neumann*, 368. This comparison, however, occurs not only on the level of mathematics or mathematization, but also on the level of heuristics, descriptions, and strategies.

104. John von Neumann and Oskar Morgenstern, *Theory of Games and Economic Behavior* (Princeton, N.J.: Princeton University Press, 1947), 9.

105. Ibid., 7.

106. Ibid., 48.

107. Ibid., 79.

108. Ibid., 49.

109. As quoted in Poundstone's *Prisoner's Dilemma*, 168.

110. Von Neumann and Morgenstern, *Theory of Games*, 31.

111. Milton Friedman, *Capitalism and Freedom*, Fortieth Anniversary Edition (Chicago: Chicago University Press, 2002), 25.

112. A. M. Turing, "On Computable Numbers, with an Application to the Entscheidungsproblem," *Proceedings of the London Mathematical Society* 2, no. 42 (1937): 230–265.

113. Ibid., section 6.

114. Von Neumann, *The Computer and the Brain*, 70–71.

115. Ibid., 71.

116. Ibid., 72–73; emphasis in original.

117. Frances A. Yates, *The Art of Memory* (Chicago: University of Chicago Press, 2001), 6–7.

118. See Michael R. Williams, *A History of Computing Technology* (Los Alamitos, Calif.: IEEE Computer Society Press, 1977), 306–316.

119. Derrida, "Freud and the Scene of Writing," 226.

120. Wolfgang Ernst, "Dis/continuities: Does the Archive Become Metaphorical in Multi-Media Space," in *New Media Old Media*, ed. Wendy Hui Kyong Chun and Thomas Keenan (New York: Routledge, 2006), 118. Although this is certainly true for CRT screens, it is not necessarily true for LCD screens, which operate more like blinds that allow certain sections of light through.

121. Matthew Kirschenbaum, *Mechanisms: New Media and the Forensic Imagination* (Cambridge, Mass.: MIT Press, 2008).

122. Repetition not only grounds the archive, it also threatens it. Drawing from Freud's work on the death drive, Derrida argues, "Repetition itself, the logic of repetition, indeed the repetition compulsion, remains . . . indissociable from . . . destruction" (*Archive Fever*, 11–12).

123. Jeffrey Toobin, "Scotus Watch," The Talk of the Town, in *The New Yorker*, November 11, 2006, <http://www.newyorker.com/talk/content/articles/051121ta_talk_toobin>, accessed 01/27/2006.

124. This is because there are no shelves, no fixed relation between what is storeable and the place they are stored. As Harriet Bradley has argued, the Internet breaks the bond between location and storage: if before "only what has been stored can be located," now "memory is no longer located in specific sites" ("The Seductions of the Archive: Voices Lost and Found," *History of the Human Sciences* 12, no. 2 [1999]: 113).

125. <http://www.archive.org/about/about.php>, accessed 2/1/2007.

126. <http://www.archive.org/about/about.php>, accessed 2/1/2007.

127. Ernst, "Dis/continuities," 119.

128. See Paul Virilio, "The Visual Crash," in *CTRL [SPACE]: Rhetorics of Surveillance from Bentham to Big Brother*, ed. Thomas Y. Levin et al. (Cambridge, Mass.: MIT Press, 2002), 108–113; *Open Sky*, trans. Julie Rose (London: Verso, 1997); and "Speed and Information: Cyberspace Alarm!," <http://www.ctheory.net/text_file.asp?pick=72>, accessed 2/1/2007.

Conclusion

1. Thomas Keenan, "The Point Is to (Ex)Change It: Reading *Capital* Rhetorically," in *Fetishism as Cultural Discourse,* ed. Emily Apter and William Pietz (Ithaca, N.Y.: Cornell University Press, 1993), 184–185.

2. Erwin Schrödinger, *What Is Life?* with *Mind and Matter* and *Autobiographical Sketches* (Cambridge, Mass.: Cambridge University Press, 1992), 21–22.

3. Ibid., 440.

4. Fredric Jameson, *Postmodernism, or The Cultural Logic of Late Capitalism* (Durham, N.C.: Duke University Press, 1991), 51.

5. For more on this see Michel de Certeau's *The Practice of Everyday Life,* trans. Steven Rendall (Berkeley: University of California Press, 1984).

6. This is the topic of my next book, "Imagined Networks."

7. Jacques Derrida stresses that writing represents the disappearance of the origin: "To repeat: the disappearance of the good-father-capital-sun is thus the precondition of discourse, taken this time as a moment and not as a principle of *generalized* writing. . . . The disappearance of truth as presence, the withdrawal of the present origin of presence, is the condition of all (manifestation of) truth. Nontruth is the truth. Nonpresence is presence. Differance, the disappearance of any originary presence, is *at once* the condition of possibility *and* the condition of impossibility of truth. At once" ("Plato's Pharmacy," *Dissemination,* trans. Barbara Johnson [Chicago: University of Chicago, 1981], 168).

8. Intriguingly, Slavoj Žižek links specters to a fear of freedom in his introduction to *Mapping Ideology* (London: Verso, 1994), 27.

9. Milton Friedman, *Capitalism and Freedom*, Fortieth Anniversary Edition (Chicago: Chicago University Press, 2002), lx. Michel Foucault discusses civil society as an umbrella term to bring together the subject of rights and the *homo oeconomicus* in *The Birth of Biopolitics: Lectures at the College de France, 1978–1979*, trans. Graham Burchell (Basingstoke, England, and New York: Palgrave Macmillan, 2008), 291–316.

10. See David Harvey, *A Brief History of Neoliberalism* (Oxford: Oxford University Press, 2005).

11. Vicente Rafael, "The Cell Phone and the Crowd: Messianic Politics in the Contemporary Philippines," *Public Culture* 15, no. 3 (2003): 419.

Epilogue

1. See Toni Morrison, *Playing in the Dark: Whiteness and the Literary Imagination* (New York: Vintage, 1993), 63.

2. For more on this, see Wendy Hui Kyong Chun and Lynne Joyrich, eds., *Race and/as Technology*, special issue of *Camera Obscura* 24 (2009), and Wendy Hui Kyong Chun, "Race as Archive," *Vectors: Journal of Culture and Technology in a Dynamic Vernacular* 3, no. 1 (2007).