The Media of Programming

Mark Priestley

Independent Scholar. m.priestley@gmail.com.

Thomas Haigh

University of Wisconsin—Milwaukee & Siegen University. Thomas.haigh@gmail.com.

Abstract: We revisit the origins of the modern, so-called "stored program" computer during the 1940s from a media-centric viewpoint, from tape-driven relay computers to the introduction of delay-line and cathode ray tube memories. Some early machines embodied fixed programs, but all general-purpose computers use a medium of some kind to store control information. The idea of a "memory space" composed of sequentially numbered "storage locations" is crucial to modern computing, but we show that this idea developed incrementally and was not fully articulated in John von Neumann's First Draft of a Report on the EDVAC. Instead, the designers of computers based around delay line memories conceptualized their structure using different temporal and spatial schemes. Referencing the correct data was not just a question of "where" but also one of "when."

The phrase "stored program computer" is often used as shorthand for computers patterned after the design set down by John von Neumann in his 1945 *First Draft of a Report on the EDVAC*. We have argued elsewhere (Haigh, Priestley, and Rope 2014) that it is misleading to compress the cluster of computing technologies developed during the mid-1940s into a single innovation and to erect a rigid barrier between (comparatively) ancient and modern computing depending on whether a machine stored coded instructions in an addressable electronic memory that was both readable and writable.

We consider it particularly unwise to speak of a stored program *concept*, as the phrase suggests that that the single innovation was an abstract idea. Professional historians have largely moved on from discussing the early computers of 1940s to focus on more recent developments. In their absence, authors such as Martin Davis (Davis 2001) and Jack Copeland (Copeland 2013) have claimed to find the source of this supposed concept in logic, reinforcing an abstract view of the origins of modern computing.[1]

[1] Of course, the "stored program concept" has been discussed at length by many other historians, including William Aspray, Paul Ceruzzi, and Doron Swade. We engage in detail with

Perhaps because of this abstract turn, the widely held belief that the essence of the modern computer is tied up with the question of the storage of programs has not led to a significant engagement with the characteristics of the digital media doing the storing. Accounts of the kind offered by Davis and Copeland privilege mathematical and abstract ideas and downplay the technological development of delay lines, Williams tubes, and other forms of information storage device.[2] Conversely, work engaging with the details of early memory technology, even by some of the same authors (Copeland et al. 2017), has not typically focused on creation and initial adoption of the EDVAC model for the modern computer or considered the interplay of memory technologies and architecture. In this chapter we retell the familiar story of the emergence of modern computing in the mid-1940s from the unfamiliar perspective of the digital media used to encode the programs of instructions that the machines carried out.

Automation and Programming

The mechanical aids to digital computation available in the 1930s, desk calculators and punched card machines, only automated individual operations. Extended computation still depended on human labor: people ferried card decks around corporate offices, or transcribed numbers from calculator to paper and back again in the computing rooms of the Mathematical Tables Project (Grier 2006) and the country homes of the retired mariners working for the UK's Nautical Almanac Office (Croarken 2003).

Automation had been carried further in non-digital machines such as the differential analyzer. In a computation set up on an analyzer, elementary operations were executed continuously and simultaneously on quantities that were not represented digitally, but by some physical analogue such as the rotation of a shaft. Analyzers contained devices to perform individual operations, such as integrators and adders, but did not sequence discrete operations over time as human and automatic digital computers did (Haigh and Priestley 2016).

Between the 1820s and the early 1940s several groups described, and some built, digital machines that automatically performed sequences of operations. In some the sequence of operations was determined solely by the physical structure of the machine. Babbage's Difference Engines fall into this category, as does the Bell Labs Complex Number Calculator of 1940, a special-purpose machine capable of performing the four fundamental operations of arithmetic on complex numbers. According to its inventor George Stibitz, it contained units handling the real and imaginary parts of the calculation which "operated in parallel, during multiplication, for example, when the real and imaginary parts of the multiplicand were multiplied by digits of the real part of the multiplier simultaneously" (Stibitz 1967, p. 39). The machine implemented the operations of real-number arithmetic but gave its users no way of re-sequencing those operations for any other purpose.

their work in {Haigh, 2016 #5964@ch. 11} and {Haigh, 2014 #5717} and will not repeat that analysis here.

[2] Copeland has in fact produced an excellent study of early tube based memory devices (Copeland et al. 2017), but this engagement with materiality does not appear to have altered his conviction that Alan Turing invented the "stored program computer" in 1936.

Other special-purpose machines of the era, with similarly fixed programs, included the Colossus machines and the Atanasoff-Berry Computer. These machines performed a small number of operation sequences defined by their physical construction. Using a phrase introduced by the ENIAC group (Grier 1996) we describe such machines as embodying a single "program of operations". As with other contemporary uses of the word "program," the exact sequence of operations carried out would vary, in this case according to properties of the input data or user configuration.

Greater flexibility could be obtained by allowing the sequence of operations to be specified in advance of the computation starting, a process usually described as giving "instructions" or "orders" to the machine. In the 1930s, Konrad Zuse (Zuse 1993) and Howard Aiken (Cohen 1999) began projects to construct machines capable of obeying sequences of instructions, and the general approach was theorized by Alan Turing (1936) and Emil Post (1936). The metaphor of "giving orders" resonated with the familiar dynamic of supervisors and human computers, something brought out clearly in Post's account. The term "automatically sequenced" was introduced to distinguish these machines from their less capable forebears.

Around the middle of 1940s, the term "program" began to be used to refer to these sequences of instructions rather than operations to which they gave rise (Haigh, Priestley and Rope 2016). The creation of these sequences is one of the key senses of what we now call "programming," In the mid-1940s, however, "program" and its derivatives were used in other senses: a program could be a sequence of operations or instructions, and programming was a task carried out by control circuitry before it became "paper work" (Turing 1946) performed by humans. These ambiguities highlight an important theme of this chapter, namely the relationship between the static program of instructions given to an automatically sequenced machine and the dynamic unfolding of those instructions into a computation, or program of operations. This was originally thought to be a fairly straightforward correspondence, and we trace the interplay of logical and technological factors that led to a more complex understanding of its nature.

The distinction between machines which embody a single program and those which carry out programs supplied to them as a set of encoded orders appears to be fundamental. In the latter case, a novel computation can be carried out by simply supplying a different set of orders instead of building a new machine. We say "appears to be fundamental," however, because the distinction is not really one between different kinds of machine. The program embodied by an automatically sequenced machine consists of the operations required to read and interpret a coded sequence of instruction. A modern computer endlessly fetches instructions from memory, decodes them, and executes the corresponding operations. Likewise, Turing's abstract machines embody a single program, defined by a table listing their possible configurations and behavior. Turing's universal machines are not a new type of machine, but regular Turing machines put to a very specific use.

The instructions defining computations were encoded in a variety of quite distinct media. Each medium offered different affordances to the designer of the machines' repertoire of instructions and to its programmers and affected the ways in which instruction sequences could be mapped into temporal sequences of operations during the execution of a program. Proceeding largely chronologically, we describe how the instruction sets and practices of organizing computations on different machines were shaped by the properties of the media used to store the program instructions.

Storing Instructions on Tapes

Three famous historical actors independently began projects to build machines that would obey arbitrary instruction sequences: Charles Babbage (Swade 2001), Konrad Zuse (Zuse 1993) and Howard Aiken (Cohen 1999). All three conceived of computation as the execution of a temporal sequence of basic operations and therefore started by assuming that simple sequences of instructions, each causing the machine to execute a single operation, would be adequate to control computations. All three came to recognize that this model was over-simple.

Their machines encoded instructions as patterns of perforations in paper or cardboard configured to make a sequential, machine-readable medium, or *tape*. For the Analytical Engine, Babbage drew upon Joseph Marie Jacquard's use of punched cards to control the operation of mechanical looms. Instructions were punched on cards which, like Jacquard, Babbage proposed to tie together to make a sequential, machine-readable medium. For the Harvard Mark I, Aiken specified a custom-designed 24-channel tape out of IBM card stock, each position holding a single encoded instruction. Other machines, such as the Relay Interpolator built at Bell Labs in the early 1940s, used standard 5 or 6-hole teleprinter tape.

Addressable memory and the format of instructions

Unlike machines that embody a single program, tape-controlled computers were designed to carry out different programs at different times. This required a different approach to the storage of data. Special-purpose machines have special-purpose memory components: the electronic counters of the Colossus machines, for example, were permanently connected to the machines' logic circuits and were used solely to tally the number of times particular conditions were satisfied. In contrast, automatically sequenced machines need general purpose storage units that can be switched to connect with different computational units as demanded by different instructions. Mark I stored numbers in "counters," and a specific counter could at one moment receive a number from the machine's multiplier and at the next moment supply it as an argument for the computation of a sine function. The relationship between the store and the mill (CPU) of Babbage's Analytical Engine was similarly flexible.

The arguments for a specific operation could be taken from any storage unit, and the result of the operation placed in any storage unit. This meant that the storage units had to be identifiable in some way. This was done by assigning what would later be called an "address" (usually a number) to each individual memory unit. Instructions to the machine to perform a specific operation therefore needed to encode not only a reference to the operation, but also the addresses of the storage units involved.

Exactly how this encoding was carried out depended on the details of a machine's architecture. Many of Mark I's instructions, for example, contained two addresses, one specifying the counter from which data was to be read and the other specifying the counter into which the data would be placed. A third field specified the operations that the machine would carry out in response to reading the instruction.

Controlling computations

The first tape machines had only a single tape reader. They appear to have been designed on the assumption that there would be a straightforward one-to-one correspondence between the instructions punched on the tape and the operations carried out by the machine, similar to the relationship between the tape of a player piano and the music it performs.

The iterative nature of many mathematical calculations, where a relatively short sequence of operations is carried out repeatedly until the results obtained approach some required level of tolerance, complicated this simple picture. It would obviously be wasteful and error-prone to punch the same sequence of operations repeatedly onto a tape, and in cases where the number of iterations was not known in advance this would not even be feasible in principle. The obvious solution to this problem was to punch the instructions once and provide some way for them to be repeatedly presented to the machine. To this end, Babbage planned to introduce a "backing" mechanism which would allow the Analytical Engine to rewind its control tape to an earlier instruction. More mundanely, the ends of Mark I's tapes were glued together to make loops, called "endless tapes" loops that, when fed through the tape reader, would generate potentially unending iterative computations.

In practice, of course, mathematical computations come to an end. To stop an iterated instruction sequence, Mark I's designers provided a conditional stop order: when a calculated quantity was sufficiently close to the required value, the sequence mechanism would be stopped, and the operator could manually adjust the machine before it carried out the next sequence of instructions.

This simple feature introduces a significant complication into the relationship between instructions and operations performed. In the case of a telegram, the structure of the message directly corresponds to the structure of the tape. The characters of the message are simply the (encoded) characters punched on the tape - the same number of them and in the same order. Likewise, there is a one to one correspondence between the notes played on a player piano and those punched on its control tape.

With conditional termination, however, the situation is different. Suppose that 100 coded instructions have been punched on a Mark I control loop. As the computation proceeds, the instructions will be read repeatedly, and the number of times the tape cycles will vary from computation to computation, depending on the initial data supplied. By changing the (spatial) topology of the tape from a linear sequence to a cycle and allowing the machine itself to determine the number of times the tape is cycled, the length of the (temporal) sequence of operations performed by Mark I becomes difficult or impossible to predict in advance. What started as an apparently simple relationship whereby a sequence of instructions was transformed into a sequence of operations has become rather more complex, as the machine "unwinds" a "loop" of instructions into a longer sequence of operations. As Herman Goldstine and John von Neumann explained a few years later, "the relation of the coded instruction sequence to the mathematically conceived procedure of (numerical) solution is not a statical one, that of a translation, but highly dynamical" (Goldstine and von Neumann 1947).

Programming with Multiple Instruction Sequences

It soon became evident that most problems required more than just a single instruction tape. Many of the examples given in the Mark I's *Manual of Operation* (Staff of the Harvard Computation Laboratory 1946) use two tapes: a linear "initial tape" which set up things like the initial values of parameters, and an endless tape which performed the iterative calculations. When the first tape finished, it was removed by a human operator and the next tape loaded.

In general, control of a computation also involved making sure that the necessary sequences were selected and performed in the right order. Various common patterns of execution were recognized, namely the need to choose between alternative sequences, the need to perform a single sequence repeatedly, and the need to allow a sequence to be performed at different points in a computation. As well as the coded instructions on the tapes, programmers wrote orders for Mark I's operators. Both were supposed to leave no room for discretion or ambiguity. The conditional stop order was used to pause the machine for manually executed conditional branches, as one of the Mark I's first programmers, Richard Bloch, recalled:

> Since orders were not in registers, but only on tape, in order to branch you physically branched by stopping the machine or causing the machine to stop; whereupon the operator would twirl the drum that held the tape over to an indicated branch line and proceed from there. You had instructions that would say "if the machine stops here, move it over to the red line next; if it stops somewhere else, ship it over to the blue line." (Bloch 1984)

Such procedures further complicated the relationship between instructions and operations, as did the practice of "interpolating" orders. Mark I included special-purpose units to perform multiplication and various other operations. To avoid the rest of the machine sitting idly by while a multiplication was being carried out, say, extra instructions could be punched between the three coded instructions that controlled the multiplier. The consequence was that a single sequence of coded instructions on the tape would give rise to parallel sequences of operations as the machine read and processed the instructions.

Mark I therefore operated with two quite distinct levels of control. Coded orders were read from tape and executed in the order in which they had been punched while the tapes themselves were managed by the operators. Sequencing was therefore only partly automated, but in view of the machine's speed and intended application, the slowdown caused by the human operators was expected to be acceptable.

Program Pulses and Program Controls

The first programmable electronic computer, ENIAC, was expected to calculate several orders of magnitude faster than Mark I. Its lead designers John Mauchly and Presper Eckert understood from the beginning that ENIAC would need to automate both levels of control and be able to switch between different sequences of operations automatically as well as following an individual sequence. They also knew that reading instructions from an external medium such as paper tape would be unacceptably slow, so individual instructions were set up by turning switches on the many "program controls" distributed around the machine's units. These instructions were then linked into sequences by plugging a fixed connection between each instruction and the next in the sequence in a problem-specific "set up." At run-time, the execution of the sequence of instructions was controlled by “program pulses” that were received by a program control to trigger an operation, and then passed on in a kind of relay race to the next control in the sequence.

There were hierarchies of control within ENIAC, however, that complicated this simple picture. Some units, notably the multiplier and the divider, made coordinated use of several accumulators to carry out the sequences of additions and subtractions necessary to form products and quotients. The number of additions needed to perform a multiplication varied as the size of the operands increased (unlike Stibitz’s complex multiplication, which required exactly the same sequence of real-number operations on every occasion). A progress report described how, when a multiplication was triggered, “accumulators will be automatically programmed to receive the multiplier and the multiplicand” (Anonymous 1944, p. IV-10). This was an extension of a lower-level usage in which the “program circuits” within a unit controlled the operation of its arithmetic circuits, setting and unsetting of flip-flops and controlling other simple electronic circuits.

The system of program pulses moving between units offered a way for ENIAC to sequence operations at electronic speed, without the huge slowdown that would be caused by waiting for the next instruction to be read from tape. As the team developed it further, it also offered a way to shift automatically from one sequence to another. They arrived at a definitive model early in 1944 after Adele Goldstine and Arthur Burks had made detailed plans for a ballistic trajectory calculation.[3] The problem was broken down into four sequences – initialization, the operations to carry out a single integration step, printing results, hardware checking – which needed to be repeated in a relatively complex pattern involving two nested loops.

Coordinating these sequences was principally the responsibility of a dedicated unit, the “master programmer.” Combining counters with devices called steppers that allowed control to branch to one of up to six basic sequences, this unit allowed complex nested sequences of sequences to be set up, sequences to be repeated a fixed number of times, conditional branching either to choose between two sequences or to terminate a loop, and control to be transferred to different places after successive invocations of a “subsidiary” sequence.

To master this complexity, the ENIAC team also developed a graphical notation (Figure 1) for depicting the sequence programming involved in particular computation. These “master programmer diagrams” black-boxed the basic instruction sequences and showed how the steppers controlled the repetition and time-varying invocation of sequences.

Multiple tapes

The creators of the tape controlled computers soon realized the usefulness of letting their machines switch automatically between control sequences. An initial step was to add more tape readers, so that several sequences could be mounted at once, and to provide the machine with instructions to shift control between them automatically. Harvard’s operating procedures had already made the operators’ work as mechanical as possible. The Harvard Mark II, designed in

[3] In (Haigh et al 2016) we attributed this work to Burks, but subsequent archival research has revealed the importance of Goldstine’s contribution.

1945 after only a few months experience of using Mark I, allowed up to four sequence units to be used. Similar proposals had been made for the Bell Labs relay computer later known as "Model V." In March 1944 Samuel Williams proposed that the description of a problem would be split between a "routine tape" describing the operations to be performed and a "problem tape" containing "information necessary for the solving of the particular problem for which the tape has been prepared" (Williams 1944). In August, following discussions with Williams and Stibitz, von Neumann (1944) reported to Robert Oppenheimer that the machine would also use "auxiliary routine tapes […] used for frequently recurring subcycles," with separate readers for the different tapes.

In 1947 a "subsidiary sequence unit" was added to the Harvard Mark I. Similar in intent to the Mark II proposals, this unit allowed 10 sequences of around 50 instructions each to be set up. The instructions comprising the sub-sequences were not punched on tape but "wired with plug wires, the codes of each line were wired with a series of plug holes for a particular line". The sequencing of these instructions was carried out by stepping switches (Bloch 1984).

Mark II's four sequence units were divided, with units 1 and 2 being on the machine's left side, and units 3 and 4 on the right.[4] Having more than one sequence unit enabled the dynamic transfer of control between two instruction sequences. The Harvard team conceptualized this as a hierarchical relationship, though the hardware itself could have supported more flexible relationships:

> Should the method of solution involve a repeated sub-routine, it is advantageous to employ the sequence units in a dominant and subsidiary relationship. One unit—the dominant—initiates the computing routine and then calls in the other—the subsidiary. The latter executes the repeated portions of the routine as many times as required and then calls back the dominant to complete the routine. Thus the repeated sub-routine need be coded but once, permitting short control tapes to be used. (267)

The orders that transferred control between units could be read at any unit. Perhaps for this reason, instructions did not use the numeric identifiers of the units, but defined them relatively. Thus, operation code 67 had the meaning "switch to the other unit on the same side," while operation codes 71 and 72 switched control to one or other of the units on the opposite side of the machine. Operation code 70, a conditional branch, transferred control to the other sequence unit if the value held in a particular register was positive.

The multiple tape readers of the Mark II and the Bell machine played the same role as ENIAC's master programmer, allowing the basic sequences of instructions to be automatically invoked and repeated according to the needs of a specific problem. We have, however, found no evidence indicating that either approach was directly influenced by the other.

Sequential Electronic Memory

Under pressure to build a highly innovative machine on a war-time contract, the ENIAC team made two significant design decisions. Reasoning that computation at electronic speed required similarly fast access to the numbers being operated on, they built ENIAC's rewritable

[4] Mark II could be operated as one large machine or split to allow independent problems to be computed simultaneously on its two "sides".

store out of the only available electronic technology: vacuum tubes. The costs of this approach meant that the store held only 200 decimal digits split between 20 accumulators which, like Mark I's counters, did not simply store numbers but also implemented addition. Small though it was, this storage capacity enabled ENIAC to solve the ballistic equations thought likely to form the bulk of its workload. Secondly, as described above, its idiosyncratic programming system was designed to get around the fact that reading instructions from paper tape would be too slow.

In mid-1944 the team began to make plans for a new machine, soon code-named EDVAC. They specifically wanted to address ENIAC's small store and awkward set-up process. Von Neumann helped focus the proposal by identifying non-linear partial differential equations as a key application, important not only to the Manhattan project but also to BRL's on-going research programs. However, the numerical material required for the solution of these equations could not be economically stored in a vacuum tube memory. The first requirement for the new project was therefore to identify a fast storage device capable of holding large amounts of numerical data.

The Invention of Delay Line Memory

Since 1942, Moore School staff, principally Eckert and T. Kite Sharpless, had been working on an NDRC contract with MIT's Radiation Laboratory on various aspects of radar systems (Eckert and Sharpless 1945). They came across the liquid delay line developed by Bell Labs' William Shockley. Intended as a timing device for use in a range detection system, this promising idea had practical limitations, such as its weight. By June 1943 the Moore School team had developed a prototype delay line using mercury instead of the water and ethylene glycol mix used by Shockley. The Moore School device was demonstrated to Rad Lab staff, but work on the MIT contract ceased shortly afterwards as because the start of the ENIAC project in May diverted the efforts of the Moore School staff. Details of the mercury line were handed over to MIT, who continued to develop it for range detection and the "removal of ground clutter" in radar systems, applications which required the line delay the pulses travelling through it for a precisely specified period of time.

In the spring of 1944, Eckert returned to the mercury line and constructed a prototype which could be used as a storage device by adding simple circuitry to read the pulses emerging from the end of the line, reform them, and reinsert them at the other end of the tank for another traverse. This process of regeneration and recirculation turned the mercury line from a device which simply delayed a train of pulses into one which could preserve them indefinitely. Thus reconfigured, the mercury delay line seemed to hold out the promise of being a relatively cheap way to store and quickly retrieve large amounts of data.[5]

[5] This paragraph is based on the account given in (Burks n.d.).

The unification of memory

In August 1944, Eckert and Mauchly circulated a description of the invention (which they hoped to patent) to the ENIAC team explaining how delay lines or, more generally, "transmission line registers" might be used in a computer:

> The transmission line register is easily adapted to the storage of large amounts of numerical information […] A number of such registers may be employed as components in a computing machine. They may be used to receive from and read back into devices which do the actual computing. The pulses stored in the registers need not represent actual numbers, but may represent operations to be performed. (Such operations, or the code which represents them, may of course be interpreted as "code numbers" for the operations.) The pulses from a transmission line register may, for instance, be fed into an electronic stepping switch so as to operate a chosen circuit at a given time. (Eckert and Mauchly 1944, p. 5)

By providing a large, fast, rewritable store, delay lines could store coded instructions, already used on the Harvard and Bell Labs machines, and supply them to EDVAC at electronic speed. Delay lines therefore held out the promise of addressing both of ENIAC's perceived shortcomings.

Beginning in the autumn of 1944, the EDVAC project progressed on the assumption that the machine would be built around a delay-line memory. Unlike ENIAC's accumulators and Mark I's counters, these would simply store numbers which would be passed to separate devices to "do the actual computing." As Samuel Williams had in his March proposals for the new Bell Labs machine (Williams 1944), which had been communicated to the Moore School group, Eckert and Mauchly separated the two functions of storage and computation. The Moore School went further in suggesting that a single medium could store the different kinds of information held by a fast electronic machine. This was a radical departure from machines that read numbers from electromechanical or electronic counters, program instructions from tapes, and functions from tapes or resistor matrices.

Von Neumann joined the group as a consultant principally responsible for working on "logical control." In practice, this meant the question of how the machine's structure could be represented to the programmer and how a set of instructions could be designed to allow it to be efficiently used. His first systematic account of the issues was contained in the *First Draft of a Report on the EDVAC* (von Neumann 1945), the manuscript of which he sent to the group towards the end of April 1945.

Memory in the First Draft

At the beginning of the *First Draft*, von Neumann enumerated various things EDVAC would have to "remember": the instructions governing the course of the computation, intermediate numerical results, tabular representations of functions, and the numerical parameters of a problem (von Neumann 1945, pp. 4-6). Generalizing the suggestion made in Eckert and Mauchly's description of delay line storage, von Neumann proposed a single functional component to hold all these disparate kinds of information. He called this the computer's "memory." Most of the *First Draft* reflects the team's commitment to delay lines, but von Neumann also briefly discussed the use of iconoscopes as an alternative memory technology, as we discuss in Section 5.

The *First Draft* is notable for its abstract approach. Putting aside the details of pulses travelling through mercury, von Neumann described the basic "unit" of memory as simply the ability to "retain the value of one binary digit." After deciding that numbers could be stored in 32 bits, von Neumann commented that it was "advisable to subdivide the entire memory [...] into groups of 32 units, to be called *minor cycles*" (von Neumann 1945, p. 58). He called the contents of minor cycles, whether coded instructions or numbers, "code words", a phrase soon shortened simply to "word."

Memory is useless unless the stored information can be easily located. Existing machines did this in a variety of ways. The counters and registers of the tape-controlled machines had index numbers which appeared in instructions and controlled switches temporarily connecting the specified storage device to other units of the machine. Instructions, however, were sequenced by the physical properties of the medium storing them, being punched in consecutive tape positions or, as on ENIAC, physically linked by cabling. Tabulated function values were often retrieved by a linear search: Mark I's function tapes stored alternating arguments and function values and to look up a tabulated value a dedicated unit "hunted" for a supplied argument and then read the following value.[6] In contrast, ENIAC used two-digit function arguments to index its portable function tables. Finally, several machines provided read-only storage devices to hold numerical parameters. Typically, like Mark I's registers and the rows on ENIAC's constant transmitter, these were indexed and accessed in a similar way to the rewritable devices holding intermediate results.

Von Neumann defined a unified indexing system for EDVAC's delay line memory but at the same time recognizing the benefits of sequential access. The 32 bits making up a number were not individually indexed, and their meaning was determined by the fact that they were stored in a contiguous sequence. He further noted that "it is usually convenient that the minor cycles expressing the successive steps in a sequence of logical instructions should follow each other automatically" (von Neumann 1945, p. 76).

EDVAC's memory, as described in the First Draft, consisted of a battery of delay lines, each holding 32 minor cycles indexed by ordered pairs *(x, y)* where x identified a delay line and y a minor cycle within a line.[7] The two components of the index signify in very different ways, however. A value of x denotes a physical delay line which could be selected by means of switching in the familiar way. As the bits in the delay line recirculated, however, the minor cycle y would only be available for reading at one specific time within the line's overall cycle. The values of y therefore denote not fixed regions of space *from* which data could be copied, but periods of time *during* which it was available. As von Neumann put it,

> the DLA organs [delay line with amplifier] are definite physical objects, hence their enumeration offers no difficulties. The minor cycles in a given DLA organ, on the other hand, are merely moving loci, at which certain combinations of 32 possible stimuli may be located. Alternatively, looking at the situation at the output end of the DLA organ, a minor cycle is a sequence of 32 periods τ, this sequence being considered to be periodically returning after every 1,024 periods τ. One might say that a minor cycle is a 32τ "hour" of a 1,024τ "day", the "day" thus having 32 "hours". (von Neumann 1945, p. 89)

[6] The tapes of Turing's abstract machines of 1936 were accessed in a similar way: temporary marks were left in squares adjacent to the squares holding data of interest and later "hunted" for.

[7] These ordered pairs were written, problematically from a typist's point of view, $\mu\rho$ in the *First Draft*.

In his proposals for the ACE, Turing (1946) followed von Neumann's proposals for a delay-line memory and simply described the two components of *x* and *y* the index as the "delay line" and the "time" at which the desired minor cycle would be available. Von Neumann, however, struggled to find an intuitive way of describing the situation. He initially used purely spatial language: writing to Herman Goldstine in February, he described the "integers x, y which denote positions in the memory" rather vividly as "house numbers", before crossing the phrase out and replacing it by "filing numbers" (von Neumann 1945b). It seems very odd now to think of numbers living on a street, but that's the idea that has been naturalized in the phrases like "memory address" (just as it today seems strange to think of a computer as a brain, but natural to think of it having a memory).

By April, however, when the *First Draft* was written, von Neumann had moved away from spatial descriptions of minor cycles, a poor fit with the delay lines' dynamic properties, to the use of the temporal metaphors of "hours" and "days." Numbers were not stored in fixed and very concrete physical device, like Mark I's counters, but in "moving loci," abstract regions of space containing waveforms in a largely static medium. A locus may even consist of two non-contiguous regions, for example when a word is being transferred bit-by-bit from one end of a tank to the other. Just as midnight is used as the starting point to number the hours in a day, the minor cycles were numbered from an arbitrarily chosen point in the delay lines' timing cycle.

English offers more ways of describing spatial divisions than temporal ones. An alternative temporal metaphor was offered by Haskell Curry (1945), an early reader of the *First Draft* who noted that there was "no generally accepted term for the fundamental unit of time." He suggested using "beat" – "the accepted term for the fundamental time unit in music" – for the time taken for one pulse to emerge from a delay line, and considered carrying the metaphor further by referring to minor cycles as "measures" or "bars" before concluding that von Neumann's terminology of minor cycles was "just as good."

In an unpublished manuscript written shortly after the *First Draft*, von Neumann gave a more precise mathematical characterization of how temporal indexes would work (see Priestley 2018). A clock would keep a count as bits emerged from the delay lines, and from this the index of the word currently emerging from a line could easily be computed. Turing's ACE report described a similar scheme in somewhat more detail (Turing 1946).

Coding in the First Draft

In the *First Draft*, von Neumann described a new approach to automatic control which we call the "modern code paradigm." We have previously (Haigh, Priestley, and Rope 2014) summarized its key aspects as the following:

1. The program is executed completely automatically.
2. The program is written as a single sequence of instructions, known as "orders" in the *First Draft*, which are stored in numbered memory locations along with data.
3. Each instruction within the program specifies one of a set of atomic operations made available to the programmer.
4. The program's instructions are usually executed in a predetermined sequence.
5. However, a program can instruct the computer to depart from this ordinary sequence and jump to a different point in the program.

6. The address on which an instruction acts can change during the course of the program's execution.

This combined the fully automatic control of ENIAC (points 1 and 5) with the ordered sequence of coded instructions found on the tape-controlled machines (points 2-4). Exploiting the fact that instructions stored in EDVAC's unified memory were themselves indexed, switching to a different instruction sequence required nothing more than (in the language of the *First Draft*) "connecting" the central control organ to a different minor cycle by executing a transfer instruction specifying the address from which the next instruction should be read. This was simpler and more efficient than adding additional tape readers to hold routine tapes or wiring up a network of program controls to represent different execution pathways as on ENIAC.

One of von Neumann's motivations for insisting that memory had a default sequential ordering was to allow the machine's control to read an instruction sequence in a simple way (point 4). Tape-controlled computers moved naturally from one instruction to the next as the tape was moved physically through its reader. On EDVAC, instructions to be executed successively would be stored in consecutive minor cycles and the address of the next instruction would only have to be explicitly specified in the special case of a transfer order (point 5). As von Neumann (1945, p. 86) put it, "as a normal routine CC should obey the orders in the temporal sequence, in which they naturally appear at the output of the DLA organ to which CC is connected." This wording suggests that operations would be carried out at the same rate as instructions appeared, and hence that a computation could progress by simply taking the instruction emerging from a delay line once the preceding operation was completed.

However, there is a significant difference between tape readers and delay lines. Mark I's instruction tape reader only advanced when the machine's control recognized that the preceding operation was complete but EDVAC's instructions would emerge from the delay lines at a fixed rate. Von Neumann's preference for sequential storage of instructions may have been a reflection of the natural properties of instructions punched on paper tape, but its implementation could not rely solely on the physical properties of the delay lines. Even if an operation could be carried out as the next instruction was being read (which would have required buffering not explicitly described in the *First Draft*) this would mean that all operations had to be carried out in the time of one minor cycle. Von Neumann knew very well that this was not the case, estimating for example that multiplication would take around 30 minor cycles. There were only a few places in the code where the delay lines' temporal properties coincided with the operation being performed, for example in an instruction to copy a number from the arithmetic unit to the minor cycle immediately following the instruction.

Orders should not be obeyed in "the temporal sequence in which they naturally appear" at the output of a delay line after the completion of the previous instruction, then, but rather in a logical sequence defined by their addresses. Most machines patterned after EDVAC include a program counter, a dedicated memory location holding the address of the instruction currently being executed. This is incremented automatically, but can be manipulated to produce a jump. In the *First Draft*, von Neumann had not yet thought this through. He specified that the bus joining the "memory organ" to the "central arithmetic organ" should be "connected" to a specific location by modifying a number stored in the "central control organ" (in later computers this would be called the address register). Whenever an instruction transferred data from memory to the arithmetic organ the address register would be overwritten, with the result that EDVAC would lose its place in the program. Realizing the need to avoid this, von Neumann specified that during such "transient transfers" the address should be "remembered" (he did not say where) and restored before the next instruction was read. Like a dedicated program counter, that scheme would consume one extra storage location. Unlike a dedicated program counter, the scheme would waste time shuffling numbers in and out of the address register. In contrast, Turing's ACE report goes into considerable detail about how a short delay line CD (for "current data") would accomplish this task (Turing 1946).

Unfortunately, executing orders in the default sequence defined by their addresses causes substantial inefficiency. In most cases, the minor cycle holding the next instruction will not be the next to appear at the end of the delay line, and EDVAC's control would have to wait for the instruction to arrive. The delay would depend on the time taken to execute the preceding operation. Von Neumann was aware of this to some extent: the length of the delay lines (32 minor cycles) was chosen to minimize the delay in finding waiting for instructions that following multiplications. Perhaps this reflected the well-known observation that multiplication time dominated most computations, but in fact the delay incurred by operations which took only a handful of minor cycles to complete would be significantly longer than the statistically expected delay of half the cycle time of the delay line.

Short Delay Lines

One approach to reduce the inefficiency inherent in naïve use of delay lines was to make critical data immediately available by moving it into a separate "fast" store. In principle, this could use any suitable medium but the second iteration of the EDVAC design proposed the use of short delay lines. Storing only one minor cycle, these would be economical to construct and their timing properties would fit well with the rest of the machine. As Eckert and Mauchly (1945) described in a progress report in September 1945, EDVAC's memory would now contain a mixture of the original long lines and the new short lines. Similarly, the memory described by Turing (1946) for the ACE consisted of a mixture of long and short lines.

This development made the code more complex. In the summer of 1945, before coding an example merge routine, von Neumann developed a new instruction set containing a range of instructions for moving data between long and short lines (Priestley 2018). New instructions allowed sequences of words, rather than individual words, to be moved in one operation. The short lines were used for a variety of purposes: to hold the variables manipulated by the code, as a place to construct new instructions for immediate execution, and as temporary storage to hold data being moved from one location in the long lines to another.

The experience of detailed planning for the use of a delay-line machine therefore led to substantial modification in both machine design and programming technique and substantially complicated the intentionally simple and abstract design proposed in the *First Draft*.

Optimum Coding and the Pilot Ace

An alternative, and complementary, way to reduce the inefficiencies arising from storing instructions in delay lines was to take into account the expected time taken by each operation and organize the code so that the instructions and numbers did in fact appear at the end of delay lines exactly when required. Von Neumann attempted this in a limited way in his merge routine by intercalating blank instructions into his code at various points, but seems to have misunderstood the temporal properties of the delay lines (Knuth 1970).

The general approach became known as "optimum coding" and was embraced more fundamentally in the ACE project. In his original report, Turing (1946) proposed that consecutive program instructions be stored in alternate minor cycles. On the assumption that many of the machine's operations, such as addition or the transfer of a number, could be completed within one minor cycle, this would significantly reduce the waiting time.

The designers of the Pilot ACE, a simplified pilot version constructed a few years later, went further. Its minimalist architecture was optimized for speed, producing a small machine that could outpace much more expensive computers built on the model of EDVAC. To eliminate instruction decoding hardware and boost performance it eschewed conventional operation codes, instead treating all instructions as specifying a transfer between a "source" and a "destination." Sources and destinations could represent delay lines for storage, circuits that performed arithmetic or logical operations, or even pervasively useful constants such as 0 and 1 (Campbell-Kelly 1981). This mechanism did not allow individual minor cycles within delay lines to be specified: transferring a number from one line to another would simply copy it between the minor cycles appearing at the end of the lines when the instruction was executed. This eliminated any waiting time, but placed a large burden on programmers who had to consider the execution time of operations explicitly and to track exactly what would be emerging from each delay line at each moment during the execution of a program.

This went for instructions as well: the time at which an instruction arrived at the end of a delay line had to be coordinated with the arrival of the data it was manipulating. Instructions were not executed in a default sequence but were carefully scattered throughout in memory, the aim being to "eliminate all the waiting time associated with fetching instructions" (Campbell-Kelly 1981, p. 140). Each instruction nominated its successor by specifying a spatial index to identify a delay line and a temporal index to pick out a specific minor cycle. The temporality of the delay line was treated very differently from the *First Draft*, however, which had imposed what we would now call addressability on the delay lines, effectively tagging each piece of data with a location number that accompanied it as it moved through the delay line. This let programmers completely ignoring the actual temporality of the delay lines by writing code as if the memory consisted of fixed locations. In contrast, the Pilot Ace specified the position of the next instructions as a "wait" of so many minor cycles from the current position. If, as von Neumann had suggested, temporality could be understood via clock-based metaphors, an EDVAC programmer executing a jump would say "carry out the instruction that emerges from line 7 at 3 o'clock" while a Pilot ACE programmer would say, in each instruction, something like "carry out the instruction that emerges from line 7 in 10 minutes time." While the *First Draft* fixed time indexes to an agreed starting point, the Pilot Ace expressed everything relative to the current time.

The Pilot Ace approach was used for one the most successful early British commercial machines, the English Electric Deuce. However, most delay line machines, such as the EDSAC and the commercial Univac I, followed the EDVAC model and accepted complexity and suboptimal memory performance as the price to pay for ease of programming. Programmers could place data for optimum retrieval, if they chose, but instructions were fetched from sequential locations and so could not be optimally positioned. Some computers, including EDVAC as built, retained the addressable memory of the *First Draft* but added an additional address field to each location specify the next instruction to be executed. This permitted, but did not mandate, the use of optimum coding for instruction placement.[8]

Random-access Memory

The other potential memory technology mentioned in the *First Draft* was the iconoscope. Developed in the 1920s by RCA as a component for television cameras, iconoscopes were modelled on the retina. A light-sensitive electrostatic screen was imagined as a matrix of tiny capacitors each holding a charge proportional to the intensity of the light falling on the screen at that point. The matrix was scanned line-by-line with an electron beam which converted the charge at each point into a sequence of pulses. that could be transmitted to a remote location where an image would be reconstructed. Von Neumann imagined that a usable memory device could be constructed by placing charges on the screen with a second electron beam rather than by light.[9]

He noted that "in its developed form" an iconoscope could remember the state of 200,000 separate points, giving a single tube a similar storage capacity as EDVAC's array of 256 delay lines. The iconoscopes used in television scanned the screen in a fixed linear order, line by line, and therefore accessed memory units in a default temporal sequence which could be exploited, as with the delay lines, to store the bits in a minor cycle next to each other and instructions sequences in consecutive minor cycles. However, the electron beam reading the charges could also be rapidly switched to any point on the screen, meaning that arbitrary transfers could be carried out without the delay caused by waiting for the desired minor cycle to emerge from a delay line. Iconoscopes seemed to be an ideal memory technology, providing "a basic temporal sequence of memory units" but also the ability to "break away from this sequence in exceptional cases" without penalty (von Neumann 1945, p. 77).

Memory devices based on iconoscopes never became a reality. When the IAS computer project began, in November 1945, it was assumed that the memory would use delay lines, as in the latest version of the EDVAC proposals (IAS 1945). At the beginning of 1946, however, RCA (who was one of the project partners) proposed a novel type of storage tube, the Selectron (Rajchman 1946). These tubes would have the crucial property of being able to switch to read any point without delay, and design progressed on the assumption that it would be feasible to develop tubes with a capacity of 4,096 bits within the timeframe of the project (Burks et al. 1946, p. 9).

[8] It became common for machines using delay lines or magnetic drums as their primary memory to include an additional address to code the location of the next instruction, thus allowing optimum coding. However that was not the only motivation. EDVAC's designer (Lubkin 1947) justified the additional address by saying it would make programs easier to change, not as a way to improve operational efficiency.

[9] See the detailed discussion in (von Neumann 1945 73-8). It is a curiosity that at this point von Neumann seems to be envisaging a "memory organ" based on the structure of the vertebrate eye.

The modest capacity of the proposed Selectrons meant that to have a sufficiently large memory the IAS machine would require an array of tubes. However, the team chose to organize this memory in a novel way. The bits in a minor cycle would no longer be stored contiguously and read in accordance with the default behaviour of the storage device. Rather, each bit would be stored in a different Selectron. Reading a word would involve reading one bit from each Selectron, the bits being located at corresponding positions in the tubes. The bits in a word would no longer be read serially but in parallel. The team believed this would be faster and require simpler switching circuits.

While it seemed very promising, the Selectron tube turned out to be very hard to produce and the final version held only 256 bits. This was used in only one computer, the RAND corporation's version of the IAS machine known as the JOHNNIAC (Ware 2008). The IAS machine itself was eventually constructed using an alternative technology developed at the University of Manchester, and known as the "Williams tube" after its inventor. Rather than trying to develop a completely new type of tube, Williams drew on his wartime experience of radar and built a functionally similar device based on standard cathode ray tubes.

In June 1946 von Neumann and his collaborators produced what they termed a *Preliminary Discussion* of the design of the IAS machine and its code (Burks et al. 1946). In fact, the code changed only in details thereafter and for many computer builders this document, rather than the *First Draft*, was the seminal reference on computer design. Rather than the two-dimensional spatial and temporal indexes used to identify minor cycles in the *First Draft*, the IAS machine's memory was conceptualized as a simple sequence of words indexed by a single integer. The instruction set was much simpler that the one von Neumann had proposed for the mid-1945 version of the EDVAC including short delay lines, and its basic capabilities were stripped back almost to the level of the *First Draft* code. The major difference arose from the different organization of the arithmetic unit and the provision of a reasonably extensive range of arithmetic orders, presumably intended to ease coding of the machine's core applications.

The *Preliminary Discussion*, then, conceptualized memory in purely spatial terms. This was a considerable departure from the *First Draft* with its mixture of spatial and abstract temporal coordinates. With von Neumann's embrace of tube storage and the assumption that any word could be accessed with equal efficiency, the temporal aspect of memory so prominent in delay line storage dropped out of consideration altogether and memory was imagined to be a consecutive array of words accessed by a single index, or address, which it was natural to imagine in spatial terms. There was no need for programmers to manage the complexities of distributing data between long and fast delay lines or to grapple with optimum coding. Later readers, ourselves initially included, have therefore tended to read the *First Draft* as if its system of coordinates represent an "address space" or specify "memory locations" – both spatial models. The abstraction of its coordinates makes that reinterpretation easier, but in reality, spatial metaphors of this kind were not generally applied to delay line memories. Well into the 1950s, materials describing commercial delay line machines, such as the Univac I, continued to talk of memory as being structured into "major cycles" rather than "locations."

The spatial model of memory remained a considerable abstraction of the physical reality of storage in the IAS machine. The bits in a word were not stored contiguously, but were distributed across all the storage tubes. The "address" of a word did not denote a box-like region of space, but rather a fragmented and distributed locus of small regions on an array of physical devices. Von Neumann had reintroduced the "house number" metaphor of memory indexes in the early stages of the IAS project, when it was still assumed that the machine would use delay lines (IAS 1945), but dropped it thereafter. The same metaphor was revived, or independently rediscovered, by Max Newman of Manchester University. In 1948 he spoke to the Royal Society about "storing numbers […] in certain places or 'houses' in the machine." However, Newman focussed on access to the stored data as much as its physical properties and spoke of needing an "'automatic telephone exchange' for 'selecting 'houses,' connecting them to the arithmetic organ, and writing the answers in other prescribed houses" (Newman 1948).

The *Preliminary Discussion* presented a simple spatial abstraction of memory as a sequence of box-like containers of data. Like all good abstractions, this model could be implemented in many different ways and served to insulate the business of programming from inevitable changes in the development of memory technology. The model could be applied as well to serial delay-line machines such as EDSAC as to the parallel computers modelled on the IAS machine, and remained essentially unchanged when both technologies were replaced by magnetic core storage in the 1950s. It was even retro-fitted to EDVAC itself. By 1947 its evolving design used a single index for minor cycles that completely hid the temporal properties of its delay lines. As Samuel Lubkin put it, "the number […] representing the location of a word, will be referred to as an `address' or `register number'" and addresses hid the "precise arrangement of the memory" allowing programmers to think that numbers are "actually stored in individual registers" (Lubkin 1947, 10-11). From this point on, programming technique could develop autonomously and relatively unaffected by changes in hardware, a point made by Maurice Wilkes and his collaborators who observed that techniques developed for EDSAC could "readily be translated into other order codes" (Wilkes et al 1951). Some delay-line machines continued to rely on optimum programming techniques, as did certain later machines with magnetic drum memories, but as memory technology evolved, these rapidly became obsolete.

Conclusions

We have examined the evolution of the media used to store programs in the period in the 1940s when the collection of ideas constitutive of the modern computer was coming together, and considered the effect of these media choices on coding and code design. This reveals a dialectical and emergent relationship between the development of storage technology and more abstract ideas about coding. Although described by both von Neumann and Turing as a new kind of logic, the development of computer programming did not follow the path of implementation of a well-formed theoretical idea but was always responsive to developments in memory technology.

We mentioned the "stored program concept" at the beginning of this chapter, and the media-inflected origins of the term are worth re-emphasizing here. The term "stored program" in this context can be traced to a prototype electronic computer assembled in 1949 by an IBM team led by Nathanial Rochester.[10] The Test Assembly, as the machine was usually known, was a

[10] See (Haigh et al 2014). We have subsequently discovered the following occurrence of "stored program" in a 1946 draft of Goldstine and von Neumann's *Planning and Coding* reports: "acoustic or electrostatic storage devices will […] provide […] the possibility to modify (erase and rewrite) stored program information under the machine's own control". This usage is adjectival rather than substantive, however, and does not appear in the published reports. It

mash-up of existing components, including the IBM 604 Electronic Calculating Punch. It could read program instructions from two different media: 60 instructions could be set up on the 604's plugboard, but there was also a magnetic drum which could hold 250 numbers or coded instructions. To distinguish between the two sources of instructions, the team began to refer to instructions held on the drum as the "stored program" (Rochester 1949). IBM disseminated the phrase in its marketing (IBM 1955) for the IBM 650 computer which likewise partitioned control information between a magnetic drum (holding the "650 stored program") and traditional punched card controls. Over the following decade the phrase gradually became established as a way of referring to the class of machines originally described more clumsily as "EDVAC-type machines." In its original use, however, it was not intended to mark any deep theoretical insight, but simply to distinguish between two media on a largely unknown experimental machine.

The turning point in the story we have told occurs in early stages of the EDVAC project, when the rather disparate collection of media used to store numbers and instructions on earlier machines was replaced by the conception of a single, internal memory storing different types of information. We have traced in some detail the tension in and around the *First Draft* between the abstract model of memory, what would later be called an "address space," and the temporal properties of delay line memory.

This highlights a tension between spatial and temporal modes of thinking that recurs in the relationship between program instructions and the operations that are executed. Initially conceived as a fairly simple relationship between corresponding sequences of instructions and operations, by 1947 the coded instructions placed in memory were seen as merely the starting point of a complex process that could generate an extremely long and complex sequence of operations, in the process also altering the coded instructions themselves.

By the mid-1950s, computer designers had settled on core memory as a new medium to replace both delay lines and display tubes. Like display tube storage this had a straight forward spatial organization of data, rather than the fundamentally temporal structure of the delay line. This eliminated the need for optimum coding as practiced with drum and delay line machines. In a broader sense, optimizations based on knowledge of the actual hardware underlying a simple instruction set and memory model never went away, as programmers and compiler creators struggled to optimize the performance over time of vector instructions, pipelines, and cache mechanisms.

References

Anonymous. 1944. ENIAC progress report dated June 30, 1944 In *MSOD-UP, b1*.

Bloch, Robert. 1984. Oral History Interview with William Aspray, Februrary 22 (CBI OH 66). Charles Babbage Institute, Minneapolis, MN.

Burks, Arthur W. n.d. *Unfinished Book Manuscript*: n.p.

is therefore unlikely to have inspired the use of the phrase within IBM in 1949, and we do not believe that this materially affects our earlier discussion of the topic.

Burks, Arthur W, Herman Heine Goldstine, and John von Neumann. 1946. *Preliminary Discussion of the Logical Design of an Electronic Computing Instrument*. Princeton, NJ: Institute for Advanced Studies, 28 June 1946.

Campbell-Kelly, Martin. 1981. "Programming the Pilot Ace: Early Programming Activity at the National Physical Laboratory." *Annals of the History of Computing* no. 3 (2):133-162.

Chun, Wendy Hui Kyong. 2011. *Programmed Visions: Software and Memory*. Cambridge, MA: MIT Press.

Cohen, I Bernard. 1999. *Howard Aiken: Portrait of a Computer Pioneer*. Cambridge, MA: MIT Press.

Cope, W. F. and Hartree D. R. The Laminar Boundary Layer in Compresible Flow. Philosophical Transactions of the Royal Society of London. Series A. 241(827), 1-69. (June 1948).

Copeland, B Jack. 2013. *Turing: Pioneer of the Information Age*. New York, NY: Oxford University Press.

Copeland, B. Jack, Andre A. Haeff, Peter Gough, and Cameron Wright. 2017. "Screen History: The Haeff Memory and Graphics Tube." *IEEE Annals of the History of Computing* no. 39 (1):9-28.

Croarken, Mary. 2003. "Tabulating the Heavens: Computing the Nautical Alamanac in 18th-Century England." *IEEE Annals of the History of Computing* no. 25 (3):48-61.

Curry, Haskell. 1945. Letter to John von Neumann, August 10, 1945 (JvN-LoC b3f2).

Davis, Martin. 2001. *Engines of Logic: Mathematicians and the Origin of the Computer*. New York, NY: Norton.

Eckert, J. P., Jr, and J. W. Mauchly, 1944. "Use of acoustical, electrical, or other transmission line in a device for the registering of pulses, the counting, switching, sorting and scaling of pulses, and the use of such devices for performing arithmetic operations with pulses". August 1944, HHG-APS box 21.

Eckert, J. Presper, and John W. Mauchly. 1945. Automatic High Speed Computing: A Progress Report in the EDVAC. Report of Work Under Contract No. W_570_ORD_1926, Supplement No 4. (Plantiff Exhibit 3540). September 30. In *ENIAC Patent Trial Collection, UPD 8.10, University of Pennsylvania Archives and Records Center, Philadelphia, PA.*

Eckert, J. P., and T. K. Sharpless. 1945. Final Report under Contract OEMsr 387. Moore School of Electrical Engineering, University of Philadelphia, November 14, 1945. (Britton Chance papers, APS, box 80, folder 7.)

Goldstine, Herman H, and John von Neumann. 1947. *Planning and Coding Problems for an Electronic Computing Instrument. Part II, Volume 1*. Princeton, NJ: Institute for Advanced Studies.

Grier, David Alan. 1996. "The ENIAC, the Verb "to program" and the Emergence of Digital Computers." *IEEE Annals of the History of Computing* no. 18 (1):51-55.

Grier, David Alan. 2006. *When Computers Were Human*. Princeton, NJ: Princeton University Press.

Haigh, Thomas, and Mark Priestley. 2016. "Where Code Comes From: Architectures of Automatic Control from Babbage to Algol." *Communications of the ACM* no. 59 (1):39-44.

Haigh, Thomas, Mark Priestley, and Crispin Rope. 2014. "Reconsidering the Stored Program Concept." *IEEE Annals of the History of Computing* no. 36 (1):4-17.

Haigh, Thomas, Mark Priestley, and Crispin Rope. 2016. *ENIAC In Action: Making and Remaking the Modern Computer*. Cambridge, MA: MIT Press.

IBM. *IBM 650 Technical Fact Sheet, July 20*. IBM Archives 1955.

IAS. 1945. Minutes of E.C. Meeting, November 19. Institute of Advanced Studies. In *Herman H. Goldstine Papers (box 27)*: American Philosophical Society, Philadelphia, PA.

Knuth, Donald E. 1970. "Von Neumann's First Computer Program." *ACM Computing Surveys* no. 2 (4):247-260.

Lubkin, Samuel. 1947. Proposed Programming for the EDVAC. Moore School of Electrical Engineering, University of Philadelphia, January 1947 (MSOD box 8).

Newman, M H A. 1948. "General Principles of the Design of All-Purpose Computing Machines." *Proceedings of the Royal Society of London, Series A* no. 195:271-274.

Priestley, Mark. 2018 (forthcoming). *Routines of substitution: John von Neumann's work on software development, 1945 -1948*. (Springer, 2018).

Rajchman, Jan. 1946 The Selectron. In Cambell-Kelly, M., and Williams, M. R. *The Moore School Lectures*. MIT Press, 1985.

Rochester, Nathaniel. 1949. A Calculator Using Electrostatic Storage and a Stored Program. IBM Archives.

Staff of the Harvard Computation Laboratory. 1946. *A Manual of Operation for the Automatic Sequence Controlled Calculator*. Cambridge, MA: Harvard University Press.

Stibitz, George R. 1967. "The Relay Computers at Bell Labs." *Datamation* no. 13:??

Swade, Doron. 2001. *The Difference Engine: Charles Babbage and the Quest to Build the First Computer*. New York: Viking Penguin.

Turing, Alan 1946. Proposed Electronic Calculator. NPL. Reprinted in Carpenter, B. E. and Doran, R. W (1986), *A. M. Turing's ACE Report of 1946 and Other Papers* (MIT Press).

von Neumann, John. Letter to Robert Oppenheimer, 1 August 1944 (Los Alamos National Laboratory, LA-UR-12-24686).

Von Neumann, John. 1945b. Letter to Herman Goldstine, February 12, 1945 (HHG-APS box 9).

von Neumann, John, 1945. *First Draft of a Report on the EDVAC*. Moore School of Electrical Engineering, University of Pennsylvania, June 30, 1945.

Ware, Willis H. 2008. *RAND and the Information Evolution: A History in Essays and Vignettes*. Santa Monica, CA: RAND Corporation.

Wilkes, M., Wheeler, D. J., Gill, S.. 1951. *The Preparation of Programs for an Electronic Digital Computer*. Addison-Wesley.

Williams, S. B. 1944. "Calculating System". Bell Telephone Laboratories, March 29, 1944 (HHG-APS box 20).

Zuse, Konrad. 1993. *The Computer--My Life*. Berling/Heidelberg: Springer-Verlag.

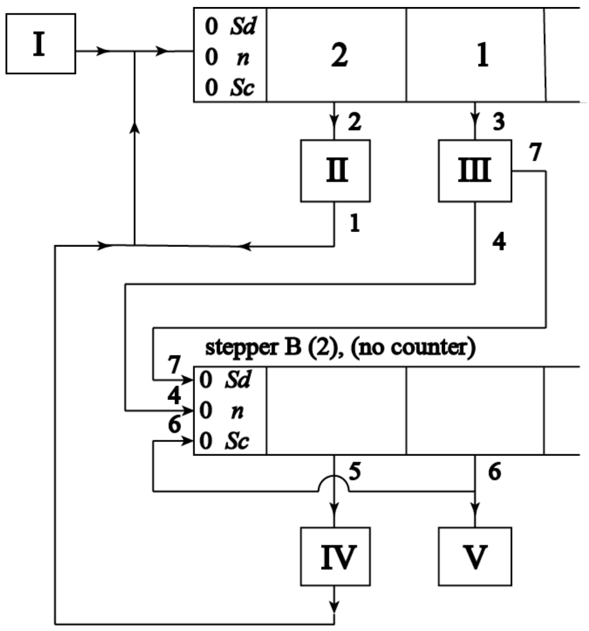

Fig 1. Douglas Hartree included this master programmer diagram in a published paper describing a computation performed on ENIAC (Cope and Hartree 1948). It describes both the configuration of the master programmer and the overall structure of the computation. (source: W. F. Cope and Douglas R. Hartree, "The Laminar Boundary Layer in Compressible Flow," *Philosophical Transactions of the Royal Society of London. Series A; Mathematical and Physical Sciences* 241, no. 827 (1948): 1–69; reproduced with permission of Royal Society)