Von Neumann's First Computer Program

DONALD E. KNUTH

Stanford University, Stanford, California*

An analysis of the two earliest sets of instruction codes planned for stored program computers, and the earliest extant program for such a computer, gives insight into the thoughts of John von Neumann, the man who designed the instruction sets and wrote the program, and shows how several important aspects of computing have evolved. The paper is based on previously unpublished documents from the files of Herman H. Goldstine.

Key words and phrases: electronic computers, computer history, stored program computers, machine organization and architecture, sorting, latency time, ENIAC, EDVAC, order code, programming techniques

CR categories: 1.2, 6.0

INTRODUCTION

A handwritten document now in the possession of Dr. Herman H. Goldstine contains what is probably the earliest extant program for a stored program digital computer. Its author, the remarkably talented mathematician John von Neumann (1903–1957), was in the process of refining the stored program concept as he was writing this code; so his program represents a significant step in the evolution of computer organization as well as of programming techniques. In this paper we will therefore investigate the contents of von Neumann's manuscript in some detail, attempting to relate its ideas to other developments in the early history of high speed computing.

The program we will study is not what we might expect an "ordinary" mathematician to have written; it does not solve a partial differential equation! Instead, it deals with what was considered at that time to be the principal example of a nonnumeric application for computers, namely, the problem of sorting data into nondecreasing order.

Von Neumann chose this application for good reason. He had sketched out an order code for a stored program computer, with numerical applications uppermost in his mind; there was no question that his proposed device could do the requisite arithmetic operations. The key question was whether or not the proposed instruction set provided a satisfactory means of logical control for complex processes, and so he felt that a sorting program would be a most instructive test case. Furthermore, the existence of IBM's special purpose machines for sorting gave him a standard against which he could measure the proposed computer's speed.

Before we start to study von Neumann's program, a few disclaimers are in order. In the first place, he probably never intended to have this program published and subjected to such scrutiny; although his manuscript is carefully documented, he probably wanted only to circulate it among a few interested colleagues. So when we find a few errors and a few instances of clumsy coding, we should realize that it was an early effort that was not supposed to represent a polished product.

Second, we should realize that the historical interest of this program is in great measure due to its connection with the development of instruction codes for stored

* Computer Science Department

CONTENTS

program computers; it is not the earliest instance of a computer program. We have Lady Lovelace's description of a program for calculating Bernoulli numbers that Babbage wrote for his Analytical Engine [1, Note G]; A. M. Turing's construction [16] of his abstract Universal Machine, which involves many important programming concepts; Eckert and Mauchly's first sample program for the ENIAC [4]; and a collection of numerical programs, dating from 1944, written by H. H. Aiken, G. M. Hopper, R. V. D. Campbell, R. M. Bloch, B. J. Lockhart, and others, for the Harvard Mark I [10, Chs. 4, 6].

A third precaution: The notation used in this paper differs considerably from that used by von Neumann, so that modern readers can more easily understand what he did. Where he would write, for instance,

$$\overline{1}_5)\ c + (m' - 1)(p + 1) \nrightarrow \overline{1}_4 \mid p + 2,$$

we will use an equivalent assembly-like language form,

MOVEIN PIK p+1,BUFFER,[YPTR].

This new notation isn't completely transparent, but it seems to be an improvement which doesn't go so far that the machine is obscured. (For further information about von Neumann's original notation, see the section on Storage Allocation and Timing.)

To set the stage for our story, let us consider briefly the developments prior to 1945 when von Neumann wrote his sorting program. Several electromechanical calculators were essentially operational in the late 1930s and early 1940s: those of Stibitz [15] and Aiken [10] in America, and Zuse [3] in Germany. Another machine, which had electronic circuitry for arithmetic although it was slowed down by mechanical memory elements, was also developed in the late 1930s at Iowa State College, by John V. Atanasoff and Clifford E. Berry [see 12]; this machine was designed to solve sets of simultaneous linear equations.

In August 1942, John W. Mauchly of the Moore School of Electrical Engineering in Philadelphia wrote a memorandum to his colleagues summarizing briefly the ad-

vantages which could be expected from an electronic high speed computer such as he felt could reasonably be developed. He was familiar with previous American developments in computing, and he was also aware of the extensive calculations needed by the Ballistic Research Laboratory (BRL) in connection with World War II; many of these calculations were currently being done on a Differential Analyzer at the Moore School. It was by no means obvious that a useful electronic computer could be built; but Mauchly and a young electrical engineer named J. P. Eckert, Jr., drew up a tentative technical outline of a suitable machine, and Prof. J. G. Brainerd decided it was worth the risk of committing the Moore School to a major effort in this direction. A technical proposal was submitted to Col. Leslie E. Simon, Col. Paul N. Gillon, and Lt. Herman H. Goldstine of the BRL in the spring of 1943, and in a remarkably short time the US government entered into a contract with the Moore School for research and development of high speed electronic calculating devices, beginning July 1, 1943. The project, supervised by Brainerd, had Eckert as chief engineer, Mauchly as principal consultant, and Goldstine in charge of technical liaison with BRL. A first machine, the ENIAC (Electronic Numerical Integrator And Computer), was soon designed, and its design was frozen at an early stage so that future efforts could be concentrated on its production and testing; it was dedicated on February 15, 1946. (For further details about the development of the ENIAC, see [6].)

The ENIAC was a highly parallel computer; weighing over 30 tons, it involved over 19,000 vacuum tubes, 1500 relays, etc. Because of its electronic circuitry, it was considerably faster than any computing machine previously built. But it had only 20 words of internal memory, and it required complicated manual operations for setting up a program on plugboards. Long before ENIAC was completed, it became clear to the designers that they could utilize the equipment more efficiently if they would adopt serial methods instead of so much parallelism; so in January 1944 they sketched out a "magnetic calculating machine" in which successive digits of numbers were transmitted serially from a memory device to central electronic computing circuits and back again. Early in 1944, Eckert and Mauchly invented an acoustic delay-line memory device which made it possible to obtain a fairly large storage capacity with comparatively little hardware; so it became evident that great improvements over ENIAC could be made, at considerably less cost. "Therefore, by July, 1944, it was agreed that when work on the ENIAC permitted, the development and construction of such a machine should be undertaken. This machine has come to be known as the EDVAC (Electronic Discrete VAriable Computer)" [5].

In the latter part of 1944, John von Neumann (a consultant to BRL) became a consultant to the EDVAC project. He contributed to many discussions on logical circuitry, and he designed the order code which was to be used. In the spring of 1945, he wrote a preliminary report [17] which gives a detailed discussion of arithmetic circuitry and the motivation for various design decisions which were made as EDVAC evolved. This takes us to the beginning of our story.

THE EARLY EDVAC

Von Neumann's first draft report [17, 18] on the EDVAC proposed building a serial computer with three 32-bit registers and 8192 32-bit words of auxiliary memory. The three registers were called i and j (for inputs to the arithmetic circuitry) and o (for output); for convenience in what follows we will denote these registers by the upper-case letters I, J, and A. The EDVAC memory was to be divided into 256 "tanks" of 32 words each, operating in a cyclic fashion. Word 0 of each tank would pass a reading station one bit at a time, then (32 bit-times later) word 1 would be available, . . . , finally word 31, then word 0 again, etc. Thus the accessing of information from tanks is essentially the same as we now have from drums or head-per-track disks. A bit-time was to be 1 μsec, so the cycle time for each tank came

to $32 \times 32 = 1024$ μsec. The tanks were to be constructed from Eckert and Mauchly's mercury delay lines; this concept was later used in the memory of the UNIVAC I computer (1951). Although von Neumann realized that faster operation could be achieved with a random-access memory, the only known way of building such memories economically was the "iconoscope" (like a TV tube, with light or dark spots created and sensed by an electron beam), and he temporarily rejected it since its reliability was still unproved.

Each 32-bit word was either a number or an instruction code; the first bit was 0 for numbers and 1 for instructions. Von Neumann suggested writing binary numbers in reverse order, with the least significant digits at the left, since binary notation was unfamiliar anyway and since the serial circuitry found it most convenient to process least significant digits first. The last bit of a number, following the most significant bit, was its sign; numbers were represented in two's complement notation. Thus the word

$$b_0b_1b_2b_3 \cdots b_{30}b_{31}, \qquad b_0 = 0,$$

denoted the number

$$2^{-30}b_1 + 2^{-29}b_2 + 2^{-28}b_3 + \cdots + 2^{-1}b_{30} - b_{31},$$

and 30-bit fractions in the range $-1 \leq x < 1$ were representable. For addition, subtraction, and conversion operations, the number could also be regarded as the integer

$$b_1 + 2b_2 + 4b_3 + \cdots + 2^{29}b_{30} - 2^{30}b_{31},$$

so that integers in the range $-2^{30} \leq x < 2^{30}$ were representable. Binary coded decimal integers $(abcdefg)_{10}$ were also allowed, in the form

$$0\,0\,0\,0\; a_1a_2a_3a_4b_1b_2b_3b_4 \cdots g_1g_2g_3g_4,$$

where $a_1a_2a_3a_4$ was the code for digit a, and $b_1b_2b_3b_4$ was the code for digit b, etc., in reverse binary order. (Thus 0000 = 0, 1000 = 1, 0100 = 2, $\cdots$, 0001 = 8, 1001 = 9.)

Instruction words were to have the form

$$1\,a_0a_1a_2a_3a_4b_0b_1b_20000000000y_0y_1y_2y_3y_4x_0x_1x_2x_3x_4x_5x_6x_7,$$

where $a = a_0a_1a_2a_3a_4$ denoted an operation code, $b = b_0b_1b_2$ denoted a variant, $y = y_0 + 2y_1 + 4y_2 + 8y_3 + 16y_4$ denoted a word position within a tank, and $x = x_0 + 2x_1 + \cdots + 128x_7$ denoted a tank number. The following arithmetic operation codes were proposed, affecting the registers I, J, and A:

- AD (a = 00000). Set $A \leftarrow I + J$.
- SB (a = 00001). Set $A \leftarrow I - J$
- ML (a = 00010). Set $A \leftarrow A + I \times J$ (rounded).
- DV (a = 00011). Set $A \leftarrow I/J$ (rounded).
- SQ (a = 00100). Set $A \leftarrow \sqrt{I}$ (rounded).
- II (a = 00101). Set $A \leftarrow I$.
- JJ (a = 00110). Set $A \leftarrow J$.
- SL (a = 00111). If $A \geq 0$, set $A \leftarrow I$; if $A < 0$, set $A \leftarrow J$
- DB (a = 01000). Set $A \leftarrow$ binary equivalent of decimal number I.
- BD (a = 01001) Set $A \leftarrow$ decimal equivalent of binary number I.

(As stated in the Introduction, we are changing von Neumann's notation; the mnemonic symbols for these codes are ad hoc symbols contrived solely for the purposes of the present paper. Note that multiplication, division, and square root were to produce *rounded* results. Not all details of these operations were fully specified by von Neumann; division and square root would change the contents of I, but it is not clear that a valid remainder would be left there. The decimal-to-binary and binary-to-decimal operations were not worked out. Apparently overflow conditions caused no special action.)

Each of the above arithmetic operations was to be used with one of several variants specified by $b = b_0b_1b_2$:

- H (b = 111). Do the operation as described above, holding the result in A.
- A (b = 100) Do the operation as described above, then set $J \leftarrow I$, $I \leftarrow A$, $A \leftarrow 0$.
- S (b = 000). Do the operation as described above, then store the result A in memory location yx and set $A \leftarrow 0$.
- F (b = 101). Do the operation as described above, then store the result into the word immediately following this instruction, set $A \leftarrow 0$, and perform the altered instruction.
- N (b = 110) Do the operation as described above, then store the result into the word immedi-

ately following this instruction, set $A \leftarrow 0$, and *skip* the altered instruction.

Thus, for example, ADS yx would have the effect of setting location yx to $I + J$, and clearing A to zero; JJA would interchange the contents of I and J, and clear A; SLH would set A to either I or J, according as the previous sign of A was 0 or 1. (The memory specification yx was ignored on all variants except S.)

Besides arithmetic operations, the machine could do the following:

- JMP (a = 11000, b = 000). Take the next instruction from location yx (then $1 + yx$, etc).
- LOD (a = 10000, b = 000). Set $J \leftarrow I$, then set I to the contents of memory location yx.

Further codes a = 01010, 01011, 01100, 01101, 01110, 01111 were reserved for input and output operations (which were not yet specified) and stopping the machine.

There was an important exception to the operations as we have described them: Only numbers (not instructions) ever appeared in the registers I, J, and A. When the LOD operations specified a memory address containing an instruction, only the yx part of that instruction was to be loaded into I; the other bits were cleared. Conversely, when storing into memory by means of variants S, F, and N, only the least significant 13 bits of the number in A were to be stored in the yx part, if the memory location contained an instruction word.

Instructions were to be executed from consecutive locations, unless the sequence of control was modified by a JMP order. If the control sequence would come across a number (not an instruction word), the effect would be as if an LOD instruction were performed referring to this number.

Most instructions would be performed in one word-time, so that the machine could keep up with the speed of the long tanks where instructions were stored. But multiplication, division, square root, and radix conversion took 33 word-times (1056 μsec). References to memory, by means of LOD operations and the S variant, would require an additional 1024 μsec unless the memory address was perfectly synchronized to match the following word of instructions. (For multiplication, division, and square root extraction, there was a little more leeway, since those operations actually were completed in about 950 μsec.)

The reader will note that much of the space in instruction words is wasted. Von Neumann realized this, but did not think it important at the time, since [17, p. 96] the programming problems he had considered required only a small fraction of the memory for instruction storage. But we will see that he changed his mind later.

The machine we have considered here differs slightly from von Neumann's description in [17], since the modifications stated in his letter [18] have been included. He wrote, from Los Alamos to Philadelphia, "The contents of this letter belong, of course, into the manuscript [17], and I will continue the manuscript and incorporate these things also, after I get it back from you—if possible with comments . . . from you, Pres Eckert, John Mauchly, and the others." But the manuscript was never completed, nor were the modifications inserted when it was typed a month later, presumably because there were so many other things to be done. It is interesting to note that von Neumann's letter [18] also proposed the design of a special typewriter for preparing programs from partially mnemonic input. Pushing a key marked + would cause the bits 100000 to be assembled (the first six bits of an addition instruction); then a key marked H would insert the subsequent bits 111000...00, forming a complete instruction word on a magnetic tape.

The differences between [17] and the machine described here are chiefly concerned with improvements in the logistics of instruction modification. (a) There was no variant N; instead, variant F would not treat the altered word as an instruction if it turned out to be a number. (b) The convention on loading only 13 bits of instructions was not present (although the convention about storing only 13 bits into instruction words was). (c) Three other variants, like S, A, F but not clearing register A, were originally included.

THE NEXT EDVAC

Von Neumann's letter says, "I have also worked on sorting questions.... I will write you about the details very soon." He said that he had written an internal sorting program requiring about 130 words of instructions; it could sort 500 p-word items on a one-word key in about $1 + .425\,(p - 1)$ minutes. "I suspect that these arrangements, which represent only a first attempt, could be improved....

"At any rate the moral seems to be that the EDVAC, with the logical controls as planned for 'mathematical' problems, and without any modifications for 'sorting' problems, is definitely faster than the [contemporary IBM sorters, about 400 cards/minute].... Since the IBM's are really very good in sorting, and since according to the above, sorting can be meshed with the other operations of the EDVAC without human intervention or need for additional equipment, etc., the situation looks reasonably satisfactory to me.... It is legitimate to conclude already on the basis of the now available evidence, that the EDVAC is very nearly an 'all-purpose' machine, and that the present principles for the logical controls are sound."

But von Neumann's code for this sorting program does not seem to have survived; we can only say that his timing estimates look reasonable, since for large p they come to slightly over 5 msec per pass per word transferred. The program which now is in Dr. Goldstine's files is roughly 80 times faster, due to important improvements in machine organization which von Neumann considered shortly afterward. This second EDVAC design was apparently never defined in as much detail as the previous one, but a brief summary of its instruction codes appears in [5, p. 76] and we can deduce other properties by studying von Neumann's program. Therefore we can reconstruct the main features of the machine.

The chief improvement incorporated into this version of EDVAC was the introduction of "short tanks" whose capacity was one word each; this provided a small fast-access memory which essentially increased the number of registers, and the old I and J disappeared. Block transfer operations between the short and the long tanks made many processes faster. The tentative plans in [5] call for 32 short tanks, and 2048 additional words in 64 long tanks. "Combining this with the almost unlimited memory capacity of the magnetic tape (even though the numbers are not available here so quickly) it seems that very few problems will exceed this capacity" [5, p. 81].

Here are the basic operations allowed by the new EDVAC code, exclusive of multiplication and division [let $C(s)$ denote the contents of short tank number s]:

- PIK s,t,x Transfer s consecutive words, starting at long tank location x, to s consecutive short tanks, starting at short tank number t. If x is unspecified, the next s words following this instruction are used, and the $(s + 1)$-th is the next instruction
- PUT s,t,x Transfer s consecutive words, starting at short tank number t, to s consecutive long tank positions starting at location x. If x is unspecified, the next s words following this instruction are used, and the $(s + 1)$-th is the next instruction.
- ADD s,t. Set $A \leftarrow C(s) + C(t)$.
- SUB s,t. Set $A \leftarrow C(s) - C(t)$.
- SEL s,t If $A \geq 0$, set $A \leftarrow C(s)$; if $A < 0$, set $A \leftarrow C(t)$
- TRA x. Go to long tank location x (then $x + 1$, etc.) for subsequent instructions
- JMP s. Go to short tank number s (then $s + 1$, etc) for subsequent instructions.
- STO s. Set $C(s) \leftarrow A$.
- SET s,t Set $C(s) \leftarrow C(t)$

As before, operations which did not refer to long tank addresses took just one word-time (32 μsec), with the exception of "long" arithmetic operations like multiplication and division. When a long tank location was specified, the machine waited until the desired word was accessible; at least two word-times were needed for the instruction TRA x, due to "long tank switching," if the instruction was executed from a short tank.

A distinction was made, as before, between numbers and instruction words: When STO or SET attempted to store a new value into an instruction word, only that part of the instruction which specified a long tank

location x was to be affected, and the value in A was regarded as an integer.

Tentative plans for representing the instructions in memory are discussed briefly in [5, pp. 83–86].

THE SORTING PROGRAM

Now we are ready to discuss von Neumann's program. His manuscript, written in ink, is 23 pages long; the first page still shows traces of the penciled phrase "TOP SECRET," which was subsequently erased. (In 1945, work on computers was classified, due to its connections with military problems.) A facsimile of page 5, the first page of the program itself, appears as Figure 1.

Von Neumann begins his memo by defining the idea of sorting records into order, and of merging two strings of records that have been sorted separately into a single sorted sequence. Then he states the purpose of the program: "We wish to formulate code instructions for sorting and for meshing [i.e. merging], and to see how much control-capacity they tie up and how much time they require."

He never actually gets around to coding the entire sorting routine in this document; only the merging process is described. For the merging problem, we assume that n records $x_0, x_1, \cdots, x_{n-1}$ are given, consisting of p words each; the first word of each record is called its "key," and we have $\text{key}(x_0) \leq \text{key}(x_1) \leq \cdots \leq \text{key}(x_{n-1})$. An additional m p-word records $y_0, y_1, \cdots, y_{m-1}$ are also given, with $\text{key}(y_0) \leq \text{key}(y_1) \leq \cdots \leq \text{key}(y_{m-1})$; the problem is to put the x's and y's together into the merged sequence $z_0, z_1, \cdots, z_{n+m-1}$, in such a way that $\text{key}(z_0) \leq \text{key}(z_1) \leq \cdots \leq \text{key}(z_{n+m-1})$.

He formulated the merging method as follows (based on a procedure then used with the IBM collator): Assume that we have found the first l records $z_0, \cdots, z_{l-1}$, where $0 \leq l \leq n + m$; and assume further that these l records consist of $x_0, \cdots, x_{n'-1}$ and $y_0, \cdots, y_{m'-1}$ in some order, where $0 \leq n' \leq n$, $0 \leq m' \leq m$, and $n' + m' = l$. There are four cases:

(α) $n' < n$, $m' < m$. There are two subcases:

(α1) $\text{key}(x_{n'}) \leq \text{key}(y_{m'})$. Let $z_l = x_{n'}$, and replace (l, n', m') by $(l + 1, n' + 1, m')$.

(α2) $\text{key}(x_{n'}) > \text{key}(y_{m'})$. Let $z_l = y_{m'}$, and replace (l, n', m') by $(l + 1, n', m' + 1)$.

(β) $n' < n$, $m' = m$. Same action as (α1).

(γ) $n' = n$, $m' < m$. Same action as (α2).

(δ) $n' = n$, $m' = m$. The process has been completed.

His program is divided up according to cases in this same way (sort of a "decision table" arrangement). In order to make his coding reasonably easy to follow, it is transliterated here into a symbolic assembly language such as people might use with the machine if it existed today. We use the pseudo-operation σ RST k (RST means "reserve short tank") to mean that symbol σ is to refer to the first of k consecutive short tank locations. The first RST in a program reserves short tank number 0, and short tanks are reserved consecutively thereafter. The other notations of our assembly language are more familiar: "EQU" denotes "equivalence", "CON" denotes an integer constant, an asterisk denotes the current location, and "**" denotes an address which will be filled in dynamically as the program runs.

Von Neumann's first step in coding the program was to consider the four-way division into cases; see (A). (Note: All numbers manipulated in the program are treated as integers.) This code assumes that the short tank locations have been set up appropriately; in particular, location SWITCH contains a TRA instruction. The code in (A) (line 22) sets the address of that TRA to either ALPHA, BETA, GAMMA, or DELTA.

Next comes the code for routines (α), (β), (γ), and (δ); see (B). Here we have a rather awkward piece of coding; von Neumann thought of a tricky way to reduce cases (β) and (γ) to case (α) by giving artificial values 0 and -1 to $\text{key}(y_{m'}) - \text{key}(x_{n'})$. But he didn't realize the far simpler approach of making (β) and (γ) identical, respectively, to (α1) and (α2). Thus, he could have

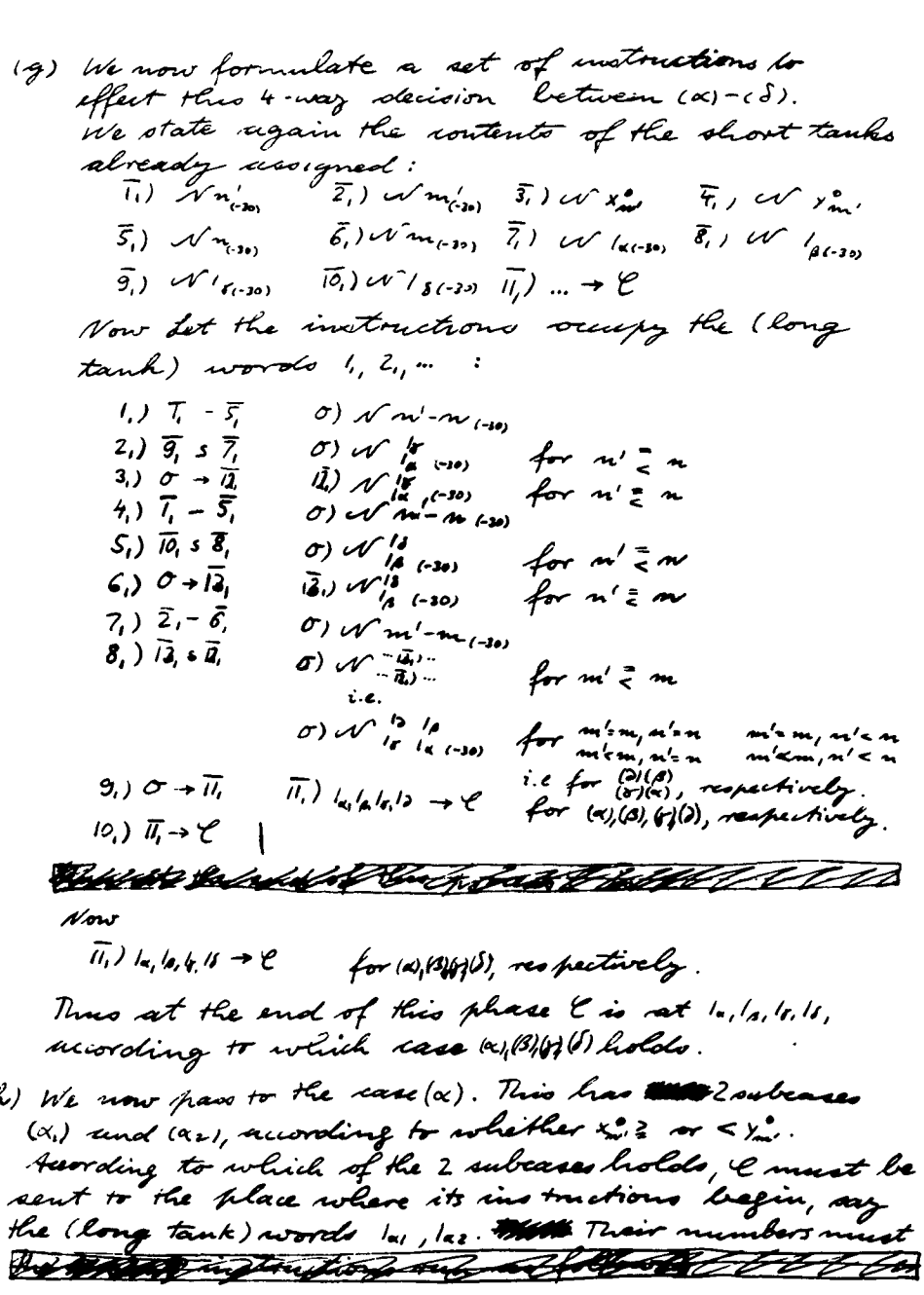

FIG. 1. The original manuscript.

simply changed line 27 to "SEL BETA, GAMMA", omitting lines 24, 25, 30, 31, 32, 33, 34, 35 entirely, and then he could have used BETA and GAMMA instead of ALPHA1 and ALPHA2 in the remainder of the program. This would have saved four of the precious short tank locations, and it would have made the calculation slightly faster. Similarly line 36 is unnecessary, since location EXIT could be stored in LDELTA.

(A)

Line no.	*Location*	*Op*	*Address(es)*	*Remarks*
1	NPRIME	RST	1	n'
2	MPRIME	RST	1	m'
3	XKEY	RST	1	key$(x_{n'})$
4	YKEY	RST	1	key$(y_{m'})$
5	N	RST	1	n
6	M	RST	1	m
7	LALPHA	RST	1	Location ALPHA
8	LBETA	RST	1	Location BETA
9	LGAMMA	RST	1	Location GAMMA
10	LDELTA	RST	1	Location DELTA
11	SWITCH	RST	1	Instruction TRA **
12	TEMP1	RST	1	Temporary storage
13	TEMP2	RST	1	Temporary storage
14	COMPARE	SUB	NPRIME,N	$A \leftarrow n' - n$.
15		SEL	LGAMMA,LALPHA	$A \leftarrow$ **if** $n' \geq n$ **then** GAMMA **else** ALPHA.
16		STO	TEMP1	TEMP1 $\leftarrow A$.
17		SUB	NPRIME,N	$A \leftarrow n' - n$.
18		SEL	LDELTA,LBETA	$A \leftarrow$ **if** $n' \geq n$ **then** DELTA **else** BETA.
19		STO	TEMP2	TEMP2 $\leftarrow A$.
20		SUB	MPRIME,M	$A \leftarrow m' - m$.
21		SEL	TEMP2,TEMP1	$A \leftarrow$ **if** $m' \geq m$ **then**[TEMP2]**else**[TEMP1].
22		STO	SWITCH	SWITCH $\leftarrow$ TRA [A].
23		JMP	SWITCH	

(B)

Line no.	*Location*	*Op*	*Address(es)*	*Remarks*
24	LALPHA1	RST	1	Location ALPHA1
25	LALPHA2	RST	1	Location ALPHA2
26	ALPHA	SUB	YKEY,XKEY	$A \leftarrow$ key$(y_{m'})$ − key$(x_{n'})$.
27		SEL	LALPHA1,LALPHA2	**if** $A \geq 0$ **then** ALPHA1 **else** ALPHA2.
28		STO	SWITCH	
29		JMP	SWITCH	
30	ZERO	RST	1	0
31	MONE	RST	1	−1
32	BETA	SUB	ZERO,ZERO	$A \leftarrow 0$.
33		TRA	ALPHA+1	Go to ALPHA+1.
34	GAMMA	SUB	MONE,ZERO	$A \leftarrow -1$.
35		TRA	ALPHA+1	Go to ALPHA+1.
36	DELTA	TRA	EXIT	Merging is complete.

Apparently the idea of making equivalent program states identical is not a natural concept, since even von Neumann missed it here.

(It is perhaps in bad taste to make such detailed criticism of the programming, since von Neumann was not intending to write an optimum program for sorting; he was merely experimenting with a tentative order code. Every great mathematician has a wastebasket full of things he doesn't want people to study carefully! On the other hand, this particular manuscript was not merely a rough sketch, it was evidently put together with some care, so it seems fair to look closely at it in an attempt to discern which aspects of programming were most difficult in their conception. The idea is not to chortle over the fact that von Neumann's program isn't perfect; it is rather to realize that the imperfections give some historical insights, because of when the program was written.)

(C)

Line no.	Location	Op	Address(es)	Remarks
37	XPTR	RST	1	Location of $x_{n'}$
38	YPTR	RST	1	Location of $y_{m'}$
39	ZPTR	RST	1	Location of $z_{n'+m'}$
40	SIZE	RST	1	p
41	MOVEIN	RST	1	Instruction PIK p+1,BUFFER,**
42	MOVEOUT	RST	1	Instruction PUT p,BUFFER,**
43	RETURN	RST	1	Instruction TRA BACK1 or BACK2
44	ONE	RST	1	1
45	BUFFER	RST	p+1	Place for record being transferred
46	ALPHA1	SET	MOVEIN,XPTR	MOVEIN ← PIK p+1,BUFFER,[XPTR]
47		SET	MOVEOUT,ZPTR	MOVEOUT ← PUT p,BUFFER,[ZPTR].
48		PIK	1,RETURN	RETURN ← TRA BACK1
49		[TRA	BACK1	(This line "picked.")
50		JMP	MOVEIN	Execute three commands in short tank.
51	BACK1	ADD	NPRIME,ONE	$A \leftarrow n' + 1$.
52		STO	NPRIME	$n' \leftarrow A$.
53		SET	XKEY,BUFFER+p	Update key($x_{n'}$).
54		ADD	XPTR,SIZE	$A \leftarrow$ [XPTR]+p.
55		STO	XPTR	Update location of $x_{n'}$.
56		ADD	ZPTR,SIZE	$A \leftarrow$ [ZPTR]+p.
57		STO	ZPTR	Update location of $z_{n'+m'}$.
58		TRA	COMPARE	Return to COMPARE.

The sorting program continues with the routine for case (α1): In (C) a block of $p + 1$ words (including the key for the next record) is transferred into short tanks, and p words are moved into the z area. This is a good way to avoid the latency problems of delay-line memories, and it accounts for the considerable increase in speed in this program compared to what was possible with the first EDVAC code.

A slight improvement could be made here if ZPTR were omitted, letting MOVEOUT keep track of the current z location; a short tank would be saved, as well as the instruction in line 47 (and a similar instruction for case (α2)). However this trick would have made the setup somewhat less symmetrical. Line 58 could have been omitted if the code for COMPARE were placed right after line 57. If line 51 were changed to "SUB NPRIME,MONE", another short tank could have been saved. Since von Neumann didn't mention these simplifications, while his work on logic design strongly suggests that he would have thought of them, it is plausible to say that he wasn't especially concerned with saving space in short tanks, although he does mention that the scarcity of short tanks places limits on the record size p. (He says that $p \leq 8$ would be required if there were only 32 short tanks, while $p \leq 40$ if there were 64; perhaps he was purposely wasting short tanks, in order to convince other people that at least 64 short tanks are desirable!)

We need not discuss the code for (α2), since it is essentially the same as that for (α1). All that is left, therefore, is to write an initialization routine that gets everything started properly. For this purpose, von Neumann juggled the short tank locations so that the six which are set up from outside this routine (namely N, M, XPTR, YPTR, ZPTR, SIZE) come first; then come two which are somewhat special (namely XKEY and YKEY, which must contain key(x_0) and key(y_0)); then come 14 which are to be set to certain constant values; and then come the remaining "scratch" locations. Figure 2 shows the resulting complete program, including the initialization of the short tanks. (At this point in his discussion, von Neumann apparently forgot about TEMP1 and TEMP2; Figure 2 assigns them to the buffer area.)

Like nearly all programs, this one has a

```
N          RST   1
M          RST   1
XPTR       RST   1
YPTR       RST   1
ZPTR       RST   1
SIZE       RST   1
XKEY       RST   1
YKEY       RST   1
NPRIME     RST   1
MPRIME     RST   1
LALPHA     RST   1
LBETA      RST   1
LGAMMA     RST   1
LDELTA     RST   1
SWITCH     RST   1
LALPHA1    RST   1
LALPHA2    RST   1
ZERO       RST   1
MONE       RST   1
ONE        RST   1
MOVEIN     RST   1
MOVEOUT    RST   1
RETURN     RST   1
BUFFER     RST   p+1
TEMP1      EQU   BUFFER+1
TEMP2      EQU   BUFFER+2
COMPARE    SUB   NPRIME,N          e+27
           SEL   LGAMMA,LALPHA     e+28
           STO   TEMP1             e+29
           SUB   NPRIME,N          e+30
           SEL   LDELTA,LBETA      e+31
           STO   TEMP2             e+32
           SUB   MPRIME,M          e+33
           SEL   TEMP2,TEMP1       e+34
           STO   SWITCH            e+35
           JMP   SWITCH            e+36
ALPHA      SUB   YKEY,XKEY         e+43
           SEL   LALPHA1,LALPHA2   e+44
           STO   SWITCH            e+45
           JMP   SWITCH            e+46
BETA       SUB   ZERO,ZERO         e+39
           TRA   ALPHA+1           e+40
GAMMA      SUB   MONE,ZERO         e+41
           TRA   ALPHA+1           e+42
DELTA      TRA   EXIT              e+81
ALPHA1     SET   MOVEIN,XPTR       e+47
           SET   MOVEOUT,ZPTR      e+48
           PIK   1,RETURN          e+49
           TRA   BACK1             e+50
           JMP   MOVEIN            e+51
BACK1      ADD   NPRIME,ONE        e+56
           STO   NPRIME            e+57
           SET   XKEY,BUFFER+p     e+58
           ADD   XPTR,SIZE         e+59
           STO   XPTR              e+60
           ADD   ZPTR,SIZE         e+61
           STO   ZPTR              e+62
           TRA   COMPARE           e+63
ALPHA2     SET   MOVEIN,YPTR       e+64
           SET   MOVEOUT,ZPTR      e+65
           PIK   1,RETURN          e+66
           TRA   BACK2             e+67
           JMP   MOVEIN            e+68
BACK2      ADD   MPRIME,ONE        e+73
           STO   MPRIME            e+74
           SET   YKEY,BUFFER+p     e+75
           ADD   YPTR,SIZE         e+76
           STO   YPTR              e+77
           ADD   ZPTR,SIZE         e+78
           STO   ZPTR              e+79
           TRA   COMPARE           e+80
BRING      EQU   NPRIME
MERGE      PIK   3,BRING           e+0
           PIK   1,XKEY,**         e+1
           PIK   1,YKEY,**         e+2
           TRA   BACK3             e+3
           SET   BRING,XPTR        e+4
           SET   BRING+1,YPTR      e+5
           JMP   BRING             e+6
BACK3      PIK   14,NPRIME         e+11
           CON   0                 e+12
           CON   0                 e+13
           CON   ALPHA             e+14
           CON   BETA              e+15
           CON   GAMMA             e+16
           CON   DELTA             e+17
           TRA   **                e+18
           CON   ALPHA1            e+19
           CON   ALPHA2            e+20
           CON   0                 e+21
           CON   -1                e+22
           PIK   p+1,BUFFER,**     e+23
           PUT   p,BUFFER,**       e+24
           CON   1                 e+25
           TRA   COMPARE           e+26
```

FIG. 2

bug: The second-last instruction "CON 1" actually belongs two lines earlier. If von Neumann had had an EDVAC on which to run this program, he would have discovered debugging!

STORAGE ALLOCATION AND TIMING

Although von Neumann didn't use a symbolic language to express his instructions, as done here, his notation wasn't completely

numeric either. He used $\bar{1}_1, \bar{2}_1, \cdots$ for short tanks NPRIME, MPRIME, etc. in the first piece of code, and later $\bar{1}_2, \bar{2}_2, \cdots$ for the short tanks in the second, etc. Long tank locations were represented by unbarred numbers with subscripts; for example, lines 32 and 33 in his notation were written as follows:

$$\begin{array}{l|l} \text{“}1_\beta)\ \bar{3}_2 - \bar{3}_2 & \sigma)\ \aleph 0 \\ 2_\beta)\ 2_\alpha \rightarrow \mathfrak{C} & \end{array}$$

and from here on like (α) with 0,0 for x_n^0, $y_{m'}^0$.” This was essentially a “regional” addressing technique, which was used by many programmers in the ensuing decade.

After having written the program, he assigned actual addresses to the subscripted ones. In order to make the code relocatable, for use as a general open subroutine, he assigned the addresses relative to an unspecified starting location e. His address assignments are shown in Figure 2 at the right of the instructions.

He made an interesting and rather subtle error of judgment here, regarding latency time. Since the instruction in location ALPHA1+4 (line 50 of the program in the preceding section) jumps into the short tanks to execute three commands and transfer to BACK1, he didn't want BACK1 to occupy location ALPHA1+5 since the long tank wouldn't be ready for that instruction until at least 33 word times after ALPHA1+4. So he intercalated 4 empty words between ALPHA1+4 and BACK1, “in order to avoid a delay of about one long tank.” But since the instructions in MOVEIN and MOVEOUT make essentially random references to long tanks, an elementary argument can be given to prove that the *average* computation time which elapses between the execution of instruction ALPHA1+4 and the execution of instruction BACK1 is $2p + 49.5$ word times, completely *independent* of the location of BACK1! Therefore BACK1 should really have been placed so that its *subsequent* instructions are optimally located, i.e. so that the TRA COMPARE takes the least amount of time. Von Neumann inserted extra blank words into the initialization routine for the same fallacious reason. On the other hand his allocation of ALPHA, BETA, and GAMMA vis-à-vis each other and the COMPARE routine was correctly handled; the instruction in SWITCH is not a random memory reference, so his intuition didn't mislead him here. (ALPHA1 and ALPHA2 were placed badly; this was apparently an oversight.)

Von Neumann discussed the relocatability of this routine by enumerating the nine instructions which are variable (those whose codes depend on p, EXIT, or the relocation factor e). He didn't say exactly how these instructions were to be changed after they have been read in from tape; he apparently did not yet realize that the limited EDVAC code he had proposed (with no shift instructions, for example) made it difficult to insert p into the “PIK” and “PUT” instructions, since the machine could only store into the address field of instruction words.

It is perhaps significant that he thought of this program as an open subroutine, not closed, since he did not regard EXIT as a parameter on a par with n, m, location(x_0), etc.

He concludes his memorandum with an analysis of the running time, leading to a total time of $2.60 + (n + m)(p/16 + 2.61)$ msec. (His actual figure was 1.61 instead of 2.61, due to a slip in arithmetic.) Some errors in the calculation of latency times, related to his misunderstanding cited above, make this analysis slightly invalid; the reader may verify that the actual running time (averaged over all possible placements of x_0, y_0, and z_0 in the long tanks) is $3056 + (n + m)(64p + 4016)$ μsec. If we incorporate all of the improvements to the coding that have been mentioned above, the average time decreases to $2576 + (n + m)(64p + 2560)$ μsec.

THE SEQUEL

After World War II came to an end, the original EDVAC group disbanded; Eckert and Mauchly remained in Philadelphia, to form their own company, while Goldstine and von Neumann went to the Institute for Advanced Study in Princeton. The veil of

secrecy surrounding electronic computers was lifted when ENIAC was dedicated, and the great potential for high speed computing was gradually realized by more and more people. The principles of EDVAC's design were very strong influences on all of the computers constructed during the next decade (see [14]).

After von Neumann's first two versions of instruction codes had been digested by a number of people, other variations began to be proposed. In November 1945, Calvin N. Mooers devised a three-address code as an alternative to von Neumann's idea; and in August 1946, he lectured at the Moore School about a further development, the use of flagged data for terminating loops [13, Vol. 4, lect. 39]. Another interesting three-address code, due to John Mauchly, was described by Eckert in the same series of lectures [13, Vol. 1, lect. 10]. Meanwhile von Neumann had developed his ideas somewhat further; he and Goldstine, in collaboration with Arthur W. Burks, prepared a monograph which was to be the first widely circulated document about high speed computers, "Preliminary discussion of the logical design of an electronic computing instrument" [2]. By this time, their proposed machine had already changed somewhat drastically: It was to have a random-access (iconoscope) memory of 4096 40-bit words. Instructions were 20 bits long, packed two to a word. The operation codes had a different flavor, too, resembling today's IBM 7094: "Clear and add x", etc. Left and right shift operations were included for the first time.

The EDVAC project itself continued at the Moore School until August 1949, when EDVAC was delivered to the BRL. In its final form, the EDVAC had a four-address instruction code (the fourth address specifying the location of the next instruction), devised by Samuel Lubkin. Its memory consisted of 128 long tanks, each containing eight 44-bit words, plus six one-word non-addressable short tanks, and an auxiliary drum. One of the only things that remained unchanged throughout most of its design was the basic clock rate of one μsec per bit; the completed machine processed one word every 48 μsec, leaving four "blank" bits between words. Further development work on input/output devices was necessary before EDVAC became operational late in 1951; then it continued steady and inexpensive operation for many years, averaging, for example, 145 hours of useful work per week in 1961 [11]. It was finally retired from service in December 1962.

For the story of von Neumann's other pioneering contributions to computing, see Goldstine's recent account [6]. Goldstine and von Neumann published three important supplements to [2] during the next years; these famous documents [7–9] formed the foundation for computer programming techniques, covering a wide range of topics from flowcharts to numerical analysis to relocatable loading routines. Reference [8, Sec. 11] deals with sorting and merging in considerable detail; von Neumann here put the finishing touches onto the work he had sketched in 1945.

ACKNOWLEDGMENTS

I wish to thank Drs. Goldstine and Mauchly for considerable assistance in checking the historical details presented in this paper, and for several delightfully informative discussions.

REFERENCES

1. Augusta, Ada, Countess of Lovelace. Annotated transl. of Menabrea, L. F., Sketch of the Analytical Engine invented by Charles Babbage. In *Charles Babbage and his Calculating Engines* (Philip Morrison and Emily Morrison, Eds.), Dover, New York, 1961, pp. 225–297; see also p. 68.
2. Burks, Arthur W., Herman H. Goldstine, and John von Neumann Preliminary discussion of the logical design of an electronic computing instrument. Inst. for Advanced Study, Princeton, N. J., June 28, 1946; 2nd ed., Sept. 2, 1947 (42 pp). (Reprinted in von Neumann's *Collected Works, Vol. 5*, A. H. Taub, Ed. (Pergamon, London, 1963), pp 34–79.
3. Desmonde, William H., and Klaus J. Berkling. The Zuse Z-3. *Datamation 12* (Sept. 1966), 30–31
4. Eckert, J. Presper, Jr., and John W. Mauchly. Application of analyzer to a set of equations for external ballistics. In Proposal for an electronic difference analyzer (J. G. Brainerd, Ed), Moore School of Elec. Eng.,

U of Pennsylvania, Philadelphia, Pa., April 1943, App. C (4 pp). (Originally classified "Confidential.")

5. ECKERT, J. PRESPER, JR., AND JOHN W MAUCHLY. Automatic high-speed computing· A progress report on the EDVAC. Moore School of Elec Eng., U. of Pennsylvania, Philadelphia, Pa., Sept. 30, 1945 (111 pages) (Originally classified "Confidential.")
6. GOLDSTINE, HERMAN H. Chapter 3 in *Computers and Their Role in the Physical Sciences* (S Fernbach and A. H. Taub, Eds), Gordon and Breach, New York, 1970.
7. GOLDSTINE, HERMAN H , AND JOHN VON NEUMANN. Planning and coding of problems for an electronic computing instrument, Vol. 1. Inst. for Advanced Study, Princeton, N. J , April 1, 1947 (69 pp.). (Reprinted in von Neumann's *Collected Works, Vol 5*, A H Taub, Ed., Pergamon, London, 1963, pp. 80–151.)
8 GOLDSTINE, HERMAN H., AND JOHN VON NEUMANN. Planning and coding of problems for an electronic computing instrument, Vol. 2 Inst. for Advanced Study, Princeton, N. J , April 15, 1948 (68 pp.). (Reprinted in von Neumann's *Collected Works, Vol. 5*, A. H. Taub, Ed., Pergamon, London, 1963, pp 152–214)
9. GOLDSTINE, HERMAN H., AND JOHN VON NEUMANN. Planning and coding of problems for an electronic computing instrument, Vol. 3 Inst. for Advanced Study, Princeton, N J., Aug. 16, 1948 (23 pp.). (Reprinted in von Neumann's *Collected Works, Vol. 5*, A H Taub, Ed., Pergamon, London, 1963, pp 215–235)
10. HARVARD UNIVERSITY, STAFF OF THE COMPUTATION LABORATORY. *Annals of the Computation Laboratory, Vol 1, A Manual of Operation for the Automatic Sequence-Controlled Calculator* (H Aiken et al , Eds.), Harvard U. Press, Cambridge, Mass , 1946

11 KEMPF, KARL. Electronic computers within the Ordnance Corps Aberdeen Proving Ground, Aberdeen, Md., Nov. 1961 (140 pp.)

12 PANTAGES, ANGELINE Computing's early years. *Datamation 13* (Oct. 1967), 60–65.

13 PATTERSON, GEORGE W. (ED). *Theory and Techniques for the Design of Electronic Digital Computers, Vol. 4* Moore School of Elec. Eng., U of Pennsylvania, Philadelphia, Pa., 1946 (4 vols.)

14 ROSEN, SAUL. Electronic computers: A historical survey *Comput Surveys 1*, 1 (March 1969), 7–36

15. STIBITZ, GEORGE R., as told to Mrs Evelyn Loveday The relay computers at Bell Labs. *Datamation 13* (April 1967), 35–44; *13* (May 1967), 45–49.

16 TURING, A. M. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc.* {2} *42* (1936), 230–265; {2} *43* (1937), 544–546.

17. VON NEUMANN, JOHN First draft of a report on the EDVAC. Moore School of Elec Eng , U. of Pennsylvania, Philadelphia, Pa , June 30, 1945 (101 pp). (This draft was written in March–April 1945.)
18. VON NEUMANN, JOHN. Letter to Herman H. Goldstine dated May 8, 1945.