Arch. Math. Logic (2010) 49:123–146
DOI 10.1007/s00153-009-0169-8

Setting-up early computer programs: D. H. Lehmer's ENIAC computation

Maarten Bullynck · Liesbeth De Mol

Received: 18 November 2009 / Accepted: 23 November 2009 / Published online: 10 December 2009

Abstract A complete reconstruction of Lehmer's ENIAC set-up for computing the exponents of p modulo two is given. This program served as an early test program for the ENIAC (1946). The reconstruction illustrates the difficulties of early programmers to find a way between a man operated and a machine operated computation. These difficulties concern both the content level (the algorithm) and the formal level (the logic of sequencing operations).

Keywords ENIAC · History of computer programs · Number theory · Derrick H. Lehmer

Mathematics Subject Classification (2000) 11-04 · 11N36 · 11A07 · 01A60 · 11Y16

1 Introduction

In 1943, the Army's Ordnance Proving Ground at Aberdeen Maryland contracted Mauchly and Prespert Eckert to build the ENIAC, the first U.S. electronic digital and (basically) general-purpose computer. Its original purpose was the computation of ballistic trajectories. However, due to its architecture, it could be used to solve many more problems. Already in 1945 the Ordnance also assembled a "Computations Committee" that had to prepare for utilizing the machine after its completion. The

M. Bullynck
Université Paris 8, Vincennes à Saint-Denis, Paris, France
e-mail: maarten.bullynck@kuttaka.org

L. De Mol (✉)
CLPS, Universiteit Gent, Blandijnberg 2, 9000 Gent, Belgium
e-mail: elizabeth.demol@ugent.be

Committee consisted of the number theorist Lehmer, the logician Curry, the astronomer Cunningham and the statistician Alt [1]. Each developed their own test program to be run on the ENIAC after it was first presented to the public at Penn University February 15, 1946. Soon afterwards, through John von Neumann, the Atomic Energy Commission got interested in using the ENIAC for computations connected to the development of the H-bomb.

These test programs, the ballistic trajectory computations and especially the problems that the Atomic Energy Commission wanted to put on ENIAC soon made clear the difficulties and limitations of "programming" the ENIAC in its original hardware configuration [2,3]. For each new problem, the ENIAC had to be programmed directly and locally, setting the switches on each individual unit, laying the cables to interconnect these units and control the timing and sequencing of the units' operations. The units, 30 in number, operated in parallel, which made the timing and sequencing a tricky job. Moreover, the amount of dynamic ("RAM") and static ("ROM") memory was limited. The memory problem could, however, be circumvented by using punch cards to store (intermediate) results and inputting them into the computation at a given point. Again, the synchronization of this was a difficult task. Programming the ENIAC in its original configuration thus came down to "the design and development of a special-purpose computer out of ENIAC component parts" [2, p. 32]. Or as Jean Bartik, one of the ENIAC's female programmers, put it, the ENIAC "was a son-of-a-bitch to program" [4, p. 94].

As a consequence , ways of simplifying the ENIAC's set-up were contemplated. Initially, often recurring instructions were codified as two-digit numbers that could be looked up in the function table unit that in its turn triggered the right program pulses [5, Sect. 7.4]. A logical coding system consisting of a 60-word vocabulary of orders was put on the ENIAC in the latter half of 1947. Mid 1948 a 100-word converter code was developed and installed on the ENIAC (i.e., the ENIAC was rewired, switches set semi-permanently) [6,7]. This code was often expanded and revised during the following years and turned the ENIAC into a "more effective serial stored-program computer" [2, pp. 32–33].[1] The set-up of a new problem on ENIAC was thus considerably simplified, though at a cost. The running time increased with a factor of six on the average.

The early (declassified) test programs are unique instances of "programming" a machine that had not the kind of logical design we know nowadays as the von Neumann architecture. Furthermore any kind of programming language was totally absent. These programs demonstrate the difficulties of adapting computations made to human measure to a machine. Also various solutions towards logically organizing the processes within that computation had to be invented. The current reconstruction of Lehmer's ENIAC program extends and completes the partial reconstruction presented in [9] and discusses in detail Lehmer's early attempt at "programming". Furthermore, the reconstruction of the sieve is now improved by removing certain redundancies.

[1] There is much controversy regarding the invention of the stored-program computer. We will not enter into this discussion but refer to the lucid discussion in [8].

2 ENIAC: architecture and program planning

2.1 Overview of ENIAC

The Electrical Numerical Integrator And Computer (ENIAC) was first described in a proposal John Mauchly submitted to Penn University, and was ultimately built with U.S. army money by a team of engineers under the direction of Presper Eckert Jr. It used about 18,000 vacuum tubes and 1,500 relays. From 1945 to 1947 this was a highly parallel computer, though its parallelism was hardly ever used [10, p. 376]. Lehmer's program was one of the notable exceptions.

The ENIAC had a modular architecture. It comprised 20 accumulators, a multiplier, a divider and square rooter, a constant transmitter, three function tables, a master programmer, a cycling unit, an initiating unit and also a card reader and a printer. The constant transmitter and the function tables were the ENIAC's main (static) memory storage units ("ROM"). The accumulators were the main (dynamic) memory storage units that could be changed during a computation ("RAM"), each storing a signed 10-digit number. However, each accumulator had 12 program controls which "gives us 240 components we can use in the computational dataflow" [11, p. 20]. Furthermore each accumulator could be used to store more than one number. E.g. it could be used to process two 5-digit numbers with the same sign. These could be processed separately by using special shifter and deleter adaptors (see Sect. 3.3.4).[2]

Another way to extend the ENIAC's small amount of both memory types is the following. The constant transmitter (CT) consisted of two panels: (CT2) where numbers could be stored by manually setting the switches before computation, and (CT1) where the counters were set automatically according to the input that was read on a punched card fed into the card reader. This could be done during a computation, though that required some ingenuity to synchronize the (slow) reading of the punched card with the steps in the computation. Therefore, the designers of the ENIAC added an interlock to this unit, to simplify exactly this kind of synchronization [5, Sect. 8.2.4].[3] With the card reader and the (CT1) unit new numbers could be brought into the computation during runtime.

Of crucial importance in the ENIAC was the central programming pulse (CPP), emitted by the cycling unit once every 1/5000th of a second, marking the beginning and end of a computation cycle. These pulses synchronized the operations of the units. When a unit completed an operation it emitted one of these as a program output pulse, stimulating the next operation. All the units required an integral number of cycles. E.g. an accumulator required 1/5000th of a second for an addition, therefore we speak also of an addition time instead of a cycle.

The ENIAC had two kinds of electronic circuits: the *numerical* circuits for storing and processing electric signals representing numbers and *programming* circuits for

[2] The possibility of splitting one number into two smaller numbers is also possible in the function tables and the constant transmitter.

[3] For similar reasons, also the printer unit and the division units had such an interlock.

controlling the communication between the different parts of the machine. Most of the units had both kind of circuits, except for the master programmer and the initiating unit which consisted only of programming circuits. Numerical pulses have a range of 0–9 P (a 10 P also exists) with 1 P being the unit voltage. Program pulses always have a voltage of one CPP (= $1'$ P). Numerical and programming pulses are strictly separated. There was, however, a technique to convert a digit pulse into a program pulse through the use of a special digit-to-program pulse adaptor and one or two dummy programs (see Sects. 2.4.1, 2.4.2).

Our further discussion of the ENIAC will focus on the salient "programming" facilities, that is, mainly the accumulator and the master programmer. In this discussion we have made use of various references. The most detailed technical description of the ENIAC is [5], less detailed but instructive and rather readable accounts are [10,12, 13]. We would also like to refer to recent research on the ENIAC: [14] analyzed the ENIAC's architecture and used VLSI to put ENIAC on a chip; [15,11] present (incomplete) Java simulations of ENIAC.[4]

2.2 Accumulators

The accumulators were the main arithmetic units of the ENIAC and could be used to add or subtract a number. Figure 1 gives a graphical representation of an accumulator. Each accumulator held a ten place decimal number and a sign (P for plus and M for minus), stored in ten decade ring counters and a PM counter. It had five input channels (α to ϵ) to receive a number. It had two output channels (A and S) to transmit a number n (through A) or its complement $10^{10} - n$ (through S). In one addition time, the accumulator could either receive a number n, adding (if $n \geq 0$) or subtracting (if $n < 0$) it to/from its content, or transmit the number it stored through one or both of its output channels. The program part of the accumulator consisted of 12 program controls: four receivers and eight transceivers. A transceiver had a program pulse input and output terminal, a clear-correct switch (to clear or not clear its content after a cycle; it could also be used to round off numerical results), an operation switch (to be set to $\alpha - \epsilon$, A, S, AS or 0, determining whether the accumulator should receive or transmit a number, or do nothing) a repeat switch (with which it could either receive or transmit up to nine times). When a transceiver received a program pulse through a program cable at addition time r, the operations set on the program switch associated with that transceiver were executed. When these had been finished after n ($1 \leq n \leq 9$) addition times, a program pulse was transmitted through the output of the transceiver at addition time $r + n$. A receiver differs from a transceiver in that it has no output terminal and no repeater switch. An accumulator furthermore contains a significant figures switch. This is used to round off numerical results. If the significant figures switch is set to a number s, then, when clearing

[4] More particularly, the master programmer, the function tables, the card reader, the printer and the special multiplication and division combinations of accumulators are missing in the Java simulations. The source code of Hansen's Java simulation can be found at: http://home.arcor.de/-ph/eniac/download.html.

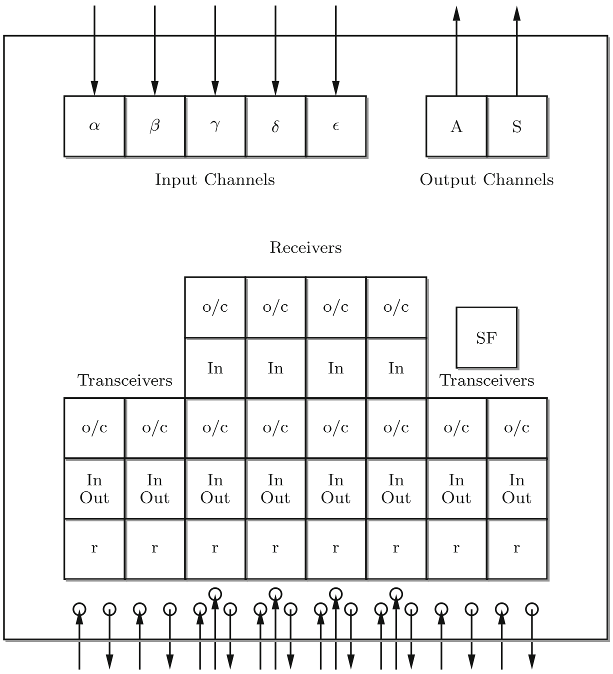


Fig. 1 A schematic representation of an accumulator. Here, "*SF*" is the significant figures switch, "*In*" rsp. "*In/Out*" the operation switches of the four receivers rsp. the eight transceivers, "*r*" the repeater switch and "*o/c*" the clear-correct switch for the 12 program controls. On the *bottom*, an *incoming arrow* indicates that a program pulse can be received, an *outgoing arrow* that a program pulse can be sent. At the *top*, α to ϵ are the five input terminals and A and S the two output terminals

takes place using the clear-correct switch, decade $10 - s$ clears to five and all other decades to 0.

2.3 Master programmer: loops and sequencing

The master programmer provided a certain amount of centralized programming memory. It consisted of ten independently functioning units, each having a 6-stage counter (called the stepper, see Fig. 2), three input terminals (the stepper input, direct input and clear input), and six output terminals for each stage of the stepper. Each such stage s was associated with a fixed number d_s by manually setting decade switches, and with one to five decade counters. If a pulse arrived at the stepper input (SI) of a stepper, one was added to the counter of stage s. If this number equaled the preset number d_s, it cleared the counter of stage s and cycled to the next stage $s + 1$. In both cases, a program pulse was emitted through the output terminal of stage s. A pulse at the direct input (DI) *immediately* cycles the stepper to the next stage and a pulse at the clear input (CI) resets the stepper to its initial configuration. In neither case a program pulse was emitted. In this way the master programmer could be used, among other things, to sequence operations and to iterate a given subroutine.

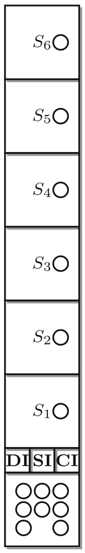


Fig. 2 A schematic (reduced) representation of a stepper counter of the master programmer. S_1 to S_6 represent the six stages of the stepper, *DI* is the direct input, *SI* the stepper input and *CI* the clear input. Note that there are three input terminals for *CI* and *DI* and only two for *SI*

2.4 Conditional branching

The ENIAC was capable of discriminating between program sequences by examining the magnitude of some numerical result. This was done by using the technique of converting digit pulses into program pulses, through special adaptors and dummy controls. "Magnitude discrimination" or "branching" could be done by exploiting the fact that nine digit pulses were transmitted for sign indication M and none for sign indication P. The fact that digit pulses were transmitted for every digit except for 0 could be exploited in a similar manner. The digit pulse corresponding to the sign (or a digit) was converted into a program pulse by connecting the PM lead (or another digit lead) of the *A* and/or *S* output terminal of an accumulator to a program cable. A special adaptor which was placed on this lead took care of the conversion. The program cable was then connected with the program pulse input terminal of (an) otherwise unused "dummy (program) control(s)" (d.p.c.) [5, Sect. 4.5].[5] There were two main branching methods: the first is explained in detail by Adèle Goldstine [5], the second one is only referred to without further explanation. In what follows we will reconstruct both methods.

2.4.1 First branching method

Figure 3 illustrates the wiring of a branching method that uses both output terminals. Assume that the discrimination is of the following nature: if a given number *n* becomes smaller than some other number *m*, then program *a* should be executed, else

[5] A dummy control had the following main functions [5, Sect. 4.5]: "The dummy program has at least three important functions: (1) conversion of digit pulses into program pulses, (2) delay of a program pulse, and (3) isolation of programs from one another."

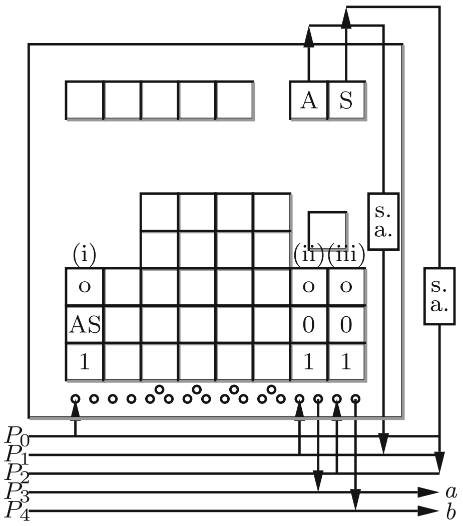


Fig. 3 First mode of branching on the ENIAC

program b should be executed. If the accumulator is activated to execute this kind of "magnitude discrimination" at time r, it should store $n - m$ (or $m - n$) at time r. For this method a first transceiver (i) should be set such that the accumulator will send its content once through both its A and S output channels, when (i) is stimulated by a program pulse. Now, in Fig. 3, when the discrimination program is activated at time r through program cable P0, $n - m$ is sent through the A and S output channels. If $n < m$, a negative number is transmitted, nine pulses are transmitted through the PM lead of the A output terminal and no pulse is transmitted at the PM lead of the S output terminal. The nine pulses transmitted through the PM lead of the A output terminal are converted into a program pulse using a special adaptor (indicated as "s.a." in Fig. 3), thus stimulating program cable P1 which in its turn activates the first dummy control (transceiver (ii)) at time $r + 1$. At time $r + 2$ a program pulse is transmitted from (ii) leading to the execution of some program a. Similarly, program b will be activated when $n \geq m$.[6]

2.4.2 Second branching method

Figure 4 shows a branching method that uses only one output terminal, together with the master programmer. This method is preferred if one needs to be economical with the accumulators, avoiding to use a complete accumulator only for the execution of branching. One stepper s_i of the master programmer is used: both the first and second stage of this stepper should be set to cycle to the next stage when they reach some value v.[7] As is the case for the first method, if one wants to discriminate between

[6] Note that the case $n = m$ is processed in the same way as $n > m$.

[7] If the stages of stepper s_i are used *only* for branching, the value of v is irrelevant so v may be set to 1. However, it may be useful to re-use these stages of the stepper at some earlier or later stage of a given program in which specific values v need to be set.

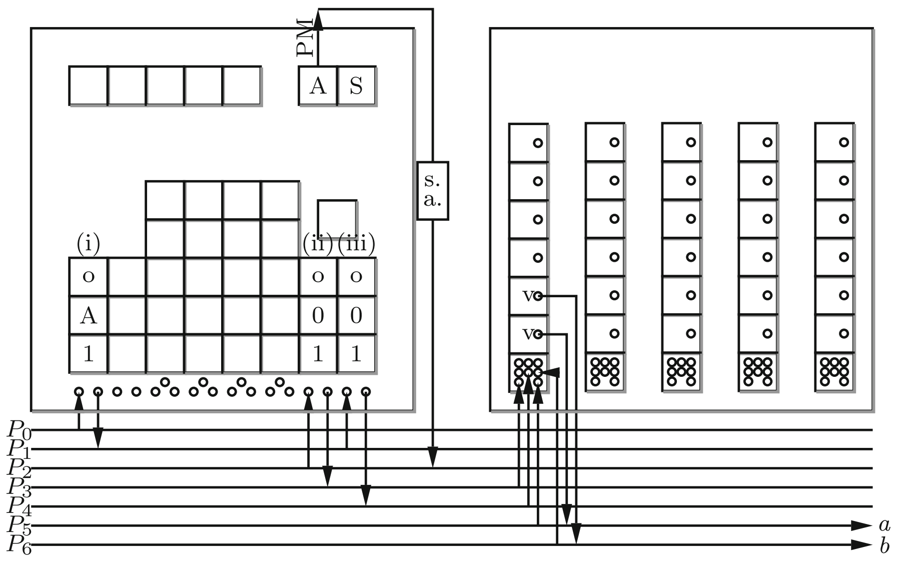


Fig. 4 Second mode of branching on the ENIAC

two programs depending on whether a number n becomes smaller than a certain value m, the accumulator doing the discrimination should store $n - m$ at the time of the discrimination. Assume that the discrimination program is activated at time r through program cable P0 in Fig. 4. This activates the transmission of $n - m$ through the A output terminal. Now let $n \geq m$. Then, at time $r + 1$ the dummy control (transceiver (ii)) will not be activated and transceiver (i) will send a program pulse to cable P1 activating another transceiver (iii) of the accumulator. This transceiver is set to do nothing except for sending a program pulse when it is "done" at time $r + 2$ to cable P4. This cable is connected to the stepper input of stepper s_i of the master programmer. The program pulse arriving at the stepper input will lead to the addition of one to the counter of stage 1. At time $r + 3$ a program pulse is then sent from stage 1 of stepper s_i to program cable P5, activating program a and also resetting the stepper.

If, on the other hand, $n < m$, then, at time $r + 1$ a dummy control (transceiver (ii)) and transceiver (iii) are activated, both set-up to do nothing except transmitting a program pulse at time $r + 2$ to program cable P3 rsp. P4. P3 is connected to the direct input of stepper s_i, P4, as we already know, to the stepper input of stepper s_i. This means that both the direct input as well as the stepper input are stimulated simultaneously. This is unproblematic because the result of a program pulse at the direct input is that the stepper *immediately* cycles to the next stage of the stepper [5, Sect. 10.5]. The program pulse at the stepper input then results in the addition of 1 to the counter of stage 2. At time $r + 3$ a program pulse is sent from stage 2 of stepper s_i to program cable P6, resulting in the activation of program b as well as the resetting of stepper s_i.

It should be pointed out that in the reconstruction of Sect. 3.3 we will only use the second branching method. There we will use the simplified and reduced scheme of Fig. 5 to keep the structure of the wiring schemes transparent.

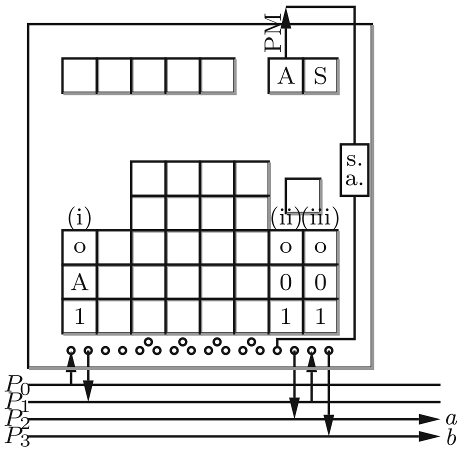


Fig. 5 Simplified representation of the 2nd mode of branching on the ENIAC

2.5 Programming 'primitives'

As has been acknowledged by Prespert Eckert, the head engineer of the ENIAC, the main inspiration for the ENIAC was the idea of "linked adding machines" [16, p. 22]. This origin is still visible in the architecture of the ENIAC, especially in the crucial role of the accumulators. However, the designers added a powerful set of "programming" machinery, like conditional branching and the master programmer. Patching together a sequence of different processes into one program that could be run without interruption was thus possible (within the limits of the hardware), though all but a straightforward task.

As a consequence, it need not wonder that among the first users of the ENIAC, many conceived aids and simplifications to program the machine. Since looping and sequencing could be controlled by the master programmer, a scheme of the wiring on the master programmer was used to represent the sequencing of and interconnections between the main steps of the computation. This kind of "reduced" wiring scheme was used by Hartree in an article that described his ENIAC computation [17].[8]

Goldstine and von Neumann developed a further abstraction from such a scheme and from the actual hardware in the context of the EDVAC project [18]. Their flow diagrams were used "to plan first the course of the process and the relationship of its successive stages to their changing codes, and to extract from this the original coded sequence as a secondary operation." [18, p. 4]. Flow diagrams are flexible tools that have remained in use until today. Finally, the logician Curry developed an abstract calculus of program composition using his theory of combinators in the process of studying how to put an inverse interpolation problem on the ENIAC [19]. Curry's work will, however, be the topic of another paper, in the meantime we refer to Knuth and Pardo's lapidary discussion [20, pp. 434–435].

[8] We will make use of this representation for Lehmer's program below (see Fig. 7). Incidentally, Hartree thanks McNulty (one of the all-female team of ENIAC programmers) and Lehmer for helping him set up and program the ENIAC.

The most difficult kind of program on the ENIAC, even with the tools Goldstine, von Neumann and Curry had devised, remained the programming of a complex parallel computation. In 1996, the historians Marcus and Akera proposed a *modus procedendi* for synchronizing such a parallel computation. For the synchronization of two branches, they proposed to program the master programmer as follows [16, p. 23]:

(1) Set the first stage of a stepper on 11.
(2) Route a program pulse into the stepper, and let the stage of the stepper provide a program pulse each addition time.
(3) Route the program output pulse of the first branch into the direct input of the 1st decade of the first stage, so that the counter goes from 0 to 1 or from 10 to 11. Route the program output pulse of the second branch into the direct input of the 2nd decade so that the counter goes from 0 to 10 or from 10 to 11.
(4) Connect the output of the second stage to whatever happens after both parallel paths have terminated.

Since a stepper stage can be associated with maximally five decades, this procedure would allow for a maximum of five parallel branches to be synchronized, i.e., each branch contributing a "1" to a decade and the next step in the program only beginning when the stage content has reached "11111". According to Marcus and Akera, this set-up was never used.

There are, however, other ways of synchronizing parallel branches that have been wired on the ENIAC. The second branching method (2.4.2) is actually an example of synchronizing two branches. Moreover, Mauchly and Lehmer synchronized a 14-branch parallel computation within Lehmer's calculation of exponents of 2 modulo p (the sieve).

3 Lehmer's program

In 1946 the number-theorist Derrick Lehmer and his wife Emma spent a Fourth-of-July weekend testing the ENIAC. The Lehmer family–Derrick, his wife Emma and two teenage kids–arrived at the Moore school on Friday 5 p.m. where they met John Mauchly. Mauchly helped them set up the ENIAC for the implementation of Lehmer's program and stayed on as an operator through the week-end [21, p. 451]. According to Akera [22, p. 40]

> [Lehmer's program] was a difficult enough problem that it attracted the attention of some mathematicians who could say, yes, an electronic computer could actually do an interesting problem in number theory–something as sophisticated in number theory–and produce useful results. There were many people who speculated about this–von Neumann among them–but to actually do it, to demonstrate it, was, I think, important to the post-war reputation of electronic computers among mathematicians.

The objective of Lehmer's program was to compute a list of exponents e of 2 mod p, p prime.[9]

[9] The exponent e of 2 modulo a prime p is defined as follows: the smallest number e such that $2^e \equiv 1 \mod p$.

3.1 Description of the problem

It was first noted by Lambert in 1770 that Fermat's little theorem ($a^{p-1} \equiv 1 \bmod p$) could be used as a primality test. If for a given number b, $2^b \equiv 2 \bmod b$ than b is with high probability a prime number. Unfortunately, an infinite set of exceptions to this primality test exists.

Early on in his career, Lehmer had written on the converse of Fermat's theorem [23], and often returned to the topic. In 1936 Lehmer established the general form of exceptions to the converse [24]. He proved that if for a composite number $n = pq$ (p and q prime), $2^{pq} - 2$ is divisible by pq, this can only occur if and only if $p - 1$ is divisible by the exponent of 2 modulo q and $q - 1$ is divisible by the exponent of 2 modulo p [24, p. 353]. Similar theorems can be proven for composites b of the forms pqr, $pqrs$ etc. that satisfy $2^b \equiv 2 \bmod b$. It is thus clear that a list of exponents of 2 mod p (p prime) is useful for the construction of prime tables.

Lehmer had been using Kraîtchik's tables for finding exponents of 2. These tables, however, only extended to 300,000, and contained rather a lot of errors. As a result of his ENIAC computation Lehmer published a list of errors to Kraîtchik's tables [25] and a list of factors of $2^n \pm 1$ [26], both in 1947. More details on his actual computation were only published 2 years later in [27] due to the "classified" label attached to many things involving the ENIAC.

In the following two sections, we will provide the details of the set-up of Lehmer's program on the ENIAC. The flow diagram and the algorithms as presented in Sect. 3.2 follow the discussion by Lehmer himself [27]. The actual reconstruction of Sect. 3.3 of the set-up is ours.

3.2 The problem from the machine's eye view

Translating a problem from man's view to a machine was not obvious when the first electronic computers emerged. Originally, this translation involved two main steps: programming and coding. According to Hartree [28, pp. 111–112]:

> "Programming" is the process of drawing up the schedule of the sequence of individual operations required to carry out the calculation, and "coding" is the process of translating these operations into instructions in the particular form in which they are read by the machine.

Coding was clearly the more difficult part, especially for a machine such as the original ENIAC where coding equalled wiring the machine. This was exactly the reason for rewiring the ENIAC (Sect. 1, p. 124) and developing so-called "stored-program logical machines", i.e., the von Neumann architecture. The difficulties of coding as well as the immense speed-up of electronic computing had an impact on the more 'high-level' programming part. The approaches and algorithms that had been in use for human computors and for desk calculators had to be rethought in function of the restrictions that coding the machine imposed and of the exponential speed-up of the ENIAC.[10] In [27] Lehmer explicitly addressed this translation problem in the

[10] In a test computation [30] the ENIAC computed a trajectory in 15 min, the second best machine, Bell's relay calculators, needed 70 h!

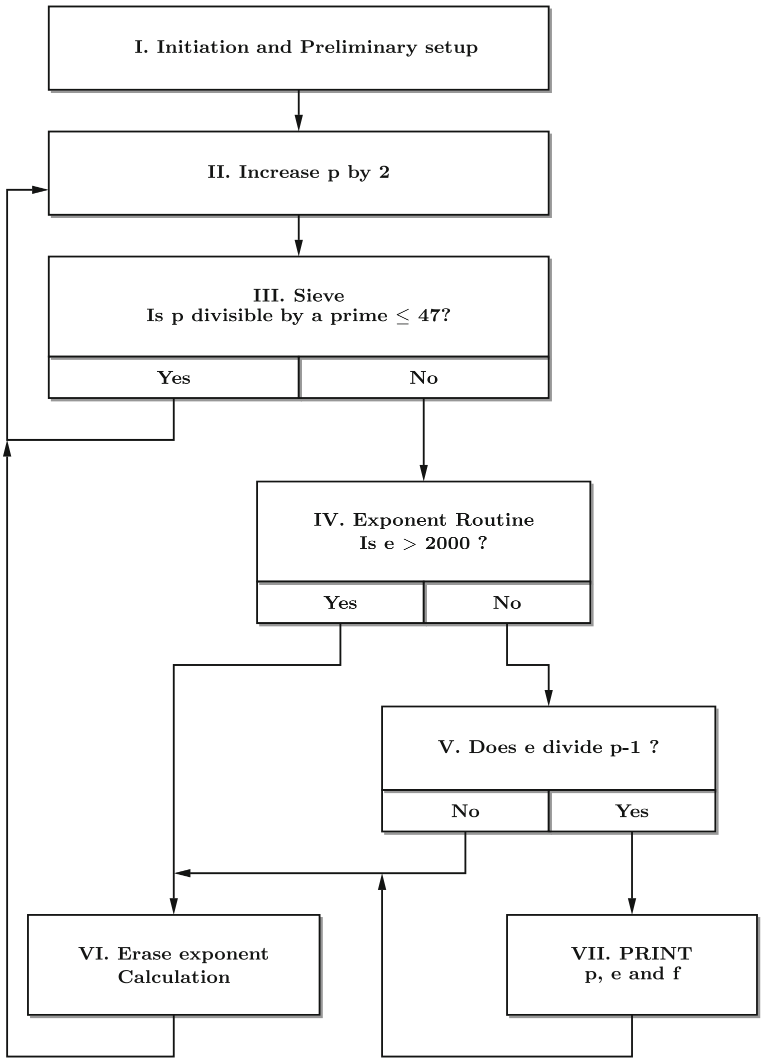


Fig. 6 Lehmer's flow chart for the ENIAC computation

discussion that accompanied the flow chart for the ENIAC's computation of the exponents e of 2 modulo p (see Fig. 6).

A human computer would calculate the exponents e more or less in the following way. First, he would take a list of primes and select the next p. Then he would, using the many abbreviations possible in modular arithmetic, calculate powers of 2 and reduce them modulo p, though not all powers, but only those that are divisors of $p-1$. This is because a number-theorist knows that if there is an exponent e ($< p-1$) of 2 so that $2^e \equiv 1 \bmod p$ (p prime), than e is either a divisor of or equal to $p-1$. He might also make use of already existing tables of exponents.

As Lehmer later described it, “[i]n contrast, the ENIAC was instructed to take an “idiot” approach” [29, p. 4]. A first problem is to let the machine recognize the “next prime”:

> The “next value of p” [i.e. the next prime] presents an interesting problem to the ENIAC. [Circumstances] prevented the introduction [of] punched cards. [...] This means that the ENIAC should somehow compute its own values of p. To this effect a “sieve” was set up which screened out all numbers having a prime factor ≤ 47 [27, p. 302].

A more sophisticated method would have required “much outside information [introduced] via punched cards [...] to be prepared by hand in advance” [27, p. 302]. Not only would this have taken quite some time, but it would have been mandatory to synchronize the card reader with the rest of the computation using the interlock, making the set-up considerably more difficult. Therefore, Lehmer entered a sieve into the program to select the next p (box (III) in the flow chart, Fig. 6). This sieve is a rather straightforward implementation of Eratosthenes’s sieve, where multiples of already known primes $p_1, p_2, \ldots, p_n$ are calculated and then rejected, leaving only those integers relatively prime to $p_1, p_2, \ldots, p_n$. In the ENIAC computation, only odd numbers were tested so 2 had not to be included as a test prime in the sieve. The ENIAC checked in parallel whether a next number p (in a progression of numbers to be checked $p = 2i + 1$) is a multiple of one or more of the first 14 odd prime numbers (3 to 47) or not. About 86% of the composite integers were thus eliminated. To reduce the number of “fake” primes remaining, Lehmer added an extra test later on in the program (box (V) in the flow chart, Fig. 6).

Next, the powers of 2 were calculated and reduced modulo p (p being a sieved number) to compute the exponents e in the following way [29, pp. 4–5] (box (IV) in the flow chart, Fig. 6):

> In contrast, the ENIAC was instructed to take an “idiot” approach, based directly on the definition of e, namely, to compute
>
> $$2^n \equiv \Gamma_n (\operatorname{mod} p), \quad n = 1, 2, \ldots$$
>
> until the value 1 appears or until $n = 2001$, whichever happens first. Of course, the procedure was done recursively by the algorithm:
>
> $$\Gamma_1 = 2, \Gamma_{n+1} = \begin{cases} \Gamma_n + \Gamma_n & \text{if } \Gamma_n + \Gamma_n < p \\ \Gamma_n + \Gamma_n - p & \text{otherwise} \end{cases}$$
>
> Only in the second case can Γ_{n+1} be equal to 1. Hence this delicate exponential question in finding $e(p)$ can be handled with only one addition, subtraction, and discrimination at a time cost, practically independent of p, of about 2 s per prime. This is less time than it takes to copy down the value of p and in those days this was sensational.

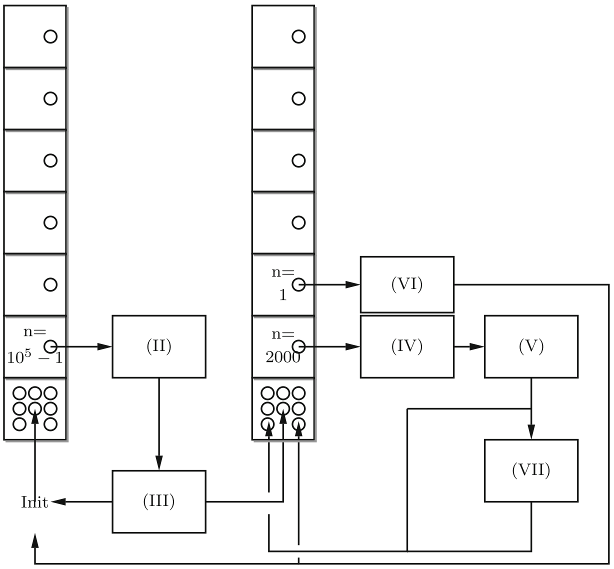


Fig. 7 The set-up of the master programmer for Lehmer's ENIAC computation

The limit value for e was changed two times during Lehmer's three-day computation. For the first 300.000 integers e was $\leq 2{,}000$, for $300{,}000 < p < 1{,}000{,}000$ this limit was reduced to 1,000 and for $10^6 < p < 4.5 \cdot 10^6$ it was set at 300 [27, p. 302].

Finally, having obtained the exponent e of 2 modulo p, an extra primality test on p was introduced (box (V) in the flow chart, Fig. 6). As known, $p - 1$ must be divisible by e if p is prime. A division by repeated subtraction checked whether $p - 1$ was divisible by e, if so, the quotient f was stored (and later punched), if not, the value p was rejected. After this final elimination of composite p's, only 25 composite numbers remained in the range $1{,}000{,}000 < p < 3{,}000{,}000$. Emma Lehmer finally eliminated these few remaining composites manually "by comparison with [a] list of primes" [27, p. 302].

3.3 Reconstruction of the program

In Fig. 7 Lehmer's flow diagram is translated in a wiring scheme of the master programmer, following Hartree's representation of ENIAC programs (see Sect. 2.5). In what follows we will first give a brief informal account of the three main steps of the program and then provide the details of the reconstruction.

The following conventions will be used in the reconstruction. First, it is assumed that for every accumulator the transceivers are numbered as $t_1, t_2, \ldots, t_8$ from left to right. Secondly, even though we will make frequent use of the constant transmitter, it will not be included in the schemes because it would take up too much space and is the less important part of the wirings. Instead, we will indicate the number transmitted by the constant transmitter directly on the relevant numerical cable. Furthermore, if the operation switch, the repeater switch and the clear-correct switch of a given transceiver of an accumulator are marked with “X”, this means that the transceiver has already been used in some other part of the program.

3.3.1 Informal account of the three main steps

The three most important and interesting steps of the computation are the sieve (III), the exponent routine (IV) and the division routine (IV).

The sieve is used to determine whether or not a given number $p = 2i + 1$ is prime relative to the first 14 odd prime numbers. Lehmer used a sieve instead of a look-up table because this was the more efficient method (see Sect. 3.2, p. 135). Now, the sieve uses hardware parallelism: in our reconstruction, 14 out of the 20 accumulators $(A_{p_1}, \ldots, A_{p_{14}})$ are used to check *in parallel* whether or not the number p being processed is divisible by one of the 14 prime numbers. The main idea behind the sieve reconstruction can best be explained as follows. Remember that the sieve method is used here to search for numbers that are not divisible by a given set of prime numbers p_i, in this case, the first 14 odd primes. The sieve can be represented by a matrix of k rows and infinitely many columns. The infinite rows are periodic, the i-th row having a period of p_i. The value in the r-th column and the i-th row equals r modulo p_i. The problem is then to find those columns that contain no 0s. For instance, suppose we have a sieve which selects numbers that are not divisible by 2, 3 and 5. Then we would get the following sieve matrix:

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1\ldots \\ 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1 & 2 & 0 & 1\ldots \\ 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3\ldots \end{pmatrix}$$

In this example, the first number (except for 1) not divisible by 2, 3 and 5 is 7, since the 7th column is the first one that contains no 0s.

Basically, what is done in the ENIAC sieve is that each of the 14 A_{p_j} corresponds to a row j of the sieve. At the start-up each accumulator stores the complement of $p_j - 1$.[11] Now, since only odd numbers p are sieved, with every loop of the sieve program each of the accumulators A_{p_j} is increased with 2 in parallel. Thus, the first number p being checked is 3. Clearly, if any of the numbers stored in one of the accumulators becomes 0 and thus positive, a divisor is detected (there is at least one 0 in the p-th column), if not, p is a prime relative to each of the 14 primes. Dummy controls assure that, whatever the case, the correct subroutine is followed. Of course, if an accumulator becomes 0, its content should be reset. I.e., the complement of $2p_j$ should be loaded into that accumulator. This is done by making use of the function tables and special adaptors. Note that the mathematics of the sieve (addition and detection happen independently in each of the accumulators) guarantees that the 14 branches are synchronous.

The program for the exponent routine was already described in detail in Sect. 3.2. It produces powers of 2 modulo the p found by the sieve. Besides the fact that Lehmer

[11] The reason for this is that the first p checked is equal to 3.

chose for the "idiot" approach, he also had to economize on the number of accumulators. For example, the doubling procedure uses only two accumulators instead of the multiplier unit (that takes up four accumulators). Also for the division routine, economization on ENIAC's accumulators was necessary. Thus, Lehmer did not use the divider unit but wired his own division procedure, basically an implementation of Euclid's algorithm for integers. It uses only 3 accumulators, of which two are also used in other subroutines. The exponent e found is successively subtracted from the prime number $p - 1$ until either $p = 0$ or $p < 0$.

Note that in this reconstruction full use is made of nearly all of ENIAC's different components and salient features, including some of its special adaptors and all of the accumulators. In this sense, the reconstructions highlight some of ENIAC's typical properties, possibilities and problems.

3.3.2 I. Initiation and preliminary set-up

As will become clear throughout the remainder of this reconstruction, certain components have to be set in advance. The 0th value ($f(0)$) of each of the function tables has to be set to a specific value. Also each of the 14 accumulators A_{p_j} used in the sieve need to be set to the complement of $p_j - 1$ (see Sect. 3.3.4). The numbers $+2$, -2 and -1 are set manually on the constant transmitter (CT). An accumulator A_P used to store the number $P = 2r+1$ being processed, should be set to store the number 1. The computation is started by an initiating pulse sent to the 14 accumulators and the CT.

3.3.3 II. Increase p by 2

In this subroutine the 14 A_{p_j}'s (see Sect. 3.3.4), A_P (storing the prime number p being processed), as well as a transceiver A_{e_1} used in subroutine IV should be set to receive once through one of their input channels. The constant transmitter is set to send the value $+2$ once. This leads to the desired increase of 2 in the several components affected by this increase. Note that after the first execution of II (the addition of 2), $P = 3$.

3.3.4 III. Sieve

In our reconstruction of the prime sieve, we have used 14 accumulators A_{p_j} for each prime $p_j \leq 47$, $(1 \leq j \leq 14)$ except for 2.[12] Upon initialization (I), each A_{p_j} is set to the complement of $p_j - 1$. e.g. $A_{p_{14}}$ will contain M 9,999,999,954.

In the first steps of the sieve implementation, it is checked for each A_{p_j} *in parallel* whether the number $P = 2r + 1$ (the first P being 3) is or is not divisible by one of the p_j. This is done with a variant of the second branching method (see Sect. 2.4), by connecting the PM lead of the S output of each of the A_{p_j} to 14 dummy controls (t_7). This works because if P is divisible by p_j, the number contained in A_{p_j} will be P 0000000000 and thus positive, while it will be negative in all other cases (this is why we use complements). If a given A_{p_j} stores P 0000000000, and P is thus divisible

[12] Since only integers $2r + 1$ are tested as primes.

by p_j, A_{p_j} has to be reset to the complement of $2p_j$.[13] This was a difficult problem to solve, because only those accumulators that store P 0000000000 should receive a value, and each of these must receive a different value. The problem for the ENIAC to decide which accumulators should receive and which should not, was solved by *directly* connecting the program pulse output terminal of each of the dummy controls of the A_{p_j} to the program pulse input terminal of t_8 of each of the A_{p_j}. This could be done by using a *loaded program jumper* [5, 11.6.1]. Each t_8 of an A_{p_j} is set to receive once through input channel α.

The transmission of 14 different numbers to the 14 A_{p_j}'s is done by using the three function tables and special digit adapters (indicated as "sd" in Fig. 8). The 14 A_{p_j}'s are divided into three groups: A_{p_1}–A_{p_5}, A_{p_6}–$A_{p_{10}}$, $A_{p_{11}}$–$A_{p_{14}}$. In each group, the program pulse output terminal of t_1 of rsp. A_{p_1}, A_{p_6} and $A_{p_{11}}$ is connected to three different program cables. The first of these cables sends a program pulse to function table 1, the second to function table 2 and the last to function table 3. The argument clear switch of each of the tables is set to 0. Without going into the details of this setting, it is important to know that in this specific wiring, the switch is set to 0 so that the function table will transmit the value $f(0)$ to the input channel it is connected to. Each of the function tables contains rsp. one of the following values: M 610142226, M 3438465862 and M 74828694 at place 0 (function value $f(0)$). These numbers are nothing but the concatenation of the values $2p_j$ which have to be sent to those A_{p_j} for which p_j divides $2r + 1$ (A_{p_j} stores P 0000000000). Five addition times after each of the function tables has received a program pulse, each of these values will be sent through the respective numerical cables. The input channels α of A_{p_1}–A_{p_5} are connected to the first numerical cable, those of A_{p_6}–$A_{p_{10}}$ to the second, and those of $A_{p_{11}}$–$A_{p_{14}}$ to the third.

Now, if e.g. accumulator A_{p_1} has been set to receive through α by its dummy control it will receive the value M 610,142,226 through α. A special adapter is inserted at the input terminal α of A_{p_1}. It is used to combine a shifter adapter–which is used to shift the digit lines a certain number of times to the left or to the right–and a deleter adapter–which makes it possible to select only those digits needed. Setting both deleter and shifter in the correct way for A_{p_1}, the number M 0000000006 (instead of M 610,142,226) will be subtracted from the content of A_{p_1}. After this, A_{p_1} will contain M 9,999,999,994 which is the value needed for the sieve to work properly. Now, if at least one A_{p_j} contains P 0000000000 (i.e., P is not prime), a program pulse is sent to the stepper direct input of a stepper from the master programmer (the leftmost stepper in Fig. 8), by t_8 (a number has been received by the function table). The stepper will thus *immediately* cycle from stage 1 to stage 2. Now, a program pulse still has to be received at the stepper input of this stepper in order to activate either II or IV. This is done by using a second stepper, activated by t_1 of A_{p_1} at the beginning of the sieve routine (program cable P1). The first stage of this second stepper is set to 3 in order to delay the program pulse to be sent to the first stepper.[14]

[13] We use $2p_j$ since only numbers of the form $2r + 1$ are sent through the sieve.

[14] In order to calculate the value 3 of stage 1 of the second stepper, we had to take into account, on the one hand, that it takes 5 addition times before a function table emits f(0), and, on the other hand, that if a stage reaches it maximum value at time r, it will only be cycled to the next stage at $r + 2$.

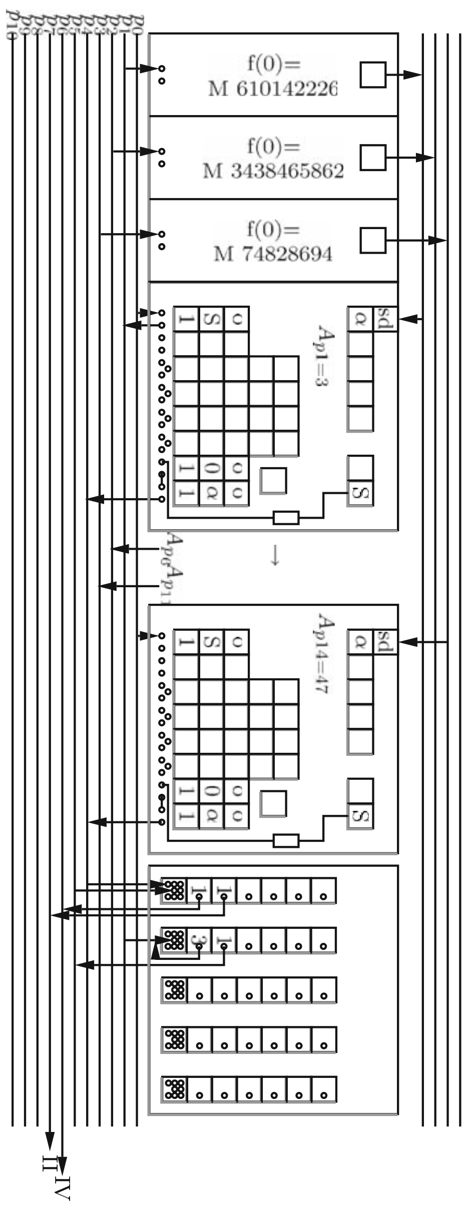


Fig. 8 The set-up of the Sieve on the ENIAC

Figure 8 shows the further details of the wiring of the sieve. It should be noted that both steppers should be reset before II or IV are stimulated by using the stepper clear input. We did not include this in Fig. 8 for reasons of clarity.

3.3.5 IV. Exponent routine

In the exponent routine 5 accumulators are used, i.e. A_{e_1}, A_{e_2}, A_{e_3}, A_P and A_E, with E the number of iterations before $2^n - p = 1$ or $n > 2{,}000$. The first stage of a stepper of the master programmer keeps track of the number of iterations n, i.e. the limit value of the stage is set to 2,000.

Accumulators A_{e_1} and A_{e_2} are used to compute the successive powers of 2 (with $r_n = 2^n \bmod p$) of the exponent routine. At the beginning of the exponent routine, A_{e_1} already contains 2.[15] A_{e_3} is the “discrimination” accumulator. It checks whether (i) $2^n - p > 0$ and (ii) $2^n - p - 2 = -1$ (i.e., an exponent of 2 mod p is found). A_E keeps track of the number of doublings and thus holds the exponent e.

In the first step of this routine, r_{n+1} is computed from r_n. This is done through program cables P0 and P1. As is shown in Fig. 9, a program pulse is sent from the master programmer to cable P0. In A_{e_1}, t_1 is set to send its content twice through A and then clear its content. In A_{e_2}, t_1 is set to receive twice. Thus, the content of A_{e_1} is doubled and stored in A_{e_2}. In the next step, the content of A_{e_2} is sent to A_{e_1}. A_{e_2} is set to clear itself after this transmission. This completes the first step of the exponent routine.

In the next step (cables P2 to P6) it is checked whether $2^n - P > 0$. First $-P$ is sent to A_{e_3} (P2). In order to check if $2^n - P > 0$, the A output terminal of A_{e_3} is wired into an if (b_1) using the second branching method. To this end transceivers t_3 to t_5 are used. If $2^n - P < 0$ a pulse goes to the stepper input (through cable P5) and the next doubling procedure is started, also increasing the content of A_E with 1. If $2^n - P > 0$ then (i) −P is sent to A_{e_1} and (ii) -2 is sent to A_{e_3}. By wiring a second if into $A_{e_3}(b_2)$ using the S output terminal and transceivers t_6 to t_8, it is checked whether or not $2^n - P - 2 = -1$. If yes, the exponent is found and the division routine (V) is started (P10). Else, a pulse is sent to the stepper input (P9), increasing n and, if $n < 2,000$ the next iteration is stimulated. The details of the wiring are shown in Fig. 9. It should be pointed out that this wiring can be made more efficient by one addition time. This is done as follows. Instead of setting A_{e_3} such that it receives r_n from A_{e_2}, it can be set such that it receives the content of A_{e_1} twice at the time A_{e_2} also receives this content twice. In this way the first discrimination can be done one addition time earlier, resulting in the desired speed-up.

3.3.6 V. Division routine

For the division routine 4 accumulators are used, A_P, A_E and the cleared A_{e_2} (which is used to store f, $fE = P - 1$) now called A_f as well as the last unused accumulator A_{20}. Essentially, the division uses only the last three accumulators, A_P only sends its value. A_{20} is the main component of this routine. At the start, A_{20} receives $-P$ from A_P (program cable P0) and then +1 from the CT (program cable P1). After this, the program enters the subprocess used to successively add E, the exponent computed in subroutine IV, to $-P + 1 + nE$ (where n is initially equal to 0). First, A_{20} receives E from A_E (program cable P2) then A_f receives +1 from the CT (program cable P2) , keeping track of the quotient f. After this, the first if (b_1) procedure is activated which is wired in A_{20} and determines whether $-P + 1 + nE$ is positive (program cable P3 to P5). If not, program cable P2 is activated, thus adding E again to A_{20} and +1 to A_f.

[15] This was done in the “increase by 2” subroutine II.

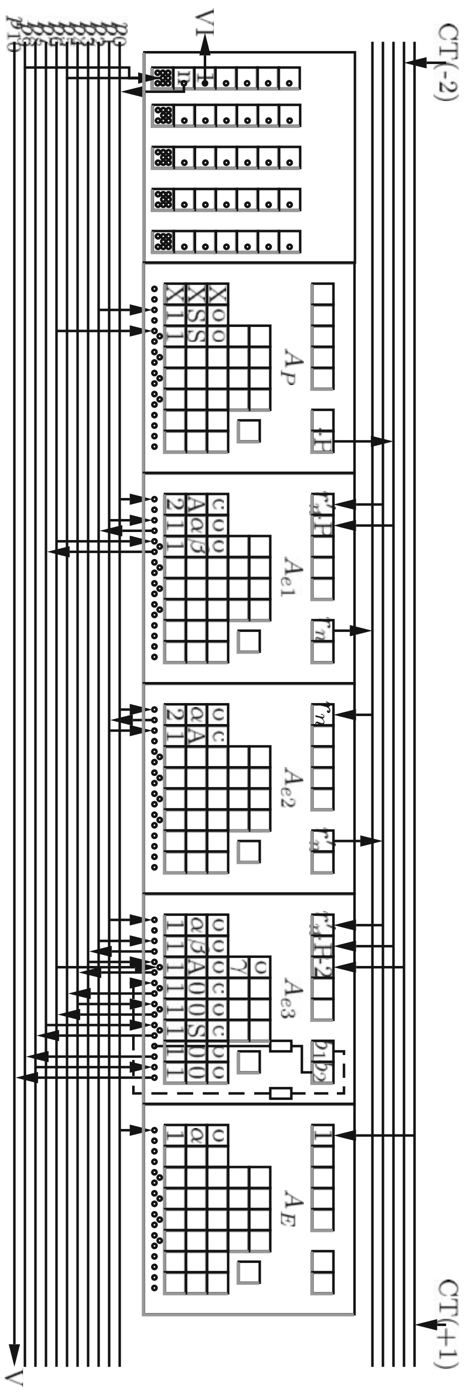
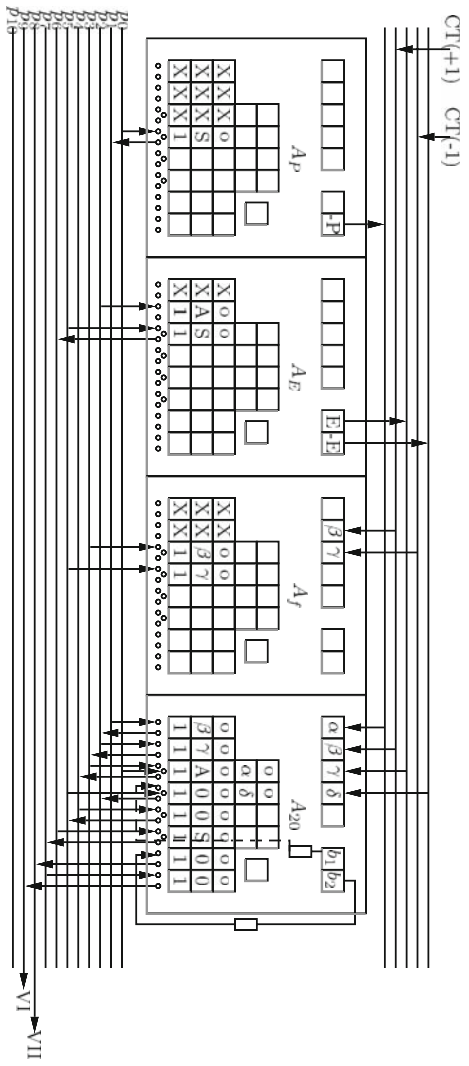


Fig. 9 The set-up of the exponent routine on the ENIAC

Fig. 10 The set-up of the division on the ENIAC



If $-P+1+nE$ is positive the next if (b_2), which is also wired into A_{20} is stimulated (P6 to P9). This is used to determine whether E is a divisor yes or no of $P-1$. Indeed, either $-P+1+nE=E$ (E is a divisor of P) or $0<-P+1+nE<E$. To this end, E is subtracted from $-P+1+nE$. If the result of this subtraction is 0 (and thus a positive number in ENIAC) a divisor is found and subroutine VII is activated. Else, subroutine VI.[16] For the details of the wiring, the reader is referred to Fig. 10.

[16] Note that the value of f needs, at the end of the routine, a correction of -1. This is achieved through cable P5.

3.3.7 VI. Erase exponent calculation

In order to erase the exponent routine, use is made of the selective clear switches on the accumulators. For each accumulator that needs to be erased, this switch should be set to clear [5, Sect. 4.1.5.]. Those accumulators that are set as such will be cleared when they receive the selective clear signal from the initiating unit [5, Sect. 2.4.].

3.3.8 VII. Print p, e and f

Besides the A and S output channels, some of the accumulators have static outputs. The static output of 80 decade counters and 16 PM counters are directly connected to the printer resulting in a total of 8 accumulators that are hard-wired to the printer. In this sense it suffices if A_P, A_E and A_f are of such nature. Of course, one needs to use the interlock of the printer, so that the next process only starts after the printer has received the to be printed values ("buffered" values to be printed). This takes about 0.4 s [5, Sect. 9.1.4]. Then, the next routine is stimulated and the punching begun.The actual punching takes another 0.42 s.

3.4 Time estimation

Because of, on the one hand, the speed of ENIAC, and, on the other hand, the possibility of executing certain processes in parallel, Lehmer's program is very fast. The "addition of 2" routine (II) only takes 1 addition time, the sieve (III) 7 addition times (independent of the size of P), the printing routine (VII) about 0.4 s [5, Sect. 9.1.4][17] and the erase exponent calculation 1 addition time (VI). Now, for the exponent routine, in the worst case the routine has to be executed 2,000 times, while one iteration takes 11 addition times[18] which means that the whole routine takes at most 22,000 addition times. In order to estimate the worst case for the division routine, note that at most $\log_2 P$ subtractions are needed. The main iterative loop, one subtraction and one discrimination takes up 4 addition times, thus the worst case amounts to $3\log_2 P$. This means that, in the worst-case, one iteration of Lehmer's program takes about $24009 + 3\log_2 P$ ($\approx$4 s) addition times.

4 Discussion

"In fact, the programmer is a kind of engineer."
[31, p. 1250]

Our reconstruction of Lehmer's program is part of a series of three reconstructions. It is our aim to describe, reconstruct and analyze three programs run on (the non-rewired) ENIAC: one by Hartree, one by Curry and the present one by Lehmer. These

[17] This is the case because for this program the puncher can do its work, while a new loop of the program is started.

[18] Using the speed-up of one addition time as explained in Sect. 3.3.5.

programs display different aspects of early programming practices. With Curry, logical aspects are emphasized, with Hartree, the translation of a differential equation into (discrete) values and program steps. With Lehmer, the parallelism of his computation is the salient feature.

Our reconstruction of Lehmer's program conveys an impression of what it was like to program the ENIAC. In particular, it is clear that the "baroque" conditional branchings complicate the set-up of programs considerably, and that the parallelism of the units is a salient feature of the ENIAC before 1947. As Barkley Fritz wrote [2, p. 31],

> Anyone now doing research in parallel computing might take a look at ENIAC during this first time period, for indeed ENIAC was a parallel computer with all of the problems and opportunities that this entails.

Unfortunately, this parallelism was hardly ever used, because the synchronization of the various branches is a hard-to-solve problem, especially with a hard-to-program machine. For often occurring parallel but asynchronous processes like reading and punching punch cards and division, interlocks were provided for that put all other processes (units) on hold as long as this particular process (unit) was busy. A generally applicable synchronization device was not available. Simpler synchronizations of parallel processes were implemented on the ENIAC. One instance is the second branching method (2 branches), another, Lehmer and Mauchly's sieve (14 synchronous branches). Interestingly, when in the 1970ies models of parallel programming were intensively studied, the sieve procedure was rediscovered as an exemplary exercise in parallel but now structured programming [32,33, pp. 27–32; p. 674].

Acknowledgments We want to thank Martin Carlé (Humboldt University Berlin) for stimulating us to get involved into the ENIAC's technical details and wirings. This reconstruction is intended as a contribution to the ENIAC NOMOI project. We would also like to thank the organizers of *Computability in Europe* 2008, Athens, Greece for having given us the opportunity to present this paper during the special session *Algorithms in the History of Mathematics* and for their invitation to this special issue. Special thanks go to the anonymous referee for his/her useful comments. The pictures in this article were made with the help of the packages dpic and PSTricks. This research was supported by the Flemish Fund of Scientific Research (FWO - Flanders) and the Alexander von Humboldt Foundation.

References

1. Alt, F.: Archaeology of computers—reminiscences, 1945–1947. Commun. ACM **15**(7), 693–694 (1972)
2. Fritz, W.B.: ENIAC—a problem solver. Ann. Hist. Comput. IEEE **16**(1), 25–45 (1994)
3. Goldstine, H.H.: The Computer—From Pascal to von Neumann. Princeton University Press, Princeton (1973)
4. McCartney, S.: ENIAC—The Triumphs and Tragedies of the World's First Computer. Walker, New York (1999)
5. Goldstine, A.K.: Report on the ENIAC, technical report I. Technical Report, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia (1946)
6. Clippinger, R.F.: A logical coding system applied to the ENIAC. BRL 673, Ballistic Research Laboratories, Aberdeen Proving Ground (1948)
7. Neukom, H.: The second life of ENIAC: ENIAC's converter code. Ann. Hist. Comput. IEEE (web extra) (2006)

8. Bauer, F.L.: Wer erfand den Von-Neumann-Rechner? Inform. Spektrum **21**, 84–89 (1998)
9. Mol, L.D., Bullynck, M.: A week-end off. The first extensive number-theoretical computation on the ENIAC. In: Beckmann, A., Dimitracopoulos, C. L., öwe, B. (eds.) Computability in Europe 2008 Logic and Theory of Algorithms Logic and Theory of Algorithms, Fourth Conference on Computability in Europe, CiE 2008, Athens, Greece, June 2008, Proceedings, Lecture Notes in Computer Science, vol. 5028, pp. 158–167 (2008)
10. Burks, A.W., Burks, A.R.: The ENIAC: first general-purpose electronic computer. Ann. Hist. Comput. IEEE **3**(4), 310–399 (1981)
11. Zoppke, T., Rojas, R.: The virtual life of ENIAC: simulating the operation of the first electronic computer. IEEE Ann. Hist. Comput. **28**(2), 18–25 (2006)
12. Goldstine, H., Goldstine, A.: The electronic numerical integrator and computer (ENIAC). Math. Tabels Other Aids Comput. **2**(15), 97–110 (1946)
13. Hartree, D.R.: The ENIAC, an electronic computing machine. Nature **158**(4015), 500–506 (1946)
14. van der Spiegel, J., Tau, J.F., Ala'ilima, T.F., Ang, L.P. : The ENIAC–history, operation and reconstruction in VLSI. In: Rojas, R., Hashagen, U. (eds.) The First Computers–History and Architectures, pp. 121–178. MIT Press, Cambridge, MA (2000)
15. Hansen, P.: A Java simulation of the ENIAC. Master's thesis, Universität Osnabrück, Osnabrück (2004)
16. Marcus, M., Akera, A.: Exploring the architecture of an early machine: the historical relevance of the ENIAC machine architecture. IEEE Ann. Hist. Comput. **18**(1), 17–24 (1996)
17. Cope, W.F., Hartree, D.R.: The laminar boundary layer in compressible flow. Philos. Trans. R. Soc. Lond. A Math. Phys. Sci. **241**, 1–69 (1948)
18. Goldstine, H.H., von Neumann, J.: Planning and coding problems for an electronic computing instrument. Institute for Advanced Study, Princeton, NJ (1947/1963). Reprinted in: Taub, A.A. (ed.), John von Neumann, Collected Works, pp. 80–151. Pergamon Press, Oxford (1963)
19. Curry, H.B.: The logic of program composition. In: Applications scientifiques de la logique mathématique. Actes du 2e Colloque International de Logique Mathématique, Paris, 25–30 août 1952, Institut Henri Poincaré, pp. 97–102. Gauthier-Villars, Paris (1954)
20. Knuth, D.E., Pardo, L.T.: The early development of programming languages. In: Belzer, J., Holzman, A., Kent, A. (eds.) Encyclopedia of Computer Science and Technology, pp. 419–496. Dekker, New York (1979)
21. Lehmer, D.H.: A history of the sieve process. In: Howlett, J., Metropolis, N., Rota, G.C. (eds.) A History of Computing in the 20th Century, pp. 445–456. Academia, New York (1980)
22. Akera, A. (interviewer): Franz Alt interview. ACM Oral History Interviews (2006)
23. Lehmer, D.H.: Tests for primality by the converse of Fermat's theorem. Bull. Am. Math. Soc. **33**, 327–340 (1927)
24. Lehmer, D.H.: On the converse of Fermat's theorem. Am. Math. Mon. **43**(6), 347–354 (1936)
25. Lehmer, D.H.: Maurice Kraitchik, Recherches sur la Théorie des Nombres, v. 1, Paris, 1924 (errata). Math. Tables Other Aids Comput. **2**(19), 313 (1947)
26. Lehmer, D.H.: On the factors of $2^n \pm 1$. Bull. Am. Math. Soc. **53**(2), 164–167 (1947)
27. Lehmer, D.H.: On the converse of Fermat's theorem II. Am. Math. Mon. **56**(5), 300–309 (1949)
28. Hartree, D.R.: Calculating Instruments and Machines. University of Illinois Press, Urbana (1949)
29. Lehmer, D.H.: The influence of computing on mathematical research and education. In: Lasalle, J. (ed.) The Influence of Computing on Mathematical Research and Education, Proceedings of Symposia in Applied Mathematics, vol. 20, pp. 3–12. American Mathematical Society, Providence (1974)
30. Hoffleit, D.: A comparison of various computing machines used in the reduction of Doppler observations. Math. Tabels Other Aids Comput. **3**(25), 373–377 (1949)
31. Hopper, G.M., Mauchly, J.: Influence of programming techniques on the design of computers. Proceedings of the IRE pp. 1250–1254. (1953)
32. Dijkstra, E.W.: Notes on structured programming. In: Structured Programming, pp. 1–82. Academic Press, New York (1972)
33. Hoare, C.A.R.: Communicating sequential processes. Commun. ACM **21**(8), 666–678 (1978)