

A Week-End Off: The First Extensive Number-Theoretical Computation on the ENIAC

Liesbeth de Mol, Maarten Bullynck

▶ **To cite this version:**

Liesbeth de Mol, Maarten Bullynck. A Week-End Off: The First Extensive Number-Theoretical Computation on the ENIAC. Lecture Notes in Computer Science, 2008, 5028, pp.158-167. ⟨10.1007/978-3-540-69407-6_19⟩. ⟨hal-01396832⟩

HAL Id: hal-01396832

https://lilloa.hal.science/hal-01396832v1

Submitted on 15 Nov 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



HAL Authorization

A week-end off. The first extensive number-theoretical computation on the ENIAC

Liesbeth De Mol[1⋆] and Maarten Bullynck[2⋆⋆]

[1] Center for Logic and Philosophy of Science, University of Ghent, Blandijnberg 2, 9000 Gent, Belgium
elizabeth.demol@ugent.be
[2] IZWT, Gaussstrasse 20, 42119 Wuppertal, Germany and REHSEIS, Paris, France
bullynck@uni-wuppertal.de

Abstract. The first extensive number-theoretical computation run on the ENIAC, is reconstructed. The problem, computing the exponent of 2 modulo a prime, was set up on the ENIAC during a week-end in July 1946 by the number-theorist D.H. Lehmer, with help from his wife Emma and John Mauchly. Important aspects of the ENIAC's design are presented and the reconstruction of the implementation of the problem on the ENIAC is discussed in its salient points.

Key words: ENIAC, Derrick H. Lehmer, number theory, Fermat's little theorem, early programming, parallelism, prime sieve

1 Introduction

Just too late to help win the Second World War, just in time to start off the computer age, the first digital electronic general-purpose computer (See e.g. [2]), the ENIAC, was presented to the public February 15, 1946 at Penn University. The Ballistic Research Laboratories (Aberdeen Proving Ground) had "assembled a 'Computations Committee' to prepare for utilizing the machine after its completion" [1, p. 693], and the ENIAC was extensively test-run during its first months.

One of the members of this committee was the number-theorist Derrick H. Lehmer who spent a Fourth-of-July weekend testing the ENIAC. The Lehmer family – Derrick, Emma and two teenage kids – arrived at the Moore school on Friday 5 p.m. where they met John Mauchly, who was, together with Presper Jr. Eckert, the father of the ENIAC-design. He helped them set up the ENIAC for the implementation of an interesting number-theoretical problem and stayed on as an operator through the week-end [10, p. 451]. Lehmer's computation was important to the post-war reputation of electronic computers among mathematicians.

⋆ Postdoctoral Fellow of the Fund for Scientific Research – Flanders (FWO)
⋆⋆ Postdoctoral Research Grant, Alexander-von-Humboldt-Stiftung

The ENIAC was an electronic and highly parallel machine. As a consequence it was revolutionary fast in its time. However, setting up a program on the ENIAC was time-consuming. The original control system of the ENIAC was decentralized, its elements were distributed over the different units of the machine. The programming had to be done "directly", i.e., there was no converter code, let alone a programming language available to set up a 'program'. Instead, one had to connect the different parts of the machine through cables and adaptors. Only in 1948 was the ENIAC rewired into a serial computer that could be set up using a converter code that selected subroutines from a function table, thus simulating a stored-program computer [11]. The original "local programming" method can best be described "as analogous to the design and development of a special-purpose computer out of ENIAC component parts for each new application." [3, p. 31] Or as Jean Bartik, one of the ENIAC's female programmers, put it: 'The ENIAC was a son-of-a-bitch to program.'

The particularities of programming the ENIAC make it interesting for present-day computer scientists. As W.B. Fritz, one of the ENIAC-operators remarks: "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 this entails." [3, p. 31] More importantly even, 'programming' the ENIAC invites one to question many things taken for granted nowadays, amongst them the relation between the user and the machine.

In this paper, we will present Lehmer's problem and show how to set it up on the ENIAC. Given the complexity of 'programming' (actually wiring) the ENIAC, we will not be able to provide all details here. Instead we will merely outline our reconstruction, providing details only for certain aspects of the set-up.[3]

1.1 How a number-theorist got involved with computers

Derrick H. Lehmer (1905-1991) was born into number theory. His father, Derrick N. Lehmer, was a reputed number-theorist, best known for his factor table up to 10,000,000 and his stencil sheets to find factors of large numbers. Already as a young boy D.H. devised a way, using bicycle chains, to mechanize these stencils, building the first electro-mechanical number sieve in 1926, the first photo-electric sieve in 1932. Given this lasting interest in making mathematical tables (always a time-consuming job) and particularly in mechanizing sieves, it is clear that D.H. Lehmer perfectly fitted into the 'Computations Committee'.

Lehmer also contributed in other ways to the advance of the computer age. He helped R.C. Archibald' to publish the important journal *Mathematical Tables and other Aids to Computation* (known as MTAC) from 1943 to 1950, and served as the editor-in-chief himself until 1959. This journal was an important channel of publication during the early years of computing, offering a fast way of

[3] We want to thank Martin Carlé (Humboldt University Berlin) for his stimulation to get involved into the ENIAC's technical details and wirings. This reconstruction is intended as a contribution to the ENIAC NOMOI project.

getting new results in print, and it reads like the diary-like records of advances in hardware and computing techniques from 1943 to 1959. Lehmer also often ventilated his strong opinions on computing in this journal, making a case for seeing mathematics (especially number theory) as an observational science for which data had to be gathered through computation. Lehmer argued strongly and convincingly that the computer would change the world of mathematics.

1.2 The Structure of the 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 [2, p. 376] one of the notable exceptions being Lehmer's program.

The ENIAC had a modular architecture. It comprised 20 accumulators, a multiplier, a divider and square rooter, a constant transmitter, 3 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 are the ENIAC's main memory storage units. Function tables had to be set manually with switches, the constant transmitter could store upto 16 numbers read from a punched card (before or during the computation) and 4 numbers that had to be set manually with switches.

The ENIAC had two kinds of electronic circuits: the *numerical circuits* for storing and processing electric signals representing numbers and *programming circuits* for 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. All units had to be programmed locally. The program switches, located on the front face of the units, had to be set before a computation started to specify which operations had to be performed. It will become clearer in the remainder of the paper how to control the order of the operations.

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, so we speak also of an addition time instead of a cycle.

We will now detail the design of the accumulator and the master programmer and explain how to set up some basic processes such as branching and iteration. For more details we refer to [2,5] and especially the detailed *Report on the ENIAC* by Adèle Goldstine [4].[4]

[4] We would also like to refer to Till Zopke's Java-applet ENIAC simulation [12], that proved handy for testing some easy cablings.

The accumulator The accumulators were the main arithmetic units of the ENIAC and could be used to add or subtract. Each accumulator held a 10-place decimal number and a sign, stored in ten decade ring counters and a PM counter. It had 5 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 outputs. The program part of the accumulator consisted of 16 program controls: 4 receivers and 12 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 α to ϵ, A, S, AS or 0, determining whether the accumulator should receive or transmit a number, or do nothing) and a repeat switch (with which it could either receive or transmit up to 9 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.

The master programmer The master programmer provided a certain amount of centralized programming memory. It consisted of 10 independently functioning units, each having a 6-stage counter (called the stepper), 3 input terminals (the stepper input, direct input and clear input), and 6 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 1 to 5 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) cycled *immediately* to the next stage, and a pulse at the clear input (CI) reset 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.

Branching The ENIAC was capable of discriminating between program sequences by examining the magnitude of some numerical result. This "magnitude discrimination" or "branching" was possible because 9 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) could then be 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 the program pulse

input terminal of an otherwise unused 'dummy (program) control' by using a special adaptor [4, Sec. 4.5].

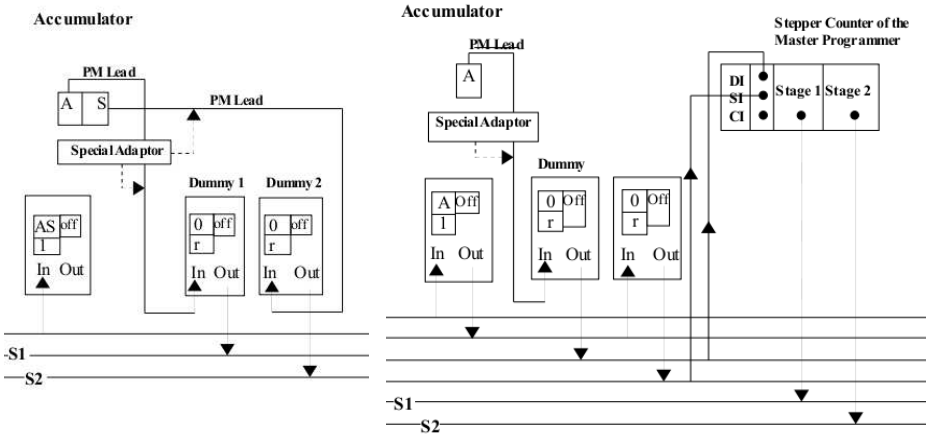


Fig. 1. Wiring schemes for each of the branching methods. The if-routine is started when a program pulse is received at the first program cable. Depending on the sign of the number in the accumulator, either subroutine S_1 or S_2 will be executed.

There were basically two ways to do this branching: either by using the two output channels A and S of an accumulator and two dummy program controls, or, by using only one output channel, A or S, one dummy control and the master programmer. Fig. 1 shows *possible* implementations of these two 'if' methods. They were reconstructed on the basis of the (limited) information on branching in [2,4].

2 Lehmer's ENIAC 'program'

2.1 The mathematical problem

Primality tests and factoring methods belong to the most important topics in number theory. One class of primality tests are (partial) converses of Fermat's little theorem. Unfortunately, the direct converse of the theorem is false in general: If $a^{p-1} \equiv 1 \bmod p$ for an arbitrary a (not dividing p), then p is often though not always a prime. A way out of the failing general converse is to list all exceptions, i.e., all composite p for which $a^{p-1} \equiv 1 \bmod p$ is true. Taking a equal to 2 is computationally the most advantageous choice. To compute the composite p's for which $2^{p-1} \equiv 1 \bmod p$ one needs a table of all exponents e of 2 modulo the prime p, e being the least value n such that $2^n \equiv 1 \bmod p$ and e is some divisor of $p-1 = ef$. Kraïtchik published such an exponent table in 1924 for $p < 300000$,

but it contained quite some errors [6]. D.H. Lehmer now proposed to use the ENIAC to compute a table of exponents correcting and extending Kraïtchik's table to $p < 4.5 \cdot 10^6$. The results of Lehmer's calculation on the ENIAC were published as a list of corrigenda to Kraïtchik's table in 1947 (MTAC 2 (19) p. 313). In 1949 an article discussing some computational details of setting up this 'program' on the ENIAC appeared [7].

2.2 Description of the main steps of the computation

As Lehmer remarks right at the beginning of his description of the ENIAC set-up: "The method used by the ENIAC to find the exponent of 2 modulo p differs greatly from the one used by human computers" [7, p. 301]. A number-theorist knows that the exponent e of 2 so that $2^e \equiv 1 \bmod p$ (p prime) is either a divisor of or equal to $p-1$, and can thus restrict himself to doing trial divisions with suitable divisors of $p-1$ only. On the ENIAC, however, it is more expeditive to compute 2^t for all $t < p-1$. This 'idiot approach' takes, in the worst case, "less than 2.4 seconds, less time than it takes to copy down the value of p", whereas the sophisticated method requires "much outside information via punched cards [...] to be prepared by hand in advance." [7, p. 302]
Lehmer's flow-diagram [7, p. 303] gives the following main steps of the ENIAC computation:
Step 1. Initiation and preliminary set-up, go to Step 2.
Step 2. Increase p by 2, goto Step 3.
Step 3. Sieve on p: Is p divisible by a prime ≤ 47? Yes/No, goto Step 2/Step 4.
Step 4. Exponent routine to find e. Is $e > 2,000$? Yes/No, goto Step 7/Step 5.
Step 5. Does e divide $p-1$? Yes/No, goto Step 6/Step 7.
Step 6. Punch p, e and f ($p - 1 = ef$), goto Step 7.
Step 7. Erase exponent calculation, goto Step 2.

The sieve set-up (step 3) can take advantage of the parallelism of the ENIAC's hardware. Thus implemented, this becomes a fast method for minimizing the chance that $p = 2n + 1$ is not a prime. The sieve implemented on the ENIAC sieved out all numbers $p = 2n + 1$ having prime factors ≤ 47. As Lehmer notes [7, p. 302], about 86 percent of the composites were eliminated after step 3 (sieve). The remaining 24 percent were required to pass a further test: namely $p - 1$ must be divisible by e (step 5). This requirement is so strict that the remaining number of composites is very small. Finally, these were eliminated by hand through comparison with D.N. Lehmer's list of primes.

The exponent e is calculated by a simple recursive routine (step 4), building up a sequence of positive integers r_k, where r_k is nothing but the remainder on division of 2^k by p. The start value r_1 is set to 2. Given r_k, the next value r_{k+1} is set to $2r_k - p$. If $2r_k - p < 0$, r_{k+1} is set to $2r_k$. If $2r_k - p > 0$ then the ENIAC checks whether r_{k+1} - 2 is negative. If the reply is yes, then r_{k+1} is 1 and $e = k + 1$. If no, then the ENIAC checks whether $k + 1 = 2001$. If not, then r_{k+2} is the next value computed. If yes, then the ENIAC gives up the search for e and tries the next value of p.

2.3 Outline of the Set-up of the computation on the ENIAC

We will now give an outline of how to wire the ENIAC to perform Lehmer's computation. Due to lack of space not all details can be given. We will focus on the parallel set-up of the sieve and then give a sketch of the complete reconstruction. The reconstruction is based upon the 7-step-diagram (sec. 2.2) and some implementational details given in Lehmer's [7,8,9]. Important to note: Lehmer's description does not completely determine the actual machine implementation, the design of the ENIAC, however, seriously limits the possibilities.

Wiring a sieve Sieves check for every subsequent number of a specific sequence if the number fulfills j conditions/congruences. If it fulfills one or more congruences, it is sieved out, if not, the number is let through. The best known sieve, Eratosthenes's, checks for every natural number n if it is divisible by a prime ($n \equiv 0 \bmod p_j$?), the numbers that pass are relative prime to the primes p_j.

Our sieve implementation uses an accumulator A_{p_j} for each prime $p_j \leq 47$, ($1 \leq j \leq 14$) except for 2 (See sec. 2.2). To begin (step 1), each A_{p_j} is set to the complement of $p_j - 1$. E.g. $A_{p_{14}}$ will contain M 9999999954. An initiating pulse is sent to the first transceiver (T1) of all A_{p_j}'s, its operation switch (OS) is set to α and the repeat switch (RS) to 1, as well as to the constant transmitter (CT) such that it will transmit the number 2. This results in the addition of 2 in all 14 accumulators. The CT then transmits a program pulse (PP), finishing the first cycle of the computation. The next steps of the computation check for each of the 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 the second branching method (See Sec. 1.2), by connecting the PM lead of the S output of each of the A_{p_j} to 14 dummy controls (T2). 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 by p_j, A_{p_j} has to be reset to the complement of $2p_j$.[5] 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 number. 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 one of the transceivers (T3) of each of the A_{p_j}. This could be done by using a *loaded program jumper* [4, 11.6.1]. Each T3 of an A_{p_j} is set to receive once through input channel α, β or γ depending on the group A_{p_j} belongs to. The transmission of 14 different numbers to the 14 A_{p_j}'s is done by using the three function tables and special digit adaptors. 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 PP output terminal of T1 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 PP to function table 1, the second to function table 2 and the last to

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

function table 3. The argument clear switch of each of the tables is set to O. Without going into the details of this setting, it is important to know that in this specific wiring, the switch is set to O 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 64828694 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 input channels α, β and γ.

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 610142226 through α. A special adaptor is inserted at the input terminal α of A_{p_1}. It is used to combine a shifter adaptor – which is used to shift the digit lines a certain number of times to the left or to the right – and a deleter adaptor – 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 610142226) will be subtracted from the content of A_{p_1}. After this, A_{p_1} will contain M 9999999994 which is the value needed for the sieve to work properly.

Figure 2 shows the details of the wiring of this sieve for A_{p_1}. Note that

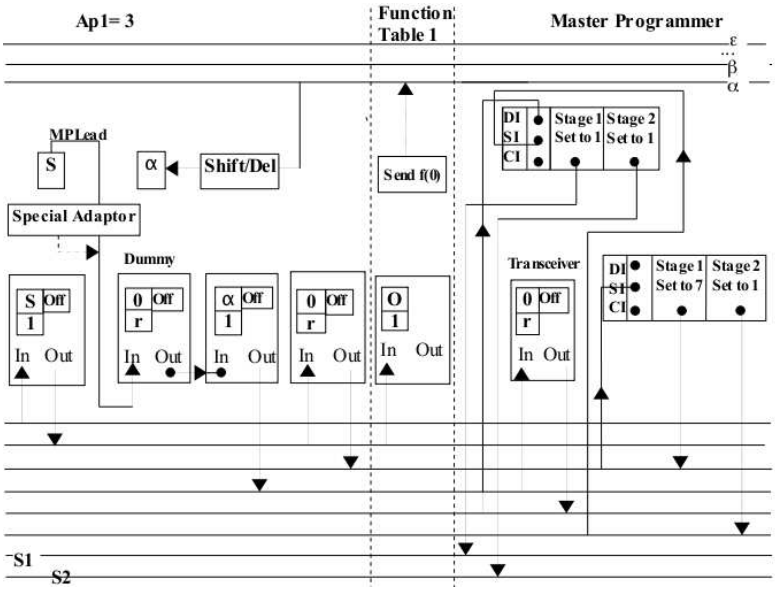


Fig. 2. Parallel Implementation of the sieve for A_{p_1}, $p_1 = 3$

two stepper counters plus two extra transceivers are used to let ENIAC decide

whether $2r+1$ passes the sieve test. The first transceiver, situated in Accumulator A_{p_1} is needed to delay this decision through the second stepper counter (waiting until the function tables have sent their values $f(0)$). The second transceiver (situated right next to one of the stepper counters) is used to send a program pulse to the direct input of a second stepper, thus "making" the decision. The whole sieve process takes 12 addition times and is thus very fast. Fundamental in this wiring was the synchronization of all the different steps.

Outline of the complete program We will now provide an outline of our reconstruction of the complete computing-the exponent-of-2-modulo-p-program (Sec. 2.2). Steps 2 and 3 were already discussed in the previous paragraph.
Step 1. The setting of the A_{p_j}'s as well as the three function tables has already been discussed. 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.
Step 2. This was already discussed. Besides the A_{p_j}'s also the transceiver of A_P, as well as a transceiver of one other accumulator used in step 4 (A_{e_1}), should be set to receive through α.
Step 3. This was already discussed.
Step 4. In this subroutine 5 accumulators are used, i.e. A_{e_1}, A_{e_2}, A_{e_3},A_P and A_E (which is used to count the number of iterations done before $2_{r_k} - p = 1$ or $k > 2000$ and thus to determine the exponent). Three stepper counters are used. One is used to check whether $k > 2000$ and receives a PP with each iteration of the subroutine. The others are used in combination with dummy controls: one to check whether $2_{r_k} - p > 0$ and one to check whether $2_{r_k} - p = 1$. We do not have the space here to give the details of the wiring.
Step 5. For this routine 4 accumulators are used, including A_P, A_E and the cleared A_{e_1} (which is used to store f). Besides these three the last unused accumulator A_{20} is introduced into the computation. Again we cannot provide details here. It is important that at the start of the subroutine, A_{20} receives P from A_P and next -1 from the CT. At the end of the computation, A_{e_1} will contain f, the number of times e can be subtracted from $p - 1$.
Step 6. There are several ways to wire this routine. One way is to use for A_P, A_{e_1} and A_E accumulators for which there is a static output to the printer.
Step 7. This can be done by using the clear-correct switch for the accumulators involved (except for A_P).

3 Discussion

Our proposed reconstruction of one of the first extensive computer 'programs', executed 1946 on the ENIAC, cannot be said to be complete nor definitive yet, but we project to present the complete wiring with discussion of difficult points in another, more extended paper in the near future. Already our outline brings out some salient points.

It is clear that 'programming' the ENIAC is perhaps sometimes nearer to engineering than to programming as we know it today. The 'if's are intricate digit-pulse-to-program-pulse conversions, which, in combination with the steppers of the master programmer, can be used to sequence the computation; different adaptors have to be inserted at the appropriate places, and the parallelism has to handled subtly by synchronizing the numerical and the program parts of the wiring. This polymorphy of combining the units, cables and adaptors of the ENIAC allows for many small tricks to be implemented in the 'program', but forbids a general approach to setting up the ENIAC. Such a general approach only arrived with the ENIAC's rewiring into a serial machine, using the function tables for 'indexing' the subroutines.

Concluding, we would like to encourage research on early computer programs on the ENIAC or on other early computers. Their study and analysis might contribute to understand the beginnings and achievements of the computer age. It might especially clarify the development of programming techniques and computational methods in correlation with the development of the hardware, as well as the evolution of the interaction and interface between the operator/programmer and the computer.

References

1. Franz Alt. Archaeology of computers – reminiscences, 1945–1947. *Comm. ACM*, 15(7): 693–694, July 1972.
2. Arthur W. Burks and Alice R. Burks. The ENIAC: First general-purpose electronic computer. *IEEE Ann. Hist. Comp.*, 3(4): 310–399, 1981.
3. W. Barkley Fritz. Eniac – A problem solver. *IEEE Ann. Hist. Comp.*, 16(1): 25–45, 1994.
4. Adele K. Goldstine. Report on the ENIAC, Technical report I. Technical report, Moore School of Electrical Engineering, University of Pennsylvania, Philadelphia, June 1946. Published in 2 vols.
5. H.H. Goldstine and Adele Goldstine. The electronic numerical integrator and computer (ENIAC). *Math. Tables Aids Comp.*, 2(15): 97–110, 1946.
6. Derrick H. Lehmer. On the converse of Fermat's theorem. *Amer. Math. Monthly*, 43(6): 347–354, 1936.
7. Derrick H. Lehmer. On the converse of Fermat's theorem II. *Amer. Math. Monthly*, 56(5): 300–309, 1949.
8. Derrick H. Lehmer. The sieve problem for all-purpose computers. *Math. Tables Aids Comp.*, 7(41): 6–14, 1953.
9. Derrick H. Lehmer. The influence of computing on research in number theory. In Proc. Sympos. Appl. Math. 20, p. 3–12, Amer. Math. Soc., 1974.
10. Derrick H. Lehmer. A history of the sieve process. In J. Howlett, N. Metropolis, and G.-C. Rota, editors, *A History of Computing in the Twentieth Century*, p. 445–456. Academia Press, New York, 1980.
11. Hans Neukom. The second life of ENIAC. *IEEE Ann. Hist. Comp.*, 28(2): 4–16, 2006.
12. Till Zoppke and Raul Rojas. The virtual life of ENIAC: Simulating the operation of the first electronic computer. *IEEE Ann. Hist. Comp.*, 28(2): 18–25, 2006.