DOI:10.1145/3624731

Mathematician and inventor Charles Babbage's code sketches are the first attempt to specify how to mechanize complex algorithms with a computer.

BY RAÚL ROJAS

The First Computer Program

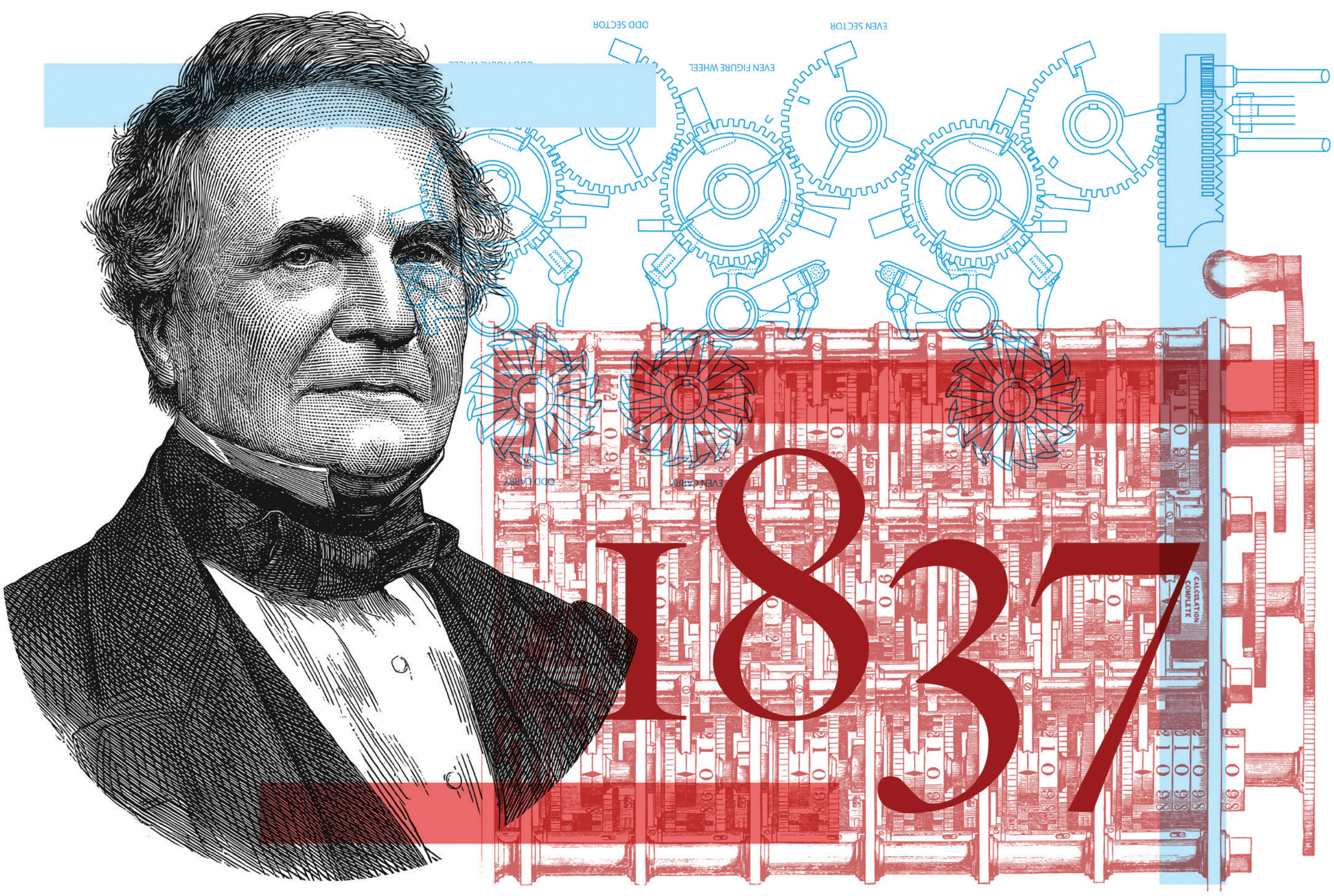
THIS ARTICLE IS a description of Charles Babbage's first computer program, which he sketched out almost 200 years ago, in 1837. The Analytical Engine (AE), the computer for which the program was intended, did not actually exist; sadly, it was to remain unfinished. Only some portions of Babbage's calculating machine were built during the lifetime of the English mathematician and inventor. Had it been completed, it would have been the world's first computer.[1,3] Of course, many algorithms had already been described before Babbage—for computing the greatest common divisor (GCD), for example—but Babbage's code is the first attempt to specify how to mechanize complex algorithms with a computer. This was the heyday of the first industrial revolution, the age of steam engines and mechanization. Electricity, light bulbs, and telephones were still decades away, but the computer was taking shape on Babbage's drawing board.

Babbage (1791–1871) developed detailed blueprints for the AE and sketched 26 programming examples between 1836 and 1841. The Science Museum in London has digitized the Babbage Archives so that today we can inspect existing diagrams of the AE (with first drafts from 1835 on) and the 26 programming examples from the comfort of a home computer. In a 2021 paper titled "The Computer Programs of Charles Babbage," I discuss the architecture of the AE and review some of its programs.[5]

The design of the AE consisted of a processor for the four arithmetic operations, called the "mill," and a separate memory for decimal integers, called the "store" (Babbage once considered building the AE to be able to store hundreds of variables with 40 digits).[2] This separation of processor and memory is typical of today's computers. However, the program was not stored in memory, but encoded on punched cards that were read one at a time. One stream of cards was used by the processor, while a separate stream was used for the memory (for the addresses of the arguments). In this article, I sometimes refer to the "mill" as the processor and the "store" as the memory of the machine. When Babbage talks about the contents of memory cells, he calls them "variables." The address of a variable is its subindex. For example, the memory cell with address 3 would be v_3. The AE performed all calculations using fixed-point arithmetic. The number of digits

» key insights

- **The Analytical Engine (AE) would have been the first computer, had it been finished.**
- **The AE had a processor ("mill") and a separate memory ("store").**
- **The AE could compute the basic arithmetic operations and was programmed using strings of punched cards.**
- **Charles Babbage wrote the first computer program in 1837.**



to the right of the decimal point could be chosen by the programmer.

For example, when the operation punched card was read to calculate an addition, the mill would go into the "addition" state, while the variable cards would instruct the store to retrieve the contents of the addresses of the two needed arguments and send them to the mill. Since the processor waits for its arguments and the store waits for the result, this automatically synchronizes the operation and the variable cards.

Reading a variable was a destructive operation; it reset the variable to zero. However, it was possible to store the complement of this variable in another memory address while sending it to the mill. Re-reading this complement and transferring it back to the original memory address (complementing again) restored this address to its original contents.

IMAGE BY ANDRIJ BORYS ASSOCIATES, USING SHUTTERSTOCK

The First Code Table

The program we are discussing was written by Babbage in 1837. The title of the sketch is "Notations and Calculations," and the first line reads "No. 1. 4 August 1837." This was the first of the series of programs that Babbage decided to carefully sketch out, and we even have the date of the program. The Babbage Archive lists this program as "BAB L1". There is a small program for computing a simple formula in the archive, but it is undated and unnumbered (BAB L26). The Babbage Archive dates this code fragment to August 1837, without further information. There is also a sketch of how to assign coefficients of a linear equation to memory addresses, but no code. The date of this fragment is given as 1836.

The program in BAB L1 deals with the solution of a system of two linear equations in two variables. It is easy to find a closed formula for the result. Babbage considered the two linear equations

$$ax + by + c = 0$$
$$a'x + b'y + c' = 0.$$

We have six parameters, the six coefficients a, b, c, a', b', c', in the two equations. The solution for x is

$$x = \frac{bc' - b'c}{b'a - ba'}.$$

Given x, the solution for y is

$$y = (-ax - c)/b$$

In the first expression, we assume that the denominator $b'a - ba'$ is non-zero, while in the second we assume that b is non-zero, so that the solutions exist. Babbage did not check these two conditions in his program.

First, Babbage assigns the six coefficients a, b, c, a', b', c', to the six variables v_1 to v_6 in the AE's memory. He then computes successively the intermediate results $b'a, b'c, ba', bc', bc' - b'c, b'a - ba'$, and finally the quotient for finding x. The complete computation for x requires four multiplications, two subtractions, and a final division. That is a grand total of five "big" and two "small" operations (as Babbage called them).

Babbage drew up two complete tables for the calculation. Table 1 shows the code and the order of the seven operations required. The second column shows the operation and the third column is a comment for each calculation. The program ends when the value of x is found.

The six memory addresses $v_1, \dots, v_6$ contain, at the beginning, the coefficients of the six terms $ax, by, c, a'x, b'y, c'$. In the first multiplication, the processor uses the variables v_1 and v_5 for computing $b'a$. The table shows that

Table 1. Computation of *x*.

Mill			Store							
Numbers of	Nature of	in variables	$+ax$	$+by$	$+c$	$+a'x$	$+b'y$	$+c'$		
Operations	operations	variables in store	v_1	v_2	v_3	v_4	v_5	v_6	v_7	
1	×	$b'a$	0				0		$b'a$	$v_7 = v_5 \cdot v_1$
2	×	$b'c$	$b'c$		0					$v'_1 = v_5 \cdot v_3$
3	×	ba'		0	ba'	0				$v'_3 = v_2 \cdot v_4$
4	×	bc'		bc'				0		$v'_2 = v_2 \cdot v_6$
5	–	$bc' - b'c$	0	0		$bc' - b'c$				$v'_4 = v'_2 - v'_1$
6	–	$b'a - ba'$	$b'a - ba'$		0				0	$v''_1 = v_7 - v'_3$
7*	÷	$\frac{bc' - b'c}{b'a - ba'}$	0		$= x$	0				$x = v''_3 = \frac{v'_4}{v''_1}$

after the first operation, both variables are reduced to zero and the result $b'a$ is stored in ν_7. The last column is a comment about the operation which has been performed, that is, $\nu_7 = \nu_5 \cdot \nu_1$.

The second multiplication computes $b'c$ and stores it in ν_1. Symbolically, the computation is $\nu'_1 = \nu_5 \cdot \nu_3$. The quote means that the original content of ν_1 has been overwritten once. However, there is a problem.

Variable ν_5 was read destructively for the multiplication in the first line. The arithmetic operations need two arguments, in this case for the multiplication. Babbage designed the AE so that an argument could be reused repeatedly. Since the first two computed terms are $b'a$ and $b'c$, the first argument b' can be kept in the processor. After the multiplication with a, we only need to load argument c to the processor. This way of reusing an argument in the processor is not described in BAB L1, but it is something that Babbage exploited in other programs. In BAB L1, Babbage explicitly mentions that b' is reused for a multiplication table with b'.

The two columns of comments, displayed side by side, tell the whole story for this computation (Table 2). In a sense, this is the program that Babbage has in mind, but what the punched cards contain are the specific operations and the addresses needed. As can be seen from the code, Babbage reuses memory addresses, and each time a memory address is overwritten he adds a quote to the variable's name. Variables 1 and 3 are reused (overwritten) twice in the program, so that their names become ν''_1 and ν''_3.

Table 2. The seven arithmetic operations and their description, line by line.

	Computation	Code
1	$b'a$	$v_7 = v_5 \cdot v_1$
2	$b'c$	$v'_1 = v_5 \cdot v_3$
3	ba'	$v'_3 = v_2 \cdot v_4$
4	bc'	$v'_2 = v_2 \cdot v_6$
5	$bc' - b'c$	$v'_4 = v'_2 - v'_1$
6	$b'a - ba'$	$v''_1 = v_7 - v'_3$
7	$\frac{bc' - b'c}{b'a - ba'}$	$v''_3 = \frac{v'_4}{v''_1}$

In other programs written after this first one, Babbage simplified. Later, he did not always keep track of variable reuse (with the quotes), since it does not affect the computation. Also, he did not always add symbolic comments to the tables, writing only the necessary arithmetic operation and the addresses used.

The Second Code Table

Babbage wrote the second part of the computation in the same document (L26 in the Babbage archive). Having found x with the first seven lines of the program, we can now compute y using the value of x. The computation for x is the same as before, but y is then computed as $y = (-c - ax)/b$, since the first linear equation is $ax + by + c = 0$. The program is shown below.

There is something new in rows 1, 2, and 3 (See Table 3). Now, Babbage has made explicit that the coefficients a, b, c need to be refreshed, storing their complements Ca, Cb, Cc in auxiliary variables. The complement of a is stored in ν_8, the complement of c in ν_5, and the complement of b in ν_9. We need a, b, and c for the computation of y.

What Babbage intended to do with the AE was to store the value of a variable, that was still needed, in another auxiliary variable when it was sent to the processor. Since the AE used gears (think of a clock face with the digits 0 to 9), storing a number was performed by turning the gear counterclockwise, for example, and retrieving the contents was accomplished by turning it in the opposite direction until the variable was reduced to zero. For example, suppose we stored the number '3' by turning a gear counterclockwise three positions (out of 10 possible positions, one for each decimal digit). When the number is read, the gear turns back three positions, clockwise. Starting from zero, a receiving auxiliary gear will be rotated clockwise to position 7 (the decimal complement of 3). Therefore, the auxiliary variable will not store the original decimal number, but its complement, for each digit of the number. If we had the number 345 in ν_5, and its complement was saved temporarily in ν_8, we would have 765 in the variable ν_8. Reading back from ν_8 to ν_5, we would complement the number again, digit by digit, and ν_5 would be restored to the original 345.

In the program, the stored complements are transferred back (complementing again) to the variables ν_1, ν_2, and ν_3 in the auxiliary steps 8, 9, and 10. The value of y is computed in steps 11, 12, 13, and the program finally stops. There is a mistake in the table in line 7: Babbage writes "=x" in the column for variable 4, which is correct, but also in the column for variable 7, which is incorrect.

In other programs, written after this one, Babbage overlapped the storing of the complement of a variable, with its subsequent reading and complementing in a single program step. That is, the

Table 3. Computation of *x* and *y*.

Mill			Store									
Numbers of	Nature of	in variables	$+ax$	$+by$	$+c$	$+a'x$	$+b'y$	$+c'$				
Operations	operations	variables in store	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	
1	×	$b'a$	0				0	$b'a$		Ca		$v_7 = v_5 \cdot v_1$
2	×	$b'c$	$b'c$		0		Cc					$v'_1 = v_5 \cdot v_3$
3	×	ba'		0	ba'	0					Cb	$v'_3 = v_2 \cdot v_4$
4	×	bc'		bc'				0				$v'_2 = v_2 \cdot v_6$
5	–	$bc' - b'c$	0	0				$bc' - b'c$				$v'_6 = v'_2 - v'_1$
6	–	$b'a - ba'$		$b'a - ba'$	0				0			$v''_2 = v_7 - v'_3$
7*	÷	$\frac{bc' - b'c}{b'a - ba'}$		0		$= x$		0	$= x$			$x = v'_4 = \frac{v'_6}{v''_2}$
8			a							0		$v''_1 = v_1 = a$
9				b							0	$v'''_2 = v_2 = b$
10					c		0					$v''_3 = v_3 = c$
11	×	ax	0			0	ax					$v''_5 = v''_1 \cdot v'_4$
12	–	$-c - ax$	$-c - ax$		0		0					$v'''_1 = -v''_3 - v''_5$
13*	÷	$\frac{-c - ax}{b}$	0	0		$= y$						$y = v'''_5 = \frac{v'''_1}{v'''_2}$

store would send a number to the mill, store it temporarily as a complement, and when the result of the computation was returned by the processor, the stored variable could be refreshed. It is not quite clear whether the refresh happened while the processor was busy or after it had delivered its result to memory. The notation used by Babbage to indicate that a variable containing the value a, for example, kept its value, was $0/a$, indicating that the variable was reduced to zero and later restored, without necessarily indicating the auxiliary address used.

Table 4. Final set of arithmetic operations for computing *x* and *y* and their description, line by line.

	Computation	Code
1	$b'a$	$v_7 = v_5 \cdot v_1$
2	$b'c$	$v'_1 = v_5 \cdot v_3$
3	ba'	$v'_3 = v_2 \cdot v_4$
4	bc'	$v'_2 = v_2 \cdot v_6$
5	$bc' - b'c$	$v'_6 = v'_2 - v'_1$
6	$b'a - ba'$	$v''_2 = v_7 - v'_3$
7*	$\frac{bc' - b'c}{b'a - ba'}$	$v'_4 = \frac{v'_6}{v''_2}$
8		$v''_1 = v_1 = a$
9		$v'''_2 = v_2 = b$
10		$v''_3 = v_3 = c$
11	ax	$v''_5 = v''_1 \cdot v'_4$
12	$-c - ax$	$v'''_1 = -v''_3 - v''_5$
13*	$\frac{-c - ax}{b}$	$v'''_5 = \frac{v'''_1}{v'''_2}$

Symbolically, the complete program written by Babbage would read as Table 4 shows. In steps 8, 9, and 10, there is no operation in the processor and only the memory is active, transferring numbers to recover the parameters a, b, c from their complements.

Conclusion

Solving systems of linear equations is very useful in many areas of mathematics and engineering. It is natural that Babbage decided to use this as a kind of benchmark problem for the AE. It is known that Chinese mathematicians could solve linear systems of up to three variables and equations more than 2,000 years ago.

In his code sketches, Babbage did not write high-level code and then compile the program. The annotations in his program are more like comments and the actual code would be the strings of punched cards for the processor and the memory. In later programs, Babbage did not include a symbolic comment about the computations being performed. Babbage wrote his programs by listing the required operations and the required arguments. Both things immediately translate to the necessary punched cards. In modern parlance, Babbage wrote his programs in "assembler." Also, since the operation cards are separate from the variable cards, they can synchronize in diverse ways, as explained in my paper.[5]

These complications are not present in the program discussed in this article. The AE was never finished, so the code shown here could never be run. It could only be tested following its execution on paper. The design of the AE was very ambitious, including also the possibility of programming loops. Babbage changed the design several times, and this was one of the problems that made it very difficult to finish the machine.[3]

It is important to point out, although it is obvious, that the first sketch ever written of a computer program is not one of those published in Menabrea.[4] That publication appeared six years after Babbage had already sketched his program "number one" for solving simultaneous linear equations and other 25 coding examples. **C**

References

1. Babbage, C. *Passages from the Life of a Philosopher*. Longman, London, (1864).
2. Babbage, C. On the mathematical powers of the calculating engine, unpublished manuscript, 1837. *The Origins of Digital Computers*. B. Randell (eds.), Springer-Verlag, Berlin, (1982).
3. Bromley, A. Charles Babbage's Analytical Engine, 1838. *Annals of the History of Computing 4*, 3 (July 1982).
4. Menabrea, L.F. *Sketch of the Analytical Engine Invented by Charles Babbage, translation with notes from Ada Augusta Lovelace*, (1843).
5. Rojas, R. The computer programs of Charles Babbage. *IEEE Annals of the History of Computing 43*, 1 (2021), 6–18.

Raúl Rojas (rojas@inf.fu-berlin.de) is a professor in the Department of Mathematics and Statistics, University of Nevada Reno, and an emeritus professor at Freie Universitaet Berlin, Germany.