The Airy Tape: An Early Chapter in the History of Debugging

MARTIN CAMPBELL-KELLY

This article describes the discovery of a paper-tape "relic" consisting of an undebugged program written for the EDSAC computer in 1949. It is believed that this program is the first real, nontrivial application ever written for a stored-program computer. An examination of the program sheds new light on the extent to which the debugging problem was unanticipated by early computer programmers, and the motivation for the development at Cambridge of systematic programming practices and debugging aids. The impact of these early developments on programming elsewhere is discussed.

Until the EDSAC ran its first programs in the spring of 1949, little serious study had been made of programming; and all that had been done was for theoretical rather than real machines.

The modern stored-program digital computer was invented by a group that included John von Neumann, J. Presper Eckert, and John W. Mauchly at the Moore School of Electrical Engineering, University of Pennsylvania, during the period from 1944 to 1945; it was first described by von Neumann in his classic "First Draft of a Report on the EDVAC," dated June 1945.[1] The "EDVAC Report" was largely silent on the subject of writing programs. Although in 1945 von Neumann had sketched out at least one program for the EDVAC, this was written mainly to help him check out the suitability of the instruction set. Naturally the program contained errors, and as Knuth has noted, "If von Neumann had had an EDVAC on which to run this program, he would have discovered debugging!"[2]

To disseminate the idea of the stored-program computer as rapidly as possible, the Moore School of Electrical Engineering organized a summer school entitled "Theory and Techniques for Design of Electronic Digital Computers," which took place during July and August 1946.[3] Even though the stored-program computer was no more than a paper design at this stage, some thirty delegates attended the course; these people included representatives from almost all the academic and industrial computing laboratories then in existence. From Britain, Maurice Wilkes was fortunate enough to attend the latter part of the course on behalf of the Cambridge University Mathematical Laboratory.

A number of computer programs were given by the lecturers on the course for expository purposes, but the fact that they contained some obvious errors caused no special comment because, like typographical errors in a mathematical proof, they did not affect the general principles involved. In fact, no evidence has yet emerged that anyone had conceived of the debugging problem until programs were tried on real computers. For example, from 1947 to 1948 von Neumann and Goldstine published their seminal reports "Planning and Coding of Problems for an Electronic Computing Instrument,"[4] and nowhere in the entire 150 pages of these reports is the possibility that a program might not work the first time it was presented to a computer so much as hinted at.

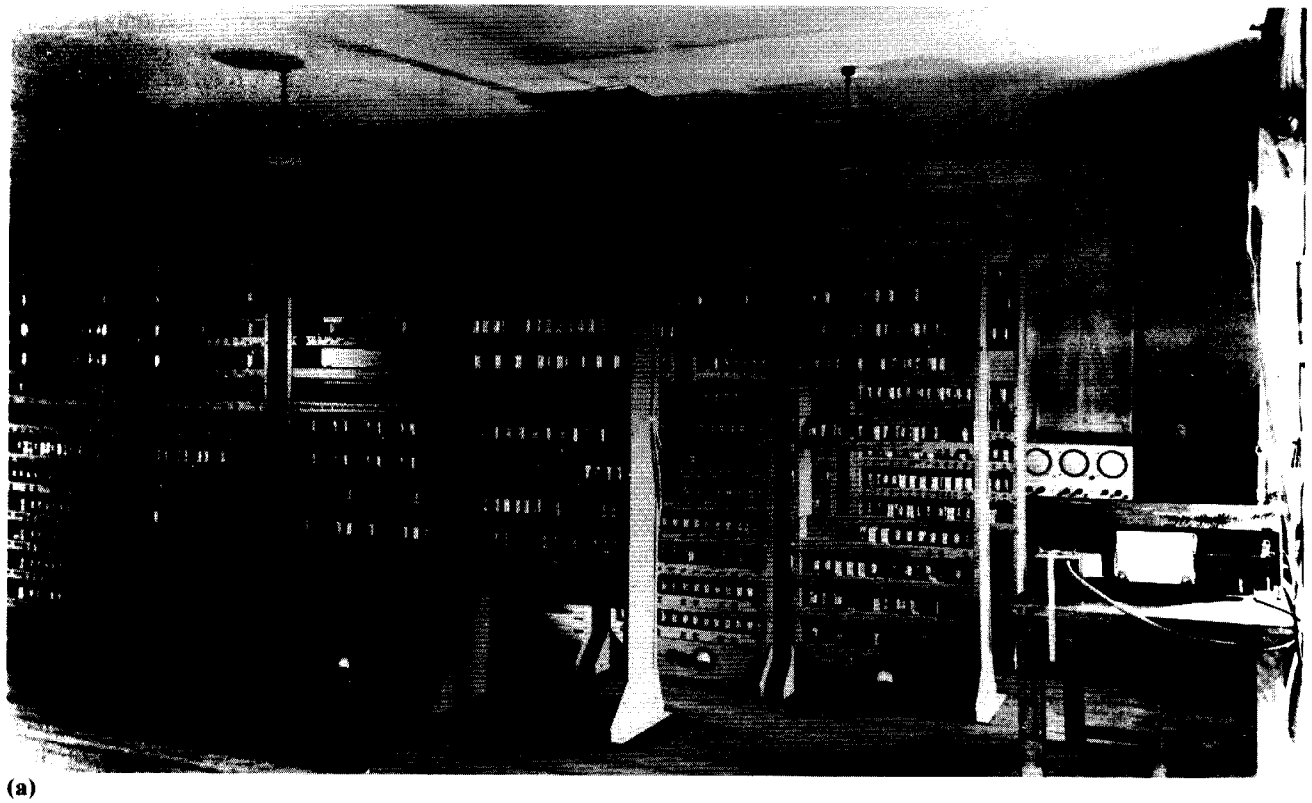
Consequently, there was a surprise in store for the first group that completed a digital computer and attempted to run a nontrivial program on it. This turned out to be the EDSAC computer group, in the summer of 1949.

The EDSAC

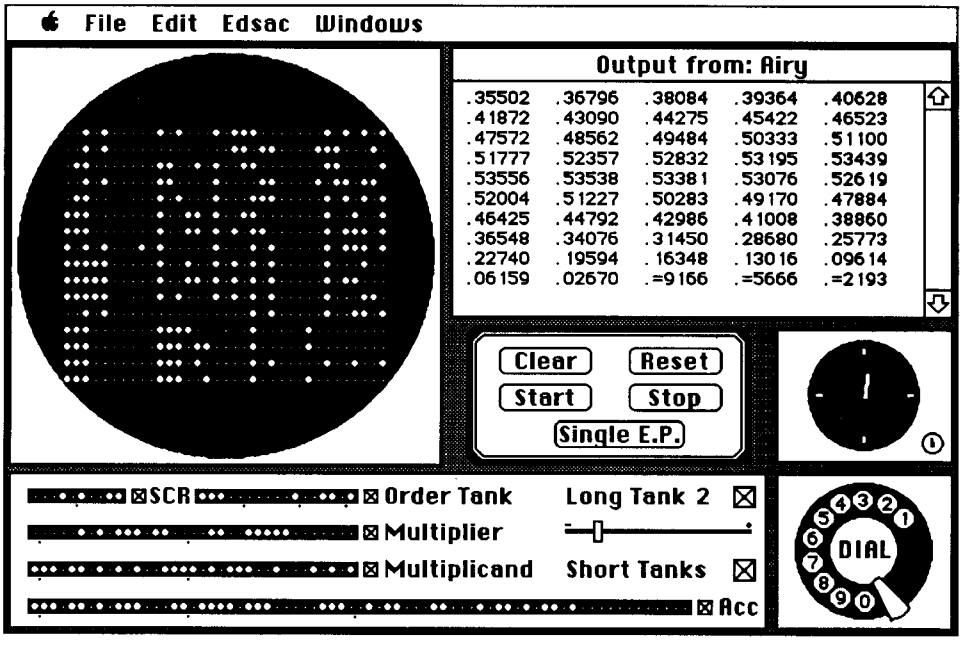
Attending the Moore School Lectures in the summer of 1946 was a turning point for Wilkes: The experience made him realize that the stored-program computer was to be the mainstream of computer development for the foreseeable future. Construction of the EDSAC (Electronic Delay Storage Automatic Calculator) began in early 1947, and it performed its first fully automatic calculation on May 6, 1949. The EDSAC was the first computer of the new type to go into practical operation, and it was a full year ahead of any comparable development in the United States.

The early completion of the EDSAC was in large part due to Wilkes' conservative approach to design. For example, he opted for a pulse rate of 0.5 MHz when "any electronic engineer worth his salt"[5] (p. 129) would have accepted the challenge of working at 1 MHz. The reason for this attitude was that Wilkes was motivated not only by the engineering challenge but also by "the desire to get a machine on which we could try out programmes, instead of just dreaming them up for an imaginary machine."[6] Wilkes was convinced that time would of its own accord provide faster and better technology; meanwhile, he wanted to get down to the business of programming.

1058-6180/92/1000-0016$03.00 © 1992 IEEE



(a)



(b)

Figure 1. The EDSAC, then and now. (a) A photograph of the EDSAC taken shortly after its completion in May 1949. The left three quarters of the picture show the main racks of the arithmetic unit, control, and memory. The input/output equipment (a paper-tape reader and teleprinter) can be seen on the table at the right. Three of the monitor tubes can be seen to the rear and right of the picture. (b) An EDSAC simulator developed for the Macintosh computer. The circular display (top left) shows the main memory monitor tube. The register panel (bottom left) displays the accumulator and other registers. The clock (middle right) gives the elapsed EDSAC time. The output panel (top right) shows the teleprinter printout. The central panel contains the five user-accessible controls: Clear, Start, Reset, Stop and Single E.P. The dial (bottom right), added in 1951, enabled a single decimal digit to be entered into the machine.

Technology and architecture

Before we discuss programming for the EDSAC, however, we need to know something of its technology and architecture. Figure 1a is a photograph of the EDSAC taken shortly after its completion in May 1949. Such has been the rate of progress in computing in the last decade that it is now possible to simulate the EDSAC on any modest 16-bit personal computer. Figure 1b shows an EDSAC "workstation" for the Macintosh computer. The functional descrip-

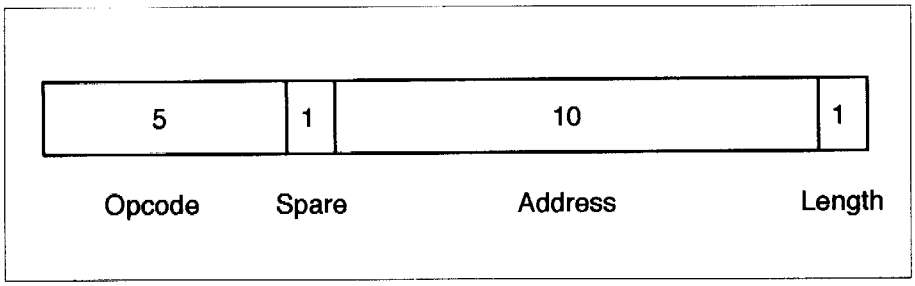


Figure 2. Format of an EDSAC instruction.

tion of the EDSAC given here applies equally to the original machine or to the simulator.*

Like all first-generation machines, the EDSAC processor was constructed using vacuum tubes, or thermionic valves as they were known as in Britain. Altogether the machine contained some 3,000 vacuum tubes and consumed about 12 kW of power. Pulse electronics had matured remarkably during the war years, so the processor itself was an engineering challenge mainly on account of its scale. A much more difficult engineering problem was the development of a suitable memory system. At the time that Wilkes began building the EDSAC, there was no proven storage technology, although during the war a number of centers, including the Moore School and some of the British radar-research laboratories, had experimented with mercury delay lines or cathode-ray tubes for radar echo cancellation. Both showed some promise as possible storage devices. There were also some special-purpose storage tubes under development, such as the RCA Selectron, but these were a long way from being practical propositions.

However, by a piece of good fortune, in early 1946 Wilkes had made the acquaintance of a research student in the Cavendish Laboratory at Cambridge, Tommy Gold, who had worked with mercury delay lines during his war service with the Admiralty Signals Establishment. As soon as Wilkes' plans for the EDSAC hardened, he sought the advice of Gold, who was able to sketch out the design of a delay line that he thought would be suitable. In line with his general policy of minimizing the engineering effort, Wilkes made use of this design for the EDSAC, essentially without modification.

Each of the mercury delay lines used in the main memory was about five feet in length and could store 576 ultrasonic binary pulses (the equivalent of 32 18-bit words). The contents of a delay line were constantly recirculated with a cycle time of approximately 1 ms; this determined the basic speed of the machine — which averaged about 600 operations per second. The EDSAC was designed to have a total of 32 main-memory delay lines or "tanks" organized in two batteries of 16 delay lines each. But for the first year or so of EDSAC's existence, only one battery of 16 delay lines was functional — giving a total of 512 words of main memory, the equivalent of a little over 1 Kbyte.

A useful feature of the delay-line technology was that it was possible to display the contents of a storage tube on a CRT monitor. The EDSAC monitors can be seen toward the rear and right of Figure 1a. One of the monitors allowed the contents of any one of the 16 main-memory delay lines to be observed. In Figure 1b, the large circular screen in the upper left shows the main-memory monitor tube, in a style which is a fairly close emulation of the original. Delay lines were also used for the central registers and accumulator, but since these were each only one or two word-lengths, much shorter delay lines of only a few inches were used. These were known as "short tanks" to distinguish them from the "long tanks" used in the main memory. In Figure 1b, the short tank displays are shown, in a slightly stylized form, at the bottom left.

Input to the EDSAC was by means of paper tape, and output was by means of a modified Creed teleprinter of the kind used by the British Post Office. This equipment can be seen on the table at the right of Figure 1a; the upper-right window of Figure 1b shows the printout generated by the simulator.

Finally, the EDSAC was controlled by five push buttons: Start, Stop, Clear, Reset, and Single E.P. Their purpose is self-evident except for the last. The Single E.P. button caused a single instruction to be obeyed, which enabled a program to be executed one instruction at a time.

The EDSAC used a single-address instruction format, shown in Figure 2. Although the EDSAC was based on an 18-bit word, only 17 bits were used, the leading bit being unusable for reasons connected with circuit setup time. The opcode (or "function") was specified in 5 bits and the address in 10 bits. A further bit specified the operand length: Most instructions could operate on either a 17-bit short word or a 35-bit double-length word; the length indicator specified which.

Table 1 shows the EDSAC instruction set as it existed in 1949. Operations were represented by letters of the alphabet, some of which suggested the function they denoted (for example, A for Add, S for Subtract, and so on). Average instruction times were 1.5 ms, although multiplication was longer and took 6 ms; input/output times were determined by the basic speeds of the peripheral equipment (the teleprinter, for example, printed at 6⅔ characters per second).

The repertoire of instructions was, of course, exceedingly sparse by later standards, and even by the standards of the time; but this was entirely in line with the EDSAC design philosophy of simplifying engineering, even if this meant lengthening programming.

Programming

Programming was an issue that was very much taken for granted by the majority of the early computer projects. For

* To provide an authentic evocation of the EDSAC environment, the simulator has been designed with exactly the same functionality and controls as the original machine; its only advantage is its greater speed and convenience, and the fact that the hardware reliability is many orders of magnitude greater. This article is in large part based on the insight on the EDSAC gained using the simulator. A description of the simulator will be published elsewhere; readers wishing to obtain a copy of the system should apply to the author.

them, all the emphasis was on the hardware, so that it was only when a machine sprang into life that the business of programming was seriously considered at all. Consequently, users of many early machines were obliged to program in pure hexadecimal or some variant. (It is interesting that this mistake was repeated in some microprocessor development systems in the 1970s.) Some groups, such as that of von Neumann and Goldstine at the IAS, Princeton, certainly came up with good programming notations, but they always assumed that a human coder would manually transform the symbolic notation into the binary machine code of the computer.

Location	Order			Notes
100	T		L	Clear accumulator using location 0L as a "rubbish bin"
101	A	8	L	Add contents of location 8L into accumulator
102	E	105	S	Jump to location 105 if accumulator positive
103	S	8	L	Subtract location 8L from accumulator
104	S	8	L	Subtract location 8L from accumulator
105	T	8	L	Store accumulator in 8L leaving accumulator clear

Figure 3. Example of EDSAC code.

Table 1. The EDSAC instruction set (1949).

A *n*	Add the number in storage location *n* into the accumulator.
S *n*	Subtract the number in storage location *n* from the accumulator.
H *n*	Copy the number in storage location *n* into the multiplier register.
V *n*	Multiply the number in storage location *n* by the number in the multiplier register and add the product into the accumulator.
N *n*	Multiply the number in storage location *n* by the number in the multiplier register and subtract the product from the accumulator.
T *n*	Transfer the contents of the accumulator to storage location *n* and clear the accumulator.
U *n*	Transfer the contents of the accumulator to storage location *n* and do not clear the accumulator.
C *n*	Collate [logical *and*] the number in storage location *n* with the number in the multiplier register and add the result into the accumulator.
R 2^{n-2}	Shift the number in the accumulator *n* places to the right.
L 2^{n-2}	Shift the number in the accumulator *n* places to the left.
E *n*	If the sign of the accumulator is positive, jump to location *n*; otherwise proceed serially.
G *n*	If the sign of the accumulator is negative, jump to location *n*; otherwise proceed serially.
I *n*	Read the next character from paper tape, and store it as the least significant 5 bits of location *n*.
O *n*	Print the character represented by the most significant 5 bits of storage location *n*.
F *n*	Read the last character output for verification.
X	No operation.
Y	Round the number in the accumulator to 34 bits.
Z	Stop the machine and ring the warning bell.

Note: The EDSAC order code was modified from time to time. For example, the X order was originally used to round off short numbers, but was changed to a no-operation instruction when it was found to serve no useful purpose. Similarly the F-check instruction was eventually removed in favor of a conventional parity checking scheme.

At Cambridge, however, the emphasis was entirely the other way. From the beginning, Wilkes decided that programs would be written in a human-oriented notation, and moreover that the conversion from the external symbolic form to the internal binary form would be done by the machine itself. There were thus two aspects to the programming problem: first, the form of symbolic instructions, and second the mechanism by which programs would be loaded into the memory.

Figure 3 shows a program fragment for the EDSAC as it would have been written in 1949. The code replaces the long number in location 8 (written 8L) by its absolute value; instructions are assumed to occupy address locations 100 onwards. This notation looks strikingly similar to an assembly language for a much later period, and compared with the programming notations used on some other contemporary machines, it was remarkably sophisticated. It should be noted, however, that the comments were not punched; only the symbols between the vertical rules would have appeared on the paper tape. Thus the fragment of program in Figure 3 would have been punched: TLA8LE105SS8LS8LT8L. This arrangement meant that program tapes were physically very short.

The second problem was translating the program into binary and loading it into the machine. This was one of the first tasks entrusted by Wilkes to a research student, David Wheeler (1927-), who joined the laboratory in October 1948. Wheeler devised a simple loading routine known as the "initial orders" that worked as follows. When the Start button of the EDSAC was pressed, the initial orders were automatically placed in the first memory tank (words 0-31). The input routine then took control of the machine and proceeded to load the symbolic program punched on paper tape into words 32 onwards — this involved a simple binary-to-decimal conversion of the address, and some other small transformations. When the entire program had been loaded, it was entered at word 32.

The initial orders were used from the earliest days of the machine, including during its commissioning. On May 6, 1949, a short test program, which printed a table of squares, was run; it worked and the EDSAC logbook was duly inaugurated. The printout from this historic program is now preserved in the Science Museum at Kensington. During the next two or three days, two more simple demonstration

The Airy Tape

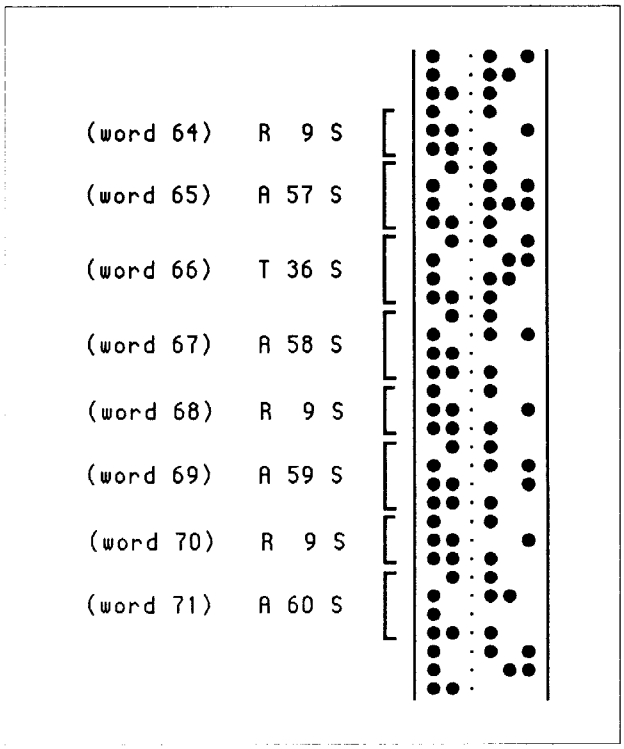


Figure 4. A section of the Airy tape, corresponding to instructions 64 to 71 of the program in the Appendix. The binary form can be seen as the bottom four rows of the main memory monitor tube in Figure 1b. The complete tape is a little over four feet long.

programs were successfully run. The first, due to Wheeler, printed prime numbers, and the second, written by Wilkes, printed a table of squares and their differences. Both of these programs were straightforward. Having been honed to perfection during the weeks prior to the completion of EDSAC, they were entirely error free; the only difficulty encountered in making them run was reluctant hardware. These early successes could have done nothing to prepare programmers for the difficulties of debugging that lay ahead.

The Airy program

Shortly after the EDSAC began operating, Wilkes organized a conference on "High Speed Automatic Calculating Machines," which took place June 22-25, 1949.[7] This was the first meeting of its kind to be held in Europe, and delegates from several countries attended. At the conference, visitors were given a tour of the EDSAC, and the demonstration programs for primes and squares were run. After the excitement of the June conference had subsided, Wilkes began to work seriously on the first real application program for the EDSAC.

This program was designed to calculate Airy's integral, which is the solution to the second-order differential equation

$$y'' = xy$$

There were a number of reasons for choosing this particular problem. Foremost, it was done as an exercise in using a computer to solve a differential equation, which was expected to be one of the main applications of the EDSAC. Naturally, Wilkes started with a simple example, and Airy's equation represented the next step in complexity from the equation for simple harmonic motion. Moreover, he was very familiar with the Airy equation, which he had encountered while still a student in connection with the propagation of radio waves in the ionosphere.

Although published tables of the Airy integral had existed since the time of Airy and Stokes in the 1850s, no completely satisfactory set of tables was available until J.C.P. Miller's *The Airy Integral* was published under the auspices of the British Association Mathematical Tables Committee in 1946.[8] These tables were — and remain — effectively the last word on the subject. Wilkes realized that the existence of Miller's table would provide a very useful check on the validity of his program.

Hence, the Airy program was not intended to result in a useful product as such, but was really a test program to explore both the capabilities of the EDSAC and the business of writing a real program. This would be a program that not only explored the problems of program logic and branching, but also the arithmetic processes intrinsic to numerical computation. The Airy program is almost certainly the first real application program ever written for a stored-program computer, and it may fairly be said to have ushered in the then-undreamt-of field of programming and programming systems.

The original program manuscript has long since disappeared because Wilkes, for one, attached no special significance to it. In fact, only a handful of programs have survived from the very early days of digital computers (say 1949 to 1951), and those that have survived tend to be ones that were polished with publication, not to say posterity, in mind.

In the summer of 1979, shortly before his retirement from the Computer Laboratory, Wilkes was clearing out his drawers and cupboards, and discovered in an obscure corner a paper tape in a rather fragile condition with the single word "Airy" written on it in pencil. Although undated, Wilkes was able to date the tape with some accuracy because on July 27, 1949, he had begun to keep a "Record of Program Tapes." Since the tape bore no serial number, it was evidently punched shortly before that. Wilkes sent me a photocopy of the tape to see what I could make of it. Figure 4 shows a fragment of the paper tape. I sensed that I was being set a challenge — one that took me some years to resolve to my satisfaction.

A reconstructed and partially corrected transcription of the Airy program is given in the Appendix on pages 25-26. Reconstituting the raw code from the paper tape was essentially a clerical task, albeit a tedious one. This enabled the sequence of symbolic instructions shown between the vertical rules to be drawn up. The comments to the left and the right of the rules were not, of course, present on the tape. What had been achieved at this stage was analogous to regenerating code from a memory dump: not difficult to do,

but the result is essentially meaningless without a specification of the original program.

However, by a stroke of luck, Wilkes had published a description of the Airy program in *Nature* in October 1949, together with a specimen of the output produced by the program (Figure 5). Wilkes chose to publish in *Nature* because it was — and still remains — *the* international journal for the rapid dissemination of new scientific results. The *Nature* article is almost certainly the first ever published account of a real numerical computation on a stored-program computer.

The method that Wilkes chose to evaluate the Airy integral was one that had been made popular for use on desk machines by Douglas Hartree, then Plummer Professor of Physics at Cambridge. Hartree was at that time the éminence grise of numerical analysts in Britain, and he had proved a constant source of encouragement to Wilkes.

Hartree's method for the integration of Airy's equation

$$y'' = xy$$

was based on the well-known central difference formula

$$\delta^2 y = (\delta x)^2 (y'' + \frac{1}{12}\delta^2 y'')$$

where δx is the interval of the argument. Given three adjacent values of y, namely y_0, y_1, and y_2, and eliminating y'', we get

$$y_2 = 2y_1 - y_0 - \frac{1}{12}(\delta x)^2 (x_0 y_0 + 10 x_1 y_1 + x_2 y_2)$$

If y_0 and y_1 are known, y_2 can be evaluated iteratively. This highly reliable and popular process was later called the "Royal Road" procedure by the astronomer Zdenek Kopal.

Wilkes' algorithm, given in the *Nature* article, is worth quoting in full, because, apart from the examples in the von Neumann and Goldstine reports, it is probably the first published example of a numerical procedure that corresponds closely to the modern notion of a computational algorithm:

> In order to set out in further detail the operations to be performed by the machine, it will be assumed that the quantities y_0 and y_1 are held in the store of the machine in "storage locations" numbered 100 and 101 respectively, and that storage location 102 contains a number η. η will change as the calculation proceeds, and will finally become equal to y_2; initially $\eta = y_1$. The various stages of the calculation are then as follows:
>
> (1) Evaluate $\eta' = 2y_1 - y_0 - \frac{1}{12}(\delta x)^2 (x_0 y_0 + 10 x_1 y_1 + x_2 y_2)$.
>
> (2) Examine the sign of $|\eta' - \eta| - \varepsilon$, where ε is a small quantity specified in advance. If the sign is positive, replace η in storage location 102 by η' and repeat (1). If the sign is negative, proceed to (3).
>
> (3) Print y_0.

x	$A_i(-x)$				
0·00	+.35503	+.36796	+.38085	+.39364	+.40628
	+.41872	+.43090	+.44276	+.45423	+.46524
	+.47573	+.48562	+.49485	+.50333	+.51100
	+.51777	+.52357	+.52833	+.53196	+.53439
1·00	+.53556	+.53539	+.53381	+.53077	+.52619
	+.52005	+.51227	+.50283	+.49170	+.47885
	+.46426	+.44793	+.42986	+.41008	+.38861
	+.36548	+.34076	+.31451	+.28680	+.25773
2·00	+.22741	+.19599	+.16348	+.13016	+.09615
	+.06160	+.02671	-.00834	-.04333	-.07807
	-.11233	-.14588	-.17850	-.20997	-.24004
	-.26849	-.29510	-.31964	-.34191	-.36169
3·00	-.37881	-.39310	-.40438	-.41254	-.41744
	-.41901	-.41718	-.41191	-.40319	-.39105
	-.37553	-.35674	-.33478	-.30981	-.28201
	-.25151	-.21886	-.18402	-.14742	-.10938
4·00	-.07027	-.03045	+.00968	+.04970	+.08921
	+.12778	+.16500	+.20044	+.23370	+.26440
	+.29215	+.31562	+.33750	+.35449	+.36737
	+.37593	+.38004	+.37959	+.37454	+.36490

Figure 5. The Airy table. The output from the final version of the Airy program, as reproduced in facsimile in *Nature*, October 1, 1949. The numerical values were printed directly by the EDSAC teleprinter, but Wilkes wrote the headings in by hand to avoid complicating the program.

> (4) Replace y_0 in storage location 100 by y_1 from storage location 101, and y_1 in storage location 101 by y_2 from storage location 102 (η remains in storage location 102). Repeat (1).[9]

The locations of 100, 101, and 102 were, of course, chosen arbitrarily for the purposes of exposition.

It is interesting to contrast this algorithm with an extract from the Royal Road procedure given by Hartree:

> The procedure is then as follows. Estimate $\delta^2 y''_0$, and obtain an approximation to $\delta^2 y_0$ from (7.3). Add this to $\delta y_{-1/2}$ to give an approximation to $\delta y_{1/2}$, and add this to y_0 to give an approximation to y_1. From this calculate y''_1 and hence $\delta^2 y''_0 = y''_1 - 2y''_0 + y''_{-1}$. Let ε be the difference between this value of $\delta^2 y''_0$ and that estimated. A change of the estimate of $\delta^2 y''_0$ by ε makes a change $\frac{1}{12}(\delta x)^2 \varepsilon$ in y_1. If this is less than $\frac{1}{2}$ in the last figure retained in y, the estimate is adequate; if not, the estimate is revised and the calculation of the interval repeated; but the interval length (δx) should be taken so that this is seldom necessary.[10]

This was then followed by a worked example.

Ignoring the notational differences, the important difference between these two descriptions is that Wilkes' was designed for an electronic digital computer, whereas Hartree's was aimed at a human computer who could be assumed to have a high level of reasoning power. These two descriptions encapsulate the transition from Hartree's prewar world of the "science and art of numerical calculation" to the postwar world of computational algorithms.

The Airy Tape

The Airy paper tape proved to conform very closely to the algorithm given by Wilkes, and this enabled the purpose of each instruction and constant in the program to be understood. The results are indicated by the comments to the Airy program in the Appendix.

One detail that Wilkes did not include in his account needs to be explained before the Airy program can be fully understood. This is the problem of scaling. Like other machines of its era, the EDSAC did not have floating-point hardware, and it could only handle fractions, α, in the range $-1 \leq \alpha \leq 1$. Consequently numbers larger than unity (or very small numbers) had to be multiplied by a scale factor. In this particular problem, the values of the Airy integral, $Ai(x)$, were fortuitously close to, but less than, unity in absolute value, and needed no scale factor at all. The x values, however, which could be up to 100 in modulus, were multiplied by $2^{-11} \cdot 10$ (which was a convenient scale factor in a binary machine). Elsewhere in the program, a complementary scale factor of $2^{11} \cdot 10^{-1}$ was introduced so that the results were printed correctly. Scaling was one of the major headaches in preparing programs on early computers, but by the late 1950s innovations such as floating-point hardware and interpretive systems meant that most users no longer had to be concerned with it.

The discovery of debugging

The most striking feature of the Airy program is the large number of errors it contains. These errors are listed in Table 2 and corrections have been handwritten in the program in the Appendix. (Where additional words have been inserted, instructions would need to be renumbered and the address fields of some instructions changed; this is essentially a clerical task which has not been done in the interests of clarity.) Altogether there were approximately twenty errors in the 126 lines of the program. By any standard this is a large number of errors in a relatively short program. However, Wilkes is one of the giants of computing, so we can assume that there was more to this than meets the eye.

Some of the errors are obvious by inspection, while others are more subtle; and one error, discussed shortly, is very subtle indeed. The most glaring of the obvious errors occur in lines 62, 64, 68, and 70, where, to obtain a right-shift of 9 places the order R 9 S is used. In fact, to simplify the hardware implementation of the shift order, to get an n-place shift, the nth bit of the instruction word had to be set to a 1, and the rest of the bits to 0. Hence the instruction should have been R 128 S. People often made this mistake when first coding for the EDSAC, but it seems unlikely that the designer of the machine could have fallen into the same trap. A more likely explanation is that Wilkes used R 9 S as a convenient but temporary notation while writing the program, and forgot to change it before giving the manuscript to his secretary to keypunch. The result was to make the results of the program meaningless, and this suggests that this version of the Airy program is quite possibly a first draft — and maybe that was why Wilkes, with a subconscious sense of history, put it to one side all those years ago.

There are also a number of errors that appear to be plain punching mistakes. For example, in order 135 a U-order is mispunched as a V-order, probably due to misreading the manuscript. Since it was possible to list program tapes only with some difficulty and inconvenience, this error would have been difficult to detect, but its existence renders the results of the program nonsense.

Elsewhere, there are several instructions missing altogether: For example, between orders 102 and 103, an order (T 35 S) is needed to store the result of a computation; and between orders 143-144 and 148-149 extra instructions are needed to count the correct number of times round a loop. In other places, long and short operands are confused. Any one of these errors would have invalidated the results of the whole computation.

From the foregoing, it seems likely that the program was not desk checked before it was punched and was not verified after it was punched. This might seem surprising, given that Wilkes had himself done some computing with a desk machine, where procedures tended to be very well organized and systematic. However, in a manual computation the aim was to minimize the possibility of computational errors. A reliable digital computer would be free from computational errors of the kind that a careless desk machine operator might introduce. On the other hand, the logic of a manual computation never had to be spelled out in detail because it could be assumed that a human being would do the sensible thing; it is only obvious in hindsight that the same is not true for a computer program.

Wilkes recalls his discovery of the debugging problem in his *Memoirs*:[5]

> By June 1949 people had begun to realize that it was not so easy to get a program right as had at one time appeared. I well remember when this realization first came on me with full force. The EDSAC was on the top floor of the building and the tape-punching and editing equipment one floor below on a gallery that ran round the room in which the differential analyser was installed. I was trying to get working my first non-trivial program, which was one for the numerical integration of Airy's differential equation. It was on one of my journeys between the EDSAC room and the punching equipment that "hesitating at the angles of stairs" the realization came over me with full force that a good part of the remainder of my life was going to be spent in finding errors in my own programs (p. 145).

How then was the program actually debugged? There were, of course, no software debugging aids whatever on the EDSAC at this time, so the program had to be debugged on a naked machine, by "single-stepping" through the program and observing the contents of the memory and registers on the monitor tubes. This process was known by the rather charming name of "peeping." On some machines it was possible to correct errors in the program using hand switches, although this was not possible on the EDSAC. Hence every debugging run would have necessitated a trip from the punching room to the EDSAC and back again to correct the program tape. Wilkes must have trod the stair-

case between the punching room and the EDSAC many times to get the Airy program working.

Using the EDSAC simulator, it has proved possible to produce some results from the partially corrected version of the Airy program. The output is shown in the upper right-hand panel of Figure 1b. The printout shows some slight departures from the results printed in *Nature*. Most obviously, the program fails to cope with the printing of negative numbers, so that from midway in the tenth line the printout becomes incoherent. It would take just half a dozen instructions to put this right. Quite possibly this was a detail that Wilkes chose to omit until the program was more or less running. Similarly, extra instructions are needed to round off the results to the fifth decimal place.

Before we leave the Airy program, the very subtle error alluded to earlier needs to be explained. When Wilkes first sent me the tape in 1979, it did not take very long to coax some numbers from the program, but although the results were correct to four decimal places, there was an error of as much as four units in the fifth place. This initially made me suspect that the algorithm might be at fault or that the step length was too great. So, in an exercise that I felt was really rather contrary to the spirit in which I wanted to debug the program, I transliterated the program into Fortran, but the results it produced were in accordance both with the table in *Nature* and with Miller's *The Airy Integral*. Since everything else in the program looked perfect — and since I had spent more time trying to debug the program than I really care to admit — I was forced to concede defeat and put it to one side. During the intervening years between then and now I looked at the program again two or three times but the bug remained obstinately hidden. Finally, one morning in early 1990, the penny finally dropped: The error was caused by the fact that the constant $(\delta x)^2/12$ in location 45 was stored only to single precision instead of double precision. With this correction made, the program is capable of producing results correct to the eight figures given in Miller's table. Of course, it has to be noted that Wilkes himself spotted the error somewhat more quickly in 1949. Although he would have been helped both by being closer to the problem and by being an experienced numerical analyst, I do not think it would have been in the least an easy error for him to detect. It was often noted in the 1950s that numerical errors were more stubborn and harder to correct than errors of program logic; the Airy program confirms that observation.

Getting programs right

The early experience of correcting the Airy program, and many subsequent programs, shaped the Cambridge philosophy toward debugging. First and foremost, the use of peeping to debug programs was found to be very extravagant of machine time and Wilkes, with the strong support of Wheeler, outlawed the practice as soon as it was possible to do so. Instead various software aids for error detection were provided. The first of these aids were "postmortem" dump routines that printed out the contents of the memory when a program had terminated abnormally or had been aborted. The postmortem program was loaded by the initial orders in the usual way, and placed in the high end of memory, where it was least likely to overwrite the user program. It would then print out the contents of a selected region of memory in a selected format. Routines were provided to print out words in the form of single-length or double-length fractions, integers, or instructions, whichever the programmer found most useful. The postmortem printout could then be studied at leisure away from the machine. The postmortem technique was so successful that by about mid-1950 it proved possible to run the EDSAC as a closed shop during the daytime with a professional operator; users would specify the postmortem procedure to be adopted by the operator should the program terminate abnormally, print spurious results, or go into an infinite loop.

Table 2. Errors in the Airy program.

Location	Error
62, 64, 68, 70	Incorrect left-shift order
72, 82	Incorrectly specified operand length
45, 50L, 53, 57	Incorrect constants
73, 144	Redundant instructions or constants
102-3, 143-4, 148-9	Missing orders
130, 141	Incorrect address in instruction
135, 149, 152	Incorrect or mispunched opcode

The real breakthrough in debugging techniques, however, was made in early 1950 by Stanley Gill (1926-1975), then a research student who joined the laboratory in October 1949. Gill invented the concept of the interpretive trace routine.[11] In Gill's scheme, instead of the user's program being obeyed directly by the control circuits of the computer, it was interpreted instruction-by-instruction by the interpreting routine. As each instruction was obeyed, diagnostic information could be printed. One trace routine, for example, printed the order-code letter of each instruction as it was obeyed. (If this routine could have been applied to the Airy program, it would have produced the output EARARATARARATTHVH etc.) This helped enormously in the detection of stubborn logic errors. Another trace routine printed the value of the accumulator whenever its contents were transferred to memory; this was particularly useful in detecting stubborn numerical errors. (For a fuller discussion of debugging routines, see David Wheeler's article "The EDSAC Programming Systems" in this issue of the *Annals of the History of Computing*.)

The Airy program also hints at another important aspect of reducing the incidence of program errors — by the use of subroutines. It was understood by all the early groups that subroutines would be an important aspect of programming. For example, in the Goldstine and von Neumann "Planning and Coding"[4] reports the authors stated

> ...we envisage that a properly organized automatic, high speed establishment will include an extensive collection of such subroutines, of lengths ranging from about 15-20 words upwards. That is, a "library"

> of records in the form of the external memory medium, presumably magnetic wire or tape (p. 288).

The advantage of the subroutine library, of course, was that common functions such as square root, input-output, and so on, could be written once and for all, and used by everybody.

Cambridge, however, discovered two further advantages conferred by the use of subroutines: first, the effect on program structure, and second the effect on program bugs.

The effect on program structure is shown clearly in Wilkes' algorithm for the Airy program (quoted earlier), which is notable for the absence of a flowchart of the kind being advocated by Goldstine and von Neumann. Possibly Cambridge was the only early computer group that did not go through a phase of using flowcharts (and consequently did not have to unlearn them when the vogue for structured programming came in). The use of subroutines in structuring programs helped programmers to think more clearly, and hence make fewer errors in program logic. In Wilkes' algorithm the coding step

> (3) Print y_0

is nothing less than a subroutine call, and the code itself appears in the lines 112-157. As a subroutine, however, it clearly has some major shortcomings. The first is that there is no proper linkage between it and the main program. And second, the subroutine cannot be relocated — that is, it operates correctly in 112-157, but would not work anywhere else.

These issues were shortly to be resolved in a classic piece of work by David Wheeler, in which he devised both a proper technique for subroutine linkage and a new set of initial orders that could be used to relocate library subroutines so that they would work anywhere in the memory.[12] The new initial orders were incorporated in the EDSAC in September 1949, and the whole subject of programming-systems development then moved into high gear.

The second unanticipated benefit of using subroutines was that it was possible to copy library tapes mechanically into a user's program; reducing the number of instructions that the user wrote or punched directly vastly reduced the incidence of program errors. In a typical application program, two thirds or more of the total number of instructions in a program would come from the library and only one third would be written by the programmer. As Wilkes noted, "this in itself would be a sufficient reason for having a library, quite apart from any other considerations."[13]

Spreading the word

By the summer of 1950 the EDSAC programming system had reached a high state of refinement, and the subroutine library contained some 70 subroutines. The coding style was so user friendly, as we would now call it, that several users from outside the computer laboratory had begun to write their own programs.

In September 1950 all the programming techniques were written up and compiled into a spirit-duplicated report entitled "Report on the Preparation of Programs for the EDSAC, and the Use of the Library of Subroutines." Wilkes sent copies of this report all over the world to people he thought would be interested. It must be recalled that, at this date, the EDSAC was still the only fully operational stored-program computer in the world, so that the report aroused a great deal of interest, especially in the United States, and it may be fairly said to have laid the foundations of the whole subject of programming systems. In 1951 the EDSAC report was published as a textbook by Addison-Wesley in Cambridge, Mass., as *The Preparation of Programs for an Electronic Digital Computer*, under the joint authorship of Wilkes, Wheeler, and Gill.[13] This was the first classic textbook on programming; it was usually known to the first generation of programmers as simply "Wilkes, Wheeler, and Gill" (often abbreviated to "WWG").

Thus the Cambridge programming methodology had a heavy influence on the programming systems of many of the first-generation computers that came into being during the early 1950s. For example, at MIT the programming system developed by Charles Adams for the Whirlwind computer wholeheartedly embraced the Cambridge model. The programming system for the ILLIAC at the University of Illinois was developed by Wheeler during his period there as an assistant professor from 1951 to 1953. In its turn, ILLIAC influenced several other computers in the United States, and in more distant countries such as Israel (the WEIZAC), Australia (the SILLIAC), and Japan (the Musasino-1). Programmers at IBM were also receptive to some of the ideas, which in due course found their way into the programming methods for the Model 701.[14] And at Univac, Grace Hopper acknowledged "from Dr. Wilkes, the greatest help of all, a book on the subject."[15]

A key chapter both of the original EDSAC report and of Wilkes, Wheeler, and Gill's book was one devoted to the subject of getting programs right. The chapter was entitled "Pitfalls" — the term debugging did not come into general use in this context until the late 1950s. In this chapter, which was written by Wilkes, readers were warned that programmers would sometimes make mistakes:

> Experience has shown that such mistakes are much more difficult to avoid than might be expected. It is, in fact, rare for a program to work correctly the first time it is tried, and often several attempts must be made before all errors are eliminated. Since much machine time can be lost in this way a major preoccupation of the EDSAC group at the present time is the development of techniques for avoiding errors, detecting them before the tape is put on the machine, and locating any which remain undetected with a minimum expenditure of machine time.[13]

Wilkes took particular pains to disabuse new computer users that program debugging using the console and CRT monitors was anything other than "a very slow and inefficient process, especially as the numbers are usually displayed in binary." The chapter then went on to describe the dump and trace procedures employed on the EDSAC.

These ideas soon permeated the whole culture of early programming, although many people who devised such procedures for the computers that were then coming into service probably did not know from where the ideas had originally sprung.

But the EDSAC programming methodology was as strong on prevention as it was on cure. Thus programmers were urged to desk check the program before submitting it to the computer, to code in a logical manner, and "not hesitate to copy it out in a more logical layout whenever necessary." And a comprehensive list was given of "points to be checked" before submitting a program to the computer.

It would be nice to think that early programmers heeded this advice, but it seems unlikely — like Wilkes, no doubt most of them learned the hard way. ■

Acknowledgments

It is a pleasure to acknowledge the help of Maurice Wilkes, who initially set me the Airy challenge, and subsequently set right some of my interpretations. I am also grateful to Don Hunter, himself an EDSAC pioneer, who made an independent check of the Airy program on his EDSAC simulator. Alan Bromley and Steve Russ provided further illumination on some of the ideas in the article.

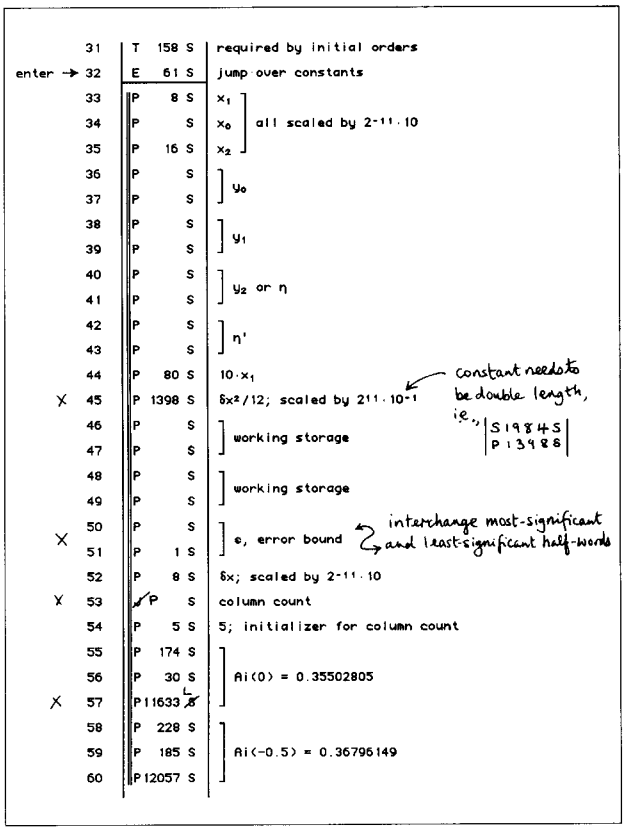
References

1. J. von Neumann, "First Draft of a Report on the EDVAC," Moore School of Electrical Engineering, Univ. of Pennsylvania, June 30, 1945; reprinted in *Papers of John von Neumann on Computing and Computer Theory*, W.F. Aspray and A.W. Burks, eds., Charles Babbage Inst. Reprint Series for the History of Computing, Vol. 12, MIT Press, Cambridge Mass., and Tomash Publishers, Los Angeles, 1986.
2. D.E. Knuth, "John von Neumann's First Computer Program," *Computer Surveys*, Vol. 2, 1970, pp. 247-260.
3. M. Campbell-Kelly and M.R. Williams eds., *The Moore School Lectures*, Charles Babbage Inst. Reprint Series for the History of Computing, Vol. 9, MIT Press, Cambridge Mass., and Tomash Publishers, Los Angeles, 1986.
4. H.H. Goldstine and J. von Neumann, "Planning and Coding of Problems for an Electronic Computing Instrument," Inst. for Advanced Study, Princeton, N.J., 1947-1948; reprinted in *Papers of John von Neumann on Computing and Computer Theory*, W.F. Aspray and A.W. Burks, eds. (see ref. 1).
5. M.V. Wilkes, *Memoirs of a Computer Pioneer*, MIT Press, Cambridge Mass., 1985.
6. M.V. Wilkes, "The EDSAC," *Int'l Symp. Automatic Digital Computation*, National Physical Laboratory, Mar. 1953, pp. 16-18; reprinted in *The Early British Computer Conferences*, M.R. Williams and M. Campbell-Kelly eds., Charles Babbage Inst. Reprint Series for the History of Computing, Vol. 14, MIT Press, Cambridge Mass., and Tomash Publishers, Los Angeles, 1986.
7. "Report of a Conference on High Speed Automatic Calculating Machines," University Mathematical Laboratory, Cambridge, Jan. 1950; reprinted in *The Early British Computer Conferences*, M.R. Williams and M. Campbell-Kelly, eds. (see ref. 6).
8. J.C.P. Miller, *The Airy Integral*, Cambridge Univ. Press, Cambridge, UK, 1946.
9. M.V. Wilkes, "Electronic Calculating-Machine Development in Cambridge," *Nature*, Vol. 164, Oct. 1, 1949, pp. 557-558.
10. D.R. Hartree, *Numerical Analysis*, Clarendon Press, Oxford, UK, 1952, p. 136.
11. S. Gill "The Diagnosis of Mistakes in Programmes on the EDSAC," *Proc. Royal Society*, Series A, Vol. 206, 1951, pp. 538-554.
12. D.J.Wheeler "Program Organisation and Initial Orders for the EDSAC," *Proc. Royal Society*, Series A, Vol. 202, 1950, pp. 573-589.
13. M.V. Wilkes, D.J. Wheeler, and S. Gill, *The Preparation of Programs for an Electronic Digital Computer*, Addison-Wesley, Cambridge, Mass., 1951; reprinted in Charles Babbage Inst. Reprint Series for the History of Computing, Vol. 6, MIT Press, Cambridge, Mass., and Tomash Publishers, Los Angeles, 1982.
14. C.J. Bashe et al., *IBM's Early Computers*, MIT Press, Cambridge, Mass., 1986, pp. 321-323.
15. G.M. Hopper, "The Education of a Computer," *Proc. ACM Nat'l Conf.*, May 1952, pp. 243-249.

Martin Campbell-Kelly's biography appears at the end of the Introduction, on page 9.

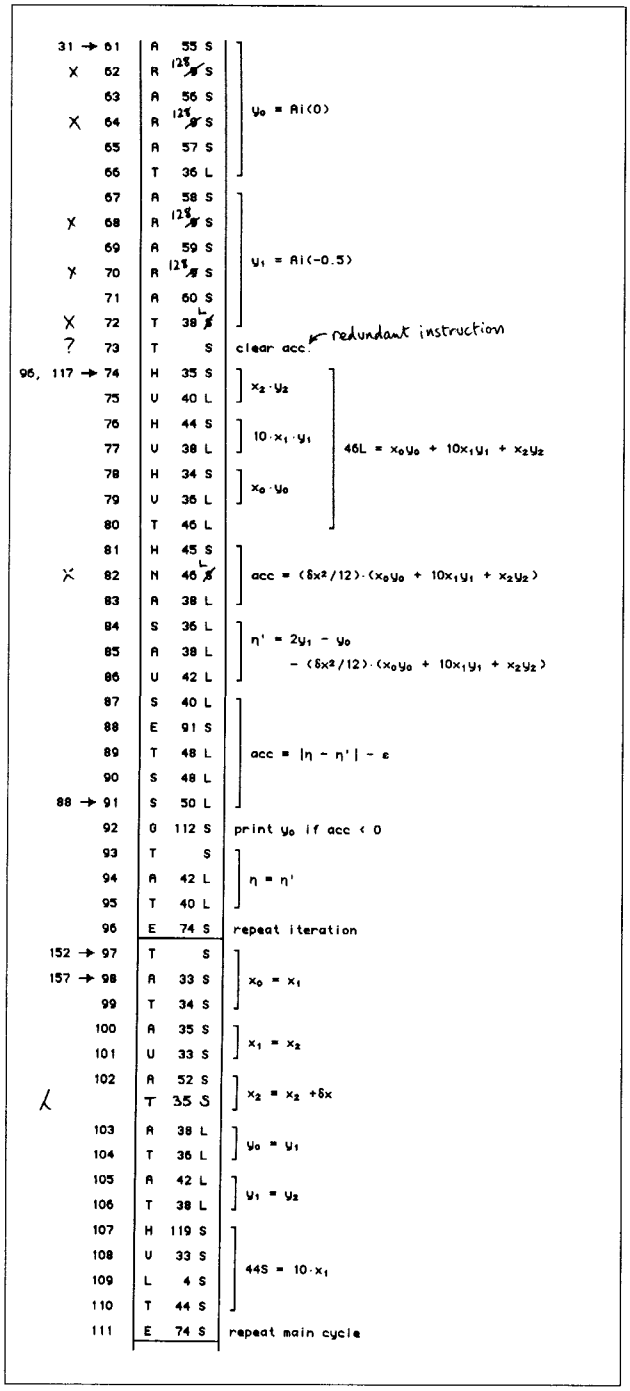
Appendix: A partially corrected version of the Airy program

Main program



(Appendix continued on the following page)

Main program, continued



Print routine