The other Turing machine

B. E. Carpenter* and R. W. Doran†

Department of Computer Science, Massey University, Palmerston North, New Zealand

In a little known report written in 1945, A. M. Turing made a detailed proposal for the construction of a stored program computer. Although sharing some ideas with von Neumann's draft report of the same year, Turing's proposal contained a wide range of novel and formative concepts. These include subroutines, the stack and a micromachine architecture. This paper analyses his report in general terms and in detail, and describes his ideas in modern terms.

(Received October 1975)

1. Introduction

In late 1945 A. M. Turing prepared a report, subsequently placed before the Executive Committee of the National Physical Laboratory (Great Britain), entitled 'Proposals for Development in the Mathematics Division of an Automatic Computing Engine (ACE)' (Turing, 1945). In fact, it was presented to the Committee on 19 March 1946. This is an extremely important document since it contains the first written statement of many of the ideas underlying computing and programming. Unfortunately Turing's report is not well known and its importance has not been widely acknowledged.

The Pilot ACE computer was publicised and well known. It has always been an object of interest because it appeared to be the only early computer of unconventional design, not following the models of EDVAC or the IAS machine. The Pilot ACE was sometimes dismissed as just another three-address machine but as Wilkinson put it 'this form of classification is not particularly appropriate' (Wilkinson, 1954). Why was Pilot ACE different?

The reason for the difference was mentioned indirectly by Wilkinson when he stated that the Pilot ACE was 'based on an earlier design by A. M. Turing'. This is the earliest published reference to Turing's report that we have found. The next is a direct reference by M. Woodger in an article in *Computer Weekly* (Woodger, 1969). This states that Turing's proposal 'even had the idea of subroutines, programs on Hollerith punched cards . . . .' To intrigue us even more, the report was discussed in somewhat greater detail by R. Malik in *Data Systems* (Malik, 1969).

Our interest in Turing's work was further aroused by a paper in *Machine Intelligence* 7 in which Brian Randell discusses the special purpose electronic code breaking machines built in Britain during the Second World War (Randell, 1972). This paper also discusses the possible wartime meeting between Turing and von Neumann and the probable exchange of ideas between the two. In his book *The Origins of Digital Computers* Randell further discusses the origin of the stored program concept and mentions that Turing's report on ACE refers to von Neumann's draft report on EDVAC (Randell, 1973; von Neumann, 1945).

The purpose of this paper is to analyse Turing's 1945 report and describe his proposals in modern terms. At the same time we shall compare his report with von Neumann's and try to show how they differ in general approach.

We shall not come to any definite conclusions as to which of these two great inventors was the originator of any particular concept and have no wish to cause controversy. Both were working as part of teams, and these reports were not intended for publication so did not contain specific references or acknowledgements. Goldstine's description of the period makes it clear how difficult it is to trace the origin of ideas within such active development teams (Goldstine, 1972). Of course, the originator of an idea was not necessarily the first to write it down, and of course many of the ideas which first became practicable in 1945 can be traced back to Babbage.

Although von Neumann's report was circulated somewhat before Turing's, the former's ideas were changing continuously, as indeed were the latter's. Many of the comments made in the present paper are specific to the situation in mid- or late-1945 and would not even apply to 1946, by which time the main direction of computing developments for at least ten years was firmly set. We know (Knuth, 1970) that even before von Neumann's draft report was circulated he had changed his mind as to some of the details of his machine. Turing's design also evolved considerably (Malik, 1969). Both reports ultimately led to the construction of computers, and although EDVAC and Pilot ACE were substantially different from the machines of the reports they clearly showed their separate origins.

In the remainder of this paper we give direct quotations from the two 1945 reports; those from Turing are in italic script. Page numbers refer directly to the original reports.

2. Overall impressions

The 1945 proposals by von Neumann and Turing both describe first attempts at computer architectures, but at different stages of development.

Von Neumann's paper is a draft and is unfinished—for example internal cross references are left blank. More importantly, it is incomplete: neither the I/O mechanisms nor details of the central control are spelled out. Knuth fills in some of the missing pieces, mainly from a letter written by von Neumann to Goldstine (Knuth, 1970).

Turing's paper, on the other hand, is a complete description of a computer, right down to logical circuit diagrams, with an exhaustive thirteen page analysis of the physical properties of the memory, and a cost estimate of £11,200. In fact, Turing's proposal is quite possibly the first complete design of a stored program computer architecture. He did realise that the design, though complete, was not necessarily final and that it would need to be modified in the light of experience. He says this explicitly on pages 18-19. ('LC' and 'CA' together form the control unit and an 'instruction table' is a program).

Although complete and workable circuits for LC and CA have been described in this report these represent only one of a considerable number of alternatives. It would be advisable to investigate some of these before making a final decision on the circuits. Too much time should not however be spent on this.

*Present address: CERN, 1211 Geneva 23, Switzerland.
†Present address: Amdahl Corporation, Sunnyvale, California 94086, USA.

We shall learn much more quickly how we want to modify the circuits by actually using the machine. Moreover if the electronic part is made of standard units our decisions will not be irrevocable. We should merely have to connect the units up differently if we wanted to try out a new type of LC and CA. . . .

The earlier stages of the making of instruction tables will have serious repercussions on the design of LC and CA. Work on instruction tables will therefore start almost immediately.

Note the stress placed on software development: Turing appeared to regard the logical design as a solved and therefore less interesting problem. In actuality the hardware design of ACE was refined for many years before it was built—the very delay of which Turing was afraid.

The difference which is most immediately apparent when comparing the two proposals is one of style. Von Neumann's report is detailed with every step completely justified and explained. Turing's is divided into two sections: a 'descriptive account' which is easy and fascinating to read because of the number of ideas gathered into a mere eighteen pages; and the 'technical proposals'. These make few concessions to the reader and are very hard to follow. This is partly because of the inherent complexity of control circuits in serial machines, and partly because Turing was building on a foundation laid by von Neumann, as he says on page 3:

The present report gives a fairly complete account of the proposed calculator. It is recommended however that it be read in conjunction with J. von Neumann's Report on the EDVAC.

However, many of Turing's obscurities must be blamed on his disregard for details as opposed to ideas. The technical proposals are marred by many trivial errors, some of which are clearly due to his changing his mind in the course of writing.

On reading more deeply into the reports, it is found that von Neumann and Turing had quite different views of what a computer is. Von Neumann regarded it as a device for performing numerical computation. Although it could be used otherwise, it was basically what we now classify as a programmable calculator. In his own words (page 3): 'Since the device is primarily a computer, it will have to perform the elementary operations of arithmetics most frequently. There are addition, subtraction, multiplication and division: $+$, $-$, $\times$, $\div$.' Von Neumann spends a much larger proportion of his draft discussing arithmetic operations than does Turing. As it happens, von Neumann's first program was for sorting—a non-numerical application chosen for its complexity (Knuth, 1970).

Turing's view was that a computer is primarily for obeying programs, most of which happen to involve numerical processing. He discusses at length how programs are to be written and processed. He also gives a list of possible applications. Along with numerical applications, Turing suggested that ACE could be used to solve jigsaw puzzles and to play chess—even perhaps *very good chess*. He pointed out that it could be used *to count the number of butchers due to be demobilised in June 1946 from cards prepared from the army records* (page 16) but he considered such tasks *can and should be done with standard Hollerith equipment*.

The probable reason why Turing was so aware of the importance of programming was that he had known about it for years. His famous paper *On Computable Numbers* (Turing, 1936) defined computability in terms of an abstract but plausible machine—nowadays called a Turing machine. This was certainly thought of as being programmed in a 'hard-wired' way. One program, that for a universal machine that could simulate any other, was rather complex and definitely non-numerical. It is reasonable to view the universal Turing machine as being programmed by the description of the machine it simulates; since this description is written on the memory tape of the universal machine, the latter is an abstract stored program computer.

The difference in attitude between von Neumann and Turing is evident in how their central processors deal with instructions. Early calculators had programs on punched tapes through which they stepped, executing each instruction as it turned up. Von Neumann retained this attitude in his 1945 report, for he thought of the processor as receiving a stream of orders from consecutive memory locations. He never explicitly mentions an instruction address register—this was unnecessary, for the next instruction comes from the current position in the memory tape. Turing did have an instruction address register explicitly containing the instruction number IN, i.e. the position of the next instruction.

Their attitudes to branch instructions also differ. Von Neumann considers these to be (page 85) 'orders for CC to transfer its own connection with M to a different point in M'. To Turing, branch instructions (page 11) *merely specify the number of the next instruction*. Similarly, he regards access to variables much as we do today. Von Neumann, in contrast, speaks of 'transient transfers' during which 'the place of the minor cycle which contained the transfer order must be remembered' (page 88), presumably in a nascent instruction address register. In summary, Turing's concept of memory was much closer than von Neumann's to a random access addressable device, though of course von Neumann had firmly formed this concept by 1946, for it is in his later proposals.

As assumed throughout the above discussion, both machines revolve around the idea of a stored program. This is often considered to be the crucial formative idea behind computers and there has been much interest in who originated it. Von Neumann's proposal was the first to describe a stored program in any detail, apart from Turing's 1936 abstract machine (from which Turing derived the term 'table' for 'program'). In his book on the history of computing (Goldstine, 1972; Chapter 7) Goldstine describes how the stored program concept gained gradual acceptance within the ENIAC team in late 1944. However, the stored program is a many-sided concept. Firstly there is the idea that instructions may be encoded as numbers—a very old idea that was common knowledge in 1945. Then there is the realisation that instructions could be stored in a memory along with other data—surely Turing's idea of 1936.

Next we come to the consideration that it is desirable, from a practical viewpoint, to store instructions as numbers in the temporary store. This is firstly so that they may be obtained quickly and secondly so that they can be modified. The first reason became important only when calculators went electronic. Of course, plugboard wired programs do not give rise to the mechanical delays of paper tape, but they are relatively difficult to change. This problem is more pressing for a calculating engine than for a code breaking device which would in fact benefit more from stored data. The first suggestion of using a stored program has been documented as due to Eckert in January 1944 (Eckert, 1944). That speed is the main advantage was mentioned on page 9 of von Neumann's 1945 report and restated in 1946 (von Neumann, 1946, page 26).

The idea of program modification has at least three aspects. Firstly, there is modification of addresses of instructions as they execute. Von Neumann allowed for this by distinguishing data and instructions by a one-bit tag. A store into an instruction could modify only the address part. This was the only means of achieving conditional branches or array indexing in the draft report.

The second aspect of modification is the manipulation of instructions as if they were numbers. Von Neumann does not take this step but Turing is clear about it, and believes it to be necessary for conditional branching (page 11):

We wish to be able to arrange that the sequence of orders can divide at various points, continuing in different ways according to the outcome of the calculations to date . . .

These requirements can largely be met by having the instructions on a form of erasible memory, such as the delay lines. This gives the machine the possibility of constructing its own orders; i.e. there is always the possibility of taking a particular minor cycle out of storage and treating it as an order to be carried out. This can be very powerful. Besides this we need to be able to take the instructions in an order different from their natural order if we are to have the flexibility we desire. This is sufficient.

Nowadays we do not regard the ability to modify instructions 'on the fly' as being important or desirable—after all, we have genuine conditional branches and index registers. However, the third aspect of modification is extremely important. This is the ability of one program to process another, treating it as data. This we shall see was also suggested in Turing's proposal. As von Neumann gave each word a nonoverrideable tag, he could not manipulate instructions in this way. Thus what we now regard as one of the fundamental characteristics of the von Neumann machine was, as far as we know, suggested independently, if not originally, by Turing.

Returning to conditional branching, both 1945 machines had to construct a branch instruction and then execute it. Von Neumann employed an instruction to select (on a condition) one of two numbers which was then stored into the address field of an unconditional branch. Four instructions were thus needed to execute a conditional branch. Turing had a more basic mechanism as he explains (page 11):

We must now explain in more detail how it comes about that we can branch the sequence of instructions and arrange for subsidiary operations. Let us take branching first. Suppose we wish to arrange that at a certain point instruction 33 will be applied if a certain digit is 0 but instruction 50 if it is 1. Then we may copy down these two instructions and then do a little calculation involving these two instructions and the digit D in question. One form the calculation can take is to pretend that the instructions were really numbers and calculate

$$D \times \textit{Instruction } 50 + (1 - D) \times \textit{Instruction } 33 \ .$$

The result may then be stored away, let us say in a box which is permanently labelled 'Instruction 1'. We are then given an order of type B saying that instruction 1 is to be followed, and the result is that we carry out instruction 33 or 50 according to the value of D.

In point of fact, in his examples, he chose one of two addresses by the above mechanism and thence constructed a branch. This took some twenty instructions. A special parallel shifting device and logical operations on entire words were included in the arithmetic unit and these facilitated the construction of instructions.

The Turing branching mechanism quoted above is used in recursive function theory, which may explain its origins. It is strange that conditional branching was a stumbling block to both von Neumann and Turing, especially since the program for an abstract Turing machine is just one large decision table.

Another difference in approach between the two authors was Turing's decision to implement what we would now call a micromachine, not because of its size but because of the limited function of each instruction. Von Neumann thought of each instruction as an operation complete in itself; Turing definitely considered his instructions to be components of basic operations. Programs were to be written in terms of basic subroutines (page 29):

The majority of actual instruction tables will consist almost entirely of the initiation of subsidiary operations and transfers of material.

Such use of subroutines was old hat to Turing, who had thus built up programs for his abstract machines (though in that case using macros).

From the standpoint of simplicity of hardware implementation, a most important criterion at the time, the idea of a micromachine seems to be very practical indeed. Here Turing left the mainstream of computers, or rather anticipated it, for it was not for many years that machines with writeable microprograms were produced. We cannot truly credit Turing with the invention of microprogramming, for nowhere does he consider the purpose of his machine to be the emulation of another. The machine is rather considered to be structured in layers of subroutines. However, his example programs do set up the machine to look like another one in a manner very suggestive of emulation.

Turing went one step further towards a micromachine. He introduced an instruction whose sole purpose was to turn on one of 256 control lines or valves. In the following explanation he defines something very reminiscent of firmware (page 26; an ITO—instruction table operation—is a subroutine):

It is intended that the outputs of these valve elements should be connected in various ways into the circuit when it is desired to try out new circuit arrangements. It is thought that they may often provide means for doing things simply which could be done lengthily as an ITO. To an extent this represents a compromise between the new system of 'control by paper' and the old plugboard and soldering-iron techniques.

Since his machine was to be programmed mainly in subroutines, much thought was needed to make this practicable. In fact Turing had thought it right through and showed clearly how programming could be efficiently organised.

To start with there was the problem of using subroutines from various points of call (page 11):

We also wish to be able to arrange for the splitting up of operations into subsidiary operations. This should be done in such a way that once we have written down how an operation is to be done we can use it as a subsidiary to any other operation.

This was to be facilitated by having all branch instructions leave their return address in a special register; thus the main purpose of the branch was for calling subroutines. There was no simple way to branch back to this saved address since a branch instruction would have to be constructed. Turing's solution is a model of clarity (page 12; TS is a temporary storage register):

When we wish to start on a subsidiary operation we need only make a note of where we left off the major operation and then apply the first instruction of the subsidiary. When the subsidiary is over we look up the note and continue with the major operation. Each subsidiary operation can end with instructions for this recovery of the note. How is the burying and disinterring of the note to be done? There are of course many ways. One is to keep a list of these notes in one or more standard size delay lines (1024), with the most recent last. The position of the most recent of these will be kept in a fixed TS, and this reference will be modified every time a subsidiary is started or finished. The burying and disinterring processes are fairly elaborate, but there is fortunately no need to repeat the instructions involved, each time, the burying being done through a standard instruction table BURY, and the disinterring by the table UNBURY.

Thus Turing described a software stack for subroutine linkage; we would nowadays refer to BURY and UNBURY as PUSH

and POP respectively.

He thought of subroutines as being maintained in libraries as machine instructions with symbolic addresses (page 28):

Permanent form.—The same instruction will appear in different machine forms in different jobs, on account of the renumbering technique as described in pp. 13, 14. Each of these machine form instructions arises from the permanent form of the instruction. These permanent forms are on Hollerith cards and are kept in a sort of library.

To simplify life he thought of programs being written mainly symbolically in what he called a popular form. Here addresses could be coded numerically relative to a symbolic label, and op-codes and other constants found in the permanent form were omitted (page 28; CAO is the central arithmetic organ):

Popular form.—Besides the cards we need some form of the table which can be easily read, i.e. is in the form of print on paper rather than punching. This will be the popular form of the table. It will be much more abbreviated than the machine form or the permanent form, at any rate as regards the descriptions of the CAO. The names of the instructions used will probably be the same as those in the permanent form.

He assigned fields to program cards (page 13):

	Columns
Genuine input	*41–72*
Repeat of destination	*26–40*
Popular name of group	*1–8*
Detail figure (popular)	*9–11*
Instruction (popular)	*12–25*
Job number	*73–77*
Spare	*78–80*

The 'popular name of group' was the name or label of a subroutine. The 'detail figure' was the displacement from the preceding label. The instruction itself was always given symbolically but the 'genuine input' of 32 digits was empty at first and partially filled in for the permanent form. As a concrete example, here is the popular form of a short program named INDEXIN (page 29):

	INDEXIN
1	*Q, 0000,0100,0000,0000*
2	*TS 6–TS2*
3	*ADD 'A'*
4	*ROTATE 16*
5	*TS 4–TS 6*
6	*TS 6–TS 9*
7	*TS 27–TS 6*
8	*TS 6–TS 10*
9	*OR*
10	*TS 8–TS 6*
11	*B, 1, INDEXIN 11*
12	*TS 6–TS 28*
13	*B, BURY*

Note that line 11 refers to its own address. Line 13, of course, should be B, UNBURY. We will discuss INDEXIN in more detail later.

Having set up his subroutine library, Turing describes the process of assembling a complete program by externally link-editing the required subroutines, rather than by wastefully loading each subroutine at a fixed address (page 14):

Instead, when a new job appears we take the complete set of cards involved and make a new copy of each of them; these we sort into the order of popular group name and detail figure. We then renumber them consecutively in the binary scale. This number goes into the columns described as 'repeat of destination'. The renumbering may be done either with a relay counter attached to a collater, or by interleaving a set of master cards with the binary numbers in serial order. To complete the process we have to fill in other instructions numbers in binary form into the genuine input, e.g. if an instruction in popular form were ". . . and carry out instruction Potpan 15" the genuine input will have to be of form ". . . and carry out instruction 001101 . . . 1" where 001101 . . . 1 is the new number given to Potpan 15 in this particular job. This is a straightforward sorting and collating process.

It would be theoretically possible to do this rearrangement of orders within the machine. It is thought however that this would be unwise in the earlier stages of the use of the machine, as it would not be easy to identify the orders in machine form and popular form. In effect it would be necessary to take an output from the calculator of every order in both forms.

Here we see Turing's recognition that programs can be processed by other programs, as mentioned earlier. He also anticipates handling difficulties by sorting the group names and suggesting an output of the program in symbolic and machine forms. Another practical point is possibly the first recorded plea for adequate program documentation (page 29):

In addition to these we must recognise the 'general description' of a table. This will contain a full description of the process carried out by the machine acting under orders from this table. It will tell us where the quantities or expressions to be operated on are to be stored before the operation begins, where the results are to be found when it is over and what is the relation between them. It will also tell us other important information of a rather dryer kind, such as the storages that must be left vacant before the operation begins, those that will get cleared or otherwise altered in the process, what checks will be made, and how various possible different outcomes of the process are to be distinguished. It is intended that when we are trying to understand a table all the information that is needed about the subsidiaries to it should be obtainable from their general descriptions.

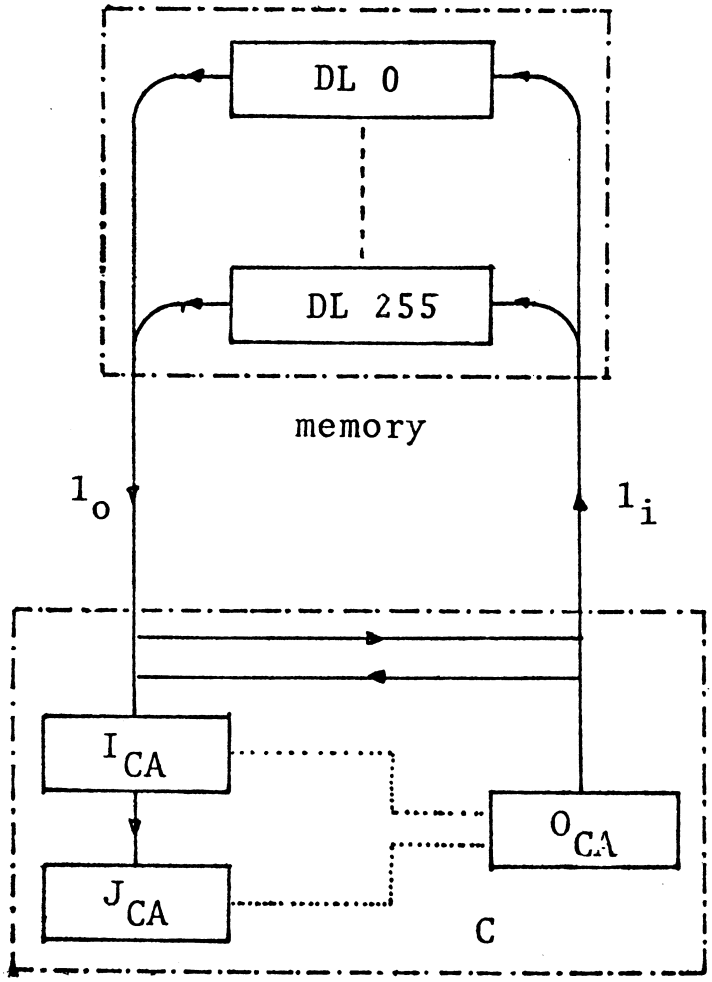
He practised what he preached: here is his description of INDEXIN (page 29):

INDEXIN (General Description). The minor cycle whose position is described in digits 17–32 of TS 27 is transferred to TS 28. The contents of TS 2, 3, 4, 5, 6, 8, 9, 10 get altered in the process.

(he did overlook that register TS7, used for the control of shifting, had to contain 16 for INDEXIN to work).

Towards the end of this report, Turing was tiring of writing programs in the detail required and was becoming worried by their size. The solution he proposed was to use the machine to help out with what appears to be runtime macro expansion. A possible interpretation of the following passage is that he intended a subroutine to be processed just before its first use, but how address relocation was to be handled is unclear (page 32; CALPOL is a large example program):

It will be evident that the table CALPOL is somewhat wasteful of space. Each time a subsidiary operation is required we have to repeat B, BURY, and each time we make a transfer we have to do it in two stages, each of which uses a whole minor cycle of which most is wasted. It is possible to avoid this waste of space by keeping the instruction tables in some abbreviated form, and



operation	result in O_{CA}
+	$I_{CA}+J_{CA}$
-	$I_{CA}-J_{CA}$
×	$I_{CA}\times J_{CA}$
/	I_{CA}/J_{CA}
√	$\sqrt{I_{CA}}$
i	I_{CA}
j	J_{CA}
sd	if $O_{CA}\geq 0$ then I_{CA} else J_{CA}
bd	binary-decimal conversion
db	

Fig. 1 Von Neumann's first machine design
(*a*) *Above* **bus structure**
(*b*) *Right* **operations of CA**

expanding each table whenever we want it. This will require a table EXPAND, and will require each table to include appropriate references to the table EXPAND. These references will however be put in by EXPAND itself (when working under contract to a higher authority), just as EXPAND will put in the references to BURY and UNBURY.

It is clear from a reading of Turing's report that whether or not he originated the stored program, it is probable that he should be regarded as the originator of programming as we now know it. Before going into rather more detail of certain aspects of his ideas, we quote the following farseeing comments in which programmers may take some delight (page 18):

Instruction tables will have to be made up by mathematicians with computing experience and perhaps a certain puzzle solving ability. There will probably be a great deal of work of this kind to be done, for every known process has got to be translated into instruction table form at some stage. This work will go on whilst the machine is being built, in order to avoid some of the delay between the delivery of the machine and the production of results. Delay there must be, due to the virtually inevitable snags, for up to a point it is better to let the snags be there than to spend such time in design that there are none (how many decades would this course take?). This process of constructing instruction tables should be very fascinating. There need be no real danger of it ever becoming a drudge, for any processes that are quite mechanical may be turned over to the machine itself.

3. Turing's micromachine

Many features and parameters of Turing's proposal were similar to and presumably taken over from the early plans for EDVAC. Both machines were to be binary, serial and use 32-bit words. The memory was to consist of 32-word delay lines or 'tanks' bussed to the control unit. Memory locations had unique addresses consisting of two fields, the address of the delay line and the position of the word in the line. The two fields were somewhat different in that a tank could be selected at random but a particular word was available during its own minor cycle (32 bit-periods) of the major cycle (32 word-periods). Thus some delay could be expected before a word could be accessed. All tanks were in step so the delay was the same for all words with the same 'internal' field regardless of which tank they were in. For many practical purposes, however, tank boundaries could be ignored. Turing follows von Neumann in using the terms 'minor cycle' for a word and 'major cycle' for a delay line, but unlike von Neumann he suggests 'word' as an alternative (page 5).

In the details of what we now call their central processors, the two machines are very different. Von Neumann proposed an

arithmetic unit with two input registers and one output register. The unit had a complete range of operations from direct transfers through to division and square-rooting. It omitted the simple Boolean operations, presumably because these were not required of a calculator.

Turing, however, organised his processor around an array of of temporary storage registers, some used for arithmetic operations and some purely as temporaries. Von Neumann subsequently included similar registers (Knuth, 1970), calling them 'short tanks'. Turing may have known this but it seems unlikely for he uses a different name, *temporary storage*, and considers many of them to be internal to the arithmetic unit rather than as a first level store. In von Neumann's modified design, instructions were to be held in short tanks but Turing allowed this for one special register only.

The bus organisations of the two 1945 proposals are sketched in **Figs. 1** and **2.** It will be noticed that at the level of detail reached in his report, von Neumann had no instruction address register or instruction register. Turing had both—CD and CI (page 20):

(*1*) *A short storage* (*like a TS*) *called* current data *CD. This contains nothing but the appropriate* instruction number *IN, i.e. the position of the next instruction to be carried out.*

(*2*) *A short storage called* current instructions *CI. This contains the instruction being or about to be carried out.*

A delay line memory causes certain design problems since it is not truly random access. If instructions are accessed at random then there is an average delay of half a major cycle, or 512 μsec, before each instruction. To overcome this, various tricks were used in designing and programming machines with cyclic memories (which include discs and drums).

As explained in Section 2, von Neumann thought of instructions streaming into the control unit at exactly the rate at which the memory could deliver them, in the 'octroyed temporal sequence' (page 76). For this to work there must be overlapping of the execution of one instruction with the receipt of the next, but von Neumann gives no details. Of course some instructions took longer than one minor cycle (32 μsec) to execute and then the control unit would skip 32 cycles and start again next time round, about one millisecond later. Instructions to access memory would also take a whole major cycle.

The registers CI and CD in Turing's machine were short delay lines. Before an instruction could be executed it had to be 'distributed' and parts of it staticised, taking one minor cycle. Turing therefore used a two cycle mode of instruction execution: distribute, execute, distribute, execute, etc.

Since the minimum execution time was one minor cycle, this meant that instructions had to be two apart in memory. To ensure that control would not always *just miss the boat* (page 22) the instruction address register CD was incremented by two during the distribution cycle. To simplify programming, however, Turing made a masterly change in his link editing process. Virtual addresses of form $a_1 \ldots a_{15}$ were to be mapped to physical addresses as $a_{15}a_1 \ldots a_{14}$, where the delay line number was now $a_5 \ldots a_{14}$ and the word address $a_{15}a_1a_2a_3a_4$ (note the digits of numbers are written with the least significant first). Thus, when a physical address was incremented by two the corresponding virtual address was incremented by one. The nett effect of this unusual scheme was to divide the main memory into two interleaved halves.

An instruction could be distributed in the same cycle during which it was fetched, so the normal two-cycle mode above can be expressed as follows: fetch, distribute, increment; execute; fetch, distribute, increment; execute; etc.

Most instructions would be transfers within temporary storage and would fit into this pattern. However, even the fastest arithmetic instruction would take two cycles. This was because Turing dealt with double length numbers, requiring 64 bit-periods, whereas von Neumann always rounded to 32 bits. In order that instruction fetches would not get behind, the following sequence was possible: fetch, distribute, increment; execute; execute, fetch; distribute, increment; execute, fetch; etc.

To enable this, as shown in Fig. 2, Turing had two memory output buses, one for instructions and one for data. If two arithmetic operations occurred in the same sequence then the processor would miss a fetch, but most sequences of instructions were to consist of transfers into registers, a single operation, and the distribution of results. Turing's scheme was therefore a very effective method of minimum access coding. Branch instructions needed no execute cycle so they had the effect of guaranteeing a return to the normal sequence. If a slow operation such as multiplication (16 cycles) were followed immediately by a suitable branch, which would be fetched during the multiplication, then there would be no delay afterwards. This degree of optimisation would, however, require virtuoso programmers.

In both von Neumann's and Turing's design, data words were to be binary two's-complement numbers. Von Neumann regarded them as fractions in the range (−1, 1) whereas to Turing they were whole numbers, though in exactly the same form. However, Turing did not consider these to be other than parts of more complicated numbers.

As mentioned in Section 2, von Neumann distinguished data from instructions with a one bit tag. However, a data word encountered among instructions was treated as an 'immediate' (page 89) load instruction. There were three other forms of instruction. The first was the unconditional branch and the second was to load the contents of some addressed location. This he considered to be a form of temporary branch or

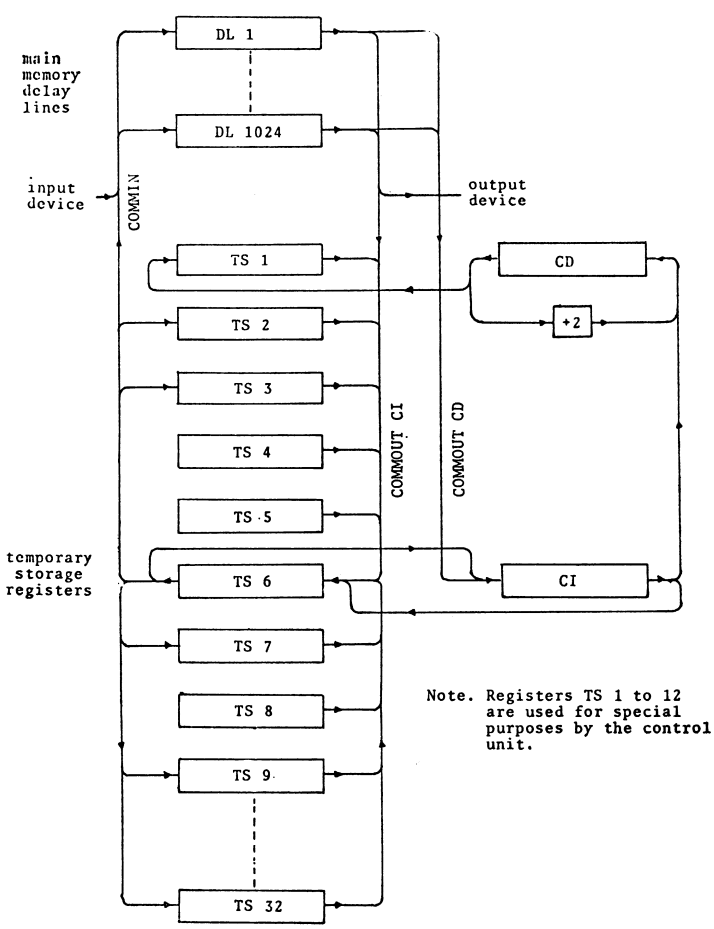


Fig. 2 Bus structure of Turing's machine

'transient transfer' of the point of connection of control to memory.

The third and major group of instructions combined an arithmetic operation (see Fig. 1) with a specification of what to do with the result. It could be left in O_{CA}, or stored in a specified address, in the next instruction position or back into I_{CA}. For each destination, O_{CA} could be optionally cleared.

Turing's machine, in contrast, had more instructions which in general did less. The two main types are branches, which alter CD, and others which increment it as well as carrying out a specified operation. For branches, bits CD18–CD32 specified the address of the next instruction, but if CD17 was on then the next instruction was to be taken instead from register TS6. Nonbranch instructions cleared CD17 to prevent indefinite looping in TS6.

We now consider the instructions in order. As indicated in Fig. 2, register TS6 plays a central role in all transfers.

Type B Notation: B, address
Branch instruction.
Load CD17–CD32 from corresponding bits of instruction.

Type K TS6–DL *x*, *y*
Store TS6 in main memory

Type L DL *x*, *y*–TS6
Load TS6 from main memory

Type M TS*x*–TS6
Load TS6 from temporary storage

Type N TS6–TS*x*
Store TS6 to temporary storage

Type O
Punch a card, as twelve 32 bit binary numbers from the first twelve words of a specified delay line

Type P
Read a card, as twelve 32 bit binary numbers into the first twelve words of a specified delay line

Type Q Q, data
Load TS6 with data in last 16 bits of instruction

Type R
Set TS8 to the specified logical combination of TS9 and TS10. Options available were $\wedge$, $\vee$, $\oplus$, $\neg$TS10 and zeros. (The mnemonic OR was used for $\vee$ and AND for $\wedge$)

Type S
Arithmetic operations

Type T
Instructions to turn on one of a set of 64 gates (as described in the last section)

The arithmetic operations need some further explanation. Registers TS2–TS5, TS7, TS11 and TS12 were connected to the arithmetic circuit shown in **Fig. 3.** Rather than performing a complete operation the arithmetic circuit was set in motion by twelve lines corresponding to bits in the instruction. This could result in a complete operation but in many cases a number of steps would be required.

Basically the arithmetic unit could, in two cycles, add to the double length value in TS4 and TS5 the product of the multiplicand (usually TS3) and four bits of the multiplier (usually

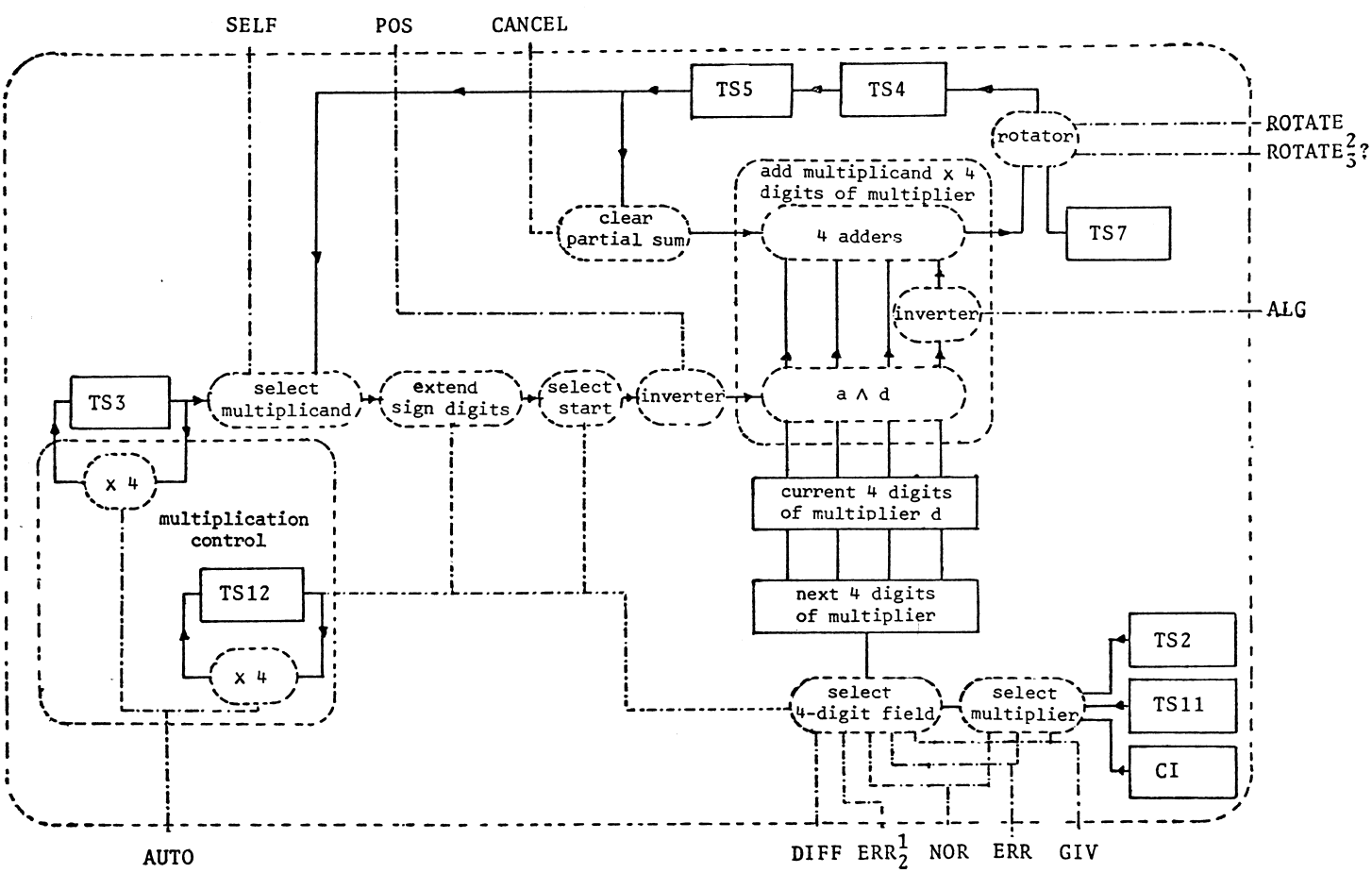
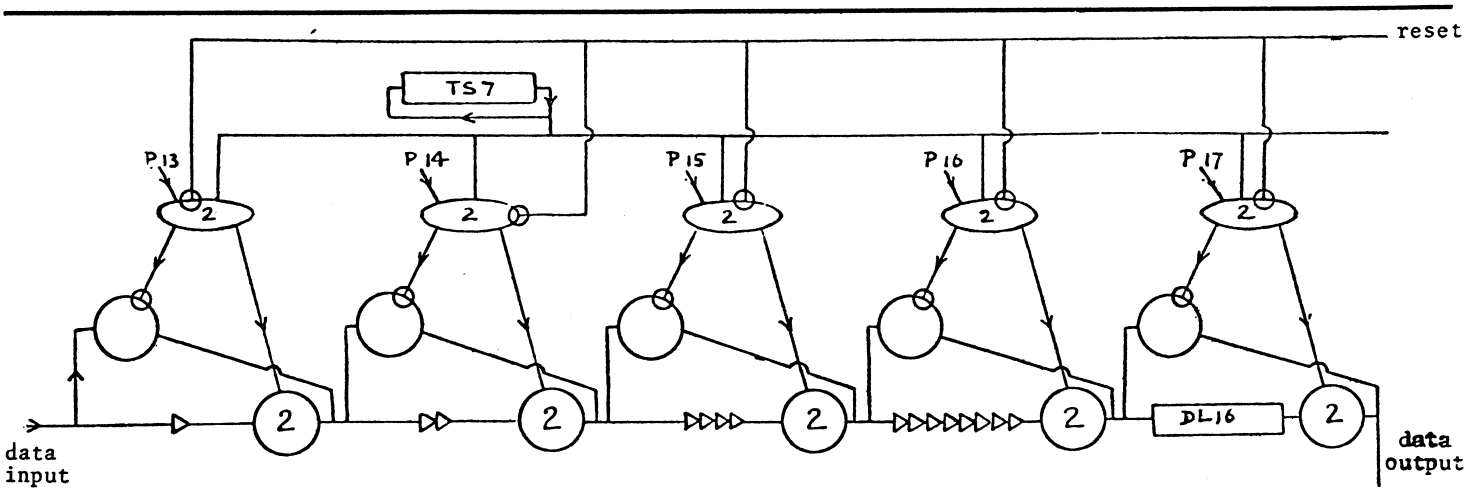


Fig. 3 Turing's arithmetic unit



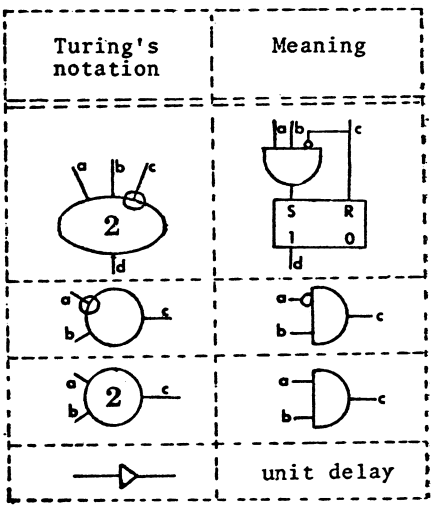


Fig. 4 Turing's rotater (DL16 is 16 unit delays)

TS2). In this case there would be no delay in accessing the next instruction. Various modifier bits in the instruction had the following effects:

NOR	multiplier to be bits 1–4 of TS2
ERR & ERR1	multiplier to be bits 10–13 of TS11 } for error calculation
ERR & ERR2	multiplier to be bits 14–17 of TS11 } for error calculation
GIV	multiplier to be bits 24–27 of instruction*
ALG	multiplier to be 2's complement (otherwise absolute)
POS	negate multiplicand
SELF	use TS4 and TS5 as multiplicand (otherwise TS3)
CANCEL	cancel previous content of TS4 and TS5.

*In some places, Turing says bits 23-26 instead.

For full multiplication, eight double cycles were required, choosing groups of four bits of the multiplier in sequence and shifting the multiplicand appropriately. This was done by initialising TS12 to zero except for a one in bit 1. When modifier bit DIFF was on, the multiplier digits were the four starting at the current position of the solitary bit in TS12 (bit 1 being the least significant). When modifier bit AUTO was on, TS3 and TS12 were shifted right by four bits each double cycle, and the arithmetic unit kept going for sixteen cycles, the position of the bit in TS12 being used to stop it.

The arithmetic circuit included a 'parallel' shift network. When modifier bit ROTATE was on, this shifted TS4, TS5 to the right (delayed it) by a quantity given somewhere in TS7. (The text specifies bits 1–5 of TS7, and the drawing shows bits 13–17, but since ROTATE was bit 19 of the instruction one would expect the relevant bits of TS7 to be later.) The rotation delay was to last for two or three minor cycles depending on whether ROTATE2 or ROTATE3 was on (again, this feature is not shown in the circuit diagrams).

Turing gave some examples of specific instructions, among them:

1. Addition, with POS GIV and value 1 in bits 24–27. i.e. TS4, 5 ← TS4, 5 + TS3 × 1
2. Short multiplication, with POS GIV CANCEL SELF and constant in bits 24–27 i.e. TS4, 5 ← TS4, 5 × constant.
3. Long multiplication with addition, with POS NOR AUTO DIFF i.e. TS4, 5 ← TS4, 5 + TS3 × TS2 (assumes TS12 contains a 1 in bit 1).

Turing used the mnemonics ROTATE to mean ROTATE2 and ADD 'A' for an addition with cancel, i.e. TS5 ← TS3. He did not propose hardware division or squarerooting, but did provide double precision, using four adders to accelerate multiplication.

The reader should now be able to follow the program INDEXIN given in Section 2. The first ten lines construct an instruction in TS6 to load the word addressed by TS27. This is then executed and the result stored in TS8. There are three points to note:

(*a*) line 13 should be B, UNBURY.

(*b*) the program assumes that TS7 contains 16 in bits 13–17 to control the shift. It is unclear how this could be arranged since only a shift could get them there in the first place. If the shift count had been taken from bits 17–21, Turing could have used Q,0000 1000 0000 0000; TS6–TS7.

(*c*) the first line, which loads the opcode for 'load into T6' into the right half of T6 should probably be Q0000, 0101, 0000, 0000.

It is difficult to comment on the originality of Turing's logical

design and electronic techniques. It is noteworthy that a large section of his report concerns engineering matters. There is a discussion of automatic error checking including the following (page 17):

There are three chief functions to be performed by the checking. It must eliminate the possibility of error, help to diagnose faults, and inspire confidence. We have not yet spoken at all of this last requirement. It would clearly not be satisfactory if the checking system in fact prevented all errors, but nobody had any confidence in the results. The device would come to no better end than Cassandra.

According to Malik (1969), by the end of the second world war Turing was an experienced, if amateur, electronics engineer, quite capable of discussing things in 'the style of a genuine four-in-the-morning system kicker'. It is now clear that Turing gained exposure to electronics while working on wartime cryptanalysis (Randell, 1976).

The discussion of the design of valve elements and delay lines is particularly detailed and mathematical—even to the level of recommending the use of 6SN7 valves! The impression gained is that the machine could readily be built, after sorting out Turing's minor inconsistencies. This is confirmed by the speed with which other delay line machines were constructed, using the same technology but different architectures.

Turing used a consistent set of symbols for describing his circuits at the logical level. His notation is derived from that of McCulloch and Pitts (1943) though apparently based on von Neumann's modifications—von Neumann, for example, used circles for logical elements. Turing extended the notation considerably, introducing a special symbol for a flip/flop. In his 1949 book *Calculating Instruments and Machines* Hartree uses Turing's notation and credits him with its origin (Hartree, 1949, p. 97, p. 102).

As an example of Turing's logical design, **Fig. 4** is his Fig. 33, the rotater. We do not know whether this is original but we have not seen it mentioned elsewhere. The idea of this circuit underlies a modern parallel shifting network. It was intended to be used over two cycles; when the instruction was distributed, the shift count in TS7 was staticised (as shown, from bits 13–17, but this must be incorrect as mentioned above). The shift took place over the following two cycles. (We have corrected a few minor ommissions in Fig. 4).

4. Example programs

It was tempting to entitle this section 'Turing's virtual machine' but this is perhaps a grandiose description of his small set of example programs, which he described as (page 28) *incomplete and crude because the whole project as yet exists only in imagination.*

However, rather than giving examples directly in machine language Turing went about building a set of basic subroutines in terms of which more complicated programs could be written. These subroutines were related in such a way that, in fact, a virtual machine was created.

The virtual machine operated primarily on floating point numbers of the following two-word form

β			
1–9	10–17	18–23	24–32
m	x	n	t

The first word was the mantissa, a two's complement integer β in the form handled directly by the underlying computer. The first field of the second word was an excess-256 exponent m:$(-256 \leq m \leq 255)$. The number represented was $\beta \times 2^m$. The second and third fields, both positive integers, were used to represent the known error in the number as $\pm x.2^m \times 2^{10-n}$. The fourth field was to hold identifying information for unspecified purposes.

Turing listed nine basic subroutines, describing their overall effects (summarised in **Table 1**). Only INDEXIN was given in its entirety. The subroutines used various temporary storage registers for specific purposes. TS27 was a sort of index register. INDEXIN could be used to load TS28 with the word addressed by TS27 and PLUSIND added one to this register.

Table 1 Turing's example basic subroutines

INDEXIN	TS28 ← M[TS27]
DISCRIM	TS24 ← if TS8 = 0 then TS16 else TS15
PLUSIND	TS27 ← TS27 + 1
TRANS45	TS20, 21 ← TS22, 23
BURY	M[TS31] ← TS1 + 1; TS31 ← TS31 + 1; go to M[TS1]
UNBURY	go to M[TS31 ← TS31 − 1]
MULTIP	TS22, 23 ← TS18, 19 × TS20, 21
ADD	TS22, 23 ← TS18, 19 + TS20, 21
BINDEC	Convert TS22, 23 to card image in DL11

Registers TS18 to TS23 formed a virtual arithmetic unit. MULTIP performed multiplication, ADD addition and TRANS45 a straight transfer. Operands were floating point. Note the parallel with registers I_{CA}, J_{CA} and O_{CA} of von Neumann's design.

TS31 was to be the return address stack pointer, apparently always pointing one 'above' the stack top. BURY would save the return address of the next subroutine branch. UNBURY retrieved the last saved address and branched to it.

To choose one of two words Turing used a routine DISCRIM. Depending on the contents of TS8 (the output of the 'logic unit') one of TS9 or TS10 was to be transferred to TS24; this parallels von Neumann's instruction 'sd'.

The final instruction BINDEC was to be used for output. The number in the 'accumulator' TS22, 23 was converted to 12 binary words in DL11 (the output buffer) which when punched on a card would give a decimal representation of the number in a Hollerith code. Von Neumann did not specify his operations 'db' and 'bd' but it is clear from Knuth (1970) that he intended a BCD representation of decimal numbers rather than standard card code.

Having set up the above basic subroutines, Turing gave an example of a larger program, CALPOL. This was to evaluate a degree 15 polynomial whose coefficients are given as double words in DL3, with the value of the variable in TS13, 14 and the result appearing in TS25, 26.

The version given by Turing (**Fig. 5**) is perhaps a little hard to follow. He has introduced abbreviations for transfers via TS6 but it is nevertheless quite complex. In **Fig. 6** we give a structured version of the same program with italicised comments and minor corrections. The coefficients are treated as pairs of words a_i, a_{i+1} with i ranging from 1 by 2 to 31. We conclude with some further notes on Fig. 6.

(*a*) TS20, 21 and TS22, 23 both hold the sum at different stages. We use the mnemonics sum1 and sum2. Turing does not specify how to clear TS20, 21 (in fact he says clear TS22, 23 in error) but we could use Q, 0; TS6–TS20; TS6–TS21.

(*b*) TS27 the index register is being loaded with the address DL3, 1 which was set up in DL1, 14. In fact Turing at this stage decided to place a load instruction in TS27 rather than just an address. This would make INDEXIN a lot simpler, but he does not mention this elsewhere.

(*c*) DL1, 15 which is placed into TS6 can be thought of as containing the address of the last element. The 1st element

was at 10000, 11, i.e. 1, 3 and the limit address 00000, 11, i.e. 32, 3. In fact Turing used 00000, 10 as the stopping value and tests by anding rather than by 'less than'.

(*d*) All labels like CALPOL8 are symbolic, not relative, though whether Turing intended this is unclear.

(*e*) We have switched these two instructions with the next to improve readability.

(*f*) We have changed TS2, 3 to TS9, 10—this reflects a change in Turing's design. Note that the limit value 32, 3 will give a zero result when anded with 1, 4 but not with the coefficient addresses 1, 3 through 31, 3.

(*g*) CALPOL40 has been changed to CALPOL50.

(*h*) Here Turing either leaves out a section of code to construct a branch instruction in TS6 or has changed the meaning of DISCRIM so that it does this. We have included code for this, assuming that TS7 contains a shift value of 16.

Table 2 Formative ideas in Turing's report

	page(s)
1. Binary implementation using standardised electronic logic elements*	throughout
2. Complete notation for combinational* and sequential circuits	throughout
3. Memory—Control—Arithmetic Unit—I/O architecture*	throughout
4. Stored program*	throughout
5. Conditional branch instructions (clumsy)*	11, 12 etc.
6. Address mapping	22
7. Instruction address register and instruction register	20 etc.
8. Multiple fast registers in CPU, for data and addressing	3 etc.
9. Microcode; hierarchical architecture	throughout
10. Whole-card I/O operations (almost DMA)	12, 24, 25
11. Complete set of arithmetic, logical and rotate orders	23 etc.
12. Built in error detection and margin tests	16, 17
13. Floating point arithmetic	6, 7
14. Hardware bootstrap loader (initial program load)	13, 25
15. Subroutine stack	12 etc.
16. Modular programming; subroutine library	28 etc.
17. Documentation standards	28 etc.
18. Link editor; symbolic addresses; programs treated as data	13, 14 etc.
19. Run time systems (I/O conversions; macro expansion??)	32 etc.
20. Nonnumerical applications	10
21. Artificial intelligence	10

*Also found in von Neumann's report

5. Conclusions

In **Table 2** we list the formative ideas which can be identified in Turing's 1945 report. By no means all of these were original, but to find them all within some 50 pages of typescript at that early date is startling.

Turing left the NPL in 1949, and was in fact out of touch with the ACE design team for some time before then. ACE changed quite drastically more than once before the pilot version was built. Many of Turing's ideas were not implemented. The microcontrol was replaced by complete arithmetic operations. The saving of the return address and the return address stack were abandoned. The instruction address register went too, so *every* instruction had to specify the delay line number and a 'wait time' for the next instruction, mixing absolute and relative addressing. Turing's random access approach to memory was therefore diluted.

There were some good new features. Rather than using TS6 as a routing register, the instruction set became a two address one and data could be transferred from a TS to any other in

CALPOL.
CALPOL 1. Clear TS 22, 23; DL 1, 14–TS 27; DL 1, 15–TS 29. CALPOL 8
CALPOL 8. B, BURY; B. INDEXIN; TS 28–TS 18; B, BURY; B, PLUSIND; B, BURY; B, INDEXIN; TS 28–TS19; B, BURY; B, ADD; B, BURY; B, PLUSIND; TS 27–TS 2; TS 29–TS 3; AND; Q, CALPOL 40; TS 6–TS 15; Q, CALPOL 37; TS 6–TS 16; B, BURY; B, DISCRIM; B, 1
CALPOL 37. TS 13–TS 18; TS 14–TS 19; B, BURY; B, TRANS 45; B, BURY; B, MULTIP; B, BURY; B, TRANS 45. CALPOL 49.
CALPOL 49. B, CALPOL 8.
CALPOL 50. TS 22–TS 25; TS 23–TS 26; B, UNBURY.

Fig. 5 Turing's example program

```
initialise
 0 → sum1     Clear TS20, 21; (a)
 1 → i        DL1, 14–TS27;   (b)
 32 → limit   DL1, 15–TS26;   (c)
repeat  CALPOL 8.  (d)
 a_i a_{i+1} + sum1 → sum2; i + 2 → i;
  a_i a_{i+1} → t; i + 2 → i
   B, BURY; B, INDEXIN;   TS28–TS18;
   B, BURY; B, PLUSIND;
   B, BURY; B, INDEXIN;   TS28–TS19
   B, BURY; B, PLUSIND;   (e)
  t + sum1 → sum2
   B, BURY; B, ADD;
 if i > limit then exit i.e. go to CALPOL 50
  (i < limit) → notdone
   TS27–TS9; TS29–TS10; AND;  (f)
  set up alternate addresses
   exit address → TS16      Q, CALPOL 50; TS6–TS16; (g)
   continue address → TS15 Q, CALPOL 37; TS6–TS15;
  if notdone then go to continue address else go to exit address
   place selected address in TS24
    B, BURY; B, DISCRIM;
   create branch instruction in TS6  (h)
    opcode → TS9
     Q, 0010, 0000, 0000, 0000; TS6–TS2;
     ADD 'A'; ROTATE 16; TS4–TS9;
    address → TS10 TS24–TS10
    OR;
    TS8–TS6
    branch via TS6 B, 1;
 x × sum2 → sum1 CALPOL 37.
  x → t
   TS13–TS18; TS14–TS19;
  sum2 → sum1
   B, BURY; B, TRANS45;
  t × sum1 → sum2
   B, BURY; B, MULTIP;
  sum2 → sum1
   B, BURY; B, TRANS45;
 B, CALPOL 8;
sum2 → poly CALPOL 50.
 TS22–TS25; TS23–TS26;
B, UNBURY;
```

Fig. 6 Turing's program structured

one cycle. Another neat idea was that of replacing op-codes entirely by transfers into functional destinations; for example, anything moved to destination 17 was added to TS16. Turing's complicated conditional branch mechanism was replaced by two conditional skip instructions. A number of the changes were, in fact, due to Turing himself (Wilkinson, 1975). All the same, Pilot ACE retained one essential characteristic of Turing's proposal. This was the distributed internal nature of the processor: an array of registers each associated with specific functions of the machine.

Nowadays we can distinguish two levels of computer architecture which correspond neatly to von Neumann's and Turing's 1945 proposals. These are the central processor which executes complete operations and the microprocessor designed to emulate other machines. Often a machine of Turing's type emulates a von Neumann machine just as Turing proposed in 1945.

It would be pleasing to think that these two extremes have both been with us continuously from the beginning, but this does not appear to be the case. The Pilot ACE was followed by the full ACE, the DEUCE, MOSAIC, and the successful Bendix G15 but the line seems to have ended there (Bell and Newell, 1971). An inspection of the early literature reveals that Turing's report was referred to little if at all (we have not found a direct reference before 1969). Early books on computers, although referencing Turing in regard to artificial intelligence, do not cite him as a computer designer. Of course, the 1945 proposal for ACE was, strictly speaking, an unpublished document but so too was the draft report on EDVAC which was widely acknowledged.

We do not believe that it can be inferred that Turing had no influence on the course of practical computing. That NPL was working on the design of a computer was well publicised at the time and, presumably, Turing's plans were discussed, at least in general terms, by others interested in computing in Britain and also the United States. As for Turing's detailed plan, some of the pioneers read it and some did not; those who did showed a tendency to regard it as too complicated. Goldstine, in his book (p. 218), recounts how D. R. Hartree (from Cambridge) brought a copy of the ACE proposal to the US in 1946 and comments—'The logical complexity of the ACE is not surprising since Turing had a preference for this type of activity to engineering. The type of complexity Turing proposed, while attractive in some respects, did not in the long run flourish and selection weeded it out'.

One can readily understand this dismissal of Turing's proposal. Even with the hindsight of thirty years it makes difficult reading, for he was describing a complete vision of how computers would be. We can now read his work sympathetically because much of his dream has been realised, but at the time it must have seemed quite far fetched and impractical in parts. On top of this Turing was an intruder, a mathematician in an engineers' domain, proposing a different approach to computing than that arising from the successful development of electronic machines in the United States.

Although his work did not have the immediate impact of von Neumann's, most of Turing's ideas have resurfaced at one time or another since 1945. As a source of ideas and inspiration Turing must have had a very significant, though indirect, influence on practical computing: a contribution to progress perhaps more important than his work on artificial intelligence and his founding of computability theory.

Acknowledgements

We would like to thank M. Woodger, Professors A. D. Booth and Brian Randell for helpful criticism and suggestions regarding this paper, Professor M. V. Wilkes and Dr. S. H. Lavington for informative letters, and National Physical Laboratory for permission to reproduce sections of Turing's proposal and for providing us with copies of reports on ACE.

References

BURKS, A. W., GOLDSTINE, H. H., and VON NEUMANN, J. (1946). Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, Report to the Ordnance Department, US Army. Reprinted in *John von Neumann—Collected Works*.

BELL, C. G., and NEWELL, A. (1971). *Computer Structures*, McGraw Hill.

ECKERT, J. P. (1944). Unpublished paper communicated by J. W. Mauchly to the Conference on the History of Computing, Los Alamos, June 1976.

GOLDSTINE, H. H., and VON NEUMANN, J. (1946). On the Principles of Large Scale Computing Machines, Unpublished lecture notes. Included in *John von Neumann—Collected Works*.

GOLDSTINE, H. H. (1972). *The Computer from Pascal to von Neumann*, Princeton University Press.

HARTREE, D. R. (1949). *Calculating Instruments and Machines*, University of Illinois Press.

HUSKEY, H. D. (1948). *News*—Electronic Digital Computing in England, Math Tables III, No. 23, p. 213.

KNUTH, D. E. (1970). Von Neumann's First Computer Program, *Computing Surveys*, Vol. 2, No. 4, pp. 247-260.

LEWIS, S. H. (1956). The G-15 Digital Computer, *Instruments and Automation*, Vol. 29, pp. 1773-1779.

MALIK, R. (1969). In the Beginning—Early Days with ACE, *Data Systems*, pp. 56-59, 62.

MCCULLOCH, W. L., and PITTS, W. (1943). A logical calculus of the ideas immanent in nervous activity, *Bull. of Math. Biophysics*, Vol. 5, pp. 115-133.

RANDELL, B. (1972). On Alan Turing and the Origins of Digital Computers, *Machine Intelligence* 7.

RANDELL, B. (1973). *The Origins of Digital Computers—Selected Papers*, Springer Verlag.

RANDELL, B. (1976). The Collossus, Conference on History of Computing, Los Alamos, June 1976.

TURING, A. M. (1936). On Computable Numbers with an Application to the Entscheidungs problem, *Proc. London Math. Soc.*, Vol. 2, No. 42, pp. 230-267.

TURING, A. M. (1945). Proposals for Development in the Mathematics Division of an Automatic Computing Engine (ACE), Report E882. Executive Committee, NPL, (Reprinted April 1972, with a foreword by D. W. Davies as NPL report, *Com. Sci.* 57).

TURING, SARAH (1959). *Alan M. Turing*, Heffer, Cambridge.

VON NEUMANN, J. (1945). First draft of a report on the EDVAC (June 30, 1945), Contract No. W-670-ORD-4926. Moore School of Electrical Engineering, University of Pennsylvania (extracts included in *The Origins of Digital Computers*, B. Randell).

WILKINSON, J. H. (1951). Report on the Pilot Model of the Automatic Computing Engine—Part II—The Logical Design of the Pilot Model, Mathematics Division and Electronics Section, NPL.

WILKINSON, J. H. (1975). The Pilot ACE at the National Physical Laboratory. *Radio and Electronic Engineer*, Vol. 45, pp. 336-340.

WILKINSON, J. H. (1948). Progress Report on the Automatic Computing Engine, *Report MA/17/1024*, Mathematics Division, NPL.

WILKINSON, J. H. (1954). The Pilot Ace in *Automatic Digital Computation*, Proceedings of a Symposium held at NPL March 25-29, 1953, (Reprinted in *Computer Strucures* by Bell and Newell).

WOODGER, M. (1951). Automatic Computing Engine of the National Physical Laboratory, *Nature*, Vol. 167, p. 270.

WOODGER, M. (1969). Article on Pilot ACE, *Computer Weekly*, April 17th 1969, p. 8.

WOODGER, M. (1958). The History and Present Use of Digital Computers at the National Physical Laboratory, *Process Control and Automation*, November 1958.