Excerpt from: Programmer's Handbook for the Manchester Electronic Computer Mark II

(Printed ca. March, 1951.)

Local Programming Methods and Conventions

(Paper read at the Inaugural Conference for the Manchester University Computer, July 1951.)

Toby Howard describes —

TURING'S CONTRIBUTIONS TO THE EARLY MANCHESTER COMPUTERS

1. Introduction

In 1948, the world's first stored program digital computer was designed and built at The University of Manchester. It was a volatile assembly of cathode ray tubes, vacuum tubes (thermionic valves), resistors and capacitors mounted on huge metal frames. Its official name was The Small-Scale Experimental Machine, but it soon became known as the Baby.

Much has been written about the extraordinary early days in Manchester. My intention in this short commentary is to focus on the specific contributions made by Alan Turing, and give a flavour of what computer programming was like in those early days. I'll look at two of Turing's Manchester publications, both from 1951: his talk 'Local Programming Methods and Conventions' at the Inaugural Computing Conference in Manchester, and his 'Programmers' Handbook for the Manchester Electronic Computer Mark II (what we now call the Ferranti Mark 1)'. Both of these writings contain very low-level detail, much of which is of interest only to specialist historians of computing. My aim is to place Turing's Manchester work in context, and paint the bigger picture. For readers wishing to follow up and discover more detail, I suggest some ideas for further reading.

2. The Baby

A CRT store capable of holding 2048 bits had been demonstrated in December 1947, and the Baby was built to find out if the store could function reliably as the memory of a practical computing machine doing realistic work.

In a University building on Oxford Road, which still stands today, Freddie Williams, Tom Kilburn and Geoff Tootill gave the machine shape. At the time there was little distinction between understanding how the machine worked, and how to make the machine do something useful. Programming as a discipline, like the computer itself, was in its infancy.

Although it worked, at least for short periods, the Baby was sometimes unstable. It was an experiment, after all, and it faltered when the trams thundered down Oxford Road, their electrical systems upsetting the Baby's delicate electronics.

Alas, no photographs of the Baby survive, but its successor, the Manchester Mark 1, from 1949, gives an idea of the 'look' of the machine:

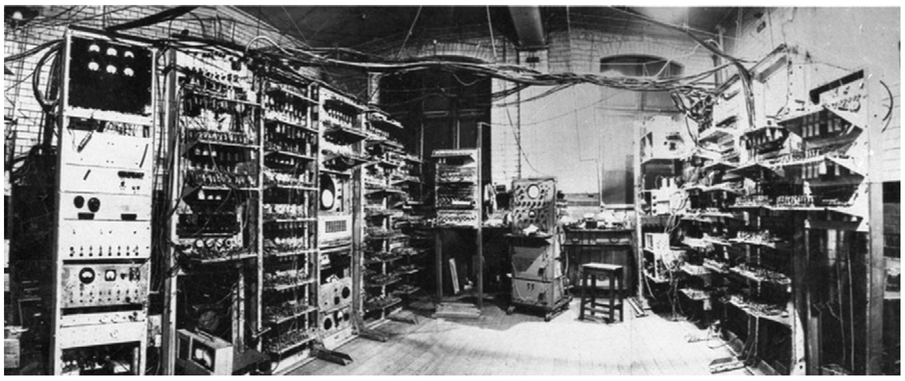


Fig. 1: The Manchester Mark 1 (1949).

Viewed from today's technical perspective, the design of the Baby is familiar, although on a vastly reduced scale. Like many modern computers, it manipulated its data in chunks of 32-bit 'words', and represented signed integers using the '2's complement' convention. It had a Random Access Memory (RAM) of 32 words, making a total of 1024 bits, or 0.128 kilobytes (kb). A modern laptop typically has 4 GB of RAM, about 32 million times more than the Baby.

Whilst modern computers have over 100 different types of instruction, the Baby had just 7. Three things could be manipulated: the 'accumulator', A, where the result of the last computation was stored; the contents of the word in RAM with address S; and the memory address of the current program instruction being executed, CI. The instruction set was as follows (the notation $P \leftarrow Q$ means 'P is given the value of Q'):

1. $A \leftarrow -S$
2. $A \leftarrow A - S$
3. $S \leftarrow A$
4. IF ($A < 0$) THEN ($CI \leftarrow CI + 1$)
5. $CI \leftarrow S$
6. $CI \leftarrow CI + S$
7. HALT

Each of the Baby's storage units was a CRT, one for each of A, CI and the RAM. A fourth CRT, called the 'display tube', could be assigned to view the contents of any of the other tubes. The photograph below (taken from the Ferranti Mark 1) shows how the CRT RAM display looked (bright dots are 1s, dim dots are 0s)[1].

[1] See `http://www.computer50.org/kgill/williams/display.html`.

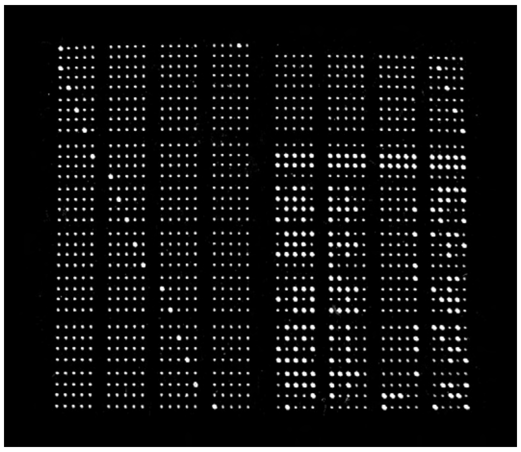


Fig. 2: CRT RAM display from Ferranti Mark 1.

The Baby ran its first program on 21 June 1948. Written by Tom Kilburn, the program took an integer N, and computed its highest factor – the largest integer that would divide into N with no remainder. After a few weeks the program was run on much larger numbers, and as reported in *Nature*, the program took 52 minutes to find the highest factor of $N = 218$, which involved 2.1 million instruction executions, and 3.5 million store accesses (Williams and Kilburn, 1948).

At this time, Turing was at Cambridge, on sabbatical from his post at the National Physical Laboratory, where he had produced a design for an as-yet unbuilt computer called the Automatic Computing Engine (ACE). Its memory would be a mercury acoustic delay line – a curious technology based on a tube several feet long containing liquid mercury. A voltage representing a binary 0 or 1 was converted to an acoustic "ping", and sent down the tube as a sound wave. The ping was detected at the other end of the tube, converted back to a 0 or 1 voltage, and then fed back to the start of the tube to repeat the process. With appropriate timing circuitry, this acoustic loop could store hundreds of bits of binary data.

The building of ACE had stalled, and in May 1948 Turing, frustrated, accepted the offer of a post at The University of Manchester. In July 1948, on hearing that the Baby was working, he wrote a program for long division and sent it up to Manchester. He subsequently sent another program for factorising numbers. Turing arrived in Manchester in October 1948, appointed to the Mathematics Department as Deputy Director of the Royal Society Computing Laboratory.

3. The early Manchester machines

The pioneering work at Manchester attracted the attention of the Ministry of Supply, who saw the potential for Britain to take the lead in the development of computing, and in 1948 commissioned Ferranti Ltd to collaborate with the University. The project now became focused on developing a general purpose computing machine. By the autumn of 1949 the Manchester Mark 1 had been completed, expanding on the prototype ideas of the Baby, by including more memory, and what was called a two-level' store comprising the CRT memory used in the Baby, and a magnetic drum which was slower to read and write, but much larger in capacity. This was the first time a combination of small/fast and larger/slower stores had been used, and it remains the system of fast RAM/slower big disk in most computers today.

Turing made several contributions to the development of the Manchester Mark 1. In the summer of 1949, working with Dai Edwards, he attached a paper tape reader to the machine, so programmers need no longer key in their programs manually on the input switches. He also wrote what we would today call the 'driver software' for the tape reader. Working with Geoff Tootill, Turing designed hardware for generating Random Numbers. In October 1949 Cicely Popplewell was appointed Turing's assistant, and shortly afterwards he took on Audrey Bates, the first of two M.Sc. students he was to supervise. The second, in 1953, was Bernard Richards, working on Morphogenesis.

4. The Manchester University Inaugural Conference

"Since the instructions are held in the machine in a form intelligible to the programmer it is very easy to follow what is going on by looking at the monitor tubes"
– Turing at the Inaugural Conference

The University of Manchester hosted a large conference (9–15 July 1951) to demonstrate the production Ferranti Mark 1, which attracted 170 international delegates. It had been 3 years since the Baby ran its first program.

Turing (1951) gave a short talk entitled 'Local Programming Methods and Conventions', in which he described a scheme for writing down the instructions used to program the Mark 1, in a convenient way for programmers. Discussing the use of a code to represent binary numbers, he said: 'The choice made at Manchester was the scale of 32. It is probable that a scale of 8 or 16 would be more convenient'. He was right. In later years, the scales of 8 (octal) and 16 (hexadecimal) would become the norm.

Today the usual convention is to write binary numbers with the most significant (or 'highest') bit on the left, so we would write decimal 5 as binary 00101. But in Turing's scheme, the most significant bit was on the right, so decimal 5 would be binary 10100. Turing had used 5-hole teleprinter tape at Bletchley Park, and adopted that encoding for representing the bit patterns in the Mark 1's instructions and data, where a teleprinter character would stand for a group of 5 bits, according to the International Telegraphy Alphabet. Using the example above, binary 10100 would be represented by the character with that code, in this case 'S'. The full table of bit/letter assignments was as follows:

0	00000	/	8	00010	$\frac{1}{2}$	16	00001	T	24	00011	O
1	10000	E	9	10010	D	17	10001	Z	25	10011	B
2	01000	@	10	01010	R	18	01001	L	26	01011	G
3	11000	A	11	11010	J	19	11001	W	27	11011	"
4	00100	:	12	00110	N	20	00101	H	28	00111	M
5	10100	S	13	10110	F	21	10101	Y	29	10111	X
6	01100	I	14	01110	C	22	01101	P	30	01111	V
7	11100	U	15	11110	K	23	11101	Q	31	11111	£

The instructions in the Mark 1 were 20 bits in length, so to represent the instruction '01100010100000101110' the programmer would split it into four groups of 5 bits, and use the teleprinter code, to give 'IRTC'. These four characters could then be input into the machine using the console switches, or paper tape.

Turing believed that programmers should follow the execution of their programs by watching the pattern of dots on the CRT screen. He called this 'peeping', and argued, somewhat eccentrically, that programmers should practise the art until they could debug their programs simply by watching for unexpected behaviour among the dancing bright and dim spots.

Turing's conference talk is all about notational conventions, and is, frankly, rather dull. He is discussing a low-level encoding that is relevant only to the Manchester machines. But despite the parochial detail, Turing was addressing a universal problem: how we can use symbols to help manage complexity.

5. The programmers' handbook for the Ferranti Mark 1

"If it is desired to give a definition of programming, one might say that it is an activity by which a digital computer is made to do a man's will, by expressing this will suitably on punched tapes, or whatever other input medium is accepted by the machine." – Programmers' Handbook, p. 50.

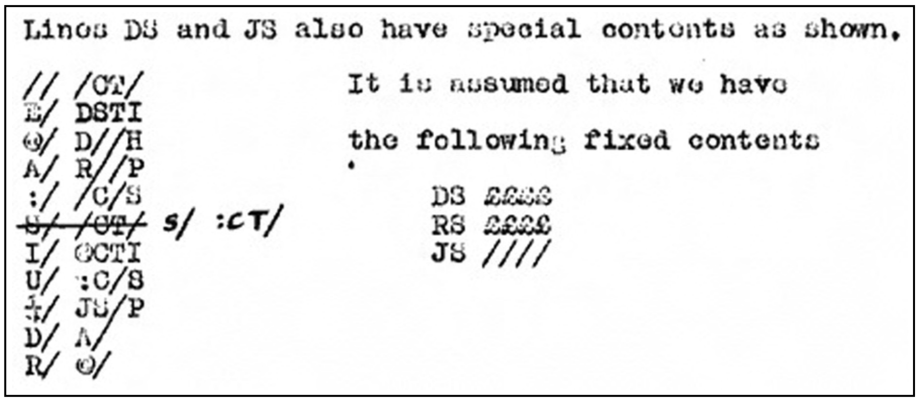
Following the Baby came the improved version, known as the Manchester University Mark 1 or MADM (Manchester Digital Machine). The subsequent fully engineered version of MADM was produced by Ferranti and became known as the Ferranti Mark 1. During this period of intense activity, Turing contributed to the addition of new instructions. The production version of the Ferranti Mark 1, the world's first commercially available general purpose digital computer, was delivered to the University in February 1951. It was for this machine that Turing wrote the programming handbook (confusingly for us, the book is actually entitled 'Programmers' Handbook for the Manchester Electronic Computer Mark II', that being Turing's name for the machine we now refer to as the Ferranti Mark 1) (Facsimile Scan , 1951; Thau, 2000). The first two models to be manufactured were sold to the Universities of Manchester and Toronto. Subsequently the design was slightly altered, and Ferranti sold seven more of the 'Mark 1 star' version of the machine.

An August 1952 Ferranti sales brochure for 'The Manchester Universal Electronic Computer' (i.e., the Ferranti Mark 1) trumpeted the selling points of the machine: '[it] can perform all the operations of arithmetic exceedingly rapidly', '[it] can remember a great many numbers' and '[it] can make decisions'. The machine comprised 'about 4000 valves, 2500 capacitors, 15,000 resistors, 100,000 soldered joints and 6 miles of wire' (Ferranti Sales Brochure, 1952).

What was it actually like to write programs for the Mark 1? First, the programmer had to decide how to organise the program so that it could be correctly loaded into the computer. Turing, together with Cicely Popplewell, devised a system called 'Scheme A', which was a set of conventions and library subroutines, for moving the data used by the program between the fast CRT store and the relatively slow magnetic drum store. Scheme A also dictated conventions for calling subroutines, and inputting programs. Today we would refer to this as 'system software' – providing a standard infrastructure to be used by programmers.

Turing's 90-page handbook is a guide for programmers written by a mathematician. He alternates between explanatory passages, and dense presentations of technical detail. Programming is described at the raw, machine code level – at the time there was no concept of a 'high-level' language. Below, for example, is a program from the handbook[2] that multiplies two numbers together by repeated additions:

[2] See `http://www.alanturing.net/turing_archive/archive/m/m01/M01-015.html`



```
Lines DS and JS also have special contents as shown.

// /CT/                It is assumed that we have
E/ DSTI
@/ D//H                the following fixed contents
A/ R//P                .
:/ /C/S                      DS ££££
~~S/ /CT/~~ S/ :CT/          RS ££££
I/ @CTI                      JS ////
U/ :C/S
½/ JS/P
D/ A/
R/ @/
```

Fig. 3: A program from Turing's "Programmer's Handbook".

Here we can see the binary program instructions represented using the base-32 teleprinter code described above. There are no mnemonics, as used in assembly-level languages. Instead, the programmer really is working in the binary of the machine's instruction set. To read and write even moderately complex programs would take considerable effort.

Programmers today would probably find this scheme impossibly slow and laborious, but such comparisons are rarely helpful. At the time it was simply the only way to program, and Ferranti employee Olaf Chedzoy, for example, has written of his fond memories of programming the Mark 1 (Chedzoy, 1952).

6. Conclusions

Turing's hands-on contributions to the development of the Manchester computers lasted from October 1948 to October 1951. For the next, and sadly final, two and a half years of his life, Turing was programming the Manchester machine to support his biological research. He had become, as we would say today, an 'applications programmer'.

Campbell-Kelly (1980) takes the view that Turing's work on the Mark 1, when compared to such achievements as the Turing Machine (Turing, 1936), and his design of ACE, was 'not an example of his finest work'. His biographer Andrew Hodges makes a plausible case for a number of Turing's novel ideas which were never properly followed up, such as self-organising systems and artificial life, chaos theory, mathematically proving programs correct, developing higher level programming languages, and the lambda calculus which inspired McCarthy's LISP in 1958 (Hodges, 1992; 2004).

Such observations, however, can never detract from the fact that Alan Turing was an extraordinary man – a unique and brilliant thinker. And in 2012 we celebrate his life and achievements.

7. Suggestions for further reading

A primary source for material on the Baby and the early Manchester scene is the Computer50 Web site (Napper, 1998). Simon Lavington's definitive 'A History of Manchester Computers' (Lavington, 1998) is essential reading, and beautifully illustrated with archive photographs. For a detailed coverage of Mark 1 programming see Martin Campbell-Kelly's 'Programming the Mark 1: Early Programming Activity at the University of Manchester' (Campbell-Kelly, 1980). See also the fascinating papers by Chris Burton, Brian Napper and Frank Sumner in 'The First Computers: History and Architecture' (Rojas, 2002), and Mary Croarken's 'The beginnings of the Manchester computer phenomenon: people and influences', which traces the pre-history of the Baby (Croarken, 1993).

8. Acknowledgements

I am indebted to my colleagues at The University of Manchester School of Computer Science for their insights and sense of history. In particular I thank, for their helpful comments on early drafts of this article, Professor Simon Lavington and Professor Steve Furber, and also Chris Burton, who oversaw the creation of an authentic working replica of the Baby for its 50th anniversary in 1998 – now a permanent exhibit at the Manchester Museum of Science and Industry. I give special thanks to Professor Bernard Richards, who generously shared with me his memories of working with Alan Turing. I dedicate this article, in memoriam, to Professor Hilary Kahn and Dr Brian Napper, both passionate custodians of Manchester's unique computing heritage.

References

Campbell-Kelly, M., 1980. Programming the Mark 1: Early programming activity at the University of Manchester. Ann. Hist. Comput. 2(2), 130–168, ISSN 0164-1239.

Chedzoy, O., 1952. Starting work on the world's first electronic computer: 1952 memories. uk.fujitsu.com/pensioner/localData/pdf/public/olaf_art2.pdf.

Croarken, M., 1993. The beginnings of the Manchester computer phenomenon: people and influences. IEEE Ann. Hist. Comput. 15(3), 9–16.

Facsimile scan, 1951. Facsimile scan of the 1st Edition of the 1951 Programming manual at the Turing Digital Archive. www.turingarchive.org/browse.php/B/32.

Ferranti Sales Brochure, 1952. www.computer50.org/kgill/mark1/sale.html.

Hodges, A., 1992. Alan Turing: the Enigma, Vintage, London ISBN 978-0099116417.

Hodges, A., 2004. www.turing.org.uk/philosophy/lausanne1.html and a revised version in Alan Turing: Life and Legacy of a Great Thinker, in: C. Teuscher (Ed.), Springer, Berlin, Heidelberg, ISBN 978-3540200208.

Lavington, S.H., 1998. A History of Manchester Computers, second edn. British Computer Society, London, ISBN 0-902505-01-8.

Napper, R.B.E., 1998. www.computer50.org.

Rojas, P. (Ed.), 2002. The First Computers: History and Architectures, MIT Press, Cambridge, MA, ISBN 9780262681377.

Thau, R.D., 2000. Transcription of Alan Turing's Manual for the Ferranti Mark1, www.computer50.org/kgill/mark1/RobertTau/turing.pdf.

Turing, A.M., 1936. On computable numbers, with an application to the Entscheidungsproblem. In: Proceedings of the London Mathematical Society, vol. 43. pp. 230–265. Available from www.scribd.com/doc/2937039/Alan-M-Turing-On-Computable-Numbers.

Turing, A.M., 1951. Local programming methods and conventions. In: Manchester University Computer: Inaugural Conference, , University of Manchester/Ferranti Ltd. 9–12 July 1951, p. 12.

Williams, F.C., Kilburn, T., 1948. Electronic digital computers. Nature 162(487), reprinted at www.computer50.org/kgill/mark1/natletter.html.

Excerpt from: Programmer's Handbook for the Manchester Electronic Computer Mark II

Programming principles

Programming is a skill best acquired by practice and example rather than from books. The remarks given here are therefore quite inadequate.

If it is desired to give a definition of programming, one might say that it is an activity by which a digital computer is made to do a man's will, by expressing this will suitably on punched tapes, or whatever other input medium is accepted by the machine. This is normally achieved by working up from relatively simple requirements to more complex ones. Thus for instance if it is desired to do a Fourier analysis on the machine it would be as well to arrange first that one can calculate cosines on it. For each process that one wishes the machine to be able to carry out one constructs a 'routine' for the process. This consists mainly of a set of instructions, which are obeyed by the machine in the process. It is not usual however to reckon all the instructions obeyed in the process as part of the 'routine'. Many of them will belong to other routines, previously constructed and carrying out simpler processes. Thus for instance the Fourier analysis process would involve obeying instructions in the routine for forming cosines as well as ones in the analysis routine proper. In a case like this the cosine's routine is described as a 'subroutine' of the analysis routine. The subroutines of any routine may themselves have subroutines. This is like the case of the bigger and lesser fleas. I am not sure of the exact meaning the poet attached to the phrase 'and so on ad infinitum', but am inclined to think that he meant there was no limit that one could assign to a parasitic chain of fleas, rather than that he believed in infinitely long chains. This certainly is the case with subroutines. One always eventually comes down to a routine without subroutines.

What is normally required of a routine is that a certain function of the state of the machine shall be calculated and stored in a given place, the majority of the content of the store being unaffected by the process, and the routine not being dependent on this part having any particular content. It is usual also for the other part of the store to be divided into a part which is altered in the process but not greatly restricted as to its original content, and a part which is unaltered in its content throughout, and such that the correct working of the routine depends on this part having a particular content. The former can be described as 'the working space for the routine' and the latter as 'the space occupied by the routine'.

Applying this to the Mark II machine, the routines usually 'occupy' various tracks of the magnetic store. The working space includes all the electronic store, with the exception of PERM[1], which can equally well be reckoned as working space which is never really altered, or as a part common to all routines. The first two pages of the electronic store are also somewhat exceptional, if normal conventions are used, as they are only altered when one is copying new routines from the magnetic store onto them.

[1] System code that was permanently kept in the electronic store; the space occupied was not therefore generally accessible to the user.

These definitions do not really help the beginner. Something more specific is needed. I describe below the principal steps which I use in programming, in the hope they will be of some small assistance.

i) Make a plan. This rather baffling piece of advice is often offered in identical words to the beginner in chess. Likewise the writer of a short story is advised to 'think of a plot' or an inventor to 'have an idea'. These things are not the kind that we try to make rules about. In this case however some assistance can be given, by describing the decisions that go to make up the plan.

a) If it is a genuine numerical computation that is involved (rather than, e.g. the solution of a puzzle) one must decide what mathematical formulae are to be used. For example if one were calculating the Bessel function $J_0(x)$ one would have, amongst others, the alternatives of using the power series in x, various other power series with other origins, interpolation from a table, various definite integrals, integration of the differential equation by small arcs and asymptotic formulae. It may be necessary to give some small consideration to a number of the alternative methods.

b) Some idea should be formed as to the supply and demand of the various factors involved. A balance must always be struck between the following incompatible desires:

To carry the process through as fast as possible
To use as little storage space as possible
To finish the programming as quickly as possible
To achieve the maximum possible accuracy

We may express this by saying that machine time, storage space, programmer's time and inaccuracy of results all cost something. The plan should take this into account to some extent, though a true optimum cannot be achieved except by chance, since programmer's time is involved, so that a determination of the optimum would defeat its own ends. The 'state of the market' for these economic factors will vary greatly from problem to problem. For instance there will be an enormous proportion of problems (40% perhaps) where there is no question of using the whole storage capacity of the machine, so that space is almost free. With other types of problem one could easily use ten million digits of storage and still not be satisfied.

The space shortage applies mainly to working space rather than to space occupied by the routines. Since these usually have to be written down by someone this in itself has a limiting effect. Speed will usually be a factor worth consideration, though there are many 'fiddling' jobs where it is almost irrelevant. For instance the calculation of tabular values for functions, which are to be stored in the machine and later used for interpolation, would usually be in this class. Programmer's time will usually be the main factor in special jobs, but is relatively unimportant in fundamental routines which are used in most jobs. Accuracy may compete with machine time, e.g. over such questions as the number of terms to be taken in a series, and with space over the question as to whether 20 or 40 digits of a number should be stored.

c) The available storage space must be apportioned to various duties. This will apply both to magnetic and electronic storage. The magnetic storage will probably be mainly either working space or unused. It should be possible to estimate the space occupied by instructions to within say two tracks, for a large part will probably be previously constructed programmes, occupying a known number of tracks. The quantities to be held in the working space should if possible be arranged in packets which it is convenient to use all at once, and which can be packed into a track or a half-track or quarter-track. For instance when multiplying matrices it might be convenient to partition the matrices into four-rowed or eight-rowed square matrices and keep each either in a track or a quarter-track. The apportionment of the electronic store is partly ruled by the conventions we have introduced, but

there is still a good deal of freedom, e.g. if eight pages are available then pages 4, 5, 6 can be used for systematic working space and may be used for various different purposes that require systematic working space.
The beginner will do well to ask for advice concerning plans. Bad plans lead to programmes being thrown away, wasting valuable programmer's time.

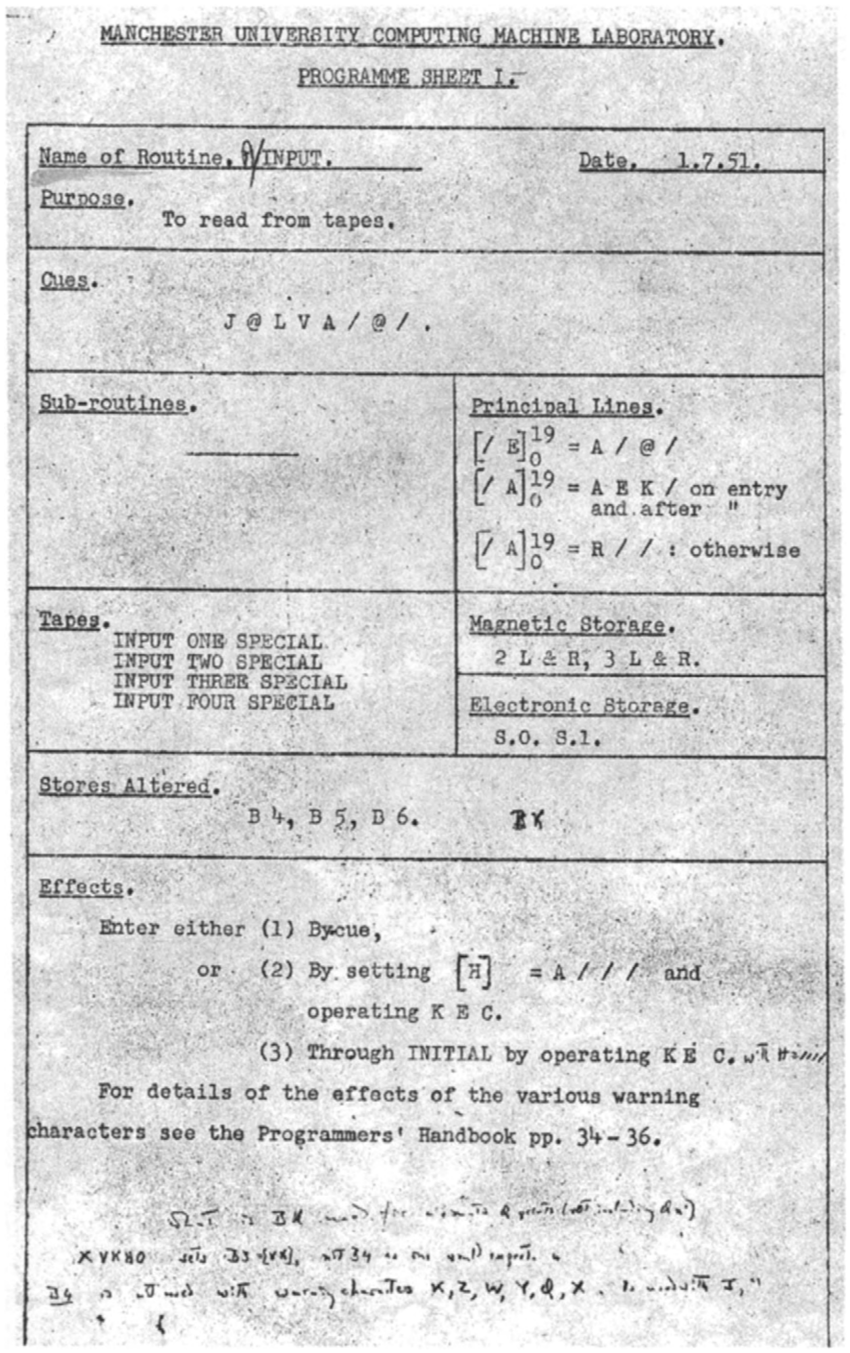
d) If questions of time are at all critical the 'plan' should include a little detailed programming, i.e. the writing down of a few instructions. It should be fairly evident which operations are likely to consume most of the time, and help decide whether the plan as a whole is satisfactory. Very often the 'omission of counting method' should be applied.
e) If one cannot think of any way, good or bad, for doing a job, it is a good thing to try and think how one would do it oneself with pencil and paper. If one can think of such a method it can usually be translated into a method which could be applied to the machine.

(ii) Break the problem down. This in effect means to decide which parts of the problem should be made into definite subroutines. The purpose of this is partly to make the problem easier to think about, the remaining work consisting of a number of 'limited objective' problems. Another reason for breaking down is to facilitate the solution of other problems by the provision of useful subroutines. For instance if the problem on hand were the calculation of Bessel functions and it had already been decided to use the differential equation, it might be wise to make and use a subroutine for the solution of linear second order differential equations. This subroutine would in general be used in connection with other subroutines which calculate the coefficients of the equation.

(iii) Do the programming of the new subroutines. It is better to do the programming of the subroutines before that of the main routine, because there will be a number of details which will only be known after the subroutine has been made, e.g. scale factors applied to the results, number of pages occupied by the subroutines, etc. It also frequently happens in the making of the subroutine that some relatively small change in its proposed properties is desirable. Changes of these details may put the main routine seriously out if it were made first. There is a danger that this procedure may result in one's 'not seeing the wood for the trees', but this should not happen if the original plan was well thought out. The programming of each subroutine can itself be divided into parts:

a) As with programming a whole problem a plan is needed for a subroutine. A convenient aid in this is the 'block schematic diagram'. This consists of a number of operations described in English (or any private notation that the programmer prefers) and joined by arrows. Two arrows may leave a point where a test occurs, or more if a variable control transfer number is used. Notes may also be made showing what is tested, or how many times a loop is to be traversed.
b) The operations appearing as blocks in a) may be replaced by actual instructions. It is usually not worth while at first to write down more than the last two characters of the (presumptive) instruction, i.e. the B line and function parts. These are quite enough to remind one of what was the purpose of the instruction.
c) One may then write the instructions into a page, deciding at the same time what are to be the addresses involved.
d) When the programming is complete, check sheets must be made. It is often advisable to start making check sheets long before the programme is complete; one should in fact begin them as soon as one feels that one has got into a muddle. It is often possible to work out most of the programme on the check sheets and afterwards transfer back onto the page or pages of instructions.

(iv) Programme the main routine. This follows principles similar to **(iii)**. Of course these remarks merely represent one possible way of doing programming. Individuals will no doubt vary as to the methods they prefer.



MANCHESTER UNIVERSITY COMPUTING MACHINE LABORATORY.

PROGRAMME SHEET I.

Name of Routine. ~~R~~ INPUT. Date. 1.7.51.

Purpose.
To read from tapes.

Cues.
J @ L V A / @ / .

Sub-routines.
____

Principal Lines.
$\left[/\ E\right]_0^{19}$ = A / @ /
$\left[/\ A\right]_0^{19}$ = A E K / on entry and after "
$\left[/\ A\right]_0^{19}$ = R / / : otherwise

Tapes.
INPUT ONE SPECIAL
INPUT TWO SPECIAL
INPUT THREE SPECIAL
INPUT FOUR SPECIAL

Magnetic Storage.
2 L & R, 3 L & R.

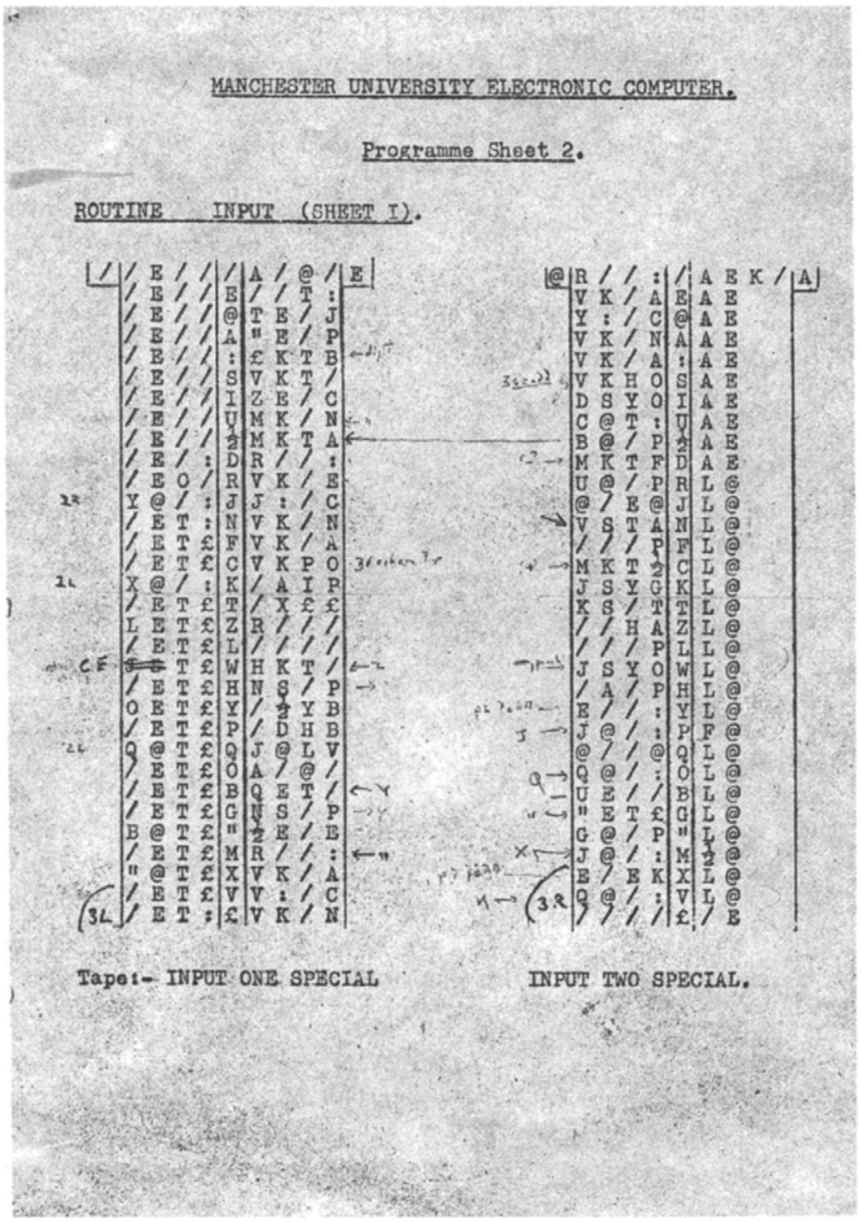
Electronic Storage.
S.0. S.1.

Stores Altered.
B 4, B 5, B 6. BK

Effects.
Enter either (1) By cue,
or (2) By setting [H] = A / / / and operating K E C.
(3) Through INITIAL by operating K E C. with [illegible]
For details of the effects of the various warning characters see the Programmers' Handbook pp. 34 - 36.

[illegible] in BK [illegible] for [illegible] (not including [illegible])
X V K 8 O sets B3 = [VK], and 34 [illegible] input.
B4 is used with warning characters K, Z, W, Y, Q, X [illegible] I, "

Fig. 1: Reproduction of Turing's INPUT routine, which was used to read programs into the machine from punched paper-tape.



MANCHESTER UNIVERSITY ELECTRONIC COMPUTER.

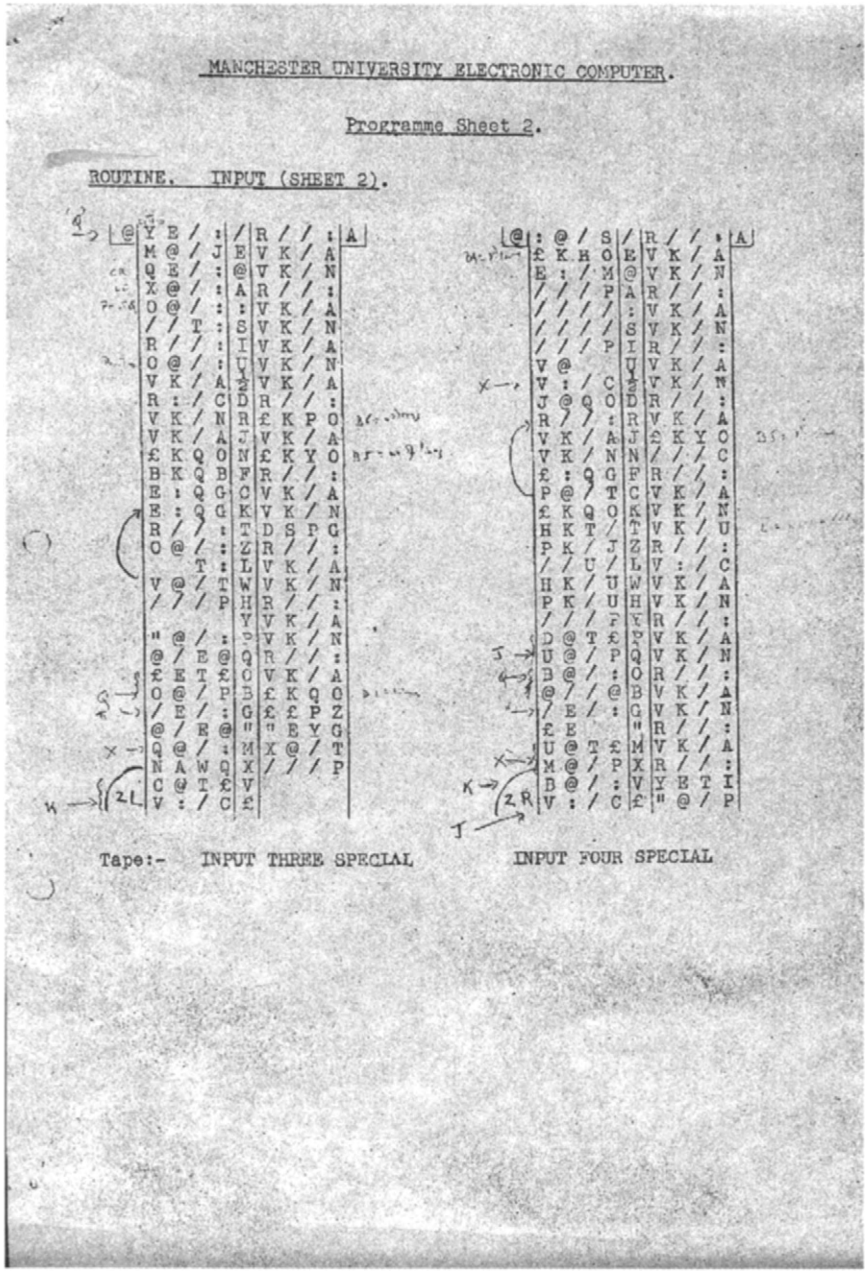
Programme Sheet 2.

ROUTINE INPUT (SHEET I).

```
    | /|/ E / / | / | A / @ / | E |         | @|R / / : | / | A E K / | A |
       / E / / | E | / / T :                    V K / A | E | A E
       / E / / | @ | T E / J                    Y : / C | @ | A E
       / E / / | A | " E / P                    V K / N | A | A E
       / E / / | : | £ K T B                    V K / A | : | A E
       / E / / | S | V K T /                    V K H O | S | A E
       / E / / | I | Z E / C                    D S Y O | I | A E
       / E / / | U | M K / N                    C @ T : | U | A E
       / E / / | ½ | M K T A                    B @ / P | ½ | A E
       / E / : | D | R / / :                    M K T F | D | A E
       / E O / | R | V K / E                    U @ / P | R | L @
 22    Y @ / : | J | J : / C                    @ / E @ | J | L @
       / E T : | N | V K / N                    V S T A | N | L @
       / E T £ | F | V K / A                    / / / P | F | L @
       / E T £ | C | V K P O                    M K T ½ | C | L @
 2L    X @ / : | K | / A I P                    J S Y G | K | L @
       / E T £ | T | / X £ £                    K S / T | T | L @
       L E T £ | Z | R / / /                    / / H A | Z | L @
       / E T £ | L | / / / /                    / / / P | L | L @
 CE    ~~/ E~~ T £ | W | H K T /                J S Y O | W | L @
       / E T £ | H | N S / P                    / A / P | H | L @
       O E T £ | Y | / ½ Y B                    E / / : | Y | L @
       / E T £ | P | / D H B                    J @ / : | P | F @
 2L    Q @ T £ | Q | J @ L V                    @ / / @ | Q | L @
       / E T £ | O | A / @ /                    Q @ / : | O | L @
       / E T £ | B | Q E T /                    U E / / | B | L @
       / E T £ | G | N S / P                    " E T £ | G | L @
       B @ T £ | " | ½ E / E                    G @ / P | " | L @
       / E T £ | M | R / / :                    J @ / : | M | ½ @
       " @ T £ | X | V K / A                    E / E K | X | L @
       / E T £ | V | V : / C                    Q @ / : | V | L @
 3L    / E T : | £ | V K / N              3R    / / / / | £ | / E
```

Tape:- INPUT ONE SPECIAL INPUT TWO SPECIAL.

Fig. 1: (continued)



MANCHESTER UNIVERSITY ELECTRONIC COMPUTER.

Programme Sheet 2.

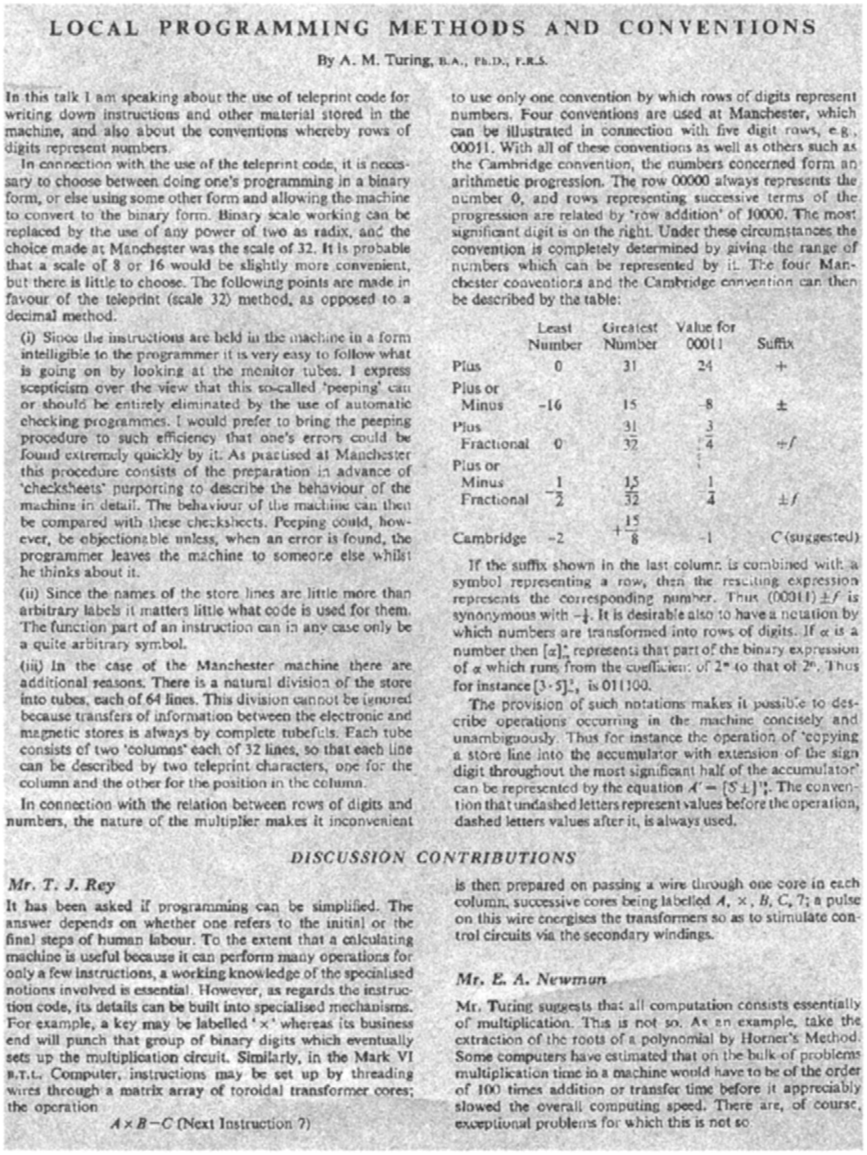
ROUTINE.   INPUT (SHEET 2).

```
      @ | Y E / : | / | R / / : | A
          M @ / J | E | V K / A
          Q E / : | @ | V K / N
          X @ / : | A | R / / :
          O @ / : | : | V K / A
          / / T : | S | V K / N
          R / / : | I | V K / A
          O @ / : | U | V K / N
          V K / A | ½ | V K / A
          R : / C | D | R / / :
          V K / N | R | £ K P O
          V K / A | J | V K / A
          £ K Q O | N | £ K Y O
          B K Q B | F | R / / :
          E : Q G | C | V K / A
          E : Q G | K | V K / N
          R / / : | T | D S P G
          O @ / : | Z | R / / :
              T : | L | V K / A
          V @ / T | W | V K / N
          / / / P | H | R / / :
                  | Y | V K / A
          " @ / : | P | V K / N
          @ / E @ | Q | R / / :
          £ E T £ | O | V K / A
          O @ / P | B | £ K Q O
          / E / : | G | £ £ P Z
          @ / E @ | " | " E Y G
     X →  Q @ / : | M | X @ / T
          N A W Q | X | / / / P
          C @ T £ | V |
K →  2L   V : / C | £ |
```

```
      @ | : @ / S | / | R / / : | A
          £ K H O | E | V K / A
          E : / M | @ | V K / N
          / / / P | A | R / / :
          / / / / | : | V K / A
          / / / / | S | V K / N
          / / / P | I | R / / :
          V @     | U | V K / A
     X →  V : / C | ½ | V K / N
          J @ Q O | D | R / / :
          R / / : | R | V K / A
          V K / A | J | £ K Y O
          V K / N | N | / / / C
          £ : Q G | F | R / / :
          P @ / T | C | V K / A
          £ K Q O | K | V K / N
          H K T / | T | V K / U
          P K / J | Z | R / / :
          / / U / | L | V : / C
          H K / U | W | V K / A
          P K / U | H | V K / N
          / / / F | Y | R / / :
          D @ T £ | P | V K / A
     J →  U @ / P | Q | V K / N
          B @ / : | O | R / / :
          @ / / @ | B | V K / A
          / E / : | G | V K / N
          £ E     | " | R / / :
          U @ T £ | M | V K / A
     X →  M @ / P | X | R / / :
     K →  B @ / : | V | Y E T I
     2R   V : / C | £ | " @ / P
     J
```

Tape:-   INPUT THREE SPECIAL   INPUT FOUR SPECIAL

Fig. 1: (continued)



LOCAL PROGRAMMING METHODS AND CONVENTIONS

By A. M. Turing, B.A., Ph.D., F.R.S.

In this talk I am speaking about the use of teleprint code for writing down instructions and other material stored in the machine, and also about the conventions whereby rows of digits represent numbers.

In connection with the use of the teleprint code, it is necessary to choose between doing one's programming in a binary form, or else using some other form and allowing the machine to convert to the binary form. Binary scale working can be replaced by the use of any power of two as radix, and the choice made at Manchester was the scale of 32. It is probable that a scale of 8 or 16 would be slightly more convenient, but there is little to choose. The following points are made in favour of the teleprint (scale 32) method, as opposed to a decimal method.

(i) Since the instructions are held in the machine in a form intelligible to the programmer it is very easy to follow what is going on by looking at the monitor tubes. I express scepticism over the view that this so-called 'peeping' can or should be entirely eliminated by the use of automatic checking programmes. I would prefer to bring the peeping procedure to such efficiency that one's errors could be found extremely quickly by it. As practised at Manchester this procedure consists of the preparation in advance of 'checksheets' purporting to describe the behaviour of the machine in detail. The behaviour of the machine can then be compared with these checksheets. Peeping could, however, be objectionable unless, when an error is found, the programmer leaves the machine to someone else whilst he thinks about it.

(ii) Since the names of the store lines are little more than arbitrary labels it matters little what code is used for them. The function part of an instruction can in any case only be a quite arbitrary symbol.

(iii) In the case of the Manchester machine there are additional reasons. There is a natural division of the store into tubes, each of 64 lines. This division cannot be ignored because transfers of information between the electronic and magnetic stores is always by complete tubefuls. Each tube consists of two 'columns' each of 32 lines, so that each line can be described by two teleprint characters, one for the column and the other for the position in the column.

In connection with the relation between rows of digits and numbers, the nature of the multiplier makes it inconvenient to use only one convention by which rows of digits represent numbers. Four conventions are used at Manchester, which can be illustrated in connection with five digit rows, e.g., 00011. With all of these conventions as well as others such as the Cambridge convention, the numbers concerned form an arithmetic progression. The row 00000 always represents the number 0, and rows representing successive terms of the progression are related by 'row addition' of 10000. The most significant digit is on the right. Under these circumstances the convention is completely determined by giving the range of numbers which can be represented by it. The four Manchester conventions and the Cambridge convention can then be described by the table:

	Least Number	Greatest Number	Value for 00011	Suffix
Plus	0	31	24	+
Plus or Minus	-16	15	-8	±
Plus Fractional	0	$\frac{31}{32}$	$\frac{3}{4}$	$+f$
Plus or Minus Fractional	$-\frac{1}{2}$	$\frac{15}{32}$	$-\frac{1}{4}$	$\pm f$
Cambridge	-2	$+\frac{15}{8}$	-1	C (suggested)

If the suffix shown in the last column is combined with a symbol representing a row, then the resulting expression represents the corresponding number. Thus $(00011)\pm f$ is synonymous with $-\frac{1}{4}$. It is desirable also to have a notation by which numbers are transformed into rows of digits. If α is a number then $[\alpha]_{n}^{m}$ represents that part of the binary expression of α which runs from the coefficient of 2^{m} to that of 2^{n}. Thus for instance $[3\cdot5]^{2}_{-3}$, is 011100.

The provision of such notations makes it possible to describe operations occurring in the machine concisely and unambiguously. Thus for instance the operation of 'copying a store line into the accumulator with extension of the sign digit throughout the most significant half of the accumulator' can be represented by the equation $A' = [S\pm]''_{;}$. The convention that undashed letters represent values before the operation, dashed letters values after it, is always used.

DISCUSSION CONTRIBUTIONS

Mr. T. J. Rey

It has been asked if programming can be simplified. The answer depends on whether one refers to the initial or the final steps of human labour. To the extent that a calculating machine is useful because it can perform many operations for only a few instructions, a working knowledge of the specialised notions involved is essential. However, as regards the instruction code, its details can be built into specialised mechanisms. For example, a key may be labelled '×' whereas its business end will punch that group of binary digits which eventually sets up the multiplication circuit. Similarly, in the Mark VI B.T.L. Computer, instructions may be set up by threading wires through a matrix array of toroidal transformer cores; the operation

$$A\times B-C \text{ (Next Instruction 7)}$$

is then prepared on passing a wire through one core in each column, successive cores being labelled A, ×, B, C, 7; a pulse on this wire energises the transformers so as to stimulate control circuits via the secondary windings.

Mr. E. A. Newman

Mr. Turing suggests that all computation consists essentially of multiplication. This is not so. As an example, take the extraction of the roots of a polynomial by Horner's Method. Some computers have estimated that on the bulk of problems multiplication time in a machine would have to be of the order of 100 times addition or transfer time before it appreciably slowed the overall computing speed. There are, of course, exceptional problems for which this is not so

Fig. 2: Turing's only formal publication on programming techniques for the Manchester Mark I, read at the Inaugural Conference for the Manchester University Computer in July 1951.