Programs and standard software for the Ferranti Mark I and Mark I* computers.

Contents.

1. System software: Scheme A and Scheme B.

Scheme A was the initial arrangement at Manchester University for providing some basic software facilities for end-users of the Ferranti Mark I. It assumed that only five out of the eight CRTs would be available. Scheme B was a later modification which extended the facilities but otherwise retained the same design-philosophy. We describe the first system below.

In addition to the housekeeping code necessary in Scheme A or Scheme B, a library of standard routines was built up at Manchester. These included:

> **Basic functions**: Division, square root, cube root and nth root.
> **Other functions**: Cosine, exponential, logarithm and arctangent.
> **Algebraic processes**: solution of linear simultaneous equations, inversion of a matrix, solution of algebraic equations and the evaluation of the latent roots and vectors of a matrix.
> **Analytical processes**: quadrature, integration of ordinary differential equations, the solution of integral equations.

Providing a framework for the use of routines and subroutines was one of the tasks of Scheme A. The central section for Scheme A is called **PERM,** which consists of code that is assumed to be permanently available in the primary (CRT) memory during run time. A copy of PERM is held on two tracks of the drum, in a four-track area for which writing is normally inhibited by hardware – (see also below). PERM contains the **Routine Changing Sequence, RCS**. The RCS uses a 40-bit quantity associated with every routine, called the **cue.** The cue holds three parameters:

(a) the drum track-address where a routine is held;
(b) the primary store address to where it is to be brought down;
(c) the line in primary memory at which it is to be entered.

There are *true cues* and *false cues*. A special track on the disc acts as a directory, holding true cues. The false cue for a routine points to the line in the directory allocated to hold the true cue for that routine – thus giving a level of indirection and hence the flexibility to

alter a routine's actual position on the drum whilst leaving its formal position apparently unaltered.

The layout of a cue (true or false) is as follows:

Bits 0 – 9: these give the routine's (entry-address – 1).
Bits 10 – 19: check characters, made up from a function of words /E and /A of the routine. (These are called the *principal lines* of the routine).
Bits 20 – 39: if bit 39 = 0, then this is a true cue and these 10 bits give the control word for bringing the routine down from the drum. If bit 39 = 1, this is a false cue and bits 20 – 29 give the directory line that holds the true cue, and bits 30 – 39 give, when bit 31 is replaced by 0, the number of the track containing the directory.

When calling a routine, the main program (ie the end-user's program) places a **link** in the least-significant half of the accumulator. The link is the return-address, employed when control is passed back to the main program.

Besides PERM, the first four tracks of the drum also hold system routines called INITIAL, ROUGHWRITE and INPUT. Assuming an initially empty primary memory, an operator would boot up the Ferranti Mark I by pressing certain switches on the console that caused the INPUT and PERM routines to be brought down from drum to the primary (CRT) memory. The INPUT routine is then available for reading in other routines, etc., from paper tape.

2. Booting up the computer initially.

A row of 20 hand-switches is provided on the console, allowing a 20-bit binary quantity to be set up in preparation for use with either the /Z or // instruction. See section F1/X3 for a full description of the instruction set. Since the primary memory can be cleared manually and since an instruction of all zeros is interpreted as 'obey the number set on the hand-switches as a magnetic instruction', bootstrapping a program from the drum is a simple matter.

3. A sample Ferranti Mark I program.

Below is a short program that places in address /C the scalar product of two 18-element vectors whose fixed-point values are stored in the following addresses (inclusive):

Vector (i) in lines /N, @N, …, LN
Vector (ii) in lines /F, @F, …, LF

Address	*Machine code*	*Explanation*
//	L///	Number of elements in the vectors
/E	IST/	Entry point; set round-off
/@	//QO	Set B7
/A	/NUK	Add product
/:	/FUF	… to partial sum
/S	A:QG	B7 := B7 – 1 (ie adjust counter)
/I	A:/T	Test for last cycle, jumping back to line A if more to do

/U	/C/A	Transfer result to address /C

Other example programs are given in [ref. 21].

4. Mark I Autocode.

In the period 1952 – 54, two people began to think about how to simplify the writing of programs using primitive forms of what we would now call *High-level languages.* Firstly, Alec Glennie, a Ferranti Mark I user from the government's atomic weapons research team (working firstly at ARDE and then at AWRE) broke new ground. Glennie wrote a simple *autocode* system entirely for his own use in about 1952. Tony Brooker, who had taken over from Alan Turing as chief systems programmer at Manchester in October 1951, then devised a richer scheme which allowed users to write programs in the style of algebraic expressions, with simple ways of implicitly calling standard library subroutines. Brooker's scheme, called Mark I Autocode, was released in March 1954 and was probably the world's first publicly-available *High Level Language.* It was about two years ahead of the first Fortran compiler. See also [ref. 21] in section F1/X5 of the *Our Computer Heritage* website.

Brooker set out to solve three of the common tasks faced by all early programmers of scientific and engineering applications:

- simplifying the written form of arithmetic statements;
- taking care of the scaling of variables (via built-in floating-point routines);
- taking care of the transfer of information between primary and secondary store.

Mark I Autocode used the symbols v1, v2, v3, .. to stand for floating-point variables and n1, n2, n3, … to stand for integer variables. An array of a hundred floating-point numbers could be represented by vn1, where n1 took on the values 1 to 100. Floating-point calculations were automatically performed at run time by interpretive library routines. Other symbols were used for standard library routines for printing, calculating a square root, etc. The symbol j was used to indicate a control transfer (ie jumps or branches in a program) and the lines of a program could be labelled. Thus, the Autocode statement:

J2, 100 ≥ n1

meant: "jump to program line 2 if the user's variable n1 is less than 100".

Below is a simple Mark I Autocode program which calculates the Root Mean Square of one hundred real (ie floating-point) variables v1, v2, v3, etc.

```
  n1 = 1
  v101 = 0
2v102 = vn1 x vn1
  v101 = v101 + v102
  n1 = n1 + 1
  j2, 100 ≥ n1
  v101 = v101/100.0
  *v101 = F1(v101)
```

The symbol * caused printing to ten decimal places on a new line; F1 signified the intrinsic function *square root.*

By 1952, external researchers from government establishments and industry had begun to use the Ferranti Mark I at Manchester University. As would be expected, most applications

were in science and engineering. There were exceptions. From the spring of 1950 Alan Turing started to work on a mathematical theory of embryology, called morphogenensis, which involved the growth and form of living things. Christopher Strachey, who later achieved fame as a theoretical Computer Scientist, made a light-hearted use of the Ferranti Mark I's random number generator in the summer of 1952 to 'create' love-letters. Here is an example:

> *Darling Sweetheart,*
> *You are my avid fellow-feeling. My affection curiously clings to your passionate wish. My liking yearns to your heart. You are my wistful sympathy; my tender liking.*
> *Yours beautifully,*
> *M.U.C.*

An emulator of the Ferranti Mark I that runs Christopher Strachey's Love-letter software has been developed by Dr. David Link:
https://web.archive.org/web/20130704223203/http://alpha60.de/research/muc/
(Snapshot dated 4th July 2013; retrieved on 5th February 2024).

The Ferranti Mark I (FERUT) that was installed at Toronto developed its own version of the basic system software – versions, that is, of Manchester's Scheme A, etc. Toronto University also produced a much more user-friendly programming system than Turing's original machine code based on 'backwards binary' teleprinter symbols. In the autumn of 1953, possibly inspired by John Backus's Speedcode system for the IBM 701, Trixie Worsley from the Toronto Computer Centre and J.N. Patterson Hume from the Physics Department, were asked to create an automatic coding system for FERUT. Their language, called TRANSCODE [Ref. 28], was ready by the end of 1954. TRANSCODE was immediately popular with users. It incorporated meaningful four-letter mnemonic instructions such as ADDN, SUBT, MULT, LOOP, PRNT, etc.

5. Mark I* utilities.

The principal Mark I* software system developed by Ferranti was known as the *Radix 32 Input and Organisation Scheme*. It was based upon "Schemes A and B of the present Manchester University computer, and on the FERUT Input Scheme, but incorporates other ideas as well" [Ref. 29].

For the Ferranti Mark I* at ARDE Fort Halstead, in 1956 John Berry introduced the *Intercode* scheme, described [Ref. 30] as "an interpretive simplified coding scheme, designed for small computations only" … and "inspired chiefly by the Mark I Autocode". It was, however, quite modest in comparison with Brooker's work.

INTINT.

At the Instituto Nazionale per le Applicazioni Calcolo (INAC) in Rome, it is believed that practically all the programming of their Ferranti Mark I* was done in machine code. An exception was a system called INTINT (INTerpretation-INTegration) for handling vectors designed by Corrado Böhm [Ref. 31]

Tabular Interpretive Programme (TIP).

In 1957 the Mathematical Services Group of Bristol Siddeley Engines Ltd., Filton, Bristol, developed what was known informally as a ‘tabular operating system’ but formally as Tabular Interpretive Programme (TIP) – see [ref. 32]. This was inspired by the General Interpretive Programme system (GIP), developed earlier by NPL for the Pilot ACE computer. GIP required only simple codewords to run a collection of programs called "bricks". Each brick could perform a single task, such as solving a set of simultaneous equations, inverting a matrix, and performing matrix multiplication. TIP was implemented for the Ferranti Mark I* and used by the aero engine designers at AVRO and at Armstrong Siddeley Engines Ltd., Anstey.