Further Autocode Facilities for the Manchester (Mercury) Computer

by R. A. Brooker

Summary: This article is a sequel to an earlier paper,* part of which described the basic facilities of the simplified programming system (Autocode) developed for the Ferranti Mercury Computer at Manchester University. The present article describes further facilities of the system and, together with the earlier programming manual for the Manchester Mercury Computer.

UNROUNDED ARITHMETIC

In those arithmetical instructions involving a variable expression on the right-hand side, each sum, difference and product is formed to a maximum precision of 29 binary digits and then rounded off by making the last digit odd. If required, the rounding operation can be omitted by using the \( \infty \) sign instead of \( = \). In this case the result will normally be biased, but if the quantities involved can be expressed in at most 29 significant binary digits, then the \( \infty \) sign provides a means of performing exact arithmetical operations on variables.

Rounding errors will be significant when using the int pt and fr pt instructions. Thus for example:

- If \( x = 3 + 2^{-27} \), then int pt \( (x) \) gives 3 and fr pt \( (x) \) gives \( 2^{-27} \).
- If \( x = 3 - 2^{-27} \), then int pt \( (x) \) gives 2 and fr pt \( (x) \) gives \( 1 - 2^{-27} \).
- If \( x = 3 \) precisely, then int pt \( (x) \) gives 3 and fr pt \( (x) \) gives \( 0.2 \cdot 2^{-26} \).

However, precise whole numbers can only be obtained as a result of unrounded \( (\infty) \) arithmetical operations (which include simple transfers, mod and int pt, but do not include division in any form) on numbers originally read in (see Note 4) as integers.

If the “nearest integer to \( x \)” is intended, it is sufficient to use (say)

\[ y = \text{int pt} \ (x + 0.5) \]

unless \( x \) itself is half an odd integer, in which case the result will again depend critically on rounding errors.

MISCELLANEOUS INSTRUCTIONS

The instructions

\[ i = \max (x_0, n, m) \]
\[ i = \min (x_0, n, m) \]

may be used to determine the index of the maximum (or minimum) element in the range

\[ x_m, x_{m-1}, \ldots, x_0. \]

Here \( i \) denotes any index, \( x_0 \) any first member, and \( n, m \) are indices or whole numbers. If there is no unique maximum (or minimum) element, then the least index is taken. The instruction \( y = \phi_2(n) \) is equivalent to \( y = (\ -\ y) \). Here \( y \) is any variable, and \( n \) stands for any index expression.

STEP-BY-STEP INTEGRATION OF DIFFERENTIAL EQUATIONS

Special facilities are provided for the integration of differential equations. The equations must be written in the form

\[ f_i = \frac{dy_i}{dx} = f_i(x, y_1, y_2, \ldots, y_n) \quad (i = 1, 2, \ldots, n) \]

involving the special variable \( x \), the main variables \( y_i, f_i \) and the index \( n \). In addition the special variable \( h \) is used for the step length, and the main variables \( g_i, h_i, (i = 1, 2, \ldots, n) \) are introduced for “working space.”

The programmer must write a subsequence for calculating the \( f_i \) in terms of \( x \) and the \( y_i \); this must not alter \( y_i, g_i, h_i, n, h, \) or \( x \). The entry should be labelled, and the sequence should terminate with the special† instruction 592, 0. With these arrangements the effect of the instruction

\[ \text{int step} \ (m) \]

(where \( m \) is the entry to the subsequence) is to advance the integration by one step so that the initial and final values of the independent and dependent variables are respectively

\[ x \quad x + h \]
\[ y_i(x) \quad y_i(x + h). \]

The method employed is that of Runge–Kutta, with a truncation error of \( O(h^5) \). However, the truncation error also depends on the higher derivatives of the function, and for this reason the step length may be adjusted between steps if desired. The time per step is \( (10m + 4T) \) msec, where \( T \) is the time (in msec) of the subsequence.

† Mercury programmers will recognise this as an ordinary machine instruction. Subject to certain restrictions it is possible to include such instructions in an Autocode program, but this is a topic beyond the scope of the present article since it involves a detailed knowledge of conventional Mercury programming. The interested reader may refer to the paper by Fotheringham and Roberts on p. 128.
Further Autocode Facilities

Example

Integrate the two equations:

\[ \frac{dy_1}{dx} = y_1^2 - 1.23y_1y_2 + 2.47y_2^2 \]
\[ \frac{dy_2}{dx} = 1.01y_1^2 - 0.84y_1y_2 + 1.59y_2^2 \]

for \( x = 0(0.02)1 \),

with the initial conditions:

(a) \( x = 0 \); \( y_1 = 0 \); \( y_2 = 1 \)
(b) \( x = 0 \); \( y_1 = 1 \); \( y_2 = 0 \).

Program

\begin{align*}
\text{chapter 1} & \quad \text{Notes} \\
f & \rightarrow 2 & \text{main directives} \\
g & \rightarrow 2 & \\
h & \rightarrow 2 & \\
y & \rightarrow 2 & \\
1) & \quad n = 2 \\
h & = 0.02 & \text{sets the number of} \\
x & = 0 & \text{equations, the step} \\
y_1 & = 0 & \text{length and the first} \\
y_2 & = 1 & \text{initial conditions} \\
2) & \quad \text{int step (3)} \\
& \text{newline} & \text{advances the step} \\
& \text{print \((x, 1, 2)\)} & \text{prints with layout} \\
& \text{space} & \\
& \text{print \((y_1, 2, 4)\)} & \\
& \text{print \((y_2, 2, 4)\)} & \\
& \text{jump 2, 0.99} & \text{repeat until } x = 1 \\
& \text{restart} & \text{prepare to read the} \\
& & \text{second set of initial} \\
3) & \quad f_1 = y_1y_1 - 1.23y_1y_2 \\
& & + 2.47y_2y_2 \\
& f_2 = 1.01y_1y_1 - 0.84y_1y_2 \\
& & + 1.59y_2y_2 \\
& = 592, 0 & \text{subsequence to calculate} \\
& \text{close} & f_1, f_2 \\
& \text{across 1/1} & \text{first chapter 0} \\
& \text{close} & \text{starts the calculation} \\
& \text{x = 0} & \text{second chapter 0} \\
& \text{y_1 = 1} & \text{sets second initial} \\
& \text{y_2 = 0} & \text{conditions} \\
& \text{across 2/1} & \text{close} \\
& \text{stop} & \text{third chapter 0} \\
& \text{close} & \text{stops the machine.} \\
\end{align*}

This example introduces the instruction

\text{restart}

which is used to read in further instructions. These will usually be confined to chapter 0, as in this case, and used to re-enter an existing program. The \text{restart} instruction may also be used, however, to read in a fresh program.

MULTI-CHAPTER PROGRAMS

The diagram shows the general layout of a program involving three chapters. Each chapter has its own labelling system, and to jump from one chapter to another an across instruction may be used: for example, one of the form

\text{across 2/3}

meaning "jump to the instruction labelled 2 in chapter 3." Since this is a rather long instruction (100 msec) it is preferable not to include it within an inner loop and, as far as possible, partition into chapters should correspond to distinct parts of the calculation.

"chapter 0" has two special features:

(1) It is entered automatically, at the first instruction, when the \text{close} directive is read.

(2) The directives used are those of the chapter last read, unless chapter 0 is actually headed "chapter 0," in which case the directives must be formally set, either in the usual way or by means of the \text{variables} directive explained in the following section.

VARIABLE DIRECTIVES

The main variables must be defined at the beginning of each chapter either explicitly or, if it is a case of repetition, by means of a directive of the form

\text{variables 2}

meaning "use the directives set in chapter 2"—which of course may have been those used in chapter 1. It may be necessary to appreciate the significance of the variable directives. Thus

\begin{align*}
\text{chapter 1} & \quad \text{allocates} \\
a & \rightarrow 99 & 0-99 \text{ to } a_0-a_{99} \\
b & \rightarrow 99 & 100-199 \text{ to } b_0-b_{99} \\
c & \rightarrow 99 & 200-299 \text{ to } c_0-c_{99} \\
\text{chapter 2} & \quad \text{locations} \\
c & \rightarrow 99 & 0-99 \text{ to } c_0-c_{99} \\
x & \rightarrow 49 & 100-149 \text{ to } x_0-x_{49} \\
y & \rightarrow 49 & 150-199 \text{ to } y_0-y_{49} \\
\end{align*}

Thus the "c's" of chapter 2 are not those of chapter 1; if it is intended that they should be, then the latter directives might be recast as follows:

\begin{align*}
x & \rightarrow 49 \\
y & \rightarrow 49 \\
\pi & \rightarrow 99 \text{ (waste)} \\
c & \rightarrow 99 \\
\end{align*}
Further Autocode Facilities

A further consequence of this scheme is that “$x_{20}$” is identical with $y_0$, “$x_{11}$” with $y_1$, and so on. Such “overlapping” references are sometimes useful.

The Auxiliary Variables

To supplement the 495 working variables there are approximately 10,000 auxiliary variables to which indirect access is possible, the exact number depending on the requirements of the program.* Such access is a time-consuming operation, however, and should only be used when the working variables are insufficient for the problem in hand.

In order to refer to these variables, fourteen new working variables

$$a', b', c', d', e', f', g', h', u', v', w', x', y', z'$$

are introduced. These are essentially similar to the special variables $a, b, \ldots, z$ and indeed may be used as such if desired. It is intended, however, that these variables will be used to designate groups of auxiliary variables. This is done by means of instructions in the program proper (unlike the main variables which are described by means of directives) which set these variables to give the address within the auxiliary stores (0, 1, ..., 10,751) of the first member of an auxiliary group.

Thus, for example, the instructions

$$a' \approx 0 \text{ or } a' \approx 0$$
$$b' \approx 100 \text{ alternatively } b' \approx a' + 100$$
$$x' \approx 200 \text{ or } x' \approx b' + 100$$
$$y' \approx 1,200 \text{ or } y' \approx x' + 1,000$$

designate two groups $a', b'$ of 100 variables, a group $x'$ of 1,000 variables, and a group $y'$ of unlimited extent. In contrast to the main variable directives the quantity on the right-hand side is the starting-point and not the extent of the group, although in both cases the extent is irrelevant if overlapping is allowed. Note that the $\approx$ form of the arithmetical instructions has been employed since the quantities in question are whole numbers.

To specify a particular member of a group (say $x'$) the notation

$$x'_{750} \text{ or } x'_a$$

is employed, where the suffix is essentially a variable (an index cannot be used because there is no limit to the size of a group).

The instruction $\phi_l(x'_{750}, a_1, 10)$ is an example of an auxiliary transfer. It means “transfer 10 consecutive variables from the auxiliary store to the working store, starting at $x'_{750}$ in the former, and at $a_1$ in the latter.” As a result $a_1, a_2, \ldots, a_{10}$ are replaced by $x'_{750}, x'_{751}, \ldots, x'_{760}$. The instruction $\phi_l(x'_{750}, a_1, 10)$ corresponds to the reverse operation, namely a transfer from the working store to the auxiliary store. In these instructions $x'_{750}$

* The number of auxiliary variables is given by $10,751 - 512n$, where $n$ is the highest character number. Up to 3,072 further variables can be obtained subject to certain restrictions on the program. (Details available on request.)

may be replaced by any particular auxiliary variable, e.g. $x'_a, y_0$, and $a_1$ by any main variable, e.g. $a_0$ or $b_{i-1}$.

The third parameter 10 is essentially an index quantity and may be replaced by an index letter.

Examples of auxiliary transfers are

$$\phi_l(x'_0, a_0, 400)$$
$$\phi_l(x'_i, b_i, i)$$
$$\phi_l(x'_{hi}, c_{(i-1)}, i)$$

An exception to the rule that the second parameter must be a main variable is provided by the instruction

$$\phi_l(x'_{750}, a, 1)$$

which replaces the special variable $a$ by $x'_{750}$.

The execution time for a group transfer cannot be given very precisely but is less than

$$\{17/[n/32] + 34 + 0.36n\} \text{ msec.}$$

where $[\ ]$ denotes “integral part of,” and $n$ is the number of variables transferred.

Subchapters

At any point within a chapter it is possible to call in a “subchapter” and subsequently to return to the original chapter at the instruction following the point of departure. This is done by means of a down instruction in the main chapter and an up instruction in the subchapter. For example

down 2/3
calls in chapter 3 as a subchapter and enters it at the instruction labelled 2. When the subchapter has completed its task the single word instruction

up

will return control to the main chapter at the instruction following the original down instruction. Alternatively the up instruction may be used in any chapter reached by means of across instructions from the original subchapter. A subchapter may have its own sub-subchapter but there the regression stops.

The Reset Instruction

If necessary the working variables (i.e. main and special) can be preserved during the operation of the subchapter and subsequently restored on return to the main chapter. This can be done by means of the single-word instruction reset used before the down instruction: the variables are “dumped” on the magnetic drum and subsequently restored by the corresponding up instruction. As a consequence, any results calculated by the subchapter would have to be stored as auxiliary variables if they were to be preserved. However, if subchapters are designed to select from and return to the drum all relevant material, they provide a convenient means of arranging a calculation for possible future use as a “subprogram.” The subchapters can be written without defining the auxiliary variables. Instead the special
Further Autocode Facilities

variables \( a', b', c' \), etc., can be defined in terms of those used in the main chapter, by including appropriate instructions between the reset and down instructions. Since they follow the reset instruction the original values of \( a', b', c' \), etc., will be restored on returning to the main chapter. In this way each “level” of organisation may use its own frame of reference for the auxiliary variables. As an example suppose that both the main chapter and the subchapter employ three groups of auxiliary variables denoted in both cases by \( a', b', c' \), and starting relative to each other as follows:

\[
\begin{align*}
    a'_{\text{(sub)}} &= c'_{\text{(main)}} \\
    b'_{\text{(sub)}} &= a'_{\text{(main)}} \\
    c'_{\text{(sub)}} &= b'_{\text{(main)}} + 50.
\end{align*}
\]

These relations may be represented diagrammatically thus:

```
    main chapter
    \hline
    a' \hspace{1cm} b' \hspace{1cm} c'
    \hline
    sub-chapter
    \hline
    \hspace{1cm} b' \hspace{1cm} c' \hspace{1cm} a'
    \hspace{1cm} \leftarrow 50 \hspace{1cm}
```

The instructions for calling in the subchapter and redefining the auxiliary variables are then as follows:

- reset
  \( \pi \approx a' \)
  \( a' \approx c' \)
  \( c' \approx b' + 50 \)
- down \( ? \)?

The special variable \( \pi \) has been used as a “shunting station” since, in any case, it will be reset on entering the new chapter (\( \pi \) is automatically reset to 3.14159... as a result of the instructions across, down, up, and reset).

Operations with Complex Numbers

Operations with complex numbers written as number pairs are provided by instructions of the form

\[
\begin{align*}
    (u, v) &= (x, y) \\
    (u, v) &= (x, y) + (a, b) \\
    (u, v) &= (x, y) - (a, b) \\
    (u, v) &= (x, y) \ast (a, b) \\
    (u, v) &= (x, y) / (a, b) \\
    (u, v) &= \text{sq rt} (x, y) \quad (v > 0) \\
    (u, v) &= \text{log} (x, y) \quad (\pi > v > -\pi) \\
    (u, v) &= \text{exp} (x, y)
\end{align*}
\]

Further functions may be added. Here \( u, v; x, y; a, b \) denote any variables or (except in the case of \( u, v \)) signed constants.

Examples of instructions in this class are:

\[
\begin{align*}
    (f_{0}, g_{0}) &= (f_{0-1}, g_{0-1}) + (f_{0 + 1}, g_{0 + 1}) \\
    (x, y) &= \text{sq rt} (1, 1) \\
    (a, b) &= \text{log} (-0.5, 6)
\end{align*}
\]

Quickies

Every time one of the functions \( \sqrt{\text{sq rt}} \cos \log \tan \) radius \( \sin \exp \arctan \)

is referred to, 17 msec are spent in transferring the necessary set of instructions (subroutine) from the magnetic drum to the instruction store. Provided there is room, however, they can be included in the chapter itself, and in this case the average execution time is reduced from about 23 msec to 6 msec. Functions treated in this way are known as quickies. All that is necessary is to list them (each preceded by \( \phi \)) in order of preference at the end of the chapter in question, immediately before the close directive (as in the accompanying diagram). Any functions for which there is not room will be treated in the usual way.

As regards the complex functions, these involve the use of certain real functions, as follows:

```
    \begin{align*}
    \text{sq rt} & \quad \text{involves} \quad \text{sq rt} \\
    \exp & \quad \text{involves} \quad \exp, \sin, \cos \\
    \log & \quad \text{involves} \quad \log, \arctan.
    \end{align*}
```

Thus in order to treat the complex functions as quickies, it is sufficient to list the relevant real functions.

Finally, it should be mentioned that \( \sin \) and \( \cos \) involve the same set of instructions, so that if either one is treated as a quicky the other will be also. The same applies to \( \sqrt{\text{sq rt}} \) and \( \text{radius} \).

Notes

1. Printing: all numerical values are rounded off by adding \( \frac{1}{2} \cdot 10^{-m} \), \( m \) being the number of decimal places specified.
2. In the case of numbers \( \geq 10^{16} \) an asterisk only is printed.
3. The ? print facility does not apply to the complex-number instructions.
4. Numbers may be read either directly, by means of a read instruction, or implicitly as constants on the right-hand side of an equation. In either case the number of digits after the decimal point is limited to 24 (no limit on integral part).
5. The description of the \( \text{arctan} \) function (given in the earlier article) is incomplete. The function calculates an angle in the range \( (-\pi, \pi) \), the quadrant being determined as if \( x, y \) were proportional to the cosine and sine of the angle respectively.

Acknowledgements

The author is indebted to Dr. B. Richards for advice and assistance at various stages of the work, and has also benefited from discussions with various friends and colleagues, in particular Dr. J. Howlett of A.E.R.E., Harwell, and Dr. G. E. Thomas of the Central Instrument Laboratory, I.C.I., Ltd.