FERRANTI PEGASUS COMPUTER

LIBRARY SPECIFICATIONS

VOLUME II

This document is a facsimile of the original book, transcribed by Christopher P Burton of the Computer Conservation Society in 2003 by the following method:

- Each page scanned at 200 dpi using Textbridge yielding 1-bit/pixel .tif files.
- Each image was then cropped by eye to have almost no white margins.
- Pages in the original (foolscap paper) which had text longer than A4 were cut and pasted to squeeze on to A4 size.
- Files were then saved as .gif image files.
- Word for Windows was then used to assemble the document, inserting one .gif image per page, with one inch left margin, 0.2 inch top margin, 0 right margin, 0.1 bottom margin on A4 paper. The images were ranged top left against those margins. It was necessary to fractionally reduce the size of each image to be slightly less than 11.38 inches high, rather than allow automatic fitting by Word.
- The document was saved and then output to an Apple Laserwriter II NTX but output to file, not actually printed. Requests to fix margins were not over-ruled. This created a PostScript file of the document, about 250 MB long.
- The PostScript file was then input to Frank Siegert's PStill program which converts to PDF to yield this document.

At the front of Volume 1 are loose pages of modifications.

R 2500

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 2
17.1.57

MATRIX INTERPRETIVE SCHEME

A scheme for simplifying the programming of matrix calculations using floating point arithmetic.

CONTENTS

1. INTRODUCTION

Many of the calculations arising in industry can be expressed in matrix form. It is therefore important that the preparation of these calculations for a digital computer should be as quick and easy as possible. This scheme is a means of specifying in easy and concise terms the transformation between the matrix form of the problem on paper and the actual operation of the machine.

The programme which carries out the operations to be described is called the Matrix Interpretive Scheme. The problem to be solved must first be expressed in matrix form. Then a method of solution has to be thought out and expressed as a sequence of elementary matrix operations, e.g. the input of a data-matrix into the computer, or the multiplication of one matrix by another. Each of these elementary steps is then written down as a single instruction which defines the operation required and the positions in the computer's store of the matrices involved. The whole sequence of matrix-instructions is called a matrix-programme.

In use the programme of the Matrix Interpretive Scheme is first read into the store of the computer by the Initial Orders in the ordinary way. The Interpretive Scheme programme then reads in the tape on which is punched the matrix programme. When the whole tape has been read and stored the individual matrix-instructions making up the matrix-programme are examined and decoded in turn by the Interpretive Scheme programme, which then carries out the operations called for.

The notation used for the matrix-instructions has been designed to be simple. The aim has been to make it possible for the person originating the problem to write the programme in this form, which is directly acceptable by the computer.

2. PRACTICAL POINTS OF THE SCHEME

2.1 Storage

The user of the scheme can visualise the store of the computer as a continuous strip of 3070 locations, (3072 if he is prepared to overwrite the date and serial number) each of which can hold one element of a matrix. The elements of a matrix occupy a block of consecutive locations on this strip. It is a very important practical point that a matrix consists of nothing more than its individual elements arranged in a certain order, and that each element can stand on its own as a number. Thus there are no row or column checksums or overall scale factors, and the dimensions of the matrix are not stored explicitly with the data. This makes it possible to change or extract individual elements, or groups of elements, to regard a rectangular matrix as a number of columns, to deal relatively easily with partitioned matrices, and in general to make efficient use of the storage space available. When a particular matrix is no longer needed the space it occupies in the store may be used again at a later stage, giving great economy of storage space.

2.2 Scaling

An important feature of the scheme is that the user can consider all numbers to be in their normal decimal form (with the decimal point wherever he wishes); when these numbers are taken into the computer they are converted automatically into binary floating-point form ($a.2^b$) which is used for all subsequent operations (the converse applies on output). This eliminates difficulties in scaling or 'overflow' and enables the numerical data to be read into the machine in the form in which they occur.

2.3 Input

The instructions and the numerical data are punched on to paper tape. The elements of the matrices are punched as written, each preceded by its sign and

During the printing of vectors in the back-substitution, row numbers will normally be punched, but may be suppressed by setting HO = 1.

5. MONITORING

After each iteration, the current approximation to the vector is compared with the previous one, and the following expression is computed.

$$\sum_{i=1}^{n'} (x_i^a - x_i^{a-1})^2$$

where x_i^a is the i^{th} element at iteration a. n' is the order of the matrix upon which the iteration is being performed.

This is a double-length quantity, and will become gradually smaller as the vector converges. The first half of this quantity is displayed in U 0.7 for as long as it is non-zero, and is then replaced by the second half. Which half is in fact being displayed can be recognised by the fact that the second half has a one in the sign digit position. This sign digit has no numerical significance, both halves being essentially positive.

This quantity will in general not come down to absolute zero, as it is not possible to obtain more accurate digits in the vector elements than are being kept in the root.

The root itself is available for monitoring in a scaled-down form in U 0.0. It is usually possible to get this to converge to the last binary digit, but in some cases it will oscillate between two adjacent values.

6. AN ADDITIONAL FACILITY FOR SPEEDING CONVERGENCE

In addition to the Aitken process, described in Section 7 below, a further facility for speeding up convergence is available. It is a modification of the process described on Page 537 of Wilkinson's paper. A fraction, ρ, may be set up on the six least significant handswitches (14-19), with 14 as its sign-digit, and the following iteration will then be performed:

$$(A - \rho \lambda I)x.$$

This has the effect of moving the origin so that, if ρ is reasonably well chosen, the effective convergence ratio is improved. The reason for the factor λ (which is the current approximation to the root) is to make it unnecessary for the operator to consider the scaling inside the machine.

7. GENERAL REMARKS ON OPERATION

7.1 The Aitken Process

This is the most useful facility for speeding convergence. It should not be applied before the root has begun to settle and the quantity in 0.7 has begun to decrease steadily. Once it has been used a few normal iterations should be allowed before it is employed again. It will not normally be of any assistance once the root has almost converged, as it will by then be operating mainly on round-off errors.

7.2 Change of Origin

This is the facility described in Section 6.

4. HOW TO USE THE SCHEME

4.1 What it Does

The matrix interpretive scheme enables a complete matrix operation to be performed by writing one single matrix-instruction. To enable a complete matrix-programme to be built up it is necessary to have certain other functions. These have been provided for in the scheme and a complete matrix-programme, comprising many separate matrix-instructions, can be built up quickly and easily.

The list of functions is comprehensive as far as the majority of matrix work is concerned. A single matrix can be copied, transposed or normalised. Two matrices can be added, subtracted, multiplied or divided. Matrix-instructions are also used for the input and output of matrices. In addition to rectangular matrices, the scheme is designed to operate on diagonal matrices, row-vectors, column-vectors, and scalar matrices. The row- and column- vectors are regarded simply as special cases of rectangular matrices.

4.2 How to Allocate the Store

Each matrix element stored in the machine occupies one storage location, of which 3,070 are available to the programmer. A matrix is stored within this space in columns; a matrix of order $m \times n$ occupies mn storage locations. For a diagonal matrix, only the diagonal line is stored so that a diagonal matrix of order $n \times n$ occupies n storage locations. A scalar or a scalar matrix is stored as a single element. The scheme has been written so that in most operations the results may be written over one or other of the operands; exact details of what may be done are given later.

4.3 How to Write Matrix-Instructions

4.3.1 Notation

In each matrix-instruction it is necessary to indicate the function, the order of the matrices and the type of matrices involved (i.e. rectangular, diagonal or scalar) and where they are held in the store. The following notation will serve in explaining how each instruction is to be written;

Matrices A, B, C, are the two operands and the result respectively. N_a, N_b, N_c are the addresses of the storage locations of the initial elements of A,B,C respectively. (N_a, N_b, N_c lie between 1 and 3,070: 0 is the address used when it is required to input or output a matrix). m, n are the orders of matrices, i.e. an $m \times n$ matrix has m rows and n columns (m, n must not be greater than 255)

$m \times n$ signifies a matrix of m rows and n columns,

$m/$ signifies a diagonal matrix of order m.

$m \times 1$ signifies a column-vector of m elements.

$1 \times n$ signifies a row-vector of n elements.

No order need be specified for a scalar or a scalar matrix.

A rectangular matrix A of order $m \times n$ is written

$$(N_a,\ m \times n)$$

A diagonal matrix A of order m is written

$$(N_a,\ m/)$$

A scalar matrix A of any order is written

$$(N_a)$$

4.3.2 Functions

The functions are represented as follows:-

Copy	A → C
Input	0 → C
Output	A(x,y) → 0 (see paragraph 4.6 for x,y)
Transpose	A* → C
Add	A + B → C
Subtract	A − B → C
Multiply	A × B → C
Divide	A,B → C (i.e. $A^{-1}B$ goes to C)
Normalise	AnB → C (scale largest element to 1 and store scale factor in N_b)
Convert to fixed-point form	AvB → C (store scale factor, i.e. exponent of maximum element, in N_b)
Transpose 'in situ'	A/ → C where C = A (For square matrices only, enables the transpose to be formed without using working space.)
Binary Output	Av → 1 For punching out matrices which it is required to read in later, and which need not be intelligible. The output will be preceded by about 6″ of blank tape, and when it is desired to re-input the matrix the tape should be placed in the reader anywhere along this blank tape. There will be a checksum at the end of the matrix.
Binary Input	0v → N_c To be used in conjunction with the binary output instruction.

A matrix which has been punched out column by column using binary output instructions can be read in later as a complete matrix.
If it is desired to output a complete rectangular matrix and later read it in, column by column, the output instruction must be of the form

Av → 9.

This instruction causes a checksum to be punched after every column. The matrix may be read back column by column, several columns at a time, or as one complete matrix.

Certain other functions are available and are described below (Para.4.3.6).

4.3.3 *Writing the Instructions*

The full instructions are written as follows:-

To *copy* matrix A from one location to another we write

$$(N_a,\ m \times n) \rightarrow N_c$$

To *add* matrix A and matrix B we write

$$(N_a,\ m \times n) + (N_b,\ m \times n) \rightarrow N_c$$

To *normalise* a row-vector we write

$$(N_a,\ 1 \times n)\ n\ (N_b) \rightarrow N_c$$

N.B. If we wished to save storage space here, the normalised vector could replace the original, and we would write

$$(N_a,\ 1 \times n)\ n\ (N_b) \rightarrow N_a$$

N_b signifies one storage location only and stores the scale factor.

To *input* a matrix, the instruction is

$$(0,\ m \times n) \rightarrow N_c$$

To *output* a matrix, the instruction is

$$(N_a,\ m \times n)\ (x,y) \rightarrow 0$$

4.3.4 *Example*

Figure 1 gives an example to show how these matrix-instructions are written and how a complete matrix-programme may be built up.

Figure 1 It is required to form the column-vector $\{y\}$ given by

$$\{y\} = \left[w\lceil D\rfloor + \Omega\left[[E] + \lceil F\rfloor\right][B]\right]\{x\},$$

where $[E]$ and $[B]$ are square matrices,
$\lceil D\rfloor$ is a diagonal matrix,
F is a scalar and $\lceil F\rfloor$ a scalar matrix,
$\{x\}$ is a column-vector,
w and Ω are scalars.

} All matrices are of order 10.

For checking purposes, it is required to print out the last row and column of the intermediate matrix

$$[C] = \Omega\left[[E] + \lceil F\rfloor\right][B].$$

The input data $[E]$, $[B]$, F, Ω, $\lceil D\rfloor$, w and $\{x\}$ are assumed to be available at input in this order.

Programme

(0, 10 × 10) → 1	Read $[E]$ into store
(0, 10 × 10) → 101	Read $[B]$ into store
(0) → 201	Read F into store
(0) → 202	Read Ω into store
(1, 10 × 10) + (201) → 1	$[E]$ replaced by $\left[[E] + \lceil F\rfloor\right]$
(202) × (1, 10 × 10) → 1	$\left[[E] + \lceil F\rfloor\right]$ replaced by $\Omega\left[[E] + \lceil F\rfloor\right]$
(1, 10 × 10) × (101, 10 × 10) → 203	Form $[C]$
(203, 10 × 10)* → 101	$[B]$ replaced by $[C]'$, the transpose of $[C]$
(191, 10 × 1)(6) → 0	Print out last column of $[C]'$
(293, 10 × 1)(6) → 0	Print out last column of $[C]$
(0, 10/) → 101	$[C]'$ replaced by $\lceil D\rfloor$
(0) → 202	Ω replaced by w
(101, 10/) × (202) → 101	$\lceil D\rfloor$ replaced by $w\lceil D\rfloor$
(101, 10/) + (203, 10 × 10) → 203	$[C]$ replaced by $\left[w\lceil D\rfloor + [C]\right]$
(0, 10 × 1) → 1	Read $\{x\}$ into store
(203, 10 × 10) × (1, 10 × 1) → 11	Form $\{y\} = \left[w\lceil D\rfloor + [C]\right]\{x\}$
(11, 10 × 1)(6) → 0	Print out $\{y\}$
*	Stop

4.3.5 Punching the Tape

In punching, each instruction must be terminated by Carriage Return and Line Feed. Spaces and erase symbols may be used in the punching and will be ignored by the scheme. The programmer should now be able to build up any matrix-instruction which he requires.

4.3.6 Other Functions

There are three other instructions which are not matrix operations but help to make the writing of a programme easier. The first is a stop-function which is punched simply as * (asterisk) on the tape and causes the machine to stop when this instruction is encountered in the matrix-programme (see also Section 6.7.4). Another instruction enables the matrix-programme to be left and ordinary machine-orders to be obeyed before re-entering the interpretive scheme. The function-character for this is the letter O and details as to how it is used will be found below. (See 'Additional Facilities'). The third instruction, with function-character J, causes a jump of control in the interpretive scheme (see 'Additional Facilities', Section 4.7).

4.4 The Programme for Reading Matrix-Instructions

It has been shown above how to punch the instructions for the scheme. It is necessary, however, to have certain other facilities associated with the programme for reading these instructions. These facilities include regulating the position in the store into which matrix-instructions are read and also the position at which the matrix-programme is to be entered. It is also possible to enter Initial Orders during input of the matrix-programme tape. There is room in the store for 80 matrix instructions, these being numbered 0 to 79. Should the matrix-programme be longer than this one can either use some of the space normally used by numerical data or else read in more instructions after the first 80 have been obeyed.

The following list gives the warning characters:-

I This causes entry into Initial Orders. When Initial Orders encounter warning character L, the interpretive scheme is re-entered, and is ready to start reading in matrix-instructions at the point it had reached when the I was read.

S*n* This causes the scheme to start obeying the matrix-programme at matrix-instruction n. (It should be noted that this is the $(n + 1)^{th}$ instruction, and not the n^{th}). S must be followed immediately by figure-shift. If, as is normal in practice, it is desired to start obeying the first matrix-instruction, it is sufficient to punch S (followed by figure-shift).

X*n* This sets the address for the next matrix-instruction to be read at n, i.e. it becomes matrix-instruction n. X must be followed by figure shift. If n is omitted (figure-shift still necessary), the programme will start reading matrix-instructions into the machine from the beginning. X is not necessary at the beginning of the tape; the programme will automatically start reading into the first location, (i.e. at instruction 0).

4.5 How to Punch Data on the Tape

The scheme is a floating-point scheme and the programme has been designed so that data can be punched either with an exponent or simply as a fixed point number. It is also possible to punch an exponent for the complete matrix or for a part of the matrix.

The following is a list of warning characters:

n This is punched to introduce an exponent which is to be carried through the matrix. It is followed immediately by a sign and then the exponent. This will be added to the exponent of each number until an asterisk or another n is read. If no exponent is put in this number will of course be zero. Space and erase will be ignored. The exponent is terminated by CR LF.

→ It is possible to print out a title for each matrix. This is done by printing an → on the data tape. Each character read after this will be printed on the output tape until two consecutive figure-shifts are encountered.

\+ These introduce a number. Spaces and erases are ignored throughout the
\- punching of a number. The number is punched in decimal with a decimal point where required; no decimal point need be punched if the number is an integer. The number of decimal digits which can be accommodated in one number is 11; if more than this are punched they will be ignored. Care should be taken not to punch more than 11 digits if the number is being punched as an integer. If the number is fractional then the extra digits will in any case be insignificant. The exponent is punched by introducing it with a sign. It must not be greater than 77.

CR LF The number is terminated by CR LF whether or not an exponent is punched Thus the number 1234.5 may be punched as +12.345+2CR LF or +1234.5CR LF.

= This is punched immediately before a check-sum. Thereafter the check-sum is punched in the same way as any other number. A certain tolerance is allowed between the check-sum computed from the elements, and the check-sum on the tape; this tolerance depends on the number of digits actually punched in the tape check-sum.

If the check-sum is not exact, i.e. if the elements of which it is the sum have been rounded off after computing the check-sum, it should be punched to as many digits as are punched in the element which is punched to the fewest number of digits, disregarding zero elements. If however the check-sum is intended to be exact, as is more common in practice, it should be punched with two more digits than are given in the separate elements, subject to one restriction. The restriction is that the sum of the moduli of the checksum and the largest element should not contain more than 9 digits. The principle can best be illustrated by examples:-

a)

```
+.2031
+.101
+.123

Sum +.4271

Punch = +.427100
```

b)

```
+10210
+  320

Sum +10530

Punch = +10530.00
```

c)

```
+1023
    +.3641259
-1000

Sum   +23.3641259

Punch = +23.36413
```

d)

```
-.12345
+.12345

Sum +.00000

Punch +.0000000
```

If +0 is punched, even such a gross error as the wrong sign for either element will not be detected.

* This must be punched after each matrix. It causes the next matrix-instruction to be obeyed, after first checking that the correct number of elements has been taken in.

4.6 The Output of Results

There are two programmes within the scheme for the output of results; they are independent of one another, and the scheme has been written so that only one may be in the machine at any one time. One programme prints numbers in floating-point form, that is, with one decimal digit preceding the decimal point and the whole argument followed by an exponent to base 10. The other programme prints numbers in a fixed-point form with a specified number of digits before the decimal point and an exponent for the whole matrix (which is printed first). There are thus two matrix-schemes, identical except for form of output, and they are known as 'floating-point matrix scheme' and 'fixed-point matrix scheme'.

A tape produced by the floating-point output programme may later be read in by the input programme, provided that not more than 9 significant figures have been punched. If the tape has been produced by the fixed-point output programme there may be occasional check-sum failures if 9 figures have been punched. It is therefore not recommended that fixed-point output tapes to 9 figures should be used for re-input. However, if this situation cannot be avoided, it would be reasonable to by-pass any check-sum failures as described in 6.7.1.(3), since the type of error which the check-sum is designed to bring to light should not normally occur on tapes output by the computer. Alternatively, the modification to the scheme which avoids the punching of check-sums (as described in 11.5(d)) may be used at the stage when the matrix concerned is output.

A titling sequence is printed to distinguish each matrix which is output. It consists of → and a decimal number specifying which matrix instruction caused output to occur. For example, →16 means that the 16^{th} instruction is being obeyed.

The elements of the matrix are printed in columns. Each column is preceded by two line-feeds and the column number, which appears centrally over the column. The column number is followed by an extra line-feed.

Each element is preceded by a row number and two spaces. These may be suppressed at any time during punching by setting the sign-bit of the handswitches to one.

4.6.1 Floating-Point Form

Each element consists of a signed argument and a signed exponent to base 10. The argument is printed with one decimal digit before the decimal point and $(x - 1)$ digits after the point, where x is a parameter, stated in the output instruction, signifying the total number of digits required. If x is not stated, it will be assumed to be 9. A typical element with $x = 7$ is, e.g.:-

51 +2.123456 -17

where the number 51 before the sign indicates the row number.

The maximum accuracy which can be expected from the scheme, which works throughout in floating-point arithmetic, is 9 significant decimal figures. (The range of exponent is ±77 approximately).

At the foot of each column a check-sum is printed, preceded by an extra line-feed and an = sign. The number of decimal digits printed in the argument of the check-sum is adjusted automatically according to the difference in magnitude between itself and the element of maximum modulus. If the check-sum has the same exponent as the maximum element, then the check-sum is printed to the same number of digits. If the check-sum is greater in exponent by 2 (say), then it is printed with two more significant figures. That is to say, the final digit printed for the

check-sum has the same significance as the final digit of the maximum element. The check-sums are rounded off before printing (as are the elements).

Figure 2 shows a typical matrix of 5 rows and 3 columns as printed from a tape produced by the floating-point form of output programme.

4.6.2 Fixed-Point Form

Elements are printed with y digits (or spaces) before the decimal point, where y is a parameter stated in the output instruction. x, the maximum number of digits required is also stated. The number of digits which is printed after the decimal point is $(x - y)$ and this is constant throughout the matrix; hence, the least-significant digits of all elements are in line with one another. The sign immediately precedes each element. The appropriate round-off constant is calculated for each element, and added to it before printing occurs. If x is not stated it will be assumed to be 9 and y will be assumed to be 1. Elements might appear as follows (with $x = 8$ and $y = 4$):-

```
51  + 1234.5678
52       -6.7809
```

At the foot of each column a check-sum is printed, preceded by an extra line-feed and an = sign. As for each element, the number of digits printed after the decimal point is $(x - y)$. The check-sum is rounded before printing. The characters →, n, =, which are used to precede the title, exponent and check-sum respectively, are the necessary warning characters to make a tape which is output completely acceptable for re-input.

Figure 3 shows a typical matrix of 5 rows and 3 columns as printed from the tape produced by the fixed-point form of output programme.

Figures 2 and 3

Specimen Output

```
→
21

      0

0   +1.3742  +3
1   +2.1840  +2
2   -7.9336  +2
3   -6.0011  +0
4   +3.6100  -10

=   +7.932   +2

      1

0   +2.6098  -1
1   -1.1122  +0
2   +1.1132  +0
3   +5.3142  -2
4   +4.1592  +1

=   +4.1907  +1

      2

0   -8.9793  +3
1   +0
2   -2.3846  -1
3   -2.6433  +1
4   +8.3279  +2

=   -8.1732  +3
*
```

Figure 2. Floating-point form

```
→
21
n+2

      0

0   +13.742
1    +2.184
2    -7.934
3    -0.060
4    +0.000

=   +7.932

      1

0    +0.002
1    -0.011
2    +0.011
3    +0.001
4    +0.416

=    +0.419

      2

0   -89.793
1    +0.000
2    -0.002
3    -0.264
4    +8.328

=   -81.732
*
```

Figure 3. Fixed-point form

The above show the two ways in which a page printed from an output tape can appear. In each case the same matrix of 5 rows and 3 columns has been printed.

4.7 Additional Facilities

4.7.1 To Enter Machine-Orders from the Scheme

There is an instruction which enables the interpretive scheme to be left for a while to obey ordinary machine-orders and then return to the interpretive scheme, either at the same place or at any other instruction. The order used for this is the 0-order. That is to say it is written as a letter 0 followed immediately by a figure-shift and then the single-word address of the first machine-order to which control is to be transferred. Spaces may be punched before the address.

The address written in the 0-order should be terminated by CR LF.

The block containing the first machine-order will be brought down to block 0 of the computing store and the appropriate order entered. This order must be the first of an order-pair but can be anywhere in the main store. On entry 5_m = Address of Machine Order Entered; 5_c is clear. Thus X5 may be used as a modifier to bring down further blocks of programme.

To return to the interpretive scheme at matrix instruction n, the following orders are necessary.

```
(n)   7 40
65   [1] 72
1.2+  0 60
```

The last two instructions must be a pair if they occur in U 1. To return to the next matrix instruction, it is merely necessary to obey

```
61   [0] 72
0.0   0 60
```

which must form an order pair if in U0.

If the matrix-programme requires more storage-locations than are available then the matrix-instruction 0 512 may be used to cause the Order-Read programme to be entered. This can be used to read in the next section of the matrix-programme from the input-tape when all the matrix-instructions that are in the store have been obeyed. In a similar way the matrix-instruction 0 4096 causes the Initial Orders to be entered.

The machine orders may be anywhere in the main store but care must be taken not to overwrite anything else which may be required. If there are less than 80 matrix instructions, space will become available in the first quarter of the store. The 80 instructions occupy two words each, from B 108.0 to 127.7; thus if only 60 instructions are required, blocks 123 to 127 inclusive are available for machine orders. Alternatively, data space may be sacrificed. Each matrix element occupies

one location and location 1 corresponds to B 128.0; from this it can be worked out what space will not be used by data. A third possibility is to overwrite such of the scheme itself as is not being used. However, since much of the interpretive programme is organisational, and the subroutines are interlinked, there will not normally be more than a few blocks available, and this method is not recommended.

4.7.2 To Jump in the Scheme

The instruction to jump within the matrix-programme is written as a J followed immediately by a figure-shift and then the address of the matrix-instruction to which it is required to jump. Spaces may be punched before the address. This address should be terminated by CR LF. Actually it must be terminated by a character other than space or erase (both of which will be ignored) and there must be at least one character before the next warning character. As other functions are terminated by CR LF these should be used here also.

5. USE OF THE SCHEME WITH PRESET PARAMETERS

There is an extension to the scheme to permit the use of preset parameters. For a problem of a given type this allows one matrix-programme to be written which can be used with matrices of different sizes.

When this facility is to be used the programme is written in general terms, and a small subsidiary programme (called the 'Interlude') is used to insert into the matrix instructions the dimensions of all the matrices for the particular calculation and also to allocate the storage locations.

If the matrices A, B, C of order $(m_a \times n_a)$, $(m_b \times n_b)$, $(m_c \times n_c)$ are the two operands and the result respectively of some operation used in the Matrix Interpretive Scheme and if N_a, N_b, N_c are the addresses of the storage locations holding the first elements of A, B, C respectively then normally

$$1 \leqslant N_a,\ N_b,\ N_c \leqslant 3070$$

$$1 \leqslant m_a,\ n_a,\ m_b,\ n_b,\ m_c,\ n_c \leqslant 255$$

When using preset parameters, however, the matrix instructions are written in the ordinary way but the storage locations (N_a, N_b, N_c) are specified by numbers above 5,000 and the dimensions $(m_a, n_a, m_b, n_b, m_c, n_c)$ of the matrices by numbers larger than 240. More precisely, if N denotes any of N_a, N_b, N_c and m any of the m_a, n_a, etc. appearing in the matrix instruction, then

if $1 \leqslant N \leqslant 3070$	N is the address of the actual storage location holding the first element of the matrix.
if $N \geqslant 5001$	The actual address of the first element of the matrix is specified separately by the programmer.
if $1 \leqslant m \leqslant 240$	m is the actual order of the matrix
but if $241 \leqslant m \leqslant 255$	the actual order of the matrix is specified separately by the programmer.

That is if $1 \leqslant N \leqslant 3070$ and $1 \leqslant m \leqslant 240$ the matrix interpretive scheme operates as usual. When the programmer is using the preset parameter scheme two lists will be kept, an address list containing the values of N and a dimension list containing the values of m; the values being appropriate to the problem to be solved.

The matrix instructions will be written with 5001 for N, 241 for m etc, and these instructions will be read into the store by the Matrix Interpretive Scheme. The Address list and Dimension list are then read into the store. The Interlude

programme processes the matrix instructions, replacing any addresses exceeding 5000 by the appropriate N from the address list, and the orders of any matrices exceeding 240 by the appropriate m from the dimension list. The set of matrix instructions so obtained can be obeyed in the usual manner.

The advantage of the preset parameter scheme is that the actual programme of matrix instructions will apply to all problems of a given type. The orders of the matrices and the storage locations which are to hold these matrices are set separately at the last moment.

Example: Q is a matrix of order $m \times n$ $(m \leqslant n)$

A is a matrix of order $m \times m$

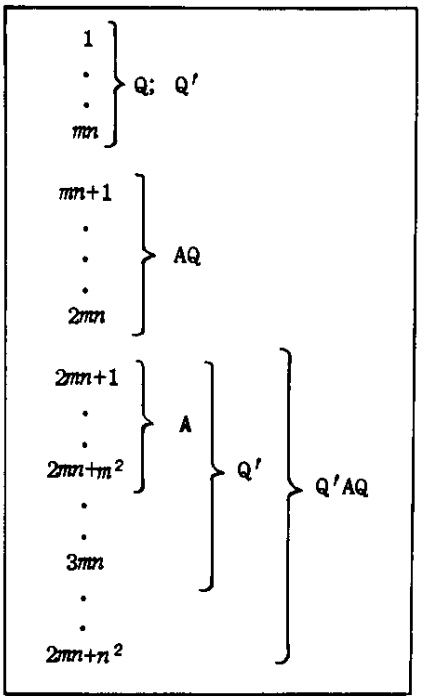
It is required to form Q'AQ for a large number of different values of m, n.

Programme of Matrix Instructions

(0, 241 x 242) → 1	Input Q ($m \times n$)
(0, 241 x 241) → 5001	Input A ($m \times m$)
(5001, 241 x 241) x (1, 241 x 242) → 5002	Form AQ ($m \times n$)
(1, 241 x 242)* → 5001	Form Q' ($n \times m$)
(5001, 242 x 241) → 1	Copy Q' over Q
(1, 242 x 241) x (5002, 241 x 242) → 5001	Form Q'AQ ($n \times n$)
(5001, 242 x 242) → 0	Output Q'AQ ($n \times n$)

Dimension List		Address List	
241	m	5001	$(2mn + 1)$
242	n	5002	$(mn + 1)$

The layout of the portion of the store containing the matrix data can be visualised during this problem to be as shown.



The above matrix programme will then carry out the operation Q'AQ for the values of m, n specified in the lists, i.e. the same programme will work for example with

$m = 2$, $n = 53$ or $m = 10$, $n = 46$ or $m = 31$, $n = 31$

For this last example the lists would be as follows:

Dimension List		Address List	
241	31	5001	1923
242	31	5002	962

Since the number of locations available for data with the Matrix Interpretive Scheme is 3070 it follows that m, n must satisfy $2mn + n^2 \leqslant 3070$ in the above example.

6. MACHINE OPERATION

6.1 Operation of the Scheme without Preset Parameters

6.1.1 Programme Tapes

Two tapes are required.

(i) Matrix Interpretive Scheme

START. RUN. Ends on 77 stop in 0.6. (Warning character Z at the end of tape). A loop stop in 3.4 indicates an error and the tape must be read again.

(ii) Matrix Programme

STOP. RUN.

Tape should read

D
N

Name of Programme

Blank Tape

Machine Orders (if any)

J 64.0

Matrix Instructions

S *n*, where it is desired to start obeying matrix instruction *n*.

6.1.2 Data Tapes

The first matrix order to be obeyed will normally be a data input order, and there is an optional stop at the beginning of this type of order. The data tape should be put into the reader at this point.

If there are further data instructions, and the corresponding matrices are all on one tape, the optional stop may be inhibited to avoid unnecessary stops.

6.2 Operation of the Scheme with Preset Parameters

6.2.1 Programme Tapes

Three tapes are usually required to set up a matrix programme. They must be put in the tape reader in the following order:

(i) Matrix Interpretive Scheme

As in section 6.1.1 above.

(ii) Matrix Programme

STOP. RUN.

Contents of Tape:

D

N

Name of Programme

Blank Tape

Machine Orders

J 64.0 (Enters Matrix Order-Read)

Matrix Instructions

I (Enters Initial Orders)

Z

Z causes a 77 stop in 0.6 at the end of the tape.

(iii) Preset Parameter Tape

Read by moving key to STOP and then to RUN.

Example from Section 5

Heading	T 1600	T 1600
	+m	+31
Dimension List	+n	+31
	⋮	
Heading	T 2000	T 2000
	+x	+1923
Address List	+y	+962
	⋮	
Entry to Interlude	J 40.0	J 40.0
	(or E 40.0)	

If a 77 stop is required at the end of the tape warning character E may be used. There is an optional stop in the matrix scheme before reading data and it is usually better to use warning character J as shown above.

6.2.2 Data Tapes

The use of preset parameters does not affect the rules for punching data or the method of operating the computer. It is sometimes convenient to combine the preset parameter tape with the data tape, but care must be taken not to read the preset parameter tape twice.

6.2.3 The Operation of the Interlude

Preset Parameters are inserted in Matrix Orders by a short programme called the Interlude. The Interlude is entered when J 40.0 is obeyed at the end of the Preset Parameter Tape.

When the Interlude is obeyed it finds the address of the last matrix instruction read in. It then processes the contents of all locations from Matrix Instruction 0 to this last address. If anything other than a matrix instruction is stored in these locations it may be altered by the interlude.

The first word of a matrix instruction contains three addresses, N_a, N_b and N_c. During the interlude each of these numbers is compared with 5001. Those addresses that are less than 5001 are left unchanged but those that exceed 5000 are replaced by the appropriate number from the address list.

The second word of a matrix instruction contains four matrix dimensions, m_a, n_a, m_b and n_b. Each of these numbers is compared with 241. Those dimensions that are less than 241 are not changed but those that exceed 240 are replaced by the appropriate number from the dimension list.

On completing the processing the Computer will obey matrix instructions, starting at number 0 (but see 6.2.5. below). The dimension list and the address list will usually be overwritten during the working of the matrix programme.

6.2.4 Changing Parameters

If it is desired to alter any of the preset parameters a complete new preset parameter tape must be prepared. The matrix scheme need not be read again, but the programme tape must be read again before the new preset parameter tape.

6.2.5 Entering the Matrix Programme

The interlude is written in such a way that the Matrix Programme is entered at Instruction Number 0. If it is required to enter at some other instruction then instruction number 0 should be a jump instruction. If this is not satisfactory the Interlude can be made to jump directly to matrix instruction number n. This may be achieved by including in the Programme Tape the sequence.

X 42.0

(n) 7 40

6.3 Special Uses of Preset Parameters

The parameters x and y used by the Matrix Print Routines may be preset. x appears in the address position of the order and must be included in the address list. y is held as a dimension and must be included in the dimension list.

If x and y are to be preset it may be useful to include an optional parameter list in the main programme.

Example: In the following programme x and y are usually 6 and 3.

```
              T 1602
Optional y:   + 3
              T 2002
Optional x:   + 6
              J 64.0
Programme:    (0, 241 × 242) → 1
              (0, 242 × 241) → 5001
              (1, 241 × 242) × (5001, 242 × 241) → 5002
              (5002, 241 × 241) (5003, 243) → 0
              I
              Z
```

If it were required to change x to 5 but leave y unaltered the preset parameter tape would read as follows:

	T 1600
Dimensions	+ 8
	+ 4
	T 2000
Addresses	+ 33
	+ 65
Value of x	+ 5
	J 40.0

The addresses in Matrix Jump Instructions (function J), and in jumps to machine orders (function 0), are stored as addresses and may be preset if desired.

6.4 Corrections to Matrix Programmes

Matrix Scheme Warning Character X may be used to correct Matrix Programmes by inserting new Matrix Instructions. The effect of X is to set a Matrix Instruction Number in the same way as Initial Orders warning character T sets a transfer address. When using Preset Parameters it is necessary to reset the Matrix Instruction Number at the end of the Correction Tape.

Suppose that it is required to correct a Matrix Programme of $(n+1)$ instructions, numbered 0 to n. If the r^{th} instruction has to be changed the correction tape should read as shown on the left below:

Example

With 30 Instructions

$n = 29$

J 64.0	J 64.0
X r	X 12
New Instruction	(5002, 243/) → 5001
X $n+1$	X 30
I	I
Z	Z

The correction tape must be read after the Programme Tape but before the Preset Parameter Tape. If the final X $n+1$ is omitted the Interlude will process instructions up to number r, but not beyond.

6.5 Gaps between Matrix Instructions

Matrix Instructions are normally read into consecutive locations in the main store, starting with Instruction number 0. It is possible, by using Matrix Scheme warning character X, to miss out some locations between Matrix Instructions. If this is done the two locations which would have been occupied by Matrix Instructions should be cleared. Otherwise the Interlude will find a random parameter: this may cause a 'Writing with Overflow' stop or a 'Drum Parity Failure', after reading from a non-existent location.

6.6 Calculation of the Address List

If a programme is used very frequently it may be helpful to arrange for it to calculate its own address list. The programme for this calculation could be stored in any block beyond 128.0; it may be overwritten by data during the matrix programme. The addresses are normally simple functions of the dimensions; any additional parameters required could be added to the end of the dimension list.

The preset parameters tape would then read

Dimension List

T 1600
+ *m*
+ *n*
..
J 132.0 (say)

The address list is replaced by a J or E sequence which causes the computer to start obeying the programme for calculating addresses. After calculating the addresses, and, if possible, checking their sum total, the programme should store them in location 2000 onwards. The interlude must then be entered by bringing its three blocks into the computing store:

B 40 to U0
B 41 to U1
B 42 to U2

and jumping to 0.0.

6.7 Stops

During the operation of the scheme the following stops may occur.

6.7.1 During data input

(1) Optional stop in 0.2 at the beginning of each data input instruction, as already described.

(2) 77 stop in 3.5. This occurs when a character is wrongly read from the tape. If the tape reader is at fault, or if the tape can be corrected immediately, it may be pulled back until the element following the last check sum is about to be read (or, if there are no checksums on the tape, the first element of the matrix) and the RUN key operated. (See also (7) below).

(3) Loop stop in 3.0+. Checksum failure.
If it is apparent that the checksum is at fault, a manual jump to 3.1 is permissible.

If a data element since the previous checksum is wrong and can be corrected, the tape may be pulled back to just after the previous checksum, and a manual jump made to 2.7. On 77 stop in 3.5, RUN.

(4) Loop stop in 0.4. Incorrect number of elements in matrix. The excess number of elements is given by the modifier of accumulator 5. If there is a deficiency accumulator 5 will be negative.

If the mistake can be corrected, a short tape as follows should be prepared.

J 64.0
S *n*

where matrix-instruction *n* is the one which went wrong. This should be read via Initial Orders, then on the optional stop in 0.2 the corrected data tape should be replaced at the beginning of the relevant matrix, and RUN.

(5) Loop stop in 2.4. Exponent of number either too large or too small.

(6) Loop stop in 0.5 in the Initial Orders. This or other stops may occur if the START key is operated (instead of the RUN key) and matrix data are read by the Initial Orders.

(7) Number tapes punched out by the Matrix Scheme may normally be read in again. A difficulty may arise if names of matrices read during the programme are repeated on the tape: they will not be preceded by an arrow and may cause a '77 stop in 3.5. The tape should be pulled past the name sequence and the RUN key operated.

This difficulty may be overcome either by adding an extra arrow to the original name sequence or omitting it altogether. Names punched by the Matrix Scheme consist only of decimal digits and do not cause a stoppage even when the arrow is removed.

The procedures described in (2) (3) and (4) involving correction of tapes whilst on the computer are not recommended unless

(a) The error can be easily found and corrected.

(b) Considerable machine time would be wasted if the programme had to be started from the beginning at some later date.

6.7.2 During binary data input

(1) Optional stop in 0.0 for tape to be put in reader.

(2) Loop stop in 0.3. Character other than erase at the head of the tape. If the tape is moved until the erase (or preceding blank tape) is under the tape reader, a manual jump to 0.1 is permissible.

(3) Loop stop in 0.3+. Checksum failure.

(4) Loop stop in 1.6+. Too many elements read. This usually indicates an error in the Matrix Instruction.

6.7.3 During Division and Normalise Orders

(1) Loop stop in 1.3. Attempting to divide by zero.

*6.7.4 Matrix-instruction * (asterisk)*

This causes 77 stop in 0.7. RUN to obey the next matrix-instruction (or may indicate end of matrix-programme).

6.7.5 During Order Input

(1) 77 stop in 3.3. Z on tape. RUN to return to Order Read.

(2) Loop stop in 0.6+ when wrong character read at beginning of matrix instruction.

(3) Loop stop in 2.2 when wrong character read during input of address and dimensions of A or B matrix.

(4) Loop stop in 3.2+ when wrong character read after ")", i.e. when expecting a function such as + or −.

(5) Loop stop in 3.7 if wrong character during C address, or if C address not terminated by CR LF or if a non-existent function is implicit in the matrix instruction, e.g.

$$(1, 2 \times 2) * (5, 2 \times 2) \rightarrow 9$$

(6) Loop stop in 1.0 if X, S, 0 or J not followed by Ø. Erase is not ignored.

6.7.6 During other matrix instructions

(1) If locations into which no data has been put are called for by mistake, non-standard floating point numbers may be obtained. These may lead to write with overflow stops at various points or to peculiar output.

(2) If errors in matrix instructions occur (there is no check on conformability on order input), looping may take place. Alternatively the machine may call for non-existent addresses, leading to parity failures. It may obey numbers on certain faulty matrix instructions, leading to optional stops or unassigned order stops or other peculiar circumstances.

7. ALLOCATION OF THE STORE

It is usually possible to write the answer, C, to a matrix operation, over one of the operands A or B. This rule is always true for functions other than transposition, multiplication and division; it applies also to special cases of these functions.

7.1 Transposition C = A′

When using the function * C may never be written over A.

When using the function / C is always written over A.

7.2 Multiplication

Scalar × Matrix Matrix × Scalar Diag × Matrix Matrix × Diag	The result may be written over the matrix, whether the matrix is rectangular, diagonal or a vector.
Rect × Vector Vector × Rect	The result may overwrite the first column of the matrix but may not overwrite the vector.
Row Vector × Col Vector	The result may overwrite any element of the operands.
Col Vector × Row Vector Rect × Rect	Overwriting is not permissible.

7.3 Division C = A^{-1} B

A	B	
Square	Rect	A and B are both spoiled and B is replaced by C′. The result C may not overwrite A or B.
Square	Diag Scalar	A is spoiled but B is unaltered; C may not overwrite A or B.
Square	Col Vector	A and B are both spoiled, B is replaced by C′. C may overwrite B but not A.

Diag	Rect	A is replaced by A^{-1} but B is unaltered. C may overwrite B but not A.
Scalar	Rect	
Scalar	Diag	
Scalar	Vector	
Diag	Diag	A is replaced by A^{-1} but B is unaltered. C may overwrite either A or B.
Diag	Vector	
Scalar	Scalar	
Diag	Scalar	A is replaced by A^{-1} but B is unaltered. C may overwrite A but not B.

8. SPEED OF OPERATION

8.1 The times of some typical matrix-instructions are given in Figure 4. (Pages 26, 27). The formulae given in the second column give the approximate time to obey one order (in milliseconds). The last column gives some typical times for vectors or square matrices of various orders (in seconds or in minutes and seconds). The decoding of each matrix instruction takes about $\frac{1}{8}$ second and this time has been included.

8.2 The bulk of the calculation time in any typical problem is likely to be occupied by multiplications of one rectangular matrix by another, and by non-trivial division instructions (such an instruction is non-trivial if A is a full square matrix); time of output may also be significant if there are many results to be punched.

8.3 It must be emphasised that the times given are only averages. Exact times cannot be given for two main reasons:-

(a) The time taken by a floating point operation depends on the actual numbers involved.

(b) Drum access times depend on the times taken by other operations, and on the dimensions of the matrices.

It has turned out in practice, however, that times computed from the formulae given are normally fairly close to the actual time, say within 10%, and the actual times are usually lower than the estimated ones.

8.4 The following abbreviations have been used in Figure 4:-

Rect.	Rectangular matrix.
Diag.	Diagonal matrix.
Col. vector	Column vector.
Scalar	Scalar or Scalar matrix.

9. THE REPRESENTATION OF FLOATING-POINT NUMBERS

9.1 In the floating-point operations used with the Interpretive Scheme each number x (i.e. each element of each matrix) is represented within the computer by one word, the least-significant 9 bits of which are used for a binary *exponent* b. The value of x is given by the exponent b and an *argument* a (numerical part); the values of a and b are defined as follows:-

(i) $x = a.2^b$

(ii) b is an integer satisfying the relationships $-256 \leqslant b \leqslant 255$.

(iii) either $\frac{1}{4} \leqslant a < \frac{1}{2}$

or $-\frac{1}{2} \leqslant a < -\frac{1}{4}$

or $a = 0$ and $b = -256$.

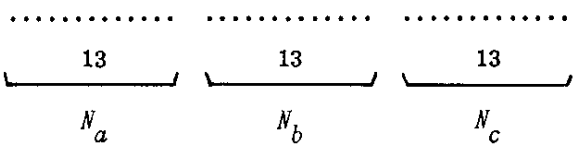
9.2 The least-significant 9 bits of the word hold not the exponent b itself, but $b + 256$; this is non-negative. The whole of the word (39 bits) is used to give the value of the argument a; i.e. the digits of the exponent are used to round-off the value of the argument. There is very little bias in this round-off provided the numbers being handled are near unity in absolute value. It should be noted that if the last 9 bits are excluded the value of a is given to about 9 decimal digits; as these are very efficiently used in this type of floating-point scheme any bias in rounding is not likely to be serious. There can be trouble in cases where the elements of the matrices are known exactly (e.g. where they are integers) but the scheme is not primarily intended for such cases. Another small point to note is that, with the above conventions, zero is held in the form 0.2^{-256} and is represented by a 'clear' word, i.e. one whose digits are all zero.

9.3 When arithmetical operations are carried out on numbers in the above form they are first 'unpacked' into their separate exponents and arguments. The answers are 'packed' just before being stored. Should an exponent ever become smaller than -256 the number is set equal to zero; there is no check on floating point overflow.

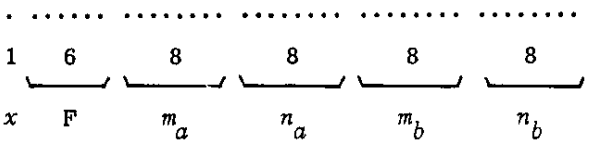
10. HOW MATRIX-INSTRUCTIONS ARE INTERPRETED

10.1 Each matrix-instruction occupies two words, the first of which must have an even address. The digits of these words are allotted to the various parts of the order as follows:-

First word



Second word



In these diagrams a dot represents a binary digit: the most-significant digit is, as usual, on the left. The notation is as follows:-

x	a spare digit (i.e. not at present used),
N_a, N_b, N_c	the three addresses in the matrix-instruction,
F	six digits indicating the function,
m_a, n_a	the number of rows and columns in the first matrix,
m_b, n_b	the number of rows and columns in the second matrix.

10.2 The interpretation or decoding of a matrix instruction is effected in the following way. First the three addresses are shifted into the modifier position in three of the accumulators and the row and column numbers go to form counters. The six function digits are then used to select a 'link' from a list held in the store and this link is used to call in the appropriate subroutine to carry out the operation called for. When the operation is complete a return is made to the central or controlling part of the programme which extracts the next matrix-instruction from the matrix-programme and interprets it.

Figure 4

Operation		Formula giving time in milliseconds	Examples of times in seconds		
A matrix	B matrix		16	24	32
ADD/SUBTRACT					
Rect.	Rect.	$16mn + 190$	4.3	9.4	16.6
Diag.	Diag.	$16m + 190$	.5	.6	.7
Vector	Vector				
Square	Diag.	$5m^2 + 21m + 200$	1.7	3.6	6.0
	Scalar				
Diag.	Square				
Scalar					
Diag.	Scalar	$15m + 190$	.4	.6	.7
Scalar	Diag.				
MULTIPLICATION (See note)					
Rect ($m \times n$)	Rect ($n \times r$)	$[(21n + 12)r + 12]m + 150$	1m 29.4	4m 57.7	11m 41.0
Rect	Col. vector	$21mn + 24m + 150$	5.9	12.8	22.4
Row vector	Rect	$21mn + 12n + 170$	5.7	12.5	22.0
Row vector	Col. vector	$21n + 180$	.5	.7	.9
Col. vector	Row vector	$33mn + 12m + 150$	8.8	19.4	34.3
Rect.	Diag.	$10mn + 11n + 170$	2.9	6.2	10.8
Row vector	Diag.	$21n + 170$	.5	.7	.9
Rect.	Scalar	$10mn + n + 180$	2.8	6.0	10.5
Row vector	Scalar	$11n + 180$	.3	.4	.5
Diag.	Rect.	$12mn + 11n + 170$	3.4	7.4	12.8
Scalar					
Diag.	Col. vector	$12m + 180$	.4	.5	.6
Scalar					
Diag.	Diag.	$12m + 180$	.4	.5	.6
Scalar	Diag.	$11m + 180$	.4	.5	.6
Diag.	Scalar				
COPY					
Vector or Diag.		$5m + 150$	.2	.3	.3
Rect.		$5mn + 150$	1.4	2.9	5.3

Note: All the multiplication times will be reduced if there are zeros in either the A or B matrix. The saving can be quite substantial, being over 30% in the rectangular times rectangular case with one matrix almost all zeros.

Operation		Formula giving time in milliseconds	Examples of times in seconds		
A matrix	B matrix		16	24	32
TRANSPOSE					
Ordinary		$12mn + 130$	3.2	7.0	12.4
'In situ'		$13n^2 + 150$	3.5	7.6	13.4
NORMALISE		$15mn + 190$	4.0	8.8	15.6
CONVERT		$12mn + 170$	3.3	7.1	12.5
INPUT (See notes)					
Rect.		5 ms. per character read, including layout characters, signs etc, plus 70-90 ms. per number	33.3	1m 14.9	2m 13.1
Vector			2.1	3.1	4.2
OUTPUT (See notes)					
Rect. } Fixed-point		30 ms. per character punched, including layout characters etc, plus about 60 ms. per number, including checksums.	1m 43.0	3m 43.5	6m 30.1
Vector } Fixed-point			6.7	9.6	12.5
Rect. } Floating-point			2m 14.6	4m 55.4	8m 36.7
Vector } Floating-point			8.7	12.5	16.3
BINARY INPUT		$40mn$	10.2	23.0	41.0
BINARY OUTPUT		$250mn$	1m 4.0	2m 24.0	4m 16.0
DIVISION					
Square $(n \times n)$	Rect $(n \times r)$	$19n^2r + 8.5n^3 + 30nr + 131n^2 + 181n + 170$	2m 37.0	7m 57.0	17m 52.0
Square	Col. vector	$8.5n^3 + 150n^2 + 211n + 170$	1m 17.0	3m 29.0	7m 19.0
Diag.	Rect.	$12nr + 11n + 11r + 170$	3.6	7.7	13.2
Diag.	Diag.	$27n + 180$	0.6	0.8	1.0
Scalar	Rect.	$12nr + 11r + 180$	3.4	7.4	12.8

Notes: 1) The specimen input times assume 6 digits, sign and decimal point for each number, with no checksums, row numbers etc.

2) The specimen output times assume 6 digits are punched for each number, and that the punching of row numbers is suppressed.

11. MODIFICATIONS TO THE SCHEME

Some of the facilities provided by the scheme can be easily changed by minor modifications. These modifications should normally be read in immediately after the scheme, before the J 64.0. A list of some of the more useful modifications follows:-

11.1 The address of the first matrix instruction is normally 108.0. It can be changed to *B.P* (*P* must be even) by

T 63.5

```
B - P0 0.
0
```

11.2 The true address of matrix element 1 is 1024 (128.0). It can be changed to *N* by

T 63.6

```
(N-9) - -0 0.
0
```

11.3 When using preset parameters, the dimension and address lists are normally read into 1600 and 2000, respectively. They can be changed to *N* by

T 42.1
~~T 40.1~~

```
N - -0 0.
0
```

(Address list)

T 42.4
~~T 40.4~~

```
N - -0 0.
0
```

(Dimension list)

11.4 The fixed-point output matrix scheme does not give true fixed-point, as there is an overall scale-factor at the beginning of the matrix. The following amendments (which can be used only with the fixed-point matrix scheme) will give a true fixed point output.

T 49.4

```
17  0 10
5.4 2 10
5.0 2 11
0
50  0 72
0.6+ 0 60
```

T 91.2

```
-1.0
```

The matrix instruction should be written

$$(N_a,\ m \times n)\ (x) \rightarrow 0$$

where x specifies the number of digits after the decimal point.

11.5 There are several possible amendments to the output, which are not the same for the fixed and floating point schemes. They are given in the following table:

Amendment	Floating-point	Fixed-point
(a) Avoid printing arrow before title.	X 91.7 `2.6 0 60`	T 91.6 `-1.0`
(b) Avoid printing title (i.e. instruction numbers).	T 43.3 `-1.0`	X 43.4+-53.5 `0` `0`
(c) Avoid printing column numbers.	X 47.0 `2.4+ 0 60`	X 52.7 `1.3+ 0 60`
(d) Avoid printing checksums.	X 55.6 `3.0 0 60`	X 58.6 `3.0 0 60`
(e) Avoid printing * at end of matrix.	X 55.2+ `0`	X 58.2+ `0`

(f) To give 6 LF's before each column number. (Not compatible with (c) above).

T 47.1

```
16  1 10
0.0+ 2 66
```

T 53.0

```
16  1 10
2.7+ 2 66
```

11.6 There is available a modification which causes each matrix instruction to be printed out before being obeyed. It is for use in developing matrix programmes. Its number is R 2501 and it is fully described in a separate specification.

12. BINARY PUNCHING OF MATRIX PROGRAMMES

Each matrix instruction occupies two locations of the Main Store and the first instruction is normally stored in B 108.0 and 108.1. It is possible to use R 1033 to punch out matrix programmes in binary form. If the programme uses preset parameters the contents of B 2.2 must be punched out as well as the actual instructions.

If, for example, a matrix programme using preset parameters has 70 instructions (0 to 69) and some machine orders in B 510, the steering tape for R 1033 would contain

```
A 4  2.2
A 4  108.0 - 125.3
A 4  510.0 - 510.7
```

Warning Characters would be inserted by name-sequences as described in the specification of R 1033.

The steering tape must be inserted after the matrix programme but before the preset parameter tape. If the programme tape ends with a warning character S it will normally be satisfactory to wait for the optional stop which occurs before reading data. At this point START and RUN to read the steering tape. If the first matrix instruction is not an input order it may be necessary to place a warning character Z before S on the programme tape, then START and RUN to read the steering tape.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON. W.1.

Issue 2
17th January, 1957.
J.F.D. D.M.

R 2501

File as 501

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
14.3.57.

ERROR TRACER FOR MATRIX PROGRAMMES

A routine designed to assist the computer operator in following the course of a Matrix Scheme Programme.

(R 2501 uses the Initial Orders number printing routine.)

Name: ERROR TRACER

Store: 3 consecutive blocks, which may be in any part of the Main Store not used by the Matrix Scheme,
plus B 42.6, 42.7 and 62.4.

Uses: U 3,4,5; X 2,3; B 0.

Entry: Direct from the Matrix Scheme Master, immediately before obeying each Matrix Scheme instruction.

Time:

With all printing suppressed	80 ms per instruction
Printing instruction numbers	275 ms per instruction
Printing complete instructions	1850 ms per instruction.

1. METHOD OF USE

1.1 Insert the Matrix Scheme, R 2500, in the main tape reader. START and RUN.

1.2 Insert ERROR TRACER, R 2501, in the main tape reader. RUN.

(This tape has T 85.0 at its head. If the programme to be run uses the Matrix Scheme normalise instruction, R 2501 must be stored elsewhere. If the tape is inserted after the T 85.0, R 2501 will go into the next available block in the store, as specified by the Transfer Address in U 5.7. See section 5.)

1.3 Set the two least significant handswitches as described in section 3.

1.4 Insert the Programme Tape, Preset Parameter Tape and Data Tape in the normal way.

2. OPTIONAL STOP

There is an optional stop in 3.3 before each Matrix Instruction. At this point the matrix instruction number is displayed as an integer in U 5.0; it will remain there until disturbed by the operation of the matrix instruction.

The handswitches may be changed at this optional stop if required.

3. HANDSWITCH CONTROL

3.1 If H 18 = 0 and H 19 = 0 } R 2501 will not cause any printing

3.2 If H 18 = 0 and H 19 = 1 } R 2501 will print CR LF i where i is the instruction number.

3.3 If H 18 = 1 R 2501 will ignore H 19 and print as shown below:

CR LF i F m_a n_a N_a N_b N_c m_b n_b

where i = Instruction number

F = Function number (see section 4)

m_a n_a = Dimensions of A matrix

N_a = Address of A matrix

N_b = Address of B matrix

N_c = Address of C matrix

m_b n_b = Dimensions of B matrix

4. THE FUNCTION NUMBER

Matrix operations are denoted in the Matrix Scheme by numbers. These numbers are printed by R 2501 and may be interpreted by reference to the list below:

0	Transpose	21	M − M, D − D, S − S
1	Copy	22	D − S
2	M × M	23	S − D
3	S × S, D × D	24	Input
4	D × S	25	Output
5	S × D	26	Normalise
6	D × M	27	Convert
7	S × M	28	Stop
8	M × D	29	Enter Machine Orders
9	M × S	30	Jump
10	M + D	31	S^{-1} M
11	M − D	32	D^{-1} M
12	D + M	33	S^{-1} D
13	D − M	34	D^{-1} S
14	S + M	35	D^{-1} D, S^{-1} S
15	S − M	36	M^{-1} M
16	M + S	37	M^{-1} D
17	M − S	38	M^{-1} S
18	M + M, D + D, S + S	39	Transpose in situ
19	D + S	40	Binary Input
20	S + D	41	Binary Output

M = Rectangular matrix D = Diagonal matrix S = Scalar matrix

5. STORAGE

5.1 If the Matrix Scheme normalise instruction is required, R 2501 cannot be stored in B 85 to 87. In this case it will usually be convenient to store it after the matrix data and working space, for instance in B 508 to 510.

5.2 If this part of the Store is already used, space may be obtained by reading the Matrix Programme before R 2501. R 2501 may then be stored in B 70 to 72, overwriting part of the Matrix Scheme Order Read.

5.3 Note that B 42 and B 65 to 69, though used for reading matrix orders, cannot be used afterwards to store R 2501.

6. MONITORING

6.1 If it is required to examine a matrix order it should normally be printed. It is possible to monitor an instruction but this is not recommended.

6.2 At the optional stop in 3.3 the instruction is displayed as shown below:

U 4.0_c	F
4.1_c	m_a
4.2_c	n_a
4.3_m	$N_a + R$
4.4_m	$N_b + R$
4.5_m	$N_c + R$
4.6_c	m_b
4.7_c	n_b
5.0_c	i
2.6_m	R, the data relativiser. R + 1.1 = Address of first matrix data location. Unless the Matrix Scheme has been altered by the programmer, R will be 126.7.

Ferranti Ltd.
London Computer Centre,
21, Portland Place,
LONDON. W.1.

Issue 1
14th March, 1957.
D.M.

File as 502

F E R R A N T I L T D

PEGASUS LIBRARY SPECIFICATION

MAGNETIC TAPE MATRIX SCHEME

This is a version of the Matrix Interpretive Scheme (R 2500) adapted to use magnetic tape as a backing store. All the facilities of R 2500 are available except the normalise instruction, but the number of ordinary matrix locations is reduced from 3070 to 3044 (or 6116 with the 7168 word store).

Names: FLOATING MAGNETIC MATRIX SCHEME MK 4
FIXED MAGNETIC MATRIX SCHEME MK 4

Store: 111 blocks which must be as follows:

4096 word store: B 1.0 to 107.7, B 508.4 to 511.5
7168 word store: B 1.0 to 107.7, B 892.4 to 895.5

Uses: B 0 and the entire Computing Store.

Time: The table of times given in the specifications of R 2500 also applies to R 2502. The time for transferring a matrix to or from magnetic tape depends on the distance the tape has to move to find the first element of the matrix.

The tape will be searched at the rate of 1 location every 2.6 milliseconds (41 milliseconds for a 16 word section).

The matrix will be transferred to or from tape at the rate of one element every 3.6 milliseconds.

A further 140 milliseconds should be allowed for each magnetic tape read instruction and 260 milliseconds for each magnetic tape write instruction.

1. METHOD OF USE

The magnetic tape matrix scheme does not carry out matrix operations directly on matrices stored on magnetic tape, but merely uses the tape as a backing store.

Each magnetic tape may be regarded as a continuous strip of some 180,000 storage locations numbered from 1 to 180,000. Matrices may be stored on it in much the same way as they are in the 3044 Main Store matrix locations.

If matrices are too large to be operated upon in the 3044 (or 6116) ordinary storage locations, they must be partitioned. The appropriate sub-matrices of the two operands may be selected from two tapes and the results written on a third tape. Sometimes it is more convenient to operate on one column at a time.

The user need not be concerned with tape sections unless he wishes to use machine orders to operate on tape. Section 0 of tape is not used, and matrix address 1 on tape corresponds to the first word of section 1: section 1, block 0, position 0.

2. MAGNETIC TAPE INSTRUCTIONS

The symbol n is used to denote a magnetic tape function and there is no normalise instruction. The magnetic tape instructions are as follows:

Main Store to Tape

$$(N_a,\ m \times n)n(T) \rightarrow N_T$$

Tape to Main Store

$$(T)n(N_T,\ m \times n) \rightarrow N_c$$

where T = $64t$ (where t = 1 + tape mechanism number)

N_T = Magnetic tape address ($1 \leqslant N_T \leqslant 180{,}000$)

N_a, N_c are Main Store matrix locations

$m \times n$ or $m/$ Denote the dimensions of the rectangular or diagonal matrix to be transferred. Scalar matrices may be denoted by $(N_a,\ 1/)$ but not by (N_a).

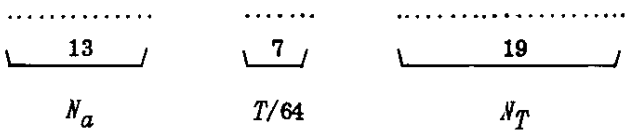
3. PRESET PARAMETERS

Preset parameters may be used with the magnetic tape matrix scheme, but if they are used all magnetic tape addresses greater than 5000 must be preset.

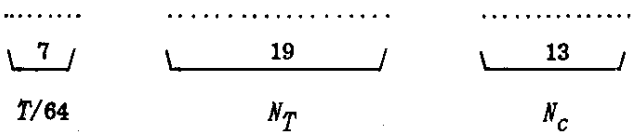
4. STORAGE OF INSTRUCTIONS

Magnetic tape addresses may be longer than the 13 bits allocated to ordinary matrix addresses. For this reason T is only allowed to take values which have their last 6 bits zero. N_T may then spill over into these 6 bits, and the first word of the matrix instruction is as follows:

Main Store to Tape



Tape to Main Store



The second word of the instruction is as described in section 10 of the specification of R 2500. The dimensions of a matrix being read from magnetic tape are $m_b \times n_b$, not $m_a \times n_a$, since the marker T occupies the A-matrix position in the instruction. The dimensions of a matrix being written to tape are $m_a \times n_a$.

5. ERROR TRACER

R 2501 (Error Tracer) may be used with the Magnetic Tape Matrix Scheme, but it may not be stored in its usual position, blocks 85 to 87.

The function numbers, F, for magnetic tape instructions are as follows:

$F = 42$ Read matrix from magnetic tape

$F = 26$ Write matrix to magnetic tape

Magnetic tape address greater than 8191 will spill over into T. The correct address may be computed by dividing T by 64, multiplying the remainder by 8192 and adding it to the tape address printed.

6. TAPE ADDRESS MODIFICATION

The tape address of magnetic tape matrix location 1 is normally 16 (section 1, block 0, position 0). It may be altered to N ($0 \leqslant N \leqslant 180,000$) by reading in the following sequence after R 2502:

T 85.0

+(N - 1)

London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 2
14th April, 1959.
J.F.D. D.M.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 2
10.12.57.

SOLUTION OF SIMULTANEOUS EQUATIONS

This subroutine solves the set of simultaneous linear equations

$$\sum_{j=0}^{n-1} a_{ij} x_j = b_{i0}, b_{i1}, \ldots\ldots b_{im-1} \qquad i = 0 \text{ to } n-1$$

in n variables with m sets of right hand sides, by triangulation and back substitution.

Name: SIM EQUNS.

Store: 15 blocks

Uses: The whole Computing Store; m Main Store locations in addition to the space occupied by the equations. (See section 1.4).

Cues: 01 - normal entry, see section 1.2
02 - see section 1.3

Time: Approximately $(3n + 8m)(n^2 + 7n + 5)$ milliseconds. If the matrix of coefficients is badly conditioned the time will be increased.

Link: Obeyed in U 5.4. Not preserved in X1.

1. Method of use

1.1 The elements of each equation should be stored consecutively in the Main Store, all right hand sides for an equation immediately following the coefficients, as follows:

$$a_{00}, a_{01}, \ldots\ldots a_{0n-1}, b_{00}, b_{01}, \ldots\ldots b_{0m-1}$$

1.2 If cue 01 is used, and if $m + n \geqslant 8$, the equations should be stored consecutively in the Store, i.e. with the first element of one equation immediately following the last right hand side of the previous equation. If $m + n < 8$, the equations must start at the beginnings of successive blocks. In this case, the registers between the end of one equation and the start of the next will be unaltered by the subroutine.

1.3 If cue 02 is used, the equations may be at regular intervals in the Store. The separation, i.e. the distance between leading coefficients of successive equations, should be set in 2_m. If $m + n < 8$, the separation of the equations must be at least 8.

1.4 After the last equation the next m locations are required for storing scale factors.

1.5 Before entry the accumulators should be set as follows:

	Modifier	Counter	
X1	L I N K		
X2	Separation	0	(cue 02 only)
X3	Address of a_{00}	0	
X4	n	0	
X5	$m + n$	0	

2. Preset Parameter

R 511 requires one preset parameter; this should specify the action required if the matrix of coefficients is singular. Preset Parameter 01 will be obeyed in U 2.0 during back substitution if the triangulation process has produced a zero diagonal element. The matrix of coefficients will have been over-written by the triangulated form (see section 3.2); and 4_m will contain $(A_{ii} - 8)$ where A_{ii} is the address of the zero element.

The parameter list should normally be punched as follows:-

R 0 0 -0 1 511 - 04 -	Title
2.0 0 60	Loop Stop or cue to 'singularity' routine.
0	

3. Results

3.1 Each solution is written by the subroutine in place of the corresponding right hand side. The solutions are scaled down if necessary and the m locations after the last solution contain the scaling factors, in the same order as the solutions. Each such location contains, in the modifier position, the number of binary places by which the corresponding solution has been shifted down.

3.2 The matrix of coefficients is replaced by a triangulated form of the matrix, with zeros below the diagonal; rows may have been transposed during triangulation since at each stage of the process the largest leading coefficient is selected. The $n-1$ zeros of the last row of the triangulated matrix will have been over-written.

3.3 On exit the contents of the accumulators will be as follows:-

	Modifier	Counter
X2	Address of first scaling factor (= 1+ address of $b_{n-1,m-1}$)	0
X3	Address of first element of first solution (= address of b_{00})	0
X4	n	0
X5	$m + n$	0
X6	The negative modulus of the least (in modulus) diagonal element of the triangulated matrix.	

4. Method

4.1 The routine uses a standard method of elimination (triangulation) and back-substitution.

4.2 During the triangulation process equations are exchanged in order to select the largest pivot at each stage.

4.3 During back substitution each right hand side is scaled separately so that overflow is prevented with minimum loss of accuracy.

5. Error

The accuracy of the solutions cannot easily be predicted in advance, but an estimate can be obtained from the diagonal elements in the triangulated matrix. The smallest of these elements is left in X6 on exit from the routine. If there are k significant binary digits in this number, the solutions should be accurate to about $k - \log_2 n$ significant binary figures.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
London, W.1.

Issue 2
10th December, 1957.
J.A.F.

R 2530

File as 530

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
20.5.57.

LATENT ROOTS PROGRAMME

A programme for finding up to 16 real roots and vectors of matrices of order up to 54.

CONTENTS

1. METHOD

The basic process is as described by J.H. Wilkinson in Proc. Camb. Phil. Soc. 1954. Briefly, it involves repeated multiplications of an approximation to a vector by the matrix leading ultimately to a sufficiently good vector. This particular vector and the corresponding root are then removed, (using Method B, Page 548 of Wilkinson's paper), and iteration for the next root and vector begins. Since the order of the matrix is reduced by one each time a root is removed, all vectors after the first are degenerate, and to obtain the true latent vectors of the original matrix a back-substitution process is necessary when all the roots required have been found.

Since virtually all the time used by the programme is spent doing a matrix times vector multiplication, it is important that this part of the programme should be as fast as possible. To this end the drum access time has been minimised by suitably arranging the matrix and vector in the store. This leads to a certain amount of wastage of storage space and so to a more severe restriction on the order of matrix which can be dealt with than would otherwise be the case.

The programme itself falls into three main parts:-

(1) Input of the matrix

(2) Iteration, root removal etc.

(3) Back-substitution.

There are three programme tapes corresponding to these parts.

2. INPUT OF THE MATRIX

The matrix should be punched up in matrix scheme notation, (see R 2500) and may be by rows or by columns. It is read by a modified version of the matrix scheme, and a steering tape consisting of 3 or 4 matrix instructions is required.

The steering tape should be as follows:-

D

N

Name

J 64.0

$(0,\ n \times n) \rightarrow N$

$(N,\ n \times n)\ /\ \rightarrow N$ (Omit if matrix is punched by rows)

$(N,\ n \times n)\ v\ (1) \rightarrow N$

O 32

S

N is given by the following table:-

Dimension n	Address N
54	153
51-53	257
41-50	569
34-40	1465

For n less than 34, R 2532 should be used, but R 2530 will work (with $N = 1465$).

The input programme tape is read in and will come to a Z stop. Put steering tape in reader and RUN.

Optional stop in 0.2 (decimal input) or 0.0 (binary input), as in matrix interpretive scheme.

Put matrix in reader and RUN.

The notes on data input in the specification to R 2500 apply here.

When the matrix has been read there is a loop stop in 0.2 after several seconds of programme (during which the matrix is rearranged in the special form required).

If the matrix is in fact in the store of the computer rather than on tape, it is not necessary to output and re-input. The matrix must however be left in the appropriate location as given above, and the data input instruction should be omitted from the steering tape. The latent roots input programme must still be read in because the machine orders required are not part of the normal matrix scheme.

3. THE MAIN PROGRAMME

The main iteration programme should now be read in.

Tape ends with 'E' stop.

RUN

77 stop in 0.1 if handswitches not clear. (Clear them and RUN if this happens).

Enters iteration loop for first root.

There is an optional stop in 3.7 encountered just after each iteration. It should normally be inhibited once the iteration has started.

The programme is now controlled by the setting of the handswitches which are examined after each iteration. The following facilities are available.

Handswitch Setting H 0 12345678	
0	Normal iteration. There is an optional stop in 3.7 in the loop, which will usually be inhibited.
1 0.....	Apply Aitken process. (Comes to 77 stop in 3.2 after a few characters printing. Clear H.S. and RUN).
1 10.....	Not now used. If set up it will cause some printing but should not interfere with the working of the programme.
1 110.....	Print current approximation to eigenvalue.
1 1110....	Remove root, and proceed to find next root. (Comes to 77 stop in 0.4, after printing out eigenvalue). Set sign bit of H.S. = 0 if it is desired to print the vector (degenerate after the first one) at this stage. The vector will not be normalised. After a few seconds programme, comes to 77 stop in 0.0 if H.S. not now clear. Clear if necessary and RUN. Enters iteration for next root.
1 11110...	Print current approximation to vector; the vector will not be normalised.
1 111110..	Prepare for back-substitution. Comes to loop stop in 1.5+.

4. THE BACK-SUBSTITUTION PROGRAMME

The last root required should be allowed to converge in the normal way. Then back-substitution should be called for and not the root removal. The loop stop in 1.5+ will occur almost immediately.

The back-substitution programme tape should then be read in by the Initial Orders, and ends with an E sequence. On operating the RUN key, the roots will be output as a vector, in a form suitable for re-input by the matrix scheme. There will be a title 'ROOTS' preceded by an arrow. After the roots will be about 6 inches of blank tape followed by a matrix of latent vectors, each one normalised to have its largest element unity. These will be preceded by a title 'VECTORS', and are suitable for re-input as a matrix if it is desired to run a checking programme. The whole process ends with a loop stop when the last vector has been output.

During the printing of vectors in the back-substitution, row numbers will normally be punched, but may be suppressed by setting HO = 1.

5. MONITORING

After each iteration, the current approximation to the vector is compared with the previous one, and the following expression is computed.

$$\sum_{i=1}^{n'} (x_i^a - x_i^{a-1})^2$$

where x_i^a is the i^{th} element at iteration a. n' is the order of the matrix upon which the iteration is being performed.

This is a double-length quantity, and will become gradually smaller as the vector converges. The first half of this quantity is displayed in U 0.7 for as long as it is non-zero, and is then replaced by the second half. Which half is in fact being displayed can be recognised by the fact that the second half has a one in the sign digit position. This sign digit has no numerical significance, both halves being essentially positive.

This quantity will in general not come down to absolute zero, as it is not possible to obtain more accurate digits in the vector elements than are being kept in the root.

The root itself is available for monitoring in a scaled-down form in U 0.0. It is usually possible to get this to converge to the last binary digit, but in some cases it will oscillate between two adjacent values.

6. AN ADDITIONAL FACILITY FOR SPEEDING CONVERGENCE

In addition to the Aitken process, described in Section 7 below, a further facility for speeding up convergence is available. It is a modification of the process described on Page 537 of Wilkinson's paper. A fraction, ρ, may be set up on the six least significant handswitches (14-19), with 14 as its sign-digit, and the following iteration will then be performed:

$$(A - \rho \lambda I)x.$$

This has the effect of moving the origin so that, if ρ is reasonably well chosen, the effective convergence ratio is improved. The reason for the factor λ (which is the current approximation to the root) is to make it unnecessary for the operator to consider the scaling inside the machine.

7. GENERAL REMARKS ON OPERATION

7.1 The Aitken Process

This is the most useful facility for speeding convergence. It should not be applied before the root has begun to settle and the quantity in 0.7 has begun to decrease steadily. Once it has been used a few normal iterations should be allowed before it is employed again. It will not normally be of any assistance once the root has almost converged, as it will by then be operating mainly on round-off errors.

7.2 Change of Origin

This is the facility described in Section 6.

It is not possible to give any simple rules for establishing a good value of ρ, but if a root is converging slowly and the Aitken process is not proving very helpful a certain amount of experiment is worthwhile. The best value is frequently somewhere around 1/4 or 1/3.

7.3 Root-removal

To ensure maximum accuracy a root should only be removed when the quantity in 0.7 has become zero. However it is not always possible to get this to happen, and if this quantity becomes stationary or cycles it is probably best to remove the root. A cycle may be broken by using the change of origin facility, but this will almost certainly cause the quantity to increase at first. After a few iterations it should start to come back again and may drop into the same cycle. Thus one can often play with the quantity for some time without achieving anything, or if one is lucky it may eventually become zero. On the later roots especially it is probably not worthwhile spending a long time attempting to get absolute convergence.

7.4 Complex and equal roots

If at any stage a complex root is next in order of magnitude of modulus, the quantity in 0.7 will not converge. The vectors corresponding to any roots already obtained can be got by entering the back-substitution process; the last root and vector printed in this case will be nonsense. Care must be taken to avoid confusing this case with that of almost equal roots, which will show similar effects at first, but should eventually converge. There comes a point however where the roots are too close for the process to converge at all, although they may be distinct in theory.

7.5 Hand-key operation

The programme is liable to examine the hand-keys at any time (as far as the operator is concerned) and as one cannot guarantee to move several keys absolutely simultaneously, care is necessary to avoid unwanted combinations being read. A combination should be set up from right to left e.g. if the keys (excluding the six right hand ones, which control the change of origin facility) are clear, and one wishes to do root removal, it is essential that H0 (the sign-bit) is depressed last. Conversely, the keys should be cleared from left to right and H0 should be raised first.

7.6 Errors in operation

If the combination 110.... is read, the letter C followed by a meaningless number will be printed; no other harm will result.

If 1111111.... is read there will be a 77 stop in 5.7. The keys should be re-set to a permissible combination and RUN.

8. PRINTING

During the main programme there is a certain amount of printing so that some indication of what facilities were used is available.

This printing is as follows:-

Handswitch Setting H 0 12345678	
1 0.....	'A' followed by the number of the iteration at which the process was applied.
1 110.....	'E', followed by iteration number and the eigenvalue.
1 1110....	'R', followed by iteration number. 'L', followed by root number, and the root.

9. TIMES OF OPERATIONS

It is impossible to give the time likely to be taken for a given size of matrix and number of roots required with any accuracy, as it depends far too much on the actual values obtained. Some information can be given however which will lead to an order of magnitude figure.

Some components of the overall time are reasonably predictable, thus

> Programme input (3 tapes), say 3 minutes.
> Data input time can be obtained from the specification of R 2500.
> Back-substitution is almost all punching time; allow about 0.75 seconds per element, or about 0.67 if row numbers are suppressed.

The remainder of the time will be almost all spent in iteration. The iteration time is given by

$$19.08\ n\left[\frac{n+7}{8}\right] + 43n + 74 \text{ milliseconds.}$$

where [ ] means 'the integral part of'.

Specimen times are

n	Iteration time in seconds
8	0.56
10	0.89
20	2.00
24	2.49
30	3.65
40	5.61
50	8.90
54	9.71

The crux of the matter is how many iterations are required per root. This depends mainly on the ratio between the root being found and the next one in order of magnitude, but also partly on whether one gets a very small or zero quantity in 0.7 without undue trouble. In practice with a well behaved matrix one might expect to run about 30 iterations per root, but large variations in this figure must be expected.

On this basis, typical overall times might be

> All roots of 14 x 14 matrix. 10 - 12 minutes.
>
> First five roots of 30 x 30 matrix. 15 - 20 minutes.
>
> First five roots of 40 x 40 matrix. 25 - 30 minutes.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
20th May, 1957
J.F.D.

R 2532

File as 532

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
20.5.57.

LATENT ROOTS PROGRAMME

A programme for finding up to 16 real roots and vectors of matrices of order up to 33.

This is the same programme as R 2530 adapted to have only one programme tape. The specification to R 2530 should be read first. The following differences should then be noted.

The steering tape will be as in R 2530, but N will be 953.

The programme tape should be read in and will come to a Z stop (after obeying a J sequence at the end of the tape).

The steering tape and data tape are read as in R 2530, but the iteration will be entered immediately without the loop stop in 0.2.

Iteration proceeds as before until back-substitution is called for. There will then be no loop stop in 1.5+, and the roots and vectors will be printed out immediately. After this the programme will come to a Z stop and be ready for a further steering tape if required.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
20th May, 1957
J.F.D.

R 7534

File as 534

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

7168 LATENT ROOTS PROGRAMME

A programme for finding up to 16 real roots and vectors of matrices of order up to 54.

This is the same programme as R 2530 adapted to have only one programme tape. The specification to R 2530 should be read first. The following differences should then be noted.

The programme tape should be read in and will come to a 77 stop in 0.5

0	0 77
0.1	0 60

after obeying a J sequence at the end of the tape.

The steering tape and data tape are read as in R 2530, but the iteration will be entered immediately, without the loop stop in 0.2.

Iteration proceeds as before until back-substitution is called for. There will then be no loop stop in 1.5+, and the roots and vectors will be printed out immediately. After this the programme will come to the 77 stop in 0.5, ready to read a further steering tape if required.

In addition to the store used by R 2530, R 7534 uses B750.0 to B882.7.

London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
9th March, 1960.
M.My.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
28.5.57

INVERT SYMMETRIC MATRIX (FLOATING POINT)

A subroutine which replaces a symmetric matrix by its inverse.

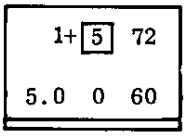
Name: INV SYMM MAT

Store: 29 blocks (including optional parameter list.)
Also working space; see section 1 below.

Uses: The whole Computing Store.

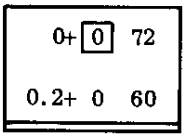
Cues:

01



Working space taken as following the matrix.

02



Working space specified on entry.

Time: About $14n^3 + 50n^2 + 111n + 1000$ milliseconds, where n is the order of the matrix.

Link: Obeyed in 1.7. Not left in X1 on exit.

1. METHOD OF USE

1.1 For either cue, n must be set in 7_C and the address of the first element of the matrix in 6_m; the rest of X6 and X7 must be clear.

1.2 The elements of the matrix should be in standard floating point form.

1.3 As the matrix is symmetric, only $\frac{1}{2}n(n+1)$ elements should be stored. If this is regarded as being the upper triangle of the matrix, the elements must be stored by columns.

For example the matrix

$$\begin{matrix} 1 & 3 & 11 \\ 3 & 2 & 9 \\ 11 & 9 & 6 \end{matrix}$$

would be stored as 1, 3, 2, 11, 9, 6.

1.4 Two sets of working space are required by R 550:-

a) Consisting of $\frac{1}{2}n\,(n-1)$ locations, not necessarily starting at the beginning of a block,

and b) Consisting of the integral part of $\frac{1}{4}\,(n+3)$ whole blocks.

1.5 When cue 01 is used, (a) is taken as starting immediately after the last stored matrix element, and (b) begins at the next available whole block after (a).

1.6 When cue 02 is used, the address of the first location in (a) must be placed in 5_m and the address of the first location in (b) must be placed in 4_m. The remainder of X4 and X5 must be clear. If the address in 4_m is not the beginning of a block, the subroutine will advance it to the beginning of the following block.

2. PRESET PARAMETER.

2.1 R 550 works entirely in single-length floating-point arithmetic as described in the Pegasus Programming Manual and in the specifications of R 11, R 610 and other floating point subroutines.

2.2 The number of binary digits used to represent the exponent of the floating-point numbers is specified by a parameter list. If no parameter list is supplied by the programmer, it will be set as 9 by an optional parameter list. If it is to have some value, n, other than 9, a parameter list of the following form must be supplied:

R0 0 -0 1
550 - 04 -
-2^{n-1}

3. METHOD OF INVERSION

3.1 The original symmetric matrix, A, is first resolved into the product of two triangular matrices, L and M; where L has all non-zero elements on or below the diagonal and M has unit diagonal elements and all other non-zero elements above the diagonal.

$$A = L.M$$

Only $\frac{1}{2}n\,(n-1)$ off-diagonal elements of M are stored.

3.2 The elements of A^{-1} can then be deduced by a back-substitution process from the equations given by

$$M.\,A^{-1} = L^{-1}$$

Since the matrix is symmetric, it is only necessary to use the diagonal elements of L^{-1}, which are easily found.

4. ERROR STOPS

4.1 There are five loop stops which may occur in R 550. If any of these or any writing with overflow stops are encountered, it will normally indicate that non-floating-point numbers are being operated on.

4.2 The loop stops are as follows:-

(a) 0.1 0 60. Floating point overflow on division during triangulation.

(b) 0.6+ 7 61. Floating point overflow during back-substitution.

(c) 1.0 2 61. Floating point overflow in forming an element of L.

(d) 1.0+ 1 60. Zero diagonal element of L. (Matrix singular).

(e) 1.0+ 7 61. Floating point overflow in forming an element of M.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
28th May, 1957
J.F.D.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
4.6.57.

FLOATING POINT MATRIX DIVISION

A subroutine to evaluate the matrix $A^{-1}B$ where A and B are matrices in standard floating point form in the Main Store.

Name: F.P. MATRIX DIVISION

Store: 14 blocks.

Uses: The whole Computing Store except X 1, 2, 3, 6 and 7. B0.

Cue:

01

```
0+[1] 72
1.5  0 60
```

Time: $19n^2r + 8.5n^3 + 118n^2 + 181n$ milliseconds.

Link: Obeyed in 1.1, and left unaltered in X1 on exit.

1. METHOD OF USE

Let the matrix A have dimensions $n \times n$, and B $n \times r$. Both matrices must be stored by rows, requiring n^2 and $n.r$ locations respectively. The position of either matrix in the store is thus determined by the address of its first element. The addresses, A and B, of A and B must be placed in the modifier positions of X4 and X5 respectively before entry, with the counter positions clear. n and r must be placed as integers in X6 and X7 respectively. Thus on entry the accumulators will contain:

X	
1	LINK
2	-
3	-
4	(A, 0)
5	(B, 0)
6	$+ n.2^{-38}$
7	$+ r.2^{-38}$

The result $C = A^{-1} B$, (which is of dimensions $n \times r$) will replace B. A will be destroyed.

2. METHOD OF DIVISION

The programme is an adaptation of the division sequence in the Matrix Interpretive Scheme (R 2500). The method used is pivotal condensation, with elimination

of variables above as well as below the pivotal row; thus no back-substitution process is necessary. The programme finds and uses the largest available pivot in the pivotal column at each stage.

3. FLOATING POINT REPRESENTATION

The standard floating point form, with 9 bits for the exponent (augmented by 256) and 30 bits for the argument is used.

4. STOPS

a) A loop stop in 1.3 (1.3 2 60) occurs if division by zero is called for. This implies that A is singular.

b) Various writing with overflow stops may occur if non-standard floating point numbers are encountered.

c) If any of the parameters are wrongly set, the subroutine may cycle indefinitely.

5. ACCURACY

Standard floating point representation gives a nominal accuracy of slightly less than 9 decimal digits. In general some of these digits will be lost due to rounding errors, and these errors will increase with the dimensions of the matrices involved. However, a much more potent factor is how well or badly conditioned the matrix A happens to be, and if it is nearly singular the subroutine may produce meaningless results, possibly including numbers with very large exponents. This is much more common in practice than an exactly singular matrix, which would lead to the loop stop mentioned in section 4 above.

Ferranti Ltd.,
London Computer Centre,
21 Portland Place,
LONDON, W.1.

Issue 1
4th June, 1957.
J.F.D.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
28.5.57.

MULTIPLY SYMMETRIC MATRICES (FLOATING POINT)

A subroutine which forms the product of two symmetric matrices.

Name: MULT SYMM MAT

Store: 11 blocks (including optional parameter list).

Uses: The whole Computing Store.

Cue:

01

```
0+ [5] 72
5.0  0 60
```

Time: $26n^3 + 66n^2 + 25$ milliseconds, where n is the order of the matrices.

Link: Obeyed in 0.4. Not left in X1 on exit.

1. METHOD OF USE

1.1 n, the order of each matrix, must be set in 7_C. If A and B are the two symmetric matrices, the product C = AB is formed. The addresses of the first elements of A, B and C should be set in 4_m, 5_m and 6_m respectively. The rest of X 4, 5, 6 and 7 must be clear.

1.2 All elements should be in the normal floating point form.

1.3 For the two symmetric matrices only $\frac{1}{2}n\,(n + 1)$ elements should be stored. If they are regarded as upper triangles, the elements must be stored by columns.

For example the matrix

$$\begin{matrix} 1 & 3 & 11 \\ 3 & 2 & 9 \\ 11 & 9 & 6 \end{matrix}$$

would be stored as 1, 3, 2, 11, 9, 6.

1.4 The product matrix C is not symmetric, and will therefore require n^2 locations. It will be stored by columns.

2. PRESET PARAMETER

2.1 R 560 works entirely in single-length floating-point arithmetic as described in the Pegasus Programming Manual and in the specifications of R 11, R 610 and other floating point subroutines.

2.2 The number of binary digits used to represent the exponent of the floating-point numbers is specified by a parameter list. If no parameter list is supplied by the programmer, it will be set as 9 by the optional parameter list. If it is to have some value, n, other than 9 a parameter list of the following form must be supplied:

R0	0	−0	1	
560	−	04	−	
-2^{n-1}				

3. ERROR STOP

0.3+ 7 61	Floating-point overflow.

If this stop or writing with overflow occurs, it will normally indicate that non-floating-point numbers are being operated on.

Ferranti Ltd,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
28th May, 1957
J.F.D.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
27.1.58.

TRANSPOSE MATRIX IN SITU - Floating Point

This subroutine transposes in situ a rectangular matrix stored in floating-point form, or in fixed-point form subject to certain restrictions.

Name: TRANSPOSE MATRIX IN SITU-F.P.

Store: 4 blocks.

Uses: U0, 1, 2; B0.

Cue:

01

```
0+2 72
2.0  0 60
```

Time: (44 mn + 93) milliseconds for an $m \times n$ matrix.

Link: Obeyed in 0.0 and left unaltered in X1.

Method of Use

The matrix has dimensions $m \times n$ (m rows, n columns). It should be stored by columns and its position in the Main Store is given by the address A of its first element. A must be placed in 5_m with 5_c clear. m and n must be placed as integers in X6 and X7 respectively, with 6_m and 7_m clear.

The subroutine may be used only for matrices made up of standard floating-point numbers or fixed-point numbers in the range $-\frac{1}{2} \leqslant x < \frac{1}{2}$. It makes use of the fact that the first two binary digits of such numbers are the same.

Notes

1. The behaviour of the subroutine, if any of the elements are not in the form specified, cannot easily be predicted. It will almost certainly produce wrong results and take considerably longer than the specified time, and it may be possible for it to cycle indefinitely in some cases.

2. The subroutine can be used for transposing a matrix stored by rows by interchanging the parameters m and n.

3. The routine examines only the first two digits of each number and is not affected by the number of bits used to represent the exponent.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
27th January, 1958.
J.F.D. J.A.H.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 2
17.6.58.

PEGASUS AUTOCODE

A conversion scheme to facilitate the programming of certain types of problem. It is especially suitable for such technical and scientific calculations as the evaluation of formulae, the tabulation of special functions and small ad hoc calculations; it can also be used in commercial and industrial work for the testing of complex flow-digrams. The user of the Autocode need not know anything of ordinary Pegasus programming.

Name: AUTOCODE MK 1

Store: B1 + 110 blocks.

Uses: The entire Computing Store; locations in Main Store for labels, indices, variables and instructions. (See section 12).

Cues:

01	35+ 4 72 4.4 0 60	to enter Autocode programme from machine orders (see section 9).

Cues 02 and 03 are partial cues containing the address of the first block of R600 (other than B1) in the address parts of the a-order and b-order respectively.

Time: Input approximately 2 instructions per second. Programme obeyed at approximately 15 instructions per second.

CONTENTS

1. Autocode Programming

To use the Autocode on a particular problem the calculation must first be broken down into a sequence of steps, each of which is the calculation of a number from one or two previously calculated numbers. Each step is written as an *instruction* and the whole sequence forms the *programme* of the calculation. The programme is punched, as written, on a teleprinter or a keyboard perforator and the resulting tape is then read into the computer by the Autocode scheme (instruction input section) which converts the programme into a suitable internal form which is stored. At the end of the tape are some special symbols which cause the programme to be obeyed. At various stages while it is being obeyed the programme can cause printing of results (they are actually punched and printed later) or it can read in numerical data from tapes in either of the tape readers.

Two types of number are handled by Autocode instructions:

(a) *Variables* denoted by $v0$, $v1$, $v2$, $v3$, ... Variables as large as 10^{76} or as small as 10^{-76} are held with a precision of between 8 and 9 significant decimal digits. Variables smaller than 10^{-76} (approximately) are treated as zero.

(Variables are stored as standard, packed, floating-point numbers with 9 bits for the exponent and may therefore lie in the range

$$-2^{254} \leqslant v \leqslant 2^{254}(1 - 2^{-28}).$$

Numbers in the range

$$-2^{-257} \leqslant v \leqslant 2^{-257}(1 - 2^{-28})$$

are stored as zero.)

(b) *Indices* denoted by $n0$, $n1$, $n2$, $n3$, ...

These are signed integers in the range

$$-8191 \leqslant n \leqslant 8191.$$

They are stored in the modifier position of words and are handled in fixed point form.

Most of the numbers entering into a calculation are variables; the indices are introduced as auxiliary quantities, mostly to facilitate counting and modifying. The v and n symbols are available in figure-shift on the teleprinter; the digits which follow them are to be thought of as suffixes although they cannot be printed as such on the teleprinter. The total available numbers of variables and indices are adjustable (see section 12) but they are normally fixed so that there are

(a) 1380 variables $v0$, $v1$, ..., $v1379$
(b) 28 indices $n0$, $n1$, ..., $n27$.

2. Arithmetic Instructions

Most of the Autocode instructions take the form of an equation giving the new value of a variable (or index) in terms of one or two numbers or previously calculated variables (or indices). For example, the instruction

$$v1 = v2 + v3$$

means that the new value of $v1$ is to be the sum of $v2$ and $v3$. The instruction

$$v5 = v5 + v1$$

means that the new value of $v5$ is to be the sum of $v1$ and the old value of $v5$. Numbers may be put instead of variables on the right of the equality sign; thus the instruction

$$v8 = v2 + 35.771$$

means that the new value of $v8$ is the sum of $v2$ and 35.771. As an example of a sequence of these instructions dealing with variables let us evaluate

$$31.41\ v5 - (v92)^2 + \{(-6.535\ v8 + v97 - 9)/v323\}$$

and put the result in $v0$. The following sequence of instructions can be used (here we use $v1$ as working space):

$$\begin{aligned} v0 &= 31.41 \times v5 \\ v1 &= v92 \times v92 \\ v0 &= v0 - v1 \\ v1 &= -6.535 \times v8 \\ v1 &= v1 + v97 \\ v1 &= v1 - 9 \\ v1 &= v1/v323 \\ v0 &= v0 + v1 \end{aligned}$$

Note that each instruction involves only two variables or numbers on the right. We can also use instructions like the following ones, which have only one variable or number on the right:

$$v8 = v4, \qquad v6 = -v2, \qquad v6 = 43$$

There are similar instructions for handling indices, for example

$$\begin{aligned} n3 &= n8 + n3 & \qquad n2 &= -79 - n5 \\ n0 &= 288 & \qquad n4 &= -1971 \end{aligned}$$

but it should be remembered that indices can take only integral values. In general indices and variables cannot be mixed in the same instruction; but a few simple instructions of this type have been provided.

The purely arithmetic instructions are summarised in Table 1. In this table $v1$, $v2$ and $v3$ represent any three variables and similarly $n1$, $n2$ and $n3$ any three indices.

Variables		Indices	
$v1 = v2$	$v1 = -v2$	$n1 = n2$	$n1 = -n2$
$v1 = v2 + v3$	$v1 = -v2 + v3$	$n1 = n2 + n3$	$n1 = -n2 + n3$
$v1 = v2 - v3$	$v1 = -v2 - v3$	$n1 = n2 - n3$	$n1 = -n2 - n3$
$v1 = v2 \times v3$	$v1 = -v2 \times v3$	$n1 = n2 \times n3$	$n1 = -n2 \times n3$
$v1 = v2/v3$	$v1 = -v2/v3$	$n1 = n2/n3$	$n1 = -n2/n3$
		$n1 = n2 * n3$ (rem)	$n1 = -n2 * n3$
Mixed: $n1 = v2$	$n1 = -v2$ (nearest integer)		
$v1 = n2$	$v1 = -n2$		
$v1 = n2/n3$	$v1 = -n2/n3$		

Table 1 Autocode instructions - arithmetic

The instruction $n1 = n2/n3$ gives the integral quotient when $n2$ is divided by $n3$ and the instruction $n1 = n2 * n3$ gives the corresponding remainder (zero or with the same sign as $n3$). The analogous instructions with a minus sign give the same numbers with their signs changed. In any of the instructions any variable (or index) may be replaced by a positive number (or integer).

3. Functions

Certain elementary *functions* may be evaluated by a single instruction; for example we can write

$$v6 = \text{SQRT } v9$$

to evaluate a square root, or

$$v3 = -\text{SIN } v99$$

to evaluate a sine. The functions available are listed in Table 2; they all apply to variables only, except for MOD which can be used to form the modulus of an index or a variable. Again any variable on the right may be replaced by a positive number; for example

$$v37 = \text{EXP } 5.1058809$$

$v1 = \text{MOD } v2$	$v1 = -\text{MOD } v2$	modulus
$v1 = \text{INT } v2$	$v1 = -\text{INT } v2$	integral part
$v1 = \text{FRAC } v2$	$v1 = -\text{FRAC } v2$	fractional part
$v1 = \text{SQRT } v2$	$v1 = -\text{SQRT } v2$	square root
$v1 = \text{SIN } v2$	$v1 = -\text{SIN } v2$	circular functions
$v1 = \text{COS } v2$	$v1 = -\text{COS } v2$	
$v1 = \text{TAN } v2$	$v1 = -\text{TAN } v2$	
$v1 = \text{CSC } v2$	$v1 = -\text{CSC } v2$	
$v1 = \text{SEC } v2$	$v1 = -\text{SEC } v2$	
$v1 = \text{COT } v2$	$v1 = -\text{COT } v2$	
$v1 = \text{ARCSIN } v2$	$v1 = -\text{ARCSIN } v2$	inverse circular functions
$v1 = \text{ARCCOS } v2$	$v1 = -\text{ARCCOS } v2$	
$v1 = \text{ARCTAN } v2$	$v1 = -\text{ARCTAN } v2$	
$v1 = \text{LOG } v2$	$v1 = -\text{LOG } v2$	natural log
$v1 = \text{EXP } v2$	$v1 = -\text{EXP } v2$	exponential
$v1 = \text{EXPM } v2$	$v1 = -\text{EXPM } v2$	exp ($-v2$)
$n1 = \text{MOD } n2$	$n1 = -\text{MOD } n2$	modulus

Table 2 Autocode instructions - functions

4. Jump Instructions

As usual, Autocode instructions are obeyed in the sequence in which they are written, until a *jump instruction* is encountered. If a jump occurs the next instruction to be obeyed is identified by its *label* which is simply a small positive integer written in front of the instruction and separated from it by a right bracket. For example

$$7)\ v4 = -9.44/v5$$

is labelled 7. Any instruction can be labelled but the same label should not be used twice. If required we can attach two or more labels to the same instruction, thus

$$9)\ 2)\ v3 = \text{LOG } v0$$

The first instruction in an Autocode programme is always automatically labelled 0, there is no need to write this label in.

A jump instruction always includes an arrow; the simplest is the unconditional jump, for example

→7

means jump to the instruction labelled 7.

Consider the following instruction:

$$\rightarrow 8,\ v1 \geqslant v6$$

This means jump to the instruction labelled 8 if $v1 \geqslant v6$, otherwise carry on with the next instruction as usual. The following three instructions resemble this one closely:

$$\rightarrow 8,\ v1 \geqslant -v6$$
$$\rightarrow 8,\ -v1 \geqslant v6$$
$$\rightarrow 8,\ -v1 \geqslant -v6$$

These four instructions can be summarised as

$$\rightarrow 8,\ \pm v1 \geqslant \pm v6$$

All the available jump instructions are summarised in this way in Table 3; as before 1,2,3 represent any three numbers.

→1 (unconditional jump)	
→1, $\pm v2 \geqslant \pm v3$	→1, $\pm v2 > \pm v3$
→1, $\pm v2 = \pm v3$	→1, $\pm v2 \neq \pm v3$
→1, $\pm n2 \geqslant \pm n3$	→1, $\pm n2 > \pm n3$
→1, $\pm n2 = \pm n3$	→1, $\pm n2 \neq \pm n3$
→1, $\pm v2 = * \pm v3$ (jump if approximately equal; more exactly, jump if the two variables agree to $n0$ significant binary digits – i.e. to about $0.3 \times n0$ significant decimal figures) $(1 \leqslant n0 \leqslant 28)$	
→1, $\pm v2 \neq * \pm v3$ (jump if not approximately equal)	

Table 3 Autocode instructions - jumps

In a jump instruction any variable may be replaced by a number and any index by an integer. For example

$$\rightarrow 4,\ 0 > v62$$
$$\rightarrow 8,\ n6 \neq -20$$

Care should be exercised in using the conditional jumps involving variables because decimal fractions are stored in binary form and therefore cannot in general be held exactly. Integers in the range $-2^{28} \leqslant v \leqslant 2^{28}$ may be stored exactly but inaccuracies may arise in calculating them.

The jumps testing approximate equality are intended to allow for the effects of the rounding errors inevitable in most floating-point work; the value of $n0$ must be set before using them (e.g. by an instruction such as $n0 = 20$).

As an example, let us suppose that we are given two positive numbers $v1$ and $v2$ and we have to replace them, respectively by

$$\tfrac{1}{2}(v1 + v2) \text{ and } \sqrt{(v1 \times v2)}.$$

This process is to be repeated until the two quantities are nearly equal, say until they agree to about 6 decimal significant figures. The result is in fact an approximation to the Gauss arithmetico - geometric mean of the two quantities. The following sequence of instructions can be used:

```
n0 = 20
3)v3 = v1
v1 = v1 + v2
v1 = .5 x v1
v3 = v3 x v2
v2 = SQRT v3
→3, v1 ≠ *v2
```

The first instruction sets $n0$ for subsequent 'approximately equal' tests. The variable $v3$ is used to hold intermediate quantities.

5. Modification

The technique of *modification* can be applied to any Autocode instructions involving variables. For example, we can write the instruction

$$v8 = v998 + vn6$$

which means that the new value of the 8th variable is the sum of the 998th variable and the $n6$th variable. The $n6$ in $vn6$ is to be thought of as a suffix (v_{n_6}) though it cannot be typed in this way on a teleprinter. Thus the instruction written above is equivalent to the instruction

$$v8 = v998 + v28$$

if $n6 = 28$ at the time the instruction is obeyed. Any variable can have this kind of suffix, for example

$$vn0 = vn3 + vn4$$

We can also write a more complicated kind of suffix, for example

$$v(8 + n2) = v(53 + n4) + v(-2 + n11)$$

in which each bracketed expression is regarded as a suffix. The integer (h say) in this kind of suffix must be written before the n and can have any value in the range

$$-2048 \leqslant h \leqslant 2047$$

but the result after the addition of the index n must not be negative. The index must be *added*, never subtracted.

Jump instructions can also be modified. The following are examples:

```
→n7, v0 > v67
→n9, -n8 ⩾ 21
→(-4 + n8), n0 ≠ 8
```

6. Input

A group of instructions is provided for the *input of numbers*. A simple one is the following

```
v6 = TAPE
```

This causes $v6$ to be set equal to a number read from the tape in the main tape reader. The instruction

```
v5 = TAPE 13
```

causes 13 numbers to be read and placed in $v5$, $v6$, ..., $v17$.

The instruction

$$v28 = \text{TAPE} *$$

causes numbers to be read and placed in $v28$, $v29$, $v30$, ... until an L-directive is read. The input instructions are given in full in Table 4. It should be noted that *any* input instruction has the following properties

(a) input ceases and the next instruction is obeyed when L is read.

(b) $n0$ is put equal to the number of numbers read in.

(c) Directive N causes the characters following it to be copied on the output tape as a name. As with the Initial Orders the name must be terminated by blank tape.

(d) Directive Z causes the computer to wait on a 77 stop in 0.2. Input may be resumed by operating the STOP/RUN key.

(e) Directive Q sets a decimal block exponent. Q must be followed by an integer $\pm q$ representing a power of 10 by which succeeding variables are to be multiplied. All succeeding variables in that input instruction, but not in later instructions, will be multiplied by 10^q unless a second Q is read. Q can not be used with indices.

$v1$ = TAPE $v1$ = TAPEB	read one number and set in $v1$.
$n1$ = TAPE $n1$ = TAPEB	read one integer and set in $n1$.
$v1$ = TAPE $n2$ $v1$ = TAPEB $n2$	read $n2$ numbers and set in $v1$, $v2$ .... ($n2 > 0$)
$n2$ = TAPE $n1$ $n2$ = TAPEB $n1$	read $n1$ integers and set in $n2$, $n3$, ... ($n1 > 0$)
$v1$ = TAPE * $v1$ = TAPEB *	read numbers up to L on tape, set in $v1$, $v2$, ...
$n1$ = TAPE * $n1$ = TAPEB *	read integers up to L on tape, set in $n1$, $n2$, ...

TAPE instructions refer to the main tape reader, TAPEB to the second tape reader.

Table 4 Autocode instructions - input

Suppose, as an example, that the instruction

$$v1 = \text{TAPE} *$$

is obeyed and that the following tape is in the main tape reader:

```
N
DATA 3
(blank tape)
+1.5
Q + 10     -.657     +0.9876
Q - 15
+0.55332
-13
L
```

When the instruction has been obeyed the following will be stored:

$$v1 = +1.5, \qquad v2 = -0.657 \times 10^{10}, \qquad v3 = +0.9876 \times 10^{10},$$
$$v4 = +0.55332 \times 10^{-15}, \qquad v5 = -13 \times 10^{-15}, \qquad n0 = 5.$$

The name DATA 3 will be printed when the tape is read.

Each number on the tape is punched as written, immediately preceded by its sign and must be terminated by Sp or CR LF. (If indices are to be read in they must be punched without a decimal point).

The TAPE input instructions actually read in numbers from whichever tape reader was last selected. Since all input instructions leave the main reader selected trouble can only occur when the Autocode programme is entered from machine orders or from Initial Orders. In both cases care should be taken to leave the main tape reader selected before entry.

7. Output

There are two main methods of output. The most used is a special instruction of which the following is an example:

PRINT $v6$, 3045

This instruction causes the value of $v6$ to be printed in a way determined by the style-number 3045. To find the style-number to write in a print instruction we must first decide whether the number is to be printed on a new line or on the same line as the previous number and whether it is to appear in floating-point form (i.e. as a number and a decimal exponent) or in fixed-point form (with the decimal point in its proper position). This determines the first digit (a) of the style-number which can be found from the following table:

	Floating-point	Fixed-point
Print on a new line (CR LF ϕ before number)	$a = 1$	$a = 3$
Print on same line (Sp before number)	$a = 2$	$a = 4$

The rest of the style-number is determined by the number of digits (b) to be printed before the decimal point and after it (c); the style-number is

$$1000a + 20b + c$$

so that style 3045 causes the number to be printed in fixed-point form on a new line with two digits before the decimal point and five *after* it. The number is preceded by its sign and is rounded.

If desired an instruction of the form

PRINT $v3$, $n9$

may be written, when the current value of $n9$ is taken as style-number.

A similar instruction may be used for printing indices; here only the first digit of the style number matters and the index is always printed as a 4- digit signed integer. Thus the instruction

PRINT $n3$, 4000

causes the value of $n3$ to be printed, preceded by a single space. In all printing non-significant zeros in the integral part are replaced by spaces, except that the digit before the decimal point is always printed. If $b = 0$ then zero is printed before the decimal point. If $c = 0$ the decimal point is not printed.

An alternative method of output is mainly useful to provide extra printing in the development stage of an Autocode programme. We can write XP (for printing on a new line) or SP (for the same line) *before* an instruction; this causes the result of the instruction to be printed if handswitch 0 on the computer is up when the instruction is obeyed. No printing occurs if this handswitch is down. Thus the instruction

$$\text{XP } v9 = v3 + v5$$

causes the new value of $v9$ to be printed on a new line provided handswitch 0 is in the up position.

To help in laying out the printing an X before an instruction causes printing of CR LF and an S causes printing of Sp, after the instruction has been obeyed. This printing can not be suppressed by handswitch 0.

Only one of XP, SP, X, S may be written before any one instruction and only arithmetic and function instructions can be preceded by XP, SP, X or S.

The output instructions are given in detail in Table 5.

PRINT $v1$, $n2$. Print $v1$ in style $n2 = 1000a + 20b + c$.
If $a = 1$ print CR LF ϕ then number in floating-point form with b digits before point and c digits after point, then Sp and two digit signed exponent, then Sp Sp. Width of printing $8 + b + c$ unless $b = 0$ (width $9 + c$) or $c = 0(7 + b)$.
If $a = 2$ print Sp then number as for $a = 1$. Width $9 + b + c$ unless $b = 0$(width $10 + c$) or $c = 0(8 + b)$
If $a = 3$ print CR LF ϕ then number in fixed-point form with b digits before the point and c digits after. Width $2 + b + c$ unless $b = 0(3 + c)$ or $c = 0(1 + b)$. If b is too small print in floating-point form as if $a = 1$.
If $a = 4$ print Sp then number as for $a = 3$. Width $3 + b + c$ unless $b = 0(4 + c)$ or $c = 0(2 + b)$. If b is too small print as if $a = 2$.

PRINT $n1$, $n2$. Print $n1$ in style-number $n2 = 1000a$
If $a = 3$ print CR LF ϕ then 4-digit index. Width 5.
If $a = 4$ print Sp then 4-digit index. Width 6.

XP before an instruction (arithmetical or functional) the result of which is a *variable*. Obey instruction, then print CR LF ϕ and result in floating-point form as for PRINT with $a = 1$, $b = 0$, $c = 9$. Width 18.

XP before an instruction the result of which is an index. Obey instruction, then print CR LF ϕ and result as four digit integer. Width 5.

SP is similar to XP but Sp is printed instead of CR LF ϕ. Width 19 for variables, 6 for indices.

Note: XP and SP only cause printing if handswitch 0 is up when the instruction is obeyed.

X, S } before any arithmetical or functional instruction. Print CR LF or Sp respectively, after obeying the instruction. This printing is not affected by the setting of handswitch 0.

Table 5 Autocode instructions - output

The following are typical results of some output instructions

PRINT v $n10$, 1064	CR LF ϕ + 345.6789 Sp Sp + 1 Sp Sp
PRINT $v3$, 2064	Sp − 100.0000 Sp Sp + 6 Sp Sp
PRINT $v4$, 2044	Sp + 12.3456 Sp + 12 Sp Sp
PRINT $v9$, 3060	CR LF ϕ − 123
PRINT v $n4$, 3062	CR LF ϕ Sp Sp + 2.36
PRINT $v(1 + n2)$, 4026	Sp + 9.876543
PRINT v $n6$, 4063	Sp Sp Sp + 0.016
PRINT $n2$, 3000	CR LF ϕ Sp Sp + 12
PRINT $n5$, 4000	Sp − 7732
XP $v4 = v3 + v2$	CR LF ϕ + 0.123456789 Sp Sp + 2 Sp Sp
SP $v7 = v9 + 10$	Sp − 0.987654321 Sp + 11 Sp Sp
XP $n3 = 1 + n3$	CR LF ϕ Sp + 103
X $n2 = 0$	CR LF
S $n4 = n7 - n1$	Sp

8. STOP Instruction, Bracketed Interludes, Complete Programme

The instruction STOP

causes a 77-stop in U 5.0. If the run key is operated the next Autocode instruction will be obeyed.

The programme tape for an Autocode Programme is headed as follows:

D
N
(name of programme)
blank tape
J 1.0

The tape up to and including the J-directive is read by the Initial Orders; the name must be terminated by blank tape. The remainder of the tape is read by the Autocode instruction input (the Autocode itself having been read previously). The Autocode instructions making up the programme are punched (as written) in their correct sequence after the J 1.0. Each instruction is converted into a block of machine orders and numbers and stored. About half a second is required to read each instruction. At the end of the programme certain special bracketed symbols must be punched to cause the programme to be entered.

An instruction or a group of instructions written in brackets is obeyed as soon as it has been read. Since the first instruction in the programme is automatically labelled 0 we can enter a programme (i.e. start obeying it) by punching

(→0)

at the end of the tape. This is an unconditional jump to the beginning of the programme and since it is written in brackets it is obeyed the moment it has been read.

A sequence such as the following may be punched:

```
(v1 = 1.76
 n2 = 10
 →3)
```

This sequence is all read and stored but a note is made of the left bracket; when the right bracket is read the instruction after the left bracket is obeyed, then the next, and so on. The last instruction in the example is an unconditional jump to the instruction labelled 3 and the main programme will be entered at this point. Such a sequence of instructions is called a *bracketed interlude*. If the last instruction of the interlude does not cause a jump then further instructions, punched after the interlude on the tape, will be read and will obliterate the interlude. If we wish, the bracketed interlude can be quite a complicated programme including loops and print instructions, for example. In fact, the whole programme could be put in brackets, so that it would be entered when the right bracket is read.

The Autocode uses one block beyond those where the programme has been stored, to record the right bracket and to cause more programme to be read if the last instruction of the interlude does not cause a jump.

As an example of a complete Autocode programme, we give below a programme to read in numbers from a tape in the main tape reader and to evaluate and print the sum of their squares and the square root of this number.

```
D
N
SUM OF SQUARES
J 1.0

v2 = TAPE *                          { numbers to v2, v3, .... and set
v0 = 0                               { n0 = number of numbers
2)v1 = v(1 + n0) × v(1 + n0)
v0 = v0 + v1                         accumulate sum in v0
n0 = n0 - 1                          count numbers
→2, n0 ≠ 0
PRINT v0, 1025                       print sum of squares
v0 = SQRT v0                         form square root
PRINT v0, 2025                       print square root
STOP                                 stop at end of programme
(STOP                                wait for number tape
→ 0)                                 enter programme
```

The next example is an Autocode programme to tabulate the function

$$z = \text{arcsech } y = -\log\left[\{1 - \sqrt{(1 - y^2)}\}/y\right]$$

for $y = 0.01(0.01)0.99$. The current value of y is generated by dividing an integer $n1$ by 100; $n1$ will start at 1 and go up to 99. This is to prevent the accumulation of rounding errors. An alternative procedure would have been to use a variable ranging from 1 to 99 and to divide by 100 to give y. This would also have prevented the accumulation of rounding errors. The numbers y and z are printed in two columns and an extra blank line is inserted (by printing CR LF) whenever y is a multiple of 0.05 (i.e. $n2$ is a multiple of 5).

```
D
N
TABULATE ARCSECH - AUTOCODE

J 1.0

n1 = 1

1)n2 = n1 * 5      } print CR LF if remainder when
→2, n2 ≠ 0         } n1 is divided by 5 is zero
X n2 = 0

2)v1 = n1/100      y = n1/100
PRINT v1,3022      print y
v2 = v1 x v1       y²
v3 = 1 - v2        1 - y²
v4 = SQRT v3       √(1 - y²)
v5 = 1 - v4        1 - √(1 - y²)
v6 = v5/v1         {1 - √(1 - y²)}/y
v7 = -LOG v6       z
PRINT v7,4027      print z
n1 = n1 + 1
→1, n1 ≠ 100
STOP               stop at end of programme
(→0)               enter programme
```

9. Further Facilities

The instruction TAPE

causes more instructions to be read in from the input tape (in the tape reader last selected i.e. the main tape reader unless the programmer has taken measures to select the second reader) and to be added at the end of the programme (in fact they overwrite the last bracketed interlude). This instruction can be used in a programme designed to do a rather complicated calculation on a few numbers or parameters. At the end of this calculation a TAPE instruction can be used to read a bracketed interlude such as

```
(n4 = 3
 v1 = 41.509
 →6)
```

This interlude can set new values of the numbers or parameters and cause the calculation to be repeated.

A variant of the TAPE instruction is written, for example

```
TAPE 6
```

This reads in more instructions and puts the first one in place of the instruction labelled 6. Similarly the instruction

```
TAPE 6,3
```

will place the first of the new instructions over the third instruction after that labelled 6. Instructions such as

```
TAPE n1
TAPE n1, n2
```

can also be used.

It is sometimes desirable to stop obeying an Autocode programme and to start obeying ordinary Pegasus machine orders; after some special calculation has been done the programmer may wish to return to the Autocode programme, either at the next instruction or at some other specified instruction.

The instruction →M 939

means: start obeying machine orders at decimal address 939 (i.e. at B117.3). The effect is similar to a J-directive with decimal address 939 so that B117 will be transferred to U0, the next three blocks to U1,2 and 3 and a jump will occur to 0.3 (a-order). A link is set in X1 and a special word in X2; if the link is obeyed with C(2) undisturbed the computer will return to the next Autocode instruction (i.e. the one after the →M 939). Similary the instruction

→M 939, $v2$

will cause exit to machine orders at decimal address 939 but in addition the variable $v2$ will be unpacked; X7 will contain its argument and X5 will contain 256 + its exponent. Instructions such as

→M $n3$, $v5$
→M 939, $n1$

can also be used. The last will leave $n1$ in 7_m. With any →M instruction the settings of the accumulators are reproduced in B0.

Entry from machine orders to an Autocode programme at the instruction labelled 15 (for example) can be done by putting 15 into X2 as an integer and obeying cue 01 to R 600:

```
 35+ [4] 72
4.4   0 60
```

i.e. normally

```
 37  [4] 72
4.4   0 60
```

Before obeying this cue a link may be set in X1; this is obeyed by the Autocode instruction

→L

Machine orders should normally be punched after the name of the programme but before the J 1.0 entering the Autocode. The space available for storing machine orders may be obtained by reference to section 12. It is usually convenient to use for machine orders the space reserved for the higher numbered variables.

The Initial Orders may be called in as a subroutine by the instruction

→I0

A link is set which causes return to the next Autocode instruction when an L-directive is read, provided the contents of B0.1 and 0.2 are undisturbed. Similarly the instruction

→I0, 1200

calls in the Initial Orders after setting the Transfer Address to decimal address 1200 (i.e. to B150.0), and the instruction

→I0, 1200, 153

will in addition set the Relativiser to 153. Instructions such as the following may also be used

→I0, *n*1
→I0, *n*1, *n*2

These various Autocode instructions are summarised in Table 6.

STOP	Stop(77), proceed with next instruction when RUN key is operated.
TAPE	Read in more instructions and add them at the end of the programme.
TAPE *n*1	Read in more instructions, the first to replace that labelled *n*1.
TAPE *n*1, *n*2	Read in more instructions, the first to replace the instruction *n*2 after that labelled *n*1.
→M *n*1	Exit to machine orders at decimal address *n*1
→M *n*1, *v*2	The same, and leave *v*2 unpacked in X7 and X5
→M *n*1, *n*2	The same but leave *n*2 in 7_m.
→L	Obey link set in X1 on entry from machine orders
→I0	Call in Initial Orders as a subroutine
→I0, *n*1	The same, but set T.A. = *n*1
→I0, *n*1, *n*2	The same and set Relativiser = *n*2

Table 6 Autocode instructions - miscellaneous

10. Re-entry and alterations

At the beginning of the Autocode programme tape the directive J 1.0 is punched; this causes the input programme to treat the following instructions as a new Autocode programme and to assign the label 0 to its first instruction. Sometimes we may wish to *restart* an Autocode programme that has already been read and stored, perhaps after changing one or two parameters or after making some alterations (this is described below). This can be done by reading a bracketed interlude ending with a jump, for example

(*n*3 = −48
→1)

J 1.2 must be punched at the head of this tape; this has the same effect as the instruction TAPE so that any instructions read will overwrite the last bracketed interlude. J 1.0 must not be punched at the beginning of this tape for the input programme would then treat it as the start of a new programme and store it over the beginning of the original programme.

Occasionally it may be necessary to alter an instruction in an Autocode programme. This can be done by using the tape-editing equipment to alter the tape but this may be inconvenient if the tape is long. Instead of doing this a correction may be inserted near the end of the tape, before the bracketed interlude causing entry to the programme. This correction is written, for example

ALTER 6,3

followed by an Autocode instruction (on a new line); this instruction will then replace the one 3 after that labelled 6. Similarly

ALTER 6

causes the instruction labelled 6 to be altered. This sequence forms a kind of *Autocode directive* and is not stored; it does not form a part of the programme. When the alteration has been made the input section of the Autocode scheme returns to read more instructions or alterations. If the alteration is to be made by a separate tape then it should be headed J 1.2 as described above.

11. Tape Preparation

Instructions are punched exactly as written. During input CR LF, LF, Sp, ϕ and Er are ignored between instructions. Er is ignored everywhere except between CR and LF, Sp is ignored in instructions except among the digits of a number. Every instruction must be terminated by CR LF.

Punching errors or tape reader errors detected during input of instructions will cause a 77-stop in 0.2. The instruction in which the stop occurs can be read in again by pulling the tape back to the beginning of the instruction and operating the STOP/RUN key.

Numbers to be taken in by input instructions must be preceded by their sign and terminated by CR LF or Sp. CR LF, LF, Sp, ϕ and Er are ignored between numbers and Er is ignored in numbers.

Punching errors or tape reader errors during input of numbers will cause a 77 stop in 0.4. The number in which the stop occurs can be read in again by pulling back the tape, clearing the handswitches and operating the STOP/RUN key. The input instruction can be repeated from the beginning by first ensuring that the handswitches are not clear. Most punching errors in numbers cause a stop immediately they are read. It should be noted however that if the sign of a number is not punched the stop does not occur until the terminating character is read.

12. Storage Space and Operation

Ordinarily the complete Autocode tape is read in with handswitch 1 up, and the Autocode programme is read after it by operating the RUN key. Space in the Main Store is then allocated as follows:-

B1 - 111.1	Autocode Scheme
B111.2 onwards	Labels (see below)
B124.0 - 127.3	Indices ($n0$ to $n27$, one location each)
B127.4 - 299.7	Variables ($v0$ to $v1379$, one location each)
B300 onwards	Instructions (one block each)

There is room for 210 instructions if the date and serial number stored in B511.6 and 511.7 are not to be disturbed. Each instruction occupies one block except for the last instruction of a bracketed interlude which takes two blocks.

The number of locations used for labels is one more than the highest number label used. For instance if the highest number label is 24 then the 25 locations B111.2 to 114.2 are used.

During input of the Autocode B126.0 to 128.6 are used temporarily for two interludes and a list of parameters.

If handswitch 1 is down a 77 stop in 2.6 occurs near the end of the tape which gives the programmer an opportunity to change the numbers of indices, variables and instructions available. The following tape puts the index $n0$ in B120.5, $v0$ in B123.3 and the first instruction in B250:

```
T128.0
```

120 0	500	0.
123 0	300	0.
250 0	-00	0.

```
J126.2
A3
Z
```

There is then no need to read the remaining portion of the Autocode tape. The variables must begin in or before B127.7 and there must be room for at least one index (stored before $v0$).

It is possible to use the Assembly facilities with the Autocode. Master programme and subroutines may be read in by going to run after the 77 stop, taking care to set the transfer address to leave room for the labels used. A1 must not appear on this tape. The tape must end with

```
J126.2
A3
Z
```

Alternatively programme, including directive A1, may be read in before R 600. The Autocode tape must be read from after the A1 directive with which it begins. The Autocode will then occupy B0+ - B109+.1 and B1. An interlude ensures that the storage used for labels begins in B109+.2.

13. Error Tracing

Each instruction occupies one block in the Main Store. The number of this block is held in 2_B while the instruction is being obeyed, except sometimes during input and output. The rest of X2 is not used. Hence, if a stop occurs while programme is being obeyed it is quite easy to trace the instruction concerned.

The following loop stops may occur:

0.0 663	SQRT argument negative
0.0+ 760	Floating-point division by zero. Infinite result in SEC, CSC, COT.
0.2+ 463	ARCCOS, ARCSIN argument outside range $-1 \leqslant x \leqslant 1$
0.3+ 660	LOG argument zero.
0.4+ 663	LOG argument negative.
1.4 060	Floating-point overflow.

A write with overflow stop is caused if the result of any index instruction overflows.

The blocks of programme are transferred into U5 before being obeyed. Hence a punch on block transfers while the programme is being obeyed will show the whole course of the programme by giving the block numbers of the instructions obeyed (interspersed with some other lower block numbers).

It may sometimes be useful to put an optional stop in an instruction: to obtain, for example a punch on block transfers after part of the programme has been obeyed.

The following is an example of how a programme tape might end in order to achieve this:

```
(→I0)
 S351.0
 L
(→0)
```

Such an optional stop, which should be in location 0 of the block concerned, will occur in U5.0 after the variables or indices involved have been read and unpacked but before the rest of the instruction has been obeyed. Note that instruction number n is in B300 + n.

If an optional stop is inserted in B44+.5 of R 600 a stop will occur between instructions as the programme is obeyed. The number of the next instruction to be obeyed will be found in 2_B.

14. Accuracy of Functions

The function instructions give a *maximum* error of 1 in the eighth significant decimal place (the maximum error is actually 2^{-29} in an argument between ¼ and ½) with the following exceptions:

LOG with argument near 1. The answer which is near 0 is correct to eight decimal places (not 8 significant figures).

SIN, COS, TAN, CSC, SEC and COT are evaluated by dividing the argument by 2π and disregarding the integral part. The accuracy falls off as the argument increases. Argument 11000 gives six decimal figures correct. These functions give the full number of significant figures for small arguments.

15. Organisation of Functional Subroutines

The subroutines performing the function instructions occupy storage as follows:

SIN, COS, TAN, CSC, SEC, COT	B77+ - 83+
LOG	B84+ - 86+
SQRT (also used by ARCCOS, ARCSIN)	B87+ - 88+.3
FRAC	B88+.4 - 89+
EXP, EXPM,	B90+ - 93+.3
ARCCOS, ARCSIN, ARCTAN	B93+.4 - 101+

A dictionary of functions starts in B104+.6, each entry occupying two words. The first word is an integer calculated as follows: let the function be $\lambda_r\lambda_{r-1}\ldots\lambda_0$. The integer is

$$32^r(31 - \lambda_r) + 32^{r-1}(31 - \lambda_{r-1}) + \ldots + (31 - \lambda_0)$$

where λ denotes the numerical value of the letter. The second word in the entry is the cue to a subroutine to perform the calculation.

Consider, for example, the LOG function. L is the 12th letter of the alphabet, O the 15th, G the 7th

$$\therefore \text{LOG} = 32^2(31 - 12) + 32(31 - 15) + (31 - 7) = 19992$$

The entry in the dictionary for LOG would therefore be

+ 19992	function number
84+ 0 72 0.3 0 60	cue to LOG subroutine

The layout of the Autocode tape is as follows:

```
A1
Programme in B1
Parameters and interludes
in B126.0 - 128.6
L
Programme headed by cue list and name
J127.0  enters an interlude which sets the
        beginning of the list of labels at
        the current Transfer Address
L
J126.0  examines handswitches
J126.2  calculates further parameters from
        those supplied on Autocode tape or
        by programmer.
A3
Z
```

The entries in the dictionary are in the order MOD, TAPE, TAPEB, EXP, EXPM, LOG, SQRT, INT, FRAC, COS, SEC, COT, SIN, CSC, TAN, ARCCOS, ARCSIN and ARCTAN. Alterations made to this list should be made on the Initial Orders tape before the directive J127.0. The word in B128.3 must be altered to contain the address of the beginning of the dictionary in the modifier position and the number of entries in the counter position. The sign digit must be present in this word. The dictionary must begin in an even address.

Any subroutine written to extend the list of functions must satisfy the following conditions: U5.0 – 5.5 and X2 must not be disturbed. Any Computing Store blocks used must be restored, U0 from B41+, U1 from B42+, ..., U4 from B45+. On entry the argument is unpacked with (256 + exponent) in X4 and numerical part in X6 (in the case of an index it is in 6_m on entry). The result should be left unpacked in the same form. Exit is by a jump to U5.1.

The Transfer Address at the J127.0 must be the address at which the list of labels is to begin. J127.0 enters an interlude which sets the beginning of the list of labels at the current Transfer Address.

16. Table of Instructions

Arithmetic Instructions

$v1 = v2$	$v1 = -v2$	$n1 = n2$	$n1 = -n2$
$v1 = v2 + v3$	$v1 = -v2 + v3$	$n1 = n2 + n3$	$n1 = -n2 + n3$
$v1 = v2 - v3$	$v1 = -v2 - v3$	$n1 = n2 - n3$	$n1 = -n2 - n3$
$v1 = v2 \times v3$	$v1 = -v2 \times v3$	$n1 = n2 \times n3$	$n1 = -n2 \times n3$
$v1 = v2/v3$	$v1 = -v2/v3$	$n1 = n2/n3$	$n1 = -n2/n3$

$n1 = n2 * n3$ (remainder of $n2/n3$)
$n1 = -n2 * n3$ (minus remainder of $n2/n3$)
$n1 = v2$ $n1 = -v2$ (nearest integer)
$v1 = n2$ $v1 = -n2$
$v1 = n2/n3$ $v1 = -n2/n3$

Functions

$v1 =$ MOD $v2$	$v1 = -$MOD $v2$
$v1 =$ INT $v2$	$v1 = -$INT $v2$
$v1 =$ FRAC $v2$	$v1 = -$FRAC $v2$
$v1 =$ SQRT $v2$	$v1 = -$SQRT $v2$
$v1 =$ SIN $v2$	$v1 = -$SIN $v2$
$v1 =$ COS $v2$	$v1 = -$COS $v2$
$v1 =$ TAN $v2$	$v1 = -$TAN $v2$
$v1 =$ CSC $v2$	$v1 = -$CSC $v2$
$v1 =$ SEC $v2$	$v1 = -$SEC $v2$
$v1 =$ COT $v2$	$v1 = -$COT $v2$
$v1 =$ ARCSIN $v2$	$v1 = -$ARCSIN $v2$
$v1 =$ ARCCOS $v2$	$v1 = -$ARCCOS $v2$
$v1 =$ ARCTAN $v2$	$v1 = -$ARCTAN $v2$
$v1 =$ LOG $v2$	$v1 = -$LOG $v2$
$v1 =$ EXP $v2$	$v1 = -$EXP $v2$
$v1 =$ EXPM $v2$	$v1 = -$EXPM $v2$
$n1 =$ MOD $n2$	$n1 = -$MOD $n2$

Notes: INT $v2 \leqslant v2$, FRAC $v2 \geqslant 0$

Input Instructions

$v1 =$ TAPE $v1 =$ TAPEB	read one number
$n1 =$ TAPE $n1 =$ TAPEB	read one integer
$v1 =$ TAPE $n2$ $v1 =$ TAPEB $n2$	read $n2$ numbers
$n2 =$ TAPE $n1$ $n2 =$ TAPEB $n1$	read $n1$ integers
$v1 =$ TAPE * $v1 =$ TAPEB *	read numbers up to L
$n1 =$ TAPE * $n1 =$ TAPEB *	read integers up to L

Notes:
TAPE implies main tape reader
TAPEB implies second tape reader
$n0$ = number of numbers read

Jump Instructions

→1 (unconditional jump)

→1, $\pm v2 \geqslant \pm v3$	→1, $\pm v2 > \pm v3$
→1, $\pm v2 = \pm v3$	→1, $\pm v2 \neq \pm v3$
→1, $\pm n2 \geqslant \pm n3$	→1, $\pm n2 > \pm n3$
→1, $\pm n2 = \pm n3$	→1, $\pm n2 \neq \pm n3$

→1, $\pm v2 = * \pm v3$ (jump if approximately equal)
→1, $\pm v2 \neq * \pm v3$ (jump if not approximately equal)

Output Instructions

PRINT $v1$, $n2$ (print $v1$ in style $n2$)
PRINT $n1$, $n2$ (print $n1$ in style $n2$)
XP before an instruction (obey instruction and print result on new line)
SP before an instruction (as XP but print on same line)
X before an instruction (print CR LF after obeying instruction)
S before an instruction (print Sp after obeying instruction)

Note: XP and SP printing is suppressed if H0 = 1

Printing Styles

Style number = $1000a + 20b + c$
For variables b = number of digits before the decimal point
c = number of digits after the decimal point
Indices are always printed as 4 digit integers

	Floating-point (Variables only)	Fixed-point
Print on a new line (CR LF ϕ before number)	$a = 1$	$a = 3$
Print on same line (Sp before number)	$a = 2$	$a = 4$

Miscellaneous Instructions

STOP (77 stop)
TAPE (read more instructions to end of programme)
TAPE $n1$ (read more instructions to $n1$ onwards)
TAPE $n1$, $n2$ (read more instructions to replace instruction $n2$ after $n1$)
→M $n1$ (exit to machine orders at decimal address $n1$)
→M $n1$, $v2$ (as →M $n1$ and leave $v2$ unpacked in X7 and X5)
→M $n1$, $n2$ (as →M $n1$ and leave $n2$ in 7_m)
→L (obey link set in X1 on entry from machine orders)
→I0 (call in Initial Orders as a subroutine)
→I0, $n1$ (as →I0 and set T.A. = $n1$)
→I0, $n1$,$n2$ (as →I0, $n1$ and set Relativiser = $n2$)

Entry to Autocode J1.0
Re-entry J1.2
Entry from machine orders –cue 01 with number of label in X2

Table 7 Autocode Instructions

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 2
13.11.56.

FLOATING-POINT ARITHMETIC

A self-preserving subroutine with four modes of entry to carry out the operations of addition, subtraction, multiplication and division on packed floating-point numbers.

Name: FLOATING POINT ARITHMETIC

Store: 4 blocks

Uses: U3, 4, 5; X1, 4, 5, 6, 7.

Cues: 01 a-order partial cue.
02 b-order partial cue.

See paragraph 1.1 below.

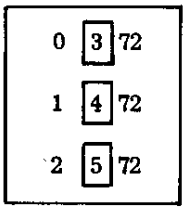
Times:

Addition	18 ms	These times are approximate; they include the two orders needed to enter R 610.
Subtraction	18½ ms	
Multiplication	14 ms	
Division	18 ms	

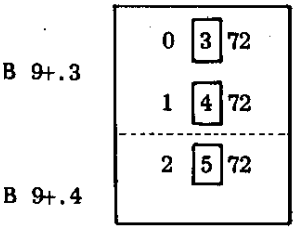
Link: Computing store link set in X1. See paragraph 1.3 below.

1. Method of Use

1.1 Before any use is made of R 610 it must be brought into blocks 3, 4, 5 of the computing store by means of three block-transfer orders:-



These orders must be tagged so that the correct block-address of R 610 is added to each of them; this block-address is given in cues 01 (for an a-order) and 02 (for a b-order) of R 610. For example, suppose that the three block-transfer orders are in block 9+ of the master-programme with the first two of them in 9+.3 and the third as the a-order of 9+.4:-



Then the following three tags calling for cues must be punched at the head of the master-programme:-

R 9 3 -0 1 610 - 01 -	cue 01 (a-order block-address)
R 9 3 -0 2 610 - 01 -	cue 02 (b-order block-address)
R 9 4 -0 1 610 - 01 -	cue 01 (a-order block-address)

When R 610 has been brought in in this way it may be used repeatedly with computing store links.

1.2 If $F(6)$ and $F(7)$ are the floating-point numbers in accumulators 6 and 7 respectively (see section 2 below), the following operations may be performed by R 610:-

Addition	$F(6) + F(7)$
Subtraction	$F(6) - F(7)$
Multiplication	$F(6) \times F(7)$
Division	$F(6) / F(7)$

In each case the result is left in 6, which may thus be regarded as a floating-point accumulator.

1.3 Two orders (which need not form an order-pair) are needed for each entry to R 610. The first order sets the link in X1 and is always as follows:-

(A) 1 40

where A is the computing-store address to which R 610 is to jump on exit.

The form of the second order depends on the operation to be carried out:-

For addition	3.0 1 66	$F(6)' = F(6) + F(7)$
For subtraction	3.0 0 60	$F(6)' = F(6) - F(7)$
For multiplication	3.0+ 0 60	$F(6)' = F(6) \times F(7)$
For division	3.0+ 1 66	$F(6)' = F(6) / F(7)$

2. Floating-Point Numbers

2.1 A floating-point number $x = A.2^a$ is held in a single word with the least-significant n bits representing the non-negative integer $a + 2^{n-1}$, and the most-significant $39 - n$ bits representing the fraction A. If such a word is held in register N its value is written as $F(N)$. The number n is specified by a parameter-list. See section 3 below.

2.2 The *exponent*, a, is restricted to the range

$$-2^{n-1} \leqslant a \leqslant 2^{n-1} - 1.$$

If x is not zero then the *argument*, A, satisfies

$$\tfrac{1}{4} \leqslant A < \tfrac{1}{2} \qquad \text{if } x \text{ is positive,}$$

or $$-\tfrac{1}{2} \leqslant A < -\tfrac{1}{4} \qquad \text{if } x \text{ is negative.}$$

Zero is represented by $A = 0$ and $a = -2^{n-1}$. This corresponds to a null word, i.e. a word all of whose digits are zero.

2.3 The result of the operation carried out by R 610 is left in X6 in standard packed form as described in the previous paragraph. Should the exponent become too small (i.e. less than -2^{n-1}; this is called *underflow*) or the argument become zero then the result is automatically set equal to zero. If the exponent becomes too large (i.e. greater than $2^{n-1} - 1$) overflow is said to occur and the overflow-indicator (OVR) will be set. In these cases the result will always be zero unless the overflow is caused by dividing by zero.

3. Preset-parameters

3.1 The value of n, i.e. the number of binary digits used to represent the exponent part of the floating-point numbers, is specified by three preset-parameters. The number n may be any positive integer up to 35. If no parameter-list is supplied then n will be set equal to 9 by an optional parameter-list.

3.2 If $n = 9$ then the exponent may lie in the range

$$-256 \leqslant a \leqslant 255,$$

which permits the use of numbers whose absolute values lie in the range

$$2.10^{-78} < |x| < 2.10^{76} \qquad \text{(approximately)}$$

The argument A will be represented with a precision of just under 9 significant decimal digits.

3.3 If it is desired to use a value of n other than 9, then a parameter-list of the following form must be read in at some stage:-

	R 0 0 -0 3 610 - 04 -	Title
01	-2^{n-1}	$= -2^{n-1}.2^{-38}$
02	37–n 0 00 0	$= (37 - n).2^{-7}$
03	0 37–n 0 00	$= (37 - n).2^{-26}$

This parameter-list will normally form part of the master programme.

Ferranti Ltd.
London Computer Centre,
21, Portland Place,
London.W.1.

Issue 2
13th November, 1956
G.E.F.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 2.
15.9.56.

FLOATING-POINT SQUARE ROOT

$$F'(6) = \sqrt{F(6)}$$

where $F(6)$ is the floating-point number held in X6.

Name: F.P. SQ. ROOT

Store: 2 blocks, plus 1 location for optional parameter-list.

Uses: U0, 1; X1, 5, 6, 7

Cue: 01 (0+.0)

Time: About 44 m.s. It varies according to the magnitude of the number as shown below

Argument	Exponent Even	Exponent Odd
0	6 m.s.	*
0.25	44 m.s.	44 m.s.
0.375	36 m.s.	
0.499	15 m.s.	

* If the argument is zero the exponent should also be zero and therefore even.

Link: Obeyed in 1.5

Error: The error in the square root will be less than one in the last binary place of the argument.

Notes: (1) A loop stop will occur in 0.1 if $F(6)$ is negative.

(2) Unlike R 610, (Floating Point Arithmetic), R 611 is not self-preserving.

FLOATING-POINT NUMBERS

A floating-point number $x = A.2^a$ is held in a single word. The least-significant n bits represent the non-negative integer $a + 2^{n-1}$, and the most significant $39-n$ bits represent the fraction A. If such a word is held in register N its value is written as $F(N)$.

The value of n is specified by a parameter list.

The *exponent* , a, is restricted to the range

$$-2^{n-1} \leqslant a \leqslant 2^{n-1} - 1$$

If the number x is not zero, the *argument* , A, satisfies

$$\tfrac{1}{4} \leqslant A < \tfrac{1}{2} \quad \text{if } x \text{ is positive,}$$

$$\text{or } -\tfrac{1}{2} \leqslant A < -\tfrac{1}{4} \quad \text{if } x \text{ is negative.}$$

Zero is represented by $A = 0$ and $a = -2^{n-1}$, so that all the digits of the word are zero.

If the number $F(6)$ is in standard floating-point form the answer $\sqrt{F(6)}$ will also be in the standard form described above. There is no possibility of overflow.

PRESET-PARAMETER

n is the number of binary digits used to represent the exponent part of the floating-point numbers; it may be any positive integer such that $2 \leqslant n \leqslant 35$. n is set by a preset-parameter; if no parameter-list is provided n will be set equal to 9.

If $n \neq 9$ then the exponent may lie in the range

$$-256 \leqslant a \leqslant 255$$

which permits the use of numbers whose absolute value lies in the range

$$2.10^{-78} < |x| < 2.10^{76} \quad \text{(approximately)}$$

The argument A will be represented with a precision of just under 9 decimal digits.

If it is desired to use a value of n other than 9 a parameter-list of the following form must be read in at some stage:-

	R 0 0 -0 1 611 - 04 -	} Title
01	-2^{n-1}	

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 2
15th September, 1956
D.M.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
10.12.57.

SHORTER FLOATING-POINT ARITHMETIC

A self-preserving subroutine with three modes of entry to carry out the operations of addition, subtraction and multiplication on packed floating-point numbers. There is a facility for adapting the routine to do division instead of multiplication.

It occupies only 2 blocks in the Computing Store and is faster than R 610 for addition, subtraction and multiplication. It will not detect floating-point overflow, and will occasionally produce slightly non-standard numbers.

Name: SHORTER F.P. ARITHMETIC

Store: 3 blocks

Uses: U4, 5; X1, 4, 5, 6, 7.

Cues: 01 to bring the routine into the Computing Store.
02 to change the rountine ready for division.
03 a-order partial cue.
04 b-order partial cue.

Times:

Addition	14½ ms.	These times are approximate; they include the two orders needed to enter R 612.
Subtraction	14½ ms.	
Multiplication	11½ ms.	
Division	29 ms.	

Link: Computing Store link set in X1. See paragraph 1.2 below.

1. Method of Use

1.1 The normal way to set the routine for multiplication, addition or subtraction is by bringing the first two blocks into blocks 4 and 5 of the Computing Store. This may be done by obeying cue 01, which consists of the following order pair:

```
0+ [4] 72
1+ [5] 72
```

1.2 Two orders (which need not form an order pair) are needed for each entry to R 612. The first order sets the link in X1 and is always as follows:-

```
(A) 1 40
```

where A is the Computing Store address to which R 612 is to jump on exit.

The form of the second order depends on the operation to be carried out:-

For addition	4.0 1 66	$F(6)' = F(6) + F(7)$
For subtraction	4.0 0 60	$F(6)' = F(6) - F(7)$
For multiplication (or division)	4.0+ 0 60	$F(6)' = F(6) \times F(7)$ $[F(6)' = F(6) / F(7)]$

1.3 Once the routine is in the Computing Store, it may be set for division by obeying cue 02:-

2+ 5 70
4.4 1 10

After obeying this cue the routine may be used for addition, subtraction or division, but not multiplication. The entries for addition and subtraction are unaltered; the third entry is now used for division instead of multiplication.

1.4 After the routine has been set for division it may be reset for multiplication by obeying cue 01.

Alternatively the following entry may be used for the last division:

For division (resetting for multiplication)	A 1 40 4.0+ 1 66	$F(6)' = F(6) / F(7)$

After the division has been performed the routine will be reset for multiplication.

If this entry is used when the routine is already set for multiplication, it will have the same effect as the normal multiplication entry.

2. Floating-Point Numbers

2.1 A floating-point number $x = A.2^a$ is held in a single word with the least-significant n bits representing the non-negative integer $a + 2^{n-1}$, and the most-significant $39-n$ bits representing the fraction A. If such a word is held in register N its value is written as $F(N)$. The number n is specified by a parameter-list. See section 6 below.

2.2 The *exponent*, a, is restricted to the range

$$-2^{n-1} \leqslant a \leqslant 2^{n-1} - 1.$$

If x is not zero then the *argument*, A, satisfies

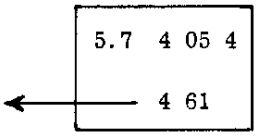
$$\tfrac{1}{4} \leqslant A < \tfrac{1}{2} \text{ if } x \text{ is positive,}$$
$$\text{or } -\tfrac{1}{2} \leqslant A < -\tfrac{1}{4} \text{ if } x \text{ is negative.}$$

Zero is represented by $A = 0$ and $a = -2^{n-1}$. This corresponds to a null word, i.e. a word all of whose digits are zero.

2.3 The result of an operation carried out by R 612 is left in X6 in standard packed form, except as described in section 4. Should the exponent become too small (i.e. less than -2^{n-1}) *underflow* is said to occur, and the answer will be set equal to zero. If the exponent becomes too large (i.e. greater than $2^{n-1} - 1$) *overflow* is said to occur, and the subroutine will give incorrect results.

3. Floating-Point Overflow

A check can be made on floating-point overflow (if it is likely to occur) by obeying the following orders on exit from the subroutine:



This causes a jump if the exponent exceeds $2^{n-1} - 1$.

4. Non-Standard Numbers

Due to the method of rounding off, the subroutine will occasionally produce answers in which the argument is exactly equal to +½ or to -¼. Such numbers are acceptable as operands for R 612, except that if two arguments of +½ are added, there will be OVR: this unlikely event will be indicated by a 77 stop in 5.6 (see section 8).

These rare non-standard numbers are also acceptable to most other floating-point subroutines. If they do cause a fault, it will be indicated by the OVR. An exception to this rule is R 570, which will give incorrect results if entered with non-standard numbers. R 2903 will print an asterisk for such numbers.

5. Exact Cancellation

If two equal numbers are subtracted the answer will not normally be zero, but it will be less than $1/10^{11}$ of the operands. It will have a positive normalised argument and a binary exponent which will be at least 37 less than that of the operands.

6. Preset-Parameters

6.1 The value of n, i.e. the number of binary digits used to represent the exponent part of the floating-point numbers, is specified by a preset-parameter. The number n may be any positive integer up to 10 (if n exceeds 10 the routine may not work correctly for widely differing exponents). If no parameter-list is supplied, n will be set equal to 9 by an optional parameter-list.

6.2 If $n = 9$ then the exponent may lie in the range

$$-256 \leqslant a \leqslant 255,$$

which permits the use of numbers whose absolute values lie in the range

$$2.10^{-78} < |x| < 2.10^{76} \text{ (approximately).}$$

The argument A will be represented with a precision of just under 9 significant decimal digits.

6.3 If it is desired to use a value of n other than 9, a parameter-list of the following form must be read in at some stage:-

	R 0 0- 0 1 612 -0 4 -	Title
01	-2^{n-1}	$= -2^{n-1} . 2^{-38}$

This parameter-list will normally form part of the master programme.

7. A Rapid Link

The use of a Computing Store link involves a shift of 25 places, taking over 3 milliseconds at each entry. If R 612 is used repeatedly in a loop of programme, much time may be saved by planting a link in U 5.5 before entry.

When using this method, the master programme must clear X1 before each entry to the subroutine. The link should include the order

A 0 64

which will return to the master programme only if the OVR is clear. U 5.5 must be restored to

(25) 1 52 0 0 64 1

before R 612 is used with a Computing Store link.

8. Error Stops

4.0	4.0 7 60 5.6 5 01	Loop stop if dividing by zero.
5.6	127 7 77 7 127 7 40	77 stop if OVR is set before or during the routine. This may happen when working with non-standard numbers.
5.7	127 7 77 7 127 7 00	77 stop if RUN key is operated after 77 stop in 5.6.

Note: The contents of 5.6 and 5.7 shown above are for $n = 9$. The actual values are always -2^{n-1} and -2^{n} respectively.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
London, W.1.

Issue 1
10th December, 1957.
J.F.D. D.M. G.E.F.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

COMPLEX FLOATING POINT ARITHMETIC

An interpretive scheme to facilitate the programming of operations on complex floating point numbers. Special facilities for matrix arithmetic are provided.

Name: COMPLEX ARITHMETIC

Store: 40 blocks plus a 4 block interlude. The interlude, which can be overwritten when input is complete, can be beyond block 127 of the Main Store.

Uses: U1, 2, 3, 4, 5; X1, 6, 7; B0; working space in the Main Store (see section 1); certain blocks down from B510 (B894 on the 7168 store) if subroutines are used (see section 7.4).

Cues: 01, 02, 03, 04, 05, (see sections 3, 7.3, 9).

Time: See section 2.3.

Exit: See end of section 2.3 and section 3.

CONTENTS

1. NUMBERS AND STORE

Numbers are stored in the Main Store working space. They are in floating point form and occupy three locations; the first two locations contain the real and imaginary arguments, while the third holds a common binary exponent as an integer. The larger of the two arguments is in the normalised form; either or both of the arguments may be zero. The values $-\frac{1}{4}$ and $+\frac{1}{2}$ are also acceptable and may sometimes be produced by functions 42 and 46. The address of a number refers to the first word, the other two being at the same position in the following two blocks; this address is relative to the start of the working space which is preset by a parameter (see section 5).

The addresses of numbers must differ by multiples of three; preferably all addresses should be divisible by three, starting at zero, as shown below:

Address:	Contents:
0 + .0	α_0 (real part)
.1	
.2	
.3	α_1 (real part)
.4	
.5	
.6	α_2 (real part)
.7	
1 + .0	α_0 (imaginary part)
.1	α_3 (real part)
.2	
.3	α_1 (imaginary part)
.4	α_4 (real part)
.5	
.6	α_2 (imaginary part)
.7	α_5 (real part)
2 + .0	ϵ_0
.1	α_3 (imaginary part)
.2	α_6 (real part)
.3	ϵ_1
.4	α_4 (imaginary part)
.5	α_7 (real part)
.6	ϵ_2
.7	α_5 (imaginary part)

α = argument ϵ = exponent

Most operations are carried out on these numbers and the contents of the floating point accumulator. The accumulator occupies U1.1, 1.3 and 1.5 and holds numbers in the same form as the Main Store.

2. ORDERS

2.1 General Description

The basis of the scheme is the interpreted order. Interpreted orders are written and punched in a form similar to ordinary Pegasus orders and are read in to the Main Store by Initial Orders. When obeyed they are in U0. The interpreting routine examines them and performs the specified operation. In U0 interpreted orders are obeyed consecutively except when a jump takes place. The next order obeyed after an interpreted order in 0.7+ is in 0.0, whether it is to be interpreted or not.

2.2 Structure and Modification

As with ordinary orders there are 19 binary digits to the order and thus two orders to the word. The stop-go digit is non-functional for interpreted orders. The orders will be described by the same letters (N, X, F, M,) as ordinary orders except when the first ten digits are thought of together. In this case the ten digits as an integer are designated S. S is usually the address (relative to the start of the working space) of a number, while s is the number stored at this location. The floating point accumulator will be denoted by A and its contents by a. a always represents a floating point number as described in section 1. Thus, for example:

$$\overbrace{N \quad X}^{S} \quad F \quad M$$
$$12 \quad - \quad 40$$

means transfer the contents of $S12$ to A, or $a' = s_{12}$. The modifier can only be X3, X4 or X5 and these are the accumulators which can be used with the counting functions. In all modification the thirteen digits of the modifier position of the accumulator, M, are added to the first ten digits of the order (i.e. S or N and X) which are extended by three digits. Jump orders always jump to an address in U0 however modified. Modification cannot cause overflow.

2.3 Order Code and Order Times

F	Operation	mean times in milliseconds
00	$a' = a + s$	about 60
04	$a' = -a - s$	"
10	$a' = a - s$	about 60
14	$a' = s - a$	"
20	$C'(X)$ = Nth constant in list (see section 8)	32
22	Read a number to A and S	-
24	Punch CRLF followed by number from A	-
30	Test indicator: Jump IN to N if clear. Clear indicator.	24
32	Test indicator: Jump OUT to N if clear. Clear indicator.	26
34	Count: $x'_m = x_m + 3$, $x'_c = x_c - 1$; jump IN to N if $x'_c \neq 0$	26
36	Count: $x'_m = x_m + 3$, $x'_c = x_c - 1$; jump OUT to N if $x'_c \neq 0$	27
40	$a' = s$	38
42	$a' = -s$	38
44	$s' = a$	37
46	$s' = -a$	37
50	Block read. U0′ = N(+); jump to 0.X	34
52	$a' = S$	25
54	Set indicator if $\epsilon_a < \epsilon_s$	30
56	Set indicator if max. $(\lvert\alpha_r\rvert, \lvert\alpha_i\rvert) \leq 2^{-S}$	24
60	$a' = a \times s$	50
62	Interchange: $a' = s$ and $s' = a$	52
70	$a' = \frac{a}{s}$	70
76	Subroutine operation S (see section 7)	-

Notes:

F 30, 32, 54, 56; the indicator is the sign of U5.7; when set it is negative. The rest of U5.7 is used for other purposes.

F 34, 36; counting does not carry from x_c to x_m. No jump occurs if x_m overflows, and a jump will always occur if accumulator 0 is specified.

F 30, 32, 34, 36; jump 'IN' means jump to an interpreted order, jump 'OUT' means jump to an ordinary order. The block part of the jump address is ignored: all these functions will only jump to orders in U0.

F 30, 32; the unused X digits in the order may take any value.

F 70; loop stop in U5.0+ on division by zero (see section 10).

F 20, 34, 36; X = 3, 4 or 5 only.

Unassigned orders; extravagant effects may normally be expected on obeying these. However, some programmes written for R 650 may be suitable for R 630, and the following information may help with conversion:

Functions 02, 06, 12, 16, 26 have identical effects to functions 00, 04, 10, 14, 24,

Function 64 becomes a null order.

The functions have been described with the second octal F-digit even. If the second function digit is made odd by the addition of 1, the next order is not interpreted but is obeyed in the normal way as a machine order. This applies to the order jumped to by F50 and to the order returned to after using a subroutine with F76. A sequence of machine orders may also be entered by a jump using F32 or 36.

Machine orders may use U0 and 4 and all the accumulators without disturbing the interpretive routine.

3. ENTRY AND EXIT

The routine has five cues. Cues 01 and 02 bring the interpreting routine into the Computing Store and jump to an order in U0. The address of the order must be specified in $X6_c$ before obeying the cue, by an instruction such as:

```
0.P(+)  6 40
```

Cue 01 is used to jump to an ordinary order and cue 02 to jump to an interpreted order.

If the interpreting routine is in the Computing Store a sequence of interpreted orders is entered by setting the address of the first order in $X6_c$ and jumping to 1.7+ e.g. by obeying orders

```
0.P(+)  6 40
 1.7+   0 60
```

If the order to be obeyed is in U0.0 all that is necessary is a jump to 1.7.

On exit from interpreted orders X2, 3, 4 and 5 will be unchanged since entry except for the result of counting or using F20 in any of the latter three.

The total extra time for entry and exit is 4 milliseconds.

4. NUMBER INPUT AND OUTPUT

4.1 Input

Function 22 reads a real or complex number from tape in fixed or floating point form. The number is then put in the accumulator and in working space location S.

All arguments, fixed or floating, must contain a decimal point, and all numbers, whether argument or exponent, must be preceded by a sign. Not more than 11 decimal digits can be accepted before a decimal point.

The routine will accept from one to four numbers separated by any number of spaces and terminated by CR LF. The way in which these numbers are interpreted is given by the table below:

Read	Interpreted as:
$\pm a.b$	$\pm a.b$
$\pm a.b \pm c$	$\pm a.b \times 10^{\pm c}$
$\pm a.b \pm c.d$	$\pm a.b + \mathrm{i}(\pm c.d)$
$\pm a.b \pm c \pm d.e$	$\pm a.b \times 10^{\pm c} + \mathrm{i}(\pm d.e)$
$\pm a.b \pm c.d \pm e$	$\pm a.b + \mathrm{i}(\pm c.d \times 10^{\pm e})$
$\pm a.b \pm c \pm d.e \pm f$	$\pm a.b \times 10^{\pm c} + \mathrm{i}(\pm d.e \times 10^{\pm f})$

No other combinations are allowed.

CR LF, LF and ϕ can precede the sign of the first number. One or more spaces must occur to separate the numbers; spaces may also be punched before the first number or after the last, before the CR LF. The decimal point may occur anywhere in an argument: it may precede or terminate it. No exponent may contain a decimal point. Erase may be punched anywhere except between CR and LF.

All other characters and arrangements cause a stop. (See section 10.)

Arguments may be punched with any number of digits after the decimal point: digits beyond the eleventh are ignored. If the sum of an exponent and the number of digits accepted after the decimal point of the argument is outside the range $-256 \leqslant x < 256$ input reaches a 77-stop in 3.0 unless the argument is zero. It will, however, continue on operating the STOP/RUN key.

BO is used during input.

4.2 Output

Function 24 punches out the number in the accumulator preceded by CR LF.

For this function, the ten binary digits of S specify the style of output. The first five, S_1, indicate the number of digits required before the decimal point (unless $S_1 = 0$), while the last five indicate the number required after the point. A simple way of writing S in the order is as the decimal number $32S_1 + S_2$.

If $S_1 = 0$ the output is in the standard form consisting of a signed argument n, where $1 \leqslant |n| < 10$, followed by a decimal exponent. The exponent is preceded by one space (and a sign) and is not followed by any spaces. If the argument is zero the exponent is punched as zero.

The imaginary part follows the real part on the same line, preceded by three spaces.

Output is rounded and signs are always printed. Left hand non-significant zeros are replaced by spaces except for the last in a zero integral part.

Unless a number is zero, there is a 77-stop in 4.2 (due to an order in 4.1+) if the binary exponent of its normalised form is outside the range $-512 \leqslant x < 512$. On continuing from this stop an asterisk is punched followed by zero.

If a number is greater than 10^{11}, the standard form is used. This is done to avoid non-significant figures in the integral part.

Not more than 12 significant digits are punched. Spaces are then punched if necessary to complete the required number of characters after the decimal point.

Output is suitable for re-input by F22 provided the programmer ensures that each complex number is followed by CR LF.

BO is used during output.

5. PRESET PARAMETER

R 630 requires one preset parameter. This must contain, in the a-order, the address at which the Main Store working space is required to start, and zero in the b-order, unless subroutines are being used during the control of F76, as described in section 7. Thus to set the beginning of the working space at B120.0 the parameter list required is

R 0 0 -0 1	
630 - 04 -	
120 0 00 0.	
0	

Normally it is punched with the master programme, but in any case it must be read before R 630.

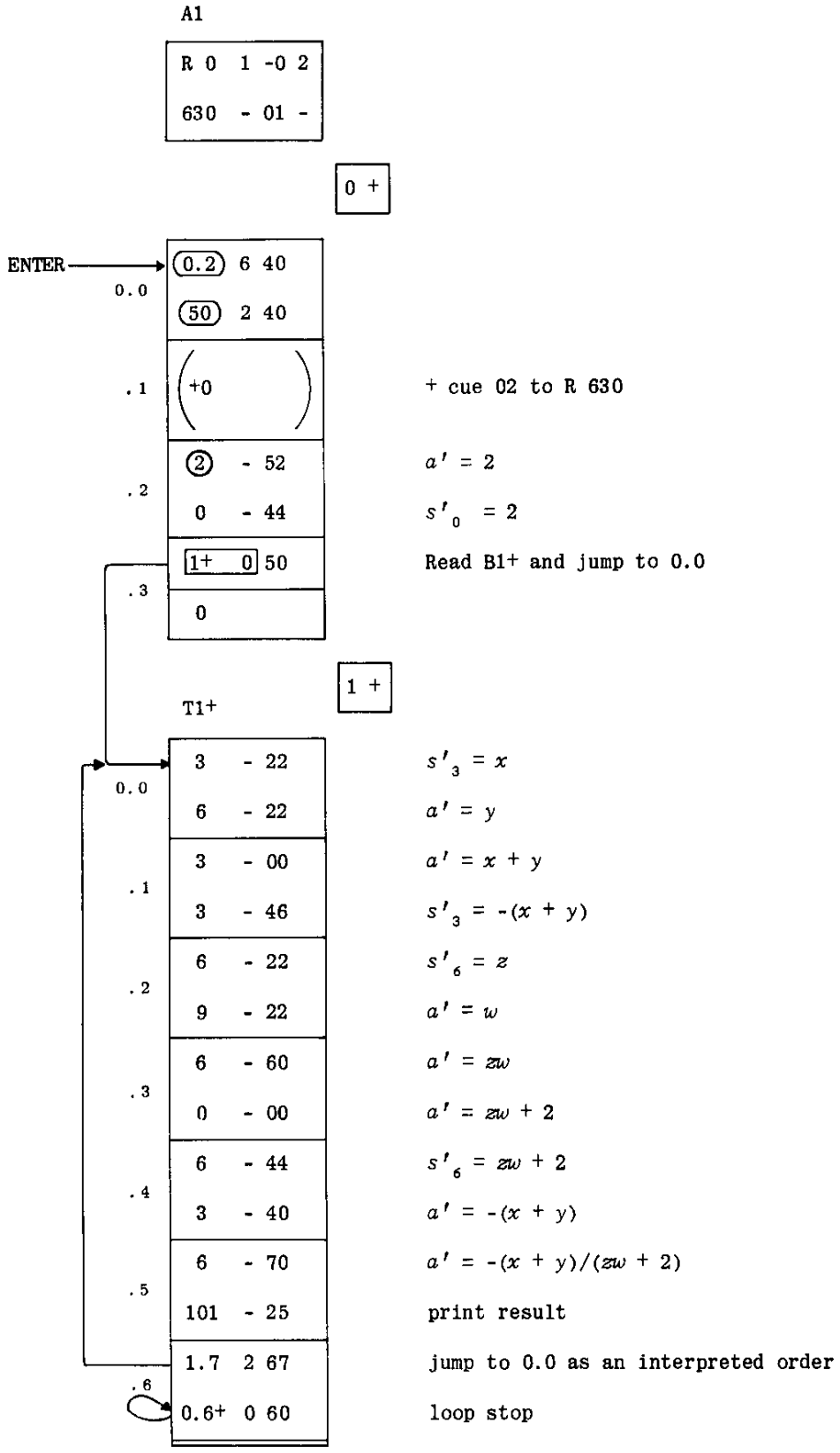
There is an optional parameter which is used if none is supplied by the programmer:

125 7 00 0.	
0	

This address of B125.7 is the largest that can be set for the working space.

6. EXAMPLE

Given fifty sets of numbers x, y, z, w, calculate the numbers $-(x + y) / (zw + 2)$ and print the answers with three digits before the point and five after.



This tape is followed by one containing the fifty sets of numbers x, y, z, w.

7. COMPLEX FLOATING POINT SUBROUTINES

7.1 The use of F76

The purpose of function 76 is to simplify the use of subroutines that work in conjunction with R 630.

If F76 is used the programmer must supply an R 630 working space parameter with the number of different subroutine operations set in the counter position. This parameter must be followed by an index of cue specifications for the required operations. Each of these must hold the subroutine number and the cue number as integers in the a-order and the b-order respectively. For example:-

R 0 0 -0 1 630 - 04 -	
120 0 00 0. 2	working space address B120.0 2 subroutine operations
690 1	R 690 cue 01
690 2	R 690 cue 02

The F76-order carries out the operation appropriate to the S-th specification in the index. For example with the index as above the order

2 - 76

causes R 690 to be entered by cue 02.

When using subroutines in this way no link has to be set. After the operation specified by the 76-order has been completed control is returned to the next order. As with other orders, the addition of 1 to the second function digit causes the next order to be treated as a machine order.

7.2 R 630 Subroutines

The following R 630 subroutines are on the same tape as R 630.

R 690: Linear combination

Store: 5 blocks

Cues: 01 $\underline{u}' = a\underline{v}$
02 $\underline{u}' = \underline{u} + a\underline{v}$

Time: cue 01, $32 + 43n$ milliseconds
cue 02, $32 + 63n$ milliseconds

Before entry set $x_4 = (u, \ )$
$x_5 = (v, n)$

The vector $\underline{u}$ is replaced by or has added to it a multiple of the vector $\underline{v}$. The vectors have n elements and the first elements have addresses u and v respectively.

If the elements of the vectors occupy locations that have a constant address difference other than 3, this difference should be set in $U5.4_m$. On exit 5.4_m is reset to 3 (see section 9).

R 691: Scalar product

Store: 3 blocks

Cues: 01 $a' = \underline{u}.\underline{v}$
02 $a' = a + \underline{u}.\underline{v}$

Time: cue 01, $147 + 91n$ milliseconds
cue 02, $163 + 91n$ milliseconds

Before entry set $x_4 = (u, \ )$
$x_5 = (v, n)$

The scalar product is formed of the vectors $\underline{u}$ and $\underline{v}$ each with n elements, whose first elements have addresses u and v respectively.

If the addresses of consecutive elements of $\underline{u}$ do not differ by 3, the address difference should be set in $U5.4_m$. 5.4_m is reset to 3 on exit. Addresses of consecutive elements of $\underline{v}$ must differ by 3.

R 693: Matrix Division

Store: 8 blocks + 5 blocks for R 690

Cue: 01 $C' = B^{-1}.\ C$

Time: $20p^3 + 340p^2 + 100p + 60p^2q$ milliseconds

p	4	10	20	30
$q = 1$	8 secs.	1 min.	5 min.22 secs.	15 min.
$q = p$	11 secs.	2 min.	13 min.	41 min.

Before entry set $x_4 = (c, q)$
$x_5 = (b, p)$

The matrices B and C, which are stored by rows, have first elements at b and c. B has dimensions $p \times p$ and C has dimensions $p \times q$.

Unlike R 686, the matrix division subroutine normally used with R 650, this routine uses the method of pivotal condensation, resulting in greater accuracy.

The subroutine uses R 690. a and B are destroyed.

R 694: Square root

Store: 4 blocks

Cue: 01 $a' = \sqrt{a}$

Mean time 270 milliseconds

R 695: Matrix Multiplication

Store: 2 blocks + 3 blocks for R 691

Cues: 01 $D' = B.C$
02 $D' = B.C + D$

Time: about $95pr\ (q + 2)$ milliseconds

Before entry set $x_3 = (d, p)$
$x_4 = (c, r)$
$x_5 = (b, q)$

The matrices B, C and D, which are stored by rows, have first elements at b, c and d and dimensions $p \times q$, $q \times r$, $p \times r$ respectively.

R 695 uses R 691 as a subroutine. D cannot be the same matrix as either B or C. a is destroyed.

Note: Function 76 ensures that on exit from these subroutines the Computing Store is unchanged apart from X1, 6, 7 and the accumulator A.

7.3 Programmer's Subroutines

Besides library subroutines, programmer's subroutines can be used with F76.

The subroutines may use interpreted orders and so operate on U0 (and perhaps U4) leaving the interpreting programme undisturbed, or otherwise use any of the rest of the Computing Store, but if U1.1, 1.3 or 1.5 is used the accumulator will be destroyed.

A subroutine should be written in the same form as an ordinary subroutine using assembly. The cues must be machine orders, for example

0+	0 72
1.7	0 60

to enter the subroutine at 0+.0 as an interpreted order. (See section 3).

During the process of entering a subroutine X1, 6, 7 are used but the rest of the Computing Store remains unaltered.

Return from a subroutine is effected by obeying cue 05 of R 630 which is called for in the normal way. The Computing Store, apart from X1, 6, 7 and the accumulator A, will be restored to its condition on entry.

The overflow indicator must be left clear on exit.

7.4 Notes on R 630 Subroutines

1. On proceeding to a 'lower level' with function 76, two blocks backwards from B510 (B894 on the 7168 drum) are used temporarily to store information.

2. The binary translation of R 690, 691, 693, 694 and 695 contains no names of the subroutines.

3. The list of cue specifications must not be written over as it is used each time F 76 is obeyed.

8. F 20

When using F20 the programmer must supply a list of constants following the working-space address parameter and the subroutine index if any.

F20 puts the Nth word in the list into accumulator X.

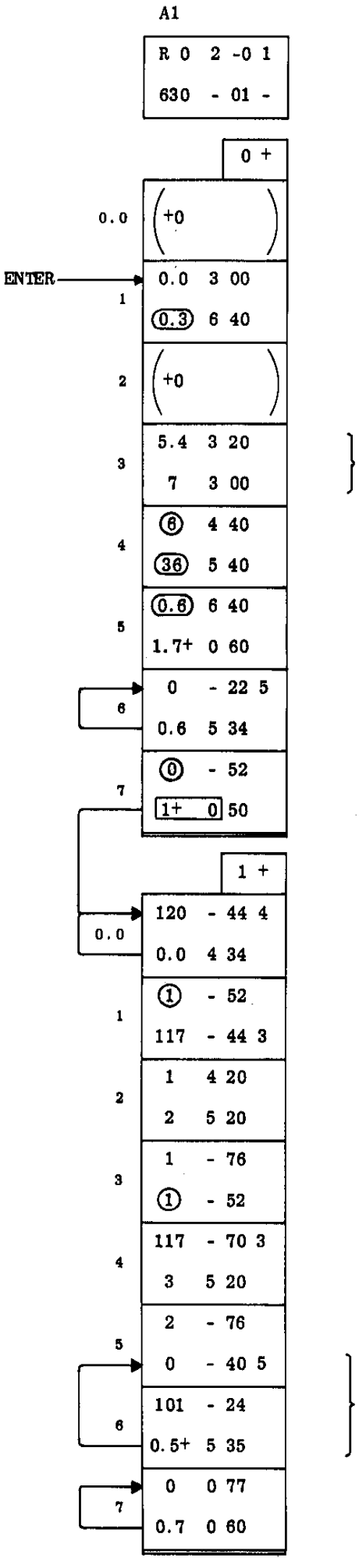
(N = 1, 2, 3..... etc.)
(X = 3, 4 or 5)

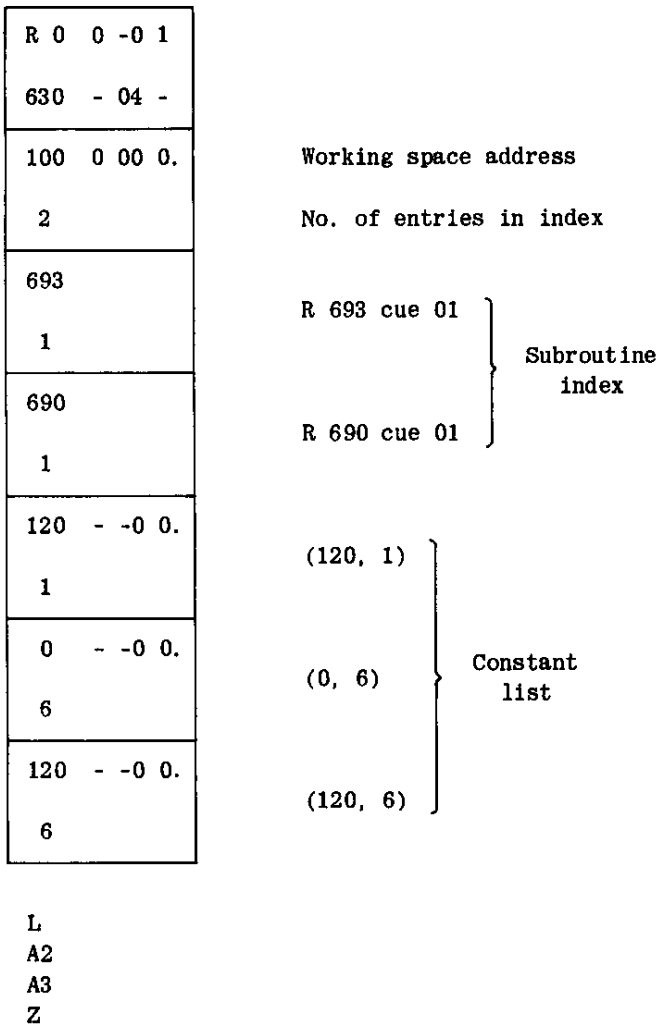
Example

To determine the resulting potentials in a network produced by unit voltage applied at one point.

Given a square matrix Y and a number k, calculate the vector $\underline{e} = Y^{-1}\underline{i}$, where the kth element of $\underline{i}$ is unity and the others are all zero, and print the answer $\underline{e}^*$ scaled so that the kth element is unity.

The matrix is 6 × 6, and the results are required with three digits before the point and five after.


A1
R 0 2 -0 1
630 - 01 -
0 +
0.0
(+0)
ENTER
0.0 3 00
0.3 6 40
2
(+0)
+ cue 01 to R 630
3
5.4 3 20
7 3 00
$3_m = 3k$
4
6 4 40
36 5 40
$x'_4 = (0, 6)$
$x'_5 = (0, 36)$
5
0.6 6 40
1.7+ 0 60
6
0 - 22 5
0.6 5 34
Input Y
7
0 - 52
1+ 0 50
Acc. = 0
Read B 1+ and jump to 0.0
1 +
0.0
120 - 44 4
0.0 4 34
Clear i
1
1 - 52
117 - 44 3
Acc. = 1
Set i
2
1 4 20
2 5 20
$x'_4 = (120, 1)$
$x'_5 = (0, 6)$
3
1 - 76
1 - 52
$e = Y^{-1} i$
4
117 - 70 3
3 5 20
Acc. = $1/e_k$
$x'_5 = (120, 6)$
5
2 - 76
0 - 40 5
$e^* = (1/e_k)\ e$
6
101 - 24
0.5+ 5 35
Print answer (3 digits before the point, 5 after)
7
0 0 77
0.7 0 60
Stop



L
A2
A3
Z

The data tape begins

T 2.0
+ *k*
J 2.1

followed by the matrix Y, punched by rows.

9. GENERAL NOTES

1. Cues 03 and 04 contain the address of the start of R 630 in the *N* and *X* portion of the a-order and the b-order respectively.

2. U5.4 normally contains 3×2^{-13}. This constant is added to the specified accumulator during functions 34 and 36 in order to step on the modifier. It can be set from programme to any desired value. For example it can be set negatively for working backwards through the store or perhaps to some multiple of 3×2^{-13} for referring to a column or the diagonal of a matrix.

However, it is also used by the subroutines that perform matrix and vector operations and must normally contain 3×2^{-13} before a 76-order concerning these is obeyed. Exceptions to this are described in section 7.2. U5.4 is reset to 3×2^{-13} after all 76-orders.

3. Unless a jump occurs, the next order obeyed after an interpreted order in 0.7+ is in 0.0, whether it is to be interpreted or not.

4. The overflow indicator is cleared by the interpreting programme and is always clear on exit.

5. Zero set by F 52 or input by F 22 has an exponent of -2^{19}. Zero resulting from the subtraction of equal numbers has an exponent 37 less than the exponent of the numbers. The exponent of zero resulting from multiplication is 37 less than the sum of the exponents of the operands, while for division it is 36 less than their difference.

6. Exponent overflow is indicated by a loop stop in 5.0+.

7. An optional stop may be inserted in B23+.2 to facilitate the development of programmes. The computer will reach this stop before obeying each interpreted order: at this point 7_m will contain the order number with the a/b digit reversed and repeated up to the sign bit; X6 will contain the order pair being obeyed.

8. If the machine stops when an order is being obeyed the address of the next order will be found in 1.6_m (again the a/b digit is repeated).

9. If the programmer wants the address of the Main Store working-space, he should call for R 630 parameters 01 or 02 which have the address in the N and X portions of the a- and b-order respectively. (see 10 c).

10. The main programme of R 630 is followed by 4 blocks of interlude. These can all be beyond B 127. While R 630 and its subroutines are being read, B 0.0 and B 0.2 are used.

The interlude ensures that R 630 subroutines are accepted. Also, if a non-optional parameter has been supplied:-

(a) Certain words are inserted into the main subroutine concerned with F 20 and F 76.

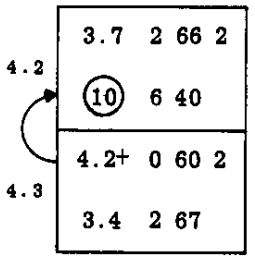
(b) If there is a subroutine index it is processed so that later Assembly inserts the specified cues into it.

(c) The optional parameter is replaced by two similar words containing the programmer's working-space address, and the programmer's parameter is overwritten by information concerned with subroutine entries. Thus the true parameter list is always in the location of the optional list, while the location of the programmer's non-optional parameter list becomes part of the list of subroutine entries and therefore is not available to the programmer.

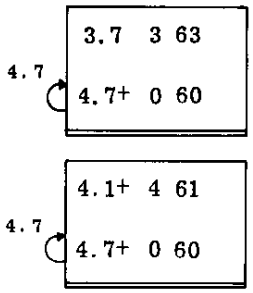
10. STOPS

10.1 During Input of Numbers

(a) Loop stop in 4.2+ and 4.3 if no sign before number.



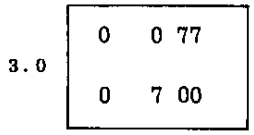
(b) Loop stop in 4.7+ for other punching errors.



(c) 77-stop in 3.0 if either of the following conditions is not satisfied:

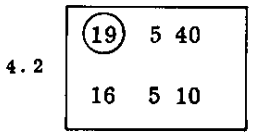
(i) Decimal exponent + number of fractional digits accepted is inside the range $-256 \leqslant x < 256$.

(ii) Number is zero.



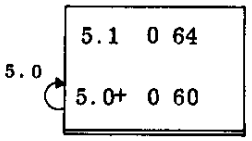
10.2 During Output

77-stop in 4.1+ if exponent is outside the range $-512 \leqslant x < 512$.



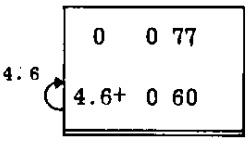
10.3 Arithmetic Orders

Loop stop in 5.0+ if overflow has occurred.



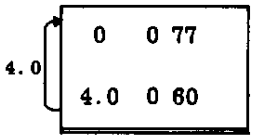
10.4 F 20

77-stop in 4.6 if no parameter is supplied by the programmer. If the RUN key is operated a loop stop will occur in 4.6+.



10.5 F 76

77-stop in 4.0 if no parameter is supplied by the programmer. This stop will be repeated if the RUN key is operated.



Author: Mr. J.G.F. Francis of the National Research Development Corporation.

Ferranti Ltd.,
London Computer Centre,
68, Newman Street,
London, W.1.

Issue 1
23rd May, 1961
J.G.F.F. M.My. T.B.W.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 2
10.7.57.

DOUBLE-LENGTH FLOATING POINT ARITHMETIC

An interpretive scheme to facilitate the programming of operations on double-length floating point numbers. Special facilities for matrix arithmetic are provided.

Name: DL. FP. ARITHMETIC

Store: 32 Blocks plus a 4 block interlude. The interlude, which can be overwritten when input is complete, can be beyond block 127 of the Main Store.

Uses: U 1, 2, 3, 4, 5; X 1, 6, 7; B0; working space in Main Store (see section 1); certain blocks down from B 510 if subroutines are used (see section 7.4).

Cues: 01, 02, 03, 04, 05 (see sections 3, 7.3, 9).

Time: See section 2.3.

Exit: See end of section 2.3 and section 3.

CONTENTS

1. Numbers and Store

Numbers are stored in the Main Store working space. They are in floating point form and occupy three locations: the first two locations contain the normalized or

zero double length argument (most then least significant half) while the third holds a binary exponent as an integer. The address of a number refers to the first word, the other two being at the same position on the following two blocks; it is relative to the start of the working space which is preset by a parameter (see section 5).

The addresses of numbers must differ by multiples of three, preferably all addresses should be divisible by three, starting at zero, as shown below:

Address:	Contents:	
0 + .0	α_0	(m.s. half)
.1		
.2		
.3	α_1	(m.s. half)
.4		
.5		
.6	α_2	(m.s. half)
.7		
1 + .0	α_0	(l.s. half)
.1	α_3	(m.s. half)
.2		
.3	α_1	(l.s. half)
.4	α_4	(m.s. half)
.5		
.6	α_2	(l.s. half)
.7	α_5	(m.s. half)
2 + .0	ϵ_0	
.1	α_3	(l.s. half)
.2	α_6	(m.s. half)
.3	ϵ_1	
.4	α_4	(l.s. half)
.5	α_7	(m.s. half)
.6	ϵ_2	
.7	α_5	(l.s. half)

α = argument ϵ = exponent

Most operations are carried out on these numbers and the contents of the floating point accumulator. The accumulator occupies U 1.1, 1.3 and 1.5 and holds numbers in the same form as the Main Store.

2. Orders

2.1 General Description

The basis of the scheme is the interpreted order. Interpreted orders are written and punched in a form similar to ordinary Pegasus orders and are read into the Main Store by Initial Orders. When obeyed they are in U0. The interpreting routine examines them and performs the specified operation. In U0 interpreted orders are obeyed consecutively except when a jump takes place. The next order obeyed after an interpreted order in 0.7+ is in 0.0, whether it is to be interpreted or not.

2.2 Structure and Modification

As with ordinary orders there are 19 binary digits to the order and thus two orders to the word. The stop-go digit is non-functional for interpreted orders. They will be described by the same leters (N, X, F, M,) as ordinary orders except when the first ten digits are thought of together. In this case the ten digits as an integer are designated S. S is usually the address (relative to the start of the

working space) of a number while s is the number stored at this location. The floating point accumulator will be denoted by A and its contents by a. a always represents a floating-point number as described in section 1. Thus for example:

$$\overbrace{N \quad X}^{S} \quad F \quad M$$

12 - 40

means transfer the contents of S 12 to A or $a' = s_{12}$. The modifier can only be X3, X4 or X5 and these accumulators can be used with the counting functions. In all modification the thirteen digits of the modifier position of the accumulator M are added to the first ten digits of the order (i.e. S or N and X) which are extended by three digits except in the case of F56. Jump orders always jump to an address in U0 however modified. Modification cannot cause overflow.

2.3 Order Code

F	Operation	mean times in milliseconds
00	$a' = a + s$	about 51
02	$a' = a + s$, set indicator if $a' = a$ or s	"
04	$a' = a + s$	"
06	$a' = a + s$, set indicator if $a' = a$ or s	"
10	$a' = a - s$	about 51
12	$a' = a - s$, set indicator if $a' = a$ or $-s$	"
14	$a' = s - a$	"
16	$a' = s - a$, set indicator if $a' = s$ or $-a$	"
20	$C'(X)$ = Nth constant in list (see section 8)	32
22	Read a number to A and S	-
24	Punch CRLF followed by number from A	-
26	Punch two spaces followed by number from A	-
30	Test indicator: Jump IN to N if not set. Clear indicator	24
32	Test indicator: Jump OUT to N if not set. Clear indicator	26
34	Count: $x'_m = x_m + 3$, $x'_c = x_c - 1$; jump IN to N if $x'_c \neq 0$	26
36	Count: $x'_m = x_m + 3$, $x'_c = x_c - 1$; jump OUT to N if $x'_c \neq 0$	27
40	$a' = s$	38
42	$a' = -s$	38
44	$s' = a$	38
46	$s' = -a$	38
50	Block read. UO' = N(+); jump to 0.X	34
52	$a' = S$	26
54	Set indicator if exponent of $a <$ exponent of s	30
56	Set indicator if $\lvert a \rvert \leqslant 2^{-S}$	24
60	$a' = a \times s$	48
64	a' = integral part of a	about 29
70	$a' = \frac{a}{s}$	74
76	Subroutine operation S (see section 7)	-

NOTES:

F 02, 06, 12, 16, 30, 32, 54, 56; the indicator is the sign of U5.7; when set it is negative. The rest of U5.7 is used for other purposes.

F 34, 36; counting does not carry from x_c to x_m. No jump occurs if x_m overflows and a jump will always occur if accumulator 0 is specified.

F 30, 32, 34, 36; jump 'IN' means jump to an interpreted order, jump 'OUT' means jump to an ordinary order. The block part of the jump address is not used: all these functions will only jump to orders in UO.

F 30, 32, 64; the unused X digits in the order may take any value.

F 64; integral part of a is less than or equal to a; e.g. integral part of -2.2 is -3.

F 70; loop stop with overflow in U3 and U4 on division by zero. (See section 10).

Unassigned orders; extravagant effects may be expected on obeying these.

The functions have been described with the second octal F-digit even. If the second function digit is made odd by the addition of 1 the next order is not interpreted but is obeyed in the normal way as a machine order. This applies to the order jumped to by F50. A sequence of machine orders may also be entered by a jump using F32 or 36.

Machine orders may use U0 and 4 and all the accumulators without disturbing the interpretive routine.

3. Entry and Exit

The routine has five cues. Cues 01 and 02 bring the interpreting routine into the Computing Store and jump to an order on U0. The address of the order must be specified in $X6_c$ before obeying the cue, by an instruction such as

```
0.P(+)   6 40
```

Cue 01 is used to jump to an ordinary order and cue 02 to jump to an interpreted order.

If the interpreting routine is in the Computing Store a sequence of interpreted orders is entered by setting the address of the first order in $X6_c$ and jumping to 1.7+ e.g. by obeying orders

```
0.P(+)   6 40
1.7+     0 60
```

If the interpreted order to be obeyed is in U0.0 all that is necessary is a jump to 1.7.

On exit from interpreted orders, X1 contains the most significant word of the accumulator (available for testing for sign or zero) and X2, 3, 4 and 5 will be unchanged since entry except for the result of counting or using F20 in any of the latter three.

The total extra time for entry and exit is 4 milliseconds.

4. Number Input and Output

4.1 Input

Function 22 reads a single number from tape and converts it to floating point form. The number is then put in the accumulator and in working space location S.

Numbers are punched as integers, fractions or with an integral and a fractional part and can be followed by a decimal exponent as an integer if required. Both argument and exponent must be signed and the whole number terminated by CR LF.

CR LF, LF and ϕ can precede the sign of the argument. Spaces may be punched anywhere except between digits or between the exponent and the terminal CR LF. One or more spaces must occur between the argument and the sign of the exponent if there is one. A decimal point may be punched anywhere in the argument and may precede or terminate it. Erase can occur anywhere except between CR and LF.

All other characters and arrangements cause a stop, (see section 10).

The argument may be punched with any number of digits. Up to twenty-two significant digits are accepted and any further digits are ignored. Numbers must be in the range $10^{-127} < |x| < 10^{127}$ or be equal to zero, otherwise input reaches a 77-stop in 2.2 but will continue on operating the STOP/RUN key.

During input the accumulators are stored in B0.

4.2 Output

Functions 24 and 26 punch out the number in the accumulator, 24 preceding it by CRLF, 26 by two spaces.

For these functions, the ten binary digits of S specify the style of output. The first five, S_1, indicate the number of digits required before the decimal point (unless $S_1 = 0$), while the last five indicate the number required after the point. A simple way of writing S in the order is as the decimal number $32S_1 + S_2$.

If $S_1 = 0$ the output is in the standard form, consisting of a signed argument n, where $1 \leqslant |n| < 10$, followed by a decimal exponent. The exponent is preceded by one space (and a sign) and is not followed by any spaces. If the argument is zero the exponent is punched as zero.

Output is rounded and signs are always printed. Left hand non-significant zeros are replaced by spaces. Right hand non-significant zeros are only replaced by spaces if the number is an exact integer. In this case the decimal point is also suppressed.

Unless the number is zero there is a 77-stop in 2.4+ if the binary exponent is outside the range $-512 \leqslant x < 512$. On continuing from this stop an asterisk is punched followed by 0.

If the number is greater than about 10^{22}, the standard form is used. This is done to avoid non-significant figures in the integral part.

Not more than 21 or 22 decimal digits are punched after and including the first non-zero digit. Spaces are then punched if necessary to complete the required number of characters after the decimal point.

Output is suitable for re-input by F22 provided the programmer ensures that each number is followed by CRLF.

During output the accumulators are stored in B0.

5. Preset Parameter

R 650 requires one preset parameter. This must contain in the a-order the address at which the Main Store working space is required to start and zero in the b-order, unless subroutines are being used under the control of F76 as described in section 7. Thus to set the beginning of the working space at B120.0 the parameter list required is

R 0	0 -0 1
650	- 04 -
120	0 00 0.
0	

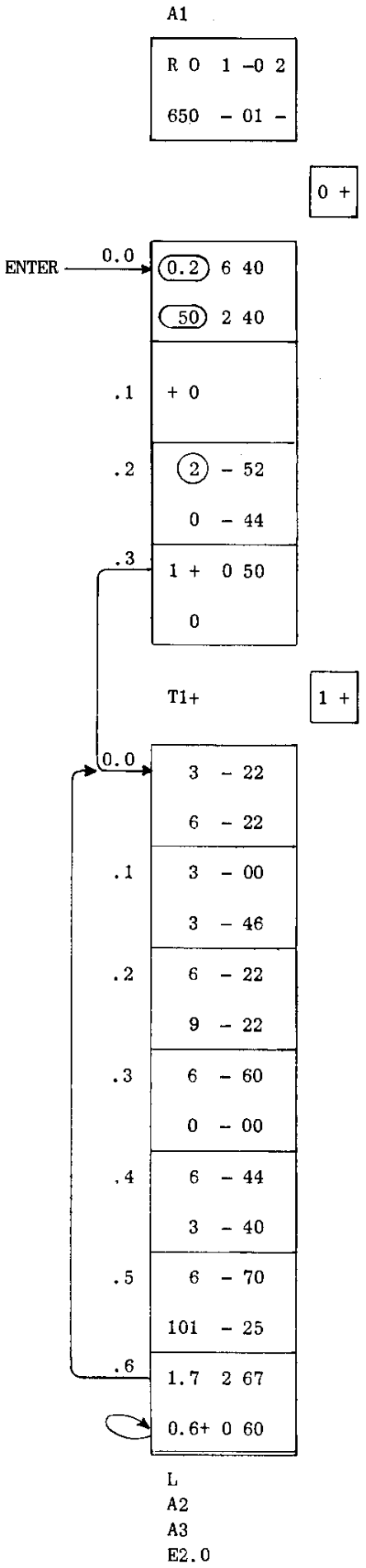
There is an optional parameter which is used if none is supplied by the programmer:-

```
125  7 00 0.
  0
```

This address of B125.7 is the largest that can be set for the working space.

6. Example

Given fifty sets of numbers x, y, z, w, calculate the numbers $-(x + y)/(zw + 2)$ and print the answers with three digits before the point and five after.



This tape is followed by one containing the fifty sets of numbers x, y, z, w.

7. Double-length Floating Point Subroutines

7.1 The Use of F76

The purpose of function 76 is to simplify the use of subroutines that work in conjunction with R 650.

If F76 is used the programmer must supply an R650 working space parameter with the number of different subroutine operations set in the counter position. This parameter must be followed by an index of cue specifications for the required operations. Each of these must hold the subroutine number and the cue number as integers in the a-order and the b-order respectively. For example:-

R 0 0 -0 1 650 - 04 -	
120 0 00 0. 2	working space address B120.0 2 subroutine operations
680 1	R 680 cue 01
680 2	R 680 cue 02

The F76-order carries out the operation appropriate to the S-th specification in the index. For example with the index as above the order

2 - 76

causes R 680 to be entered by cue 02.

When using subroutines in this way no link has to be set. After the operation specified by the 76-order has been completed control is returned to the next order. As with other orders the addition of 1 to the second function digit causes the next order to be treated as a machine order.

7.2 R 650 Subroutines

The following R 650 subroutines are on the same tape as R 650.

R 680: Linear combination

Store; 4 blocks
Cues: 01 $\underline{u}' = a\underline{v}$
02 $\underline{u}' = \underline{u} + a\underline{v}$
Time: cue 01, $126 + 43n$ milliseconds
cue 02, $126 + 58n$ milliseconds

Before entry set $x_4 = (u, \ )$
$x_5 = (v, n)$

The vector $\underline{u}$ is replaced by or has added to it a multiple of the vector $\underline{v}$. The vectors have n elements and the first elements have addresses u and v respectively.

If the elements of the vectors occupy locations that have a constant address difference other than 3, this difference should be set in 5.4_m. On exit U5.4_m is reset to 3 (see section 9).

R 681: Scalar product

Store: 3 blocks
Cues: 01 $a' = \underline{u}.\underline{v}$
02 $a' = a + \underline{u}.\underline{v}$
Time: cue 01, $117 + 59n$ milliseconds
cue 02, $131 + 59n$ milliseconds

Before entry set $x_4 = (u, \ )$
$x_5 = (v, n)$

The scalar product is formed of the vectors $\underline{u}$ and $\underline{v}$ each with n elements, whose first elements have addresses u and v respectively.

If the addresses of consecutive elements of $\underline{u}$ do not differ by 3, the address difference should be set in U 5.4_m. 5.4_m is reset to 3 on exit. Addresses of consecutive elements of $\underline{v}$ must differ by 3.

R 684: Square root

Store: 3 blocks
Cue: 01 $a' = \sqrt{a}$
Mean time 224 milliseconds

A loop stop (0.0+ 6 63) will occur if a is negative on entry.

R 685: Matrix Multiplication

Store: 2 blocks + 3 blocks for R 681
Cues: 01 D′ = B.C
02 D′ = B.C + D
Time: about $60pr\ (q + 3)$ milliseconds

Before entry set $x_3 = (d, p)$
$x_4 = (c, r)$
$x_5 = (b, q)$

The matrices B, C and D which are stored by rows have first elements at b, c and d and dimensions $p \times q$, $q \times r$, $p \times r$ respectively. R 685 uses R 681 as a subroutine. D cannot be the same matrix as either B or C. a is destroyed.

R 686: Matrix Division

Store: 5 blocks + 9 blocks for R 680, 681 and 685
Cues: 01 $C' = B^{-1}.C$, $B' = B^*$
02 $C' = C.B^{-1}$, $B' = B^*$
03 $C' = B^{-1}.C$, given B^*
04 $C' = C.B^{-1}$, given B^*
Time: $60pq\ (p + 3)$ milliseconds $(+ p(29p + 240)$ milliseconds for cues 01 and 02$)$

Before entry set $x_4 = (c, q)$
$x_5 = (b, p)$

The matrices B and C which are stored by rows have first elements at b and c. B has dimensions $p \times p$, C has dimensions $p \times q$ (cues 01 and 03) or $q \times p$ (cues 02 and 04).

The method of division is to replace the dividing matrix B by the basic information B* required to reduce it to the unit matrix by means of straightforward

elimination above and below the diagonal. This requires $\frac{1}{2}n^3$ arithmetical operations. No search is made for pivots. This information B* is all that is required to divide another matrix. To operate with it is similar to a matrix multiplication - and requires n^3 arithmetical operations. Thus once B* has been found it can be used again by cues 03 and 04 (for example to find B^{-2} C). The subroutine uses R 680 and R 685. a is destroyed.

NOTE: Function 76 ensures that on exit from these subroutines the Computing Store is unchanged apart from X 1, 6, 7 and the floating point accumulator A.

7.3 Programmer's Subroutines

Besides library subroutines, programmer's subroutines can be used with F76.

The subroutines may use interpreted orders and so operate on U 0 (and perhaps U 4) leaving the interpreting programme undisturbed, or otherwise use any of the rest of the Computing Store, but if U 1.1, 1.3, 1.5 are used the accumulator will be destroyed.

A subroutine should be written in the same form as an ordinary subroutine using assembly. The cues must be machine orders, for example

```
0+   0 72
1.7  0 60
```

to enter the subroutine at 0+.0 as an interpreted order. (See section 3).

During the process of entering a subroutine X 1, 6, 7 are used but the rest of the Computing Store remains unaltered. (X1 will not contain the most significant word of the accumulator at this stage.)

Return from a subroutine is effected by obeying cue 05 of R 650 which is called for in the normal way. The Computing Store, apart from X 1, 6, 7 and the accumulator A, will be restored to its condition on entry.

7.4 Notes on R 650 Subroutines

1. On proceeding to a 'lower level' with function 76 two blocks backwards from B510 are used temporarily to store information. For example a matrix division order will temporarily use B 505-510 if it is called in from the master programme, since R 686 uses R 685 which uses R 681.

2. The binary translation of R 680, 681, 684, 685 and 686 contains no names of the subroutines.

3. The list of cue specifications must not be written over as it is used each time F76 is obeyed.

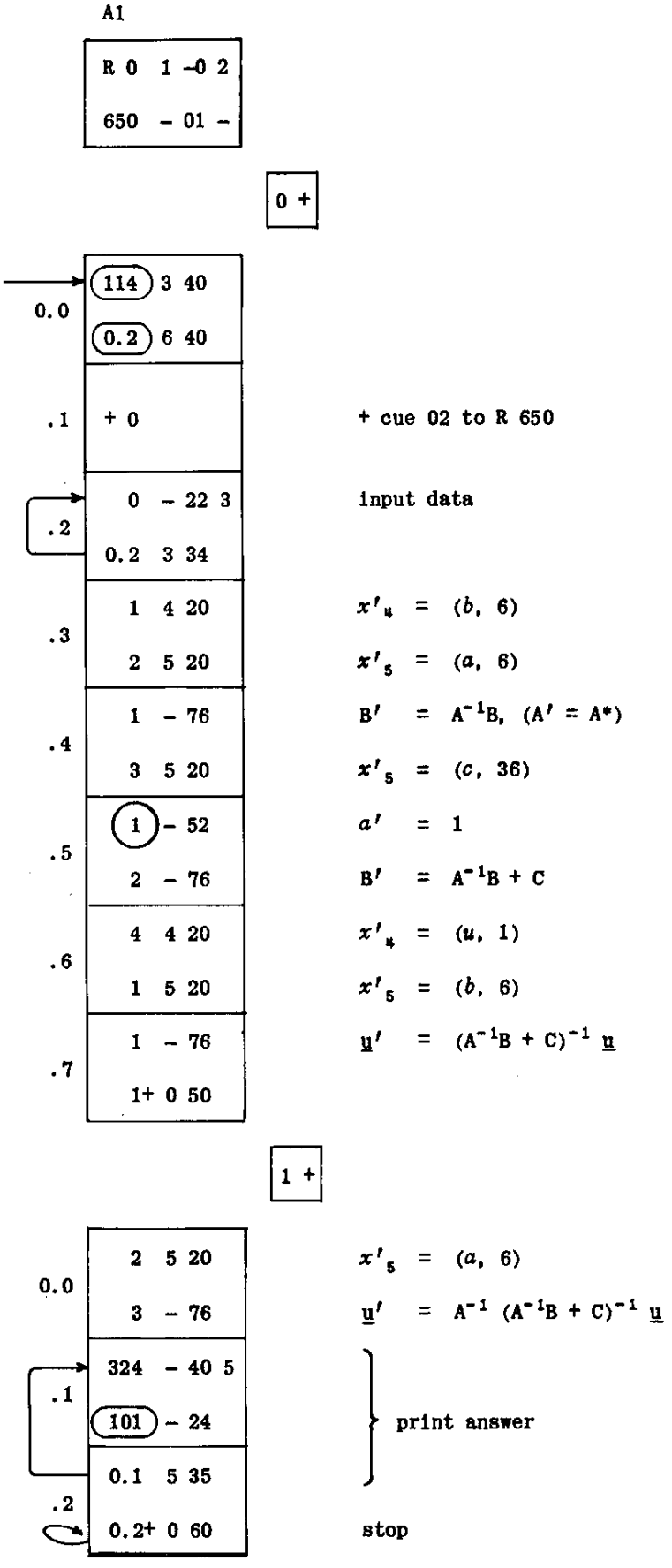
8. F 20

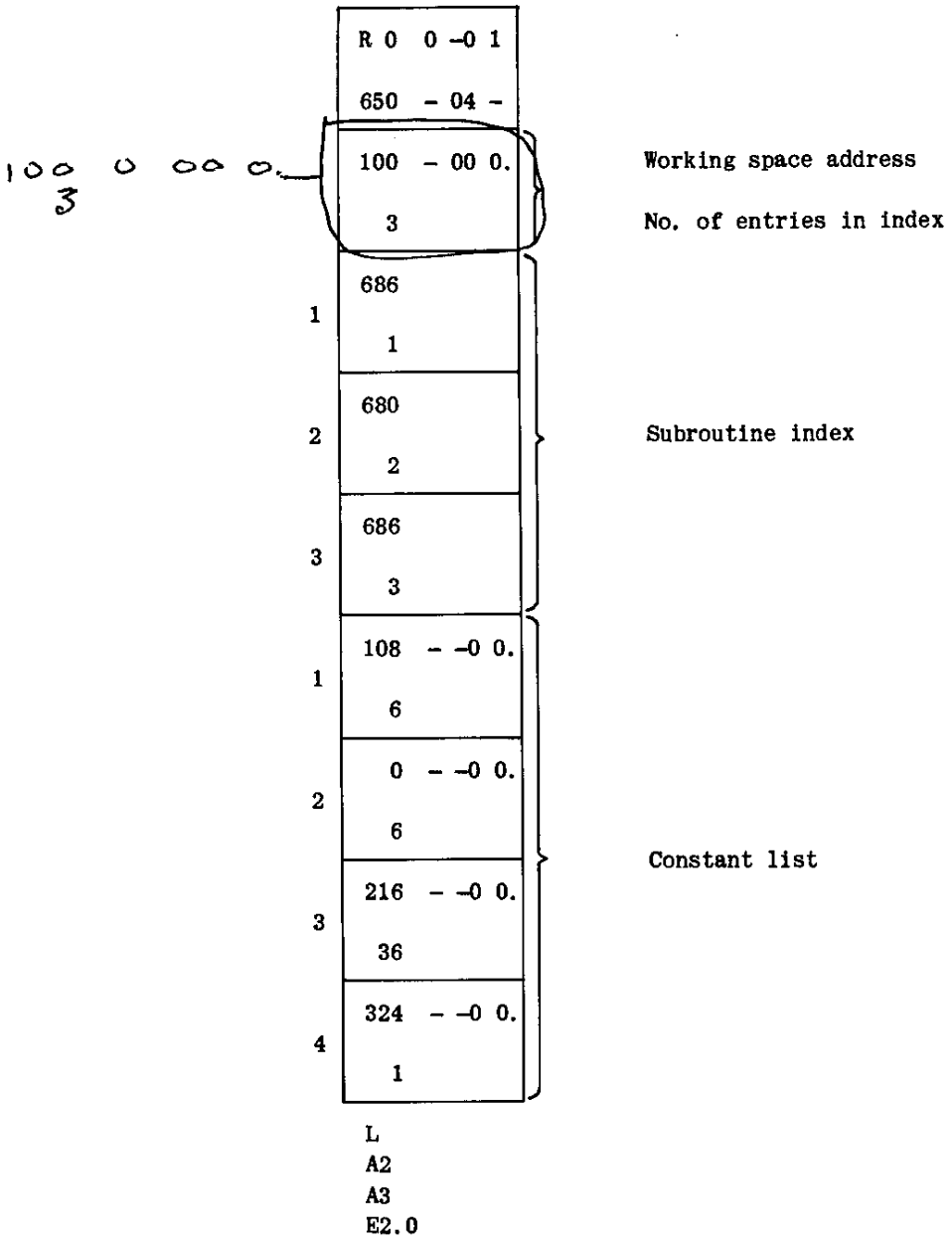
When using F 20 the programmer must supply a list of constants following the working-space address parameter and the subroutine index if any.

F 20 puts the Nth word in the list into accumulator X. (N = 1, 2, 3 .... etc.)

Example

Given square matrices A, B and C and a vector u punched by rows in this order on tape. The matrices are 6 x 6 and the vector 6 x 1. Find $A^{-1}(A^{-1}B + C)^{-1}$ u and print the answer with three digits before the point and five after.


A1
R 0 1 -0 2
650 - 01 -
0 +
114 3 40
0.0
0.2 6 40
.1
+ 0
+ cue 02 to R 650
0 - 22 3
.2
0.2 3 34
input data
1 4 20
.3
2 5 20
x'4 = (b, 6)
x'5 = (a, 6)
1 - 76
.4
3 5 20
B' = A^-1B, (A' = A*)
x'5 = (c, 36)
1 - 52
.5
2 - 76
a' = 1
B' = A^-1B + C
4 4 20
.6
1 5 20
x'4 = (u, 1)
x'5 = (b, 6)
1 - 76
.7
1+ 0 50
u' = (A^-1B + C)^-1 u
1 +
2 5 20
0.0
3 - 76
x'5 = (a, 6)
u' = A^-1 (A^-1B + C)^-1 u
324 - 40 5
.1
101 - 24
print answer
0.1 5 35
.2
0.2+ 0 60
stop



This tape is followed by the data tape.

9. General Notes

1. Cues 03 and 04 contain the address of the start of R 650 in the N and X portion of the a-order and the b-order respectively.

2. U 5.4 normally contains 3×2^{-13}. This constant is added to the specified accumulator during functions 34 and 36 in order to step on the modifier. It can be set from programme to any desired value. For example it can be set negatively for working backwards through the store or perhaps to some multiple of 3×2^{-13} for referring to a column or the diagonal of a matrix.

However, it is also used by the subroutines that perform matrix and vector operations and must normally contain 3×2^{-13} before a 76-order concerning these is obeyed. Exceptions to this are described in section 7.2. U 5.4 is reset to 3×2^{-13} after all 76-orders.

3. The next order obeyed after an interpreted order in 0.7+ is in 0.0, whether it is to be interpreted or not.

4. The overflow register is cleared by the interpreting programme and is always clear on exit.

5. Zero set by F52 or input by F22 has an exponent of -2^{19}. Zero resulting from the subtraction of equal numbers has an exponent 76 less than the exponent of the numbers. The exponent of zero resulting from multiplication or division is the sum or difference respectively of the exponents of the two numbers concerned.

6. No allowance is made for the possibility of any exponent overflowing.

7. An optional stop may be inserted in B 23+.2 to facilitate the development of programmes. The computer will reach this stop before obeying each interpreted order: at this point 7_m will contain the order number with the a/b digit reversed and X6 will contain the order pair being obeyed. On initial entry to the interpretive programme the stop will occur once before X6 and 7 have been set.

8. If the machine stops when an order is being obeyed the address of the next order will be found in 1.6_m.

9. If the programmer wants the address of the Main Store working space, he should call for R 650 parameters 01 or 02 which have the address in the N and X positions of the a- and b-order respectively.

10. The main programme of R 650 is followed by 4 blocks of interlude. These can all be beyond B127. While R 650 and its subroutines are being read B 0.0 and B 0.2 are used.

The interlude ensures that R 650 subroutines are accepted. Also if a non-optional parameter has been supplied:-

(a) Certain words are inserted into the main subroutine concerned with F20 and F76.

(b) If there is a subroutine index it is processed so that later Assembly inserts the specified cues into it.

(c) The two optional parameters are replaced by similar words containing the programmer's working-space address, and the programmer's parameter is overwritten by information concerned with subroutine entries. Thus the true parameter list is always in the location of the optional list while the location of the programmer's non-optional parameter becomes part of the list of subroutine entries and therefore is not available to the programmer.

10. Stops

10.1 During Input of Numbers

(a) Loop stops in 2.1, 3.4, 3.5+, 4.6+ for punching errors.

(b) 77 stop in 2.2 unless either $10^{-127} < |x| < 10^{127}$ or $x = 0$.

10.2 During Output

77 stop in 2.4+ if the binary exponent, ϵ, is outside the range $-512 \leqslant \epsilon < 512$.

10.3 Division Order

Closed cycle 4.5 - 4.7 - 3.0 - 3.1 - 4.5 .... etc. on division by zero.

10.4 F20

77 stop in 4.6 if no parameter supplied by programmer. If the STOP/RUN key is operated a loop stop will occur in 4.6+.

10.5 F76

77 stop in 4.0 if no parameter supplied by programmer. This stop will be repeated if the STOP/RUN key is operated.

Author: Mr. J.G.F. Francis of the National Research Development Corporation.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
11.12.57.

ARITHMETIC WITH RATIONAL FRACTIONS

A subroutine with three modes of entry to carry out the operations of addition, multiplication and cancellation on rational fractions.

Name: RATIONAL ARITHMETIC

Store: 5 blocks.

Uses: U0, 1; X1, 4, 5, 6, 7.

Cues:
01 Addition.
02 Multiplication.
03 Cancellation.

Times:
Addition: $(61 + 7m_1 + 7m_2)$ milliseconds,
Multiplication: $(56 + 7m_1 + 7m_2)$ milliseconds,
Cancellation: $(14 + 7m)$ milliseconds,
where m, m_1, m_2 are the numbers of cycles of the Euclid algorithm carried out in the computation. (See notes below.)

Link:
Addition: obeyed in U 1.3.
Multiplication: obeyed in U 1.5.
Cancellation: obeyed in U 1.0.
} Not left in X1.

1. Method of Use

1.1 Conventions

Rational fractions are represented as pairs of integers which usually lie in successive registers. In the case of addition and multiplication certain restrictions are placed on these integers:

(i) The numerator, n, must satisfy

$$-2^{38} \leqslant n < 2^{38}$$

(ii) The denominator must always be positive and non-zero

$$0 < d < 2^{38}$$

(iii) The numerator and denominator must be expressed in their lowest terms, i.e. their highest common factor, (n, d), $= 1$.

The only restriction applicable to the cancellation sequence is that

$$-2^{38} < n, d < 2^{38}$$

Neither n nor d may be equal to -2^{38} before cancellation.

The results of each sequence satisfy the above rules and are acceptable as operands for the other sequences.

1.2 Addition Cue 01

This sequence carries out the operation represented by

$$\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{N}{D}$$

where n_1, d_1 are previously put into X4 and X5 respectively, and n_2, d_2 into X6 and X7. The result N, D appears in X4 and X5.

The contents of X6 and 7 after the operation are 0 and D respectively.

The computation is carried out as follows:-

We write $d_1 = d_1'g, \quad d_2 = d_2'g$ where $g = (d_1, d_2)$

$$N_0 = n_1 d_2' + n_2 d_1'$$

Then $$\frac{N}{D} = \left(\frac{N_0}{g}\right)_c \cdot \frac{1}{d_1' d_2'}$$

where the symbol $(N_0/g)_c$ indicates that cancellation has been carried out. The numerator of this expression is N and the denominator will be represented by g', i.e. $D = g' d_1' d_2'$.

1.3 Multiplication Cue 02

This sequence carries out the operation represented by

$$\frac{n_1}{d_1} \cdot \frac{n_2}{d_2} = \frac{N}{D}$$

where (as in the case of addition) n_1, d_1, n_2, d_2 are previously put into accumulators X4, 5, 6, 7 respectively. The result is left in X4 and 5.

The contents of X6 and 7 after the operation are 0 and D respectively.

The computation is carried out as follows:-

We write $n_1 = n_1'g, \quad d_2 = d_2'g$ where $(|n_1|, d_2) = g > 0$

and $n_2 = n_2'h, \quad d_1 = d_1'h$ where $(|n_2|, d_1) = h > 0$

The result is given by $N = n_1' n_2', \quad D = d_1' d_2'$.

1.4 Cancellation Cue 03

This sequence takes a pair of integers representing a rational fraction, which may not be in the conventional form specified in 1.1, and leaves a pair with the same numerical value which does satisfy the conventions; i.e. the denominator is positive and the numerator and denominator are in their lowest terms.

The operation actually carried out corresponds to the expressions

$$\pm gN = n,\ gD = |d| \qquad \text{where} \quad g = (|n|,\ d) > 0$$

and where n, d are the input pair (set in X6, 7) and N, D are the output pair (which replace n, d in X6, 7). The minus sign on the L.H.S. is taken when $d < 0$. The H.C.F. g is left in U0.0.

The H.C.F. g of n and d is determined by use of the Euclid algorithm; the time taken by the cancellation sequence (which is also used by the addition and multiplication sequences) depends on the number of cycles needed to complete the algorithm.

Accumulators 4 and 5 are left unaltered by the cancellation sequence; they may thus be used for accumulation of a sum (using the addition sequence) while the cancellation sequence is employed for the standardisation of new fractions, which are then added in.

2. Subtraction and Division

2.1 Subtraction of rational fractions may be carried out using cue 01 (as for addition). The operation is represented by

$$\frac{n_1}{d_1} + \left(\frac{-n_2}{d_2}\right) = \frac{N}{D}$$

where n_1, d_1 and d_2 are previously put into X4, 5 and 7, and $-n_2$ into X6.

2.2 Division of rational fractions may be carried out using cue 02 (as for multiplication). The operation is represented by

$$\frac{n_1}{d_1} \cdot \frac{d_2}{n_2} = \frac{N}{D}$$

where n_1, d_1, d_2 and n_2 are in X4, 5, 6 and 7 respectively.

The contents of X7 must be positive on entry. If n_2 is negative division must be represented by

$$\frac{n_1}{d_1} \cdot \frac{-d_2}{-n_2} = \frac{N}{D};$$

n_1, d_1, $-d_2$, $-n_2$ must be set in X4, 5, 6, 7 on entry.

3. Error Stops

1.1

```
5  7 20

1.1+ 6 61
```

Loop stop for overflow of $g'd_1'$ or $g'd_1'd_2'$ during addition.

1.2

```
32  7 01

1.2+ 6 61
```

Loop stop for overflow of $n_1'n_2'$ during multiplication.

1.4	1.4 6 61 7 5 00	Loop stop for overflow of $d_1'd_2'$ during multiplication.
1.4	1.4 7 60 0.1 7 62	Loop stop if denominator is zero on entry to cancellation routine.
1.6	32 7 01 1.6+ 6 61	Loop stop for overflow of $n_1d_2' + n_2d_1'$ during addition.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
London, W.1.

Issue 1
11th December, 1957.
A.R.E.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
10.4.58

OUTPUT CONVERSION

A self-preserving subroutine to convert an integer of up to 6 digits into 6-bit characters packed into a word, as used with the magnetic tape Bull printer.

Name: OUTPUT CONVERSION

Store: 1 block, plus 2 blocks and 2 locations for an interlude.

Uses: U0; X 3, 4, 5, 6, 7.

Cues: 01 a-order partial cue.
02 b-order partial cue.

Time: If n is the number of decimal digits specified and r is the number actually present, the time is approximately

$$(n + 3r) \text{ milliseconds.}$$

Link: The subroutine always exits to U 1.0 so no link is required.

1. Method of Use

The integer to be converted should be placed in X6. The subroutine can be brought down at any time by the order

0 [0] 72	+ R 700 cue

which must be tagged by the appropriate partial cue. It is self-resetting and is entered by the order

0.1 0 60

The resulting characters will be left in X5. There will be from 2 to 6 of them, in order of significance, with the last one at the least significant end of the word. For example if the integer 123 is to be converted X5 will be left as

No. of bits	3	6	6	6	6	6	6
Contents	0	0	0	0	1	2	3

2. Preset Parameter

A parameter list must be provided to specify the number of characters required. It should take the form

R 0 0 -0 1
700 - 04 -
+*n*

n must be in the range 2 to 6 inclusive.

3. Zero-suppression

Non-significant zeros will (ultimately) appear as spaces, since 0 is the code for 'space' on the Bull printer. As the code for zero is 48 the subroutine will replace significant zeros by 48.

The integer zero will be correctly converted into one '48' character preceded by zeros, i.e. spaces.

4. Errors

If the integer in X6 has more than *n* digits, it will not be correctly converted even if the number of digits is less than 7.

5. Stops

During the interlude there will be a loop stop if *n* is not in the range 2 to 6:

0.7 1 63	if $n < 2$

1.0 1 62	if $n \geqslant 7$

The interlude may be stored beyond block 127 if required.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
10th April, 1958.
J.F.D. D.M.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
11.3.58.

INPUT CONVERSION 1

A self-preserving subroutine to convert a word of n 6-bit characters, as used with the magnetic tape Bull printer, into an n-digit integer; where $n \leqslant 6$.

Name: INPUT CONVERSION 1

Store: 2 blocks.

Uses: U0, 1; X1, 6, 7 on entry 1
U0, 1, 5; X1, 5, 6, 7 on entry 2

Cues:

01

```
0+[0]72
1+[1]72
```

02 a-order partial cue.

03 b-order partial cue.

Time: 13 milliseconds on entry 1
(4 + 10k) milliseconds on entry 2.

Link: A computing store link must be set in the counter position of X1.

Method of Use:

The two blocks of subroutine can be brought down to U0 and U1 at any time by obeying cue 01.

The subroutine is self-preserving and can be entered in the following manner:-

Entry 1 Word to be converted in X6

Enter by `0.0 0 60`

The subroutine leaves the integer in X6.

Entry 2 Words to be converted in U5

X5 = (0.P,k); where P is the position in U5 which holds the first word, and k is the number of words. ($P \leqslant 7$; $k \leqslant 8 - P$)

Enter by | 0.1 0 60 |

The subroutine replaces the k words in U5 by the corresponding integers. It does not change any other words in U5.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
11th March, 1958.
C.M.B.L. R.P.S.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
27.11.57

MERGING SORT BY MAXIMUM STRING METHOD

This routine sorts into ascending order n non-negative numbers in locations A to $A + n - 1$ of the Main Store. Locations B to $B + n - 1$ are used as working space. The sorted sequence is left in locations A to $A + n - 1$.

Name: MAX. SORT

Store: 7 blocks

Uses: U 0, 1, 2, 3, 4, 5.3 to 5.7; X 2, 3, 4, 5, 6, 7; B0

Main Store locations A to $A + n - 1 + \epsilon_A$
and B to $B + n - 1 + \epsilon_B$ $\quad 0 \leqslant \epsilon \leqslant 7$
where $A + n - 1 + \epsilon_A$ and $B + n - 1 + \epsilon_B$ are the last words in the blocks containing $A + n - 1$ and $B + n - 1$ respectively.

Cue: 01

```
0+[0]72

0.0  0 60
```

Time: Approximately $(6.5n + 80)\left\lceil \log_2 \frac{n}{t} \right\rceil_u + 6\frac{n}{t} + 22 + (4n + 30)\delta$ milliseconds

where $\left\lceil \log_2 \frac{n}{t} \right\rceil_u = \log_2 \frac{n}{t}$ rounded up to the next highest integer.

$\delta = 0$ if $\left\lceil \log_2 \frac{n}{t} \right\rceil_u$ is even

$= 1$ if $\left\lceil \log_2 \frac{n}{t} \right\rceil_u$ is odd

t = Average starting string length.

For random data $t = \frac{2n}{n + 1}$ and the formula reduces to

$(6.5n + 80)\left\lceil \log_2 \frac{n + 1}{2} \right\rceil_u + 3n + 25 + (4n + 30)\delta$ milliseconds.

For data in descending order $t = 1$; for data already sorted $t = n$.

For random data:

n	10	25	100	250	500	1000	2000
Time (seconds)	.56	1.1	4.7	14	28	66	137

The term $\left[\log_2 \frac{n}{t}\right]_u$ in the formula is an estimate of the number of merges required for the complete sort. When t is not near to $\frac{2n}{n+1}$ the number of merges may rise to $\left[\log_2 \frac{n}{t} + 1\right]_u$. Even with random data this may happen when $\log_2 \frac{n}{t}$ is close to the next highest integer.

Link: Obeyed in 0.7 and left in X1 on exit.

Parameters: n, A and B are programme parameters which must be set before entry as follows:

	Modifier	Counter
X6	A	n
X7	B	n

A and B are not restricted to the start of a block, but they must be chosen so that there is no overlap between locations A to $A + n - 1 + \epsilon_A$ and B to $B + n - 1 + \epsilon_B$.

If $n = 0$ or 1 the routine will obey the link without having disturbed the Main Store.

On exit U 5.6 will contain (A, n)

Notes:

(1) If it is required to store the accumulators before entering the sub-routine they must not be stored in B0, since this is required as working space.

(2) If OVR is set on entry there will be a write with OVR stop in 4.3, due to a 73 order in 4.2+.

(3) Numbers to be sorted may be zero but must not be negative. If negative numbers are included there may be a write with OVR stop in 2.2. If this stop does not occur the negative numbers will cause the routine to enter a closed loop from 0.2 to 2.7.

Ferranti Ltd.
London Computer Centre,
21, Portland Place,
LONDON, W.1

Issue 1
27th November, 1957
M.L.B.L. D.M.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
3.4.58.

MERGING SORT OF KEYWORDS WITH ONE DATA-WORD

This routine sorts into ascending order n non-negative numbers in even locations A, A+2, A+4,....., A+2n-2 of the Main Store, with associated data-words in A+1, A+3,....., A+2n-1. Locations B to B+2n-1 are used as working space. A and B must both be even numbered addresses. The sorted sequence is left in locations A to A+2n-1. A maximum string merging process is used.

Name: DATA SORT

Store: 8 blocks

Uses: U0, 1, 2, 3, 4, 5.3 to 5.7; X2, 3, 4, 5, 6, 7; B0.

Main Store locations A to $A+2n-1+\epsilon_A$ and B to $B+2n-1+\epsilon_B$, $0 \leqslant \epsilon \leqslant 6$

where $A+2n-1+\epsilon_A$ and $B+2n-1+\epsilon_B$ are the last words in the blocks containing $A+2n-1$ and $B+2n-1$ respectively.

Cue:

01

```
0+[0]72

0.0  0 60
```

Time: $\frac{1}{4}\left\{(41n+360)\left[\log_2\left(\frac{n}{t}+1\right)\right]_u + 74\left(\frac{n}{t}\right) + 142 + (33n+100)\delta\right\}$ milliseconds

where $\left[\log_2\left(\frac{n}{t}+1\right)\right]_u$ = $\log_2\left(\frac{n}{t}+1\right)$ rounded up to the next highest integer.

δ = 0 if $\left[\log_2\left(\frac{n}{t}+1\right)\right]_u$ is even.

= 1 if $\left[\log_2\left(\frac{n}{t}+1\right)\right]_u$ is odd.

t = Average starting string length.

For random data, $t = \frac{2n}{n+1}$ and the formula reduces to

$\frac{1}{4}\left\{(41n+360)\left[\log_2\left(\frac{n+3}{2}\right)\right]_u + 37n + 179 + (33n+100)\delta\right\}$ milliseconds

For data in descending order t = 1; for data already sorted t = n.

For random data:

n	10	25	100	250	500	1000
Time (seconds)	.82	1.66	7.7	23	46	111

The term $\left[\log_2\left(\frac{n}{t}+1\right)\right]_u$ in the formula is an estimate of the number of merges required for the complete sort. When t is not near to $\frac{2n}{n+1}$ the number of merges may rise to $\left[\log_2\left(\frac{n}{t}+1\right)+1\right]_u$. Even with random data this may happen when $\log_2\left(\frac{n}{t}+1\right)$ is close to the next highest integer.

Link: Obeyed in 0.7 and left in X1 on exit.

Parameters: n, A and B are programme parameters which must be set before entry as follows:

	Modifier	Counter
X6	A	n
X7	B	n

A and B are not restricted to the start of a block, but they must be even numbered addresses, and be chosen so that there is no overlap between locations A to $A+2n-1+\epsilon_A$ and B to $B+2n-1+\epsilon_B$.

If n = 0 or 1 the routine will obey the link without having disturbed the Main Store.

On exit U5.6 will contain (A-1.0, n).

Notes:

(1) Numbers to be sorted (i.e. keywords) may be zero but must not be negative. There is no such restriction on data words. If negative keywords are included the routine will enter a closed loop from 0.2 to 2.7, or write with OVR in 2.4, due to a 73 order in 2.3+.

(2) If OVR is set on entry, there will be a writing with OVR stop in 4.3, due to a 73 order in 4.2+.

(3) If it is required to store the accumulators before entering the routine they must not be put into B0, since this is required as working space.

(4) If A or B or both are odd numbered addresses the routine will enter a large closed loop.

Author: Mr. M.W.G. Duff of Robson, Morrow and Company.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
11.6.58.

DOUBLE-LENGTH MERGING SORT

This routine sorts into ascending order n non-negative double-length numbers in locations A to $A + 2n - 1$ of the Main Store. Locations B to $B + 2n - 1$ are used as working space. The sorted sequence is left in locations A to $A + 2n - 1$. A and B must both be even numbered addresses. A maximum string merging process is used.

Name: D.L. SORT

Store: 9 blocks.

Uses: The entire Computing Store except 5.1 and 5.2; B0

Main Store locations A to $A + 2n - 1 + \epsilon_A$ and B to $B + 2n - 1 + \epsilon_B$ $\quad 0 \leqslant \epsilon \leqslant 6$
where $A + 2n - 1 + \epsilon_A$ and $B + 2n - 1 + \epsilon_B$ are the last words in the blocks containing $A + 2n - 1$ and $B + 2n - 1$ respectively.

Cue: 01

```
0+[0]72

0.3  0 60
```

Time: Approximately $(12n + 140)\left[\log_2 \frac{n}{t}\right]_u + 33\frac{n}{t} + 22 + (8n + 30)\delta$ milliseconds

where $\left[\log_2 \frac{n}{t}\right]_u$ = $\log_2 \frac{n}{t}$ rounded up to the next highest integer.

δ = 0 if $\left[\log_2 \frac{n}{t}\right]_u$ is even

= 1 if $\left[\log_2 \frac{n}{t}\right]_u$ is odd

t = Average starting string length.

For random data $t = \frac{2n}{n+1}$ and the formula reduces to

$(12n + 140)\left[\log_2 \frac{n+1}{2}\right]_u + 16n + 38 + (8n + 30)\delta$ milliseconds.

For data in descending order $t = 1$; for data already sorted $t = n$.

For random data:

n	10	25	100	250	500	1000
Time (seconds)	1.1	2.2	10	26	57	133

The term $\left[\log_2 \frac{n}{t}\right]_u$ in the formula is an estimate of the number of merges required for the complete sort. When t is not near to $\frac{2n}{n+1}$ the number of merges may rise to $\left[\log_2 \frac{n}{t} + 1\right]_u$. Even with random data this may happen when $\log_2 \frac{n}{t}$ is close to the next highest integer.

Link: Obeyed in 4.7. Not left in X1 on exit.

Parameters: n, A and B are programme parameters which must be set before entry as follows:

	Modifier	Counter
X6	A	n
X7	B	n

A and B are not restricted to the start of a block, but they must be even numbered addresses, and be chosen so that there is no overlap between locations A to $A + 2n - 1 + \epsilon_A$ and B to $B + 2n - 1 + \epsilon_B$.

If $n = 0$ or 1 the routine will obey the link without having disturbed the Main Store.

On exit U 5.6 will contain (A, n)

Notes:

(1) If it is required to store the accumulators before entering the sub-routine they must not be stored in B0, since this is required as working space.

(2) If OVR is set on entry there will be a write with OVR stop in 1.0.

(3) Numbers to be sorted may be zero but must not be negative. If negative numbers are included there may be a write with OVR stop in 0.6. If this stop does not occur the negative numbers will cause the routine to enter a closed loop from 0.0 to 2.7.

(4) All numbers must be proper double-length numbers, with the second half of the number positive.

(5) If A or B or both are odd numbered addresses the routine will enter a large closed loop.

Ferranti Ltd.
London Computer Centre,
21, Portland Place,
London, W.1

Issue 1
11th June, 1958
D.M.

R 723

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
11.7.58.

N-LENGTH MERGING SORT

This routine sorts into ascending order n N-length numbers in locations A to $A + nN - 1$ of the Main Store. Locations B to $B + nN - 1$ are used as working space. The sorted sequence is left in locations A to $A + nN - 1$. A fixed string merging process is used. If $N = 1$ or 2, this routine is much slower than the special routines R 720 and 722.

Name: N-LENGTH SORT

Store: 9 blocks.

Uses: The entire Computing Store except X1; B0.

Main Store locations A to $A + nN - 1 + \epsilon_A$ $\quad 0 \leqslant \epsilon \leqslant 7$
and B to $B + nN - 1 + \epsilon_B$

where $A + nN - 1 + \epsilon_A$ and $B + nN - 1 + \epsilon_B$ are the last words in the blocks containing $A + nN - 1$ and $B + nN - 1$.

Cue: 01 (0+.0) Enter Routine for sorting

02 a-order partial cue

03 b-order partial cue

Time: $\{(9k + 6d + 3)n + 140\}\{[\log_2 n]_u + \delta\} + 29n$ milliseconds

where k *(keyword) = Number of words to be compared before numbers differ $(1 \leqslant k \leqslant N)$

d (data) = $N - k$

$[\log_2 n]_u$ = $\log_2 n$ rounded up to the next highest integer

δ = 0 if $[\log_2 n]_u$ is even

= 1 if $[\log_2 n]_u$ is odd

The following table gives typical sorting times in seconds with $k = \frac{1}{2}N$

N \ n	10	25	100	250	500	1000
1	1.3	2.9	14	32	76	150
2	1.6	3.7	18	44	106	210
3	1.9	4.8	26	62	151	-
4	2.2	5.6	30	74	181	-
6	2.8	7.4	42	104	-	-
8	3.4	9.3	54	134	-	-
16	5.8	17	102	-	-	-
50	15	47	-	-	-	-
127	37	-	-	-	-	-

Note that in this table a slightly more accurate formula was used when $k > 8$.

Link: Obeyed in 0.7 and left in X1 on exit.

1. Programme Parameters

1.1 n, A and B are programme parameters which must be set before entry as follows:

	Modifier	Counter
X6	A	n
X7	B	n

On exit (A,n) will be left in X6 and also in U 0.0. The sign bit of X6 and 7 may be 0 or 1 on entry, but on exit X6 will always be positive.

1.2 A and B are not restricted to the start of a block, but they must be chosen so that there is no overlap between locations A to $A + nN - 1 + \epsilon_A$ and B to $B + nN - 1 + \epsilon_B$

1.3 If $n = 0$ or 1 the routine will obey the link without having disturbed the Main Store.

2. Preset Parameters

2.1 N, the number of locations occupied by each number, is specified by Preset Parameter 01. N may not exceed 127. The parameter list should be punched as follows:-

R 0 0 -0 1 723 - 04 -	Title of parameter list
N 0 00 0	N = Length of numbers being sorted

If no parameter list is supplied by the programmer, N will be set equal to 3 by an optional parameter list.

2.2 If it is required to change N during the programme, words 4+.0 and 6+.0 of the subroutine must be altered.

4+.0 should read

```
(N)  7 40 6
0.4  7 61
```

6+.0 should read

```
(N)  6 40
 6   7 20
```

The amended words may be stored by the orders

```
[4  0]71        + partial cue

[6  0]71        + partial cue
```

these orders being tagged by cue 02 (a-orders) or cue 03 (b-orders).

3. Restrictions on Numbers

3.1 Numbers may be all positive or all negative, where the term 'positive' includes zero numbers. Mixed positive and negative numbers may be sorted provided that the difference between the smallest and largest first words does not overflow. This means that the range of numbers, when regarded as N-length fractions, must be less than $(1.0 - 2^{-38})$. If this restriction is broken the routine may encounter a write with OVR stop in 1.0, 1.2 or 1.6: if these stops do not occur the routine will have sorted correctly.

3.2 All numbers should be proper N-length numbers, with zero sign bits in all but the first word. If this restriction is broken the routine will not sort correctly on the negative words, and it may encounter a write with OVR stop in 1.0, 1.2 or 1.6.

3.3 Although the routine sorts on the whole N-length numbers, it may be used to sort k-length keywords each with d data words. The data words should still be positive as described in section 3.2 above. The data words will be compared only when the corresponding keywords are equal.

4. Notes

4.1 If it is required to store the accumulators before entering the subroutine they must not be stored in B0, since this is required as working space.

4.2 If OVR is set on entry there will be a write with OVR stop in 1.6.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
London, W.1.

Issue 1
11th July, 1958
D.M.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
11.4.58.

SUMMARISING SORT OF KEYWORDS WITH ONE DATA-WORD

This routine sorts into ascending order n non-negative keywords in locations A, $A+2$, $A+4$,....., $A+2n-2$ of the Main Store, with associated data-words in $A+1$, $A+3$, ..., $A+2n-1$; it combines equal keywords and adds the corresponding data-words. Locations B to $B+2n-1$ are used as working space. The sorted sequence is left in locations A to $A+2m-1$, where m ($\leqslant n$) is the number of distinct keywords. A maximum string merging process is used.

Name: SUMSORT 2

Store: 10 blocks.

Uses: U0, 1, 2, 3, 4, 5.0, 5.3 to 5.7; X2, 3, 4, 5, 6, 7; B0.
Main Store locations A to $A+2n-1+\epsilon_A$ and B to $B+2n-1+\epsilon_B$ } $0 \leqslant \epsilon \leqslant 6$

where $A+2n-1+\epsilon_A$ and $B+2n-1+\epsilon_B$ are the last words in the blocks containing $A+2n-1$ and $B+2n-1$ respectively.

Cue:

01

```
0+ 0 72

0.0  0 60
```

Time: Approximately $(16n + 11n \log_2 m + 1000)$ milliseconds for randomly arranged keywords, provided $m \geqslant 2$. If $m = 1$ all keywords are equal and no sorting, only summarising, is required.

The following table gives typical times in seconds:

n \ m	1	2	8	16	32	64	128	n
25	.14	1.2	2	2.3	-	-	-	3.2
100	.54	3.5	6	7	8.2	9	-	10
250	1.4	8	13.5	16	19	23	25	29
800	4.2	23	40	49	58	67	76	99

Note that this table is based on experimental results and does not agree exactly with the formula. If the keywords are partially sorted on entry to the routine the time will be considerably reduced.

Link: Obeyed in 0.7 and left in X1 on exit.

Parameters: n, A and B are programme parameters which must be set before entry as follows:-

	Modifier	Counter
X6	A	n
X7	B	n

A and B are not restricted to the start of a block, but they must be even numbered addresses, and be chosen so that there is no overlap between locations A to $A+2n-1+\epsilon_A$ and B to $B+2n-1+\epsilon_B$.

If n = 0 or 1 the routine will obey the link without having disturbed the Main Store.

On exit X5 will contain (A, n), and X2 the final value of the necessary modifier and counter, (A, m).

Notes:

(1) Keywords may be zero but must not be negative. If negative keywords are included the routine will enter a closed loop from 0.0 to 2.7, or write with OVR in 0.6, 1.5 or 2.7.

(2) Data-words may be positive or negative. If their addition causes overflow, the routine will write with OVR in 0.6, 1.5 or 2.7.

(3) If OVR is set on entry, there will be a writing with OVR stop in 4.3.

(4) If it is required to store the accumulators before entering the routine they must not be put into B0, since this is required as working space.

(5) If A or B or both are odd numbered addresses the routine will enter a large closed loop.

Author: Mr. M.W.G. Duff of Robson, Morrow and Company.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
11th April, 1958.
M.W.G.D. D.M.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

SUMMARISING SORT (INDEXING)

A sorting routine which requires that n single-length keywords should be presented to it one at a time to be stored and indexed. Each keyword may have a number of data words associated with it. If two keywords are equal an auxiliary subroutine may be entered to combine the corresponding items. When all m such combined items have been indexed a different section of the subroutine may be used to extract items one at a time in ascending keyword order. R 737 is normally slower than R 736 but can deal with more items and more data per item.

Name: MONKEY PUZZLE MARSHAL

Store: 4 blocks.

Uses: U0, 1, 2; X1, 5, 6, 7 on cue 01
U0, 1; X1, 3, 7 on cues 02 and 03
Plus two words of Main Store for each keyword plus space for data words (sections 1.2 and 1.5).

Cues:

01	0+ [0] 72 0.6 0 60	Index keyword in X7
02	3+ [0] 72 0.0 0 60	Extract smallest keyword
03	2+ [1] 72 1.0 0 60	Return from Auxiliary to extract next keyword
04	a-order partial cue	
05	b-order partial cue	

Time: If n items in random keyword order are read, and if items with equal keywords are combined to form m distinct items, the average time for each entry to R 737 would be

Part 1 (Cue 01) $.02 \log_2 m$ seconds ($m > 1$)

Part 2 (Cues 02 and 03) .06 seconds

The total time used by R 737 would therefore be about $.02\ n \log_2 m + .06\ m$ seconds (this formula does not apply when $m = 1$).

If the keywords are not random the time for part 1 may increase: in the worst case, with keywords completely pre-sorted, the term $\log_2 m$ would become $\frac{1}{2}m$.

For random data the following table gives typical times in seconds:

n \ m	1	2	8	32	128	512	1024	$m = n$
25	.5	.6	2.0	-	-	-	-	3.8
100	1.6	2.1	6.5	12	-	-	-	20
250	3.8	5.1	16	27	43	-	-	55
800	12	16	49	82	120	175	-	202
2000	30	40	120	200	290	390	460	560
10000	150	200	600	1000	1400	1830	2060	-

Link: Part 1 (Cue 01) Obeyed in U 0.6. Not left in X1.

Part 2 (Cue 02) Obeyed in U 1.7. Not left in X1.

1. READING KEYWORDS

1.1 Keywords

Keywords must be such that overflow can not occur on subtracting any one from any other: it would be sufficient, for example, if they were all positive.

1.2 Store

The master programme has to specify an even numbered address A which is the start of the working space for the subroutine. A may not be B 0.0. R 737 requires $2m$ locations of working space, two for each distinct keyword, plus space for data (section 1.5).

Thus $$2 \leqslant A < N - 2m$$

where N = 4096 or 7168, the number of words available in the Main Store.

1.3 First Keyword

On reading the first keyword the master programme must clear location A and place the keyword in location $A + 1$.

1.4 Subsequent Keywords

Subsequent keywords must be presented by obeying cue 01 to R 737 with the following information in the accumulators:

X1 Link, must be reset before each entry.

3 Sign and counter clear

3_m A First Address (Even-numbered)

X3 is not overwritten by R 737 and A need only be set once by the master programme.

5 Sign and counter clear

5_m B Current Address (Even-numbered)

At each entry R 737 will use locations B and $B + 1$ and leave $5_m = B' = B + 2$. Unless the master programme is storing data after keywords it should set $5_m = B = A + 2$ initially and not disturb 5_m thereafter.

7 The new keyword.

1.5 Storing Data

On each entry to R 737 by cue 01 a tagword is stored in B and the current keyword in $B + 1$. On entry $5_m = B$; on exit $5_m = B' = B + 2$. In general there will be d words of data associated with each keyword which may be dealt with in one of the following ways:

1.5.1 Interlacing

The d data words may be stored in B' to $B' + d - 1$ provided that 5_m is increased to $B' + d + \delta$ before the next entry to R 737. $\delta = 0$ or 1 to make $B' + d + \delta$ even. d may vary from item to item.

1.5.2 Indexing

If d is constant the d data words may be stored in D to $D + d - 1$, where D is a simple function $C + \frac{1}{2}d(B - A)$ of B.

$D = (C - \frac{1}{2}dA) + \frac{1}{2}dB$ $\quad [= (C - \frac{1}{2}dA - d) + \frac{1}{2}dB']$

C = First address for data.

Both A and B should be even-numbered addresses and the above formula should therefore never give a fractional D. This method has the advantage that it is not necessary to allocate an even number of locations for data.

2. IDENTICAL KEYWORDS

When the current keyword is found to be the same as a previous one preset parameter 01 is obeyed.

2.1 No Merging

If identical keywords are to be marshalled consecutively but separately, without merging the associated items, preset parameter 01 should be

```
1.2+ 0 60

  0
```

The identical keywords will then be marshalled in the order in which they are presented.

2.2 Merging Items

If items with identical keywords are to be merged preset parameter 01 should be a cue to Auxiliary 1, a merging subroutine which must be provided by the user. On entry to Auxiliary 1 the following information will be available in the computing store:

X1	Link to return to the master programme (As set on entry to R 737)
2,3,4,5	As left by master programme
6	B_i, where $B_i + 1$ is the address of the previous keyword K_i, which is identical with the current keyword K_c
7	Zero
U2	The block containing K_i, the previous keyword (K_i can be extracted by an order 2.1 7 00 6)
U3,4,5	As left by the master programme.

Auxiliary 1 should merge the two items and leave the data for the combined item in the Main Store in place of the data associated with K_i. It should then return to the master programme to read the next item by obeying the link left in X1.

It may use	U0,1,2; X1,6,7	without restriction
	U3,4,5; X2,4	if not required by the master programme.
	X3	only if reset before re-entry to R 737.

It may not use X5.

Note that the number of words of data in the combined item may not exceed the number of locations allocated for the data associated with K_i, which it replaces.

3. OUTPUT OF KEYWORDS

3.1 Entry

To initiate output of a set of keywords cue 02 to R 737 should be obeyed with the address A in 3_m and a link for return to the master programme in X1.

3.2 Auxiliary Output Subroutine

The user must provide Auxiliary 2, a subroutine to deal with the sorted items, which are presented to it one at a time. For each item R 737 obeys preset parameter 02, which must be a cue to Auxiliary 2. Auxiliary 2 will find the address B_i in 3_m, where $B_i + 1$ is the address of the keyword: it can therefore read the keyword into X1 by the orders

```
| 0  1 | 70 3 |
```

OR

```
| 0  2 | 72 3 |
| 2.1  1 00 3 |
```

Auxiliary 2 may use the entire computing store. When it has dealt with an item it should obey cue 03 to R 737.

3.3 Exit

When all items have been dealt with R 737 will obey the link set in X1 by the master programme.

4. INDEPENDENT SETS OF DATA

Two or more quite independent sets of data may be handled at the input stage, with the keys, tags and data for each set interlaced. As each keyword of a given set is presented to R 737 the appropriate address A must be put into 3_m, corresponding

to the first keyword in the set. If a different merging process is required for the different sets, Auxiliary 1 must use 3_m or some other marker to decide which procedure to follow.

At the output stage R 737 must be entered by cue 02 once for each set, with the appropriate address A in 3_m. R 737 may not be used to output any one set of keywords more than once.

5. PRESET PARAMETERS

The preparation of the parameter-list is made easier if Auxiliary 1 and Auxiliary 2 are combined into one subroutine with two cues. The parameter list may then be punched as part of the combined subroutine and relative addresses may be used in parameters as follows:

R 0 0 -0 2 737 - 04 -	Title of parameter-list
+ □72 0 60	Cue to Auxiliary 1 (Merging) (or 1.2+ 0 60 if no merging)
+ □72 0 60	Cue to Auxiliary 2 (Output)

Note that if the auxiliaries are made up as assembly subroutines the master programme must include an assembly tag calling for them in order to satisfy Assembly. If this is done the parameter-list may be punched at the end of the programme tape, just before L A2, and the relative addresses set by punching a C-directive before each parameter.

6. CHANGING PARAMETERS

If it is desired to use R 737 more than once on one run it may be necessary to change the preset parameters.

P.P. 01 is planted in 1+.6

P.P. 02 is planted in 3+.5

New parameters may therefore be written to these locations from X1 by obeying the orders

1 6 71	+ partial cue
3 5 71	+ partial cue

each tagged by the appropriate partial cue, 04 or 05, to R 737.

7. METHOD OF COMPUTATION

7.1 The Tree

The process can best be explained by considering a tree, with a keyword and an associated tagword at each branch point. At each branch point there is one branch going down and there may be two branches going up, one to the left and one to the right.

The first keyword, together with its tagword is placed at the bottom of the tree. The positions of all subsequent keywords can be described by three store addresses: those of the two keywords above and the one below it in the tree. A zero address indicates that a branch point is not occupied. These three addresses are stored in the tagword associated with the keyword. The keywords are stored in the Main Store in the order in which they are presented and the 'geometry' of the tree is entirely defined by the addresses in the tagwords.

7.2 Climbing Up

The process of inserting a new keyword into the tree is as follows. Start at the bottom of the tree and test the new keyword against each keyword encountered. Go left up the tree if the new keyword is the lesser, right if it is the greater. Carry on up the tree until a vacant branch point is found, and place the new keyword there, inserting addresses in tagwords as required.

7.3 Climbing Down

On first entry by cue 02 the smallest keyword is found by starting at the bottom and going up the tree, turning left always, until the end of the leftmost branch is reached.

As each keyword is output its tagword is marked and its address is recorded for the next entry from Auxiliary 2.

On each entry by cue 03 the routine starts from the last keyword output. It goes upwards to the right one place if there is a keyword there and then goes up the tree, turning left, until the end of a branch is reached.

If there is no keyword going upwards to the right, the routine goes down the tree until an unmarked tagword is reached: this is the next keyword. When the bottom of the tree is reached and the bottom tagword is already marked the process is complete.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

COMPREHENSIVE P.A.Y.E. CALCULATION

This subroutine evaluates income tax for weekly or monthly paid employees who have the following types of tax rating:-

1. Normal Coding.
2. Low Income Staff not subject to P.A.Y.E.
3. Emergency Coding.
4. Week 1 (Month 1) Basis Coding.
5. Code '0'.
6. Code 'Standard Rate'.
7. Code 'No Tax'.

Name: P.A.Y.E. - MK.2.

Store: 23 blocks (including all constants as preset parameters).

Uses: U0, 1, 2, 4, 5; 3.3, 3.4; All Accumulators.

Cues:

01	0 [0] 72 0.1 0 60	Weekly entry when B0+ of R 740 is stored in B127 or below
02	0 [0] 72 0.1+ 0 60	Monthly entry as in 01
03	0 [0] 72 2 0.1 0 60	Weekly entry when B0+ of R 740 is stored in B128 or above. (See section 3).
04	0 [0] 72 2 0.1+ 0 60	Monthly entry as in 03

05	0+ - 00 0. 0	Modifier to be set in X2 before using cue 03 or 04

Time: The following are some typical times for weekly paid employees. Times for monthly paid employees are very similar. The values are approximate.

Normal coding	$(79 \leqslant t \leqslant 136)$ milliseconds
Emergency coding	$(95 \leqslant t \leqslant 152)$ milliseconds
L.I.S.	16 milliseconds
Code '0'	$(81 \leqslant t \leqslant 97)$ milliseconds
Code 'S.R.'	81 milliseconds

Link: Must be set in X1 and is obeyed in 2.7. It is left unchanged in X1 on exit.

1. METHOD OF USE

On entry to the subroutine, the current week's (month's) pay p_n (in pence) must be set in X7, and the number, n, of the week must be set in 3_c. Block 3 of the Computing Store must contain the following information:-

U3.1	Tax coding	See section 2
U3.2	A_1 (pence)	Table A value for week (month) 1.
U3.3	P_{n-1} (pence)	Cumulative pay to the end of the previous week.
U3.4	t_{n-1} (pence)	Cumulative tax to the end of the previous week.

On exit from the subroutine, the current week's (month's) tax, $+T_n$, will be held in X7. U3.3, 3.4 will contain the cumulative pay and tax, respectively, to the end of the present week (month). If X7 is negative on exit, a refund is due to the employee.

2. TAX CODING

2.1 The table below shows in what form the tax coding should be stored in U3.1 on entry. Where the coding is Normal or Week (Month) 1 Normal Basis, i.e. where there is a numerical value, n, attached to the coding, this number, which must be a positive integer in the range $1 \leqslant n \leqslant 225$, is held in the last 8 bits of the word. In all other cases these last 8 bits must be zeros. If the routine is entered with any other coding, it will come to a loop stop in 2.6.

In the table below, bits that are not specified must be zeros.

Type of Coding	Tax coding in binary
Normal Coding	0.000000.....$\overbrace{\cdots\cdots}^{n}$ (8 bits)
Low Income Staff (not subject to P.A.Y.E.)	1.000000.....
Emergency Coding	1.1000000.....
Week 1 / Month 1 — Normal Coding	1.0100000.....$\overbrace{\cdots\cdots}^{n}$ (8 bits)

Type of Coding	Tax coding in binary
Code '0'	1.0010000.....
Code Standard Rate	1.0001000.....
Code No Tax	1.0000100.....

2.2 The binary form of the tax coding and the Table A value will normally be stored with the employees permanent record. Another subroutine, R 741, has been written to form these numbers when the record is being compiled or revised.

3. ENTRY

3.1 If BO+ of R 740 is to be stored in B 127 or below in the Main Store then entry is in the normal way using either cue 01 or 02.

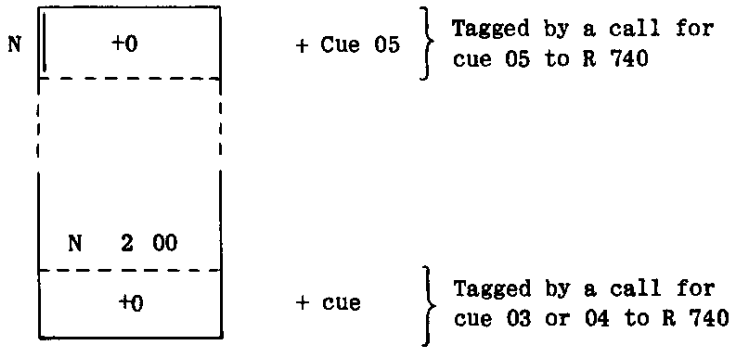
3.2 If BO+ is to be stored in B128 or above then the following procedure must be obeyed:-

Call for, but do not obey, cue 05 which is to be used as a modifier.

Place this modifier in X2

Call for cue 03 or 04

The Master Programme should therefore appear as follows:-



If cue 01 or 02 is called for when BO+ of R 740 is stored above B127, then on entry to the subroutine, the computer will come to a 77-stop. On going to RUN it will come to an unassigned order stop (33 order).

4. PRESET PARAMETERS

A parameter list of the following form must be punched with the master programme:-

			1959/60 Values
	R 0 0 -3 4 740 - 04 -	Title of Parameter List	
01	L_W	Low Income Staff Upper limit. Weekly	+900
02	L_M	Monthly	+3720
03	1.7 0 60 0	LOOP STOP or cue to exception routine when L.I.S. limit exceeded	
04	$+a$	Rates of weekly pay at which steps in the official tables change from 5/- to 10/- (04), and from 10/- to £1 (05)	+2400
05	$+b$		+3840
06	$+d$	Rates of monthly pay at which steps in the official tables change from 5/- to 10/- (06), from 10/- to £1 (07), and from £1 to £2 (10).	+2400
07	$+g$		+4800
10	$+h$		+9600
11	$+T_x$	The amount to which month 1 table proceeds in £2 steps.	+40800
12	$+T_y$	The amount to which week 1 table proceeds in 5/- steps.	+9600
13	$+k$	Rates of gross pay for the week at which pay steps change from 5/- to 10/-, and 10/- to £1	+2400
14	$+m$		+4800
15	$+x$	Rates of gross pay for the month at which pay steps change from 10/- to £1, and £1 to £2	+4800
16	$+y$		+9600
17	$+r_1$	First	+21
20	$+r_2$	Second reduced rates of tax (pence per pound).	+51
21	$+r_3$	Third	+75
22	$+s$	Standard rate of tax (pence per pound).	+93
23	$+B_1$	Widths of the three reduced rate bands	+14400
24	$+B_2$		+36000
25	$+B_3$		+36000
26	$+(e_2 - e_1)$	$\frac{e_1}{e_2} = e =$ the full rate earned income relief fraction $\frac{f_1}{f_2} = f =$ the reduced rate earned income relief fraction	+7
27	$+(f_2 - f_1)$		+8
30	$+e_2$		+9
31	$+f_2$		+9
32	$+T_E$	Maximum earned income qualifying for earned income relief at: full rate	+961200
33	$+T_L$	reduced rate	+2386800
34	$+C$	Single persons personal allowance	+33600
35	$+Z$	Rate of gross pay for the month at which pay steps change from £5 to £1	+80400

Note: All sterling sums must be punched in pence.

5. STOPS

1.7	1.7 0 60 0	Loop stop if Low Income Staff limit exceeded and no exception routine provided.
2.6	2.6 5 60 0 7 00	Loop stop if unacceptable tax coding in U3.1.

6. NOTES

Income Ranges for which R 740 is applicable:

Normal Coding: Tax is calculated on income up to T_E (£4,005 in the 1959/60 values) at the full rate of earned income relief, from T_E to T_L (T_L is £9,945 for 1959/60) at the reduced rate of earned income relief, and above T_L with no earned income relief.

Emergency Coding: Tax is calculated, as described above, on equivalent weekly or monthly incomes, at the same rates of earned income relief.

Low Income Staff:

These employees are generally not subject to P.A.Y.E. deductions. There is an upper limit for weekly and monthly pay in this coding, above which tax becomes payable. These upper limits are, in the 1959/60 values, £3.15.0 per week, and £15.10.0 per month (preset parameters 01 and 02 respectively).

If R 740 is entered with any employee coded as Low Income Staff, having a weekly or monthly income exceeding the limit, then preset parameter 03 will be encountered, which can be either a loop stop as shown in the list, or can be a cue to some exception routine written to deal with this particular case.

Inland Revenue:

It is suggested that before incorporating R 740 in any wages programme, the user should apply to the Inland Revenue for their documents on P.A.Y.E. calculations on electronic computers (obtainable from the Statistics and Intelligence Division), and also for the list of associated constants for the current year, which are required by the parameter list. It would be useful to have also the 'Employer's Guide to "Pay as you Earn"' issued by the Board of Inland Revenue. This gives details of every aspect of P.A.Y.E.

Ferranti Ltd., London Computer Centre, 21, Portland Place, LONDON W.1.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

P.A.Y.E. CODE INPUT AND TABLE A LOOK UP

This subroutine reads and converts income tax codings into the form required by R 740 and looks up the Table A value where appropriate. The converted code is left in X7 and the Table A value (in pence) in X6. These values would normally be stored with the employees record and used weekly or monthly on entry to R 740.

Name: CODE READ

Store: 12 blocks.

Uses: U0, 1, 2; B0; X2, 7.

Cue:

```
01   0+ [0] 72
     0.0  0 60
```

Time: The following are some typical times. The values are approximate.

C *n* CRLF	123 milliseconds
E CRLF	97 milliseconds
W1 C *n* CRLF	116 milliseconds

Link: Obeyed in 2.7 and left unchanged in X1.

1. METHOD OF USE

Income Tax Table A must be held in the Main Store, each element of the table being in pence and stored in consecutive locations. Before entering R 741, the address of the first word of the table must be set in 2_m.

Note: Code number n = 65, for which there is no Table A value, must be given some arbitrary value to ensure that there are no discontinuities in the table.

Table A entitled "Free Pay" is obtainable from the Inland Revenue.

2. FORMS OF PUNCHING

ϕ, Sp, Er, LF, CRLF are ignored before the coding. Er only is ignored between the λ at the start of the code and the terminating character. Either CRLF or Sp may be used as a terminating character.

Note: Where the coding is Normal or Week (Month) 1 Normal Basis, i.e. where there is a numerical value, n, attached to the coding, this number, which must be an integer in the range $1 \leqslant n \leqslant 225$, is held in the last 8 bits of the word. In all other cases these last 8 bits will be zeros.

Type of Coding	Form of Punching
Normal Coding	λ C ϕ n CRLF
Low Income Staff (not subject to P.A.Y.E.)	λ LI ϕ CRLF
Emergency Coding	λ E ϕ CRLF
Week 1 } Normal Coding	λ W ϕ 1 λ C ϕ n CRLF
Month 1 } Normal Coding	λ M ϕ 1 λ C ϕ n CRLF
Code '0'	λ C ϕ 0 CRLF
Code Standard Rate	λ SR ϕ CRLF
Code No Tax	λ NT ϕ CRLF

3. ERROR STOPS

0.0	(0 7 73)	A stop on writing with overflow will be encountered in 0.0 if R 741 is entered with OVR set.
0.3+	(0.3+ 5 61)	If W or M not followed by ϕ 1 λ C
0.4	(0.4 5 61)	If coding not terminated by Sp or CRLF
0.4+	(0.4+ 5 63)	If an inadmissible character before first λ
1.3+	(1.3+ 5 61)	If C not followed by ϕ
1.4	(1.4 5 63)	If $n > 225$
1.5	(1.5 7 60)	If $n = 0$ for week or month 1
1.7	(1.7 5 61)	If S not followed by R
2.0+	(2.0+ 4 60)	If no decimal digits after λ C ϕ
2.2	(2.2 5 61)	If an inadmissible coding, or if L not followed by I, or if N not followed by T.
2.5	(2.5 5 61)	If incorrect terminating characters.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

FILE UPDATING MK.2

This routine deals with the magnetic tape organisation for file updating work using variable length items. It brings each amendment item and the corresponding main file item into the Main Store and then enters one of two auxiliary subroutines provided by the user: an amendment subroutine and an insertion subroutine.

R 761 is not self-preserving in the Main Store, and therefore cue 01 may only be called for once each time R 761 is read.

Name: FILE UPDATING MK.2

Store: 53 blocks.

Uses: In general the whole Computing Store, but at certain re-entry points some parts are left undisturbed. See below.

Cues:

01 (41+.0)	Initial entry
02 (21+.2)	Return 1 from S1 (section 4) and Return 1 from S2 (section 5)
03 (21+.4)	Return 2 from S1
04 (52+.0)	Return 2 from S2
05	a-order partial cue
06	b-order partial cue

Time: Exact times are difficult to calculate, but the following should be a sufficient guide.

Let A be the number of blocks in the Output Buffer (see section 6, P.P.02)
B be the number of keywords on the Main File not corresponding to any amendment keyword.
C be the number of keywords on the Main File with corresponding amendments.
D be the number of 32-word sections on the Main File.
E be the number of 32-word sections on the Amendment File.
F be the total number of blocks in the Amendment Buffer (see section 6, P.P.07).
G be the number of repeated keywords in the Amendment File.

Then the time, in milliseconds, used by R 761 is

$$2B + 193C + D\left(166 + \frac{300}{A - 8}\right) + E\left(133 + \frac{400}{F - 8}\right) + 40G$$

As an example, for a main file of 5000 items, of average length 48 words (almost 3000 feet of tape), with 500 amendments of average length 8 words, all different, with reasonably long amendment and output buffers, the time would be about 23 minutes; to this would be added the time required for the programmer's subroutines.

Link: Should be preset as preset parameter 21. It is obeyed in 3.4 and is not left in X1. (R 761 is in any case not self-resetting, i.e. it may be used once only).

Contents

Appendices

1. General Concept

R 761 is used with a main file, on one or more 32-word magnetic tapes, consisting of a large number of separate items, each headed by a keyword (which, for example, might be a part number for a stock control job). The items must be in keyword order. There will also be an amendment file held similarly as separate items in keyword order on 32-word magnetic tape, but this will in general be a much shorter file as only a selection of the items on the main file will have counterparts on the amendment file. The amendment file items may have been supplied in random order on paper tape and have been sorted by the computer, or they may have been converted on to magnetic tape from sorted punched cards.

The basic file updating process on magnetic tape involves scanning a brought forward (B.F.) main file, together with the amendment file, and producing a carried forward (C.F.) main file. This will include most of the items on the B.F. file, but in general those for which a corresponding item exists on the amendment file will be altered, whilst those for which there is no amendment will be copied unchanged from the B.F. to the C.F. file. Also, some items may be deleted and others inserted.

The actual process of effecting an amendment will vary from job to job and is dealt with by a programmer's subroutine. R 761 deals with the scanning of the files, the finding of coincident keywords, and the detection of end of tapes, end of job etc.

2. ORGANISATION OF FILES

2.1 Form of items

R 761 uses variable length working for both main file and amendment items. This is described in List CS.180, but the main features will be given here.

The lengths of items are not affected by the sections on magnetic tape. Each item must be headed by its keyword, which must be of negative sign. The remaining words in the item must be positive (but see CS.180 if some data words represent negative quantities).

R 761 recognises the beginning of an item by the fact that the keyword is negative. Thus, apart from restrictions imposed by the overall size of the Main Store, an item may be of any length up to 1023 words (127.7 blocks), provided that this is allowed for in the preset parameters (section 6).

The keywords on the main file will generally all be different. If there is an amendment to a keyword which occurs more than once on the main file, R 761 will select the first of the identical keywords.

2.2 Form of keywords

Keywords should be in ascending order of magnitude; they are all to have negative signs and the recommended procedure is to add -1.0 to each keyword (e.g. by adding register 32).

Suppose there are keywords 100, 102, 108 in this order. These are stored as integers in Pegasus words, and adding -1.0 (as a fraction) to each gives

$$-1.0 + 100 \times 2^{-38}$$
$$-1.0 + 102 \times 2^{-38}$$
$$-1.0 + 108 \times 2^{-38}$$

These are still in correct order of magnitude and are all negative, thus satisfying the requirements. (Note that if the keywords were just negated, they would be in descending order of magnitude).

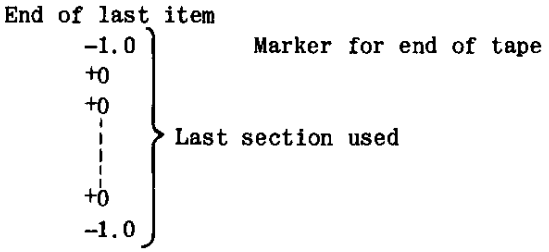
2.3 Arrangement of files on tape

A file is considered to be made up of a number of 32-word tapes. The required starting address on tape can be preset (see section 6) and will not normally be the first word of the first section, as at least one section will be reserved for permanent information about the tape itself. Also, in general the information on each tape will not go to the last available word on the tape, but a certain amount will be left unused.

In producing a C.F. file from a B.F. one, R 761 assumes that the contents of one tape goes on to one other tape; thus if a file tends to increase in length through the amendment process, there must be sufficient space at the end of the C.F. tape to take the excess. As the C.F. file from one run becomes the B.F. file for another, this gradual growth cannot go on indefinitely. When a tape becomes almost full, it will be necessary to re-organise the whole file, for instance 4 almost full tapes would have to be copied on to 5 tapes, each about 4/5 full. The re-organisation runs would, in general, be much less frequent than the updating runs.

2.4 End of tape

It is necessary for R 761 to be able to recognise the end of information on a tape. This is done by using a keyword -1.0, which is a zero keyword with a negative sign. The use of this as a real keyword is therefore banned. The -1.0 will follow immediately the last word of the last item on the tape. This will not in general be the last word of a section, but R 761 does not examine the remainder of the section on the B.F. file; the end of the section on the C.F. file will be made up as follows:



If the last item ends in the 31st word of a section, there will be only one -1.0 and the next section will not be written; if it ends in the last word, the next section will contain -1.0 in the first and last words and +0 in every other word.

The end of the information on an amendment tape must also be indicated by a -1.0, and R 761 does not examine anything beyond this. However, as the amendment tape is read several sections at a time, there must be at least n_{11} properly recorded 32-word sections on the tape after the final -1.0, where n_{11} is given by preset parameter 11 (section 6).

2.5 End of file

There is no special way of indicating end of file rather than end of tape; the number of amendment tapes is a preset parameter, and when a count on this terminates, R 761 assumes that it is on the last main file tape. (If it is not, the remaining tapes of the file do not require processing.)

3. THE MASTER PROGRAMME

The master programme will normally be short, as all the file updating will be done by R 761 and associated programmer's subroutines. It should check that the correct reels of tape have been put on the mechanisms, using R 930 or some similar routine, and then enter R 761 by cue 01 with the following programme parameters set:

X2 Tape mechanism number $\times 2^{-38}$ for C.F. file
X3 Tape mechanism number $\times 2^{-38}$ for B.F. file
X4 Tape mechanism number $\times 2^{-38}$ for amendment file

On return from R 761 the main job will have been completed, but it may be necessary for the master programme to print out summary information etc.

4. AMENDMENT SUBROUTINE (S1)

4.1 Entry

The amendment subroutine (S1) will be entered from R 761 when the main file item associated with the next amendment keyword is found. At this point both the amendment and the main file items will be in the Main Store.

S1 may use the entire Computing Store except the sign of X4. If it requires to use the tape buffer it may first have to preserve it (see Appendix A1.1). On entry the following information will be available:

X4 sign — if tape buffer is free
\+ if tape buffer is engaged (see Apendix A1.1)

$X4_m$ Address in Main Store of main file item to be amended. The block containing the first word (the keyword) will be in U4 as though the order 0 4 72 4 had been obeyed.

$X4_c$ The number of words in the main file item.

X5 sign +

$X5_m$ Address in the Main Store of the amendment item. The block containing the first word (the keyword) will be in U5 as though the order 0 5 72 5 had been obeyed.

$X5_c$ The number of words in the amendment item.

X6 The keyword on which coincidence has been found.

4.2 Function

The function of S1 is to use the information supplied in the amendment item to alter the B.F. item, thereby forming the C.F. item. It may write the C.F. item on the Main Store in place of the B.F. item, even if they differ in length, or it may write it to some other part of the Main Store. It may not, in general, leave the C.F. item in place of the amendment item even if they are identical (but see appendix A1.4).

When S1 has dealt with one amendment it will usually return to R 761 by cue 02. For this return the only information required of it is the sign of X4, the new address in 4_m and the length in 4_c of the amended main file item. The subroutine need not move the item from its original position in the store (as the B.F. item) even if its length is altered; in this case 4_m should be left unchanged by the subroutine. Similarly, the counter in 4_c will not be changed if the item length is unaffected by the amendment. The sign of X4 must be preserved (except as stated in appendix A1.1)

The main file item is copied out to the C.F. file using X4 as a counter; it is thus possible to split an item into two (or more) items, provided the total length of them is specified to R 761 in X4. (This would not be so if R 761 were at this point testing the sign of each word to detect end of item.) If 4_c is set equal to zero, the main file item will be deleted, i.e. nothing will be copied to the C.F. file.

4.3 Repeated keywords

It may happen that there is more than one amendment item for a particular main file item. To deal with this case, R 761, on return from S1, will test to see if the keyword for the next amendment is the same as for the one just dealt with; if so it will return to S1 without having written away the main file item from the Main Store.

The return may be to a different point in the amendment subroutine than the normal entry if desired.

5. ERROR AND INSERTION SUBROUTINE (S2)

5.1 Entry

A subroutine (S2) must be supplied to deal with errors and, if appropriate, with insertions. S2 will be entered when R 761 finds an amendment keyword which is not on the B.F. file (it recognises this situation when it finds that the next keyword on the B.F. file is greater than the current amendment keyword).

On entry to S2 the following information will be available:

X4 sign – if tape buffer is free
+ if tape buffer is engaged (see Appendix A2.1)

X5 sign +

$X5_m$ Address in Main Store of amendment item

$X5_c$ Number of words in amendment item

X6 Keyword.

Note that the first block of the amendment item will not be in the Computing Store.

S2 may use the whole Computing Store, except that in the case of an insertion it must preserve X5 and the sign of X4. If it requires to use the tape buffer it may first have to preserve it (see Appendix A2.1).

5.2 Error

An amendment keyword not corresponding to an existing keyword may be an error (the keyword may be either wrong or out of order). In this case a likely course of action is for S2 to print an indication (such as the erroneous keyword) and to return to R 761 via cue 02 to deal with the next amendment item. Note that the action of cue 02 is different in this case from that described in section 4.2, and there is no need for S2 to preserve X4 and 5 before obeying cue 02.

5.3 Insertion

If the item is not an error it must be a new one, to be inserted into the C.F. file. In this case S2 must copy the complete item to some other part of the Main Store, at the same time making any desired adjustment to it. It may not leave the amendment item in situ unless it first makes the tests described in Appendix A2.2. S2 should set the new address in 4_m and the number of words in the item in 4_c. The sign of X4 must be as left by R 761 (except as stated in Appendix A2.1). S2 should then obey cue 04 to R 761, which will copy the new item to the C.F. file.

6. PRESET PARAMETERS

R 761 requires a parameter list of 18 parameters, numbered in octal from 01 to 22. They should be as follows:

01 **Main Store Address of Input Buffer**

In the modifier position, with the sign and counter clear. It must be a multiple of 4 blocks, (i.e. it must be word 0 of a block number which is divisible by 4).

The input buffer is used by R 761 to hold the incoming tape section from the B.F. file. It may be any available sequence of 4 blocks, subject to the starting address being a multiple of 4 blocks, but the user must allow space beyond the input buffer for the maximum length of main file item, rounded up to an integral number of sections.

When a main file item requiring amendment is recognised, R 761 begins to read it into the Main Store starting at whatever address it had reached in the input buffer.

02 **Main Store Address of Output Buffer**

In the modifier position, with the sign and counter clear. It must be a multiple of 4 blocks.

The output buffer may be any available sequence of at least 20 blocks, but it must be a multiple of 4 blocks in length. It should be as long as possible in order to minimise computer time. It is used to build up sections for output to the C.F. file.

03 **Output Buffer Warning Address**

In the modifier position, with the sign and counter clear. It must be 12 blocks before the end of the output buffer, and must therefore be a multiple of 4 blocks.

R 761 uses this warning address to detect when it is necessary to move the incomplete section up to the top of the buffer.

04 **Cue to S1 (first entry)**

This is obeyed for the first (or only) amendment to a particular main file item.

05 **Cue to S1 (second entry)**

This is obeyed for the second and all subsequent amendments to a particular main file item.

06 **Cue to S2**

This is obeyed for an amendment keyword without a corresponding main file item.

07 **Address of Amendment Buffer**

In the modifier position, with the sign and counter clear. It must be a multiple of 4 blocks.

The amendment buffer must be at least twice as long as the longest possible amendment, and it is suggested that it should be no less than 20 blocks, but in any case it must be a multiple of 4 blocks in length. It is filled from tape whenever an address is reached such that it cannot be guaranteed that the next amendment is all in the Main Store.

10 **Amendment Buffer Danger Address**

In the modifier position, with the sign and counter clear. It must be a multiple of 4 blocks, and should be such that the longest possible amendment can be held in the amendment buffer above this address.

11 **Counter for Total Number of Sections in the Amendment Buffer**

In the counter position, with the rest of the word clear. Section, in this case, means multiple of 4 blocks.

12 **Counter for Number of Sections in the Danger Zone of the Amendment Buffer**

In the counter position, with the rest of the word clear. Section, in this case, means multiple of 4 blocks.

13 **Number of Sections in the Normal Part of the Amendment Buffer**

This should be the number of 4-block sections in the amendment buffer before the danger address. It must be in the *N* address position of the *a*-order with the rest of the word clear, for instance:-

```
10   0 00 0.

0
```

This implies that the normal part of the amendment buffer cannot be more than 127 sections (508 blocks) in length.

14 **Initial Tape Address for the Main Files**

In the tape address position for tape control words, with the rest of the word clear, for instance:-

```
10   - -- 0.

0
```

This specifies the first section used for file items on the main file tapes: it may not be section 0.

15 **Initial Tape Address for the Amendment Files**

In the tape address position for tape control words, with the rest of the word clear. This specifies the first section used for file items on the amendment file tapes: it may not be section 0.

16 **Mask for the Amendment Keyword**

It has been arranged that a mask can be collated into each amendment keyword before searching for coincidence. If this facility is not required the mask should be -1 (all ones).

17 **Address for Storing the Final T.C.W.s of the C.F. File**

In the modifier position.

To facilitate the organisation of a multi-tape file, it has been arranged for R 761 to store the final T.C.W. for each C.F. tape. These can then be printed at the end of the run, so that the final tape addresses used can be determined. The T.C.W.s will be stored in order starting at the specified address. If they are not required an address in the isolated store may be specified.

20 **Counter for Number of Amendment Tapes**

In the counter position. This will usually be +1.

21 **Link**

This is the order pair obeyed at the end of the updating.

22 **Mask for Main File Item**

A mask can be collated into each main file keyword before searching for coincidence. If this facility is not required the mask should be -1 (all ones).

7. STOPS

1.3+ (1.3+ 6 62) Loop stop if the first word in the first section used of an amendment tape is not negative, i.e. not a keyword.

2.3+ (0 0 77 in 2.3) 77 stop to change amendment tape. At this point, the amendment tape will be rewinding. When this has finished a new tape should be loaded. If it is desired to remove the tape from the top spool to be rewound later, the rewinding can be suppressed by the following sequence:-

```
X  49+.2
0
```

If a new mechanism (say *C*) is to be used to hold the amendment tape, the following manual orders should be obeyed:-

```
(64C) 1 40
2.3+  0 60
```

2.7+ (0 0 77 in 2.7) 77 stop to change main file tapes. At this point both the B.F. and C.F. tapes will be rewinding. The rewinding may be suppressed by the sequence:-

```
X  47+.4+ - 47+.5
0
0
```

If further mechanisms are available and it is desired to use them for the next two tapes, the following manual orders can be obeyed:-

```
(64A) 2 40
(64B) 3 40
2.7+  0 60
```

where *A* and *B* are the mechanism numbers for the new C.F. and B.F. tapes respectively.

3.4 The main link at the end of the updating process is obeyed in 3.4, and may be a stop order.

8. NEW FACILITIES

R 761 can take advantage of the new facilities in Pegasus 2 as follows:-

8.1 Quick Access Storage

With B0-15 of the drum replaced by 8-word delay lines, a suitable arrangement of buffers might be

B 0-11	Output buffer
B12-15	Input buffer

Note that in this special case the output buffer is specified as being only 12 blocks long, which is less than the usual minimum. Note also that in fact it will only use B1 to 8, leaving B0, 9, 10, 11 available for other purposes. This might not be the best arrangement if the maximum length of main file items were more than about twelve blocks, as too much of the first 128 blocks of store might be used up. Each case would have to be considered on its merits.

8.2 Two Tape Controls

Some modification to R 761 would be required as there would be 8 buffer blocks available, and it would be possible to segregate further the B.F. and C.F. information. The routine could be made much faster in this case.

8.3 More than Four Tape Mechanisms

Tape mechanism numbers from 0 to 4 are provided for.

London Computer Centre,
68, Newman Street,
LONDON, W.1.

Issue 2
24th August, 1960.
J.F.D.

APPENDIX

The following sections give details of the use of S1 and S2 under certain special circumstances.

A1 Amendment Subroutine (S1)

A1.1 Use of the Buffer Store

The sign of X4 is used to indicate whether or not the tape buffer is free for use by S1. It will be negative if the buffer may be used. If it is positive and S1 has to use the buffer there are two possible courses of action.

1. S1 can ensure that the buffer is left unchanged by storing it and restoring it. This may be the simplest course if S1 only needs two blocks (e.g. to write a 16-word section).

2. S1 may dump the tape buffer in B28+ - 31+ of R 761 and make X4 negative to indicate that this has been done. This is the recommended course of action, as R 761 normally has to dump the buffer there anyway (if it is not free), and will thus not have to do so if S1 has done it already.

   The sequence of orders required for dealing with the situation where it becomes necessary for S1 to use the buffer is:-

```
      4  63     Jump if buffer is free
 0    U  76
28    U  73     + partial cue
 1    U  76
29    U  73     + partial cue        Dump buffer in
 2    U  76                          B28+ to 31+
30    U  73     + partial cue
 3    U  76
31    U  73     + partial cue
32    4  06     Make X4 negative
```

   The 73 orders are to be tagged by the appropriate partial cues. It is recommended that the orders are written out thus rather than as a loop, as a loop would take one drum revolution per block, whereas the 73 orders above are optimised.

A1.2 Special Treatment of Frequently Repeated Keywords

Where repeated keywords are frequent, it may be advantageous to use a mode of working which does not involve returning to R 761 for each separate amendment.

The amendments are read from the amendment tape in batches. A batch will consist of several tape sections, the number being controlled by a preset parameter. Thus at any one time there will be a part of a batch, i.e. a number of amendment items, in the Main Store. S1 may therefore extract the keyword of the next amendment and test to see if it is the same as the keyword of the current main file item. If not, it will return to R 761 for the main file item to be written away, but if so it can process the amendment immediately without using R 761.

If this is done it is necessary for S1 to set X4 as before, and also to set in 5_m the Main Store address of the first amendment not used. Return is then made to R 761 by cue 03. This mode of return could be used when only one amendment has been processed, but there is no advantage in this.

It is necessary that S1 should not attempt to use amendments which have not been read from the tape. To ensure this the part of the Main Store allocated for amendments (called the *amendment buffer*) is divided into two zones, known as the *normal zone* and the *danger zone*. The sizes and positions of these zones are determined by preset parameters. The address of the first word of the danger zone is termed the danger address. When S1 has dealt with an amendment and is about to inspect the next keyword for identity with the current keyword, it must first test the address from which the next keyword is coming aginst the danger address; if it is greater than or equal to it, S1 should exit immediately to R 761 via cue 03, with the address of the next keyword in 5_m.

In these circumstances R 761 will refresh the amendment buffer from tape (by moving what remains in the danger zone to the top of the normal zone, then filling the remainder from tape). It will then, as usual, test for repeated keyword, and return immediately to S1 if the test succeeds (with the address in 5_m adjusted, as the item has been moved).

A1.3 Summary Operation after a Series of Repeated Keywords

If certain operations on the main file cannot be carried out until all the relevant amendment items have been processed, it is essential for S1 to identify its own amendments. The procedure when it is at the end of an amendment is then:-

1. Test to see whether next keyword is the same as the present one.
2. If not, perform summary operation and exit to R 761 via cue 03.
3. If so, test to see whether address is at or beyond the danger address.
4. If not, process repeated amendment.
5. If so, exit to R 761 via cue 03.

A1.4 Writing the C.F. item over the Amendment

S1 should not normally leave the C.F. item in place of the amendment item, because the amendment buffer is liable to be disturbed by R 761 before the C.F. item is copied to magnetic tape. If, however, it is required to write the C.F. item over the amendment the following procedure can be applied:

S1 can establish whether the amendment buffer will be disturbed by considering where the next amendment item begins: it can compute this address from the contents of X5 (on entry to S1). If this address is less than the amendment buffer danger address (P.P.10), and if the C.F. item is no longer than the amendment item, then S1 may write the C.F. item in place of the amendment item. Otherwise S1 must write the C.F. item to some other address.

A2 Error and Insertion Subroutine (S2)

A2.1 Use of the Buffer Store

No provision has been made for the restoration of the tape buffer on return from S2 to R 761 by cue 02. However, on entry to S2, the sign of X4 still indicates whether or not the buffer is in use. Thus if X4 is negative, S2 may use the buffer freely; if it is positive and S2 needs to use the buffer, it must make arrangements to restore the buffer before obeying cue 02 to R 761.

However, if the second return is to be used (new item, cue 04), the situation is as with S1. X4 will be negative if the buffer is free and positive if not; in the latter case S2 should dump the buffer in $B28^+$ to 31^+ of R 761 and mark X4 negative before using the buffer. In any case S2 should ensure that the sign of X4 is preserved (it may be changed from positive to negative) for R 761. For instance if S2 decides

to treat the amendment as a new item and to do no further operations (i.e. it does not need to use the tape buffer), a possible sequence of orders might be:-

	32	4 05	Preserve sign of X4
	5	4 01	Place modifier and counter for new item in X4
	5	6 00	(A, n) for current amendment to X6
	(25)	6 52	$(n, 0)$ to X6
	6	5 01	$(A + n, n)$ to X5 i.e. address of next amendment
		5 03	Subtract danger address (P.P.10)
←		5 62	
	+0		+ cue 04 to R 761

The 5 62 order should lead to a sequence of orders moving the new item out of the amendment buffer, and changing the address in 4_m accordingly.

A2.2 Leaving the Amendment Item in Situ

A new item cannot in general be left "in situ" in the amendment buffer as the buffer is liable to be disturbed by R 761 before the new item is copied out. S2 can establish whether the amendment buffer will be disturbed by considering where the next amendment item begins. It can compute this address from the contents of X5 (on entry to S2). If this address is greater than or equal to the amendment buffer danger address, S2 will have to move the current amendment, i.e. the new main file item, elsewhere and change the address in 4_m accordingly. Otherwise it can safely leave the new item in the amendment buffer, and re-enter R 761 via cue 04 with 4_m containing an address within the amendment buffer.

It would, of course, be sufficient for S2 to move the new item out of the amendment buffer in all cases, but this may lead to a significant loss of time if there is a large number of new items to be inserted.

R 2900

File as 900

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
18.3.57.

COMPARE TAPES

A programme which will compare two tapes placed one in each Tape Reader and stop if they are not identical.

Name: COMPARE TAPES.

Store: 2 blocks.

Uses: The whole Computing Store.

Entry: E 0+ at the end of the tape.

Speed of Comparison: About 85 characters per second.

METHOD OF USE

1. Place the tape of R 2900 in the main Tape Reader. START and RUN.

(This tape has no T-sequence on it and hence the programme will normally be read into Blocks 2 and 3. If it is required to preserve the contents of these blocks the programme can be stored in any other pair of consecutive blocks, say *B* and *B*+1, by setting T *B*.0 on the handswitches before operating the START key. The Relativiser does not need to be set since there is a B warning-character on the tape.)

2. When the 77 stop on reading the E at the end of the R 2900 tape is reached, insert the tapes to be compared one in each Tape Reader and operate the RUN key. Providing both tapes are initially set on a leader of blank tape they need not be lined up exactly. However, if the comparison is to be started on any non-blank character, then the tapes must be set on exactly corresponding characters.

3. R 2900 reads in 56 characters from each Tape Reader (ignoring only a blank tape leader at the head of each tape) and then compares them. If the comparison is satisfactory the process is repeated; but if there is any discrepancy a loop stop will occur. This will be in 0.7 unless the error is in the very first non-blank character. In this special case the stop will be in 0.3+.

4. If a loop stop in 0.7 occurs at least one discrepancy between the tapes within the last 5.6 inches read is indicated. Having marked the tapes at this point, it may be required to re-enter the Compare Tapes programme to check the remainder of the tapes. This can be achieved by a manual E 2.0 or, in general, E *B*.0 where *B* is the address of the first block of R 2900.

5. The comparison process is continued until one of the tapes runs out of the Tape Reader causing an Input Busy stop. Unless the remainder of the other tape is blank, the last few inches should be compared visually.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON. W.1.

Issue 1
18th March, 1957.
C.R.M. E.J.D.

R 2901

File as 901

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
14.3.57.

CLEAR STORE

This routine transfers zero to every Main Store location from B0.0 to B511.5 inclusive. The date and serial number are preserved in B511.7 and 511.6.

Name: CLEAR STORE

Store: B0

Uses: The entire Computing Store and the Main Store from B0.0 to B511.5 inclusive.

Entry: By J0.0 at the end of the tape.

Time: 1.6 seconds.

METHOD OF USE

1. Place CLEAR STORE tape in the main Tape Reader. START and RUN.

2. As soon as the CLEAR STORE tape has been read it may be replaced in the main Tape Reader by the next tape required.

3. R 2901 clears the Main Store (except the date and serial number) and then comes to a 77 stop in 0.6.

4. On reaching this stop the RUN key may be operated in order to read the new tape placed in the main Tape Reader. There is no need to operate the START key because R 2901 arranges to enter the START sequence of Initial Orders at this stage.

Ferranti Ltd.
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
14th March, 1957.
D.M.

R 2902

File as 902

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
20.5.57

IDENTIFICATION

This routine is designed to assist the programmer in identifying the origin of words in the Store. It is especially useful in identifying the Computing Store when a new programme is being developed. The routine writes the integers 0 to 4093 into Main Store locations 0 to 4093. The date and serial number are preserved in B 511.6 and 511.7.

Name: IDENTIFICATION

Store: 3 blocks.

Uses: The entire Computing Store and the Main Store from B 0.0 to B 511.5 inclusive.

Entry: By J 1+ at the end of the tape.

Time: 3 seconds.

METHOD OF USE

1. Place IDENTIFICATION tape in the main Tape Reader. START and RUN.

2. As soon as the IDENTIFICATION tape has been read it may be replaced in the main Tape Reader by the next tape required.

3. R 2902 writes the integer +0 into location 0, +1 into location 1 and so on through the store, ending with + 4093 in location 4093 (511.5). It then comes to a 77 stop in U 1.0.

4. On reaching this stop the RUN key may be operated to read the new tape placed in the main Tape Reader. There is no need to operate the START key because R 2902 arranges to enter the START sequence of Initial Orders at this stage.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
20th May, 1957
D.P. D.M.

File as 903

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
21.5.57.

FLOATING POINT PRINT (NON-ASSEMBLY)

An independent programme to punch out floating point numbers from the Main Store in a manner analogous to the F printing of the Initial Orders. Primarily intended for the examination of intermediate results when developing floating point programmes.

Name: INDEPENDENT FLOATING POINT PRINT

Store: 10 blocks.

Uses: The whole Computing Store except U 4 and U 5.4 to 5.7.

Entry: By an E- or J-sequence to 0+.2 to examine the handswitches or to 0+.0 to read addresses from tape. The programme tape is terminated by E 0+.2.

1. METHOD OF USE

1.1 Insert the programme tape of R 2903 in the main Tape Reader when required.

(This tape has T 500.0 at its head. If it is not convenient to store the programme in Blocks 500 to 509, the Transfer Address should be set as required and the tape inserted after this T-sequence.)

1.2 When the programme has been read in an E stop occurs on the sequence E 0+.2. Floating point numbers can now be printed under the control of either the handswitches or a steering tape.

2. MANUAL CONTROL

2.1 Set the address of the first number required on handswitches 1 to 13 in the same way as for a manual warning-character sequence and RUN. The setting of the five least significant handswitches is irrelevant except in the special case described in paragraph 2.4.

2.2 The floating point number in the specified address will then be punched out in the form described in Section 4 below, preceded by its address if H0 = 0. There is an optional stop in 0.6 after each number has been punched. This should be suppressed if a sequence of numbers from consecutive locations is required. The process continues until the handswitches are changed when it will start again at the new reading.

2.3 The address of the number currently being punched is available for monitoring in U 5.3_m.

2.4 If it is required to print under manual control from B 0.0, it is necessary to set something other than zero on the five least significant handswitches, otherwise the programme will try to read tape.

3. TAPE CONTROL

3.1 Clear all the handswitches with the exception of H0 if address printing is to be suppressed. Place the steering tape in the main Tape Reader and RUN.

3.2 The steering tape should be punched as one or more J 0+ sequences, each being followed by a single address or by two addresses separated by a minus sign.

e.g. J 0+ CR LF
16.0 - 17.7 CR LF

or J 0+ CR LF
143.2 CR LF

3.3 R 2903 will print out floating point numbers from the specified location or sequence of locations and then return control to the Initial Orders to read more tape.

4. FORM OF PUNCHING

4.1 If H0 = 0 each number is punched as:-

CR LF Address, Sp, Argument, Sp, Decimal exponent.

If H0 = 1 the form is:-

CR LF Argument, Sp, Decimal exponent.

4.2 The argument is printed as a signed number with nine significant figures, the most significant of which precedes the decimal point. If more or less than eight figures are required after the point the order in B 7+.5 of the programme (⑧ 1 40) must be replaced by ⑦ 1 40, where ν is the required number of decimal places.

4.3 The exponent is printed as a signed one or two digit integer.

4.4 Zero is printed as +0 without an exponent.

4.5 There is an extra LF between blocks.

4.6 If the argument of any number does not lie between ¼ and ½ an asterisk (*) will be printed instead of the number. This indicates that the number is not in standard floating point form.

5. FORM OF NUMBERS

5.1 This programme requires numbers to be stored in standard floating point form. i.e. The binary exponent, a, should be held as $256 + a$ in the nine least significant binary digits of the word, and must satisfy

$$-256 \leqslant a \leqslant 255$$

and the argument, A, should be held as a normalized fraction in the thirty most significant bits including the sign digit.

5.2 If n, the number of binary digits allocated to holding the binary exponent, is to have some value other than 9, the numbers in 3+.0 and 3+.1 (+256 and +511) must be replaced by $+2^{n-1}$ and $+2^{n} - 1$ respectively.

5.3 It should be noted, however, that this programme is not very suitable for printing decimal exponents greater than 100, which can arise if $n \geqslant 10$. This is because only two digits are allowed for in the decimal exponent. Thus an exponent

of +102 will be printed as ++2 (since + = 10). This could lead to serious confusion in certain cases. For example Sp = 14 and ϕ = 16, causing exponents of +142 and +162 to be printed as + 2 and +2 respectively.

Author: Mr. H.P. Goodman of the De Havilland Aircraft Company.

R 2904

File as 904

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
20.5.57.

CLEAR MAGNETIC TAPE

This routine transfers zeros to specified sections of magnetic tapes. It may be used to ensure that required sections on a new tape will not cause checksum failures when read by a subsequent programme. When developing programmes it may be useful to clear tape sections so that all entries on such sections may be attributed to the programme under test.

Name: CLEAR TAPE

Store: 3 blocks.

Uses: The entire Computing Store.

Entry: By JO+ at the end of a parameter tape.

Time: Approximately 44 milliseconds per 16 word section
or 56 milliseconds per 32 word section.
plus search time for the required section on each tape.

1. METHOD OF USE

1.1 First prepare a parameter tape as described in section 2 below.

1.2 Insert R 2904 programme tape in the main tape reader.

1.3 START and RUN to read R 2904.

(The tape has T 508.0 at its head. If it is required to preserve blocks 508 to 510, R 2904 must be stored elsewhere. If the tape is inserted after the T 508.0, R 2904 will go into the next available block in the store, as specified by the Transfer Address in U 5.7.)

1.4 There is a Z stop near the end of R 2904. On reaching this the parameter tape should be placed in the main tape reader.

1.5 Operate the RUN key to read the parameter tape.

1.6 After clearing the required tapes R 2904 will reach an optional stop in 1.4. On operating the RUN key (or if the optional stop is suppressed) the START sequence of Initial Orders will be entered.

2. THE PARAMETER TAPE

2.1 The parameter tape should contain up to 8 integers, specifying the section numbers which are to be cleared. No transfer address need be punched.

2.2 The first pair of integers refers to tape mechanism 0, the second pair to mechanism 1, the third to mechanism 2 and the last to mechanism 3.

2.3 The first integer of each pair specifies the first tape section to be cleared, the second specifies the last section to be cleared. All intermediate sections will be cleared.

2.4 If the first integer of a pair is negative the corresponding mechanism will not be operated upon.

2.5 The parameter tape must end with the warning-character sequence J0+.

2.6 If mechanisms 2 and 3 are not required, only the first four integers need be punched on the parameter tape. The last four will be left as -1 by an optional parameter list.

2.7 The parameter tape is read by the Initial Orders and the integers on it must be punched in standard I.O. form. It is convenient in practice to terminate the first integer of a pair by Sp and the second by CR LF.

2.8 D and N-sequences may be punched on the parameter tape if required.

3. OPTIONAL PARAMETERS

3.1 There is an optional parameter list incorporated in R 2904 which has the same effect as a parameter tape punched as shown on the left below:

	Clear Sections	Mechanism
CR LF		
+1 Sp + 200 CR LF	1 to 200	0
+1 Sp + 200 CR LF	1 to 200	1
-1 Sp -1 CR LF	Do not clear	2
-1 Sp -1 CR LF	Do not clear	3
λ Jϕ 0+ CR LF	Enter R 2904	

3.2 If the optional parameter list is to be used the RUN key should be operated on reaching the Z stop near the end of R 2904.

4. TAPE FAILURES

4.1 If a tape order fails it will be repeated twice.

4.2 If a tape order fails three times there will be a 77 stop in 1.6. The tape instruction which has failed may be monitored in X4. The tape instruction may be repeated by operating the RUN key.

Ferranti Ltd,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
20th May, 1957.
D.M.

R 2905

File as 905

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
12.11.57

TESTAID BREAK-POINT

A routine to aid programme development by facilitating the printing of intermediate results at specified 'break-points' in a programme.

The routine should be especially useful for development by remote control, when the programmer is not available to operate the computer himself.

Name: TESTAID D

Store: 10 blocks, including 5 blocks of working space. The first block must not be stored beyond block 127. The other part of the programme may occupy any 9 consecutive blocks in the Store.

Uses: The entire Computing Store, B0, and 5 blocks of working space.

Entries: E1 (*G*.0) Set break-point.
E2 (*G*.4) Re-enter programme.

1. INTRODUCTION

1.1 Break-point

A break-point is a point at which it is required to interrupt the running of a programme and to print the results obtained at that stage. Any order pair in a programme may be used as a break-point.

1.2 Setting a Break-point

The address of the desired break-point word is supplied by a steering tape and Testaid is entered at a first entry point E1. This stores the break-point word and replaces it by a cue to itself. It then returns to Initial Orders Input.

1.3 Initial Entry to the Programme

After setting the first break-point the programme may be entered in the usual way by a warning character J on the steering tape.

1.4 Printing at a Break-point

The programme will run normally until it reaches the break-point. Testaid then stores the whole Computing Store except U0, and replaces the Testaid cue by the original contents of the break-point word. Control is returned to Initial Orders Input so that F, I and P sequences may be used to print the required information from the Main Store or the Computing Store.

1.5 Next Break-point

If a further break-point is required it must be set, as described in 1.2, before re-entry to the programme. Break-points must be specified in the order in which they occur and the programme normally runs from one break-point to the next.

1.5 Re-entry to the Programme

Testaid entry E2 may be used to restore the Computing Store and re-enter the programme after a break-point. The steering tape must specify the block to be brought into U0 and the Computing Store address at which the programme is to be entered.

2. METHOD OF USE

2.1 First prepare a steering tape as described in section 3.

2.2 The programme under development and the test data should be read by the Initial Orders as usual, but the programme should not be entered. See also section 4.5.

2.3 Insert Testaid in the main tape reader. START and RUN to read it into blocks 1 and 500 to 508.

(If the programme uses any of these blocks Testaid must be stored elsewhere. It may be stored in block G ($G \leqslant 127$) and blocks H to $H + 8$ by inserting a tape reading

T G.0 - H.0
Z

Two addresses must always be punched on this tape. START and RUN to read it; insert Testaid after the sequence T 1.0 - 508.0 at its head; RUN to read it).
T 1.0 — 500.0

2.4 Insert the steering tape in the main tape reader. RUN to read it in (or START and RUN if preferred).

3. THE STEERING TAPE

3.1 Setting a Break-point

A break-point may be set in $B.P$ by the sequence

J G.0 - $B.P$ (Normally G = 1)

3.2 Printing

F, I and P sequences are used for printing.

The Computing Store is stored as follows:

Accumulators	in	B 0	
U1	in	B H + 1	(Normally B 501
U2	in	B H + 2	" 502
U3	in	B H + 3	" 503
U4	in	B H + 4	" 504
U5	in	B H + 5	" 505)

3.3 Re-entry to the Programme

To bring block *A* into U0, restore the rest of the Computing Store and jump to address *C.D*, the steering tape should read

```
J G.4                          (Normally G = 1)
A - C.D
```

If required, *D* may be followed by + to indicate a jump to a b-order. Normally *C.D* will be the position in the Computing Store formerly occupied by the break-point cue. *A* will be the block containing the break-point only if the break-point was in U0.

3.4 A Typical Steering Tape

The steering tape below assumes that Testaid is stored in B1 and 500 to 508.

Tape	Notes
J 1.0 - 5.3	Set break-point in 5.3.
J 2.0	Enter programme.
N 5.3	Name of break-point.
F 0-7	Print accumulators as fractions.
P 0.3 - 0.3+	Print X3 as an order pair.
I 504.0 - 505.7	Print U4 and 5 as integers.
J 1.0 - 8.7	Set new break-point in 8.7.
J 1.4 4 - 1.3	Restore B4 to U0 and jump to U 1.3.
N 8.7	Name of break-point.
I 6	Print X6 as an integer.
P 501.0 - 501.7+	Print U1 as order pairs.
F 350.0 - 350.7	Print block 350 as fractions.
J 1.4 8 - 0.7	Restore B8 to U0 and jump to U 0.7 to complete programme.

Punching conventions are the same as for the Initial Orders. In particular, name sequences must be terminated by blank tape and addresses (or pairs of addresses) by CR LF.

Relative addresses are not permitted.

4. PITFALLS

Testaid can be very helpful provided that it is used with care. Special attention should be paid to the following limitations:-

4.1 U0 is replenished from the Main Store after each break-point. The programme will fail if it uses U0 as working space or changes orders in U0 so that they differ from the original in the Main Store.

4.2 B0 is used by Testaid and, therefore, a programme will fail if information left in B0 is required after a break-point.

4.3 Testaid break-points are usually set in the Main Store. If the required break-point is already in the Computing Store (other than in U0) the break-point must be set at the appropriate address in blocks *H* + 1 to *H* + 5.

4.4 If the main programme branches, it is only possible to put a break-point into one of the branches. If the programme follows a different course this break-point will not be reached and the steering tape will never be read to set the succeeding break-points. See also paragraph 5.4.

4.5 If data is read from the main tape reader after entry to the programme, the data tape and the steering tape must be amalgamated. The items on this tape must be in the chronological order dictated by the requirements of the programme.

4.6 The restoring programme restores the original break-point order to its Main Store address. This could cause a fault if that part of the programme has been overwritten by other information.

4.7 The restoring programme also restores the original break-point order in the Computing Store. It does this by examining the appropriate position in U1 to U5 inclusive. If one of these happens to contain a number equal to the order pair

```
A [0] 72
0.1+ 0 60
```

this number would be overwritten as well as the break-point cue. This unlikely possibility need only be contemplated when other sources of error have been exhausted.

5. ADDITIONAL FACILITIES

5.1 If the contents of U0 are not needed on re-entry after a break-point the re-entry sequence may be punched

```
J G.4                    (Normally G = 1)
C.D
```

5.2 An alternative method of re-entry is to re-start from the beginning of the programme by means of a warning character J. This will fail if data or programme is overwritten in the Main Store.

5.3 If the same break-point is required again in a loop of programme, the complete break-point sequence must be repeated on the steering tape. If there are two or more break-points in the loop there are no additional limitations. If there is only one break-point and if it is in U0 the break-point cue will be brought down on re-entry and the steering tape must not jump to this. If the break-point is in U1 to 5 the original break-point order will be in the Computing Store, unless the rule given in 4.3 is applicable.

5.4 It is possible to set a break-point in *B.P* by the sequence

```
T  B.P
G  [0] 72                (Normally G = 1)
0.1+ 0 60
```

The original contents of *B.P.* will not be restored when Testaid is entered, and it is not usually possible to re-enter the programme after such a break-point. This method may be used to set several break-points at once, thus partially overcoming the difficulty described in section 4.4.

5.5 The operating procedure for Testaid can be simplified by joining together the programme tape, data tape, Testaid and steering tape in that order. The operator then has only to START and RUN at the beginning of this tape.

Ferranti Ltd.
London Computer Centre,
21, Portland Place,
London, W.1

Issue 1
12th November, 1957
D.P. D.M.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
9.8.57

FAST BLOCK TRANSFER TRANSLATION

This programme translates a tape produced by depressing the punch on block transfers key. It is about twice as fast as the Initial Orders? warning-character. The block numbers are printed in lines across the full width of the teleprinter paper and separated by single spaces. The greater speed and economy on paper make it convenient to use this programme for translating long block transfer tapes.

Name: ??

Store: 2.3 blocks

Uses: The whole Computing Store.

Entry: By E 0+ at the end of the tape.

1. Method of Use

1.1 Prepare a block transfer tape in the normal way, i.e. run the programme under development with the punch on block transfers key depressed. Then tear off the output tape.

1.2 *Return the punch on block transfers key to its normal position.*

1.3 START and RUN to read R 2906. (This tape has T 508.0 at its head. If it is not convenient to store the programme in B508.0 to 510.2, the tape should be inserted after this T-sequence, the appropriate Transfer Address having been set in U5.7$_m$).

1.4 On the 77 stop (E 0+) at the end of R 2906, insert the block transfer output tape in the main tape reader and RUN.

2. Form of Output

2.1 The block numbers are printed in lines across the teleprinter page separated by single spaces. There is a carriage return and three line feeds at the end of each line.

2.2 Non-significant left-hand zeros in the block numbers are omitted.

2.3 An asterisk will be punched at any point where output other than block numbers appears on the original block transfer tape.

Ferranti Ltd.
London Computer Centre,
21, Portland Place,
LONDON, W.1

Issue 1
9th August, 1957
T.F.

R 2907

File as 907

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
1.5.58.

STORE USE

This routine prints out a summary of the contents of all the blocks of the Main Store.

Name: STORE USE

Store: B0

Uses: The entire Computing Store.

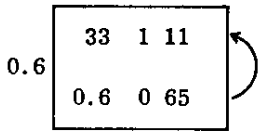
Entry: J0 at the end of the tape.

Time: 20 seconds including input of the routine.

Method of Use

In order to obtain the maximum amount of information from this routine it will usually be desirable to make use of R 2901 (Clear Store) or R 2902 (Identification) before reading in the programme to be investigated. When the programme has reached the point at which the investigation is to be made, place the R 2907 tape in the main tape reader, START and RUN.

After the summary of the Store has been printed there will be a loop stop in 0.6 :



One character is printed for each block of the Main Store from 0 to 511 inclusive. They are printed in order, 20 per line in 26 lines.

If all the words in a block are zero a dash is printed, otherwise the number of negative words will be counted and printed as a single digit. B0 is an exception as it has been overwritten by R 2907 and will always be printed as a point.

Author: Mr. J. Doughty of Babcock & Wilcox Ltd.

Ferranti Ltd.
London Computer Centre,
21, Portland Place,
London, W.1

Issue 1
1st May, 1958
J.D. M.M.

File as 910

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

PEGASUS 2 COMPARE PAPER TAPES

A programme which will compare two paper tapes placed one in each tape reader and stop if they are not identical.

Name: PEGASUS 2 COMPARE TAPES

Store: 1 block.

Uses: The whole Computing Store.

Entry: J 0+ at the end of the tape.

Speed of Comparison: About 300 characters per second.

Method of Use

1. Place the tape of R 7910 in the main tape reader. START and RUN.

(This tape has no T-sequence on it and hence the programme will normally be read into Block 2. If it is required to preserve the contents of this block the programme can be stored in any other block, say *B*, by setting T *B*.0 on the handswitches before operating the Start key.)

2. There will be a 77 stop in 0.0 after reading the J0+ at the end of the R 7910 tape; when this is reached, insert the tapes to be compared one in each tape reader and operate the Run key. If both tapes are initially set on a leader of blank tape they need not be lined up exactly. However, if the comparison is to be started on any non-blank character, the tapes must be set on exactly corresponding characters.

3. R 7910 reads in a character from each tape reader (ignoring only a blank tape leader at the head of each tape) and then compares them. If the comparison is satisfactory the process is repeated, but if there is a discrepancy a 77 stop will occur.

4. A 77 stop in 0.0 indicates that the last two characters read are different. Having marked the tapes at this point, it may be required to re-enter the Compare Tapes programme to check the remainder of the tapes. This can be achieved by operating the Run key. The routine can also be re-entered by a manual J 2.0 (or J *B*.0, see Section 1 above).

5. The comparison process is continued until one of the tapes runs out of the tape reader causing an Input Busy stop.

London Computer Centre,
68, Newman Street,
London, W.1.

Issue 1
9th October, 1962
D.M. M.J.B.

R 2915

File as 915

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
1.5.58.

INDEPENDENT DOUBLE-LENGTH FRACTION PRINT

An independent programme to punch out double-length fractions from the Main Store in a manner analagous to the F printing of the Initial Orders. It is primarily intended for the examination of intermediate results during the development of double-length programmes. The more significant half of each number must be stored in an even-numbered address.

Name: INDEPT. D.L. FRACTION PRINT

Store: 7 blocks.

Uses: The whole Computing Store.

Entry: By an E- or J-sequence to 0+.2 to examine the handswitches or to 0+.0 to read addresses from tape. The programme tape is terminated by E 0+.2.

1. Method of Use

1.1 Insert the programme tape of R 2915 in the main tape reader when required.

(This tape has T 500.0 at its head. If it is not convenient to store the programme in blocks 500 to 506, the Transfer Address should be set as required and the tape inserted after this T-sequence.)

1.2 When the programme has been read a 77 stop occurs on the sequence E 0+.2. Double-length fractions can now be printed under the control of either the handswitches or a steering tape.

2. Manual Control

2.1 Set the address (which must be even) of the first fraction required on handswitches 1 to 13 in the same way as for a manual warning character sequence, and RUN. The setting of the five least significant handswitches is irrelevant except in the special case described in paragraph 2.4.

2.2 The double-length fraction starting in the specified address is then punched out in the form described in section 4 below. There is an optional stop in 2.2 after each number has been punched. This should be suppressed if a sequence of numbers from consecutive locations is required. The process continues until the handswitches are changed when it will start again at the new reading.

2.3 The address of the number currently being punched is available for monitoring in $U5.3_m$.

2.4 If it is required to print under manual control from B 0.0, it is necessary to set something other than zero on the five least significant handswitches, otherwise the programme will try to read tape.

2.5 If an odd-numbered address is set on the handswitches there will be a loop stop in 0.6+ (0.6+ 3 61).

3. Tape Control

3.1 Clear all the handswitches (with the exception of H0 if address printing is to be suppressed). Place the steering tape in the main tape reader, and RUN.

3.2 The steering tape should be punched as one or more J 0+ sequences, each being followed by a single address or by two addresses separated by a minus sign. (All addresses must be even).

e.g. J 0+ CR LF
143.2 CR LF

or J 0+ CR LF
16.0 – 17.6 CR LF

3.3 R 2915 will print fractions from the specified location or sequence of locations and then return control to the Initial Orders to read more tape.

3.4 If the first tape address is odd there will be a loop stop in 0.6+ (0.6+ 3 61). If the second tape address is odd the computer will punch indefinitely.

4. Form of Punching

4.1 If H0 = 0 each number is punched as:- CR LF Address Sp Number

If H0 = 1 the form is:- CR LF Number

where Number = Sign Integer . Fraction

4.2 The number is signed and is printed with one figure before the point (0 or exceptionally 1) and 23 figures after the point. The fraction is unrounded.

4.3 -1.0 is correctly printed.

4.4 If the more significant part = -1.0 and the less significant part is < 0 the number is correctly printed.

4.5 There is an extra LF between blocks (i.e. after every four numbers).

4.6 If less than 23 figures are required after the point the order in B 4+ .7 of the programme (23 5 40) must be replaced by ν 5 40, where ν is the required number of decimal places.

5. Form of Numbers

This programme requires the double-length fractions to be stored with the more significant and less significant parts in consecutive locations in the Main Store, with the more significant parts in even numbered locations. They need not be in the standard double-length form, i.e. the less significant halves may be negative.

Author: Mr. H.P. Goodman of the De Havilland Aircraft Company.

Ferranti Ltd.
London Computer Centre,
21, Portland Place,
London, W.1

Issue 1
1st May, 1958
H.P.G. M.M.

R 2921

File as 921

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 2
28.3.58.

INITIAL ORDERS-BINARY TRANSLATION

This routine reads a tape which is punched in Initial Orders notation and produces a binary tape which is an exact functional equivalent of the original tape, complete with relative addresses and warning characters. This binary tape should be about half as long as the original tape and may be read in about one third of the time.

The routine is primarily designed for translating library tapes. It is normally less efficient than R 1033, Binary Punch, for master programmes which do not require relative addresses.

Name: BINARY TRANSLATION

Store: 17 blocks.

Uses: The entire Computing Store.

Entry: By E 0+ at the end of the tape.

Time: Approximately 3.2 seconds for each block translated.

1. Method of Use

1.1 Insert R 2921 in the main tape reader. START and RUN to read it into blocks 2 to 18.

(If it is required to preserve these blocks, R 2921 may be stored in blocks B to B+16 ($B \leqslant 111$) by previously setting the transfer address equal to B.0.)

1.2 On reaching a 77-stop in 4.0 (E 0+) at the end of R 2921, insert the tape to be translated in the main tape reader.

1.3 Clear the handswitches and tear off previous output tape.

1.4 RUN to start the translation.

2. Stops

The usual Initial Orders stops will be encountered if there are errors on the tape or if the tape reader misreads the tape being translated. In addition, the following stops may occur:-

Loop Stops

1.2+ if there is more than one - after a relative address in an a-order.

1.6+ if there is a - after a relative address in a b-order.

2.6 if OVR is set during input of an order or a number.

4.6 if a relative address in a b-order follows a relative modifier in the a-order.

Optional stops

3.0 on reading any warning character other than B, J, L, R or T. RUN if the warning character is correct.

3.0+ on reading a J sequence other than J 560.0+-*A*, entering an optional interlude at address *A*. RUN if the J sequence is correct.

77 stop

3.5 on reading any character other than CR after the blank tape following a warning character L (λ, Er and LF may be expected). Set the handswitches as described in section 3 below, and RUN to continue the translation.

3. Manual Control

The translation routine consults the handswitches after the 77 stop in 3.5.

Handswitch Setting H 0	1234.....	
0	0000.....	Handswitches clear. The warning character sequence, if any, is reproduced.
1	1000.....	Handswitches = -½. The warning character sequence is read but not reproduced.
1	0000.....	Handswitches = -1.0. The warning character sequence is not read. A blank tape and erase sequence is punched to terminate the tape; the 77 stop in 3.5 is then encountered again.

0.8 inches of blank tape is automatically run out after each name sequence and 0.6 inches at the end of each subroutine. Additional blank tape may be run out manually, if required, at the 77 stop in 3.5; RUN to continue the translation.

The routine may be re-entered by the equivalent to E 0+ (suppress optional punching), when it will punch 5.6 inches of blank tape before starting to translate. Alternatively E 0+.1+ will cause only 0.6 inches of blank tape to be punched.

4. Restrictions on items which can be correctly translated

4.1 L is always translated as a transfer to Assembly, and should, therefore, not be used for other purposes.

4.2 An interlude entered with J (but not E) must not have a fixed exit to Initial Orders. It should exit by obeying the link set in X1 either by Initial Orders or Binary Input.

4.3 Warning characters other than B, J, L, R and T should not occur between the name and the final L of a subroutine, unless the subroutine will never be rejected. This is because the translated equivalent consists of an 'Optional J' to Initial Orders (J 560.0+-522.0+ in binary), a reproduction if required of the warning character sequence, and A0 to return to Binary Input. If the whole sequence is being rejected the 'Optional J' will be ignored and the next item, which is in Initial Orders notation, will be read by Binary Input.

X sequences should therefore not be used in subroutines except while they are being tested. The same effect could be produced by T sequences, but normally the tape should be edited and corrections incorporated before a binary translation is made.

4.4 If X sequences do occur they will be translated in the same way as an N sequence: if the handswitches are clear the routine will copy until it finds two consecutive figure shifts. Blank tape must therefore be punched at the end of the X sequence but must not occur within it. If there are several X sequences they may be punched in one long string with blank tape at the end.

4.5 T or J warning characters followed by two addresses will not be correctly translated except in the case of J560.0+-*A*(+).

4.6 The warning character R must always be preceded by CR LF.

4.7 Certain unlikely combinations of relative addresses in orders have no counterpart in Binary Input notation: these will cause a loop stop in 1.2+, 1.6+ or 4.6. For example, the following combinations cannot be translated:-

Combination	Example
Relative modifier in *b*-order.	0 3 00 5+ - 00
Relative modifier in *a*-order AND relative address in *b*-order.	2+ - 00 1+ 1 72
Relative address followed by two or more minus signs.	0+ - -0 0. 0

Author: Mr. C.R. Merton of the National Research Development Corporation.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 2
28th March, 1958.
C.R.M. D.M.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 2
29.7.56.

END OF LIBRARY

A programme which appears at the end of the library tape and is entered if Assembly fails to find all it needs.

Name; END OF LIBRARY

Store: 4 blocks

Entry: by J on tape

1. This programme is entered if Assembly is not satisfied at the end of the library tape, e.g. if a non-existent subroutine has been called for or if a parameter-list has not been supplied. It indicates the missing tags by printing their functions and the corresponding routine numbers.

2. For example, a missing parameter-list is indicated by printing 04, and a missing subroutine by printing 08. If a programmer's subroutine is supplied but not called for then 01 is printed, showing that there has been no call for a cue. The following is a typical specimen of the output:-

END OF LIBRARY

ERROR

ROUTINE NUMBER	MISSING TAG FUNCTIONS
205	08
402	04
+03	01 04
5	02

3. Function 02 shows here that an unwanted parameter-list has been provided. Routine +03 means programmer's routine 1003. It should be noted that a missing routine is indicated by printing 08, never 28; in addition 02 will be printed if a parameter-list has been supplied.

4. After the printing there is a 77 stop, after which the Initial Orders are re-entered (with the T.A. reset). This enables a missing subroutine to be supplied by placing its tape in the second tape-reader and operating the Stop/Run key.

Ferranti Ltd.
London Computer Centre
21, Portland Place,
LONDON, W.1.

R 2923

File as 923

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 2
27.5.58.

BINARY - INITIAL ORDERS TRANSLATION

This routine reads a binary tape and produces a tape in Initial Orders notation, punched in standard form, which is an exact functional equivalent of the binary tape.

If a binary tape only of a programme is available, R 2923 provides the only convenient way of finding out what tags, optional interludes and relative addresses the tape contains. It is also a means of preparing print-outs of a programme in a standard form of punching.

Name: BINARY TO I.O. TRANSLATION ROUTINE

Store: 18 blocks.

Uses: The entire Computing Store.

Entry: By E 0+ at the end of the tape.

Exit: Must be made manually.

Time: Approximately 6½ seconds for each block translated.

1. Method of Use

1.1 Insert R 2923 in the main tape reader. START and RUN to read it into blocks 2 to 19.

(If it is required to preserve these blocks, R 2923 may be stored in blocks B to $B + 17$ ($B \leqslant 110$) by previously setting the transfer address equal to $B.0$)

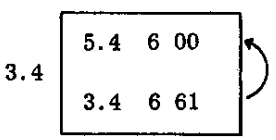
1.2 On reaching a 77 stop in 4.0 (E 0+) at the end of R 2923, insert the tape to be translated in the main tape reader and RUN.

2. Stops

Optional Stops:

0.3 on entry to R 2923.
0.3+ when R 2923 comes to the end of a binary section, or of an Initial Orders section of tape.

Loop Stop:



For checksum failure.
If this occurs the tape must be translated again.

3. Form of Translation

R 2923 merely copies those parts of the tape to be translated which are already in Initial Orders notation. The ordinary binary words are translated as order-pairs, so that some numbers and pseudo-orders will not be recognizable. The routine produces blocks of eight order-pairs, with .8 inches of blank tape between blocks. Provided there is enough information on the original tape in the form of binary B, L and T warning characters, the block divisions will come in the right places.

Every J translated from binary by this routine is followed by two addresses, i.e. is treated as referring to an optional interlude.

4. Re-entry

The routine may be re-entered by the equivalent to E 0+ (suppress optional punching).

Ferranti Ltd.
London Computer Centre,
21, Portland Place,
London, W.1

Issue 2
27th May, 1958
T.F.

R 7924

File as 924

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

7168 CONVERSION

A routine to convert a binary punched programme tape (punched from the 4096 word store) for use with the 7168 Initial Orders. It will alter the usual sequences calling in the Initial Orders, but will fail to detect sequences not listed in section 5. All parts of the tape in Initial Orders notation will be copied unchanged.

Names: 7168 CONVERSION (For use on 7168 word store)
4096 CONVERSION TO 7168 (For use on 4096 word store)

Store: 53 blocks

Uses: The entire Computing Store
Block 53+
One word for each alteration and one more for each section of binary punching, starting at 54+.0.
Plus the Main Store space normally occupied by the programme being converted.

Entries:

0+.0	E 512.0 (or 384.0)	Convert programme
0+.1	E 512.1 (or 384.1)	Punch programme already stored and converted
0+.2	E 512.2 (or 384.2)	Print record of alterations
0+.3	E 512.3 (or 384.3)	Re-entry to convert more programme, not resetting the printing marker.

Time: Approximately 1 block of binary tape every 3 seconds.
Approximately 25 characters per second for other sections of tape.

1. Restrictions

1.1 The programme is intended for use with binary punched tapes and will not normally work with binary translated tapes. This is because binary translated tapes contain many warning characters which will cause the programme to reach a loop stop.

1.2 The programme was designed for use on the 7168 word store. The 4096 version will only work if the programme being converted does not use blocks 384 to about 445, or some other selected section of about 60 blocks. Note that if R,2057 was used to punch the programme it may include Assembly working space and subroutine interludes.

2. Method of Use

2.1 Insert R 7924 in the main tape reader. START and RUN to read it into blocks 512 to 564. (384 to 436 on the 4096 store).

(If required the programme may be stored elsewhere by setting the appropriate transfer address and inserting the tape after the T 512.0 (or T 384.0) at its head.)

2.2 There is a 77 stop (E 0+) at the end of R 7924. On reaching this stop R 7924 should be removed and the programme to be converted placed in the main tape reader. Operate the RUN key to commence conversion.

2.3 During conversion there may be several optional stops and 77 stops. These may normally be inhibited, but see section 3 for a full description.

2.4 When the new programme tape has been punched, an indication of the changes made to the programme may be obtained by entering R 7924 at 0+.2, by means of the directive E 512.2 (or 384.2).

3. Amendment Stops

The following stops should normally be suppressed. Only users wishing to monitor the progress of a conversion need study the following.

In U 0.3 77 stop after amending the programme in the store. RUN to binary punch the amended programme. X - sequences may be read after this stop if required (see section 7).

1.3 Optional stop before making one of the standard amendments listed in section 5.1. RUN to make the amendment, jump to 1.5+ to omit it.

Monitor:	5_m	Address of Order
	6	Old order
	7	New order
	U 0	Block containing order.

2.4 Optional stop after binary punching a section of programme. RUN to read and convert more tape.

2.6 77 stop after printing a list of alterations made to a programme.

2.7 77 stop if a subroutine has apparently been found but the order to be changed is not correct. RUN to leave the order unchanged and cease treating that part of the programme as a subroutine.

Monitor:	2_m	Address of block wrongly recognised
	5_m	Address of order
	U 0	Block containing order
	U 1.4	Routine number in octal.

This stop may occur if only part of the subroutine has been binary punched, or it may be that the block 'recognised' happens to have the same sum as the selected block of the subroutine.

3.1 Optional stop before altering an order in a subroutine. RUN to make the alteration.

Monitor:	5_m	Address of order
	6	Old order pair
	7	New order pair
	U 0	Block containing order
	U 1.4	Routine number in octal

3.4 Optional stop after finding an order 33 X 00 not followed in the same block by 0 0 72 X or 10 0 72 X. RUN to leave the order unchanged. Jump to 2.4 to change it to 37 X 00.

Monitor:	5_m	Address of order
	6	Order 33 X 00 (In a-order position)
	7	Order 33 X 00 (In a-order position)
	U 0	Block containing order.

Note that this stop will also occur after an order 33 X 02, but the jump to 2.4 is not then applicable.

4. Error Stops

In U 0.7+ Loop stop if no figure shift after a warning character.

2.2+ Loop stop if there is no binary transfer address at the head of a section of binary tape. This may occur if the tape has been binary translated, or binary punched using R 2054. All binary punches formed by R 2057 or the directive A4 will be satisfactory in this respect.

3.4 Loop stop if there is a checksum failure in reading a binary tape.

1.6, 3.6+, 4.2+, 4.5 Loop stops on reading an impermissible binary directive. These can not occur on binary punched tapes, but are often present on binary translated tapes.

5. Alterations to Programmes

5.1 The conversion programme searches for the orders listed on the left below, and changes them to those shown on the right.

448 - 00	832 - 00	(a-order only)
449 - 00	833 - 00	(a-order only)
91 0 72 4	92 0 72 4	
92 0 72 4	93 0 72 4	
59 3 72 1	100 3 72 1	
33 X 00	33 X 00	(Only changed if followed by
	37 X 00	0 0 72 X or 10 0 72 X)

5.2 R 7924 also searches for the following subroutines and makes certain alterations to them:-

R 4, 11, 40, 42, 53, 102, 120, 650.

Note that if the programme used R 51, the rarely used subroutine ORDER PRINT, the order in 1+.1+ will not be changed. It will be necessary to insert an X-sequence to change 1+.1+ of R 51 to 13 4 40.

5.3 If the programme contains any other orders which require modification for the 7168 store, these should be made by X-sequences, fed in as described in section 7.

6. Printing a record of Alterations

6.1 On entry to the printing sequence (0+.2) by E 512.2 (or 384.2), each altered order will be listed as follows:

Address of order Old order New order

If the order has not been changed an asterisk will be printed in place of the new order.

6.2 If the subroutine has been altered, the address printed will be the address of its block 0+, and this will be followed by the routine number. Standard alterations to subroutines are sometimes made before the routine has been recognised by the conversion programme: the printing described in section 6.1 will then precede the subroutine number, but the addresses will be within the subroutine.

6.3 Binary tapes are often punched in several sections. At the end of each section the printing sequence will print the last address of the section followed by two Erases.

7. Manual Alterations

7.1 If it is required to make alterations other than those made by the conversion programme, a tape containing the appropriate X and T directives should be prepared before the programme is converted. This amendment tape should end with the directive E 512.1 (or 384.1) to enter R 7924 at 0+.1.

7.2 After the appropriate section of binary tape has been read and converted, there will be a 77 stop in 0.3. The addresses of the first and last words of the section to be punched will be displayed in U 5.6 and 5.7 respectively. The programme tape should be removed from the tape reader and replaced by the amendment tape. Suppress optional punching and START and RUN to read the amendment tape. On reaching the 77 stop (E 512.1 or 384.1) at the end of the amendment tape, replace the programme tape on the same character (which will usually be the beginning of a section of blank tape) and operate the RUN key.

7.3 Note that if the corrections thus inserted alter the length of the section of programme the new first and last addresses must be set in blocks 53+.6 and 53+.7 before re-entering at 0+.1.

8. Checking

Before using a programme converted for the 7168 word store it is advisable to clear the store and then run the programme with some test data for which the results are known.

London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
1st June, 1959.
D.M.

R 7925

File as 925

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

CONVERT PROGRAMME FOR 4096 STORE

A routine to convert a 7168 binary punched programme tape for use on the 4096 word store. This routine is the converse of R 7924: the user should first study the specification of R 7924 and then note the following changes.

Names: 7168 CONVERSION TO 4096 (For use on 7168 word store)
4096 CONVERSION (For use on 4096 word store)

Store: 62 blocks

Uses: As R 7924 except that the list of alterations is stored in B62+.0 onwards.

Entries: } As R 7924
Time: }

1. Restrictions

As R 7924 but add:

1.3 It is only possible to convert programmes which do not use locations above B511.7. Note that if R 7057 or 7059 was used to punch the programme it may include Assembly working space near B882 and 895: these two sections of binary tape must be omitted when the programme is converted.

2. Method of Use

As R 7924.

3. Amendment Stops

It is more often necessary to monitor the progress of a conversion using R 7925, due to the fact that an a-order 5.0 1 00 is equivalent to a modifier 833.0, used to enter the 7168 Initial Orders Fraction Print Routine, and may be wrongly changed to 449.0. This should normally be detected from the alteration printing and prevented, on a second run, by jumping to 1.5+ after the appropriate optional stop in U 1.3.

The amendment stops are the same as for R 7924 except for the following changes and additions

In U 2.2 Optional stop before changing FAST R5 to the original version of R5. RUN to make the alteration.

Monitor: 5_m Address of BO+ of R5
U0 BO+ of FAST R5
U1.4 +5 (Subroutine number)

3.4 Optional Stop after finding an order 37 X OF where F ≠ 0. RUN to leave the order unchanged. Jump to 2.4 to change it to 33 X OF.

Monitor:	5_m	Address of Order
	6	Order 37 X OF (As an a-order)
	7	Order 37 X OF (As an a-order)
	UO	Block containing order

4. Error Stops

As R 7924.

5. Alterations to Programmes

5.1 R 7925 searches for the orders listed on the left below and changes them to those shown on the right.

832 - 00	448 - 00	(a-order only)
833 - 00	449 - 00	(a-order only)
92 0 72 4	91 0 72 4	
93 0 72 4	92 0 72 4	
100 3 72 1	59 3 72 1	
37 X 00	33 X 00	
37 X 0F	37 X 0F	(F ≠ 0)

5.2 It also searches for the following subroutines and makes certain alterations to them:

R4, FAST R5, 11, 40, 42, 53, 120, 650.

Note that if the programme used R 51 or 58 these will not be recognised. It will be necessary to insert appropriate X-sequences to change their non-standard references to the Initial Orders. These X-sequences must have absolute addresses but should otherwise be as follows:

R 51	R 58
X 1+.1+	X 0+.4
7 4 40	56 1 72 4
	X 0+.5 - 0+.5+
	57 2 72 4
	2.6 7 00
	X 0+.6+ - 0+.7
	13 2 70 4
	55 0 72 4

5.3 If the programme contains any other orders which require modification for the 4096 store, these should be made by X-sequences, fed in as described in section 7 of R 7924.

6. Printing a Record of Alterations

As R 7924 except that the layout of the printing should read:

Address of Order	New Order (4096)	Old Order (7168)

7. Manual Alterations
8. Checking
} As R 7924.

Issue 1
12th April, 1960.
D.M.

R 7927

File as 927

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

BINARY TO MAGLIB TRANSLATION

This programme reads a binary translated subroutine tape and records the subroutines on 16-word magnetic tape in a form suitable for use with MAGLIB, the Magnetic Tape Library.

Name: MAGTRANS 2

Store: 10 blocks plus B0

Uses: U0 to 5; X1 to 7; B10+ onwards; W0 to 3 (space must be allowed from B10+.0 onwards to store the longest subroutine in Maglib form).

Entry: By E0+.1 at the end of the tape.

Time: Approximately 0.5 seconds per block translated.

1. Method of Use

1.1 Load a 16-word magnetic tape on the selected mechanism (see section 2.3).

1.2 Insert R 7927 in the main tape reader. START and RUN to read it into blocks 2 to 11 (and B0).

(If it is required to preserve block 2 onwards, R 7927 may be stored in blocks B to B+9 ($B \leqslant 116$) by previously setting the Transfer Address equal to B.0).

1.3 On reaching a 77 stop in 0.6 (Z-stop) near the end of R 7927, insert the tape to be translated in the second tape reader. If a parameter tape is required (section 2) it should be inserted in the main tape reader in place of the J0+.1 at the end of R 7927.

1.4 Ensure that handswitch 1 is up and RUN to start the translation.

1.5 There will be an optional stop in 0.3 whenever an L or J directive is encountered. On this stop, at the L ending a subroutine, the tape may be replaced, if desired, by another subroutine tape.

1.6 When the optional stop in 0.3 is encountered and there is no further tape to be read the translation must be completed by depressing handswitch 1 and operating the RUN key.

1.7 When the translation has been completed there will be an optional stop in 1.5: on going to RUN the magnetic tape will be rewound and there will be a 77 stop in 1.6.

2. Parameters

The operation of R 7927 is controlled by parameters set in B0. If desired some of these may be changed by means of a parameter tape fed in after R 7927. The parameter tape should normally be terminated by J0+.1.

2.1 Name

The name of the subroutine library is recorded in two locations and is printed out whenever Assembly reads the magnetic tape. The name set in B0 by R 7927 is

CR LF λ MAGLIB φ Sp 7301

The first 8 characters (CR LF λ MAGLI) are used by Assembly for identification and may not be altered. The last 7 characters are stored as 5-bit characters (register 17 code), reading from right to left, in B0.7, which contains

```
 0   6 04 6.          |  |  1 |  0 |  3 |  7 | Sp |  φ | B
                  =    |  |    |    |    |    |    |    |
59   4 00 2          0 0000000 110 000100 110 0111011 100 000000 010
```

The number of the tape could be changed to 7302 for example, by punching on the parameter tape:

```
X 7

 1   2 04 6.
```

2.2 Tape Reader

If it is desired to read the subroutine tape on the main tape reader the parameter tape should be terminated by E0+.1+ instead of J0+.1.

2.3 Tape Mechanism and Section

R 7927 records the start of the first subroutine in section 512 on tape mechanism 3, where Assembly expects to find the start of the library. This may be altered to section *A* and mechansim *m* by punching on the parameter tape

```
T4
A   - -- 0.
0   0 m0 1
```

If it is desired to set *A* = 0 the parameter tape must be terminated by J0+.0 so that the first three words on the magnetic tape are preserved.

2.4 Start of first subroutine

R 7927 always stores the library name (Maglib 7301) in W0.4 and 0.5 of the first section used. It stores the start of the subroutine in W0.7 and an L-directive in W0.6, where Assembly expects to find the start of the library.

If for some reason it is desired to store the L in W*l* ($0 \leqslant l \leqslant 15$) and the start of the subroutine in W*s* ($l+1 \leqslant s \leqslant 16$) this may be done by punching on the parameter tape.

```
T2
(s-1)  - -0 0.
0
+(s-l-1)

T5
l      - -0 0.
0
```

Note that if $l \leqslant 5$ the name of the library will be overwritten.

3. Restrictions

3.1 R 7927 will only accept binary translated subroutine tapes punched by R 7921 (or 2921).

3.2 Warning characters other than A0, B, J, L, N, R and T are not permitted. Only A0 and N may appear in non-binary form.

3.3 All library subroutines except R 600 and 730 should conform to these rules, but note that a version of R 52 has been issued which has had an Initial Orders L-directive erroneously punched after it: this will cause a loop stop in 4.3 but translation may be resumed by a manual jump to 4.4.

3.4 Complete programmes may not be translated by means of R 7927.

4. Stops

U 1.5	1	5 41 0.	Optional stop when translation complete. RUN to rewind the tape.
U 1.6	0 1.6	0 77 0 60	77 stop after rewinding the tape.
U 3.4+	3.4+	6 61	Loop stop on checksum failure in binary tape. If this occurs the tape must be translated again.
U 3.5	15	1 00 0.	Optional stop after L or J directive. RUN to read more tape or depress handswitch 1 and RUN to terminate the translation.
U 4.3	4.3	1 61	Loop stop on reading something other than CR LF or λN after an L-directive. Jump to 4.4 to ignore the impermissible characters. (This may occur with R 52, see section 3.3).

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

CHECK MAGNETIC TAPE

This subroutine checks the settings of the write and 16/32 word switches on a magnetic tape mechanism and leaves certain information about the tape in the Computing Store. If required it will also print this information.

Before running a magnetic tape programme it is vital to check that the write and 16/32 word switches are correctly set. It is not sufficient to rely on the computer operator, and R 930 should be used for this check.

On the 7168 store there is a version of R 930 in the Initial Orders.

Name: CHECK TAPE

Store: 6 blocks.

Uses: U0, 1, 2; $X6_m$, 7; B0;
W0, 1, 2, 3. (W = buffer block)

Cues:

01

```
0+  0 72
0.2 6 66
```

Check and print

02

```
0+  0 72
0.2 0 60
```

Check without printing

Time: In milliseconds:

	16-word	32-word
Cue 01	1200	1300
Cue 02	500	600

The times given above assume that, on entry, section 0 of the tape is under the reading head.

Link: Obeyed in 1.0 and left unaltered in X1.

1. Initial Settings

Before obeying a cue to R 930 the programme must set X6 as follows:

$$x_6 = m + 8i + t$$

where m = tape mechanism number

i = 1 if the write switch should be off (writing inhibited)
= 0 otherwise

t = 16 or 32, according to the setting of the 16/32 word switch.

The modifier and sign of X6 should normally be clear, but if it is required to ignore the setting of the write switch the sign bit should be 1 and i = 0.

2. Checking

Check Tape first checks the settings of the write and 16/32 word switches on the specified mechanism. If either is wrong there will be a 77 stop in 0.0; if the 16/32 word switch is wrong the OVR will also be set. On correcting the settings and operating the RUN key the tape will be checked again.

A tape read failure will occur if the setting of the 16/32 word switch is not consistent with the type of tape mounted. On correcting the fault and operating the REPEAT key the tape will be checked again.

3. Section 0 on Magnetic Tape

Check Tape assumes that the following information has been written in the first three words of section 0 on each magnetic tape.

0.0	$+L$		Length of tape in feet (nominal)
0.1	$+S$		Serial number of tape
0.2	A	---0.	Address of last section on the tape $\times 2^{-16}$
	0		(A = Number of sections - 1)

This information will not be recorded on a tape when it is first received at a Pegasus installation, but it should be written on section 0 before the tape is brought into use. All programmes using magnetic tape should be checked to ensure that they do not overwrite the first three words of section 0.

4. Printing

If entry to R 930 is by cue 01, and if the settings of the write and 16/32 switches are correct, the following information will be printed on a new line:

$m\ t/L/S/n =$

where m = mechanism number (1 digit)

t = 16 or 32: the setting of the 16/32 word switch

L = nominal Length of tape in feet (4 digits)

S = Serial number of tape (5 digits)

n = number of sections on the tape (5 digits)

$=$ indicates that the write switch is on

$\neq$ indicates that writing to the tape is inhibited

Check Tape printing occupies 22 character positions across the page, leaving room for the master programme to print further identifying information on the same line if required. The layout may be seen from the following example:

3 16/3000/00192/10816≠

5. Exit

On exit from Check Tape the contents of Section 0 will be available as follows:-

Section 0, block 0 in U2

1 in W1 (Buffer block 1)

{2 in W2, 3 in W3} if checking 32-word tape

The master programme may therefore easily transfer further identifying information to or from section 0 and print it if required.

6_C is not changed by R 930 and may afterwards be used by the master programme to determine m and t, but if the sign of X6 is 1 it is not possible to use i to determine the setting of the write switch.

A writing marker is left in X7 and U 1.1. This marker will be zero if writing is inhibited but non-zero otherwise.

6. Stops

U 0.0

0	0	77
20	7	13

77 stop if the 16/32 or write switches are incorrect

If OVR clear, $x_4 = ½$: write switch wrong
If OVR set, $x_4 = -1.0$: 16/32 switch wrong
If OVR set, $x_4 = -½$: both switches wrong

On correcting the fault and going to RUN the tape will be re-checked.

U 1.0

0	[2]	76
2.2	4	00

Tape failure stop if the 16/32 word switch is not consistent with the type of tape mounted. On correcting the fault and operating the REPEAT key the tape will be re-checked.

U 1.3

20	7	11
2	[3]	76

Tape failure stop, with writing inhibited OR Writing with OVR stop. These may occur if the first three words of section 0 do not contain the information specified in section 3.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

CHECK PSEUDO OFF-LINE

R 7931 is designed to be run after a pseudo off-line operation, on completion of or at a suitable point in the current computer programme. It reads registers 54 and 55, thus checking their parity, and prints their contents so that there is a printed record of the correct completion of the off-line operation. It also clears register 55 to ensure that the lockouts associated with the off-line operation are lifted.

If a pseudo off-line operation is stopped before it is completed, as described in section 11.8.2 of CS 303A and CS 333, R 7931 should be used both to check the operation and to produce a restart tape.

Name: CHECK P.O.L.
PRINT REGISTERS 54, 55

Store: 9 blocks

Uses: The whole Computing Store

Entry: By E0+.0 on the programme tape.

1. Method of Use

1.1 Place R 7931 in the main tape reader. START and RUN, preferably with H0 = 1 to suppress optional punching.

(There is a T 885.0 at the beginning of the tape. If it is required to store the programme in some other place in the store, the Transfer Address should be set as required and the tape inserted after the T-sequence.)

1.2 The computer will reach a 77-stop in 4.0 (E0+.0): RUN to enter the programme.

1.3 When R 7931 has checked the parity of registers 54 and 55, has cleared 55 and printed out the contents of 54 and of 55 before it was cleared, there will be a 77-stop in 0.6 on reading a Z from the programme tape.

1.4 A restart tape, if required, can be punched as follows. (It may be necessary to run in a steering tape as described in section 3.3; this should be run into the main tape reader by operating the Run key, after which R 7931 should be replaced in the tape reader.) On operating the Run key the remainder of R 7931 will be read and the restart tape will be punched out, preceded by a leader of blank tape and terminated by a blank tape and erase sequence.

1.5 There will be a 77-stop on reading the Z at the end of R 7931.

2. Form of Printing

The contents of registers 54 and 55 will appear on the print-out as two order pairs with the 16 most significant bits interpreted as a tape address in each case.

3. The Restart Tape

3.1 The restart tape is punched in Initial Orders notation so that a printed record may be obtained.

3.2 When the pseudo off-line operation is to be restarted, the restart tape should be placed in the main tape reader and the Start and Run keys operated. This restart programme will usually be read into B2 but may be read into any other block in the store by setting the appropriate Transfer Address. It is entered by E0+.2 at the end of the tape. When the off-line operation has been started, the computer will come to a 77-stop in 0.4.

3.3 The restart programme sets the contents of registers 54 and 55 to the values which they had when the off-line working was stopped, except that A'_s is usually changed. A'_s is the address of the next section of magnetic tape to be referred to, or, if tape is not being used, is the number of the next transfer. If the pseudo off-line transfers are being made to magnetic tape A'_s will not be altered, but otherwise it will be reduced by 5 so that a few lines or cards are repeated as a check. If it is desired to set back A'_s by some number, n, other than 5 (n may equal zero), the following steering tape should be punched:

$$\begin{matrix} +n \\ Z \end{matrix}$$

This should be inserted at the Z-stop before the restart tape is punched (section 1.4).

4. Parity Failure

If there has been a parity failure in register 54 or 55 during the pseudo off-line working, this will be detected by R 7931 which will stop in 0.1 or 0.3 on a computing store parity failure. If this happens the results of the pseudo off-line operation must be regarded as suspect and an engineer should be called if possible.

Issue 1
5th September, 1962.
M.J.B. D.M.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

CARD READ ERROR SUBROUTINE

R 932 should be entered when register 48 is found to be negative during card input. It will discover whether the fault is a check reading failure, a double punching or a "hard fault", and in which character the error has occurred. Certain information about the fault can be printed if required.

R 932 is an extended version of the error routine given in CS 303A and CS 333, Section 11.7.2.

Name: CARD READ ERROR S/R

Store: 6 blocks + 3 blocks if the standard printing is required for check read failures only + 2 further blocks if standard printing is required for double punching.

Uses: U0, 1, 2, 4, 5; B0; plus any space used by additional programmer's error routines.

Cues:
01 (0+.0)
02 a-order partial cue
03 b-order partial cue

Link: Normally set in X1 and obeyed in U2.3. Alternatively it may be preset as P.P.04 which is again obeyed in U2.3.

1. Mode of Operation

1.1 R 932 decides whether or not the fault indication in register 48 can be cleared; if it is not clearable (hard fault) a further fault routine is entered (see Section 4).

1.2 If register 48 can be cleared, the routine searches for check read failures and enters a check failure routine for each word which has failed (see Sections 2 and 6). After one or more check failures have been found, there will be a 77-stop in 2.2 before the link is obeyed.

1.3 If no check read failure is detected but the routine does detect a double punching, a routine is entered to deal with double punching (see Sections 3 and 7). If the standard double punching routine is used, return will be made to the 77-stop in 2.2 before the link is obeyed.

1.4 If R 932 fails to find either a check read failure or a double punching after successfully clearing register 48, a fault in the checking equipment is assumed and the part of the routine which deals with hard faults is entered.

2. Check Failure Printing

2.1 If a check read failure is detected R 932 will, if required, enter a print routine. After the words CHECK FAIL the following information will be printed: the

buffer address(es) of the word(s) containing the faulty character(s), the character number(s) in the word(s) and the corresponding character(s), in binary, from the not-equivalent buffer. For example:

```
CHECK FAIL
0.6  2 ≠101100
     5 ≠100001
1.7  1 ≠000011
```

2.2 As an alternative to the standard printing described above, the user may supply his own routine to deal with check read failures (see Section 6).

3. Double Punch Printing

3.1 If a double punching but no check read failure has occurred R 932 will, if required, enter the standard print routine to print information similar to that for a check read failure. The words DOUBLE PUNCH will be printed, followed by the buffer address(es) of the word(s) containing the double punched character(s), the character number(s) in the word(s) and the corresponding character(s), in binary, from the not-equivalent buffer.

3.2 As an alternative to the standard printing, the user may supply his own routine to deal with double punching (see Section 7).

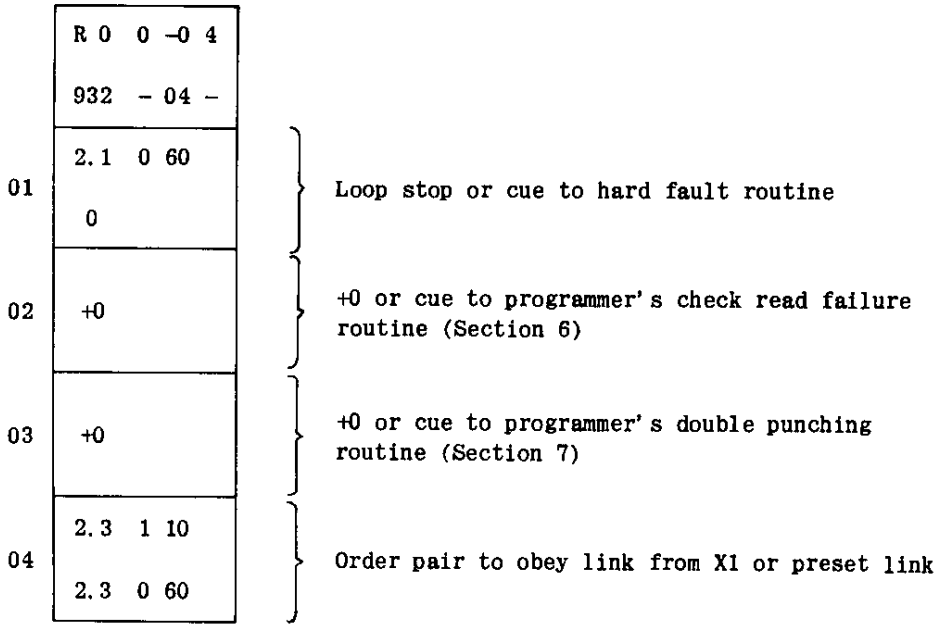
4. Action on Finding a Hard Fault

4.1 If a hard fault is present, R 932 will print HARD FAULT and come to a 77-stop in 2.0 before restoring the accumulators and obeying P.P.01. On operating the Run key a loop stop in 2.1 will be encountered if the optional value of P.P.01 is used.

4.2 P.P.01 may be set as the cue to a further part of the error routine supplied by the user. This piece of programme will normally be designed to store, in the Main Store, certain information which may be used to restart the card input programme after the fault has been rectified. When P.P.01 is obeyed, R 932 will not have disturbed the card input data buffer.

5. Preset Parameters

5.1 A parameter list of the following form may be supplied:-



5.2 If no parameter list is supplied by the user an optional list will set the parameters to the values shown above.

5.3 If P.P.02 or P.P.03 is set to +0 as shown, the standard R 932 printing will occur on finding a check read failure or double punching respectively. At the 77-stop in 2.2 (after printing) $x2 \neq 0$ after a check read failure but $x2 = 0$ after a double punching.

5.4 If no action is required on finding a check read failure and/or double punching, P.P.02 and/or P.P.03 may be set to the order pair:

```
2.2  0 60

 0
```

This will simply cause R 932 to come to a 77-stop in 2.2 before restoring the accumulators and obeying the link. After a check read failure $x3 \geqslant 0$ at the 77-stop but after a double punching $x3 < 0$.

6. Alternative Check Failure Routine

6.1 If the user wishes to supply a routine to deal with check reading failures it should have the following specification. It will be entered once for each **word** in the not-equivalent buffer indicating check read failure(s).

The cue should be set as P.P.02.

The routine may not use U4, 5; X2, 4; B0.

It should return to R 932 by jumping to U1.3. R 932 will then search for further check read failures. If U0, 1, 2 have been overwritten they should be restored by reading B0+, 1+, 2+ of R 932 respectively, using the a- and b-order partial cues.

6.2 On entry to the routine, the following information will be available:

$X2_m$, $X4_m$	Buffer address of word containing failure
X3	Faulty word from data buffer
X5, X6	Corresponding word from $\not\equiv$ buffer
U4	Block from $\not\equiv$ buffer
U5	Block from data buffer

The data buffer will not have been disturbed.

X4 will be positive on the first entry to the routine for a given card and negative on subsequent entries, if any, for that same card.

6.3 If it is required to enter the routine once only, i.e. to ignore all but the first check read failure in a card image, return may be made to R 932 by restoring B2+ to U2, if necessary, and jumping to the 77-stop in U2.2, after which the accumulators will be restored and the link obeyed. In this case the routine may use any locations except B0. Alternatively it need not return to R 932, in which case it should include an order to restore the accumulators from B0.

7. Alternative Double Punching Routine

7.1 If a routine to deal with double punching is supplied by the user, it should be written to the following specification. It will be entered once only for each **card** containing double punching.

The cue should be set as P.P.03.

The routine may **not** use B0.

It should return to R 932 by restoring B2+ to U2, if necessary, and jumping to the 77-stop in U2.2, after which the accumulators will be restored and the link obeyed. Alternatively it need not return to R 932, in which case it should include an order to restore the accumulators from B0.

7.2 On entry to the routine the following information will be available:

$X2_m$ Buffer address of faulty word. (If there are several double punched columns this will be the address of the last one to appear in the input buffer.)

X6 ≠ of faulty word

U4, U5 } Card image

The next card will have been read into the data buffer.

The ≠ buffer will be in B3+ and 4+ of R 932.

8. Stops

(i) 0.2

```
104 [4] 76
3+  [4] 73
```

Writing with overflow due to 73-order in 0.1+ if R 932 entered with overflow set

(ii) 2.0

```
0   0 77
0   [7] 72
```

77-stop on finding a hard fault

(iii) 2.1

```
2.1 0 60
0
```

Loop stop on going to RUN after stop in 2.0 if the optional value of P.P.01 is used

(iv) 2.2

```
0   0 77
0   [7] 72
```

77-stop before obeying the link

9. Notes

9.1 If a check read failure occurs, all double punchings on that card will be ignored.

9.2 If a double punching is detected, R 932 will not detect a check read failure which occurs in the same word of the buffer.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
20.5.57.

RANDOM NORMAL DEVIATES

A tape containing random normal deviates punched correct to two decimal places.

FORM OF PUNCHING

The deviates are punched in a form suitable for reading by R 113. They are all punched correct to two decimal places but the decimal point has been omitted. The sign has also been omitted in the case of positive numbers. The terminating character may be either Sp or CR LF.

SCALING

The deviates appear on the tape as integers, and would have to be divided by +100 in order to convert them to their true value. The +100 must be multiplied by some number not less than 4 in order to ensure that the scaled deviates are less than one.

If the preset parameter of R 113 were set as +800, for example, the random normal deviates would be read in with a scale factor of 2^{-3}.

Example:

Normal equivalent deviate	+1.64	−0.32
Number on Tape	164	−032
Number in Computer	+0.205	−0.04

SOURCE

The deviates were taken from 'Tracts for Computers' number XXV by Herman Wold, published by the Department of Statistics, University College, London.

Only the first and second thousands are at present available, but others are being prepared.

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
LONDON, W.1.

Issue 1
20th May, 1957.
D.M.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1.
16.8.56.

STANDARD ATMOSPHERE

A subroutine to calculate the values of the Standard Atmosphere.

Given $h.2^{-17}$ in X6,
where $0 \leqslant h \leqslant 119,141$ ft.,
This subroutine places:

ρ in X5,
$p.2^{-20}$ in X6,
$a.2^{-12}$ in X7,

where h is height above sea level in feet,
ρ is air density in slugs per ft.^{3},
p is air pressure in lbs. per ft.^{2},
a is the speed of sound in ft. per second.

(R 970 uses R 200 and R 220)

Name: I.C.A.M.

Store: 6 blocks for R 970 + 3 blocks for R 200 and R 220.

Uses: U 0, 1; X 1,5,6,7.
(If $h = 0$, U1 and X1 are not used.)

Cue: 01 (0 + .0)

Link: Obeyed in 0.4 and left undisturbed in X1 if $h = 0$
Obeyed in 1.1 if $0 < h \leqslant 36,093$ ft.
Obeyed in 1.5 if 36,093 ft. $< h \leqslant 119,141$ ft.

Time:

If $h = 0$	2 milliseconds.
If $0 < h \leqslant 36,093$ ft.	109 milliseconds.
If 36,093 ft $< h \leqslant 119,141$ ft.	53 milliseconds.

Error Stop: There is a loop stop in 1.5 if $h > 119,141$ ft.

Formulae

R 970 uses the following formulae to calculate ρ, p and a.

1. $h = 0$ ft.

$a = a_0 = 1117$ ft. per second
$p = p_0 = 2116.2$ lbs. per ft.^{2}
$\rho = \rho_0 = 0.002378$ slugs per ft.^{3}

2. $0 < h \leqslant 36{,}093$ ft.

$$\left.\begin{aligned} a &= a_0 (1 - kh)^{1/2} \\ p &= p_0 (1 - kh)^{5.256} \\ \rho &= \rho_0 (1 - kh)^{4.256} \end{aligned}\right\} \quad a_0,\ p_0,\ \rho_0 \text{ as for } h = 0$$

where $k = 0.000{,}006{,}8785$ ft.$^{-1}$

$(1 - kh)^{1/2}$ is formed by using the square root subroutine, R 200.

$(1 - kh)^{0.256}$ is formed by using the approximation:

$$(1 - kh)^{0.256} = 1 - 0.256\,(kh) - 0.095232\,(kh)^2 - 0.0553615\,(kh)^3 - 0.0379780\,(kh)^4$$

3. $36{,}093$ ft. $< h \leqslant 119{,}141$ ft.

$$a = a_T = 968.47$$

$$p = 472.64 \quad \exp\left(-\frac{1}{20762}\ (h - 36{,}093)\right)$$

$$\rho = 0.000{,}706{,}403 \quad \exp\left(-\frac{1}{20762}\ (h - 36{,}093)\right)$$

The exponential subroutine, R 220, is used to find

$$\exp\left(-\frac{(h - 36{,}093)}{20762} \cdot \frac{1}{4}\right)$$

where

$$-1 \leqslant -\frac{(h - 36{,}093)}{4 \times 20762} < 0$$

or

$$36{,}093 < h \leqslant 119{,}141 \text{ ft.}$$

Author: Mr. B.W. Gregory of Sir W.G. Armstrong Whitworth Aircraft Ltd.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

TRIPLE EXPONENTIAL STANDARD ATMOSPHERE

A subroutine to calculate the values of the Triple Exponential Standard Atmosphere.

Given h. 2^{-19} in X5, where $0 \leqslant h < 2^{19}$ (= 524288) ft. this subroutine places

p. 2^{-12} in X6,

ρ. 2^{8} in X7,

where h is height above sea level in feet,

ρ is air density in slugs per ft.^{3},

p is air pressure in lb. per ft.^{2}.

(R 971 uses R 220)

Name: Triple Exponential Standard Atmosphere.

Store: 4 blocks for R 971 + 2 blocks for R 220.

Uses: U0, 1; X1, 4, 5, 6, 7.

Cue: 01 (0+.0)

Link: Obeyed in 1.0. Not left in X1.

Time: Depends on the value of h; as h increases from 0 to about 274,000 ft., the time fluctuates between 92 and 103 milliseconds; as h increases from there up to 2^{19} ft. the time increases from 103 to 129 milliseconds.

Formulae: R 971 uses the following formulae to calculate p and ρ (see Ref. 1 or 2):

$$p = p_1 - g\rho_1 H(1 - e^{-h/H});$$

$$\rho = \rho_1 e^{-h/H}$$

The values of the constants p_1, ρ_1 and H are given in the following table:

Range of values of h (ft)	p_1 (lb./ft^2)	ρ_1 (slugs/ft.^{3})	H (ft.)
$0 \leqslant h < 35000$	2116.216	0.0023769	30801
$35000 \leqslant h < 140000$	2728.8	0.0040344	21016
$140000 \leqslant h < 524288$	818.4486	0.0009471	26859

The value of g is taken as 32.17405 ft/sec.^{2}.

References:

1. Royal Aeronautical Society Data Sheet 00.01.04 "Properties of the Upper Atmosphere." June, 1956.

2. R.A.E. Tech. Memo. G.W.250. "A proposed triple exponential standard atmosphere extending to 220,000 ft." F.G. Chapman, August, 1955.

Notes:

1. The formulae used by R 971 are intended to apply only to altitudes below 220,000 ft. (see Ref.1 or 2). R 971, however, uses the same formulae for all altitudes up to 2^{19} (= 524288) ft.

2. In References 1 and 2 the value of p_1 is taken as 818.3 lb./ft.^{2} for 140,000 ft. $\leqslant h \leqslant$ 220,000 ft. This has the drawback that the pressure calculated from the formula is negative for $h >$ 231,600 ft. (about), and the value of p_1 has therefore been amended as shown above to remove this anomaly. However, this introduces a discontinuity of about 0.15 lb./ft.^{2} in the calculated values of the pressure at 140,000 ft. A version of the subroutine which uses the original value of p_1 may be obtained by adding the following correction tape after the A3 on the master programme tape:-

```
C 971
T 3+.1
- 0.00003627844
```

Author: Mr. J. Stafford of Saunders-Roe Ltd.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
6.8.57.

PSEUDO-RANDOM NUMBER GENERATOR

A self-preserving subroutine for the computation of sequences of pseudo-random numbers. One number is produced each time the subroutine is entered and is left in X2.

Name: PSEUDORANDOM NUMBERS.

Store: 1 block.

Uses: U5; X1, 2, 6, 7.

Cues: 01 a-order partial cue.
02 b-order partial cue.

Time: 5½ milliseconds.

Link: Computing Store link obeyed in 5.4+. The jump address must be set in the modifier position of X1 before entry.

1. Method

1.1 The method used is given by the congruence

$$x_{n+1} \equiv kx_n \pmod{2^{31} - 1}$$

where x_{n+1} is the smallest positive integer satisfying the congruence and

$$k = 455\ 470\ 314 \equiv 13^{13} \pmod{2^{31} - 1}$$

x_0 is an arbitrary starting number and should be a positive integer of the order of 10^9 but less than $2^{31} - 1$. It is specified as a preset parameter. (See section 3.)

The sequence is repeated after $2^{31} - 2 = 2\ 147\ 483\ 646$ steps.

1.2 Each time the subroutine is used a new value of x_n replaces the previous one in X2. It is always a positive integer less than $2^{31} - 1 = 2\ 147\ 483\ 647$. Bits 0-7 in this register are thus always zero. Bits 8-38 may be considered as sources of random binary digits.

2. Instructions for Use

2.1 R 980 is intended to be used as a Computing Store subroutine. It is brought into U5 by the order

0 5 72

tagged by the appropriate partial cue, 01 or 02. The subroutine is self-preserving and is normally entered at U5.0+.

2.2 Before entering the subroutine for the first time, the starting value, x_0, (already set in 5.7 as a preset parameter) must be put into X2. This may be done either by the order 5.7 2 00 or by entry to U 5.0 instead of the usual 5.0+. The current value x_n is left in X2 after use of the subroutine, and must be preserved until the next entry. The series may be re-started at x_0 by jumping to 5.0.

3. Preset Parameter

The starting value (x_0) is specified as a preset parameter. The parameter list should be punched as follows:-

R 0 0 -0 1 980 - 04 -	Title of parameter list
+1357916503	x_0 (The value shown is only an example.)

Ferranti Ltd.,
London Computer Centre,
21, Portland Place,
London, W.1.

Issue 1
6th August, 1957.
A.R.E.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

Issue 1
6. 8. 57.

RANDOM NUMBERS ON DRUM

This subroutine fills a chosen sequence of locations in the Main Store with pseudo-random digits.

Name: RANDOM NUMBERS ON DRUM

Store: 5 blocks.

Uses: U 0, 1, 2, 3, 4; B0.

Cue: 01 (0+.1+)

Time: About 120 milliseconds per block of random digits generated.

Link: Obeyed in 0.0 and left unaltered in X1.

Notes: (1) The initial and final addresses ($B_i.P_i$ and $B_f.P_f$ respectively) of the sequence of locations to be filled with random numbers are set in X2 before entry to R 981 as a pseudo-order in the following manner:-

B_i – P_i0 0.
B_f – -- P_f

(2) All the 39 positions in the locations affected are filled with random binary digits; i.e. if the contents of such a location is considered as a fraction, y_i then $-1.0 \leqslant y_i < 1.0$.

(3) The method of generation of the random numbers is similar to that used in R 980.

(4) If the routine is entered a second time it will generate the same sequence of random numbers. If a different set is required a new starting value x_0 must be set in B4+.7 of the subroutine. x_0 should be a positive integer of the order of 10^9, (= 2 147 483 647) but less than $2^{31} - 1$. The initial value of x_0 set in the subroutine is 1 357 916 503.

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

MAGNETIC TAPE READ/WRITE - 16/32 WORD SECTIONS

R 990 can be used in any part of the store to read from or write to magnetic tape of 16- or 32- word sections.

Name: M.T.R/W - 16/32

Store: 4 blocks which may be stored above B127.

Uses: U0, 1, 2, 3, 4, 5; B0. The accumulators are stored in B0 and restored before exit except for X2 which will be shifted up 14 places.

Cues:

01

0+ - 00 0.	
0	

Modifier to be set in 2_m before obeying cue 02

02

0 [0] 72 2	
0.0 0 60	

Enter R990 with X2 containing cue 01

Link: Set in X1, obeyed in U4.5 and left unaltered in X1.

Time: About 48ms per section, reading from 16-word tape
45ms per section, writing to 16-word tape
64ms per section, reading from 32-word tape
64ms per section, writing to 32-word tape
plus search time for the required section.

METHOD OF USE

1. A subroutine modifier must be set in the most significant 14 bits of X2. This is effected by tagging a pseudo order-pair with cue 01. The subroutine modifier will be added into the order-pair, which should then be copied or added into X2. The most significant 25 bits of X2 will not be lost if the contents are shifted down 14 places before addition of the word containing the subroutine modifier: the routine shifts the contents of X2 up 14 places on exit.

2. A Tape Control Word, set in X6, should specify the address of the first section, *a*, and the number of the tape mechanism to be used, *m*. The Tape Control Word should be a "stop" order-pair if the tape has 16-word sections and a "go" order-pair if it has 32-word sections.

X6

a - -- 0(.)	
0 0 *m*0	

3. X7 should contain a Drum Control Word specifying the address of the first block, *B*, and the number of blocks, *n*. The word should be set as a "stop" pseudo order-pair if a read operation is required or a "go" order-pair if a write operation is required.

X7

B - 00 0(.)
n

Note that *n* is the number of 8-word Main Store blocks and not the number of tape sections. When writing to tape the last of the *n* blocks may not be the last block in the section of tape, but the remainder of that section will be overwritten.

Author: Mr. A.J. Jones of Babcock & Wilcox, Ltd.

London Computer Centre,
68, Newman Street,
LONDON, W.1.

Issue 1
12th November, 1962
A.J.J. J.L.

File as 992

FERRANTI LTD

PEGASUS LIBRARY SPECIFICATION

MAGNETIC TAPE COMPARISON

A complete programme to compare two 16 or 32 word magnetic tapes and print a record of the sections which disagree. The corresponding error sections are stored consecutively after the programme in the Main Store so that, if desired, the contents of these sections may be punched out when comparison of the tapes has been completed.

Name: COMPARE MAGNETIC TAPES

Store: 7 blocks

Uses: The entire Computing Store and 8 (or more) blocks in the Main Store. (See sections 5 and 6).

Entry: J 0+.0 at the end of the steering tape.

Time:
About 245ms for each 32 word section
160ms for each 16 word section
} Using Drum Store

About 150ms for each 32 word section
90ms for each 16 word section
} Using I.A. Store (Pegasus 2)

plus search time for the required section on each tape.

METHOD OF USE

1. The programme requires a steering tape punched as shown:

T 2	
$+m_1$	First mechanism number
$+a_1$	Address of first section of m_1
$+m_2$	Second mechanism number
$+a_2$	Address of first section on m_2
$+n$	Number of sections to be compared
$+N$	16 or 32
J 0+.0	

2. Place the programme tape in the main tape reader. START and RUN. (There is no T-sequence on this tape, and hence the programme will normally be read into B2 - B5 and B14 - B16. If it is required to preserve the contents of these blocks, the programme can be stored in any part of the store from block B onwards, by setting T B.0 on the handswitches before operating the Start key. The Relativizer will be set by a warning character B on the tape.

3. When the 77 stop (Z) at the end of the tape is reached, insert the steering tape in the main tape reader and operate the Run key.

4. The tapes are compared one section at a time and the corresponding sections are stored temporarily in B4+, 5+, (6+, 7+) and B8+, 9+, (10+, 11+) for 16 or 32 word tapes.

5. If an error is found, there will be an optional stop in 3.0, before the number of the section on m_1 is printed: this number will be in X7. The blocks of information currently being compared will be in U4 and 5. UI.7 will contain the address of block B, where $(B + 15)$ is the next Main Store block in which the error section will be stored.

6. When the Run key is operated, Initial Orders Number Print is used to print the contents of X7; and the two sections of tape are copied from B4+, 5+, (6+, 7+) and B8+, 9+ (10+, 11+) to B15+ onwards, using either 4 (or 8) blocks for each error section.

7. A 77 stop in 2.5 occurs when comparison is complete. Comparison may be recommenced by inserting another steering tape in the main tape reader and operating the Start and Run keys.

London Computer Centre,
68, Newman Street,
LONDON, W.1.

Issue 1
12th November, 1962
J.L.

END OF VOLUME 2