Marconi-Elliott Computer Systems Limited
A GEC-English Electric Company

Elstree Way, Borehamwood, Hertfordshire
Telegrams and Cables: Elliotauto Borehamwood
Telex: 22777 Answerback: Ellauto Borhwd
Telephone: 01-953 2030

Printed by Quadrant Offset Ltd., Hertford, England

Issue 1. E.S.D. 5/70

MECS

'Workshop'

marconi
elliott
computer
systems

T. J. Froggatt.

MANUAL OF WORKSHOP SYSTEM

M.H. Beilby.

Department of Transportation and Environmental Planning.

University of Birmingham

Head of Department: Professor J. Kolbuszewski.

T.

WORKSHOP

INTRODUCTION

'Workshop' is a computer program that allows Elliott 900 series 18 bit machines to calculate required formulae on demand. It has been developed by M.H. Beilby of the Department of Transportation and Environmental Planning at Birmingham University to provide rapid solutions to a wide range of numerical problems.

The program provides a system which can be used without any knowledge of computer programming. Once given mathematical formulae, the machine writes its own programs, runs them and provides the results without further attention. The formulae can range in complexity from the simplest addition of a calculating machine to the intricacles of design calculations and statistical tests.

The system can be fully utilised when a teleprinter is connected to the computer directly. In this 'on-line' mode, 'Workshop' becomes truly interactive. It checks the information that has been typed to ensure that the requirements of the operator can be fulfilled. Any errors are reported and may be corrected as they occur. Also on completion of the calculations, the required values are left in store for subsequent manipulations. In this form, the user who is not familiar with computers can take advantage of their speed and power of calculations while his more experienced counterpart may be relieved of repetitive simple programming.

When the teleprinter is 'off-line' to the computer, the system will read the formulae as they have been punched on paper-tape. It replies via the paper-tape punch on the computer. In this case the machine can process formulae in batches, working and commenting on them in turn without any intervention from the operator.

T.

CONTENTS

OPERATION OF 'WORKSHOP' ON-LINE

ENTRY TO SYSTEM

Once the 'Workshop' program has been loaded into the computer and triggered, the bell of the teleprinter rings and a > symbol is printed. The > symbol is a signal that always indicates that the computer is waiting for something to be typed.

Initially the operator must enter a reference code for the following calculations. To do this he will type a £ sign followed by a series of letters and digits. As a convention it is suggested that this might be his initials followed by a number e.g.

```
£MHB001
```

If 'cr lf' (carriage returnline-feed) is now typed, the machine will check whether the code is of an acceptable form. If, say, there were no code of this form e.g. the typing commenced with

```
>375.6 cr lf
```

then a message would appear asking the operator to identify himself so

```
>375.6 cr lf           ..... as typed by the operator
$ 3  <-----            computer report:
                       points to offending character
PSE GVE CDE $          "please give code"
$ BEG LST LNE$         "begin again at last line"
>                      ..... signal for more information
```

He would then start typing the correct code.

If the code is acceptable, the > symbol alone is printed and the computer waits for the continuation of instructions about the required calculation so

```
> MHB001 cr lf         ..... as typed
>                      ..... signal to continue.
```

At any time subsequently the operator may clear the system and begin again by retyping the £ sign with his initials and number. This eleminates all the information from the stores and provides him with a fresh start.

SPECIFICATION OF THE CALCULATIONS

Once in the system, the operator specifies the calculations he requires. He does this a line at a time. On completion of each line he types cr lf characters and waits for the computer to check the information. Any error that is found is reported inside $ signs and an indication is made of where, if possible, the operator is to start again. He will continue when the > symbol appears.

If an error is notice'd by the operator before the end of the line is reached, a < symbol may be typed so clearing the current line and providing a fresh > symbol. In certain circumstances (see p.A10) the operator may clear to an earlier stage than the beginning of the current line. To do this he must type the < symbol as the first character on a new line and await the computer report.

If the computer finds no fault with the line, it punches out a separate record at the paper-tape punch and provides a further > symbol on the teleprinter. The operator then continues with the next line.

The calculations themselves are specified by a series of typed commands. The use of the commands can best be illustrated by considering the 'Workshop' system as a substitute for a calculating machine and then developing the facilities.

1) THE 'WORKSHOP' SYSTEM AS A CALCULATING MACHINE

In the 'Workshop' system there are two commands that will provide the services of a calculating machine. They are the typed words PRINT and RUN. Once a PRINT has been typed the computer will expect a formula to follow. It will read and check it. If RUN is then typed, the formula will becalculated and the result printed on the teleprinter.

A simple operation might be to add up the values

$$67932.421$$
$$73645$$
$$374594.83 \times 10^5$$
$$0.632 \times 10^{-7}$$
$$.542$$

This calculation would be presented to the system by typing

```
>£MHB001   cr lf
>PRINT 67932.421 +73645 +  cr lf
>374594.83₁₀5 + 0.632₁₀-7 + .542  cr lf
>RUN cr lf
```

Here the > symbol is printed by the computer and the remainder typed by the operator. The $_{10}$figure is available of the teleprinter as a single character.

After the last lf character has been typed, the computer would print the result so

```
37459624578.0
>
```

to a rounded accuracy of 12 significant figures. The > symbol shows that the machine is ready for more formulae.

A further operation might be to calculate

$$\frac{67932.4210932}{.542 \times 73645.67777778} + 3.74958 - 2 \times 0.632 \times 10^{-7}$$

for which the user would then type

```
>PRINT cr lf
>67932.4210932  /  .542 * 73645.67777778 + 3.74958 cr lf
>-2 * 0.632₁₀-7  cr lf
>RUN cr lf
```

The formulae is typed as its figures and symbols are read, to assist in layout, the line can be broken anywhere and extra cr lf or space characters inserted. After each set of cr lf characters, however, the operator must wait for the computer to reply with a > symbol before continuing. The line may be started, though, with a single or string of lf characters on their own and the comments continued without a > symbol being printed by the computer.

After the last lf character of the above script, the computer would provided the result so

```
5.45146617657
```

with a signal to continue.

As the computer store is limited, it is necessary to restrict the number of values entered between RUN commands to less than 50. If this is exceeded an error message is printed and the current line of type rejected. Similarly, if the formula is too long (this is unlikely in this type of operation) the current line is rejected.

If the computer cannot perform the calculations for reasons, say, of dividing by zero, a report is made after the RUN command has been accepted e.g.

```
> £MHB002          cr lf
> PRINT            34/0.0 cr lf
> RUN              cr lf
  $DIV BY ZRO $                      Computer reports
                                     'Divide by zero'
  $$KP CUR CAL$                      'Skip current calculation'
>                                    ..... signal to continue
```

Sometimes the SKP CUR CAL report is not made. In this case the computer will attempt to continue the calculation with its own corrections. This procedure is then described in a computer report.

Any values can be entered or printed within the range

$$2^{1024} - 1 \text{ to } -2^{1024}$$

i. e. approximately

10^{308} to -10^{308}

The machine is not able generally to cope with values whose size is below 2^{-1024} i. e. approx. 10^{-308}. However, while it is calculating a formula, these ranges may be violated to the order of 64 times. If the ranges are exceeded, then an error message will be printed.

Other mathematical operations are provided besides those used above. The following is a list of the operations that are useful in the calculating machine operation. It will be appreciated that the teleprinter keyboard is limited with regard to symbols and so the typed forms are listed alongside the mathematical symbols.

	mathematical symbol	teleprinter symbol
Addition	+	+
Subtraction	-	-
Division	$\div$ or /	/
Multiplication	x	*
Raise to the power e.g.	$6^{-.33}$	↑ e.g. 6↑ -.33
Brackets ( ) e.g. 3 x (4+2)		( ) e.g. 3 * (4 + 2)
Value of Pi	π	Q
e.g. 2 π		e.g. 2 * Q
Square root	$\surd$	QSQT
e.g. $\sqrt{5}$		e.g. QSQT 5
Natural log ln or $\log_e$		QLNE
Natural antlog) 3.4)e or exp (3.4) Exponentiation)		QEXP e.g. QEXP 3. 4
Cosine of angle in radians cos		QCOS
Sine of angle in radians sin		QSIN
Arctangent as angle in radians $\tan^{-1}$		QATN
Integer part [ ] or int		QINT
Modulus or mod		QMOD
Delta (x) = 1 if x = 0 = 0 otherwise		QDEL

Sign	Sign (x) = -1 if x < 0 = 0 if x = 0 = +1 if x>0	QSGN
Factorial ! or L e.g. 7! or L7		QFAC e.g. QFAC7

These operations may be nested inside each other and may refer to compound expressions inside round brackets. It is suggested that they are typed exactly as they are read from a written formula. If there is any doubt as to the construction of the formula it is better to be liberal with the round brackets.

It is worth noting that the / symbol is stronger than the multiplication *. Therefore if the following is presented

5/2 * 3

the result is 5/6 = .83333 and <u>not</u> 15/2 = 7.5.

Some examples will demonstrate the use of 'Workshop' as a calculating machine.

1) Area and circumference of a circle radius, r = 7.245

Formulae: Area πr^2, Circumference = $2\pi r$

```
>£MHB003      cr lf
>PRINT        Q * 7.245 ↑ 2   cr lf
>RUN          cr lf
```

and

```
>PRINT  2 * Q * 7.245 cr lf
>RUN  cr lf
```

2) Annual percentage growth rate represented by an increase from 572 to 894 over a period of $3\frac{1}{2}$ years.

i.e. $\left(\left(\frac{894}{572}\right)^{1/3\frac{1}{2}} - 1\right) \times 100$

```
>£MHB004      cr lf
>PRINT  ((894/572) ↑ (1/3.5) - 1) * 100 cr lf
>RUN          cr lf
```

3) Length of a parabolic segment of height, x = 3.9 and base, y = 7.349

$$= \sqrt{4x^2 + \left(\frac{y}{2}\right)^2} + \frac{y^2}{8x} \log_e \frac{\left(2x + \sqrt{4x^2 + \left(\frac{y}{2}\right)^2}\right)}{y/2}$$

```
>£MHB005      cr lf
> PRINT QSQT (4 * 3.9 ↑ 2  + (7.349/2) ↑  2) + cr lf
```

```
>(7.349 ↑ 2/8*3.9)  *  QLNE ((2*3.9 + QSQT (4*3.9↑2 cr lf
>+ (7.349/2) ↑ 2)) / (7.349/2)  )   cr lf
>RUN cr lf
```

2) WORKSHOP SYSTEM FOR REPEATED CALCULATIONS

An immediate development of the calculating machine operation would be to expect the system to be capable of repeating the formula calculation for different values. This it will do on the introduction of algebraic letters instead of numbers. To extend the use of the system, three additional commands are relevant - WHERE, CALCULATE and REPEAT.

In this new type of operation, the formula is used in its text-book form including the algebraic letters.

An example might be

$$\sqrt{x^2 + y^2}$$

which the system would recognise as

```
QSQT (X ↑ 2  +  Y ↑ 2)
```

whether the X and Y are the variables of the formula. There are two acceptable forms for the variables:

i. A single letter taken from the list

ABCDEFGH OP RSTUVWXYZ

ii A pair of letters, the first of which is from the list

ABCDEFGHIJKLMNOP RSTUVWXYZ

and the second of which is from the list

ABCDEFGH OP RSTUVWXYZ

Thus the following are acceptable variable names:

AB A D T OT IR OR TX ZF KC S W:

while these are not acceptable

Q A1 L N FK WJ QA DFR GYTU

It will be noted that some of the non-acceptable forms have other meanings to the system.

The introduction of the WHERE command provides a means of assigning values to the variables. In the same way that a PRINT command is followed by a statement of a formula, the WHERE command is followed by a statement of the value that a variable is to take. The form of the statement is

WHERE   variable name = value

For example

```
WHERE  A = 5.8974
```

would instruct the machine to give the value 5.8974 to the variable A whenever it was found in a formula. Therefore the example of the area of the circle πr^2 would become

```
>£MHB006
>WHERE  R  =  7.245
>PRINT  Q  *  R ↑ 2
>RUN
```

It will now be possible to see the advantage of using variables instead of numbers. If it was required to repeat the calculation for a different value, then all that need be done is to type a further WHERE statement after the first result has been obtained. For example, once the result had appeared in the above calculation, the operator would then type

```
>WHERE  R  =  8.435

>RUN
```

to get the result for a circle of radius 8.435. This process can be repeated for the values required.

An alternative means of assigning a value to a variable is by means of a calculate statement. This will calculate a formula and assign the resulting value to the variable. The form of a CALCULATE statement is

CALCULATE   variable name = formula

and it can be used to break down a complicated formula. If the example MHB005 of the parabolic segment where to be re-written, it could be presented as:

```
> £MHB007
> WHERE  X  =  3.9
> WHERE  Y  =  7.349
> CALCULATE  SR  = QSQT  (4 * X ↑  2  +  (Y/2)  ↑ 2)
> PRINT  SR  +  (Y ↑ 2/8*X)  *  QLNE (  (2*X  +  SR)  /  (Y/2) )
> RUN
```

Here X and Y would be assigned the values 3.9 and 7.349 respectively, the square root calculated and assigned to SR and, finally, the value of the formula printed. Further results could be obtained by typing WHERE statements followed by a RUN command. If a new value of X were to be 4.23, then the operator would type

```
WHERE  X  =  4.23
RUN
```

after the first results had been printed. In this case, the Y value would remain the same and the SR value modified.

In order to avoid repetitive typing of command words it is possible to group command statements together. In this case one command word is typed followed by a statement. If the statement is terminated with a ; symbol a second statement may be typed with the command word assumed. For example:

```
>WHERE X = 4
>WHERE Y = 5
```

may be replaced by

```
>WHERE X = 4 ;
>        Y = 5
```

In the particular case of the WHERE command it is possible to omit the ; symbols altogether.

The ; symbol can, in fact, be used to terminate a statement even though the same command is not to be repeated. This is especially useful when the operator is unsure of the statement that he has just typed. In this case the ; symbol followed by the characters cr lf will help clear up any problems before the next command is entered. For similar reason it is useful to always start a new command or new command statement on a fresh line. This can give more freedom in error recovery messages.

The system can handle up to 26 CALCULATE or PRINT statements at any one time. They can be written in any order intermingled with WHERE statements. On the command RUN the computer will attempt to execute the statements in the order given. If at any stage this is not possible, then it will switch the order, holding back the difficult statement until it is possible to operate it.

As many as 25 variables names may be used in the statements. Their values are held in the store until they are superceded at which time they are removed to make way for fresh values.

One of the facilities of the Workshop system is that the computer continually re-organises itself automatically as the calculation proceeds. Once an initial set of results have been obtained the values remain assigned to the variables. If it is required to repeat the calculations for different variable values as has been shown, it was only necessary to re-type the WHERE statement. In this case any previous WHERE or CALCULATE statement that assigns values to the same variable is removed from the memory. Now if some variation of the formula is required for the same variable values, further PRINT or CALCULATE statements may be entered. Once a PRINT or CALCULATE statement has been entered to the system after the first RUN, all such statements up to the last RUN are deleted from the store. If in example MHB007 it was subsequently required to learn of the value of SR, the operator would simply have to type a further

```
>PRINT          SR ;
>RUN
```

In such a situation the value of SR would remain and be printed without the computer recalculating SR or printing out the formula for a second time.

The automatic re-arrangement of the calculations can be avoided if a REPEAT command is used. The word REPEAT may be typed at the beginning of a further series of runs to preserve the PRINT and CALCULATE statements that have so far been accumulated. For example if instead of the above PRINT modification it had been required to recalculate the formula of example MHB007 for the case of X = Y = 7.349, the operator would have typed

```
>REPEAT
>CALCULATE    X = Y ;
>RUN
```

and the original value of X would have been overwritten by the value of Y, the value of SR recalculated and the new value of the formula printed.

As a general rule it can be considered that further WHERE statements will overwrite previous assignments to the same variable, and further PRINT or CALCULATE statements will supercede any such statements previously accumulated. It will not harm the system to add REPEAT statements at the beginning of a further set of calculations if the previous calculations are to be repeated. The following worked example will serve to illustrate the organisation of the store:

```
>£MHB008
> WHERE A = 3
> PRINT  A↑2
> RUN

  9.00000000000

> WHERE A = 2
> RUN

  4.00000000000

> REPEAT
> CALCULATE A = B
> WHERE B = 5
> RUN

  25.0000000000

> PRINT A + B
> RUN

  10.0000000000
```

```
>WHERE  A  =  3
>RUN
 8.0000000000
>PRINT
>A   ;
>B   ;
>RUN
 3.00000000000
 5.00000000000
>
```

Two further facilities are useful when entering the formulae. These are the insertion of comments and the extended use of the < symbol.

After the initial reference code it is possible to type a comment about the formulae that follow. It must be formed by letters and digits that do not spell out exactly any command word. This facility may also be used immediately after a RUN command, e.g.

```
> £MHB009
>  THIS IS A COMMENT AND MUST NOT CONTAIN
>  IN EXACT FORM ANY PRINTS CALCULATES WHERES ETC.
>  PRINT  3/4+2
>  RUN
    2.75000000000
>  THIS IS ANOTHER COMMENT
>  PRINT 3 + 4
>  RUN
    7.00000000000
>
```

As has been mentioned earlier, the < symbol can be used to delete the entered formulae further back than the current line of type. If the symbol is typed immediately at the beginning of a line, the machine will refer the operator back to the beginning of the command statement that he was typing at the beginning of the current line. For example in the case

```
> £MHB010              cr lf
> PRINT  A + B -       cr lf
> C * D                cr lf
> <
```

the operator would be referred back to the beginning of the current command i.e. back to just after the word PRINT. This is especially useful after some error in an extended formula e.g.

```
> £MHB011
> PRINT (A  +  B  - C  *  D
> * (e  -  f) ;
  $  *  (E  -  F)  ;  <----          Computer report:
  TOO MNY  ($                        Too many brackets opened
  $BEG LST LNE  $                    Recover beginning last line
> <                                 < Symbol typed
  $BEG LST COM$                      Computer report
                                     recover last command
> (A  +  B  -  C)  *  D
> *(E  -  F)  ;                      Correct statement typed
>                                    and accepted
```

This form of recovery is not always possible, especially when there are several statements on one line.

3. WORKSHOP SYSTEM WITH MATRICES AND INDICES

It is possible within the system to extend any variable name to refer to an array of values rather than a single value. Workshop will handle formulae involving vectors and matrices and provides general facilities to manipulate them.

Any variable name can be used to refer to an array if index letters are attached. These letters can be from the list

IJKLMN

and can appear in any order. The following are possible array names

X1 FG11 FJJ V1KJ CLMJK F1KJLMN

The number of indices attached will indicate the dimension of the array and once a name is used the system will expect the operator to be consistent with the number of dimensions. However, the size of each dimension may vary over different calculation runs. It is possible to accommodate up to 6 dimensions for a particular variable.

The WHERE statement provides a means of assigning values to an array. This time the statement includes all the array values in place of the single value previously e.g.

```
WHERE A1 = 45  5.678  34.6  45.9 ;
```

This statement will assing to the variable A the values 45, 5.678, 34.6 and 45.9 in turn wherever it is used in a formula. If more than one dimension is required, the additional values are laid out as they might appear on paper e.g.

```
        > WHERE  BIJ
        > 1  0  0  0
        > 0  1  0  0
        > 0  0  1  0
        > 0  1  4  1  ;
or,     > WHERE  CIJK
        > 1  0  0
        > 0  1  0
        > 0  0  1
        >
        > 2  3  4
        > 5  6  7
        > 8  9  0  ;
```

which represents a 4 x 4 matrix and a 3 x 3 x 2 matrix respectively. In the case of CIJK the last dimension is separated by more lf characters than any other dimension. As a general rule higher dimension blocks are separated by more lf characters than any lower dimension. The system is prepared to automatically compound down the number of dimensions if there are less required than might at first appear. For example if the last case had been

```
        > WHERE  CIJ  =
        > 1  0  0
        > 0  1  0
        > 0  0  1
        > 2  3  4
        > 5  6  7
        > 8  9  0 ;
```

then the first dimension would be dropped to provide CIJ with the 9 x 2 matrix of values, the first dimension being extended over three lines of type.

As a convention, the system expects the I dimension of a variable to be typed across the script, possibly extending to further lines if the dimension is a long one. The J dimension will be typed downwards, each matrix row being separated by the same number of lf characters. This must, of course, be greater than the number of lf characters used to space the elements inside the I dimension. Similarly the K dimension blocks will be spaced by the same number of lf characters between each block which in turn is greater than any number of lf characters used before, and so on to the N dimension.

When the variable appears in a formula that is part of a CALCULATE or PRINT statement it is always assigned values in order of the index letters IJKLMN. The computer will automatically loop through the dimensions with I in them, making I = 1, 2, 3 etc. before looping through the dimension with J. This applies along the list. An example might be to follow the above WHERE statement that assigns values to CIJ with a CALCULATE statement so

```
        >CALCULATE DIJ  =  CJI  ;
```

in which case DIJ would become a 2 x 9 matrix

```
        1  2
        0  3
        0  4
        0  5
        1  6
        0  7
        0  8
        0  9
        1  0
```

This is exactly the pattern the PRINT statement will adhere to in its layout of results. If the operator had wanted to print out the values of the DIJ matrix in the above form, he would have typed

```
        > PRINT DIJ  ;
```

If he has wanted to print the matrix in the original form he could have typed

```
        > PRINT DJI  ;
```

in which case the computer would have tried to print 2 rows of nine figures. In the process of trying this it would find that as many figures as could have been typed on a line had been typed and would automatically continue on a fresh line. The 9 x 2 matrix would appear as

```
        1.00000000000    000000000000    000000000000    000000000000
        1.00000000000    000000000000    000000000000    000000000000
        1.00000000000

        2.00000000000    3.00000000000   4.00000000000   5.00000000000
        6.00000000000    7.00000000000   8.00000000000   9.00000000000
        000000000000
```

The PRINT statement automatically spaces out the results according to the convention of the WHERE statement for all dimensions.

When the system is scrutinising the statements that are typed it examines the number of the dimensions to check for consistency both for the same variable name and within a particular formula. The question it asks itself is whether the statements make sense logically. If not, then an error message is printed. If the size of dimensions do not match, then an error message will appear during the calculation run.

The dimensions of a matrix may be contracted by typing one of the digits 1, 2, 3, 4, 5, 6, 7, 8 or 9 in place of the index. This will then cause the variable to take on the values only in the specific dimension e.g.

```
> CALCULATE E1  = D21 ;
```

where DIJ is the variable defined above. This would cause E1 to adopt the values 2 3 4 5 6 7 8 9 0 in its first dimension. Alternatively CALCULATE F = D15 ; would assign the value 1 to the variable F.

There is a third method of accessing elements of a matrix by index declarations. This involves using a formula to find the index value required. If the variable name only is followed by an open square bracket, a formula for each dimension separated by commas and a closed square bracket thus

```
C  [formula, formula]
```

then wherever this variable might appear the formulae inside the brackets is calculated and the integer parts of the values used instead of indices. Further any index that is included inside the brackets has its range of values interpreted in the main formula. These index declarations <u>cannot</u> be nested. An example might be

```
> WHERE AIJ =
> 10   20   30
> 40   50   60
> 70   80   90
> 100 110  120  :
> B1  =  1  3  2  2  ;
> C1  =  2  2  3  4  ;
> CALCULATE D1  =  A  [BI, CI]  ;
```

which would assign to the variable D the values 40, 60, 80, 110 respectively.

It is possible to use the index values on their own in the calculation. Then the index is written as though it were a variable with zero dimensions e.g.

```
CALCULATE EI = CI + I ;
```

would provide the variable E with values 3, 4, 6, 8. It must be noted that if the operator entered

```
CALCULATE EI  =  1  ;
```

there would be no way by which the system could assign a range of values to the index. He may create a matrix from indices by writing, for example

```
CALCULATE EI  =  1  +  CI  -  CI  ;
```

which would construct an array of values for the variable E based on the values of the indices and the size of the array of values for the variable C. Otherwise the statement would be assigned the dimension zero.

If it is required to overwrite the assumed ranges for indices a range declaration may be placed either at the beginning or end of the statement by typing

[index = formula. $\uparrow\uparrow$ formula, index = formula $\uparrow\uparrow$......]

and can be read as

'''for' index 'equals' formula 'up to' formula 'and for ''' where the equivalents are

[----- 'for' = ----- 'equals' $\uparrow\uparrow$ ----- 'up to' , ----- 'and for

with the close brackets as the end of the declaration. It will therefore be possible to subdivide an existing array or create a new one, for example

CALCULATE GIJ = AIJ [I = 2 $\uparrow\uparrow$ 3,   J = 3 $\uparrow\uparrow$ 4] ;

would partition out of the above matrix the values

```
80   90

110  120
```

and assign them to the variable G. Another example is

CALCULATE UIJ = QDEL (I - j) [I = 0 $\uparrow\uparrow$ 3, J = 0 $\uparrow\uparrow$ 3]

which would assign to U a 4 x 4 unity matrix. Any formula can be placed inside the range declarations as long as it itself does not include such range declarations or have any loose indices. This is a reinforcement of the statement that the system cannot cope with any illogicalities.

There are a number of loop functions that are of particular help in manipulating matrices. They are QSUM, QPRD, QMAX and QCNT and enable the operator to sum, find the product, find the maximum and count the elements in an array. The functions are immediately followed either by the indices over which it is required to execute the function or by an index range declaration of the above form. The full six dimensions may be used and the functions can be nested. Also it is possible to repeat the same index letter both outside and within a function and inpute the logical meaning to it. For example

```
WHERE A1  =  1  1  2  ;

PRINT  QSUMI  (A1 + QSUMI A1)  ;
```

would provide the value 16. Further examples are

```
WHERE A1 = 1 1 2

CALCULATE BIJ = A1 [ J = 1↑↑QCNTI A1 ] ;
```

which would construct for the values of B a square matrix of the same row values as the variable A i.e.

```
1   1   2

1   1   2

1   1   2
```

which could be followed by

```
PRINT QSUMI BIJ ; QSUMJ BIJ ;
```

which would then print out the row and column totals as a horizontal and vertical set of values.

There is a further facility that enables the operator to find the inverse of a square non-singular matrix. This is the matrix function QMIN which can operate <u>only</u> on a two-dimensional array. If the array is not square the system will attempt to cut the second dimension size down to match the first dimension. If this is not possible, it will print out an error message. An example of its use might be

```
> WHERE XIJ =

> 3    7    9

> 12   5    3

> 76   45   64

> CALCULATE IXIJ = QMIN XIJ ;
```

In this case it can be considered that QMIN XIJ is the IJ element of the matrix inverse of X. In the example the inverted matrix would be passed in value to the variable IX. The matrix inversion usually uses a large amount of store and it is not usually possible to invert a matrix of size greater than 11 x 11.

If during the calculations it is found that the store restrictions are pressing, it is possible to discard some of the information using a FORGET command. If the operator types in the word FORGET followed by a variable name only e.g.

```
> FORGET X ;
```

where X is the name of the above array, then all assignments of values either by WHERE or CALCULATE statements are cleared from the store. This comman is particularly useful when inversion is required.

4. WORKSHOP FOR ITERATED CALCULATIONS

There is a further command that enables the system to be used for iterated calculations. This is the command word ITERATE. Basically it is a version of the CALCULATE command with the following differences.

An ITERATE command may assign values to a variable which already has values. Therefore it is possible to write, for example

```
> WHERE A = 4

> CALCULATE B = 2 * A

> PRINT B

> INTERATE A = B

> RUN

> RUN

> RUN

> RUN
```

to find the values 8, 16, 32 and 64 printed out between the RUN commands.

A second difference is that the ITERATE statement must come logically at the end of the list of WHERE, PRINT and CALCULATE statements. Then the values are updated ready for the next iteration.

Several ITERATE statements can be used together. They may generally be used with all the other facilities of the CALCULATE statement.

It is important to note that, in line with the other commands, it is <u>not</u> generally possible to assign a value to a variable in terms of a formula containing the same variable. This is an important difference to all other computer languages. A secondary variable must be used when the ITERATE statement reassigns values to an array.

OPERATION OF WORKSHOP OFF-LINE

ENTRY TO THE SYSTEM

When Workshop is used off-line to the computer the user's requirements are presented to the machine on paper tape. Once the program has been loaded and triggered, the tape is offered to the paper-tape reader on the computer. This then scans through the tape and extracts the necessary information.

The paper-tape is punched on a flexowriter. The information typed is in the form of a script resembling closely the layout used for the on-line teleprinter. In this case, however, the script is typed in its entirety and errors cannot be discovered until the whole off-line operation has been completed. If there are errors in the script, the lines of script containing the errors will be ignored by the system.

The script must begin with a reference code. This comprises of a £ sign, the user's initials and his own reference number. They are typed as in the example

```
£MHB001          nl
```

where the nl character is a newline of the flexowriter. This replaces the cr lf characters of the teleprinter. The £ sign will serve to clear the system and punch out a series of header blanks on the computer punch, the initials to identify the user when the results are finally printed, and the reference digits will provide the user with his own referencing code.

The user may string together on the tape a series of separate runs by simply repeating a reference code at the beginning of each run. Each time the computer reads the code it will completely clear the system and start afresh.

SPECIFICATION OF THE CALCULATIONS

Once he has typed his introduction to the system the user will specify the calculations that he requires. This is done by a series of command statements which are logical descriptions of the required formulae and the values that are to be placed in them. The form of the statements is described below.

If the user wishes to remove an error during typing, then he may effectively erase the current line by typing a < symbol.

OPERATION OF THE SYSTEM

The completed script is offered to the computer which will then read it a line at a time. A fresh copy of the correct script is output at the paper-tape punch interspersed with reports on errors that have been detected. The machine will then 'run' the calculations as requested in the script describing any further errors by means of the output tape. The output tape is finally printed on the flexowriter complete with results.

SPECIFICATION OF FACILITIES

The following is a specification of all the facilities used in the Workshop system. Cross-references to extended descriptions are provided in brackets.

SYMBOLS

i. The characters o, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z and - are generally used to describe command words, reference codes, variable names, functions, values and comments.

ii. The characters a, b, c, d, e, f, g, h, i, j, k, ı, m, n, o, p, q, r, s, t, u, v, w, x, y, z, which can be used in off-line operation, are read as capital letters.

iii. The character nl (newline) has the same effect as the character lf (line-feed) both terminating the current line of type (see pages A1, A2, B1). This will occur when they are not the first characters typed on a line or when they do not immediately follow a nl or lf character. They are treated otherwise as separators which have a particular meaning in the WHERE statement (see page A6).

iv. The characters cr (carriage-return), bell, tab, space and halt are all read as sepratators, used only for terminating words and assisting in the layout of formulae. In a formula separators are ignored unless they are essential to the meaning.

v. Erase characters and blank tape are ignored by the system.

vi. The sign $ has a meaning on output only when it is used to bracket error reports or computer comments.

vii. The & symbol, when read, throws the system into a 'systems jump' This has the effect of providing an exist from Workshop to the surrounding program system environment. The jump is effected immediat ely the computer receives the character.

viii. Open and closed brackets ( ) are used to partition sections of a formula. They can be nested up to 49 times in a formula, but in practice the limit is usually less than this.

ix. The mathematical symbols *, -, +, and / provide the operations multiplication, subtraction, addition and division for use in writing formulae. The system logic places a priority on them of the order + - / * i. e. multiplication is carried out before division which is carried out before subtraction which is carried out before addition.

x. The symbol $\uparrow$ is used in a formula to denote 'raise to the power'. It is considered of a higher priority than the multiplication and will be executed before it. It is permitted to write $\uparrow$+ or $\uparrow$- in a formula to denote 'raise to the positive/negative power' as a special case of two mathematical symbols occurring together. The double characters $\uparrow\uparrow$ are used in range declaractions to mean 'up to' (see page A 15).

xi. The symbol , is used in index and range declarations (see page A15). In the former it can be read as 'and the index for the next dimension is given the value of' and so serves to partition the formulae of the declaration. In the latter it means 'and for' where it partitions the declaration of the index ranges.

xii. The symbol ; is used to terminate a statement. It is not essential, but will cause any error report to be made on the last line of the statement rather than at the beginning of the next command word, thus aiding error recovery. It has the effect of terminating the statement and setting the last command word as the assumed command word for the next statement.

xiii. The symbol < is used to recover from errors. If it is not the first character to be typed on a line, it will clear the line from the buffer in the system. If it is the first character, then it is interpreted as a request to 'begin at the last command word' (see pages A10, A11).

xiv. The = sign is used to read as 'equal to' in the WHERE, CALCULATE and ITERATE statements or in a range declaration. (see pages A6-9, 15-17) As such it separates the variable or index from a formula or a set of values.

xv. The > symbol is used with a space character by the computer to signal that more information is expected. It has no meaning when typed by the operator.

xvi. Square brackets [ ] are used to contain index or range declarations. If the opening bracket immediately follows a variable name, then it is assumed to open an index declaration with the meaning 'where the index of the first dimension takes the value of the formula' (see page A14) Otherwise, if it immediately follows the name of a loop function or a separator it is used to open a range declaration, 'for' (see page A15) The closed bracket terminates the declarations.

xvii. The £ sign has the effect of clearing the system completely and must be followed by the operator's reference code. It also provides a length of blanks at the paper-tape punch.

Unless specifically mentioned, the symbols are read by the system from a buffer of 120 characters. They will not, in general, be examined immediately except for a parity check.

OPERATOR'S REFERENCE CODE

To enter the system the operator has to type a £ sign, to clear the system, followed by his reference code (see pages A1, B1) The reference code must begin with a letter and be a continuous sequence of letters or digits terminated by a nl / lf character or a separator. The first six characters of the sequence are stored in the system in internal code.

The initial triggering of the system has the same effect as the £ sign except that it will also provide a > symbol on a newline at the teleprinter and a halt code at the paper-tape punch.

It is impermissible to have a reference code of a command word.

COMMENTS

A comment may be typed immediately after the reference code or after a RUN command has been executed. It may contain any of the writeable characters as long as there is nowhere an exact command word with its terminating separator (see page A9)

SPECIFICATION OF THE CALCULATIONS

The calculations are specified by a series of statements each of which is made up from various combinations of command word, variable, formulae and values.

Command words. There are seven command words - PRINT, CALCULATE, WHERE, RUN, REPEAT, FORGET and ITERATE. The words are recognised only after an initiating separator or lf / nl characters and must be formed by the exact letters followed by a ; , a terminating separator or lf / nl character. If a second command word follows a PRINT, CALCULATE, WHERE, FORGET or ITERATE word immediately, then first command word is ignored.

Variables. A variable can appear in one of two forms -

i. A single letter from the list ABCDEFGH OP RSTUVWXYZ followed by a combination of indices from the list IJKLMN.

ii. A pair of letters, the first of which is from the list ABCDEFGHIJKLMN OPQRSTUVWXYZ and the second from the list ABCDEFGH OPQRSTUVW XYZ. Index letters may again follow.

In particular, the combination RUN will be interpreted as a command word.

The number of index letters attached to the variable name give dimension to the variable. It must be the number of dimensions whose size is greater than one, i.e. unity dimensions are not allowed and are considered as an operator error. The number of dimensions assigned to to a particular variable name must be consistent throughout the calculations. It is possible to assign up to six dimensions to a variable.

A maximum of 25 different variable names may be used in a calculation. This is reduced by one every time a different variable matrix is inverted.

Values. Values are formed by a continuous string of characters from the set 0123456789_{10}+- . and are ordered by an optional initial sign, a set of digits with a decimal point if necessary, and an optional $_{10}$ followed by more digits. A + or - sign may follow the $_{10}$ character.

The letter Q on its own is used to represent the value of IL in a formula.

Except for the values of indices, all values are entered and stoored by the system in three-word packed real form, giving permissible ranges of $+2^{1024}-1$ to $+2^{-1024}$, 0 and -2^{-1024} to -2^{1024} i.e. in terms of decimal exponents, 10^{308} to 10^{-308}, 0 and -10^{-308} to -10^{308} approx. During manipulations of values in mathematical operations the ranges may be violated by a factor of 2^{64} with operation in terms of four-word unpacked real form. Results can generally be considered as accurate to 12 significant figures if no matrix operations are used. With matrix operations this may be reduced as a result of manipulation.

Formulae. Formulae are written as combinations of mathematical symbols, variable names, constant values, index and range declarations, calculation functions, loop functions and matrix functions. They appear in strict mathematical form with the teleprinter symbols and functions substituting the mathematical symbols. (Each facility of the formulae is described in more detail below).

In general, any indices that are not explained within the formulae are assigned ranges from the associated variables. Therefore the formulae themselves may be considered as having several dimensions.

The framework of the formula is expected of the form symbol, non-symbol, symbol, non-symbol etc. This can be broken with the combinations $\uparrow$ + or $\uparrow$ - or by brackets. The formula or sub-formula within brackets may commence with a symbol or non-symbol, where, in the former case, an initial value of zero is assumed.

The function of the individual statements can be described in terms of the above components.

WHERE Statement (see pages A6, A7)

The WHERE statement allows values to be assigned to a variable name. It is written in the form

WHERE variable = values

If the variable is an array, it must have indices attached to indicate the number of dimensions. By convention these indices must preserve alphabetical order.

The values are layed out according to the number of dimensions required. The lf / nl character is used to break the dimensions. A new dimension is opened as soon as there are more consecutive lf / nl characters than have so far appeared between values. The only exception is the first dimension which can extend over several lines of type.

The order of the dimensions is understood as the same as the order in which the dimensions are opened in the layout of the values.

Any errors in layout are reported as they are detected. It is not possible to make the system accept a layout in part, it will only accept layouts that completely match the variables' indices.

If several variables are to be assigned values, the ; symbol is assumed as soon as following variable name is written. Several assignments may therefore be made without repetition of the WHERE command or ; symbol.

CALCULATE statement (see page A7)

The CALCULATE statement allows a new variable to be assigned values of formula of old variables. The form is

CALCULATE variable = formula

If the variable has indices attached, the formula must have the same dimension as the indices indicate. The values will be assigned in the order of the dimensions which will follow also the alphabetical order of the indices.

If there are dimensions of the formula that are not deduceable from information already available, then the dimension must be declared with index range declarations. The information about all index ranges will be checked when the statement is read by the system.

A variable name cannot appear on both sides of the = sign.

PRINT statement (see page A2)

The PRINT statement enables the values of a formula to be printed out when the calculations are run.

PRINT formula.

The values of the formula will be printed according to the alphabetical order of the indices. The I dimension across the page, possibly overflowing to extra lines, the J dimension in blocks separated by two nl characters, the K dimension separated by three nl characters, and so on.

All nl characters on output are accompanied by cr and blank characters. Also additional newlines are output between PRINT statements. The order that the statements are executed will generally be the order in which they are specified. If, due to some run time error, it is not possible to execute some PRINT statements the order may be changed, i.e. when the NOT ENH INF report is made the order of the statements in store is changed.

ITERATE statement (see page A17)

The ITERATE statement will overwrite a variable's value by the values of a formula.

ITERATE   variable = formula

The variable which is the subject of the statement must have values previously defined by either a WHERE or CALCULATE statement. Each time the calculations are run, the statement will then redefine the values assigned to the variable. The dimensions must match on either side of the = sign (c.f. the CALCULATE statement pages

A variable of zero dimensions may appear both as the subject variable and in the formula. However, if the variable has an array of values attached, the old values will be cleared before the statement is executed.

FORGET statement (see page A2)

Values may be deleted from store by means of a FORGET statement.

FORGET   variable name

The variable name must not be accompanied by its indices. The number of dimensions of the variable is preserved.

REPEAT statement (see pages

The command REPEAT preserves all the calculations that were in operation on the last RUN command. It must either follow a RUN command or not be preceded by a PRINT, CALCULATE or ITERATE command within a set of statements.

RUN (see pages

The command RUN causes the system to execute all PRINT, CALCULATE and ITERATE statements so far accumulated. Values assigned by the ITERATE or CALCULATE statements are preserved (see pages

When the PRINT, CALCULATE and ITERATE statements are read a series of programs are written in store. These programs are executed on a RUN command. The system continually reorganises the placing of the programs, removing redundant calculations wherever possible. WHERE, FORGET, REPEAT and RUN statements are, in contrast, executed as soon as they are entered, the values being placed immediately in store amongst the above programs.

Redundant calculations and variable values are removed according to a flag set in the system. It comes into operation as soon as the first set of formulae have been calculated. The flag has two states 1 or 0 and the latter state is set immediately after a RUN command has been obeyed. When subsequent command words are entered the flag is adjusted and the store organised accordingly.

	Flag initially	Flag set to	Existing Store
WHERE	0 or 1	No effect	No effect
CALCULATE) PRINT) ITERATE)	0	1	Previous formula removed
	1	1	No effect
FORGET	0 or 1	No effect	No effect
REPEAT	0 or 1	1	Variables calculated from current formulae cleared
RUN	0	0	Variables calculated from current formulae cleared
	1	0	No effect

Once the organisation has been completed the full command statement is obeyed. Error recovery over a command word reverts to the situation before the command was read.

In addition, a set of variable values may be overwritten when new values are described by means of a WHERE, CALCULATE or ITERATE statement. Previous values are deleted from the store as soon as the statement is entered. Error recovery, however, in general will allow the operator to revert to a situation in store before the command word had been read.

FACILITIES AVAILABLE IN FORMULAE

i. Value of PI. The value of PI is available for use inside a formula. It is obtained by typing the letter Q. The value is retained as a constant in store throughout the calculations and is accurate to 15 significant figures.

ii. Additions and subtractions. Additions and subtractions are obtained from the symbols + and - typed within a formula. The operations are accurate to 15 significant figures. There is, however, a test for zero in the result of the order of 13 significant figures.

iii. Division. Division is available through the symbol/. The process used is that of long division accurate to 15 significant figures.

iv. Multiplication. Multiplication is available through the symbol *. The system adopts a procedure of multiplication by parts and is accurate to 15 significant figures.

v. Raise to the power. A value may be raised to a positive/negative integer/real power. If the power is negative the value is inverted using division. An integer power less than 2^{17} is then obtained by a contracted repeated multiplication involving at most 17 steps. Large integer powers are calculated by taking logarithms of the modulus of the value with an appropriate sign adjustment. If the sign adjustment is ambiguous then a comment is printed. Real powers are handled by means of logarithms.

vi. Logarithm. The natural logarithm routine can be called by the group of letters QLNE. The subject of the logarithm can appear as a value or as an expression within round brackets. Whichever is the case the routine will find the natural logarithm of a real number. The value is taken as a fraction and a series calculated on the basis

$$\log_e \left(\frac{x - 1}{x + 1} \right)$$

The multiplier of the series is always less than 1/9 and the series is collected until terms are no longer of significance at 15 figures. The value is adjusted according to the exponent of the original real number.

vii. Exponential. The exponential routine is called by the letter set QEXP. The value or expression will be reduced to a real number which becomes the subject of the routine. The real number is amended and a series of exp x constructed with a multiplier between -0.7 and 0.7. The series is terminated at 15 decimal places and the result modified in its final exponent value.

viii. Sine and Cosine. Routines to calculate the sine and cosine of an angle in radians can be called by use of the sets of letters QSIN and QCOS. The subject of a cosine is modified to provide the subject of a sine. The series is calculated from

$$\cos \frac{\pi}{4}$$

and the multiplier of the series is always less than 0.16. The series terminates when additional terms are no longer of significance at 15 figures. Angles may be of any size but will, of course, be only interpreted at the accuracy to which it can be reduced to the range of 0 to $\pi/2$.

ix. Arctangent. The arctangent of an angle may be found in radians by the set of letters QATN. The routine uses the series for

$$\arctan x$$

where the multiplier of the series is never in excess of 1/4. The series is terminated when the terms are no longer of significance to 15 figures and the final result is modified to fit the range

$$+\frac{\pi}{2} \text{ to } -\frac{\pi}{2}$$

x. Square root. Square root can be obtained by using a set of letters QSQT. The routine finds the square root by using the logarithm and exponential routines.

xi. Factorial. The factorial of a value can be obtained by writing QFAC. The subject of the factorial is initally reduced to an integer part and then the value of the factorial calculated by repeated multiplication.

xii. Delta. A logic function is provided to recognise values which are exactly zero. The consequence of QDEL is unity if the subject is exactly zero and zero otherwise.

xiii. Sign. A logic function is provided to recognise the sign of a value. The consequence of QSGN is to provide zero if the subject is exactly zero, +1 if it is positive and -1 if it is negative.

xiv. Modulus. The modulus function is provided to extract the size of a value regardless of sign.

xv. Integer part. The integer part of a value can be extracted by the letter set QINT. The subject value will then be cut off downwards to an integer value.

xvi. Loop Functions. It is possible to obtain the operations of summation, product, maximum value and count by use of the sets of letters QSUM, QPRD, QMAX and QCNT, each of which calls a loop function.

Each loop function must be immediately followed by either a list of indices or a range declaration to describe the dimensions over which the operation is required. In the former case the ranges of the indices are assumed from the following arrays and in the latter case they are set by the declaration. The system will write the appropriate program loops to count in steps of 1 through up to 6 index ranges in their alphabetical order.

The loop functions can be nested up to 10 deep and the indices are attached to the loop functions in the conventional way.

Once a summation loop has been opened, it is subsequently closed at the priority level of a + or - sign. A product loop is closed by the priority level of a /, *, + or - sign. The maximum and count loops are closed by the priority of any of the mathematical signs.

xvii. Matrix functions. There is one matrix function provided which is written as QMIN. This will provide the inverse matrix of the subject of the function. It operates on a two-dimensional array in a formula and will provide the element of the inverse indicated by the indices.

When the function letters are entered the system automatically declares an inverse array in store. If they are used several times in a set of calculations the array is shared. The indices that are attached to the subject variable will then serve to extract a value from the inverse array rather than from the original. This secondary array is organised in the system in the same way as the original.

The values for the inverse array are constructed as soon as the RUN command is entered. This will be before any formulae are calculated. Should the original array not be calculated until some formula has been run, then the inverse is constructed as soon as possible.

The calculation of the inverse has as an initial stage a check on dimensions. If the original matrix is not square, the system will attempt to partition it. Should the attempt fail, an error message is printed. The partitioning is only possible if the first dimension of the original array does not exceed the second dimension in size. The matrix used for inversion will be the largest square matrix that starts with the first row of the original.

Singularity is tested by means of a breakdown in the calculations. The construction of the inverse involves building a double matrix in store from the original matrix and a unity matrix. The double matrix is then reduced by a series of row and column manipulations comparable with the solutions of a set of simultaneous equations. The manipulations are all carries out in unpacked real representation at an accuracy of 15 significant figures and involve multiplication, division, addition and subtraction operations.

As a final stage the resulting unpacked matrix is packed to provide a rounded 13 significant figure representation in store in common with other storage of values.

xviii Range declarations. Range declarations may be used to declare or modify the ranges that indices will adopt in a formula. They can appear immediately after a loop function name and so control the indices of the loops or otherwise be placed at the beginning or end of a formula and so modify the index ranges over which the formula is to be calculated.

The ranges of indices are set when a formula is read by the system. Indices are initially set to take the unity value only. If an index occurs after a variable name then the ranges are modified to follow the size of dimensions in the variable array.

These ranges are considered completely overwritten by a range declaration. However, the ranges are set, the indices will be updated in steps of one from the lower bound to the upper bound. The range declaration involves specifying the bounds by means of separate formulae.

The complete declaration must be contained inside square brackets and the separate range declarations for each index separated by commas. For each index the declaration takes the following form

Index name = formula $\uparrow\uparrow$ formula.

The index name is separated from the rest of the declaration by means of an equal sign. The lower bound value is specified by a formula followed by $\uparrow\uparrow$ characters. These characters can be read as 'up to' and will precede the upper bound value specified by a formula.

When the declaration is read separate programs are written to compute the values of the formulae, and extract the integer part of the result. At the end of the declaration these programs are placed within the main calculation program so as to immediately precede the opening of the index loop in question. This means that at the beginning of the loop all the necessary information must be available to calculate the bounds. Also any indices mentioned in the declarations and not themselves set in range in the declaration will be carried outside to the main formula to a position before the loop in question starts. As such there is no logical way in which a free index can be interpreted inside a range declaration that applies to a whole calculation formulae.

The range declarations cannot be nested. They also involve a reshuffling of the calculation program currently under construction and so provide some restriction on error recovery. Recovery messages are, however, diverted and the extent of the recovery changes can be deduced from the computer reports.

xix. Index values. The current value of an index can be obtained in a formula by writing the name of the index required. If the index is not used to access elements in an array directly, there is no reason why the range declaration should not allow it to take on negative or zero values.

xx. Index declarations. A particular element of an array may be accessed by either mixing in with the indices that follow the variable name a digit in the range 1.......9 inclusive or by replacing the whole set of indices by declaration inside square brackets.

In the first case a digit is used to replace the index letter in the dimension that is being contracted. The array is then treated as one with one less dimension than the original.

In the second case the variable name is followed by a set of formulae separated by commas all inside square brackets. There must be one formula for each dimension of the variable array. When such a declaration is read a separate program is constructed to find an integer value for the particular dimension. When the full calculation program is run these sub-programs will provide the values of indices to eventually access the array element. As such there is no conclusion drawn by the system at entry time about the sizes of dimensions of the variable

If values indicate an element outside the variable array, then an error message is made when the programs are run. Ranges on indices inside the declaration are interpreted as the placing of the variable name might suggest in the main formula.

Index declarations may not be nested.

ERROR MESSAGES

Errors may be noticed by the system when the statements are entered or when the calculations are run. They result in messages being printed inside $ symbols.

i. Errors on statement entry. The entry of statements provides the system with a means of reporting on errors and the operator with a chance of immediate correction. The form of an error report at this stage is to print out the last line of the current statement entered up to and including the offending character. This character is then emphasised by an arrow and a following message is printed to indicate the nature of the error (a list of the messages is given below).

Once the error has been reported, the system will attempt to provide the operator with an opportunity to start his script again from the beginning of his incorrect line of type. Should this not be possible, it will attempt to start him at the beginning of the erroneous statement i. e. immediately after the statement's command word. As a final resort the operator will be referred back to the beginning of his calculations. The system's conclusion about the recovery point is reported in a further message.

In particular, difficulty in error recovery may arise when range declarations are used in formulae or when there is a large amount of information already in store.

It is possible for the operator to call for recovery to the beginning of the current line or to the beginning of the current statement by means of a < symbol. It may not always be possible for the system to follow this direction.

ii. Errors at run time. There are two main methods of reporting errors that are detected when the system is running the package of calculations, by using an assumed value and by skipping the current calculation.

Some errors are reported whilst the results are being printed. The system will attempt to assume values for the particular calculation and so enable other results to be printed. In this case a message is typed about the action that the system has taken and the printing of results resumed below the part at which it broke off. The following result printed will have resulted from the system's escape from the error.

If an error is repeated, or if there is no appropriate action to take the current calculation is finished and the system moves to other calculations. In this case the order of the calculations may be affected. After a report on the error an indication of where it has occurred may be made followed by a report indicating that the system is skipping over the calculation for the current run.

The error messages are written in a contracted word form. The words are designed to quickly indicate what the system sees the mistake to be. It is quite possible that the error has arisen from another source! The following is an explanation of the contracted words used together with a reference to the aspect of the system that is either under question or will provide further explanation.

PSE GVE CDE	Please give code. Asks the operator to specify his reference code again or otherwise to give his reference code (see page C3).
NO REC LBL	Not recognise label. A new variable name has been specified that should have occurred before. Usually found in a FORGET or ITERATE statement (see page C6).
TOO MNY VAL	Too many values. There is not enough room in store to accommodate the values given.
DIM NOT BAL	Dimensions do not balance. The layout of values in a WHERE statement does not provide a rectangular array (see pages C4, C5).
DIM INX ERR	Dimension index error. The dimensions given to the variable which is the subject of a WHERE, CALCULATE or ITERATE statement do not logically match the layout of the free indices in the formula (see pages C4, C5, C6).
TOO MNY CAL	Too many calculations. Too many formulae have been specified in the current set of calculations i. e. the list of formulae is too long (see page A8).
NOT ENH INF	Not enough information. There is not enough information to complete the current set of calculations (see page D1).
VAR TWO LTS	A variable name has at most two letters. The current statement appears to use a variable name which does not fit the requirements (see page C3).
NO VAR OR=	No variable or = sign. The current WHERE, CALCULATE or ITERATE statement does not have a subject variable (see pages C5 & C6).
OVR WRT VAR	Over-write variable values. The statements so far will over-write values already allocated to a variable name.
TOO MNY VAR	Too many variables. In the set of calculations specified so far there are more variable names declared than can be accommodated (see page. A8)
NO ROM LFT	No room left. The working store of the system has no space in which it can place programs or values. There are too many values in store or too long a set of calculations.
NO SYM =	No symbol =. An = symbol has been omitted from either a WHERE or CALCULATE statement (see pages C4 & C5).

INX NOT ORD	Indices not ordered. The indices attached to the variable which is the subject of a WHERE, CALCULATE or ITERATE statement are not presented in alphabetical order (see pages C5 & C6).
TOO MNY BRK	Too many brackets. There is not enough room in the system to accommodate such a degree of nesting of round brackets in the formula (see pages C1).
TOO MNY)	Too many closed brackets. The number of round brackets opened is less than the number closed in the formula.
NO NST DEC	Not nest declarations. An attempt has been made to nest either index or range declarations (see pages C10, C11 & C12).
TOO FEW)	Too few closed brackets. The number of round brackets opened exceeds the number closed in the formula.
FN LBL ONY	Function label only. A group of letters appears to have been used for a variable which can only be used to indicate a function operation (see pages C3).
MIS WRT FN	Mis-write function. It appears that a function operation has been mis-spelt (see pages C4).
TOO MNY LBL	Too many labels. There is not enough room to accommodate so many variable names. (see pages C4).
DIM NOT CST	Dimensions not consistent. The number of dimensions attached to the variable name is not the same as was used before (see pages C4).
NO REL INX	No related index. There is no logical way of determining the index values (see pages C10 & C11).
TOO MNY INX	Too many indices. The number of indices indicate that the number of dimensions exceed 6 (see pages C3 & C4).
TOO MNY CST	Too many constants. There is not enough room in the system to accommodate so many constant values in formulae (see pages C9).
NO CAL ROM	No calculation room. There is not enough storage space left to run the calculations.
TOO MNY LPS	Too many loops. The system cannot accommodate so many loops nested (see pages C9).
LNE TOO LNG	Line too long. Too many characters have been placed on a line of type (see pages C2).
PAR ERR	Parity error. A character has been read for which the parity check has failed.
IN INV	In matrix inversion of A comment on where the above error has been detected.

IN CAL	In calculation of . A comment on where the above error has been detected.
SKP CUR CAL	Skip current calculation. A comment on what the system has done (see pages D1).
NO REC POS	No recovery possible. The system is unable to recover from the error (see pages D1).
BEG LST LNE	Begin at last line oi type.
BEG LST COM	Begin at the end of the command word attached to the current statement i. e. the statement which has not yet been terminated by a further command word (see pages A11).
INX TOO SML	Index too small. A value of an index is too small to be used to access an element (run a variable array) (see pages C11).
ASS VAR ZRO	Assume variable value zero. A comment that indicates that the system has not been able to access an element from a variable array and is assuming the variable has zero value.
INX TOO LRG	Index too large. A value of an index is too large to be used to access an element from a variable array (see pages C11).
MIX INV DIM	Matrix inverse dimensions. An attempt has been made to invert an array of values which is not two-dimensional (see pages C10).
MTX INV NSQ	Matrix inverse not square. An attempt has been made to invert a matrix whose dimension sizes do not match (see pages C10). In particular whose first dimension exceeds the second.
NO ROM INV	No room for inverse. There is not enough room to invert the required matrices.
MTX INV SNG	Matrix inverse singular. An attempt has been made to invert a singular matrix (see pages C10).
INT TOO LRG	Integer too large. An attempt has been made to find the integer part of a value which is larger than 2^{17}
ASS LRG VAL	Assume large value. A comment that the system is continuing with the calculation on the assumption of the largest acceptable value.

DIV BY ZRO	Divide by zero. An attempt has been made to divide by zero.
ROT NGV VAL	Root negative value. An attempt has been made to find the root of a negative value.
PWR SGN AMB	Power sign ambiguous. An attempt has been made to raise a value to a large power with the result that the sign of the result is ambiguous (see pages C8).
ASS SGN POS	Assume sign positive. A positive sign is assumed as a conclusion to the above report.
WNG CHR	Wrong character . The character upsets the logic of the statement.
VAL TOO LRG	Value too large. A value has arisen in calculation that exceeds the ranges allowed (see pages C4).
VAL TOO SML	Value too small. A value has arisen in calculation that exceeds the ranges allowed (see pages C4).
ASS ZRO VAL	Assume zero value. A comment on the continuation of a calculation assuming the current value is zero.
CAL TOO LRG	Calculations too large. The formula at present being entered is too large for the amount of store available.
LNE ZRO VAL	Logarithm zero value. An attempt has been made to find the natural logarithm of a zero value.
LNE NGV VAL	Logarithm negative value. An attempt has been made to find the natural logarithm of a negative value.
FAC NGV VAL	Factorial negative value. An attempt has been made to find the factorial of a negative value.

WORKSHOP SYSTEM PROGRAM

The 'Workshop' program is written in Elliott S. I. R. language to operate in 8k of a 900 series machine. The program occupies locations 32 to 5950 inclusive and uses as store all locations up to 8000. There are two versions, on-line and off-line, differing by one instruction only. When an on-line teleprinter is used the copy of the entered text that could be produced at the paper-tape punch is suppressed.

The binary-dump version is a contracted binary form allowing quick loading. It has an additional 200 locations attached which are overwritten as soon as the system is used. These extra locations hold a self copying program which can be triggered at location 8. The system is loaded under initial instructions and the tape is self-triggering at 32 on a successful load. If it is misread a string of erase characters is output at the punch.

M. H. Beilby

Department of Transportation
and Environmental Planning