FOREWORD

The Tabular Interpretive Programme (T.I.P.) was devised by the Mathematical Services Group of Bristol Siddeley Engines Limited in 1957 and was intended primarily as a computing aid to be used directly by the Company's engineers. Largely through the agency of the DEUCE Users Association T.I.P. came to be widely used in many other computing installations in this country. The extensive use made of the scheme encouraged further development and a new version T.I.P.2. was published in 1958.

T.I.P.2 was considerably faster than the original scheme, mainly as a result of working in block floating arithmetic. Changes were also made in the order code structure to achieve greater simplification from the users viewpoint.

T.I.P.3 to which this manual refers was completed for DEUCE in August 1960 and although a form of fully floating arithmetic has been reintroduced a further increase of speed is obtained by the avoidance of unnecessary drum transfers of data between the execution of successive codewords. The order code is basically the same as that for T.I.P.2. except that codewords are now identified by a relative address system. Input and output may be in a variety of formats, the programme itself reading and punching in binary using auxiliary programmes for conversion. The scheme consists of the following parts.

1) T.I.P.2 to T.I.P.3 converter, ZC29T.
   This programme produces binary T.I.P.3 codewords from binary T.I.P.2 codewords.

2) T.I.P.3 Compiler, ZC26T.
   Reads decimal codewords, carries out validity checks. If correct a binary version of the programme is punched out, if incorrect error cards are produced ready for printing on the card operated typewriter.

3) T.I.P.3 Interpreter, ZC27T.
   Main interpretive programme

4) T.I.P.3 Restore Control Programme, ZC28T.
   Special programme to restore control in the event of a machine failure.

5) LK15T and LK16T
   DEUCE Library service programmes for decimal to binary and binary to decimal conversion of data.

PART I

COMPUTING WITH INTERPRETIVE SCHEMES

The purpose of the Tabular Interpretive scheme is to provide a quick and easy method of programming computers without specialised knowledge of the computer itself. This is done by presenting instructions to the computer in a notation which is simply a formalised version of that already in everyday use with desk calculating machines. The Tabular Interpretive programme (T.I.P. for short) transcribes each instruction into a corresponding piece of computer programme (i.e. a network of specialised instructions in the computer's own order code) and carries out the operations specified, all this without the intervention of a programmer.

It must be pointed out, however, that whilst T.I.P. is an efficient technique to employ on calculations of tabular form it does not make efficient use of a computer for non-tabular work. Before using T.I.P. it is necessary to ensure either that the problem is one of tabular arithmetic or that it can be conveniently expressed in tabular form by, for example, solving several cases of a "single line" problem in parallel.

Because the language of T.I.P. is independent of the order code of the computer used, it is possible, in principle, to carry out calculations specified by T.I.P. instructions on any computer having sufficient storage capacity. The interpretive programme itself must, of course, be couched in the language of each particular type of computer and this, as one might expect, puts a practical limitation on the theoretical generality of T.I.P. To date T.I.P. has been made available for the DEUCE Marks I, II, IIA and the Ferranti Mark One Star computers.

PART I

CALCULATIONS IN TABULAR FORM

The tabular layout of a calculation for a desk machine usually takes the form of a sheet of paper marked out in columns, the first few columns being filled with data; a list of values of the constants being used, such as π or g; instructions to perform arithmetic operations on one or two columns and to put the results in a third column. When using T.I.P. the computer can be envisaged as having internal storage laid out in a similar manner. This store consists of 128 columns (numbered 0 - 127) each having 30 rows (numbered 0 - 29) together with a small additional store for 128 constants (N0 to N127). This store is effectively larger than may appear at first sight since unlike a desk machine tabulation, where each result must be put in a fresh column on the paper, the T.I.P. store is erasible so that results can overwrite any information which is no longer required. In fact an instruction to write anything in a T.I.P. column automatically erases information previously in it. In this way large calculations can be handled which would otherwise require several times the space actually available.

T.I.P. instructions are called codewords and consist of four elements of the form:

a (or Na) b(or Nb) c (or Nc) r

which, in general, can be interpreted as meaning 'Upon the numbers in column a (or that in constant position Na) perform, with the numbers in column b (or with Nb) the operation whose code number is r, and store the results in column c (or in Nc).' Any one or more of a, b and c may be replaced by Na, Nb or Nc provided that data and results are compatible (i.e. an operation on two constants cannot produce a column of results and vice versa). The range of r is 0 - 31 and the various operations denoted by this code will be discussed in the following pages.

The first 32 constants (N0 to N31) are pre-set with useful numbers, physical constants, conversion factors, etc. They may be overwritten if required, and will be automatically reset at the end of each programme. The pre-set constants are as follows:-

N0 = 0	N8 = 57.29578	N16 = 1.23	N24 = 550
N1 = 1	N9 = 2240	N17 = 65.7633	N25 = 0.491
N2 = 2	N10 = 10	N18 = 3.5	N26 = 0.6849
N3 = "dash"	N11 = 386.088	N19 = 96.014	N27 = 7.8125×10^{-7}
N4 = 3.141593	N12 = 12	N20 = 14.656	N28 = -3.8281×10^{-5}
N5 = 32.174	N13 = 288.16	N21 = 360	N29 = 2.0711×10^{-3}
N6 = -1	N14 = 14.696	N22 = -26,200	N30 = -2.9890×10^{-2}
N7 = 0.33	N15 = 273.16	N23 = 0.1	N31 = 100

The "dash" stored in N3 is a number such that any operation on it leaves it unchanged. The way in which the dash is used will become evident from examples given later in this manual.

Throughout T.I.P. numbers are stored to six significant figures. All numbers must be less than 2^{62} (approximately 4.6×10^{18}) in magnitude. Numbers smaller than 2^{-66} (approximately 5.3×10^{-20}) will be automatically set to zero.

The codewords for the basic arithmetic operations are as follows:

Codeword				Interpretation
a	b	c	0	(col. a) x (col. b) putting result in (col. c)
a	b	c	1	(col. a) ÷ (col. b) putting result in (col. c)
a	b	c	2	(col. a) + (col. b) putting result in (col. c)
a	b	c	3	(col. a) - (col. b) putting result in (col. c)

As previously mentioned the column numbers may be replaced by constant numbers wherever necessary. If an operation is performed on columns of different lengths, the resulting column will be the same length as the shorter of the two.

Example 1

With the operations already described it is possible to write a number of simple programmes. Consider, for example, the evaluation of the power series:

$$y = ax^4 + bx^3 + cx^2 + dx + e$$

for a number of values of x. Let us assume that the values of x are stored in col. 0 and the coefficients a,b,c, d and e in N32 - N36. A suitable programme to form y in col. 1 would be:-

Codeword				Description
a	b	c	r	
N32	0	1	0	Form ax in col. 1
1	N33	1	2	" $ax + b$ in col. 1
1	0	1	0	" $ax^2 + bx$ in col. 1
1	N34	1	2	" $ax^2 + bx + c$ in col. 1
1	0	1	0	" $ax^3 + bx^2 + cx$ in col. 1
1	N35	1	2	" $ax^3 + bx^2 + cx + d$ in col. 1
1	0	1	0	" $ax^4 + bx^3 + cx^2 + dx$ in col. 1
1	N36	1	2	" $y = ax^4 + bx^3 + cx^2 + dx + e$ in col. 1

Replacing Logarithmic and Other Tables

It is very often necessary in doing a calculation by hand to refer to books of tables to find such functions as sines of angles or logarithms. It is, however, uneconomic to store such tables in a computer as these functions can usually be computed more quickly than they can be read from tables. Seven T.I.P. codes are given over to the generation of such functions:

Codeword				Description
a	-	c	6	Form the square root of col. a in col. c
a	-	c	8	" " logarithm of col. a in col. c
a	-	c	9	" " anti-log of col. a in col. c
a	-	c	11	" " sine of col. a in col. c
a	-	c	12	" " cosine of col. a in col. c
a	-	c	23	" " inverse sine of col. a in col. c
a	-	c	24	" " inverse cosine of col. a in col. c

As before a and c of the above codes may be replaced by Na and Nc. All angles are assumed to be given in radians and all logarithms are to base e. An automatic check is carried out by T.I.P. to ensure that the numbers in col. a (or in Na) are compatible with the function it is required to evaluate. In the event of incompatability a "dash" will be inserted in the appropriate row or rows of col. c. A situation such as this arises, for example, if any negative numbers appear in a column of which the square root or logarithm is required.

Example 2

The following programme will evaluate $y = \sinh(\cos^{-1}(x \log x))$ for a set of values of x, assuming the initial set of values of x in col. 0:

Codeword				Description
0	-	1	8	Form $\log x$ in col. 1
0	1	0	0	" $x \log x$ in col. 0
0	-	0	24	" $\cos^{-1}(x \log x) = t$ in col. 0
0	-	0	9	" e^t in col. 0
N1	0	1	1	" e^{-t} in col. 1
0	1	0	3	" $e^t - e^{-t}$ in col. 0
0	N2	1	1	" $y = \sinh t = \frac{1}{2}(e^t - e^{-t})$ in col. 1

Summing Series

In numerical work it is frequently necessary to sum power series. This can, of course be done as was shown in Example 1 with a few multiplication and addition codewords, but for convenience a single codeword is available to do the entire sequence. The instruction:

a Nb c 7

will produce in column c the sum of a power series, where the values of the argument are given in col. a and the order, followed by the coefficients (the coefficient of the largest power first) in Nb onwards. Thus Example 1 could have been done with a single codeword. Assuming this time that the constant stores N32 to N37 contain 4, a, b, c, d and e, and col. 0 the values of x then the instruction 0 N32 1 7 would evaluate the series and put the resulting values of y in column 1.

The Shift Instruction

Finite difference calculus frequently calls for differences of the form $x_{n+1} - x_n$. To facilitate this type of work a T.I.P. instruction is available to shift a column up or down. The codeword

a 0 c 10

produces in column c the elements of column a shifted up one row. If, for instance, column a contained x_0, x_1, x_2, x_3 and x_4 the above codeword would produce in column c: x_1, x_2, x_3 and x_4. The length of column c automatically becomes one less than column a.

If it is required to shift a column down the instruction:

a Nb c 10

will give column a shifted down by one row in column c with the constant stored in Nb inserted in the first row. If after a shift down the first row becomes meaningless, a dash (constant N3) can be inserted and no further operations will be carried out on that row. Thus if column a originally contained x_0, x_1, x_2 and x_3 the codeword a N3 c 10 would produce in column c, -, x_0, x_1, x_2, and x_3. The length of column c will automatically become one more than the length of column a unless column a contained the full 30 elements initially in which case the last element of column a will not appear in column c.

Summation.

It is occasionally necessary to sum the elements in a column and this may be done with the codeword

a - c 15

which will form in row i of column c the sum of all rows of column a up to and including row i. This will be easily understood from the example below, showing the contents of columns a and c after obeying the codeword.

Row	Col. a	Col. c
0	x_0	x_0
1	x_1	$x_0 + x_1$
2	x_2	$x_0 + x_1 + x_2$
3	x_3	$x_0 + x_1 + x_2 + x_3$
4	x_4	$x_0 + x_1 + x_2 + x_3 + x_4$

Example 3

The shift and summate codewords may be used to perform simple numerical integration. Let us suppose that it is required to evaluate the integral

$$Z_n = \int y \, dx$$

over a range of values for the function y, the numerical value of which is known for various x in the range 0 to n. A simple approximation to the integral is:

$$Z_n = \sum_{i=1}^{i=n} \tfrac{1}{2}((y_i + y_{i-1})(x_i - x_{i-1}))$$

Assuming x to be in column 0 and y in column 1 the following sequence of codewords will compute the values of Z_n in column 2.

	Codeword			Description
0	N3	2	10	Form x_{i-1} putting dash in first row
1	N3	3	10	" y_{i-1} " " " " "
0	2	2	3	" $x_i - x_{i-1}$
1	3	3	2	" $y_i + y_{i-1}$
2	3	2	0	" $(x_i - x_{i-1})(y_i + y_{i-1})$
2	N2	2	1	" $\frac{1}{2}(x_i - x_{i-1})(y_i + y_{i-1})$
2	-	2	15	" $\Sigma\frac{1}{2}(x_i - x_{i-1})(y_i + y_{i-1})$

Transfer

In the integration described above a column of values of Z_n covering various values of n was obtained. It is quite possible that only one of these would be required (probably the last in the column) to be used at some later stage of a tabulation. It is, therefore, convenient to have a codeword which can select an element from a column and transfer it to the constant store. This is one of the functions of the transfer code in T.I.P. The instruction:

a b(or Nb) Nc 13

will put row b ($0 \leqslant b \leqslant 29$) of column a into Nc or alternatively put the row of column a specified in Nb into Nc. Thus by adding the instruction 2 25 N40 13 to the set of codes in example 3 the value of $Z_{25} = \sum_1^{25} y \, dx$ would be stored in N40. Other forms of the transfer instruction exist.

a 0 c 13

or Na 0 Nc 13

will put a copy of column a into column c, or of Na into Nc, leaving a or Na unchanged.

Na b(or Nb) c 13

will put Na into row b of column c, or into the row specified by the number stored in Nb. This form of transfer can also be used to extend the length of a column. If, for instance, column c has n rows and $b \geqslant n$ all the rows from n to b inclusive will be filled up with copies of Na and the length of the column will become (b + 1). Thus if the length of column c is originally 3, and the column contains x_0, x_1 and x_2 then the instruction Na 5 c 13 obeyed with Na containing t, will produce in column c:
x_0, x_1, x_2, t, t and t and the length will be 6, (rows 0 to 5 now being occupied).

Modulus

The modulus instruction calls for no special comment. The code

a - c 14

will put in column c the moduli of the numbers in column a.

Reading Graphs

In many cases the algebraic equation for a function is not known and only graphical representation is available. It is usually difficult to find equations approximating satisfactorily to empirical functions so in T.I.P. graphical data is given to the computer as a set of cartesian co-ordinates from which required values are computed by linear interpolation.

Given a graph representing a function of a single variable a number of points, not exceeding 30, must be selected along the curve and the values of the dependant and independant variables at these points listed. Taking as an example the function.

$$y = F(x)$$

then the points must be distributed in such a manner as to give sufficiently accurate representation of the original curve. The values of x need not be given at equal increments. If column a contains the values of x for which the function is required; column b contains the selected co-ordinate values of x (in order of increasing magnitude) and column b + 1 contains the corresponding values of y = F(x) then the instruction

a b c 22

will calculate y for the given values of x by linear interpolation on the graphical co-ordinates stored in columns b and b + 1 putting the results in column c. Extrapolation is not permitted. Before the codeword is obeyed T.I.P. checks the values of x given in column a to see whether they lie within the range of the co-ordinates of x given in column b. If any are outside these limits a dash will be inserted in the appropriate row of the result.

If column b + 1 has been formed by the previous codeword then this column must not be overwritten by the results of the interpolation (i.e. if the interpolation codeword is numbered n, then $c_{n-1} = (b+1)_n = c_n$ is not permissible, though any two of the three may be equal).

End of Programme

All programmes must end with the instruction

\- - - 31

This will cause all stores to be cleared and the constants NO to N31 to be reset. The computer is then prepared to read in codewords for the next programme.

Example 4

To calculate the roots of the quadratic equation $ax^2 + bx + c = 0$ for a number of sets of values for a, b and c. This example assumes a in col. 0, b in col. 1, c in col.2 and computes the two roots in columns 4 and 5 using the formula

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Codeword				Description
N2	N2	N32	0	Form 4 in N32
1	1	3	0	" b^2 in col. 3
0	2	4	0	" ac in col. 4
N32	4	4	0	" 4 ac in col. 4
3	4	3	3	" $b^2 - 4ac$ in col. 3
3	-	3	6	" $\sqrt{b^2 - 4ac}$ in col. 3
NO	1	4	3	" -b
4	3	5	3	" $-b - \sqrt{b^2 - 4ac}$ in col. 5
4	3	4	2	" $-b + \sqrt{b^2 - 4ac}$ in col. 4
N2	0	3	0	" 2a in col. 3.
4	3	4	1	" $(-b + \sqrt{b^2 - 4ac})/2a$ in col. 4
5	3	5	1	" $(-b - \sqrt{b^2 - 4ac})/2a$ in col. 5

If this example were a complete programme three additional codes would be needed, one to read the values of a, b and c into columns 0, 1 and 2, one to print out the results from columns 4 and 5, followed by an end of programme codeword.

No mention has been made so far as to how numbers can be put in the computer or of how selected columns or constants, representing final results, can be printed out. As with other T.I.P. operations reading and printing functions are controlled by codewords. Physically the input and output to the computer will be by means of punched paper tape or cards, but this is the province of the computer operator so the mechanics of the process are irrelevant to the T.I.P. user. The user may assume that the read instructions cause the computer to transfer data from a typed or handwritten sheet to the appropriate part of the computer storage. Similarly the print instructions can be imagined to effect a typewritten copy of selected parts of the computer store.

Reading

The instruction

a - c 29

will read 'a' consecutive columns into column c onwards.

Constants are read in by the instruction

1 - Nc 29

which will read one group of constants (a group consists of any number of consecutively stored constants) into position Nc onwards.

Printing

Before describing the printing instructions it will be necessary to distinguish between fixed and floating point numbers. This distinction should be clear from the example below.

Fixed Point Numbers	Equivalent Floating Point Numbers
317.5	3.175×10^{2} (printed as 3.175 2)
96.31	9.631×10^{1} (printed as 9.631 1)
0.00328	3.28×10^{-3} (printed as 3.28 -3)

It is evident, as can be seen from this example, that fixed point numbers varying widely in magnitude require more digit positions than in floating point representation. Because of the limited number of digit positions available it is only possible to print out from T.I.P. in fixed point when magnitudes of numbers within a group do not vary too greatly. Constants are always printed out in floating arithmetic form, but with columns, four alternatives are available. They are:

(i) A codeword to print out a group of consecutive columns in fixed point. The codeword to do this is a b 0 28. This will print b consecutive columns starting with column a. (c = 0 indicates that the output is to be in fixed point). If this codeword is used two conditions must be fulfilled:

  (a) Elements of all the relevant columns must be within a reasonable range for fixed point working

  (b) No columns may be longer than the first of the group

(ii) If the elements of the required result columns vary too much to satisfy (i) (a), but do not vary unduly within any one column, the columns may be printed out separately (i.e. each from a separate print codeword) by means of the codeword a 1 0 28. This causes column a to be printed out in fixed point.

(iii) If floating point output is necessary then the instruction a b 1 28 will print out b consecutive columns beginning with column a, in floating point notation (c = 1 denotes floating point output) provided condition (i) (b) is complied with.

(iv) If this latter condition is not met, individual columns may be printed with the instruction a 1 1 28 which again produces floating point notation.

The column output may be divided up if necessary so that some columns are printed out individually and some as groups. Similarly some may be in fixed point and others in floating.

Constants are printed by means of the codeword

Na b - 28

which prints a consecutive string of b constants beginning with Na.

Example 6

Evaluate $y = 1 - \sin\left[\sin^{-1}\left\{ ax^{\frac{1}{3}} \left(\frac{\cos\theta}{\cos\phi} - 1\right)^{\frac{1}{2}} \right\} + \lambda\right]$

for given values of the constants a and ϕ and a set of values of x, θ and λ.

Codeword				Interpretation
3	0	1	29	Read x, θ and λ into columns 1, 2 and 3 respectively
1	0	N32	29	" $\frac{1}{3}$, a and ϕ " N32, N33 and N34 respectively
2	0	2	12	Form cos θ col.2.
N34	-	N34	12	" cos ϕ in N34
2	N34	2	1	" cos θ/cos ϕ = S in col. 2
2	N1	2	3	" S - 1 in col. 2
2	0	2	6	$(S - 1)^{\frac{1}{2}}$ in col. 2
1	0	1	8	" log x in col. 1
N32	1	1	0	" $\frac{1}{3}$ log x in col. 1
1	0	1	9	" $x^{\frac{1}{3}}$ in col. 1
N33	1	1	0	" $ax^{\frac{1}{3}}$ in col. 1
1	2	1	0	" $ax^{\frac{1}{3}}(S - 1)^{\frac{1}{2}}$ in col. 1
1	0	1	23	" $\sin^{-1}(ax^{\frac{1}{3}}(S - 1)^{\frac{1}{2}})$ in col. 1
1	3	1	2	" $\sin^{-1}(ax^{\frac{1}{3}}(S - 1)^{\frac{1}{2}}) + \lambda = t$ in col. 1
1	0	1	11	" sin t in col. 1
N1	1	1	3	" 1 - sin t = y in col. 1
1	1	0	28	Print one column (col. 1) in fixed point form
0	0	0	31	End of programme

The data for this problem would need to be presented on a sheet set out as illustrated below:

Sample Data Sheet

Programme No. . . . . Name . . . . . .

Row No.	x	θ	λ	Constant	Value
0				$\frac{1}{3}$	0.3333
1				a	
2				ϕ	
3					
4					
5					
6					
7					
8					
9					
10					
11					

In setting out the data sheet it is important that the columns and constants should appear in the same order as the read instructions read them into the computer.

P A R T I I

For many applications the T.I.P. functions described in Part I will suffice. However, more powerful and flexible programmes require some kind of instruction modification facility. Such a facility enables a set of instructions to be repeated many times or enables alternative routes to be taken through a calculation according to the results obtained at various stages. Part II of this manual is therefore devoted to these slightly more complex operations and to consideration of the codewords controlling them.

Codewords are normally obeyed in the sequence in which they are written. Frequently it is necessary to break out of sequence so that, for example, codeword n leads to codeword m instead of n + 1. This "jump" may be used to route a programme to two alternative sequences (one beginning with codeword m and the other with n+1) the choice being made dependent upon the satisfaction of some condition, in which case the jump is said to be conditional, or codeword n may be made always to lead to codeword m, this being referred to as an unconditional jump.

Any instruction which refers to another instruction must, of course, identify the instruction to which it refers. This could be done by quoting the number of the instruction, assuming they have all been numbered in sequence. The main disadvantage of doing this is that insertions or deletions arising from a programme alteration change the numbers of all the following codewords, this in turn necessitating alterations to instructions referring to those codewords.

This difficulty is removed by having an optional fifth element on the codeword - the codeword's reference number (designated by a number in the range 1-63 prefixed with the letter R). In this way reference numbers need only be assigned to codewords referred to by other codewords. If, for example, an instruction reads "jump to R1" then the codeword to which it refers must be labelled R1.

Unconditional Jump

If codeword n contains the instruction

- - Rc 19

the programme will ignore codeword n + 1 which would normally be the next to be obeyed and obey instead the sequence of instructions which starts with the codeword labelled Rc.

Example 7

Given 20 values of x, (x_0 to x_{19}) in column 2 and m values of y, (y_0 to y_{m-1}) in column 3, compute and print the m values of the expression

$$Z = \sum_{n=0}^{n=19} x_n(x_n - y)$$

Codeword						Description
No.	a	b	c	r	R	
0	3	0	N33	13	1	Put y_0 in N33
1	2	N33	4	3		$x - y_0$
2	2	4	4	0		$x(x-y_0)$
3	4	-	4	15		Summate
4	4	19	N33	13		$\sum_{n=0}^{n=19}$ selected from row No.19 = Z_0
5	N33	1	-	28		Print Z_0
6	3	0	3	10		Shift up col. 3 (y_1 now at top)
7	-	-	R1	19		Jump back to reference 1.
8	-	-	-	31		End of programme.

In the last example codeword 7 is an instruction to jump back to the codeword labelled R1. This will cause the programme to recalculate Z for the next value of y and continue in this manner until all values of y have been dealt with. However, from the programme as given the computer has no information on when to stop this process and will in fact continue to try to calculate Z for non-existent y until stopped manually. What is necessary here is a conditional rather than an unconditional jump, that is the jump should be subject to the condition that the required number of values of Z have not been computed, the jump being ignored when they have. The conditional jump has two principal uses. One enables a section of programme to be repeated a definite number of times (a counted loop) the other enabling repetition an indefinite number of times (iterative loop).

Counted loops.

If instruction n reads

- b Rc 16

the programme will jump back to reference c each time it comes to codeword n until it has come to it b times when it will then resume the normal sequence by going to codeword n + 1.

The effect of this instruction can be illustrated by replacing codeword 7 of the last example by

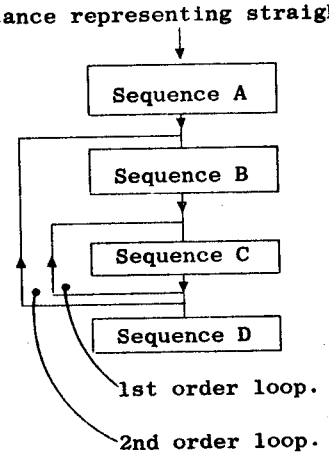
- m R1 16

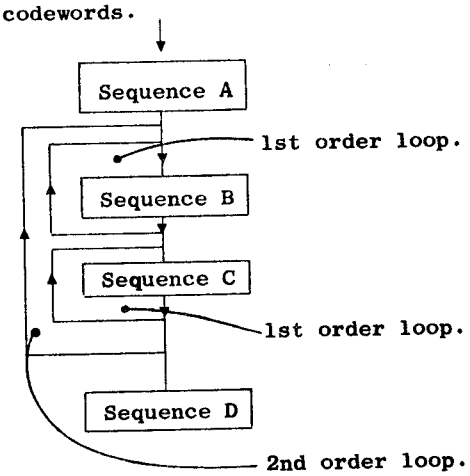
On reaching this instruction the programme would, as before, return to reference 1 and begin computing Z for the next value of y, but on reaching codeword 7 for the mth time (i.e. after having calculated m values of Z) it would move on to codeword 8. Of course m cannot remain in the codeword as an algebraic symbol. If at the time of writing the programme the value of m is known, the value can be written in place of the symbol m. If not the value of m can be read into a constant store along with the data, and the instruction can be modified to read

- Nb R1 16

the number of repetitions now being given by the number stored in Nb.

Any group of instructions which is automatically repeated is said to be an instruction loop. If the loop is obeyed a fixed number of times it is called a counted loop. Codewords 0 to 7 of the last example are an instance of a counted loop. A loop containing no other loops is defined as a first order loop. A second order loop will by the same definition be one containing one or more first order loops. The block flow diagrams below illustrate this point, the blocks in this instance representing straight sequences of codewords.





The number of counting loops that may be used in T.I.P. programmes is effectively unlimited, but none must exceed 4th order.

Iterative loops.

When using iterative techniques in numerical work the number of repetitions of certain loops will not be known in advance. In these cases repetition must continue until two sets of results converge, within a specified tolerance, to a common set of values. For this purpose the instruction

a Nb Rc 17

stored as codeword n, will cause the programme to loop back to reference c and iterate until the numbers in columns a and a + 1 are within the percentage tolerance given in Nb, when it will go on to codeword n + 1. If the tolerance is to be 2.0%, Nb should contain 2.0 not 0.02.

Instruction modification.

When using counted loops it is often desirable to be able to alter some of the instructions slightly each time they are obeyed in the loop. In example 7 a shift instruction (codeword 6) was used to feed successive values of y into position to be stored in N33 by codeword 0. However, another and better method is to add one to the b element of codeword 0 after obeying it, so that on the first occasion it will cause row 0 to be moved to N33, on the next occasion move row 1 to N33 and so on. Any of the first three elements of a codeword may be modified in this way by placing an asterisk alongside the appropriate element. After the instruction has been obeyed the asterisk will cause the element to be increased by one. For example the instruction

N35* 7 9* 0

will be obeyed as written, but the second time this instruction is reached it will read

N36* 7 10* 0

and on the third occasion

N37* 7 11* 0 etc.

It should be noted that the asterisk must not be used in control codewords, that is codewords with operation numbers 16 to 21.

Reset

On exit from a loop containing asterisked instructions it is usually necessary to reset the modified parts of the loop thus leaving it in a state of readiness should it be required to use the loop again in the same programme. This would be essential for example, with a first order loop contained within a second order loop. Any asterisked instructions in the first order loop would be left in their modified form on exit, hence in the second time round the second order loop (i.e. at the second entry of the first order loop) they would be incorrect. The reset instruction.

- - - 18

will reset all asterisked instructions in a loop to their original form, but several points about it should be noted:-

(1) If the looping instruction is codeword n the reset instruction must be codeword n + 1.

(2) If reset is used after a second or higher order loop, asterisked instructions in the lower order loops will be ignored. (The lower order loops should have their own reset instructions).

(3) If the looping instruction is codeword n jumping back to Rc, only codewords from the one labelled Rc to codeword n will be reset. Any instructions which are out of sequence (i.e. obeyed by using jump instructions) will not be reset.

(4) If Nb is used to store the counter for a counted loop, it must not be altered or disturbed in any way until after the reset instruction has been obeyed.

(5) If any asterisked part of an instruction is obeyed as 127* (or N127*) it will be changed to 128* after obeying the instruction once more and in this condition cannot be reset as store 128 does not exist. N127* should be avoided altogether if possible as it has special significance in some circumstances.

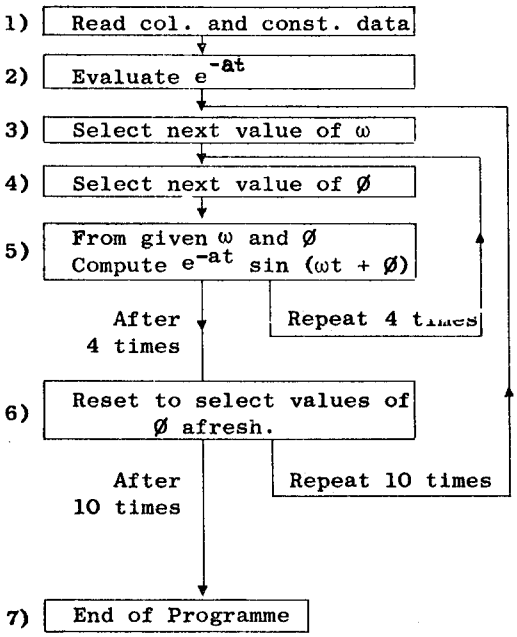
Example 8

Compute a table of values of the expression

$$y = e^{-at} \sin(\omega t + \phi)$$

for 25 values of t, 10 of ω and 4 of ϕ.

Brief consideration shows that this may be conveniently arranged on T.I.P. by computing for the 25 values of t simultaneously. Hence t will appear as column data. A flow diagram can now be drawn.



It is instructive to note certain points in connection with this flow diagram. The inner loop (1st order loop consisting of blocks 4 and 5) will be obeyed 40 times in all, therefore no instructions should appear in this loop that could be placed outside it. For this reason the evaluation of e^{-at} which is independent of ω and t is placed in block 2, outside the loops. The storage of results from block 5 must be made in successive columns. From the flow diagram it will be evident that these columns will appear as 10 groups of 4, one group for each ω, each of the four columns within a group being for a given ϕ. On initial entry to blocks 3 and 4 the word "next" will be interpreted as "first". A suitable T.I.P. programme would be:-

	Codewords				Description
a	b	c	r	R	
1	-	0	29		Read 25 values of t into col.0.
15	-	N33	29		Read constants -a, ϕ_1 to ϕ_4, ω_1 to ω_{10} into N33 onwards.
N33	0	1	0		- at
1	0	1	9		e^{-at} in col.1.
N38*	0	2	0	2	$\omega_n t$ in col.2 (starting with $\omega_1 t$)
N34*	2	2	2	1	$\omega_n t + \phi_m$ in col.2 (starting with ϕ_1)
2	0	2	11		$\sin(\omega_n t + \phi_m)$
1	2	3*	0		y_1 in col.3, y_2 in col.4 etc.
-	4	R1	16		Repeat from ref.1, 4 times (4 values of ϕ)
-	-	-	18		Reset (ready to begin again with ϕ_1)
3	4	1	28		Print 4 columns starting with col.3 (Floating point)
-	10	R2	16		Repeat from ref.2, 10 times (10 values of ω)
-	-	-	31		End of programme.

Sub Routines

Sometimes a certain sequence of instructions will be used more than once in the same programme. For example, it may be necessary to find the hyperbolic sine and cosine of the elements in a column at several different points in the programme. This would mean writing the same group of instructions at several points in the programme. Effort is saved, and the risk of errors lessened, if this group of instructions appears once only and is entered by the main programme on the required occasions by something like a jump instruction, with another rather specialised jump instruction at the end of the group to enable the main programme sequence to be resumed. The "enter subroutine instruction"

\- - Rc 20

causes the programme to jump to the codeword labelled Rc and continue from that point until the "end of subroutine" instruction

\- - - 21

is reached. This indicates the end of the subroutine and causes the programme to jump back to the instruction following the last "enter subroutine" codeword. A subroutine will of course refer to specific columns for data and results and since it is to be used more than once data will need to be moved to the required positions before the subroutine is entered. Similarly the results from a subroutine must be moved before the subroutine is entered a second time if overwriting is to be avoided. The following example will illustrate this point. No subroutine must exceed seventh order (i.e. must not contain any subroutine of order higher than six).

Example 9

Consider a programme in which it is required to calculate sinh x and cosh x for given values of x on more than one occasion in the programme. A group of instructions to do this might be

Codeword				Description
127	-	127	9	form e^x
N1	127	126	1	" e^{-x}
126	127	127	2	$e^x + e^{-x}$
127	N2	127	1	$\cosh x = \frac{1}{2}(e^x + e^{-x})$
127	126	126	3	$\sinh x = \frac{1}{2}(e^x - e^{-x})$

Given x in column 127 these instructions will form sinh x and cosh x in columns 126 and 127. In order to use these instructions as a subroutine it is necessary to add an end of subroutine instruction. The main programme would then have the following form.

No.	a	b	c	r	R	
0	a_0	b_0	c_0	r_0		)First part of programme and formation of x.
	.	.	.	.		)
	.	.	.	.		)
	.	.	.	.		)
n-1	a_{n-1}	b_{n-1}	c_{n-1}	r_{n-1}		)
n	a_n	-	127	13		Transfer x to column 127
n+1	-	-	Rc	20		Enter subroutine (beginning at ref.c)
n+2	a_{n+2}	b_{n+2}	c_{n+2}	r_{n+2}		)Next stage of programme, including transfer of cols.126
	.	.	.	.		)and 127 if necessary and formation of another set of x
	.	.	.	.		)values.
	.	.	.	.		)
m-1	a_{m-1}	b_{m-1}	c_{m-1}	r_{m-1}		)
m	a_m	-	127	13		Transfer x to column 127
m+1	-	-	Rc	20		Enter subroutine
m+2	a_{m+2}	b_{m+2}	c_{m+2}	r_{m+2}		)Remainder of programme.
	.	.	.	.		)
	.	.	.	.		)
p	-	-	-	31		End of programme.
p+1	127	-	127	9	c	)
p+2	N1	127	126	1		)Subroutine for sinh x.
p+3	126	127	127	2		)and cosh x.
p+4	127	N2	127	1		)
p+5	127	126	126	3		)
p+6	-	-	-	21		End of subroutine.

On reaching instruction n+1 the programme will jump to reference c and continue from there to the end of subroutine instruction which will initiate a jump back to n+2. The normal sequence will then continue down to m+1 at which stage a jump is again made to codeword p+1 (reference c), the

first instruction of the subroutine. After obeying the subroutine sequence a second time the end of subroutine instruction will route the programme back to codeword m+2 and from there it will proceed through the final section to the end of programme instruction (p).

Branching and Mixing

In some problems the operations to be performed on certain quantities will depend on their numerical value. In these instances it is necessary for the computer to examine individually the numbers in a column, split them into two or more groups and operate on the groups separately, probably recombining them into one column at a later stage. The codeword

a b c 25

will cause each element of column b to be examined in turn and if positive put the corresponding row of column a into column c with 'dashes' in all rows of column c for which column b is negative. Similarly into column c+1 will be put all rows of column a for which the equivalent rows of column b are negative, with dashes where they are positive. The effect of this instruction is illustrated below.

Data		Result	
a	b	c	c + 1
a_0	b_0	a_0	-
a_1	$-b_1$	-	a_1
a_2	$-b_2$	-	a_2
a_3	b_3	a_3	-
a_4	b_4	a_4	-
a_5	$-b_5$	-	a_5
a_6	b_6	a_6	-

The resulting columns can be operated on separately and then if necessary combined into a single column by the instruction

a b c 26

If a and b are the two columns being combined then the dashes in column a are replaced by the corresponding elements of column b, this merged result being put in column c.

If columns a and b are of different lengths, column c will be made the length of the longer one, but care must be exercised here since if col.b is longer than col. a the extra rows of col.a which will be examined will appear to contain zeros, not dashes, and these zeros will therefore be transferred to column c.

Example 10.

Suppose that in the course of a calculation it is required to find sin θ for $0 \leqslant \theta < \pi/4$ and cos θ for $\pi/4 < \theta \leqslant \pi/2$ where $\theta (0 \leqslant \theta \leqslant \pi/2)$ is given in column 0 and $\pi/4$ in N32. A section of programme showing how this may be done by using the Branch and Mix instructions is given below

	Codeword			Description
N32	0	1	3	$\pi/4 - \theta$
0	1	2	25	If $\pi/4 \geqslant \theta$ put θ col.2, if $< \theta$ put θ in col.3
2	-	2	11	sin θ for $\pi/4 \geqslant \theta$
3	-	3	12	cos θ for $\pi/4 < \theta$
2	3	1	26	Mix sin θ and cos θ

Interpolation on a three dimensional Graph.

In addition to the ordinary interpolation for functions of a single variable a T.I.P. operation is available which will perform linear interpolation on a carpet of curves representing a function of two variables. The values of the function f(x,y) must be given at the nodal points of a rectangular grid at equal increments of x and y. The data required to define the carpet is as follows:-

1) m = number of values of x at which the function is given.

2) n = " " " " y " " " " " " "

3) x_0) Minimum values of x and y.
4) y_0)

5) Δx) Incremental values of x and y.
6) Δy)

7) x_m) Maximum values of x and y.
8) y_m)

9) $f(x_0,y_0)$, $f(x_0,y_1)$,.....$f(x_0,y_{n-1})$, $f(x_1,y_0)$, $f(x_1,y_1)$....$f(x_1,y_{n-1})$....$f(x_{m-1},y_{n-1})$

Usually more than one column of the data store will be needed to contain the carpet. The number of columns required will be (mn + 8) /32, rounded up to the nearest integer. The carpet may be read by referring only to the number of the first column of those in which it is stored, but the other columns must not be overwritten by the programme. Thus if columns a and a+1 contain the values of x and y respectively for which the function is required, and column b is the first of the columns in which the carpet is stored then the instruction.

a b c 27

will form f(x,y) in column c. If the column containing y (col a+1) has been formed by the preceding codeword it must not be overwritten by the results of operation 27 (i.e. if the interpolation codeword is numbered n, then c_{n-1} = $(a+1)_n$ = c_n is not permissible, though any two of the three may be equal). As the data for this operation is stored in a non-standard form, a special instruction is required to read it into the machine. This is

0 - c 29

which will read a single carpet of curves into column c onwards, a = o distinguishing it from ordinary read instructions.

DEUCE LIBRARY SERVICE,
ENGLISH ELECTRIC CO. LTD., KIDSGROVE.

DEUCE News No.: 65
Report No.: K/AA y 34
Sheet No.: 17

Short Guide to T.I.P. Order Code.

Codeword.				Description
a	b	c	0	a x b ⟶ c
a	b	c	1	a ÷ b ⟶ c
a	b	c	2	a + b ⟶ c
a	b	c	3	a - b ⟶ c
a	-	c	6	$\sqrt{a}$ ⟶ c
a	Nb	c	7	Eval. y = $ax^n + bx^{n-1}$ ....where x in col.a, n in Nb, a in N(b + 1) etc.
a	-	c	8	log a ⟶ c
a	-	c	9	e^a ⟶ c
a	o	c	10	a shifted up one place ⟶ c
a	Nb	c	10	a shifted down one place ⟶ c with Nb in row o of c.
a	-	c	11	sin a ⟶ c
a	-	c	12	cos a ⟶ c
a	-	c	13	Move a ⟶ c
a	b	Nc	13	Move row b of a ⟶ Nc
Na	b	c	13	Na ⟶ row b of c
a	-	c	14	| a | ⟶ c
a	-	c	15	Σ a ⟶ c
-	b	Rc	16	Loop back to ref.c until instr. has been reached b times
-	Nb	Rc	16	Loop back to ref.c until instr. has been reached Nb times
a	Nb	Rc	17	Loop back to ref.c until a and (a+1) are within tolerance given in Nb.
-	-	-	18	Reset asterisked codes in counting loop.
-	-	Rc	19	Jump to ref.c.
-	-	Rc	20	Jump to subroutine starting at reference c.
-	-	-	21	End of subroutine.
a	b	c	22	With data from column a read graph stored in b and b + 1 putting the linearly interpolated results in col.c.
a	-	c	23	$\sin^{-1}$ a ⟶ c
a	-	c	24	$\cos^{-1}$ a ⟶ c
a	b	c	25	if(b is + ve put a ⟶ c (b is - ve put a ⟶ c + 1.
a	b	c	26	Mix a and b ⟶ c
a	b	c	27	Linear interpolation on a function of two variables.
a	b	(c)	28	Print b cols. from a onwards.
Na	b	-	28	Print b constants from Na onwards
a	-	c	29	Read a columns into c onwards.
-	-	Nc	29	Read string of constants into N_c onwards.
o	-	c	29	Read graph of function of two variables.
-	-	-	31	End of programme.

PART III. OPERATING INSTRUCTIONS.

26T - The Compiler.

Input.

Program cards 0-67
Decimal codewords
Blank card

Decimal Codewords.

These are punched one per card with 3-digit integers,

a in DEUCE cols. 2- 4
b in DEUCE cols. 7- 9
c in DEUCE cols. 12-14
r in DEUCE cols. 17-19
R in DEUCE cols. 22-24

The letter R is not punched on the cards. If a codeword has no reference then cols. 22-24 must be left blank.

If Na, Nb or Nc are required then the N is represented by a punching in the X-row in col. 1, 6 or 11.
Asterisks are represented by punchings in the Y-row in cols. 5, 10 or 15.

Operation.

The compiler reads in the codewords stopping to punch out failure cards for any codeword in which a, b, c, r or R exceeds the permitted value. When all the cards have been read, if any failures occurred or if more than 512 codewords have been presented, the machine will stop on 1-1 X. A S.S. will return the compiler to the beginning to read the next program.
If no failures occurred during reading the compiler carries out further checks. If these produce no failure cards or if the failures which occur can be corrected by the compiler to the form which was probably intended, the binary codewords will be punched and the machine will stop on 2-24 X. If the failures are such that the program cannot be run the compiler stops on 2-2 X and a S.S. will cause it to return to read the next program.
At 2-24 X a S.S. without TIL will return to read the next program but a S.S. with TIL will punch out a decimal version of the current program in a form convenient for printing, before returning to read the next one.

Failures.

6, 12-24 X. Non decimally punched card. S.S. to re-read corrected card.

One failure card is punched for each codeword which fails to pass each test, the card containing the codeword but with only the parts of the codeword which are incompatible with the function number. They are punched as 3-digit integers,

codeword no. in DEUCE cols. 2- 4
a in DEUCE cols. 7- 9
b in DEUCE cols. 12-14
c in DEUCE cols. 17-19
r in DEUCE cols. 22-24
R in DEUCE cols. 27-29

N's are punched in cols. 6, 11 and 16 and asterisks (in the 2-row for the card operated typewriter) in cols. 10, 15 and 20. If c is a reference (Rc) the R is punched in col. 16.

OPERATING INSTRUCTIONS. (CONT'D)

Failures (Cont'd)

There is also another type of failure card which indicates one of 4 possible errors:-

(i) n Rm does not exist. Codeword n calls on reference Rm which does not exist in this program.

(ii) n not preceded by r=16. This occurs when r=18 in codeword n and r $\neq$ 16 in codeword (n-1)

(iii) n b is not 3-D graph. This occurs when r=27 in codeword n and column b has not been read in the special form required for a 3-D graph. Not more than 31 3-D graphs are allowed.

(iv) n m has same reference. Codewords n and m have the same reference number.

All failure cards are punched with instructions for the card operated typewriter.

Decimal Output.

In addition to the binary codewords a decimal version can be punched, see operation above. This decimal version is in the same form as the failure cards.

General.

In addition to changes (from T.I.P. 2) in input and output, codewords and the use of reference numbers, some of the other functions have been compacted in the binary form in T.I.P. 3 and this has left function numbers 4, 5, 9, 12, 15, 20, 21, 30 and 31 free for special functions. These binary operation numbers will be assigned by the compiler to the decimally numbered operations 32 to 40 respectively and the only checks made on them will be to see that a, b, c and R do not exceed the permitted size, with the exception of functions 20 and 21 (converted from 37 and 38) for which c will be assumed to be Rc and will be converted and checked accordingly. The use of functions 4, 5 and 30 in the decimal instructions will be treated as failures.

ZC27T - The Interpreter.

Input.

Program cards 0-160.
Binary codewords from ZC26T or ZC29T.
Data.

Data.

All input data for and output data from the Interpreter is in binary and can be converted using LK15T and LK16T.
In the binary form the parameter card contains

Y-row m P_{17} (m = no. of cols. in block)

X-row n P_{17} block floating) n = no. of elements in a
2n P_{17} fully floating) T.I.P. column.

0-row p P_{17} (p = no. of b.p.)

1-row P_{17} (row sum shift = 1)

2-row { zero if block floating
non-zero if fully floating

A block of column data may be either a matrix or a vector (each row of a matrix becomes a T.I.P. column) and may be fully or block floated. A block of constants must be a fully floating vector.

Note that in the codeword a-c, 29 "a" is the number of matrices to be read which is not necessarily the number of T.I.P. columns.

The data for a 3-D graph must be read into the Interpreter as a 6 element block floating vector followed by a block floating matrix (not necessarily to the same number of b.p.). The 6 elements of the vector are x_o, y_o, Δx, Δy, x_{m-1}, y_{m-1} and the matrix consists of the values of f(x, y) punched in the following order:-

1st row $f(x_o\ y_o)$, $f(x_o\ y_1)$ .... $f(x_o\ y_{n-1})$

2nd row $f(x_1\ y_o)$, $f(x_1\ y_1)$ .... $f(x_1\ y_{n-1})$

⋮

mth row $f(x_{m-1}\ y_o)$, $f(x_{m-1}\ y_1)$ .... $f(x_{m-1}\ y_{n-1})$

Output.

See "Data" above.

Failures.

2, 13-29 X	n > 30. S.S. leads to next codeword.	
2, 22-29 X	Column row sum wrong)	S.S. without TIL leads to next
1, 23-29 X	Constant row sum wrong)	codeword. S.S. with TIL to re-read corrected row.
1, 23-29 X	3-D graph row sum wrong)	S.S. without TIL leads to
1, 3- 3 X	x_m and/or y_m do not agree with calculated values)	next codeword. S.S. with TIL to re-read corrected vector and matrix.

Failures. (Cont'd)

2, 31-14 X	A result is $\geqslant 2^{62}$	S.S. will insert a dash in the result for the offending row and continue.
3, 31-21 X	A divisor is zero	
2, 31-15 X	$a < 0$ in square root	
3, 31-13 X	$a \leq 0$ in log.	
3, 31-14 X	$e^a \geqslant 2^{62}$	
6, 31-14 X	lin. interpolation Extrapolation is	
6, 31-15 X	3-D interpolation being attempted.	
1, 31-14 X	$a > 1$ in $\sin^{-1}$ or $\cos^{-1}$	
1, 2- 2 X	Shift - the col. to be shifted up contains only one element.	S.S. leads to next codeword.
2, 1- 1 X	Counting loop and reset - N_b has no integral part	
0, 15-14 X	Wrong track read i.e. no. times P_1 in TS15 and number in track which has been brought down do not agree. S.S. re-reads track whose no. is in TS15. This failure will also be caused by trying to read a blank column, other than column 0, except in the transfer operation where it is permissible to transfer a constant to a blank column. A blank column 0 will be accepted by the magnetic routine but will cause most bricks to go into a loop.	
0, 13-29 X	(followed by 0, 15-14X) Sum check failure. 2 S.S.'s will re-read the track whose no. is in TS15.	
2, 21-29 X	Square root result is negative.	S.S. will insert a dash in offending row and continue.
2, 13-29 X	Square root, power of result is too large or too small.	
1, 21-29 X	$\text{Log}_e a >$ possible maximum	

Testing Facilities.

P_{32} on I.D. will stop program on every codeword. Stops on 1, 8-29 X displaying codeword on 0,S. The number of the next codeword times P_1 will be found in 12_{30} at this stage and throughout the brick. Since no distinction is made between function numbers the failure 0,15-14 X is liable to occur if c is an unused track. ZC28T can then be used to restart.

n P_{17} on I.D. will stop program before obeying codeword n.

P_{31} on I.D. will punch column c after every codeword.

n P_{17} + P_{31} on I.D. will stop program on codeword n and then punch column c after obeying it.

Columns punched out will all be block floating and constants fully floating.

To insert a codeword with machine stopped on any instruction.

Set codeword to be inserted on I.D.
Put TIL on and S.S.
Put TIL off and S.S.

This will obey the codeword set on the I.D. and then return to obey the codeword which it replaced.

Restore Control. See program ZC28T on page 23.

Program Details.

The interpreter uses a form of semi-floating arithmetic in which the mantissa is the top 27 bits and the characteristic the bottom 5 bits of a word. The characteristic is such that the exponent = 4 times characteristic minus 62 and the mantissa has its binary point between P_{30} and P_{31}. Since the characteristic can take the values 0-31, numbers are restricted to between $\pm 2^{62}$. The most significant digit of the mantissa will be in the P_{27}, P_{28}, P_{29} or P_{30} position.

In order to reduce the number of magnetic transfers the interpretive routine checks each codeword (n) before it is obeyed, to see, firstly whether it uses a_{n-1} or c_{n-1} which are still available in the D.L's (b_{n-1} is overwritten by c_{n-1}) and, secondly, whether the result, c_n, will be immediately overwritten by c_{n+1}, in which case it will not be stored on the drum. Checks are included in routines where c_n is not a column or constant number, to make sure that c_{n-1} is written on the drum. However a difficulty arises if restore control has to be used, as the results for the current codeword may have been partly formed and have overwritten c_{n-1}, which is not necessarily still available on the drum. The operator must decide, after restoring control, whether the program may safely be continued from the point of failure or whether it is necessary to go back to an earlier codeword to recalculate results which may have been lost.

When columns are brought down from the drum m.c's 0-29 contain the 30 rows of data, m.c. 30 contains $n\,P_1 + TP_{17}$ (where n = no. of rows and T is the column number) and m.c. 31 contains a negative sum check of the other 31 m.c's. If the column has been left by the previous operation, T and the sum check may not be present. A dash is represented by P_{1-32}.

ZC28T - Restore Control.

Input. Program cards 0-2.

This must be run in using the "Standard Post Mortem Procedure," i.e. as follows:-

Stop machine
Depress the Ext. Tree key
Set 30-0, 1 on I.S. keys
Give a S.S.
Set Ext. Tree key to central position
Put the machine to Normal
Press the READ key
Run in ZC28T

The program will stop on 1, 2-24 X displaying the number of the codeword which was being obeyed times P_1 on O.S. A S.S. without TIL will go back to repeat this codeword; a S.S. with TIL will go to 1, 0-13 X and another S.S. without TIL will go to the codeword whose no. is set times P_1 on the I.D.

Note. It is advisable, when restoring control, to go back to an earlier codeword than the one on which the failure occured. See "Program Details for the Interpreter" on page 22

ZC29T - T.I.P.2 to T.I.P.3 Converter.

Input.

Program cards 0-9
Parameter card with kP_{17} on Y-row
k triads of T.I.P.2 codewords.

Output.

T.I.P.3 binary codewords.
These will be ready to read in to the T.I.P.3 Interpreter except for any codewords to read in data for 3-D interpolation. This codeword cannot be produced automatically through the conversion due to an ambiguity in the T.I.P.2 binary read instruction and must be altered in the T.I.P.3 codewords so that a = 0. It is also possible in T.I.P.2 to read more than one 3-D graph with one codeword and this must be altered in the T.I.P.3 version.

Failure.

1, 1-1 X r = 30. This has no equivalent in T.I.P.3.

General Description.

The differences in the binary codewords of T.I.P.2 and T.I.P.3 are listed below.

T.I.P.2.	T.I.P.3 binary.	Notes.
- - c 4 a - c 4	1 - c 29	(i)
a - Nc 4	1 - Nc 29	
a - - 5 a b - 5	a 1 0 28	(ii)
Na - - 5	Na 1 - 28	(iii)
a - c 9	a 1 c 8	
a - c 12	a 1 c 11	
a - c 15	a 1 c 14	
- - c 20	- 1 c 19	
- - - 21	- 2 - 19	
a - c 24	a 1 c 23	
- - c 29 1 - c 29	1 - c 29	
a - c 29	a - c 29	(iv)
- - - 31	- - 1 18	

(i) a indicates the no. of elements to be read into a column in T.I.P.2; this is done by the matrix parameter card in T.I.P.3.

(ii) b indicates the number of rows to be punched out in T.I.P.2 but T.I.P. always punches the complete column. c=0 as the col. would be stored in block floating form.

(iii) This codeword will punch out one constant in both T.I.P.2 and T.I.P.3 but a group of constants is better punched out as one block in T.I.P.3.

(iv) This is the codeword used in T.I.P.2 to read 3-D graphical data but, as it may also be used to read a number of cols. of ordinary binary data, it is left in this form in T.I.P.3. If it was being used to read a 3-D graph it must be altered to 0 - c 29 in T.I.P.3. (Data for only one 3-D graph may be read by one instruction in T.I.P.3.)

As in T.I.P.2 a, b and c are each 9 binary digits and r 5 binary digits but the 9 digits are sub-divided as follows

1st digit	2nd to 8th digits	9th digit
asterisk	no. of col. or constant	N

When c is a codeword no. (i.e. when r = 16, 17 or 19) it uses all 9 digits.