EASICODE

The two main reports known to have been written on EASICODE are:

EASICODE, Kelly, M. J. and Randell, B.
Technical Report No: W/AT 585, Atomic Power Division,
English Electric Co., Whetstone, Leics., October 1960

Multi-Section EASICODE, Randell, B.
Technical Report No: W/AT 603, Atomic Power Division,
English Electric Co., Whetstone, Leics., November 1960

Though no copies of these actual reports can now be traced, they were reprinted with only very minor editing in DEUCE News, No. 68 (October 1961). This also includes a section describing the EASICODE Subroutine Library - it is not now known whether this was originally part of Report W/AT 585 or was a separate W/AT Report.

The present PDF version of DEUCE News No 68 was produced from scans of the pages of an original copy kindly provided by David Green.

Brian Randell, March 2007

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

DEUCE News No.: 68
Report No.: K/AA y 40
Sheet No.: 1

DEUCE News No. 68. October, 1961.

Easicode.

SUMMARY.

EASICODE is a fixed point translation program, written by English Electric, Whetstone, intended for use in most cases where a fixed point DEUCE program could be written. EASICODE would be used in preference to conventional programming when the urgency of the problem, the probability of later alterations being required, or the limited amount of use envisaged outweigh the inevitable loss of speed of the resulting program.

In addition to the method of translation, EASICODE gains in efficiency compared to existing interpretive schemes, because working is in fixed point, and because organisation of the two-level storage of data is left to the programmer. It has been found that the somewhat increased difficulty in programming, compared with floating point interpretive schemes, is more than compensated for by the greatly increased speed and efficiency of program testing and production running of EASICODE programs.

The report consists of the original Whetstone reports with minor editing.

The DEUCE Library numbers for Easicode are ZC30T, No. 715.

CONTENTS.

Contents of Part I. — Sheet No.

Contents of Part II. Sheet No.

Contents of Part III. Sheet No.

PART I.

1. INTRODUCTION.

Experience at the Atomic Power Division, Whetstone, in developing programs, required urgently, for use on a heavily used computer, has shown the need for an automatic programming system to bridge the gap between normal fixed point **DEUCE** programming and the extremely useful **ALPHACODE** and **ALPHACODE TRANSLATOR** system.

Experience has shown that, in the field of problems for which it was intended, considerable savings of programmers' and of machine time can be made by using this system. Naturally however, for one off jobs, or for those few jobs which are not suitable for fixed point working, the **ALPHACODE** system is to be preferred. In other cases, for very long running programs, it has been essential to use conventional programming to bring the time taken for a task down to reasonable limits. Even in some of these cases **EASICODE** may be of use, for already some programs have been written, which use **EASICODE** as a sub-routine for the more rarely used parts of the computation.

2. OUTLINE.

The **EASICODE** translator is a **DEUCE** program which allows programmers to use **DEUCE** as a fixed point 3-address serial instruction computer, with two level data storage, and verful instruction modification facilities.

The purpose of this report is to describe how to program and operate this fictitious **EASICODE** computer, and also how to operate the translator program which allows **DEUCE** to be used in this way.

Due to the boundless variety of functions which might, fairly reasonably, be required in a programming system of this kind, it was decided that the standard order code of **EASICODE** should only include the basic arithmetic functions, housekeeping and input-output orders etc. This order code may be extended by any user, by choosing such extra functions as are required from the library of **EASICODE** subroutines, and reading these subroutines in with the input orders.

Fixed point working is governed by the convention that all numbers must be of modulus < 1, and to this end scaling is permitted by factors of 2 or 10, on all arithmetic orders. Furthermore, any operation which would produce a result out of the permitted range, has its result validity checked.

Input and output is catered for by a wide range of orders, all of which however conform to the **A.P.D.** Computing Office standard fixed point punching conventions, namely up to six nine-digit signed numbers per card.

All communication with the **EASICODE** computer is in decimal, including operation during program testing and production runs. This, it is claimed, makes for increased efficiency of computer usage, even by persons skilled in the binary notation.

3. STORAGE.

The **EASICODE** data storage is on two levels; the working or high-speed storage, and the backing or drum storage.

The working storage consists of 6 blocks of 32 positions. The blocks are labelled **A, B, ... F,** and the positions within these blocks 01, 02, ... 32. Thus, C05 is the fifth position in block C. Storage in blocks is cyclic, so position C01 is immediately following C32, just as C05 follows C04.

There is a convention of using 00 to refer to the whole of the block, thus D00 means the complete set of 32 numbers in D.

This working storage is used for numbers of modulus < 1 and for integers used in counting and order modification, which must be of modulus < 32768. In addition, if functions not in the basic order code are needed, then the extra subroutines will, when required, occupy one or more of blocks A, B, C and D.

Drum storage is in tracks of 32 positions numbered from 001 to (176-n) where n is the number of extra subroutine tracks. The maximum number of subroutine tracks is 16, so there is drum storage for over 5000 numbers.

4. ORDERS.

4.1 Basic Form.

EASICODE programs are essentially made up of a set of 3-address orders, obeyed serially.

Thus, an order is basically of the form

$$F(X,Y) \longrightarrow Z$$

i.e. quantities denoted by X and Y are operated on by function F, the result being denoted by Z.

Normally X, Y and Z will be given as positions in the high-speed storage. F is a two digit number in the range 01 to 63, where the first 31 functions form the basic order code, and the remaining ones are for use with extra functions obtained from the subroutine library.

Ex.	F	X	Y	Z
	01	A03	B05	C17

F = 01 is the function 'Multiply' so this order causes the number in position 3 of block A to be multiplied by the number in position 5 of block B and the result to be placed in block C, position 17.

Not all functions use all of X, Y and Z.

Thus, with F = 25 'Read one number'

the order

F	X	Y	Z
25			B02

will cause the program to read one number and place it in position 2 of block B, whilst F = 26, 'Punch one number' can be used to punch this number by the order.

F	X	Y	Z
26	B02		

4.2 Jump Facilities.

Orders are normally obeyed in the sequence in which they are given, and hence do not need to specify the next order. However, provision is made for breaking this natural sequence by the inclusion of conditional and unconditional jump facilities. In such cases the out of sequence order to which the jump is made must be given a location L, a number between 01 and 96. Obviously no two orders can be given the same location, but any number of jumps can be made to the same order. The jumps are indicated by giving an order an N, used to specify the next location.

This is best illustrated by demonstrating how these jumps could be used in practice.

Suppose it is required to punch the number in A13, if it is non-zero, and then to read in another number to A13.

A program to do this is as follows:-

Card No.	F	X	Y	Z	L	N	
21	08	A13	000			11	Is A13 = 0?
22	26	A13					Punch A13
23	25			A13	11	17	

The card numbers have been arbitrarily assigned. Order 21 checks if A13 is zero, and if this is so it jumps to the order given by N. (This is a special form of the order, which is usually used for checking whether X = Y).

(a) If A13 is zero then the natural sequence is broken and the order with an L of 11 is taken next, i.e. No. 23. This order reads a fresh number into A13. As this order has an N punched as 17, the program now jumps to the order (not in the above three) with L = 17.

(b) If A13 is non-zero the next order in natural sequence is obeyed, i.e. No. 22, and this punches A13. The program then continues to No. 23 as in the first case.

An L of 96 has a special extra significance, see section on 'Data Failures in Translated Programs'.

4.3 Complete Order Format.

All the facilities of scaling, order modification etc., are, when required, punched on the same card as the basic order, and the complete format is as follows:-

Name	Columns	Description	Form
F	17,18	Function No.	2 dec. digits
X	19-21	X Address	1 alpha and 2 dec. digits or 3 dec. digits
Y	22-24	Y Address	1 alpha and 2 dec. digits or 3 dec. digits
Z	25-27	Z Address	1 alpha and 2 dec. digits or 3 dec. digits
S	28	Parameter	1 dec. digit
L	29,30	Location	2 dec. digits
N	31,32	Next location	2 dec. digits
M_1	33 - 35	1st Modifier address (X)	1 alpha and 2 dec. digits or zero and 2 dec. digits
M_2	36 - 38	2nd Modifier address(Y)	1 alpha and 2 dec. digits or zero and 2 dec. digits
M_3	39 - 41	3rd Modifier address (Z)	1 alpha and 2 dec. digits or zero and 2 dec. digits
M/T	42	Modify/tabular digit	Alphabetic M or T
P	43	Optional punch digit	Alphabetic P

Normally only the F and such of X, Y and Z as are necessary need be punched. S is punched only if the function needs a parameter. L and N are only used for breaking the natural sequence of orders. M_1, M_2, M_3 and M/T are needed for modification of orders, and P for optional punching, for testing purposes.

5. SCALING.

Due to the limits imposed on the size of numbers it may prove necessary at certain points in the program to scale the numbers in order that they should remain valid and also that full accuracy be maintained.

The result of any arithmetical operation may be scaled by the inclusion in the order of a parameter S. This parameter is punched immediately after Z, the result address.

There are four scaling factors available:-

$\frac{1}{2}$, 2, 1/10, 10,

which correspond to parameters 1, 2, 3 and 4 respectively.

Thus, to multiply the number in A29, by the number in F02, and place the result, after having been scaled down by 10, in C13, the following order is sufficient:-

F	X	Y	Z	S
01	A29	F02	C13	3

Scaling cannot be used to reduce a number which has already exceeded the prescribed limits. Should scaling up, by 2 or 10, cause a number to exceed capacity, a failure indication is given.

6. ORDER MODIFICATION.

It is frequently useful to be able to obey one or more orders repeatedly, but working on different data locations each time. There are two methods of order modification facilities available in EASICODE which can be used for this purpose. For cases when one operation is to be done to a set of consecutive numbers forming part or the whole of a block, tabular modification is most suitable. For order modification of a less simple kind, the second system, modify, is used.

These two systems are distinguished by punching T in the M/T column for 'tabular' working, or M for 'modify'.

6.1 Tabular.

This enables certain operations to be carried out repeatedly down a block, starting as given by the X, Y and Z address. Column length can be fixed (e.g. 017), or variable (by giving the address where the column length is stored, e.g. B23).

Thus,

F	X	Y	Z	S	L	N	M_1	M_2	M_3	M/T	P
01	A02	B11	C12				011	011	011	T	

multiplies A02 by B11 and puts result in C12,

A03 by B12 and puts result in C13,

and so on down to

A12 by B21 putting result in C22.

This would also have happened if, for example, M_1, M_2 and M_3 had been punched E05, which at time of execution contained the integer 11.

The storage in blocks is cyclic, i.e. a column of length 10 starting at B28 would end at B05.

It is not necessary for all three addresses to have tabular modification. Thus, a constant can be subtracted from a column, for example. Naturally any of the M_1, M_2, and M_3 addresses which are punched must have the same value, whether a number or an address.

Ex.

F	X	Y	Z	S	L	N	M_1	M_2	M_3	M/T	P
01	C21	C22	C21					009		T	

This order will form the product of the column of 10 numbers in the column starting at C21, putting the result in C21.

There are only two disallowed combinations of constant and tabular working,

(i) X and Y constant, Z tabular, on an order of form F(X,Y) → Z.

(ii) X tabular, Z constant, on an order of form F(X) → Z.

6.2 Modify.

This allows one operation to be performed on varying data positions.

The contents of the high speed store addresses given in M_1, M_2 and M_3 (presumed to be integers) modify the corresponding X, Y, or Z addresses before the order is obeyed.

Thus the order:

F	X	Y	Z	S	L	N	M_1	M_2	M_3	M/T	P
01	A03	A07	B11				C01	C02	C03	M	

when C01, C02 and C03 contain 10, 3 and 25 respectively, will cause A13 to be multiplied by A10, the result being placed in B04.

As with 'tabular', storage in a block is cyclic.

7. SUBROUTINES OF ORDERS.

Any set of EASICODE orders, which performs a calculation which is frequently required, can be used as a subroutine by following it by an 'end of subroutine' order, and giving its first order an L. Subroutines of orders which must be of type 1 or 2 (specified in S parameter), are entered by means of an unconditional jump on an 'enter subroutine' order. The enter and end of subroutine orders must be consistent in their use of the type number. The 'exit subroutine' order, when reached, passes control to the order following the original 'enter subroutine' order. It is possible to use subroutines of type 1 inside subroutines of type 2, or vice versa.

8. DEUCE SUBROUTINES.

The list of basic orders in EASICODE does not include such functions as Sin, Log etc. Should such functions be required then the appropriate set of cards must be taken from the EASICODE subroutine library, and added to the program of basic orders.

The subroutines will have to be used in one or more of blocks A to D, and will be brought down into the correct blocks by use of the 'Fetch S.R.' order. Thus, the programmer can organise use of the high speed storage so that frequently used functions are left in their blocks and do not need frequent re-fetching, the final program thus gaining considerably in speed and efficiency. The order in which the subroutines are read in automatically assigns function numbers to them, in ascending order from 32.

The EASICODE subroutine library consists, in general, of standard DEUCE subroutines, with slight modifications so that EASICODE requirements are met. Part II gives details of subroutines at present available, and complete details of the use of subroutines in EASICODE.

9. PROGRAM CONSTANTS.

Program constants, i.e. numbers or integers which are permanently required in a program, may be incorporated at the translation stage by the order 'program constant'. For each of these orders the translator will expect to read, after the orders and subroutines, one program constant card.

10. OPERATION OF THE TRANSLATOR.

The punched cards are numbered as follows:

Translator	0 - 237
Multi-section read	1000 - 1012
Subroutines:	card nos. (0-n) in cols 13 - 16
	A,B,C or D (position of subroutine) in col. 12
	subroutine no. in cols. 8 - 11
	The code no. (ZC30T) is not punched in cols. 6 - 12
	but the serial number (715) is punched in cols. 2 - 5
Telescope	2000 - 2022
Steam Tables	3000 - 3071
Bessel Functions	4000 - 4150

The translator takes as input:

1. Decimal orders, which must terminate with an order 31.
2. Subroutines, if required.
3. Program constants, if required.

There is an alphanumeric read failure check on the orders, indicated by 12 - 24X, with the O.S. lights on if it is the functions number (Cols. 17,18) which is invalid, and a decimal read failure check, also 12 - 24 X, on the program constants. From both these positions a single shot will cause program to return to re-read the card.

Once the input is complete the translator checks the orders, and desymbolises the L's and N's. If no errors are found the orders are then translated into codes, i.e. DEUCE instructions entering subroutines, storing results, etc. There follows a short section which replaces the N's with the code numbers.

The resulting program is then punched out (on 64 columns).

If the I.D. is clear the program stops on 30 - 17 X. A single shot causes the translator to reconvert orders to an alpha-numeric form and punch out an EASICODE flow-diagram and then stop on 1 - 1 X. From this position a single shot causes the program to enter the translated EASICODE program, without the necessity of reading it in.

If the I.D. contains a P32 when the machine finishes punching out the translated program, the machine stops on 0 - 25 X. If the translated program is then run in the translator will stop on 30 - 17 X if the program was punched correctly and the translator proceeds as above. If the translated program does not check, the translator stops on 8 - 24 X and after a S.S. repunches the program.

11. CHECKING DURING TRANSLATION.

With the sole exception of the alpha-numeric and decimal read failures, all errors cause a card to be punched out (which should be tabulated on a 64 column straight listing board) and the translator to loop on 9 - 24 either immediately, or, for failures in compatibility check and desymbolise, after the checking has been completed.

Types of error cards.

1. Input - such errors as too many orders, subroutines numbered incorrectly, etc.

2. Compatibility check - error cards give order number, address number, and address, error type number, and error type. F is address No. 1, X is No. 2 etc., whilst error types are respectively non-alphabetic, non-numeric, not blank, not punched, and invalid, corresponding to type numbers 1, 2, 3, 4 and 5.

If F is valid the rest of the order is checked, assuming that F is correct. If M_1, M_2 or M_3 are in error then checking on the M/T column is done as though the address in error was not punched. Any of the more subtle errors in the use of the order modification facilities will be shown as an error in the M/T column.

3. Desymbolise - error cards give order number and type of error.

4. Translation - it is possible that an EASICODE program, that is within the limit of 256 orders, will produce a translated program that is too large for the machine (this is due to a very high proportion of tabular orders, in a large program). This can only be checked during actual translation and will produce an error card and loop on 9-24.

A tabulation of some representative error cards is given in Appendix 3.

12. TESTING A TRANSLATED PROGRAM.

A P32 on the I.D. when an EASICODE program is running will cause it to stop on 8 - 24 and display current code number in arabic numerals in DL11. This number can be used, in conjunction with the EASICODE flow-diagram, to check which order is being obeyed. A single shot will cause program to continue. This position is also reached by failures which also put on one of the O.S. lights, and by order 14 Halt.

From this position various program testing facilities are available.

12.1 Request Stop.

Put discrim up, single shot, discrim normal again. The program will now read a 4 digit decimal number from the I.D. as for R08, and will then run on until this code number is reached. A single shot will cause the program to cancel the request stop and obey this order. A second request stop must not be set up until the first has been cancelled.

12.2 Jump.

Put discrim down, single shot, discrim normal again. A 4 digit number put in on the I.D. will cause the program to jump to and stop on this code number. A single shot will cause it to obey this code.

12.3 Post Mortem.

Jump the program to 0000. The complete High speed store is punched, six per card, and the program then returns to the original code number. This function has no effect on the running of a program.

13. OPTIONAL PUNCHING.

On certain orders the P column can be used to give the result of the order. These P digits become effective only if the translated program is read in with a P32 on the I.D. throughout the drum clearing period). The I.D. can be cleared as soon as the main program starts reading in.

14. EASICODE FLOW DIAGRAM.

This should be tabulated on a straight listing board (64 columns) with double spacing, and no zero suppression.

The first columns give order number and the next ones equivalent code number. Only these code numbers should be used in Request Stopping or Jumping.

A specimen tabulation of part of a flow-diagram is given in Appendix 3.

15. DATA FAILURES IN TRANSLATED PROGRAMS.

During the running of the translated program failures due to numbers becoming invalid etc., cause the program to reach 8 - 24 X with one of the OPS lights on and the current code number displayed in DL11. This code number will be one or more greater than the code number of the order that failed, which can be found by reference to the tabulated flow-diagram

The result of a single shot after a failure depends on whether an L of 96 has been used on any order.

If no L of 96 has been used then a single shot causes program to continue, taking zero as result of the operation that failed.

An L of 96 is used, however, to provide a 'Restart' procedure. If this has been used then a single shot from a failure stop causes program to jump to and stop on the order with L = 96, which could thus be used, say, to punch out relevant information and call the next set of data.

16. PUNCHING OF DATA.

Data is punched in the form of up to 6 signed 9 digit decimal numbers per card. Numbers are to 9 decimal places, integers to zero decimal places, signs in columns 17, 27 etc., numbers in columns 18 - 26, 28 - 36, etc. If less than six numbers are specified per card, then unwanted columns must be left blank.

However, in any set of cards read in by one order, using function 27, or 29, each card must be punched with the number of fields given in the S parameter, if necessary completing the last card with zeros.

e.g.

F	X	Y	Z	S	L	N	M_1	M_2	M_3	M/T	P
29		011	A01	5							

This will read 11 numbers, 5 per card into block A, starting at A01. Hence these numbers must be punched on 3 cards, the third card having the eleventh number and four zeros punched on it.

Orders 27 - 30 should be used for normal input and output, but orders 25 and 26 can be used when only a very few numbers are needed in scattered locations. Optional punching should be used only for testing purposes.

17. TIMES AND COMPARISONS.

"A brief case history of one job with a system seldom gives a good measure of its usefulness, particularly when the selection is made by the authors of the system".

J. W. Backus, 1957 Western Joint Computer Conf.

The author's only defence against the above comment is that the following evaluation of EASICODE is based on the example given in the ALPHACODE report, which is certainly not the most flattering that could have been chosen for EASICODE.

The example is that of solving linear simultaneous equations, as in the standard fixed point DEUCE program, LEO6.

5 equations, 1 R.H.S.	LEO6	EASICODE	ALPHACODE
Total time	1m - 8s.	59s.	2m - 54s.
Reading program	49	32	1 - 3
Reading data	5	7	10
Solving equations	11	16	1 - 38
Punching results	3	4	3

5 equations, 4 R.H.S.			
Total time	1m - 22s.	1m - 10s.	4m - 28s.
Reading program	49	32	1 - 3
Reading data	6	8	14
Solving equations	12	22	2 - 59
Punching results	15	8	12

10 equations, 1 R.H.S.			
Total time	1m - 28s.	1m - 52s.	9m - 5s.
Reading program	49	32	1 - 3
Reading data	15	15	34
Solving equations	17	58	7 - 22
Punching results	7	7	6

10 equations, 4 R.H.S.			
Total time.	1m - 52s.	2m - 19s.	13m - 54s.
Reading program	49	32	1 - 3
Reading data	18	19	43
Solving equations	19	74	11 - 44
Punching results	26	14	24

From these figures, it will be seen that the ratio of times taken by the EASICODE program and LEO6 for the above cases varies between 0.85 and 1.3, as compared to the ratio of time taken by the ALPHACODE program and LEO6, which varies between 2.6 and 7.5 .

More instructive however, is the comparison of the actual computation time, as the input and output arrangements of the three programs are so different, EASICODE in particular taking full advantage of 64 column input and output.

The ratio of computation times is as follows:-

EASICODE to LEO6 - from 1.5 to 3.9

ALPHACODE to LEO6 - from 9 to 37

Lastly the number of EASICODE orders is 87, as compared to 112 ALPHACODE orders, and over 1700 instructions in LEO6.

Naturally this single example is nowhere near sufficient to assess the relative capabilities of the three systems, but from the considerable experience so far obtained at Whetstone, it seems that **EASICODE** can be up to 3 or 4 times slower than a fixed point DEUCE program, as regards actual computation, and 10 times faster than **ALPHACODE**, and is normally very efficient as regards input and output.

18. ACKNOWLEDGEMENTS.

EASICODE was originally developed and programmed by M. J. Kelly and the author. Valuable assistance in the later stages of programming, and in the preparation of the subroutine library, was received from L. J. Russell.

The author also wishes to acknowledge the help received from N. D. Lam, M.E.L., and Mrs. J. Smith, with various input-output routines, and the encouragement received from the many people whom the authors originally plagued with their ideas for this scheme, in particular Dr. P. G. Wakely, M.E.L., who made the machine time available for the first pilot version.

APPENDIX I

00 MULTIPLY

Multiplies X by Y and puts result in Z.

02 DIVIDE

Divides X by Y and puts result in Z.

03 ADD

Adds X to Y and puts result in Z.

04 SUBTRACT

Forms X - Y and puts result in Z.

Orders 01 - 04 can all be scaled, P-digited, given locations or next addresses, and be made 'modify' or 'tabular'.

X, Y, Z must all be High Speed Store addresses, e.g. A01, F29, etc., with one possible exception:- X can be punched 000 in order 04 to form - Y, but in this case X must not be modified.

05 MODULUS

Modulus of X goes to Z. Similar to above orders except that Y and M_2 must not be punched.

06 COUNT

Adds the parameter S to the integer in X. Other than F, X and S only L and N may be used. S from 1, ... 9. X a high-speed store address.

07 UP TO

This order needs F, X, Y, S and N. X is a high speed store address. Y is of this form or a positive integer less than 1000. S from 1, ... 9. Each time the order is obeyed S is added to X. When X = Y (after addition) the following order is the next to be obeyed, otherwise the program jumps to the position given by N. L can also be punched.

08 EQUAL

If X = Y the program jumps to order given by N. X is a high speed store address. Y is a high speed store address or an integer between 000 and 999 inclusive. X and/or Y can be modified if they are addresses. L can also be punched.

09 BIG AS

If $X \geqslant Y$ the program jumps to N. X and Y are high speed store addresses or Y can be 000. L can also be punched.

10 EXCEEDS

If $X > Y$ the program jumps to N. Otherwise this order is similar to 09.

11 UNEQUAL

If $X \neq Y$ the program jumps to N. Similar to 08.

13 TEST

N. and L can be punched. This order initiates 15 seconds of machine checking, which includes random magnetics and both random and standard multiplications and divisions. Whilst the checking is done all the OPS lights are on. If a machine failure is found the machine buzzes and a card is punched with ones on Y-row α, $'$ and β -fields, track number failing, X P5, with a P15 if a writing failure, or all ones if an arithmetic failure. After a failure is found the routine returns to the beginning of the 15 seconds.

If TIL is on the test program continues indefinitely but failures are displayed on the OPS instead of being punched on Y row β -field.

4, 4 - 17 X BUZZ is sum-check failure in test program itself.

14 HALT

Stops 8 - 24 X, with code number of next instruction displayed in arabic numerals in DL11. A single shot causes program to proceed..

N and L can be punched.

15 ENTER SR

This order, which needs N and S, causes program to enter a subroutine of orders, by jumping, according to N, to its start. S = 1 or 2 gives type of subroutine.

L can also be used.

16 END OF SR

Needs S (1 or 2). L can be used.

This order must follow a set of orders to be used as a subroutine, and will cause program to resume at the order following the ENTER SR order which jumped to the subroutine.

The subroutine type number S must be used consistently with orders 15 and 16. Thus, if a subroutine ends with an order specifying it as type 2 the order 15 leading to it must say enter subroutine of type 2.

It is possible to use subroutines of type 1 inside subroutines of type 2, or vice-versa.

17 MOVE HS—HS

This order is used for transferring numbers, in the high-speed store from the address given by X to that given by Z. Can be scaled, P digited, and modified or tabular. With tabular modification, transfer is done one word at a time, so if Z and X are in the same block, overwriting may occur.

Alternatively this order can be used for clearing, by putting X = 000, but in this case the order cannot be scaled or P digited, and M_1 must not be used.

18 MOVE HS—DR

Moves a block or blocks from the high-speed store to the drum. X must be of form A00, B00 etc., and Z must be a drum track number, (001 upwards) or 000 if it is modified. See examples on Page 18.

19 MOVE DR—HS

Moves tracks of data on the drum. Similar to 18 and 19, but both X and Z refer to the drum.

20 MOVE DR—DR

Moves tracks of data on the drum. Similar to 18 and 19, but both X and Z refer to the drum.

21 FETCH SR

Fetches Deuce Subroutine to correct place in high-speed store. Subroutine number is given in X (e.g. 032 for first subroutine). N and L can also be used.

For multiple entry subroutines, which have consecutive function numbers corresponding to each entry, only the function number of the first entry should be used as X in a fetch SR order.

23 PROG CONST

This causes one program constant to be placed in the high-speed address given by Z. S must be punched as 1 for a number to 9 dec. places, or 2 for an integer.

Thus for every order 23 in the pack of input orders the translator expects to read in one number or integer (punched as sign and 9 digits from column 17). These constants will be used in the order that the function 23 orders appear in the pack.

M_3 can be used to modify Z, and N and L can also be used.

A maximum of 96 program constants can be used in one program.

24 READ PARAM

Reads one parameter (i.e. integer) to high speed store address given by Z. N and L can be used, and also M_3 to modify Z.

25 READ ONE

Reads one 9 decimal place number to high speed store address given by Z. N and L can be used, and also M_3 to modify Z.

26 PUNCH ONE

Punches one 9 decimal place number from high speed store address given by X. N and L can be used, and also M_1 to modify X.

27 READ ROWS

Reads a number of cards given in Y (as a number e.g. 010, or as contents of a high-speed store address, e.g. B27) to Z. S gives number of numbers per card (S from 1, ... 6). N and L can also be used.

Thus

F	X	Y	Z	S	L	N	M_1	M_2	M_3	M/T	P
27		012	C05	3							

will read 12 cards (3 numbers on each card) the first set of numbers going to C05, D05, E05, the second to C06, D06, E06, etc.

With Z punched as a high-speed store address as above, M_3 can be used to modify the address given in Z.

Alternatively Z can be a track number or 000 with M_3 giving the address of a track number. Then the first set of numbers will go to position 1 of the tracks indicated and the positions in the tracks which are not read into will be cleared. A copy of these tracks will also be left in the high speed store, starting at block A.

e.g.

F	X	Y	Z	S	L	N	M_1	M_2	M_3	M/T	P
27		009	000	2					F01	M	

where F01 contains 087,

causes 9 cards to be read, the first card being read to track 87 position 1, and track 88 position 1, and also to A01 and B01, the second to track 87 position 2, and track 88 position 2, and also to A02 and B02, and so on. The space, under the numbers read in, on the drum tracks 87 and 88 will be cleared, as will A10 to A32, and B10 to B32.

28 PUNCH ROWS

Similar to 27 but punches from X. Once again, if X is a drum address, a copy of the tracks punched out will be left in the high-speed store starting at A. M_1 can be used for modification.

The tracks from which punching takes place are not changed in any way by the order.

29 READ COLS

Reads Y numbers, S per card, into one block, starting at address given by Z. N and L can also be used. Y can be a fixed number, or an address in the high-speed store, S from 1, ... 6.

If Z is a high-speed store address M_3 can be used to modify the starting position, alternatively Z can be a track number. Reading to drum will put first number in position 1 of a track, but does not disturb the high-speed store. The spaces underneath the input numbers, when reading to drum, will be cleared.

30 PUNCH COLS

Similar to 29, but punches from X, and M_1 can be used for modification. The block or track from which numbers are punched is not changed in any way by the order.

With orders 27 to 30, Y, the number of cards or numbers should be between 1 and 32, but will be taken modulo 32 if it exceeds this limit.

31 FINISH

Indicates end of orders. If reached in translated program it shows as 12 - 1 looping on itself.

Deuce subroutines are either of type f(x,y) → Z, or f(x) → Z, and have all the facilities available to 01 MULTIPLY or 05 MODULUS, respectively, unless stated otherwise in their write ups.

Exs.

F	X	Y	Z	S	L	N	M_1	M_2	M_3	M/T
1	18	A00	012							
2	18	A00	000					B01		M
3	18	B00	010				A03		A03	T
4	18	B00	010						003	T

Example 1 transfers block A to track 012, whilst example 2 transfers it to the track whose number is given in B01. Example 3 transfers B to 010, C to 011 etc., the number of transfers being given in A03, and example 4 transfers B to 010, 011 and 012.

Tabular working can move up to 6 blocks at a time if starting with block A, otherwise only 4 if starting from block C for example.

L and N can be used.

APPENDIX II

As stated in the section on data failures, these cause the program to reach 8 - 24 X, with the code number displayed in DL11 and one of the OPS lights on.

Types of Failure

P1	Scale
P2	
P3	
P4	Graph
P5	Log
P6	Replace
P7	
P8	
P9	
P10	Square Root
P11	
P12	
P13	
P14	
P15	
P16	
P17	Add and Subtract
P18	
P19	
P20	Gen. read and punch - invalid drum tracks
P21	
P22	
P23	
P24	
P25	Exp(-x)
P26	
P27	Integral
P28	
P29	
P30	Drum Transfer
P31	
P32	Division

This list includes failures in various subroutines - details are given in Part II.

EASICODE PROGRAM

APD P100

1	0001	PROG CONST			A18	1						
2	0004	READ PARAM			A01							
3	0006	READ PARAM			A02							
4	0008	READ ROWS		A01	C01	3						
5	0012	MOVE HS HS	D01		F01	1	A01		A01	T		
6	0022	MOVE HS HS	D32		A04		A02			M		
7	0025	MULTIPLY	F01	F01	D01		A01	A01	A01	T	P	
8	0034	MULTIPLY	C01	E01	E01		A01	A01	A01	T	P	
9	0043	FETCH SR	032									
10	0051	MOVE HS HS	000		A03							
11	0052	MOVE HS HS	D01		A07		A03			M		
12	0055	MOVE HS HS	E01		A08		A03			M		
13	0058	EXCEEDS	A08	A07								124
14	0062	SUBTRACT	A07	A08	A09						P	
15	0065	SQ ROOT	A09		A10						P	
16	0067	SUBTRACT	000	F01	A11			A03		M	P	
17	0071	ADD	A11	A10	A12	3					P	
18	0075	SUBTRACT	A11	A10	A13	3					P	
19	0079	MOVE HS HS	C01		A14		A03			M		

Tabulated Error Cards

1. Input

TOO MANY ORDERS

TOO MANY PROG CONSTS

S.R. NOS- INCORRECT

TOO MANY S.R.TRACKS

2. Compatability Check

1	5	S	3 NOT BLANK
1	10	M3	5 INVALID
2	2	X	1 NON ALPHA
4	4	Z	2 NON NUM
4	9	M2	3 NOT BLANK
11	11	MT	4 NOT PCHED
12	3	Y	3 NOT BLANK
12	5	S	3 NOT BLANK

3. Desymbolise

2 L REPEATED

10 N INVALID

4. Translation

TOO MANY CODES-

PART II

EASICODE SUBROUTINES.

INTRODUCTION.

The Easicode subroutines provide a means of including Deuce programs within Easicode programs or of using Easicode as a subroutine of a standard Deuce program. Thus it is possible to combine the ease of programming achieved with Easicode and the speed of a standard Deuce program.

20 USE OF EASICODE SUBROUTINES.

20.1 Determination of Function Number of Subroutines.

The program pack is assembled behind the translator in the following manner:

1. Decimal Orders
2. Easicode Subroutines
3. Program Constants

The subroutines are numbered sequentially from f = 32 to a maximum of f = 63, according to the order in which they are placed behind the decimal orders.

20.1.1 Function Numbering of Multi-Entry Subroutines.

When a multi-entry subroutine is used, the subroutines in it are numbered sequentially (i.e. for subroutine No. 18 in the list given in paragraph 5, if the $\frac{1}{2}$ Sinh x entry is f = 35 then the $\frac{1}{2}$ Cosh x entry is f = 36).

20.2 Fetch Subroutine.

The order f = 21 fetches the subroutine, whose function number is given in the X address, to its correct place in the high speed store.

The subroutine need not be re-fetched before subsequent use provided that no part of the block, or blocks, containing the subroutine have been over written.

It is the responsibility of the programmer to ensure that the subroutine is in position before using it.

20.2.1 Fetch Multi-Entry Subroutines.

Whichever subroutine, of the batch included in the multi-entry subroutine, is to be used the function number of the first subroutine in the batch is to be used in the X address of a fetch subroutine order (f = 21). (i.e. for the example given in paragraph 4.1.1, to use the $\frac{1}{2}$ Cosh x subroutine, the $\frac{1}{2}$ Sinh x subroutine is fetched).

20.3 Use of Subroutines.

Subroutines are either of type f(x,y) $\longrightarrow$ z or f(x) $\longrightarrow$ z and have all the facilities available to f = 01 Multiply or f = 05 Modulus respectively, unless otherwise stated in section 21.

Examples: To use the square root subroutine, which was the first subroutine to be read in behind the decimal orders. This subroutine is read into block A.

	F	X	Y	Z	S	L	N	M_1	M_2	M_3	M/T
1.	21	032									
2.	32	B01		B01							
3.	32	C01		D01				032		032	T

Order 1 fetches the square root subroutine to its correct place (i.e. block A) in the high speed store.

Order 2 replaces x from B01 with its square root.

Order 3 takes the number in C01 and puts its square root in D01. Similarly for C02 and D02 etc.

20.3.1 Using Every Subroutine.

Every subroutine that is included behind the decimal orders must be used at least once in the program. If a multi-entry subroutine is used, every entry in that subroutine must be used at least once in the program.

The reason for the above is that the routine used to read in the decimal orders, subroutines and program constants, checks the function numbers used in the program as the decimal order cards are read in. It uses the dictionary so formed to decide how many subroutines it has to read in.

20.4 Data Range for Easicode Subroutines.

The subroutines will either accept the full range of data allowable in Easicode (i.e. $-1 < x < +1$) or will give a failure indication for unacceptable data input.

The details of the range of data acceptable and the failure indication, if any, are given in the list of subroutines in paragraph 5.

In all cases the data output, if any, of the subroutines will be within the range allowable in Easicode.

20.5 Subroutine Failures.

A failure to observe the conditions of use of a subroutine, given in paragraph 5, will cause the machine to stop at 8 - 24 X with the code number displayed in DL11 and one of the O.P.S. lights on.

A list of the failure indications and the respective O.P.S. lights used is given below.

O.P.S.	Subroutine giving failure
P4	Graph
P5	Log
P6	Replace
P10	Square Root
P25	Exponential -x
P27	Integral

20.5.1 Restart Facility.

If there is no order having an L = 96 in the program, a single shot after failure causes the program to continue, taking zero as the result of the operation that failed. However, if an L = 96 has been allocated to an order in the program, a single shot after failure causes the program to jump to, and stop on, this order.

21 EASICODE SUBROUTINE LIBRARY.

All the subroutines, which are at present in the library, are listed below along with ails for their use, data ranges, failure indications, etc.

1. Square Root.

Type: $f(x) \rightarrow z$
Uses: One Block - available in blocks A; B; C; or D.
Input: x
Output: Square root of x
Failures: Machine stops at 8 - 24 X with the P10 O.P.S. light on if $x < 0$.

2. Log.

Type: $f(x) \rightarrow z$
Uses: Two Blocks - available in blocks A, B; B, C; or C, D.
Input: x
Output $2^{-5} \log_e x$
Failures: Machine stops at 8 - 24 X with the P5 O.P.S. light on if $x \leqslant 0$.

3. Exponential.

Type: $f(x) \rightarrow z$
Uses: Two Blocks - available in blocks A, B; B, C; or C, D.
Input: x
Output: $2^{-2} e^x$
Failures: None.

4. Exponential(-x)

Type: $f(x) \rightarrow z$
Uses: Two Blocks - available in blocks A, B; B, C; or C, D.
Input: x_{-2}
Output $e^{-x^{10}}$
Failures: Machine stops at 8 - 24 X with the P25 O.P.S. light on if $x \leqslant 0$ or $x \geqslant 16$.

5. Graph.

Type: $f(x,y) \rightarrow z$
Uses: Two Blocks - available in blocks A, F; B, F; C, F; or D, F.
Input: X in the X address and N (integer) in Y address positions.
Output: Y in Z address position.
Failures: Machine stops at 8 - 24 X with the P4 O.P.S. light on if

$$x_N < X \leqslant x_1$$

This subroutine performs linear interpolation on a graph of N points ($N \leqslant 16$). If these points are denoted by $(x_i y_i)$ (where $2 \leqslant i \leqslant N$) then, given a value X, the subroutine calculates Y:-

$$Y = y_{r+1} + (y_r - y_{r+1}) \frac{x_{r+1} - X}{x_{r+1} - x_r}$$

where $x_r < X \leqslant x_{r+1}$

(y_i, x_i) is stored in Block F from F01 onwards (in order $y_1, x_1, y_2, x_2 \cdots y_N, x_N$) the x_i ing in algebraic ascending order.

Note: N must be an integer.

6. Integral.

Type: $f(x,y) \longrightarrow z$
Uses: Two Blocks - available in blocks A, B; B, C; C, D. f_o-f_n are stored in F01 onwards.
Input: h in the X address and n (integer) in the Y address positions.
Output: $Z = \int_0^{nh} f(x).dx \qquad 10^{-2}$
Failures: The machine stops at 8 - 24 X with the P27 O.P.S. light on if:
1. n is not an integer
2. $n < 0$
3. $n > 31$

This subroutine uses the $\frac{3}{8}$ rule and Trapezoidal formulae as follows:

$$\int_0^{3h} f(x)\,dx = \frac{3h}{8}\,(f_o + 3f_1 + 3f_2 + f_3)$$

$$\int_0^{h} f(x)\,dx = \frac{h}{2}\,(f_o + f_1)$$

7. Sin.

Type: $f(x) \rightarrow z$
Uses: Three blocks - available in blocks A, B, C or B, C, D.
Input: $x \quad . \quad 10^{-1}$
Output: Sin x

8. Cos.

Type: $f(x) \longrightarrow z$
Uses: Three Blocks - available in blocks A, B, C or B, C, D.
Input: $x \quad . \quad 10^{-1}$
Output Cos x

9. Sin/Cos Multi-Entry

Type: $f(x) \longrightarrow z$
Uses: Three Blocks - available in blocks A, B, C or B, C, D.
Input: $x \quad . \quad 10^{-1}$
Output: Sin x or Cos x.

10. Sin

Type: $f(x) \longrightarrow z$
Uses: One Block - available in blocks A, B, C or D.
Input: x
Output: $\frac{1}{2}$ Sin $(\pi/2)$ x

11. Cos

Type: $f(x) \longrightarrow z$
Uses: One Block - available in blocks A, B, C or D.
Input: x
Output: $\frac{1}{2}$ Cos $(\pi/2)$ x

12. Sin/Cos Multi-Entry

Type: $f(x) \longrightarrow z$
Uses: One Block - available in blocks A, B, C, or D.
Input: x
Output: $\frac{1}{2}$ Sin $(\pi/2)$ x or $\frac{1}{2}$ Cos $(\pi/2)$ x

13. $\underline{\text{Sin}^{-1}}$

Type: $f(x) \rightarrow z$
Uses: One Block - available in blocks A, B, C or D.
Input: x
Output: $(1/2\pi)\ \text{Sin}^{-1}\ x$

14. $\underline{\text{Cos}^{-1}}$

Type: $f(x) \rightarrow z$
Uses: One Block - available in blocks A, B, C or D.
Input: x
Output: $(1/2\pi)\ \text{Cos}^{-1}\ x$

15. $\underline{\text{Sin}^{-1}/\text{Cos}^{-1}}$ Multi-Entry

Type: $f(x) \rightarrow z$
Uses: One Block - available in blocks A, B, C or D.
Input: x
Output: $(1/2\pi)\text{Sin}^{-1}\ x$ or $(1/2\pi)\ \text{Cos}^{-1}\ x$

26. Sinh

Type: $f(x) \rightarrow z$
Uses: Two Blocks - available in blocks A, B; B, C; or C, D.
Input: x
Output: $\frac{1}{2}$ Sinh x

17. Cosh

Type: $f(x) \rightarrow z$
Uses: Two Blocks - available in blocks A, B; B, C; or C, D.
Input: x
Output: $\frac{1}{2}$ Cosh x

18. Sinh/Cosh Multi-Entry

Type: $f(x) \rightarrow z$
Uses: Two Blocks - available in blocks A, B; B, C; or C, D.
Input: x
Output: $\frac{1}{2}$ Sinh x or $\frac{1}{2}$ Cosh x

19. $\underline{\text{Sinh}^{-1}}$

Type: $f(x) \rightarrow z$
Uses: Three Blocks - available in blocks A, B, C or B, C, D.
Input: x
Output: $\text{Sinh}^{-1}\ x$

20. Steam Subroutines - Multi-Entry.

Before using the steam subroutines the steam tables must be in position on the drum.

These tables are read to their correct position on the drum by means of the "READ" Easicode Subroutine No. 24 which treats them as program data.

The steam tables occupy tracks 1 - 16, 21 - 27, 29 - 32 inclusive.

There are ten steam subroutines which are listed below.

FAILURES: Machine stops at 9-24 X with P5, P31 or P32 O.P.S. light on if the data is inconsistent. By putting a P32 on the I.D. and giving a single shot the program will re-enter Easicode and stop at 8-24 X (this will cause zero to be transferred to the Z address). Thus it will be possible to note the position at which failure occurred from DL11.

20/1 Enthalpy of Superheated Steam.

Type:	$f(x,y) \rightarrow z$
Uses:	Three Blocks A, B and C.
Input:	Pressure x 10^{-4} in X address; temperature x 10^{-4} in Y address
Output:	Enthalpy of superheated steam x 10^{-4}.

20/2 Entropy of Superheated Steam.

Type:	$f(x,y) \rightarrow z$
Uses:	Three Blocks A, B, and C
Input:	Pressure x 10^{-4} in X address; temperature x 10^{-4} in Y address.
Output:	Entropy of superheated steam x 10^{-4}

20/3 Saturation Temperature of Water.

Type:	$f(x) \rightarrow z$
Uses:	Three Blocks A, B and C.
Input:	Pressure x 10^{-4}
Output:	Saturation temperature of water x 10^{-4}

20/4 Enthalpy of Water at Saturation.

Type:	$f(x) \rightarrow z$
Uses:	Three Blocks A, B and C.
Input:	Pressure x 10^{-4}
Output:	Ethalpy of water at saturation x 10^{-4}

20/5 Entropy of Water at Saturation.

Type:	$f(x) \rightarrow z$
Uses:	Three Blocks A, B and C.
Input:	Pressure x 10^{-4}
Output:	Emtropy of water at saturation x 10^{-4}

20/6 Saturation Pressure of Water.

Type:	$f(x) \rightarrow z$
Uses:	Three Blocks A, B and C.
Input:	Temperature x 10^{-4}
Output:	Saturation pressure of water x 10^{-4}

20/7 Enthalpy of Compressed Water.

Type:	$f(x,y) \rightarrow z$
Uses:	Three Blocks A, B and C.
Input:	Temperature x 10^{-4} in X address, pressure x 10^{-4} in Y address.
Output:	Enthalpy of compressed water x 10^{-4}

20/8 Entropy of Superheated Steam.

Type:	$f(x,y) \rightarrow z$
Uses:	Three Blocks A, B and C.
Input:	Pressure x 10^{-4} in X address; temperature guess x 10^{-4} in Y address; and Enthalpy of superheated steam x 10^{-4} in E01.
Output:	Entropy of superheated steam x 10^{-4}

20/9 Temperature of Superheated Steam.

Type:	$f(x,y) \rightarrow z$
Uses:	Three Blocks A, B and C.
Input:	Pressure x 10^{-4} in X address; temperature guess x 10^{-4} in Y address; and Enthalpy or Entropy of superheated steam x 10^{-4} in E01.
Output:	Temperature of superheated steam x 10^{-4}

20/10 Enthalpy of Superheated Steam.

Type:	$f(x,y) \rightarrow z$
Uses:	Three Blocks A, B and C.
Input:	Pressure x 10^{-4} in X address; temperature guess x 10^{-4} in Y address; and Entropy of superheated steam x 10^{-4} in E01.
Output:	Enthalpy of superheated steam x 10^{-4}

The units of the data used in the above 10 steam subroutines are as follows:

Pressure	in p.s.i.a.
Temperature	in degress fahrenheit
Enthalpy	in B.T.U./lb.
Entropy	in B.T.U./lb.

21. Read 4's.

Type:	$f(x) \rightarrow z$
Uses:	Four Blocks A, B, C and D. The subroutine reads input data into block F.
Failures:	Machine stops at 6, 12 - 24 X if the input data card is not punched correctly. A single shot causes the card to be re-read.

This subroutine will read two cards each containing sixteen 4 digit unsigned numbers and stores them as positive numbers in block F from F01 - F32. The numbers being punched across the cards from columns 17 - 80 inclusive.

The subroutine must be used with a waste order of the form:

X	Y	Z
E01		E01

The position used in the Z address is cleared by the subroutine and should thus be position in block E or F.

Note: There is no point in using the 'modify' or 'tabular' facilities with this subroutine.

22. Punch 4's.

Type:	$f(x) \rightarrow z$
Uses:	Four Blocks A, B, C and D. The subroutine punches out block F.

This subroutine punches out the modulus of the numbers in block F from F01- F32 as 4 digit unsigned numbers 16 per card.

The subroutine must be used with a waste order of the form:

X	Y	Z
E01		E01

The position used in the Z address is cleared by the subroutine and should thus be a position in block E or F.

Note: There is no point in using the 'modify' or 'tabular' facilities with this subroutine.

23. Replace.

Type: f(x) → z
Uses: Four Blocks A, B, C and D.
Failures: Machine stops at 6, 12 - 24 X if the replacement data card is not punched correctly. A single shot causes the card to be re-read.
Machine stops at 8 - 24 X with the P6 O.P.S. light on if
1. col. 17 parameter is not a positive sign,
2. cols. 23 - 26 are not all zeros,
3. track number is ≤ 0 or $\geq (176 - n)$,
4. position number is ≤ 0 or > 32,

Track numbers 1 to (176 - n) of the drum are available for data storage in Easicode, where n is the number of tracks used by the Easicode subroutines.

This subroutine allows any of the positions in these tracks to be replaced by signed nine digit numbers, which are read in by the subroutine as data and are punched one number per card as follows:

col. 17	positive sign
cols. 18 - 20	track number
cols. 21, 22	position number
cols. 23 - 26	zeros
cols. 27 - 36	sign and nine digit numbers

This subroutine continues to read replacement cards until it encounters a blank card.

The subroutine must be used with a waste order of the form:

X	Y	Z
E01		E01

The position used in the Z address is cleared by the subroutine and should thus be a position in block E or F.

Note: There is no point in using the 'modify' or 'tabular' facilities with this subroutine.

24. Read.

Type: f(x) → z
Uses: Block A.

This subroutine reads in humby triads, finishing after the triad which has a P32 on the Y row in addition to the normal humby fillers. A humby triad is one which has the first 4 rows of the first card as follows:-

Y-row	Blank
X-row	0, a-31, 1, 0, 1 X
0-row	Blank
1-row	0, b-30, 1, 0, 1 G0

where the triad is to be read to track a/b.

The subroutine is used with a transfer order as f = 17.

Example:

X	Y	Z
E01		D02

This will read in humby triads and also transfer the contents of E01 to D02.

25. Exit.

Type: $f(x) \rightarrow z$
Uses: Blocks A and B
Input: Drum minor cycle number stored as an integer in the X address position.

This subroutine must have the function number f = 32.

The subroutine is used to exit from the Easicode program into a Deuce program which is stored on the drum. It fetches the track containing the minor cycle given in the X address position into DL1 and enters at that minor cycle. It thus assumes the drum to be a continuous store from minor cycle 0 to 8191 inclusive.

To re-enter the Easicode program the Deuce program must call track 10/15 into DL5 and enter at minor cycle nought.

The Z address position will be filled by the contents of temporary store 16 upon re-entry to the Easicode program as a number to 30 b.p. Thus care must be taken to see that this number is valid in Easicode.

26. This is a triple entry subroutine which calculates the following:-

Entry 1. Power.

Type: $f(x,y) \rightarrow z$
Uses: Four blocks A, B, C and D.
Input: b x 10^{-2} in the X address, a in the address, N (integer) in E01 and n (integer) in E02.
Output: $(a \times 10^{n})^{b} \times 10^{-N}$
Failures: Machine stops at 8 - 24 X with the P28 O.P.S. light on if:-

$$|n| \geqslant 16$$
$$|N| \geqslant 16$$
$$a \leqslant 0$$
$$|b| \geqslant 16$$
$$|(a \times 10^{n})^{b}| \times 10^{-N} > 1$$

If $(a \times 10^{n})^{b} \times 10^{-N} = 1$ the subroutine substitutes 0.999999999 in order that the output be within the range allowed in Easicode.

Entry 2. Log.

Type: $f(x) \rightarrow z$
Uses: Four blocks A, B, C and D.
Input: x
Output: $2^{-5} \log_{e} x$
Failures: Machine stops at 8 - 24 X with the P28 O.P.S. light on if: $x \leqslant 0$.

Entry 3. Exponential(- x)

Type: $f(x) \rightarrow z$
Uses: Four blocks A, B, C and D
Input: x x 10^{-2}
Output: e^{-x}
Failures: Machine stops at 8 - 24 X with the P28 O.P.S. light on if $x<0$

For x = 0 the subroutine gives 0.999999999 as the result for e^{-x} for this to be within the range allowed in Easicode.

27. Bessel Function Subroutine.

This is a single entry subroutine which calculates the value of a zero or first order Bessel function of the first or second kind for a wide range of values of the argument, using floating arithmetic.

Before using the Bessel function subroutine the Bessel functions must be in position on the drum. These functions are read to their correct position on the drum by means of the "READ" EASICODE subroutine No. 24, which treats them as program data.

The Bessel functions occupy tracks 20-23 and 33-83 inclusive. (If used in the same program as the EASICODE steam table subroutine No. 20, tracks 21-23 are used jointly by the two subroutines).

Tracks 20-23 are used as 'buffer' stores by the subroutine, but they can be used as temporary drum stores elsewhere in the program.

Type: $f(x,y) \longrightarrow z$

Uses: Block A

Input:
X Address ..... $x.10^{-m}$
Y Address ..... B (integer)
E01 ..... M (integer)
E02 ..... m (integer)

where B is:-

0 for Jo	10 for J_1
1 for Yo	11 for Y_1
2 for Io	12 for I_1
3 for Ko	13 for K_1

Output: $B(x).\ 10^{-M}$

Failures: Machine Stops 8-24 X with the P21 O.P.S. light on if:-

$x < 0$

$x = 0$ for $Yn(x)$ or $Kn(x)$

$x \geqslant 2^{12}$ for $In(x)$

$-1 \geqslant B(x).\ 10^{-M} \geqslant 1$

B is invalid

M is not an integer

m is not an integer

(N.B. $\underline{B(x).\ 10^{-M} = 0}$ for values of

$x \geqslant 2^{12}$ for $Kn(x)$

and $x > 2^{258}$ for $Jn(x)$ and $Yn(x)$)

This subroutine incorporates the General Bessel Function subroutine J22FM, No. 353.

28. Subroutine 'Out'.

This subroutine, occupying one block (A), is of the form $F(X) \longrightarrow Z$. This is used to leave one section and jump to the first order of the section whose number is given in X (an integer between 1 and 15). If X is invalid, or if there is no corresponding section then the subroutine fails 8 - 24 X with P18 on the O.P.S. lights.

A single shot from the failure stop causes the program to put zero in Z and carry on to the next order in the current section, unless the restart facility (i.e. use of an L of 96) has been used. If this is the case then a single shot causes program to jump, still in the current section, to the order with an L of 96.

PART III

MULTISECTIONED EASICODE.

INTRODUCTION.

Normal Easicode programs are limited to a maximum size of 256 orders. There is however data storage space on the drum for 5,000 numbers. Multi-section Easicode has been developed to allow programs to be written which may be considerably larger than is allowed in normal Easicode, but at the expense of the amount of data storage available.

Up to 16 separately translated Easicode programs may be combined to form one Multi-Section Easicode program, but the limit on total number of orders permissible depends on the amount of translated program generated from the original Easicode orders. A probable limit on total number of orders is about 1,500, but this may drop to below 1,000 if a very high proportion of tabular orders are used.

Separate Easicode programs are combined together by reading them into the machine as data to the Multi-section Read-in program, which checks that the programs being read are within the size limitations. The order in which these programs are placed automatically gives them section numbers, from 0 up to a maximum of 15.

23. SECTION CHANGING.

The complete multi-section program must start at the first order of Section 0. Entry to the first order of any other section can be made by using a special subroutine 'OUT'. This can be used in any section to jump to any of sections 1 to 15. Thus, it is not possible to re-enter Section 0. As section changing, especially with large sections, can take a second or so, it should be minimised, particularly in a small inner loop of a program.

24. USE OF SUBROUTINES.

All the various sections in a multi-section program use the set of subroutines given in Section 0. Furthermore, each section is an Easicode program in its own right, and hence must obey the rules governing use and numbering of subroutines.

This means that Section 0 must contain all the subroutines used in the complete program. In order that the numbering of subroutines is consistent from one section to another it may be necessary for some sections to include subroutines they do not use. This can be done by the inclusion of dummy orders which need not be obeyed.

For example, if Section 1 uses 'LOG', Section 2 uses 'SIN' and 'COS', and Section 3 uses 'SIN', then the subroutines actually needed in each section would be

Section 0	LOG, SIN, COS as functions 32, 33, 34
Section 1	LOG as function 32
Section 2	LOG, SIN, COS as functions 32, 33, 34
Section 3	LOG, SIN as functions 32, 33

Here certain sections must include subroutines they do not use in order that SIN and COS have the same function numbers in each section.

Naturally with each section calling on the Section 0 subroutine library, identical copies of subroutines must be used in each section. In particular a subroutine must be used in the same high speed blocks in each section. All these rules as regards use of subroutines are however checked by the multi-section read-in program.

25. READ-IN PROGRAM.

25.1 Assembly of Sections.

Sections are assembled in order, starting with Section 0, behind the read-in program. The last section must be followed by a card with non-zero punching on the Y-row.

To cut down duplication, cards for drum tracks 13/11 to 15/15 may be removed from all but one of the translated sections. Tracks are double field and have track Nos. x P1 on Y-row of first card.

25.2 Operation of Read-in Program.

The program reads in the sections, checks the assembly of cards and the use of subroutines, stores the sections on the drum, and then enters Section 0 at its first order.

As sections are read in a check is kept on the availability of drum storage space, and during reading the O.P.S. lights are used to indicate (x P1) the number of drum tracks left vacant.

25.3 Optional Punching.

A P32 on the I.D. causes all the P digits on the orders to be made operative.

Selective P digiting is possible by using such of the I.D. keys P1 to Pn + 1 as required to indicate selection of sections 0 to n. Thus, P2 and P4 will cause P digiting on sections 1 and 3.

25.4 Failures.

12 - 24 X - Sections incorrectly assembled. If the O.P.S. lights are all on error is in subroutines. A single shot causes program to return to read all sections in again.

9 - 24 looping - no room left on drum.

26. EASICODE 'TELESCOPE'.

EASICODE multi-section program packs can include Steam Tables, on α-field only, and many decimally-punched cards (graph data, etc); also, subroutines are often included in more than one of the sections involved.

It is therefore sometimes worthwhile to telescope these packs into an all binary double-field state, for use on production runs.

There are additional benefits in that the telescoped version is zero sum-checked after being read to drum, and it is a continuous read in.

Reference is made in the following paragraphs to multi-section programs only, but there is no reason why a single-section program should not be telescoped if it is thought worthwhile.

26.1 Telescoping an EASICODE Program.

If a program is allowed to proceed until all the input items required in the telescoped version have been read to drum, then the contents of the drum at that point will be the telescoped version, and the function of 'Telescope' is to punch this out (in the manner of a ZP.29T/3 drum postmortem) complete with the necessary read-to-drum routine, zero sum-check, and EASICODE entry routine.

A program which is to be telescoped must have the section(s) containing the read-to-drum (and f = 14, referred to below) orders which are obeyed before telescoping modified so that these orders are 'jumped' when the telescoped version enters the same section(s), otherwise data will be called for which is already on the drum.

The program must be stopped, preferably with an f=14 order, after the last relevant read-to-drum order has been obeyed, in the **EASICODE** section (usually 00) at the beginning of which it is required the telescoped version to start.

At this point the 'Telescope' pack (23 cards) should be initial-inputted.

26.2 The Telescoped Version.

After initial input 'Telescope' will punch out the telescoped version ready for use, comprising:-

1. Read-to-drum routine (ZP49T - 10 cards)
2. Triads (as ZP29T/3 post-mortem) containing the contents of the drum. The first word of Track 14/10 will be the negative drum sum.
3. ZP.49T 'end of read' parameter card (P.26 on α -field Y-row)
4. Zero sum-check and **EASICODE** entry triad..

and finish 9 - 24

(It is a good plan at this point to initial input this program to see if the zero sum check is successful, which will indicate that the program has been punched out correctly).

26.3 Using a Telescoped Program.

After initial input the triads are read to drum, and the contents of the drum zero sum-checked; if the check is successful the appropriate **EASICODE** section is entered.

If the drum does not sum to zero the program will stop 6-24X buzz; a **S.S.** will cause the check to be repeated.

Please Note.

The '**P**' digit optional punch facility is not available in a telescoped program.