THE ENGLISH ELECTRIC COMPANY LIMITED

DEPT.: DEVELOPMENT.

WHETSTONE.

REPORT No. W/AT 216
A.C.E.D. Ref. No.
Deptl. Ref.
Date 19th Sept.1958

Copies	TO:—	
1	Mr.R.H.Davies.	A.P.D.Whetstone.
2	Dr.J.Gardner.	"
3	Mr.R.W.Hockney.	"
4	Mr.M.J.Kelly.	"
5	Mr.B.Randell.	"
6	Library.	"
7-27	Spares.	"
28	Dr.P.Wakely.	M.E.L.Whetstone.
29	Mr.E.W.Worth.	"
30	Mr.N.Lam	"
31	Mr.W.E.Scott.	N.R.L.Stafford.
32	Mr.C.Robinson.	"
33	Dr.T.Deurden.	Electronics and Computing Dept. Warton.
34	Mr.J.Mallooh.	"
35	Dr.W.Hackett.	Maths, Physics Section, E.E.Co. Luton.
36	Dr.V.E.Price.	N.R.L. Marconi House, London.

Preliminary Report on EASICODE.

Report by
M.J.KELLY.
B.RANDELL.

1. SUMMARY.

EASICODE is a translation programme intended for use in most cases where a normal fixed-point D.E.U.C.E. programme could be written. It adopts the convention that all numbers have modulus less than one. When greater accuracy is required, the programmer must resort to other techniques such as floating-point arithmetic (which is used in most Interpretive Schemes.) Experience gained at Whetstone shows that such accuracy is seldom required and so EASICODE Mark I is designed to work in fixed-point.

Early tests indicate that an Easicode programme is two or three times slower than a conventional D.E.U.C.E.programme but several times faster than floating-point interpretive schemes. Consequently it will frequently be worth while to use EASICODE in preference to 'hand-writing' a programme in view of the speed of preparation obtained with EASICODE.

2 INDEX.

3. INTRODUCTION.

This is a preliminary report on a translation programme for D.E.U.C.E., called EASICODE, which is being developed at A.P.D., Whetstone.

There is a large class of problems, which, if programmed using existing floating point interpretive schemes, would use a prohibitive amount of machine time, yet which do not justify the writing of a conventional D.E.U.C.E. programme. EASICODE is a programme which can be used by programmers or others in order to produce a fixed point D.E.U.C.E. programme, without the usual time and effort this would take if programmed conventionally.

4. OUTLINE.

EASICODE is a translation programme that produces a D.E.U.C.E. programme from a set of decimal multi-address orders. It is not an interpretive scheme as the translated programme contains only D.E.U.C.E. instructions. The decimal orders are semi-symbolic in form as a mixture of real and symbolic addresses is employed in order to simplify programming to the extreme.

There are two main types of order : mathematical and organisational. The latter are automatically included in the programme, as are the arithmetic functions, and are always immediately available due to their being retained in the fast access store. Orders which require functions other than those referred to above use subroutines from the D.E.U.C.E. library. Thus the input to EASICODE consists of the decimal orders and any subroutines required by the programmer. The resulting fixed point D.E.U.C.E. programme is inevitably slower than a hand-written programme, as little attempt is made during translation to obtain the standard of optimisation such as is normally achieved by a D.E.U.C.E. programmer. However, it is worthy of note here that M.E.L. Whetstone are preparing a programme called AUTOCODE, which will optimumly code any programme. Thus it will be possible to convert a tested EASICODE programme to an optimised D.E.U.C.E. programme.

As speed is the main object fixed point working is maintained in preference to floating point, and all numbers used must be scaled to have modulus less than one. This introduces some inconvenience during programming, but by use of the scaling facilities provided this has been minimised. The diversity of orders permissible gives great scope to the ingenuity of the programmer without destroying the essential basic simplicity of the scheme.

Great consideration has been given to the question of input and output of data and consequently it has been decided that a wide range of punching arrangements should be included. Thus, problems of a data processing nature are also suitable for EASICODE programmes.

To enable physicists and engineers to use EASICODE with a minimum of assistance during programme testing it was decided that none of the testing facilities should require any knowledge of binary. This principle has been extended to cover the entire use of EASICODE.

5. STORAGE in D.E.U.C.E.

D.E.U.C.E. has two main storage levels; the high speed (which is the actual working store) and the backing store. To attain reasonable speed on the machine some control of the use of the former is vested with the programmer. This allows subroutines and data which are frequently required to be retained in the high speed store for as long as necessary.

This facility results in the need for fetch and store orders connecting the two levels of storage. These orders, though simple to write, are fully comprehensive and consequently great saving in machine time is achieved at the low cost of a few additional orders. The translator incorporates a large checking routine to ensure that the fast store does, in fact, contain the correct subroutines and data at each stage in the programme. This check, when performed without failure, will guarantee that the resulting programme contains no continuity errors which might arise from faulty logic on the part of the user, or just from mispunched order cards. Hence it is impossible for data to be obeyed as instructions or conversely for instructions to be used as data as can happen in conventional D.E.U.C.E. programmes.

Storage is required for the following three purposes :-

1. D.E.U.C.E. instructions - used in subroutines.
2. Numbers (modulus $\lesssim 1$).
3. Integers (modulus < 32768) - used in counting.

Each number, integer, or D.E.U.C.E. instruction occupies one word-space and the machine stores these in groups of 32, which are defined as BLOCKS. The user of EASICODE is allocated 6 blocks in the high speed store (A, B, C, D, E and F) and 192 in the backing store, numbered sequentially from 1. The 32 spaces in each block are numbered from 01 to 32. Thus A15 designates the 15th word space in block A of the high speed store. A00 is taken to mean the whole block A. Each block of numbers is cyclic in that just as position 1 is followed by 2, so 32 is followed by 1. Of the 6 fast blocks only 4 can be used to hold subroutines but all 6 are available for data storage. Note, however, that only the library subroutines introduced by the programmer need space in the four blocks possible as the standard functions as listed in Appendix I are permanently retained elsewhere. This means that the 6 blocks are used to store data and special subroutines which are in current and continuous use. The backing store contains permanent copies of all the library subroutines needed, so actually less than 192 blocks are available for data storage. It is reasonable to assume that the data storage comprises at least 160 blocks (i.e. over 5000 numbers).

6. ORDERS.

(i) Basic Form.

An order is basically of the form $f(x,y) \rightarrow z$, i.e. quantities denoted by x and y are operated on by function f, and the result placed in z. (In some cases the function is not mathematical in nature; for example - housekeeping orders like "Fetch Subroutine" which merely positions a subroutine in the fast store ready for use).

Depending on the function specified some of x, y and z may be unnecessary, in which case they are omitted from the order. The transaltor automatically checks for compatibility of f with x, y and z. This provides some check that f has not been mispunched.

An example of a simple order is that of multiplication. This is specified by f = 1

f	x	y	z
1	A03	B05	C17

This 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.

Similarly, for non-arithmetic functions.

e.g. f = 25 will read one decimal number from a card.

Thus	f	x	y	z
	25	-	-	B02

will read one number and place it in position 2 of block B.

To punch one result on a card the function No. 26 is used.

f	x	y	z
26	B19	-	-

will punch the number in block B position 19.

To read three numbers, multiply them together, and punch the result; the following orders could be used.

25	-	-	A01	Read No. to A01.
25	-	-	A02	" " " A02.
25	-	-	A03	" " " A03.
1	A01	A02	A04	A01xA02 placed in A04.
1	A04	A03	A05	A04xA03 placed in A05.
26	A05	-	-	Punch No. in A05

If the numbers are not required after their product is punched out, then the partial products can be placed in the positions occupied originally by the numbers.

Thus	1	A01	A02	A04

could be rewritten as

1	A01	A02	A01

and so on.

(ii) 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 specified symbolically by a number between 1 and 96, called its symbolic location (S.L.). Obviously no two orders can have the same symbolic location but any number of jumps can be made to any one order.

For example, f = 8 is the conditional jump for x = z or not. If x = z then the next order in sequence is obeyed, otherwise, the program takes as its next order the one specified by the symbolic next address (S.N.A.) element of the order.

This is best illustrated by demonstrating how this order would 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 programme to do this is as follows :-

Card No.	f	x	y	z	SL	SNA	
21	8	A13	-	000	-	11	Is A13 = 0?
22	25	-	-	A13	10	17	Road toA13.
23	26	A13	-	-	11	10	Punch A13.

The card numbers have been arbitrarily assigned. Order 21 checks if A13 is zero. (This function usually considers whether x = z or not. The above is a special form.)

a) If it is zero then the next order in natural sequence is obeyed - i.e. No.22, and this reads a fresh number into A13 as required. As this order has a S.N.A. punched, the programme now jumps to the order (not in the above three), which has a S.L. of 17.

b) If A13 is non-zero the natural sequence is broken and the order with S.L. of 11 is taken as next, i.e. No.23. This order punches A13 and due to the S.N.A. of 10 jumps to order 22 which has a S.L. of 10. The programme then continues as in the first case.

(iii) Scaling.

Due to the limits imposed on the size of numbers (i.e. modulus ≤ 1) it may prove necessary at certain points in the programme to scale the numbers in order that they should remain valid and also that full accuracy can be maintained.

The result of any mathematical operation can 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, and 10,

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

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

f	x	y	z	S
1	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 by 10, cause a number to exceed capacity, a failure indication is given (see Appendix II).

(iv) Complete Order Form.

At this stage it is necessary to describe the complete layout of an order.

Col. Nos.		
1-2	f	The function No., between 1 and 96
3-5	x	The first data address e.g. A15.
6-8	y	The second data address.
9-11	z	The result address.
12	s	parameter - for scaling, etc.
13-14	S.L.	Symbolic Location, between 1 and 96.
15-16	S.N.A.	Symbolic Next Address, between 1 and 96
17-19	M_1	See Modification of Orders.
20-22	M_2	
23-25	M_3	
26	P	Optional punch digit.
27	G	Stop digit.

Normally only f and such of x, y and z as are necessary need be punched. S is punched if the function needs a parameter. S.L. need only be punched if the order is a join point ; whilst S.N.A. is punched for conditional and unconditional jumps. The elements M_1, M_2, M_3 of the order are connected with order modification.

The final two elements i.e. Optional punch, and Stop digits are used mainly for programme testing (q.v.).

The elements x, y and z (and also M_1, M_2, M_3 - see order modification) are each of 3 digits, usually a block letter followed by a two-digit position number. However when using any of these elements to specify a drum block all three digits are used numerically.

7 MODIFICATION OF ORDERS.

It is frequently required to obey one or more orders repeatedly but with different data locations each time. This is made possible by employing one of the two methods of modification of orders which are incorporated in the Mark I version of Easicode. For cases where one operation is to be applied to a set of consecutive numbers forming part or whole of a block tabular modification is most suitable. If the modifications which must be made to the address specified in the order are not just of a simple tabular form the second system, called modify is used.

These facilities are called by specifying suitable parameters in the order.

(i) Tabular

Tabular modification is best illustrated by an example:-

Suppose we have a set of numbers in consecutive positions of a block - then this is defined as a column of numbers.

To multiply each number of a column, of length 10, starting at A04 by the corresponding numbers of a similar column, starting at B01 and place the resulting column of numbers in block A starting at position 19, the following order is sufficient:-

f	x	y	z	s	M_1	M_2	M_3
1	A04	B01	A19	-	010	010	010

The parameters M_1, M_2 and M_3, corresponding to x, y, z respectively, indicate the column length of the tabular working, by a two digit number, which must be ≤ 32. The zero preceeding this column length is to distinguish tabular working from the general modification of orders (q.v.).

In order to perform some operation on a constant and a column the M element corresponding to the data position of the constant is ignored.

Thus:-

f	x	y	z	s	M_1	M_2	M_3
1	A04	B01	A19	-	010	-	010

would cause each number in the column starting at A04 to be multiplied by the number in B01, with the resulting column of numbers being placed in A starting at position 19.

As has been previously stated each block is cyclic - hence a column of length 3 starting in A31 would consist of the numbers in A31, A32 and A01

The data address quoted in the x, y and z positions do not have to be different. For example to find the product of the 10 numbers in the column starting at C21 the following order will suffice:-

f	x	y	z	s	M_1	M_2	M_3
1	C21	C22	C21	-	-	009	-

The result is given in C21.

(ii) Modify

At times it is required to vary some or all of the addresses specified in an order. This can be done using Modify.

Suppose a programmer finds he needs to use one number in block A for a calculation, whose position in A depends on a parameter N (i.e. an integer). To enable him to use the number in A, position N directly without extracting it separately 'Modify' is used by punching, in the required M columns of an order, the address in the high speed store where the parameter is stored.

e.g.	f	x	y	z	s	M_1	M_2	M_3
	1	A32	D29	F32	-	C01	-	C01

The basic order reads:- A32 x D29 goes to F32.

Now by storing the parameter N in C01 and punching C01 in columns M_1 and M_3 the actual order obeyed is

A'N' x D29 goes to F'N'

This is because the integer stored in C01 is added (modulo 32) to the position numbers of x and z before the order is obeyed. Obviously any or all of x, y and z can be modified quite independently by using different parameters.

Although this system is more flexible than 'Tabular' it must be emphasised that the latter is very much faster in operation. 'Modify' may nearly double the time of an order in some cases but 'Tabular' results in nearly orthodox-Deuce timing, so wherever possible 'Tabular' should be used in preference to 'Modify'. Naturally 'Modify' and 'Tabular' cannot be used together but by always using the most suitable one a balance between ease of programming and economy of machine time is reached. 'Modify' is often very useful when used in conjunction with a 'Subroutine of Orders' (q.v.)

SUBROUTINES OF ORDERS.

Any set of orders which performs a calculation which is frequently required may be used as a subroutine by adding an extra order at the end. This order is f= 16 'End of subroutine' and its function is to re-enter the main programme at the correct point. To use such a subroutine there is another order 'Enter subroutine' which is an unconditional jump instruction:-

f	x	y	z	s	SL	SNA	M_1	M_2	M_3
15	-	-	-	-	-	R	-	-	-

This order enters the subroutine whose first order has a symbolic location R. Provision is made for re-entering the programme, after obeying subroutine "R", at the order following 'Enter subroutine'. This join point does not need a S.L.

DEUCE SUBROUTINES

When any mathematical function is required which is not in the standard pack the appropriate DEUCE subroutine may be used. There are however certain restrictions which must be obeyed.

(i) The translator must be given details of the subroutine in a standard form.

(ii) In the Mark I version of Easicode subroutines must be of no higher than second order.

(iii) No links (Except internal ones) are used, as subroutines have fixed exit points.

(iv) The contents of $17_{1,2,3}$ must always be preserved, and also 18 if the subroutine is to be used with tabular modification.

(v) In general only delay lines 5-11 (i.e. blocks A-F) are available.

(vi) Input data must be in $13,14,15,16$ or 21_3

(vii) Results must be either in 13,16 or a delay line.

It is recommended that subroutines be modified to use the arithmetic functions in control which are normally in delay lines 3 and 4. For subroutines which cannot conform to these rules it is necessary to add the extra D.E.U.C.E. programme to make them do so. There are some subroutines (e.g. Bessel Functions) which require more of the high speed store than is available by the above rules. In this case Delay lines 2-4 may be overwritten, and then replaced afterwards.

TRANSFER FUNCTIONS

(i) Between High-Speed Stores.

Transfers between high speed stores are accomplished using the order

f = 17 - 'Transfer'

e.g.	f	x	y	z
	17	A03		B20

will copy contents of A03 into position B20 . To effect a transfer of a column of numbers the tabular facility is used by specifying the column length in positions M_1 and M_3. If only M_3 is punched then A03 is copied into the column starting at B20. For clearing stores 000 is punched for x.

(ii) Between High-Speed and the Backing Store.

Transfers to and from the backing store (Drum) must be done in complete blocks, using f = 19 for fetching data from the drum and f = 18 for storing data on the drum. Normally these transfers concern only one block but up to 6 successive blocks can be transferred between high speed store and the drum, by specifying the number of blocks to be transferred in the parameter S.

Thus:-

f	x	y	z	s
18	A00	-	127	-

Transfers block A to block 127, whilst

18	A00	-	127	5

transfers blocks A-E to blocks 127-131 on the drum.

Normally the high-speed block to be transferred is specified by its block letter followed by two zeros but by specifying instead a position in the block a cyclic rotation can be made on a block as it is transferred.
this is illustrated by:-

f	x	y	z	s
18	A12	-	127	-

This transfers block A to block 127 after rotating it so that A12 is stored in position 1 in block 127, A13 in position 2 etc. The contents of A remain in their original positions.

(iii) Between Backing Stores.

This order f = 20 is included for the sake of completeness and copies block x into block z. Here the S parameter may be used to effect the copying of up to 9 consecutive blocks.

(iv) Fetch Subroutine.

f = 21 places a copy of (D.E.U.C.E.) subroutine x in its correct position in the high speed store.

1. INPUT AND OUTPUT OF DATA.

The facilities for reading and punching single numbers have already been described. A similar function (f = 24) , can be used for reading an integer to a position in the high speed store. When it is required to read or punch a set of numbers the orders 'General Read' and 'General Punch' should be used.

(i) General Read to High Speed Store.

This function is used for reading a column of numbers into a block in the high speed store starting at the position specified in 2. The column length ($\leqslant 32$) is given in M_3. The numbers may be punched 1,2,3,4,5 or 6 per card. The number per card must be specified by the S parameter.

Thus:-

f	x	y	z	s	M_1	M_2	M_3
27	-	-	A02	4	-	-	009

will cause 9 numbers to be read, 4 per card, into block A starting at A02.

If necessary the last card must be completed with zeros which will be ignored during the storage of the data.

(ii) General Punch from High Speed Store.

This is exactly similar to general read.

f	x	y	z	s	M_1	M_2	M_3
28	A02			4	009		

will punch, 4 per card the column of 9 numbers starting at A02.

(iii) General Read to Drum.

Numbers may be read to the drum, in blocks of 32, using f = 29.

For example:-

f	x	y	z	s	M_1	M_2	M_3
29			113	4			010

will read numbers, 4 per card, to 10 consecutive blocks on the drum, starting at block 113.

(iv) General Punch from Drum.

This is exactly similar to general read to drum.

Thus:-

f	x	y	z	s	M_1	M_2	M_3
30	113			4	010		

will punch, 4 per card, the contents of 10 blocks starting at block 113.

(v) Programme Constants.

Programme constants i.e. numbers and integers which are permanently required may be incorporated, during the translation stage, into a programme, by use of the order f = 23.

f	x	y	z	s
23	012		A03	1

will place the 12th constant in position A03. These constants are assembled in order immediately after the decimal orders.

s = 1 means the constant is a number (i.e. datum) and s = 2 means it is a parameter. (i.e. integer)

If any other value is given for s the translator indicates a failure.

2 MACHINE OPERATION

(i) Use of the Translator.

The translator takes its input in the following order.

1. Subroutines.
2. Decimal orders.
3. Program constants.

After these have been read in the programme is translated into D.E.U.C.E. instructions, and careful checking is carried out on compatibility of orders and use of the high speed store. Should the programme pass these checks the resulting D.E.U.C.E. programme will be punched out, followed by a set of decimally punched cards which is basically a flow diagram of orders but which includes the additional information required to make programme testing possible. If however, the checking routines disclose any errors a list of these is punched out in decimal.

(ii) Failures in Completed Programme.

A failure stimulates the alarm, illuminates one or more of the O.P.S. lights and enters the display routine. This routine shows in arabic numerals in D.L.11 a code number which, by reference to the tabulated flow diagram, is used to find the order being dealt with.

3. TESTING FACILITIES.

(i) Stopping the Programme.

A P32 on the I.D. will cause the programme to stop and display the code number in D.L.11.

A single shot will cause the programme to proceed. When the programme has entered this display routine the facilities of request stop , jump and pre- and post-mortem are available to the operator.

(ii) Request Stop.

This facility enables the operator to request stop on any code number by setting this number in decimal on the I.D. keys. Request Stop does not have any effect on the speed of operation of the programme.

(iii) Jump.

The operator can jump to a code number by setting this number in decimal on the I.D. keys.

(iv) Pre- and Post-Mortem

By jumping, in effect, to Code Number 0000, the operator can cause the contents of any or all the high speed blocks to be punched out in decimal. This facility does not interfere in any way with the storage at the disposal of the programmer, and when the punching has finished a single shot causes the programme to carry on (without any ill-effects) where it left off.

(v) Optional Punch Digit.

On any order which performs a mathematical function it is possible to use the optional punch digit facility to punch the result. This facility is intended for the punching of intermediate results whilst programme testing, as results should be punched out using one of the special punch facilities available. The optional punch digits are operative only if the resulting EASICODE programme is read in with a P32 on the I.D. Hence there is no need to retranslate the orders once programme testing is complete.

APPENDIX I

Function numbers 1-31 are reserved for routines permanently included in Easicode. D.E.U.C.E. subroutines introduced by the programmer are numbered in order 32, 33 etc. This is the number used in the 'Fetch Subroutine' order.

Functions already in Easicode are listed below with their function numbers.

1	Multiply	17	Transfer fast to fast store.
2	Divide	18	Transfer fast to slow store.
3	Add	19	Transfer slow to fast store.
4	Subtract	20	Transfer slow to slow store.
5	Modulus	21	Fetch Subroutine.
6	Count Integers	22	
7	Count Up To	23	Programme Constant.
8	Equal	24	Read One Parameter.
9	Big As	25	Read One Number.
10	Exceeds	26	Punch One Number.
11	Unequal	27	General Read to Fast Store.
12		28	General Punch from Fast Store.
13		29	General Read to Slow Store.
14	Buzz and Halt.	30	General Punch from Slow Store.
15	Enter Subroutine (Of orders)	31	Finish.
16	End of Subroutine (of orders)		

APPENDIX II

When a failure occurs the current Code number is displayed in the Monitor tube and an indication of the type of failure is given on the O.P.S. lamps.

The following configurations have so far been fixed:-

P1 Scale Up.

P2 Square Root.

P3 Division.

P4 Linear Interpolation.

P5.

P6.