Programs and software for the English Electric DEUCE.

Over the years, NPL and English Electric produced a large number of standard library routines, including matrix-handling routines, for DEUCE. There were also several *interpreter* systems in existence, including General Interpretive Program, GIP, Alphacode and Easycode. A description of all these systems is beyond the scope of this section.

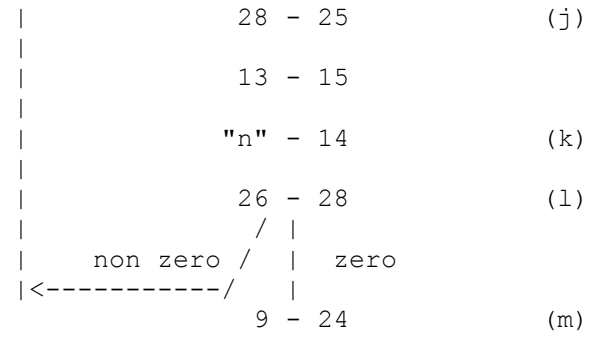
Below are given two examples of DEUCE machine-code programs, in order to give a general impression of the use of the instruction set. In the first (simpler) example, instructions are given in their basic form of <source> - <destination>, with the other fields (next-instruction address, characteristic, wait-number, timing number, etc.) omitted. In the second example, the instructions in the left-hand column are written in the basic <source> - <destination> form whilst those in the right-hand column have the timing details added. The second program contains two conditional branch instruction, whose alternate paths are shown in diagram form in the left-hand column. For both the first and second program, the annotations (a), (b), etc., refer to notes given at the end.

Example 1, taken from pages 40 and 41 of the English Electric DEUCE programming manual – see: http://users.tpg.com.au/eedeuce/pm.htm

To punch out results, up to 128 in number, on successive rows of several cards and then proceed to read in more data. The number of results is determined by a parameter "n", and they are stored in 7_0, 7_1, --- 7_{31}, 8_0 etc. up to 10_{31}. They will be referred to as A(0), A(1), --- A(i), --- A(n-1), the number "i" being stored in TS15.

```
                  30 - 15

                  10 - 24
      |---------------->|
      |              "I" - 13            (a)
      |
      |               15 - 14            (b)
      |
      |               14 - 25            (c)
      |
      |               23 - 14 (21 m.c.)  (d)
      |
      |               24 -14 ( 4 m.c.)   (e)
      |
      |               14 - 25            (f)
      |
      |               13 -  0 (m.c. 30)  (g)
      |
      |        Q30 A(i) - 29X            (h)
      |
      |        131   15 - 13             (i)
      |
```



NS-y-16/5-56

Notes.

(a) "I" is "1,7 - 29, 0,31,X".

(b) Place "i" in TS14, which has shifting facilities.

(c) Add "i" to the Wait number. Since the maximum value of "i" is 127, reaching the 23rd digit, this addition can never spill over into the Timing number, which starts in the 26th digit.

(d) Shift down 21 places, wiping out the lowest five digits of "i". What is left must be added to the Source number.

(e) Shift the result of instruction (d) into the Source number position.

(f) Add it to the instruction.

(g) The modified instruction enters TS COUNT in m.c. 30, so that its Wait and Timing numbers are respectively the minor cycles of transfer and of the next instruction.

(h) The modified instruction must be "1, (7 + a) - 29, b,31,X", where "i" has reached the value (32a + b) "b", in fact, is the lowest five digits of "i", and "a" the remaining digits, reduced to units of P5.

(i) The storage location of this instruction determines the N and T numbers of "I".

(j) Add P17 to "i". This happens in every cycle, so that TS15 contains "i" units of P17, ready to add into the Wait number without the need for any shifting.

(k) Place "n" in TS14. Since this is to be compared with the "i" x P17 in TS15. "n" must originally have been punched (or set on the

I.D.) in units of P17. This is very common practice.

(l) S26 gives zero only if the numbers in TS14 and TS15 are equal; that is if "i" and "n" are equal. The machine therefore proceeds back to instruction (a) if there are more results to be punched, and otherwise to instruction (m).

(m) Stop the Punch.

(n) Start the Reader.

Example 2: taken from pages 48 and 49 of the manual.

To multiply the numbers in TS14 and TS16, allowing for the sign of either.

DL2. m.c.			Coding.
(31)	$30 - 21_2$	(a)	m.c.
(19)	$14 - 21_3$	(b)	19 2, 14 - 21, 0, 0
(21)	0 - 24	(c)	20 2, 16 - 27, 0, 1
(25)	14 - 27	(d)	21 2, 0 - 24, 0, 2
	/ \		
	+/ \-	(e)	22 2, 13 - 23, 1, 2
	/ \		
(29)	30 - 13(30) 16 - 13	(f)	23 2, 30 - 29, 0,29
	\ /		
	\ /		24 2, 14 - 25, 0,28
	\ /		
(20)	16 - 27	(g)	25 2, 14 - 27, 0, 2
	/ \		
	+/ \-		26 2, 21 - 22,1,(32-2n),31
	/ \		
(23)	(h) dummy (24) 14 - 25	(i)	27 1, 29 - 23, 1,1
	\ /		
	\ /		28
	\ /		
(22)	(j) $13 - 23_3$(after mult)		29 2, 30 - 13 0,21
(26)	(k) 21 - 22 (2n m.c. e,o)		30 2, 16 - 13 0,20
(27)	(l) $29 - 23_2$		31 2, 30 - 21, 1,18

1_{30} Next part of programme.

Notes.

(a),(b),(c) Start multiplication as before.

(d) to (i) Build up the sign correction ready to subtract into the top half of the product after multiplication. The correction consists of zero if both factors are positive, one factor if only the other is negative, and the sum of both factors if both are negative.

(h) The coding must allow a time of at least two major cycles between (c) and (j). If the dummy were omitted, a third major cycle would elapse between (g) and (j) if the number in TS16 were positive.

(j) Subtract in the correction. This happens in m.c. 25 (see coding), whereas multiplication started in m.c. 23 two Major Cycles before. In other words, we are just all right.

(k) Shift the product up n places. n can be any number from 1 to 16, and is determined by the Wait number of the instruction (in m.c. 26). This allows for binary places. When working to 27 b.p., for example, the product of two numbers with 27 b.p. each has 54 b.p. The binary point after multiplication comes between P22 and P23 of the top half. Shifting up by five b.p. brings the point between P27 and P28 of the top half. Taking this top half as the answer leaves us still with 27 b.p.

(l) Subtract P32 into 21_2. Well, that's what it looks like. The actual effect is to add P32 into 21_2. Remember that a transfer to D22 or D23 in an even m.c. only, with TCB off, of a number which has a "1" in P32 position, is automatically followed by the transfer of 32 "1" s to the same destination. This means that P32 (even) when sent to D22 or D23 looks just like -P32 (even) so that in order to add it you have to subtract. The object is to balance the error, as was done at the end of the division and square root examples. It is assumed that only the top half of the product is to be taken on to the next stage of the calculation. Without this preliminary addition of P32 (even) the error caused by dropping the bottom half would vary from zero to -1 in the bottom binary place. With the round off, the resultant error varies from -½ to +½ in the bottom place.

The coding of this example may appear a little eccentric, but there is method in the madness. The salient points are that all the instructions are crammed as tightly as possible into bottom of DL2 (except for the gap at 2_{28} which will be explained) and that the last instruction of the multiplication, 2_{27}, leads on to 1_{30} for the first instruction in the next part of the programme. The reason for both of these manoeuvres will appear in section 7.1.