'ENGLISH ELECTRIC'

DEUCE

LECTURE NOTES

1 - 30

DATA PROCESSING AND CONTROL SYSTEMS DIVISION
THE ENGLISH ELECTRIC CO. LTD.
KIDSGROVE, STOKE-ON-TRENT.

Norton
Gote Lane
Ringmer
Nr Lewes
E Sussex
BN8 5HX
26 Feb 13

Dear Peter,

Here's that EDSAC manual as mentioned by email – hope it's of some interest. Definitely a donation.

Best wishes,

Des Watson

DEUCE PROGRAMMING COURSE.

INDEX OF LECTURES.

Deuce Programming Practice - graded exercises for students to programme and test on the machine during the course. Students should aim to complete exercises 1, 2, 3, or 1 and 4, depending on the type of work they intend to do.

LECTURE 1

1.1 GENERAL INTRODUCTION.

The DEUCE is a stored programme digital computer capable of carrying out at high speed the operations of arithmetic, logic and information transfer. In common with other high speed computers, the DEUCE can only perform these operations when it is instructed to do so.

The first task in getting any useful work out of a computer is the presentation of the problem to the machine in a form and language which it can understand. A problem may take many forms. It may be the evaluation of an expression written in mathematical symbols, in which case the machine is used as a calculator. On the other hand the problem may be a commercial activity such as the control of stock or the preparation of invoices. It is customary to separate these two main types of problem into scientific and commercial applications. In the former case the data may be the result of measurements in an experiment or the coefficients of equations arising in a design calculation. In the latter case the data may be the quantity of some commodity held in a warehouse or the outstanding debt on the ledger account of a customer. In both cases the machine may be said to act as a data-processor, for data is "something you are given" and processing means "doing something with it". Whatever the type of problem, it can only be successfully handled by a computer if it is first subjected to three necessary operations.

These are:
(a) Problem Analysis.
(b) Programming.
(c) Coding.

The definition given to analysis is common to all computers. It means thinking about the problem and formulating it in such a way that the objective is clearly known and specified. In mathematical and technical work this formulation frequently involves a formula and the objective is clearly defined. In commercial data processing no such formula may be available and the problem must be specified as a series of statements outlining the problem requirements. For either type of problem it is essential that it shall be understood, for if it is not, then there is no hope of enlisting the aid of a computer in its solution. Here we have the first axiom of computer operation, that no one can do on a computer what he does not know how to do by other means.

The terms "Programming" and "Coding" have different meanings for different computers. For the DEUCE, programming means the preparation of a plan of the method of work to be employed in a given problem and the translation of this plan into machine instructions. These instructions are written in a machine language or code and are prepared on a diagram known as an "Instruction Flow Diagram". The broad plan of the method of work may be prepared either as a series of statements or in diagrammatic form as a "Block Schematic Diagram" or "Flow Chart".

After the machine instructions have been written they must be prepared for punching on cards before entry to the machine. This operation is known as "Coding". In DEUCE the operations of instruction writing, i.e., making a flow diagram of instructions, and coding are closely associated. A beginner would be well advised to separate these processes, learning first to do the one and then the other. In time, as his knowledge of DEUCE programming increases he will realise that a knowledge of coding is helping to produce more efficient flow diagrams.

In the definition of DEUCE given in the first sentence of this introduction two terms have been used which may require explanation. These are "stored-programme" and "digital". A machine is said to be of the stored-programme type when all the instructions for a problem are held within the machine. Once such a machine has been loaded with a programme it can proceed through the steps of the calculation at high speed and without the need for human intervention. The term digital refers to the manner in which information is represented within the computer. Numbers, instructions, alphabetic descriptions are all contained in DEUCE as strings of electrical pulses representing the digits of coded information. DEUCE works in the BINARY CODE and an understanding of this is essential to efficient programming and operation of the machine.

1.2 REPRESENTATION OF INFORMATION.

1.2.1 Scale of Notation.

For everyday use it is common practice to use the decimal (or denary) scale of notation to represent numbers. Ten symbols or cyphers are used to convey the notion of no things, one thing, two things, etc., up to nine things, and these are written

$$0, 1, 2, 3, \ldots 9.$$

When it is necessary to convey the notion of more than nine things more than one cypher must be used. The order in which these cyphers are written denotes the order of the quantity represented by the cypher.

As an example

$$5327$$

means the number which is five thousands added to three hundreds added to two tens added to seven units. The 7 representing seven units is placed on the right in the least significant position and the 5 representing five thousands is placed on the left in the most significant position.

More explicitly,

$$5327$$

stands for the sum

$$5.10^3 + 3.10^2 + 2.10^1 + 7.10^0$$

There is no single cypher to convey the notion of ten. This is represented by the pair of cyphers 10 and ten is known as the radix of the denary scale.

If we use a radix of seven, no single cypher will be used to convey the notion of seven. Again we use the pair of cyphers 10 and this time we need only seven single cyphers to convey the notions of numbers of things from nought to six.

In this scale 354 stands for

$$3.7^2 + 5.7^1 + 4.7^0$$

1.22 The Binary Scale of Notation.

Although the binary scale has been known for centuries, (it was in use in ancient China,) its modern importance derives from its use in electronic computers. In this scale only two cyphers are needed, 0 and 1 representing the notions of no things or one thing. The radix is two and all numbers are counted as sums of two and powers of two.

The binary number 11001 stands for

$$1.2^4 + 1.2^3 + 0.2^2 + 0.2^1 + 1.2^0$$

which is the quantity twenty-five.

The first eight common numbers are represented in the binary scale as

0 = 0	4 = 100
1 = 1	5 = 101
2 = 10	6 = 110
3 = 11	7 = 111

The rules of addition in the binary scale are quite simple.

$$0+0 = 0; \ 1+0 = 1; \ 0+1 = 1;$$

are self evident; 1+1 = two which is written as 10 and from this 1+1 = 0 with a one to carry to the next higher place.

Note that if a binary number consists of a string of ones as in 1111 (= 15), and one is added at the least significant position, all the digits of the original number become zero and a one appears at the most significant end.

$$\text{e.g.}\quad \begin{array}{r} 1111\ + \\ 1 \\ \hline 10000 \\ \hline \end{array}$$

If this addition takes place in a binary computer which has room for only four places of significance in its registers, the one appearing at the most significant end will be lost and the apparent result will be zero. Now adding one to something and producing nothing leads us to suggest that the "something" is minus one and indicates a method of representing negative numbers.

1.2.3 Binary Non-Integers.

In the denary scale, non-integers are represented as tenths, hundredths, etc., and

$$5.10^2 + 3.10^1 + 2.10^0 + 7.10^{-1} + 4.10^{-2}$$

is written as

$$532.74$$

The indicator . denotes the point where the indices of the radix change from positive to negative. The quantity to the left of the indicator (the decimal point) is the integral or whole number part of the number. The quantity 74 to the right of the decimal point represents the fractional (non-integer) part. We say that we have two decimal places.

Similarly, non-integers may be represented in binary in quantities of halves, quarters, eighths, etc., and the number

$$1101.101$$

stands for

$$1.2^3 + 1.2^2 + 0.2^1 + 1.2^0 + 1.2^{-1} + 0.2^{-2} + 1.2^{-3}$$

or

$$13 + \tfrac{1}{2} + \tfrac{1}{8} = 13\tfrac{5}{8}.$$

The first eight binary places are represented as

$2^{-1} = 1/2 = 0.1$	$2^{-5} = 1/32 = 0.00001$
$2^{-2} = 1/4 = 0.01$	$2^{-6} = 1/64 = 0.000001$
$2^{-3} = 1/8 = 0.001$	$2^{-7} = 1/128 = 0.0000001$
$2^{-4} = 1/16 = 0.0001$	$2^{-8} = 1/256 = 0.00000001$

In decimal, multiplying by ten involves moving the number up (left) one place and adding a nought.

e.g. 523 x 10 = 5230

In binary, moving a number up one place and adding a nought corresponds to multiplying by two

$$1101 \times 2 = 11010$$

In decimal this corresponds to moving the decimal point to the right and likewise in binary the process is the same, moving the binary point to the right

$$1101.00 \times 2 = 11010.0$$

Division by two in the binary scale involves moving the binary point to the left

$$110.01 = 6.25 = 6\ 1/4$$
$$110.01 \div 2 = 11.001 = 3.125 = 3\tfrac{1}{8}$$

1.2.4 Binary Numbers in DEUCE.

Binary numbers in DEUCE are represented by trains of pulses representing ones and spaces representing noughts. Each number group has room for 32 pulses or spaces and the binary numbers in DEUCE thus have 32 digit positions. The digits follow one another in time sequence, the least significant digit appears first followed by the next and so on, until the arrival of the 32nd digit completes one number. The next digit to appear will be the first of the same or another number, and the timing circuits of the DEUCE ensure that the numbers are not in danger of being mixed up.

The digit positions are conveniently numbered from 1 to 32 and we speak of the digit position as

$$P_1\ P_2\ P_3\ \cdots\ \cdots\ P_{30}\ P_{31}\ P_{32}$$

P_1 is the least significant position and P_{32} is the most significant position. It will be noticed that the order of significance is reversed. As many good reasons can be put forward for this method of representation as for the more conventional right to left order. From this point all binary numbers will be written in the reverse order. This should not prove an inconvenienceas you are probably learning binary for the first time and it is as easy to learn the reverse method as any other.

The first 32 binary numbers are written in the DEUCE convention below and all these must be committed to memory without delay.

0 = 00000	8 = 00010	16 = 00001	24 = 00011
1 = 10000	9 = 10010	17 = 10001	25 = 10011
2 = 01000	10 = 01010	18 = 01001	26 = 01011
3 = 11000	11 = 11010	19 = 11001	27 = 11011
4 = 00100	12 = 00110	20 = 00101	28 = 00111
5 = 10100	13 = 10110	21 = 10101	29 = 10111
6 = 01100	14 = 01110	22 = 01101	30 = 01111
7 = 11100	15 = 11110	23 = 11101	31 = 11111

1.2.5 Binary Arithmetic.

Addition of binary numbers is quite simple and is frequently of use in modification of DEUCE instructions. The rules, already given, are

0+0 = 0, 1+0 = 1, 0+1 = 1, 1+1 = 0 and carry 1

Example	13 +	1011
	12	0011
	25	10011

Subtraction of binary numbers is also straightforward. The rules are

1-0 = 1, 1-1 = 0, 0-1 = 1 and borrow 1.

```
Example        25 -      10011
               12        00110
               --        -----
               13        10110
```

Multiplication of binary numbers is considerably simpler than decimal multiplication. There are no multiplication tables to learn. To multiply 19 by 13 proceed as follows:

```
               19 =      11001
               13 =      1011
                         -----
(a)                      11001
(b)                       00000
(c)                        11001
(d)                         11001
                         --------
                         11101111
                         --------
```

(a) There is a one in the units position of 13 so we copy 19 in the units position of the partial product.

(b) The digit in the 2's position of 13 is zero so we make no copy of 19 in the 2's position. This step would normally be omitted.

(c) The next digit of 13 is a one and we make a copy of 19 starting at this digit position.

(d) The next digit of 13 is also a one and again we make a copy of 19 with its least significant digit in the appropriate position. All the copies of 19 are added together to form the result.

Division of binary numbers is rarely necessary for preparing work for DEUCE. The usual method long division is used and further mention of division will be made later in dealing with the DEUCE divider.

1.2.6 Scale of 32 Notation.

Since binary numbers in DEUCE have 32 digits it would be both time consuming and exhausting to speak of or write 32 digits every time we refer to a number. A compressed notation has been agreed upon as a useful convention to simplify operations on binary numbers.

Suppose the binary number is

1011011011010110011001011

This is divided into groups of five digits each

10110/11011/01011/00110/01011

The value of each group is written in place of the group with the group numbers in order from the left.

We thus have

13/27/26/12/26

The separator lines are omitted and this is finally written as

13, 27, 26, 12, 26.

This is a standard method of representing binary numbers. All groups consist of five binary digits except where a number represents a number in DEUCE. As there are 32 digits in a

DEUCE number this is represented by six groups of five digits and a top group of two digits. The number in DEUCE

11000, 11010, 11011, 10110, 11011, 00011, 10

is written

3, 11, 27, 13, 27, 24, 2.

1.2.7 Equivalent Numbers of Binary and Decimal Places.

It will have become apparent that binary numbers are longer than their decimal equivalents. To estimate the ratio of binary digits to decimal digits necessary to represent any given quantity, let the quantity Q be represented as 10^d with d decimal digits and 2^b with b binary digits.

Then $Q = 10^d = 2^b$

or $d = b \log 2$

Then $b/d = 1/\log 2 = 1/.3010 \triangleq 3.$

Roughly three times as many places are needed in the binary representation of a number as in the corresponding decimal representation.

1.2.8 DEUCE Sign Conventions.

The largest integer which can be accommodated in one DEUCE register is

$$P_1\ P_2\ P_3 \ldots \ldots P_{32}$$

with a one in each P position.

This has the value $2^{32} - 1$. If two numbers are added together in DEUCE and their sum exceeds $2^{32} - 1$, the digits which should occupy positions P_{33}, P_{34} etc., are lost since no provision is made for recording digits in these positions. Working with positive numbers only, DEUCE can handle numbers in the range 0 to $2^{32} - 1$.

Exactly half these numbers from 0 to $2^{31} - 1$ have a zero in P_{32} and the remainder, from 2^{31} to $2^{32} - 1$, have a one in the P_{32} position. All the numbers in the range 0 to $2^{31} - 1$ may be taken as positive numbers and the remainder, with a one in the P_{32} position acting as a sign digit, are usedas a range of negative numbers. In section 1.2.2 it has been suggested that a string of ones represents minus one since adding one to such a string gives a zero result. The actual value of any negative number is thus not the value of the first 31 digits, with P_{32} acting as the sign.

If $11111111111111111111111111111111 = 2^{32} - 1$

plus $1 = 1$

$= \overline{00000000000000000000000000000000}$ /1 lost

Then $2^{32} - 1 = -1.$

In general, $2^{32} - x = -x,$

where x is any number in the range 0 to $2^{31} - 1$.

The reader should check that this is a consistent system of positive and negative number representation by working with a six-digit model DEUCE.

e.g. $1 = 100000 \qquad -1 = 111111$

$5 = 101000 \qquad -5 = 000000 -$

101000

$\overline{110111}$

now $2^6 - 5 = 000000/1$

$101000/$

$\overline{110111}$

Similarly, 0 - (-5) should give +5. This may be checked as follows

```
000000/
110111/
101000/  1(lost)
```

The number range of DEUCE integers is thus

$$\boxed{-2^{31} \leqslant x < 2^{31},}$$

corresponding to positive and negative numbers of about ten decimal digits.

One useful fact associated with the relation of positive and negative binary numbers should be noted. Take any number in binary code x and change all the noughts to ones and the ones to noughts. This gives a number known as "not x" written as $\sim x$ (or $\bar{x}$). Now add x to $\sim x$ and it will be apparent that we produce a string of ones, i.e., minus 1.

Therefore, $x + \bar{x} = -1$ or

$$-x = \bar{x} + 1$$

This is illustrated for a six bit model by the following example:

$x = 6 =$	011000
$\bar{x} = 6 =$	100111
$x + \bar{x} = (-1)$	111111

From this simple observation we have a rapid method of recognising the negative of any number. The rule is "To form the negative of any number, form the 'not' of the number and add one in the least significant place".

This process can also be done with the compressed notation, using the groups of five method of writing binary numbers.

$$10^4 = 16,\ 24,\ 9,\ 0,\ 0$$
$$\sim 10^4 = 15,\ 7,\ 22,\ 31,\ 31$$

which has been formed by subtracting each group from 31.

Then $-10^4 = 16, 7, 22, 31, 31.$

This is equivalent to subtracting the least significant group from 32 and subtracting the other groups from 31.

1.2.9 Binary Coded Decimal Notation.

It is sometimes necessary to work in the half world of binary and decimal. This may be done by representing each digit of a decimal number by its binary equivalent and placing these binary coded decimal digits one after the other as follows.

Using four binary digits for each decimal digit, the number

1010/1100/0000/1110

represents 5 3 0 7

or, allowing for our DEUCE reversed order, the number

7,035

This method of representation is said to be in 4-bit B.C.D., meaning Binary Coded Decimal using four binary digits per decimal digit. Each four bit group constitutes a character. Using four digits for each group it is possible to represent 15 different characters. As this is not enough to cover the decimal digits and the letters of the alphabet, it becomes necessary to use a 6-bit group code.

Such a code is used in DEUCE in connection with the 80-column Input/Output machine and for paper-tape. The DEUCE 6-bit code is given in section 1.2.12.

1.2.10 Useful Constants.

All values expressed in the scale of 32, with the most significant digit on the right.

10	.10
10^{2}	. 4, 3
10^{3}	. 8, 31
10^{4}	.16, 24, 9
10^{5}	. 0, 21, 1, 3
10^{6}	. 0, 18, 16, 30
10^{7}	. 0, 20, 5, 17, 9
10^{8}	. 0, 8, 24, 11, 31, 2
10^{9}	. 0, 16, 18, 21, 25, 29
10^{10}	. 0, 0, 25, 23, 0, 10, 9
10^{11}	. 0, 0, 26, 13, 7, 4, 29, 2
10^{12}	. 0, 0, 4, 10, 10, 10, 3, 29

10^{-1}	26, 12, 6, 19, 25, 12, 6, 3.
$2^{5} .10^{-2}$	24, 21, 7, 10, 24, 21, 7, 10.
$2^{5} .10^{-3}$	25, 11, 26, 13, 18, 24, 0, 1.
$2^{10} .10^{-4}$	28, 24, 26, 5, 14, 27, 8, 3.
$2^{15} .10^{-5}$	14, 28, 8, 12, 13, 17, 15, 10.
$2^{15} .10^{-6}$	27, 2, 20, 23, 23, 17, 1, 1.
$2^{20} .10^{-7}$	29, 21, 6, 5, 31, 11, 11, 3.
$2^{25} .10^{-8}$	2, 6, 2, 23, 3, 19, 23, 10.
$2^{25} .10^{-9}$	19, 0, 29, 11, 16, 11, 2, 1.
$2^{30} .10^{-10}$	24, 14, 22, 31, 13, 30, 13, 3.
$2^{35} .10^{-11}$	6, 15, 1, 31, 31, 26, 31, 10.
$2^{35} .10^{-12}$	23, 4, 16, 25, 28, 5, 3, 1.

$\frac{1}{2}$	0, 0, 0, 0, 0, 0, 0, 16.0
$\frac{1}{3}$	21, 10, 21, 10, 21, 10, 21, 10.0
1/5	19, 25, 12, 6, 19, 25, 12, 6.0
1/7	18, 4, 9, 18, 4, 9, 18, 4.0
1/9	18, 3, 7, 14; 28, 24, 17, 3.0
1/11	29, 2, 29, 2, 29, 2, 29, 2.0
1/13	17, 29, 4, 22, 19, 24, 14, 2.0
1/15	17, 8, 4, 2, 17, 8, 4, 2.0
1/17	15, 24, 3, 30, 16, 7, 28, 1.0

$\frac{1}{2}!$	0,	0,	0,	0,	0,	0,	0,	16.
$\frac{1}{3}!$	11,	21,	10,	21,	10,	21,	10,	5.
$\frac{1}{4}!$	11,	21,	10,	21,	10,	21,	10,	1.
1/5!	2,	17,	8,	4,	2,	17,	8,	0.
1/6!	27,	2,	12,	11,	16,	13,	1,	0.
1/7!	13,	0,	20,	1,	16,	6,	0,	0.
$\frac{1}{8}!$	2,	16,	6,	0,	26,	0,	0,	0.
1/9!	7,	30,	14,	28,	2,	0,	0,	0.
1/10!	20,	28,	7,	9,	0,	0,	0,	0.
1/11!	25,	28,	26,	0,	0,	0,	0,	0.
1/12!	23,	7,	2,	0,	0,	0,	0,	0.
1/13!	17,	5,	0,	0,	0,	0,	0,	0.
1/14!	13,	0,	0,	0,	0,	0,	0,	0.
1/15!	1,	0,	0,	0,	0,	0,	0,	0.
π	6,	4,	2,	21,	22,	31,	16,	4.3
e	11,	20,	24,	2,	21,	16,	31,	22.2
$\sqrt{2}$	20,	31,	25,	12,	30,	4,	8,	13.1
γ	30,	27,	24,	15,	6,	2,	15,	18.0
$\sqrt{3}$	5,	12,	1,	29,	26,	19,	13,	23.1
$1/\sqrt{3}$	2,	4,	11,	20,	19,	6,	15,	18.0
$\log_{10}e$	28,	4,	21,	24,	30,	22,	28,	13.0
$\log_e 10$	11,	21,	29,	14,	3,	27,	21,	9.2
$\log_2 e$	24,	21,	20,	12,	7,	10,	5,	14.1
$\log_e {}^2$	18,	30,	29,	15,	1,	25,	5,	22.0

π = 3.141, 592, 653, 6

e = 2.718, 281, 828, 46

n	2^n	n	2n
1	2	36	6 87194 76736
2	4	37	13 74389 53472
3	8	38	27 48779 06944
4	16	39	54 97558 13888
5	32	40	109 95116 27776
6	64	41	219 90232 55552
7	128	42	439 80465 11104
8	256	43	879 60930 22208
9	512	44	1759 21860 44416
10	1024	45	3518 43720 88832
11	2048	46	7036 87441 77664
12	4096	47	14073 74883 55328
13	8192	48	28147 49767 10656
14	16384	49	56294 99534 21312
15	32768	50	1 12589 99068 42624
16	65536	51	2 25179 98136 85248
17	1 31072	52	4 50359 96273 70496
18	2 62144	53	9 00719 92547 40992
19	5 24288	54	18 01439 85094 81984
20	10 48576	55	36 02879 70189 63968
21	20 97152	56	72 05759 40379 27936
22	41 94304	57	144 11518 80758 55872
23	83 88608	58	288 23037 61517 11744
24	167 77216	59	576 46075 23034 23488
25	335 54432	60	1152 92150 46068 46976
26	671 08864	61	2305 84300 92136 93952
27	1342 17728	62	4611 68601 84273 87904
28	2684 35456	63	9223 37203 68547 75808
29	5368 70912	64	18446 74407 37095 51616
30	10737 41824	65	36893 48814 74191 03232
31	21474 83648	66	73786 97629 48382 06464
32	42949 67296	67	1 47573 95258 96764 12928
33	85899 34592	68	2 95147 90517 93528 25856
34	1 71798 69184	69	5 90295 81035 87056 51712
35	3 43597 38368	70	11 80591 62071 74113 03424

<u>NEGATIVE POWERS.</u>

2^{-3}	.125	$10^5\ 2^{-8}$	390.625
2^{-4}	.0625	$10^5\ 2^{-10}$	97.65625
2^{-5}	.03125	$10^5\ 2^{-16}$	1.52587 89062 5
2^{-6}	.01562 5	$10^{10}\ 2^{-32}$	2.32830 64365 38696 28906 25
2^{-7}	.00781 25		
2^{-8}	.00390 625		

1.2.12 Binary Coding of Alpha-Numeric Characters in DEUCE.

Card Input Code.	6-Bit Equivalent (Decimal)	6-Bit Equivalent (Binary).	Associated Symbol.	Card Output Code.
YX08 other		(Most Sig. on right)		YX08 other
..0..	0	0000 00	0	..0..
....1	1	1000 00	1	1
....2	2	0100 00	2	2
....3	3	1100 00	3	3
....4	4	0010 00	4	4
....5	5	1010 00	5	5
....6	6	0110 00	6	6
....7	7	1110 00	7	7
...8.	8	0001 00	8	...8.
....9	9	1001 00	9	9
...82	10	0101 00	10	...82
...83	11	1101 00	11	...83
...84	12	0011 00	12	...84
...85	13	1011 00	13	...85
...86	14	0111 00	Free	...86
Blank	15	1111 00	Space	Blank
Y....	16	0000 10	Plus	Y....
Y...1	17	1000 10	A	Y...1
Y...2	18	0100 10	B	Y...2
Y...3	19	1100 10	C	Y...3
Y...4	20	0010 10	D	Y...4
Y...5	21	1010 10	E	Y...5
Y...6	22	0110 10	F	Y...6
Y...7	23	1110 10	G	Y...7
Y...8	24	0001 10	H	Y...8
Y...9	25	1001 10	I	Y...9
Y..82	26	0101 10	Free	Y..82
Y..83	27	1101 10	Free	Y..83
Y..84	28	0011 10	Free	Y..84
Y..85	29	1011 10	Free	Y..85
Y..86	30	0111 10	Free	Y..86
Y..87	31	1111 10	Free	Y..87
.X...	32	0000 01	Minus	.X...
.X..1	33	1000 01	J	.X..1
.X..2	34	0100 01	K	.X..2
.X..3	35	1100 01	L	.X..3
.X..4	36	0010 01	M	.X..4
.X..5	37	1010 01	N	.X..5
.X..6	38	0110 01	O	.X..6
.X..7	39	1110 01	P	.X..7
.X.8.	40	0001 01	Q	.X.8.
.X..9	41	1001 01	R	.X..9
.X.82	42	0101 01	Free	.X.82
.X.83	43	1101 01	Free	.X.83
.X.84	44	0011 01	Free	.X.84
.X.85	45	1011 01	Free	.X.85
.X.86	46	0111 01	Free	.X.86
.X.87	47	1111 01	Free	.X.87
No Code	48	0000 11	Free	..0..
..0.1	49	1000 11	Free	..0.1
..0.2	50	0100 11	S	..0.2
..0.3	51	1100 11	T	..0.3
..0.4	52	0010 11	U	..0.4
..0.5	53	1010 11	V	..0.5
..0.6	54	0110 11	W	..0.6
..0.7	55	1110 11	X	..0.7
..08.	56	0001 11	Y	..08.
..0.9	57	1001 11	Z	..0.9
..082	58	0101 11	Free	..082
..083	59	1101 11	Free	..083
..084	60	0011 11	Free	..084
..085	61	1011 11	Free	..085
..086	62	0111 11	Free	..086
..087	63	1111 11	Space/Ignore	Blank

CLASS EXERCISES.

1. Write the number 163 in

   (a) binary

   (b) octal (scale of eight)

   (c) scale of 32 Place the IS digit or group on the left.

   (d) 4-bit B.C.D.

   (e) 6-bit B.C.D.

2. Write the number 0.05625 in

   (a) binary (32 digits signed with 16 binary places)

   (b) binary, 30 binary places

   (c) scale of 32 for DEUCE working with 30 b.p.

3. From the table of useful constants write the compressed form of the following numbers in a form suitable for DEUCE

   (a) $2^5 . 10^{-2}$ to 30 b.p.

   (b) $2^{15} . 10^{-4}$ to 25 b.p.

   (c) $2^{31} . 10^{-4}$ integral part

4. Use the table of useful constants to write the compressed form of

   (a) $2^3 . 10^4$

   (b) $2^{21} . 10^2$

   (c) $2^6 . 10^7$

5. If P_3 represents 2^2 write the values of

   $$P_{10} \quad P_{15} \quad P_{23} \quad P_{29}$$

   What P position corresponds to 2^3 2^{12} 2^{14} 2^{25}?

6. Using a six-digit model DEUCE, perform the following operations

   (a) Add 13 and 9 in binary

   (b) Add 25 and-8 in binary

   (c) Subtract 13 from 23 in binary

   (d) Subtract 15 from 13 in binary

   (e) Write the negative of +27, -13, +7, -2

   (f) Check each of (e) by an alternative method.

7. Using the table of useful constants, write out the binary and compressed binary for each of the following

   (a) 1/10th to 5 b.p.

   (b) 1/10th to 12 b.p.

   (c) 1/10th to 15 b.p.

   (d) 1/7th to 20 b.p.

8. Write the binary patterns for

   (a) (1/3)rd to 30 b.p.

   (b) (2/3)rds to 30 b.p.

   (c) Add these together

   (d) If the sum of (a) and (b) is not 1, what should be done to either (a) or (b) to make the sum one.

9. Write out the patterns for each of the following in DEUCE six-bit code (least significant at Left)

   (a) 76324

   (b) -4321

   (c) DEUCE

(d) Names.

10. Write out the binary (five digits) for

(a) 19
(b) 25
(c) 27
(d) 13
(e) 26
(f) 31
(g) 15
(h) 30
(i) 10
(k) 23.

Exercise Solutions.

1.	(a)	11000101	
	(b)	3, 4, 2	
	(c)	3, 5.	
	(d)	1100 : 0110 : 1000	
	(e)	110000 : 011000 : 100000.	
2.	(a)	00 00 00 00 00 01 01 00 . 00 00 00 00 00 00 00.00	
	(b)	00 00 00 - - - - - - - - 00 01 01 00.00	
	(c)	0, 0, 0, 0, 0, 5.0	
3.	(a)	8, 10, 24, 21, 7, 10.0	
	(b)	27, 5, 14, 27, 8, 3.0	
	(c)	28, 22, 17, 6, 0, 0.0	
4.	(a)	0, 4, 14, 2, 0, 0.0	
	(b)	0, 0, 0, 0, 8, 6.0	
	(c)	0, 0, 8, 11, 2, 19.0	
5.		2^9, 2^{14}, 2^{22}, 2^{28}	
		P_4, P_{13}, P_{15}, P_{26}	
6.	(a)	101100	
		$\underline{100100}$	
		111010	
	(b)	100110	
		$\underline{000111}$	
		100010	
	(c)	111010	
		$\underline{101100}$	
		010100	
	(d)	101100	
		$\underline{111100}$	
		011111	
	(e)	101001	
		101100	
		100111	
		010000	
7.	(a)	11000.0	3, 0, 0, 0, 0, 0.0
	(b)	010110011000.0	26, 12, 0, 0, 0, 0.0
	(c)	101100110011000.0	13, 6, 3, 0, 0, 0.0
	(d)	101001001001001001000.0	5, 9, 18, 4, 0, 0.0
8.	(a)	10 10 10 10 10 10 10 10 10 10 10 10 10 10 10.00	
		$\underline{\text{11 01 01 01 01 01 01 01 01 01 01 01 01 01 01.00}}$	
		00 00 00 00 00 00 00 00 00 00 00 00 00 00 00.10	
9.	(a)	001000 010000 110000 011000 111000	
		4 2 3 6 7	
	(b)	100000 010000 110000 001000 000001	
		1 2 3 4 -	
	(c)	101010 110010 001011 101010 001010	
		E C U E D	
	(d)	010011 101010 001001 100010 101001	
		S E M A N	

10. (a) 11001

(b) 10011

(c) 11011

(d) 10110

(e) 01011

(f) 11111

(g) 11110

(h) 01111

(i) 01010

(k) 11101

2

LECTURE 2.

2.1 MACHINE ORGANISATION.

Figure 2.1 shows a schematic diagram of DEUCE. The main functional parts are a High Speed Store, Backing Store, Arithmetic Unit, Input and Output units and a CONTROL Unit. The last mentioned exercises control over all other parts of the machine in accordance with the instructions which enter CONTROL from the storage unit. In the diagram, solid lines indicate paths of information transfer and dotted lines denote control pathways.

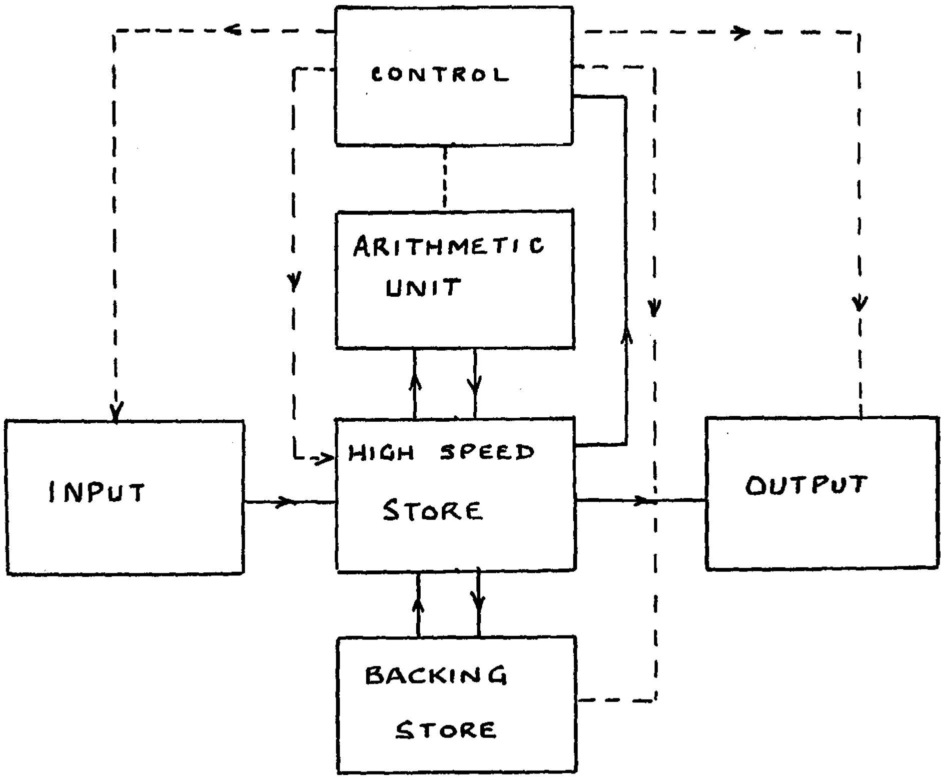


FIG 2.1 BASIC MACHINE ORGANISATION

2.2 PULSE REPRESENTATION OF NUMBERS.

The DEUCE is a serial machine, in which numbers are represented by trains of uniformly spaced pulses. Each pulse represents a one in the binary code and gaps between pulses represent zeros. These pulses pass any point in the machine at the rate of one million pulses per second. The time between pulses is thus 1 microsecond (1 /usec.).

2.3 DEUCE WORDS AND MINOR CYCLES.

As has been mentioned, there are 32 digit positions in each DEUCE register. The information contained in any register or storage position may be a binary number, an instruction, a group of characters representing alphabetic information or merely a pattern used for selection purposes. For this reason it is convenient to refer to the groups of 32 pulses by a generic term applicable to all these and it is customary to refer to a group of 32 pulses as a WORD. Since pulses are spaced at 1 /usec. intervals, one WORD passes any point in the machine in 32 /usec., one WORD TIME or MINOR CYCLE. 32 words pass any point in 1024 /u seconds, known as a MAJOR CYCLE.

2.4 DEUCE STORAGE STRUCTURE.

There are two types of storage unit in DEUCE, Mercury Delay Lines or Tanks and a Magnetic Drum. Each storage unit has within it a number of storage positions, 402 in the

Tanks and 8192 on the DRUM and each storage position can contain one word. All the current working data and instructions must be contained in the High Speed Store (H.S.S.) and the Drum serves as a backing store for data and instructions not in current use. A description of the Drum will be given in a later Lecture and we shall concentrate on the structure of the H.S.S.

As previously noted, there are 402 storage positions in the H.S.S. and it would be reasonable to give these names (or addresses) from No. 1 to No. 402. We do not do this, however, for the design of DEUCE is such that these stores are not uniform in their properties and the H.S.S. consists of four types of store. A diagram of the DEUCE H.S.S. is given in Figure 2.4 and will be referred to as the STORE DIAGRAM.

	DL1	DL2	DL3	DL4	DL5	DL6	DL7	DL8	DL9	DL10	DL11	DL12
TS 13	1_0	2_0	3_0	4_0	5_0	6_0	7_0	8_0	9_0	10_0	11_0	12_0
	1_1											
TS 14	1_2											
	1_3											
TS 15	1_4											
	1_5											
TS 16	1_6											
	1_7											
DS 19_2	1_8											
DS 19_3	1_9											
	1_{10}											
	1_{11}											
DS 20_2	1_{12}											
DS 20_3	1_{13}											
	1_{14}											
	1_{15}											
DS 21_2	1_{16}											
DS 21_3	1_{17}											
	1_{18}											
	1_{19}											
QS 17_0	1_{20}											
QS 17_1	1_{21}											
QS 17_2	1_{22}											
QS 17_3	1_{23}											
	1_{24}											
	1_{25}											
	1_{26}											
	1_{27}											
QS 18_0	1_{28}											
QS 18_1	1_{29}											
QS 18_2	1_{30}											
QS 18_3	1_{31}											

FIG 2.4 DEUCE HIGH SPEED STORE

A careful inspection of the store diagram will reveal the following important features:

(a) The store is discontinuous. There are four main types of storage block. One type (e.g. TS13) contains only one storage position. Other types contain two (e.g. DS19),four (QS17), or 32 storage positions.

(b) For all storage positions in blocks containing more than one word, the address given to the storage position consists of two numbers, the number of the block and a position within the block. For example all storage positions in Block 8 are numbered from 8_0 to 8_{31}. There would appear to be an oddity about the suffix given to a word in the two word stores. Instead of 19_0 and 19_1 we have 19_2 and 19_3 but there is a good reason for this.

2.4.1 The Temporary Stores.

There are five single word mercury delay lines. Four of these are data storage positions and the fifth is a special store into which all instructions are transferred for control purposes.

The four storage positions are known as

TS13
TS14
TS15 and
TS16

The single word contained in each of these stores circulates out of and back into the storage position once every minor cycle.

2.4.2 The Double Stores.

There are three of these, DS19, DS20, and DS21. Each can contain two words one in each of two storage positions. These storage positions are denoted as 19_2 and 19_3 for DS19, 20_2 and 20_3 for DS20 and 21_2, 21_3 for DS21.

Since there are two words in a DS tank it takes two word times for both words to circulate in the tank. In one minor cycle (for DS19) the word in 19_2 emerges and reenters the storage unit and in the next minor cycle the word in 19_3 emerges and re-enters.

In one major cycle therefore the word in 19_2 emerges sixteen times (every other minor ycle) and so does the word in 19_3 but in those minor cycles when 19_2 is not available.

2.4.3 The Quadruple Stores.

Two storage units QS17 and QS18 contain four storage positions each, referred to as 7_0, 17_1, 17_2, 17_3 and 18_0, 18_1, 18_2, 18_3. As the storage unit has four storage positions he total content of the unit (all four words) circulate out of and back into the unit once very four word times. If 17_0 emerges and re-enters in the first minor cycle it will be ollowed by 17_1 then 17_2 and finally 17_3 when it becomes the turn for 17_0 again. In each ajor cycle, each of the words in a QS emerge eight times.

2.4.4 The Long Delay Lines.

Th main storage blocks of DEUCE consist of twelve long delay lines each containing 2 storage positions. The storage units are numbered as

D.L. 1
D.L. 2
⋮
D.L. 11
D.L. 12

and the storage positions in each D.L. are identified as 8_0, 8_1, 8_2, 8_3 ... 8_{30}, 8_{31} taking D.L. 8 as an example.

As the long mercury delay lines contain 32 storage positions it will be clear that each word emerges from a long delay line once in every 32 minor cycles. In fact this explains the suffixes attached to the storage positions. The word stored in position 8_0 emerges from D.L. 8 in minor cycle 0. Similarly the word in 8_{18} emerges from the line in m.c.18 and only in this minor cycle can a copy of it be sent to another part of the computer or another word sent to 8_{18} to replace it.

In general the word stored in Delay Line A in minor cycle m is said to occupy storage position A_m. Only in minor cycle m can it be replaced or a copy moved to another corresponding storage position B_m or to one of the temporary stores.

2.4.5 Sources and Destinations.

Most of the work which goes on in a digital computer involves moving information out of storage positions into other storage positions, operating on the information in some specified way and moving the information back again into the original (or possibly another) storage position.

This means that storage positions must be capable of emitting copies of the information stored and capable of receiving information which is transmitted to them from some other storage position. Storage positions, in fact have an outlet and an inlet. These are known as SOURCES and DESTINATIONS. A source is not provided for each storage position, only for each block. All the words in D.L. 8 for example, must come out from the source associated with D.L. 8, written as SOURCE 8 or S8.

In like manner any one of the words of D.L. 5 may be replaced by sending a word to the DESTINATION of D.L. 5 (D5). If the new information is sent to D5 in minor cycle 15 then the new word will replace the information previously stored in 5_{15}.

On the Store Diagram there are 21 blocks of storage. With each of these is associated a SOURCE and DESTINATION having the same number as the block with which it is associated. For example S13, S14, S20, S6 are the Sources of TS13, TS14, DS20 and D.L. 6 while D15, D21, D17, D9 are the DESTINATIONS of TS15, DS21, QS17 and D.L. 9 respectively.

2.4.6 Rules of Word Transfer.

One basic operation in computers is the transfer of words from one storage position to another. In most machines this can only be done by moving a word from one storage position to a special register and from the register to another storage location. Sometimes this has to be done on DEUCE, sometimes not. It all depends on which storage positions are involved in the transfer. This may be deduced from the following rules:

(1) Words in TS13, 14, 15, 16 may be sent directly to any other storage position and words in any storage position may be sent directly to TS13, 14, 15 or 16. We say that S16 and D16 are available in every minor cycle.

(2) Words in 19_2, 20_2, 21_2 may only be sent directly to other storage positions with even suffixes and vice versa. Words in 19_3, 20_3, 21_3 may be transferred directly to storage locations with odd suffices. We say that 19_2, 20_2, 21_2 are available only in EVEN minor cycles and 19_3, 20_3, 21_3 are available only in ODD minor cycles.

(3) The four words in QS17 and QS18 are each available in eight minor cycles out of 32.

17_0 is available in m.c.	0	4	8	12	16	20	24	28
17_1 is available in m.c.	1	S	9	13	17	21	25	29
17_2 is available in m.c.	2	6	10	14	18	22	26	30
17_3 is available in m.c.	3	7	11	15	19	23	27	31

A word in 17_0 can only replace a word in a corresponding minor cycle, e.g. 17_0 can replace 5_{12} or 9_{20} while 17_1 can replace 8_5 or 12_{21} and vice versa.

(4) Each word in a D.L. is unique with respect to its ability to transfer directly to another D.L. 8_0 can only replace, say, 1_0, 2_0, 3_0, 4_0 etc. while 5_{25} can only replace another m.c. 25 word.

It will be noted from these rules that access is highest in the Temporary Stores and lowest in the D.L.'s. This is a common feature of cyclic storage. Low capacity stores have high access and high capacity stores, low access.

Examples.

The transfer from any SOURCE S to any Destination D is the basic DEUCE instruction . All DEUCE instructions are written

S-D

Sometimes it is necessary to specify a particular minor cycle in which case we write

S-D m.c. m

There is nothing more to DEUCE programming than arranging the correct sequence of S-D instructions with the appropriate S and D in each to meet the requirements of any problem.

Example 1.

To transfer a copy of the word in TS13 to each of TS14, DS19_2, 17_0, 5_6 and 12_{11} would require five instructions

13 - 14
13 - 19_2
13 - 17_0
13 - 5_6
13 - 12_{11}

Example 2.

To transfer a copy of the word in 5_8 to 17_0 and to 12_{11} would require three instructions

5_8 - 17_0	can be made direct (Rule 3)
5_8 - 13	because 5_8 can not transfer directly
13 - 12_{11}	to 12_{11} nor can 17_0.

Another way of doing this (also assuming that TS13 is available as a shunting or buffer storage position) is

5_8 - 13
13 - 17_0
13 - 12_{11}

Example 3.

To transfer the word from 17_3 to 21_2 we may proceed as follows, assuming TS14 is available as a buffer,

17_3 - 14	in any m.c. of 3, 7, 11 ... 31
14 - 21_2	in any even m.c.

Example 4.

If both storage positions are even or both are odd an even or odd D.S. position may be used as the buffer, EVEN buffers for EVEN minor cycles, ODD buffers for ODD. To transfer 4_{10} to 8_{16} we may proceed as follows

4_{10} - 20_2 m.c. 10 of course
20_2 - 8_{16} m.c. 16 only.

using 20_2 as an intermediate position.

Example 5.

Transfers 17_0 - 12_9
18_1 - 19_2
8_6 - 10_{16}

and a whole lot more are impossible. Why?

2.4.7 Functional Sources and DESTINATIONS.

Since every DEUCE instruction consists of a pair of numbers S and D written as

S-D,

the order code of DEUCE is contained in a description of all the 32 SOURCES and 32 DESTINATIONS.

At this stage the student already knows 21 of each of these. Let us look at some of the others.

SOURCES of Special Numbers.

S27 is a source emitting a 32-digit word in which P1 is one and all other digits are zero.

S28 is a source which emits a 32 digit word in which P_{17} is one and all other digits are zero.

S29 is a source which emits a 32 digit word in which P_{32} is one and all other digits are zero.

S30 is a source which emits an all zero 32 digit word.

S31 is a source which emits a word in which all 32 digits are ones, i.e. S31 emits -1.

Sources 27 to 31 are available in every minor cycle of machine operation.

The sources of special numbers are used for counting, marking, and formation of strobe digits. Source 30 (zero) is used to clear storage positions at the start of loops and for clearing the magnetic drum.

Accumulator Destinations.

Addition and subtraction take place in three of the 402 storage positions. These are TS13, $DS21_2$, $DS21_3$. TS13 is known as the short accumulator and DS21 is the long accumulator.

Any transfer to D13 causes the transferred word to replace the contents of TS13 and in this TS13 behaves just like any other storage position. There are however, two other Destinations associated with TS13. These are D25 and D26.

D25 A word sent to D25 is added to the contents of TS13.

D26 A word sent to D26 is subtracted from the contents of TS13.

D25 and D26 are available in all minor cycles of machine operation.

Example 6.

To form the number $5P_1$ in TS14. This may be done in several ways. With the facilities so far described however, we may proceed as follows

27-13	Send P_1 to TS13
13-25	Add TS13 to itself i.e. Form $2P_1$ in TS13.
13-25	Repeat giving $4P_1$
27-25	Add another P_1
13-14	Transfer to TS14

Example 7.

To form the negative number corresponding to a number in TS14.

Method.

Subtract the number in TS14 from zero

30-13	Clear TS13 (zero)
14-26	Subtract contents of TS14. Result in TS13.

Example 8.

To form a word which has $P_1 = 1$, $P_{17} = 1$, $P_{32} = 1$.

27-13	Send P_1 to TS13
28-25	Add P_{17} to the P_1 in TS13
29-25	Now add P_{32}.

Result in TS13.

Example 9.

To form a word which has $P_1 = 1$ and P_{17} to P_{32} all ones.

Method.

Subtract P_{17} from P_1

27-13	Send P_1 to TS13
28-26	Subtract P_{17}

Result in TS13.

2.4.8 Long and Double Transfers.

It has already been stated that all DEUCE instructions are of the form

S-D in m.c. m

So far we have not given any indication that these transfers can be for other than one minor cycle only. One very useful facility of the DEUCE is the ability to make transfers for any number of minor cycles up to 32. This is known as a long transfer. We indicate on a diagram that a transfer is for n minor cycles as follows

S-D (n m.c.)

It should be appreciated that this facility comes from providing each block of storage with a common SOURCE instead of providing each storage position with its own SOURCE. By keeping the SOURCE of D.L. 8, S8, open only for m.c. 5 then 8_5 comes out. By keeping S8 open for 20 m.c. starting at m.c. 5 all the words from 8_5 to 8_{24} come out in succession. If we open the SOURCE of TS13 for, say 10 m.c., then 10 copies of the word in TS13 come out of S13 one after the other.

Since 2 minor cycles is less than 32 a transfer may be made for 2 m.c. using the normal long transfer facility. However, transfers for 2 m.c.-are so useful that a special 2 m.c. transfer facility is provided which gives greater flexibility in organising the storage of instructions in the machine. We indicate that a transfer is for 2 m.c. by writing

S-D (d)

where the (d) stands for double.

The long and double transfers increase the power of DEUCE enormously and save instructions.

Example 10.

To form $5P_1$ in TS14, as in Example 6.

Method (a)

Clear TS13
Add P_1 five times.

30-13	Clear TS13
27-25(5mc)	Send P_1 to add to TS13 in five successive m.c.
13-14	Transfer to TS14.

Method (b)

Add four to one in TS13.

27-13	Send P_1 to TS13
27-25(4mc)	Add $4P_1$ to P_1
13-14	Transfer to TS14

Example 11.

To form a zero sum check in m.c. 31 of D.L. 8.

NOTE When m.c. 0-30 of a DELAY LINE contain information, m.c. 31 is frequently used to store a number which is the negative of the sum of all the other words in the DELAY LINE. This means that the sum of all words 0-31 is (or should be) zero and the D.L. is said to be "zero-summed".

30 - 8_{31}	clear 8_{31}
30 - 13	clear TS13
8_0- 26	(32 m.c. any 32 m.c.) This subtracts the sum of the words in D.L. 8 from zero.
13 - 8_{31}	Write the sum check in 8_{31}

CLASS EXERCISES.

SINGLE, DOUBLE and LONG TRANSFERS MAY BE USED with SOURCES 1-21, 27-31 and DESTINATIONS 1-21, 25 and 26.

1. Write instructions to perform the following operations.

(a) Put P_2 in 8_{16}.
(b) Put P_2 to P_{31} in 21_3.
(c) Put P_4 and P_{19} in 10_{15}.
(d) Put digits in all positions of D.L.9 except P_{32} in 9_{31}.
(e) Clear D.L.11.
(f) Put -10 in TS16.
(g) Put P_1 in all minor cycles of QS17.
(h) Put P_{1-16}, P_{18-31} in 19_2.
(j) Put P_1 in 21_2 P_{17} in 21_3.
(k) Clear 21_2.

2. A counter is stored in 5_{17} at P_1 position. Write the instructions to reduce the counter by one and replace it in 5_{17}.

3. Two counters are held in 19_2 and 19_3 at P_{17} position. Write instructions to Add one to both counters writing the updated count in 19_2 and 19_3.

4. Write opposite each instructions the content of the relevant destination:

30 - 14
29 - 13
29 - 25
27 - 13
28 - 25
29 - 26
27 - 26
30 - 25
31 - 25 (3 m.c.)
28 - 25
13 - 26

5. Write instructions to generate the following

(a) P_{10} in TS16.
(b) $-P_{10}$ in TS15.
(c) P_{20} in TS14.
(d) P_{1-10}, P_{20} in TS13.

6. Write the instructions to add the contents of 8_5 and 7_{13} placing the results in 5_{31}.

7. Write the instructions to add the contents of all minor cycles of D.L.11 placing the result in TS14.

8. Write instructions to add 17_0 and 17_3, subtract 18_1 and 18_2 and place the result in 21_3.

EXERCISE SOLUTIONS.

1. (a) 27 - 13

13 - 25

13 - 8_{16}

(b) 29 - 13

27 - 26 (2 m.c.)

13 - 21_{3}

(c) 27 - 13

13 - 25 (3 m.c.)

28 - 25 (4 m.c.)

13 - 10_{15}

(d) 31 - 9 (32 m.c.)

31 - 13

29 - 26

13 - 9_{31}

(e) 30 - 11 (32 m.c.)

(f) 30 - 13

27 - 26 (10 m.c.)

13 - 16

(g) 27 - 17 (4 m.c.)

(h) 29 - 13

27 - 26

28 - 26

13 - 19_{2}

(j) 27 - 21_{2}

28 - 21_{3}

(k) 30 - 21_{2}

2. 5_{17} - 13

27 - 26

13 - 5_{17}

3. 19_{2} - 13

28 - 25

13 - 19_{2}

19_{3} - 13

28 - 25

13 - 19_{3}

4.	30 - 14	TS14 = ZERO.
	29 - 13	TS13 = P_{32}.
	29 - 25	TS13 = ZERO.
	27 - 13	TS13 = P_{1}.
	28 - 25	TS13 = $P_{1} + P_{17}$.
	29 - 26	TS13 = $P_{1} + P_{17} + P_{32}$
	27 - 26	TS13 = $P_{17} + P_{32}$
	30 - 25	TS13 = $P_{17} + P_{32}$
	31 - 25 (3 m.o.)	TS13 = $P_{1} + P_{3-16} + P_{32}$
	28 - 25	TS13 = $P_{1} + P_{3-17} + P_{32}$
	13 - 26	TS13 = ZERO.

5. (a) 27 - 13
13 - 25 (9 m.c.)
13 - 16

(b) 31 - 13
13 - 25 (9 m.c.)
13 - 15

(c) 28 - 13
13 - 25 (3 m.c.)
13 - 14

(d) 27 - 13
13 - 25 (10 m.c.)
27 - 26
28 - 25 (8 m.c.)

6. 8_5 - 13
7_{13}- 25
13 - 5_{31}

7. 30 - 13
11 - 25 (32 m.c.)
13 - 14

8. 30 - 13
17_3 - 25 (2 m.c.)
18_1 - 26 (2 m.c.)
13 - 21_3

3

LECTURE 3.

3.1

This lecture will be concerned with describing the following functional sources and destinations

S0,	S23,	S24,	S25,	S26,	S22.
D29,	D27,	D28,	D24,	D22,	D23.

and the operation of the two trigger control circuits T.C.A. and T.C.B.

3.2 SOURCE 0. S.0.

All computers have to be provided with a means for taking in data and instructions. On DEUCE this is done using Source 0.

When the card reader is not running SOURCE 0 supplies whatever word is set on 32 handswitches on the Control Panel. These 32 switches and the 32 lights which monitor them are known as the I.D. (Input Dynamiciser). When the Card Reader is running SOURCE 0 supplies words corresponding to the patterns of holes punched on each row of the card.

SOURCE 0 is available in every minor cycle of machine operation.

Example 1.

To add a number set on the I.D. to another number in TS14, placing the result in TS15 we need three instructions.

	14 - 13	Send number in TS14 to TS13
	0 - 25	Add in the number from ID
	13 - 15	Place the result in TS15
or	0 - 13	Send number from ID to TS13
	14 - 25	Add in the number from TS14
	13 - 15	Send the result to TS15

3.3 DESTINATION 29 D29.

The output destination on DEUCE is D29. Any word sent to D29 from another part of the machine appears as a binary pattern of lights on a row of 32 lights on the Control Panel. The row of lights is known as the O.S. (Output Staticisor).

If the card punch is running then any word sent to D29 is punched on a row of a card.

Sending a word to D29 does not remove the word which was previously present. The two words merge in a manner known as an 'OR' combination.

Before any word is sent to D29 it is usual to clear the O.S. previously. This can be done by a special instruction and is done automatically by the punch after each row is dealt with.

Example 2.

To send the Contents of TS16 to the O.S. we write

16 - 29

Example 3.

To put a P_1, P_{17}, and P_{32} on the O.S. we write

27 - 29	Send P_1 to O.S.
28 - 29	Send P_{17} to O.S.
29 - 29	Send P_{32} to O.S.

Example 4.

If we send two words successively to the O.S. and these have common digits we shall obtain a pattern where digits exist in either or both words.

27 - 29
31 - 29

has no more effect than the single instruction

31 - 29

because 31 is all ones and this includes a P_1.

D29 is available in every minor cycle of machine operation.

3.4 DESTINATIONS 27 and 28.

When DEUCE is obeying the instructions of a programme, each instruction usually specifies only one successor. All computers are provided with ability to choose one of two possible next instructions whenever this is necessary.

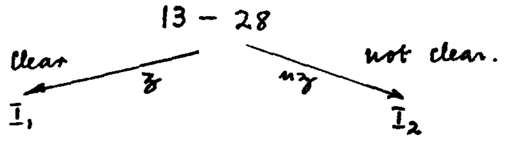
At all points in a programme where the sequence of events may take one of two paths we insert a special instruction known as a DISCRIMINATION instruction. The effect of this is to test whether some criterion is satisfied and if it is,lead to the first of two possible successive instructions. If the test is not satisfied the machine obeys the second of the pair of next instructions.

D27 tests if any word sent to it is positive or negative i.e. it looks to see if a P_{32} digit is present (negative) or absent (positive)

D28 tests if any word sent to it is zero or non zero i.e. it looks to see if there is a complete absence of digits in the incoming word or not.

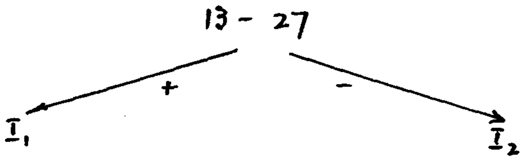
Example 5.

To test if the short accumulator is clear write



Example 6.

To test if the word in TS13 is positive or negative write

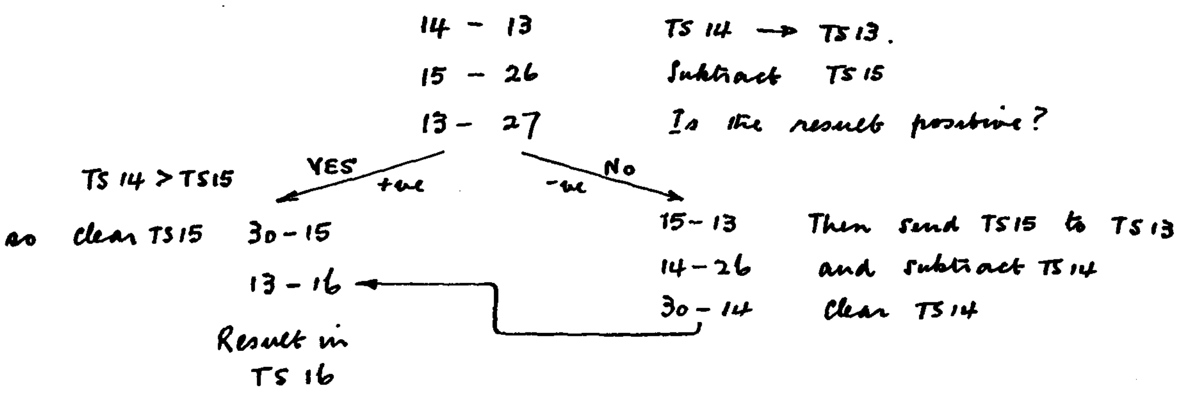


Example 7.

To subtract whichever is the smaller of the two words in TS14 and TS15 from the other placing the result in TS16. If TS14 is greater than TS15 clear TS15 otherwise clear TS14.

We do not know which word is the greater but we must start somewhere so we subtract TS15 from TS14. If the result is positive then TS14 is greater than TS15. If the result is negative we repeat the operation in reverse order.

The Instructions are

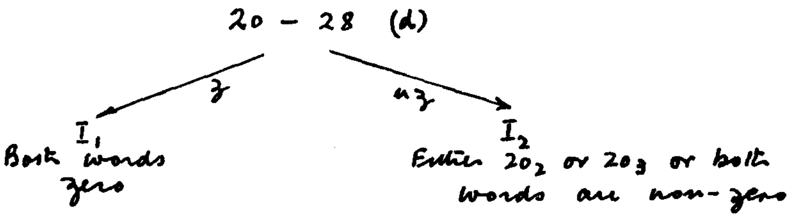


Notice how we can use the same 13 - 16 instruction (which places the result in TS16) irrespective of the route followed.

Remembering the ability to perform transfers for up to 32 m.c. we can test if all the numbers in any DL are all zero, all positive or any particular set of numbers is an all zero or all positive set using only one (appropriate) discrimination.

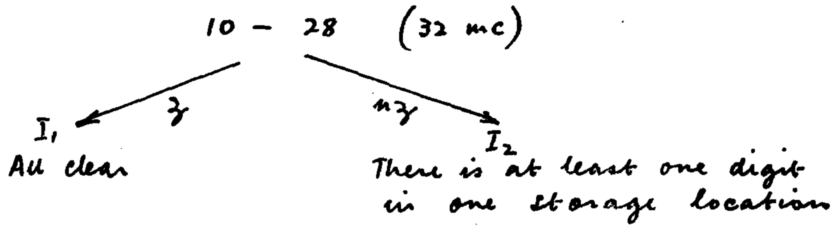
Example 8.

To test if both numbers in DS20 are zero write



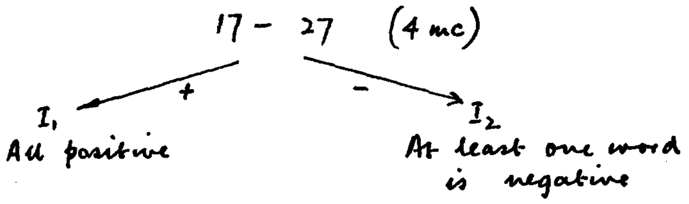
Example 9.

To test if all the storage positions of DL10 are empty write



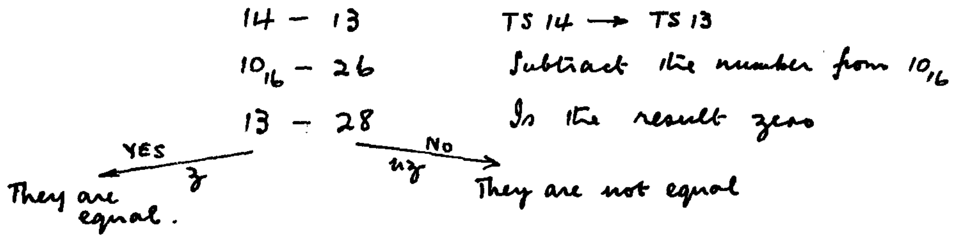
Example 10.

To test if there is at least one negative number in QS17 write



Example 11.

To test if the numbers in 10_{16} and TS14 are equal write



D27 and D28 are available in every minor cycle of machine operation.

3.5 SOURCE 24 - SHIFT UP.

TS14 is the single length shifting register. Numbers may be shifted up (multiplied by 2 for one shift or 2^n for n shift) using SOURCE 24.

If TS14 contains any pattern of digits, SOURCE 24 emits a word which is the same as the pattern in TS14 moved up one place, i.e. multiplied by 2. If TS14 has a P_{32} present this P_{32} will not be evident from SOURCE 24 because it will have been pushed off the top of the word length.

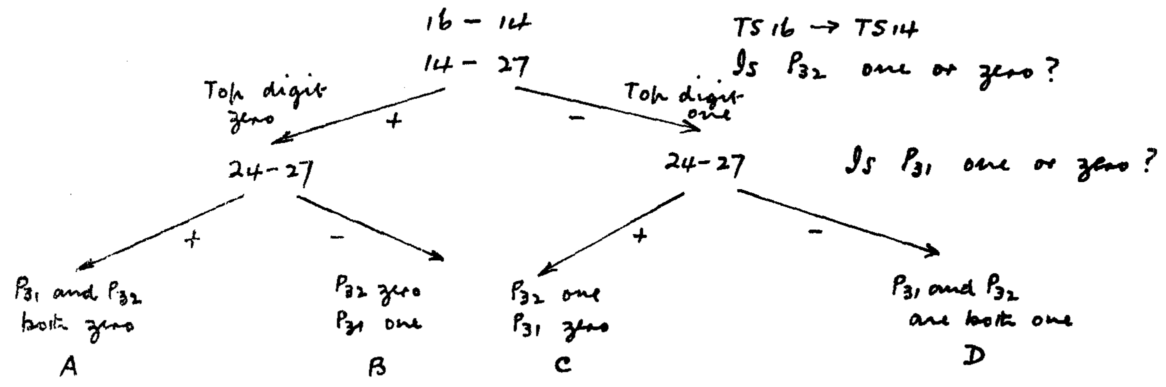
Example 12.

To put a P_{18} in TS13 using SOURCE 24. write

28 - 14 $P_{17} \longrightarrow$ TS14
24 - 13 P_{17} x 2 (i.e. P_{18})→ TS13

Example 13.

Are the top two digits of TS16 the same?



Notice that we have a four exit decision sequence using three discriminations. If Paths A and D are followed the top digits of TS16 are the same, if paths B and C are followed the top digits of TS16 are different.

Example 14.

To move the word in TS15 eight places to the right (i.e. multiply by 2^8)

15 - 14 TS15 → TS14
24 - 14 (8 m.c.) Shift up 8 places
14 - 15 Place the shifted result back in TS15

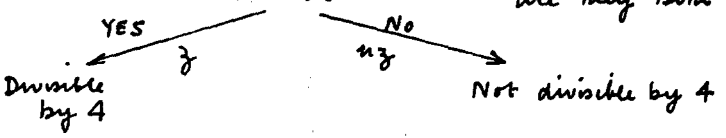
Notice that we use a long instruction to replace the contents of TS14 with its former contents shifted up one place for each minor cycle of transfer. In fact 24 - 14 (8 m.c.) has the same effect as eight separate single minor cycle transfers of 24 - 14 and why use eight instructions when one will suffice?

Example 15.

To test if the number in TS15 is divisible by four. If it is, the bottom two digits of TS15 will both be zero.

Proceed as follows

15 - 14		Send TS15 to TS 14
24 - 14	(30 mc)	Shift up 30 places. Now only the P_1 and P_2 digits of the word are left in TS 14
14 - 28		Are they both zero?



3.6 SOURCE 23 - SHIFT DOWN

Just as SOURCE 24 supplies the contents of TS14 moved up one place, so SOURCE 23 supplies the contents of TS14 shifted down (÷ 2) one place.

(a) If TS14 has a P_1 present this will be lost off the bottom from SOURCE 23.

(b) If TS14 contains a P_{32} SOURCE 23 will provide a copy of this, i.e. if TS14 is negative then SOURCE 23 will provide a negative number also.

By sending SOURCE 23 to TS14 for several minor cycles we can move words down in TS14 several places in one instruction.

Example 16.

If TS14 contains P_2 to P_{31}, SOURCE 23 emits P_1 to P_{30}.

Example 17.

If TS14 contains P_{32} SOURCE 23 emits P_{31} and P_{32} (i.e. the original P_{32} moved down to P_{31} and a copy of the original P_{32} supplied for good measure)

Example 18.

TS15 contains the number of years (since 1900) in a date. To calculate the number of leap years from 1900 to the given date. The number of leap years is the result given by dividing the number of years by 4.

15 - 14	Send "years" to TS14
23 - 14	(d) ÷ 4
14 - 13	"Leap years" to TS13.

NOTE If there is a P_1 or a P_2 in TS14 before the shift, these will be lost, but in this instance it does not matter. We are interested in the quotient not the remainder.

3.7 SOURCE 25 LOGICAL "AND" TS14 & TS15

SOURCE 25 emits a word which is (at first sight) a curious mixture of the two words in TS14 and TS15. The word emitted has a one in every P position in which both TS14 and TS15 have corresponding ones and zero in every other digit position.

Two binary digits can give rise to four combinations as follows

Word in TS14	$P_1 = 0$;	$P_1 = 1$;	$P_1 = 0$;	$P_1 = 1$
Word in TS15	$P_1 = 0$;	$P_1 = 0$;	$P_1 = 1$;	$P_1 = 1$
SOURCE 25	$P_1 = 0$;	$P_1 = 0$;	$P_1 = 0$;	$P_1 = 1$

This may be written much more explicitly as

TS14	0101
TS15	0011
S25	0001

From the definition of the word COLLATE (to compare in detail) it will be seen that TS14 and TS15 are being Collated digit by digit to signify through SOURCE 25 where both words agree to the extent of having 'ones' in corresponding digit positions.

Example 19.

To select out the bottom ten digits from the word in TS16, proceed as follows.

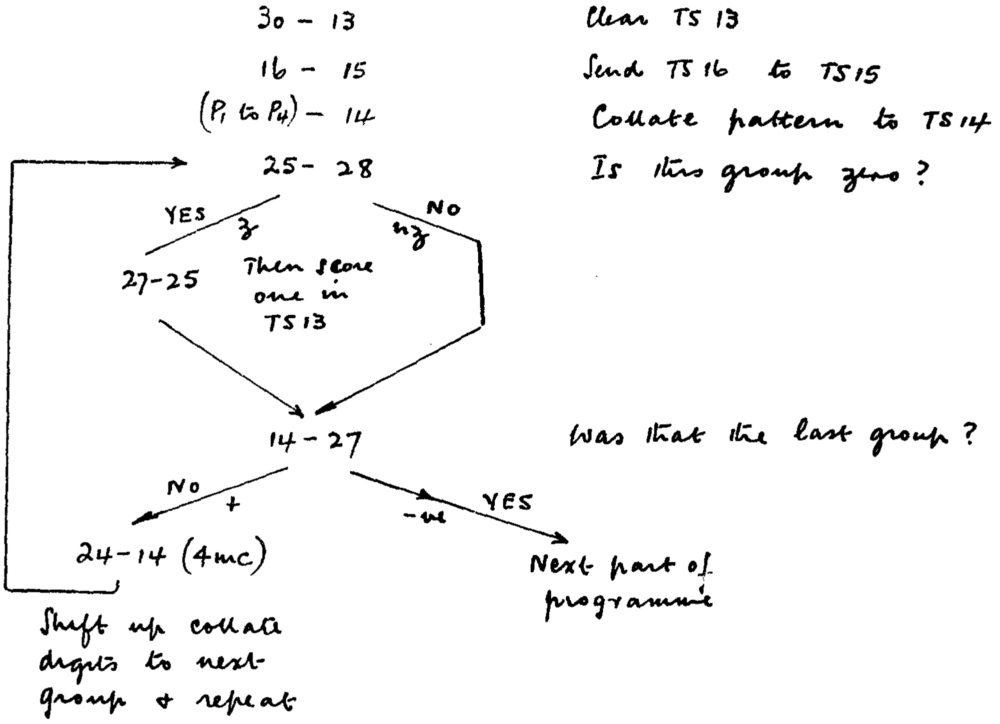
16 - 14	TS16 sent to TS14
(P_1-P_{10}) - 15	10 consecutive COLLATE digits go to TS15
25 - 16	Select bottom 10 digits.

Example 20.

TS16 contains a number in 4 - bit binary coded decimal. To search each character in turn to find zeros and count them in TS13. Suppose the number is 9307, given in TS16 by

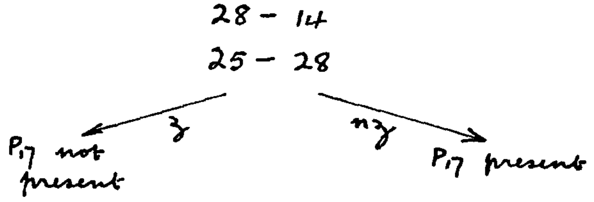
1110	0000	1100	1001	0000	0000	0000	0000
7	0	3	9				
$P_1 - P_4$	$P_5 - P_8$	$P_9 - P_{12}$	etc.				

Our task is to select each group of four digits in the word, adding one in TS13 each time a zero character is found.



Example 21.

To test if the word in TS15 has a P_{17} present write



3.8 SOURCE 26 NOT EQUIVALENT TS14 $\not\equiv$ TS15

This is another SOURCE which emits a word composed of a mixture of the two words in TS14 and TS15. SOURCE 26 provides a word which has ones in those digit positions where the digits of TS14 and TS15 are different, i.e. where one is zero and the other is one and vice versa. The zeros in SOURCE 26 occur in those positions where the digits of TS14 and TS15 are both zero or both one.

The results obtained from SOURCE 26 may be seen from the two standard patterns in TS14 and TS15

TS14	0101
TS15	<u>0011</u>
S 26	0110

This function is known as the <u>NOT EQUIVALENT</u> function though readers with knowledge of logical algebra will recognise the <u>EXCLUSIVE - 'OR'</u>.

<u>Example 22.</u>

To obtain the 'not' word from the word in TS16 (Remember from Lecture 1. that the 'not' or inverse of a binary number x is ~ x, a word having zeros in place of ones in x and ones in place of zeros).

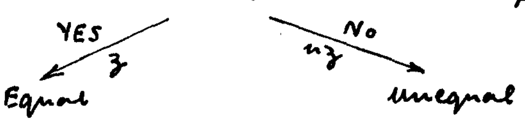
The instructions are

16 - 14	x sent to TS14
31 - 15	"all ones" to TS15
26 - 13	~ x sent to TS13

<u>Example 23.</u>

To test if the numbers in 8_6 and 7_5 are equal

8_6 - 14	8_6 sent to TS 14
7_5 - 15	7_5 sent to TS 15
26 - 28	Are the numbers identical ?



<u>Example 24.</u>

To obtain in TS16 the INCLUSIVE OR combination of the words in TS15 and TS14. The inclusive OR word is a word which has ones where <u>either</u> or <u>both</u> words in TS15 and TS14 have ones.

Since S25 gives ones where both have ones and S26 has one where either has ones and the other zeros we add these together

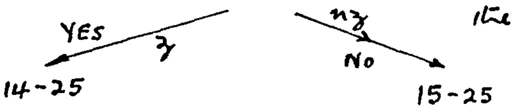
x - 14	x sent to 14
y - 15	y sent to 15
25 - 13	Common 'ones' to 13
26 - 25	ones in either but not the other added
13 - 16	Result to TS16.

<u>Example 25.</u>

To examine the $P_1 - P_4$ digits of the word in TS16 and if these are equal to 7 to add them to the contents of TS13 otherwise to add the contents of TS15 to TS13.

Here we need both S25 and S26, the former to select and the latter to compare.

16 - 15	Send TS 16 to TS 15
(P_1-4) - 14	Collate digits to TS 14
25 - 14	Select first four digits
(P_1-3) - 15	7 P_1 sent to TS 15
26 - 28	Are the selected digits the same as 7 P_1 ?



SOURCES 23, 24, 25 and 26 are available in every minor cycle of machine operation.

3.9 D24 DESTINATION TRIGGERS.

It will have become apparent that in all SOURCES and destinations so far described each SOURCE emits a word of 32 digits and each destination receives a word of 32 digits. In fact information in the form of a DEUCE word flows from the SOURCE to the DESTINATION. There are a number of Control instructions which must be performed on Digital Computers which do not necessitate the transfer of word information. Executive commands such as "Start the Reader", "CLEAR THE O.S." etc. are typical of such instructions. Because the DEUCE is limited to 32 SOURCES and 32 DESTINATIONS and because the number of necessary instructions is greater than can be accommodated in these combinations, use is made of a special device. This is D24.

D24 is a common trigger destination. Whenever D24 is used one of 32 special control circuits is operated to initiate a control instruction. Which particular control circuit is operated depends on the SOURCE number associated with D24. We have in effect a separate order code for Control instructions associated with D24.
For example, if we write

9 - 24

this does not mean "Send one of the words in DL9 to Destination 24", it means "Operate Control circuit No. 9". Here is the list of the first 12 control instructions. Most of the others are concerned with Paper Tape or Magnetic Tape and are given later.

0 - 24	Start Multiplication
1 - 24	Start Division
* 2 - 24	Send TIL to D28
3 - 24	Put TCA on
4 - 24	Put TCB off
5 - 24	Put TCB on
6 - 24	Clear the alarm
7 - 24	Sound the alarm
8 - 24	Clear the O.S.
9 - 24	Clear either or both READER AND PUNCH
10 - 24	Start the PUNCH
11 - 24	not used
12 - 24	Start the Reader.

* TIL is a signal which indicates that the last row of a card has passed the reading (INPUT) or punching position (OUTPUT) of the READER OR PUNCH respectively. Further explanation of TIL is given later.

TCA and TCB will be explained later in this lecture and the operations of READING and PUNCHING will be explained in Lecture 8.

The alarm is a buzzer mounted on the control panel. When it is sounded (usually to call the operators attention to an error) it is put on by an instruction 7 - 24 and it may be put off by an instruction 6 - 24.
8 - 24, "Clear the Output Staticisor" cancels any previous D29 instruction and is the special instruction referred to in Section 3.3.

3.10 TRIGGER CIRCUIT A T.C.A.

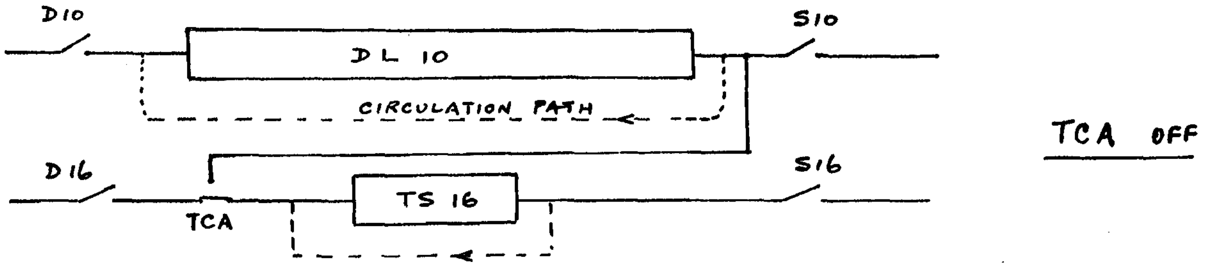
This control facility connects TS16 and DL10 in a special way. Normally TS16 can store one word and when requested to do so S16 emits whatever word is stored in TS16.

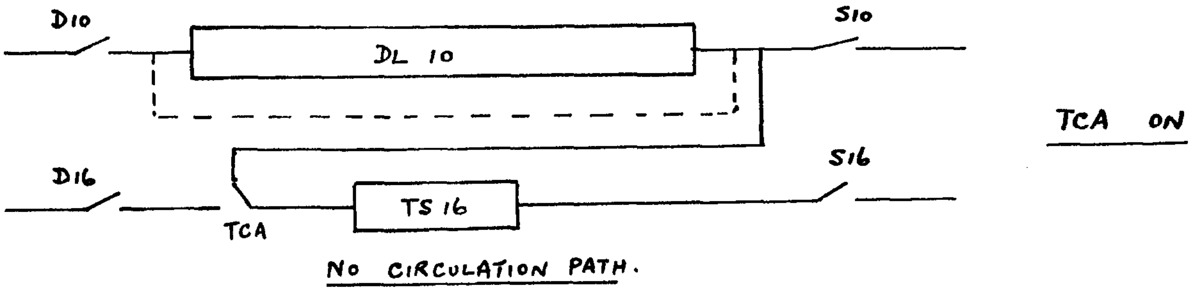
When TCA is ON, however, TS16 can not store any information because its circulation loop is broken. All the words in DL10 flow successively through TS16 and S16 acts as an echo of S 10.

In minor cycle 9 for example, the word available from DL10 is 10_9. The word available from TS16 (with TCA ON) is 10_8.

We therefore define S 16, (TCA ON) as emitting the contents of DL10 delayed one word time.

The following diagrams may be helpful in appreciating the function of TCA.





With TCA off, DL10 and TS16 are separate storage tanks. With TCA ON TS16 has been connected like a one word caboose behind the 32 word train of DL10. Its circulation path is broken and TS16 merely acts as a one word delay on all the words stored in DL10.

Example 26.

To transfer 10_{10} to 9_{11} in one instruction. It will be recollected that the Rules of Transfer forbid operations of the form

$$10_{10} - 9_{11}.$$

As these minor cycles are only one apart and only because the earlier one is to be transferred to the later one we can do this in one transfer if TCA is ON by

16 - 9 (m.c. 11) TCA ON.

The most useful feature of TCA is to shift words in DL10.

Example 27.

To replace the words in 10_{1-6} by the words from 10_{0-5} i.e. move m.c. 0-5 down one place in DL10, losing m.c. 6 altogether

3 - 24 Put TCA ON.
16 - 10 (m.c. 1 - 6) Puts 10_{0-5} (out of S 16) in 10_{1-6}

Example 28.

To move all the minor cycles of DL10 one place

3 - 24 Put TCA ON
16 - 10(32mc) Delay each word in DL10 and send the delayed words to DL10.

When TCA is ON TS16 does not store information. It will revert to a storage position and TCA will be put off when any transfer is made to D16.

X - 16	Puts TCA off

Where X is any SOURCE. The information from SOURCE X passes into TS16 and is stored there in the normal way.

3.11 TCB. TRIGGER CIRCUIT B.

Additions, subtractions and downward shifts may be performed in DS21 and as this is a two word storage tank it is known as the "Long Accumulator". It has been stated that words in DEUCE have 32 digits with P_1 as the least significant and P_{32} as thesign digit. 32 digits are often sufficient for most purposes but there are occasions when it is necessary to double the word length and work with words of 64 digits.

To work with double length words it is necessary to be able to add them, subtract them and shift them either up or down. This can be done in DS21.

Any double length word consists of two halves, a less significant 32 digit word and a more significant word. By convention the word in an Even storage position of a pair is the less significant half and the word in the ODD storage position is the more significant.

Furthermore, a 64 digit word only needs one sign digit and this is the P_{32} digit of the ODD minor cycle word. When words are treated as double-lengths the P_{32} digit of the less significant word is just another digit.

When TCB is OFF all words in DS21 are treated as double length numbers. In additions and subtractions, carry digits can pass from 21_2 to 21_3 and in shift operations only the sign digit of 21_3 is copied (See SOURCE 23 for comparison).

When TCB is ON, however, DS21 is treated as a two word store. There are now sign digits P_{32} in 21_2 for the word in the EVEN half and P_{32} in 21_3 for the word in the ODD half.

Numbers added together in 21_2 do not cause carry propagation to 21_3.

3.12 SOURCE 22 SHIFT DOWN IN DS21

SOURCE 22 with respect to DS21 corresponds to SOURCE 23 with respect to TS14. SOURCE 22 emits a word which is the word in DS21 shifted down one place.

NOTE For TCB OFF, the instruction

22_2-20_2

will take the word in 21_2, shifted down one place, and transfer it to $DS20_2$. Since TCB is OFF, (P_{32}) EVEN is NOT a sign digit and is NOT copied. The P_{32} position of the word from $S22_2$ is occupied by the P_1 digit of 21_3 .

The instruction

$$22_3 - 20_3 \quad \text{(TCB OFF)}$$

transfers a copy of the word in 21_3 shifted down one place to $DS20_3$. $(P_{32})_{\underline{ODD}}$ is a sign digit and will be copied in the word emitted by 22_3.

The instruction

$$22 - 20 \quad \text{(d)} \quad \text{(TCB OFF)} \quad \text{(e,o)}$$

transfers one place left shifted copies of both words in 21 to DS20. Again only the sign digit of 21_3 is copied.

With TCB ON, the instruction

$$22_2 - 20_2$$

transfers the one place shifted down copy of $DS21_2$ to $DS20_2$. Now however $(P_{32})_{\underline{EVEN}}$ is treated as the sign digit of 21_2 and a copy of this digit is made in any transfer from S22. Similarly

$$22_3 - 20_3$$

shifts down the content of 21_3 one place and transfers the result to 20_3. Note that the shifted copy of 21_3 appears in 20_3 not in 21_3. This can be done by

$$22_3 - 21_3.$$

Example 29.

To shift the double length number in DS21 four places to the left. Since 23 - 14 (4 m.c.) will shift the single length word in TS14 four places left we may be tempted to infer that

$$22 - 21 \text{ (4 m.c.)} \quad \text{(e,o)}$$

will perform a corresponding operation in DS21. This is not so however, for it takes 2 m.c. to replace both words in DS21 and we need to replace both words four times, shifting down each time.

The instruction is

$$22 - 21 \text{ (8 m.c.)} \quad \text{(e,o)}$$

with an 8 m.c. transfer to shift 4 places.

3.13 D22. ADD INTO DS21.

D23. SUBTRACT FROM DS21

For additions and subtractions in DS21, D22 and D23 have the same effect as D25 and D26 in TS13.

Words sent to D22 are added to the appropriate word in DS21. Words sent to D23 are subtracted from the appropriate word in DS21. D22 and D23 may be used in the single length (TCB ON) mode or the double length (TCB OFF) mode.

With TCB ON the words in TS14 and TS15 may be added in either $DS21_2$, $DS21_3$ or TS13 as follows

(a) In 21_2 — $14 - 21_2$
$15 - 22_2$
Result in 21_2

In 21_3 — $14 - 21_3$
$15 - 22_3$
Result in 21_3

In TS13 14 - 13
15 - 25
Result in TS13.

In a similar manner D23 permits single length subtractions to occur in either 21_2 using $D23_2$ or 21_3 using $D23_3$. With TCB OFF however, D22 and D23 deal with double length additions and subtractions.

Example 30.

To add the double length number in $DS20_{2,3}$ to the double length number in $DS19_{2,3}$. We write

```
20 - 21     (d)
19 - 22     (d)    (e,o)
```

Result in $DS21_{2,3}$.

Example 31.

To subtract the double length number in $DS19_{2,3}$ from the double lenth number in $DS20_{2,3}$ the instructions are

```
20 - 21     (d)
19 - 23     (d)    (e,o)
```

Example 32.

To shift up the double length number in DS21 four places. This may be done by successive additions of the number to itself in $DS21_{2,3}$. One long instruction for 8 m.c. is required

```
21 - 22     (8 m.c.)    (e,o)
```

FOUR IMPORTANT POINTS.

(A) FOR DOUBLE LENGTH WORKING TCB has not been shown to be put OFF and for single length working TCB has not been shown to be put ON. This is because it is the duty of every programmer to know the State of TCB at each point of his programme. If TCB is required ON and is already ON it is not necessary to put it ON again. If TCB is ON but operations are to be performed with TCB OFF then

4 - 24

must precede the double length working instructions.

(B) 0 - 24 START A MULTIPLICATION also PUTS TCB OFF AUTOMATICALLY

(C) 1 - 24 START A DIVISION also PUTS TCB ON AUTOMATICALLY.

(D) ALL TRANSFERS TO D22, D23 and from S22 for more than one minor cycle must occur in the order EVEN minor cycle first, ODD minor cycle second, and so ON.
There is no programmer using DEUCE who has not made this mistake at some time or other. The resulting errors can be very difficult to find and all D22, D23 and S22 long transfers should have (e,o) written on the flow diagram in an attempt to avoid this, probably the greatest DEUCE pitfall of all.

3.14 EXTENDED ADDITION AND SUBTRACTION.

One facility built into the DEUCE is the ability to convert any single length word sent to $D22_2$ or $D23_2$ into a correctly signed double length word.

For example, if the single length word in TS15 is

101011... ...1100

The correct double length equivalent is

101011... ...1100 0000... ...0000.

For a negative single length word in DEUCE such as

111011... ...0111

with P_{32} present as shown, the double length word is

111011... ...0111 11111111... ...111111

The sign digit must be propagated correctly up to the P_{64} ($(P_{32})_{ODD}$) position.

If therefore single length words sent to $D22_2$ or $D23_2$ are to be converted to double length automatically the computer must recognise whether the single length numbers are positive or negative and continue the transfer for a further word time with a word which is either 32 zeros for positive numbers or 32 ones for negative numbers. This is exactly what happens if a transfer is made to $D22_2$ or $D23_2$ for TCB OFF. With TCB ON however numbers are not treated as double length and no extension occurs.

The rule is as follows:-

Any transfer ending in an EVEN minor cycle to D22 or D23 with TCB OFF will be extended for a further minor cycle. The word transferred in the extra minor cycle will be all zeros if the last digit sent to D22 or D23 is zero and will be all ones if the last digit sent to D22 or D23 is one.

It should be noted that any transfer ending in an EVEN minor cycle can be just one minor cycle or the last minor cycle of a double or long transfer

CLASS EXERCISES.

1. The word on the I.D. represents

$AP_1 + BP_9 + CP_{17} + DP_{23}$ (A, B, C and D are positive integers)

Write instructions to read this into DEUCE and separate out the sections placing

$(A + D)\ P_1$ in TS13.
$B\ P_1$ in TS16.
$C\ P_1$ in 2_8.

HINT. Use SOURCE 25 to COLLATE.
SOURCE 23 to shift.

2. Write instructions to examine whether P_{32} is on the I.D. and if so stop the DEUCE to read the I.D. into TS16. If P_{32} is not present, place P_{31} in TS15.

NOTE. Place a X on the right of the instruction at which the machine must stop.

3. Write instructions to invert the ones and zeros of the word in 19_2 (i.e. form the 'not' word).

4. Write instructions to test if the digits in P_9 to P_{12} of the word in 21_3 are 1001 (i.e. represent 9 in binary coded decimal).

5. Write instructions to invert the digits P_{1-4} of the word in TS16 preserving the rest of the word intact.

6. Write instructions to test if P_{5-9} of the two words in DS19 are the same or different.

7. Form digits in one word A marking the positions where zero exist in P_5, P_9, P_{17}, P_{17} of another word B.

P_5 P_9 P_{13} P_{17}
e.g. 1000 $\underline{0}$110 1000 $\underline{0}$110 1000 ... B
0000 $\underline{1}$000 0000 $\underline{1}$000 0000 ... A

8. Write two different sets of instructions to test if two words are the same. Use TS13 in the first set link not in the second.

9. Write a small programme to light successive P digit lamps on the O.P.S.

10. Write a small programme to light successive P digit lamps on the O.P.S. at a rate determined by a digit set on the I.D.

HINT. Use the digit on the I.D. as a counter. Light a lamp when the counter is zero, next reset the counter and continue. Clear the O.P.S. when all lamps are lit.

11. Write instructions to test if 19_2 is zero. If it is write all ones in 19_2, if non zero clear 19_2. What technique does this illustrate?

12. Write instructions to test if the top two digits of TS14 are 00, 10, 01 or 11 leading out to exits A, B, C, D for each case.

13. Repeat 12 obeying the following instructions at A, B, C, D.

A, Sound the Alarm.
B, Clear the Alarm.
C, Call the Reader.
D, Call the Punch.

14. What does 30 - 29 do?

15. T.C.B, is ON. Write instructions to add the double length content of DS20 to DS19 placing the result in $5_{10,\ 11}$.

16. T.C.B. is OFF. Write the instructions to shift NP_{28} in $DS21_2$ to NP_1 in 21_3.

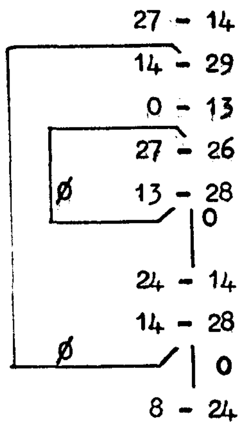
17. Subtract the contents of 18_0 from TS14 and 18_1 from TS15 using DS21. T.C.B. is OFF originally.

18. The number in 21_2 is originally held to 30 b.p. Write the instructions to obtain a copy of the number to 20 b.p. in TS13.

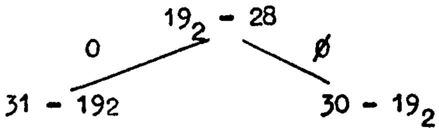
EXERCISE SOLUTIONS.

```
1.    0 - 14
        - 15      $P_{1-8}$
     25 - 13      A
     23 - 14 (8 m.c.)
     25 - 16      B
     23 - 14 (8 m.c.)
     25 - $2_8$   C
     23 - 14 (8 m.c.)
     14 - 25      A + D

2.          0 - 27
          +  /  \  -
     28 - 14     0 - 16X
     24 - 14 (14 m.c.)
     14 - 15

3.   $19_2$ - 14
     31  - 15
     26  - $19_2$

4.   $21_3$ - 14
         - 15      $P_{9-12}$
     25  - 14      Collate out $P_{9-12}$field only.
         - 15      $P_9$ + $P_{12}$
     26  - 28
    0  /   \  Ø
   YES       NO

5.   16 - 14
        - 15      $P_{1-4}$
     26 - 16

6.   $19_2$ - 14
         - 15      $P_{5-9}$
     25  - 13
     $19_2$ - 14
     25  - 26
     13  - 28
    0  /   \  Ø
   YES       NO

7.       - 14      B
         - 15      $P_5$ + $P_9$ + $P_{13}$ + $P_{17}$
     25  - 14
     26  -         A

8.       - 13      A                      - 14      A
         - 26      B                      - 15      B
       13 - 28                          26 - 28
    0  /   \  Ø                       0  /   \  Ø

9.              27 - 14
             ┌─ 14 - 29
             │  24 - 14
             │  14 - 28
            Ø└──────┘
                     |Ø C
```

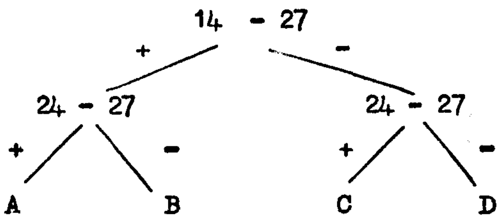
10.



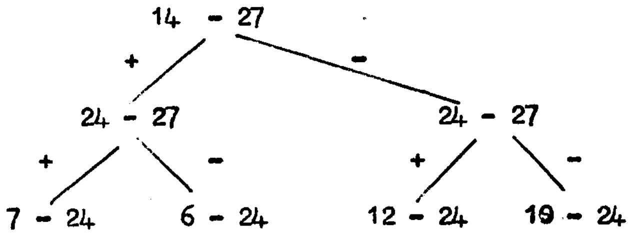
11.



Illustrates MARKER technique.



13.



14. Nothing. It superimposes zeros on the word at present appearing on the O.P.S. lights, but does <u>not</u> clear it.

15. 4 - 24
20 - 21 (d) (e, 0)
19 - 22 (d) (e, 0)
21 - 5_{10} (d)

16. 21 - 22 (8 m.c. e, o)

17. 14 - 21_2
15 - 21_3
5 - 24
18_0- 23 (d)

18. 21_2- 14
23 - 14 (9 m.c.) Shift to 21 b.p.
14 - 13 Copy into TS13.
23 - 26 Subtract $\frac{1}{2}$ of TS14 rounded down to give correctly rounded result.

4

LECTURE 4.

4.0 MULTIPLICATION.

All digital machines work in terms of integers, and "all digital machines" includes desk calculators as well as high speed electronic computers. On desk machines the integers are in the decimal scale of notation, whereas most high speed computers use the binary scale. The processes involved in multiplying two binary numbers on DEUCE are the same in principle as the processes involved in multiplying two numbers on a desk machine such as a FACIT or BRUNSVIGA. The difficulty encountered by most programmers at their first introduction to the DEUCE multiplier arises from the fact that the operands and the multiplication process are hidden from view whereas they can be seen on a desk machine.

4.1 DESK CALCULATOR MULTIPLICATION.

To multiply 501 by 25 on a desk calculator the first number 501 is set in the "set up" register, with the 1 in the least significant digit position. The handle is turned until 25 appears in the COUNTING (OR MULTIPLIER) REGISTER with the 5 in least significant digit position. The product 12525 appears in the Product register with the 5 units in the least significant place. If the input numbers are positioned with their least significant digits in the least significant digit positions of the respective registers the product register will hold the product positioned with the least significant digit in the least significant digit position of the register.

Now let us consider multiplication of non-integers on a desk machine which can handle integers only.

To multiply

75.346 by 53.42 we multiply

75346 by 5342 to give 402498332.

As the original numbers have a total of 5 decimal places the result is adjusted by inserting a decimal point five places from the least significant end to give

4024.98332

What, in fact does this process amount to?

The original numbers are changed to integers

$$75.346 \text{ becomes } 75.346 \times 10^3$$
$$53.42 \text{ becomes } 53.42 \times 10^2$$

The integers are multiplied together and the result is too large by a factor of $10^3 \times 10^2 = 10^5$ This may be generalised as follows:

When a number X with p decimal places is multiplied by a number Y with q decimal places the integer product is $\left(X \cdot 10^b\right)x\left(y \times 10^q\right) = x.y \cdot 10^{p+q}$, representing the true product xy which has (p+q) decimal places.

4.2 BINARY MULTIPLICATION.

Any non-integer x having p binary places is represented in DEUCE as the integer $x.2^p$

Example 1.

$$\tfrac{1}{3}^{rd} = \ldots 1010101010101010101010101010101010\cdot$$

$\frac{1}{3}^{rd}$ held in DEUCE to 15 binary places would appear as

$$\begin{array}{l}\qquad\qquad\quad\downarrow\\ \boxed{\texttt{01010101010101010000000000C00000}}\end{array}$$

If a binary point could be indicated it would be placed between P_{15} and P_{16} as shown by the arrow. The integer relative to P_1 is the integral part of $\frac{1}{3} \times 2^{15}$, which may be checked by moving the binary point 15 places to the left

$$\tfrac{1}{3} \text{ x } 2^{15} = 1010101010101010101.\underline{\underline{01010101010101010}}$$

Two numbers, x with p binary places and y with q binary places, are multiplied as integers $x.2^p$ and $y.2^q$.

The product $xy \,.\, 2^{p+q}$ is an integer representing xy to (p+q) binary places. The product of two 10 digit decimal numbers produces a 20 digit number, and similarly multiplication of two 32 digit binary numbers in DEUCE produces a 64 digit product (a double-length number) in $DS21_{2,3}$.

After multiplying two binary non-integers, each having 30 binary places, the binary product which has 60 binary places is double length. To obtain a single length number it is necessary to discard 32 digits from the bottom end. This is known as **truncation**. If the product is truncated without any other intervening action the resulting product (single length) will have 60-32 = 28 binary places. If the desired result is a number also having 30 b.p. after truncation the original product must be shifted up two places (giving 62) before truncation takes place to produce 62-32 = 30 binary places.

Example 2.

$$\frac{1}{10}\text{th (30 b.p.)} = \overset{\xleftrightarrow{30bp}}{\boxed{0110011001100110011001100110011000.00}}$$

$$\frac{1}{10}\text{th x } \frac{1}{10}\text{th} = \frac{1}{100}\text{th to 60 b.p.}$$

$$\underset{\xleftrightarrow{60bp}}{\boxed{10101111\ 000101 \quad 1010111100010100000\cdot0000}}$$

After truncation.

$$\frac{1}{100}\text{th (28 d.p.)} \ \underset{\xleftrightarrow{28}}{\boxed{01 \ldots \ldots \ldots \ldots 1010111100010100000\ 0000}}$$

To obtain 30 b.p. in the single length result the product should be moved up two places. This would produce

$$\underset{\xleftrightarrow{62}}{\boxed{101011\ 11000101 \ldots 1010111100010100000.00}}$$

which after truncation would give

$$\underset{\xleftrightarrow{30bp}}{\boxed{0101 \ldots 1010111100010100000.00}}$$

There is one other process which must be performed if balanced errors are to be produced. When two five digit decimal numbers are multiplied, each having five decimal places the result has ten decimal places.

e.g. 0.12345 x 0.41235

= 0.0509046075

If we drop the last five figures (truncate), the number remaining is accurate to five figures

= 0.05090

If the original numbers were

0.12345 x 0.32375

giving a product

0.0399669375

the truncated result 0.03996 would not be a true five figure representation of the product as the highest figure of those discarded is a 6.

It is necessary to ROUND-OFF to the correct number of figures, in this case five, before truncating to produce

0.03997

Working in binary is no excuse for not following normal computation practice and it is usual to round-off the double length product before truncating. To round off in binary we add a one in the highest-digit position of those discarded. If this digit is a one, the next higher digit is increased by one carried up, otherwise the truncated number is unchanged.

Multiplication of Positive Integers.

To form the product of two positive integers X and Y the rules are

(a) Send X to TS16

(b) Send Y to $DS21_3$

(c) Clear $DS21_2$

(d) Start Multiplication in an ODD MINOR cycle using the instruction 0-24.

The multiplication lasts for 65 minor cycles and throughout this time DS21 and TS16 must not be used for other purposes. When the programmer has aquired a knowledge of advanced programming technique he may use DS21 and TS16 during 0-24 operations but the beginner is not advised to do so.
The product XY is left in $DS21_{2,3}$ with the least significant digit of XY in the P_1 position of 21_2 (the less significant half).

The instructions required are

X - 16		Send X to TS16
Y - 21_3		Send Y to $DS21_3$
30 - 21_2		Clear 21_2
0 - 24	(ODD)	Start multiplication
1 - 1		Waste instruction
21 - 20	(d)	XY sent to DS20.

The waste instruction is a dummy which is necessary to waste time until 65 minor cycles have elapsed between the commencement of multiplication by 0-24 and the extraction of the result by 21-20 (d).

Multiplication of Positive Non-Integers.

Multiplication of two single length numbers results in a double length product which is usually truncated to single length. Truncation only occurs after the double length product is shifted and rounded off to give the desired number of binary places in the single length result.

If the factors to be multiplied are x and y with x to p binary places represented as the integer $X = x.2^p$ and y to q binary places represented as $Y = y.2^q$ and if the result xy is required to r binary places, the final single length number after shift and truncation will be represented as $XY.2^r$.

After multiplication the result is

$$XY . 2^{p+q}$$

After a shift up of s places the result is

$$XY . 2^{p+q+s}$$

After truncation we lose 32 places giving

$$XY . 2^{p+q+s-32}$$

and we require

$$p + q + s - 32 = r$$

therefore $\boxed{s = 32 + r - p - q}$

To obtain an upward shift of s places in DS21 we use the instruction

21 - 22 (2s m.c.) (e, o)

T.C.B. must be OFF (as it will be after multiplication) and (e, o) has been included as a reminder that the transfer must end in an odd minor cycle.

When the double length product has been shifted up and before the bottom 32 digits are discarded, the top digit of the discarded word is P_{32} in 21_2. To round off one might assume that we use

$29 - 22_2$

This is not so however, for SOURCE 29 looks like a negative number transferred to 22_2 in an even minor cycle. By the rules of EXTENDED ADDITION this will cause 32 copies of the sign digit to be added in 22_3.

To overcome this difficulty we round off by subtracting a P_{32} in 21_2. The instruction

$29 - 23_2$

rounds off the single length number in 21_3 and the instruction

$21_2 - 23_2$

rounds off 21_3 and clears 21_2, a useful feature in continued multiplication. To multiply two non-integers the instructions are the same as for integers with the addition of the extra shift and round off instructions

x - 16	$x = X.2^p$	TS16
$y - 21_3$	$y = Y.2^q$	21_3
$30 - 21_2$	Clear 21_2	
0 - 24	MULT (ODD m.c.)	
1 - 1	Waste instruction	
21 - 22 (2s m.c.)	e, o Shift-up s places.	
$29 - 23_2$	Round off	
Result in 21_3	xy to (p+q-32+S) b.p.	

Multiplication of Signed Numbers.

The multiplier treats all numbers as positive integers, which means that a sign correction must be applied when negative numbers are included in the factors of multiplication.

We may work out the correction rule as follows:

Both factors positive.

Factors X, Y	Multiplier INPUTS X, Y
Product in $DS21_{2,3}$	XY
Required product	XY
Correction	NONE.

One Factor Negative.

Factors, X, -Y	Multiplier inputs X, 2^{32}-Y
Product in $DS21_{2,3}$	X $(2^{32} - Y)$

Required product -XY signed double length = $2^{64} - XY$

Correction required $2^{64} - 2^{32}X = 2^{32}(2^{32} - X) = -2^{32}X$

Both Factors Negative.

Factors -X, -Y	Multiplier inputs $2^{32} - X$, $2^{32} - Y$.
Product in $DS21_{2,3}$	$(2^{32} - X)(2^{32} - Y)$ $= 2^{64} - 2^{32}X - 2^{32}Y + XY$
Correction required	$- 2^{32}(-X) - 2^{32}(-Y)$

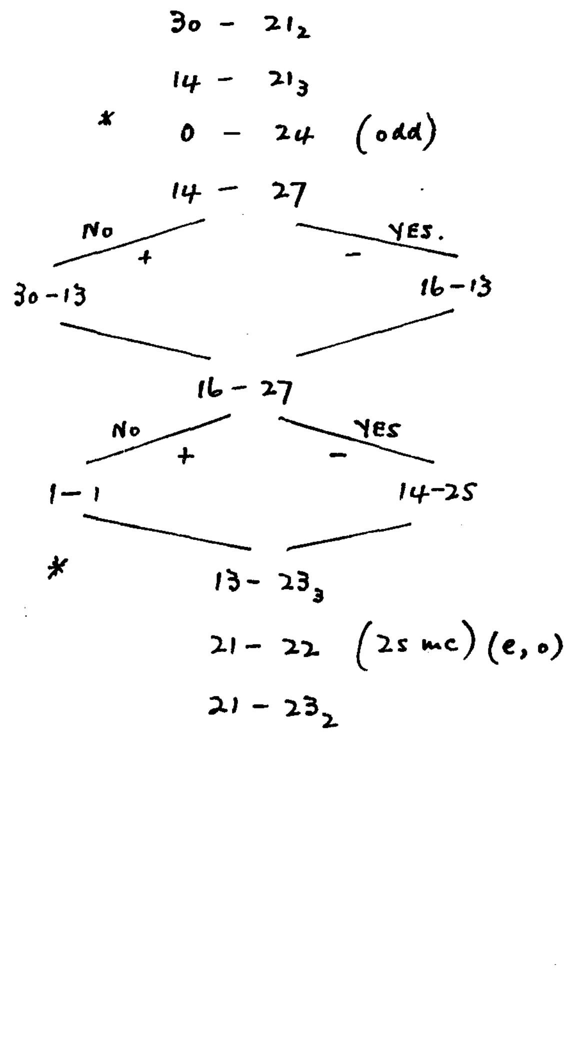
When only one factor is negative the correction needed is that of subtracting the other factor shifted up 32 places. When both factors are negative we must subtract both factors shifted up 32 places. These observations may be combined in the correction rule. <u>"If either factor is negative, subtract the other factor from the top half of the product in 21_3."</u>

Most often a programmer who requires multiplication of signed numbers will use one of the many multiplication subroutines which exist in the DEUCE Library of Subroutines. In these subroutines, signs and binary place shifts are dealt with. If it is necessary for a programmer to write his own sign corrected multiplication routine the instructions required are not likely to be different from those given below which form the basis of most multiplication subroutines.

<u>Example</u>.

To obtain the signed product of two factors stored in TS14 and TS16.

NOTE One factor is already in TS16.



Clear 21_2

1st factor to 21_3

Start MULTIPLICATION

Is 1st factor negative?

YES: Send other factor to TS 13

NO: Clear TS 13

Is 2nd factor negative?

YES: Add other factor to TS 13

No Do nothing

Subtract correction from 21_3

Shift up s places

Round off and clear 21_2.

* The sign correction instruction $13-23_3$ must not be executed less than 65 minor cycles after multiplication starts. The time between starting the multiplication and making the correction can be usefully occupied by the instructions which sort out the signs of the factors and prepare the correction.

<u>Example</u>.

Two numbers, each less than 10^4 are to be multiplied and the result placed in 8_5. The numbers are known to be positive integers and are stored originally in 7_5 and 7_{13}. The result will be less than 10^8 and as this is single length the product will not extend beyond 21_2.

7_5	- 21_3	one factor to 21_3
7_{13}	- 16	other factor to TS16
30	- 21_2	Clear 21_2
0	- 24 (odd)	Start MULT
1	- 1	Waste time
21_2	- 16	
16	- 8_5	Transfer product to 8_5

Example.

Two numbers x (30 b.p.) and y (30 b.p.) are to be multiplied and the product obtained also to 30 b.p.

x	- 16	x → 16
y	- 21_3	y → 21_3
30	- 21_2	Clear 21_2
0	- 24 (odd)	Start mult
1	- 1	to give 60 b.p.
21	- 22 (4 m.c.)	Shift to give 62 b.p.
		Round off and clear 21_2
21_2	- 23_2	leaving 30 b.p. in 21_3

Result xy (30 b.p.) in 21_3 with 21_2 clear.

Example.

In one small section of a programme the area of circles is calculated. The values of radius range up to 2 inches and the radius is held to 29 b.p. π is held to 29 b.p. and the final result is required to as many b.p. as possible.

NOTES (a) Since π = 3.14159 ... has an integral part 3 it can not be held to more than 29 b.p. for one digit (P_{32}) is required for the sign and two digits P_{31}, P_{30} are required to accommodate the integer 3. This becomes 32-3 = 29 digits available for the fractional part .14159 ....etc.

(b) The maximum value of radius is 2 therefore maximum area is ≃ 12.5 sq. inches. To accommodate an integer 12 we need 4 digits which, allowing one for the sign, leaves 32-5 = 27 binary places for the fractional part.

(c) The partial result r^2 can be held to 28 binary places.

The instructions are

r - 16	Radius to TS16
16 - 21_3	and to 21_3
30 - 21_2	Clear 21_2
0 - 24 (odd)	Start mult to give
1 - 1	58 b.p.
21 - 22 (4 m.c.) (e.o.)	Shift up to give 60 = 32 + 28 b.p.
21_2 - 23_2	Round off to 28 b.p. and clear 21_2
π - 16	r^2 is now in 21_3
0 - 24 (odd)	Start to form πr^2
1 - 1	with 29 + 28 = 57 b.p.
21 - 22 (4) (e,o)	Shift up to give 59 = 32 + 27 b.p.
21 - 23_2	Round off to 27 b.p. and clear 21_2
Result πr^2 (27 b.p.)	
in 21_3	
with 21_2 clear.	

CLASS EXERCISES.

1. Write the instructions to multiply the two positive integers in TS15 and TS16 placing the result in TS13 [the original numbers are each less than 10^4]

2. Write the instructions to multiply the signed integers in TS15 and TS16 placing the result in DS19. The result must be signed, double length.

3. The number in TS15 represents the radius of a sphere to 16 b.p. Write the instructions to calculate $4 \pi r^2$. All partial products may be held to 16 b.p. [π is held to 16 b.p. and 4 is held as an integer x P_1.]

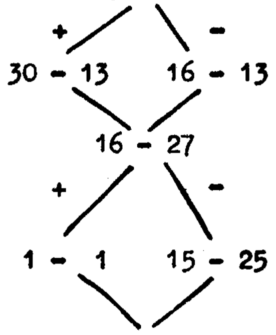
Trap. Is the 4 really necessary?

4. Write the instructions to form x^n to 30 b.p. given x to 30 b.p. and a counter n. [$x \leq 1$]. Each partial product should be correctly rounded off.

5. Calculate $\frac{x}{1000}$ by multiplying x by $\frac{1}{1000}$ [$x < 1000$ held to 20 b.p.] $2^{40} \times \frac{1}{1000}$ is available as a multiplier. The result is required to 30 b.p.

EXERCISE SOLUTIONS.

1.	15 - 21_3	
	30 - 21_2	
	0 - 24	(odd m.c.)
	1 - 1	waste time.
	21_2- 13	
2.	15 - 21_3	
	30 - 21_2	
	0 - 24	
	15 - 27	



	13 - 23_3	after mult. finished.
	21 - 19	(2 m.c.)
3.	15 - 16	r (16 b.p.)
	15 - 21_3	
	30 - 21_2	
	0 - 24	(odd) Mult. gives r^2 (32 b.p.)
	1 - 1	waste time.
	21 - 22	(32 m.c.) shift to 48 b.p. (start shift in even m.c.)
	21 - 23_2	clear 21_2 and round off 21_3 to 16 b.p.
	- 16	π (16 b.p.)
	0 - 24	(odd m.c.) gives πr^2 to 32 b.p.
	1 - 1	waste time.
	21 - 22	(4 m.c.) multiply by 4 by shifting. Given 4 π r^2 to 32 b.p. double length.
4.	- 16	x (30 b.p.)
	16 - 21_3	
	30 - 21_2	clear 21_2.
	- 13	n (0 b.p.)
	27 - 26	(n - 1)
	13 - 28	z
	nz	Result in 21_3 to 30 b.p.
	0 - 24	(odd m.c.)
	1 - 1	waste time.
	21 - 22	(4 m.c. e, o) shift to 62 b.p. double length.
	21 - 23_2	clear 21_2 and round off to 30 b.p.
5.	- 16	x. (20 b.p.)
	- 21_3	$^1/1000$ (40 b.p.)
	30 - 21_2	Clear 21_2
	0 - 24	(odd m.c.)
	1 - 1	waste time. Mult. gives $^x/1000$ to 60 b.p. double length.
	21 - 22	(4 m.c. e, o) $^x/1000$ (62 b.p.)
	29 - 23_2	round off to single length, 30 b.p.
	21_3-	result from 21_3

5

LECTURE 5.

5.0 DIVISION.

One arithmetic process on DEUCE which causes a beginner trouble and makes even experienced programmers proceed with caution is DIVISION using the automatic divider. The action of this facility on DEUCE is entirely consistent with normal division practice in the decimal scale, if the same initial conditions are postulated, so a preliminary study of decimal division is indicated.

5.1 DECIMAL DIVISION.

If one integer, say 43 is divided by another integer, say 5, the result is 8 with 3 over.

i.e.	DIVIDEND	43 (A)	QUOTIENT	8	Q
	DIVISOR (less than dividend)	5 (B)	REMAINDER	3	R

or $43 = 8 \times 5 + 3$

$A = Q \times B + R$

This is very simple and straightforward if the divisor "goes into" the dividend. If the divisor is greater than the dividend matters are not as straightforward. If A (the DIVIDEND)=1 and B (the DIVIDEND)=7 we can write

$$\frac{1}{7} = 0 \text{ and } 1 \text{ Remainder}$$

but we have not really progressed very far. We want to divide 1 by 7 and obtain a decimal fraction as the quotient. Let us settle for five figure accuracy and proceed as at school

$$\begin{array}{r}
.14285 \\
7)\,1.\underline{0} \\
\underline{7} \\
3\underline{0} \\
\underline{28} \\
2\underline{0} \\
\underline{14} \\
6\underline{0} \\
\underline{56} \\
4\underline{0} \\
\underline{35} \\
5
\end{array}$$

We place a decimal point after the 1, add a nought and proceed as if we are dividing 10 by 7. This gives a remainder of 3, so we add another nought and carry on until our required 5 figures have appeared in the quotient. The noughts added one at a time are underlined in the example and the same sequence of quotient digits would be obtained if we first multiplied the dividend by 10^5 (because we require 5 figures) and convert the process to an integer divisor

i.e. $10^5 \times 1 = (14285) \times 7 + 5$

or $10^5 \times A = Q \times B + R$

Now we require $^A/B$ which is given by

$$^A/B = {}^Q/10^5 + \frac{1}{10^5}\left(\frac{R}{B}\right)$$

and $\frac{Q}{10^5} = 0.14285$, i.e. we put the decimal point in the right place.

If we required ten figures in the quotient this is equivalent to performing the division of

$10^{10}A$ by B expressed in the equation

$$10^{10} A = Q B + R$$

When these processes are carried out using pencil and paper or (within limits) on a desk machine we are free to stop when we have enough figures in the quotient. If the process is mechanised and made fully automatic we must decide that all divisions are performed to a fixed number of places, say 10 for a decimal division. This is what we arrange to do on DEUCE except that the number system is binary and we settle for 31 binary digits in the quotient.

5.2 BINARY DIVISION.

Division of one binary integer by another, say 13 (1011) by 3 (11) gives 4 (001) as quotient and 1 (1) as remainder. The equation

$$A = QB + R$$

is still valid and independent of the scale of notation.

If we divide 1 by 7 in binary to produce a binary fraction, we proceed just as in decimal

```
          .001001001      Least significant digit
    111 ) 1.000           on the RIGHT
            111
          0 001000
               111
                 1000
                  111
                    1
```

We appear to require a greater number of added noughts to start with but this is perfectly reasonable, since each nought corresponds to multiplying by 2 in binary.

For the 9 binary figures in the quotient we have used 9 extra noughts. This corresponds to multiplying the original dividend by 2^9

i.e. $2^9 A = QB + R$

check $A = 1 \qquad Q = 1001001 = 73$

$2^9 = 512 \qquad R = 1$

and $\underline{512 = 7 \times 73 + 1}$

obtaining a binary fractional quotient with p binary places corresponds to performing the division of A by B to satisfy

$$2^p A = QB + R$$

The DEUCE automatic divider works to 31 binary places and the formal rules governing the dividend (A) divisor (B), quotient (Q) and remainder (R) are set out below.

1. A, B, Q and R are integers.
2. $|A| < |B|$
3. A and B can be of either sign
4. The quotient will be correctly signed automatically and this and the remainder are related to A and B by the equations

$$\underline{2^{31} A = QB + R}$$

with $\underline{0 \leq R < B \text{ if } B > 0}$

$\underline{B \leq R < 0 \text{ if } B < 0}$

DEUCE divisions are valid only when we divide one integer (A) by another integer (B) which is numerically greater than A. The fractional result is obtained in 21_2 with 31 binary places, i.e. the position of the binary point is between P_{31} and P_{32} in 21_2.

Example 1.

To divide 3 by 5.

Result $^3/5$ to 31 b.p. in 21_2

$$\boxed{\ldots\ \ldots\ \ldots\ 1001001001001001001.0}$$

The integer in 21_2 is $2^{31} \cdot \frac{3}{5}$

Example 2.

To divide 5 by 10.

Result = $\frac{1}{2}$ in 21_2 to 31 b.p.

The result is exact since $^5/10$ is an exact binary fraction whereas $^3/5$ in the previous example leads to a recurring binary fraction.

$$21_2 \ \boxed{00\ \ldots\ \ldots\ \ldots\ 000001.0}$$

The integer in 21_2 is $2^{31} \cdot {}^5/10 = 2^{31} \cdot \frac{1}{2} = 2^{30}$

Example 3.

To divide 50 by 10.

We cannot use the automatic divider until the numbers are adjusted so that B is greater than A. Let us shift our divisor 10 up 3 places (binary places) and so multiply by $2^3 = 8$. If now we divide 50 by 80 we shall obtain a result which is 8 times too small. We adjust the position of the binary point in the final result.

$\frac{50}{80} = \frac{5}{8} =$ 101.0 positioned in 21_2 as

$$\boxed{\underset{\longleftarrow\ 31\text{bp}\ \longrightarrow}{000\ \text{- - - - - - - - - - - -}\ 00001 01.0}}$$

$\frac{50}{10} = \frac{50}{80} \times 8$, so we move the binary point three places to the left giving 31-3 = 28 binary places with the integer '5' above the position of the binary point

$$\underset{\longleftarrow\ 28\text{bp}\ \longrightarrow}{\boxed{000\ \text{- - - - - - - - - - - -}\ 0000.101 0}}$$

So the DEUCE can after all divide 50 by 10 and obtain 5 as the result. The quotient 5 may be positioned at some odd place in the register but it can be moved to the P_1 position in 21_2 by a left shift or, if T.C.B. is OFF, moved right into 21_3, whichever is the more convenient.

5.3 BINARY PLACES IN DIVISION.

The beginner is apt to think that the DEUCE divider is restricted because it can only divide one integer A by a numerically larger integer B. Division of any number x, not necessarily integral, by any other number can be performed if the process is carried out in a methodical manner.

Suppose we have x to p binary places and y to q binary places and we wish to form $\frac{x}{y}$.

In DEUCE we have $X = 2^p x$
and $Y = 2^q y$
where X and Y are integers.

We must now consider whether Y is greater than X. If it is, then X and Y are already in a form suitable for automatic division. In this case the result obtained in 21_2 is

$$\begin{aligned} Q &= 2^{31} \frac{X}{Y} \\ &= 2^{31} \left(\frac{2^p x}{2^q y}\right) \\ &= 2^{31+p-q} \left(\frac{x}{y}\right) \end{aligned}$$

which is the quotient $(\frac{x}{y})$ to $(31 + p - q)$ binary places.

Example.

We want to convert lengths in KILOMETRES to corresponding lengths in MILES. All the KILOMETRE lengths are less than 100 and expressed to 24 b.p. 1.609 KM = 1 mile and 1.609 is held to 30 b.p.

Method.

Divide KILOMETRES by 1.609.

Integers $\dfrac{\text{KM} \times 2^{24}}{1.609 \times 2^{30}}$ Result: $2^{31} \left(\dfrac{2^{24} \times \text{KM}}{2^{30} \times 1.609}\right)$

therefore miles are given to $31 + 24 - 30 = 25$ b.p.

The division of K.M. x 2^{24} by 1.609 x 2^{30} is valid since the maximum value of KM is 100 and

$$100 \times 2^{24} < 1.609 \times 64 \times 2^{24}$$
$$< 1.609 \times 2^{30}$$

The maximum number of miles obtained in the result will be 62.1 corresponding to 100 KM. The integer part of this is contained in 6 digits and, allowing one for the sign, there remains a maximum of 25 binary places which can be used.

It is not always possible to divide without first shifting either the DIVIDEND down or the DIVISOR up to make sure that the integer representing the DIVIDEND is less than the integer representing the DIVISOR.

Example.

To convert MILES to KILOMETRES. The variables representing MILES are not greater than 100 and held to 24 b.p. 0.621 MILES = ONE KM. and 0.621 is held to 31 b.p.

If we divide MILES by 0.621 the result will be KM., and we should expect more KM than corresponding miles. Since the miles are already held to the maximum number of binary places (24) and because the greatest number of KM which can occur is 160.9 requiring 23 b.p., we shall need fewer b.p. in the quotient than in the dividend. Further the divisor must be shifted up or the dividend shifted down. The divisor is already held to the maximum number of binary places (31) and shifting this up would cause it to go out of length. We must therefore shift down the dividend before starting the division. Let us shift it down two places which gives 24-2 = 22 b.p. We now divide the integers

$$\text{MILES} = 2^{22} \times \text{MILES}$$

$$\text{by } 0.621 = 2^{31} \times .621$$

$$\text{and obtain KM} = 2^{31} \times \frac{2^{22} \text{ miles}}{2^{31} \times .621}$$
$$= 2^{22} \left(\frac{\text{miles}}{.621}\right)$$

i.e. we obtain KM expressed to 22 binary places.

When more binary places are obtained from a division operation than are required the quotient can be shifted down (left) after rounding off. If the result of a division is ultimately required to, say, 30 b.p. it is advisable to obtain first the quotient to 31 b.p., round off and then shift down one place. This is only possible if a quotient to 31 b.p. is within single length.

In general, to obtain a rounded quotient to r binary places, adjust A and B to give (r+1) binary places after division, round off and shift down one place. If r binary places is the maximum which can be accommodated, this will not be possible since (r+1) b.p. would cause the result of division to be out of single length.

<u>Example</u>.

To calculate $\frac{x}{100}$ given the integers 10^3x and 10^5, and knowing $1 \leqslant x \leqslant 10$. The maximum value of $\frac{x}{100} = \frac{10}{100} = {}^1/10^{th}$. ${}^1/10^{th}$ in binary is

$$- - \text{-----------} \; 00110011001100110011000\cdot0$$

We can hold ${}^1/10$th to 33 binary places (34 if necessary) in which case ${}^1/10$th in a DEUCE register would appear as

$$\boxed{\qquad\qquad\qquad \cdots\cdots 11001100110011001100}\,\bigcirc\cdot$$

With the position of the binary point outside the register. To obtain $\frac{x}{100}$ rounded off to 32 b.p. proceed as follows

1. Shift 10^3x up two places giving $2^2\ (10^3x)$.
2. Divide $2^2\ (10^3x)$ by 10^5

   giving $2^{31}\ (\frac{2^2\ 10^3x}{10^5}) = 2^{33}\ (\frac{x}{100})$

   i.e. $\frac{x}{100}$ to 33 b.p.
3. Add P_1 to round off.
4. Shift down one place.

5.4 <u>DIVISION INSTRUCTIONS.</u>

The instructions required to divide A by B are

1. Send A to $DS21_3$
2. Send B to TS16.
3. Start Division in an odd minor cycle with 1-24.
4. Do not use TS16 or DS21 for a further 66 m.c.

<u>NOTE</u> (a) Do not send B to TS16 in the odd minor cycle immediately preceding the odd minor cycle in which DIVISION starts.

(b) Remember 1-24 turns T.C.B. ON and that T.C.B. is ON after division is complete.

<u>Example</u>.

10^3x (an integer) is held in TS15. 10^5 is available in 2_4. Obtain $\frac{x}{100}$ to 32 b.p., knowing that $\frac{x}{100}$ does not exceed $\frac{1}{10}$th.

The instructions are

15 - 14	$10^3x \longrightarrow 14$		
24 - 14 (d)	$2^2\ (10^3x) \longrightarrow 14$		
14 - 21_3	$2^2\ (10^3x) \longrightarrow 21_3$	A	
2_4 - 16	$(10^5) \longrightarrow 16$	B	
1 - 24 (odd)	START DIVISION (33 b.p.)		
1 - 1	Waste time		
27 - 22_2	Round off to 32 b.p.		
22_2 - 21_2	Shift down to leave 32 b.p.		

Result in 21_2
with T.C.B. ON.

Example.

To obtain $\frac{x}{100}$ to 20 b.p., rounded off, knowing $\frac{x}{100} \leqslant \frac{1}{10}$th. 10^3x is present in TS15 and 10^5 is available in 2_4.

$15 - 21_3$		(10^3x) to 21_3 A
$2_4 - 16$		10^5 to TS16 B
$1 - 24$	(odd)	DIVIDE giving 31 b.p.
$1 - 1$		Waste time
$P_{11} - 22_2$		Round off to 20 b.p.
$22 - 21$	(22 m.c. e, o)	Shift down 11 places
Result in		to leave 20 b.p.
21_2 with T.C.B.		
ON.		

In this case we divide the integers originally given and obtain 31 binary places. We are about to discard 11 places and leave 20. Before doing so we add a digit in the top position of those which will be discarded to round off those which are retained.

In general, before a left shift of s places add a P_s as round off and shift left s places by a transfer of 22-21 for 2S minor cycles (e, o).

5.5 QUOTIENT ERROR.

The quotient produced by the automatic divider will either give the exact quotient to 31 binary places or a quotient which falls short by P_1.

e.g. $\frac{+5}{+10}$ gives $+\frac{1}{2}$

while $\frac{-5}{-10}$ should give $+\frac{1}{2}$ but actually produces $(+\frac{1}{2} - P_1)$

5.6 TRUE REMAINDER RECOVERY.

The contents of 21_3 after division represent a quantity r known as the MACHINE REMAINDER. This does not correspond with R in the relation

$$2^{31} A = QB + R \qquad 0 \leqslant R < B \text{ for } B > 0$$
$$B \leqslant R < 0 \text{ for } B < 0$$

To obtain R from r $\left(\text{and this is only possible if } B < 2^{30}\right)$ we use the relation

$$R = \tfrac{1}{2}\left(\tfrac{1}{2}r + B\right)$$

This is the process

1. Shift 21_3 down one place.
2. Add B.
3. Shift 21_3 down one more place.

The instructions required at the end of division are

$22_3 - 21_3$	Shift down r in 21_3
$16 - 22_3$	Add B from TS16.
$22_3 - 21_3$	Shift down to obtain R in 21_3.

CLASS EXERCISE.

1. $5P_1$ and $7P_1$ are stored in 8_{16} and 5_{13}. Write the instructions to calculate $\frac{5}{7}$ to 31 b.p.

2. Repeat Example 1 to product a rounded off result to 30 b.p.

3. Repeat Example 1 to produce a rounded off result to 20 b.p.

4. The binary number N in 8_4 represents binary pence. Convert this to £ and fractional £ by dividing by 240.

$$N < 240 \times 10^6$$

$240P_1$ is available in 2_0.

How many binary places are available in the result?

5. The binary numbers in 17_0, 17_1, 17_2 represents three quantities x_1, x_2, x_3 to 16 b.p.

$$x_1 < 100$$
$$x_2 < 1000$$
$$x_3 < 500$$

Scale these numbers so that each is less than 10 and held to 27 b.p. The same scaling factor should be used for each number and you are to choose this factor.

EXERCISE SOLUTIONS.

1.	5_{13} - 16	$7P_1$ as divisor.
	8_{16} - 13	
	13 - 21_3	$5P_1$ as dividend (use short store to transfer from even to odd m.c.)
	1 - 24	(odd m.c.)
	1 - 1	waste time.
	21_2 -	result from 21_2.
2.	5_{13} - 16	
	8_{16} - 13	
	13 - 21_3	
	1 - 24	(odd m.c.)
	1 - 1	
	27 - 22_2	Add P_1 for round-off.
	22_2 -	Result obtained to 30 b.p. from Source 22_2.
3.	5_{13} - 14	$7P_1$
	24 - 14	(10 m.c.) $7P_{11} = 7 \times 2^{10}$.
	14 - 16	
	8_{16} - 14	
	14 - 21_3	dividend = 5×2^0
	1 - 24	(odd m.c.) division gives $\left[5 \times 2^0/7 \times 2^{10}\right]$. $2^{31} = \frac{5}{7} \times 2^{21}$
	1 - 1	
	27 - 22_2	add P_1 as round-off.
	22_2 -	Source 22_2 gives result to20 b.p.
4.	2_0 - 14	$240\ P_1 = 240 \times 2^0$
	24 - 14	(20 m.c.) 240×2^{20}.
	14 - 16	
	8_4 - 14	
	14 - 21_3	
	1 - 24	(odd m.c.) $\left[\frac{N}{240} \cdot \frac{2^0}{2^{20}}\right]$. $2^{31} = \frac{N}{240}$. 2" therefore 11 b.p. result.
	1 - 1	
	27 - 22_2	round off.
	22_2 -	Source 22_2 gives result to 10 b.p.

5. Since x_1, x_2, x_3 are all less than 1000, greatest possible is 999 for x_2 therefore scale by 100 to make greatest 9.99 (i.e. less than 10).

If scale factor is 100×2^p (i.e. p binary places)

$$\frac{x}{100} \cdot \frac{2^{16}}{2^p} \cdot 2^{31} = 2^{27}$$

therefore $p = 16 + 31 - 27 = 20$

therefore scale by 100×2^{20}

- 16	100 x 2^{20} as divisor.
17_0 - 13	
13 - 21_3	x_1 as dividend.
1 - 24	(odd m.c.)
1 - 1	waste time (nothing else to do)
17_1 - 21_3	x_2 as dividend after division completed.
21_2 - 17_0	store $^{x}1/100$ back in 17_0.
1 - 24	(odd m.c.)
17_2 - 13	x_3 into 13 as waste time instruction.
13 - 21_3	x_3 as dividend after division completed.
21_2 - 13	$^{x}2/100$ into 13.
1 - 24	(odd m.c.)
13 - 17_1	store $^{x}2/100$ back in 17_1 as waste instruction.
21_2 - 17_2	store $^{x}3/100$ back in 17_2 after division.

__N.B.__ Useful instructions that do not interfere with the division process are used where possible as the waste time instruction whilst division takes place.

6

LECTURE 6.

6.1 INTRODUCTION.

Previous lectures have been concerned with describing the ORDER CODE of DEUCE, that is the facilities which are available through each of the 32 Sources and Destinations and the use of single, double or long transfers. Once a programmer knows this order code he can prepare an outline programme in the following steps.

(a) Prepare a LOGICAL FLOW DIAGRAM of the pr oblem.

(b) Prepare a DEUCE flow diagram of instructions using, where necessary, "pseudo instructions".

A pseudo instruction is one in which a symbol is used to indicate the transfer of a variable before storage positions are allocated to each variable.

Example 1.

It is required to form and store a pattern consisting of P_1, P_{17}, P_{32}.

We could write

P_1 - 13

P_{17} - 25

P_{32} - 25

13 - (P_1, P_{17}, P_{32})

As we already have sources for P_1, P_{17} and P_{32} we would in fact write, in this case,

27 - 13

28 - 25

29 - 25

13 - (P_1, P_{17}, P_{32})

leaving open, for the moment, where (P_1, P_{17}, P_{32}) will be stored. When, eventually, it is decided to store this quantity in 7_{15} the final flow diagram becomes

27 - 13

28 - 25

29 - 25

13 - 7_{15} (P_1, P_{17}, P_{32})

Example 2.

To calculate the length along a railway line at which stations occur, knowing the lengths of each gradient. The process of summing the incremental distances might appear on the outline programme as

L - 13

Δ L - 25

13 - L

indicating that the quantity L is brought from wherever it is stored, increased by the amount Δ L from wherever it is stored and the new value of L put back in its rightful place. When it is finally decided to store L in 5_{13} and Δ L in 6_3 these instructions would becor

5_{13} - 13 (L)

6_3 - 25 (Δ L)

13 - 5_{13} ($L + \Delta L) \rightarrow L$

Before instructions representing a programme enter DEUCE they must be subjected to two further processes

(c) Storage allocation of Data and Instructions.

(d) Detail coding.

The above examples illustrate the storage allocation of data. Instruction storage allocation involves specifying the storage position which each instruction will occupy when the programme is in the machine.

Some machines store their instructions in the same sequence as they appear on the flow diagram. Each instruction leads to the one in the next storage location (with special provision for discriminations).

DEUCE, however, stores its instructions in any order and it is one of the functions of each instruction to nominate its successor. Using the instructions in example 2 above, we could allocate 5_{13} - 13 to 2_0, 6_3-25 to 3_5 and 13-15_{13} to 4_{11} and indicate these instructions storage positions as follows

2_0	5_{13} - 13	(L)
3_5	6_3 - 25	(Δ L)
4_{11}	13 - 5_{13}	(L + Δ L) $\rightarrow$ L

The programmer knows that 2_0 leads to 3_5, 3_5 leads to 4_{11} and so on, but how does DEUCE know? How, for that matter does DEUCE know that instruction 2_0 must cause 5_{13} to transfer to TS13 and not, say 5_{15} or any other minor cycle of D.L. 5? The answer is, by coding the instructions so that they enter the machine with much more information than just a SOURCE and a DESTINATION.

6.2 DEUCE INSTRUCTION WORD.

Each instruction of the form S-D (m minor cycles) appearing on a DEUCE flow diagram must be expanded on a coding sheet to contain at least seven items of information showing

(a) Which Delay Line contains the next instruction.

(b) Which SOURCE is involved in the transfer.

(c) Which DESTINATION is involved in the transfer.

(d) Whether the transfer is for one, two or more than two minor cycles.

(e) When the transfer starts.

(f) When the next instruction transfer starts.

(g) Whether the machine can proceed normally or whether it must stop.

For single transfers (one minor cycle) the transfer ends in the minor cycle in which it starts, naturally. For double transfers (two minor cycles) the transfer ends in the minor cycle after the one in which it starts.

For long transfers, (more than 2 m.c.) the transfer ends when the next instruction transfer ends. This means that, for long transfers, (e) and (f) above go together.

Each instruction is contained in a 32 digit word and sections of a word are allocated to specify the requirements (a) to (g). Each instruction has the value

$$NP_1 + SP_5 + DP_{10} + CP_{15} + WP_{17} + TP_{26} + GP_{32}$$

N, N.I.S. or NEXT INSTRUCTION SOURCE uses P_2, P_3, P_4 and takes values of 0 to 7 to indicate which Delay Line contains the next instruction (N = 0 means D.L. 8)

S, SOURCE uses P_5 to P_9 and takes values 0 to 31 to indicate the SOURCE used in the transfer.

D, DESTINATION uses P_{10} to P_{14} and takes values 0 to 31 to indicate the DESTINATION of the transfer.

C, CHARACTERISTIC uses P_{15} and P_{16} and takes values 0 to 2 to specify the type of transfer

C = 0	SINGLE transfer,	1 minor cycle.
C = 1	LONG transfer,	3 to 32 m.c.
C = 2	DOUBLE transfer,	2 minor cycles.

W, WAIT NUMBER uses P_{17} to P_{21} and takes on values 0 to 31 to specify when the instruction starts.

T, TIMING NUMBER uses P_{26} to P_{30} and takes values 0-31 to specify when the next instruction transfer starts and when long transfers end.

G, GO DIGIT using P_{32} is 0 for STOP INSTRUCTIONS and 1 for GO instructions.

Since N can only take values 0-7 it follows that instructions can only be obeyed from D.L.'s 1-8. These are the instruction delay lines. Instructions may be stored anywhere but they must be in the NIS D.L.'s when they are operative.

Each instruction enters control in the minor cycle in which it is stored. Taking the instructions in Example 2

2_0	5_{13} - 13
3_5	6_3 - 25
4_{11}	13 - 5_{13}

the instruction 5_{13}- 13 enters control in m.c. 0, 6_3-25 enters control in m.c. 5 and 13-5_{13} enters control in m.c. 11. In general, an instruction

N S-D C W T G

stored in D.L. A in m.c. m is designated as

A_m S-D

on a flow diagram and enters control in m.c. m.

All the actions caused by an instruction are determined by the minor cycle in which it enters control.

TRANSFER The transfer starts in minor cycle

$$m + W + 2$$

NIS TRANSFER The next instruction enters control in minor cycle

$$m + T + 2$$

This gives the minor cycle of entry for the next instruction from which point we can start again.

DOUBLE TRANSFERS Start in m + W + 2 and end in m + W + 3.

LONG TRANSFERS Start in m + W + 2 and end in m + T + 2.

therefore number of minor cycles of transfer is

$$n = T - W + 1$$

NOTE (a) If any value of m + W + 2 or m + T + 2 exceeds 32, subtract 32.

(b) If T is less than W add 32 to T.

6.3 RULES OF CODING.

1. Determination of T.

An instruction stored in A_m leading to the next instruction B_{m}' requires a value of T given by

$$T = m' - m - 2$$

if T comes out negative add 32.

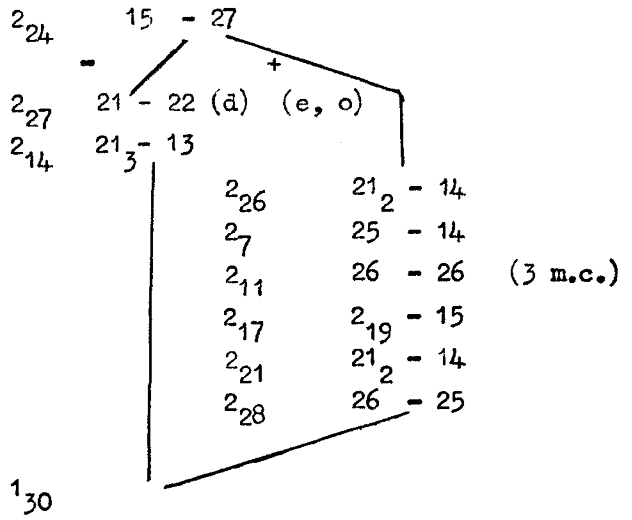
DPCS2

CLASS EXERCISES.

1. Write out the coding for each instruction in the following sequences:-

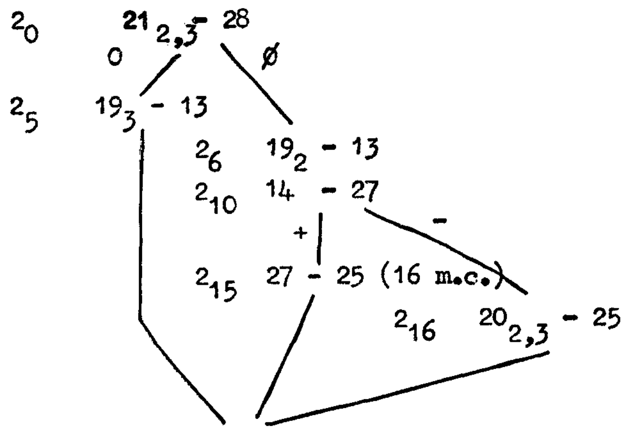
(a) 2_{0} $13 - 14$
2_{3} $19_{2} - 15$
2_{7} $17_{0} - 19_{2}X$
2_{13} $14 - 17_{0}$
2_{19}

(b)



2_{24} $15 - 27$ (− / +)
2_{27} $21 - 22$ (d) (e, o)
2_{14} $21_{3} - 13$
2_{26} $21_{2} - 14$
2_{7} $25 - 14$
2_{11} $26 - 26$ (3 m.c.)
2_{17} $2_{19} - 15$
2_{21} $21_{2} - 14$
2_{28} $26 - 25$
1_{30}

(c)



2_{0} $21_{2,3} - 28$ (0 / ∅)
2_{5} $19_{3} - 13$
2_{6} $19_{2} - 13$
2_{10} $14 - 27$ (+ / −)
2_{15} $27 - 25$ (16 m.c.)
2_{16} $20_{2,3} - 25$

3_{7} $13 - 1_{24,25,26,27}$
2_{27} $1_{20} - 17_{0}$ (4 m.c.)
2_{23} $11 - 2$ (32 m.c.)
1_{0}

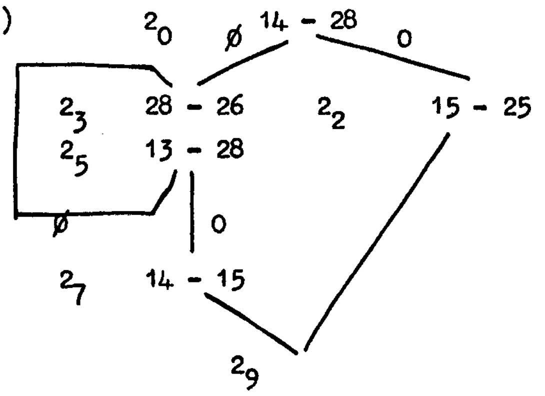
(d) 6_{10} $16 - 1_{17}$
6_{12} $18_{1} - 1_{21}$
7_{15} $19_{2} - 13$
8_{11} $18_{3} - 25$
6_{19} $13 - 14$
6_{28} $19_{3} - 17_{1}$
5_{1} $1_{22} - 15$
6_{24} $15 - 1_{26}$
2_{4}

2. Why is it not possible to code the following sequences of instructions?

(a) 1_{20} $1_{22} - 17_{0}$
1_{24}

(b) 1_{16} $1_{20} - 18_{0}$ (3 m.c.)
1_{26}

(c)



EXERCISE SOLUTIONS.

Storage Position.			N	S	D	C	W	T	G
1.	(a)	2_{0}	2	13	14		0	1	
		2_{3}	2	19	15		1	2	
		2_{7}	2	17	19		3	4	X
		2_{13}	2	14	17		1	4	
	(b)	2_{24}	2	15	27		0	0	
		2_{27}	2	21	22	2	1	17	
		2_{14}	1	21	13		1	14	
		2_{26}	2	21	14		0	11	
		2_{7}	2	25	14		0	2	
		2_{11}	2	26	26	1	2	4	
		2_{17}	2	2	15		0	2	
		2_{21}	2	21	14		1	5	
		2_{28}	1	26	25		0	0	
	(c)	2_{0}	2	21	28	2	0	3	
		2_{5}	3	19	13		0	0	
		2_{6}	2	19	13		0	2	
		2_{10}	2	14	27		0	3	
		2_{15}	3	27	25	1	7	22	
		2_{16}	3	20	25	2	0	21	
		3_{7}	2	13	1	1	15	18	
		2_{27}	2	1	17	1	23	26	
		2_{23}	1	11	2	1	8	7	
	(d)	6_{10}	6	16	1		5	0	
		6_{12}	7	18	1		7	1	
		7_{15}	0	19	13		1	26	
		8_{11}	6	18	25		2	6	
		6_{19}	6	13	14		0	7	
		6_{28}	5	19	17		3	3	
		5_{1}	6	1	15		19	21	
		6_{24}	2	15	1		0	10	

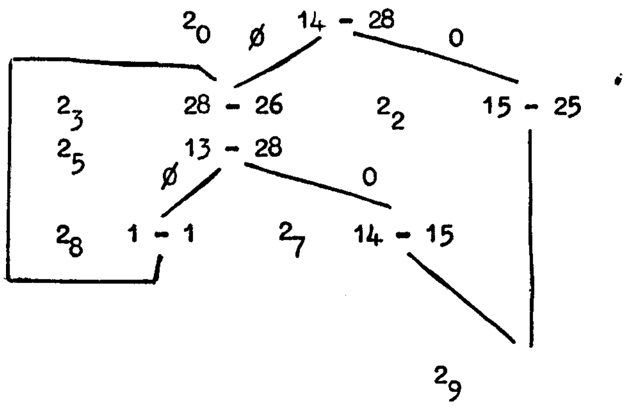
N.B. C and G left blank for the not common single, unstopped instruction.

2. (a) 1_{22} corresponds to 17_{2}, not 17_{0}.

(b) A long transfer for 3 minor cycles starting at 1_{20} <u>MUST</u> finish in minor cycle 22 and therefore the next instruction must be taken from a minor cycle 22.

(c) 2_{5} is incompatable. It is never possible to have two discriminations leading down similar arms to the same instruction, unless the other arms also lead to the same place.

This is often overcome by a waste instruction, as at 2_{8} below.


2_0
Ø
14 - 28
0
2_3
28 - 26
2_2
15 - 25
2_5
13 - 28
Ø
0
2_8
1 - 1
2_7
14 - 15
2_9

7

LECTURE 7.

7.1 INTRODUCTION.

The previous lecture has described the process of detailed coding and shown how each DEUCE instruction nominates a unique successor. All instructions taking part in the active part of a programme must be stored in D.L.'s 1-8 and each instruction enters control by an instruction transfer along an instruction highway. Since instructions are binary patterns it is possible to make arithmetic changes to instructions and we can add or subtract digits or groups of digits from instructions and so change them to different instructions while they are in their storage positions. Of course the action of altering instructions needs other instructions to do the work and it is obviously wasteful to change an instruction only once if it takes four instructions to make the change.

Instruction modification as it is called is a powerful technique but it is done only when it improves the efficiency of a programme and not because it looks smart.

Changes may be made to any or all the parts of an instruction N, S, D, C, W, T, G but the most usually altered part of an instruction is W.

7.2 INSTRUCTION MODIFICATION.

The wait number of an instruction determines the transfer minor cycle. By altering the wait number we can cause the same instruction to change the transfer minor cycle, advancing or retarding it by any step we choose.

Example 1.

To add the contents of TS14 to each minor cycle of D.L. 10.

(a) Without instruction modification we should proceed as follows

10_0 - 13
14 - 25 change 10_0
13 - 10_0
10_1 - 13
14 - 25 change 10_1
13 - 10_1
10_2 - 13
14 - 25 change 10_2
13 - 10_2
⋮
10_{31} - 13
14 - 25 change 10_{31}
13 - 10_{31}

needing 96 instructions!

(b) With instruction modification we might do better. Suppose the Source 10 instruction is I, stored in 5_{30} and the D10 instruction (I_2) is stored in 5_{29}. If we wish to change the wait number of the instructions by adding P_{17} (Source 28) we shall need to use TS13 (we could use DS21 but will not do so) and a slight re-organisation will be needed to effect our purpose. The instructions now become:

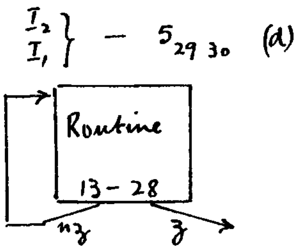
```
 ┌──────→  5_30 - 13     5_30 is  10_m - 13   (I_1)
 │         28   - 25     Add P_17
 │         13   - 5_30   Replace modified instruction in 5_30
 │  5_30   10_m - 13     Obey 530
 │         14   - 25     Temporary storage of result
 │         13   - 15     in TS15.
 │         5_29 - 13     5_29 is   15-10m   I_2
 │         28   - 25     Add P_17
 │         13   - 5_29   Replace 5_29 modified.
 │  5_29   15   - 10_m   Obey 5_29
 │         n    - 13
 │         27   - 26     Count and test using
 │         13   - n      initial counter
 │         13   - 28     of 32.
 └──────── nz      z
                          EXIT
```

This routine should be carefully studied. Two instructions are modified, 5_{29} and 5_{30}. Each is brought from where it is stored, has P_{17} added to it, replaced in its storage location and obeyed from where it is stored. Notice that as TS13 is replacing I_1 in 5_{30}, the same word is taken to control and obeyed. This is an important feature of DEUCE. An instruction may be transferred to a N.I.S. D.L. and obeyed from that D.L. in the same minor cycle.

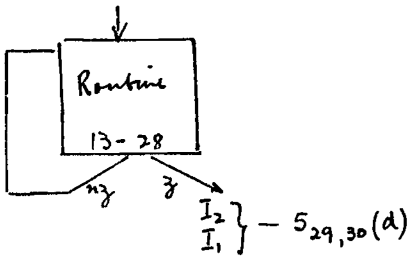
To determine when all the words in D.L.10 have been dealt with it is necessary to strike them off and a 'counter' initially set to 32 is counted down to zero. When it is zero the process is complete.

At the end of the process the instructions in 5_{29} and 5_{30} are the final modified copies. If, at some later stage in the programme it is required to repeat the same set of instructions then 5_{29} and5_{30} will need to be replaced by copies of the original instructions. This can be done before the routine is entered or after the routine is complete.

either



or

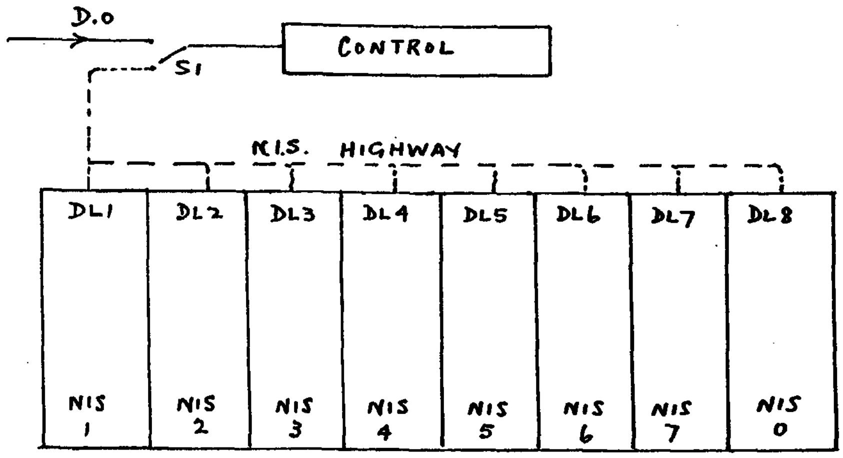


On the first occasion in the second case, copies of I_1 and I_2 will already be in 5_{30} and 5_{29} from the initial loading of programme into the machine.

7.3 DESTINATION O.

Instructions normally enter control through their own private highway. Special provision is made however to allow an instruction to enter control by D.O. using a normal S-D transfer. When this occurs the instruction which the machine would take from a NIS D.L. is prevented from entering control and the word sent to D.O. becomes the next instruction.

Figure 1 shows schematically the arrangement:



When a word is sent to D.O., the switch S1 changes over from the NIS highway to the D.O. path. D.O. is available in any minor cycle of machine operation and any SOURCE may be used.

Example 2.

To send the word in TS13 to CONTROL

write 13 - 0

Example 3.

To send the word from SOURCE 25 to CONTROL

write 25 - 0

Example 4.

To obey an instruction set up on the I.D. keys would require

0 - 0

In any sequence of instructions written on a flow diagram the storage location of each instruction is shown

e.g. 2_0 6_{13} - 13
5_2 27 - 25
7_{10} 13 - 6_{13}

2_0 leads to 5_2 which leads to 7_{10} and so on. Now suppose we include a 13-0 instruction in say, a sequence

2_0 6_{13} - 13 6_{13} is A-B
5_2 28 - 25
1_{28} 13 - 0

what instruction do we write next? The instruction obeyed is the word in TS13. This is the instruction in 6_{13} with P_{17} added; further more instructions enter control in the minor cycle in which they are stored only when they enter control on the N.I.S. highway. Here an instruction originally stored in 6_{13} is entering control by a transfer from TS13 we can not write

	2_{0}	$6_{13} - 13$
	5_{2}	$2_{8} - 25$
	1_{28}	$13 - 0$
	6_{13}	$A - B$

The instruction obeyed after 1_{28} is A-B but it is not obeyed from D.L. 6 and is probably not even obeyed in minor cycle 13. It enters control in the minor cycle of transfer from S13 to D0 and we give the instruction this storage minor cycle but as it is not obeyed from a NIS D.L. directly we describe it as a QUASI instruction and write

	2_{0}	$6_{13} - 13$
	5_{2}	$28 - 25$
	1_{28}	$13 - 0$
(6_{13})	Q_{30}	$(A - B)$

It is customary to write the quasi instruction in brackets to distinguish it and we also write its storage position alongside, also in brackets.

The instruction A-B, stored in 6_{13} but obeyed from m.c. 30 must be coded relative to the minor cycle from which it is obeyed (in this case m.c. 30).

Any instruction obeyed as Q_{q} is coded relative to minor cycle q irrespective of the minor cycle in which it is stored.

Any instruction of the form A-0 followed by a quasi instruction Q_{q} must be coded with equal wait and TIMING numbers such that

$$W = T = q - m - 2$$

where m is the storage position of A-0.

In any instruction of the form A-0 with equal timing numbers the NIS is not used except when A = 17 or 18 . It is usual to choose NIS = 0 to save punching holes in a card.

Examples.

	1_{28}		$13 - 0$			
	Q_{30}		( )			
is coded	N	S	D	C	W	T
	0	13	0		0	0
	1_{3}		$14 - 0$			
	Q_{15}		( )			
is coded	N	S	D	C	W	T
	0	14	0		10	10

If the wait number and timing number of an A-0 instruction are not EQUAL the A-0 instruction is wasted and leads to an instruction in the instruction store given by the NIS and T of the A-0 instruction.

e.g.	1_{10}		$14 - 0$			
coded as	N	S	D	C	W	T
	5	14	0		0	6
leads to	5_{18}					

An instruction A-0 with a long transfer characteristic causes W to be ignored and transfers A to control in m.c.

$$q = m + T + 2$$

where A-0 is stored in minor cycle m. In this case only can W and T be unequal. This case is important when using the Automatic Instruction Modifier.

7.4 INSTRUCTION MODIFICATION USING D.O.

We shall repeat Example 1 using D.O. We shall also use DS21 and DS22 to modify the instructions

```
                   $5_{29,30}$ - 21 (d)    $5_{29}$ is 13-10
  ┌──────────→     $21_2$     -  0         $5_{30}$ is 10-13
  │       $Q_{30}$ (10        - 13)        Fetch $10_m$
  │                14         - 25         Add   TS14
  │                $21_3$     -  0
  │       $Q_{29}$ (13        - 10)        Store $10_m$
  │                28         - 22 (d)     Increase W No. of
  │                                        both instructions.
  │                n          - 13
  │                27         - 26         Count from
  │                13         -  n         initial value
  │                13         - 28         n = 32 and test
  └──────────────────/          \
                   nz            z ──→ EXIT
```

NOTES (a) 21_2 - 0 can generate only even values of quasi minor cycle and 21_3 - 0 can generate only odd minor cycles.

(b) One instruction 28-22 (d) modifies both instructions in DS21.

(c) The original instructions in 5_{29}, 5_{30} are preserved unchanged in those storage positions.

(d) The wait numbers of the instructions in $5_{29,\ 30}$ are chosen so that the first minor cycle of transfer is 10_0 and relative to Q_{29}, Q_{30}. It is a coincidence that the storage minor cycles $5_{29,\ 30}$ are the same as the quasi minor cycles.

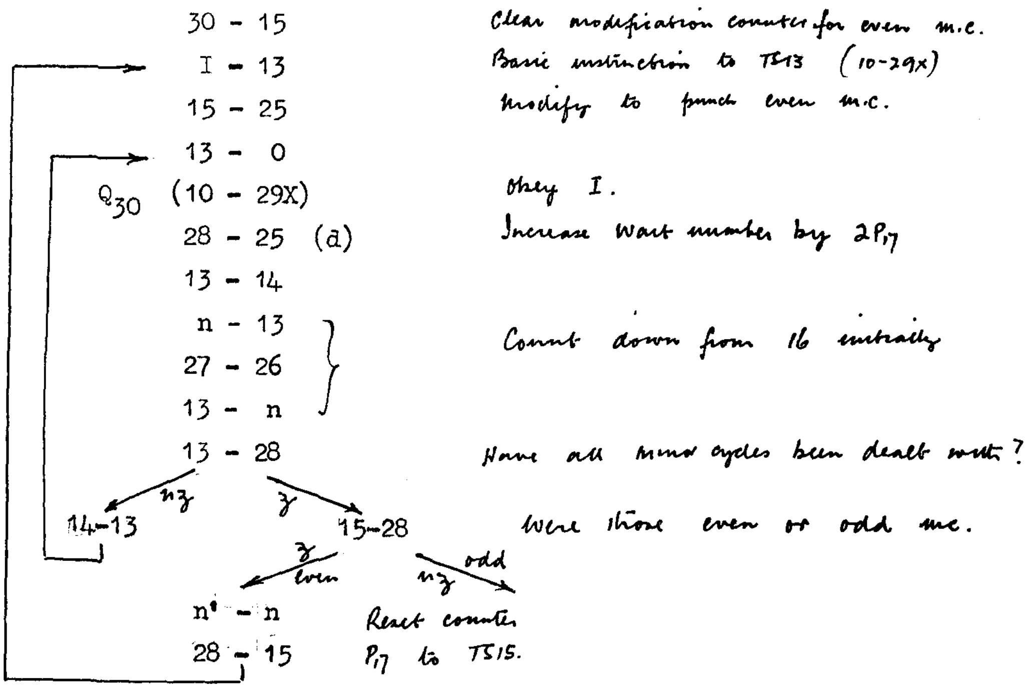
Example 2.

To read 32 rows of 3 cards to D.L. 10. The instructions are

```
                       I - 13     I is  0 - 10X  Basic instruction.
    ┌──────────→      13 -  0     I ─→ control.
    │         $Q_{30}$ (0 - 10X)  Obey I  $Q_{30}$.
    │                 28 - 25     Add 1 to W No.
 14 └ 13              13 - 14     Transfer to TS14 because TS13 is needed for
    │                  n - 13     counting, from an initial
    │                 27 - 26     value of n = 32
    │                 13 -  n
    │                 13 - 28
    └──────────────────/    \
                     nz      z ──→ EXIT
```

Example 3.

To punch 16 even minor cycles of D.L. 10 followed by 16 odd minor cycles, starting the even m.c. at 10_0



```
      30 - 15          Clear modification counter for even m.c.
  →    I - 13          Basic instruction to TS13 (10-29X)
      15 - 25          Modify to punch even m.c.
  →   13 -  0
Q30  (10 - 29X)        Obey I.
      28 - 25  (d)     Increase Wait number by 2P17
      13 - 14
       n - 13  ⎫
      27 - 26  ⎬       Count down from 16 initially
      13 -  n  ⎭
      13 - 28          Have all minor cycles been dealt with?
   nz ↙     ↘ z
 14-13          15-28  Were those even or odd mc.
           z     ↙    ↘ nz
           even          odd
       n' - n       Reset counter
      28 - 15       P17 to TS15.
```

CLASS EXERCISES.

1. Write the instructions to fetch successive words from D.L.10 (starting at 10_0) collate them with the word in TS16 and place the results in D.L.11, starting at m.c. 0. Use separate fetch and store instructions.

2. Repeat example No. 1 using constants to change the fetch instruction to a store instruction and back again to a fetch with increased wait number.

HINT. The instruction

$$N \quad 10 - 14 \quad W \quad T_1 \quad GO$$

may be changed to

$$N \quad 25 - 11 \quad W \quad T_2 \quad GO$$

by subtracting

$$0 \quad 17 - \overset{2}{\cancel{3}} \quad 0 \quad (T_1 - T_2) \quad X$$

Note if $T_1 < T_2$ the negative value is inserted at P_{26} position.

3. Write the instructions to fetch successive words from D.L.10 to TS14 starting at 10_{31} and working towards 10_0.

4. Write the instructions to fetch word pairs from D.L.12 to DS21 starting at 12_{01}.

EXERCISE SOLUTIONS.

1.

```
      16 - 14
      30 - 16
         - 13
      16 - 25
      13 -  0
     10_0- 15
         - 13
      16 - 25
      13 -  0
      25 - 11_0
      16 - 13
      28 - 25
      13 - 16
      28 - 26   (32 m.c.)
      13 - 28
  ø       | 0
      all m.c. finished.
```

2.

```
          16 - 14
             - 21_3   32 P_17
             - 13
          13 -  0
Q_30    [10_0- 15]            N,   10 - 15   0   T_1          GO
             - 26             0,   17 -  3   0   (T_1 - T_2) X
          13 -  0             ------------------------------------
Q_30    [25 - 11_0]           N_1  25 - 11,  0   T_2          GO
          28 - 23_3
         21_3- 28
            ø|         0
                       1
             - 25             0    17 -  3   1   (T_1 - T_2) X
```

N.B. The factor added before looping back to the beginning is the factor previously subtracted plus one in the wait number.

3.

```
      30 - 16
         - 13     Wait No. of 31 in basic instruction.
      16 - 26     SUBTRACT counter.
      13 -  0
    [10_31-14]
      16 - 13
      28 - 25     Increase counter by P_17.
      13 - 16
      28 - 26     (32 m.c.)   Stop at 32.
   ø  13 - 28
        |0
```

4.

```
          30 - 16
             - 13
          16 - 25   (d)       Add counter twice.
          13 -  0
          $12_{0,1}$-21
          16 - 13
          28 - 25             increase counter by $P_{17}$
          13 - 16
          28 - 26   (16 m.c.)Stop when counter = $16P_{17}$
          13 - 28
      ø      |  0
```

N.B. By adding counter twice, it saves time on the long transfer 28 -26 at the end.

8

LECTURE 8.

THE MAGNETIC DRUM STORE.

8.1 INTRODUCTION

The capacity of the DEUCE high speed store is 384 words in 12 DL s and 18 words in the Temporary Stores. 402 words is by no means adequate for the majority of programmes and secondary storage must be provided either in the form of a magnetic drum or magnetic tape. Both forms are available on DEUCE, the former as a standard machine facility and the latter as an optional extra on MK II machines.

8.2 THE MAGNETIC DRUM.

The magnetic drum is a cylinder, 6" deep and 4" in diameter. The surface is coated with magnetic oxide and DEUCE word patterns may be written around the circumference of the drum as magnetic pulse patterns. Ones are written as magnetic dipoles with the N poles pointing in one direction and zeros are written with the polarity reversed.

One circumferential band of information is known as a 'track' and it is arranged that 1024 digits are written on one track corresponding to 32 words of 32 digits each. One track of the drum is thus equivalent in storage capacity to one long delay line.

ONE TRACK = 32 words = 1 D.L.

On the surface of the drum there is room for 256 tracks of information. The tracks are so close to each other that 16 tracks occupy only $\frac{1}{4}$ inch of the drum surface.

The total storage capacity of the drum is 256 x 32 = 8192 words

To provide separate recording heads for each of the 256 tracks spaced so close together is impracticable and the reading and writing mechanism is organised on a block system.

Writing on the drum is performed by 16 writing heads rigidly held in a block. This block is free to move along a line parallel to the surface generators of the drum and can take up any one of 16 positions. Each one of the 16 heads can thus record one track in each of the 16 positions giving 256 tracks in all.

These tracks are numbered 0 to 255 and are sometines referred to by the track number in this way. More often the track is identified by the combination of the Block Position and Head used to record the track, as in the following table

Track No.	Position	Head	P/H Track No.
0	0	0	0/0
1	0	1	0/1
⋮			
14	0	14	0/14
15	0	15	0/15
16	1	0	1/0
17	1	1	1/1
⋮			
31	1	15	1/15
32	2	0	
⋮			
239	14	15	14/15
240	15	0	15/0
241	15	1	15/1
⋮			
254	15	14	15/14
255	15	15	15/15.

Sometimes programmers use A for P and B for H. Thus track 220 is A/B = 13/12 = P/H. The A/B or P/H notation is very convenient. It will have been noticed that the absolute track number is given by

Track No. (T.N.) = P x 16 + H (or A x 16 + B)

e.g. 220 = 13 x 16 + 12

Furthermore if we write the binary number representing 220 at the P_1 position of a DEUCE word it may be separated into two parts P_1 to P_4 and P_5 to P_8.

$$220 = \underset{P_{1-4}}{0011} \quad \underset{P_{5-8}}{1011}$$

P_{1-4} represents H (or B) and P_{5-8} represents P (or A). The P/H value of a track number is thus quickly read from the binary number of the track.

Reading and writing are not performed with the same head for any track. A separate block of read heads, identical to the write heads is mounted on the opposite side of the drum. This can take up positions identical with the write block and yet independent of it. There is no necessity to have the same block positions selected on the read and write heads.

8.3 ACCESS TO THE MAGNETIC DRUM.

Information is written on to the drum and read from the drum in groups of 32 words. To achieve this DL 11 is employed as a "buffer store". All information passes to and from the drum via DL 11, one delay line (= 1 track) at a time.

Data or instructions to go up to the drum are placed in DL 11 and written on a preselected track from which they may be subsequently read down into D.L.11 for transfer elsewhere in the high speed store.

8.4 MAGNETIC INSTRUCTIONS.

Two destinations only are available for magnetic operation.

D31	Selects the block position (shift)
D30	Selects the head and initiates the magnetic transfer.

Since the SOURCE and characteristic are not required in the usual context they are given special significance in conjunction with D30 and D31. With D31 a characteristic 0 specifies a block shift of the Reading Heads and Characteristic 1 specifies a block shift of the WRITING HEADS.

For D30, characteristic 0 initiates a READ, i.e. transfer from track to DL 11 while characteristic 1 initiates a WRITE transfer from DL 11 to a track on the drum.

In D31 instructions the SOURCE number specifies the P (or A) section of a track number, the so called shift position and in D30 instructions the SOURCE number specifies the H (or B) section of the track required.

Example 1.

To read track 220 to DL 8. Track 220 is 13/12.

13 - 31 (s)	Select Read Shift position 13
12 - 30 (s)	Read with head 12 causing track 13/12 to enter
11 - 8 (32 m.c.)	DL 11 from which it is transfered to DL 8.

Example 2.

To write DL 7 on track 130 Track 130 is 8/2.

7 - 11 (32 m.c.)	Fill the magnetic buffer DL
8 - 31 (1)	Set WRITE HEADS
2 - 30 (1)	WRITE up to 8/2.

It will be apparent that the order of events in READ and WRITE instructions is not complementary. In READ instructions, the order is Shift, Read, Transfer, whereas in WRITE instructions the order is either Shift, Transfer, Write, or Transfer, Shift, Write.

Example 3.

To write DL 7 to track number 130 we could use

8 - 31 (1)	Set write shift position 8
7 - 11 (32 m.c.)	Transfer DL 7 to DL 11
2 - 30 (1)	Write DL 11 to track 8/2.

8.5 MAGNETIC INTERLOCKS.

One operational disadvantage of the magnetic drum is its relatively long access time. To shift either set of heads requires 35 major cycles and to read or write one complete track takes 15 major cycles. Any magnetic operation requiring a shift and a transfer can take 35 + 15 = 50 MAJOR CYCLES. This can be more than enough to ruin an efficient programme when magnetic operations are running in parallel with other synchronous operations such as reading from cards or punching results. Under these circumstances it is necessary to adopt the technique of head-shift anticipation whenever possible. In any 'ordered' programme where all operations are predictable from the nature of the problem it is possible to shift the heads well in advance of the transfer instruction, separating the D31 and D30 operations by other instructions. In this way the total time may be reduced appreciably by utilising magnetic interlock time for computing instructions.

When writing up a track of data or instructions it is essential that writing shall not start until the correct shift position has been set up, and once a write instruction has been initiated, DL 11 must not be disturbed or another shift operated until the current writing operation has been completed. To relieve programmers of the responsibility of arranging matters, all potentially destructive operations are automatically prevented by a magnetic interlock.

Without interlocks the instructions

2_0	13 - 31 (s)
2_2	12 - 30 (s)
2_4	11 - 10 (32 m.c.)

would take just over one major cycle to complete. With interlocks however, the D30 instruction is not allowed to proceed until 35 m. sec. after the D31 instruction and the 11 - 10 transfer is not allowed to start until 15 m. sec. after the D30 instruction.

The interlocks are

(a) Within an interval of 35 m. sec. following a D31 instruction, any instruction with D30, D31, S11 or D11 will be held up until the end of the 35 m.s. interlock time.

(b) Within an interval of 15 m. sec. following a D30 instruction any instruction with D30, D31, S11 or D11 will be held up until the end of the 15 m.s. interlock time. Interlocks may be thought of as barriers, D30 & D31 are the only instructions which can put up the barriers. All instructions except other D30 & D31 instructions and S11 & D11 instructions can pass through the barrier. D30's barrier is up for 15 m. sec., D31's for 35 m. sec.

There is one exception to these rules. If the heads are in say position 5 and a 5 - 31 instruction is received by control for the appropriate set of heads, no barrier is raised as the heads are not required to move.

An instruction calling for a head shift to the position currently in use does not generate an interlock.

Some DEUCE computers have a more rational set of interlock rules, e.g. S11 can be used after D30 (l) and D30 (s) is not interlocked after D31 (l). Such interlocks result in faster operation for random access programmes where shift anticipation is not possible.

CLASS EXERCISES.

1. Write the instructions to perform the following.

   (a) Write D.L.9 to track 10/7.

   (b) Read track 0/3 to D.L. 8.

   (c) Clear track 0/0.

   (d) Write a P_{31} in minor cycle 16 of track 15/15.

2. Write instructions to read tracks 7/0 to 7/7 into D.L.'s 1 to 8. Make use of the fact that the tracks are

   (a) All in one head position.

   (b) consecutive.

to use simple instruction modification.

3. Write instructions to write D.L.'s 1 to 8 to tracks 14/14 to 15/5.

4. Write a programme to clear all tracks of the drum.

HINT. Use two counters and instruction modification.

5. Write the instructions to write up D.L.8 to track 13/12 with the track zero summed in m.c. 31.

EXERCISE SOLUTIONS.

1. (a)

```
10 - 31 (1)         Shift write heads to position 10.
 9 - 11 (32 m.c.)Copy D.L.9 into D.L.11.
 7 - 30 (1)         Write D.L.11 using head 7.
```

(b)

```
 0 - 31 (s)         Shift read heads to position 0.
 3 - 30 (s)         Read head 3 into D.L.11.
11 -  8 (32 m.c.)Copy D.L.11 into D.L.8.
```

(c)

```
 0 - 31 (1)         Shift to position 0.
30 - 11 (32 m.c.)Clear D.L.11.
 9 - 30 (1)         Write zeros on head 9.
```

(d)

```
15 - 31 (1)         Shift to position 15.
30 - 11 (32 m.c.)Clear D.L.11.
   - 11_16(P_31)    Put P_31 in 11_16.
15 - 30 (1)         Write D.L.11 on head 15.
```

2.

```
     - 21 (d)          Basic instructions to DS21.
   7 - 31 (s)          Shift heads to position 7.
21_2 - 0
   7 - 30 (s)          Read required head.
21_3 - 0
  11 -  8 (32 m.c.)Copy into required D.L.
     - 23 (d)          Modify instruction:  Subtract P_5 from 21_2
                                                     P_10 from 21_3
```

N.B. This routine overwrites ALL NIS D.L.'s. The next instruction is therefore taken from the NEW D.L.1 (i.e. Track 7/0) and the minor cycle is the same as the one previously containing -23 (d). All the instructions in the loop of this routine must be in D.L.1, otherwise some will be overwritten before the routine is completed.

3.

```
     - 21_3                              Set basic instructions.
     - 14              14P_1 + 14P_5     Set track counter.
     - 15              P_5-8             Set collate digits.
21_3 - 0
[  1 - 11 (32 m.c.)]                     Copy D.L. into D.L.11.
     - 13                                Basic instruction 0 - 31 1
  25 - 25                                Modify source for shift position.
  13 - 0
[ 13 - 31 (1) ]                          Shift heads.
  24 - 14 (4 m.c.)                       Move up track counter.
     - 13                                Basic instruction 0 - 30 1.
  25 - 25                                Modify source for head number.
  13 - 0
[ 14 - 30 1 ]                            Write using correct head.
  23 - 14 (4 m.c.)                       Move track counter down again.
  14 - 13
  27 - 25                                Increase track counter by P_1
  13 - 14
     - 26              6P_1 + 15P_5      have we done all D.L.'s?
  13 - 28
  nZ           Z
     - 22_3 + P_5                        Increase source of transfer instruction
                                         by one.
```

```
4.      30 - 21 (d)          Clear 21.
        30 - 11 (32 m.c.)    Clear D.L.11.
           - 13              Basic instruction 0 - 31 (1)
        21_2- 25             Add counter for head shift.
        13 -  0
       [0 - 31 1]            Shift heads.
           - 22_2            Add P_5.
        21_2- 13
           - 26              have we reached 17?
        13 - 28              (N.B. 16 - 31 is equivalent to 0 - 31)
        nZ        Z          FINISH.
        30 - 21_3            Clear 21_3.
           - 13              Basic instruction 0 - 30 (1)
        21_3- 25             Add counter for head number.
        13 -  0
       [ 0 - 30 1]           Write zeros on track.
           - 22_3            Add P_5.
        21_3- 13
           - 26              Have we reached 16?
     Z  13 - 28    nZ
5.      13 - 31 (1)          Shift heads.
        30 -  8_31           Clear 8_31 for sum check.
        30 - 13              Clear 13.
         8 - 26 (32 m.c.)    Subtract D.L.8 from TS13.
        13 -  8_31           Copy TS13 into 8_13.
         8 - 11 (32 m.c.)    Copy D.L.8 into D.L.11.
        12 - 30 (1)          Write on track.
```

__N.B.__ When the track is reach back from the drum, the sum is checked by:-

```
        30 - 13
        11 - 25 (32 m.c.)
        13 - 28
      Z          nZ
                FAILURE.
```

9

INPUT AND OUTPUT (α FIELD WORKING).

9.1 INTRODUCTION.

This lecture deals with the input to and output from DEUCE of DATA. It should be realised right from the beginning that DATA and INSTRUCTIONS are two completely separate entities, and require completely different treatments. Another lecture deals with the reading of instructions; this lecture deals with the reading of data and punching of results, two aspects of the programme which should be settled before a single instruction can be written. It cannot be stressed to highly that until the programmer knows what data he will be supplied with and what results he is expected to produce, he can do nothing.

9.2 INPUT AND OUTPUT MEDIUM.

The usual medium for reading data into DEUCE, and for getting results out is the standard 80 column punched card. It is the normal practice, however, to use only 32 of these columns (card columns 17-48), - one column for each digit in the binary word - and the rest are ignored by DEUCE although they may be used for card numbering and other useful information to assist in card handling off the machine. Later models of DEUCE are fitted to use 64 columns on each card, but this will be dealt with in a later lecture, and we will only refer to the simpler 32 col. system here.

9.3 BASIC REQUIREMENTS.

In order to allow the programmer as much scope as possible, it is necessary to have orders in the instruction code to start and stop the reader, start and stop the punch, read a row of a card and punch a row of the card.

The orders concerned with the input and output are therefore:-

(a) 12-24 Start cards passing through the reader. When this order has been obeyed cards will pass continuously through the reader until either it runs out of cards, or until the reader is stopped by the appropriate DEUCE instruction.

(b) 10-24 Start cards passing through the punch, in a similar manner to the reader.

(c) 9-24 Stop cards passing through either the reader or the punch.

(d) 0- A Read the word from the row of the card at present under the reading brushes into destination A.

(e) B-29 Punch the contents of source B on the row of the card at present under the punch knives.

9.4 TIMING.

As the card reader and card punch are mechanical appliances, they do not function as fast as DEUCE itself. It is therefore necessary to have a system whereby DEUCE will wait for the reader or punch, and only carry on when the right stage of reading or punching is reached. For this reason, all instructions having source 0 (for reading) or destination 29 (for punching) are made "stoppers" - i.e. they do not have a GO digit punched on them.

When DEUCE comes to such a STOPPED INSTRUCTION, it waits until it gets a signal to continue. The reader and punch are so organised as to emit a suitable signal - called a "single shot" - every time a row of a card is in position to be read or punched. Hence, it is possible to guarantee that the correct row of the card is in position before causing DEUCE to read or punch any information, but it should always be remembered that if DEUCE takes too long between such stopped instructions, it will miss the single shot completely and all subsequent rows will be read one late. The relevant timings will be given later.

9.5 SIMPLE EXAMPLES.

(a) Read the first two rows of a card into DS19$_2$, DS19$_3$.

12-24	Start the reader.	
0-19$_2$X	First row into 19$_2$	(Y row of card)
0-19$_3$X	Second row into 19$_3$	(X row of card)
9-24	Stop the reader.	

Note that both source 0 instructions are "stoppers" indicated by the cross, but that the two instructions to start and stop the reader are always "goers", to be obeyed as fast as possible.

(b) Punch the contents of the four short stores.

10-24	Start the punch.
13-29X	Punch TS13 on Y row.
14-29X	Punch TS14 on X row.
15-29X	Punch TS15 on 0 row.
16-29X	Punch TS16 on 1 row.
9-24	Stop punch.

N.B. If the reader is stopped part way through a card, it is not possible to read the subsequent rows of that card, and the next single shot will appear on the Y row of the next card when the reader is recalled. A similar rule applies to the punch.

9.6 PROGRAMMES INVOLVING READING AND PUNCHING.

It is not possible to read from and punch into cards at the same time, due to the inherent difficulties of running two machines at once with the consequent muddling up of the two sets of single shots. Therefore, before the programmer writes an instruction to start either the reader or the punch he should make certain that the other is stopped. The best way to ensure this is to always shut off the reader or punch as soon as the current operation is finished.

9.7 DECIMAL OPERATION.

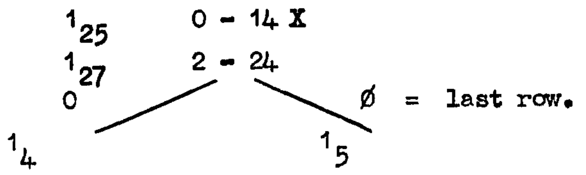
So far, we have only considered binary reading or punching, but it is the accepted practice for all data to be read in decimal if possible. Firstly, how do we punch decimal. The normal system is to only have one punching in each of the card columns, and its value is determined by the row of the card in which it is punched. For this purpose, the rows 0-9 indicate the decimal numbers 0-9, and the Y and X rows indicate positive (Y) and negative (X) signs. Thus to punch the two numbers + 432 and - 748, we would punch. Y432 X248, one punching to each card column, using a total of eight columns.

As the rows of the card come in order Y-9 to the reading station, it is possible to determine the value of each decimal digit by counting the single shots before a punching appears, and using suitable instructions to convert these to binary. However, since this task is common to all programmers, a selection of subroutines has been produced to deal with the more usual layout of decimal cards. It is therefore advised that the programmer should select a suitable read (prefix R) or punch (prefix P) subroutine from the library to perform the reading and converting to binary, or converting from binary and punching, rather than attempt to write his own. It is often possible to convert an existing subroutine for a special purpose, but attention should be paid to the timing restrictions before this is attempted.

9.8 TIL.

Since the time between rows on a card is for less than the time between the last row of one card and the first row of the next, it is desirable to have a method of ascertaining if the row now being read (or punched) is the last on a card.

For this reason, a special signal TIL is provided to indicate the last row. It appears shortly before the last row single shot, and lasts for some time after it. It is inspected by the instruction 2 - 24, which in effect discriminates on the state of the TIL signal, going non-zero for the last row only.



In the example, having read any row of a card other than the last, the TIL line is zero, and 1_4 is called. However, for the last row of a card, the TIL signal is present and the next instruction comes from 1_5.

In this way, the last row of a card can be detected and use made of the extra time available between single shots.

9.9 POINTS TO NOTE.

(a) It is not allowable to leave stoppers in instructions which do not refer to the read or punch, if either machine is called at the time the instruction is obeyed, unless it is inserted deliberately to waste a row of a card. The usual failing is for stoppers to be left in a programme so that the rows of the card get out of sequence with the programme, resulting in either programme failures or incorrect results.

(b) It is not stated on some of the decimal punch subroutines that they will not operate correctly with T.C.B. ON. A sympton resulting from T.C.B. ON is that the result is always zero. For these routines, and for several others in various categories it is assumed that T.C.B. is OFF, as is usually the case during a calculation, since most division subroutines put T.C.B. off before exit.

(c) It is often possible to write a programme that will work on one machine and not on another, due to slight differences in reader or punch speed. Remember always to write your programme to conform to the specifications and, if you are in doubt, stop the read or punch and recall it - it is easier to do this when writing a programme than it is to alter it later.

9.10 TIMING SPECIFICATION.

Time measured:-		Time in major cycles for:-	
FROM	TO	READ	PUNCH
Single shot (Y to 8 row)	Next single shot.	15	38
Single shot (9 row)	Single shot (Y row)	57	116
Single shot (9 row)	Look for TIL.	18	18
Any single shot.	Read same row.	2	-
	Punch same row.	-	4
Clear read or punch.	Wait before Using I.D. keys.	14	-
	Wait before using O.S. lights.	-	20
	Wait before recalling read or punch.	2	2
Call punch	First single shot		180

INPUT AND OUTPUT.

C9.1 INTRODUCTION.

In order to maintain a permanent record of all data fed into the computer, it is the normal practice to punch all data in decimal on cards, using all 80 columns if required. Each column of the card may then represent:-

(a) A single numerical digit (10 possibilities).

(b) A single alphabetic symbol (26 possibilities).

(c) A special symbol for use by the peripheral card equipment.

In order to represent all of these in a binary system, it is necessary to have a six-bit binary group, allowing 64 different combinations. The code used for DEUCE is based on the I.B.M. 4-zone code, to line up with tabulations etc. This code has the following basic rules:-

(1) A single punching in any of rows 0 - 9 of a single card column shall be interpreted as a number, in the range 0 - 9 corresponding to the row in which it is punched.

(2) The alphabet shall be represented by a punching in one of the Y, X or 0 rows, together with one punching in any of the rows 1 - 9, giving two holes in the same card column. The Y, X or 0 punching is referred to as the "overpunching". Of the 27 combinations given by this system, 26 are for the alphabet and the odd one (1 overpunched 0) is a FULL STOP.

C9.2 BASIC REQUIREMENTS.

For the majority of purposes, the information that is required to be fed into a computer consists of either a series of numbers, or a series of alphabetic characters, or a combination of both. The 80 column input-output machine is designed to read (or punch) standard 80 column cards with numeric or alphabetic information in any desired combination.

C9.3 GENERAL ORGANISATION.

In order to make the system as flexible as possible, the machine is designed to treat each of the 80 card columns as a separate entity, and to produce a binary character to represent each column. A six digit binary character is used, giving 64 different combinations, of which 36 are in general use - 10 for the numbers 0 - 9 and 26 for the alphabet - leaving the remaining ones for special purposes.

C9.4 CODE.

The code used to represent alphanumeric characters on the punched card is based on the I.B.M. 4-zone code. The choice of this is based on the obvious necessity to keep the system used as close as possible to existing punched card practice to enable full use to be made of existing peripheral equipment - tabulator, sorter, etc.

The code is based on the principles that:-

(a) A single punching in any of rows 0 - 9 of a card shall be interpreted as a number.

(b) A single punching in any of the rows 1 - 9, together with a single "overpunching" in any of the rows Y, X or 0 shall represent an alphabetic character. It will be noted that this allows 27 possible combination, so the combination 0, 1 is not used, following tabulater practice.

The table indicates how each letter of the alphabet is represented. For example, the letter K appears in row 2 underneath the X. Therefore the character X, 2 represents the letter K.

	Y	X	0
1	A	J	-
2	B	K	S
3	C	L	T
4	D	M	U
5	E	N	V
6	F	O	W
7	G	P	X
8	H	Q	Y
9	I	R	Z

C9.5 BINARY REPRESENTATION.

The binary representation for any character can be calculated using the following rules:-

(a) For a punching in rows 1 - 9, add the value of the row.
(b) For a Y overpunching, add 16.
(c) For an X overpunching, add 32.
(d) For an 0 overpunching, add 48.
(e) For an 0 punching on its own, add zero.
(f) For a blank column add 15.

Therefore, if a K is required, this will be punched X2 on the card, and will be interpreted as 32 + 2 (= 34) in the binary code.

The code used for input and output is identical, except for two special cases .

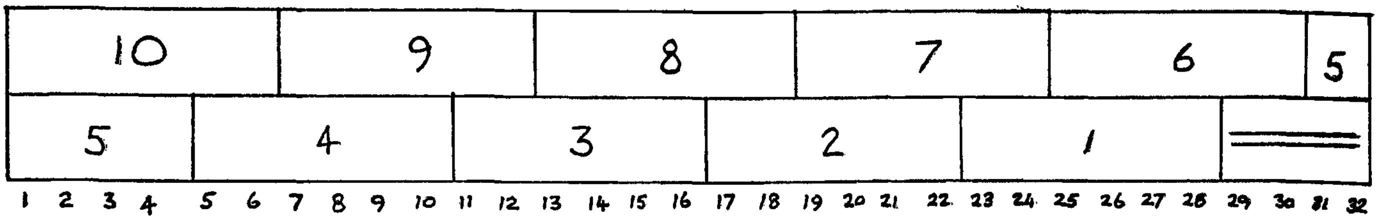
(a) Because of the difficulty of putting 15 in for a blank column, especially when several blank columns together are required, it has been decided that the binary character 63 (all ones) shall also punch a blank column. Consequently, it is not possible to obtain a character 63 from the reader, as it will automatically be changed to 15, the normal code for a blank column.

(b) Because of the two different roles performed by the 0 row (either a numeric zero, or an alphabetic over punching) the combination 48 will always be punched as zero, and it is not possible to produce the combination 48 from the reader.

This leaves 26 possible combinations as yet unused. These all require three punchings in the single row of a card, and will be of the form Y,X or 0, any one in the range 1 - 7, and 8. The binary equivalent can be calculated by adding the contribution of each hole individually following the rules given above.

C9.6 ARRANGEMENTS OF BINARY CHARACTERS.

So far, we have dealt with a single card column, and the binary character resulting from it. As we have 80 columns to deal with, each producing a 6 digit binary character, at least 15 m.c. will be required to store these. For convenience sake however, it is more simple to allow 10 card columns to be placed in each pair of minor cycles. This leads to a far more simple layout and reduces the effort involved in sorting out the characters when required.



The diagram shows how the group of 10 card columns are arranged in a pair of minor cycles (a word pair) in 6 bit binary form. As Col. 1 is the most significant on the decimal card it has been located at the most significant end (binary-wise) of the word pair.

The word pair containing the characters for cols. 11 - 20 will be located in the two minor cycles immediately before cols. 1-10 and so on, so that col. 80 appears in the top left-hand corner of the block of 16 minor cycles.

80	79	78	77	76	
75	74	73	72	71	—
70	69	68	67	66	
65	64	63	62	61	—
60	59	58	57	56	
55	54	53	52	51	—
50	49	48	47	46	
45	44	43	42	41	—
40	39	38	37	36	
35	34	33	32	31	—
30	29	28	27	26	
25	24	23	22	21	—
20	19	18	17	16	
15	14	13	12	11	—
10	9	8	7	6	
5	4	3	2	1	—

It should be noted that the most significant four digits of each word pair are not usuable. Before a card is read they will be cleared and nothing will be put into them by the reader. Conversely, the punch will completely ignore the existance of these four digits.

C9.7 <u>LOCATION OF BINARY CHARACTERS.</u>

The machine is designed to read from cards into D.L.12, or to punch from D.L.12 onto cards. As it is sometimes necessary to read two cards, one into one half of D.L.12 and the second into the other, we have to allow the programmer to dictate which part of D.L.12 shall be used. Therefore, the rule is that whenever we ask for a card to be read, we also stipulate the first minor cycle of the "read store", i.e. the minor cycle in which cols. 80-76 shall be placed. The machine then knows that minor cycle and the following 15 consecutive minor cycles are required to hold the binary characters produced from the card about to be read.

C9.8 <u>ORDER CODE - READER.</u>

The operative instruction for reading on 80 col. card into D.L.12 is of the form 12 - 24 (1) The characteristic 1 indicates that it is to be read in the 80 col. fashion .

The wait no. defines the first minor cycle of the read store for this card only, and the reader will stop automatically at the end of a single card cycle.

<u>Example 1.</u>

1_{23} 12 - 24 (1) (m.c. 0)
1_{3}

Coded as:- 1, 12 - 24 1, 7, 10.

The wait number of 7, following the normal rules governing the minor cycle of transfer defines the start of the read store in minor cycle 0. To make the flow diagram clear, it is usual to write (m.c. 0) in brackets beside the instruction to indicate where the read store starts.

Once this instruction has been obeyed the card reader will pass one card through the feed, clear any information previously stored in the 16 minor cycles of the read store, and replace it with the characters generated from the punchings on the card. All of this is done purely automatically, and the computer can proceed with other things, provided the 16 minor cycles of D.L.12 into which the characters are being assembled are not disturbed.

C9.9 ORDER CODE - PUNCH.

The instruction to command the computer to punch an 80 col. card is 10 - 24 (1). The wait number is again used to determine which 16 minor cycles of D.L.12 shall be punched but the minor cycle specified must be the first of the READ store, as the machine assumes that the 16 not being used for reading shall be the ones from which it shall punch.

This rule applies even if no reading operation is actually taking place.

Example 2.

```
                        1₂₅      10 - 24 (1)       (m.c. 0)
                        1₈
Coded as:-              1,       10 - 24, 1, 5, 13.
```

The wait No. of 5 defines m.c. 0 (which is written beside the instruction) as the fist minor cycle of the READ store, and therefore this instruction will punch the contents of D.L. 12 minor cycles 16 - 31.

As for the reader, the card feed will pass one card only and then stop automatically. The 16 minor cycles used for the PUNCH store must not be interfered with during the punching period.

C9.10 INTERLOCKS.

As it is not allowable to interfere with D.L.12 during either reading or punching, a signal is provided on the TIL line to enable the programme to check whether such operations have been completed. As soon as any 80 column reading or punching operation is called (by obeying 10 - 24 1 or 12 - 24 1) the TIL line emits a signal, until the operation is complete. As soon as the TIL line reverts to its ZERO state, the programmer is free to interfere with D.L.12 without predjudice to any card reading or punching operations.

It should be noted that in the event of any mal-operation of the card feeds such as a jammed card, the TIL signal will persist until such time as the trouble is cleared and the card is fed correctly, thus causing the programme to wait until it is safe to proceed.

Example 3.

Read a card into 12_{0-15} and transfer the result into D.L.9.

```
          12 - 24 1 (m.c. 0)      Read the card.
           2 - 24                 Wait for the TIL line to revert
  nZ             Z                to zero.
          12 - 9₀₋₁₅              Now transfer from D.L.12.
```

Following any 80 col. read or punching instruction, a TIL loop as in Example 3 above should be used before D.L.12 is again disturbed.

C9.11 COMBINED READING AND PUNCHING.

It is allowable to both read into and punch from D.L.12 at the same time, but the read and punch stores will be the two separate halves of D.L. 12. If combined operation is required, the two instructions 10 - 24 1 and 12 - 24 1 must be obeyed within a period of 32 m.c.'s but the order in which they occur has no effect on the operation. The only requirement is that the first of the two instructions shall define the start of the read store i.e. exactly the same as if only single operation is desired. When the second of the two orders calling for combined operation is obeyed, the start of the Read Store for the next card cycle has already been defined - by the first order - so the wait number of this instruction is of no significance whatsoever, as the machine will completely ignore it.

If this restriction of 32 minor cycles between obeying the two instructions is exceeded, the second of the two is liable to be completely ignored, or produce a condition that works occasionally, in a random manner.

C9.12 PLUG BOARD FACILITIES.

The 80 col. input-output machine is equipped with a plug board, which allows the relative positions of binary characters in D.L. 12 and the resulting punching (or the reading) column on the card to be varied. For the purposes of this lecture, all plugging is assumed "straight", i.e. in the form normally used.

If it is essential for a different layout to be obtained, for some reason beyond the control of the programmer, he should draw up a chart indicating what plugging is required, to reduce the chances of error. It is also recommended that he prepares a suitable test programme which checks that the plugging is correct, by reading a set of cards, punching them out again, and checking against a master pack to see that they are all in fact correct. In all cases, remember that a standard board should never be interfered with: a spare should be obtained for any non-standard arrangement.

C9.13 RESTRICTIONS.

Because of the circuit layout, it is not possible to use D29 during an 80 col. punching cycle. Also, S.O. is unreliable during an 80 col. read cycle.

It is recommended that in any section of programme where there is the slightest possibility of any 80 col. operations being performed, Source 0 and destination 29 should NOT be used.

10

LECTURE 10.

READING PROGRAMME INTO DEUCE.

10.1 INTRODUCTION.

Having written a programme of instructions, the problem of actually getting them stored inside the computer has to be considered. It should be realised from the start that instructions and data are two completely different things, and are read into the machine in different ways. The reason for this is that instructions always occur in blocks of 32 (one delay line or one track on the magnetic drum) and can be dealt with in a standard manner, using a minimum of extra information supplied by the individual programmer, whereas no two programmes want the data in the same form or the same location. Consequently systems for reading programmes into the machine in the simplest manner possible have been developed, allowing the programmer to fill what stores he likes with instructions and to have his first instruction in any location he selects, but from that point he is left to read his own data in the peculiar form he requires.

This lecture is only concerned with bringing the instructions into their correct location inside the machine, and sending the first one on the flow diagram into control to be obeyed.

10.2 WHERE SHALL PROGRAMMES BE PUT

Before deciding where to store his programme, the programmer should decide what type of programme he is writing:

(a) A small programme which may be entirely contained in the high speed store.

(b) A larger programme in which the drum will be required to store instructions until they are required for use.

These two types will now be dealt with in greater detail.

10.3 PROGRAMME IN THE HIGH SPEED STORE ONLY.

Whenever instructions are punched, they are punched in what is called 'triad' formation. That is, the 32 instructions for any one delay line are punched on three cards, leaving the first four rows at the top of the first of the three cards blank (at present). It should be remembered that once these three cards are punched, the order in which they appear defines the minor cycles in which the instructions will eventually be stored, so they should never under any circumstances be disturbed in their order amongst themselves, although the position of the three cards relative to other 'triads' may be changed. The first four rows of each triad are reserved for a prescribed formula to instruct the computer where they are to be stored. This formula is different for NIS and non - NIS delay lines, and is of the form:

DL 9-12 (non NIS)	Y row	1, 0-B,	(1)	29, 28 x
	X row	B, 0-D,		30, 31 x
	0 row	B, 0-B,		27, 28 x
	1 row	1, 0-D,		30, 31 x

For these, D is the number of the delay line to be filled, B is the number of a "buffer Delay line" (in the range 2-8) used in the filling sequence (B is usually taken as 7).

N.B. The buffer delay line B will be left filled with superfluous instructions at the end, unless it is subsequently filled with a triad of instructions.

DL 1-8 (NIS)	Y row	BLANK		
	X row	D, 0-D,	(1)	26, 25 x
	0 row	D, 0-D,		30, 31 x
	1 row	M, 0-D,		30(T-1)x

Again, D is the number of the delay line to be filled.

(N.B. for NIS D where D = 8, punch ZERO). If D is the lowest numbered delay line to be filled, punch M and (T-1) when the first instruction of the programme is in M_T. i.e. for first instruction in 2_{19}, M = 2, (T-1) = 18: if 7_0, M = 7, (T-1) = 31.

If D is NOT the lowest numbered delay line, punch M = 1, (T-1) = 31.

When the triads for all the delay lines that are required to be filled are punched, they should be arranged with the triad for the HIGHEST NUMBERED D.L. FIRST, followed by other triads in descending order, with the LOWEST NUMBERED D.L. LAST. If this is not observed, the programme WILL NOT WORK.

An "Initial Pack" (see 10.5) should be placed before the first triad, and the pack is then ready to be fed into the computer.

10.4 PROGRAMMES REQUIRING THE DRUM.

If it is necessary, because of the number of instructions required for the programme, to store parts of it on the magnetic drum until required, it is always advisable to store ALL the programme on the drum as it is read into the machine and then fetch it back into the high speed store as required.

A programme has been prepared (ZE49T) to perform all necessary initial functions, read triads of instructions onto the track defined on the top row of the triad, and finally, when all triads have been read, to bring those tracks defined on a parameter card down into any specified delay lines and enter at any required point.

All that is required of the programmer is that he shall label each triad with the track number on which it is to be written (track number punched xP_1 on the Y row of first card of triad,) and punch a parameter card to stipulate what tracks go in which delay line and where the programme starts.

He assembles his pack in the following order:

(a) A copy of ZE49T.

(b) His triads for the drum labelled correctly.

(c) His parameter card.

On reading this pack into the machine the drum will be filled, the required delay lines filled from the drum, and the first instruction of the programme sent to control to be obeyed.

10.5 INITIAL PACK.

All programmes for DEUCE must start with an initial pack, in order to set the machine into a standard state before any programme starts.

The programme ZE49T starts off with three cards which put the machine into a suitable initial state, and these three cards may be used in front of any programme. If delay lines only are to be filled, these three cards should precede the triads of instruction. If ZE49T is used to fill the drum, they will automatically be included.

The initial pack ensures that we have an even minor cycle for m.c. 0, that the triggers TCA and TCB are off, that all tracks of the magnetic drum are cleared and that track 15/15 contains a "clock track". The clock track consists of a P_{31} in minor cycle 16 of track 15/15, and is used for certain programming aids like POST MORTEM and RESTORE CONTROL. P_{31} is selected as that is not used in instructions, so the track may be also used for instructions, in which case a P_{31} should be punched in m.c. 16 of the relevant triad. Track 15/15 should never be used for data storage.

10.6 POINTS TO NOTE.

(a) The reader should always be switched off by the first instruction of the programme unless data is to be read immediately. The safe rule is that if you are in doubt SWITCH IT OFF N.B. If ZB49T is used to read to the drum, the reader will automatically be switched off.

(b) Whilst a programme is being read, the instruction staticiser lights on the control panel should maintain a steady cyclic rhythm. Any error in the filling instructions or in the order of the cards becomes immediately apparent by the breaking up of the rhythm, and indicates the approximate point at which the trouble started.

If the lights are not observed, the pack will probably all pass through the reader (as it needs a 9-24 instruction to stop it) but the programme will not be stored correctly and the wrong instructions will enter control. By this time, the evidence of the source of trouble is gone, and all the programmer knows is that "something is wrong", leading to a complete waste of time.

If on the other hand, the 'STOP' key on the reader is operated as soon as signs of trouble are noted, the region in which it occured is localised leading to an easier detection and correction. Remember, if the reader is stopped during reading and it is found to be in fact operating correctly, the RUN IN key will restart it without losing anything.

11

LECTURE 11.

FETCH AND STORE ROUTINES

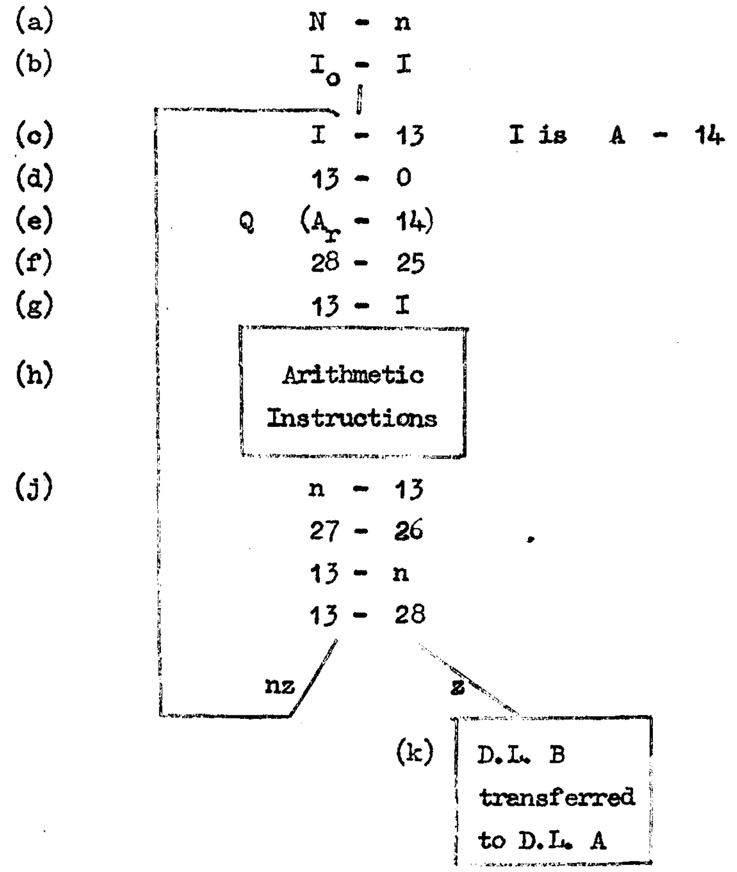
11.1 INTRODUCTION.

The DEUCE store is sometimes referred to as a multi-level store because of its discontinuous nature. Earlier lectures have shown that the mercury store consists of single word lines, double and quadriple word lines and long delay lines. Data must be stored in the long delay lines in blocks of 32 words. On the drum there are two levels of reference to a track, a head number and a shift number. In any practical problem it is unlikely that data will be provided in convenient blocks of 32 and some means must be provided to make the physically discontinuous storage system of DEUCE appear, from a programming point of view, as a continuous store. Any sequence of instructions which extract words from one or more long delay lines successively is known as a FETCH routine and sequences of instructions for placing words in successive storage locations of one or more D.L.s is known as a STORE routine. Similarly there exist sequences of instructions for storing data in blocks of 32 words on successive tracks of the drum and bringing successive tracks of the drum into the high speed store. Such sequences of instructions are known as magnetic Fetch and Store routines.

11.2 CONTINUOUS FETCH FROM ONE DELAY LINE.

If data is stored in one delay line and it is required to bring successive words from the delay line into the temporary store for arithmetic operations the following sequence of instructions may be used.

To fetch N items successively from D.L. A. (N ≤ 32)



```
(a)          N - n
(b)          $I_o$ - I
(c)          I - 13      I is A - 14
(d)          13 - 0
(e)     Q  ($A_r$ - 14)
(f)          28 - 25
(g)          13 - I
(h)     Arithmetic
        Instructions
(j)          n - 13
             27 - 26
             13 - n
             13 - 28
        nz         z
                  (k) D.L. B
                      transferred
                      to D.L. A
```

Notes.

(a) Set a working counter n at the initial value N.

(b) Set a working instruction I at an initial value I_o.

(c) Transfer I to TS 13.

(d) Obey I using Destination 0.

(e) Remember to code I from the Quasi minor cycle, and not from the minor cycle in which I is originally stored.

(f) Add P_{17} to the wait number of the instruction.

(g) Replace the modified instruction in the working storage location.

(h) A block of processing instructions using the word which has been fetched from D.L.A

(j) Count off the words. Reduce the working counter by one and replace it in its working store. Test if the counter is reduced to zero.

(k) If the same sequence of instructions is required again on other data, the data must be transferred to D.L. A and the working counter and working instruction set at the initial values.

When single words are brought successively from a delay line they must be transferred to a TS since these are the only stores which can cover all minor cycles. An examination of the above sequence shows that two counting operations are taking place, in the wait number of the instruction which undergoes modification and separately in the counter n. It should also be noted that it is necessary to replace the working instruction and the working counter by copies of the original instruction I_o and the original counter N each time the loop is entered afresh.

These objections can be partially overcome by an alternative scheme, which will be illustrated as a STORE routine.

To Store N successive words in D.L. A (N ≤ 32)

```
(a)                30 - 15
(b)                 I - 13       I is (14 - A)
(c)                15 - 25
(d)         (      13 - 0
            (  Q  (14 - A_r)
            (      15 - 13
(e)         (      28 - 25
            (      13 - 15
            (    NP_17 - 14
(f)         (      26 - 28
                 nz              z
(g)        [ B_1 ]       (g)  [ B_2 ]
```

Notes.

(a) A counter is kept at P_{17} position in TS 15. This instruction sets the initial value at zero.

(b) Basic instruction sent to TS 13.

(c) The current value of the counter is added as WAIT number to the instructi

(d) The modified instruction is obeyed using D.O.

(e) The counter is increased by P_{17} ready for the next operation.

(f) NP_{17} is compared with the current value of the counter to detect when all words have been stored in D.L. A.

(g) After the counter test the blocks B_1 and B_2 represent sequences of instructions for other work in the programme.

This sequence of instructions has the advantage that only one counting operation is used and the basic instruction I is never altered in the position where it is stored. This instruction is brought from its storage location, obeyed from TS13 as a modified instruction but the modified copy is not re-stored. Instead, the counter in TS15 is used as the modifying parameter and this is increased in successive trips round the loop.

The TS storage locations are so useful that it may be necessary to keep the counter in a long delay line storage location. When this is necessary the instructions would first appear on an outline programme as

```
        30 - n

         I - 13
         n - 25
        13 - 0
   Q   (14 - Ar)
         n - 26
        28 - 25
        13 - n
      NP17 - 26
        13 - 28

   nz               z
```

11.3 <u>TO FETCH ITEMS SUCCESSIVELY THROUGH SEVERAL DELAY LINES.</u>

When data is stored in a continuous manner through a succession of delay lines some device must be employed to change the SOURCE number of the FETCH instruction after every 32 operations. The following sequence of instructions is usually employed.

```
(a)              30 - 15

(b)               I - 13        I is  (A - 16)
(c)              15 - 14
(d)              14 - 25
(e)              23 - 14 (21 m.c.)
(f)              24 - 14 ( 4 m.c.)
(g)              14 - 25
                 13 - 0
(h)          Q   (S - 16)

         (       15 - 13
(j)      {       28 - 25
         (       13 - 15

         (     NP17 - 14
(k)      {       26 - 28
         (
               nz           z

          Repeat          Loop repetitions
                          complete.
```

The clue to this routine lies in the use made of digits P_{22} to P_{25} in the counter held in TS15. When the counter, originally zero, is increased by 32 a digit appears in P_{22} position, (counting is taking place at P_{17}). This is the indication that the SOURCE number should be increased by one. To extract the digits at P_{22} position, the counter is transferred to TS14, shifted down 21 places leaving only digits which were present at or above P_{22}. These digits are next shifted up to the P_5 position (the SOURCE position in an instruction word) and added to the instruction before it is obeyed.

Notes on the routine.

(a) Set the initial value of the counter modifier at OP_{17}.

(b) Basic instruction transferred to TS13.

(c) Counter transferred to TS14.

(d) Counter added to instruction in P_{17} position.

(e) Remove all digits below P_{22}.

(f) Shift up remaining digits to P_5 position.

(g) Add remaining digits to SOURCE position of instruction.

(h) Obey modified instruction.

(j) Increase counter in TS15.

(k) Test if all items dealt with.

11.4 TO STORE SUCCESSIVE ITEMS THROUGH SEVERAL DELAY LINES.

The fetch routine of 11.2 is simply modified to change it to a STORE routine. This time the DESTINATION number of the modified instruction must be increased after every 32 operations. The instructions are

```
                 30 - 15
     ┌─────────────┘
     ↑            I - 13       I is  (16 - A)
                 15 - 14
                 14 - 25
                 23 - 14 (21 m.c.)
                 24 - 14 ( 9 m.c.)
                 14 - 25
                 13 - 0
            Q   (16 - D)
                 15 - 13
                 28 - 25
                 13 - 15
              NP17 - 14
                 26 - 28
                /      \
             nz          z

        Repeat          Loop repetitions
                        complete.
```

The only difference is seen to be in the length of transfer of the 24 - 14 instruction. In the FETCH routine this shifts the contents of TS14 up to the SOURCE position by 24 - 14 (4 m.c.) whereas the STORE routine shifts the contents of TS14 up to the DESTINATION position by 24 - 14 (9 m.c.). These two sequences for Fetch and Store are combined in DEUCE Subroutine B01.

11.5 TO FETCH SUCCESSIVE TRACKS FROM THE MAGNETIC DRUM STARTING AT TRACK T_0.

It was pointed out in the lecture on the magnetic drum that a number TP_1 representing a track number on the drum may be considered as $AP_5 + BP_1$ where A/B is the alternative designation of the track.

e.g. $T = 189$ or A/B = 11/13

$$TP_1 = \underbrace{1011}_{B} \mid \underbrace{1101}_{A}$$

All magnetic fetch and store routines make use of this in modifying two instructions, a D31 to shift the heads and a D30 to read or write.

The instructions are:-

```
(a)              TN  -  14        TN is $TP_1$
(b)           $I_1$  -  13        $I_1$ is (0 - 31)
(c)       $P_{5-8}$  -  15
(d)              25  -  25
(e)     {        13  -  0
        {   Q ($A^+$ -  31)
(f)              24  -  14 (4 m.c.)
(g)           $I_2$  -  13        $I_2$ is (0 - 30)
(h)              25  -  25
(j)     {        13  -  0
        {   Q ($B^+$ -  30)
(k)              23  -  14 ( 4 m.c.)
        {        14  -  13
        {        27  -  25
        {        14  -  TN        TN now (T + 1) $P_1$
```

Notes

(a) Track number sent to TS14.

(b) Basic shift instruction to TS13.

(c) Collate digits in SOURCE position to TS15.

(d) Add AP_5 to Basic instruction.

(e) Obey modified shift instruction.

(f) Shift up Track No. to P_5 position.

(g) Basic Read instruction to TS13.

(h) Add BP_5 to Basic instruction.

(j) Obey modified instruction.

(k) Shift down Track number to P_1 position again.

(l) Increase track number and store it away.

11.6 A COMBINED READ - WRITE MAGNETIC FETCH AND STORE ROUTINE.

The sequence described in 11.4 can be used for either Read or Write depending on the characteristic of the magnetic instructions. A routine in which the characteristic is introduced separately permitting the same basic instructions for Read and Write is now given. This routine has two entries for the separate functions.

Track number TP_1 is in TS14.

READ $\qquad\qquad\qquad\qquad$ WRITE

P_{5-8} - 15 $\qquad\qquad\qquad\qquad$ $(P_{5-8} + P_{15})$ - 15

I_1 - 13 $\qquad$ I_1 is 17 - 30

15 - 25

25 - 25

13 - 0

Q $\quad$ (A^+ - 31)

24 - 14 (4 m.c.)

I_2 - 13 $\qquad$ I_2 is 17 - 29

15 - 25

25 - 25

13 - 0

Q $\quad$ (B^+ - 30)

23 - 14 (4 m.c.)

14 - 13

27 - 25

13 - TN

The secret of this sequence lies in the instructions 15 - 25. Through the Read entry this converts basic instructions 17 - 30 and 17 - 29 to 0 - 31 and 0 - 30 respectively. Through the WRITE entry where a P_{15} is now present the instructions are converted to 0 - 31 (1) and 0 - 30 (1).

The instructions

15 - 25

25 - 25

13 - 0

Q $\quad$ ($\qquad$)

occur twice in the routine and the same instructions are used on both occasions.

11.7 TO FETCH SUCCESSIVE WORDS FROM THE MAGNETIC DRUM.

The drum may be considered as a continuous word store with word addresses ranging from 0 (for the word in m.c. 0 of track 0/0) to 8191 (for the word in m.c. 31 of track 15/15). Word addresses stored at P_{17} position may be used to modify instructions and fetch successive words from the drum.

The sequence of instructions is

```
(a)           WN  -  14          (W.N WORD NUMBER x P17)
              14  -  13
(b)           28  -  25
              13  -  WN
(c)           I1  -  13          I1 is  (11 - 16)
          P17-21  -  15
(d)           25  -  25
              13  -  0
         Q30 (11  -  16)
(e)           25  -  14          (k)  PREPARE ENTRY
              26  -  28
                        W = 31
              nz      z
       W ≠ 31         (f)  WN  -  14
                      (g)  23  -  14 (21 m.c.)
                           I2  -  13          I2 is  (0 - 31)
                      (h) P5-8 -  15
                           25  -  25
                           13  -  0
                         Q(0   -  31 s)
                           I3  -  13          I3 is  (0 - 30)
                      (j)  24  -  14 ( 4 m.c.)
                           25  -  25
                           13  -  0
                         Q(0   -  30 s)
```

Notes

(a) WORD number x P_{17} to TS14.

(b) Increase WORD number and re-store.

(c) Delay Line fetch instruction to TS13.

(d) Add wait number and obey to fetch word to TS16.

(e) Test if 32 words fetched and if so:-

(f) WORD number (updated) to TS14.

(g) Shift AP_{26} + BP_{22} to AP_5 + BP_1 position.

(h) Modify shift instruction and obey.

(j) Modify Read instruction and obey.

(k) A track must be down in the high speed store before a word can be fetched. By entering at the point indicated on the first occasion the correct first track is brought down.

11.8 TO STORE SUCCESSIVE WORDS ON THE MAGNETIC DRUM.

Slight modifications to the Fetch routine may be made to change it to a Store sequence.

The instructions are

```
          WN  -  14
          14  -  13
          28  -  25
          13  -  WN
          I1  -  13         I1 is  (16 - 10)
      P17-21  -  15
          25  -  25
          13  -  0
     Q30 (16  -  10)
          25  -  14
          26  -  28
                  W = 31
          nz     z
  W ≠ 31        (a)   10  -  11 (32 m.c.)
                      23  -  14 (21 m.c.)
                      I2  -  13          I2 is  (0 - 31)1
                    P5-8  -  15
                      25  -  25
                      13  -  0
                    Q(A   -  31 1)
                      24  -  14 ( 4 m.c.)
                      I3  -  13
                      25  -  25
                      13  -  0
                    Q(B   -  30 1)
                (b) 10_31 -  28

                       nz        z

          (c)  30 -  10 (32 m.c.)

          n   -  13
          27  -  26
 (d)      13  -  n
          13  -  28

          nz     z

                (e)  10_0 -  28

                           nz      WN  -  14
                    Z
                                            All items
                                            written up.
```

Notes

(a) The Word number in TS14 is the correct one (see fetch routine) and the instruction here transfers DL 10 to DL 11.

(b), (c), (d) and (e) are concerned with the logic which decides whether it is necessary to write up the last track. If 10_{31} is non zero a complete track has just been written up, if 10_{31} is zero the last track has just been written. The counting sequence (d) decides when all words have been written in DL 10. If DL 10 is only partially filled this incomplete track must be written up on the drum. In the event of the number of words being a multiple of 32 the last track will have been written up by the normal method and the instruction 10_0 - 28 prevents an extra track of zeros going up to the drum.

12

SUBROUTINES.

12.1 INTRODUCTION.

DEUCE or any other computer has a strictly limited number of different instructions which it can perform, and its virtue lies (i) in the speed with which it can perform this limited repertoire of instructions and (ii) that its range of instructions is so fundamental that it is always possible to break down any well defined requirement into a sequence of simple DEUCE instructions which perform the same operation. For instance the multiplier on DEUCE only gives the correct product if the two numbers being multiplied together are positive. If they may be negative, not one instruction (0 - 24) but 8 DEUCE instructions are necessary to produce a correctly signed product. Similarly, to calculate a logarithm takes about 40 DEUCE instructions.

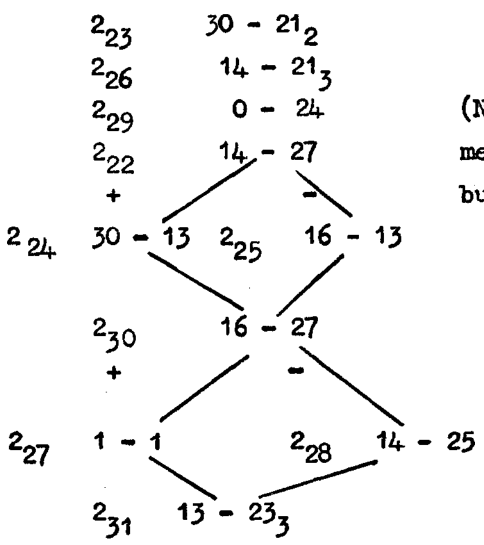
If a programmer wishes to calculate signed products or logarithms over and over again in the course of his programme, the fact that he has to use 8 or 40 DEUCE instructions each time would use up far too much instruction space, and so he uses a SUBROUTINE, which is a device which enables him to have the necessary sequence of instructions only once in his programme, and to jump into that sequence of instructions wherever he needs, and to return to his main programme, at the appropriate point, when the sequence has been obeyed.

Firstly then a SUBROUTINE saves storage space. It does much more, for a large library of proved subroutines exists and in using them the programmer knows that he is using a sequence of DEUCE instructions which is fully tested and probably faster and more economical in the use of storage space than he would make it himself.

Since a DEUCE instruction usually specifies exactly its successor (by means of NIS and T in the instruction word) it is clear that some 'trick' will be necessary in using a subroutine if the next instruction after the subroutine is to vary with each entry to the subroutine. The 'trick' is as follows:- each subroutine has associated with it a store (usually a short D.L.) in which the programmer is to place the instruction which he wishes to have obeyed when the subroutine is finished. This instruction is called the link. The subroutine itself is so arranged (i) to transfer the link to a fixed storage space, usually 1_{30} and (ii) to go to that space, 1_{30} on completion of the instructions in the subroutine. In this way the subroutine can be obeyed over and over again, jumping back to the main programme each time at a point defined by the link. Let us now look at subroutines in more detail, outlining the rules which must be followed in using them.

12.2 OPEN SUBROUTINES.

An open subroutine is usually defined as any sequence of instructions used for some specific operation. For instance, the following sequence which multiplies two signed integers in TS14 and TS16 and produces a signed product in DS21 is, as given here, an open subroutine. (We shall examine later the reasons for our particular choice of coding):



(N.B. This is not the most efficient method of performing this operation but serves as a simple illustration).

Because each inst~~u~~ction in DEUCE specifies its successor uniquely (by virtue of the timing number), it would appear that the above sequence would have to be recoded if it were to be used a second time in the programme. In fact, this difficulty is overcome by the following simple device.

12.3 CLOSED SUBROUTINES.

Suppose we add the instruction $15 - 1_{30}$ to the beginning of the sequence and call 1_{30} at the end thus:

2_{21}	$15 - 1_{30}$
2_{23}	$30 - 21_2$
	⋮
2_{31}	$13 - 23_3$
1_{30}	

Now each time the sequence is required we have only to place a new instruction in TS15 and call in the subroutine. Since the new instruction will be obeyed each time at the end of the sequence from 1_{30}, we have a completely free choice of path. The instruction planted in TS15 is called the LINK and is the first instruction following the subroutine.

The subroutine is now considered to be 'closed' (as a rule, the term 'subroutine' implies that it is 'closed').

An example of a section of programme using this subroutine is as follows:

	-
	-
	-
	$L_1 - 15$
2_{21}	$\boxed{\text{MULT}}$
L_1	$21 - 10_{12,13}$
N_T	-
	-
	-
	-
	$L_2 - 15$
2_{21}	$\boxed{\text{MULT}}$
L_2	$21 - 10_{8,9}$
	-
	-
	-
	-
	$L_3 - 15$
	$\boxed{\text{MULT}}$
L_3	$21 - 10_{14,15}$
	-
	-
	-

N.B. L_1, L_2, L_3 merely denote links 1, 2, and 3 and do not refer to m.c.'s of a D.L.

12.4 CODING THE LINK.

It is important to note that the LINK is not coded in the normal manner. Since the planted instruction is actually obeyed from 1_{30} it must be coded as though it had been there all along, i.e. coded 'quasi 30' (The reader will recall the case of 13 - 0 dealt with earlier, wherein the instruction is obeyed quasi 30). Thus, L_1 above would be coded

N, 21 - 10, d, 12, T where the next instruction is N_T. The LINK is usually written on the flow diagram as follows:

$L_1 - 15$
MULT
1_{30} $(L_1\ \ 21 - 10_{12,\ 13})$

12.5 HIGHER ORDER SUBROUTINES.

The simplest type of subroutine is one which contains no 'sub-subroutines' within its sequence of instructions. For DEUCE, such a sequence is called a 'zero order' subroutine. Zero order subroutines make use of the instruction I - 0 (where I can be any store at all). The link is obeyed quasi 30 (i.e. coded as if it were stored in m.c. 30) but, unlike the example given above, specifies no particular delay line. For example, the subroutine might put its LINK into 19_2.

$L - 19_2$
⋮
$19_2 - 0$
Q_{30} $(L - \ )$

A first order subroutine may or may not contain a zero order subroutine within it. They always obey this link from 1_{30} (the example given in the section on coding the link is a first order subroutine). A subroutine which contains a first order subroutine within it is called a second order subroutine. It will obey its link from 1_{31}. For example:

$13 - 1_{31}$
-
-
$L - 13$
$\boxed{\text{MULT}}$ First order subroutine
L -
1_{31}

Note that the first order subroutine can still be used as a subroutine in its own right, i.e. independent of the second order routine. This is really the only reason for going to second order, otherwise we could use the MULT sequence as an open subroutine.

Higher order subroutines use successive stores for their links, i.e. third order in 1_0, fourth order in 1_1, etc. (obeyed quasi 0, 1 etc.), but such orders are not often used. The choice of 1_{30}, 1_{31}, 1_0 etc. for storage of links is arbitrary and is largely in the interests of standardisation.

The order of subroutines is dealt with in more detail in DEUCE News 26 (NS y 103) paragraph 1.3.

12.6 SUBROUTINE PARAMETERS.

Particular subroutines can often be generalised to fit a wider number of cases by using a parameter instead of a constant in the appropriate places. For example, if we insert an instruction 21 - 22 (1) after 2_{31} in the MULT routine we can make the answer appear to a given number of binary places (which depends on the length of transfer of the 21 - 22 instruction). Some library MULT routines allow the programmer to specify the length of transfer (by punching in the appropriate wait number). In these cases 21 - 22 is a 'parametric' instruction and the wait number is a 'parameter'.

If the parameters vary in a programme we must take care to replace them during the programme and to restore the original values in the case of Loop Calculations. This might be done as follows. Suppose the parameter is now in 3_{12}. A copy of this is placed in 4_{12}, say, and a variant possibly in 5_{12}. Then it changes the parameter, the transfer

$$5_{12} - 3_{12}$$

is used, and to restore the original

$$4_{12} - 3_{12}$$

is used. (Remember that the entire word is replaced, although the parameter may be only a part of this word).

It is advisable to mention here that library subroutines are, in general, available in all D.L.'s. For example, a 3 D.L. subroutine will be coded in D.L.'s 2, 3, 4; 3, 4, 5; 4, 5,6,; 5, 6, 7; and 6, 7, 8. The first D.L. of each set is known as the position of the subroutine.

12.7 CODING OF SUBROUTINES.

If a subroutine is going to be obeyed many times in a programme it is worth taking considerable care in coding it to be as fast as possible. The MULT example is deficient in this respect since there is a big gap between 2_{31} and 1_{30}. (Note that between 2_{29} and 2_{31} two m.s.'s must elapse). It is also useful to code subroutines with the instructions packed neatly together. In the same example the instructions have been packed into $2_{21} - 31$ (Packing the instructions is important in library subroutines but not so necessary in private subroutines). Coding for speed and space are to some extent contradictory and a compromise usually has to be made.

12.8 SUBROUTINE REPORTS.

Subroutine reports are written in a more or less standard layout as can be seem by inspection. There are however, one or two points which require mention.

(i) The order of the subroutine is always given under the sub-heading "Description".

(ii) The subheading "Uses" lists the lower order subroutines required. In general these are not included in the punched cards, nor are the parameters listed. It is in fact a good plan to refer to the reports of the lower order subroutines used to make sure that parameters are not forgotten, that any special rules are being obeyed, etc. "Uses" may also include 13 - 0 in 1_{28}. This must also be provided by the programmer.

(iii) A good subroutine will have built in "failure" instructions i.e. if the subroutine is given data outside the range for which it will work then it obeys or stops at specified instructions. Many subroutines describe the action to be taken when this happens, e.g. subroutines that read decimal cards will usually read the card again.

(iv) The "Stores Used" section lists the short stores used by the subroutine. (It sometimes also gives minor cycles of D.L.'s used as temporary stores).

Example.

Stores Used.	13	14	15	19_2	21	11
Contents at Entry.	a	LINK	b	-	-	-
Contents at Exit.	-	-	b	-	c	-

This means that TS13, 14, 15, DS19_2, DS21 and D.L.11 are used by the subroutine. Before entering, a, b and the link must be planted in TS13, 15 and 14 respectively and the contents of 19_2, 21_2 and 21_3 are irrelevant (they do not have to be zero); and after leaving the subroutine DS21 contains the result (double length), TS15 still contains b, and TS13, 14, DS19_2 and 21 contain nothing of importance. (They are not zero).

(v) The section "Occupies" consists of a list of the D.L.'s and minor cycles punched in the cards. It does not, in general, include the D.L.'s used by the lower order subroutines nor, sometimes, the D.L.'s used as temporary stores (These may come under "Stores Used").

(vi) The "Entry" is the location of the first instruction in the subroutine. In our "MULT" example this is 2_{21}.

(vii) "Parameters" have already been dealt with. Probably more mistakes in using subroutines arise through incorrectly punched parameters than through any other cause.

(viii)A most important point about reports is that they always refer to the copy in position 2. This must therefore be taken into account when the subroutine is being used in any other position. This applies particularly to the sections "Parameters" and "Entry".

12.9 THE NUMBERING SYSTEM.

Each subroutine has four types of number associated with it.

(i) A serial number.

(ii) A report number (of the form NS t xxx or K/AA t xxx)

(iii) Figure sheets numbers (each flow diagram or coding sheet has a number of the form S6/xxxxx).

(iv) A code number.

The first three are not usually the concern of the programmer who generally refers to subroutines by the code number. This consists of four parts: a category letter, a serial number within that category, a modfiying letter if the subroutine does not deal with single length binary numbers and a stroke number if the subroutine is a modification of an existing subroutine.

Example.

(a) D01 is the first division subroutine.

(b) P13F/4 is the thirteenth punch subroutine, it works on Floating numbers; and it is the fourth modification of P13F. The code number for programmes is similar except that there are two category letters at the front.

A full list of categories and modifying letters can be seen in the latest list of subroutines and programmes.

12.10 THE SUBROUTINES IN THE LIBRARY.

It is rather difficult to try and pick out which subroutines are important since importance depends largely on the type of programme being written. The programmer is advised to consult the latest list of subroutines to see just what is available. This list will also tell him which subroutines have been superseded and will indicate if there is another subroutine which may be more suitable.

At the risk of diverting attention from other subroutines which may be equally important, the following are considered to be worth noting:-

A13F-A16F:	Floating Point Arithmetic.
B08/1 & B14:	For Fetching Programme from the Drum.
D05 & D20M:	Division.
M04/1:	Multiplication.
P & R	Subroutines for Reading Decimal Cards. Most card layouts can be read or punched but it is a good plan to consult the tabulator operator before deciding on a particular layout.

12.11 NEW SUBROUTINES (AND PROGRAMMES).

When subroutines (and programmes) are submitted for publication in the library it is a great help if these are written up in the standard form. This is not difficult in the case of subroutines it is mainly a question of checking, from DEUCE News 26, paragraph 1.4, whether all the information has been included. Subroutines should naturally be thoroughly tested, especially with limiting values. (Zero is a value which commonly upsets a subroutine). In the case of programmes it is more difficult to specify a standard layout since they differ so widely. Fairly good examples are FP08, LE06, ZP34T (Nos. 367, 352 and 314).

Further notes on this subject can be found in DEUCE News 26, paragraph 1.5 and 27, paragraph 10.

13

LECTURE 13.

ANCILLARY EQUIPMENT.

13.1 PUNCHED CARDS.

Whilst cards provide only a means to an end, too much emphasis cannot be placed on correct care and handling. Many hours of computing time may be lost and much anxiety caused, by failure to observe the few simple rules described here. It should be remembered that computer time wasted through careless card handling is charged for.

Generally, computer time is at a premium and it is most frustrating when allocated a few minutes programme testing time, to find that the programme cannot be read into the machine because of mis-shaped cards. This is generally because cards have been:-

(a) Left on a desk or window ledge in direct sunlight.

(b) Carried around in a pocket.

(c) Exposed to damp.

(d) Stored near radiators.

Wherever possible cards should be stored in a standard tray and supported by a spring type card support.

It is permissible to use rubber bands to hold together small quantities of cards: the best size for this purpose is made of 1/16" dia. rubber, the band itself having a diameter of 2½". These should not be twisted several times around the cards but used singly and kept away from the centre section. The centre ½" of the card is most critical since this section has about 3th" tolerance when passing through most machines,and the slightest indentation in this section will cause misfeeding.

It should be stated at this point that writing on the top edge of a pack of cards produces bad indentations.

Packs of cards should always have the programmers name and title of the job written on the face of the top card. All cards within a batch should be serially numbered, preferably by punching, in order that they may be sorted mechanically into sequence, in the event of them getting out of order. (A method for mechanically numbering a pack of cards will be described later).

Where trays are used for storing cards, the trays should be labelled with full details of contents. If several packs of cards are filed in one tray then these should be divided by tagged guide cards, with identification data written on the tag. Guide cards are supplied with tags in five horizontal positions.

Cards are supplied in various colours with a variety of colour stripes. The convention for using these is published in DEUCE News No. 1 and No. 32.

Generally speaking the main card used for programmes is the instruction card with a green stripe: if the programme has several sections, then these may be defined by using cards with a blue stripe or in solid green. On no account must the following instruction cards be used for general work as they are reserved for libraries; Pink Stripe, Orange Stripe, Solid Yellow, Solid Pink.

Input data, wherever practicable, should be punched into buff cards, but several colour stripes are available as alternatives. Cards with stripes are more expensive than buff, consequently larger stocks of buff cards are maintained.

Two types of output data cards are available, red stripe and green stripe. Large stocks of red stripe are maintained, green stripe to be used as an alternative e.g. Binary output, green stripe, decimal conversions, red stripe.

Violet striped instruction cards are available for programme display and are reserved exclusively for this purpose. The practise of punching out Programme Display into red or green striped cards and subsequent reproduction into violet striped cards for ease of reading is extremely wasteful and is not to be recommended.

Stocks of cards are stored in cabinets located near to machines wherever the demand for a particular type of card is the greatest. Where several trays are used to accommodate a certain type of card, commence using from the top tray of the group and work downwards. On emptying a tray, invert, and replace, handle first.

Having removed a few cards from a tray, always ensure that the cards remaining are clamped tightly with the support provided. Failure to do this permits the cards to distort and subsequently causes card feeding troubles.

On completing a session on a machine, remove all cards, punched or otherwise. Special bins are provided for waste cards only. Waste cards have a much higher value than ordinary waste card or paper.

13.2 HAND PUNCHES.

Operation.

(a) Set swivel stop so that the pointer on the card carrier is stopped at the column from which it is required to start punching.

(b) In order to skip over unpunched fields of the card, set skip stops on columns where it is required to recommence punching. Care should be taken of unwanted skip stops. Generally it is possible to fit them to the skip bar in positions where they will not interfere with punching operations.

(c) With card carrier pointer at column 80 left hand places card into bed of machine, ensuring that the card is flat and held by the clip on the card carrier.

(d) Card carrier is moved up to the swivel stop by the left hand. If too much force is used, swivel stop will be strained and punching will start in the wrong column.

(e) Decimal punching is performed by the second finger of the right hand, supported by the forefinger and thumb. This should be a key tapping operation rather than key pressing.

(f) Right hand removes completed card from the feed bed.

NOTES. Left hand must not hold card carrier during punching as this causes "off punching".

"Off punching" is a shift right or left of the punching from the centre of a card column. Off punched digits will usually cause reading failures in subsequent machine operations.

In cases where off punching is suspected the card should be placed onto a card gauge, the card located, with Col. 80 angled by the right hand corner frame of the gauge.

Alphabetical Punching.

Several codes are in existence for the recording of alphabetical characters in cards. The code is determined by the type of equipment to be used, the usual criterion being the make and type of tabulator.

These codes usually consist of two punchings in one column of a card to provide one alphabetical character i.e. a zone digit Y, X or O combined with one of the digits 1-9 as follows:-

IBM FOUR ZONE ALPHABETICAL CODE

	ZONE	DIGITS	
NUMERIC DIGIT	Y	X	O
1	A	J	-
2	B	K	S
3	C	L	T
4	D	M	U
5	E	N	V
6	F	O	W
7	G	P	X
8	H	Q	Y
9	I	R	Z

HOLLERITH FOUR ZONE ALPHABETICAL CODE

	ZONE	DIGITS	
NUMERIC DIGIT	Y	X	O
1	A	B	C
2	D	E	F
3	G	H	I
4	J	K	L
5	M	N	O
6	P	Q	R
7	S	T	U
8	V	W	X
9	Y	Z	-

The spare position in each code is reserved for special symbols and depends on the specification of the machine that prints this information.

N.B. The alphanumeric code for the 80 column input-output unit on DEUCE Mk.II is based on the IBM 4-zone code.

Sterling.

Pounds are usually punched one card column for each digit required. Tens of shillings one card column, units of shillings one card column. Pence one card column, with X = 10d, Y = 11d.

Card Jams.

If a card becomes jammed in a punch, press all of the keys in turn, if the card does not clear, tear off card at point where now fully perforated and attempt to remove from other end. Card will usually come clear.

In the event of a small piece being left behind, use tool called "scraper" to remove. Small saw edge is provided on scraper, to saw away badly jammed pieces. All bits of card must be removed before punching can recommence.

On no account must one attempt to dismantle the machine.

13.3 HOLLERITH VERIFIER MODEL 103.

This is similar in appearance to the hand punch, and is used for checking hand punched decimal or alphabetical data.

The keyboard layout and skips are exactly the same as the hand punch.

The machine is,in part, electrically controlled, and before operation can commence switch on at the rear of the machine.

The verifier operator 'keys' the original information into the cards but instead of holes being cut, sensing pins check that the "keyed" hole is present.

The presence of an error is detected when the escapement of the machine will not operate to the next column, whereupon the operator should release the card and ascertain the error. It will be found that correctly verified columns will be indicated by a small red dot at the base of the column.

The machine will locate errors under the following conditions.

(a) More than one hole per column (excepting Alpha).

(b) A blank col. instead of a punching.

(c) A punching instead of a blank column.

(d) Wrong key depression.

(e) Off punched cards.

The skip and release key operations are not checked.

13.4 SORTER.

This is a machine which arranges decimally or alphabetically punched cards into sequence.

Depending on make and type, machines are available to operate at speeds ranging from 250 cpm to 2000 cpm.

The machine has a hopper into which cards are placed (face downwards '9' edge leading) and 12 "receiving" pockets, one for each punching position in a single card column. A 13th pocket marked 'R' for reject is for unselected cards.

A single sensing brush mounted in a carrier, senses the cards and directs them to the appropriate receiving pocket. The brush carrier may be set to permit sorting on any of the 80 card columns.

The brush carrier may be moved one card column up or down by rotating the handle found at the front of the machine. If it is desired to move over several card columns, make half a revolution with the handle, when the card column pointer will lift from the card column scale. The brush assembly may now be moved by depressing the finger lever at the top of the assembly and pushed to the desired column. Complete the full revolution of the handle, so that the pointer indicates the column to be sorted.

To sort a pack of cards into numerical sequence:-

(a) Switch on main switch.

(b) Set the brush at the least significant column i.e. units column of the field to be sorted.

(c) Place cards in hopper.

(d) Press start button.

(e) Cards will then be distributed between pockets 0-9.

(f) Take cards out of 0's pocket (retain face down).

(g) Place on top of these the 1's pocket followed by 2's etc. to 9.

(h) Set brush at next significant column.
i.e. tens column and repeat c,d,e,f,g,h until field completed.

Sorting as above will result in the cards being in ascending order when face up. Should they be required in descending order, then cards from pockets should be picked up from 9-0 on each sort.

N.B. cards should be sight checked, by looking through the cards to check selection as they are removed from each pocket. Alternatively a sorting needle may be passed through the holes.

Digit Selection.

Located below the brush operating handle is a selection drum; mounted in the drum are 12 small black switches, one for each row position of a card. If a switch or switches are pushed to the centre of the drum, then selection for these particular pockets is made inoperative and cards of that particular designation pass into the reject pocket.

It will be seen therefore that if it is desired to select a particular digit from a particular card column this may be accomplished without disturbing the order of the remainder of the cards.

Alphabetical Sorting.

This is done by again starting at the least significant column, but in this instance each column needs to be sorted twice - one pass to sort them in order of numeric digits and a second to sort them in order of their respective Y, X and 0 zones.

NOTE. When there are two or more holes punched in one column, the sorter will always select the first one it reads from 9 through to Y, unless that particular pocket has been made inoperative by the switch on the selection drum.

On the selection drum, in addition to the 12 black switches there is a larger red one; its function is, when pushed to the centre, to eliminate all the pockets 9-1 and therefore used when sorting the cards to alphabetical zones.

Having sorted on zones the cards are taken out of the pockets in the sequence Y, X and 0 and the brush turned to the next most significant column, remembering to revert the red switch to permit sorting on the numeric zone.

Card Count.

Some sorters are fitted with a switchable card counting device, which counts the number of cards passing through the selection unit of the machine. Its use is limited and optional but is useful for counting the exact number of cards in a pack. This may be done with the sorter brush raised to eliminate the possibility of disturbing their sequence.

13.5 REPRODUCER.

The two main functions of this machine are defined as follows:

Reproducing.

Is the copying of punched data from one set of cards into another set, card for card.

The copying need not be column for column, and information may be eliminated or transposed at will. The two sets of cards may also be automatically checked.

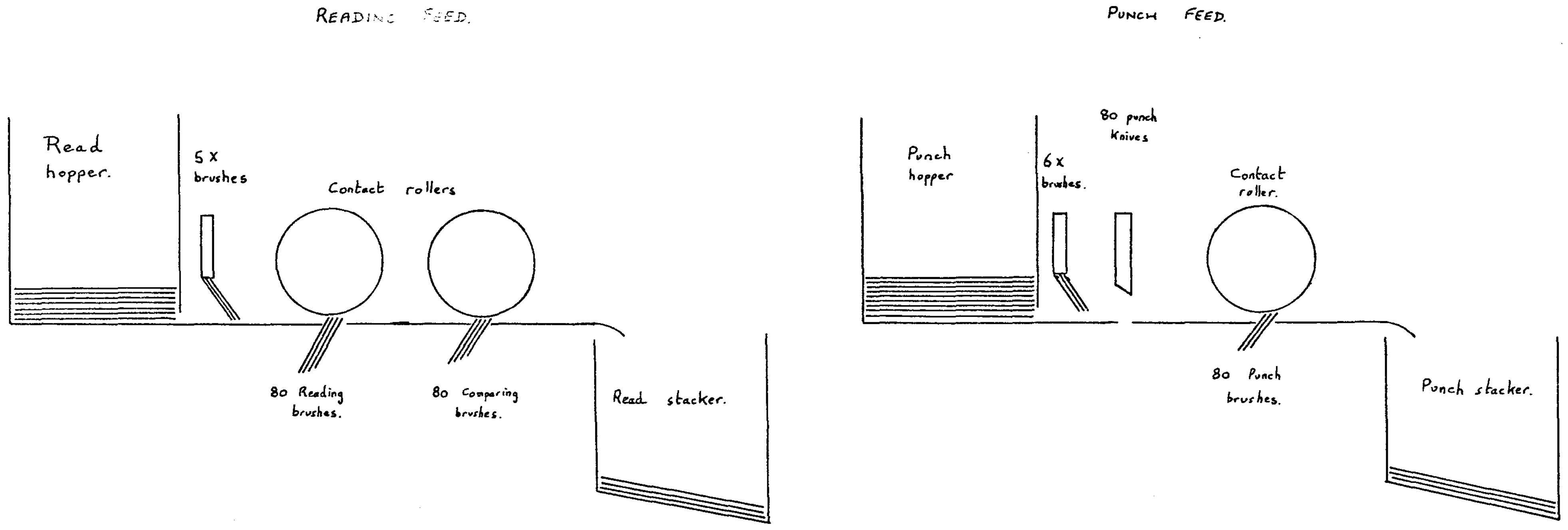


Fig. 1. Schematic Diagram of Reproducer Card Feed Units.

Gang Punching.

Is the automatic transfer of punched data from one card into a number of others.

Both of the above functions are carried out at a speed of 100 cards a minute.

Operation.

The machine may be divided into two main sections as follows (refer to schematic diagram, Fig. 1.)

Reading Unit consists of

(a) Hopper.

(b) 5.X. Brushes internally connected to plugboard.

(c) 80 Reading Brushes internally connected to plugboard.

(d) 80 Comparing Brushes internally connected to plugboard.

(e) Stacker.

Cards placed in this feed pass under the X Brushes, over the Reading and Comparing Brushes and into the Read Stacker.

Punching Unit consists of

(a) Hopper.

(b) 6.X. Brushes internally connected to plugboard.

(c) 80 Punch knives (Magnets) internally connected to plugboard.

(d) 80 Punch brushes internally connected to plugboard.

(e) Stacker.

Cards placed in this feed pass under the X Brushes, Punch knives, and over the Punch Brushes into the Punch Stacker.

If cards are placed in both feeds (pack to be copied in the Reading Feed) they will feed, move and stack in unison. The sections in each feed are not associated unless connected by plugging.

Cards are normally fed through the machine, face down Y edge leading.

Switches.

Machine is switched on by a main switch located to the right of the punch stacker.

The card 'running key' is located to the left of the reading stacker and covered by a hood. This key must not be operated until the green generator light, mounted above the plugboard, comes on.

The 'stop key' is located to the left of the 'running key' and the machine may be stopped at any time by holding down this key.

Reproduce Switch.

This consists of three switches coupled together, by a bar, and located below the 'card running' key, and switched on when reproducing. The switch introduces interlocks to both feeds, and causes the feeds to operate in unison, checking that each of the card stations i.e. Hopper, Reading Brushes, Comparing Brushes of the Reading Feed are in an identical state with the three stations of the Punching Feed.

An unequal state of any two opposing stations will cause the machine to stop.

(a) If one of the hoppers runs out of cards. The action to be taken here is to replenish the empty hopper, or to remove the excess cards from the other.

(b) A bad card failing to feed forward from the hopper to the first station. If this occurs, correct card, by straightening or repunching,replace in the hopper, and press 'running key'.

The machine will then take one cycle to feed the card to its appropriate station and will then automatically revert to reproducing.

N.B. After a card feed failure, check also the card following the bad card.

Gang Punch Compare.

This is a single switch to the right of the 'reproduce' switch. With this switch on, the interlocks between the two feeds are removed and permit each feed to operate independently of the other. It's main function is to permit Gang Punching to take place from the Punching Feed, whilst previously Gang Punched cards are compared in the Reading Feed.

It is also useful if trouble arises during reproducing and circumstances prevent cards from running out from one feed.

N.B. Do not attempt to remove cards from the machine by this means if a card jam is suspected.

X Detail and Summary Punch.

The next pair of switches in this row are marked X Detail and Summary Punching. It is not proposed to describe the function of these switches here. However during normal Gang Punching and Reproducing functions these switches should be off (UP).

Comparing Only.

The last switch in this row. The normal position for this is down. If however it is desired to compare only, two like packs of cards, then it should be switched off (UP). The function of this switch in its 'off' position is to prevent impulses from any source from operating the Punch Magnets.

Row Eliminate.

Usually these are situated immediately above the main operating switches. There is a switch for each row of the card. If a switch or switches are in their 'off' position (UP) then reproducing and comparing for that row or rows, will be eliminated.

Generally there is a signal light which operates in conjunction with the switches. The light glows when any of these switches are in the 'off' position.

Stacker Switch.

Will stop machine if stacker is full. Remove cards from stacker and press 'running' key to continue.

Plugboards.

Certain standard plugboards are available e.g. Reproduce and Compare the full 80 columns. Reproduce and Compare the 64 columns of the DEUCE fields. Reproduce and Compare old 32 column DEUCE field into Alpha field etc.. These boards always remain plugged. Other boards are available for plugging non-standard jobs.

Care must be taken when inserting plugboards into the machine. Firstly ascertain that they are inserted the correct way up i.e. with the flanges on the board at the top. Serious damage can result from boards being forced into the machine upside down.

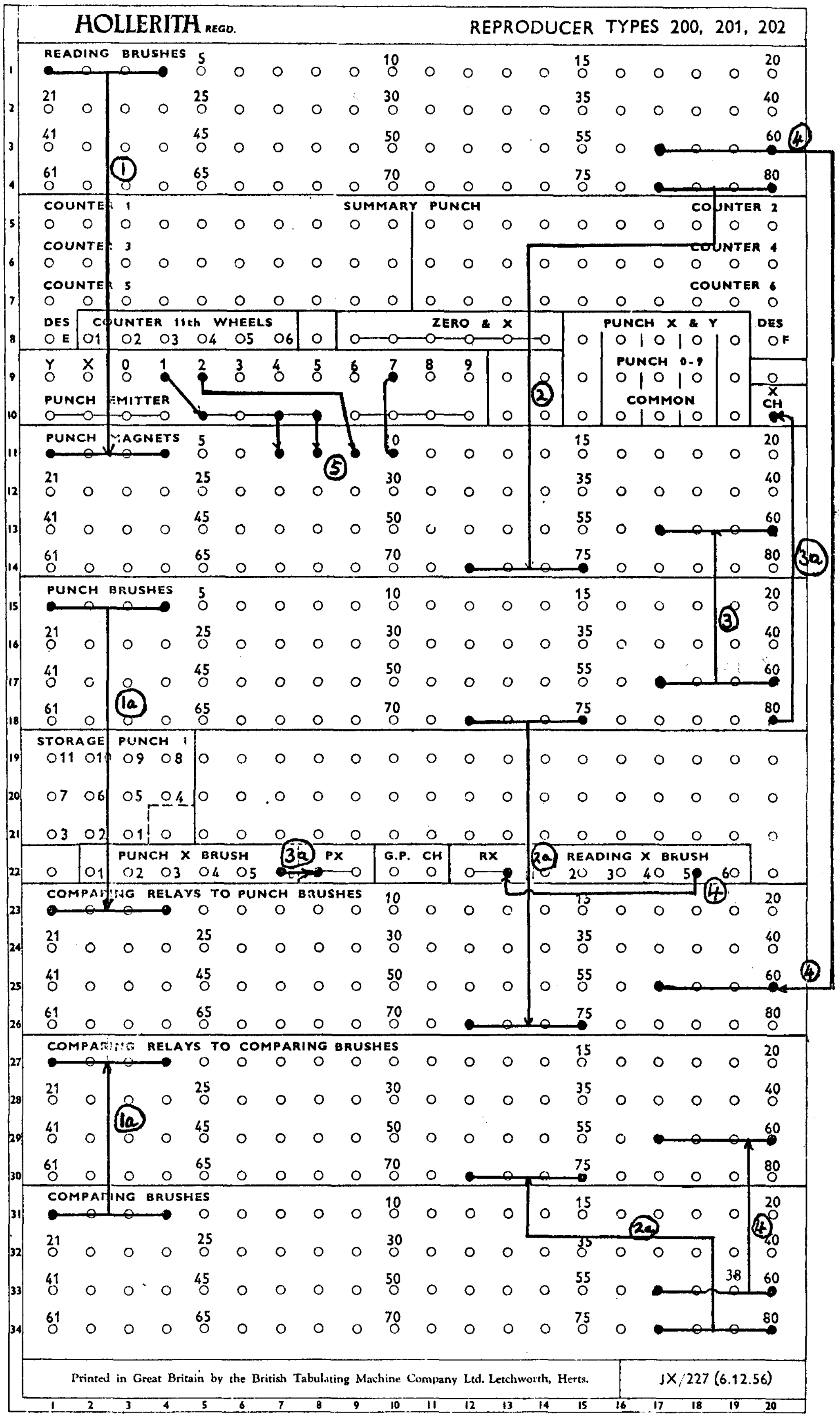


Fig 2. Plugboard Diagram for Hollerith Reproducer.

IBM

AUTOMATIC REPRODUCING PUNCH, TYPE 513 CONTROL PANEL
FOR SUMMARY PUNCHING--ALPHABETIC ACCOUNTING MACHINE

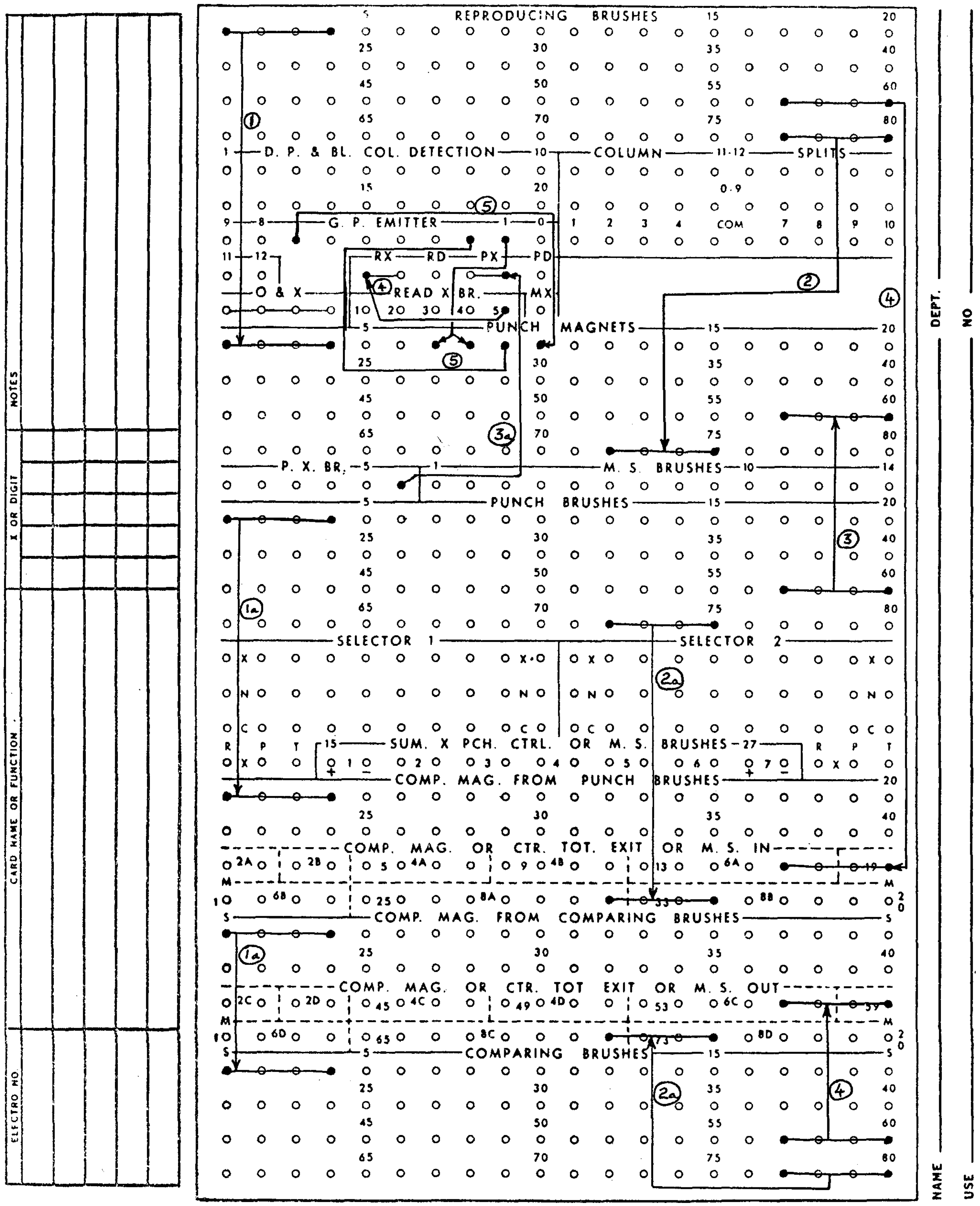


Fig. 3. Plugboard Diagram for I.B.M. reproducer.

UK Form 25-5304

Care must be taken that plug wires are not trapped when locking the board home.

Machines should not be left without a plugboard.

Plugging for Reproducing. (Refer to Plugboard Diagrams, Figs. 2 and 3).

Section (1) of diagram illustrates the plugging required in order to reproduce data from columns 1-4 of a punched card into columns 1-4 of another card.

This may be extended as required by following the same pattern.

NOTE. The single line connecting to each set of four hubs, is the convention employed to eliminate excessive line drawing, and means in actual fact that four separate plugs are used.

Section (1a) illustrates the additional plugging required to compare (verify) that the newly punched card agrees with the original.

In the event of a comparing failure the machine will stop, and a red signal light, mounted above the plugboard will glow. Reference to the indicator situated to the right of the plugboard will show which column(s) are in error.

It will be found impossible to restart the machine until the handle on the left of the indicator is 'pulled upwards'. This will reset the error indicators and switch off the indicator lamp.

In order to visually check this error, the 'Stop Key' should be held down and the 'Start Key' given a 'One Shot'. N.B. hold the 'Stop Key' down for the whole of this operation until machine stops. This will have had the effect of feeding one card into each 'Stacker'. These two cards will be the ones responsible for the error signal.

Section (2) of diagram illustrates plugging required in order to reproduce data from columns 77-80 into columns 72-75 of another card.

Section (2a) of diagram illustrates additional plugging to compare that the newly punched card is correct.

Gang Punching.

Section (3) of the plugboard diagram illustrates the plugging required to perform 'Gang Punching' operations. The requirement in this instance is to copy from columns 57-60 of a 'Master Card' into columns 57-60 of the required quantity of blank cards.

For this purpose only the 'Punch Feed' of the machine is used and therefore the 'Reproduce' and 'Gang Punch Compare' switches should be off.

Operation.

Place 'Master' card, followed by required quantity of blanks, into 'Punch' feed. Run in from Card Running Key. When the 'Master' card reaches the 'Punch Brushes' the first blank card will be at the 'Punch Knives'. The information read at the 'Punch Brushes' will be directed via the plugwires, to operate the 'Punch Knives' and perforate the blank card. On the next machine cycle the 'Master' card will eject into the 'Stacker' and the newly punched card will take up the station of 'Master' and cause punching into the succeeding card etc.

Interspersed Master, Gang Punching.

This is a method of 'Gang Punching' when 'Master' cards containing differing punched data are interspersed throughout a pack of blanks. Using the ordinary Gang Punching method described above, each 'Master' card would be superimposed with the data from the previous card. To overcome this difficulty a set of 'X' sensing brushes are provided. There are six of these located immediately in front of the 'Punch Knives'. They may be set to read from any card column. The object of them is to suspend punching operations for one card feed cycle following the sensing of an 'X' punching, in some predetermined card column. Therefore each 'Master' card employed in this method, must be punched with an 'X' and an 'X' brush set to read from that card column.

The additional plugging required to extend 'Gang Punching' example (3) to 'Interspersed Masters' is illustrated by (3a). This assumes that 'X' brush 6 is set at column 80, and that the 'Master' cards are punched with an 'X' in column 80.

NOTE: The plug connecting 'Punch' brush 80 to 'X CH' is for checking that the 'X' in column 80 was read by the 'X' brush. A failure to read by the 'X' brush would cause the machine to stop and the red error light to glow. The error condition would have to be cleared by pulling the handle at the side of the 'Comparing Indicator', plugboard removed (to avoid erroneous punching), the cards run out, and the 'X' brush setting and plugging checked.

Interspersed Master Gang Punching, Checking.

By using the 'Reading' feed of the machine, this type of 'Gang Punching' may be automatically checked.

The machine has a further set of 5 'X' brushes, located immediately in front of the 'Reading' brushes. The plugging required for checking the previous example is illustrated in Section (4) of the plugboard diagram. It is assumed that 'Reading X' brush 5 is set on column 80.

To use the machine for this purpose the 'Gang Punch Compare' switch must be ON.

It is permissable to operate the machine simultaneously in the two different modes. i.e. Gang Punching from the Punch Feed and Comparing from the Reading Feed.

In the event of a comparing error, the same action should be taken as for a Reproducing error, described in 'Plugging For Reproducing', except that two 'One Shots' should be given and the error cards will be the two top cards in the 'Reading Stacker'.

Punch Emitter.

Consists of twelve hubs each of which emit their indicated digit, once in each card feed cycle. They are usually connected to the 'Punch Magnets' and are used as an alternative to, or in addition to Gang Punching and Reproducing.

Section (5) on the plugboard diagram illustrates the emission of the number 1127 into card columns 7-10.

The three sets of four hubs, each set connected by a line and located immediately below the 'Emitter' (Hollerith Reproducer only) are known as 'Bus Hubs' or Common Hubs. Their function is to provide three sources from a single source plugged into them e.g. the digit '1' in the previous example.

Column Splits.

Are a group of hubs located to the right of the 'Digit Emitter'. Let it be sufficient to say that if a reading source is connected to the 'Common' of one of these hubs, the digits 0-9 from that source will be available at the 'Punch 0-9' hub and the zones X and Y available at the 'Punch X and Y' hub.

13.6 TABULATORS.

As tabulators are usually constructed to a customers' specification, very few are identical. It is therefore proposed to discuss these very generally.

Basically the tabulator is a printing machine, capable of reading decimal or alphabetic punching from any of the 80 columns of a card and directing this information to a set of print bars or print wheels, to produce one line of printed data. Print bars/wheels may vary in number between 50 and 120.

The machine is programmed by means of plugwires inserted into a control panel, making it possible for any of the 80 card columns to be connected to any of the print bars/wheels, and a variety of other flexible functions.

Speed of printing varies according to machine, from 80 to 250 cards per minute.

Fields from the card may be accumulated in adding units of the machine, and totals printed out at determinable points, usually when a punched field common to one set of data changes i.e. Date, Part No. Clock No. etc., or by a single punching unique to a particular type of card.

Printing out of totals may be accomplished at varying levels of precedence, according to the significance of change in data. e.g. One level of total may be printed out on a change of day, a sum of the days to date on a change in month, and a sum of the months to date on a change in year.

The range in counting units, which may be grouped to form counters of the desired size vary between 20 and 120 and they are available for adding or subtracting decimal or sterling data.

Information may be added into counters direct from the card without simultaneous printing and only totals from groups of cards printed out. Usually this can be accomplished at a higher card feeding rate.

Discriminatory facilities are available to permit, amohg other things, re-allocation of type bars/wheels for a particular type of card e.g. Positive numbers printed on one set of type bars/wheels and negative numbers on another set by discriminating on the sign of the number. Likewise information may be allocated to different counting units.

A marker digit is a hole punched in any position in any column of the card and made unique to any particular function required of the tabulator e.g. a Y punched in column 80 could mean 2 or 3 extra spaces before printing this card. An X punched in the same or another column, 2 or 3 extra spaces after printing this card. A 9 similarly could cause the tabulator to feed to a new sheet of paper before or after printing this card. All of these markers punched in one card would call all the nominated functions.

If all 80 columns of the card contain data, marker digits may be allocated in the X and Y rows of the card, provided that the columns chosen will never contain alphabetic data.

Attachments are available for fitting to machines to enable them to produce up to four carbon copies of results. It is also possible to use tabulators to produce stencils and duplimats.

It is always advisable before introducing a punch routine into a programme, to discuss the desired print layout etc. with the tabulating supervisor.

13.7 ENGLISH ELECTRIC CARD OPERATED TYPEWRITER.

This equipment consists of an electric typewriter mounted on a control console, and a card reading device.

The card reading device will read punched cards at a rate of 10 card columns per second, and operate all the standard functions of the typewriter.

Control over the printed layout is maintained by a set of instructions punched in a single programme card mounted in the card reader.

Control is also available from instructions punched into the input cards, or a combination of programme card and input card instructions. This provides extreme flexibility in the layout of data.

13.8 AUTOMATIC PUNCHES/VERIFIERS.

Several types of machine are available, designed for punching and verifying large quantities of decimal or alphabetic data.

Card feeding and stacking is automatic.

Two types of keyboard are available; 'Numeric' which is similar in layout to the hand punch, and 'Numer/Alpha' which is similar in layout to the standard typewriter keyboard, (depression of one Alpha key causes the equivalent two hole Alphabetic code to be punched). Keys are of the 'touch' type.

Card layout, skipping, gang punching etc. is controlled either by a simple 'programme card' or 'plugboard'.

Additionally one particular machine will print the digits or characters punched, along the top edge of the card.

13.9 COLLATORS.

These machines are controlled by 'plugboard programme' and are extremely versatile. All the uses to which they may be put are too numerous to include here. It is hoped however that the following list will be of some assistance:-

(a) Merge together two packs of cards of like sequence.

(b) Select unmatched cards from either of two groups, and simultaneously merging matched cards if required.

(c) Select into one pocket, cards of a particular denomination, rejecting the remainder.

(d) Sequence check packs of cards.

Speeds vary according to model and make between 200 and 650 c.p.m. from each of the two card feeding units.

13.10 INTERPRETERS.

Cards may have punched data 'interpreted' or printed on the actual card itself.

Several models of Interpreter are available, all controlled by simple plugboard programme. Some will interpret all 80 columns of a card; others only 60 columns on one run. Certain models can accommodate up to 19 rows of print on one card. Speeds vary between 8 and 100 c.p.m. according to model and make.

13.11 MARK SENSING REPRODUCER.

From a horizontal, or near horizontal pencil mark, three card columns in width, the Mark Sensing Reproducer can produce a punched hole in a card. Up to 27 such pencil marks may be made on each side of a card. (Two runs through the machine would be required if each side of a card were marked).

Mark Sensed columns may be plugged to cause punching in any desired card column.

Specially printed cards and very soft lead pencils (e.g. 2B) must be used.

Standard reproducing and gang punching facilities are fitted to these machines and at all times their speed of operation is 100 c.p.m. from each feed.

13.14 DATA TRANSCEIVERS.

These machines transmit or receive punched card data over telegraph or telephone lines, or by radio.

Operating speeds are up to 11 c.p.m.

13.15 CARD TO TAPE, TAPE TO CARD.

Machines are available to translate paper tape into punched cards and vice versa.

14

AUTOMATIC CODING BY JOE DIGITS.

14.1 INTRODUCTION.

Lecture 6 defined the DEUCE instruction word as

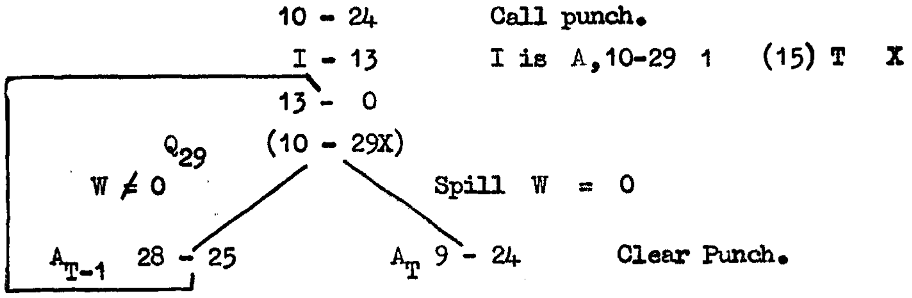
$$NP_1 + SP_5 + DP_{10} + WP_{15} + WP_{17} + TP_{26} + GP_{32}$$

and stated that W takes values 0 - 31 using digits P_{17} to P_{21}. T does not start until P_{26} leaving a gap of P_{22} to P_{25} unaccounted for.

These digits are known as the JOE digits, historically because a JOE is a useful stooge. These four digits provide a useful link between the WAIT number and the TIMING number and if present can be used in conjunction with instruction modification of the W number to alter the TIMING number after a predetermined number of modifications. This process, known as SPILLING out of an instruction is now described.

14.2 EXIT FROM AN INSTRUCTION BY SPILLING UP.

The process is best illustrated by an example. Suppose we wish to punch the successive minor cycles of D.L.10 on 32 rows of 3 cards and exit from an instruction modification loop without using a separate counter. The instructions are:

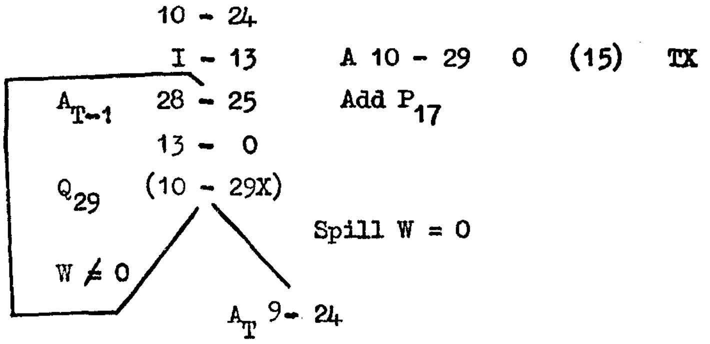


By arranging for the modified instruction to be obeyed Q_{29} and the initial wait number to be 1, the first minor cycle punched is 10_0. Since the NIS is A and the timing number T, Q_{29} leads to an instruction in D.L. A m.c. (T-1). This instruction increases the wait number by one, causing the next m.c. of D.L.10 to be punched on the next trip round the loop. Eventually, after 31 additions to the instruction W becomes 32 $(\equiv 0.)$ The carry to P_{22} is transmitted through P_{22} to P_{25} and increases the timing number to T + 1. On the 32nd time Q_{29} causes 10_{31} to be punched and the instruction now leads to A_T which takes us out of the loop.

There is the basic idea. Add to the wait number until a carry passes through digits P_{22} to P_{25} to increase the timing number by one. Like all simple ideas it has lots of possibilities; we can add two to the wait number to operate on double transfers and spill after 16 operations; we can add P_{22} digits and obey an instruction or series of instructions with unchanged wait numbers, spilling after any number up to 16 by suitably choosing the initial value of P_{22} to P_{25}. We can subtract P_{17} from a Wait number, leaving P_{22} - P_{25} blank and eventually subtract one from the timing number. All these are the same in principle and their efficient use in programming is a coding exercise.

In the example given above the instruction 10 - 29 is obeyed once unmodified and 28 - 25 is obeyed after the instruction is obeyed. This is defined as POST-ADDITION. Another routine to do the same operation adds P_{17} to the instruction before it is obeyed the first time .

This is defined as PRE-ADDITION, the instructions are



Note that the instruction has W = 0 in the coding but **is obeyed the first time with W = 1**

14.3 RULES FOR CODING SPILL ROUTINES.

It will be apparent that spill occurs when W = 32 and the instruction is acting as an automatic full count detector. We require to know 3 facts before coding a spill routine.

(a) How many operations* are required before spill occurs.

(b) The minor cycle of first operation transfer.

(c) Whether POST ADDITION or PRE-ADDITION is to be used. In the case of 13 - 0 or 21_2 or 21_3 - 0 this is within the programmer's choice but not in the case of 17 or 18 - 0 with A.I.M.

* By "operation" we mean obeying the quasi instruction.

If POST ADDITION is used the number of additions made is one less than the number of operations. If PRE-ADDITION is used the number of additions is equal to the number of operations.

To code the routine we must choose W_0 the initial wait number and q the quasi minor cycle.

Rules.

1. Choose W_0 such that

$$W_0 + (\text{No. of additions}) = 32$$

2. Choose q so that

$$q + W_0 + 2 = \text{first operation minor cycle}.$$

Example 1.

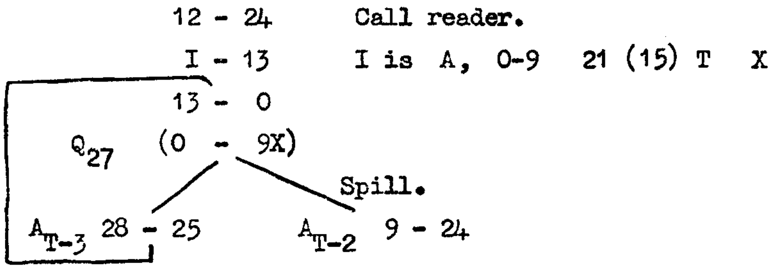
To read successive rows of a card into D.L. 9 starting at 9_{18}. POST-ADDITION.

No. of operations 12.
No. of additions 11.
Unit quantity added (P_{17})
Therefore W_0 + 11 = 32.
Therefore W_0 = 21

First minor cycle of operation = 18.

Therefore q + 21 + 2 = 18.
or q = 27.

The instructions are



Example 2.

To punch successive minor cycles of D.L. 9 from 9_{10} to 9_{17} on successive rows of a card. Post-addition is used.

No. of operations 8.
No. of additions 7.
Unit quantity (P_{17}) added.

Therefore $W_0 + 7 = 32 \quad W_0 = 25$

First minor cycle of operation = 10.

Therefore q = 10 - 25 - 2 = 15. [Adding 32 to give positive result]

The instructions are:

```
                    10 - 24     Call punch.
                     I - 13     I is   A,  9-29   25 (15) T  X
        ┌───────────13 - 0
        │   Q15     (9 - 29X)
        │            /     \
        │       28 - 25     Spill.
        │ A T-15         A T-14  9 - 24.
        └─────────┘
```

Example 3.

To add corresponding minor cycles of D.L.9 and D.L.10 from m.c. 0 to m.c. 30, placing the results in 12_0 to 12_{30}.

Two fetch and one store are needed. Only one of these need be of the counting spill type and the store instruction is chosen.

Assume Pre-addition is used and 21_2 and 21_3 are available for instruction modification.

No. of operations 31
No. of additions 31
Unit quantity (P_{17}) added.

Therefore $W_0 + 31 = 32$ or $W_0 = 1$.
$q + 2 + 1 = 0 = 32$ therefore $q = 29$.

The instructions are:

```
                 I, J - 21₂₃     I is  9 - 13
                                 J is 10 - 25
                    K = 14       K is  A  15 - 12   1  (15)   T
      ┌───────────21₂ - 0
      │    Q30     (9 - 13)      Fetch from D.L.9.
      │           21₃ - 0
      │    Q29    (10 - 25)      Fetch and add D.L.10.
      │            28 - 22 (d)   Modify I and J.
      │            13 - 15       Store result TS15.
      │            14 - 13  ┐
      │            28 - 25  │    STORE result in D.L.12 and modify store
      │            13 - 0   │    instruction.
      │    Q29    (15 - 12) ┘
      │            /      \
      │ A T-1  13 -14      A T  Spill.
      │          │              EXIT.
      └──────────┘
```

Example 4.

To fetch m.c. 0 to 15 of D.L.9, multiply each by m.c. 0 to 15 of D.L.10 and store double length results in D.L.12. All numbers considered positive. Post addition is used To store double length results P_{18} will be added to the store instruction each time. The quantity added is thus $2P_{17}$.

No. of operations	16
No. of additions	15
Addition unit	$2P_{17}$

Therefore $W_0 + 15 \times 2 = 32$

Therefore $W_0 = 2$

First operation minor cycle is m.c. 0.

Therefore $q + 2 + 2 = 32$

$q = 28.$

The instructions are

	I, J - $19_{2,3}$	I is 9 - 16
		J is 10 - 15
	k - 14	K is A, 21 - 12 (d) 2 (15) T
	19_2 - 13	
	13 - 0	
Q_{30}	(9 - 16)	Fetch from D.L.9
	16 - 21_3	Multiplier to 21_3
	30 - 21_2	Clear 21_2
	28 - 25	Modify D.L. 9 fetch
	13 - 19_2	instruction.
	19_3 - 13	
Q_{30}	(10 - 16)	Fetch from D.L.10.
	0 - 24	Multiply.
	28 - 25	Modify D.L.10 fetch
	13 - 19_3	instruction.
	14 - 13	
	13 - 0	
Q_{28}	(21 - 12 (d))	Store result.
		Spill
A_{T-2}	28 - 25 (d)	
	13 - 14	A_{T-1} EXIT.

The above sequence of instructions is adequate but may be improved by modifying the source number of the fetch instruction.

```
                    I - 15          I is   9 - 16
                    K - 14          K is   A  21-12  (d)  2  (15)  T
   ┌────────────────┘
   │               15 - 13
   │               13 -  0
   │        Q30    (9 - 16)         Fetch from D.L.9
   │               28 - 25          Modify D.L.9 fetch
   │               13 - 15
   │               16 - 21_3
   │               30 - 21_2
   │   P5 + (-P17)  - 25            Change source to D.L.10 and remove P17
   │               13 -  0
   │        Q30   (10 - 16)         Fetch from D.L.10
   │                0 - 24
   │               14 - 13
   │               13 -  0
   │        Q28   (21 - 12 (d))     Store in D.L.12.
   │                  /   \
   │   A_T-2   28 - 25(d)    A_T-1 EXIT.
   │           13 - 14
   └──────────────┘
```

It may be apparent that the addition of P_{18} to an instruction permits two correct choices of coding. Spill can occur when the wait number is either 32 or 33. This permits two alternative values of initial wait number and two quasi-instructions. On this basis

	No. of operations = 16
	No. of additions = 15
	Quantity added $2P_{17}$
Therefore	$W_0 + 15 \times 2 = 33$
or	$W_0 = 3.$
First operation	m.c. = 0
Therefore	$q + 3 + 2 = 32$
Therefore	$q = 27.$

The reader should check that an initial wait number of 3 and a quasi minor cycle of 27 will effect the appropriate spill after 16 operations.

14.4 EXIT FROM AN INSTRUCTION BY SPILLING DOWN.

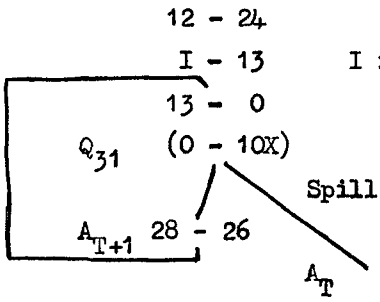
This process is used for backward fetch and store routines. P_{17} is subtracted from an instruction until the carry through P_{22} to P_{25} subtracts one from the TIMING number. Again post or pre-subtraction may be used and criterion for spilling is $W = -1$.

Example.

To read 32 successive rows from cards to D.L.10 m.c. 31 to 0. Post subtraction.

	No. of operations 32.
	No. of subtractions 31.
	Unit quantity subtracted.
Therefore	$W_0 - 31 \times 1 = -1$
or	$W_0 = 30$
First operation	m.c. = 31
Therefore	$q + 2 + 30 = 31$
or	$q = 31.$

The instructions are



```
          12 - 24
          I - 13      I is A, 0-10 30 T (1) X
          13 - 0
     Q31  (0 - 10X)
                    Spill.
   AT+1   28 - 26
                      AT
```

If T > 0 it will be safe to subtract one from T. If T = 0 the subtractive carry will pass through P_{31} to P_{32} and change a GO instruction to a stop and vice versa. P_{31} = 1 is used as a guard digit to avoid this unpleasant feature of subtractive instruction modification.

15

LECTURE 15

AUTOMATIC INSTRUCTION MODIFICATION (AIM).

15.1 BASIC PRINCIPLES.

Modification of instruction words is a device of frequent occurrence in programming a digital computer and any simplification of the process is likely to lead to considerable saving of time and storage space and to vitally affect the economy of the programme.

The 'Destination 0' facility of DEUCE assists the modification of instructions in the accumulators. The disadvantage of this method is that the accumulator used is not available for the calculation in progress at the time of modification. Furthermore a modifying instruction is necessary and in many cases a modifying word also.

The most common modification is simply the addition of one to the Wait Number of an instruction enabling that instruction to operate on successive minor cycles of a long delay line. Frequently two is added to the Wait Number in order that successive pairs or alternative minor cycles may be dealt with. Sometimes subtraction is more appropriate as when data is stored 'Backwards'. Next most common are modifications of the Source and Destination numbers by unity for continuous storage of larger quantities of data. Other parts of the instruction word are modified comparatively rarely.

Exit from a modification sequence is often achieved by the spilling of the counted up Wait Number into the Timing number via the intermediate 'Joe digits'. Greater flexibility is obtained by provision of a facility whereby the 'Joe' number is increased by one at the same time as the remainder of the instruction is modified.

Most programme 'loops' require some form of count to ensure the correct number of cycles of the loop. This usually requires the use of an accumulator but a method of automatic counting is made available by the new facility.

15.2 OPERATION.

During the first, and only the first minor cycle of transfer of a Destination 0 instruction of the form 17 - 0 or 18 - 0, the word stored in the appropriate minor cycle of the quadruple store is automatically modified according to the rules given below. Different modifications are available on QS17 and QS18 and also according to the P_1, P_2, P_3, P_4, and P_{15} digits of the Destination 0 instruction. Modification does not occur in subsequent minor cycles of transfer.

Operation of 'Destination 0' is unchanged in that the normal next instruction is replaced by a word from the quadruple store only if transfer is still taking place when the timing number has counted down. Hence, if it is required that the word modified by the transfer enters control as the next instruction, it is necessary that the last minor cycle of transfer should either be the first, or differ from it by an exact multiple of four minor cycles.

It is important to note that it will be the modified word that is obeyed, not the original word planted in the quadruple store, and furthermore that a copy of the modified word will be left in the quadruple store after the modification has taken place, ready for subsequent modification if required.

15.3 RULES FOR SPECIFYING MODIFICATION.

The type of modification effected depends on the coding of the D.O. instruction, according to the following scheme.

DIGITS PRESENT IN D.O. INSTRUCTION.		DIGITS ADDED TO WORD IN QS.
NO P_1, NO P_2	$(= 0 \times P_1)$	P_5
P_1, NO P_2	$(= 1 \times P_1)$	P_{10}
NO P_1, P_2	$(= 2 \times P_1)$	P_{17}
P_1, P_2	$(= 3 \times P_1)$	P_{18}

If a P_{15} is present, then the presence of a P_3 will cause an additional P_{22} to be added and the presence of a P_4 will cause the digits determined by P_1, P_2, and P_3 to be subtracted instead of added.

If a P_{15} is not present, the P_3 and P_4 have no effect. (This case is normally used for automatic counting. The use of the P_2 here clearly imposes a slight restriction on the choice of Next Instruction Source).

The above coding holds for 17 - 0 instructions.

For 18 - 0 instructions, a similar scheme holds but subtractions replace additions and vice versa.

N.B. Suppression of 'carry' occurs only when two digits are added simultaneously. Thus there will be no carry to the P_{22} position if a P_{22} is being called.

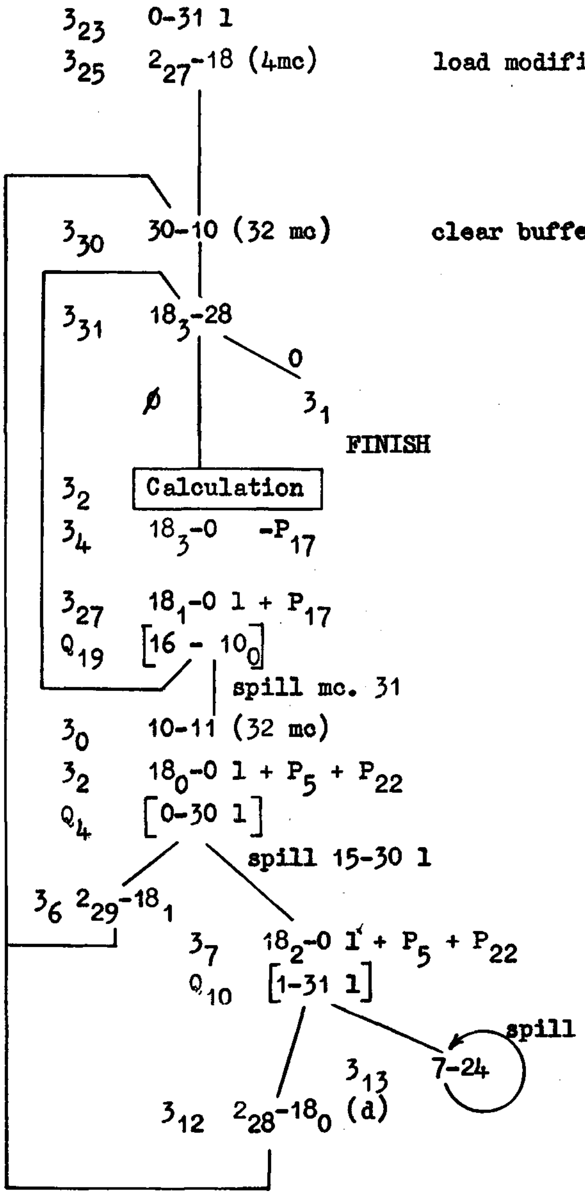
15.4 AUTOMATIC COUNTING.

By obeying an instruction of the form 17 - 0 or 18 - 0 during a loop and coding it in such a manner that the 'Destination 0' facility does not take effect (see 15.2) an automatic count can be set up and exit from the loop achieved by discriminating on sign or nullity in the appropriate minor cycle of QS17 or QS18, the original word placed there having been suitably chosen according to the number of times the loop is to be cycled.

It should be noted that counting may be positive or negative (in general counting up will take place in QS17 and counting down in QS18) and in terms of P_5, P_{10}, P_{17} or P_{18}, but that the specification of the counting field will impose a restriction on the Next Instruction Source of the D.O. Instruction which, of course, acquires its normal significance in this application. This restriction is unimportant in practice except in the case of reproduction of subroutines into alternative delay lines, an aspect which will concern only the DEUCE library organisers.

15.5 EXAMPLE.

To calculate values of a function and store them in successive minor cycles of the drum, exiting from the loop after a given number of repititions.



load modifier.

clear buffer DL 10.

Anticipate head shift first time

2_{27} = counter x P_{17}
2_{28} = 3, 31-29 1, 0, 0.
2_{29} = 3, 16-10, 0(15)0.
2_{30} = 3, 0-31, 1, 0, 0.

Is count finished?

Calculate next value - store in TS16.

Count. (Instruction is $6P_1$, 18-0, 0, 4) NOT obeyed.

($10P_1$, 18-0 1, 0, 0)

Store in next available mc. of DL10.

Copy DL10 into DL11,

($12P_1$, 18-0 1, 0, 0)

Write on next available track.

Reset DL storage instruction.

($12P_1$, 18-0 1, 1, 1)

Shift to next head position.

Failure alarm - drum full.

Reset DL storage and write instructions.

LECTURE 16

PROGRAMME TESTING.

16.1 INTRODUCTION.

These notes set out a procedure for programme testing that will be adequate for beginners. Although programme testing is a skill which can be developed to a high degree of sophistication, it is possible to get satisfactory results with an elementary knowledge of the method of attack and correct use of the tools of the trade.

It is a popular fallacy that a programme tester needs to be a wizard at the DEUCE Control Panel; in fact it is more valuable for you to learn to plan your programme testing run before going on the computer, and to learn to analyse the information the computer gives you after leaving the computer. In this way you will be able to do most of your thinking away from the computer, calmly and at your natural speed.

A bibliography of more advanced reading is given at the end of these notes.

16.2 METHOD OF ATTACK.

You have just read that there are three stages in a programme testing session, which are:

(a) Before going on the DEUCE.
(b) While on the DEUCE.
(c) After leaving the DEUCE.

These stages will now be described separately.

(a) Before Going on the DEUCE.

You should eliminate, by systematic checking, as many mistakes as possible before going on the DEUCE. You should have a clear idea of what the DEUCE should do when it is executing your programme. Knowing this, you can decide what things you will look for and in what order. You should prepare test data. These are the general aims. To make it easier for you, we have prepared the following questionnaire which you can check through each time before you go on the computer, until you know the routine.

QUESTIONNAIRE

	Question.	Method.
1.	Have you checked your coding?	Take the coding sheets and calculate, from the Wait and Timing numbers, the transfer minor cycle and next instruction minor cycle and write these on the coding sheet as subscripts to the NIS, S and D. Also calculate the length of transfer if this applies, and write this in the 'go' column. Ask someone to call out the instructions, in flow diagram order, while you check them against the flow diagram.
2.	Have you checked your subroutines?	Preferably ask someone else to check that you have put the link and data in the correct stores before entry, that the link is coded from the correct quasi minor cycle, that everything under the headings 'Parameters' and 'Uses' in the subroutine write up have been correctly incorporated.
3.	Have you checked your punched programme?	Ask someone to call back your binary programme cards while you check them on your coding sheets. Check triad headings, lead-in instruction, decimal card numbering (in cols. 13-16) which should tally with your coding sheet card numbers. Reproduce all cards which have holes filled in. Finally, check that the cards have not got out of order, and that the necessary standard cards are on the front of the pack.
4.	Have you made a list of the failure indications in your programme?	These should give the pattern (NIS, S, D) that will appear on the I.S. lamps on failure. Do not forget those in the subroutines you use. Write against each what it means.
5.	Have you prepared test data?	In the early stages, you should use a small quantity of data (if the quantity is variable), making the numbers arithmetically convenient so that you can easily check both final results and intermediate numbers. Do not overlook that some of these numbers must be checked against the DEUCE's binary results (e.g. 0.3 is less favourable than 0.25!). Later you should prepare test data (maybe several runs with different data will be necessary) to stretch the programme in all permissible directions. (You will find it invaluable if you write out at the beginning the exact format of all the data and a specification of the programme giving the maximum and minimum values the data may take and the limits on the quantity of data). Make sure that every string of instructions gets tested at some time or another, including the failure indications. See whether you can make use of any data to hand with worked-out results obtained by old fashioned hand methods.
6.	Have you put stoppers in convenient places in the flow diagram, that will help you to check the continuity of your programme?	Convenient places are the first instruction after the Reader has been cleared (not on the 9-24 instruction itself, however), the first instruction after leaving a loop, the first instruction after new instructions have been fetched, the last instruction of all.

7. Have you marked all the places you want to 'Request Stop'?

Convenient places are on the instruction that tests for leaving a loop (so that you can examine the counter the first two or three times round the loop to make sure it is being altered correctly) or at a point where you want to make sure that data is being fetched or stored correctly the first two or three times. Avoid request stop on 17-0 or 18-0 or on instructions with destination 24 as the DEUCE will not function correctly in these circumstances.

8. Have you marked the places where you want to Post Mortem?

Strategic places are where you have numbers that need checking and that will otherwise be lost (normally you will not take more than one Post Mortem, and that just before you leave the computer. This is because after a Post Mortem it is usually necessary to restart from the beginning).

9. Are you sure how to use the programmed and engineered facilities you plan to use?

Make sure that you understand the written instructions for using these (the more usual ones, are described in the section 'Tools of the Trade' and that you can identify on the DEUCE console all the keys, lamps etc. that are referred to.

10. Have you made a plan of the things you expect DEUCE to do and that you have to do, in the order in which you expect them to happen?

This will help you to recognise when anything goes wrong, and to be sure you take away from the computer all the punched information you intended to. This is particularly important since you will probably find that computer operation is not conductive to lucid thought! In this context you may find the presentation of information on the P.T.A.T.O. sheet (back page of these notes) worth copying. P.T.A.T.O., incidentally, stands for Programme Testing Assigned to Operators, and is a scheme for giving stage (b) of a programme testing session to another person who is proficient at operating the DEUCE but who knows nothing about the programme being tested that is other than what is written on the P.T.A.T.O. sheet.

(b) While on the DEUCE.

Take with you:

(1) Plan of campaign.

(2) Programme cards, test data.

(3) Coding sheets, flow diagrams and list of failure indications.

(4) Instructions for using the computer and cards of the Post Mortem programme.

When you arrive at the computer, put the keys, lamps, etc. on the Control Panel, Punch and Reader in the following standard state.

Control Panel Keys.

All keys level except Stop Key.
Stop key raised (on NORMAL).

Control Panel Lamps.

32 O.S. lamps off.
32 I.D. lamps off.
Red lamp above stop key off.
Lamp above Read key off.
Lamp above Punch key off.
No red lamps on the upper right hand corner of the control panel (a red light here should be reported to the engineer).

Lamps on Reader.

All lamps off.

Lamps on Punch.

Ready lamp on.
All other lamps off.

Switches on Punch.

Counters re-set to zero.
Parameter switches OFF or set to required pattern.

Cards in Punch.

Cards removed from hopper.
Punch run out and cards removed from stacker.
Hopper refilled (salmon stripe data cards for results, mauve stripe programme cards for Programme Display and Post Mortem).
Punch run in.

During the run, follow your plan of campaign carefully. Write down anything peculiar that happens (e.g. an unexpected pattern on the I.S. lamps, Read still called when you thought you had cleared it) as you may find your memory plays tricks.

Before you leave the computer, press the run out key on the Punch, collect your cards from the Reader stacker and the Punch stacker, sign the log book and then move away as quickly and completely as possible.

(c) After Leaving the DEUCE.

The Use of Incorrect Answers.

Do not throw these away, as they may give you information of two kinds. First examine the general layout and the number of cards. These may make you suspicious about some particular part of the programme such as a counting operation if you get the wrong number of results, or missing or extra stoppers if there is a systematic displacement of the rows.

Second, even if they are wrong the actual numbers may be useful to pinpoint an error, particularly if numbers which should be different are the same or that should be the same are different.

The Use of PROGRAMME DISPLAY Results.

Programme display cards can be checked against the flow diagram. Usually, it is adequate to check just the S and D (incidentally to do more is quite hard work) if you are trying to find out whether the continuity is alright or if the programme meanders off in an unintentional direction.

There are some snags. If you reach a patch of instructions you do not recognise , they might be from a subroutine and therefore not on your coding sheet. A large pack of cards can look formidable but can be examined rapidly, if there is some repetition as the programme goes round a loop several times, by looking for the discrimination instruction or for an instruction that is obeyed repeatedly with modified Wait number. Remember that the NIS of one instruction specifies the D.L. in which its successor is stored.

The Use of POST MORTEM Results.

Post Mortem might have been used with a specific purpose in mind, e.g. to check whether certain data were stored in the correct place, or that certain intermediate numbers were correct. When something unexpected has been found further use can be made of them.

In trying to explain some peculiarity or if you are not even sure where the Post Mortem was taken it may be worth looking at the following things:

(1) Counters.

(2) Modified instructions.

(3) Stores to which instructions are sent during the programme, especially 1_{30}, 1_{31} etc. if subroutines are used.

(4) Stores to which numbers are sent consecutively during the programme e.g. a D.L. which holds successive tracks of the drum, or a TS which holds successive words of a D.L.

Having found a Mistake.

After finding a mistake as the result of evidence collected from the computer, you should find out just what the mistake accounts for. It may happen that the evidence you have can be used to reveal more than the one mistake. Also, if your mistake was due to a misunderstanding of how the computer works, look ahead to see if you have made the same mistake later on . (An obvious example is not clearing 21_2 before starting multiplication).

16.3 TOOLS OF THE TRADE.

The Reader.

Place cards in the Reader hopper, face inwards and with the Y-row edge leading. Put in as many as you can comfortably hold in your hand. Read in the first handful of cards with the Initial Input key. When the Reader hopper is empty and card position lamps 1-5 are off, remove cards from the stacker and refill the hopper. Read in these cards with the Run In Key.

The Punch.

If the punch contains the wrong colour cards, remove cards from the hopper and press the Run Out key. Place blank cards in the Punch hopper, face downwards and with the Y-row edge leading, and press the Run In key. When the Punch hopper is empty, remove cards from the stacker, refill the hopper and press the Run In key.

To Continue when a Stopped Instruction has been Reached.

Press down the Single Shot key and release it.

To Start Programme Display.

(1) Put the stop key at AUG. STOP.

(2) Fill the punch hopper with Programme Display cards (i.e. instruction cards with a mauve edge).

(3) Press down Programme Display key to 'ON' and release it.

To Stop Programme Display.

RAISE the programme DISPLAY KEY to 'OFF'. The Punch may stop for any of the following reasons:

(1) No more cards. The card feed is empty and the punch is not READY. To continue remove cards from the stacker, refill the feed hopper and RUN-IN.

(2) The programme on display comes to a stopper. The Punch CALLED lamp goes off, the red light above the RELEASE Key comes ON. To continue press the RELEASE key and let go, (the red light should go off) and then press the PROGRAMME DISPLAY key again.

(3) The programme on display comes to 9-24 and puts the punch off. The Punch called light goes off but the red STOP light is not on. To continue, RAISE the Programme Display key to OFF, and then depress it to ON.

Request Stop.

To Request Stop on NIS, S, D (where one, two or all of NIS, S and D may be specified).

(1) Stop key to STOP.

(2) Set the given NIS, S and/or D on the I.S. keys (level = 0, down = 1).

(3) Press down one, two or all Request Stop switches to indicate NIS specified, and/or D specified.

(4) Raise the External Tree Key.

(5) Stop key to normal.

(6) The computer will Request Stop on the next instruction it reaches of the specified form.

(7) Stop key to STOP.

Post Mortem.

(1) Put Stop key to STOP.

(2) Set 0 30-0 (1) on I.S. keys and check key. (for I.S. keys level = 0, down = 1. For check key, raised = 0, level = 1, down = 2).

(3) Depress External Tree key.

(4) Single Shot.

(5) Put External Tree key level.

(6) Put Stop key to NORMAL.

(7) Call Read manually (i.e. press down the Read key on the Control Panel).

(8) Run in Post Mortem (ZP29/2).

Cards are then punched (between reading various bits of programme) which give the contents of the computer stores at the time of Post Morteming. For fuller details see the programme report.

Lining up the Monitor Display.

The R.H.S. monitor display tube shows the 32 m.c.'s, in order, of a D.L. selected by the rotary switch. The display is said to be lined up when m.c. 0 of the programme is at the top of the screen. (N.B. the position of m.c. 0 on the screen is no way affects the operation of the computer).

To line up:

(1) If the contents of some minor cycle or group of minor cycles can easily be recognised on the display tube, use rotary switch to bring the D.L. containing this pattern

onto the R.H.S. tube, identifying the pattern and press m.c. SLIP button until the pattern is in the correct position.

(2) While programme is being stored on the drum, some people find it possible to line up in D.L.11. Use the rotary switch to bring D.L.11 on the R.H.S. display tube, and press the m.c. SLIP button until the longest interval between filling consecutive rows occurs between the bottom and the top row.

16.4 BIBLIOGRAPHY.

(1) "Preparing and testing DEUCE programmes" by P.J. Landin. (Report No. NS y 80).

(2) "DEUCE Control Panel Manual" by Miss A. Birchmore. (Report No. NS y 79).

(3) "Standard Operating Instructions for DEUCE" by Miss A. Birchmore. (Report No. NS y 78).

(4) "Post Mortem" Programme Report No. ZP29/1 and 2.

P.T.A.T.O.

Programmer	Programme No. G.I.P.?	I/J	Estimated Time.	Actual Time.	Number of Output Cards.

Step No.	Action Operator.	What should happen.	What does happen	Next step.	
				✓	×
1.	Initial Input Pack A	Stops on 1 12-1	✓	2	9
2	Single shot	Punches 5 cards	✓	3	9
		Stops on 6 21-2			
3.	Set Request stop on 1 5-16, Single shot	5 min. calculation then stops on 1 5-16 request	✓	4	9
4	External Tree key level	Stops on 3 0-14 with Read Called	✓ ✓	5	9
5	Run in Pack B	Reads all cards, stops on 4 9-16	✓	6	9
6	Programme Display	not more than 20 PD cards	Stops on 1 12-1	7	9
		punched and then stops 3 12-14			
7	Single Shot	Stops on 1 12-1		8	9
8	Post mortem (high speed store		✓	END	END
	and drum to track 8/0				
9.	If in a loop P.D for about				
	10 cards.				

LECTURE 17

DEUCE CONTROL PANEL.

17.1 INTRODUCTION.

At first sight, the DEUCE control panel appears as a bewildering set of lights, switches and keys, and is quite sufficient to persuade most people that it is impossible for them ever to get to understand it, let alone use it in a competent manner.

It should be understood, however, right from the start, that many good programmers with years of experience have never fully understood the functions of a lot of the keys on the panel.- because it is possible to test programmes without this knowledge. The panel is designed to assist all types of computer personnel - programmers, operators and engineers - and therefore these are parts which are of special interest to one person but need not exist as far as others are concerned.

For the purposes of this lecture, the panel will be dealt with in three distinct sections:-

(a) Those parts essential to computer operation, which all should know.

(b) Those facilities designed to help the programmer to test his programmes, which it is advisable to know.

(c) Those extra facilities which are used occasionally, but are by no means essential to efficient programme testing.

The card reader and card panel each have a few controls on them, and these will be included in this lecture.

Not all DEUCE control panels are identical - here we will consider the latest and most complicated panel, and point out the differences as we go along.

17.2 ESSENTIAL PARTS.

Before any programme can be fed into the computer, certain points must be checked. They are:-

(a) Switches.

All the keys and switches on the panel should be in the UP position (i.e. the metal handled switches) or the CENTRE position (i.e. the 3-position plastic handled keys) except for the seventh key from the left in the top row. This key (usually called the Stop Key) is marked NORMAL, STOP, AUG. STOP, from top to bottom, and controls the speed of the computer. NORMAL is for full speed and STOP for testing, at which point every instruction becomes a STOPPER. This key should be UP at the NORMAL position before starting to read a programme. This is the ideal state, although several switches need not in fact be UP. However, for beginners it is better to do too much than to miss an essential point .

(b) Lights.

There should be no RED or ORANGE lights showing on the panel, except in the Magnetic Tape panel on the top right. These may be ignored (unless of course the programme requires tape).

The red lights that may be on are:

(i) Above the STOP key. Nothing can be done by the computer whilst this lamp is alight. It can be cleared by pressing the RELEASE key to the left of the STOP key.

(ii) Several immediately to the right of the dial at the top of the panel. If any of these are on, report to the maintenance engineer.

(iii) One above the ALARM key in the second row from the top. This is cleared by depressing the key below the light. This is the only lamp cleared by depressing a key; in all other cases the key is raised.

The orange lights that may be on are:-

(i) Above the PROGRAMME DISPLAY KEY in the top row (fifth from left). This is cleared by raising the programme display key.

(ii) Above the PUNCH key on the second row. This also is cleared by raising the corresponding key.

(iii) Above the READ key on the second row. This can be cleared by raising the corresponding key, but, since the first operation on starting a new programme is to read the instruction,S this is usually omitted as unnecessary.

There are also two sets of 32 white lights near the bottom of the main panel. These can be extinguished by pressing the keys immediately to the right of each row.

(c) Punch.

A supply of blank cards should be in position in the feed hopper (face down, Y row leading) and the RUN IN PUNCH key on the punch housing should be pressed. This makes the punch ready for use.

An automatic card numbering device is fitted to the punch, and the numbers should be set to ZERO by pressing the key beside the numbering unit. There are also 8 manual switches to punch identifying numbers on the cards. These should be set to the required number.

The computer is now in a position to accept a new programme. The complete pack of cards should be placed in the feed of the reader with the Y row of the first card nearest to the reading station. The INITIAL INPUT key is now pressed, causing three separate operations:-

(a) The mercury store is cleared of all instructions left by the previous programme.

(b) The cards are RUN IN to the reader (this brings the Y row of the leading card into position just before the reading station).

(c) When the key is released the READ trigger is called, to start the computer reading instructions from the cards and obeying them.

(d) End of Programme.

When the programme reaches its conclusion, all lights on the panel will remain in a steady state, and both reader and punch will be stopped assuming, of course, that the programme is fully tested . The RUN OUT key on the PUNCH should now be pressed and released, and the punch will turn over for two (or three for some machines) card cycles to ensure that all punched results are in the PUNCH STACKER. They are then removed, and the completely blank cards on the front of the pack are discarded - the card numbering will show which is the first result card.

The programme is removed from the READ stacker and the READ hopper should be checked to ensure that all cards have in fact been read. If not, the RUN OUT READ key should be pressed, causing any unread cards to pass into the stacker, from which they can be extracted.

(e) General Warning.

The centre portion of the cards is rather critical when feeding into punched card machinery, and cards should NEVER be held by an elastic band around the centre. Two bands should be used, one at each end, and the tension should not cause the cards to buckle. As far as possible, cards should be kept in trays and the spring clip pushed up tight to prevent distortion.

NO CARD HAVING ANY PUNCHING WHATEVER should be left in the vicinity of the computer.

17.3 AIDS TO PROGRAMME TESTING.

(a) Instruction Staticiser (I.S.)

Half way down the panel, and on the left hand side, there is a row of switches, with a lamp above each switch. These are divided into groups:-

(i) P_1 - not always fitted.
(ii) P_{2-4} - the N.I.S. section of an instruction.
(iii) P_{5-9} - the source section of an instruction.
(iv) P_{10-14} - the destination section of an instruction.
(v) P_{15-16} - the characteristic section. One switch is often used for characteristic. and the lamp is not always fitted.

The I.S. lamps always show the instruction that is at present in CONTROL waiting to be obeyed. Consequently, if the STOP key is set at STOP, making each instruction into a stopper, the I.S. lamps will show N.I.S., S and D of the next instruction to be obeyed. This should agree with the flow diagram. If the computer is told to obey this instruction by giving a "single shot" (see 17.3 (b) below) the instruction on the I.S. lamps is obeyed and the next instruction takes its place. Hence, the flow of instructions can be checked against the flow diagram, and any errors of coding or punching detected.

(b) The Single Shot Key.

This is located immediately on the right hand side of the STOP key. Every time the key is depressed, a "single-shot" is emitted, causing the machine to obey the instruction at present in control.

If the key is raised, a succession of single shots is given, allowing the programme to be obeyed at a very much reduced rate, and enabling the operator to check visually on certain aspects of the programme (i.e. are successive results being stored in successive minor cycles of a long D.L. as they are calculated).

On some machines, a second and duplicate single shot key is fitted near the bottom right hand side of the main panel.

(c) Programme Display.

It is probably already apparent that to test a programme by looking at each instruction independently on the I.S. lamps is very wasteful in computer time, and subject to error. A built-in facility of DEUCE is the ability to punch out complete instruction in the order in which they are obeyed, and as they are obeyed. They are punched in exactly the same form as they are obeyed, and therefore modifications can be checked using programme display.

The rules for Programme Display are:-

(i) Put the STOP key to its bottom position - marked AUG. STOP.

(ii) Put Programme Display cards into the punch and run them in. Programme Display cards are ruled as instruction cards, to make interpretation easy.

(iii) Press the key marked PROGRAMME DISPLAY, the fifth from the left on the top row.

(iv) Note that Programme Display should not be started on an 80-column DEUCE if an 80-column read or punch instruction has been obeyed, but the operation is not yet completed, as incorrect operation will occur under these circumstances.

(v) If a stopped instruction enters control during programme display, the punch will stop and programme display is cleared. Indication of what has occurred is provided by the RED lamp over the STOP key, which will go on under these circumstances. Before anything more can be done, the RELEASE key to the left of the STOP key should be pressed to cancel the red light. Note that the stopped instruction will be left on the I.S. lights, as it is still in control awaiting a single shot.

(vi) If a 9-24 instruction is obeyed during Programme Display, the punch will be cleared and therefore will stop. The Programme Display key should be raised and then pressed again to restart. Indication of this fault is that the orange lamp above the programme display key remains ON, but the lamp above the PUNCH key goes OUT. It should be noted that the instruction following the 9-24 instruction is punched out before the punch stops, due to the organisation of the circuitry.

(vii) Programme Display may be stopped at any time by raising the Programme Display key. This is required if the programme is in a loop having no exit (as often happens during testing). Always have an idea of how many cards will be required (12 instructions per card) to Programme Display any section of programme.

(d) Request Stop.

A facility exists to enable a programmer to stop his programme as soon as it reaches a specific instruction, but without requiring any alterations to his programme cards. This is called Request Stop, and the only action required from the operator is that he shall define the instruction at which he wishes to stop.

The first step in using Request Stop is to decide which parts of the instruction word shall be used to define the instruction. The possibilities are:-

(i) The P_1 digit (on certain machines only).
(ii) The N.I.S. group.
(iii) The source group.
(iv) The Destination group.
(v) The Characteristic (on certain machines only).

Switches are provided on a Request Stop panel, located on the left hand side of the panel (or to the right of the I.S. keys on some older machines) to select one or more of the above groups in any combination. Only those sections selected by these switches will be considered for Request Stop purposes.

The second step is to set on the I.S. keys the value of each group selected, putting the key down for a ONE and UP for a zero in the binary form. When the instruction is reached, the switch below each I.S. lamp that is ON will be DOWN for the groups selected at stage one

The conditions are now set up to stop at the required point, but a third step is required before the Request Stop is operative. These first two steps can be performed before initial input of the programme, as all the keys concerned so far come within the provision at the end of section 17.2 (a), but step 3 must wait until after initial input.

The Request Stop becomes operative as soon as the White key immediately to the right of the I.S. keys is RAISED. DO NOT PUSH IT DOWN . Now, every instruction entering control is compared with the setting of the switches, and if each and every one of the selected groups agree, the instruction is retained in control unobeyed (It should be remembered that certain instructions will be obeyed over and over again if a Request Stop is called on them. These are the trigger (D24) and the AIM instructions 17-0 and 18-0. Some machines are modified to remove this effect, but it is recommended that beginners do not use these instructions for request stop. Note that if the source only is selected to stop on any instruction having source 1,it is possible to stop on 1-24 and upset the calculation. For this reason, it is always advisable to select as many groups as possible to reduce or eliminate this possibility).

Once the computer has stopped on a Request Stop, it will not proceed until the interlock is removed by switching off the Request Stop. Before this is done, the machine should be set to STOP, otherwise the programme will continue at top speed and the effort will be wasted. With the machine on STOP, the white key is reset to its centre position and the computer moves to the next instruction and stops there. The next step may now be decided at leisure.

To sum up Request Stop, the steps are:-

(i) Select required groups.

(ii) Set value of each of the required groups.

(iii) Raise the white key.

(iv) Wait for the computer to stop.

(v) Put the machine on STOP.

(vi) Reset the white key to its central position.

(e) Post Mortem.

A programme called POST MORTEM (ZP29) has been designed to punch out as much as possible of the contents of the machine.

To start this programme, it is necessary to get the instruction 0-0X into control, with no disturbance to the rest of the store.

To do this, we use another facility built into the Control Panel, called the "External Tree". This uses the same I.S. switches that are used for Request Stop, but in a different way. The white key located to the right of the I.S. keys (which is raised to make Request Stop operative) has a third position (down) to connect the External Tree. With this key down, the computer will ignore the N.I.S. Source, Destination and Characteristic sections of the instruction in Control, and obey the instruction set up on the I.S. switches.

Therefore, to start POST MORTEM, by sending the instruction 0-0 X into control, we have to use the External Tree facility. Source 30 provides a suitable instruction to go into control, and therefore we need to set as the instruction 30-0 (1) on the I.S. keys (Note that the characteristic is long. Since it is quite probable that the instruction in control has unequal wait and timing numbers, the long characteristic is therefore essential to ensure that the transfer to Destination 0 does in fact take place).

Having set the I.S. keys to 30-0 (1) and put the External Tree on by depressing the white key, a single shot will send the required instruction into control. It is now necessary to press the READ key to call the reader, before the POST MORTEM programme can be obeyed, and also to press the RUN IN key to move the first card into the reading position.

17.4 ADDITIONAL FACILITIES.

There are several useful keys on the Control Panel to assist in testing a programme, especially when errors have been detected. Among them are:-

(a) The Monitor Tubes.

These are two cathode ray tubes on the lower right hand panel, which given a visual display of the contents of the high speed store. The left hand one shows, from top to bottom:-

TS13
TS14
TS15
TS16
DS19
DS20
DS21
QS17
QS18

The right hand tube shows the 32 minor cycles of any one of the long delay lines, or the contents of the Control register, or the contents of the auxilliary buffer store for magnetic tape (if fitted). A multi-position rotary switch below the tube controls which section of the store is displayed.

The construction of the machine is such that the minor cycle called m.c. 0 by one programme is not of necessity that called m.c. 0 by the previous programme In fact, any even m.c. may be selected . A button between the twotubes is provided to move the complete display down two minor cycles, so that, whatever minor cycle is called 0, it can be moved to the top.

The monitor tubes should not be used to check each individual instruction - a look to see if the correct result is obtained at a few stragecally placed points is all that can be done without wasting large amounts of computer time.

(b) The Discrimination Key.

It is often the case that, in programme testing, the condition arises in which the computer goes round and round a loop and never comes out, due to some error in the calculation.

Since the usual way out of a loop is by means of a discrimination instruction, it is normally possible to get out of a loop if the discrimination can be forced. The discrimination key (the fourth from the left on the top row) allows the programmer to do just this. With the key in its normal central position, the computer decides which path to take at all discriminations. If the key is raised, the computer is overuled, and all discriminations will be forced to take the early path (i.e. the zero or positive path) whatever the condition of the pertinent store. If the key is pressed down, the delayed path is taken.

(c) The T.I.L. key.

Immediately to the left of the Discrimination key is the TIL key. This operates in a similar manner to the discrim. key (i.e. with the key at central, the computer makes up its own mind: with the key UP, T.I.L. is always considered OFF, with the key DOWN, T.I.L. is always considered ON.

(d) The Cont. T.T. Key.

The second key from the left on the top row, the Cont. T.T. key, is of occasional use. If an instruction is set on the external tree (as described in 17.3 (e)) it can either be obeyed using a single shot, causing the instruction in control to be over-ridden or it can be obeyed using CONT. T.T. by pressing the CONT. T.T. key.

There are certain limitations to the use of CONT. T.T. Firstly, the transfer continues for as long as the key is pressed, so it is impossible to transfer one word from a multi-word tank, but any instruction that is not definitely tied to a particular minor cycle may be obeyed by this method. The second restriction is that no transfer to destination 24 will operate by this means - a single shot is essential for all triggers.

A typical use for cont. T.T. is to fetch a track down from the magnetic drum into the high speed store for inspection. To fetch track 13/7 into D.L.10 the sequence is:-

Set I.S. keys to 13-31 (s)
Put External Tree ON.
Press and release the cont. T.T. key i.e. obey 13-31 (s) .
Set I.S. keys to 7-30 (s).
Press and release the cont. T.T. key i.e. obey 7-30 (S) .
Set I.S. keys to 11-10 (the characteristic here is immaterial since cont. T.T. operates for as long as the key is depressed).
Press and release cont. T.T. key i.e. obey 11-10 continuously .
Put External Tree off.

N.B. The instruction in control (and displayed on the I.S. lamps) will not be changed by these operations, unless the instruction had a double characteristic, in which case the 'double' will be forgotten when the instruction is finally obeyed.

(e) The I.D. Keys.

Near the bottom of the panel there is a row of 32 lamps with a key below each. This is the I.D. (input dynamiciser) and source 0 refers to this as long as the reader is not called. If a switch is depressed, the corresponding lamp is lit; if the switch is raised, the lamp is extinguished. The lampsthat are lit indicate the ones in the word obtained from source 0.

It is not recommended that the I.D. keys should be used to affect the course of a programme, except when one wants the programme to punch out additional information. The reason for this is that no operator can be certain that the right value was set on the I.D. at the correct time, which may cast doubt upon the validity of results.

A typical use is for the case of a programme going round a loop until some condition is satisfied. Should the results fail to converge, the loop will persist indefinitely, so a test of the I.D. could be incorporated such that when the loop appears, a digit is put on the I.D. which could cause the machine to punch out sufficient information to show what is causing the trouble.

(f) The O.P.S. Lights.

Normally, destination 29 refers to the punch. If the punch is not called, however, anything sent to destination29 will appear on the O.P.S. lights, which are located immediately above the I.D. lights.

Typical uses of the O.P.S. are to display a marker in long programmes to show how far the calculation has progressed, or to show up an error should some check on the calculation fail.

17.5 CONCLUSION.

This lecture does not claim to deal with each and every facility on the Control Panel, but it is hoped that it gives sufficient guidance for the average programmer to attain a standard of competence with the keys to satisfy his personal wishes. Remember, that nothing that a programmer does on the keys can do worse than spoil the programme - it is impossible to upset the computer permanently.

The more advanced uses of the panel all result as a combination of one or more of the techniques described, together with suitable instructions for the computer to obey - the hardest part is to decide what instructions are required to fit the case: the actual operation is then a routine operation. Even when controlled manually from the panel, the computer is still obeying standard form instructions and following the normal rules. Therefore the problem of using the keys is a logical extension of normal DEUCE programming following the same rules - all that is required is a plan of action together with care in execution (with a bit of practice to gain confidence).

18

SPECIAL PROGRAMMING TECHNIQUES.

18.1 INTRODUCTION.

The aim of this lecture is to show a number of programming devices which while not being necessarily essential tools for the programmer will help him to make more efficient programmes.

18.2 SCALING FACTORS.

Most read routines for a 64 column DEUCE treat all numbers read as integers, and most punch routines require the numbers which are to be punched to be numerically less than 10. Since it may not be practicable in a particular problem to operate on all numbers in integral form or as numbers less than 10 some method of scaling will be required to convert the input data into the form required for the main programme and to convert this working data into a form suitable for output. Although there are a number of routines which carry out this scaling the programmer may find them somewhat cumbersome to use and may prefer to do the scaling himself.

Supposing we wish to read into the machine the number 7.6359. The DEUCE will read this as 76359. If we merely divided this number by 10^4 the result would be invalid since the result of dividing a by b in the machine is $\frac{a}{b} . 2^{31}$.

However, since we will have already decided on the number of binary places (p) we require in the number (and this will be such that the number is single length) we can divide the input data by $10^4 \text{ x } 2^{(31-p)}$ and we can be sure that the answer is valid. Thus, in this example, if we wished to store 7.6359 with 28 b.p. we would divide 76359 by $10^4 \text{ x } 2^3$ so that the resulting operation would be $76359 \div (10^4 \text{ x } 2^3) = \frac{76359 \text{ x } 2^{31}}{10^4 \text{ x } 2^3} = 7.6359 \text{ x } 2^{28}$ = 76359 to 28 b.p.

Supposing now that we have the number 763.59 stored in the machine to 15 b.p. and wish to punch it out as a decimal number. We would select a punch subroutine for the problem we are dealing with (e.g. punch 4 numbers per card) and in nearly all cases we would find that the number must be scaled to be numerically less than 10 and to have not more than 31 and not less than 16 b.p. Following a similar argument as before we would see that we must divide 763.59 by 10^2 so that it is numerically less than 10 but this is clearly not sufficient since we must end up with a number which has a number of binary places lying between 16 and 31.

Supposing we wish to scale numbers so that they have 25 b.p. then the calculation would be

$$763.59 \text{ (15 b.p.)} \div 10^2 \text{ x } 2^q = 7.6359 \text{ (25 b.p.)}$$

and q would be

$$31 + 15 - 25 = 21 \text{ for}$$

$$763.59 \text{ (15 b.p.)} \div 10^2 \text{ x } 2^{21} = \frac{763.59 \text{ x } 2^{15} \text{ x } 2^{31}}{10^2 \text{ x } 2^{21}} = 7.6359 \text{ x } 2^{25}$$

$$= 7.6359 \text{ to 25 b.p.}$$

To sum up we can say that when reading we determine the scaling factor by asking the questions:-

(a) How many d.p. are there in the data?
(b) How many b.p. do we want?

When punching, the scaling factor is determined by asking the questions:-

(a) What is the maximum size of number to be punched?
(b) How many d.p. in the number and how many b.p. are required in the scaled result.

18.3 SAVING INSTRUCTIONS.

There are many dodges for saving instructions which programmers have devised, many of which have appeared in DEUCE News. The following is a selection of the most frequently used ones which should help programmers to write more efficient programmes.

(a) Instruction Modification.

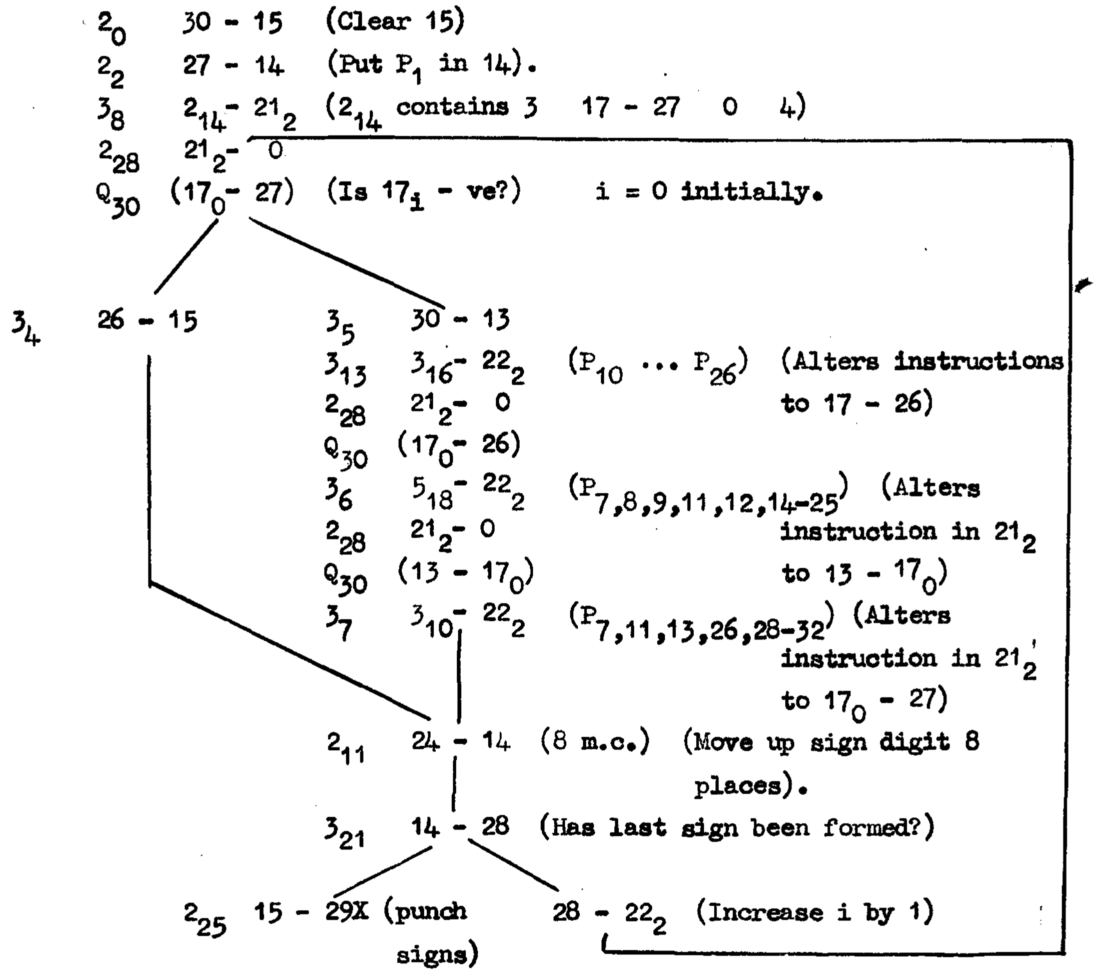
It is often advantageous to modify instructions in the long accumulator. For example if we wished to transfer successive m.c.'s of D.L.9 into 14 and after some computation has been performed transfer the contents of 14 back into successive m.c. of D.L.10 a programme of instructions might read

$6_{17,18} - 21_{3,2}$ A
$21_2 - 0$
$Q_{30}\ (9 - 14)$
⋮
$21_3 - 0$
$Q_{29}(14 - 10)$
$28 - 22$ (2 m.c.) B

This has the advantage that in instruction A 6_{17}, 6_{18} which contain the instructions 14 - 10 and 9 - 14 are transferred to $21_{3,2}$ using only one instruction and in B the two instructions are modified, again using only one instruction.

(b) Modifying Instructions by Adding Complex Patterns.

Sometimes instructions can be saved by modifying an instruction a number of times with rather complex patterns. This is best illustrated by an example and a useful one occurs in Subroutine P04 which punches 4 decimal numbers, the binary data having been stored in 17_{0-3}. This section of the subroutine puts punchings on the Y row for positive numbers and on the X row for negative numbers:



2_0 $30 - 15$ (Clear 15)
2_2 $27 - 14$ (Put P_1 in 14).
3_8 $2_{14} - 21_2$ (2_{14} contains 3 17 - 27 0 4)
2_{28} $21_2 - 0$
Q_{30} $(17_0 - 27)$ (Is 17_i - ve?) i = 0 initially.

3_4 $26 - 15$

3_5 $30 - 13$
3_{13} $3_{16} - 22_2$ ($P_{10} \ldots P_{26}$) (Alters instructions to 17 - 26)
2_{28} $21_2 - 0$
Q_{30} $(17_0 - 26)$
3_6 $5_{18} - 22_2$ ($P_{7,8,9,11,12,14-25}$) (Alters instruction in 21_2 to $13 - 17_0$)
2_{28} $21_2 - 0$
Q_{30} $(13 - 17_0)$
3_7 $3_{10} - 22_2$ ($P_{7,11,13,26,28-32}$) (Alters instruction in 21_2 to $17_0 - 27$)

2_{11} $24 - 14$ (8 m.c.) (Move up sign digit 8 places).
3_{21} $14 - 28$ (Has last sign been formed?)

2_{25} $15 - 29X$ (punch signs)
$28 - 22_2$ (Increase i by 1)

By means of the patterns in 3_{16}, 3_{18} and 3_{10} the instruction 17_0 - 27 is changed into 17_0 - 26, 13 - 17_0 and back to 17_0 - 27 again so that the correct m.c. is used in every case. The arithmetic which has been performed at each stage of modification is of interest and is shown below.

	NIS	S	D	Ch	W	J	T	$P_{31,32}$
2_{14}	3	17	27				4	2
3_{16}			31	3	31	15	1	
Sum	3	17	26				6	2
3_{18}		28	22	3	31	15		
Sum	3	13	17				7	2
3_7		4	10				29	3
Sum = 2_{14}	3	17	27				4	2

(c) Instructions before Subroutines and Destination 0 Instructions.

It should be clear already that if a Destination 0 instruction with a certain quasi minor cycle is used in a programme it may be used time and time again.

Thus we can have

6_{14}	6_{16} - 13
1_{28}	13 - 0
Q_{30}	(16 - 9_0)
6_{30}	21_2 - 14

and later on in the programme

4_{12}	5_{15} - 13
1_{28}	13 - 0
Q_{30}	(10_0 - 14)
4_{15}	4_{17} - 16

But also it is clear that we can use any group of instructions ending in a destination 0 instruction many times. Thus we might have

6_{14}	6_{16} - 13	and	4_{12}	5_{15} - 13
1_{26}	28 - 25		1_{26}	28 - 25
1_{28}	13 - 0		1_{28}	13 - 0
Q_{30}	(16 - 9_0)		Q_{30}	(10_0 - 14)
6_{10}	21_2 - 14		4_{15}	4_{17} - 16

Advantage can be taken of similar ideas when entering subroutines. For example it often occurs in a programme that we wish to multiply a by b and at a later stage a by c using a subroutine. This could be written,

4_8	12_{10}- 14 (a)		3_6	12_{10}- 14	(a)
4_0	12_2 - 16 (b)	and	3_1	12_3 - 16	(c)
4_2	8_4 - 13 (link)		4_5	5_1 - 13	(link)
6_{10}	$\boxed{\text{MULT}}$		6_{10}	$\boxed{\text{MULT}}$	

But we can save an instruction here by writing

4_0	12_2 - 16	3_1	12_3 - 16
4_2	8_4 - 13	4_5	5_1 - 13
4_8	12_{10}- 14	4_8	12_{10}- 14
6_{10}	$\boxed{\text{MULT}}$	6_{10}	$\boxed{\text{MULT}}$

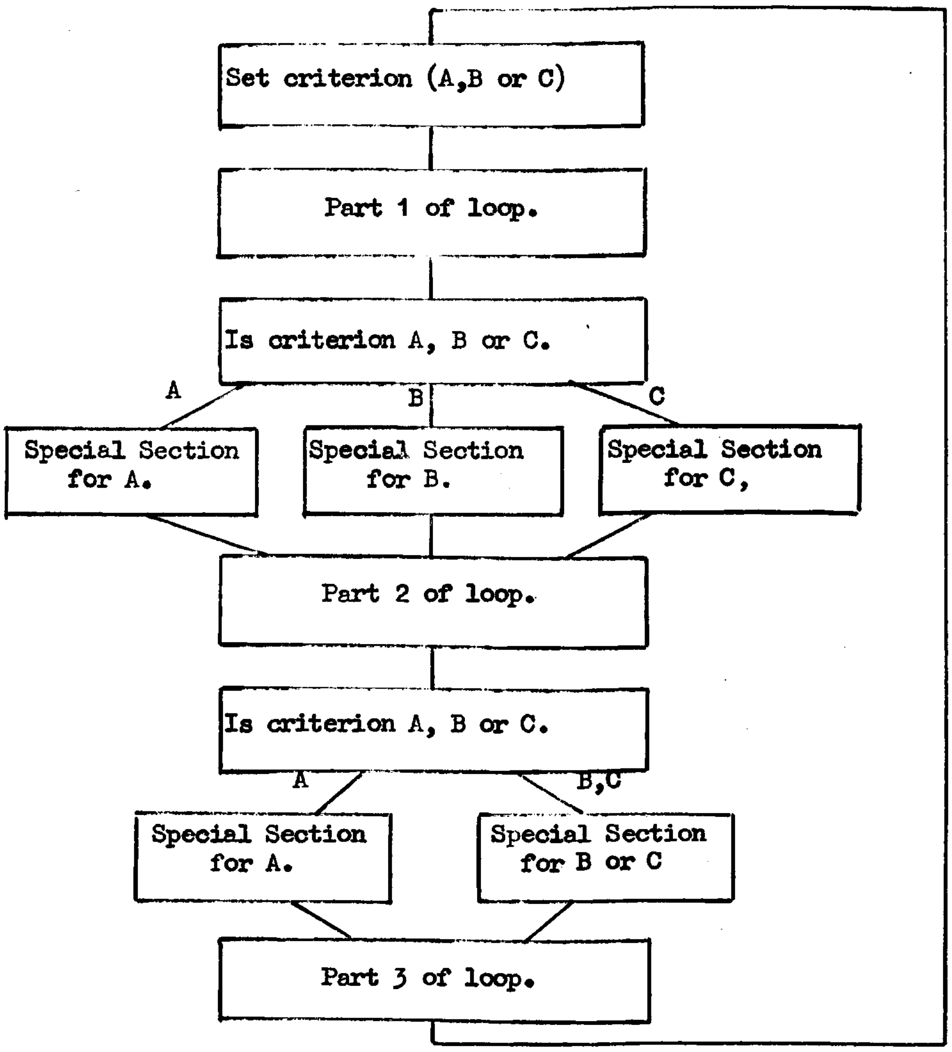
(c) DEUCE News Omnibus.

The first section of the DEUCE News Omnibus (issued 13.3.59) contains details of programming devices which have appeared in previous DEUCE News.

18.4 MARKER WORDS AND MULTI-WAY DISCRIMINATIONS.

A beginner may think that at any point of a programme it is only possible to take one of two courses of action by using the discrimination destinations 27 and 28. However, in many problems it may be necessary to take a number of different courses of action according to some criterion. This can be done in a number of ways one of which is by using what are called marker words.

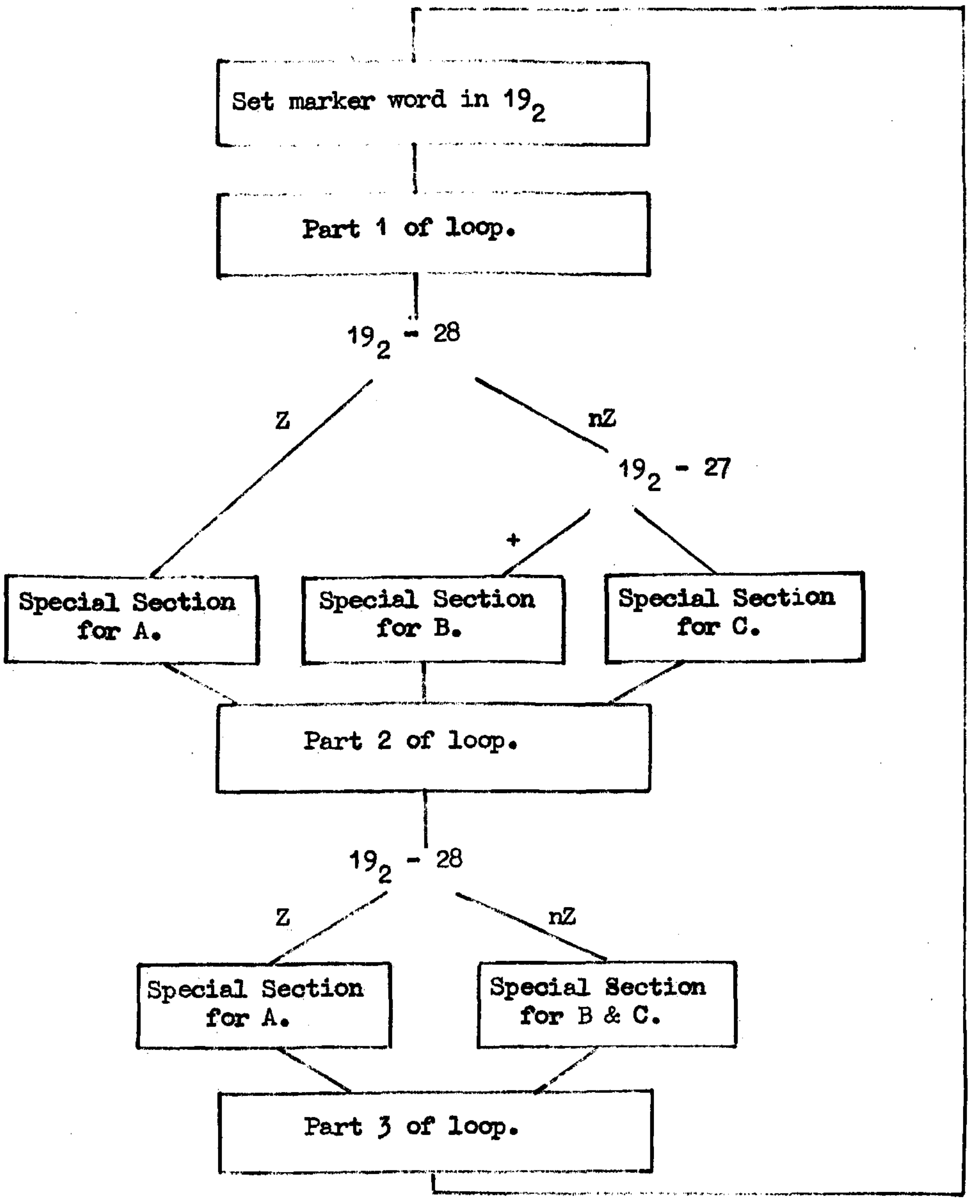
For example, if we are doing a programme which contains a major loop but requires small variations in instructions for different criteria the following condition might arise.



This could be performed by putting into 19_2 say,

Zero for criterion A.
P_{17} for criterion B.
P_{32} for criterion C.

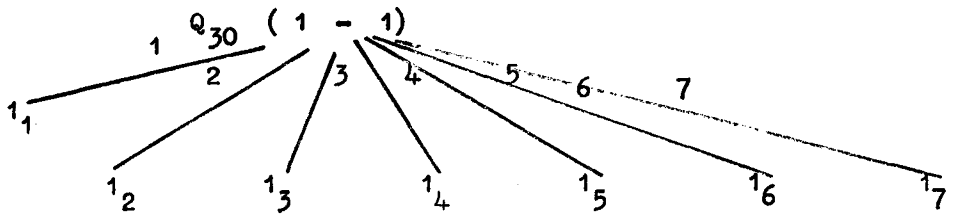
19_2 is then called a marker word and the programme would read:



An alternative method is to modify an instruction which may be a waste instruction.

For example if we wished to go to one of 7 different instructions according to one of 7 different criteria we could do this by making some store (19_2) say equal to 1, 2 ... 7 P_{26} for each of these criteria and use 19_2 to modify an instruction obeyed via destination 0, thus:

```
$4_{14}$ - 13      (1   1-1   0   0)
$19_2$ - 25        (n $P_{26}$)
13 - 0
```



18.5 USE OF TIL, I.D. AND O.S.

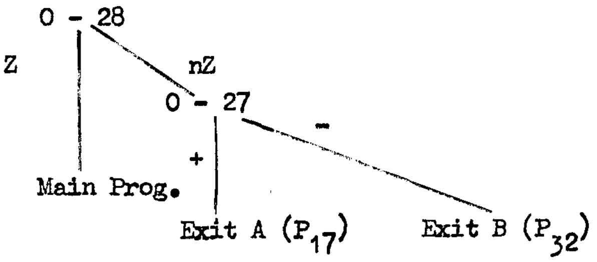
From an operational point of view the ideal programme is one which reads cards in and punches out answers and does not require any other operation.

Sometimes, however, it is difficult to achieve this ideal state and it is necessary for the operator to exercise some judgement and interfere with the programme. (Programme

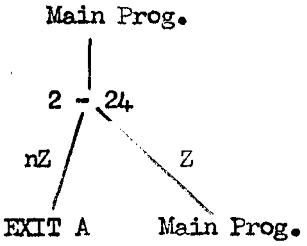
LL01 is a good example of this).

Either I.D. or T.I.L. are very useful for altering the course of a programme which may consist of a major loop.

If we wish a programme to take one of two courses of action different from the normal route whenever the operator wishes this could be done by setting P_{16} or P_{32} on the I.D. and at some portion of the loop of instructions the programme would read



A path different from the normal route could be obtained by using TIL thus,



If the operator wishes to know how a certain store in the programme is varying this can be done by sending the contents of this store to the O.S. In programme LL01 for example the successive iterates to a latent root are displayed on the O.S. If the operator wishes to speed up the convergence of the process he presses one of the I.D. keys whereas if he wishes to obey the next section of programme he depresses the TIL key.

LECTURE 19.

THE ORGANISATION OF LARGE PROGRAMMES ON DEUCE.

19.1 INTRODUCTION.

There are many advantages in writing a large programme in sections in which each section occupies the whole or part of the high speed store and some form of assembly programme is used to join the sections together.

The advantages of such a scheme are

(a) The construction of the programme is much more closely related to the logical flow diagram. In some cases one block of the logical flow diagram equals one block of the programme.

(b) It is much easier to test a programme since it may be possible to test each section independently so that when they are joined together it is simpler to find where a fault has occurred.

(c) Most assembly programmes contain a number of programme testing facilities so that it is possible to restart a programme if something has gone wrong without reading the programme in again, or else, for example, the order in which sections are obeyed may be changed by the programmer while the programme is still in the machine.

(d) In a logically complicated programme it may be virtually impossible to find ones way around the programme when something goes wrong at a later stage of programme testing unless it has been written in sections.

The disadvantage of such a scheme, however, is that no use is made of the facility on DEUCE for anticipating magnetic shifts so that the minimum amount of time is spent transferring programme from the drum to the high speed store. In many commercial programmes and in some scientific ones (e.g. Monte Carlo calculations) where a logically complicated programme has also to be extremely fast the system of breaking the programme into sections will not be suitable, although it may be desirable to write the programme in this manner initially and then to modify it to gain speed.

19.2 EXISTING ASSEMBLY PROGRAMMES.

The following programmes are the principal assembly programmes written for DEUCE.

(1) The General Interpretive Programme ZC01 (Latest version ZC01T/5.)

(2) ZC 13.

(3) ZC 14.

All these programmes have a number of common features, which impose certain conditions on the construction of sections (or bricks) of the main programme.

Each section can use any of the stores of the machine except 12_{29-31}, track 15/15 and the tracks containing programme sections or the control programme. Also each section ends by placing a parameter stating what happens next and leads out with the instructions

$$12 - 1 \text{ (32 m.c.)}$$
$$1_{30}$$

and leaves the contents of the Q.S. s (and of certain other short stores depending on the control programme used) and D.L.s except DL 11 to be found by the next section.

Another condition to be satisfied by sections is that each section is transferred to consecutive D.L.s including DL 1 before it is obeyed and it is stored on successive tracks in the drum, (the lowest DL number in the lowest track number).

The General Interpretive Programme which can be used quite generally has mainly been used for Linear algebra problems and a full discussion of this programme will be given in Lecture 23. The instructions which join sections together are written in a specially coded form and are known as G.I.P. Code words. G.I.P. contains many facilities for counting, discriminations and programme testing. At the end of a brick the G.I.P. is obeyed, which, itself, obeys the next code word. This may indicate that one of the G.I.P. facilities mentioned is to be obeyed or that the next section of programme is to be brought down into the high speed store and obeyed.

One of the drawbacks of G.I.P. is that the time taken to obey a code word is about 1 second. If each section deals with fairly lengthy calculation of a few minutes (such as the multiplication of large matrices) then the time to obey code words is not particularly important. When, however, the operating times of the sections are small the time to obey a codeword becomes important. For this reason programme ZC 14 was written for which the time to obey a codeword is only $\frac{1}{4}$ sec.. This gain in speed is obtained at the expense of some programming on the part of the user, to join the sections together. As already mentioned the instructions for joining sections together in G.I.P. are in the form of code words which are very simple to write. In ZC 14 these instructions are contained in a master programme to be written by the user. The master programme plants all the necessary parameters for a section (e.g. in the case of matrix multiplication the track numbers of the matrices to be multiplied and the product matrix), and plants a cue and a link. The cue indicates whereabouts on the drum the section is to be found and the link indicates where the next instruction in the master programme is to be found after the completion of the section.

G.I.P. and ZC 14 will be found most generally useful when the programme is chiefly made up of standard bricks which are already in the subroutine library.

ZC 13 (Programme Store and Fetch) is an organised version of subroutine B08 which is described below. It has the advantage that for very large programmes which cannot be stored on the drum all at once, it can read in programme from cards to the drum as and when required.

19.3 ASSEMBLY SUBROUTINES.

If a programmer is preparing a large programme built up of sections which are not library bricks he will probably find it simpler to join them together with an assembly subroutine. B08 which is used in ZC 13 was, until recently, the most useful routine to use. Sections joined together by B08 must obey the rules mentioned in paragraph 2 and in addition must start in 1_{30} and end with the instructions

$$\begin{array}{l} \text{I} - 16 \\ 12 - 1(32 \text{ m.c.}) \\ 1_{30} \end{array}$$

where I is of the form TP1 + DP17, T being the track No. of the highest track of the next section to be obeyed and D being the highest DL number.

B08 is stored on track 15/15, is obeyed in DL 1 and only uses the T.S. In order to enter a section at different points is is advisable for each exit of section to end with the following instructions

$$\begin{array}{l} \text{I}' - 19_2 \\ \text{I} - 16 \\ 12 - 1(32 \text{ m.c.}) \\ 1_{30} \end{array}$$

I is as above and I' is a link instruction obeyed Quasi 0 in the next section. Then if each section commences in 1_{30} with the instruction $0, 19_2 - 0, 0, 0$, entry to the section will be determined by the link planted by the previous section.

B08 has numerous useful programme testing facilities. A P_{32} on the I.D. will ensure that the machine stops before each section is obeyed. If a particular code word is set up on the I.D. the machine will stop when the corresponding section is reached. When the machine stops on a code word it is possible by manipulation of the TIL key to lead to any section desired.

These features are also included in the assembly programmes already mentioned.

In addition it is quite easy to modify B08 so that if a P_{32} is set on the I.D. the machine punches out code words instead of stopping at them. This is extremely useful in testing a very logically complicated programme since it produces a programme display of the logics of the programme.

The main disadvantage of B08 is that it is slow in operation. If for example the last magnetic read instruction in a section is from track 13/10 say then in order to bring down B08 into the high speed store the heads must be shifted to 15/15 and may be shifted again to read the programme into the high speed store.

In order to reduce this time another version of B08/1 has been written which is stored in a D.L. instead of on 15/15 and has reduced programme testing facilities.

However, another routine B14 has now been written which while lacking the programme testing facilities of B08 has many advantages in speed and compactness. As will be seen in the next section, in most large programmes some form of magnetic fetch and store routine will be necessary to fetch blocks of data from the drum and to store results back on the drum. Most of the magnetics fetch and store routines occupy half a D.L. and B14 has the advantage that as well as carrying out all the fetch programme facilities of B08 it contains a magnetic fetch and store routine and the whole of this is stored in one D.L. of the high speed store.

To fetch or store a track from the drum the number of the track x P_1 is stored in TS 14 and the routine is entered. To fetch tracks T to (T - n+1) of programme into D.L. D to (D - n+1) the code word $TP_1 + DP_{10} + (N + 16) P_{22}$ is stored in TS 14 and the routine entered.

If a P_{17} is included in the programme fetch codeword the routine can be used for filling D.L. s 1A to 7A in the case of a DEUCE Mark IIA.

In view of the importance of these two routines B08 and B14, the flow diagrams of each of them are included at the end of these lecture notes.

19.4 MAGNETIC FETCH AND STORE ROUTINES.

An extremely useful description of magnetic fetch and store routines is contained in DEUCE News 26 (pp. 20 - 26) and the following notes serve to underline the main points mentioned in this DEUCE News and to make one or two minor additions. The process by which blocks of numbers are brought from the drum to the high speed store uses a buffer D.L. (often D.L. 10). First this D.L. is filled with the first track of numbers to be fetched and while these are being transferred to the required stores the next track of numbers is transferred from the drum to D.L. 11. A similar process is used for storing numbers on the drum.

Subroutine B09 is somewhat different from the other B series of routines since it is specifically designed for fetching or storing one, two or a set of numbers up to 32 in specified m.c. s of the D.L. s or on the drum. In other words it is a means of using the drum as a random access store. One point worth noticing here is that although it is possible to transfer a pair of words in successive m.c. s if these are m.c. 31 and 0 these will be taken from the same track of the drum not from successive tracks.

If a programmer wishes to fetch and store information in blocks of a track at a time routine given in DEUCE News 22 is the best to use.

19.5 USEFUL AIDS TO PROGRAMMERS WHEN WRITING LARGE PROGRAMMES.

In a large programme it is essential to have a simple method of reading programme from cards to the drum which will also make it possible to restart the programme at any point in the case of a data error or a machine failure. In order to do this numerous aids have been devised for programmers. These may briefly be described as

(1) Clear drum.

(2) Set Clock Track.

(3) Synchronise with clock track.

(4) Read to drum.

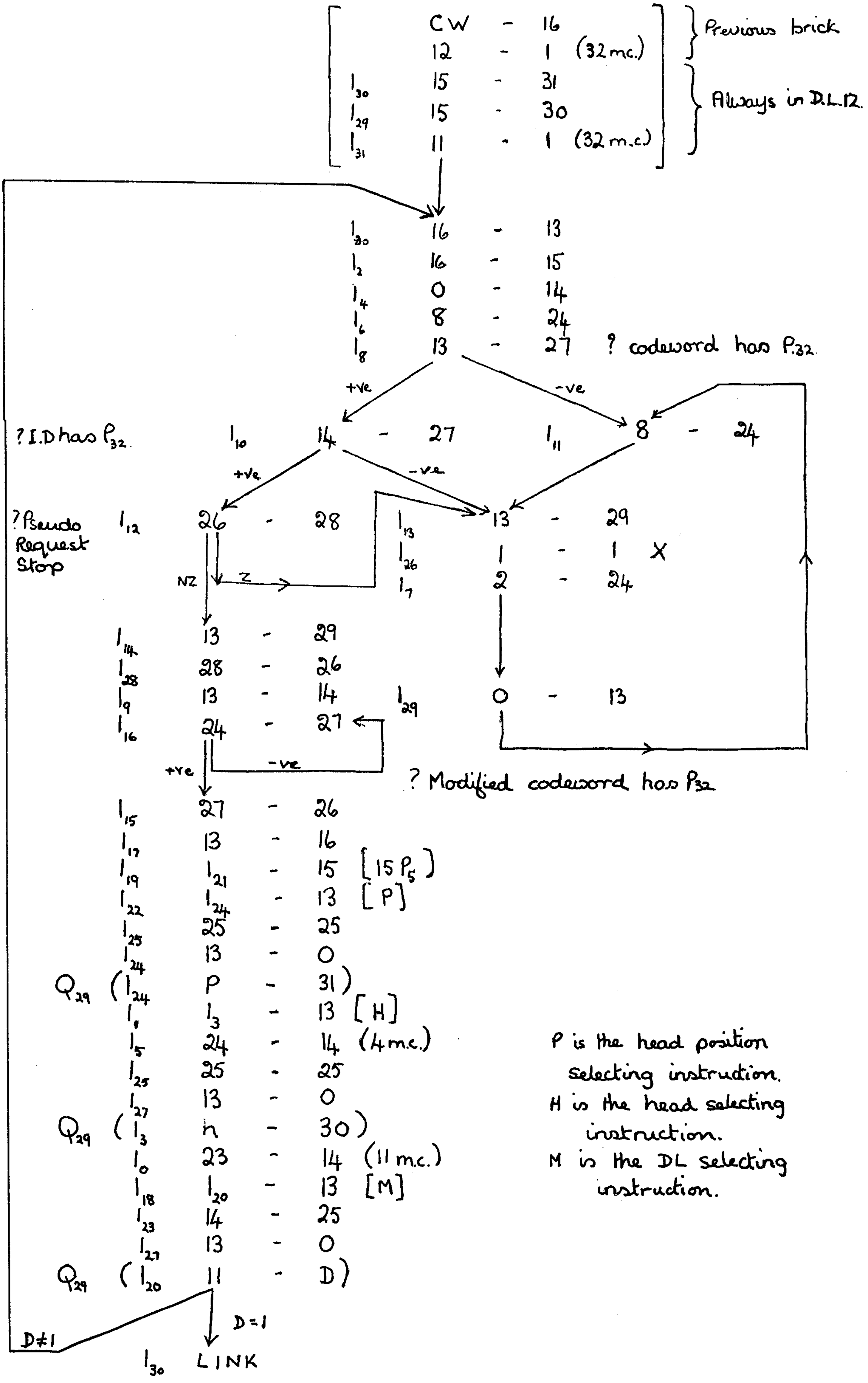
(5) Enter programme.

The purpose of Clear Drum is to ensure that before the programme is read to the drum the whole of the machine is cleared so that at any subsequent examination during the operation of the programme no information found can be due to the previous programme.

If at some point a programme is restarted there is no guarantee that the m.c.s of a track brought down from the drum correspond to those before the restart. In order to arrange this a clock track is used which is a track on the drum containing a P_{31} in a certain m.c. (the track can also be used for storing programme). The purpose of a synchronise with clock track programme is to arrange that after a restart this digit is still in the same m.c. as it was before the restart.

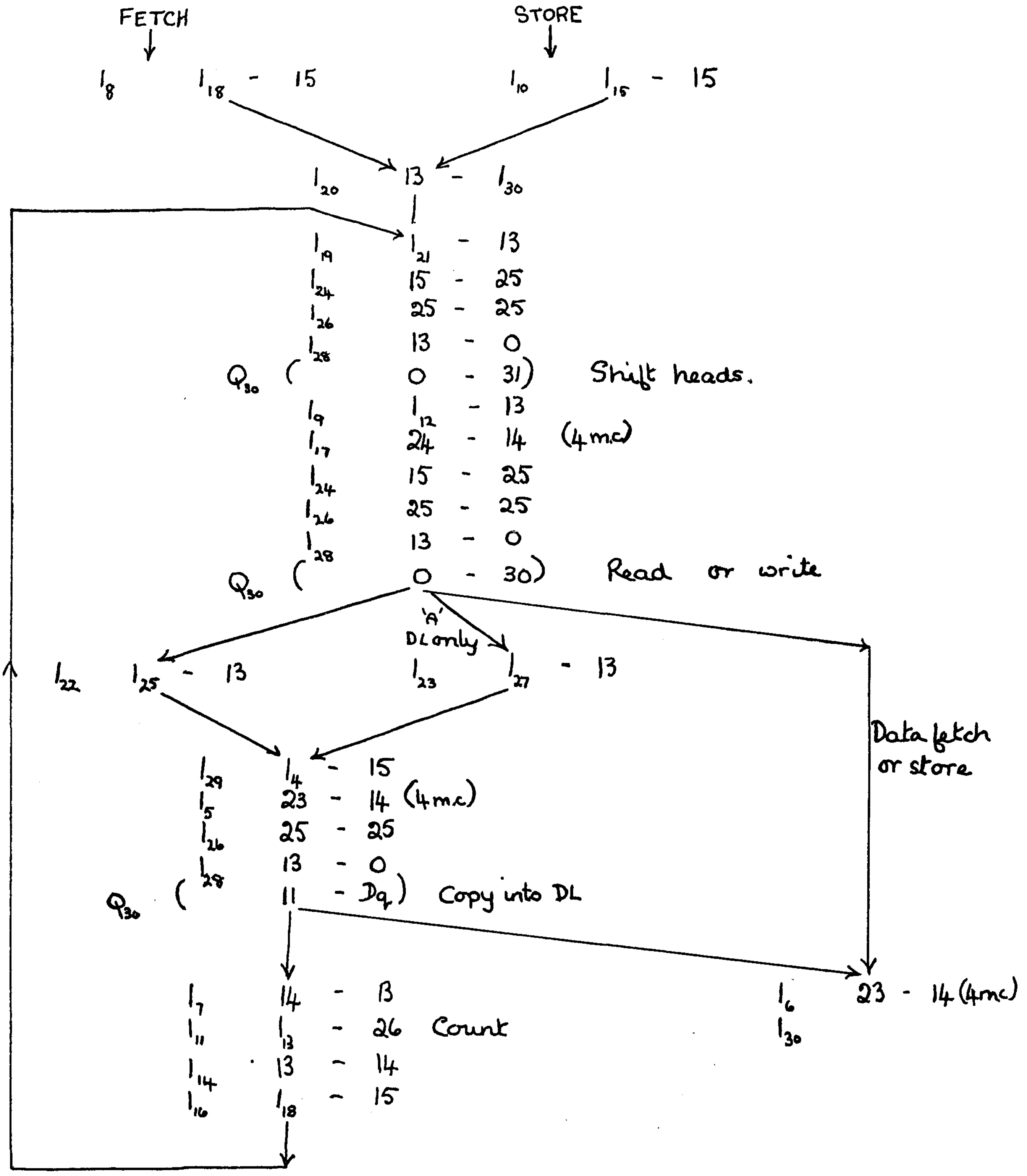
Another use of the clock track is in programme testing when a POST MORTEM is required (i.e. the output of the contents of D.L. s and non-zero tracks of the drum). The use of the clock track here ensures that m.c. 0 of a track appears as m.c. 0 on a triad of post-mortem cards.

Various programmes have been written to provide these facilities (c.f.DEUCE News 16) but an all-embracing programme which provides all these facilities, allows re-entry into the programme at any point and can be used on a DEUCE Mk 1, Mk II or Mk IIA has now been written. This routine, ZP 46T, will read triads punched either in the α or β fields or both and can read drum triads punched out by POST MORTEM (ZP29T/2) back on to the drum.



P is the head position selecting instruction.
H is the head selecting instruction.
M is the DL selecting instruction.

Programme Fetch B08



CODE: TP_1 Fetch or store to/from DL11.

$TP_1 + DP_{10} + (N+16)P_{22}$: fetch T into DL.D
(T-1) into DL(D-1)
up to (T-N+1) into DL(D-N+1)

Add P_{17} to code for aux. DL's

B14

20

DEUCE LIBRARY SERVICES

THE ENGLISH ELECTRIC COMPANY LIMITED,
DATA PROCESSING AND CONTROL SYSTEMS DIVISION,
KIDSGROVE STOKE-ON-TRENT, STAFFS.

January, 1961.

DEUCE LIBRARY SERVICES

THE ENGLISH ELECTRIC COMPANY LIMITED,
DATA PROCESSING AND CONTROL SYSTEMS DIVISION,
KIDSGROVE STOKE-ON-TRENT, STAFFS.

January, 1961.

CONTENTS

1.0 GENERAL

Since DEUCE was first introduced a few years ago a large and comprehensive library of information on its use has been accumulated. This information, the most comprehensive to be offered with any computer, is supplied free with all DEUCES, and can be made available to all DEUCE Users. The wealth of published material, on subroutines, programmes and programming techniques can save countless hours to the potential DEUCE User.

There are four main types of DEUCE programming literature.

(i) DEUCE Programme News

This publication appears at regular intervals and contains a wide variety of information of interest to DEUCE programmers. In addition to the regular issues, there are irregular special issues, each devoted to a study of one particular topic.

(ii) DEUCE Subroutines

The library subroutines (over 300 subroutines have been published up to April, 1959) consists of a wide range of programming sequences to perform all the frequently required DEUCE Operations. The potential user can therefore expect to be able to build a number of these subroutines into any programme and so save himself much programming time and be sure of using fully tested sequences of instructions.

(iii) DEUCE Programmes

The library of DEUCE programmes (nearly 500 have been published up to April, 1959) contains ready made programmes to perform all the commonly required operations on the computer. In addition to such normal DEUCE programmes, two special types may be mentioned. Firstly Standard Bricks - self contained programmes which can be easily and flexibly merged into comprehensive programmes and secondly engineers Test Programmes normally issued with every machine. The programme library also includes various automatic programming schemes and programmes designed to aid programmers.

(iv) DEUCE Information Cards

This recent innovation follows upon efforts of the DEUCE Users Association to reduce redundancy in programming and to circulate information on unpublished routines. They contain brief specifications of routines in course of preparation, and of completed routines which are not of sufficiently general interest to warrant full publication.

2.0 DETAILS OF PUBLICATIONS

Details of these four classes of publications are given below.

2.1 DEUCE News

This publication is at present issued every two months, its dates of publication being the end of every odd month (January, March, May, etc.). It contains lists of new subroutines and programmes, notes on programming devices and techniques and any other items of interest to DEUCE programmers. Contributions to DEUCE News, on any topic of interest to DEUCE programmers are always welcomed, and should be sent in by the 15th of the month of publication.

Apart from the regular issues of DEUCE News, special editions are published from time to time, each devoted to one particular topic. Among these special issues are the following -

DEUCE News No. 16 - Programmes to Aid Programmers.
DEUCE News No. 23 - Scheme B Trapezoids.
DEUCE News No. 31 - Colloquium on Automatic Programming.
DEUCE News No. 33 - Colloquium on Differential Equations.
DEUCE News No. 36 - Multiplier/Divider Techniques.
DEUCE News No. 44 - Colloquium on Curve Fitting.
DEUCE News No. 45 - Collaquium on Partial Differential Equations.

DEUCE News No. 51 - Colloquium on Linear Algebra.
DEUCE News No. 52 - Colloquium on Monte CarloMethods.
DEUCE News No. 55 - Library Lists.
DEUCE News No. 58 - Colloquium on Business Applications.

2.2 DEUCE Subroutines

The subroutines vary from the small and simple (which would nevertheless save a beginner, for example, a fair amount of effort) to the large and complicated using advanced techniques and representing many hours of an experienced programmers time. Whatever the complexity however, the library subroutine offers an efficient and fully tested method of carrying out part of a calculation. On occasions the first published version is not the best that can be produced. Some of the more important ones have therefore been re-written a number of times, incorporating the latest programming techniques. This does not invalidate the earlier versions, the later ones are simply more efficient, i.e. use fewer instructions and/or are faster.

A large proportion of the 300 library subroutines are for input of data and output of results, and subroutines now exist for reading and punching almost any card layout the programmer may choose. A full copy of a subroutine write up consists of -

(i) One or more cards (size $6\frac{1}{2}$" x 8") which contain all the information which the programmer needs to know to use the subroutine.

(ii) A report (on foolscap paper) on the subroutine, including all the information in (i) above and including also the full details of the flow-diagram and coding.

2.3 DEUCE Programmes

A DEUCE programme differs from a subroutine in that it is a self-contained entity, the punched cards for which can be fed directly to DEUCE. Programmes may be divided into the following categories.

1. Programmes for standard calculations. Among these may be mentioned those for roots of algebraic or differential equations, the simplex and transportation problems, regression analysis, auto and cross correlation, etc.

2. Standard Bricks. These are DEUCE programmes which fulfil a few special conditions which are used with the General Interpretive Programme (G.I.P.). There are over 200 bricks which in conjunction with the General Interpretive Programme form a very flexible and powerful method for solving a wide range of problems. By suitably combining the bricks a special programme can frequently be constructed in a very short time.

3. Programmes associated with the various automatic programming schemes. Apart from the General Interpretive Programme, and a number of other allied programmes, the Tabular Interpretive Programme and Alphacode and their associated programmes come in this category. These powerful programming schemes are the most frequently used DEUCE programmes.

4. "Programmers Aid" Programmes. In this category are a wide range of programmes designed specifically to assist programmers. Typical of such programmes are those to load the magnetic drum with programmes, and the 'post-mortem' routines which assist a programmer during programme testing.

5. Test Programmes. This comprehensive set of routines (for the use of maintenance engineers) is designed to test and measure margins of safety on the machine. Such programmes are issued with every machine, but are not usually of general interest.

6. The full publication of a programme consists of:

   (a) A summary card ($6\frac{1}{2}$" x 8") giving the title and brief description of the routine.

   (b) The operating instructions (usually on cards $6\frac{1}{2}$" x 8") which give all the information a machine operator needs to know about the routine, i.e. method of preparation of input data, order and number of cards to be fed to DEUCE, the expected output, and the failure instructions in the routine and instructions for the action to be taken if a failure is encountered.

   (c) The full report, which includes summary and operating instructions together with any notes on the method used, and a full flow diagram and coding where practicable.

2.4 <u>DEUCE Information Cards</u>

In an effort to eliminate duplication in programming and also to obtain and circulate more programming information, the DEUCE Users Association recently set up a committee to investigate the situation and suggest remedies. The committee found that apart from duplication of effort, a number of programmes were not being published because their authors considered them not to be of sufficiently general interest, though it may frequently happen that one or two other programmers would be interested. The outcome has been a liaison scheme which arranges for the circulation of brief information about projected routines and about completed routines which would not otherwise be published.

A brief specification is given for each programme or subroutine, together with the name and organisation of the author, in case further information is required. The information cards $6\frac{1}{2}$" x 8" in size, with details of one routine on one card.

All programmers are asked to help themselves by co-operating in the scheme.

3.0 PUNCHED CARDS

Punched cards for subroutines and programmes are normally circulated to all DEUCE establishments so that any prospective user should find the cards for his required routines in the library with the machine to which he has access.

4.0 NUMBERING SYSTEM

All subroutines and programmes are referred to, by programmers, by the code number. In the case of subroutines this consists of a category letter and a number possibly followed by a modifying letter or letters and possibly a stroke number.

e.g. (i) P13F/4. P stands for "punch" and the number 13 indicates that it is the thirteenth punch subroutine. The modifying letter F means that the subroutine deals with floating point numbers and the /4 that it is the fourth modification of the original P13F.

(ii) D01 D stands for "division" and the number 01 indicates that it is the first division subroutine. There are no modifying letters or stroke number, denoting that it works on ordinary single length binary numbers and that it is the original version.

The code number for programmes is similar except that there are two letters at the front, the main category and sub-category respectively.

A complete list of subroutines and programme categories can be seen in DEUCE News No. 35.

Full details of the numbering system are given in DEUCE News Nos. 26 and 35.

5.0 COSTS OF THE DEUCE LIBRARY SERVICE

All library literature is available on request at prices which cover the costs of printing and distributing the literature. Back copies of all literature may be ordered, issued with attractive plastic files in which reports may be stored. The cards are issued loose. A number of copies of the literature will be issued free with each DEUCE installed. For other users, however, the literature can be supplied at an annual subscription rate. The current rates are:-

	No. of free copies with a DEUCE.	Annual subscription (guineas)	Cost of * all back numbers (guineas)
DEUCE News	4	5	15
DEUCE Subroutines	4	5	25
DEUCE Programmes (excluding Test Progs. and Bricks)	2	3	20
DEUCE Bricks	4	3	15
DEUCE Test Programmes	2	-	- †
DEUCE Information Cards	4	3	-

* These costs are subject to some increase as the volume of literature increases.

† Subscription rates are not quoted as these are of interest to DEUCE owners only, to whom they are supplied free.

The code number for programmes is similar except that there are two letters at the front, the main category and sub-category respectively.

A complete list of subroutines and programme categories can be seen in DEUCE News No. 35.

Full details of the numbering system are given in DEUCE News Nos. 26 and 35.

5.0 COSTS OF THE DEUCE LIBRARY SERVICE

All library literature is available on request at prices which cover the costs of printing and distributing the literature. Back copies of all literature may be ordered, issued with attractive plastic files in which reports may be stored. The cards are issued loose. A number of copies of the literature will be issued free with each DEUCE installed. For other users, however, the literature can be supplied at an annual subscription rate. The current rates are:-

	No. of free copies with a DEUCE.	Annual subscription (guineas)	Cost of * all back numbers (guineas)
DEUCE News	4	5	15
DEUCE Subroutines	4	5	25
DEUCE Programmes (excluding Test Progs. and Bricks)	2	3	20
DEUCE Bricks	4	3	15
DEUCE Test Programmes	2	-	-†
DEUCE Information Cards	4	3	-

* These costs are subject to some increase as the volume of literature increases.

† Subscription rates are not quoted as these are of interest to DEUCE owners only, to whom they are supplied free.

It will also be possible, in certain circumstances, to obtain separate copies of certain items of literature. This facility is offered for those requiring selected copies of the manuals on Alphacode, the Tabular Interpretive Programme, and certain other documents, listed in the appendix.

All requests for any DEUCE literature shoudl be forwarded to the undermentioned address for the attention of Mr. P.J. Marriott. It should be mentioned that all the documents are copyright and may not be reproduced in whole or in part.

DEUCE Library Service,
The English Electric Company Limited,
Kidsgrove,
Stoke-on-Trnet,
Staffs.

APPENDIX.

The following documents can be purchased individually at the prices shown:

G.I.P. Manual (Scheme B etc.)		10/-
Alphacode Manual		10/-
Subroutine Manual		10/-
Tabular Interpretive Programme (T.I.P) Manual		7/6
DEUCE Magnetic Tape Manual		10/-
DEUCE Stac Programming Manual		10/-
Lists of Subroutines and Programmes		Free
DEUCE Programming Manual		£1. 1. 0.
DEUCE Logical Manual	Part I	£1.10. 6.
	Part II (Drawings)	£2. 2. 0.

2

LECTURE 21

64 COLUMN READING AND PUNCHING.

21.1 INTRODUCTION.

This is a logical extension of the 32 column operation described previously. Two separate 32 digit fields are used, card columns 17 - 48 (as used for 32 col.) being referred to as the α - field and cols. 49 - 80 for the β - field. For 32 col. working, the α - field only is used: for 64 col. working, both fields are used.

21.2 64 COLUMN READING.

The method of reading the α - field is completely unchanged from the 32 column system that is, an instruction using SOURCE 0 refers to the α - field and shall be a stopper. As soon as this instruction has been obeyed, and the next instruction enters control, the machine will automatically start to switch over to the β - field, but with the provision that the α - field will still be available at SOURCE 0 for FOUR MINOR CYCLES. Thereafter, the state of SOURCE 0 is not guaranteed until TWENTY MINOR CYCLES AFTER THE STOPPED INSTRUCTION IS REPLACED IN CONTROL. After this time, the β - field is available at SOURCE 0 up to the normal limit of SOURCE 0 availability of TWO MAJOR CYCLES after the stopped instruction.

In the simplest language, the rules are:-

(a) Read α - field with stopped instruction.

(b) If one or two more copies of the α - field are required, they must be taken immediately, using UNSTOPPED instructions with wait numbers of ZERO, and if two copies are required, the timing number of the first SHALL BE ZERO.

(c) The β - field is read using an UNSTOPPED instruction, and the wait number shall be at least 18, unless another instruction is obeyed following the stopped instruction and before the first β - field instruction. In this case, the wait number may be reduced by an amount equal to the TIMING NUMBER of the intervening instruction, and by a further TWO in view of the two minor cycles that automatically occur when any instruction is obeyed.

(d) Thereafter, copies of the β - field may be obtained up to 2 MAJOR CYCLES after the stopped instruction.

Example 1.

1_0	$0 - 14_x$	α field
1_2	$0 - 15$	W = 18 to wait for β field (obey in m.c. 22)

Example 2.

1_0	$0 - 14_x$	First α field.
1_2	$0 - 19_2$	copies of α field.
1_4	$0 - 20_2$	
1_6	$26 - 14$	
1_8	$14 - 17_2$	
1_{10}	$17 - 18_0$	
1_{12}	$21 - 18_2$	
1_{14}	$18_0 - 21$	
1_{16}	$30 - 18_2$	
1_{18}	$16 - 17_0$	
1_{20}	$0 - 14$	Earliest β field in m.c. 22

W=T=0 for all the instructions.

N.B.

The wait number of 1_{20} can be zero, as there are 9 intervening instructions between 1_0 and 1_{20}, allowing the wait number to be reduced by 18.

As is to be expected, the earliest β - field reading in this example is in the same minor cycle as in Example 1. A check to see if the β - field is yet available is to imagine a wait number of 18 on the instruction following the stopper; the normal rules (m + w + 2) give the first minor cycle of β - field availability.

Example 3.

$1_{26}\ \ 17_0 - 0\ \ + P_{17}$	
$Q_{28}\ [0 - 90x]$	Put α - fields into D.L. 9
$1_{11}\ \ 17_1 - 0\ \ + P_{17}$	
$Q_{29}\ [0 - 10_0]$	Put β - fields into D.L. 10
spill	
1_{27}	

The 17_1 - 0 instruction must not be later than m.c. 11, giving a timing number of 16 to call Q_{29}, in view of the variable wait number on 0 - 10.

21.3 64 COLUMN PUNCHING

As for single field punching, the instruction to punch the α - field must be a stopper. However, the time factor is not so important for 64 column punching, as an extra instruction is required before acess to the β - field is possible. This instruction, 8 - 24 1, will switch immediately to the β - field, and the required word can then be sent to destination 29 (using an unstopped instruction) as soon as is convenient, but not more than 4 major cycles after the stopped instruction. If the 8 - 24 1 instruction is omitted, the word intended for the β - field will be punched on top of the α - field word.

If β - field punching only is required the 8 - 24 1 instruction should be the stopper.

Example 1.

1_0	$15 - 29_x$	Punch α - field.	
1_2	8 - 241	Switch fields.	fastest possible.
1_4	16 - 291	Punch β - field.	

Example 2.

1_0	$15 - 29_x$	Punch α - field.
1_1	16 - 15	
1_2	14 - 16	
1_{31}	8 - 241	Switch.
1_{30}	15 - 29	Punch β - field in m.c. 0.

This example shows the β - field punching delayed as long as is possible.

Example 3.

1_0	8 - 241	Switch on stopped instruction.
1_2	15 - 29	Punch β - field only.

In this case, the α - field will remain unpunched.

21.4 GENERAL PROVISIONS

The normal rules for calling and de-calling the reader and punch for 64 column operation are exactly as for 32 column operation.

Since for any 64 column operation, there must be one or more unstopped instructions, such routines can never operate sucessfully with the machine on "STOP". This fact should be bourne in mind during programme testing.

LECTURE 21 (C)

DEUCE PROGRAMMING FOR DATA PROCESSING APPLICATIONS.

21C.1 INTRODUCTION.

The DEUCE Mk. I is a high speed stored programme digital computer originally designed for scientific and technical problems. Such problems involve large amounts of computation on relatively small amounts of data which, in nearly all cases, is numeric information. The input and output operations are not buffered and reading and punching is accomplished by using the appropriate DEUCE instructions. Only one conversion is needed, from decimal to binary and vice versa, and this may be done during the passage of cards through either the reader or the punch. Indeed, the only real problem on input and output arose after the DEUCE reading and punching capacities were doubled (by the introduction of the 64-column machine) to 12800 and 6400 characters per minute respectively. Some of the conversion subroutines for dealing with fully punched 64 column cards have to work fairly hard to keep pace with the reader and punch. The large number of read and punch routines in the published library is impressive evidence of the flexibility of input and output with a restricted field of 64 columns.

Data processing applications require rather more than can be provided on a DEUCE Mk. I The input and output data is likely to be alphabetic as well as numeric and all 80 columns of the card must be used if necessary. These changes of requirement have been met by the introduction of DEUCE Mk. II. The separate input and output machines have been replaced by a combined 80 column input-output machine which is capable of simultaneous dual operation. Reading and punching operations are now fully buffered and automatic. Built-in logical circuits convert each column of the card to a 6-bit character in 80 column code (Appendix I). Twenty-six of the characters in the code are allocated to the letters of the alphabet and characters punched on cards in I.B.M. 4-ZONE code are stored within DEUCE in the form shown in Appendix I. Standard numeric punching is also converted to 6-bit code and numbers on cards are represented within DEUCE in the binary-coded decimal (B.C.D.) notation. The DEUCE arithmetic and logical circuits are not designed to operate directly on numbers stored in this form and conversions from B.C.D. to Binary and back to B.C.D. are frequently occurring requirements. Such conversions can take place only after a card has been completely read into DEUCE or before a card is punched out, unlike the conversions on 64 column operation which occur during the passage of cards.

Not unnaturally these differences of input-output organisation have led to differences in programming technique. Whereas mathematical computations are rarely concerned with more than one pair of conversions from decimal to binary and back again, data processing applications are likely to call for many others, such as those from binary pence to sterling and binary pounds weight to tons, hundredweights and quarters. All these conversions are accomplished by subroutines of DEUCE instructions which, ideally, should be very fast and economical of instructions. Such subroutines should also be as comprehensive as possible and techniques are available for two way conversions between binary basic units and binary coded characters in practically any other form of unit in current use. There are difficulties however in making a completely general routine. It is rather like inventing a machine for peeling potatoes which also mows the lawn and anyone faced with tricky conversions is advised to consult the programming staff of English Electric when the full benefit of their advice and experience will be made available.

Since most programming difficulties in data-processing applications start with the punched cards themselves this topic will be a reasonable one with which to begin a detailed discussion.

21C.2 PUNCHED CARDS.

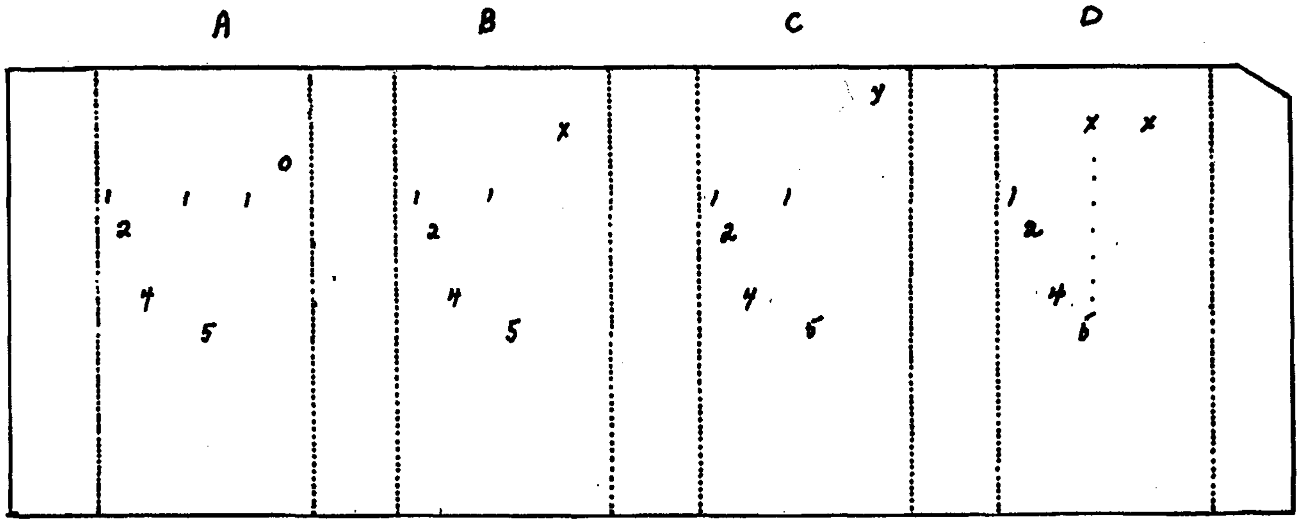
Computers use punched cards and human beings use computers. Human beings, moreover, claim the freedom to punch their own cards in any manner they choose. Not only do card layouts differ for different applications (which is reasonable) but punching conventions differ for identical forms of information (which is not as reasonable).

There are 12 rows on a standard punched card and any single column may be punched in any or all rows, giving rise to 2^{12} (4096) possible combinations of hole patterns in any column. Fortunately no one ever tries to push things quite as far as this and it is generally agreed that 64 different combinations are sufficient for most purposes. The same number of different patterns can be obtained in a 6 digit binary number and it is possible to arrange a one to one correspondence between 64 selected punch characters and 64 six-digit binary characters.

The binary numbers 0 to 9 are allocated to the conventional 0-9 punch patterns and in general the code is chosen to make things as convenient as possible. But once the code is chosen it is fixed for all time. The computer is wired in accordance with the chosen code and correspondence between information on cards outside DEUCE and the same information inside DEUCE is completely determined.

There are other things to consider however. Punched cards are used by Tabulators as well as by computers and there is also a correspondence between punched hole patterns and the characters printed by a selected type bar. Now Tabulators can be obtained with a fair degree of flexibility and any given punched card installation can choose its Tabulator to conform with its own punch conventions. On a computer however different punching conventions are met by writing different sequences of instructions in the programme and it is this which makes general subroutine construction difficult.

As an example, consider only four of the many different methods of representing sterling on a punched card. Figure I shows a card punched with £124. 15. 10d. in four different ways, A, B, C and D.



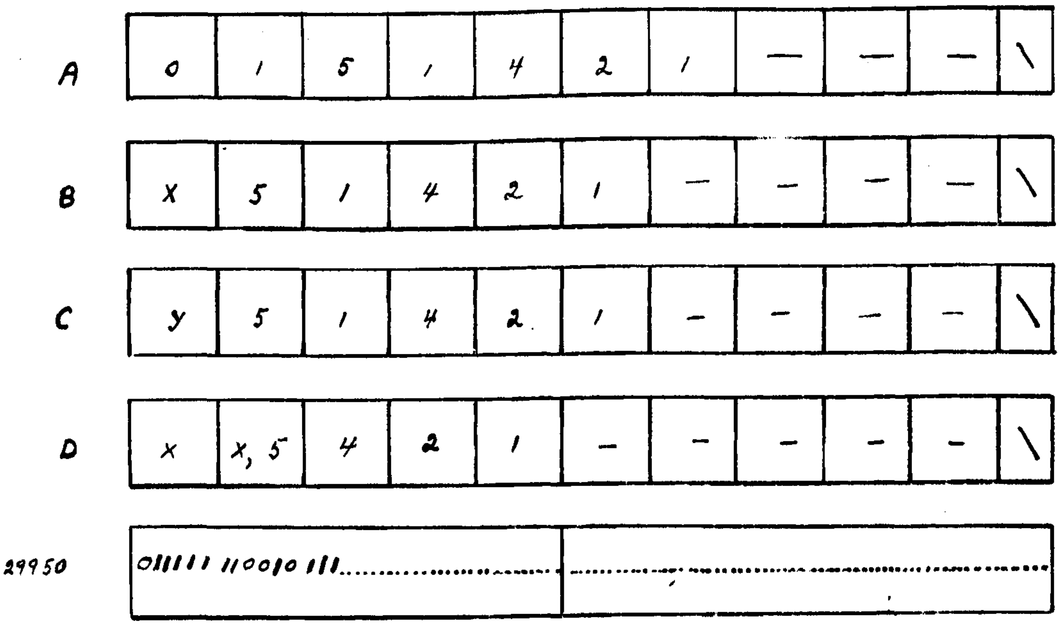
A uses 7 columns. Two columns are used for both shillings and pence.

B uses 6 columns. Two columns are used for shillings and only one for pence with the convention that X = 10^d and Y = 11^d.

C uses 6 columns. It is the same as B with the difference of convention that X = 11^d and Y = 10^d.

D uses 5 columns. Only one column is used for shillings and one for pence. In the shillings column X = 10/- and the pence convention is X = 10^d and Y = 11^d.

Figure 2 shows the characters which would appear in DEUCE corresponding to each type of punching, together with the binary number 29950 (representing pence) to which each of these must be converted before any arithmetic operations can be performed.



A	0	1	5	1	4	2	1	—	—	—	\
B	X	5	1	4	2	1	—	—	—	—	\
C	Y	5	1	4	2	1	—	—	—	—	\
D	X	X, 5	4	2	1	—	—	—	—	—	\
29950	011111 110010 111........	....									

So there we are for a start. Four different methods of punching the same thing, four different sets of DEUCE characters and all must be turned into the same binary number by programme. That is not all however. The same kind of low cunning in saving columns of cards takes place in decimal too. By using the X row for 10 and the Y row for 2C, numbers up to, say, 29999 may be punched in only four columns of a card. This is a trick which would appeal to anyone who wanted to represent up to 27 lbs. (28 lb. = 1 Qtr.) using only one column.

Wherever possible card layout and punching conventions should be chosen to simplify programming. If the card layout is determined by previous usage or for reasons of column economy these considerations take priority, but the routines for packing, unpacking and conversion are not likely to be as simple as they might be.

21C.3 THE DEUCE 80-COLUMN CODE.

The complete code is given in Appendix I but it may be necessary to predict the character which will be formed for a punching combination not given in the list. The construction of any character from a given pattern of holes in a column is accomplished by allocating values to each row as follows:

Row	Row Value	Character	Final Result
Y	16	000010	000010
X	32	000001	000001
0	48	000011	000000
1	1	100000	100000
2	2	010000	010000
3	3	110000	110000
4	4	001000	001000
5	5	101000	101000
6	6	011000	011000
7	7	111000	111000
8	8	000100	000100
9	9	100100	100100

After a card is read into DEUCE the complete array is scanned automatically and all '48' characters representing 0 Row punchings are converted to zero. Blank columns are read as zero and converted to 15 (111100). If two or more holes are punched in a column the final character is the logical sum (i.e. without carry digits) of the characters appropriate to each row. Any common digit in the separate row patterns are shared in the final character.

Examples.

```
(a)   2 Row and 8 Row.    2 Row gives 010000
                          8 Row gives 000100
                                      ------
                          Result      010100
                                      ------

(b)   2 Row and 3 Row.    2 Row gives 010000
                          3 Row gives 110000
                                      ------
                          Result      110000
                                      ------
```

Two important cases are:

```
(c)   Y row and 0 Row.    Y row gives 000010
                          0 row gives 000011
                                      ------
                          Result      000011
                                      ------
```

which is finally changed to 000000

```
(d)   X row and 0 Row.    X row gives 000001
                          0 row gives 000011
                                      ------
                          Result      000011
                                      ------
```

which is finally changed to 000000

Examples (c) and (d) explain one of the slight inconveniences of the code. Where Y and X punchings are used for positive and negative signs overpunched zeros are indistinguishable from unsigned zeros.

21C.4 THE DEUCE CHARACTER STORE.

To read a card with the 80 column machine an instruction 12-24 1 is required. This initiates the passage of one card through the reader. During continuous reading operations, (12-24 1 is needed for each card) the cards pass through the reading station at a speed of 200 cards per minute and one card cycle is thus

$$\frac{60 \times 1000}{200} = 300 \text{ milliseconds.}$$

There are twelve rows on each card (and eleven spaces between them) and the equivalent of three spaces separates one card from the next. The time to read all rows of one card is thus

$$\frac{11}{14} \times 300 \doteqdot 235 \text{ milliseconds.}$$

from the Y row to the 9's row.

Roughly this length of time elapses after the Y row passes the reading brushes before the complete card is presented in character form in D.L.12. The standard arrangement of characters in D.L.12 is shown in Fig. 3 which assumes:-

1. the use of a standard plugboard,

and 2. that the card is read into minor cycles 0-15.

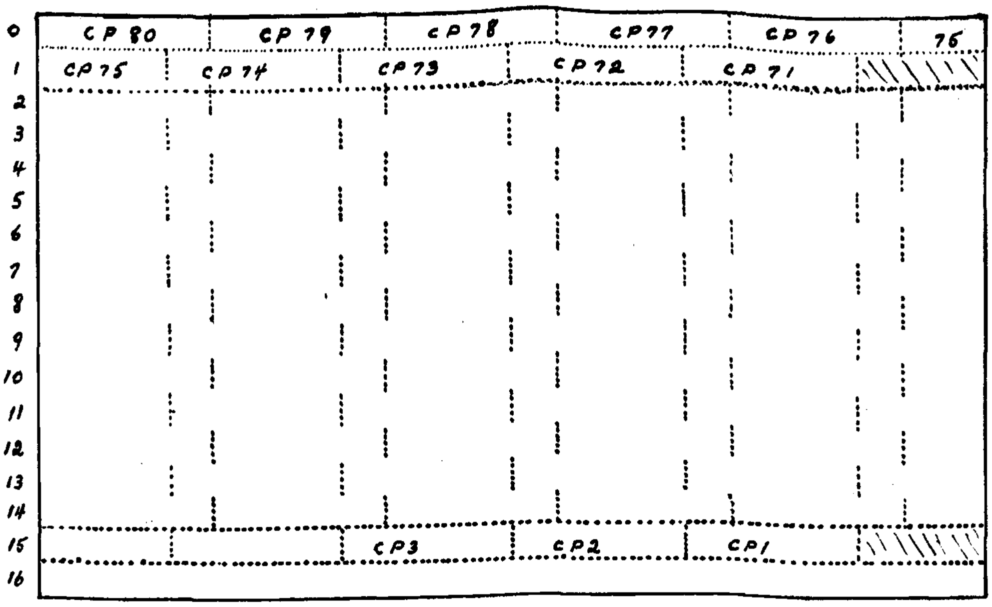
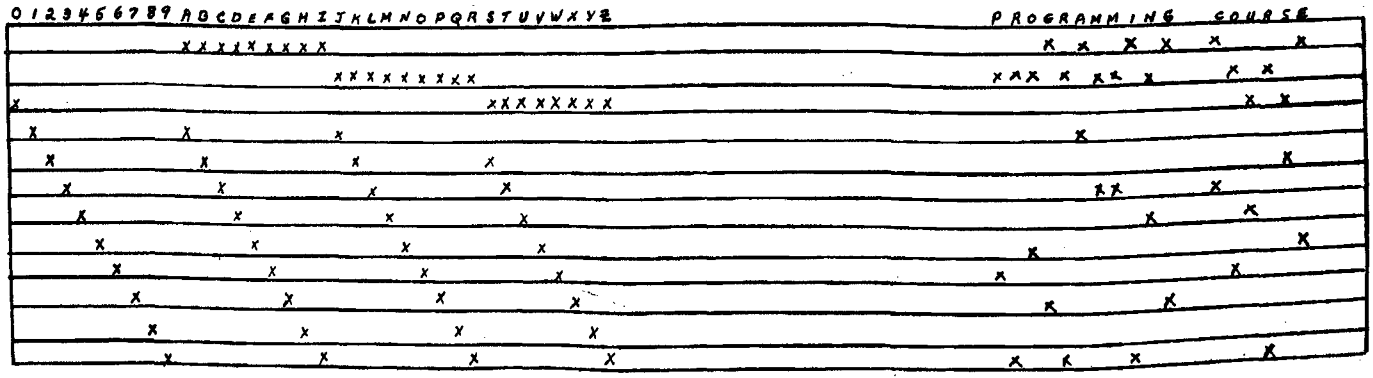
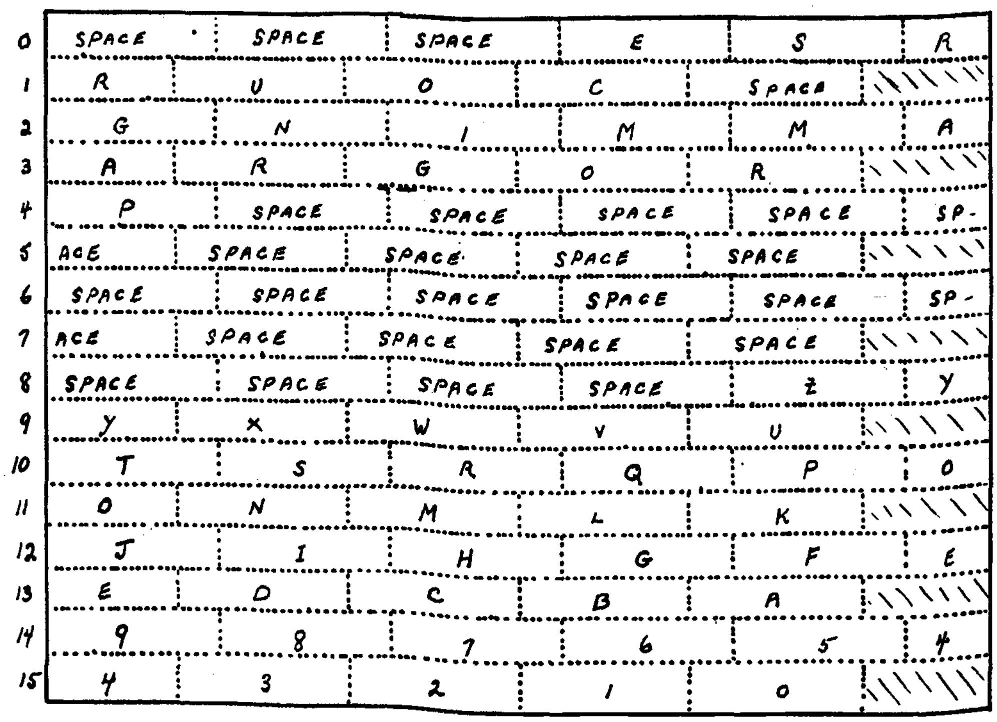


Figure 4 shows a punched card and Figure 5 shows the arrangement of characters appropriate to it. Each character is indicated in its correct position and denoted by the numeric value of the character as well as by an alpha numeric symbol.


0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
PROGRAMMING COURSE


0 SPACE · SPACE SPACE E S R
1 R U O C SPACE
2 G N I M M A
3 A R G O R
4 P SPACE SPACE SPACE SPACE SP-
5 ACE SPACE SPACE SPACE SPACE
6 SPACE SPACE SPACE SPACE SPACE SP-
7 ACE SPACE SPACE SPACE SPACE
8 SPACE SPACE SPACE SPACE Z Y
9 Y X W V U
10 T S R Q P O
11 O N M L K
12 J I H G F E
13 E D C B A
14 9 8 7 6 5 4
15 4 3 2 1 0

21C.5 TERMS USED IN 80 COLUMN PROGRAMMING.

Read Store (R.S.)

16 consecutive minor cycles of D.L.12 into which the eighty columns of a card are read following an instruction 12-24 1. The wait number and instruction minor cycle of either 12-24 1 or 10-24 1, whichever is first given, determines the READ STORE.

Standard Read Store.

Minor cycles 0 to 15 of D.L.12.

Standard Character Positions. (S.C.P.)

The character positions in a standard read store with CP80 in P_{1-6} of m.c. 0 and all character positions following in descending sequence to CP1 in P_{22-27} of m.c. 15 as shown in Fig. 3.

Read Store Locations. (R.S.L.)

Some programmers prefer to define a character position as a READ STORE LOCATION, with R.S.L.'s numbered from 1 to 80 starting at R.S.L. 1 in P_{1-6} of m.c. 0.

i.e. R.S.L. 1 CP80
R.S.L. 2 CP79
and so on.

Standard Plugboard.

A plugboard which causes all columns of a card to be read into the correspondingly numbered character positions i.e. such that

Col. 80 is read into CP80
Col. 79 is read into CP79
and so on.

Pivots.

It is convenient to have some term for the least significant character position in each minor cycle. CP's 80, 75, 70, 65, 60, 55, 50, 45, 40, 35, 30, 25, 20, 15, 10, 5 are defined as the PIVOTS and CP's 80, 70, 60, 50, 40, 30, 20, 10 are defined as the EVEN PIVOT. The even pivots are the bottom character positions in each word pair.

Punch Store.

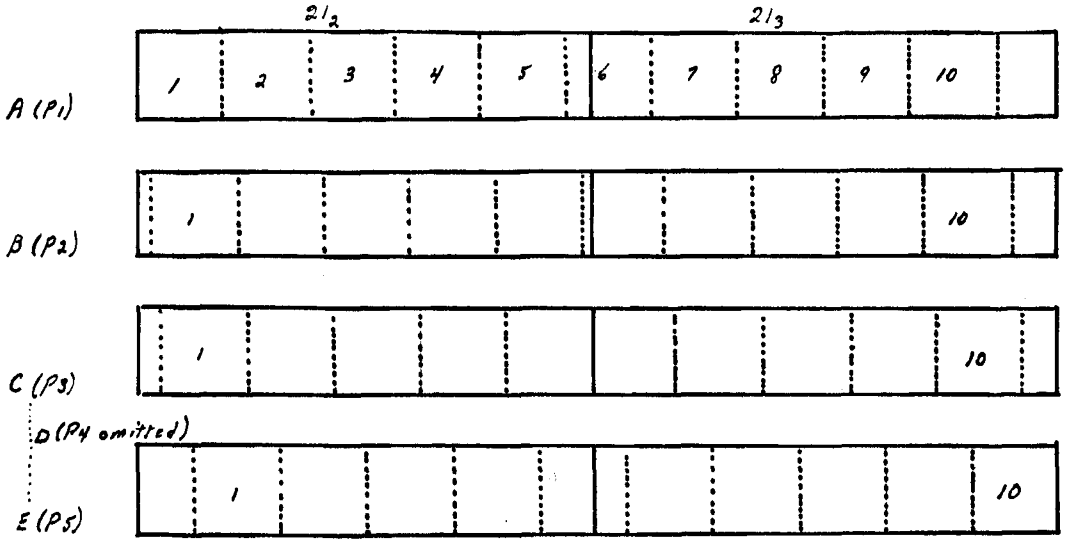
The 16 consecutive minor cycles of D.L.12 not occupied by the read store.

Standard Punch Store.

Minor cycles 16 to 31 of D.L.12.

Word Pair P Positions.

One DEUCE word pair can accommodate 10 continous characters in five different ways. The least significant digit of the least significant character may be located at $P_{1,2,3,4}$ or 5 of the less significant word. Taking 10 characters in DS21 by way of illustration we have the following possibilities.



C is an important configuration. The 5 least significant characters are contained in 21_2 with P_1 and P_2 as the spare digits and the 5 more significant characters are in 21_3 with spare digits in P_{31} and P_{32}. In this arrangement the 10 characters are said to be "split in 21_2". A statement "Fetch ten characters to P_3 in 21" means "Extract ten characters from wherever they may be stored and place them in DS21 as shown at C above".

Packing and Unpacking.

These are terms meaning the extraction or assembly of selected characters from the character store.

Left Justify.

Up to ten characters may be extracted from any ten consecutive character positions of the character store and placed in a DS with the least significant character in the least significant digit position. Characters so placed are said to be "Left justified", i.e. lined up with the least significant character at the left-hand end of the D.S.

21C.6 DATA PROCESSING SUBROUTINES.

The logical structure of DEUCE, which determines its order code, permits operations on words (or blocks of words) of 32 digits. There is now available a vast store of programming technique for handling problems which are concerned solely with 32 digit word operations.

Data processing applications however give rise to a requirement for operations on groups of characters. Since the DEUCE has no built-in facilities especially adapted to character manipulation, all such operations must be programmed within the framework of an order code which is designed to deal with 32 digit words. The compact form of character store is such that an item of information comprising several characters may extend over two or more DEUCE words starting in the middle of one and ending in the next and this implies that packing and unpacking will involve extensive use of the logical operation and shifting functions.

Several subroutines have been constructed to illustrate techniques which are useful in character manipulation. These routines fall into two categories, those which deal with fetching, storing and re-arrangement of the characters within the character store and those which are concerned with arithmetic operations on characters and conversions between binary coded decimal and binary.

10 Character Fetch from D.L. A.

If a standard read store is transferred from D.L.12 to any other delay line D.L.A (or left in 12) there is a requirement to extract any number of consecutive characters from any position of the store. 10 characters is a reasonable maximum for one entry to the routine and this limit has been chosen. Before entering the subroutine it is necessary to specify two things.

1. The number of characters required n ($\leqslant$ 10).
2. The least significant character position.

On exit from the subroutine the n characters are in DS21 at P_3 position in 21_2. Unoccupied character positions at the most significant end are filled out with zeros. A full description is given in the instructions for use in Appendix 2.

Character Select or Reject.

It may be necessary to block out certain areas of the character store, replacing selected characters by zeros. A subroutine, described in detail in Appendix 2, has been constructed to do this. Characters in one word pair may be selected (i.e. left in the character store) or rejected (i.e. replaced by zeros) at one entry to the routine.

Before entering the subroutine it is necessary to specify

1. Which word pair is to be modified.

2. How many characters in each word of the pair are affected and whether selection or rejection is required.

Insertion of Space Symbols or Zeros in Character Store.

This is similar to character select or reject. This routine however permits the user to specify whether rejected characters are to be replaced by zeros or space symbols (to create blank columns on punch out).

21C.7 CONVERSION ROUTINES.

Before describing the conversion routines it is probably not out of place to discuss conversion methods. These in turn are dependent on a number of factors among which are

(a) The maximum value of the numbers to be converted.
(b) The speed of the method.
(c) The number of instructions required.

(a) Number Range.

Bearing in mind that conversions discussed in this section are those from 6-bit B.C.D. to binary and back again, the maximum limits may be set by either (1) the binary capacity of a single DEUCE word, (2^{31} - 1) or (2) the decimal capacity of 5 or 10 characters. We could of course choose to work in double length binary numbers and use more than 10 characters, but if we do, the conversions will be more complicated. The basic binary units also determine number capacity in characters as the following examples show.

Example 1.

10^0	10^1	10^2	10^3	10^4	

using 5 characters for representation of DECIMAL data the maximum number which can be held is 99999, i.e. < 10^5. In binary this is 31, 20, 1, 3 i.e. less than 2^{18}, showing that binary notation is a much more compact form than 6-bit B.C.D.

If two DEUCE words are used for 10 characters the maximum number which may be represented in B.C.D. is 9,999,999,999. The binary equivalent of this is outside single length, however, and if we limit our range to single length binary the maximum number will be 2^{31} - 1.

Example 2.

5 character Sterling.

d	s	10/-	£10^0	£10^1	

The maximum number of pence in this case is that corresponding to £99.19.11d. which is 23999. In binary this is 31, 13, 23, ie., less than 2^{15}. The packing efficiency of B.C.D. is worse in this case. The pence character is holding slightly more than in a decimal number but the 10/- character is holding considerably less.

Example 3.

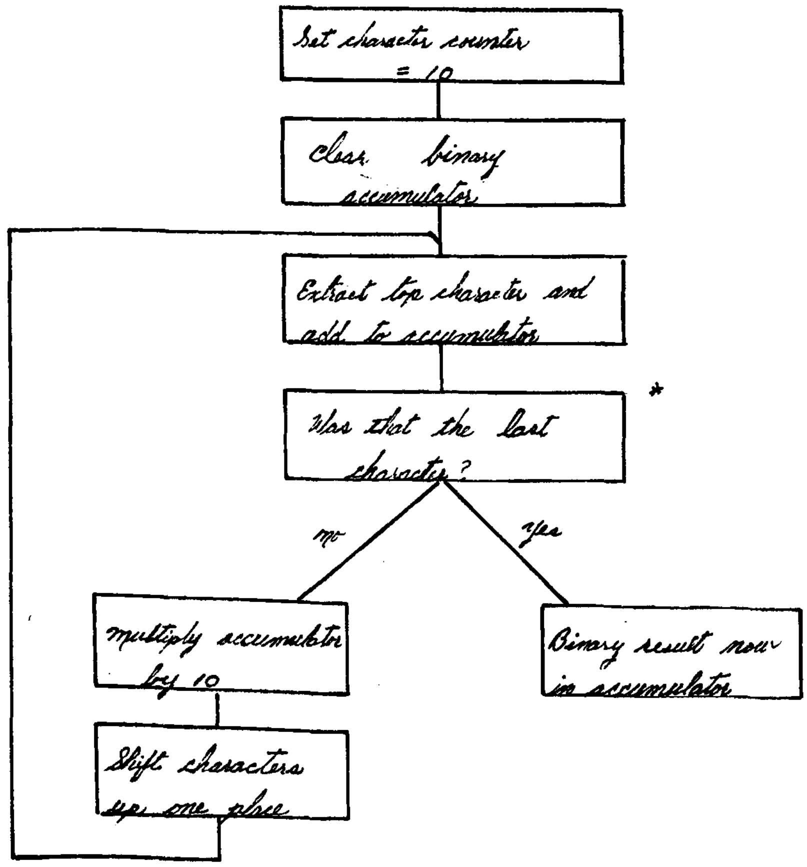
10 character sterling.

Using 10 characters and with the same pence and shillings arrangement as in Example 2 we can hold £9999999. 19. 11d. in two words. This represents more pence than can be held in one single length number and if we limit ourselves to $(2^{31} - 1)$ pence, the maximum value in sterling will be £8947848. 10. 7d. which may not be large enough for banking or insurance applications but will probably be quite acceptable to most people.

(b) and (c) Speed and Instruction Space.

It is probably true to say that the best routine made to date is always twice as slow and uses twice as many instructions as the customer wants. When we make a routine taking no time at all, handling an unlimited range of numbers and using only one instruction, we shall put the flag out. In the meantime the efforts towards this ideal have not been unimpressive much depends on the basic method of attack. There are programming techniques, for example, which process each digit of a word in conversions from either B.C.D. to binary or binary to B.C.D. Bearing in mind the fact that it will probably take a millisecond loop to deal with each digit such methods are almost bound to be too slow. There are tricks of the trade which permit such methods to be done with only 10 trips round the loop, instead of 32, but even this improvement is hardly worthwhile.

The next class of conversion methods deals with one character per millisecond loop. Taking the conversion of a ten character number from B.C.D. to Binary the block diagram of the method would be as follows


Set character counter = 10
Clear binary accumulator
Extract top character and add to accumulator
Was that the last character?
*
no
yes
multiply accumulator by 10
Shift characters up one place
Binary result now in accumulator

* This block is a "count character and test zero" routine

To convert binary numbers to characters the first move is that of dividing the number by a scaling factor to produce a binary fraction, preferably to 32 b.p.

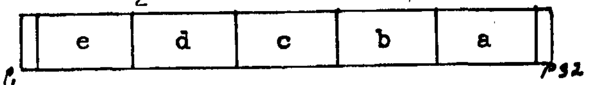
Suppose we have a number N less than 10^8 which is to be converted from binary to B.C.D. We divide by 10^8 (to 1 b.p.) and obtain $^N/10^8$ to 32 b.p.

Successive multiplication by 10 will bring up each figure of the number one at a time and these can be packed as successive six bit characters. This method (and the corresponding character conversion of B.C.D. to binary) is adopted in Subroutines P23E and R27E. The flow diagrams of these subroutines which are the most comprehensive yet made are given in Appendix 2. In addition to performing both input and output conversions, these routines also extract and insert characters at specified positions in the character store.

Following the more conventional one character at a time type of conversion are the methods which deal with 5 6-bit characters or 8 four bit characters in one multiplication or division operation. The DEUCE multiplier-divider can be programmed to perform the functions of an automatic converter. These techniques, which are slightly unorthodox, are fully described in DEUCE News No. 36. Several routines which use these methods are included in Appendix 2 and the instructions for use adequately describe the special purpose ones. However, one routine using a form of radix control is fully generalized. This will now be described in some detail.

General Purpose Conversion Routine.

This is a double entry routine using 26 instructions which converts 5 6-bit B.C.D. characters to binary or binary to 5 6-bit B.C.D. characters, taking $2\frac{1}{2}$m.s. for the first operation and $3\frac{1}{2}$ m sec. for the second. The type of conversion (and the relation between the units) is specified by constants which must be placed in QS17 and $DS19_2$ before entering the routine. The constants placed in 19_2 are known as "fillers" and are needed in the case of Binary to B.C.D. only. Suppose we have any five digit number N = ABCDE situated in character form at P_2 position in a DEUCE storage location as shown



To convert this to binary we use Entry 1 of the subroutine as follows

$N\ (abcde)_{P_2}$ - 21_3

Constants - 17 (4 m.c.)

Link - 13

Entry 1

1_{30} (LINK)

Result NP_1 (binary) in 21_3

after $2\frac{1}{2}$ m sec.

Now suppose we wish to reverse this procedure. We have binary number N which we wish to convert to character form. The procedure is

NP_1 - 21_3

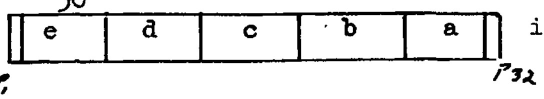
Constants - 17 (4 m.c.)

Fillers - 19_2

Link - 13

Entry 2

1_{30} LINK

Result  in 21_2 after $3\frac{1}{2}$ msec.

Now let us look at some of the possible forms of N = abcde as listed in the table below:

Units.	N = abcde	N (binary)
Decimal.	$ax10^4 + bx10^3 + cx10^2 + dx10^1 + ex10^0$	binary
Sterling.	£a.10^1 + £b.10^0 + C(10/-) + d (S) + e (pence)	binary pence.
Weight.	a$x10^1$ cwt. + b.10^0 cwt. + C(QTRS.)+ d (stones) + e (lbs.)	binary L.B.S.
Length.	a furlongs + b chains + C* (blank) + d (yds.) + e (ft.)	binary feet.
TIME (or angular measure)	a hours + b.10^1 min. + c.10^0 min. + d.10^1 sec. + e. sec. degrees.	binary seconds.

In each of the systems of units shown an integral multiplier connects one unit to the next higher unit. Denoting the multipliers quite generally we have, from

e to d, m_1 i.e. $m_1 e = d$
d to c, m_2 i.e. $m_2 d = c$
c to b, m_3 i.e. $m_3 c = b$
b to a, m_4 i.e. $m_4 b = a$

For the cases above m_1, m_2, m_3, m_4 take the following values.

	m_1	m_2	m_3	m_4
DECIMAL	10	10	10	10
Sterling	12	10	2	10
Weight	14	2	4	10
Length	3	22	1*	10
TIME	10	6	10	6

*NOTE: Blank characters may be generated to allow expansion from single conversion to double character representation. This is necessary in cases such as 12 pence, 22.yards etc.

The rules for blank units are, for example,

22 yds. = 1 blank unit.
1 blank unit = 1 chain.

Another useful dodge is the introduction of unusual units such as 6^d or ½ chains.

e.g. 6^d, 6 pence = 1 (6^d)
½ chain, 11 yrd. = ½ chain.

IMPORTANT: Conversions can only be carried out for systems of units for which integral multipliers exist between units.

The conversion routine can not directly handle systems such as the following

pence	10^d	shillings	
yds.	10' yds.	chains.	
lb.	10' lb.	QTRS.	

because there are no integral multipliers between 10^d and shillings, 10 yds. and chains or 10 lb. and QTRS. In these cases blank units can be left or use made of the 6^d or ½ chain dodge to leave a character space for subsequent adjustment outside the routine.

The constants which would be required in QS17 in each case are listed below for each minor cycle.

1	DECIMAL	Sterling	LB WT	Length	TIME
17_0	$2^6 .10^4$	$2^6 .2400$	$2^6 .1120$	$2^6 .660$	$2^6 .3600$
17_1	$2^{12}.10^3$	$2^{12}.240$	$2^{12}.112$	$2^{12}.66$	$2^{12}.600$
17_2	$2^{18}.10^2$	$2^{18}.120$	$2^{18}.28$	$2^{24}.3$	$2^{18}.60$
17_3	$2^{24}.10^1$	$2^{24}.12$	$2^{24}.14$	$2^{24}.3$	$2^{24}.10^1$

There is another constant 2^{30} built into the subroutine to handle the bottom character which in all systems must be the same in character form as in binary.

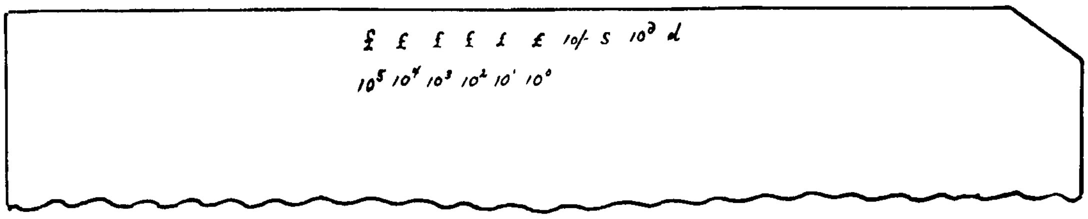
The Fillers required are

DECIMAL	$54\ P_2 + 54\ P_8 + 54\ P_{14} + 54\ P_{20}$
STERLING	$52\ P_2 + 54\ P_8 + 62\ P_{14} + 54\ P_{20}$
LB WT	$50\ P_2 + 62\ P_8 + 60\ P_{14} + 54\ P_{20}$
Length.	$61\ P_2 + 42\ P_8 + 63\ P_{14} + 54\ P_{20}$
TIME.	$54\ P_2 + 58\ P_8 + 54\ P_{14} + 58\ P_{20}$
General.	$(64-m_1)P_2 + (64-m_2)P_8 + (64-m_3)P_{14} + (64-m_4)P_{20}$

Some examples will now be given illustrating the use of this subroutine.

Example 1.

Input cards are punched with Sterling in the range less than £10^6. Ten columns of the card are used as shown



It is required to convert amounts expressed in sterling to binary pence.

The characters which are produced within DEUCE are as follows.

d	10d	S	10/-	£10^0	£10^1	£10^2	£10^3	£10^4	£10^5

Method: (i) Convert the 5 most significant characters as DECIMAL, multiply by 2400.

(ii) Change the form of the 5 least significant characters (by logic) to either of those shown

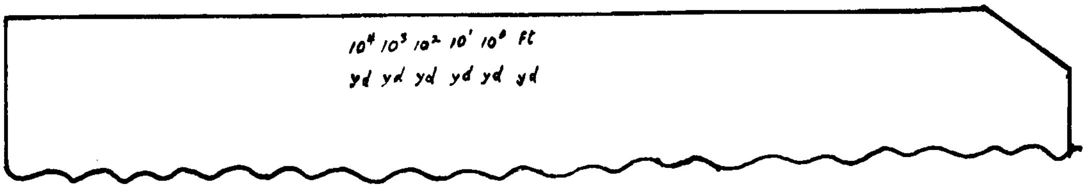
d	Blank	S	10/-	£10^0
d	6^d	S	10/-	£10^0

(iii) Using the appropriate constants convert the 5 least significant characters to pence.

(iv) Add the contributions from (1) and (3).

Example 2.

Lengths are punched on cards in units of 10,000 yds. and ft. as shown



It is required to convert these numbers to binary ft. There are 6 characters and it would be a waste of effort to follow a similar method to Example I, and use the conversion routine to convert only one character (representing 10^4 yds.) when all we need is a multiplication.

Method: (i) Extract the 10^4 character and multiply by 3×10^4.

(ii) Convert the 5 least significant characters to feet using the constants

$$2^6 .10^3 . 3$$
$$2^{12}.10^2 . 3$$
$$2^{18}.10^1 . 3$$
$$2^{24}.10^0 . 3$$

(iii) Add the contributions from (i) and (ii).

In the reverse conversion it is necessary to know how to perform integer divisions. Details of this technique are given in the next section and for the present it will be sufficient to assume that integer divisions are possible.

Example 3.

To convert binary pence to sterling, using single column pence (with the convention $Y = 10^d$, $X = 11^d$) up to a maximum of $(2^{31} - 1)$ pence.

Method: (i) Divide the pence by 240×10^2 (the number of pence in £100) to obtain a quotient (Q) representing the number of £100 units and a remainder (R) representing pence.

(ii) Convert Q using DECIMAL conversion to obtain characters as shown

£10^2	£10^3	£10^4	£10^5	£10^6

(iii) Convert R using STERLING conversion with single column pence to give

d	s	10/-	£10^0	£10^1

(iv) By logic change the 'd' character to 16 if d = 10 and to 32 if d = 11.

(v) Merge the two sets of characters from (ii) and (iii)

Example 4.

A binary number representing yards is to be converted to MILES and yards. Powers of 10 of MILES and yards are to be generated directly.

Method: (i) Divide the original number by 1760 to give a binary quotient Q representing MILES and a remainder R (< 1760) representing yards.

(ii) Convert Q using DECIMAL constants to give

10^0 MILES	10^1 MILES	10^2 MILES	10^3 MILES	10^4 MILES

(iii) Convert R using DECIMAL constants to give

10^0 yds.	10^1 yds.	10^2 yds.	10^3 yds.	10^4 yds.

The 10^4 yds character will be zero and the two sets of characters from (2) and (3) may be merged finally to close up this gap giving

10^0 yds	10^1 yds	10^2 yd	10^3 yd	10^0M	10^1M	10^2M	10^3M	10^4M

The universal conversion subroutine is a fairly powerful programming tool but like any other tool the operator must be prepared to use it intelligently. A universal screw cutting machine is also a powerful tool and anyone knowing how it operates can work out the settings to cut any given length of screw to specified tooth depths and pitch but no one ever does. A table of settings is provided and it is only necessary to look up the details for a particular requirement and screw cutting ceases to be a skilled operation. The ability to think is replaced by the ability to read. To overcome the errors which are likely to creep in through miscalculation of constants a table is provided for the universal conversion routine. This shows a fairly extensive range of possible conversions together with the constants required in each case, in both decimal and binary. Limits are given for 5 characters and 10 characters for each character layout and the 10 character limit assumes that the 5 more significant characters are powers of ten of the same unit. The divisor required for integer division is also provided and the sequences of instructions for integer division will now be described.

21C.8 INTEGER DIVISION.

The DEUCE divider normally produces a binary fraction to 31 b.p. and it is necessary to use the divider in a rather special way to perform integer divisions.

Two methods are available to permit division of a positive integer A by a positive integer B to give positive integral quotient and remainder related by

$$A + QB + R$$

e.g. 3 goes into 29to give 7 and 2 over

$$\text{or} \quad 29 = 7.3 + 2$$

$$\text{where} \quad A = 29,\ B = 3,\ Q = 7 \text{ and } R = 2$$

The methods are known as Type I and Type II, and Type II is used to obtain true remainders in these cases where Type I is not adequate.

Procedure.

(1) Consider the range of numbers A which are to be dealt with. If the top digit of the maximum number in the range is P_{30} or less use Type I. If it reaches P_{31} Type II is necessary.

(2) Shift B up n places until the top digit of 2^nB is the same as the top digit of $(A)_{max}$.

(3) Type I.

(a) Start a DIVISION (1-24) of A by 2^nB in minor cycle m.

(b) Clear 21_2 in minor cycle m+3.

(c) Extract the contents of DS21 in minor cycles m+4 + 2n and m+5+2n using SOURCE 22. These are the quotient (from 22_2) and an uncorrected remainder (from 22_3)

(d) Add the divisor (2^nB) to the uncorrected remainder to give the true remainder shifted up by 2^{n+1}.

(4) Type II.

(a) Start a DIVISION (1-24) of A by 2^nB in minor cycle m.

(b) Clear 21_2 in minor cycle m+3.

(c) Extract the uncorrected remainder in minor cycle m + 2 + 2n.

(d) Extract the true quotient in minor cycle m+5+2n.

(e) Correct the machine remainder as follows

(i) If the bottom digit of the quotient (P_1) is zero add the divisor (2^nB)

(ii) If the corrected remainder has a P_{32} digit remove it.

The correct remainder will be produced with a shift of 2^n.

Example 1.

Given a binary number A (< 2240 . 10^5) representing LB WT. To obtain a quotient representing binary TONS and a remainder in LBS. We need to divide by 2240 = B. Now 2240 is 0 6 2 and the top digit is P_{12}. The maximum value of A is 2240 . 10^5 which is 0 0 30 19 21 6 with a top digit of P_{28}, so Type I may be used. The top digit of B is P_{12} and we shall need a shift of 28 - 12 = 16 to move the top digit of 2^{16} to P_{28}.

Now we can go.

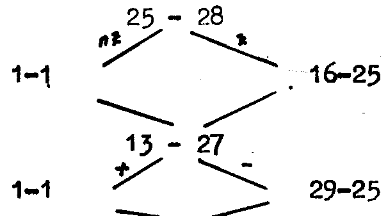
$A - 21_3$
$(2^{16}.2240) - 16$
$1 - 24$ (m.c. m)
$30 - 21_2$ (m.c. m+3)
$16 - 13$ prepare to correct remainder
$22 - 20$ (d) m.c. m+4+(32)
m+5+(32)
$20_3 - 25$ correct the remainder

Result. QP_1 in 20_2
RP_{18} in 13 (i.e. 2^{17} . R)

Example 2.

Given a binary number A ($< 2^{31}$) representing pence. To obtain a quotient representing £10^2 and a remainder in pence. The top digit of A is P_{31} so Type II is needed . We shall require to divide by the number of pence in £100 i.e. B = 2400 which is binary is 0 14 23 with a top digit of P_{15}. The number of shifts to move the top digit to P_{31} is 31-15 = 16 so we form 2^{16}.24000 and proceed as follows

$A - 21_3$
$(2^{16}.24000) - 16$
$1 - 24$ (m.c. m)
$30 - 21_2$ (m.c. m+3)
$27 - 14$
$22_3 - 13$ m.c. m+2 + (32)
$22_2 - 15$ m.c. m+5 + (32)



Result. QP_1 in 15
RP_{17} in 13 (i.e. 2^{16}.R)

21C.9 MISCELLANEOUS SUBROUTINES.

1. Character Add-Subtract.

Where only additions or subtractions of characters are required this can be done directly and conversions from B.C.D. to binary and binary to B.C.D. may be eliminated. Suppose we wish to add two numbers in character form such as 1 7 3 9 2 4 and 4 8 3 2 7 1 5. These are presented as

4	2	9	3	7	1	0	0	0	0

and

5	1	7	2	3	8	5	0	0	0

and the sum can be formed directly as

9	3	6	6	0	0	5	0	0	0

Subtractions can also be performed by a different entry to the subroutine and in the event of a negative result (which would produce the 10's complement) a second subtraction is performed automatically to give the modulus with a sign digit in the top character.

2. Read and Punch Binary File.

Some data processing applications may be small enough to permit the standing file to be kept on cards. Here the requirement after an updating run is to punch out the file in the most compact form (binary) with the ability to read it in again to exactly the same place (and fully checked) on the next updating run. A routine has been written to do this which will be published shortly.

3. Character Compress and Expand.

Characters of 6 binary digits are wasteful for numeric information and tighter packing may be obtained if characters are compressed to four bit form. Subroutines for 6 to 4 compression and 4 to 6 expansion are in preparation. The techniques are well established and the subroutines will be published as soon as possible.

APPENDIX 2.

This appendix contains instructions for use (together with flow diagrams and coding) for some of the subroutines which are in course of publication for character manipulation and conversion. Some routines have been published and a list of routines is given below:

1. P23E and R27E are a pair of general fetch convert and store routines. Full details are available in the relevant reports NS t 257 and NS t 256.

2. 10 character Binary to BCD conversion (DECIMAL) $N < 2^{14}. 10^{5}$.

3. 10 character BCD to Binary conversion (DECIMAL) $N < 2^{31}$.

4. Double entry 5 character Binary to BCD or BCD to binary $N < 10^{5}$.

5. General integer division.

6. 10 character fetch from character store.

7. Character select or reject in character store.

8. Insert zeros or space symbols in character store.

9. Double entry DECIMAL character add-subtract.

10. Binary to 4 bit BCD and 4 bit BCD to binary conversions ($N < 10^{8}$). These are in preparation as separate routines or as a combined double entry routine.

11. Binary pence to 4 bit £. s. d.

12. Double entry 10 character binary to BCD or the reverse which covers the maximum range of up to 2^{31}. This is a combined version of (2) and (3) with the range of (2) extended.

15. Double entry binary pence to £. s. d. and the reverse for amounts less than £100 using single character pence.

16. Universal 10 character Add-Subtract. Any units may be added provided integral multipliers exist between adjacent characters.

17. Universal double entry conversion of 5 characters from BCD to binary and the reverse.

18. Six to four bit compression and four to six bit expansion.

19. Read and Punch binary file.

APPENDIX 1.

Card Input Code.	6-Bit Equivalent (Decimal).	6-Bit Equivalent (Binary).	Associated Symbol.	Card Output. Code.
YX08 other		(Most Sig. on right)		YX08 other
..0..	0	0000 00	0	..0..
....1	1	1000 00	1	1
....2	2	0100 00	2	2
....3	3	1100 00	3	3
....4	4	0010 00	4	4
....5	5	1010 00	5	5
....6	6	0110 00	6	6
....7	7	1110 00	7	7
...8.	8	0001 00	8	...8.
....9	9	1001 00	9	9
...82	10	0101 00	10	...82
...83	11	1101 00	11	...83
...84	12	0011 00	12	...84
...85	13	1011 00	13	...85
...86	14	0111 00	Free	...86
Blank	15	1111 00	Space	Blank
Y....	16	0000 10	Plus	Y....
Y...1	17	1000 10	A	Y...1
Y...2	18	0100 10	B	Y...2
Y...3	19	1100 10	C	Y...3
Y...4	20	0010 10	D	Y...4
Y...5	21	1010 10	E	Y...5
Y...6	22	0110 10	F	Y...6
Y...7	23	1110 10	G	Y...7
Y...8	24	0001 10	H	Y...8
Y...9	25	1001 10	I	Y...9
Y..82	26	0101 10	Free	Y..82
Y..83	27	1101 10	Free	Y..83
Y..84	28	0011 10	Free	Y..84
Y..85	29	1011 10	Free	Y..85
Y..86	30	0111 10	Free	Y..86
Y..87	31	1111 10	Free	Y..87
.X...	32	0000 01	Minus	.X...
.X..1	33	1000 01	J	.X..1
.X..2	34	0100 01	K	.X..2
.X..3	35	1100 01	L	.X..3
.X..4	36	0010 01	M	.X..4
.X..5	37	1010 01	N	.X..5
.X..6	38	0110 01	O	.X..6
.X..7	39	1110 01	P	.X..7
.X.8.	40	0001 01	Q	.X.8.
.X..9	41	1001 01	R	.X..9
.X.82	42	0101 01	Free	.X.82
.X.83	43	1101 01	Free	.X.83
.X.84	44	0011 01	Free	.X.84
.X.85	45	1011 01	Free	.X.85
.X.86	46	0111 01	Free	.X.86
.X.87	47	1111 01	Free	.X.87
No Code	48	0000 11	Free	..0..
..0.1	49	1000 11	Free	..0.1
..0.2	50	0100 11	S	..0.2
..0.3	51	1100 11	T	..0.3
..0.4	52	0010 11	U	..0.4
..0.5	53	1010 11	V	..0.5
..0.6	54	0110 11	W	..0.6
..0.7	55	1110 11	X	..0.7
..08.	56	0001 11	Y	..08.
..0.9	57	1001 11	Z	..0.9
..082	58	0101 11	Free	..082
..083	59	1101 11	Free	..083
..084	60	0011 11	Free	..084
..085	61	1011 11	Free	..085
..086	62	0111 11	Free	..086
..087	63	1111 11	Space/Ignore	Blank

DEUCE SUBROUTINE.

TEN CHARACTER FETCH.

Description.

Fetches and left justifies (to P_3) up to ten characters from any specified character position in an 80 col. character store.

Data

(a) Number of characters (n) required n $\leqslant$ 10.

(b) Character position of least significant character.

The number of characters is specified as n P_{17} in TS 16. The least significant character position is specified as TP_{21} + u P_{17} where T and u are the tens and units digits of the character position.

Example:

Fetch five characters at Character position 73

TS 16 contains $5P_{17}$

TS 14 contains $7P_{21}$ + $3P_{17}$

Result.

Up to ten characters in DS 21 with the least significant character at P_3 position in 21_2. Character positions unoccupied are clear.

Example:

C.P. 80	79	78	77	76
75	74	73	72	71
50	49	48	47	46
45	44	43	43 (9)	41 (3)
40 (4)	39 (2)	38 (1)	37 (7)	36

To fetch 6 characters from L.S. position 42.

Result in $21_{2,3}$.

P_3

9	3	4	2	1	7	zero	zero	zero	zero
42	41	40	39	38	37				
21_2					21_3				

Parameters.

For characters in Delay LINE A insert A as source number in 2_0

Stores Used.	13	14	15	16	20	21	17_2	18_2
Contents at Entry.	Link	TP_{21}+ UP_{17}	-	nP_{17}	-	-	-	-
Contents at Exit.	-	-	-	-	-	Result	-	-

Time.

The time depends on the number of characters required and the position of the least significant character.

For $u = 0$ Time $= 5$ msec. (any value of n).

For $n \leqslant u$ Time $= (5 + (10-u))$msec.

For $n > u$ Time $= 16$ msec.

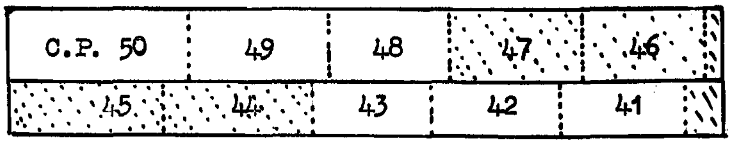
DEUCE SUBROUTINE.

CHARACTER SELECT OR REJECT.

Description.

This subroutine modifies the 80 col. character store, replacing continuous groups of characters by zeros. The first n ($\leqslant$ 5) characters in either or both words of a word pair may be selected (i.e. retained) or rejected (i.e. replaced by zeros).

Example:



C.P. 50	49	48	47	46	
45	44	43	42	41	

C.P. = Character Position.

Suppose it is required to replace character positions 47, 46, 45 and 44 by zeros. This requirement is specified to the subroutine as follows:

"In the word pair located on Character Position 50 retain (the first) three characters in the even word and reject (the first) two characters in the odd word".

The character position defining the word pair must always be one of 80, 70, 60, 50, 40, 30, 20, 10.

It is specified parametrically as:

$$\frac{\text{(character position)}}{10} \times P_{18} \quad \text{in TS 14.}$$

e.g. 50 is specified as $5P_{18}$

30 is specified as $3P_{18}$.

The number of characters and whether these are to be rejected or selected is specified as a codeword in 20_2 for the even word and in 20_3 for the odd word.

Retain the first n characters' is specified as:

$$nP_{17} + 12P_{26}$$

'Reject the first n characters' is specified as:

$$nP_{17} + (0P_{26})$$

For the example we require:

$$3P_{17} + 12P_{26} \quad \text{in} \quad DS20_2$$

and $2P_{17}$ + in $DS2C_3$.

Subroutine Input.

1. Parameter specifying word pair as $\frac{\text{(Character Position)}}{10} \times P_{18}$ in TS 14. The Character Position must be a multiple of ten.

2. Codewords in $20_{2,3}$ specifying the number of characters to be selected or rejected in the even and odd words respectively.

In	$DS20_2$	$n_1\ P_{17}$	$+ K\ P_{26}$
In	$DS20_2$	$n_2\ P_{17}$	$+ K\ P_{26}$

K = 12 to select characters

K = 0 to reject characters

3. Link in TS 13.

Stores Used.	13	14	15	$20_{2,3}$	$21_{2,3}$
Contents at Entry.	Link	$(\frac{CP}{10})xP_{18}$		Codewords	
Contents at Exit.	-	-	-	-	-

Entry.

2_{26}.

Occupies.

$2_0 - 23$ 25 - 30.

Time.

6 m sec.

NOTES:

1. The routine assumes that the character store is m.c. 0 - 15 of a DELAY LINE.

2. The character store can be in any DL A by insertion of A as SOURCE number in 2_9.

3. The modified character store can be in the same minor cycles of any delay line B by insertion of (21 - A) and (B + 11) as S and D numbers in 2_{27}.

4. If the final character store (i.e. after modification) is required in minor cycles W' to W' + 15 change the wait number of 2_{27} to (31 + W')

DEUCE SUBROUTINE.

INSERT SPACE SYMBOLS OR ZEROS.

Description.

This subroutine modifies the character store by eliminating continuous groups of characters, replacing then either by zeros or by space symbols. The first n ($\leqslant$ 5) characters in either or both words of a word pair may be specified. To indicate whether the eliminated characters are replaced by zeros or space symbols a "spaces format" parameter is used.

Code.	Meaning.
0 0	Characters in both words replaced by zeros.
1 0	Characters in even word replaced by spaces. Characters in odd word replaced by zeros.
0 1	Characters in even word replaced by zeros. Characters in odd word replaced by spaces.
1 1	Characters in both words replaced by spaces.

Example:

C.P. 50	49	48	spaces 47	spaces 46
zeros 45	zeros 44	zeros 43	zeros 42	41.

C.P. = Character Position.

Suppose it is required to replace characters 47 and 46 by space symbols and replace 45, 44, 43, and 42 by zeros.

The requirement is specified to the subroutine as follows:

"In the word pair located on character position 50 select (i.e. retain) the first 3 characters in the evenword and eliminate the first 4 characters in the odd word. Replace rejected characters in the even word by spaces and in the odd word by zeros".

In TS 14 $\frac{\text{(Character Position of word pair)}}{10} \times P_{18}$ i.e. $5P_{18}$

DS 20_2 $3P_{17} + P_{26}$

20_3 $4P_{17}$

TS 16 Space Format in P_{31} P_{32} i.e. P_{31} in this example.

The codewords in DS 20 specify the number of characters and whether these are to be retained or rejected

i.e. DS 20_2 is $n_1P_{17} + K\,P_{26}$ for even word

20_3 is $n_2P_{17} + K\,P_{26}$ for odd word

K = 1 means Keep.

K = 0 means replace.

	13	14	15	16	20_2	20_3	19_2	19_3	21_2	21_3
Stores Used.	13	14	15	16	20_2	20_3	19_2	19_3	21_2	21_3
Contents at Entry.	Link	$(\frac{CP}{10})xP_{18}$	-	Space Format	Codewords					
Contents at Exit.	-	-	-	-	-	-	-	-	-	-

Occupies.

$^{2}0 - 31$ $^{3}0 - 16$.

Entry.

$^{2}24$.

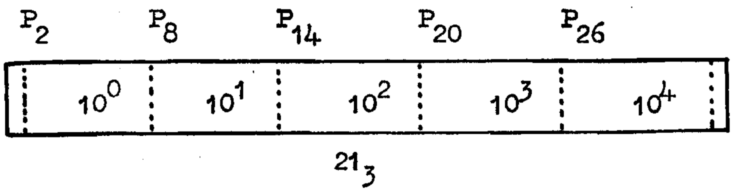
DEUCE SUBROUTINE.

5 CHARACTER CONVERSION.

Description.

A double entry subroutine which converts a 5 6-bit binary coded decimal integer to binary and vice versa.

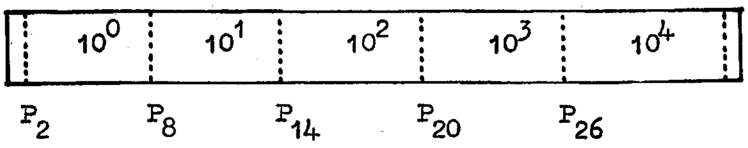
Input. **Entry 1.** A 6-bit binary coded decimal integer $N(< 10^5)$ located at P_2 position in 21_3 as shown.



Output. **Entry 1.** $N \times P_1$ in 21_3.

Input. **Entry 2.** A binary integer $N(< 10^5)$ as NP_1 in 21_3.

Output. **Entry 2.** A 5 digit 6-bit BCD integer N in 21_2 located at P_2 position.



Time.

3 milliseconds at each entry.

Occupies.

$^2 0 - 3 \quad 5 - 15 \quad 22 \quad 26 - 31.$

Stores Used.	13	15	16	21_2	21_3	
Contents at Entry.						
	Link	-	-	-	N (B.C.D)	Entry 1.
Contents at Exit.						
	Link	-	-		NP_1 (binary)	

Stores Used.	13	14	15	16	21_2	21_3	
Contents at Entry.							
	Link	-	-	-	-	NP_1 (binary)	Entry 2.
Contents at Exit.							
	Link	-	-	-	N(BCD)	-	

The subroutines employ multiplier divider techniques and both subroutines will fail if operated on single shots.

DEUCE SUBROUTINE.

TEN CHARACTER ADD/SUBTRACT.

Description.

A double entry subroutine to add or subtract ten digit positive decimal numbers in 6-bit binary coded decimal notation. Negative differences are given as the modulus with a sign in the most significant character location.

Data.

Character Add	AUGEND A	ADDEND B
Character Subtract	MINUEND A	SUBTRAHEND B.

A and B must be $\leqslant 10^9$ expressed as ten digit 6-bit BCD integers with the least significant digit of the least significant character at P_2 position.

Result.

Character Add	SUM A + B	DIFFERENCE A - B /A/ ≥ /B/
		-(B - A) /B/ > /A/

If $A + B > 10^9$ the result will be out of ten character length. Within the ten characters the result will be $(A+B) - 10^9$ with a digit at P_{30} position of the more significant word.

Examples:

1.	A	3	2	5	7	9	1	3	6	0	0	
		P_2										P_{30}
	B	5	7	9	6	4	2	5	7	0	0	
		P_2										P_{30}
	A+B	8	9	4	4	4	4	8	3	1	0	
		P_2										
2.	A	7	3	2	1	4	6	5	7	9	9	
		P_2										
	B	2	1	3	2	5	8	7	4	3	3	
		P_2										
	A+B	9	4	5	3	9	4	3	2	3	3	1
		P_2										P_{30}
3.	A	5	7	3	2	1	7	9	9	0	0	
	B	2	1	7	4	3	1	2	0	0	0	
	A-B	3	6	6	7	7	5	7	9	0	0	

4. A	1	2	3	4	5	6	7	0	0	0
B	5	4	7	2	9	9	8			
-(B-A)	4	2	4	8	3	3	1			1
										P_{29}

NOTE:

The top character includes P_{29} as -ve sign.

Time.

Character Add	1½ milliseconds.	
Character Subtract	3 milliseconds	/A/ ≥ /B/
	6 milliseconds	/A/ < /B/

Stores Used.

13	14	15	16	$20_{2,3}$	$21_{2,3}$

Contents at Entry.

Link	-	-	-	B	A

Contents at Exit.

	-	-	-	B (add) A-B* (subtract)	A+B (add) A-B (subtract) or -(B-A) "

* If /A/ ≥ /B/, $20_{2,3}$ contains B at the exit from SUBTRACT.

If /A/ < /B/, $20_{2,3}$ contains A-B as 10's complement with $P_{30\ 31\ 32}$.

Failures.

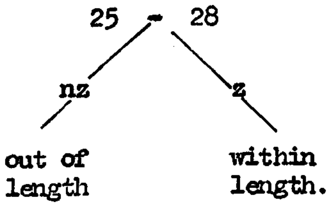
NONE.

If A+B > 10^9 the result will be out of ten character length. This may be detected outside the routine by

P_{30} - 14

21_3 - 15

25 - 28



DEUCE SUBROUTINE.

UNIVERSAL 5 CHARACTER CONVERSION FROM BINARY TO BCD OR BCD TO BINARY.

Description.

A double entry subroutine which converts 5 6-bit binary coded characters to binary and vice versa. Any system of units may be used provided integral multipliers exist between successive characters.

Input. **Entry 1. BCD to Binary.** A five character number N located with the least significant character at P_2 position in 21_3.

e	d	c	b	a
P_2	P_8	P_{14}	P_{20}	P_{26}

Output. **Entry 1.** NP_1 in 21_3.

Input. **Entry 2.** A binary integer NP_1 in 21_3.

Output. **Entry 2.** A five character 6-bit number N located at P_2 position in 21_2.

e	d	c	b	a

Time. **Entry 1.** $2\frac{1}{2}$ msec.

Entry 2. $3\frac{1}{2}$ msec.

Occupies.

mc. 4, 6, 8-31.

Stores Used.	13	14	15	16	19_2 (Entry 2)	21_2	21_3	17_{0-3}
Contents at Entry.	Link	-	-	-	FILLERS		N	CONSTANTS .
Contents at Exit.	-	-	-	-	FILLERS	N (entry 2)	N (entry 1)	CONSTANTS.

Entry 1. mc. 18.

Entry 2. mc. 17.

Parameters.

For either entry four constants must be provided in QS 17. These depend on the type of conversion taking place. A table of constants for several systems of units is attached.

For entry 2 only a set of FILLERS must be provided in 19_2. These are shown in the table of constants.

INPUT.	OUTPUT.					PARAMETERS.		5ch LIMIT	10ch LIMIT	DIVISOR	INTEGER DIV.
Binary x P_1	10^0	10^1	10^2	10^3	10^4	CONSTANTS.	$2^6 . 10^4$ 0,0,17,19	$< 10^5$	$< 2^{31}$	$2^{14}.10^5$	Type 2.
							$2^{12}.10^3$ 0,0,0,29,3	(0 21, 1, 3)		0 0 0,16,26,16,1	
							$2^{18}.10^2$ 0,0,0,0,25				
							$2^{24}.10^1$ 0,0,0,0,0,5				
						FILLERS	$54P_2 + 54P_8 + 54P_{14} + 54P_{20}$				
							(12,27,22,13,27,0)				
Binary Pence x P_1.	d	s	10/-	£10^0	£10^1.	CONSTANTS.	$2^6.2400$ 0,0,22,4	$< 240.10^2$	$< 2^{31}$	$2^{16}.24000$	Type 2.
							$2^{12}.240$ 0,0,0,30	(0 14 23)		0 0 0 0 28 14 1	
							$2^{18}.120$ 0,0,0,0,30				
							$2^{24}. 12$ 0,0,0,0,0,6				
						FILLERS	$52P_2 + 54P_8 + 62P_{14} + 54P_{20}$				
							(8 27 22 15 27 0)				
Binary Pence x P_1	d	Blank	s	10/-	£10^0.	CONSTANTS.	$2^6. 240$ 0,0,15	< 2400	$< 10^6.240$	2^{16} 2400	Type 1.
							$2^{12}.120$ 0,0,0,15	(0 11 2)	(0 0 7 28 4 7)	0 0 0 0, 22, 4	
							$2^{18}. 12$ 0,0,0,0,3				
							$2^{30}.$ 0,0,0,0,0,0,1				
						FILLERS	$52P_2 + 63P_8 + 54P_{14} + 62P_{20}$				
							(8 31 23 13 31 0)				
Binary Pence x P_1	d	6d	s	10/-	£10^0	CONSTANTS.	$2^6. 240$ 0,0,15	< 2400	$< 10^6.240$	$2^{16}.2400$	Type 1.
							$2^{12}.120$ 0,0,0,15	(0 11 2)	(0 0 7 28 4 7)	(0 0 0 0 22 4)	
							$2^{18}. 12$ 0,0,0,0,3				
							$2^{30}.$ 0 0,0,0,0,0,1				
						FILLERS	$58P_2 + 62P_8 + 54P_{14} + 62P_{20}$				
							(20 27 23 13 31)				

INPUT.	OUTPUT.						PARAMETER.	5ch LIMIT.	10ch LIMIT.	DIVISOR	INTEGER DIV.
Binary Pence x P_1	d	s	£10^0	£10^1	£10^2	CONSTANTS	2^6. 24000 0,0,28,14,1	240000	2^{31}	2^{31}.240000	Type 2.
							2^{12}. 2400 0,0,0,12,9	(0 12 10 7)		(0 0 0 0,19,26,1)	
							2^{18}. 240 0,0,0,0,28,1				
							2^{24}. 12 0,0,0,0,0,6				
						FILLERS	$52P_2+ 44P_8+ 54P_{14}+ 54P_{20}$				
							(8 19 21 13 27 0)				
Binary feet x P_1	ft.	yd.	½ch	ch	f	CONSTANTS	2^6. 660=0,8,9,1	< 5280	5280 x 10^5	2^{16} 5280	Type 1.
							2^{12}. 66=0,0,8,8	(0,5,5)	(0 0 9 17 23 15)	(0 0 0 0 10 10)	
							2^{18}. 33 0,0,0,8,8				
							2^{24}. 3 0,0,0,0,16,1				
						FILLERS	$61P_2+ 53P_8+ 62P_{14}+ 54P_{20}$				
							(26 23 22 15 27 0)				
Binary feet x P_1	ft.	yd.	Blank	ch	f	CONSTANTS	2^6. 660 0,8,9,1	< 5280	5280 x 10^5	2^{16}.5280	Type 1.
							2^{12}. 66 0,0,8,8	(0 5 5)	(0 0 9 17 23 15)	(0 0 0 0 10 10)	
							2^{24}. 3 0,0,0,0,16,1				
							2^{24}. 3 0,0,0,0,16,1				
						FILLERS	$61P_2+ 42P_8+ 63P_{14}+ 54P_{20}$				
							(26 11 29 15 27 0)				
Binary feet x P_1	ft.	10^0yd	10^1yd	10^2yd	10^3yd	CONSTANTS	2^6. 3000 0,16,27,5	< 3.10^4	5280.10^5	2^{16}. 5280	Type 1.
							2^{12}. 300 0,0,16,5,1	(16 9 29)	(0 0 9 17 23 15)	0 0 0 0 10 10	
							2^{18}. 30 0,0,0,16,7		(For MILES in top	(For MILES in top	
							2^{24}. 3 0,0,0,0,16,1		5 characters)	5 characters)	
						FILLERS	$61P_2+ 54P_8+ 54P_{14}+ 54P_{20}$				
							(26 27 22 13 27 0)				

DFCS2

INPUT.	OUTPUT.						PARAMETER.	5ch LIMIT.	10ch LIMIT.	DIVISOR.	INTEGER DIV.
Binary yd x P_1	Blank	10^0yd	10^1yd	10^2yd	10^3yd	CONSTANTS	$2^6.\ 10^3$ 0,16,30,1	1760	< 10^5.1760	2^{17}.1760	Type 1.
							$2^{12}.10^2$ 0,0,16,12	(0 23 1)	(0 0 3,27,7,5)	(0 0 0 0 28 6)	
							$2^{18}.10^1$ 0,0,0,16,2				
							$2^{24}.10^0$ 0,0,0,0,16				
						FILLERS	$54P_2+ 54P_8+ 54P_{14}+ 54P_{20}$				
							(12 27 22 13 27 0)				
Binary inches x P_1	in.	ft.	yd.	Blank	ch.	CONSTANTS	$2^6.\ 792$ 0,16,17,1	< 7920	15840.10^4	2^{15}.7920	Type 1.
							$2^{18}.\ 36$ 0,0,0,0,9	(16 23 7)	(0,16,31,1,23,4)	(0 0 0 16,23,7)	
							$2^{18}.\ 36$ 0,0,0,0,9				
							$2^{24}.\ 12$ 0,0,0,0,0,6				
						FILLERS	$52P_2+ 61P_8+ 42P_{14}+ 63P_{20}$				
							(8 23 31 26 31 0)				
Binary inches x P_1	in.	ft.	yd.	½ch.	ch.	CONSTANTS	$2^6.\ 792$ 0,16,17,1	< 7920	< 15840.10^4	2^{15}.7920	Type 1.
							$2^{12}.396$ 0,0,16,17,1	(16 23 7)	(0,16,31,1,23,4)	(0 0 0 16 23 7)	
							$2^{18}.\ 36$ 0,0,0,0,9				
							$2^{24}.\ 12$ 0,0,0,0,0,6				
						FILLERS	$52P_2+ 61P_8+ 53P_{14}+ 62P_{20}$				
							(8 23 15 13 31 0)				
Binary LBWT x P_1	LB.	QTR.	CWT.	10^1CWT	TONS	CONSTANTS	$2^6.2240$ 0,0,12,4	2240 x 10	2^{31}	$2^{16}.\ 22400$	Type 2.
							$2^{12}.1120$ 0,0,0,12,4	(0 28 21)	(958698 TONS	(0 0 0 0 24 11 1)	
							$2^{18}.\ 112$ 0,0,0,0,28		1 CWT 4 LB).		
							$2^{24}.\ 28$ 0,0,0,0,0,14				
						FILLERS	$36P_2+ 60P_8+ 54P_{14}+ 62P_{20}$				
							(8 18 23 13 31 0)				

INPUT.	OUTPUT.		PARAMETER.	5ch LIMIT.	10ch LIMIT.	DIVISOR.	INTEGER DIV.
<u>Binary LB</u>. x P_1	LB STONE QTR CWT 10^{1}CWT	<u>CONSTANTS</u>	2^{6}. 1120 0,0,6,2	2240	2240.10^{5}	2^{16} 2240	Type 1.
			2^{12}. 112 0,0,0,14	(0,6,2)	(0 0 30 19 21 6)	(0 0 0 0 12 4)	
			2^{18}. 28 0,0,0,0,7				
			2^{24}. 14 0,0,0,0,0,7				
		<u>FILLERS</u>	$50P_2$+ $62P_8$+ $60P_{14}$+ $54P_{20}$				
			(4 19 6 15 27 0)				
Binary LBWT x P_1	10^{0}LB 10^{1}LB 10^{2}LB 10^{3}LB Blank	<u>CONSTANTS</u>	2^{6}. 10^{4} 0,0,17,19	2240	2240.10^{5}	2^{16}.2240	Type 1.
			$2^{12}.10^{3}$ 0,0,0,29,3	(0 6 2)	(0 0 30 19 21 6)	(0 0 0 0 12 4)	
			$2^{18}.10^{2}$ 0,0,0,0,25				
			$2^{24}.10^{1}$ 0,0,0,0,0,5				
		<u>FILLERS</u>	$54P_2$+ $54P_8$+ $54P_{14}$+ $54P_{20}$				
			(12 27 22 13 27 0)				
Binary LBWT P_1	Blank 10^{0}LB 10^{1}LB 10^{2}LB 10^{3}LB	<u>CONSTANTS</u>	2^{6}. 10^{3} 0,16,30,1	2240	2240.10^{5}	2^{16}.2240	Type 1.
			$2^{12}.10^{2}$ 0,0,16,12	(0 6 2)	(0 0 30 19 21 6)	(0 0 0 0 12 4)	
			$2^{18}.10^{1}$ 0,0,0,16,2				
			$2^{24}.10^{0}$ 0,0,0,0,16				
		<u>FILLERS</u>	$54P_2$+ $54P_8$+ $54P_{14}$+ $54P_{20}$				
			(12 27 22 13 27 0)				
Binary CWT x P_1	CWT 10^{1}CWT 10^{0}T 10^{1}T 10^{2}T	<u>CONSTANTS</u>	2^{6}. 2000 0,0,29,3	20 x 10^{3}	20.10^{8}	$2^{16}.(20.10^{3})$	Type 2.
			2^{12}. 200 0,0,0,25	(θ 17 19)	(0 0 5 11 19 27 1)	(0 0 0 0 2,7,1)	
			2^{18}. 20 0,0,0,0,5				
			2^{24}. 10 0,0,0,0,0,5				
		<u>FILLERS</u>	$54P_2$+ $62P_8$+ $54P_{14}$+$54P_{20}$				
			(12 27 23 13 27 0)				

INPUT.		OUTPUT.		PARAMETER.	5ch LIMIT.	10ch LIMIT.	DIVISOR.	INTEGER DIV.
Binary Seconds	DPCS2	sec. 10^1sec min. 10^1min HRS or degrees	CONSTANTS.	2^6. 3600 0 ,0 ,1 ,7	36000	3600.10^4	2^{10} 36000	Type 1.
				2^{12}. 600 0 ,0 ,0 ,11 ,2	(0 5 3 1)	(0 8 20 10 2 1)	(0 0 0 5,3,1)	
				2^{18}. 60 0,0,0,0,15				
				2^{24}. 10 0,0,0,0,0,5		NOTE: This is probably as far as practical problems are likely to go. The actual maximum is 2^{31} but Type 2 integer division will be required.		
			FILLERS.	$54P_2 + 58P_8 + 54P_{14} + 58P_{20}$				
				(12 7 23 13 29 0)				
						2^{31}		
							2^{15} 36000	
							(0 0 0 0,5,3,1)	Type 2.

DEUCE SUBROUTINE.

UNIVERSAL 10 CHARACTER ADD/SUBTRACT.

Description.

A subroutine to add or subtract ten characters in any set of units provided integral multipliers exist from one character to the next.

Input.

Ten character numbers A and B. A is placed in DS 21 at P_2 position in 21_2 and B is placed in DS 20 at P_2 position.

Result.

A + B in 21_{23} for ADD.

A - B in 21_{23} for SUBTRACT (if $A \geq B$)

-(B - A) in $21_{2,3}$ for SUBTRACT (if $B > A$). In this case a sign digit is inserted in the most significant character (P_{29} in 21_3).

If the result is out of ten character length a P_{30} digit will be present in 21_3.

Examples:

A	5	7	6	3	1	2	1	3	0	0
B	4	1	2	5	9	9	8	9	1	
A+B	9	8	8	8	0	2	0	3	2	

A	7	3	9	5	4	7	2	1	9	
B	1	2	1	3	1	4	5	1		
A-B	6	1	8	2	3	3	7	9	8	

	d	s	£10^0	£10^1	£10^2	£10^3	£10^4	£10^5	£10^6	£10^7
A	5	15	1	3	4	7	2	1	9	0
B	9	2	3	4	1	2	1	3	2	0
A+B	2	18	4	7	5	9	3	4	1	1

	ft	yd	Blank	ch	f	m	m	m	m	
A	1	10	0	4	3	1	2	1	3	
B	1	20	0	5	3	4	7	1	2	
A+B	2	8	0	0	7	5	9	2	5	

Time.

ADD	3 m.s.	
SUBTRACT	3 m.s.	$A \geq B$
	6 m.s.	$B > A$.

Occupies.

m.c. 0 - 31.

Entry.

Add m.c. 30
Subtract m.c. 7

	13	14	15	16	$19_{2,3}$	$20_{2,3}$	21_{23}
Stores Used.	13	14	15	16	$19_{2,3}$	$20_{2,3}$	21_{23}
Contents at Entry.	Link	-	-	-	FILLERS	B	A
Contents at Exit.	-	-	-	-	FILLERS	B (add) A-B *(Subtract)	A+B (add) A-B (subtract) or -(B-A) "

* If $/A/ \geq /B/$, 20_{23} contains B on exit.

If $/B/ > /A/$, 20_{23} contains A-B as 10's complement with $P_{30\ 31\ 32}$.

NOTES:

The fillers are constants which control the carries from one unit to the next. Denoting a ten character number as N = a b c d e f g h j k

If	m_1 k = j	m_6 c = d
	m_2 j = h	m_7 d = c
	m_3 h = g	m_8 c = b
	m_4 g = f	m_9 b = a
	m_5 f = e	m_{10}a = next unit. (for out of length detection)

Then the fillers required are:

19_2 $(64 - m_1)P_2 + (64 - m_2)P_8 + (64 - m_3)P_{14} + (64 - m_4)P_{20} + (64 - m_5)P_{26}$

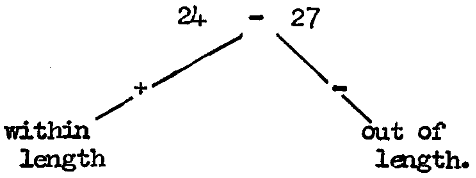
19_3 $27\ P_1 + 54\ P_6 + 54\ P_{12} + 54\ P_{18} + 54\ P_{24}$.

This routine assumes that m_6 to m_{10} are all 10. Another slower routine is available for units other than with radix 10 in the upper five characters.

Failures. NONE.

If the result is out of ten character length P_{30}will be present in 21_3. This may be detected outside the routine by,

21_3 - 14
24 - 14
24 - 27



22

LECTURE 22

INTERPRETIVE ROUTINES.

22.1 WHAT IS AN INTERPRETIVE ROUTINE?

Despite the creation of comprehensive libraries of subroutines and programmes, it is still necessary to possess a considerable knowledge of the DEUCE order code to write a programme to calculate results for a particular problem. The time taken to acquire this knowledge is often the limiting factor in deciding whether to put a problem onto DEUCE for solution.

To offset this disadvantage, completely new order codes have been devised incorporating powerful instructions which reduce considerably the actual number of instructions needed to evaluate the results of any problem. These order codes are completely different from the DEUCE machine order code, and so the interpretive routine is required to convert from one code to the other.

The interpretive routine therefore does the job of converting from a relatively simple order code, which can be learnt in a matter of days, into the far more complex machine code, but at the expense of time. As the programmer is relieved of a lot of the hard work, the interpretive routine has to do this for him, and therefore it would be slower than a conventional DEUCE programme. This is not of any real significance for a "one-off" job as the increased calculation time may not be as great as the programme testing time for a conventional programme, but it should be considered before an interpretive scheme is used for "many-off" jobs.

Let us now consider the various schemes that exist in the DEUCE Library.

22.2 GENERAL INTERPRETIVE PROGRAMME (G.I.P.).

Instructions to this programme are of the 'three address plus function' type and the operation is briefly as follows.

The General Interpretive Programme is fed to the machine and begins by reading in and storing a number (< 63) of self contained programmes which are referred to as bricks. The programmer has previously decided which bricks he will need to do his job, and he chooses these from the library. G.I.P. then reads in its own instructions which consist of 4 parts "a, b, c, r". These instructions are, in general, interpreted as "take brick number r of those which have been read in and stored, provide it with parameters a, b and c and obey it". 16 special values of r are set aside for instruction modification, discrimination and other housekeeping operations, so that all the requirements of a full machine order code are available.

The greatest use of G.I.P. has been for linear algebra calculations and a vast library of bricks has been built for this purpose. The parts a, b and c of the instruction then usually refer to tracks on the magnetic drum, a matrix being specified by the track number at which it begins to be stored. It is important to note that the instructions do not need to make reference to the size of the matrices, the dimensions being automatically stored away with a matrix. As a simple example, to add two matrices together, one would need to read them in, add them and punch out the sum. Thus G.I.P. would be followed by three bricks: "Read Matrix", "Add Matrix", "Punch Matrix" numbered from 1 to 3 respectively. The interpretive programme would then consist of 4 instructions.

a	b	c	r	
0	0	0	1	Obey brick No. 1 (i.e. read a matrix) and store it in track 0 onwards on the magnetic drum. The number in the c position indicates where the matrix is to be stored and in this instruction a and b have no use.
0	0	110	1	Read a matrix into track 110 onwards.
0	110	110	2	Obey brick No. 2 (add matrices) providing it with parameters 0, 110 and 110. The add matrix brick will interpret this as a request to add the matrix at track 0 onwards to that at track 110 onwards and put the result at track 110 onwards, overwriting the one originally there.
110	0	0	3	Obey brick No. 3 (i.e. Punch Matrix) and the matrix to be punched is to be found at track 110 onwards.

This programme would be valid for adding any matrices which occupied less than 110 tracks, i.e. as long as the matrices being added have less than 3516 elements each.

A particular feature of all the bricks made for use with G.I.P. is the elaborate checks on arithmetical accuracy, and reliability of the operator or programmer. Thus any matrix stored on the drum has the sum of all its elements stored with it, and whenever reference is made to a matrix, the sum is checked. Thus the above programme would have a characteristic failure if the data being read in has some inconsistency, if the matrices are incompatible, or if the sum of a matrix written on the drum does not agree with the sum of the elements. (The usual cause of the latter is a programmer unwittingly overwriting part of matrix). Whenever possible all arithmetical operations are checked by a separate method.

Of the special values of r, 5 are for various discriminations, 8 are for instruction modification, and the remainder provide facilities for reading in new bricks, and new instructions.

The scheme has proved of such value in dealing with a wide range of problems that several DEUCES are using it for well over 50% of all production time. Although it has been so extensively used for linear algebra, it has also been widely used for other problems, including the solution of differential equations, (using a brick for RungeKutta), harmonic analysis (using a brick for sine and cosine), curve fitting and many other applications which have no apparent connection with linear algebra.

The interpretation time for G.I.P. is of the order of 1 second per G.I.P. instruction, which is insignificant when dealing with large matrices or arrays. However, if the matrices are small this can be a major part of the total calculating time. In this case, a variation of G.I.P. can often be used. This variation, known as the Special Interpretive Programme or S.I.P. reduces the interpretation time to .4 - .5 second, but loses a little in flexibility.

22.3 TABULAR INTERPRETIVE PROGRAMME (T.I.P.).

This is the simplest imaginable interpretive scheme in which the user looks upon DEUCE as a vast sheet of computing paper ruled into columns, with up to 30 rows ruled across the sheet. Orders to the machine then consist of instructions to perform an operation on all the numbers in one or two columns and write the results in another column.

Thus the instruction

7 18 23 2

has the significance "Add the numbers in column 7 to the corresponding numbers in column 18 and write the answers in column 23, the number 2 being interpreted as "add".

Similarly the instruction

101 0 127 8

has the significance "Take the logarithms of the number in column 101, and list the results in the corresponding rows of column 127". The 8 here is interpreted as "logarithm", and the 0 has no significance.

A number of functions are available including read, print, add, subtract, multiply, divide, sin, cos, $\sin^{-1}$, $\cos^{-1}$, log, exp. Facilities are also available for reading the ordinates of a graph, and for subsequently interpolating in it. It is also possible to feed constants to the machine and to operate with a constant on a column of figures, e.g. to multiply all the numbers in a column by a constant. The operations have been kept to a reasonable minimum in order to encourage scientists and engineers who would not otherwise trouble to learn to programme a machine, to make use of the facilities. For those with problems in a special field however, it is possible to fit special purpose functions into the scheme. It is possible to learn all that need be known about this kind of programming in an hour. As an example it is possible to evaluate the expression

$$\sum_{i=1}^{n} a_i e^{b_i t} \cos (c_i t + d_i)$$

for any n, for up to 32 values of t, and to print out the results at the end, all in 12 instructions.

This type of operation makes it particularly convenient to calculate functions of say, 32 values of a variable in parallel, and this is one of the chief uses of the scheme. Since up to 32 floating arithmetic operations can be carried out for each interpretation of an instruction, the interpretation time is not of paramount importance, and it averages in fact about .8 secs per instruction.

In addition to the arithmetic operations described, the scheme includes facilities for modification of instructions, discriminations, jumps, etc. so that all the requirements of a full machine order code are met.

22.4 DEUCE ALPHACODE.

This scheme was designed to the following specification:

(i) A comprehensive order code is particularly suitable for scientific and engineering use. For instance the functions "Solve Differential Equations", "Sum Power Series", "Interpolate in Graphical Data" and "Integrate by Simpsons Rule" are available in the order code among the 64 possible orders. These orders include all arithmetical, trigonometric and hyperbolic functions, and a wide range of other functions.

(ii) The scheme is simple to learn to programme, and it is possible to train someone to use it in the order of one day. It does not call for any knowledge of orthodox programming, punched cards, scaling or binary arithmetic.

(iii) All instructions are written and presented to the machine in English, avoiding the jargon so frequently associated with autocodes and not even using a numerical notation for the various functions.

(iv) It is possible to insert new instructions at will, without the interference caused in renumbering of instructions in a machine where the instructions are obeyed in sequence, and have an absolute address. Any address specified in jumps or discriminations are such that they cannot be misinterpreted and do not have to be changed if extra instructions are inserted in the programme.

(v) There are particularly easy facilities for programme testing. In particular the results of obeying any specified instruction may be printed out during programme testing, and this print out may be inhibited for later production runs.

(vi) Facilities for instruction modification and counting round loops are particularly easy, and no resetting of counters or instructions is normally called for. This removed a fruitful source of error in programming.

The method of programming is best demonstrated by an example,

Suppose it is required to calculate $(x_1 + x_2)(x_3 - x_4)$ where x_1 to x_4 are given, then the programme is literally,

	Read	4	data into	x_1 onwards		(i.e. read x_1 x_2 x_3 and x_4)
x_5	=	x_1	plus	x_2		
x_6	=	x_3	minus	x_4		
x_7	=	x_5	multiplied by	x_6	P	(where the P indicates "Print").

In addition to the 'three address plus function' nature of these individual instructions it is possible to label any instruction with unique and characteristic number, called a reference, and then any instruction which specifies the reference of the instruction to which the jump is made. For instance the programme

63 $x_1 = 1$ plus x_1

If $x_2 > x_1$ jump to R63

will add 1 to x_1 and the test if $x_2 > x_1$. If x_2 is $> x_1$, it will jump to the instruction bearing the reference of 63, i.e. repeat the cycle. When eventually $x_1 \geqslant x_2$ then the programme continues normally. The interesting point here is that if it is subsequently found necessary to insert an extra instruction between these two, the connection will not upset the validity of the jump instruction or its reference. The relative numbers given in a reference are translated into absolute numbers as the programme is run in.

Subroutines can be readily accommodated without the necessity for planting links. Any sequence of instructions which forms a subroutine is merely headed "S15 Subroutine" (for the 15th subroutine) and terminated "End of Subroutine S15" and the links are automatically supplied. The procedure caters automatically with higher order subroutines.

All operations are carried out in floating arithmetic, with the exception of 63 stores (n_1 to n_{63}) which may only contain integers. These are used primarily for counting in loops, but simple arithmetical operations may be carried out on these numbers. A particularly useful instruction is

"Count n_3 up to n_{10} jumping to R7" which has the effect of adding one to the store n_3, comparing its contents with n_{10}, and if the contents are not equal, jumping to R7. When n_3 eventually contains the same as n_{10}, it is automatically reset to zero, and the programme continues normally.

Instructions are obeyed at a rate varying from 50 per second to much lower speeds in the case of the comprehensive instructions "Integrate", "Solve Differential Equations" or "Sum Series". In most programmes an average speed of 10 to 15 instructions per second will be maintained. Since the instruction code is so economical (saving by a factor of 10 to 20 on the number required for a corresponding orthodox programme) the speed is considerably greater than is at first sign apparent. The obvious use of the scheme is for those who require a quick solution to a 'one-off' problem, in which operations in parallel on 'bulk' data is not convenient. However 'orthodox' DEUCE Programmers also find the scheme of great value not only for 'one-off' or 'few-off' problems, but also for constructing major programmes by obtaining trial results for checking purposes, and perhaps more important for exploring the logical difficulties before the programme is written.

22.5 DEUCE ALPHACODE TRANSLATOR.

Whilst not strictly an Interpretive Programme, the DEUCE Alphacode Translator is worthy of mention as it extends the scope of alphacode considerably.

Having produced a working alphacode programme, the translator programme will read this programme as its data, and from it produce a programme in conventional DEUCE code to perform exactly the same calculation, but without the time loss in the interpretive stages. The resultant DEUCE programme can be up to 10 times as fast as the original alphacode programme, but will take 1½ - 3 times as long as the same calculation programmed in DEUCE code. However, the reduction in programming time and, more important, in programming knowledge, makes a translated alphacode an attractive proposition if results are required quickly or if there is a lack of available programmers.

It should be noted that the alphacode translator will work only if the library alphacode is used. Variations of alphacode incorporating local modification exist at several installations - these are not allowed for in the translater.

22.6 REFERENCES.

G.I.P. DEUCE Programme Report ZC01T/S (No. 475)

T.I.P. DEUCE News No. 28.

DEUCE Alphacode DEUCE News No. 49.

DEUCE Alphacode Translator DEUCE Programme Report ZC23T.
DEUCE News 47 pages 4-7.

GENERAL CONSIDERATIONS OF SORTING.

C22.1 REQUIREMENT FOR SORTING.

On any file updating run, it is necessary to deal with current data (e.g. new transactions, stock movements, or sales) in the same order as the main file or files (e.g. balance, history and reference files). In most cases the input cannot be prepared in the required order, and it therefore has to be sorted. Output from the computer may be of several different kinds, e.g. rejections, queries, invoices, and statistical details, each being put out as it occurs: before any further action can be taken the output must be sorted into these different classes, and any class may be needed for another computer run in which a different order is required.

C22.2 REVIEW OF METHODS.

Basically, there are three methods of sorting, though numerous variations exist. The three are:-

(a) Diverging,

(b) Merging,

(c) Replacing.

The first two are used by punched-card equipment; all are available in a computer sort, but it is merging which finds the most general application.

2.(a) Diverging.

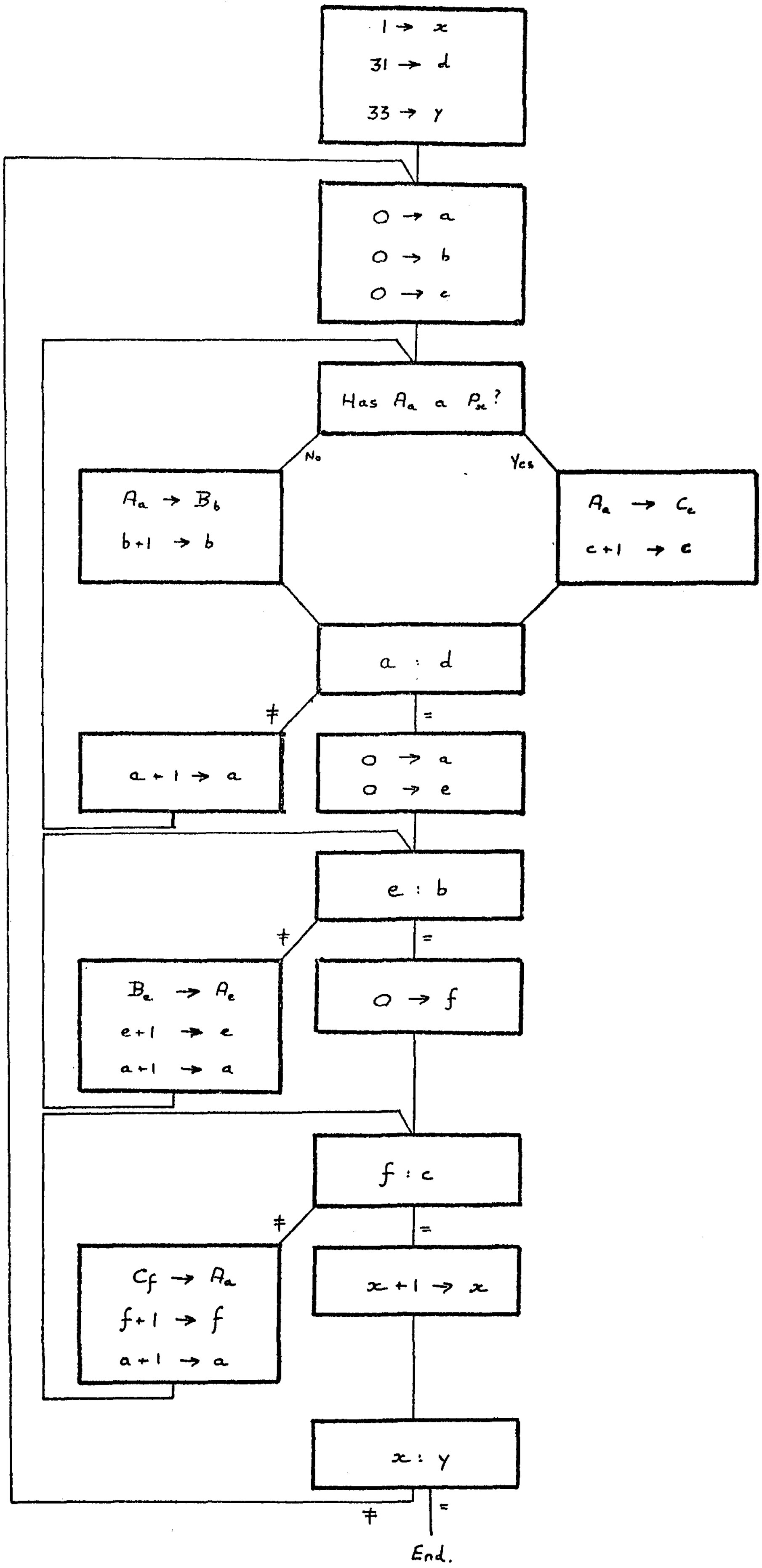
This is the basic method used in a punched-card sorter. The cards are placed face down in the hopper and the machine set to read the least significant digit of the key, placing each card (still face down) in one of ten pockets, according to the value of the digit. At the end of the passage, the cards are collected from the lowest-numbered pocket first, the contents of each pocket being placed on top of the pile. This pile is transferred to the hopper, the next least significant column is set, and the process repeated, until each column of the key has been used.

(i) This is a sort by least significant digit first (L.S.D). The opposite method (M.S.D.) does not lend itself readily to a punched-card sort.

(ii) This is a purely numeric sort. Alphanumeric sorters can be obtained, but their operation is somewhat more complicated.

It will readily be seen that in the computer the above method can be precisely emulated, except that there will be only two "pockets", as each binary digit is considered. The block diagram for a trivial case follows. In this and subsequent block diagrams, capital letters represent areas of the store (in DEUCE each is a Delay Line) and small subscripts represent a part of the area large enough to contain one of the items being sorted. Comparisons of numbers are represented by a colon, and if two items are shown as being compared, it is understood that only the keys are in fact compared. Thus "Aa : Bb" means "Has the item stored in the ath part of area A a key greater than, equal to, or less than, that of the bth item in area B?"

Example 1: Block diagram for sorting 32 items, each of one word in length, stored in Delay Line A.


1 → x
31 → d
33 → y
0 → a
0 → b
0 → c
Has A_a a P_x?
No
Yes
A_a → B_b
b+1 → b
A_a → C_c
c+1 → c
a : d
≠
=
a+1 → a
0 → a
0 → e
e : b
≠
=
B_e → A_e
e+1 → e
a+1 → a
0 → f
f : c
≠
=
C_f → A_a
f+1 → f
a+1 → a
x+1 → x
x : y
≠
=
End.

Notes on Example 1.

(i) The key has been treated as occupying the whole word in each item; in practice it is to be expected that if the item is only one word long, the key would be a few digits merely. If in this example the key was contained at P21-27, then x would be set at 21 and y at 28. More will be said in Section 3 (c) about the range of keys.

(ii) The routine is an exact simulation of the card sorter operation. It is easy to see that economies can be made both in the storage space used and in transferring items to and fro between the Delay Lines, although the organisation involved will be somewhat more complicated. In a M.S.D. sort, for example, the technique would be to examine the first word in the D.L.; if the most significant digit of its key is zero, leave it in the first place, and examine the second; if non-zero, place it at the end of the D.L., first extracting the word already there and now examining it in its turn. When all 32 words have been placed, the D.L. effectively has two sections, with smaller keys in the first and larger in the second. The process can be repeated on the next most significant digit, but now treating each section separately, and thereby creating from smaller sections, in ascending order as compared with each other but not in themselves. When all the digits of the key have been dealt with the sort is complete.

Exercise 1.

Draw the block diagram forsorting 32 items, one per word in D.L.A., each with its key at P1-6, without moving any items into another D.L. (Use of the short stores is permitted. The M.S.D. method should be used).

It is evident that more than one digit of the key can, in general, be used at a time: using two digits would require four "pockets", and n digits at a time would require 2^n pockets. The difficulty is that each pocket must be capable of containing the whole of the items to be sorted, unless it is in the nature of the data that the keys are spread fairly evenly over their possible range. In the extreme case, i.e. that where no two items have the same key and no possible key value is unused, one is able to use the extreme form of diverging sort, namely the "pigeonhole" method (provided always that there is sufficient storage space in the machine: the method would not be feasible on magnetic tape). Here each storage address is, as it were, "labelled" for a particular key, and when the key is found the item is sent to that address. In computer terms this means that each item's destination address is calculated from its key; hence this method is often called "Address Calculation".

Where the extreme condition of complete correspondence between actual and possible key values is not attained, "Approximate Address Calculation" is still available. This method, which needs a certain amount of spare storage space, calculates an address from the key; if this address is already occupied, the item is placed in an adjacent address. Some shuffling may be necessary in parts of the store which become overcrowded: this may be done either as the sort proceeds, or when gathering up the data at the end. Some overflow arrangements must also be made. As may be imagined, the method is highly complicated to programme, for which reason we cannot go more deeply into it here. Nevertheless, where the right conditions (of (i) near correspondence of actual to possible key-values, and (ii) about 150% spare storage space) apply, very good results are obtained by this method.

2.(b) Merging.

This is the basic method used in a punched-card collator. Two inputs are needed, and the keys of the two cards currently available are compared, the card containing the smaller key being passed through. (Alternatively the machine can be set to take the larger key where descending order is required). Thus two packs each in order in itself are brought together into one completely ordered pack. Where a pack is not in order in itself, it may nevertheless be considered as consisting of several smaller packs which are, although in some cases these "packs" may consist of one card only. If we call each of the "packs" in which complete order exists "strings" then it is clear that by repeated merging the strings can be lengthened (and the number of strings in the whole pack reduced) until the whole pack forms one string, i.e. is completely in order. In practice this collator method would not normally be used for data commencing in random order, but for the computer merging is very often the most suitable method even in this case.

In the "fixed string" method the initial data is considered to consist of strings of one item each, disregarding any order already existing. At the first pass the leading item from one input is compared with that from the second input; the lower is placed first on the output and the higher second; the next two input items are treated similarly, and so on. As the output will have to be split anyway to form two inputs to the next pass, it is convenient to use two outputs, switching from one to the other at the end of each string. These two outputs, now consisting of strings of two items each, become inputs to the next pass, in which strings of four items each can be created.

Example 2.

Original Input.		After 1st pass.		After 2nd pass.		After 3rd pass.	
7 •••	3 •••	3	4	3	1	1	5
4 •••	5 •••	7 •••	5 •••	4	2	2	6
1 •••	8 •••	1	2	5	6	3	7
6 •••	2 •••	8 •••	6 •••	7 •••	8 •••	4	8

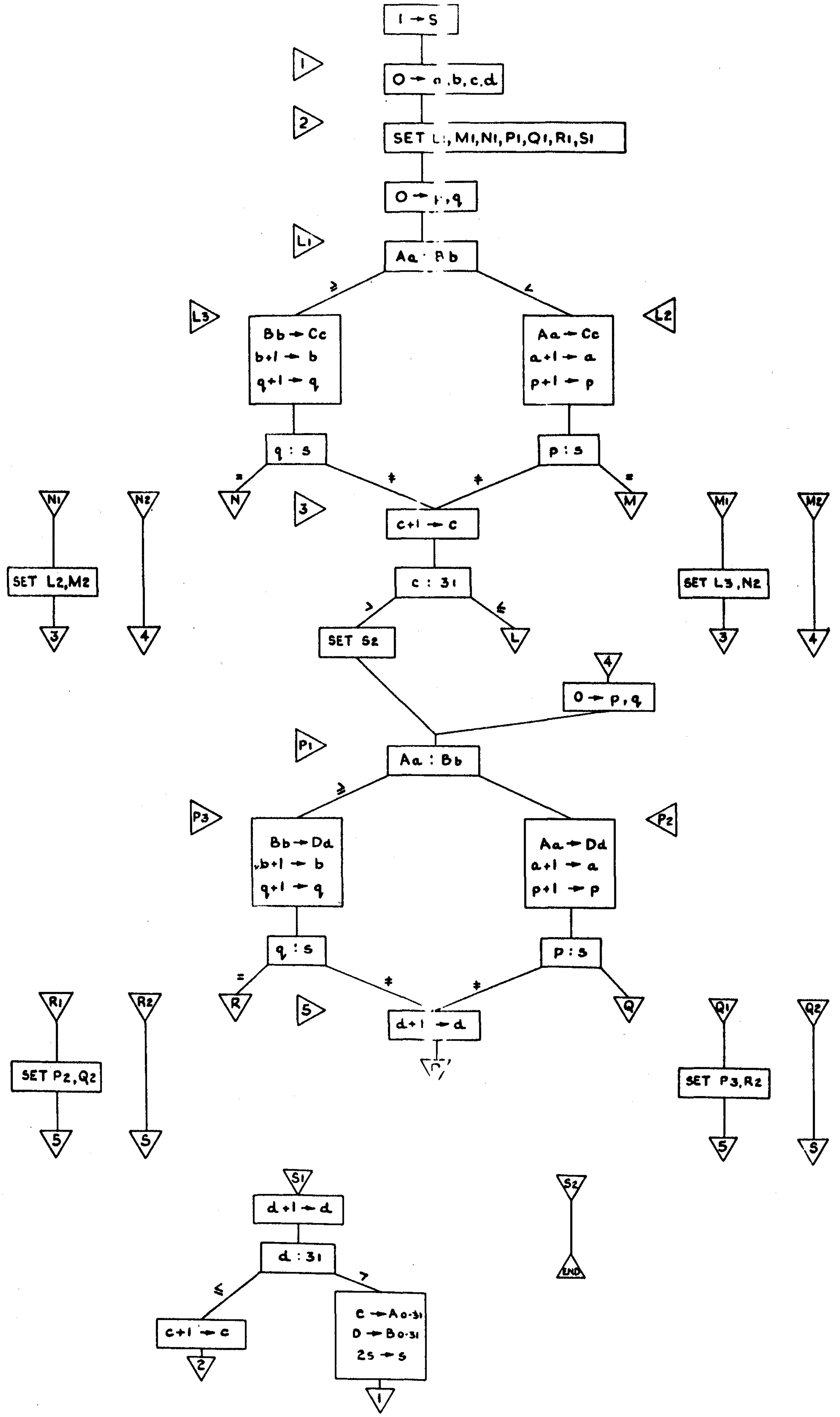
Notes on Example 2.

(i) Notional ends of strings are marked by dotted lines in this illustration; a physical marker is not, in fact, necessary, although we shall see later that it is in certain cases convenient.

(ii) With 8 items to be sorted, 3 passes were necessary: in general $\log_2 N$ passes are necessary where N is the nearest number which is a power of 2, above the number of items to be sorted.

Example 3.

Block diagram for a merging sort of 64 items, stored one per minor-cycle in D.L's. A and B, with the keys at the most significant end of the words. The fixed-string technique is used, and the results will appear in D.L.'s C and D.
(Here the capital letters L,M,N,P,Q,R,S are links or switches: other letters have the same significance as in the previous example).


1 → S
1
0 → a,b,c,d
2
SET L1, M1, N1, P1, Q1, R1, S1
0 → p,q
L1
Aa : Bb
≥
<
L3
Bb → Cc
b+1 → b
q+1 → q
L2
Aa → Cc
a+1 → a
p+1 → p
q : s
=
≠
N
p : s
≠
=
M
3
c+1 → c
N1
SET L2,M2
3
N2
4
M1
SET L3,N2
3
M2
4
c : 31
>
≤
SET S2
L
4
0 → p,q
P1
Aa : Bb
≥
P3
Bb → Dd
b+1 → b
q+1 → q
P2
Aa → Dd
a+1 → a
p+1 → p
q : s
=
≠
R
p : s
≠
Q
5
d+1 → d
R1
SET P2,Q2
5
R2
S
Q1
SET P3,R2
5
Q2
S
S1
d+1 → d
d : 31
≤
>
c+1 → c
2
C → A0-31
D → B0-31
2s → s
1
S2
END

Notes on Example 3.

(i) The diagram should be easy enough to follow by reference to actual numbers, e.g. those in example 2.

(ii) It will be noted that there are considerable similarities between the first and second halves of the diagram; in fact several instructions are duplicated. It is obviously possible to combine the two halves, using suitable links, thereby reducing the number of instructions.

Exercise 2.

Redraft the diagram of example 3, to reduce it to a simpler form.

In the "variable string" method advantage is taken of any tendency towards the required order, existing in the input data. It does this by forming strings as long as possible, finding by comparison the actual end of each string, instead of counting as in the "fixed string" method.

Example 4.

Original Input.		After 1st pass.		After 2nd pass.		After 3rd pass.
7 •••	3	3	2	2	1	1
4 •••	5	5	4 •••	3	6 •••	2
1	8 •••	7		4		3
6 •••	2 •••	8 •••		5		4
		1		7		5
		6 •••		8 •••		6
						7
						8 •••

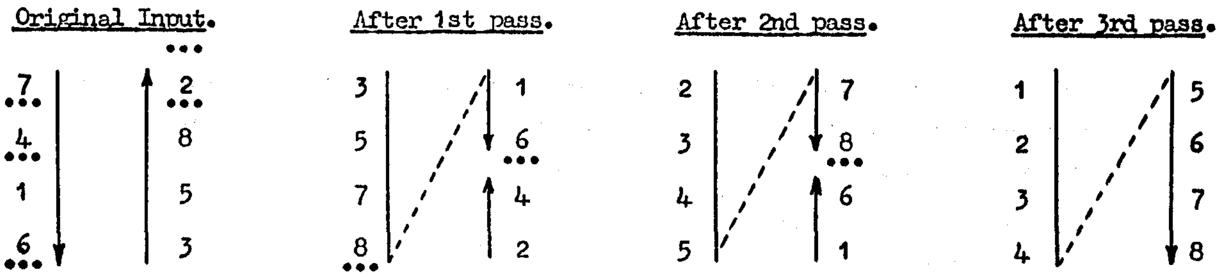
Notes on example 4.

(i) Actual string-ends are indicated by dotted lines; again in our larger examples to follow we shall be able to make use of end-of-string markers, but for the moment they are not necessary.

(ii) With 5 strings initially, 3 passes were necessary; in the general case $\log_2 N$ still applies, but N refers to the number of strings, not as before to the number of items.

(iii) It would appear that each storage area needs to be large enough to contain all the items being sorted; in practice the technique for avoiding this requirement is to work from and to opposite ends of the areas, so that an overflow can be accommodated.

Example 5.


Original Input.

After 1st pass.

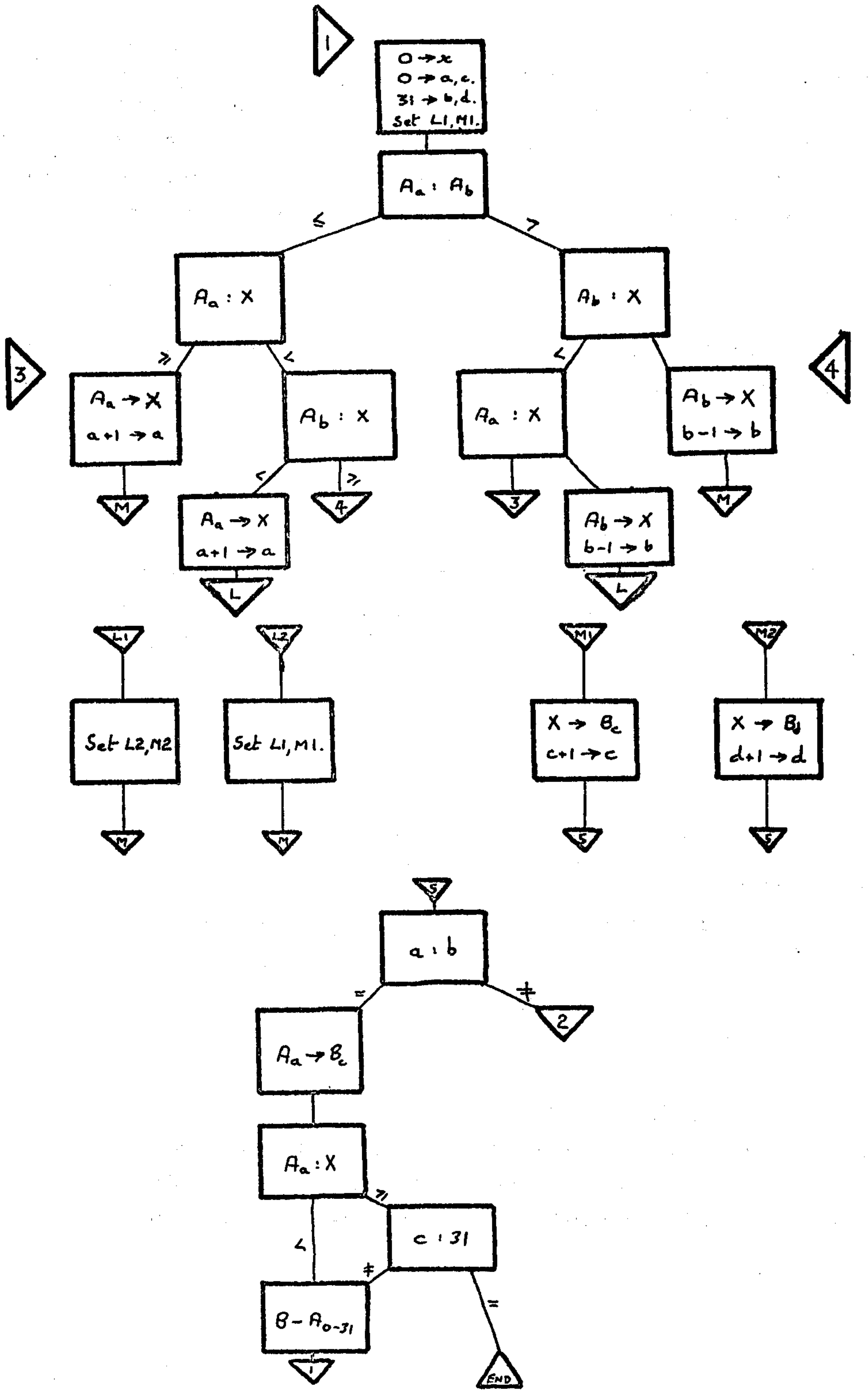
After 2nd pass.

After 3rd pass.

Example 6.

Block diagram for a merging sort of 32 items of 1 mc each in D.L. A, using variable-string technique.

(X represents a short store: other symbols as previously used).



Notes on Example 6.

(i) The task has been simplified by having only one D.L. to sort; if there had been 64 items so that we were working from and to opposite ends of two D.L's. the overflow arrangements would have added complications.

(ii) This method admits of ready extension to larger sorting tasks: working from and to opposite ends of the drum is feasible.

So far we have been considering merging from two inputs to two outputs, i.e. a "2 - way" merge. Clearly the same principles can be extended to a 3 - way merge (3 inputs, 3 outputs forming inputs for the next pass), and in this case our formula for the number of passes necessary becomes $\log_3 N$. Though the number of passes is reduced, as compared with 2 - way, the sequence of instructions is somewhat more complicated; the same applies to each further extension to 4 - way, 5 - way, etc.

A further variation is introduced by the "cascade" merge, particularly applicable to magnetic tape sorting. This merges data from several tapes on to one, the quantity on each tape at the beginning being so arranged that each in turn of the input tapes is exhausted at a strategic point and is rewound to form the new output tape, the previous output tape rewinding simultaneously to form a new input tape. As will be seen later, before a tape sort we will normally have created quite long strings by internal and drum sorting.

Example 7.

Suppose that when starting the tape sort, there are nine strings, each of 1,000 items, which have been laid out as follows:-

Position A.	Tape 1 :	4 x 1,000
	Tape 2 :	3 x 1,000
	Tape 3 :	2 x 1,000
	Tape 4 :	NIL.

It is clear that the first string of 1,000 from each of the three tapes can be merged into a single string of 3,000 on tape 4; similarly with the second strings, so that when tape 3 is exhausted the position (disregarding the items which have been dealt with) is:-

Position B.	Tape 1 :	2 x 1,000
	Tape 2 :	1 x 1,000
	Tape 3 :	-
	Tape 4 :	2 x 3,000

Tapes 3 and 4 have to be rewound, and now tapes 1, 2 and 4 are the input tapes and tape 3 the output. The first string of 3,000 from tape 4 can be merged with a string of 1,000 each from tapes 1 and 2, giving the following position:-

Position C.	Tape 1 :	1 x 1,000
	Tape 2 :	-
	Tape 3 :	1 x 5,000
	Tape 4 :	1 x 3,000

After the rewind, these can be merged in one operation to complete the sort. The transition from position A to position B involved the transfer (tape-to-tape) of 6,000 items; from B to C 5,000, and from C to completion 9,000, a total of 20,000 item-passages.

In tabular form:-

Table 1.

Position.	Tape 1.	Tape 2.	Tape 3.	Tape 4.	No. of item - passages to reach this position from last.
A	4 x 1,000	3 x 1,000	2 x 1,000	-	-
B	2 x 1,000	1 x 1,000	-	2 x 3,000	6,000
C	1 x 1,000	-	1 x 5,000	1 x 3,000	5,000
END	-	1 x 9,000	-	-	9,000
					20,000

How do we arrive at our opening strategic values of 4,3 and 2? This can be discovered by working backwards from the end and analysing into the number of strings on each tape. At each stage, the strings on one tape (the one chosen being that with the greatest number of strings) are dissected into that number of strings on each of the other tapes, which are of course added to those already there. Thus:

Table 2.

Stage.	Number of strings.				
	Tape 1.	Tape 2.	Tape 3.	Tape 4.	Total.
1	1	0	0	0	1
2	0	1	1	1	3
3	1	0	2	2	5
4	3	2	0	4	9
5	7	6	4	0	17
6	0	13	11	7	31
7	13	0	24	20	57

and so on.

In example 7, the total number of strings originally was 9, so from the table it can be seen that these must be mounted 4,3 and 2. But in practice our opening string total may not correspond with a value in the total column of Table 2. What then? It has been found that the best method is to build up to the next higher value with dummy strings. i.e. strings each of length 0 items: these dummies should be allocated as early as possible equally between the tapes. Suppose that we had started with 13 strings of 1,000 items each, Table 1 would become:-

Table 3.

Position.	Tape 1.	Tape 2.	Tape 3.	Tape 4.	Item - passages.
A	2 x 0 5 x 1,000	1 x 0 5 x 1,000	1 x 0 3 x 1,000	-	-
B	3 x 1,000	2 x 1,000	-	1 x 0 1 x 2,000 2 x 3,000	8,000
C	1 x 1,000	-	1 x 2,000 1 x 4,000	2 x 3,000	6,000
D	-	1 x 6,000	1 x 4,000	1 x 3,000	6,000
END	1 x 13,000	-	-	-	13,000
				Total	33,000

So far we have used four tapes in illustrating the method, thus obtaining a "three-to-one" cascade. Again this can be extended to any "N-to-one" where N is greater than one.

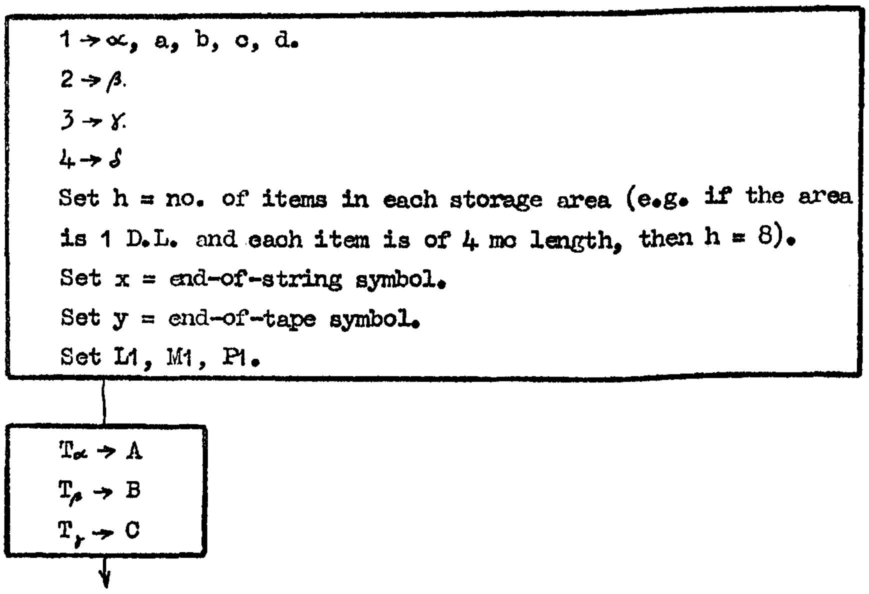
Exercise 3.

Construct the equivalent of Table 2 where 6 tapes are available, i.e. for a five-to-one cascade.

If the data of Table 3 were sorted by an ordinary 2-way (two-to-two) merge (3-way or more cannot be accommodated on only four tapes) we would expect 4 passes to be needed, for $\log_2 13$ lies between 3 and 4. At each pass, every item is transferred, so the number of item-passages would be 4 x 13 x 1,000 = 52,000. The cascade method shows an appreciable improvement, to 33,000. In general for any significant number of strings the cascade method will require considerably fewer item-passages than the N-way merge available on the same number of tapes.

Example 8.

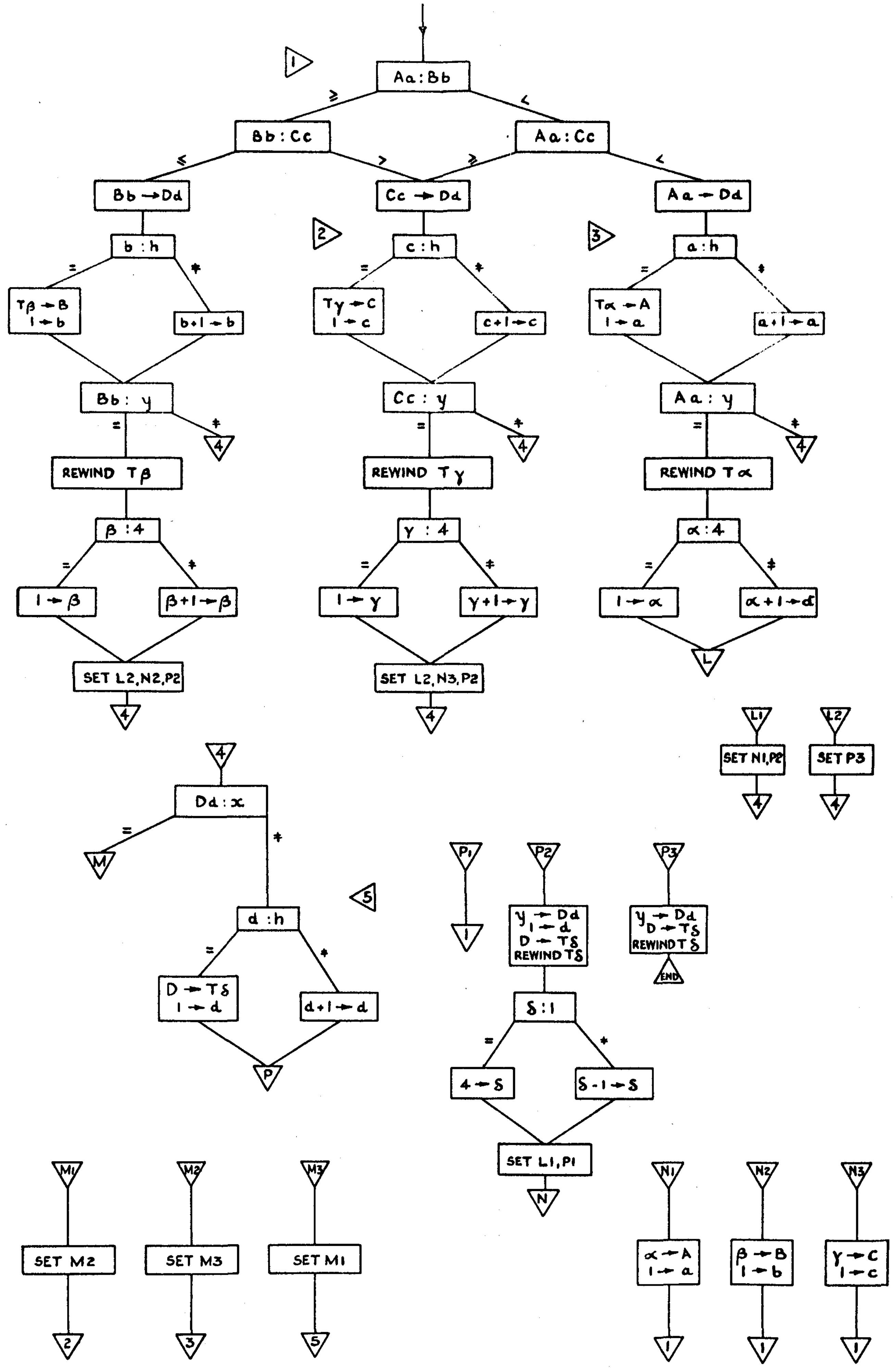
Four-tape cascade sort. The items are already in strings, each concluding with an "end-of-string" marker, i.e. a dummy item with a key larger than any possible key in the data. In addition each tape must be concluded by an "end-of-tape" marker, i.e. another dummy whose key is different from any possible key. It is assumed that the data is on tapes 1, 2 and 3 and has been brought up to the appropriate strategic number of strings by the addition of "end-of-string" dummies at the beginning of each tape. Each block (except the last) on tape must correspond in length with the area of the store in which it is to be worked (in DEUCE this would normally be 1 D.L.)
(In this example "T " means "tape number "; other symbols as previously used).



$1 \rightarrow \alpha$, a, b, c, d.
$2 \rightarrow \beta$.
$3 \rightarrow \gamma$.
$4 \rightarrow \delta$
Set h = no. of items in each storage area (e.g. if the area is 1 D.L. and each item is of 4 mc length, then h = 8).
Set x = end-of-string symbol.
Set y = end-of-tape symbol.
Set L1, M1, P1.

$T_\alpha \rightarrow A$
$T_\beta \rightarrow B$
$T_\gamma \rightarrow C$

Enter main loop (see next page)


1
Aa:Bb
≥
<
Bb:Cc
Aa:Cc
≤
>
≥
<
Bb → Dd
Cc → Dd
Aa → Dd
2
3
b : h
c : h
a : h
=
≠
Tβ → B
1 → b
b+1 → b
Tγ → C
1 → c
c+1 → c
Tα → A
1 → a
a+1 → a
Bb : y
Cc : y
Aa : y
4
REWIND Tβ
REWIND Tγ
REWIND Tα
β : 4
γ : 4
α : 4
1 → β
β+1 → β
1 → γ
γ+1 → γ
1 → α
α+1 → α
SET L2, N2, P2
SET L2, N3, P2
L
L1
L2
SET N1, P2
SET P3
4
Dd : x
M
5
d : h
D → Tδ
1 → d
d+1 → d
P
P1
P2
P3
1
y → Dd
1 → d
D → Tδ
REWIND Tδ
y → Dd
D → Tδ
REWIND Tδ
END
δ : 1
4 → δ
δ-1 → δ
SET L1, P1
N
M1
M2
M3
SET M2
SET M3
SET M1
2
3
5
N1
N2
N3
α → A
1 → a
β → B
1 → b
γ → C
1 → c
1

Notes on example 8.

(i) This important method is worthy of close study and should be thoroughly understood.

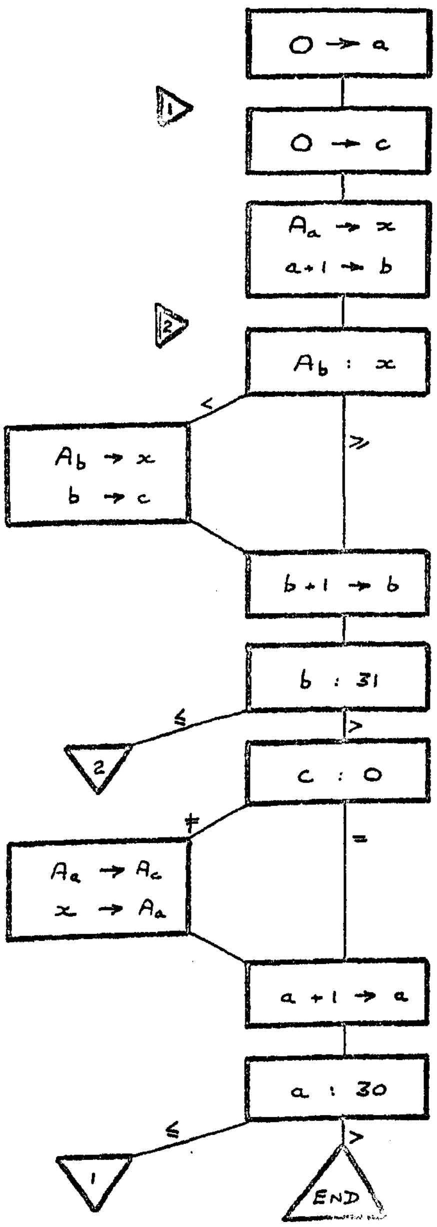
(ii) The striking thing about this diagram is its simplicity (it is simple compared with other sorts which involve tape organisation). This is chiefly due to the inherent simplicity of the cascade method, but there are also some points of technique involved. Note particularly the use of end-of-string and end-of-tape markers, and how these are reproduced on the output tape. Other points will be brought out in section 4.

2.(c) Replacing. (Variation (i))

The simplest example of replacement method is that in which a series of numbers are examined to find the smallest, which then changes places with the first. The remainder are scanned and the smallest of these changes place with the second; and so on.

Example 9.

A simple block diagram for sorting the 32 numbers in D.L. A by this method.



This variation takes too long to be of much practical value, but there are some more sophisticated ones which are sometimes useful for sorting in the high-speed store. A feature of replacement methods is that little or no additional storage space is needed than is already taken up by the data.

Variation (ii).

Compare the first item with the second, and if they are in the wrong order, interchange them. Compare the (new) second with the third, interchanging if required, and so on to the end. Now start afresh, unless no interchanging has taken place, in which case the sort is complete.

This also is rather slow as several passes may be needed, but an improvement can be obtained by including the rule: whenever an interchange takes place, compare the item moved up with the one above it, interchanging if necessary and comparing again until no interchange takes place; then resume where you left off. In this way the whole sort comes out in one passage.

Variation (iii).

Compare the first with the $(n + 1)^{th}$ where n is first set at half the number of items; interchange if necessary; compare second with $(n + 2)^{th}$ and interchange if required; continue till last item has been examined. Reduce n by half, and provided it is not less than 1, restart. When n becomes less than 1, the sort is complete. (Special rules are needed where the number of items is not a power of 2).

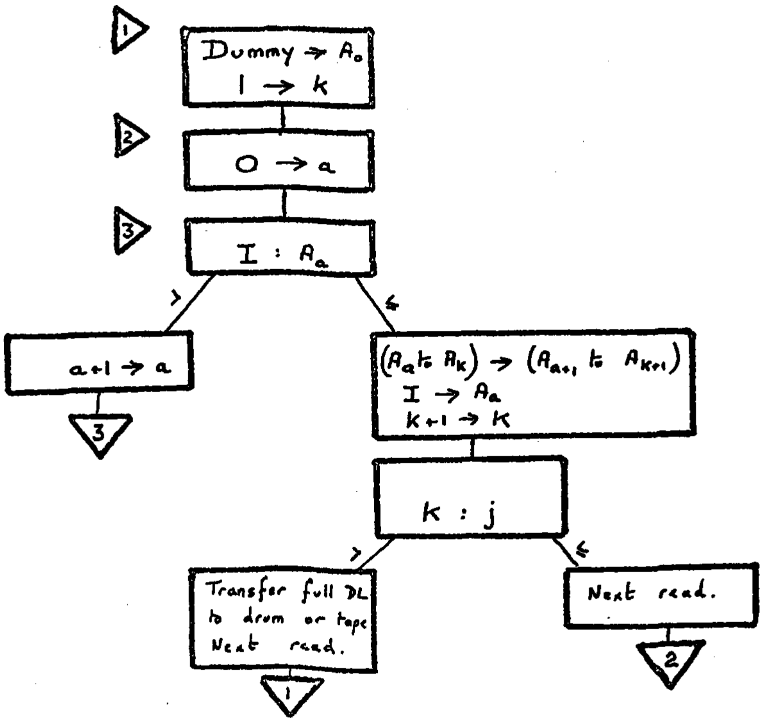
Again this is slow unless it is improved in the same way as Variation (ii), by allowing an item which has moved up to go further up if required in the same passage.

Insertion. In this method an incoming item is compared with each in turn of the items already dealt with; when its place is found the remainder are pushed down one place to leave room for this item to be put in. Its main use is when items are coming in singly, e.g. from card reading, and each can be dealt with during the card cycle; this method is then preferable to one which requires a batch of items before anything can be done and then has to interrupt the reading while the sort takes place.

Example 10.

To insert items coming in singly, into a D.L.

Set j as the number of items the D.L. can contain.
Let each incoming item be called I.
The "dummy" must be larger than any possible key.



Notes on example 10.

(i) This is quite a simple subroutine on DEUCE, because the D.L. used is No. 10 and T.C.A. is used to make the transfer (Aa to Ak) → (Aa + 1 to Ak + 1).

(ii) This transfer must not be cyclic, i.e. an item in mc31 must not come round to mc0. As will be seen, the dummy item is thus, as it were, pushed off the end of the D.L. when the last item to fill the D.L. is inserted.

C22.3 FACTORS INFLUENCING CHOICE OF METHODS.

It is assumed that the method chosen will be that which completes the sort most quickly, with ease of programming a secondary consideration to decide close cases.

Up to this point nearly all that has been said is applicable with but slight modification, to any computer. What follows is, however, specific to DEUCE.

3.(a) Item Size.

The most manageable sizes of item are one, two or four minor-cycles. This is because, although in block diagrams we blithely send Aa to Bb, we know that on DEUCE (e.g.) $9_{15} - 10_{22}$ is impossible, and we must do (e.g.) $9_{15} - 16$ and $16 - 10_{22}$. Even with four-word items certain difficulties are encountered in moving items through the short stores; for example $9_{0-3} - 17$ must go to its next instruction in m.c.3, so that if on successive visits to a point in the loop we wish to do $9_{0-3} - 17$, $9_{4-7} - 17$, $9_{8-11} - 17$, etc., we must provide a fresh instruction (in m.c. 3,7,11 etc) each time. At first sight the same applies in the reverse case 17 - 10 (x to x + 3) but most of the time a technique which can be used to overcome this is to do $17 - 10_{x-31}$ where x takes successive values 0, 4, 8, etc. Extra copies are obtained throughout the D.L. but are over-written in due course by the new items.

Therefore where items are more than four words long, some method which involves the minimum of transfers, such as Approximate Address Calculation, will commend itself; or if merging is chosen an effort will be made to use 4-way or even more despite the programming complications, so that the number of passes is minimised.

Where items are of variable length, it may pay to go straight into a tape sort, because such items are difficult to handle on the drum; but sometimes a special technique (see 4 (a)) may be available.

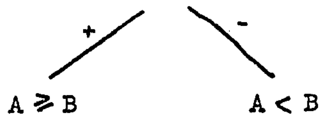
3 (b). Key Length.

Merging and replacing sorts are virtually unaffected, within limits, by the length of keys, for the process is one of comparison which must be done by arithmetics:-

A - 13

B - 26

13 - 27



and it makes no difference whether the key has 1 digit or 31. Indeed if the key has up to 63 digits it merely means that the comparison is made in DS 21 instead of TS 13.

In diverging, on the other hand, the number of passes required is in direct proportion to the number of digits in the key; hence such a method would be most unattractive where long keys were in evidence.

3 (c). Key range.

It has already been said that Address Calculation is likely to be the appropriate method where there is correspondence between actual and possible key values. For example on the input to a payroll run we might expect one item for each employee on the books: no key unused, no key used twice.

Similarly, where near correspondence exists, the method of Approximate Address Calculation commends itself. For example, on a share call payment accounting run, we would expect on the due date that most shareholders would have made one payment and few two payments; so that given a small amount of spare space on the drum to allow for the few cases, all the payments would be sorted in one passage. Even where the correspondence is not so near as in this example the method can be quite powerful if given rather more spare storage space.

3 (d). Quantity of data.

Where the quantity is small enough to be sorted internally (i.e. on drum and mercury alone) the choice will be made according to the above considerations; but where tape sorting is necessary merging will almost certainly be the most attractive method. Only in the case of a very short key (say about 12 binary digits) will diverging be quicker, and a short key is unlikely in so much data. Replacing and Address Calculation are hardly feasible at all on tape, due to the long random access time.

It has already been shown that among the tape merging sorts the cascade method invoves the fewest item-passages and its organisation is if anything simpler than that of other merges, once it is under way. The only point on which the cascade loses out is that it demands special assembly of the strings being put on tape before starting its programme. However the technique of this assembly is not so difficult as may appear. Therefore in my opinion cascade merging is likely to be accepted as the universal tape sorting routine.

C22.4 TECHNIQUES OF SORTING.

This section(indeed the whole lecture)is by no means comprehensive, but is merely intended to pass on a few ideas which may be found useful.

4 (a). Special Techniques.

Where items are very long (probably about 9 m.c. in the critical point, but this can only be determined by reference to the specific job) it pays to send them straight to the drum, extracting from each its key and storing this with the address to which the item has been sent. Then these keys-plus-addresses can be sorted, and the items fetched ih order from their addresses.

Where items are of variable length it will sometimes be convenient to treat them as of the same length but providing continuation facilities. Suppose, for example, that most items are 3 or 4 words but some are more, up to a maximum of 8. Then it would be convenient to treat them as 4-word blocks; the programme would be written to recognise a continuation marker (e.g. a P32 in the 4th word) as showing that the following block was a continuation of this and must be transferred with it.

One technique which the block diagrams may help to bring out is that of choosing carefully the means to exit from numerous loops, and in particular in finding out when the sort is complete. By a careful examination of the method adopted in any particular case, it can often be found that some feature is naturally present only at the end of a pass, or only at the end of the sort, so that a ready check is available to find the way out. In other cases it may be necessary to kick over some trigger, e.g. when an interchange is made, and to check on the state of this trigger to find out when the sort is complete.

4 (b). Combining Methods.

More highly sophisticated sorting systems can be devised by combining the basic methods outlined. For example in a multi-way merge it is required as each fresh item is introduced to compare its key with that of the several other leading items to select which is next to be placed on the output string (and, in the variable-string technique, to compare with the item last put out). The natural process is to keep the leading items in order amongst themselves, using an insertion technique to place each fresh entrant, and many fruitless comparisons eliminated. So this is an insertion-cum-merge. Many more examples will occur to the student once he is familiar with the basic methods.

4 (c). Balancing the Use of Mercury, Drum and Tape.

Usually the procedure for any major sorting operation will be:-

(i) Read a batch of data into the high-speed store, sort it there, and write it in one string on the drum; repeat this process till the drum is filled to the requisite degree (most often this will be half-full);

(ii) Perform a drum sort (most often by a 2-way merge) until all the data there can be written in one string on to tape; repeat steps (i) and (ii) till all data has been read.

(iii) Perform a tape sort (probably using a cascade merge).

During stages (ii) and (iii) transfers can be made concurrently with computer working, provided that sufficient high-speed storage can be found to act as a kind of reservoir. This is difficult on a Mark II DEUCE; for a 2-way merge or a 3-to-1 cascade from D.L's. four are needed to work in, plus D.L. 11 for drum transfers or D.L. 9 for tape transfers: the remaining seven D.L's. will surely be almost completely occupied by programme. However on a Mark II A space is available, and should be used, to absorb the transfer times, even though the programming organisation becomes quite complicated. The programme must always be looking ahead to see what transfer is next required, and positioning the drum heads or calling down the appropriate track or tape block, in advance of actual requirements; it must be able to ascertain the current state of transfers and as there is no TIL-like signal but only an interlock (which it cannot invoke without loss of time), its timings must be very carefully calculated.

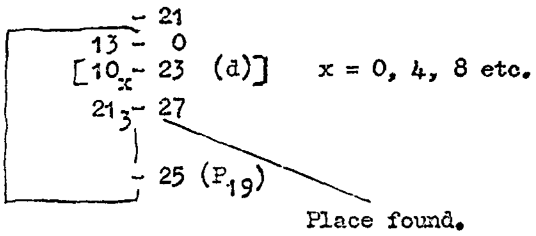
In tape working, it will sometimes pay to use the drum as a reservoir, for this permits longer block sizes to be written on tape, consequently reducing the incidence of tape start-stop time.

4 (d). Calculation of Times.

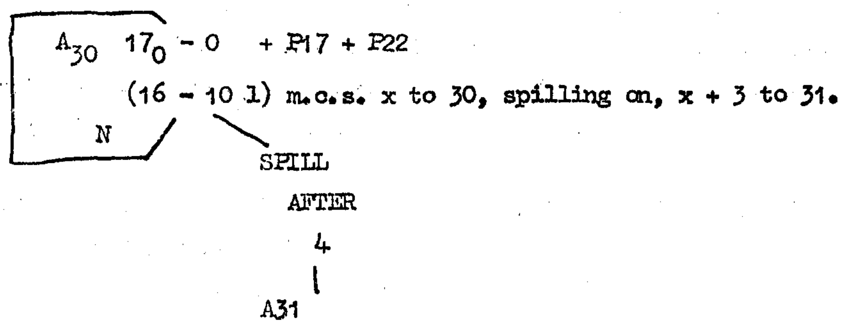
This is a most important subject, yet one on which it is almost impossible to lay down general rules: an attempt will be made to show the method of calculation, however, for a sort using the procedure of 4 (c)

(i) It is quite likely that the data is being read in from cards. In this case, the method of high-speed sort and the batch size will be chosen, if at all possible (and generally it will be) so as to be accomplished, together with any other work (e.g. conversion to binary) that needs to be done on the items, within the card cycle time.

Suppose that the items are of four-word length, including 2-word key, and that an insertion technique is used. First, compare the key of the incoming item with each in turn of those already stored until its place is found:-



Each time through this loop must cost 2 major-cycles. The first item will go once through it (being compared with the dummy), the second item either once or twice (average $1\frac{1}{2}$), the third item once, twice, or thrice (average 2) and so on till the 8th and last. The average for any one of the eight then, is $\frac{1}{8}$ $(1 + 1\frac{1}{2} + 2 + 2\frac{1}{2} + 3 + 3\frac{1}{2} + 4 + 4\frac{1}{2})$ = thrice, so we can say that on the average 6 m.s. will be spent on this part. Next after modifying the appropriate instructions by reference to 13 we must move down the items after the required place to make room for our new item:-


A_{30} 17_0 - 0 + P17 + P22
(16 - 10 1) m.c.s. x to 30, spilling on, x + 3 to 31.
N
SPILL
AFTER
4
A31

Each time through this loop is usually 1 m.s. but sometimes 2 m.s. and can be averaged at $1\frac{1}{8}$, so that four times through costs $4\frac{1}{2}$ m.s. A further 2 m.s. is needed for transferring the new item through a Q.S. to its appointed place, and about $2\frac{1}{2}$ m.s. more for general organisation and instruction time. Hence total time per item is $6 + 4\frac{1}{2} + 2 + 2\frac{1}{2} = 15$ m.s. The transfer to drum can proceed in parallel.

(ii) Suppose that we build up half a drum-full before going to the second stage, we shall have 1024 items in strings of 8. A two-way merge will require $\log_2 \frac{1024}{8} = 7$ passes, or 7,168 item-passages. The average time per item-passage can be calculated on the same lines as above, but allowance must also be made for drum transfer times, when these cannot be absorbed. In practice a figure of about 10 m.s. per item-passage can be attained on a Mark II DEUCE, so that the drum-sort time for each such batch is approximately 72 seconds. To this should be added the time for transferring the resulting string to tape. Let us assume that in the next (tape sort) stage we shall not be able to make concurrent transfers due to lack of storage space, but instead we plan to use the drum as a buffer, and thus deal with tape blocks of length 4 tracks, i.e. 32 items. There are 32 such blocks to be written now, each taking 88 m.s. (assuming binary write) + 30 m.s. start-stop time. 32 x 118 m.s. = 4 seconds approx.

(iii) Steps (i) and (ii) having been taken alternately, we can now calculate the number of item-passages which will be needed in the cascade sort, by the methods indicated in section 2 (b). The time for the passage of each block (of 32 items) is calculated thus:-

Read from tape to drum (88 M.S. + 15 m.s. start time)	103 m.s.
Transfer each track (8 items) as required to high-speed store (4 tracks @ 17 m.s. each)	68 m.s.
Sort in high-speed store, 32 items @ 5 m.s. each	160 m.s.
Transfer results as formed from high-speed store to drum	24 m.s. *
Write from drum to tape	103 m.s.
	458 m.s.

* An average, since transfer to drum can proceed in parallel with other work, and the machine will have to wait for its completion only when shortly afterwards it is required to make a transfer from the drum.

Hence the time for this stage = No. of item-passages x $\frac{458}{32}$ m.s.

REFERENCES

The following articles, in particular the first, are recommended for the student who wishes to examine sorting procedure in more detail:-

(1) E. H. FRIEND. Sorting on Electronic Computer Systems. Journal of the Association for Computing Machinery, Vol.3, 1956,pp 134 - 168

(2) H. H. SEWARD Information Sorting in the Application of Electronic Digital Computers to Business Operations, Master's thesis, Massachusetts Institute of Technology 1954.

(3) D. L. SHELL A High - Speed Sorting Procedure, JACM Vol.2 No. 7 1959,p.30.

(4) E. J. ISAAC and R. C. SINGLETON
Sorting by Address Calculation. Presented at meeting of ACM Sep 1955.

(5) BENJAMIN L SCHWARTZ
Criteria for Comparing Methods of Sorting. Presented at meeting of ACM, Sep. 1955.

(6) WRIGHT AIR DEVELOPMENT CENTER
Mathematical Analysis of Sorting Procedure.

23

SIMPLE ALPHACODE PROGRAMMING

1. INTRODUCTION.

It is now well known that computers can be very useful to engineers, scientists and business men who have a lot of dull repetitive arithmetic to do or who would like to do this arithmetic for design work but have neither the time nor the resources to do it. What is not so well known, perhaps is that in order to make computers do this work a programme of instructions has to be written which are so basic and full of details that the very labour of writing a programme to do the sums required may be as large as the work involved in doing the sums themselves.

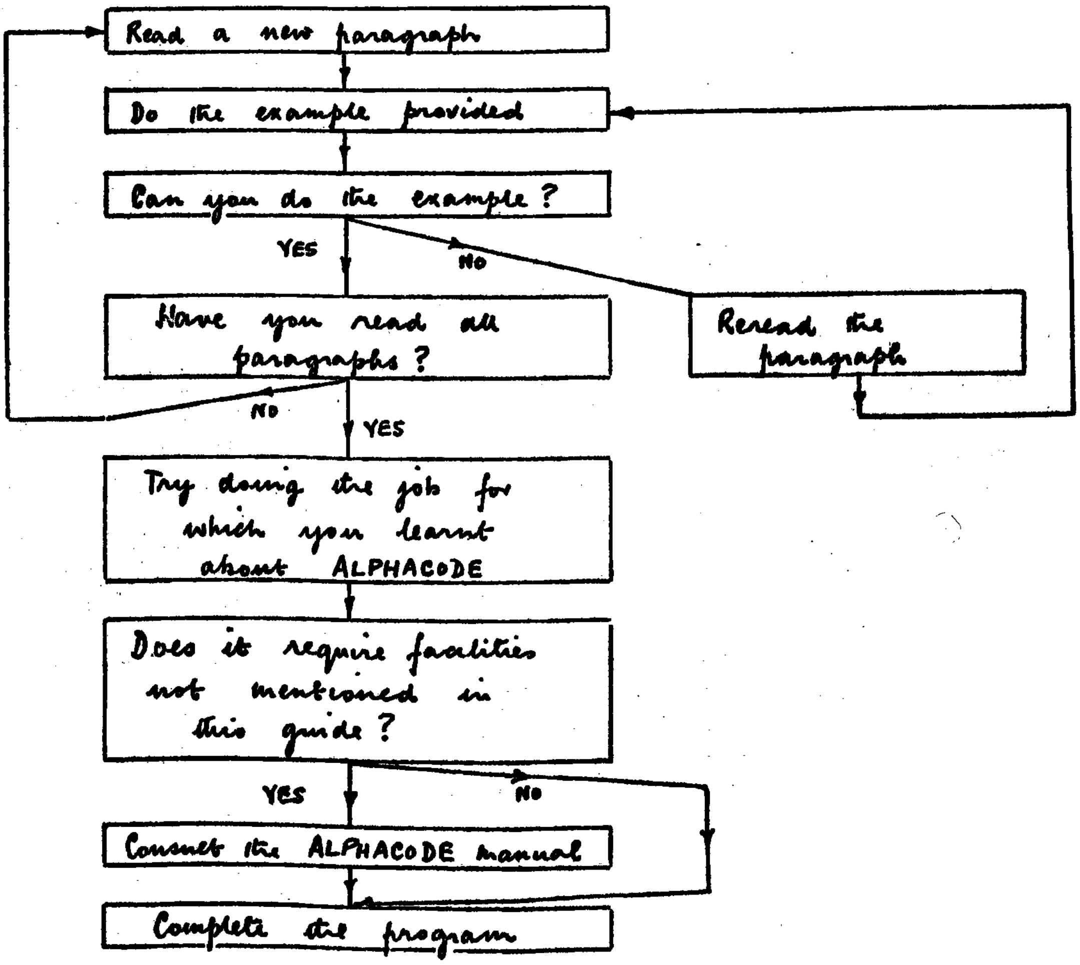
In order to overcome this difficulty the English Electric Company has produced an interpretive programme for their computer, DEUCE. By the use of this scheme called ALPHACODE the nonprofessional programmer can write a programme in simple language and let DEUCE interpret this language into its own order code.

The following few pages are to introduce laymen to alphacode and to enable them to write simple and unrefined programmes. Those wishing to use all the facilities of alphacode should after digesting this introduction refer to the complete alphacode manual.

2. APPRECIATION OF THE PROBLEM.

The first and most important requirement not only in alphacode but in all programming is to know exactly what the problem is which is to be solved. The method normally used to dissect the problem is to write a block diagram. It is important not to be lasy in writing this as logical problems not resolved in the block diagram are more difficult to solve when writing the alphacode instructions themselves.

We will illustrate this concept by writing the block diagram in which we set out the elementary steps required to read this paper.

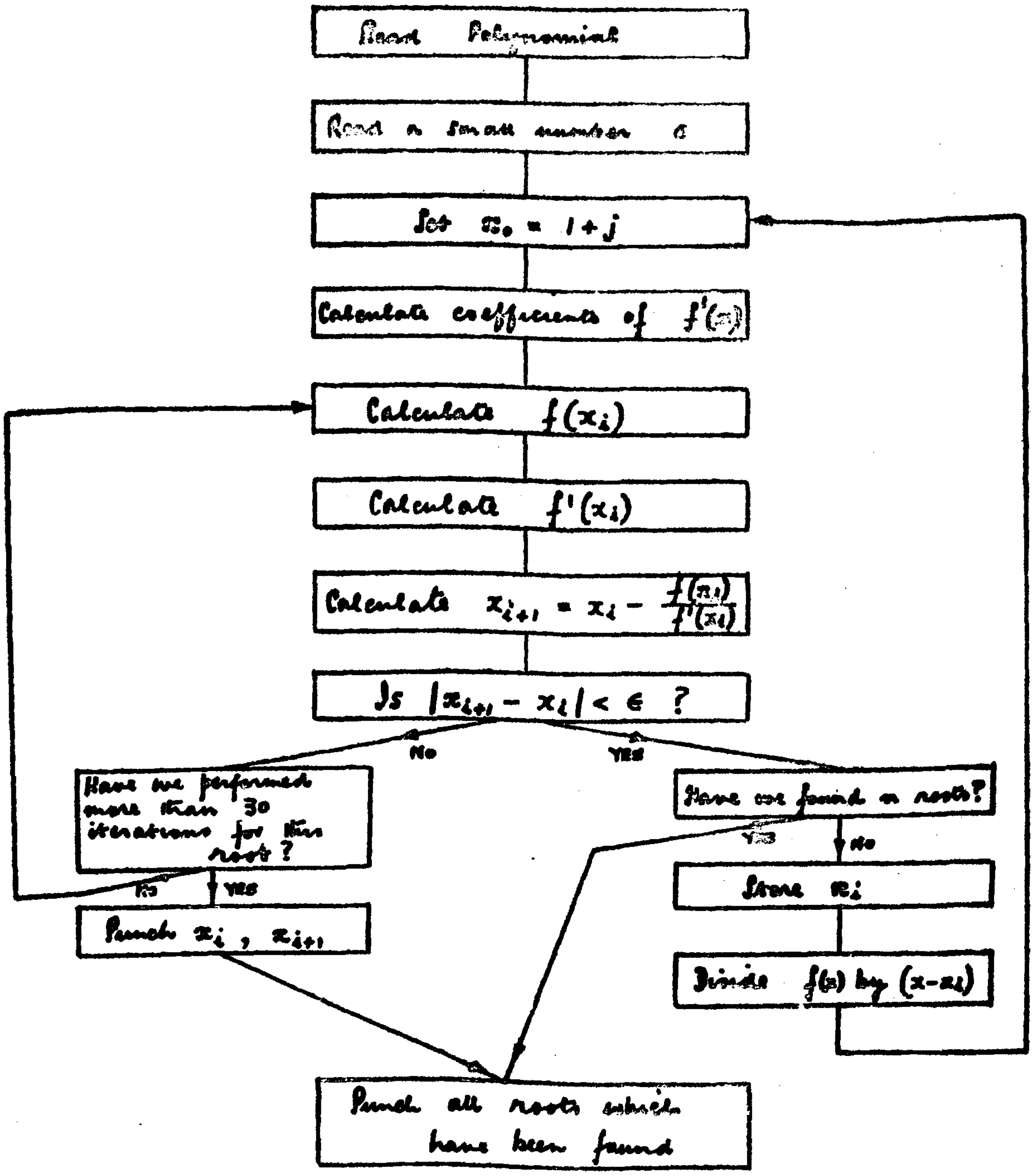


Read a new paragraph
Do the example provided
Can you do the example?
YES
NO
Have you read all paragraphs?
Reread the paragraph
NO
YES
Try doing the job for which you learnt about ALPHACODE
Does it require facilities not mentioned in this guide?
YES
NO
Consult the ALPHACODE manual
Complete the program

A second, slightly less facetious example is provided by the solution of polynomials by Newton's method. This method is as follows. If f(x) is a polynomial and x_i is an approximation to a root then a better approximation is given by

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

The block diagram will then be as follows:


Read Polynomial
Read a small number ε
Set $x_0 = 1+j$
Calculate coefficients of $f'(x)$
Calculate $f(x_i)$
Calculate $f'(x_i)$
Calculate $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$
Is $|x_{i+1} - x_i| < \epsilon$?
No
YES
Have we performed more than 30 iterations for this root?
No
YES
Punch x_i, x_{i+1}
Have we found a root?
YES
No
Store x_i
Divide $f(x)$ by $(x-x_i)$
Punch all roots which have been found

Having written the block diagram of the job we can now go on to write the necessary alphacode instructions.

Exercise 1.

Write the block diagram to solve a quadratic equation by the formulas

$$x_1 = \frac{-b + \sqrt{b^2 - ac}}{a}$$

$$x_2 = \frac{-b - \sqrt{b^2 - ac}}{a}$$

3. FORM OF INSTRUCTION.

Let us consider the evaluation of $3^6 = 3.3.3.3.3.3$. We can, if we let X1 = 3 write

X2 = X1 x X1
X3 = X2 x X1
X4 = X3 x X1
X5 = X4 x X1
X6 = X5 x X1

and this is precisely the type of formula used in alphacode. The symbol X1 however does more than signify a quantity, it is the address of a store. Thus the instruction

X3 = X2 x X1

should be read as "multiply the quantity in the store whose address is X1 by the quantity in the store whose address is X2 and store the result in the store whose address is X3, eliminating the previous contents of X3 and leaving the contents of X1 and X2 untouched".

The particular piece of jargon used to describe this type of instruction is to say that it is a three address code i.e. the form of instruction is

A = B function C

A glance at figure one will show that the coding paper for alphacode has ten columns. We are at present interested only in those headed A, B, FUNCTION, C although later we shall use those headed R and D.

In the above example the reader will note that X2, X3, X4, X5 contain intermediate results in which we are not interested. We could write

X2 = X1 x X1
X2 = X2 x X1
X2 = X2 x X1
X2 = X2 x X1
X2 = X2 x X1

ana remembering the interpretation of these instructions given above we see that if X1 contains 3 X2 would now contain 3^6.

4. ARITHMETIC.

The four basic instructions of alphacode are

Xa = Xb + Xc
Xa = Xb - Xc
Xa = Xb x Xc
Xa = Xb ÷ Xc

and the meaning of these is obvious. When an alphacode programme has been written it can be printed to give a clear copy and in this copy the operations defined above will appear as words thus

Xa = Xb MULTIPLIED by Xc

A specimen of such a tabulation is shown in figure two.

Exercise 2.

Given the three numbers a, b, c in X1, X2, X3 write the alphacode instructions necessary to calculate $b^2 + ac$ and $b^2 - ac$.

5. TRANSFER.

Although at this stage it is not obvious why, it is often desirable to transfer numbers from one store to another. To do this we write

Xa = Xb MOVED

which overwrites the previous contents of Xa with those of Xb leaving Xb unchanged.

6. INPUT AND OUTPUT.

There are two difficulties which have been glossed over in paragraph three. We said "if we let X1 = 3 ". How do we do this? Or, put another way, how do we get numbers into the machine? The second difficulty is, having obtained the result in X2, how do we get it out to look at it?

The answer to both of these questions is of course, that we have to use instructions and these are as follows:

Read 1 DATA into Xc

and Print 1 RESULT from Xc

We can, and indeed it is desirable to read or print more than one result at a time and so we can use an instruction like

Read 20 DATA into X21 onward

and this will read data into X21, X22,......,X40.

We seem already to have departed from our three address format. The vital parts of this instruction are however

20 DATA X21

the A position being left blank in this case. Whenever we write an instruction like this we shall always print the function name in capital letters, the words in small letters being embroidery to make the instruction sound correct.

Let us now consider a programme for calculating the areas of circles by the formula $A = \pi r^2$. There are two different sorts of data to be read in here. The first type is the radius of the circle which we may specify. π however is a constant which can never change and is as much an integral part of the programme as the instructions in it. For this type of data we have a special instruction

Xa is CONSTANT

3.14159

This constant is then built into the programme and does not need to be read in with each new set of data. The programme now reads as follows:

No.	r	R	A	B	FUNCTION	C	D	P	O S	NOTES
			X1		CONSTANT					Read in π to X1
					3.14159					
			Read	1	DATA	X2				Read radius r
			X2	X2	×	X2				Form r^2
			X2	X2	×	X1				Form πr^2 and
			Print	1	RESULT	X2				print the result.

The two constants 0 and 1 are already in the machine and can be used in the Xb and Xc position of any instruction. Thus if we required 1 in X1 we would simply write

X1 = 1 MOVED

Exercise 3.

Given a value of v write the alphacode instructions to calculate the quantity $\beta^2 = 1 - \frac{v^2}{c^2}$ where c is the velocity of light = 3.10^8 metres/second and print the result.

7. N STORES.

Before we proceed further we must mention two facilities viz. N stores and the jump facility without at present giving a reason for them.

In addition to the X stores, of which there are over 2,000 we have another type of store in alphacode, the 63 N stores N1 to N63. These stores contain integers only and are all initially zero (as indeed are the X stores). We can operate on them with + - x ÷ just as for X stores except that in the case of divide we get the integral part e.g. 5 - 3 would give 1 and not $1\frac{2}{3}$ or even 2.

We may transfer numbers between N stores by MOVED or we can transfer from N stores to X stores or from X stores to N stores although in the last case we shall get the nearest integer to the number in the X store transferred to the N store (not the integral part as in division).

Input to and output from N stores is effected by the same instruction as for X stores except that we may read or print only one N store with one instruction i.e. the instruction "Read 20 Data into N1" is not allowed. Further, a constant instead of being on a separate line must be in the C position of the constant instruction thus:

Na = CONSTANT 3

As with X stores the constants 0 and 1 are already in the machine.

A final word on N stores. You may not mix X stores and N stores in a single instruction (other than MOVED, DATA and RESULTS).

8. REFERENCES AND JUMP.

If we wish to refer to an instruction from another part of the programme we may write a reference number (say 6) in the R column of the coding sheet. If now we wish to go from one part of a programme to another we write "Jump to Rd". The programme now looks like this

No.	r	R	A	B	FUNCTION	C	D	P OS	NOTES
6		6							
11					JUMP TO		R6		

Let us now see how these facilities are used.

9. COUNTING.

Consider again the example in paragraph three.We see that the same instruction is repeated five times. This is going to become very clumsy if we wish to calculate x^{100}. To overcome this difficulty we have the following instruction

Count Nb UP TO 100 jumping to Rd

where we can write any number we please in place of 100.

Written in full this instruction would read "Add 1 to the number in Nb. If this number is not equal to 100 jump to the instruction whose reference number is d. If it is 100 proceed to the next instruction". Its effect is to cause the loop of instructions which it defines to be obeyed 100 times and then to carry on with the programme.

Our programme to calculate x^{100} would now look like this:

No.	r	R	A	B	FUNCTION	C	D	P OS	NOTES
				1	DATA	X1			Read x into X1
			X2	1	MOVED				Set X2 at $x^0 = 1$
		1	X2	X2	X	X1			Calculate x^n
			COUNT	N1	UP TO	100	R1		and when $n = 100$
			PRINT	1	RESULT	X2			print the result

You may ask why do we need N stores at all? Why do we not count in X stores? In answer to this, suppose we wished for some obscure reason to count on a desk machine by ⅔ up to 2. This seems to require three additions. However, in decimal ⅔ is represented by 0.6667 and so our three additions would give 0.6667, 1.3334 and 2.0001 and we would never get exactly 2. We thus see that we require stores in which we can count exactly without any possibility of rounding errors. As a corollary we see that it is unwise to expect exact equality of numbers in X stores. We shall refer to this again later.

Exercise 4.

Write a set of instructions to calculate $P(1 + \frac{x}{100})^{25}$ and print the result.

10. DISCRIMINATIONS.

The count instruction just described is a form of discriminatory function. Alphacode contains several more of these functions all of which say "if some condition is satisfied jump to somewhere else in the programme. If it is not satisfied proceed normally". These functions are

If Xb = (EQUALS) Xc jump to Rd

If Xb ≠ (UNEQUAL) Xc jump to Rd

If Xb ⩾ (AS BIG AS) Xc jump to Rd

If Xb > (BIGGER THAN) Xc jump to Rd

All these instructions may be used with N stores e.g.

If Nb EQUALS Nc jump to Rd

Thus in our example if we wished to multiply x by itself until it was larger than 10,000 ($1 < x < 10,000$) and print the power of x for which this was first fulfilled we could write

No.	r	R	A	B	FUNCTION	C	D	P	O S	NOTES
			X3		CONSTANT					Set X3 = 10,000
					10,000					once and for all
			Read	1	DATA	X1				Read x into X1
			X2	X1	MOVED					and X2
		2	N1	N1	+	1				
			If	X2	>	X3	R1			Form x^n until
			X2	X2	×	X1				$x^n > 10,000$
					JUMP TO		R2			
		1	Print	1	RESULT	N1				and print n

Exercise 5.

Write a set of instructions which divide the number in X2 by that in X1 where if the number in X1 is zero it is to be replaced by a very small number. (10^{-100} may be considered small enough and must be read using the appropriate instruction).

11. INSTRUCTION MODIFICATION.

There is still one shortcoming left in the programme just outlined. It works for only one value of x. Suppose we had twenty values of x in X21 - 40. Then in order to repeat the calculation we would have to rewrite the instructions for each separate case. In order to avoid the necessity of doing this we have a facility in alphacode for modifying instructions.

The instructions

N1 MODIFY next instruction

X200 = X100 PLUS X1

would cause the contents of N1 to be added to the address in the A position of the next instruction. Thus if N1 contains 2 the two instructions above would be equivalent to X202 = X100 + X1. We can vary any of the addresses or any two or all three with any N store by putting the appropriate N store in the appropriate position of the modifying instruction. Thus if N1 contains 2 and N2 contains 6 the instructions

N1 N1 MODIFY next inst N2 also

X200 X100 PLUS X1

would be equivalent to X202 = X102 + X7.

Thus if we wish to multiply each of the numbers in X21 - X40 by that in X1 and store the products in X41 - X60 we would write

```
1        N1        N1        MODIFY next instruction
         X41       X21       MULTIPLIED by        X1
              Count N1       UP TO                20  jumping to      R1
```

remembering that UP TO adds 1 to the contents of N1 each time it is obeyed.

Notice that a MODIFY instruction refers to one and only one instruction and that if we wish to modify two successive instructions we must use two MODIFY instructions. We cannot of course modify an instruction with the contents of an X store nor can we modify N store addresses e.g. we cannot modify the instruction N1 = N3 + N4. Negative values of a modifier are not permitted.

The example quoted above would now look like this:

No.	r	R	A	B	FUNCTION	C	D	P	O S	NOTES
			X3		CONSTANT					Set X3 = 10,000
					10,000					once and for all
			Read	20	DATA	X21				Read 20 values of x
		3		N1	MODIFY					Fetch the next
			X1	X21	MOVED					value of x
			N2	0	MOVED					Set N2 = 0
			X2	X1	MOVED					Form x^n until
		2	N2	N2	+	1				x^n > 10,000
			If	X2	>	X3	R1			recording n in
			X2	X2	×	X1				N2
					JUMP TO		R2			
		1	N1		MODIFY					Store this value
			X41	N2	MOVED					of n
			COUNT	N1	UP TO	20	R3			Perform the calculation
			Print	20	RESULTS	X41				for all values of x
										and print the result.

Exercises

6. Given a set of radiuses in X41 - X70 write a programme which calculates the area of the circles with these radiuses writing the results in X71 - X100.

7. Given ten values of P and twenty values of x calculate $P(1 + \frac{x}{100})^{25}$ for all combinations of P and x and print the results.

8. Given twenty values of a, b, c calculate b^2 - ac for each set and print the results.

12. COMPREHENSIVE FUNCTIONS.

We do not propose here to mention many further facilities in alphacode except to point out that other facilities do exist of which we give three examples here.

If Xc contains y then

```
Xa  =        ROOT        Xc
```

puts + $\sqrt{y}$ in Xa,

```
Xa  =        EXP         Xc
```

puts e^y in Xa; and

```
Xa  =        LOG         Xc
```

puts $\log_e y$ in Xa.

None of these instructions can be used with N stores.

13. PROGRAMME TESTING.

One of the facts of life one learns when programming is just how silly human beings are compared with machines - machines do not make errors, human beings do; and nowhere is this more marked than in writing programmes. For this reason there are some alphacode facilities for programme testing, the most important of which is the P facility.

If we write a P against any of the instructions in alphacode a signal from the programme operator will cause the result of each of these instructions to be printed as it is calculated. Thus, with a liberal sprinkling of P's throughout the programme the course of a calculation can be followed and errors easily detected. Since the printing with this facility requires a signal from the operator, the P's may be left in when the programme is working and have no effect on the programme.

The numbers printed with a P will appear in a form called standard floating decimal i.e. in the form $a.10^b$ where $1 \leqslant a < 10$ and $-999 \leqslant b \leqslant 999$, b being an integer. For example the number 1234.56789 will be printed as

```
1.23456789      3
```

14. FINISH.

The last instruction of any alphacode programme must always be

```
FINISH
```

This instruction tells the machine that it has finished reading instructions and must now start obeying the first of them. Notice that the machine does not obey the instructions as it reads them. Notice also that after reading the instructions it will obey the first one and then carry on obeying them relentlessly one after the other until it reaches FINISH. It may be as well to look at your programmes again from the machine's point of view and make sure you have written the instructions in the order in which they are to be obeyed.

Exercise 9.

This exercise enables you to use most of the functions mentioned in this guide.

Given a set of numbers a b and c write an alphacode programme to calculate

$$\frac{-b + \sqrt{b^2 - ac}}{a} \quad \text{and} \quad \frac{-b - \sqrt{b^2 - ac}}{a}$$

punching the two results if $b^2 - ac$ is positive; $-\frac{b}{a}$ and 0 if $b^2 - ac$ is zero and two zeros if $b^2 - ac$ is negative.

Now extend this programme to deal with twenty sets of numbers a, b, c using only one read and one punch instruction.

No.	r	R	A	B	FUNCTION	C	D	P OS	NOTES
			X_A	X_B	MOVED				
			X_A	N_B	MOVED				TRANSFER
			N_A	N_B	MOVED				
			N_A	X_B	MOVED (nearest integer)				
			X_A	X_B	+	X_C			
			N_A	N_B	+	N_C			ADD
			X_A	X_B	−	X_C			
			N_A	N_B	−	N_C			SUBTRACT
			X_A	X_B	×	X_C			
			N_A	N_B	×	N_C			MULTIPLY.
			X_A	X_B	÷	X_C			
			N_A	N_B	÷	N_C	(integral part)		DIVIDE.
					JUMP TO		R_D		UNCONDITIONAL JUMP
				X_B	EQUALS	X_C	R_D		JUMP TO R_D if $X_B = X_C$
				N_B	EQUALS	N_C	R_D		JUMP TO R_D if $N_B = N_C$
									otherwise take next
									instruction
				X_B	UNEQUAL	X_C	R_D		JUMP TO R_D if $X_B \neq X_C$
				N_B	UNEQUAL	N_C	R_D		JUMP TO R_D if $N_B \neq N_C$
									otherwise take next
									instruction
				X_B	AS BIG AS	X_C	R_D		JUMP to R_D if $X_B \geq X_C$
				N_B	AS BIG AS	N_C	R_D		JUMP to R_D if $N_B \geq N_C$
									otherwise take next
									instruction
				X_B	BIGGER (than)	X_C	R_D		JUMP to R_D if $X_B > X_C$
				N_B	BIGGER (than)	N_C	R_D		JUMP to R_D if $N_B > N_C$
									otherwise take next
									instruction

Figure 1: Alphacode functions

No.	r	R	A	B	FUNCTION	C	D	P	O S	NOTES
				N_B	UP TO	10	R_D			COUNT N_B UP TO 10
				N_B	UP TO	N_C	R_D.			(or to the number in N_C) JUMP TO R_D when the COUNT IS NOT COMPLETE

This instruction adds one to N_B each time it is obeyed. It tests if N_B is equal to 10 (or the number in N_C). If not the next instruction taken is R_D. When the count is complete the next instruction in sequence is obeyed and N_B is cleared to zero

No.	r	R	A	B	FUNCTION	C	D	P	O S	NOTES
			X_A		CONSTANT 3.14159					Reads the number 3.14159 from a card into X_A.
			N_A		CONSTANT	12				Reads the number in Column C into N_A
				1	DATA	X_C				Reads one number from a card into X_C
				1	DATA	N_C				Reads one number from a card into N_C
				10	DATA	X_C				Reads 10 numbers from 10 cards into successive Stores starting at X_C
				1	RESULT	X_C				Punch 1 Result from X_C
				1	RESULT	N_C				Punch 1 Result from N_C
				6	RESULTS	X_C				Punch 6 Results starting at X_C
			X_A		ROOT	X_C				Puts $\sqrt{y}$ in X_A if y is in X_C
			X_A		EXP	X_C				Puts e^y in X_A if y is in X_C
			X_A		LOG	X_C				Puts $\log_e y$ in X_A if y is in X_C

Figure 1 (continued): Alphacode functions.

No.	r	R	A	B	FUNCTION	C	D	P O S	NOTES
			N_A	N_B	MODIFY	N_C			

If n_a n_b, n_c are the contents of N_A N_B and N_C
this instruction adds n_a n_b n_c to the A B and C
addresses of the next instruction

Figure 1 (concluded): Alphacode functions.

r	R	A	B	FUNCTION	C	D	P	O/S
0		X0001 =	X0002	MOVED				
0	2	N0001 =	N0002	MOVED				
0		X0001 =	N0002	MOVED				
0		N0001 =	X0002	MOVED nearest		integer.		0
1	1	X0001 =	X0002	PLUS	X0003			
1		N0001 =	N0002	PLUS	N0003		P	
2		X0001 =	X0002	MINUS	X0003			
2		N0001 =	N0002	MINUS	N0003			S
3		X0001 =	X0002	MULTIPlied	by X0003			
3		N0001 =	N0002	MULTIPlied	by N0003			
4	3	X0001 =	X0002	DIVIDEd	by X0003		P	O
4		N0001 =	N0002	DIVIDEd	by N0003	,integral part		
5				JUMP		to R 4		
6		If	X0002	EQUALS	X0003	jump to R 4		
6		If	N0002	EQUALS	N0003	jump to R 4		
7		If	X0002 is	UNEQUAl	to X0003	jump to R 4		
7	4	If	N0002 is	UNEQUAl	to N0003	jump to R 4		
8		If	X0002 is	AS BIG	as X0003	jump to R 4		
8		If	N0002 is	AS BIG	as N0003	jump to R 4		
9		If	X0002 is	BIGGER	than X0003	jump to R 4		
9		If	N0002 is	BIGGER	than N0003	jump to R 4		
10		Count	N0002	UP TO	0003 ,	jumping to R 4		
11		X0001 =		CONSTAnt		+3.14159265 +000		
11		N0001 =		CONSTAnt	0003			
12		N0001 and	N0002	MODIFY next inst.	N0003	also		
18				FINISH				
23		Read	0002	DATA	into X0003	onward.		
23		Read	0001	DATA	into N0003			
24		Batch 0001 Print	0002	RESULTs	from X0003	onward Type 0		
24		Batch 0001 Print	0001	RESULT	from N0003			
25		X0001 =		ROOT	X0003			
28		X0001 =		LOG	X0003			
29		X0001 =		EXP	X0003			

Figure 2; Tabulated alphacode functions.

LECTURE C23.

DATA PROCESSING I.

AIM: Summarise the steps in getting a business application mechanised; assess the importance of each step; and examine the calibre of person concerned in these steps.

C23.1 INVESTIGATION.

Having been presented with either a department and the general directive to mechanise it, or a carefully chosen aspect of an established system to put onto data processing machinery, the person so directed must stand back and view first the whole situation.

This is necessary for the two reasons:-

(a) it is most advantageous to get acquainted at the earliest stage with the terms in common use in the department under investigation.

(b) no part of a system is complete in itself, so that even if it is the initial intention to look at only, say, the stock control aspect of a production system, it is necessary to see how all such aspects fit together.

Therefore the first step is a thorough investigation of as many aspects of the processes as are likely to have any bearing on the final mechanised system. This entails finding out not only what goes on, but why it happens, and if necessary asking the same questions of different people to verify what actually does happen.

This investigation should be backed by management in as high a position as possible since this is the only way of gaining both confidence and co-operation.

The result of all this should be a comprehensive survey of the salient points of the system, unencumbered by operational details; arrangements should be made to ensure that any procedural changes during or after this initial survey are notified to the investigators.

C23.2 FORMULATION.

Once the investigation has been completed, there should now be sufficient information to decide whether the aspect that was originally chosen for mechanisation is suitable because of its relation to the rest of the business, or, if no decision has been made, to decide what aspect to tackle.

This means that a particular problem can be formulated. The problem may be to mechanise the monthly payroll, or all the payroll, or to examine the machine loading in a particular shop, or to keep premium payment records in the system, or to consider all the production aspects of a manufacturing organisation and to integrate all these in one go. In any case the problem to be tackled is defined.

The preliminary investigation will have provided enough facts, providing it has been adequately carried out, to make the problem well defined, both itself, and in its relation to allied problems.

C23.3 ANALYSIS.

So far there need not have been any reference to a particular type of data processing machine, but the time has now arrived. The problem formulated must be knocked into a set of logical block diagrams, showing both the information flow and machine operations.

At this stage deficiencies in the specification of the problem will inevitably be thrown up, requiring re-investigation of the system, and a variety of suitable approaches to answer the problem will appear. This means that a choice of computing machine may have to be considered, with regard to their available facilities-paper tape or punched card input, simultaneous magnetic tape reading and writing or not. It may mean deciding the relative merits of available facilities on a given machine - whether to keep a file on punched cards or on magnetic tape for example.

In any case this analysis will produce one or more suitable schemes which will be logically sound, practicable on the machinery available, and suitably costed. These schemes must be able to be sold to whoever is sponsoring the work, so that the costing must be carefully done. It is not at all easy to assess accurately the financial side of hidden gains such as savings in stock holdings, nor is it easy to put a price to shortening delivery periods, so that the places where one hopes to get the most advantages are the least easy to specify in money terms.

It is not easy to make a very close estimate of machine running time without a lot of experience of similar jobs or a great deal of pilot programming, but by and large it is possible to make a reasonable assessment of the financial merits of any of the suggested schemes.

C23.4 RECAPITULATION.

The result of going through the three stages investigation, formulation and analysis should be a set of logical block diagrams which have been agreed to by all concerned. However if the process has not provcd satisfactory, it may be necessary to keep going through the three steps again and again until a satisfactory system is specified; then and only then is it wise to start programming.

Indeed these investigations may have shown that there is no merit in mechanising at all, but that a clerical re-adjustment will be enough.

C23.5 PROGRAMMING.

A business application requires a considerable proportion of the effort spent on it to be other than programming. It is necessary to produce both a clerical procedure manual and a programme operation manual, the former capable of being used by people with no computer knowledge.

At this stage, then, when the system has been defined and agreed, the production of these manuals can proceed in parallel. It is usually the clerical manual which is the more straightforward to produce, since it is less liable to change during the development of the system than the programme operation manual which is tied very much to the final programme specification.

It must not be forgotten, however, that the closer the liaison between the paper work preparation and the programme inputs and outputs the easier the programming and tabulation becomes.

C23.6 TESTING.

From this stage on, it makes little difference that the programme under test is a business one, though it is true that business programmes are usually easier to write, and harder to get consistent test data for. Mistakes found are usually logical ones of the system, and thus more in the heart stopping class, so that the testing process can be quite time consuming.

C23.7 TRIAL RUNNING.

There comes a time when the programmes comprising the system have been satisfactorily tested, and a process of trial running of the system or parts of it must be inaugurated. This is the stage at which the clerical and computer procedures are tied together, and at which both the clerical and machine operational staffs become familiarised with their jobs.

The various operational snags will be shown up, the stock rules in a stock system can be shown to be valid, the things that have been forgotten can be incorporated, and a planned turnover begun.

C23.8 ROUTINE OPERATION AND MAINTENANCE.

No system runs for very long without there being a proposed modification to it, either as an improvement or to cater for a changed exigency in manufacturing policy. The amount of effort required to keep a mechanised system well maintained must not be underestimated; this means that it is imperative to have detailed and clear accounts of the clerical procedures, the programme and peripheral operations, and the programmes themselves. Without these it is virtually impossible to make a successful change; good write-ups mean good running.

C23.9 TIME SCALE.

It is of no value to try to specify possible times for particular types of application, since experience is not yet wide enough, but it is worthwhile to emphasize where the bulk of the work lies.

There have been eight stages specified in getting a mechanised application going, and of these it is the first three and their recapitulation which are the most vital, and usually the most time consuming. It cannot be over-emphasised that it is unwise to start writing the programme until a suitable system has been specified properly, for the capital cost of programme writing and testing is high and not worthy of waste.

Once, however, a system has been truly specified, programming should get under way at full speed so that changes in the manufacturing system during the turnover period have as little effect as possible on the programmes to be written. Programmes should however be as flexible as reason will allow.

Trial running in parallel can take longer than one first imagines. Not only do mistakes show up for the first time at this late stage, when they are the most time consuming to rectify, but it takes quite some time for users to get accustomed to the system. There are penalties, however, in running in parallel for too long; until the time when it is decided to stop parallel running and depend only the mechanised system, the mechanised system's records must continually be brought into line with those of the running system. This means, for example, that a stock balance file must be brought up to date, or a payroll record put right weekly, and these are the more difficult if the rules followed on the mechanised system are more sophisticated than those under the ordinary system.

Remember, however, that the more effort that goes into the three early stages, the easier and quicker the later stages become.

C23.10 STAFF CALIBRE.

It will have been apparent that not all the eight stages of mechanisation require the same type of person, and there are broadly three categories needed.

The early stages need computer minded systems men, who do not necessarily have to be programmers, but by the analysis stage programmers are definitely needed. Then a combination of machine operators and programmers to run the trials are required, whilst the maintenance and extension of the system require a combination of systems men, operators and programmers.

C23.11 EMPHASIS.

It must finally be re-emphasised that the earlier stages are the most important, and that no effort should be spared on those aspects.

24

LECTURE 24.

MAGNETIC TAPE STORAGE SYSTEM.

N.B. These lecture notes are not meant as a programming manual, but one intended to give some insight into the Deuce Magnetic Tape Storage System.

24.1 THE TAPE TRANSPORTER.

24.1.1 The tape handling equipment used with the Deuce Magnetic Tape Storage System consists of Decca Twin Tape Transporters. The Twin Tape Transporter consists of two tape decks and and their associated control and drive equipment, on which up to 2,400 feet of $\frac{1}{2}$" wide magnetic tape may be moved past the read/write heads under computer control. The tape may be run in either direction at an operational speed of 104" per second, but the tape can only be written on or read from when it is going in the forward direction, e.g. from the right hand spool to the left hand spool. Up to 4 twin tape transporters may be connected to Deuce.

24.1.2 About 15 feet from either end of the tape the magnetic oxide is removed from the surface for about $1\frac{1}{2}$ inches leaving a transparent window. The window nearest the left hand end of the tape is called the beginning of tape mark and the other the end of tape mark. Only the tape between these two marks is moved under the read/write heads by computer control. If the beginning of tape mark is sensed by the computer, backward movement of the tape is inhibited; similarly, when the end of tape mark is sensed the tape can no longer be moved forward under computer control.

24.1.3 While the tape is on the tape deck and the tape deck is ready to obey any order given by the computer, the tape deck is said to be in the remote phase; otherwise it is said to be inoperable. If a tape deck is selected for use and it is inoperable a signal is sent to the computer.

24.1.4 In order to provide a warning that an end of tape mark is approaching the read/write heads and that forward movement of the tape is about to be inhibited, an early end of tape warning system is provided. If the last start order received was for tape movement in the forward direction and the end of tape mark is seen by the end of tape warning photo-cell, a signal will be sent to the computer. This signal, which can be detected by a programmer, will continue to be sent until the end of tape mark is sensed by the end of tape warning photocell when the tape is moving backwards, or when the tape deck is taken out of the remote phase.

24.1.5 If it is required that the beginning of a usable section of the tape is under the read/write head, a rewind signal is sent to the tape deck. Tape will then be spooled onto the right hand reel at high speed (approx. twice normal tape speed), until the beginning of tape mark is under the beginning of tape photocell. During initial loading of a deck, the full spool of tape is normally loaded onto the right hand reel spindle and an empty tape reel is put on the left hand spindle. Tape is then fed under the read/write heads and erase head and is fastened onto the left hand reel. On closing the doors of the deck, the deck automatically winds the tape forward until the beginning of tape mark is sensed by the beginning of tape photo-cell.

24.2 THE READ WRITE HEADS.

24.2.1 The writing and reading to and from the tape is carried out using Decca Seven Channel read/write heads. Six of the channels are used for information, the seventh for a lateral parity check. The six information digits simultaneously handled by the read/write heads, one digit per information channel of tape, are known as a character. The lateral parity digit is written such that there are an odd number of ones across the tape for each character.

24.3 WORD PAIRS.

24.3.1 When writing, information is sent to D.S.20 (by the programme), and it is written onto the tape via a buffer store. Programmers have no access to this buffer store, and so they do not need to bother about it. Once information is sent to D.S.20 the rest of the writing process is automatic, therefore the smallest unit of information that may be written is a word pair. If the writing of one word pair has been completed and new information has not been sent to D.S.20, within a certain time, writing will stop.

24.3.2 When reading, information is assembled in the buffer store and then put into D.S.20 when a word pair has been read. The information must be extracted from S.20 before another word pair has been assembled in order that information is not lost.

24.3.3 D.S.20 has storage for 64 binary digits of information, which is not an integral number of 6-bit characters. Therefore alternative instructions fcr reading or writing either 10 characters or 11 characters per word pair are provided.

On transferring (i.e. reading or writing) 10 characters per word pair, sixty binary digits of D.S.20 are transferred, digits P_{29} - P_{32} of 20_3 are ignored. On transferring 11 characters per word pair the full 64 digits plus two extra digits are transferred. The two extra digits are always written as zeros and are ignored on reading back. Effectively, to the programmer, 11 character transfers transfer only the 64 binary digits of D.S.20.

24.3.4 The ten characters per word pair transfer system is necessary if a tape printer is to be used, and it also gives compatibility with the 80 column card reader. In addition to these, up to 10% economy in tape may be obtained if character working is being carried out.

24.4 DEFINITION OF TERMS.

24.4.1 Character - this consists of six binary digits of information which may represent 0-9. A-Z and many other symbols.

24.4.2 Block. The characters on tape are recorded in blocks. A block is a train of characters recorded without stopping the tape. The block length is variable in integral numbers of word pairs.

24.4.3 Block Gaps. Since reading and writing may only be carried out while the tape is travelling at full speed (i.e. 104"/sec) the tape takes a definite time to start or stop, therefore a piece of blank tape is always produced while the tape is attaining full speed, and another while the tape is stopping. Thus between two blocks of information there is a gap. This gap is known as a block gap and is always less than two inches in length.

24.4.4 Record. A group of successive blocks in the tape is called a record.

24.4.5 Record Gap. This is a programmed space on the tape written before and after a record.

24.4.6 The use of a record is that it makes it possible to overwrite information in the middle of a tape without affecting information stored before or aftèr the record being overwritten.

24.5 RE/DING AND SKIPPING.

24.5.1 It has been explained that between blocks of information there are either record or block gaps, and the amount of information written in a block is programmed. Reading however will continue until a block has been read and the following block gap found, after which the tape will stop. Thus when reading the tape will always stop with the read/write head in a gap. This also applies when skipping, the skipping facilities provided being -

Skip forward one block (SF)
Skip backward one block (SB)
Skip forward to the next recrod gap (SFR)
Skip back to the previous record gap (SBR)

These skipping facilities are provided so that the tape may be moved until the gap preceding the next block of information to be read is beneath the read/write head before the transfer is called.

24.6 INTERLOCKS.

24.6.1 If a magnetic tape instruction is put into the Deuce Control store and some part of the Deuce Magnetic Tape storage equipment is not available for use, such that the instruction cannot be obeyed, the instruction is interlocked until the equipment is again ready. The particular cases where instructions will be interlocked are given below.

24.6.2 If a tape deck is selected and the tape deck is connected but not in the remote condition, no further tape instruction may be obeyed until the tape deck is ready for use.

24.6.3 While writing, any tape instructions will be interlocked until writing is complete and the tape stopped.

24.6.4 While reading or skipping, tape instructions will be held until the skipping is complete, but the interlock is released before the tape has actually stopped.

24.6.5 While a record gap is being written on tape no interlocks are provided since any tape instruction will clear write record gap. However only one particular instruction should be used to clear write record gap; this is because other instructions may or may not be obeyed. After it has been cleared no further tape instructions can be obeyed until the tape has stopped.

24.6.6 When a rewind is called, it interlocks any other tape instruction for 250 milliseconds; at the end of this 250 milliseconds the tape deck is deselected and so any other tape instruction may be obeyed. In fact it is possible to have up to 8 tape decks rewinding simultaneously.

24.7 GROUP SEQUENCE TRANSFERS.

24.7.1 As was described in section 24.3 information is transferred to and from the tape via D.S.20., but this system leaves very little time for carrying out any other work while doing tape transfers. In order to make more time available for parallel working when carrying out tape transfers, a facility is available whereby information can be automatically transferred between D.S.20 and D.L.9 while writing or reading is in progress. This enables up to 16 word pairs to be transferred between tape and D.L.9 without any intermediate reference to D.S.20 or D.L.9. These word pairs that are transferred between D.S.20 and D.L.9 without intermediate reference to them are said to comprise a group. Several groups may be transferred within a block; this transfer is known as a Group Sequence transfer.

24.8 PARITY CHECKING.

24.8.1 Parity Checking is optional with the Deuce magnetic tape storage system, it being necessary to obey a check parity instruction before a parity failure can be detected. Only the parity of a block that has been read can be checked, since while skipping the parity check circuits are not used. The check parity instruction is of the discrimination type, i.e. if a failure has been found when the parity check instruction is obeyed then the timing number of that instruction will appear to have been increased by one; if however no failure has occurred the normal next instruction will be taken.

24.8.2 While reading, each character is checked to ensure that together with the parity bit there are an odd number of ones written across the tape. Also each track of the tape is checked to ascertain that it contains an even number of ones in the block. (There is a parity character written at the end of every block which should make the number of ones in every track even). The lateral parity of the longitudinal parity check character is not included in the parity checking since it may or may not contain an odd number of ones.

24.9 THE DEUCE ORDER CODE FOR THE MAGNETIC TAPE STORAGE SYSTEM.

24.9.1 The instructions containing destination 24 and which have source numbers greater than or equal to 16 are used for magnetic tape. The instructions are

16 - 24	Select tape deck 0	
17 - 24	Select tape deck 1	
18 - 24	Select tape deck 2	
19 - 24	Select tape deck 3	
20 - 24	Select tape deck 4	
21 - 24	Select tape deck 5	
22 - 24	Select tape deck 6	
23 - 24	Select tape deck 7	
24 - 24s	read eleven characters/word pair	R1 *
24 - 24l	write eleven characters/word pair	W1
(P1)24 - 24s	Group sequence read eleven characters/word pair	R1
(P1)24 - 24l	Group sequence write eleven characters/word pair	W1
25 - 24s	read ten characters/word pair	R2
25 - 24l	write ten characters/word pair	W2
(P1)25 - 24s	Group sequence read ten characters/word pair	R2
(P1)25 - 24l	Group sequence write ten characters/word pair	W2
26 - 24	Write record gap	WG
27 - 24	rewind	Rewd.
28 - 24s	skip forward one block	SF
28 - 24l	skip forward to next record gap	SFR
29 - 24s	skip back one block	SB
29 - 24l	skip backward to previous record gap	SBR
30 - 24	Check Parity	
31 - 24	Clear write record gap and Inactive.	

* This column of letters are the letters engraved under various lamps on the Deuce console; when a particular lamp is ON that particular function is being carried out.

25

LECTURE C25.

PRINCIPLES OF COMMERCIAL PROGRAMMING.

C25.1 INTRODUCTION.

I propose in this lecture to approach the work of the programmer from the point of view of a systems consultant, observing what demands he makes of the computer installation, and what demands this implies for the individual programmer, leading to an examination of a particular instance.

C25.2 SYSTEM OBJECTIVES.

The systems consultant is concerned with discovering the most effective use of the time, labour and capital available. To him the computer is a particular device by which in certain circumstances some large-scale functions can be discharged more economically. The systems consultant may have to examine the structure of operational sequences, the allocation of functions, the chains of responsibility, work-flow, office layout, form design, machine utilization and so forth. From his study of these he is to derive suggestions for the more economic provision of services or products at the most effective time and place. And clearly it is only as instruments of such a purpose that data processors can justify themselves to their employers. The term 'data processors' is preferred here in order to draw attention to the facts that nearly all office and factory work is essentially processing of data, and that the equipment with which we are concerned is only useful in so far as we can treat the task to be performed as an orderly and finite sequence of elementary choices and simple data manipulations.

Each of you will be able to name some ways in which use of a data processor can provide more economic management.

(1) Consolidation of functions. In the past administrative considerations may have required separation of functions, not merely among different persons but also among several departments.

(2) Rationalization of procedures. The more comprehensive approach compelled by preparation for a data processing system should lead to a simplification of the functions pattern.

(3) Speeding of control functions, enabling

(i) capital economy through lower stock levels.
(ii) time economy through quicker integration of order into production plans.
(iii) labour economy through more sensitive production control.

(4) Increased reliability of control functions. Mechanised control decisions may be able to take account of more comprehensive data in selecting appropriate action.

(5) More effective presentation of the data for management decisions. On the one hand more searching analyses of received data will be possible, and on the other hand only a more critical selection of special cases will need to be referred to management. Prediction of trends, costs, delivery dates will be more reliable.

(6) Earlier output or cheaper output - where the whole service or operation is transferred to the data processor. Here the equipment offers a more efficient use of the time and capital available for the desired end.

Each of these corresponds to a fruitful field for the work of systems consultants.

C25.3 SYSTEM ANALYSIS.

Just as the objectives to be realized by data processors and systems consultants are the same, so we may learn something from their methods of enquiry. It is important that we should have a good understanding of the processes by which the job has reached the form in which it comes to the programmer. We should in all cases be prepared to verify that these previous processes have been carried out correctly.

The following catalogue, first designed to be borne in mind when an existing system is being appraised, is also of direct value to a programmer examining the specification of a new job.

(1) Make sure you understand the purpose and the proposed scope of the programme. But be prepared to initiate variations in the scope and objectives. Quite probably some aspects, or some exceptions, are unnecessarily being held back for special treatment.

(2) Attempt to set down a priori the information you would need for programming the job. Extend this list wherever information given seems to imply further requirements. Compare this list in detail with the information provided.

(3) Check information for
(i) accuracy and clarity,
(ii) completeness, i.e. that there are no exceptions,
(iii) validity, i.e. the exact sphere of application of a given piece of information.

Double-check all aspects which do not appear reasonable. Where in any doubt note both the doubt and the possible sources of fuller information.

(4) Be sceptical. The information may, for example, be based on an ambiguous question. The enquiry 'How is this job done' has at least five distinct legitimate answers:-
(i) The way the book says it should be done.
(ii) The way the supervisor thinks it is done.
(iii) The way the supervisor tells the employee to do it.
(iv) The way the employee thinks the supervisor wants it done.
(v) The way it is actually done.

(5) Make sure you know about all the unusual, rare duties which may arise within or on the borders of the programme's scope.

(6) Attempt to have available copies of all the forms employed and specimens of all reports, with analyses of their actual usefulness. Check that the output is required, is used, and is used in the form specified.

(7) Record recent and possible future variations in the system.

The aim here is an understanding of (a) the purpose of the present system and (b) requirements imposed on the programme. The basic needs at this stage are a clear appreciation of the assumptions underlying the present system and an insight into the possible and impossible alternatives at every point.

Any improvements which occur to you must themselves be subjected to the same critical analysis as the existing pattern.

C25.4 PROGRAMME OBJECTIVES.

When the systems consultant feels that he has a sufficient grasp of the operation under examination, he will already have developed some ideas as to how it may be re-organised. Likewise a programmer will by now have some ideas about the general structure of his programme. Certain principles have to be borne in mind in shaping the programme. Some may seem obvious, but often it is the obvious which is overlooked, and often a principle entails questions the answers to which have not yet been clarified.

(1) The required services must be provided at the right time at minimum cost.
(2) Functions should be so allocated as to minimise the required preparation, transportation, co-ordination and paper-work external to the data processor. .

(3) The work of the operator should be as light and as simple as possible. (Conversely, clear indications of emergency measures should be built in, together with as many as possible of the useful criteria that the operator may need for choosing his line of action).

(4) The total time taken should be as low as possible.

(5) The programme logic should be as simple as possible. A modular type of programme is always preferable where the cost in time is negligible.

(6) A choice where one alternative involves less processing time but more handling may be influenced by overall machine loading considerations.

(7) The data processor should not be required to provide any service at a cost greater then the value of that service, e.g. the value of any control should not exceed its cost.

(8) Where possible the programme pattern should allow for ease of amendment in the expected directions of change.

A merit in a programme is its suitability for introduction in parallel with the existing procedures, and preferably for only a cross-section of the work. This would expose the programme to a random selection of the exceptional situations under control conditions.

C25.5 PROGRAMME STRUCTURE.

When we turn to the construction of the programme there are many useful questions that can be asked.

(1) Are results obtained at the most convenient points in the system? Can the input and output requirements be amended profitably?

(2) Is there any input information which could more economically have been derived from other data?

(3) Where sub-functions make use of common derived data, can programming be simplified through a rearrangement of the structure?

(4) Can the structure be split into a main sequence and a secondary sequence of functions which can supply controls for the main sequence?

(5) Know the reasons for each step made and the reasons why it is where it is.

(6) Draw and redraw diagrams of the operation structure.

(7) Check how, when and where the variations, exceptions and special cases are to be dealt with. List the cross-links which lead to and from each of these.

C25.6 PROGRAMME DESIGN.

In any given cycle of operation of a programme each possible path of control has an assignable probability, and for any possible allocation of storage and programme space each path of control in the cycle will take a calculable time. It follows that the optimum solution of the programmer's task is theoretically possible for a cycle of operation and for the programme as a whole.

In practice a programmer will be able to judge which parts of the programme will occupy a negligible proportion of the operation time and will concentrate on optimizing the more important parts of the programme. Nevertheless it must be remembered that the programme is a whole and not a set of separate optimization problems.

The programmer has to resist the temptations both of perfectionism and of trying to write the programme in the shortest possible time. Generally speaking, any refinement which effects a noticeable saving in the final timing will be worth carrying out unless the programme is for infrequent use.

Often a major influence on the total operation time will be some particular transfer time, so that the programmer must keep one part of the equipment running as efficiently as possible.

Sometimes it will be necessary to programme in depth, keeping Card Input, Card Output, Magnetic Tape, Magnetic Drum, Paper Tape and Data Manipulation - or a selection of these - in simultaneous operation, proceeding, so far with one level and then turning to another level clamouring for attention. Here it may be desirable to picture a typical slice of actual operation before deciding on the priorities to be allotted to each level's claim for attention.

With these thoughts in mind we now turn to an insurance file - maintenance demonstration programme. This programme was written for DEUCE Mk.II with Magnetic Tape Auxiliary Storage, but without the facility now available for group-sequenced tape transfers.

C25.7 REQUIREMENTS FOR PROGRAMME.

The Policy file is examined, as it is updated, to ascertain and list those cases which fall due for premium billing in a specified month.

Input to the computer comprises:-

(i) the current file on Magnetic Tape (called the "Existing File"),

and (ii) 80-Column cards bearing full details of the cases for which some "Amendment" to the existing file record will be needed.

Each item in the Existing File is allocated 240 characters on the Magnetic Tape, corresponding to the information capacity of three 80-Column punched cards. Any policy can be adequately represented by an item of this size. In practice 200 characters would be sufficient in most cases and advantage could be taken of this fact to improve the speed of processing. However for the purposes of this demonstration a fixed item length of 240 characters has been adopted.

The Amendments may be divided into three types:-

(i) 'Offs', requiring removal of a policy from the file,

(ii) 'Ons', requiring insertion of a policy into the file,

and (iii) 'Alterations', requiring alteration of some detail of a policy held in the file.

For 'Offs' the Amendment comprises one card only. For 'Ons' the Amendment comprises 3 cards in all cases. For 'Alterations' not affecting the later parts of a policy-record, only the first, or the first two cards, as appropriate, are required.

The last card of a given Amendment is designated by an overpunching in column 1 of that card. All Amendment Cards are assembled in Policy Number order to form the "Amendments File".

Output from the computer comprises:-

(i) the revised Existing File on Magnetic Tape (called the "Updated File"),

(ii) the record on Magnetic Tape of those cases falling due for premium billing in the specified month, (called the "Renewals File"),

(iii) Error Cards giving details of any cases where the operation required conflicts with the available records (e.g. Offs for which no corresponding policy is recorded in the Existing File),

and (iv) indications on the Control Panel if cards in the Amendment File are found to be out of order. In such cases the computer waits for appropriate action by the operator.

Each item in the Updated File comprises three cards-worth of information in the same form as for the Existing File. However in the case of the Renewals File only two cards-worth are recorded unless the length of the Payer's Address demands a third. In respect of cases being entered on the Renewals File a statistical analysis is obtained by Period of Renewal for two classes of policy (Funds 1 and 2).

The block diagrams (Section 11) should be examined in conjunction with the text of Sections 8 - 10. Diagram 11.2 exhibits the normal operation cycle of the computer programme while Diagram 11.3 shows the interrelationships of the various routines into which the programme is subdivided in Sections 8 and 10. In diagram 11.3 the setting of one block into another indicates that it may function at that point as a subroutine of the larger block. Where 'CARD INPUT' occurs at the lower right hand corner of another block it is to be regarded as an alternative exit from that block, subject to the condition indicated by the asterisk.

In the interests of clarity no attempt is made to illustrate the effect of the "Data Incorrect" Routine in diagrams 11.2 and 11.3, nor that of the "Fail" Routine in diagram 11.2. It is hoped that the text of Sections 8 and 10 gives sufficient explanation on these points.

C25.8 GENERAL DESCRIPTION OF PROGRAMME.

Programme Input reads in the programme, storing some parts on the Magnetic Drum.

Entry checks the indicative data at the start of the Existing File (E - File) Tape and records indicative data at the start of the Updated File (U - File) and Renewal File (R - File) Tapes. The required month is inserted via the Control Panel and the programme leads to

Amendment File (A - File) Read which, if data from less than 3 cards are in reserve, refers to

Card Input which (i) calls a further 80-Column card read operation

(ii) stores the data from the last card read, first in the high speed store and then on to successive tracks of the magnetic drum when appropriate,

and (iii) adjusts the card reserve count by unity.

When at least 3 cards are in reserve A-File Read proceeds. Details of 1 case are taken to the A-Store from the developing A-File on the Magnetic Drum (* See notes 1 and 2). (If the policy number of this case is lower than the previous one,

Data Incorrect offers the operator the alternatives of

(i) ignoring this case or

(ii) reprocessing from the point where this case should occur, either by following it from there from the A-File on the Magnetic Drum or by correcting the order of the cards and reloading the A-File.)

(If A-File Read was entered from Re-entry 3 the programme skips to Organisation (Part 1), but otherwise:-)

Re-Entry 2 follows, leading to

Tape Read which reads details of 1 case from the E-File Tape to the E-Store (* See note 1).

Organisation (Part 1) then compares the policy numbers of the E-Store and A-Store cases and, if the E-Store policy number is the lesser, selects instructions for recording that case on the U-File Tape and selects data from that case for Premium Billing Test.

Organisation (Part 2) is entered if the E-Store policy number is not the lesser. Instructions and data appropriate to the A-Store case are selected for the U-File write operation and Premium Billing Test, provided that the policy number relationship is consistent with the type of amendment required. If an error is suspected

Fail is used to punch out details of the suspect record, leading for 'offs' only to Re-entry 3. In the case of 'Offs' the programme now skips to Re-entry 1.

Tape Write makes the appropriate entry on the U-File Tape. (* See notes 1 and 3).

Premium Billing Test next examines whether this case is due for renewal in the required month. If not, then the programme skips to a Re-entry (1, 2 or 3) previously selected by Organisation as being appropriate to the type of operation developed there.

Billing Accumulation, if this case is due for renewal, accumulates the premium in binary pence by renewal period and fund.

Tape Write enters the case on the R-File Tape (* See note 1) and leads to the Re-entry previously selected by Organisation as being appropriate to the type of operation developed there.

Re-Entry 1 leads to A-File Read.

Re-Entry 2 leads to Tape Read.

Re-Entry 3 leads to A-File Read but then skips to Organisation.

NOTE 1 The programme tests at these points whether the card input is in operation. If not, Card Input is entered before the link to the next section is obeyed.

NOTE 2 If the new case for the A-Store has a policy number less than its predecessor the programme diverts to Data Incorrect, which (at the option of the operator) -

(1) ignores the case and proceeds to the next one or

(2) accepts the case, turns back the files to the correct place and either

(a) prepares to take again all later numbers from the A-File or

(b) prepares to restart the A-File as from this point from the Amendment Cards.

NOTE 3 If the final case has been dealt with, the programme here diverts to Final (Parts 1 and 2) which reconverts the billing accumulations to sterling, records them on the R-File and writes end of file indications.

C25.9 STORAGE USED.

Magnetic Tape deck 0 carries the Existing File (E-File) Tape.
" " " 1 " " Updated File (U-File) Tape.
" " " 2 " " Renewals File (R-File) Tape.
Magnetic Drum tracks 246-250 carry the Data Incorrect Routine instructions.
Magnetic Drum tracks 251-253 carry the Final Routine instructions.
" " " 254 carries a copy of Delay Line 3.
" " " 255 " the Fail Routine instructions.

Mercury Delay Lines 1-6 (All mcs), 7 and 9 (mcs 24-31) and 8 and 10 (mcs 16-23) carry the Main Programme.

Mercury Delay Lines 7 and 9 (mcs 0-23) and 8 (mcs 0-15 and 24-31) carry the Entry Routine but are cleared at the end of that Routine.

During the File up-dating Mercury Delay Lines 7 (mcs 0-23) and 8 (mcs 0-15 and 24-31), together called the E-Store, carry the item from the Existing File currently under consideration and Mercury Delay Lines 9 (mcs 0-23) and 10 (mcs 0-15 and 24-31), together called the A-Store, carry the item from the Amendment File currently under consideration.

Mercury Delay Line 11 is the buffer-store allotted to data being transferred to or from the Magnetic Drum. Mercury Delay Line 12 is the buffer-store allotted to data being transferred to or from the Card Input/Output. Magnetic Drum Tracks from 0 onwards store the Amendment File as it is received from Delay Line 12.

C25.10 DESCRIPTION OF PROGRAMME SECTIONS.

(a) Entry Routine.

Write record gap and description of file on Tape decks 2 and 1. Verify description of file on tape deck 0. Set up initial link instructions, initial magnetic drum read and write instructions. Read required month from Control Panel. Initiate reading of 1st card. Restore magnetic drum read and write heads to block position 0. Clear D.L's 7 and 9 (mcs 0-23), 8 (mcs 0-15 and 24-31) and 11.

(b) Card Input Routine.

Test whether the card just read is the Final Item card marking the end of the file. If so, plant large negative card count marker, decall reader, and overwrite 'call read' instructions with dummy instructions. If not the final item, discriminate bewtween 1st card, 2nd card, 3rd card and single card groups. If single card group, or 3rd card, clear the 'last card of group' designation recorded from column 1 of the card. If 1st card, call further card read to D.L. 12 mcs 16-31 (=store for 2nd cards). If 2nd Card, single card group, or 3rd card -

(i) call further card read to D.L.12 mcs 0-15 (= store for 1st/3rd cards); then (and also for Final Item)

(ii) transfer DL12 to DL11 and thence to the next track of the Magnetic Drum, shifting write head blocks if necessary, and

(iii) clear DL12 mcs 16-31 (store for 2nd cards). For all cases reduce card count marker by 1 (card count marker commences at +2).

(c) A-File Read Routine.

Test whether card count marker is negative (i.e. whether either 3 cards at least are in reserve, or the card input operation is complete). If marker is positive -

(i) x await end of current card read operation,

(ii) call Card Input Routine and

(iii) test card count marker again, repeating as necessary.

If marker is negative increase it by one and read next track of magnetic drum to DL11, shifting read head blocks if necessary. If the new policy number is the lesser the programme enters Data Incorrect Routine. If track contains 2 cards increase card count marker by 1, transfer data to the first 2 sections of the A-Store and test whether this amendment is a 2 card set (X overpunching in col. 1 of 2nd card). If not, increase card count marker by 1, read next track of magnetic drum to DL11, shifting read head blocks if necessary, and transfer data to the 3rd section of the A-Store. If amendment is a 1 card set, transfer contents of DL11 to the first 2 sections of the A-Store. In the case of 1- or 2- card amendment sets clear the 3rd section of the A-Store. Test whether the card input mechanism needs priming and if so refer to Card Input Routine before obeying link. (Entry to this routine from the Entry Routine is at the point marked (x).)

NOTE: As the time taken by this Routine exceeds the time margin within which recalling the card reader ensures continuous running, temporary measures are taken during this routine to maintain the continuous running.

(d) Data Incorrect Routine.

An indication is given on the control panel that the latest policy number taken from the A-File is less than the previous number in that file and the machine awaits selection of one of the following options by the operator:-

(1) Overwrite the new item with the next following item from the A-File, then link to the Tape Read on the Organisation Routine as appropriate.

(2) Set back the E-, U-, and R-Files to the block gap after the last item having a policy number less than this new A-File number, complete the A-File Read operation and then either

(a) set back A-File read instructions to the track having the first policy number greater than this A-File number and refer to Re-entry 2, or

(b) set back A-File write and read instructions to track 0, set back card count to +2 and refer to Re-entry 2.

(e) Tape Read-Write Routine.

According to the nature of the instructions previously prepared for this routine,

(1) read a block of 24 word pairs from the E-File Tape via DS20 to the E-Store, or

(2) write a block of 24 word-pairs on the U-File Tape via DS20 either from the E-Store or from the A-Store, or

(3) write on the R-File Tape a block as specified in (2), or

(4) write a block of 16 word pairs on the R-File Tape via DS20 either from the first 2 sections of the E-Store or from the first 2 sections of the A-Store. Wher a block is being written on the R-File, alternative (4) is preferred to alternative (3) unless the address spills to the 3rd section of the store (= card 3). Test whether the card input mechanism needs priming and if so refer to Card Input Routine before obeying link.

The Note to the A-File Read Routine applies here.

(f) Re-entry 1.

This link plants a link for Re-entry 2 and then enters the A-File Read Routine.

(g) Re-entry 2.

This link plants a link for the Organisation Routine, prepares the Tape Read Instructions, and then enters the Tape Read Routine.

(h) Re-entry 3.

This link plants a link for the Organisation Routine, and then enters the A-File Read Routine.

(j) Organisation Routine (Part 1).

The Re-entry link for Re-entry 2 is prepared. The Due Date and Premium Period specified in the E-Store case are noted in T.S.16. The quantity (E-Store Policy No. minus A-Store Policy No.) is calculated in DS21. The parity check on the tape-read operation (if any) is now available and is consulted. If there has been a parity failure provision is made for re-reading the suspect block. Unless the E-Store Policy number is less than the A-Store number, Part 2 is now entered.

If the E-Store No. is the lesser, instructions for tape write on the U-File Tape from the E-Store are prepared, the E-Store case Fund and Address Spill indications are noted in T.S.15 and the Premium in D.S.19. The Note to the A-File Read Routine applies here.

(k) Organisation Routine (Part 2).

The Re-entry link is amended for Re-entry 3 and the A-Store case Due Data and Premium Period recorded in T.S.16. Instructions are prepared for Tape write on the U-File Tape from the A-Store. If the amendment is an "Off" the routine tests that E-Store and A-Store policy numbers are equal. If they are equal a link to Re-entry 2 is planted and the A-File Read Routine entered. If they are unequal (i.e. if the A-Store No. is the lesser) an indication for Re-entry 3 is planted and the Fail Routine entered. If the amendment is an "On" the routine tests whether the A-Store contains the full 3 cards. If it contains less than 3 cards appropriate transfers are made from the E-Store to the A-Store. If the "On" is 'Alteration' and the policy numbers are unequal, or if the "On" is 'New business' and the policy numbers are equal, an indication for return to Organisation Routine (Part 2) is planted and the Fail Routine entered. If the 'On' is 'Alteration' and the policy numbers are equal the Re-entry link is amended for Re-entry 1. If the case is the Final Item a link to Final Routine is planted, appropriate transfers are made from DL's 7 and 8 to DL's 9 and 10 and the routine checks that the policy numbers are equal For all "Ons" the A-Store Fund and Address Spill indications are noted in T.S.15 and the Premium in D.S.19.

(l) Fail Routine.

The Fail Routine instruction track (No. 255) is read from the magnetic drum to DL.3 via DL11. When the current card read operation is complete the contents of of DL12 are transferred to DL11. The contents of the A-Store (if the error is an "Offs" or an "Alteration" error) or of the E-Store (if the error is a "New Business" error) are punched on to 3 cards via DL12. If the error is a "New Business" error the 1st 5 columns of the 1st card are overpunched X and the Re-entry link is amended for Re-entry 1. The former contents of DL12 are re-instated from DL11, the Main Programme contents of DL3 are re-instated from track 254 of the Magnetic Drum via DL11 and the Drum Read head block is returned to its former position. According to the indication planted the Fail Routine leads either back to the Organisation Routine (Part 2) or to Re-entry 3.

(m) Premium Billing Test Routine.

Instructions are prepared for tape write on the R-File Tape from whichever store is appropriate (the E-Store unless organisation Routine (Part 2) was entered). By examining the value 'Due Month - Required Month' and the premium period a decision is made as to whether premium billing is required in this case. If premium billing is not required test whether the card input mechanism needs priming and if so refer to the Card Input Routine before obeying the link to the selected Re-entry. If premium billing is required, test whether address spill is indicated and if not indicated amend tape write instructions to limit the block to 16 word pairs.

(n) Billing Accumulation Routine.

The premium in Binary Coded Sterling is converted to a premium in Binary pence. The instructions which:

(1) add the premiums so far accumulated to this premium and

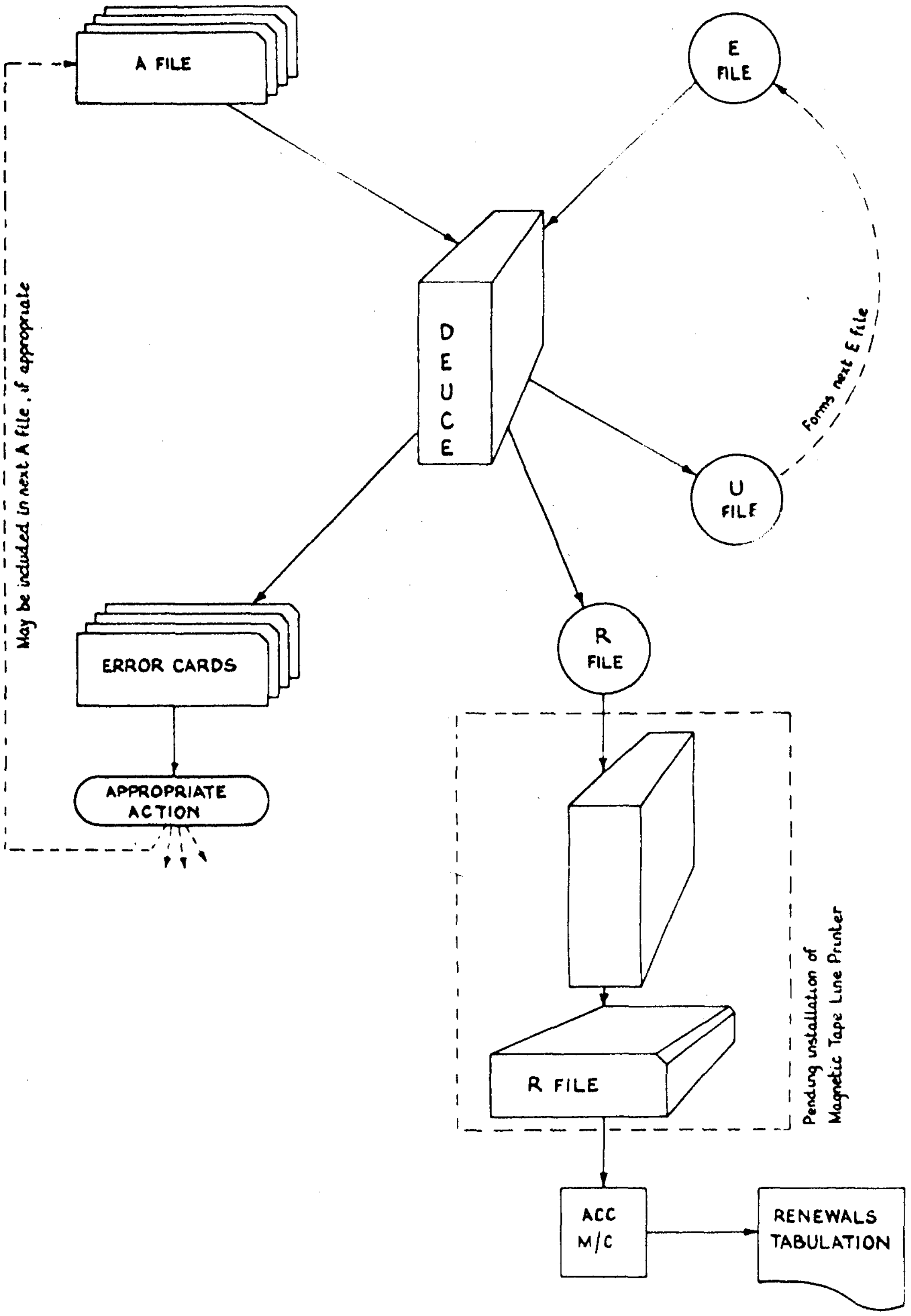
(2) store the new accumulated total are each in turn amended to select the minor cycle in which is stored the total appropriate to the premium period and fund of this case and then obeyed.
The Note to a A-File Read Routine applies here.

(p)
(q) Final Routine (Parts 1 and 2).

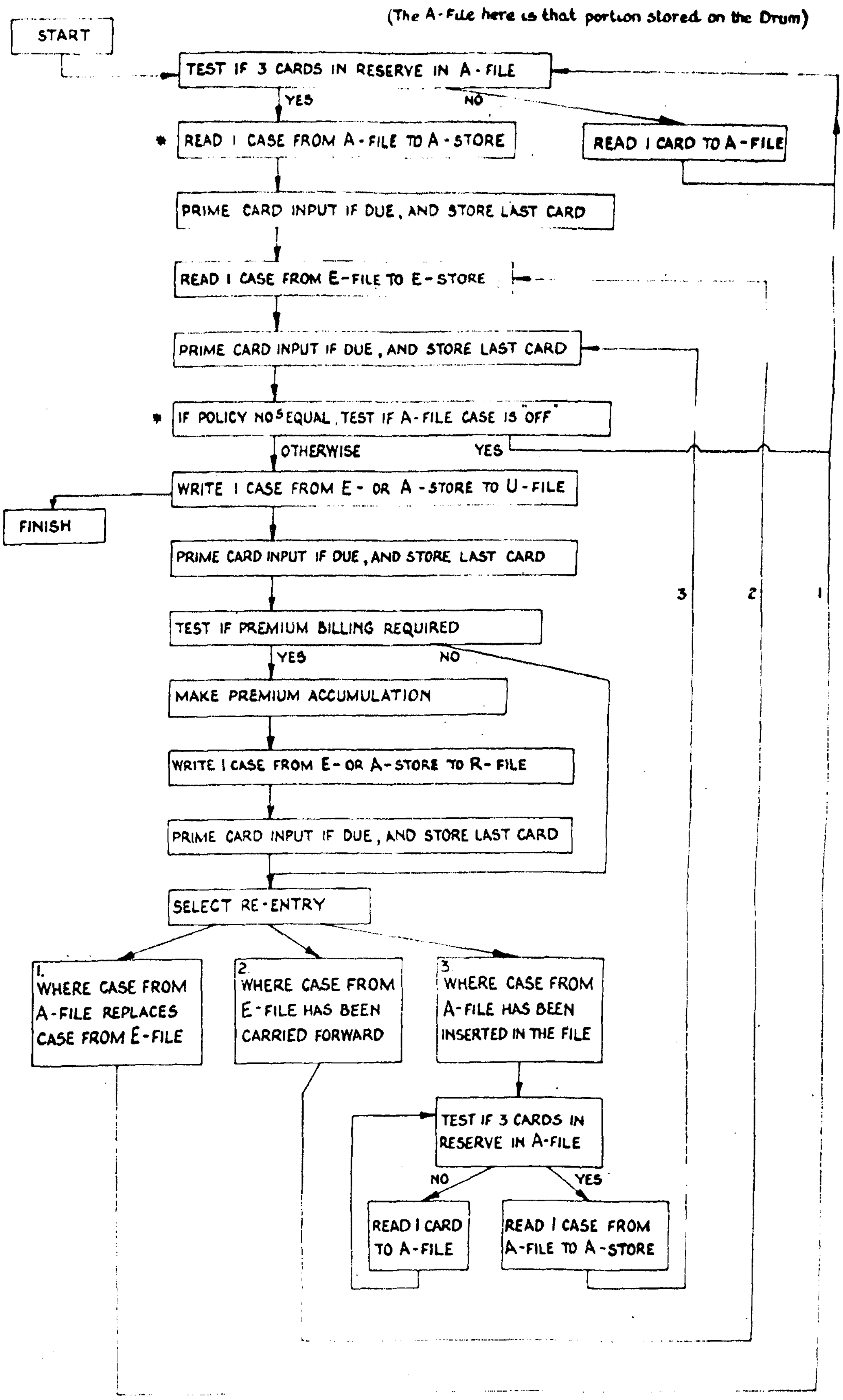
The Final Routine instruction tracks (251-3) are read from the Magnetic Drum to D.L.'s 2, 3 and 4 via D.L.11, D.L.6 is cleared to accept the Binary Coded Sterling equivalents of the premium billing accumulators. Part 2 is referred to to perform the conversions from Binary Pence to Binary Coded Sterling and to store the results in D.L. 6. Returning to Part 1 the Routine writes a block of 11 word pairs on the R-File Tape via DS20 comprising a last block marker and the 10 accumulated billing totals. End of File markers and final record gaps are written on the R-File and U-File Tapes which are then rewound. The programme ends by switching on the Output Staticisor lights on the control panel.

25/11 BLOCK DIAGRAMS

G.25.11



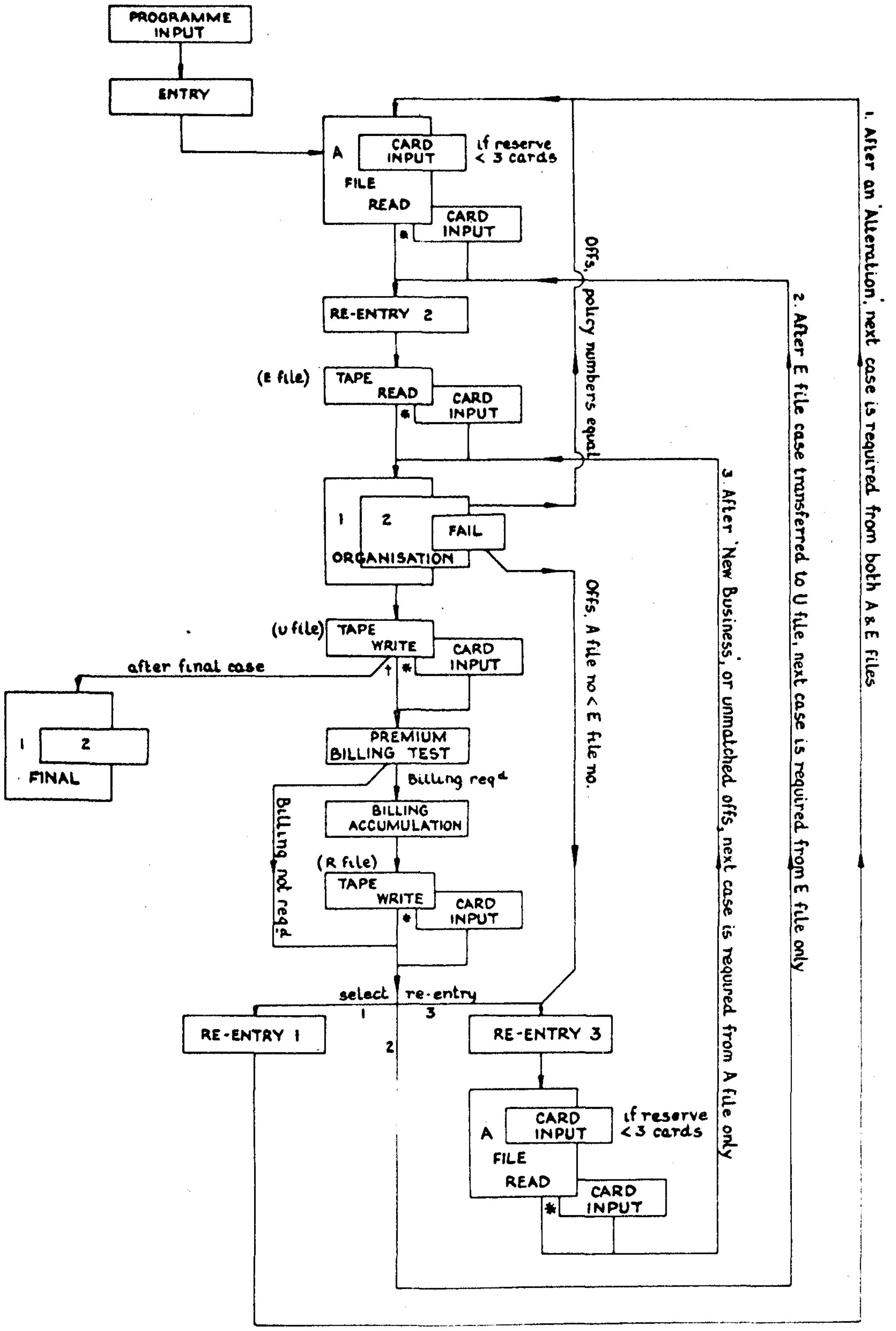
25/11·1 SYSTEM FLOW OF INSURANCE DEMONSTRATION PROGRAMME :- FILE MAINTENANCE AND FORMATION OF RENEWAL FILE


START
TEST IF 3 CARDS IN RESERVE IN A-FILE
YES
NO
READ 1 CARD TO A-FILE
* READ 1 CASE FROM A-FILE TO A-STORE
PRIME CARD INPUT IF DUE, AND STORE LAST CARD
READ 1 CASE FROM E-FILE TO E-STORE
PRIME CARD INPUT IF DUE, AND STORE LAST CARD
* IF POLICY NOS EQUAL, TEST IF A-FILE CASE IS "OFF"
OTHERWISE
YES
WRITE 1 CASE FROM E- OR A-STORE TO U-FILE
FINISH
PRIME CARD INPUT IF DUE, AND STORE LAST CARD
TEST IF PREMIUM BILLING REQUIRED
YES
NO
MAKE PREMIUM ACCUMULATION
WRITE 1 CASE FROM E- OR A-STORE TO R-FILE
PRIME CARD INPUT IF DUE, AND STORE LAST CARD
SELECT RE-ENTRY
1. WHERE CASE FROM A-FILE REPLACES CASE FROM E-FILE
2. WHERE CASE FROM E-FILE HAS BEEN CARRIED FORWARD
3. WHERE CASE FROM A-FILE HAS BEEN INSERTED IN THE FILE
TEST IF 3 CARDS IN RESERVE IN A-FILE
NO
YES
READ 1 CARD TO A-FILE
READ 1 CASE FROM A-FILE TO A-STORE
3
2
1

(The A-File here is that portion stored on the Drum)

* The diagram is simplified by ignoring the effect of incorrect or out of order A-cards

25/11·2. LOGICAL OPERATION OF INSURANCE DEMONSTRATION PROGRAMME:- FILE MAINTENANCE AND FORMATION OF RENEWAL FILE



* If card input requires priming, CARD INPUT is called

† When final case is written on U file, FINAL (1 + 2) are called

25/11·3 LOGICAL STRUCTURE OF INSURANCE DEMONSTRATION PROGRAMME

FILE MAINTENANCE AND FORMATION OF RENEWAL FILE

LECTURE 26.

FIXED POINT AND FLOATING POINT ARITHMETIC.

26.1 INTRODUCTION

The following two columns of numbers are identical in meaning:

30,000,000,000	3	$\times 10^{10}$
0.001118	1.118	$\times 10^{-3}$
3.142	3.142	$\times 10^{0}$
1760.	1.760	$\times 10^{3}$
-0.5	-5	$\times 10^{-1}$

These numbers cover a very wide range; if they were to be stored in a computer in their natural form (left hand column) storage registers of at least 17 decimal digits capacity would be required. However, none of the numbers is given to more than 4 significant figures, as can be seen in the right hand column. We see here how a register of only six decimal digits (four for the "Mantissa" and two for the exponent of the radix 10) can accomodate the same range of numbers as a much larger register. The point is that in the left hand column the decimal point is fixed in position, and non-significant zeros have to be inserted; in the right hand column only the significant figures are expressed, together with an integer of one or two digits to indicate the position of the decimal point. We say the numbers are expressed in FIXED POINT and FLOATING POINT form, respectively.

Let us consider the problem of computing with numbers expressed in the two forms. There is little difference in multiplication or division between the two systems: but for addition or subtraction floating point numbers have to be expressed with a Common exponent and the result normalised, whereas fixed point numbers can be added or subtracted directly.

In the great majority of Scientific calculations we find ourselves in difficulties due to the finite size of the computing registers. If the decimal (or binary) point is fixed, and the numbers involved in the calculation have a wide range of values, there may be a considerable loss of significant figures at the lower end of the range, or a danger of the variable exceeding the register capacity, or, in extreme cases, of both. Pure fixed point working is thus impossible in these cases.

The solution to the difficulty depends very much on the programmer's prior knowledge of the behaviour of the quantities in the calculation.

If he can predict accurately upper limits for every intermediate quantity in the calculation, he can programme shifts of the binary point at appropriate stages so that, on the one hand, his numbers do not exceed capacity, nor do they lose significant figures by being too far down in the register. The programmed shifts will be the same whatever particular data are being processed at the moment; in other words, for any one operation, the binary point will be in the same position whatever numbers are involved. This system is the one usually referred to as FIXED POINT working, because the programmer fixes the position of the binary point for each operation.

If, however, the programmer cannot predict the behaviour of the numbers, or if the variables are known to take such a wide range of values that any standard shifting arrangement would be unsatisfactory, then FLOATING POINT working is adopted. In this system use is made of subroutines to Calculate an appropriate shift for each operation; each number must therefore carry with it an exponent to indicate the position of its binary point.

The two systems will be described more fully below.

In what follows the following notation is used consistently:

x (small letter) - a number to be represented in the computer.

X (capital letter) - the integer actually stored (DEUCE is regarded as operating exclusively on integers).

26.2 FIXED POINT WORKING ON DEUCE

In general in normal fixed point working we have the relation $x = X.2^{-p}$ where p is an integer, constant for the variable x. For maximum accuracy, p will be chosen so that for the upper limit of x, X will be on the verge of exceeding capacity. We say either x is stored to p binary places (which means that the binary point is p places up from the lower end) or that we store $X = x. 2^{p}$ and talk consistently about the numbers as integers. The latter idea is simpler, and is adopted here as far as possible.

In general p is never stored explicitly in the machine, but is implicit in the shifting instructions of the programme, as the following examples will show.

(1) Shift

The instructions 24 - 14, (n m.c.) and 21 - 22, (2n m.c.) (TCB off) will shift a number up n places in 14 or 21. However, care must be taken that the most significant digit of the number is not "spilt" by being shifted too far.

In shifting down there is the problem of round off. The following sequences shift down a single length number n places, with correct round off:

X - 14	X - 21_3
23 - 14(m-1)m.c.	22 - 21 (2n m.c.)(TCB off)
14 - 13	21_2 - 23_2
27 - 25	21_3 - Y
23 - Y	

($Y = X.2^{-n}$, rounded off to nearest integer)

(2) Addition and Subtraction.

Consider operations on $x = X.2^{-p}$ and $y = Y.2^{-q}$ giving results $z = Z.2^{-r}$.

The simplest ease is when $p = q = r$.

X - 13
Y - 25/26
13 - Z

If $p = q \neq r$ the same sequence will do, but it must be followed by a shift up of (r - p) places (if (r - p) is negative this is of course a shift down).

There is a possibility of X + Y exceeding capacity; this can be avoided by limiting the most significant digit of each to the P_{30} position.

If $p \neq q$ we have to shift down whichever of X,Y is associated with the higher of p,q, not forgetting the round off. The shift is (p - q) places. Addition or subtraction is then possible; another shift is required if min $(p,q) \neq r$. If this shift is down a round off is necessary.

(3) Multiplication.

(Sign connection assumed to be understood).

The multiplier multiplies the integers in 16 and 21_2 producing a double length result in 21.

With the notation as before, we have

$$X.Y = x\,y.2^{p+q} = z.2^{p+q} = Z.2^{p+q-r} \quad \text{in } 21_{2,3}$$

$$\text{i.e.} \quad Z.2^{p+q-r-32} \quad \text{in } 21_3$$

We have Z (the required result) in 21_3 if we shift up $(32 + r - p - q)$ places (TCB off).

(4) Division.

The divider divides the integer in 21_3 (X) by the integer in 16 (Y) and gives the integer

$$\frac{X}{Y}.2^{31} \quad \text{in } 21_2$$

Now, $$\frac{X}{Y}.2^{31} = \frac{x}{y}.2^{31+p-q} = z.2^{31+p-q} = Z.2^{31+p-q-r}.$$

Therefore, to give the required result Z the number in 21_2 must be shifted down, with round off, $(31 + p - q - r)$ places.

Warning.

This shift must be done with TCB still ON.

The detailed rules about the divider, given elsewhere, must be carefully observed, or a division subroutine used.

For same useful remarks on multiplication and division subroutines, see R.A. Smith, DEUCE News No. 26. (the thick grey-covered book).

26.3 FLOATING POINT WORKING ON DEUCE.

The use of floating point subroutines makes it unnecessary, for most purposes, to think about shifting, binary points, exceeding capacity, and scaling factors for reading and punching.

The basic idea is that every number x is represented in the machine in two parts, corresponding to the x and p of the previous chapter. In one scheme these two parts are exactly x and p as previously defined, but in others the two parts are defined differently. Precise definitions are given in the following paragraphs.

In any floating point scheme the subroutines treat each quantity on its own merits and deal with shift, limits and round off accordingly.

For each scheme there is a normalised form. The read routines take numbers punched in STANDARD FLOATING DECIMAL (that is, such that $x = \alpha.10^{\beta}$, where $1 \leqslant |\alpha| < 10$ and β is an integer) and leave standard binary floating numbers in store. Each arithmetical routine takes standardised numbers and produces standardised results. There is usually a subroutine called PREPARE which takes a non-standard number and standardises it. It is usually possible to punch standard decimal numbers.

(1) Standard Floating Binary (s.f.b.)

A number n is represented by (a,b) such that

$$x = a.2^{b}$$

and $\quad -1 \leqslant a < -\frac{1}{2}$

or $\quad a = 0$ (in which case $b = 0$ also)

or $\quad \frac{1}{2} \leqslant a < 1$

and $\quad b$ is an integer.

a is stored to 30 b.p. in an odd minor cycle (i.e. the integer $a.2^{30}$ is stored)
b is stored as an integer in an adjacent even minor cycle.

A simpler way of remembering the hints on a is this: the most significant digit of a is always P_{30}, except of course when a = 0.

The recommended subroutines are A13P to A16F. However it may be necessary to use an older set to fit in with some higher order subroutine (e.g. G03F). Care should be taken since these older subroutines may contain subtle errors. The following read and punch subroutines are recommended:

R19F	R22F	R22F/1
	P10F/2	P13F/4

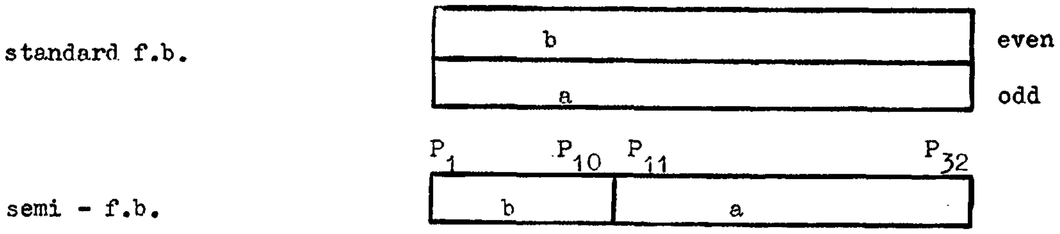
Standard floating arithmetic is used in ALPHACODE.

(2) Semi-floating Binary.

Standard floating binary numbers occupy two complete words each.

Where space is valuable, and where accuracy to more than 6 significant decimal figures is not required, SEMI-FLOATING BINARY can be used: ("semi" refers to space requirement, not to "floating").

The exponent b is restricted to 10 digits (i.e. $-512 \leqslant b < 512$) and these replace the bottom 10 digits of a, which is rounded off at this point.



The limit on b is not restrictive; the loss of significant figures is of greater importance. The operation of semi floating binary is slower than that of s.f.b. since packing and unpacking are necessary.

(3) Block-floating Binary (Scheme B)

This is another space saving compromise, but without the loss of accuracy of semi-floating binary. It relies for its success on the fact that normally the elements of a matrix (or other array) have the same order of magnitude (because, for example, they have the same physical meaning - pressures, stresses, coefficients, etc.).

Each element x_i of the array is represented by its own X_i; but there is only one p for the whole array. The value chosen for p is such that the most dignificant digit of the element (X_i) of longest modulus is P_{30} (c.f. notes on standard floating binary).

In scheme B matrix convention, an array of m,n elements (m rows, n columns) is stored in mn + 4 consecutive minor cycles of the drum, beginning with m.c. 0 of a track, as follows:

1st track,	m.c. 0.	$\Sigma\Sigma$
	m.c. 1.	$m\,P_{17}$
	m.c. 2.	$n\,P_{17}$
	m.c. 3.	$p\,P_{17}$
	m.c. 4.	X_{11}
	m.c. 5.	X_{12}
		etc.

($\Sigma\Sigma$ is the literal sum of the next m.n. + 3 minor cycles).

The relationships between x, X, p is as follows:

$$x_{ij} = X_{ij}.2^{-p}.$$

(p is the number of binary places to which x_{ij} are stored).

26.4 FLOATING AND UNFLOATING A NUMBER.

(1) When working in s.f.b. it is sometimes necessary to "prepare" (or normalise) a fixed point number to s.f.b.

Suppose the number is $x = X.2^{-p}$ x is stored to p b.p.

We require	$x = a.2^{b}$
where	$-1 \leq a < -\frac{1}{2}$
or	$a = 0$
or	$\frac{1}{2} \leq a < 1$

We have X and we know p. We require a, b

and	$a.2^{b} = X.2^{-p}$
therefore	$a = X.2^{-p - b}$
Now	$a = A.2^{-30}$ where A is an integer.
therefore	$A.2^{-30} = X.2^{-p - b}$
therefore	$A = X.2^{30 - p - b}$

Therefore the shift up to produce A is

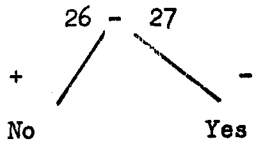
$$s = 30 - p - b$$

therefore $b = 30 - p - s$

Where s is the shift up required to bring the most significant digit to the P_{30} position.

The following sequence of instructions tests whether the most significant digit of TS 14 is a P_{30}

```
24 - 14
24 - 15
26 - 27
```



(2) The converse problem may also occur. We have a standard floating binary number,

$x = A.2^{-30 + b}$ and we wish to produce

$n = X.2^{-p}$ where p is specified.

We have	$X.2^{-p} = A.2^{-30 + b}$
therefore	$X = A.2^{-30 + b + p}$

Therefore the shift down is $s = 30 - b - p$ places.

Note that we cannot have $s < -1$ (i.e. we cannot shift up more than one place) because this will cause the most significant digit to be lost. A shift up of 2 places is, however, theoretically possible in the case of unsigned numbers.

26.5 POINTS TO CONSIDER.

Fixed point working is fast in operation,economic in data storage but is difficult to programme (and programmes are difficult to modify), requires knowledge of behaviour of all variables and intermediate quantities, loses accuracy if a variable has a very wide range.

S.f.b. working is simple to programme, requires no knowledge of behaviour of quantities, is good for problems where wide ranges are involved but is slow in operation (especially add, subtract), requires double storage space.

Semi floating binary working needs no more space than fixed but gives far less accuracy and is a little slower than s.f.b. There is a slight danger of exponent spill.

All the above assumes that single length working is sufficiently accurate for the problem. If it is not double length working may be necessary, but this is beyond the scope of the present note.

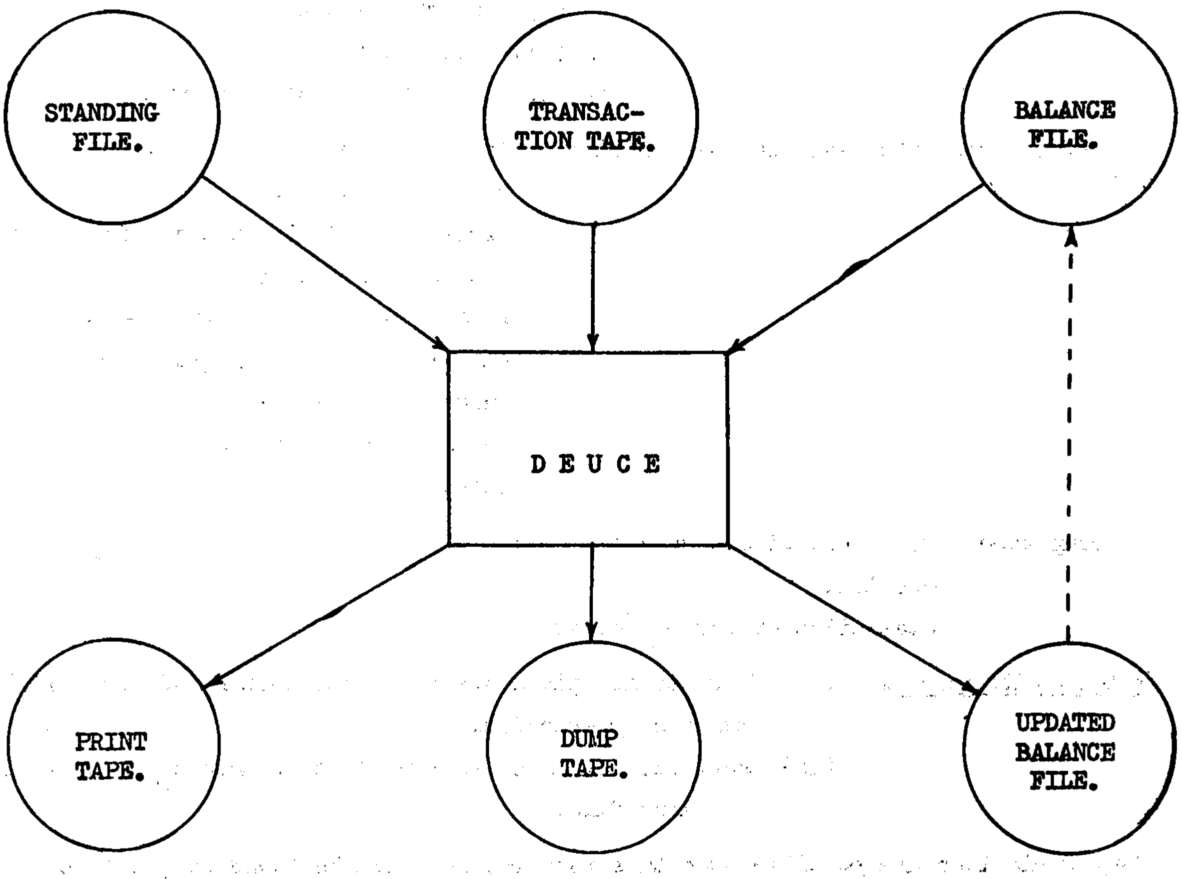
LECTURE 26.

MAGNETIC TAPE FILE UPDATING.

A. GENERAL CONSIDERATIONS OF FILE UPDATING ON MAGNETIC TAPE.

C26.1 Outline of Method.

Routines may be expected to take something like the following form:-



The standing file contains details needed for reference; it is never amended on the routine run, though like all works of reference it needs to be revised periodically.

The balance file contains current information subject to change during the updating run. The whole file is rewritten, incorporating the changes which arise, and the updated balance file becomes the input for the next run.

The print tape contains the output which it is desured to print at once in the order in which it comes from the updating run, whilst the dump tape contains output which requires further processing or ordering, or is to be held back and printed after the print tape.

Example 1.

On a public utility billing run, the various tapes might well contain:-

Standing file: (for each customer): Account number,
Name and address,
Key to scale of charges applicable,
Details of Hire Agreements.

Balance file: (for each customer): Account number,
Cash due or overpaid,
Details of current Hire Purchase Agreements and amount outstanding thereon,
Sundry statistics.

Transaction tape: (each transaction): Account number, and details of either:-
(i) Meter reading,
or (ii) Cash paid
or (iii) New Hire Purchase Agreement.

Print tape: (for each customer billed): Account number,
Name and address,
Present and last meter readings, consumption, rate and extension,
Amount due on Hire Agreements,
Amount due and remaining outstanding on Hire Purchase Agreements,
Balance brought forward,
Total due.

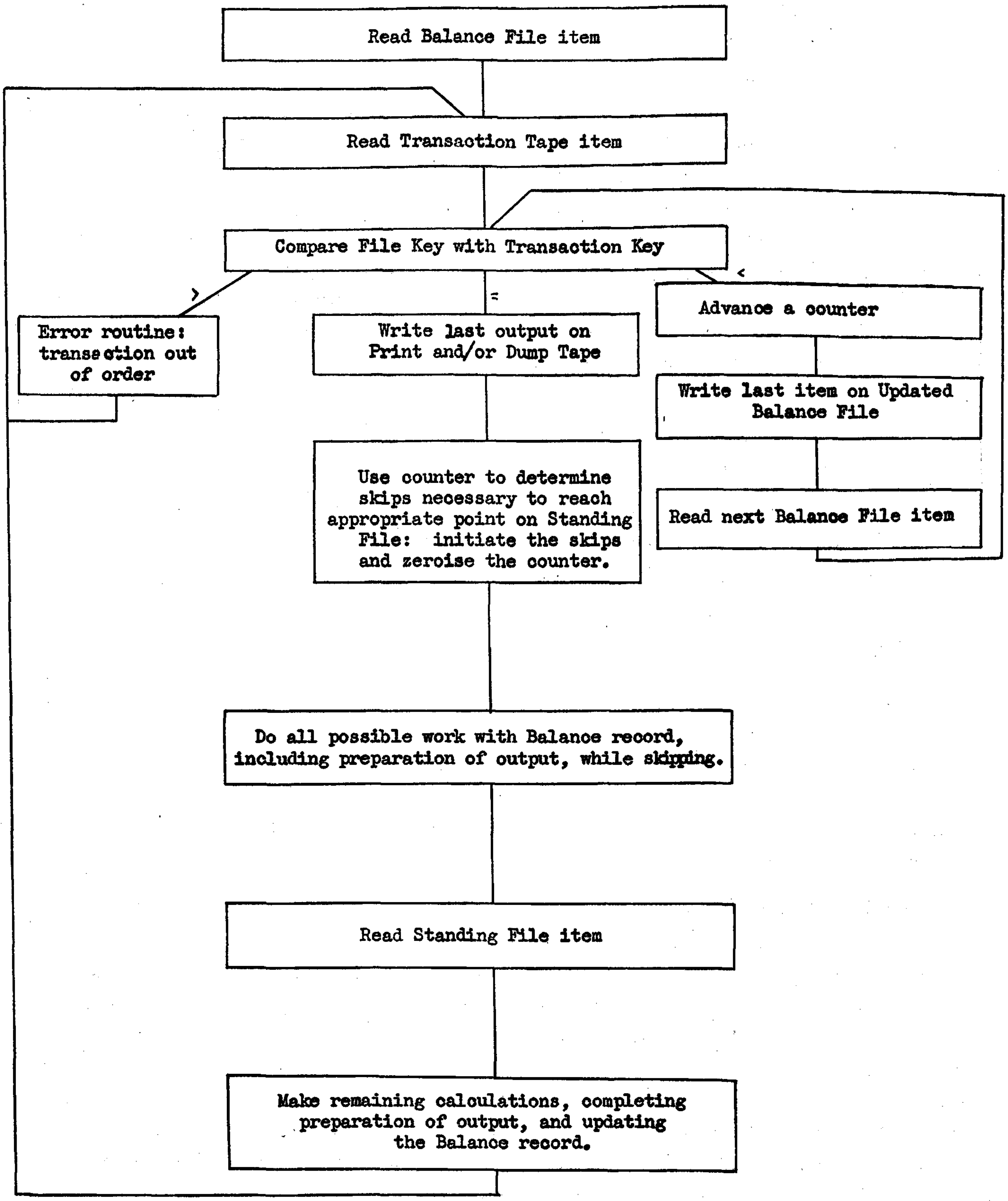
Dump tape: Reminders of accounts overdue;
Queries;
Statistics of various kinds.

Notes on Example 1. (i) The details given are for illustration only and are by no means comprehensive.
(ii) Periodically an extra run would be needed to revise the standing file.

Many variations are possible upon this pattern, and will be found in practice. Thus it will sometimes pay to consolidate the standing and balance files into a single record as in our banking illustration in part B; sometimes a dump tape is unnecessary, e.g. in a Payroll routine if it was necessary to dump only departmental abstracts, cash breakdowns and cost analyses, the drum might offer adequate capacity. Sometimes a tape-printer will not be available and output will have to be on cards; sometimes transaction input will be on pre-sorted cards; these last two cases represent wide variations from the standard pattern and will be given separate consideration at several points in this lecture.

Reverting to the standard pattern a simplified version of the procedure will be along these lines:-

Example 2.


Read Balance File item
Read Transaction Tape item
Compare File Key with Transaction Key
>
=
<
Error routine: transaction out of order
Write last output on Print and/or Dump Tape
Advance a counter
Write last item on Updated Balance File
Read next Balance File item
Use counter to determine skips necessary to reach appropriate point on Standing File: initiate the skips and zeroise the counter.
Do all possible work with Balance record, including preparation of output, while skipping.
Read Standing File item
Make remaining calculations, completing preparation of output, and updating the Balance record.

The above diagram does not represent quite the best technique, because not enough of the calculation is done in parallel with tape-skipping. It would be better to look slightly ahead and be positioning the Standing File ready for the next item whilst doing all the work on the current item. This kind of consideration becomes of paramount importance where card input and/or output is being used. Here the technique is to be working on the 2nd transaction while reading the 3rd and punching the results of the work on the 1st transaction.

C26.2 Precautions needed.

(a) Protecting the Tape.

The greatest danger is that of over-writing by mistake information that will be needed again. A normal precaution will be to retain each tape for at least one period (where period represents the frequency of the updating run) after it has been used to generate a fresh tape. This would enable the files to be reconstructed after an accident; nevertheless it should be looked on as a last line of defence, for such reconstruction is at least costly in terms of time. Therefore further precautions are needed, and two will usually be adopted: (i) tape identification, and (ii) use of the inhibitor ring, which prevents writing on any tape bearing the ring. Thus a safeguard is obtained against both types of error: (i) operator loading the wrong reel, and (ii) machine writing when not called on to do so. The usual system will incorporate the following points:-

(i) Every tape to begin with an identification number to correspond with one visibly written on its container;

(ii) Machine on calling for reload to indicate to operator the identity of the tape coming off and that of the tape to go on;

(iii) Machine on starting a new tape to read its identification block and check that this is the one called for.

(iv) All tape reels to be ringed, except when being loaded as output tapes.

(b) Other Precautions.

Automatic parity checking is available, and should always be used. It may also be advisable to sum each block as it is being written on tape, and to write the sum at the end of the block. On reading the sum can then be checked in addition to parity. In general, too, all computer routines should include at least the same checks as are included in manual systems, like double entry, cross-footing and control totals.

C26.3 Reproducing the Balance File.

In the foregoing it has been assumed that the whole balance file will be reproduced on each run, including those records on it which are unchanged. However, where there is a low "strike density" on the file, i.e. where quite a large proportion of the records are unaffected, it may pay to introduce a supplementary Balance File containing details only of the revised records. In this case an extra tape station is needed; the main and supplementary Balance Files are read and an updated supplementary written. Of course the supplementary file gradually increases in length and eventually reaches a point where no saving is apparent from its use. A special run is then needed to update the main Balance File introducing all the revised records, and the supplementary file is started from scratch again.

On a Savings and Deposit Account routine for one bank, it was found that of nearly half a million accounts, only about 7,000 were affected each day. This represents a strike density of about 1.4% and here it would clearly pay to keep a supplementary file, rewriting the main file about once a month.

In such a case it is also possible to employ the technique of keeping an active and an inactive file. Accounts diplaying a very low level of activity are culled from the main file and kept on an inactive file; whenever a transaction does occur on one of the accounts thus culled it would be necessary to run the inactive file. Hence the criterion for culling would be such that there was a low probability of striking even one of the inactive accounts on any day.

C26.4 Ordering the Data.

As the random access time on tape is quite prohibitive it is necessary to maintain each file in a given order, and to sort transactions to the same order prior to the updating run. This sort may be done off the computer if (i) the transactions are on punched cards, not paper-tape, and (ii) they are punched one per card.

Similarly the output will be in the same order as the file and may need to be sorted before printing or before input to another computer run. For example on a Raw Material Stores updating run, the files would doubtless be kept in part number order but the main output - priced requisitions - had best be printed in cost code order and/or input to the costing run in that order. Similarly the orders initiated on the run would be gathered up by supplier for issue. Once again if the output is on cards and is one item per card, a punched-card sorter can be used; otherwise a computer run is necessary.

C26.5 Number of Tape Decks.

Most of the time it will be possible to show operational savings by increasing the number of decks used and it is necessary to strike the correct balance between these savings and the capital cost involved. The minimum number is two, one for the Balance File and one for the updated Balance File. The ways in which additional decks may be used to bring operational savings are:-

(i) To separate standing details from the balance details, thus introducing a Standing File.

(ii) To create a supplementary Balance File as outlined in (3).

(iii) To permit input on tape instead of cards.

(iv) To permit output on a print tape and perhaps on a dump tape also.

(v) To allow the tape-operated printer to be run simultaneously with the computer.

(vi) To improve sorting times.

(vii) To allow tape reloading times to be absorbed. Often the main files will each consist of several reels of tape, and on each occasion that a reel is exhausted, it will take about two minutes to rewind and about one minute more to change. Thus if there are no spare decks, the computer will stand idle for about three minutes each time a change is necessary, and this loss of time is likely to become serious on any long run. If spare decks are available these can be loaded in advance of of the computer's requirements, and the machine will then have more data to work on while tape changes are taking place.

C26.6 Block Length.

On DEUCE the length of each block is variable - from block to block if desired - at the programmer's discretion. Sometimes it will be most convenient to use blocks of one or two D.L's each, so that all the work can be done in the high-speed store: normally to accomplish this a Mark II A will be necessary. The incidence of tape start-stop times may, however, be quite serious when such short blocks are used, and often the use of longer blocks will show time savings.

Here the technique will be to use the drum as a buffer store for the tape, using blocks of length up to 16 tracks. With such long blocks it will often prove advantageous to keep an index of its contents with each block: this enables the programme to determine at once the whereabouts of any record to be updated. Generally the objective will be to make the blocks as long as possible without incurring head shifts; to find the optimum length in any particular case requires calculations of the times involved and no general formula can be laid down. An illustration of the calculations in one case studied may be helpful.

Example 3.

On a banking current account run, it was required to update the balance file by transactions already sorted and on magnetic tape. Each transaction record contained an account number (6 decimal digits), other numerical matter (20 digits plus sign) and 6 alphabetical characters; the decimal information had, however, already been converted to binary so that each transaction occupied 4 words in DEUCE, and it was found best to have this arranged in blocks of length each 4 tracks, i.e. 32 transactions.

Each account record was composed of account number (6 digits), balance (10 digits plus sign), other numerical matter (48 digits) and name and address (70 alphabetical characters), and again the numerical information was held in binary, so that each account occupied 21 words; these were packed 17 to a block leaving room in the 12-track block for an index of 17 words.

There were 60,000 such accounts and 20,000 transactions, assumed to be distributed:-

1 on every 6th account, 2 on every 20th account, 10 on every 150th account.

A DEUCE Mk II with four tape decks was to be used.

The method was:-

The first block of the transactions tape was read into four tracks of the drum, and the first block of the accounts tape into the remaining twelve tracks of the same head position. The first track of each was brought into the high-speed store and the required account, determined from the transactions data, was found on the index to the accounts data and the appropriate track brought into the high-speed store. While this transfer proceeded (taking 16 ms) the index could be scanned for the next required account, and its transfer ordered as soon as the first was complete. Similarly during the second transfer the work of updating the first account could be done, and the transfer back to drum of the revised figures ordered as soon as the last transfer was completed. In this way, all computation time, except the first look-up of the index to any block, was absorbed in the transfer times. Now the theoretical time of the operation could be calculated.

Time for dealing with one block of the transaction tape:-

Read tape	103 m s
Transfer each track in turn, 4 x 16	64 m s
	167 m s

If the number of transactions is called y, then there were $\frac{y}{32}$ such blocks, and the time for dealing with them all $\frac{167y}{32}$ m s.

Let the number of accounts be x and the number of affected accounts be z. Time for dealing with one block of accounts tape:-

Read tape	282 m s
Transfer index to high-speed store	16 m s
Look up first affected account: find (on average) after	$\frac{z}{x}$ m s

Transfer (on average) $\frac{17z}{x}$ accounts affected to high-speed store	$\frac{16 \times 17z}{x}$ m s
Transfer to drum the updated accounts	$\frac{16 \times 17z}{x}$ m s
Write updated tape	294 m s
TOTAL	$592 + \frac{545z}{x}$ m s

There are $\frac{x}{17}$ such blocks, so that the time for dealing with all was $\frac{592\,x + 545z}{17}$ m s.

Rewind time for the last reel must be added: allow 2 minutes. As, at any time, only three of the four available decks were being used, the fourth could be being rewound and reloaded without loss of time; thus nothing needed to be added for rewinding any reel except the last.

The total time for this exercise then was:-

$\left(\frac{167y}{32} + \frac{592x + 545z}{17} \right)$ m s + 2 minutes, where there were x accounts, y transactions and z affected accounts. Putting x = 60,000, y = 20,000, z = 12,500 the formula gave 45 minutes approx.

If short blocks had been used, the calculation would have been:-

Let each block of the accounts tape consist of one account only, i.e. 11 word-pairs.

Let each block of the transactions tape consist of eight transactions, i.e. 16 word-pairs.

The time for reading the accounts tape would be 33 m s per account, and for writing the updated tape 45 m s per account, a total of 78 x m s for x accounts. That for reading the transactions tape would be 40 m s per block, or 5y m s for y transactions. The number of affected accounts is immaterial, and all computation could be done during the tape start-stop times, but the final rewind would again have to be added, so that the total time required for updating would become:-

(78x + 5y) m s + 2 minutes, or setting the values for this case as before, 82 minutes.

Notes on Example 3.

(i) The difference in time between the two methods (long and short blocks) is quite marked; this is because in this case there is very little computation, and the tape start-stop times become a major factor.

(ii) In looking up the index in this example, the programme simply started at the beginning and worked through till it found the item required; in this case as good a method as any. Had the strike density been very low, however, time could have been saved by the more sophisticated look-up technique: examine the middle item first, which will indicate which half the required item is in. Look next at the middle item in this half; and so on.

B. ILLUSTRATIVE EXAMPLE: BANKING APPLICATION.

C26.7 Specification.

This example is a demonstration current account routine: 100 accounts are held on magnetic tape and 100 transactions (not evenly spread) are fed for each day of a 3-day run.

(a) File data.

For each account is to be held name, account number, overdraft limit, report level, current balance, details of stopped cheques, and history of the account, i.e. details of all transactions since last statement and statistics relating to the current period of maximum, minimum and average balances, turnover and products (i.e. sum of each balance times no. of days held).

(b) Transaction data.

For each transaction is to be fed account number, narrative (last 3 digits of cheque no., or 3 alpha characters, e.g. CSH, DIV), and amount.

(c) Other input.

Provision is also required for the file details to be amended, accounts to be opened and closed, and statement requests to be made.

(d) The Daily Run.

It is required to read the input data, sort it to the same order as the file, update the balances, history and statistical details by the acceptable transactions while rejecting any stopped cheques or debits which would reduce a balance below its limit; at the same time to make any indicated amendments to the standing details and issue statements when requested. When a statement is issued, the transactions shown on it no longer need to be stored on the file.

(e) The Final Run.

After the three days' transactions have been processed, another run is required to print out all statements and the statistical details related to each account.

(f) General.

As many checks as reasonably necessary are built into the programme against punching error, machine and tape error.

26.8 Drawing up the Details.

(a) Input.

It was found that the normal transaction could be accommodated within 20 digits (= 4 DEUCE words) and could therefore be prepared four per card. The layout decided was:-

(i) Account No. - 7 digits - cols. 1 - 7. (The 7th is a check digit to ensure correct punching of the account number.)

(ii) Card Class - 1 digit - col. 8.

(iii) Narrative - 3 digits - cols. 9 - 11.

(iv) Amount - 9 digits - cols. 12 - 20. (Shillings and pence each being punched in one column only, so that amounts up to £9,999,999/19/11 are allowed).

Repeat for cols. 21 - 40, 41 - 60 and 61 - 80.

An X overpunching in col. 1 (or 21, 41 or 61) to indicate last transaction of the day's run.

"Statement request" or "close account" will simply occupy the first 8 columns (A/C No. plus Card Class), but provision must be made for more details to be shown on "open new account" or "amend standing details" so for these two classes a whole card is allocated.

The card class must be taken into account in the sorting run, so that where there are two or more inputs for the same account, they will be dealt with in the correct order. Therefore card class numbers are used as follows:-

1. New account.
2. Amendment.
3. Correction of Cr. transaction.
4. Cr. transaction.
5. Correction of Dr. transaction.

6. Dr. transaction: override limit.

7. Dr. transaction: normal.

8. Statement request.

9. Close account.

(b) File data.

This is also kept mainly in character form and arranged as follows:-

(i) Word-pair 1: Indicative information (no. of tracks occupied by this a/c, no. of stopped cheques, no. of entries in its history).

(ii) Word-pairs 2 - 4: Accountholder's name (up to 30 characters).

(iii) Word-pair 5: Account No. (7 digits).

(iv) Word-pair 6: Current balance (up to £$10^8 - 1^d$) and sign.

(v) Word-pair 7: Report level (up to £$10^8 - 1^d$) and sign.

(vi) Word-pair 8: Limit (up to £$10^8 - 1^d$) and sign.

(vii) Word-pair 9: Maximum balance (up to £$(10^{10} - 1)$) and sign.

(viii) Word-pair 10: Minimum balance (up to £$(10^{10} - 1)$) and sign.

(ix) Word-pair 11: Dr. Products (up to $(10^{10} - 1)$ £-days).

(x) Word-pair 12: Cr. Products (up to $(10^{10} - 1)$ £-days).

(xi) Word-pair 13: Dr. Turnover (up to £$(10^{10} - 1)$)

(xii) Word-pair 14: Cr. Turnover (up to £$(10^{10} - 1)$).

(xiii) Word-pairs 15 & 16: each details of a stopped cheque: last 3 digits of cheque no. and last 8 digits of amount, where known.

If there are more than two stops, a fresh track is allocated to the overflow, and the history then starts on the third track. Otherwise it starts on the second track. Each transaction recorded in the history occupies three word pairs:-

(xiv) Word-pair 1: Date (5 characters) and Narrative (3 characters).

(xv) Word-pair 2: Amount (up to £$10^8 - 1^d$) and sign.

(xvi) Word-pair 3: Balance (up to £$10^8 - 1^d$) and sign.

Thus up to ten transactions can be accommodated in each track. No provision is made for any account to occupy more than 16 tracks; each account appears as one block on the magnetic tape.

(c) Daily output.

First, a list of all accounts on the file, tabulated from punched cards put out by DEUCE. Each card corresponds to one account, and shows:-

(i) Cols. 1 - 30: name.

(ii) Cols. 31 - 37: account no.

(iii) Cols. 41 - 50: balance, with sign overpunched in col. 41.

(iv) Cols. 51 - 60: total of debit entries for day.

(v) Cols. 61 - 70: total of credit entries for day.

(vi) Cols. 71 - 80: the report level, if the balance on the account has fallen below that level.

Second, the list of rejected transactions: this shows the details of each transaction as input and the reason for rejection, which may be one of seven:-

(i) Closure requested on account whose balance is not nil.

(ii) Error in account no.: check digit does not agree.

(iii) No account held for this number.

(iv) Cheque is a stopped one.

(v) This transaction would take balance on account below its limit.

(vi) Request to open new account when an account exists for this number already.

(vii) Correction without corresponding transaction.

(d) Final output.

First, a statement for each account on the file. Each statement is tabulated from the following cards:-

(i) A heading card "STATEMENT".

(ii) Name, account no. and date.

(iii) Headings: Date, Narrative, Dr. Amount, Cr. Amount, Dr. Balance, Cr. Balance.

(iv) A card for each entry on the statement.

Second, a list of the statistics held for each account, as already shown on the file.

(e) The Programme.

The daily run divides into three parts:-

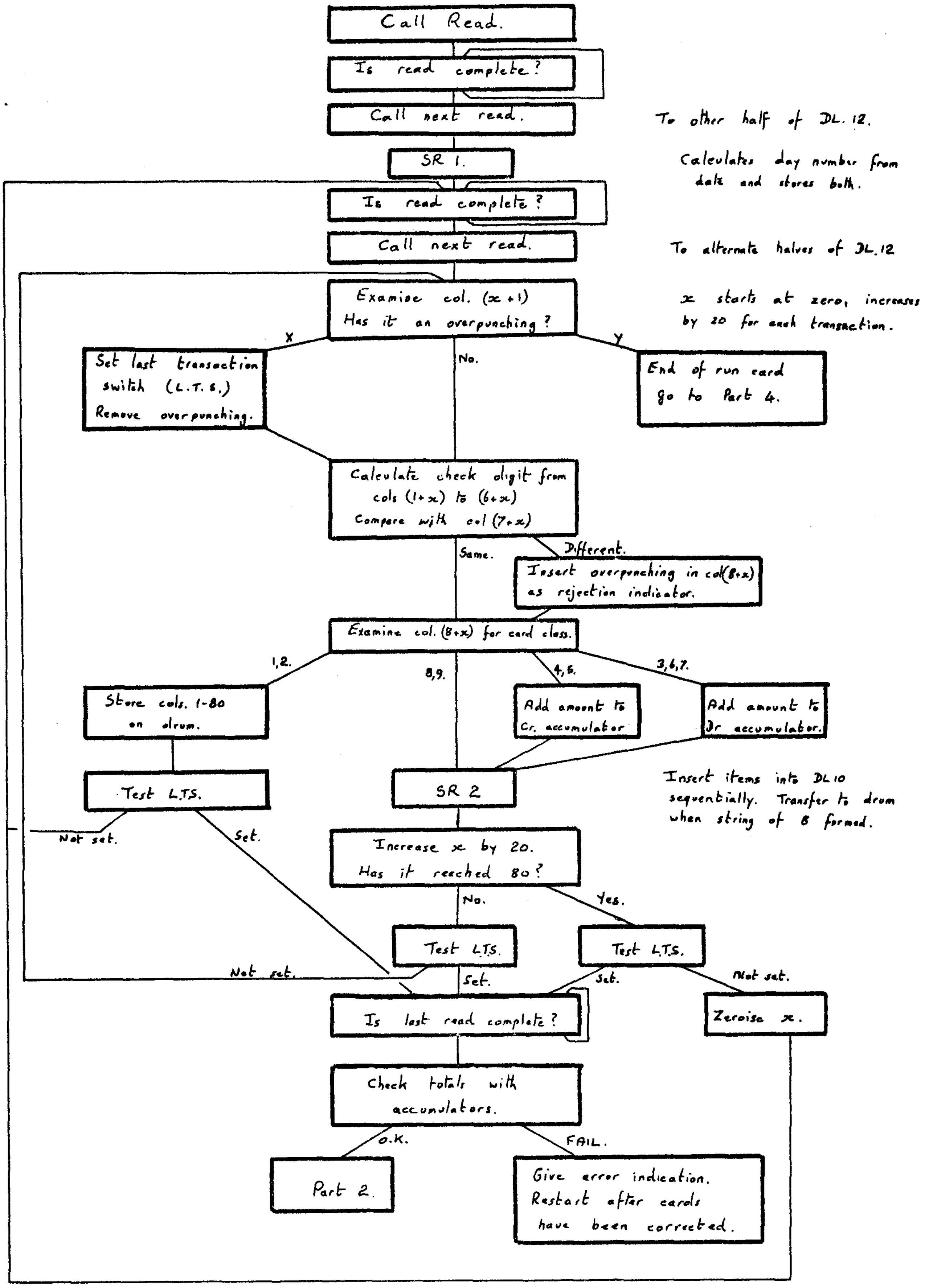
(i) Read transactions. Read the date card and calculate the date as a day number (this makes easier the subsequent calculations); read each transaction and check its account number; accumulate the amounts; partially sort the transactions, making strings of 8 before storing on the drum; read the total card and check that this agrees with the accumulations. Block flow diagram follows .

(ii) Drum sort of transactions.
This is done by a standard programme in the library.

(iii) Update File and Punch Balances.
Read each account; if it is an affected one determine the card class of transaction and take appropriate action on the account. When work (if any) on each account is finished, punch a card for the Balance List. Store details for any statements called for and of transactions rejected, punching cards for these classes at the end. Block flow diagram follows.

In addition, the final run produces statements and statistics, eliminating the history from the file.

PART 1 - READ TRANSACTIONS.


Call Read.
Is read complete ?
Call next read.
To other half of DL. 12.
SR 1.
Calculates day number from date and stores both.
Is read complete ?
Call next read.
To alternate halves of DL.12
Examine col. (x + 1)
Has it an overpunching ?
x starts at zero, increases by 20 for each transaction.
X
No.
Y
Set last transaction switch (L.T.S.)
Remove overpunching.
End of run card
go to Part 4.
Calculate check digit from cols (1+x) to (6+x)
Compare with col (7+x)
Same.
Different.
Insert overpunching in col(8+x) as rejection indicator.
Examine col. (8+x) for card class.
1,2.
8,9.
4,5.
3,6,7.
Store cols. 1-80 on drum.
Add amount to Cr. accumulator
Add amount to Dr accumulator.
Test L.T.S.
SR 2
Insert items into DL 10 sequentially. Transfer to drum when string of 8 formed.
Not set.
Set.
Increase x by 20.
Has it reached 80 ?
No.
Yes.
Test L.T.S.
Test L.T.S.
Not set.
Set.
Set.
Not set.
Is last read complete ?
Zeroise x.
Check totals with accumulators.
O.K.
FAIL.
Part 2.
Give error indication.
Restart after cards have been corrected.

PART 2 sorts the transactions into order on the drum.

PART 3 - UPDATE FILE.

The last entry on the Balance tape is a dummy having a key greater than any possible account number: this is the end-of-file marker. In the sort routine a similar dummy has been generated after the last transaction.

Tape 1 contains the Balance File, with stops and history.

Tape 2 is to contain the updated Balance File.

Tape 3 is to contain rejected transactions.

Tape 4 is to contain details for statements requested on the run.

The drum contains the transactions, and area A in the high-speed store is reserved for the transaction currently being operated on.

Area B (one head-position of the drum) contains the account being operated on.

Area C of the high-speed store is reserved for corrections.

Area D is reserved for the formation of output information until it is ready to punch.

LECTURE 27.

AN OUTLINE OF STAC.

27.1 INTRODUCTION.

Half the battle of solving a problem is finding a set of rules which, faithfully carried out, give the answer. This work is usually quite difficult, and is tackled with energy and enthusiasm by human brains of high calibre. Having achieved this much, the brains tend to be somewhat less interested in the other half of the battle, which consists of slogging out these rules to produce a result, and seek to delegate the work.

On this course, you have been learning how this work may be delegated to DEUCE, and, more than likely by now, you are dismayed to find that you have jumped from the frying pan of manual computation into the fire of programming! But you are at an advantage in that other people have jumped before you, and they have made good progress towards quenching the flames. Many devices now exist to alleviate the burden of programming; the simplest of these are the subroutines which save replanning and recoding operations of frequent occurrence such as finding a square root or sorting a string of numbers into ascending order of size, but a bolder type of programme actually throws back some of the drudgery onto the DEUCE itself.

27.2 THE STORAGE ALLOCATION AND CODING PROGRAMME.

The STAC programme (an abbreviation for STorage Allocation and Coding) is of the latter type, and it can be used to eliminate the more mechanical steps when writing a programme in the basic DEUCE instruction code. What you have to do is:-

(1) Describe the computational method by diagrams in which short sentences and/or algebraic expressions represents steps in the computation and arrows indicate the order in which the steps are to be performed, including repetitions and alternative paths which are to be decided from results of previous steps. (See Example 1).

(2) Divide these steps into smaller simpler steps each of which can be written as a single DEUCE instruction or an existing subroutine of instructions. This sequence of DEUCE instructions is called the Flow Diagram.

Each DEUCE instruction specifies a Source and a Destination. Either of these may be a computer address. However, it is not necessary to fix all these addresses while writing the flow diagram; in fact there are places where you may write symbolic addresses in the flow diagram which can later be allocated real addresses. (See Example 2).

(3) Punch this flow diagram, in decimal and with one instruction per card, using the layout specified for STAC input and hand over to the DEUCE and STAC.

What STAC and the DEUCE will do for you is:-

(1) Allocate real computer addresses to each symbolic address and to each instruction.

(2) Code the instructions i.e. write them in the form the DEUCE understands.

(3) Punch the binary instruction cards, and also alphanumeric cards that can be printed to give a flow diagram with instruction locations and a directory of symbolic addresses with their real address allocation.

27.3 PREPARING THE FLOW DIAGRAM.

This section, on preparing the flow diagram, is intended to give you an appreciation of the scope and limitations of STAC; it does not give a full account of all the facilities included in STAC, nor does it include operating details such as punching formats or pack assembly, since this is written out in the STAC Programming Manual (Ref.1).

27.4 OVERALL ORGANISATION.

Some of the facilities provided in the STAC scheme have been chosen to make the writing of a large programme as convenient as possible. However, the presence of these facilities does not exclude the possibility of doing small jobs such as coding a subroutine.

In dealing with a large programme, the flow diagram should be split into sections. These sections will be translated separately from STAC to DEUCE code and then should be assembled into a complete pack using some sort of control scheme.

A control scheme has been specially devised for STAC-produced sections which incorporate various programme testing facilities (Ref.2).

The minimum requirements a section must meet are:-

(1) All the instructions must fit into the high speed store (i.e. overwriting of blocks of instructions from cards or the drum during the execution of the section is not permissible).

(2) The available stores in D.L's 1 - 8 (i.e. the NIS D.'Ls) must be sufficient to accommodate all instructions without explicit computer addresses and all symbolic address (whether they are instructions or not).

From this it may be seen that division into sections is necessary to accommodate all the instructions. In fact, at the end of each section new instructions are fetched from cards or from the drum. Therefore, when splitting a flow diagram into sections, care should be taken to ensure that where possible divisions do not cut across inner loops.

27.5 A SIMPLE SEQUENCE OF INSTRUCTIONS.

The flow diagram should consist of sequences of instructions, which will normally be DEUCE instructions. Each DEUCE instruction should specify at least a Source and a Destination.

e.g. $1_{10} - 13$

$19_2 - 25$

$13 - 1_{10}$

The instruction sequence is normally specified by the order in which instructions are written down.

27.6 SYMBOLIC ADDRESSES.

NIS D.L. storage positions used to store instructions, numbers or patterns may be referred to symbolically. There are 100 symbolic addresses, written R_1, R_2, ..., R_{100}.

27.7 A SEQUENCE OF INSTRUCTIONS USING SYMBOLIC ADDRESSES.

The Source or Destination of an instruction (not both) may be a symbolic address

e.g. $R_{22} - 13$

$14 - 25$

$13 - R_{22}$

However, an instruction may not specify transfer between a D.L. store and a symbolic address.

e.g. $3_{27} - R_{19}$ is not allowed.

Note that the various requirements of a symbolic address must be compatible with its being allocated a D.L. address by STAC

e.g. $R_{18} - 19_2$ and later

$R_{18} - 19_3$ is not allowed.

27.8 LENGTH OF TRANSFER.

An instruction may also specify a length of transfer

e.g. $14 - 13$

$27 - 25$ (5 m.c's)

$13 - 14$

Provided neither Source nor Destination are symbolic addresses, the length of transfer may lie between 2 m.c's and 33 m.c's inclusive. Otherwise, only a transfer of length 2 m.c's is allowed. For example, $R_{16} - 18_1$ (2 m.c's) may be written and this means trans fer R_{16} to 18_1 and R_{17} to 18_2. R_{16} and R_{17} will be allocated real addresses in the same D.L. and in adjacent m.c's.

27.9 INSTRUCTION LOCATION AND NEXT INSTRUCTION.

Since a sequence is normally specified by the order in which instructions are written down, it is not normally necessary to refer explicitly to the addresses at which instructions are stored. However, there are circumstances in which explicit references to instruction addresses have to be made, namely

(a) the instruction is to be moved or involved in arithmetic operations, in which case all references to it must be by its address (symbolic or real) including its execution.

(b) the above sequencing rule is b roken

(c) storage is allocated manually.

To deal with these cases, an instruction may specify its instruction location and/or its next instruction. They can be either real or symbolic addresses.

27.10 JUMPS.

Unconditional or conditional jumps in a sequence of instructions may be made. An instruction which leads to an instruction which is not the next in the STAC pack will be called a jump instruction. An example of an unconditional jump is:-

$14 - 13$

$28 - 25$

$13 - 19_2$

.
.
.

$14 - 13$

$19_2 - 14$

An unconditional jump is specified by allocating an instruction location to the instruction to which the jump is to be made and put the next instruction of the jump instruction equal to this instruction location.

e.g. the above example might be written

R_{19} $14 - 13$

$28 - 25$

$13 - 19_2$

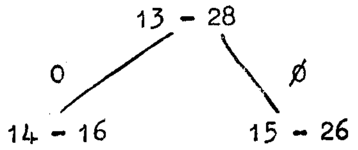
.
.
.

$14 - 13$

$19_2 - 14$ R_{19}

A conditional jump occurs when the jump instruction is a discriminative instruction, i.e. has destination 27 or 28 or is 2 - 24. Such an instruction has two possible next instructions, the natural next instruction and the unnatural next instruction (the natural next instruction being taken if the source contains zero in the case of destination 28, or a positive number in the case of destination 27, or if the TIL signal is not on in the case of 2 - 24).

e.g.



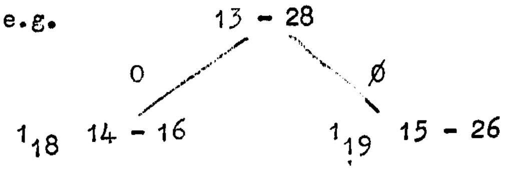
This is specified as follows:-

(1) Allocate an instruction location to the natural next instruction.

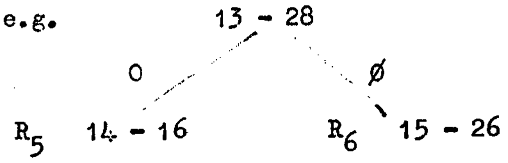
(2) Make the next instruction of the jump instruction equal to this instruction location.

(3) Then allocate the instruction location of the unnatural next instruction as follows:- if the instruction location of the natural next instruction is a real computer address, then allocate the next m.c. of the same D.L. If it is a sumbolic address, allocate the sumbolic address with subscript one greater.

e.g.



e.g.



27.11 MANUAL STORAGE ALLOCATION.

The programmer may wish to allocate real computer addresses to a sequence of instructions, for example to obtain optimum coding in a tight loop. These addresses are written as instruction locations.

Quasi.

An instruction may also specify a 'quasi' m.c. i.e. the minor cycle from which the instruction is obeyed if this is different from the one in which it is stored. It must be a real minor cycle number not symbolic. One of its uses if for instructions obeyed via destination 0.

e.g. 13 - 0

Q_{30} $(9_{11}$ 14 - $10_0)$

27.12 SUBROUTINES.

Library subroutines may be conveniently included in the flow diagram. To this end, each subroutine or each entry of a multiple entry subroutine in the library has been identified by a three-digit serial number. A subroutine also has a position which is defined as the number of the lowest D.L. occupied. Most subroutines are available in several positions.

To use a subroutine, the subroutine cards in the position required must be reproduced from the subroutine library and included in the STAC pack. The STAC instruction to include the subroutine in the flow diagram is written SXYZ/T where XYZ is the (3 digit) serial number of the subroutine (Ref. 1, Appendix 1) and T is the position of the subroutine. STAC automatically produces instructions which plant the link, enter the subroutine in the correct m.c. and obey the link from the correct quasi m.c. (the link is taken to be the instruction immediately following the subroutine instruction).

e.g. a possible sequence is

15 - 14

S146/5

$13 - 10_1$

If S146/5 were a subroutine of order 2 with link in 13 and entry in 5_{26}, this might get coded

4_{20} 15 - 14

4_{22} 4_{24} - 13

5_{26}

S146/5

Q_{31} (4_{24} $13 - 10_1$)

However, STAC does not plant parameters in subroutines and these must be dealt with manually.

27.13 ORDER OF PRIORITY.

The speed of the resulting programme can be affected by the storage allocation and is generally better when there is a bigger selection of unallocated addresses. A sequence of instructions may be given priority over other sequences of instructions, when instruction addresses are being allocated, by giving the first instruction in each sequence a priority number. (A sequence is taken to mean a sequence in the STAC card pack). Priority numbers range from 1 to 9, a small number indicating a high priority.

27.14 FACILITIES FOR THE MORE EXPERIENCED DEUCE PROGRAMMER.

Facilities are included which aid the experienced DEUCE programmer to perform arithmetical operations on instructions to generate new instructions.

27.15 FUTURE DEVELOPMENT.

At the time of writing, STAC is one of the newest additions to the DEUCE programme library and we have not had much chance of seeing it in action.

We have done our best to make the programme do all the manual claims it does, and we are hopeful that you will have no trouble on that score. However we are less sure that the manual describes the programme for storage allocation and coding that is most convenient for your application, as we were able to sound the opinions of a rather limited number of DEUCE users when drawing up the original design. We are already aware that it has various shortcomings, and we would welcome your criticisms and suggestions. Please address these to Miss A. Birchmore, E.E.Co. Ltd., London Computing Service, Marconi House, Strand, London W.C.2, who will attempt a second version if there is sufficient demand.

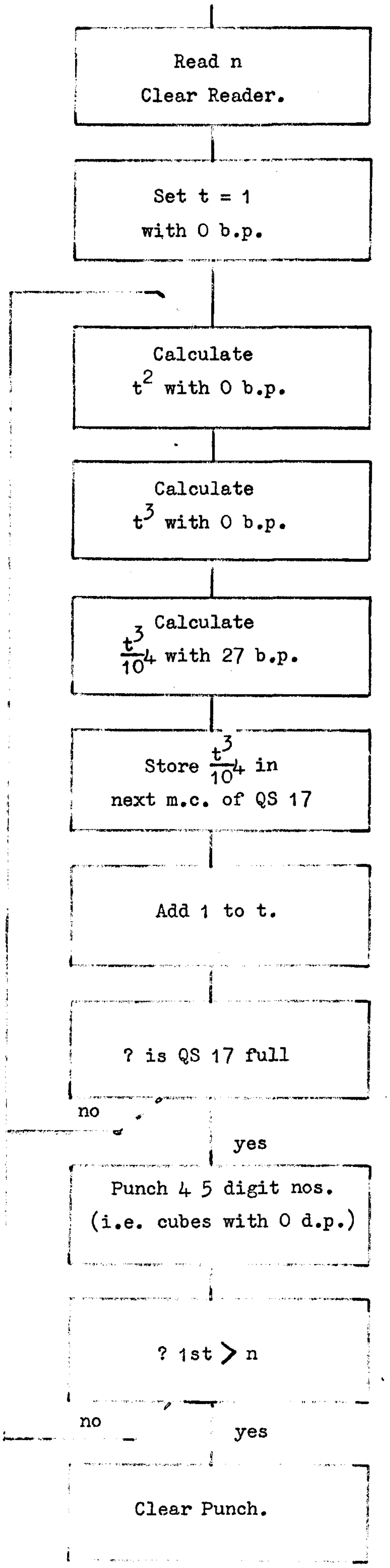
References.

(1) STAC Programming Manual. (DEUCE News No. 38).

(2) Control Programme for STAC-Produced Sections. (ZC20T, No. 483).

Example 1.

Logical Flow Diagram of a Programme to Calculate the Cubes of the First n Natural Numbers ($n \leq 44$).



Example 2.

STAC Programme to Calculate Cubes of First n Natural Numbers ($n \leq 44$).

1_{30} 0 - R_1 X

9 - 24

27 - 19_2

R_6 19_2 - 16

16 - 21_3

MULT

21_2 - 15

15 - 21_3

MULT

21_2 - 15

R_2 - 16 ($R_2 = 10^4\ 2^4$)

15 - 21_3

DIV

19_2 - 14

24 - 14 (16)

R_3 - 15 ($R_3 = 3P_{17}$)

R_4 - 13

25 - 25

21_2 - 16

13 - 0

Q_{30} (R_4) 16 - 17_3)

19_2 - 13

27 - 25

13 - 19_2

25 - 28

R_5 13 - 13

S507/2 (i.e. P04/1)

19_2 - 13

R_1 - 26

13 - 27

R_8 13 - 13 R_7 9 - 24

28

29

Extra D.L's on Deuce MK.IIA

The Deuce MK.IIA is fitted with an extra seven long DLs, available for either data or instruction storage. They are referred to as DL 1^A to 7^A, but require some addition to the normal order code to control their use.

Source and Destination.

The source and/or destination may either refer to the ordinary DL's 1 - 7 or to the extra DL's 1^A - 7^A. The distinction between the two is controlled solely by the P_1 digit of the instruction, irrespective of whether the instruction itself is obeyed from the ordinary DL's or the extra DL's

The rules are therefore:-

1) If either the source or the destination refers to the "A" delay lines, the instruction will have a P_1
2) If neither source or destination refers to the "A" delay lines, the instruction will not have a P_1
3) Only sources and destination in the range 1 - 7 are affected in this way by the P_1 digit.
4) Transfers from DL 1 - 7 to the "A" DLs or vice versa are not possible.

Next Instruction.

The decision on whether the next instruction comes from the ordinary DL's or from the 'A' DL's depends on the state of trigger circuit 'C' (TCC). If TCC is off, the next instruction source refers to the ordinary DL's. If TCC is on, the next instruction source refers to DL 1^A - 7^A or DL8.

The control instructions for TCC are:-

14 - 24 (s) Put TCC off.

14 - 24 (l) Put TCC on.

The next instruction source of the TCC switching instruction depends on the state of TCC after obeying the switching instruction,

i.e. 1, 14 - 24 (l) will take the next instruction from DL 1^A

1, 14 - 24 (s) will take the next instruction from DL 1

TCC does not affect NIS 8

TCC is automatically cleared by initial input.

Deuce Destination Triggers.

The following is a complete list of Deuce Destination Trigger as at 1st June, 1960. It should be noted that some only apply to particular types of Deuce, and use various parts of the instruction word in addition to the normal source. In view of this, any destination trigger instruction should be coded exactly as required with no extraneous digits at all, to avoid incorrect operation on other Deuce installations.

1) Control of Deuce and Peripheral Equipment.

0 - 24	Stim mult
1 - 24	Stim divide
2 - 24	TIL (64 or 80 col)
2 - 241	TIL (Paper Tape Punch)
3 - 24	Stim TCA
4 - 24	Clear TCB
5 - 24	Stim TCB
6 - 24	Clear alarm
7 - 24	Stim alarm
8 - 24	Clear OPS
8 - 241	Switch to β field for 64 col card punch.
9 - 24	Clear card read or punch or paper tape punch or graph plotter.
9 - 241	Clear paper tape read
10 - 24	Stim card punch for 64 col operation
10 - 241	Stim card punch for 80 col operation. Read store starts in m.c. (m + W + 2) of DL12 therefore punch store starts in m.c. (m + W + 18) of DL12.
11 - 24	Not used
12 - 24	Stim card reader for 64 col. operation
12 - 241	Stim card reader for 80 col. operation. Read store starts in m.c. (m + W + 2) of DL12 therefore punch store starts in m.c. (m + W + 18) of DL12.
13 - 24	Stim paper tape read
13 - 241	Stim paper tape punch or graph plotter.
14 - 24	Clear T.C.C.
14 - 241	Stim T.C.C.
15 - 24	Not used.

2) Control of Magnetic Tape Equipment.

16 - 24 23 - 24	Select tape deck N where N = S - 16
24 - 24	Transfer 11 characters/word pair from magnetic tape
24 - 241	Transfer 11 characters/word pair to magnetic tape
* P1,24 - 24	Group sequence transfer: Transfer 11 character/word pair from magnetic tape
* P1,24 - 241	Group sequence transfer: Transfer 11 character/word pair to magnetic tape
25 - 24	Transfer 10 character/word pair from magnetic tape
25 - 241	Transfer 10 character/word pair to magnetic tape
* P1,25 - 24	Group sequence transfer: Transfer 10 character/word pair from magnetic tape
* P1,25 - 241	Group sequence transfer: Transfer 10 character/word pair to magnetic tape
26 - 24	Stim Write record gap
27 - 24	Rewind
28 - 24	Skip forward one block
28 - 241	Skip forward to next record gap

29 - 24 Skip backward one block

29 - 241 Skip backward to next record gap

30 - 24 Test parity indicator

31 - 24 Clear write record gap, or interlock until outstanding magnetic tape function (except rewind) are completed. Previously called 'inactive'.

* The P22 - 25 digits in a group sequence transfer instruction indicate the number of word pairs to be transferred to or from the <u>first</u> D.L., commencing at the word pair indicated by (m + w + 2). There after, transfer will be for 16 word pairs. If P22 - 25 digits are all zero, the first transfer will be for 16 word pairs.

30

DEUCE PROGRAMMING PRACTICE.

INTRODUCTION.

The lectures given in the first week of the Programming Course have aimed at explaining the facilities which are available on DEUCE. The short examples after each lecture have been in the nature of five finger exercises designed to illustrate the use of each of the several functions. The next task is that of learning how to use the DEUCE to solve problems. Three programming examples have been chosen to give students some practical experience.

Any attempt to programme a problem should be started by asking and answering certain questions. In broad outline these are:-

(a) How will the data be presented?

(b) How will the results be put-out?

(c) Are subroutines required?

(d) Is this a single or mult-section programme?

Many other points of detail will need to be cleared up before detailed programming can begin and each example is accompanied by a questionnaire which illustratesthe kind of question an experienced programmer would formulate in his attempt at the problem.

Programming Example No. 1 has been largely completed in the preliminary stages. A logical block diagram is provided and the problem will be completed when the logical diagram is translated into DEUCE instructions. All students are requested to maintain the highest standard of neatness in preparing logical diagrams, flow diagrams and coding sheets.

DEUCE PROGRAMMING EXERCISE NO.1.

SPECIFICATION OF PROGRAMME.

Write a programme which calculates, stores and punches the cubes of the natural numbers from n_1 to n_2.

Limits of Data.

$$n_2 \leqslant 1024\ (2^{10})$$
$$n_2 - n_1 + 1 \leqslant 320$$

1. The programme should read in n_1 and n_2 in binary from the Y and X rows respectively of the one input card.

2. Test if $n_2 \leqslant 1024$.

3. Test if $n_2 - n_1 + 1 \leqslant 320$.

4. Calculate and store cubes from n_1^3 to n_2^3.

5. Punch out cubes in binary, 12 per card from n_1^3 to n_2^3.

6. Return to the beginning to repeat for different n_1, n_2 values.

Test Data.

(a) $n_1 = 1$
$n_2 = 3$

(b) $n_1 = 1$
$n_2 = 33$

(c) $n_1 = 1$
$n_2 = 320$

(d) $n_1 = 1$
$n_2 = 2''$ to check failure.

(e) $n_1 = 1$
$n_2 = 512$. to check failure.

(f) Any n_1, n_2.

Questionnaire.

1. Why is n_1 limited to 2^{10}?

Answer. $(2^{10})^3 = 2^{30}$ which is inside single length. 2^{10} is a convenient upper limit.

2. Why is $n_2 - n_1 + 1$ limited to 320?

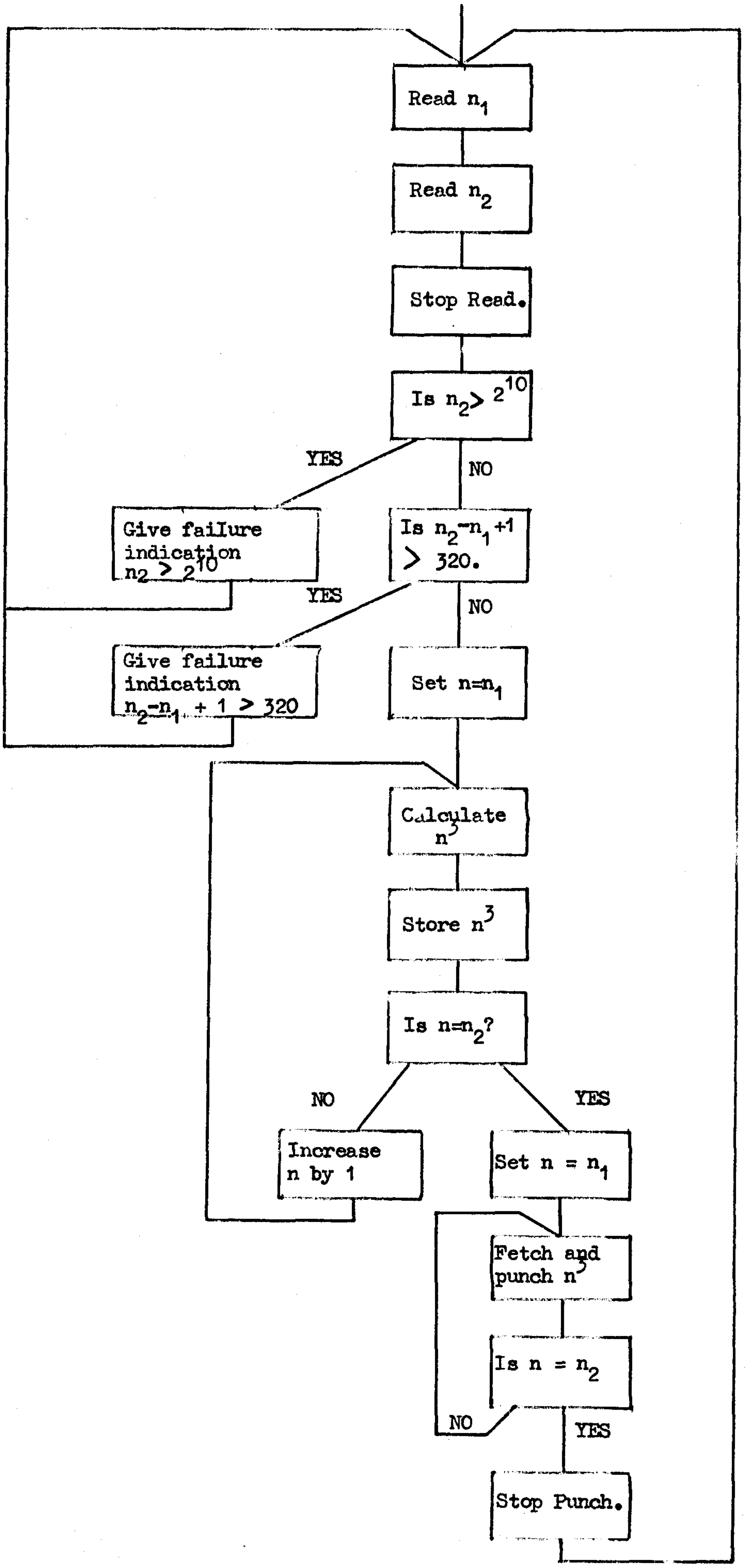
Answer. The number of cubes is $n_2 - n_1 + 1$. It is estimated that the programme will need 2 delay lines only leaving 10 delay lines for results i.e. 320 words.

3. Why are so many sets of test data recommended.

Answer.
(a) Checks that cubes are being formed and stored.
(b) Checks that storage extends correctly beyond one delay line.
(c) Checks that storage extends correctly through all delay lines.
(d) Checks the failure loop for $n_1 > 2^{10}$.
(e) Checks the failure loop for $n_2 - n_1 + 1 > 320$.

DEUCE PROGRAMMING EXERCISE NO.1.

LOGICAL FLOW DIAGRAM.



DEUCE PROGRAMMING EXERCISE NO.2.

STATEMENT OF PROBLEM.

A sum of money £C is borrowed at p% per annum compound interest, to be repaid over a period of n years in equal annual instalments. What is the amount of the annual repayment, expressed in pounds, shillings and pence rounded up to the nearest penny above?

Formula to be Used.

The annual repayment £x is given by the formula:-

$$x = \frac{\frac{p}{100}}{1 - \alpha^n} \quad \text{where } \alpha = \frac{1}{1 + p/100}$$

Limits of Data.

The programme should be valid for any combination using the following ranges:-

$0 \leqslant C \leqslant 10{,}000$ to nearest integer.
$1 \leqslant p \leqslant 10$ to three decimal places.
$1 \leqslant n \leqslant 40$ to nearest integer.

Specification of Problem.

Write a DEUCE programme which:-

(a) Reads the 3 values C, p and n from a single card punched in decimal, using subroutine R03.

(b) Calculates the repayment x, using fixed point single length arithmetic, without using multiplication or division subroutines.

(c) Punches the result in decimal, using subroutine P03.

(d) Returns to step (a) to read another card, so that any number of cases may be computed without reading in the whole programme again.

The following questions and instructions are typical of those which a competent programmer will ask and follow himself when planning the method of attack on the problem.

All students are expected to answer the questions and following the directions.

Questionnaire.

1. Do you understand the problem as specified in the problem statement?

2. What is the significance of data limits?

3. What is the significance of the 3 decimal places in the specification of p?

4. What will be the representation of p in DEUCE after it is read from a card and converted to binary by subroutine R03?

5. What is the maximum value of p?

6. What is the maximum value of $^p/100$?

7. What is the binary value of $(^p/100)_{max.}$?

8. What is the maximum number of binary places permissible for $(^p/100)_{max.}$ when held as a single-length signed DEUCE number?

9. What is the maximum value of α ? What value of p determines this?

10. If it is decided to form α by first forming $1 + {}^p/100$, how many binary places are permitted? With this number of binary places, what is 1 (in the expression $1 + {}^p/100$) as a DEUCE WORD?

11. What other method can be used to form α ?

12. How many binary places can α have as a single-length signed number?

13. To round off α correctly how many places should be obtained:-

(a) before round off?

(b) after round off?

14. If α is calculated by dividing 1 by $1 + {}^{p}/100$ what value must the 1 in the numerator have to obtain 31 b.p. from the divider assuming $1 + {}^{p}/100$ has 30 b.p.

15. Is α^{n} ever greater than 1?

16. How many binary places are required for α^{n}?

17. How will α^{n} be formed in the programme?

18. Is $\dfrac{\frac{p}{100}}{1-\alpha^{n}}$ ever greater than one? What value of n maximises $\dfrac{p/100}{1-\alpha^{n}}$?

19. How many b.p. are suitable for $\dfrac{p/100}{1-\alpha^{n}}$

20. Why is it preferable to form $\dfrac{p/100}{1-\alpha^{n}}$ and subsequently multiply by C instead of forming CP/100 and dividing by $1-\alpha^{n}$?

21. What is implied by the requirement that x is rounded upwards?

22. What units does x represent after the calculation $\dfrac{C\,P/100}{1-\alpha^{n}}$?

23. How can x be converted to £. s. d.?

24. What will be the maximum number of figures in any of the £. s. d. output figures.

25. Do you fully understand all the information on R03?

26. Do you fully understand all the information on P03?

Instruction 1.

After all the above questions have been answered by a student to a tutor's satisfaction he may proceed to prepare a LOGICAL FLOW DIAGRAM of the programme. No student will be permitted to write DEUCE instructions until a LOGICAL FLOW DIAGRAM has been approved by a tutor.

Instruction 2.

After a students logical flow diagram is approved he should prepare a DEUCE flow diagram of instructions. This must be annotated to show binary places used in calculations and to indicate partial result variables.

When an instruction flow diagram has been prepared and checked by a tutor it will be punched and coded by N.R.L. coding assistants.

While a programme is being coded and punched students should prepare test data figures.

Instruction 3.

For test data C = 10000, n = 1, p = 10 prepare binary patterns corresponding to those used at different stages of your programme for

$^{p}/100$

$1 + {}^{p}/100$ (if used)

α

$\dfrac{^{p}/100}{1-\alpha^{n}}$

Integral part of $\dfrac{C^{p}/100}{1-\alpha^{n}}$

Make sure you know the TS, DS or D.L. positions of each of these before you go on the 1st machine run.

DEUCE PROGRAMMING EXERCISE NO.3.

STATEMENT OF PROBLEM.

It is required to construct a table of values of the annual repayment £x for £1 of capital borrowed at p% for a period of n years. Entries are required in the table at intervals of δ n years between the limits n_1 and n_2, and δ p% between the limits of p_1 and p_2.

Formulation of Problem.

For an interest rate of p% and a term of n years, the repayment £x on £1 of borrowed capital is given by:-

$$x = \frac{\frac{p}{100}}{1 - \alpha^n} \quad \text{where} \quad \alpha = \frac{1}{1 + {}^p/100}$$

Limits of Problem.

$$\begin{array}{lll} p_1 \geqslant 1.000 & \qquad & n_1 \geqslant 1 \\ p \geqslant 0.001 & & n \geqslant 1 \\ p_2 \leqslant 10.000 & & n_2 \leqslant 40 \end{array}$$

Specification of Problem.

Write a programme to:-

(a) Read p_1, δp, p_2 in decimal from one card.

(b) Read n_1, δn, n_2 in decimal from one card.

(c) Calculate x for all combinations of n and p within the limits defined by the data.

(d) Store the values of x continuously on the magnetic drum, separating each set of entries having the same value of p from those having a different value of p by a suitable marker. (i.e. ${}^p/100$ to the same number of binary places as the values of x).

(e) A failure indication should be given if the capacity of the magnetic drum is exceeded.

(f) When all values of x have been stored, the read should be called, 1_0 cleared, and the next instruction taken from 1_0 to enable a second programme to be read in to puch out the results. Students are <u>NOT</u> asked to programme this punch routine.

Testing of Programme.

The following values should be used for testing the programme:-

Test 1 (to check calculation).

$$p_1 = 1, \ \delta p = 9, \ \ p_2 = 10$$
$$n_1 = 1, \ \delta n = 1, \ \ n_2 = 10$$

Test 2. (to check storage) $p_1 = 1$, $\delta p = .125$, $p_2 = 10$

$$n_1 = 1, \ \delta n = 1, \ \ n_2 = 40$$

DEUCE PROGRAMMING EXERCISE NO.4

1. STATEMENT OF PROBLEM.

A manufacturing and retailing organisation uses a computer for centralised stock-taking. Each branch is supplied from one of three warehouses and sells goods which are classified in three main groups, WET, DRY and BOUGHT OUT. Stock totals are required for all goods of each type from each warehouse. In addition, notification of excess stock value is required for one item in the BOUGHT OUT category.

2. DATA DESCRIPTIONS.

(a) Branch Stock Card. Use: INPUT

Card Col.	Remarks.	
1 - 15	Branch Name.	

16	Warehouse Code.	A = Kidderminster
		B = Reading
		C = Doncaster

17	Goods Code.	J = WET
		K = DRY
		L = BOUGHT OUT.

21 - 22	Catalogue Number.	
26 - 30	Quantity	
31 - 32	Catalogue Number	
36 - 40	Quantity	
41 - 42	Catalogue Number	
46 - 50	Quantity	
51 - 52	Catalogue Number	
56 - 60	Quantity	
61 - 62	Catalogue Number	
66 - 70	Quantity	
71 - 72	Catalogue Number	
76 - 80	Quantity.	

Notes:- (i) All quantities are five decimal digits, fully punched.

(ii) Catalogue codes are a numeric character preceded by the goods code character. All catalogue numbers on one card have the same first character. For example, on a J card (col. 17) the catalogue numbers are:

Col	21 - 22	J1
Col	31 - 32	J2
Col	41 - 42	J3
Col	51 - 52	J4
Col	61 - 62	J5
Col	71 - 72	J6

Catalogue numbers are not required in the programme.

(b) Description Card. Use INPUT.

Output cards bearing the totals for each warehouse and goods type are required to carry the appropriate code characters in Cols. 16 and 17 and the warehouse name in Cols. 1 - 15. These descriptions must be read into the machine on description cards which are identified by a "D" punched in Col. 20.

Card Col.	Remarks.
1 - 15	Warehouse Name
16	Warehouse Code
17	Goods Code
18 - 19	Blank
20	"D"
21 - 80	Zeroes

Note:- There are nine description cards.

(c) End Card. Use INPUT

To indicate the end of input data an "END CARD" is recognised by the code letter "Z" in Column 20. This END CARD is also used to place the description GRAND TOTAL in the delay line used for check totals (see later).

Card Col.	Remarks.
1 - 15	GRAND TOTAL
16 - 19	Blank
20	"Z"
21 - 80	Blank

(d) Warehouse/Goods Totals. Use. OUTPUT.

At the end of processing, nine warehouse/goods totals are required.

Card Column.	Remarks.
1 - 15	Warehouse Name
16	Warehouse Code
17	Goods Code
18 - 19	Blank
20	"D"
21 - 30	Total of all input data Cols. 26 - 30 from input cards with same warehouse and goods codes.
31 - 40	Totals of relevant input data Cols. 36 - 40
41 - 50	Totals of relevant input data Cols. 46 - 50
51 - 60	Totals of relevant input data Cols. 56 - 60
61 - 70	Totals of relevant input data Cols. 66 - 70

Card Column.	Remarks.
71 - 80	Totals of relevant input data Cols. 76 - 80

Note:- It will be required to read these cards in as descriptor cards if a restart is necessary.

(e) Grand Total Card.

A card with six totals of all corresponding input data from all cards

Card Col.	Remarks.
1 - 15	GRAND TOTAL
18 - 20	Blank
21 - 30	Total of all input card Cols. 26 - 30
31 - 40	Total of all input card Cols. 36 - 40
41 - 50	Total of all input card Cols. 46 - 50
51 - 60	Total of all input card Cols. 56 - 60
61 - 70	Total of all input card Cols. 66 - 70
71 - 80	Total of all input card Cols. 76 - 80

(f) Error Cards. USE. OUTPUT.

Any Branch Stock Card with a code other than A. B. or C in Column 16 or bearing a code other than J. K. or L in Column 17 must be treated as faulty. The erroneous code must be replaced by an X in the appropiate column and the card punched out with all other columns the same as the input card.
Card Format. Same as Branch Stock Card except in either Col. 16 or Col. 17 which should bear an X code if the corresponding input column does not bear a valid code.

(g) Excess Stock Card.

Certain branches are overstocking items L6 (card columns 76 - 80). Notification is required if the stock value of these items exceeds £200. Stock evaluation may be taken at 10/- per 100 items.

Card Format. Card layout is the same as for Branch Stock Cards with the addition of a code letter E in Column 20.

3. PROGRAMME SPECIFICATION.

Write a programme which reads all input data in the 80 Column mode and forms the sums of all corresponding quantities on cards of similar type as follows:

W/H Code	Goods Code	Sums formed on Track.
A	J	0/0
A	K	0/1
A	L	0/2
B	J	1/0
B	K	1/1
B	L	1/2
C	J	2/0

W/H Code.	Goods Code .	Sums formed on Track.
C	K	2/1
C	L	2/2

In addition sums of corresponding columns from all cards are required in D.L.1A.

Card Columns 1 - 20 of descriptor cards are to be placed in the corresponding minor cycles of the sum tracks designated by the codes in Columns 16 and 17.

Note. All cards must be checked for valid codes in Columns 16 and 17 and in the event of an error the appropriate column must be changed to an X code and an error card punched out. The stock value of quantities for items L6 are to be compared with £200. Any branch overstocking must be indicated by an output card with all branch details as on Branch Stock Cards and a code letter E in Column 20. The cost of L6 stock is to be taken as 10/- per 100.

When input card processing is complete - indicated by an END CARD - the title from the END CARD (Cols. 1 - 20) must be placed in the appropriate minor cycles of D.L.1A (after the Z designator has been replaced by a blank column) and the Grand Total and all warehouse/goods totals are to be punched out.

4. SUBROUTINES.

All addition is to be performed in characters, i.e. using a decimal addition subroutine No C 03. Full descriptions and instructions for use are provided.

To multiply the quantities L6 by 1.2 pence it will be necessary to convert the quantities from character form to binary. To do this use subroutine C15; details are provided.

5. STORAGE ALLOCATION AND PROGRAMME METHOD.

(a) Use D.L. 10 as the buffer store for input card data while the next card is being read into D.L. 12.

(b) Use D.L. 9 as the auxiliary D.L. in which to form the sums of quantities before transfer (via D.L.11) to the drum or to D.L. 1A.

(c) Error Cards and Excess Notification Cards may be punched while another input card is being read, using the DUAL READ-PUNCH facility of the 80 column machine.

(d) All programmes must start:

```
12 - 24  1
 2 - 24  nz
z ↓
12 - 10  (16 m.c.)
12 - 24  1
```

6. PROGRAMMING PROCEDURE.

1. Make out 80 column storage allocation sheets for each type of input and output card.

2. Prepare a block schematic diagram of the problem.

3. Get the above work checked by a tutor before proceeding.

4. Prepare a DEUCE flow diagram from the block diagram.

5. Code the programme.

6. In return for a similar service get your colleague to check the coding thoroughly.

7. Submit the coding sheets for punching.

8. Whilst the cards are being punched, prepare a plan of campaign for testing the programme on DEUCE.

Programme Construction and Testing.

Before any attempt is made to draw the block diagram for this programme or to write the DEUCE flow diagram students should consider very carefully the following questions.

1. Have you read the Problem Specification?
2. Do you understand the requirements of the problem?
3. How is a Descriptor Card recognised?
4. Do you know the character value of "D" in DEUCE code?
5. Do you know where to find the value of "D"?
6. How is an END card recognised?
7. What is the value of Z in DEUCE code?
8. How is a Branch Stock Card recognised?
9. What information exists in Cols. 16 and 17 of Descriptor Cards?
10. Is card order essential for descriptor cards?
11. What is the function of the codes in Cols. 16 and 17
    (a) in the problem generally?
    (b) in the programme?
12. If the code in Col. 16 is not A or B or C what action is to be taken?
13. If the code in Col. 17 is not J, K or L what action is requested?
14. Do you know the value of an X in DEUCE code?
15. Is the same amount of processing required for all types of Branch Stock Card?
16. Which type of Branch Stock Card require extra processing?
17. What extra processing is required for Type L cards?
18. What calculation will be necessary for the extra processing?
19. Are constants required in the calculation?
20. Do you know what these constants are (a) in decimal?
    (b) in binary?
21. Do you know how to calculate the binary constants?
22. Have you prepared 80 col. storage allocation sheets for all types of card?
23. What purpose do you think is served by making out such sheets?
24. If the programme is constructed to calculate and process one card while reading the next card is there a starting problem?
25. Until one card is completely read can any processing begin?

26. How does the machine know when a card cycle is complete?

27. How does a programme interrogate the reader (or punch) to find if a card cycle is complete?

28. If there are 2000 Branch Stock Cards how long should the programme take to run assuming 10% error and excess cards?

29. If there are 2000 Input Cards and no error or excess cards how long is a processing run?

30. If all 2000 Input Cards are errors or excess cases how long is a processing run?

31. Is there any information on an end card other that Z in Column 20?

32. Do you know how to use the extra information?

33. Do you know how to change a Z to a blank column?

34. Do you know why a restart facility is necessary?

35. Do you understand how to restart?

Programme Testing.

It is essential for programmers to acquire efficiency in testing their own programmes. This does not come all at once and entails a fair degree of pre-machine preparation.

A programme is not correct if it fails to pass through all the intended instructions in the correct sequence. A programme is not necessarily correct even when it does pass through all the instructions. Some of the instructions may not do what the programmer intended and some essential steps may have been omitted.

To check programme continuity we use PROGRAMME DISPLAY and for correct operation we can use a combination of test data runs and POST MORTEM.

Programme Testing Exercise No. 4.

The following paths must be checked

(a) The read-in path

(b) The A code path

(c) The B code path

(d) The C code path

(e) The three paths for J, K and L.

(f) The error code path

(g) The D code path

(h) The main addition path

(j) The excess stock path

(k) The punch out path

To test (a) to (g) omitting (f) we can use programme display. If we read a card into the computer, allow another card to be read into DL12 and then stop the machine we can cause the programme to be executed and punched out instruction by instruction and take away a record for inspection off the computer.

Instructions for Programme Display.

(a) Place a stopper on 12 - 10 following 2 - 24

(b) Initial Input programme followed by all descriptor cards

(c) Machine stops on 12 - 10X

(d) Set request stop on the instruction after 12 - 241

(e) Set Augmented Stop and release Request Stop and Programme Display.

(f) Machine stops on 12 - 10X

(g) Repeat steps (d) (e) and (f) until all descriptor cards are finished.

(h) Post Mortem

This is sufficient for the first run on the machine. When all errors are corrected repeat the procedure to check correction of errors and also include a Branch Stock Card. These tests will record more than 90% of all instructions.

The next checks can be made to ensure that additions are being performed correctly and to obtain sample error cards and finally punch out the totals.

Special test data has been provided. It will be noticed that the numbers on the test cards are easily recognisable patterns and these are intentional - not accidental.

TEST DATA

Descriptor Cards.

KIDDERMINSTER	A J D	00000	00000	00000	00000	00000	00000
KIDDERMINSTER	A K D	00000	00000	00000	00000	00000	00000
KIDDERMINSTER	A L D	00000	00000	00000	00000	00000	00000
READING	B J D	00000	00000	00000	00000	00000	00000
READING	B K D	00000	00000	00000	00000	00000	00000
READING	B L D	00000	00000	00000	00000	00000	00000
DONCASTER	C J D	00000	00000	00000	00000	00000	00000
DONCASTER	C K D	00000	00000	00000	00000	00000	00000
DONCASTER	C K D	00000	00000	00000	00000	00000	00000
GRAND TOTAL	Z						

Test Cards.

12345678988888	A J	11111	22222	33333	44444	55555	11111
12345678988888	A K	11111	22222	33333	44444	55555	11111
12345678988888	A L	11111	22222	33333	44444	55555	11111
12345678988888	B J	11111	22222	33333	44444	55555	11111
12345678988888	B K	11111	22222	33333	44444	55555	11111
12345678988888	B L	11111	22222	33333	44444	55555	11111
12345678988888	C J	11111	22222	33333	44444	55555	11111
12345678988888	C K	11111	22222	33333	44444	55555	11111
12345678988888	C L	11111	22222	33333	44444	55555	11111
		99999	199998	299997	399996	499995	99999

Excess Test Cards.

12345678988888	A L	00000	00000	00000	00000	00000	40000
1234567898888	C L	00000	00000	00000	00000	00000	39999
12345678988888	B L	00000	00000	00000	00000	00000	39999
12345678988888	A L	00000	00000	00000	00000	00000	39999
12345678988888	C L	00000	00000	00000	00000	00000	40000
12345678988888	B L	00000	00000	00000	00000	00000	40000

Error Test Cards.

RAMSGATE .	B B	K1	47199	K2	17789	K3	19947	K4	18808	K5	41009	K6	10094
CHELMSFORD	L L	L1	49521	L2	25125	L3	21003	L4	41126	L5	11264	L6	39109
CAERNARVON	M J	J1	10090	J2	09989	J3	16315	J4	19432	J5	54132	J6	13421
WARWICK	A A	J1	17941	J2	29726	J3	19876	J4	37422	J5	23461	J6	08436
CARDIGAN	N K	K1	50102	K2	31114	K3	19633	K4	49110	K5	44932	K6	34119
LEAMINGTON	A B	K1	59014	K2	30122	K3	30721	K4	43002	K5	45591	K6	49660
HEREFORD	S L	L1	48663	L2	46002	L3	41011	L4	10498	L5	49441	L6	39456
PLYMOUTH	J J	J1	41563	J2	18359	J3	33789	J4	33459	J5	37541	J6	17883
GLOUCESTER	A F	L1	50009	L2	41036	L3	33331	L4	39993	L5	44193	L6	19145
EASTBOURNE	B B	J1	24142	J2	23945	J3	35491	J4	17824	J5	29579	J6	27489
PORTSMOUTH	K K	K1	41099	K2	36120	K3	19904	K4	13050	K5	50054	K6	40550
RAMSGATE	B B	K1	47199	K2	17789	K3	19947	K4	18808	K5	41009	K6	10094
CHELMSFORD	L L	L1	49521	L2	25125	L3	21003	L4	41126	L5	11264	L6	39109
ROCHESTER	B B	L1	38533	L2	10059	L3	12952	L4	27651	L5	25167	L6	32100
LEICESTER	J J	J1	43696	J2	17302	J3	38745	J4	06300	J5	46302	J6	19119
NEWCASTLE - O - T	C C	J1	45660	J2	15944	J3	36021	J4	21009	J5	39191	J6	29092
NOTTINGHAM	Z K	K1	45556	K2	21772	K3	10975	K4	10635	K5	29703	K6	50006
GATESHEAD	C Z	K1	27716	K2	45615	K3	42295	K4	32105	K5	46635	K6	49715
STOKE - ON - TRENT	Q L	L1	47543	L2	35892	L3	30952	L4	25983	L5	29123	L6	12602
MIDDLESBOROUGH	C Q	L1	43746	L2	18935	L3	37464	L4	19358	L5	39553	L6	17734
			828513		517760		541375		526699		739144		558933

Branch Stock Cards.

WORCESTER	A J	J1	12745	J2	1734	J3	40606	J4	12707	J5	40712	J6	24361
SHREWSBURY			16437		25621		31276		21202		20741		30461
BARMOUTH			11003		74542		56543		32339		30297		52642
HEREFORD			14927		13126		10210		15741		41123		40741
TEWKSBURY			9374		50013		13198		18547		50123		12036
CARDIGAN			12121		42172		18742		40120		12471		42712
CAERNARVON			10090		9989		16315		19432		54132		13421
PWLLHELI			19243		10010		10027		17415		20200		9760
WREXHAM			15741		18547		41111		12472		21309		8231
BIRMINGHAM			12707		21202		32339		31561		31316		12742
STAFFORD			11333		28765		25073		20101		41745		40741
TENBY			14726		40777		42054		27419		17455		32172
WEST. BROMWICH			15438		30943		27341		54728		16234		10221
WARWICK			17941		29726		19876		37422		23461		8436
LEAMINGTON			14576		54731		54001		25433		33333		7321
GLOUCESTER			742		27310		10725		33765		12345		19877
STRATFORD			4761		18726		20403		34512		23456		40312
RUGBY			25789		54321		21297		43217		34567		12304
COVENTRY			18888		30010		34726		50743		45678		21340
WALSALL			20935		22291		21802		20198		19089		13240
			279517		604556		547665		569074		589787		453071

DPCS2

Branch Stock Cards.

WORCESTER	A K	K1	23691	K2	11623	K3	41474	K4	46701	K5	47301	K6	50888
SHREWSBURY			47234		47012		32097		41009		12345		10107
BARMOUTH			41032		36091		50001		39113		35241		39802
HEREFORD			16732		43906		49302		41107		40094		40109
TEWKSBURY			31032		20126		40631		50019		22391		44013
CARDIGAN			50102		31114		19633		49110		44932		34119
CAERNARVON			10777		35281		12494		10137		50922		50001
PWLLHELI			32100		49599		50029		16179		41394		46351
WREXHAM			11725		51329		47898		47961		23992		40111
BIRMINGHAM			54321		44304		10018		37099		11498		25551
STAFFORD			32098		45906		44478		19999		37332		40396
TENBY			50595		23101		24711		46113		10431		49991
WEST. BROMWICH			10091		11101		44900		11346		43101		10119
WARWICK			44377		19333		10666		34611		45506		50101
LEAMINGTON			59014		30122		30721		43002		45591		49660
GLOUCESTER			40732		22322		10079		39003		41999		41332
STRATFORD			32159		16671		47988		41094		33595		19656
RUGBY			29012		30009		32109		50096		23101		50009
COVENTRY			10962		20883		46591		49865		44379		42391
WALSALL			30743		34361		43333		30149		10109		44999
			658529		624194		689153		743713		665254		779706

Branch Stock Cards.

WORCESTER	A L	L1	45630	L2	10963	L3	36408	L4	46321	L5	10664	L6	79591
SHREWSBURY			39019		49323		41991		49480		30496		39001
BARMOUTH			45905		12543		25002		23075		10966		33339
HEREFORD			48663		46002		41011		10498		49441		39456
TEWKSBURY			10447		31006		20914		50000		39555		10105
CARDIGAN			50012		29006		22544		36591		15069		90795
CAERNARVON			49634		39712		39355		49000		19099		11432
PWLLHELI			41019		49101		35942		32098		41019		20119
WREXHAM			31055		10098		41092		32117		10194		67359
BIRMINGHAM			10965		46669		22100		43109		39150		10336
STAFFORD			39999		41991		21998		41008		44225		29321
TENBY			41077		37399		13980		37999		21793		28693
WEST. BROMWICH			47988		40001		49310		40412		19914		99561
WARWICK			37911		30000		31049		10873		29974		39999
LEAMINGTON			21100		47992		49002		41114		49623		15194
GLOUCESTER			50009		41036		33331		39993		44193		19145
STRATFORD			48110		50014		29009		60019		19563		28665
RUGBY			41099		32694		19080		19843		19606		34005
COVENTRY			41013		49391		41009		20478		35125		21256
WALSALL			10936		39144		24066		44012		28119		27741
			751591		734085		638193		728040		577788		745113

DPCS2

Branch Stock Cards.

OXFORD	B J	J1	38041	J2	47852	J3	21934	J4	12786	J5	27485	J6	48549
SALISBURY			46370		12534		37596		19592		29594		25942
DORCHESTER			31452		47548		15983		43659		17761		37802
SOUTHAMPTON			27567		26782		25798		24698		19567		41769
PORTSMOUTH			19238		33952		41543		19274		27891		28541
PLYMOUTH			41563		18359		33789		33459		37541		17883
CHATHAM			24739		21986		25351		13987		46782		32789
BOURNEMOUTH			25297		39752		29481		49012		39760		19789
WESTON.UPON-SEA			38473		16534		57781		37301		20917		39321
CAMBRIDGE			47541		44678		15762		29591		27819		47929
CHELMSFORD			29786		29782		18543		17908		39891		37417
ROCHESTER			17432		31894		27583		34909		45591		24919
TUNBRIDGE - WELLS			36598		16782		43109		21578		32784		16584
HASTINGS			42731		36452		28081		33892		19861		48764
EASTBOURNE			24142		23945		35491		17824		29579		27489
RAMSGATE			19876		17451		27872		32594		31492		15972
CANTERBURY			45251		27789		31458		25869		27568		34851
NORTHAMPTON			29891		38487		29561		17898		14591		18706
MAIDENHEAD			19891		24682		37492		47582		17039		24894
KINGSTON			24768		39487		27569		21659		47079		41764
			630647		596728		611777		555072		600592		631674

DPCS2

Branch Stock Cards.

OXFORD	B L	L1	50000	L2	19904	L3	10639	L4	20750	L5	25007	L6	39106
SALISBURY			46792		32149		49321		39424		42439		28999
DORCHESTER			19425		20963		41097		20694		49206		71094
SOUTHAMPTON			36913		21145		25151		36262		26226		15125
PORTSMOUTH			24519		30106		51251		40146		14604		12551
PLYMOUTH			46555		17762		14396		27474		47427		36914
CHATHAM			44439		26615		13411		29424		42944		34111
BOURNEMOUTH			10199		18735		45994		35592		35925		65009
WESTON.UPON-SEA			19000		33853		11041		29871		17298		11104
CAMBRIDGE			25125		21495		49069		49999		41009		95906
CHELMSFORD			49521		25125		21003		41126		11264		39109
ROCHESTER			38533		10059		12952		27651		25167		32100
TUNBRIDGE - WELLS			35187		19901		16879		10982		10289		29125
HASTINGS			26706		43944		19678		20901		20009		91876
EASTBOURNE			21677		45655		39333		26309		30926		33339
RAMSGATE			10630		19542		14506		30692		23069		10654
CANTERBURY			45211		13369		39156		10669		10696		35691
NORTHAMPTON			32609		42519		19794		41092		10924		17999
MAIDENHEAD			49321		26794		37915		50002		49004		39777
KINGSTON			41099		41005		37944		19499		39435		22379
			673461		530640		570530		608559		572868		761968

DPCS2

Branch Stock Cards.

OXFORD	B K	K1	41094	K2	36142	K3	49014	K4	13470	K5	41096	K6	19460
SALISBURY			31065		43061		16530		21256		31015		15130
DORCHESTER			10654		24256		45106		45427		10596		10695
SOUTHAMPTON			32094		41407		20943		26311		47091		41907
PORTSMOUTH			41099		36120		19904		13050		50054		40550
PLYMOUTH			36455		12427		45536		17247		40555		45505
CHATHAM			10994		42113		19409		19992		33394		34933
BOURNEMOUTH			20656		29607		25606		31133		45931		43195
WESTON.UPON-SEA			17952		23108		15279		27471		21251		21251
CAMBRIDGE			19522		42721		15229		43871		25630		30652
CHELMSFORD			16531		41407		15316		19745		25559		29553
ROCHESTER			14442		17223		14244		57641		10459		45901
TUNBRIDGE - WELLS			16536		22101		13656		40072		45108		41085
HASTINGS			17655		43608		15576		47061		48321		32184
EASTBOURNE			27109		21307		21709		29857		28359		25983
RAMSGATE			47199		17789		19947		18808		41009		10094
CANTERBURY			31098		31204		30891		35809		16118		18161
NORTHAMPTON			44118		24031		41184		40015		18106		40969
MAIDENHEAD			50001		40321		10058		34127		41056		10564
KINGSTON			49392		24310		32949		22676		37012		12370
			575666		614263		488086		605039		657720		570142

DPCS2

Branch Stock Cards.

MANCHESTER	C J	J1	30645	J2	10954	J3	41006	J4	25360	J5	35719	J6	10733
SHEFFIELD			19039		40910		32095		10999		50002		20966
HUDDERSFIELD			45590		39996		21009		21210		10196		49110
HARROGATE			43686		10734		10009		21660		41777		50000
STOKE - ON - TRENT			44701		40102		41067		33630		12979		39067
NOTTINGHAM			12005		10999		29444		40963		40791		30671
LEICESTER			43696		17302		38745		6300		46302		19119
RAMSBOTTOM			12902		14555		12954		41500		42166		37108
BLACKBURN			39402		41009		42519		21600		23016		41019
BLACKPOOL			10636		31027		26122		49513		41010		30009
ACCRINGTON			41099		46661		26301		42163		46360		41086
LANCASTER			10945		31092		21606		46302		31010		40554
CARLISLE			10355		41091		41635		45630		49098		10195
NEWCASTLE - O - T			45660		15944		36021		21009		39191		29092
GATESHEAD			30916		46915		46300		30098		40617		10635
SUNDERLAND			49110		10996		22166		10091		30719		41009
MIDDLESBOROUGH			36506		29177		26070		19191		41999		10109
DARLINGTON			41009		13992		49125		10263		30673		30107
LINCOLN			41002		32110		39125		26309		33603		41977
KINGS - LYNN			30610		40009		27741		41090		10616		31250
			639514		565575		631060		564881		697844		613816

DPCS2

Branch Stock Cards.

MANCHESTER	C K	K1	9435	K2	14990	K3	13906	K4	39061	K5	26351	K6	31010
SHEFFIELD			26794		21349		32149		41293		26309		46092
HUDDERSFIELD			42591		39602		10974		47901		26154		42210
HARROGATE			13693		45112		15125		12551		41997		39109
STOKE - ON - TRENT			25194		10630		42551		25514		30972		38463
NOTTINGHAM			45556		21772		10975		10635		29703		50006
RAMSBOTTOM			19901		13578		46395		29918		59990		41053
BLACKBURN			17251		35385		12825		30867		40163		36033
BLACKPOOL			12525		49512		12563		46076		30792		40014
ACCRINGTON			21549		25521		6309		31019		20930		16960
LANCASTER			35338		19005		21659		49105		40129		21663
CARLISLE			18753		10199		35908		41009		30663		1639
NEWCASTLE - O - T			27066		34449		49991		10941		49125		30630
GATESHEAD			27716		45615		42295		32105		46635		49715
SUNDERLAND			13006		42519		42100		19553		29124		40016
MIDDLESBOROUGH			21154		16339		29534		30591		10066		16004
DARLINGTON			29063		25149		36911		40099		34163		9915
LINCOLN			32194		47962		10863		30738		46302		15099
KINGS - LYNN			10949		10054		12169		41008		21616		12165
			449728		528742		485202		609984		641184		577796

Branch Stock Cards.

MANCHESTER	CL	L1	29791	L2	37826	L3	49179	L4	26783	L5	30174	L6	12361
SHEFFIELD			37542		43824		42375		38424		34521		12047
HUDDERSFIELD			19748		23549		48791		49523		24135		31936
HARROGATE			26481		19793		48162		13799		49400		36094
STOKE - ON - TRENT			47543		35892		30952		25983		29123		12602
NOTTINGHAM			41763		28431		31746		42381		43294		91311
LEICESTER			34871		31849		37148		18394		22950		28153
RAMSBOTTOM			15542		29532		14255		35292		41493		95999
BLACKBURN			21653		37456		13526		46573		29329		19325
BLACKPOOL			14796		17798		46791		19877		49811		34440
ACCRINGTON			33498		24652		43397		26542		32735		89605
LANCASTER			11897		39041		19781		41093		43101		32101
CARLISLE			13596		48503		35964		30584		10134		3942
NEWCASTLE - 0 - T			48019		37911		10984		39117		40506		31933
GATESHEAD			25541		27843		41552		28437		15941		13202
SUNDERLAND			31734		42647		31347		27644		19499		34925
MIDDLESBOROUGH			43746		18935		37464		19358		39553		17734
DARLINGTON			17926		20431		26179		40231		10123		30900
LINCOLN			26593		35749		25936		49375		34794		23088
KINGS LYNN			32748		17847		27483		18747		19010		66313
			575028		619509		663012		638157		619626		718011