With compliments and many thanks
PAS

TECH. NOTE
M.S.37

ROYAL AIRCRAFT ESTABLISHMENT
FARNBOROUGH, HANTS

TECHNICAL NOTE No: M.S.37

A DETAILED CODING PROGRAMME FOR DEUCE

by

P.A.SAMET, Ph.D., B.Sc.

AUGUST, 1957

MINISTRY OF SUPPLY, LONDON, W.C.2

UNCLASSIFIED

U.D.C. No. 518.5:518.12:(DEUCE)

Technical Note No. M.S. 37

August, 1957

ROYAL AIRCRAFT ESTABLISHMENT, FARNBOROUGH

A Detailed Coding Programme for DEUCE

by

P. A. Samet, Ph.D., B.Sc.

SUMMARY

This note describes a programme, RAE 415, which performs the detailed coding of DEUCE programmes.

LIST OF CONTENTS

LIST OF APPENDICES

1 Introduction

Experience of coding DEUCE programmes has shown that the detailed coding of instructions, necessary after storage has been allocated, is both tedious and fraught with error. The actual process of detailed coding is, however, quite mechanical and determinate. It is therefore well suited to computing machines.

The construction of a programme for DEUCE involves the programmer in the following six steps.

(1) Making up the logical structure of the programme.

(2) Breaking down this logical structure into separate instructions in their correct sequence (the 'flow diagram'). At this stage constants and links for subroutines still have symbolic addresses.

(3) Allocation of storage space for the instructions in the flow diagram and constants.

(4) Detailed coding of the instructions since each one names its successor, which may be anywhere in the store, and also refers to the particular minor cycle during which the operation starts.

(5) Punching these coded instructions in binary on instruction cards. All programme constants have to be converted to binary as well.

(6) The binary punching and the coding must be checked very carefully before testing of the programme can begin.

Most of these steps are common to programming for any machine, although 4 and part of 5 can be avoided at some machines (e.g. on Pegasus). The present note describes a programme that enables the programmer to use the machine to perform steps 4 and 5, and will materially assist in step 6 by reducing the number of errors that have to be eliminated.

Certain general routines exist, at present mainly for American machines such as the I.B.M. 704 and Univac, which will take over at step 2. These are the so-called 'Compiler routines', which can construct a complete programme from its logical steps. These routines are all characterised by requiring vast amounts of storage space, even though the programme they produce may be quite short, and are very elaborate. No such programme exists for DEUCE, nor is one planned.

Programmes with the more limited objective of taking over at step 3 are fairly common. Examples of such 'Assembly routines' are the 'Initial Orders' of EDSAC II and Pegasus.

Compiler and assembly routines have, so far, only been constructed for 1-address machines. The problems associated with writing efficient routines of this nature for multi-address machines are very complicated and have not yet been tackled.

The present programme is a much less ambitious one and takes over the purely mechanical parts of DEUCE programming. As far as the author is aware it represents the only attempt so far by any DEUCE users to employ the machine for assisting in the preparation of programmes.

The programme has been in successful operation at RAE for a period of more than six months. Experience has shown that it is simple to use and that errors are few and easily detected.

2 Specifications for a Coding Programme*

The immediate requirements are as follows:

(1) The step from flow diagram, with storage already allocated, to input is to be as short as possible. In particular, punching of instructions should be in decimal, not binary.

(2) Only the absolute minimum of information about an instruction should be used.

(3) It is desirable to be able to include library subroutines and to read constants (punched in decimal or binary).

(4) There must be checks against inconsistency of information, punching errors, over-writing of instructions and faulty scaling of numbers.

(5) The operation should be simple and rapid.

(6) It must be easy to make amendments to a programme.

(7) The programme must be sufficiently general to cater for all the whims of the programmer.

Programme RAE 415 satisfies all these seven conditions.

3 Operation of RAE 415 (I)

The programme consists of four sections, linked by a master routine. The working sections are, in order of use:

(1) Read library subroutines;

(2) Main coding section;

(3) Read constants punched in binary or decimal;

(4) Punch out coded programme.

Details for the assembly of data cards and punching of the programme are given in Section 7 and Appendix I. The detailed operation of the programme is described in the succeeding sections.

Instructions to be coded are punched in decimal, one per card. Decimal constants may be either integers or fractions.

An example of the punching required is given in Appendix II.

4 Notation

Q - Minor cycle in which the instruction is to be obeyed, if different from location of instruction.

n - Block number

A - Delay line)
m - Minor cycle) in which instruction is stored

S - Source

---

* In what follows, 'Coding Programme' is used to mean 'Detailed Coding Programme'.

D - Destination

F - First minor cycle of transfer

Ch - Characteristic

L - Length of transfer

J - 'Joe' digits (P22-25)

N - Delay line } of next instruction
t - Minor cycle }

C - Constant (decimal or binary)

r - Position of decimal constant. (Also special use for binary constants)

The above quantities can be punched, in decimal, on data cards. Many are optional and are punched only when required.

Three further symbols are used.

X - This denotes that a particular quantity has not been punched; i.e. F = X means that F has been omitted.

ℓ - Number of minor cycles occupied by the larger of S and D. This is calculated by the programme (The 'length').

d - Number of decimal digits in a decimal constant.

I - An instruction.

If two instructions are involved at any point they will be denoted by I_1 and I_2. Subscripts are used to distinguish the appropriate quantities, i.e. Q_1, A_2, etc.

The block number n needs some explanation. It frequently happens that a programme cannot be kept in the high speed store in its entirety. In such a case the contents of one or more of the DL's are replaced, from time to time, by new instructions. The block number distinguishes the different occupants of the DL's. The instruction destined for A_m of block n is stored temporarily, during coding, in minor cycle m of track n/A.

For decimal constants the value of r gives the scaling factor. The constant is taken as an integer if r is punched and is stored as a multiple of P_r* (r = 0 is not allowed). If r = X the constant is regarded as fraction, to be stored with 30 binary places.

5 The Numbering of Minor Cycles

In the DEUCE instruction code the Wait and Timing numbers are always relative. That is to say, they never refer to the actual minor cycle required but only to its position relative to the current instruction. While writing the flow diagram, however, one normally writes down actual minor cycle numbers. This system has been adopted for the coding programme.

---

* P_r is the r^{th} digit in the word, counting from the least-significant end.

Whenever a delay line (DL) is involved in a transfer, any reference to the first minor cycle of transfer, i.e. F, is to the actual minor cycle involved. The programme codes in such a way that transfer starts as soon as possible whenever F = X.

A modification of this method is required when the transfer involves a double store (DS) or a quadruple store (QS). The minor cycles of a QS are numbered 0, 1, 2, 3. Fairly obviously it is undesirable that references to these minor cycles should be coded to take effect only in minor cycles 0, 1, 2, 3. On the other hand, there are occasions when we want an operation to take place in a particular minor cycle although it could have been done earlier - for instance we may wish to waste time during a multiplication. Similar considerations hold for a DS (where the two minor cycles are numbered 2 and 3).

Normally it is unnecessary to punch the characteristic Ch as this can be found from L, the length of transfer. It is therefore possible to use the characteristic for another purpose. Care must be taken, however, to punch a value of Ch that is compatible with L and F.

The coding programme uses punching of the characteristic to indicate absolute timing. If Ch = X the instruction is coded to take effect as soon as possible. Actual minor cycles are taken whenever Ch ≠ X.

In the cases where Ch is punched, the compatibility of Ch, F and L is checked whenever this is possible. Whenever Ch = X, F is ignored in all instructions with destination 24 (except 0 - 24 and 1 - 24) and in instructions concerned with the magnetic drum. Multiplication and division are coded to start in the first available odd minor cycle: an even value of F will be recorded as a failure.

Some examples will make these ideas clearer.

	A	m	S	D	F	Ch	L	
I_0	1	0	17	13	1	X	X	1_0: 17_1 - 13 (in m.c. 5)
I_1	1	6	21	14	3	0	X	1_6: 21_3 - 14 (in m.c. 3)
I_2	1	4	9	15	7	X	X	1_4: 9_7 - 15
I_3	1	7	19	16	X	X	X	1_7: 19_3 - 16
I_4	1	10	14	25	X	X	X	1_{10}: 14 - 25
I_5	1	12	19	21	2	X	X	1_{12}: 19_2 - 21_2
I_6	1	15	21	22	2	X	2	1_{15}: 21 - 22 (2 mc, e,o)
I_7	1	17	21	22	12	2	X	1_{17}: 21 - 22 (2 mc, starting in m.c.12)
I_8	1	3	17	22	1	X	X	1_3: 17_1 - 22_3 (in m.c. 5)

6 Operation of RAE 415 (II)

This section describes in some detail the operations of the various sections of the programme.

6.1 Reading Library Subroutines

As its name implies, this is the part of the programme which reads library subroutines and places them in the store.

Subroutines are preceded by a 'block number card', which has n punched in binary, times P5, on the Y-row. All subroutines for the same block can make use of the same block number card, although this is not essential. If required, every subroutine can have its own block number card.

It often happens that two subroutines occupy different minor cycles of the same delay line, e.g. certain multiplication and sum series routines can be fitted together like this. It is not necessary to merge such routines on the Hollerith equipment first, as is normally the case. The programme takes care not to over-write one instruction by a different one. If this should happen, because two subroutines should happen to use the same minor cycle, a failure will be recorded. A count is kept of such failures.

When all the required subroutines have been stored a special end card is read. This is the signal that the next part of the coding is to start.

6.2 Main Coding Section

This part is concerned with coding instructions.

As indicated earlier the instruction for minor cycle m of DL A in block n is temporarily stored in minor cycle m of track n/A.

For all instructions it is necessary to punch

A, m, S, D .

This, in fact, is what is always on the flow diagram. Further items of information are punched as required. The last instruction of the programme is followed by an end card - actually the same as the one following the subroutines.

The method of procedure is now fairly simple in outline. The next instruction source and timing number of I_1, are, in general, determined by the location of the next instruction to be read, I_2. The only exception to this rule occurs when there is a jump to an instruction already stored, as will happen at the end of a loop or on entry to a subroutine. In this case N_1 and t_1 tell us the location of the next instruction after I_1. The only other part of I_1 that can possibly depend on I_2 is the wait number when the characteristic is odd. The programme must therefore have I_2 available before the coding of I_1 is completed. In fact, the programme is always 'one instruction behind'.

Ten steps are involved in the coding of an instruction. They are

(i) Read I_s. Fetch track where I_{s-1} is to be stored.

(ii) Unpack I_s.

(iii) Find next instruction source and timing number of I_{s-1}.

(iv) Find wait number of I_{s-1}; check that this is consistent with the length of transfer if F is given.

(v) Check for over-writing. Store I_{s-1}.

(vi) Test for end of programme.

(vii) Find S_s, D_s and ℓ_s.

(viii) Find Ch_s; check the consistency of Ch_s and L_s.

(ix) Add the space digits* and the 'Joe' digits.

(x) Find the minor cycle in which I_s is obeyed and return to step (i).

It turns out that steps (ii) - (x) can be executed in the time available between cards while the reader is running. This means that coding can proceed at 200 instructions per minute.

In outline the above is quite simple. The actual details are rather intricate. Some of them are given below.

(i) Read I_s

This is done by DEUCE subroutine 160 (R14), which reads 16 2-digit numbers. There are only twelve quantities that can occur for one instruction. The subroutine has been slightly altered to allow exit after reading (three) particular rows. These exits lead to instructions which perform the head-shifts and read the track where I_{s-1} is to be stored. In addition, the Y-row is stored, to be added to the instruction later. Any punching on the X-row implies that the instruction is a stopper. The Go digit is added to the stored contents of the Y-row at this point.

There is no check on decimal punching.

(ii) Unpack I_s

This is a specially written subroutine. The twelve quantities are unpacked and stored as integers x P17 in DL 10. (The corresponding quantities for I_{s-1} are in DL 12 at this time). Whenever a quantity is unpacked it is examined to see whether it was actually punched. This is quite easy, since the read subroutine regards a blank column as having a '9'. An unpunched quantity therefore appears to have the value 99. All such blanks are replaced by a P32 in the appropriate minor cycle of DL 10. Quantities that have actually been punched are checked to see that they do not exceed 31, with the exception of L which may go up to 33.

(iii) Next Instruction Source and Timing Number of I_{s-1}

Here we first examine t_{s-1} and N_{s-1}. If t_{s-1} has been punched it is assumed that N_{s-1} was also punched, (N_{s-1} = X has the same effect as N_{s-1} = 0 or 8), and these two now specify the next instruction. In this case I_s is not necessary to finish the coding of I_{s-1}.

If t_{s-1} = X, however, A_s and m_s give the location of the next instruction. There is one exception to this. When Q_s is punched I_s is not obeyed in minor cycle m_s. The programme will now take Q_s as the minor cycle of the next instruction and put N_{s-1} = 0. If the transfer is to destination 0 it is necessary to punch the characteristic 1 if Q_s and the first possible minor cycle of transfer differ. As an example:

---

* These are the P1 and P31 digits, which are not used in instructions.

$$1_{28} : \quad 13 - 0$$

$$Q_{30} \qquad \cdots$$

does not need a characteristic, but

$$1_{27} : \quad 13 - 0$$

$$Q_{30} \qquad \cdots$$

would do so.

When a subroutine is coded by RAE 415 the last instruction must specify the location where the link is stored as its N and t.

Let us denote the minor cycle of the instruction following I_{s-1} by t'_{s-1}. The timing number of I_{s-1} is now given by

$$T_{s-1} = t'_{s-1} - m'_{s-1} - 2 \pmod{32},$$

where m'_{s-1} is the minor cycle from which I_{s-1} is obeyed. (m'_{s-1} is m_{s-1} or Q_{s-1} if the latter is punched).

The timing number and next instruction source are now shifted to their respective positions in the instruction word and added to what has already been built up of I_{s-1}.

At this stage I_{s-1} is complete except for its wait number.

(iv) Wait Number of I_{s-1}

This section computes the wait number, W, and, if possible checks its consistency with the given data. W is given by

$$W_{s-1} = T_{s-1} - L_{s-1} + 1 \pmod{32}$$

if L_{s-1} is punched, and by

$$W_{s-1} = F_{s-1} - m'_{s-1} - 2 \pmod{\ell_{s-1}}$$

if F_{s-1} is punched, (m'_{s-1} has been defined above).

In all other cases $W_{s-1} = 0$, except for multiplication and division orders obeyed from an even minor cycle.

W_{s-1} depends on I_s if the transfer has an odd characteristic; this includes the case when $L_{s-1} > 3$.

If L_{s-1} and F_{s-1} are both punched W_{s-1} is computed by both the possible methods and the results compared. For consistency we must have

$$T_{s-1} - L_{s-1} + 1 \equiv F_{s-1} - m'_{s-1} - 2 \pmod{\ell_{s-1}}$$

This simply means that the two ways must agree.

For multiplication and division instructions, which must start in an odd minor cycle, W_{s-1} has the opposite parity to m'_{s-1}. If F_{s-1} is punched a check is made to see that this condition is satisfied.

Once we have found W_{s-1} we add this to the instruction I_{s-1}, which is now complete. All that remains is to store this instruction, first making sure that we do not over-write a different instruction.

(v) Check for Over-Writing

During steps (ii) - (iv) track n_{s-1}/A_{s-1} has been read from the drum. It is in minor cycle m_{s-1} of this track that I_{s-1} is to be stored. Therefore we inspect minor cycle m_{s-1}. If this minor cycle is empty we place I_{s-1} in it. If it contains a non-zero word, we compare this with I_{s-1}: should the two differ a failure is recorded and a distinctive pattern placed in minor cycle m_{s-1}. DL 11 is now written up on the drum once more and we proceed to the coding of I_s.

This over-writing check differs from the one used when reading sub-routines, because instructions already in the store do not get replaced by zero. In the present case if minor cycle m_{s-1} is non-zero and I_{s-1} is zero a failure is shown.

Note that instructions which are different may have the same effect. Thus:

$$1_{28} : \quad 0, 13 - 0, 1, 0, 0$$

and

$$1_{28} : \quad 0, 13 - 0, 0, 0, 0$$

are identical in operation, but trying to over-write one by the other will give a failure.

(vi) Test for end of Programme

We first move the quantities associated with I_s to DL 12 from DL 10. DL 10 will now be free to store I_{s+1}. Next, we see whether an 'end card' has been read. Such a card is recognised because the source has been left unpunched (among other things); it is a simple matter to recognise that S = X. If this is the case we are ready to proceed to the next section, which reads constants. Otherwise we proceed to code I_s.

(vii) Find S_s, D_s and Calculate ℓ_s

The source and destination are first picked out from DL 12 and shifted to their respective positions in the instruction word and added together. I_s now consists of S - D. The next thing to do is find out whether Ch was punched. If so, $\ell_s = 32$ and we are dealing with absolute numbering of minor cycles. A special marker is left for later use if the instruction is 0 - 24 or 1 - 24, as this has to take place in an odd minor cycle.

The case Ch = X needs more careful attention. For all destinations except 23, 24, 30, 31 the "length", ℓ, (as defined below) is the same as the length of the source with the same number. Destination 23 is special because it concerns DS 21, whereas source 23 is connected to TS 14. Destinations 24, 30 and 31 are special in the sense that the source no longer refers to a part of the high-speed store and these instructions are all 'single-length'*. An exception occurs for 0 - 24 and 1 - 24, which are double-length (a DS is involved) and need special treatment when the wait number is calculated.

We define the length of a source (destination) as the number of distinct words sharing this source (destination) number.

This means that the length of a DL is 32, of a QS is 4, of a DS is 2, of a TS is 1. The length of a functional source like S23 or S26 is 1, of a functional destination like D22 is 2. We put the length of destinations 27, 28, 29 equal to 1, although strictly they do not conform to the definition.

Denote the length of the source by ℓ^S, the length of the destination by ℓ^D. We then take

$$\ell = \max(\ell^S, \ell^D) .$$

To find ℓ we therefore have to find ℓ^S and ℓ^D for the source and destination of I_s. The simplest method is to have a table of ℓ^S and ℓ^D and to pick out the appropriate values. S and D are available to us in the P17 position and so may be used to modify instructions which select ℓ^S and ℓ^D respectively. As indicated above it is necessary to pick out destination 23 separately.

ℓ is used in calculating the wait number, which is given by a congruence modulo ℓ. As ℓ is always a power of 2, say 2^a, the congruence is best evaluated by collating with the d digits of $\ell - 1$. We require the larger of two such quantities. It was found convenient to store the ones-complement of $(\ell^S - 1)$ P17 and $(\ell^D - 1)$ P17, instead of $(\ell^S - 1)$ P17 and $(\ell^D - 1)$ P17. Denote these quantities by $\sim(\ell^S - 1)$ P17 and $\sim(\ell^D - 1)$ P17.

We now pick-out $\sim(\ell^S - 1)$ P17 and place this in TS 15, $\sim(\ell^D - 1)$ P17 into TS 14, place a string of 1's in TS 13 and subtract (14 & 15) from TS 13. The result of these manoeuvres is $(\ell - 1)$ P17 in TS 13.

An example will make this clear. Let $\ell^S = 4$, $\ell^D = 2$; the instruction could be something like 17 - 22. Then

$\sim(\ell^S - 1)$ P17 = 11111, 11111, 11111, 10011, 11111, 11111, 11

$\sim(\ell^D - 1)$ P17 = 11111, 11111, 11111, 10111, 11111, 11111, 11

$\{\sim(\ell^S - 1)\}$ & $\{\sim(\ell^D - 1)\}$ = 11111, 11111, 11111, 10011, 11111, 11111, 11

which gives $\ell - 1$ = 00000, 00000, 00000, 01100, 00000, 00000, 00 = 3P17,

as expected.

---

* Although these instructions may have Ch = 1, which has a special significance. Ch is therefore punched, but F = X = L. These instructions are 'single-length' in the sense that the minor cycle of transfer may be picked arbitrarily.

(viii) Characteristic of I_S; Check Consistency of Chs and L_S

This is a simple part of the programme, which finds the correct characteristic for I_S. If $Ch \neq X$ we check that this is compatible with L whenever the latter is punched. On the other hand, if $Ch = X$ the characteristic is found from L. When $Ch = X$, $L = X$ we put the characteristic equal to zero.

The characteristic, thus found, is now added to the part of I_S that has already been assembled.

In cases where Ch and L are incompatible the instruction is destroyed and is later replaced by a distinctive marker. A failure is recorded.

(ix) Spare Digits and Joe* Digits

The spare digits, i.e. the P1 and P31, which were punched on the Y-row of the card carrying I_S and stored earlier in the programme, are now added to I_S. The Joe digits (P22 - P25) are shifted up to their correct position in the word and added to I_S.

(x) Find Minor Cycle from which I_S is Obeyed

Previously this has been denoted by m'_S. It is given by

$$m'_S \begin{cases} = m_S & \text{if} \quad Q_S = X, \\ = Q_S & \text{if} \quad Q_S \neq X. \end{cases}$$

We replace Q_S by m'_S and use this in the subsequent operations on I_S.

At this stage we have coded as much of I_S as is possible without having I_{S+1} available. The instruction now consists of

(P1), ——, S - D, Ch, ——, Joe, ——, (P31), Go .

We now return to step (i) and read the next card.

It will be noticed that at the time of reading the first card we appear to be half-way through coding an instruction. This must be arranged to be a harmless dummy that is actually stored on a track not used by the coding section. This initial instruction is arranged to be

$$Q_{30} : \quad 0, \quad 0 - 0, \quad 0, \quad 0, \quad 0\ X ;$$

it is stored in minor cycle 0 of track 0/0.

Similarly, after the last instruction has been read it is necessary to read one extra card that allows the coding to be completed. This particular instruction must be complete by itself. It must either return to a previous instruction of the programme (possibly itself) or, if it is the last instruction of a 'brick' or subroutine, it has a definite instruction, like 1_{30}, to go to. If the last instruction does not have N and t specified the programme will fall out when this instruction is reached.

---

* The etymology of this is unknown.

The end card is accordingly made blank, except for a P32 on the Y-row and punching in (Hollerith) column 54 on the 9's row. (This is the standard 'end card'). In this case the P32 is actually unnecessary and is put in to make all end cards the same. As indicated earlier the test for ending is the examination of the source. The punching in column 54 is necessary to stop the reader, because at the time the end-test is done too many major cycles have elapsed after the 9's row to stop the reader without passing the next card.

This particular end card will also pick out the case where the last instruction did not specify its successor. The reader stops and the programme enters a loop of five instructions from which there is no escape. The loop is

```
→ 3,   16 - 27
  4,   30 - 14
  4,   15 - 27
  2,   10 - 14
  3,   10 - 16  ;
```

it is easily spotted when operating the machine, because all the I.S. lights are on and nothing happens in the monitor tubes.

It was mentioned earlier that the whole of the Y-row is added to the instruction. This gives one method of inserting binary constants. The constant is punched on the Y-row and an instruction is made up which is the zero word. This can be

$$Q_{30} : \quad 0 - 0\,X$$

leading to 8_0.

In the first version of the coding programme, RAE 315, this was the only method of putting a constant into the programme. The present version, however, allows constants punched in either decimal or binary to be read in a more convenient form. The section that performs this task will be described next: it is called in after an end card has been read by the coding section.

6.3 Reading Constants punched in Decimal or Binary

This is the last of the input programmes associated with the coding programme. It reads decimal and binary constants; they may be in any order.

(i) Decimal Constants

These may be integers, stored as a multiple of P_r, or fractions with 30 binary places. Constants must satisfy either

(i) $\quad |\,C\,| < 2^{32-r}$, integer C stored $\times P_r$,

or (ii) $\quad -2 \leqslant C < 2$, fractional C.

Within certain limits C may have any number of decimal digits, provided only that it lies in the above ranges. The address of C, i.e. n, A, m, is punched in the same way as that of an instruction. The sign

is punched in the column before the first decimal digit, (a positive sign need not be punched), and an end mark is punched on the X-row in the column following the last digit. Omission of this punching causes a failure to be recorded. For fractions there is one digit before the decimal point.

The first two columns of the DEUCE field, previously used for Q, are now used to distinguish integers from fractions. This is where r is punched. If $r = X$ C is read as a fraction. It is not allowed to have more than 10 decimal digits (i.e. 9 decimal places). The other case, $r \neq X$, indicates that C is an integer, required as a multiple of P_r. This allows one to put in constants at various places in the word, for instance $\times P_5$ or $\times P_{17}$. $r = 0$ is not allowed and we must have $r \leq 31$. Integer C may have any number of non-significant zeros punched before the actual digits of C: these zeros are ignored.

(ii) Binary Constants

These are punched on the Y-row. n, A, m are punched as usual and the rest of the card is left blank except for a P1 on the X-row. This is the mark that the card carries a binary constant. Any card with a P1 on the X-row is assumed to carry a binary constant on the Y-row: anything else on the card is ignored.

Over-writing of constants by identical constants is allowed, as for instructions. An end card follows cards bearing constants and tells the programme to proceed. This card is the same as end cards for previous sections.

6.4 Punch out Programme

We are now ready to punch out the programme that has just been coded.

The punch out section will first produce an initial card, which may be suppressed by setting anything on the I.D. keys. It then punches out the non-zero delay lines of each block in order, starting with DL 12 and working downwards. Filler instructions are punched on the cards; DL's 9, 10, 11 and 12 use DL 2 as auxiliary DL. All fillers lead back to 10. On the cards for DL 1 the next instruction source and timing number on the 1's row are left blank, to be punched by the programmer.

When all the DL's have been punched the programme produces one extra card, which records the number of failures that have occurred. Control is then sent to the master routine, which clears the working part of the magnetic drum and then returns to the beginning, ready to code the next programme.

7 Assembly of Cards

Data cards are assembled in the following order:

(Block number card, with $n \times P_5$, in binary, on the Y-row
(Subroutines for this block.

(Repeat for as many blocks as necessary).

End card

Decimal Instructions

End card

Constants (Decimal and binary, in any order).

End card.

The end cards are all the same, with a P32 on the Y-row and punching in (Hollerith) column 54 of the 9's row. The rest of the card is blank. Even if there are no cards of a particular category an end card is still necessary.

Subroutines for the same DL do not have to be merged beforehand, nor do they have to be read consecutively. Ordinary library copies, with fillers, are used for the subroutine cards. There must not be any punching on the Y-row of the first card of a triad.

The complete programme to be coded may not have more than 14 delay lines which occupy the same DL in the high speed store. Thus 168 delay lines is the maximum permitted length for a programme that is to be coded.

8 Output of Cards

RAE 415 produces the coded version of a programme. Output consists of

Initial card (if required),
Non zero DL's of block 0,
Non zero DL's of block 1,
- - -
Failure card.

The initial card is the original one that only sets even minor cycles and does nothing else. Punching of this card may be suppressed by setting any non-zero number on the I.D. keys while the data cards are being read.

The DL's are punched out in the order 12, 11, ... 1. DL's 9, 10, 11, 12 use DL 2 as auxiliary DL and all fillers assume 1_0 to be empty. On the 1's row of the first card for DL 1 the next instruction source and timing number are left blank.

The failure card may have punching on the Y and X rows. On the Y row there are two numbers in binary, x P1 and x P17. These give the number of times a word has been over-written by a different one and the number of punching mistakes and inconsistencies respectively. The X-row has one number, x P1, which gives the number of decimal constants outside the permitted ranges. Wherever possible the failures are marked in the programme (see below). Ideally, of course, the failure card is blank, showing that there are no failures.

9 Failures

These are of three types, each with a distinctive marker.

(i) Over-writing failures. These occur when a word is replaced by a different one. Both words are lost and P1 - P16 is placed there instead.

These are counted times P1: the number of such failures is punched, x P1, on the Y-row of the failure card.

(ii) Inconsistency failures. These occur when incompatible information has been given (e.g. Ch = 2 and L = 3, or 0 - 24 with F even, etc). The instruction is lost and is replaced by P17 - P32.

Faulty punching, which may give a quantity a value exceeding 31 (greater than 33 for L) in general means that the instruction is lost completely. It may also affect the previous instruction, as calculation of the wait number may be impossible. If this is so the previous instruction is also destroyed. A failure will be counted but there may be no indication of where it occurred.

All of these failures are counted x P17 and their number appears on the Y-row of the failure card.

~~Their~~ There is no check on decimal punching.

(iii) Numbers out of range. We must have

$|C| < 2^{32-r}$, integer C, stored x P_r,

or $-2 \leq C < 2$, fractional C.

Numbers that do not satisfy either of these conditions are replaced by P9 - P24. The number of such failures is recorded and appears on the X-row of the failure card.

There is no check on decimal punching.

In the main coding section there is no limit on n and A (except that they are ≤ 31). In the 'Read Constants' section we must have $n \leq 13$ and $A \leq 12$. These limit must always be observed, even for the coding section. Otherwise there is a possibility of over-writing RAE 415 itself: the punch out section, in any case, would ignore anything outside the limits given.

Acknowledgment

The author wishes to thank his colleagues, in particular D.G. Burnett-Hall and J.M. Watt, for many useful suggestions.

Attached:

Detachable Abstract Cards

Advance Distribution:

DGSR(A)	
DWR(D)	
TIL1(b)	150
ARDE (Dr. Maccoll)	
RRE (Mr. Taylor, Mr. Magowan)	
NPL (Maths. Div.)	5
DRAE	
DDRAE(A)	
DDRAE(E)	
Aero Dept	3
GW "	
Arm "	
Str. "	
RPD, Westcott	
Trials Dept	
CSEE	
IAM	
IAP Dept	
Naval Air Dept, Bedford	
Radio Dept	
Library	
D/NGTE	

APPENDIX I

Layout of Cards

Cols.	21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44	Remarks
Instructions	Q n A m S D F Ch L Joe N T	A, m, S, D must be punched. All others are optional. 'Stoppers' have punching (anywhere) on the X-row. P1 and P31 are punched on the Y-row. Punching in cols. 45-52 is not allowed (except Y and X rows).
Decimal Constants (Integer)	r n A m ± ⟵C⟶	r punched, C is integer, stored × P_r, $\|C\| < 2^{32-r}$. An end mark is punched in the X row after the last decimal digit. Positive signs are optional, but a space must be left. The first digit of C is in col. 30. Non-significant zeros will be ignored. $1 \leqslant r \leqslant 31$. r = 0 is not allowed.
(Fraction)	r n A m ± ⟵C⟶	r unpunched. $-2 \leqslant C \leqslant 2$, 30 b.p. An end-mark is punched, as for integers. Not more than 10 decimal digits are allowed.
Binary Constants	X row ⟵C⟶ (whole Y row) n A m	Constant on Y-row. P1 on X-row.

The end-cards have punching in (Hollerith) column 52 on the Y-row and column 54 on the 9's row.

In most cases single digit quantities may be punched as such in the appropriate units column. The only exception is a single 9, which must always be punched 09.

Data.		Remarks
Instructions		
Decimal Constants (Integer)		
(Hexadec)		
Binary Constants		

APPENDIX II

An Example

The following is part of a programme for finding correlation coefficients. It illustrates most of the points of the previous sections.

1_0: 30 - 13

5_2: 15 - 26

5_4: 5_6 - 15

4_6: 30 - 21 (2 mc)

4_9: 13 - 23_3

4_{11}: 0 - 24

4_{12}: 15 - 1_{30}

1_{30}: (5_6) 21 - 20 (2 mc, in mc 13,14)

4_{16}: 30 - 21 (2 mc)

4_{21}: 17_1 - 16

4_{25}: 16 - 23_3

4_{27}: 4_{29} - 13

4_7: 16 - 14

3_{30}: 0 - 24 (Ch = 2)

5_1: 8 - 24 (in mc 4)

3_3: 14 - 22_3

5_5: 21 - 22 (32 mc, starting even)

5_7: 6 - 24

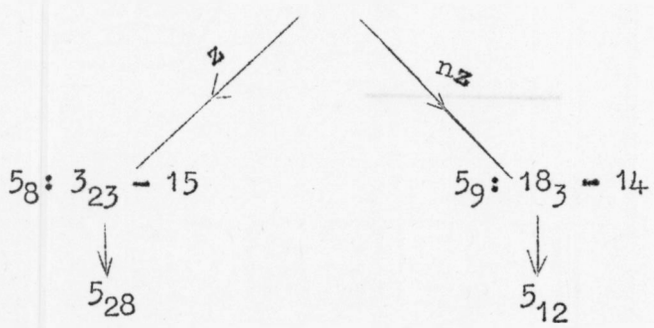
4_{16}: 16 - 14

1_{28}: 13 - 0

$Q_{30}(4_{29})$ $18_{2,3}$ - 22

1_4: 21 - $18_{2,3}$

4_{22}: 18 - 28 (4 mc)



The punching of the above for RAE 415 is given below.

Q	n	A	m	S	D	F	Ch	L	J	N	t	Remarks
		1	0	30	13							
		5	2	15	26							
		5	4	5	15	6						
		4	6	30	21			2				
		4	09	13	23	3						
		4	11	0	24							
		4	12	15	1	30				1	30	See Note 1
30		5	6	21	20	13	2	(2)				See Note 2
		4	16	30	21			2				
		4	21	17	16	1						
		4	25	16	23	3						
		4	27	4	13	29						
		4	7	16	14							
		3	30	0	24		2					
		5	1	8	24	4	0					See Note 3
		3	3	14	22	3						
		5	5	21	22	2		32				See Note 4
		5	7	6	24							
		4	16	16	14							
		1	28	13	0							
30		4	29	18	22	2		2				
		1	4	21	18	2		2				
		4	22	18	28			4				
		5	8	3	15	23				5	28	
		5	09	18	14	3				5	12	

Note 1. N and t will make sure that the next instruction is taken from 1_{30}; otherwise it would be 8_{30}, because the instruction in 56 is obeyed Q_{30}.

Note 2. L is optional in this case. Ch must be punched, because the instruction is to be obeyed in mc 13. In this case L, F and Ch will be checked.

If only F and L were punched the instruction would be obeyed in mc 1 and 2.

Note 3. Ch punched to indicate that the instruction is obeyed in mc 4.

Note 4. F = 2 implies 'start even', not 'start in mc 2'; if Ch were punched, this instruction would start in mc 2 and would give a failure.

UNCLASSIFIED

Royal Aircraft Est. Technical Note No. M.S. 37
1957.8
Samet, P.A.

518.5:
518.12:
(DEUCE)

A DETAILED CODING PROGRAMME FOR DEUCE

This note describes a programme, RAE 415, which performs the detailed coding of DEUCE programmes.

UNCLASSIFIED

UNCLASSIFIED

Royal Aircraft Est. Technical Note No. M.S. 37
1957.8
Samet, P.A.

518.5:
518.12:
(DEUCE)

A DETAILED CODING PROGRAMME FOR DEUCE

This note describes a programme, RAE 415, which performs the detailed coding of DEUCE programmes.

UNCLASSIFIED

UNCLASSIFIED

Royal Aircraft Est. Technical Note No. M.S. 37 — 518.5: 518.12: (DEUCE)
1957.8
Samet, P.A.

A DETAILED CODING PROGRAMME FOR DEUCE

This note describes a programme, RAE 415, which performs the detailed coding of DEUCE programme.

UNCLASSIFIED

UNCLASSIFIED

Royal Aircraft Est. Technical Note No. M.S. 37 — 518.5: 518.12: (DEUCE)
1957.8
Samet, P.A.

A DETAILED CODING PROGRAMME FOR DEUCE

This note describes a programme, RAE 415, which performs the detailed coding of DEUCE programmes.

UNCLASSIFIED

DETACHABLE ABSTRACT CARDS

These abstract cards are inserted in Reports and Technical Notes for the convenience of Librarians and others who need to maintain an Information Index.