2.501/22

THE ENGLISH ELECTRIC CO., LTD.

NELSON RESEARCH LABORATORIES
STAFFORD
MATHEMATICS DEPARTMENT.

Telephone :—Stafford 700.

Report No. NS y 81
Date 25.11.57.
Reference
Order No.

Front Sheet.

Data - Sheets 1-5.

NMDC/REC/D22

PRINCIPLES OF PROGRAMMING	Report by Miss A. Birchmore

SUMMARY

This report briefly describes an approach to the job of writing a computer program which might lead to more efficient working both on and off the computer. Reference is made to various programmed and engineered facilities which are available to those writing programs for DEUCE.

pp A. Birchmore.
MATHEMATICS DEPARTMENT.

ET

PRINCIPLES OF PROGRAMMING

1. INTRODUCTION

A good program will be able to satisfy criteria beyond that of "working. The four most important are that it is a) simple to write,
b) simple to test,
c) simple to use,
d) simple to understand and alter.

Beginners often underestimate the importance of simplicity and method in program writing; these two are as important as optimal coding and "tricks" which frequently have more attention than they deserve. This note frames some questions which will help to achieve a good style, if they are answered at a sufficiently early stage in planning and writing the program, and gives advice on finding the answers. Answers will often need to be found before any actual programming is attempted and in all cases before the program reaches the computer for its first test. Complexities in programming usually arise because a problem has been tackled too late, or in the wrong way.

Questions.

1) Is the program methodically organised?

2) Have checks been programmed for computations wherever possible?

3) Does the program start by establishing uniform conditions in the computer?

4) Has the programming been reduced to a minimum by using library subroutines?

5) Can the program be restarted without reading in the cards again?

6) Have you the following written information?

a) Logical flow diagrams?
b) Flow diagrams?
c) Coding sheets?
d) List of subroutines used with their requirements, results and failure indications?
e) List of all the cards going into the computer?
f) Information on data, and results expected?

7. Has the data been prepared for test cases?

8) Have you the two types of practical knowledge for testing your program:

a) Use of keys and switches?
b) Use of programmed aids, notably POST MORTEM ZP29?

2. PROGRAM ASSEMBLY

2.1. Introduction.

A program is usually stored on the drum as it is read into the computer, and is brought down into the high-speed store in manageable chunks called sections.

2.2. Storing Programs on Drum

Programs exist which will read triads continuously and store them on tracks of the drum.

a) Read to Drum ZP14 - is a two card program which is placed in front of the triads; each triad must have P15 + T x P17 on the Y-row of its first card, except that if the triad is to go to the next track (T+1), the Y-row may be left blank. The last triad also has a P31 on this row.

b) Track Assembly ZP04 - is a 3 card program which needs a parameter card for each group of triads going to consecutive tracks. The parameter card has T_0 x P5 on the Y-row where T_0 is the first track of the block to be filled, and N x P5 on X-row where N is the number of tracks in this block. 0-row is non-zero if the block is not the last.

c) Advantages of ZP14 and ZP04 - both these programs make it possible to put normal headings for NIS DLs on triads, which is useful for identification when making alterations, etc.

The Humby Card (described on page 58 of the Programming Manual) cannot be recommended for this reason.

d) Misuse of ZP14 and ZP04 - these programs are not intended to be used for storing data on the drum. Data is uaually in an unsuitable form for them, e.g. in decimal or in standard matrix form, or in the wrong order. It will be necessary to write a section of your program to read the data and store it on the drum.

2.3. Bringing Down the Sections

Control Routines have been made that can be entered with a parameter to indicate which section of program is next required; they bring down this section and enter it. Two are described in DEUCE News 12, paras 9-11. Each is part of an Assembly Scheme which lays down rules for using its routine to bring down the program most simply and powerfully. It will therefore usually be best to adopt one of these schemes; a brief comparison may show which is the more suitable for your program.

a) The R.A.E. Scheme - is the more general. The sections it controls need not specify their successors; so any section can be used as a closed subroutine with entry and exit specified by a master routine compiled by the programmer. This means that a section can be used in several contexts, and so existing sections may be able to be incorporated unaltered into the program.

b) The English Electric Scheme - is simpler to use; each section specifies its successor. This means, however, that a section cannot be used unaltered in another context.

c) Restrictions - both schemes impose a few restrictions on the layout of instructions and data in the high-speed store, etc., and should therefore be studied before the program is written.

2.4. G.I.P.

The General Interpretive Program (G.I.P.5) is another control routine which incorporates several useful features at the expense of speed, and this sets it apart from the two routines mentioned above. It is used mainly for compiling programs from ready-made programs called "bricks" of which there are already a large number in the DEUCE library. This is fully described in DEUCE News 10.

2.5. Restarting.

An advantage of using an assembly scheme is that, should the program inexplicably go wrong, it will often be possible to go back and repeat more cautiously that part of the program which is troublesome. The procedure is

to restore control to the organising routine, and then to bring down from the drum and enter the appropriate section of the program. Restoring control is achieved by reading in the Scheme's own small Restore Control program (there is also a rather slow manual method of restoring control to G.I.P.)

3. PROGRAM DESIGN

3.1. Block Diagram

An overall diagram should be written in which the "units" are blocks which each contain sufficient action for one section of the final program. This action will usually employ most of the high-speed store. Experience will show how much ground a block can cover, and at first it is better to make blocks too small rather than too large. Block divisions should not cut across iterative loops unless the size of the loop makes this unavoidable.

3.2. Logical Flow Diagram

The action of each section must be broken down into small and simple steps, and a diagram made in which each step is represented by a phrase or a question. This is the logical flow diagram. The construction of the logical flow diagram is the "meat" of program writing, and if it is done with care and imagination its translation into computer language and the subsequent testing and operation of the program will be achieved without any great mental effort.

3.3. Use of Store.

a) Use of Drum within Section - it should not be necessary to fetch instructions from the drum within a section. However, data and results will often need to be fetched or stored, and some of the subroutines classified under B (in particular B06) can be recommended for fetching and storing continuously words in consecutive mcs of the drum (where track T, mc 31 is followed by track T+1, mc0).

b) Use of H.S.Store within Section - a general layout of the high-speed store within each section, taking into account volume of data to be processed and subroutines to be used, should be made.

Do not mix data with instructions; keep instructions in lower DLs and data in higher DLs.

DLs 9 and 10 are often useful as a consecutive store for working data, and DL11 is used by magnetic operations, so it is best to arrange that parameters, counters and markers, especially those which have to be preserved from section to section, are stored in DL12.

3.4. Instruction Flow Diagram.

The instruction flow diagram can be attempted when the block diagram, layout of store and logical diagram have all been written.

3.5. Coding and Punching

Coding and punching should be thoroughly checked before the program is put on the computer for its first test; the program ZP06/1 is often used for checking. Alternatively, coding sheets can be checked by calculating and writing on the sheets the mc of transfer and of next instruction from the Wait and Timing numbers, and hence reconstructing the flow diagram.

3.6. Program Testing

A manual on program testing is being prepared.

4. UNIFORM CONDITIONS

Before a new program is read into it, the computer will have scraps of data and instructions left by the last user's program, and will have drum heads and possibly external keys and switches in undefined positions. The program being read in may not be independent of the state of the machine, and while this may be intentional, e.g. a program may use data left on the drum by the last program, it is sometimes unintentional, so that the machine appears to behave inconsistently; this is sometimes wrongly attributed to a machine fault. Therefore, when the program is the first of a self-contained sequence, the computer should be in a standard state.

4.1. A Standard State

A. ID lamps cleared
All keys level, except stop key
Stop key at NORMAL
Punch run out, cleared and Ready.
Reader run out and any cards removed from stacker before pack is put in hopper.

B TS COUNT cleared
High-speed store cleared.

C TCB off and parity established.

D Drum cleared.
Drum heads at zero.

E TCA off

4.2. How to Achieve a Standard State

A. Everything under heading A should be checked first, although it is considered good manners for the person handing over the machine to do this.

B is achieved by using Initial Input key.

C is achieved by making first card an Initial Card.

D is achieved by using Clear Drum program - ZP13/1.

E is achieved by never assuming that TCA is off!

It is often useful to be able to put the computer into a state for reading in more program in phase with that already there. This is most conveniently achieved by using one track of the drum as a "Clock Track". A "Synchronise Clock Track" program can examine the clock track and thus put the computer into this state. The three-card program "Set or Synchronise Clock Track" (ZP34) will, if placed after the Clear Drum program, put on Track 15/15 a clock track which is characterised by a P31 in mc 16. If ZP34 is subsequently put at the beginning of another program this will be read in in phase with the one already there, provided, of course, that the clock track has not been overwritten. Incidentally, ZP34 clears TCA. Also the program Post Morton (ZP29) will synchronise with this clock track.

To sum up, a program pack normally starts with:

1) Initial Card;
2) Clear Drum - ZP13/1.
3) Set Clock Track - ZP34;

and is run in on Initial Input key after keys and switches on Control Panel and Hollerith equipment have been checked.

5. DATA FOR TESTING THE PROGRAM

It is wise to test the program first with data that is slight in volume and complexity so that results can be checked with the minimum of effort. Usually not merely final results but also partial results are recovered from the computer, the latter at frequent intervals throughout the program. These are compared with hand calculations. The object of this is to localise programming errors where they exist.

The output of the intermediate results can be programmed (punched card output is more valuable than OS lamps output) and this part of the program short-circuited when the results are correct. Alternatively, where the tester is prepared to sacrifice the facility of continuing without restarting the program, the Post Mortem program (ZP29) can be used to punch out relevant parts of the computer store. Usually only trivial verifications, such as the state of a counter, can economically be made from the monitor display.

Data for later tests should strain the program by its volume or complexity. When a program has been altered to cope with an extreme case, it is important to check that it still works for the normal case.